11. A bit of Smalltalk

Oscar Nierstrasz
Roadmap

- The origins of Smalltalk
- What is Smalltalk?
- Syntax in a nutshell
- Seaside — web development with Smalltalk
Roadmap

> The origins of Smalltalk
> What is Smalltalk?
> Syntax in a nutshell
> Seaside — web development with Smalltalk
The origins of Smalltalk

Alan Kay’s Dynabook project (1968)

Alto — Xerox PARC (1973)

gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html
# Smalltalk vs. Java vs. C++

<table>
<thead>
<tr>
<th></th>
<th>Smalltalk</th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object model</strong></td>
<td>Pure</td>
<td>Hybrid</td>
<td>Hybrid</td>
</tr>
<tr>
<td><strong>Garbage collection</strong></td>
<td>Automatic</td>
<td>Automatic</td>
<td>Manual</td>
</tr>
<tr>
<td><strong>Inheritance</strong></td>
<td>Single</td>
<td>Single</td>
<td>Multiple</td>
</tr>
<tr>
<td><strong>Types</strong></td>
<td>Dynamic</td>
<td>Static</td>
<td>Static</td>
</tr>
<tr>
<td><strong>Reflection</strong></td>
<td>Fully reflective</td>
<td>Introspection</td>
<td>Introspection</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>Semaphores</td>
<td>Monitors</td>
<td>Some libraries</td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>Categories, namespaces</td>
<td>Packages</td>
<td>Namespaces</td>
</tr>
</tbody>
</table>
Smalltalk-80 and Pharo

• Everything is an object
• Everything is there, all the time
• First windowing system with mouse
• First graphical IDE
What are Squeak and Pharo?

> Squeak is a modern, open-source, highly portable, fast, full-featured Smalltalk implementation
  —Based on original Smalltalk-80 code

> Pharo is a lean and clean fork of Squeak
  —www.pharo-project.org
Pharo by Example

http://pharobyexample.org/

- Free download
- Open-Source
- Print-on-demand
Don’t panic!

New Smalltalkers often think they need to understand all the details of a thing before they can use it.

Try to answer the question with

“How does this work?”

with

“I don’t care”.

Alan Knight. Smalltalk Guru
Roadmap

- The origins of Smalltalk
- What is Smalltalk?
- Syntax in a nutshell
- Seaside — web development with Smalltalk
Two rules to remember
Everything is an object
Everything happens by sending messages
What is Smalltalk?

Image

Virtual machine

Changes

Sources
Demo: Running Pharo

Welcome to Pharo, a clean, innovative, free, and open-source Smalltalk environment. This is version 1.0, the first production release of Pharo. For more information about this release see: www.pharo-project.org/pharo-download/release-1.0

You can find information about Pharo on www.pharo-project.org. In particular, you may be interested in:
- How to join us and get help: www.pharo-project.org/community
- Getting the Pharo By Example book (also available as a free PDF): www.pharobyexample.org/
- Watching the Screencasts: http://pharocasts.blogspot.com/
- Reporting problems: www.pharo-project.org/community/issue-tracking

If you are new to Smalltalk, we recommend you to do the integrated, interactive tutorial. To start it, evaluate the following expression (select the text -> right button -> do it):

ProfStef go

If you are interested in installing and using external tools and projects, such as Seaside, Magritte, Pier, AidaWeb, Magma, Moose, O2, etc., evaluate:

DEVImageWorkspaces openExternalProjectWorkspace
Mouse Semantics
Standard development tools
Debuggers, Inspectors, Explorers
Do it, Print it, …

You can evaluate any expression anywhere in Smalltalk.
Roadmap

> The origins of Smalltalk
> What is Smalltalk?
> **Syntax in a nutshell**
> Seaside — web development with Smalltalk
Three kinds of messages

> Unary messages

5 factorial
Transcript cr

> Binary messages

3 + 4

> Keyword messages

3 raisedTo: 10 modulo: 5
Transcript show: 'hello world'
First unary, then binary, then keyword:

2 raisedTo: 1 + 3 factorial

Same as:

2 raisedTo: (1 + (3 factorial))

Use parentheses to force order:

1 + 2 * 3
1 + (2 * 3)
Precedence

First unary, then binary, then keyword:

\[ 2 \text{ raisedTo: } 1 + 3 \text{ factorial} \]

Same as:

\[ 2 \text{ raisedTo: } (1 + (3 \text{ factorial})) \]

Use parentheses to force order:

\[ 1 + 2 \times 3 \]
\[ 1 + (2 \times 3) \]
Precedence

First unary, then binary, then keyword:

2 raisedTo: 1 + 3 factorial

Same as: 2 raisedTo: (1 + (3 factorial))

Use parentheses to force order:

1 + 2 * 3
1 + (2 * 3)

9 (!)
First unary, then binary, then keyword:

\[
\begin{align*}
2 \text{ raisedTo: } & 1 + 3 \text{ factorial} & \rightarrow 128 \\
\text{Same as: } & 2 \text{ raisedTo: } (1 + (3 \text{ factorial}))
\end{align*}
\]

Use parentheses to force order:

\[
\begin{align*}
1 + 2 \times 3 & \rightarrow 9 \\
1 + (2 \times 3) & \rightarrow 7
\end{align*}
\]
A typical method in the class Point

\[
\leq \text{aPoint}
\]

"Answer whether the receiver is neither below nor to the right of aPoint."

\[
^x \leq \text{aPoint } x \text{ and: } [y \leq \text{aPoint } y]
\]
A typical method in the class Point

\[
\leq \text{aPoint}
\]

"Answer whether the receiver is neither below nor to the right of aPoint."

\[
^x \leq \text{aPoint} \ x \ \text{and:} \ [y \leq \text{aPoint} \ y]
\]
A typical method in the class Point

\[ \leq \text{aPoint} \]

"Answer whether the receiver is neither below nor to the right of aPoint."

\[ ^\wedge x \leq \text{aPoint } x \text{ and: } [y \leq \text{aPoint } y] \]
A typical method in the class Point

\[ \leq \text{aPoint} \]

"Answer whether the receiver is neither below nor to the right of \text{aPoint}."

^ \ x \leq \text{aPoint} \ x \ \text{and:} \ [y \leq \text{aPoint} \ y]
A typical method in the class `Point`:

```plaintext
<= aPoint
   "Answer whether the receiver is neither below nor to the right of aPoint."
   \[ x \leq aPoint \cdot x \text{ and: } [y \leq aPoint \cdot y]\n```
A typical method in the class Point

```
<= aPoint
   "Answer whether the receiver is neither below nor to the right of aPoint."

^ x <= aPoint x and: [y <= aPoint y]
```
A typical method in the class Point

\[ \leq \text{aPoint} \]

"Answer whether the receiver is neither below nor to the right of \text{aPoint}."

\[ \wedge x \leq \text{aPoint} \times \text{and: [y } \leq \text{aPoint} \times \text{y] \]
A typical method in the class Point

\[
\leq \text{aPoint}
\]

"Answer whether the receiver is neither below nor to the right of \text{aPoint}."

\[
\wedge x \leq \text{aPoint} \ x \ \text{and:} \ [y \leq \text{aPoint} \ y]
\]
A typical method in the class Point

\[ \leq \ aPoint \]

"Answer whether the receiver is neither below nor to the right of aPoint."

\[ \wedge \ x \leq \ aPoint \ x \ \text{and:} \ [y \leq \ aPoint \ y] \]
A typical method in the class `Point`

>`<= aPoint`

"Answer whether the receiver is neither below nor to the right of aPoint."

`^ x <= aPoint x and: [y <= aPoint y]`

`(2@3) <= (5@6)`
A typical method in the class Point

Method name: <= aPoint

"Answer whether the receiver is neither below nor to the right of aPoint."

^ x <= aPoint x and: [y <= aPoint y]

Return

Binary message

Instance variable

Block

Keyword message

(2@3) <= (5@6) true
Statements and cascades

| p pen |
p := 100@100.
pen := Pen new.
pen up.
pen goto: p; down; goto: p+p
Statements and cascades

| p pen |
p := 100@100.
pen := Pen new.
pen up.
pen goto: p; down; goto: p+p
Statements and cascades

| p pen |
p := 100@100.
pen := Pen new.
pen up.
pen goto: p; down; goto: p+p
Statements and cascades

Temporary variables

Assignment

Statement

<table>
<thead>
<tr>
<th>p pen</th>
</tr>
</thead>
<tbody>
<tr>
<td>p := 100@100.</td>
</tr>
<tr>
<td>pen := Pen new.</td>
</tr>
<tr>
<td>pen up.</td>
</tr>
<tr>
<td>pen goto: p; down; goto: p+p</td>
</tr>
</tbody>
</table>
Statements and cascades

| p pen |
p := 100@100.
pen := Pen new.
pen up.
pen goto: p; down; goto: p+p
## Literals and constants

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strings &amp; Characters</td>
<td>'hello' $a</td>
</tr>
<tr>
<td>Numbers</td>
<td>1 3.14159</td>
</tr>
<tr>
<td>Symbols</td>
<td>#yadayada</td>
</tr>
<tr>
<td>Arrays</td>
<td>#(1 2 3)</td>
</tr>
<tr>
<td>Pseudo-variables</td>
<td>self super</td>
</tr>
<tr>
<td>Constants</td>
<td>true false</td>
</tr>
</tbody>
</table>
> **Local variables** are delimited by `| var |`

**Block variables** by `: var |`

```
OrderedCollection>>collect: aBlock
    "Evaluate aBlock with each of my elements as the argument."
    | newCollection |
firstIndex to: lastIndex do:
    [ :index |
        newCollection addLast: (aBlock value: (array at: index))].
^ newCollection
```
> **Local variables** are delimited by `| var |`

**Block variables** by `:var |`

``` Smalltalk
OrderedCollection>>collect: aBlock
    "Evaluate aBlock with each of my elements as the argument."
    | newCollection |
    firstIndex to: lastIndex do:
        [ :index |
            newCollection addLast: (aBlock value: (array at: index))].

^ newCollection
```

```
(OrderedCollection with: 10 with: 5) collect: [:each| each factorial ]
```
Variables

> **Local variables** are delimited by `| var |`

**Block variables** by `: var |

```
OrderedCollection>>collect: aBlock
    "Evaluate aBlock with each of my elements as the argument."
    | newCollection |
    firstIndex to: lastIndex do:
        [ :index |
            newCollection addLast: (aBlock value: (array at: index))].
^ newCollection
```

```
(OrderedCollection with: 10 with: 5) collect: [:each| each factorial ]
```

```
an OrderedCollection(3628800 120)
```
Control Structures

> Every control structure is realized by message sends

```
max: aNumber
  ^ self < aNumber
  ifTrue: [aNumber]
  ifFalse: [self]
```

```
4 timesRepeat: [Beeper beep]
```
Creating objects

> **Class methods**

OrderedCollection new
Array with: 1 with: 2

> **Factory methods**

1@2  a Point
1/2  a Fraction
Creating classes

> Send a message to a class (!)

```
Number subclass: #Complex
  instanceVariableNames: 'real imaginary'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'ComplexNumbers'
```
Demo: Defining classes and methods
Roadmap

> The origins of Smalltalk
> What is Smalltalk?
> Syntax in a nutshell
> Seaside — web development with Smalltalk
Seaside — a Smalltalk web development platform
Demo: PostOffice in Seaside
What you should know!

- What are the key differences between Smalltalk, C++ and Java?
- What is at the root of the Smalltalk class hierarchy?
- What kinds of messages can one send to objects?
- What is a cascade?
- Why does $1 + 2/3 = 1$ in Smalltalk?
- How are control structures realized?
- How is a new class created?
- What are categories for?
- What are Factory methods? When are they useful?
Can you answer these questions?

- Which is faster, a program written in Smalltalk, C++ or Java?
- Which is faster to develop & debug, a program written in Smalltalk, C++ or Java?
- How are Booleans implemented?
- Is a comment an Object? How would you check this?
- What is the equivalent of a static method in Smalltalk?
- How do you make methods private in Smalltalk?
- What is the difference between = and ==?
- If classes are objects too, what classes are they instances of?
Attribution-ShareAlike 3.0

You are free:
- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

http://creativecommons.org/licenses/by-sa/3.0/