

Conceptes Avançats de Programació

Introducció a l'Smalltalk

basada en el curs del Prof. Oscar Nierstrasz (Universitat de Berna)

Dynamic Object Oriented Programming with Smalltalk

Jordi Delgado (jdelgado@cs.upc.edu)
Omega - SIIIS



Què és Smalltalk?



Llenguatge i Entorn Orientat a Objecte

Tot és un objecte: **Uniformitat**

Origen de moltes innovacions

Refactorització, **IDEs**, **MVC**, **xUnit** . . .

Dinàmic i completament interactiu

Millora els seus successors (!)



Què és Smalltalk?



- Llenguatge petit i uniforme
- Herència Simple
- Dinàmicament tipat
- Principi subjacent: TOT és un objecte
- Els objectes existeixen en una imatge persistent



Origen d'Smalltalk

Neix el 1972 a PARC Xerox



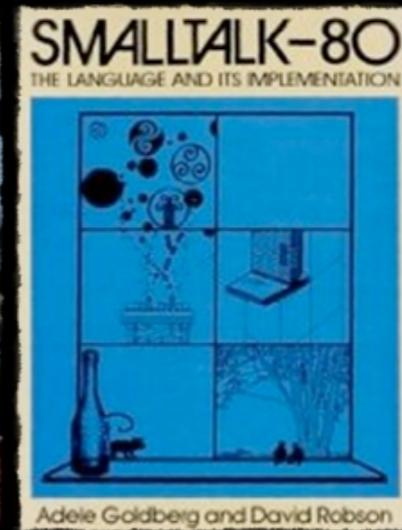
ALAN KAY



ADELE GOLDBERG



DANIEL INGALLS Jr.



SMALL TALK 80

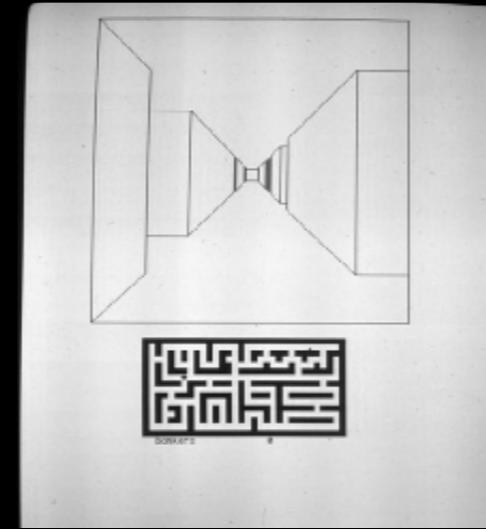
Primer llenguatge basat en entorn gràfic

Smalltalk-80 és l'estàndard



Origen d'Smalltalk

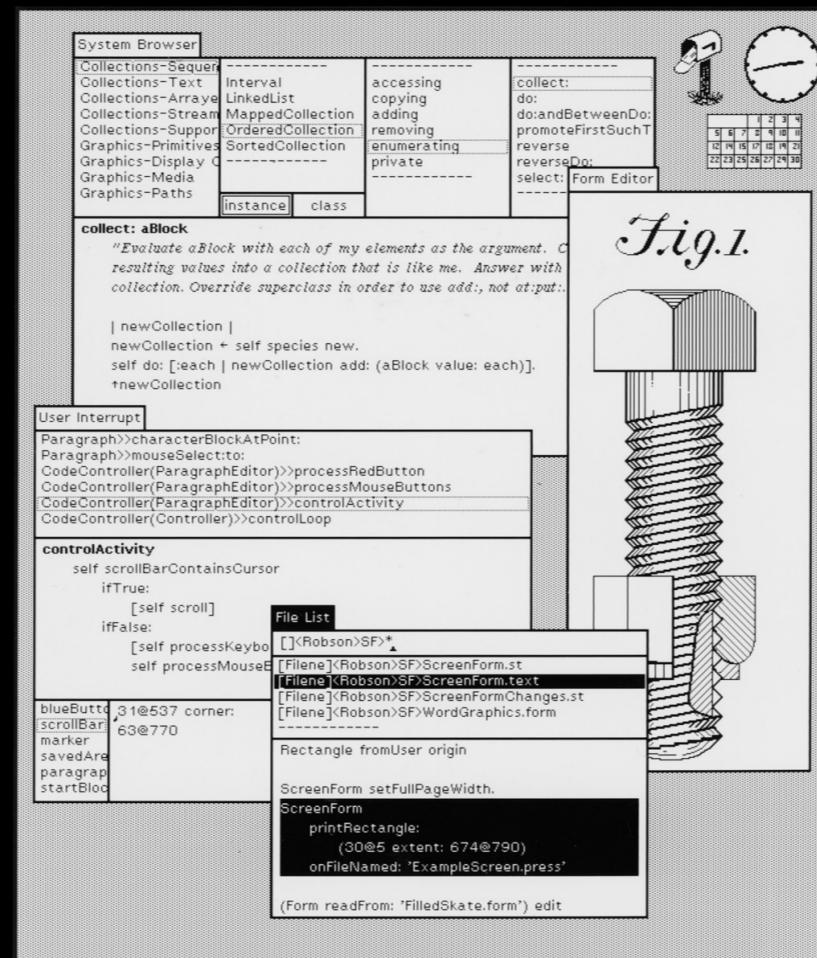
PARC Xerox





Origen d'Smalltalk

Anys 70

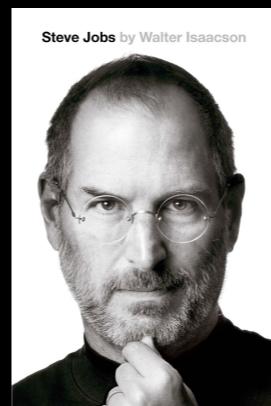


Smalltalk en Xerox Alto - Demo
https://youtu.be/9H79_kKzmFs?t=718



Origen d'Smalltalk

The Smalltalk demonstration showed three amazing features. One was how computers could be networked; the second was how object-oriented programming worked. But Jobs and his team paid little attention to these attributes because they were so amazed by the third feature, the graphical interface that was made possible by a bitmapped screen. “It was like a veil being lifted from my eyes,” Jobs recalled. “I could see what the future of computing was destined to be.”



Steve Jobs (p. 137)
Walter Isaacson
Simon & Schuster, 2011

CAP: Introducció a l'Smalltalk

Agost 1981





Actualment hi ha moltes implementacions modernes d'Smalltalk.
Veieu en.wikipedia.org/wiki/Smalltalk per a més informació.

Open Source:

→ **Pharo (<http://pharo.org>)**

- Squeak (<http://squeak.org>)
- Amber (<http://amber-lang.net/>)

- GNU Smalltalk (<http://smalltalk.gnu.org/>)

Empreses:

→ **Cincom Smalltalk (<http://www.cincomsmalltalk.com/>)**

- VA Smalltalk (<http://www.instantiations.com/>)
- Gemstone (<http://www.gemstone.com/>)



<http://pharo.org>

Farem servir la versió de Pharo 6.1, que és una versió molt semblant a Smalltalk.

La versió 3.0 és la darrera versió en que Pharo es pot considerar una implementació d'Smalltalk.

A partir de la versió 7.0 Pharo és ja un llenguatge semàcticament diferent d'Smalltalk

**Si voleu provar la versió 3.0 la podeu trobar a:
files.pharo.org/platform/Pharo3.0-portable.zip**



<http://pharo.org>

Instal·lació de Pharo 6.1 (a dia 17/8/19) a Linux:

1.- Baixar màquina virtual

```
curl -L https://get.pharo.org/vm61 | bash
```

i la imatge

```
curl -L https://get.pharo.org/61 | bash
```

2.- Renombrar: Pharo.* ==> Pharo.6.1.*

3.- Executar: ./pharo-ui Pharo.6.1.image (el path fins a pharo-ui no ha de contenir espais!)

4.- Fer el que diu a <https://github.com/pharo-vcs/iceberg>

5.- Per veure si el punt anterior s'ha fet correctament, seguir les instruccions de <https://github.com/pharo-vcs/iceberg/wiki/Tutorial> (ho podeu fer més tard amb l'exemple de la Pasta)

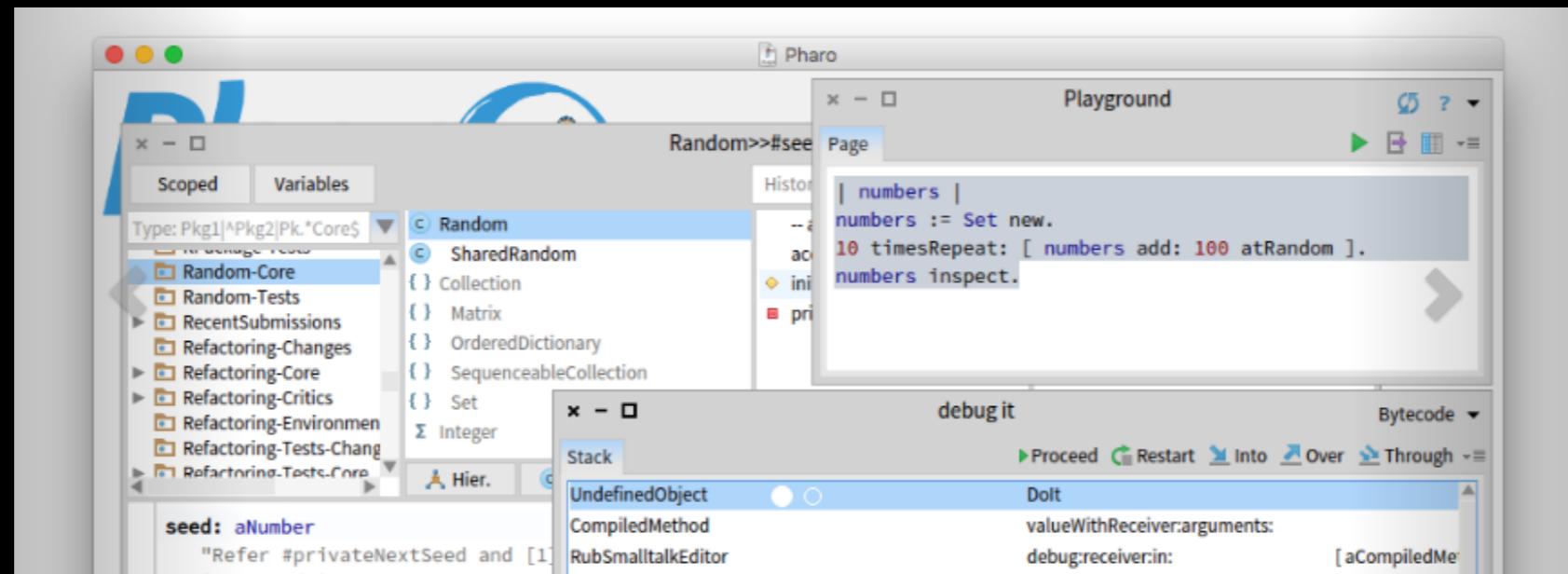
CAP: Introducció a l'Smalltalk



<http://pharo.org>

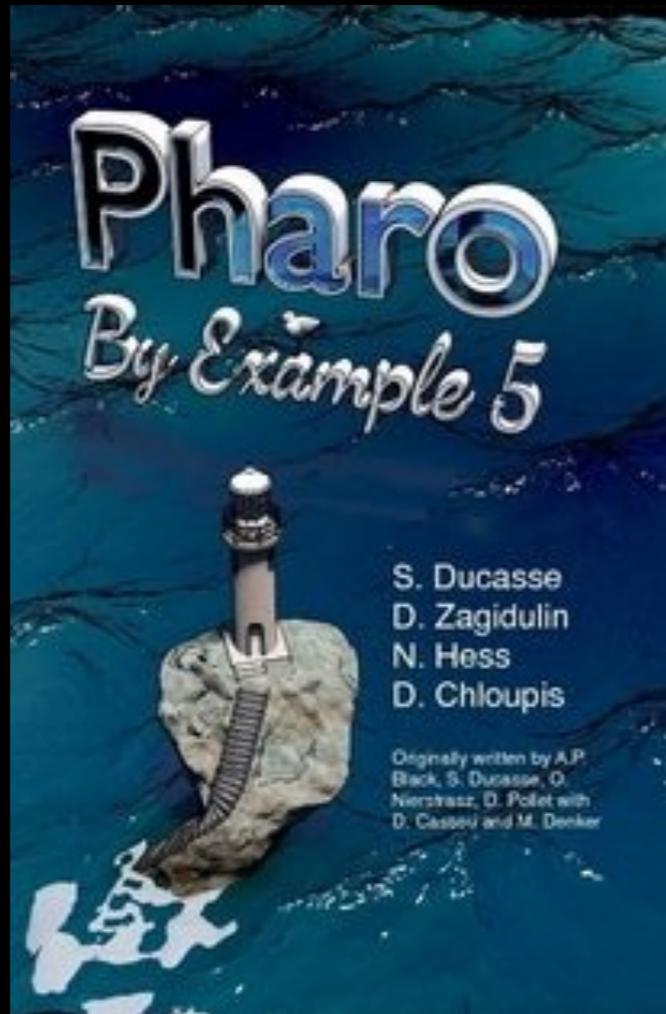
Les darreres versions de Pharo són MOLT interessants, però en aspectes importants han anat més enllà d'Smalltalk.

Per aprendre'n més, vegeu el Pharo MOOC



<http://mooc.pharo.org/>

Versió per a Pharo 5.0



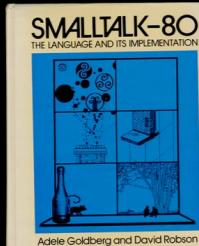
Pharo by example 5

S.Ducasse, D.Zagidulin, N.Hess & D.Chloupis

Darrera versió del llibre (2017):

<http://files.pharo.org/books-pdfs/updated-pharo-by-example/2017-01-14-UpdatedPharoByExample.pdf>

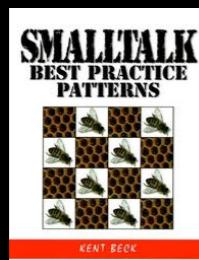
Altres llibres recomanats:



Adele Goldberg, David Robson

Smalltalk-80, The Language and its Implementation

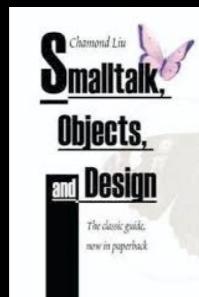
Longman Higher Education, 1983



Kent Beck

Smalltalk Best Practice Patterns

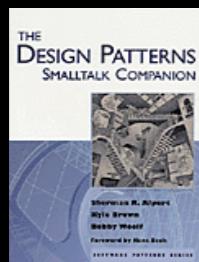
Prentice Hall, 1996



Chamond Liu

Smalltalk, Objects and Design

iUniverse, 2000

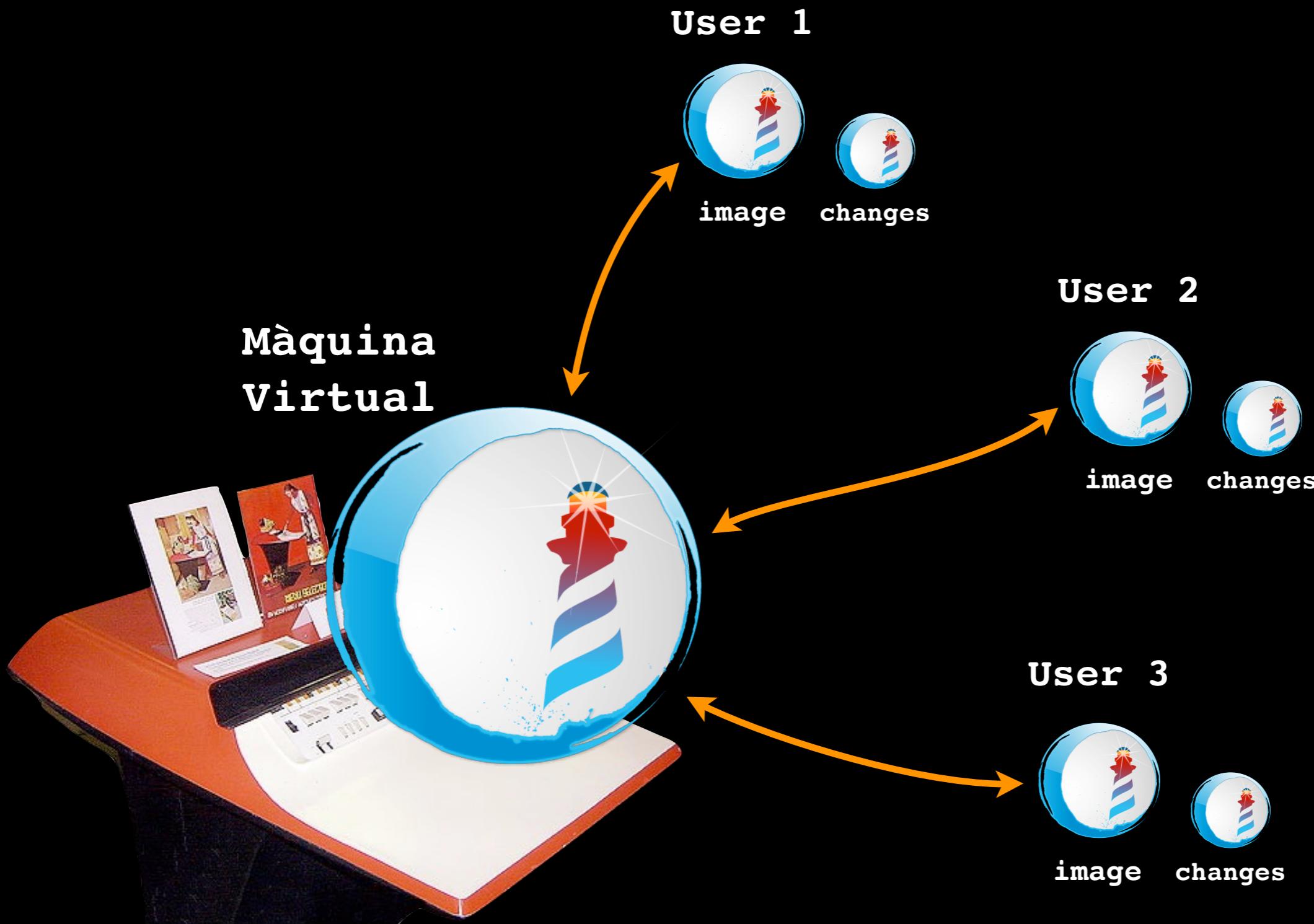


S. Alpert, K. Brown, B. Wolf

The Design Patterns Smalltalk Companion

Addison Wesley, 1998

CAP: Introducció a l'Smalltalk

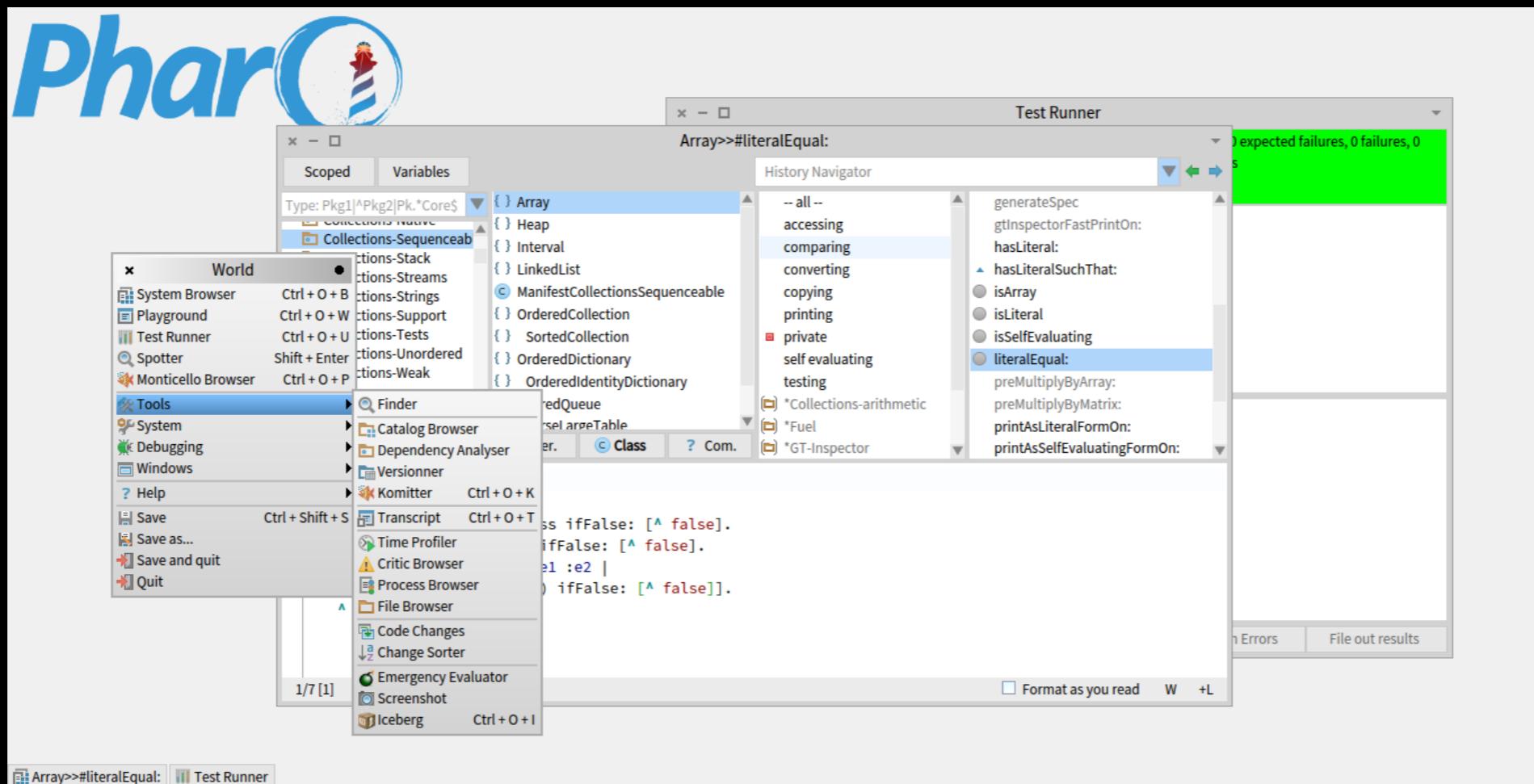


Objectes a Smalltalk

- **Tot** és un objecte. Absolutament tot.
- Res passa si no és per **pas de missatges**
(hi ha, però, una excepció: l'assignació).
- L'estat dels objectes (atributs, camps,...) és **privat**
- Els mètodes són **públics**
(els podem considerar privats per convenció)
- Les variables contenen **referències** (tipat dinàmic).
(els objectes no referenciats són eliminats, i.e. *garbage collection*)
- Herència **simple**.

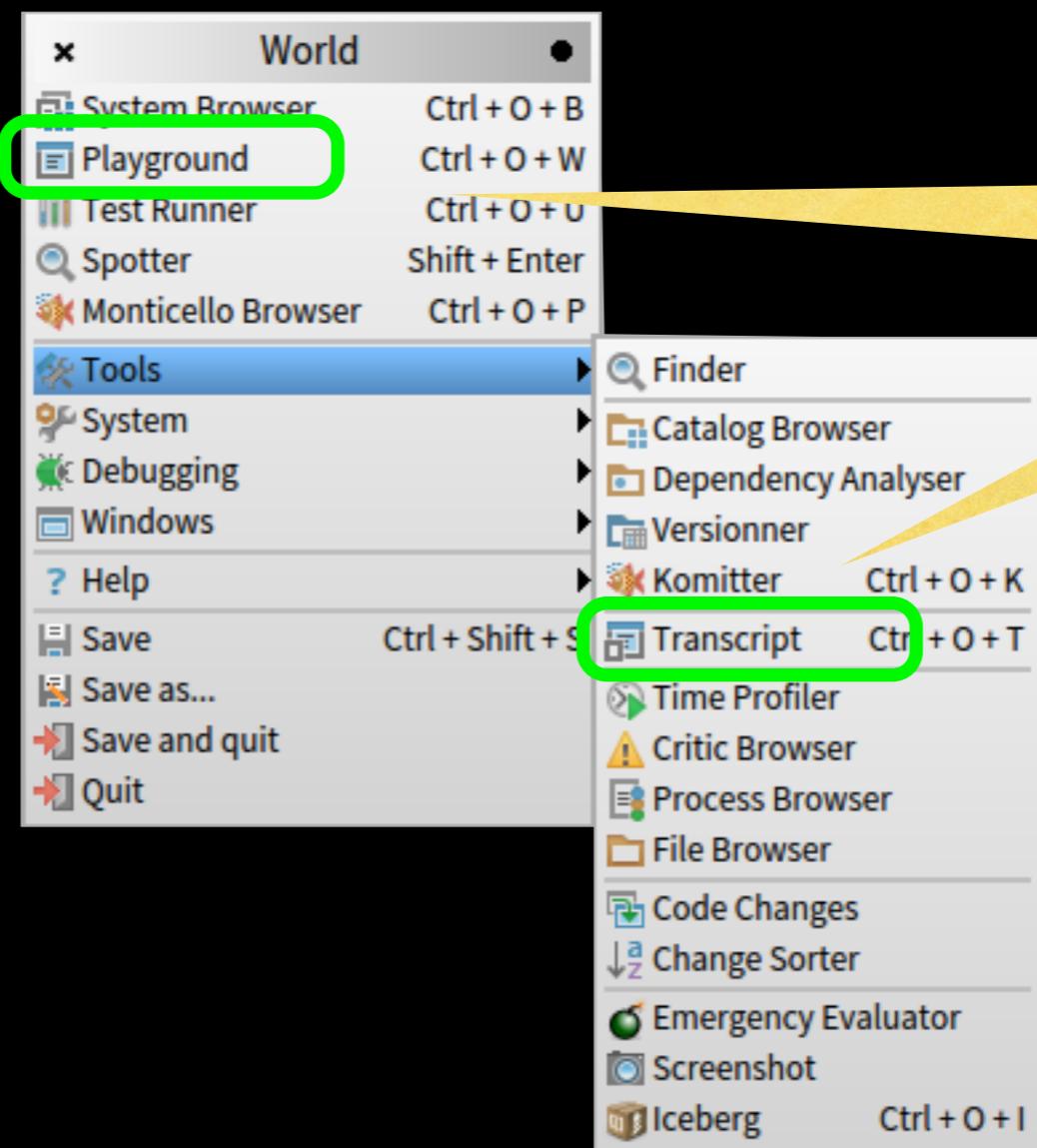
L'entorn

Un món on ho tenim tot a l'abast, fins i tot el *codi font del sistema* i de la *màquina virtual* (que, naturalment, està escrita en Smalltalk)



L'entorn

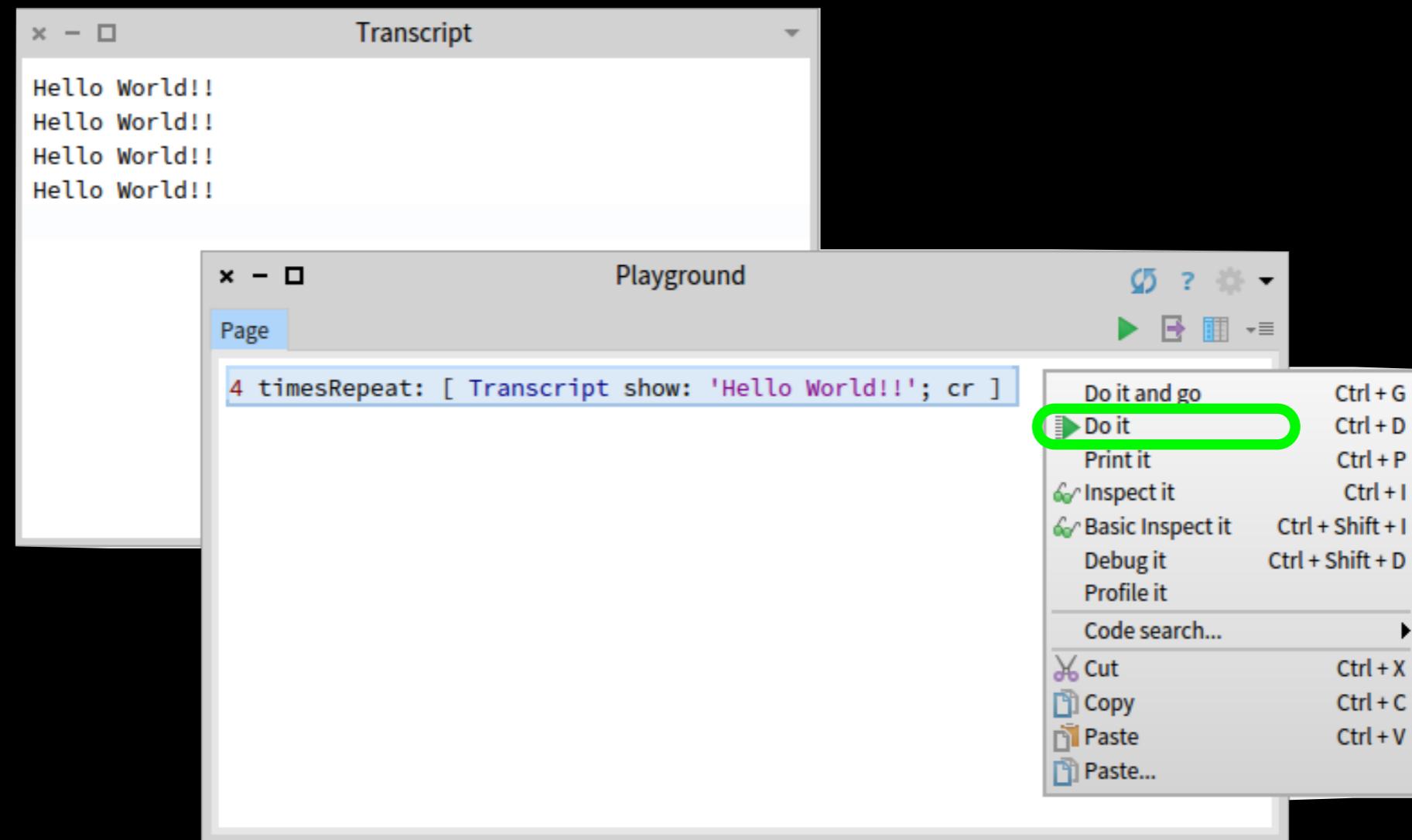
Menú **World** per començar a interactuar amb el món



*Transcript i
Playground seran
especialment
importants*

L'entorn

El **Transcript** és una mena d'estàndard *output* i el **Playground** és on provem el codi



L'entorn

Usualment treballarem dins del **Browser** . . .

The screenshot shows the Smalltalk Browser interface. The top title bar reads "OrderedCollection>>#join:". The left pane is a "History Navigator" showing a list of methods and objects. The "join:" method is selected. The right pane shows the source code for the "join:" method. The bottom pane displays the actual implementation of the method.

Type: Pkg1|^Pkg2|Pk.*Core\$

History Navigator

join:

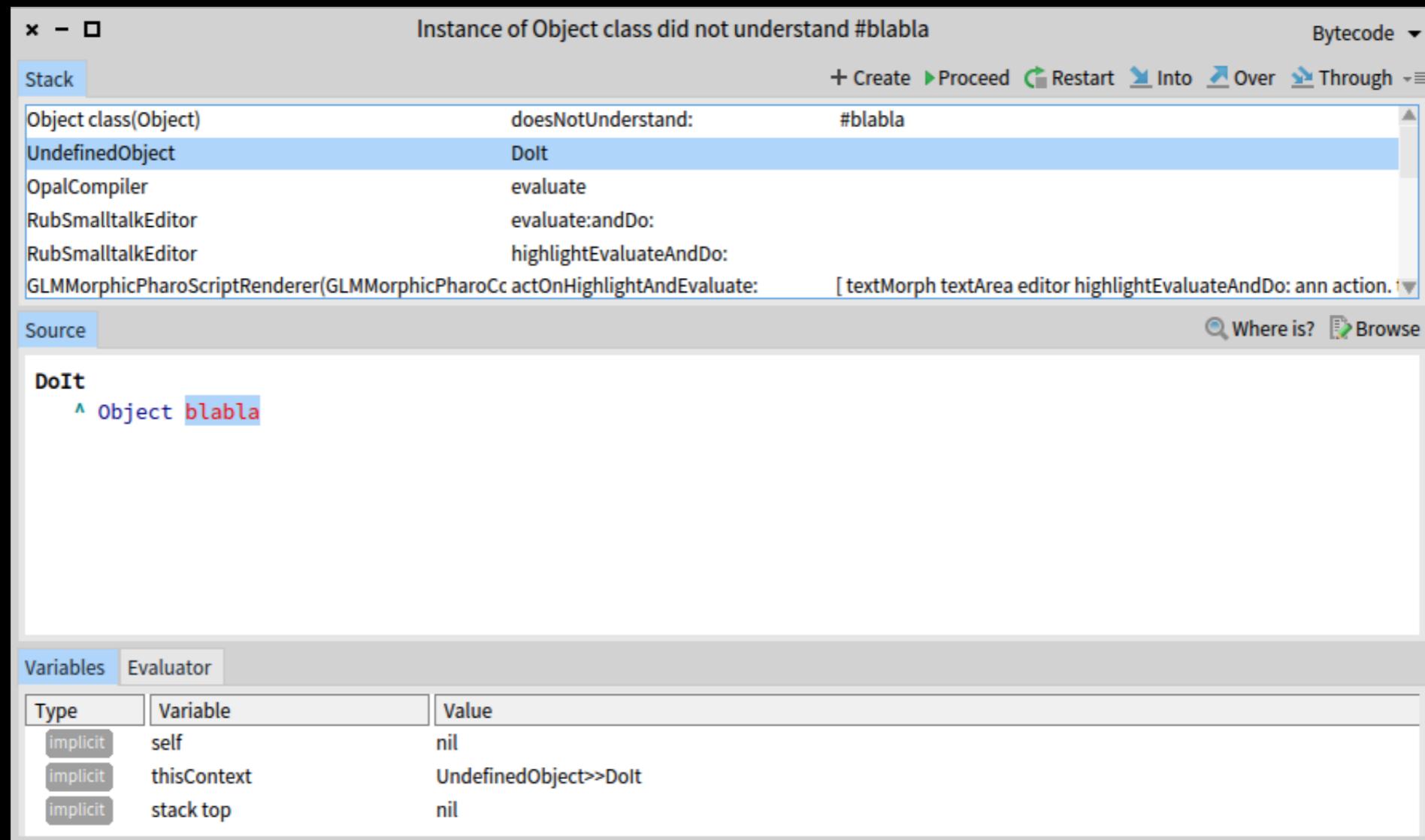
```
join: aCollection
    | result |
    result := self class new.
    aCollection
        do: [:each | each appendTo: result]
        separatedBy: [self appendTo: result].
    ^ result
```

1/7 [1]

Format as you read W +L

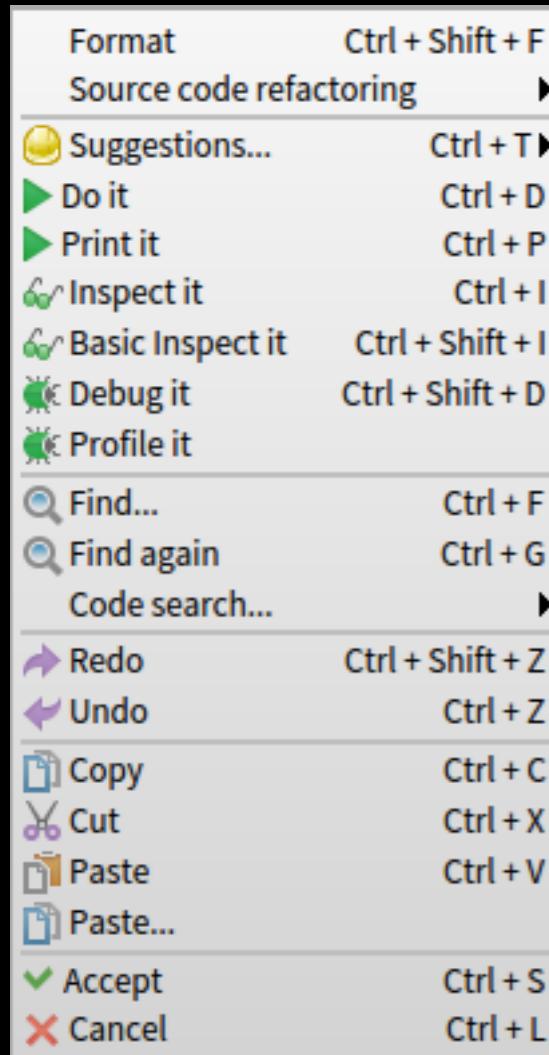
L'entorn

. . . i del **debugger**



L'entorn: Menus contextuais

accept - do it - print it - inspect it



accept: Compila un mètode o definició de classe

do it: Avaluà una expressió

print it: Avaluà una expressió i escriu el resultat

inspect it: Avaluà una expressió i inspecciona el resultat

La sintàxi d'Smalltalk és *molt* simple

Elements de la sintàxi:

^	retorn
“ ”	comentari
#	símbol o array
' '	string
[]	bloc
.	separador d'instruccions
;	cascada de missatges
	declaració de variables locals/bloc
:=	assignació
\$ _	caràcter
:	final de selector de nom
<primitive: >	crides a la MV

La sintàxi d'Smalltalk és *molt simple*

Exemples:

comentari

"això és un **comentari**"

caràcter

\$c \$g \$h \$# \$@

string

'això és una **string**'

símbol

#mac #linux #anonymous

array

#{1 2 3 (1 3) \$a 'pozzí'}

array dinàmic

{ 1+2 . 3/4 }

(no Smalltalk-80)

enter

2 2r1011

real

1.4 6.03e-21 2.5e4

booleà

true false

pseudo variable

self super

A Smalltalk tot passa per enviament de missatges

Molts llenguatges defineixen els operadors bàsics i les estructures de control com a *paraules clau* del llenguatge.

A Smalltalk només hi ha missatges

Exemples:

10 bitshift: 2 → Missatge #bitshift: enviat a l'objecte (número) 10, té com a paràmetre un altre número - el 2

(x > 1) ifTrue: [Transcript show: 'més gran que 1'; cr] → Missatge #ifTrue: enviat a un booleà (resultat d'avaluar ($x > 1$)) amb un bloc com a paràmetre.

A Smalltalk tot passa per enviament de missatges

Molts llenguatges defineixen els operadors bàsics i les estructures de control com a *paraules clau* del llenguatge.

A Smalltalk només hi ha missatges

Exemples:

```
#(a b c) do: [:each | each asString traceCr ] ➔
```

Missatge `#do:` enviat a un array que escriu al `Transcript` els seus elements. Fixem-nos que `asString` és un missatge que s'envia a cada element per transformar-lo en una String

```
1 to: 10 do: [:each | each asString traceCr ] ➔
```

Missatge `#to:do:` enviat a un objecte (número) 1 amb dos paràmetres, un enter (el 10) i un bloc. Aquí també fem servir `asString`

A Smalltalk tot passa per enviament de missatges

Tota expressió en Smalltalk és un enviament de missatge

Hi ha **tres tipus** de missatges:

Missatges *Unaris*: Transcript cr
5 factorial

Missatges *Binars*: 3 + 4

Missatges de *Paraula Clau*: Transcript show: 'Hola Món'
(*keyword messages*) 5 raisedTo: 10 modulo: 5

Precedència: (...) > **unaris** > **binaris** > **paraula clau**

- 1.- S'avalua d'esquerra a dreta
- 2.- Missatges unaris tenen la precedència més alta
- 3.- Després els missatges binaris
- 4.- Finalment els missatges de paraula clau
- 5.- Sempre podem posar parèntesi per forçar l'ordre d'avaluació

Exemple: 2 **raisedTo:** 1 + 3 **factorial**
és equivalent a

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

Precedència: (...) > **unaris** > **binaris** > **paraula clau**

- 1.- S'avalua d'esquerra a dreta
- 2.- Missatges unaris tenen la precedència més alta
- 3.- Després els missatges binaris
- 4.- Finalment els missatges de paraula clau
- 5.- Sempre podem posar parèntesi per forçar l'ordre d'avaluació

Atenció!! **1 + 2 * 3** és equivalent a **(1 + 2) * 3**
Ja que **+** i ***** són missatges binaris, per tant tenen la mateixa
precedència, aleshores s'avaluen d'esquerra a dreta

Missatges Binaris - Sintaxi

unReceptor unSelector unArgument

On **unSelector** pot ser **un** o **dos** caràcters dels següents:

+ - / \ * ~ < > = @ % | & ! ? , \$
(el segon caràcter no pot ser \$)

Exemples:

2 + 3 - 5
5 >= 7
6 = 7
'hola' , '' , 'món'
(3 @ 4) + (1 @ 2)
2 << 5

Separadors de sentències

Punts (.) - Separen sentències:

Transcript show: 'Hola'.

Transcript show: ' '.

Transcript show: 'món' "aquí no cal"

Punt i coma (;) - Enviament d'una **cascada** de missatges al **mateix** objecte.

Transcript show: 'Hola'; show: ' '; **show: 'món' .**

Més sintaxi . . .

Variables:

- Declaració: Es llisten entre barres verticals, | **x y z** |
- Assignació: Té la forma **var := expressió**
NO es fa per pas de missatges.

Mètodes:

- Retornen un valor **sempre** (**self** per defecte).
- S'utilitza ^ per explicitar el valor de retorn
- El selector d'un mètode és un símbol: **#collect:**

Blocs (*closures*):

- Es delimiten entre []
- Un bloc retorna el valor de la darrera expressió
avaluada dins del bloc
- Els paràmetres dels blocs es delimiten amb :par |

Exemple:

Definim un mètode per trobar el màxim de dos nombres:

```
max: unNombre
    ^ self < unNombre
        ifTrue: [unNombre]
        ifFalse: [self]
```

Exemple:

Definim un mètode per trobar el màxim de dos nombres:

Nom del
mètode(selector)
de paraula clau

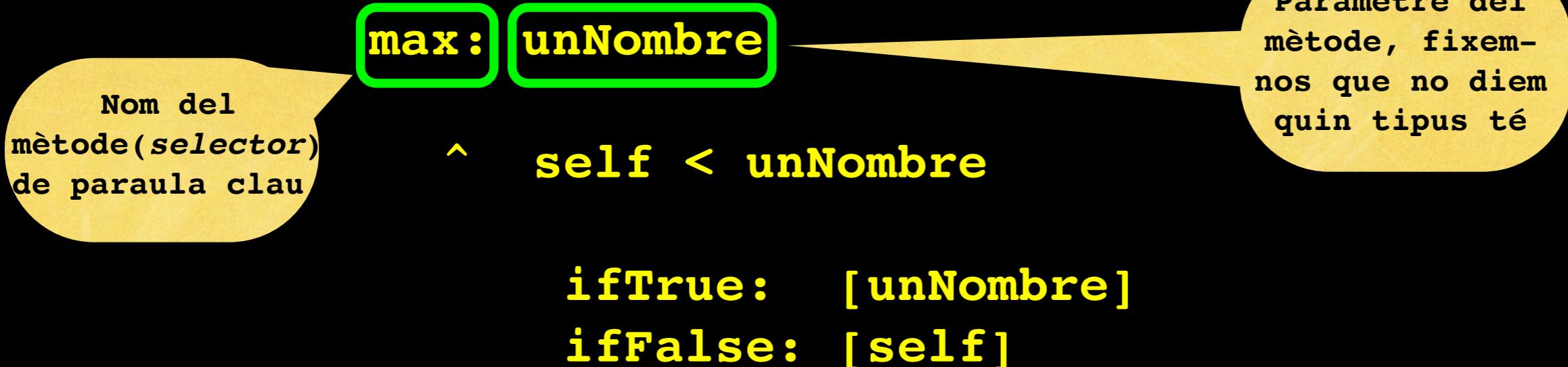
max: **unNombre**

^ **self < unNombre**

ifTrue: [**unNombre**]
ifFalse: [**self**]

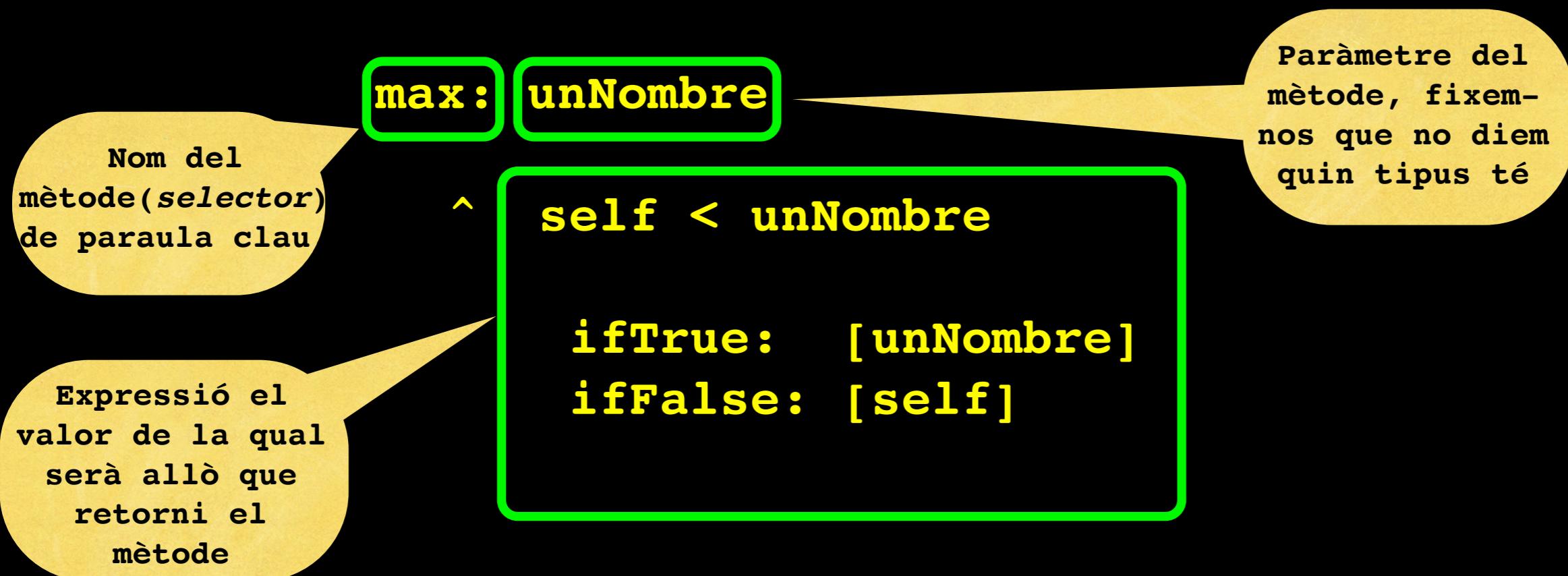
Exemple:

Definim un mètode per trobar el màxim de dos nombres:



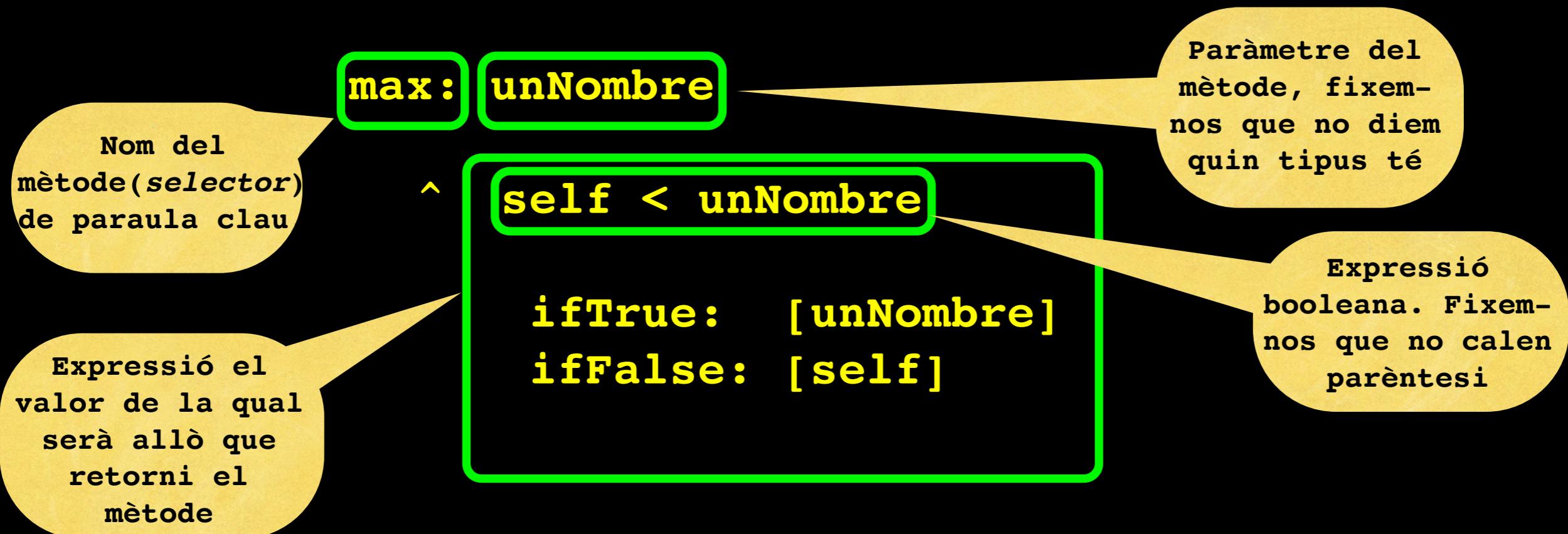
Exemple:

Definim un mètode per trobar el màxim de dos nombres:



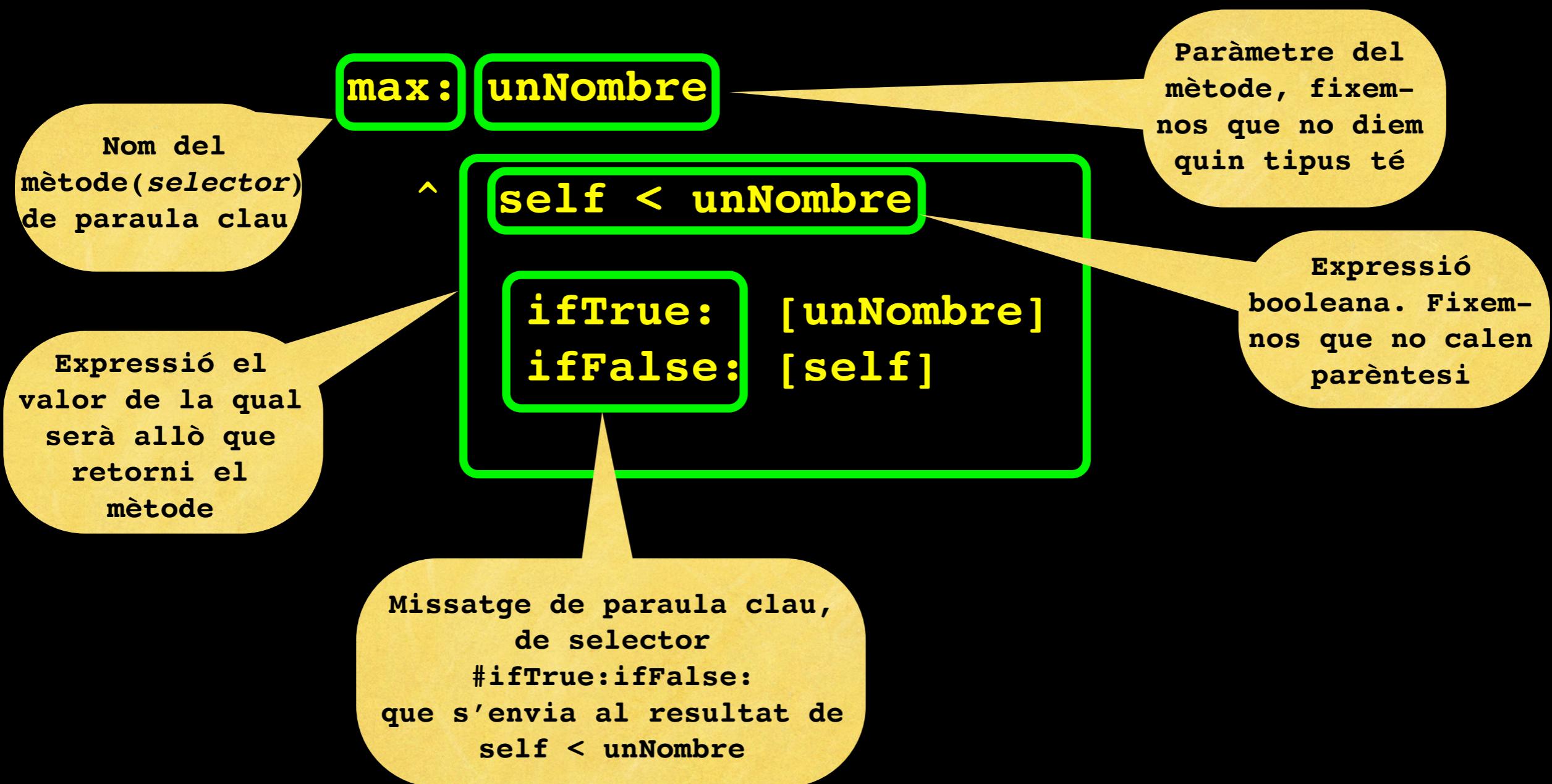
Exemple:

Definim un mètode per trobar el màxim de dos nombres:



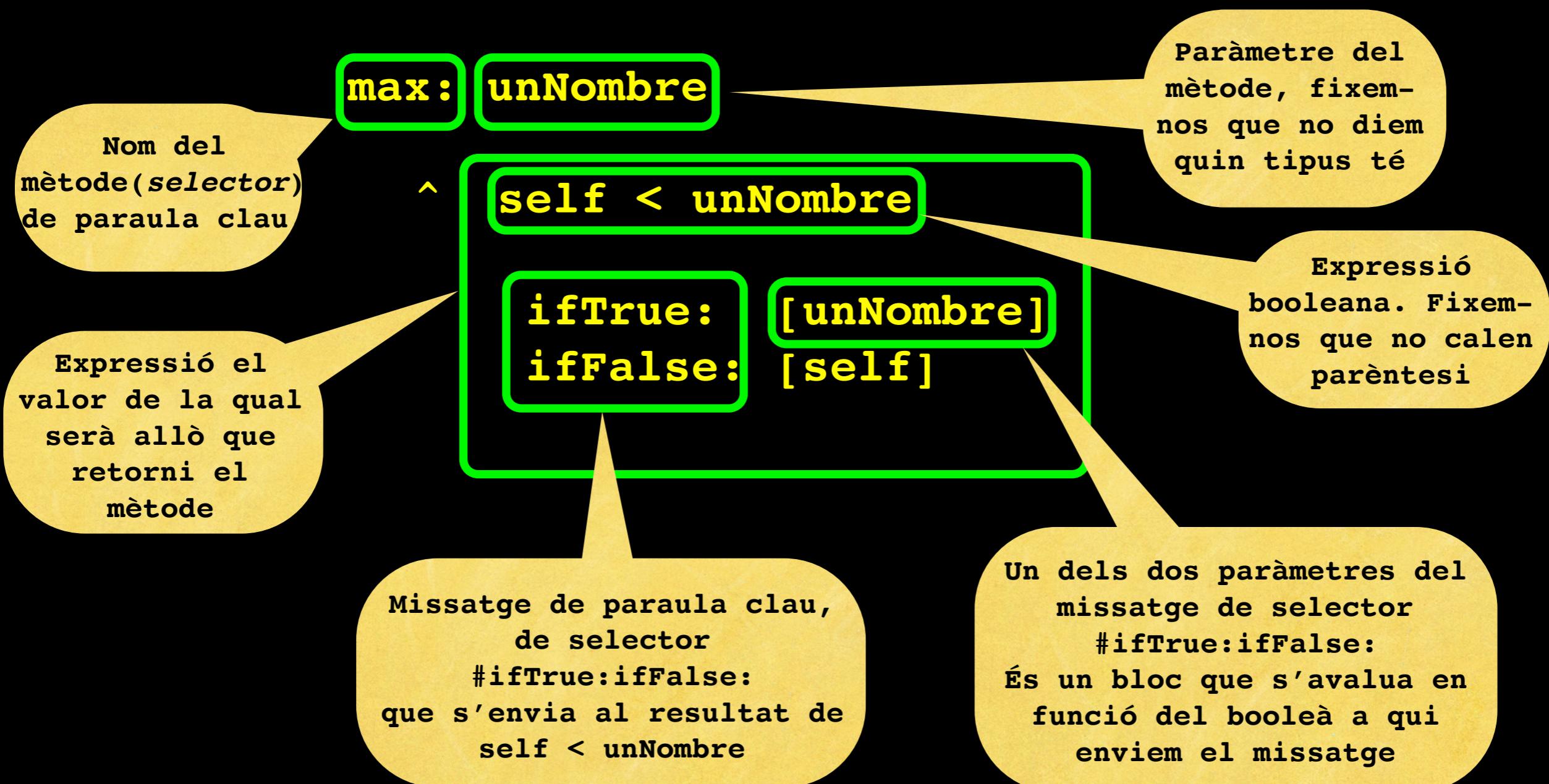
Exemple:

Definim un mètode per trobar el màxim de dos nombres:



Exemple:

Definim un mètode per trobar el màxim de dos nombres:



Exemple:

També podríem haver fet el retorn explícit, des dels blocs

max: unNombre

self < unNombre

ifTrue: [^unNombre]

ifFalse: [^self]

Totes les estructures de control són enviaments de missatges

Exemples d'iteracions; totes fan el mateix:

```
|n|      "Programant C en Smalltalk :)"
```

```
n := 1.
```

```
[ n <= 10 ] whileTrue:
```

```
  [ Transcript show: n asString; cr.
```

```
    n := n + 1 ]
```

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

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

Exemple:

OrderedCollection >> collect:

```
collect: aBlock
    "Evaluate aBlock with each of my elements as the argument. Collect the
     resulting values into a collection that is like me. Answer the new
     collection. Override superclass in order to use addLast:, not at:put:."
    | newCollection |
    newCollection := self species new: self size.
    firstIndex to: lastIndex do:
        [:index |
        newCollection addLast: (aBlock value: (array at: index))].
    ^ newCollection
```

Crear Objectes

Hi ha essencialment dues maneres de crear objectes:

Mètodes de classe: Usualment, però no necessàriament, **new**

Exemples: **OrderedCollection new**

Array with: 1 with: 2

Mètodes constructors: Missatges a objectes que crean instàncies d'altres classes.

Exemples: **1 @ 2** “**un punt**”

1 / 2 “**una fracció**”

Crear Classes

Enviant missatges a altres classes!

```
Magnitude subclass: #Number  
instanceVariableNames: ''  
classVariableNames: ''  
package: 'Kernel-Numbers'
```

```
Object variableSubclass: #BlockClosure  
instanceVariableNames: 'outerContext startpc numArgs'  
classVariableNames: ''  
package: 'Kernel-Methods'|
```

```
SequenceableCollection subclass: #OrderedCollection  
instanceVariableNames: 'array firstIndex lastIndex'  
classVariableNames: ''  
package: 'Collections-Sequenceable'
```

#subclass:instanceVariableNames:classVariableNames:package:
és un missatge amb 4 paràmetres!

Exercici: Endevinar nombres

Fer un programa que generi a l'atzar un nombre entre 1 i 100 i demani a l'usuari que l'endevini. A cada resposta de l'usuari escriurem pistes com 'més gran' o 'més petit' per ajudar-lo. Al final, s'escriurà el nombre de vegades que l'usuari ho ha intentat.

Ajuts:

- Demanar *inputs*:
UIManager >> #request:initialAnswer:title:
(atenció! retorna una **String**)
Enviarem aquest missatge a l'objecte **UIManager default**
- Nombres enters a l'atzar entre 1 i 100: **100 atRandom**
- Escriurem el programa en el **Playground** i els resultats en el **Transcript**

Solució

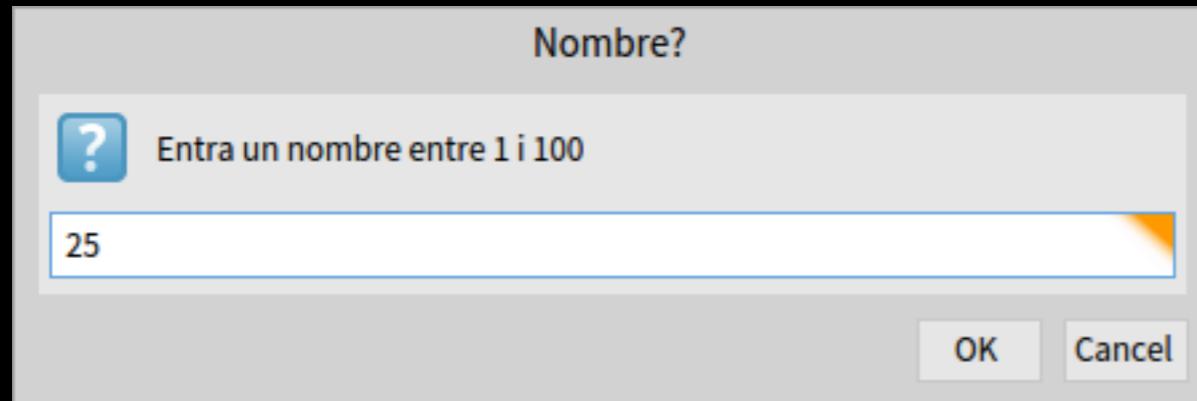
```
| nombreAEndevinar nombreProposat intents |

nombreAEndevinar := 100 atRandom.
intents := 0.
nombreProposat := (UIManager default request: 'Entra un nombre entre 1 i 100'
                     initialAnswer: 0
                     title: 'Nombre?') asInteger.

[ nombreAEndevinar = nombreProposat ]
  whileFalse: [
    Transcript show: ( nombreProposat > nombreAEndevinar
                      ifTrue: [ 'més petit' ]
                      ifFalse: [ 'més gran' ] ); cr.
    nombreProposat := (UIManager default
                      request: 'Entra un nombre entre 1 i 100'
                      initialAnswer: 0
                      title: 'Nombre?') asInteger.
    intents := intents + 1 ].

Transcript cr;
show: 'Enhorabona! Ho has aconseguit després de ' , intents asString , ' intents!';
cr.
```

CAP: Introducció a l'Smalltalk



```
x - □ Transcript
més gran
més gran
més gran
més gran
més gran
més petit

Enhorabona! Ho has aconseguit després de 6 intents!
```

The screenshot shows a Smalltalk playground window titled "Playground". The code in the window is as follows:

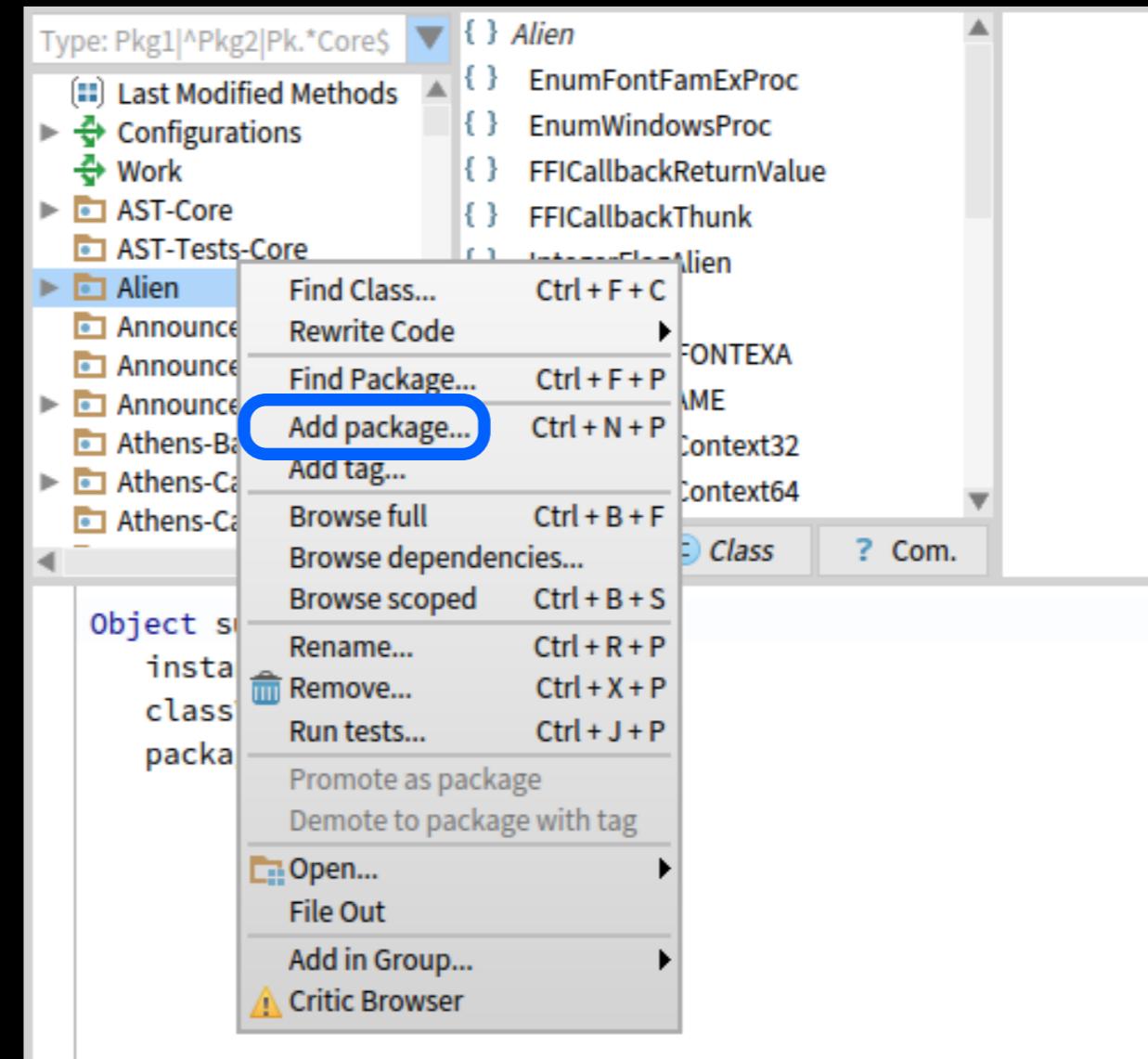
```
| nombreAEndevinar nombreProposat intents |
nombreAEndevinar := 100 atRandom.
intents := 0.
nombreProposat := (UIManager default request: 'Entra un nombre entre 1 i 100'
                     initialAnswer: 0
                     title: 'Nombre?') asInteger.
[ nombreAEndevinar = nombreProposat ]
  whileFalse: [
    Transcript show: ( nombreProposat > nombreAEndevinar
                      ifTrue: [ 'més petit' ]
                      ifFalse: [ 'més gran' ] ); cr.
    nombreProposat := (UIManager default
                     request: 'Entra un nombre entre 1 i 100'
                     initialAnswer: 0
                     title: 'Nombre?') asInteger.
    intents := intents + 1].
Transcript cr;
  show: 'Enhorabona! Ho has aconseguit després de ', intents asString , ' intents!';
  cr.|
```

Exercici: TDD en Smalltalk → la classe Pasta

Volem implementar la classe **Pasta**, per poder manipular diferents quantitats en diferentes monedes (euro, dolar, etc.)

Programarem com és habitual, en el **System Browser**

Afegirem un **Package** mitjançant el menu contextual de la finestra dels paquets.



Exercici: TDD en Smalltalk → la classe Pasta

Anomenarem **Proves** (o com vulgueu) a la nova categoria i el primer que farem és definir la classe **PastaTest**.

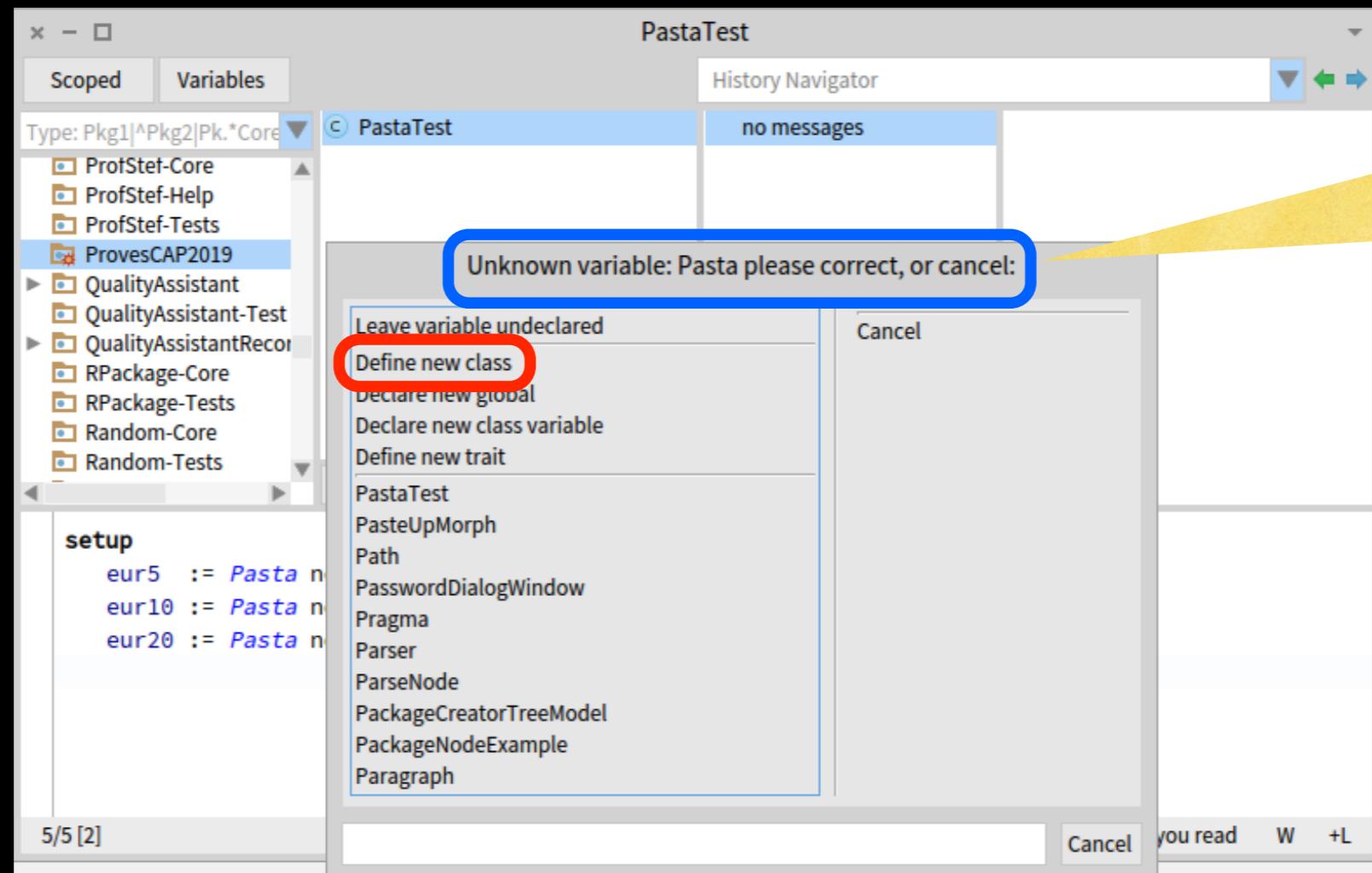
The screenshot shows the Smalltalk IDE interface with the following details:

- Toolbar:** Includes buttons for 'x - □', 'Scoped' (selected), 'Variables', and a dropdown menu.
- Left pane (Type browser):** Shows a tree view of packages:
 - Pkg1|^Pkg2|Pk.*Core
 - ProfStef-Core
 - ProfStef-Help
 - ProfStef-Tests
 - ProvesCAP2019 (highlighted)
 - QualityAssistant
 - QualityAssistant-Test
 - QualityAssistantRecorder
 - RPackage-Core
 - RPackage-Tests
 - Random-Core
 - Random-Tests
- Central pane (History Navigator):** Titled "PastaTest", it displays the message "no messages".
- Bottom pane (Code editor):** Displays the code for the class "PastaTest".

```
TestCase subclass: #PastaTest
instanceVariableNames: 'eur5 eur10 eur20'
classVariableNames: ''
package: 'ProvesCAP2019'
```
- Callout bubble:** A yellow callout points from the text "Atenció! És un test, per tant és subclasse de TestCase" to the word "TestCase" in the code editor.
- Status bar:** Shows "1/4 [1]" on the left and "Format as you read W +L" on the right.

Exercici: TDD en Smalltalk → la classe Pasta

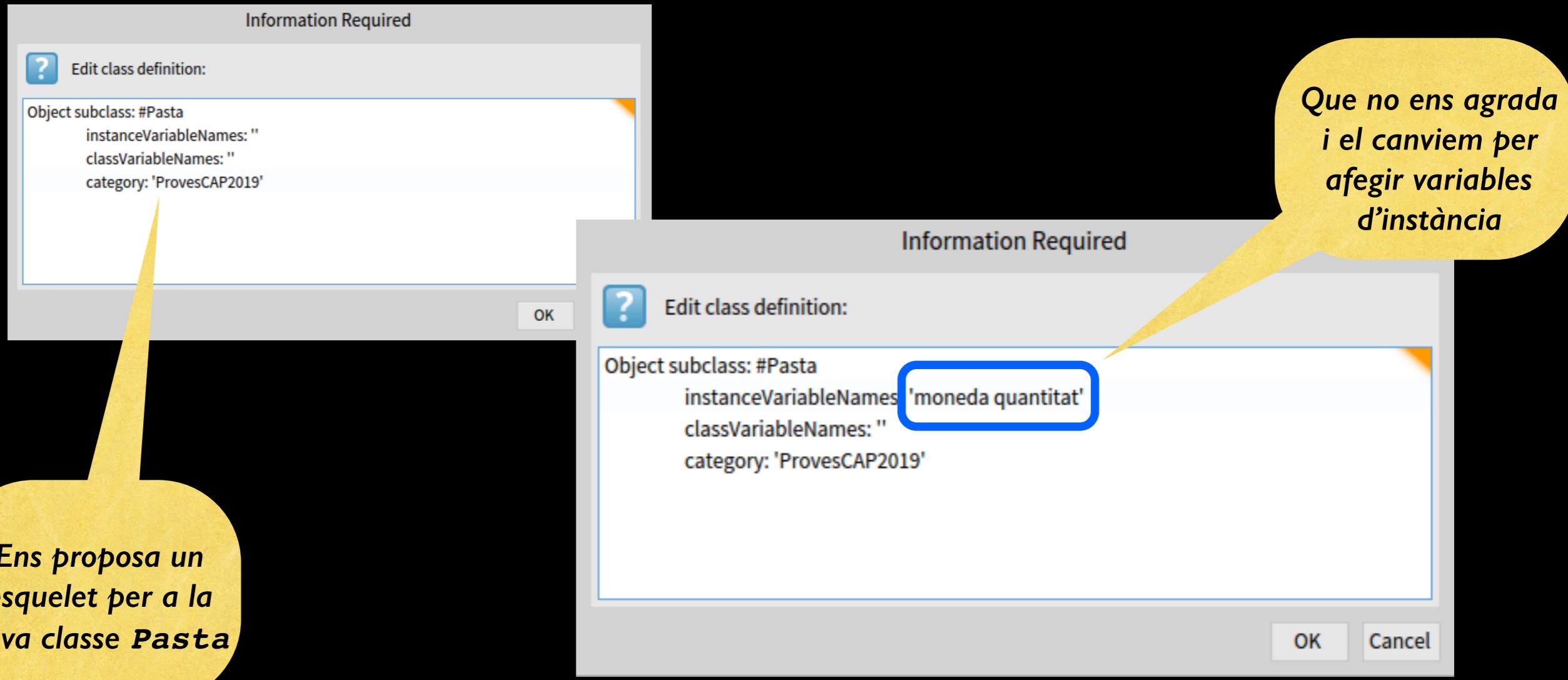
Crearem un mètode **setUp** per inicialitzar les variables d'instància de **PastaTest** és a dir, **eur5**, **eur10**, i **eur20**.



Atenció, el **setUp** s'executa abans de **cada** test!!

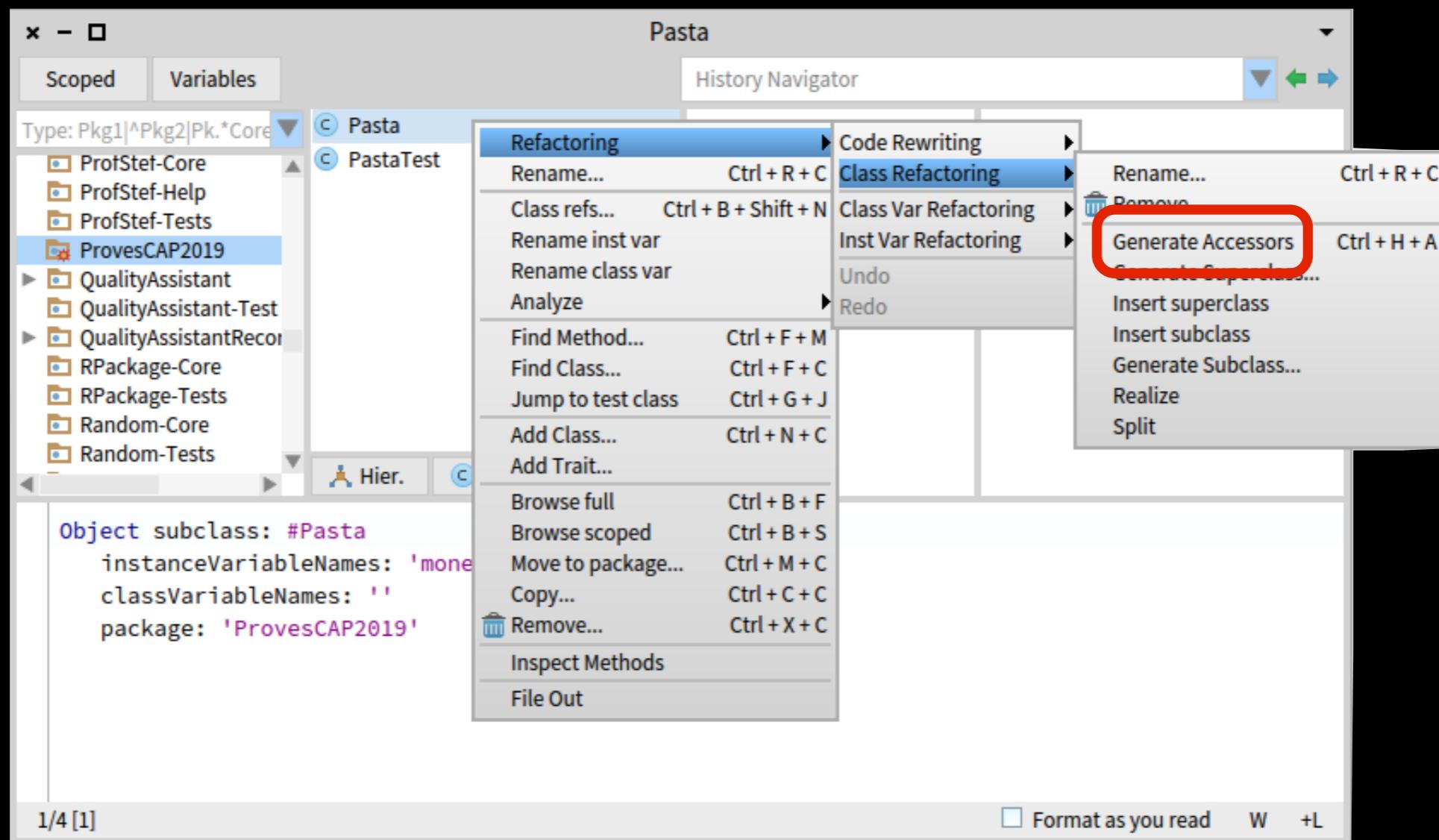
Exercici: TDD en Smalltalk → la classe Pasta

Crearem un mètode **setUp** per inicialitzar les variables d'instància de **PastaTest** és a dir, **eur5**, **eur10**, i **eur20**.



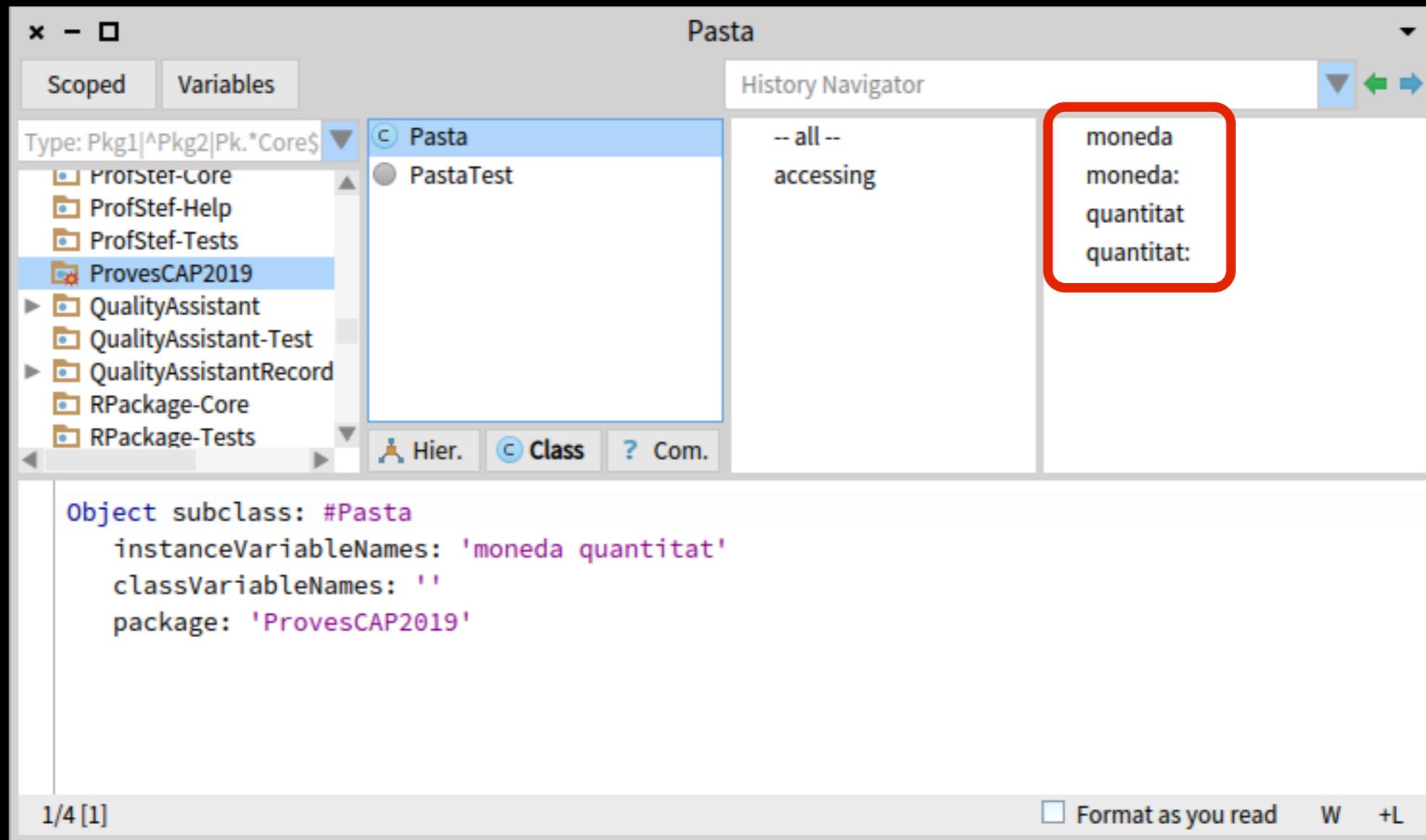
Exercici: TDD en Smalltalk → la classe Pasta

Crearem *setters* i *getters* per a totes les variables d'instància. Ho podem fer automaticament amb el menu contextual de les classes.



Exercici: TDD en Smalltalk → la classe Pasta

Crearem *setters* i *getters* per a totes les variables d'instància. Ho podem fer automaticament amb el menu contextual de les classes.



Exercici: TDD en Smalltalk → la classe Pasta

Amb aquesta infraestructura mínima, podem crear el nostre primer test, per mirar si les igualtats tenen algun sentit...

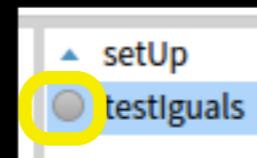
The screenshot shows a Smalltalk IDE interface with the following details:

- Title Bar:** PastaTest>>#testIquals
- Toolbar:** Includes buttons for Scoped, Variables, History Navigator, and others.
- Left Panel (Type Browser):** Shows a list of packages:
 - Pkg1|^Pkg2|Pk.*Core\$
 - Polymorph-Widgets
 - PragmaCollector
 - ProfStef-Core
 - ProfStef-Help
 - ProfStef-Tests
 - ProvesCAP2019 (highlighted)
 - QualityAssistant
 - QualityAssistant-Test
- Middle Panel (History Navigator):** Shows a tree structure:
 - all --
 - Initialization tests
 - setUp
 - testIquals (highlighted)
- Bottom Panel (Code Editor):** Displays the source code for the testIquals method:

```
testIquals
    self assert: eur5 = eur5.
    self assert: eur10 = (Pasta new moneda: 'EUR'; quantitat: 10).
    self assert: eur20 ~= eur10.
```
- Status Bar:** Shows "1/5 [1]" and "Format as you read W +L".

Exercici: TDD en Smalltalk → la classe Pasta

Podem mirar si el test és correcte...



PastaTest>>#testIguals

Type: Pkg1|^Pkg2|Pk.*Core\$

History Navigator

Pasta

PastaTest

-- all --

initialization

tests

setUp

testIguals

TestFailure: Assertion failed

Proceed Abandon Debug Report

PastaTest(Testasserter) assert:
PastaTest testIguals
PastaTest(TestCase) performTest
PastaTest(TestCase) runCase [self setUp. self performTest]
BlockClosure ensure:
PastaTest(TestCase) runCase
self assert: eur20 ~= eur10.

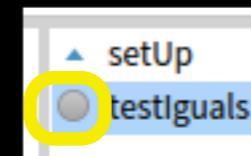
5/5 [1]

Format as you read W +L

```
PastaTest(Testasserter) assert:  
PastaTest testIguals  
PastaTest(TestCase) performTest  
PastaTest(TestCase) runCase [ self setUp. self performTest ]  
BlockClosure ensure:  
PastaTest(TestCase) runCase  
self assert: eur20 ~= eur10.
```

Exercici: TDD en Smalltalk → la classe Pasta

Podem mirar si el test és correcte...



The screenshot shows the Smalltalk IDE interface. At the top, a window titled "PastaTest>>#testIquals" is open. Below it, a "History Navigator" window shows the same list of methods. A yellow arrow points from the "testIquals" entry in the History Navigator to the corresponding line in the code editor below. The code editor displays the following:

```
PastaTest>>#testIquals
| self assert: testIquals performTest | [ self setUp. self performTest ]
self assert: eur20 ~= eur10.
```

The status bar at the bottom left shows "5/5 [1]" and the bottom right has options "Format as you read" and "W +L".

NO HO ÉS!

CAP: Introducció a l'Smalltalk

Exercici: TDD en Smalltalk → la classe Pasta

Per què?!?!



Exercici: TDD en Smalltalk → la classe Pasta

The screenshot shows a Smalltalk debugger interface with the following details:

- Stack:** The stack trace shows the execution path:
 - PastaTest(TestAssertioner)
 - PastaTest
 - PastaTest(TestCase)
 - PastaTest(TestCase)
- Source:** The source code for the `testIquals` method is displayed:

```
testIquals
    self assert: eur5 = eur5.
    self assert: eur10 = (Pasta new moneda: 'EUR'; quantitat: 10).
    self assert: eur20 ~= eur10.
```
- Variables:** A table showing variable values:

Type	Variable	Value
implicit	self	PastaTest>>#testIquals
attribute	eur10	a Pasta
attribute	eur20	a Pasta

Exercici: TDD en Smalltalk → la classe Pasta

Hauriem de revisar què vol dir *ser igual a...* Com **Pasta** és subclasse d'**Object**, caldria revisar com aquest implementa el missatge **=** (ja que **Pasta** no l'implementa).

The screenshot shows the Smalltalk Object browser interface. The left pane displays a class hierarchy tree with categories like Objects, Pragmas, Processes, Protocols, Kernel-Rules, Kernel-Tests, and Kernel-Tests-Rules. The right pane shows the implementation of the `Object>>#=` message. A yellow callout bubble points to the message selector in the history navigator.

```
x - □
Type: Pkg1|^Pkg2|Pk.*Core$ Object>>#=

Objects
Pragmas
Processes
Protocols
Kernel-Rules
Kernel-Tests
Kernel-Tests-Rules

Hier. Class ? Com.

= anObject
"Answer whether the receiver and the argument represent the same object. If = is redefined in any subclass, consider also redefining the message hash."
^self == anObject

History Navigator
comparing
converting
copying
dependencies
deprecation
displaying
error handling
generalizing
AcceptDroppingMorph...
acceptSettings:
actAsExecutor
actionForEvent:

1/6 [1] Format as you read W +L
```

La igualtat s'implementa com la identitat...
Nosaltres NO volem això

Exercici: TDD en Smalltalk → la classe Pasta

Per tant, caldrà que **Pasta** implementi el missatge `=`

The screenshot shows the Smalltalk IDE interface with the following details:

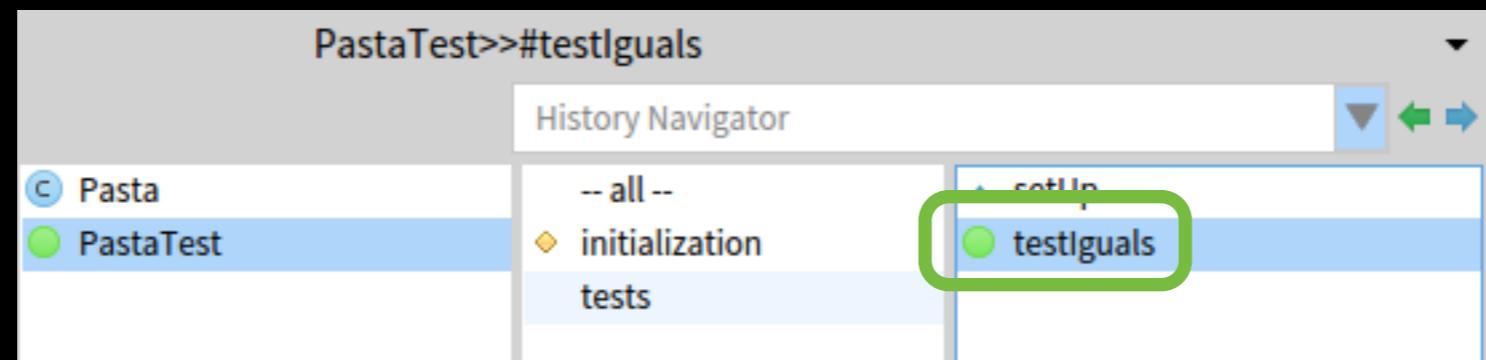
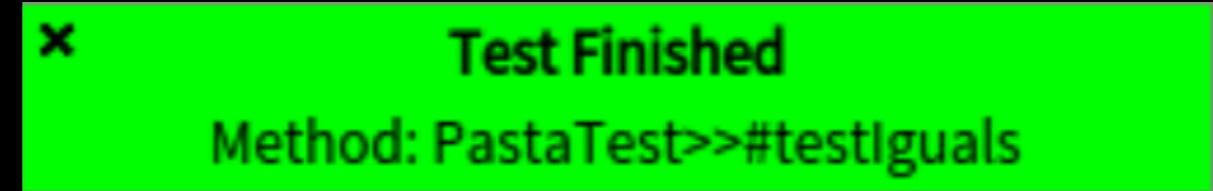
- Project Browser:** On the left, under "Type: Pkg1|^Pkg2|Pk.*Core\$", the "ProvesCAP2019" package is selected.
- Class Browser:** In the center, the class **Pasta** is selected. The browser shows the following code:

```
= unaPasta
  ^ self moneda = unaPasta moneda and: [ self quantitat = unaPasta quantitat ].
```
- History Navigator:** On the right, the history navigator shows the message `=` being implemented. It lists the following steps:
 - all --
 - accessing
 - equality
- Status Bar:** At the bottom, it shows "1/2 [1]" on the left and "Format as you read W +L" on the right.

Per què no cal que Pasta implementi el missatge `=` ???

Exercici: TDD en Smalltalk → la classe Pasta

Ara el test ja és correcte...



Exercici: TDD en Smalltalk → la classe Pasta

Podriem implementar la suma de monedes, i fer el test corresponent...

The screenshot shows the Smalltalk IDE interface with the following details:

- Top Left:** A tree view of packages: Pkg1|^APkg2|Pk.*Core\$.
- Top Center:** The class **Pasta** is selected in the browser.
- Top Right:** A history navigator pane.
- Bottom Left:** A code editor pane containing the following Smalltalk code:

```
+ unaPasta
  ^ self moneda = unaPasta moneda
    ifTrue: [ Pasta new moneda: self moneda; quantitat: (self quantitat + unaPasta quantitat) ]
    iffFalse: [ self error: 'No es poden utilitzar monedes diferents' ]
```
- Bottom Right:** Status bar showing "1/5 [1]" and "Format as you read W +L".

Exercici: TDD en Smalltalk → la classe Pasta

Podriem implementar la suma de monedes, i fer el test corresponent...

The screenshot shows the Smalltalk IDE interface with two open windows:

- Pasta>>#+**: This window displays the implementation of the `+` message for the `Pasta` class. It includes a history navigator with items like `accessing`, `arithmetic`, and `moneda`. The code shows a method named `unaPasta` which creates a new `Pasta` object with a specific moneda value based on a condition.
- PastaTest>>#testSumes**: This window displays the test implementation for the `+` message. It includes a history navigator with items like setUp, testEquals, and testSumes. The test code uses assertions to verify the sum of different moneda values.`

```
1/5 [1]
```

```
1/5 [1]
```

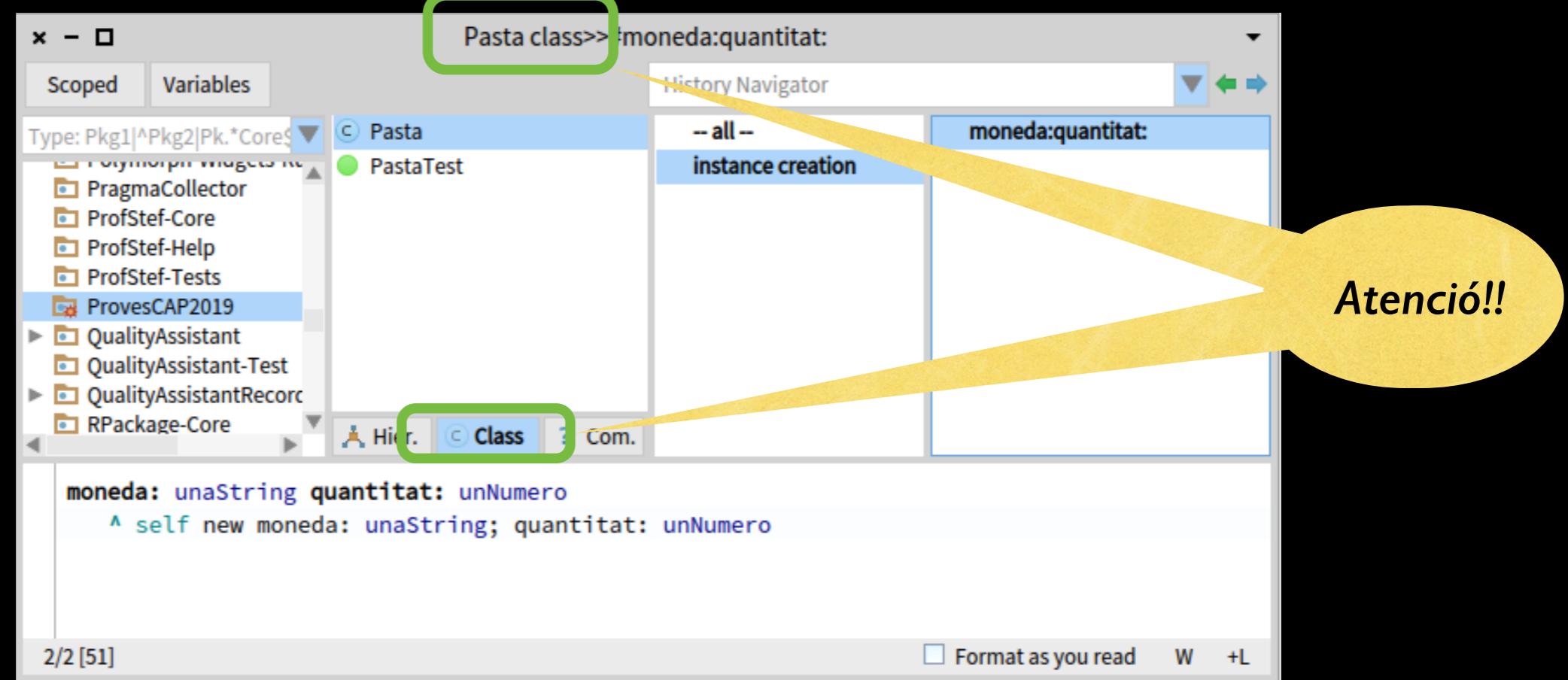
```
Format as you read W +L
```

```
+ unaPasta
  ^ self moneda = 
    ifTrue: [ Pa ]
    ifFalse: [ s ]
```

```
self assert: eur5 + eur5 = eur10.
self assert: eur5 + eur10 = (Pasta new moneda: 'EUR'; quantitat: 15).
self assert: (eur5 + eur5 + eur10) = eur20.
```

Exercici: TDD en Smalltalk → la classe Pasta

Voldriem tenir un constructor per a la classe **Pasta**, ja que la construcció **Pasta new moneda: x; quantitat: y** no ens agrada.



Ara podem construir nous objectes de classe **Pasta** fent:
Pasta moneda: x quantitat: y

Exercici: TDD en Smalltalk → la classe Pasta

Caldria canviar tot el codi on construim objectes de classe **Pasta** i utilitzar el nou constructor. Per exemple, la suma + ...

The screenshot shows the Smalltalk IDE interface with the following details:

- Toolbar:** Includes buttons for 'x - □' (close/minimize/maximize), 'Scoped' (selected), 'Variables', and tabs for 'Hier.', 'Class' (selected), and 'Com.'.
- Top Bar:** Shows the class name **Pasta>>#+**.
- History Navigator:** A panel on the right containing a tree view of methods: --all--, accessing, arithmetic (selected), equality, +, and =.
- Code Editor:** The main pane displays the source code for the **Pasta** class.

```
+ unaPasta
  ^ self moneda = unaPasta moneda
    ifTrue: [ Pasta moneda: self moneda quantitat: (self quantitat + unaPasta quantitat) ]
    ifFalse: [ self error: 'No es poden utilitzar monedes d'altres' ]
```

A green oval highlights the line `Pasta moneda: self moneda quantitat: (self quantitat + unaPasta quantitat)`.
- Status Bar:** Shows '3/5 [62]' at the bottom left and 'Format as you read' with checkboxes for 'W' and '+L' at the bottom right.

Després caldrà tornar a passar els tests...

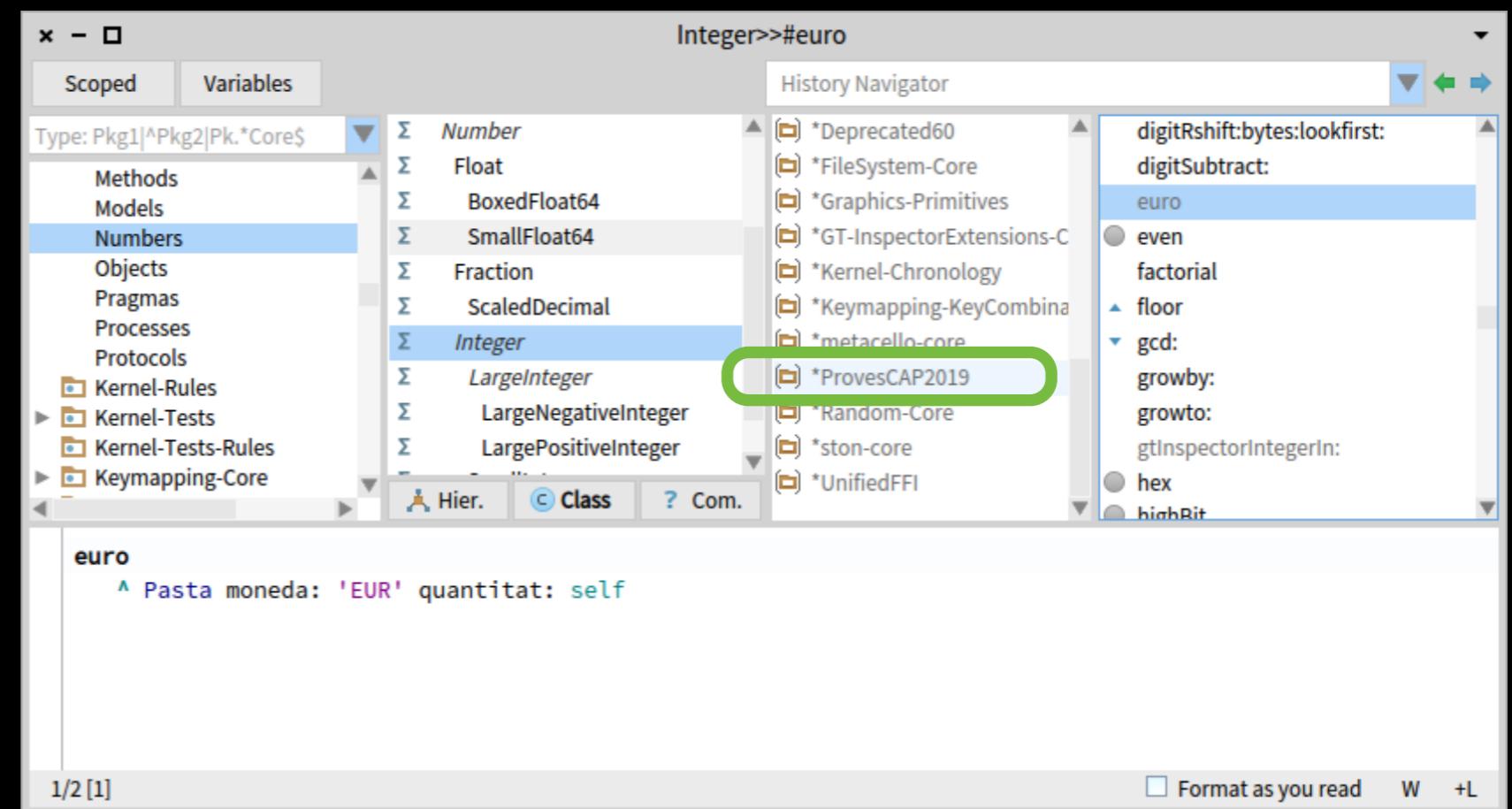
Exercici: TDD en Smalltalk → la classe Pasta

Però... no estem del tot satisfets. Estaria bé tenir un constructor pels enters, un per a cada moneda:

5 euro \leftrightarrow Pasta moneda: 'EUR' quantitat: 5

Primer es crea un nou protocol a la classe **Integer**, anomenat ***Proves**.

El nom del protocol és important



Exercici: TDD en Smalltalk → la classe Pasta

Podem redefinir el codi on construim euros i...
passar altre cop els tests.

The screenshot shows the Smalltalk IDE interface with the following details:

- Project Browser (left):** Shows packages like PragmaCollector, ProfStef-Core, ProfStef-Help, ProfStef-Tests, ProvesCAP2019, QualityAssistant, QualityAssistant-Test, QualityAssistantRecord, and RPakage-Core.
- Class Browser (center):** The current class is **PastaTest**, which has a dependency on **Pasta**. It also includes **Integer**.
- History Navigator (right):** Shows the history of changes for the **setUp** method, including **initialization** and **tests**, with methods **testEquals** and **testSumes**.
- Code Editor (bottom-left):** Displays the **setUp** method code:

```
setUp
    eur5 := 5 euro.
    eur10 := 10 euro.
    eur20 := 20 euro.
```
- Status Bar (bottom-right):** Shows "Format as you read" and other UI controls.

I ara podríem continuar fent... però no ho farem.
Guardeu la imatge si voleu conservar el que heu fet.

Exercici: TDD en Smalltalk → la classe Pasta

Podem redefinir el codi on construim euros i...
passar altre cop els tests.

The screenshot shows a Smalltalk development environment. The top bar displays the title "PastaTest>>#testSumes". The left pane is a "History Navigator" showing a list of methods: "setUp", "testIguals", and "testSumes". The right pane shows the source code for "testSumes". The code is as follows:

```
testSumes
    self assert: 5 euro + 5 euro = 10 euro.
    self assert: 5 euro + 10 euro = 15 euro.
    self assert: (5 euro + 5 euro + 10 euro) = 20 euro.
```

At the bottom, there are navigation buttons: "Hier.", "Class", and "? Com.". The status bar at the bottom right shows "5/5 [2]" and "Format as you read W +L".

I ara podríem continuar fent... però no ho farem.
Guardeu la imatge si voleu conservar el que heu fet.

Paquets (*Packages*)

Guardar la imatge és, però, molt poc pràctic. Per manipular codi amb certa agilitat a Pharo hi ha el concepte de *Package*, o Paquet.

Il·lustrem-ho amb el nostre exemple. Hem creat un nou paquet anomenat **Proves** (o similar).

Què hi ha dins el paquet?

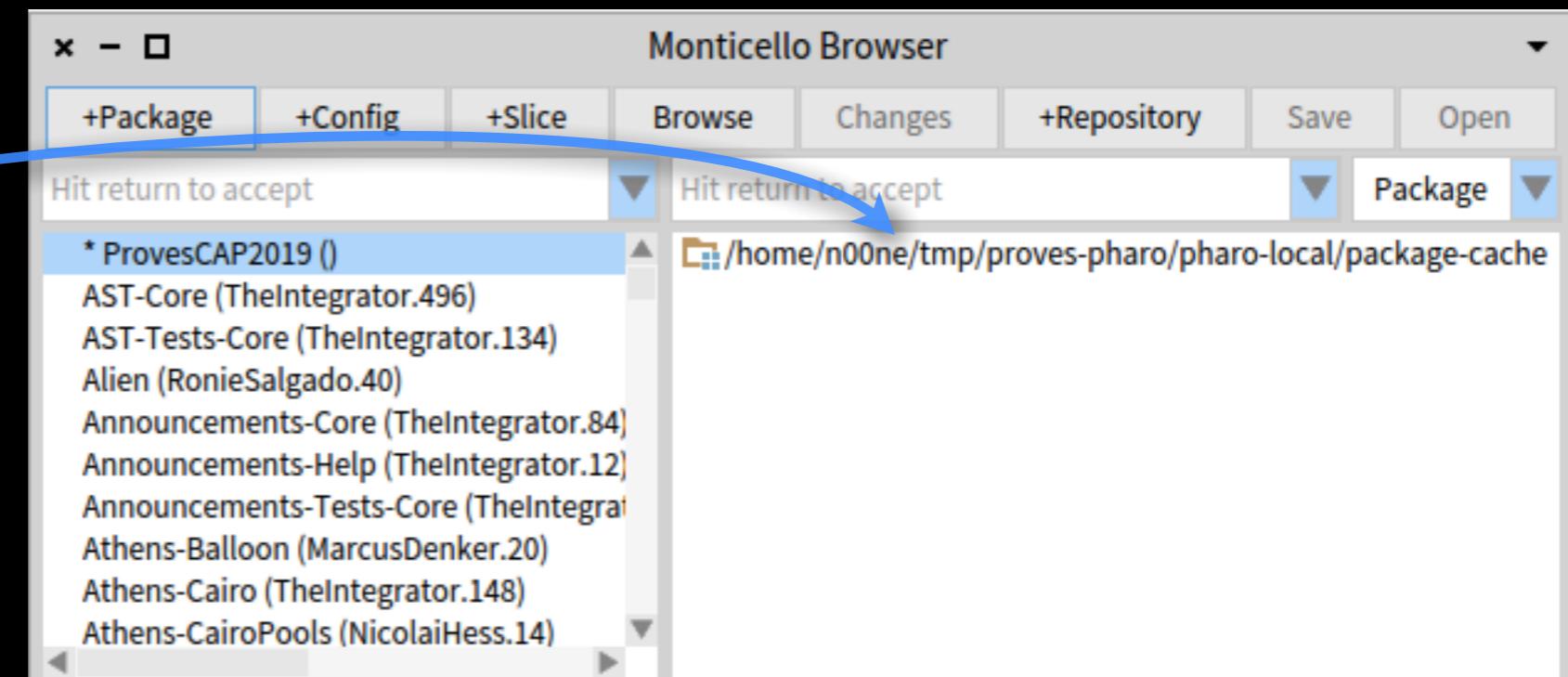
- Totes les definicions de classes del paquet **Proves** o dels paquets que comencen amb **Proves-**...
- Tots els mètodes del protocol ***Proves** o ***Proves-**... allà on estiguin definits, no importa la classe (en el nostre cas, s'inclouria el mètode **euros** de la classe **Integer**).
- Tots els mètodes de les classes que pertanyen a les categories **Proves** i **Proves-**..., **excepte** aquells que pertanyen a protocols els noms dels quals comencin amb ***** (ja que pertanyen a altres paquets).

Paquets (*Packages*)

Els paquets els gestionem mitjançant el **Monticello Browser**.

Fixem-nos que ens marca el paquet **Proves** amb un asterisc (*), indicant-nos que cal actualitzar-lo (l'hem modificat i encara no l'hem guardat).

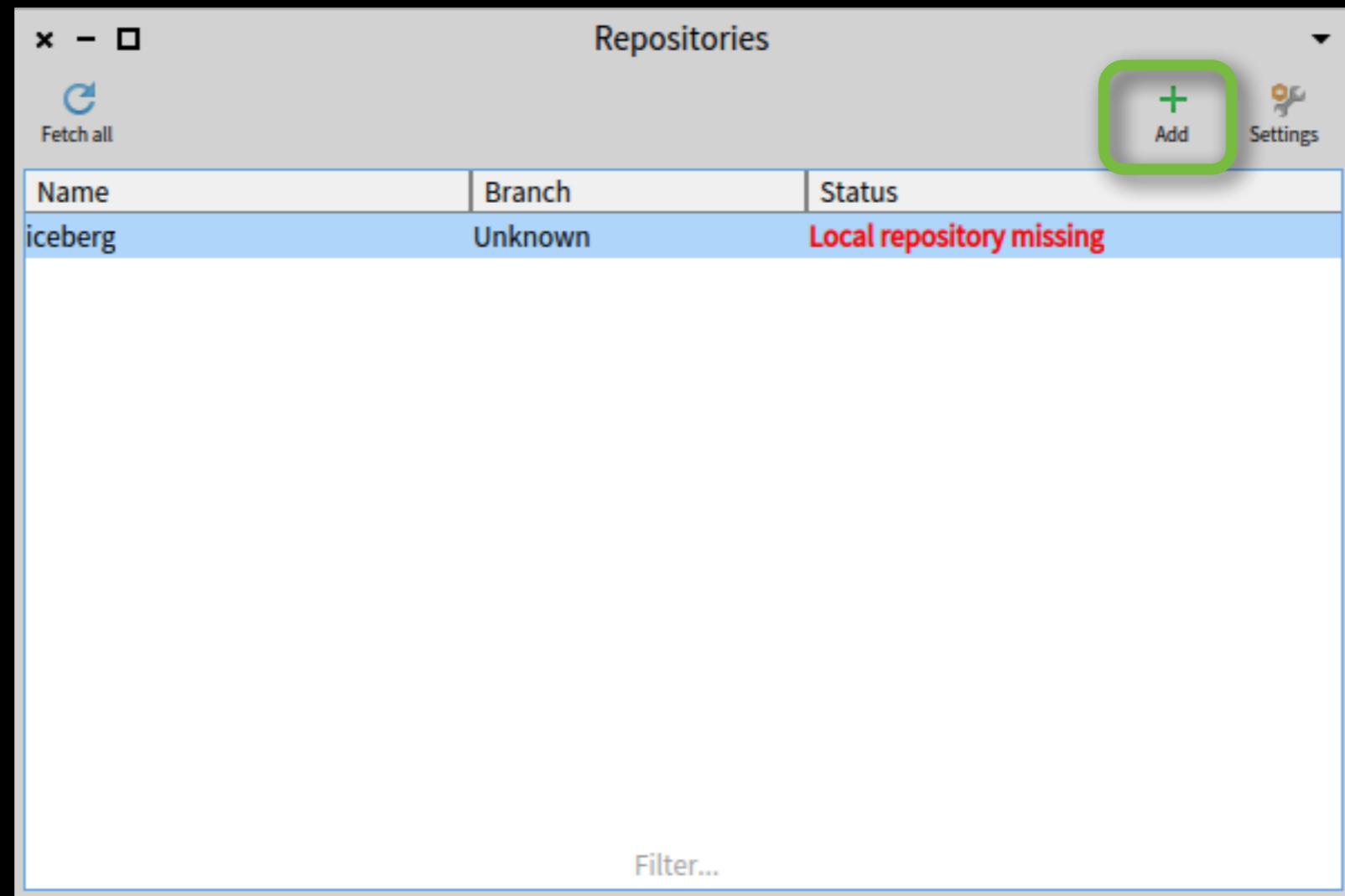
Els paquets es guarden a **repositorys**, i sempre n'hi ha un per defecte, al disc local, que dependrà de la nostra instal.lació de Pharo. Es diu **package-cache**



Ara bé, nosaltres preferirem guardar el codi *on-line*, fent servir GIT.

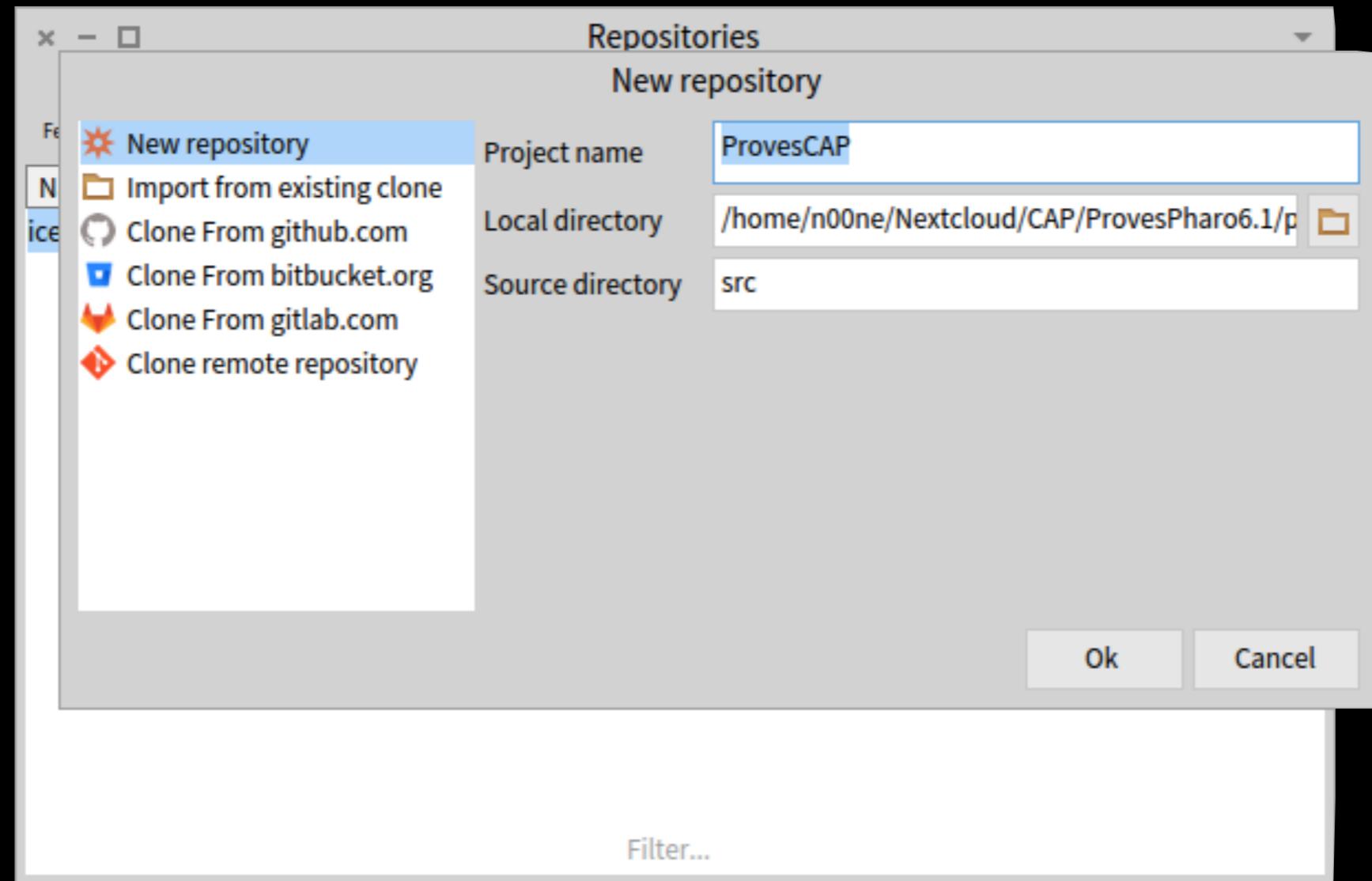
Paquets (*Packages*)

Podem gestionar el paquet **Proves** amb **Git** (a github, per exemple), utilitzant l'eina anomenada **Iceberg** (aquí suposarem que heu seguit les indicacions mencionades en la recepta d'instal·lació de Pharo que hem vist al principi i heu actualitzat Iceberg al vostre Pharo 6.1). Afegirem un repositori nou...



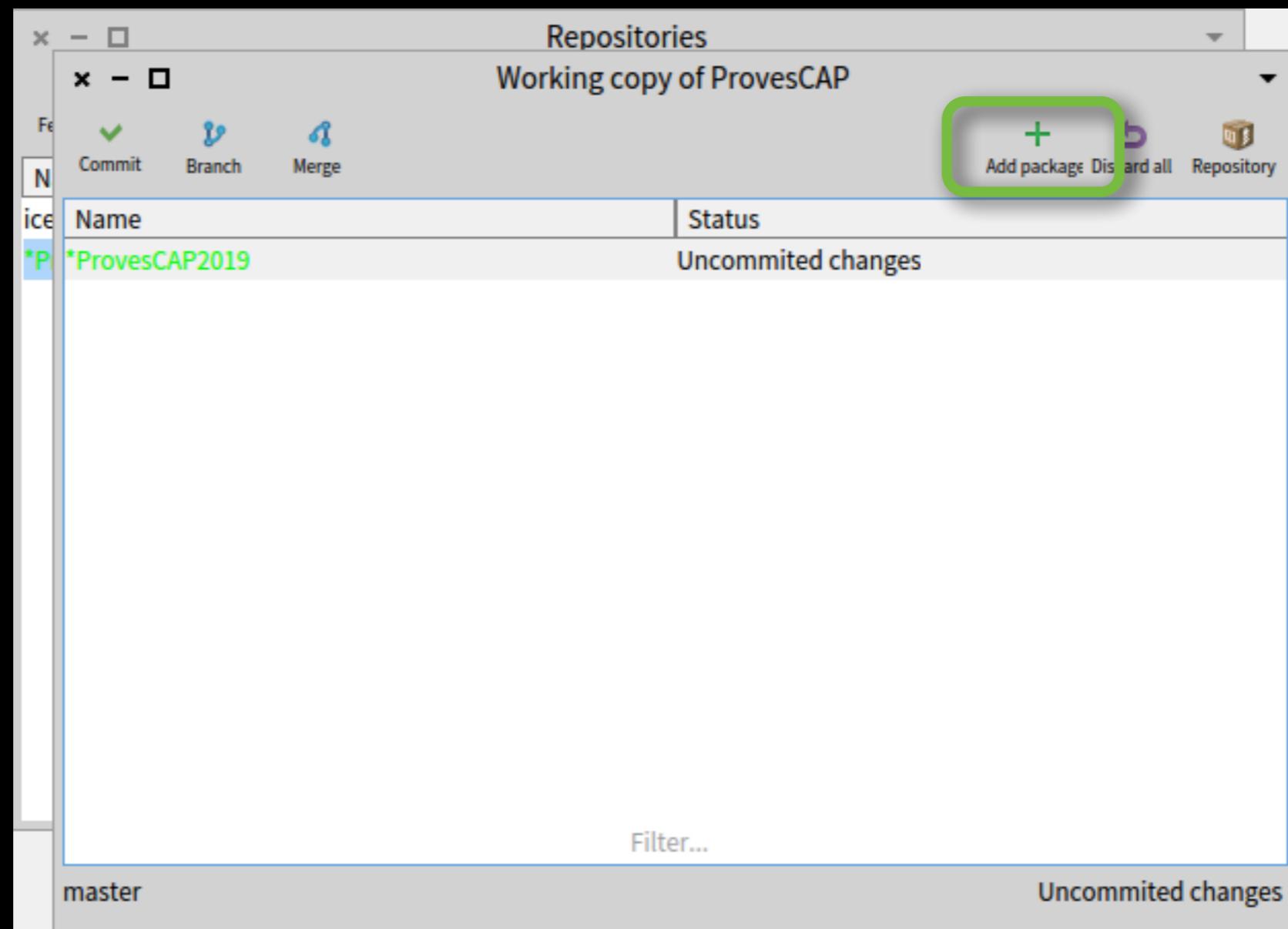
Paquets (*Packages*)

Podem gestionar el paquet **Proves** amb **Git** (a github, per exemple), utilitzant l'eina anomenada **Iceberg** (aquí suposarem que heu seguit les indicacions mencionades en la recepta d'instal·lació de Pharo que hem vist al principi i heu actualitzat Iceberg al vostre Pharo 6.1). Afegirem un repositori nou...



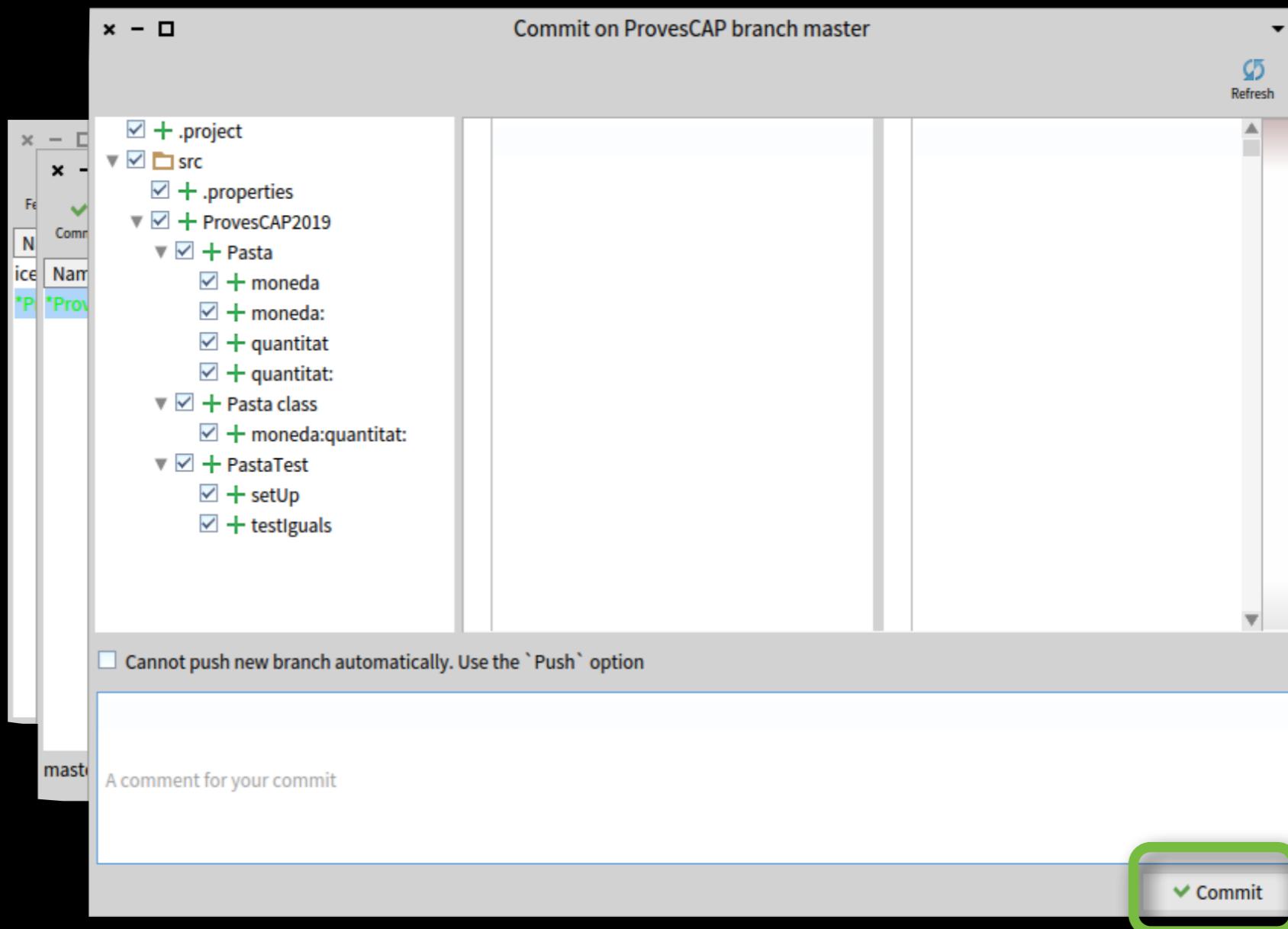
Paquets (*Packages*)

... i afegirem el paquet **Proves** a aquest repositori, amb **Add package**



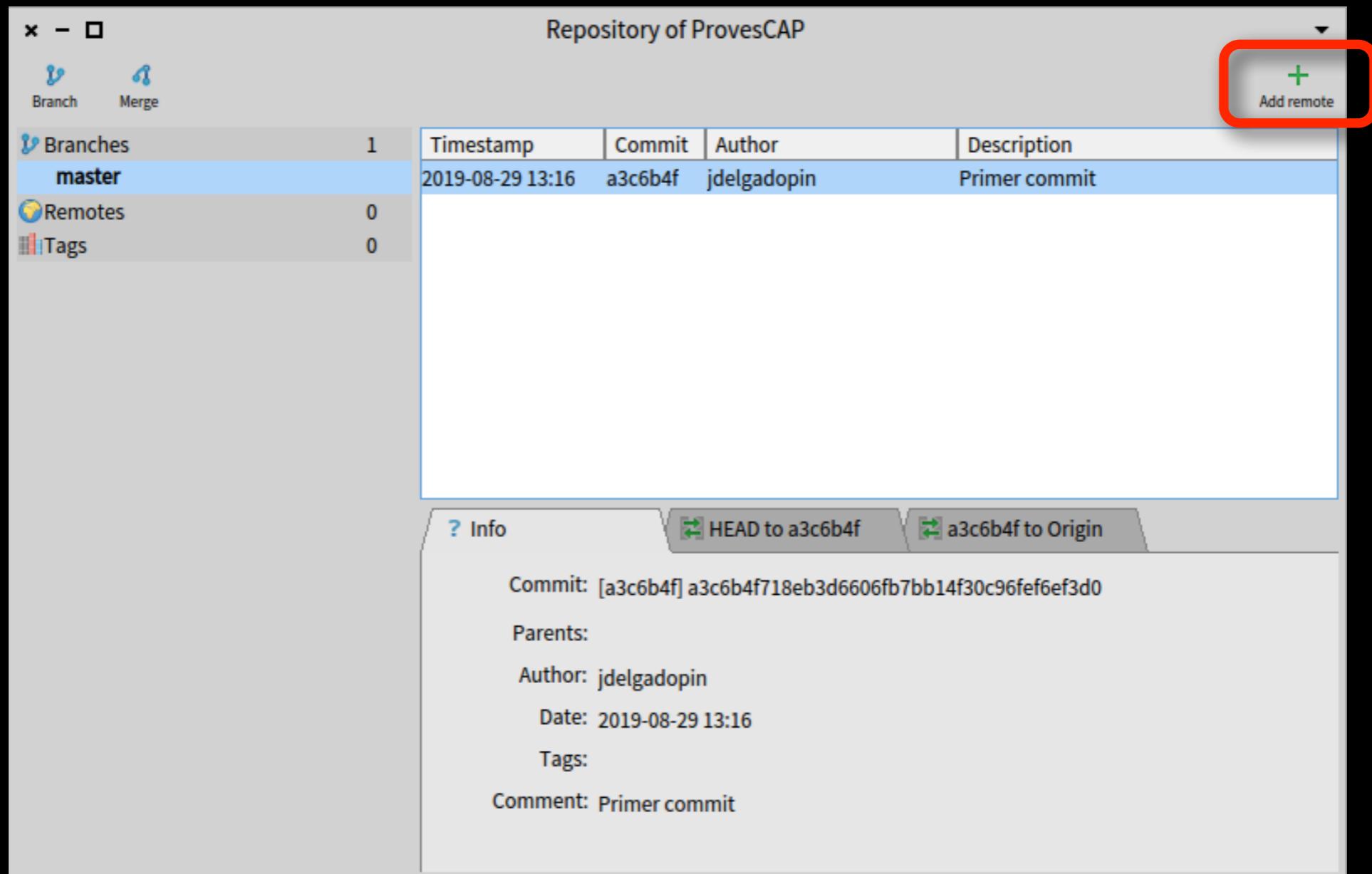
Paquets (*Packages*)

Ara, farem el primer (i únic en aquest exemple) **commit**, decidint què volem afegir i què no marcant-ho a la llista de fitxers. També hem de decidir quin repositori remot fem servir.



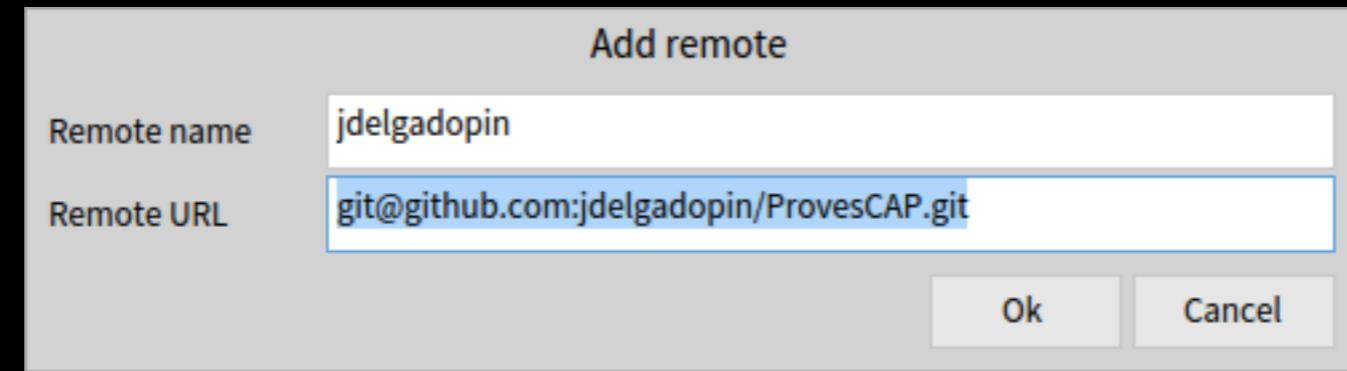
Paquets (*Packages*)

Ara, farem el primer (i únic en aquest exemple) **commit**, decidint què volem afegir i què no marcant-ho a la llista de fitxers. També hem de decidir quin repositori remot fem servir.



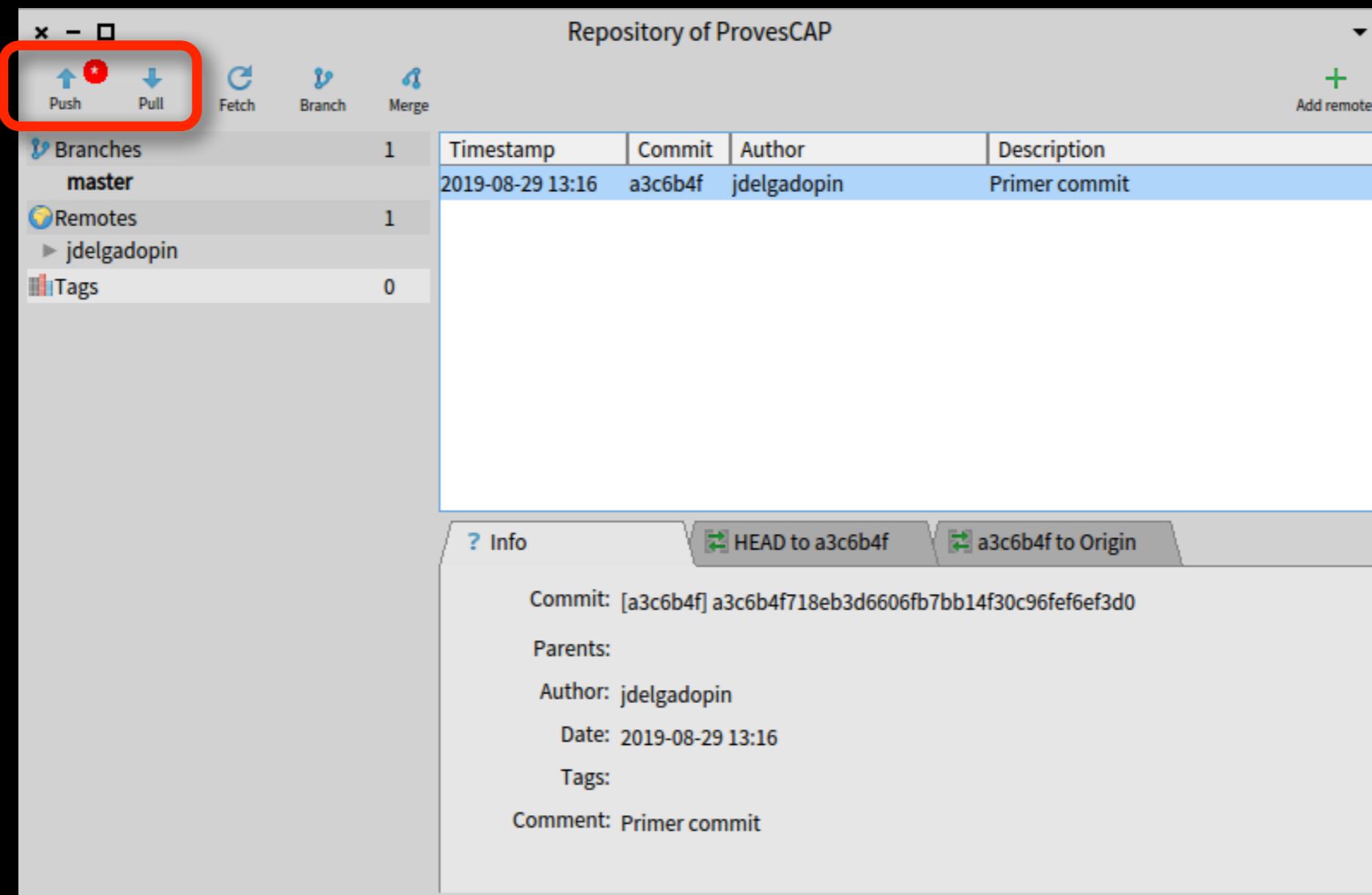
Paquets (*Packages*)

Vincularem el repositori remot (en aquest cas a **github**) al nostre repositori local (amb **ssh** segur que funciona):



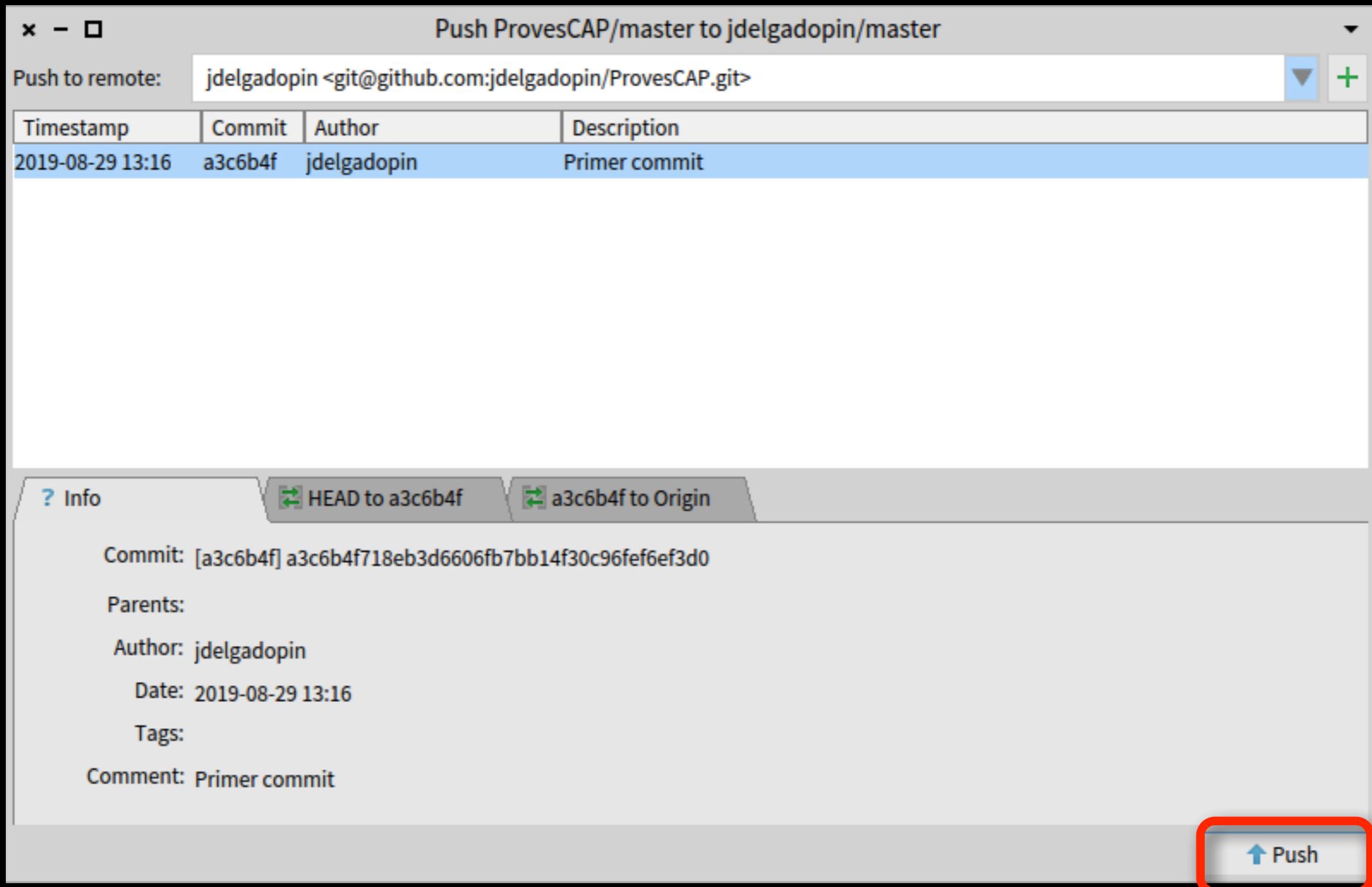
Paquets (*Packages*)

...i ja podem fer un **push**



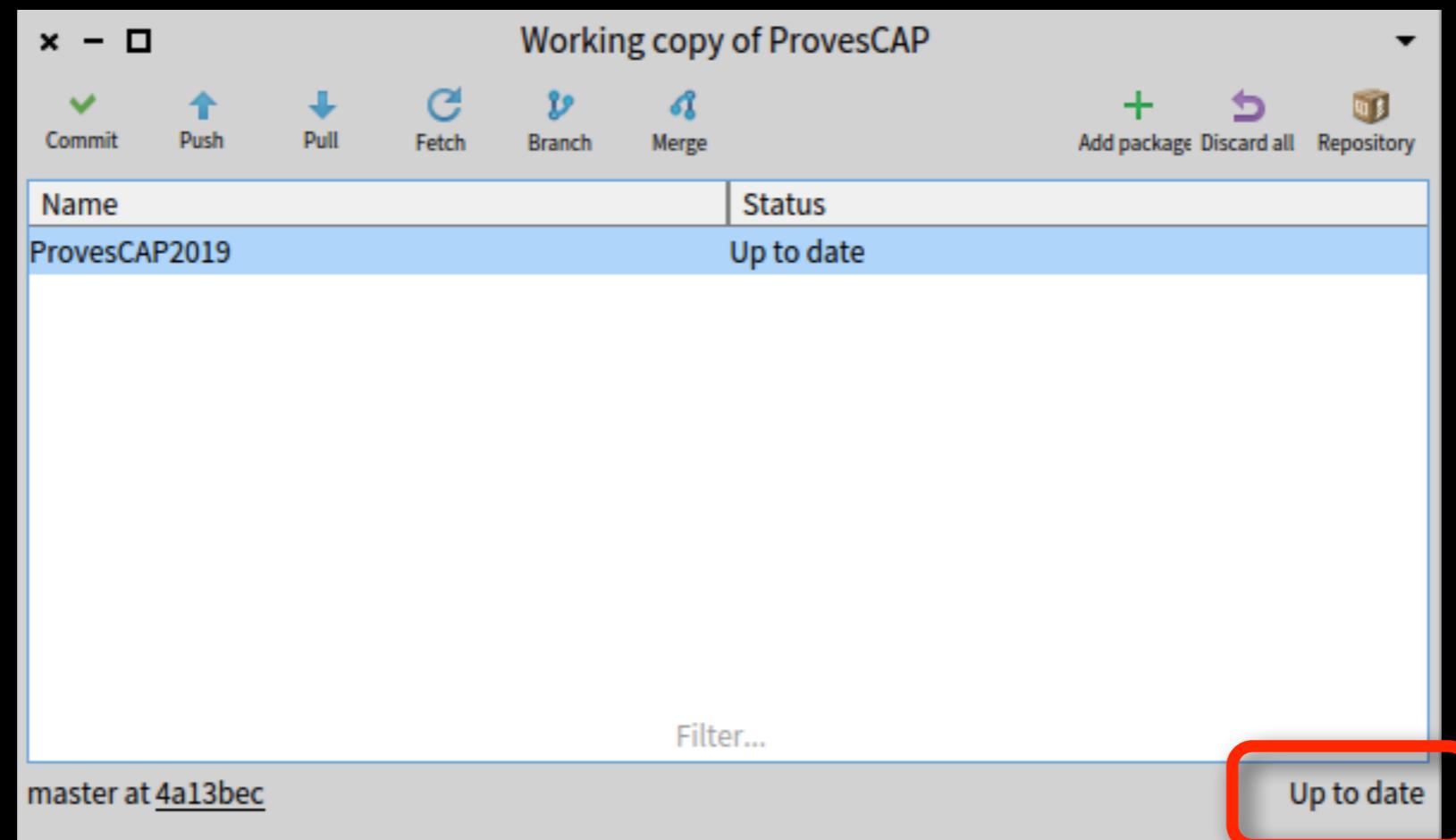
CAP: Introducció a l'Smalltalk

Paquets (*Packages*)



Paquets (*Packages*)

Finalment, tot està en ordre...



Recordeu de consultar
<https://github.com/pharo-vcs/iceberg/wiki/Tutorial>

Recapitulem...

- **Tot** és un objecte. Absolutament tot. **Tots** els objectes són instància d'una classe. **TOTS!!** Les classes defineixen el comportament i l'estructura de les seves instàncies.
- Res passa si no és per **pas de missatges**
- L'estat dels objectes (atributs, camps,...) és **privat**
- Els mètodes són **públics**
(els podem considerar privats per convenció)
- Les variables contenen **referències** (tipat dinàmic).
(els objectes no referenciats són eliminats, i.e. *garbage collection*)
- Herència **simple**.

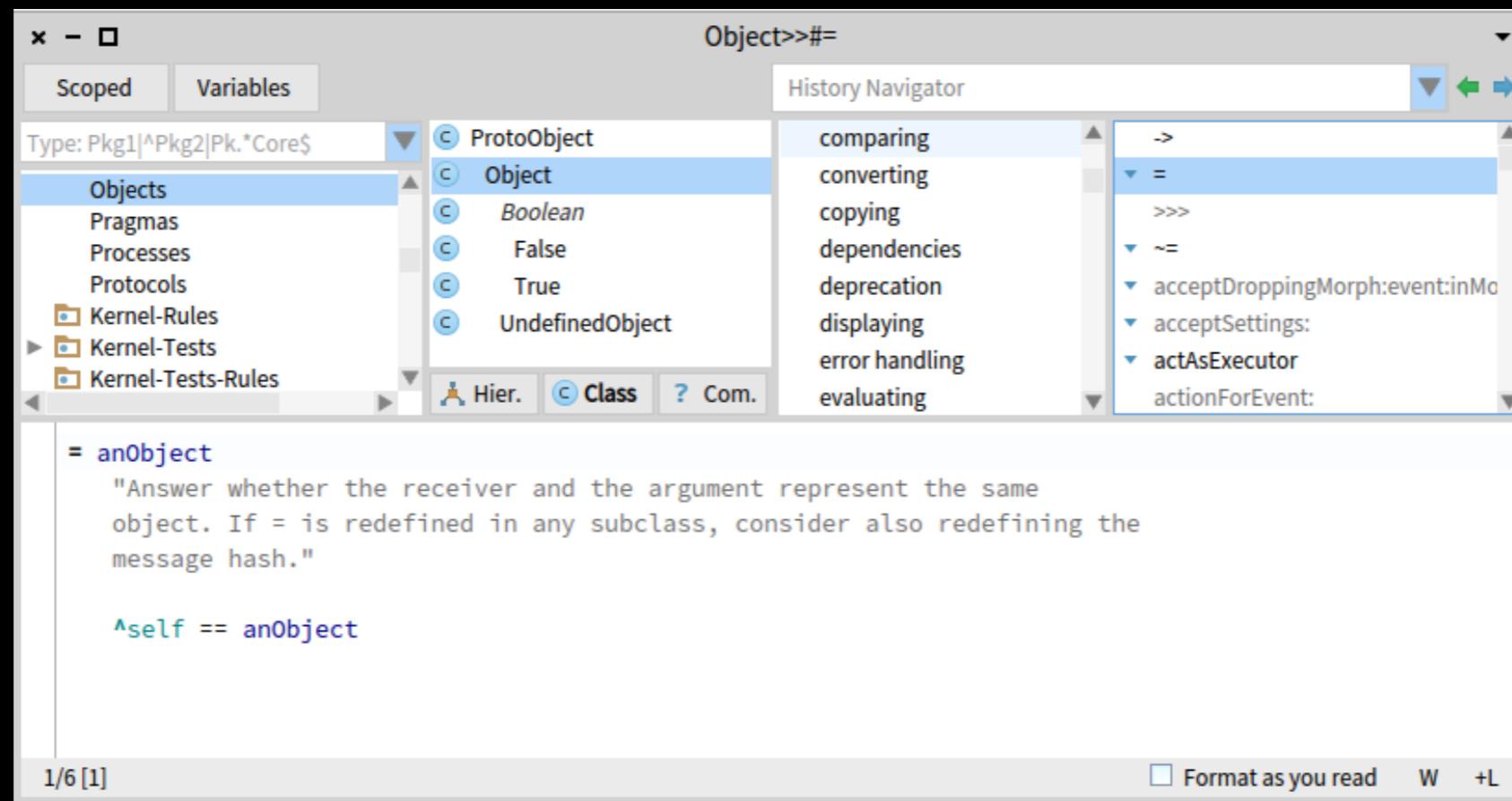
La classe **Object**

- A l'estàndard Smalltalk-80, **Object** és l'arrel de la jerarquia de classes (a Pharo i Squeak l'arrel és **ProtoObject**, que només té a com a subclasse a **Object**)
- Defineix el comportament comú i mínim per a tots els objectes del sistema: **Object numberOfMethods** → 429
- Comparació d'objectes: **==**, **~~**, **isNil**, **=**, **~=**, **notNil**
(a Pharo i Squeak **==**, **~~**, i **isNil** són a **ProtoObject**).
- Representació textual d'objectes: **#printString**, **#printOn**:

Identitat vs. Igualtat

== compara la **identitat** entre objectes. *NO s'ha de sobreescrivir mai!*

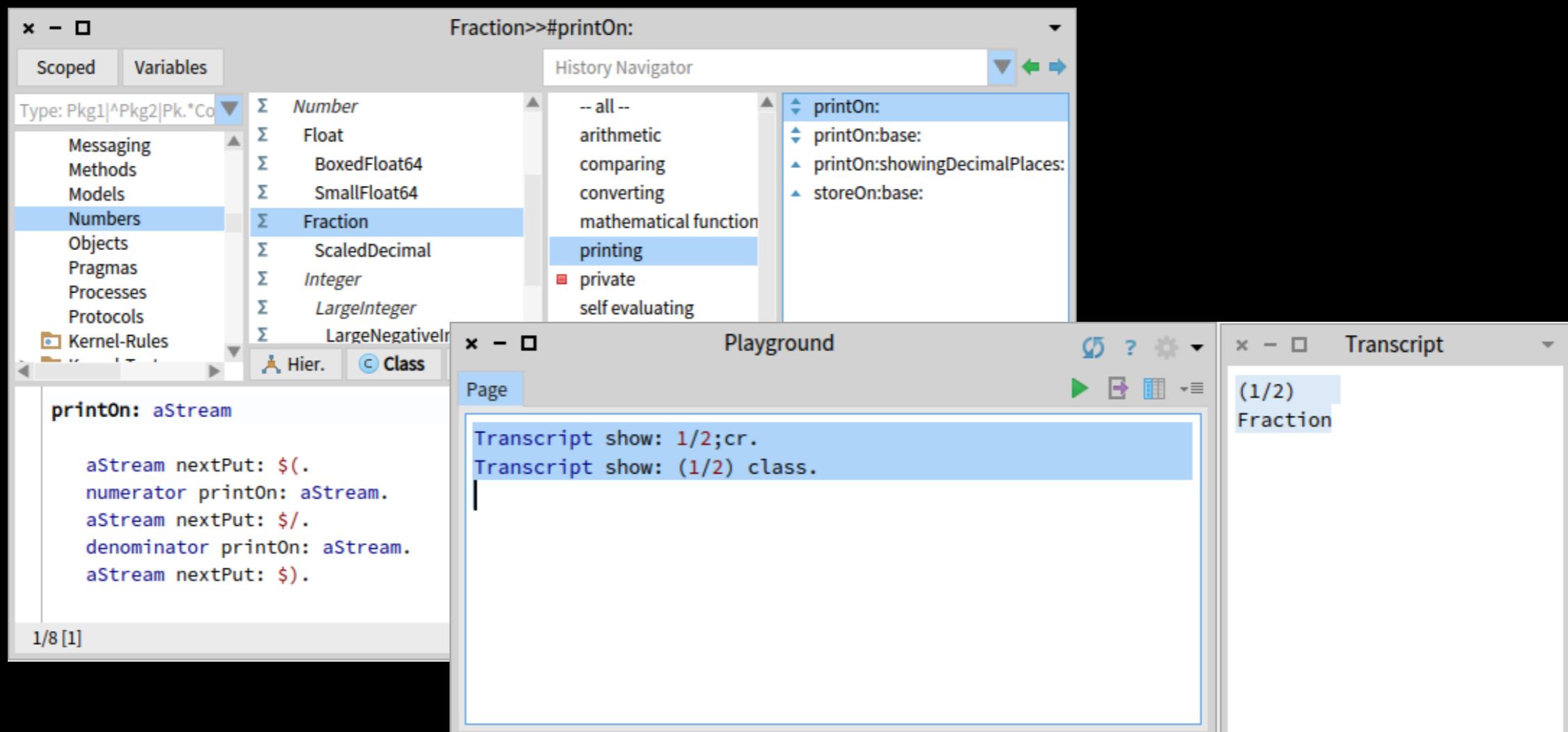
- = compara la **igualtat** entre objectes. Usualment es sobreescriva ja que la implementació per defecte és la identitat.
Però, si sobreescrivim **=** també cal sobreescrivir **Object >> hash**



(no vam sobreescrivir **Object >> hash** en l'exemple de la classe **Pasta** perque era un exemple de joguina)

Representació Textual d'Objectes

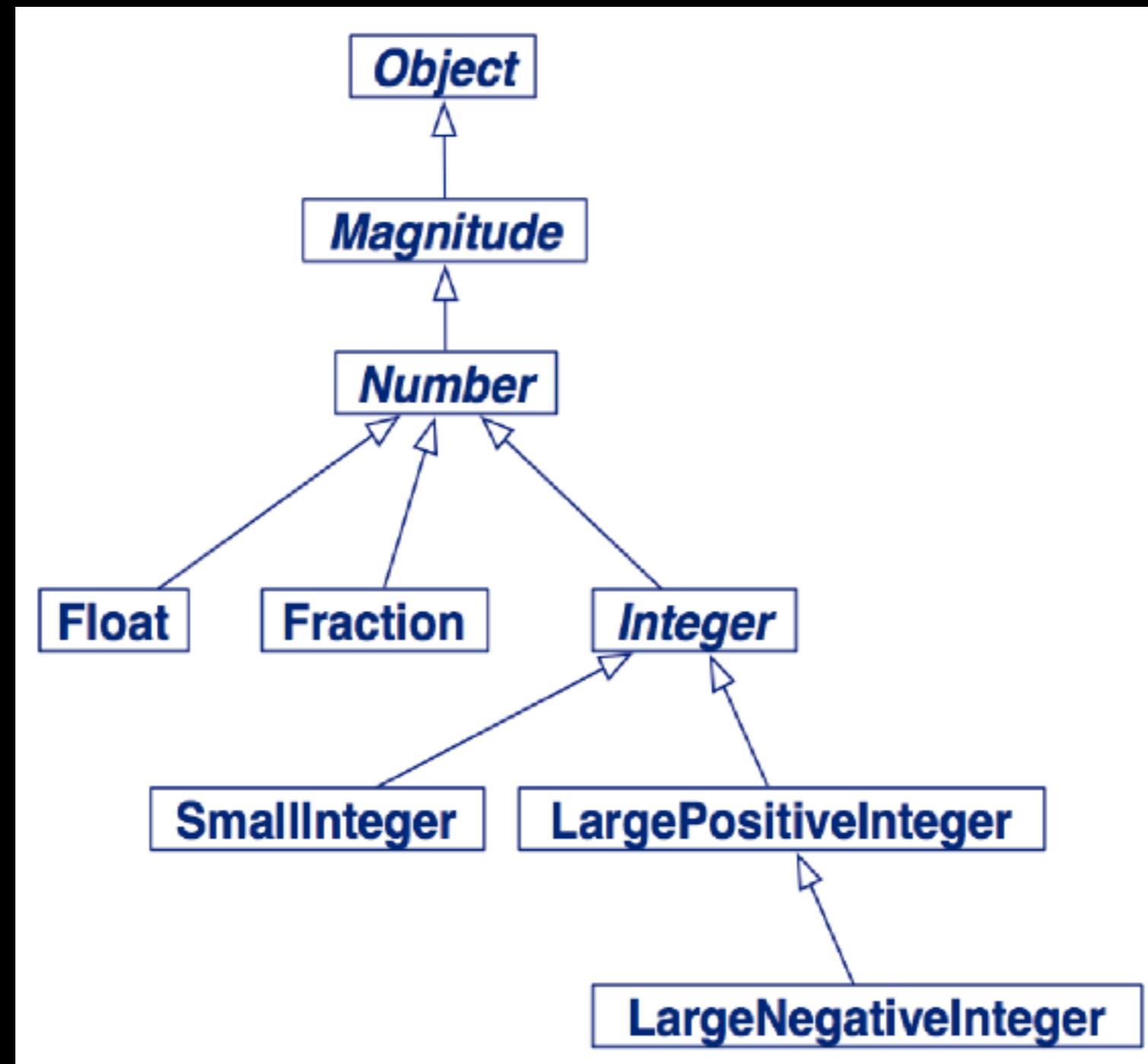
Normalment sobreescrivem **#printOn:** per donar representació textual als objectes. És el mètode per defecte utilitzat en diverses situacions, p.ex. pel **Transcript**



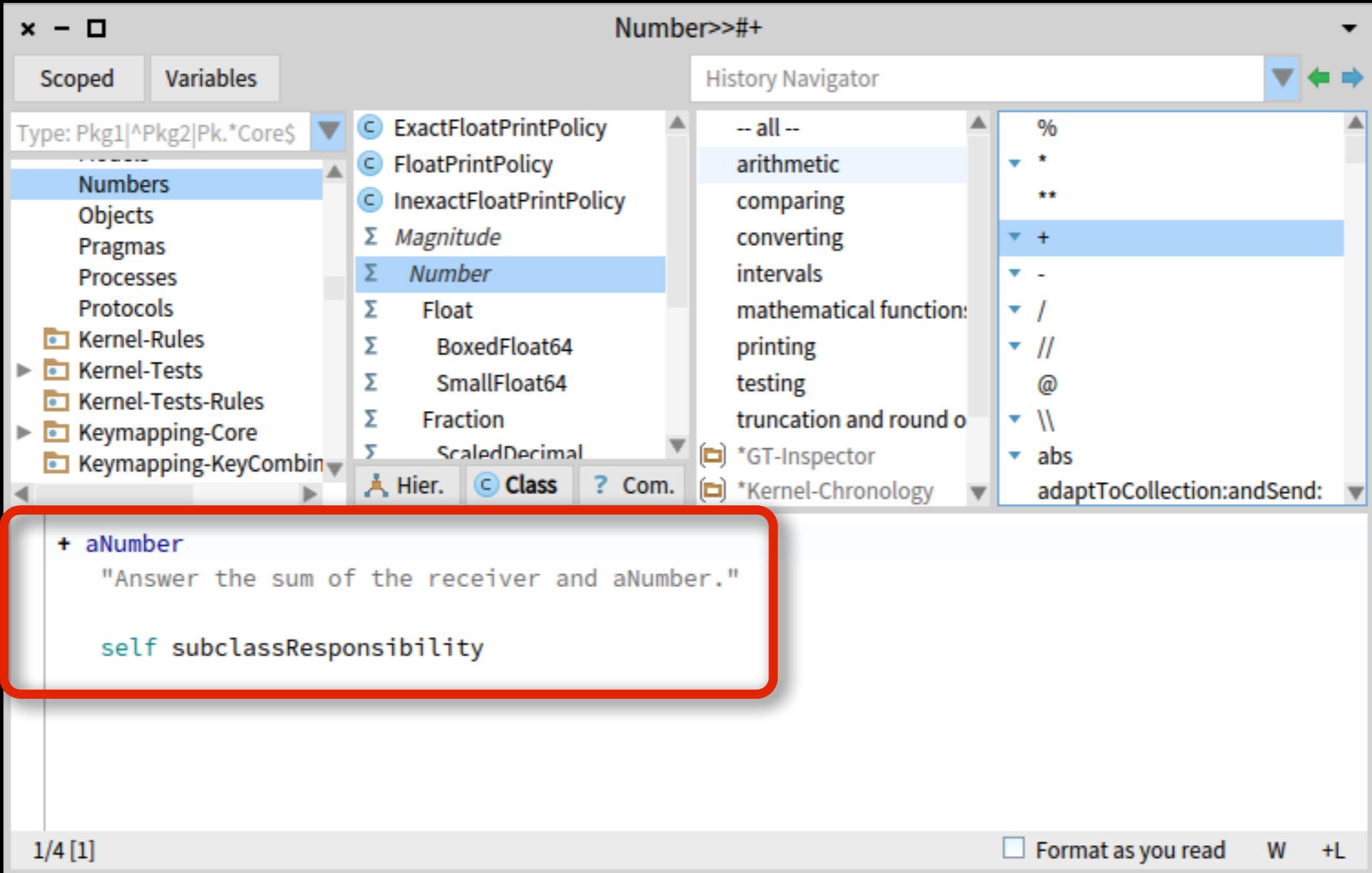
Mètodes d'**Object** per donar suport al programador

#error:	genera un error
#doesNotUnderstand:	gestiona missatges no implementats
#halt, #halt:	invoca el <i>debugger</i>
#subClassResponsibility	el mètode que l'envia és abstracte
#shouldNotImplement	desactiva un missatge heretat

Nombres



Mètodes abstractes (I)



The screenshot shows the Smalltalk History Navigator interface. The title bar says "Number>>#+". The left pane shows a class hierarchy with "Numbers" selected. The right pane lists various methods, with "+>>#" highlighted. A red box highlights the implementation of the '+' method in the Number class.

```
+ aNumber
    "Answer the sum of the receiver and aNumber."
    self subclassResponsibility
```

1/4 [1] Format as you read W +L

Mètodes abstractes (i 2)

The screenshot shows the Smalltalk Object browser interface. The title bar reads "Object>>#subclassResponsibility". The left pane displays a class hierarchy tree under "Type: Pkg1|^Pkg2|Pk.*Core\$". The "Objects" node is selected. The right pane lists various system messages. A red box highlights the "subclassResponsibility" message in the bottom-left corner of the main pane.

Object>>#subclassResponsibility

History Navigator

Type: Pkg1|^Pkg2|Pk.*Core\$

Objects

ProtoObject

Object

Boolean

False

True

UndefinedObject

error handling

evaluating

finalization

flagging

halting

introspection

literal testing

logging-DDeprecated

memory usage

message performing

model - stepping

stonProcessSubObjects:

stonShouldWriteNilInstVars

storeAt:inTempFrame:

storeDataOn:

storeOn:

storeString

subclassResponsibility

systemIcon

systemIconName

systemNavigation

tail

subclassResponsibility

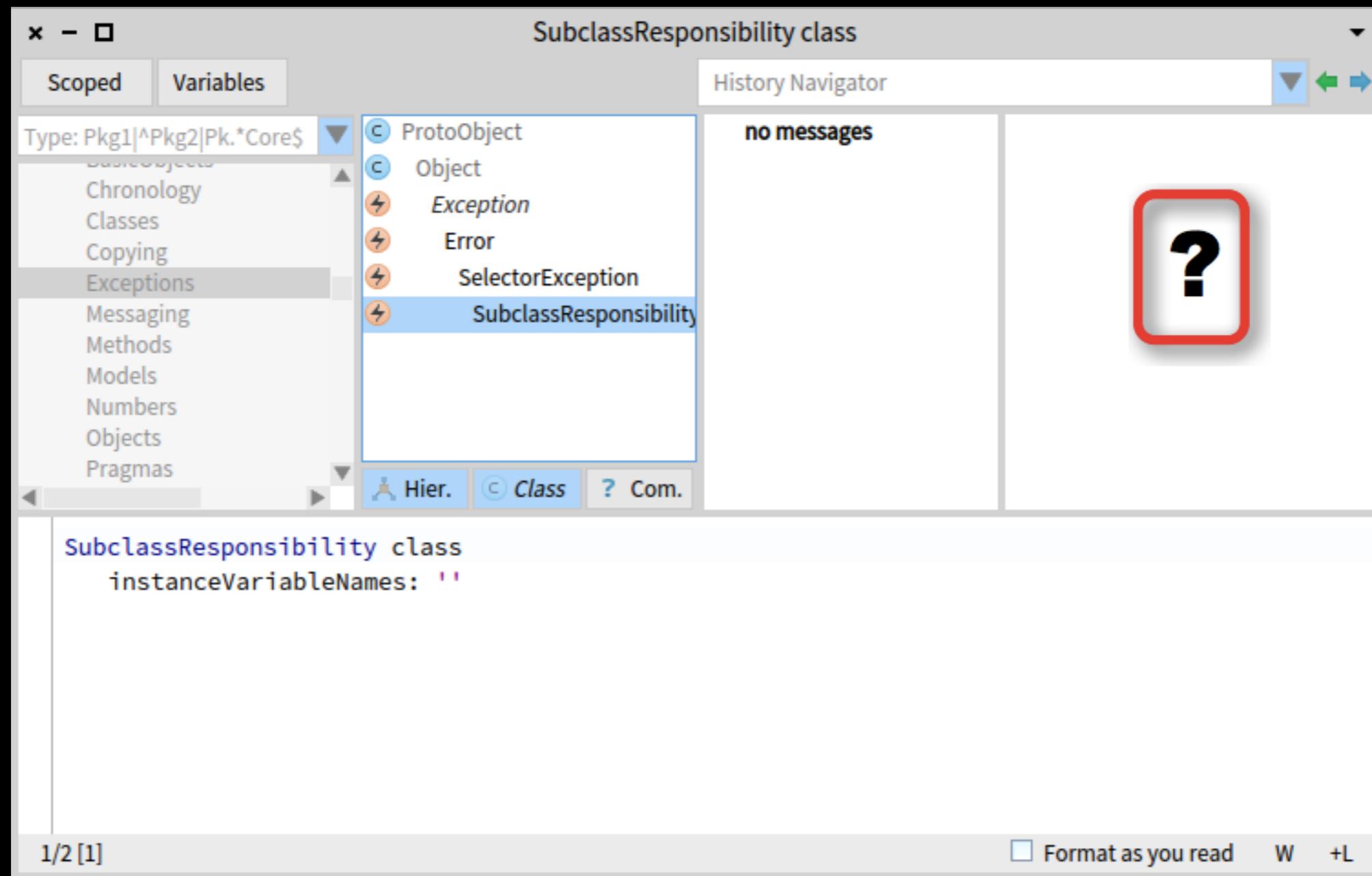
"This message sets up a framework for the behavior of the class' subclasses.
Announce that the subclass should have implemented this message."

SubclassResponsibility signalFor: thisContext sender selector

1/5 [1]

Format as you read W +L

Mètodes abstractes (i 2)



Mètodes abstractes (i 2)

The screenshot shows the Smalltalk History Navigator interface. The title bar reads "SelectorException class>#signalFor:". The left pane lists categories like BasicObjects, Chronology, Classes, Copying, Exceptions, Messaging, Methods, Models, Numbers, Objects, and Pragmas. The right pane shows a list of methods under "signalFor:" and "signalFor:in:", with "signalFor:" being the selected method. The bottom pane displays the source code for the "signalFor:" method, which is highlighted with a red box.

```
signalFor: aSelector
    "Create and signal an exception for aSelector in the default receiver."
    ^ self new
        selector: aSelector;
        signal
```

1/6 [1] Format as you read W +L

Mètodes abstractes (i 2)

The screenshot shows the Smalltalk IDE interface with the following details:

- Title Bar:** SelectorException>>#selector:
- Toolbars:** History Navigator (with buttons for History, Navigator, and Back/Forward).
- Left Panel (Type Browser):** Shows the current type as Pkg1|^APkg2|Pk.*Core\$. Other categories listed include BasicObjects, Chronology, Classes, Copying, Exceptions, Messaging, Methods, Models, Numbers, Objects, and Pragmas.
- Center Panel (Method Browser):** Displays the source code for the #selector: method:

```
selector: aSelector
selector := aSelector
```

The first line is highlighted with a red box.
- Right Panel (History Navigator):** Shows a history of recent changes with items like -- all --, accessing, printing, messageText, selector, selector:, and standardMessageText.
- Status Bar:** Shows 1/2 [1] and Format as you read W +L.

Mètodes abstractes (i 2)

The screenshot shows the Smalltalk Inspector window for the class `Exception>>#signal`. The left pane displays a tree of objects under the type `Pkg1|^Pkg2|Pk.*Core$`, with `Exceptions` selected. The center pane shows the class hierarchy: `ProtoObject`, `Object`, `Exception` (selected), `Error`, `SelectorException`, and `SubclassResponsibility`. The right pane lists various methods. A red box highlights the `signal` method in the bottom-left pane, which contains its definition:

```
signal
    "Ask ContextHandlers in the sender chain to handle this signal. The default is to
execute and return my defaultAction."

    signalContext := thisContext contextTag.
    signaler ifNil: [ signaler := self receiver ].
    ^ signalContext nextHandlerContext handleSignal: self
```

The bottom status bar shows `1/7 [1]`, `Format as you read`, `W`, and `+L`.

Coerció Automàtica

1 + 2.3 → 3.3

1 class → SmallInteger

1 class maxVal class → SmallInteger

(1 class maxVal + 1) class → LargePositiveInteger

(1/3) + (2/3) → 1

1000 factorial / 999 factorial → 1000

2 / 3 + 1 → (5/3)

Caràcters i *Strings*

Caràcters:

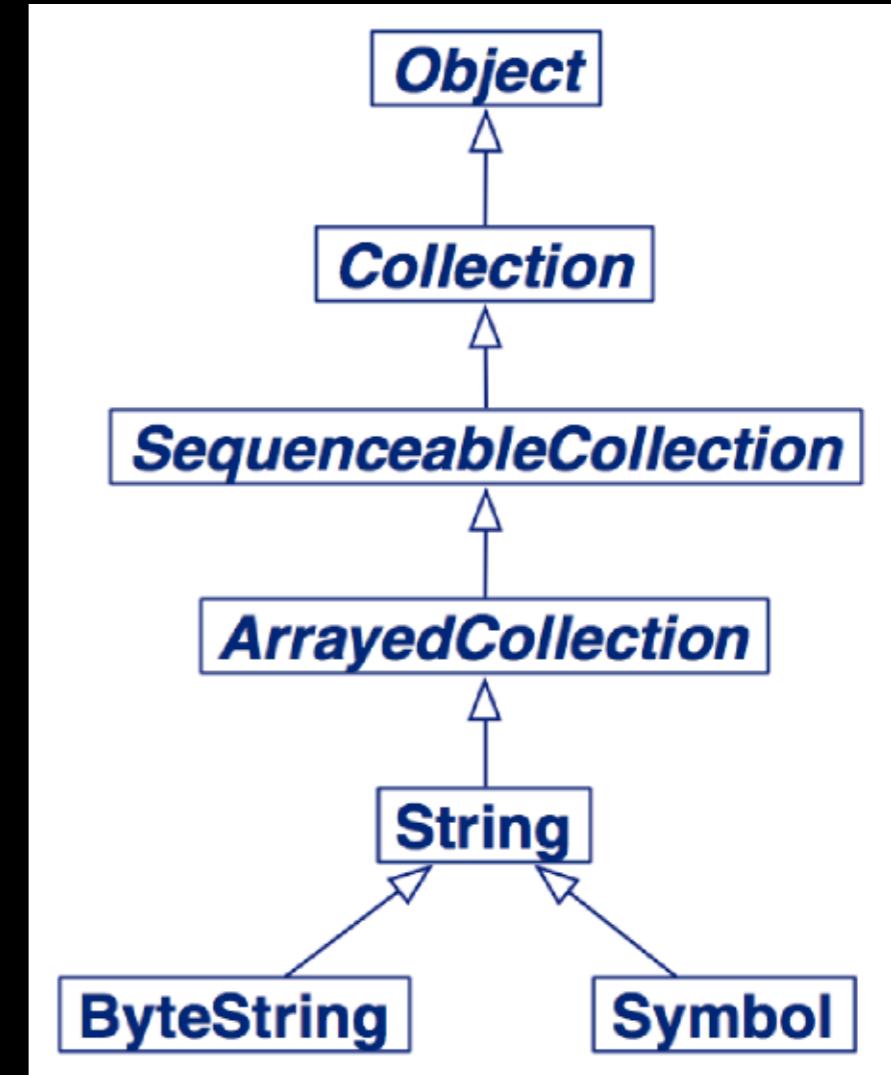
\$a \$F \$,
\$(\$ñ \$4

Caràcters no imprimibles:

Character cr,
Character space,
Character tab

Strings

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



Arrays Literals

`#('hola' #(1 2 3)) → #(‘hola’ #(1 2 3))`

`#{a b c} → #{#a #b #c}`

`#{1+2} → #(1 #+ 2)`

`#(Transcript show: ‘hola’) → #(Transcript #show: ‘hola’)`

Arrays i Arrays Literals

Els *Arrays* i els *Arrays literals* només es diferencien en el moment de ser creats: Els *Arrays literals* es coneixen en *temps de compilació* i en canvi els *Arrays* en *temps d'execució*.

Exemple:

El resultat d'avaluar

```
#(Set new) → #(#Set #new)
```

és un *Array* amb dos símbols (NO és un *Array* amb una instància de **Set**)

El resultat d'avaluar

```
Array with: (Set new) → an Array(a Set())
```

és un *Array* amb un element, una instància de **Set**

Arrays: {} com a *shortcut* per a **Array with:**
(no és estàndard, però a Pharo i a Squeak està implementat)

Exemple:

#(1 + 2 . 3) \Rightarrow #(1 #+ 2 #. 3)

{1 + 2 . 3} \Rightarrow #(3 3)

Array with: 1 + 2 **with:** 3 \Rightarrow #(3 3)

Símbols i *Strings*

Els símbols són utilitzats com a claus úniques per a diccionaris i com a selectors de mètodes. Els símbols són *read-only*, immutables i únics, les *Strings* són mutables i no són úniques.

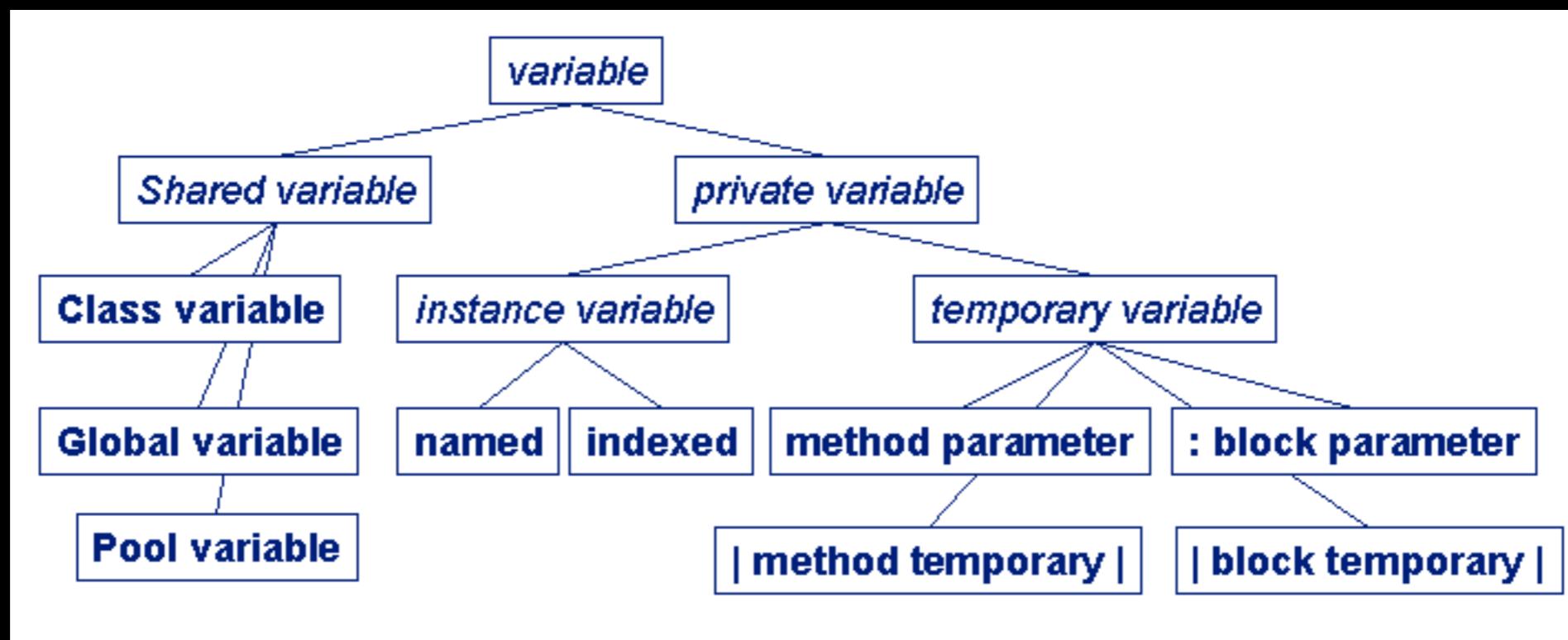
```
'calvin' = 'calvin' → true
'calvin' == 'calvin' → true
'cal','vin' = 'calvin' → true
'cal','vin' == 'calvin' → false
```

```
#calvin = #calvin → true
#calvin == #calvin → true
#cal,#vin = #calvin → true
#cal,#vin == #calvin → false
#cal,#vin → 'calvin'
(#cal,#vin) asSymbol == #calvin → true
```

Variabes

A Smalltalk les variables contenen referències a objectes, per això no cal dir quin tipus tenen (*tipat dinàmic*).

Hi ha una convenció: si el nom d'una variable comença per majúscula, aquesta és *shared*, si no, és local (*private*).



Variables



Assignació

L'assignació lliga (*binds*) un **nom** a una **referència** a un objecte
(això NO es fa amb pas de missatges)

- No podem assignar valors als paràmetres dels mètodes
(no hi ha cap problema, utilitzar variables temporals)
- Noms (variables) diferents poden referenciar el mateix objecte
(*aliasing*)

```
| p1 p2 |
p1 := 3@4. "un Punt"
p2 := p1.
p1 setX: 5 setY: 6.
p2 → 5@6
```

Variables d'Instància

Les variables d'instància definides a una determinada classe són **privades** de l'objecte instància de la classe, tot i que, lògicament, són visibles pels mètodes de la classe on estan definides i pels mètodes de les subclasses d'aquesta classe.

Tenen la mateixa vida que l'objecte instància de la classe on estan definides.

S'aconsella crear *getters* i *setters* per a cada variable d'instància i mai accedir-ne directament (per encapsular inicialitzacions i manipulacions). Poseu-los dins d'un protocol anomenat *private*.

Pseudo-variables

Definides al compilador, formen part de l'estàndard Smalltalk-80

nil → referència a l'**UndefinedObject**

true → instància única de la classe **True**

false → instància única de la classe **False**

self → referència a l'objecte al que pertany el mètode que s'està executant

super → referència a l'objecte al que pertany el mètode que s'està executant, *PERÒ* la cerca del mètode comença a la superclasse de la classe on està *definit* el mètode que envia el missatge a **super**

thisContext → reificació del context d'execució

Blocs (*closures*)

- Un bloc (o *closure*) es pot entendre com un fragment de codi que capture el seu context lèxic en el moment de la creació.
- A Smalltalk els blocs són valors qualsevol, se'ls pot passar com paràmetre de missatges, guardar en variables, etc. - són valors de *primera classe*.
- S'utilitzen per *ajornar* l'avaluació de codi.
- Sintaxi:

```
[ :arg1 :arg2 ... |
| temp1 temp2 ... |
| expressio1.
| expressio2. ... ]
```
- Un bloc retorna el valor de la **darrera expressió** (avaluada) del bloc.

Blocs (*closures*)

- Per avaluar un bloc cal enviar el missatge **#value**, **#value:**, **#value:value:**, etc. dependent dels paràmetres que tingui el bloc.

Exemple:

```
|sqr|  
sqr := [ :n | n*n ].  
sqr value: 5 → 25
```

Recordem que un bloc retorna el valor de la **darrera expressió** (avaluada) del bloc.

- Tenim fins a quatre **#value:value:value:value:**, per a més de quatre paràmetres cal utilitzar **#valueWithArguments:**, al que se li passa un **Array** amb els arguments.

Blocs (*closures*)

Més exemples:

```
[2+3+4+5] value → 14
```

```
[:x | x+3+4+5] value:2 → 14
```

```
[:x :y | x+y+4+5] value:2 value:3 → 14
```

```
[:x :y :z | x+y+z+5] value:2 value:3 value:4 → 14
```

```
[:x :y :z :w | x+y+z+w] value:2 value:3 value:4 value:5 → 14
```

Blocs (*closures*)

The screenshot shows the Smalltalk IDE interface with three main windows:

- History Navigator**: Shows the method `ProvaClosures class>>#provaClosures`. The class browser pane lists classes like Pasta, PastaTest, ProvaClosures, and ProvaSelfSuper. The code pane displays the implementation of `provaClosures`:

```
| comptador resetComptador incComptador valorComptador |
comptador := 0.
resetComptador := [ comptador := 0. comptador ].
incComptador := [ :i | comptador := comptador + i. comptador ].
valorComptador := [ comptador ].
^ { resetComptador . incComptador . valorComptador }.
```

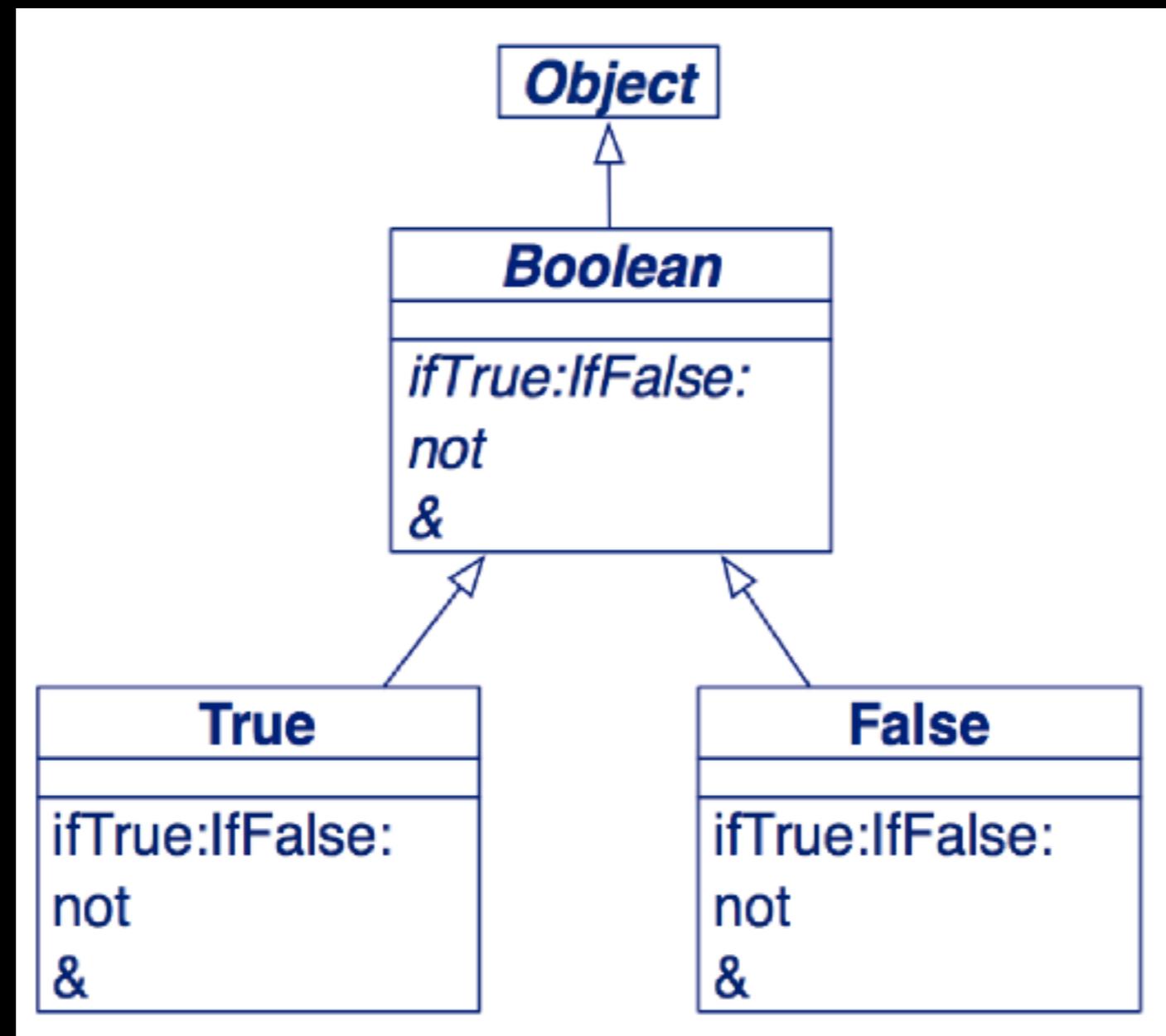
- Playground**: Shows the execution of the closure returned by `provaClosures`. The transcript pane shows the following output:

```
| r1 r2 i1 i2 v1 v2 tmp1 tmp2 |
tmp1 := ProvaClosures provaClosures.
r1 := tmp1 at: 1.
i1 := tmp1 at: 2.
v1 := tmp1 at: 3.
r1 value.
i1 value: 1.
v1 value traceCr.
i1 value: 3.
v1 value traceCr.
tmp2 := ProvaClosures provaClosures.
r2 := tmp2 at: 1.
i2 := tmp2 at: 2.
v2 := tmp2 at: 3.
r2 value.
i2 value: 5.
v2 value traceCr.
v1 value traceCr.
```

- Transcript**: Shows the final values printed by the code:

```
1
4
5
4
```

Booleans



Booleans

The screenshot shows the Smalltalk IDE interface with the following details:

- Title Bar:** True>>#ifTrue:ifFalse:
- Toolbars:** History Navigator.
- Left Panel (Type Browser):** Shows the current type as Pkg1|^Pkg2|Pk.*Core\$. It lists categories: Classes, Copying, Exceptions, Messaging, Methods, Models, Numbers, Objects (which is selected), Pragmas, Processes, and Protocols. Under Objects, it lists ProtoObject, Object, Boolean, False, and True (which is highlighted).
- Right Panel (Message Browser):** Shows the implementation of the ifTrue:ifFalse: message. The code is:

```
ifTrue: trueAlternativeBlock ifFalse: falseAlternativeBlock
    "Answer with the value of trueAlternativeBlock. Execution does not
     actually reach here because the expression is compiled in-line."
    ^trueAlternativeBlock value
```
- Bottom Status Bar:** Shows 1/5 [1] and various toolbar icons.

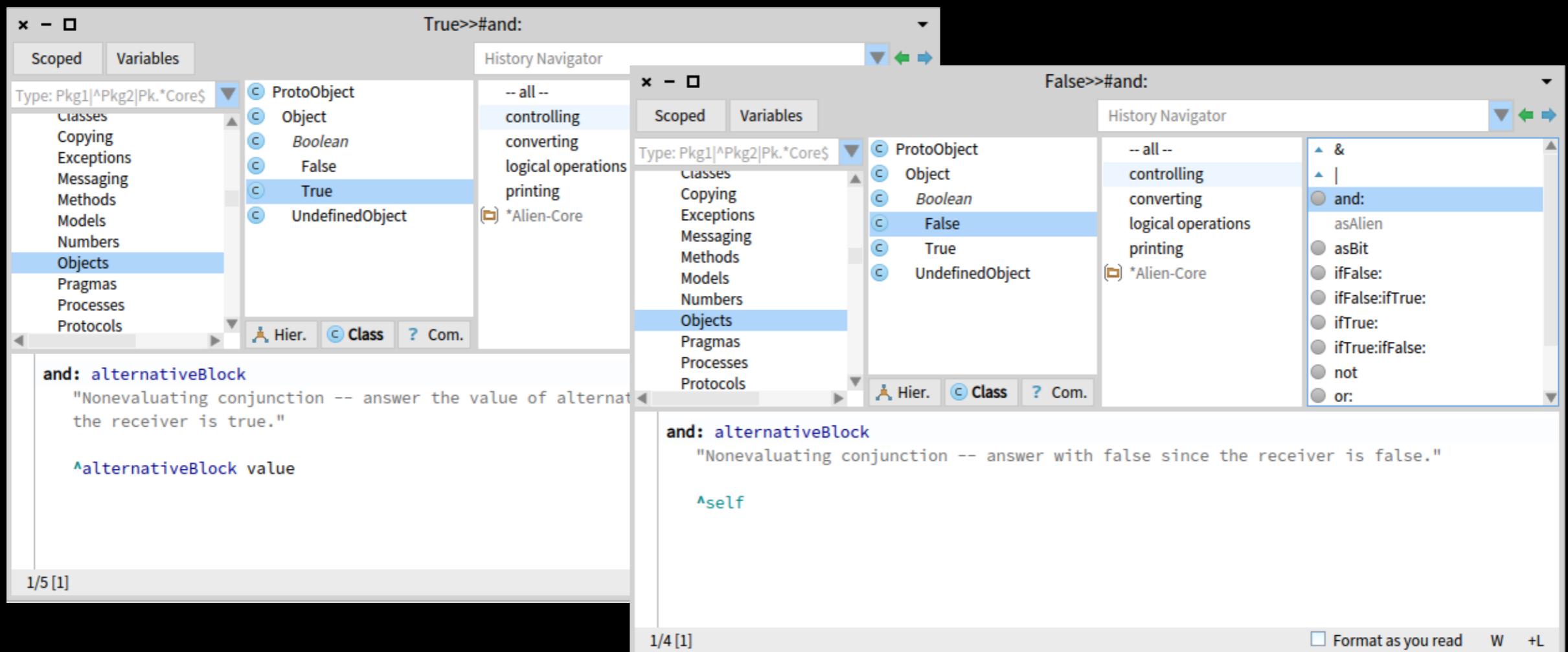
A red box highlights the message implementation code.

Com implementariu **not**, **and:**, **or:**, **&** etc...?

true i false

true i **false** són les úniques instàncies (*singletons*) de les classes **True** i **False**, respectivament.

and: i **or:** no avaluen tots els seus arguments, depén del valor del booleà a qui enviem el missatge.



Iteracions

Diversos tipus d'iteracions:

```
| n |
n := 1.
[ n <= 10 ] whileTrue:
    [ Transcript show: n asString; cr.
      n := n + 1 ]

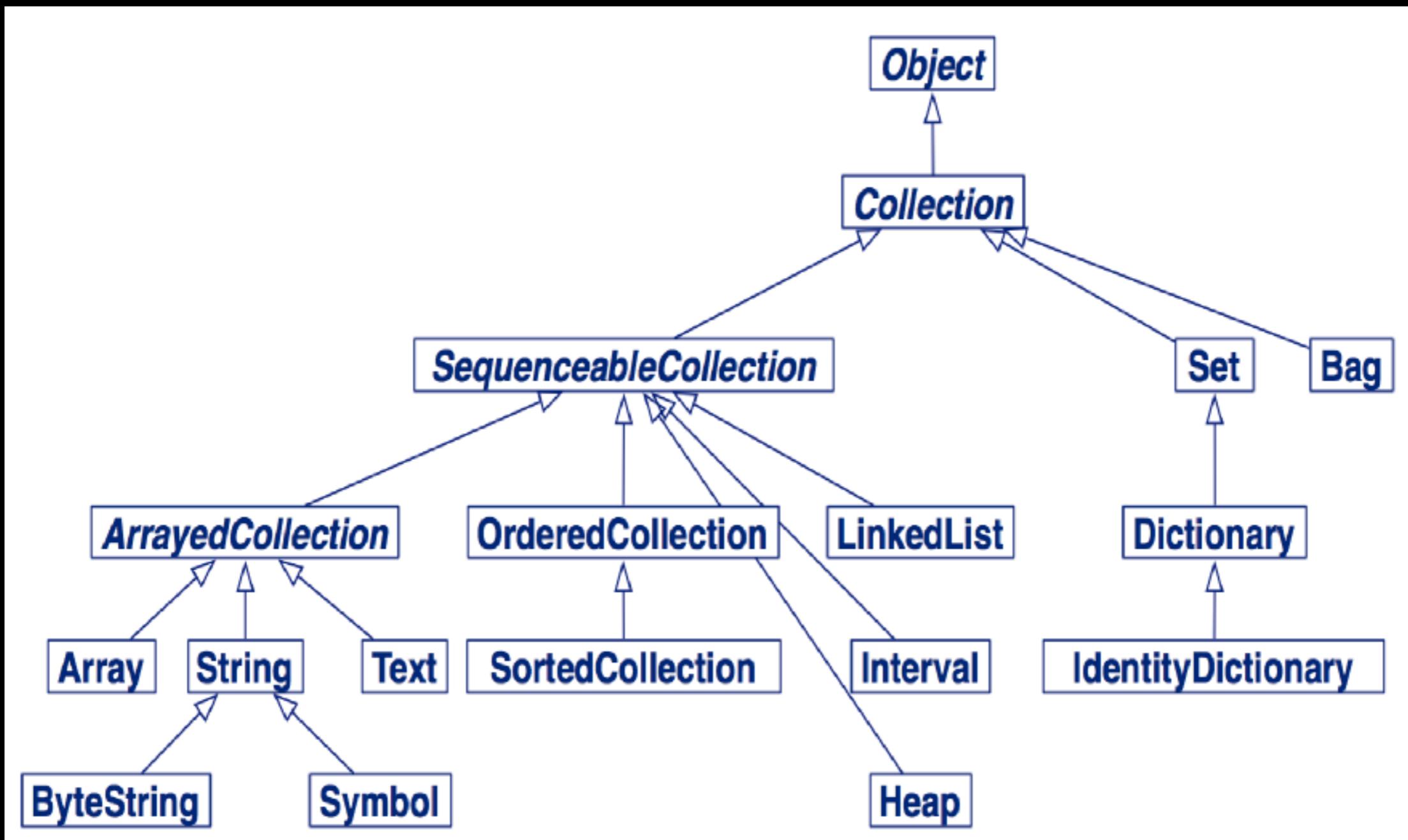
1 to: 10 do: [ :n | Transcript show: n asString; cr]

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

10 timesRepeat: [ Transcript show: '1'; cr]
```

Penseu, en cada cas, quin és l'objecte receptor?

Col.leccions



Col.leccions

- La jerarquia que penja de **Collection** ofereix moltes de les classes més útils d'un sistema Smalltalk. Un bon consell és, *resistiu-vos de programar les vostres pròpies col.leccions!* Segur que Smalltalk en té alguna que us farà el servei que us cal.
- Criteris de classificació:
 - Accés:** *indexat, seqüencial o basat en claus*
 - Mida:** *fixada o dinàmica*
 - Tipus d'element:** *fixat o arbitrari*
 - Ordre:** *definible o cap*
 - Duplicates:** *possibles o no*

Col·leccions

Sequenceable

ordenada

ArrayedCollection

mida fixada + índex enter
qualsevol mena d'element
elements → caràcters

Array

elements → enters
progressió aritmètica

String

encadenament dinàmic

IntegerArray

mida dinàmica + ordre d'arribada

Interval

ordre explícit

LinkedList

cap ordre + permet duplicats

OrderedCollection

cap ordre + no duplicats

SortedCollection

elements → associacions

Bag

identificació basada en la identitat

Set

Dictionary

IdentitySet

Accés → `#size #capacity #at: #at:put:`

Test → `#isEmpty #includes: #contains:`
`#occurrencesOf:`

Afegir → `#add: #addAll:`

Eliminar → `#remove: #remove:ifAbsent: #removeAll:`

Enumerar → `#do: #collect: #select: #detect:`
`#detect:ifNone: #reject: #inject:into:`

Convertir → `#asBag #asSet #asOrderedCollection`
`#asSortedCollection #asSortedCollection:`

Crear → `#with: #with:with: #with:with:with:`
`#with:with:with:with: #withAll:`

Col·leccions

Alguns exemples:

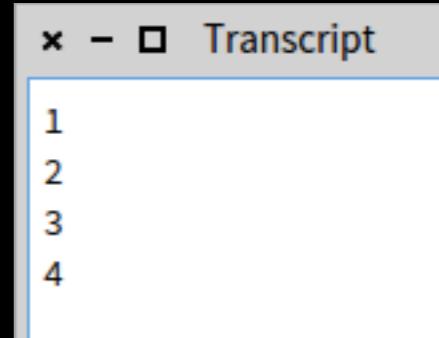
The image displays two side-by-side screenshots of a Smalltalk playground window titled "Playground".

Left Screenshot: Shows code defining an array. The code is:| exemple |
exemple := #(a b c). "Array estàtic compost per simbols"
exemple at: 2 put: #bb.
exemple.The result of the last message is displayed below in a yellow-highlighted box: `#(#a #bb #c)`.

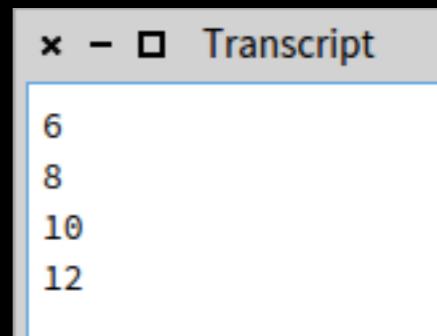
Right Screenshot: Shows code defining a dictionary. The code is:| dict |
dict := Dictionary new.
dict at: 'foo' put: 3.
dict at: 'bar' ifAbsent: [4].
dict at: 'bar' put: 5.
dict removeKey: 'foo'.
dict keys.The result of the last message is displayed below in a yellow-highlighted box: `#('bar')`.

Col·leccions: Alguns missatges comuns

```
#(1 2 3 4) includes: 5 → false
 #(1 2 3 4) size → 4
 #(1 2 3 4) isEmpty → false
 #(1 2 3 4) contains: [:some | some < 0] → false
 #(1 2 3 4) do: [:each | Transcript show: each; cr] →
```



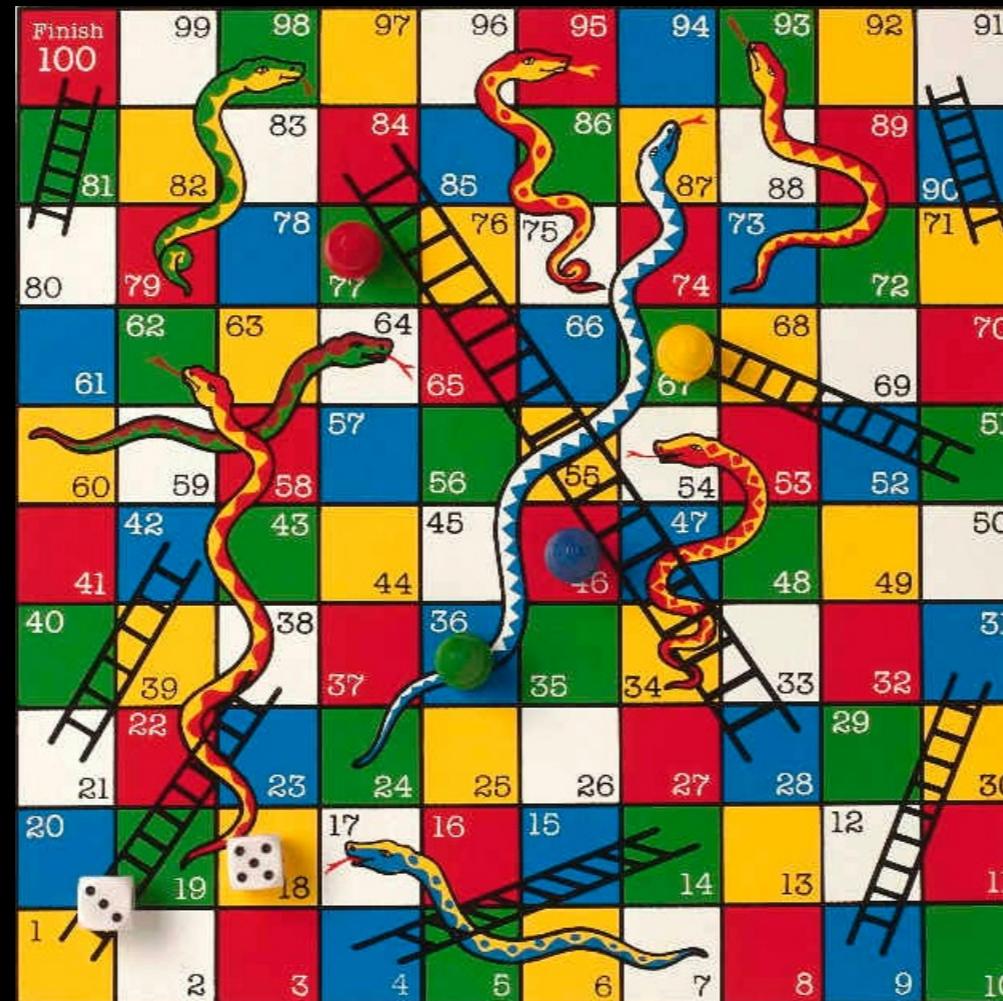
```
#(1 2 3 4) with: #(5 6 7 8)
    do: [:x :y | Transcript show: x+y; cr] →
```



```
#(1 2 3 4) select: [:each | each odd] → #(1 3)
 #(1 2 3 4) reject: [:each | each odd] → #(2 4)
 #(1 2 3 4) detect: [:each | each odd] → 1
 #(1 2 3 4) collect: [:each | each even] → #(false true false true)
 #(1 2 3 4) inject: 0
    into: [:sum :each | sum + each] → 10
```

CAP: Introducció a l'Smalltalk

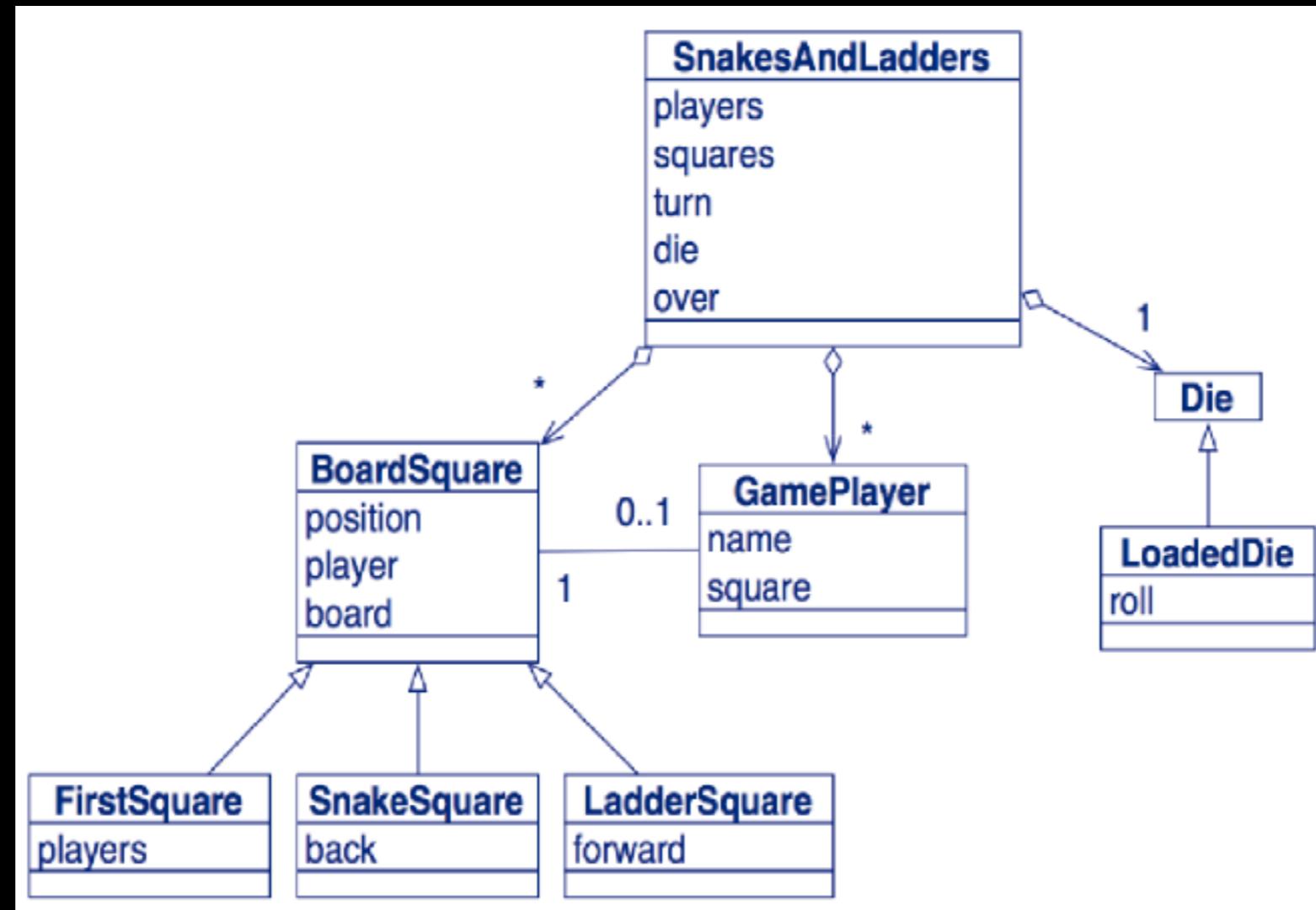
Per posar en context el que explicarem a partir d'ara prendrem com a exemple conductor el joc de *Snakes and Ladders*



Us passaré el fitxer **.st** pel Racó, només us caldrà fer **FileIn entire file**

CAP: Introducció a l'Smalltalk

Aquest diagrama de classes mostra l'estructura del codi de *Snakes and Ladders*



Exemple: Cas d'ús

The screenshot shows the Pharo Smalltalk IDE interface. The title bar reads "SnakesAndLadders class>#example". The left pane displays a package browser with categories like EpiceaBrowsers, EpiceaBrowsersTests, EpicealnPharo60, EpiceaTests, Examples-SnakesAndLadder, FFI-Kernel, FFI-Pools, and FileSystem-Core. The right pane shows a history navigator with items: "example", "documentation", and "example2". The main workspace contains the following code:

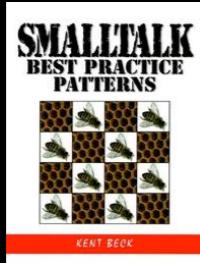
```
example
"self example playToEnd"

^ (self new)
    add: FirstSquare new;
    add: (LadderSquare forward: 4);
    add: BoardSquare new;
    add: BoardSquare new;
    add: BoardSquare new;
    add: BoardSquare new;
    add: (LadderSquare forward: 2);
    add: BoardSquare new;
    add: BoardSquare new;
    add: BoardSquare new;
    add: (SnakeSquare back: 6);
    add: BoardSquare new;
    join: (GamePlayer named: 'Jack');
    join: (GamePlayer named: 'Jill');
yourself
```

A yellow speech bubble points to the word "yourself" in the last line of the code, which is highlighted with a red oval.

Veiem una gran cascada i yourself, que no sabem què és

1/19 [1] Format as you read W +L



Quin format donem a múltiples missatges enviats al mateix receptor?

Utilitzeu una cascada. Separeu els missatges amb un ‘;’. Poseu cada missatge en una línia separada i sagneu un tabulador.

Utilitzeu cascades només per missatges amb un argument, com a molt.



Com podeu utilitzar el valor d'una cascada si el darrer missatge no retorna el receptor del missatge?

Acabeu la cascada amb el missatge yourself

A screenshot of the Smalltalk Object browser interface. The title bar says "Object>>#yourself". The left pane shows a class hierarchy tree with "Objects" selected. The center pane lists methods starting with "accessing", and the method "yourself" is highlighted. The right pane shows the definition of "yourself".

Object>>#yourself

History Navigator

Object>>#yourself

Method list:

- all --
- accessing
- asserting
- associating
- binding
- block support
- casing-To be deprecated
- class membership
- comparing
- converting
- wantsVisualFeedback
- when:evaluate:
- when:send:to:
- when:send:to:with:
- when:send:to:withArguments:
- windowIsClosing
- withoutListWrapper
- writeSlot:value:
- writeSlotNamed:value:
- yourself

Method definition:

```
yourself
"Answer self. The message yourself deserves a bit of explanation (...)"
```

Aself

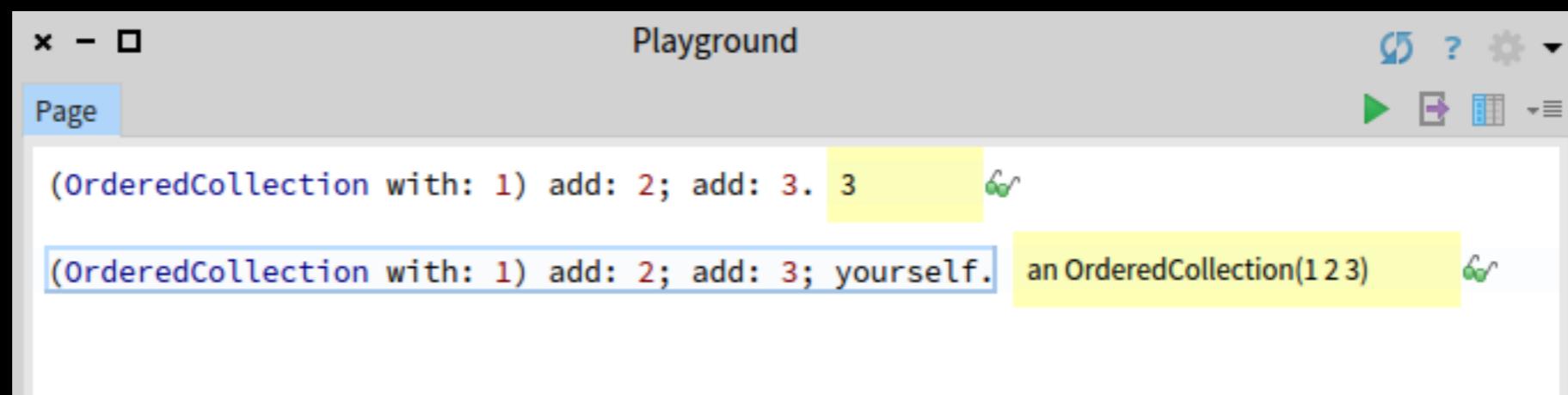
3/4 [1] Format as you read W +L

Sobre **yourself...**

- L'efecte d'una cascada és enviar *tots* els missatges al *receptor* del primer missatge de la cascada:

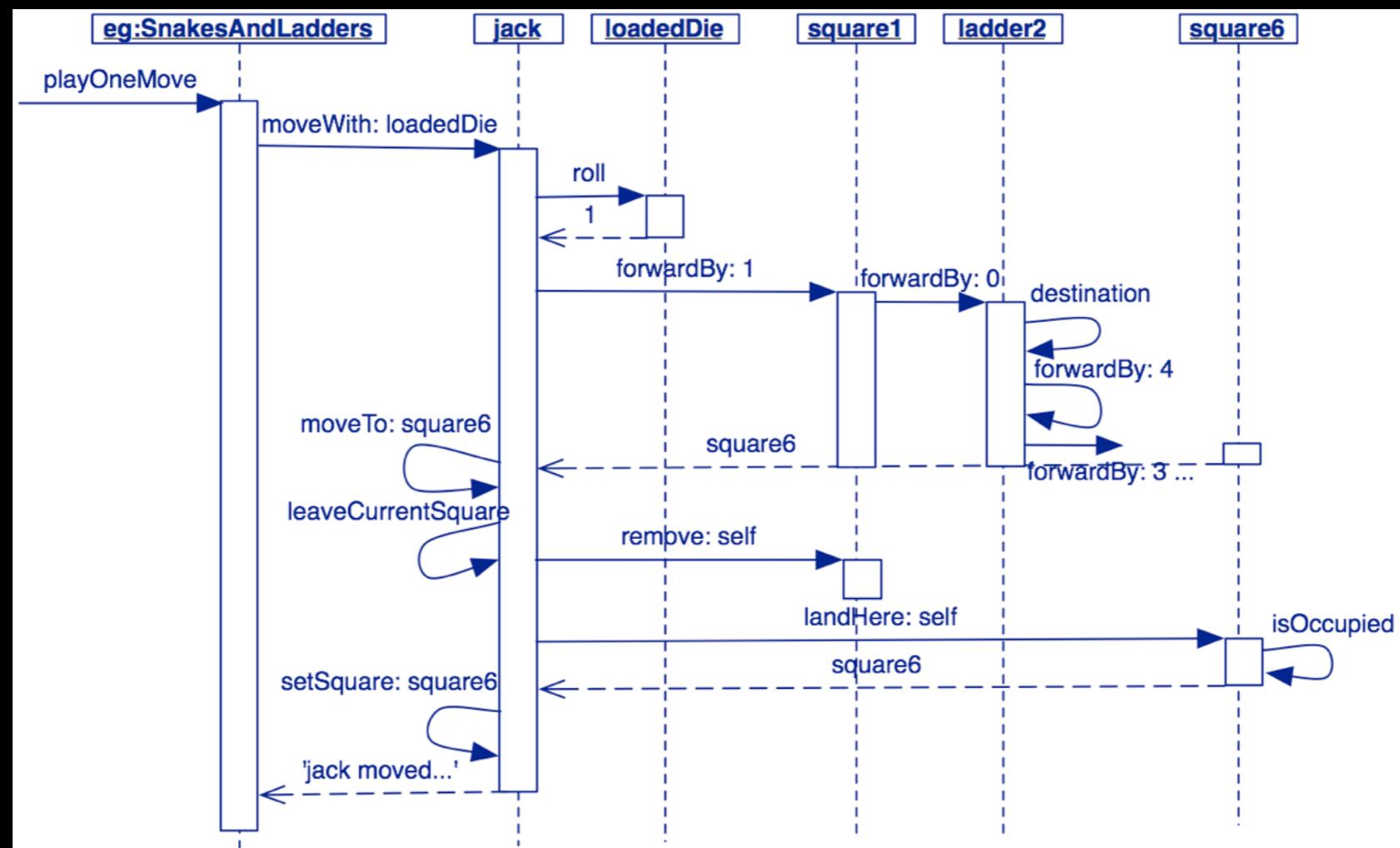
```
self new ←  
add: FirstSquare new;  
add: (LadderSquare forward: 4);  
....
```

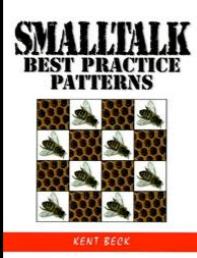
- El valor retornat per una cascada és el valor retornat pel darrer missatge enviat. Per aconseguir el *receptor* com a resultat de la cascada, hem d'enviar el missatge **yourself**.



Distribuint responsabilitats...

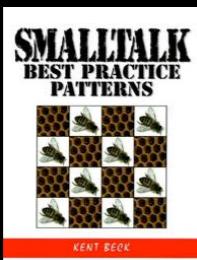
En un sistema orientat a objectes ben dissenyat típicament trobarem molts mètodes *petits*, amb el nom triat amb cura. Això promou interfícies fluides, reutilitzables i amb facilitat de manteniment.





Un cop i només un

En un programa escrit en bon estil, tot el que cal dir es diu un sol cop.



Un munt de peces petites

El bon codi té invariablement mètodes petits i objectes petits. Només factoritzant el sistema en moltes peces petites d'estat i funció podem esperar satisfer la regla ‘un cop i només un’

Herència a Smalltalk

- Herència SIMPLE

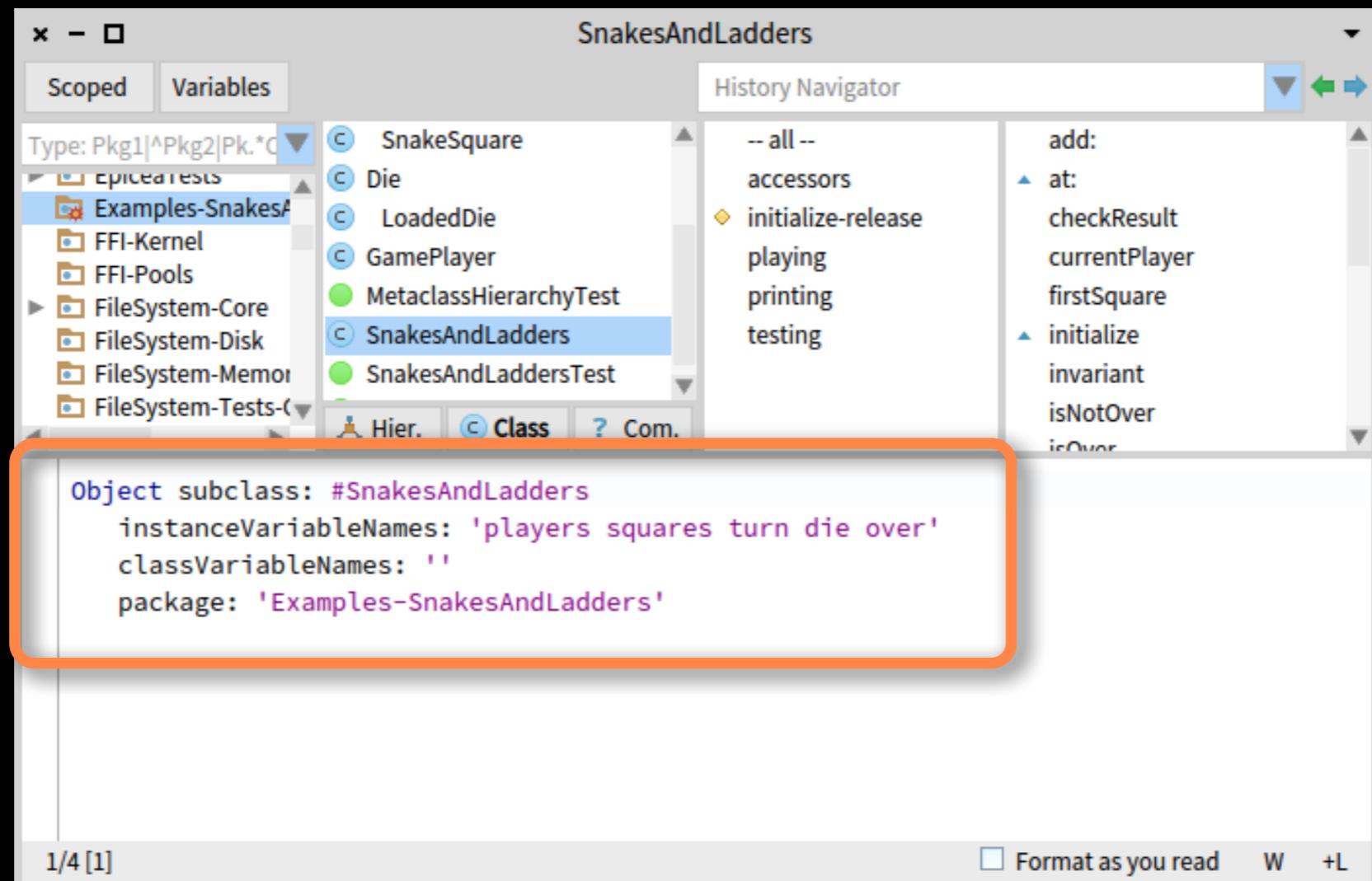
- Estàtica per a les variables d'instància.

Les variables d'instància d'un objecte són aquelles variables definides en la classe de la que l'objecte és instància, a més de totes les variables d'instància de les superclasses corresponents.

- Dinàmica pels mètodes.

Els mètodes corresponents als missatges enviats a un objecte (el receptor) es busquen *en temps d'execució*, en funció de la classe del receptor.

Recordem que crear classes és només enviar missatges a altres classes



**#subclass:instanceVariableNames:classVariableNames:package:
és un missatge amb 4 paràmetres!**

Variables d'instància (amb nom)

- El seu nom comença amb una lletra *minúscula*.
- Cal declarar-les *explícitament* quan es crea la classe.
- El seu nom ha de ser *únic* dins la cadena d'herència
- El seu valor per defecte és **nil**
- Poden accedir-hi *tots* els mètodes de la classe *i de les subclasses*
- No hi poden accedir *els mètodes de classe*
- Els clients han d'utilitzar mètodes d'accés per consultar i/o modificar una variable d'instància.

Consell de Disseny: No accedir mai directament a les variables d'instància d'una superclasse des dels mètodes d'una subclass. Així evitarem vincles forts entre classes.

Com inicialitzem objectes?

- **Problema:** Per crear una nova instància d'una classe hem d'enviar el missatge **new** a la classe. Els mètodes de classe, però, no poden accedir a les variables d'instància.

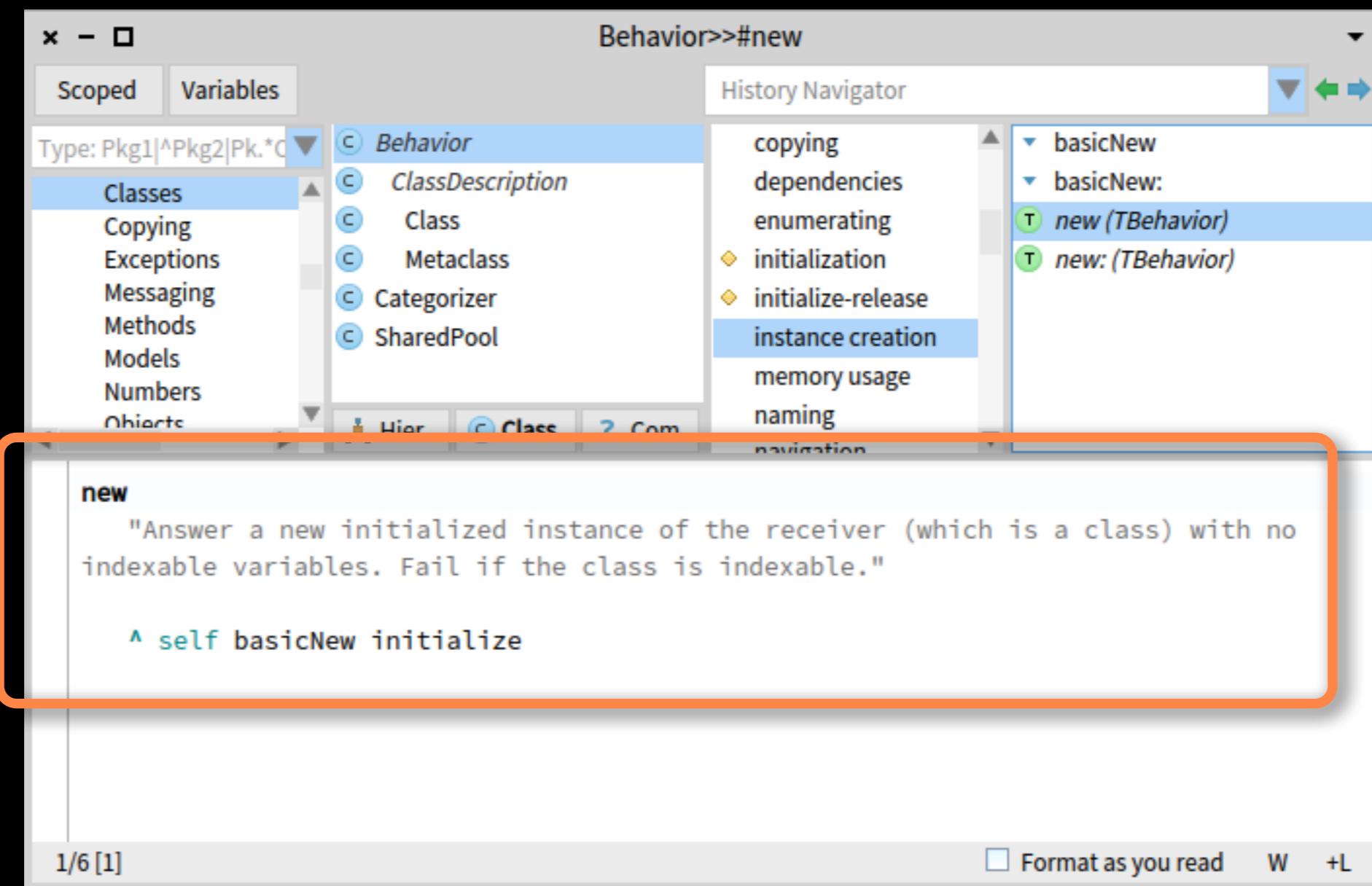
Com ho fem per inicialitzar els valors de les variables d'instància?

- **Solució:** Proporcionar un mètode (*d'instància*) anomenat **initialize** (s'aconsella utilitzar el protocol **initialize-release**) que inicialitzi les variables d'instància com calgui.

*Aquest mètode sempre es crida en crear una classe amb **new***

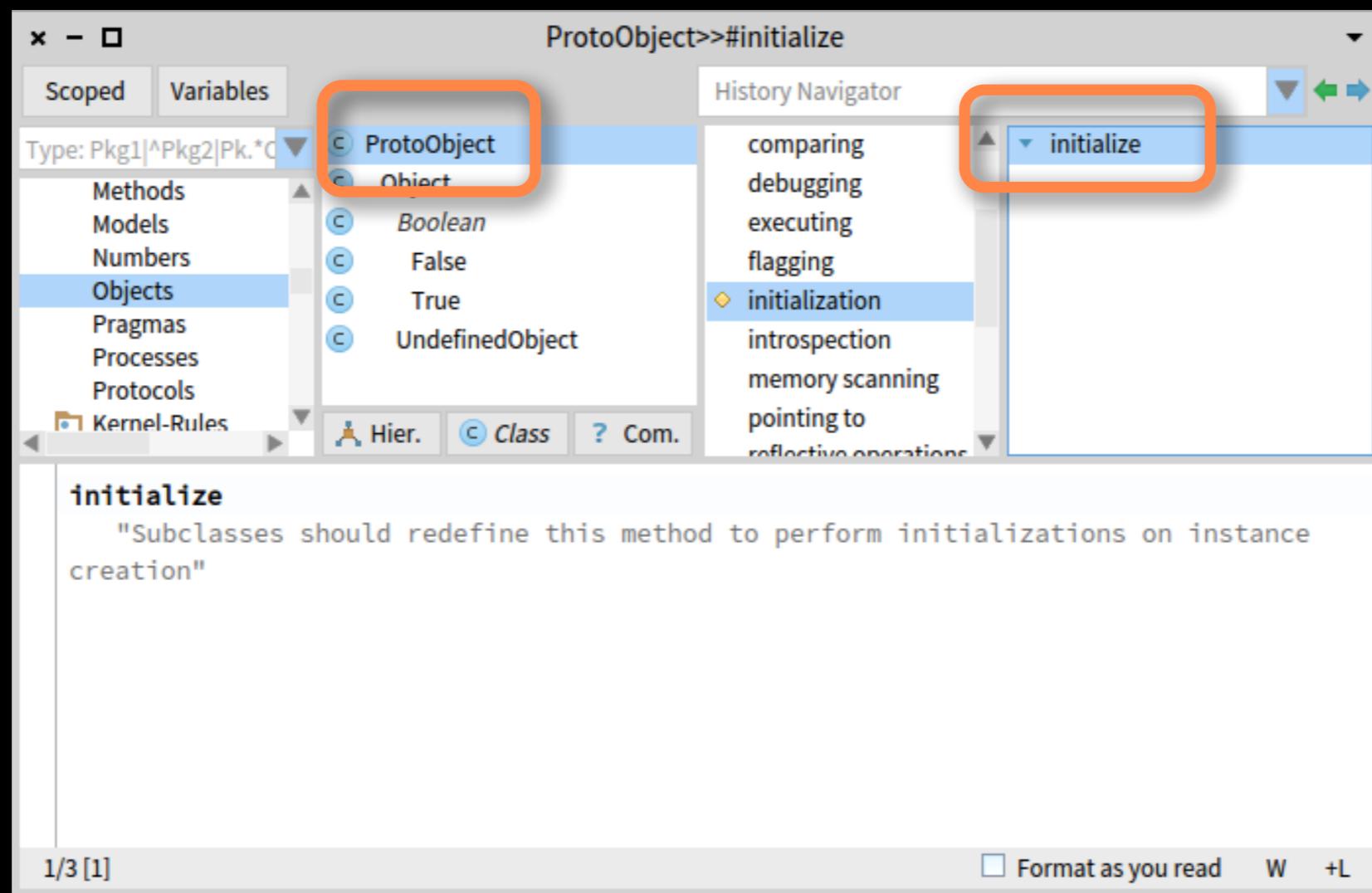
Com inicialitzem objectes?

Aquest mètode sempre es crida en crear una classe amb **new**



Com inicialitzem objectes?

I si no en definim cap, de mètode **initialize**?



No passa res! Sempre en trobarà un a l'arrel de la jerarquia de classes

Com inicialitzem objectes?

Exemple d'initialize

The screenshot shows the Pharo Smalltalk IDE interface. The top title bar says "SnakesAndLadders>>#initialize". The left pane is a browser showing a list of classes and packages. The "Examples-SnakesAndLadders" package is selected. The "SnakesAndLadders" class is highlighted in the list. The bottom pane displays the source code for the initialize method:

```
initialize
    super initialize.
    die := Die new.
    squares := OrderedCollection new.
    players := OrderedCollection new.
    turn := 1.
    over := false.
```

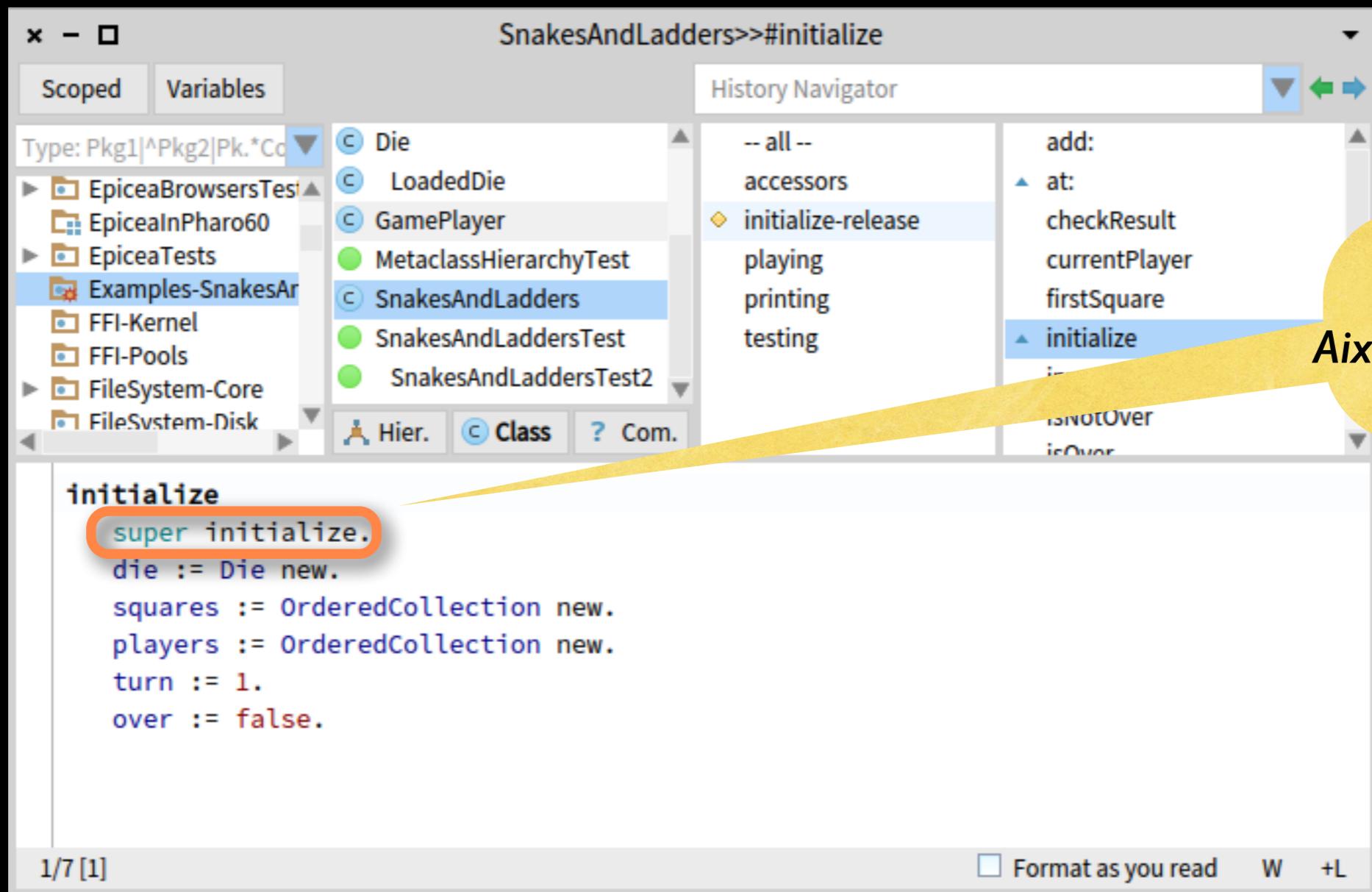
The right pane is a "History Navigator" showing a list of messages related to initialization:

- all --
- accessors
- initialize-release
- playing
- printing
- testing
- add:
- at:
- checkResult
- currentPlayer
- firstSquare
- initialize
- invariant
- isNotOver
- isOver

At the bottom of the interface, there is a status bar with "1/7 [1]" and "Format as you read W +L".

Com inicialitzem objectes?

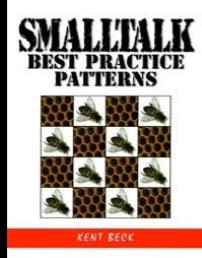
Exemple d'initialize



The screenshot shows the Pharo Smalltalk IDE interface. The title bar says "SnakesAndLadders>>#initialize". The left pane shows a browser with various packages and classes listed. The "Class" tab is selected. In the center, the code for the `initialize` method is displayed:

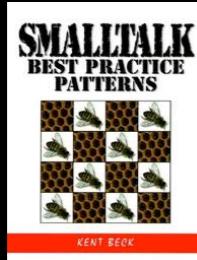
```
initialize
    super initialize.
    die := Die new.
    squares := OrderedCollection new.
    players := OrderedCollection new.
    turn := 1.
    over := false.
```

The line `super initialize.` is highlighted with a red rectangle. To the right, the "History Navigator" pane shows categories like "initialize-release", "add:", "at:", etc. A yellow callout bubble points from the "initialize-release" category to the highlighted line of code. The callout contains the text "Atenció!! Això és freqüent".



Com podem utilitzar col·leccions la mida de les quals NO és coneguda en el moment de crear-les?

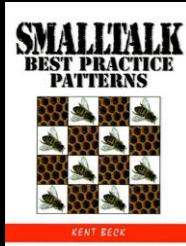
*Utilitzeu **OrderedCollection** com la col·lecció de mida dinàmica per defecte.*



Com podem representar la creació d'instàncies?

*Proporcioneu mètodes de classe, dins el protocol **instance creation**, que construeixin instàncies ben formades. Passeu tots els paràmetres que siguin necessaris.*

The image displays two side-by-side Smalltalk environments. Both environments show the 'History Navigator' tab selected. In the left environment, the protocol is 'LadderSquare class>>#forward:' and the selected step is 'instance creation'. In the right environment, the protocol is 'SnakeSquare class>>#back:' and the selected step is also 'instance creation'. Both environments show a list of classes including BoardSquare, FirstSquare, LadderSquare, SnakeSquare, Die, LoadedDie, and GamePlayer. The 'GamePlayer' class is highlighted in both. Below the protocols, the implementation code is visible. In the bottom right corner of each environment, there is a button labeled 'Format as you read'.



Com podeu donar valor a les variables d'instància a partir dels paràmetres del mètode constructor?

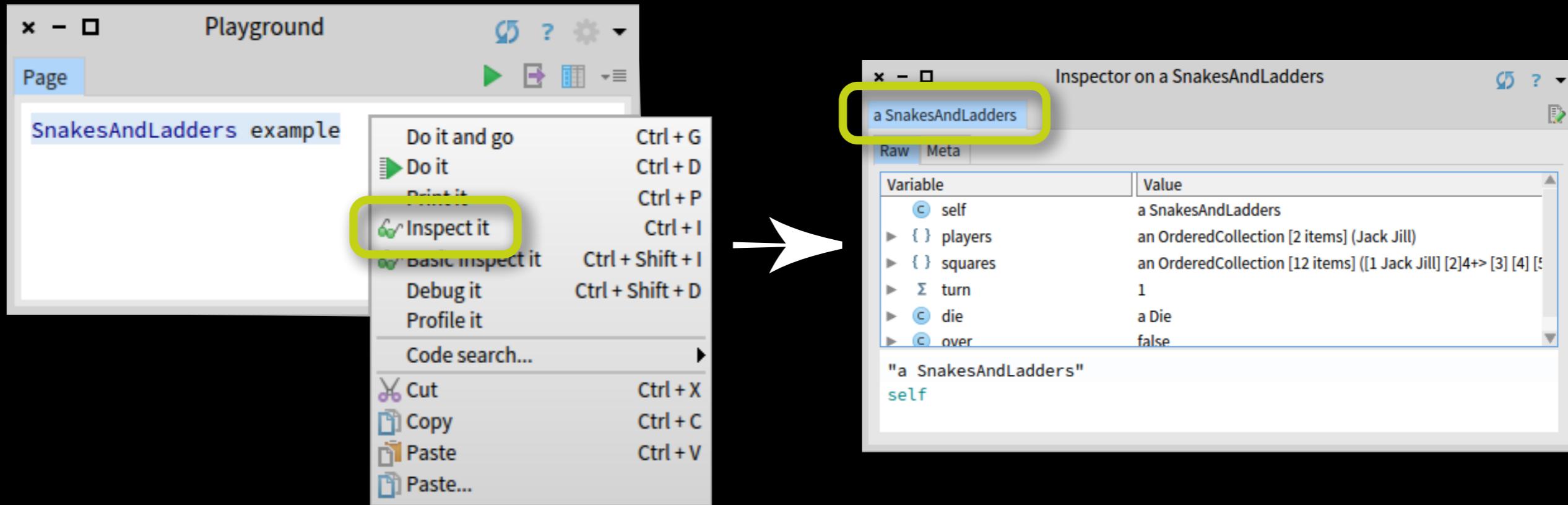
Proporcioneu setters anomenats *set* i després el nom de la (les) variable(s)

A screenshot of the Pharo Smalltalk IDE interface. It displays two code editors side-by-side. The left editor shows the code for the `BoardSquare` class, specifically the `setPosition:board:` method. The right editor shows the code for the `SnakeSquare` class, specifically the `setBack:` method. Both editors include a history navigator at the top right and a navigation bar at the bottom right.

```
BoardSquare>>#setPosition:board:  
Scope: Global  
Type: Pkg1|^Pkg2|Pk.*Co  
-- all --  
initialize-release  
setPosition:board:  
playing  
printing  
testing  
  
setPosition: aNumber board: aBoard  
position := aNumber.  
board := aBoard  
1/3 [1]  
  
SnakeSquare>>#setBack:  
Scope: Global  
Type: Pkg1|^Pkg2|Pk.*Co  
-- all --  
initialize-release  
setBack:  
destination  
printOn:  
playing  
printing  
  
setBack: aNumber  
back := aNumber.  
1/2 [1]
```



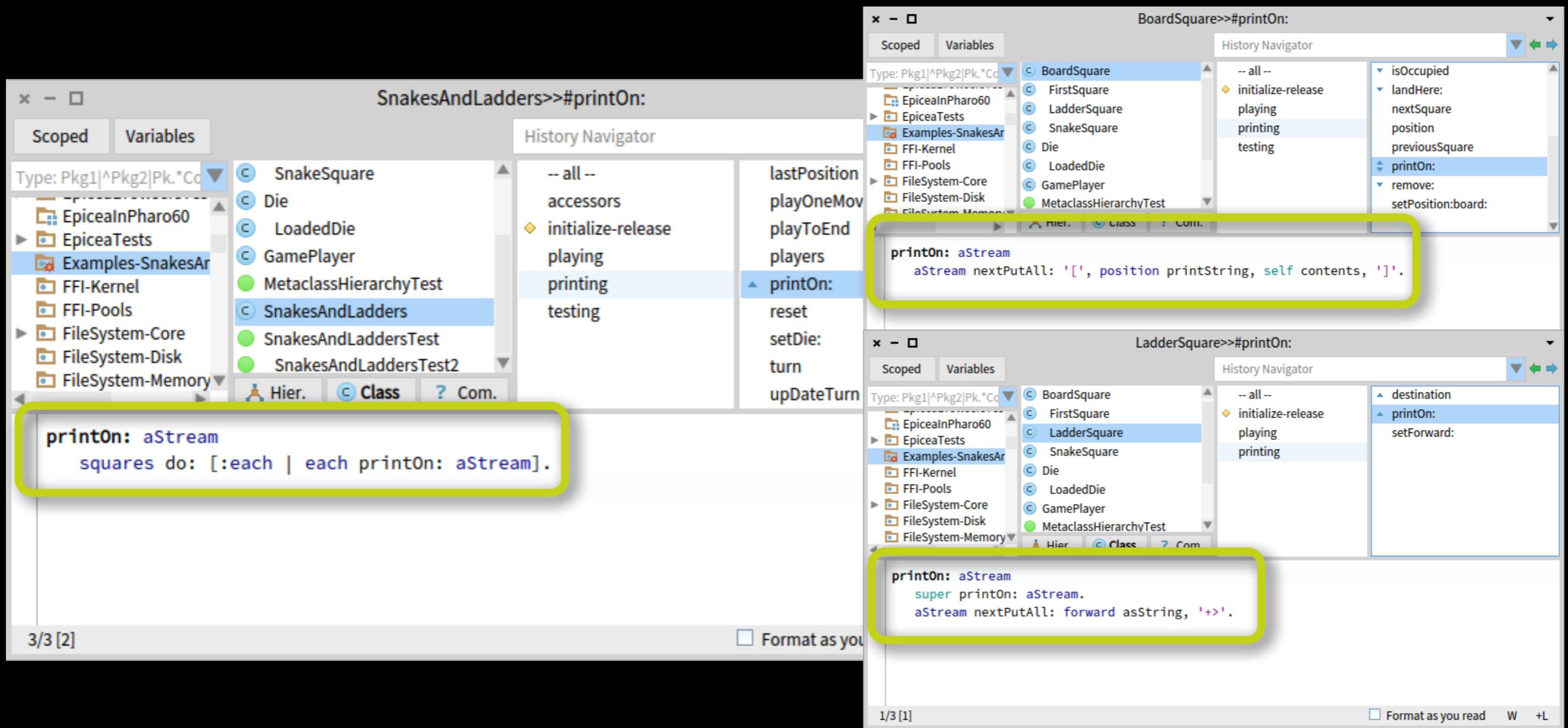
Com podem programar el mètode d'escriptura per defecte?



Vull coneixer l'estat del meu sistema, en aquest cas el joc que fem servir com a exemple. Puc *inspeccionar-lo*, però la representació textual per defecte de l'estat (**a SnakesAndLadders**) no és gaire informativa...

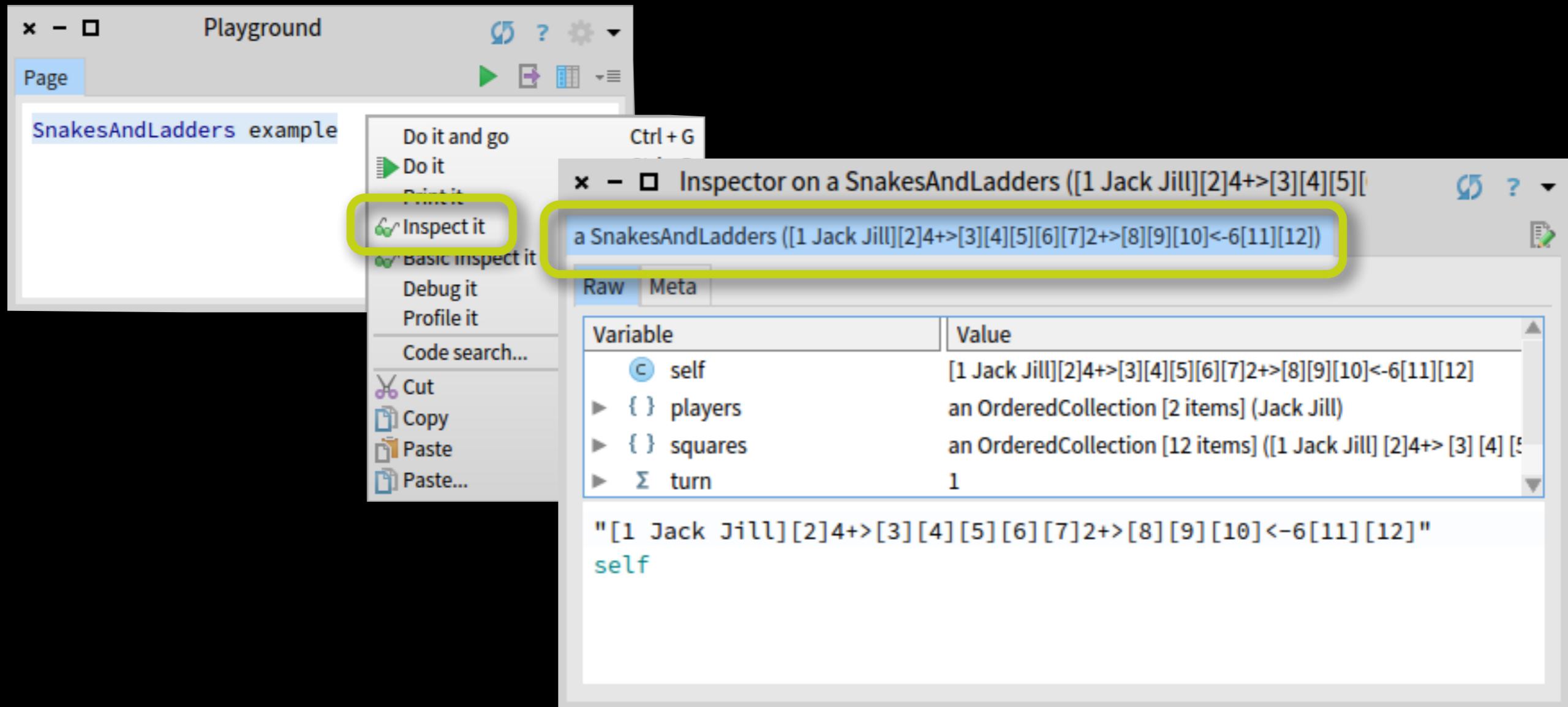
Visualitzant l'estat

Cal sobreescrivir el mètode **#printOn:**, que és el que fan servir totes les eines de l'entorn per escriure la representació textual.



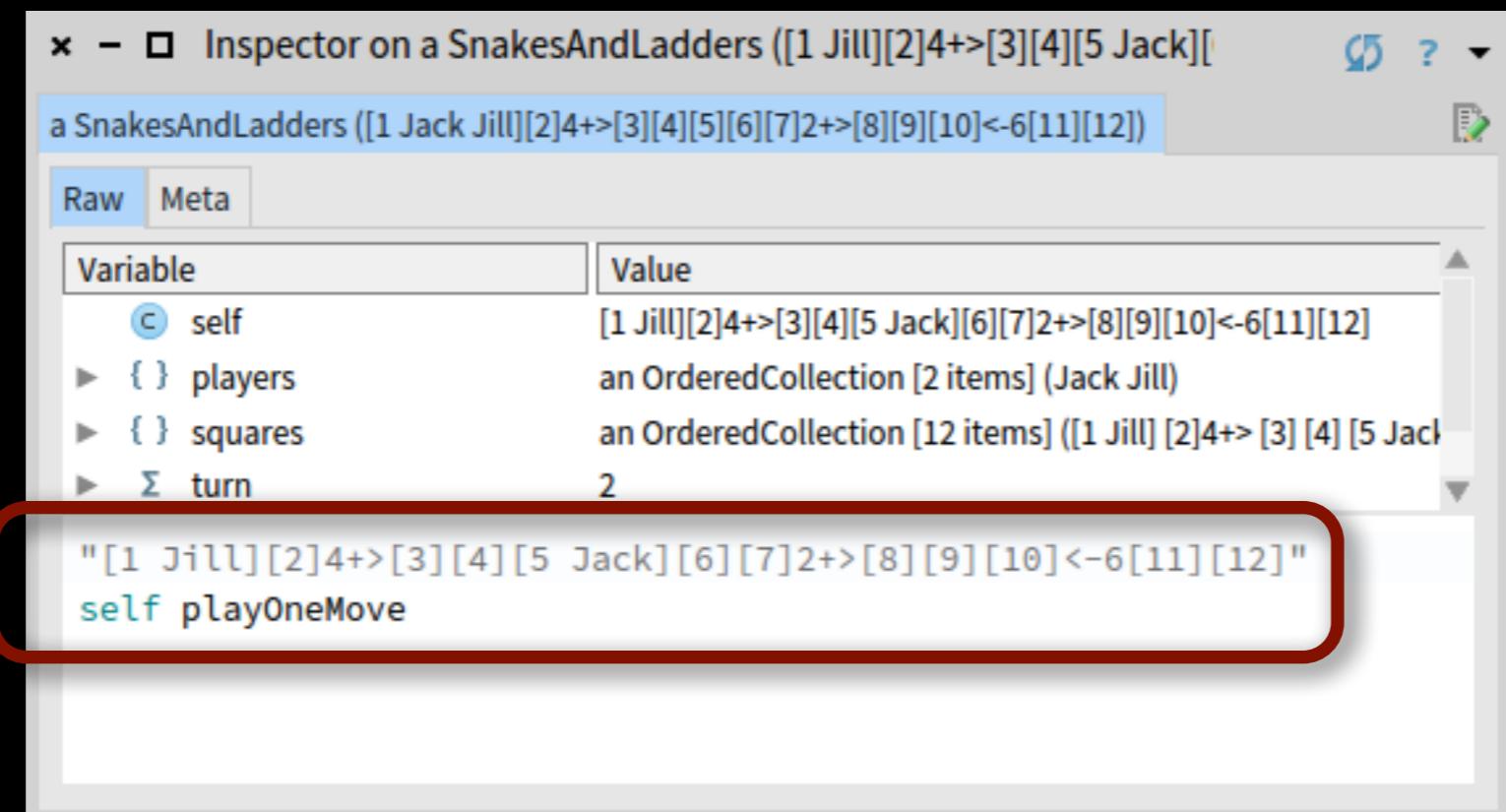
Visualitzant l'estat

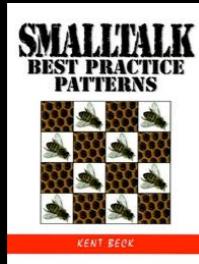
Cal sobreescrivir el mètode `#printOn:`, que és el que fan servir totes les eines de l'entorn per escriure la representació textual.



Visualitzant l'estat

Fins i tot podeu utilitzar l'inspector com a interficie pel joc...





Com podem invocar comportament a les superclasses?

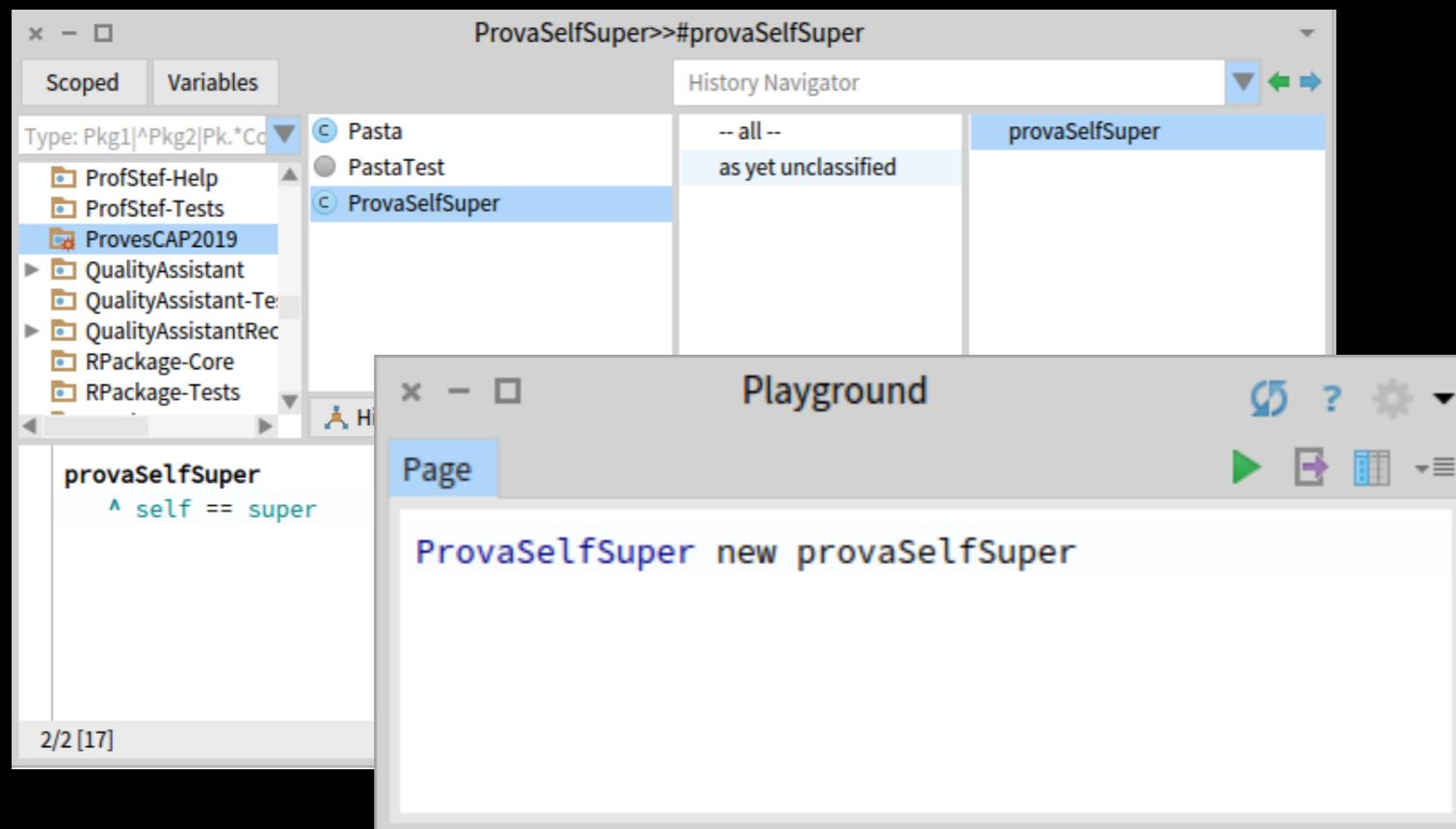
*Invoqueu codi de la superclasse explícitament enviant un missatge a **super** (i no a **self**)*

Com funciona? **super** conté una referència a l'objecte receptor del missatge corresponent al mètode que s'està executant (igual que **self!**), però...

*la cerca del mètode corresponent al missatge enviat a **super** canvia*

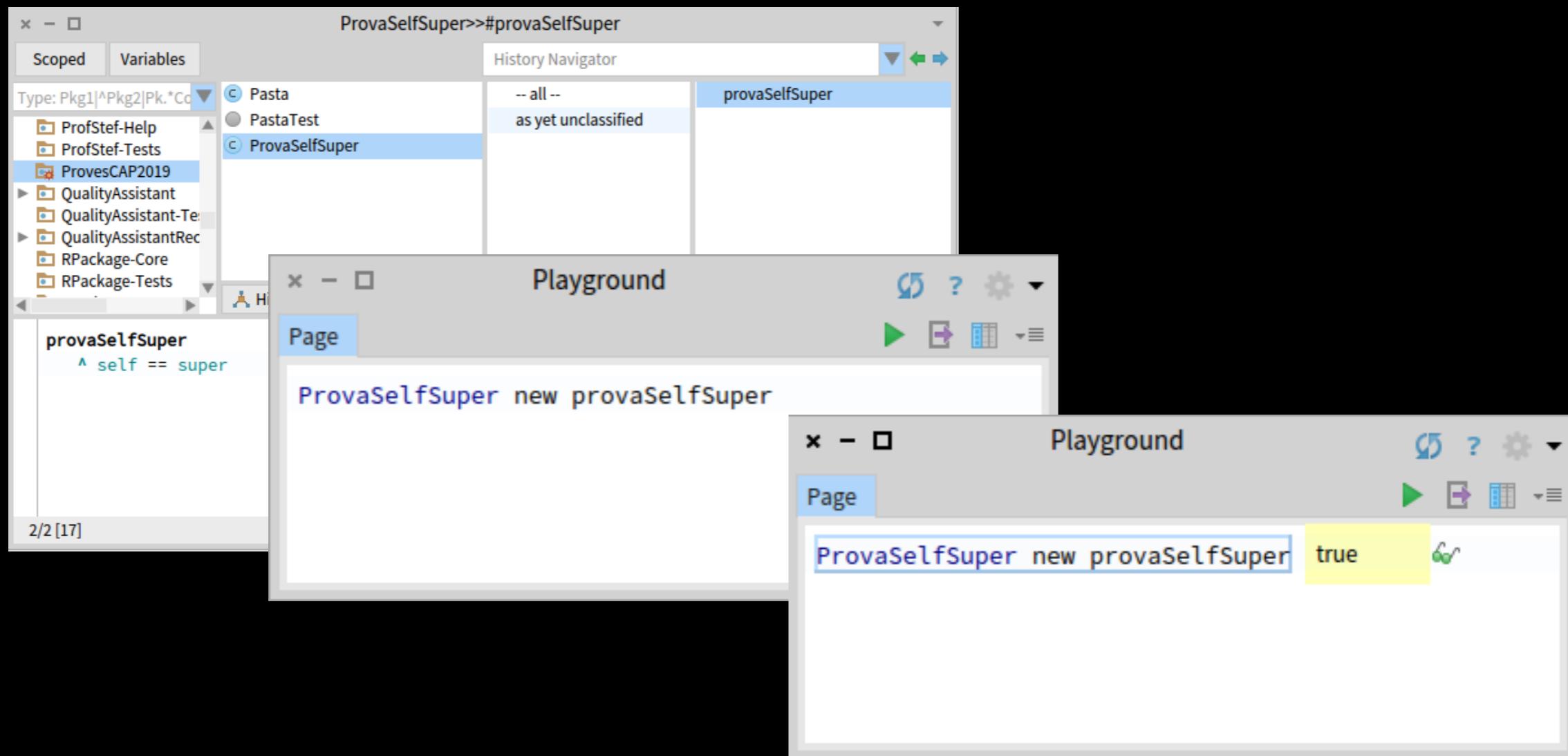
Comença a buscar aquest mètode a la superclasse de la classe que implementa el mètode que s'està executant

“**super** conté una referència a l’objecte receptor del missatge correspondent al mètode que s’està executant (igual que **self!**)”

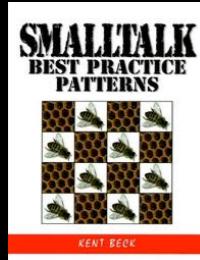


Cal utilitzar **super** amb cura. Canvieu **super** per **self** si això no modifica l’execució del codi...

“**super** conté una referència a l'objecte receptor del missatge correspondent al mètode que s'està executant (igual que **self!**)”



Cal utilitzar **super** amb cura. Canvieu **super** per **self** si això no modifica l'execució del codi...



Com podem afegir funcionalitat a la implementació d'un mètode heredat?

Sobreescriviu el mètode i envieu un missatge a super en el mètode que sobreescrivia

A screenshot of two Smalltalk environments side-by-side. The left environment shows the implementation of the #printOn: method for LadderSquare. It includes code to call super and then print a ladder symbol. The right environment shows the implementation for SnakeSquare, which overrides the #printOn: method by sending a message to super to print a snake symbol and then printing a backslash symbol itself.

LadderSquare>>#printOn:

```
printOn: aStream
super printOn: aStream.
aStream nextPutAll: forward asString, '+>'.
```

History Navigator

- all --
- initialize-release
- playing
- printing

Variables

Type: Pkg1|^Pkg2|Pk.*Core\$

- BoardSquare
- FirstSquare
- LadderSquare
- SnakeSquare
- Die
- LoadedDie
- GamePlayer
- MetaclassHierarchyTest

Scoped

Hier. Class Com.

1/3 [1]

SnakeSquare>>#printOn:

```
printOn: aStream
aStream nextPutAll: '<-', back asString.
super printOn: aStream.
```

History Navigator

- all --
- initialize-release
- playing
- printing

Variables

Type: Pkg1|^Pkg2|Pk.*Core\$

- FirstSquare
- LadderSquare
- SnakeSquare
- Die
- LoadedDie
- GamePlayer
- MetaclassHierarchyTest
- SnakesAndLadders

Scoped

Hier. Class Com.

destination

printOn:

setBack:

1/3 [1]

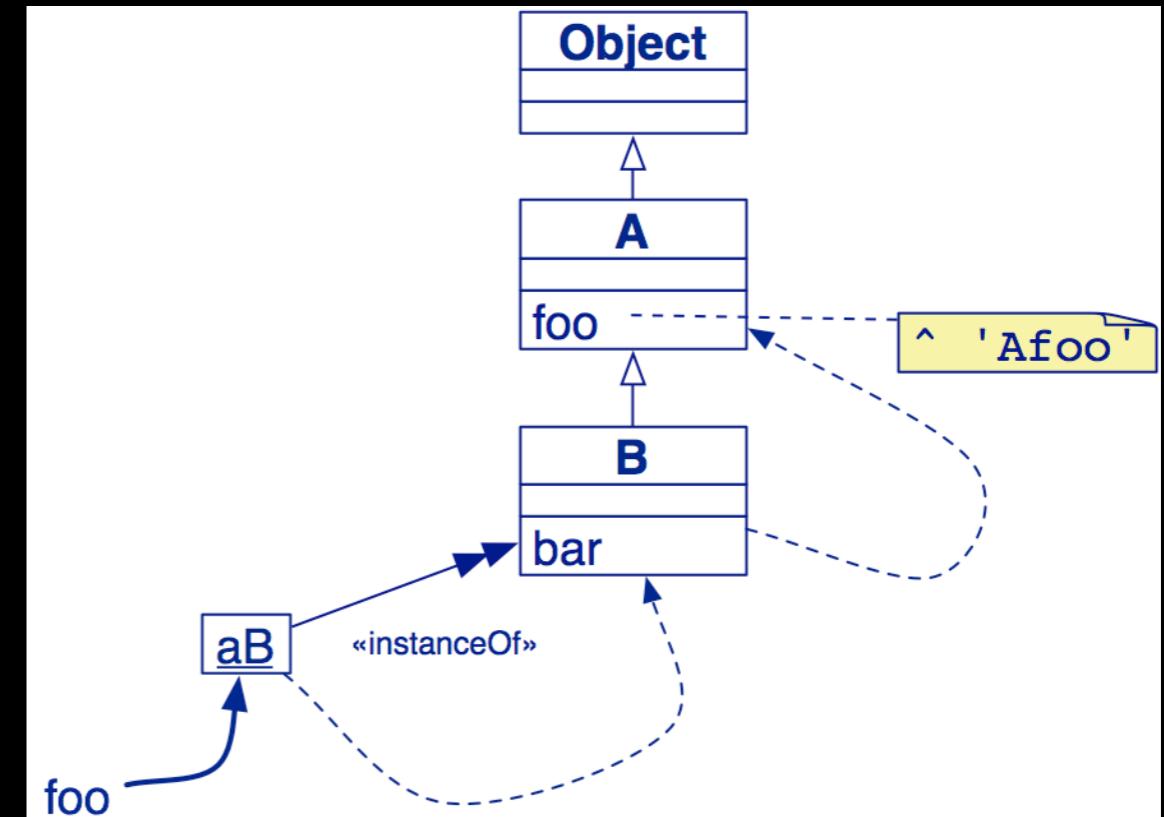
Format as you read W +L

Cerca normal de mètodes

La cerca comença a la classe de l'objecte receptor del missatge:

1.- Si el mètode és al diccionari de mètodes, s'utilitza.

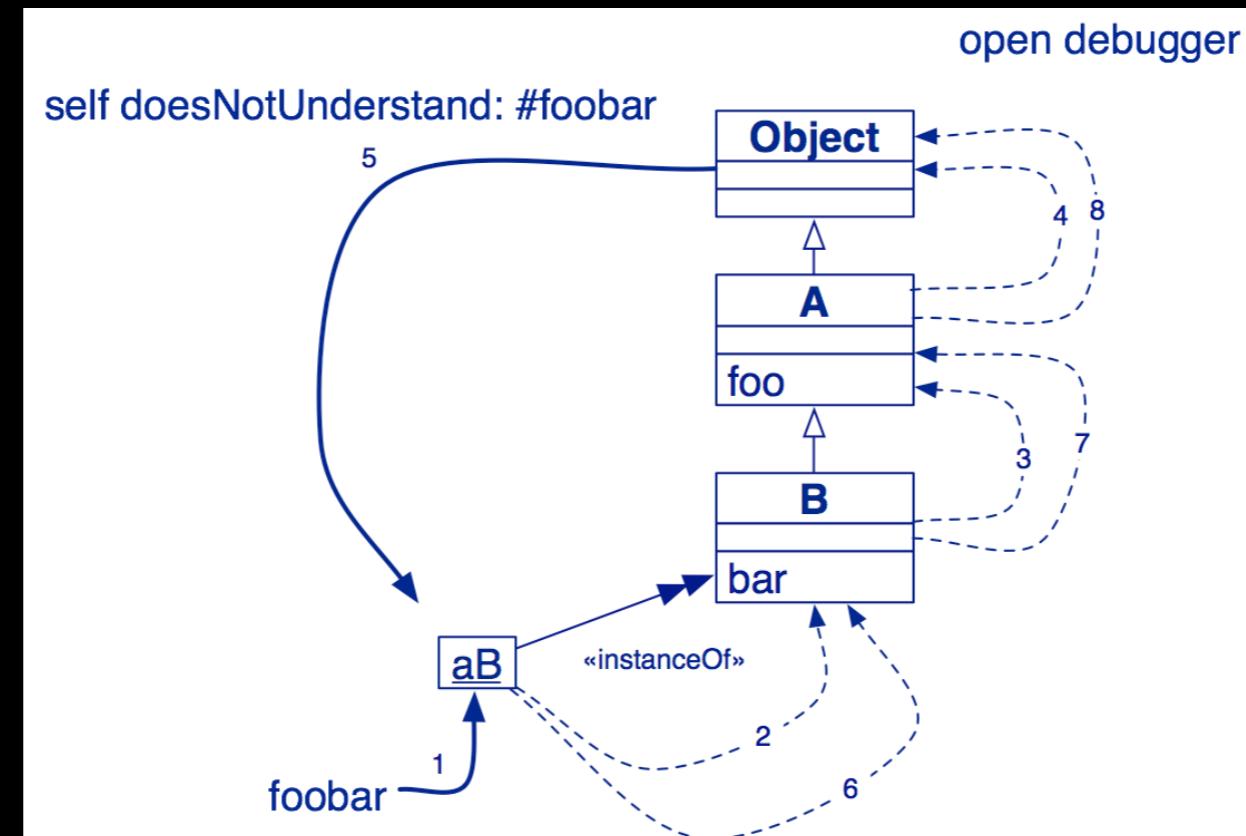
2.- Si no hi és, la cerca continua a la superclasse.



Si no es troba cap mètode això provoca un error.

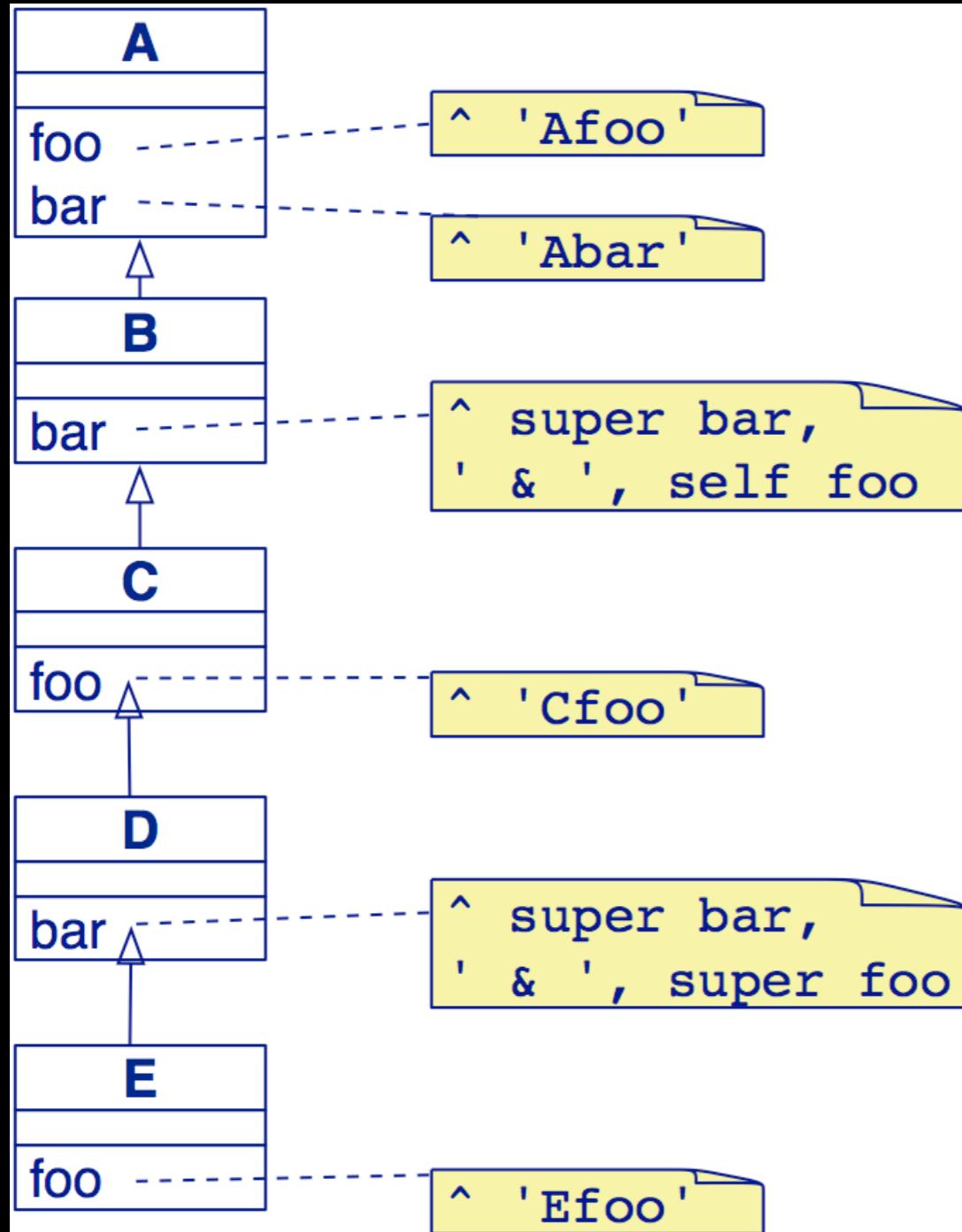
Cerca normal de mètodes

Quan la cerca falla, s'envia un missatge d'error **#doesNotUnderstand**: a l'objecte i la cerca torna a començar amb aquest missatge...



La implementació per defecte de **#doesNotUnderstand:** pot ser sobreescrita per qualsevol classe

CAP: Introducció a l'Smalltalk



A new bar → 'Abar'

B new bar → 'Abar & Afoo'

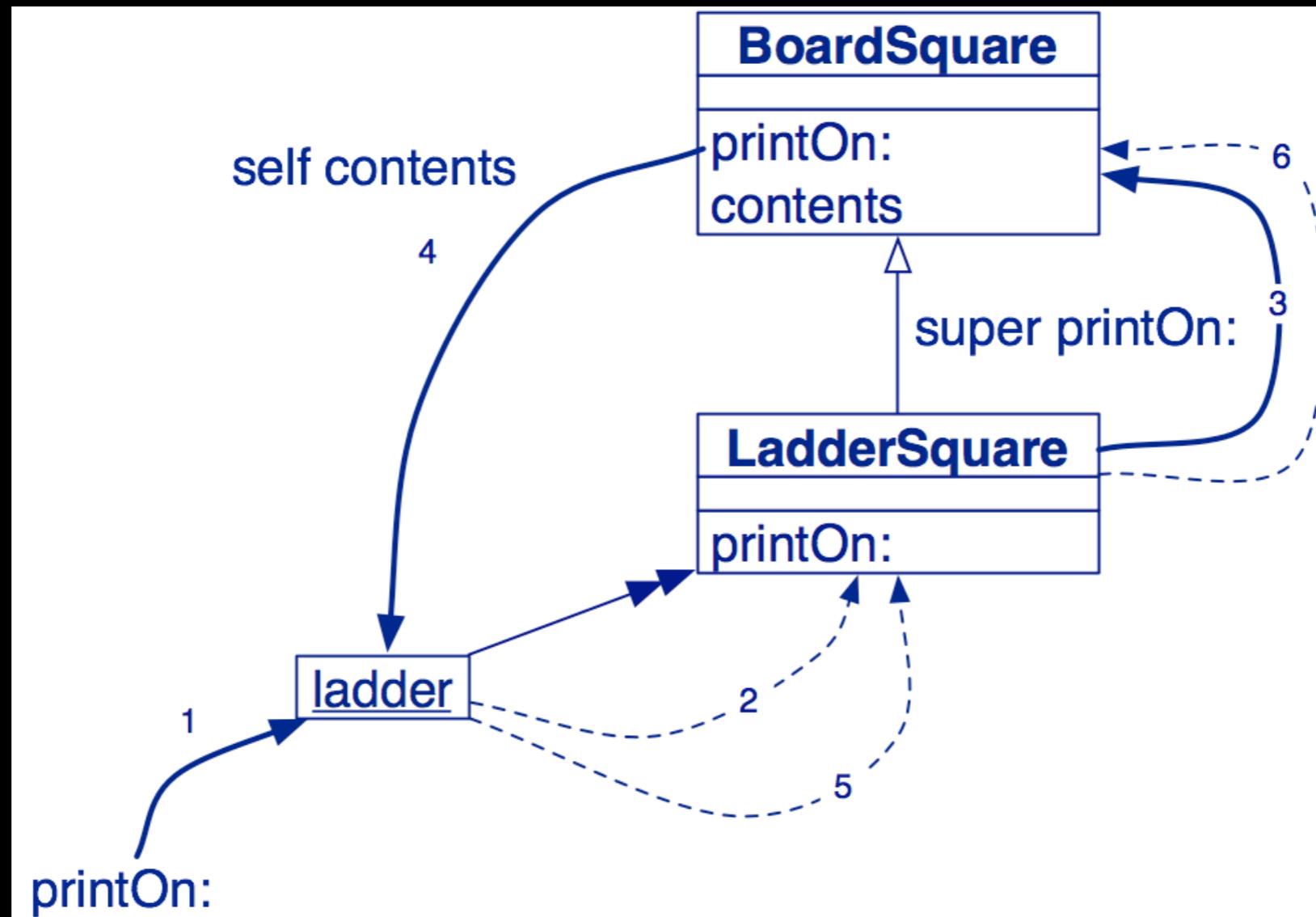
C new bar → 'Abar & Cfoo'

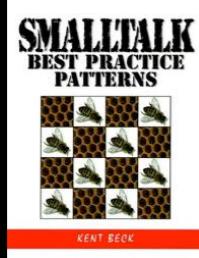
D new bar → 'Abar & Cfoo & Cfoo'

E new bar → 'Abar & Efoo & Cfoo'

CAP: Introducció a l'Smalltalk

Els enviaments a **self** sempre es resolen *dinàmicament*.
Els enviaments a **super** es resolen *estàticament*.





Com podem dividir el nostre programa en mètodes?

Dividiu el vostre programa en mètodes que realitzen una tasca identifiable

Manteniu totes les operacions dins d'un mètode al **mateix nivell d'abstracció**.

Això farà que el vostre programa estigui compost de **molts mètodes petits, cada un d'ells de poques línies de codi**.

La majoria dels mètodes seran petits i *auto-documentats*

Algunes excepcions: algorismes complexos, scripts de configuració, tests, etc.

The screenshot shows a Smalltalk development environment with the following interface elements:

- Title Bar:** SnakesAndLadders>>#playOneMove
- Toolbar:** Includes "Scoped" and "Variables" tabs.
- History Navigator:** A panel on the right containing a tree view of method names:
 - all --
 - accessors
 - initialize-release
 - playing (selected)
 - printing
 - testing
- Code Editor:** A large panel at the bottom containing the source code for the `playOneMove` method.

```
playOneMove
| result |
self assert: self invariant.
^ self isOver
  ifTrue: ['The game is over!']
  ifFalse:
    [result := (self currentPlayer moveWith: die) , self checkResult.
    self updateTurn.
    result]
```
- Status Bar:** At the bottom, it shows "1/9 [1]" on the left and "Format as you read W +L" on the right.

Invariants

Podem fer servir **#assert:** per comprovar que l'estat dins d'un mètode, abans o després d'alguna acció, sigui el que s'espera.

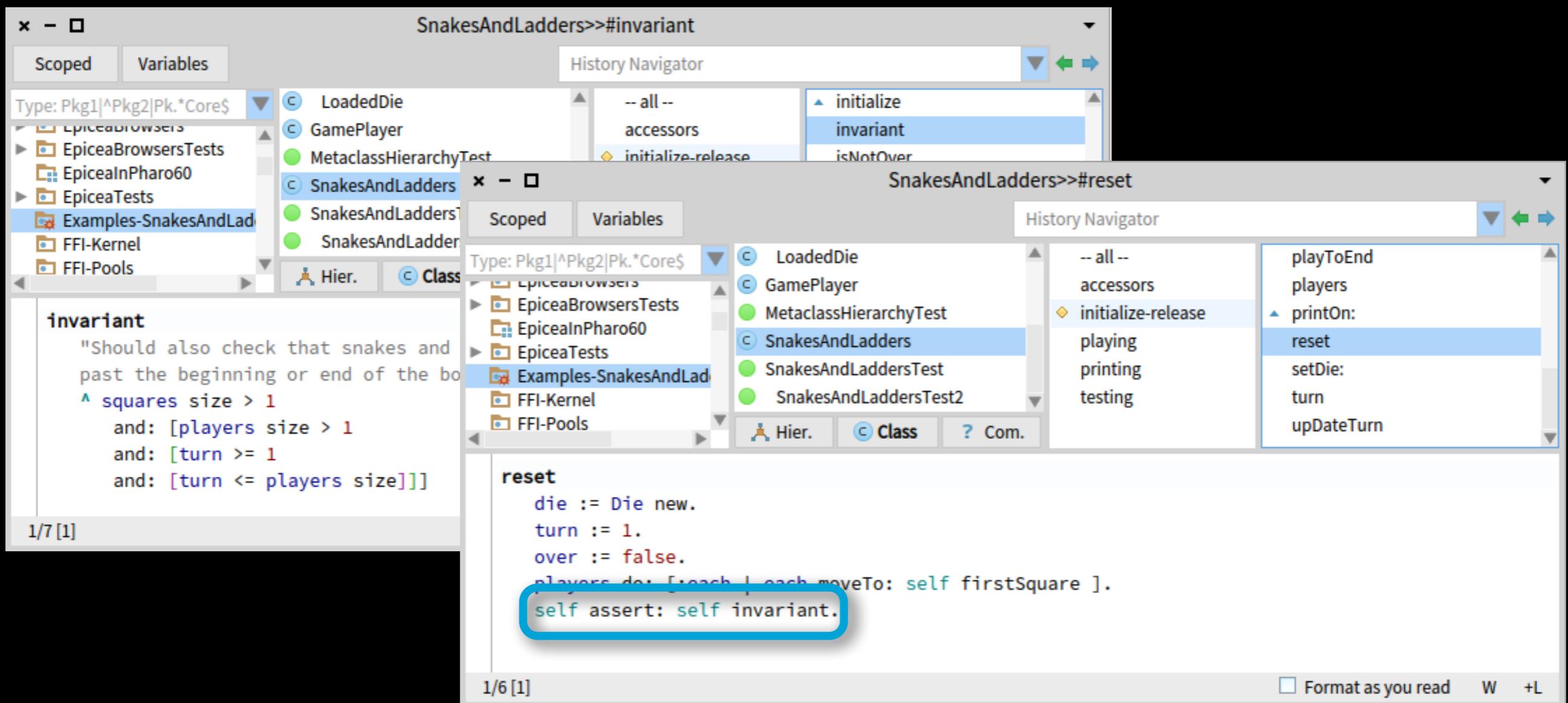
The screenshot shows the Pharo Smalltalk IDE interface. The top title bar reads "LoadedDie>>#roll:". Below it is a "History Navigator" pane with three tabs: "all", "playing", and "roll". The "playing" tab is selected, showing the history of the current session. The left side features a "Variables" pane with tabs "Scoped" and "Variables", and a "Type" dropdown set to "Pkg1|^\Pkg2|Pk.*Core\$". The main workspace shows the code for the "#roll:" method of the "LoadedDie" class. The code is as follows:

```
roll: aNumber
    self assert: ((1 to: 6) includes: aNumber).
    roll := aNumber.
```

The line "self assert: ((1 to: 6) includes: aNumber)." is highlighted with a blue oval. At the bottom of the workspace, there are navigation buttons: "1/3 [1]" on the left and "Format as you read" with a "W" icon and a "+L" icon on the right.

Invariants

També podem establir un *invariant* de classe per comprovar-lo abans o després d'alterar l'estat de l'objecte.



SnakesAndLadders>>#invariant

```
invariant
    "Should also check that snakes and
     past the beginning or end of the board"
    ^ squares size > 1
        and: [players size > 1]
        and: [turn >= 1]
        and: [turn <= players size]]
```

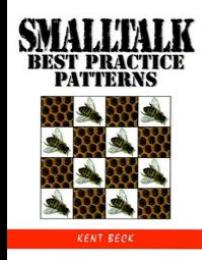
1/7 [1]

SnakesAndLadders>>#reset

```
reset
    die := Die new.
    turn := 1.
    over := false.
    players do: [:each | each moveTo: self firstSquare].
    self assert: self invariant.
```

1/6 [1]

Format as you read W +L



Com comentem els mètodes?

Comuniqueu informació important que no sigui obvia a partir del codi. Poseu els comentaris al començament del mètode.

Consell:

- Els comentaris poden fer pudor de codi disfressat!
- Proveu de refactoritzar el codi i canviar els noms dels mètodes per poder prescindir dels comentaris.

Debugging: L'inspector

Podeu inspeccionar tot allò que vulgueu: Qualsevol expressió, la representació textual, interaccionar amb qualsevol objecte, inspeccionar variables d'instància, navegar pel sistema...

The screenshot shows the Smalltalk Inspector interface with the title "Inspector on a SnakesAndLadders ([1 Jack Jill][2]4+>[3][4][5][6][7]2+>[8][9][10]<-6[11][12])". The left pane displays the instance variables of the selected object:

Variable	Value
self	[1 Jack Jill][2]4+>[3][4][5][6][7]2+>[8][9][10]
players	an OrderedCollection [2 items] (Jack Jill)
squares	an OrderedCollection [12 items] ([1 Jack J...]
turn	1
die	a Die

The right pane shows the contents of the "squares" variable, which is an OrderedCollection of 12 items:

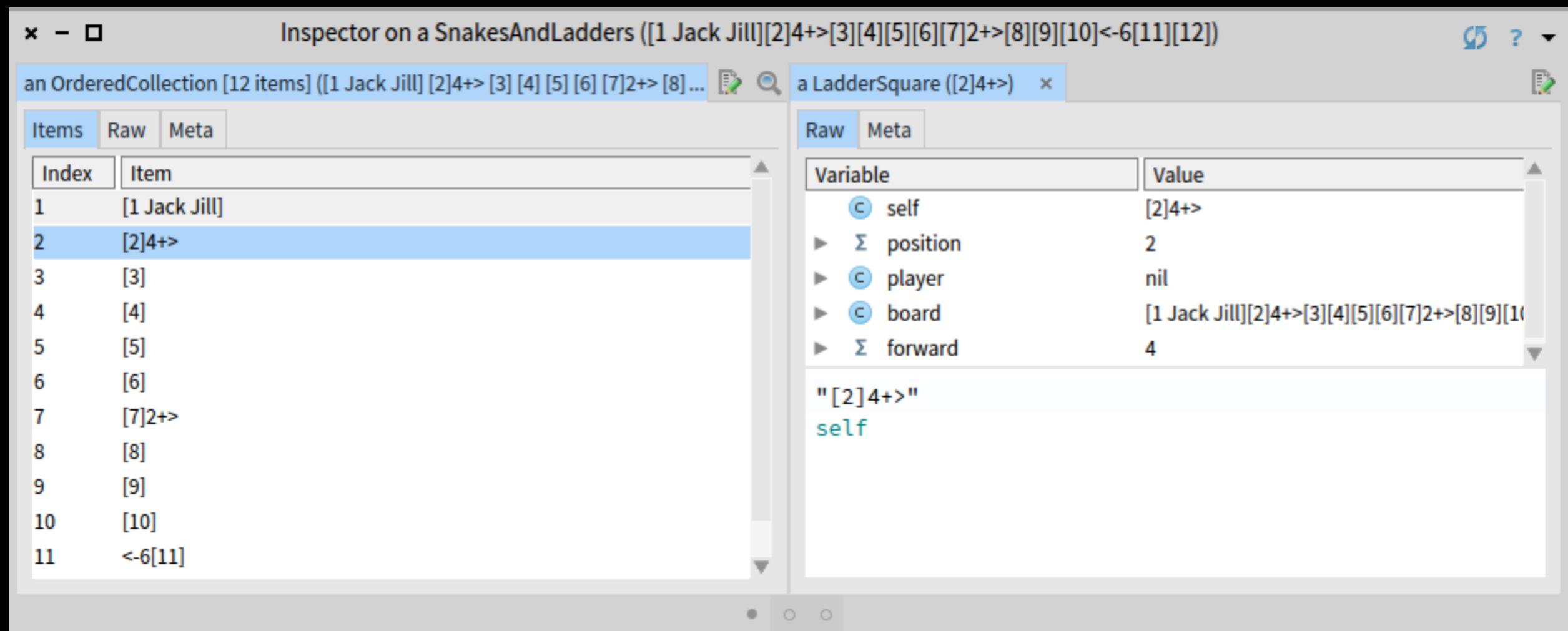
Index	Item
1	[1 Jack Jill]
2	[2]4+>
3	[3]
4	[4]
5	[5]
6	[6]
7	[7]2+>
8	[8]
9	[9]
10	[10]
11	<-6[11]

At the bottom, there is some Smalltalk code:

```
[1 Jack  
Jill][2]4+>[3][4][5][6][7]2+>[8][9][10]<-6[11][12]"  
self playOneMove
```

Debugging: L'inspector

Podeu inspeccionar tot allò que vulgueu: Qualsevol expressió, la representació textual, interaccionar amb qualsevol objecte, inspeccionar variables d'instància, navegar pel sistema...



Debugging: Breakpoints

Podeu enviar el missatge **self halt** per iniciar el debugger en qualsevol moment

The screenshot shows the Pharo Smalltalk debugger interface. The top title bar reads "SnakesAndLadders>>#playOneMove". The left pane displays a class browser with the current class set to "SnakesAndLadders". The code editor shows the following method:

```
playOneMove
    | result |
    self assert: self invariant.
    self halt. self isOver
    ^ self isOver
        ifTrue: ['The game is over!']
        ifFalse:
            [result := (self currentPlayer ...)]
```

A red box highlights the line "self halt.". The bottom-left status bar shows "5/10 [16]". To the right, the "History Navigator" pane shows a list of recent method executions, and the "Halt" button is highlighted in blue.

Debugging

Amb el debugger puc executar els enviaments de missatges en un sol pas (**Over**) o anar a veure els detalls de l'execució del mètode corresponent (**Into**), i moltes altres coses també... *EXPLOREU!*

The screenshot shows the Smalltalk debugger interface with the following components:

- Stack:** Shows the call stack with frames:
 - SnakesAndLadders playOneMove
 - SnakesAndLadders Dolt
 - OpalCompiler evaluate
 - RubSmalltalkEditor evaluate:andDo:
 - RubSmalltalkEditor highlightEvaluateAndDo:
- Source:** Displays the source code for the `playOneMove` method:

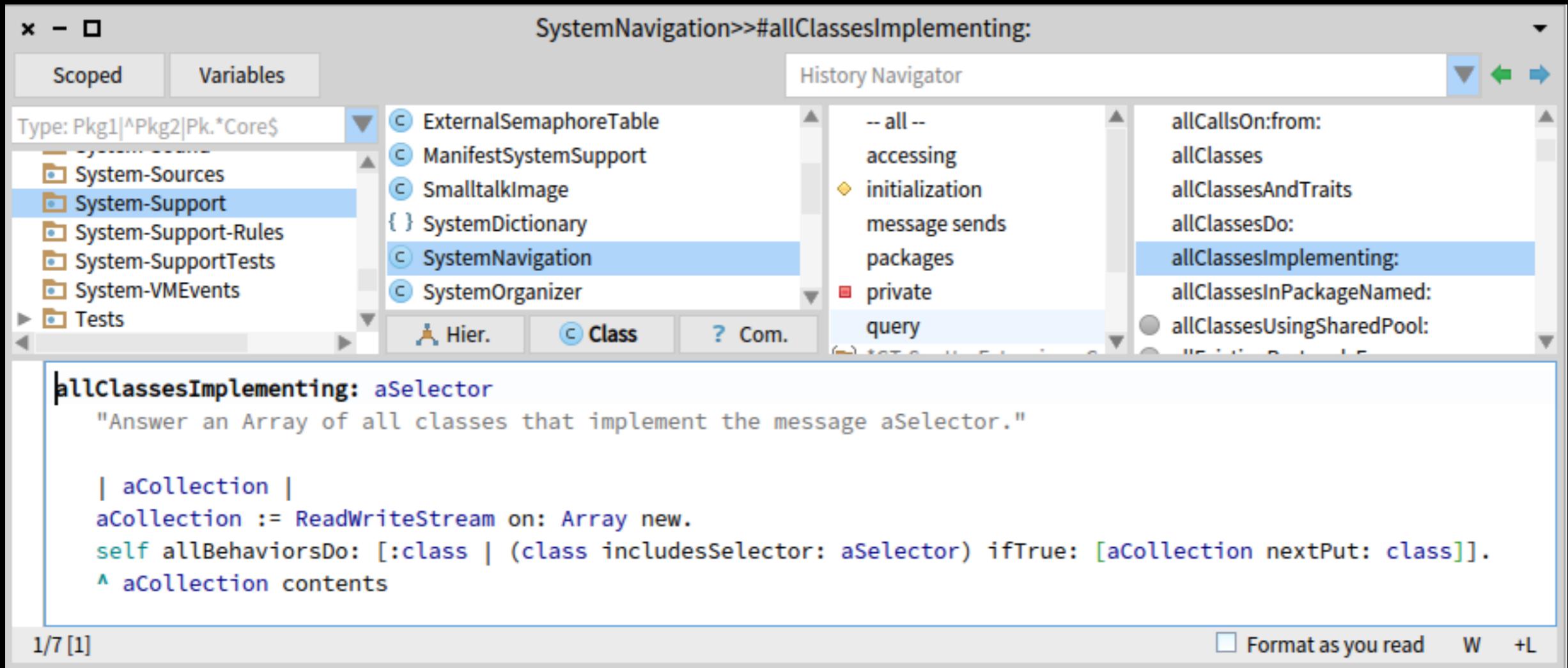
```
playOneMove
| result |
ifTrue: self invariant.
self halt.
ifTrue: ['The game is over!']
iffalse:
[result := (self currentPlayer moveWith: die) , self checkResult.
self updateTurn.
result]
```

The line `self halt.` is highlighted with a red rectangle.
- Variables:** Shows the current variables:

Type	Variable	Value
implicit	self	[1 Jack Jill][2]4+>[3][4][5][6][7]2+>[8][9][10]<-6[11][12]
attribute	die	a Die
attribute	over	false
attribute	players	an OrderedCollection [2 items] (Jack Jill)

Debugging

Apreneu a jugar amb el sistema. Exploreu els mètodes de la classe **SystemNavigation** i trobeu informació sobre el sistema enviant missatges a **SystemNavigation default**.



The screenshot shows the Smalltalk IDE interface with the following details:

- Top Bar:** Shows the title "SystemNavigation>>#allClassesImplementing:".
- Left Sidebar (Type Browser):** Shows categories like "System-Sources", "System-Support", etc., with "System-Support" currently selected.
- Middle Area (History Navigator):** Shows a list of methods:
 - all --
 - accessing
 - initialization
 - message sends
 - packages
 - private
 - query
- Right Area (Method Browser):** Shows the method "allClassesImplementing:" highlighted in blue, along with other related methods like "allCallsOn:from:", "allClasses", etc.
- Bottom Area (Code Editor):** Displays the source code for "allClassesImplementing:":

```
allClassesImplementing: aSelector
    "Answer an Array of all classes that implement the message aSelector."
    | aCollection |
    aCollection := ReadWriteStream on: Array new.
    self allBehaviorsDo: [:class | (class includesSelector: aSelector) ifTrue: [aCollection nextPut: class]].
    ^ aCollection contents
```
- Bottom Right:** Includes buttons for "Format as you read", "W", and "+L".

Debugging

Apreneu a jugar amb el sistema. Exploreu els mètodes de la classe **SystemNavigation** i trobeu informació sobre el sistema enviant missatges a **SystemNavigation default**.

The screenshot shows the Smalltalk playground interface with the following components:

- Top Left:** A tree view of packages: Pkg1|¹Pkg2|Pkgs.*Core\$. The **System-Support** package is selected.
- Top Center:** A history navigator window titled "SystemNavigation>>#allCallsOn:".
- Top Right:** A list of methods for **SystemNavigation**:
 - all --
 - accessing
 - initialization
 - message sends
 - packages
 - private
 - query
- Middle Left:** A code editor window titled "Playground" with the following code:

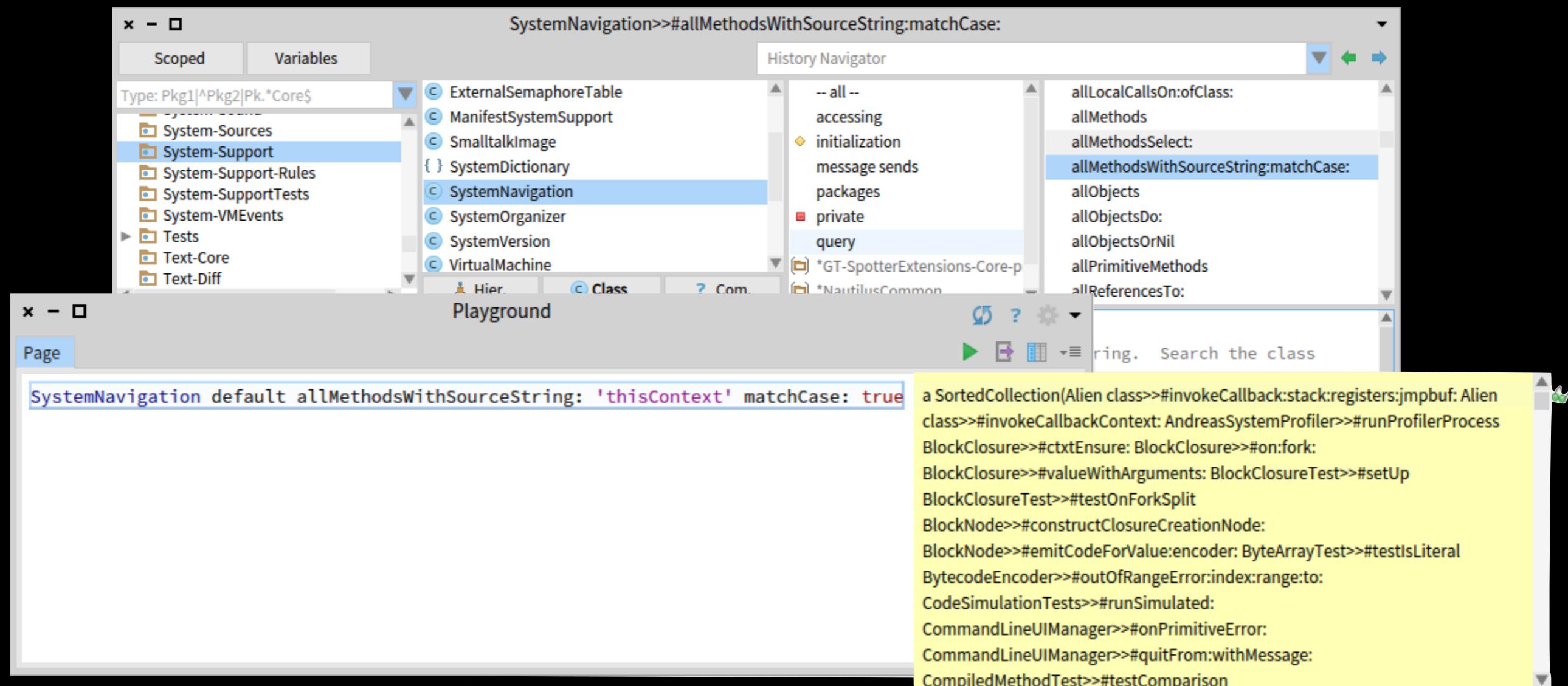
```
allCallsOn: aSymbol
    "Answer a Collection of all the messages sent to aSymbol"
    "self new allCallsOn: #allCallsOn"
    ^ self allReferencesTo: aSymbol
```
- Middle Right:** A code editor window titled "Page" with the following code:

```
SystemNavigation default allCallsOn: #currentPlayer
```
- Bottom Right:** A yellow-highlighted tooltip showing the definition of **#currentPlayer**:

```
an OrderedCollection(SnakesAndLadders>>#playOneMove
SnakesAndLadders>>#checkResult SnakesAndLaddersTest>>#testExample
SnakesAndLaddersTest>>#testStartPosition
SnakesAndLaddersTest2>>#testExample)
```

Debugging

Apreneu a jugar amb el sistema. Exploreu els mètodes de la classe **SystemNavigation** i trobeu informació sobre el sistema enviant missatges a **SystemNavigation default**.



<http://creativecommons.org/licenses/by-sa/3.0/>



Attribution-ShareAlike 3.0 Unported

You are free:

- to **Share** — to copy, distribute and transmit the work
- to **Remix** — to adapt the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work.

The best way to do this is with a link to this web page.

Any of the above conditions can be waived if you get permission from the copyright holder.
Nothing in this license impairs or restricts the author's moral rights.