Principles of Object-Oriented Languages

Smalltalk Introduction (2)

Kris Gybels
kris.gybels@vub.ac.be

(slides version: october 11th 2007)

http://proq.vub.ac.be/~kgybels/Teaching/POOL/
Methods

Method header
Each keyword of the message selector followed by name of argument variable

Method body
Sequence of messages and assignments separated with dots
Methods

For methods for binary and unary messages:

+ point
  x := x + point x.
  y := y + point y.

increment
  x := x + 1.
  y := y + 1.
Method Return Values

Default return is object itself

```
doNothing
```

```
da LazyObject
```

```
= 
```

Return expressions for returning something else

```
giveMeFive
```

```
^ 5
```

```
da LazyObject
```

```
x
```

```
= 
```

```
a Number
```

```
5
```

```
da LazyObject
```

```
x
```
message selector and argument names

"comment stating purpose of message"

I temporary variable names I
statements
Define methods on the Counter class so that Counter instances can respond to these messages:

- initialize
- increment
- incrementBy: a Number
- decrementBy: a Number
- current
Hands-On

Define a class method so that:

```
Counter newCounter =

newCounter = Counter
  count

= a Counter:
  count

= a Number:
  0
```
Construction & Initialization Pattern

newCounter \rightarrow Counter

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
</table>

newCounter ^ Counter new initialize

initialize
count := 0
Construction&Initialization Pattern

newCounter

^ Counter new initialize

Counter

count

initialize

count := 0
Construction & Initialization Pattern

newCounter

Counter

^ Counter new initialize

count := 0

initialize
count := 0
Construction&Initialization Pattern

newCounter

Counter

count

initialize
count := 0

new

a Counter

count

an UndefinedObject

initialize
count := 0
Construction & Initialization Pattern

newCounter

Counter

^ Counter new initialize

count := 0

initialize

a Counter

count

initialize

count := 0

an UndefinedObject
Construction & Initialization Pattern

```
newCounter
^ Counter new initialize
```

```
a Counter

count

an UndefinedObject

initialize
count := 0
```
Construction & Initialization Pattern

newCounter
^ Counter new initialize

a Counter
count

initialize
count := 0

a Number
0
Construction & Initialization Pattern

```
Counter
  count
```

```
newCounter
  ^ Counter new initialize
```

```
a Counter
  count
```

```
a Number
  0
```

```
initialize
  count := 0
```

returns "self"
Construction & Initialization Pattern

newCounter
\textsuperscript{Counter} new initialize

initialize
\texttt{count := 0}

Returns the now initialized Counter instance
returns “self”

a Counter
\texttt{count}

a Number
\texttt{0}
Special Variables

- **nil** = an UndefinedObject
- **true** = a True
- **false** = a False
“self” always refers to the object for which method is evaluated

“super” = “self”
BUT has special influence on method lookup semantics
=> cfr. lectures
“self” always refers to the object for which method is evaluated

```
a SomeObject
   x
```

```
aTest
   self anotherTest
```

“super” = “self”

BUT has special influence on method lookup semantics

=> cfr. lectures
“self” always refers to the object for which method is evaluated

```
SomeObject
```

```
self anotherTest
```

“super” = “self”

BUT has special influence on method lookup semantics  
=> cfr. lectures
Special Variables

“self” always refers to the object for which method is evaluated

"super" = "self"

BUT has special influence on method lookup semantics

=> cfr. lectures
Some Notes on Variables

Global Variables: first letter capitalized

- Global
- ASharedValue
- Point

Classes are stored in global variables!

Point := nil

Instance variables, method arguments, ... first letter not capitalized

- aNumber
- computedValue
- sum
Global Variables: first letter capitalized

Global  ASharedValue  Point

Classes are stored in global variables!

Point := nil

Instance variables, method arguments, ... first letter not capitalized

aNumber  computedValue  sum
Temporary variables used only in a method are declared before the method body between | |
**Block Expressions & Closures**

Blocks are Smalltalk’s “lambda’s”

```
[ :foo :bar | foo + bar ]
```

= 

a BlockClosure

---

Blocks can be sent messages to evaluate the body of the block

```
value: value:
```

= 

a BlockClosure

```
[ :foo :bar | foo + bar ] value: 4 value: 2
```

= 

a Number

6
Block Expressions & Closures

Blocks are Smalltalk’s “lambda’s”

\[ [ \text{:foo :bar | foo + bar} ] \]

= 

a BlockClosure

Blocks can be sent messages to evaluate the body of the block

\[ \text{value: :value:} \]

\[ [ \text{:foo :bar | foo + bar} ] \text{ value: 4 value: 2} \]

= 

a Number

6
Blocks are Smalltalk’s “lambda’s”

\[ [:foo :bar | foo + bar ] \]

= 

a BlockClosure

Blocks can be sent messages to evaluate the body of the block

\[ [:foo :bar | foo + bar ] \text{ value: } 4 \text{ value: } 2 \]

= 

a Number

6
Blocks

[ Transcript show: ‘Boo!’ ] value

ablock := [ :someName | someName asUppercase ].
ablock value: ‘kris’

[ :a :b | a + b ] valueWithArguments: (Array with: 1 with: 2)
Blocks

Structure similar to that of methods

Argument List

Temporaries

Block Body

[ :argA :argB |
  sum difference |
  sum := argA + argB.
  difference := argA - argB.
  Array with: sum with: difference.
]

Return value of ‘value’, ‘value:’ etc.
messages is always return value of
last expression in block
Extend the Counter class with a method `doWithCurrentValue:` which takes a block as argument and executes the block with the current value of the counter

Using the `doWithCurrentValue:` message, show the current counter value on the Transcript
Blocks & Booleans

ifTrue: ifFalse: a True

a BlockClosure a BlockClosure
Blocks & Booleans

ifTrue: ifFalse: → a True

a BlockClosure

a BlockClosure

value
Blocks & Booleans

ifTrue: ifFalse:  a False

a BlockClosure  a BlockClosure
Blocks & Booleans

```
ifTrue: ifFalse:
```

- a BlockClosure
- a BlockClosure
- value
- a False
true ifTrue: [ Transcript show: 'waar' ] ifFalse: [ Transcript show: 'vals' ]

(a < b) ifTrue: [ Transcript show: 'a kleiner' ]
ifFalse: [ Transcript show: 'a niet kleiner' ]

Transcript show: ((a < b) ifTrue: [ 'a' ] ifFalse: [ 'a niet' ]) , ' kleiner'
[ Date today weekday = #Wednesday ]
whileFalse:
[ Transcript show: ‘Today is not a Wednesday yet’ ]

Note: Press Ctrl-Y to break an evaluation