3 Grundzüge der objektorientierten Programmierung mit Smalltalk

Die hervorstechendste Eigenschaft der Sprache Smalltalk könnte in dem Satz zusammengefasst werden: Programme bestehen ausschließlich aus *Nachrichten*, die an *Objekte* gesandt werden. Damit sind zwei elementare Begriffe der objektorientierten Programmierung benannt, deren Bedeutung für die Programmiersprache Smalltalk in Abschnitt 3.1 detailliert behandelt wird. Hier werden zunächst einige einführende Betrachtungen in die Motivation, den objektorientierten Programmierstil zu benutzen, angestellt.

Objekte empfangen Nachrichten

Naturgemäß taucht der Begriff *Objekt* immer wieder auf. Was aber meint der Software-Techniker damit? Wenn man sich in seiner Umgebung umschaut, findet man viele Objekte der realen Welt. Da sind z. B. Gegenstände, die man sehen und anfassen kann: ein Fahrzeug, ein Fernsehapparat, ein Möbelstück. Aber auch eine Katze oder eine Person bezeichnen wir – vielleicht etwas respektlos – als Objekt. Allen diesen Objekten ist gemeinsam, dass sie einen Zustand besitzen, der durch bestimmte Merkmalsausprägungen gegeben ist, und ein Verhalten aufweisen. Der Zustand einer Hauskatze könnte z. B. durch ihren Namen, ihre Fellzeichnung und den Grad ihres Wachseins bestimmt sein. Zu ihrem Verhalten zählt u. a. Fressen, Jagen und das Spitzen der Ohren. Der Zustand eines Fahrzeuges wird vielleicht durch seine Antriebsart, die Anzahl der Räder und seine momentane Geschwindigkeit beschrieben, das Verhalten durch Beschleunigen, Abbremsen und Richtungsänderung. Objekte können aber auch abstrakter Natur sein, wie z. B. Ereignisse, Geldbeträge oder Kaufverträge.

Eine wesentliche Aufgabe bei der Entwicklung von Software besteht nun darin, die reale Welt im Computer gewissermaßen zu rekonstruieren, indem der Zustand und das Verhalten realer Objekte durch Programme nachgebildet werden. Dieses Abbild der realen Welt bezeichnet man auch als *Modell*, den Vorgang es zu erstellen als *Modellierung* oder *Modellbildung*. Es ist vielleicht hier schon einleuchtend, dass eine Programmiersprache, die über ein Objekt-Konzept verfügt, das dem realer Objekte nachempfunden ist, diese Rekonstruktion der Realität erleichtert. Objektorientierte Programmiersprachen kennen daher Software-Objekte, die ebenfalls Zustand und Verhalten besitzen. Wie diese technisch genau ausgestaltet werden, wird im folgenden Abschnitt erläutert.

Modellierung

Rekonstruktion der Realität

Eine wichtiger Aspekt der Modellbildung ist die *Abstraktion*. Das Abbild eines Objekts der realen Welt im Computer wird in der Regel weniger Eigenschaften aufweisen als das Realwelt-Objekt selbst. Die Gesamtheit aller Merkmale von Hauskatzen ist so vielfältig, dass sie in einem Modell kaum erfasst werden könnten. Auch das Verhalten eines lebenden Organismus ist viel zu komplex, um vollständig technisch nachgebildet werden zu können. Die Reaktion einer Hauskatze auf bestimmte äußere Reize ist weder exakt vorhersagbar noch überhaupt vollständig determiniert.

Abstraktion

Die exakte Nachbildung von Realwelt-Objekten ist daher häufig nicht möglich, aber auch gar nicht notwendig. Denn die Modellierung geschieht ja immer vor dem Hintergrund der Entwicklung eines bestimmten Anwendungsprogramms, so dass von denjenigen Eigenschaften der Realwelt-Objekte *abstrahiert* werden kann, die für die jeweilige Anwendung bedeutungslos sind. Wenn Kraftfahrzeuge für die Verwaltung eines Fuhrparks modelliert werden sollen, spielt das Merkmal *momentane Geschwindigkeit* keine Rolle und auch Verhaltensweisen wie *Lenken* oder *Beschleunigen* sind bedeutungslos. Will man hingegen die Steuerungs-Software für einen Fahrsimulator entwickeln, darf von diesen Objekteigenschaften gerade nicht abstrahiert werden.

Abstraktion ist ein wichtiges Prinzip, das ganz allgemein im menschlichen Zusammenleben eine wichtige Rolle spielt. Man denke nur daran, wie sich die Durchführung von Bezahlvorgängen verändert hat. Der Übergang vom Tauschhandel zum Bezahlen mit Geld war schon ein bedeutender Abstraktionsprozess. Der bargeldlose Zahlungsverkehr, der sich erst durch seine technische Umsetzung in dem heute festzustellenden Ausmaß hat durchsetzen können, stellt eine weitere Abstraktionsstufe dar. Bestimmte Eigenschaften eines Bargeldbetrags – z. B. seine Stückelung in bestimmte Münzsorten oder Geldscheine – spielen nun keine Rolle mehr.

Da die Automatisierung von menschlicher Arbeit immer mit Modellbildung verknüpft ist, stellt Abstraktionsvermögen eine der wichtigsten Fähigkeiten dar, die von einem Software-Entwickler verlangt werden müssen. Die objektorientierte Programmierung soll den Programmierer dabei unterstützen, indem sie ihm sozusagen dadurch entgegenkommt, dass sie ihm mit den Software-Objekten ein mächtiges Modellierungsmittel an die Hand gibt. Damit verbindet man die Hoffnung, dass sich das Ziel, anforderungsgerechte Software wirtschaftlich zu erstellen, leichter erreichen lässt als mit anderen Programmierstilen.

objektorientiertes Programm

Nachrichtensequenz

In der objektorientierten Programmierung wird ein Programm als eine geordnete, zielgerichtete Abfolge von Nachrichten, die an geeignete Objekte gesandt werden, gesehen. Das Verhalten der Objekte als Reaktion auf den Empfang von Nachrichten kann in einer Zustandsänderung der Objekte oder auch in dem Versenden von Nachrichten an andere Objekte bestehen.

3.1 Objekte, Nachrichten, Methoden

Gegenüber der herkömmlichen Programmierung ist mit der Objektorientierung eine neue Terminologie verbunden, zum großen Teil für Konzepte und Techniken, die auch schon vorher bekannt waren. Dessen ungeachtet werden jetzt die grundlegenden Begriffe der Objektorientierung im Vordergrund stehen, so wie sie mit Smalltalk eingeführt und für die meisten anderen objektorientierten Sprachen übernommen wurden.

Zahlen sind Objekte

Nun wurde in Kapitel 2 bereits ein vollständiges Smalltalk-Programm vorgestellt, ohne dass einer der beiden Begriffe *Objekt* oder *Nachricht* dabei verwendet worden wäre. Im Währungsumrechner und in dem Programm für die Lösung einer quadratischen Gleichung wurde eigentlich nur mit Zahlen operiert. Zu den typischen Eigenschaften von Smalltalk gehört, dass auch Zahlen als Objekte betrachtet werden. Das mag auf den ersten Blick insofern überraschen, als wir mit Zahlen normalerweise kein Verhalten verbinden. Smalltalk ist nun aber sehr konsequent objektorientiert: Alles ist ein Objekt, also auch Zahlen. Was es dann mit dem Verhalten solcher Zahl-Objekte auf

sich hat, wird im Verlaufe dieses Abschnitts deutlich werden.

Bevor wir die im Kapitel 2 entwickelten Programme unter dem Blickwinkel der Objektorientierung noch einmal anschauen, werden hier die Betrachtungen zur „Natur“ von Objekten im Sinne der objektorientierten Programmierung vertieft werden. Ein Objekt besteht aus zwei Teilen:

- der *Zustand* des Objekts, repräsentiert durch eine Datenstruktur — Zustand
- das *Verhalten* des Objekts, das die Art und Weise festlegt, in der das Objekt auf den Empfang einer Nachricht reagiert. — Verhalten

Der Ablauf eines objektorientierten Programms besteht also darin, dass Objekte Nachrichten austauschen. Reaktionen von Objekten auf den Empfang von Nachrichten können z. B. sein:

- das Objekt gibt Auskunft über seinen Zustand
- das Objekt verändert seinen Zustand

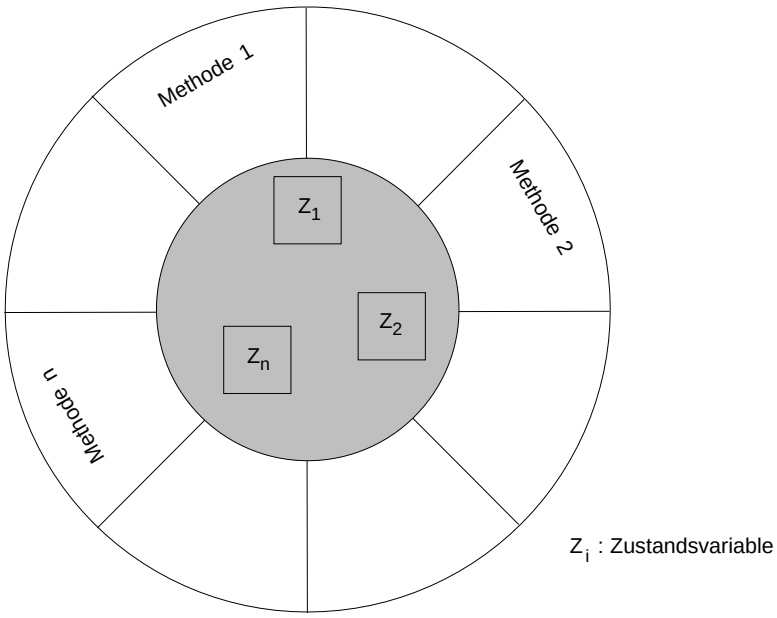

Abbildung 3.1: Objektaufbau

Grundsätzlich gilt, dass der Zustand eines Objektes von außen nicht direkt sichtbar ist, sondern nur durch das Senden von Nachrichten sichtbar gemacht, bzw. verändert werden kann. Dieses Merkmal von Objekten bezeichnet man als *Geheimnisprinzip* (engl.: information hiding), das zu den wichtigsten Kennzeichen der Objektorientierung gehört. Dies wird durch die Darstellung in Abbildung 3.1 verdeutlicht. Ein Objekt verfügt für jede Nachricht, die es versteht, über eine so genannte *Methode*[1], in der die Reaktion des Objekts auf den Empfang der entsprechenden Nachricht festgelegt (programmiert) ist. Die Methoden bilden sozusagen die nach außen sichtbare „Schale“ – auch *Schnittstelle* (engl.: interface) genannt – des Objekts. Die Datenstruktur, die

Geheimnisprinzip

Methode

[1] vergleichbar mit einer Prozedur in einer herkömmlichen, imperativen Programmiersprache

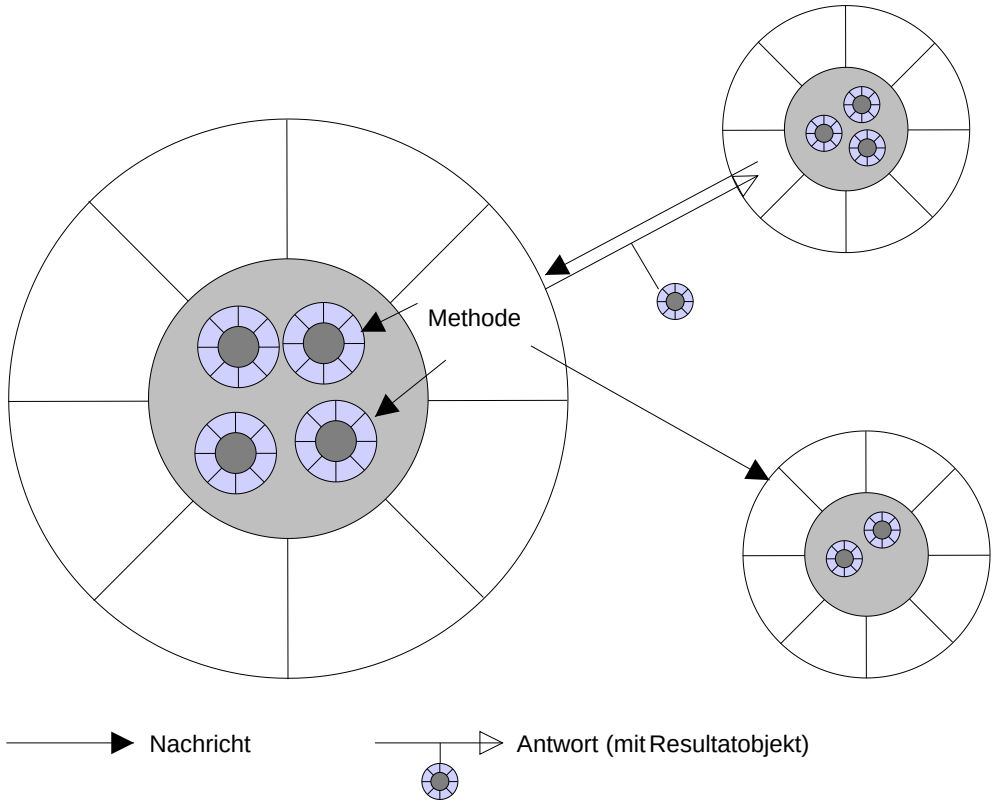

Abbildung 3.2: Nachrichtenaustausch

durch die Zustandsgrößen Z_i gebildet wird, ist der nach außen unsichtbare „Kern“ des Objekts. Die Zustandsgrößen heißen auch interne Variablen (oder *Exemplarvariablen* (engl.: instance variables)) und dienen zur Aufnahme der Objekte , die den internen Zustand des Objektes repräsentieren.

Exemplarvariablen

Die Ausführung einer Methode als Reaktion auf den Empfang einer Nachricht besteht nun wiederum im Versenden weiterer Nachrichten an andere Objekte einschließlich der durch die internen Variablen repräsentierten. Am Ende wird immer ein Objekt als Resultat an den Sender der Nachricht zurückgeliefert. In Abbildung 3.2 wird dieser Vorgang veranschaulicht. Die internen Variablen der Objekte sind hier auch als Objekte (mit „Schale“ und „Kern“) dargestellt. Aus Gründen der Übersichtlichkeit ist nur eine Rückantwort eingezeichnet, obwohl das Versenden jeder Nachricht die Rücksendung eines Objekts zur Folge hat.

> Anmerkung zur Terminologie: In Kapitel 2 haben wir den Begriff *Methode* im umgangssprachlichen Sinn als Synonym für Verfahren, Vorgehensweise, Algorithmus o. ä. verwendet. Hier haben wir jetzt einen speziellen Begriff eingeführt, wie er in der Terminologie der objektorientierten Programmierung üblich ist. Um Verwechslungen der jeweiligen Bedeutung des Begriffs Methode zu vermeiden, wird er im Folgenden nur noch im speziellen, objektorientierten Sinn verwendet werden. Ist die umgangssprachliche Bedeutung gemeint, wird stattdessen der Begriff *Verfahren* benutzt. Wie oben erläutert, hat das Senden einer Nachricht die Ausführung einer Methode zur Folge. Diesen Vorgang bezeichnet man als *Methodenaktivierung* oder in Anlehnung an den in der prozeduralen Programmierung benutzten Begriff des Prozeduraufrufs auch als *Methodenaufruf*.

Betrachten wir nun noch einmal Teile unserer im Kapitel 2 entwickelten Programme. Dort traten z. B. arithmetische Ausdrücke der Art

```
((2.1 * 2.1) + (4 * 5.4)) sqrt
```

auf. Der Teilausdruck `4 * 5.4` ist in objektorientierter Terminologie folgendermaßen zu lesen: Dem Objekt „4“ wird die Nachricht (engl: message) „* 5.4“ gesendet. Das Empfängerobjekt „4“ antwortet auf den Empfang der Nachricht mit dem Ergebnis der Multiplikation, dem Objekt „21.6“. Selbstverständlich ist auch „5.4“ ein Objekt, das hier als so genanntes *Argument*[2] Bestandteil der Nachricht „* 5.4“ ist. Das Zeichen „*“ bildet hier den so genannten *Nachrichtenselektor* (oder kurz: Selektor), der dem Empfänger der Nachricht letztlich sagt, welche Art von Operation (hier die Multiplikation) ausgeführt werden soll, d. h. welche Methode des Empfängerobjekts zum Einsatz kommt. Dem Objekt „4“ wird durch die Nachricht also die Aufforderung übermittelt: „Multipliziere deinen Wert mit 5.4 und gib das Ergebnis der Multiplikation als Antwort zurück.“

Argument

Nachrichtenselektor

> **Merke:** In Smalltalk ist ein wichtiges Merkmal der Reaktion von Objekten auf den Empfang von Nachrichten, dass das Empfängerobjekt immer eine Antwort in Form eines Objektes zurückliefert.

Betrachten wir nun, wie der gesamte Ausdruck

```
((2.1 * 2.1) + (4 * 5.4)) sqrt
```

durch *SmaViM* abgearbeitet wird. Grundsätzlich gilt in Smalltalk für zusammengesetzte Nachrichtenausdrücke, dass die darin enthaltenen Nachrichten strikt von links nach rechts abgearbeitet werden, sofern nicht durch Setzen von Klammern eine abweichende Ausführungsreihenfolge erzwungen wird. Es gibt auch für die Nachrichtenselektoren wie „*“ oder „+“, die für die Grundrechenarten stehen, keine Vorrangregeln der Art „Punktrechnung geht vor Strichrechnung“. Damit ergibt sich für den Ausdruck folgende Ausführungssequenz:

1. Dem Objekt „2.1“ wird die Nachricht „* 2.1“ geschickt. Das Objekt antwortet mit dem Resultatobjekt „4.41“. Den Ausdruck kann man sich nun ersetzt denken durch `(4.41 + (4 * 5.4)) sqrt`.

2. Aufgrund der Klammersetzung wird als nächstes dem Objekt „4“ die Nachricht „* 5.4“ gesendet, was wiederum mit dem Objekt „21.6“ antwortet. Der Ausdruck hat nun die Form `(4.41 + 21.6) sqrt`.

3. Das Objekt „4.41“ tritt nun als Empfänger der Nachricht „+ 21.6“ auf. Das Antwortobjekt ist also „26.01“.

4. Der resultierende Ausdruck lautet nun `26.01 sqrt`. Der Nachrichtenselektor heißt hier „sqrt“. Durch ihn wird das Empfängerobjekt aufgefordert, die Wurzel seines Wertes zu berechnen und mit dem Ergebnis zu antworten.

5. Das Objekt „5.1“ stellt das Endergebnis dar ($\sqrt{26.01} = 5.1$).

[2] gebräuchlich ist hier auch der Begriff *Parameter*

3.1.1 Nachrichten

Nachrichtenausdruck

Die Grundform des Nachrichtenausdrucks in Smalltalk lautet:

```
<objekt> <nachricht>
```

Dabei benennt `<objekt>` das Empfängerobjekt der Nachricht `<nachricht>`. Das Empfängerobjekt kann aber seinerseits das Ergebnis eines Nachrichtenausdrucks sein. Im obigen Beispiel ist das Antwortobjekt des Ausdrucks „2.1 * 2.1“ der Empfänger der Nachricht „+ (4 * 5.4)“.

unäre Nachrichten

Wie wir in dem Beispiel bereits gesehen haben, gibt es Nachrichten, die keine Argumente benötigen. Diese heißen *unäre Nachrichten* (engl.: unary messages). „sqrt“ ist ein Beispiel für eine unäre Nachricht. Nachrichten, die mit genau einem Argument versehen werden müssen, wie z. B. „*“ oder „+“, heißen *binäre Nachrichten* (engl.: binary messages).

binäre Nachrichten

Hier einige weitere Beispiele für unäre Nachrichten:

2 negated	Ergebnis: -2
25 factorial	Ergebnis: $25! = 15511210043330985984000000$
Window new	Ergebnis: eine neues Fenster auf dem Bildschirm
alpha sin	Ergebnis: *sin alpha*, wobei alpha eine Variable sein muss, der ein numerischer Wert zugewiesen wurde, der als Winkel im Bogenmaß interpretiert wird

Kennzeichnend für unäre Nachrichten ist, dass sie immer durch ein einzelnes Wortsymbol (z. B. `negated`) dargestellt werden.

Zu den binären Nachrichten in Smalltalk gehören die Grundrechenarten Addition, Subtraktion, Division und Multiplikation (`+, -, /, *`). Hinzu kommen die so genannten Vergleichsoperationen ($=, <, >, <=, >=, \sim=$). Diese wurden im Programm für die Lösung der quadratischen Gleichung schon mehrfach benutzt, z. B. um zu prüfen, ob einer der Koeffizienten gleich 0 ist:

```
(a = 0)  ifTrue: [x := c negated / b]
```

true, false

Die Nachricht „= 0“ bedeutet also dem Empfängerobjekt „a“ zu prüfen, ob sein Wert gleich 0 ist. Das Objekt antwortet mit ja oder nein. Genauer: Das Antwortobjekt heißt entweder `true` oder `false`. Bei `true` und `false` handelt es sich um so genannte Pseudovariablen, deren Wert nicht verändert werden kann. Die Pseudovariable `true` steht für den Wahrheitswert „wahr“, während `false` den Wahrheitswert „falsch“ repräsentiert. `true` und `false` werden auch als boolesche Objekte bezeichnet.

Das Argument einer binären Nachricht kann wiederum ein beliebiger Nachrichtenausdruck sein, dessen Auswertung ja ein Objekt ist, das dann letztlich als Argument der Nachricht zum Tragen kommt. In dem Ausdruck `4.41 + (4 * 5.4)` ist der geklammerte Ausdruck das Argument der Nachricht „+“. Wie schon erläutert, wird dieser ausgewertet und das Ergebnisobjekt mit der Nachricht „+“ an das Empfängerobjekt „4.41“ übergeben.

Weitere binäre Nachrichten werden in späteren Kapiteln behandelt. Kennzeichnend für binäre Nachrichten ist, dass sie immer durch einfaches (z. B. +) oder zusammengesetztes (z. B. <=) Operationszeichen dargestellt werden.

Schlüsselwortnachricht

Die dritte und letzte Nachrichtenform ist die *Schlüsselwortnachricht* (engl.: keyword message). Schlüsselwörter sind einfache Wortsymbole mit angehängtem Doppelpunkt.

In

```
(a = 0)  ifTrue: [x := c negated / b]
```

ist `ifTrue:` ein solches Schlüsselwort. Zu jedem Schlüsselwort gehört ein Argument. Hier ist es der in eckige Klammern eingeschlossene Block. Die Schlüsselwortnachricht in diesem Beispiel lautet

```
ifTrue: [x := c negated / b]
```

Das Empfängerobjekt ist das aus der Auswertung von `(a = 0)` resultierende boolesche Objekt. Die Wirkung von „`ifTrue:`“ ist ja in Kapitel 2 bereits erläutert worden. Genauer wird dies noch einmal in Abschnitt 3.1.2 behandelt.

Ein weiteres einfaches Beispiel für die Anwendung einer Schlüsselwortnachricht ist:

```
a max: b
```

Vorausgesetzt, die beiden Variablen repräsentieren Objekte, die hinsichtlich ihrer Größe vergleichbar sind, also z. B. Zahlen, liefert dieser Ausdruck als Ergebnisobjekt `a`, falls a größer als b ist, sonst liefert er `b`.

Wichtig ist, dass Schlüsselwortnachrichten aus mehreren Schlüsselwörtern bestehen können. Die Nachricht `ifTrue:ifFalse:` ist ein Beispiel dafür, das wir bereits kennen. Zu jedem Schlüsselwort gehört ein Argument, das hinter den jeweiligen Doppelpunkt geschrieben wird. Ein Anwendungsbeispiel aus Kapitel 2 ist:

```
(c = 0)
  ifTrue: [Dialog warn: 'Lösung trivial']
  ifFalse: [Dialog warn: 'Widerspruch']
```

Die Anzahl der Schlüsselwörter für eine Schlüsselwortnachricht ist in Smalltalk nicht begrenzt.

Zu klären bleibt noch, in welcher Reihenfolge Nachrichten an Objekte gesendet werden, wenn in einem komplexen Nachrichtenausdruck mehrere Nachrichten unterschiedlichen Typs auftreten. Grundsätzlich gelten folgende Regeln:

1. Nachrichten in Klammern werden zuerst ausgewertet.
2. Unäre Nachrichten werden vor binären ausgewertet.
3. Binäre Nachrichten werden vor Schlüsselwortnachrichten ausgewertet.
4. Treten mehrere Nachrichten der gleichen Art auf, werden sie strikt von links nach rechts abgearbeitet.

Beispiele:

- In `3 + 4 sqrt` wird wegen Regel 2 erst die Wurzel aus 4 berechnet und diese anschließend auf 3 addiert.
- In `7 + 2 * 3` wird wegen Regel 4 zuerst die Summe gebildet und anschließend multipliziert.

- In `6 max: 3 + 4` wird wegen Regel 3 zuerst `3 + 4` ausgeführt und anschließend die Nachricht `max: 7` an `6` gesendet.
- In der folgenden Tabelle wird die Auswertung des Ausdrucks

  ```
  2 + 4 * (6 negated max: 8 - 3) negated
  ```

  verdeutlicht. Dabei wird in der linken Spalte die innerhalb des Ausdrucks als nächste auszuwertende Nachricht hervorgehoben dargestellt. Die mittlere Spalte zeigt das Resultat dieser Nachrichtenauswertung. Dieses Resultat erscheint innerhalb des Ausdrucks in der jeweils nachfolgenden Zeile anstelle der Nachricht. Die dritte Spalte gibt die Regeln an, die der Hervorhebung der Nachricht zugrunde liegen.

nächste auszuwertende Nachricht	**Antwort**	**Regel(n)**
2 + 4 * (**6 negated** max: 8 - 3) negated	-6	1 und 2
2 + 4 * (-6 max: **8 - 3**) negated	5	3
2 + 4 * (**-6 max: 5**) negated	5	1
2 + 4 * **5 negated**	-5	2
2 + 4 * -5	6	4
6 * -5	-30	-

Eine kleine Schwierigkeit ergibt sich, wenn in einem Ausdruck mehrere Schlüsselwort-Nachrichten auftreten. Nehmen wir an, wir wollten das Maximum aus einer Variablen `z` und dem Minimum der Variablen `x` und `y` ermitteln. Naiverweise könnte der Ausdruck dann folgendermaßen aussehen:

```
x min: y max: z
```

Die Annahme, dass die Nachrichten `min:` und `max:` von links nach rechts abgearbeitet werden ist aber falsch. Versucht man den Ausdruck im Workspace auszuführen, erscheint die in Abbildung 3.3 gezeigte Fehlermeldung. *SmaViM* ist nämlich nicht in der

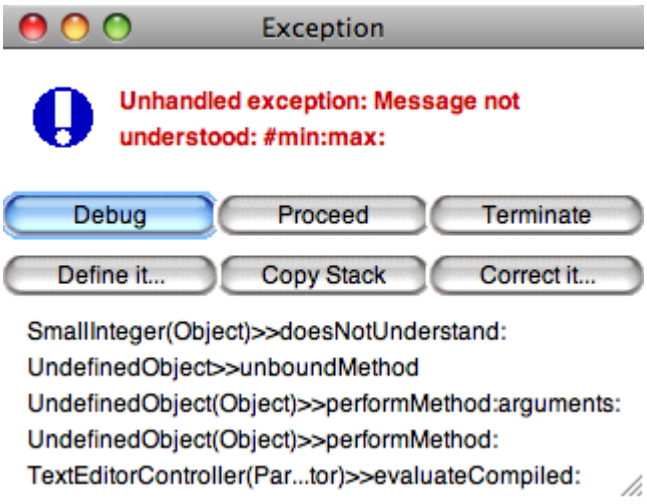

Abbildung 3.3: Fehler: Empfänger versteht die Nachricht `min:max:` nicht.

Lage, die Hintereinanderausführung von zwei Schlüsselwort-Nachrichten, die jeweils aus einem Schlüsselwort (hier `min:` und `max:`) bestehen, von einer Schlüsselwort-Nachricht, die aus zwei Schlüsselwörtern besteht (hier `min:max:`), zu unterscheiden. Hintereinander auszuführende Schlüsselwort-Nachrichten müssen daher immer geklammert werden. Das obige Beispiel muss daher so aufgeschrieben werden:

```
(x min: y) max: z
```

3.1.2 Fallunterscheidung

Fallunterscheidungen werden in Smalltalk durch eine der vier Nachrichten `ifTrue:`, `ifFalse:`, `ifTrue:ifFalse:` oder `ifFalse:ifTrue:` programmiert, wovon in den vorangegangenen Beispielen ja schon Gebrauch gemacht worden ist.

> Anmerkung für Kenner konventioneller höherer Programmiersprachen, wie z. B. PASCAL: In diesen Sprachen steht für Fallunterscheidungen in der Regel ein eigener Anweisungstyp mit spezieller Syntax zur Verfügung (z. B. die if-then-else-Anweisung in PASCAL). In Smalltalk hingegen werden Fallunterscheidungen als gewöhnliche Schlüsselwortnachrichten realisiert.

Diese Nachrichten werden nur von den beiden booleschen Objekten, die durch die Pseudo-Variablen `true` und `false` repräsentiert werden, verstanden. Betrachten wir noch einmal das Beispiel

```
(a = 0) ifTrue: [x := c negated / b].
```

Etwas präziser als bisher formuliert, wäre die Wirkung folgendermaßen zu beschreiben: Ist `true` das Empfängerobjekt von `ifTrue:`, so antwortet dieses mit dem Objekt, das aus der Auswertung des Blocks resultiert. Ist das Empfängerobjekt hingegen `false`, ignoriert dieses den als Parameter übergebenen Block – d. h. er wird nicht ausgewertet – und antwortet mit dem undefinierten Objekt, das durch die Pseudo-Variable `nil` repräsentiert wird. Die Wirkung der Nachrichten `ifFalse:`, `ifTrue:ifFalse:` und `ifFalse:ifTrue:` ergibt sich aus einer analogen Überlegung.

`nil` repräsentiert das undefinierte Objekt

3.1.3 Blöcke

Die Argumente der Nachrichten für die Fallunterscheidungen sind Blöcke. Ein Block ist eine in eckige Klammern eingeschlossene Sequenz von Nachrichten, wobei diese Nachrichten zunächst nicht ausgewertet werden. Ob der Block einer `ifTrue:`-Nachricht ausgewertet werden darf, kann nur das Empfängerobjekt entscheiden, wie im vorigen Abschnitt erläutert wurde. Blöcke werden also immer dann verwendet, wenn Nachrichtensequenzen bedingt oder – wie wir später noch sehen werden – wiederholt ausgeführt werden sollen.

Blöcke sind aber – wie alles in Smalltalk – Objekte, die einer Variablen zugewiesen werden können und denen man auch Nachrichten schicken kann. Insbesondere versteht ein Block die Nachricht `value`, die ihn veranlasst, die Nachrichtensequenz in seinem Innern abzuarbeiten. Betrachten wir die folgenden Anweisungen:

```
x := 0.
einBlock := [x := 7].
einBlock value.
```

Blöcke sind Objekte

In der zweiten Zeile wird der Variablen `einBlock` der Block `[x := 7]` zugewiesen. Da die Anweisungen dieses Blocks aber nicht ausgeführt werden, behält die Variable `x` den Wert 0. In der dritten Zeile wird nun dem Block die Nachricht `value` geschickt. Die Anweisungen des Blocks werden daraufhin ausgeführt, die Variable `x` erhält den Wert 7.

Weitere Anwendungen für Blöcke werden in späteren Kapiteln behandelt.

3.1.4 Erzeugung von Objekten – Klassen

Bisher haben wir drei Arten von Objekten kennen gelernt, nämlich Zahlen, die durch die Pseudo-Variablen `true` und `false` repräsentierten booleschen Objekte und Zeichenketten, die wir bisher lediglich zur Anzeige von Texten in Dialogfenstern benutzt haben. Inzwischen wissen wir auch, wie wir Objekten Nachrichten schicken können. Gelegentlich ist davon gesprochen worden, dass ein Objekt bestimmte Nachrichten versteht. So verstehen Zahlen z. B. die Nachrichten „+“ oder `negated`. Es ist aber offensichtlich sinnlos, einem booleschen Objekt die Nachricht `sqrt` zu schicken oder zwei Zeichenketten multiplizieren zu wollen. D. h. es stellt sich die Frage: Welche Objekte verstehen welche Nachrichten?

Klassen

In der objektorientierten Programmierung werden „gleichartige“ Objekte in so genannten *Klassen* zusammengefasst. Objekte einer Klasse weisen die gleiche innere Struktur auf und verstehen dieselben Nachrichten. Jede Klasse hat einen Namen, der vereinbarungsgemäß mit einem großen Anfangsbuchstaben beginnt. Z. B. trägt die Klasse der ganzen Zahlen den Namen `Integer`; die Klasse `String` steht für die Zeichenketten. Die innere Struktur der Objekte der Klasse `Integer` besteht aus einem einzigen Zahlenwert, während die Struktur der Objekte der Klasse `String` aus den die Zeichenkette bildenden Einzelzeichen besteht.

Welche Nachrichten ein Objekt nun versteht, richtet sich nach der Klassenzugehörigkeit. Für jede Nachricht steht in der Klassendefinition eine Methode bereit, d. h. zu jedem Nachrichtenselektor, der von den Objekten einer Klasse verstanden wird, gehört eine Methode, deren Name mit dem Nachrichtenselektor übereinstimmt. Die Liste aller Methoden einer Klasse heißt auch das *Methodenprotokoll* der Klasse.

Methodenprotokoll

Es kann durchaus der Fall eintreten, dass Objekte verschiedener Klassen den gleichen Nachrichtenselektor akzeptieren, aber jeweils eigenständige Methoden ausführen. Z. B. können sowohl ganze Zahlen als auch Zeichenketten mit der Nachricht „<“ miteinander verglichen werden, wie die folgenden Beispiele zeigen:

`3 < 4` **Print it** liefert: `true`

`'Karl' < 'Otto'` **Print it** liefert: `true`

`'Rosa' < 'Luise'` **Print it** liefert: `false`