Pharo:
the best system for object-oriented *thinking*

http://stephane.ducasse.free.fr
http://www.pharo.org

Reflecting on our daily practice and experience

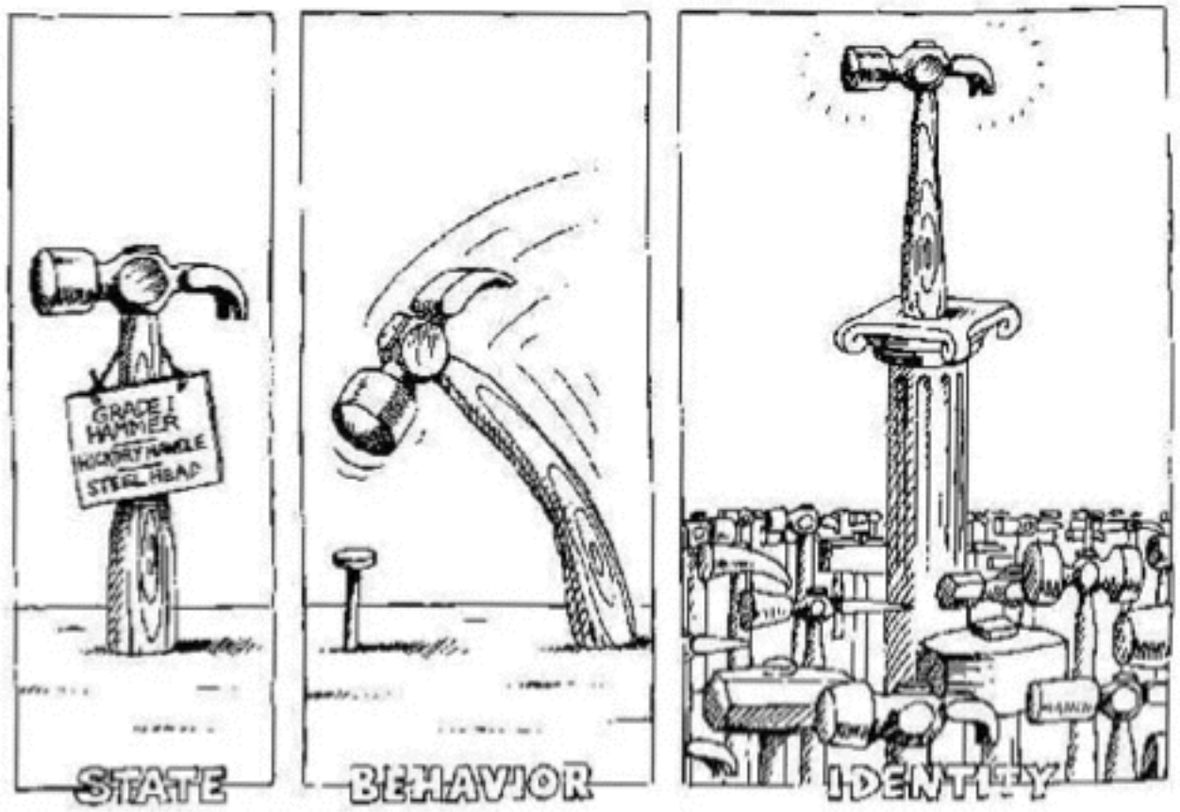
Properties we see

- Turtles all the way down
- Uniform and elegant
- Simple oo model
- But powerful to build really large applications
- Open system
- Learn from the system itself

What is the essence of OOP?

objects + late binding

objects!


GRADE I
HAMMER
HICKORY HANDLE
STEEL HEAD
STATE
BEHAVIOR
IDENTITY

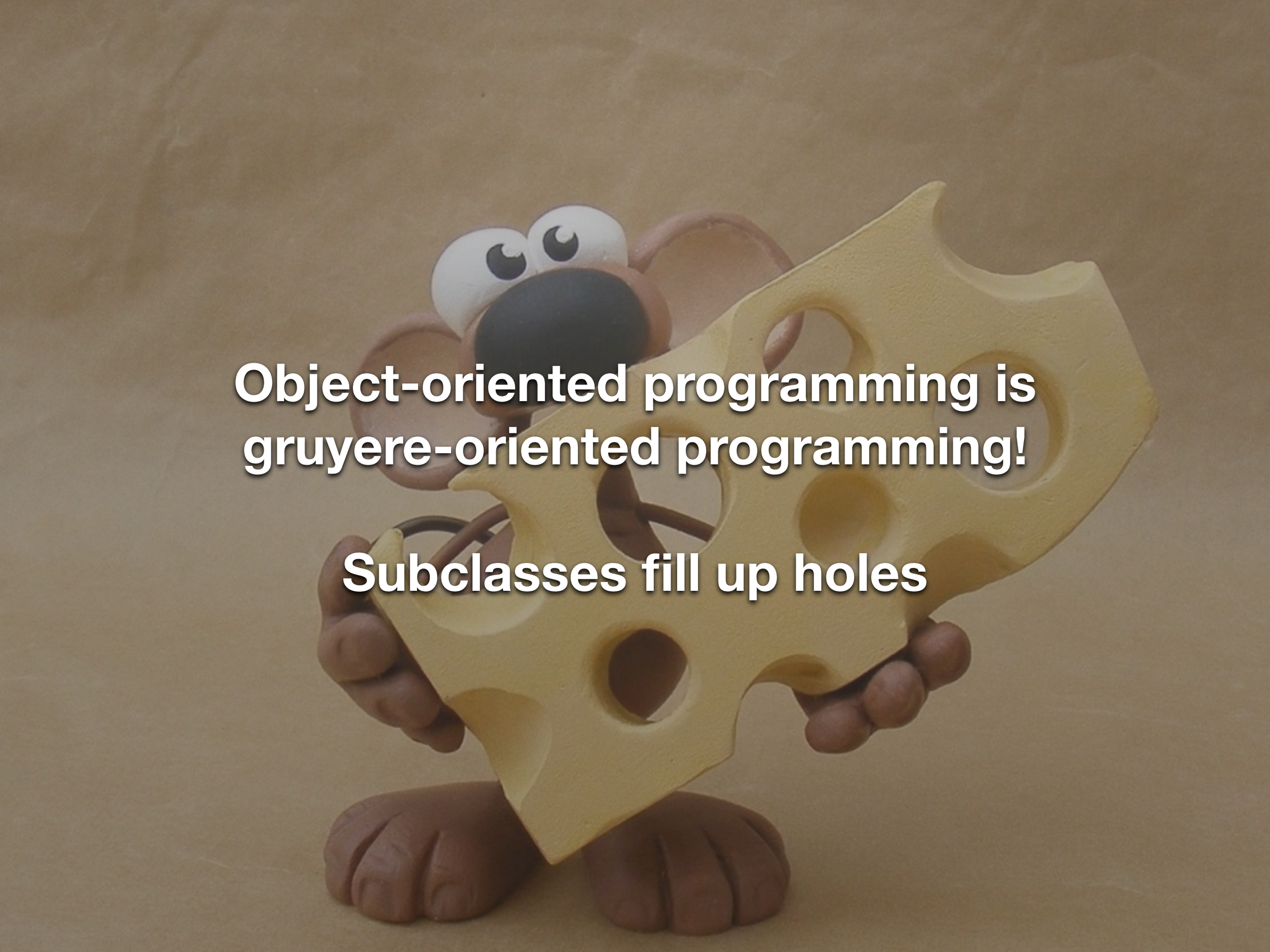
late binding

Late binding

The fact that the “procedure” to be executed depends on the object on which the computation is requested.

Sending a message is a choice operator

Classes embed choices


Object-oriented programming is
gruyere-oriented programming!
Subclasses fill up holes

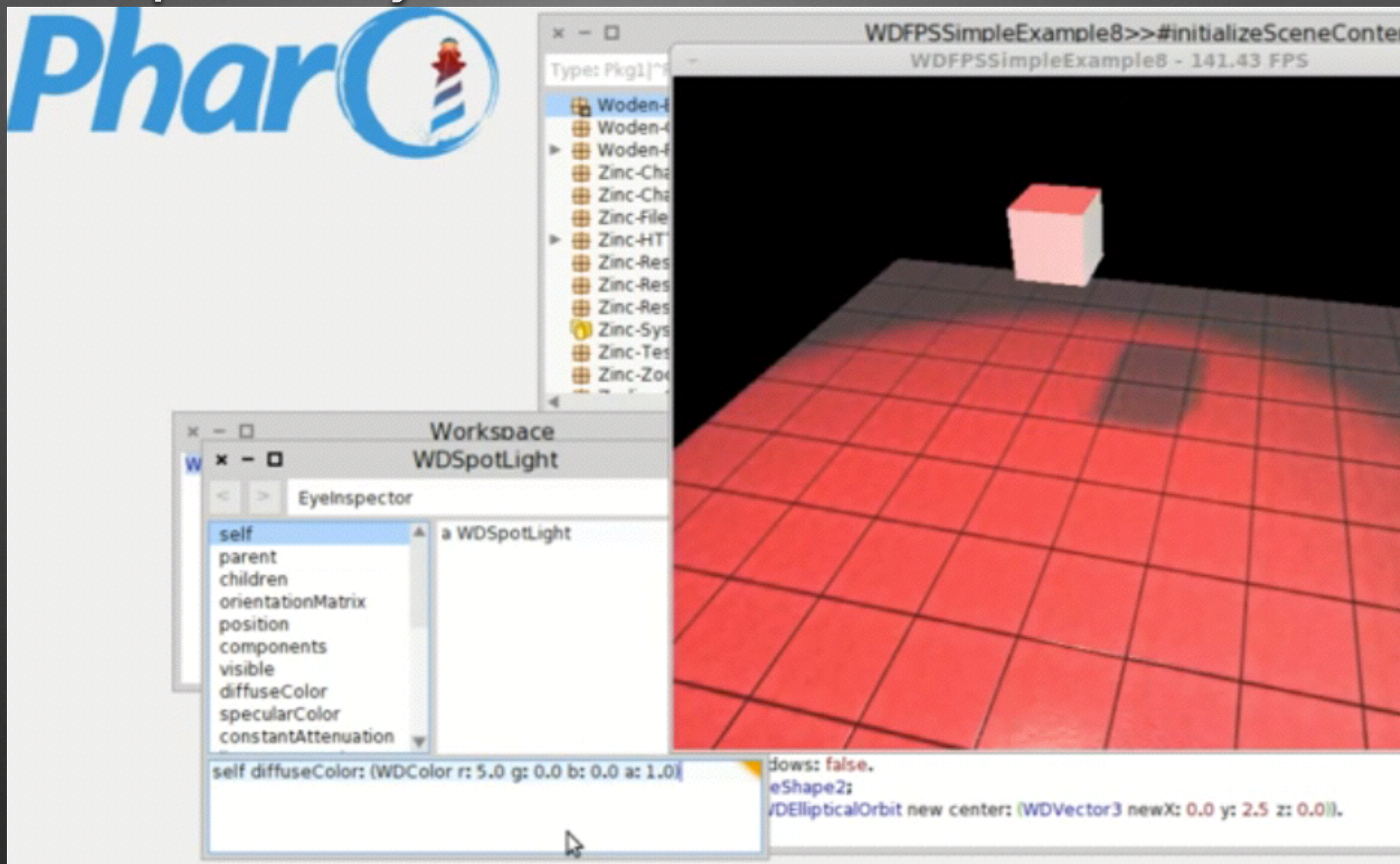
How to convey that objects are living/interactive entities?

How to convey the object feel?

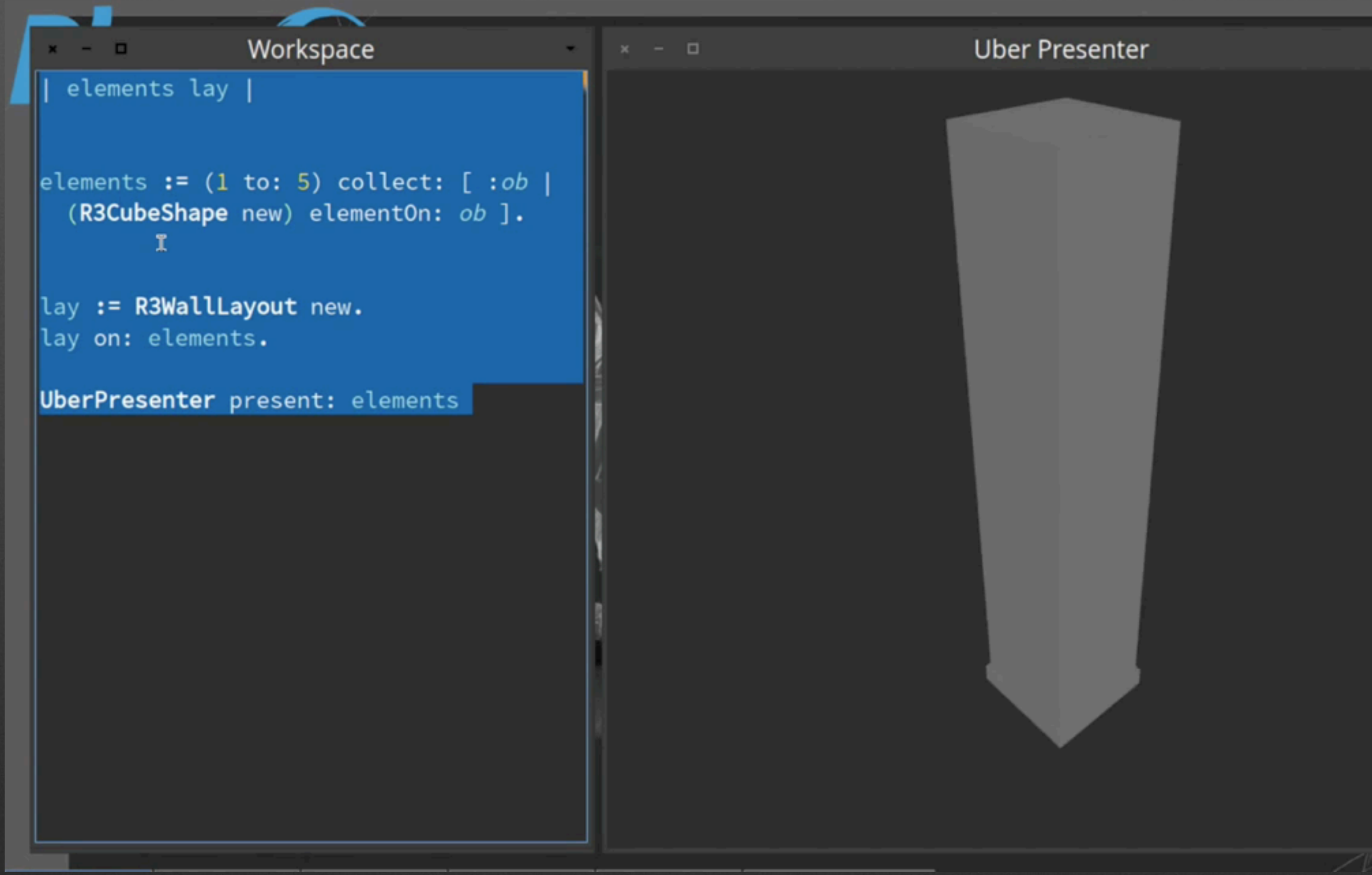
- Live interaction
- Using Inspectors
    - Talk and interact with objects
- Programs are objects too

Scripting live

https://www.youtube.com/watch?v=1Nze9tnwYxY



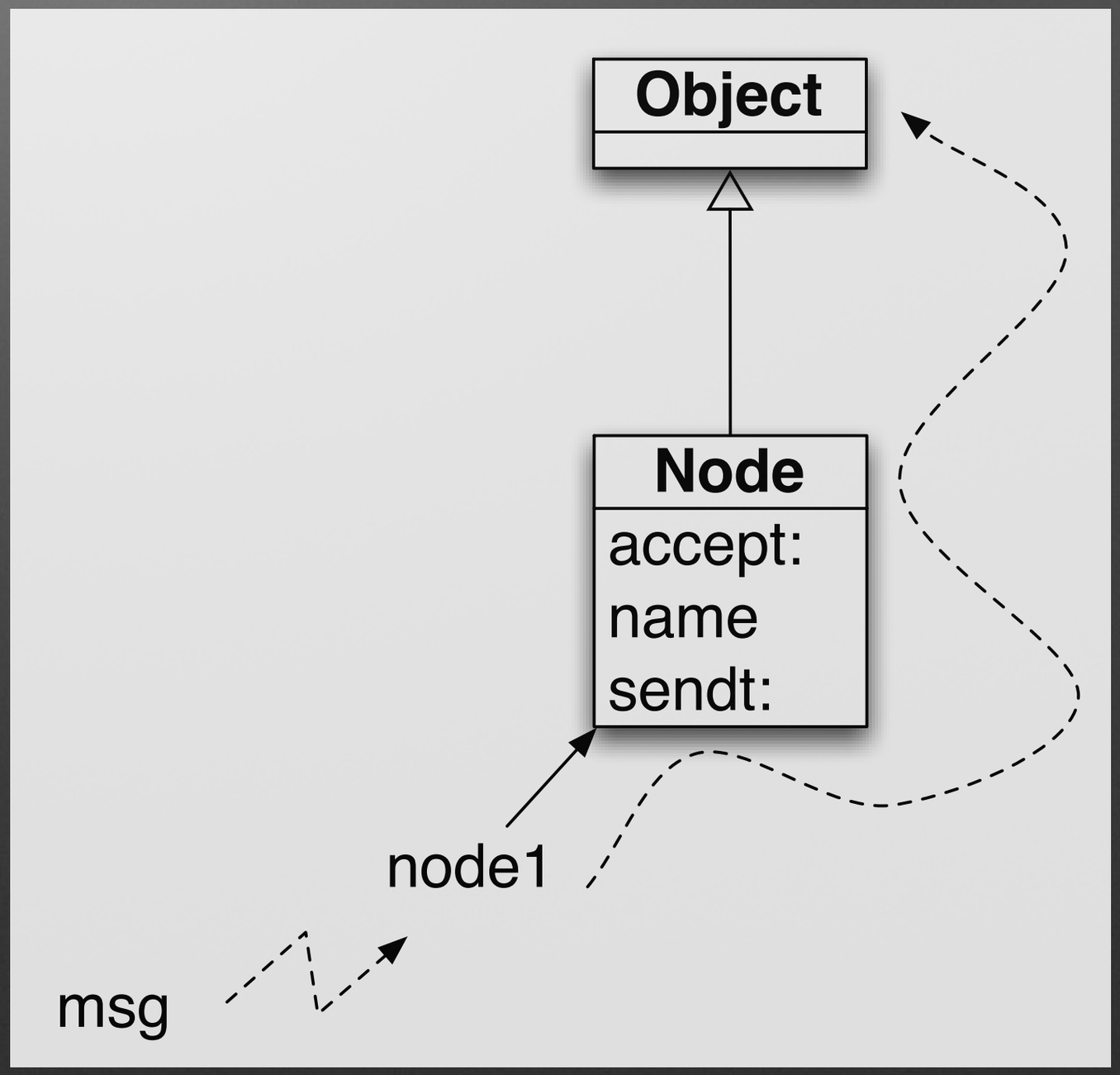
http://youtu.be/CuimMwuZiGA



Even with the vocabulary

- We send messages to objects.
- We do not invoke methods!

Sending a message implies looking up for a method



Simplicity and elegance

- Syntax fits on a postcard!
- Simple but complete object model

Complete Syntax on a Postcard

```
exampleWithNumber: x
	“A method that illustrates every part of Smalltalk method syntax”
	<menu>
	| 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
```

A Pure OO World

Only objects!

mouse, booleans, arrays, numbers, strings, windows, scrollbars, canvas, files, trees, compilers, sound, url, socket, fonts, text, collections, stack, shortcut, streams, …

Closure all the way

```
#(1 0 -2 33 -66) collect: [ :each | each abs ]

        -> #(1 0 2 33 66)
```

```
4 timesRepeat: [ Transcript show: ‘hello’ ]

1 to: 100 by: 3 do: [ :i | Transcript show: i ; cr ]
```

Less is more!

No constructors, no static methods, no operators

No type declaration, no primitive types,

No interfaces, no need for factory

No packages/private/protected modifiers

No parametrized types

No boxing/unboxing

Still powerful

Object Model

- Dynamically typed
- **Everything** is an instance of a class
- All methods are public and virtual
- Attributes are protected
- Single Inheritance

Everything is an object

Objects are instances of Classes

Objects are instances of Classes

(10@200)

Objects are instances of Classes

(10@200) class

Objects are instances of Classes

(10@200) class

Point

Classes are objects too

Classes are objects too

Point selectors

Classes are objects too

Point selectors

an IdentitySet(#eightNeighbors #+ #isZero #sortsBefore: #degrees #printOn: #sideOf: #fourNeighbors #hash #roundUpTo: #min: #min:max: #max #adaptToCollection:andSend: #quadrantOf: #crossProduct: #= #nearestPointOnLineFrom:to: #bitShiftPoint: #* #guarded #insideTriangle:with:with: #grid: #truncateTo: #y #setR:degrees: #normal #directionToLineFrom:to: #truncated #nearestPointAlongLineFrom:to: #theta #scaleTo: #encodePostscriptOn: #> #asPoint #extent: #r #roundTo: #max: #interpolateTo:at: #triangleArea:with: #angleWith: #dotProduct: #isSelfEvaluating #'<=' #to:intersects:to: #'//' #isInsideCircle:with:with: #< #scaleFrom:to: #corner: #to:sideOf: #x #'>=' #roundDownTo: #onLineFrom:to:within: #transposed #ceiling #angle #basicType #translateBy: #asFloatPoint #'\\' #adaptToNumber:andSend: #abs #negated #octantOf: #asIntegerPoint #flipBy:centerAt: #scaleBy: #floor #onLineFrom:to: #isPoint #reflectedAbout: #/ #dist: #asNonFractionalPoint #bearingToPoint: #reciprocal #rotateBy:centerAt: #rotateBy:about: #rounded #setX:setY: #squaredDistanceTo: #normalized #veryDeepCopyWith: #- #storeOn: #rect: #deepCopy #isIntegerPoint #min #adhereTo: #adaptToString:andSend:)

Classes are objects too

Point instVarNames

Classes are objects too

Point instVarNames

>#('x' 'y')

Methods are public

Methods are all late-bound

Instance variables are protected

Single Inheritance

Messages + Objects

```
2 + 5
> 7
```

Yes + is a message sent to 2

Turtles all the way down as an enabling technology

Enabler

“One of the things that drew me to do the Delay refactoring, is simply that I could. That is, I was amazed that I could dig so deep so easily, see a path to improvement and effect change at a fundamental level. Excepting complexities with the Continuous Integration due to “changing the wheels on the car at 100km/h” (and one slip), it seems to have gone reasonably smoothly. That sense of mastery is seductive.”

Enabler: Turtles all the way

A Bryant developed Seaside in Pharo ancestor (while he knew ruby, python, scheme, C, objective-C) because he could manipulate the stack behind the back of developers. Seaside is based on stack on-demand reification.

A bit of fun with objects

1

1 class

1 class

> SmallInteger

1 class maxVal

> 1073741823

The largest small integer!

```
1 class maxVal

> 1073741823
```

(1 class maxVal + 1)

(1 class maxVal + 1)

> 1073741824

The smallest large integer!

(1 class maxVal + 1)

> 1073741824

(1 class maxVal + 1) class

(1 class maxVal + 1) class

> LargePositiveInteger

1000 factorial

>

1000 factorial

>
4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084869694048004799886101
9719605863166687299480855890132382966994459099742450408707375991882362772718873251977950595099527612087497546249
7043601418278094646496291056393887437886487337119181045825783647849977012476632889835955735432513185323958463075
5574091142624174743493475534286465766116677973966688202912073791438537195882498081268678383745597317461360853795
3452422158659320192809087829730843139284440328123155861103697680135730421616874760967587134831202547858932076716
9132448426236131412508780208000261683151027341827977704784635868170164365024153691398281264810213092761244896359
9287051149649754199093422215668325720808213331861168115536158365469840467089756029009505376164758477284218896796
4624494516076535340819890138544248798495995331910172335555660213945039973628075013783761530712776192684903435262
5200015888535147331611702103968175921510907788019393178114194545257223865541461062892187960223838971476088506276
8629671466746975629112340824392081601537808898939645182632436716167621791689097799119037540312746222899880051954
4441428201218736174599264295658174662830295557029902432415318161721046583203678690611726015878352075151628422554
0265170483304226143974286933061690897968482590125458327168226458066526769958652682272807075781391858178889652208
1643483448259932660433676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063
4463179746059468257310379008402443243846565724501440282188525247093519062092902313649327349756551395872055965422
8749774011413346962715422845862377387538230483865688976461927383814900140767310446640259899490222221765904339901
8860185665264850617997023561938970178600408118897299183110211712298459016419210688843871218556461249607987229085
1929681937238864261483965738229112312502418664935314397013742853192664987533721894069428143411852015801412334482
8015051399694290153483077644569099073152433278288269864602789864321139083506217095002597389863554277196742822248
7575867657523442202075736305694988250879689281627538488633969099598262809561214509948717012445164612603790293091
2088908694202851064018215439945715680594187274899809425474217358240106367740459574178516082923013535808184009699
6372524230560855903700624271243416909004153690105933983835777939410970027753472000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1000 factorial / 999 factorial

1000

Implementors of factorial [1]

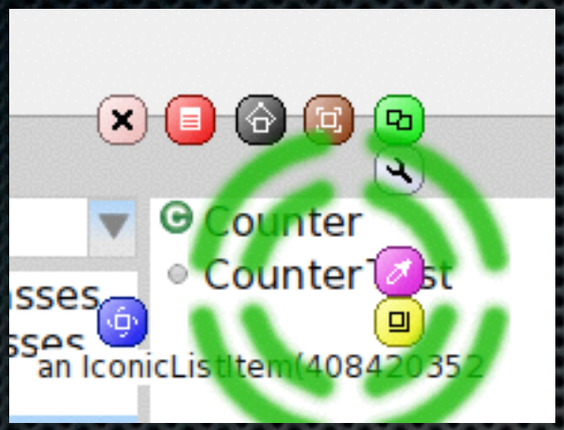
Integer (mathematical functions) factorial [Kernel]

Browse | Senders | Implementors | Version | Source

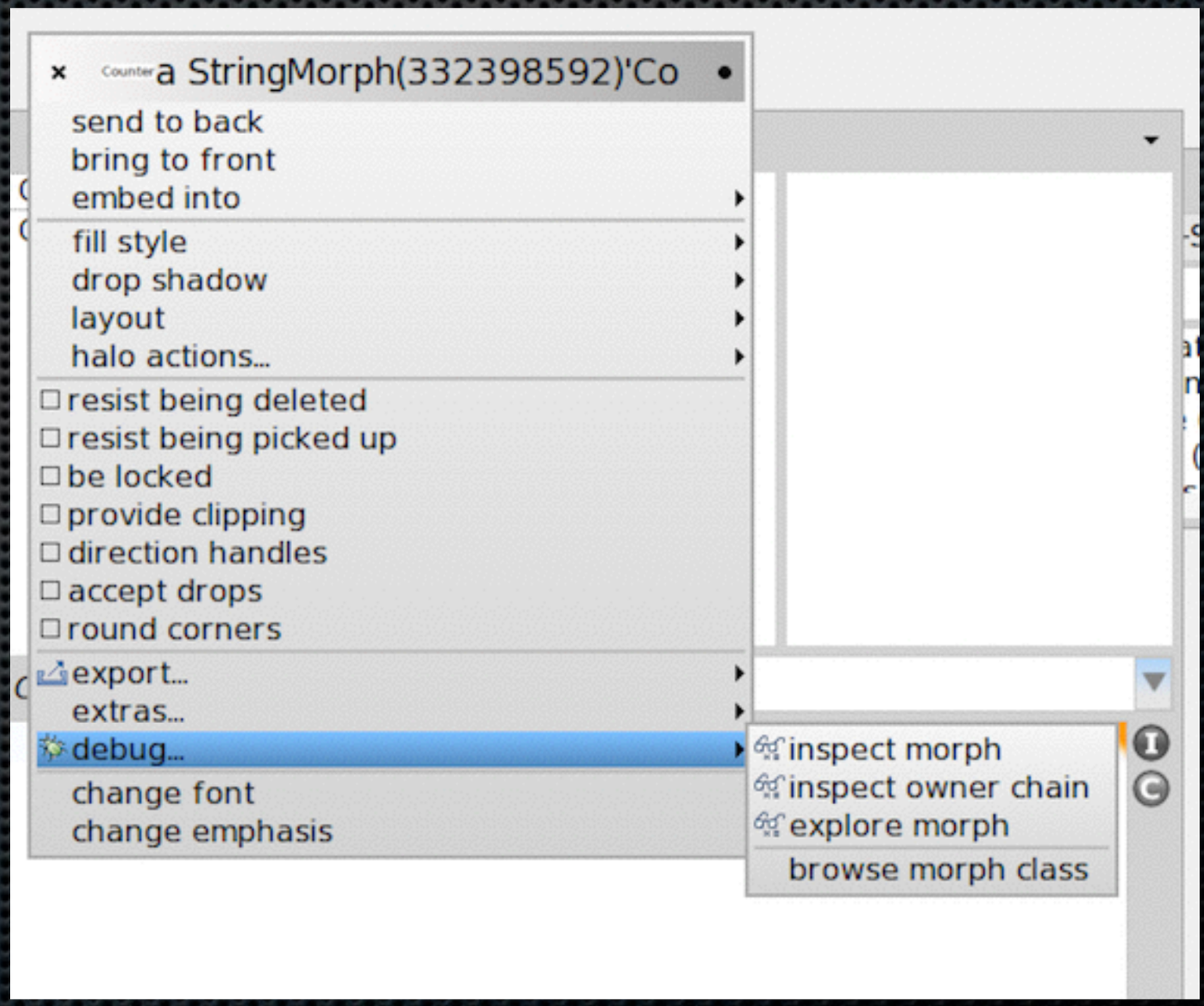
```
factorial
    "Answer the factorial of the receiver."

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

Finding information in the system…

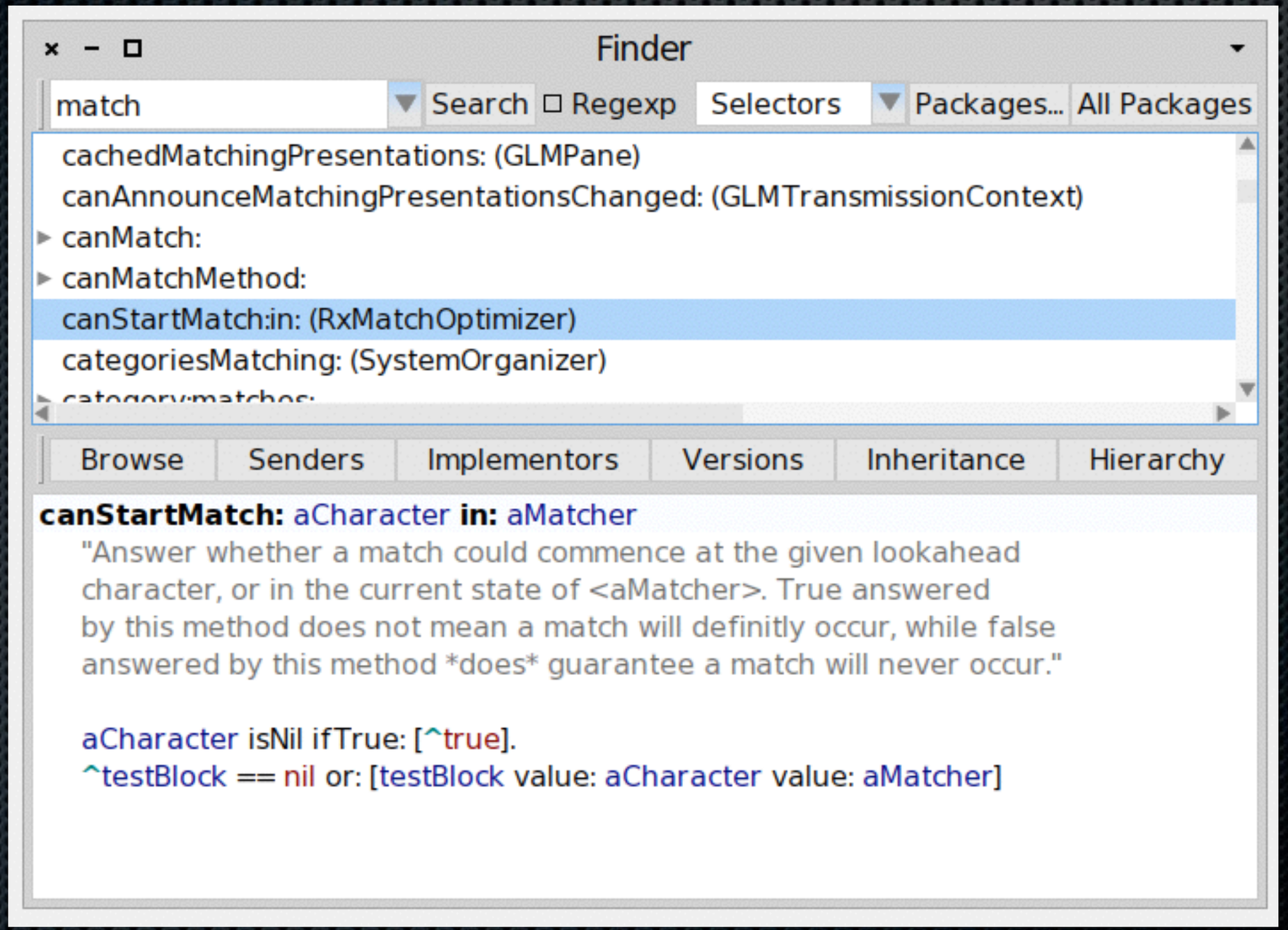
Click on it :): Cmd+shift +option



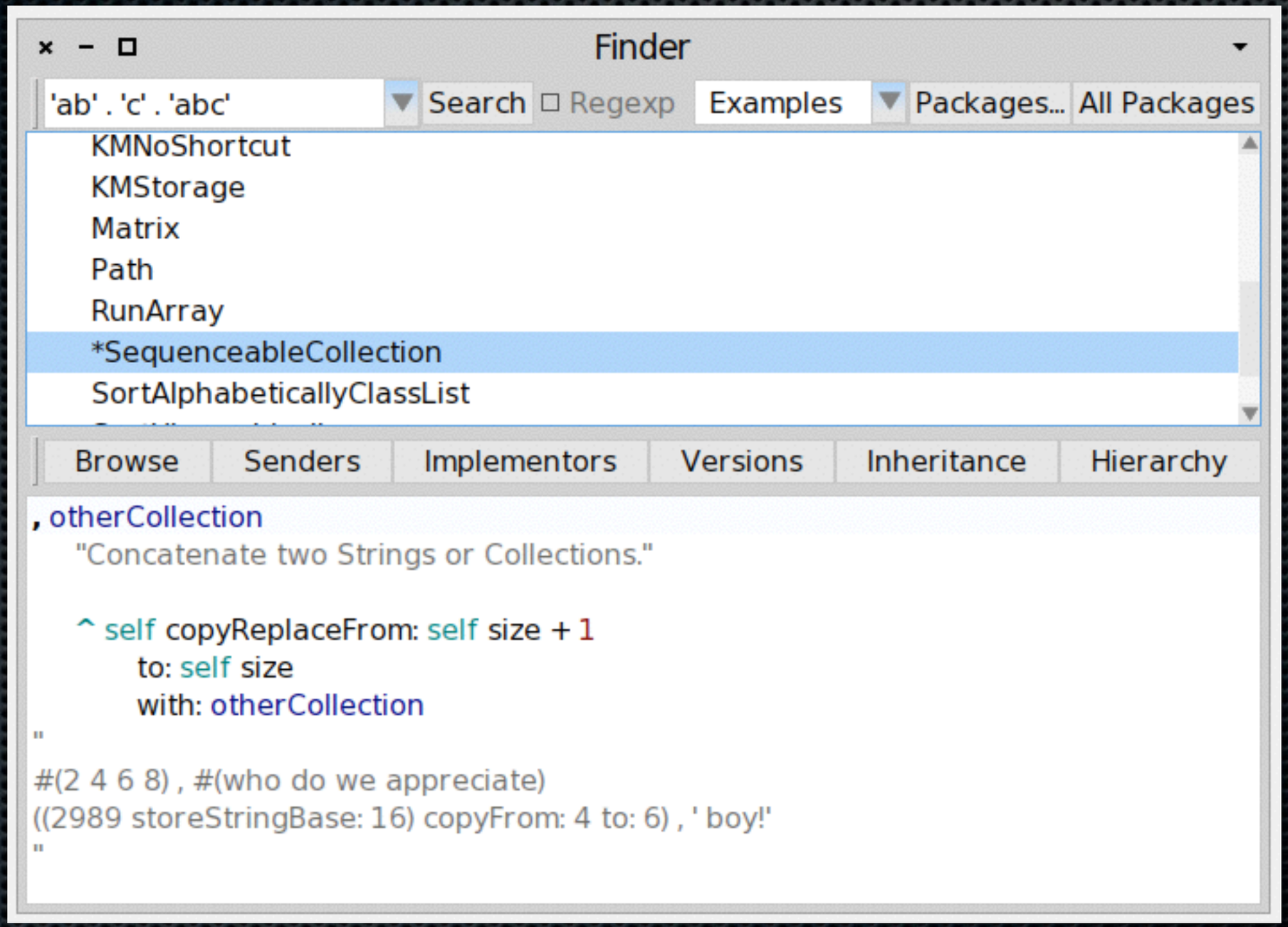
- Cmd-Shift+option



Finder :)


Finder
match
Search
Regexp
Selectors
Packages...
All Packages
cachedMatchingPresentations: (GLMPane)
canAnnounceMatchingPresentationsChanged: (GLMTransmissionContext)
canMatch:
canMatchMethod:
canStartMatch:in: (RxMatchOptimizer)
categoriesMatching: (SystemOrganizer)
Browse
Senders
Implementors
Versions
Inheritance
Hierarchy
canStartMatch: aCharacter in: aMatcher
"Answer whether a match could commence at the given lookahead
character, or in the current state of <aMatcher>. True answered
by this method does not mean a match will definitly occur, while false
answered by this method *does* guarantee a match will never occur."
aCharacter isNil ifTrue: [^true].
^testBlock == nil or: [testBlock value: aCharacter value: aMatcher]

Finder: *give* examples



Learning from the system...

Three questions

- Question 1: how to implement not
- Question 2: how to implement or or ifTrue:ifFalse:
- Question 3: Why these questions?

Booleans

&, |, not, or: ,and:, xor:

ifTrue:ifFalse:, ifFalse:ifTrue:

3 > 0

```
    ifTrue: ['positive']
    ifFalse: ['negative']
```

-> ‘positive’

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

How to implement not?

- false not -> true
- true not -> false

Without any IF statements!!!

Let the receiver decide!

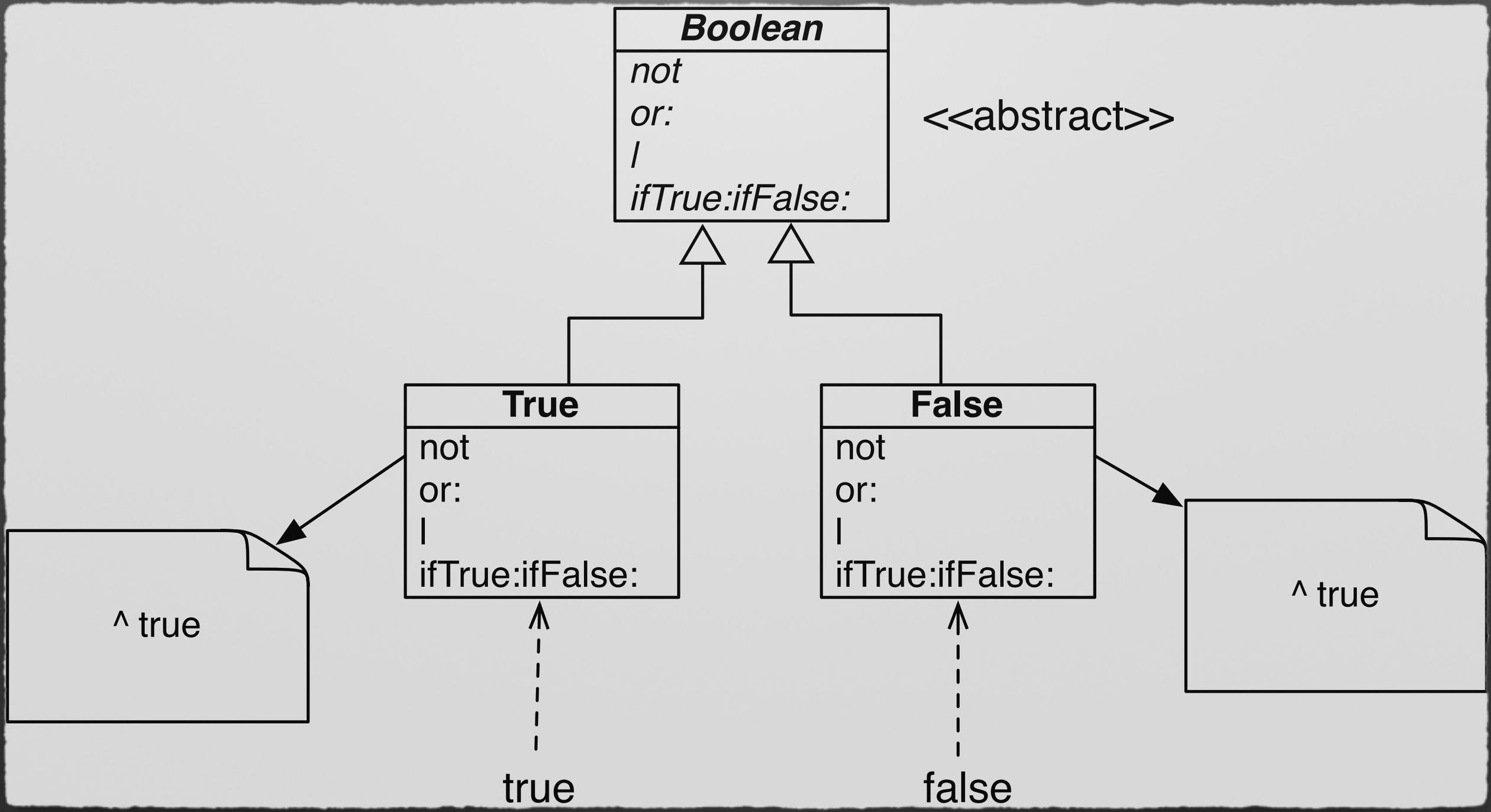
Not implementation

```
Boolean>>not

    self subclassResponsibility
```

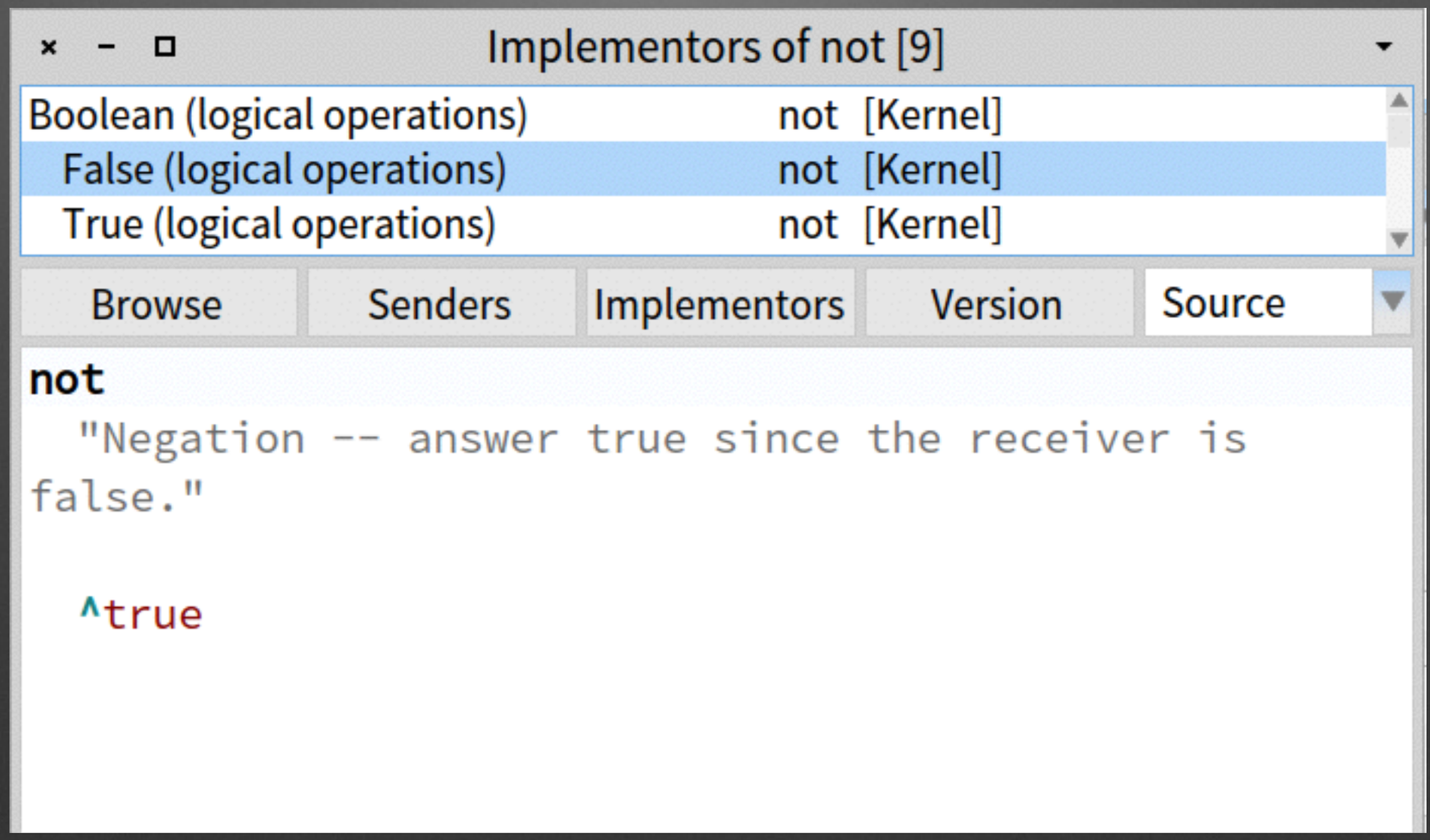
```
False>>not

  ^true
```

```
True>>not

  ^false
```

Not implementation



Check in the system



How to implement or?

true | true -> **true**

true | false -> **true**

true | anything -> **true**

false | **true** -> **true**

false | **false** -> **false**

false | **anything** -> **anything**

Implementors of | [12]

Boolean (logical operations)	[Kernel]
False (logical operations)	[Kernel]
True (logical operations)	[Kernel]

Browse | Senders | Implementors | Version | Source

```
| aBoolean
    "Evaluating disjunction (OR) -- answer with the argument, aBoolean."

    ^aBoolean
```

Implementors of | [12]

Boolean (logical operations)	[Kernel]
False (logical operations)	[Kernel]
True (logical operations)	[Kernel]

Browse | Senders | Implementors | Version | Source

```
| aBoolean
    "Evaluating disjunction (OR) -- answer true since the receiver is true."

    ^self
```

Ok so what?

- You will probably not implement another Boolean classes in your live
- So is it really that totally useless?

Message sends act as case statements

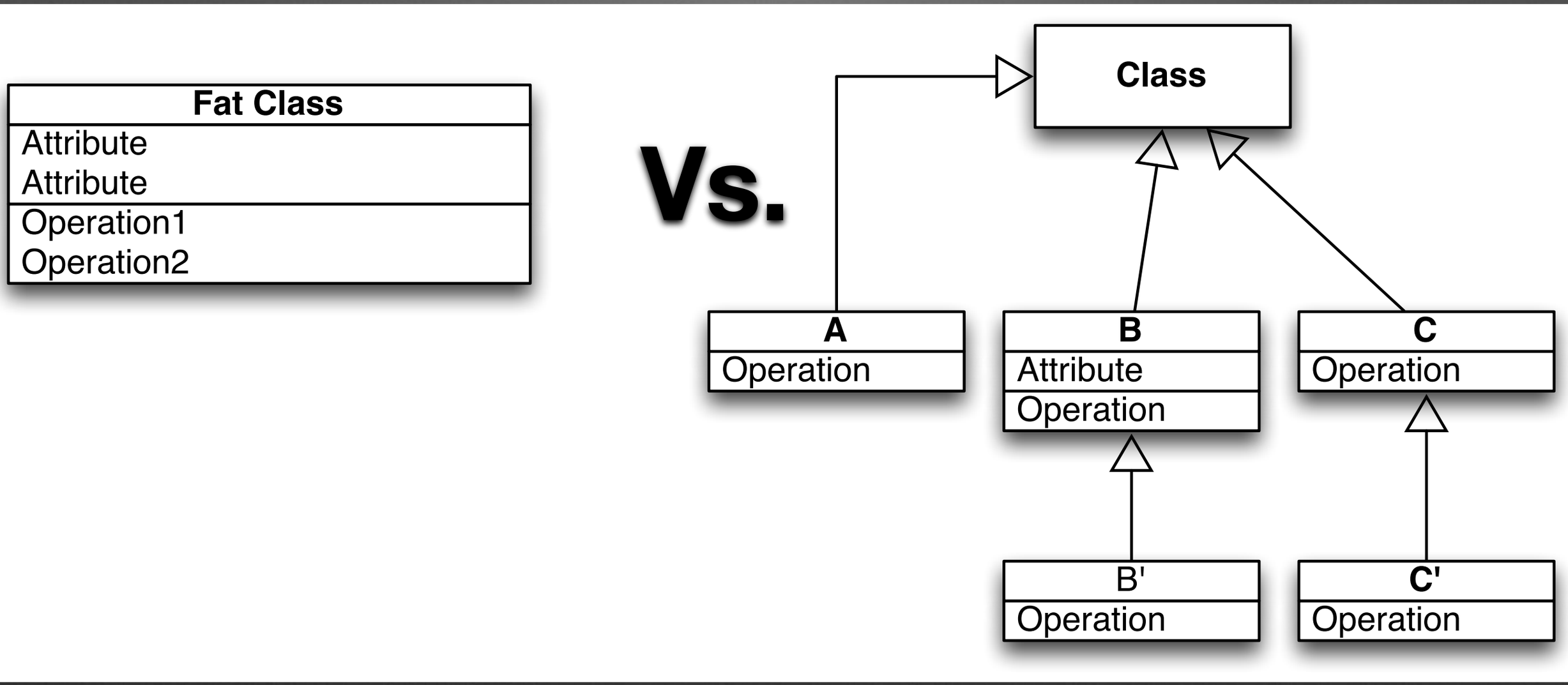
Message sends act as case statements

- But dynamic and open
- Can add a branch (define a subclass)
- Branches can be defined in different packages

OOP: the art of dispatching

Subclasses create your vocabulary

To be able to select we need classes



Pharo

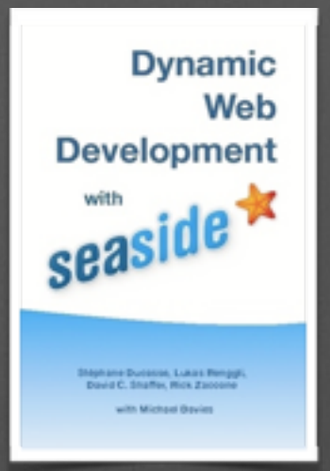
- http://www.pharo.org
- Pure object language
- Great community of active doers
- Powerful
- Elegant and fun to program
- Living system under your fingers

A powerful engine

to invent (y)our future

An ecosystem: teachers + researchers + companies + opensource community of individuals

Books



- Pharo by example http://www.pharobyexample.org
  - translated to french, merci! spanish, gracias!
  - translated to japanese, ありがとう!
- Dynamic web development with Seaside
- Deep into Pharo http://www.deepintopharo.org
- New 2015! Entreprise Pharo: a Web perspective
- Numerical Methods
- https://github.com/SquareBracketAssociates/NumericalMethods/releases
- New books in preparation
  - Updated Pharo by Example
  - Fun with Pharo

www.pharo.org

- Turtles all the way down
- Uniform and elegant
- Simple oo model
- But powerful to build really large applications
- Open system
- Learn from the system itself

Latest news

- a Mooc is under preparation
- Pharo Starter Kit is ready to be released
  - all resources in one USB