

# Smalltalk in a Nutshell

Stéphane Ducasse  
stephane.ducasse@inria.fr  
<http://stephane.ducasse.free.fr/>

# 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

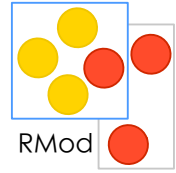
Garbage collector

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' | 1.0)

do: [:each | Transcript

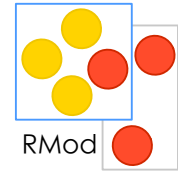
show: (each class name);

show: (each printString);

show: ' '].

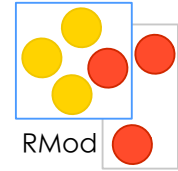
^ x < y

# 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

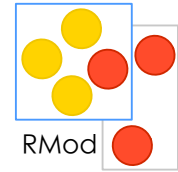
# Syntax



comment:	“a comment”
character:	<code>\$c \$h \$a \$r \$a \$c \$t \$e \$r \$s \$# \$@</code>
string:	<code>'a nice string' 'lulu' 'l'idiot'</code>
symbol:	<code>#mac #+</code>
array:	<code>#{1 2 3 (1 3) \$a 4}</code>
byte array:	<code>#[1 2 3]</code>
integer:	<code>l, 2r10l</code>
real:	<code>l.5, 6.03e-34,4, 2.4e7</code>
float:	<code>l/33</code>
boolean:	<code>true, false</code>
point:	<code>l0@l20</code>

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)



assignment: `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...

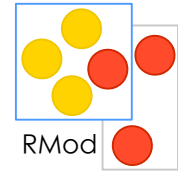
cascade: message ; selector ...

separator: message . message

result: `^`

parenthesis: `(...)`

# Class Definition in St-80



```
NameOfSuperclass subclass: #NameOfClass  
instanceVariableNames: 'instVarName I'  
classVariableNames: 'classVarName I'  
poolDictionaries: "  
category: 'LAN'
```



# Method Definition



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

*Node*>>accept: thePacket

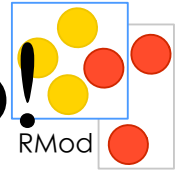
"If the packet is addressed to me, print it.  
Else just behave like a normal node"

(thePacket isAddressedTo: self)

ifTrue: [self print: thePacket]

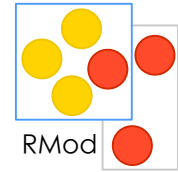
ifFalse: [super accept: thePacket]

# Instance Creation: Messages Too!



- 'I', '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 |
```

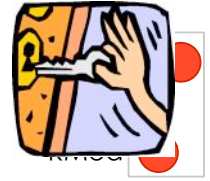
```
tmp:= 1.
```

```
2 to: self do: [:i| tmp := tmp * i]
```

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

# Yes ifTrue: is sent to a boolean

---



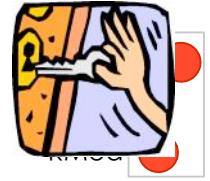
Weather isRaining

**ifTrue:** [self takeMyUmbrella]

**ifFalse:** [self takeMySunglasses]

ifTrue:ifFalse is sent to an object: a boolean!

# Yes a collection is iterating on itself



```
 #(1 2 -4 -86)  
  do: [:each | Transcript show: each abs  
  printString ;cr ]
```

```
> 1  
> 2  
> 4  
> 86
```

***Yes we ask the collection object to***

# Summary



Objects and Messages

Three kinds of messages

unary

binary

keywords

Block: a.k.a innerclass or closures or lambda

Unary>Binary>Keywords

# Goals

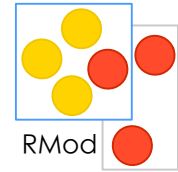
Syntax in a Nutshell

**OO Model in a Nutshell**



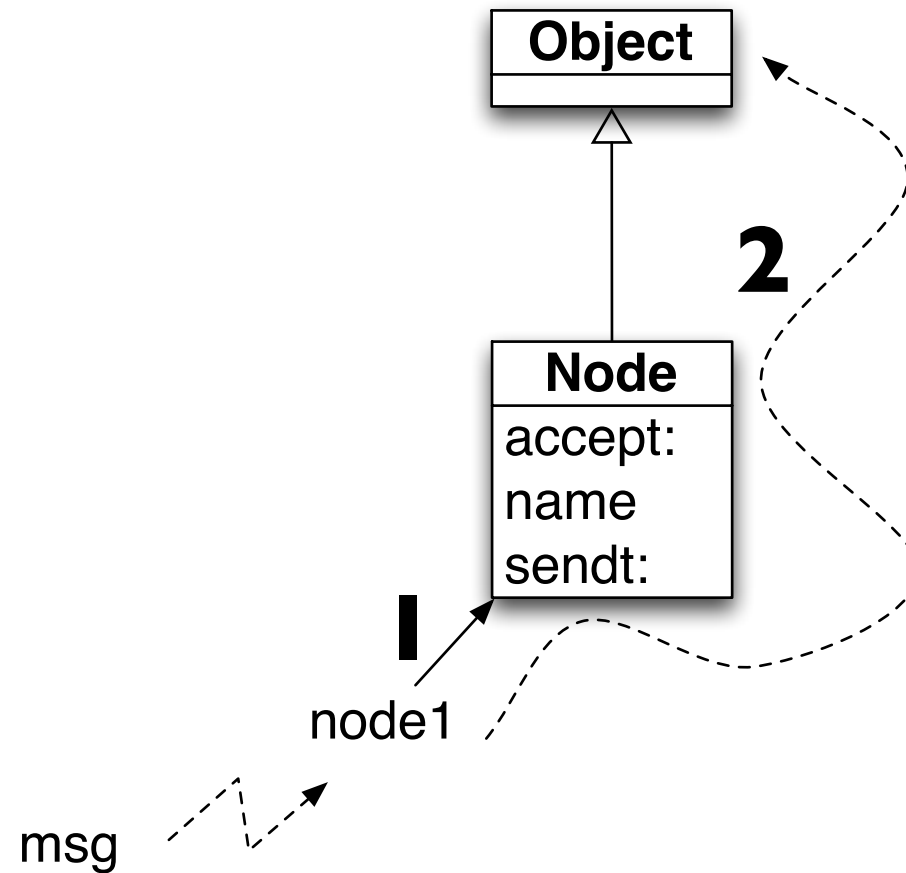


# Instance and Class



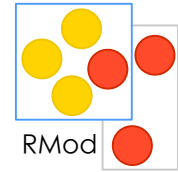
- Only one model
- Uniformly applied
- Classes are objects too

# Lookup...Class + Inheritance



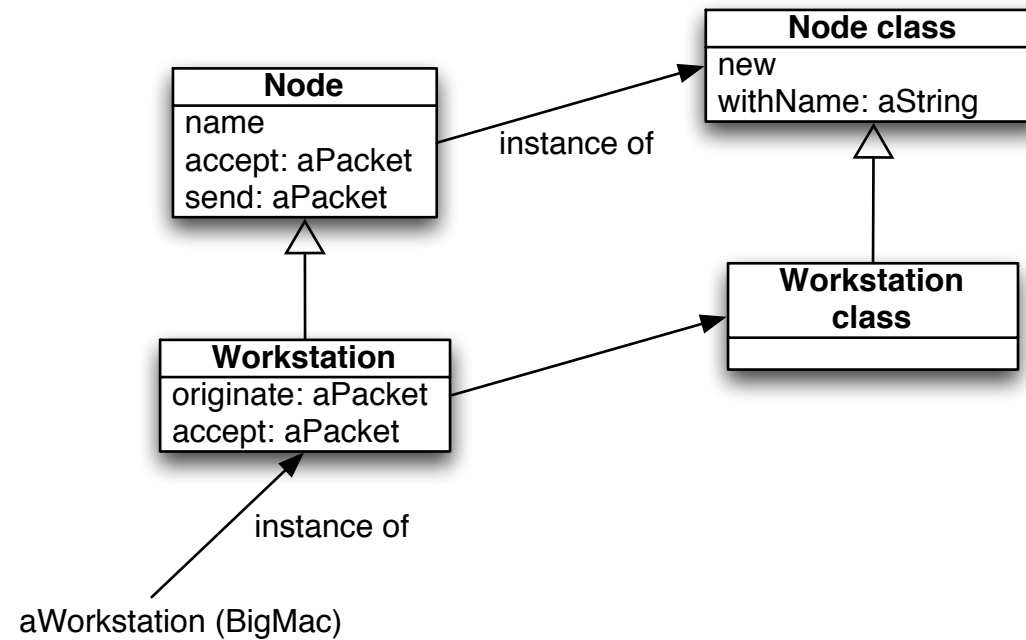
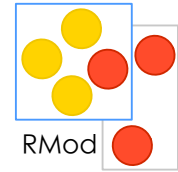
# Classes are objects too

---

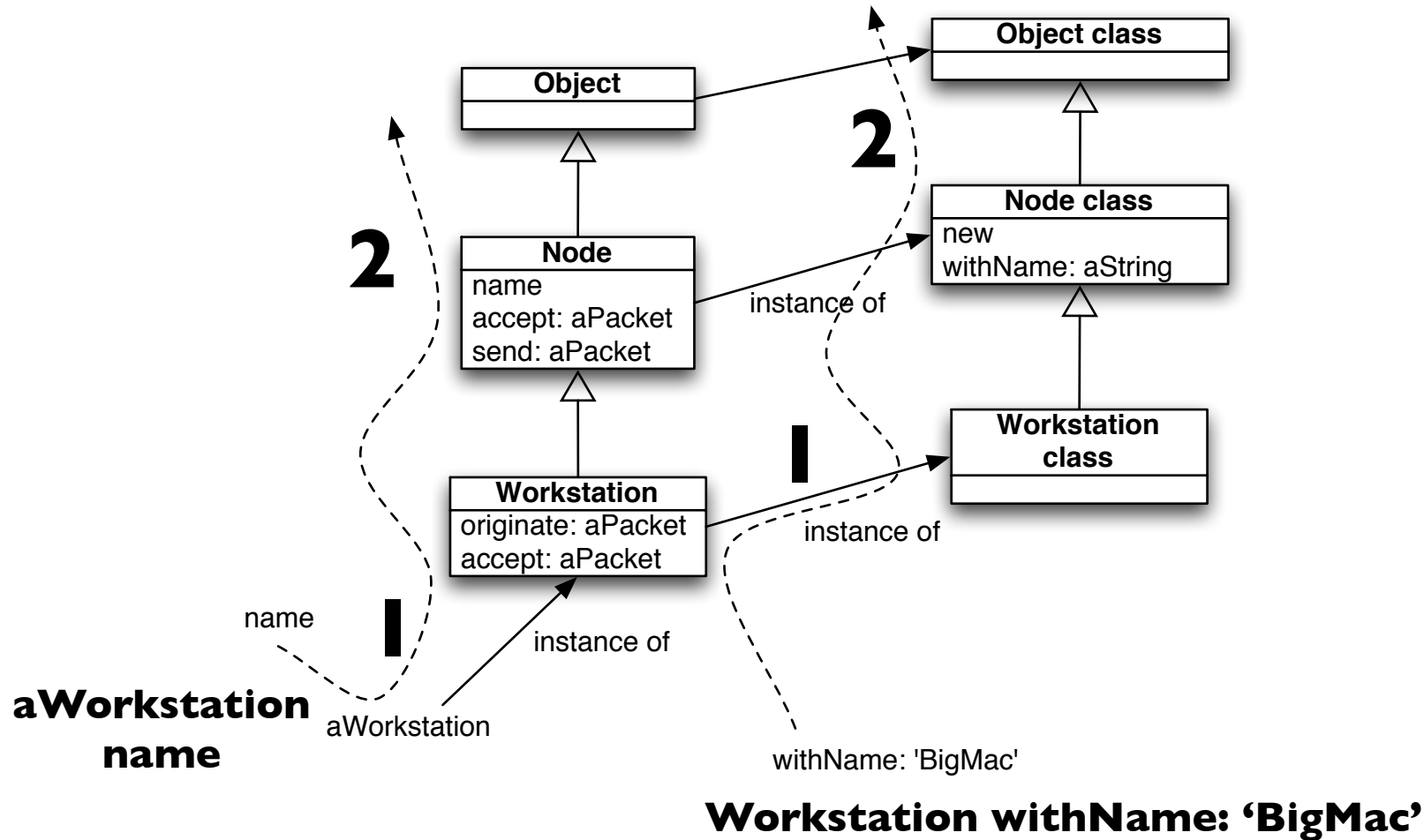
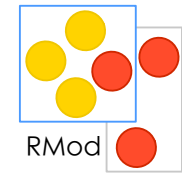


- Instance creation is just a message send to a ... Class
- Same method lookup than with any other objects
- a Class is the single instance of an anonymous class
  - Point is the single instance of Point class

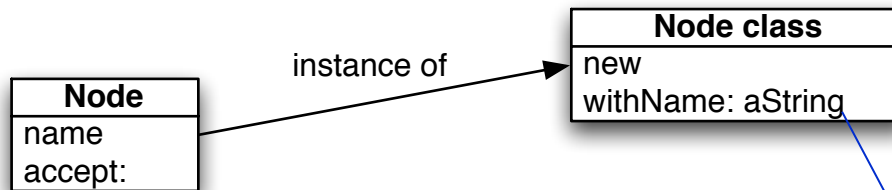
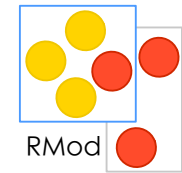
# Class Parallel Inheritance



# Lookup and Class Methods



# About the Buttons



System Browser: Node

Refactory-Supp	Node	-- all --	withName:
Refactory-Parse		instance creati	
Refactory-Parse			
Refactory-Sque			
Refactory-Scann			
Refactory-Refac	instance ? class		

browse senders implementors versions inheritance hierarchy inst v

**withName: aSymbol**

↑ self new name: aSymbol

System Browser: Node

Refactory-Supp	Node	-- all --	name
Refactory-Parse		accessing	
Refactory-Parse			
Refactory-Sque			
Refactory-Scann			
Refactory-Refac	instance ? class		

browse senders implementors versions inheritance hierarchy inst v

**name**

↑ name

# Summary

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