b
UNIVERSITÄT
BERN

2. Smalltalk — a reflective language

Oscar Nierstrasz

Selected material courtesy Stéphane Ducasse

Birds-eye view

Less is More — simple syntax and semantics uniformly applied can lead to an expressive and flexible system, not an impoverished one.

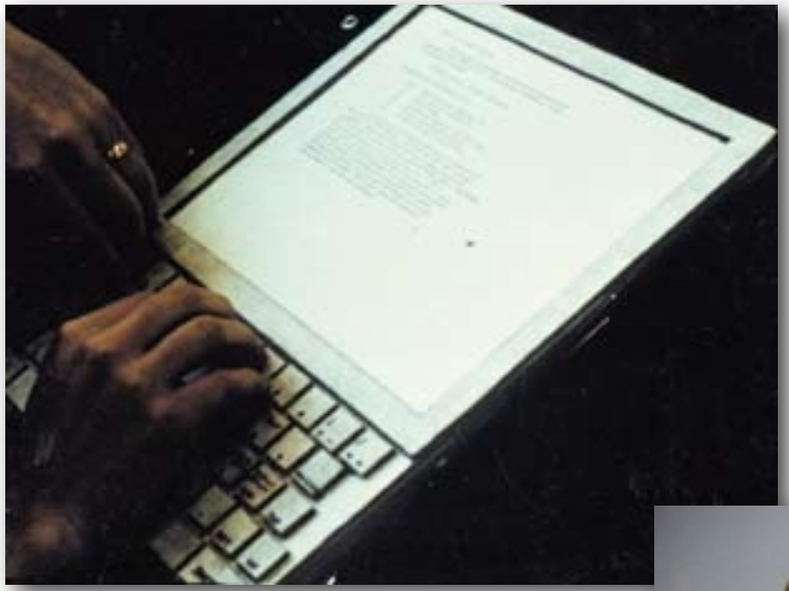
Roadmap

- Smalltalk Basics
- The Environment
- Standard classes

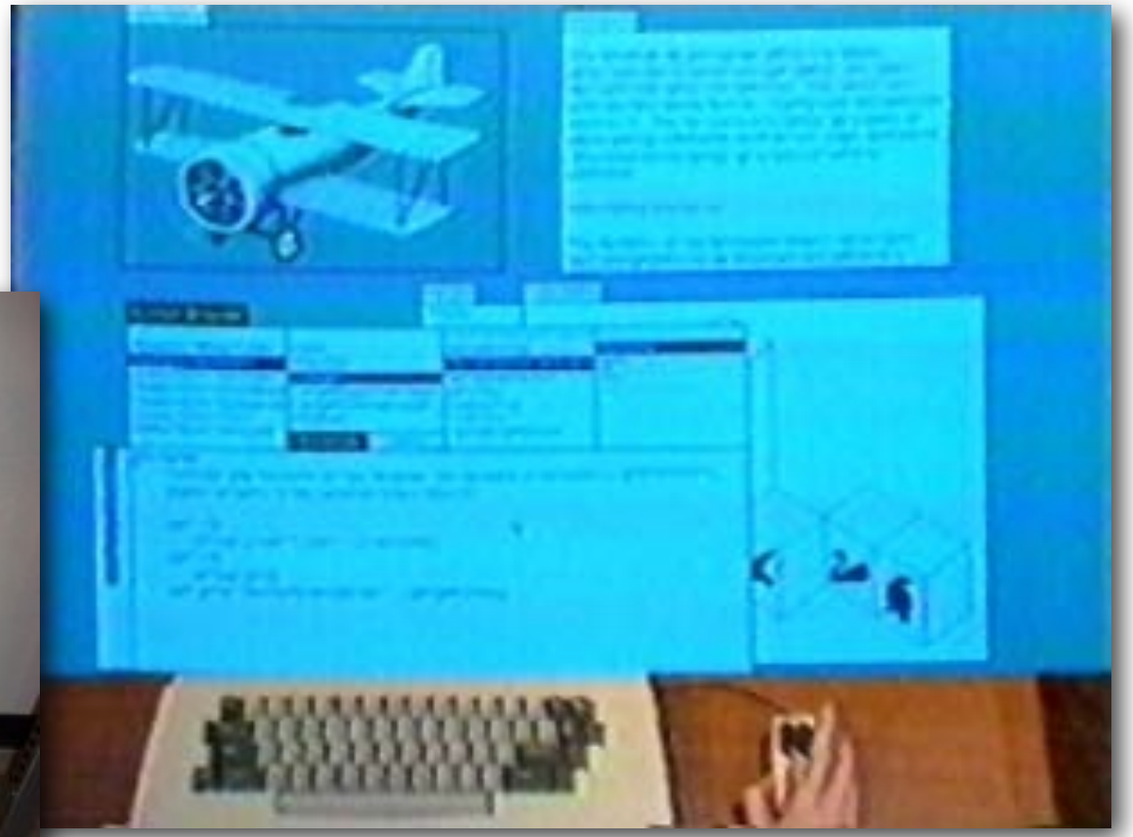
Roadmap

- **Smalltalk Basics**
- The Environment
- Standard classes

The origins of Smalltalk



Alan Kay's Dynabook project (1968)



Alto — Xerox PARC (1973)

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

“How does this work?”

with

“I don’t care”.

— Alan Knight. Smalltalk Guru

Two things to remember ...

Everything is an object

Everything happens by sending messages

The Smalltalk object model

- **Every object is an instance of one class**
  - ... which is also an object
  - Single inheritance
  - A class defines the structure and the behavior of its instances.
- **Dynamic binding**
  - (Nearly) every object is a reference
  - All variables are dynamically typed and bound
- **State is private to objects**
  - “Protected” for subclasses
  - Encapsulation boundary is the object
- **Methods are public**
  - “private” methods by convention only

Smalltalk Syntax

Every expression is a message send

- Unary messages

```
Transcript cr
5 factorial
```

- Binary messages

```
3 + 4
```

- Keyword messages

```
Transcript show: 'hello world'
2 raisedTo: 32
3 raisedTo: 10 modulo: 5
```

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)
```

Precedence

First unary, then binary, then keyword:

```
2 raisedTo: 1 + 3 factorial
```

128

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 raisedTo: 1 + 3 factorial
```

128

Same as:

```
2 raisedTo: (1 + (3 factorial))
```

Use parentheses to force order:

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

9 (!)

7

A typical method in the class Point

Method name

Argument

Comment

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

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

Return

Instance variable

Binary message

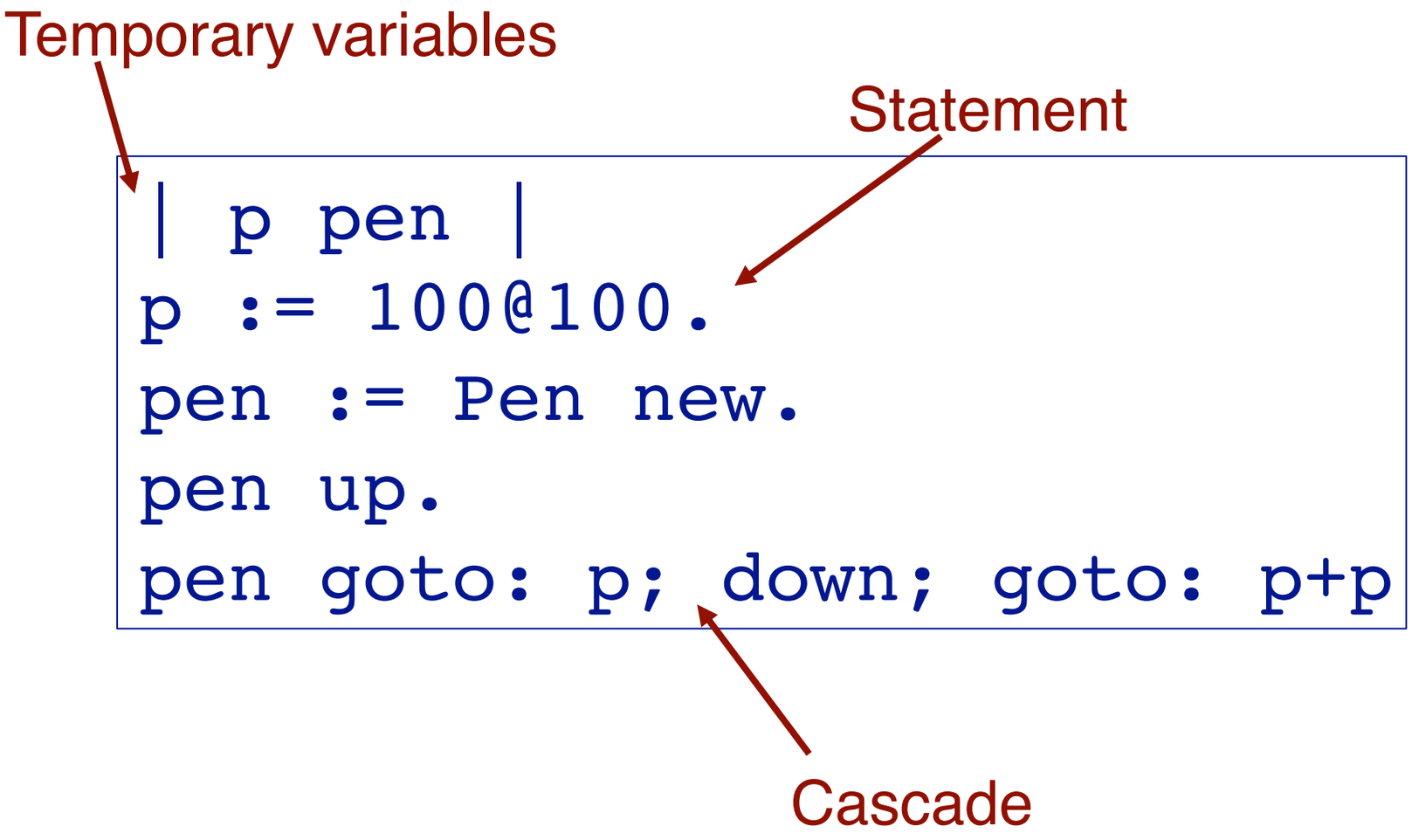
Keyword message

Block

```
(2@3) <= (5@6)
```

```
true
```

Statements and cascades



Literals and constants

Strings & Characters	`'hello'  $a`
Numbers	`1  3.14159`
Symbols	`#yadayada`
Arrays	`#(1 2 3)`
Pseudo-variables	`self super`
Constants	`true false`

Creating objects

> *Class methods*

```
OrderedCollection new
Array with: 1 with: 2
```

> *Factory methods*

```
1@2
1/2
```

```
a Point
a Fraction
```

Creating classes

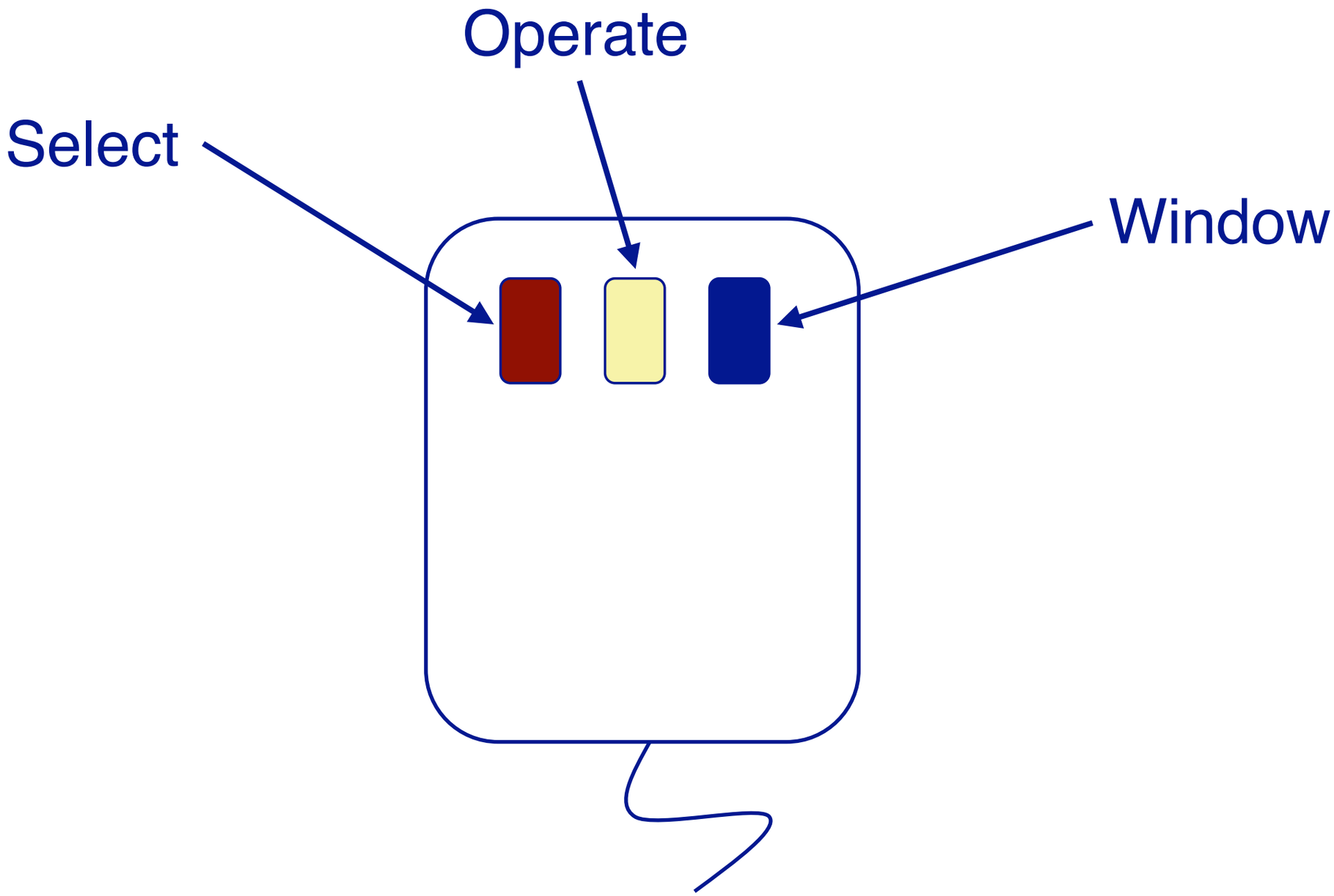
> Send a message to a class (!)

```
Number subclass: #Complex
   instanceVariableNames: 'real imaginary'
   classVariableNames: ''
   poolDictionaries: ''
   category: 'ComplexNumbers'
```

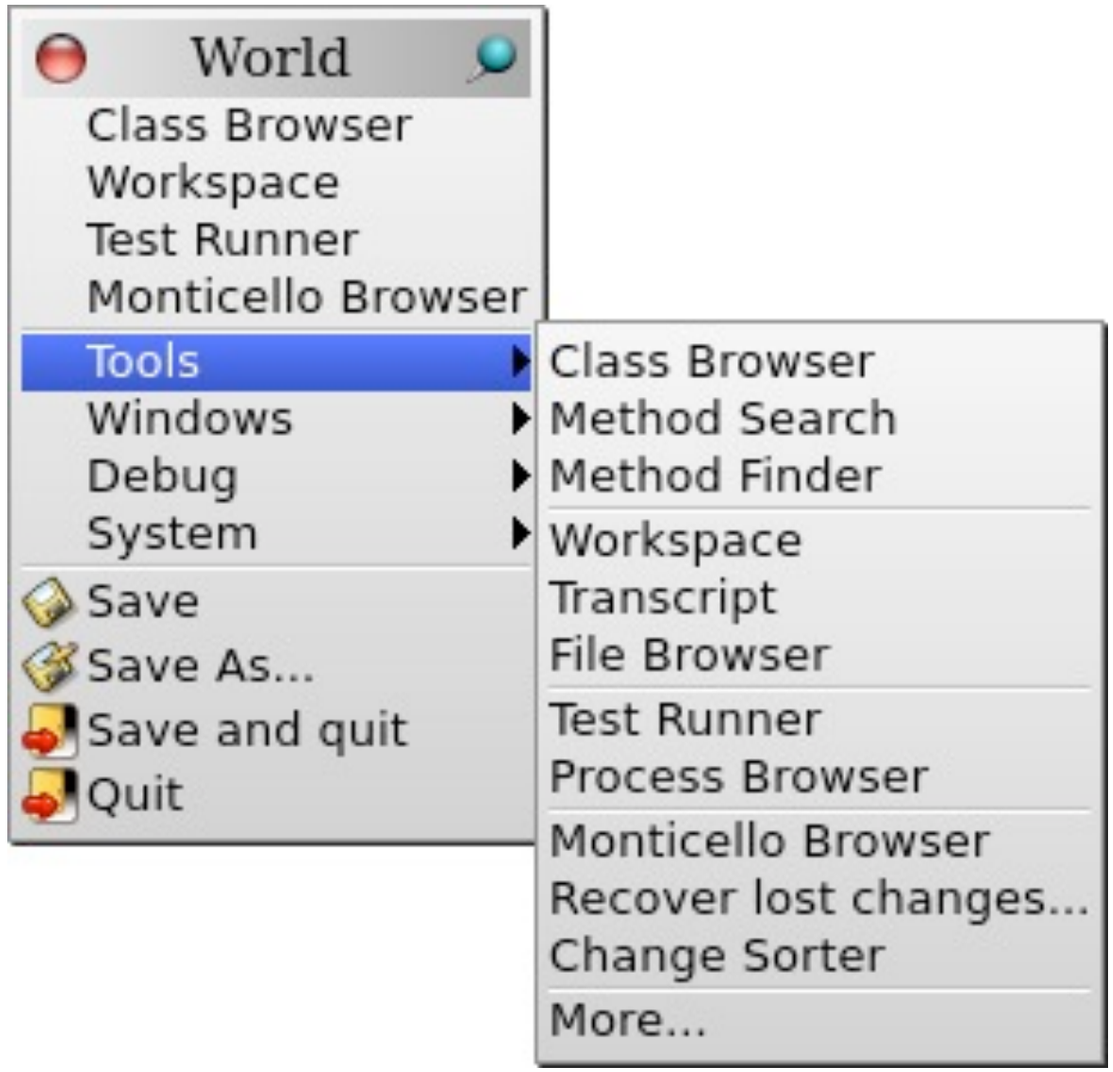
Roadmap

- Smalltalk Basics
- **The Environment**
- Standard classes

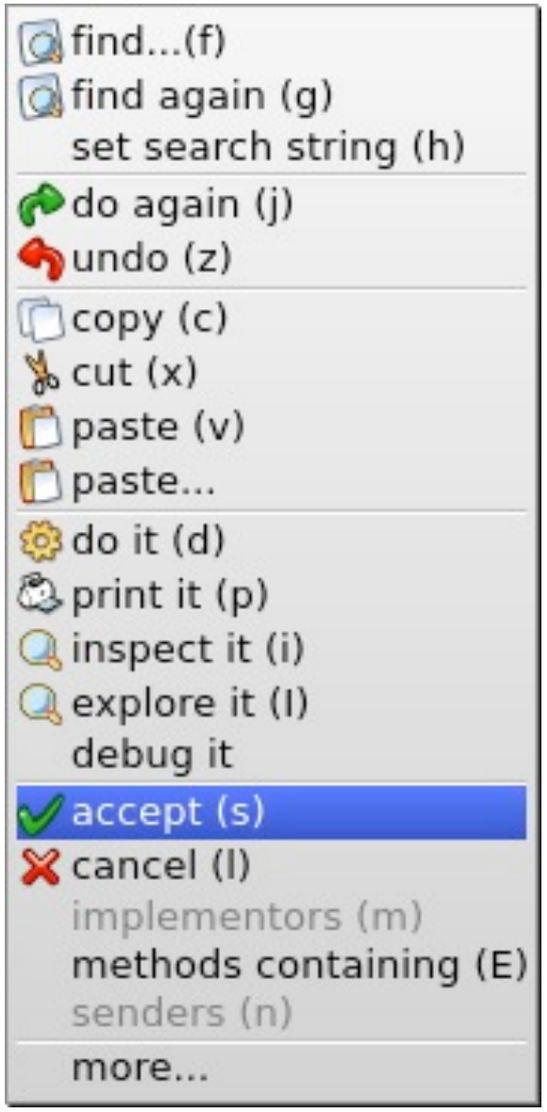
Mouse Semantics



World Menu

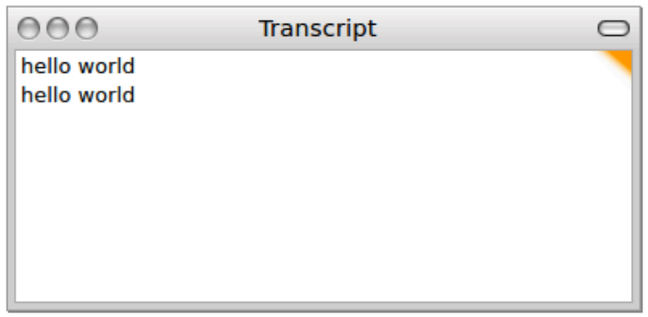

World
Class Browser
Workspace
Test Runner
Monticello Browser
Tools
Windows
Debug
System
Save
Save As...
Save and quit
Quit
Class Browser
Method Search
Method Finder
Workspace
Transcript
File Browser
Test Runner
Process Browser
Monticello Browser
Recover lost changes...
Change Sorter
More...

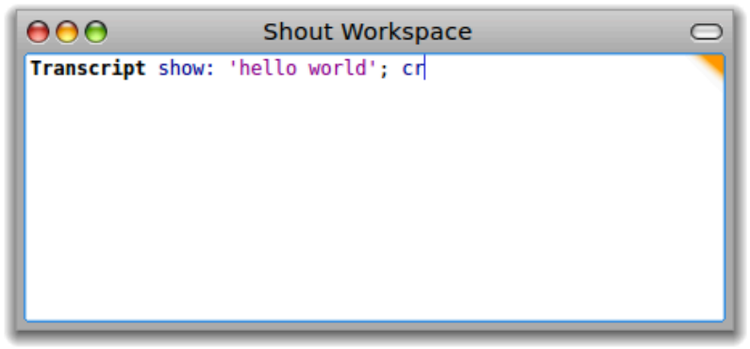
Accept, DoIt, PrintIt and InspectIt

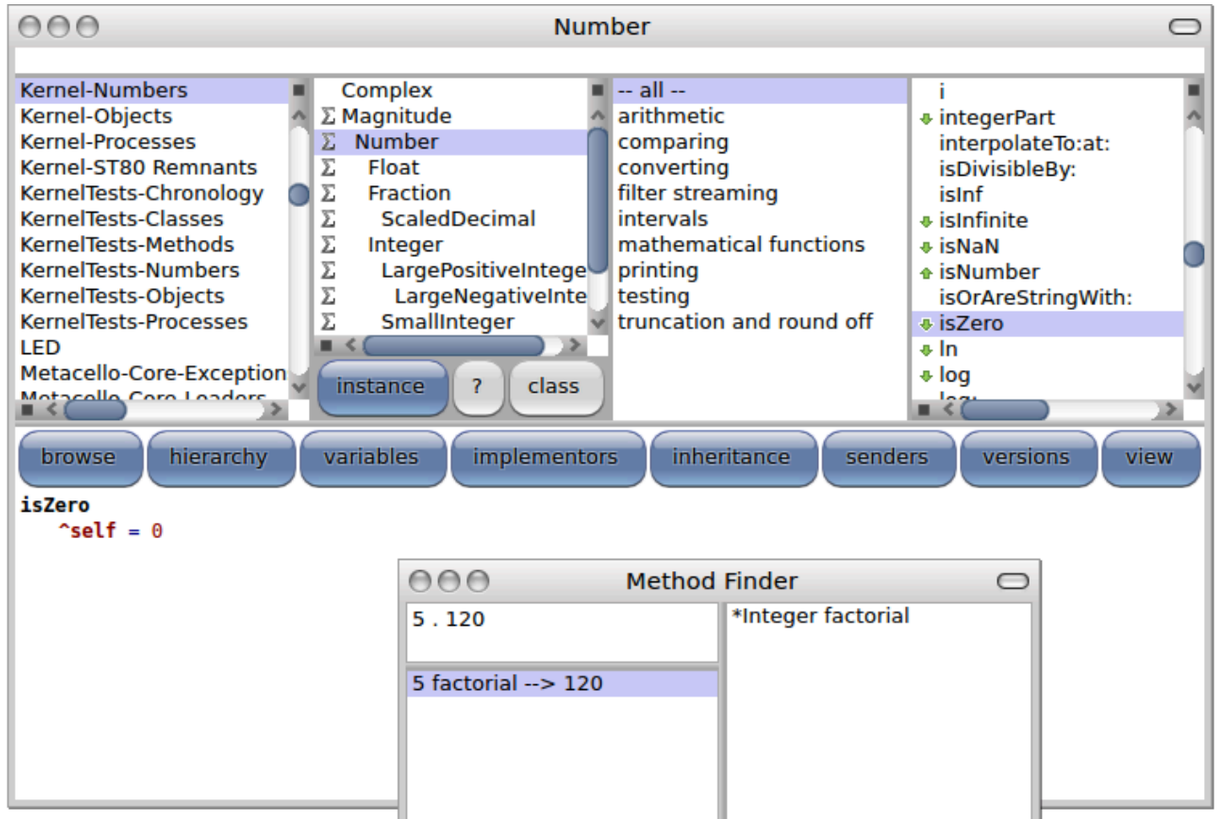


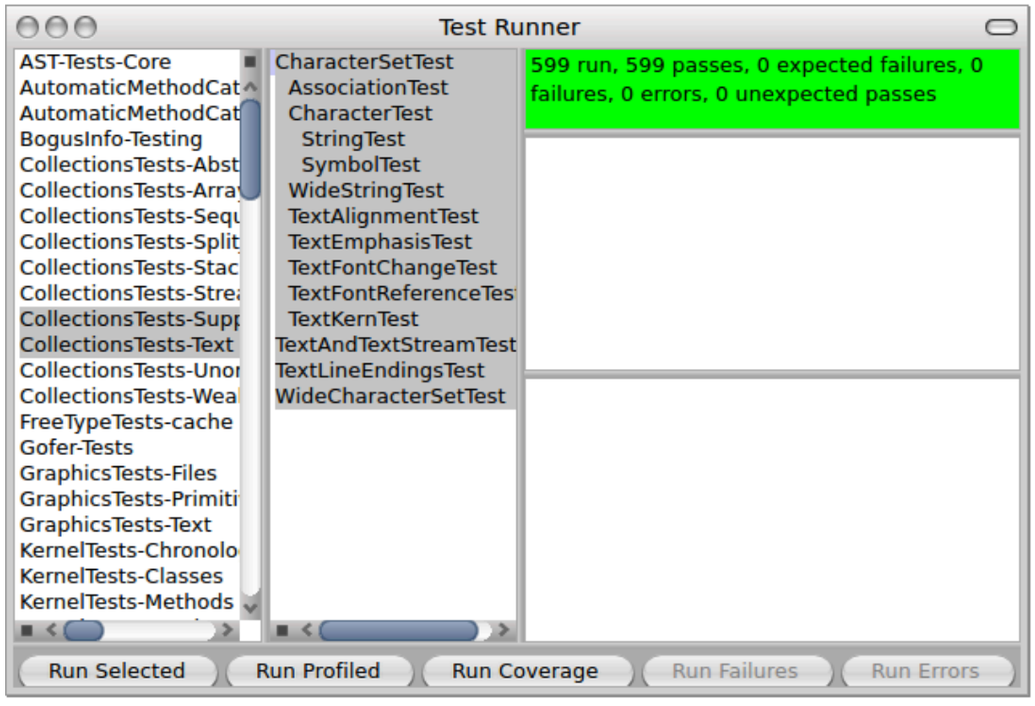
- Accept
  - Compile a method or a class definition
- DoIt
  - Evaluate an expression
- PrintIt
  - Evaluate an expression and print the result (#printOn:)
- InspectIt
  - Evaluate an expression and inspect the result (#inspect)

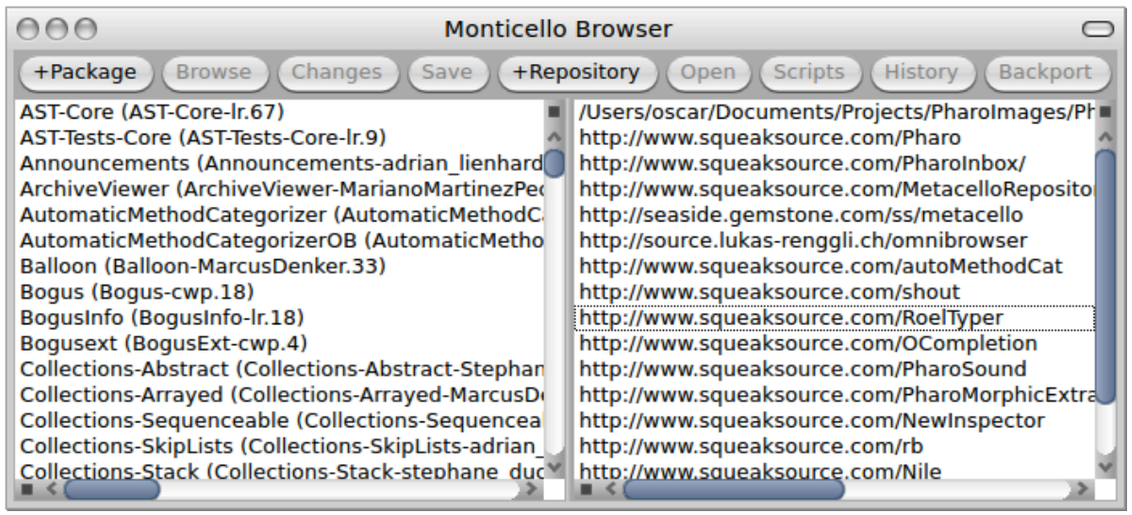
Standard development tools



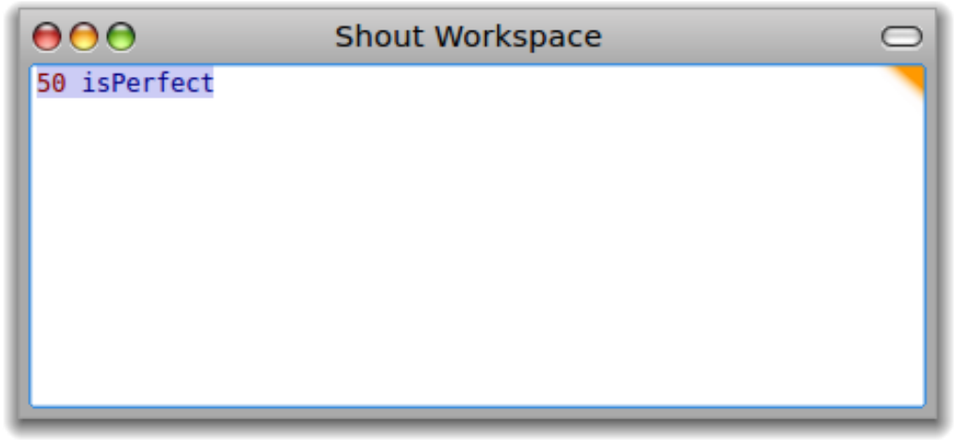


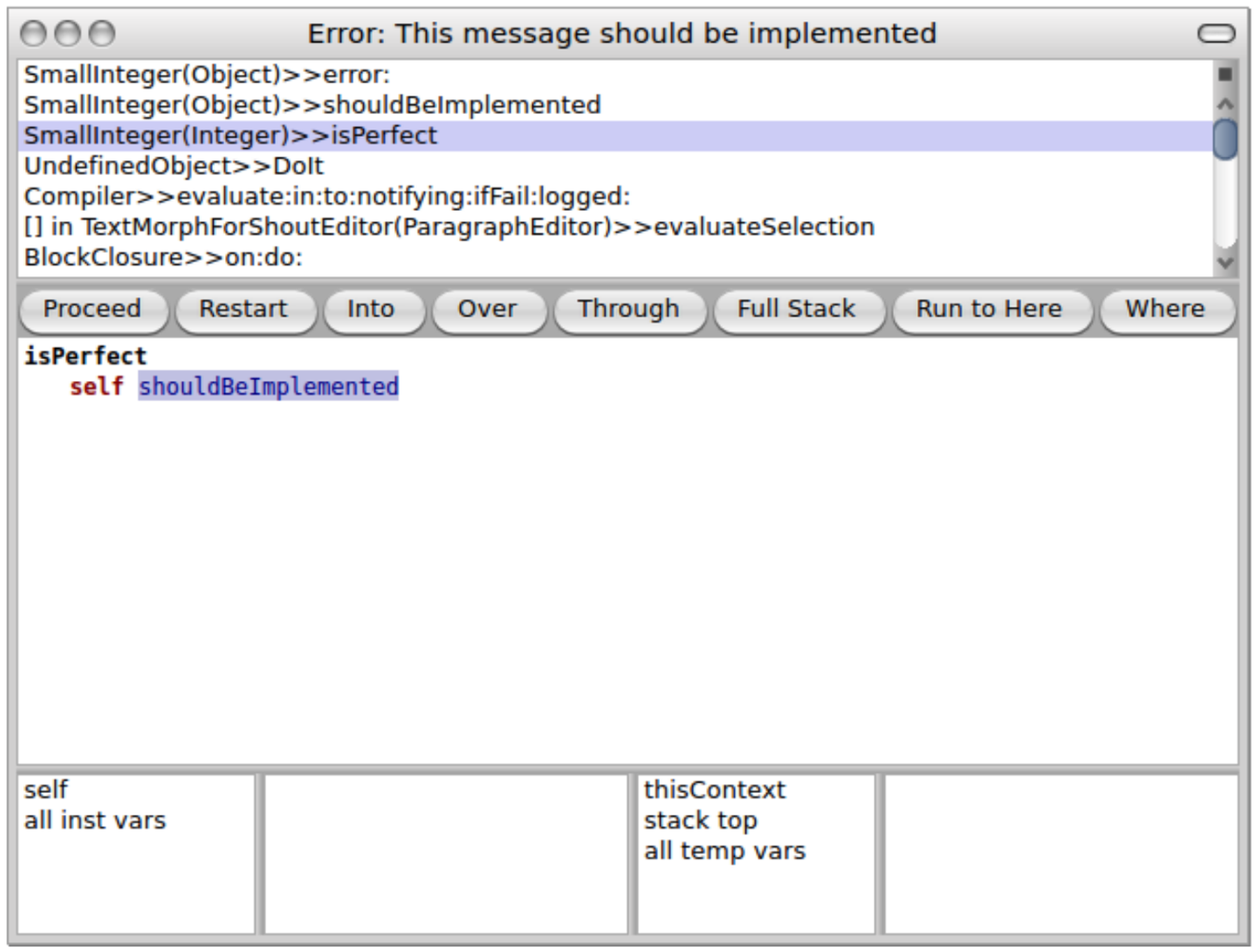


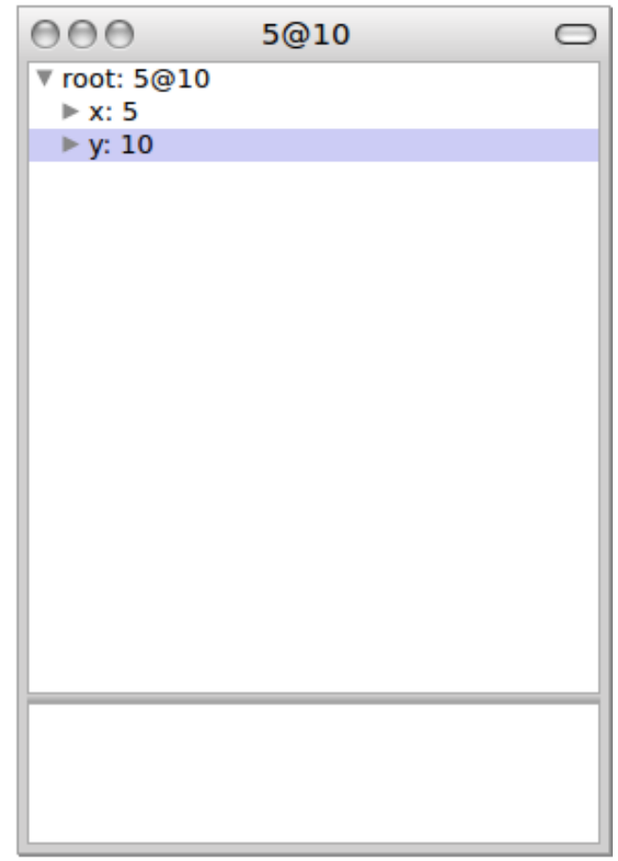




Debuggers, Inspectors, Explorers







SqueakSource.com

SqueakSource

Version 1.3

Home | Projects | Members | Groups | Help

Actions
RSS feed
Back

Authentication
Login

Squeak Examples

Overview | Wiki | RSS Feed | Releases | Blessings | Versions | Latest

Project Description

Examples for the Smalltalk course http://www.iam.unibe.ch/~scg/Teaching/Smalltalk/index.html

Members

Creator: Oscar Nierstrasz
Admin: Oscar Nierstrasz

Registration

```
MCHttpRepository
    location: 'http://www.squeaksource.com/SqueakExamples'
    user: ''
    password: ''
```

Links

http://www.squeaksource.com/SqueakExamples.html
http://www.squeaksource.com/SqueakExamples

Statistics

Registered:	19 March 2006 3:59:41 pm
Total Releases:	0
Total Versions:	3
Total Downloads:	5

XHTML | CSS | RSS — 20 March 2006

Categories, Projects and Packages

- A system category `MyProject` (and possibly `MyProject-*`) contains the classes of your application

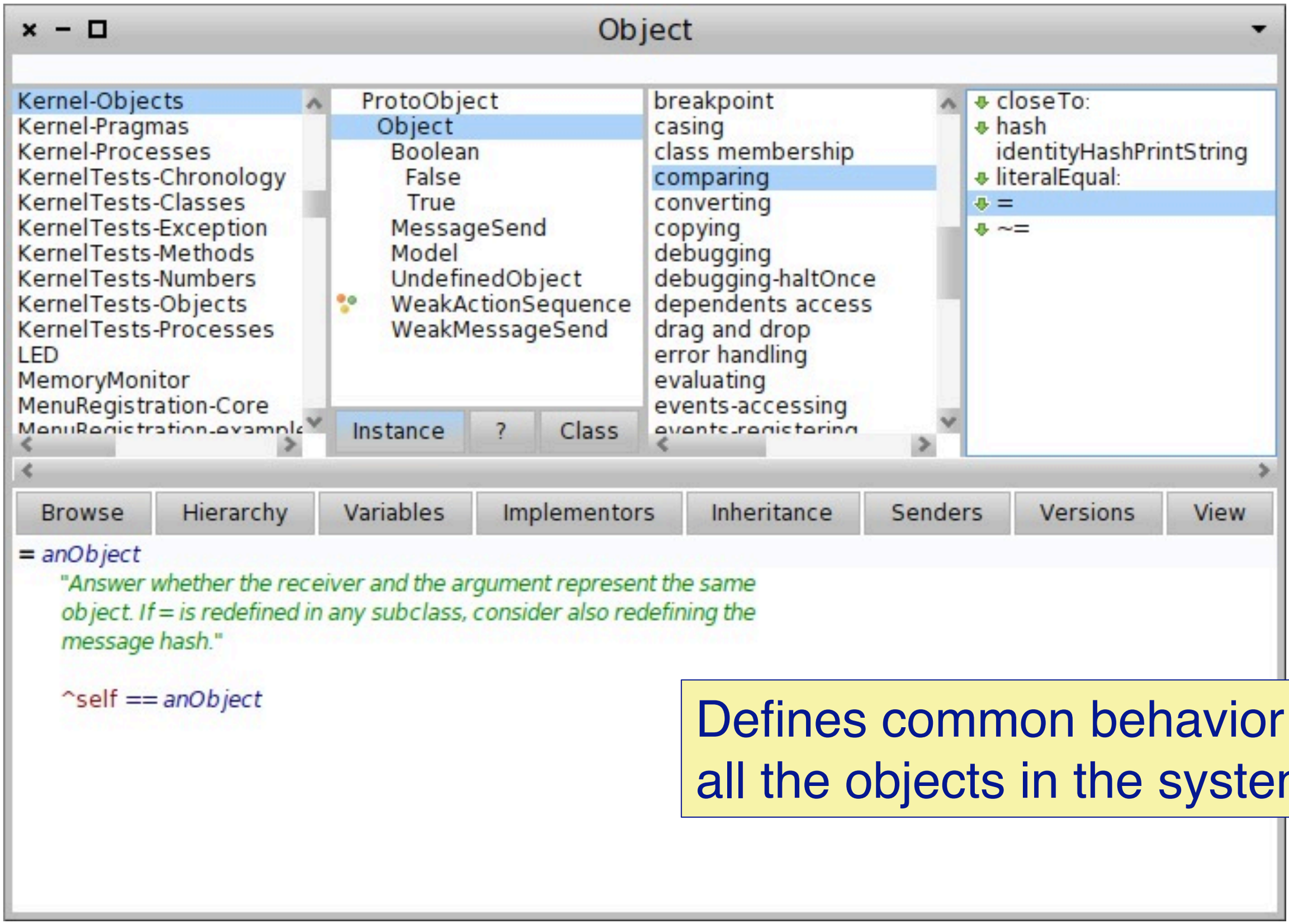
- A Monticello package `MyProject` contains the categories `MyProject` and `MyProject-*`

- A SqueakSource project `MyProject` stores everything in the Monticello package `MyProject`

Roadmap

- Smalltalk Basics
- The Environment
- **Standard classes**

Object



Defines common behavior for all the objects in the system.

Identity vs. Equality

- = tests Object value
  - Should normally be overridden
    - *Default implementation is == !*
  - You should override hash too!

- == tests Object identity
  - *Should never be overridden*

```
'foo','bar' = 'foobar'
'foo','bar' == 'foobar'
```

Identity vs. Equality

- = tests Object value
  - Should normally be overridden
    - *Default implementation is == !*
  - You should override hash too!

- == tests Object identity
  - *Should never be overridden*

```
'foo','bar' = 'foobar'
'foo','bar' == 'foobar'
```

```
true
false
```

Printing

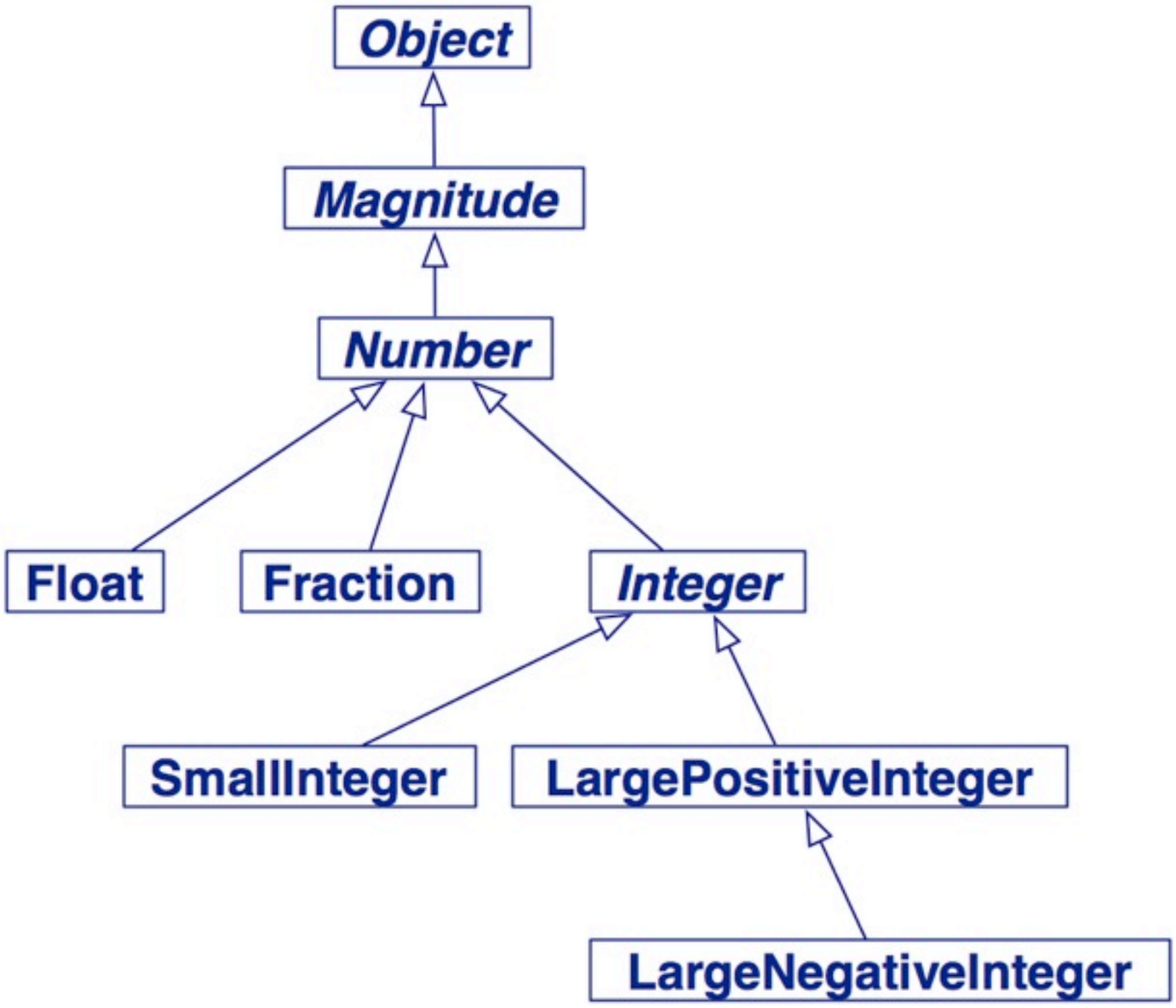
- Override `printOn:` to give your objects a sensible textual representation

```
Fraction>>printOn: aStream
  aStream nextPut: $(.
  numerator printOn: aStream.
  aStream nextPut: $/.
  denominator printOn: aStream.
  aStream nextPut: $).
```

Object methods to support the programmer

`error: aString`	Signal an error
`doesNotUnderstand: aMessage`	Handle unimplemented message
`halt, halt: aString`	Invoke the debugger
`subclassResponsibility`	The sending method is abstract
`shouldNotImplement`	Disable an inherited method
`deprecated: anExplanationString`	Warn that the sending method is deprecated.

Numbers

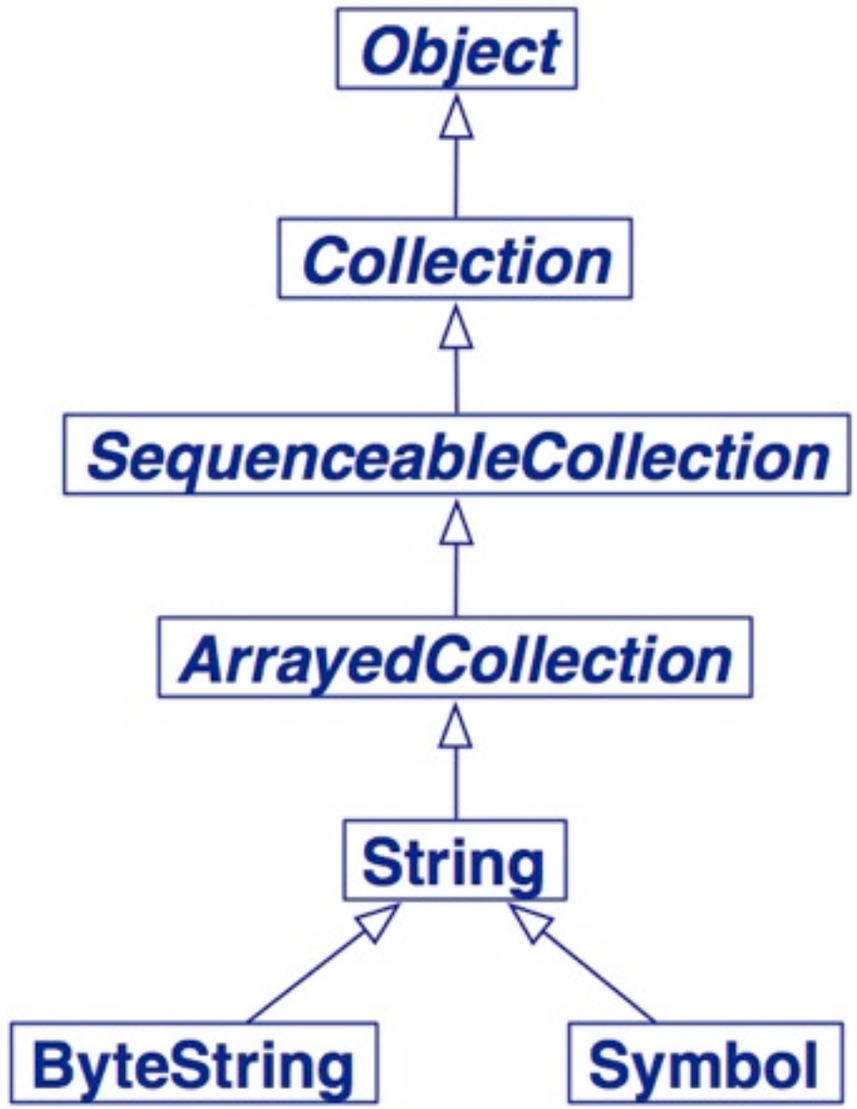

Object
Magnitude
Number
Float
Fraction
Integer
SmallInteger
LargePositiveInteger
LargeNegativeInteger

Abstract methods in Smalltalk

```
Number>>+ aNumber
   "Answer the sum of the receiver and aNumber."

   self subclassResponsibility
```

```
Object>>subclassResponsibility
   "This message sets up a framework for the behavior of the
   class' subclasses. Announce that the subclass should have
   implemented this message."

   self error: 'My subclass should have overridden ',
      thisContext sender selector printString
```

Strings


Object
Collection
SequenceableCollection
ArrayedCollection
String
ByteString
Symbol

Strings

```
#mac asString
12 printString
String with: $A
'can''t' at: 4
'hello', ' ', 'world'
```

> To introduce a single quote inside a string, just double it.

Strings

```
#mac asString
12 printString
String with: $A
'can''t' at: 4
'hello', ' ', 'world'
```

```
'mac'
'12'
```

- To introduce a single quote inside a string, just double it.

Strings

```
#mac asString
12 printString
String with: $A
'can''t' at: 4
'hello', ' ', 'world'
```

```
'mac'
'12'
'A'
```

> To introduce a single quote inside a string, just double it.

Strings

```
#mac asString
12 printString
String with: $A
'can''t' at: 4
'hello', ' ', 'world'
```

```
'mac'
'12'
'A'
$'
```

- To introduce a single quote inside a string, just double it.

Strings

```
#mac asString
12 printString
String with: $A
'can''t' at: 4
'hello', ' ', 'world'
```

```
'mac'
'12'
'A'
$'
'hello world'
```

> To introduce a single quote inside a string, just double it.

Literal and dynamic arrays

Literal arrays

```
#(1 + 2 . 3 )
```

```
#(1 #+ 2 #. 3)
```

Dynamic arrays

```
{ 1 + 2 . 3 }

Array with: 1+2 with: 3
```

```
#(3 3)

#(3 3)
```

`{ … }` is a shortcut for `Array new …`

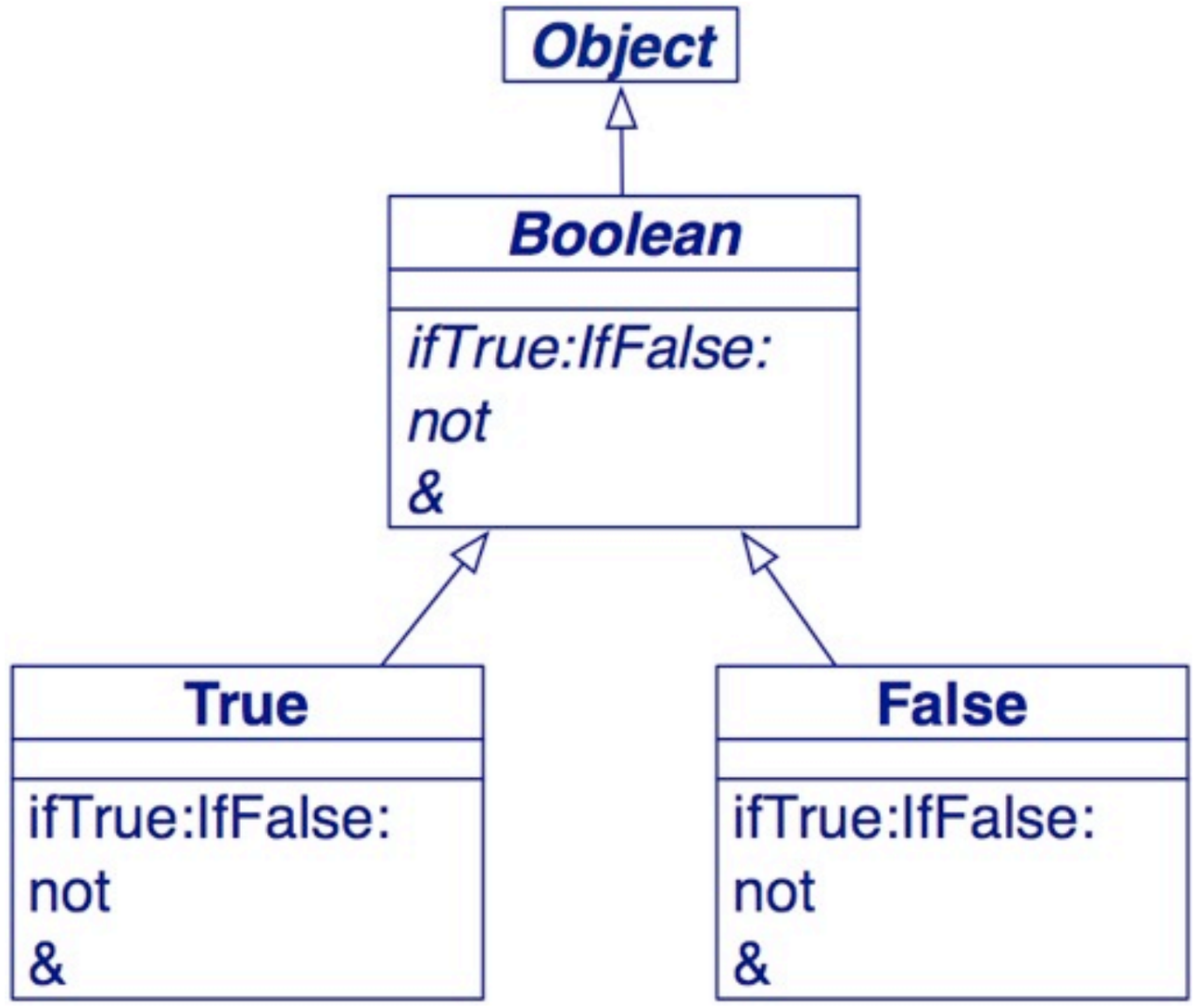
Symbols vs. Strings

- Symbols are used as method selectors and unique keys for dictionaries
  - Symbols are read-only objects, strings are mutable
  - A symbol is unique, strings are not

```
'cal', 'vin' == 'calvin'.

('cal', 'vin') asSymbol == #calvin.
```

```
false

true
```

Booleans


Object
Boolean
ifTrue:IfFalse:
not
&
True
ifTrue:IfFalse:
not
&
False
ifTrue:IfFalse:
not
&

IfTrue: IfFalse:

```
Integer>>factorial
   "Answer the factorial of the receiver."

   self = 0 ifTrue: [^ 1].
   self > 0 ifTrue: [^ self * (self - 1) factorial].
   self error: 'Not valid for negative integers'
```

Six Pseudo-Variables

The following pseudo-variables are hard-wired into the Smalltalk compiler.

`nil`	A reference to the UndefinedObject
`true`	Singleton instance of the class True
`false`	Singleton instance of the class False
`self`	Reference to this object Method lookup starts from object's class
`super`	Reference to this object (!) Method lookup starts from the superclass
`thisContext`	Reification of execution context

Control Constructs

- All control constructs in Smalltalk are implemented by message passing
  - No keywords
  - Open, extensible
  - Built up from Booleans and Blocks

Blocks

- A Block is a *closure*
  - A function that captures variable names in its lexical context
  - I.e., a lambda abstraction
  - First-class value: can be stored, passed, evaluated
- Use to delay evaluation
- Syntax:

```
[ :arg1 :arg2 | |temp1 temp2| expression. expression ]
```

  - Returns last expression of the block

Block Example

```
|sqr|
sqr := [:n | n*n ].
sqr value: 5
```

25

Block evaluation messages

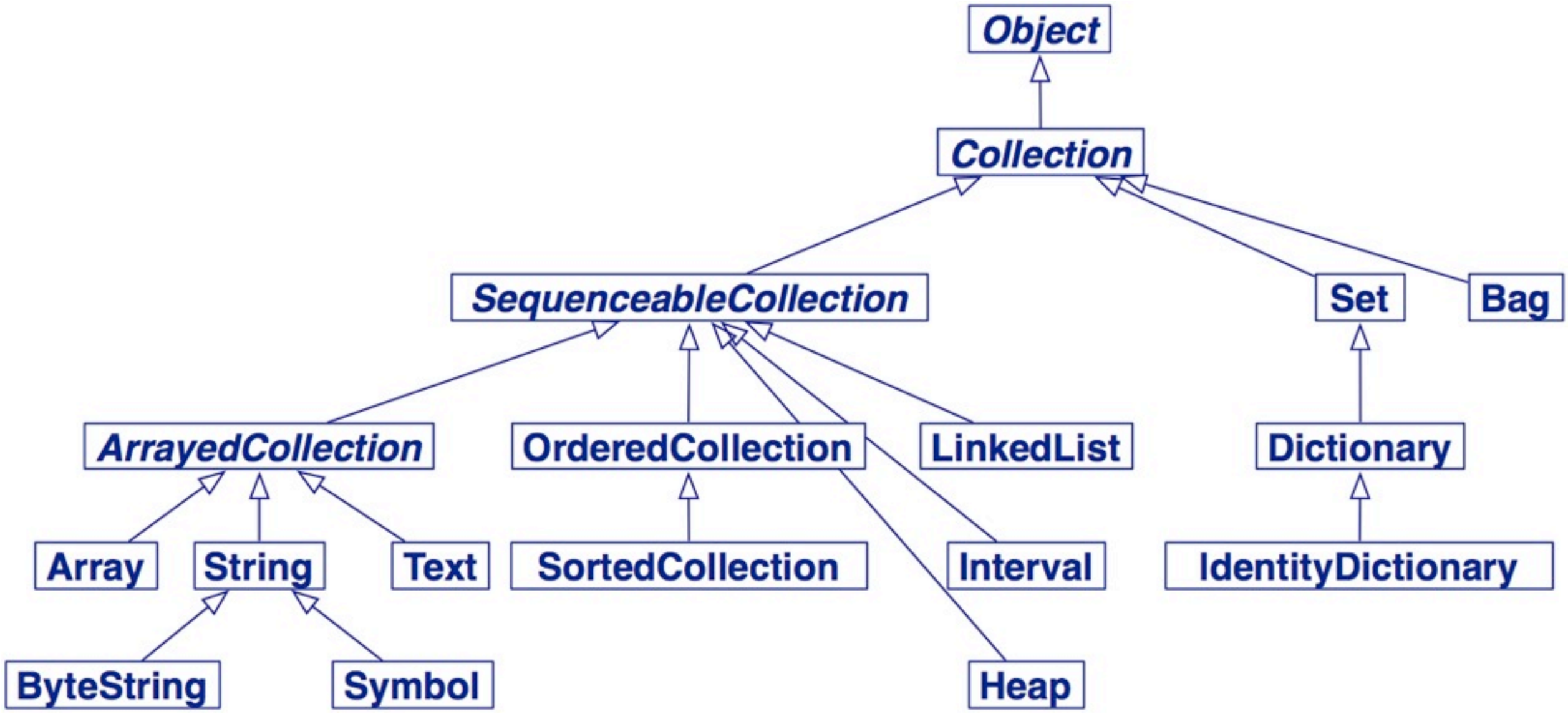
```
[2 + 3 + 4 + 5] value
[:x | x + 3 + 4 + 5 ] value: 2
[:x :y | x + y + 4 + 5] value: 2 value: 3
[:x :y :z | x + y + z + 5] value: 2 value: 3 value: 4
[:x :y :z :w | x + y + z + w] value: 2 value: 3 value: 4 value: 5
```

Various kinds of Loops

```
|n|
n:= 10.
[n>0] whileTrue: [ Transcript show: n; cr. n:=n-1]

1 to: 10 do: [:n | Transcript show: n; cr ]

(1 to: 10) do: [:n | Transcript show: n; cr ]

10 timesRepeat: [ Transcript show: 'hi'; cr ]
```

In each case, what is the target object?

Collections

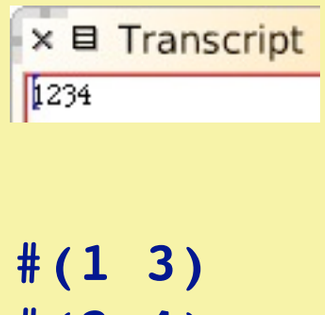


Resist the temptation to program your own collections!

Common messages

```
#(1 2 3 4) includes: 5
#(1 2 3 4) size
#(1 2 3 4) isEmpty
#(1 2 3 4) contains: [:some | some < 0 ]
#(1 2 3 4) do:
   [:each | Transcript show: each ]
#(1 2 3 4) with: #(5 6 7 8)
   do: [:x : y | Transcript show: x+y; cr]
#(1 2 3 4) select: [:each | each odd ]
#(1 2 3 4) reject: [:each | each odd ]
#(1 2 3 4) detect: [:each | each odd ]
#(1 2 3 4) collect: [:each | each even ]
#(1 2 3 4) inject: 0
   into: [:sum :each | sum + each]
```

```
false
4
false
false
```




```
#(1 3)
#(2 4)
1
{false.true.false.true}

10
```

Iteration — the hard road and the easy road

How to get absolute values of a collection of integers?

```
|aCol result|
aCol :=  #( 2 -3 4 -35 4 -11).
result := aCol species new: aCol size.
1 to: aCol size do:
   [ :each | result at: each put: (aCol at: each) abs].
result
```

#(2 3 4 35 4 11)

```
#( 2 -3 4 -35 4 -11) collect: [:each | each abs ]
```

#(2 3 4 35 4 11)

NB: The second solution also works for indexable collections and sets.

What you should know!

- What is the difference between a comment and a string?
- Why does 1+2*3 = 9?
- What is a cascade?
- How is a block like a lambda expression?
- How do you create a new class?
- How do you inspect an object?
- Why does Smalltalk have no special syntax for defining an abstract method or class?

Can you answer these questions?

- Why does Smalltalk support single (and not multiple) inheritance?
- What is the difference between `Point x: 1 y: 2` and `(1@2)`?
- In Smalltalk, what is the difference between “compile time” and “run time”?
- If instance variables are really private, why can we see them with an inspector?