Peter P. Bothner
Wolf-Michael Kähler

SMALLTALK

Einführung in die objekt-orientierte Programmierung

Diese Schrift ist inhaltlich identisch mit:

SMALLTALK: Einführung in die objekt-orientierte Programmierung
Peter P. Bothner; Wolf-Michael Kähler.
Friedr. Vieweg & Sohn, Braunschweig 1999
ISBN 3-528-05700-9

Vorwort

Die zuerst erlernte Programmiersprache prägt die Denkweise! Im Hinblick auf diese anerkannte Einschätzung hat in letzter Zeit eine verstärkte Diskussion darüber stattgefunden, welche Art von Programmiersprachen Gegenstand der informatorischen Grundausbildung sein sollten.

Nicht mehr vorteilhaft erscheint die Ausbildung in den klassischen problem-orientierten Sprachen, die zu Zeiten der traditionellen Datenverarbeitung eine Vormachtstellung besaßen.

Um Software-Produkte herzustellen, bedient man sich heutzutage in zunehmendem Maße der *objekt-orientierten* Programmierung. Bei diesem Ansatz nehmen die Daten, die bei den klassischen Programmiersprachen als passive Größen behandelt werden, eine *aktive* Rolle ein, indem – zur Lösung einer Problemstellung – *Objekte* als Träger von Daten eingerichtet werden. Der Lösungsplan beschreibt, wie diese Objekte miteinander zu kommunizieren haben. Durch diesen Nachrichtenaustausch wird die Ausführung von Handlungen veranlaßt, die die Durchführung des Lösungsplans bewirken.

Die objekt-orientierte Sicht besitzt gegenüber dem klassischen Vorgehen besondere Vorteile, wenn zwischen dem Anwender und dem erstellten Software-Produkt kommuniziert werden muß. Während sich diese Kommunikation zu früheren Zeiten – aus technischen Gründen – auf die Ein- und Ausgabe von Zeichen gründete, stehen dem Anwender heutzutage – durch den Einsatz von Grafik-Bildschirmen – neue Formen der Kommunikation mittels der Menü-, der Maus- und der Fenster-Technik offen.

Da sich der fenster-gestützte Dialog nicht mehr auf die Beantwortung von Eingabeanforderungen beschränkt, sondern Ereignisse – wie z.B. die Betätigung von Schaltfeldern innerhalb eines Fensters – bestimmte Anforderungen zur Ausführung bringen können, müssen entsprechende Kommunikationsmechanismen der Programmierung zugänglich sein. Diese Art von dialog-orientierter Programmierung zählt nicht nur zu den Stärken von objekt-orientierten Programmiersprachen, sondern bestimmt auch vornehmlich deren Einsatzfeld.

Die vorliegende Einführung in die objekt-orientierte Programmierung orientiert sich an der Programmiersprache SMALLTALK, die in den siebziger Jahren von einer Forschungsgruppe am Xerox Palo Alto Research Center entwickelt und 1981 vorgestellt wurde. Obwohl SMALLTALK der “Klassiker” im Bereich der objekt-orientierten Programmierung ist, gewinnt diese Sprache erst heutzutage in zunehmenden Maße im Bereich der professionellen Programmierung an Bedeutung.

Die jetzt einsetzende größere Verbreitung von SMALLTALK sowie die Möglichkeit, durch den Einsatz dieser Sprache auch die klassischen Programmieransätze – wie z.B. die prozedurale Programmierung – erläutern zu können, trägt entscheidend dazu bei, daß SMALLTALK – bei der informatorischen Grundausbildung – in zunehmenden Maße als *erste* Sprache gelehrt wird.

Um diese Entwicklung zu unterstützen, werden in diesem Buch die Sprachelemente von SMALLTALK nicht summarisch beschrieben, sondern durch Anwendungsbeispiele motiviert, die betont einfach gehalten sind und aufeinander aufbauen. Bei diesem Ansatz wird der Versuch unternommen, sowohl den am methodischen Vorgehen als auch den an einer programmier-technischen Umsetzung interessierten Lesern gerecht zu werden.

Dabei werden die Begriffe, die bei der objekt-orientierten Programmierung von grundsätzlicher Bedeutung sind, nicht isoliert, sondern Schritt für Schritt am durchgängig zugrundegelegten Anwendungsbeispiel vorgestellt.

Da es sich bei diesem Buch um eine Einführungsschrift handelt, werden keine spezifischen Vorkenntnisse erwartet – allerdings sollte dem Leser der elementare Umgang mit einem Personalcomputer geläufig sein.

Als objekt-orientierte Programmierumgebung, unter der der Einsatz von SMALLTALK vorgestellt wird, haben die Autoren das Software-Produkt "Smalltalk Express" der Firma "ObjectShare" verwendet. Da es als "Public Domain Software" zur Verfügung gestellt wird, entstehen für den interessierten Leser keine Kosten, wenn er es für den nicht-kommerziellen Einsatz beziehen will. Es ist im INTERNET unter der URL-Adresse "http://www.objectshare.com/se/seinfo.htm" abrufbar. Der Einsatz dieses Produktes, das unter Windows 3.1 und höheren Versionen von Windows sowie unter Windows NT lauffähig ist, wird in der vorliegenden Beschreibung unter Windows 95 vorgestellt.

Für das Korrekturlesen und Hinweise zur Verbesserung der Darstellung danken wir Herrn Dipl.-Inf. O. Bergst, Herrn Dipl.-Wirtschaftsinf. J. Hafner und Herrn Dipl.-Ökonom A. Schröter.

In besonderem Maße sind wir Herrn Dr. Klockenbusch vom Vieweg Verlag für seine Geduld und die traditionell gute Zusammenarbeit zu Dank verpflichtet.

Bremen/ Ritterhude
im Mai 1998

Peter P. Bothner und Wolf-Michael Kähler

Inhaltsverzeichnis

Kapitel 1

Problemstellung und Planung der Lösung

1.1 Problemstellung und Problemanalyse

Um eine *Problemstellung* durch den Einsatz der Datenverarbeitung zu lösen, ist zunächst eine *Problemanalyse* durchzuführen. Hierbei ist eine komplexe Problemstellung in möglichst überschaubare Teilprobleme zu gliedern und eine Strategie für einen *Lösungsplan* zu entwickeln.

- Da wir die Lösung einer Problemstellung unter Einsatz der objekt-orientierten Programmiersprache SMALLTALK beschreiben wollen, setzen wir uns im folgenden zum Ziel, sowohl die Leitlinien des objekt-orientierten Programmierens als auch die Grundlagen der Programmiersprache SMALLTALK kennenzulernen.

Im Hinblick auf diese Zielsetzung orientieren wir uns an einer einfachen Problemstellung und betrachten den folgenden Sachverhalt:

- Schüler unterschiedlicher Jahrgangsstufen führen Sportwettkämpfe in mehreren Disziplinen durch, bei denen die erreichten Leistungen durch ganzzahlige Punktwerte gekennzeichnet werden.

Im Rahmen des Wettkampfvergleichs möchten wir uns über die durchschnittliche Leistung der einzelnen Jahrgangsstufen informieren, so daß wir die folgende Problemstellung formulieren:

- PROB-1:
  Die erreichten Punktwerte sind dialog-orientiert zu erfassen und einer Auswertung zu unterziehen, so daß aus den individuellen Punktwerten der jeweilige jahrgangsstufen-spezifische Leistungsdurchschnitt ermittelt wird!

Als Rahmenbedingungen, unter denen eine Lösung angegeben werden soll, legen wir fest:

- Unter Leistungsdurchschnitt soll der Durchschnittswert der Punktwerte verstanden werden, so daß die Summe aller Punktwerte zu bilden und durch die Anzahl der Summanden zu teilen ist.

  Hinweis: Damit diese Rechenoperationen sinnvoll sind, wird angenommen, daß mindestens ein Punktwert erhoben wird.

1.2 Ansatz für einen Lösungsplan

Zur Lösung von PROB-1 entwickeln wir in einem ersten Schritt eine verbal gehaltene Beschreibung. Dabei stellen wir die grundlegenden Begriffe vor, die zur Formulierung eines objekt-orientierten Lösungsplans benötigt werden.

Da der Dialog mittels grafischer Benutzeroberflächen sich als besonders geeignetes Einsatzfeld für die objekt-orientierte Programmierung erweist, werden wir den *Erfassungsprozeß* auf die Fenster-Umgebung des Windows-Systems gründen.
Für diesen Erfassungsprozeß stellen wir ein *Erfassungsformular* zur Verfügung, das am Bildschirm innerhalb eines eigenständigen Fensters angezeigt wird.
Das Erfassungsformular soll aus einem *Eingabefeld*, einem *Textfeld* (mit dem Inhalt "Wert:") und zwei *Schaltfeldern* (mit den Aufschriften "erfasse" und "Erfassungsende") aufgebaut sein.
Das Fenster, in dem dieses Erfassungsformular angezeigt werden soll, nennen wir fortan *Erfassungsfenster*.

Sofern wir z.B. die Punktwerte für die Schüler der Jahrgangsstufe 11 erfassen wollen, soll das Erfassungsfenster durch die Titel-Zeile "Jahrgangsstufe 11" gekennzeichnet sein und das folgende *Erscheinungsbild* besitzen:

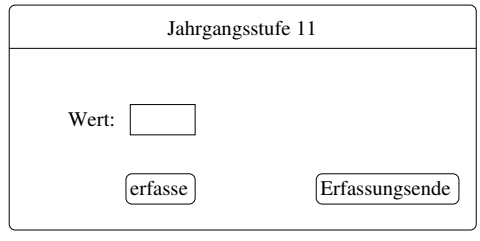


Abbildung 1.1: Erfassungsfenster mit Erfassungsformular

Um diese Bildschirmanzeige für den durchzuführenden *Erfassungsprozeß* vorzubereiten, ist die folgende Handlung auszuführen:

- "Initialisieren der Erfassung":
  Es ist das Erfassungsformular und der Aufbau des zugehörigen Erfassungsfensters festzulegen.

In der Abbildung enthält das Erfassungsfenster die Titel-Zeile "Jahrgangsstufe 11". Dies verdeutlicht, daß in dem Eingabefeld, dem der Text "Wert:" vorangestellt ist, die Punktwerte der Jahrgangsstufe 11 erfaßt werden sollen.

Damit erkennbar ist, für welche Jahrgangsstufe der Erfassungsprozeß ablaufen soll, ist die folgende Handlung durchzuführen:

- "Festlegen der Überschrift des Erfassungsfensters":
  Zur Kennzeichnung der Jahrgangsstufe ist eine geeignete Titel-Zeile des Erfassungsfensters festzulegen.

Um einen Erfassungsprozeß zu beginnen, ist die folgende Handlung vorzunehmen:

- "Durchführen der Erfassung":
  Es ist das Erfassungsfenster mit dem Erfassungsformular anzuzeigen.

Bei der Durchführung der Erfassung müssen die zu erfassenden Werte schrittweise mittels der Tastatur in das Eingabefeld eingetragen werden.

Hinweis: Dabei ist zu gewährleisten, daß vor der Eingabe eines neuen Wertes der Cursor automatisch auf das Eingabefeld positioniert und das Eingabefeld mit Leerzeichen belegt ist.

Damit am Erfassungsende der Durchschnittswert berechnet werden kann, sind die im Eingabefeld angezeigten Werte in einen geeigneten *Sammel-Behälter* zu übernehmen.
Um während des Erfassungsprozesses mitteilen zu können, daß ein über die Tastatur eingegebener Wert in diesen Sammel-Behälter zu übertragen und daß die Erfassung zu beenden ist, sind im Erfassungsfenster die beiden Schaltfelder mit den Aufschriften "erfasse" und "Erfassungsende" vorgesehen. Dabei soll ein Mausklick auf das Schaltfeld "erfasse" die Übernahme eines Wertes vom Eingabefeld in den Sammel-Behälter und ein Mausklick auf das Schaltfeld "Erfassungsende" die Beendigung der Erfassung signalisieren, so daß das Erfassungsfenster vom Bildschirm entfernt wird.

Fassen wir zusammen! Für den Lösungsplan benötigen wir somit ein Erfassungsfenster, das ein Eingabefeld zur Aufnahme von Punktwerten enthält, und einen Sammel-Behälter, in dem die erfaßten Punktwerte – zur weiteren Verarbeitung – gesammelt werden können. Zusätzlich sind zwei Schaltfelder zu verwenden, durch deren Einsatz sich die Erfassung steuern läßt.

Diese Größen, die die Grundlagen für unseren Lösungsplan bilden, müssen geeignet miteinander in Beziehung treten, indem z.B. ein Wert aus dem Eingabefeld an den Sammel-Behälter weitergeleitet wird, oder aber – durch die Betätigung des Schaltfeldes "Erfassungsende" – dem Sammel-Behälter signalisiert wird, daß sich die angestrebte Ermittlung eines Durchschnittswertes jetzt vornehmen läßt.

Im Hinblick auf derartige Mitteilungen sind für den Lösungsplan unter anderem daher die beiden folgenden Handlungen, die mit den Schaltfeldern "erfasse" und "Erfassungsende" zu verknüpfen sind, zur Ausführung vorzusehen:

- "Erfassen eines Wertes":
  Übertragung eines Wertes vom Eingabefeld in den Sammel-Behälter, Löschung dieses Wertes im Eingabefeld durch die Überschreibung mit Leerzeichen und Positionierung des Cursors auf den Anfang des Eingabefeldes.

- "Entfernen des Erfassungsfensters":
  Entfernung des Erfassungsfensters vom Bildschirm.

Um nach dem Erfassungsende für die im Sammel-Behälter erfaßten Werte die gewünschte Durchschnittsbildung vorzunehmen, ist die folgende Handlung durchzuführen:

- "Berechnen des Durchschnittswertes":
  Es sind alle im Sammel-Behälter eingetragenen Werte zu summieren. Anschließend ist die resultierende Summe durch die Anzahl der Summanden zu teilen und der Ergebniswert am Bildschirm anzuzeigen.

Zusammenfassend läßt sich der bisherige Ansatz für einen Lösungsplan wie folgt skizzieren:

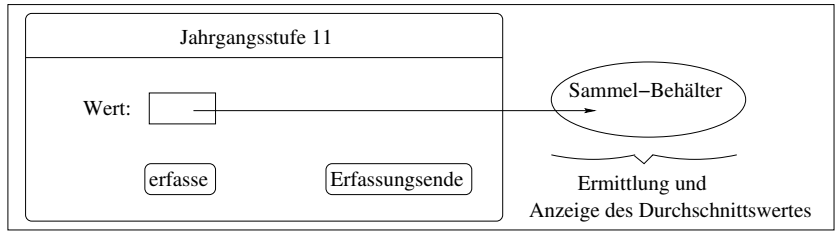


Abbildung 1.2: Ansatz für einen Lösungsplan

Es ist erkennbar, daß sich die Problemstellung PROB-1 in die beiden folgenden Teilprobleme gliedern läßt:

- 1. Teilproblem PROB-1-1:
  Die Punktwerte sind zu erfassen und in einen Sammel-Behälter zu übertragen!

- 2. Teilproblem PROB-1-2:
  Der Durchschnittswert aller Punktwerte, die in dem Sammel-Behälter aufbewahrt werden, ist zu berechnen und anzuzeigen!

Der Lösungsplan von PROB-1-2, der die Weiterverarbeitung der zuvor erfaßten Daten beschreibt, basiert auf der Lösung von PROB-1-1. Es ist erkennbar, daß die Lösung von PROB-1-1 als Grundlage für die Lösung jeder Problemstellung dienen kann, bei der statistische Kennziffern – wie z.B. die Berechnung des Durchschnittswertes – aus Daten zu ermitteln sind. So könnten wir z.B. – im Rahmen einer anderen Problemstellung – an der Bestimmung eines Wertes interessiert sein, der am häufigsten aufgetreten ist bzw. der im Sinne einer Reihenfolge den mittleren Wert aller Daten darstellt.

Im folgenden wenden wir uns zunächst der Lösung von PROB-1-1 zu. Auf der Basis des oben angegebenen Lösungsplans läßt sich der Erfassungsprozeß wie folgt darstellen:

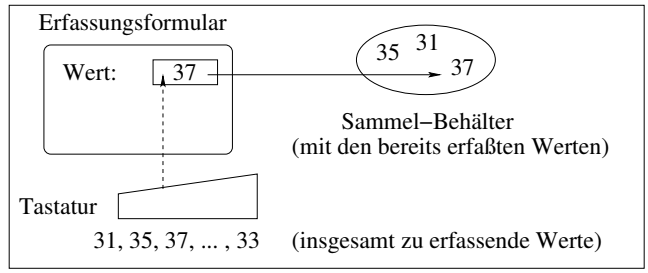


Abbildung 1.3: Erfassungsprozeß

Zur Lösung von PROB-1-1 muß es – neben dem Sammel-Behälter – weitere Behälter in Form von *Verwaltungs-Behältern* geben, durch deren Inhalte die Struktur des Erfassungsformulars und das Erscheinungsbild des Erfassungsfensters bestimmt sind, indem z.B. Angaben über die Position des Fensters auf dem Bildschirm, die Größe des Eingabefeldes sowie die Position des Cursors festgelegt werden. Da sich die insgesamt erforderlichen Verwaltungs-Behälter – wie wir später sehen werden – automatisch einrichten lassen, verzichten wir an dieser Stelle auf eine nähere Beschreibung.

1.3 Konkretisierung des Lösungsplans

1.3.1 Der Begriff des "Objekts"

Wir haben bei der Entwicklung des Lösungsplans festgestellt, daß ein Erfassungsfenster mit einem Eingabefeld und zwei Schaltfeldern sowie ein Sammel-Behälter benötigt werden.

Durch den Einsatz dieser Größen läßt sich der Erfassungsprozeß dadurch modellieren, daß der Sammel-Behälter und alle zusätzlich benötigten Verwaltungs-Behälter zusammengefaßt und als eine *Einheit* angesehen werden. Der jeweilige Inhalt der Behälter kennzeichnet die konkreten Eigenschaften, die der Erfassungsprozeß zu einem Zeitpunkt besitzt.

Dabei verkörpert der Sammel-Behälter die Eigenschaft, Kenntnis von den bislang erfaßten Punktwerten zu besitzen.

- Derartige Eigenschaften, mit denen sich die Zustände der durch eine Modellbildung beschriebenen Einheiten kennzeichnen lassen, werden als *Attribute* bezeichnet.

Der jeweils konkrete Zustand des Erfassungsprozesses spiegelt sich in den jeweiligen *Attributwerten* – wie z.B. der Gesamtheit der bislang gesammelten Punktwerte – wider.

Diese Art der Modellierung, bei der Träger von Attributen betrachtet werden, deren jeweils aktueller Zustand durch die aktuellen Attributwerte gekennzeichnet wird, ist grundlegend für die objekt-orientierte Programmierung.

- Die Träger von Attributen – wie z.B. das Erfassungsfenster, das Eingabefeld, die Schaltfelder sowie der Sammel-Behälter und die Verwaltungs-Behälter –, die im Rahmen eines Lösungsplans Gegenstand der Betrachtung sind, werden *Objekte* genannt.
  Der jeweilige Zustand eines Objektes wird durch die Gesamtheit seiner Attributwerte verkörpert.

- Die Attribute und die zugehörigen Attributwerte werden unter einer objektspezifischen "Schale" gemeinsam gekapselt, so daß sie "nach außen hin" verborgen bleiben. Dies wird als "Geheimnisprinzip" oder auch als "Prinzip der Datenkapselung" bezeichnet.
  Dies bedeutet, daß nur die Objekte selbst ihre jeweils aktuellen Attributwerte preisgeben können, wozu sie durch spezifische Anforderungen gezielt aufgefordert werden müssen.

  Das Geheimnisprinzip hat zur Konsequenz, daß die Konzeption von zusätzlich einzubeziehenden Objekten keinen unmittelbaren Einfluß auf die Struktur der bereits vorhandenen Objekte hat.

Unter einer gemeinsamen Schale sollten z.B. die Informationen über die Breite und die Höhe von Rechtecken gekapselt werden, sofern Rechtecke als Objekte Gegenstand von Lösungsplänen sein sollen. Gleichfalls ist es sinnvoll, die Eigenschaften "Radius" und "Mittelpunkt" eines Kreises innerhalb einer Objekt-Schale zu verbergen, falls Kreise zum Gegenstand eines objekt-orientierten Lösungsplans gemacht werden sollen.

Hinweis: Geometrische Gebilde wie Kreise und Rechtecke könnten z.B. dann in einem Lösungsplan eine Rolle spielen, wenn geometrische Muster auf dem Bildschirm erzeugt werden sollen, um Prototypen von grafischen Strukturen darzustellen.

Damit Objekte innerhalb eines Lösungsplans angegeben und deren Zustände verändert werden können, müssen sie durch geeignet gewählte *Namen* ansprechbar sein.

- Dazu sind Objekte an *Bezeichner* zu binden. Eine derartige Bindung an einen Bezeichner wird *Variable* genannt.
  Ist ein Objekt durch eine Variable an einen Bezeichner gebunden, so handelt es sich bei diesem Bezeichner um den Namen der Variablen, so daß anstelle eines Bezeichners auch von einem *Variablennamen* gesprochen wird.

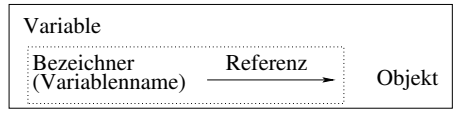


Abbildung 1.4: Referenzierung auf ein Objekt

Hinweis: Obwohl der Begriff der "Variablen" nicht nur den Namen, sondern gleichzeitig die Bindung an das zugehörige Objekt beinhaltet, wird der Begriff "Variable" im folgenden – aus Gründen der Vereinfachung – auch als Synonym für "Variablenname" verwendet.

- Zu einem bestimmten Zeitpunkt wird durch eine Variable jeweils genau ein Objekt referenziert. Es ist zulässig, daß verschiedene Objekte nacheinander an denselben Variablennamen gebunden werden können, so daß sich verschiedene Objekte zu unterschiedlichen Zeitpunkten durch ein und denselben Variablennamen referenzieren lassen. Dabei ist es zulässig, daß diese Objekte, die nacheinander an denselben Variablennamen gebunden werden, Träger unterschiedlicher Attribute sein können.

  Hinweis: Man spricht davon, daß Variablen nicht statisch, sondern dynamisch typisiert sind.

Sollen z.B. die Punktwerte der Jahrgangsstufe 11 erfaßt werden, so können wir z.B. den Bezeichner "WerteErfassung11" wählen, um den diesbezüglichen Erfassungsprozeß zu kennzeichnen.

Sofern wir beabsichtigen, die Erfassung für die Jahrgangsstufe 11 und 12 *parallel* auszuführen, wäre z.B. die folgende Bezeichnerwahl sinnvoll:

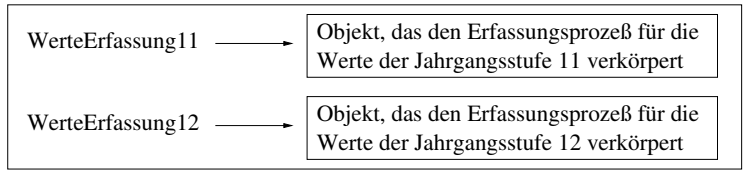


Abbildung 1.5: Bezeichner für Objekte

Zur Durchführung eines Lösungsplans werden die Objekte *dynamischen Zustandsänderungen* unterworfen, die – z.B. durch die Änderung der Titel-Zeile im Erfassungsfenster – eine Änderung ihres äußeren Erscheinungsbildes bewirken oder – wie z.B. bei der Ergänzung der bislang erfaßten Werte durch einen weiteren erfaßten Wert – keine Konsequenz auf ihr äußeres Erscheinungsbild haben.

Um derartige Zustandsänderungen zu bewirken, ist der folgende Sachverhalt von grundlegender Bedeutung:

- Eine Zustandsänderung muß das betreffende Objekt *selbst* vornehmen, indem es eine ihm bekannte *Handlung* ausführt.

Daher ist es *unmöglich*, die jeweils gewünschte Zustandsänderung eines Objektes von "außen" zu bewirken. Von "außen" kann hierzu nur der Anstoß gegeben werden, indem eine geeignete Anforderung formuliert wird, um das betreffende Objekt zur Ausführung der erforderlichen Handlung zu veranlassen.

1.3.2 Der Begriff der "Klasse"

Als Ergebnis der bisherigen Erörterungen müssen zur Lösung von PROB-1-1 geeignete Objekte – z.B. mit den Bezeichnern "WerteErfassung11" oder "WerteErfassung12" – eingerichtet und diejenigen Handlungen festgelegt werden, die die gewünschten Zustandsänderungen bewirken können.

Dieses Vorgehen ist grundsätzlich beim objekt-orientierten Programmieren:

- Die Entwicklung eines Lösungsplans basiert auf geeignet einzurichtenden Objekten und der Festlegung derjenigen Handlungen, die von ihnen auszuführen sind, damit ihre Zustände im Sinne des Lösungsplans verändert werden können.

Da es im Hinblick auf die Lösung von PROB-1-1 unerheblich ist, ob Werte für die Jahrgangsstufe 11 oder für die Jahrgangsstufe 12 zu erfassen sind, lassen sich die einzelnen Erfassungsprozesse durch Objekte modellieren, die demselben *Bauplan* unterliegen.

Dieser Bauplan ist das Muster (Schablone, Schema, Prototyp-Beschreibung), nach dem Objekte als individuelle Exemplare zu gestalten sind, um innerhalb des Lösungsplans die ihnen zugedachte Aufgabe erfüllen zu können. Der Bauplan für die einzelnen *Objekt-Exemplare* muß folgendes enthalten:

- Angaben über die Attribute von Objekt-Exemplaren, durch die der Zustand der einzelnen nach diesem Bauplan eingerichteten Objekte gekennzeichnet wird.
- Beschreibungen von Handlungen, die durch Objekt-Exemplare ausgeführt werden können, um Änderungen ihrer Zustände zu bewirken.

Für die Objekte, mit denen die Erfassungsprozesse modelliert werden sollen, ist demzufolge ein Bauplan der folgenden Form festzulegen:

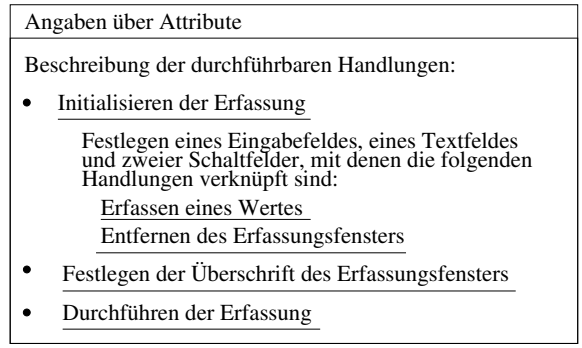


Abbildung 1.6: Bauplan für den Erfassungsprozeß

Hinweis: In diesem Bauplan sind keine Angaben über Verwaltungs-Behälter enthalten, da sie sich – wie bereits erwähnt – automatisch bereitstellen lassen.

Um die Gesamtheit aller Objekt-Exemplare, die sich nach einem einheitlichen Bauplan als individuelle Größen erzeugen lassen, *typisieren* zu können, wird die folgende Verabredung getroffen:

- Die Zusammenfassung aller Angaben, die als Muster zur Einrichtung einzelner Objekt-Exemplare dienen, wird als *Klasse* bezeichnet.

Somit handelt es sich bei einer Klasse um eine Vorlage für den Aufbau aller individuell einrichtbaren Objekt-Exemplare.

Durch die Vereinbarung einer Klasse wird daher festgelegt, über welche Attribute ihre Objekt-Exemplare verfügen und welche Handlungen ihre Objekt-Exemplare ausführen können.

- Um eine Klasse zu kennzeichnen, wird ein eindeutiger *Klassenname* vergeben. In SMALLTALK muß dieser Name durch einen *Großbuchstaben* eingeleitet werden, dem Klein- und Großbuchstaben sowie Ziffern folgen können.

Da der jeweilige Klassenname zum Ausdruck bringen soll, welche Bedeutung die zugehörigen Objekt-Exemplare im Rahmen des Lösungsplans spielen, ist es sinnvoll, der von uns konzipierten Klasse den Klassennamen "WerteErfassung" zu geben.

Hinweis: Diese Namenswahl steht im Einklang mit den oben gewählten Bezeichnern "WerteErfassung11" und "WerteErfassung12", durch die die Objekt-Exemplare für die beiden Erfassungsprozesse zur Durchführung einer parallelen Erfassung gekennzeichnet wurden.
Sofern Namen aus Wörtern bzw. Wortfragmenten zusammengesetzt werden, ist es nützlich, den jeweiligen Wortanfang mit einem Großbuchstaben einzuleiten. Deshalb haben wir z.B. "WerteErfassung11" anstelle von "Werteerfassung11" verwendet.

1.3.3 Der Begriff der "Instanz"

Damit ein Erfassungsprozeß zur Ausführung gelangt, muß ein *Objekt-Exemplar* der Klasse "WerteErfassung" erzeugt werden. Der Vorgang, bei dem ein Objekt-Exemplar – nach dem durch die Klasse verkörperten Bauplan – eingerichtet wird, heißt *Instanziierung*. Ein durch eine Instanziierung erzeugtes Objekt-Exemplar wird *Instanz* genannt. Für die Instanzen einer Klasse sind die folgenden Sachverhalte grundlegend:

- Da jede Instanziierung zu einem neuen, eigenständigen Objekt-Exemplar führt, unterscheidet sich jede Instanz einer Klasse von jeder weiteren Instanz derselben Klasse.
- Verschiedene Instanzen derselben Klasse können sich in gleichen oder in unterschiedlichen Zuständen befinden. Die Zustandsänderung einer Instanz wird dadurch bewirkt, daß eine Instanz ihre Attributwerte durch die Ausführung einer geeigneten Handlung ändert.

Eine Instanz läßt sich an einen Bezeichner binden, der bei der Instanziierung festgelegt wird.

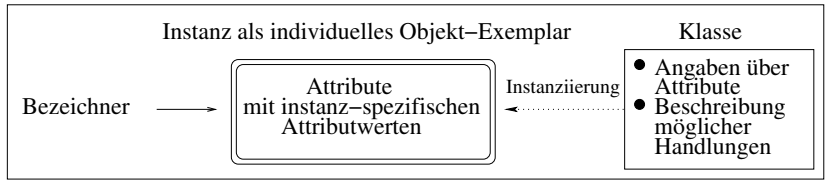


Abbildung 1.7: Instanziierung

Hinweis: Fortan werden wir Klassen durch ein einfaches Rechteck und Instanzen durch "doppelte" Rechtecke mit abgerundeten Ecken kennzeichnen.

In unserer Situation soll eine Instanz der Klasse "WerteErfassung" die erfaßten Punktwerte in einem Sammel-Behälter aufnehmen. Daher stellt dieser Behälter – als Träger der jeweils konkret erfaßten Punktwerte und damit *Bestandteil* der Instanz – ebenfalls ein Objekt dar, so daß er über einen Bezeichner referenziert werden kann.

Sofern der Name "werteBag" als Bezeichner des Sammel-Behälters gewählt wird, ergibt sich der folgende Sachverhalt:

Abbildung 1.8: Referenzierung auf den Sammel-Behälter

Hinweis: Dabei soll der Namensbestandteil "Bag" (englische Bezeichnung für "Beutel") einen Hinweis auf die Form des Sammel-Behälters geben, in dem sämtliche erfaßten Punktwerte – ohne festgelegte Reihenfolge – aufbewahrt werden sollen.
Insofern wäre es auch sinnvoll gewesen, den Namen "werteKorb" zur Referenzierung des Sammel-Behälters festzulegen. Dagegen spricht allein, daß der Name der Klasse, aus der die Instanziierung erfolgt, in die Bezeichnung des Objekt-Exemplars aufgenommen werden sollte. Da eine Instanziierung aus einer Klasse namens "Bag" erfolgen wird, ist die Verwendung von "werteBag" angemessener.

Um die Variablen, über die die Instanzen *selbst* referenziert werden, von denjenigen Variablen abzugrenzen, die *innerhalb* der Instanzen für die instanz-spezifischen Attributwerte verwendet werden, wird zwischen dem Begriff der *globalen Variablen* und der *Instanz-Variablen* unterschieden.

- Die Variablen, die die jeweils instanziierten Objekt-Exemplare bezeichnen, heißen *globale Variablen*.

  Hinweis: Später werden wir kennenlernen, daß sich auch Instanzen einrichten lassen, die nicht an globale Variable gebunden sind.

- Die Variablen, durch die die Attributwerte einer Instanz referenziert werden, werden *Instanz-Variablen* (Exemplar-Variablen) genannt.

Als Ergebnis der vorausgehenden Darstellung können wir z.B. den Erfassungsprozeß, durch den die Werte der Jahrgangsstufe 11 erfaßt werden sollen, durch eine

Instanziierung der folgenden Form kennzeichnen:

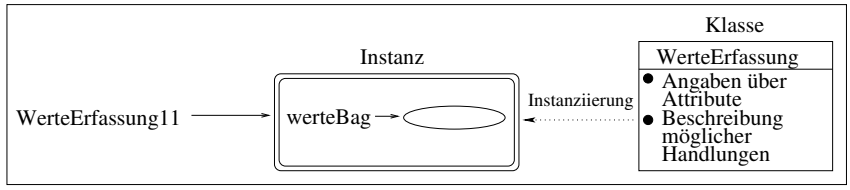


Abbildung 1.9: Instanziierung eines Erfassungsprozesses

Bei der abgebildeten Instanziierung wird eine globale Variable namens “WerteErfassung11” eingerichtet, die auf eine Instanz der Klasse “WerteErfassung” weist, deren Zustand durch die Instanz-Variable “werteBag” gekennzeichnet ist.

Führen wir eine weitere Instanziierung der Klasse “WerteErfassung” durch, indem wir eine Instanz mittels der globalen Variablen “WerteErfassung12” einrichten, so besitzen sowohl “WerteErfassung11” als auch “WerteErfassung12” jeweils eine Instanz-Variable namens “werteBag”.

Da Instanz-Variablen jeweils Bestandteile individueller Instanzen sind, werden sie durch die jeweiligen Objekt-Schalen nach “außen” hin abgeschirmt. Daher ist es unproblematisch, daß Instanz-Variablen, die gleiche Attribute verkörpern, in unterschiedlichen Instanzen gleichnamig sind.

Somit läßt sich grundsätzlich feststellen:

- Alle Instanzen einer Klasse verfügen über Instanz-Variablen gleichen Namens. Jede einzelne Instanz hat Kenntnis von den eigenen Instanz-Variablen, die bei ihrer Instanziierung – gemäß dem Bauplan ihrer Klasse – eingerichtet wurden.

Anders ist der Sachverhalt bei den globalen Variablen. Da die Namen dieser Variablen jeweils individuelle Instanzen kennzeichnen, können keine zwei globale Variablen gleichen Namens existieren.

In SMALLTALK werden Instanz-Variablen und globale Variablen syntaktisch durch die Art ihres ersten Zeichens unterschieden:

- Namen von Instanz-Variablen sind grundsätzlich mit einem *Kleinbuchstaben* einzuleiten. Ihm dürfen Klein- und Großbuchstaben sowie Ziffern folgen.
- Namen, durch die globale Variablen gekennzeichnet werden, müssen mit einem *Großbuchstaben* beginnen. Sie unterliegen ansonsten dem Bildungsgesetz für Namen von Instanz-Variablen.

1.3.4 Der Begriff der “Methode”

Nachdem wir kennengelernt haben, daß der Bauplan für die Erfassungsprozesse durch eine *Klassen-Vereinbarung* festzulegen ist, wenden wir uns jetzt den Handlungen zu, die von den Erfassungsprozessen ausführbar sein müssen.

- Damit eine Handlung von einer Instanz ausgeführt werden kann, muß sie als *Methode* (Algorithmus) innerhalb einer Klassen-Vereinbarung festgelegt sein.

  Durch die Ausführung einer Methode – und nur dadurch – ist es möglich, die Attributwerte einer Instanz ändern zu lassen. Um eine derartige Änderung "von außen" herbeizuführen, muß die Instanz veranlaßt werden, die jeweils erforderliche Methode selbst auszuführen.

- Damit eine Methode für eine Instanz ausführbar ist, muß die Instanz diese Methode *kennen*. Eine Möglichkeit, eine Methode für eine Instanz bekannt zu machen, besteht darin, die Methode innerhalb derjenigen Klasse zu vereinbaren, aus der die Instanz instanziiert wird.

Zur Identifizierung der Methoden, die für eine Instanz ausführbar sein sollen, werden Namen verwendet.

- Der Name, der eine Methode kennzeichnet, wird als *Methoden-Selektor* bezeichnet. Jeder Methoden-Selektor besteht aus einem oder mehreren Wörtern, die *Selektoren* genannt werden. Diese Selektoren haben die Funktion von Schlüsselwörtern, mit denen ergänzende Angaben zur Durchführung der jeweiligen Methoden gemacht werden können.
  Die einzelnen Selektoren sind nach denselben Regeln zu bilden, die zuvor für die Namensgebung von Instanz-Variablen festgelegt wurden.

  Hinweis: Sofern ein Methoden-Selektor zur Kennzeichnung einer Methode zu vereinbaren ist, sollten aussagekräftige Schlüsselwörter verwendet werden.

Zur Benennung der von uns im Abschnitt 1.2 konzipierten Handlungen, die für Instanzen der Klasse "WerteErfassung" ausführbar sein sollen, legen wir die folgenden Methoden-Selektoren fest:

- "initialisierenErfassung":
  Zum Aufbau des Erfassungsfensters und des Erfassungsformulars.
- "erfassenWert":
  Zur Übertragung eines Wertes aus dem Eingabefeld des Erfassungsfensters nach "werteBag".
- "entfernenErfassungsfenster":
  Zur Entfernung des Erfassungsfensters vom Bildschirm.
- "festlegenUeberschrift":
  Zur Kennzeichnung des Erfassungsfensters durch eine geeignete Titel-Zeile.
- "durchfuehrenErfassung":
  Zur Anzeige des Erfassungsfensters.

Damit diese Methoden einer Instanz der Klasse "WerteErfassung" bekannt sind, werden wir sie innerhalb dieser Klasse vereinbaren.
Aus didaktischen Gründen beschränken wir uns zunächst darauf, die insgesamt erforderlichen Angaben stichwortartig zusammenzufassen:

Name der Klasse: WerteErfassung
Instanz-Variablen: werteBag
Methoden:

- "initialisierenErfassung"
  - Initialisierung von "werteBag" als *Bag*
  - Vorbesetzung der Titel-Zeile des Erfassungsfensters mit Leerzeichen
  - Festlegung des Eingabefeldes
  - Festlegung, daß das Textfeld mit dem Text "Wert:" vor dem Eingabefeld angezeigt werden soll
  - Festlegung des Schaltfeldes "erfasse", dessen Betätigung die Methode *"erfassenWert"* auslösen soll
  - Festlegung des Schaltfeldes "Erfassungsende", dessen Betätigung die Methode *"entfernenErfassungsfenster"* auslösen soll
- *"erfassenWert"*
  - Übertragung eines Wertes vom Eingabefeld nach "werteBag"
  - Eintragung von Leerzeichen in das Eingabefeld
  - Plazierung des Cursors auf das Eingabefeld
- *"entfernenErfassungsfenster"*
  - Entfernung des Erfassungsfensters vom Bildschirm
- "festlegenUeberschrift"
  - Festlegung der Überschrift des Erfassungsfensters
- "durchfuehrenErfassung"
  - Anzeige des Erfassungsfensters am Bildschirm
  - Festlegung, daß der Cursor auf das Eingabefeld plaziert wird

Hinweis: Um hervorzuheben, daß die beiden Methoden "erfassenWert" und "entfernen-Erfassungsfenster" – mittels der Schaltfelder "erfasse" und "Erfassungsende" – innerhalb der Methode "initialisierenErfassung" zur Ausführung gelangen sollen, haben wir ihre Namen *kursiv* geschrieben.

Die Klasse "WerteErfassung" kennzeichnen wir abkürzend durch die folgende Grafik:

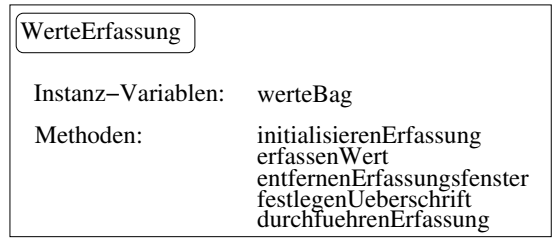


Abbildung 1.10: Die Klasse "WerteErfassung"

Diese Klassen-Beschreibung basiert auf der folgenden Darstellung, mit der sich ein Überblick über die für eine Problemlösung erforderliche Objekt-Struktur in Form

der benötigten Attribute und Methoden angeben läßt:

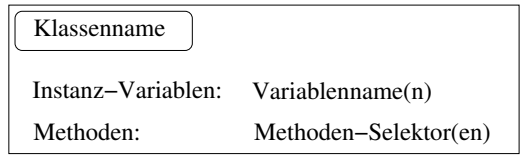


Abbildung 1.11: Grafische Beschreibung einer Klasse

Im Hinblick darauf, daß Methoden innerhalb einer Klasse vereinbart und damit von den Instanzen dieser Klasse ausgeführt werden können, sind die folgenden Aussagen von grundsätzlicher Bedeutung:

- Alle Methoden lagern in der Klasse, in der sie vereinbart werden.

  Hinweis: Die Gesamtheit aller Methoden, die innerhalb einer Klasse vereinbart sind, wird "Protokoll" genannt.

- Aus Gründen der redundanzfreien Speicherung werden die Methoden bei der Instanziierung *nicht* in die Instanzen übernommen, so daß sie *nicht* als Bestandteil einer Instanz angesehen werden. Da jede Instanz "weiß", aus welcher Klasse sie instanziiert wurde, kann die Instanz jederzeit Einblick in die Gesamtheit der Methoden nehmen, die von den Instanzen dieser Klasse ausgeführt werden können.

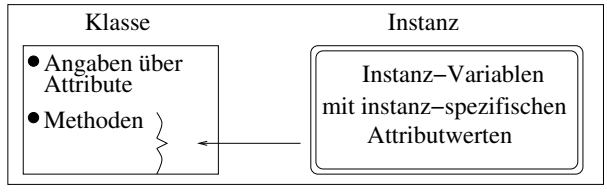


Abbildung 1.12: Instanz und Methoden

1.3.5 Der Begriff der "Message"

Damit eine Instanz zur Ausführung einer Methode veranlaßt wird, muß ihr eine Nachricht (Botschaft) zugestellt werden.

- Diese Nachricht ist in Form einer *Message* (Methodenaufruf) an die Instanz zu richten.
  Diejenige Instanz, die Empfänger einer Message ist, wird als *Empfänger-Objekt* bezeichnet.

Damit eine Methode von einer Instanz ausgeführt wird, muß ihr eine Message in Form einer *Anforderung* zugestellt werden.

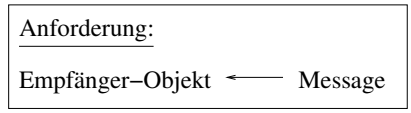


Abbildung 1.13: Benachrichtigung einer Instanz durch eine Message

Hierdurch wird eine charakteristische Eigenschaft des objekt-orientierten Programmierens beschrieben:

- Das Grundprinzip besteht darin, daß Objekte miteinander *kommunizieren*, um die Leistungen zu erbringen, die zur Lösung einer Problemstellung erforderlich sind.
  Der Lösungsplan muß daher beschreiben, welche Objekte einzurichten sind und welchen Objekten welche Messages zugestellt werden müssen.
- Dabei kann jedes Objekt, das Empfänger-Objekt einer Message ist, eine weitere Message an sich *selbst* oder an ein *anderes* Objekt senden.

Damit erkennbar ist, welche Methode zur Ausführung gelangen soll, ist folgender Sachverhalt wichtig:

- Jede Message wird durch einen Namen beschrieben, der als *Message-Selektor* bezeichnet wird. Entsprechend dem Aufbau von Methoden-Selektoren besteht ein Message-Selektor daher aus einem oder aus mehreren Selektoren.

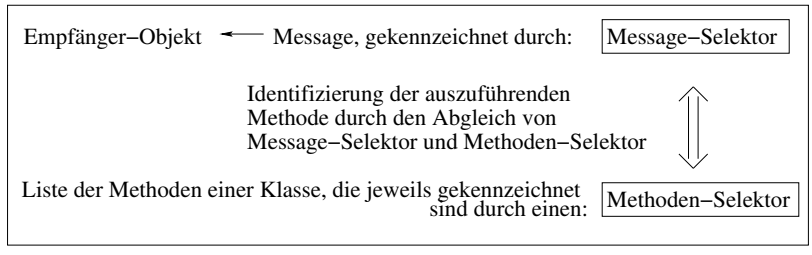


Abbildung 1.14: Message- und Methoden-Selektor

Nach der Zustellung einer Message prüft das Empfänger-Objekt, welche Methode zur Ausführung gelangen soll. Dazu wird der Message-Selektor – innerhalb der Klasse, aus der das Objekt instanziiert wurde – mit sämtlichen Methoden-Selektoren abgeglichen, deren zugehörige Methoden dem Empfänger-Objekt bekannt sind. Es gelangt diejenige Methode zur Ausführung, deren Methoden-Selektor mit dem Message-Selektor übereinstimmt.
Zum Beispiel kann, nachdem eine Instanz namens “WerteErfassung11” – durch eine Instanziierung aus der Klasse “WerteErfassung” – eingerichtet wurde, dieser Instanz eine Message mit dem Message-Selektor “initialisierenErfassung” zugestellt werden.
Um diese Anforderung beim Einsatz von SMALLTALK zu formulieren, ist die folgende Vorschrift zu beachten:

- Der Name des Empfänger-Objektes ist dem Message-Selektor voranzustellen.

In unserem Fall ist somit der Message-Selektor “initialisierenErfassung” wie folgt hinter dem Namen “WerteErfassung11” aufzuführen:

```
WerteErfassung11 initialisierenErfassung
```

Durch diese Anforderung wird bewirkt, daß dem Empfänger-Objekt “WerteErfassung11” eine Message mit dem Message-Selektor “initialisierenErfassung” zugestellt wird. Daraufhin wird die Methode “initialisierenErfassung” von der Instanz “WerteErfassung11” innerhalb der für sie zugänglichen Methoden – als Bestandteil der Klasse “WerteErfassung” – erkannt und ausgeführt.
Damit Objekte wie z.B. “WerteErfassung11” eingerichtet und geeignete Messages wie z.B. “initialisierenErfassung” zur Ausführung des Lösungsplans an sie gerichtet werden können, muß eine *Programmierumgebung* zur Verfügung stehen, innerhalb der die Klasse “WerteErfassung” mit der Instanz-Variablen “werteBag” und den Methoden “initialisierenErfassung”, “erfassenWert”, “entfernenErfassungsfenster”, “festlegenUeberschrift” und “durchfuehrenErfassung” erstellt werden kann.

Hinweis: Die von uns eingesetzte Programmierumgebung stellen wir im Kapitel 2 vor.

Um die von uns zur Lösung von PROB-1-1 benötigte Klasse “WerteErfassung” einzurichten, nutzen wir die Vorzüge der objekt-orientierten Programmierung, indem wir uns auf eine Grundausstattung von Klassen stützen, die innerhalb der Programmierumgebung einer objekt-orientierten Programmiersprache bereitgestellt wird.

- Die Gesamtheit aller Klassen, die unmittelbar nach der Installation der Programmierumgebung dem Anwender zur Verfügung stehen, werden zur Unterscheidung von denjenigen Klassen, die im Rahmen der Programmierung *neu* eingerichtet werden, als *Basis-Klassen* (System-Klassen) bezeichnet. Entsprechend nennen wir eine Methode dann eine *Basis-Methode*, wenn sie in einer Basis-Klasse vereinbart ist.

Mit der SMALLTALK-Programmierumgebung steht eine Vielzahl von Basis-Klassen bereit, die sich sukzessiv um weitere Klassen ergänzen lassen, indem geeignete Attribute festgelegt und neue Methoden unter Einsatz von Basis-Methoden vereinbart werden.
Auf der Grundlage aller Basis-Klassen sowie der Möglichkeit, neue Klassen einrichten zu können, läßt sich folglich jeder objekt-orientierte Lösungsplan unter dem Gesichtspunkt entwerfen, daß eine Programmierung gemäß der Forderung “vom Allgemeinen zum Speziellen” durchgeführt wird, indem z.B. eine geeignete Auswahl von *allgemein* zur Verfügung stehenden Fenster-Bausteinen zum Aufbau eines *speziellen* Fensters verwendet wird.

Hinweis: Fenster und Fenster-Bausteine – wie z.B. Schaltfelder und Menüs – sind *Objekte*, die als Instanzen spezieller Basis-Klassen eingerichtet werden und deren Erscheinungsformen durch die Bildschirmanzeige wiedergegeben werden. Um die Anzeige von speziell aufgebauten Fenstern anzufordern, die im Rahmen eines Lösungsplans von Bedeutung sind, müssen folglich geeignete Instanziierungen derartiger Basis-Klassen vorgenommen werden.

Wir werden diese Möglichkeiten nutzen, indem wir die benötigte Instanz-Variable “werteBag” als *Sammler* einrichten und dazu eine Instanziierung aus der Basis-

Klasse “Bag” vornehmen werden.

Um die innerhalb der Klasse “WerteErfassung” festzulegenden Methoden zu vereinbaren, werden wir uns unter anderem auf die Basis-Methoden “openWindow” (zur Anzeige eines Fensters am Bildschirm), “close” (zur Entfernung eines Fensters vom Bildschirm) und “setFocus” (zur Plazierung des Cursors in einem Fenster) stützen.

Wie wir derartige Basis-Methoden zur Umsetzung unserer Lösungspläne unter Einsatz der Programmierumgebung des SMALLTALK-Systems verwenden können, stellen wir in den folgenden Kapiteln dar.

1.3.6 Zusammenfassung

Bevor wir beschreiben, welche Vorkehrungen zu treffen sind, um einen Lösungsplan formulieren und zur Ausführung bringen zu können, stellen wir abschließend einige der Kerngedanken zusammen, die für das objekt-orientierte Programmieren grundlegend sind:

- Die Träger problem-spezifischer Daten sind als Objekte zu modellieren, indem ihre charakteristischen Eigenschaften in Form von Attributen bestimmt und geeignete Instanz-Variablen in einer diese Objekte typisierenden Klassen-Vereinbarung festgelegt werden (statische Sicht).

- Um Problemstellungen zu lösen, müssen geeignete Handlungen von Objekten ausführbar sein. Diese Handlungen sind innerhalb von Klassen-Vereinbarungen als Methoden anzugeben (funktionale Sicht).

- Die Problemlösung wird dadurch bewirkt, daß Objekt-Exemplare – durch Instanziierungen aus geeigneten Klassen – als Instanzen eingerichtet und Zustandsänderungen unterworfen werden. Dazu sind Anforderungen zu formulieren, durch die den Instanzen jeweils eine geeignete Message zugestellt und damit die Ausführung der jeweils benötigten Methode bewirkt wird (dynamische Sicht).

Kapitel 2

Vorbereitungen zur Durchführung des Lösungsplans

2.1 Start des SMALLTALK-Systems

Im Kapitel 1 haben wir erläutert, daß es zur Lösung der Problemstellung PROB-1-1 erforderlich ist, Erfassungsprozesse durch Instanziierungen der Klasse "WerteErfassung" zur Ausführung zu bringen und anschließend die erfaßten Werte einer nachfolgenden Auswertung zugänglich zu machen.

Wir haben festgestellt, daß zur Klasse "WerteErfassung" sowohl Angaben zu der Instanz-Variablen "werteBag" als auch zu den Methoden "initialisierenErfassung", "erfassenWert", "entfernenErfassungsfenster", "festlegenUeberschrift" und "durchfuehrenErfassung" gehören.

Im Hinblick auf die Lösung unserer Problemstellung ist folglich zunächst die Klasse "WerteErfassung" zu vereinbaren, so daß anschließend Instanziierungen dieser Klasse vorgenommen und geeignete Messages an die eingerichteten Instanzen geschickt werden können.

- Zur Durchführung dieser Schritte setzen wir die Programmiersprache SMALLTALK in Verbindung mit einer Programmierumgebung ein, die von der Firma "ObjectShare" in Form des Software-Produktes "Smalltalk Express" zur kostenlosen Nutzung bereitgestellt wird.

  Hinweis: "Smalltalk Express" ist im INTERNET unter der folgenden URL-Adresse abrufbar: "http://www.objectshare.com/se/seinfo.htm".

Diese Programmierumgebung, die wir im folgenden als "SMALLTALK-System" bezeichnen, setzt sich aus mehreren Komponenten zusammen, zu denen ein Editor (zur Texteingabe und Textbearbeitung), ein Sprach-Übersetzer (zur Umformung von Anforderungen in ausführbare Maschineninstruktionen), ein Linkage-Editor (zum Erzeugen ausführbarer Methoden), ein Loader (zum Starten einer Methodenausführung), ein Werkzeug zur Klassen-Verwaltung (dem Klassen-Hierarchie-Browser) und ein Werkzeug zum Programmtest zählen. Das von uns eingesetzte System stellt außerdem in Form von "WindowBuilder Pro/V" ein komfortables Werkzeug zur interaktiven Erstellung von Fenstern zur Verfügung.

Hinweis: Das Handbuch für dieses Werkzeug ist im INTERNET unter der folgenden URL-Adresse abrufbar: "http://www.objectshare.com/se/stxdocs.htm".

Nachdem das SMALLTALK-System installiert ist, führt der Start dieses Systems zur folgenden Bildschirmanzeige mit der Meldung “Welcome to Smalltalk Express!”:

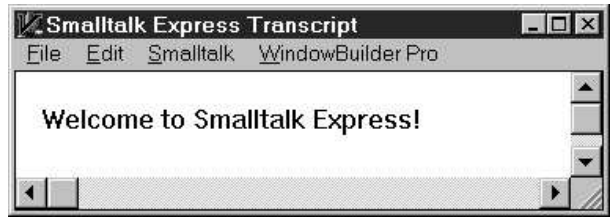


Abbildung 2.1: Das Transcript-Fenster

Hinweis: Um den Dialog mit dem SMALLTALK-System zu beenden, kann die Tastenkombination “Alt+F4” betätigt und im anschließend angezeigten Dialogfeld das Schaltfeld mit der Aufschrift “Yes” bestätigt werden.

Das angezeigte *Transcript-Fenster* wird von uns dazu verwendet, Ergebnisse ausgeben zu lassen, die im Rahmen der Ausführung eines Lösungsplans angezeigt werden sollen.

2.2 Aufbau des Erfassungsfensters

Im folgenden wollen wir das von uns im Abschnitt 1.2 beschriebene Erfassungsfenster durch einen Dialog mit dem Werkzeug “WindowBuilder Pro/V” einrichten lassen. Daher wählen wir aus der Menü-Leiste des Transcript-Fensters das Menü “WindowBuilder Pro” aus. Anschließend stellt sich der Bildschirminhalt wie folgt dar:

Abbildung 2.2: Menü-Optionen von “Window Builder Pro/V”

Hinweis: Da wir im folgenden erläutern werden, wie sich unser Erfassungsfenster schrittweise aufbauen läßt, muß jeder Leser, der diese Schritte am Computer nachvollziehen will, peinlichst genau auf seine Eingaben achten!
Diese Forderung ist natürlich auch für alle weiteren Beispiele, die am Computer nachvollzogen werden sollen, in gleicher Weise zu berücksichtigen.

Nachdem wir die Menü-Option “New Window” ausgewählt haben, wird das WindowBuilder-Fenster mit der Titel-Zeile

`WindowBuilder Pro: [Untitled]`

wie folgt am Bildschirm angezeigt:

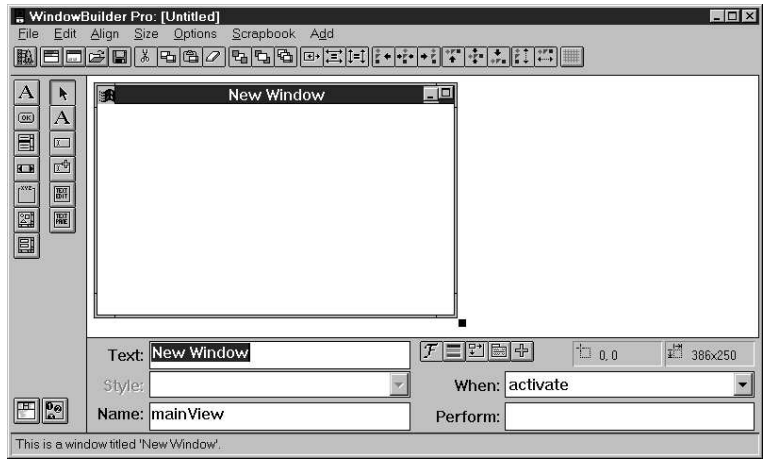


Abbildung 2.3: Das WindowBuilder-Fenster mit dem Arbeitsfeld

Im Arbeitsfeld des WindowBuilder-Fensters ist ein Rahmenfenster mit der Titel-Zeile “New Window” als Basis für das von uns aufzubauende Erfassungsfenster enthalten. Dieses Rahmenfenster, in dem wir die von uns benötigten Fenster-Bausteine schrittweise – im Dialog mit dem Werkzeug “WindowBuilder Pro/V” – erzeugen müssen, wird fortan als “Erfassungsfenster” bezeichnet.
Um die standardmäßig eingestellte Titel-Zeile zu ändern, betätigen wir die Leerzeichen-Taste. Daraufhin wird der im WindowBuilder-Feld “Text:” (unten links) eingetragene Text “New Window” gelöscht sowie die Titel-Zeile des Erfassungsfensters mit Leerzeichen gefüllt. Anschließend ist im WindowBuilder-Fenster der folgende Ausschnitt mit dem Erfassungsfenster erkennbar:

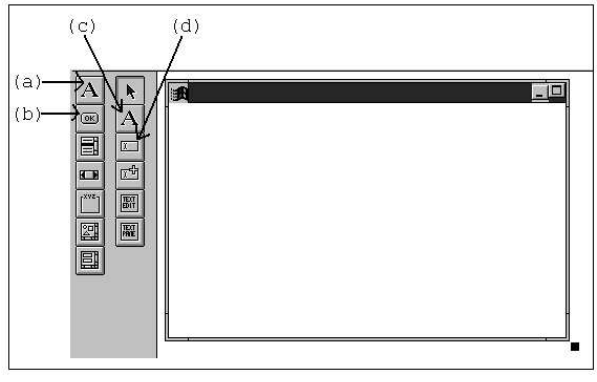


Abbildung 2.4: Leiste mit den Basis-Tool-Feldern

Zur Festlegung der von uns benötigten Fenster-Bausteine sind die Tool-Felder am linken Rand des WindowBuilder-Fensters in geeigneter Form einzusetzen.
Die Einrichtung des Textfeldes mit dem Text “Wert:” bereiten wir dadurch vor, daß wir zunächst auf das Tool-Feld (a) und anschließend auf das Tool-Feld (c) klicken. Nachdem wir die linke obere Ecke des Textfeldes durch einen Mausklick im Erfassungsfenster festgelegt haben, erscheint ein Textfeld mit dem Text “Static Text”, der gleichzeitig im WindowBuilder-Feld “Text:” angezeigt wird. Durch die Eingabe des Textes “Wert:” legen wir die dem Eingabefeld vorangestellte Beschriftung fest.

Um unser Eingabefeld innerhalb des Erfassungsfensters aufzubauen, klicken wir zunächst auf das Tool-Feld (a) und anschließend auf das Tool-Feld (d), so daß wir mit dem Mauszeiger die linke obere Ecke des Eingabefeldes festlegen können.
Nachdem wir mit der Maus neben das zuvor eingerichtete Textfeld gezeigt haben, rufen wir im Erfassungsfenster das Eingabefeld durch einen nachfolgenden Mausklick ab.

- Um im Rahmen des Lösungsplans auf einen in das Eingabefeld eingetragenen Wert zugreifen und – durch Vorbesetzung mit Leerzeichen – das Erscheinungsbild des Eingabefeldes ändern zu können, vergeben wir für dieses Feld einen Namen. Wir kennzeichnen das Eingabefeld durch den Namen “eingabeFeld” und verabreden diese Bezeichnung, indem wir diesen Namen in das WindowBuilder-Feld “Name:” (unten links) eintragen.

Zum Festlegen des Schaltfeldes “erfasse” klicken wir als nächstes auf das Tool-Feld (b). Daraufhin erhalten wir die folgende Anzeige:

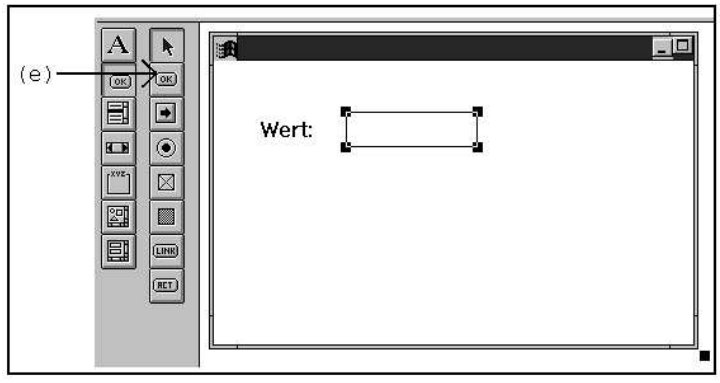


Abbildung 2.5: Leiste mit geänderten Basis-Tool-Feldern

Zum Festlegen des Schaltfeldes klicken wir zunächst auf das Tool-Feld (e). Anschliessend legen wir die linke obere Ecke des Schaltfeldes innerhalb des Erfassungsfensters fest, indem wir den Mauszeiger unter den zuvor erzeugten Text “Wert:” positionieren.

Nach dem sich anschließenden Mausklick erscheint ein Schaltfeld mit der Aufschrift “Button”. Wir ändern diese Aufschrift, indem wir den Text “erfasse” im WindowBuilder-Feld “Text:” eintragen.
Daraufhin erscheint das WindowBuilder-Fenster in der folgenden Form:

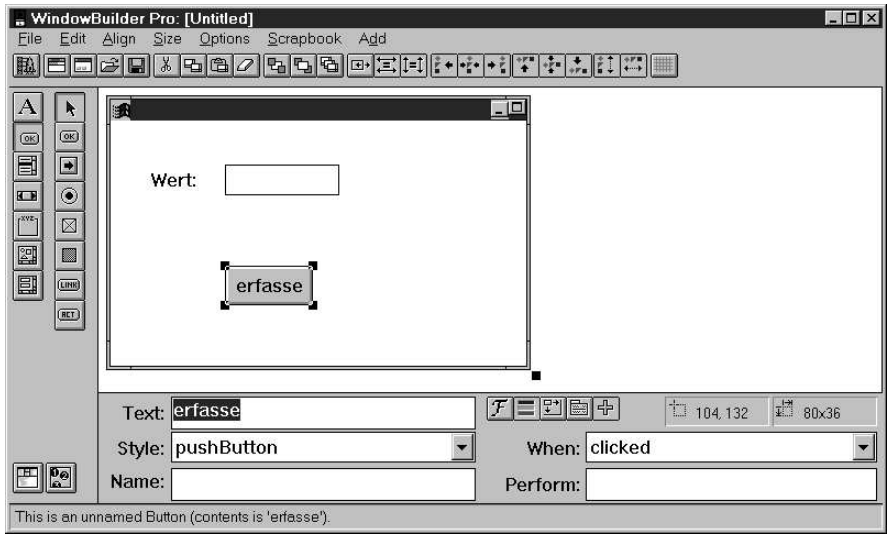


Abbildung 2.6: Einrichtung eines Schaltfeldes

Da ein Mausklick auf das Schaltfeld “erfasse” während des Erfassungsprozesses die Ausführung der durch den Methoden-Selektor “erfassenWert” gekennzeichneten Methode bewirken soll, muß dieser Methoden-Selektor dem Schaltfeld “erfasse” zugeordnet werden. Dies läßt sich dadurch erreichen, daß dieser Methoden-Selektor in das WindowBuilder-Feld “Perform:” (unten rechts) eingetragen wird.

Hinweis: Dabei müssen wir darauf achten, daß in dem WindowBuilder-Kombinationsfeld “When:” (unten rechts) der Text “clicked” angezeigt wird.

- Es ist zu beachten, daß allein die *Zuordnung* des Schaltfeldes “erfasse” zur Methode “erfassenWert” hergestellt wurde. Die Methode “erfassenWert” selbst werden wir zu einem späteren Zeitpunkt festlegen.

Zur Einrichtung des Schaltfeldes “Erfassungsende” ist genauso zu verfahren, wie wir es soeben für den Aufbau des Schaltfeldes “erfasse” geschildert haben. Wir müssen somit wiederum zunächst auf das Tool-Feld (b) und auf das Tool-Feld (e) klicken, anschließend den Mauszeiger rechts von dem Schaltfeld “erfasse” positionieren und danach einen Mausklick durchführen. Die voreingestellte Aufschrift “Button” ist daraufhin durch den Text “Erfassungsende” zu ersetzen. Letztlich ist der Methoden-Selektor “entfernenErfassungsfenster” in das WindowBuilder-Feld “Perform:” einzutragen, so daß das eingerichtete Schaltfeld mit der durch den Methoden-Selektor “entfernenErfassungsfenster” gekennzeichneten Methode verbunden ist.

Nach dieser Zuordnung wird das WindowBuilder-Fenster wie folgt angezeigt:

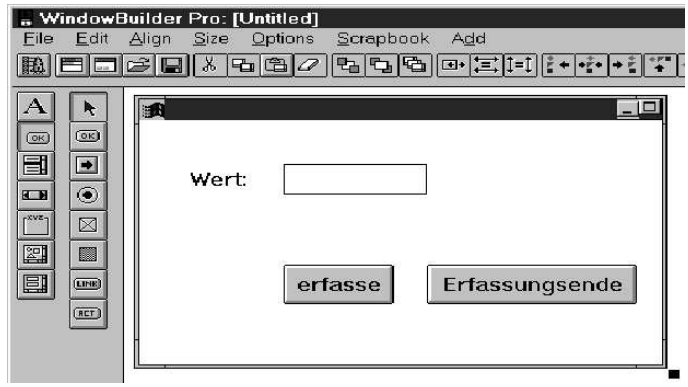


Abbildung 2.7: Vollständig aufgebautes Erfassungsfenster

Nachdem wir die Form unseres Erfassungsfensters innerhalb des WindowBuilder-Fensters festgelegt haben, wählen wir zunächst das Menü "File" aus dessen Menü-Leiste und anschließend die Menü-Option "Save As..." aus diesem Menü aus. In dem daraufhin angezeigten Dialogfeld

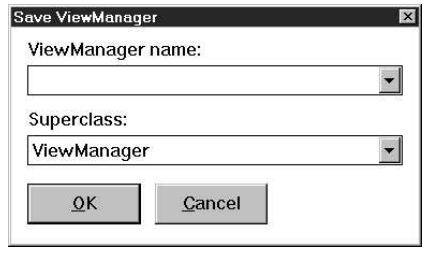


Abbildung 2.8: Sicherung des Erfassungsfensters

tragen wir den von uns vorgesehenen Klassennamen "WerteErfassung" in das Textfeld "ViewManager name:" ein und bestätigen das Dialogfeld durch das Schaltfeld "OK".

Hinweis: Der Name "ViewManager" kennzeichnet eine Basis-Klasse des SMALLTALK-Systems, die neben einem allgemein gehaltenen Bauplan für den Aufbau von Fenstern eine Fülle von Basis-Methoden enthält, über die sich die Kommunikation des Anwenders – unter Einsatz von geeignet gewählten Fenster-Bausteinen – abwickeln läßt.

Durch diese Sicherung richtet das SMALLTALK-System die Klasse "WerteErfassung" ein, indem es den von uns vorgenommenen Aufbau des Erfassungsfensters in geeignete Anforderungen an das SMALLTALK-System umsetzt. Aus dieser Umformung resultiert z.B. die Methode "createViews", durch deren Ausführung das von uns konzipierte Erfassungsfenster strukturell bestimmt wird. Um einen Eindruck

von der automatischen Methoden-Generierung zu geben, enthält der Anhang A.1 die vollständige Methoden-Vereinbarung von “createViews”.

Zusätzlich zur Methode “createViews” sind vom SMALLTALK-System zwei weitere Methoden-Vereinbarungen innerhalb der Klasse “WerteErfassung” vorgenommen worden, durch die die zuvor festgelegten Bezüge zu den Schaltfeldern “erfasse” und “Erfassungsende” beschrieben werden.

Die drei in dieser Form *automatisch* eingerichteten Methoden sind für uns als Gerüste für die klassen-spezifischen Angaben anzusehen, die von uns noch – durch geeignete Arbeitsschritte – nachbereitet werden müssen.

Hinweis: Zum Beispiel müssen wir den Methoden-Selektor “createViews” in den von uns konzipierten Methoden-Selektor “initialisierenErfassung” umbenennen und diese Methode durch zusätzliche Anforderungen ergänzen (siehe unten).

2.3 Der Klassen-Hierarchie-Browser

Nachdem wir die Anzeige des WindowBuilder-Fensters durch die Menü-Option “Exit” des Menüs “File” vom Bildschirm entfernt haben, wollen wir die Vereinbarung der Klasse “WerteErfassung” ergänzen und modifizieren.

Dazu rufen wir den *Klassen-Hierarchie-Browser* im Transcript-Fenster durch die Menü-Option “Browse Classes” des Menüs “File” auf. Dies führt zur folgenden Anzeige des *Klassen-Hierarchie-Browser-Fensters*:

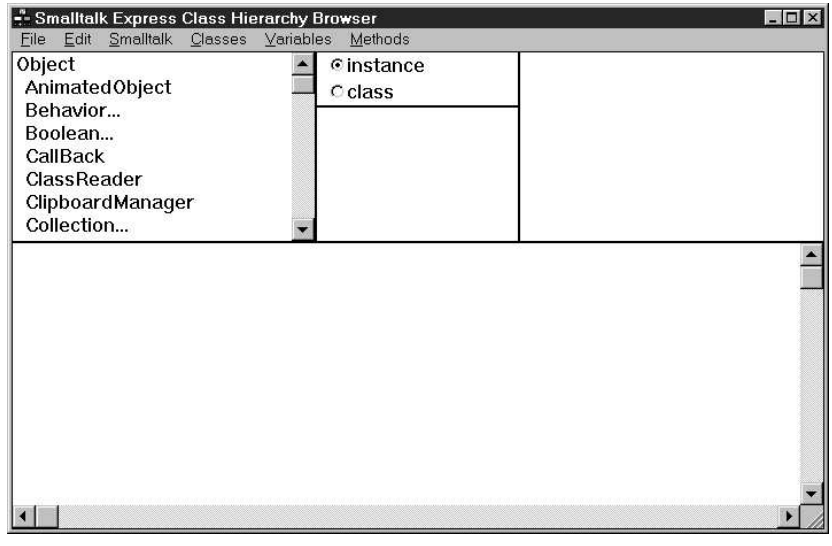


Abbildung 2.9: Das Klassen-Hierarchie-Browser-Fenster

Dieses Fenster ist wie folgt strukturiert:

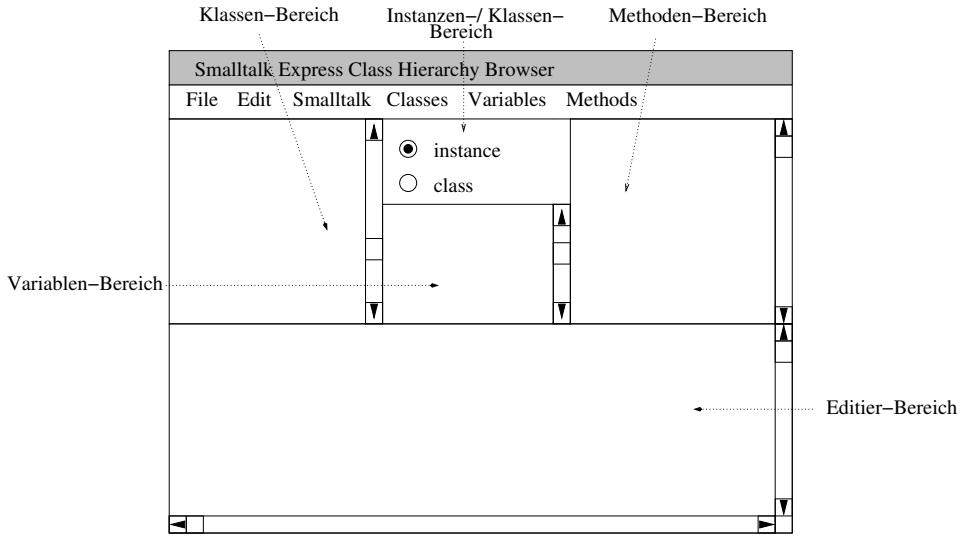


Abbildung 2.10: Struktur des Klassen-Hierarchie-Browser-Fensters

Hinweis: In sämtlichen Bereichen – mit Ausnahme des Instanzen-/Klassen-Bereichs – lassen sich Rollbalken zum "Blättern" einsetzen.
Je nachdem, welches der beiden Optionsfelder aktiviert ist, werden verschiedene Sichtweisen auf eine Klasse angezeigt.
Es ist zu beachten, daß für unsere jetzigen Betrachtungen das Optionsfeld "instance" im Instanzen-/ Klassen-Bereich aktiviert sein muß. Ist dies geschehen, so werden im Methoden-Bereich die Methoden-Selektoren derjenigen Methoden angezeigt, die von Instanzen der aktuell eingestellten Klasse ausgeführt werden können.

- Das Klassen-Hierarchie-Browser-Fenster zählt – genauso wie das Transcript-Fenster, das WindowBuilder-Fenster und das Workspace-Fenster, in dem sich Anforderungen an das SMALLTALK-System eintragen lassen (siehe Abschnitt 3.1) – zu den vom SMALLTALK-System zur Kommunikation mit dem Anwender bereitgestellten Fenstern.

  Grundsätzlich kann – über einen Klick mit der *rechten* Maustaste – in jedem dieser Fenster ein *Kontext-Menü* angefordert werden, dessen jeweils angezeigte Menü-Optionen *kontextabhängig* sind, so daß das jeweilige Fenster den Aufbau des Kontext-Menüs bestimmt.

  Hinweis: Es ist möglich, mehrere Klassen-Hierarchie-Browser-Fenster gleichzeitig zu öffnen.

Im Klassen-Bereich des Klassen-Hierarchie-Browser-Fensters sind die Namen der dem SMALLTALK-System bekannten Klassen – beginnend mit der Basis-Klasse "Object" – angezeigt.

Hinweis: Sofern einem Klassennamen – wie z.B. “Behavior” – drei Punkte folgen, können weitere Klassen durch einen Doppelklick auf diesen Namen sichtbar gemacht werden. Die daraus resultierende Anzeige kann durch einen weiteren Doppelklick wieder aufgehoben werden. Die jeweilige Form der Anzeige kann alternativ durch das Menü “Classes” unter Einsatz der Menü-Optionen “Show Subclasses” (zur Einblendung von Klassennamen) bzw. “Hide Subclasses” (zur Ausblendung von Klassennamen) gesteuert werden.

Wird eine Klasse durch einen Mausklick auf den Klassennamen ausgewählt, und ist im Instanz-/Klassenbereich das Optionsfeld “instance” aktiviert, so erscheinen im Variablen-Bereich – unterhalb der beiden Optionsfelder “instance” und “class” – alle in dieser Klasse bekannten Instanz-Variablen.
Sollen die Angaben, mit denen eine bestimmte Klasse vereinbart ist, zur Anzeige gebracht werden, so ist die betreffende Klasse als *aktuelle* Klasse einzustellen. Dazu können wir das Dialogfeld “Smalltalk Express Prompter” in der Form

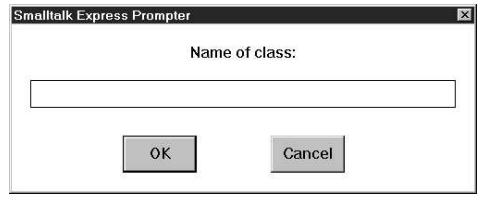


Abbildung 2.11: Suchen einer Klasse

– mittels der Menü-Option “Find Class...” des Menüs “Classes” – anfordern und den Klassennamen im Eingabefeld “Name of class:” mitteilen.
Tragen wir z.B. den Namen “WerteErfassung” in dieses Dialogfeld ein und bestätigen ihn durch das Schaltfeld “OK”, so ergibt sich die folgende Anzeige:

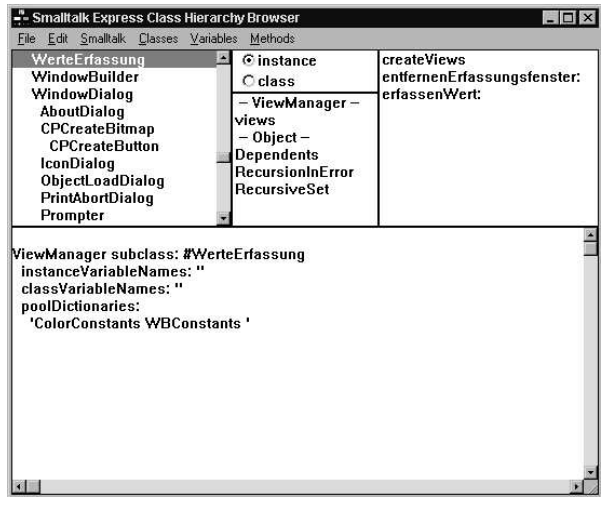


Abbildung 2.12: Anzeige der Klasse “WerteErfassung”

Im Editier-Bereich ist das zuvor innerhalb des WindowBuilder-Fensters erzeugte Gerüst für die Vereinbarung der Klasse “WerteErfassung” eingetragen. Da wir “werteBag” als Namen der von uns benötigten Instanz-Variablen festlegen wollen, tragen wir daher diesen Namen in die zweite Zeile ein, so daß sich der folgende Text ergibt:

```
instanceVariableNames: 'werteBag'
```

Hinweis: Eine derartige Änderung darf nur vorgenommen werden, sofern noch keine Instanziierung der Klasse “WerteErfassung” erfolgt ist.

Wir sichern diese Änderung, indem wir im Klassen-Hierarchie-Browser-Fenster das Menü “File” anwählen und dort die Menü-Option “Save” bestätigen.

Die Methoden-Selektoren, die für die Klasse “WerteErfassung” bislang *automatisch* festgelegt wurden, sind in dem angezeigten Fenster innerhalb des Methoden-Bereichs aufgeführt – es sind dies die Selektoren “createViews”, “entfernenErfassungsfenster:” und “erfassenWert:”.

Hinweis: Die von uns bestimmten Namen “entfernenErfassungsfenster” und “erfassenWert” sind vom SMALLTALK-System automatisch um den Doppelpunkt “:” ergänzt worden (siehe unten).

Der zuerst aufgeführte Methoden-Selektor “createViews” kennzeichnet die Methode, durch deren Ausführung das von uns zuvor – innerhalb des WindowBuilder-Fensters – gestaltete Erfassungsfenster aufgebaut wird. Da wir im Hinblick auf unsere Problemlösung diese Methode in unserem Lösungsplan mit dem Methoden-Selektor “initialisierenErfassung” gekennzeichnet haben, wollen wir die standardmäßig vom Werkzeug “WindowBuilder Pro/V” gewählte Namensvergabe “createViews” in den Namen “initialisierenErfassung” ändern.

Wie wir diese von uns gewünschte Namensänderung erreichen können, schildern wir im nachfolgenden Abschnitt.

2.4 Anzeige und Veränderung von Methoden

Da sich im Dialog mit dem SMALLTALK-System keine unmittelbare Umbenennung einer Methode vornehmen läßt, werden wir die Methode “initialisierenErfassung” neu einrichten und dabei auf die in der Methode “createViews” festgelegten Anforderungen zurückgreifen.

Dazu lassen wir uns zunächst die Vereinbarung der Methode “createViews” innerhalb des Editier-Bereichs des Klassen-Hierarchie-Browser-Fensters anzeigen, indem wir auf den im Methoden-Bereich aufgeführten Methoden-Selektor “createViews” klicken. Anschließend erscheinen im Editier-Bereich die Angaben, durch die die Methode “createViews” festgelegt ist, so daß sich das Klassen-Hierarchie-Browser-Fenster wie folgt darstellt:

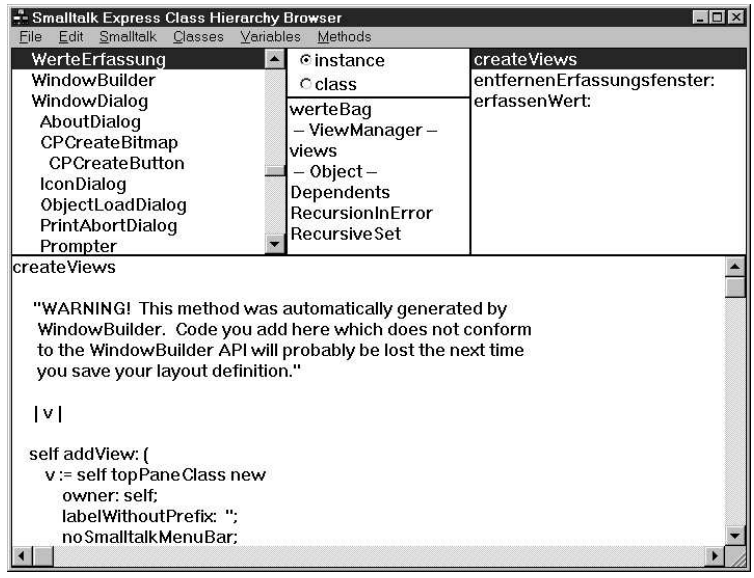


Abbildung 2.13: Anzeige der Methode "createViews"

Hinweis: Die gesamte Methoden-Vereinbarung von "createViews", die vom Werkzeug "WindowBuilder Pro/V" *automatisch* erzeugt wurde, ist im Anhang A.1 angegeben.

Um den Inhalt des Editier-Bereichs zu kopieren, wählen wir zunächst die Menü-Option "Select All" und daraufhin die Menü-Option "Copy" aus dem Menü "Edit". Anschließend aktivieren wir im Menü "Methods" die Menü-Option "New Method". Mit der Menü-Option "Paste" des Menüs "Edit" kopieren wir anschließend die Anforderungen zur Vereinbarung der Methode "createViews" in den Editier-Bereich und löschen daraufhin die im Editier-Bereich automatisch angezeigten Basisangaben, die die Struktur beschreiben, gemäß der eine Methode formal zu vereinbaren ist.

Danach nehmen wir die folgenden Änderungen vor:

- Zunächst löschen wir alle Einträge oberhalb der folgenden Zeile:

  ```
  | v |
  ```

  Danach tragen wir vor dieser Zeile den Methoden-Selektor

  ```
  initialisierenErfassung
  ```

  und unterhalb von "`| v |`" die folgende Anforderung ein:

  ```
  werteBag := Bag new.
  ```

Sichern wir anschließend den Inhalt des Editier-Bereichs über die Menü-Option "Save" des Menüs "File", so stellt sich das Klassen-Hierarchie-Browser-Fenster wie folgt am Bildschirm dar:

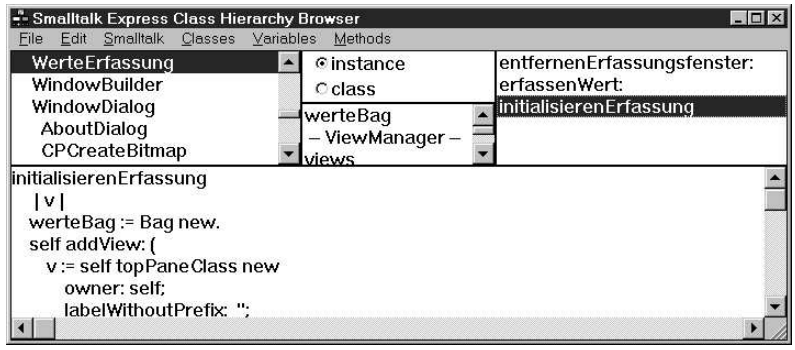


Abbildung 2.14: Anzeige der Methode "initialisierenErfassung"

Es ist erkennbar, daß "initialisierenErfassung" als neuer Methoden-Selektor im Methoden-Bereich aufgenommen worden ist.

Hinweis: Die Methode "createViews" ist nicht mehr im Methoden-Bereich aufgeführt. Da sie nicht weiter benötigt wird, haben wir sie anschließend gelöscht, indem wir im Methoden-Bereich auf den Methoden-Selektor geklickt und die Menü-Option "Remove" im Menü "Methods" bestätigt haben.
Falls wir beim Eintragen der Anforderung "werteBag := Bag new." z.B. statt "werteBag" fälschlicherweise "werrteBag" angegeben haben, erscheint nach der Auswahl der Menü-Option "Save" die markierte Fehlermeldung "undefined". Wir löschen diese Markierung, korrigieren die fehlerhaften Angaben in "werteBag" und bestätigen erneut die Menü-Option "Save".

Beim Aufbau unseres Erfassungsfensters haben wir die Methoden-Selektoren "entfernenErfassungsfenster" und "erfassenWert" im WindowBuilder-Fenster angegeben. Dadurch sind zwei Methoden festgelegt worden, die im Methoden-Bereich durch die Methoden-Selektoren "entfernenErfassungsfenster:" und "erfassenWert:" – mit abschließendem Doppelpunkt – gekennzeichnet sind.

Im Hinblick auf diesen Sachverhalt haben wir – in Ergänzung zur im Abschnitt 1.3.5 angegebenen Benennung von Methoden – die folgende Regel zu beachten:

- Ein Methoden-Selektor, der nur aus einem einzelnen Selektor besteht, *muß* in besonderen Fällen durch einen *Doppelpunkt* beendet werden.

Dies ist z.B. dann erforderlich, wenn die zugehörige Methode – wie im Fall von "erfassenWert:" und "entfernenErfassungsfenster:" – an ein Schaltfeld gebunden wird.

Da bislang nur die Benennung dieser beiden Methoden festgelegt ist, müssen die Anforderungen, die bei diesen Methoden jeweils zur Ausführung gelangen sollen, jeweils für sich im Editier-Bereich eingetragen und anschließend gesichert werden.

Im Hinblick auf die Vorgaben, die wir im Kapitel 1 getroffen haben, müssen innerhalb der Methode "erfassenWert:" Anforderungen formuliert werden, durch die die folgenden Handlungen durchgeführt werden:

- *"erfassenWert"*
  - Übertragung eines Wertes vom Eingabefeld nach "werteBag"
  - Eintragung von Leerzeichen in das Eingabefeld
  - Plazierung des Cursors auf das Eingabefeld

Wir nehmen an dieser Stelle – ohne nähere Begründung – zur Kenntnis, daß diese Handlungen formal durch die folgenden Anforderungen beschrieben werden können:

```
werteBag add: (self paneNamed: 'eingabeFeld') contents.
(self paneNamed: 'eingabeFeld') contents: ''.
(self paneNamed: 'eingabeFeld') setFocus
```

Hinweis: Durch die erste Anforderung wird die Message "add:" mit dem Argument "(self paneNamed: 'eingabeFeld') contents" an das Empfänger-Objekt "werteBag" gerichtet, woraufhin der aktuelle Wert des Eingabefeldes dem Objekt "werteBag" als Wert hinzugefügt wird.
Durch die zweite Anforderung erhält das Eingabefeld die Message "contents: ' ' ", so daß dessen aktueller Wert durch Leerzeichen überschrieben wird.
Durch die dritte Anforderung wird die Message "setFocus" an das Eingabefeld gerichtet, woraufhin der Cursor in diesem Feld positioniert wird.

Im Hinblick auf die Schreibweise ist für aufeinanderfolgende Anforderungen grundsätzlich die folgende Vorschrift zu beachten:

- Je zwei Anforderungen sind durch einen Punkt "." zu trennen.

Nachdem wir im Methoden-Bereich auf den Methoden-Selektor "erfassenWert:" geklickt haben, tragen wir die oben angegebenen Anforderungen hinter dem angezeigten Text im Editier-Bereich ein. Dies führt zur folgenden Anzeige:

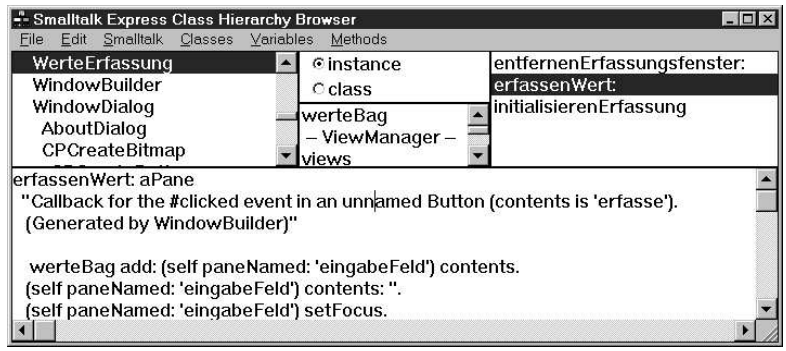


Abbildung 2.15: Festlegen der Methode "erfassenWert:"

Wir sichern diese Änderung, indem wir das Menü “File” anwählen und dort die Menü-Option “Save” bestätigen.
Die im Editier-Bereich enthaltene Angabe

```
"Callback for the #clicked event in an unnamed Button
(contents is 'erfasse'). (Generated by WindowBuilder)"
```

enthält erläuternde Angaben über die Methode “erfassenWert:”, die bei der Ausführung dieser Methode überlesen werden. Es gilt:

- Alle durch das Anführungszeichen “"” eingeleiteten und abgeschlossenen Angaben werden vom SMALLTALK-System als Kommentar-Informationen aufgefaßt.

  Hinweis: Kommentar-Informationen dürfen an beliebigen Positionen innerhalb einer Methode aufgeführt werden.

Um die Methode “entfernenErfassungsfenster:” festzulegen, formen wir die Vorgabe

- *“entfernenErfassungsfenster”*
  – Entfernung des Erfassungsfensters vom Bildschirm

aus Abschnitt 1.3.5 in die Anforderung

```
self close
```

um und legen sie innerhalb der Methode “entfernenErfassungsfenster:” fest. Dazu klicken wir auf den Methoden-Selektor “entfernenErfassungsfenster:” im Methoden-Bereich, tragen diese Anforderung hinter dem einleitenden Kommentar im Editier-Bereich ein, und sichern diese Änderung mittels der Menü-Option “Save” des Menüs “File”.
Nach dieser Sicherung besitzt die Klasse “WerteErfassung” insgesamt die Form:

WerteErfassung	
Instanz–Variablen:	werteBag
Methoden:	initialisierenErfassung erfassenWert: entfernenErfassungsfenster:

Abbildung 2.16: Aktuelle Form der Klasse “WerteErfassung”

2.5 Vereinbarung neuer Methoden

Um die Klasse “WerteErfassung” im Hinblick auf die von uns konzipierten Handlungen zu vervollständigen, müssen wir noch Methoden zur Beschreibung der Handlungen “festlegenUeberschrift” und “durchfuehrenErfassung” vereinbaren.

Grundsätzlich läßt sich eine neue Methode dadurch verabreden, daß die Menü-Option “New Method” des Menüs “Methods” ausgewählt und die zu vereinbarenden Anforderungen nach dem folgenden Schema im Editier-Bereich eingetragen werden:

```
Methoden-Selektor
"Kommentar-Information"
eine oder mehrere Anforderungen
```

Hinweis: Der die Methoden-Vereinbarung einleitende “Methoden-Selektor” wird auch als *Methodenkopf* und die sich anschließenden Kommentar-Informationen und Anforderungen als *Methodenrumpf* bezeichnet. Die Kommentar-Informationen, mit denen beschrieben werden sollte, was die Ausführung der Methode bewirkt, dürfen auch fehlen.

Nachdem wir im Editier-Bereich die Muster-Einträge für den allgemeinen Aufbau von Methoden in der Form

```
messagePattern
   "comments"
  | temporaries |
  statements
```

gelöscht haben, tragen wir den Methoden-Selektor der zu vereinbarenden Methode und die zugehörigen Anforderungen in diesen Bereich ein und sichern die Vereinbarung anschließend durch die Menü-Option “Save” des Menüs “File”.

Zunächst legen wir die von uns konzipierte Handlung

- *“durchfuehrenErfassung”*
  - Anzeige des Erfassungsfensters am Bildschirm
  - Festlegung, daß der Cursor auf das Eingabefeld plaziert wird

durch die folgende Methoden-Vereinbarung fest:

```
durchfuehrenErfassung
self openWindow.
(self paneNamed: 'eingabeFeld') setFocus
```

Anschließend vereinbaren wir für

- *“festlegenUeberschrift”*
  - Festlegung der neuen Überschrift des Erfassungsfensters

die folgende Methoden-Vereinbarung:

```
festlegenUeberschrift: aString
self labelWithoutPrefix: aString
```

Es ist zu beachten, daß wir bei der Methode “festlegenUeberschrift:” den ursprünglich in der Form “festlegenUeberschrift” konzipierten Methoden-Selektor um einen Doppelpunkt ergänzt haben. Dieses Vorgehen stützt sich auf die folgende Regel:

- Ein Methoden-Selektor, der nur aus einem einzelnen Selektor besteht, *muß* durch einen Doppelpunkt ":" beendet werden, sofern in der Message – hinter dem einleitenden Message-Selektor – eine ergänzende Information (wie z.B. eine Überschrift) aufgeführt werden soll, die innerhalb der Methode verwendet wird. Besteht ein Methoden-Selektor aus mehr als einem Selektor, so *muß* der zweite und auch jeder weitere Selektor durch einen Doppelpunkt beendet werden.

Nachdem wir die Methoden-Vereinbarungen vorgenommen haben, sind alle zur Durchführung des Lösungsplans von PROB-1-1 erforderlichen Angaben in der Klasse "WerteErfassung" enthalten.
Wird berücksichtigt, daß einige Methoden-Selektoren durch das Anfügen von Doppelpunkten geändert werden, so stellt sich die Klasse "WerteErfassung" jetzt wie folgt dar:

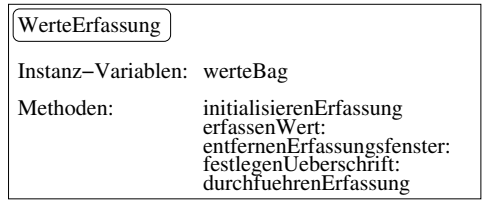


Abbildung 2.17: Klasse "WerteErfassung" zur Lösung von PROB-1-1

Um das Arbeiten mit dem Klassen-Hierarchie-Browser zu beenden, betätigen wir die Tastenkombination "Alt+F4". Anschließend erscheint das Transcript-Fenster, das unmittelbar nach dem Start des SMALLTALK-Systems angezeigt wurde.

Hinweis: Im Transcript-Fenster sind Eintragungen der Form "recompiling WerteErfassung" enthalten, die aus der Sicherung der zuvor festgelegten Angaben für die Klasse "WerteErfassung" resultieren.

2.6 Sicherung

2.6.1 Sicherung des SMALLTALK-Systems

Die Vereinbarung der neuen Klasse mit ihren Methoden ist solange gültig, bis der aktuelle Dialog mit dem SMALLTALK-System beendet wird. Nach erneutem Dialogbeginn steht die Klassen-Vereinbarung, die unter Einsatz des Klassen-Hierarchie-Browsers gesichert wurde, nicht mehr zur Verfügung. Dies liegt daran, daß Sicherungen innerhalb der jeweils aktuellen Klassen-Hierarchie nur temporär für den jeweils aktuellen Dialog Gültigkeit haben.
Dieser Sachverhalt beruht darauf, daß das SMALLTALK-System beim Start in den Hauptspeicher geladen wird und sämtliche Ergänzungen der Klassen-Hierarchie im Hauptspeicher bzw. temporären Auslagerungsdateien gespeichert werden. Wird der aktuelle Dialog beendet, so wird das SMALLTALK-System *nicht* automatisch

geändert, so daß bei einem erneuten Start sämtliche zuvor durchgeführten Änderungen nicht mehr zur Verfügung stehen.

Um eine langfristige Speicherung der geänderten Klassen-Hierarchie zusammen mit den aktuellen Klassen-Vereinbarungen durchzuführen, muß im Transcript-Fenster die Menü-Option “Save Image...” des Menüs “File” bestätigt werden. Daraufhin wird ein Dialogfeld mit der Titel-Zeile “Smalltalk Express Please Confirm” angezeigt, in dem die Sicherung des aktuellen SMALLTALK-Systems durch einen Mausklick auf das Schaltfeld “Yes” angefordert werden muß.

Hinweis: Bei Änderungen am SMALLTALK-System wird dieses Dialogfeld automatisch angezeigt, wenn der Dialog mit dem SMALLTALK-System beendet wird.

Durch diese Sicherung ist nicht nur die *Persistenz* der Klassen und aller globalen Variablen gewährleistet, sondern es wird auch der Inhalt und damit der Zustand aller aktuell am Bildschirm angezeigten Fenster festgehalten, so daß der Zustand, der unmittelbar vor der Sicherung vorlag, beim erneuten Start des SMALLTALK-Systems *automatisch* wieder hergestellt wird (siehe Abschnitt 5.2).

Hinweis: Unter einem persistenten Objekt ist ein Objekt zu verstehen, das *dauerhaft* zur Verfügung steht, d.h. ein persistentes Objekt ist nach seiner Einrichtung jederzeit wieder abrufbar.

2.6.2 Sichern einer Klassen-Vereinbarung

Neben der Möglichkeit, eine neue Klassen-Vereinbarung zu sichern, können einzelne oder mehrere Klassen-Vereinbarungen auch gezielt in *Klassen-Dateien* übertragen werden.

Wollen wir z.B. die aktuelle Vereinbarung der Klasse “‘WerteErfassung” innerhalb einer *Klassen-Datei* sichern, so stellen wir diese Klasse als aktuelle Klasse im Klassen-Hierarchie-Browser-Fenster ein und wählen die Menü-Option “File Out...” des Menüs “Classes” aus. Daraufhin wird ein Dialogfeld angezeigt, in dessen Eingabefeld “Dateiname:” der Dateiname der *Klassen-Datei* einzutragen ist.

Hinweis: Als Grundname ist der Klassenname (reduziert um bestimmte Vokale) und als Ergänzung die Zeichenfolge “cls” voreingestellt. In unserem Fall wird der Name “WrtErfss.cls” als Name für die *Klassen-Datei* vorgeschlagen.

Nach der Festlegung des Dateinamens und der Bestätigung des Dialogfeldes werden alle Angaben der Klassen-Vereinbarung von “WerteErfassung” innerhalb der *Klassen-Datei* gesichert.

- Die *Klassen-Datei* ist eine Text-Datei, in der die Klassen-Vereinbarung in einer SMALLTALK-spezifischen Struktur – dem *Chunk-Format* – gespeichert wird. Dieses Format sichert eine plattform-unabhängige Speicherung von Vereinbarungen, die bei der Portierung von einem zu einem anderen SMALLTALK-System von Bedeutung ist.

Hinweis: Die *Klassen-Datei* ist in drei Abschnitte gegliedert, die jeweils durch das Ausrufungszeichen “!” voneinander abgetrennt sind. Wie die Darstellung der Klassen-Vereinbarung von “WerteErfassung” im Chunk-Format aussieht, ist im Anhang unter A.2 angegeben.

Sofern nicht nur die Vereinbarung der aktuell eingestellten Klasse, sondern zusätzlich auch alle Vereinbarungen sämtlicher dieser Klasse – im Rahmen der Klassen-

Hierarchie – untergeordneten Klassen gesichert werden sollen, ist die Menü-Option "File Out All..." anstelle der Menü-Option "File Out..." des Menüs "Classes" zu verwenden.

2.6.3 Laden einer Klassen-Vereinbarung

Soll eine Klassen-Vereinbarung aus einer *Klassen-Datei* in das SMALLTALK-System übernommen werden, so ist im Klassen-Hierarchie-Browser-Fenster die Menü-Option "Install..." des Menüs "File" anzuwählen.
Nachdem der Dateiname der *Klassen-Datei* in das Eingabefeld "Dateiname:" im Dialogfeld eingetragen und durch das Schaltfeld "OK" bestätigt ist, übernimmt das SMALLTALK-System die Klassen-Vereinbarung aus der *Klassen-Datei*.

Hinweis: Dabei ist zu beachten, daß die Aktualisierung der Anzeige des Klassen-Hierarchie-Browser-Fensters über die Menü-Option "Update" des Menüs "Classes" angefordert werden muß. Sollen vor der Übernahme noch Änderungen am Inhalt der *Klassen-Datei* durchgeführt werden, so kann die *Klassen-Datei* im Klassen-Hierarchie-Browser-Fenster durch die Menü-Option "Open..." des Menüs "File" eröffnet, anschließend editiert und danach wieder geschlossen werden.
Bei der Editierung ist auf die Schreibweise des Chunk-Formats zu achten. So sind z.B. zwei aufeinanderfolgende Ausrufungszeichen durch ein Leerzeichen zu trennen.

2.6.4 Sichern und Laden von Methoden-Vereinbarungen

Es lassen sich nicht nur vollständige Klassen-Vereinbarungen gezielt sichern, sondern es besteht auch die Möglichkeit, ausgewählte Methoden einzeln in eine *Methoden-Datei* zu übertragen, von wo sie wieder in eine Klassen-Vereinbarung übernommen werden können.
Um die Sicherung durchzuführen, ist die zu speichernde Methode als aktuelle Methode im Klassen-Hierarchie-Browser-Fenster einzustellen und die Menü-Option "File Out..." des Menüs "Methods" auszuwählen. Daraufhin wird ein Dialogfeld angezeigt, in dessen Eingabefeld "Dateiname:" der Dateiname der *Methoden-Datei* einzutragen ist.

Hinweis: Als Grundname ist der Methodenname (reduziert um bestimmte Vokale) und als Ergänzung die Zeichenfolge "mth" voreingestellt.

Nach der Festlegung des Dateinamens und der Bestätigung des Dialogfeldes wird die Methoden-Vereinbarung innerhalb der *Methoden-Datei* (unter Einschluß von Angaben der zugehörigen Klasse) – im Chunk-Format – gesichert.
Soll eine Methoden-Vereinbarung aus einer *Methoden-Datei* in die aktuell im Klassen-Hierarchie-Browser-Fenster eingestellte Klasse übernommen werden, so ist die Menü-Option "Install..." des Menüs "File" anzuwählen.
Nachdem der Dateiname der *Methoden-Datei* in das Eingabefeld "Dateiname:" im Dialogfeld eingetragen und durch das Schaltfeld "OK" bestätigt ist, übernimmt das SMALLTALK-System die Methoden-Vereinbarung in die aktuelle Klassen-Vereinbarung.

Kapitel 3

Durchführung des Lösungsplans

3.1 Einrichtung einer Instanz

Das Workspace-Fenster

Nachdem wir im Kapitel 2 erläutert haben, wie die im Kapitel 1 – zur Lösung der Problemstellung PROB-1-1 – entwickelte Klasse "WerteErfassung" aufgebaut und dem SMALLTALK-System bekannt gemacht werden kann, wenden wir uns jetzt der *Ausführung* unseres Lösungsplans zu.

Um den Erfassungsprozeß durch eine Instanziierung der Klasse "WerteErfassung" zur Ausführung zu bringen, müssen wir geeignete Anforderungen an das SMALLTALK-System richten. Dazu verwenden wir ein *Workspace-Fenster*, das sich über die Menü-Option "New Workspace" des Menüs "File" eröffnen läßt.

Sofern wir die Zeile

```
WerteErfassung11 := WerteErfassung new
```

in das Workspace-Fenster eingetragen haben, wird es wie folgt angezeigt:

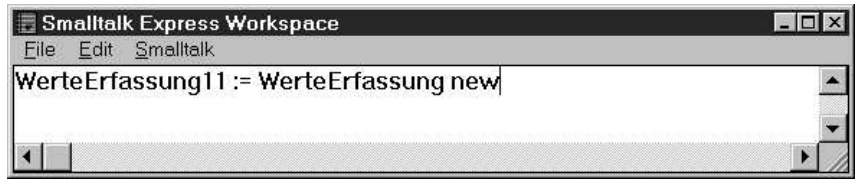


Abbildung 3.1: Das Workspace-Fenster

Hinweis: Weil wir deutlich machen wollen, daß ein Erfassungsprozeß für die Punktwerte der Jahrgangsstufe 11 durchgeführt werden soll, verwenden wir die Bezeichnung "WerteErfassung11".

Um die Anforderung vom SMALLTALK-System bearbeiten zu lassen, markieren wir sie und klicken auf das Menü "Smalltalk", so daß wir die folgende Anzeige erhalten:

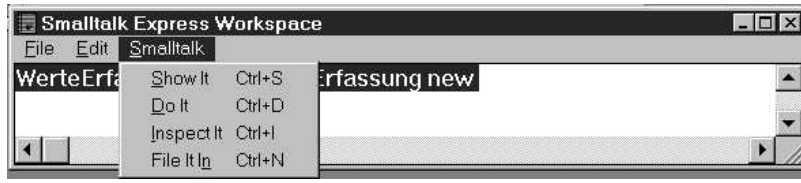


Abbildung 3.2: Das Menü “Smalltalk”

Indem wir die Menü-Option “Do It” wählen, wird die markierte Anforderung vom SMALLTALK-System bearbeitet.

- Im folgenden verfahren wir entsprechend, wenn Anforderungen an das SMALLTALK-System zu richten sind.
  Um eine oder mehrere (im Workspace-Fenster eingetragenen und durch einen Punkt voneinander abgegrenzten) Anforderungen *ausführen* zu lassen, führen wir zunächst eine Markierung durch und bestätigen anschließend die Menü-Option “Do It” des Menüs “Smalltalk”.

Die Methode “new”

Durch die Ausführung der Anforderung

```
WerteErfassung11 := WerteErfassung new
```

wird – wie wir es durch den im Kapitel 1 angegebenen Lösungsplan beabsichtigt haben – vom SMALLTALK-System eine globale Variable namens “WerteErfassung11” eingerichtet, nachdem die im Dialogfeld

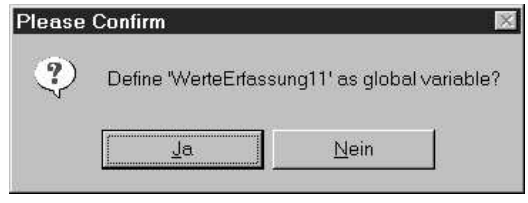


Abbildung 3.3: Einrichtung einer globalen Variablen

gestellte Frage mittels des Schaltfeldes “Ja” beantwortet worden ist.
An die globale Variable “WerteErfassung11” wird ein Objekt gebunden, das durch diejenige Vorschrift bestimmt wird, die auf der rechten Seite des Zuweisungssymbols “:=” angegeben ist.
Grundsätzlich gilt:

- Eine Anforderung, die das Zuweisungssymbol “:=” enthält, wird *Zuweisung* genannt.

Durch das Zuweisungssymbol wird die Bindung eines Objekts an einen Variablennamen festgelegt. Der Variablenname muß auf der linken Seite und das Objekt bzw. die Angaben zur Bestimmung des Objekts müssen auf der rechten Seite des Zuweisungssymbols – in Form einer Message – aufgeführt werden.

Hinweis: Links vom Zuweisungszeichen ":=" dürfen lediglich Variablen und niemals Objekte aufgeführt werden.
Es ist zu beachten, daß zunächst die rechts vom Zuweisungszeichen ":=" stehende Message und erst dann die Zuordnung an die Variable ausgeführt wird.
Da auf der rechten Seite einer Zuweisung nicht nur eine Message, sondern eine beliebige Anforderung stehen darf, ist es möglich, rechts vom Zuweisungszeichen weitere Zuweisungen vorzunehmen. Dadurch lassen sich Mehrfachzuweisungen formulieren.

Dasjenige Objekt, das durch die oben angegebene Zuweisung an die Variable "WerteErfassung11" gebunden wird, bestimmt sich daher aus der Ausführung der folgenden Message:

```
WerteErfassung new
```

Diese Message besteht aus dem Empfänger-Objekt "WerteErfassung" und dem Message-Selektor "new".
Damit "WerteErfassung" überhaupt als Empfänger-Objekt einer Message angesehen werden kann, muß es sich bei "WerteErfassung" um ein Objekt handeln. Daß dies richtig ist, basiert auf dem folgenden Sachverhalt:

- Jede dem SMALLTALK-System bekannte Klasse wird – ebenso wie ihre Instanzen – als *Objekt* angesehen.

  Hinweis: Wir werden zu einem späteren Zeitpunkt kennenlernen, daß jede Klasse gemäß eines Musters aufgebaut ist, das in einer ihr zugehörigen *Meta-Klasse* festgelegt ist (siehe Kapitel 8).

Da (im Normalfall) jeder Klasse die Basis-Methode mit dem Methoden-Selektor "new" bekannt ist, wird durch die Message "new" bewirkt, daß die Klasse "WerteErfassung" die Methode "new" ausführt.

- **"new"**:
  Die Message "new", die einer dem SMALLTALK-System bekannten Klasse als Empfänger-Objekt zugestellt wird, bewirkt (im Normalfall), daß diese Klasse die zugehörige Basis-Methode namens "new" ausführt und dadurch eine Instanz aus der Klasse des Empfänger-Objektes als neues Objekt erzeugt wird.

  Hinweis: Es ist zu beachten, daß Instanzen der Klasse zugeordnet sind, aus der sie instanziiert wurden. Sie können ihre Klasse nicht wechseln.

Die an "WerteErfassung11" gerichtete Message

```
WerteErfassung new
```

bewirkt folglich, daß eine Instanz von "WerteErfassung" eingerichtet wird.
Insgesamt hat die Zuweisung

```
WerteErfassung11 := WerteErfassung new
```

die von uns im Workspace-Fenster eingetragen und dem SMALLTALK-System übergeben wurde, somit die folgende Auswirkung:

- Die globale Variable “WerteErfassung11” wird eingerichtet und weist auf eine Instanz der Klasse “WerteErfassung”. Somit wird in Zukunft jede Message, die an “WerteErfassung11” gerichtet wird, vom SMALLTALK-System an die dieser Variablen zugeordneten Instanz der Klasse “WerteErfassung” gesandt.

Die Pseudovariable “nil”

Um eine Instanz wie z.B. “WerteErfassung11” wieder zu löschen, läßt sich eine spezielle Zuweisung in der Form

```
WerteErfassung11 := nil
```

verwenden.
Diese Zuweisung bewirkt, daß die Zuordnung der Variablen “WerteErfassung11” an eine Instanz der Klasse “WerteErfassung” aufgehoben wird. Die Variable “WerteErfassung11” verweist jetzt auf das hinter dem Zuweisungssymbol aufgeführte Objekt “nil”.

Hinweis: Dabei ist zu beachten, daß die globale Variable “WerteErfassung11” weiterhin existiert. Eine derartige Aufhebung ist z.B. dann erforderlich, wenn Änderungen bei der Vereinbarung der Instanz-Variablen der Klasse “WerteErfassung” vorzunehmen sind. In diesem Fall darf – aus Gründen der Konsistenz – keine Instanz dieser Klasse eingerichtet sein.

- Bei *“nil”* handelt es sich um eine *Pseudovariable*, die – in bestimmten Fällen – anstelle einer Variablen innerhalb einer Message verwendet werden darf.

  Hinweis: Die Pseudovariable “nil” ist die *einzige* Instanz der Basis-Klasse “UndefinedObject”. Im Normalfall werden Variablen und Instanz-Variablen – bei ihrer Einrichtung durch das SMALLTALK-System – an die Pseudovariable “nil” gebunden.

- Unter einer *Pseudovariablen* wird ein durch ein Schlüsselwort gekennzeichnetes Objekt verstanden, das anstelle einer Variablen innerhalb einer Anforderung aufgeführt werden darf und eine fest verabredete Bedeutung besitzt.

  Im Gegensatz zu einer Pseudovariablen ist eine *Variable* kein Objekt. Sie besitzt die alleinige Funktion, daß ihr ein Objekt zugeordnet werden kann, so daß sich dieses Objekt durch den Variablennamen – in Form einer Referenz auf das Objekt – bezeichnen läßt.

3.2 Festlegung und Anzeige des Erfassungsfensters

Bestimmung der Titel-Zeile

Nach der Einrichtung der Instanz “WerteErfassung11” kann jede Methode, die wir bisher innerhalb der Klasse “WerteErfassung” vereinbart haben, von der Instanz “WerteErfassung11” ausgeführt werden.

Generell gilt nämlich der folgende Sachverhalt:

- Um von einer Instanz die Ausführung einer Methode abzurufen, muß eine Message an diese Instanz gerichtet werden, bei der der Message-Selektor mit dem Methoden-Selektor der auszuführenden Methode übereinstimmt.

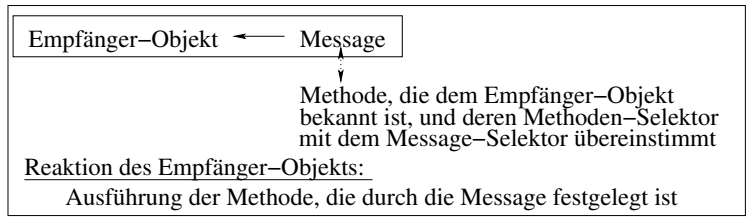


Abbildung 3.4: Wirkung einer Message

Da die Methode "initialisierenErfassung" dazu dient, den strukturellen Aufbau des Erfassungsprozesses sowie das Erfassungsfenster festzulegen, müssen wir – zur Lösung unserer Aufgabenstellung – die Methode "initialisierenErfassung" von der Instanz "WerteErfassung11" ausführen lassen.

Damit die Message mit dem Message-Selektor "initialisierenErfassung" an die Instanz "WerteErfassung11" gesendet wird, tragen wir die Anforderung

```
WerteErfassung11 initialisierenErfassung
```

in das Workspace-Fenster ein und lassen sie vom SMALLTALK-System ausführen.

Da wir die Punktwerte der Jahrgangsstufe 11 erfassen wollen, soll die Überschrift "Jahrgangsstufe 11" in der Titel-Zeile des Erfassungsfensters erscheinen. Daher muß die Methode "festlegenUeberschrift:" von der Instanz "WerteErfassung11" ausgeführt werden. Folglich tragen wir die Anforderung

```
WerteErfassung11 festlegenUeberschrift: 'Jahrgangsstufe 11'
```

in das Workspace-Fenster ein und rufen ihre Ausführung ab.

Keyword-Message und unäre Message

Im Gegensatz zum Message-Selektor "initialisierenErfassung" ist der Message-Selektor "festlegenUeberschrift:" durch einen *Doppelpunkt* abgeschlossen. Dies liegt daran, daß diesem Selektor eine Angabe darüber folgt, welcher Text als Überschrift verwendet werden soll.

- Sofern eine Message ergänzende Angaben enthält, die bei der Ausführung der jeweils korrespondierenden Methode zusätzlich benötigt werden, bezeichnet man sie als *Keyword-Message*. Jede einzelne Angabe wird als *Argument* innerhalb der Keyword-Message aufgeführt und dadurch kenntlich gemacht, daß ihr ein Selektor – als Bestandteil des Message-Selektors – vorausgeht, der durch einen *Doppelpunkt* beendet wird.

Hinweis: Besteht eine Keyword-Message aus mehreren Selektoren, so werden die Selektoren – ohne Trennzeichen – hintereinander aufgeführt, sofern wir den Message-Selektor im Rahmen unserer Darstellung angeben. Folglich kennzeichnen wir z.B. eine Keyword-Message mit den Selektoren "when:" und "perform:" in der Form "when:perform:".

- Eine Message, die durch einen einzigen Selektor – ohne einen abschließenden Doppelpunkt – gekennzeichnet ist, wird als *unäre* Message bezeichnet.

  Hinweis: Derartige Messages dienen häufig zum Prüfen oder für den Zugriff auf die Instanz-Variablen des Empfänger-Objekts.

Während es sich bei den oben angegebenen Messages "new" und "initialisieren-Erfassung" jeweils um eine unäre Message handelt, stellt "festlegenUeberschrift:" eine Keyword-Message dar, mit der sich die wie folgt vereinbarte Methode (siehe Abschnitt 2.5) zur Ausführung abrufen läßt:

```
festlegenUeberschrift: aString
self labelWithoutPrefix: aString
```

Dabei dient "aString" als Platzhalter für das Argument, das innerhalb der Message, mit der diese Methode zur Ausführung abgerufen wird, hinter dem Message-Selektor "festlegenUeberschrift:" aufgeführt ist.
Da innerhalb der Message "festlegenUeberschrift:" das Argument "Jahrgangsstufe 11" aufgeführt wurde, wird es für den Platzhalter "aString" innerhalb der Methode "festlegenUeberschrift:" eingesetzt und die Methode in dieser Form zur Ausführung gebracht.
Die Vereinbarung der Methode "festlegenUeberschrift:" ist ein Beispiel dafür, wie Methoden innerhalb einer Klasse festgelegt werden müssen, deren Ausführung durch eine Keyword-Message abgerufen werden soll.

- Für jedes Argument der Keyword-Message ist der Name eines Platzhalters zu bestimmen, der – zu Beginn der Methoden-Vereinbarung – hinter dem zugehörigen Selektor aufzuführen ist und dem Bildungsgesetz für Instanz-Variablen folgen muß. Innerhalb der Anforderungen, die in der Methode festgelegt werden, dürfen Platzhalter an geeigneten Positionen angegeben werden. Sie werden vor der Ausführung der Methode jeweils durch die mit ihnen korrespondierenden Argumente der Keyword-Message ersetzt.

Hinweis: Bei der Vereinbarung von Methoden ist es sinnvoll, die benötigten Platzhalter durch Namen zu bezeichnen, die mit den Namen derjenigen Klassen korrespondieren, aus denen sie instanziiert sein sollten, wenn sie als Argumente der zugehörigen Keyword-Message aufgeführt sind.
Sofern eine Zeichenkette als Argument verwendet werden soll, ist es daher sinnvoll, die Bezeichnung "aString" für den Platzhalter zu wählen, da jede *Zeichenkette* eine Instanz der Basis-Klasse "String" darstellt (siehe unten).
Setzen wir Platzhalter – wie z.B. "aString" – ein, so spricht man von "typed parameter", da dadurch ein Hinweis auf den Typ des Objekts gegeben wird. Statt "typed parameter" ist es in SMALLTALK auch üblich, sog. "semantic parameter" zu verwenden. Darunter sind Platzhalter zu verstehen, die die Bedeutung des Platzhalters charakterisieren. Im Falle der Methode "festlegenUeberschrift:" könnten wir z.B. als Platzhalter "eineUeberschrift" verwenden.

Literale

Als Argument der Keyword-Message

```
festlegenUeberschrift: 'Jahrgangsstufe 11'
```

ist die Zeichenkette 'Jahrgangsstufe 11' aufgeführt.

- Da es sich bei den *Argumenten* von Keyword-Messages stets um *Objekte* handeln muß, werden *Zeichenketten* ("Strings") wie z.B.

  ```
  'Jahrgangsstufe 11'
  ```

  ebenfalls – genauso wie Instanzen und Klassen – als *Objekte* angesehen. Jede *Zeichenkette*, die durch das Anführungszeichen (') einzuleiten und abzuschließen ist, stellt eine Instanz der Basis-Klasse "String" dar.

  Hinweis: In der Klasse "String" sind grundlegende Basis-Methoden zur Verarbeitung von Zeichenketten – wie z.B. die Ermittlung der Zeichenkettenlänge – enthalten.

- Grundsätzlich werden alle *Literale* (Konstanten), d.h. neben den Zeichenketten z.B. auch einzelne Zeichen (als Instanzen der Basis-Klasse "Character") oder Zahlen (als Instanzen der Basis-Klasse "Integer", "Real" oder "Fraction"), den *Objekten* zugerechnet (siehe Kapitel 5).

Da sämtliche Eigenschaften eines Literals und dessen Zugehörigkeit zur jeweiligen Basis-Klasse bereits durch die Schreibweise des Literals festgelegt sind, wird ein Literal *automatisch* instanziiert, wenn es Bestandteil einer Message ist.
Die Instanziierung eines Literals darf *nicht* dadurch angefordert werden, daß die Message "new" an die jeweilige Basis-Klasse gerichtet wird.

Anzeige des Erfassungsfensters

Durch die Anforderung

```
WerteErfassung11 festlegenUeberschrift: 'Jahrgangsstufe 11'
```

führt die Instanz "WerteErfassung11" die durch die Keyword-Message

```
festlegenUeberschrift: 'Jahrgangsstufe 11'
```

bestimmte Methode "festlegenUeberschrift:" aus, so daß die Titel-Zeile des Erfassungsfensters durch die Überschrift "Jahrgangsstufe 11" festgelegt ist.
Damit das Erfassungsfenster mit dieser Titel-Zeile am Bildschirm angezeigt und mit der Erfassung der Punktwerte der Jahrgangsstufe 11 begonnen werden kann, muß – entsprechend unserem Lösungsplan – die Methode "durchfuehrenErfassung" von "WerteErfassung11" ausgeführt werden. Um diese Ausführung abzurufen, müssen wir die unäre Message mit dem Message-Selektor "durchfuehrenErfassung" in der folgenden Form an die Instanz "WerteErfassung11" richten:

```
WerteErfassung11 durchfuehrenErfassung
```

Tragen wir diese Anforderung in das Workspace-Fenster ein, so besitzt es den folgenden Inhalt:

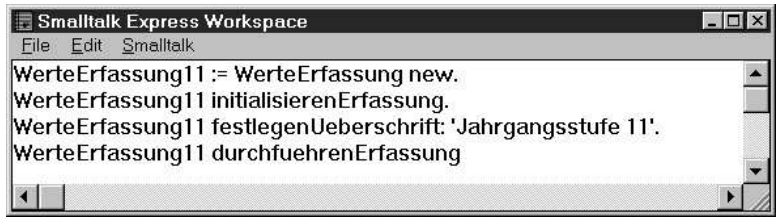


Abbildung 3.5: Anforderungen zur Durchführung der Erfassung

Hinweis: Anstatt die Anforderungen einzeln ausführen zu lassen, können sie auch *insgesamt* zur Ausführung gebracht werden, indem sie – vor der Bestätigung durch die Menü-Option "Do It" des Menüs "Smalltalk" – insgesamt markiert werden.

Wird die zuletzt eingegebene Anforderung ausgeführt, so wird das Erfassungsfenster gemäß Abbildung 3.6 am Bildschirm angezeigt.

Hinweis: Sofern die oben aufgeführten Anforderungen fehlerbehaftet in das Workspace-Fenster eingetragen wurden, erscheint das Walkback-Fenster, das wir mit der Tastenkombination "Alt+F4" schließen (siehe im Anhang A.3).

In dieser Situation läßt sich die Erfassung der Punktwerte – wie im Kapitel 1 beschrieben – schrittweise vornehmen. Nach der Eingabe eines Punktwertes wird seine Erfassung durch die Betätigung des Schaltfeldes "erfasse" durchgeführt. Ist der letzte Punktwert erfaßt worden, so wird der Erfassungsprozeß durch die Betätigung des Schaltfeldes "Erfassungsende" beendet und das Erfassungsfenster vom Bildschirm entfernt.

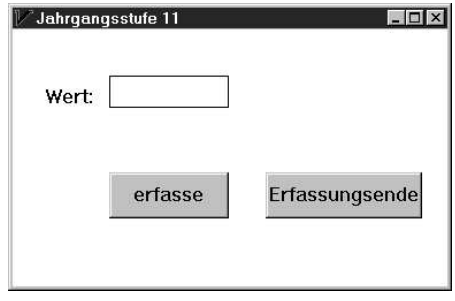


Abbildung 3.6: Beginn des Erfassungsprozesses

Soll eine begonnene Erfassung *weitergeführt* werden, so läßt sich dies über die Anforderung

```
WerteErfassung11 durchfuehrenErfassung
```

abrufen.
In diesem Fall wird die Instanz-Variable "werteBag" von "WerteErfassung11" um die zusätzlich erfaßten Punktwerte ergänzt.

3.3 Instanzen und ihr Erscheinungsbild

Erfassung in einem Fenster

Nachdem wir die Punktwerte der Jahrgangsstufe 11 erfaßt haben, enthält die Instanz-Variable "werteBag" der Instanz "WerteErfassung11" sämtliche über die Tastatur eingegebenen Werte. Im Hinblick auf die beabsichtigte Durchschnittsbildung müssen wir dafür sorgen, daß die gespeicherten Punktwerte konserviert werden und nicht durch eine sich unmittelbar anschließende Erfassung anderer Punktwerte verlorengehen.

Hinweis: Die erfaßten Punktwerte der Jahrgangsstufe 11, die wir innerhalb der Instanz "WerteErfassung11" gesammelt haben, gehen z.B. dann verloren, wenn die Anforderungen

```
WerteErfassung11 := WerteErfassung new
```

oder

```
WerteErfassung11 initialisierenErfassung
```

erneut gestellt werden.

Um die Punktwerte der Jahrgangsstufe 12 zu erfassen, richten wir daher ein *neues* Objekt als weitere Instanz der Klasse "WerteErfassung" ein. Wir erstellen diese Instanz unter dem Namen "WerteErfassung12", indem wir die Instanziierung wie folgt anfordern:

```
WerteErfassung12 := WerteErfassung new
```

Hinweis: Da eine neue globale Variable eingerichtet werden soll, ist die vom SMALLTALK-System gestellte Frage "Define as global variable?" wiederum durch die Bestätigung des Schaltfeldes "Ja" zu beantworten.

Nach der Ausführung der Anforderungen

```
WerteErfassung12 initialisierenErfassung.
WerteErfassung12 durchfuehrenErfassung
```

wird das Erfassungsfenster wie folgt am Bildschirm angezeigt:

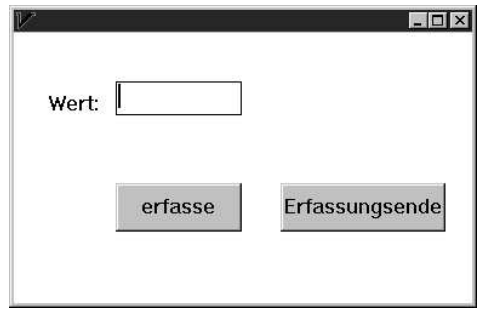


Abbildung 3.7: Erfassungsfenster ohne Überschrift

Hinweis: Da noch keine Message an "WerteErfassung12" gerichtet wurde, durch die eine Überschrift bestimmt wird, enthält die Titel-Zeile des Erfassungsfensters keinen Text.

Genauso wie das zuvor – bei der Erfassung der Punktwerte der Jahrgangsstufe 11 – verwendete Erfassungsfenster das äußere Erscheinungsbild der Instanz "WerteErfassung11" verkörperte, stellt das jetzt am Bildschirm angezeigte Erfassungsfenster das äußere Erscheinungsbild der Instanz "WerteErfassung12" dar.

- Obwohl das Erscheinungsbild der Instanz "WerteErfassung12" im Hinblick auf die Anzeige am Bildschirm – bis auf die fehlende Überschrift des Erfassungsfensters – vollständig mit dem Erscheinungsbild der Instanz "WerteErfassung11" übereinstimmt, haben wir es mit *unterschiedlichen* Instanzen zu tun, was durch die *verschiedenen* Namen der Instanzen dokumentiert wird.

Im Hinblick auf ihre Darstellung am Bildschirm werden durch die beiden Erscheinungsbilder nur bestimmte und nicht sämtliche Eigenschaften der Instanzen "WerteErfassung11" und "WerteErfassung12" gekennzeichnet. Von zentraler Bedeutung ist der Sachverhalt, daß sich die Inhalte der jeweils zugehörigen Instanz-Variablen "werteBag" grundlegend unterscheiden. Während "werteBag" in der Instanz "WerteErfassung11" zu diesem Zeitpunkt (nach der vorausgegangenen Erfassung) bereits sämtliche Punktwerte der Jahrgangsstufe 11 enthält, ist "werteBag" in der Instanz "WerteErfassung12" zum jetzigen Zeitpunkt leer, weil von uns noch keine Punktwerte der Jahrgangsstufe 12 erfaßt wurden.

Änderung des äußeren Erscheinungsbildes

Im Hinblick darauf, daß bei der anstehenden Erfassung Punktwerte der Jahrgangsstufe 12 eingegeben werden sollen, ist es sinnvoll, den Text "Jahrgangsstufe 12" als Überschrift des Erfassungsfensters erscheinen zu lassen.
Damit dies geschieht, müssen wir die Methode "festlegenUeberschrift:" mittels der Keyword-Message

```
festlegenUeberschrift: 'Jahrgangsstufe 12'
```

von "WerteErfassung12" ausführen lassen.

Dazu stellen wir im Workspace-Fenster die folgende Anforderung:

```
WerteErfassung12 festlegenUeberschrift: 'Jahrgangsstufe 12'
```

Nach der Ausführung der Methode "festlegenUeberschrift:" erscheint *unmittelbar* die wie folgt veränderte Anzeige des Erfassungsfensters:

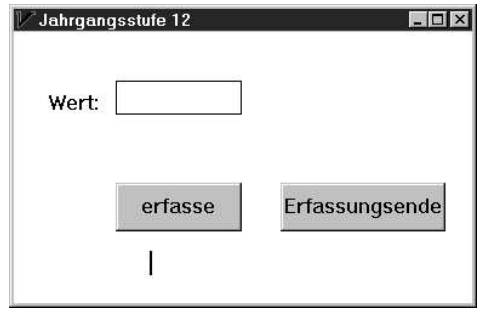


Abbildung 3.8: Erfassungsfenster mit Überschrift

Das Erscheinungbild der Instanz "WerteErfassung12" hat sich dahingehend verändert, daß sich der Inhalt der Titel-Zeile – als charakteristische Eigenschaft der Instanz "WerteErfassung12" – durch die Ausführung der Methode "festlegenUeberschrift:" gewandelt hat.
Die derart bewirkte Änderung des äußeren Erscheinungsbildes ist ein Beispiel für den folgenden allgemein gültigen Sachverhalt:

- Der Zustand von Instanzen läßt sich durch geeignete Messages *dynamisch* verändern. Dazu sind Methoden zur Ausführung zu bringen, mit denen eine Änderung der Attributwerte in den zugehörigen Instanz-Variablen bewirkt wird.
  Um eine derartige Änderung anzufordern, muß die betreffende Instanz-Variable – wie z.B. "werteBag" – als Empfänger-Objekt einer geeigneten Message aufgeführt werden.

In dem durch die Instanz "WerteErfassung12" verkörperten Erfassungsfenster läßt sich die Erfassung der Punktwerte der Jahrgangsstufe 12 genauso durchführen, wie es zuvor für die Punktwerte der Jahrgangsstufe 11 im Hinblick auf die Instanz "WerteErfassung11" geschehen ist.

Erfassung in zwei Fenstern

Neben der soeben beschriebenen Form, in der die Erfassungen für die Jahrgangsstufen 11 und 12 *nacheinander* vorgenommen wurden, ist es auch möglich, die beiden Erfassungsfenster *gleichzeitig* im Zugriff zu haben. Um dies zu erreichen, müssen wir die folgenden Anforderungen im Workspace-Fenster stellen:

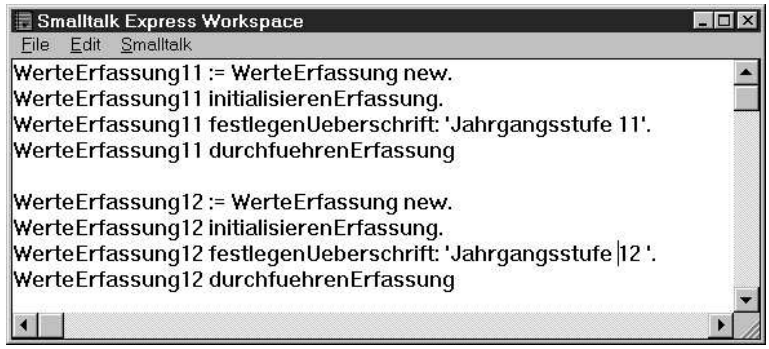


Abbildung 3.9: Anforderung zweier Erfassungsprozesse

Hinweis: Durch die angegebenen Zuweisungen an die globalen Variablen "WerteErfassung11" und "WerteErfassung12" erfolgen neue Instanziierungen. Dadurch sind die zuvor durch diese Variablen gekennzeichneten Instanzen nicht mehr zugänglich, so daß die ursprünglichen Attributwerte ihrer Instanz-Variablen verloren sind.

Nachdem die Anforderungen an das SMALLTALK-System übermittelt sind, erhalten wir die folgende Bildschirmanzeige:

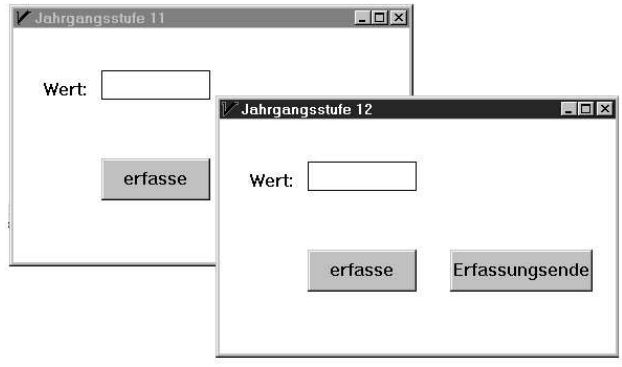


Abbildung 3.10: Zwei Erfassungsfenster

Hinweis: Es ist zu beachten, daß beide Erfassungsfenster zunächst – wegen der zugehörigen Verwaltungs-Behälter mit zunächst identischem Inhalt – an der gleichen Bildschirmposition angezeigt werden. Um die Erfassungsfenster in der angegebenen Form auf dem Bildschirm zu plazieren, ist das zuletzt eröffnete Erfassungsfenster – durch das Ziehen mit der Maus – nach rechts unten verschoben worden.
Um Punktwerte zu erfassen, muß das jeweils benötigte Erfassungsfenster aktiv sein. Dies läßt sich stets über einen Mausklick bewirken.

Hierdurch wird deutlich, daß "WerteErfassung11" und "WerteErfassung12" zwei *verschiedene* Instanzen ein und derselben Klasse sind, die sich äußerlich insofern in ihrem Erscheinungsbild voneinander unterscheiden, als daß unterschiedliche Titel-Zeilen in den Erfassungsfenstern angezeigt werden. In welchem internen Zustand sie

sich jeweils befinden, ist durch die individuellen Punktwerte der jeweiligen Instanz-Variablen “werteBag” – und durch die Werte in den Verwaltungs-Behältern – festgelegt. Dieser Zustand ist jedoch von außen nicht erkennbar und auch nicht einsehbar, da es nicht möglich ist, in ein Objekt unmittelbar Einblick zu nehmen.

3.4 Anzeige von Werten der Instanz-Variablen

Entwicklung der Methode “anzeigenWerte”

Damit wir uns über die jeweils aktuellen Attributwerte von Instanzen unterrichten können, müssen geeignete Methoden in der Klasse vereinbart sein, aus der sie instanziiert wurden.

Hinweis: Eine Möglichkeit besteht darin, den Namen einer Instanz – wie z.B. “WerteErfassung11” – im Workspace-Fenster zu markieren und im Menü “Smalltalk” die Menü-Option “Inspect It” zu bestätigen oder aber “WerteErfassung11” die Message “inspect” in Form der Anforderung

```
WerteErfassung11 inspect
```

zu schicken. Wird anschließend in dem daraufhin angezeigten Inspect-Fenster mit der Titel-Zeile “Smalltalk Express Inspecting: WerteErfassung” z.B. auf den Namen “werteBag” geklickt, so wird die gewünschte Information über den aktuellen Inhalt von “werteBag” angezeigt (siehe Anhang A.4).

Um in unserer Situation Einblick in den Inhalt der Instanz-Variablen “werteBag” nehmen zu können und die bislang erfaßten Punktwerte im Transcript-Fenster anzeigen zu lassen, wollen wir eine Methode namens “anzeigenWerte” entwickeln, bei deren Ausführung der in der Abbildung 3.11 geschilderte Ablauf erfolgen soll.

Diese grafische Darstellung wird *Struktogramm* genannt. Ein Struktogramm besteht aus *Strukturblöcken*, die von oben nach unten ausgeführt werden.

Das angegebene Struktogramm besteht aus den beiden einleitenden “einfachen Strukturblöcken” (1) und (2) und einem abschließenden “Schleifenblock” (3), durch den die wiederholte Ausführung der beiden in seinem Innern eingetragenen einfachen Strukturblöcke (4) und (5) gekennzeichnet wird.

Die “Wiederholungsbedingung”, mit der festgelegt wird, wie oft die Blöcke (4) und (5) zu durchlaufen sind, ist zu Beginn des Schleifenblockes eingetragen.

Im folgenden stellen wir dar, wie die Strukturblöcke umgeformt werden müssen, damit die Methode “anzeigenWerte” unter Einsatz der daraus resultierenden Anforderungen vereinbart werden kann.

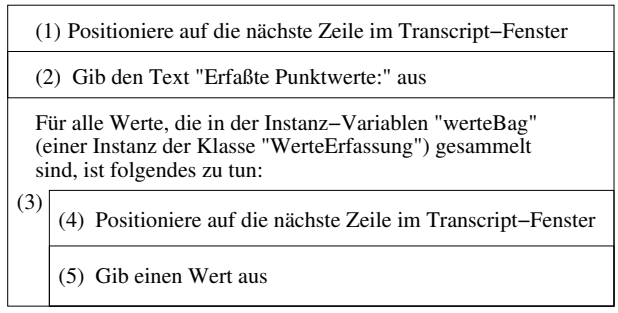


Abbildung 3.11: Struktogramm zur Anzeige im Transcript-Fenster

Die globale Variable “Transcript”

Das Transcript-Fenster, in dem die Anzeige erfolgen soll, ist durch eine globale Variable namens “Transcript” gekennzeichnet. Dieser Variablen ist eine Instanz zugeordnet, die durch eine Instanziierung der Basis-Klasse “MDITranscript” beim Start des SMALLTALK-Systems *automatisch* eingerichtet wird.

Hinweis: Um den Namen der Klasse zu erfragen, aus der “Transcript” instanziiert wurde, läßt sich die Basis-Methode “class” verwenden, indem wir im Workspace-Fenster die Anforderung

```
Transcript class
```

stellen und mittels der Menü-Option “Show It” des Menüs “Smalltalk” ausführen lassen.

Bei der Umformung des Struktogramms beginnen wir mit dem Strukturblock (1). Um auf die nächste Zeile des Transcript-Fensters zu positionieren, muß “Transcript” die Message “cr” zugestellt werden, so daß wir

```
Transcript cr
```

als erste Anforderung innerhalb der Methode “anzeigenWerte” festlegen.

- **“cr”**:
  Bei “cr” handelt es sich um eine Basis-Methode, die dem Objekt “Transcript” bekannt ist.
  Wird “Transcript” die unäre Message mit dem Message-Selektor “cr” gesandt, so ändert “Transcript” den Attributwert eines seiner Attribute, die durch die Basis-Klasse bestimmt sind, aus der “Transcript” instanziiert wurde. Dadurch ändert sich das äußere Erscheinungsbild von “Transcript”, indem der Cursor auf den Anfang der nächsten Zeile des Transcript-Fensters positioniert wird.

Den Strukturblock (2) formen wir in die Anforderung

```
Transcript show: 'Erfaßte Punktwerte:'
```

um, indem “Transcript” die Keyword-Message “show:” – mit einem Argument in Form der Zeichenkette 'Erfaßte Punktwerte:' – zugestellt wird.

- **“show:”**:
  Bei “show:” handelt es sich ebenfalls um eine Basis-Methode, die dem Objekt “Transcript” bekannt ist.
  Wird “Transcript” die Message “show:” mit dem Argument ’Erfaßte Punktwerte:’ gesandt, so ändert sich das äußere Erscheinungsbild des Transcript-Fensters, indem der Text “Erfaßte Punktwerte:” mit Beginn der aktuellen Cursor-Position im Transcript-Fenster ausgegeben wird.

Sammler

Um in dem oben angegebenen Struktogramm den Schleifenblock (3) umzuformen, verwenden wir die folgende Anforderung:

```
werteBag do:[:einObjekt|Transcript cr.Transcript show:einObjekt]
```

Hierbei wird eine Message mit dem Message-Selektor *“do:”* an ein Empfänger-Objekt namens “werteBag” gerichtet.

Damit diese Message sinnvoll ist, muß es sich bei dem Empfänger-Objekt um eine *Sammlung* von Objekten handeln.

- Als *Sammler* werden Instanzen von Basis-Klassen bezeichnet, die sich zur Sammlung von Objekten verwenden lassen.

Als Beispiel für einen Sammler haben wir den “Bag” als Instanz der Basis-Klasse “Bag” kennengelernt.

- Ein *Bag* stellt die allgemeinste Form eines Sammlers dar.
  Bei einem derartigen Sammler können gleiche Objekte mehrfach auftreten. Allerdings ist zu berücksichtigen, daß es keine direkte Zugriffsmöglichkeit auf einzelne gesammelte Objekte gibt, so daß z.B. keine Message gezielt an ein erstes oder ein letztes Objekt eines Bags gerichtet werden kann.

Hinweis: Neben einem Bag gibt es eine Vielzahl anderer Sammler wie z.B. ein “Set” (eine Menge) und einen “Array” (ein indizierter Sammler, bei dem über einen ganzzahligen Index auf jedes der gesammelten Objekte zugegriffen werden kann). Derartige Sammler werden wir im Kapitel 9 vorstellen.

Das Objekt “Block”

In der oben angegebenen Anforderung

```
werteBag do:[:einObjekt|Transcript cr.Transcript show:einObjekt]
```

wird als Argument des Message-Selektors “do:” ein *Objekt* verwendet, das die folgende Form besitzt:

```
[:einObjekt | Transcript cr. Transcript show: einObjekt]
```

Dieses Objekt, das durch die eckige Klammer “[” eingeleitet und durch die eckige Klammer “]” beendet wird, bezeichnet man als *Block*.

- In einem *Block* sind eine oder mehrere *Block-Anforderungen* enthalten, die paarweise durch einen Punkt zu trennen sind. Sämtliche Block-Anforderungen werden in derjenigen Reihenfolge ausgeführt, in der sie innerhalb des Blockes eingetragen sind.
  Sofern innerhalb einer Block-Anforderung *Platzhalter* – wie etwa in unserem Beispiel "einObjekt" – als Stellvertreter für Argumente von Messages benötigt werden, die erst zum Zeitpunkt der Ausführung des Blockes konkret vorliegen, müssen sie als *Block-Parameter* zu Beginn des Blockes vereinbart sein. Der jeweils für einen Block-Parameter verwendete Name, der die Regeln für die Namensvergabe von Instanz-Variablen erfüllen muß, ist durch einen Doppelpunkt ":" einzuleiten und von der 1. Block-Anforderung durch den senkrechten Strich "|" abzugrenzen.

  Hinweis: Nähere Angaben zur Ausführung von Blöcken machen wir im Abschnitt 6.2.

Die Basis-Methode "do:"

In der obigen Anforderung

```
werteBag do:[:einObjekt|Transcript cr.Transcript show:einObjekt]
```

wird die Keyword-Message "do:" verwendet. Hierdurch wird bewirkt, daß das Empfänger-Objekt die Basis-Methode "do:" zur Ausführung bringt.
Bei der Ausführung dieser Methode werden die Block-Anforderungen desjenigen Blockes, der als Argument hinter dem Message-Selektor "do:" aufgeführt ist, wiederholt bearbeitet. Bei jedem einzelnen Ausführungs-Schritt steht der Block-Parameter wie z.B. "einObjekt" – als Platzhalter – stellvertretend für jedes einzelne Objekt, das Bestandteil des Empfänger-Objektes ist. Die Ausführung der Methode "do:" ist beendet, wenn die Block-Anforderungen für alle gesammelten Objekte bearbeitet wurden.
Durch den Aufbau der Anforderung

```
werteBag do:[:einObjekt|Transcript cr.Transcript show:einObjekt]
```

ist erkennbar, daß der Strukturblock (4) durch die Block-Anforderung

```
Transcript cr
```

und der Strukturblock (5) durch die Block-Anforderung

```
Transcript show: einObjekt
```

umgesetzt worden sind.
Welche Punktwerte bei der Ausführung von

```
werteBag do:[:einObjekt|Transcript cr.Transcript show:einObjekt]
```

im Transcript-Fenster angezeigt werden, richtet sich danach, von welcher Instanz der Klasse "WerteErfassung" die Methode "anzeigenWerte" ausgeführt wird.
Grundsätzlich gilt nämlich:

- Führt eine Instanz eine Methode aus, in deren Anforderungen Variablennamen von Instanz-Variablen enthalten sind, so handelt es sich um Instanz-Variablen *dieser* Instanz.
  Soll eine Instanz durch die Ausführung einer Methode auf eine ihrer Instanz-Variablen zugreifen, so reicht es aus, daß in der Methode der Name dieser Instanz-Variablen als Empfänger-Objekt (oder auch als Argument einer Keyword-Message) verwendet wird.

Ist die angegebene Anforderung z.B. in einer Methode enthalten, die von der Instanz "WerteErfassung11" ausgeführt wird, so werden die Block-Anforderungen zunächst für einen ersten Wert, der in der Instanz-Variablen "werteBag" von "WerteErfassung11" enthalten ist, bearbeitet, anschließend für einen weiteren Wert usw. – bis hin zum letzten Wert.

Die Methode "anzeigenWerte"

Unter Einsatz der oben vorgestellten Basis-Methoden und der mit ihrer Hilfe angegebenen Anforderungen können wir die Methode "anzeigenWerte", durch deren Ausführung die Anzeige der erfaßten Punktwerte erfolgen soll, insgesamt wie folgt vereinbaren:

```
anzeigenWerte
Transcript cr.
Transcript show: 'Erfaßte Punktwerte:'.
werteBag do:[:einObjekt|Transcript cr.Transcript show: einObjekt]
```

Hinweis: Wie im Abschnitt 2.5 beschrieben, ist bei der Vereinbarung von "anzeigenWerte" wie folgt vorzugehen:
Durch die Menü-Option "Browse Classes..." des Menüs "File" ist zunächst der Klassen-Hierarchie-Browser zu aktivieren, so daß mittels der Menü-Option "Find Class..." des Menüs "Classes" das Dialogfeld "Smalltalk Express Prompter" abgerufen werden kann. Nachdem der Name "WerteErfassung" eingetragen und durch das Schaltfeld "OK" bestätigt worden ist, werden die bislang vereinbarten Methoden im Methoden-Bereich des Klassen-Hierarchie-Browser-Fensters angezeigt. Um "anzeigenWerte" als *neue* Methode zu verabreden, ist die Menü-Option "New Method" des Menüs "Methods" zu bestätigen und das daraufhin im Editier-Bereich angezeigte Muster einer Methoden-Vereinbarung zu löschen. Nachdem der Methoden-Selektor "anzeigenWerte" zusammen mit den drei zugehörigen Anforderungen im Editier-Bereich eingetragen ist, läßt sich die Methode "anzeigenWerte" dem SMALLTALK-System mittels der Menü-Option "Save" des Menüs "File" bekanntmachen.

Nach der Vereinbarung der Methode "anzeigenWerte" besitzt die Klasse "WerteErfassung" die folgende Form:

WerteErfassung	
Instanz–Variablen:	werteBag
Methoden:	initialisierenErfassung erfassenWert: entfernenErfassungsfenster: festlegenUeberschrift: durchfuehrenErfassung anzeigenWerte

Abbildung 3.12: Aktuelle Form der Klasse “WerteErfassung”

Haben wir z.B. die Punktwerte “31”, “35” und “37” für die Jahrgangsstufe 11 und den Punktwert “37” für die Jahrgangsstufe 12 erfaßt, so wird das Transcript-Fenster in der Form

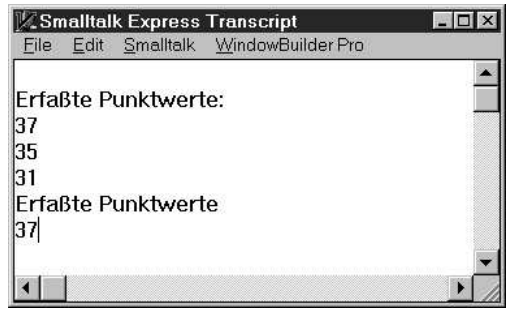


Abbildung 3.13: Anzeige der erfaßten Punktwerte

angezeigt, sofern die beiden folgenden Anforderungen im Workspace-Fenster zur Ausführung gebracht worden sind:

```
WerteErfassung11 anzeigenWerte.
WerteErfassung12 anzeigenWerte
```

Hinweis: Es ist zu beachten, daß die Reihenfolge der im Transcript-Fenster angezeigten Werte einer Jahrgangsstufe nicht mit der Reihenfolge übereinstimmen muß, in der die Punktwerte erfaßt wurden. Dies liegt daran, daß die Sammlung in einen “Bag” erfolgt ist, so daß keine Aussagen über die interne Ablage der gespeicherten Werte gemacht werden kann (siehe Kapitel 9).

Kapitel 4

Erweiterung des Lösungsplans

4.1 Anzahl der Punktwerte

Ergebnis-Objekte

Bislang haben wir erläutert, wie sich die Klasse "WerteErfassung" vereinbaren und – zur Durchführung von Erfassungsprozessen – die Instanziierung und das Senden von Messages als Anforderungen an das SMALLTALK-System abrufen lassen. Mit diesen Kenntnissen wenden wir uns jetzt der im Abschnitt 1.2 angegebenen Problemstellung PROB-1-2 zu, durch die wir uns die folgende Aufgabe gestellt haben:

- PROB-1-2:
  Es sind die jahrgangsstufen-spezifischen Durchschnittswerte für alle erfaßten Punktwerte zu berechnen und am Bildschirm anzuzeigen!

Für das Folgende setzen wir voraus, daß wir bereits Punktwerte der Jahrgangsstufen 11 und 12 erfaßt haben, so daß diese Werte in den beiden Instanz-Variablen namens "werteBag" der Objekte "WerteErfassung11" und "WerteErfassung12" gespeichert sind.
Zur Lösung von PROB-1-2 wollen wir zunächst eine Methode entwickeln, durch die sich die erfaßten Punktwerte zur weiteren Bearbeitung bereitstellen lassen. Im Hinblick auf diese Absicht ist der folgende Sachverhalt bedeutsam:

- Über eine Message läßt sich nicht nur der Zustand des zugehörigen Empfänger-Objektes ändern, sondern jede Message liefert grundsätzlich ein *Ergebnis-Objekt*.

Um was für ein Objekt es sich bei dem jeweiligen Ergebnis-Objekt einer Message handelt, wird durch diejenige Methode bestimmt, die durch die Message zur Ausführung gelangt.

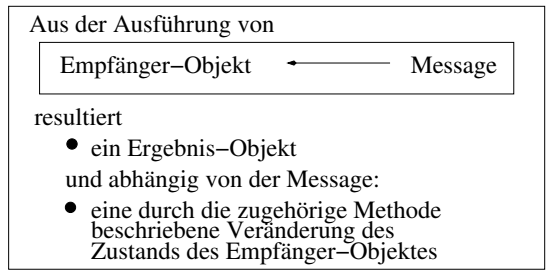


Abbildung 4.1: Ergebnis einer Message

Um Objekte in einen Sammler einzufügen, steht die Basis-Methode "add:" zur Verfügung.

- **"add:"**:
  Bei der Ausführung der Basis-Methode "add:" wird der Sammler, der als Empfänger-Objekt von "add:" aufgeführt ist, um das als Argument der Message "add:" angegebene Objekt ergänzt.
  Als Ergebnis-Objekt der Message "add:" resultiert das dem Sammler hinzugefügte Objekt.

Hieraus folgt, daß bei der Zustellung der Message

```
werteBag add: 37
```

der Bag "werteBag" die Methode "add:" ausführt. Folglich wird der aktuelle Inhalt des Bags um den Wert "37" ergänzt und dieser hinzugefügte Wert – und nicht der um den Wert ergänzte Sammler selbst – als resultierendes Ergebnis-Objekt erhalten.

- Normalerweise ist das Ergebnis-Objekt dadurch festgelegt, daß es dem *Empfänger-Objekt* der Message *gleichgesetzt* ist. Dies ist stets dann der Fall, wenn innerhalb der jeweiligen Methode keine andere Verabredung getroffen wird.

  Soll *nicht* das Empfänger-Objekt *selbst*, sondern ein anderes Objekt – wie z.B. der Inhalt einer Instanz-Variablen – als Ergebnis-Objekt resultieren, so ist dieses Objekt durch die (dynamisch) *letzte* Anforderung der mit der Message korrespondierenden Methode festzulegen, indem diese Anforderung durch das *Return-Zeichen* "∧" (Caret-Zeichen) eingeleitet wird.

Wird einer Anforderung innerhalb einer Methode das *Return-Zeichen* "∧" vorangestellt, so wird diese Methode mit der Ausführung dieser Anforderung beendet. Dasjenige Ergebnis-Objekt, das aus der Bearbeitung dieser Anforderung resultiert, wird zum Ergebnis-Objekt der Message, durch die diese Methode zur Ausführung angefordert wurde.

- Damit das Ergebnis-Objekt einer Anforderung vom SMALLTALK-System im Workspace-Fenster angezeigt wird, setzen wir – statt der Menü-Option "Do It" – die Menü-Option "Show It" des Menüs "Smalltalk" ein.

Hinweis: Das Ergebnis-Objekt wird in markierter Form direkt hinter der ausgeführten Anforderung angezeigt, so daß es sich unmittelbar wieder löschen läßt.
Enthält die Methode, die mit der zuletzt ausgeführten Message korrespondiert, keine Anforderung mit dem Return-Zeichen "∧", so können wir aus der Anzeige erkennen, aus welcher Klasse das Empfänger-Objekt der zuletzt ausgeführten Message instanziiert wurde. Dazu wird (im Normalfall) der Name der zugehörigen Klasse – mit dem einleitenden Artikel "a" bzw. "an" – ausgegeben.

Die primäre Message

Um Ergebnis-Objekte von Messages festzulegen, können – in besonderen Situationen – primäre Messages verwendet werden.

- Eine *primäre* Message besteht entweder aus einem *Variablennamen* oder aus einem *Literal*, so daß die letzte zu bearbeitende Anforderung einer Methode von der Form

  ∧ variablenName

  bzw. von der Form

  ∧ literal

  sein kann.

  Hinweis: Genaugenommen zählen auch "Blöcke" (siehe Abschnitt 3.4) zu den primären Messages.

- Aus der Zustellung der primären Message "variablenName" resultiert – durch bloßes Hinschreiben von "variablenName" – der Inhalt der Variablen als Ergebnis-Objekt. Entsprechend ergibt sich aus der Ausführung der primären Message "literal" das aufgeführte Literal als Ergebnis-Objekt.

Um den Inhalt der Instanz-Variablen "werteBag" zur weiteren Bearbeitung – als Ergebnis-Objekt – bereitzustellen, vereinbaren wir innerhalb von "WerteErfassung" die Methode "bereitstellenWerte" in der folgenden Form:

```
bereitstellenWerte
∧  werteBag
```

Hinweis: Innerhalb des SMALLTALK-Systems sind oft Methoden der folgenden Form vereinbart:

```
<name>
∧ <name>
```

Durch den Einsatz einer derartigen Methode wird durch den Selektor "<name>" auf die gleichnamige Instanz-Variable "<name>" zugegriffen.

Bei der Anforderung

```
werteBag
```

handelt es sich um eine primäre Message, so daß der Inhalt von “werteBag” das Ergebnis-Objekt dieser primären Message und – wegen des einleitenden Return-Zeichens “∧” – auch das Ergebnis-Objekt der Message “bereitstellenWerte” darstellt.

Nachdem wir die Methode “bereitstellenWerte” innerhalb der Klasse “WerteErfassung” vereinbart haben, können wir von der folgenden Situation ausgehen:

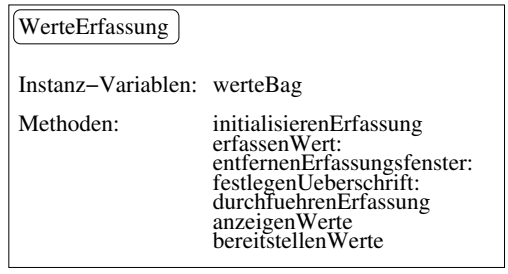


Abbildung 4.2: Aktuelle Form der Klasse “WerteErfassung”

Auf dieser Basis können wir die Message “bereitstellenWerte” mittels der Anforderung

```
WerteErfassung11 bereitstellenWerte
```

an das Empfänger-Objekt “WerteErfassung11” richten. Als *Ergebnis-Objekt* resultiert der Inhalt der Instanz-Variablen “werteBag”, der Bestandteil der Instanz “WerteErfassung11” ist, so daß sich z.B. ein *Bag* mit den Werten ’31’, ’35’ und ’37’ ergibt, sofern diese drei Werte zuvor erfaßt wurden.

Hinweis: Tragen wir diese Anforderung im Workspace-Fenster ein und bringen sie durch die Menü-Option “Show It” des Menüs “Smalltalk” zur Ausführung, so wird z.B. “Bag(’31’ ’35’ ’37’)” im Workspace-Fenster angezeigt.

Verschachtelung von Messages

Das Ergebnis-Objekt, das aus der Ausführung der Message

```
WerteErfassung11 bereitstellenWerte
```

resultiert, läßt sich für eine weitere Bearbeitung bereitstellen.

- Grundsätzlich gibt es die beiden folgenden Möglichkeiten, um das Ergebnis-Objekt einer Message weiterzuverarbeiten:
  - Das Ergebnis-Objekt einer Message dient als *Empfänger-Objekt* einer anderen Message.
  - Das Ergebnis-Objekt einer Message wird als *Argument* einer anderen Message verwendet.

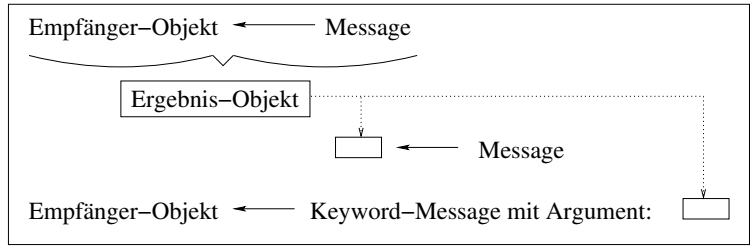


Abbildung 4.3: Verwendung des Ergebnis-Objektes

Um die angegebenen Möglichkeiten nutzen zu können, sind Anforderungen durch eine *Verschachtelung* von Messages zusammenzusetzen. Dabei ist das Ergebnis einer Verschachtelung dasjenige Ergebnis-Objekt, das aus der zuletzt ausgeführten Message resultiert.

Hinweis: Um bei verschachtelten Messages zu kontrollieren, ob nach der Ausführung einer Methode die als nächstes auszuführende Methode auch das gewünschte Empfänger-Objekt erhält, läßt sich sukzessiv die Menü-Option "Show It" des Menüs "Smalltalk" verwenden.

Als Beispiel für den Sachverhalt, daß das Ergebnis-Objekt einer Message als Empfänger-Objekt einer weiteren Message dient, können wir z.B. die Messages mit den Message-Selektoren "bereitstellenWerte" und "size" wie folgt verschachteln:

```
WerteErfassung11 bereitstellenWerte size
```

Bei dieser Anforderung werden die beiden unären Messages "bereitstellenWerte" und "size" nacheinander zugestellt. Dabei wird das Ergebnis-Objekt der Message "bereitstellenWerte" zum Empfänger-Objekt der Message "size".

- **"size"**:
  Falls es sich beim Empfänger-Objekt der Message "size" um eine Sammlung von Objekten – wie z.B. einen *Bag* – handelt, wird bei der Ausführung der zugehörigen Basis-Methode "size" die *Anzahl* der Objekte ermittelt, die im Empfänger-Objekt gesammelt sind. Diese Anzahl ist als Ergebnis-Objekt der Message "size" festgelegt.

Der Sachverhalt, daß bei der Anforderung

```
WerteErfassung11 bereitstellenWerte size
```

zuerst die Message "bereitstellenWerte" und anschließend die Message mit dem Message-Selektor "size" zugestellt wird, beruht auf der folgenden Regel über die Auswertungsreihenfolge:

- Mehrere unäre Messages, deren Message-Selektoren innerhalb von verschachtelten Messages unmittelbar aufeinanderfolgen, werden stets von "links nach rechts" bearbeitet.

Wie oben angegeben, stellt der Bag mit den für die Jahrgangsstufe 11 erfaßten Punktwerten das Ergebnis-Objekt von

```
WerteErfassung11 bereitstellenWerte
```

dar. Wird die unäre Message "size" anschließend an dieses Objekt gerichtet, so resultiert als Ergebnis-Objekt die Anzahl der für die Jahrgangsstufe 11 erfaßten Punktwerte. Entsprechend läßt sich die Anzahl der für die Jahrgangsstufe 12 erfaßten Punktwerte durch die Ausführung von

```
WerteErfassung12 bereitstellenWerte size
```

als resultierendes Ergebnis-Objekt ermitteln.

4.2 Summation der Punktwerte

Entwurf eines Lösungsplans

Nachdem wir erörtert haben, wie wir die Anzahl der erfaßten Punktwerte bestimmen können, wollen wir jetzt eine Möglichkeit kennenlernen, wie sich die Summe der gespeicherten Punktwerte ermitteln läßt. Um die Summation über alle Punktwerte durchzuführen, die in "werteBag" gesammelt sind, können wir z.B. den folgenden Plan entwickeln:

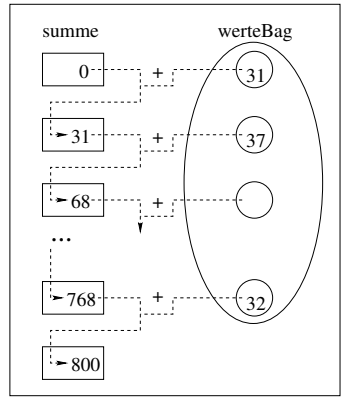


Abbildung 4.4: Summation der Punktwerte

Wir benötigen somit eine Variable ("summe"), der die Zahl "0" als Anfangswert zugeordnet ist und in der das jeweilige Zwischenergebnis der Summation gespeichert werden kann.
Im Hinblick auf die Verarbeitung der in "werteBag" gesammelten Objekte ist der folgende Sachverhalt von Bedeutung:

- Sämtliche bei der Erfassung über die Tastatur in das Eingabefeld eingetragenen Daten werden vom SMALLTALK-System grundsätzlich als *Zeichenketten* entgegengenommen. Daher besteht der Inhalt von "werteBag" nicht aus Zahlen, sondern aus Zeichenketten, die zwar die jeweiligen ganzen Zahlen textmäßig wiedergeben, mit denen jedoch nicht gerechnet werden kann.

Aus diesem Grund muß jeder Zeichenketten-Wert von “werteBag” – vor der Summation – in eine ganze Zahl gewandelt werden.
Den angegebenen Lösungsplan für die Summation können wir daher durch das folgende Struktogramm beschreiben:

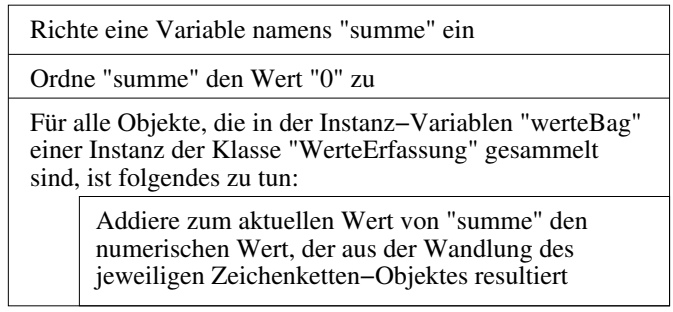


Abbildung 4.5: Struktogramm zur Summation der Punktwerte

Temporäre Variablen

Um eine Variable zur Verfügung zu haben, die allein für die Summation benötigt wird und nach der Berechnung des Summenwertes entbehrlich ist, ist folgender Sachverhalt zu beachten:

- Zur Ausführung von Anforderungen können eine oder mehrere *temporäre* Variablen vereinbart werden, die allein für die Dauer der Ausführung zur Verfügung stehen.
  Die Namen von temporären Variablen, die nach denselben Regeln wie die Namen von Instanz-Variablen aufgebaut sein müssen, sind *vor* der ersten Anforderung anzugeben. Zur Abgrenzung ist die Gesamtheit dieser Namen durch einen senkrechten Strich “|” einzuleiten und abzuschließen.

Der von uns benötigten temporären Variablen, die das jeweilige Zwischenergebnis der Summation aufnehmen soll, geben wir den Namen “summe”, so daß wir vor der ersten Anforderung die folgende Angabe machen müssen:

```
|summe|
```

Die Zuordnung der Zahl “0” als Anfangswert läßt sich durch die folgende Zuweisung festlegen:

```
summe := 0
```

Innerhalb dieser Zuweisung kennzeichnet “summe” dasjenige Objekt, das sich durch die Bearbeitung der primären Message “0” ergibt. Da aus der Ausführung dieser Message der Wert “0” als Ergebnis-Objekt resultiert, wird die Zahl “0” an die temporäre Variable “summe” gebunden.

Arithmetische Operationen

Das oben angegebene Struktogramm schreibt vor, daß zunächst ein erster Wert von "werteBag" zu dem Wert zu addieren ist, der an die Variable "summe" gebunden ist, und das Ergebnis dieser Addition wieder durch "summe" referenziert werden soll. Im nächsten Schritt ist ein weiterer Wert von "werteBag" zum aktuellen Wert von "summe" zu addieren und der resultierende Wert als neuer aktueller Wert an die Variable "summe" zu binden. Diese Verarbeitung ist in der angegebenen Form schrittweise solange durchzuführen, bis der letzte in "werteBag" enthaltene Wert zu dem Wert hinzuaddiert wurde, der an "summe" gebunden ist.

- Um eine *Summation* innerhalb einer Anforderung festzulegen, muß eine Message mit dem Message-Selektor "+" formuliert werden, indem als Empfänger-Objekt der erste Summand (vor dem Pluszeichen) und als Argument der Message der zweite Summand (hinter dem Pluszeichen) aufzuführen ist.

So wird z.B. durch die Anforderung

```
31 + 37
```

dem Objekt "31" eine Message mit dem Argument "37" gesandt, so daß "68" als Summenwert ermittelt und zum Ergebnis-Objekt dieser Message wird.
Bei einer Anforderung der Form

```
summe := summe + 37
```

richtet sich die Message "+" an das Empfänger-Objekt "summe", so daß die Zahl "37" zu dem Wert hinzuaddiert wird, der an "summe" gebunden ist. Das Ergebnis wird mittels des Zuweisungssymbols ":=" der Variablen "summe" als neuer Wert zugeordnet.

- Bei den anderen arithmetischen Grundoperationen *Subtraktion*, *Multiplikation* und *Division* wird genauso verfahren, wie wir es soeben für die Vorgehensweise bei der Summation geschildert haben.
  Als zugehörige Message-Selektoren werden die Rechen-Symbole "−", "∗" und "/" verwendet.
  Der jeweils erste Operand wird als Empfänger-Objekt und der jeweils zweite Operand als Argument hinter dem betreffenden Selektor aufgeführt.
  Um zu unterstreichen, daß es sich bei dieser Art von Messages um *besondere* Keyword-Messages handelt, wird von *binären* Messages gesprochen.

  Hinweis: Selektoren derartiger Messages werden nur dann erkannt, wenn sie aus höchstens zwei Zeichen bestehen, die aus einer festgelegten Menge spezieller Zeichen stammen müssen.

- Verschachtelte binäre Messages werden stets von "links nach rechts" ausgeführt, so daß die Regel "Punktrechnung geht vor Strichrechnung" *nicht* gilt.

  Hinweis: Bei der Ausführung einer Anforderung der Form "`!37 + 32 / 2!'`' wird zunächst die Addition und erst dann die Division ausgeführt.

Wandlung einer Zeichenkette

Zur Umsetzung des Schleifenblockes, der in Abbildung 4.5 im Struktogramm enthalten ist, verwenden wir die Message "do:". Um die im Empfänger-Objekt von "do:" gesammelten Werte zu summieren, könnte der Block

```
[:einObjekt|summe := summe + einObjekt]
```

als Argument der Message "do:" verwendet werden. Dabei ist zu beachten, daß die Block-Anforderung

```
summe := summe + einObjekt
```

allerdings nur dann sinnvoll ist, wenn der Block-Parameter "einObjekt" als Platzhalter für Zahlen dient.
Da es sich jedoch bei den in "werteBag" gesammelten Objekten um Zeichenketten handelt, ist jede Zeichenkette – vor der Summation – in eine ihrem numerischen Wert entsprechend zugeordnete ganze Zahl zu wandeln.

- **"asInteger"**:
  Um Zeichenketten, die nur aus Ziffern-Zeichen bestehen, in eine ganze Zahl zu wandeln, steht die Basis-Methode "asInteger" zur Verfügung. Als Ergebnis-Objekt der unären Message "asInteger" ist diejenige Zahl festgelegt, die aus dieser Wandlung resultiert.

Um die von uns gewünschte Summation durchführen zu lassen, ist daher der Block

```
[:einObjekt|summe := summe + einObjekt asInteger]
```

einzusetzen.

Prioritäten bei verschachtelten Messages

Die korrekte Bearbeitung der Block-Anforderung

```
summe := summe + einObjekt asInteger
```

ist aus folgendem Grund gesichert:

- Sofern binäre und unäre Messages verschachtelt sind, wird eine Ausführung von "links nach rechts" vorgenommen, wobei unäre Messages innerhalb der Auswertungsreihenfolge eine höhere Priorität als binäre Messages besitzen.

Im Hinblick auf die Priorität, nach der verschachtelte unäre Messages ausgeführt werden, haben wir im Abschnitt 4.1 die folgende Regel kennengelernt:

- Mehrere aufeinanderfolgende unäre Messages werden stets von "links nach rechts" bearbeitet.

In diesem Zusammenhang sind auch die folgenden Regeln von grundsätzlicher Bedeutung:

- Bei der Verschachtelung einer unären Message und einer Keyword-Message hat die unäre Message die jeweils höhere Priorität – unabhängig davon, in welcher Reihenfolge beide Messages in einer Anforderung aufgeführt sind.

- Ebenso haben binäre Messages bei einer Verschachtelung eine jeweils höhere Priorität als Keyword-Messages.

Diese Angaben lassen sich tabellarisch insgesamt in der Form

1. unäre Messages
2. binäre Messages
3. Keyword-Messages

zusammenstellen. Dabei kennzeichnet "1." die höchste und "3." die niedrigste Priorität für die Bearbeitung von verschachtelten Messages.

Bei der Ausführung verschachtelter Messages werden die Messages von "links nach rechts" unter Beachtung der Prioritätsstufen verglichen. Dabei werden Messages mit niedriger Prioritätsstufe zuerst ausgeführt. Treten im Hinblick auf die tabellarisch aufgeführten Messages mehrere Messages gleicher Prioritätsstufe hintereinander auf, so werden sie von "links nach rechts" ausgeführt.

Beim Einsatz von Keyword-Messages ist zu beachten, daß diese geklammert werden müssen, wenn sie ein Empfänger-Objekt oder ein Argument bestimmen. Wollen wir verhindern, daß eine Folge von Keyword-Messages als eine einzige Keyword-Message mit mehreren Message-Selektoren aufgefaßt wird, so müssen wir ebenfalls Klammern einsetzen.

Hinweis: Nach der Ausführung der Anforderung

```
WerteErfassung11 initialisierenErfassung
```

können wir einen Erfassungsprozeß z.B. durch die verschachtelten Messages

```
(WerteErfassung11 festlegenUeberschrift: 'Jahrgangsstufe: 11')
   durchfuehrenErfassung
```

einleiten.

Umsetzung und Test des Lösungsplans

Bevor ein Lösungsplan in eine Methoden-Vereinbarung umgesetzt und die resultierende Methode in einer Klasse vereinbart wird, empfiehlt es sich, die als Lösung entwickelten Anforderungen im Workspace-Fenster zu testen.

Da die Punktwerte der Jahrgangsstufe 11, die im Bag "werteBag" der Instanz "WerteErfassung11" gesammelt sind, summiert werden sollen, muß das Empfänger-Objekt der Message "do:" durch

```
WerteErfassung11 bereitstellenWerte
```

festgelegt werden. Die gewünschte Summation kann somit insgesamt – durch die Verschachtelung der unären Message "bereitstellenWerte" und der Keyword-Message "do:" – wie folgt angefordert werden:

```
WerteErfassung11 bereitstellenWerte
       do: [:einObjekt|summe := summe + einObjekt asInteger]
```

Sofern der durch das Struktogramm in Abbildung 4.5 beschriebene Lösungsplan für das Objekt "WerteErfassung11" ausgeführt und zusätzlich das Ergebnis der Summation angezeigt werden soll, können wir im Workspace-Fenster die folgenden Anforderungen stellen:

```
|summe|
summe := 0.
WerteErfassung11 bereitstellenWerte
       do: [:einObjekt|summe := summe + einObjekt asInteger].
summe
```

Rufen wir anschließend deren Ausführung durch die Menü-Option "Show It" des Menüs "Smalltalk" ab, so erhalten wir daraufhin im Workspace-Fenster als Ergebnis-Objekt der zuletzt ausgeführten Anforderung den Wert angezeigt, der an "summe" gebunden ist.

Hinweis: Soll der ermittelte Summenwert zusätzlich im Transcript-Fenster – ab der aktuellen Cursor-Position – angezeigt werden, so ist die Anforderung
`Transcript show: summe printString`
innerhalb der angegebenen Anforderungen *vor* der primären Message "summe" einzutragen.

Bei der Ausführung erhalten wir in dem Fall, in dem z.B. die Werte "31" "35" und "37" erfaßt worden sind, die folgende Anzeige im Workspace-Fenster:

Abbildung 4.6: Test im Workspace-Fenster

Hinweis: Der im Workspace-Fenster enthaltene Ergebniswert "103" ist markiert, so daß er unmittelbar anschließend wieder gelöscht werden kann.
Es ist zu beachten, daß die Anforderung

`summe`

bzw. die Anforderung

```
Transcript show: summe printString
```

nicht isoliert zur Ausführung gelangen kann. Dies liegt daran, daß “summe” als temporäre Variable nur solange zur Verfügung steht, wie die Anforderungen ausgeführt werden, in deren Verbindung diese Variable eingangs vereinbart wurde. Daher muß die Anforderung zur Anzeige immer zusammen mit den Anforderungen zur Summation ausgeführt werden.

4.3 Berechnung und Anzeige des Durchschnittswertes

Entwicklung des Lösungsplans

Nachdem wir kennengelernt haben, wie die erfaßten Werte summiert werden können, läßt sich der Lösungsplan für PROB-1-2 insgesamt durch das folgende Struktogramm beschreiben:

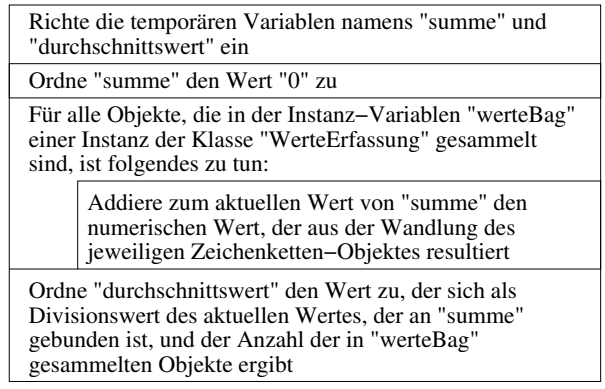


Abbildung 4.7: Struktogramm zur Durchschnittsberechnung

Um die Division von “summe” durch die Anzahl der summierten Werte durchzuführen, machen wir – unter Berücksichtigung des Ergebnis-Objektes, das wir durch eine Verschachtelung der Messages “bereitstellenWerte” und “size” erhalten – von der Möglichkeit Gebrauch, daß das Ergebnis-Objekt einer Message als Argument einer anderen Message weiterverarbeitet werden kann.

Sofern wir – ergänzend zu “summe” – eine weitere *temporäre* Variable namens “durchschnittswert” in der Form

```
|summe durchschnittswert|
```

vereinbaren, läßt sich der zu ermittelnde Durchschnittswert – für die Punktwerte

der Jahrgangsstufe 11 – durch die Anforderung

```
durchschnittswert := summe /
     (WerteErfassung11 bereitstellenWerte size)
```

errechnen und der Variablen “durchschnittswert” zuordnen.

Klammerung von Messages

Bei der Ausführung dieser Anforderung, bei der das Argument der binären Message “/” als Ergebnis-Objekt der verschachtelten Messages

```
WerteErfassung11 bereitstellenWerte size
```

bestimmt wird, ist die folgende Regel zu beachten:

- Sofern die standardmäßige Reihenfolge “von links nach rechts” bei der Abarbeitung von verschachtelten Messages beeinflußt werden soll, ist eine Klammerung mit den öffnenden und schließenden Klammern “(” und “)” vorzunehmen. Dabei können die Messages beliebig tief geklammert werden.
  Aus einer geklammerten Message resultiert entweder ein Empfänger-Objekt oder ein Argument einer Message.
  Bei der Bestimmung der Bearbeitungsreihenfolge ist zu beachten, daß eine eingeklammerte Message eine höhere Priorität als diejenige Message besitzt, deren Empfänger-Objekt bzw. Argument durch diese Klammerung festgelegt wird.

  Hinweis: Beim Einsatz von Klammern muß die Anzahl der öffnenden “(”-Klammern insgesamt gleich der Anzahl der schließenden “)”-Klammern sein. Außerdem muß das “Klammergebirge” ausbalanciert sein, d.h. die öffnenden und schließenden Klammern müssen paarweise einander – in sinnvoller Form – zugeordnet sein.

- Grundsätzlich müssen Keyword-Messages, die innerhalb verschachtelter Messages ein Empfänger-Objekt oder ein Argument einer Message bestimmen, eingeklammert werden.

Hinweis: Innerhalb der Anforderung

```
!durchschnittswert := summe / (WerteErfassung11 bereitstellenWerte size)!
haben wir die verschachtelte Message ‘‘WerteErfassung11 bereitstellenWerte size’’
aus didaktischen Gründen eingeklammert. Dies ist nicht notwendig, da zuerst die
beiden unären Messages ‘‘bereitstellenWerte’’ und ‘‘size’’ -- in dieser Reihenfolge
-- und erst danach die binäre Message ‘‘/’’ bearbeitet werden.
```

Gemäß der Regeln zur Ausführungsreihenfolge von verschachtelten Messages wird bei der Ausführung der Anforderung

```
durchschnittswert := summe /
     (WerteErfassung11 bereitstellenWerte size)
```

zuerst die Verschachtelung

```
WerteErfassung11 bereitstellenWerte size
```

verarbeitet und daher zunächst die Methode “bereitstellenWerte” ausgeführt. Hieraus resultiert als Ergebnis-Objekt ein *Bag*, dessen Bestandteile die erfaßten Punktwerte der Jahrgangsstufe 11 sind. Dieses Ergebnis-Objekt ist das Empfänger-Objekt der unären Message “size”. Das aus dieser Message resultierende Ergebnis-Objekt ist gleich der Anzahl der erfaßten Punktwerte. Diese Zahl stellt das Argument der binären Message “/” dar, die dem Empfänger-Objekt “summe” zugestellt wird. Durch die Ausführung der Methode “/” erfolgt eine Division, so daß der resultierende Quotient der Variablen “durchschnittswert” zugeordnet wird.

Umsetzung des Lösungsplans

Um den Durchschnittswert für die Punktwerte der Jahrgangsstufe 11 zu ermitteln, können wir das in Abbildung 4.7 angegebene Struktogramm wie folgt umsetzen:

```
|summe durchschnittswert|
summe := 0.
WerteErfassung11 bereitstellenWerte
      do: [:einObjekt|summe := summe + einObjekt asInteger].
durchschnittswert := summe /
     (WerteErfassung11 bereitstellenWerte size)
```

Lassen wir diese Anforderungen – unter Einsatz der Menü-Option “‘Show It” des Menüs “Smalltalk” – ausführen, so wird der Durchschnittswert der Punktwerte der Jahrgangsstufe 11 im Workspace-Fenster angezeigt.

Indem wir “WerteErfassung11” durch “WerteErfassung12” ersetzen, läßt sich der Durchschnittswert für die innerhalb der Instanz “WerteErfassung12” gesammelten Punktwerte der Jahrgangsstufe 12 errechnen.

Soll der Durchschnittswert im Transcript-Fenster angezeigt werden, so sind die angegebenen Anforderungen wie folgt zu ergänzen:

```
Transcript cr.
Transcript show: 'Der Durchschnittswert ist: '.
Transcript show: durchschnittswert asFloat printString
```

Hinweis: In diesem Fall ist zu beachten, daß – beim Einsatz von “Show It” – im Workspace “a MDITranscript” als Ergebnis-Objekt der letzten Anforderung mit der Message “show:” angezeigt wird.

Durch diese Anforderungen wird festgelegt, daß der ermittelte Durchschnittswert im Transcript-Fenster hinter dem Text “Der Durchschnittswert ist: ” angezeigt werden soll.

Damit der Durchschnittswert als Dezimalzahl ausgegeben wird, setzen wir die Basis-Methode “asFloat” ein. Es gilt nämlich:

- **“asFloat”**:
  Die unäre Message “asFloat” läßt sich einer beliebigen Zahl zustellen. Als Ergebnis-Objekt resultiert eine Dezimalzahl.

  Hinweis: Zum Beispiel liefert die Anforderung “10 / 4 asFloat” die Zahl “2.5” als Ergebnis-Objekt. Dagegen führt z.B. “10 / 4” zum Ergebnis-Objekt “5 / 2”,

das eine Instanz der Basis-Klasse “Fraction” darstellt (Instanzen sind Brüche mit ganzzahligem Zähler und Nenner, siehe Abschnitt 5.5.4).

Es ist zu beachten, daß innerhalb der Anforderung

```
Transcript show: durchschnittswert asFloat printString
```

das Argument von “show:” durch das Ergebnis-Objekt von

```
durchschnittswert asFloat printString
```

bestimmt werden muß, da die Keyword-Message “show:” eine Zeichenkette als Argument benötigt und folgendes gilt:

- **“printString”**:
  Zur Wandlung einer Zahl in eine Zeichenkette kann die Basis-Methode “printString” eingesetzt werden. Als Ergebnis-Objekt der unären Message “printString” resultiert eine Zeichenkette, die sich aus der Wandlung des Empfänger-Objektes ergibt.

Kaskade

Um Schreibarbeit zu sparen, können mehrere aufeinanderfolgende Anforderungen, die sämtlich *dasselbe* Empfänger-Objekt besitzen, in kaskadierter Form angegeben werden.

- Eine *Kaskade* besteht aus zwei oder mehreren Messages, die sich sämtlich an dasselbe Empfänger-Objekt richten. Dieses Empfänger-Objekt wird durch das Empfänger-Objekt der zuerst aufgeführten Message bestimmt. Jede nachfolgende Message wird ohne Empfänger-Objekt angegeben und von der jeweils unmittelbar zuvor aufgeführten Message durch ein Semikolon “;” abgegrenzt. Als Ergebnis-Objekt der Kaskade ist das Ergebnis-Objekt der zuletzt ausgeführten Message festgelegt.

Die Anforderungen

```
Transcript cr.
Transcript show: 'Der Durchschnittswert ist: '.
Transcript show: durchschnittswert asFloat printString
```

können daher abkürzend in der Form

```
Transcript cr;
     show: 'Der Durchschnittswert ist: ';
     show: durchschnittswert asFloat printString
```

geschrieben werden, so daß wir den Lösungsplan zur Berechnung und Anzeige des Durchschnittswertes – für die Punktwerte der Jahrgangsstufe 11 – insgesamt wie folgt angeben können:

```
|summe durchschnittswert|
summe := 0.
WerteErfassung11 bereitstellenWerte
      do: [:einObjekt|summe := summe + einObjekt asInteger].
durchschnittswert := summe /
     (WerteErfassung11 bereitstellenWerte size).
Transcript cr;
     show: 'Der Durchschnittswert ist: ';
     show: durchschnittswert asFloat printString
```

Ergebnis-Objekt einer Kaskade

Durch die Ausführung kaskadierter Messages erhalten wir als Ergebnis-Objekt der Kaskade das Ergebnis-Objekt der zuletzt ausgeführten Message. Soll das Ergebnis-Objekt einer Kaskade durch das Empfänger-Objekt der Kaskade festgelegt sein, so ist die Basis-Methode "yourself" einzusetzen.

- Wird die Basis-Methode *"yourself"* als letzte Message aufgeführt, so wird das Empfänger-Objekt der Kaskade zum Ergebnis-Objekt der Kaskade.

Zum Beispiel ist das Empfänger-Objekt von

```
Bag new add: 1; add: 2
```

ein Bag und das Ergebnis-Objekt dieser Kaskade gleich der Zahl "2", da das Ergebnis-Objekt der Message "add:" gleich dem als Argument aufgeführten Wert ist.

Damit als Ergebnis-Objekt der Kaskade das Empfänger-Objekt, d.h. der Bag selbst resultiert, ist die Message "yourself" wie folgt zu ergänzen:

```
Bag new add: 1; add: 2; yourself
```

Hinweis: Setzen wir nach der Erfassung von Punktwerten die Message "yourself" z.B. innerhalb der Kaskade

```
WerteErfassung11 bereitstellenWerte; yourself
```

ein, so erhalten wir "a WerteErfassung" als Ergebnis-Objekt angezeigt, sofern wir die Ausführung über die Menü-Option "Show It" des Menüs "Smalltalk" abrufen. Diese Ausgabe kennzeichnet den Sachverhalt, daß "WerteErfassung11" eine Instanz der Klasse "WerteErfassung" ist.

Ist als letzte Anforderung innerhalb einer Methoden-Vereinbarung eine Kaskade enthalten, die durch das Return-Zeichen "∧" eingeleitet wird, so ist das Ergebnis-Objekt dieser Methode bekanntlich *durch* das Ergebnis-Objekt der letzten, innerhalb der Kaskade aufgeführten Message bestimmt. Soll erreicht werden, daß stattdessen das Empfänger-Objekt der Kaskade als Ergebnis-Objekt der Methodenausführung resultiert, so muß die Kaskade durch die unäre Message "yourself" beendet werden.

4.4 Kommunikation eines Objektes mit sich selbst ("self")

Die bisherigen Erörterungen dienten allein dazu, die Vorbereitungen zur Entwicklung einer geeigneten Methode zu treffen, mit der sich die Durchschnittsbildung durchführen und deren Ergebnis im Transcript-Fenster anzeigen läßt.

Auf der Basis der oben aufgeführten Anforderungen wollen wir jetzt eine Methode namens "anzeigenDurchschnitt" entwickeln, mit der sich – nach ihrer Vereinbarung innerhalb der Klasse "WerteErfassung" – jahrgangsstufen-spezifische Durchschnittswerte ermitteln und anzeigen lassen.

Dazu müssen die Anforderungen innerhalb der Methode so aufgebaut sein, daß überall dort, wo "WerteErfassung11" als Empfänger-Objekt einer Message angegeben ist, dasjenige Objekt verwendet wird, das das jeweilige Empfänger-Objekt der Message "anzeigenDurchschnitt" darstellt, durch die die Methode "anzeigenDurchschnitt" zur Ausführung gebracht werden soll.

- Damit das Empfänger-Objekt einer Message als Empfänger-Objekt in die Anforderungen der korrespondierenden Methode übernommen wird, steht die Pseudovariable *"self"* zur Verfügung. Diese Pseudovariable läßt sich innerhalb einer Methode an den Positionen als Platzhalter verwenden, an denen das Empfänger-Objekt der Message aufgeführt werden soll. Bei der Ausführung der Methode wird anstelle von "self" jeweils das Empfänger-Objekt der Message eingesetzt, durch die die Methode zur Ausführung gelangt.

Durch den Einsatz der Pseudovariablen "self" ist gewährleistet, daß ein Objekt mit sich selbst kommunizieren kann.
Wird nämlich bei einer Methoden-Vereinbarung, in der "self" enthalten ist, die mit dieser Methode korrespondierende Message an ein Empfänger-Objekt geschickt, so sendet dieses Objekt – bei der Methoden-Ausführung – die innerhalb der Methoden-Vereinbarung enthaltenen Messages an die jeweils aufgeführten Empfänger-Objekte. Bei derjenigen Message, bei der "self" als Empfänger-Objekt angegeben ist, adressiert das ausführende Objekt diese Message an sich selbst, d.h. es kommuniziert mit sich selbst.

Sofern ein Objekt mit sich selbst kommuniziert, ist die folgende Aussage von Bedeutung:

- Enthält eine auszuführende Methode eine Message mit der Pseudovariablen "self", so wird die mit dieser Message korrespondierende Methode in derjenigen Klasse gesucht, zu deren Instanziierungen dasjenige Empfänger-Objekt zählt, für das der Platzhalter "self" steht.

Die Methode "anzeigenDurchschnitt"

Übernehmen wir die obigen Anforderungen in die Methoden-Vereinbarung von "anzeigenDurchschnitt" und verwenden wir anstelle des Namens "WerteErfassung11" die Pseudovariable "self", so ergibt sich die folgende Methoden-Vereinbarung:

```
anzeigenDurchschnitt
|summe durchschnittswert|
summe := 0.
self bereitstellenWerte
     do: [:einObjekt|summe := summe + einObjekt asInteger].
durchschnittswert := summe / (self bereitstellenWerte size).
Transcript cr;
     show: 'Der Durchschnittswert ist: ';
     show: durchschnittswert asFloat printString
```

Die Form, in der wir die Methoden-Vereinbarung von "anzeigenDurchschnitt" angegeben haben, ist grundlegend für die folgende Struktur, nach der die Vereinbarung einer Methode vorgenommen werden muß:

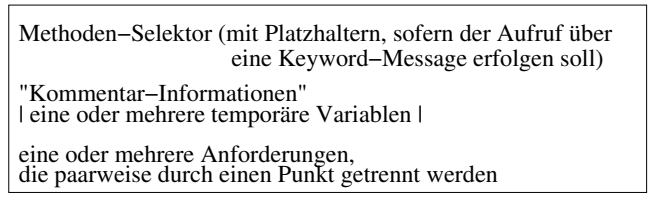


Abbildung 4.8: Aufbau einer Methoden-Vereinbarung

Hinweis: Bei der Vereinbarung einer Methode können sowohl die Kommentar-Informationen als auch die Angabe von temporären Variablen fehlen.

Nachdem wir die Methode "anzeigenDurchschnitt" in der Klasse "WerteErfassung" vereinbart haben, können wir uns den Durchschnittswert für die Jahrgangsstufe 11 durch die Anforderung

```
WerteErfassung11 anzeigenDurchschnitt
```

und den Durchschnittswert für die Jahrgangsstufe 12 durch die Anforderung

```
WerteErfassung12 anzeigenDurchschnitt
```

ermitteln und im Transcript-Fenster anzeigen lassen.

Hinweis: Als Alternative könnte die Methode "anzeigenDurchschnitt" auch wie folgt vereinbart werden:

```
anzeigenDurchschnitt
|summe durchschnittswert|
summe := 0.
werteBag do: [:einObjekt|summe := summe + einObjekt asInteger].
durchschnittswert := summe / werteBag size.
Transcript cr;
     show: 'Der Durchschnittswert ist: ';
     show: durchschnittswert asFloat printString
```

Hierdurch wird der Zugriff auf die Werte der Instanz-Variablen nicht durch die Ausführung der Methode "bereitstellenWerte", sondern direkt über die Verwendung des Variablennamens "werteBag" festgelegt.
Wir verzichten auf diese vereinfachte Form, um die Methode "anzeigenDurchschnitt" auch zur Lösung anderer Problemstellungen einsetzen zu können.
Werden z.B. im Rahmen eines erweiterten Bildschirmformulars nicht nur die Punktwerte, sondern zusätzlich kennzeichnende Daten wie z.B. das Geschlecht in Form von "m" bzw. "w" erfaßt, so können die eingegebenen Werte – entsprechend dem ursprünglichen Lösungsplan – wiederum in den Sammel-Behälter "werteBag" eingetragen werden – und zwar in Form von geordneten Paaren wie z.B. "('37' 'm')". Um anschließend den Inhalt von "werteBag" geeignet auswerten zu können, muß auf die einzelnen Komponenten der geordneten Paare zugegriffen werden. Da folglich keine direkte Bearbeitung von "werteBag" mehr durchführbar ist, müßte z.B. die Methode "anzeigenDurchschnitt" für eine Durchschnittsberechnung neu vereinbart werden. Somit ist es vorteilhaft, bereits innerhalb der Methode "anzeigenDurchschnitt" einen indirekten Zugriff auf die Werte von "werteBag" mittels der Methode "bereitstellenWerte" durchzuführen, so daß im Lösungsplan allein eine neue Vereinbarung der Methode "bereitstellenWerte" im Rahmen einer neuen Klassen-Vereinbarung vorgesehen werden muß.

Die Methode "sammelnWerte:"

Um für das Folgende eine Methode zur Verfügung zu haben, mit der sich die detaillierten Anforderungen zur Initialisierung und Durchführung der Erfassung abkürzend beschreiben lassen, vereinbaren wir innerhalb der Klasse "WerteErfassung" ergänzend die Methode "sammelnWerte:", die wir – unter Einsatz einer Kaskade – in der folgenden Form festlegen:

```
sammelnWerte: aString
self initialisierenErfassung;
     festlegenUeberschrift: aString;
     durchfuehrenErfassung
```

Wird die Methode "sammelnWerte:" durch die Message "sammelnWerte:" zur Ausführung gebracht, so wird zuvor – vom SMALLTALK-System – an allen Positionen, an denen "aString" innerhalb der Methode "sammelnWerte:" aufgeführt ist, die als Argument der Message "sammelnWerte:" angegebene konkrete Überschrift wie z.B. "Jahrgangsstufe 11" eingetragen.

Nach der Vereinbarung der Methode "sammelnWerte:" befindet sich die Klasse "WerteErfassung" in folgendem Zustand:

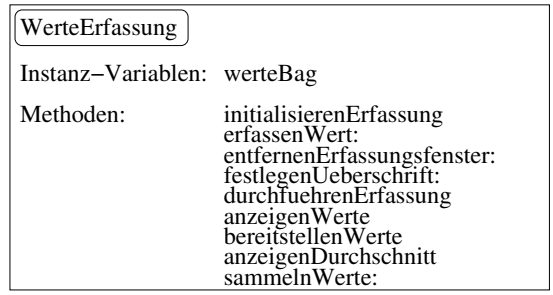


Abbildung 4.9: Aktuelle Form der Klasse “WerteErfassung”

Auf dieser Basis kann der Erfassungsprozeß für die Punktwerte der Jahrgangsstufe 11, der bislang durch die Anforderungen

```
WerteErfassung11 := WerteErfassung new.
WerteErfassung11 initialisierenErfassung;
   festlegenUeberschrift: 'Jahrgangsstufe 11';
   durchfuehrenErfassung
```

gestartet werden konnte, durch

```
WerteErfassung11 := WerteErfassung new.
WerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

in vereinfachter Form abgerufen werden.

Hinweis: Statt dieser Anforderungen hätten wir auch

```
WerteErfassung11 := WerteErfassung new
  initialisierenErfassung;
  festlegenUeberschrift: 'Jahrgangsstufe 11';
  durchfuehrenErfassung
```

oder

```
WerteErfassung11 := WerteErfassung new
  sammelnWerte: 'Jahrgangsstufe 11'
```

angeben können.

Kapitel 5

Spezialisierung von Lösungsplänen

5.1 Vererbung

Um die formular-gestützte Erfassung von Werten und die Anzeige des Durchschnittswertes zu programmieren, haben wir einen *gestuften* Lösungsplan entwickelt.

Aufbauend auf der Lösung von PROB-1-1, durch die die Werte erfaßt und gesammelt werden können, haben wir einen Lösungplan für PROB-1-2 vorgeschlagen, durch dessen Ausführung sich der Durchschnittswert berechnen und anzeigen läßt.

Aus didaktischen Gründen haben wir zunächst eine Gesamtlösung in Form der Klasse "WerteErfassung" angegeben, bei der die Berechnung und Anzeige des Durchschnittswertes mittels der Methode "anzeigenDurchschnitt" festgelegt wurde.

Dieses Vorgehen ist nicht im Einklang mit den generellen Zielsetzungen der objekt-orientierten Programmierung, weil strikt zwischen der *Erfassung* der Werte und einer *Verarbeitung* der Werte getrennt werden sollte.

Diese Strategie beruht auf der folgenden Sichtweise:

- Die Programmierung mit SMALLTALK basiert auf Basis-Klassen des SMALLTALK-Systems, die sich zur Lösung von Problemstellungen durch neue Klassen ergänzen lassen.
- Jede aus einer Problemlösung resultierende Klasse erweitert die Gesamtheit der vorhandenen Klassen um eine neue Klasse.
- Durch objekt-orientiertes Programmieren läßt sich das jeweils vorliegende SMALLTALK-System daher schrittweise durch zusätzliche Bausteine ergänzen. Diese Erweiterung sollte möglichst so erfolgen, daß die Beschreibung eines neuen Lösungsplans als *Spezialisierung* eines anderen, bereits durch eine Klassen-Vereinbarung umgesetzten Lösungsplans erreicht werden kann.

  Hinweis: Das im Kapitel 1 verwendete Schlagwort "vom Allgemeinen zum Speziellen" müßte eigentlich "Erweiterung durch Spezialisierung" lauten, da die bereits zur Verfügung stehenden Klassen durch eine neue Klasse so zu erweitern sind, daß der Lösungsplan einer speziellen Problemstellung beschrieben und zur Ausführung gebracht werden kann.

In Anbetracht dessen, daß die Lösung von PROB-1-1 nicht nur als Basis für die Lösung von PROB-1-2, sondern auch vieler anderer Problemstellungen dienen kann,

bei denen Berechnungen mit erfaßten Werten durchzuführen sind, sollten wir den Lösungsplan für PROB-1-2 nicht mit dem Lösungsplan für PROB-1-1 vermengen, sondern seine Umsetzung durch die Einrichtung einer *neuen* Klasse herbeiführen, die als Spezialisierung von "WerteErfassung" angesehen werden kann.

Deswegen löschen wir innerhalb der Klasse "WerteErfassung" die zuvor verabredete Methode "anzeigenDurchschnitt", indem wir den zugehörigen Methoden-Selektor im Methoden-Bereich des Klassen-Hierarchie-Browser-Fensters anklicken und die Menü-Option "Remove" des Menüs "Methods" bestätigen.

Anschließend besitzt die Klasse "WerteErfassung" den folgenden Zustand, der die Basis für die Lösung der nachfolgend angegebenen Problemstellungen darstellen soll:

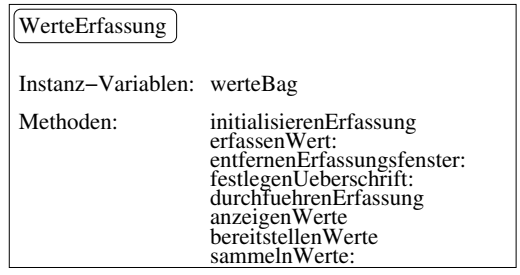


Abbildung 5.1: Aktuelle Form der Klasse "WerteErfassung"

Im folgenden streben wir die Einrichtung einer Klasse an, mit der der zu PROB-1-2 gehörende Lösungsplan zur Ausführung gebracht werden kann.

Hinweis: Im Abschnitt 5.3 geben wir an, wie die resultierende Klasse unter dem Namen "InWerteErfassung" eingerichtet wird.

Damit wir die Forderung erfüllen können, einen Lösungsplan als *Spezialisierung* eines anderen Lösungsplans anzugeben, müssen wir zuvor klären, in welcher Beziehung eine Klasse, die neu eingerichtet wird, zu sämtlichen bereits vom SMALLTALK-System verwalteten Klassen steht. Dazu sind die folgenden Aussagen von grundlegender Bedeutung:

- Eine neue Klasse wird immer als *Unterklasse* genau einer bereits vorhandenen Klasse eingerichtet, die als *Oberklasse* der neuen Klasse bezeichnet wird.
- Durch die Unterordnung erfolgt eine *Vererbung*. Dies bedeutet, daß die neue Unterklasse sämtliche Attribute der ihr übergeordneten Klasse übernimmt und jede Instanz der Unterklasse Kenntnis von sämtlichen Methoden besitzt, die innerhalb der Oberklasse vereinbart oder bekannt sind.

Hinweis: Im Klassen-Bereich des Klassen-Hierarchie-Browser-Fensters wird der Name der Klasse, die als aktive Klasse eingestellt ist und sich als Unterklasse einer unmittelbar

übergeordneten Oberklasse darstellt, gegenüber dem Namen der Oberklasse nach rechts eingerückt angezeigt.

Durch die Vererbung werden sämtliche Instanz-Variablen, die innerhalb der Oberklasse festgelegt und bekannt sind, an die Instanzen der Unterklasse vererbt. Die Attribute von Instanzen der Unterklasse sind daher auch durch Instanz-Variablen bestimmt, die als Instanz-Variablen der Oberklasse festgelegt sind.

Ferner kann eine Instanz der Unterklasse jede Methode ausführen, die in der Oberklasse vereinbart oder bekannt ist.

Der Sachverhalt, daß eine Klasse “U” einer Klasse “O” untergeordnet ist, läßt sich wie folgt veranschaulichen:

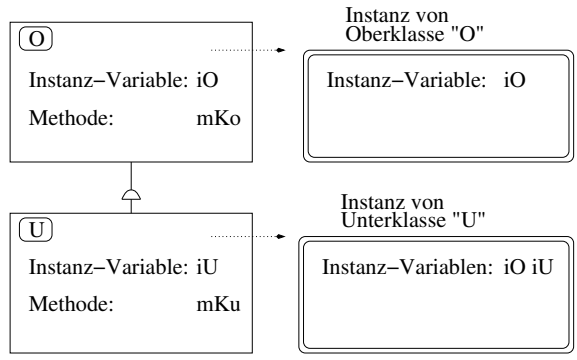


Abbildung 5.2: Spezialisierung durch Vererbung

Hinweis: In dieser Darstellung sind die grafischen Symbole für die beiden Klassen miteinander verbunden. Die Verbindungslinie beginnt am unteren Rand der übergeordneten Klasse und endet am oberen Rand der untergeordneten Klasse. Dabei zeigt die Grundseite des zugehörigen Halbkreises immer in Richtung der untergeordneten Klasse.

Die Klasse “O” vererbt ihre Instanz-Variablen und die Zugriffsmöglichkeit auf ihre Methoden an die Klasse “U”, d.h. “U” erbt die Vereinbarungen von “O”.

Somit besitzt jede Instanz der Klasse “U” sowohl die Instanz-Variable “iO” als auch die Instanz-Variable “iU”. Sie kennt nicht nur die Methode “mKu”, sondern auch die Methode “mKo”.

Jede Instanz der Klasse “O” verfügt über die Instanz-Variable “iO” und kennt die Methode “mKo”. Sie besitzt *nicht* die Instanz-Variable “iU” und kennt auch *nicht* die Methode “mKu”.

Die Klasse “U” stellt eine *Spezialisierung* der Klasse “O” dar, da in der Unterklasse “U” eine zusätzliche Instanz-Variable vereinbart und eine weitere Methode bekannt ist.

Hinweis: In der Unterklasse “U” müssen nicht notwendigerweise neue Instanz-Variablen und zusätzliche Methoden vereinbart werden. Oftmals ist allein die Vereinbarung von Instanz-Variablen bzw. allein die Vereinbarung weiterer Methoden sinnvoll.

Bei der Vereinbarung der Klasse "WerteErfassung" haben wir diese Klasse – unter Einsatz des Werkzeugs "WindowBuilder Pro/V" – der Basis-Klasse "ViewManager" untergeordnet, weil die Methoden dieser Klasse und ihre Instanz-Variablen benötigt werden, um einen Lösungsplan für die dialog-orientierte Erfassung innerhalb unseres Erfassungsfensters entwickeln zu können.
Dadurch erbt die Klasse "WerteErfassung" alle Attribute, die in der Klasse "ViewManager" durch Instanz-Variablen festgelegt sind.

- Dies ist die Erklärung dafür, daß wir zu keinem Zeitpunkt Aussagen über Verwaltungs-Behälter machen mußten, in denen fenster-spezifische Angaben enthalten sind. Durch die Vererbung wird bewirkt, daß jede Instanz von "WerteErfassung" über sämtliche Instanz-Variablen verfügt, in denen die Daten für den Aufbau und die Anzeige von Erfassungsfenstern festgelegt sind.

5.2 Klassen-Hierarchie

Das Prinzip der Vererbung wirkt nicht nur einstufig im Hinblick auf die unmittelbare Unterordnung, sondern über *alle* Stufen sämtlich einander untergeordneter Klassen. Von entscheidender Bedeutung für die Programmierung in SMALLTALK ist der folgende Sachverhalt:

- Alle Klassen, die vom SMALLTALK-System verwaltet werden, sind *hierarchisch* geordnet. Dies bedeutet, daß eine Klasse im Rahmen einer *Klassen-Hierarchie* mehr als einer Klasse untergeordnet sein kann. Sie besitzt jedoch immer genau eine direkte Oberklasse.

  Hinweis: Diese Art von Vererbung wird als *Einfachvererbung* bezeichnet.

- Die Vererbung ist über alle Hierarchiestufen wirksam, so daß jede Klasse des SMALLTALK-Systems sämtliche Attribute aller ihr hierarchisch übergeordneter Klassen erbt und jede Instanz einer Klasse Kenntnis von allen Methoden besitzt, die in hierarchisch übergeordneten Klassen vereinbart sind.
  Daher verkörpert jede Instanz einer Klasse die Gesamtheit aller Instanz-Variablen, die in dieser und sämtlichen ihr hierarchisch übergeordneten Klassen festgelegt sind. Ferner kann jede Instanz einer Klasse sämtliche Methoden ausführen, die in diesen Klassen und in hierarchisch übergeordneten Klassen vereinbart sind.

  Hinweis: Im Hinblick auf die Klassen-Hierarchie sind die Instanzen immer denjenigen Klassen zugeordnet, aus denen sie instanziiert wurden. Eine Veränderung dieser Zuordnung ist *nicht* möglich.
  Attribute sollten in einer Hierarchie "so hoch wie möglich" angesiedelt werden. Dadurch können redundante Vereinbarungen in verschiedenen, einander untergeordneten Klassen vermieden werden.

Sind z.B. die Klassen "K1", "K2" bis "Kn" der Klasse "K0" hierarchisch untergeordnet, so können wir dies folgendermaßen darstellen:

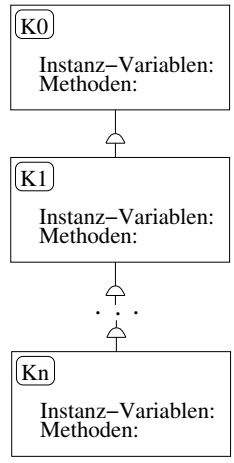


Abbildung 5.3: Spezialisierung durch hierarchische Unterordnung

Jede Instanz der Klasse "Kn" enthält sämtliche Instanz-Variablen der Klassen "K0", "K1" bis hin zu "Kn" und kann – außer den eigenen Methoden – jede Methode ausführen, die innerhalb einer dieser Oberklassen vereinbart ist.

Im Hinblick auf die Ausführung von Messages hat die Klassen-Hierarchie die folgende Auswirkung:

- Wird eine Message an ein Objekt geschickt, so sucht dieses Objekt die zugehörige Methode zunächst in der Klasse, als deren Instanz es erzeugt wurde. Ist die Suche erfolgreich, so wird die ermittelte Methode von der Instanz ausgeführt. Wird jedoch die gesuchte Methode nicht in dieser Klasse gefunden, so wird die Suche in der *unmittelbar* übergeordneten Klasse fortgesetzt, bei erneut erfolgloser Suche anschließend in der nächst übergeordneten Klasse, usw.

Hinweis: In dem Fall, in dem bei einer Anforderung mittels einer Message keine dazu korrespondierende Methode von dem jeweiligen Empfänger-Objekt gefunden werden kann, wird vom SMALLTALK-System das Walkback-Fenster mit einer erläuternden Fehlermeldung angezeigt, das durch die Tastenkombination "Alt+F4" geschlossen werden kann (nähere Angaben zum Walkback-Fenster machen wir im Anhang unter A.3).

Im Hinblick auf diesen Sachverhalt lassen sich, sofern Änderungen an der grundsätzlichen Form eines Lösungsplans erforderlich sind, die hiermit korrespondierenden Angaben innerhalb einer Oberklasse ändern. Dadurch stehen die geänderten Methoden für sämtliche Instanzen, die aus den zugehörigen Unterklassen instanziiert wurden, *automatisch* zur Verfügung.

Hinweis: Eine Änderung innerhalb einer Oberklasse hat Auswirkungen auf sämtliche Unterklassen dieser Oberklasse.

Als Folge der Vererbung gilt für die Attribute einer Instanz:

- Sie sind festgelegt als die Gesamtheit aller Instanz-Variablen, die in der Klasse, zu der die Instanz gehört, und allen ihr hierarchisch übergeordneten Klassen vereinbart sind.
- Da Instanz-Variablen sich namensmäßig unterscheiden müssen, darf es unter allen Instanz-Variablen einer neu einzurichtenden Klasse und sämtlichen Instanz-Variablen der jeweils hierarchisch übergeordneten Klassen *keine gleichnamigen* Instanz-Variablen geben.

Hinweis: Im Klassen-Hierarchie-Browser-Fenster werden, sofern die Klasse "WerteErfassung" aktiviert ist, innerhalb des Variablen-Bereichs zuoberst die in der Klasse festgelegten Instanz-Variablen (hier: "werteBag") und weiter unterhalb die innerhalb dieser Klasse desweiteren bekannten Instanz-Variablen anderer Klassen (hier: "views") und – durch Striche markiert – der jeweilige Name (hier: "– ViewManager –" und "– Object –") der zugehörigen Klasse angezeigt.

Im Rahmen der hierarchischen Gliederung können einer Klasse nicht nur eine, sondern auch zwei oder mehrere Klassen auf *derselben* Hierarchiestufe untergeordnet werden.
Sind z.B. der Klasse "K0" die Klassen "K1", "K2" bis "Kn" auf derselben Hierarchiestufe untergeordnet, so ergibt sich die folgende Situation:

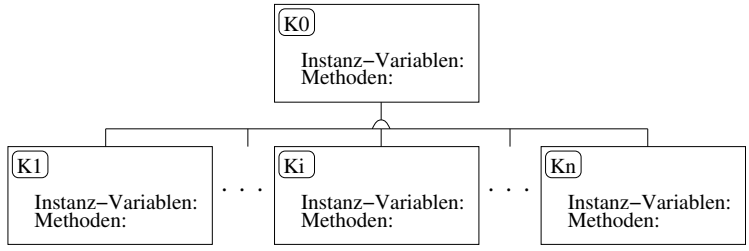


Abbildung 5.4: Unterordnung auf derselben Hierarchiestufe

Hinweis: Im Klassen-Hierarchie-Browser-Fenster werden die Namen von Klassen, die einer Klasse auf derselben Hierarchiestufe untergeordnet sind, sämtlich – ohne Einrückungen – untereinander im Klassen-Bereich angegeben.

Eine Instanz der Klasse "Ki" kann nur diejenigen Methoden ausführen, die innerhalb der Klasse "Ki" und in der Oberklasse "K0" (sowie den "K0" übergeordneten Klassen) vereinbart und bekannt sind. Eine Instanz von "Ki" besitzt keine Instanz-Variablen aus den anderen Klassen, die "K0" untergeordnet sind, sondern nur die Instanz-Variablen, die in "Ki" und "K0"(sowie den "K0" übergeordneten Klassen) festgelegt sind.

Für die objekt-orientierte Programmierung ist die folgende Feststellung von grundlegender Bedeutung:

- Da die Vererbung nicht nur über eine, sondern über alle Hierarchiestufen wirkt, gehört zur Entwicklung eines Lösungsplans nicht nur die Fähigkeit,

eine geeignete Klasse zu vereinbaren, sondern in besonderem Maße auch die Kenntnis von Attributen und Methoden, die als Bestandteil bereits vorhandener Klassen für die Programmierung zur Verfügung stehen. Erst durch diese Kenntnisse lassen sich objekt-orientierte Programmentwicklungen effizient durchführen.

Für einen Anfänger ist es nicht ganz einfach, die für die jeweilige Problemlösung erforderlichen Anforderungen spontan anzugeben. Dies liegt daran, daß man sich zunächst einmal einen Kenntnisstand über eine Vielzahl von Basis-Methoden verschaffen muß. Erst anschließend kann man beurteilen, ob eine für eine Problemlösung benötigte Methode bereits als Basis-Methode zur Verfügung steht oder als Methode aus geeigneten Anforderungen aufzubauen ist, die auf Basis-Methoden basieren.

Hinweis: Nähere Angaben zu den von uns bereits verwendeten Basis-Methoden machen wir im Kapitel 6.
Wollen wir Informationen über Klassen und Methoden des Basis-Systems abrufen, so können wir – auf der Ebene des Window-Systems – das Ikon "Encyclopedia of classes" aktivieren.

Um zu wissen, welche Basis-Methoden von den Instanzen einer Klasse ausgeführt werden können, muß man einen Einblick in die Gesamtheit der Methoden nehmen, die in dieser oder in ihr hierarchisch übergeordneten Klassen vereinbart sind.
Dies ist durch den Einsatz des Klassen-Hierarchie-Browsers möglich, indem die Menü-Optionen "Senders", "Implementors", "Local Senders" und "Local Implementors" des Menüs "Methods" verwendet werden.
Stellen wir z.B. im Klassen-Hierarchie-Browser-Fenster die Klasse "WerteErfassung" als aktive Klasse ein und markieren anschließend z.B. im Methoden-Bereich die Methode "entfernenErfassungsfenster:", so führt die Anwahl der Menü-Option "Implementors" zur Anzeige eines Fensters mit dem folgenden Eintrag:

```
WerteErfassung >> entfernenErfassungsfenster:
```

Dies bedeutet, daß die Methode "entfernenErfassungsfenster:" lediglich innerhalb der Klasse "WerteErfassung" vereinbart (implementiert) ist.

Wollen wir z.B. wissen, innerhalb welcher Methode die Message "entfernenErfassungsfenster:" eingesetzt wird, so schließen wir dieses Fenster und wählen im Menü "Methods" die Option "Senders". Daraufhin erhalten wir ein neues Fenster mit der folgenden Anzeige:

```
WerteErfassung >> initialisierenErfassung
```

Hieraus können wir erkennen, daß die Methode "entfernenErfassungsfenster:" innerhalb der Methode "initialisierenErfassung", die in der Klasse "WerteErfassung" vereinbart ist, aufgerufen wird.

Grundsätzlich läßt sich folgendes feststellen:

- Um sich darüber zu informieren, in welchen Klassen die im Methoden-Bereich markierte Methode *zusätzlich* vereinbart ist, kann die Menü-Option "Implementors" angewählt werden.

Hinweis: Die resultierende Anzeige läßt sich auch unter Einsatz der Keyword-Message "implementorsOf:" abrufen, indem die globale Variable "Smalltalk" und der Methodenname "<selektor>" in der folgenden Form aufgeführt werden:
`Smalltalk implementorsOf: #<selektor>`
Dabei handelt es sich bei der globalen Variablen "Smalltalk" um die einzige Instanz der Basis-Klasse "SystemDictionary", in der die Klassen und die globalen Objekte gesammelt werden (siehe Abschnitt 9.4.2).

- Sollen diejenigen Unterklassen der aktuellen Klasse festgestellt werden, in denen die im Methoden-Bereich markierte Methode durch eine Methode gleichen Namens *überdeckt* wird, so läßt sich dies über die Menü-Option "Local Implementors" erreichen.

- Um die Namen aller derjenigen Methoden anzuzeigen, in deren Anforderungen die im Methoden-Bereich markierte Methode zur Ausführung aufgerufen wird, muß die Menü-Option "Senders" bestätigt werden.

  Hinweis: Die resultierende Anzeige läßt sich auch unter Einsatz der Keyword-Message "sendersOf:" abrufen, indem der Methodenname "<selektor>" und die globale Variable "Smalltalk" in der Form
  `Smalltalk sendersOf: #<selektor>`
  aufgeführt werden.

- Sollen die Namen aller derjenigen Methoden angezeigt werden, in deren Anforderungen die im Methoden-Bereich markierte Methode zur Ausführung aufgerufen wird und die Bestandteil von Methoden-Vereinbarungen untergeordneter Klassen sind, kann die Menü-Option "Local Senders" verwendet werden.

5.3 Vereinbarung einer Klasse

Nachdem wir das Prinzip der Vererbung kennengelernt haben, greifen wir unser in Abschnitt 5.1 formuliertes Ziel wieder auf, für PROB-1-2 einen *gestuften* Lösungsplan auf der Basis der Klasse "WerteErfassung" zu realisieren.

Um den Durchschnittswert der erfaßten Punktwerte berechnen und anzeigen zu können, wollen wir eine Klasse namens "InWerteErfassung" entwickeln, die als Unterklasse der Klasse "WerteErfassung" eingerichtet werden soll.

Hinweis: Zur Bezeichnung einer Unterklasse sollte man einen Namen wählen, in dem der Name der Oberklasse enthalten ist.
Der Name "WerteErfassung" wird um die Vorsilbe "In" zur Abkürzung von "Intervallskaliert" ergänzt. Die Eigenschaft, daß die Daten eine durch die Bezeichnung "intervallskaliert" gekennzeichnete Eigenschaft besitzen, ist die Voraussetzung dafür, daß die *Durchschnittsbildung* für die Daten eine sinnvolle statistische Auswertung darstellt.

Für die Klasse "InWerteErfassung" sehen wir vor, daß der Durchschnittswert der Punktwerte einmal berechnet und daraufhin jederzeit wieder abgerufen werden kann, ohne daß eine erneute Durchschnittsbildung erfolgen muß.

Sofern wir beabsichtigen, bei der Klassen-Vereinbarung von "InWerteErfassung" die Instanz-Variable "durchschnittswert" – zur Sicherung des jeweils ermittelten Durchschnittswertes – als Attribut vorzusehen, können wir die Berechnung des

Durchschnittswertes durch die wie folgt zu vereinbarende Methode “durchschnitt” festlegen:

```
durchschnitt
|summe|
summe := 0.
self bereitstellenWerte
     do: [:einObjekt|summe:= summe + einObjekt asInteger].
durchschnittswert:= summe / (self bereitstellenWerte size)
```

Hinweis: Im Gegensatz zur ursprünglich entwickelten Lösung in Form der Methode “anzeigenDurchschnitt” (siehe Abschnitt 4.4) wird in diesem Fall durch den Variablennamen “durchschnittswert” keine temporäre Variable, sondern eine Instanz-Variable gekennzeichnet.

Zur Anzeige des gesicherten Durchschnittswertes im Transcript-Fenster sehen wir die folgende Methode vor:

```
anzeigenDurchschnittswert
Transcript cr;
    show: 'Der Durchschnittswert ist: ';
    show: durchschnittswert asFloat printString
```

Bei der Einrichtung der Klasse “InWerteErfassung” mit der Instanz-Variablen “durchschnittswert” sowie den Methoden “durchschnitt” und “anzeigenDurchschnittswert” gehen wir in der nachfolgend beschriebenen Weise vor.
Um “InWerteErfassung” als Unterklasse von “WerteErfassung” einzurichten, stellen wir im Klassen-Bereich des Klassen-Hierarchie-Browser-Fensters die Klasse “WerteErfassung” – durch einen Mausklick auf den Klassennamen – als aktuelle Klasse ein. Anschließend fordern wir mittels der Menü-Option “Add Subclass...” des Menüs “Classes” das folgende Dialogfeld “Add a Subclass” an:

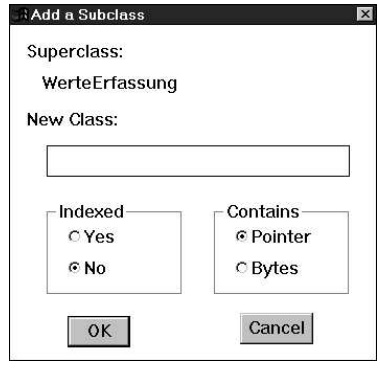


Abbildung 5.5: Einrichtung einer Unterklasse

Nachdem wir “InWerteErfassung” in das Eingabefeld “New Class:” eingetragen und das Dialogfeld über das Schaltfeld “OK” bestätigt haben, erscheint das Klassen-

Hierarchie-Browser-Fenster in der folgenden Form:

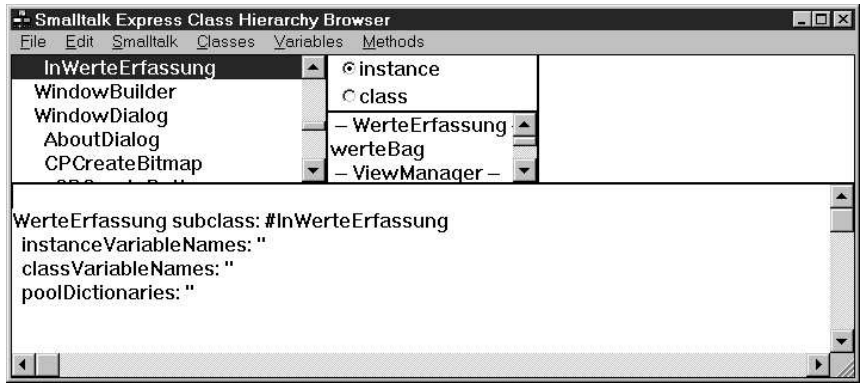


Abbildung 5.6: Vereinbarung der Instanz-Variablen

Der Editier-Bereich des Klassen-Hierarchie-Browser-Fensters enthält den folgenden Eintrag:

```
WerteErfassung subclass: #InWerteErfassung
```

Hierdurch ist "InWerteErfassung" als Name der zu vereinbarenden Unterklasse in der Form "#InWerteErfassung" als Symbol angegeben.

- Bei einem *Symbol* handelt es sich um ein *Objekt*, das als Instanz der Basis-Klasse "Symbol" zur eindeutigen Bezeichnung von Klassen, Methoden und Variablen verwendet wird.
  Der Name eines Symbols ist aus einer Folge von alphanumerischen Zeichen (Buchstaben, Ziffern oder dem Doppelpunkt ":") aufgebaut, die durch das # Symbolzeichen "`#`" mit nachfolgendem Buchstaben eingeleitet wird.
  Symbole zählen – genauso wie Zahlen, Zeichen und Zeichenketten – zu den *Literalen*, so daß Symbole unmittelbar verwendet werden können und nicht erst durch eine Instanziierung aus der Basis-Klasse "Symbol", als deren Instanzen sie anzusehen sind, eingerichtet werden müssen.

Hinweis: Symbole werden dann verwendet, wenn Klassennamen, Methodennamen oder Variablennamen innerhalb bestimmter Messages als Argumente aufgeführt werden müssen. In Sonderfällen kann ein Symbol auch aus einem oder zwei Sonderzeichen bestehen – wie z.B. bei den Symbolen "`#`," oder "`#>=`".

Nachdem wir eine Klassen-Vereinbarung in der Form

```
WerteErfassung subclass: #InWerteErfassung
  instanceVariableNames: 'durchschnittswert'
```

im Editier-Bereich eingetragen haben, bestätigen wir diese Klassen-Vereinbarung durch die Menü-Option "Save" des Menüs "File".

Hinweis: Ist bei der Vereinbarung einer Klasse keine Instanz-Variable festzulegen, so muß dem Message-Selektor "instanceVariableNames:" das Argument ' ' folgen.

Anschließend richten wir die beiden Methoden "durchschnitt" und "anzeigenDurchschnittswert" mit Hilfe der Menü-Option "New Method" des Menüs "Methods" ein.

Hinweis: Soll eine von uns bereits eingerichtete Klasse wieder gelöscht werden, so muß sichergestellt sein, daß die zu löschende Klasse keine Unterklassen hat und daß keine Instanzen dieser Klasse existieren.
Gibt es z.B. eine Instanz "InWerteErfassung11", die aus der zu löschenden Klasse instanziiert wurde, so ist die Anforderung "InWerteErfassung11 := nil" – vor der Löschung der Klasse – zur Ausführung zu bringen.
Wie wir uns alle Instanzen einer Klasse anzeigen lassen können, geben wir im Abschnitt 9.4.2 an.
Um zu prüfen, ob "InWerteErfassung11" bereits aus "InWerteErfassung" instanziiert wurde, kann z.B. die Anforderung

```
InWerteErfassung11 isMemberOf: InWerteErfassung
```

gestellt werden.

Haben wir die Klassen- und Methoden-Vereinbarung in der soeben geschilderten Form vorgenommen, so liegt die folgende Situation vor:

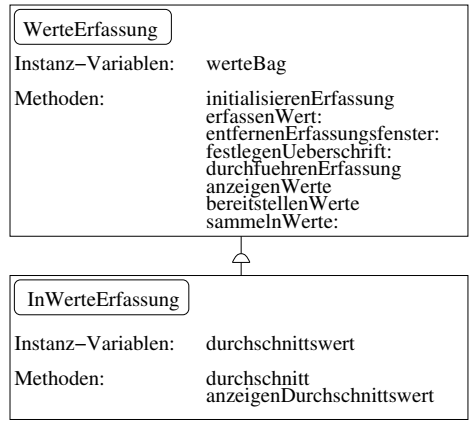


Abbildung 5.7: "InWerteErfassung" als Unterklasse von "WerteErfassung"

Während die Klasse "WerteErfassung" die Lösung von PROB-1-1 darstellt, spiegelt "InWerteErfassung" die gemeinsame Lösung von PROB-1-1 und PROB-1-2 wider.

Durch eine Instanziierung von "InWerteErfassung" wird eine Instanz eingerichtet, die die beiden Instanz-Variablen "werteBag" und "durchschnittswert" umfaßt und die sowohl die in "InWerteErfassung" als auch die in "WerteErfassung" vereinbarten Methoden kennt.

Nach der Erfassung der Punktwerte kann der Durchschnittswert errechnet und mit-

tels der Methode “anzeigenDurchschnittswert” im Transcript-Fenster ausgegeben werden.
Zur Ausführung unseres Lösungsplans können wir daher z.B. die Anforderung

```
InWerteErfassung11 := InWerteErfassung new
```

stellen und der daraufhin eingerichteten Instanz “InWerteErfassung11” anschließend die Message

```
sammelnWerte: 'Jahrgangsstufe 11'
```

in der folgenden Form zusenden:

```
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

Hinweis: Die Suche nach der Methode “sammelnWerte:”, die innerhalb der Klasse “InWerteErfassung” begonnen wird, führt in der unmittelbar übergeordneten Klasse “WerteErfassung” zum Erfolg.

Nach dem Ende der Erfassung können wir die folgende Anforderung stellen:

```
InWerteErfassung11 durchschnitt; anzeigenDurchschnittswert
```

Hinweis: Da die Methoden “durchschnitt” und “anzeigenDurchschnittswert” innerhalb der Klasse “InWerteErfassung” vereinbart sind, werden sie bereits in dieser Klasse gefunden, so daß in der Oberklasse “‘WerteErfassung” keine Suche erforderlich ist.

Durch die Ausführung der Methode “durchschnitt” wird der Durchschnittswert der erfaßten Werte der Jahrgangsstufe 11 berechnet und – innerhalb der Instanz “InWerteErfassung11” – der Instanz-Variablen “durchschnittswert” zugeordnet.
Die sich anschließende Ausführung der Methode “anzeigenDurchschnittswert” bewirkt, daß der zuvor ermittelte Durchschnittswert im Transcript-Fenster angezeigt wird.
Wollen wir uns den berechneten Durchschnittswert später nochmals anzeigen lassen, so können wir dies allein durch die Anforderung

```
InWerteErfassung11 anzeigenDurchschnittswert
```

erreichen.

Hinweis: Wird versehentlich die Methode “anzeigenDurchschnittswert” ausgeführt, ohne daß zuvor ein Durchschnittswert errechnet wurde, so erscheint das Walkback-Fenster mit der Fehlermeldung “asFloat not understood”, weil dem Empfänger-Objekt “durchschnittswert” der Message “asFloat” noch kein Zahlen-Objekt zugeordnet worden ist.
Es ist ebenfalls zu beachten, daß die *gemeinsame* Ausführung der beiden Anforderungen

```
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'.
InWerteErfassung11 durchschnitt; anzeigenDurchschnittswert
```

ebenfalls zur Anzeige des Walkback-Fensters führt, weil durch die 2. Anforderung auf die in “werteBag” gesammelten Werte zugegriffen wird. Da jedoch der Erfassungsprozeß bislang nur gestartet und im zugehörigen Erfassungsfenster noch keine Eingabe erfolgt ist, gibt es auch noch keinen verwertbaren Inhalt von “werteBag”.

5.4 Abstrakte Klassen

Durch die stufenmäßige Gliederung innerhalb der Klassen-Hierarchie ist festgelegt, in welcher Reihenfolge die einzelnen Klassen nach Methoden durchsucht werden, sofern deren Ausführung für Instanzen dieser Klassen angefordert wird.

Im Hinblick auf diese Suchreihenfolge stellt das SMALLTALK-System bestimmte Basis-Methoden, deren Ausführung sich für Instanzen verschiedener, auf derselben Hierarchiestufe angeordneter Unterklassen eignen, zusammenfassend in einer *gemeinsamen Oberklasse* zur Verfügung.

In einer derartigen Oberklasse sind z.B. alle diejenigen Basis-Methoden festgelegt, mit denen sich Vergleiche von Zahlen oder Texten durchführen lassen.

Oberklassen mit diesen Eigenschaften haben eine grundlegende Funktion innerhalb der Klassen-Hierarchie.

- Eine Klasse mit mindestens einer Unterklasse, die vornehmlich der Vereinbarung von Methoden dient, die für untergeordnete Klassen zur Verfügung gehalten werden, wird *abstrakte* Klasse genannt.
  Von dieser Art von Klassen werden in der Regel keine Instanzen erzeugt. Sie dienen vor allem zur zusammenfassenden Beschreibung gemeinsamer Eigenschaften ihrer Unterklassen.

Im Zuge der Erweiterung der Klassen-Hierarchie können abstrakte Klassen auf jeder Hierarchiestufe eingerichtet werden.

Hinweis: Ausgenommen ist lediglich die unterste Ebene, da es dann keine konkreten Objekte gäbe, die die vereinbarten Attribute und Methoden verwenden könnten.

Abstrakte Klassen werden zum Beispiel dann vereinbart, wenn Klassen zwar gewisse Gemeinsamkeiten besitzen, aber dennoch nicht in Ober- und Unterklassen gegliedert werden können. In diesem Fall ist es sinnvoll, eine neue gemeinsame Oberklasse als abstrakte Klasse festzulegen.

Dieser Sachverhalt ist z.B. in der folgenden Situation der Fall:

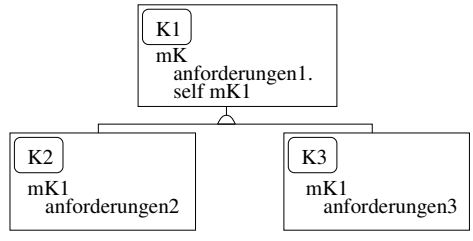


Abbildung 5.8: Abstrakte Klasse

Mit der Methode “mK” haben wir eine allgemein gehaltene Methode in der Klasse “K1” vereinbart, die von Instanzen der Unterklassen “K2” und “K3” ausgeführt werden kann. Dabei erreichen wir durch den Einsatz von “self mK1”, daß durch die Ausführung dieser Anforderung die Methode “mK1” der Klasse “K2” oder die

Methode “mK1” der Klasse “K3” ausgeführt wird.
Ist das Empfänger-Objekt der Message “mK” aus der Klasse “K2” instanziiert, so werden die Anforderungen “anforderungen1” und “anforderungen2” ausgeführt. Handelt es sich beim Empfänger-Objekt dagegen um eine Instanz der Klasse “K3”, so werden die Anforderungen “anforderungen1” und “anforderungen3” zur Ausführung gebracht.

Hinweis: In den abstrakten Klassen des Basis-Systems sind häufig Methoden-Vereinbarungen enthalten, in denen darauf hingewiesen wird, daß bestimmte Methoden innerhalb untergeordneter Klassen festgelegt sind.
Zum Beispiel ist innerhalb der abstrakten Klasse “Number” die Vereinbarung der binären Methode “+” in der Form

```
+ aNumber
∧ self implementedBySubclass
```

– unter Einsatz der Basis-Methode “implementedBySubclass” aus der Klasse “Object” – festgelegt. Sie besagt, daß innerhalb ihrer Unterklassen jeweils eine eigene Methode “+” vereinbart sein muß. Dies ist deswegen erforderlich, weil die Addition zweier Zahlen stets von der Klasse abhängig ist, zu der die jeweiligen Zahlen gehören.

Betrachten wir z.B. den Ausschnitt

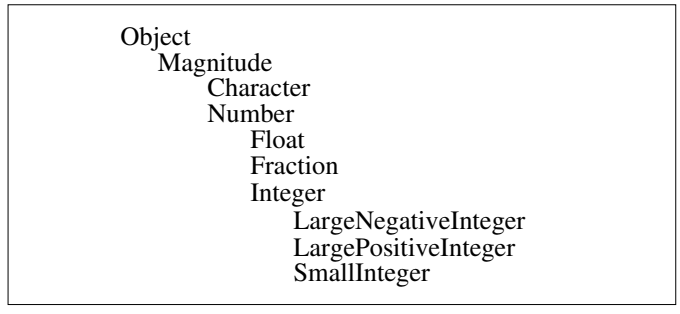


Abbildung 5.9: Ausschnitt aus der Klassen-Hierarchie

aus der Klassen-Hierarchie, so handelt es sich bei den Klassen “Magnitude”, “Number” und “Integer” um abstrakte Klassen.
Die Klasse “Magnitude” ist die Oberklasse aller Klassen, deren Instanzen untereinander verglichen und geordnet werden können. In der Klasse “Magnitude” sind Basis-Methoden festgelegt, durch deren Einsatz sich Zahlen, d.h. Instanzen aus den der Klasse “Number” untergeordneten Basis-Klassen, und Zeichen, d.h. Instanzen aus der Basis-Klasse “Character”, vergleichen lassen.

Die Basis-Klasse “Number” enthält unter anderem Methoden, die sich auf beliebige numerische Größen anwenden lassen, während es sich bei der Klasse “Integer” um die direkte Oberklasse ganzzahliger Zahlenwerte handelt.

5.5 Grundlegende Basis-Klassen für Zahlen und Zeichen

5.5.1 Die Basis-Klasse "Number"

Zahlen zählen zu den Literalen, so daß sie aus den jeweiligen Basis-Klassen durch "Hinschreiben" instanziiert werden.
Innerhalb der abstrakten Klasse "Number" und deren Unterklassen lassen sich folgende unterschiedliche Formen von Zahlen unterscheiden:

- *Fließkommazahlen* als Instanzen der Klasse "Float",
- *ganze Zahlen* als Instanzen der Klassen "Integer", "SmallInteger", "LargePositiveInteger" bzw. "LargeNegativeInteger" und
- *Brüche* als Instanzen der Klasse "Fraction".

Zur Verarbeitung von Zahlen sind innerhalb der Klasse "Number" und deren Unterklassen Methoden vereinbart, die sich z.B. durch die folgenden Messages abrufen lassen:

- *binäre Messages*
  für arithmetische Operationen – wie z.B. "+" (Addition), "−" (Subtraktion), "∗" (Multiplikation), "/" (Division), "//" (ganzzahliges Ergebnis einer ganzzahligen Division) und "\\" (Rest einer ganzzahligen Division) – sowie die Vergleichs-Operationen "="(gleich), "∼=" (ungleich), ">=" (größer oder gleich), "<=" (kleiner oder gleich), "<" (echt kleiner) und ">" (echt größer).
- *unäre Messages*
  für Prüfungen – wie z.B. "even", "odd", "positive", "negative", "isFloat", "isInteger", "isFraction" –, für Rundungen – wie z.B. "rounded", "floor", "ceiling" und "truncated" –, für Konvertierungen – wie z.B. "asFloat", "asInteger" und "asCharacter" – sowie für arithmetische Operationen – wie z.B. "negated" (Negations-Operator), "sqrt" (Quadratwurzel), "squared" (Quadrieren), "abs" (Absolutwert), "sin" (trigonometrische Sinusfunktion) und "ln" (Logarithmusfunktion).
- *Keyword-Messages*
  für arithmetische Operationen – wie z.B. "quo:" (ganzzahlige Division), "rem:" (Rest einer ganzzahligen Division), "raisedTo:" (Potenzieren), "between:and:" (Intervall-Prüfung), "min:" (Minimum), "max:" (Maximum) und die logarithmische Funktion "log:".

Hinweis: Durch die Vereinbarung gleichnamiger Methoden in der Klasse "Number" und deren Unterklassen, sowie dem Einsatz der Messages zur Zahlenkonvertierung "asInteger" und "asFloat" ist sichergestellt, daß die aufgeführten Messages von allen Zahlen ausgeführt werden können.

Verknüpfen wir zwei Operanden durch den Einsatz dieser Messages, so gilt grundsätzlich:

- Handelt es sich bei den Operanden um Instanzen *derselben* Klasse, so ist das Ergebnis-Objekt ebenfalls aus dieser Klasse.
  Sind beide Operanden aus *verschiedenen* Klassen instanziiert, so wird versucht, sie – ausgehend von "SmallInteger" über "LargePositiveInteger ("LargeNegativeInteger") zu "Float" bzw. ausgehend von "Fraction" zu "Float" – (in dieser Reihenfolge) zu konvertieren, sodaß beide Operanden zu derselben Klasse gehören.

Stellen wir z.B. eine Anforderung der Form

```
5 - 8.0
```

so erhalten wir als Ergebnis-Objekt den Wert "−3.0". Bei der Ausführung dieser Anforderung wird das Empfänger-Objekt der binären Message "−" (dies ist eine Instanz der Klasse "SmallInteger") in eine Instanz der Klasse "Float" konvertiert, so daß beide Operanden der gleichen Klasse angehören. Erst anschließend wird die Subtraktion durchgeführt.

Hinweis: Dabei ist zu beachten, daß eine Anforderung der Form "5 −8.0" zu der Fehlermeldung "should be selector" führt. Dies liegt daran, daß das dem Wert "8.0" unmittelbar vorangestellte Minuszeichen nicht als Symbol zur Kennzeichnung einer binären Message für die Subtraktion, sondern als Zeichen zur Kennzeichnung eines negativen Zahlenwertes aufgefaßt wird.

Bevor wir die Wirkung ausgewählter Methoden der Unterklassen von "Number" beschreiben, stellen wir in der folgenden Tabelle Beispiele für Messages zur ganzzahligen Division und zur Rundung von Zahlen vor:

Anforderung	Ergebnis-Objekt	Kommentar
`9.5 // 2`	`4`	(Modulo-Funktion) Ganzzahlige Division, Quotient wird gerundet zur nächst kleineren ganzen Zahl
`-9.5 // 2`	`-5`	
`9.5 \\ 2`	`1`	Rest einer ganzzahligen Division mit "//", positiver (negativer) Rest wird gerundet zur nächst kleineren (größeren) ganzen Zahl
`-9.5 \\ 2`	`0`	
`9.5 quo: 2`	`4`	Ganzzahlige Division, positiver (negativer) Quotient wird gerundet zur nächst kleineren (größeren) ganzen Zahl
`-9.5 quo: 2`	`-4`	
`9.5 rem: 2`	`1`	Rest einer ganzzahligen Division mit "rem:", positiver (negativer) Rest wird gerundet zur nächst kleineren (größeren) ganzen Zahl
`-9.5 rem: 2`	`-1`	
`9.7 floor`	`9`	Abrunden zur nächst kleineren ganzen Zahl
`-9.7 floor`	`-10`	
`9.7 ceiling`	`10`	Aufrunden zu nächst größeren ganzen Zahl
`-9.7 ceiling`	`-9`	
`9.5 rounded`	`10`	Kaufmännisches Runden zur nächsten ganzen Zahl (Auf- oder Abrunden zur nächsten ganzen Zahl)
`-9.3 rounded`	`-9`	
`9.3 truncated`	`9`	Abschneiden der Nachkommastellen
`-9.3 truncated`	`-9`	

5.5.2 Die Basis-Klasse "Integer"

Bei Instanzen unterhalb der abstrakten Klasse "Integer" handelt es sich um ganze Zahlen. In Abhängigkeit vom jeweiligen Zahlenwert lassen sich die Klassen

- "SmallInteger",
- "LargeNegativeInteger" und
- "LargePositiveInteger"

unterscheiden.
Während durch Instanzen der Klasse "LargePositiveInteger" ("LargeNegativeInteger") beliebig große positive (negative) ganze Zahlen dargestellt werden, beschränken sich Instanzen der Klasse "SmallInteger" auf den Wertebereich zwischen "$-2^{15}+1$ (-32767)" und "$2^{15}-1$ (32767)".

Hinweis: Beliebig große ganze Zahlen können wir z.B. durch den Einsatz der Message "factorial" in der Form

```
100 factorial
```

erzeugen. Wir erhalten als Ergebnis-Objekt die Fakultät der Zahl "100". Dies ist eine ganze Zahl mit 158 Stellen. Die Fakultät einer ganzen Zahl "n > 0" wird durch die Multiplikation "n * (n−1) * (n−2) ... * 2 * 1" berechnet.

Die größte ganze Zahl der Klasse "SmallInteger" kann durch die Anforderungen

```
| ganzZahl |
ganzZahl := 0.
[ganzZahl class == SmallInteger]
   whileTrue: [ganzZahl := ganzZahl + 1].
ganzZahl - 1
```

ermittelt werden. Es resultiert der Wert "32767".

Hinweis: Durch den Einsatz von "class" wird diejenige Klasse bestimmt, aus der das Empfänger-Objekt dieser Message instanziiert ist. Anschließend wird durch "==" geprüft, ob die ermittelte Klasse mit der Klasse "SmallInteger" identisch ist (siehe auch die Abschnitte 6.3.3 und 8.1).

Für Instanzen der Klasse "Integer" lassen sich unter anderem die folgenden Methoden einsetzen:

- **"asCharacter"**:
  Als Ergebnis-Objekt resultiert ein Zeichen (Instanz der Klasse "Character"), dessen ASCII-Kode dem Empfänger-Objekt entspricht (siehe unten).
- **"asFloat"**:
  Die als Empfänger-Objekt der Message "asFloat" aufgeführte ganze Zahl wird in eine Fließkommazahl konvertiert (siehe unten).

- **“gcd:”**:
  Als Ergebnis-Objekt dieser Message wird der größte gemeinsame Teiler zwischen dem ganzzahligen Empfänger-Objekt und der als Argument von “gcd:” angegebenen ganzen Zahl ermittelt.

- **“=”, “∼=”**:
  Durch den Einsatz der Message “=” (“∼=”) werden zwei Zahlenwerte auf Gleichheit (Ungleichheit) geprüft, so daß als Ergebnis-Objekt eine der Pseudovariablen “true” oder “false” resultiert (siehe Abschnitt 6.3.1).

5.5.3 Die Basis-Klasse “Float”

Durch Instanziierungen aus der Klasse “Float” lassen sich nicht-ganzzahlige Zahlen in Form von *Fließkommazahlen* erzeugen. Beim “Hinschreiben” einer Fließkommazahl sind die Nachkommastellen durch die Eingabe eines Punktes “.” von den Vorkommastellen abzutrennen.
Für Instanzen der Klasse “Float” können z.B. die folgenden Methoden zur Rundung eingesetzt werden:

- **“floor”, “ceiling”**:
  Mittels der Message “floor” (“ceiling”) wird das Empfänger-Objekt zur nächst kleineren ganzen Zahl abgerundet (aufgerundet).

- **“truncated”**:
  Als Ergebnis-Objekt ergibt sich aus der als Empfänger-Objekt aufgeführten Fließkommazahl diejenige ganze Zahl, die durch Abschneiden der Nachkommastellen ermittelt wird.

- **“rounded”**:
  Als Ergebnis-Objekt wird die ganze Zahl erhalten, die sich aus dem kaufmännischen Runden (Auf- oder Abrunden) des Empfänger-Objektes zur nächsten ganzen Zahl ergibt.

Innerhalb der Klasse “Float” sind mathematische Funktionen wie z.B. die Methode “ln” zur Berechnung des Logarithmus sowie “sin”, “cos” und “tan” zur Berechnung trigonometrischer Funktionen abrufbar.

5.5.4 Die Basis-Klasse “Fraction”

Zur Darstellung nicht-ganzzahliger Zahlenwerte stellt das SMALLTALK-System – neben den Fließkommazahlen – eine weitere Möglichkeit in Form von Instanzen der Klasse “Fraction” zur Verfügung. Sie dienen zur Darstellung von *Brüchen* mit ganzzahligem Zähler (“numerator”) und Nenner (“denominator”).

Hinweis: Jedes Objekt, das aus der Klasse “Fraction” instanziiert ist, besitzt eine Instanz-Variable namens “numerator” zur Speicherung des Zählers und eine Instanz-Variable namens “denominator” zur Speicherung des Nenners.

Werden die arithmetischen Operationen “+” (Addition), “−” (Subtraktion), “∗” (Multiplikation) und “/” (Division) mit Operanden durchgeführt, die die Form von

Brüchen oder ganzen Zahlen besitzen, so resultiert als Ergebnis-Objekt wiederum ein Bruch, sofern sich das Ergebnis-Objekt *nicht* durch Kürzen zu einer ganzen Zahl vereinfachen läßt.

Hinweis: Zum Beispiel resultiert als Ergebnis-Objekt einer Division in Form von "22/7" eine Instanz der Klasse "Fraction". Anders ist dies bei "22.0/7". Hier wird als Ergebnis-Objekt eine Instanz der Klasse "Float" in Form des Näherungswertes "3.14285714" erhalten. Dies liegt daran, daß zur Auswertung von "22.0/7" die binäre Message "/" der Klasse "Float" ausgeführt wird. Dabei wird die ganze Zahl "7" in die Fließkommazahl "7.0" konvertiert.

Die Methode zur Addition zweier Brüche ist innerhalb der Klasse "Fraction" in der folgenden Form vereinbart:

```
+ aNumber
∧  ((numerator * aNumber denominator) +
    (denominator * aNumber numerator)) /
    (denominator * aNumber denominator)
```

Hinweis: Bei der Ausführung dieser Methode liefert die unäre Message "numerator" ("denominator") innerhalb der Anforderung "aNumber numerator" ("aNumber denominator") den Zähler bzw. den Nenner des durch den Platzhalter "aNumber" gekennzeichneten Empfänger-Objektes.
Das Empfänger-Objekt der binären Messages "∗" ergibt sich aus den Ergebnis-Objekten der primären Messages "numerator" und "denominator". Aus der Zustellung dieser primären Messages resultieren die Werte, die den Instanz-Variablen "numerator" und "denominator" des Empfänger-Objektes von "+ aNumber" zugeordnet sind.

Wird z.B. die Addition "3/8 + (3/8)" ausgeführt, so ergibt sich

```
+ (3/8)
∧  ((3 * (3/8) denominator) +
     (8 * (3/8) numerator)) /
    (8 * (3/8) denominator)
```

und damit

```
∧  ((3 * 8) +
     (8 * 3)) /
    (8 * 8)
```

und folglich:

```
∧  48 /
    64
```

Da es sich beim Empfänger-Objekt der zuletzt auszuführenden Anforderung "48/64"

um eine ganze Zahl handelt, wird die Division der ganzen Zahlen “48” und “64” durch die Methode “/” der Klasse “Integer” ausgeführt, so daß als Ergebnis-Objekt der Bruch “3/4” erhalten wird.

Hinweis: Innerhalb der Message “/” der Klasse “Integer” werden – sofern es sich beim Divisor nicht um eine Instanz der Klasse “Float” handelt – die Message “gcd:” zur Bestimmung des größten gemeinsamen Teilers von Zähler und Nenner und die ganzzahlige Division durch den Einsatz der Message “//” ausgeführt, so daß der Bruch “48/64” zu “3/4” gekürzt wird.
Wird bei “3/8 + (3/8)” der zweite Operand nicht geklammert, so erhalten wir den Bruch “27/64”. Dies liegt daran, daß – entsprechend der Auswertungsreihenfolge von “links nach rechts” – zunächst “3/8 + 3” berechnet und das Zwischenergebnis “27/8” anschließend durch “8” dividiert wird.

5.5.5 Die Basis-Klasse “Character”

Die Basis-Klasse “Character” ist eine direkte Unterklasse der abstrakten Klasse “Magnitude”. Instanzen der Klasse “Character” sind einzelne *Zeichen*. Sie zählen ebenso wie die Zahlen zu den Literalen.
Entsprechend dem ASCII-Kode gibt es 256 verschiedene Zeichen, die mit den ganzzahligen ASCII-Kodewerten “0”, “1”, ..., “255” korrespondieren und als unterschiedliche Instanzen aus der Klasse “Character” instanziierbar sind.
Die darstellbaren Zeichen wie z.B. “a”, “1”, “F”, “:” oder “␣” lassen sich in der folgenden Form angeben:

- $<zeichen>

Darstellbare Zeichen sowie nicht-darstellbare Zeichen wie z.B. das Tabulator-Zeichen können unter Einsatz der Basis-Methode “asCharacter” erzeugt werden.

- **“asCharacter”:**
  Als Empfänger-Objekt der Message “asCharacter” muß eine Zahl mit einem ganzzahligen Wert aufgeführt werden, der zwischen “0” und “255” liegt. Als Ergebnis-Objekt resultiert dasjenige Zeichen, dessen zugehöriger ASCII-Kodewert gleich dieser Zahl ist.

  Hinweis: Der ASCII-Kodewert der Ziffern liegt zwischen “48” und “57”.

So ergibt sich z.B. das Zeichen “$C” als Ergebnis-Objekt der Message “67 asCharacter” oder das Tabulator-Zeichen als Ergebnis-Objekt der Message “9 asCharacter”.

Hinweis: Aus dem Zeichen “$C” läßt sich durch die Message “$C asInteger” die ganze Zahl “67” als Ergebnis-Objekt ermitteln.
Zur Wandlung von Zeichen können z.B. ferner die Basis-Methoden “asLowerCase” (Umwandlung in Kleinbuchstaben), “asUppercase” (Umwandlung in Großbuchstaben) und “asString” (Darstellung als String mit einem Zeichen) eingesetzt werden.

Für Instanzen der Klasse “Character” stehen neben der Methode “=”, mit der die Gleichheit zweier Zeichen geprüft werden kann, zusätzlich die Basis-Methoden “∼=” (ungleich), “>=” (größer oder gleich), “<=” (kleiner oder gleich), “<” (echt kleiner) und “>” (echt größer) zur Verfügung, so daß die Sortierfolge von Zeichen geprüft werden kann. Beim Einsatz dieser Methoden werden beim Vergleich zweier Zeichen die mit den jeweiligen Zeichen korrespondierenden ASCII-Kodewerte verglichen.

Kapitel 6

Einsatz von Basis-Methoden

Um die von uns entwickelten Lösungspläne "schnellstmöglich" umsetzen zu können, haben wir für die jeweils benötigten Anforderungen nur in bestimmten Fällen erläutert, welche Wirkung mit der Ausführung einer verwendeten Basis-Methode verbunden ist.

Hinweis: Zum Beispiel haben wir die Wirkung der Basis-Methoden "new", "cr", "show:", "do:", "size", "asInteger", "asFloat" und "printString" ausführlicher beschrieben.

Im folgenden werden wir Begründungen dafür nachliefern, daß die von uns entwickelten Methoden tatsächlich die Dienstleistungen erbringen, die wir im Rahmen unserer Handlungsvorgaben zur Lösung von PROB-1-1 und PROB-1-2 innerhalb der Klasse "WerteErfassung" vereinbart haben.

6.1 Ausgewählte Methoden zur Bearbeitung von Fenster-Bausteinen

6.1.1 Die Methoden "paneNamed:" und "contents"

Zum Beispiel haben wir die im Abschnitt 1.3.5 in der Form

- *"erfassenWert"*
  - Übertragung eines Wertes vom Eingabefeld nach "werteBag"
  - Eintragung von Leerzeichen in das Eingabefeld
  - Plazierung des Cursors auf das Eingabefeld

vorgeschlagene Handlung "erfassenWert" im Abschnitt 2.4 in der folgenden Form in die Methode "erfassenWert:" umgeformt:

```
erfassenWert: aPane
werteBag add: (self paneNamed: 'eingabeFeld') contents.
(self paneNamed: 'eingabeFeld') contents: ''.
(self paneNamed: 'eingabeFeld') setFocus
```

Hinweis: In der Methode "erfassenWert:" tritt der Platzhalter "aPane" nicht innerhalb der Anforderungen auf. Er muß aus formalen Gründen hinter dem Methoden-Selektor "erfassenWert:" angegeben werden, da er die Verknüpfung zwischen dem Schaltfeld "erfasse"

und der Methode “erfassenWert:” herstellt.

In der ersten Anforderung

```
werteBag add: (self paneNamed: 'eingabeFeld') contents
```

wird dem Empfänger-Objekt “werteBag”, das zu derjenigen Instanz von “WerteErfassung” gehört, an die die Message “erfassenWert:” gerichtet wird, die verschachtelte Message

```
add: (self paneNamed: 'eingabeFeld') contents
```

übermittelt. Diese Message besteht aus der Keyword-Message “add:”, der als Argument aufgeführten eingeklammerten Anforderung der Form “(self paneNamed: 'eingabeFeld')” und der unären Message “contents”.
Im Hinblick auf die Ausführungsreihenfolge ist zu erkennen, daß die Message

```
(self paneNamed: 'eingabeFeld')
```

zuerst ausgeführt wird. Durch “self” ist festgelegt, daß die Message

```
paneNamed: 'eingabeFeld'
```

an das Objekt gerichtet wird, dem die Message “erfassenWert:” zugestellt wird, d.h. einer Instanz der Klasse “WerteErfassung”. Da diese Klasse eine Unterklasse der Basis-Klasse “ViewManager” ist, erbt jede Instanz von “WerteErfassung” die Eigenschaften der Klasse “ViewManager” und kennt auch die Methoden, die in “ViewManager” vereinbart sind. Da in der Klasse “ViewManager” die Methode “paneNamed:” festgelegt ist, kann sie folglich von einer Instanz von “WerteErfassung” ausgeführt werden.

- **“paneNamed:”**:
  Bei der Ausführung der Keyword-Message

  ```
  paneNamed: 'eingabeFeld'
  ```

  mit dem Argument “ 'eingabeFeld' ” wird als Ergebnis-Objekt das Eingabefeld des Erfassungsfensters ermittelt. Dieses Eingabefeld wurde – bei der Ausführung der Methode “initialisierenErfassung” – durch eine Instanziierung aus der Basisklasse “EntryField” eingerichtet. Es verkörpert daher eine Instanz der Basis-Klasse “EntryField”.

  Hinweis: Im Abschnitt 2.2 haben wir den Namen “eingabeFeld” zur Kennzeichnung dieses Eingabefeldes festgelegt, als wir – unter Einsatz des Werkzeugs “WindowBuilder Pro” – den Aufbau unseres Erfassungsfensters bestimmt haben.

Da bei

```
add: (self paneNamed: 'eingabeFeld') contents
```

zuerst

```
(self paneNamed: 'eingabeFeld') contents
```

zur Ausführung gelangt, wird das Eingabefeld des Erfassungsfensters zum Empfänger-Objekt der unären Message "contents".
Als Instanz der Basis-Klasse "EntryField" kennt das Eingabefeld die Methode "contents", da diese Methode in dieser Basis-Klasse vereinbart ist.

- **"contents"**:
  Bei der Ausführung von "contents" wird als Ergebnis-Objekt dasjenige Zeichenketten-Objekt ermittelt, das gleich dem unmittelbar zuvor in das Eingabefeld – mittels der Tastatur – übertragenen Wert ist.

Wegen der Anforderung

```
werteBag add: (self paneNamed: 'eingabeFeld') contents
```

ist dieses Zeichenketten-Objekt das Argument der Keyword-Methode "add:".

6.1.2 Die Methoden "contents:" und "setFocus"

Da das Objekt "werteBag" aus der Basis-Klasse "Bag" instanziiert wurde, kennt dieses Objekt die Basis-Methoden, die innerhalb der Basis-Klasse "Bag" vereinbart sind.
Ein Bag läßt sich mit der Methode "add:" füllen, indem als Empfänger-Objekt der Bag und als Argument dasjenige Objekt aufgeführt wird, das dem Bag hinzugefügt werden soll.
Daher wird durch die Anforderung

```
werteBag add: (self paneNamed: 'eingabeFeld') contents
```

festgelegt, daß die geforderte Handlung "Übertragung eines Wertes vom Eingabefeld nach "werteBag" " durchgeführt wird.

Entsprechend wird die Handlung "Eintragung von Leerzeichen in das Eingabefeld" durch die zweite Anforderung der Methode "erfassenWert:"

```
(self paneNamed: 'eingabeFeld') contents ' '
```

umgesetzt, weil durch

```
(self paneNamed: 'eingabeFeld')
```

das Eingabefeld des Erfassungsfensters als Empfänger-Objekt der Keyword-Message "contents:" ermittelt und innerhalb dieser Anforderung das Argument " .''. '' zur Kennzeichnung eines Leertextes aufgeführt wird. Da "contents:" zu den Basis-Methoden der Basis-Klasse "EntryField" zählt, kennt das Eingabefeld diese Methode und füllt seinen Inhalt – im Hinblick auf die Anzeige innerhalb des Erfassungsfensters – mit Leerzeichen.

- **“contents:”**:
  Bei der Ausführung der Basis-Methode “contents:” wird dem Empfänger-Objekt der Message “contents:” das angegebene Argument in Form einer Zeichenkette zugeordnet.

Entsprechend den vorausgegangenen Erörterungen ist unmittelbar einsichtig, daß innerhalb der dritten Anforderung

```
(self paneNamed: 'eingabeFeld') setFocus
```

der Methode “erfassenWert:” das Empfänger-Objekt der unären Message “setFocus” durch das Eingabefeld des Erfassungsfensters verkörpert wird.
Da die Basis-Klasse “EntryField” eine Unterklasse der Basis-Klasse “Window” ist und in dieser Klasse die Basis-Methode “setFocus” vereinbart ist, kann das Eingabefeld die unäre Message “setFocus” ausführen.

- **“setFocus”**:
  Durch die Ausführung der Basis-Methode “setFocus” wird das Eingabefeld aktiviert, indem der Cursor auf die durch das Empfänger-Objekt der Message “setFocus” bestimmte Bildschirmposition plaziert wird.

Folglich wird der Cursor im Erfassungsfenster auf das Eingabefeld positioniert, so daß die Handlung “Plazierung des Cursors auf das Eingabefeld” umgesetzt wird.

6.1.3 Die Methode “close”

Die von uns im Abschnitt 1.3.5 in der Form

- *“entfernenErfassungsfenster”*
  – Entfernung des Erfassungsfensters vom Bildschirm

vorgeschlagene Handlung “entfernenErfassungsfenster” haben wir im Abschnitt 2.4 wie folgt in die Methode “entfernenErfassungsfenster:” umgeformt:

```
entfernenErfassungsfenster: aPane
self close
```

Diese Vereinbarung basiert auf dem folgenden Sachverhalt:

- **“close”**:
  Durch die Ausführung der Basis-Methode “close” wird das Bildschirmfenster, das dem Empfänger-Objekt der Message “close” zugeordnet ist, vom Bildschirm entfernt.

Durch den Einsatz der Pseudovariablen “self” wird die Instanz der Klasse “WerteErfassung”, die als Empfänger-Objekt der Message “entfernenErfassungsfenster:” dient, zum Empfänger-Objekt von “close”. Daher wird die Methode “close” innerhalb der Basis-Klasse “ViewManager” als Basis-Methode identifiziert, deren Ausführung zum gewünschten Effekt führt.

6.1.4 Die Methode "openWindow"

Im Abschnitt 1.3.5 haben wir für unseren Lösungsplan die Handlung "durchfuehrenErfassung" in der folgenden Form vorgesehen:

- *"durchfuehrenErfassung"*
  - Anzeige und Eröffnung des Erfassungsfensters am Bildschirm
  - Festlegung, daß der Cursor auf das Eingabefeld plaziert wird

Die zugehörigen Anforderungen wurden von uns im Abschnitt 2.5 durch die Methode "durchfuehrenErfassung" in der folgenden Form festgelegt:

```
durchfuehrenErfassung
self openWindow.
(self paneNamed: 'eingabeFeld') setFocus
```

Durch die erste Anforderung wird die Basis-Methode "openWindow" zur Ausführung gebracht.

- **"openWindow"**:
  Die Basis-Methode "openWindow" ist Bestandteil der Basis-Klasse "ViewManager". Durch sie wird das zuvor für die Anzeige vorbereitete Fenster, das durch das Empfänger-Objekt der Message "openWindow" gekennzeichnet ist, am Bildschirm angezeigt.

Entsprechend der Vereinbarung mittels der Pseudovariablen "self" wird die Instanz der Klasse "WerteErfassung", die als Empfänger-Objekt der Message "entfernenErfassungsfenster:" dient, zum Empfänger-Objekt von "openWindow". Folglich wird das durch die Instanz von "WerteErfassung" gekennzeichnete Fenster angezeigt.
Damit der Cursor im Eingabefeld des Erfassungsfensters positioniert wird, ist die Anforderung

```
(self paneNamed: 'eingabeFeld') setFocus
```

als zweite Anforderung innerhalb der Methode "durchfuehrenErfassung" festgelegt.

6.1.5 Die Methode "labelWithoutPrefix:"

Die von uns im Abschnitt 1.3.5 in der Form

- *"festlegenUeberschrift"*
  - Festlegung der neuen Titel-Zeile des Erfassungsfensters

vorgesehene Handlung "festlegenUeberschrift" haben wir im Abschnitt 2.5 wie folgt in die Methode "festlegenUeberschrift:" umgeformt:

```
festlegenUeberschrift: aString
self labelWithoutPrefix: aString
```

Diese Vereinbarung führt zur erwünschten Wirkung, weil folgendes gilt:

- **“labelWithoutPrefix:”**:
  Durch die Ausführung der Basis-Methode “labelWithoutPrefix:” wird derjenige Text in die Titel-Zeile eines Fensters eingetragen, der als Argument der Message “labelWithoutPrefix:” aufgeführt ist.

Durch den Einsatz der Pseudovariablen “self” wird die Instanz derjenigen Klasse, die als Empfänger-Objekt der Message “festlegenUeberschrift:” aufgeführt ist, zum Empfänger-Objekt von “labelWithoutPrefix:”. Daher bewirkt die Methode “labelWithoutPrefix:”, sofern sie von einer Instanz der Klasse “WerteErfassung” ausgeführt wird, daß die als Argument verwendete Zeichenkette innerhalb der Titel-Zeile des Erfassungsfensters erscheint.

6.2 Einsatz von Blöcken

6.2.1 Das Objekt “Block”

Zur wiederholten Ausführung von Anforderungen haben wir im Abschnitt 3.4 die Basis-Methode “do:” kennengelernt, bei deren Einsatz ein Block mit einem Block-Parameter als Argument aufgeführt werden muß.

- Jeder Block ist eine Instanz der Basis-Klasse “HomeContext” und somit ein Objekt.

Gemäß der im Abschnitt 3.4 angegebenen Beschreibung wird ein Block durch die Klammer “[” eingeleitet und durch die Klammer “]” abgeschlossen. Er kann Block-Parameter und eine oder mehrere Anforderungen – als Block-Anforderungen – enthalten, so daß seine allgemeine Form wie folgt angegeben werden kann:

```
[:parameter1 :parameter2 ... |
   anforderung1.
   anforderung2.
   ...
]
```

Hinweis: Mehrere Block-Anforderungen werden jeweils durch einen Punkt “.” voneinander getrennt.

- Enthält ein Block keine Block-Anforderungen, so wird er als *leerer* Block bezeichnet.

Es lassen sich die folgenden Arten von Blöcken unterscheiden:

- Blöcke ohne Block-Parameter wie z.B.:
  “[Transcript cr; show: ’Jahrgangsstufe’; show: ’11’]”

- Blöcke mit Block-Parametern wie z.B.:
  "[:nummer | Transcript cr; show: 'Jahrgangsstufe'; show: nummer]"

Hinweis: Anstelle der Kaskade "show: 'Jahrgangsstufe'; show: '11' " können wir auch die Basis-Methode "," aus der Basis-Klasse "FixedCollection", die zur Reihung von Zeichenketten eingesetzt werden kann, in der folgenden Form verwenden: "show: 'Jahrgangsstufe', '11' " (siehe Kapitel 9).

Da es sich bei einem Block um ein Objekt handelt, kann ein Block auch einer Variablen zugeordnet werden, wie dies z.B. bei der Zuweisung

```
VarBlock := [Transcript cr; show: 'Jahrgangsstufe '; show: '11']
```

der Fall ist.

6.2.2 Ausführung eines Blockes

Durch den Einsatz der Basis-Methoden "value", "value:" und "value:value:", die in der Basis-Klasse "Context" – einer Oberklasse der Basis-Klasse "HomeContext" – vereinbart sind, können die in einem Block enthaltenen Block-Anforderungen zur Ausführung gebracht werden.

- **"value"**, **"value:"**, **"value:value:"**:
  Durch den Einsatz der Basis-Methode "value" läßt sich ein Block ausführen, der keinen Block-Parameter besitzt und als Empfänger-Objekt der Message "value" verwendet wird.
  Enthält ein Block einen oder zwei Block-Parameter, so sind die Keyword-Messages "value:" bzw. "value:value:" einzusetzen und als Argumente diejenigen Objekte anzugeben, die für die Block-Parameter bei der Ausführung der Block-Anforderungen eingesetzt werden sollen.

  Hinweis: Beim Einsatz der Keyword-Message "value:value:" werden die beiden Argumente den vereinbarten Block-Parametern gemäß ihrer Reihenfolge zugeordnet.

  Als Ergebnis-Objekt jeder dieser Messages ergibt sich dasjenige Objekt, das als Ergebnis-Objekt aus der zuletzt ausgeführten Block-Anforderung resultiert.

Hinweis: Wird einem *leeren* Block der Form "[ ]" die Nachricht "value" zugestellt, so ist als Ergebnis-Objekt der Block-Ausführung die Pseudovariable "nil" festgelegt. Mit dem Begriff "logischer Block" werden Blöcke bezeichnet, deren Ergebnis-Objekt gleich einer der beiden Pseudovariablen "true" bzw. "false" ist (siehe Abschnitt 6.3). Blöcke mit zwei Block-Parametern werden *binäre* Blöcke genannt.
Ist es notwendig, einen Block mit mehr als zwei Block-Parametern zur Ausführung zu bringen, so können sämtliche Argumente als Bestandteil eines geeigneten Sammlers festgelegt und dieser Sammler als Argument der Message "value:" dem Block-Parameter zugeordnet werden.

Sofern z.B. ein Block, der der globalen Variablen "VarBlock" durch die Zuweisung

```
VarBlock := [Transcript cr; show: 'Jahrgangsstufe ', '11']
```

zugeordnet ist, zur Ausführung gelangen soll, ist die Anforderung

```
VarBlock value
```

zu stellen. Dadurch erhalten wir im Transcript-Fenster den Text "Jahrgangsstufe 11" angezeigt.
Um den Block

```
[:nummer|Transcript cr; show: 'Jahrgangsstufe '; show: nummer]
```

ausführen und dabei für den Platzhalter "nummer" das Zeichenketten-Objekt '12' einsetzen zu lassen, können wir z.B. die Anforderungen

```
VarBlock:=[:nummer|Transcript cr;show:'Jahrgangsstufe';show:nummer].
VarBlock value: '12'
```

ausführen lassen. Anschließend erhalten wir im Transcript-Fenster den Text "Jahrgangsstufe 12" angezeigt.
Es erfolgt die Ausgabe der Texte "Jahrgangsstufe 12" und "Jahrgangsstufe 13", sofern wir im Workspace-Fenster die folgende Anforderung stellen:

```
VarBlock value: '12'; value: '13'
```

Hinweis: Innerhalb eines Blockes können wir einer zuletzt auszuführenden Block-Anforderung das Return-Zeichen "∧" voranstellen. Wird innerhalb einer Methode ein derartiger Block ausgeführt, so wird durch die Ausführung der zugehörigen Anforderung sowohl die Block-Ausführung als auch die Ausführung der jeweiligen Methode beendet. Als Ergebnis-Objekt der zugehörigen Message resultiert dann das Ergebnis-Objekt der durch "∧" eingeleiteten Block-Anforderung.

6.2.3 Einsatz eines Blockes zur Einrichtung eines Sammlers

Neben der Basis-Methode "do:" gibt es weitere Basis-Methoden, bei deren Einsatz ein Block als Argument verwendet werden muß. Zu diesen Methoden zählt z.B. die Basis-Methode "collect:", mit der sich ein Sammler einrichten läßt, dessen Inhalt durch die Ausführung von Block-Anforderungen festgelegt wird.

- **"collect:"**:
  Beim Einsatz der Keyword-Message "collect:" muß ein Block mit einem Block-Parameter als Argument und ein Sammler als Empfänger-Objekt angegeben werden.
  Aus der Ausführung der Basis-Methode "collect:" resultiert ein neuer Sammler als Ergebnis-Objekt, der aus derselben Klasse instanziiert ist wie das Empfänger-Objekt von "collect:". Er wird mit denjenigen Objekten gefüllt, die als Ergebnis-Objekte aus der wiederholten Ausführung des Blockes resultieren, der innerhalb der Message "collect:" als Argument aufgeführt ist.
  Bei der Ausführung der Block-Anforderungen wird jedesmal ein anderes Objekt aus dem Sammler, der als Empfänger-Objekt der Message "collect:" aufgeführt ist, für den Block-Parameter eingesetzt.

Zum Beispiel ergibt sich aus der Ausführung von

```
WerteErfassung11 bereitstellenWerte
   collect: [:einObjekt|einObjekt asInteger]
```

ein Sammler in Form eines Bags, weil es sich beim Empfänger-Objekt von "collect:" um denjenigen Bag handelt, der der Instanz-Variablen "'werteBag" zugeordnet ist.
Jedes Objekt des Bags, der aus der Ausführung von

```
WerteErfassung11 bereitstellenWerte
```

resultiert, wird für den Block-Parameter "einObjekt" eingesetzt, so daß sich durch die Ausführung der Message "einObjekt asInteger" der ganzzahlige Wert der Zeichenkette ergibt. Diese Zahl wird in den Sammler aufgenommen, der als Ergebnis-Objekt erhalten wird.
Die Ausführung der Methode "collect:" ist dann beendet, wenn alle Zeichenketten des durch die Anforderung "WerteErfassung11 bereitstellenWerte" bestimmten Sammlers bearbeitet wurden.
Daß die Basis-Methode "collect:" in der angegebenen Form wirkt, liegt an ihrer Vereinbarung, die innerhalb der Basis-Klasse "Collection" in der folgenden Form vorliegt:

```
collect: aBlock
|answer|
answer := self species new.
self do: [:element|answer add: (aBlock value: element)].
^ answer
```

Um die Ausführung von "collect:" – im Hinblick auf diese Methoden-Vereinbarung – zu beschreiben, betrachten wir die folgenden Anforderungen:

```
VarBag := Bag new.
VarBag add: '32'; add: '37'; add: '37'; add: '34'.
VarBag collect: [:einObjekt|einObjekt asInteger]
```

Hinweis: Sofern wir diese Anforderungen mittels der Menü-Option "Show It" des Menüs "Smalltalk" zur Ausführung bringen, erhalten wir z.B. "Bag(37 37 34 32)" als Ergebnis-Objekt angezeigt.

Wird das Argument

```
[:einObjekt|einObjekt asInteger]
```

der Keyword-Message "collect:" für den Parameter "aBlock" der oben angegebenen Methode "collect:" eingesetzt, so ergibt sich die folgende Situation:

```
collect: [:einObjekt|einObjekt asInteger]
|answer|
answer := self species new.
self do: [:element|
  answer add:([:einObjekt|einObjekt asInteger] value: element)].
∧ answer
```

Durch die Ausführung der Anforderung

```
answer := self species new.
```

wird eine Instanz der Basis-Klasse “Bag” eingerichtet und der temporären Variablen “answer” zugeordnet.

- In dieser Situation bewirkt die Message “species”, daß als Ergebnis-Objekt diejenige Klasse erhalten wird, aus deren Instanziierung das Empfänger-Objekt von “collect:” hervorgegangen ist.

Bei der anschließenden Ausführung der Message “do:” wird zunächst dem Block-Parameter “element” ein erstes Objekt von “VarBag” übergeben.
Anschließend erfolgt die Auswertung der folgenden eingeklammerten Message:

```
([:einObjekt|einObjekt asInteger] value: element)
```

Dabei wird die Auswertung der Block-Anforderung “einObjekt asInteger” durch den Einsatz von “value:” veranlaßt. Dies führt dazu, daß dem Block-Parameter “einObjekt” ein erstes Objekt von “VarBag” übergeben und dann die Anforderung “einObjekt asInteger” ausgeführt wird.
Daraufhin wird das Ergebnis-Objekt dieser Block-Anforderung durch den Einsatz von “add:” in den Sammler aufgenommen, auf den die Variable “answer” weist.
In dieser Weise wird für alle weiteren Objekte von “VarBag” verfahren.
Die Ausführung von “collect:” ist dann beendet, wenn alle in “VarBag” enthaltenen Objekte bearbeitet wurden.
Da das Ergebnis-Objekt der Message “collect:” durch die zuletzt innerhalb der Methode “collection:” zur Ausführung gebrachte Anforderung

```
∧ answer
```

bestimmt ist, resultiert als Ergebnis-Objekt der Message “collect:” derjenige Sammler, auf den die temporäre Variable “answer” weist, d.h. in diesem Fall ein Bag.

6.3 Logische Methoden

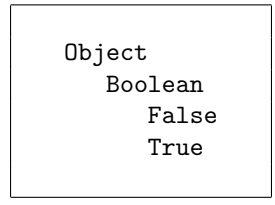
6.3.1 Die Pseudovariablen “true” und “false”

Um bestimmen zu können, daß eine oder mehrere Block-Anforderungen nur dann ausgeführt werden, wenn eine bestimmte Bedingung erfüllt ist, werden die Pseudovariablen “true” und “false” benötigt.

- Mit den Pseudovariablen *“true”* und *“false”* werden *Wahrheitswerte* von Bedingungen beschrieben. Ist eine Bedingung erfüllt, so wird dies durch die

Pseudovariable “true” gekennzeichnet. Ist dagegen eine Bedingung nicht zutreffend, so wird dieser Sachverhalt durch die Pseudovariable “false” charakterisiert.

Die Pseudovariable “true” ist *alleinige* Instanz der Basis-Klasse “True”, und “false” ist *alleinige* Instanz der Basis-Klasse “False”. Beide Basis-Klassen sind unmittelbare Unterklassen der abstrakten Basis-Klasse “Boolean” und in der Form



Bestandteil der Klassen-Hierarchie.

6.3.2 Vergleichsbedingungen

Ein Beispiel für eine Bedingung stellt eine Vergleichsbedingung dar, die durch die Anforderung

```
WerteErfassung11 bereitstellenWerte size > 3
```

mit den beiden Operanden “WerteErfassung11 bereitstellenWerte size” und “3” beschrieben wird. Bei dieser Vergleichsbedingung wird unter Einsatz der Basis-Methode “>” geprüft, ob die Bedingung zutrifft, daß mehr als drei Punktwerte erfaßt wurden.
Als Ergebnis-Objekt wird die Pseudovariable “true” erhalten, sofern diese Bedingung erfüllt ist. Im anderen Fall ergibt sich “false” als Ergebnis-Objekt.
Bei diesem Beispiel haben wir die *logische* Basis-Methode “>” zur Formulierung einer Bedingung eingesetzt, indem wir einen ersten Operanden (das vor “>” aufgeführte Empfänger-Objekt) mit einem zweiten Operanden (dem hinter “>” als Argument aufgeführten Objekt) auf die Beziehung “größer als” verglichen haben.

- “>”:
  Bei der *logischen* Basis-Methode “>” wird geprüft, ob der erste Operand größer als der zweite Operand ist.
  Trifft die in dieser Form formulierte *Bedingung* zu, so wird die Pseudovariable *“true”* zum Ergebnis-Objekt bestimmt. Ist die Bedingung *nicht* erfüllt, so resultiert die Pseudovariable *“false”* als Ergebnis-Objekt.

Neben der Methode “>” können auch die folgenden *logischen* Basis-Methoden zum Vergleich von Zeichenketten und zum Vergleich von numerischen Werten verwendet werden:

- “<”: zum Vergleich, ob der erste Operand kleiner als der zweite Operand ist;
- “>=”: zum Vergleich, ob der erste Operand größer oder gleich dem zweiten Operanden ist;

- "<=": zum Vergleich, ob der erste Operand kleiner oder gleich dem zweiten Operanden ist;
- "=": zum Vergleich, ob der erste Operand *gleich* dem zweiten Operanden ist (siehe unten);
- "==": zum Vergleich, ob der erste Operand *identisch* mit dem zweiten Operanden ist (siehe unten).

Damit geprüft werden kann, ob eine Zahl ungerade ist, läßt sich eine Vergleichsbedingung unter Einsatz der Basis-Methode "odd" angeben.

- **"odd"**:
  Das Ergebnis-Objekt der Message "odd", deren Empfänger-Objekt eine Zahl sein muß, ist gleich der Pseudovariablen *"true"*, sofern es sich um eine ungerade Zahl handelt. Ansonsten ist das Ergebnis-Objekt gleich der Pseudovariablen *"false"*.

Entsprechend läßt sich die *logische* Basis-Methode "even" verwenden, sofern geprüft werden soll, ob eine Zahl gerade ist.

- **"even"**:
  Das Ergebnis-Objekt der Message "even", deren Empfänger-Objekt eine Zahl sein muß, ist gleich der Pseudovariablen *"true"*, sofern es sich um eine gerade Zahl handelt. Ansonsten ist das Ergebnis-Objekt gleich der Pseudovariablen *"false"*.

Zum Beispiel erhalten wir durch

```
WerteErfassung11 bereitstellenWerte size even
```

die Pseudovariable "true" als Ergebnis-Objekt, sofern die Anzahl der erfaßten Punktwerte geradzahlig ist.
Um zu prüfen, ob in einem Bag ein oder mehrere Objekte enthalten sind, läßt sich die unäre Message *"isEmpty"* einsetzen.

- **"isEmpty"**:
  Das Ergebnis-Objekt dieser Message, deren Empfänger-Objekt ein Sammler sein muß, ist gleich der Pseudovariablen *"false"*, sofern das Empfänger-Objekt mindestens ein Objekt enthält. Andernfalls ist das Ergebnis-Objekt gleich der Pseudovariablen *"true"*.

Zum Beispiel ergibt sich auf der Basis von

```
VarBag := Bag new.
```

aus der Anforderung

```
VarBag isEmpty
```

die Pseudovariable “true” als Ergebnis-Objekt.

Um für ein bestimmtes Objekt prüfen zu können, ob es in einem Sammler enthalten ist, läßt sich die Basis-Methode “includes:” einsetzen.

- **“includes:”**:
  Die Message “includes:” liefert die Pseudovariable “true” als Ergebnis-Objekt, sofern das Argument dieser Message in demjenigen Sammler enthalten ist, der als Empfänger-Objekt dieser Message aufgeführt wird.

6.3.3 Prüfung auf Gleichheit und Identität

Definition von “Gleichheit” und “Identität”

Beim Vergleich zweier Objekte ist darauf zu achten, ob sie auf “Identität” oder auf “Gleichheit” geprüft werden sollen. Diese differenzierte Betrachtung gründet sich darauf, daß Objekte die Kapselung ihrer Attribute darstellen, deren zugehörige Attributwerte Speicherbereiche im Hauptspeicher belegen.

In dieser Hinsicht ist der folgende Sachverhalt bedeutsam:

- Durch eine Zuweisung verweist eine Variable, die links vom Zuweisungssymbol aufgeführt ist, auf dasjenige Objekt, das rechts vom Zuweisungssymbol ermittelt wird.

Wird daher einer Variablen eine Variable zugewiesen, so bezeichnen beide Variablennamen ein und dasselbe Objekt. Folglich stimmen die Speicherbereiche der Attributwerte überein, die über die beiden Variablennamen referenziert werden.

Nach dieser Vorbetrachtung legen wir jetzt wie folgt fest, wann zwei Objekte als “identisch” bzw. als “gleich” angesehen werden:

- Zwei Objekte sind *identisch*, wenn ihre Bezeichner auf *denselben Speicherplatz* verweisen.
- Zwei Objekte sind *gleich*, wenn sie in ihren Attributen wertmäßig übereinstimmen.

  Hinweis: Zwei identische Objekte sind folglich auch gleich.

Zur Prüfung der Identität sind die Messages “==” und “∼∼” einsetzbar.

- **“==”, “∼∼”**:
  Die zu diesen Messages gehörigen Methoden sind in der Klasse “Object” vereinbart. Als Ergebnis-Objekt der Message “==” (“∼∼”) wird die Pseudovariable “true” (“false”) ermittelt, sofern sowohl das Empfänger-Objekt als auch das als Argument angegebene Objekt (nicht) denselben Speicherplatz belegen. Andernfalls ergibt sich die Pseudovariable “false” (“true”).

Hinweis: Innerhalb der Klasse “Object” ist – zum Vergleich zweier Objekte – darüberhinaus die Methode “=” vereinbart. Diese Methode ist genauso festgelegt, wie die durch

"==" gekennzeichnete Methode.

Zur Prüfung, ob zwei Objekte gleich sind, lassen sich die Messages "=" und "~=" verwenden.

- **"="**, **"~="**:
  Als Ergebnis-Objekt der Message "=" wird bei gleichen (ungleichen) Objekten die Pseudovariable "true" ("false") ermittelt. Ansonsten resultiert die Pseudovariable "false" ("true").
  Als Ergebnis-Objekt der Message "~=" resultiert bei gleichen (ungleichen) Objekten die Pseudovariable "false" ("true"). Andernfalls wird die Pseudovariable "true" ("false") ermittelt.

Hinweis: Es ist zu beachten, daß die Messages zur Prüfung auf Gleichheit nur in bestimmten Klassen vereinbart sind. Die Methode "=" der Basis-Klasse "Object" kommt immer dann zum Einsatz, wenn innerhalb der Klasse des Empfänger-Objekts und deren Oberklassen die Methode "=" nicht definiert ist. Somit hat die Methode "=" dieselbe Wirkung wie die Methode "==", so daß in diesen Fällen mit "=" auf Identität geprüft wird.

Identität

Bringen wir die Anforderungen

```
InWerteErfassung11 := InWerteErfassung new.
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

zur Ausführung, tragen in das Eingabefeld des Erfassungsfensters die Werte "32" und "37" ein und stellen abschließend die Anforderung

```
InWerte11 := InWerteErfassung11
```

so ergibt sich die folgende Situation:

Abbildung 6.1: Identität von Objekten

Da die Instanz-Variable "werteBag" von "InWerte11" und die Instanz-Variable "werteBag" von "InWerteErfassung11" auf *ein* und *denselben* Speicherbereich verweisen, resultiert aus der Message

```
InWerte11 == InWerteErfassung11
```

die Pseudovariable "true" als Ergebnis-Objekt.

- Instanzen der Klasse “SmallInteger” (d.h. ganze Zahlen von “−32767” bis “32767”) und der Klasse “Character” sind Objekte, die im SMALLTALK-System nur einmal vorkommen (Unikate).
  Somit wird beim Einsatz von Instanzen dieser Klassen immer dasselbe Objekt angesprochen, so daß zwei in gleicher Weise geschriebene Literale stets identisch sind.

  Hinweis: Weitere Objekte, die im SMALLTALk-System ebenfalls nur einmal vorkommen, sind die Pseudovariablen “true” bzw. “false” (Instanzen der Klasse “True” bzw. “False” sowie die Pseudovariable “nil” (eine Instanz der Klasse “UndefinedObject”).

Anders ist der Sachverhalt z.B. bei Strings. Stellen wir z.B. die Anforderungen

```
Var1 := 'a'.
Var2 := 'a'
```

so verweisen beide Variablen auf Instanzen der Klasse “String”, deren Attributwerte gleich dem Zeichen “$a” sind.
Da der Attributwert aus der 1. Zuweisung einen anderen Speicherbereich einnimmt als der Attributwert aus der 2. Zuweisung, resultiert aus der Prüfung

```
Var1 == Var2
```

die Pseudovariable “false” als Ergebnis-Objekt.
Ist es notwendig, identische Strings zu verwenden, so müssen Instanzen der Klasse “Symbol” eingesetzt werden.

Zum Beispiel liefert der Vergleich

```
Var1 == Var2
```

die Pseudovariable “true” als Ergebnis-Objekt, sofern diese Prüfung auf den Zuweisungen

```
Var1 := 'abc' asSymbol.
Var2 := 'abc' asSymbol
```

basiert.

Dabei hat die Message “asSymbol” die folgende Wirkung:

- **“asSymbol”**:
  Wird die Message “asSymbol” einer Instanziierung der Klasse “String” geschickt, so wird als Ergebnis-Objekt ein Symbol erhalten, d.h. eine Instanziierung der Basis-Klasse “Symbol”.

Hinweis: Natürlich resultiert aus dem Vergleich

```
'abc' asSymbol == 'abc' asSymbol
```

ebenfalls die Pseudovariable “true”.

Gleichheit

Stellen wir die Anforderungen

```
InWerteErfassung11 := InWerteErfassung new.
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

sowie die Anforderungen

```
InWerte11 := InWerteErfassung new.
InWerte11 sammelnWerte: 'Jahrgangsstufe 11'
```

und tragen in das Eingabefeld des Erfassungsfensters jeweils die Werte "32" und "37" ein, so ergibt sich die folgende Situation:

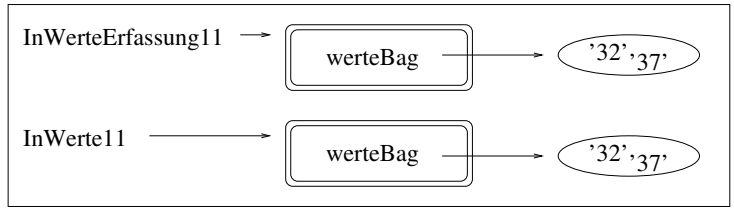


Abbildung 6.2: Gleichheit von Objekten

Die anschließende Prüfung auf Gleichheit in der Form

```
InWerte11 = InWerteErfassung11
```

bzw. in der Form

```
InWerte11 bereitstellenWerte =
  InWerteErfassung11 bereitstellenWerte
```

liefert – erstaunlicherweise – in beiden Fällen die Pseudovariable "false" als Ergebnis-Objekt.
Dies liegt daran, daß beim Vergleich durch die Verwendung von "=" die Methode "==" zum Einsatz kommt, so daß eine Prüfung auf Identität erfolgt. Dieser Sachverhalt basiert darauf, daß die Methode "=" der Klasse "Object", die mit der Methode "==" zur Prüfung auf Identität übereinstimmt, aktiviert wird. Dies liegt daran, daß innerhalb der Klasse "Bag" und deren Oberklasse "Collection" die Methode "=" nicht vereinbart ist.

Kopieren

Neben den Zuweisungen, bei denen nicht die Objekte, sondern Verweise kopiert werden, gibt es weitere Möglichkeiten, Kopiervorgänge auszulösen. Dazu dienen die beiden Basis-Methoden "shallowCopy" und "deepCopy".

Der Sachverhalt, daß die beiden Objekte “InWerte11” und “InWerteErfassung11” existieren, deren Instanz-Variablen “werteBag” jeweils ein Bag der Form Bag(’32’ ’37’) bzw. Bag(’23’ ’73’) zugeordnet ist, läßt sich wie folgt darstellen:

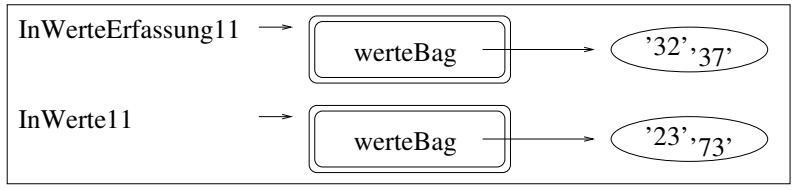


Abbildung 6.3: Ausgangssituation

Stellen wir auf dieser Basis die Anforderung

```
InWerte11 := InWerteErfassung11 shallowCopy
```

so ergibt sich die folgende Situation:

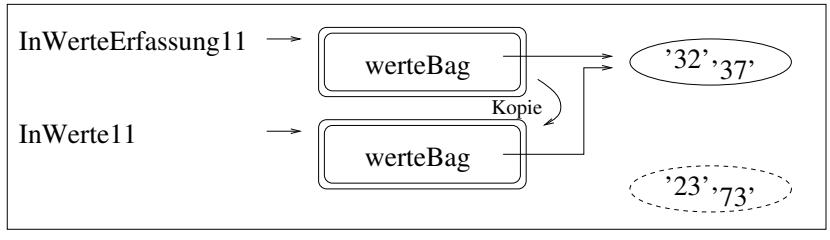


Abbildung 6.4: Seichtes Kopieren

Durch den Einsatz der Methode **“shallowCopy”** erfolgt ein *seichter Kopiervorgang*, bei dem die Instanz-Variable “werteBag” der Instanz “InWerteErfassung11” und damit deren Verweis auf das Objekt “Bag(’32’ ’37’)” kopiert wurde. Somit nehmen die Attributwerte der Instanz-Variablen “werteBag” – sowohl von der Instanz “InWerteErfassung11” als auch von “InWerte11” – denselben Speicherbereich ein.

Stellen wir daher in dieser Situation die Anforderung

```
InWerte11 bereitstellenWerte ==
  InWerteErfassung11 bereitstellenWerte
```

so wird die Pseudovariable “true” als Ergebnis-Objekt ermittelt.

Hinweis: Wollen wir anschließend zusätzliche Punktwerte erfassen, indem wir die Anforderung

```
InWerteErfassung11 durchfuehrenErfassung
```

ausführen lassen, so stehen die zusätzlich in das Erfassungsfenster z.B. von “InWerteErfassung11” eingegebenen Werte sowohl über die Instanz “InWerteErfassung11” als auch

über die Instanz “InWerte11” zur Verfügung.

Auf der Basis der oben angegebenen Ausgangssituation resultiert dagegen aus der Anforderung

```
InWerte11 := InWerteErfassung11 deepCopy
```

der folgende Sachverhalt:

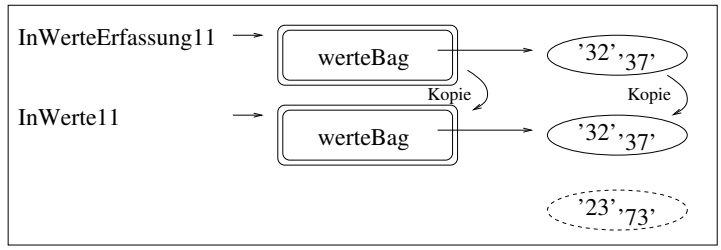


Abbildung 6.5: Tiefes Kopieren

Durch die Methode **“deepCopy”** erfolgt ein *tiefer Kopiervorgang*, bei dem sowohl die Instanz-Variable “werteBag” der Instanz “InWerteErfassung11” als auch das Objekt “Bag('32' '37')”, auf das “werteBag” verweist, kopiert werden.
Da sich durch *tiefes Kopieren* ein Abbild des kopierten Objekts an einem anderen Speicherbereich ergibt, wird folglich die Pseudovariable “false” als Ergebnis-Objekt von

```
InWerte11 bereitstellenWerte ==
  InWerteErfassung11 bereitstellenWerte
```

ermittelt.
Wollen wir anschließend zusätzliche Punktwerte erfassen, indem wir die Anforderung

```
InWerteErfassung11 durchfuehrenErfassung
```

ausführen lassen, so ist zu beachten, daß die zusätzlich in das Erfassungsfenster eingegebenen Werte nur über die Instanz “InWerteErfassung11” zur Verfügung stehen.

6.3.4 Logische “Und”-Verknüpfungen

Die “Und”-Verknüpfung

Neben der Möglichkeit, die Wahrheitswerte von Vergleichsbedingungen zu prüfen, können auch die Wahrheitswerte mehrerer miteinander verknüpfter Bedingungen ermittelt werden.
Zum Beispiel läßt sich durch den Einsatz der *logischen* Basis-Methode “and:” in der Form

```
(WerteErfassung11 size > 3) and: [WerteErfassung12 size > 3]
```

untersuchen, ob sowohl die Anzahl der in "WerteErfassung11" als auch der in "WerteErfassung12" gesammelten Punktwerte größer als 3 ist.
Als Argument von "and:" ist der Block "[WerteErfassung12 size > 3]" aufgeführt, aus dessen Ausführung sich die Pseudovariable "true" bzw. "false" als Ergebnis-Objekt ergibt.

- Sofern bei einem Block aus der Ausführung der dynamisch zuletzt ausgeführten Block-Anforderung die Pseudovariable "true" bzw. "false" resultiert, wird der Block als *logischer* Block bezeichnet.

Bei der Methode "and:" handelt es sich um eine *logische* Basis-Methode, bei deren Aufruf ein *logischer Block* als Argument aufgeführt werden muß.

- **"and:", "&"**:
  Aus einer zusammengesetzten Bedingung der Form

  - <bedingung_1> and: [<bedingung_2>]

  bzw.

  - <bedingung_1> & [<bedingung_2>]

  resultiert genau dann die Pseudovariable "true" als Ergebnis-Objekt, wenn sowohl die als Empfänger-Objekt "<bedingung_1>" von "and:" ("&") angegebene Bedingung als auch die innerhalb des Arguments von "and:" ("&") aufgeführte Bedingung "<bedingung_2>" erfüllt sind.
  Ist dies nicht der Fall, so wird das Ergebnis-Objekt "false" ermittelt.

  Hinweis: Es ist zu beachten, daß das Empfänger-Objekt durch "<bedingung_1>" gekennzeichnet ist. Sofern sich "true" als Ergebnis-Objekt ergibt, wird die in der Basis-Klasse "True" vereinbarte Basis-Methode "and:" ausgeführt – andernfalls ist es diejenige Methode "and:", die in der Basis-Klasse " False" festgelegt ist.
  Innerhalb der Basis-Klasse "True" ist die Basis-Methode "and:" durch

  ```
  and: aBlock
       ∧ aBlock value
  ```

  und innerhalb der Basis-Klasse "False" durch

  ```
  and: aBlock
       ∧ false
  ```

  vereinbart.
  Soll eine zusammengesetzte Bedingung aus mehr als zwei Bedingungen aufgebaut werden, so ist zu beachten, daß bei logischen Methoden zuerst das Empfänger-Objekt ermittelt wird. Daher kann es erforderlich sein, daß Klammern gesetzt werden müssen, was z.B. in der Situation

  (<`bedingung_1`> `and:` [<`bedingung_2`>]) and: [<`bedingung_3`>]

  der Fall ist.

Mehrere logische "Und"-Verknüpfungen

Zur abkürzenden Beschreibung der Prüfung mehrerer logischer "Und"-Verknüpfungen läßt sich in bestimmten Fällen die Basis-Methode "inject:into:" verwenden.
Durch diese Methode läßt sich prüfen, ob die Objekte eines Sammlers bestimmte Eigenschaften haben. Dabei ist der Sammler als Empfänger-Objekt der Message "inject:into:" aufzuführen. Als Argument des Selektors "inject:" ist ein *Anfangswert* und als Argument des Selektors "into:" ein Block anzugeben, für den zwei Block-Parameter vereinbart sind.

- **"inject:into:"**:
  Bei der Ausführung von "inject:into:" nimmt der 1. Block-Parameter zunächst den Wert an, der als Argument des Keyword-Selektors "inject:" aufgeführt ist. Für den 2. Block-Parameter wird ein erstes Objekt des Empfänger-Objektes eingesetzt.
  Im nächsten Schritt wird für den Wert des 1. Block-Parameters das Ergebnis-Objekt der letzten Block-Anforderung und für den 2. Block-Parameter ein weiteres Objekt des Empfänger-Objektes eingesetzt.
  Die Block-Anforderungen werden in dieser Form solange ausgeführt, bis alle Objekte des Empfänger-Objektes verarbeitet sind.
  Als Ergebnis-Objekt resultiert dasjenige Ergebnis-Objekt, das bei der zuletzt ausgeführten Block-Anforderung erhalten wird.

Ist z.B. ein Bag durch die Anforderungen

```
VarBag := Bag new.
VarBag add: 32; add: 37; add: 34; add: 37
```

eingerichtet worden, so kann wie folgt geprüft werden, ob alle in ihm enthaltenen Objekte ungeradzahlig sind:

```
VarBag inject: true into:
  [:pruefWert :einObjekt|pruefWert and: [einObjekt odd]]
```

Als Ergebnis-Objekt der Message "inject:into:" ergibt sich die Pseudovariable "false", weil z.B. die Zahl 32 nicht ungeradzahlig ist.

6.3.5 Logische "Oder"-Verknüpfungen

Soll z.B. geprüft werden, ob die Anzahl der in "WerteErfassung11" oder der in "WerteErfassung12" gesammelten Punktwerte größer als 3 ist, so läßt sich hierzu die Basis-Methode "or:" wie folgt verwenden:

```
(WerteErfassung11 size > 3) or: [WerteErfassung12 size > 3]
```

Bei der Methode "or:" handelt es sich ebenfalls um eine *logische* Basis-Methode, bei deren Aufruf ein *logischer Block* als Argument aufgeführt werden muß.

- **"or:", ".|.''"**:
  Aus einer zusammengesetzten Bedingung der Form

- <bedingung_1> or: [<bedingung_2>]

bzw.

- <bedingung_1> | [<bedingung_2>]

resultiert genau dann die Pseudovariable "true", wenn die als Empfänger-Objekt von "or:" ("|") angegebene Bedingung "<bedingung_1>" oder die innerhalb des Arguments von "or:" ("|") aufgeführte Bedingung "<bedingung_2>", d.h. mindestens eine dieser beiden Bedingungen, erfüllt sind.
Ist dies nicht der Fall, d.h. sind beide Bedingungen nicht zutreffend, so ergibt sich "false" als Ergebnis-Objekt.

Hinweis: Sofern sich als Ergebnis-Objekt die Pseudovariable "true" ergibt, wird die Basis-Methode "or:" ausgeführt, die in der Basis-Klasse "True" vereinbart ist – andernfalls wird die Basis-Methode "or:" eingesetzt, die in der Basis-Klasse "False" festgelegt ist.
Sollen mehrere Bedingungen verknüpft werden, so muß geeignet geklammert werden, z.B. in der Form:

(<`bedingung_1`> `or:` [<`bedingung_2`>]) `or:` [<`bedingung_3`>]

6.3.6 Verneinung einer Bedingung

Zur Verneinung einer Bedingung steht die *logische* Basis-Methode "not" zur Verfügung.

- **"not"**:
  Eine verneinte Bedingung der Form

  - <bedingung> not

  ist genau dann erfüllt, wenn die als Empfänger-Objekt von "not" angegebene Bedingung "<bedingung>" nicht zutrifft. Ist dies der Fall, so resultiert die Pseudovariable "true" als Ergebnis-Objekt. Andernfalls wird die Pseudovariable "false" ermittelt.

Zum Beispiel erhalten wir durch

```
WerteErfassung11 bereitstellenWerte size even not
```

die Pseudovariable "false" als Ergebnis-Objekt, sofern die Anzahl der erfaßten Punktwerte geradzahlig ist.

6.4 Auswahl-Methoden

Soll beim Einsatz von logischen Methoden festgelegt werden, daß bestimmte Block-Anforderungen nur dann ausgeführt werden, wenn spezielle Bedingungen erfüllt sind, so sind Basis-Methoden der Form "ifTrue:ifFalse:", "ifTrue:" und "ifFalse:" einzusetzen.
Um z.B. die Problemstellung

- Sammle alle Punktwerte der Instanz “WerteErfassung11”, die ungeradzahlig sind, und bestimme deren Anzahl!

lösen zu können, muß im Hinblick auf den Sachverhalt, daß die Anzahl der erfaßten Punktwerte geradzahlig oder ungeradzahlig ist, eine Möglichkeit zur *Verzweigung* existieren.

Um eine Verzweigung in einem Struktogramm darzustellen, setzen wir einen “Bedingungs-Strukturblock” der folgenden Form ein:

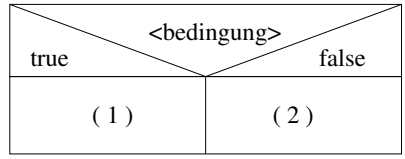


Abbildung 6.6: Bedingungs-Strukturblock

Hierbei werden die durch “(1)” gekennzeichneten Strukturblöcke dann durchlaufen, wenn die aufgeführte Verzweigungs-Bedingung “<bedingung>” zutrifft. Ist diese Bedingung *nicht* erfüllt, so werden die durch “(2)” gekennzeichneten Strukturblöcke durchlaufen.

Die Umsetzung des “Bedingungs-Strukturblockes” wird über die Message “ifTrue:ifFalse:” vorgenommen, mit der die Ausführung der Basis-Methode “ifTrue:ifFalse:” abgerufen wird. Dabei muß die Verzweigungs-Bedingung als Empfänger-Objekt dieser Message, der Inhalt von “(1)” als Argument des Selektors “ifTrue:” und der Inhalt von “(2)” als Argument des Selektors “ifFalse:” formuliert werden. Es ist zu beachten, daß die beiden Argumente in Form von Blöcken innerhalb der Message “ifTrue:ifFalse:” aufgeführt werden müssen.

Sofern für “(1)” bzw. “(2)” *keine* Strukturblöcke angegeben sind, ist beim Selektor “ifTrue:” bzw. “ifFalse:” ein leerer Block in der Form “[ ]” als Argument aufzuführen.

Mit der Basis-Methode “ifTrue:ifFalse:” läßt sich folglich eine Verzweigung formulieren, indem die Ausführung eines von zwei Blöcken bewirkt wird.

- **“ifTrue:ifFalse:”**:
  Ist das Empfänger-Objekt der Message “ifTrue:ifFalse:” gleich der Pseudovariablen *“true”*, so wird derjenige Block ausgeführt, der als Argument des Selektors “ifTrue:” angegeben ist. Handelt es sich beim Empfänger-Objekt um die Pseudovariable *“false”*, so wird der Block ausgeführt, der als Argument des Selektors “ifFalse:” aufgeführt ist.

Hinweis: Die Basis-Methode “ifTrue:ifFalse:” ist sowohl in der Basis-Klasse “True” als auch in der Basis-Klasse “False” festgelegt.
In der Basis-Klasse “True” ist die Methode “ifTrue:ifFalse:” in der Form

```
ifTrue: trueBlock ifFalse: falseBlock
    ∧ trueBlock value
```

und in der Basis-Klasse “False” in der Form

```
ifTrue: trueBlock ifFalse: falseBlock
    ∧ falseBlock value
```

vereinbart.
Ist das Empfänger-Objekt von “ifTrue:ifFalse:” gleich der Pseudovariablen “true”, so wird die Methode “ifTrue:ifFalse:” der Basis-Klasse “True” und damit “∧ trueBlock value” ausgeführt. Ist jedoch das Ergebnis-Objekt gleich “false”, so erfolgt die Ausführung der Methode “ifTrue:ifFalse:” der Basis-Klasse “False” und damit die Ausführung von

```
∧ falseBlock value
```

Im Fall von *leeren Blöcken* darf anstelle von

```
<bedingung> ifTrue: <block> ifFalse: [ ]
```

die verkürzte Schreibweise

```
<bedingung> ifTrue: <block>
```

verwendet werden. Entsprechend kann

```
<bedingung> ifTrue: [ ] ifFalse: <block>
```

abgekürzt werden zu:

```
<bedingung> ifFalse: <block>
```

Durch den Einsatz des Bedingungs-Strukturblockes läßt sich für die oben angegebene Problemstellung der folgende Lösungsansatz entwickeln:

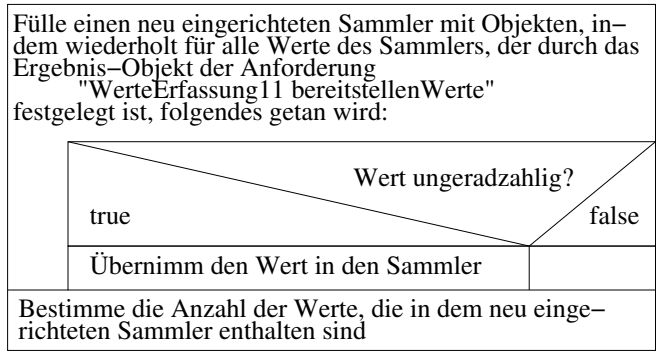


Abbildung 6.7: Struktogramm “Ungeradzahlige Werte”

Mit Hilfe der Basis-Methode “ifTrue:ifFalse:” können wir dieses Struktogramm wie folgt umformen:

```
(WerteErfassung11 bereitstellenWerte
   collect: [:einObjekt| (einObjekt asInteger odd)
                         ifTrue: [einObjekt]
                         ifFalse: [ ]
            ])
              size
```

Da der *leere* Block "[ ]" als Argument des Selektors "ifFalse:" aufgeführt ist, kann diese Anforderung in der Form

```
(WerteErfassung11 bereitstellenWerte
   collect: [:einObjekt| (einObjekt asInteger odd)
                         ifTrue: [einObjekt]
            ])
              size
```

abgekürzt werden.

6.5 Wiederholungs-Methoden

Sollen eine oder mehrere Block-Anforderungen nicht nur einmalig, sondern wiederholt ausgeführt werden, so läßt sich hierzu z.B. die Basis-Methode "timesRepeat:" aus der Basis-Klasse "Integer" einsetzen.
Zum Beispiel wird durch die Ausführung von

```
|summe zaehler|
zaehler := 1.
summe := 0.
5 timesRepeat: [summe := summe + (zaehler * zaehler).
                zaehler := zaehler + 1].
summe
```

bewirkt, daß sich als Ergebnis-Objekt der letzten Anforderung "summe" die Zahl 55 als Summe der Quadratzahlen von 1 bis 5 ergibt.
Durch die Basis-Methode "timesRepeat:" kann ein Block eine konkret festgelegte Anzahl von Malen wiederholt ausgeführt werden.

- **"timesRepeat:"**:
  Die Häufigkeit, mit der der als Argument aufgeführte Block ausgeführt wird, ist durch das ganzzahlige Empfänger-Objekt der Message "timesRepeat:" festgelegt. Als Ergebnis-Objekt resultiert die ganze Zahl 0.

Die Ausführung von "timesRepeat:" wird mittels der Basis-Methode "whileTrue:" gemäß der folgenden Methoden-Vereinbarung vorgenommen:

```
timesRepeat: aBlock
|anInteger|
anInteger := self.
[anInteger > 0] whileTrue: [anInteger := anInteger - 1.
                            aBlock value]
```

Das Empfänger-Objekt von "timesRepeat:" wird somit der temporären Variablen "anInteger" zugeordnet, und für "aBlock" wird das hinter dem Selektor "timesRepeat:" angegebene Argument eingesetzt.
Bei der Ausführung der Basis-Methode "whileTrue:", die die Form

```
[<bedingung>] whileTrue: <block>
```

besitzt, wird der als Argument des Selektors "whileTrue:" angegebene Block wiederholt ausgeführt. Diese Wiederholung wird durch die Bedingung gesteuert, die als Empfänger-Objekt von "whileTrue:" aufgeführt ist.

- Durch den Einsatz der Methode "whileTrue:" läßt sich die Wiederholung einer Block-Ausführung über eine *Bedingung* steuern, so daß die Anzahl der gewünschten Wiederholungen *nicht* bereits vor der erstmaligen Block-Ausführung feststehen muß.

Im Hinblick auf die Ausführung der Basis-Methode "whileTrue:" ist folgendes festzustellen:

- **"whileTrue:"**:
  Das Empfänger-Objekt der Message "whileTrue:" muß ein *logischer* Block sein.
  Ist das Ergebnis-Objekt, das aus der erstmaligen Ausführung des logischen Blockes resultiert, gleich der Pseudovariablen "false", so wird der Argument-Block, d.h. der Block, der als Argument des Selektors "whileTrue:" angegeben ist, nicht ausgeführt.
  Ergibt sich dagegen die Pseudovariable "true" als Ergebnis-Objekt, so erfolgt die erstmalige Ausführung des Argument-Blockes. Anschließend wird der logische Block erneut ausgewertet. In Abhängigkeit davon, welche Pseudovariable als Ergebnis-Objekt resultiert, wird die Wiederholung beendet oder fortgesetzt.
  Die Ausführung des Argument-Blockes wird solange wiederholt, bis sich die Pseudovariable "false" als Ergebnis-Objekt des logischen Blockes ergibt.
  Als Ergebnis-Objekt der Message "whileTrue:" ist die Pseudovariable "nil" festgelegt.

Wollen wir z.B. die Summe der Quadratzahlen von 1 bis 5 bestimmen, so läßt sich dies auch über die folgenden Anforderungen erreichen:

```
|summe zaehler|
zaehler := 1.
summe := 0.
[zaehler < 6] whileTrue: [summe := summe + (zaehler * zaehler).
                          zaehler := zaehler + 1].
summe
```

Da es in bestimmten Fällen sinnvoll sein kann, die Wiederholung einer Block-Ausführung davon abhängig zu machen, daß eine Bedingung zu einem bestimmten Zeitpunkt *nicht* mehr zutrifft, steht neben "whileTrue:" zusätzlich die Basis-Methode "whileFalse:" zur Verfügung, die wie folgt verwendet werden muß:

- [<bedingung>] whileFalse: <block>

Im Hinblick auf die Ausführung dieser Methode ist folgendes festzustellen:

- **"whileFalse:"**:
  Das Empfänger-Objekt der Message "whileFalse:" muß ein *logischer* Block sein.
  Ist das Ergebnis-Objekt, das aus der erstmaligen Ausführung des logischen Blockes resultiert, gleich der Pseudovariablen "true", so wird der Argument-Block, d.h. der Block, der als Argument des Selektors "whileFalse:" angegeben ist, nicht ausgeführt.
  Ergibt sich dagegen die Pseudovariable "false" als Ergebnis-Objekt, so erfolgt die erstmalige Ausführung des Argument-Blockes. Anschließend wird der logische Block erneut ausgewertet. Je nachdem, welche Pseudovariable als Ergebnis-Objekt resultiert, wird die Wiederholung beendet oder fortgesetzt.
  Die Ausführung des Argument-Blockes wird solange wiederholt, bis die Pseudovariable "true" als Ergebnis-Objekt des logischen Blockes erhalten wird.
  Als Ergebnis-Objekt der Message "whileFalse:" ist die Pseudovariable "nil" festgelegt.

Unter Einsatz der Methode "whileFalse:" kann die oben angegebene Berechnung des Summenwertes wie folgt modifiziert werden:

```
|summe zaehler|
zaehler := 1.
summe := 0.
[zaehler > 5] whileFalse: [summe := summe + (zaehler * zaehler).
                           zaehler := zaehler + 1].
summe
```

Hinweis: Als Alternative läßt sich die Berechnung des Summenwertes auch in der Form

```
VarBag := Bag new. VarBag add: 1; add: 2; add: 3; add: 4; add: 5.
VarBag inject:0 into:[:summe :eineZahl|summe + (eineZahl * eineZahl)]
```

durch den Einsatz der Basis-Methode "inject:into:" ermitteln.

6.6 Ausgewählte Methoden zum Aufbau von Sammlern

6.6.1 Die Basis-Methode “add:”

Wie wir bereits im Abschnitt 3.4 festgestellt haben, läßt sich die Basis-Methode “add:” zum Füllen eines Bags verwenden.

- **“add:”**:
  Bei der Ausführung der Basis-Methode “add:” wird das Objekt, das als Argument des Selektors “add:” aufgeführt ist, demjenigen Bag hinzugefügt, der als Empfänger-Objekt der Message “add:” verwendet wird.
  Als Ergebnis-Objekt resultiert das Objekt, das dem Bag hinzugefügt wurde.

Zum Beispiel wird durch die Anforderung

```
VarBag := Bag new add: 32; add: 37; add: 34; add: 37; yourself
```

bewirkt, daß die vier ganzzahligen Werte “32”, “37”, “34” und “37” innerhalb des Bags gesammelt werden, auf den die Variable “VarBag” verweist.

Da die Methode “yourself” am Ende der Kaskade verwendet wird, ergibt sich nicht das Objekt “37” als Ergebnis-Objekt der innerhalb der Zuweisung angegebenen Kaskade, sondern der resultierende Bag, der sich z.B. in der Form “Bag(37 37 34 32)” durch den Einsatz der Menü-Option “Show It” des “Smalltalk”-Menüs anzeigen läßt.

Hinweis: Setzen wir innerhalb der angegebenen Kaskade *nicht* die Message “yourself” ein, so wird der Variablen “VarBag” der Wert “37” als Ergebnis-Objekt der zuletzt ausgeführten Message “add:” zugewiesen.

Wie sich z.B. mehrere Bags innerhalb eines Bags sammeln lassen, zeigen die folgenden Anforderungen:

```
|varBag1 varBag2|
varBag1 := Bag new.
varBag2 := Bag new.
VarBag := Bag new.
varBag1 add: '11'; add: 'w'.
varBag2 add: '12'; add: 'm'.
VarBag add: varBag1; add: varBag2; yourself
```

Rufen wir die Ausführung dieser Anforderungen mit der Menü-Option “Show It” des “Smalltalk”-Menüs ab, so resultiert z.B. die folgende Anzeige:

```
Bag(Bag('11' 'w')Bag('m' '12'))
```

Soll die Gesamtheit der Objekte angezeigt werden, die in beiden Bags gesammelt wurden, so läßt sich dies in der Form

```
VarBag do: [:einBag|Transcript cr.
             einBag do: [:einObjekt|
                Transcript show: einObjekt printString]]
```

anfordern.

6.6.2 Die Basis-Methoden "collect:", "select:" und "reject:"

Mit der Basis-Methode "collect:" haben wir bereits im Abschnitt 6.2.3 eine Möglichkeit kennengelernt, mit der sich aus einem Sammler ein jeweils *gleichartiger* Sammler aufbauen läßt.
Beim Einsatz der Message "collect:" ist zu beachten, daß der Sammler, aus dessen Objekten ein neuer Sammler aufgebaut werden soll, als Empfänger-Objekt und der Block, durch den der Aufbau des neuen Sammlers beschrieben wird, als Argument angegeben wird.

- **"collect:"**:
  Als Ergebnis-Objekt von "collect:" resultiert ein Sammler, der aus derselben Klasse wie das Empfänger-Objekt von "collect:" instanziiert ist. Er wird mit denjenigen Objekten gefüllt, die als Ergebnis-Objekte aus der wiederholten Ausführung des Blockes resultieren, der innerhalb der Message "collect:" als Argument aufgeführt ist.

Sollen beim Aufbau eines neuen Sammlers nicht alle, sondern nur ausgewählte Objekte eines Sammlers übernommen werden, so können hierzu die Basis-Methoden "select:" und "reject:" verwendet werden.

- **"select:"**, **"reject:"**:
  Als Ergebnis-Objekt resultiert jeweils ein Sammler, der aus derjenigen Klasse instanziiert wird, aus der das Empfänger-Objekt der betreffenden Message instanziiert wurde.
  Als Argumente dieser Messages sind *logische Blöcke* zu verwenden, bei denen die jeweils letzte Block-Anforderung das Ergebnis-Objekt der Block-Ausführung in Form der Pseudovariablen "true" oder "false" ermittelt.
  Beim Einsatz von "select:" ("reject:") werden alle die Objekte des Empfänger-Objektes gesammelt, für die die Ausführung des *logischen Blockes* den Wahrheitswert "true" ("false") liefert.

Zum Beispiel läßt sich aus dem durch

```
VarBag := Bag new.
VarBag add: 32; add: 37; add: 34; add: 37
```

eingerichteten Bag wie folgt ein neuer Bag aufbauen, der allein die ungeraden Zahlen enthält:

```
VarBagNeu := VarBag select:[:eineZahl|eineZahl odd]
```

Hiermit ist gleichbedeutend:

```
VarBagNeu := VarBag reject:[:eineZahl|eineZahl even]
```

Anstelle von "select:" und "reject:" kann z.B. auch die Basis-Methode "remove:" verwendet werden, mit der sich Objekte gezielt löschen lassen.

- **"remove:"**:
  Durch die Ausführung der Methode "remove:" wird in dem Sammler, der als Empfänger-Objekt der Message "remove:" aufgeführt ist, dasjenige Objekt gelöscht, das als Argument der Message "remove:" angegeben ist.

Zum Beispiel läßt sich aus dem durch

```
VarBag := Bag new.
VarBag add: 32; add: 37; add: 34; add: 37
```

eingerichteten Bag wie folgt ein neuer Bag aufbauen, der allein die geraden Zahlen enthält:

```
VarBagNeu := VarBag collect: [:eineZahl| (eineZahl odd)
                                          ifTrue:[nil]
                                          ifFalse:[eineZahl]].
[VarBagNeu includes:nil] whileTrue:[VarBagNeu remove: nil]
```

6.7 Ausgewählte Methoden zur Bearbeitung sortierter Sammler

6.7.1 Einrichtung eines sortierten Sammlers

Bei der Lösung der bisherigen Problemstellungen haben wir einen Bag als Sammler für Objekte verwendet. Damit Lösungspläne sich in besonders einfacher Form angeben lassen, ist es oftmals nützlich, anstelle eines Bags einen Sammler einzusetzen, der über spezielle Eigenschaften verfügt.

Von besonderer Bedeutung sind Sammler, in dem Objekte gemäß einer festgelegten *Sortierreihenfolge* eingetragen sind, so daß gemäß der zugehörigen *Sortierfolgeordnung* auf einzelne Objekte zugegriffen werden kann.

Ein derartiger Sammler wird *SortedCollection* genannt, weil dazu eine Instanz der Basis-Klasse "SortedCollection" verwendet wird.

- Genau wie bei einem Sammler in Form eines Bags dürfen auch bei einer Instanz der Klasse "SortedCollection" gleiche Objekte mehrfach auftreten. Jedoch ist eine SortedCollection – im Gegensatz zu einem Bag – ein strukturierter Sammler, da die Objekte gemäß der aufsteigenden Sortierfolge geordnet sind und auf sie gezielt über eine ganze Zahl – als *Index-Position* – zugegriffen werden kann.

Um aus einem Sammler eine SortedCollection einzurichten, in die der Inhalt des Sammlers übertragen wird, läßt sich die Basis-Methode "asSortedCollection" verwenden.

- **“asSortedCollection”**:
  Durch die Ausführung der Basis-Methode “asSortedCollection” läßt sich – als Ergebnis-Objekt – eine Instanz der Basis-Klasse “SortedCollection” einrichten, die aus den Objekten desjenigen Sammlers aufgebaut ist, der als Empfänger-Objekt dieser Message verwendet wird.

6.7.2 Zugriff auf gesammelte Objekte

Um gezielt auf ein Objekt, das in einer Instanz der Basis-Klasse “SortedCollection” enthalten ist, zugreifen zu können, kann die Basis-Methode “at:” eingesetzt werden.

- **“at:”**:
  Sofern die ganze Zahl, die innerhalb der Message “at:” als Argument aufgeführt ist, eine zulässige Index-Position innerhalb der Objekte des Sammlers kennzeichnet, legt das an dieser Index-Position plazierte Objekt das Ergebnis-Objekt der Message “at:” fest.

Der Einsatz einer SortedCollection bietet sich z.B. zur Lösung der folgenden Problemstellung an:

- Bestimme das Maximum der durch “WerteErfassung11” erfaßten Punktwerte!

Unter Einsatz der Basis-Methoden “asSortedCollection” und “at:” läßt sich die Lösung wie folgt angeben:

```
WerteErfassung11 bereitstellenWerte asSortedCollection
                 at: (WerteErfassung11 bereitstellenWerte size)
```

Da eventuell der Wert “35” in der Form “ 35” (mit einem Leerzeichen vor der Ziffer “3”) oder “35 ” (mit einem Leerzeichen hinter der Ziffer “5”) erfaßt wurde, ist es sinnvoll, alle Zeichenketten – vor ihrer Übernahme in eine SortedCollection – in ganze Zahlen umzuwandeln.
Um diesen Lösungsplan zu formulieren, formen wir die angegebene Anforderung wie folgt um:

```
(WerteErfassung11 bereitstellenWerte
   collect: [:einObjekt| einObjekt asInteger])
   asSortedCollection
   at: (WerteErfassung11 bereitstellenWerte size)
```

6.7.3 Bestimmung einer Index-Position

Zur Lösung der Problemstellung

- Bestimme die Anzahl der erfaßten Punktwerte, die kleiner als der Wert “35” sind!

kann die Basis-Methode “indexOf:” verwendet werden, durch deren Ausführung sich die Index-Position ermitteln läßt, an der ein bestimmtes Objekt innerhalb einer Instanz der Klasse “SortedCollection” positioniert ist.

- **“indexOf:”**:
  Als Ergebnis-Objekt der Message “indexOf:” wird die Index-Position bestimmt, an der das als Argument aufgeführte Objekt *erstmalig* in demjenigen Sammler auftritt, der als Empfänger-Objekt der Message angegeben ist.

Unter Einsatz der Methode “indexOf:” können wir – für “WerteErfassung11” – die Anforderung

```
((WerteErfassung11 bereitstellenWerte
    collect: [:einObjekt| einObjekt asInteger])
    asSortedCollection
    indexOf: 35) - 1
```

als Lösung der Problemstellung angeben.

6.7.4 Berechnung eines Medians

Der statistische Kennwert, mit dem sich die der Größe nach geordneten Werte in zwei Hälften aufteilen lassen, wird als “mittlerer Wert” errechnet und *Median* genannt.
Sind sämtliche Werte in eine *Rangreihe* der Form

$$x_1 \leq x_2 \leq x_3 \leq ... \leq x_n$$

gebracht, muß der Median in Abhängigkeit von der Anzahl der Werte “n” wie folgt berechnet werden:

- Ist “n” eine *ungerade* Zahl, so ist der Median gleich demjenigen Wert innerhalb der Rangreihe, der an der Stelle “$\frac{n+1}{2}$” plaziert ist.

- Ist “n” eine *gerade* Zahl, so ergibt sich der Median dadurch, daß der Durchschnittswert der beiden Werte gebildet wird, die innerhalb der Rangreihe an den Positionen “$\frac{n}{2}$” und “$\frac{n}{2} + 1$” positioniert sind.

Auf der Basis dieser Berechnungsvorschrift können wir die Grundidee zur Lösung der Problemstellung

- Bestimme den Median der durch “WerteErfassung11” erfaßten Werte, d.h. den Wert, der sich als mittlerer Wert auf der Basis einer aufsteigenden Sortierordnung ergibt!

wie folgt skizzieren:

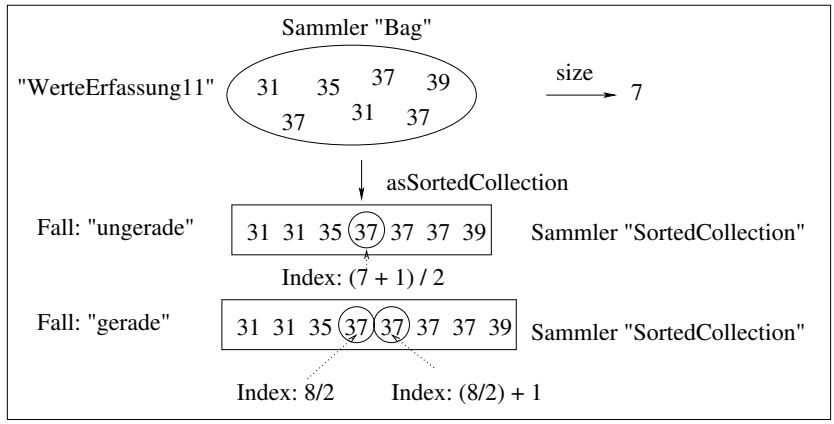


Abbildung 6.8: Grundidee zur Berechnung eines Medians

Auf der Basis dieser Skizze läßt sich der Lösungsplan durch das folgende Struktogramm angeben:

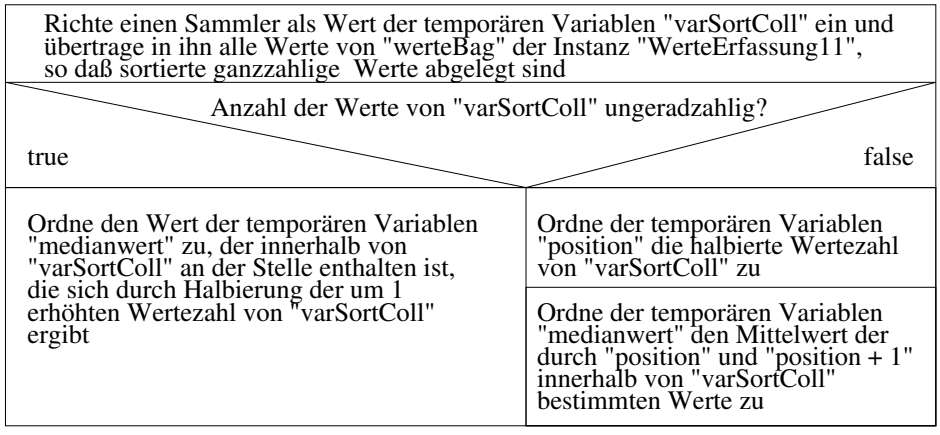


Abbildung 6.9: Struktogramm zur Berechnung eines Medians

Indem wir auf die zuvor vorgestellten Basis-Methoden “asSortedCollection”, “odd”, “ifTrue:ifFalse:” und “at:” zurückgreifen, können wir das Struktogramm wie folgt umformen:

```
|varSortColl position medianwert|
varSortColl := (WerteErfassung11 bereitstellenWerte
                  collect: [:einObjekt|einObjekt asInteger])
                  asSortedCollection.
varSortColl size odd
 ifTrue: [medianwert:=varSortColl at:(varSortColl size)+1/2]
```

```
ifFalse:[position:=(varSortColl size)/2 .
         medianwert:=((varSortColl at: position) asInteger
           + (varSortColl at: position + 1) asInteger)/2]
```

Hinweis: Es ist wichtig, daß wir vor dem Punkt, mit dem die 1. Block-Anforderung des Arguments von "ifFalse:" abgeschlossen wird, mindestens ein Leerzeichen aufführen. Würden wir

```
position:=(varSortColl size)/2.
```

angeben, so würde ein Walkback-Fenster eröffnet. Dies liegt daran, daß "2." keine ganze Zahl ist und beim Einsatz von "at:" nur ganzahlige Argumente als Index-Position zulässig sind.

6.7.5 Prüfung einer Häufigkeit

Zur Lösung der Problemstellung

- Bestimme, ob die Häufigkeit, mit der der Punktwert "35" in der Instanz "WerteErfassung11" auftritt, größer als 3 ist!

läßt sich die folgende Anforderung zur Ausführung bringen:

```
(WerteErfassung11 bereitstellenWerte occurrencesOf: '35') > 3
```

Hierbei haben wir die Basis-Methode "occurrencesOf:" eingesetzt, mit der sich die Häufigkeit feststellen läßt, mit der ein Objekt in einem Sammler enthalten ist.

- **"occurrencesOf:"**:
  Durch die Ausführung der Basis-Methode "occurrencesOf:" wird die Häufigkeit ermittelt, mit der das innerhalb der Keyword-Message "occurrencesOf:" angegebene Argument in dem Sammler enthalten ist, der als Empfänger-Objekt der Message "occurrencesOf:" aufgeführt ist.

6.7.6 Berechnung eines Modus

Der statistische Kennwert, der den Wert mit der größten Häufigkeit charakterisiert, wird *Modus* genannt.
Zur Lösung der Problemstellung

- Bestimme den Modus der durch "WerteErfassung11" erfaßten Werte!

läßt sich die folgende Grundidee skizzieren:

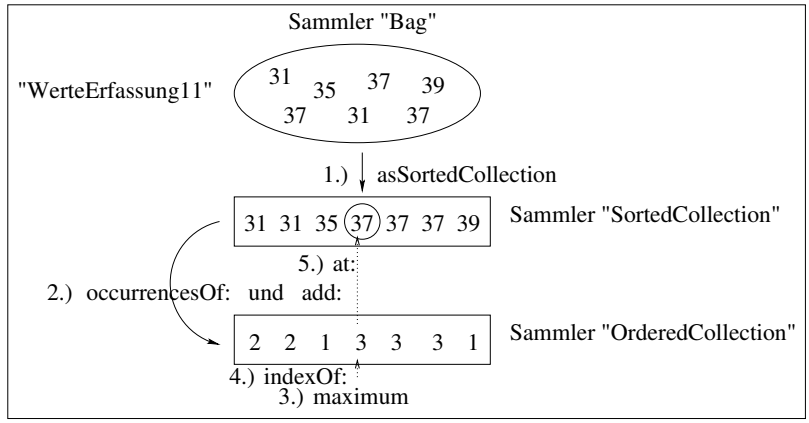


Abbildung 6.10: Grundidee zur Bestimmung eines Modus

Auf der Basis dieser Skizze läßt sich der Lösungsplan in das folgende Struktogramm umsetzen:

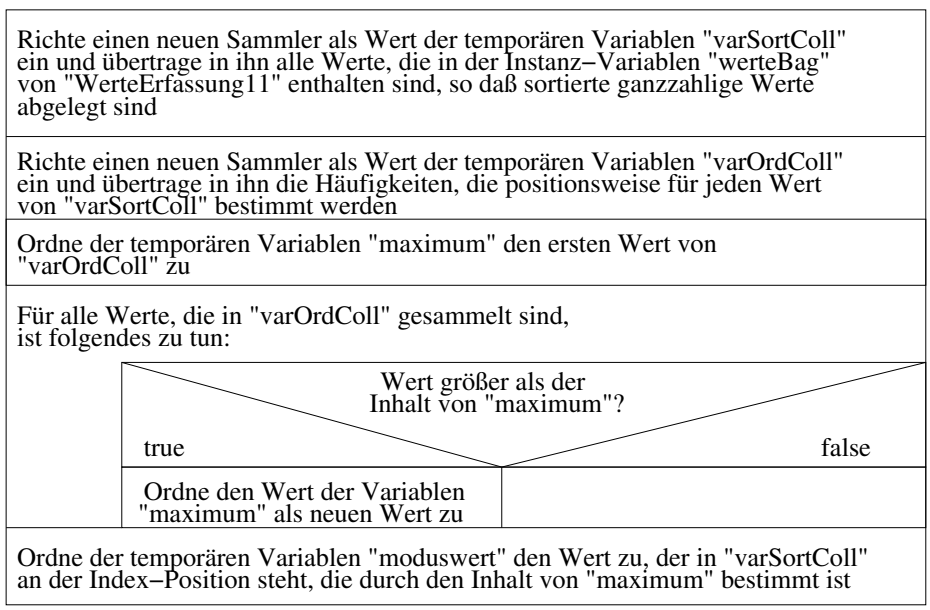


Abbildung 6.11: Struktogramm zur Berechnung eines Modus

Unter Einsatz der Basis-Methoden “asSortedCollection”, “collect:”, “occurrencesOf:”, “at:”, “>”, “ifTrue:” und “indexOf:” läßt sich das Struktogramm wie folgt umsetzen:

```
|varSortColl varOrdColl maximum moduswert|
varSortColl := (WerteErfassung11 bereitstellenWerte
                   collect: [:einObjekt|einObjekt asInteger])
                   asSortedCollection.
varOrdColl := OrderedCollection new.
varSortColl do: [:einObjekt|varOrdColl add:
                  (varSortColl occurrencesOf: einObjekt)].
maximum := varOrdColl at: 1.
varOrdColl do: [:einObjekt|einObjekt > maximum
                           ifTrue: [maximum := einObjekt]].
moduswert := varSortColl at: (varOrdColl indexOf: maximum)
```

Kapitel 7

Hierarchische Gliederung von Lösungsplänen

7.1 Methoden-Vereinbarung zur Berechnung und Sicherung eines Medians

Im Abschnitt 5.3 haben wir die Klasse "InWerteErfassung", für deren Instanzen der Durchschnittswert der erfaßten Werte mit der Methode "durchschnitt" ermittelt werden kann, als Unterklasse von "WerteErfassung" eingerichtet.

Damit sich in unserer Situation ein Durchschnittswert als geeignete Schätzung für das Zentrum der Daten sinnvoll interpretieren läßt, muß gesichert sein, daß Wertedifferenzen empirisch bedeutsam sind, d.h. gleiche Wertedifferenzen müssen auch gleiche Leistungsunterschiede widerspiegeln. Sofern die Daten dieses Qualitätskriterium erfüllen, spricht man von *intervallskalierten* Daten.

Da wir bislang bei unserer Datenerfassung von intervallskalierten Daten ausgehen konnten, kann der Durchschnittswert als gute Annäherung an das *Zentrum* der Punktwerte angesehen werden. Deshalb haben wir bei der Lösung von PROB-1 derjenigen Klasse, die wir der Klasse "WerteErfassung" untergeordnet haben, den Namen "InWerteErfassung" gegeben.

Hinweis: Die Vorsilbe "In" wird als Abkürzung von "intervallskaliert" verwendet.

Erfüllen die erfaßten Werte die Forderung, daß die Wertedifferenzen empirisch bedeutsam sind, nicht, sondern spiegeln die Werte allein eine *Ordnungsbeziehung* zwischen den Schülern wider (z.B.: ein Schüler ist "leistungsfähiger" als ein anderer Schüler), so handelt es sich um *ordinalskalierte* Daten.

Ordinalskalierte Daten liegen z.B. auch dann vor, wenn die Schüler einen Turnwettkampf durchführen und die dabei erhaltenen Bewertungen in Form von (ganzzahligen) Punktwerten festgelegt sind. In dieser Situation kann man den *Median* als geeignete Schätzung des Zentrums verwenden (siehe Abschnitt 6.7.4).

Da wir im Hinblick auf die Schätzung des Zentrums darauf achten wollen, ob es sich um intervallskalierte oder nur um ordinalskalierte Daten handelt, richten wir eine *eigenständige* Klasse zur Lösung der folgenden Problemstellung ein:

- PROB-2:
  Auf der Basis der erfaßten Werte soll der Median errechnet und als Inhalt einer Instanz-Variablen gesichert werden, so daß er jederzeit abgerufen werden kann!

Im Hinblick auf unsere bisherige Vorgehensweise erscheint es sinnvoll, für den Lösungsplan eine *neue* Klasse namens "OrWerteErfassung" zu verabreden, die – ebenfalls wie "InWerteErfassung" – als Unterklasse von "WerteErfassung" festgelegt werden soll.

Hinweis: Die Vorsilbe "Or" wird als Abkürzung von "ordinalskaliert" verwendet.

Zur Sicherung eines errechneten Medians sehen wir die Instanz-Variable "medianwert" vor, so daß wir die Klasse "OrWerteErfassung" – mittels des Klassen-Hierarchie-Browser-Fensters – wie folgt vereinbaren:

```
WerteErfassung subclass: #OrWerteErfassung
  instanceVariableNames: 'medianwert'
  classVariableNames: ''
  poolDictionaries: ''
```

Auf der Basis der Instanz-Variablen "medianwert" legen wir für die Klasse "OrWerteErfassung" eine Methode namens "median" in der folgenden Form fest:

```
median
|varSortColl position|
varSortColl := (self bereitstellenWerte
                  collect: [:einObjekt|einObjekt asInteger])
                  asSortedCollection.
varSortColl size odd
  ifTrue: [medianwert:=varSortColl at: (varSortColl size)+1/2]
  ifFalse:[position:=(varSortColl size)/2 .
           medianwert:=((varSortColl at: position) asInteger
             + (varSortColl at: position + 1) asInteger)/2]
```

Diese Vereinbarung gründet sich auf den Lösungsplan, den wir im Abschnitt 6.7.4 entwickelt haben.

Hinweis: Es ist zu beachten, daß die Variable "medianwert" im ursprünglichen Lösungsplan als temporäre Variable verwendet wurde. Desweiteren war an der Stelle, an der die Pseudovariable "self" aufgeführt ist, ursprünglich eine konkrete Instanz der Klasse "WerteErfassung" – in Form von "WerteErfassung11" – angegeben.

Um sich den *gesicherten* Median im Transcript-Fenster anzeigen zu lassen, legen wir die Methode "anzeigenMedianwert" wie folgt fest:

```
anzeigenMedianwert
Transcript cr;
   show: 'Der Median ist: ';
   show: medianwert asFloat printString
```

Somit liegt insgesamt die folgende Situation vor:

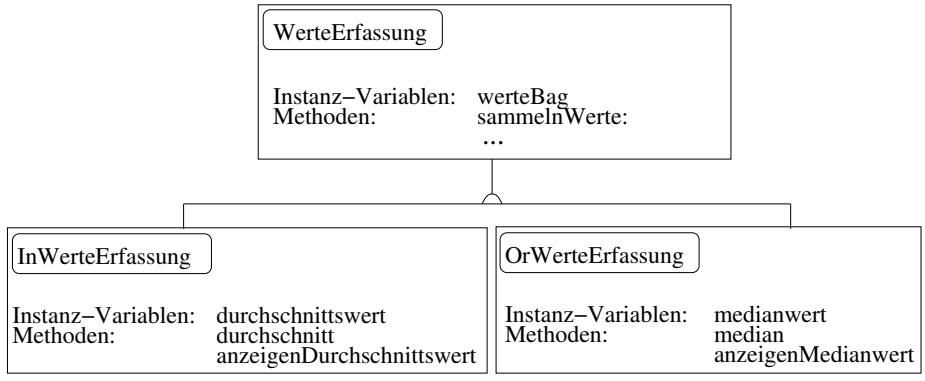


Abbildung 7.1: "InWerteErfassung" und "OrWerteErfassung"
auf derselben Hierarchiestufe

Hierdurch ist erkennbar, daß jede Instanz der Klassen "InWerteErfassung" und "OrWerteErfassung" die Instanz-Variable "werteBag" erbt. Zusätzlich besitzt eine Instanz von "InWerteErfassung" die Instanz-Variable "durchschnittswert" und eine Instanz von "OrWerteErfassung" die Instanz-Variable "medianwert".
Sofern die Punktwerte auf der Basis der Anforderungen

```
OrWerteErfassung11 := OrWerteErfassung new.
OrWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

erfaßt wurden, läßt sich der Median durch die Anforderung

```
OrWerteErfassung11 median; anzeigenMedianwert
```

errechnen und im Transcript-Fenster anzeigen.

7.2 Polymorphe Wirkung von Messages

Unabhängig davon, ob das *Zentrum* der Werte in Form des Durchschnittswertes oder des Medians ermittelt werden soll, ist es wünschenswert, einen *einheitlichen* Namen wie z.B. "zentrum" zur Kennzeichnung derjenigen Methode zu verwenden, mit der sich das Zentrum der Werte abrufen läßt.
Wir betrachten daher die folgende Problemstellung:

- PROB-3:
  Die Berechnung des Durchschnittswertes für intervallskalierte Daten und die Berechnung des Medians für ordinalskalierte Daten soll *einheitlich* mittels des Message-Selektors "zentrum" angefordert werden können. Ob der Durchschnittswert oder der Median zu ermitteln ist, soll dadurch gesteuert werden, daß die Methode "zentrum" von einer Instanz der Klasse "InWerteErfassung" oder einer Instanz der Klasse "OrWerteErfassung" zur Ausführung gebracht wird.

Innerhalb des Lösungsplans ist daher festzulegen, daß durch den Message-Selektor "zentrum" die Methode "durchschnitt" dann auszuführen ist, wenn das Empfänger-Objekt der Message "zentrum" eine Instanz der Klasse "InWerteErfassung" ist. Wenn es sich dagegen beim Empfänger-Objekt der Message "zentrum" um eine Instanz der Klasse "OrWerteErfassung" handelt, ist die Methode "median" auszuführen.
Diese Forderung ist erfüllbar, weil SMALLTALK den *Polymorphismus* (Mehrgestaltigkeit) von Messages wie folgt unterstützt:

- Da das Empfänger-Objekt einer Message die korrespondierende Methode zunächst innerhalb der Klasse sucht, zu deren Instanzen dieses Objekt zählt, sind *gleichnamige* Methoden innerhalb *unterschiedlicher* Klassen möglich.

  Ein und dieselbe Message kann daher die Ausführung völlig unterschiedlicher Methoden bewirken – je nachdem, aus welcher Klasse das Empfänger-Objekt der Message instanziiert wurde.

  Die polymorphe Wirkung einer Message beruht somit darauf, daß *dynamisch*, d.h. zum Zeitpunkt der Ausführung einer Anforderung, die jeweilige Methode durch das Empfänger-Objekt bestimmt wird. Dabei wird der Ausgangspunkt für die Suche der Methode durch diejenige Klasse festgelegt, aus der das Empfänger-Objekt der Message instanziiert wurde.

- Sofern gleichnamige Methoden in einander hierarchisch untergeordneten Klassen vorliegen, gilt:
  Die Methode, die in einer untergeordneten Klasse vereinbart ist, *überdeckt* jede gleichnamige Methode, die innerhalb einer übergeordneten Klasse festgelegt ist. Dies liegt daran, daß vom Empfänger-Objekt *immer* die zuerst identifizierte Methode ausgeführt wird, unabhängig davon, ob eine *gleichnamige* Methode in einer überordneten Klasse vereinbart ist.

Hinweis: Unter *Polymorphismus* ist allgemein die Eigenschaft zu verstehen, daß ein Name verschiedene Dinge mit unterschiedlichen Formen bezeichnen kann.

Da wir die Methode "durchschnitt" in der Klasse "InWerteErfassung" festgelegt haben, können wir die Berechnung des Durchschnittwertes mittels der Message "zentrum" dadurch erreichen, daß wir die Methode "zentrum" wie folgt innerhalb der Klasse "InWerteErfassung" vereinbaren:

```
zentrum
self durchschnitt
```

Zur Berechnung des Medians – mittels der Message "zentrum" – läßt sich entsprechend verfahren. Dazu ist die Methode "zentrum" wie folgt innerhalb der Klasse "OrWerteErfassung" zu vereinbaren:

```
zentrum
self median
```

Entsprechend diesem Vorgehen wollen wir auch die Anzeige des jeweils errechneten Zentrums über eine Message – namens "anzeigenZentrum" – anfordern.
Um die Anzeige eines bereits errechneten Durchschnittwertes bzw. Medians *einheitlich* durch den Message-Selektor "anzeigenZentrum" abrufen zu können, legen wir die Methode "anzeigenZentrum" in der Klasse "InWerteErfassung" durch die Vereinbarung

```
anzeigenZentrum
self anzeigenDurchschnittswert
```

fest. Entsprechend vereinbaren wir die Methode "anzeigenZentrum" wie folgt in der Klasse "OrWerteErfassung":

```
anzeigenZentrum
self anzeigenMedianwert
```

Insgesamt resultiert der folgende Sachverhalt:

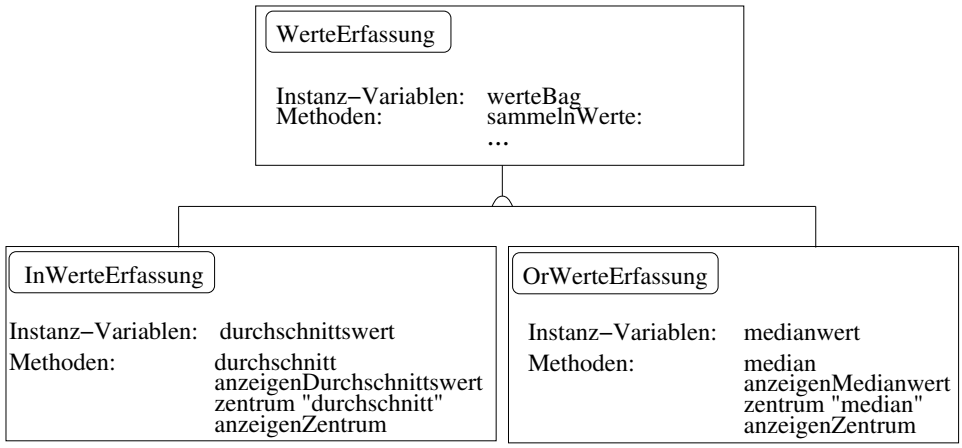


Abbildung 7.2: Basis für die polymorphe Wirkung von Messages

Hinweis: Um zu dokumentieren, daß mit dem Methoden-Selektor "zentrum" der Durchschnittswert bzw. der Median errechnet werden kann, sind in der Grafik die Kommentar-Informationen `"durchschnitt"` sowie `"median"` hinter den betreffenden Methoden-Selektoren aufgeführt.

Sofern einer Instanz der Klasse "InWerteErfassung" die Message "zentrum" zugestellt wird, führt dies zur Ausführung der Methode "zentrum" und daher zur Ausführung der Anforderung "self durchschnitt", d.h. die jeweilige Instanz sendet sich selbst die Message "durchschnitt". Da die Methode "durchschnitt" innerhalb der Klasse "InWerteErfassung" identifiziert wird, führt die Message "zentrum" folglich zur Berechnung des Durchschnittwertes.

Entsprechend läßt sich von einer Instanz der Klasse "OrWerteErfassung" die Berechnung des Medians über die Message "zentrum" abrufen.

Genauso wie sich mit der Message “zentrum” der Durchschnittswert bzw. der Median ermitteln läßt, kann über die Methode “anzeigenZentrum” die Anzeige des jeweils berechneten Zentrums im Transcript-Fenster abgerufen werden.
Die Erfassung der Punktwerte und die Anzeige des Durchschnittswertes können wir – auf der Basis von *intervallskalierten* Daten – z.B. durch

```
InWerteErfassung11 := InWerteErfassung new.
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

und

```
InWerteErfassung11 zentrum; anzeigenZentrum
```

anfordern.
Gehen wir von *ordinalskalierten* Daten aus, so können wir die Erfassung durch die Anforderungen

```
OrWerteErfassung11 := OrWerteErfassung new.
OrWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

und die Berechnung des Medians durch die Anforderung

```
OrWerteErfassung11 zentrum; anzeigenZentrum
```

abrufen.
Nach der Berechnung des Zentrums kann die Anzeige der jeweils gesicherten statistischen Kennzahlen entweder durch

```
InWerteErfassung11 anzeigenZentrum
```

oder durch

```
OrWerteErfassung11 anzeigenZentrum
```

angefordert werden.

7.3 Methoden-Vereinbarung zur Berechnung und Sicherung eines Modus

In bestimmten Fällen kann es vorkommen, daß die statistischen Kennwerte “Durchschnittswert” und “Median” für eine sinnvolle Berechnung eines Zentrums *ungeeignet* sind.
Dies ist z.B. dann der Fall, wenn durch die erfaßten (ganzzahligen) Werte die Stilart gekennzeichnet wird, mit der ein Schüler seinen Schwimmwettkampf bestritten hat – z.B. “1” für Brustschwimmen, “2” für Rückenschwimmen und “3” für Schmetterlingschwimmen.
In dieser Situation kennzeichnen die Werte unterschiedliche Schwimmstile, so daß für diese Werte keine Vergleiche sinnvoll sind. Derartige Daten, die allein eine *Gruppenzugehörigkeit* beschreiben, werden als *nominalskalierte* Daten bezeichnet. Für sie ist es z.B. sinnvoll, das Zentrum durch den statistischen Kennwert *Modus*, d.h. durch den häufigsten Wert, zu beschreiben.
Im Hinblick auf diesen Sachverhalt betrachten wir die folgende Problemstellung:

- PROB-4:
  Auf der Basis der erfaßten Werte soll der Modus ermittelt und als Inhalt einer Instanz-Variablen gesichert werden, so daß er jederzeit angezeigt werden kann!

Auf der Basis der Lösung PROB-3 ist es sinnvoll, nominalskalierte Daten mittels einer Instanz zu erfassen, die aus einer *neu* einzurichtenden Unterklasse von "WerteErfassung" namens "NoWerteErfassung" stammt.

Hinweis: Die Vorsilbe "No" wird als Abkürzung von "nominalskaliert" verwendet.

Dieser Ansatz führt zur folgenden Konzeption:

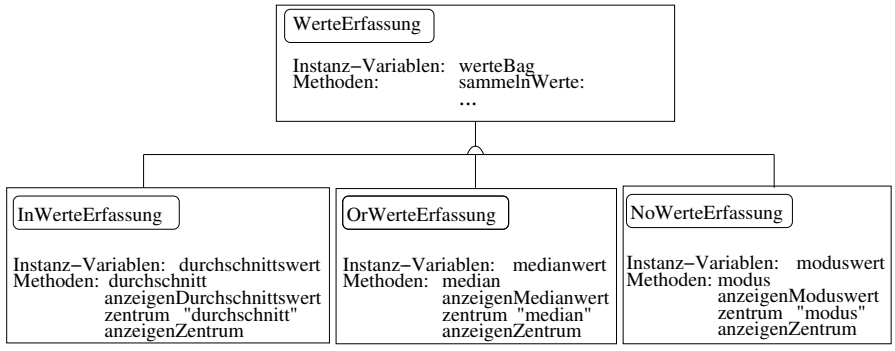


Abbildung 7.3: Erweiterung polymorpher Wirkungen von Messages

Hieraus ist erkennbar, daß innerhalb der Klasse "NoWerteErfassung" eine Instanz-Variable namens "moduswert" verabredet werden soll, durch die ein errechneter Modus gesichert werden kann.
Aus Konsistenzgründen zur Lösung von PROB-3 ist eine Methode "zentrum" aufgeführt, mit der die Methode "modus" ausgeführt werden soll, falls die Message "zentrum" einer Instanz von "NoWerteErfassung" zugestellt wird. Entsprechend ist eine Methode namens "anzeigenModuswert" sowie – ebenfalls aus Konsistenzgründen – die Methode "anzeigenZentrum" vorgesehen, mit denen sich der Modus im Transcript-Fenster anzeigen lassen können soll.
Bei der Vereinbarung der Klasse "NoWerteErfassung", die wir innerhalb des Klassen-Hierarchie-Browser-Fensters als Unterklasse von "WerteErfassung" vornehmen, müssen wir die Instanz-Variable "moduswert" folgendermaßen aufführen:

```
WerteErfassung subclass: #NoWerteErfassung
  instanceVariableNames: 'moduswert'
  classVariableNames: ''
  poolDictionaries: ''
```

Indem wir die Vereinbarung der Methode "modus" auf denjenigen Lösungsplan gründen, den wir im Abschnitt 6.7.6 angegeben haben, können wir auf der Basis der Instanz-Variablen "moduswert" die folgende Vereinbarung vornehmen:

```
modus
|varSortColl varOrdColl maximum|
varSortColl := (self bereitstellenWerte
                  collect: [:einObjekt|einObjekt asInteger])
                  asSortedCollection.
varOrdColl := OrderedCollection new.
varSortColl do: [:einObjekt|varOrdColl add:
                  (varSortColl occurrencesOf: einObjekt)].
maximum := varOrdColl at: 1.
varOrdColl do: [:einObjekt|einObjekt > maximum
                           ifTrue: [maximum := einObjekt]].
moduswert := varSortColl at: (varOrdColl indexOf: maximum)
```

Hinweis: Dabei ist zu beachten, daß die Variable "moduswert" ursprünglich als temporäre Variable verwendet wurde. Desweiteren war an der Stelle, an der die Pseudovariable "self" aufgeführt ist, ursprünglich eine konkrete Instanz der Klasse "WerteErfassung" angegeben.

Auf dieser Basis können wir die Methode "zentrum" anschließend wie folgt festlegen:

```
zentrum
self modus
```

Für eine Instanz von "NoWerteErfassung" läßt sich der Modus durch die Ausführung der Methode "modus" bzw. der Methode "zentrum" errechnen und der Instanz-Variablen "moduswert" zuordnen.
Um sich den derart *gesicherten* Modus im Transcript-Fenster anzeigen zu lassen, legen wir die Methode "anzeigenModuswert" durch die Vereinbarung

```
anzeigenModuswert
Transcript cr;
   show: 'Der Modus ist: ';
   show: moduswert asFloat printString
```

und – aus Konsistenzgründen – zusätzlich die Methode "anzeigenZentrum" durch die Vereinbarung

```
anzeigenZentrum
self anzeigenModuswert
```

innerhalb der Klasse "NoWerteErfassung" fest.
Die Erfassung *nominalskalierter* Werte ist durch die Anforderungen

```
NoWerteErfassung11 := NoWerteErfassung new.
NoWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

und die anschließende Berechnung des Modus durch

```
NoWerteErfassung11 zentrum
```

abrufbar.
Damit der *gesicherte* Modus im Transcript-Fenster angezeigt wird, können wir entweder die Anforderung

```
NoWerteErfassung11 anzeigenModuswert
```

oder die Anforderung

```
NoWerteErfassung11 anzeigenZentrum
```

stellen.

7.4 Überdecken von Methoden ("super")

Bei unserem bisherigen Vorgehen haben wir die Entscheidung, ob eine Instanziierung aus der Klasse "InWerteErfassung", "OrWerteErfassung" oder "NoWerteErfassung" vorgenommen werden soll, darauf gegründet, ob es sich bei den Werten um intervall-, ordinal- oder um nominalskalierte Daten handelt. Der bisherige Lösungsansatz ist *vorteilhaft*, um die jeweilige statistische Kenngröße, mit der das Zentrum beschrieben werden soll, über einen einzigen Methodennamen ("zentrum") anfordern zu können. Als *Nachteil* ist allerdings festzustellen, daß sich die anderen statistischen Kennwerte nicht unmittelbar abrufen lassen.

Aus statistischer Sicht ist es in bestimmten Situationen auch für intervallskalierte Daten wünschenswert, den Median und den Modus zu errechnen. Entsprechend besteht eventuell ein Interesse, für ordinalskalierte Daten nicht nur den Median, sondern ebenfalls den Modus zu ermitteln, so daß wir den folgenden Sachverhalt angeben können:

	Beschreibung des Zentrums durch:
Intervallskalierte Daten	Durchschnittswert
Ordinalskalierte Daten	Median
Nominalskalierte Daten	Modus

	Sinnvolle statistische Kennzahlen:		
	Durchschnittswert	Median	Modus
Intervallskalierte Daten	√	√	√
Ordinalskalierte Daten		√	√
Nominalskalierte Daten			√

Um die Berechnung der statistischen Kennzahlen in dieser Form zugänglich zu machen, müßten wir – nach dem jetzigen Stand der Klassen-Hierarchie – in der Klasse "InWerteErfassung" die Methoden "median" und "modus" und in der Klasse "OrWerteErfassung" die Methode "modus" *zusätzlich* vereinbaren.
Dies ist *nicht* sinnvoll, da redundante Methoden-Vereinbarungen vorliegen würden und wir eine möglichst *redundanzfreie* Vereinbarung von Klassen anstreben. Im Hinblick auf diese Forderung wollen wir daher die folgende Problemstellung lösen:

- PROB-5:
  Auf der Basis der erfaßten Werte soll – mittels einer geeigneten Instanziierung – einheitlich über den Message-Selektor "zentrum" der jeweils charakteristische statistische Kennwert angefordert werden können. Ergänzend sollen der Modus, der Median und der Durchschnittswert immer dann, wenn die jeweilige statistische Kennzahl sinnvoll ist, auch *gezielt* abrufbar sein.

Um eine Lösung zu entwickeln, nutzen wir die Möglichkeit, daß sich eine Methode innerhalb einer Oberklasse durch eine gleichnamige Methode einer untergeordneten Klasse *überdecken* läßt. Dabei machen wir von der Möglichkeit Gebrauch, daß sich zur Kommunikation eines Objektes mit sich selbst – neben der Pseudovariablen "self" – eine *Pseudovariable* namens "super" verwenden läßt.

Hinweis: Die Pseudovariable "super" ergänzt die Liste der bislang vorgestellten Pseudovariablen "self", "true", "false" und "nil".

- Die Pseudovariable *"super"* darf – genauso wie die Pseudovariable "self" – als Platzhalter innerhalb der Anforderungen einer Methode verwendet werden.
  Soll eine Methode ausgeführt werden, in der "super" aufgeführt ist, so wird "super" durch das Empfänger-Objekt der Message ersetzt, durch die die Methode zur Ausführung gelangen soll.

- Das Empfänger-Objekt, das den Platz von "super" bei der Ausführung einnimmt, beginnt die Suche nach der auszuführenden Methode – im Gegensatz zur Situation beim Einsatz von "self" – *nicht* innerhalb der Klasse, aus der es instanziiert wurde, sondern in dessen "Super-Klasse". Diese Klasse ist dadurch bestimmt, daß sie innerhalb der Klassen-Hierarchie gleich der direkten Oberklasse derjenigen Klasse ist, in der die Methode identifiziert wird, in deren Anforderungen das Empfänger-Objekt den Platz von "super" einnimmt.

Hinweis: Es ist zu beachten, daß die Pseudovariablen "super" und "self" *immer* das Empfänger-Objekt der auszuführenden Methode bezeichnen. Sie unterscheiden sich lediglich durch die Klasse, ab der die Methodensuche beginnt.
Im Unterschied zur Pseudovariablen "super" ist es beim Einsatz von "self" möglich, die Pseudovariable "self" als Ergebnis-Objekt einer Methode festzulegen. Auch ist es möglich, "self" als Argument einer Keyword-Message aufzuführen oder innerhalb einer Methode einer Variablen zuzuweisen. Die Pseudovariable "self" sollte nur dann als Platzhalter eingesetzt werden, wenn die auszuführende Methode auch tatsächlich innerhalb der Klasse vereinbart ist, aus der das Empfänger-Objekt instanziiert wurde. Andernfalls ist die Pseudovariable "super" zu verwenden.

Die Pseudovariable "super" läßt sich z.B. dann einsetzen, wenn wir in mehreren Klassen, die innerhalb der Klassen-Hierarchie einander untergeordnet sind, *gleichnamige* Methoden-Selektoren derart verwenden wollen, daß eine Methode durch eine gleichnamige Methode einer Oberklasse *überdeckt* werden soll.
Um diesen Sachverhalt zu veranschaulichen, betrachten wir die folgende Situation:

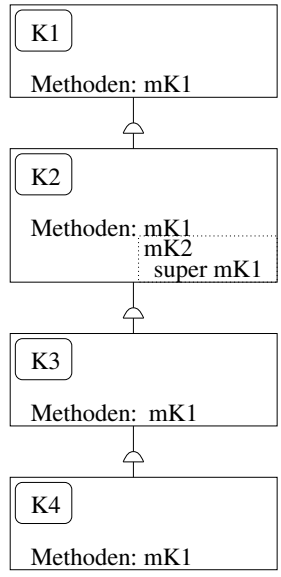


Abbildung 7.4: Identifikation von Methoden beim Einsatz von "super"

Ist "I" eine Instanz von "K4", und soll "I" die Methode "mK2" ausführen, so wird "mK2" innerhalb von "K2" identifiziert. Die in dieser Methode enthaltene Anforderung besagt, daß "I" die Methode "mK1" ausführen soll. Obwohl "mK1" sowohl in der Klasse "K3" als auch in der Klasse "K4" festgelegt ist, wird die Methode "mK1" von "I" innerhalb der Klasse "K1" identifiziert.

Dies liegt daran, daß – durch die Verwendung der *Pseudovariablen "super"* – die auszuführende Methode "mK1" von "I" ab der "Super-Klasse" von K2, d.h. ab "K1" gesucht wird.

Hinweis: Sofern wir die in der Klasse "K2" vereinbarte Methode "mK2" in der Form

```
mK2
self mK1
```

abändern, würde durch die Anforderung
`I mK2`
die Methode "mK1" der Klasse "K4" ausgeführt. Dies liegt daran, daß sich die Pseudovariable "self" – genauso wie "super" – *immer* auf das Empfänger-Objekt der ursprünglichen Message bezieht.
Vereinbaren wir diese vier Klassen dagegen in der Form

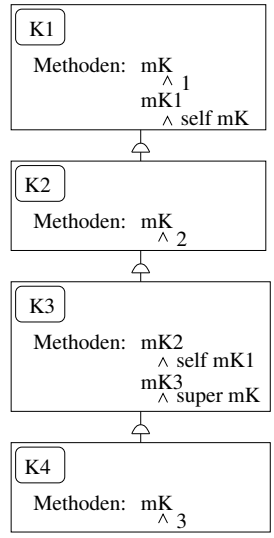


Abbildung 7.5: Identifikation von Methoden beim Einsatz von "self" und "super"

so erhalten wir auf der Basis von

```
I3 := K3 new.
I4 := K4 new
```

als Ergebnis-Objekte der Anforderungen

```
I3 mK3.
I4 mK3
```

jeweils den Wert "2". Dies liegt daran, daß in beiden Fällen die Methode "mK" ab der Oberklasse von "K3" bzw. "K4" gesucht und unmittelbar in der Klasse "K2" gefunden wird.
Anders ist dies bei der Anforderung

```
I4 mK1
```

mit dem Ergebnis-Objekt "3". In dieser Situation wird zunächst die Methode "mK1" der Klasse "K1" ausgeführt. Da die Pseudovariable "self" in diesem Fall das Empfänger-Objekt der Message "mK1" (Instanz der Klasse "K4") kennzeichnet, wird die Methode "mK" anschließend ab der Klasse "K4" gesucht. Sie wird in der Klasse "K4" gefunden und abschließend ausgeführt.

Der Einsatz der Pseudovariablen "super" ist insbesondere dann sinnvoll, wenn zur Lösung von ähnlichen Aufgabenstellungen gleichnamige Methoden verwendet werden sollen, wobei die Methode aus einer Unterklasse das gleiche wie die gleichnamige Methode aus der zugehörigen Oberklasse leisten und zusätzlich noch weitere Aufgaben erfüllen soll (siehe Abschnitt 8.2).

Zur Lösung von PROB-5 sehen wir – unter Einsatz der Pseudovariablen "super" – eine hierarchische Stufung der Klassen "NoWerteErfassung", "OrWerteErfassung" und "InWerteErfassung" als Unterklassen von "WerteErfassung" in der folgenden Form vor:

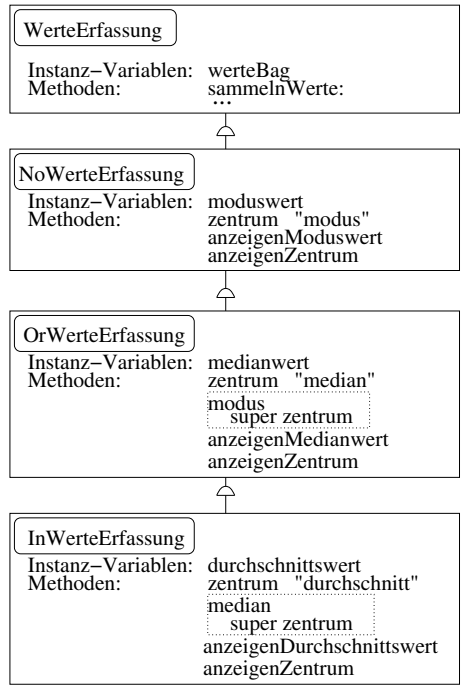


Abbildung 7.6: Überdeckung der Methode "zentrum"

Auf der Basis dieser Verschachtelung enthält eine Instanz aus einer dieser Klassen eine oder mehrere der klassen-spezifisch aufgeführten Instanz-Variablen. Welche Instanz-Variablen zu welchen Instanzen gehören, läßt sich wie folgt skizzieren:

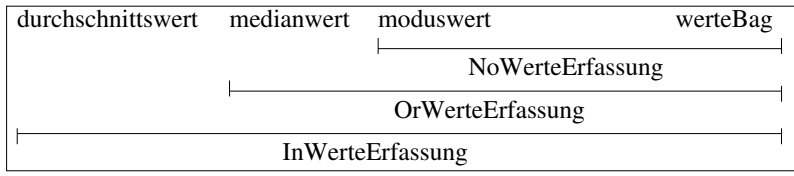


Abbildung 7.7: Instanz-Variablen von Instanzen

Falls wir die angegebene Klassen-Hierarchie mittels des Klassen-Hierarchie-Browser-Fensters vereinbaren wollen, müssen wir im Hinblick auf die aktuelle Klassen-Hierarchie zunächst die Voraussetzungen für eine derartige Vereinbarung schaffen.

Hinweis: Da Klassennamen eindeutig vergeben werden müssen, kann die Lösung nicht auf der bislang aufgebauten Klassen-Hierarchie basieren, es sei denn, es würden andere als die von uns vorgeschlagenen Klassennamen verwendet.

Dazu sind zunächst die bislang als direkte Unterklassen von "WerteErfassung" vereinbarten Klassen "InWerteErfassung" und "OrWerteErfassung" – unter Einsatz der Menü-Option "Remove Class" des Menüs "Classes" – schrittweise zu löschen.

Hinweis: Es ist zu beachten, daß sämtliche Instanzen der zu löschenden Klassen zuvor ebenfalls gelöscht werden müssen, indem die Pseudovariable "nil" – mittels einer Zuweisung wie z.B. "InWerteErfassung11 := nil" – geeignet zugeordnet wird.
Wie wir alle Instanzen einer Klasse anzeigen und löschen lassen können, geben wir in Abschnitt 9.4.2 an.

Wird anschließend die Klasse "OrWerteErfassung" als unmittelbare Unterklasse von "NoWerteErfassung" eingerichtet und die Methode "zentrum" in dieser Klasse festgelegt, so erscheint die Meldung "You are redefining a superclass method". Diese Meldung weist darauf hin, daß eine gleichnamige Methode in einer übergeordneten Klasse *überdeckt* wird. Da wir dies beabsichtigen, bestätigen wir diese Meldung.
Entsprechend gehen wir vor, wenn wir "InWerteErfassung" als Unterklasse von "OrWerteErfassung" vereinbart haben und die Methode "zentrum" innerhalb der Klasse "InWerteErfassung" festgelegt werden soll.
Auf der Basis der oben angegebenen Klassen-Hierarchie ist – genau wie beim Lösungsplan von PROB-4 – die Ausführung der Methode "zentrum" abhängig davon, durch welche Instanziierung das Empfänger-Objekt der Message "zentrum" zuvor eingerichtet wurde.
Wie angestrebt, ist es gegenüber der Lösung von PROB-4 jetzt auch zusätzlich möglich, den Median und den Modus für intervallskalierte Daten und den Modus für ordinalskalierte Daten anzufordern, obwohl diese Methoden jeweils nur innerhalb einer einzigen Klasse vereinbart sind.
Sobald z.B. der Modus für ordinalskalierte Daten ermittelt werden soll, muß er für eine Instanz von "OrWerteErfassung" über die Message "modus" angefordert werden. Hierdurch gelangt die durch

```
modus
super zentrum
```

innerhalb der Klasse "OrWerteErfassung" vereinbarte Methode "modus" zur Ausführung. Demzufolge wird – wegen der Verwendung der Pseudovariablen "super" – die Methode "zentrum" nicht in der Klasse "OrWerteErfassung", sondern in der zugehörigen Super-Klasse gesucht. Da es sich bei dieser Super-Klasse um die Klasse "NoWerteErfassung" handelt, gelangt die dort vereinbarte Methode "zentrum" zur Ausführung, so daß der Modus für die zuvor erfaßten Daten der Instanz errechnet wird.
Sofern z.B. für intervallskalierte Daten der Median errechnet werden soll, ist die Message "median" an eine Instanz von "InWerteErfassung" zu richten. Dadurch wird die Methode "median", die in der Form

```
median
super zentrum
```

innerhalb der Klasse "InWerteErfassung" vereinbart ist, und folglich die Methode "zentrum" aus der Klasse "OrWerteErfassung" ausgeführt.
Für eine Instanz von "InWerteErfassung" kann zusätzlich der Modus über den Aufruf der Methode "modus" ermittelt werden. Diese Methode wird in der übergeordneten Klasse "OrWerteErfassung" identifiziert, so daß mittels der Pseudovariablen "super" die Methode "zentrum" der Klasse "NoWerteErfassung" zur Ausführung gelangt.

Insgesamt kann die Datenerfassung und die Ermittlung der jeweils sinnvollen statistischen Kennwerte und deren Anzeige – im Hinblick auf verschiedenartige Punktwerte – z.B. wie folgt im Workspace-Fenster angefordert werden:
Zunächst sind die Anforderungen

```
InWerteErfassung11 := InWerteErfassung new.
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

anschließend die Anforderungen

```
InWerteErfassung11 zentrum "Durchschnitt";
  median "Median"; modus "Modus";
  anzeigenDurchschnittswert;
  anzeigenMedianwert; anzeigenModuswert.
OrWerteErfassung11 := OrWerteErfassung new.
OrWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

daraufhin die Anforderungen

```
OrWerteErfassung11 zentrum "Median"; modus "Modus";
  anzeigenMedianwert; anzeigenModuswert.
NoWerteErfassung11 := NoWerteErfassung new.
NoWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

und abschließend die Anforderungen

```
NoWerteErfassung11 zentrum "Modus"; anzeigenModuswert
```

zu stellen.
Wird z.B. einer Instanz der Klasse "NoWerteErfassung" die Message "median" zugestellt, was – wie oben erläutert – aus statistischer Sicht nicht sinnvoll ist, so erhalten wir das Walkback-Fenster (siehe Anhang A.3) mit einer Fehlermeldung angezeigt. Wollen wir die Anzeige des Walkback-Fensters verhindern, so können wir z.B. innerhalb der Klasse "WerteErfassung" die folgende zusätzliche Methode vereinbaren:

```
doesNotUnderstand: aString
Transcript cr;
   show: ('Nicht bekannt ist in: ',
   self class printString,
   ' die Methode ',
   aString selector printString)
```

Kapitel 8

Klassen und Meta-Klassen

8.1 Meta-Klassen und Klassen-Methoden

Ausschnitt aus der Klassen-Hierarchie

Die von uns zuvor vereinbarten Klassen “WerteErfassung”, “NoWerteErfassung”, “OrWerteErfassung” und “InWerteErfassung” bilden eine Klassen-Hierarchie, die wie folgt strukturiert ist:

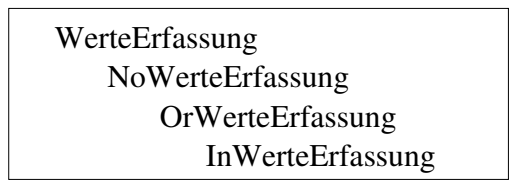


Abbildung 8.1: Aktuelle Klassen-Hierarchie

Bei der Lösung der Problemstellung PROB-5 haben wir diese Klassen-Hierarchie aufgebaut, damit wir verschiedenartige Methoden – unter Beibehaltung des charakteristischen Namens “zentrum” – vereinbaren und die jeweils sinnvollen statistischen Kennzahlen für die erfaßten Daten abrufen konnten.
Während wir bei den von uns eingerichteten Klassen die Klassennamen und die Methoden-Selektoren – in gewissem Rahmen – frei wählen konnten, stellen die Methoden, die vom SMALLTALK-System in den Basis-Klassen zur Verfügung gestellt werden, eine starre, auf der gesamten Klassen-Hierarchie basierende Grundlage für die Entwicklung von Lösungsplänen dar.
Zum Beispiel ist die Kenntnis darüber, welche Methoden insgesamt durch Instanzen der Basis-Klasse “Bag” zur Ausführung gelangen können, von der Beantwortung der folgenden Fragen abhängig:

- Welche Basis-Methoden sind in der Klasse “Bag” vereinbart?
- Welche Position auf welcher Hierarchiestufe nimmt die Basis-Klasse “Bag” im Rahmen der gesamten Klassen-Hierarchie ein, und welche Methoden sind in den Basis-Klassen vereinbart, denen die Klasse “Bag” untergeordnet ist?

Einen Einblick davon, welche Oberklassen von “Bag” existieren, vermittelt der folgende Ausschnitt aus der Klassen-Hierarchie des SMALLTALK-Systems:

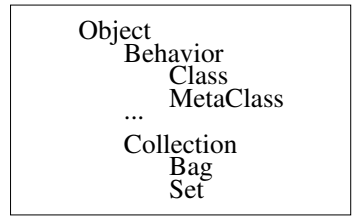


Abbildung 8.2: Ausschnitt aus der Klassen-Hierarchie

Hieraus ist erkennbar, daß “Behavior” und “Collection” direkte Unterklassen von “Object” sind, die auf derselben Hierarchiestufe angeordnet sind. Ferner ist diesem Ausschnitt zu entnehmen, daß “Class” und “MetaClass” auf derselben Hierarchiestufe angesiedelt und der Klasse “Behavior” direkt untergeordnet sind. Entsprechendes gilt für “Bag” und “Set” als direkte Unterklassen von “Collection”.

Die Klasse “Object”

Da die Klasse “Object” die einzige Klasse ist, die der Klasse “Collection” übergeordnet ist, sind sämtliche einsetzbaren Methoden für Instanzen der Klasse “Bag” genau die Methoden, die innerhalb von “Bag” zur Verfügung stehen, sowie alle diejenigen Methoden, die in den Klassen “Collection” und “Object” vereinbart sind.

Hinweis: Einen Überblick über die Messages, die Bags und anderen Sammlern geschickt werden können, geben wir im Kapitel 9.

- Die Klasse “Object” stellt die *Wurzel* der gesamten Klassen-Hierarchie dar. Sie besitzt keine übergeordnete Klasse, und alle Basis-Klassen sowie sämtliche neu vereinbarten Klassen sind dieser Klasse untergeordnet. Daher sind z.B. alle Methoden, die innerhalb der Klasse “Object” festgelegt sind, für alle Instanzen verfügbar, die aus einer Basis-Klasse oder einer neu aufgebauten Klasse eingerichtet werden.

Beispiele für Methoden, die innerhalb der Basis-Klasse “Object” vereinbart sind, stellen z.B. die folgenden Basis-Methoden dar:

- **“isMemberOf:”**:
  Das Ergebnis-Objekt der Message “isMemberOf:” ist gleich der Pseudovariablen “true” (“false”), sofern das Empfänger-Objekt (nicht) als Instanz derjenigen Klasse eingerichtet wurde, die als Argument von “isMemberOf:” aufgeführt ist.

  Hinweis: Zum Beispiel ist das Ergebnis-Objekt der Anforderung

  ```
  WerteErfassung11 isMemberOf: WerteErfassung
  ```

  die Pseudovariable “true”, sofern “WerteErfassung11” zuvor durch

```
WerteErfassung11 := WerteErfassung new
```

eingerichtet wurde.

- **“isKindOf:”:**
  Das Ergebnis-Objekt der Message “isKindOf:” ist gleich der Pseudovariablen “true” (“false”), sofern das Empfänger-Objekt (nicht) als Instanz derjenigen Klasse oder Unterklasse eingerichtet wurde, die als Argument von “isKindOf:” aufgeführt ist.

- **“isClass”:**
  Das Ergebnis-Objekt der Message “isClass” ist gleich der Pseudovariablen “true” (“false”), sofern das Empfänger-Objekt (nicht) Bestandteil der Klassen-Hierarchie ist.

  Hinweis: Zum Beispiel ist die Pseudovariable “true” das Ergebnis-Objekt der Message “WerteErfassung isClass”.

- **“class”:**
  Das Ergebnis-Objekt der Message “class” ist gleich der Klasse, aus der das Empfänger-Objekt instanziiert wurde.

  Hinweis: Zum Beispiel ergibt sich die Klasse “WerteErfassung” als Ergebnis-Objekt der Message “WerteErfassung11 class”.

Der Begriff der “Meta-Klasse”

Um uns über die Gesamtheit der innerhalb der Basis-Klasse “Object” vereinbarten Methoden zu informieren, läßt sich der Klassen-Hierarchie-Browser einsetzen. Sofern “Object” als *aktuelle* Klasse eingestellt ist, werden die Selektoren der vorhandenen Basis-Methoden im Methoden-Bereich des Klassen-Hierarchie-Browser-Fensters angezeigt.
Hinweis: Wir setzen dabei voraus, daß das Optionsfeld “instance” im Instanzen-/ Klassen-Bereich des Klassen-Hierarchie-Browser-Fensters aktiviert ist.

Unter den angezeigten Methoden ist die Methode “new”, die von uns zur Einrichtung einer Instanz verwendet wurde, *nicht* aufgeführt. Entsprechend läßt sich z.B. auch für die Basis-Klassen “Bag” und “Set”, die der Basis-Klasse “Object” untergeordnet sind, feststellen, daß in ihnen keine Methode namens “new” vereinbart ist. Daß die Methode “new” in diesen Klassen nicht aufgefunden werden kann, ist nicht verwunderlich, da die Message “new” keiner Instanziierung einer Klasse, sondern stets der Klasse *selbst* zugestellt wurde.
Hinweis: Zum Beispiel haben wir durch “Bag new” einen neuen Bag und durch “WerteErfassung new” einen neuen Erfassungsprozeß eingerichtet.

Da Messages sich stets an Objekte richten, sind die von uns verwendeten Messages mit dem Selektor “new” nur sinnvoll, weil die folgende Konvention besteht:

- Jede Klasse der Klassen-Hierarchie stellt ein *Objekt* dar.

Da es grundsätzlich zu jedem Objekt eine Klasse geben muß, aus der dieses Objekt als Instanz eingerichtet wurde, muß es – im Hinblick auf die strukturelle Sicht – konsequenterweise auch zu jeder Klasse eine ihr zugeordnete Klasse geben, aus der sich diese Klasse in Form einer Instanziierung erhalten läßt. Um diese Forderung zu erfüllen, gilt die Konvention:

- Zu jeder Klasse gibt es eine ihr zugeordnete Klasse, die *Meta-Klasse* genannt wird, so daß sich jede Klasse als *einzige* Instanz der ihr zugeordneten Meta-Klasse auffassen läßt.

Ergänzend zu den Basis-Klassen gibt es folglich Meta-Klassen, die mit den Basis-Klassen korrespondieren, so daß die Methode "new", die wir in den Basis-Klassen nicht auffinden konnten, vermutlich Bestandteil einer geeigneten Meta-Klasse bzw. einer den Meta-Klassen übergeordneten Klasse ist.

Hinweis: Dieser Sachverhalt wird nachfolgend bestätigt, indem "new" z.B. als Methode der Meta-Klasse von "Bag" erkannt wird.

Die Zuordnung von Klassen zu ihren Meta-Klassen gilt nicht nur für die Basis-Klassen, sondern auch für jede *neu* eingerichtete Klasse. Sofern wir eine *neue* Klasse vereinbaren, wird die zugehörige Meta-Klasse vom SMALLTALK-System *automatisch* erzeugt.

Klassen-Hierarchie mit Meta-Klassen

Die Meta-Klassen sind – genau wie die Klassen – hierarchisch geordnet. Die Hierarchie der Meta-Klassen stellt eine Parallele zur Hierarchie der Klassen dar. Ist eine Klasse "A" einer Klasse "B" unmittelbar übergeordnet, so stellt auch die Meta-Klasse von "A" die direkte Oberklasse der Meta-Klasse von "B" dar.
Im Hinblick auf die beiden Klassen-Hierarchien ist folgendes hervorzuheben:

- Während sämtliche Klassen der Klasse "Object" untergeordnet sind, stellt die Klasse "Class" die Oberklasse aller Meta-Klassen dar.

- Um eine Klasse von der ihr zugeordneten Meta-Klasse namensmäßig zu unterscheiden, besteht die Konvention, den Namen einer Meta-Klasse dadurch zu bilden, daß dem Klassennamen das Wort "class" angefügt wird.

Einen Ausschnitt aus der Klassen-Hierarchie und der Hierarchie der Meta-Klassen können wir auf der Grundlage der zuvor vorgestellten Hierarchie der Basis-Klassen "Object", "Collection", "Bag" und "Set" – unter Einbeziehung der für die gesamte Hierarchie besonders wichtigen Klassen "Behavior", "Class" und "MetaClass" – wie folgt beschreiben:

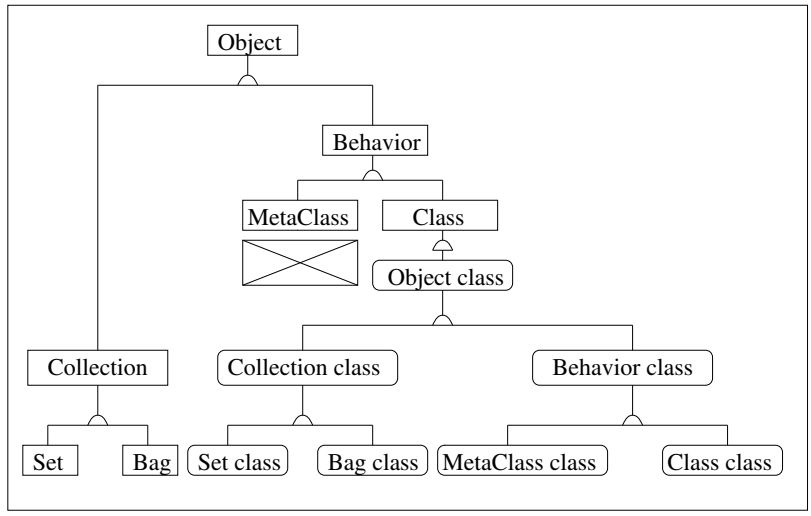


Abbildung 8.3: Hierarchie der Klassen und der Meta-Klassen

Bei dieser Darstellung ist jeder Klassenname durch ein Rechteck eingefaßt. Die einer Klasse zugeordnete Meta-Klasse, die die Klasse mit diesem Klassennamen als *einzige* Instanz enthält, wird durch einen Namen gekennzeichnet, der aus dem Klassennamen mit dem nachfolgendem Wort "class" gebildet ist. Die Namen der Meta-Klassen sind innerhalb eines abgerundeten Rechtecks angegeben.

Aus Gründen einer besseren Übersicht enthält die Darstellung den folgenden Eintrag:

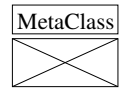


Abbildung 8.4: Instanzen von "MetaClass"

Diese Angabe soll bedeuten, daß jede einzelne Meta-Klasse als Instanz der Basis-Klasse "MetaClass" angesehen wird.

Hinweis: Durch die Verwendung der Basis-Klasse "MetaClass" ist gesichert, daß jede Meta-Klasse – als Instanz einer Klasse (nämlich der Klasse "MetaClass") – ein Objekt darstellt.
Dieser Sachverhalt gilt auch für die Meta-Klasse "MetaClass class", da auch sie als Instanz der Klasse "MetaClass" aufgefaßt wird.

Aus der abgebildeten Hierarchie der Klassen und Meta-Klassen ist folgendes erkennbar:

- Während die Methoden von "Class" und ihrer Oberklasse "Behavior" das *generelle* Verhalten aller *Meta-Klassen* bestimmen, wird das *generelle* Verhalten *aller* Objekte durch die Methoden der Klasse "Object" festgelegt.

Um sich über die Klassen-Hierarchie des SMALLTALK-Systems im Klassen-Hierarchie-Browser-Fenster zu informieren, kann wie folgt verfahren werden:

- Bekanntlich muß das Optionsfeld “instance” im Instanzen-/Klassen-Bereich aktiviert sein, sofern die Klassen, die der aktuell eingestellten Klasse übergeordnet sind, im Variablen-Bereich angezeigt werden sollen.
- Sollen dagegen diejenigen Klassen im Variablen-Bereich angezeigt werden, die der Meta-Klasse der aktuell eingestellten Klasse übergeordnet sind, so ist das Optionsfeld “class” im Instanzen-/Klassen-Bereich zu aktivieren.

Klassen-Methoden

Haben wir z.B. die Klasse “Bag” als *aktuelle* Klasse innerhalb des Klassen-Hierarchie-Browser-Fensters eingestellt, so führt die Aktivierung des Optionsfeldes “class” zum folgenden Fenster-Inhalt:

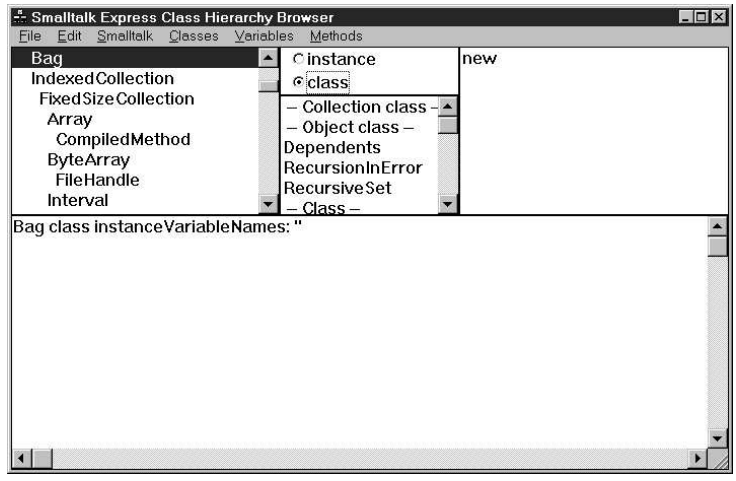


Abbildung 8.5: Anzeige der Hierarchie von Meta-Klassen

Der Anzeige im Variablen-Bereich kann – entsprechend der oben angegebenen Klassen-Hierarchie – entnommen werden, daß die Meta-Klasse “Collection class” eine Unterklasse von “Object class” und “Object class” eine Unterklasse der Basis-Klasse “Class” ist.

Im Methoden-Bereich ist der Name “new” eingetragen. Dies bedeutet, daß “new” eine Methode der Klasse “Bag class” ist.

Da innerhalb der Message “Bag new” das Empfänger-Objekt von “new” die Klasse “Bag”, d.h. die Instanz der Meta-Klasse “Bag class” ist, stellt “new” ein Beispiel für eine Methode dar, die in einer Meta-Klasse vereinbart ist und deren Empfänger-Objekt diejenige Klasse ist, der diese Meta-Klasse zugeordnet ist.

Da somit eine neue Art von Methoden vorliegt, wird die folgende Verabredung getroffen:

- Methoden, die in Meta-Klassen vereinbart sind, werden *Klassen-Methoden* genannt – im Unterschied zu den Methoden, die innerhalb von Klassen festgelegt und von Instanzen der Klasse ausführbar sind. Diese Methoden werden zur Unterscheidung als *Instanz-Methoden* bezeichnet.

  Hinweis: Bislang haben wir von "Methoden" gesprochen und damit – mit Ausnahme der Methode "new" – implizit Instanz-Methoden gemeint, die von einer z.B. mit der Message "new" eingerichteten Instanz ausgeführt werden können.

  Während sich eine Instanz-Methode allein durch eine Instanz einer Klasse ausführen läßt, muß das Empfänger-Objekt einer Message, die mit einer Klassen-Methode korrespondiert, eine Klasse sein.

Bei der Unterscheidung von Instanz- und Klassen-Methoden ist der folgende Sachverhalt bei der Zuordnung von Klasse und Meta-Klasse zu berücksichtigen:

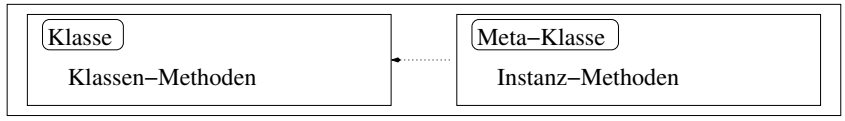


Abbildung 8.6: Zuordnung von Klasse und Meta-Klasse

Die Klassen-Methoden einer Klasse sind die Instanz-Methoden der zur Klasse zugehörigen Meta-Klasse. Diese Methoden sind von der Klasse ausführbar, die die *alleinige* Instanz der korrespondierenden Meta-Klasse ist.

- Im Hinblick auf die Klassen-Hierarchie werden die Klassen-Methoden einer Klasse – genauso wie ihre Instanz-Methoden – auf sämtliche ihr untergeordneten Klassen vererbt.

Daß darüberhinaus auch Instanz-Methoden als Klassen-Methoden vererbbar sind, läßt sich aus der in Abbildung 8.3 angegebenen Darstellung der Klassen-Hierarchie entnehmen. Es gilt nämlich der folgende Sachverhalt:

- Bei den Instanz-Methoden der Basis-Klassen "Behavior", "MetaClass" und "Class" handelt es sich um Methoden, die als Klassen-Methoden an die Basis-Klasse "Object" und alle Klassen vererbt werden, die "Object" untergeordnet sind.

Demzufolge kann z.B. die in der Basis-Klasse "Behavior" vereinbarte Instanz-Methode "allSuperclasses" wie folgt als Klassen-Methode zur Ausführung abgerufen werden:

```
WerteErfassung allSuperclasses
```

Als Ergebnis erhalten wir – beim Einsatz der Menü-Option "Show It" des Menüs "Smalltalk" – sämtliche Oberklassen von "WerteErfassung", d.h. "ViewManager" und "Object", in der folgenden Form angezeigt:

```
OrderedCollection(ViewManager Object)
```

Hinweis: Entsprechend lassen sich z.B. sämtliche Instanzen einer Klasse mittels der Methode "allInstances" und alle Unterklassen einer Klasse mittels der Methode "allSubclasses" anzeigen.

Unter Einsatz der Methode "implementorsOf:" können die Klassen ermittelt werden, in denen eine bestimmte Methode vereinbart ist.
Zum Beispiel läßt sich durch die Message "Smalltalk implementorsOf: #size" die Gesamtheit der Klassen feststellen, in denen eine Methode namens "size" vereinbart ist (siehe Abschnitt 9.4.2).

Entsprechend können durch den Einsatz der Methode "sendersOf:" sämtliche Methoden (sowie die Klassen, in denen diese Methoden vereinbart sind) ermittelt werden, in denen eine bestimmte Methode verwendet wird.
Zum Beispiel läßt sich durch die Message "Smalltalk sendersOf: #size" die Gesamtheit der vereinbarten Methoden feststellen, in deren Methoden-Vereinbarung eine Methode namens "size" verwendet wird.

Sofern eine Instanz-Methode von "Behavior" – wie z.B. "new" – nicht als Klassen-Methode auf eine Klasse – wie z.B. "Bag" – vererbt werden soll, besteht die Möglichkeit, die in "Behavior" vereinbarte Methode dadurch zu "überdecken", daß eine Instanz-Methode gleichen Namens ("new") in der Meta-Klasse "Bag class", d.h. als Klassen-Methode in der Klasse "Bag", vereinbart wird.
Davon wird beim SMALLTALK-System bei der Einrichtung eines Bags Gebrauch gemacht, da eine Klassen-Methode namens "new" innerhalb der Klasse "Bag" existiert.

Da die Meta-Klasse "Bag class" eine Unterklasse von "Collection class", "Collection class" eine Unterklasse von "Object class", "Object class" eine Unterklasse der Basis-Klasse "Class" und diese wiederum eine Unterklasse von "Behavior" ist, *überdeckt* die in der Klasse "Bag" definierte Klassen-Methode "new" die gleichnamige Instanz-Methode in "Behavior".

Hinweis: Die Methode "new" der Basis-Klasse "Behavior" kommt bei der Instanziierung einer Klasse immer dann zum Einsatz, wenn – im Rahmen der Klassen-Hierarchie – innerhalb der zugehörigen Meta-Klasse und unterhalb der Basis-Klasse "Behavior" keine gleichnamige Methode vereinbart ist.

Grundsätzlich läßt sich feststellen:

- Soll das *spezifische* Verhalten einer Klasse festgelegt werden, so kann dies dadurch geschehen, daß geeignete Klassen-Methoden als Instanz-Methoden der zugehörigen Meta-Klasse vereinbart werden.
- Indem eine Klassen-Methode namens "new" für eine Klasse festgelegt wird, ist es möglich, für jede Klasse eine spezielle Methode zum Einrichten einer Instanz zu verabreden.

Die Klassen-Methode “new”

Nachdem wir in den einleitenden Kapiteln die Methode “new” – ohne eine gesonderte Begründung zu geben – zur Instanziierung verwendet haben, versetzt uns die Kenntnis der Klassen-Hierarchie und der Existenz von Meta-Klassen in die Lage, den Hintergrund zu erkennen, auf dem sich der Einsatz der Basis-Methode “new” gründet.
Wie wir oben festgestellt haben, gibt es nicht nur eine, sondern mehrere Vereinbarungen von Methoden namens “new”. Grundsätzlich handelt es sich bei diesen Basis-Methoden um Klassen-Methoden, durch deren Ausführung sich jeweils ein *neues* Objekt einer Klasse durch eine Instanziierung einrichten läßt.

Von welcher Art das jeweilige Objekt ist, richtet sich nach der Klasse, aus der das Objekt instanziiert wird. Sofern die Methode “new” als Instanz-Methode innerhalb der zu dieser Klasse zugehörigen Meta-Klasse vereinbart ist, wird diese Methode bei der Instanziierung verwendet. Besitzt diese Meta-Klasse keine Methode namens “new”, so wird in den dieser Meta-Klasse übergeordneten Meta-Klassen nach dieser Methode gesucht. Spätestens bei der Prüfung der Instanz-Methoden der Basis-Klasse “Behavior” ist die Suche nach einer Methode namens “new” erfolgreich.

Soll *nicht* die durch ein derartiges Suchverfahren bestimmte Methode “new” zur Ausführung gelangen, sondern soll eine klassen-spezifische Instanziierung festgelegt werden, so muß eine geeignete Vereinbarung einer Klassen-Methode namens “new” erfolgen.

Bei der Vereinbarung von *neuen* Klassen ist es z.B. dann sinnvoll, eine Klassen-Methode namens “new” festzulegen, wenn den Instanz-Variablen der erzeugten Instanzen voreingestellte Werte zuzuordnen sind.

Hinweis: Diesen Vorteil werden wir bei der Lösung von PROB-6 im Abschnitt 8.2 nutzen.

Ist in einer Klasse eine Klassen-Methode “new” vereinbart worden, so *überdeckt* sie diejenige Methode “new”, die in einer ihr übergeordneten Klasse als Klassen-Methode festgelegt ist.

Durch die Möglichkeit, für jede einzelne Klasse eine individuelle Klassen-Methode “new” in der ihr zugeordneten Meta-Klasse festlegen zu können, kann sich die Form, in der ein Objekt als Instanz einer Klasse eingerichtet wird, von Klasse zu Klasse unterscheiden.
Von dieser Möglichkeit wird bei den Basis-Klassen des SMALLTALK-Systems nur dann Gebrauch gemacht, wenn die jeweils zu erzeugenden Instanzen durch besondere Eigenschaften geprägt sein sollen, so daß sie nicht durch die Klassen-Methode “new”, die innerhalb der Klasse “Behavior” zur generellen Verfügung steht, eingerichtet werden können.

Wie wir bereits festgestellt haben, ist dies z.B. bei der Klasse “Bag” der Fall. Zur Einrichtung einer Instanz dieser Klasse, führt die Klasse “Bag” nicht die Methode “new” aus der Klasse “Behavior”, sondern diejenige Methode “new” aus, die als Instanz-Methode von “Bag class” und damit als Klassen-Methode von “Bag” festgelegt ist. Diese innerhalb der Klasse “Bag” vereinbarte Klassen-Methode “new” *überdeckt* somit diejenige Methode “new”, die in der übergeordneten Klasse “Behavior” als Instanz-Methode festgelegt ist.
Im Hinblick auf den Einsatz der Methode “new” ist hervorzuheben, daß bestimmte

Objekte *nicht* durch "new" instanziiert werden dürfen. Wie wir bereits im Abschnitt 3.2 beschrieben haben, werden alle Literale wie z.B. Zahlen oder Zeichenketten (Strings) allein dadurch instanziiert, daß sie innerhalb von Messages als Empfänger-Objekte bzw. als Argumente durch "Hinschreiben" aufgeführt werden.

Hinweis: Eine Anforderung der Form "Integer new" führt z.B. zur Anzeige des Walkback-Fensters (siehe Anhang A.3) mit der Fehlermeldung "inappropriate message for this object". Diese Meldung erklärt sich daraus, daß innerhalb einer unmittelbaren Oberklasse ("Number") der Klasse "Integer" die zugehörige Klassen-Methode "new" zur Verfügung gehalten wird, bei deren Ausführung die Ausgabe der Fehlermeldung durch die Ausführung der Methode "invalidMessage" der Klasse "Object" vorgenommen wird.

8.2 Klassen-Variablen

Vereinbarung von Klassen-Variablen

Bei der Vereinbarung einer Klasse lassen sich nicht nur Klassen-Methoden, sondern auch klassen-spezifische Eigenschaften verabreden. Diese Eigenschaften sind durch Klassen-Attribute festgelegt, die auf sämtliche Instanzen der jeweiligen Klasse wirken.

- In jeder Klasse lassen sich eine oder mehrere *Klassen-Variablen* einrichten. Es handelt sich – ähnlich wie bei den globalen Variablen – um Variablen, die von allen Instanzen der jeweiligen Klasse *gemeinsam* benutzt werden können. Daher sind ihre Variablennamen nach denselben Regeln aufzubauen, wie sie für die Namen von globalen Variablen festgelegt sind.

  Hinweis: Im Unterschied zu den globalen Variablen, die wir bisher kennengelernt haben, sind Klassen-Variablen nur "global" bezüglich der Klasse, in der sie vereinbart sind, sowie allen dieser Klasse untergeordneten Klassen.

Klassen-Variablen sind bei der Vereinbarung einer Klasse festzulegen. Dabei sind die Namen einer oder mehrerer Klassen-Variablen insgesamt in Hochkommata einzufassen und hinter den zu verabredenden Instanz-Variablen in der folgenden Form aufzuführen:

```
instanceVariableNames: 'instanz-variablen'
classVariableNames: 'klassen-variablen'
```

- Im Gegensatz zu den Instanz-Variablen sind Klassen-Variablen *nicht* Bestandteile einer Instanz.

Da jede Instanz "weiß", aus welcher Klasse sie instanziiert wurde, kann von jeder Instanz der Klasse, in der eine Klassen-Variable vereinbart ist – mittels einer Klassen- oder Instanz-Methode – auf das der Klassen-Variablen zugeordnete Objekt zugegriffen werden. Ferner kann jede Instanz einer untergeordneten Klasse auf die Klassen-Variablen der übergeordneten Klassen zugreifen.

Hinweis: Es ist zu beachten, daß eine Klassen-Vereinbarung lediglich die Variablennamen der Attribute – in Form von Instanz-Variablen oder Klassen-Variablen – spezifiziert. Sie enthält keine Attributwerte. Um einer Klassen-Variablen ein Objekt zuzuordnen, kann eine Instanz- oder eine Klassen-Methode zur Ausführung gebracht werden. Bei unseren

Beispielen werden wir nur Klassen-Methoden einsetzen.
Da der Attributwert einer Klassen-Variablen nicht Bestandteil einer Instanz ist, besitzen sämtliche bereits eingerichteten und später erzeugten Instanzen einer Klasse unmittelbar Zugriff auf das *neue* Objekt, das einer Klassen-Variablen durch eine Zuweisung zugeordnet wurde.

- Welche Klassen-Variablen jeweils für eine Klasse vereinbart und folglich für die Instanzen dieser Klasse zugreifbar sind, läßt sich im Variablen-Bereich des Klassen-Hierarchie-Browser-Fensters anzeigen, indem das Optionsfeld “class” aktiviert wird.

Hinweis: Dabei ist zu beachten, daß die durch einen Großbuchstaben eingeleiteten Klassen-Variablen und die mit einem Kleinbuchstaben beginnenden Instanz-Variablen innerhalb der durch Bindestriche “–” gekennzeichneten Klassen-Hierarchie erscheinen.

Um mitzuteilen, welche Klassen-Variablen und welche Klassen-Methoden für eine Klasse vereinbart sind, ergänzen wir die bisherige Form, in der wir eine Klasse grafisch darstellen, durch die jeweils zusätzlichen Angaben, so daß eine Klassen-Vereinbarung wie folgt angegeben werden kann:

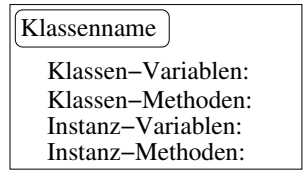


Abbildung 8.7: Vereinbarung einer Klasse

Somit können wir z.B. den Sachverhalt, daß eine Oberklasse “O” mit einer Klassen-Variablen “KO” der Unterklasse “U” mit der Klassen-Variablen “KU” übergeordnet ist, wie folgt beschreiben:

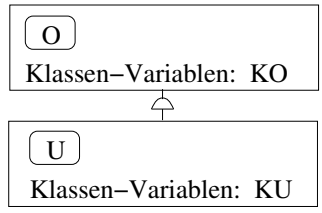


Abbildung 8.8: Klassen-Variablen

In dieser Situation hat jede Instanz von “U” und “O” Zugriff auf “KO”. Eine Instanz von “O” hat Zugriff auf “KO”, jedoch nicht auf “KU”.

Verwendung von Klassen-Variablen

Um den Einsatz einer Klassen-Variablen vorzustellen, gehen wir *nicht* von der im Abschnitt 7.4 in der Form

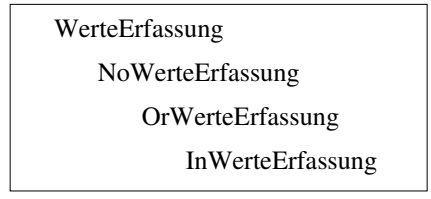


Abbildung 8.9: Hierarchische Unterordnung

festgelegten Hierarchie aus, sondern verzichten auf die zuvor verabredeten Spezialisierungen der Klasse "WerteErfassung".

Um den Einsatz einer Klassen-Variablen an einem einfachen Beispiel darzustellen, wollen wir die folgende Problemstellung lösen:

- PROB-6:
  Auf der Basis der Klasse "WerteErfassung" soll eine Klasse namens "ZaehlerWerteErfassung" – als Spezialisierung von "WerteErfassung" – entwickelt werden, durch deren Einsatz die Gesamtzahl der *insgesamt* erfaßten Punktwerte gezählt werden kann.
  Diese Zählung soll nicht jahrgangsstufen-spezifisch erfolgen, sondern die Gesamtheit aller eingegebenen Werte ermitteln.

Hinweis: Im Hinblick auf später zu entwickelnde Problemlösungen ist es sinnvoll, Vorkehrungen zu treffen, um die zuvor aufgebaute Hierarchie – mit den bereits vorliegenden Klassen-Vereinbarungen von "InWerteErfassung", "OrWerteErfassung" und "NoWerteErfassung" – wieder einrichten zu können.

Um diese drei Klassen-Vereinbarungen zu sichern, läßt sich im Klassen-Hierarchie-Browser-Fenster die Menü-Option "File Out..." des Menüs "Classes" einsetzen.

Nach der Übertragung in drei verschiedene Sicherungsdateien kann die Löschung der Klassen "InWerteErfassung", "OrWerteErfassung" und "NoWerteErfassung" – in dieser Reihenfolge – durch die Menü-Option "Remove Class" des Menüs "Classes" erfolgen.

Es ist zu beachten, daß – vor dem Löschen – alle Instanzen dieser Klassen gelöscht werden müssen.

Soll z.B. die ursprüngliche Hierarchie der drei Klassen unterhalb der – im Zusammenhang mit der Lösung von PROB-6 – neu einzurichtenden Klasse "ZaehlerWerteErfassung" in der Form

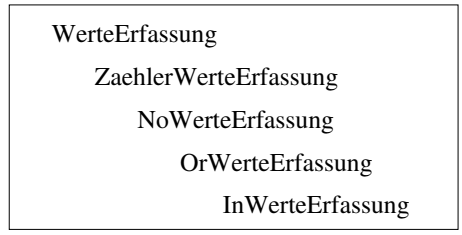


Abbildung 8.10: Hierarchie-Erweiterung

zu einem späteren Zeitpunkt wieder aufgebaut werden, muß – vor dem erneuten Laden der gespeicherten Klassen-Vereinbarungen von "InWerteErfassung", "OrWerteErfassung" und "NoWerteErfassung" – zunächst mit der Menü-Option "Open..." des Menüs "File" die Sicherungsdatei von "NoWerteErfassung" eröffnet werden. Anschließend muß der Klassenname "ZaehlerWerteErfassung" – anstelle der ursprünglichen Angabe "WerteErfassung" – als neue Oberklasse von "NoWerteErfassung" in die daraufhin angezeigte Vereinbarung eingetragen und mittels der Menü-Option "Save" des Menüs "File" gesichert werden.
Danach lassen sich im Klassen-Hierarchie-Browser-Fenster mit der Menü-Option "Install..." des Menüs "File" die Vereinbarungen der Klassen "NoWerteErfassung", "OrWerteErfassung" und "InWerteErfassung" – in dieser Reihenfolge – laden.
Es ist zu beachten, daß diese Klassen erst dann im Klassen-Bereich des Klassen-Hierarchie-Browser-Fensters angezeigt werden, wenn die Menü-Option "Update" des Menüs "Classes" angewählt wurde.

Zur Lösung von PROB-6 beabsichtigen wir, in der Klasse "ZaehlerWerteErfassung" eine Klassen-Variable namens "Zaehler" zu vereinbaren, der die Anzahl der erfaßten Werte als ganzzahliger Wert zugeordnet werden soll. Wir sehen für den Lösungsplan somit die folgende Strukturierung vor:

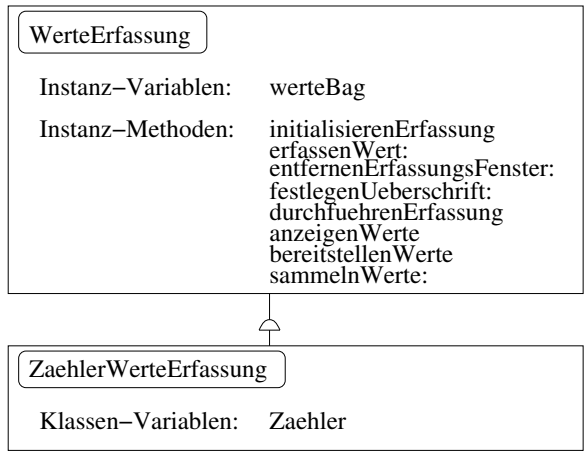


Abbildung 8.11: Lösungsplan für PROB-6

Durch den Einsatz des Klassen-Hierarchie-Browsers vereinbaren wir daher die Klasse "ZaehlerWerteErfassung" in der folgenden Form:

```
WerteErfassung subclass: #ZaehlerWerteErfassung
  instanceVariableNames: ''
  classVariableNames: 'Zaehler'
  poolDictionaries: ''
```

Dabei haben wir den Namen der von uns vorgesehenen Klassen-Variablen "Zaehler" durch die folgende Angabe festgelegt:

```
classVariableNames: 'Zaehler'
```

Der Klassen-Variablen "Zaehler" soll bei der *erstmaligen* Instanziierung der Klasse "ZaehlerWerteErfassung" der Wert "0" zugeordnet werden. Dieser Wert ist jeweils dann, wenn ein Punktwert erfaßt wird, um den Wert "1" zu erhöhen.

Die gewünschte Initialisierung läßt sich dadurch erreichen, daß wir die standardmäßig zur Verfügung stehende Basis-Methode "new" durch eine *neue* Klassen-Methode "new" überdecken und für diese Methode in der Klasse "ZaehlerWerteErfassung" die folgende Vereinbarung vorsehen:

```
new
|varWerteErfassung|
varWerteErfassung := super new.
varWerteErfassung initZaehler.
^ varWerteErfassung
```

Hinweis: Fordern wir die Vereinbarung von "new" durch die Menü-Option "Save" des Menüs "File" im Klassen-Hierarchie-Browser-Fenster an, so erscheint die Meldung "You are redefining a superclass method". Dies weist darauf hin, daß eine gleichnamige Methode namens "new" in einer übergeordneten Klasse (in diesem Fall "Behavior") *überdeckt* wird. Da wir dies beabsichtigen, bestätigen wir diese Meldung.

Durch den Einsatz der Pseudovariablen "super" erreichen wir, daß die Instanziierung in gewohnter Form durch die dafür standardmäßig zuständige, in einer übergeordneten Klasse vereinbarte Basis-Methode "new" ausgeführt wird. Die resultierende Instanz führt anschließend die Methode "initZaehler" aus, die sicherstellen soll, daß die Klassen-Variable "Zaehler" den Wert "0" erhält, sofern die *allererste* Instanziierung der Klasse "ZaehlerWerteErfassung" erfolgt.

Hinweis: Setzen wir bei der Vereinbarung der Methode "new" statt der Pseudovariablen "super" die Pseudovariable "self" ein, so erhalten wir eine Endlosschleife und damit im Walkback-Fenster (siehe Anhang A.3) die Meldung "stack overflow" angezeigt.

Damit die Anforderung

```
ZaehlerWerteErfassung new sammelnWerte: 'Jahrgangsstufe 11'
```

sinnvoll ist, muß das Ergebnis-Objekt von "ZaehlerWerteErfassung new" eine Instanz der Klasse "ZaehlerWerteErfassung" sein. Damit dies sichergestellt ist, lautet

die letzte Anforderung innerhalb der Methoden-Vereinbarung von "new":

```
∧ varWerteErfassung
```

Hinweis: Ohne den Einsatz von "∧" wäre die Klasse "ZaehlerWerteErfassung" das Ergebnis-Objekt von "`ZaehlerWerteErfassung new`", so daß die Message
`sammelnWerte: 'Jahrgangsstufe 11'`
die Suche nach einer Klassen-Methode namens "sammelnWerte:" auslösen würde. Da eine derartige Klassen-Methode nicht existiert, würde eine Fehlermeldung im Walkback-Fenster resultieren.

Da eine Klassen-Variable grundsätzlich mit der *Pseudovariablen "nil"* vorbesetzt ist, läßt sich die Zuordnung der ganzen Zahl "0" bei der erstmaligen Instanziierung dadurch erreichen, daß die Instanz-Methode "initZaehler" in der folgenden Form innerhalb der Klasse "ZaehlerWerteErfassung" verabredet wird:

```
initZaehler
Zaehler isNil ifTrue: [Zaehler := 0]
```

Nachdem wir mittels des Klassen-Hierarchie-Browser-Fensters die Klassen-Methode "new" – bei aktiviertem Optionsfeld "class" – und die Instanz-Methode "initZaehler" – bei aktiviertem Optionsfeld "instance" – eingerichtet haben, können wir eine Instanziierung der Klasse "ZaehlerWerteErfassung" z.B. wie folgt im Workspace-Fenster anfordern:

```
WerteErfassung11 := ZaehlerWerteErfassung new
```

Bei dieser Instanziierung wird die Instanz-Methode "initZaehler" durch

```
varWerteErfassung initZaehler
```

zur Ausführung gebracht, so daß durch

```
Zaehler isNil ifTrue: [Zaehler := 0]
```

der Klassen-Variablen "Zaehler" der Wert "0" zugeordnet wird.
Damit die Erhöhung des Inhalts von "Zaehler" bei der Erfassung eines neuen Punktwertes durchgeführt wird, müßte eigentlich die Methode "erfassenWert:", die innerhalb der Klasse "WerteErfassung" in der Form

```
erfassenWert: aPane
werteBag add: (self paneNamed: 'eingabeFeld') contents.
(self paneNamed: 'eingabeFeld') contents: ''.
(self paneNamed: 'eingabeFeld') setFocus
```

vereinbart ist, durch die folgende Zuweisung ergänzt werden:

```
Zaehler := Zaehler + 1
```

Durch den Einsatz der Pseudovariablen “super” ist es jedoch möglich, die ursprüngliche Vereinbarung der Methode “erfassenWert:” – in ihrer allgemeinen Form – innerhalb der Klasse “WerteErfassung” unverändert zu erhalten und eine geeignete Spezialisierung in einer gleichnamigen Instanz-Methode innerhalb der Klasse “ZaehlerWerteErfassung” in der folgenden Form vorzunehmen:

```
erfassenWert: aPane
super erfassenWert: aPane.
Zaehler := Zaehler + 1
```

Dieses Vorgehen verdeutlicht den Vorteil der Pseudovariablen “super”:

- Eine in einer Klasse vereinbarte Methode läßt sich – mit Hilfe der *Pseudovariablen “super”* – dadurch verändern, daß sie durch eine gleichnamige Methode in einer untergeordneten Klasse mittels einer Ergänzung durch eine oder mehrere Anforderungen *spezialisiert* wird.

  Sofern z.B. eine Methode namens “mK”, die in der Klasse “O” vereinbart ist und unter dem gleichen Namen “mK” innerhalb einer untergeordneten Klasse “U” – um weitere Anforderungen ergänzt – aufgerufen werden soll, stellt sich dies wie folgt dar:

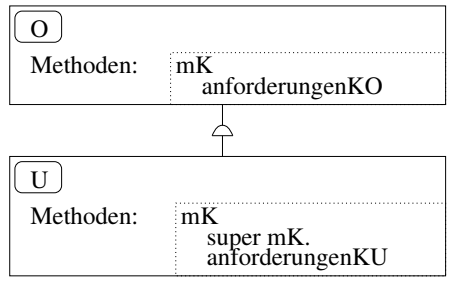


Abbildung 8.12: Spezialisierung von Methoden

  Führt eine Instanz der Klasse “U” die Methode “mK” aus, so werden zunächst die Anforderungen “anforderungenKO” und anschließend die Anforderungen “anforderungenKU” ausgeführt.

Damit sich der Zähler, der der Klassen-Variablen “Zaehler” zugeordnet ist, abrufen läßt, vereinbaren wir für die Klasse “ZaehlerWerteErfassung” eine *Klassen-Methode* namens “anzeigenZaehler” in der folgenden Form:

```
anzeigenZaehler
Transcript cr;
    show: 'Anzahl der bislang insgesamt erfaßten Werte: ';
    show: Zaehler printString
```

Hinweis: Die Methode “anzeigenZaehler” hätten wir auch als Instanz-Methode vereinbaren können.

Damit ergibt sich für die beiden Klassen “WerteErfassung” und “ZaehlerWerteErfassung” insgesamt die folgende Situation:

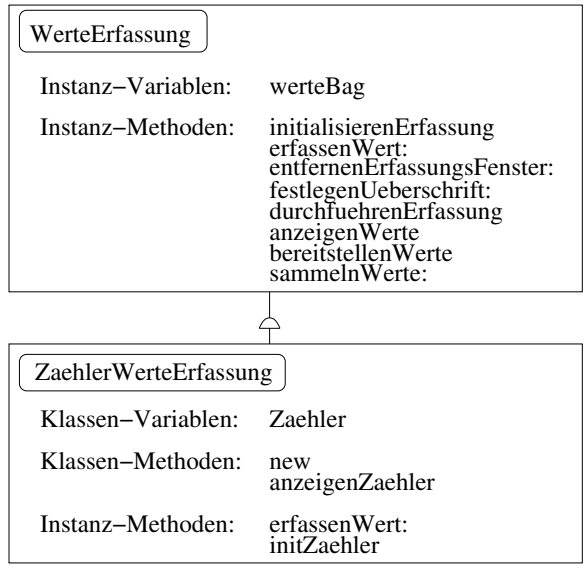


Abbildung 8.13: Vereinbarung von “ZaehlerWerteErfassung”

Werden im Anschluß an die Anforderungen

```
WerteErfassung11 := ZaehlerWerteErfassung new.
WerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

z.B. 30 Werte der Jahrgangsstufe 11 erfaßt, so führt die Anforderung

```
ZaehlerWerteErfassung anzeigenZaehler
```

zur Anzeige von “30”.
Folgt der Eingabe der Anforderungen

```
WerteErfassung12 := ZaehlerWerteErfassung new.
WerteErfassung12 sammelnWerte: 'Jahrgangsstufe 12'
```

anschließend die Erfassung von z.B. 25 Werten der Jahrgangsstufe 12, so führt die Anforderung

```
ZaehlerWerteErfassung anzeigenZaehler
```

zur Anzeige von “55”.

8.3 Pool-Dictionary-Variablen

Um einen Zähler für die Anzahl der insgesamt erfaßten Punktwerte mitzuführen, haben wir – zur Lösung von PROB-6 – geeignete Vorkehrungen in der Klasse "ZaehlerWerteErfassung" getroffen, die von uns als *unmittelbare* Unterklasse von "WerteErfassung" vereinbart wurde.
Darüberhinaus ist es auch in der im Abschnitt 7.3 angegebenen Situation

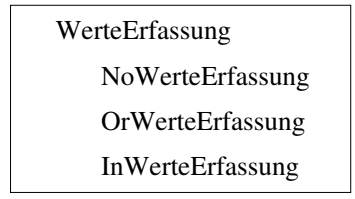


Abbildung 8.14: Unterordnung auf derselben Hierarchiestufe

möglich, die insgesamt erfaßten Werte dadurch zu zählen, daß die zuvor entwickelte Klasse "ZaehlerWerteErfassung" *unterhalb* von "WerteErfassung" festgelegt und die Klassen "InWerteErfassung", "OrWerteErfassung" und "NoWerteErfassung" gemeinsam auf derselben Hierarchiestufe *unterhalb* von "ZaehlerWerteErfassung" vereinbart werden.
Wird jedoch die angegebene Situation zugrundegelegt, und soll die Lösung durch eine *Spezialisierung* der vorliegenden Hierarchie vorgenommen werden, so muß der Zugriff auf eine Variable möglich sein, die in mehr als einer Klasse vereinbart ist.
Hinweis: Die Lösung, eine globale Variable zur Speicherung des Zählers zu verwenden, schließen wir bewußt aus.

In dieser Situation bietet es sich an, die Einrichtung eines Zählers auf den folgenden Sachverhalt zu gründen:

- Gemeinsame Attribute von Instanzen aus zwei oder mehreren Klassen, die einander *nicht* hierarchisch untergeordnet sein müssen, können mit Hilfe von *Pool-Dictionary-Variablen* beschrieben werden, so daß auch in diesem Fall klassen-übergreifende Attribute vereinbart werden können.

  Der Name einer Pool-Dictionary-Variablen muß so aufgebaut sein, wie es die Regeln für die Bildung von Namen globaler Variablen festlegen.

  Hinweis: Zu den wichtigen Pool-Dictionary-Variablen, die innerhalb des SMALLTALK-Systems verwendet werden, zählen z.B. "CharacterConstants", "WinConstants" und "VirtualKeyConstants".
  Dabei sind in "CharacterConstants" wichtige Bezeichnungen für den Zugriff auf besondere Zeichen (z.B. Verabredungen für das Leerzeichen und das Tabulator-Zeichen) und in "WinConstants" wichtige Bezeichnungen für die Interaktion mit dem Windows-System gespeichert.
  In "VirtualKeyConstants" sind Bezeichnungen für bestimmte Tasten bzw. Tastenkombinationen der Eingabetastatur enthalten.

Um den Einsatz einer "Pool-Dictionary-Variablen" vorzustellen, betrachten wir *nicht* die oben angegebene Hierarchie, sondern beabsichtigen – der Einfachheit halber –, die folgende Problemstellung zu lösen:

- PROB-7:
  Auf der Basis der Klasse "WerteErfassung" und der ihr auf derselben Hierarchiestufe untergeordneten Klassen "InWerteErfassung" und "OrWerteErfassung" sollen zwei Klassen namens "ZaehlerInWerteErfassung" und "ZaehlerOrWerteErfassung" – als Spezialisierungen von "InWerteErfassung" bzw. "OrWerteErfassung" – entwickelt werden, durch deren Einsatz die Gesamtzahl der *insgesamt* erfaßten Punktwerte gezählt werden kann. Diese Zählung soll nicht jahrgangsstufen-spezifisch und auch nicht skalenniveau-spezifisch erfolgen, sondern die Gesamtheit aller erfaßten Werte berücksichtigen.

Für die Lösung von PROB-7 sehen wir die folgende Strukturierung vor:

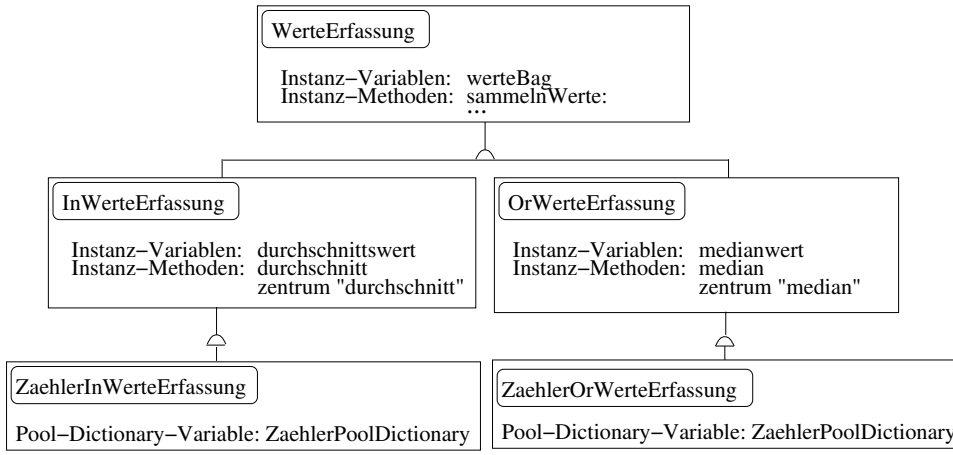


Abbildung 8.15: Lösungsplan für PROB-7

Wir beabsichtigen somit, eine Pool-Dictionary-Variable namens "ZaehlerPoolDictionary" zu verwenden.
Einer Pool-Dictionary-Variablen muß ein *Dictionary*, d.h. eine Instanz der Basis-Klasse "Dictionary", zugeordnet sein.

- Ein *Dictionary* ist ein Sammler, in dem sich Paare von Objekten als *Key-Value-Paare* eintragen lassen. Jedes Key-Value-Paar besteht aus einem *Key* (Schlüssel) und einem *Value* (Wert), auf den der Key *eindeutig* weist. Im Gegensatz zum Key, bei dem es sich um ein Symbol, eine Zeichenkette (String) oder um eine Zahl handeln darf, dürfen als Value beliebige Objekte verwendet werden.

Für die von uns zur Lösung von PROB-7 vorgesehene Pool-Dictionary-Variable namens "ZaehlerPoolDictionary" beabsichtigen wir, ein Key-Value-Paar der folgenden

Form einzurichten:

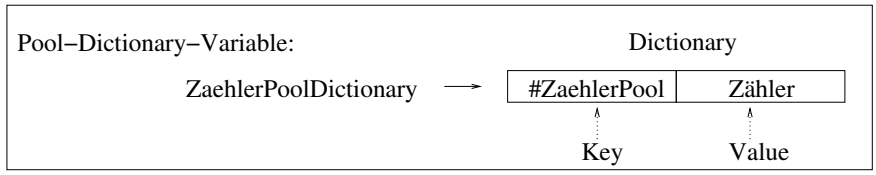


Abbildung 8.16: Pool-Dictionary-Variable mit zugeordnetem Dictionary

Als "Objekt" soll dem Key "`#ZaehlerPool`" der Zähler, mit dem die Anzahl der insgesamt erfaßten Punktwerte gezählt werden soll, als Value zugeordnet werden.

Damit "ZaehlerPoolDictionary" in den einzurichtenden Klassen "ZaehlerInWerte-Erfassung" und "ZaehlerOrWerteErfassung" als Pool-Dictionary-Variable festgelegt werden kann, muß "ZaehlerPoolDictionary" zuvor als globales Objekt – in Form einer Instanz der Basis-Klasse "Dictionary" – wie folgt vereinbart werden:

```
Smalltalk at: #ZaehlerPoolDictionary put: Dictionary new
```

Hinweis: "Smalltalk" ist eine globale Variable, die auf die *einzige* Instanz der Basis-Klasse "System-Dictionary" weist. In diesem *System-Dictionary* sind die Namen aller globalen Objekte des SMALLTALK-Systems als Keys innerhalb von Key-Value-Paaren gespeichert (siehe Abschnitt 9.4.2).

Bei dieser Anforderung wird die Message "at:put:" verwendet, mit der sich Key-Value-Paare innerhalb eines Dictionarys vereinbaren lassen.

- **"at:put:"**:
  Durch die Basis-Methode "at:put:" wird dem Key, der hinter dem Selektor "at:" als Argument angegeben ist, dasjenige Objekt als Value zugeordnet, das als Argument des Selektors "put:" aufgeführt wird. Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Key zugeordnet wird.
  Dabei wird das Key-Value-Paar in demjenigen Dictionary eingerichtet, das als Empfänger-Objekt innerhalb der Message "at:put:" aufgeführt ist.

Soll innerhalb eines Dictionarys über einen Key auf ein Objekt zugegriffen werden, so ist die Message "at:" einzusetzen.

- **"at:"**:
  Durch die Basis-Methode "at:" wird mit dem Key, der als Argument hinter dem Selektor "at:" aufgeführt ist, auf den Value zugegriffen, der dem Key innerhalb des Dictionarys, das als Empfänger-Objekt der Message "at:" verwendet wird, in Form eines Key-Value-Paares zugeordnet ist. Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Key zugeordnet ist.

Um für den Zählvorgang ein geeignetes Key-Value-Paar innerhalb der Pool-Dictionary-Variablen "ZaehlerPoolDictionary" festzulegen, ordnen wir dem Key "#ZaehlerPool" durch

```
ZaehlerPoolDictionary at: #ZaehlerPool put: nil
```

die Pseudovariable “nil” als Objekt zu. Dadurch soll dokumentiert werden, daß bislang *keine* Instanziierung eines Erfassungsprozesses durchgeführt wurde.

Hinweis: Sofern erstmalig eine Instanziierung vorgenommen wird, soll dies dadurch dokumentiert werden, daß der dem Key “`#ZaehlerPool`” zugeordnete Value “nil” durch die ganze Zahl “0” ersetzt wird.

Nachdem wir im Workspace-Fenster die Anforderungen

```
Smalltalk at: #ZaehlerPoolDictionary put: Dictionary new.
ZaehlerPoolDictionary at: #ZaehlerPool put: nil
```

eingetragen und zur Ausführung gebracht haben, vereinbaren wir die Klasse “ZaehlerInWerteErfassung” als Unterklasse von “InWerteErfassung”, indem wir innerhalb des Klassen-Hierarchie-Browser-Fensters die folgende Vereinbarung treffen:

```
InWerteErfassung subclass: #ZaehlerInWerteErfassung
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: 'ZaehlerPoolDictionary'
```

Genauso gehen wir bei der Vereinbarung der Klasse “ZaehlerOrWerteErfassung” vor:

```
OrWerteErfassung subclass: #ZaehlerOrWerteErfassung
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: 'ZaehlerPoolDictionary'
```

Durch diese Klassen-Vereinbarungen ist das von uns eingerichtete Dictionary “ZaehlerPoolDictionary” als Pool-Dictionary-Variable in beiden Klassen festgelegt.

Innerhalb jeder dieser beiden Klassen verabreden wir die Klassen-Methode “new” in der folgenden Form:

```
new
|varWerteErfassung|
varWerteErfassung := super new.
varWerteErfassung initZaehlerPool.
^ varWerteErfassung
```

Hierdurch ist bestimmt, daß bei der Instanz der Klasse “ZaehlerInWerteErfassung” bzw. der Klasse “ZaehlerOrWerteErfassung” stets die Instanz-Methode “initZaehlerPool” zur Ausführung gelangt.

Die Methode “initZaehlerPool” verabreden wir in den Klassen “ZaehlerInWerteErfassung” und “ZaehlerOrWerteErfassung” in der folgenden Form als Instanz-Methode:

```
initZaehlerPool
(ZaehlerPoolDictionary at: #ZaehlerPool) isNil
    ifTrue: [ZaehlerPoolDictionary at: #ZaehlerPool put: 0]
```

Um zu prüfen, ob noch keine Instanziierung vorgenommen wurde und daher dem Key “`#ZaehlerPool`” die Pseudovariable “nil” nach wie vor als Value zugeordnet ist, wird die Basis-Methode “isNil” eingesetzt.

- **“isNil”**:
  Handelt es sich beim Empfänger-Objekt der Message “isNil” um die Pseudovariable *“nil”*, so resultiert die Pseudovariable *“true”* als Ergebnis-Objekt – andernfalls ist das Ergebnis-Objekt gleich der Pseudovariablen *“false”*.

Um den aktuellen Zählerstand abzurufen, vereinbaren wir in den beiden Klassen “InWerteErfassung” und “OrWerteErfassung” die Klassen-Methode “anzeigenZaehlerPool” in der folgenden Form:

```
anzeigenZaehlerPool
Transcript cr;
    show: 'Anzahl der bislang insgesamt erfaßten Werte: ';
    show: (ZaehlerPoolDictionary at: #ZaehlerPool) printString
```

Hinweis: Die Methode “anzeigenZaehlerPool” hätten wir auch als Instanz-Methode vereinbaren können.

Als weitere Methode legen wir in beiden Klassen jeweils die Instanz-Methode “erfassenWert:” durch die Vereinbarung

```
erfassenWert: aPane
super erfassenWert: aPane.
ZaehlerPoolDictionary
   at: #ZaehlerPool
   put: ((ZaehlerPoolDictionary at: #ZaehlerPool) + 1)
```

fest, so daß sich insgesamt die folgende Situation ergibt:

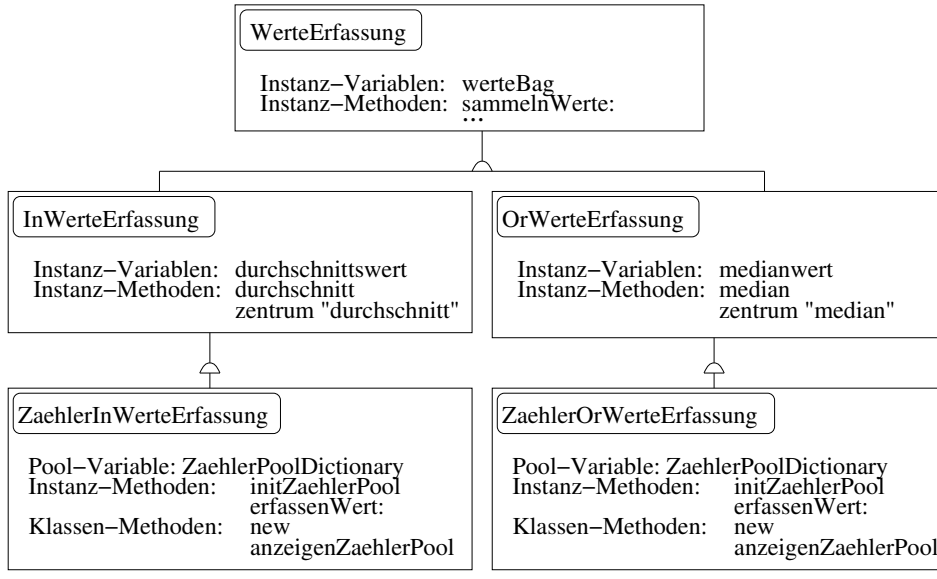


Abbildung 8.17: Vereinbarung von "ZaehlerInWerteErfassung" und "ZaehlerOrWerteErfassung"

Werden im Anschluß an die Anforderung

```
InWerteErfassung11 := ZaehlerInWerteErfassung new
  sammelnWerte: 'Jahrgangsstufe 11'
```

z.B. 30 Werte der Jahrgangsstufe 11 und im Anschluß an

```
InWerteErfassung12 := ZaehlerInWerteErfassung new
  sammelnWerte: 'Jahrgangsstufe 12'
```

für die Jahrgangsstufe 12 z.B. 25 Werte erfaßt, so wird sowohl durch die Anforderung

```
ZaehlerInWerteErfassung anzeigenZaehlerPool
```

als auch durch die Anforderung

```
ZaehlerOrWerteErfassung anzeigenZaehlerPool
```

der Wert "55" im Transcript-Fenster angezeigt.

Hinweis: Dabei ist zu beachten, daß wir die Methode "anzeigenZaehlerPool" als Klassen-Methode vereinbart haben.

8.4 Elemente einer Klassen-Vereinbarung

Nachdem wir Klassen-Variablen und Pool-Dictionary-Variablen kennengelernt haben, besitzen wir sämtliche Kenntnisse, um eine zusammenfassende Aussage über die charakteristischen Eigenschaften einer Klassen-Vereinbarung machen zu können.

Auf der Basis der im Abschnitt 5.3 angegebenen vorläufigen Form läßt sich eine Klassen-Vereinbarung insgesamt durch die folgende Anforderung festlegen:

```
<klassealt> subclass: #<klasseneu>
     instanceVariableNames: 'instanz-variablen'
     classVariableNames: 'klassen-variablen'
     poolDictionaries: 'pool-dictionary-variablen'
```

Durch diese Anforderung wird dem Empfänger-Objekt "<klassealt>" eine Keyword-Message mit den Selektoren "subclass:", "instanceVariableNames:", "classVariableNames:" und "poolDictionaries:" zugestellt.

Die neu eingerichtete Klasse, deren Name als Argument des Selektors "subclass:" aufgeführt werden muß, wird der Klasse "<klassealt>" unmittelbar untergeordnet und erhält den Klassennamen "<klasseneu>", d.h. "<klasseneu>" wird zur direkten Unterklasse der Oberklasse "<klassealt>".

Die Eigenschaften einer Instanz der Klasse "<klasseneu>" sind bestimmt durch

- den Besitz von *Instanz-Variablen*,
- die Zugriffsmöglichkeit auf *Klassen-Variablen* und
- auf *Pool-Dictionary-Variablen*.

Die dazu erforderlichen Angaben werden als Argumente hinter den Message-Selektoren "instanceVariableNames:", "classVariableNames:" und "poolDictionaries:" aufgeführt.
Sofern für einen dieser Selektoren keine Angabe gemacht werden soll, ist eine Zeichenkette in der Form (' ') als Argument anzugeben.

Zusammenfassend können wir die Zugriffsmöglichkeiten, die einzelne Instanzen auf temporäre Variablen, Instanz-Variablen, Pool-Dictionary-Variablen, Klassen-Variablen und globale Variablen besitzen, durch die nachfolgende Abbildung 8.18 beschreiben.

Wichtig ist der folgende Sachverhalt:

- Während der Zugriff auf die innerhalb einer Klasse vereinbarten Klassen- und Pool-Dictionary-Variablen von *jeder* Instanz dieser Klasse ausgeführt werden darf, kann eine Instanz immer nur auf ihre *eigenen* Instanz-Variablen zugreifen.

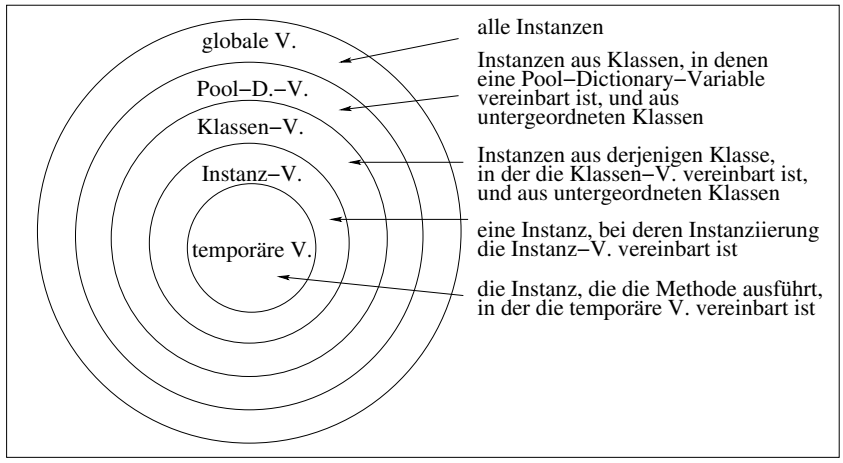


Abbildung 8.18: Zugriffsmöglichkeiten von Instanzen auf Variablen

8.5 Einordnung in die Klassen-Hierarchie des Basis-Systems

Bei den zuvor in Abschnitt 8.2 und 8.3 erläuterten Lösungen von PROB-6 und PROB-7 haben wir die Lösung einer Problemstellung als Spezialisierung einer Basis-Klasse bzw. einer zuvor neu eingerichteten Klasse entworfen.

Dieses Vorgehen ist typisch für die objekt-orientierte Programmierung, bei der versucht wird, einen Lösungsplan auf der Basis bereits vorhandener Klassen zu entwickeln.

Somit kommt der Kenntnis über die bereits aufgebaute Klassen-Hierarchie eine zentrale Bedeutung zu. In dem Moment, in dem bereits auf einen realisierten Lösungsplan und damit auf eine vereinbarte Klasse zurückgegriffen werden kann, sind im Zuge der Spezialisierung noch die benötigten Ergänzungen durch die Einrichtung einer oder mehrerer geeigneter Unterklassen vorzunehmen.

Schwierig wird es dann, wenn noch keine Vorstellung davon besteht, welche Klasse als Basis zur Lösung einer Problemstellung herangezogen werden kann. Genau diese Situation stellt sich grundsätzlich einem Neuling, der seine ersten Schritte bei der objekt-orientierten Programmierung unternimmt. Von einem Anfänger sind daher die folgenden Forderungen zu beachten:

- Sammle Erfahrungen!
- Versuche, die Hierarchie der Basis-Klassen im Hinblick auf die Objekte, die durch Instanziierungen eingerichtet werden können, zu durchschauen!
- Sofern sich eine Basis-Klasse anbietet, auf der ein Lösungsplan aufgesetzt werden kann, überdenke, ob nicht eine allgemeinere (Ober-)Klasse oder eine speziellere (Unter-)Klasse geeigneter ist!
- Unter Einsatz des Klassen-Hierarchie-Browsers erkunde die jeweiligen Klassen und damit die jeweils zur Verfügung stehenden Methoden!

Die ersten eigenen Schritte sollten darin bestehen, ein elementares Anwendungsbeispiel – wie z.B. die dargestellte Erfassung von Punktwerten – durch den Einsatz des SMALLTALK-Systems am Rechner nachzuvollziehen. Anschließend sollte versucht werden, die vorliegende Lösung zu modifizieren, indem z.B. nicht nur die Punktwerte, sondern mit ihnen gemeinsam das jeweilige Geschlecht des Schülers erfaßt wird, so daß anschließend geschlechts-spezifische Auswertungen der Punktwerte möglich sind. Desweiteren bietet es sich an, eine Spezialisierung der Lösungspläne anzustreben, indem die Punktwerte nicht mehr jahrgangsstufen-spezifisch getrennt, sondern insgesamt innerhalb eines einzigen Sammlers gespeichert werden sollen.
Im Hinblick auf derartig durchgeführte Spezialisierungen wird die Klassen-Hierarchie schrittweise verfeinert. Bei einer fortschreitenden Erweiterung besteht die Gefahr, daß der Überblick verlorengeht.
Hiervor kann man sich dadurch schützen, daß man aussagekräftige, sprechende Klassennamen wählt, die im Rahmen der Spezialisierungen geeignet erweitert werden, so daß die hierarchische Positionierung der einzelnen Klassen erkennbar ist.

In diesem Zusammenhang stellt sich die Frage, ob sich nicht die neu eingerichteten Klassen grundsätzlich so in die Hierarchie integrieren lassen, daß eine maximale Trennung der Anwendungen vom Basis-System erfolgen kann.

Die Antwort lautet, daß dies theoretisch und auch technisch möglich ist und diese Zielsetzung – im Hinblick auf die Intension des objekt-orientierten Programmierens – aus Effizienzgründen auch angestrebt werden sollte. Dabei ist jedoch zu beachten, daß derartige Gliederungen bestimmten Konventionen genügen müssen, damit Teile eines Lösungsplans unmittelbar für weitere Lösungspläne zur Verfügung stehen.

Hinweis: Ausführungen zu der Situation, bei der eine geeignete Gliederung des Lösungsplans und eine damit verbundene unterschiedliche Positionierung der resultierenden Klassen innerhalb der Klassen-Hierarchie vorgenommen wird, behandeln wir im Kapitel 13 unter der Thematik “Das Model/View-Konzept”.

Mit “technisch möglich” ist gemeint, daß jede neue Klasse als direkte Unterklasse der Basis-Klasse “Object” festgelegt werden kann. Dies liegt daran, daß innerhalb einer Klasse – egal auf welcher Hierarchiestufe – Variablen in Form von Instanz-Variablen eingesetzt werden können, denen sich Instanzen beliebiger Klassen zuordnen lassen. Bei der Verwendung dieser Variablen ist allein zu berücksichtigen, daß sie in geeigneter Weise als Empfänger-Objekte derjenigen Messages verwendet werden, die – im Hinblick auf den Lösungsplan – innerhalb von Anforderungen formuliert werden müssen.

Hinweis: Zum Beispiel kann die von uns zur Lösung von PROB-1-1 festgelegte Klasse “WerteErfassung”, die als Unterklasse der Basis-Klasse “ViewManager” eingerichtet wurde, auch als direkte Unterklasse von “Object” aufgebaut werden.

Dazu ist innerhalb der Klasse “WerteErfassung” eine zusätzliche Instanz-Variable mit z.B. dem Namen “fenster” zu vereinbaren.

Zu Beginn der Methode “initialisierenErfassung” muß durch die Zuweisung

```
fenster := ViewManager new
```

dieser Instanz-Variablen “fenster” eine Instanz der Basis-Klasse “ViewManager” zugeordnet werden, so daß “fenster” alle Methoden von ViewManager bekannt sind.

Ergänzend muß in der Methoden-Vereinbarung von “initialisierenErfassung” – unmittelbar vor der Message ”labelWithoutPrefix:” statt des Argumentes “self” die Instanz-Variable

"fenster" als Argument innerhalb der Message "owner:" aufgeführt werden.

Zusätzlich ist die Pseudovariable "self" durchgängig in den Methoden "durchfuehrenErfassung", "entfernenErfassungsfenster:", "erfassenWert:", "festlegenUeberschrift:" und "sammelnWerte:" durch die Instanz-Variable "fenster" zu ersetzen.

Dieses Vorgehen sichert, daß die Klasse "WerteErfassung" in dieser abgewandelten Form als direkte Unterklasse von "Object" die Lösung von PROB-1-1 sicherstellt.
Diese "technische Lösung" ist jedoch nicht empfehlenswert, da sie der Zielsetzung der objekt-orientierten Programmierung im Hinblick auf die These "vom Allgemeinen zum Speziellen" nur indirekt gerecht wird. Dies liegt vor allen Dingen daran, daß einem Aussenstehenden erst durch die Kenntnis der innerhalb von "WerteErfassung" vereinbarten Methoden klar wird, daß es sich bei dieser Klasse um eine Fenster-Anwendung, d.h. um eine Spezialisierung der Basis-Klasse "ViewManager" handelt.

Abschließend weisen wir darauf hin, daß es sinnvoll ist, die Anforderungen, die zur Ausführung eines Lösungsplans benötigt werden, als Methoden-Vereinbarung mittels eines charakteristischen Selektors wie z.B. "imWorkspaceFensterPROB11" als Klassen-Methode in derjenigen Klasse festzulegen, die den Lösungsplan repräsentiert.

So ist es z.B. sinnvoll, die Methode "imWorkspaceFensterPROB11" wie folgt als Klassen-Methode innerhalb der Klasse "WerteErfassung" zu verabreden:

```
imWorkspaceFensterPROB11
WerteErfassung11 := WerteErfassung new.
WerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

Ein derartiges Vorgehen dokumentiert die Anforderungen, die zur Lösung eines Problems erforderlich sind, sowie die Klasse(n) und die Methode(n), die unter Einsatz des SMALLTALK-Systems entwickelt und als zusätzlicher Bestandteil des SMALLTALK-Systems eingerichtet wurden.

Kapitel 9

Einrichtung und Verarbeitung von Sammlern

In den vorausgehenden Kapiteln haben wir zur Lösung unserer Problemstellungen geeignete *Sammler* zur Speicherung von Objekten verwendet, die sich als Instanzen z.B. der Basis-Klassen "Bag" und "OrderedCollection" einrichten ließen.

Nachdem wir bislang lösungs-orientiert vorgegangen sind und dabei die jeweils benötigten Sammler und die für sie verwendeten Methoden in den Abschnitten vorgestellt haben, in denen sie zur Umsetzung eines Lösungsplans benötigt wurden, geben wir im folgenden einen systematischen Überblick.

Grundsätzlich ist festzustellen:

- Jeder *Sammler* muß als Instanz einer geeigneten Unterklasse der Basis-Klasse "Collection" eingerichtet werden.

9.1 Unterklassen der Basis-Klasse "Collection"

Zur Sammlung von beliebigen Objekten lassen sich Instanzen der Klasse "Bag" immer dann einsetzen, wenn kein direkter Zugriff auf Objekte des Sammlers erforderlich ist, die Anzahl der im Sammler gesicherten Objekte variabel sein soll und es unerheblich ist, ob ein Objekt mehrfach gespeichert ist.

Sofern jedoch z.B. die Forderung, daß sich *beliebig* viele Objekte sammeln lassen, erfüllt und zusätzlich gesichert sein soll, daß ein Objekt höchstens *einmal* innerhalb eines Sammlers auftreten darf, sind Instanzen der Klasse "Bag" ungeeignet.
In diesem Fall muß die Sicherung der zu sammelnden Objekte in Form eines *"Sets"* durchgeführt werden.

Andere Formen von Sammlern sind z.B. in den Fällen zu wählen, in denen der Zugriff auf einzelne gesammelte Objekte – über einen Zugriffsschlüssel wie z.B. einen Key oder eine Index-Position – ermöglicht oder die Speicherung in sortierter Reihenfolge vorgenommen werden soll.

Derartige Forderungen lassen sich dadurch erfüllen, daß geeignete Instanziierungen aus anderen Basis-Klassen des SMALLTALK-Systems vorgenommen werden. In dieser Hinsicht sind alle Unterklassen der Basis-Klasse "Collection" von Interesse.

Eine geeignete Auswahl dieser Klassen und deren hierarchische Gliederung gibt die folgende Darstellung wider:

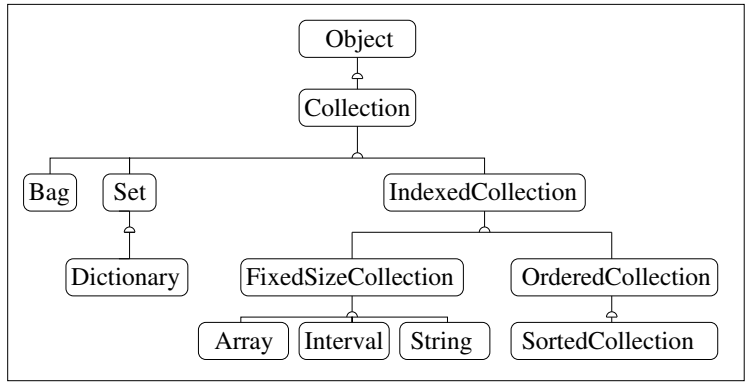


Abbildung 9.1: Unterklassen von "Collection"

Hinweis: Innerhalb des Klassen-Hierarchie-Browser-Fensters wird die hierarchische Strukturierung wie folgt angegeben:

```
Collection
 Bag
 Set
   Dictionary
 IndexedCollection
   FixedSizeCollection
     Array
     Interval
     String
   OrderedCollection
     SortedCollection
```

Als Unterklasse von "String" ist die Basis-Klasse "Symbol", deren Instanziierungen wir im Abschnitt 5.3 vorgestellt haben, innerhalb der Klassen-Hierarchie eingeordnet.

Durch diese Gliederung ist festgelegt, in welcher Reihenfolge die einzelnen Klassen nach Methoden durchsucht werden, sofern deren Ausführung von Instanzen dieser Klassen angefordert wird.

In der oben angegebenen Darstellung handelt es sich bei den Klassen "Collection", "IndexedCollection" und "FixedSizeCollection" um *abstrakte* Klassen, in denen Methoden zur Verfügung gehalten werden, die von jedem Sammler bzw. nur von Sammlern mit speziellen Eigenschaften ausgeführt werden können.

Als Beispiel einer generell zur Verfügung stehenden Methode ist die Basis-Methode "isCollection" innerhalb der Klasse "Collection" vereinbart, mit der sich prüfen läßt, ob es sich beim Empfänger-Objekt dieser Message um einen Sammler handelt.

- **"isCollection"**:
  Durch die Ausführung dieser Methode läßt sich prüfen, ob das Empfänger-

Objekt eine Instanz der Klassen "Bag", "Set", "Dictionary", "Array", "Interval", "String", "OrderedCollection" oder "SortedCollection" ist.
Ist dies zutreffend, so ist die Pseudovariable "true" als Ergebnis-Objekt festgelegt – andernfalls die Pseudovariable "false".

Während die Methode "collect:", die innerhalb der Klasse "Collection" festgelegt ist, zur generellen Verfügung gehalten wird, ist eine Methode gleichen Namens zusätzlich in der abstrakten Klasse "FixedSizeCollection" vereinbart.
Sie überdeckt die Methode "collect:" aus der Klasse "Collection" und wird für den Aufbau derjenigen Sammler benötigt, in denen eine vorab festgelegte Anzahl von Objekten gesammelt werden soll.

9.2 Eigenschaften der Unterklassen von "Collection"

Bevor wir Unterklassen von "Collection" und ausgewählte Methoden dieser Unterklassen näher beschreiben, geben wir zunächst die wichtigsten Eigenschaften dieser Unterklassen in summarischer Form an:

- Bag: *Ungeordneter* Sammler mit *variabler* Anzahl von Objekten, von denen gleiche Objekte mehrfach auftreten dürfen und auf die *kein* direkter Zugriff möglich ist;
- Set: *Ungeordneter* Sammler mit *variabler* Anzahl von Objekten, von denen *keine* Objekte mehrfach auftreten dürfen und auf die *kein* direkter Zugriff möglich ist;
  - Dictionary: *Ungeordneter* Sammler mit *variabler* Anzahl von Key-Value-Paaren, bei denen über einen Key direkt auf einen Value zugegriffen werden kann, wobei *kein* Key mehrfach auftreten kann.
- IndexedCollection: Alle Sammler, die aus Instanziierungen von Unterklassen dieser abstrakten Klasse resultieren, haben die Eigenschaft, daß die gesammelten Objekte – gemäß einer bestimmten Reihenfolgevorschrift – geordnet sind und daß ein direkter Zugriff – über eine ganze Zahl als Index-Position – auf diese Objekte möglich ist.
  - FixedSizeCollection: Alle Sammler, die aus Instanziierungen von Unterklassen dieser abstrakten Klasse resultieren, erfüllen neben der Eigenschaft, daß sie geordnet sind und daß ein direkter Zugriff – über eine ganze Zahl als Index-Position – auf ihre Objekte möglich ist, die Zusatzeigenschaft, daß die Anzahl ihrer Objekte *nicht variabel* ist.
    - Array: *Geordneter* Sammler von beliebigen Objekten, der *nicht variabel* vereinbart wird und auf dessen Objekte (gleiche Objekte dürfen mehrfach auftreten) direkt zugegriffen werden kann;

- Interval: *Geordneter* Sammler von Zahlen, der *nicht variabel* vereinbart wird und auf dessen Zahlen (gleiche Zahlen dürfen *nicht mehrfach* auftreten) direkt zugegriffen werden kann;
- String: *Geordneter* Sammler von Zeichen, der *nicht variabel* vereinbart wird und auf dessen Zeichen (gleiche Zeichen dürfen mehrfach auftreten) direkt zugegriffen werden kann.

- OrderedCollection: Gemäß der *Einfügereihenfolge* geordneter Sammler mit variabler Anzahl von Objekten, auf die ein direkter Zugriff möglich ist und von denen gleiche Objekte mehrfach auftreten dürfen.
  - SortedCollection: Gemäß der *Sortierreihenfolge* geordneter Sammler mit variabler Anzahl von Objekten, auf die ein direkter Zugriff möglich ist und von denen gleiche Objekte mehrfach auftreten dürfen.

Diese Eigenschaften lassen sich tabellarisch in Form der folgenden Übersicht darstellen:

Klasse	Ordnung	identische Objekte möglich	Anzahl sammelbarer Objekte	direkter Zugriff möglich	Art der gesammelten Objekte
Collection					
Bag	nein	ja	variabel	nein	beliebig
Set	nein	nein	variabel	nein	beliebig
Dictionary	nein	nein	variabel	Key	beliebige Key–Value–Paare
IndexedCollection					
FixedSizeCollection					
Array	ja, intern	ja	fest	Ganzzahl	beliebig
Interval	ja, intern	nein	fest	Ganzzahl	Zahlen
String	ja, intern	ja	fest	Ganzzahl	Zeichen
OrderedCollection	ja, gemäß Einfüge–reihen–folge	ja	variabel	Ganzzahl	beliebig
SortedCollection	ja, gemäß Sortier–folge	ja	variabel	Ganzzahl	beliebig

Abbildung 9.2: Eigenschaften der Unterklassen von “Collection”

Teilt man die Sammler im Hinblick darauf, ob Objekte geordnet oder ungeordnet gespeichert werden und über einen Zugriffsschlüssel direkt zugreifbar sind, in drei Gruppen ein, so ergibt sich die folgende Gliederung:

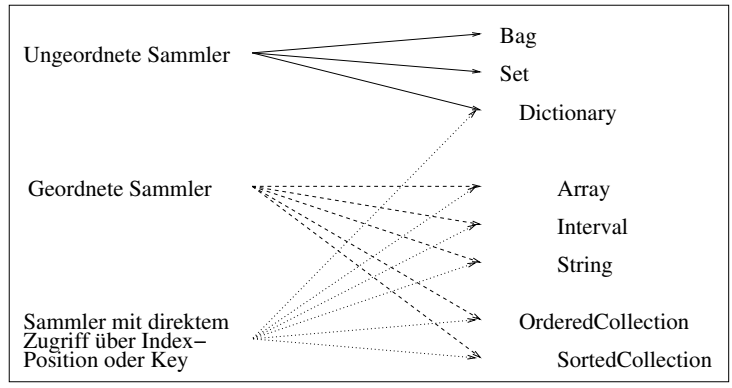


Abbildung 9.3: Einteilung der Unterklassen von “Collection”

9.3 Sammler ohne direkten Zugriff

9.3.1 Die Basis-Klasse “Bag”

Als unmittelbare Unterklasse der abstrakten Klasse “Collection” ist die Basis-Klasse “Bag” in der Klassen-Hierarchie eingetragen.
Ein Bag stellt die allgemeinste Form eines Sammlers dar.

- Bei einem Bag ist die Anzahl der sammelbaren Objekte *variabel*, so daß sie nicht bei der Instanziierung festgelegt werden muß.
- Es können *beliebige* Objekte gesammelt werden. Die Sammlung der Objekte erfolgt *ungeordnet*, so daß die Objekte nicht gemäß einer Reihenfolge gespeichert werden. Ferner ist auch kein direkter Zugriff auf ein einzelnes Objekt über einen Zugriffsschlüssel möglich.
- Die gesammelten Objekte müssen *nicht eindeutig* sein, so daß gleiche Objekte mehrfach auftreten dürfen.

In der Klasse “Bag” stehen Basis-Methoden zur Verfügung, mit denen sich Instanzen der Klasse “Bag” erweitern, prüfen oder iterativ bearbeiten lassen. In dieser Hinsicht sind uns unter anderem die Methoden *“add:”*, *“remove:”*, *“occurrencesOf:”*, *“size”* und *“do:”* bekannt.
Hinweis: Neben der Methode “remove:” steht z.B. die Basis-Methode “remove:ifAbsent:” zur Verfügung. Mit dieser Methode kann in dem Fall, in dem ein Objekt aus einem Bag gelöscht werden soll, das nicht vorhanden ist, eine qualifizierte Meldung im Transcript-Fenster angezeigt werden – z.B. ist die Anforderung

```
VarBag remove:99 ifAbsent:[Transcript cr; show: 'ungültiger Wert']
```

möglich.

9.3.2 Die Basis-Klasse “Set”

Muß bei einer Sammlung von Objekten sichergestellt werden, daß kein Objekt mehrfach auftritt, ist der Einsatz eines Bags nicht geeignet. In dieser Situation besteht die Möglichkeit, eine Instanziierung der Basis-Klasse “Set” durchzuführen.

- Unter einem *Set* wird eine Sammlung von Objekten verstanden, die bis auf die Ausnahme, daß gesammelte Objekte *nicht mehrfach* auftreten dürfen, die Eigenschaften eines Bags besitzt.

Genau wie in der Klasse “Bag” sind innerhalb der Klasse “Set” Methoden – wie z.B. “add:”, “size”, “remove:”, “do:” und “collect:” – festgelegt, mit denen sich die Eigenschaften von Instanzen der Klasse “Set” prüfen, ändern oder iterativ bearbeiten lassen.

Wichtig sind die in der Klasse “Collection” vereinbarten Methoden, mit denen der Inhalt eines Sammlers in einen anderen Sammler übertragen werden kann.

Sollen z.B. mehrfach auftretende Objekte eines Bags entfernt werden, so ist zunächst – durch die Basis-Methode “asSet” – der Inhalt eines Bags in ein Set zu übernehmen. Anschließend sind die resultierenden Objekte – durch die Basis-Methode “asBag” – wiederum als Inhalt eines Bags bereitzustellen.

- **“asSet”**:
  Das Ergebnis-Objekt der Message “asSet” ist ein Set, dessen Inhalt aus den Objekten desjenigen Sammlers aufgebaut wird, der als Empfänger-Objekt der Message “asSet” aufgeführt ist. Dabei werden mehrfach auftretende Objekte nur in einfacher Ausführung übernommen.

- **“asBag”**:
  Das Ergebnis-Objekt der Message “asBag” ist ein Bag, dessen Inhalt aus den Objekten desjenigen Sammlers aufgebaut wird, der als Empfänger-Objekt der Message “asBag” aufgeführt ist.

Hinweis: Beim Einsatz dieser Messages ist zu beachten, daß das ursprüngliche Empfänger-Objekt erhalten bleibt.

Ist z.B. “VarBag” als Bag eingerichtet worden, so lassen sich durch die Anforderung

```
VarBag := VarBag asSet asBag
```

sämtliche mehrfach auftretenden Objekte entfernen.

Hinweis: Wird anstelle dieser Zuweisung allein die Anforderung

```
VarBag asSet asBag
```

ausgeführt, so bleibt das Empfänger-Objekt “VarBag” in seiner ursprünglichen Form erhalten.

9.4 Sammler mit direktem Zugriff

Sofern die gesammelten Objekte direkt zugreifbar sein sollen, dürfen es *keine* Instanziierungen der Klassen "Bag" und "Set" sein. In diesem Fall müssen sie in Abhängigkeit vom jeweiligen Lösungsplan aus Instanziierungen der Klasse "Dictionary", der Unterklassen der abstrakten Klasse "FixedSizeCollection" oder der Klassen "OrderedCollection" bzw. "SortedCollection" resultieren.

9.4.1 Die Basis-Klasse "Dictionary"

Die Basis-Klasse "Dictionary" ist eine direkte Unterklasse der Klasse "Set". Eine Instanz dieser Klasse wird als *Dictionary* bezeichnet. Über die Eigenschaften eines Dictionarys haben wir im Abschnitt 8.3 folgendes festgestellt:

- Ein *Dictionary* ist eine Sammlung von Key-Value-Paaren, bei denen ein Key – als Zugriffsschlüssel – *eindeutig* auf einen Value weist, bei dem es sich um ein beliebiges Objekt handeln kann.

Hinweis: Key-Value-Paare sind Instanzen der Basis-Klasse "Association".
Wird eine Variable an der Position eines Keys innerhalb einer Message eingetragen, so wird dasjenige Objekt als Key verwendet, auf das diese Variable verweist.

Dictionarys werden z.B. dann eingesetzt, wenn in Lösungsplänen festgelegt ist, daß Objekte über einen Schlüssel assoziativ zugreifbar sein sollen – wie es z.B. bei einem Lexikon der Fall ist, bei dem Texte über einzelne Wörter auffindbar sind.

Key-Value-Paare lassen sich durch die Basis-Methode *"at:put:"* in ein Dictionary eintragen.

- **"at:put:"**:
  In dem Dictionary, das als Empfänger-Objekt der Message "at:put:" aufgeführt ist, wird ein neues Key-Value-Paar als Objekt eingetragen. Dabei wird das Argument von "at:" als Key und das Argument von "put:" als Value verwendet.
  Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Key zugeordnet wird. Es ist zu beachten, daß ein Key *nicht mehrfach* auftreten kann.

  Hinweis: Ist ein hinter "at:" angegebener Key bereits vorhanden, so wird der zugehörige Value durch das als Argument von "put:" aufgeführte Objekt ersetzt.

Zum Beispiel bewirken die Anforderungen

```
VarDic := Dictionary new.
VarDic at: 'eins' put: 'one'; at: 'zwei' put: 'two'
```

die Einrichtung einer Instanz der Klasse "Dictionary", in der zwei Key-Value-Paare gesammelt sind. Dabei ist dem Key 'eins' der Value 'one' und dem Key 'zwei' der Value 'two' zugeordnet.

Hinweis: Sind wir nicht am Ergebnis-Objekt der Message "at:put:", sondern an der Anzeige des resultierenden Empfänger-Objektes interessiert, so können wir die Message "yourself" oder aber eine primäre Message durch die alleinige Angabe des Dictionarys

"VarDic" einsetzen. Wir erhalten in beiden Fällen – beim Einsatz von "Show It" des Menüs "Smalltalk" – eine Anzeige in der Form "Dictionary('one' 'two')".

Um die Values eines Dictionarys zu ermitteln, kann die Basis-Methode *"values"* verwendet werden.

- **"values"**:
  Als Ergebnis-Objekt der Message "values" resultiert ein Bag, in dem sämtliche Values aus demjenigen Dictionary enthalten sind, das als Empfänger-Objekt der Message "values" aufgeführt ist.

Informieren wir uns z.B. durch die Message

```
VarDic values
```

über die Values von "VarDic", so erhalten wir als Ergebnis-Objekt eine Instanz der Klasse "Bag" – z.B. in Form von "Bag('one' 'two')".

Soll ein Key-Value-Paar aus einem Dictionary entfernt werden, so ist die Basis-Methode "removeKey:" einzusetzen.

- **"removeKey:"**:
  In dem Dictionary, das als Empfänger-Objekt der Message "removeKey:" aufgeführt ist, wird dasjenige Key-Value-Paar gelöscht, dessen Key mit dem als Argument von "removeKey:" angegebenen Key übereinstimmt.
  Als Ergebnis-Objekt resultiert das geänderte Dictionary.

Zum Beispiel können wir durch die Message

```
VarDic removeKey: 'zwei'
```

das Key-Value-Paar mit dem Key 'zwei' aus dem Sammler "VarDic" löschen.

Hinweis: Um eine qualifizierte Fehleranzeige für den Fall anzufordern, daß der Key nicht existiert, ist anstelle von "removeKey:" die Basis-Methode "removeKey:ifAbsent:" z.B. in der Form

```
VarKey removeKey: 'zwei' ifAbsent:[Transcript cr; show: 'ungültiger Key']
```

zu verwenden.

Beim Einsatz eines Dictionarys kann nicht nur mit einem Key auf den zugeordneten Value, sondern auch mit einem Value auf einen Key, der auf den vorgegebenen Value weist, zugegriffen werden.

Dazu stehen die Basis-Methoden "at:" und "keyAtValue:" zur Verfügung.

- **"at:"**:
  Als Ergebnis-Objekt der Message "at:" resultiert der Value, der dem als Argument aufgeführte Key in demjenigen Dictionary zugeordnet ist, das als Empfänger-Objekt der Message "at:" angegeben ist.

Zum Beispiel erhalten wir als Ergebnis-Objekt der Message

```
VarDic at: 'eins'
```

die Zeichenkette 'one'.

- **"keyAtValue:"**:
  Als Ergebnis-Objekt der Message "keyAtValue:" resultiert der Key, der auf den als Argument aufgeführten Value in demjenigen Dictionary weist, das als Empfänger-Objekt der Message "keyAtValue:" angegeben ist.
  Ist der als Argument von "keyAtValue:" aufgeführte Value mehrfach vorhanden, so wird der intern zuerst gefundene Key ermittelt.

Zur Prüfung, wie oft ein bestimmtes Objekt innerhalb eines Dictionarys als Value enhalten ist bzw. ob ein Key innerhalb eines Dictionarys existiert, stehen die Basis-Methoden *"occurrencesOf:"* und *"includesKey:"* zur Verfügung.

- **"occurrencesOf:"**:
  Als Ergebnis-Objekt der Message "occurrencesOf:" resultiert die Pseudovariable "true", sofern das als Argument aufgeführte Objekt in demjenigen Dictionary als Value enthalten ist, das als Empfänger-Objekt der Message " occurrencesOf:" angegeben ist. Andernfalls ergibt sich die Pseudovariable "false".

- **"includesKey:"**:
  Als Ergebnis-Objekt der Message "includesKey:" resultiert die Pseudovariable "true", sofern der als Argument aufgeführte Key in demjenigen Dictionary enthalten ist, das als Empfänger-Objekt der Message "includesKey:" angegeben ist. Andernfalls ergibt sich die Pseudovariable "false".

Während die bisher vorgestellten Methoden den Zugriff auf einzelne Keys bzw. einzelne Key-Value-Paare ermöglichten, setzen wir nachfolgend die Basis-Methoden "associationsDo:" und "keysDo:" ein, mit denen der Zugriff auf sämtliche Keys bzw. Key-Value-Paare möglich ist.

- Sollen die Key-Value-Paare eines Dictionarys *gemeinsam* verarbeitet werden, so läßt sich dies durch die Basis-Methode **"associationsDo:"** erreichen.

Wollen wir z.B. sämtliche Key-Value-Paare von "VarDic" anzeigen lassen, so können wir die folgende Anforderung stellen:

```
VarDic associationsDo: [:einPaar|Transcript cr;
                                  show: einPaar printString]
```

Hinweis: Durch die Ausführung dieser Anforderung erhalten wir " 'eins' ==> 'one' " im Transcript-Fenster angezeigt.

- Um die Gesamtheit aller Keys eines Dictionarys zu verarbeiten, läßt sich die Basis-Methode **"keysDo:"** einsetzen.

Sollen z.B. sämtliche Keys von "VarDic" angezeigt werden, so können wir die Anforderung

```
VarDic keysDo:[:einKey|Transcript cr; show:einKey printString]
```

stellen.

Hinweis: Zur Anzeige der Keys können wir auch die unäre Message “keys” in der Form “VarDic keys” einsetzen. Diese Message liefert als Ergebnis-Objekt eine Instanz der Klasse “Set”.

9.4.2 Das System-Dictionary “Smalltalk” und das Methoden-Dictionary

Das System-Dictionary “Smalltalk”

Als besondere Form eines Dictionarys haben wir im Abschnitt 8.3 das *System-Dictionary* in Form der globalen Variablen “Smalltalk” vorgestellt, die als *einzige* Instanz aus der Basis-Klasse “SystemDictionary” als Dictionary mit Key-Value-Paaren existiert.

Dieser Sammler enthält sämtliche *Namen* der im SMALLTALK-System vereinbarten globalen Objekte, d.h. alle Klassennamen und die Namen aller globalen Variablen (d.h. auch die Namen der vereinbarten Pool-Dictionary-Variablen). Diese Namen sind in Form von Symbolen gespeichert, da sämtliche Keys des *System-Dictionarys* Instanzen der Basis-Klasse “Symbol” sind. Als Values sind diesen Symbolen jeweils Instanzen bzw. Klassen zugeordnet.

Wurde z.B. der Variablen “WerteErfassung11” eine Instanz der Klasse “WerteErfassung” und der Variablen “VarBag” eine Instanz der Klasse “Bag” zugeordnet, in der die Objekte “32” und “37” gesammelt wurden, so stellt die folgende Abbildung einen Ausschnitt aus dem *System-Dictionary* dar:

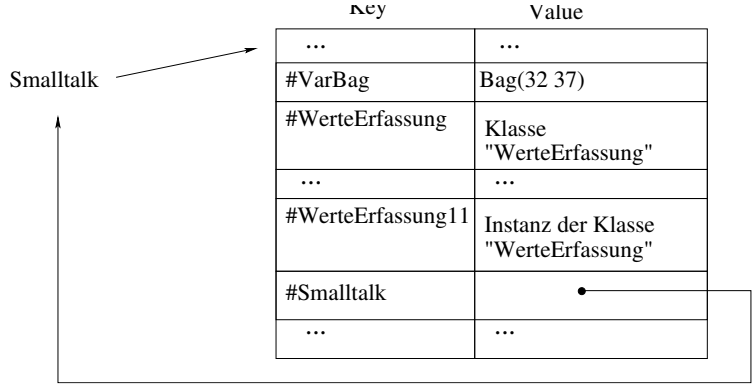


Abbildung 9.4: System-Dictionary

Hinweis: Da die Variable “Smalltalk” ebenfalls ein globales Objekt des SMALLTALK-Systems ist, erscheint das Symbol “#Smalltalk” in dieser Tabelle ebenfalls als Key.

Um z.B. das Key-Value-Paar zu ermitteln, auf das die Variable “WerteErfassung11” weist, läßt sich die Message

```
Smalltalk lookUpKey: #WerteErfassung11
```

einsetzen. Es resultiert

```
WerteErfassung11 ==> a WerteErfassung
```

als Ergebnis-Objekt.

- Dabei kennzeichnet die Anzeige von "a WerteErfassung" den Sachverhalt, daß es sich bei dem betreffenden Objekt um eine Instanz der Klasse "WerteErfassung" handelt.

Um z.B. *sämtliche* globalen Variablen, die auf Instanzen der Klasse "InWerteErfassung" verweisen, anzeigen zu lassen, können wir die folgende Anforderung verwenden:

```
Smalltalk associationsDo:
  [:einPaar|einPaar value class == WerteErfassung
            ifTrue:[Transcript cr;
                    show: einPaar key printString]]
```

Hinweis: Innerhalb dieser Anforderung haben wir die Methoden "value" und "key" aus der Basis-Klasse "Association" verwendet, um auf die Keys bzw. Values aller Key-Value-Paare im *System-Dictionary* "Smalltalk" zuzugreifen.

Um uns z.B. darüber zu informieren, ob Instanzen der Klasse "WerteErfassung" existieren, können wir die Methode "allInstances" der Basis-Klasse "Behavior" wie folgt einsetzen:

```
WerteErfassung allInstances
```

Hinweis: Sofern die Instanz "WerteErfassung11" eingerichtet wurde, resultiert "(a WerteErfassung)" – als Indiz für die Existenz einer Instanz – als Ergebnis-Objekt.

Soll z.B. die Instanz "WerteErfassung11" von "WerteErfassung" gelöscht werden, so läßt sich dies durch die folgende Anforderung erreichen:

```
Smalltalk removeKey: #WerteErfassung11
```

Hinweis: Sollen *alle* Instanzen von "WerteErfassung" gelöscht werden, so läßt sich dies über die Anforderung
`WerteErfassung allInstances do:[:einObjekt|Smalltalk removeKey:einObjekt]`
erreichen.

Zur Anzeige und Bearbeitung des *System-Dictionarys* "Smalltalk" kann auch der Klassen-Hierarchie-Browser eingesetzt werden.

Um z.B. sämtliche globalen Variablen anzuzeigen, ist zunächst die globale Variable "Smalltalk" in das Workspace-Fenster einzutragen und zu markieren. Durch die Bestätigung der Menü-Option "Inspect It" des Menüs "Smalltalk" läßt sich anschließend anfordern, daß die Namen der globalen Objekte im Fenster "Smalltalk Express Inspecting: SystemDictionary" angezeigt werden.

Soll eine globale Variable gelöscht werden, so ist sie zu markieren und die Menü-Option “Remove” des Menüs “Dictionary” zu bestätigen.

Soll eine globale Variable dem *System-Dictionary* hinzugefügt werden, so ist im Fenster “Smalltalk Express Inspecting: SystemDictionary” die Menü-Option “Add” des Menüs “Dictionary” zu bestätigen. Anschließend kann der Name der globalen Variablen im Fenster “Smalltalk Express Prompter” eingetragen werden.

Das Methoden-Dictionary

Jeder vereinbarten Klasse ist ein *Methoden-Dictionary* in Form einer Instanz der Basis-Klasse “MethodDictionary” zugeordnet, in dem die Methoden dieser Klasse gesammelt sind.

Hinweis: Bei den Methoden handelt es sich um Zeichenketten-Objekte, die den Keys in Form von Symbolen als Values zugeordnet sind.

Für die Klasse “WerteErfassung” können wir uns diese Sammlung folgendermaßen vorstellen:

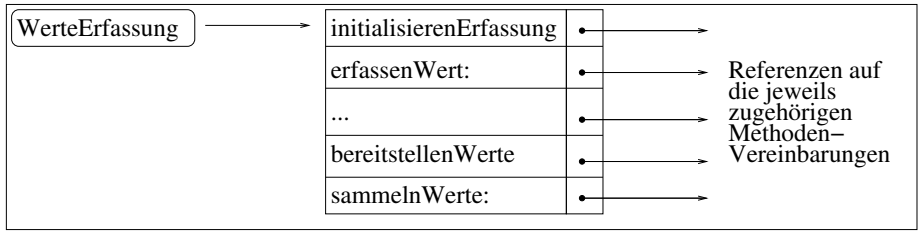


Abbildung 9.5: “WerteErfassung” zugeordnetes Methoden-Dictionary

Um z.B. sämtliche Namen der Instanz-Methoden von “WerteErfassung” anzeigen zu lassen, können wir die Basis-Methode “methodDictionary” verwenden und die folgende Anforderung stellen:

```
WerteErfassung methodDictionary associationsDo:
     [:einPaar|Transcript cr; show: einPaar key]
```

Zur Anzeige der Klassen-Methoden von “WerteErfassung” – d.h. der Instanz-Methoden der zugehörigen Meta-Klasse –, läßt sich die Anforderung

```
WerteErfassung class methodDictionary associationsDo:
     [:einPaar|Transcript cr; show: einPaar key]
```

stellen.

Um sich diejenigen Klassen anzeigen zu lassen, in denen eine bestimmte Methode vereinbart ist, kann die Message “implementorsOf:” verwendet werden. Zum Beispiel erhalten wir durch

```
Smalltalk implementorsOf: #class
```

den Klassennamen “Object” im eröffneten Fenster in der Form

```
Object>>class
```

angezeigt.

9.4.3 Die Basis-Klassen “IndexedCollection” und “FixedSizeCollection”

Die Basis-Klasse “IndexedCollection”

Wie wir zuvor festgestellt haben, ist die abstrakte Klasse “IndexedCollection” eine Oberklasse aller derjenigen Klassen, deren Instanzen die Eigenschaft besitzen, daß ein direkter Zugriff über eine Index-Position auf die in ihnen gesammelten Objekte möglich ist.

- Dabei wird unter einer Index-Position – kurz: *“Index”* – ein ganzzahliger Zugriffsschlüssel verstanden.

Wird in einen derartigen Sammler erstmalig ein Objekt eingetragen, so ist dieses Objekt über den Index “1” zugreifbar. Wird ein zweites Objekt dem Sammler hinzugefügt, so korrespondiert der Index “2” mit diesem Objekt, usw.
Von der Index-Position wird Gebrauch gemacht, wenn einem Sammler ein Objekt mittels der Message “at:put:” hinzugefügt oder ein Objekt durch den Einsatz der Message “at:” geprüft werden soll.

- **“at:put:”**:
  Durch die Basis-Methode “at:put:” wird dem Index, der hinter dem Selektor “at:” als Argument angegeben ist, dasjenige Objekt zugeordnet, das als Argument des Selektors “put:” aufgeführt wird. Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Index zugeordnet wird.
  Dabei wird das Objekt in demjenigen Sammler eingetragen, der als Empfänger-Objekt innerhalb der Message “at:put:” aufgeführt ist.

- **“at:”**:
  Durch die Basis-Methode “at:” wird mit dem Index, der als Argument hinter dem Selektor “at:” aufgeführt ist, auf das Objekt zugegriffen, das dem Index innerhalb des Sammlers, der als Empfänger-Objekt der Message “at:” verwendet wird, als Objekt zugeordnet ist. Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Index zugeordnet ist.

Objekte aus Sammlern, die aus Klassen instanziiert wurden, die der Basis-Klasse “IndexedCollection” untergeordnet sind, können unter Einsatz der Basis-Methode “,” (Komma) *aneinandergereiht* werden.

- **“,”**:
  Als Ergebnis-Objekt der binären Message “,” resultiert ein Sammler, der aus

derjenigen Klasse instanziiert wird, aus der der als Empfänger-Objekt aufgeführte Sammler instanziiert wurde. In diesem Sammler werden diejenigen Objekte aneinandergereiht, die im Empfänger-Objekt und im Argument der binären Message "," gesammelt sind. Dabei wird die Reihung der Objekte von den Objekten eingeleitet, die aus dem Empfänger-Objekt übernommen wurden.

Zum Beispiel kann man einen Sammler "VarGesamt" durch die Anforderung

```
VarGesamt := VarIC1,VarIC2
```

einrichten, in dem sämtliche Objekte gesammelt werden, die in den Sammlern "VarIC1" und "VarIC2" enthalten sind. Dabei werden die in "VarIC2" gesammelten Objekte den Objekten von "VarIC1" angefügt.

Die Basis-Klasse "FixedSizeCollection"

Unterhalb der abstrakten Klasse "FixedSizeCollection" sind alle diejenigen Klassen als Unterklassen von "IndexedCollection" vereinbart, bei denen die Anzahl ihrer Objekte *nicht variabel* ist. Dies bedeutet, daß bei einer Instanziierung die Anzahl derjenigen Objekte festgelegt werden muß, die innerhalb des Sammlers enthalten sein können. Die jeweils vorzusehende Anzahl ist *unveränderbar* und kann folglich nicht erhöht werden.
Zum Beispiel wird durch die Anforderung

```
VarArray := Array new: 3
```

bestimmt, daß "VarArray" auf eine Instanz der Klasse "Array" weist, in der 3 Objekte gesammelt werden können (siehe unten).
In diesem Fall weist "VarArray" auf eine Sammlung, in der die Pseudovariable "nil" in dreifacher Ausführung gespeichert ist.

- Grundsätzlich wird ein Sammler, bei dem die Anzahl der in ihm sammelbaren Objekte eine starre Größe ist, derart instanziiert, daß er mit der für ihn festgelegten Anzahl von Objekten geeignet – z.B. mit "nil" – vorbesetzt wird.

9.4.4 Die Basis-Klasse "Array"

Um einen geordneten Sammler für beliebige Objekte einzurichten, auf dessen Objekte – über eine ganze Zahl als Index – direkt zugegriffen werden kann und in dem sich gleiche Objekte in mehrfacher Ausführung sammeln lassen, kann eine Instanz der Basis-Klasse "Array" verwendet werden. Dabei ist zu beachten, daß die Anzahl der sammelbaren Objekte *nicht variabel* ist, sondern bei der Instanziierung festgelegt werden muß. Eine Instanz der Klasse "Array" wird als *Array* bezeichnet.

Hinweis: Instanzen der Klasse "Array" lassen sich innerhalb von Keyword-Messages zur Übergabe von Argumenten einsetzen. Anstatt für jedes Argument einen Keyword-Selektor zu verwenden, ist es bei mehr als vier Argumenten üblich, sie innerhalb eines Arrays zu sammeln und den Array als Argument zu übergeben.

Um eine Instanz der Klasse “Array” einzurichten, kann die *Klassen-Methode* “new:” verwendet werden.

- **“new:”**:
  Als Ergebnis-Objekt der Keyword-Message “new:” resultiert ein *Array* mit Pseudovariablen “nil”, deren Anzahl durch diejenige ganze Zahl bestimmt ist, die als Argument innerhalb der Message “new:” aufgeführt ist.

  Hinweis: Wird einer Instanz der Klasse “Array” die Message “do:” geschickt, so ist zu beachten, daß die Pseudovariable “nil” als Objekt auftreten kann.

Zum Beispiel wird durch die Zuweisung

```
VarArray := Array new: 3
```

ein Array eingerichtet, in dem das Objekt “nil” in dreifacher Ausführung eingetragen ist.
Für die in einem Array gesammelten Objekte ist eine Reihenfolge festgelegt. Dabei korrespondiert jedes Objekt mit einem ganzzahligen Index. Es gibt ein erstes Objekt, dem die Zahl “1” als Index zugeordnet ist, ein zweites Objekt, mit dem die Zahl “2” als Index korrespondiert, usw.
Sollen z.B. die Zeichenketten ’eins’, ’zwei’ und ’drei’ in “VarArray” – in dieser Reihenfolge – gesammelt werden, so läßt sich dies wie folgt erreichen:

```
VarArray at:1 put:’eins’; at:2 put:’zwei’; at:3 put:’drei’
```

Dabei haben wir die Basis-Methode “at:put:” aus der Klasse “Object” verwendet, mit der sich Objekte in Sammlern eintragen lassen.

- **“at:put:”**:
  Das als Argument von “put:” angegebene Objekt wird an der Index-Position, die durch das Argument von “at:” bestimmt ist, in demjenigen Array gesammelt, der als Empfänger-Objekt der Keyword-Message “at:put:” aufgeführt ist. Als Ergebnis-Objekt resultiert das hinter “put:” angegebene Argument.

Alternativ zum bisherigen Vorgehen können wir die zu sammelnden Werte bereits bei der Instanziierung eines Arrays festlegen, indem wir in unserem Fall die folgende Zuweisung ausführen lassen:

```
VarArray := #(’eins’ ’zwei’ ’drei’)
```

In dieser Situation spricht man von einem *Array-Literal*, da in dem der globalen Variablen “VarArray” zugeordneten Array die drei Literale ’eins’, ’zwei’ und ’drei’ gesammelt werden.

- Grundsätzlich läßt sich ein Array durch die Angabe eines *Array-Literals* erzeugen. Dieses Literal muß durch das Symbolzeichen “#” eingeleitet werden. Diesem Symbolzeichen müssen ein oder mehrere Literale folgen, die durch die Klammer “(” eingeleitet und durch die Klammer “)” abgeschlossen werden.

Hinweis: Da auch Symbole zu den Literalen zählen, dürfen in einem *Array-Literal* auch Symbole angegeben werden. Dabei ist jedoch zu beachten, daß sie in diesem Fall nicht durch das Symbolzeichen eingeleitet werden dürfen.
Wollen wir ein Array einrichten, in dem z.B. der String 'eins', die Zahl "2" und der Array "#(3 4)" gesammelt werden sollen, so dürfen wir den Array "#(3 4)" nicht durch das Symbolzeichen kennzeichnen, sondern müssen die Anforderung

```
VarArray := #('eins' 2 (3 4))
```

stellen.

Um ein Array z.B. mit den Zahlen von "1" bis "10" zu besetzen, kann wie folgt vorgegangen werden:

```
|zahl|
zahl := 0.
VarArray := Array new: 10.
VarArray do:[:einObjekt|zahl:=zahl + 1 .
                        VarArray at: zahl put: zahl]
```

Hinweis: Bei diesem Beispiel ist zu beachten, daß der Block-Parameter "einObjekt" nicht innerhalb der Block-Anforderungen vorkommt.
Dieses Vorgehen können wir vereinfachen. Im nächsten Abschnitt stellen wir eine geeignetere Lösung vor.

Sollen die Objekte zweier Arrays aneinandergereiht werden, so kann die Basis-Methode "," (Komma) z.B. wie folgt eingesetzt werden:

```
|varArray1 varArray2|
varArray1 := #(1 2).
varArray2 := #(3 4).
VarArray := varArray1,varArray2
```

In diesem Fall ist "VarArray" ein Array mit den vier Objekten "1", "2", "3" und "4" zugeordnet.

9.4.5 Die Basis-Klasse "Interval"

In Abhängigkeit vom jeweiligen Lösungsplan ist es oftmals erforderlich, einen Block wiederholt ausführen zu lassen, wobei bei jeder Ausführung eine geeignete Zahl als Block-Parameter zur Verfügung gehalten wird.

Für derartige Fälle eignen sich Instanzen der Basis-Klasse "Interval". Die Instanzen dieser Klasse werden *Intervalle* genannt und sind geordnete Sammler von verschiedenen Zahlen, auf die gezielt zugegriffen werden kann und deren Anzahl fest vorgegeben ist.

Hinweis: Beim Einsatz von Instanzen der Klasse "Interval" ist es nicht möglich, die Zahlen – z.B. mit der Methode "at:put:" – zu ändern.

- Instanzen der Klasse "Interval" lassen sich durch den Einsatz der Basis-Methode "from:to:by:" einrichten, die als Klassen-Methode in "Interval" vereinbart ist.

Zum Beispiel erhalten wir als Ergebnis-Objekt von

```
Interval from: 1 to: 10 by: 1
```

das Intervall

```
Interval(1 2 3 4 5 6 7 8 9 10)
```

mit aufsteigend geordneten Werten.

Hinweis: Dieses Ergebnis-Objekt kann alternativ durch die Ausführung der Message "1 to: 10 by: 1" ermittelt werden. Dabei ist zu beachten, daß die Keyword-Message "to:by:" dem Empfänger-Objekt "1" geschickt wird. Dieses Empfänger-Objekt ist eine Instanz der Klasse "SmallInteger". Die korrespondierende Methode wird innerhalb der Klasse "Number" gefunden. In dieser Klasse ist die Methode "to:by:" in der Form

```
to: sNumber by: iNumber
     ∧ Interval from: self to: sNumber by: iNumber
```

als Instanz-Methode vereinbart.

Um dieselben Werte in umgekehrter Reihenfolge zu erhalten, ist die folgende Anforderung zu stellen:

```
Interval from: 10 to: 1 by: -1
```

Durch den Einsatz eines Arrays läßt sich die oben angegebene Lösung für das Beispiel, bei dem ein Array mit den Zahlen von "1" bis "10" gefüllt wurde, wie folgt vereinfachen:

```
VarArray := Array new: 10.
(Interval from:1 to:10 by:1) do: [:eineZahl|
                                    VarArray at:eineZahl put:eineZahl]
```

Soll z.B. von den Objekten dieses Arrays jedes zweite Objekt im Transcript-Fenster angezeigt werden, so läßt sich dies durch die Anforderung

```
(1 to: VarArray size by: 2) do: [:einIndex|Transcript cr;
                                 show:(VarArray at:einIndex) printString]
```

erreichen. In dieser Situation werden im Transcript-Fenster die Werte "1", "3", "5", "7" und "9" angezeigt.

9.4.6 Die Basis-Klasse "String"

Um Objekte zur Verfügung zu haben, bei denen ein oder mehrere Zeichen zu einer Einheit zusammengefaßt sind, müssen *Zeichenketten* aus der Basis-Klasse "String" instanziiert werden.

- Im Kapitel 3 haben wir kennengelernt, daß eine *Zeichenkette* aus einem oder mehreren Zeichen aufgebaut ist. Dem ersten Zeichen muß ein Hochkomma (') vorangestellt sein, und dem letzten Zeichen muß ein Hochkomma nachfolgen.

Hinweis: Falls eine Zeichenkette ein Hochkomma enthalten soll, muß dieses Zeichen in Form zweier, direkt aufeinanderfolgender Hochkommata angegeben werden.

Instanzen der Klasse “String” bestehen aus einer fest vorgebenen Anzahl von Zeichen, d.h. Objekten, die aus der Klasse “Character” instanziiert wurden.
Instanzen der Klasse “String” zählen zu den Literalen. Daher können sie implizit durch Hinschreiben eingerichtet werden. Zum Beispiel wird durch

```
VarString := 'abc'
```

der globalen Variablen “VarString” eine Zeichenkette der Länge 3 mit den Zeichen “a”, “b” und “c” zugeordnet.

Um eine Zeichenkette der Länge 3 zu instanziieren, kann die Keyword-Message “new:” z.B. wie folgt eingesetzt werden:

```
VarString := String new: 3
```

In diesem Fall ist “VarString” eine Zeichenkette aus drei Zeichen zugeordnet, deren numerischer Wert gleich dem dezimalen ASCII-Kodewert “0” entspricht.

Sollen anschließend die Zeichen “a”, “b” und “c” in die “VarString” zugeordnete Zeichenkette eingetragen werden, so ist die folgende Anforderung auszuführen:

```
VarString at: 1 put: $a; at: 2 put: $b; at: 3 put: $c
```

Für Instanzen der Klasse “String” stehen neben der Basis-Methode “=” zusätzlich die Methoden “>=”, “<=”, “<” und “>” für den zeichenweisen Vergleich an den einzelnen Zeichenketten-Positionen zur Verfügung. Dabei werden Zeichenketten in ihrer lexikographischen Ordnung – wie wir es vom Telefonbuch her kennen – auf der Basis der ganzzahligen ASCII-Kodewerte der einzelnen Zeichen verglichen. Dabei ist zu beachten, daß – mit Ausnahme der Prüfung auf Gleichheit – nicht zwischen Klein- und Großbuchstaben unterschieden wird.

Zur Reihung von Zeichenketten läßt sich die binäre Message “,” verwenden, so daß z.B. durch die Anforderung

```
'abc','def'
```

als Ergebnis-Objekt die Zeichenkette ’abcdef’ resultiert.

Enthält ein String führende oder abschließende Leerzeichen, so können wir diese Leerzeichen durch den Einsatz der Message “trimBlanks” in der Form

```
VarString := VarString trimBlanks
```

entfernen.

Um Zeichenketten, die nur aus Ziffern bestehen, in jeweils korrespondierende ganze Zahlen zu überführen, stellt das SMALLTALK-System die Message “asInteger” zur Verfügung. Weitere Umwandlungs-Methoden innerhalb der Klasse “String” sind z.B. die Basis-Methoden “asLowerCase”, “asUpperCase” und “asSymbol”.

9.4.7 Die Basis-Klasse "OrderedCollection"

Steht für einen Sammler, dessen Objekte geordnet und über einen Index direkt zugreifbar sein sollen, die maximale Anzahl der zu sammelnden Objekte nicht von vornherein fest, so ist die Instanziierung eines Arrays nicht sinnvoll. In dieser Situation bietet es sich an, eine Instanz der Basis-Klasse "OrderedCollection" einzurichten.

- Bei einer Instanz der Klasse "OrderedCollection" handelt es sich um einen geordneten Sammler mit einer variablen Anzahl von Objekten, auf die ein direkter Zugriff möglich ist und von denen gleiche Objekte mehrfach auftreten dürfen. Die Ordnung der gesammelten Objekte ist durch die *Einfügereihenfolge* bestimmt, in der die einzelnen Objekte dem Sammler hinzugefügt werden.

Um z.B. eine Instanz der Klasse "OrderedCollection" zu erzeugen, die mit den Objekten "2", "1" und "3" – in dieser Reihenfolge – gefüllt ist, können wir die folgenden Anforderungen stellen:

```
VarOrd := OrderedCollection new.
VarOrd add: 2; add: 1; add: 3
```

Grundsätzlich werden Objekte durch den Einsatz der Basis-Methode *"add:"* derart gesammelt, daß neu hinzugefügte Objekte immer hinter sämtlichen bereits im Sammler enthaltenen Objekten angefügt werden.

Hinweis: Die gesammelten Objekte selbst brauchen untereinander nicht vergleichbar zu sein.

- **"add:"**:
  In dem Sammler, der als Empfänger-Objekt der Message "add:" angegeben ist, wird das als Argument aufgeführte Objekt hinter allen bereits gesammelten Objekten angefügt.
  Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Sammler hinzugefügt wurde.

Soll als Ergebnis-Objekt ein Sammler mit seinen Objekten ermittelt werden, so können wir dazu die Messages "with:", "with:with:", "with:with:with:" oder "with:with:with:with:" einsetzen.

- **"with:", "with:with:", "with:with:with:", "with:with:with:with:"**:
  Durch den Einsatz einer dieser Klassen-Methoden läßt sich eine Instanz der Klassen "Bag", "Set", "Array", "String", "OrderedCollection" oder "SortedCollection" mit einem, zwei, drei oder vier Objekten einrichten. Dabei sind die jeweiligen Objekte als Argument des Selektors "with:" aufzuführen. Als Ergebnis-Objekt resultiert der als Empfänger-Objekt aufgeführte Sammler mit den hinzugefügten Objekten.

Hinweis: Eine mehr als vierfache Wiederholung des Selektors "with:" ist nicht erlaubt.

Somit ist die folgende Anforderung mit dem Einsatz der Message "with:with:with:" in der Form

```
VarOrd := OrderedCollection with: 2 with: 1 with: 3
```

äquivalent zu den beiden obigen Anforderungen mit den Messages "new" und "add:".

Durch den Einsatz der Basis-Methode "addFirst:" besteht die Möglichkeit, das neue Objekt *vor* allen bereits gesammelten Objekten in den Sammler einzutragen.
Sofern eine Einfügung vor (nach) dem erstmaligen Auftreten eines im Sammler enthaltenen Objektes vorgenommen werden soll, müssen wir die Methode "add:before:" ("add:after:") einsetzen.

- **"addFirst:"**:
  In dem Sammler, der als Empfänger-Objekt der Message "addFirst:" aufgeführt ist, wird das als Argument angegebene Objekt *vor allen* bereits gesammelten Objekten eingetragen.
- **"add:after:"**:
  In dem Sammler, an den die Message "add:after:" als Empfänger-Objekt gerichtet ist, wird das als Argument von "add:" aufgeführte Objekt unmittelbar *hinter* dem als Argument von "after:" angegebenen Objekt eingetragen.
- **"add:before:"**:
  In dem Sammler, an den die Message "add:before:" als Empfänger-Objekt gerichtet ist, wird das als Argument von "add:" aufgeführte Objekt unmittelbar *vor* dem als Argument von "before:" angegebenen Objekt eingetragen.

Auf der Basis der oben erzeugten Instanz "VarOrd" erhalten wir durch die Ausführung der Anforderung

```
VarOrd addFirst: 'Null'; add: 'zweieinhalb' before: 3;
       add: 'zweieinviertel' after: 2
```

einen Sammler der Form "OrderedCollection('Null' 2 'zweieinviertel' 1 'zweieinhalb' 3)".
Um gezielt auf ein bestimmtes Objekt des Sammlers zugreifen zu können, stehen die Basis-Methoden "at:", "after:" und "before:" zur Verfügung.

- **"at:"**:
  Als Ergebnis-Objekt resultiert das Objekt, das an der als Argument aufgeführten *Index-Position* in dem als Empfänger-Objekt von "at:" angegebenen Sammler enthalten ist.
- **"before:"**:
  Als Ergebnis-Objekt resultiert dasjenige Objekt, das in dem als Empfänger-Objekt von "before:" angegebenen Sammler unmittelbar *vor* dem als Argument aufgeführten Objekt enthalten ist.
- **"after:"**:
  Als Ergebnis-Objekt resultiert dasjenige Objekt, das in dem als Empfänger-Objekt von "after:" angegebenen Sammler unmittelbar *hinter* dem als Argument aufgeführten Objekt enthalten ist.

Zum Beispiel resultiert – auf der Basis der oben veränderten Instanz "VarOrd" – aus den folgenden Anforderungen jeweils die Zahl "2":

```
VarOrd at: 2.
VarOrd after: 'Null'.
VarOrd before: 'zweieinviertel'
```

Um die Index-Position eines gesammelten Objektes festzustellen, läßt sich die Basis-Methode "indexOf:" einsetzen.

- **"indexOf:"**:
  Als Ergebnis-Objekt von "indexOf:" resultiert die Index-Position, an der das als Argument aufgeführte Objekt in dem als Empfänger-Objekt von "at:" angegebenen Sammler erstmalig enthalten ist.

Zum Beispiel resultiert aus der Anforderung

```
VarOrd indexOf: 'zweieinviertel'
```

die Zahl "3" als Ergebnis-Objekt.
Durch den Einsatz der Basis-Methode "," (Komma) lassen sich Objekte reihen, die in zwei Instanzen der Klasse "OrderedCollection" gesammelt sind.
Zum Beispiel resultiert (mit "Show It") aus den Anforderungen

```
|varOrd1 varOrd2|
varOrd1 := OrderedCollection new.
varOrd2 := OrderedCollection new.
varOrd1 add: 1; add: 2.
varOrd2 add: 3; add: 4.
varOrd1 :=  varOrd1 , varOrd2.
varOrd1
```

die Anzeige von "OrderedCollection(1 2 3 4)".
Die Anzeige von

```
OrderedCollection(OrderedCollection(1 2) OrderedCollection(3 4))
```

erhalten wir dann, wenn wir die Anforderungen

```
|varOrd1 varOrd2 varOrd3|
varOrd1 := OrderedCollection new.
varOrd2 := OrderedCollection new.
varOrd3 := OrderedCollection new.
varOrd1 add: 1; add: 2.
varOrd2 add: 3; add: 4.
varOrd3 add: varOrd1; add: varOrd2.
varOrd3
```

(mit "Show It") ausführen lassen.

9.4.8 Die Basis-Klasse “SortedCollection”

Sollen die in einem Sammler enthaltenen Objekte gemäß der Sortierfolgeordnung gespeichert sein, so muß der Sammler als Instanz der Basis-Klasse “SortedCollection” eingerichtet werden.

- Bei einer Instanz der Klasse “SortedCollection” handelt es sich um einen geordneten Sammler mit einer variablen Anzahl von Objekten, die gemäß der *aufsteigenden* oder *absteigenden Sortierreihenfolge* im Sammler enthalten sind. Auf diese Objekte, von denen gleiche Objekte mehrfach auftreten dürfen, ist ein direkter Zugriff möglich.

Standardmäßig lassen sich durch den Einsatz der Basis-Methode *“add:”* Objekte, die mit der Vergleichsbedingung “<=” verglichen werden können, in aufsteigender Sortierfolgeordnung sammeln. Dies bedeutet, daß in einer Instanz von “SortedCollection” Instanzen der Klasse “Float”, “Fraction”, “Integer”, “String” und “Character” gesammelt werden können.

- **“add:”**:
  In dem Sammler, der als Empfänger-Objekt der Message “at:” aufgeführt ist, wird das als Argument angegebene Objekt so eingetragen, daß sich alle gesammelten Objekte anschließend in aufsteigender Sortierfolgeordnung befinden. Als Ergebnis-Objekt resultiert dasjenige Objekt, das dem Sammler hinzugefügt wird.

Zum Beispiel resultiert aus den Anforderungen

```
VarSort := SortedCollection new.
VarSort add: 2; add: 1; add: 3
```

ein Sammler, in dem die Zahlen “1”, “2” und “3” – in dieser Reihenfolge – enthalten sind.
Soll bei einer Instanz der Klasse “SortedCollection” eine Speicherung in *absteigender* Sortierfolgeordnung vorgegeben werden, so muß die Instanziierung unter Einsatz der Message “sortBlock:” vorgenommen und als Argument der *Sortier-Block*

```
[:x :y | x >= y]
```

wie folgt verwendet werden:

```
VarSort := SortedCollection sortBlock: [:x :y| x >= y]
```

Als Ergebnis-Objekt der Anforderung

```
VarSort add: 2; add: 1; add: 3; yourself
```

resultiert “SortedCollection(3 2 1)”.
Zum gleichen Ergebnis können wir auch durch die folgenden Anforderungen gelangen:

```
VarSort := SortedCollection new.
VarSort add: 2; add: 1; add: 3.
VarSort := VarSort sortBlock: [:x :y| x >= y]
```

Sollen die in einer Instanz "VarOrd" von "OrderedCollection" gesammelten Objekte sortiert werden, so können wir dazu die Message "asSortedCollection" einsetzen. Soll dabei nicht die voreingestellte aufsteigende Sortierfolgeordnung gelten, so kann die Keyword-Message "asSortedCollection:" mit dem Sortier-Block "[:x :y | x >= y]" – als Argument – z.B. in der folgenden Form eingesetzt werden:

```
VarSort := VarOrd asSortedCollection: [:x :y| x >= y]
```

Es lassen sich auch Instanzen der Klasse "Array" derart innerhalb einer Instanz der Klasse "SortedCollection" sammeln, daß eine absteigende Sortierung nach der Größe der gesammelten Arrays erfolgt.

Dies zeigen die Anforderungen

```
VarSort := SortedCollection sortBlock: [:x :y|x size >= y size].
VarSort add: #('eins' 'zwei'); add: #(1 2 3); yourself
```

die zum Ergebnis-Objekt "SortedCollection((1 2 3)('eins' 'zwei'))" führen.
Zum Erstellen einer "SortedCollection", in der eine aufsteigende Sortierung nach der Größe des zweiten Objekts innerhalb der gesammelten Arrays vorgenommen wird, können wir z.B. die folgenden Anforderungen stellen:

```
VarArray := Array new: 2.
VarArray at: 1 put: #(1 2 3).
VarArray at: 2 put: #(100 200 300).
VarArray asSortedCollection: [:x :y|(x at: 2) >= (y at: 2)]
```

Als Ergebnis-Objekt wird in diesem Fall

```
SortedCollection((100 200 300) (1 2 3))
```

erhalten.

9.5 Methodenübersicht

Um eine Aussage über die Funktion einer Methode zu machen, können die innerhalb einer Klasse verabredeten Instanz- und Klassen-Methoden in die folgenden Arten eingeteilt werden:

- *Konstruktor-Methoden*, die zur Instanziierung von Objekten dienen;
- *Destruktor-Methoden* zum Löschen von Objekten;
- *Verbindungs-Methoden*, mit denen Objekte ergänzt bzw. erweitert werden können;

- *Prüf-Methoden*, mit denen sich Eigenschaften von Objekten prüfen lassen;
- *Iterations-Methoden*, mit denen sich gesammelte Objekte einheitlich verarbeiten lassen, und
- *Zugriffs-Methoden*, mit denen auf Objekte bzw. deren Attributwerte zugegriffen werden kann.

Gliedern wir die wichtigsten Basis-Methoden, die sich von Instanzen aus den Unterklassen der Basis-Klasse “Collection” ausführen lassen, so können wir die folgende tabellarische Übersicht angeben:

Klasse	Konstruktor–M.	Verbindungs–Methoden	Destruktor–M.
Collection	with: with:with: with:with:with:	addAll:	remove: removeAll:
Bag	new	add: add:withOccurrences:	remove:ifAbsent:
Set	new	add:	remove:ifAbsent:
Dictionary		add: at:put:	removeAssociation: removeKey: removeKey:ifAbsent:
IndexedCollection		, atAllPut: atAll:put: copyFrom:to: copyWith: copyWithout: copyReplaceFrom:to:with: replaceFrom:to:with: replaceFrom:to:with:startingAt: replaceFrom:to:withObject:	
FixedSizeCollection	with: with:with: with:with:with:		
Array	#(element ...) new:	at:put:	
Interval	from:to: from:to:by:		
String	, , new:	at:put: replace:with: replaceFrom:to:with:startingAt: replaceFrom:to:withObject: withCrs	
OrderedCollection	new	, add: add:after: add:before: add:afterIndex: add:beforeIndex: addAllFirst: addAllLast: addFirst: addLast: at:put: replaceFrom:to:with:	remove:ifAbsent: removeIndex: removeFirst removeLast
SortedCollection	new sortBlock:	add: addAll: sortBlock:	

Abbildung 9.6: Methoden-Arten

Hinweis: Neben den bisher vorgestellten Methoden von Sammlern haben wir in dieser und der nächsten Übersicht eine Auswahl weiterer Messages aufgeführt, deren Einsatzmöglichkeit sich meist aus ihrem Namen ableiten läßt. Nähere Angaben über diese Methoden lassen sich – auf der Ebene des Window-Systems – über das Ikon "Encyclopedia of classes" abrufen oder aus den Kommentaren der jeweiligen Methodenvereinbarung entnehmen.

Klasse	Prüf-Methoden	Iterations-Methoden	Zugriffs-Methoden
Collection	includes: isCollection isEmpty notEmpty occurrencesOf:	collect: detect: detect:ifNone: inject:into: reject: select:	
Bag	includes: occurrencesOf: size	do:	elements
Set	includes: occurrencesOf: size	do:	contents
Dictionary	includes: includesKey: inspect isDictionary occurrencesOf: size	associationsDo: associationsSelect: do: keysDo: select:	associationAt: associationAt:ifAbsent: at: at:ifAbsent: keyAtValue: keyAtValue:ifAbsent: keys lookUpKey: values
IndexedCollection	= includes: checkIndex:	do: reverseDo: with:do:	copyFrom:to: first findFirst: last findLast: indexOf: indexOf:ifAbsent: indexOfCollection: reversed
FixedSizeCollection	size	collect: select:	
Array	isArray		at:
Interval	increment size		at:
String	< <= = > >= equals: isString size		at: upTo:
OrderedCollection	includes: size	do:	at: after: after:IfNone: before: before:IfNone: copyFrom:to:
SortedCollection	sortBlock		copyFrom:to: reSort sort:to:

Abbildung 9.7: Methoden-Arten

Hinweis: Bei dieser Aufstellung ist zu beachten, daß *nicht* alle innerhalb einer übergeordneten Klasse aufgeführten Methoden auch für die zugehörigen Unterklassen ausführbar sein müssen. So ist z.B. die Ausführung von "at:put:" der Klasse "OrderedCollection" für

Instanzen von “SortedCollection” ausgeschlossen. Dies wird in der Klasse “SortedCollection” innerhalb der Methode “at:put:” durch den Einsatz der Methode “invalidMessage” in der Form

```
at: anInteger put: anObject
    ∧ self invalidMessage
```

erreicht.

9.6 Indirekte Message-Zustellung

Neben der bislang vorgestellten Möglichkeit, ein Objekt durch die ihm zugestellte Message *direkt* zur Ausführung einer Methode zu veranlassen, kann eine Message auch *indirekt* zugestellt werden.

Hierzu lassen sich die beiden Basis-Methoden “perform:” und “perform:with:” verwenden.

- **“perform:”**:
  Durch den Einsatz der Message “perform:” wird dem Empfänger-Objekt diejenige unäre Message zugestellt, deren Message-Selektor – eingeleitet durch das Symbolzeichen “#” – als Argument von “perform:” aufgeführt ist.

- **“perform:with:”**:
  Durch die Keyword-Message “perform:with:” läßt sich dem Empfänger-Objekt eine binäre Message oder eine Keyword-Message mit einem Argument zustellen. Dabei ist der zugehörige Message-Selektor – eingeleitet durch das Symbolzeichen “#” – als Argument hinter dem Selektor “perform:” und das zu diesem Message-Selektor zugehörige Argument hinter dem Selektor “with:” als Argument aufzuführen.

  Hinweis: Ist es notwendig, eine Keyword-Message mit zwei oder drei Argumenten indirekt zustellen zu lassen, so können die Messages “perform:with:with:” bzw. “perform:with:with:with:” verwendet werden. Sollen mehr als drei Argumente mitgeteilt werden, so können die Argumente z.B. innerhalb einer Instanz der Klasse “Array” gesammelt und diese Instanz als Argument des Selektors “with:” innerhalb der Message “perform:with:” übermittelt werden.

Vereinbaren wir z.B. in der Klasse “WerteErfassung” eine Methode namens “aendernBag:wert:” in der Form

```
aendernBag: einSelektor wert: einString
werteBag perform: einSelektor with: einString
```

so können wir eine Anforderung der Form

```
WerteErfassung11 aendernBag: #add: wert: '37'
```

an die durch die globale Variable “WerteErfassung11” gekennzeichnete Instanz der Klasse “WerteErfassung” richten.
Durch diese Anforderung wird “WerteErfassung11” die Message “add:” mit dem Argument “37” zugestellt, so daß der Instanz-Variablen “werteBag” der Wert “37” hinzugefügt wird.

Soll der Wert “37” aus “werteBag” entfernt werden, so kann die Message “aendernBag:wert:” dem Objekt “WerteErfassung11” in der folgenden Form geschickt werden:

```
WerteErfassung11 aendernBag: #remove: wert: '37'
```

In dem angegebenen Beispiel besteht keine Notwendigkeit, die Messages *indirekt* zuzustellen. Anders ist dies bei Anwendungen, bei denen in Abhängigkeit von speziell eingestellten Rahmenbedingungen die eine oder andere Methode zur Ausführung kommen soll. In derartigen Fällen ist die vorgestellte Form der Nachrichtenübermittlung eine elegante Möglichkeit, entsprechende Problemstellungen zu lösen.

Kapitel 10

Aufbau und Einsatz von Fenstern

10.1 Der Aufbau von Fenstern

Wie wir bereits an anderen Stellen ausgeführt haben, ist der Einsatz des SMALLTALK-Systems prädestiniert für die Umsetzung von Lösungsplänen, bei denen Fenster für die Mitteilung von Anforderungen eingesetzt werden sollen.

- Grundsätzlich besteht jedes Fenster, das vom SMALLTALK-System zur Kommunikation mit dem Anwender eingesetzt bzw. vom Anwender zur Lösung einer Problemstellung eingerichtet wird, aus einem *Rahmenfenster*, in dem geeignete *Fenster-Bausteine* integriert sind.

  Hinweis: Durch den Aufruf des SMALLTALK-Systems wird das Transcript-Fenster als erstes Fenster (nach dem Begrüßungs-Fenster) am Bildschirm angezeigt. Es steuert die Ausführung des SMALLTALK-Systems und stellt in der System-Schaltfeld-Zeile die Menü-Option "Exit Smalltalk/V..." zur Verfügung, mit der sich die Ausführung des SMALLTALK-Systems beenden läßt.
  Jedes im Dialog verwendete Fenster wird unmittelbar unterhalb des Transcript-Fensters in die vom Windows-System verwaltete Hierarchie aller am Bildschirm angezeigten Fenster integriert.

Das von uns zur Lösung der Problemstellung PROB-1-1 entwickelte Fenster war von uns so konzipiert, daß ein Textfeld, ein Eingabefeld sowie zwei Schaltfelder als Fenster-Bausteine in ein Rahmenfenster integriert wurden.

Als Beispiel für ein Fenster, das vom SMALLTALK-System zum Dialog bereitgestellt wird, haben wir das Klassen-Hierarchie-Browser-Fenster kennengelernt (siehe Abschnitt 2.3). Bei diesem Fenster enthält das Rahmenfenster die folgenden Fenster-Bausteine:

- jeweils ein *Listenfeld* im "Klassen-Bereich", "Variablen-Bereich" und "Methoden-Bereich",
- zwei *Optionsfelder* im "Instanzen-/ Klassen-Bereich" und
- ein *Editierfeld* im "Editier-Bereich".

Das Rahmenfenster des Klassen-Hierarchie-Browsers ist in eine System-Schaltfeld-Zeile und eine Menü-Leiste mit den Menüs "File", "Edit", "Smalltalk", "Classes",

"Variables" und "Methods" gegliedert. Neben der Titel-Zeile "Smalltalk Express: Class Hierarchy Browser" enthält die System-Schaltfeld-Zeile das System-Menü sowie die System-Schaltfelder zur "Minimierung", "Maximierung" und zum "Schliessen".

Hinweis: Neben dem Klassen-Hierarchie-Browser-Fenster haben wir mit dem Transcript-Fenster und dem Workspace-Fenster, in deren jeweiligen Rahmenfenstern jeweils ein Editierfeld als Fenster-Baustein integriert ist, zwei weitere SMALLTALK-spezifische Fenster kennengelernt.
Zu weiteren system-spezifischen Fenstern zählen z.B. das Debug-Fenster und das Inspect-Fenster (siehe im Anhang unter A.3 und A.4).

Während die system-seitig eingesetzten Fenster unmittelbar beim Start des SMALLTALK-Systems bzw. über die Anwahl einer geeigneten Menü-Option – wie z.B. durch "New Workspace" oder "Browse Classes" des Menüs "File" – automatisch eingerichtet und angezeigt werden, sind die zur Lösung einer Problemstellung benötigten Fenster "von Hand" aufzubauen und zur Anzeige zu bringen.

Um das von uns zur Lösung des Problems PROB-1 konzipierte Erfassungsfenster zu erstellen, haben wir das Werkzeug "WindowBuilder Pro/V" eingesetzt (siehe Kapitel 2). Dieses Vorgehen hatte den Vorteil, daß wir die Vorzüge der dialogorientierten Entwicklung eines Fensters kennenlernen konnten. Darüberhinaus ließ sich die angestrebte Programmierung zügig vornehmen, da wir keine Detailkenntnisse benötigten, die zur Einrichtung von Fenstern erforderlich sind.

Um Methoden-Vereinbarungen, die vom Werkzeug "WindowBuilder Pro/V" für die Einrichtung eines Fensters *automatisch* erzeugt werden, verstehen und zukünftig auch eigenständig Anforderungen zum Aufbau von Fenstern angeben zu können, werden wir im folgenden beschreiben, wie sich Fenster einrichten und bearbeiten lassen.

Grundsätzlich gilt:

- Der Aufbau von Fenstern ist durch Anforderungen mit Hilfe von Methoden der Basis-Klasse "Window" und deren Unterklassen bestimmbar.
- Beim Einsatz der grafischen Benutzeroberfläche basiert die Steuerung der Kommunikation und die Fenster-Verwaltung auf den Methoden der Basis-Klasse "ViewManager" und deren Unterklassen.

Unter Berücksichtigung dieser Rahmenbedingungen muß daher die Gestaltung von Fenstern durch die Ausführung von Methoden festgelegt werden, die Bestandteil der Klasse "Window" und deren Unterklassen sind.

Das Verhalten von Fenstern – im Hinblick auf die durchzuführende Kommunikation – ist durch geeignete Methoden zu bestimmen, die auf Basis-Methoden zurückgreifen, die in der Basis-Klasse "ViewManager" und deren Unterklassen vereinbart sind.

10.2 Klassen zum Aufbau von Fenstern

Im Hinblick auf die Kommunikation, die sich auf der grafischen Benutzeroberfläche gründet, ist der folgende Ausschnitt der Klassen-Hierarchie von Bedeutung:

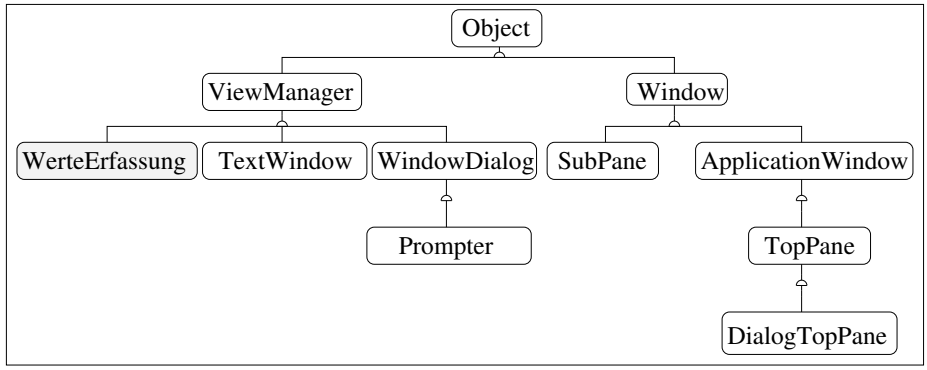


Abbildung 10.1: Ausschnitt aus der Klassen-Hierarchie

Hinweis: Bei den aufgeführten Klassen "ViewManager", "Window", "ApplicationWindow" und "SubPane" handelt es sich um abstrakte Klassen.

Der Aufbau von *Rahmenfenstern*, die *individuell* gestaltet werden können, basiert auf den Klassen "TopPane" und "DialogTopPane", die beide Unterklassen der abstrakten Klasse "ApplicationWindow" sind.

- Durch die Instanziierung der Klasse "TopPane" läßt sich ein Rahmenfenster einrichten, das verschiebbar und in seiner Ausdehnung durch den Einsatz der Maus veränderbar ist.
- Soll ein Rahmenfenster zwar verschiebbar, aber in seiner Ausdehnung durch den Einsatz der Maus *nicht* veränderbar sein, so ist eine Instanziierung aus der Klasse "DialogTopPane" vorzunehmen.
  Ein derartiges Rahmenfenster zeigt zudem ein *modales* Verhalten, d.h. dasjenige Fenster, aus dem heraus die Anzeige dieses Rahmenfensters angefordert wird, ist solange blockiert, bis das Rahmenfenster wieder vom Bildschirm entfernt ist.

Welche Funktion ein Fenster besitzen soll, läßt sich durch die jeweilige Instanziierung bestimmen, die sich auf die Basis-Klassen "ViewManager", "TextWindow", "WindowDialog" und "Prompter" gründet.

- Um *Text-Fenster* einzurichten, in denen Texte editiert werden können, sind Instanziierungen der Klasse "TextWindow" vorzunehmen.

  Hinweis: Als Beispiele für Text-Fenster haben wir das Transcript- und das Workspace-Fenster kennengelernt, die beide system-seitig zum Dialog mit dem SMALLTALK-System bereitgestellt werden.

Sie bestehen aus einem Rahmenfenster, das durch eine Instanziierung der Klasse "TopPane" (mit einem Fenster-Baustein in Form einer Instanz der Klasse "TextPane") bestimmt ist.

- Um ein *Dialog-Fenster* festzulegen, das ein *modales* Verhalten zeigen soll und dessen Aufbau *individuell* gestaltet werden kann, ist eine Instanziierung der Klasse "WindowDialog" (oder deren Unterklassen) durchzuführen.
  Durch Instanziierungen der Klasse "Prompter" können *Prompter* als *spezielle* Dialog-Fenster eingerichtet werden, bei deren Bildschirmanzeige eine Text-Eingabe angefordert wird.

  Hinweis: Die Rahmenfenster von Dialog-Fenstern werden durch Instanziierungen der Klasse "DialogTopPane" festgelegt.

- Um Fenster festzulegen, die *kein modales* Verhalten zeigen und deren Aufbau *individuell* gestaltet werden kann, sind Instanziierungen der Klasse "ViewManager" (oder deren Unterklassen) durchzuführen.

  Hinweis: Derartige Fenster bestehen aus einem Rahmenfenster, das durch eine Instanziierung der Klasse "TopPane" bestimmt wird.

Im folgenden stellen wir die allgemeinste Form der Einrichtung von Fenstern vor, d.h. den Aufbau von Fenstern, die unterhalb der Klasse "ViewManager" vereinbart werden.

10.3 Einrichtung und Initialisierung von Views

Um ein Fenster zu entwickeln, über das im Dialog mit einer Anwendung kommuniziert werden kann, müssen zunächst Angaben über den Aufbau des Fensters zu einem Objekt zusammengefaßt werden, das *View* (Sicht) genannt wird. Da in einem View die virtuelle Form eines am Bildschirm anzuzeigenden Fensters festgelegt wird, läßt sich ein Fenster als physikalische Gestalt eines Views ansehen.

Erst dann, wenn ein View in der durch die Problemstellung bestimmten Form eingerichtet wurde, kann es als Fenster am Bildschirm angezeigt und zur Kommunikation verwendet werden.

Die Anzeige des Fensters wird *Eröffnung* des Views genannt. Vom *Schließen* des Views wird dann gesprochen, wenn das zugehörige Fenster vom Bildschirm entfernt wird.

Für jedes *View* muß ein *Rahmenfenster* festgelegt werden, in das ein oder mehrere Fenster-Bausteine integriert werden können.

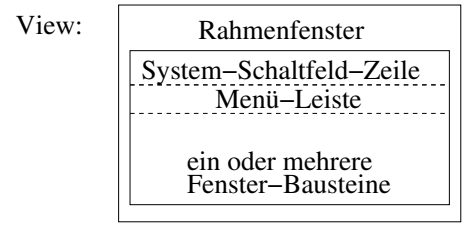


Abbildung 10.2: Strukturierung eines Views

Ein Rahmenfenster wird stets durch die System-Schaltfeld-Zeile eingeleitet, die aus dem System-Menü, der Titel-Zeile und den System-Schaltfeldern zur "Minimierung", "Maximierung" und zum "Schließen" besteht. Dieser Zeile folgt standardmäßig eine Menü-Leiste mit dem Menü "File", die sich den anwendungs-spezifischen Anforderungen entsprechend abändern läßt.

- Zur *Einrichtung* eines Views muß eine Instanziierung einer Unterklasse der Basis-Klasse "ViewManager" erfolgen. Dadurch ist gesichert, daß die zur Verwaltung eines Views benötigte Instanz-Variable "views" der Klasse "ViewManager" geerbt wird und diejenigen Methoden zur Verfügung stehen, die innerhalb der Klasse "ViewManager" zur Fenster-Verwaltung vereinbart sind.

Beim Einsatz des Werkzeugs "WindowBuilder Pro/V" haben wir zur Lösung von PROB-1-1 die Klasse "WerteErfassung" als direkte Unterklasse von "ViewManager" eingerichtet. Auf dieser Basis erhalten wir z.B. durch

```
WerteErfassung11 := WerteErfassung new
```

eine zum Aufbau eines Views benötigte Instanz in Form von "WerteErfassung11".
Um ein Rahmenfenster festzulegen, kann eine Instanziierung der Basis-Klasse "TopPane" oder der Basis-Klasse "DialogTopPane" vorgenommen werden.

- Da wir *kein* Dialog-Fenster erzeugen wollen, muß das Rahmenfenster aus der Klasse "TopPane" instanziiert werden.

  Hinweis: Um ein Dialog-Fenster einzurichten, ist das Rahmenfenster durch eine Instanziierung aus der Klasse "DialogTopPane" festzulegen.

Die angegebene Forderung läßt sich erfüllen, indem einer Instanz von "ViewManager" bzw. einer ihrer Unterklassen – wie z.B. der Instanz "WerteErfassung11" – die Message "topPaneClass" wie folgt zugestellt wird:

```
WerteErfassung11 topPaneClass
```

Hierdurch kommt die Basis-Methode "topPaneClass" zur Ausführung, die innerhalb der Klasse "ViewManager" vereinbart ist.

- **"topPaneClass"**:
  Wird die Methode "topPaneClass" der Klasse "ViewManager" ausgeführt, so resultiert die Klasse "TopPane" als Ergebnis-Objekt dieser Message.

> **Hinweis:** Ist das Empfänger-Objekt der Message "topPaneClass" aus "Window-Dialog" oder einer ihr untergeordneten Klasse instanziiert, so resultiert die Klasse "DialogTopPane" als Ergebnis-Objekt der Message "topPaneClass".

Wird der Klasse "TopPane" durch die Anforderung

```
WerteErfassung11 topPaneClass new
```

die Message "new" zugestellt, so erfolgt die Einrichtung des Views als Instanz der Klasse "TopPane".

- Bevor das View auf dem Bildschirm als Fenster angezeigt werden kann, sind geeignete Vorbereitungen zu treffen. In dieser Hinsicht ist zunächst ein *Eigentümer* für das View festzulegen.

Dazu ist die Basis-Methode "owner:" zur Ausführung zu bringen.

- **"owner:"**:
  Dem View, das als Empfänger-Objekt der Message "owner:" aufgeführt ist, wird die als Argument von "owner:" angegebene Instanz als *Eigentümer* zugeordnet.

Soll "WerteErfassung11" zum Eigentümer eines Views bestimmt werden, so ist insgesamt die folgende Anforderung zu stellen:

```
WerteErfassung11 topPaneClass new owner: WerteErfassung11
```

Der Sachverhalt, daß ein View einen *Eigentümer* besitzen muß, beruht auf dem folgenden Hintergrund:

- Fenster stellen bei der Umsetzung von Lösungsplänen den Rahmen dar, in dem der Anwender seine Anforderungen formulieren kann. Dies geschieht dadurch, daß *Ereignisse* über einen Mausklick oder eine geeignete Tastatureingabe ausgelöst werden (ausführliche Angaben hierzu erfolgen im Kapitel 11).

  Im Hinblick auf den jeweiligen Lösungsplan müssen daher Vorkehrungen getroffen werden, so daß auf die Auslösung eines Ereignisses geeignet reagiert werden kann. Dazu ist die jeweilige Klasse festzulegen, in der die Methoden gesucht werden, deren Ausführung mit dem Auftreten von bestimmten Ereignissen verknüpft sind.
  Dies wird dadurch erreicht, daß als *Eigentümer* eine Instanz aus der jeweiligen Klasse angegeben wird.

Bevor das View eröffnet und damit am Bildschirm als Fenster angezeigt und bearbeitet werden kann, muß eine *Initialisierung* durchgeführt werden. Diese erfolgt dadurch, daß das View – durch die Basis-Methode "addView:" – wie folgt an seinen Eigentümer gekoppelt wird:

```
WerteErfassung11 addView:
  (WerteErfassung11 topPaneClass new owner: WerteErfassung11)
```

Nach dieser Initialisierung enthält die Instanz-Variable “views” der Instanz “WerteErfassung11” einen Verweis auf dasjenige View, das aus der Klasse “TopPane” instanziiert wurde und dessen Eigentümer die Instanz “WerteErfassung11” ist.

Hinweis: Die Instanz-Variable “views” ist eine Instanz der Klasse “OrderedCollection”. Der ursprüngliche Inhalt dieses Sammlers hat sich für die Instanz “WerteErfassung11” von “OrderedCollection()” in “OrderedCollection(a TopPane)” geändert.

- **“addView:”**:
  Zur *Initialisierung* eines Views muß das Empfänger-Objekt der Message “addView:” eine Instanz der Klasse “ViewManager” oder einer ihr untergeordneten Klasse sein.
  Durch die Ausführung von “addView:” erhält die Instanz-Variable “views” des Empfänger-Objektes einen Verweis auf dasjenige View, das als Argument der Message “addView:” aufgeführt ist (zu näheren Angaben siehe Abschnitt 10.7).

10.4 Eröffnen und Schließen von Views

Das View, auf das durch die Instanz-Variable “views” der Instanz “WerteErfassung11” verwiesen wird, läßt sich durch die Basis-Methode “openWindow” in der Form

```
WerteErfassung11 openWindow
```

eröffnen und damit am Bildschirm als Fenster anzeigen.

Hinweis: Ist für das View kein Eigentümer explizit festgelegt, so ist die Pseudovariable “nil” – eine Instanz der Klasse “UndefinedObject” – Eigentümer des Views. Ist dies der Fall, so erhalten wir beim Eröffnen des Views die Fehleranzeige “"maxWindowSize" not understood” im angezeigten Walkback-Fenster (siehe Anhang A.3). Dabei handelt es sich bei “maxWindowSize” um eine Basis-Methode, die beim Eröffnen eines Views *automatisch* ausgeführt und in der Klasse des Eigentümers, d.h. in diesem Fall in der Klasse “UndefinedObject”, gesucht wird. Da diese Methode jedoch in der Klasse “ViewManager” vereinbart ist, kann sie nicht gefunden werden.

- **“openWindow”**:
  Das View, auf das durch die Instanz-Variable “views” derjenigen Instanz verwiesen wird, die als Empfänger-Objekt von “openWindow” aufgeführt ist, wird eröffnet und am Bildschirm als *Fenster* angezeigt (zu näheren Angaben siehe Abschnitt 10.7).

Standardmäßig erscheint auf dem Bildschirm ein Rahmenfenster mit einer voreingestellten Größe und Bildschirmposition. Dieses Verhalten läßt sich dadurch beeinflussen, daß in der Klasse, aus der der *Eigentümer* des mit dem Rahmenfenster korrespondierenden Views instanziiert ist, eine Instanz-Methode namens “initWindowSize” vereinbart wird.

Hinweis: Zum Beispiel wird durch die Vereinbarung

```
initWindowSize
∧ Display width @ Display height
```

– unter Einsatz der globalen Variablen “Display” als einer Instanz der Basis-Klasse “Screen” – festgelegt, daß das Rahmenfenster den gesamten Bildschirm ausfüllen soll. Wollen wir sowohl die Position als auch die Ausdehnung des Rahmenfensters vorgeben, so können wir die Methode “initWindowSize” z.B. durch

```
initWindowSize
∧ 100 @ 100 rightBottom: 400 @ 400
```

vereinbaren (zu näheren Angaben siehe Abschnitt 10.8).

Da das von uns eingerichtete View bislang nur aus einem Rahmenfenster besteht, werden nur die System-Menü-Schaltfelder, die aus Leerzeichen bestehende Titel-Zeile und das Menü “File” angezeigt.

Um das eröffnete View zu schließen und das korrespondierende Fenster vom Bildschirm zu entfernen, kann die Menü-Option “Schließen” des System-Menüs oder die Tastenkombination “Alt + F4” betätigt werden. Alternativ kann das “Schließen” auch durch den Einsatz der Basis-Methode “close” mit der Anforderung

```
WerteErfassung11 close
```

abgerufen werden.

- **“close”**:
  Durch die Ausführung der Basis-Methode “close” wird das View, auf das durch die Instanz-Variable “views” des Empfänger-Objektes von “close” verwiesen wird, geschlossen und das mit dem View korrespondierende Fenster vom Bildschirm entfernt (zu näheren Angaben siehe Abschnitt 10.7).

Der Einsatz der Methoden “openWindow” und “close” zur Anzeige und zur Entfernung eines Fensters ist nicht erforderlich, sofern ein *Prompter* als Dialogfenster verwendet werden soll. Zum Einsatz eines *Prompters* läßt sich die Klassen-Methode “prompt:default:” aus der Klasse “Prompter” einsetzen, indem “Prompter” als Empfänger-Objekt aufgeführt wird.

So kann z.B. durch die Anforderungen

```
|eingabeString|
eingabeString := Prompter prompt:'Gib Eingabe:' default:' '.
WerteErfassung11 perform: eingabeString asSymbol
```

ein Dialog mit einem *Prompter* geführt werden, in dem der Text “Gib Eingabe:”, ein Eingabefeld und die beiden Schaltfelder “OK” und “Cancel” angezeigt werden. Nachdem auf das Schaltfeld “OK” geklickt ist, steht die zuvor über die Tastatur eingegebene Zeichenkette in der temporären Variablen “eingabeString” zur weiteren Bearbeitung zur Verfügung. Geben wir in das Eingabefeld z.B. “initialisieren-Erfassung” ein, so wird vom Empfänger-Objekt “WerteErfassung11” die Methode “initialisierenErfassung” ausgeführt.

Hinweis: Hinzuweisen ist auch auf die Basis-Klasse “FindReplaceDialog”, durch deren Instanziierung ein Dialogfeld für Such- und Ersetzungsprozesse mittels der Basis-Methode “open:replace:” eingerichtet werden kann.

10.5 Methoden zur Gestaltung von Views

Bislang haben wir zur Einrichtung eines Views die – unabdingbar – erforderlichen Messages "topPaneClass" und "owner:" kennengelernt. Um zusätzliche Eigenschaften für ein View festlegen zu können, stellen wir im folgenden weitere Methoden vor, mit denen sich geeignete Verabredungen treffen lassen.

Zur Festlegung der System-Schaltfeld-Zeile können wir die Basis-Methode "pStyle:" z.B. mit dem Array-Literal "#(sysmenu titlebar minimize maximize sizable)" in der Form

```
pStyle: #(sysmenu titlebar minimize maximize sizable)
```

einsetzen.

- **"pStyle:"**:
  Durch die Message "pStyle:" lassen sich für das als Empfänger-Objekt aufgeführte View die voreingestellten Eigenschaften der System-Schaltfeld-Zeile ändern, indem ein geeignetes Array-Literal als Argument aufgeführt wird.

Ferner kann es wünschenswert sein, die standardmäßig eingestellte Größe und Position des Rahmenfensters zu ändern. Sofern die linke obere Ecke des Rahmenfensters z.B. in der Bildschirmmitte angezeigt und das Rahmenfenster ein Viertel der Bildschirmfläche einnehmen soll, kann die Basis-Methode "framingRatio:" durch die folgende Message zur Ausführung gebracht werden:

```
framingRatio:((Rectangle leftTopUnit
   rightAndDown:1/2 @ (1/2)) extentFromLeftTop:1/2 @ (1/2))
```

- **"framingRatio:"**:
  Für das als Empfänger-Objekt der Message "framingRatio:" aufgeführte View wird festgelegt, welche Größe und Position das zugehörige Rahmenfenster bei der Bildschirmanzeige einnehmen soll.
  Dabei führt die Auswertung der geklammerten Anforderung hinter "framingRatio:" zu einem Rechteck, das durch die Angabe des linken oberen Eckpunktes sowie des rechten unteren Eckpunktes bestimmt wird.

  Hinweis: Dabei wird die Methode "framingRatio:" der Basis-Klasse "TopPane" ausgeführt.
  Die geklammerte Anforderung kennzeichnet eine Instanz aus der Basis-Klasse "Rectangle", mit der sich ein Rechteck beschreiben läßt (zu näheren Angaben siehe Abschnitt 10.8).

Um – beim Einsatz mehrerer Views – ein bestimmtes View identifizieren zu können, ist es empfehlenswert, die einzelnen Views durch den Einsatz der Basis-Methode "viewName:" zu benennen – z.B. in der Form:

```
viewName: 'erfassungsfenster'
```

Hinweis: Sofern wir keinen Namen vergeben, ist standardmäßig der Name "mainView" festgelegt.

- **"viewName:"**:
  Durch die Ausführung der Message "viewName:" wird für das als Empfänger-Objekt aufgeführte View ein Name vergeben, der durch die als Argument angegebene Zeichenkette festgelegt ist.

  Hinweis: Wie einzelne Views über ihren Namen identifiziert werden können, stellen wir im Abschnitt 10.7 dar.

Vervollständigen wir die oben angegebene Vereinbarung eines Views, auf das durch die Instanz "WerteErfassung11" verwiesen wird, durch die bislang ergänzend angegebenen Messages, so können wir das View insgesamt wie folgt festlegen:

```
WerteErfassung11 addView:
 ( WerteErfassung11 topPaneClass new
   owner:WerteErfassung11;
   pStyle: #(sysmenu titlebar minimize maximize sizable);
   framingRatio:((Rectangle leftTopUnit
      rightAndDown:1/2@(1/2)) extentFromLeftTop:1/2@(1/2));
   viewName: 'erfassungsfenster' )
```

Als weitere Methoden, mit denen die Darstellung des Rahmenfensters beeinflußt werden kann, lassen sich z.B. einsetzen:

- **"labelWithoutPrefix:"**:
  Für das View, das als Empfänger-Objekt der Message "labelWithoutPrefix:" aufgeführt ist, wird der Inhalt der Titel-Zeile des zugehörigen Rahmenfensters durch die Zeichenkette bestimmt, die als Argument dieser Message angegeben ist.

- **"noSmalltalkMenuBar"**:
  Für das View, das als Empfänger-Objekt der Message "noSmalltalkMenuBar" aufgeführt ist, wird festgelegt, daß das zugehörige Rahmenfenster als anwendungs-spezifische Menü-Leiste *nicht* das Menü "File" (aus dem Transcript-Fenster) erhält.

Zusammenfassung

Sofern mit "<aKlasse>" eine Instanz einer Klasse namens "Klasse", die der Basis-Klasse "ViewManager" untergeordnet ist, und mit "<aTopPane>" eine Instanz der Klasse "TopPane" gekennzeichnet wird, lassen sich die oben aufgeführten Angaben zum Aufbau eines Views wie folgt zusammenfassen:

- Erzeugen einer Instanz eines Rahmenfensters:
  `<aKlasse> topPaneClass new`
- Festlegung des Eigentümers des Rahmenfensters:
  `<aTopPane> owner: <aKlasse>`

- Festlegung der Position und Größe des Rahmenfensters:
  `<aTopPane> framingRatio:`
  `(<angaben-zur-plazierung-des-rahmenfensters>)`
- Benennung des Rahmenfensters:
  `<aTopPane> viewName: 'name-des-rahmenfensters'`
- Bestimmung von rahmenfenster-spezifischen Eigenschaften:
  `<aTopPane> pStyle:#(sysmenu sizable titlebar minimize maximize)`
  `<aTopPane> labelWithoutPrefix: ' '`
  `<aTopPane> noSmalltalkMenuBar`

Damit ein View eröffnet und am Bildschirm als Fenster angezeigt werden kann, muß es zuvor durch

- `<aKlasse> addView: <aTopPane>`

initialisiert werden.
Die oben erläuterten Schritte, die zur Vereinbarung, Initialisierung, Eröffnung und zum Schließen eines Views erforderlich sind, gibt die folgende Darstellung zusammenfassend wider:

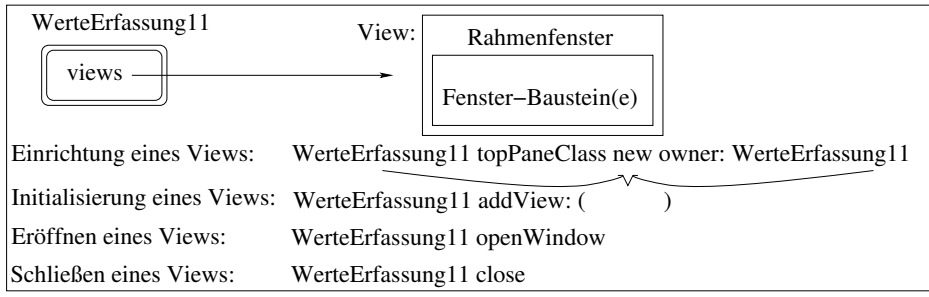


Abbildung 10.4: Aufbau, Eröffnen und Schließen eines Views

10.6 Einbeziehung von Fenster-Bausteinen

Vereinbarung von Fenster-Bausteinen

Im Hinblick auf den jeweiligen Lösungsplan besteht ein View nicht allein aus einem Rahmenfenster, sondern ist – der Anwendung entsprechend – aus geeigneten Fenster-Bausteinen aufgebaut.

- Grundsätzlich werden Fenster-Bausteine mit dem Einsatz der Basis-Methode "new" aus Unterklassen der Basis-Klasse "SubPane" instanziiert und einem View mit Hilfe der Basis-Methode "addSubpane:" hinzugefügt.

Als Fenster-Bausteine, die sich in einem Rahmenfenster festlegen lassen, können z.B. die folgenden *Objekte* verwendet werden:

- Textfelder, Eingabefelder, Schaltfelder, Kontrollfelder, Optionsfelder, Listenfelder und Kombinationsfelder.

Zum Beispiel läßt sich ein Eingabefeld wie folgt als Instanziierung der Basis-Klasse "EntryField" – einer Unterklasse von "SubPane" – vereinbaren:

```
EntryField new
  framingRatio:( (Rectangle leftTopUnit
     rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) )
```

Hinweis: Hierbei ist zu beachten, daß noch keine Vorkehrung getroffen wurde, um auf die Zeichenkette, die über die Tastatur in das Eingabefeld übertragen wird, zugreifen zu können (siehe unten).

Genau wie bei einem Rahmenfenster kann die Plazierung der Fenster-Bausteine durch den Einsatz der Basis-Methode "framingRatio:" festgelegt werden.

- **"framingRatio:"**:
  Für den als Empfänger-Objekt der Message "framingRatio:" aufgeführten Fenster-Baustein wird festgelegt, welche Größe und welche rahmenfensterspezifische Position er bei der Bildschirmanzeige einnehmen soll.
  Die Größe und Position wird durch ein Rechteck bestimmt, das durch die Angabe des linken oberen Eckpunktes sowie des rechten unteren Eckpunktes als Argument der Message "framingRatio:" aufgeführt wird.

  Hinweis: Dabei wird die Methode "framingRatio:" der Basis-Klasse "SubPane" ausgeführt.
  Werden durch "framingRatio:" innerhalb eines Rahmenfensters mehrere Fenster-Bausteine festgelegt, so ist darauf zu achten, daß es zu keinen Überschneidungen kommt.

Um einen Fenster-Baustein – wie z.B. das oben vereinbarte Eingabefeld – in ein View einzubeziehen, ist die Basis-Methode "addSubpane:" in der Form

- `<aTopPane> addSubpane: <aPane>`

einzusetzen.

- **"addSubpane:"**:
  Durch die Message "addSubpane:" wird dem View, das als Empfänger-Objekt dieser Message aufgeführt ist, die als Argument angegebene Instanz eines Fenster-Bausteins hinzugefügt.

Durch den Einsatz der Methode "addSubpane:" kann ein View jederzeit, d.h. auch wenn es bereits als Empfänger-Objekt von "addView:" aufgeführt war, nachträglich durch einen Fenster-Baustein ergänzt werden.

Zum Beispiel kann das oben angegebene View, auf das durch die Instanz "WerteErfassung11" verwiesen wird, wie folgt durch das oben spezifizierte Eingabefeld

ergänzt werden:

```
WerteErfassung11 addSubpane:
  (EntryField new
     framingRatio:( (Rectangle leftTopUnit
        rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) )
  )
```

Hinweis: Sofern das zugehörige Fenster am Bildschirm bereits angezeigt wird, ist diese Ergänzung nicht unmittelbar wirksam.
Entfernen wir jedoch dieses Fenster vom Bildschirm und stellen anschließend die Anforderung

`WerteErfassung11 openWindow`

so wird das Fenster – zusammen mit dem Eingabefeld – angezeigt.

Um ein View festzulegen, das aus einem Rahmenfenster mit Fenster-Bausteinen besteht, sind insgesamt die folgenden Angaben zu machen:

```
addView: ( <aKlasse> topPaneClass new
   framingRatio: ( <aRectangle> );
   owner: <aKlasse>;
   ... ;
   addSubpane: ( <aPane> new
         framingRatio: ( <aRectangle> );
      ...      );
   addSubpane: ( <aPane> new
         framingRatio: ( <aRectangle> );
      ...      )
         )
```

Hinweis: Dabei ist zu beachten, daß sämtliche Angaben, die
`<aKlasse> topPaneClass new`
folgen, als Empfänger-Objekt jeweils "<aTopPane>" haben.
Hinter der Message "owner:" lassen sich z.B. die folgenden Messages einsetzen:

- `pStyle: #(sysmenu sizable titlebar minimize maximize)`
- `viewName: 'erfassungsfenster'`
- `labelWithoutPrefix: ' '`
- `noSmalltalkMenuBar`

Benennung von Fenster-Bausteinen

Damit einem Fenster-Baustein – im Hinblick auf den jeweiligen Lösungsplan – eine Message zugestellt werden kann, ist eine Zeichenkette als Name für diesen Baustein festzulegen. Dies läßt sich unter Einsatz der Basis-Methode "paneName:" erreichen. Anschließend kann diesem Fenster-Baustein eine Message unter Einsatz der Basis-Methode "paneNamed:" zugestellt werden.

- **“paneName:”**:
  Durch die Keyword-Message “paneName:” wird dem Empfänger-Objekt, bei dem es sich um einen Fenster-Baustein handeln muß, derjenige Name zugeordnet, der als Argument der Message in Form einer Zeichenkette aufgeführt ist.

- **“paneNamed:”**:
  Als Ergebnis-Objekt der Message “paneNamed:” wird derjenige Fenster-Baustein ermittelt, der den als Argument aufgeführten Namen besitzt. Hierbei wird dasjenige View zugrundegelegt, das als Empfänger-Objekt der Message “paneNamed:” aufgeführt ist.

Sofern z.B. ein Eingabefeld innerhalb eines Rahmenfensters festgelegt und diesem Eingabefeld durch die Message

```
paneName: 'eingabeFeld'
```

ein Name zugeordnet ist, kann anschließend durch die Anforderung

```
(WerteErfassung11 paneNamed: 'eingabeFeld') contents
```

auf den Inhalt des Eingabefeldes zugegriffen werden, sofern das Fenster durch die Instanz “WerteErfassung11” gekennzeichnet wird.
Um diesen Zugriff zu ermöglichen, muß die oben angegebene Form der Vereinbarung des Eingabefeldes somit wie folgt ergänzt werden:

```
WerteErfassung11 addSubpane:
   (EntryField new
      framingRatio:( (Rectangle leftTopUnit
        rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) );
      paneName: 'eingabeFeld'
   )
```

Hinweis: In dieser Weise haben wir bei der Lösung von PROB-1-1 auf die jeweils im Eingabefeld erfaßten Werte zugegriffen.

10.7 Einrichtung und Einsatz mehrerer Views

Durch die vorausgehenden Erläuterungen haben wir beschrieben, wie sich ein View, das aus einem Rahmenfenster und einem Fenster-Baustein in Form eines Eingabefeldes aufgebaut ist, initialisieren, eröffnen und schließen läßt.

Zusätzlich haben wir kennengelernt, daß bei der Ausführung der Methode “addView:” ein Verweis auf das – unter Einsatz der Message “viewName:” – durch den Namen “erfassungsfenster” gekennzeichnete View in der Instanz-Variablen “views” der Instanz “WerteErfassung11” gesammelt wurde.

Um zu zeigen, wie wir mit mehreren Views arbeiten können, deren zugehörige Verweise im Sammler “views” enthalten sind, betrachten wir ein Beispiel, bei dem zwei Views mit jeweils einem Eingabefeld erzeugt werden.

Stellen wir – auf der Basis der Klasse "WerteErfassung" – die Anforderung

```
WerteErfassung1112 := WerteErfassung new
```

so können wir unter Einsatz der Methode "addView:" ein oder mehrere Verweise auf Views in der Instanz-Variablen "views" von "WerteErfassung1112" sammeln. Durch die Anforderung

```
WerteErfassung1112 addView:
 ( WerteErfassung1112 topPaneClass new
   owner:WerteErfassung1112;
   pStyle: #(sysmenu titlebar minimize maximize sizable);
   framingRatio:( (Rectangle leftTopUnit
      rightAndDown:1/2@(1/2)) extentFromLeftTop:1/2@(1/2) );
   viewName: 'erfassungsfenster1';
   noSmalltalkMenuBar;
   labelWithoutPrefix: 'Jahrgangsstufe 11';
   addSubpane:
     (EntryField new owner: WerteErfassung1112;
      framingRatio:( (Rectangle leftTopUnit
         rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) );
     paneName: 'eingabeFeld1')
  )
```

wird der Verweis auf das View "erfassungsfenster1" in die Instanz-Variable "views" eingetragen. Diese Instanz-Variable wird durch einen Verweis auf das View "erfassungsfenster2" ergänzt, indem die folgende Anforderung ausgeführt wird:

```
WerteErfassung1112 addView:
 ( WerteErfassung1112 topPaneClass new
   owner:WerteErfassung1112;
   pStyle: #(sysmenu titlebar minimize maximize sizable);
   framingRatio:( (Rectangle leftTopUnit
      rightAndDown:0@(1/2)) extentFromLeftTop:1/2@(1/2) );
   viewName: 'erfassungsfenster2';
   noSmalltalkMenuBar;
   labelWithoutPrefix: 'Jahrgangsstufe 12';
   addSubpane:
     (EntryField new owner: WerteErfassung1112;
     framingRatio:( (Rectangle leftTopUnit
        rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) );
     paneName: 'eingabeFeld2')
  )
```

- **"addView:"**:

Durch die Ausführung der Message “addView:” wird der Verweis auf dasjenige View, das als Argument der Message “addView:” aufgeführt ist, in den geordneten Sammler in Form der Instanz-Variablen “views” eingetragen. Dieser geordnete Sammler ist eine Instanz der Klasse “OrderedCollection”, die zu der als Empfänger-Objekt von “addView:” aufgeführten Instanz gehört.

Hinweis: Dabei ist zu beachten, daß die Verweise auf die Views in den geordneten Sammler namens “views” in der Reihenfolge eingetragen werden, in der sie bei der Ausführung der Message “addView:” als Argument angegeben wurden.

Den Sachverhalt, daß die Instanz-Variable “views” – der Instanz “WerteErfassung1112” – zwei Verweise auf Views enthält, die aus der Klasse “TopPane” instanziiert wurden und deren Eigentümer die Instanz “WerteErfassung1112” ist, können wir folgendermaßen darstellen:

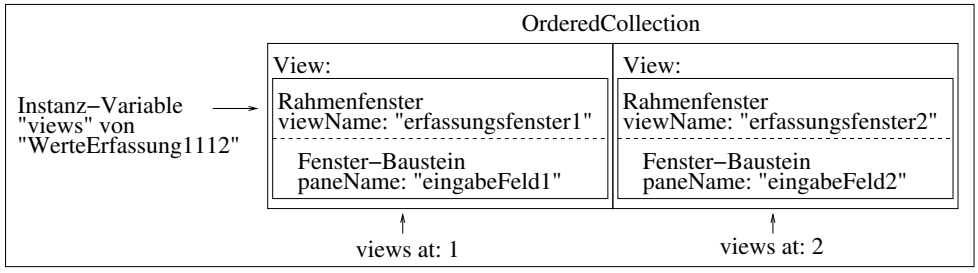


Abbildung 10.5: Instanz-Variable “views” von “WerteErfassung1112”

Um am Bildschirm sowohl das durch “erfassungsfenster1” als auch das durch “erfassungfenster2” gekennzeichnete View anzeigen zu lassen, ist die Anforderung

```
WerteErfassung1112 openWindow
```

auszuführen.

- **“openWindow”**:
  Durch den Einsatz dieser Message werden sämtliche Views, deren Verweise in der Instanz-Variablen “views” des Empfänger-Objektes von “openWindow” enthalten sind, eröffnet und am Bildschirm als Fenster angezeigt.

  Hinweis: Bei unserem Beispiel ist zu beachten, daß durch die Ausführung von

  ```
  WerteErfassung1112 openWindow
  ```

  beide Views an der gleichen Bildschirmposition angezeigt werden. Sollen beide Fenster gleichzeitig sichtbar sein, so müssen wir das zuletzt eröffnete View verschieben. Soll nur ein einzelnes View, wie z.B. das View namens “erfassungsfenster1” eröffnet und am Bildschirm angezeigt werden, so läßt sich dies durch eine Anforderung der Form

  ```
  (WerteErfassung1112 views at: 1) openWindow
  ```

  erreichen.

Wollen wir den Namen des Views bestimmen lassen, dessen zugehöriger Verweis an der ersten Position innerhalb der Instanz-Variablen "views" von "WerteErfassung1112" plaziert ist, so kann dies durch die Anforderung

```
(WerteErfassung1112 views at:1) paneName
```

erreicht werden.

- **"paneName"**:
  Das Empfänger-Objekt der Message "paneName" muß ein Verweis auf ein View sein, der innerhalb der Instanz-Variablen "views" enthalten ist. Als Ergebnis-Objekt wird der Name des Views ermittelt.

Unter Einsatz der Methoden "paneName" und "setFocus" kann z.B. mittels der Methode

```
aktivieren: aView
self views do: [:view|view paneName = aView ifTrue:[view setFocus]]
```

ein am Bildschirm angezeigtes Fenster als *aktives* Fenster eingestellt werden.

- **"setFocus"**
  Durch die Ausführung der Basis-Methode "setFocus" wird dasjenige am Bildschirm angezeigte Fenster aktiviert, das mit dem als Empfänger-Objekt aufgeführten View korrespondiert.

Um z.B. das durch den Viewnamen 'erfassungsfenster1' gekennzeichnete Fenster zu aktivieren, ist die folgende Anforderung zu stellen:

```
WerteErfassung1112 aktivieren: 'erfassungsfenster1'
```

Hinweis: Die Aktivierung dieses Views können wir auch durch die Anforderung

```
(WerteErfassung1112 views at: 1) setFocus
```

erreichen.
Um das Eingabefeld namens "eingabeFeld2" und damit das zugehörige, durch den Viewnamen "erfassungsfenster2" gekennzeichnete Fenster zu aktivieren, können wir die folgende Anforderung stellen:

```
(WerteErfassung1112 paneNamed: 'eingabeFeld2') setFocus
```

Zur Prüfung, ob ein View geöffnet und als Fenster angezeigt ist, läßt sich die Message "isVisible" in der Form

```
(WerteErfassung1112 views at:1) isVisible
```

einsetzen. Sofern das View eröffnet ist, resultiert die Pseudovariable "true" – andernfalls wird "false" ermittelt.
Soll auf die Titel-Zeile eines Fensters zugegriffen werden, so können wir die Message "label" innerhalb der folgenden Anforderung einsetzen:

```
(WerteErfassung1112 views at:1) label
```

Um die am Bildschirm angezeigten Views zu schließen, müssen wir die Anforderung

```
WerteErfassung1112 close
```

ausführen lassen.

- **“close”**:
  Die Views, auf die durch die Instanz-Variable “views” derjenigen Instanz verwiesen wird, die als Empfänger-Objekt von “close” aufgeführt ist, werden geschlossen und die korrespondierenden Fenster vom Bildschirm entfernt.

  Hinweis: Soll nur ein einzelnes View, wie z.B. das View namens “erfassungsfenster1” geschlossen und vom Bildschirm entfernt werden, so läßt sich dies durch eine Anforderung der Form

  ```
  (WerteErfassung1112 views at: 1) closeView
  ```

  erreichen.

10.8 Größenbestimmung von Fensterbereichen

Durch den Einsatz des Werkzeugs “WindowBuilder Pro/V” haben wir den Aufbau unseres Erfassungsfensters dadurch bestimmt, daß wir die Plazierung und die Größe der einzelnen Fenster-Bausteine durch das Ziehen mit der Maus geeignet festgelegt haben.
Durch die sich anschließende automatische Umsetzung in die Methode “createViews” resultierte für das Rahmenfenster und für jeden Fenster-Baustein eine Anforderung, in der die Message “framingBlock:” aufgeführt war.
Durch diese Message wird die Größe des Rahmenfensters bzw. die Ausdehnung und die Position eines Fenster-Bausteins festgelegt.

- Die Verwendung der Basis-Methode “framingBlock:” ist unbefriedigend, da die Größe der Fenster-Bausteine nicht angepaßt wird, sofern die Größe des Rahmenfensters mit der Maus verändert wird.

Deshalb ist es empfehlenswert, die Größen- und Positionsangaben – durch den Einsatz der Message “framingRatio:” – *relativ* zum zugehörigen Rahmenfenster festzulegen. Dies hat den Vorteil, daß bei einer Größenänderung des Rahmenfensters, die unter Einsatz der Maus vorgenommen wird, die Position und die Ausdehnung von Fenster-Bausteinen *automatisch* neu berechnet und deren Anzeige – entsprechend dem geänderten Rahmenfenster – *automatisch* angepaßt wird.
Grundsätzlich basiert die Bestimmung von rechteckigen Fensterbereichen auf der Festlegung von *Rechtecken*, die jeweils durch einen linken oberen Eckpunkt sowie den zugehörigen rechten unteren Eckpunkt gekennzeichnet sind.

- Während ein Rechteck als Instanziierung der Basis-Klasse “Rectangle” erhalten wird, werden einzelne Bildschirmpunkte (Pixel) als Instanzen der Basis-Klasse “Point” bestimmt.
  Eine Instanz der Klasse “Point” wird dadurch erzeugt, daß einem numerischen Empfänger-Objekt – der X-Koordinate – die binäre Message “@” mit einem numerischen Argument – der Y-Koordinate – zugestellt wird.

> **Hinweis:** Während die Werte der X- und der Y-Koordinaten bei einer Instanz der Klasse "Point" in den Instanz-Variablen "x" und "y" gespeichert werden, sind die Angaben über die beiden Eckpunkte eines Rechtecks bei einer Instanz der Klasse "Rectangle" den Instanz-Variablen "leftTop" und "rightBottom" zugeordnet.

Zum Beispiel wird durch die Anforderung

```
100 @ 200
```

der Punkt mit der X-Koordinate "100" und der Y-Koordinate "200" gekennzeichnet.

Hinweis: Mit Instanzen der Klasse "Point" können genauso wie es von Zahlen bekannt ist, numerische Berechnungen durchgeführt werden. Somit lassen sich Instanzen der Klasse "Point" z.B. addieren, subtrahieren, multiplizieren, dividieren und vergleichen.

Sofern ein Fensterbereich innerhalb eines Rahmenfensters in *absoluter* Form festgelegt werden soll, ist zu berücksichtigen, daß die X-Koordinate – horizontal – zum rechten Rand des Rahmenfensters mit jedem Bildschirmpunkt um den Wert "1" und die Y-Koordinate – vertikal – zum unteren Rand des Rahmenfensters ebenfalls um den Wert "1" wächst.

Hinweis: VGA-Bildschirme haben je nach Auflösung z.B. $640 * 480$ oder auch $1024 * 768$ Bildschirmpunkte.

Um z.B. einen rechteckigen Fensterbereich mit einer Höhe von "100" Bildschirmpunkten und einer Breite von "200" Bildschirmpunkten zu kennzeichnen, ist daher z.B. die linke obere Ecke durch die Koordinaten "100 @ 200" und die rechte untere Ecke durch die Koordinaten "200 @ 400" festzulegen.

Dies läßt sich z.B. durch die Message

```
100 @ 200  rightBottom: 200 @ 400
```

bewirken.

Wie oben angegeben, ist es im Hinblick auf die Beschreibung von Fensterbereichen sinnvoller, die Höhe eines Rechtecks nicht durch *absolute* Angaben, sondern durch eine Festlegung zu beschreiben, bei der die Höhe des Fensterbereichs als *relative* Höhe zur Höhe des Rahmenfensters und die Breite des Fensterbereichs als *relative* Breite zur Breite des Rahmenfensters angegeben wird.

Um diesen Sachverhalt durch eine geeignete Message festzulegen, müssen die linke obere und der rechte untere Eckpunkt des Rechtecks durch *Verhältnisangaben* gekennzeichnet werden.

Zum Beispiel beschreibt "2/8 @ (1/8)" einen Punkt, dessen X-Koordinate "2/8" Anteile der Breite des gesamten Rahmenfensters und dessen Y-Koordinate "1/8" Anteile der Höhe beträgt.

Hinweis: Es ist zu beachten, daß "2/8 @ (1/8)" mit der Klammerung angegeben werden muß, da die binären Messages "/" und "@" verwendet werden.

Sofern relative Fensterbereiche von Rahmenfenstern festzulegen sind, beziehen sie sich auf einen Koordinatenursprung, der durch die Angabe "0 @ 0" bestimmt ist und bei der Bildschirmanzeige des Rahmenfensters in der linken oberen Ecke festgelegt ist – unmittelbar unterhalb der angezeigten Menü-Leiste.

Um das von uns konzipierte Erfassungsfenster durch den Einsatz der Message “framingRatio:” festzulegen, betrachten wir die folgende Darstellung, bei der das Rahmenfenster in 64 rechteckige Zellen eingeteilt ist:

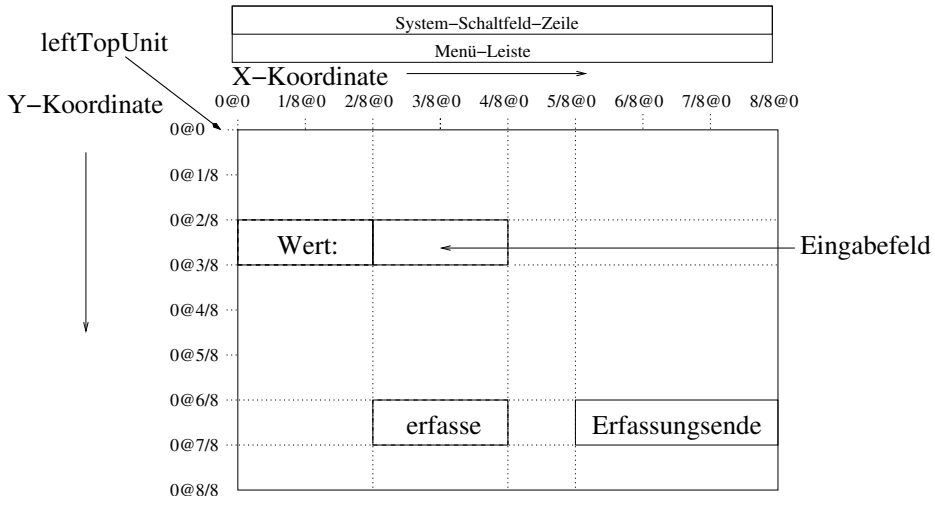


Abbildung 10.6: Aufteilung des Erfassungsfensters

Auf dieser Basis läßt sich z.B. der Fensterbereich, den das Eingabefeld einnehmen soll, durch die folgende Message bestimmen:

```
(Rectangle leftTopUnit rightAndDown: 2/8 @ (2/8))
   extentFromLeftTop: 2/8 @ (1/8)
```

Diese Anforderung basiert auf der folgenden Wirkung der Messages “leftTopUnit”, “rightAndDown:” und “extentFromLeftTop:”:

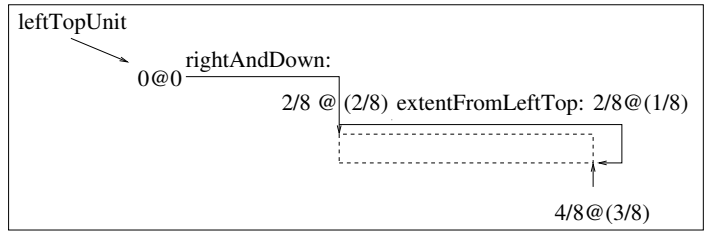


Abbildung 10.7: Bestimmung eines rechteckigen Fensterbereichs

Als Ergebnis-Objekt der Message “Rectangle leftTopUnit” wird ein Punkt mit den Koordinaten “0 @ 0” als Instanz der Klasse “Point” erhalten. Anschließend wird

dieser Instanz die Keyword-Message “rightAndDown:” mit dem Argument “2/8 @ (2/8)” in der Form

```
rightAndDown: 2/8 @ (2/8)
```

geschickt. Als Ergebnis-Objekt dieser Message resultiert ein Punkt mit den Koordinaten “2/8 @ (2/8)”, der die linke obere Ecke des Rechtecks kennzeichnet.

Wird diesem Punkt daraufhin durch die Keyword-Message “extentFromLeftTop:” der Wert “2/8” als Ausdehnung in der X-Koordinatenrichtung und der Wert “1/8” als Ausdehnung in der Y-Koordinatenrichtung in der Form

```
extentFromLeftTop: 2/8 @ (1/8)
```

zugestellt, so ergibt sich eine Instanz der Klasse “Rectangle” mit der linken oberen Ecke im Punkt “2/8 @ (2/8)” und der rechten unteren Ecke im Punkt “4/8 @ (3/8)”.

Hinweis: Als Ergebnis-Objekt der obigen Anforderung wird “1/4 @ 1/4 rightBottom: 1/2 @ 3/8” erhalten.

Damit dieser Fensterbereich für das Eingabefeld festgelegt wird, ist die Message “framingRatio:” – bei der Vereinbarung dieses Fenster-Bausteins – z.B. wie folgt einzusetzen:

```
EntryField new owner: WerteErfassung11;
  framingRatio:( (Rectangle leftTopUnit
     rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) )
```

Hinweis: Durch die Message “framingRatio:” wird die erzeugte Instanz der Klasse “Rectangle” in die Anforderungen eines Blockes der Form

```
[:box|box scaleTo: aRectangle]
```

eingetragen und der Instanz-Variablen “framingBlock” des jeweiligen Fenster-Bausteins zugeordnet. Dieser Block wird vom SMALLTALK-System für die Anzeige des Fenster-Bausteins ausgewertet. Soll die Größe von Fenster-Bausteinen sich an der eingestellten Schriftgröße orientieren, so kann das Ergebnis-Objekt der Anforderung “SysFont height”, das in Pixelform ermittelt wird, einer temporären Variablen zugewiesen und sämtliche Angaben unter Einsatz dieser Variablen skaliert werden.

Kapitel 11

Ereignis-gesteuerte und indirekte Kommunikation

11.1 Ereignisse

Während des Dialogs mit dem SMALLTALK-System werden Anforderungen dadurch zur Ausführung gebracht, daß mit der Maus auf die Menü-Option "Do It" ("Show It") des Menüs "Smalltalk" oder – innerhalb eines Fensters – auf einen Fenster-Baustein, wie z.B. auf das Schaltfeld mit der Aufschrift "erfasse" geklickt wird. In Abhängigkeit davon, welcher Fenster-Baustein von einem Mausklick betroffen ist und in welcher Situation ein Mausklick erfolgt, wird ein jeweils zugehöriges *Ereignis* ausgelöst.

Sämtliche Ereignisse, die in dieser Hinsicht eintreffen können, sind durch *Ereignisnamen* – in Form von Symbolen – festgelegt. Jedem Ereignisnamen ist system-seitig eine Methode zugeordnet, die dann zur Ausführung gelangt, wenn das durch den Ereignisnamen gekennzeichnete Ereignis eintritt.

Da alle Anforderungen entweder über Mausklicks in einem system-seitig bereitgestellten Fenster – wie z.B. einem Workspace-Fenster oder dem Klassen-Hierarchie-Browser-Fenster – oder über Mausklicks auf Fenster-Bausteine eines durch den Anwender aufgebauten Fensters gestellt werden, beruht der gesamte Dialog mit dem SMALLTALK-System auf dem Grundprinzip, daß das SMALLTALK-System in geeigneter Weise auf Ereignisse reagiert.

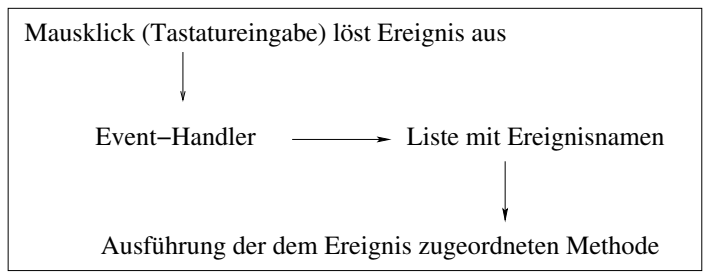


Abbildung 11.1: Reaktion auf ein Ereignis

- Daher können wir uns die Ausführung des SMALLTALK-Systems so vorstel-

len, daß ein Event-Handler, d.h. ein Ereignis-Überwachungsprogramm, gestartet ist, dessen einzige Aufgabe darin besteht, auf Ereignisse, die durch einen Mausklick oder durch eine geeignete Tastatureingabe ausgelöst werden, zu reagieren, indem zuvor zugeordnete Methoden zur Ausführung gebracht werden.

Damit die Ausführung bestimmter Anforderungen – im Hinblick auf den jeweiligen Lösungsplan – über einen geeigneten Mausklick abgerufen werden kann, müssen die system-seitig zur Verfügung gehaltenen Ereignisnamen verwendet werden. Dies bedeutet, daß beim Aufbau eines Rahmenfensters oder zugehöriger Fenster-Bausteine eine Verbindung zwischen einem Mausklick und der Ausführung der gewünschten Anforderungen über einen Ereignisnamen hergestellt werden muß.

Soll beim Aufbau eines Rahmenfensters verabredet werden, daß eine Methode mit dem Methoden-Selektor "<selektor>:" zur Ausführung gelangt, sofern das Ereignis mit dem Ereignisnamen "<ereignisname>" eintritt, so ist unter Einsatz der Basis-Methode "when:perform:" die Verbindung zwischen dem Ereignisnamen und dem Methoden-Selektor wie folgt herzustellen:

- `<aPane> when: #<ereignisname> perform: #<selektor>:`

Hinweis: Das Empfänger-Objekt "<aPane>" steht für eine Instanz der Klasse "TopPane" bzw. "DialogTopPane".

Entsprechend ist

- `<aSubPane> when: #<ereignisname> perform: #<selektor>:`

beim Aufbau eines Fenster-Bausteins zu formulieren, sofern durch ein Ereignis, wie z.B. einen Mausklick auf den durch "<aSubPane>" gekennzeichneten Fenster-Baustein auf das Ereignis "#<ereignisname>" reagiert werden soll.

Hinweis: Das Empfänger-Objekt "<aSubPane>" steht für eine Instanz, die aus einer der Klasse "SubPane" untergeordneten Basis-Klasse instanziiert wurde.

Die durch "<selektor>:" gekennzeichnete Methode ist innerhalb derjenigen Klasse festzulegen, aus der der *Eigentümer* des Rahmenfensters bzw. des Fenster-Bausteins instanziiert wurde.

Es ist grundsätzlich zu beachten, daß durch den Einsatz der Message "when:perform:" jedem Ereignis nur eine auszuführende Methode zugewiesen werden kann.

- **"when:perform:"**:
  Durch die Message "when:perform:" wird einem Ereignis eine auszuführende Methode zugeordnet. Diese Methode ist als Argument des Selektors "perform:" und der das Ereignis kennzeichnende Ereignisname als Argument des Selektors "when:" aufzuführen. Sofern das Ereignis eintritt, wird die Methode in der Klasse gesucht, aus der der Eigentümer des Rahmenfensters bzw. des Fenster-Bausteins instanziiert wurde.
  Es ist zu beachten, daß die Argumente der Selektoren "when:" und "perform:" Symbole sein müssen. Außerdem muß das hinter "perform:" aufgeführte Argument durch einen Doppelpunkt abgeschlossen werden, da die Methode, die

dem Ereignisnamen zugeordnet wird, aus formalen Gründen mit einem Argument vereinbart sein muß. Dadurch wird implizit das Rahmenfenster bzw. der Fenster-Baustein festgelegt, in dem das Ereignis ausgelöst wird.

Tritt durch die Bearbeitung eines Fensters ein Ereignis ein, das für das Rahmenfenster oder einen Fenster-Baustein system-seitig festgelegt ist, und ist diesem Ereignis kein Methoden-Selektor durch die Message "when:perform:" zugeordnet worden, so ist das Ereignis wirkungslos.

Welche Ereignisse für Rahmenfenster und für Fenster-Bausteine aktiviert werden können und wie auf derartige Ereignisse reagiert werden kann, stellen wir in den nachfolgenden Abschnitten dar. An dieser Stelle skizzieren wir die unterschiedlichen Möglichkeiten für die ereignis-gesteuerte Kommunikation durch die folgende Darstellung:

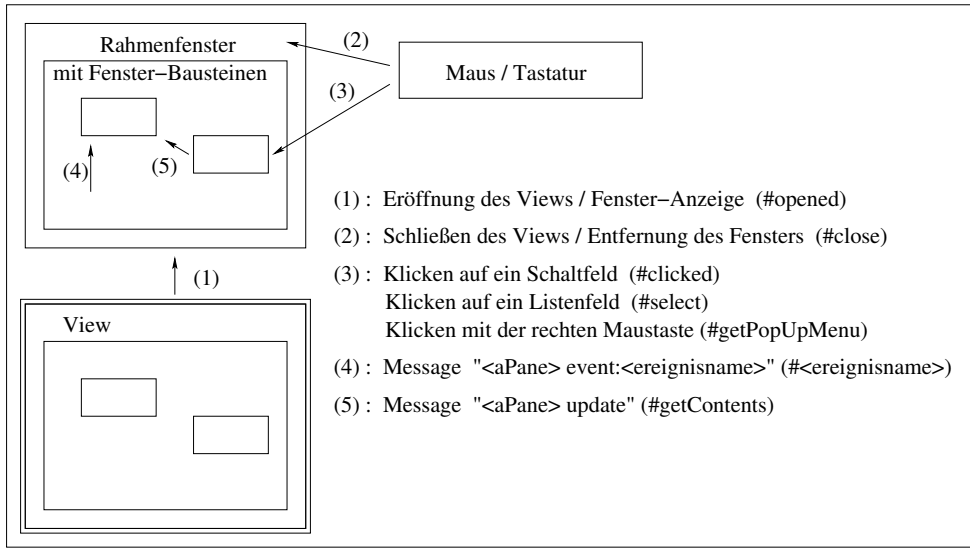


Abbildung 11.2: Ereignis-gesteuerte Kommunikation

11.2 Ereignisse für Rahmenfenster

Bei der Festlegung eines Rahmenfensters kann z.B. der Ereignisname *"#opened"* verwendet werden. Dieser Name kennzeichnet das Ereignis "das View wird eröffnet und am Bildschirm als Fenster angezeigt". Beim Eröffnen des Views wird dieses Ereignis *automatisch* ausgelöst, indem es – vom SMALLTALK-System – dem Rahmenfenster durch den Einsatz der Basis-Methode "event:" in der Form

```
<aTopPane> event: #opened
```

zugestellt wird.

Hinweis: Neben dem Ereignis "#opened" tritt bei der Eröffnung eines Views auch das Ereignis "#menuBuilt" ein, so daß eine geeignet vereinbarte Methode "<selektor>:" über die Zuordnung

```
when: #menuBuilt perform: #<selektor>:
```

zur Ausführung gelangen kann (siehe Abschnitt 11.4).

- **"event:"**:
  Wurde die Message "when:perform:" bei der Vereinbarung eines Rahmenfensters in der Form

  when: #<ereignisname> perform: #<selektor>:

  eingesetzt, so wird der *Eigentümer* des Empfänger-Objektes, bei dem es sich um ein View handelt, durch die Ausführung der Message

  <aTopPane> event: #<ereignisname>

  benachrichtigt, auf das durch den Namen "<ereignisname>" gekennzeichnete Ereignis durch die Ausführung der Methode "<selektor>:" zu reagieren.

Soll z.B. als geeignete Reaktion für den Ereignisnamen "#opened" festgelegt werden, daß die Methode "vereinbarungMenuLeiste:" ausgeführt werden soll, so muß die Message

```
when: #opened perform: #vereinbarungMenuLeiste:
```

wie folgt bei der Vereinbarung eines Views verwendet werden:

```
WerteErfassung11 addView:
 ( WerteErfassung11 topPaneClass new owner:WerteErfassung11;
     framingRatio: (  .........  );
     viewName: 'erfassungsfenster';
     when: #opened perform: #vereinbarungMenuLeiste:;
     ... ;
     addSubpane: (  ...........  );
     ... ;
     addSubpane: (  ...........  );
     ...
 )
```

Wird dieses aus der Klasse "WerteErfassung" instanziierte View am Bildschirm als Fenster eröffnet, so tritt das Ereignis "#opened" ein, und es wird die Methode "vereinbarungMenuLeiste:" in der Klasse des Eigentümers des Views, d.h. in der Klasse "WerteErfassung", gesucht und ausgeführt.
Hinweis: Durch die Methode "vereinbarungMenuLeiste:" kann z.B. festgelegt werden, daß das Fenster mit einer bestimmten Menü-Leiste versehen wird (siehe Abschnitt 11.4).

Neben dem Ereignisnamen "#opened" ist der Name *"#close"* als weiterer Ereignisname für ein Rahmenfenster vereinbart. Dieser Name korrespondiert mit einem Ereignis, das beim Schließen des Rahmenfensters ausgelöst wird, sofern dies über

die Tastenkombination “Alt + F4” bzw. die Menü-Option “Schließen” des System-Menüs angefordert wird.
Um eine geeignete Reaktion für das Ereignis “#close” festzulegen, kann die Message “when:perform:” z.B. in der Form

```
when: #close perform: #beendenErfassung:
```

innerhalb der Vereinbarung eines Rahmenfensters verwendet werden.
Sofern wir z.B. die Methode “beendenErfassung:” in der Form

```
beendenErfassung: aView
werteBag add: (self paneNamed: 'eingabeFeld') contents.
Transcript cr; show: 'Erfassung beendet'.
self close
```

innerhalb der Klasse “WerteErfassung” vereinbart haben, bewirkt das Ereignis “#close”, daß der Inhalt des Eingabefeldes in den Sammler “werteBag” übertragen, der Text “Erfassung beendet” im Transcript-Fenster angezeigt und das Erfassungsfenster vom Bildschirm entfernt wird.
Hinweis: Das Ereignis “#close” tritt auch durch das Betätigen des System-Schaltfeldes “Schließen” ein. Es ist jedoch zu beachten, daß das Erfassungsfenster erst durch die Ausführung der Methode “close” vom Bildschirm entfernt wird.
Welche weiteren Ereignisnamen – neben “#opened” und “#close” – in Verbindung mit einem Rahmenfenster zulässig sind, läßt sich unter Einsatz der Klassen-Methode “supportedEvents” feststellen.

- **“supportedEvents”**:
  Als Ergebnis-Objekt der Message “supportedEvents” resultiert ein Set, in dem sämtliche Ereignisse eingetragen sind, die für Instanzen aus der Klasse des Empfänger-Objektes zulässig sind.

Als Empfänger-Objekt der Message “supportedEvents” können – neben den Klassen “TopPane” und “DialogTopPane” – auch alle Unterklassen von “SubPane” dienen.

11.3 Ereignisse für Fenster-Bausteine

Um z.B. für den Fenster-Baustein “Schaltfeld” – d.h. einer Instanz der Basis-Klasse “Button”–, zu erfahren, welche Ereignisse mit ihm verknüpft sind, ist die folgende Anforderung zu stellen:

```
Button supportedEvents
```

Im Ergebnis-Objekt dieser Anforderung ist unter anderen Namen der Ereignisname “clicked” enthalten, durch den das Ereignis “Klicken auf ein Schaltfeld” gekennzeichnet ist.
Sofern z.B. die Ausführung der Methode “erfassenWert:” durch einen Mausklick auf ein Schaltfeld – mit der Aufschrift “erfasse” – ausgelöst werden soll, ist dieses

Schaltfeld wie folgt als Instanz der Basis-Klasse “Button” zu vereinbaren:

```
Button new owner: WerteErfassung11;
 framingRatio:( .... );
 contents: 'erfasse';
 when: #clicked perform: #erfassenWert:
```

Für ein Eingabefeld, das aus einer Instanziierung der Basis-Klasse “EntryField” eingerichtet wird, ist z.B. das Ereignis “Ändern des Feldinhaltes” festgelegt. Es wird durch den Ereignisnamen *“#textChanged”* gekennzeichnet und tritt dann ein, wenn der Inhalt des Eingabefeldes geändert und ein anderes Fenster oder ein anderer Fenster-Baustein des Fensters, in dem das Eingabefeld als Baustein enthalten ist, aktiviert wird.

Falls z.B. das durch den Namen “eingabeFeld” benannte Eingabefeld in der Form

```
EntryField new owner: WerteErfassung11;
 framingRatio:( ... );
 paneName: 'eingabeFeld';
 when: #textChanged perform: #untersuchenWert:
```

als Fenster-Baustein verabredet ist, wird die Methode “untersuchenWert:” zur Ausführung gebracht, sofern das Ereignis “#textChanged” eingetreten ist.

Hinweis: Um mitzuteilen, daß eine Änderung erfolgt ist, kann die Methode “untersuchenWert:” testweise z.B. wie folgt als Instanz-Methode innerhalb der Klasse des Eigentümers des Eingabefeldes und somit in der Klasse “WerteErfassung”, in der auch das Rahmenfenster mit diesem Fenster-Baustein vereinbart wurde, festgelegt werden:

```
untersuchenWert: anEntryField
Transcript cr; show: 'Eingabe erfolgt ';
  show: (self paneNamed: 'eingabefeld') contents size printString
```

- Grundsätzlich läßt sich mit Hilfe von Ereignissen steuern, welche Views in welcher Form als Fenster am Bildschirm eröffnet werden. Dabei kann in Abhängigkeit von bestimmten Tastatureingaben oder Mausklicks eine Kommunikation der Fenster-Bausteine ausgelöst werden. Dies kann z.B. die Form des Views beeinflussen, indem dadurch die Anzeige bestimmter Fenster-Bausteine geändert wird.

Als *allgemein* für Fenster-Bausteine zulässige Ereignisnamen stehen unter anderem zur Verfügung:

- “#losingFocus” bzw. “#gettingFocus”:
  Diese Ereignisse treten dann ein, wenn der jeweilige Fenster-Baustein bzw. ein anderes Fenster durch den Einsatz der linken Maustaste deaktiviert bzw. aktiviert wird.
- “#getPopupMenu”:

Dieses Ereignis wird ausgelöst, sofern ein Mausklick mit der rechten Maustaste erfolgt.

- "#getContents":
  Dieses Ereignis tritt dann ein, wenn ein View eröffnet und am Bildschirm als Fenster angezeigt oder die Message "update" an einen Fenster-Baustein gesandt wird.

Grundsätzlich läßt sich eine unmittelbare Kommunikation zwischen einzelnen Fenster-Bausteinen durch den Einsatz der Basis-Methode "update" ermöglichen.

- **"update"**:
  Dem Empfänger-Objekt der Message "update" wird das Ereignis "#getContents" in der Form "self event: #getContents" zugestellt.

Soll sich z.B. der Inhalt eines Textfeldes durch das Klicken auf ein Optionsfeld ändern, so wird bei der Ausführung der Methode, die mit dem Klicken auf das Optionsfeld durch die Message "when:perform:" verbunden ist, *automatisch* die Message "update" an das Textfeld geschickt.
Für dieses Textfeld müssen wir durch die Message "when:perfom:" eine Verbindung zwischen dem Ereignis "#getContents" und einer Methode hergestellt haben, durch deren Ausführung in geeigneter Weise auf einen Mausklick auf das Optionsfeld reagiert wird.

Neben den generell zur Verfügung stehenden Ereignissen gibt es weitere Ereignisse, die nur in Verbindung mit *bestimmten* Fenster-Bausteinen ausgelöst werden können. Hierzu zählen:

- "#clicked" für Instanzen der Basis-Klasse "Button":
  Das Ereignis "#clicked" tritt dann ein, wenn ein Mausklick auf ein Schaltfeld erfolgt.

- "#textChanged" für Instanzen der Basis-Klassen "EntryField" und "TextEdit":
  Das Ereignis "#textChanged" wird ausgelöst, wenn der Inhalt eines Eingabefeldes bzw. eines Editierfeldes durch eine Tastatureingabe geändert und danach ein anderer Fenster-Baustein oder ein anderes Fenster aktiviert wird.

  Hinweis: Dieses Ereignis tritt auch dann ein, wenn das Fenster über das System-Schaltfeld "Schließen" geschlossen wird.
  Da intern registriert wird, ob der Inhalt des Eingabe- bzw. des Editierfelds geändert wurde, tritt das Ereignis "#textChanged" nur dann ein, wenn der Inhalt des Feldes durch eine Tastatureingabe auch tatsächlich verändert wurde. Somit wird dieses Ereignis z.B. dann *nicht* ausgelöst, wenn der ursprüngliche Inhalt identisch überschrieben wird.

- "#select" und "#doubleClickSelect" für Instanzen der Basis-Klassen "ListBox" und "ComboBox":
  Die Ereignisse "#select" bzw. "#doubleClickSelect" werden ausgelöst, wenn ein Mausklick bzw. ein Doppelklick auf ein Listen- oder Kombinationsfeld erfolgt.

11.4 Pulldown-Menüs

Innerhalb der Fenster des SMALLTALK-Systems – wie z.B. dem Klassen-Hierarchie-Browser-Fenster oder dem Workspace-Fenster – lassen sich bestimmte Anforderungen über die Menüs der jeweiligen Menü-Leiste stellen, die in diesen Fenstern unmittelbar unterhalb der System-Schaltfeld-Zeile angezeigt wird. Dabei ist die jeweils gewünschte Menü-Option einer dieser *Pulldown-Menüs* über einen Mausklick bzw. eine geeignete Tastenkombination der Alt-Taste mit einer Buchstaben-Taste abzurufen.

Hinweis: Welcher Buchstabe jeweils zu verwenden ist, zeigt die Unterstreichung innerhalb des Textes, der die jeweilige Menü-Option kennzeichnet. Zum Beispiel kann im Klassen-Hierarchie-Browser-Fenster eine Sicherung über die Tastenkombination "Alt+S" ("S" ist innerhalb des Textes "Save" unterstrichen) abgerufen werden.

Standardmäßig wird jedes anwender-seitig eingerichtete Rahmenfenster mit einer system-seitig voreingestellten Menü-Leiste angezeigt, die das Menü "File" als einziges Menü enthält.

Soll diese standardmäßige Menü-Leiste durch eine eigene Menü-Leiste ersetzt werden, so muß dies bei der Einrichtung des zugehörigen Views festgelegt werden. Dazu kann die Message "when:perform:" eingesetzt werden, wobei dem Ereignis *"#menuBuilt"*, das bei der Eröffnung des Views ausgelöst wird, eine geeignete Methode zum Aufbau des gewünschten Menüs zugeordnet wird.

Hinweis: Statt des Ereignisses "#menuBuilt" kann auch das Ereignis "#opened" verwendet werden.

Sofern wir z.B. in der Instanz-Methode "initialisierenErfassung", die von uns innerhalb der Klasse "WerteErfassung" zum Aufbau eines Views für den Erfassungsprozeß vereinbart wurde, die Message

```
when: #menuBuilt perform: #vereinbarungMenuLeiste:
```

eintragen und innerhalb der Klasse "WerteErfassung" die Methode "vereinbarungMenuLeiste:" in der Form

```
vereinbarungMenuLeiste: aView
(aView menuWindow)
   removeMenu: (aView menuTitled: 'File');
   addMenu: (Menu new owner: self;
               title: 'Pruefen';
               appendItem: 'Anzahl' selector: #anzeigenAnzahl;
               appendItem: 'Werte' selector: #anzeigenWerte)
```

festlegen, wird durch die Anforderungen

```
WerteErfassung11 := WerteErfassung new.
WerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

ein Erfassungsfenster angezeigt, das in der Menü-Leiste das Pulldown-Menü "Pruefen" mit den Menü-Optionen "Anzahl" und "Werte" enthält.

Dieser Sachverhalt ist durch die folgenden Messages innerhalb der Methode "vereinbarungMenuLeiste:" gesichert:

- **“menuWindow”**:
  Als Ergebnis-Objekt wird eine *Menü-Leiste* – als Instanziierung der Klasse “MenuWindow” – desjenigen Views ermittelt, das als Empfänger-Objekt der Message “menuWindow” aufgeführt ist.

  Hinweis: Auf diese Menü-Leiste wird über die Instanz-Variable “menuWindow” des Rahmenfensters verwiesen.

- **“menuTitled:”**:
  Für das als Empfänger-Objekt aufgeführte View wird dasjenige Menü als Ergebnis-Objekt ermittelt, dessen Name als Argument der Message “menuTitled:” angegeben ist.

- **“removeMenu:”**:
  Aus der Menü-Leiste, die als Empfänger-Objekt aufgeführt ist, wird das durch das Argument gekennzeichnete Menü entfernt.

- **“addMenu:”**:
  In der Menü-Leiste, die als Empfänger-Objekt aufgeführt ist, wird das durch das Argument gekennzeichnete Menü ergänzt.

- **“Menu new”**:
  Es wird ein Menü als Instanz der Basis-Klasse “Menu” eingerichtet.

- **“title:”**:
  Das als Empfänger-Objekt aufgeführte Menü erhält den Namen, der als Argument der Message “title:” angegeben ist.

- **“appendItem:selector:”**:
  Das als Empfänger-Objekt aufgeführte Menü wird um eine Menü-Option ergänzt, deren Name als Argument von “appendItem:” angegeben ist und deren zugeordnete Methode, die bei einer Bestätigung dieser Menü-Option zur Ausführung gelangen soll, als Argument von “selector:” aufgeführt ist.

In der oben – innerhalb der Methode “vereinbarungMenuLeiste:” – angegebenen Vereinbarung des Menüs “Pruefen” ist festgelegt, daß die Menü-Option “Anzahl” mit der Methode “anzeigenAnzahl” und die Menü-Option “Werte” mit der Methode “anzeigenWerte” verbunden ist.

Das Menü “Pruefen” hat somit den folgenden Aufbau:

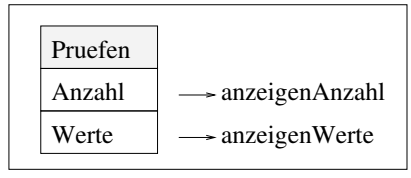


Abbildung 11.3: Das Pulldown-Menü “Pruefen”

Damit die Anzahl der erfaßten Werte angezeigt werden kann, ist z.B. die folgende Methode innerhalb der Klasse “WerteErfassung” zu vereinbaren:

```
anzeigenAnzahl
Transcript cr;
   show: 'Anzahl: ';
   show: self bereitstellenWerte size printString
```

Sollen die beiden Menü-Optionen “Anzahl” und “Werte” jeweils durch eine geeignete Tastenkombination – z.B. “Anzahl” durch “Alt+A” und “Werte” durch “Alt+W” – aktiviert werden können, so müssen in der oben angegebenen Vereinbarung anstelle der Messages “appendItem:selector:” die Messages “appendItem:selector:accelKey:accelBits:” in der Form

```
appendItem: '&Anzahl' selector: #anzeigenAnzahl
  accelKey: $A accelBits: AfAlt
```

bzw. in der Form

```
appendItem: '&Werte' selector: #anzeigenWerte
  accelKey: $W accelBits: AfAlt
```

verwendet werden.

Hinweis: Hierbei ist vorauszusetzen, daß bei der Vereinbarung der Klasse “WerteErfassung” das Dictionary “VirtualKeyConstants” als Pool-Dictionary-Variable angegeben wurde.
Durch das im Argument des Selektors “appendItem:” aufgeführte Zeichen “&” wird festgelegt, daß der diesem Zeichen folgende Buchstabe bei der Anzeige der Menü-Option in unterstrichener Form ausgegeben wird.

Sofern bei einer Anwendung eine Verschachtelung von Pulldown-Menüs gewünscht wird, indem durch eine Menü-Option ein weiteres Menü – als *Sub-Menü* – zur Anzeige gebracht werden soll, läßt sich die Message “appendSubMenu:” einsetzen.
Beabsichtigen wir, das Pulldown-Menü “Pruefen” in der Form

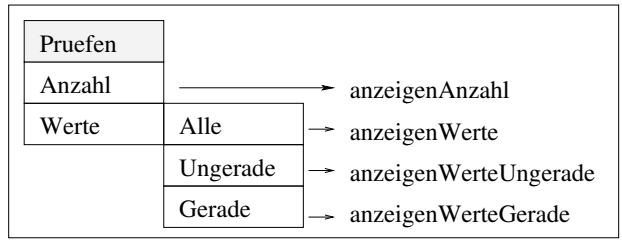


Abbildung 11.4: Das erweiterte Pulldown-Menü “Pruefen”

zu erweitern, so können wir die oben angegebene Methode “vereinbarungMenuLeiste:” z.B. wie folgt abändern:

```
vereinbarungMenuLeiste: aView
(aView menuWindow)
   removeMenu: (aView menuTitled: 'File');
   addMenu: (Menu new owner: self;
         title: 'Pruefen';
         appendItem: 'Anzahl' selector: #anzeigenAnzahl;
         appendItem: 'Werte'  selector: #anzeigenWerte;
         appendSubMenu: (Menu new owner: self;
           title: 'Werte';
           appendItem: 'Alle' selector: #anzeigenWerte;
           appendItem: 'Ungerade' selector: #anzeigenWerteUngerade;
           appendItem: 'Gerade' selector: #anzeigenWerteGerade)
            )
```

Hinweis: Dabei sind die aufgeführten Methoden "anzeigenWerteGerade" und "anzeigenWerteUngerade" geeignet zu vereinbaren.

11.5 Kontext-Menüs

Eine besondere Form von Menüs stellen die *Kontext-Menüs* (Popup-Menüs) dar, mit denen sich standardisierte Anforderungen in kontextabhängiger Form innerhalb eines Fensters abrufen lassen. Im Gegensatz zu den Menüs einer Menü-Leiste, die grundsätzlich unterhalb der System-Schaltfeld-Zeile im Rahmenfenster plaziert ist, können Kontext-Menüs durch einen Mausklick mit der rechten Maustaste an fast jeder beliebigen Fensterposition abgerufen werden.
Die Grundlage hierfür besteht darin, daß ein Kontext-Menü in Verbindung mit einem Fenster-Baustein vereinbart wird. Ist dies geschehen (siehe unten) und der Mauszeiger auf diesen Fenster-Baustein bewegt worden, so bewirkt der Klick auf die *rechte* Maustaste, daß das für den Fenster-Baustein festgelegte Kontext-Menü am Bildschirm erscheint.

- Grundsätzlich wird stets das Ereignis *"#getPopupMenu"* ausgelöst, sofern ein Mausklick mit der *rechten* Maustaste erfolgt.

Daher ist für die betreffenden Fenster-Bausteine, mit denen geeignete Kontext-Menüs zu verbinden sind, jeweils eine Zuordnung – durch die Methode "when:perform:" – zwischen dem Ereignisnamen *"#getPopupMenu"* und derjenigen Methode herzustellen, durch deren Ausführung die Anzeige des jeweils gewünschten Kontext-Menüs erreicht werden soll.

Hinweis: Die Vereinbarung einer Message "when:perform:" mit dem Ereignisnamen "#getPopupMenu" ist für das Rahmenfenster *nicht* zulässig, da dieser Ereignisname weder für die Klasse "TopPane:", noch für die Klasse "DialogTopPane" verwendet werden darf.

Um einem Fenster-Baustein ein Kontext-Menü zuzuordnen, ist die Methode "setPopupMenu:" einzusetzen.

- **"setPopupMenu:"**:
  Dem Empfänger-Objekt der Message "setPopupMenu:" wird ein Kontext-Me-

nü zugeordnet. Dieses Menü ist als Instanz der Basis-Klasse “Menu” festzulegen und als Argument von “setPopupMenu:” anzugeben.

Soll z.B. ein Kontext-Menü der Form

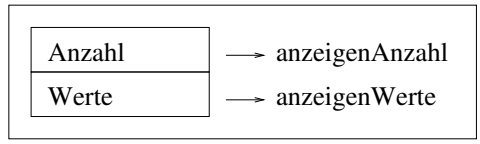


Abbildung 11.5: Kontext-Menü für einen Fenster-Baustein

angezeigt werden, wenn – innerhalb eines Erfassungsfensters – mit der rechten Maustaste auf das Schaltfeld mit der Aufschrift “erfasse” geklickt wird, so ist bei der Vereinbarung dieses Schaltfeldes die Zuordnung

```
when:#getPopupMenu perform: #vereinbarungPopupMenu:
```

anzugeben und die Methode “vereinbarungPopupMenu:” wie folgt – innerhalb der Klasse “WerteErfassung” – festzulegen:

```
vereinbarungPopupMenu: aButton
(self paneNamed: 'erfassungsButton')
   setPopupMenu:
     (Menu new owner: self;
        appendItem: 'Anzahl' selector: #anzeigenAnzahl;
        appendItem: 'Werte' selector: #anzeigenWerte)
```

Hinweis: Dabei setzen wir voraus, daß bei der Vereinbarung des Schaltfeldes mit der Aufschrift “erfasse” diesem Schaltfeld – innerhalb der Methode “initialisierenErfassung” – durch den Einsatz der Message “paneName:” in der Form

```
paneName: 'erfassungsButton'
```

der Name “erfassungsButton” zugeordnet wurde.

11.6 Indirekte Kommunikation

Neben der bislang vorgestellten Möglichkeit, Objekte *direkt* miteinander kommunizieren zu lassen, gibt es mit dem *Abhängigkeits-Mechanismus* eine weitere Kommunikations-Möglichkeit, bei der die Kommunikation *indirekt* erfolgt. Grundlage für diese *indirekte* Kommunikation ist die Festlegung von abhängigen und unabhängigen Objekten.

Um Objekte als *unabhängige* Objekte zu vereinbaren und diesen Objekten andere Objekte als *abhängige* Objekte zuzuordnen, ist die Basis-Methode “addDependent:” zu verwenden.

- **"addDependent:"**: Durch den Einsatz der Message "addDependent:" wird das Empfänger-Objekt als *unabhängiges* Objekt und das als Argument aufgeführte Objekt als das diesem Objekt zugeordnete *abhängige* Objekt vereinbart. Diese Zuordnung wird in einem Dictionary festgehalten, auf das die globale Variable "Dependents" verweist.

  Hinweis: Beim Einsatz der Message "addDependent:" ist es möglich, einem unabhängigen Objekt mehrere abhängige Objekte zuzuordnen.
  Um sämtliche Abhängigkeitsbeziehungen wieder aufzuheben, kann die Anforderung
  `Dependents := Dictionary new`
  gestellt werden.

Sind Abhängigkeiten zwischen Objekten festgelegt worden, so lassen sich den abhängigen Objekten automatisch spezielle Messages – namens "update:" – zustellen, sofern eine besondere Message – namens "changed:" – einem unabhängigen Objekt zugestellt wird.

- **"changed:"**:
  Sämtlichen dem Empfänger-Objekt als abhängige Objekte zugeordneten Objekten wird automatisch die Message "update:" zugestellt. Das aufgeführte Argument wird an die Message "update:" als Argument weitergereicht.

  Hinweis: Als Argument der Message "changed:" kann z.B. unter Einsatz der Pseudovariablen "self" das unabhängige Objekt oder z.B. ein Methoden-Selektor in Form eines Symbols aufgeführt werden.

- **"update:"**:
  Die zu dieser Message gehörige Methode *muß* geeignet – als Redefinition der Basis-Methode "update:" – vereinbart werden, so daß sie den Instanzen der Klasse der abhängigen Objekte bekannt ist. Bei dieser Methoden-Vereinbarung sind diejenigen Anforderungen festzulegen, die von den abhängigen Objekten ausgeführt werden sollen, falls dem zugehörigen unabhängigen Objekt die Message "changed:" geschickt wird. Dabei wird das Argument der Message "changed:" an die Message "update:" weitergereicht.

Hinweis: Es ist zu beachten, daß die Message-Selektoren "changed:" und "update:" vom SMALLTALK-System fest vorgegeben sind. Zur Übergabe zweier bzw. dreier Argumente sind die Methode "changed:with:" und die korrespondierende Methode "update:with:" bzw. "changed:with:with:" und die korrespondierende Methode "update:with:with:" zu verwenden.

Um ein einfaches Beispiel für die indirekte Kommunikation zu geben, betrachten wir die folgende Problemstellung:

- PROB-8:
  Während eines oder mehrerer paralleler Erfassungsprozesse soll die Anzahl der insgesamt erfaßten Punktwerte gezählt und bei jedem erfaßten Punktwert die Anzahl der bislang erfaßten Werte sowie der Name der Instanziierung des jeweiligen Erfassungsprozesses im Transcript-Fenster protokolliert werden.

Als Lösungsplan konzipieren wir die folgende Strukturierung:

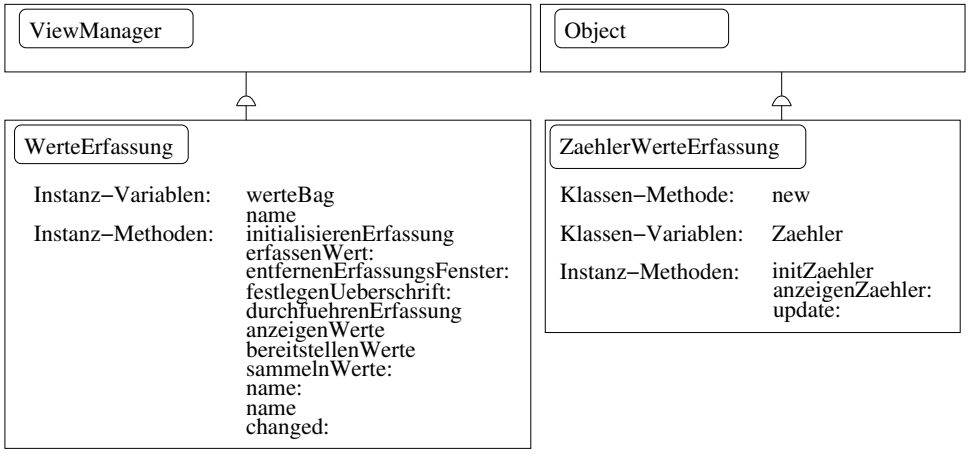


Abbildung 11.6: Die Klassen “WerteErfassung” und “ZaehlerWerteErfassung”

Um den Namen der jeweiligen Instanziierung der Klasse “WerteErfassung” anzeigen zu können, vereinbaren wir die Instanz-Variable “name” als weitere Instanz-Variable der Klasse “WerteErfassung”.

Um diesen Namen in die Instanz-Variable “name” sichern zu können, verabreden wir die Methode “name:” in der Klasse “WerteErfassung” wie folgt:

```
name: aString
name := aString
```

Damit die in “name” enthaltene Zeichenkette bereitgestellt werden kann, vereinbaren wir die Methode “name” in der folgenden Form:

```
name
^ name
```

Die zur Auslösung der indirekten Kommunikation benötigte Instanz-Methode “changed:” vereinbaren wir in der Klasse “WerteErfassung” wie folgt:

```
changed: aParameter
 (Dependents at: self ifAbsent: [#()])
    do: [:einObjekt|einObjekt update: aParameter]
```

Hinweis: Ist – bei der Ausführung dieser Methode – das Objekt, für das die Pseudovariable “self” steht, in der globalen Variablen “Dependents” *nicht* als Key enthalten, so resultiert aus der geklammerten Anforderung “(Dependents at: self ifAbsent: [#()])” das leere Array-Literal “#()”.

Diese Vereinbarung von “changed:” stimmt mit der Vereinbarung überein, die innerhalb der Klasse “Object” für die Methode “changed:” vorliegt.

Hinweis: Die vorzunehmende Vereinbarung ist nicht redundant, da die in der Klasse “ViewManager” eingetragene Vereinbarung von “changed:” überdeckt werden muß.

Damit die Methode “changed:” bei der Erfassung eines Wertes zur Ausführung gelangt, erweitern wir die Methode “erfassenWert:” durch die Anforderung

```
self changed: self
```

so daß diese Methode jetzt die folgende Form besitzt:

```
erfassenWert: aPane
werteBag add: (self paneNamed: 'eingabeFeld') contents.
(self paneNamed: 'eingabeFeld') contents: ''.
(self paneNamed: 'eingabeFeld') setFocus.
self changed: self
```

Hinweis: Indem wir “self” als Argument der Message “changed:” aufführen, erreichen wir, daß die jeweilige Instanziierung, die die Methode “changed:” ausführt, als Argument an die Message “update:” weitergereicht wird.

Um den Zähler, der bei der Ausführung der Methode “erfassenWert:” um die Zahl “1” erhöht werden soll, zu sichern, sehen wir – als Unterklasse von “Object” – die Klasse “ZaehlerWerteErfassung” vor. In dieser Klasse vereinbaren wir – analog zu der Vereinbarung im Abschnitt 8.2 – die Klassen-Methode “new” in der Form

```
new
|varWerteErfassung|
varWerteErfassung := super new.
varWerteErfassung initZaehler.
^ varWerteErfassung
```

und die Instanz-Methode “initZaehler” wie folgt:

```
initZaehler
Zaehler isNil ifTrue: [Zaehler := 0]
```

Damit eine Zähler-Erhöhung durch die indirekte Kommunikation erfolgt, verabreden wir in der Klasse “ZaehlerWerteErfassung” die Instanz-Methode “update:” durch

```
update: aParameter
Zaehler := Zaehler + 1.
self anzeigenZaehler: aParameter
```

und die hierdurch auszulösende Anzeige des Zählerstandes in Form der Methode “anzeigenZaehler:” durch:

```
anzeigenZaehler: aParameter
Transcript cr;
   show: aParameter name;
   show: ' - Anzahl der bislang insgesamt erfaßten Werte: ';
   show: Zaehler printString
```

Stellen wir nach diesen Vereinbarungen die Anforderungen

```
WerteErfassung11 := WerteErfassung new.
WerteErfassung11 name: 'WerteErfassung11'.
WerteErfassung12 := WerteErfassung new.
WerteErfassung12 name: 'WerteErfassung12'
```

sowie die Anforderungen

```
ZaehlerAbhaengig := ZaehlerWerteErfassung new.
WerteErfassung11 addDependent: ZaehlerAbhaengig.
WerteErfassung12 addDependent: ZaehlerAbhaengig
```

so ist das Objekt "ZaehlerAbhaengig" jeweils als *abhängiges* Objekt für die *unabhängigen* Objekte "WerteErfassung11" und "WerteErfassung12" festgelegt. Diesen Sachverhalt kennzeichnet die folgende Abbildung:

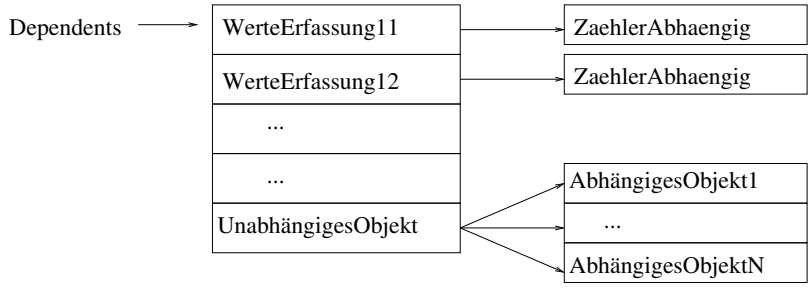


Abbildung 11.7: "Dependents" als Instanz der Klasse "Dictionary"

Hinweis: Innerhalb dieser Abbildung haben wir zusätzlich die Objekte "UnabhängigesObjekt", "AbhängigesObjekt1" und "AbhängigesObjektN" aufgeführt. Dadurch wollen wir darauf hinweisen, daß von einem unabhängigen Objekt nicht nur ein Objekt, sondern auch mehrere Objekte abhängig sein können.

Starten wir anschließend zwei parallele Erfassungsprozesse durch die beiden Anforderungen

```
WerteErfassung11 sammelnWerte: 'Jahrgangsstufe11'.
WerteErfassung12 sammelnWerte: 'Jahrgangsstufe12'
```

so wird bei der Erfassung jedes einzelnen Wertes die Methode "erfassenWert:" zur Ausführung gebracht und damit – durch die Message "changed:" mit dem Argu-

ment "self" – vom unabhängigen Objekt (einer Instanziierung der Klasse "WerteErfassung") die automatische Zustellung der Message "update:" an das abhängige Objekt "ZaehlerAbhaengig" durchgeführt. Hierdurch wird die Ausführung der Methode "anzeigenZaehler:" bewirkt, durch die die aktuelle Anzahl der insgesamt bislang erfaßten Werte – sowie der Name der jeweiligen Instanziierung des Erfassungsprozesses – im Transcript-Fenster angezeigt wird.

Hinweis: Dabei ist zu beachten, daß wir innerhalb der Methode "erfassenWert:" der Klasse "WerteErfassung" keine Angaben über die Instanz "ZaehlerAbhaengig", die den Zähler erhöht und die Methode "anzeigenZaehler:" zur gewünschten Anzeige im Transcript-Fenster ausführt, gemacht haben.

Geben wir in die beiden Erfassungsfenster z.B. jeweils einen Wert ein, so werden im Transcript-Fenster der Name des unabhängigen Objektes, der Text "– Anzahl der bislang erfaßten Werte:" und der aktuelle Wert des Attributs "Zaehler" des abhängigen Objekts z.B. in der folgenden Form angezeigt:

```
WerteErfassung11 - Anzahl der bislang erfaßten Werte: 1
WerteErfassung12 - Anzahl der bislang erfaßten Werte: 2
```

Die zuvor beschriebene Lösung läßt sich zusammenfassend wie folgt skizzieren:

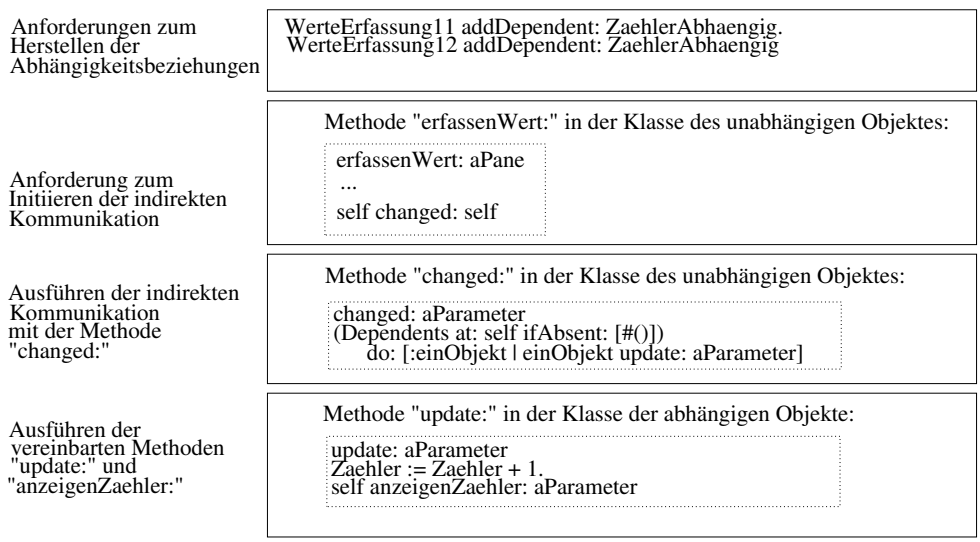


Abbildung 11.8: Indirekte Kommunikation

Kapitel 12

Fenster-Bausteine

12.1 Unterklassen der Basis-Klasse “SubPane”

Im Abschnitt 10.6 haben wir bereits darauf hingewiesen, daß sich neben den bislang von uns verwendeten Text-, Eingabe- und Schaltfeldern weitere Fenster-Bausteine – wie z.B. Listen-, Kombinations-, Options- und Kontrollfelder – innerhalb eines Rahmenfensters vereinbaren lassen.

Ähnlich dem Vorgehen, das innerhalb des Kapitels 2 beim Einsatz des Werkzeugs “WindowBuilder Pro/V” eingeschlagen wurde, können wir die jeweils gewünschte Gestaltung eines Rahmenfensters dahingehend festlegen, daß wir die Art und die Position der gewünschten Fenster-Bausteine dialog-orientiert vornehmen.

Da Fenster-Bausteine oftmals in gleicher oder ähnlicher Form einzusetzen sind, ist es wünschenswert, genauere Kenntnisse über die mit den Fenster-Bausteinen korrespondierenden Methoden zu besitzen.

Dies hat den Vorteil, daß zur Gestaltung eines Fensters nicht immer aufs Neue die gleichen Anforderungen an das Werkzeug “WindowBuilder Pro/V” gestellt werden müssen, sondern aus zuvor durchgeführten Methoden-Vereinbarungen gezielt die jeweils benötigten Anforderungen zum Aufbau einer neuen Methode übernommen werden können.

Im Hinblick auf diese Arbeitserleichterung stellen wir im folgenden ausgewählte Fenster-Bausteine und die zu deren Bearbeitung erforderlichen Methoden exemplarisch an einfachen Beispielen vor.

Grundsätzlich gilt:

- Sämtliche Fenster-Bausteine sind als Instanziierungen geeigneter Unterklassen der Basis-Klasse “SubPane” innerhalb eines Rahmenfensters einzurichten.

In dieser Hinsicht bietet der folgende Ausschnitt aus der Klassen-Hierarchie die Basis für die nachfolgenden Ausführungen:

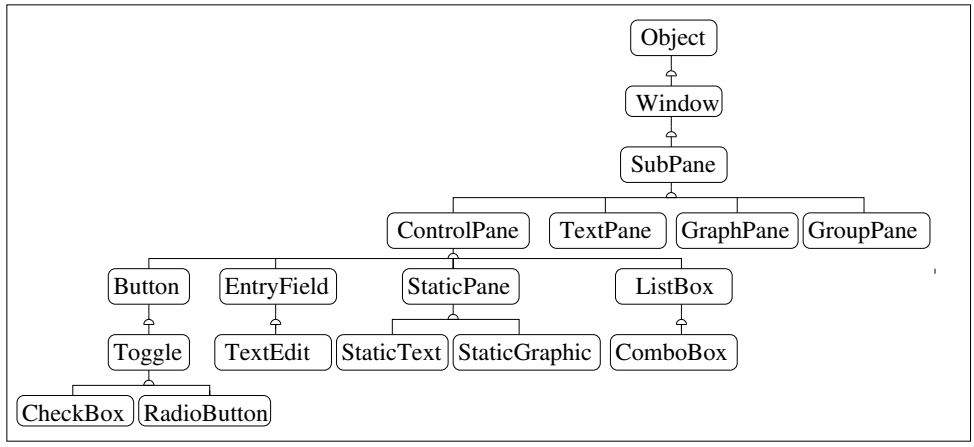


Abbildung 12.1: Ausschnitt aus der Klassen-Hierarchie

Hinweis: Bei den aufgeführten Klassen “Window” “SubPane”, “ControlPane”, “StaticPane” und “Toggle” handelt es sich um abstrakte Klassen.

Unterhalb der abstrakten Klasse “SubPane” sind die Klassen für die Vereinbarung von Instanzen standardisierter Fenster-Bausteine angesiedelt.
Im Hinblick auf die Funktion von Fenster-Bausteinen, die sich aus diesen Klassen instanziieren lassen, ist folgendes festzustellen:

- Um einen Text in einem Fenster anzuzeigen, ist ein *Textfeld* einzurichten und dazu eine Instanziierung der Klasse “StaticText” vorzunehmen.

- Zur Einrichtung eines *Schaltfeldes*, durch dessen Anklicken sich eine zugeordnete Methode zur Ausführung bringen läßt, ist eine Instanziierung der Klasse “Button” durchzuführen.

- Um ein- oder mehrzeilige *Bildschirmfelder* – als *Eingabe-* bzw. *Editierfelder* – zu vereinbaren, in denen Texte über die Tastatur eingetragen und editiert werden können, sind Instanziierungen der Klassen “EntryField” bzw. “TextEdit” durchzuführen.

- Um eine Auswahl aus zwei oder mehreren Alternativen treffen zu können, lassen sich Instanzen der Klassen “CheckBox” und “RadioButton” verwenden.

  Um eine Auswahl aus genau zwei sich ausschließenden Möglichkeiten treffen zu können, eignet sich die Anzeige eines *Kontrollfeldes*, das als Instanz der Klasse “CheckBox” zu erstellen ist.
  Ein Kontrollfeld läßt sich durch einen Mausklick aktivieren und durch einen weiteren Mausklick wieder deaktivieren.

  Gibt es zwei oder mehrere mögliche Alternativen, von denen genau *eine* Alternative ausgewählt werden muß, empfiehlt sich der Einsatz von *Optionsfeldern*, die als Instanzen der Klasse “RadioButton” einzurichten sind.
  Dabei ist dafür Sorge zu tragen, daß durch einen Mausklick jeweils genau ein

Optionsfeld aus der Gruppe der zueinander gehörenden Optionsfelder aktiviert werden kann.

- Um ein *Listenfeld* einzurichten, in dem sich einzelne Listenelemente durch einen Mausklick auswählen lassen, eignen sich Instanzen der Klasse "ListBox". Sofern darüberhinaus auch eine Auswahl möglich sein soll, die über eine Tastatureingabe festgelegt werden kann, sind *Kombinationsfelder* – als Kombination aus Listenfeld und Eingabefeld – als Instanzen der Klasse "ComboBox" zu verwenden.

- Um Fenster-Bausteine zu gruppieren, müssen *Gruppenfelder* als Instanzen der Klasse "GroupPane" eingerichtet und diesen die jeweils zusammengehörenden Fenster-Bausteine zugeordnet werden.

Hinweis: Um Grafiken darzustellen und bearbeiten zu können, sind geeignete Instanzen der Klasse "GraphPane" und "StaticGraphic" zu verwenden (zum Einsatz eines "GraphPane" siehe Abschnitt 15.3).
Desweiteren kann ein Fenster unter Einsatz einer Instanziierung der Basis-Klasse "TextPane" zur Eingabe und Editierung umfangreicher Texte eingerichtet werden.

12.2 Klassen-übergreifende Methoden

Damit ein Fenster-Baustein geeignet auf ein Ereignis reagieren kann, muß für ihn ein *Eigentümer* – durch den Einsatz der Basis-Methode "owner:" – festgelegt werden.

- **"owner:"**:
  Dem Fenster-Baustein, der als Empfänger-Objekt der Message "owner:" aufgeführt ist, wird die als Argument von "owner:" angegebene Instanz als *Eigentümer* zugeordnet.

Die jeweilige Methode, deren Ausführung – durch die Message "when:perform:" – mit einem Ereignis verknüpft worden ist, wird innerhalb der Klasse gesucht, aus der der *Eigentümer* instanziiert wurde.

Als weitere Basis-Methoden, die sich für unterschiedliche Fenster-Bausteine einsetzen lassen, sind zu nennen:

- **"contents:"**:
  Bei der Ausführung der Basis-Methode "contents:" wird dem als Empfänger-Objekt der Message "contents:" aufgeführten Fenster-Baustein das als Zeichenkette angegebene Argument zugeordnet. Empfänger-Objekte können z.B. Instanzen der Klassen "StaticText", "Button", "TextEdit", "EntryField", "CheckBox", "ListBox" und "ComboBox" sein.

- **"contents"**:
  Als Ergebnis-Objekt der Message "contents" wird diejenige Zeichenkette ermittelt, die dem als Empfänger-Objekt aufgeführten Fenster-Baustein zugeordnet ist. Empfänger-Objekte sind z.B. Instanzen der Klassen "StaticText", "Button", "TextEdit", "EntryField", "CheckBox", "ListBox" und "ComboBox".

- **"setFocus"**
  Der Fenster-Baustein, der durch das Empfänger-Objekt der Message "setFocus" bestimmt ist, wird am Bildschirm aktiviert, sofern das zugehörige View vorher eröffnet wurde.

12.3 Ausgewählte Fenster-Bausteine

12.3.1 Rahmenfenster

Bei der Vereinbarung eines oder mehrerer Fenster-Bausteine ist festzulegen, auf welche Ereignisse die einzelnen Fenster-Bausteine reagieren können sollen. Die jeweiligen Ereignisnamen sind mittels der Message "when:perform:" mit geeigneten Methoden zu verbinden, die beim Eintritt der jeweiligen Ereignisse zur Ausführung gebracht werden sollen.
Im folgenden werden wir einen summarischen Überblick geben. Dabei beziehen wir uns bei den nachfolgenden Beispielen grundsätzlich auf ein Rahmenfenster, das wir durch die Ausführung der Anforderungen

```
Fenster := FensterBausteine new.
Fenster vereinbarungRahmenfenster
```

als View aufgebaut haben. Nachdem wir in dieses View einen oder mehrere geeignete Fenster-Bausteine integriert haben, läßt es sich anschließend durch die Anforderung

```
Fenster openWindow
```

eröffnen und als Fenster am Bildschirm anzeigen.
Dabei gehen wir davon aus, daß die Methode "vereinbarungRahmenfenster" wie folgt in der als Unterklasse der Basis-Klasse "ViewManager" vereinbarten Klasse "FensterBausteine" eingerichtet wurde:

```
vereinbarungRahmenfenster
self addView:
 (self topPaneClass new
  owner: self;
  pStyle: #(sysmenu titlebar minimize maximize sizable);
  framingRatio:((Rectangle leftTopUnit rightAndDown:1/2@(1/2))
                              extentFromLeftTop:1/2@(1/2)))
```

Hinweis: Auf dieser Basis geben wir in der nachfolgenden Darstellung die jeweils erforderlichen Messages an, die an die globale Variable "Fenster" zu richten sind, damit die gewünschten Fenster-Bausteine eingerichtet werden.
Wir machen grundsätzlich keine präzise Angabe darüber, an welcher Position der jeweilige Fenster-Baustein im Rahmenfenster plaziert sein soll, sondern setzen stets die Message "framingRatio:" in der Form

```
framingRatio: ( ... )
```

ein. Wie das jeweilige Argument dieser Message lauten muß, ist individuell durch die betreffende Anwendung – auf der Basis der Angaben des Abschnitts 10.8 – zu bestimmen.

12.3.2 Textfelder

Soll im Rahmenfenster z.B. der Text "Wert:" innerhalb eines *Textfeldes* angezeigt werden, so ist eine Instanz der Basis-Klasse "StaticText" einzurichten. Sofern für dieses Textfeld der Name "staticText" gewählt wird, läßt sich dies durch die Ausführung der Methode

```
vereinbarungStaticText
self addSubpane:
  (StaticText new owner: self;
     framingRatio:( ... );
     rightJustified;
     paneName: 'staticText';
     contents: 'Wert:')
```

bewirken. Dazu ist die Anforderung

```
Fenster vereinbarungStaticText
```

zu stellen.

Hinweis: Durch den Einsatz der Methode "rightJustified" ist bestimmt, daß die Zeichenkette 'Wert:' – im Bereich des Textfeldes "staticText" – rechtsbündig angezeigt wird.

12.3.3 Schaltfelder

Soll im Rahmenfenster ein *Schaltfeld* z.B. mit der Aufschrift "OK" festgelegt werden, so ist eine Instanziierung der Basis-Klasse "Button" durch die Methode

```
vereinbarungOKButton
self addSubpane:
  (Button new owner: self;
     framingRatio:( ... );
     contents: 'OK';
     when: #clicked perform: #ausfuehrenOKButton:)
```

vorzunehmen. Dies läßt sich durch die folgende Anforderung abrufen:

```
Fenster vereinbarungOKButton
```

Innerhalb der Methode "vereinbarungOKButton" ist festgelegt, daß die Methode "ausfuehrenOKButton:" zur Ausführung gelangt, sofern das Ereignis "#clicked" – durch einen Mausklick auf das Schaltfeld "OK" – ausgelöst wird.

Hinweis: Innerhalb der Methode "ausfuehrenOKButton:" ist festzulegen, welche Methoden – im Hinblick auf den jeweiligen Lösungsplan – ausgeführt werden sollen. Zum Test kann die Methode "ausfuehrenOKButton:" z.B. wie folgt vereinbart werden:

```
ausfuehrenOKButton: anOKButton
Transcript cr; show: 'OK-Schaltfeld betaetigt'
```

12.3.4 Eingabe- und Editierfelder

Ein Eingabefeld ist als Instanz der Basis-Klasse “EntryField” und ein Editierfeld als Instanz der Basis-Klasse “TextEdit” einzurichten.

Eingabefelder

Soll im Rahmenfenster ein *einzeiliger* Text eingegeben werden können, so ist eine Instanziierung der Klasse “EntryField” durchzuführen.
Sofern dieses Eingabefeld durch den Namen “entryField” gekennzeichnet werden soll, läßt sich dessen Einrichtung durch die Anforderung

```
Fenster vereinbarungEntryField
```

zur Ausführung der folgenden Methode bewirken:

```
vereinbarungEntryField
self addSubpane:
  (EntryField new owner: self;
     framingRatio:( ... );
     paneName: 'entryField';
     when: #textChanged perform: #verarbeitenEntryField:)
```

Durch die Message “when:perform:” ist gesichert, daß die Methode “verarbeitenEntryField:” ausgeführt wird, sofern das Ereignis “#textChanged” eintritt. Dies ist z.B. dann der Fall, wenn der Inhalt des Eingabefeldes durch eine Tastatureingabe geändert und danach ein anderer Fenster-Baustein des Rahmenfensters aktiviert wird.
Hinweis: Innerhalb der Methode “verarbeitenEntryField:” ist festzulegen, welche Methoden – im Hinblick auf den jeweiligen Lösungsplan – ausgeführt werden sollen. Zum Test kann die Methode “verarbeitenEntryField:” z.B. wie folgt vereinbart werden:

```
verarbeitenEntryField: anEntryField
Transcript cr; show: (self paneNamed: 'entryField') contents
```

Editierfelder

Soll im Rahmenfenster die Möglichkeit bestehen, einen *mehrzeiligen* Text in ein Editierfeld eingeben zu können, so ist eine Instanz der Klasse “TextEdit” erforderlich. Soll das Editierfeld den Namen “textEdit” tragen, so kann die Einrichtung dieses Feldes durch die Methode

```
vereinbarungTextEdit
self addSubpane:
  (TextEdit new owner: self;
     framingRatio:( ... );
     paneName: 'textEdit';
```

```
when: #textChanged perform: #verarbeitenTextEdit:)
```

erreicht werden, deren Ausführung sich durch die Anforderung

```
Fenster vereinbarungTextEdit
```

abrufen läßt.
In dieser Vereinbarung bestimmt die Message "when:perform:", daß beim Eintreten des Ereignisses "#textChanged" die Methode "verarbeitenTextEdit:" ausgeführt wird. Dies ist z.B. dann der Fall, wenn der Inhalt des Editierfeldes durch eine Tastatureingabe geändert und danach ein anderer Fenster-Baustein des Rahmenfensters aktiviert wird.
Hinweis: Innerhalb der Methode "verarbeitenTextEdit:" ist festzulegen, welche Methoden – im Hinblick auf den jeweiligen Lösungsplan – ausgeführt werden sollen. Zum Beispiel kann die Methode "verarbeitenTextEdit:" wie folgt vereinbart werden:

```
verarbeitenTextEdit: aTextEdit
Transcript cr; show: (self paneNamed: 'textEdit') contents
```

12.3.5 Kontroll- und Optionsfelder

Ein Kontrollfeld ist als Instanz der Basis-Klasse "CheckBox" und ein Optionsfeld als Instanz der Basis-Klasse "RadioButton" einzurichten.
Ein Kontrollfeld kann system-seitig die sich ausschließenden Zustände "aktiviert" bzw. "nicht aktiviert" annehmen. Der Wechsel läßt sich über einen Mausklick vornehmen.
Der Einsatz zweier Kontrollfelder ist z.B. in der Situation angebracht, in der die folgenden Anforderungen über einen Mausklick abrufbar sein sollen:

- Allein die erfaßten Werte sollen angezeigt werden (Methode "anzeigenWerte").
- Allein der Durchschnittswert der erfaßten Werte soll ausgegeben werden (Methode "anzeigenDurchschnittswert").
- Es soll sowohl die Anzeige der erfaßten Werte als auch des Durchschnittswertes erfolgen (Methoden "anzeigenWerte" und "anzeigenDurchschnittswert").

Somit bietet sich z.B. die folgende Anzeige innerhalb eines Rahmenfensters an:

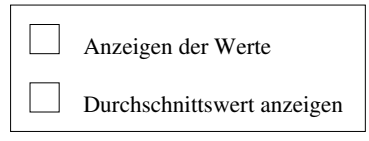


Abbildung 12.2: Beispiel für Kontrollfelder

Anders ist der Sachverhalt, sofern die Möglichkeit bestehen soll, über einen Mausklick die folgenden Leistungen anfordern zu können:

- Die Werte sollen nur für die Jahrgangsstufe 11 erfaßt werden (Methode "sammelnWerte:").
- Die Werte sollen sowohl für die Jahrgangsstufe 11 als auch für die Jahrgangsstufe 12 erfaßt werden (zweimalige Ausführung der Methode "sammelnWerte:").

In diesem Fall ist es sinnvoll, zwei Optionsfelder zu verwenden, so daß sich z.B. die folgende Anzeige innerhalb eines Rahmenfensters anbietet:

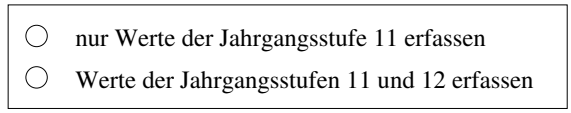


Abbildung 12.3: Beispiel für Optionsfelder

Bei diesem Sachverhalt schließen sich die beiden möglichen Anforderungen inhaltlich gegenseitig aus, so daß nur eine einzige Möglichkeit sinnvollerweise aktiviert sein kann. Welches der beiden Optionsfelder vorab aktiviert sein soll, ist durch die Ausführung einer geeigneten Methode zu bestimmen. Entsprechend ist dafür zu sorgen, daß das aktivierte Optionsfeld deaktiviert wird, sofern das nicht aktivierte Optionsfeld durch einen Mausklick aktiviert wird.

Kontrollfelder

Da Kontrollfelder als Instanzen der Basis-Klasse "CheckBox" einzurichten sind, ist z.B. die folgende Methode – im Hinblick auf die oben angegebene Vorgabe – zur Ausführung zu bringen:

```
vereinbarungCheckBoxen
self addSubpane:
       (CheckBox new owner: self;
          framingRatio:( ... );
          paneName: 'checkBox1';
          contents: 'Anzeigen der Werte');
     addSubpane:
       (CheckBox new owner: self;
          framingRatio:( ... );
          paneName: 'checkBox2';
          contents: 'Durchschnittswert anzeigen')
```

Hierzu ist die folgende Anforderung auszuführen:

```
Fenster vereinbarungCheckBoxen
```

Sind die jeweils gewünschten Anforderung(en) durch geeignete Mausklicks auf die beiden Kontrollfelder bestimmt worden, so kann die getroffene Wahl durch die

Ausführung der folgenden Methode umgesetzt werden:

```
auswertungCheckBoxen: anOKButton
(self paneNamed: 'checkBox1') selection
   ifTrue: [WerteErfassung11 anzeigenWerte].
(self paneNamed: 'checkBox2') selection
   ifTrue: [WerteErfassung11 anzeigenDurchschnittswert]
```

Hinweis: Hierbei setzen wir voraus, daß der Erfassungsprozeß durch die globale Variable "WerteErfassung11" gekennzeichnet ist, wobei "WerteErfassung11" auf eine Instanz der Klasse "InWerteErfassung" weist (siehe Abschnitt 5.3) und bereits die Message "WerteErfassung11 zentrum" – zur erstmaligen Berechnung des Durchschnittswertes – zugestellt wurde.
Die Ausführung der Methode "auswertungCheckBoxen:" können wir z.B. mit dem Ereignis "#clicked" verknüpfen, wobei dieses Ereignis einem zusätzlich eingesetzten Schaltfeld zugeordnet ist.

Welche Kontrollfelder aktiviert sind, ist durch den Einsatz der Methode "selection" zu prüfen.

- **"selection"**:
  Als Ergebnis-Objekt der Message "selection" resultiert die Pseudovariable "true", sofern das als Empfänger-Objekt aufgeführte Kontrollfeld aktiviert ist. Andernfalls ergibt sich die Pseudovariable "false" als Ergebnis-Objekt.

Optionsfelder

Um die oben festgelegte Vorgabe zur Anzeige der beiden Optionsfelder mit den Texten "nur Werte der Jahrgangsstufe 11 erfassen" und "Werte der Jahrgangsstufen 11 und 12 erfassen" umzusetzen, müssen geeignete Instanziierungen der Basis-Klasse "RadioButton" vorgenommen werden. Hierzu kann z.B. die folgende Methode durch die Anforderung

```
Fenster vereinbarungRadioButtons
```

zur Ausführung gebracht werden:

```
vereinbarungRadioButtons
self addSubpane:
 (RadioButton new owner: self;
   framingRatio:( ... );
   paneName: 'radioButton1';
   contents: 'nur Werte der Jahrgangsstufe 11 erfassen';
   when: #clicked perform: #umschalten1:);
 addSubpane:
 (RadioButton new owner: self;
   framingRatio:( ... );
```

```
   paneName: 'radioButton2';
   contents: 'Werte der Jahrgangsstufen 11 und 12 erfassen';
   when: #clicked perform: #umschalten2:)
```

Sofern die jeweils gewünschte Anforderung durch einen Mausklick auf eines der beiden Optionsfelder bestimmt wird, ist dafür zu sorgen, daß nur das jeweils angeklickte Optionsfeld aktiviert ist.

Um dieser Forderung Rechnung zu tragen, ist dem Ereignis "#clicked" durch die Message "when:perform:" für das zuerst festgelegte Optionsfeld die Methode "umschalten1:" und für das zweite Optionsfeld die Methode "umschalten2:" zugeordnet.

Damit diese beiden Methoden gewährleisten, daß nur das jeweils angeklickte Optionsfeld aktiviert ist, können diese beiden Methoden z.B. wie folgt vereinbart werden:

```
umschalten1: aRadioButton
(self paneNamed: 'radioButton1') selection: true.
(self paneNamed: 'radioButton2') selection: false

umschalten2: aRadioButton
(self paneNamed: 'radioButton1') selection: false.
(self paneNamed: 'radioButton2') selection: true
```

Die jeweilige Umschaltung von "aktiviert" in "nicht aktiviert" bzw. umgekehrt wird durch den Einsatz der Methode "selection:" sichergestellt.

- **"selection:"**:
  Das als Ergebnis-Objekt der Message "selection:" aufgeführte Optionsfeld wird aktiviert (deaktiviert), sofern die Pseudovariable "true" ("false") als Argument dieser Message angegeben wird.

Hinweis: Der Einsatz der Message "selection:" ist in dieser Situation nicht notwendig, da beim Einsatz mehrerer Optionsfelder das Aktivieren des einen Optionsfeldes und das Deaktivieren des anderen Optionsfeldes *automatisch* erfolgt.

Ist die jeweils gewünschte Anforderung durch einen geeigneten Mausklick formuliert worden, so kann die getroffene Wahl durch die Ausführung der Methode

```
auswertungRadioButtons: anOKButton
(self paneNamed: 'radioButton1') selection
   ifTrue: [WerteErfassung11 := WerteErfassung new.
            WerteErfassung11 sammelnWerte: '11'].
(self paneNamed: 'radioButton2') selection
   ifTrue: [WerteErfassung11 := WerteErfassung new.
            WerteErfassung11 sammelnWerte: '11'.
            WerteErfassung12 := WerteErfassung new.
            WerteErfassung12 sammelnWerte: '12']
```

umgesetzt werden.

Hinweis: Die Ausführung der Methode “auswertungRadioButtons:” können wir z.B. mit dem Ereignis “#clicked” eines zusätzlichen Schaltfeldes mit der Aufschrift “OK” verknüpfen, wobei diesem Ereignis ein zusätzlich eingesetztes Schaltfeld mit der Aufschrift “OK” zugeordnet ist.

12.3.6 Listen- und Kombinationsfelder

Ein Listenfeld ist als Instanz der Basis-Klasse “ListBox” und ein Kombinationsfeld als Instanz der Basis-Klasse “ComboBox” einzurichten.

Um aus mehreren Zeichenketten genau eine Zeichenkette durch einen Mausklick auswählen zu können, lassen sich sowohl Listen- als auch Kombinationsfelder als Fenster-Bausteine einsetzen.

Bei beiden Formen werden Zeichenketten – als Listen-Items – untereinander auf dem Bildschirm angezeigt, so daß das jeweils gewünschte Listen-Item durch das Klicken mit der linken Maustaste bestimmt werden kann.

Der Unterschied der beiden Bausteine besteht darin, daß bei einem Kombinationsfeld die Listen-Items dadurch sichtbar werden, daß mit der Maus auf die angezeigte Pfeilfläche geklickt wird. Sichtbar ist zunächst allein das Kombinations-Eingabefeld (mit einer durch eine Voreinstellung bestimmten Zeichenkette), in das eine beliebige Zeichenkette über die Tastatur eingetragen werden kann. Dies ist eine Ergänzung der für ein Listenfeld zur Verfügung stehenden Möglichkeiten, da eine Zeichenkette gewählt werden kann, die nicht als Listen-Item innerhalb der angezeigten Liste enthalten ist. Ist eine Zeichenkette über die Tastatur eingetragen bzw. durch einen Mausklick aus den Listen-Items ausgewählt worden, so ist sie im Kombinations-Eingabefeld enthalten und steht zur Abfrage zur Verfügung. Der Einsatz eines Kombinationsfeldes ist z.B. dann nützlich, wenn die Anzahl der auswählbaren Zeichenketten so groß ist, daß die Auswahl durch eine Tastatureingabe abgekürzt werden kann. Der Einsatz ist dann unumgänglich, wenn einerseits – aus Platzgründen – nur die gebräuchlichsten Zeichenketten als Listen-Items zur Auswahl vorgehalten werden, andererseits auch ein seltener Auswahlwunsch realisierbar sein soll.

Als Beispiel für die Einrichtung eines Listenfeldes bzw. eines Kombinationsfeldes betrachten wir die Situation, in der die Art der Datenerfassung über die Zeichenketten ’11’ und ’12’ festgelegt werden soll, so daß die Felder in der Form

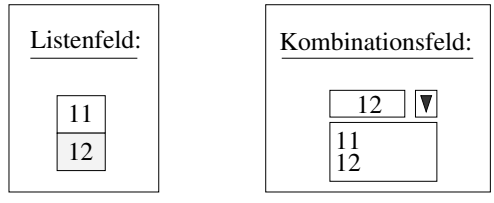


Abbildung 12.4: Beispiel für Listen- und Kombinationsfelder

zu vereinbaren sind. In dieser Situation soll eine Erfassung für die Jahrgangsstufe 11 mit der Auswahl des Listen-Items “11” und eine Erfassung für die Jahrgangsstufe 12 mit der Auswahl des Listen-Items “12” verbunden sein.

Es wird unterstellt, daß das Listen-Item “12” voreingestellt sein soll. Dies bedeutet, daß die Zeichenkette ’12’ bei einem Listenfeld in markierter Form angezeigt und bei einem Kombinationsfeld im Kombinations-Eingabefeld eingetragen ist.
Da bei einem Kombinationsfeld eine beliebige Tastatureingabe – z.B. in Form der Zeichenkette ’13’ – möglich ist, kann über ein Kombinationsfeld z.B. auch eine Erfassung für die Werte der Jahrgangsstufe 13 angefordert werden.

Listenfelder

Um das Listenfeld mit dem Text “11” und “12” zu vereinbaren, ist eine geeignete Instanziierung der Basis-Klasse “ListBox” vorzunehmen. Hierzu kann z.B. die Methode

```
vereinbarungListBox
self addSubpane:
  (ListBox new owner: self;
     framingRatio:( ... );
     paneName: 'listBox';
     insertItem: '11';
     insertItem: '12';
     selection: '12')
```

in der Form

```
Fenster vereinbarungListBox
```

zur Ausführung gebracht werden.
Um die beiden Zeichenketten ’11’ und ’12’ innerhalb der Liste als Listen-Items aufzuführen, wird die Methode “insertItem:” verwendet.

- **“insertItem:”**:
  Durch die Message “insertItem:” wird das als Empfänger-Objekt der Message “insertItem:” aufgeführte Listenfeld um dasjenige Listen-Item ergänzt, das durch die als Argument angegebene Zeichenkette bestimmt ist.

Hinweis: Bei einem Listenfeld werden die Listen-Items in einem geordneten Sammler gespeichert, auf den über die Instanz-Variable “list” verwiesen wird.
Sollen sämtliche Listen-Items durch neue Listen-Items – z.B. durch “12” und “13” – ersetzt werden, so läßt sich dies durch die Anforderung

```
Fenster ersetzenItemsListBox
```

erreichen, sofern die Methode “ersetzenItemsListBox” in der folgenden Form vereinbart wird:

```
ersetzenItemsListBox
|liste|
liste := OrderedCollection new.
liste add: '11'; add: '12'; add: '13'.
(self paneNamed: 'listBox') contents: liste
```

Um eine Markierung für das Listen-Item “12” zu bewirken, ist die Basis-Methode “selection:” einzusetzen.

- **“selection:”**:
  In dem als Empfänger-Objekt der Message “selection:” aufgeführten Listenfeld wird dasjenige Listen-Item markiert, das als Argument dieser Message in Form einer Zeichenkette angegeben wird.

Um festzustellen, welches Listen-Item zuvor durch einen Mausklick ausgewählt wurde, läßt sich die Methode “selectedItem” einsetzen.

- **“selectedItem”**:
  Als Ergebnis-Objekt der Message “selectedItem” resultiert diejenige Zeichenkette, die zuvor als Listen-Item des als Empfänger-Objekt aufgeführten Listenfeldes durch einen Mausklick markiert wurde.

Ist das durch die Voreinstellung bzw. durch einen Mausklick gewünschte Listen-Item bestimmt worden, so kann anschließend z.B. die Ausführung der Methode “auswahlListBox:” vorgenommen werden, die wie folgt vereinbart ist:

```
auswahlListBox: anOKButton
| var |
var := (self paneNamed: 'listBox') selectedItem.
Smalltalk at: ('WerteErfassung',var) asSymbol
          put: (WerteErfassung new).
(Smalltalk at: ('WerteErfassung',var) asSymbol)
   sammelnWerte: 'Jahrgangstufe ',var
```

Nachdem die Zeichenkette ’11’ bzw. ’12’ der lokalen Variablen “var” zugeordnet wurde, wird durch die Methode “,” aus der Zeichenkette ’WerteErfassung’ und dem jeweils gewählten Listen-Item ein Symbol zur Kennzeichnung der globalen Variablen “WerteErfassung11” bzw. “WerteErfassung12” gebildet.
Durch die Message “at:put:” wird die globale Variable “WerteErfassung11” bzw. “WerteErfassung12” in das System-Dictionary “Smalltalk” eingetragen und dieser Variablen eine Instanz der Klasse “WerteErfassung” zugeordnet.
Abschließend wird der eingerichteten globalen Variablen die Message “sammelnWerte:” mit dem zugehörigen Argument geschickt, durch die die Titel-Zeile des Erfassungsfensters bestimmt ist.
Die Ausführung der Methode “auswahlListBox:” können wir mit dem Ereignis “#clicked” verknüpfen, wobei dieses Ereignis einem zusätzlich eingerichteten Schaltfeld zugeordnet ist.

Kombinationsfelder

Soll ein Kombinationsfeld mit den Texten “11” und “12” als Listen-Items vereinbart und das Listen-Item “12” im Kombinations-Eingabefeld als Voreinstellung eingetragen werden, so ist eine geeignete Instanziierung der Basis-Klasse “ComboBox” vorzunehmen. Hierzu läßt sich durch die Anforderung

```
Fenster vereinbarungComboBox
```

die folgende Methode ausführen:

```
vereinbarungComboBox
self addSubpane:
  (ComboBox new owner: self;
    framingRatio: ( ... );
    paneName: 'comboBox';
    insertItem: '11';
    insertItem: '12';
    selection: '12')
```

Hinweis: Die beiden Methoden "insertItem:" und "selection:" haben für Kombinationsfelder dieselbe Wirkung wie für Listenfelder.

Die im Kombinations-Eingabefeld enthaltene Zeichenkette läßt sich durch den Einsatz der Methode "text" bestimmen.

- **"text"**:
  Als Ergebnis-Objekt der Message "text" resultiert diejenige Zeichenkette, die im Kombinations-Eingabefeld des als Empfänger-Objekt aufgeführten Kombinationsfeldes enthalten ist.

Auf der Basis der ermittelten Zeichenkette erfolgt die Anzeige des gewünschten Erfassungsfensters.
Ist die gewünschte Zeichenkette durch die Voreinstellung, durch einen Mausklick oder eine Tastatureingabe festgelegt worden, so können wir dies z.B. mit dem Ereignis "#clicked" verknüpfen oder aber durch die Anforderung

```
Fenster auswahlComboBox
```

die folgende Methode ausführen lassen:

```
auswahlComboBox
| var |
var := (self paneNamed: 'comboBox') text.
Smalltalk at: ('WerteErfassung',var) asSymbol
          put: (WerteErfassung new).
(Smalltalk at: ('WerteErfassung',var) asSymbol)
   sammelnWerte: 'Jahrgangstufe ',var
```

Hinweis: Die in der Methode "auswahlComboBox" – nach der 1. Anforderung – enthaltenen Anforderungen sind identisch mit den in der oben beschriebenen Methode "auswahlListBox:" aufgeführten Anforderungen.
Genau wie bei einem Listenfeld werden auch bei einem Kombinationsfeld alle Listen-Items in einem geordneten Sammler gespeichert, auf den über die Instanz-Variable "list" verwiesen wird.

Sollen sämtliche Listen-Items durch neue Listen-Items – z.B. durch '12' und '13' – ersetzt werden, so kann dies durch die Ausführung der Anforderung

```
Fenster ersetzenItemsComboBox
```

mit der in der Form

```
ersetzenItemsComboBox
|liste|
liste := OrderedCollection new.
liste add: '11'; add: '12'; add: '13'.
(self paneNamed: 'comboBox') contents: liste
```

vereinbarten Methode geschehen.

12.3.7 Gruppenfelder

Sollen bestimmte Fenster-Bausteine wie z.B. mehrere Optionsfelder gruppiert werden, so daß deren Zusammengehörigkeit durch eine Umrahmung optisch auf dem Fenster erkennbar ist, müssen sie durch die Message "addSubPane:" als Bausteine einem *Gruppenfeld* zugeordnet werden, das zuvor als Instanz der Basis-Klasse "GroupPane" vereinbart wurde.

Um die in Abschnitt 12.3.5 angegebenen Optionsfelder zu einer Gruppe zusammenzufassen, können wir z.B. die Methode "vereinbarungGroupPane" in der folgenden Form verwenden:

```
vereinbarungGroupPane
|gruppe|
self addSubpane: ( gruppe := GroupPane new
                                framingRatio:( ... ) ).
gruppe addSubpane:
  (RadioButton new owner: self;
  framingRatio:( ... );
  paneName: 'radioButton1';
  contents: 'nur Werte der Jahrgangsstufe 11 erfassen';
  when: #clicked perform: #umschalten1:);
addSubpane:
(RadioButton new owner: self;
  framingRatio:( ... );
  paneName: 'radioButton2';
  contents: 'Werte der Jahrgangsstufen 11 und 12 erfassen';
  when: #clicked perform: #umschalten2:)
```

Bei der Ausführung dieser Methode wird zunächst ein Gruppenfeld als Instanz der Klasse "GroupPane" eingerichtet und der temporären Variablen "gruppe" zugeordnet. Anschließend werden die beiden gruppierten Optionsfelder durch die Message "addSubpane:" als gruppierte Fenster-Bausteine verabredet.

12.4 Vereinbarung des Erfassungsfensters

Im Abschnitt 2.2 haben wir erläutert, wie sich eine Methode ("createViews") durch den Einsatz des Werkzeugs "WindowBuilder Pro/V" erzeugen läßt, durch die der Aufbau des von uns gewünschten Erfassungsfensters beschrieben wird. Die automatisch generierte Methode haben wir – den Erfordernissen entsprechend – im Abschnitt 2.4 geeignet verändert. Zusätzlich haben wir dieser Methode den von uns im Lösungsplan konzipierten Namen "initialisierenErfassung" gegeben.

Auf der Basis der in diesem Kapitel und im Kapitel 10 erlangten Kenntnisse über den Aufbau eines Rahmenfensters und der Möglichkeit, geeignete Fenster-Bausteine in das Rahmenfenster zu integrieren, können wir an dieser Stelle ohne den Einsatz des Werkzeugs "Window Builder Pro/V" auskommen und die Methode "initialisierenErfassung" wie folgt vereinbaren:

```
initialisierenErfassung
werteBag := Bag new.
self addView: (self topPaneClass new owner: self;
  labelWithoutPrefix:  ' ';
  noSmalltalkMenuBar;
  viewName: 'erfassungsfenster';
  framingRatio: ( (Rectangle leftTopUnit
    rightAndDown:1/2@(1/2)) extentFromLeftTop:1/2@(1/2) );
 addSubpane: (StaticText new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:1/8@(2/8)) extentFromLeftTop:2/8@(1/8) );
     contents: 'Wert:');
 addSubpane: (EntryField new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/8) );
     paneName: 'eingabeFeld');
 addSubpane: (Button new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:2/8@(6/8)) extentFromLeftTop:2/8@(1/8) );
     contents: 'erfasse';
     when: #clicked perform: #erfassenWert:);
 addSubpane: (Button new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:5/8@(6/8)) extentFromLeftTop:2/8@(1/8) );
     contents: 'Erfassungsende';
     when: #clicked perform: #entfernenErfassungsfenster:)
             )
```

Hinweis: Wie bereits in den Abschnitten 10.5 und 10.8 erläutert, geben wir die Plazierung eines Fenster-Bausteins stets durch den Einsatz der Methode "framingRatio:" zusammen mit den Messages "leftTopUnit", "rightAndDown:" und "extentFromLeftTop:" an. Im Vergleich mit der im Anhang A.1 angegebenen Methode "createViews" setzen wir nicht die vom "WindowBuilder Pro/V" generierten Messages "iDUE:", "xC", "yC" und "cRDU:"

ein.
Es ist ebenfalls erkennbar, daß wir die vom “WindowBuilder Pro/V” erzeugten Messages “startGroup” und “pStyle:” sowie die temporäre Variable “‘v” nicht verwendet haben. Ebenso haben wir bei den Anforderungen zur Instanziierung des Rahmenfensters und der einzelnen Fenster-Bausteine auf den Einsatz der Message “yourself” verzichtet.

Kapitel 13

Das Model/View-Konzept

13.1 Das Grundprinzip

Zur Lösung unserer bisherigen Problemstellungen haben wir die Größen, die innerhalb eines Lösungsplans bedeutungsvoll waren, durch geeignete Objekte – als Träger der problem-spezifischen Daten – modelliert und auf dieser Basis die jeweiligen Klassen-Vereinbarungen zur Instanziierung dieser Objekte entwickelt.
Zum Beispiel wurde von uns im Hinblick auf die Zielsetzung, eine didaktisch orientierte Entwicklung von Lösungsplänen vorzustellen, die den Grundsätzen des objektorientierten Programmierens genügen, zur Lösung der Problemstellung PROB-5 die folgende Klassenhierarchie entwickelt (siehe Abschnitt 7.4):

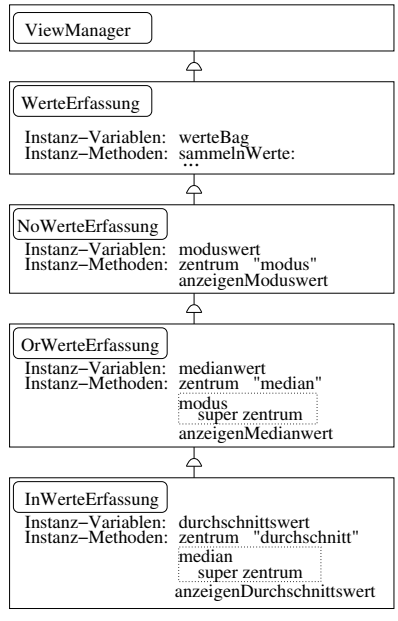


Abbildung 13.1: Klassen zur Lösung von PROB-5

Diese Gliederung haben wir dem Lösungsplan deswegen zugrundegelegt, weil die Thematik der Vererbung in Verbindung mit den Mechanismen des Polymorphismus im Vordergrund des Interesses stand.
Wollen wir diese didaktisch-orientierte Zielsetzung aufgeben, können wir einen anders strukturierten Lösungsplan entwickeln.
Unter dem Aspekt der Wiederverwendbarkeit von Software-Komponenten liegt es nahe, eine *Zweiteilung* der zur Lösung zu entwickelnden Klassen vorzunehmen. Dabei sollten die zu verarbeitenden Daten und die zur jeweiligen *Zentrums-Berechnung* benötigten Methoden strikt von denjenigen Daten und denjenigen Methoden getrennt werden, die zur *Kommunikation* zwischen dem Anwender und dem SMALLTALK-System – auf der Basis von geeigneten Bildschirmoberflächen – benötigt werden.

Bei der professionellen Programmierung sollte die Gliederung von Lösungsplänen grundsätzlich nach den Prinzipien des *Model/View-Konzeptes* vorgenommen werden, d.h. es sollte eine Gliederung in die beiden folgenden Teile angestrebt werden:

- In den *Model-Teil* (Modell-Teil), in dem die Angaben zur Modellierung der Daten und die algorithmische Verarbeitung der Daten, d.h. diejenigen Methoden zu vereinbaren sind, mit denen die im Sinne der Problemstellung gewünschte Verarbeitung der Daten durchgeführt werden soll.
- In den *View-Teil* (Sicht-Teil), der allein diejenigen Methoden und Daten enthält, die für den Aufbau und den Dialog mittels grafischer Benutzeroberflächen – zur Interaktion zwischen dem Anwender und dem SMALLTALK-System – verwendet werden sollen.

Im Hinblick auf die Problemstellung PROB-5 sind also die Klassen, die für die algorithmische Verarbeitung und den Zugriff auf die erfaßten Daten – wie z.B. zur Berechnung des Modus, des Medians oder des Durchschnitts – benötigt werden, von den Klassen zu trennen, mit deren Hilfe der Erfassungsprozeß fenster-gestützt zur Ausführung gelangt.

Sofern der Lösungsplan auf dem Model/View-Konzept beruhen soll, muß die vorgenommene Gliederung zur folgenden Aufteilung führen:

Abbildung 13.2: Model/View-Konzept mit einer Sicht

Auf der Basis dieser Gliederung ist es möglich, für einen Lösungsplan unterschiedliche Benutzeroberflächen zu realisieren, ohne daß die Methoden zur Verarbeitung der Daten im Model-Teil geändert werden müssen.

Hinweis: Dieses Vorgehen bietet zudem die Möglichkeit, daß zur Erfassung anderer Daten als der jahrgangsstufen-spezifischen Punktwerte alle diejenigen Klassen, die im View-Teil zusammengefaßt sind, wiederverwendet werden können, d.h. die Struktur des Erfassungsfensters könnte unverändert bzw. mit kleinen Modifikationen direkt zur Lösung einer anderen Problemstellung eingesetzt werden.

Sofern eine andere Anwendung auf der Basis der bereits für den Model-Teil festgelegten Klassen entwickelt werden soll, kann dies allein durch die Programmierung eines neuen View-Teils geleistet werden. Diesen Vorteil des Model/View-Konzeptes verdeutlicht die folgende Abbildung:

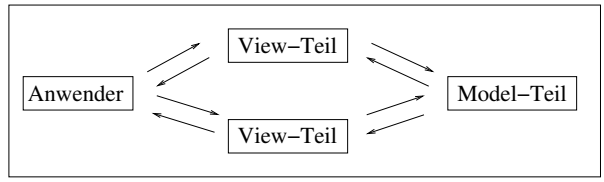


Abbildung 13.3: Model/View-Konzept mit mehreren Sichten

So können z.B. die Einrichtung und Anzeige von Fenstern mit unterschiedlichen Grafiken, die den Informationsgehalt von Daten beschreiben, in Form verschiedener View-Teile programmiert werden, die sämtlich auf einem einzigen Model-Teil basieren.

13.2 Beispiel für die Verwendung des Model/View-Konzeptes

Gliederung des Lösungsplans

Um das grundsätzliche Vorgehen bei der Gliederung in einen Model-Teil und einen View-Teil zu schildern, streben wir im folgenden eine weitere Lösung der Problemstellung PROB-5 an, die in Abschnitt 7.4 wie folgt formuliert wurde:

- PROB-5:
  Auf der Basis der erfaßten Werte soll – mittels einer geeigneten Instanziierung – einheitlich über den Message-Selektor "zentrum" der jeweils charakteristische statistische Kennwert angefordert werden können.
  Ergänzend sollen der Modus, der Median und der Durchschnittswert immer dann, wenn die jeweilige statistische Kennzahl sinnvoll ist, auch *gezielt* abrufbar sein.

Im Hinblick auf den früher – in Abschnitt 7.4 – entwickelten Lösungsplan stellen wir fest:

- Als Objekte, die Gegenstand von algorithmischen Verarbeitungsschritten sind, wurden die Punktwerte in einem Sammler verwaltet, der bei einem Erfassungsprozeß durch die Instanz-Variable "werteBag" der jeweiligen Instanziierung einer der drei untergeordneten Klassen von "WerteErfassung" realisiert war.

- Zur Sicherung von algorithmischen Berechnungen des Zentrums wurden die Instanz-Variablen "moduswert","medianwert" und "durchschnittswert" verwendet.

Die vier aufgeführten Instanz-Variablen sind die einzigen Instanz-Variablen, die für die zu erfassenden und zu sichernden Daten benötigt werden. Daher legen wir diese Instanz-Variablen – innerhalb des Model-Teils – in einer Klasse namens “WerteErfassungModell” fest.
Der für den View-Teil – zur Verwaltung eines Views – benötigten Instanz-Variablen, die in einer Unterklasse von “ViewManager” festgelegt sein muß, geben wir den Namen “daten”.
Sofern wir “daten” als Instanz-Variable einer Klasse namens “WerteErfassungView” vereinbaren, bietet sich die folgende strukturelle Gliederung des im Sinne des Model/View-Konzeptes zu realisierenden Lösungsplans an:

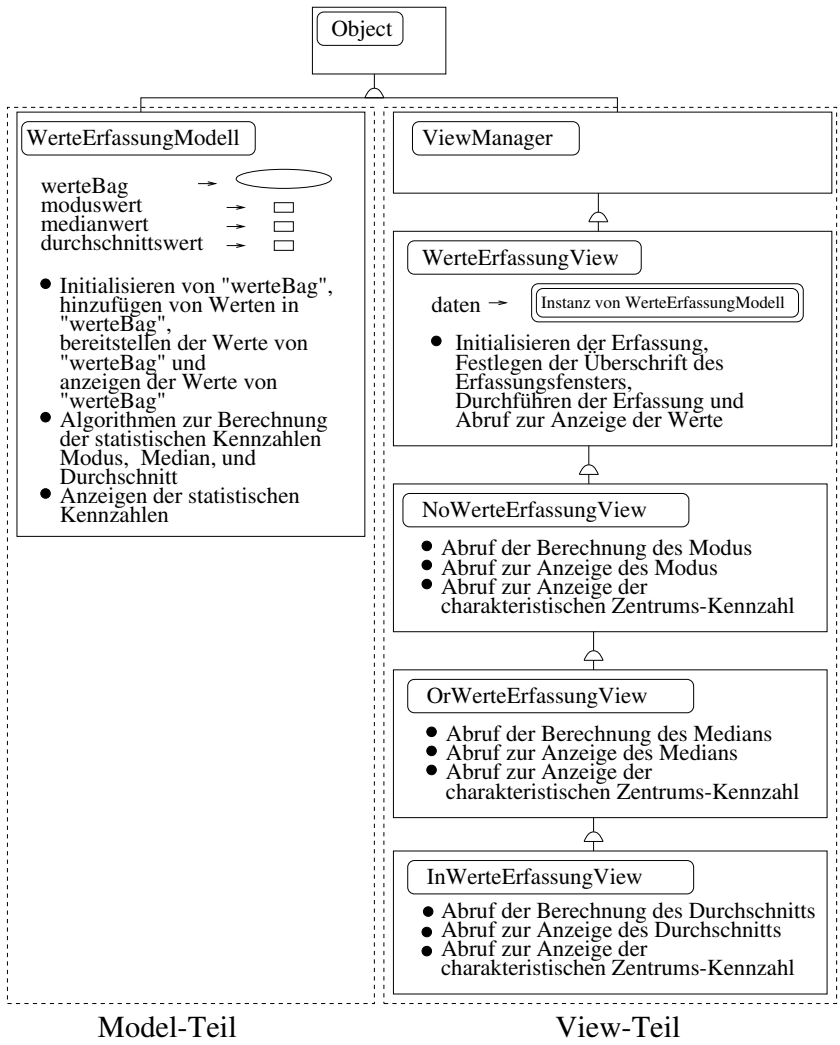


Abbildung 13.4: Struktur zur Lösung von PROB-5

Die Klasse "WerteErfassungModell"

Um einem Erfassungsprozeß die jeweils erfaßten Daten zuordnen zu können, soll bei einer Instanziierung aus der Klasse "WerteErfassungModell" die Instanz-Variable "werteBag" als Bag instanziiert werden.
Dazu vereinbaren wir innerhalb der Klasse "WerteErfassungModell" die Instanz-Methode "initialisierenBag" in der folgenden Form:

```
initialisierenBag
werteBag := Bag new
```

Damit diese Methode bei jeder neuen Instanziierung von "WerteErfassungModell" ausgeführt wird, legen wir die folgende Klassen-Methode fest:

```
new
|var|
var := super new.
var initialisierenBag.
∧ var
```

Wir beabsichtigen, daß – wie bisher – jeder neu erfaßte Wert in der Instanz-Variablen "werteBag" einer Instanz der Klasse "WerteErfassungModell" gesammelt werden soll. Wegen der Datenkapselung läßt sich dies nicht mehr in der ursprünglichen Form – unter Einsatz der Basis-Methode "add:" – erreichen. Damit die erfaßten Werte durch Instanzen von Klassen, die keine Unterklassen von "WerteErfassungModell" sind, der Instanz-Variablen "werteBag" hinzugefügt werden können, vereinbaren wir innerhalb der Klasse "WerteErfassungModell" die folgende Instanz-Methode "hinzufuegenWert:":

```
hinzufuegenWert: aString
werteBag add: aString
```

Die ursprünglich innerhalb der Klasse "WerteErfassung" festgelegten Methoden (siehe Abschnitt 7.4) "anzeigenWerte", "anzeigenDurchschnittswert", "anzeigenMedianwert", "anzeigenModuswert", "bereitstellenWerte", "durchschnitt", "median" und "modus" übernehmen wir in unveränderter Form in die Klasse "WerteErfassungModell".
Da hierdurch alle Methoden zur Sicherung, Verwaltung und Auswertung in der Klasse "WerteErfassungModell" vereinbart sind, ist die zur Erstellung des Model-Teils erforderliche Programmierung durchgeführt.

Die Klasse "WerteErfassungView" und deren Unterklassen

Um den View-Teil zu programmieren, richten wir – als Pendant zu der zur Lösung von PROB-5 ursprünglich konzipierten Klasse "WerteErfassung" – die Klasse "WerteErfassungView" als direkte Unterklasse der Basis-Klasse "ViewManager" und unterhalb von "WerteErfassungView" die Klassen "NoWerteErfassungView", "OrWerteErfassungView" sowie "InWerteErfassungView" – in dieser Reihenfolge – ein.

Hinweis: Diese drei Klassen sollen den zur Lösung von PROB-5 eingesetzten Klassen "NoWerteErfassung", "OrWerteErfassung" sowie "InWerteErfassung" entsprechen (siehe Abschnitt 7.4).

Zur Benennung der jeweils eingesetzten Instanz der Klasse "WerteErfassungModell" wird in der Klasse "WerteErfassungView" eine Instanz-Variable benötigt.
Anstelle der ursprünglich verwendeten Instanz-Variablen "werteBag" der Klasse "WerteErfassung" setzen wir jetzt in der Klasse "WerteErfassungView" die Instanz-Variable "daten" ein, durch die auf eine Instanz der Klasse "WerteErfassungModell" verwiesen werden soll.
Für die Initialisierung des Erfassungsfensters verabreden wir in der Klasse "WerteErfassungView" wiederum die Methode "initialisierenErfassung". Dabei legen wir die in Abschnitt 12.4 angegebene Methoden-Vereinbarung zugrunde. Als einzige Änderung müssen wir die ursprünglich eingesetzte Zuweisung

```
werteBag := Bag new.
```

in die Zuweisung

```
daten := WerteErfassungModell new.
```

abändern.
Somit besitzt die Methode "initialisierenErfassung" innerhalb der Klasse "WerteErfassungView" die folgende Form:

```
initialisierenErfassung
daten := WerteErfassungModell new.
self addView: ( self topPaneClass new owner: self;
  labelWithoutPrefix:  ' ';
  noSmalltalkMenuBar;
  viewName: 'erfassungsfenster';
  framingRatio: ( (Rectangle leftTopUnit
    rightAndDown:1/2@(1/2)) extentFromLeftTop:1/2@(1/2));
 addSubpane: (StaticText new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:1/8@(2/8)) extentFromLeftTop:2/8@(1/8));
     contents: 'Wert:');
 addSubpane: (EntryField new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:2/8@(2/8)) extentFromLeftTop:2/8@(1/16));
     paneName: 'eingabeFeld');
 addSubpane: (Button new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:2/8@(6/8)) extentFromLeftTop:2/8@(1/8));
     contents: 'erfasse';
     when: #clicked perform: #erfassenWert:);
 addSubpane: (Button new owner: self;
     framingRatio: ( (Rectangle leftTopUnit
        rightAndDown:5/8@(6/8)) extentFromLeftTop:2/8@(1/8));
```

```
    contents: 'Erfassungsende';
    when: #clicked perform: #entfernenErfassungsfenster:)
          )
```

Da wir die Punktwerte nicht mehr unmittelbar in die Instanz-Variable "werteBag" einer Instanz von "WerteErfassungModell" eintragen und auf die gesammelten Punktwerte nicht mehr direkt zugreifen können, ist es nicht möglich, die Methoden "erfassenWert:" und "anzeigenWerte" der Klasse "WerteErfassung" – ohne Änderung – in die Klasse "WerteErfassungView" zu übernehmen.
Indem wir die Anforderung

```
werteBag add: (self paneNamed: 'eingabeFeld') contents
```

in die Form

```
daten hinzufuegenWert:(self paneNamed: 'eingabeFeld') contents
```

abändern, können wir die derart modifizierte Methode "erfassenWert:" wie folgt in der Klasse "WerteErfassungView" vereinbaren:

```
erfassenWert: aPane
daten hinzufuegenWert:(self paneNamed: 'eingabeFeld') contents.
(self paneNamed: 'eingabeFeld') contents: ''.
(self paneNamed: 'eingabeFeld') setFocus
```

Um die gesammelten Punktwerte im Transcript-Fenster anzeigen zu können, vereinbaren wir in der Klasse "WerteErfassungView" die Methode "abrufenAnzeigenWerte" in der folgenden Form:

```
abrufenAnzeigenWerte
daten anzeigenWerte
```

Die innerhalb von "WerteErfassung" vereinbarten Methoden "durchfuehrenErfassung", "entfernenErfassungsfenster:", "sammelnWerte:" und "festlegenUeberschrift:" (siehe die Abschnitte 2.4, 2.5 und 4.4) legen wir in ihrer ursprünglichen Form innerhalb der Klasse "WerteErfassungView" fest.
Da von Instanziierungen, die aus Unterklassen von "WerteErfassungView" erfolgen, nicht unmittelbar auf die gesammelten Punktwerte und die berechneten statistischen Kennzahlen zugegriffen werden kann, ist es notwendig, auf der Basis der Methoden der Klassen "NoWerteErfassung", "OrWerteErfassung" und "InWerteErfassung" neue Methoden zur Anzeige der jeweils sinnvollen Kennzahlen und zur Berechnung der charakteristischen Zentrums-Kennzahlen zu vereinbaren.
In dieser Hinsicht legen wir für die Klasse "NoWerteErfassungView" die Methoden "abrufenAnzeigenModuswert", "abrufenAnzeigenZentrum" und "zentrum" in Form von

```
abrufenAnzeigenModuswert
daten anzeigenModuswert
```

und

```
abrufenAnzeigenZentrum
daten anzeigenModuswert
```

sowie

```
zentrum
daten modus
```

fest. Entsprechend vereinbaren wir innerhalb von “OrWerteErfassungView” die Methoden

```
abrufenAnzeigenMedianwert
daten anzeigenMedianwert
```

und

```
abrufenAnzeigenZentrum
daten anzeigenMedianwert
```

sowie:

```
zentrum
daten median
```

Hinweis: Zur Berechnung des für ordinalskalierte Daten sinnvollen Moduswertes können wir für die Klasse “OrWerteErfassungView” die Methode “‘modus” in der Form

```
modus
super zentrum
```

aus der Klasse “OrWerteErfassung” identisch übernehmen.

Innerhalb der Klasse “InWerteErfassungView” treffen wir die Methoden-Vereinbarungen

```
abrufenAnzeigenDurchschnittswert
daten anzeigenDurchschnittswert
```

und

```
abrufenAnzeigenZentrum
daten anzeigenDurchschnittswert
```

sowie:

```
zentrum
daten durchschnitt
```

Hinweis: Zur Berechnung des für intervallskalierte Daten sinnvollen Medianwertes können wir die Methode “median” aus der Klasse “InWerteErfassung” identisch in die Klasse “InWerteErfassungView” übernehmen. Sie hat die folgende Form:

```
median
super zentrum
```

Ausführung des Lösungsplans

Zusammenfassend können wir somit die Lösung von PROB-5 – unter Einsatz des Model/View-Konzeptes – wie folgt angeben:

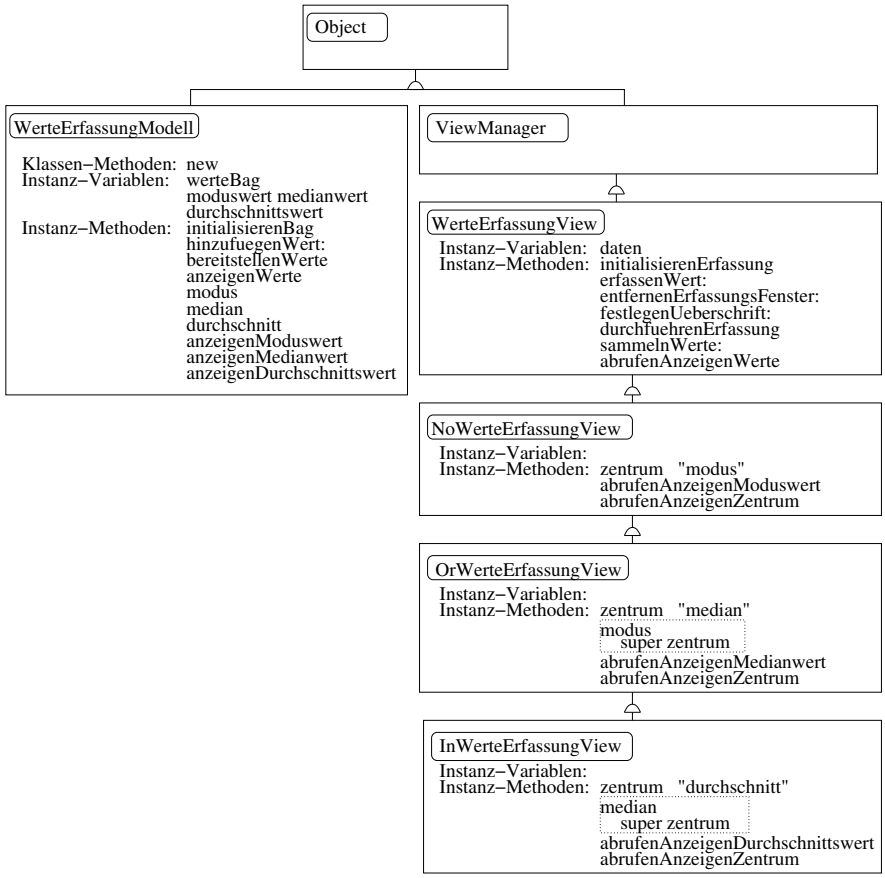


Abbildung 13.5: Lösung von PROB-5 gemäß dem Model/View-Konzept

Sofern wir z.B. für die Jahrgangsstufe 11 die Punktwerte erfassen und die jeweils sinnvollen und charakteristischen statistischen Kennwerte ermitteln wollen, können z.B. zunächst die Anforderungen

```
InWerteErfassungView11 := InWerteErfassungView new.
InWerteErfassungView11 sammelnWerte: 'Jahrgangsstufe 11'
```

und anschließend die Anforderungen

```
InWerteErfassungView11 zentrum "Durchschnitt";
  median "Median"; modus "Modus";
  abrufenAnzeigenDurchschnittswert; abrufenAnzeigenMedianwert;
```

```
  anzeigenModuswert
```

oder aber

```
OrWerteErfassungView11 := OrWerteErfassungView new.
OrWerteErfassungView11 sammelnWerte: 'Jahrgangsstufe 11'
```

und daraufhin

```
OrWerteErfassungView11 zentrum "Median"; modus "Modus";
  abrufenAnzeigenMedianwert; abrufenAnzeigenModuswert
```

bzw. die Anforderungen

```
NoWerteErfassungView11 := NoWerteErfassungView new.
NoWerteErfassungView11 sammelnWerte: 'Jahrgangsstufe 11'
```

und abschließend die Anforderungen

```
NoWerteErfassungView11 zentrum "Modus"; abrufenAnzeigenModuswert
```

gestellt werden.

Hinweis: Wollen wir z.B. überprüfen, welche Werte in den Instanz-Variablen einer Instanz der Klasse "WerteErfassungModell" eingetragen sind, auf die die Instanz-Variable "daten" von "InWerteErfassung11" verweist, so können wir die Anforderung
`InWerteErfassung11 inspect`
ausführen lassen. Markieren wir anschließend im angezeigten Inspect-Fenster den Eintrag "daten" und wählen im Menü "Inspect" die Option "Inspect", so müssen wir letztlich im eröffneten zweiten Inspect-Fenster den Eintrag "werteBag" aktivieren.

13.3 Dialog-orientierte Ablaufsteuerung

Bei der bislang entwickelten Lösung von PROB-5 müssen wir die jeweils gewünschte Erfassung und die angeforderten Statistiken in Form von Anforderungen in ein Workspace-Fenster eintragen und die angezeigten Ergebnisse im Transcript-Fenster ablesen.

Dieser Lösungsplan läßt sich verbessern, indem wir die Möglichkeiten der fensterorientierten Dialogführung nutzen. Daher wollen wir die ursprüngliche Zielsetzung durch die folgende Problemstellung ersetzen:

- PROB-9:
  Bei der Erfassung der Punktwerte und der Berechnung der statistischen Kennzahlen soll ein rein fenster-gestützter Dialog ablaufen. Dazu ist ein geeignetes Anforderungsfenster zu konzipieren, in dem Anforderungen durch Mausklicks gestellt werden können.
  Um festzulegen, welche statistischen Kennzahlen berechnet werden sollen, ist ein Auswertungsfenster zu konzipieren, in dem die berechneten Werte anzuzeigen sind.

Im Hinblick auf den zuvor im Rahmen des Model/View-Konzeptes entwickelten Lösungsplan setzen wir zur Lösung dieser Problemstellung folgendes voraus:

- Das Erfassungsfenster wird aus der Klasse "WerteErfassungView" und das Objekt zur Sicherung der Punktwerte aus der Klasse "WerteErfassungModell" instanziiert.

Im Hinblick auf die Problemstellung wollen wir für den Dialog – neben dem bisherigen Erfassungsfenster – zusätzlich die folgenden Fenster einrichten:

- Ein *Anforderungsfenster* mit der Titel-Zeile "Anforderungen", das aus der Klasse "AnforderungenView" instanziiert werden soll. Für dieses Fenster sehen wir den folgenden Aufbau vor:

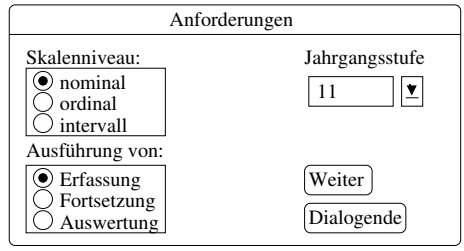


Abbildung 13.6: Aufbau des Anforderungsfensters

- Ein *Auswertungsfenster* mit der Titel-Zeile "Auswertung", das aus der Klasse "AuswertungView" instanziiert und die folgende Form haben soll:

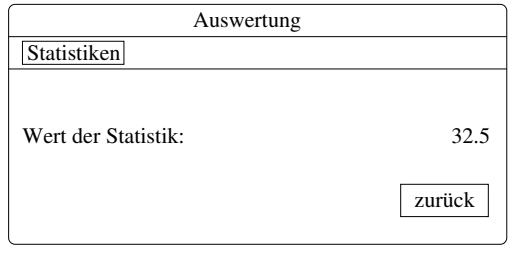


Abbildung 13.7: Aufbau des Auswertungsfensters

Dabei soll das Menü "Statistiken" die Menü-Optionen "Modus", "Median" und "Durchschnitt" enthalten, über die sich die jeweiligen statistischen Kennzahlen abrufen lassen.

Im Hinblick auf die in dieser Form zu entwickelnden Fenster sehen wir für den Lösungsplan von PROB-9 die folgende Klassen-Hierarchie vor:

```
Object
 WerteErfassungModell  (mit den Instanz-Variablen: werteBag,
                        moduswert, medianwert, durchschnittswert)
 ViewManager
    WerteErfassungView (mit der Instanz-Variablen: daten)
    AnforderungenView  (mit den Klassen-Variablen:
                        Skalenniveau, Jahrgang und
                        der Instanz-Variablen:
                        auswertungsFenster)
       AuswertungView
```

Dabei setzen wir voraus, daß die Klassen "WerteErfassungModell" und "WerteErfassungView" in der im vorigen Abschnitt beschriebenen Form vereinbart sind.

Zur Lösung von PROB-9 werden wir unter Einsatz der Klassen-Variablen "Skalenniveau" festhalten, um welche Art von Werten es sich bei den erfaßten Punktwerten handelt. Daher ist die ursprünglich entwickelte Klassen-Hierarchie mit den Klassen "NoWerteErfassungView", "OrWerteErfassungView" und "InWerteErfassungView" entbehrlich.

Die Klasse "AnforderungenView"

Die Klasse "AnforderungenView" vereinbaren wir als direkte Unterklasse der Klasse "ViewManager". Als Instanz-Variable legen wir "auswertungsFenster" fest, und als Klassen-Variablen vereinbaren wir "Skalenniveau" und "Jahrgang".

Um Vorkehrungen dafür treffen zu können, daß das Anforderungsfenster in der oben angegebenen Form angezeigt wird, ist zu berücksichtigen, daß dieses Fenster aus den folgenden Bausteinen besteht:

- der Titel-Zeile "Anforderungen" und der System-Schaltfeld-Zeile, die das System-Menü sowie die System-Schaltfelder zur "Minimierung", "Maximierung" und zum "Schließen" enthält;
- dem Textfeld "Skalenniveau:", dem die Gruppierung von drei Optionsfeldern mit den Texten "nominal", "ordinal" bzw. "intervall" zur Charakterisierung der erfaßten Punktwerte zugeordnet ist (je nachdem, welches dieser Optionsfelder aktiviert wird, soll die Ausführung der Methoden "umschaltenNo:", "umschaltenOr:" oder "umschaltenIn:" und "vorbereitenErfassungsfenster" zur Einrichtung geeigneter Instanzen der Unterklassen von "WerteErfassungView" erfolgen);
- dem Textfeld "Ausführung von:", dem die Gruppierung von drei Optionsfeldern mit den Texten "Erfassung", "Fortsetzung" bzw. "Auswertung" zugeordnet ist (hierdurch soll durch die Methoden "umschaltenErf:", "umschaltenFor:" oder "umschaltenAus:" gesteuert werden, ob eine Erfassung, die Fortsetzung der Erfassung oder die Auswertung der erfaßten Punktwerte abgerufen werden soll);

- dem Textfeld “Jahrgangsstufe”, das das Kombinationsfeld mit den Listen-Items “11”, “12” und “13” beschreibt (dies soll die Jahrgangsstufe der erfaßten Punktwerte kennzeichnen);
- dem Schaltfeld “Weiter” (dadurch soll ein Erfassungsfenster bzw. ein Auswertungsfenster – durch die Ausführung der Methode “weiter:” – zur Anzeige gebracht werden);
- dem Schaltfeld “Dialogende” (durch dieses Schaltfeld soll die dialog-orientierte Ablaufsteuerung durch die Ausführung der Methode “ende:” beendet werden).

Zum Aufbau der erforderlichen Fenster-Bausteine für das Anforderungsfenster vereinbaren wir in der Klasse “AnforderungenView” die Instanz-Methode “initialisierenAnforderungen” in der folgenden Form:

```
initialisierenAnforderungen
self addView: ( self topPaneClass new owner: self;
  labelWithoutPrefix:  'Anforderungen';
  noSmalltalkMenuBar;
  viewName: 'dialogfeld';
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:0@0) extentFromLeftTop:1/2@(1/2));
  pStyle: #(sysmenu sizable titlebar minimize maximize);

addSubpane: (StaticText new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(1/16)) extentFromLeftTop:(3/8)@(1/16));
  contents: 'Skalenniveau:');
addSubpane: ( GroupPane new
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(1/8)) extentFromLeftTop:(6/16)@(1/3));
addSubpane: (RadioButton new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(1/8)) extentFromLeftTop:(7/8)@(2/8));
  paneName: 'dialogNo';
  selection: true;
  when: #clicked perform: #umschaltenNo:;
  contents: 'nominal' );
addSubpane: (RadioButton new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(3/8)) extentFromLeftTop:(7/8)@(2/8));
  paneName: 'dialogOr';
  when: #clicked perform: #umschaltenOr:;
  contents: 'ordinal');
addSubpane: (RadioButton new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(5/8)) extentFromLeftTop:(7/8)@(2/8));
  paneName: 'dialogIn';
```

```
  when: #clicked perform: #umschaltenIn:;
  contents: 'intervall') );

addSubpane: (StaticText new owner: self;
   framingRatio:((Rectangle leftTopUnit
    rightAndDown:(1/8)@(9/16)) extentFromLeftTop:(3/8)@(1/16));
   contents: 'Ausführung von:');

addSubpane: ( GroupPane new
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(5/8)) extentFromLeftTop:(6/16)@(1/3));
addSubpane: (RadioButton new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(1/8)) extentFromLeftTop:(7/8)@(2/8));
  paneName: 'dialogErf';
  selection:true;
  when: #clicked perform: #umschaltenErf:;
  contents: 'Erfassung');
addSubpane: (RadioButton new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(3/8)) extentFromLeftTop:(7/8)@(2/8));
  paneName: 'dialogFor';
  when: #clicked perform: #umschaltenFor:;
  contents: 'Fortsetzung');
addSubpane: (RadioButton new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(1/8)@(5/8)) extentFromLeftTop:(7/8)@(2/8));
  paneName: 'dialogAus';
  when: #clicked perform: #umschaltenAus:;
  contents: 'Auswertung') );

addSubpane: (StaticText new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(5/8)@(1/16)) extentFromLeftTop:(3/8)@(2/16));
  contents: 'Jahrgangsstufe');
addSubpane: (ComboBox new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(5/8)@(3/16)) extentFromLeftTop:(1/4)@(6/16));
  paneName: 'dialogJahrgang';
  insertItem:'11';
  insertItem:'12';
  insertItem:'13';
  selection:'11');

addSubpane: (Button new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(5/8)@(10/16)) extentFromLeftTop:(3/16)@(2/16));
```

```
  when: #clicked perform: #weiter:;
  contents: 'Weiter');
addSubpane: (Button new owner: self;
  framingRatio:((Rectangle leftTopUnit
   rightAndDown:(5/8)@(13/16)) extentFromLeftTop:(6/16)@(2/16));
  when: #clicked perform: #ende:;
  contents: 'Dialogende')
             )
```

Hinweis: Zur übersichtlichen Gliederung haben wir logisch zusammengehörige Anforderungen durch Leerzeilen voreinander abgegrenzt.

Um das Anforderungsfenster anzeigen zu lassen, vereinbaren wir innerhalb der Klasse "AnforderungenView" die Methode "anzeigenAnforderungen" in der folgenden Form:

```
anzeigenAnforderungen
self openWindow
```

Damit eines der durch den Text "Skalenniveau:" gekennzeichneten Optionsfelder durch einen Mausklick aktiviert werden kann, sehen wir die Methoden "umschaltenNo:", "umschaltenOr:" und "umschaltenIn:" wie folgt vor:

```
umschaltenNo: aRadioButton
(self paneNamed: 'dialogNo') selection: true.
(self paneNamed: 'dialogOr') selection: false.
(self paneNamed: 'dialogIn') selection: false

umschaltenOr: aRadioButton
(self paneNamed: 'dialogNo') selection: false.
(self paneNamed: 'dialogOr') selection: true.
(self paneNamed: 'dialogIn') selection: false

umschaltenIn: aRadioButton
(self paneNamed: 'dialogNo') selection: false.
(self paneNamed: 'dialogOr') selection: false.
(self paneNamed: 'dialogIn') selection: true
```

Um die Art der Auswertung über die durch den Text "Ausführung von:" gekennzeichneten Optionsfelder festlegen zu können, vereinbaren wir in der Klasse "AnforderungenView" die Methoden "umschaltenErf:", "umschaltenFor:" und "umschaltenAus:" in der folgenden Form:

```
umschaltenErf: aRadioButton
(self paneNamed: 'dialogErf') selection: true.
(self paneNamed: 'dialogFor') selection: false.
(self paneNamed: 'dialogAus') selection: false

umschaltenFor: aRadioButton
```

```
(self paneNamed: 'dialogErf') selection: false.
(self paneNamed: 'dialogFor') selection: true.
(self paneNamed: 'dialogAus') selection: false

umschaltenAus: aRadioButton
(self paneNamed: 'dialogErf') selection: false.
(self paneNamed: 'dialogFor') selection: false.
(self paneNamed: 'dialogAus') selection: true
```

Desweiteren sind die Methoden "weiter:" und "ende:", die bei der Bestätigung des Schaltfeldes "Weiter" bzw. "Dialogende" zur Ausführung gelangen sollen, wie folgt zu vereinbaren:

```
ende: aTopPane
self close

weiter: aButton
Jahrgang:=(self paneNamed: 'dialogJahrgang') text.
(self paneNamed: 'dialogNo') selection
   ifTrue:[Skalenniveau:='No'].
(self paneNamed: 'dialogOr') selection
   ifTrue:[Skalenniveau:= 'Or'].
(self paneNamed: 'dialogIn') selection
   ifTrue:[Skalenniveau:= 'In'].
(self paneNamed: 'dialogErf') selection
   ifTrue: [self vorbereitenErfassungsfenster].
(self paneNamed: 'dialogFor') selection
   ifTrue: [self vorbereitenFortsetzungsfenster].
(self paneNamed: 'dialogAus') selection
   ifTrue: [auswertungsFenster:=AuswertungView new.
            auswertungsFenster anzeigenAuswertung]
```

Hinweis: Aus Gründen der Vollständigkeit führen wir die Methode "weiter:" bereits an dieser Stelle auf.
Da innerhalb dieser Methode in der Anforderung

```
auswertungsFenster:=AuswertungView new
```

"AuswertungView" für die noch einzurichtende Klasse "AuswertungView" (siehe unten) stehen soll, dürfen wir die Methode "weiter:" erst nach der Vereinbarung von "AuswertungView" verabreden. Dies liegt daran, daß das SMALLTALK-System versucht, die durch einen Großbuchstaben eingeleitete Angabe "AuswertungView" als globale Variable einzurichten.

Dabei sind ergänzend die Methoden "vorbereitenErfassungsfenster" und "vorbereitenFortsetzungsfenster" in der Form

```
vorbereitenErfassungsfenster
Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang)
                asSymbol
          put: (WerteErfassungView new).
(Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang) asSymbol)
   initialisierenErfassung;
   festlegenUeberschrift: ('Jahrgangsstufe ',Jahrgang);
   durchfuehrenErfassung
```

und in der Form

```
vorbereitenFortsetzungsfenster
(Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang) asSymbol)
   festlegenUeberschrift: ('Jahrgangsstufe ',Jahrgang);
   durchfuehrenErfassung
```

innerhalb der Klasse "AnforderungenView" zu vereinbaren.

Hinweis: Es ist zu beachten, daß durch die Ausführung der Methode "vorbereitenErfassungsfenster" der Klasse "AnforderungenView" eine Instanz der Klasse "WerteErfassungView" (mit der Instanz-Variablen "daten") eingerichtet und einer globalen Variablen – wie z.B. der Variablen "InWerteErfassung14" – zugeordnet wird. Dabei verweist die Instanz-Variable "daten" auf eine Instanz der Klasse "WerteErfassungModell", innerhalb der die erfaßten Punktwerte – in der Instanz-Variablen "werteBag" – gesammelt werden.

Die Klasse "AuswertungView"

Als Unterklasse von "AnforderungenView" legen wir die Klasse "AuswertungView" fest und vereinbaren in ihr wie folgt die Methode "initialisierenAuswertung" zur Initialisierung des Auswertungsfensters:

```
initialisierenAuswertung
self addView: ( self topPaneClass new owner: self;
  labelWithoutPrefix: 'Auswertung';
  noSmalltalkMenuBar;
  viewName: 'auswertung';
  framingRatio:((Rectangle leftTopUnit
     rightAndDown:(1/2)@0) extentFromLeftTop:(1/2)@(1/2));
  pStyle: #(sysmenu sizable titlebar minimize maximize);

addSubpane:(StaticText new owner: self;
  framingRatio:((Rectangle leftTopUnit
     rightAndDown:(1/8)@(3/8)) extentFromLeftTop:(4/8)@(1/8));
  contents: 'Wert der Statistik:');
addSubpane: (StaticText new owner: self;
  framingRatio:((Rectangle leftTopUnit
     rightAndDown:(5/8)@(3/8)) extentFromLeftTop:(2/8)@(1/8));
  rightJustified;
```

```
  paneName: 'statistikanzeige');
addSubpane: (Button new owner: self;
  framingRatio:((Rectangle leftTopUnit
     rightAndDown:(1/2)@(5/8)) extentFromLeftTop:(2/8)@(1/8));
  when: #clicked perform: #zurueck:;
  contents: 'zurück')
              ).

(Skalenniveau = 'No')
ifTrue: [(self menuWindow) addMenu: (Menu new owner: self;
               title: 'Statistiken';
               appendItem: 'Modus' selector: #zeigenModus)].
(Skalenniveau = 'Or')
ifTrue: [(self menuWindow) addMenu: (Menu new owner: self;
               title: 'Statistiken';
               appendItem: 'Modus' selector: #zeigenModus;
               appendItem: 'Median' selector: #zeigenMedian)].
(Skalenniveau = 'In')
ifTrue: [(self menuWindow) addMenu: (Menu new owner: self;
              title: 'Statistiken';
              appendItem: 'Modus' selector: #zeigenModus;
              appendItem: 'Median' selector: #zeigenMedian;
              appendItem: 'Durchschnitt'
                selector: #zeigenDurchschnitt)]
```

Um die Anzeige des Auswertungsfensters anfordern zu können, treffen wir die folgende Vereinbarung:

```
anzeigenAuswertung
self initialisierenAuswertung.
self openWindow
```

Zum Entfernen des Auswertungsfensters vom Bildschirm legen wir die Instanz-Methode "zurueck:" wie folgt fest:

```
zurueck: aButton
self close
```

Damit über das im Auswertungsfenster enthaltene Pulldown-Menü "Statistiken" – mit den Menü-Optionen "Modus", "Median" und "Durchschnitt" – die jeweils zugehörigen Statistiken angezeigt werden können, vereinbaren wir die Methoden "zeigenModus" "zeigenMedian", und "zeigenDurchschnitt" in der folgenden Form:

```
zeigenDurchschnitt
(self paneNamed: 'statistikanzeige') contents:
   ( ((Smalltalk at: ((Skalenniveau,'WerteErfassung',Jahrgang)
                        asSymbol))
```

```
       bereitstellenDurchschnitt) asFloat printString )

zeigenMedian
(self paneNamed: 'statistikanzeige') contents:
   ( ((Smalltalk at: ((Skalenniveau,'WerteErfassung',Jahrgang)
                      asSymbol))
       bereitstellenMedian) asFloat printString )

zeigenModus
(self paneNamed: 'statistikanzeige') contents:
   ( ((Smalltalk at: ((Skalenniveau,'WerteErfassung',Jahrgang)
                      asSymbol))
       bereitstellenModus) asFloat printString )
```

Erweiterung der Klasse "WerteErfassungView"

Bei der Vereinbarung dieser Methoden wird auf Methoden zurückgegriffen, mit denen die jeweils sinnvollen statistischen Kennzahlen aus der betreffenden Instanz der Klasse "WerteErfassungView" berechnet und abgerufen werden können.
Diese Methoden sind in der Klasse "WerteErfassungView" wie folgt – ergänzend – festzulegen:

```
bereitstellenDurchschnitt
daten durchschnitt.
∧ daten bereitstellenDurchschnittswert

bereitstellenMedian
daten median.
∧ daten bereitstellenMedianwert

bereitstellenModus
daten modus.
∧ daten bereitstellenModuswert
```

Erweiterung der Klasse "WerteErfassungModell"

Abschließend ist zu berücksichtigen, daß für den Zugriff, den Instanzen der Klasse "WerteErfassungView" auf die berechneten Statistiken (zugreifbar über die Instanz-Variablen "durchschnittswert", "medianwert" und "moduswert" einer Instanz der Klasse "WerteErfassungModell") durchführen können müssen, noch die Methoden "bereitstellenDurchschnittswert", "bereitstellenMedianwert" und "bereitstellenModuswert" in den Formen

```
bereitstellenDurchschnittswert
∧ durchschnittswert

bereitstellenMedianwert
∧ medianwert

bereitstellenModuswert
∧ moduswert
```

in der Klasse “WerteErfassungModell” zu vereinbaren sind.
Insgesamt ergibt sich somit die folgende Struktur des Lösungsplans:

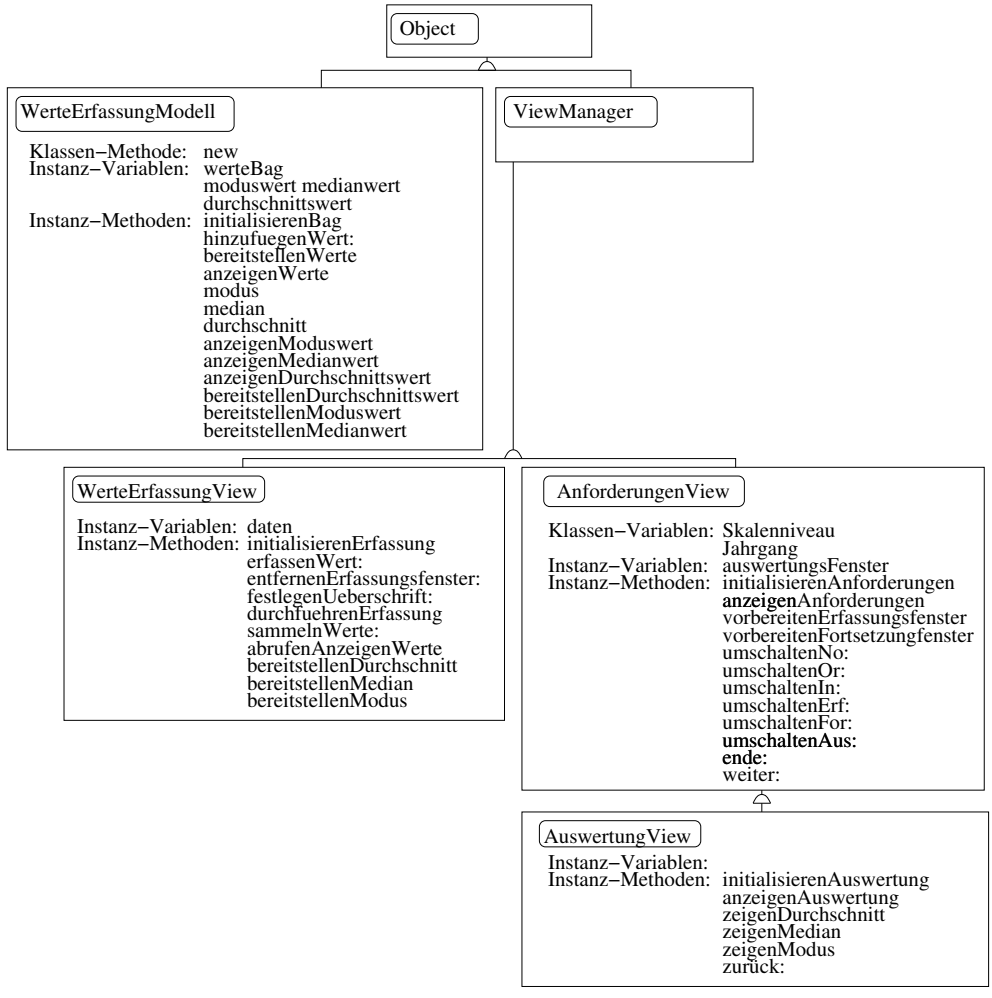


Abbildung 13.8: Klassen zur Lösung von PROB-9

Ausführung des Lösungsplans

Nachdem wir die Anforderung

```
AnforderungenView new initialisierenAnforderungen;
  anzeigenAnforderungen
```

gestellt haben, wird das Anforderungsfenster wie folgt am Bildschirm angezeigt:

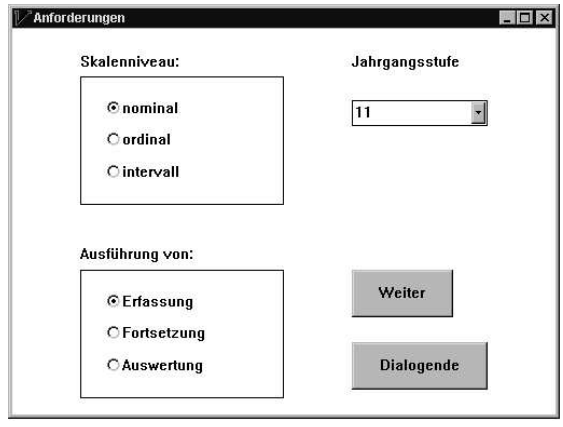


Abbildung 13.9: Anforderungsfenster

Um z.B. die intervallskalierten Daten für die Jahrgangsstufe 13 zu erfassen, aktivieren wir in diesem Fenster das Optionsfeld "intervall" und das Optionsfeld "Erfassung". Nachdem wir im Kombinationsfeld "Jahrgangsstufe" den Wert "13" festgelegt haben, betätigen wir das Schaltfeld "Weiter". Daraufhin erscheint das Erfassungsfenster in der uns bekannten Form. Nach der Erfassung wird durch die Betätigung des Schaltfeldes "Erfassungsende" wiederum das Anforderungsfenster zur Anzeige gebracht.

Um das Auswertungsfenster abzurufen, aktivieren wir das Optionsfeld "Auswertung" und betätigen das Schaltfeld "Weiter". Daraufhin erscheint das Auswertungsfenster in der folgenden Form:

Abbildung 13.10: Auswertungsfenster

Über das Menü “Statistiken” lassen sich die Menü-Optionen “‘Modus”, “Median” und “Durchschnitt” zur Anzeige der gewünschten Statistiken abrufen. Die jeweils angeforderte Verarbeitung stützt sich dabei auf den folgenden Sachverhalt:

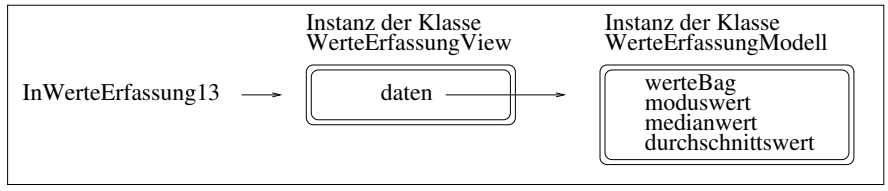


Abbildung 13.11: Objekte bei der Lösung von PROB-9

Nach der Anzeige der Statistiken läßt sich das Auswertungsfenster über das Schaltfeld “zurück” schließen, so daß im Anforderungsfenster der Dialog weitergeführt oder aber – über das Schaltfeld “Dialogende” – beendet werden kann.

Hinweis: Es ist zu beachten, daß die beim Ablauf der dialog-orientierten Ablaufsteuerung eingerichteten Instanzen der Klassen “AnforderungenView” und “AuswertungView” nur temporär eingerichtet werden. Sie stehen somit nach dem Ende der Ausführung von

```
AnforderungenView new initialisierenAnforderungen;
  anzeigenAnforderungen
```

nicht mehr zur Verfügung.

13.4 Szenarien und Ereignisfolgen-Diagramme

Im Hinblick auf die unterschiedlichen Sichten, die bei der Lösung von Problemstellungen zu berücksichtigen sind, haben wir bislang die *statische* und die *funktionale* Sicht vorgestellt. Während bei der statischen Sicht die Beschreibung der Klassen Gegenstand des Interesses ist, geht es bei der funktionalen Sicht um die Beschreibung der Methoden, die die Instanzen der Klassen ausführen können.

Um bei der Planung einer Lösung den zeitlichen Ablauf zu kennzeichnen, in dem die jeweiligen Objekte *Zustandsänderungen* unterworfen werden, kann der Lösungsplan auch aus einer *dynamischen* Sicht – unter Einsatz von *Szenarien* und *Ereignisfolgen-Diagrammen* – betrachtet werden.

Szenarien

Als *Szenario* wird eine mögliche, bei der Ausführung von Anforderungen – im zeitlichen Ablauf – eintretende Folge von Handlungen des *Anwenders* bezeichnet, die zur Darstellung der Zusammenhänge einer Problemstellung konstruiert wird. Dabei wird mit einer verbalen Niederschrift (einem *Skript*) angegeben, wie das durch den Lösungsplan modellierte System reagieren soll.

Bei einer derartigen Beschreibung ist darauf zu achten, daß jede wichtige Handlung, die sich zur Laufzeit bei der Ausführung eines Lösungsplans ergeben kann, in einem Szenario erfaßt ist.

Für die im vorigen Abschnitt beschriebene dialog-orientierte Ablaufsteuerung ergeben sich z.B. die folgenden Szenarien:

Szenario 1:

1. Der Anwender startet die dialog-orientierte Ablaufsteuerung zur Erfassung der Punktwerte.
2. Am Bildschirm wird das Anforderungsfenster angezeigt.
3. Zur Erfassung intervallskalierter Punktwerte – z.B. der Jahrgangssstufe 13 – sind die Optionsfelder "intervall" und "Erfassung" zu aktivieren, und es ist der Wert "13" in das Kombinationsfeld einzutragen.
4. Abschließend ist das Schaltfeld "Weiter" zu betätigen.
5. Am Bildschirm erscheint das Erfassungsfenster, in das die Punktwerte sukzessive eingetragen und durch Aktivieren des Schaltfeldes "erfasse" gesammelt werden.
6. Sind alle Punktwerte erfaßt, ist im Erfassungsfenster das Schaltfeld "Erfassungsende" anzuklicken. Daraufhin wird das Erfassungsfenster vom Bildschirm gelöscht.
7. Zur Berechnung der Statistiken für die erfaßten intervallskalierten Punktwerte der Jahrgangsstufe 13 sind vom Anwender – im Anforderungsfenster – die Optionsfelder "intervall" und "Auswertung" zu aktivieren und daraufhin der Wert "13" in das Kombinationsfeld einzugeben.
8. Abschließend ist das Schaltfeld "Weiter" zu betätigen.
9. Am Bildschirm erscheint das Auswertungsfenster, in dem die jeweils sinnvollen statistischen Kennzahlen (Modus, Median und Durchschnitt) für die intervallskalierten Punktwerte der Jahrgangsstufe 13 berechnet und angezeigt werden können.
10. Soll das Auswertungsfenster vom Bildschirm entfernt werden, so ist das Schaltfeld "zurück" zu aktivieren.
11. Zum Beenden der dialog-orientierten Ablaufsteuerung ist im angezeigten Anforderungsfenster das Schaltfeld "Dialogende" zu betätigen.

Szenario 2:

1. Zur Fortsetzung der Erfassung intervallskalierter Punktwerte der Jahrgangsstufe 13 sind im Anforderungsfenster die Optionsfelder "intervall" und "Fortsetzung" zu aktivieren sowie der Wert "13" in das Kombinationsfeld einzutragen.
2. Abschließend ist das Schaltfeld "Weiter" zu betätigen.

3. Am Bildschirm erscheint das Erfassungsfenster, in das jeweils die Punktwerte eingetragen und durch Aktivieren des Schaltfeldes “erfasse” den bereits erfaßten Punktwerten der Jahrgangsstufe 13 sukzessiv hinzugefügt werden.

4. Sollen für diese Jahrgangsstufe keine weiteren Punktwerte mehr erfaßt werden, so ist das Schaltfeld “Erfassungsende” zu betätigen. Daraufhin wird das Erfassungsfenster vom Bildschirm entfernt.

5. Zur Beendigung der dialog-orientierten Ablaufsteuerung ist im angezeigten Anforderungsfenster das Schaltfeld “Dialogende” zu betätigen.

Hinweis: Um das Verhaltensmodell für die dialog-orientierte Ablaufsteuerung vollständig zu beschreiben, müßten noch Szenarien für die Betrachtung nominalskalierter und ordinalskalierter Punktwerte beschrieben werden. Da sich diese Fälle aus den Szenarien 1 und 2 ohne weiteres übertragen lassen, verzichten wir auf die Angabe dieser Szenarien.

Ereignisfolgen-Diagramme

Nachdem die durch die Problemstellung gekennzeichneten Szenarien beschrieben sind, kann die Programmierung des Lösungsplans vorbereitet werden. Dazu sind die einzelnen Szenarien in korrespondierende Ereignisfolgen-Diagramme abzubilden, in dem die jeweiligen Objekte sowie die jeweils verschickten Messages aufgeführt werden, die das durch die Szenarien beschriebene Verhalten bewirken.

- Ein *Ereignisfolgen-Diagramm* gibt eine formale Darstellung der durch ein Szenario beschriebenen Objekt-Kommunikation wieder. Es dient dazu, die Aufeinanderfolge der Kommunikation zwischen den Objekten des modellierten Systems darzustellen.

Somit stellt jedes Ereignisfolgen-Diagramm einen möglichen zeitlichen Ablauf bei der Ausführung eines Lösungsplans dar und erleichtert somit die Prüfung des Verhaltens eines modellierten Systems aus dynamischer Sicht.
Bei dieser Sicht ist es bedeutsam, ob ein Objekt dauerhaft zur Verfügung steht oder nicht. In dieser Hinsicht kann zwischen persistenten und transienten Objekten unterschieden werden.

- Ein *persistentes* Objekt steht permanent über den Dialog mit dem SMALLTALK-System hinaus, in dem es durch eine Instanziierung erzeugt wurde, zur Verfügung. Existiert ein Objekt nicht über das Ende des Dialogs hinaus, bei dem dieses Objekt instanziiert wurde, so wird von einem *transienten* Objekt gesprochen.

Für die Darstellung von Szenario 1 ergibt sich für die Erfassung zweier intervallskalierter Punktwerte der Jahrgangsstufe 13 und deren Auswertung das folgende Diagramm:

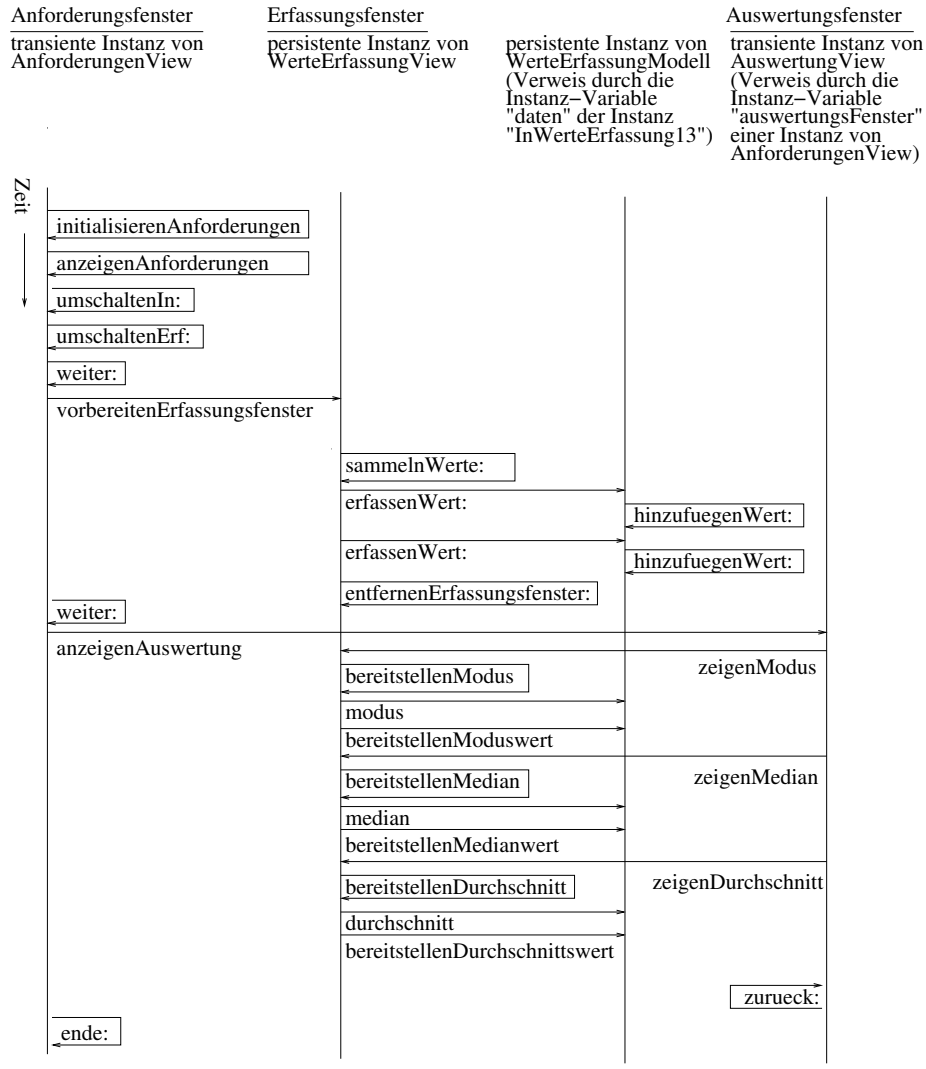


Abbildung 13.12: Ereignisfolgen-Diagramm für Szenario 1 von PROB-9

Hinweis: Der Vollständigkeit halber müßten wir auch für sämtliche Instanzen der Fenster-Bausteine des Anforderungsfensters, des Auswertungsfensters und des Erfassungsfensters eine eigene Spalte einführen. Zur vollständigen Dokumentation wäre es auch notwendig, die implizit aktivierten Messages – wie z.B. "openWindow" und "close" – zum Eröffnen und Schließen von Views oder die Message "new" zum Einrichten von Instanzen aufzuführen.

In diesem Diagramm werden die zur Lösung von PROB-9 eingesetzten Objekte – in Form von Instanzen der Klassen "AnforderungenView", "WerteErfassungView", "WerteErfassungModell" und "AuswertungView" – durch vertikale Linien dargestellt.

Horizontale Pfeile zwischen zwei vertikalen Linien sollen diejenigen Methoden symbolisieren, durch deren Ausführung ein Objekt einem anderen Objekt eine Message

schickt. So wird z.B. durch den Pfeil mit der darunterstehenden Angabe “anzeigen-Auswertung” ausgedrückt, daß eine Instanz der Klasse “AnforderungenView” diese Message einer Instanz der Klasse “AuswertungView” zustellt.

Hinweis: Genau genommen wird zuvor die Message “weiter:” einer Instanz der Klasse “AnforderungenView” zugestellt. Erst durch die Ausführung der zugehörigen Methode “weiter:” erfolgt das Schicken der Message “anzeigenAuswertung” an ein anderes Objekt in Form einer Instanz der Klasse “AuswertungView”.

Pfeile, die zu ihrer Ausgangslinie wieder zurückführen, sollen Methoden kennzeichnen, durch deren Ausführung ein Objekt sich selbst eine Message zustellt. Beispiele für derartige Methoden sind “hinzufuegenWert:” und “bereitstellenModus”.

In diesem Diagramm wird die Reihenfolge, in der die angegebenen Methoden ausgeführt werden, durch ihre vertikale Position festgelegt. Somit befinden sich im oberen Teil des Diagramms Messages, die zuerst verschickt werden. Im unteren Teil sind die zuletzt zugestellten Messages aufgeführt.

Hinweis: Dabei ist zu beachten, daß weder durch die Länge der Pfeile noch durch ihre vertikale Position die Zeitdauer für die Ausführung einer Message beschrieben wird.

Neben den bisher beschriebenen Szenarien und Ereignisfolgen-Diagramme lassen sich bei der professionellen Programmierung auch Zustands-Diagramme zur Darstellung der dynamischen Sicht eines modellierten Systems einsetzen.

Kapitel 14

Langfristige Sicherung von Objekten

14.1 Das Image

14.1.1 Das Entwicklungs- und Laufzeit-System

Alle während eines Dialogs mit dem SMALLTALK-System durchgeführten Änderungen am Bestand der im System vorhandenen Objekte – sei es die Einrichtung neuer Klassen, die Löschung von Klassen, die Vereinbarung neuer Methoden, die Änderung von Methoden sowie die Einrichtung bzw. Änderung von globalen Variablen – müssen am Dialogende gesichert werden, sofern der aktuelle Zustand zu Beginn des nächsten Dialogs zur Verfügung stehen soll.

Bei dieser Sicherung kann – im Transcript-Fenster – die Menü-Option "Save Image" des Menüs "File" bestätigt werden, so daß das folgende Dialogfeldfenster am Bildschirm angezeigt wird:

Abbildung 14.1: Sicherung des Images

Sofern eine Sicherung erfolgen soll, ist das Schaltfeld "Yes" zu bestätigen. Sollen die während des aktuellen Dialogs durchgeführten Änderungen wieder rückgängig gemacht werden, so muß eine Bestätigung des Schaltfeldes "No" erfolgen.

Da Fehlbedienungen des Anwenders und betriebsbedingte Systemzusammenbrüche nicht auszuschließen sind, ist es grundsätzlich empfehlenswert, nicht nur eine derartige Sicherung vorzunehmen, sondern im Hinblick auf die Art der Sicherung auch darauf zu achten, daß das Kontrollfeld "Backup" aktiviert ist. Diese Empfehlung hängt damit zusammen, daß ein bestimmter Teil des SMALLTALK-Systems durch ein Fehlerverhalten des Anwenders besonders gefährdet ist.

Zum Verständnis dieses Sachverhalts ist zunächst der folgende Tatbestand von Bedeutung:

- Das SMALLTALK-System besteht aus dem Entwicklungs-System und einem Laufzeit-System, das *Image* genannt wird.

Das Image enthält neben den vom SMALLTALK-System zur Verfügung gestellten Basis-Klassen und globalen Objekten zusätzlich die vom Anwender eingerichteten Klassen, Methoden und globalen Variablen sowie die Informationen über geöffnete Fenster und deren Inhalte, die zum Abschluß des zuletzt geführten Dialogs am Bildschirm angezeigt wurden.

- Der Arbeitsspeicher des SMALLTALK-Systems, in dem diese Objekte verwaltet werden, wird *automatisch* vom SMALLTALK-System daraufhin überwacht, ob bestimmte Bereiche durch vom Anwender angeforderte Löschvorgänge freigeworden sind.
  Ist dies der Fall, so erfolgt *automatisch* eine Speicherbereinigung ("garbage collection"), so daß stets nur der Speicherbereich belegt wird, der auch tatsächlich benötigt wird.

Das Entwicklungs-System stellt die Werkzeuge – in Form von graphischen Benutzeroberflächen wie z.B. den Klassen-Hierarchie-Browser, das Transcript-Fenster und ein oder mehrere Workspace-Fenster – zur Verfügung, mit denen der Anwender die Ausführung seiner Anforderungen vom SMALLTALK-System abrufen kann.

Hinweis: Zu den Bausteinen des Entwicklungs-Systems zählen der Compiler und der Interpreter. Durch den Compiler werden Methoden-Vereinbarungen in Instruktionen eines Zwischenkodes – den sogenannten "Byte-Kode" – umgeformt. Der Interpreter, der als "virtuelle Maschine" arbeitet, führt diesen Zwischenkode dann aus, wenn die zugehörige Methode durch eine mit ihr korrespondierende Message aktiviert wird.
Bestimmte Basis-Methoden des SMALLTALK-Systems liegen bereits im Zwischenkode vor. Dazu zählen z.B. die *Primitiva* als grundlegende Methoden, auf deren Basis andere Basis-Methoden entwickelt sind. Primitiva sind innerhalb von Vereinbarungen von Basis-Methoden in der Form

```
<primitive: nummer>
```

festgelegt, wobei der ganzzahlige Wert "nummer" die jeweilige Zwischenkode-Sequenz kennzeichnet, unter der sie bei der "virtuellen Maschine" registriert ist.

- Das Entwicklungs-System ist in der Datei "vw.exe" und das standardmäßig – durch die Installation des SMALLTALK-Systems – zur Verfügung stehende Image in der Datei "v.exe" gespeichert, so daß der jeweils aktuelle Zustand des SMALLTALK-Systems insgesamt durch die Inhalte der Dateien "vw.exe" und "v.exe" gekennzeichnet wird.

- Um das SMALLTALK-System zu starten, ist der Inhalt der Datei "vw.exe" zur Ausführung zu bringen.

Sofern sich das Image durch die vom Anwender gestellten Anforderungen ändert, wird der Inhalt der Datei "v.exe" modifiziert. Diese zunächst temporäre Änderung läßt sich durch eine Sicherung des Images (siehe die Abbildung 14.1) in eine dauerhafte Form überführen.

Durch eine Fehlbedienung des Anwenders oder einen betriebsbedingten Systemzusammenbruch kann der Inhalt von "v.exe" unbrauchbar werden, so daß das SMALLTALK-System nicht mehr gestartet werden kann.

Um für eine derartige Situation geeignete Vorkehrungen zu treffen, sollte bei einer Sicherung (siehe das oben abgebildete Dialogfeldfenster) das Kontrollfeld "Backup" aktiviert sein. Dies bewirkt, daß nicht nur der aktuelle Zustand in der Datei "v.exe" gesichert, sondern zusätzlich eine *Backup-Kopie* des Images in Form einer Datei namens "v.bak" durchgeführt wird.

In der Situation, in der der Start des SMALLTALK-Systems wegen eines Systemzusammenbruchs nicht erfolgen kann, ist wie folgt vorzugehen:

- Es ist zunächst der Inhalt der Datei "v.bak" in die Datei "v.exe" zu kopieren und anschließend – nach dem Start des SMALLTALK-Systems im Transcript-Fenster die Menü-Option "Open..." des Menüs "File" zu bestätigen. In dem daraufhin angezeigten Fenster ist "change.log" als Name der *Protokoll-Datei* in das Eingabefeld einzutragen, so daß der Inhalt dieser Datei bearbeitet werden kann.

  In der Protokoll-Datei "change.log" sind alle Angaben – im Chunk-Format (siehe Anhang A.2) – enthalten, durch die das Image verändert wurde. Um den zuletzt erreichten Zustand des SMALLTALK-Systems herzustellen, müssen daher alle Angaben, die nach der letzten Sicherung durchgeführt wurden (der Zeitpunkt, zu dem die letzte Sicherung des Images erfolgte, ist innerhalb der Protokoll-Datei eingetragen), markiert und über die Menü-Option "Do It" des Menüs "Smalltalk" zur Ausführung gebracht werden. Anschließend steht das SMALLTALK-System in der Form zur Verfügung, in der es sich vor dem Systemzusammenbruch befunden hat.

  Hinweis: Wenn die zuletzt ausgeführte Anforderung, die in der Protokoll-Datei eingetragen ist, zum Systemzusammenbruch geführt hat, muß diese Anforderung von der Ausführung ausgeschlossen bleiben.

Da die Inhalte der beiden Dateien "v.exe" und "change.log" unmittelbar miteinander verzahnt sind, ist unbedingt darauf zu achten, daß die Konsistenz dieser beiden Dateien erhalten bleibt. Demzufolge dürfen Änderungen an diesen Dateien nur unter Einsatz des SMALLTALK-Systems vorgenommen werden.

Da die Protokoll-Datei "change.log" mit jeder Änderung des Images erweitert wird, sollte – in regelmäßigen Abständen – die folgende Anforderung gestellt werden:

```
Smalltalk compressChanges
```

Dadurch werden alle redundanten Angaben innerhalb der Protokoll-Datei gelöscht, so daß z.B. aus ursprünglich wiederholt vorgenommenen Methoden-Vereinbarungen allein die zuletzt durchgeführte Vereinbarung in der Protokoll-Datei erhalten bleibt.

Hinweis: Grundsätzlich enthält die Protokoll-Datei keine Angaben über syntaktisch nicht korrekte Methoden-Vereinbarungen oder Anforderungen.
Um die Protokoll-Datei "change.log" zu leeren, besteht die Möglichkeit, alle bislang das Basis-System ergänzenden Klassen- und Methoden-Vereinbarungen innerhalb einer Quellkode-Datei namens "sources.sml" zu sichern. Dies läßt sich durch die folgende Anforderung bewerkstelligen:

```
SMALLTALK compressSources
```

Durch die Ausführung der Message “compressSources” wird die Datei “sources.sml” erweitert, indem sämtliche gültigen Klassen- und Methoden-Vereinbarungen aus der Protokoll-Datei “change.log” hinzugefügt und diese Angaben innerhalb der Protokoll-Datei gelöscht werden.

14.1.2 Erstellen eines eigenständigen Laufzeit-Systems

Nachdem eine Anwendung zur Lösung einer Problemstellung programmiert wurde, kann sie als eigenständiges Laufzeit-System gesichert werden. Dadurch wird es möglich, einen Dialog mit der Anwendung zu starten, ohne daß ein Dialog mit dem Entwicklungs-System – unter Einsatz des Transcript-Fensters oder eines Workspace-Fensters – erforderlich ist.

Um z.B. die im Abschnitt 13.3 entwickelte Anwendung zur Erfassung von Punktwerten als eigenständiges Laufzeit-System einzurichten, muß die Basis-Methode “startUpApplication:” wie folgt innerhalb der Basis-Klasse “NotificationManager” abgeändert werden:

```
startUpApplication: oldWindows
AnforderungenView new initialisierenAnforderungen;
                      anzeigenAnforderungen
```

Hinweis: Die Methode “startUpApplication:” enthält ursprünglich die folgende Anforderung:

```
GraphicsDemo new open
```

Zum Schließen sämtlicher Fenster mit Ausnahme des Transcript-Fensters läßt sich die folgende Anforderung stellen:

```
Notifier reinitialize
```

Dabei ist das Empfänger-Objekt der Message “reinitialize” in Form der globalen Variablen “Notifier” die einzige Instanz der Basis-Klasse “NotificationManager”.

Anschließend ist das aktuelle Image zu sichern, so daß die Datei “v.exe” das Laufzeit-System für die Anwendung enthält.
Um dieses Laufzeit-System isoliert einsetzen zu können, kopieren wir den Inhalt der Datei “v.exe” in eine Datei, deren Dateiname der Anwendung entsprechend gewählt werden sollte.
Wählen wir bei der Lösung von PROB-9 für die einzurichtende Datei z.B. den Dateinamen “Anforder.exe”, so läßt sich die Anwendung anschließend dadurch starten, daß die Ausführung von “Anforder.exe” veranlaßt wird.
Dabei ist folgendes zu beachten:

- Im Rahmen einer Laufzeit-Version gibt es keine *persistenten* Objekte. Somit stehen bei der Ausführung des Laufzeit-Systems keine Instanziierungen zur Verfügung, die beim vorausgegangenen Start des Laufzeit-Systems vorgenommen wurden.

Sofern wir z.B. die von uns im Abschnitt 13.3 entwickelte Anwendung zur Lösung

von PROB-9 in Form eines Laufzeit-Systems erstellen, gehen sämtliche erfaßten Punktwerte verloren, wenn die Ausführung der Anwendung beendet wird. Dies liegt daran, daß das Laufzeit-System stets in seiner letzten Form geladen wird. Da Instanzen zur Aufnahme von zu erfassenden Punktwerten daher immer wieder erneut eingerichtet werden müssen, ist die *Fortsetzung* einer Erfassung – im Gegensatz zu dem auf dem Entwicklungs-/Laufzeit-System basierenden bisherigen Lösungsplan – in dieser Form nicht möglich.

Um den ursprünglichen Lösungsplan so abzuändern, daß er für den *alleinigen* Einsatz des Laufzeit-Systems ausgelegt ist und die Persistenz der bereits erfaßten Punktwerte garantiert, bietet sich die folgende Vorgehensweise an:

- Vor dem Ende der Anwendung sind die erfaßten Punktwerte in einer *Datei* – einem auf einem magnetischen Datenträger eingerichteten Datenspeicher – zu sichern, so daß sie beim erneuten Start der Anwendung bei Bedarf wieder zur weiteren Bearbeitung bereitgestellt werden können.

Wie sich die hierdurch erforderliche Änderung des Lösungsplans durchführen läßt, stellen wir im folgenden Abschnitt dar.

14.2 Einsatz von Datenströmen

14.2.1 Datenströme

Um auf die Inhalte von Sammlern *sequentiell*, d.h. der Reihe nach, zugreifen zu können, müssen Objekte verwendet werden, die als Instanz einer geeigneten Unterklasse der abstrakten Klasse “Stream” einzurichten sind. Derartige Instanzen besitzen die Struktur eines “Datenstroms”, dessen Komponenten aus *gereihten* Objekten bestehen.

Welche Position die einzelnen Objekte innerhalb des Datenstroms einnehmen, wird durch eine ihnen zugeordnete *Positionsnummer* gekennzeichnet, so daß von einem ersten Objekt, einem zweiten Objekt usw. gesprochen werden kann.

Grundsätzlich weist der *Positionszeiger* auf das jeweils aktuelle Objekt. Da ein Datenstrom nach seiner Einrichtung noch kein Objekt enthält, besitzt der Positionszeiger in diesem Fall den Wert “0”.

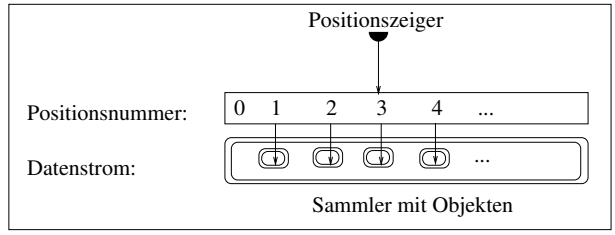


Abbildung 14.2: Datenstrom über einem Sammler

Hinweis: Ist ein Datenstrom nicht leer, so verweist der Positionszeiger in dem Fall, in dem er den Wert “0” besitzt, auf die Position *vor* dem 1. Objekt.

Ein Datenstrom wird dadurch gefüllt, daß ein erstes Objekt eingetragen wird, danach ein zweites Objekt, usw. Anschließend kann ein Zugriff auf die Objekte durchgeführt werden, der entweder über eine vorgegebene Positionsnummer oder aber *sequentiell* erfolgt, indem von einem Objekt auf das unmittelbar nachfolgende Objekt positioniert wird.
Sofern der Inhalt eines Datenstroms die Form einer Zeichenkette besitzt, kann er in einer *Datei* gesichert werden.
Dieser Sachverhalt versetzt uns in die Lage, die Persistenz von Objekten auch für den Fall zu ermöglichen, in dem eine Anwendung *allein* auf den Einsatz eines Laufzeit-Systems gegründet werden soll.

14.2.2 Einrichtung und Bearbeitung einer Ausgabe-Datei

Zur Einrichtung einer Datei stellt das SMALLTALK-System die Basis-Klasse "File" zur Verfügung. Durch eine Instanziierung dieser Klasse läßt sich ein Datenstrom einrichten, dessen Inhalt die Form einer Zeichenkette besitzt. Dieser Datenstrom wird nicht im Speicherbereich des Laufzeit-Systems, sondern *extern* unter Einsatz des Betriebssystems als Datei auf einem magnetischen Datenträger verwaltet.
Um eine Datei einzurichten, kann die Message "pathName:" verwendet werden, bei der der jeweils gewählte Dateiname als Argument aufzuführen ist.

- **"pathName:"**:
  Durch den Einsatz der *Klassen-Methode* "pathName:" wird ein externer Datenstrom in Form einer Datei eingerichtet, deren Dateiname als Argument des Selektors "pathName:" in Form einer Zeichenkette anzugeben ist.

Zum Beispiel wird durch die Anforderung

```
|ausgabeDatei|
ausgabeDatei := File pathName: 'bag.dat'
```

eine Datei namens "bag.dat" als externer Datenstrom vereinbart und zur Bearbeitung eröffnet.
Soll z.B. der Inhalt eines Bags, in dem mehrere Zeichenketten – wie z.B. "33", "31" und "34" – gesammelt sind, in die Datei "bag.dat" übertragen werden, so können die folgenden Anforderungen gestellt werden:

```
|ausgabeDatei bag|
bag := Bag new.
bag add: '33'.
bag add: '31'.
bag add: '34'.
ausgabeDatei := File pathName: 'bag.dat'.
bag do: [:einObjekt|ausgabeDatei space; nextPutAll: einObjekt].
ausgabeDatei close
```

Durch diese Anforderungen wird ein externer Datenstrom aufgebaut, der sich aus den Zeichenketten “33”, “31” und “34” zusammensetzt, wobei diesen Zeichenketten jeweils ein Leerzeichen vorangestellt ist. Anschließend wird die Datei “bag.dat” geschlossen, d.h. von der Verarbeitung abgemeldet, so daß z.B. von einer anderen Anwendung auf den Inhalt dieser Datei zugegriffen werden kann.
Hinweis: Als Inhalt der Datei “bag.dat” ergibt sich die Zeichenkette “ 33 31 34”.

- Grundsätzlich ist zu beachten, daß die Datei “bag.dat” durch ein geeignetes Betriebssystem-Kommando zu löschen ist, sofern die in ihr gespeicherten Daten entfernt werden sollen.

Um die eingerichtete Datei “bag.dat” als *Ausgabe-Datei* zu beabeiten, haben wir die folgenden Basis-Methoden eingesetzt:

- **“space”**:
  Durch die Ausführung der Methode “space” wird ein Leerzeichen in denjenigen Datenstrom eingetragen, der als Empfänger-Objekt der Message “space” aufgeführt ist.

- **“nextPutAll:”**:
  Durch die Ausführung dieser Methode wird der Wert des Positionszeigers um “1” erhöht, und die einzelnen Objekte des als Argument aufgeführten Sammlers werden ab der Position, auf die der Positionszeiger verweist, sukzessive in den Datenstrom eingetragen, der als Empfänger-Objekt der Message “nextPutAll:” aufgeführt ist. Als Ergebnis-Objekt der Message “nextPutAll:” resultiert der als Argument angegebene Sammler.

- **“close”**:
  Um den Zugriff auf die durch das Empfänger-Objekt gekennzeichnete Datei zu beenden, muß die Datei geschlossen werden. Hierzu ist die Message “close” an ein Empfänger-Objekt zu richten, das als Ergebnis-Objekt aus der Ausführung der Message “pathName:” resultierte.

  Hinweis: Da die Zeichenketten – vor ihrer Übertragung in die Datei – zunächst in einem Zwischenspeicher (Puffer) gesammelt werden, wird durch die Message “close” zusätzlich sichergestellt, daß der Inhalt dieses Zwischenspeichers am Verarbeitungsende ordnungsgemäß in die Datei übernommen wird.
  Der Einsatz der Message “close” ist auch deshalb notwendig, weil eine als “geöffnet” gekennzeichnete Datei nicht parallel bearbeitet werden darf. Bei einem Verstoß erscheint ein Walkback-Fenster (siehe Anhang A.3) mit der Fehlermeldung “Smalltalk Express sharing violation”.

Soll in der im Abschnitt 13.3 angegebenen Lösung – im Hinblick auf den alleinigen Einsatz eines Laufzeit-Systems – der Inhalt des Bags mit den erfaßten Punktwerten in einer Datei gesichert werden, so kann dazu die folgende Methode “schreibenDatei:sichernBag:” verwendet werden:

```
schreibenDatei: aString sichernBag: aBag
|ausgabeDatei|
ausgabeDatei := File pathName: aString.
aBag do: [:einObjekt|ausgabeDatei space; nextPutAll:einObjekt].
ausgabeDatei close
```

Diese Methode ist innerhalb der von uns im Abschnitt 13.3 eingerichteten Klasse "AnforderungenView" zu vereinbaren. Ferner ist die innerhalb dieser Klasse festgelegte Methode "ende:" wie folgt abzuändern:

```
ende: aTopPane
self close.
self schreibenDatei:(Prompter prompt:'Gib Dateiname:' default:'')
     sichernBag:
       ( (Smalltalk at: ((Skalenniveau,'WerteErfassung',Jahrgang)
                          asSymbol))
          bereitstellenDateiwerte )
```

Hinweis: Als Ergebnis-Objekt von "(Prompter prompt: 'Gib Dateiname:' default:")" resultiert derjenige Name, der in das Eingabefeld eines zuvor angezeigten Dialogfeldes als gewünschter Dateiname eingetragen wurde.

Damit in dieser Form auf die bereits erfaßten Werte zugegriffen werden kann, ist die Methode "bereitstellenDateiwerte" zusätzlich wie folgt innerhalb der Klasse "WerteErfassungView" festzulegen:

```
bereitstellenDateiwerte
∧ daten bereitstellenWerte
```

Wird nach dieser Ergänzung die Lösung von PROB-9 in Form eines Laufzeit-Systems erstellt und – ohne Entwicklungs-System – zur Ausführung gebracht, so ist die Sicherung der erfaßten Punktwerte durch die Speicherung in einer Datei gewährleistet.

Dies gibt uns die Möglichkeit, nach einem erneuten Start des Laufzeit-Systems auf die gesicherten Punktwerte zuzugreifen, so daß auch die *Fortsetzung* einer Erfassung durchführbar ist.

Um die Lösung von PROB-9 ergänzen zu können, stellen wir im folgenden Abschnitt eine Methode vor, mit der der Zugriff auf den Inhalt der zuvor eingerichteten Datei ermöglicht wird.

Hinweis: Es lassen sich nicht nur die Punktwerte selbst, sondern auch Messages sichern, durch deren Ausführung eine ursprünglich vorhandene Instanz – samt der in ihr zuvor gesammelten Punktwerte – automatisch wiederhergestellt werden kann. Damit diese Rekonstruktion möglich ist, muß allerdings das Entwicklungs-System zur Verfügung stehen. Derartige Messages werden in Form von Zeichenketten erhalten, die sich als Ergebnis-Objekt der Message "storeString" ergeben. Dabei ist als Empfänger-Objekt jeweils dasjenige Objekt aufzuführen, das wiederhergestellt werden können soll.
Weist z.B. die Instanz-Variable "daten" aus der Klasse "WerteErfassungModell" auf einen Bag mit dem Punktwert "32", der über den Einsatz von "InWerteErfassung11" als Instan-

ziierung der Klasse “InWerteErfassung” erfaßt wurde, so resultiert das Ergebnis-Objekt
`'((Bag new) add: "32"; yourself)'`
aus der Anforderung:
`(InWerteErfassung11 bereitstellenDateiwerte) storeString`
Wird diese Zeichenkette innerhalb einer Datei gesichert und zu einem späteren Zeitpunkt z.B. der Variablen “text” zugeordnet, so wird die ursprüngliche Instanz der Klasse “Bag” durch die Anforderung
`Compiler evaluate: text`
wiederhergestellt.

14.2.3 Bearbeitung einer Eingabe-Datei

Nachdem wir erläutert haben, wie sich Zeichenketten in einer Datei namens “bag.dat” sichern lassen, stellen wir jetzt eine Methode vor, mit dem die gespeicherten Werte zur weiteren Bearbeitung bereitgestellt werden können.
Damit die Zeichenketten *sequentiell* aus einer *Eingabe-Datei* gelesen und in einen Bag namens “bag” übernommen werden, stellen wir die folgenden Anforderungen:

```
|eingabeDatei bag|
bag := Bag new.
eingabeDatei := File pathName: 'bag.dat'.
[eingabeDatei atEnd] whileFalse:[bag add:(eingabeDatei nextWord)].
eingabeDatei close.
bag
```

Zur Bearbeitung einer Eingabe-Datei setzen wir – neben den bereits oben vorgestellten Basis-Methoden “pathName:” und “close” – die folgenden Methoden ein:

- **“atEnd”**:
  Als Empfänger-Objekt der Message “atEnd” ist ein zuvor vereinbarter Datenstrom aufzuführen.
  Als Ergebnis-Objekt resultiert die Pseudovariable “true”, falls das *Dateiende* erreicht ist, d.h. falls der Positionszeiger hinter das letzte Zeichen des Datenstroms weist – andernfalls wird die Pseudovariable “false” ermittelt.

- **“nextWord”**:
  Als Ergebnis-Objekt der Message “nextWord” wird diejenige Zeichenkette erhalten, auf deren Anfang der Positionszeiger weist und die bis zum Dateiende bzw. bis zum nächsten auftretenden *Leerzeichen* reicht. Der Positionszeiger wird im Datenstrom auf das erste hinter einem nachfolgenden Leerzeichen auftretende Zeichen gesetzt. Weist der Positionszeiger unmittelbar hinter das letzte im Datenstrom enthaltene Zeichen, so ist das *Dateiende* erreicht.

Soll in der im Abschnitt 13.3 angegebenen Lösung dafür gesorgt werden, daß die *Fortsetzung* einer Datenerfassung auch dann ermöglicht wird, sofern nur mit dem Laufzeit-System gearbeitet wird, so müssen die zuvor in einer Datei gesicherten Punktwerte – vor der Weiterführung der Erfassung – geeignet bereitgestellt werden.

Um die Punktwerte in einen bereits instanziierten Sammler zu übernehmen, ist z.B. die folgende Methode "lesenDatei:" geeignet:

```
lesenDatei: aString
|eingabeDatei|
eingabeDatei := File pathName: aString.
[eingabeDatei atEnd] whileFalse:
  [(Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang) asSymbol)
    uebertragenWert: (eingabeDatei nextWord)].
eingabeDatei close
```

Diese Methode ist innerhalb der von uns im Abschnitt 13.3 eingerichteten Klasse "AnforderungenView" zu vereinbaren.

Damit durch den Einsatz der Methode "uebertragenWert:" ein jeweils aus der Datei übernommener Wert in den Sammler "daten" eingetragen werden kann, muß diese Methode in der Form

```
uebertragenWert: aString
daten hinzufuegenWert: aString
```

ergänzend in der Klasse "WerteErfassungView" festgelegt werden.

Ferner ist im Hinblick auf den ursprünglich festgelegten Lösungsplan zu berücksichtigen, daß bei der *Fortsetzung* der Erfassung zunächst dafür gesorgt werden muß, daß eine geeignete globale Variable eingerichtet wird, die auf eine Instanziierung der Klasse "WerteErfassungView" weist.

Um dies zu gewährleisten, ist die innerhalb der Klasse "AnforderungenView" festgelegte Methode "vorbereitenFortsetzungsfenster" z.B. wie folgt abzuändern:

```
vorbereitenFortsetzungsfenster
Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang) asSymbol
          put: (WerteErfassungView new).
(Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang) asSymbol)
   initialisierenErfassung;
   festlegenUeberschrift: ('Jahrgangsstufe ',Jahrgang).
self lesenDatei: (Prompter prompt:'' default:'').
(Smalltalk at: (Skalenniveau,'WerteErfassung',Jahrgang) asSymbol)
   durchfuehrenErfassung
```

Hinweis: Der Name der Datei, in der die zuvor erfaßten Punktwerte als Datenstrom gesichert worden sind, wird durch das Argument von "lesenDatei:" bestimmt.

Wird die Methode "vorbereitenFortsetzungsfenster" in dieser Weise vereinbart, so wird ein Sammler zur Übernahme der zuvor erfaßten Punktwerte eingerichtet. In diesen Sammler werden alle innerhalb der Datei gesicherten Punktwerte übertragen. Insgesamt garantiert die soeben vorgestellte Änderung für den Fall, daß die Anwendung allein als Laufzeit-System zur Ausführung gelangen soll, eine geeignete Lösung der Problemstellung PROB-9.

14.2.4 Weiterer Einsatz von Datenströmen

Anzeige von Objekten

Wie zuvor erwähnt, besteht die Möglichkeit, Datenströme nicht nur zur Sicherung von Daten in einer Datei, sondern auch – für besondere Anwendungen – zur temporären Sicherung zu verwenden.

Von dieser Möglichkeit wird z.B. systemseitig Gebrauch gemacht, wenn ein Ergebnis-Objekt einer Anforderung unter Einsatz der Menü-Option "Show It" des Menüs "Smalltalk" zur Anzeige gebracht werden soll.

In dieser Situation wird vom SMALLTALK-System automatisch die Methode "printString" ausgeführt, die folgendermaßen innerhalb der Klasse "Object" vereinbart ist:

```
printString
|aStream aString|
RecursiveSet := Set new.
aString := String new: 20.
self printOn: (aStream := WriteStream on: aString).
∧ aStream contents
```

Hierbei wird auf die ebenfalls in der Klasse "Object" vereinbarte Methode "printOn:" zurückgegriffen, die wie folgt festgelegt ist:

```
printOn: aStream
|aString|
aString := self class name.
(aString at: 1) isVowel
    ifTrue: [aStream nextPutAll: 'an ']
    ifFalse: [aStream nextPutAll: 'a '].
aStream nextPutAll: aString
```

Hinweis: Die innerhalb der Methode "printOn:" eingesetzten Messages "name" bzw. "isVowel" dienen dazu, den Klassennamen des Empfänger-Objekts als Zeichenkette zu ermitteln bzw. zu prüfen, ob es sich bei einem Zeichen um einen Vokal handelt.

Wollen wir z.B. eine aussagefähigere Darstellung als "an InWerteErfassung" erhalten, wenn wir uns Objekte anzeigen lassen, die als Instanziierungen einer Unterklasse von "WerteErfassung" eingerichtet wurden, so sollten wir innerhalb der Klasse "WerteErfassung" die Methode "printOn:" z.B. in der folgenden Form redefinieren:

```
printOn: aStream
aStream nextPutAll: self class name.
aStream nextPutAll: ' Inhalt von werteBag: '.
self bereitstellenWerte do:
  [:einObjekt|aStream nextPutAll: einObjekt,' ']
```

Bestätigen wir in dieser Situation – nach der Markierung von "InWerteErfassung11" – die Menü-Option "Show It", so erhalten wir für "InWerteErfassung11" die Anzeige

```
InWerteErfassung Inhalt von werteBag: 32 37
```

sofern die Punktwerte "32" und "37" gesammelt worden sind.

Weitere Methoden zur Bearbeitung von Datenströmen

In der oben vorgestellten Vereinbarung der Methode "printString" wurde die Anforderung "WriteStream on: aString" verwendet. Durch deren Bearbeitung wird ein Datenstrom als Instanziierung der Basis-Klasse "WriteStream" eingerichtet. Da "aString" stellvertretend für eine Instanz der Klasse "String" steht, ist bestimmt, daß der Datenstrom aus Zeichenketten aufgebaut werden soll.

- **"on:"**:
  Wird die Message "on:" der Klasse "WriteStream" zugestellt, so wird ein Datenstrom in Form einer Instanz aus der Klasse des Empfänger-Objekts eingerichtet, der auf dem als Argument angegebenen geordneten Sammler basiert.

Die Klassen-Methode "on:" steht nicht nur für die Basis-Klasse "WriteStream", sondern auch für die Basis-Klassen "ReadStream" und "ReadWriteStream" zur Verfügung, die in der folgenden Weise der abstrakten Klasse "Stream" untergeordnet sind:

```
Object
  Stream
    ReadStream
    WriteStream
      ReadWriteStream
```

Diese Klassen unterscheiden sich im Hinblick auf die Möglichkeiten, wie auf die in den zugehörigen Datenströmen eingetragenen Objekte zugegriffen werden darf.

Während auf einen Datenstrom aus der Klasse "ReadStream" *nur lesend* und auf einen Datenstrom der Klasse "WriteStream" *nur schreibend* zugegriffen werden kann, besteht bei einem Datenstrom der Klasse "ReadWriteStream" die Möglichkeit, sowohl lesend als auch schreibend zuzugreifen.

Neben den zuvor bereits von uns eingesetzten Methoden "on:", "pathName:", "nextPutAll:", "space", "close", "atEnd" und "nextWord" lassen sich – im Hinblick auf andere Formen von Anwendungen – z.B. auch die folgenden Methoden einsetzen, die jeweils von einem Datenstrom – als Empfänger-Objekt der korrespondierenden Messages – ausgeführt werden können:

Hinweis: Durch "R" bzw. "W" wird gekennzeichnet, daß die betreffende Methode sinnvollerweise nur von einem Empfänger-Objekt aus der Klasse "ReadStream" bzw. "WriteStream" verwendet werden kann. Grundsätzlich läßt sich jede Methode von einer Instanz der Klasse "ReadWriteStream" ausführen.

- **“contents”**: (W,R)
  Als Ergebnis-Objekt resultieren sämtliche Objekte des Datenstroms.

- **“size”**: (R)
  Als Ergebnis-Objekt wird die Anzahl der Objekte ermittelt, die im Datenstrom enthalten sind.

- **“reset”**: (W,R)
  Es wird der Positionszeiger des Datenstroms vor das erste Objekt gesetzt, indem ihm der Wert “0” zugeordnet wird.

- **“skip:”**: (W,R)
  Es wird der Positionszeiger um die als Argument angegebene Stellenzahl (positive oder negative Ganzzahl) versetzt.

- **“position”**: (W,R)
  Es wird der Positionszeiger als Ergebnis-Objekt ermittelt.

- **“position:”**: (W,R)
  Der Positionszeiger wird an die als Argument angegebene Stelle versetzt.

- **“setToEnd”**: (W,R)
  Der Positionszeiger wird auf das letzte Objekt des Datenstroms gerichtet.

- **“next”**: (R)
  Der Positionszeiger wird auf das nachfolgende Objekt im zugehörigen Datenstrom versetzt und dasjenige Objekt als Ergebnis-Objekt erhalten, auf das der Positionszeiger weist.

- **“peek”**: (R)
  Als Ergebnis-Objekt resultiert dasjenige Objekt des Datenstroms, auf das der Positionszeiger weist.

- **“nextPut:”**: (W)
  Der Positionszeiger wird auf das nachfolgende Objekt versetzt und das als Argument von “nextPut:” angegebene Objekt im Datenstrom an der aktuellen Position eingetragen.

- **“cr”**: (W)
  In den Datenstrom werden die beiden Steuerzeichen für den Zeilenvorschub (Carriage-Return “Cr” und Line-Feed “Lf”) eingetragen.

- **“nextLine”**: (R)
  Der Positionszeiger wird im Datenstrom bis zum nächsten Steuerzeichen für den Zeilenvorschub versetzt und als Ergebnis-Objekt diejenige Zeichenkette ermittelt, die vor dem Positionszeiger eingetragen ist.

Kapitel 15

Programmierung von Suchverfahren

15.1 Prüfung von Direktverbindungen

Problemstellung

Bislang haben wir die Strategien der objekt-orientierten Programmierung und den Einsatz des SMALLTALK-Systems überwiegend an einem Beispiel aus dem Bereich der Datenerfassung und der Berechnung von einfachen statistischen Kennwerten kennengelernt. Um zu demonstrieren, daß sich der Ansatz der objekt-orientierten Programmierung auch in gänzlich anderen Anwendungsbeispielen vorzüglich zur Programmierung von Lösungsplänen eignet, wollen wir mit dem erworbenen Kenntnisstand ein klassisches Problemfeld bearbeiten, in dem die Entwicklung von Algorithmen im Vordergrund steht. Dabei werden wir feststellen, wie vorteilhaft sich eine Strukturierung im Rahmen des Model/View-Konzeptes verwirklichen läßt.

Die uns interessierenden Algorithmen sind Gegenstand des Anwendungsbereichs der 'Künstlichen Intelligenz" (KI). In diesem Bereich geht es vorwiegend um die Lösung von Problemstellungen, deren Lösungspläne sich nicht durch einen *konstruktiven* Algorithmus beschreiben lassen. Dies bedeutet, daß eine Problemlösung nur dadurch erhalten werden kann, daß mögliche Lösungen durchprobiert werden. Man sagt, daß derartige KI-Probleme durch die "Versuch-und-Irrtum-Strategie" gelöst werden. Bei dieser Strategie wird – durch den Einsatz geeigneter Suchverfahren – systematisch versucht, mögliche Lösungen eines Problems zu finden.
Als Beispiel für eine Aufgabenstellung, bei deren Lösung sich ein derartiger Ansatz anbietet, betrachten wir zunächst das folgende Problem:

- PROB-10:
  Es sind Anfragen nach *direkten* IC-Verbindungen zu beantworten. Dies bedeutet, daß es möglich sein soll, eine Auskunft darüber zu erhalten, ob es zwischen zwei Stationen eine direkte Zugverbindung im IC-Netz gibt oder nicht.

Dazu legen wir den folgenden Ausschnitt eines stark vereinfachten IC-Netzes der Deutschen Bahn AG zugrunde:

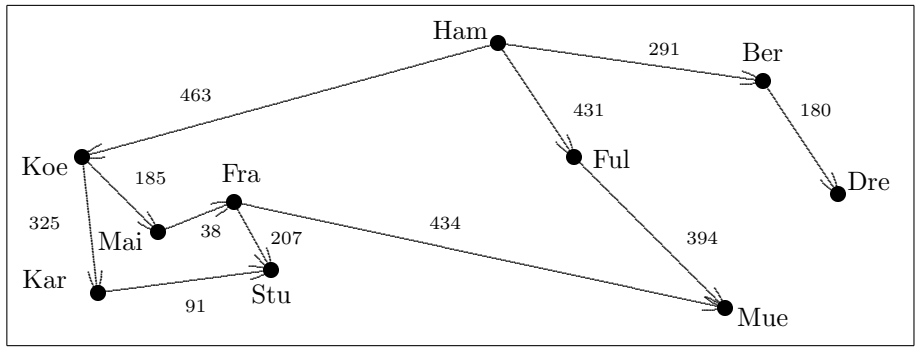


Abbildung 15.1: IC-Netz

Dieses Netz enthält die Stationen "Ham" (Hamburg), "Koe" (Köln), "Kar" (Karlsruhe), "Stu" (Stuttgart), "Mai" (Mainz), "Fra" (Frankfurt), "Mue" (München), "Ful" (Fulda), "Ber" (Berlin) und "Dre" (Dresden).

Hinweis: Wir orientieren uns zunächst an den eingetragenen Richtungen – unabhängig davon, daß es im "richtigen Bahnnetz" natürlich auch die Verbindungen in der Gegenrichtung gibt. Die in der Abbildung angegebenen Entfernungsangaben werden wir erst später berücksichtigen (siehe Abschnitt 15.2.4).

Lösungsansatz

Da *nur* Direktverbindungen angefragt werden sollen, liegt es nahe, bei der Festlegung des in der Zeichnung dargestellten Sachverhalts folgendermaßen vorzugehen:

- Wir ordnen den Bahnstationen jeweils einen Sammler zu, in dem als Objekte *sämtliche* Stationen eingetragen sind, zu denen eine *Direktverbindung* existiert. Unter der Voraussetzung, daß "( )" einen leeren Sammler kennzeichnet, bedeutet dies, daß die folgenden Zuordnungen zu treffen sind:

Ham	→	(Koe Ful Ber)
Koe	→	(Kar Mai)
Kar	→	(Stu)
Stu	→	( )
Mai	→	(Fra)
Fra	→	(Stu Mue)
Mue	→	( )
Ful	→	(Mue)
Ber	→	(Dre)
Dre	→	( )

Anschließend muß bei Vorgabe eines Abfahrts- und Ankunftsortes geprüft werden, ob ein dem Abfahrtsort in dieser Weise zugeordneter Sammler nicht leer ist. Sofern dies der Fall ist, muß untersucht werden, ob der Ankunftsort ein Objekt dieses Sammlers ist.

Die Klassen “Suchverfahren” und “Station”

Zur Lösung von PROB-10 sehen wir die folgende hierarchische Strukturierung vor:

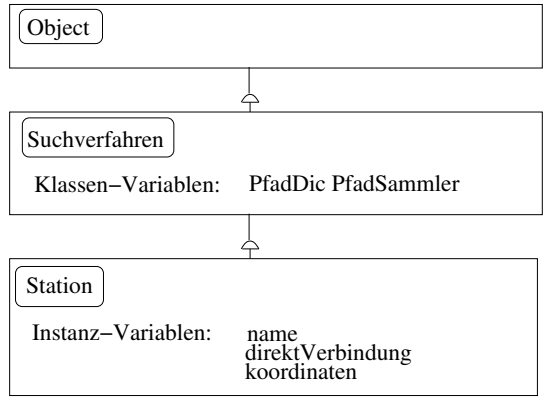


Abbildung 15.2: Hierarchische Strukturierung

Die Klasse “Suchverfahren”, die wir als direkte Unterklasse der Klasse “Object” vereinbaren, soll die Methoden enthalten, mit denen Suchverfahren durchführbar sind.

Hinweis: In dieser Klasse sehen wir für eine spätere Verwendung die Klassen-Variablen “PfadDic” und “PfadSammler” vor.

Zur Beschreibung der Objekte, die Gegenstand der Suchverfahren sind, richten wir die Klasse “Station” als Unterklasse von “Suchverfahren” ein. In dieser Klasse vereinbaren wir die Instanz-Variablen “name”, “direktVerbindung” und “koordinaten”. Dabei soll die Instanz-Variable “name” die Bezeichnung einer Station und die Instanz-Variable “direktVerbindung” die von einer Station ausgehenden Direktverbindungen aufnehmen.

Hinweis: Die Instanz-Variable “koordinaten” setzen wir später für eine grafische Darstellung des IC-Netzes ein.

Methoden zur Instanziierung der Stationen

Zur Instanziierung der Stationen des IC-Netzes vereinbaren wir in der Klasse “Station” die folgende Klassen-Methode namens “erzeugenInstanzen”:

```
erzeugenInstanzen
Ham := Station new eintragenName: 'Hamburg'.
Koe := Station new eintragenName: 'Köln'.
Kar := Station new eintragenName: 'Karlsruhe'.
Stu := Station new eintragenName: 'Stuttgart'.
Mai := Station new eintragenName: 'Mainz'.
Fra := Station new eintragenName: 'Frankfurt'.
```

```
Mue := Station new eintragenName: 'München'.
Ful := Station new eintragenName: 'Fulda'.
Ber := Station new eintragenName: 'Berlin'.
Dre := Station new eintragenName: 'Dresden'.
Ham eintragenDirektverbindung: Koe.
Ham eintragenDirektverbindung: Ful.
Ham eintragenDirektverbindung: Ber.
Koe eintragenDirektverbindung: Kar.
Koe eintragenDirektverbindung: Mai.
Kar eintragenDirektverbindung: Stu.
Mai eintragenDirektverbindung: Fra.
Fra eintragenDirektverbindung: Stu.
Fra eintragenDirektverbindung: Mue.
Ful eintragenDirektverbindung: Mue.
Ber eintragenDirektverbindung: Dre
```

Dabei setzen wir die Instanz-Methoden "eintragenName:" und "eintragenDirektverbindung:" ein, die wir ebenfalls in der Klasse "Station" durch

```
eintragenName: einName
direktVerbindung := OrderedCollection new.
name := einName
```

bzw. durch

```
eintragenDirektverbindung: einObjekt
direktVerbindung add: einObjekt
```

festlegen.
Um für eine Instanz der Klasse "Station" – z.B. beim Einsatz der Methode "inspect" – eine aussagefähigere Anzeige als "OrderedCollection(a Station)" zu erhalten, redefinieren wir in der Klasse "Station" die Basis-Methode "printOn:" in der folgenden Form:

```
printOn: aStream
aStream nextPutAll: name
```

Für den Zugriff auf die einer Station zugeordneten Direktverbindungen legen wir die Instanz-Methode "bereitstellenDirektverbindung" zusätzlich wie folgt fest:

```
bereitstellenDirektverbindung
∧ direktVerbindung
```

Insgesamt haben wir somit die folgende Situation:

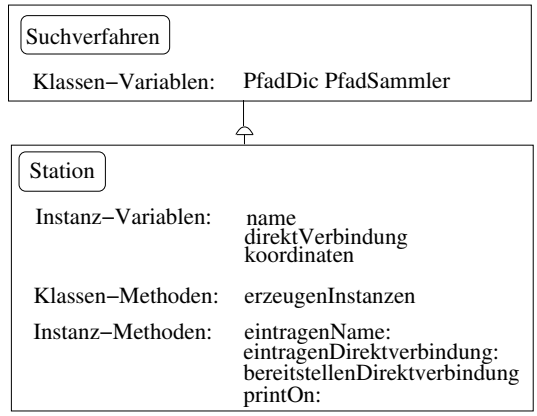


Abbildung 15.3: Die Klassen “Suchverfahren” und “Station”

Auf dieser Basis richten wir die Stationen “Ham”, “Koe”, “Kar”, “Stu”, “Mai”, “Fra”, “Mue”, “Ful”, “Ber” und “Dre” als Instanzen der Klasse “Station” durch die folgende Anforderung ein:

```
Station erzeugenInstanzen
```

Somit haben wir z.B. für die globale Variable “Ham” die folgende Situation:

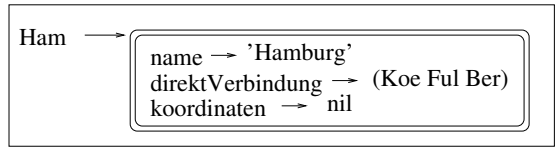


Abbildung 15.4: Attribute der Station “Ham”

Auskunft über Direktverbindungen

Zur Prüfung, ob eine *Direktverbindung* von einem Abfahrtsort zu einem Ankunftsort besteht, vereinbaren wir in der Klasse “Suchverfahren” die Instanz-Methode “direkt:” in der folgenden Form:

```
direkt: ankunft
(self bereitstellenDirektverbindung includes: ankunft)
  ifTrue: [Transcript cr;show:'Direktverbindung existiert']
  ifFalse: [Transcript cr;show:'Direktverbindung existiert nicht']
```

Als Ergebnis der Anforderung

```
Ham direkt: Koe
```

erhalten wir daraufhin im Transcript-Fenster den Text

```
Direktverbindung existiert
```

angezeigt.

15.1.1 Ein iterativer und ein rekursiver Lösungsplan

Problemstellung

Im folgenden wollen wir Methoden entwickeln, mit denen nicht nur das Bestehen von Direktverbindungen, sondern von *beliebigen* IC-Verbindungen untersucht werden kann. Daher stellen wir uns die folgende Aufgabe:

- PROB-11:
  Es sind Anfragen nach IC-Verbindungen zu beantworten, bei denen der Abfahrtsort *nicht direkt* mit dem Ankunftsort verbunden sein muß, d.h. die Verbindung kann über eine oder mehrere *Zwischenstationen* führen.

In der nachfolgenden Darstellung werden wir zunächst einen intuitiven Lösungsansatz verfolgen. Daran anschließend stellen wir die klassischen Suchverfahren in Form der Breiten-, der Tiefen- und der Bestwegsuche vor.

Lösungsplan

Die Prüfung, ob es eine Verbindung vom Abfahrtsort zum Ankunftsort gibt, soll auf dem folgenden Lösungsplan basieren:

- Bilde einen Nachfolger-Sammler, der zunächst nur den Abfahrtsort enthält!
- Ersetze den Abfahrtsort durch seine direkten Nachfolger (Nachfolger 1. Grades)!
- Ist der Ankunftsort *kein* Objekt des Nachfolger-Sammlers, so ersetze die Nachfolger 1. Grades *sämtlich* durch ihre direkten Nachfolger (Nachfolger 2. Grades)!
- Ist der Ankunftsort wiederum kein Objekt des so erhaltenen Sammlers, so fahre mit der Ersetzung fort!
- Ist der Ankunftsort erstmals im Nachfolger-Sammler enthalten, so ist das Suchverfahren *erfolgreich* beendet!
- Ist der Ankunftsort in keinem der ermittelten Nachfolger-Sammler enthalten und kann letztendlich für keines der Sammler-Objekte ein weiterer Nachfolger ermittelt werden, so erhalten wir einen *leeren* Nachfolger-Sammler. Damit ist das Suchverfahren *erfolglos* beendet.

Insgesamt können wir die Lösung von PROB-11 wie folgt beschreiben:

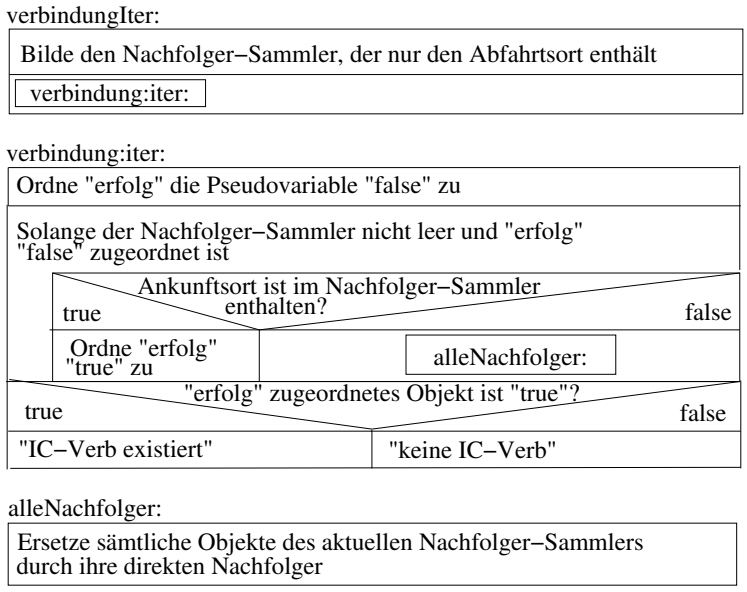


Abbildung 15.5: Struktogramme zur Lösung von PROB-11

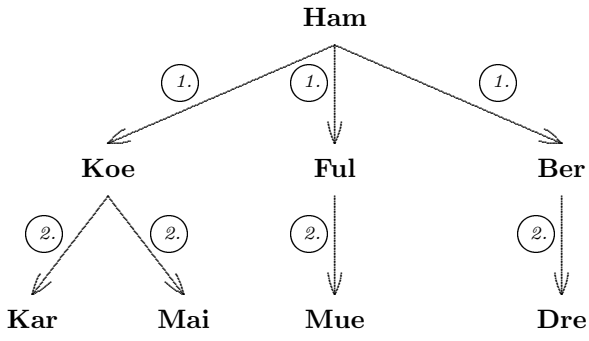
- Zur besseren Darstellung leiten wir jedes Struktogramm mit einer Überschrift ein, die mit dem Selektor derjenigen Methode übereinstimmt, mit der der durch das Struktogramm beschriebene Ablauf zur Ausführung gelangen soll.
- Ferner tragen wir in einem Struktogramm an der Stelle, an der eine Methode ausgeführt werden soll, einen Strukturblock ein, der den jeweiligen Methoden-Selektor mit einer rechteckigen Umrahmung enthält.

Bei der Prüfung, ob z.B. eine IC-Verbindung von “Ham” nach “Mue” existiert, ist somit folgendermaßen vorzugehen:

Abfahrtsort:				Ham	
Nachfolger-Sammler:				(Ham)	
Nachfolger-Sammler:		(Koe		Ful	Ber)
Nachfolger-Sammler:	(Kar		Mai	Mue	Dre)

Der erste Nachfolger-Sammler wird aus “Ham” und der zweite Nachfolger-Sammler aus den direkten Nachfolgern des Abfahrtsortes “Ham” aufgebaut. Der dritte Nachfolger-Sammler wird dadurch erhalten, daß die Station “Koe” durch ihre direkten Nachfolger “Kar” und “Mai”, die Station “Ful” durch ihren direkten Nachfolger “Mue” und die Station “Ber” durch ihren direkten Nachfolger “Dre” ersetzt wird. Der Ankunftsort “Mue” ist bereits in diesem Nachfolger-Sammler als Objekt enthalten, so daß der Suchprozeß in dieser Situation *erfolgreich* beendet wird.

Das durchgeführte Suchverfahren läßt sich insgesamt wie folgt beschreiben:



Nachfolger-Sammler:

a) (Ham)
b) (Koe Ful Ber)
c) (Kar Mai Mue Dre)

Hinweis: In dem Graphen haben wir durch die eingekreisten Nummern die Reihenfolge gekennzeichnet, in der die möglichen Nachfolger bestimmt werden.

- Bei diesem Suchverfahren wird ein *gerichteter Graph* – als Suchgraph – aufgebaut, der aus Knoten besteht, die durch *gerichtete Kanten* miteinander verbunden sind. Jede Kante legt die Nachfolgerbeziehung zwischen zwei Knoten fest.

Ein Knoten ohne Vorgänger – wie in diesem Fall "Ham" – wird *Startknoten* (Wurzelknoten) genannt. Ein direkter Nachfolger wird als "Knoten 1. Grades" bezeichnet, ein direkter Nachfolger eines Knotens 1. Grades als "Knoten 2. Grades", usw. Ein Knoten, der einen Zielzustand – wie z.B. der Knoten "Mue" – beschreibt, wird *Zielknoten* genannt. Die Lösung des Suchverfahrens entspricht einem Weg, der den Startknoten mit dem Zielknoten verbindet. Um von einem Knoten die zugehörigen Nachfolgerknoten zu erhalten, muß der Knoten *expandiert* werden, d.h. zur jeweiligen Station sind die zugehörigen *direkten* Nachfolgerstationen zu bestimmen.

Hinweis: Da in diesem Netz jeder Knoten – mit Ausnahme des Abfahrtsortes – *einen* Vorgängerknoten hat, wird der Suchgraph auch *Baum* genannt.

Iterative Lösung von PROB-11

Im Hinblick auf den oben als Struktogramm angegebenen Lösungsplan vereinbaren wir in der Klasse "Suchverfahren" die Methode "alleNachfolger:" wie folgt:

```
alleNachfolger: sammlerVonStationen
| nachfolgerSammler |
nachfolgerSammler := OrderedCollection new.
sammlerVonStationen do: [:einObjekt|
  nachfolgerSammler addAll: einObjekt bereitstellenDirektverbindung].
^ nachfolgerSammler
```

Die Methoden "verbindungIter:" und "verbindung:iter:" vereinbaren wir in der Form

```
verbindungIter: ankunft
| nachfolgerSammler |
nachfolgerSammler := OrderedCollection new.
nachfolgerSammler add: self.
self verbindung: nachfolgerSammler iter: ankunft
```

bzw. wie folgt:

```
verbindung: nachfolgerSammler iter: ankunft
| varSammler erfolg |
varSammler := nachfolgerSammler.
erfolg := false.
[ (erfolg == false) and: [varSammler isEmpty not] ]
  whileTrue: [(varSammler includes: ankunft)
            ifTrue: [erfolg := true]
            ifFalse:[varSammler := self alleNachfolger: varSammler]].
erfolg ifTrue: [Transcript cr; show:'IC-Verb existiert']
       ifFalse:[Transcript cr; show:'keine IC-Verb']
```

Hinweis: Da das Argument "nachfolgerSammler" der Methode "verbindung:iter:" nicht auf der linken Seite einer Zuweisung auftreten darf, müssen wir den jeweils aktuellen Nachfolger-Sammler der temporären Variablen "varSammler" zuweisen.

Bei der Ausführung der Methode "verbindung:iter:" werden – solange der aktuelle Nachfolger-Sammler nicht leer ist – durch den Einsatz der Methode "whileFalse:" gleiche Anforderungen wiederholt ausgeführt.

Dabei setzen wir zur Ermittlung des jeweils aktuellen Nachfolger-Sammlers die Message "alleNachfolger:" mit dem zuletzt ermittelten Nachfolger-Sammler als Argument ein und weisen das Ergebnis-Objekt dieser Message jeweils der temporären Variablen "varSammler" durch die Anforderung

```
varSammler := self alleNachfolger: varSammler
```

sukzessive zu.

Im folgenden wollen wir eine elegantere Lösung vorstellen, bei der wir sowohl auf die Programmierung einer Schleife als auch auf die Zuweisung an die Variable "varSammler" verzichten. Dies läßt sich dadurch erreichen, daß der jeweils aktuelle Nachfolger-Sammler als Argument einer rekursiven Methode aufgeführt wird.

Der Begriff der "rekursiven" Methode

Eine Methode, deren Selektor im eigenen Methodenrumpf innerhalb einer Anforderung auftritt, nennt man eine *rekursive* Methode.

Charakteristisch für die Ausführung einer rekursiven Methode ist es, daß immer die gleichen Anforderungen – jedoch mit unterschiedlichen Werten in den Argumenten der Methode – ausgeführt werden.
Damit gesichert ist, daß dieser Prozeß zu irgendeinem Zeitpunkt abbricht, muß innerhalb einer rekursiven Methode mindestens ein *Abbruchkriterium* enthalten sein.

Im Normalfall wird das Abbruchkriterium als erste Anforderung innerhalb der Methode angegeben, so daß es *vor* einem weiteren rekursiven Methodenaufruf derselben Methode ausgeführt und somit geprüft werden muß.

Hinweis: Da bei jedem rekursiven Aufruf ein Methodenexemplar im Hauptspeicher eingerichtet wird, kann es vorkommen, daß der Speicherplatz nicht ausreicht. Das SMALLTALK-System eröffnet in diesem Fall ein Walkback-Fenster (siehe Anhang A.3), das die Titel-Zeile
`Smalltalk Express stack overflow`
enthält.

Rekursive Lösung von PROB-11

Um PROB-11 durch den Einsatz einer rekursiven Methode lösen zu können, werden wir die in der ursprünglichen iterativen Lösung verwendeten Methoden "verbindungIter:" und "verbindung:iter:" wie folgt durch die Methode "verbindungRek:" und die rekursive Methode "verbindung:rek:" ersetzen:

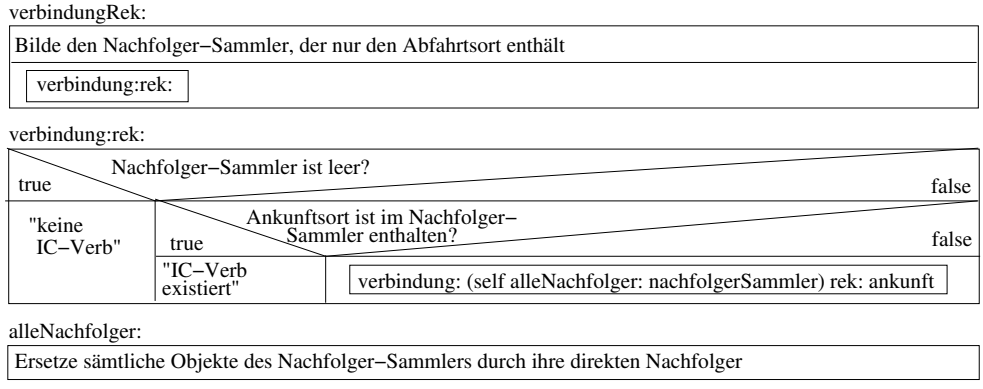


Abbildung 15.6: Struktogramme für die rekursive Lösung von PROB-11

Die ersten beiden Struktogramme setzen wir durch die Vereinbarung der Methode "verbindungRek:" in der Form

```
verbindungRek: ankunft
| nachfolgerSammler |
nachfolgerSammler := OrderedCollection new.
```

```
nachfolgerSammler add: self.
self verbindung: nachfolgerSammler rek: ankunft
```

und durch die Methode "verbindung:rek:" in der folgenden Form um:

```
verbindung: nachfolgerSammler rek: ankunft
(nachfolgerSammler isEmpty)
  ifTrue:  [Transcript cr;show:'keine IC-Verb']
  ifFalse: [(nachfolgerSammler includes: ankunft)
     ifTrue: [Transcript cr;show:'IC-Verb existiert']
     ifFalse: [self
                verbindung:(self alleNachfolger: nachfolgerSammler)
                rek: ankunft]
           ]
```

Nach der Vereinbarung dieser Methoden stellt sich die Klasse "Suchverfahren" zum jetzigen Zeitpunkt folgendermaßen dar:

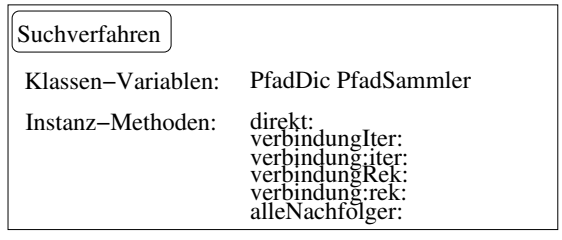


Abbildung 15.7: Die Klasse "Suchverfahren" mit den aktuellen Methoden

Hinweis: Fügen wir innerhalb der Methoden-Vereinbarung von "verbindung:rek:" die Anforderung

`Transcript cr; show: nachfolgerSammler printString.`

als 1. Anforderung ein, so erhalten wir z.B. durch die Ausführung von

`Ham verbindungRek: Mue`

im Transcript-Fenster zusätzlich die Texte

```
OrderedCollection(Hamburg)
OrderedCollection(Köln Fulda Berlin)
OrderedCollection(Karlsruhe Mainz München Dresden)
```

angezeigt.

15.1.2 Prüfung durch Breitensuche

Bei dem bisherigen Lösungsplan wurde der Nachfolger-Sammler dadurch bestimmt, daß für *sämtliche* Objekte des zuvor bestimmten Nachfolger-Sammlers die jeweils direkten Nachfolger ermittelt wurden.

Dieses Verfahren hat den Nachteil, daß in dem Fall, in dem bereits eine IC-Verbindung festgestellt werden kann, weitere Methodenaufrufe aktiviert werden, obwohl dies nicht erforderlich ist.
Deshalb wollen wir die Strategie, stets *alle* Objekte eines Nachfolger-Sammlers *gleichzeitig* zu expandieren und durch ihre Nachfolger zu ersetzen, wie folgt – in Form einer *Breitensuche* – modifizieren:

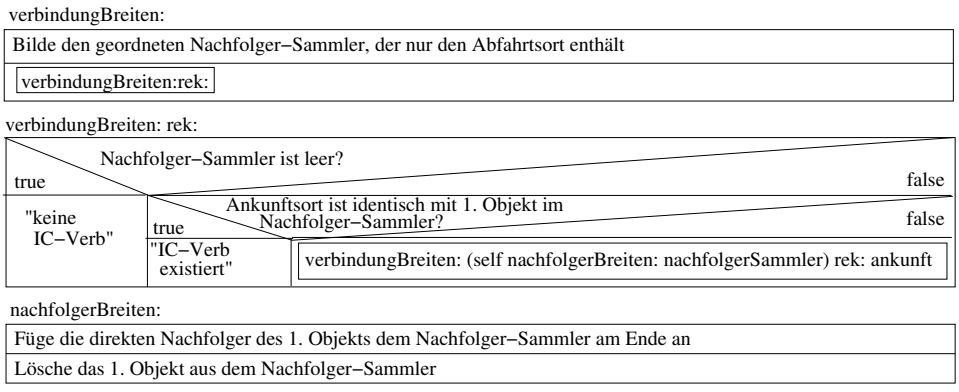
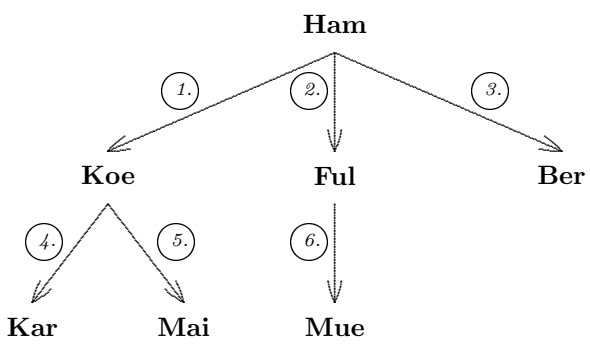


Abbildung 15.8: Struktogramme für die Breitensuche

Die Ausführung der Breitensuche läßt sich z.B. für die Anfrage, ob eine IC-Verbindung zwischen "Ham" und "Mue" besteht, folgendermaßen graphisch veranschaulichen:



Nachfolger-Sammler:

a) (Ham)
b) (Koe Ful Ber)
c) (Ful Ber Kar Mai)
d) (Ber Kar Mai Mue)
e) (Kar Mai Mue Dre)
f) (Mai Mue Dre Stu)
g) (Mue Dre Stu Fra)

Hinweis: Bei der Angabe der Nachfolger-Sammler haben wir nicht nur deren Reihenfolge angegeben, sondern zusätzlich bestimmte Objekte unterstrichen. Diese Objekte stellen die direkten Nachfolger derjenigen Objekte dar, die im unmittelbar vorausgehenden Nachfolger-Sammler die jeweils erste Position einnehmen.

Bei der Umsetzung der Struktogramme vereinbaren wir die Methode "nachfolgerBreiten:" durch

```
nachfolgerBreiten: sammlerVonStationen
| erstesOb |
```

```
erstesOb := sammlerVonStationen first.
sammlerVonStationen addAll: erstesOb bereitstellenDirektverbindung.
sammlerVonStationen remove: erstesOb.
^ sammlerVonStationen
```

und die Methode “verbindungBreiten:” durch

```
verbindungBreiten: ankunft
| nachfolgerSammler |
nachfolgerSammler := OrderedCollection new.
nachfolgerSammler add: self.
self verbindungBreiten: nachfolgerSammler rek: ankunft
```

sowie die Methode “verbindungBreiten:rek:” durch:

```
verbindungBreiten: nachfolgerSammler rek: ankunft
(nachfolgerSammler isEmpty)
  ifTrue:  [Transcript cr;show:'keine IC-Verb']
  ifFalse: [(nachfolgerSammler includes: ankunft)
     ifTrue: [Transcript cr;show:'IC-Verb existiert']
     ifFalse: [self verbindungBreiten:
                      (self nachfolgerBreiten:nachfolgerSammler)
                  rek: ankunft]
 ]
```

Nach diesen Vereinbarungen besitzt die Klasse “Suchverfahren” den folgenden Inhalt:

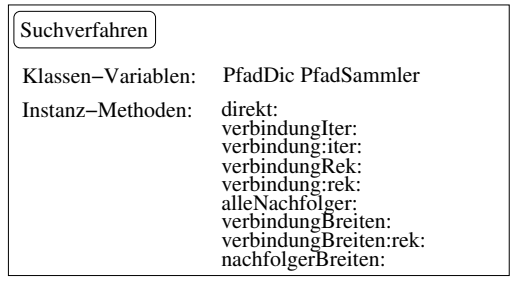


Abbildung 15.9: Die Klasse “Suchverfahren” mit den aktuellen Methoden

15.1.3 Prüfung durch Tiefensuche

Eine weitere Suchstrategie besteht darin, die Suche – unabhängig vom Grad des Nachfolgers – immer mit dem zuerst erreichten Knoten fortzusetzen. Daraus resultiert ein Suchverfahren, das sich mehr in die Tiefe als in die Breite entwickelt. Dieses

Verfahren, das *"Tiefensuche"* genannt wird, läßt sich dadurch realisieren, daß die Nachfolger des 1. Sammler-Objektes an den *Sammleranfang* eingefügt werden und die Suche wiederum mit dem jeweils neuen ersten Sammler-Objekt fortgesetzt wird. Eine derartige Tiefensuche läßt sich durch die folgenden Struktogramme beschreiben:

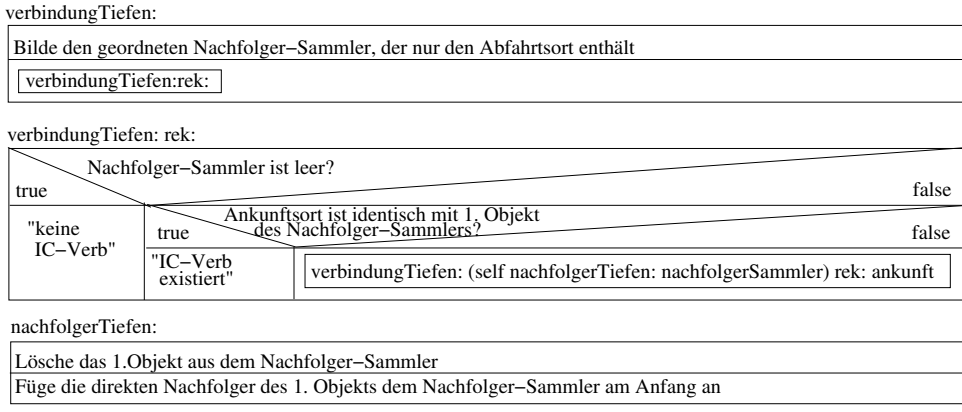
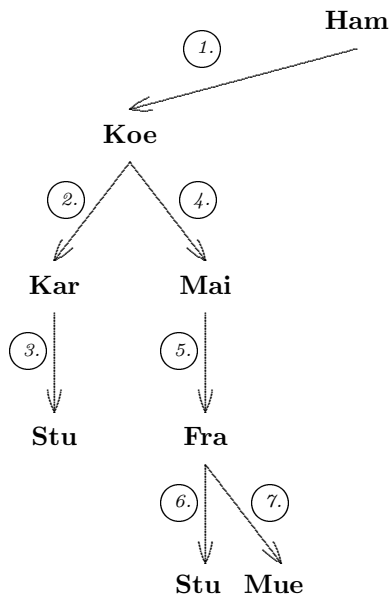


Abbildung 15.10: Struktogramme für die Tiefensuche

Der Suchgraph für die Tiefensuche stellt sich z.B. bei der Prüfung, ob es eine IC-Verbindung von "Ham" nach "Mue" gibt, wie folgt dar:



Nachfolger-Sammler:

a) (Ham)
b) (Koe Ful Ber)
c) (Kar Mai Ful Ber)
d) (Stu Mai Ful Ber)
e) (Mai Ful Ber)
f) (Fra Ful Ber)
g) (Stu Mue Ful Ber)
h) (Mue Ful Ber)

Hinweis: Es ist zu beachten, daß der Knoten "Stu" nicht expandiert werden kann.

Aus der Darstellung ist zu erkennen, daß die Kanten des IC-Netzes – von links beginnend – so tief wie möglich nach unten hin durchsucht werden. Immer dann, wenn

ein Knoten erreicht wird, der *keine* Nachfolger hat, wird zum unmittelbar vorausgehenden Verzweigungsknoten im Suchgraphen zurückgesetzt und gegebenenfalls von dort aus die nächste Direktverbindung des IC-Netzes überprüft.
Gemäß der Struktogramme vereinbaren wir die Methode "nachfolgerTiefen:" durch

```
nachfolgerTiefen: sammlerVonStationen
| erstesObjekt nachfolgerSammler |
nachfolgerSammler := OrderedCollection new.
erstesObjekt := sammlerVonStationen first.
sammlerVonStationen remove: erstesObjekt.
nachfolgerSammler addAll: erstesObjekt bereitstellenDirektverbindung.
sammlerVonStationen do: [:einObjekt|nachfolgerSammler add: einObjekt].
∧ nachfolgerSammler
```

und die Methode "verbindungTiefen:" durch

```
verbindungTiefen: ankunft
| nachfolgerSammler |
nachfolgerSammler := OrderedCollection new.
nachfolgerSammler add: self.
self verbindungTiefen: nachfolgerSammler rek: ankunft
```

sowie die Methode "verbindungTiefen:rek:" durch:

```
verbindungTiefen: nachfolgerSammler rek: ankunft
(nachfolgerSammler isEmpty)
  ifTrue:  [Transcript cr;show:'keine IC-Verb']
  ifFalse: [(nachfolgerSammler includes: ankunft)
     ifTrue: [Transcript cr;show:'IC-Verb existiert']
     ifFalse: [self verbindungTiefen:
                      (self nachfolgerTiefen:nachfolgerSammler)
                   rek: ankunft]
           ]
```

Insgesamt ist die Klasse "Suchverfahren" jetzt wie folgt aufgebaut:

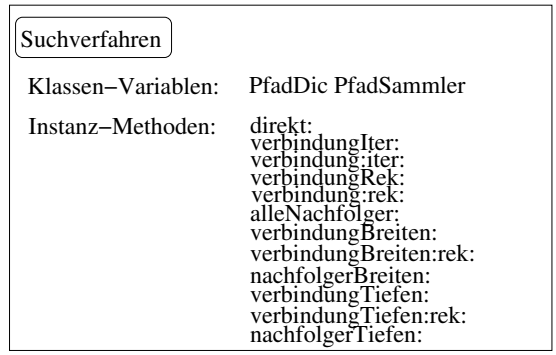


Abbildung 15.11: Die Klasse "Suchverfahren"

15.1.4 Prüfung durch Bestwegsuche

Problemstellung

Bei der Vereinbarung der in den letzten Abschnitten beschriebenen Methoden haben wir auf die Ermittlung der Zwischenstationen einer IC-Verbindung verzichtet. Um jeweils sämtliche Stationen einer IC-Verbindung zu erhalten, stellen wir uns die folgende Aufgabe:

- PROB-12:
  Es sollen Methoden entwickelt werden, die die jeweils *kürzeste* IC-Verbindung vom Abfahrts- zum Ankunftsort ermitteln und die jeweiligen Stationen anzeigen.

  Hinweis: Dabei setzen wir – im Unterschied zu den zuvor vorgestellten Suchverfahren – voraus, daß es die in Abbildung 15.1 angegebenen Direktverbindungen auch in der Gegenrichtung gibt.

Zur Bestimmung der *kürzesten* IC-Verbindung setzen wir ein Suchverfahren ein, das die Entfernung als problemspezifische Information nutzt. Da zur Auswahl der jeweils nächsten Zwischenstation das Kriterium "Minimierung der Entfernung vom Abfahrtsort zum Ankunftsort" dienen soll, ist der Lösungsplan als *Bestwegsuche* zu entwickeln.

Hinweis: Die jeweils ermittelten Zwischenstationen sind allein dann nicht eindeutig bestimmt, wenn es zwei unterschiedliche IC-Verbindungen mit gleicher Gesamtentfernung zwischen einem Abfahrts- und einem Ankunftsort gibt.

Um die jeweils kürzeste IC-Verbindung ermitteln zu können, müssen wir die Entfernungsangaben der Direktverbindungen für das in Abb. 15.1 angegebene IC-Netz in geeigneter Form festlegen. Dabei gehen wir wie folgt vor:

- Wir ordnen den Stationen jeweils einen Sammler zu, dessen Objekte wiederum Sammler sind, in denen jeweils der Nachfolger und die zugehörige Länge der Direktverbindung eingetragen ist.

Somit sind die folgenden Zuordnungen zu treffen:

Ham	→	((Koe 463) (Ful 431) (Ber 291))
Koe	→	((Kar 325) (Mai 185) (Ham 463))
Kar	→	((Stu 91) (Koe 325))
Stu	→	((Fra 207) (Kar 91))
Mai	→	((Fra 38) (Koe 185))
Fra	→	((Stu 207) (Mue 434) (Mai 38))
Mue	→	((Fra 434) (Ful 394))
Ful	→	((Mue 394) (Ham 431))
Ber	→	((Dre 180) (Ham 291))
Dre	→	((Ber 180))

Eintragen von Entfernungen

Genau wie bei den zuvor vorgestellten Suchverfahren gehen wir auch jetzt wieder davon aus, daß die Stationen bereits als Instanzen der Klasse "Station" eingerichtet worden sind (siehe Abschnitt 15.1).

Um die angegebene Struktur für die Sammler festzulegen, vereinbaren wir die Methode "initialisierenDirektverbindung" innerhalb der Klasse "Station" in der Form

```
initialisierenDirektverbindung
direktVerbindung := OrderedCollection new
```

und lassen die folgende Anforderung ausführen:

```
Station allInstances do:
    [:eineStation|eineStation initialisierenDirektverbindung]
```

Im Hinblick auf eine spätere grafische Darstellung des IC-Netzes sehen wir in der Klasse "Station" die folgende Methode vor:

```
eintragenKoordinaten: einPunkt
koordinaten := einPunkt
```

Um das IC-Netz mit den Stationen und den Entfernungsangaben der Direktverbindungen aufbauen zu können, vereinbaren wir zunächst die Klassen-Methode "eintragenKoordinatenNeuDirektVerbindung" wie folgt:

```
eintragenKoordinatenNeuDirektverbindung
Ham eintragenKoordinaten: (150 @ 30).
Ham eintragenDirektverbindung: (OrderedCollection with: Koe with: 463).
Ham eintragenDirektverbindung: (OrderedCollection with: Ful with: 431).
Ham eintragenDirektverbindung: (OrderedCollection with: Ber with: 291).
Koe eintragenKoordinaten: (20 @ 90).
Koe eintragenDirektverbindung: (OrderedCollection with: Kar with: 325).
Koe eintragenDirektverbindung: (OrderedCollection with: Mai with: 185).
Koe eintragenDirektverbindung: (OrderedCollection with: Ham with: 463).
```

```
Kar eintragenKoordinaten: (30 @ 190).
Kar eintragenDirektverbindung: (OrderedCollection with: Stu with: 91).
Kar eintragenDirektverbindung: (OrderedCollection with: Koe with: 325).
Stu eintragenKoordinaten: (120 @ 180).
Stu eintragenDirektverbindung: (OrderedCollection with: Fra with: 207).
Stu eintragenDirektverbindung: (OrderedCollection with: Kar with: 91).
Mai eintragenKoordinaten: (50 @ 140).
Mai eintragenDirektverbindung: (OrderedCollection with: Fra with: 38).
Mai eintragenDirektverbindung: (OrderedCollection with: Koe with: 185).
Fra eintragenKoordinaten: (100 @ 110).
Fra eintragenDirektverbindung: (OrderedCollection with: Stu with: 207).
Fra eintragenDirektverbindung: (OrderedCollection with: Mue with: 434).
Fra eintragenDirektverbindung: (OrderedCollection with: Mai with: 38).
Mue eintragenKoordinaten: (260 @ 190).
Mue eintragenDirektverbindung: (OrderedCollection with: Fra with: 434).
Mue eintragenDirektverbindung: (OrderedCollection with: Ful with: 394).
Ful eintragenKoordinaten: (190 @ 100).
Ful eintragenDirektverbindung: (OrderedCollection with: Mue with: 394).
Ful eintragenDirektverbindung: (OrderedCollection with: Ham with: 431).
Ber eintragenKoordinaten: (240 @ 60).
Ber eintragenDirektverbindung: (OrderedCollection with: Dre with: 180).
Ber eintragenDirektverbindung: (OrderedCollection with: Ham with: 291).
Dre eintragenKoordinaten: (290 @ 120).
Dre eintragenDirektverbindung: (OrderedCollection with: Ber with: 180)
```

Anschließend bringen wir diese Methode durch die Anforderung

```
Station eintragenKoordinatenNeuDirektverbindung
```

zur Ausführung.
Somit haben wir z.B. für die Station “Ham” die folgende Situation:

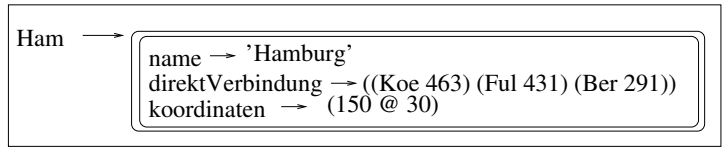
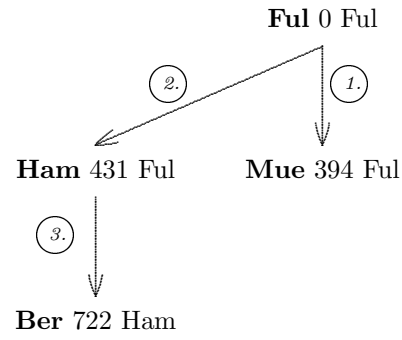


Abbildung 15.12: Attribute der Station “Ham”

Lösungsplan

Um die Bestwegsuche programmieren zu können, stellen wir die von uns gewählte Strategie zunächst an einem Beispiel dar. Dazu betrachten wir die folgende Skizze, die das Auffinden der kürzesten Verbindung zwischen "Ful" und "Ber" beschreibt:



- Grundsätzlich sollen sämtliche noch nicht expandierte Knoten in einem *Knoten-Sammler* festgehalten werden.

- In einem *Geprüft-Sammler* sollen Informationen über bereits expandierte Knoten in der Form

  #(nachfolger-knoten entfernung vorgänger-knoten)

  gesammelt werden.

  Hinweis: Zu Beginn der Bestwegsuche ist der Geprüft-Sammler gleich dem leeren Sammler.

- In einem *Erreicht-Sammler* sollen als Objekte alle diejenigen Sammler der Form

  #(nachfolger-knoten entfernung vorgänger-knoten)

  eingetragen werden, für die "nachfolger-knoten" ein bereits als Nachfolger ermittelter Knoten ist, der jedoch bislang noch nicht expandiert wurde.

  Hinweis: Wir werden den Erreicht-Sammler so aufbauen, daß dessen Objekte automatisch aufsteigend nach ihrer Entfernung sortiert werden. Dazu setzen wir eine Instanziierung der Klasse "SortedCollection" mit einem Sortier-Block der Form

  ```
  [:x :y | (x at:2) < (y at:2)]
  ```

  ein.

Damit stellt sich für das Beispiel die folgende Ausgangssituation dar:

Knoten-Sammler:	(Ham, Koe, Mai, Ful, Fra, Kar, Stu, Mue, Ber, Dre)
Geprüft-Sammler:	()
Erreicht-Sammler:	(#(Ful 0 Ful))

- Zunächst ist aus den Sammler-Objekten des aktuellen Erreicht-Sammlers ein *Minimal-Sammler* zu bestimmen. Dieser Sammler ist dadurch festgelegt, daß das erste Objekt des Erreicht-Sammlers – an seiner ersten Position – den *minimalen Knoten* enthält. Dabei ist der minimale Knoten dadurch ausgezeichnet, daß ihm die geringste Entfernungsangabe unter allen Knoten des Erreicht-Sammlers zugeordnet ist.

  Hinweis: Sofern es mehrere Knoten mit gleicher Entfernungsangabe gibt, wird der zuerst ermittelte Knoten weiter betrachtet.

Im Beispiel enthält der Erreicht-Sammler nur ein Sammler-Objekt, so daß "Ful" als minimaler Knoten und "#(Ful 0 Ful)" als zugehöriger Minimal-Sammler ermittelt wird.

- Anschließend ist zu prüfen, ob der Ankunftsort mit dem minimalen Knoten übereinstimmt.

  Hinweis: Dies kann bei der erstmaligen Prüfung nicht der Fall sein, sofern der Ankunftsort nicht mit dem Abfahrtsort übereinstimmt.

- Fällt diese Prüfung *positiv* aus, so ist die Bestwegsuche zu beenden, nachdem die Zwischenstationen, die zur ermittelten kürzesten IC-Verbindung gehören, mit Hilfe des Geprüft-Sammlers bestimmt worden sind.
  Stimmt der Ankunftsort *nicht* mit dem minimalen Knoten überein, so ist für den minimalen Knoten zu prüfen, ob er noch im Knoten-Sammler enthalten ist und ob er Nachfolger besitzt.

  Hinweis: Durch die Prüfung, ob der minimale Knoten noch im Knoten-Sammler enthalten ist, erreichen wir, daß jeder Knoten und dessen Direktverbindungen höchstens einmal betrachtet werden.

Im Beispiel ist "Ful" noch im Knoten-Sammler enthalten, und es lassen sich somit die Nachfolger "Ham" und "Mue" ermitteln.

- Da der minimale Knoten im Knoten-Sammler enthalten ist, wird für jeden Nachfolger-Knoten ein *potentieller Minimal-Sammler* aufgebaut. Dieser Sammler soll wiederum Sammler-Objekte der Form

  #(nachfolger-knoten entfernung vorgänger-knoten)

  enthalten, sofern Nachfolger zum minimalen Knoten existieren.
  In diesen Sammler-Objekten ist der jeweilige Nachfolger-Knoten, der (bisherige) minimale Knoten als Vorgänger-Knoten und die neue Entfernung einzutragen. Dabei ergibt sich die neue Entfernung als Summe aus der Länge der Direktverbindung vom (bisherigen) minimalen Knoten zum Nachfolger-Knoten und der aktuellen Entfernung in dem (bisherigen) Minimal-Sammler.

Durch die Expandierung von "Ful" ergibt sich als neuer potentieller Minimal-Sammler:

(#(Mue 394+0 Ful) #(Ham 431+0 Ful))

- Um den (bisherigen) Minimal-Sammler zu sichern, ist er als neues Objekt in den Geprüft-Sammler aufzunehmen.

Wenden wir diese Vorschrift auf unser Beispiel an, so erhalten wir als neuen Geprüft-Sammler:

neuer Geprüft-Sammler: (#(Ful 0 Ful))

- Anschließend wird der bisherige Minimal-Sammler in dem Erreicht-Sammler durch die Gesamtheit aller potentiellen Minimal-Sammler ersetzt.

Die Ersetzung des bisherigen Minimal-Sammlers "#(Ful 0 Ful)" durch den potentiellen Minimal-Sammler "(#(Mue 394 Ful) #(Ham 431 Ful))" ergibt einen Erreicht-Sammler (vor Löschung) in der Form:

(#(Mue 394 Ful) #(Ham 431 Ful))

- Damit der Erreicht-Sammler so klein wie möglich gehalten wird, sind sämtliche Objekte aus dem Erreicht-Sammler zu löschen, die als jeweils 1. Objekt eine Station enthalten, die bereits im Geprüft-Sammler enthalten ist.

Im Beispiel ist der zuvor ermittelte Erreicht-Sammler bereits in seiner endgültigen Form, da die Station "Ful" aus dem Geprüft-Sammler von allen Stationen des Erreicht-Sammlers verschieden ist und somit keine Löschungen in dem Erreicht-Sammler vorzunehmen sind. Also gilt:

neuer Erreicht-Sammler (nach Löschung):(#(Mue 394 Ful) #(Ham 431 Ful))

- Letztlich ist aus dem Knoten-Sammler diejenige Station zu entfernen, die mit dem minimalen Knoten korrespondiert.

Auf der Basis der oben formulierten Ausgangskonstellation

Knoten-Sammler:	(Ham, Koe, Kar, Stu, Mai, Fra, Mue, Ful, Ber, Dre)
Geprüft-Sammler:	()
Erreicht-Sammler:	(#(Ful 0 Ful))

lassen sich die angegebenen Verfahrensschritte folgendermaßen zusammenfassen:

Minimal-Sammler:	#(Ful 0 Ful)
minimaler Knoten:	Ful
Ankunftsort "Ber" stimmt nicht mit "Ful" überein	
Nachfolger:	Ham, Mue
potentieller Minimal-Sammler:	(#(Mue 394+0 Ful) #(Ham 431+0 Ful))
neuer Geprüft-Sammler:	(#(Ful 0 Ful))
neuer Erreicht-Sammler: (vor Löschung)	(#(Mue 394 Ful) #(Ham 431 Ful))
neuer Erreicht-Sammler: (nach Löschung)	(#(Mue 394 Ful) #(Ham 431 Ful))
aus dem Knoten-Sammler wird der Eintrag "Ful" gelöscht	

Auf der Basis von

Knoten-Sammler:	(Ham, Koe, Kar, Stu, Mai, Fra, Mue, Ber, Dre)
Geprüft-Sammler:	(#(Ful 0 Ful))
Erreicht-Sammler:	(#(Mue 394 Ful) #(Ham 431 Ful))

ergeben sich bei der Fortsetzung der Bestwegsuche die folgenden Werte:

Minimal-Sammler:	#(Mue 394 Ful)
minimaler Knoten:	Mue
Ankunftsort "Ber" stimmt nicht mit "Mue" überein	
Nachfolger:	Fra, Ful
potentieller Minimal-Sammler:	(#(Ful 394+394 Mue) #(Fra 434+394 Mue))
neuer Geprüft-Sammler:	(#(Ful 0 Ful) #(Mue 394 Ful))
neuer Erreicht-Sammler: (vor Löschung)	(#(Ham 431 Ful) #(Ful 394+394 Mue) #(Fra 434+394 Mue))
neuer Erreicht-Sammler: (nach Löschung)	(#(Ham 431 Ful) #(Fra 828 Mue))
aus dem Knoten-Sammler wird der Eintrag "Mue" gelöscht	

Eine weitere Wiederholung des Verfahrensschrittes führt auf der Basis von

Knoten-Sammler:	(Ham, Koe, Kar, Stu, Mai, Fra, Ber, Dre)
Geprüft-Sammler:	(#(Ful 0 Ful) #(Mue 394 Ful))
Erreicht-Sammler:	(#(Ham 431 Ful) #(Fra 828 Mue))

zu den folgenden Ergebnissen:

Minimal-Sammler:	#(Ham 431 Ful)
minimaler Knoten:	Ham
Ankunftsort "Ber" stimmt nicht mit "Ham" überein	
Nachfolger:	Koe, Ful, Ber
potentieller Minimal-Sammler:	(#(Ber 291+431 Ham) #(Ful 431+431 Ham) #(Koe 463+431 Ham))
neuer Geprüft-Sammler:	(#(Ful 0 Ful) #(Mue 394 Ful) #(Ham 431 Ful))
neuer Erreicht-Sammler: (vor Löschung)	(#(Ber 291+431 Ham) #(Fra 828 Mue) #(Ful 431+431 Ham) #(Koe 463+431 Ham))
neuer Erreicht-Sammler: (nach Löschung)	(#(Ber 722 Ham) #(Fra 828 Mue) #(Koe 894 Ham))
aus dem Knoten-Sammler wird der Eintrag "Ham" gelöscht	

Im nächsten Verfahrensschritt wird festgestellt, daß der minimale Knoten "Ber" identisch mit dem Ankunftsort ist. Somit ist eine IC-Verbindung von "Ful" nach "Ber" ermittelt, und die Bestwegsuche wird erfolgreich mit den folgenden Sammlern beendet:

Minimal-Sammler:	#(Ber 722 Ham)
Knoten-Sammler:	(Koe, Kar, Stu, Mai, Fra, Ber, Dre)
Geprüft-Sammler:	(#(Ful 0 Ful) #(Mue 394 Ful) #(Ham 431 Ful))
Erreicht-Sammler:	(#(Ber 722 Ham) #(Fra 828 Mue) #(Koe 894 Ham))

Aus dem aktuellen Minimal-Sammler ist abzulesen, daß “722” die Gesamtentfernung vom Abfahrtsort “Ful” zum Ankunftsort “Ber” ist. Aus dem Minimal-Sammler können wir ferner erkennen, daß die Station “Ham” die Vorgänger-Station von “Ber” ist. Mit dieser Station können wir dann anschließend – durch “Zurückhangeln” innerhalb des Geprüft-Sammlers – die Zwischenstationen bestimmen, die auf der ermittelten IC-Verbindung zum Abfahrtsort “Ful” hin vorliegen. Dabei zeigt sich, daß die Vorgänger-Station von “Ham” identisch mit dem Abfahrtsort “Ful” ist. Somit ist “Ham” die einzige Zwischenstation auf der kürzesten IC-Verbindung von “Ful” nach “Ber”.

Lösung von PROB-12

Das oben angegebene Vorgehen bei der Bestwegsuche läßt sich durch die folgenden Struktogramme beschreiben:

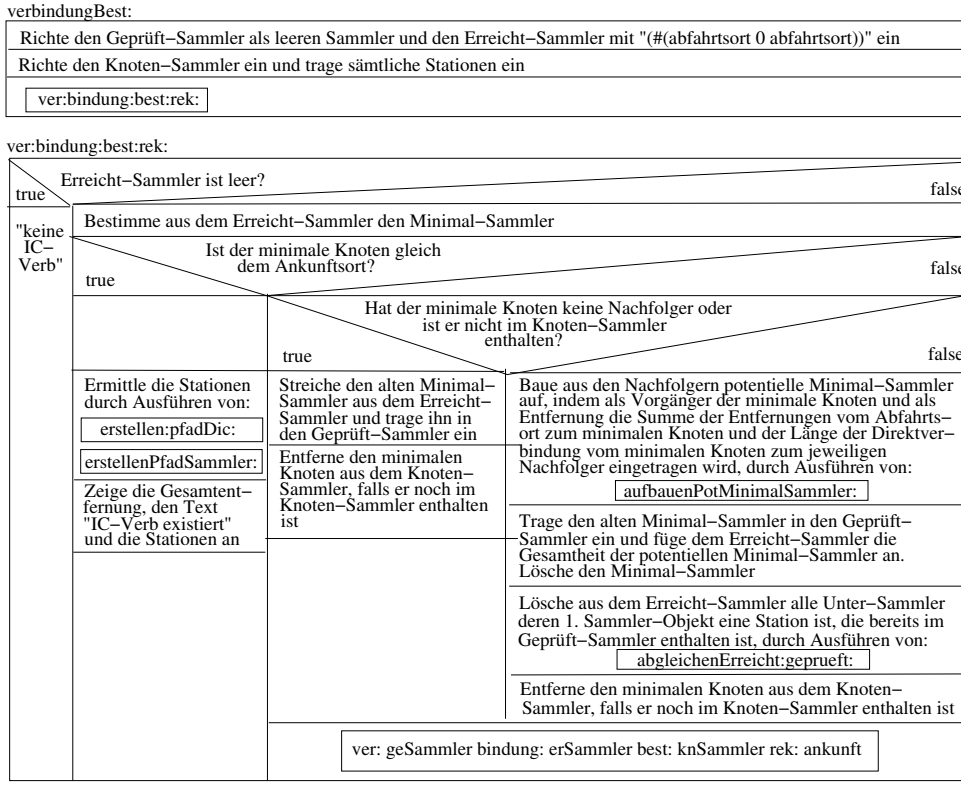


Abbildung 15.13: Struktogramme für die Bestwegsuche

Auf dieser Basis vereinbaren wir Methoden innerhalb der Klasse “Suchverfahren”.

Für die Methode “verbindungBest:” definieren wir:

```
verbindungBest: ankunft
| feld geprueftSammler erreichtSammler knotenSammler |
geprueftSammler := Set new.
feld := Array new: 3.
feld at: 1 put: self; at:2 put: 0; at:3 put: self.
erreichtSammler :=  SortedCollection
                      sortBlock:[:x :y|(x at:2)<(y at:2)].
erreichtSammler add: feld.
knotenSammler := Set new.
Station allInstances do:[:eineStation|knotenSammler add:eineStation].
self ver: geprueftSammler bindung: erreichtSammler
     best: knotenSammler rek: ankunft
```

Die Methode “ver:bindung:best:rek:” vereinbaren wir wie folgt:

```
ver: geSammler bindung: erSammler best: knSammler rek: ankunft
| minimalSammler potMinimalSammler neuerErSammler |
(erSammler isEmpty)
 ifTrue: [Transcript cr;show:'keine IC-Verb']

 ifFalse: [minimalSammler := erSammler first.
          (minimalSammler at: 1) == ankunft
  ifTrue: [self erstellen: geSammler pfadDic: minimalSammler.
           self erstellenPfadSammler: ankunft.
           Transcript cr;show:'IC-Verb existiert';
             show:'Abstand: ';show:(minimalSammler at:2) printString;
             cr; show: PfadSammler reversed printString]
  ifFalse:[( (minimalSammler at:1)
               bereitstellenDirektverbindung isEmpty
           or: [(knSammler includes: (minimalSammler at:1)) not] )
      ifTrue: [erSammler remove: minimalSammler.
               geSammler add: minimalSammler.
               knSammler remove:(minimalSammler at:1) ifAbsent: [ ]
              ]
      ifFalse:[potMinimalSammler := (minimalSammler at: 1)
                          aufbauenPotMinimalSammler:minimalSammler.
               geSammler add: minimalSammler.
               erSammler addAll: potMinimalSammler.
               erSammler remove: minimalSammler.
               neuerErSammler :=
                 self abgleichenErreicht:erSammler geprueft:geSammler.
               knSammler remove: (minimalSammler at: 1) ifAbsent: [ ]
               ].
      self ver: geSammler bindung: neuerErSammler
           best: knSammler rek: ankunft
          ]
         ]
```

Die Methode "aufbauenPotMinimalSammler:" legen wir wie folgt fest:

```
aufbauenPotMinimalSammler: minSammler
| feld  sammler |
sammler := Set new.
self bereitstellenDirektverbindung do:
   [:einObjekt|feld := Array new:3.
               feld at:1 put: (einObjekt at:1);
                    at:2 put: (einObjekt at:2) + (minSammler at:2);
                    at:3 put: self.
               sammler add: feld].
^ sammler
```

Die Methode "abgleichenErreicht:geprueft:" vereinbaren wir wie folgt:

```
abgleichenErreicht: erSammler geprueft: geSammler
| sammler1 sammler2 |
sammler1:=geSammler collect:[:einArray|einArray at: 1].
sammler2:=SortedCollection sortBlock:[:x :y|(x at:2)<(y at:2)].
erSammler do: [:einArray |( sammler1 includes: (einArray at:1) )
                            ifTrue:  [ ]
                            ifFalse: [sammler2 add:einArray]].
^ sammler2
```

Um die Zwischenstationen eines Bestweges vom Abfahrtsort zum Ankunkftsort sowie die zugehörige Entfernung zu ermitteln, vereinbaren wir noch die beiden Methoden "erstellen:pfadDic:" und "erstellenPfadSammler:".
Dazu legen wir zunächst fest:

```
erstellen: geSammler pfadDic: minimalSammler
PfadDic at: (minimalSammler at: 1) put: minimalSammler.
geSammler do: [:einArray | PfadDic at:(einArray at:1) put:einArray]
```

Hinweis: Durch den Einsatz der Methode "erstellen:pfadDic:" erreichen wir, daß in der Klassen-Variablen "PfadDic" (als Dictionary eingerichtet) der Klasse "Suchverfahren" das jeweilige erste Objekt der Unter-Sammler des Minimal-Sammlers und des Geprüft-Sammlers als *Key* eingetragen wird.
Als *Value* wird der jeweils zugehörige gesamte Unter-Sammler des Geprüft-Sammlers eingetragen.

Anschließend verabreden wir:

```
erstellenPfadSammler: station
| varArray |
varArray := PfadDic at: station.
```

```
(station == self)
   ifFalse:[PfadSammler add: station.
            self erstellenPfadSammler:(varArray at:3)]
   ifTrue: [PfadSammler add:(varArray at:3)]
```

Hinweis: Somit greifen wir ausgehend vom Ankunftsort auf den Vorgänger dieser Station zu und hangeln uns solange zurück, bis wir den Abfahrtsort erreicht haben.
Um im Transcript-Fenster die Stationen in der richtigen Reihenfolge anzuzeigen, schicken wir – innerhalb von "ver:bindung:best:rek:" – dem Empfänger-Objekt "PfadSammler" eine Message mit dem Selektor "reversed".

Um vor jeder neuen Anfrage den bisher ermittelten kürzesten Pfad in den Klassen-Variablen "PfadDic" und "PfadSammler" der Klasse "Suchverfahren" löschen zu können, vereinbaren wir die Klassen-Methode "initialisierenKlassenvariablen" innerhalb der Klasse "Suchverfahren" in der folgenden Form:

```
initialisierenKlassenvariablen
PfadDic := Dictionary new.
PfadSammler := OrderedCollection new
```

Insgesamt besitzt die Klasse "Suchverfahren" jetzt den folgenden Inhalt:

Abbildung 15.14: Die Klasse "Suchverfahren" mit den aktuellen Methoden

Ausführung der Bestwegsuche

Die Ausführung der Bestwegsuche – bezogen auf die kürzeste IC-Verbindung von "Ful" nach "Ber" – können wir durch die Ausführung der folgenden Anforderungen abrufen:

```
Station allInstances do:
    [:eineStation|eineStation initialisierenDirektverbindung].
Station eintragenKoordinatenNeuDirektverbindung.
Suchverfahren initialisierenKlassenvariablen.
Ful verbindungBest: Ber
```

Im Transcript-Fenster erhalten wir daraufhin den Text

```
IC-Verb existiert Abstand: 722
OrderedCollection(Fulda Hamburg Berlin)
```

angezeigt.

15.2 Bestwegsuche und grafische Anzeige von Verbindungen

Problemstellung

Nachdem wir die Programmierung der Suchverfahren durchgeführt haben, wollen wir abschließend beschreiben, wie sich das Ergebnis einer Bestwegsuche grafisch – in einem eigenständigen Anfrage-Fenster – darstellen läßt.
Dazu stellen wir uns die folgende Aufgabe:

- PROB-13:
  Es ist die jeweils *kürzeste* IC-Verbindung vom Abfahrts- zum Ankunftsort zu ermitteln und diese Verbindung im IC-Netz grafisch anzuzeigen. Außerdem soll es möglich sein, für jede der Stationen eines Bestweges die Entfernung vom Abfahrtsort abzurufen.

Zur Lösung von PROB-13 greifen wir auf die zuvor entwickelten Methoden der Klasse “Suchverfahren” zurück. Insbesondere gehen wir im folgenden davon aus, daß die Stationen als Objekte mit den jeweils benötigten Eigenschaften durch Instanziierungen der Klasse “Station” eingerichtet worden sind.

Aufbau des Anfrage-Fensters

Das Anfrage-Fenster, in dem die gesuchten Informationen abgerufen und angezeigt werden sollen, gliedern wir gemäß der folgenden Skizze:

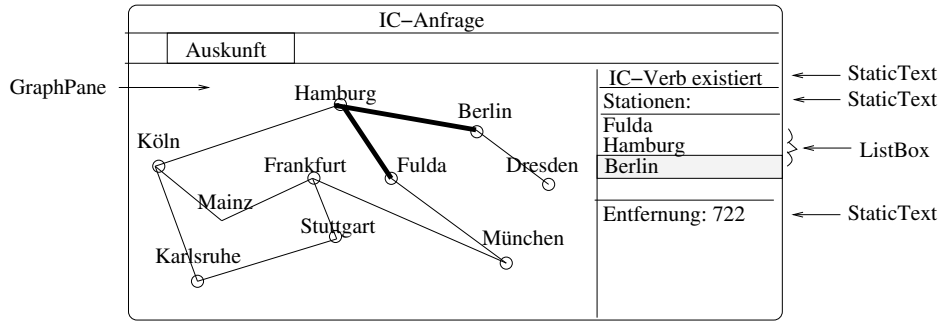


Abbildung 15.15: Struktur des Anfrage-Fensters

Diese Anzeige basiert darauf, daß die kürzeste Verbindung zwischen dem Abfahrtsort "Ful" und dem Ankunftsort "Ber" bestimmt werden sollte.
Das Anfrage-Fenster besteht aus einer Titel-Zeile mit dem Text "IC-Anfrage", drei Textfeldern ("StaticText"), einem Grafikfeld ("GraphPane") und einem Listenfeld ("ListBox").
Im Grafikfeld soll das betrachtete IC-Netz zusammen mit dem jeweils ermittelten kürzesten Weg angezeigt werden. In dem ersten der drei Textfelder soll mitgeteilt werden, ob eine IC-Verbindung existiert ("IC Verb existiert") oder nicht ("keine IC Verb").

Existiert eine IC-Verbindung zwischen dem Abfahrtsort und dem Ankunftsort, so soll im zweiten Textfeld der Text "Stationen:" erscheinen. In diesem Fall sollen die Namen der Stationen im Hinblick auf die ermittelte IC-Verbindung angezeigt werden. Dabei soll es möglich sein, durch Mausklicks sukzessiv die Entfernung vom Abfahrtsort zu jeder der angezeigten Stationen abzurufen.
Innerhalb der Menü-Leiste des Anfrage-Fensters ist das Menü "Auskunft" vorgesehen, mit dessen Menü-Optionen eine weitere IC-Auskunft eingeholt oder Anfragen an das IC-Netz beendet werden können.

Methoden-Vereinbarung zum Aufbau des Anfrage-Fensters

Zum Aufbau des Anfrage-Fensters und zur Steuerung der Kommunikation vereinbaren wir – gemäß des Model/View-Konzeptes – die Klasse "ICAnfrageView" als Unterklasse der Basis-Klasse "ViewManager".

Zur Festlegung des benötigten Views sehen wir die Methode "initialisierenAnfrageView" in der folgenden Form vor:

```
initialisierenAnfrageView
self addView: (self topPaneClass new owner: self;
    labelWithoutPrefix: 'IC-Anfrage';
    noSmalltalkMenuBar;
    viewName: 'ic-fenster';
    framingRatio:((Rectangle leftTopUnit rightAndDown:0@0)
                    extentFromLeftTop: (1/2)@(1/2));
```

```
    when: #menuBuilt perform: #vereinbarungMenuLeiste:;
    addSubpane: (GraphPane new owner: self;
       framingRatio:((Rectangle leftTopUnit rightAndDown:0@0)
                        extentFromLeftTop: (3/4)@1);
       paneName: 'ic-netz');
     addSubpane: (StaticText new owner: self;
       framingRatio:((Rectangle leftTopUnit rightAndDown:(3/4)@0)
                        extentFromLeftTop: (1/4)@(1/16));
       paneName: 'ic-ueberschrift1');
     addSubpane: (StaticText new owner: self;
       framingRatio:((Rectangle leftTopUnit rightAndDown:(3/4)@(1/16))
                        extentFromLeftTop: (1/4)@(1/16));
       paneName: 'ic-ueberschrift2');
     addSubpane: (ListBox new owner: self;
       framingRatio:((Rectangle leftTopUnit rightAndDown:(3/4)@(2/16))
                        extentFromLeftTop: (1/4)@(7/16));
       paneName: 'ic-zwischenstationen';
       when: #select perform: #anzeigenEntfernung:);
    addSubpane: (StaticText new owner: self;
       framingRatio:((Rectangle leftTopUnit rightAndDown:(3/4)@(9/16))
                        extentFromLeftTop: (1/4)@1);
       paneName: 'ic-entfernung')
              )
```

Hinweis: Der Fenster-Baustein "Grafikfeld" wird in Form einer Instanz der Klasse "GraphPane" eingerichtet.

Zum Aufbau der Menü-Leiste vereinbaren wir die folgende Methode:

```
vereinbarungMenuLeiste: aView
(aView menuWindow)
    removeMenu: (aView menuTitled: 'File');
    addMenu:
      (Menu new owner: self;
         title: 'Auskunft';
         appendItem: 'Neue  Anfrage' selector: #fortsetzenAnfragen;
         appendItem: 'Beenden' selector: #beendenAnfragen)
```

Durch die Menü-Optionen "Neue Anfrage" und "Beenden" sollen die Methoden "fortsetzenAnfragen" bzw. "beendenAnfragen" zur Ausführung gelangen. Diese Methoden legen wir durch

```
fortsetzenAnfragen
self close.
ICAnfrageView new initialisierenAnfrageView; anfragenPrompter
```

sowie durch die folgende Vereinbarung fest:

```
beendenAnfragen
self close.
Station allInstances do: [:eineStadt|eineStadt become: nil]
```

Um einen Abfahrts- und einen Ankunftsort mitteilen zu können, soll ein Prompter verwendet werden. Dieser Prompter soll durch die Ausführung der folgenden Methode angezeigt werden:

```
anfragenPrompter
| ab an |
Suchverfahren initialisierenKlassenvariablen.
ab := (Prompter prompt: 'Gib Abfahrt:' default:' ') asSymbol.
an := (Prompter prompt: 'Gib Ankunft:' default:' ') asSymbol.
 ( Smalltalk at: ab ifAbsent: [∧ Transcript cr;
                                show: 'Abfahrtsort existiert nicht'])
 verbindungBest:
 ( Smalltalk at: an ifAbsent: [∧ Transcript cr;
                                show: 'Ankunftsort existiert nicht']).
self anzeigenFenster
```

Es ist erkennbar, daß nach der Eingabe eines zulässigen Abfahrtsortes und Ankunftsortes die Methode "verbindungBest:" – zur Durchführung der Bestwegsuche – zur Ausführung gelangt.
Zur Anzeige des Anfrage-Fensters sehen wir die Methode

```
anzeigenFenster
self openWindow.
self anzeigenErgebnisse
```

und zur Ergebnis-Darstellung innerhalb des Anfrage-Fensters die folgende Methode vor:

```
anzeigenErgebnisse
self anzeigenStationen.
self zeichnenICGrafiken
```

Um die jeweils ermittelten Zwischenstationen im Listenfeld anzeigen zu können, vereinbaren wir:

```
anzeigenStationen
| namenListe |
namenListe := Suchverfahren bereitstellenPfadSammler reversed.
(self paneNamed:'ic-zwischenstationen') contents: namenListe
```

Erfolgt auf eine der im Listenfeld angezeigten Stationen ein Mausklick, so soll – ausgehend vom Abfahrtsort – die Entfernung zu der betreffenden Station angezeigt werden. Dies läßt sich durch die Ausführung der folgenden Methode erreichen:

```
anzeigenEntfernung: aListPane
| station |
station := (self paneNamed:'ic-zwischenstationen') selectedItem.
(self paneNamed:'ic-entfernung') contents:
      ('Entfernung: ',
      ( (Suchverfahren bereitstellenPfadDic at:station) at:2 )
        printString)
```

Um das IC-Netz und die jeweils ermittelte kürzeste Verbindung als Grafik – in dem Grafikfeld – anzuzeigen, setzen wir die folgende Methode ein:

```
zeichnenICGrafiken
| blei |
blei := (self paneNamed: 'ic-netz') pen.
blei erase.
blei drawRetainPicture: [self zeichnenICNetz: blei.
                         self zeichnenICBestweg: blei]
```

Hinweis:

- **"pen"**:
  Durch die Ausführung dieser Methode ordnen wir dem Empfänger-Objekt – in unserer Situation einer Instanz der Klasse "GraphPane" – ein Objekt der Form "Stift" zu, mit dem wir sowohl schreiben als auch zeichnen können.
- **"erase"**:
  Durch den Einsatz dieser Methode wird die jeweilige grafische Anzeige "ausradiert", die dem Empfänger-Objekt zugeordnet ist.
- **"drawRetainPicture:"**:
  Durch die Ausführung dieser Message wird es möglich, einer grafischen Anzeige weitere Grafiken hinzuzufügen. Dabei können die angezeigten Grafiken z.B. innerhalb des Fenster-Bausteins "GraphPane" mit dem Rollbalken verschoben werden.

Innerhalb der Methode "zeichnenICGrafiken" wird die Methode "zeichnenICNetz:" zur Ausführung gebracht, die in der folgenden Form zu vereinbaren ist:

```
zeichnenICNetz: stift
| varDirekt|
stift defaultNib: 1.
Station allInstances do:
[:eineStation|stift place: eineStation gibKoordinaten.
              stift circle:6.
              stift centerText: eineStation gibName].
Station allInstances do:
[:eineStation|varDirekt := eineStation bereitstellenDirektverbindung.
    (varDirekt isEmpty)
    ifFalse: [varDirekt do:
             [:einPaar|stift place: eineStation gibKoordinaten.
                       stift goto: (einPaar at:1) gibKoordinaten]]]
```

Hinweis: Den im folgenden aufgeführten Messages ist gemeinsam, daß sie ein Empfänger-Objekt haben, das die Form eines "Stiftes" besitzt.

- **"defaultNib:"**:
  Durch die Ausführung dieser Methode läßt sich für einen "Stift" eine Strichstärke festlegen.
- **"place:"**:
  Durch diese Methode wird der "Stift" gemäß der aufgeführten Lage-Koordinate positioniert.
- **"circle:"**:
  Durch die Ausführung dieser Methode wird ein Kreis gezeichnet, dessen Radius als Argument angegeben ist.
- **"centerText:"**:
  Durch diese Methode wird die Zeichenkette, die als Argument aufgeführt ist, an der Stelle zentriert geschrieben, die durch die Position des "Stiftes" gekennzeichnet ist.
- **"goto:"**:
  Durch die Ausführung dieser Methode wird eine Linie gezogen. Sie beginnt an der aktuellen Position des "Stiftes" und reicht bis zu der als Argument angegebenen Lage-Koordinate.

Desweiteren wird die Methode "zeichnenICBestweg:" eingesetzt, die wie folgt festzulegen ist:

```
zeichnenICBestweg: stift
| index stadt1 stadt2 varPfadSammler |
index := 1.
stift defaultNib: 5.
varPfadSammler := Suchverfahren bereitstellenPfadSammler.
(varPfadSammler isEmpty)
  ifFalse: [(self paneNamed: 'ic-ueberschrift1')
              contents: 'IC-Verb existiert'.
            (self paneNamed: 'ic-ueberschrift2')
              contents: 'Stationen:'.
            (varPfadSammler size - 1) timesRepeat:
```

```
                [stadt1 := varPfadSammler at: index.
                 stadt2 := varPfadSammler at: (index + 1).
                 stift place: stadt1 gibKoordinaten.
                 stift goto: stadt2 gibKoordinaten.
                 index := index + 1]
            ]
  ifTrue: [(self paneNamed: 'ic-ueberschrift1')
                  contents: 'keine IC-Verb']
```

Bevor die innerhalb der Klasse "ICAnfrageView" insgesamt vereinbarten Methoden ausgeführt werden können, müssen innerhalb der Klasse "Station" noch Methoden festgelegt werden, durch die auf die Namen sowie die Lage-Koordinaten der Stationen zugegriffen werden kann.
Die Stationsnamen werden durch die Methode

```
gibName
∧ name
```

und die Koordinatenangaben durch die Methode

```
gibKoordinaten
∧ koordinaten
```

bereitgestellt.
Damit auf die ermittelten Zwischenstationen und die damit verbundenen Informationen zugegriffen werden kann, muß die Klasse "Suchverfahren" noch um die Klassen-Methode

```
bereitstellenPfadSammler
∧ PfadSammler
```

sowie um die Klassen-Methode

```
bereitstellenPfadDic
∧ PfadDic
```

erweitert werden.

Lösung

Insgesamt erhalten wir für die Lösung von PROB-13 die folgende hierarchische Struktur:

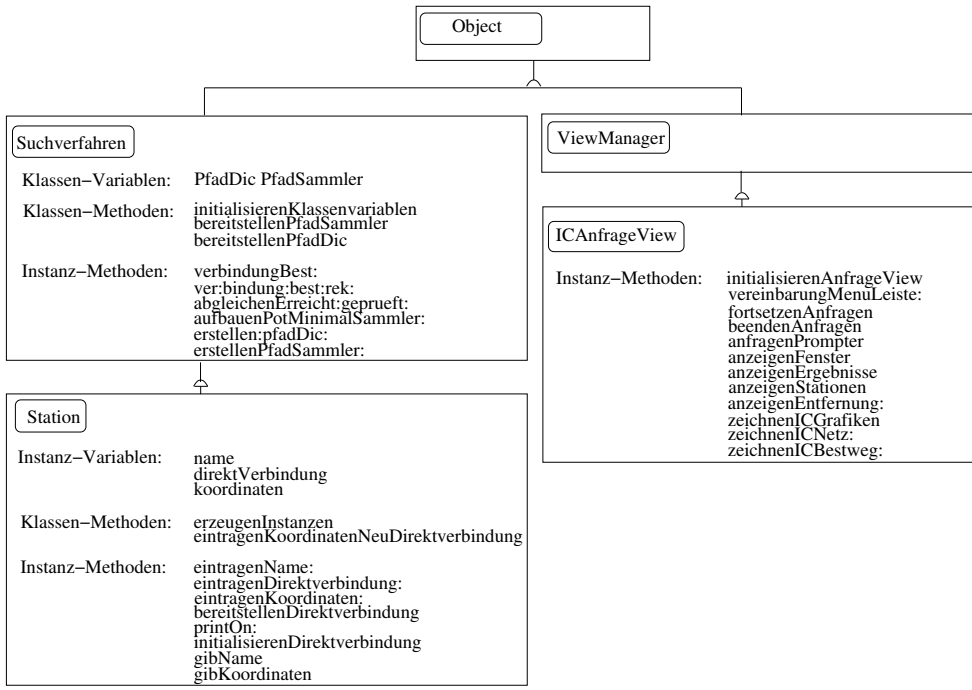


Abbildung 15.16: Klassen zur Lösung von PROB-13

Hinweis: In dieser Darstellung haben wir bei der Klasse “Suchverfahren” – aus Gründen der Übersichtlichkeit – lediglich diejenigen Methoden-Selektoren aufgeführt, die zur Bestimmung des Bestweges notwendig sind.

Zur Anzeige des Anfrage-Fensters und zur Initialisierung des Dialogs ist die folgenden Anforderung zu stellen:

```
ICAnfrageView new initialisierenAnfrageView; anfragenPrompter
```

Rufen wir z.B. nach der Eingabe des Abfahrtortes “Ful” und des Ankunftortes “Ber” sämtliche Informationen ab, so erhalten wir die folgende Anzeige:

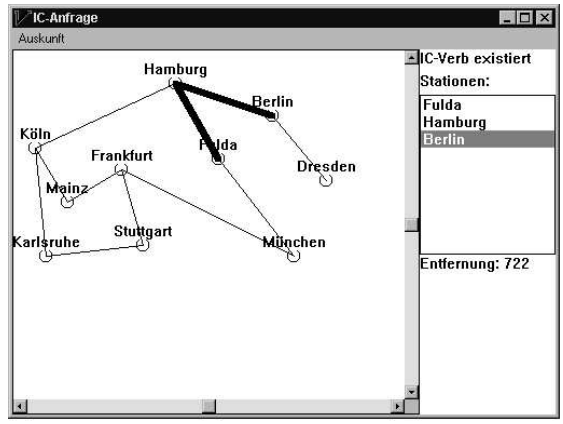


Abbildung 15.16: Anfrage-Fenster

Ausblick

Zum Abschluß ist anzumerken, daß wir in diesem Kapitel – aus didaktischen Gründen – für jede der Bahnstationen, wie z.B. die Station "Hamburg" eine globale Variable – die Variable "Ham" – eingesetzt haben. Der Einsatz dieser globalen Variablen läßt sich vermeiden, indem wir z.B. in der Klasse "Suchverfahren" eine zusätzliche Klassen-Variable vorsehen, um die Stationen des IC-Netzes festzuhalten. Setzen wir für diese Instanz-Variable ein Dictionary ein, so können wir z.B. als Key "#ham" und als Value die zugehörige Instanziierung der Klasse "Station" verwenden.

Unter der Voraussetzung, daß der Leser sich den Inhalt der vorausgehenden Kapitel erarbeitet hat, dürfte es nicht schwerfallen, eine entsprechende Modifikation der vorgestellten Methoden und verwendeten Instanz- und Klassen-Variablen durchzuführen. Mit dem bislang erworbenen Wissen wird der Leser ebenfalls in der Lage sein, die vorliegende Struktur des Lösungsplans nutzen können, um geeignete Ergänzungen und Verfeinerungen in der Darstellung zu erreichen.

Insgesamt sollte der erreichte Kenntnisstand den Leser befähigen, zu einem gestellten Problem einen objekt-orientierten Lösungsansatz zu entwickeln und diesen – durch den Einsatz des vorgestellten SMALLTALK-Systems – zur Ausführung zu bringen.

Anhang

A.1 Automatisch erstellte Methode zum Aufbau des Erfassungsfensters

Im Hinblick auf die Lösung unserer Problemstellung PROB-1-1 resultiert aus dem Einsatz des Werkzeugs "WindowBuilder Pro/V" ein Erfassungsfenster, dessen Aufbau durch eine *automatisch* generierte Methode namens "create Views" beschrieben wird. Diese Methode besitzt die folgende Form:

```
createViews

     "WARNING!!  This method was automatically generated by
      WindowBuilder.  Code you add here which does not conform
      to the WindowBuilder API will probably be lost the next time
      you save your layout definition."
| v |
self addView: (
    v := self topPaneClass new
        owner: self;
        labelWithoutPrefix:  '';
        noSmalltalkMenuBar;
        viewName: 'mainView';
        framingBlock: ( FramingParameters new iDUE: 686 @ 400;
                        xC; yC; cRDU: (9 @ 392 rightBottom: 677 @ 46));
        pStyle: #(sysmenu sizable titlebar minimize maximize);
        addSubpane: (
            StaticText new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 110 @ 32;
                                lDU: 91 r: #left; rDU: 201 r: #left;
                                tDU: 64 r: #top; bDU: 96 r: #top);
                startGroup;
                contents: 'Wert:';
                yourself
        );
        addSubpane: (
            EntryField new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 229 @ 48;
                                lDU: 302 r: #left; rDU: 530 r: #left;
                                tDU: 64 r: #top; bDU: 112 r: #top;
```

```
                                indent: 3 @ 4);
                paneName: 'eingabeFeld';
                startGroup;
                yourself
        );
        addSubpane: (
            Button new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 174 @ 64;
                                lDU: 101 r: #left; rDU: 274 r: #left;
                                tDU: 200 r: #top; bDU: 264 r: #top);
                startGroup;
                when: #clicked perform: #erfassenWert:;
                contents: 'erfasse';
                yourself
        );
        addSubpane: (
            Button new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 302 @ 64;
                                lDU: 302 r: #left; rDU: 603 r: #left;
                                tDU: 200 r: #top; bDU: 264 r: #top);
                startGroup;
                when: #clicked perform: #entfernenErfassungsfenster:;
                contents: 'Erfassungsende';
                yourself
        );
    yourself
).
```

Sofern – ohne den Einsatz des "WindowBuilders" – eine zum Aufbau des Erfassungsfensters benötigte Methode vereinbart werden soll, kann die Methode in dieser Form festgelegt werden (nähere Angaben zu Methoden-Vereinbarungen von Fenstern und Fenster-Bausteinen sind Bestandteil von Kapitel 10 und 12).

Der Methoden-Vereinbarung von "createViews" liegt die folgende Grobstruktur zugrunde, die die Grundlage für den individuellen Aufbau von Fenstern darstellt (siehe Abschnitt 10.6):

```
createViews
|v|
self addView: (
 v := self topPaneClass new owner: self;
   labelWithoutPrefix:  '';
   noSmalltalkMenuBar;
   framingBlock: (<Angaben zur Plazierung des Erfassungsfensters>);
  addSubpane: (StaticText new owner: self;
    framingBlock: (<Angaben zur Plazierung des Textes:``Wert:''>);
    contents: 'Wert:'
              );
  addSubpane: (EntryField new owner: self;
    framingBlock: (<Angaben zur Plazierung des Eingabefeldes>);
    paneName: 'eingabeFeld'
              );
```

```
addSubpane: (Button new owner: self;
  framingBlock: (<Angaben zur Plazierung des Schaltfeldes ‘‘erfasse’’>);
  when: #clicked perform: #erfassenWert:;
  contents: 'erfasse'
            );
addSubpane: (Button new owner: self;
  framingBlock: (<Angaben zur Plazierung des Schaltfeldes ‘‘Erfassungsende’’>);
  when: #clicked perform: #entfernenErfassungsfenster:;
  contents: 'Erfassungsende'
            )
        )
```

A.2 Das Chunk-Format

Um Klassen- und Methoden-Vereinbarungen in Dateien zu sichern bzw. von einem SMALLTALK-System in ein anderes SMALLTALK-System zu portieren, wird ein besonderes Ablageformat verwendet, das *Chunk-Format* genannt wird.
Sofern z.B. die Vereinbarung der Klasse "WerteErfassung" mit den Methoden "initialisierenErfassung", "erfassenWert:", "entfernenErfassungsfenster:", "festlegenUeberschrift:" und "durchfuehrenErfassung" exportiert wird (siehe die Angaben im Abschnitt 2.6.2), erhält die resultierende Klassen-Datei "WrtErfss.cls" die folgenden Einträge im Chunk-Format:

```
ViewManager subclass: #WerteErfassung
  instanceVariableNames:
    'werteBag'
  classVariableNames: ''
  poolDictionaries: '' !
!WerteErfassung class methods ! !
!WerteErfassung methods !
durchfuehrenErfassung
self openWindow.
(self paneNamed: 'eingabeFeld') setFocus!
entfernenErfassungsfenster: aPane
self close!
erfassenWert: aPane
werteBag add: (self paneNamed: 'eingabeFeld') contents.
(self paneNamed: 'eingabeFeld') contents: ''.
(self paneNamed: 'eingabeFeld') setFocus!
festlegenUeberschrift: aString
self labelWithoutPrefix: aString!
initialisierenErfassung
| v |
werteBag := Bag new.
self addView: (
    v := self topPaneClass new
        owner: self;
        labelWithoutPrefix:  '';
        noSmalltalkMenuBar;
        viewName: 'mainView';
        framingBlock: ( FramingParameters new iDUE: 686 @ 400;
                        xC; yC; cRDU: (9 @ 392 rightBottom: 677 @ 46));
        pStyle: #(sysmenu sizable titlebar minimize maximize);
        addSubpane: (
            StaticText new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 110 @ 32;
                                lDU: 91 r: #left; rDU: 201 r: #left;
                                tDU: 64 r: #top; bDU: 96 r: #top);
                startGroup;
                contents: 'Wert:';
                yourself
        );
        addSubpane: (
            EntryField new
```

```
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 229 @ 48;
                                lDU: 302 r: #left; rDU: 530 r: #left;
                                tDU: 64 r: #top; bDU: 112 r: #top;
                                indent: 3 @ 4);
                paneName: 'eingabeFeld';
                startGroup;
                yourself
        );
        addSubpane: (
            Button new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 174 @ 64;
                                lDU: 101 r: #left; rDU: 274 r: #left;
                                tDU: 200 r: #top; bDU: 264 r: #top);
                startGroup;
                when: #clicked perform: #erfassenWert:;
                contents: 'erfasse';
                yourself
        );
        addSubpane: (
            Button new
                owner: self;
                framingBlock: ( FramingParameters new iDUE: 302 @ 64;
                                lDU: 302 r: #left; rDU: 603 r: #left;
                                tDU: 200 r: #top; bDU: 264 r: #top);
                startGroup;
                when: #clicked perform: #entfernenErfassungsfenster:;
                contents: 'Erfassungsende';
                yourself
        );
    yourself
).! !
```

Die Eintragungen im Chunk-Format gliedern sich in drei Abschnitte, die jeweils durch ein Trennzeichen in Form des Ausrufungszeichens "!" beendet werden.

Hinweis: Mit Ausnahme des ersten Abschnitts werden die einzelnen Abschnitte durch das Zeichen "!" eingeleitet.
Sofern zwei Trennzeichen aufeinanderfolgen, werden sie durch mindestens ein Leerzeichen "␣" getrennt.
Beim Sichern einer Klassen-Vereinbarung werden die Methoden gemäß der alphabetischen Reihenfolge der Methoden-Selektoren abgelegt.

Im ersten Abschnitt ist der Name der Klasse und die Bezeichnung der Instanz-Variablen angegeben.
Der nächste Abschnitt enthält die Klassen-Methoden (siehe Abschnitt 8.1). Da in diesem Fall keine Klassen-Methoden vereinbart sind, gibt es keine Angaben außer der Überschrift "WerteErfassung class methods !".
Im dritten Abschnitt sind die Instanz-Methoden der Klasse eingetragen. Dabei wird jede Methode durch ihren Methoden-Selektor eingeleitet. Die innerhalb der Methode festgelegten Anforderungen, die anschließend aufgeführt sind, werden durch das Trennzeichen "!" abgeschlossen.

A.3 Fehlermeldungen und Unterbrechung der Ausführung

Syntaktische Fehler und Laufzeitfehler

Grundsätzlich gibt es zwei Arten von Fehlermeldungen:

- Fehlermeldungen als Folge syntaktischer Fehler
- Fehlermeldungen zur Laufzeit als Folge semantischer Fehler

Ein syntaktischer Fehler liegt z.B. dann vor, wenn der Anforderung

```
WerteErfassung11 initialisierenErfassung
```

eine Zahl oder eine Zeichenkette folgt, indem z.B. die Anforderung

```
WerteErfassung11 initialisierenErfassung 'Jahrgangsstufe 11'
```

gestellt wird.

Aus der Ausführung dieser Anforderung resultiert die Fehlermeldung "should be selector". Dies liegt daran, daß das SMALLTALK-System erkennt, daß es sich bei 'Jahrgangsstufe 11' nicht um einen Methoden-Selektor handelt.
Ein syntaktischer Fehler liegt z.B. auch dann vor, wenn wir die Keyword-Message "festlegenUeberschrift:" ohne die Angabe eines Arguments an das Empfänger-Objekt "WerteErfassung11" in der Form "WerteErfassung11 festlegenUeberschrift:" schicken. In diesem Fall wird die Fehlermeldung "argument missing" angezeigt.
Fehler zur Laufzeit treten insbesondere dann auf, wenn einem Empfänger-Objekt eine Message geschickt wird und die mit dieser Message korrespondierende Methode dem Empfänger-Objekt nicht bekannt ist.
Dies ist z.B. dann der Fall, wenn wir innerhalb der Methode "anzeigenDurchschnitt" (siehe Abschnitt 4.3) statt der Anforderung

```
self bereitstellenWerte
  do: [:einObjekt|summe := summe + einObjekt asInteger]
```

fälschlicherweise die folgende Anforderung eingetragen haben:

```
self bereitstellenWerte
  do: [:einObjekt|summe := summe + einObjekt]
```

Wird die Anforderung "summe := summe + einObjekt" ausgeführt, so wird die binäre Methode "+" innerhalb der Klasse des Empfänger-Objekts "summe" gesucht und in der Klasse "Integer" gefunden. Die Methode "+" ist in der Klasse "Integer" folgendermaßen vereinbart:

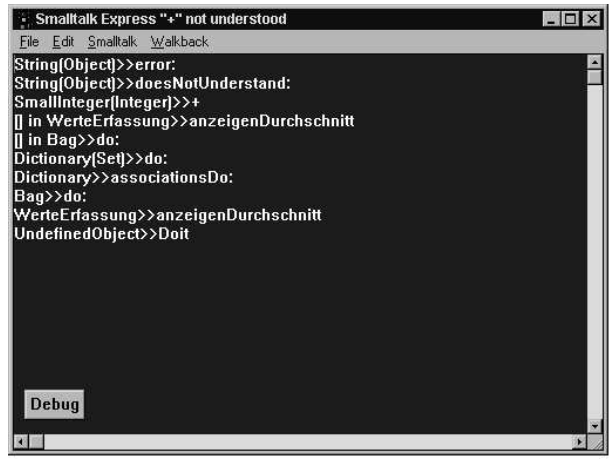
```
+ aNumber
<primitive: 21>
^  aNumber + self
```

Hinweis: Die mit "`<primitive: 21>`" gekennzeichnete Basis-Methode zählt zu den Primitiva des SMALLTALK-Systems.

In dieser Situation steht die Pseudovariable "self" für die temporäre Variable "summe" und das Argument "aNumber" für eine Instanz der Klasse "String". Somit wird die Methode "+" innerhalb der Klasse "String" gesucht. Da diese Methode den Instanzen der Klasse "String" nicht bekannt ist, kann sie nicht ausgeführt werden, so daß ein Laufzeitfehler festgestellt wird.

Das Walkback-Fenster

Im Fall eines Laufzeitfehlers erscheint ein Fenster mit roter Hintergrundfarbe, das als *Walkback-Fenster* bezeichnet wird und in der zuvor geschilderten Situation die folgende Form besitzt:



Hinweis: Ein Walkback-Fenster kann über das System-Schaltfeld "Schließen" vom Bildschirm entfernt werden.

In der Titel-Zeile dieses Fensters wird die Fehlermeldung

```
Smalltalk Express "+" not understood
```

angezeigt. Sie weist darauf hin, daß die Methode "+" dem Empfänger-Objekt nicht bekannt ist.

Die im Walkback-Fenster enthaltenen Angaben beschreiben – von unten nach oben

gelesen – die bei der Ausführung der Anforderung

```
WerteErfassung11 anzeigenDurchschnitt
```

vom SMALLTALK-System durchgeführten Aktionen.
So wird z.B. durch

```
SmallInteger(Integer)>>+
```

angegeben, daß einem Empfänger-Objekt der Basis-Klasse "SmallInteger" die Message "+" geschickt und in der Klasse "Integer" gefunden wird.

Gezielte Anforderung des Walkback-Fensters

Ein Walkback-Fenster wird nicht nur durch Laufzeitfehler aktiviert, sondern läßt sich auch wie folgt abrufen:

- Mittels der Message "halt" in Form einer Anforderung der Form "self halt".
- Während der Ausführung einer Anforderung durch die Tastenkombination "Ctrl + Break" ("Strg + Pause").

  Hinweis: In diesen Fällen läßt sich das Walkback-Fenster durch die Menü-Option "Resume" des Menüs "Walkback" schließen und die Ausführung fortsetzen.

Sofern wir zur Lösung von PROB-1 die Anforderungen

```
WerteErfassung11 := WerteErfassung new.
WerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'.
WerteErfassung11 anzeigenDurchschnitt
```

insgesamt zur Ausführung bringen, wird zusätzlich zum Erfassungsfenster ein Walkback-Fenster mit der Titel-Zeile

```
Smalltalk Express (User I/F) divisor is zero
```

angezeigt.
Dies liegt daran, daß durch die Anforderung

```
WerteErfassung11 anzeigenDurchschnitt
```

auf die innerhalb der Instanz-Variablen "werteBag" gesammelten Werte zugegriffen wird. Da jedoch der Erfassungsprozeß bislang nur gestartet und im zugehörigen Erfassungsfenster noch keine Eingabe erfolgt ist, liefert – innerhalb der Methode "anzeigenDurchschnitt" – die Auswertung von

```
self bereitstellenWerte size
```

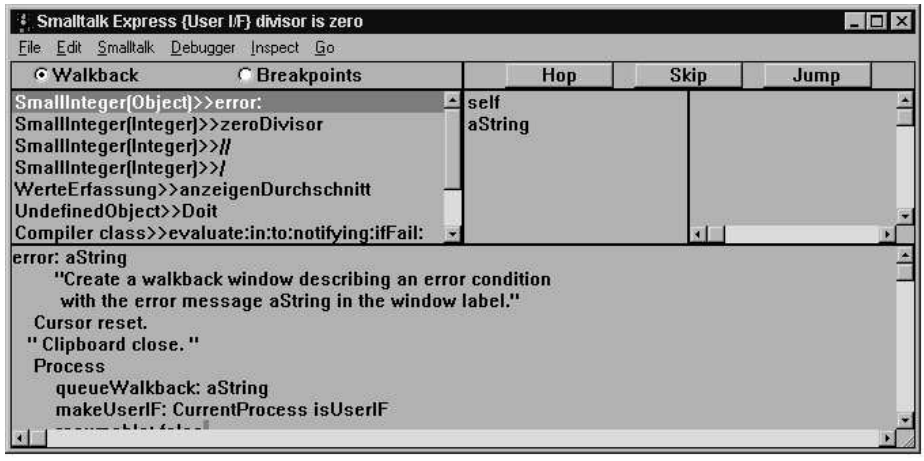
den Wert “0” als Ergebnis-Objekt, so daß die Division

```
summe / (self bereitstellenWerte size)
```

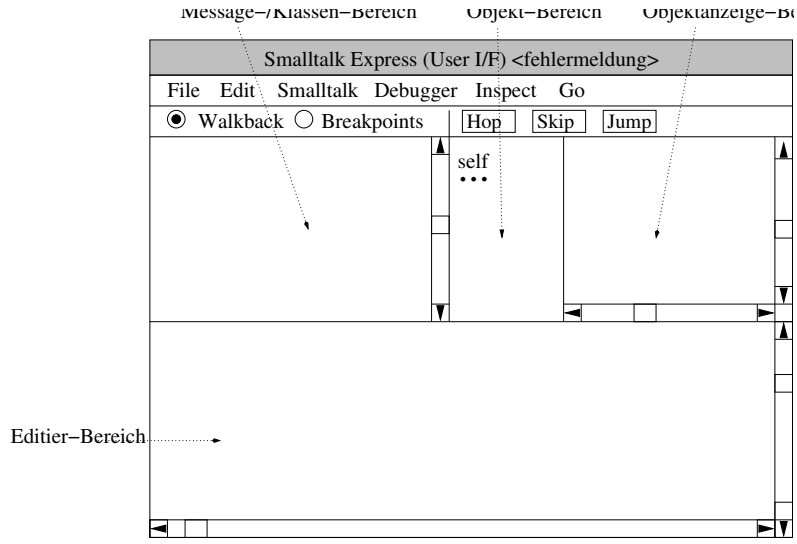
nicht durchgeführt werden kann.

Das Debug-Fenster

Aktivieren wir im oben angezeigten Walkback-Fenster das Schaltfeld mit der Aufschrift “Debug”, so wird das Walkback-Fenster durch ein anderes Fenster ersetzt, das *Debug-Fenster* genannt wird und sich in unserer Situation in der folgenden Form darstellt:



Dieses Fenster ist wie folgt strukturiert:



Es besteht aus den folgenden Bereichen:

- dem Message-/ Klassen-Bereich:
  In diesem Bereich werden sämtliche Klassennamen und Messages zeilenweise angezeigt, die bis zum Zeitpunkt der Eröffnung des Debug-Fensters bearbeitet wurden.
  Jede Zeile ist in der Form

  ```
  <Klassenname> >> <message>
  ```

  strukturiert, d.h. sie besteht aus der Angabe einer Message und dem Namen der Klasse, zu der das Empfänger-Objekt der jeweiligen Message gehört.
  Da die zuletzt ausgeführten Messages und die jeweils zugehörige Klasse zuoberst angezeigt werden, enthält die erste Zeile den Namen der zuletzt ausgeführten Methode und jede nachfolgende Zeile eine Information über diejenige Message, von der aus die in der darüberstehenden Zeile stehende Message durch das Schicken einer entsprechenden Nachricht aktiviert worden ist.

  Hinweis: Wir setzen voraus, daß im angezeigten Debug-Fenster das Optionsfeld "Walkback" aktiviert ist.
  Wird eine Zeile in der allgemeinen Form
  `<Klassenname1>(<Klassenname2>) >> <message>`
  angezeigt, so ist dies ein Hinweis dafür, daß die aufgeführte Message einem Empfänger-Objekt aus der Klasse namens "<Klassenname1>" geschickt und die korrespondierende Methode in der Klasse "<Klassenname2>" gefunden wurde.

- dem Editier-Bereich:
  In diesem Bereich wird die Vereinbarung der Methode angezeigt, die mit der im Message-/Klassen-Bereich markierten Message korrespondiert.

- dem Objekt-Bereich und dem Objektanzeige-Bereich:
  In diesem Bereich werden nähere Angaben über Empfänger-Objekte angezeigt.
  Sofern im Message-/Klassen-Bereich eine Zeile markiert wird, erscheint im Objekt-Bereich die Anzeige "self". Wird diese Anzeige markiert, so erscheint im Objektanzeige-Bereich ein Hinweis auf das Empfänger-Objekt, das zu der im Message-/Klassen-Bereich markierten Message gehört.

  Hinweis: Handelt es bei der im Message-/Klassen-Bereich ausgewählten Message um eine Keyword-Message, so wird im Objekt-Bereich zusätzlich der Name des Arguments (der Argumente) angezeigt, der bei der Vereinbarung dieser Methode festgelegt wurde. Sind innerhalb der zu einer Message gehörigen Methode temporäre Variablen vereinbart, so werden die Namen dieser Variablen ebenfalls im Objekt-Bereich angezeigt.
  Auf die aktuellen Werte der angezeigten Argumente bzw. Variablen können wir jeweils einzeln zugreifen, indem wir im Objekt-Bereich den jeweiligen Namen aktivieren. Dabei werden die zugehörigen Werte im Objektanzeige-Bereich angezeigt.
  Wird z.B. in der oben angegebenen Situation im Message-/ Klassen-Bereich die Zeile mit dem Eintrag "WerteErfassung>>anzeigenDurchschnitt" markiert, so werden im Objekt-Bereich die Angaben "self", "summe", "durchschnittswert" und "einObjekt" angezeigt.

Gezielte Unterbrechung der Ausführung

Um die Programmierung neuer Methoden zu prüfen, ist es möglich, die Ausführung einer Methode zu unterbrechen und den aktuellen Zustand von Objekten zu untersuchen.
Sofern der gezielte Halt als Bestandteil einer Methode festgelegt werden soll, können wir die Basis-Methode "halt" einsetzen.

- **"halt"**:
  Wird die Message "halt" in der Form "self halt" eingesetzt, so bewirkt diese Anforderung, daß die Ausführung der aktuell bearbeiteten Methode unterbrochen und der aktuelle Status in dem daraufhin angezeigten Walkback-Fenster ausgewiesen wird.

Um z.B. die Ausführung der Methode "durchschnitt" (siehe Abschnitt 5.3) zu verfolgen, tragen wir daher die Anforderung "self halt" als erste auszuführende Anforderung in die Methode "durchschnitt" der Klasse "InWerteErfassung" ein, so daß die Methoden-Vereinbarung von "durchschnitt" die folgende Form besitzt:

```
durchschnitt
|summe|
self halt.
summe := 0.
self bereitstellenWerte
     do: [:einObjekt|summe:= summe + einObjekt asInteger].
durchschnittswert:= summe / (self bereitstellenWerte size)
```

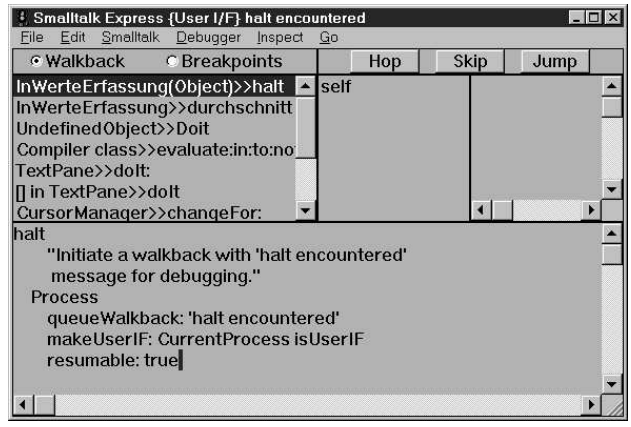
Stellen wir anschließend die Anforderungen

```
InWerteErfassung11 := InWerteErfassung new.
InWerteErfassung11 sammelnWerte: 'Jahrgangsstufe 11'
```

und nach dem Ende der Erfassung die Anforderung

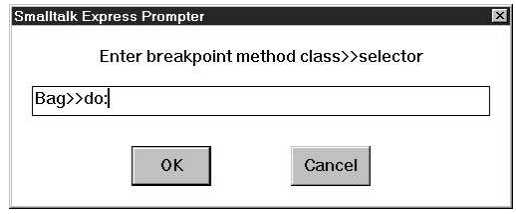
```
InWerteErfassung11 durchschnitt
```

so wird – durch die Ausführung von "self halt" – die Ausführung der Methode "durchschnitt" unterbrochen und ein Walkback-Fenster angezeigt. Aktivieren wir in diesem Fenster das Schaltfeld mit der Aufschrift "Debug", so erhalten wir das folgende Debug-Fenster:



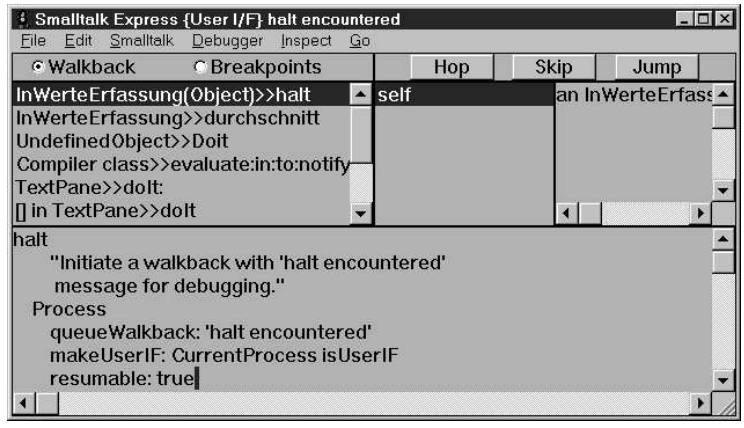
Soll die durch den Einsatz der Anforderung "self halt" unterbrochene Ausführung fortgesetzt werden, so ist das Schaltfeld "Jump" zu betätigen.
Sollen dagegen gezielt eine oder mehrere weitere Unterbrechungen vorgesehen werden, so müssen *Breakpoints* vereinbart werden.

Um z.B. die Ausführung der Methode "durchschnitt" an der Stelle zu unterbrechen, an der die Methode "do:" der Basis-Klasse "Bag" aufgerufen wird, ist ein Breakpoint für die Methode "do:" festzulegen. Dazu ist im Debug-Fenster die Menü-Option "Add Breakpoint" des Menüs "Debugger" auszuwählen und im nachfolgend angezeigten Dialogfeldfenster der Text "Bag>>do:" einzutragen, so daß sich der folgende Bildschirminhalt ergibt:



Nachdem der Breakpoint bestätigt und anschließend im Debug-Fenster das Schaltfeld "Jump" betätigt wurde, wird die durch den Einsatz der Anforderung "self halt" unterbrochene Ausführung fortgesetzt und bis zu derjenigen Stelle weitergeführt, an der zum ersten Mal die Methode "do:" der Klasse "Bag" ausgeführt werden soll.

Wird die Ausführung an dem festgelegten Breakpoint unterbrochen und im Objekt-Bereich des daraufhin angezeigten Debug-Fensters die Zeile mit dem Eintrag "self" markiert, so erhalten wir die folgenden Anzeige:



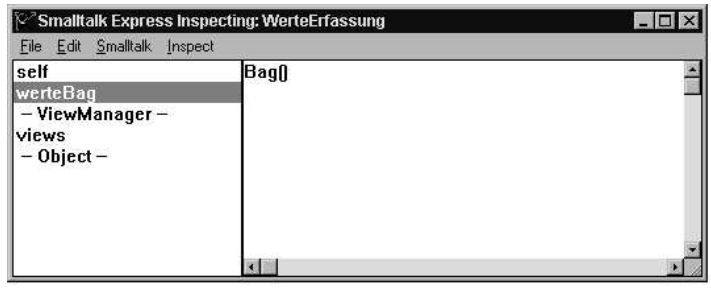
Hinweis: Um die Ausführung einer Anforderung sukzessiv verfolgen zu können, stehen im Debug-Fenster die Schaltfelder "Hop" und "Skip" zur Verfügung.
Beim Einsatz von "Hop" wird die im Editier-Bereich markierte Message aktiviert und die zugehörige Methode zur Ausführung gebracht, so daß die Anforderungen der durch die Message aktivierten Methode vollständig bearbeitet werden.
Dagegen wird beim Einsatz von "Skip" die im Editier-Bereich markierte Message lediglich an das Empfänger-Objekt geschickt, so daß im Message-/ Klassen-Bereich der Sprung von der aktuellen Methode zu der Methode sichtbar wird, die durch die markierte Message zur Ausführung gebracht wird.

A.4 Das Inspect-Fenster

Um die Eigenschaften von Objekten anzeigen zu lassen, kann das *Inspect-Fenster* angefordert werden. Dazu ist der Name des Objekts im Workspace-Fenster zu markieren und die Menü-Option "Inspect" des Menüs "Smalltalk" zu bestätigen.

Hinweis: Das gleiche können wir erreichen, wenn wir dem jeweiligen Objekt die Message "inspect" schicken.

Wollen wir z.B. untersuchen, welche Werte im Sammel-Behälter "werteBag" der Instanz "WerteErfassung11" enthalten sind, so müssen wir den Namen "WerteErfassung11" in das Workspace-Fenster eintragen. Wird nach dessen Markierung die Menü-Option "Inspect" des Menüs "Smalltalk" bestätigt, so wird das Inspect-Fenster angezeigt. Markieren wir in diesem Fenster den Eintrag "werteBag", so erhalten wir z.B. die folgende Anzeige:



Hinweis: Dabei wird durch "Bag()" mitgeteilt, daß in dem Sammler, auf den die Instanz-Variable "werteBag" der Instanz "WerteErfassung11" weist, noch kein Wert enthalten ist. Das Inspect-Fenster läßt sich über das System-Schaltfeld "Schließen" vom Bildschirm entfernen.
Die Message "inspect" können wir auch innerhalb einer Methode einsetzen. Zum Beispiel läßt sich – zu Testzwecken – "sammelnWerte:" in der Form

```
sammelnWerte: aString
self initialisierenErfassung;
     festlegenUeberschrift: aString;
     inspect;
     durchfuehrenErfassung
```

vereinbaren. Hierdurch können wir – in der Testphase – die erfaßten und in "werteBag" gesammelten Werte sukzessiv in dem neben dem Erfassungsfenster zusätzlich angezeigten Inspect-Fenster zur Kontrolle abrufen.

Literaturverzeichnis

BYTE Publication Inc., Volume 6, Number 8, August 1981.

Digitalk: Smalltalk/V for Windows, Encyclopedia of classes, Digitalk Inc., Los Angeles, 1992.

Dugerdil Ph.: Smalltalk-80, Programmation par objets, Presses polytechniques et universitaires romandes, Lausanne, 1990.

Goldberg A., Robson D.: Smalltalk-80, The language, Addison-Wesley, 1989.

Informatik-Spektrum: Band 20, Heft 6, Springer 1997.

LaLonde W.R., Pugh J.R.: Inside Smalltalk, Vol. I, Prentice-Hall International, 1990.

LaLonde W.R.: Smalltalk/V: practice and experience, Prentice-Hall Inc., 1994.

Mayr H.C., Wagner R. (Hrsg.): Objektorientierte Methoden für Informationssysteme, Springer, 1993.

Schader M., Rundshagen M.: Objektorientierte Systemanalyse, Eine Einführung, Springer, 1996.

Skublics S., Klimas E.J., Thomas D.A.: Smalltalk with Style, Prentice-Hall Inc., 1996.

Tietjen S., Voss E.: Objektorientierte Programmierung mit Smalltalk/V, Vieweg, 1994.

Vinek, G.: Objektorientierte Softwareentwicklung mit Smalltalk, Springer, 1997.

Index

Index