Kostenloses eBook

LERNEN

smalltalk

Free unaffiliated eBook created from
Stack Overflow contributors.

#smalltalk

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: smalltalk

It is an unofficial and free smalltalk ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official smalltalk.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Smalltalk

Bemerkungen

In diesem Abschnitt erhalten Sie einen Überblick darüber, was Smalltalk ist und warum ein Entwickler es verwenden möchte.

Es sollte auch alle großen Themen in Smalltalk erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für Smalltalk neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Examples

Installation oder Setup

Der Name Smalltalk bezieht sich normalerweise auf ANSI Smalltalk oder Smalltalk 80 (von denen der erste basiert). Während die meisten Implementierungen dem Standard nahekommen, unterscheiden sie sich in verschiedenen Aspekten (normalerweise als *Dialekte bezeichnet*).

Jede Implementierung verfügt über eine eigene Installationsmethode.

Bekannte FOSS-Implementierungen sind:

Pharo begann als Quietschgabel. (Windows / Linux / Mac OSX). Pharo hat einen eigenen Dokumentationseintrag bei Stackoverflow Documentation. Schauen Sie sich also dort um

Squeak (Windows / Linux / Mac OSX)

GNU Smalltalk (Windows / Linux / Mac OSX)

Dolphin Smalltalk Ursprünglich kommerziell, jetzt kostenlos Open Source. (Nur Windows)

Cuis Smalltalk Eine Squeak-Gabel, deren Schwerpunkt auf der Reduzierung der Systemkomplexität liegt.

Kommerzielle Smalltalks umfassen:

VisualWorks / Cincom Smalltalk Kostenlose Testversion verfügbar.

VisualAge Smalltalk Ursprünglich von IBM, jetzt Instatiations. Kostenlose Testversion verfügbar

Smalltalk / x (Kostenlos für den persönlichen Gebrauch?)

GemStone / s Kostenlose Community Edition verfügbar.

Andere Smalltalk-Dialekte

Amber Smalltalk Ein Smalltalk, der im Browser lebt.

Redline Smalltalk Smalltalk für die JVM.

Liste der Smalltalk-Implementierungen auf world.st

Hallo Welt in Smalltalk

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

Dies wird `Hello World!` drucken `Hello World!` zum Transcript-Fenster in Smalltalk. `Transcript` ist die Klasse, mit der Sie im Transcript-Fenster drucken können, indem Sie die Nachricht `show:` an dieses Objekt senden. Der Doppelpunkt zeigt an, dass für diese Nachricht ein Parameter erforderlich ist, in diesem Fall eine Zeichenfolge. Zeichenfolgen werden durch einfache Anführungszeichen und einfache Anführungszeichen dargestellt, da doppelte Anführungszeichen für Kommentare in Smalltalk reserviert sind.

Erste Schritte mit Smalltalk online lesen: https://riptutorial.com/de/smalltalk/topic/5316/erste-schritte-mit-smalltalk

Kapitel 2: Smalltalk-Syntax

Examples

Literale und Kommentare

Kommentare

```
"Comments are enclosed in double quotes. BEWARE: This is NOT a string!"
"They can span
 multiple lines."
```

Zeichenketten

```
'Strings are enclosed in sigle quotes.'
'Single quotes are escaped with a single quote, like this: ''.'
''''  "<--This string contains one single quote"

'Strings too can span
multiple lines'

''   "<--An empty string."
```

Symbole

```
#thiIsaSymbol  "Symbols are interned strings, used for method and variable names,
                and as values with fast equality checking."
#'hello world' "A symbol with a space in it"
#''            "An empty symbol, not very useful"
#+
#1             "Not the integer 1"
```

Zeichen

```
$a     "Characters are not strings. They are preceded by a $."
$A     "An uppercase character"
$      "The spacecharacter!"
$→     "An unicode character"
$1     "Not to be confused with the number 1"
```

Zahlen

Zahlen gibt es in allen Varianten:

Ganze Zahlen:

- Dezimal

  10

  -1

  0

  1000000000000000000000000000000000000000000

- Hexadezimal

  16rAB1F

  16r0

  -16rFFFFFFFFFFFFFFFFFFFFFFFFFFFF

- ScaledDecimal

  17s0

  3.14159265s8

- Andere

  8r7731 "oktal"

  2r1001 "binär"

  10r99987 "wieder dezimal!"

Fließpunkt

```
3.14 1.2e3     "2 floating-point numbers"
```

Bruchteile

Brüche sind nicht identisch mit Fließkommazahlen, sie sind exakte Zahlen (kein Rundungsfehler).

```
4/3            "The fraction 4/3"
355/113        "A rational approximation to pi"
```

Arrays

```
#(#abc 123)    "A literal array with the symbol #abc and the number 123"
```

Byte-Arrays

```
#[1 2 3 4]       "separators are blank"
#[]              "empty ByteArray"
#[0 0 0 0 255]   "length is arbitrary"
```

Dynamische Arrays

Dynamische Arrays werden aus Ausdrücken erstellt. Jeder Ausdruck innerhalb der geschweiften Klammern wird im konstruierten Array zu einem anderen Wert ausgewertet.

```
{self foo. 3 + 2. i * 3}   "A dynamic array built from 3 expressions"
```

Blöcke

```
[ :p | p asString ] "A code block with a parameter p.
                     Blocks are the same as lambdas in other languages"
```

Einige Notizen:

Beachten Sie, dass Literal-Arrays eine beliebige Art und Anzahl von Leerzeichen als Trennzeichen verwenden

```
#(256 16rAB1F 3.14s2 2r1001 $A #this)
"is the same as:"
#(256
  16rAB1F
  3.14s2
  2r1001
  $A #this)
```

Beachten Sie auch, dass Sie Literale zusammenstellen können

```
#[255 16rFF 8r377 2r11111111]       (four times 255)

#(#[1 2 3] #('string' #symbol))     (arrays of arrays)
```

Es gibt eine gewisse "Toleranz" gegenüber entspannter Notation

```
#(symbol) = #(#symbol)              (missing # => symbol)

#('string' ($a 'a'))                (missing # => array)
```

Aber nicht hier:

```
#([1 2 3]) ~= #(#[1 2 3])           (missing # => misinterpreted)
```

jedoch

```
#(true nil false)                   (pseudo variables ok)
```

Aber nicht hier!

```
#(self) = #(#self)                  (missing # => symbol)
```

Wie Sie sehen können, gibt es einige Inkonsistenzen:

- Während die Pseudovariablen `true` , `false` und `nil` als Literale innerhalb von Arrays akzeptiert werden, werden die Pseudovariablen `self` und `super` als **Symbole** interpretiert (unter Verwendung der allgemeineren Regel für nicht qualifizierte Strings).
- Es ist zwar nicht zwingend erforderlich, `#(` zu schreiben `#(` um ein verschachteltes Array in einem Array zu starten und die Klammer genügt, ist es jedoch erforderlich, `#[` für das Starten eines verschachtelten `ByteArray` zu schreiben.

Einige davon wurden entnommen aus:

http://stackoverflow.com/a/37823203/4309858

Nachricht senden

In Smalltalk *senden* Sie fast ausschließlich *Nachrichten* an Objekte (in anderen Sprachen als Aufrufmethoden bezeichnet). Es gibt drei Arten von Nachrichten:

Unäre Nachrichten:

```
#(1 2 3) size
"This sends the #size message to the #(1 2 3) array.
 #size is a unary message, because it takes no arguments."
```

Binäre Nachrichten:

```
1 + 2
"This sends the #+ message and 2 as an argument to the object 1.
 #+ is a binary message because it takes one argument (2)
 and it's composed of one or two symbol characters"
```

Keyword-Nachrichten:

```
'Smalltalk' allButFirst: 5.
"This sends #allButFirst: with argument 5 to the string 'Smalltalk',
 resulting in the new string 'talk'"
```

```
3 to: 10 by: 2.
"This one sends the single message #to:by:, which takes two parameters (10 and 2)
 to the number 3.
 The result is a collection with 3, 5, 7, and 9."
```

Mehrere Nachrichten in einer Anweisung werden nach der Rangfolge ausgewertet

```
unary > binary > keyword
```

und von links nach rechts.

```
1 + 2 * 3 " equals 9, because it evaluates left to right"
1 + (2 * 3) " but you can use parenthesis"

1 to:   #(a b c d) size   by:   5 - 4
 "is the same as:"
1 to: ( #(a b c d) size ) by: ( 5 - 4 )
```

Wenn Sie viele Nachrichten an dasselbe Objekt senden möchten, können Sie den Kaskadierungsoperator verwenden ; (Semikolon):

```
OrderedCollection new
  add: #abc;
  add: #def;
  add: #ghi;
  yourself.
```

"Dies sendet zuerst die Nachricht #new an die Klasse OrderedCollection (#new ist nur eine Nachricht und kein Operator). Sie führt zu einer neuen OrderedCollection. Dann sendet sie der neuen Collection dreimal die Nachricht #add (mit verschiedenen Argumenten). und die Nachricht selbst. "

Klassen und Methoden

Klassen und Methoden werden normalerweise in der Smalltalk-IDE definiert.

Klassen

Eine Klassendefinition sieht im Browser etwa so aus:

```
XMLTokenizer subclass: #XMLParser
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'XML-Parser'
```

Dies ist die Nachricht, die der Browser an Sie sendet, um eine neue Klasse im System zu erstellen. (In diesem Fall ist dies

`#subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:` aber es gibt andere, die auch neue Klassen `#subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:`).

Die erste Zeile zeigt an, welche Klasse Sie subclassing (in diesem Fall XMLTokenizer) und den Namen, den die neue Subklasse haben wird (#XMLParser).

In den nächsten drei Zeilen werden die Variablen definiert, über die die Klasse und ihre Instanzen verfügen.

Methoden

Methoden sehen im Browser so aus:

```
aKeywordMethodWith: firstArgument and: secondArgument
    "Do something with an argument and return the result."

    ^firstArgument doSomethingWith: secondArgument
```

Das `^` (Caret) ist der Rückkehroperator.

```
** anInteger
    "Raise me to anInteger"
    | temp1 temp2 |

    temp1 := 1.
    temp2 := 1.
    1 to: anInteger do: [ :i | temp1 := temp1 * self + temp2 - i ].
    ^temp1
```

Dies ist nicht der richtige Weg zur Exponentiation, aber es zeigt eine *binäre Nachrichtendefinition* (sie ist wie jede andere Nachricht definiert) und einige *temporäre Methodenvariablen* (oder temporäre *Methoden der* Methoden, *Temp1* und *Temp2*) sowie ein *Blockargument* (*i*).

Schleifen in Smalltalk

In diesem Beispiel wird eine `Ordered Collection` `OrderedCollection` verwendet, um die verschiedenen Nachrichten `OrderedCollection` , die an ein `OrderedCollection` Objekt `OrderedCollection` werden können, um die Elemente zu `OrderedCollection` .
Der folgende Code instanziiert eine leere `OrderedCollection` mit der Nachricht `new` und `OrderedCollection` sie mit 4 Zahlen mit der Nachricht `add:`

```
anOrderedCollection := OrderedCollection new.
anOrderedCollection add: 1; add: 2; add: 3; add: 4.
```

Für alle diese Meldungen wird ein Block als Parameter verwendet, der für jedes der Elemente in der Sammlung ausgewertet wird.

1. `do:`

Dies ist die grundlegende Aufzählungsnachricht. Wenn wir beispielsweise jedes Element in der Sammlung drucken möchten, können wir dies als solches erreichen:

```
anOrderedCollection do:[:each | Transcript show: each]. "Prints --> 1234"
```

Jedes der Elemente in der Sammlung wird nach den Wünschen des Benutzers mit der folgenden Syntax definiert:: `:each do:` loop-Anweisung druckt jedes Element in der Sammlung in das `Transcript` Fenster.

2. `collect:`
   Mit der Meldung `collect:` können Sie für jedes Element in der Sammlung etwas tun und das Ergebnis Ihrer Aktion in einer neuen Sammlung speichern
   Wenn wir beispielsweise jedes Element in unserer Sammlung mit 2 multiplizieren und einer neuen Sammlung hinzufügen möchten, können Sie die `collect:` -Meldung als solche verwenden:

```
evenCollection := anOrderedCollection collect:[:each | each*2]. "#(2 4 6 8)"
```

3. `select:`
   Mit der Nachricht `select:` können Sie eine Unterauflistung erstellen, in der Elemente aus der Originalsammlung basierend auf bestimmten Bedingungen ausgewählt werden. Wenn Sie beispielsweise eine neue Sammlung ungerader Zahlen aus unserer Sammlung erstellen möchten, können Sie die `select:` -Meldung als solche verwenden:

```
oddCollection := anOrderedCollection select:[:each | each odd].
```

   Da `each odd` Zahl einen Booleschen `oddCollection` zurückgibt, werden nur die Elemente zu `oddCollection` hinzugefügt, die `#(1 3)` .

4. `reject:`
   Diese Nachricht funktioniert im Gegensatz zu `select:` und lehnt alle Elemente ab, die dazu führen, dass der Boolesche Wert `true` zurückgibt. Oder mit anderen Worten, es werden alle Elemente ausgewählt, die dazu führen, dass der Boolesche Wert `false` . Zum Beispiel, wenn wir die gleiche `oddCollection` wie das vorige Beispiel `oddCollection` wollten. Wir können `reject:` als solche verwenden:

```
oddCollection := anOrderedCollection reject:[:each | each even].
```

   `oddCollection` wieder `#(1 3)` als Elemente.

Dies sind die vier grundlegenden Aufzählungstechniken in Smalltalk. Sie können jedoch in der `Collections` Klasse nach weiteren Nachrichten suchen, die möglicherweise implementiert werden.

Smalltalk-Syntax online lesen: https://riptutorial.com/de/smalltalk/topic/5422/smalltalk-syntax

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Smalltalk	Bananeweizen, Community, fede s., Leandro Caniglia, Stephan Eggermont, ybce
2	Smalltalk-Syntax	aka.nice, fede s., ybce