Smalltalk in a Nutshell

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.iam.unibe.ch/~ducasse/

Goals

- Syntax in a Nutshell
- OO Model in a Nutshell

Smalltalk OO Model

- ***Everything*** is an object
  - ⇒ Only message passing
  - ⇒ Only late binding
- Instance variables are private to the object
- Methods are public
- Everything is a pointer

- Garbage collector
- Single inheritance between classes
- Only message passing between objects

Complete Syntax on a PostCard

```
exampleWithNumber: x
“A method that illustrates every part of Smalltalk method syntax except primitives. It
has unary, binary, and key word messages, declares arguments and temporaries (but
not block temporaries),  accesses a global variable (but not and instance variable), uses
literals (array, character, symbol, string, integer, float), uses the pseudo variable true
false, nil, self, and super, and has sequence, assignment, return and cascade. It has both
zero argument and one argument blocks. It doesn’t do anything useful, though”
    |y|
    true & false not & (nil isNil) ifFalse: [self halt].
    y := self size + super size.
    #($a #a ‘a’ 1 1.0)
                do: [:each | Transcript
                        show: (each class name);
                        show: (each printString);
                                                        show:‘ ‘].
        ^ x < y
```

Language Constructs

^	return
“	comments
#	symbol or array
‘	string
[ ]	block or byte array
.	separator and not terminator (or namespace access in VW)
;	cascade (sending several messages to the same instance)
|	local or block variable
:=	assignment
$	character
:	end of selector name
e, r	number exponent or radix
!	file element separator
<primitive: ...>	for VM primitive calls

Syntax

comment: “a comment”
character: $c $h $a $r $a $c $t $e $r $s $# $@
string: ‘a nice string’ ‘lulu’ ‘l’’idiot’
symbol: #mac #+
array: #(1 2 3 (1 3) $a 4)
byte array: #[1 2 3]
integer: 1, 2r101
real: 1.5, 6.03e-34,4, 2.4e7
float: 1/33
boolean: true, false
point: 10@120

Note that @ is not an element of the syntax, but just a message sent to a number. This is the same for /, bitShift, ifTrue:, do: ...

Syntax in a Nutshell (II)

assigment: var := aValue
block: [:var ||tmp| expr...]

temporary variable: |tmp|
block variable: :var
unary message: receiver selector
binary message: receiver selector argument
keyword based: receiver keyword1: arg1 keyword2: arg2...
cascade: message ; selector ...
separator: message . message
result: ^
parenthesis: (...)

Messages vs. a predefined Syntax

- In Java, C, C++, Ada constructs like >>, if, for, etc. are hardcoded into the grammar
- In Smalltalk there are just messages defined on objects
- (>>) bitShift: is just a message sent to numbers
  - 10 bitShift: 2
- (if) ifTrue: is just messages sent to a boolean
  - (1> x) ifTrue:
- (for) do:, to:do: are just messages to collections or numbers
  - #(a b c d) do: [:each | Transcript show: each ; cr]
  - 1 to: 10 do: [:i | Transcript show: each printString; cr]
- Minimal parsing
- Language is extensible

Class Definition Revisited VW

- Class Definition: A message sent to a namespace

Smalltalk **defineClass**: #NameOfClass
superclass: #{NameOfSuperclass}
indexedType: #none
private: false
instanceVariableNames: ''
classInstanceVariableNames: ''
imports: ''
category: 'Browser-Commands'

Class Definition Revisited Squeak

```
NameOfSuperclass subclass: #NameOfClass
    instanceVariableNames: 'instVarName1'
    classVariableNames: 'classVarName1'
    poolDictionaries: ''
    category: 'LAN'
```

Method Definition Revisited

- Normally defined in a browser or (by directly invoking the compiler)
- Methods are **public**
- **Always return self**

```
Node>>accept: thePacket
    "If the packet is addressed to me, print it.
     Else just behave like a normal node"

     (thePacket isAddressedTo: self)
             ifTrue: [self print: thePacket]
             ifFalse: [super accept: thePacket]
```

Instance Creation

- 1, 'abc'
- Basic class creation messages are
  - new, new:,
  - basicNew, basicNew:
  - Monster new
- Class specific message creation (messages sent to classes)
  - Tomagoshi withHunger: 10

Messages and their Composition

- Three kinds of messages
  - **Unary**: Node new
  - **Binary**: 1 + 2, 3@4
  - **Keywords**: aTomagoshi eat: #cooky furiously: true
- **Message Priority**
  - **(Msg) > unary > binary > keywords**
  - Same Level from left to right
- Example:
  - (10@0 extent: 10@100) bottomRight
  - s isNil ***ifTrue:*** [self halt]

Blocks

- Anonymous method
- Passed as method argument or stored
- Functions
  - fct(x)= x*x+3, fct(2).
  - fct :=[:x| x * x + 3]. fct value: 2

```
Integer>>factorial
    | tmp |
    tmp:= 1.
    2 to: self do: [:i| tmp := tmp * i].
    ^ tmp

#(1 2 3) do: [:each | Transcript show: each printString ; cr]
```

Summary

Objects and Messages
Three kinds of messages
  unary
  binary
  keywords
Block: a.k.a innerclass or closures or lambda
Unary>Binary>Keywords

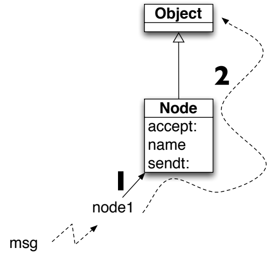
Goals

- Syntax in a Nutshell
- ***OO Model in a Nutshell***

Instance and Class

- Only one model
- Uniformly applied
- Classes are objects too

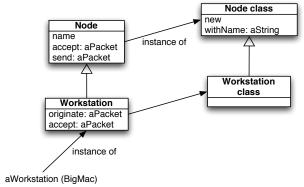
Lookup…Class + Inheritance



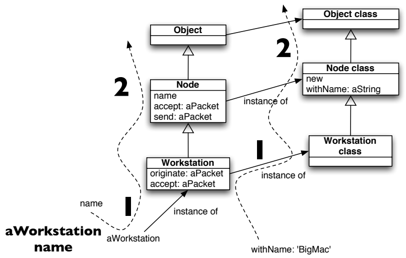
Classes are objects too

- Instance creation is just a message send to a ... Class
- Same method lookup than with any other objects
- a Class is the single instance of amanonymous class
  - Point is the single instance of Point class

Class Parallel Inheritance



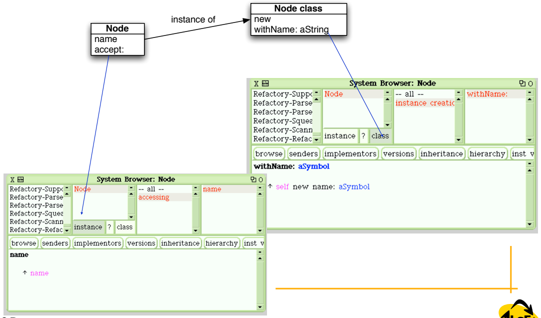
Lookup and Class Methods



aWorkstation name

Workstation withName: ‘BigMac’

About the Buttons



```
withName: aSymbol

  ↑ self new name: aSymbol
```

```
name

  ↑ name
```

Summary

- Everything is an object
- One single model
- Single inheritance
- Public methods
- Private attribute
- Classes are simply objects too
- Class is instance of another class
- One unique method lookup
  look in the class of the receiver