# Smalltalk (Programmiersprache)

### aus Wikipedia, der freien Enzyklopädie

**Smalltalk** ist eine dynamisch typisierte objektorientierte Programmiersprache und zugleich eine vollständige Entwicklungsumgebung, die in den 1970er Jahren am Xerox PARC Forschungszentrum durch Alan Kay, Dan Ingalls, Adele Goldberg und anderen entwickelt wurde. Sie wurde allgemein unter dem Namen Smalltalk-80 freigegeben und hat die Entwicklung vieler späterer Programmiersprachen, wie z.B. Java und Ruby beeinflusst. Smalltalk wurde von Lisp und Simula mit seinem Klassen-Konzept beeinflusst und wurde zur ersten objektorientierten Programmiersprache. Der Begriff "Objektorientierung" wurde von Alan Kay erfunden. Smalltalk ist im Gegensatz zu Sprachen wie C++, Java oder C# eine rein objektorientierte Programmiersprache, d.h. Datentypen wie Integer, String o.ä. sind ebenfalls vollwertige Objekte. Die Smalltalk-Entwicklungsumgebung enthielt viele Ideen, die später mit der Macintosh- und dann Windows-Benutzeroberfläche verbreitet wurden. Verwendet wurde ein Grafikbildschirm mit verschiebbaren Fenstern, Aufklappmenüs und Schriften von verschiedener Größe. Eine Maus mit drei Tasten - rot, blau und gelb - diente erstmals als zusätzliches Eingabegerät. Das Model-View-Controller-Konzept (MVC) spielte in der Smalltalk-80-Entwicklungsumgebung eine wesentliche Rolle.

**Smalltalk**

Objektorientierte Sprache

**Basisdaten**

| | |
|---|---|
| **Entwickler:** | diverse |
| **Aktuelle Version:** | Smalltalk 80 () |
| **Betriebssystem:** | Windows, Linux, Mac OS X, uvm. |
| **Kategorie:** | objektorientierte Programmiersprache |
| **Lizenz:** | implementationsspezifisch |
| **Deutschsprachig:** | implementationsspezifisch |
| **Website:** | http://www.whysmalltalk.com |

## Inhaltsverzeichnis



## Wichtige Eigenschaften von Smalltalk

- Alles in Smalltalk ist ein Objekt, auch Zeichenketten, Integer, Boolesche Werte, Klassen(-definitionen), ausführbarer Code, Stack-frames, der Speicher, Codeblöcke etc.
- Objekte werden dadurch aktiviert, dass man ihnen Nachrichten (Messages) schickt. Dies führt dazu, dass die entsprechende Methode (Funktion) des Objekts ausgeführt wird.
- Der gesamte Quelltext ist i.d.R. offen und kann somit verändert werden. Nur sehr wenige Methoden greifen auf echte 'Primitives' der VM (virtuellen Maschine) zurück.
- Dynamische Typisierung - erst zur Laufzeit wird ermittelt, welche Methode tatsächlich aufgerufen wird. Eine frühe („statische“) Bindung, wie dies bei der starken Typisierung vorgesehen ist, wurde in

der Smalltalk Programmiersprache bewusst ausgespart.
- Vererbung: Jede Klasse (außer der Wurzelklasse `Object`) ist von genau einer Oberklasse abgeleitet, deren Verhalten sie erbt und beliebig erweitern kann (keine Mehrfachvererbung).
- Automatische Speicherbereinigung (englisch *garbage collection*), die nicht durch den Programmierer beeinflusst werden muss. Ein Objekt 'lebt' solange, wie es von anderen Objekten referenziert wird.
- Smalltalkprogramme werden in Bytecode übersetzt, der durch eine virtuelle Maschine ausgeführt wird. Dadurch laufen Smalltalk-Programme ohne jegliche Änderung auf jedem System, für das eine virtuelle Maschine existiert. Ursprünglich wurde der Bytecode interpretiert; kommerzielle Umgebungen arbeiten inzwischen fast ausschließlich mit dynamischer Übersetzung.
- Üblicherweise hat man eine Programmierumgebung (selbst in Smalltalk geschrieben), in der man Quelltext am "lebenden" Objekt ändern und dann auch direkt in der geänderten Form (weiter)ausführen kann. "Smalltalkprogramme" kann man ändern, während sie laufen.
- Die Quelltexteingabe erfolgt üblicherweise im Klassenbrowser.
- Eine überraschende Eigenschaft im Vergleich zu traditionellen Sprachen ist, dass die Kontrollstrukturen wie *if-then-else*, *for*, *while* nicht in die Sprache eingebaut sind. Zumindest erscheint dies dem Programmierer so. Zum Beispiel wird eine IF-Anweisung so ausgeführt, indem eine *ifTrue:*-Botschaft an ein Boole'sches Objekt gesandt wird. Als Parameter wird ein Block (Anweisungsfolge) übergeben. Lediglich in der Klasse True wird dieser Block ausgeführt. In der Klasse False ist diese Methode zwar auch implementiert, aber sie führt den Block nicht aus.

Es gibt nur drei eingebaute ausführbare Konstrukte:

- Senden einer Botschaft an ein Objekt
- Zuweisen eines Objekts an eine Variable
- Ein Objekt als Rückgabewert einer Methode liefern

# Bedeutung von SmallTalk

SmallTalk hat nicht den Lebenslauf, den es eigentlich verdient hätte. Die Vorgehensweise der Programmierung über einen Klassenbrowser und die Verwendung einer virtuellen Maschine zu Ausführung stellt von Anfang an gewisse Mindestanforderungen an die Hardware. Dies geschah zu einer Zeit in der die meisten erschwinglichen Computer noch nicht über graphische Benutzeroberflächen verfügten und die Rechenleistung nicht für eine solche ausreichte. SmallTalk war seiner Zeit vorraus und konnte zu Beginn nur auf Workstations effektiv eingesetzt werden. Im Ergebniss gab es daher nur wenige SmallTalk-Entwickler - dafür umso mehr C-Programmierer. Mit dem Aufkommen der graphischen Benutzeroberflächen im Mikrocomputerbereich entstand C++ als ojektorientierte Alternative und da die meisten Entwickler C bereits kannten, konnte sich C++ sehr schnell verbreiten und SmallTalk blieb eine Randerscheinung.

Zudem geschah dies in einer Zeit, in der Performance sehr wichtig war und SmallTalk ist nicht gerade für extreme Ausführungsgeschwindigkeiten bekannt - dies ist aber auch nicht der Anspruch dieser Sprache. Anfang und Mitte der Neunziger konnte SmallTalk vor allem im Bereich Expertensysteme Punkten und Boden gutmachen - bis Java kam. Java hat die wesentlichen Konzepte von SmallTalk übernommmen, war jedoch in Bezug auf die Syntax an C/C++ angelehnt. Alle oben genannten Eigenschaften mit einer Ausnahme treffen auch auf Java zu - im Gegensatz dazu kennt Java jedoch aus Performancegründen primitive Datentypen.

SmallTalk wird heute nur noch in wenigen Bereichen eingesetzt und hält sich dort aus historischen Gründen.

# Ausdrücke (Expressions)

Ausdrücke haben folgende Form:

```
objekt nachricht
```

D.h. man sendet einem Objekt eine Nachricht. Das Objekt antwortet mit einem Antwortobjekt. Ausdrücke müssen mit einem Punkt getrennt werden.

Es gibt drei Arten von Nachrichten, unäre Nachrichten, binäre Nachrichten und Schlüsselwort-Nachrichten.

Unäre Nachrichten haben keinen Parameter und bestehen aus einem Bezeichner:

```
objekt nachricht
```

Eine binäre Nachricht hat genau einen Parameter und besteht aus einem oder mehreren Sonderzeichen. Vier Beispiele:

```
1 + 3.
100 @ 200.
vorname , nachname.
10 // 3
```

Die meisten arithmetischen Operationen sind in Smalltalk als binäre Nachrichten implementiert.

Eine Schlüsselwort-Nachricht hat eine oder mehrere Parameter, wobei vor den Parametern Doppelpunkte stehen.

```
objekt nachricht: parameter
```

Diese Schlüsselwort-Nachricht heißt "nachricht:" und hat einen Parameter "parameter".

```
objekt nachricht: parameter1 nachricht: parameter2
```

Diese Schlüsselwort-Nachricht heißt "nachricht:nachricht:" und hat zwei Parameter. D.h. Parameter können bei Schlüsselwort-Nachrichten mitten in die Nachricht eingefügt werden. Diese Besonderheit macht es möglich, in Smalltalk besonders leicht lesbare Programme zu schreiben:

```
collection copyFrom: 1 to: 10
```

Das klingt wie ein Satz. In Java würde man das folgendermaßen schreiben:

```
collection.copyFromTo(1, 10);
```

Ausdrücke können kombiniert und mit Klammern geschachtelt werden. Ohne Klammern werden Ausdrücke in folgender Reihenfolge ausgeführt: Unäre Nachrichten vor binären Nachrichten vor Schlüsselwort-Nachrichten

Sollen einem Objekt in Folge mehrere Nachrichten geschickt werden, besteht die Möglichkeit, diese Nachrichten mit einem Semikolon (;) zu verketten:

```
objekt
  nachricht1;
  nachricht2;
```

```
  nachricht3
```

# Zuweisungen

Eine Zuweisung hat folgende Form:

```
variable := expression
```

# Blöcke

Blöcke sind Sequenzen von Zuweisungsanweisungen und Expressions. Sie werden durch eckige Klammern eingeschlossen. Blöcke können parameterlos sein oder auch Parameter aufweisen.

```
blockEins := [ Anweisungen ]
blockZwei := [ :einParameter | Transcript show: einParameter ]
```

Diese Blöcke behalten ihren Kontext, so dass diese bei der Ausführung mit den Objekten arbeiten, die bei der Erzeugung von dort aus verfügbar waren. Um einen Block ohne Parameter auszuführen gibt es eine Parameterlose Methode. Bei dem Aufruf von Blöcken mit Parametern müssen diese mit Werten versorgt werden.

```
blockEins value
blockZwei value: 'Test'
```

# Kontrollstrukturen

Die Kontrollstrukturen werden mit booleschen Ausdrücken und Blöcken implementiert. Ein boolescher Ausdruck liefert nach der Auswertung ein Boolesches Objekt. Diesem wird dann eine Message zugesandt, die als Parameter einen ausführbaren Block hat. Der Programmtext sieht aber ähnlich aus, wie bei anderen Programmiersprachen auch, so dass man wie dort einfach gewisse Kontrollstrukturen als gegeben anwenden kann.

Einige Beispiele.

## IF-Anweisung

```
 aBoolean ifTrue: [ ein Block mit Anweisungen ]
```

oder

```
 ( einAusdruck ) ifTrue: [ ein Block mit Anweisungen ]
```

Das aus anderen Programmiersprachen bekannte *else* sähe folgendermaßen aus:

```
 aBoolean ifTrue: [ ... ] ifFalse: [ ... ]
```

### Schleifen

```
 10 timesRepeat: [ Transcript show: '.' ].
 1 to: 10: do: [ :i | Transcript show: i printString ].
 [ ein Block, der ein boole'sches Objekt zurückgibt ] whileTrue.
 [ ein Block, der ein boole'sches Objekt zurückgibt ] whileFalse.
 [ ein Block, der ein boole'sches Objekt zurückgibt ] whileTrue: [ einAndererBlock ].
 [ ein Block, der ein boole'sches Objekt zurückgibt ] whileFalse: [ einAndererBlock ]
```

# Collections

Die Smalltalkumgebungen sind mit einer großen Klassenbibliothek ausgestattet. Ein wichtiger allgemeiner Objekttyp (Klasse) sind die *Collections*, d.h. Sammlungen von Objekten. Die Klasse *Collection* steht an der Wurzel einer ganzen Hierarchie von Klassen.

Eine wichtige Arbeit, die man mit einem Collection-Objekt durchführen kann ist für jedes Element der Collection einen Block von Anweisungen auszuführen. Andere Programmiersprachen brauchen hierfür spezielle Konstrukte (Iteratoren).

```
 aCollection do: [ :einElement  | auszuführendeArbeitFürDiesesElementMit: einElement ]
```

Es hat sich eingebürgert, den Parameter als *each* zu bezeichnen, so dass direkt klar ist was hierbei gemeint ist. Bei Verwendung von mehreren Collections sollte man dem noch eine genaue Bezeichnung nachsetzten. Iteriert man bspw. über alle x und y Werte aus zwei unterschiedlichen Collections:

```
 xWerte do: [:eachX | yWerte do: [:eachY | map addPoint: eachX withY: eachY]]
```

# Programmbeispiel

Das klassische *HelloWorld*-Beispiel sieht wie folgt aus:

```
Transcript show: 'Hello World!'.
```

*Transcript* ist eine in jeder Smalltalkumgebung vordefinierte globale Variable, die ein Objekt enthält, auf dem man Dinge protokollieren kann (ein Ausgabefenster).

Wir senden diesem Objekt die Message

```
   show: aString
```

Ein bereits etwas komplexeres Beispiel:

```
'Hello World' do: [ :eachChar|
    Transcript show: eachChar ; cr.
].
```

gibt den Text "Hello World" vertikal aus. 'Hello World' ist ein String (Zeichenkette). Die Klasse dieses Objektes ist eine Unterklasse von Collection. Ein String ist also eine Collection von Zeichen. Indem wir dem Objekt String die Message

```
aString do: [ :eachChar | OperationenMitDiesemBuchstaben ]
```

senden gehen wir alle Buchstaben (Elemente) des Strings einzeln durch.

Das *Dictionary* (in Perl *Hash*, in Java *HashMap*) ist eine in Smalltalk häufig verwendete Datenstruktur:

```
d := Dictionary new.
d at: 'grün' put: 'green'.
d at: 'blau' put: 'blue'.
d at: 'rot'  put: 'red'.

Transcript show: (d at: 'blau').
```

Alternativ können die Nachrichten auch wie bereits beschrieben verkettet werden:

```
d := Dictionary new.
d at: 'grün' put: 'green';
  at: 'blau' put: 'blue';
  at: 'rot'  put: 'red'.

Transcript show: (d at: 'blau').
```

Auch das *Dictionary* ist eine Unterklasse von *Collection*. Diese Datenstruktur entspricht dem assoziativen Array in anderen Programmiersprachen.

# Literatur


- Adele Goldberg: *Smalltalk-80, The Interactive Programming Environment.* Addison-Wesley, 1983, ISBN 0201113724
- Glen Krasner: *Smalltalk-80, Bits of History, Words of Advice.* Addison-Wesley, 1. August 1983, ISBN 0-201-11669-3
- Johannes Brauer: *Grundkurs Smalltalk - Objektorientierung von Anfang an. Eine Einführung in die Programmierung* Vieweg-Verlag, 2. Auflage 2004, ISBN 3-528-15818-2
- Sherman R. Alpert, Kyle Brown, Bobby Woolf: *The Design Patterns Smalltalk Companion.* Addison-Wesley Professional, 10. Februar 1998, ISBN 0201184621
- Mark J. Guzdial: *Squeak: Object-Oriented Design with Multimedia Applications.* Prentice Hall, 20. Dezember 2000, ISBN 0130280283
- Mark J. Guzdial, Kimberly M. Rose: *Squeak: Open Personal Computing and Multimedia.* Prentice Hall, 2. August 2001, ISBN 0130280917
- Simon Lewis: *The Art and Science of Smalltalk.* Prentice Hall, 1. März 1995, ISBN 0133713458
- Smalltalk Bücher als PDF Dokumente zum herunter laden (http://www.iam.unibe.ch/~ducasse/FreeBooks.html)


# Weblinks

- "Why Smalltalk?" (http://www.whysmalltalk.com/), eine Portalseite rund um Smalltalk
- **Smalltalk User Groups:**
  - European Smalltalk Users' Group (http://www.esug.org/)
  - German Smalltalk Users' Group (http://www.gsug.org/)

### Implementierungen

- Squeak, eine freie Multimedia-Smalltalk-Implementierung
  - Deutschsprachige Seite für Squeak (http://www.squeak.de/)
  - englische Hauptseite für Squeak (http://www.squeak.org/)
- GNU Smalltalk (http://directory.fsf.org/smalltalk.html) - Smalltalk des GNU-Projekts.
- VisualWorks (http://www.cincom.com/smalltalk) - Smalltalk Entwicklungsumgebung von Cincom, als nicht-kommerzielle Version frei erhältlich.
- IBM VisualAge for Smalltalk (http://www-4.ibm.com/software/ad/smalltalk/) - Smalltalk-Umgebung von IBM
- Smalltalk/X (http://www.exept.de/sites/exept/deutsch/Smalltalk/frame_uebersicht.html) - Smalltalk der Firma *eXept Software*, die für sehr viele verschiedene Plattformen verfügbar ist.
- Dolphin Smalltalk (http://www.object-arts.com/) - Ein komfortables auf Windows spezialisierter Smalltalk-Dialekt
- Cincom Smalltalk (http://smalltalk.cincom.com/index.ssp) - Smalltalk-Dialekt der Firma Cincom (http://www.cincom.com/). Angeboten werden folgende Umgebungen:
  - ObjectStudio (http://smalltalk.cincom.com/prodinformation/index.ssp?content=osfactsheet) - Entwicklungsumgebung für Microsoft Windows mit COM- und ODBC-Schnittstellen.
  - VisualWorks (http://smalltalk.cincom.com/prodinformation/index.ssp?content=vwfactsheet) Für Client-Server, Server und Web-basierte Entwicklung, auf mehreren Plattformen verfügbar: Microsoft Windows, PowerMac, Linux sowie diversen Unix-Derivaten
- Smalltalk MT bei Object Connect (http://www.objectconnect.com/) oder Smalltalk MT bei Genify (http://www.genify.com/) – Ein C++-nahes Smalltalk-Derivat (C++ mit Smalltalk-Syntax), das zu nativen Code kompiliert.
- Ambrai Smalltalk, http://www.ambrai.com/

## Siehe auch

- Objective-C
- Self (Programmiersprache)
- Slate (Programmiersprache)
- Strongtalk

Von "http://de.wikipedia.org/wiki/Smalltalk_%28Programmiersprache%29"

Kategorie: Objektorientierte Programmiersprache

Korrigiere Fehler oder erweitere diesen Artikel!

- Diese Seite wurde zuletzt geändert um 12:54, 2. Apr 2006.
- Ihr Inhalt steht unter der GNU-Lizenz für freie Dokumentation
- Datenschutz
- Über Wikipedia
- Impressum