2 Konstruktion von Algorithmen

In diesem Kapitel nähern wir uns der Beantwortung der Frage an, wie man für ein gegebenes Problem einen Algorithmus findet. Leider gibt es auf diese Frage keine einfache Antwort, schon gar nicht können irgendwelche „Kochrezepte“ angegeben werden, die es ermöglichten, zielsicher vom Problem zum Programm zu gelangen. Es gibt in der Software-Technik bis heute auch keine wohl definierte Konstruktionslehre, wie das in reiferen Ingenieurwissenschaften eine Selbstverständlichkeit ist. Vielmehr gibt es verschiedene Formen von Handlungsanleitungen, die den Programmierer bei seiner Tätigkeit unterstützen und die Wahrscheinlichkeit erhöhen sollen, dass die entstehenden Programme bestimmte Qualitätskriterien, wie z. B. *Korrektheit* und *Lesbarkeit*, erfüllen. Nähere Hinweise dazu werden im Abschnitt 2.4 gegeben.

Die Konstruktion von Algorithmen kann man in zwei Teilaufgaben zerlegen: Der erste Schritt besteht darin, zu einer Problemstellung die Methode zu finden, mit der es gelöst werden kann. Diese Lösungsmethode – der Algorithmus im eigentlichen Sinne – kann zunächst unabhängig davon formuliert werden, welche Sprache der Automat versteht, der zur automatischen Lösung des Problems letztlich herangezogen werden soll. Im zweiten Schritt muss dann aber die maschinengerechte Formulierung des Algorithmus in Form eines Programms, aufgeschrieben in einer für den gewählten Automaten verständlichen Programmiersprache, erfolgen. Zunächst werden zwei einfache Beispiele betrachtet, bei denen der erste Schritt kein Problem darstellt. Im ersten Beispiel soll das Problem gelöst werden, einen bestimmten Betrag von einer Währung in eine andere umzurechnen. Hier ist die Lösungsmethode offensichtlich. Das zweite Beispiel betrachtet das Problem der Lösung quadratischer Gleichungen, dessen Lösungsmethode uns die Mathematik schon liefert. In beiden Fällen können wir uns daher auf den zweiten Schritt konzentrieren, nämlich die Lösungsmethode „nur“ noch maschinengerecht zu formulieren.

Lösungsmethode

maschinengerechte Formulierung

2.1 Fallbeispiel Währungsumrechnung

In Zeiten vor Einführung des Euro war man als Tourist in Europa gerne mit kleinen Pappkärtchen unterwegs, auf denen eine Umrechnungstabelle von einer Währung in eine andere für ausgewählte Beträge abgedruckt war. So konnte man z. B. – ein wenig umständlich zwar – DM-Beträge in Österreichische Schillinge umrechnen, wobei der Wechselkurs natürlich nicht tagesaktuell war. In Zeiten, wo Wechselkurse ubiquitär online verfügbar sind, wird man hierfür wohl eher das Mobiltelefon benutzen.

An dieser Stelle soll nun das konkrete Problem gelöst werden, einen vorzugebenden Betrag in Schwedischen Kronen in Euro umzurechnen. Der Wechselkurs sei fest: 1 Schwedische Krone entspricht 0,108 Euro[1]. Zur Berechnung des Euro-Betrags kann

[1] Wechselkurs gültig am 12. Mai 2008

J. Brauer, *Grundkurs Smalltalk – Objektorientierung von Anfang an,*
DOI 10.1007/978-3-658-00631-0_2, © Springer Fachmedien Wiesbaden 2014

also die folgende simple Formel herangezogen werden:

$$F : euBetrag = skBetrag \cdot 0,108$$

An dieser Stelle muss nun klargestellt werden, über welche Fähigkeiten unser Automat eigentlich verfügt bzw. durch welche programmiersprachlichen Formulierungen diese Fähigkeiten nutzbar gemacht werden können. Die einzige Rechenoperation, die hier notwendig ist, die Multiplikation, gehört selbstverständlich zum Repertoire der Maschine, die wir benutzen werden.

Zu den Eigenheiten unseres Automaten zählt jedoch, dass er nur Programme verarbeiten kann, die in Form eines Textes aufgeschrieben sind, der ausschließlich Zeichen enthält, die auf einer „normalen“ Schreibmaschinentastatur (oder PC-Tastatur) zu finden sind. Zu diesem Zeichenvorrat gehört nicht der Malpunkt. Stattdessen wird das Zeichen „*“ verwendet. Außerdem muss noch das Dezimalkomma durch den Dezimalpunkt ersetzt werden. Unsere Formel sieht jetzt also so aus:

$$F : euBetrag = skBetrag * 0.108$$

Die „Konstruktion“ des Algorithmus für die Währungsumrechnung ist damit bereits abgeschlossen. Er ist insofern universell verwendbar, als er die Umrechnung jedes beliebigen Betrags von Schwedischen Kronen in Euro erlaubt. Für eine konkrete Berechnung muss für die Variable *skBetrag* ein konkreter Zahlenwert eingesetzt werden.

Im folgenden Abschnitt wird nun gezeigt werden, wie dieser Algorithmus auf einem konkreten Automaten zur Ausführung gebracht werden kann.

2.2 Das erste Smalltalk-Programm

SmaViM

Der Automat, den wir für die Ausführung von Algorithmen benutzen, soll vorerst den Namen *SmaViM* erhalten, was als Akronym für *Smalltalk Virtual Machine* steht. Was hat es nun mit diesem Namen auf sich?

Maschinensprache

In Kapitel 1 wurde darauf hingewiesen, dass Computer letztlich nur mit binären Symbolfolgen umgehen können. Das bedeutet, dass auch Programme selbst binär codiert werden müssen. Der Binärcode, den ein Computer unmittelbar als Handlungsanweisung interpretieren kann, wird auch als Maschinensprache bezeichnet. Das direkte Programmieren in Maschinensprache ist, weil Binärcodes für den Menschen schwer lesbar sind, nicht nur außerordentlich mühsam und fehleranfällig, sondern weist auch das Problem auf, dass jeder Rechnertyp seine eigene Maschinensprache besitzt. Wollte man also ein Programm, das für einen Maschinentyp entwickelt wurde, auf einem anderen bereitstellen, müsste es erst mühsam umcodiert werden. Daher hat schon in den fünfziger Jahren des 20. Jahrhunderts eine Entwicklung hin zu so genannten *höheren* oder *problemorientierten* Programmiersprachen eingesetzt. Bekannte Beispiele solcher Sprachen sind das schon erwähnte *ALGOL* sowie *FORTRAN, COBOL, PASCAL, C*, aber natürlich auch *Java* und *Smalltalk*. Problemorientiert nannte man diese Sprachen deshalb, weil sie den Programmierer nicht mehr zwangen, sich bei der Programmierung des Befehlssatzes der konkreten Maschine zu bedienen, sondern Problemlösungen in einer gewohnten, z. B. mathematischen, Notation aufschreiben zu können. Das Beispiel der Formel F macht dies deutlich.

problemorientierte Programmiersprache

Übersetzung von Programmen

Damit Programme, die in einer höheren Programmiersprache geschrieben sind, auf einem konkreten Rechner ablaufen können, müssen sie aber in die jeweilige Maschinensprache übersetzt werden. Dieser Übersetzungsvorgang kann nun seinerseits automatisiert werden, d. h. für diesen Vorgang gibt es Programme, so genannte *Compiler*. Der deutsche Begriff *Übersetzer* ist nicht sehr gebräuchlich. Diese müssen ihrerseits natürlich in Maschinensprache vorliegen, damit sie von dem Rechner ausgeführt werden können. Für jede Programmiersprache und jeden Maschinentyp braucht man einen eigenen Compiler.

Compiler

Eine etwas andere Vorgehensweise wird für die Ausführung von Smalltalk- und Java-Programmen verwendet. Um Programme, die von vornherein auf verschiedenen Maschinentypen bereitgestellt werden sollen, nicht durch die jeweils speziellen Compiler übersetzen lassen zu müssen, definiert man zunächst einen universellen, maschinenunabhängigen Befehlssatz[2]. Programme werden dann nur noch in diese Quasi-Maschinensprache übersetzt, unabhängig davon, auf welcher konkreten Maschine sie letztlich ablaufen sollen. Für jeden konkreten Maschinentyp braucht man dann aber ein Programm, das Befehle der Quasi-Maschinensprache interpretiert und für jeden dieser Befehle jeweils eine binäre Befehlsfolge der konkreten Maschine erzeugt. Dieses Programm lässt damit den Rechner *scheinbar* den universellen Befehlssatz verstehen, deswegen werden derartige Programme als *virtuelle Maschinen* bezeichnet. Wegen der eben beschriebenen Arbeitsweise werden sie aber auch *Interpreter* genannt. *SmaViM* ist also die virtuelle Maschine, die Smalltalk-Programme (übersetzt in den Byte-Code) ausführen kann.

maschinenunabhängiger Befehlssatz

virtuelle Maschine

Interpreter

Ein Smalltalk-System besteht seit jeher nicht nur aus einer Programmiersprache und einer virtuellen Maschine, sondern auch aus einer integrierten Entwicklungsumgebung mit einer modernen graphischen Bedienoberfläche, die es dem Programmierer erlaubt, Smalltalk-Programme zu schreiben, übersetzen und ausführen zu lassen sowie darüber hinaus auch Fehlersuche zu betreiben. Solche Smalltalk-Entwicklungsumgebungen gibt es von verschiedenen Herstellern, die sich in Art und Umfang der mitgelieferten Komponenten sowie deren Bedienung unterscheiden. Für unsere Zwecke genügt es, auf elementare Funktionen zurückzugreifen, die sich zumindest in ähnlicher Form in allen Entwicklungsumgebungen wieder finden lassen. Die Bildschirmfotos, die im Folgenden zur Illustration der Arbeitsweise bei der Programmierung in Smalltalk gezeigt werden, wurden unter Verwendung des Produkts *VisualWorks*[3] der Firma *Cincom* erstellt, das in einer Ausbildungsversion frei erhältlich ist.

Entwicklungsumgebung

Die Benutzung von *VisualWorks* wird in Kapitel 5. Im Abschnitt 5.3.2 werden die erforderlichen Grundeinstellungen erläutert, damit das Erscheinungsbild von *VisualWorks* den in diesem Band gezeigten Bildschirmfotos entspricht.

2.2.1 Eingabe von Programmtexten

Jede Smalltalk-Entwicklungsumgebung stellt einen so genannten *Workspace* zur Verfügung. Dabei handelt es sich um ein Fenster zur Eingabe und Modifikation von Smalltalk-Programmtexten, das wie ein entsprechendes Fenster in einem Textverarbeitungs-

Workspace

[2] häufig als Byte-Code bezeichnet
[3] in der Version 7.10

programm benutzt werden kann. Abbildung 2.1 zeigt einen solchen Workspace. Das Erscheinungsbild des dargestellten Fensters entspricht demjenigen, das man erhält, wenn man *VisualWorks* unter dem Betriebssystem *Mac OS X* auf einem Macintosh-Rechner der Firma Apple betreibt.

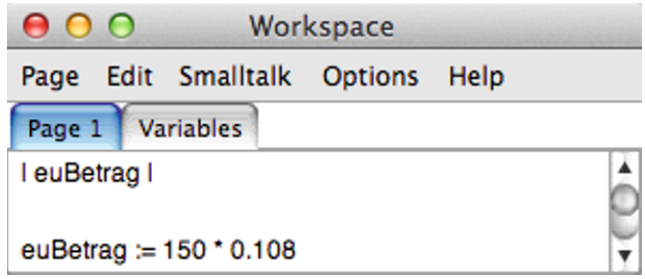

Abbildung 2.1: Workspace (Mac OS-Version)

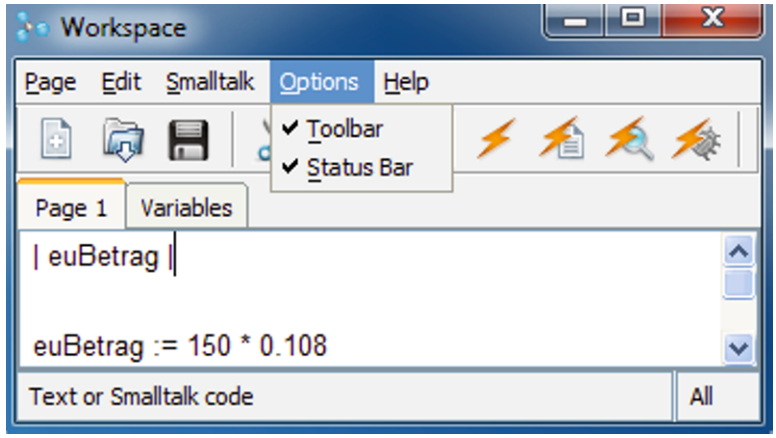

Abbildung 2.2: Workspace (Windows-Version)

Zum Vergleich zeigt Abbildung 2.2 das gleiche Workspace-Fenster unter dem Betriebssystem *Microsoft Windows 7*. Hier sind zusätzlich die Tools- und die Status-Zeile mit dargestellt, die über das **Options**-Menü ausgewählt werden können. Im weiteren Verlauf wird nur jeweils die Mac OS-Variante gezeigt.

Der Text in dem Fenster zeigt das Smalltalk-Programm für die Währungsumrechnung gemäß dem im vorigen Abschnitt besprochenen Algorithmus. Gegenüber dem dort gezeigten Text ist allerdings die Variable *skBetrag* durch den Zahlenwert 150 ersetzt worden. Das Programm soll hier also ausrechnen, wieviel Euro 150 Schwedische Kronen sind. Außerdem sind noch weitere kleine Textänderungen vorgenommen worden, die der Syntax von Smalltalk geschuldet sind:

Variablendeklaration

1. In der ersten Zeile steht der Name der im Programm benutzten Variablen (`euBetrag`) eingeschlossen zwischen zwei senkrechten Strichen. Hierbei handelt es sich um die so genannte *Deklaration* der Variablen. Generell gilt, dass Varia-

blen deklariert werden müssen, bevor sie verwendet werden können. Zwischen den senkrechten Strichen können beliebig viele Variablen deklariert werden, die durch Leerzeichen, Tabulatoren oder Zeilenwechsel voneinander getrennt werden. Genauer gesagt, handelt es sich hier um so genannte *temporäre* oder *lokale* Variablen. Andere Arten von Variablen werden wir später kennen lernen.

temporäre, lokale Variable

> Anmerkung für Kenner konventioneller höherer Programmiersprachen, wie z. B. PASCAL: In Smalltalk haben Variablen keinen Typ. Einer Variablen kann somit ein beliebiger Wert zugewiesen werden.

2. Anstelle von „ `euBetrag = ...`“ schreiben wir hier „ `euBetrag := ...`“ . Durch die Zeichenfolge „:=“ wird die so genannte *Zuweisung* ausgedrückt. Der links stehenden Variablen wird der Wert des rechts stehenden Ausdrucks zugeordnet (zugewiesen). Das Gleichheitszeichen (ohne Doppelpunkt davor) dient in Smalltalk dem Vergleich von Ausdrücken (vgl. Abschnitt 2.3.3).

Zuweisung

2.2.2 Ausführung von Programmen

Ein im Workspace eingegebener Programmtext kann sofort *SmaViM* zur Ausführung übergeben werden. Dazu muss dieser zunächst mit der linken Maustaste selektiert werden. Für die Ausführung des ausgewählten Textes gibt es in *VisualWorks* zwei Möglichkeiten. Die eine besteht darin, im Workspace-Menü **Smalltalk** den Menüpunkt **Do it** zu wählen (s. Abbildung 2.3). Den gleichen Effekt kann man durch

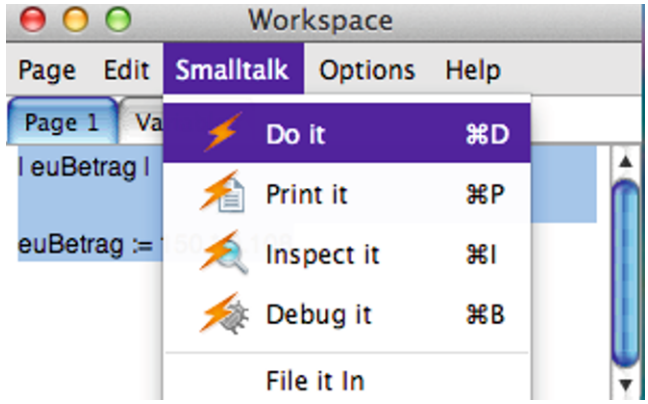

Abbildung 2.3: Das Menü **Smalltalk** im *VisualWorks*-Workspace

Aktivieren des Kontext-Menüs[4] erzielen. Dies enthält ebenfalls einen gleichnamigen Menüpunkt (s. Abbildung 2.4).

> Anmerkung: Klassische Smalltalk-Entwicklungsumgebungen, wie sie z. B. schon in der „Smalltalk-80-Bibel“ Goldberg und Robson (1989) beschrieben werden,

[4] unter Windows durch Betätigen der rechten Maustaste; falls unter Mac OS nur eine Eintasten-Maus zur Verfügung steht, kann das Kontext-Menü durch Ctrl-Klick aktiviert werden

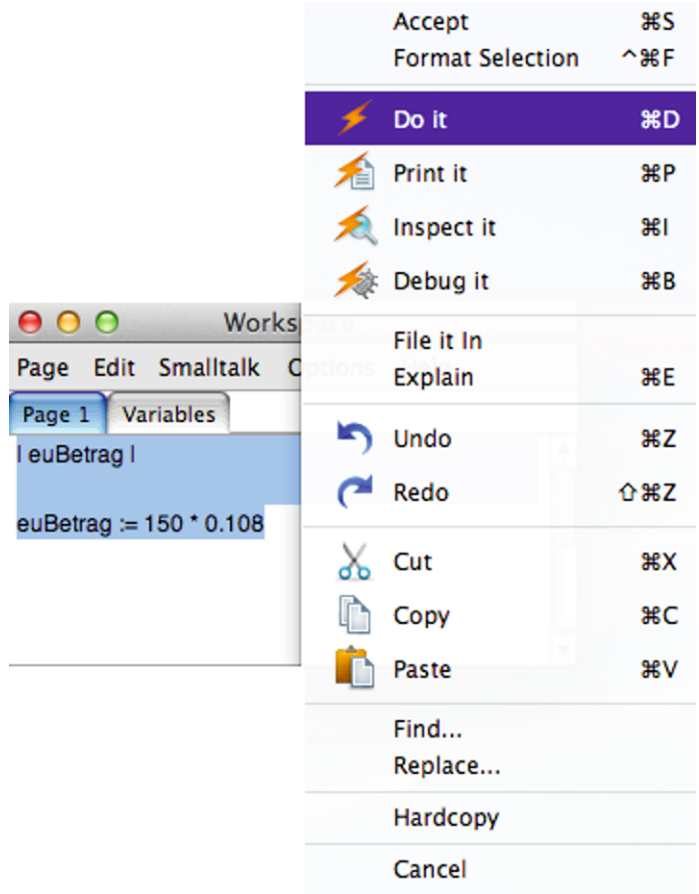

Abbildung 2.4: Ausführen eines Programms über das Kontext-Menü

setzen i. d. R. eine Drei-Tastenmaus voraus. Die linke Maustaste (*select button*) hat die „übliche“ Funktion (Auswahl von Menüpunkten, Selektion von Text usw.). Das Betätigen der rechten Maustaste (*operate button*) lässt ein kontextabhängiges Menü, das so genannte *Operate-Menü*[5], mit smalltalkspezifischen Einträgen aufklappen (vgl. Abbildung 2.4). Die mittlere Maustaste (*window button*) ruft ein Aufklapp-Menü mit fensterspezifischen Aktionen (Vergrößern, Verkleinern, Umbenennen usw.) hervor.

Welche der beiden Methoden man auch wählt, in jedem Fall erhält man den in Abbildung 2.5 gezeigten Warnhinweis, der den Entwickler des Programms darauf hinweisen soll, dass hier ein Wert berechnet und der Variablen `euBetrag` zugewiesen wird, der Wert dieser Variablen aber im weiteren Verlauf des Programms nicht benutzt wird. Das muss uns hier an dieser Stelle nicht stören. Das Betätigen der **proceed**-Schaltfläche hat dann aber keinen weiteren sichtbaren Effekt. Das Programm wird

[5] Im Folgenden werden wir in der Regel vom *Kontextmenü* sprechen.

Abbildung 2.5: Warnhinweis

zwar ausgeführt und die Lösung wird der Variablen `euBetrag` zugewiesen, um deren Wert aber sichtbar zu machen, bedarf es aber weiterer Vorkehrungen. Es gibt vielfältige Möglichkeiten, die Werte von Variablen oder Ausdrücken von *SmaViM* ausgeben zu lassen. Die erste besteht darin, den Menüpunkt **Print it** anstelle von **Do it** zu wählen. Das Ergebnis zeigt Abbildung 2.6.

Anzeigen von Variablenwerten

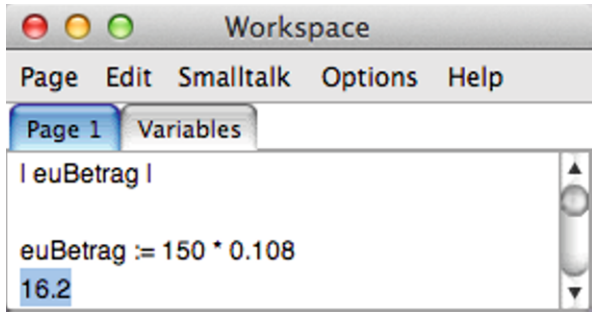

Abbildung 2.6: Ausführung des Programms mit dem Menüpunkt **Print it**

Bei der Ausführung mit **Print it** wird immer der Wert des zuletzt berechneten Ausdrucks in den Workspace ausgegeben. Abbildung 2.6 zeigt daher den Wert der Variablen `euBetrag` als selektierten Text an.

Eine weitere Möglichkeit von *SmaViM* besteht darin, die Ergebnisanzeige in einem kleinen Dialogfenster vorzunehmen. Dies muss aber programmiert werden, d. h. unser kleines Programm muss um eine Anweisung an *SmaViM* ergänzt werden, die ein solches Dialogfenster öffnet und darin die Ergebnisse darstellt. Die Anweisung hierfür könnte z. B. so aussehen:

Dialogfenster

```
Dialog warn: 'Betrag in Euro:', euBetrag printString
```

Form und Inhalt dieser Anweisung werden wir erst zu einem späteren Zeitpunkt genauer analysieren können. Zum Verständnis sei hier vorweg nur Folgendes gesagt: Die Anweisung `Dialog warn:` öffnet ein Dialogfenster und zeigt darin den Text an, der hinter dem Doppelpunkt angegeben wird. Texte, die auf diese Art unmittelbar angezeigt werden können, sind beliebige in einfache Hochkommata eingeschlossene Zeichenfolgen (z. B. `'Betrag in Euro:'`). Zahlenwerte, also z. B. der Wert der Variablen `euBetrag`, müssen durch Anhängen der Anweisung `printString` erst in einen Text umgewandelt werden. Das Komma dient dazu, die beiden Teiltexte zu einem Text zusammenzufügen. Dies ist notwendig, weil die Anweisung `Dialog warn:` eben genau

einen Text als Parameter erwartet. Abbildung 2.7 zeigt unser entsprechend erweitertes Programm. Um die Ausgabeanweisung der ersten hinzufügen zu können, muss diese

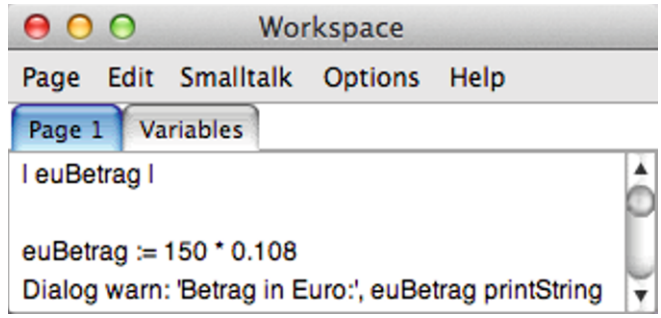

Abbildung 2.7: Programmergänzung für die Ergebnisausgabe im Dialogfenster

mit einem Punkt abgeschlossen werden. Vereinfacht gesprochen: Zwei aufeinander folgende Anweisungen werden durch einen Punkt voneinander getrennt. Selektiert man den Programmtext wieder und wählt erneut den Menüpunkt **Do it**, erscheint das in Abbildung 2.8 gezeigte Dialogfenster auf dem Bildschirm.

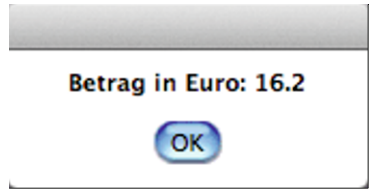

Abbildung 2.8: Ergebnisausgabe im Dialogfenster

Somit haben wir es nun geschafft, das Problem der Währungsumrechnung für einen konkreten festen Wechselkurs und einen bestimmten Betrag in Schwedischen Kronen zu lösen. Diese Lösung ist insofern unbefriedigend, als eine Änderung des Wechselkurses oder des umzurechnenden Betrags eine Programmänderung erfordert.

2.2.3 Flexibilisierung der Währungsumrechnung

Es ist offenkundig, dass ein Programm, das einen bestimmten Betrag zu einem fest stehenden Wechselkurs umrechnen kann, nicht besonders nützlich ist. Um das Programm flexibler einsetzen zu können, betrachten wir zunächst eine verallgemeinerte Umrechnungsformel:

$$F' : nachBetrag = vonBetrag \cdot wechselkurs$$

In dieser Formel spielt es jetzt keine Rolle mehr, welche Währungen beteiligt sind, es muss nur der passenden Umrechnungskurs verwendet werden. Das führt dann zu folgendem Smalltalk-Programm:

```
| nachBetrag vonBetrag wechselkurs |
nachBetrag := vonBetrag * wechselkurs.
Dialog warn: 'umgerechneter Betrag: ' nachBetrag printString
```

Dieses ist durch *SmaViM* so natürlich nicht ausführbar, da die Variablen `vonBetrag` und `wechselkurs` keinen Wert besitzen und damit das Produkt aus beiden undefiniert ist. Letztlich muss der Anwender sagen, welcher Betrag zu welchem Wechselkurs umgerechnet werden soll. Dazu werden nun Anweisungen benötigt, die, wenn sie von *SmaViM* ausgeführt werden, dem Anwender die Eingabe eines Zahlenwerts ermöglichen und den eingegebenen Wert dann einer der Variablen `vonBetrag` bzw. `wechselkurs` zuordnen. Für diesen Zweck stehen auch wieder einfache Dialogfenster zur Verfügung. So wird z. B. durch die Anweisung

Dialogfenster für Eingabe

```
Dialog request: 'umzurechnender Betrag: ' initialAnswer:'0'
```

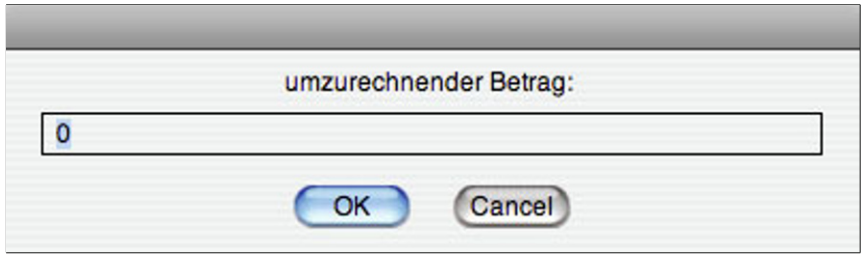

Abbildung 2.9: Eingabedialog

das in Abbildung 2.9 dargestellte Dialogfenster aufgeblendet. Hinter `request:` kann ein beliebiger Text (wiederum einzuschließen in einfache Hochkommata) angegeben werden, der hier – als Eingabeaufforderung verstanden – im oberen Teil des Dialogfensters erscheint. Das darunter liegende umrandete Feld dient der Benutzereingabe. Es enthält den hinter dem Schlüsselwort `initialAnswer:` angegebenen Text als Voreinstellung, die als Eingabewert übernommen würde, wenn der Benutzer keine weitere Eingabe vornähme. Führt man die obige Anweisung im Workspace mit dem Kommando **Print it** aus, und gibt in das Eingabefeld dann z. B. 250 ein, erscheint nach der Bestätigung des Dialogs durch Betätigen der **OK**-Schaltfläche mit der Maus im Workspace als Ergebnis der Ausführung der Text `'250'` (s. Abbildung 2.10).

Das Ergebnis einer Eingabe in einem Dialogfenster ist immer ein Text. Da wir aber nur mit Zahlen rechnen können, muss dieser Text noch in eine Zahl verwandelt werden, was natürlich nur dann möglich ist, wenn der eingegebene Text als Zahlenwert betrachtet werden kann. Was das genau heißt, d. h. welche syntaktischen Regeln für das Aufschreiben von Zahlen gelten, wird in Abschnitt 8.1.2 betrachtet. Die Umwandlung eines Textes in eine Zahl ist mithilfe der Anweisung `asNumber` möglich. Abbildung 2.11 zeigt die entsprechend ergänzte Anweisung und wiederum das Ergebnis ihrer Ausführung mit **Print it**. Dieses ist jetzt die *Zahl* 250 und kein Text mehr; man beachte die fehlenden Hochkommata (vgl. Abbildung 2.10).

Auf diese Art eingegebene Zahlen können nun den Variablen `vonBetrag` und `wechselkurs` zugewiesen werden. Dazu dient das folgende Programmstück:

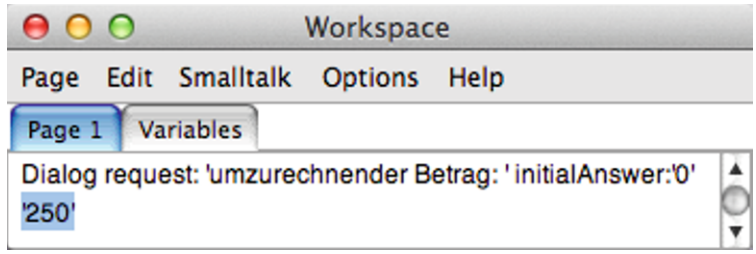

Abbildung 2.10: Ergebnis der Eingabe in das Dialogfenster

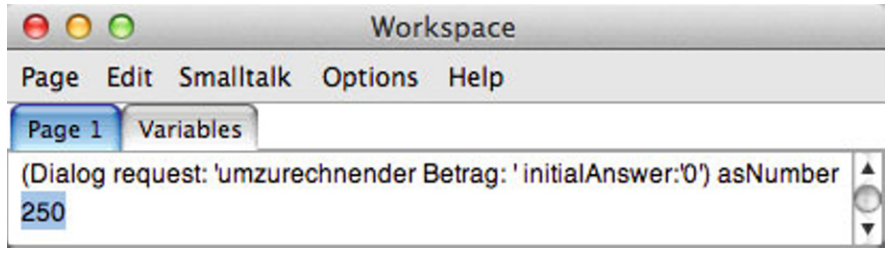

Abbildung 2.11: Umwandlung der Dialogeingabe in eine Zahl (Number)

```
vonBetrag := (Dialog request: 'umzurechnender Betrag: '
                     initialAnswer: '0') asNumber.
wechselkurs := (Dialog request: 'Wechselkurs: '
                     initialAnswer: '1') asNumber
```

Man beachte hier, dass die Anweisungen für die Dialogeingabe hier über zwei Zeilen geschrieben sind. Das ist nicht notwendig, aber erlaubt. Leerraum und Zeilenwechsel sind in Smalltalk-Programmen in der Regel bedeutungslos.

Damit können wir nun unser Programm für die Währungsumrechnung vervollständigen:

```
| nachBetrag vonBetrag wechselkurs |
vonBetrag := (Dialog request: 'umzurechnender Betrag: '
                     initialAnswer: '0') asNumber.
wechselkurs := (Dialog request: 'Wechselkurs: '
                       initialAnswer: '1') asNumber.
nachBetrag := vonBetrag * wechselkurs.
Dialog warn: 'umgerechneter Betrag: ' ,nachBetrag printString
```

Mit diesem Programm können nun beliebige Beträge zu beliebigen Wechselkursen umgerechnet werden. Es ist hier an dieser Stelle darauf hinzuweisen, dass die Ein- und Ausgabe über primitive Dialogboxen, wie hier geschehen, natürlich nichts mit einer ergonomisch akzeptablen Benutzungsoberfläche, wie man sie von modernen Programmen erwartet, zu tun hat. Ein „professionell“ gemachtes Programm präsentierte sich dem Bediener eher wie in Abbildung 2.12 gezeigt. Ein solches Programm holte sich

http://www.springer.com/978-3-658-00630-3

Grundkurs Smalltalk - Objektorientierung von Anfang an
Eine Einführung in die Programmierung
Schröder, H.
2014, XIII, 403 S. 224 Abb., Softcover
ISBN: 978-3-658-00630-3