

Conceptes Avançats de Programació

Reflexió en Smalltalk

I X C U I C V I O C H I O U I C U I C A U I V

Basat en el curs

Dynamic Object Oriented Programming with Smalltalk
del Prof. Oscar Nierstrasz, Universitat de Berna

https://www.iam.unibe.ch/scg/svn_repos/Lectures/Smalltalk/

Jordi Delgado (jdelgado@cs.upc.edu)

Abans de començar a tractar la reflexió en Smalltalk cal entendre com és que les classes també són objectes... i si ho són, de qui són instàncies?

Reifica el metamodel: *Un sistema reflexiu té un model del seu propi metamodel*



Pròleg:

- Les Metaclasses en 7 parts
- Classes indexades
- Variables de classe-instància
- Variables de classe

Part del material és cortesia d'Stéphane Ducasse

Pròleg:

- *Les Metaclasses en 7 parts*
- Classes indexades
- Variables de classe-instància
- Variables de classe

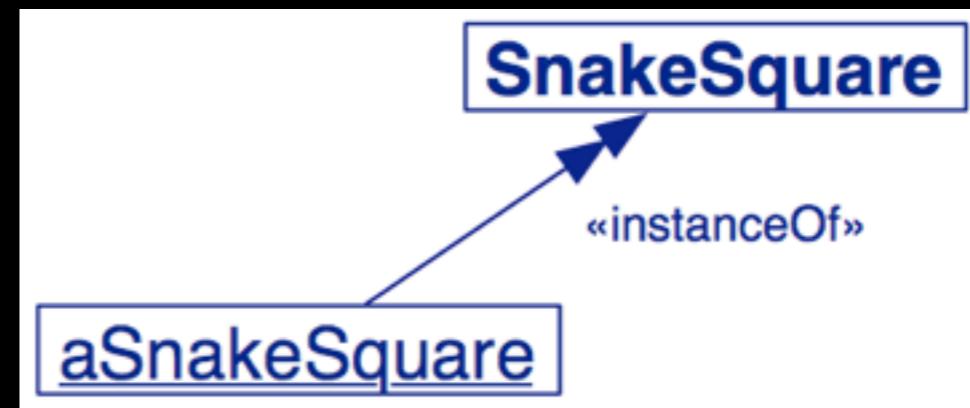
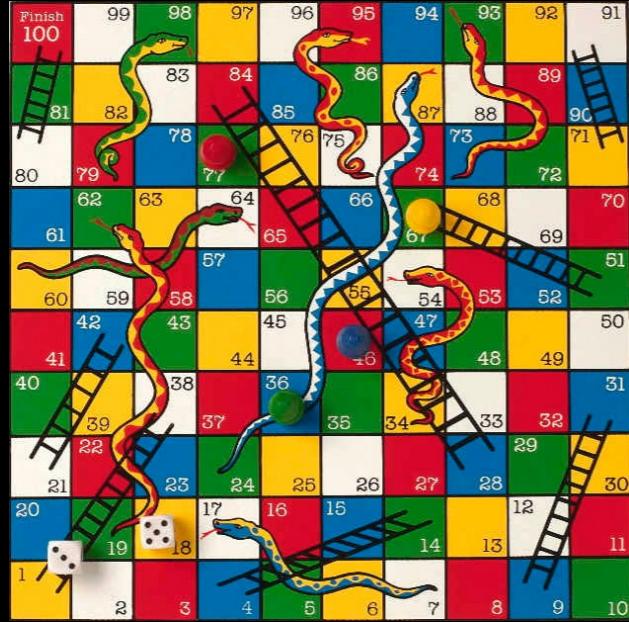
Les metaclasses en 7 parts:

- 1- Tot objecte és instància d'una classe
- 2- Tota classe hereta eventualment d'**Object**
- 3- Tota classe és instància d'una metaclass
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

Les metaclasses en 7 parts:

- 1- ***Tot objecte és instància d'una classe***
- 2- Tota classe hereta eventualment d'**Object**
- 3- Tota classe és instància d'una metaclass
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

CAP: Reflexió en Smalltalk



Recordeu el joc dels *Snakes and Ladders*...

Les metaclasses en 7 parts:

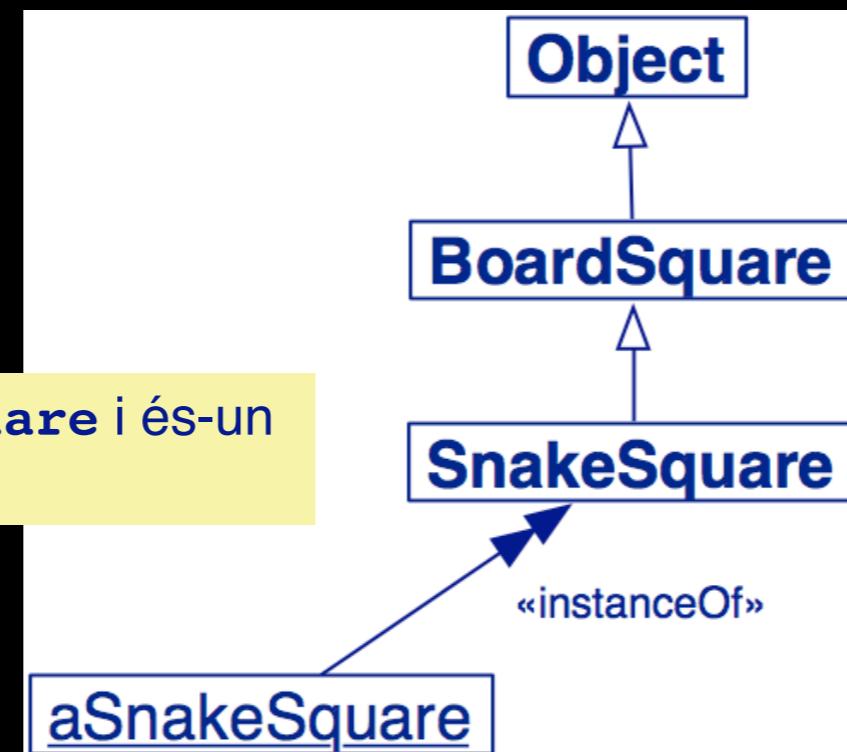
- 1- Tot objecte és instància d'una classe
- 2- ***Tota classe hereta eventualment d'Object***
- 3- Tota classe és instància d'una metaclass
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

Tot objecte és-un (*is-an*) Object

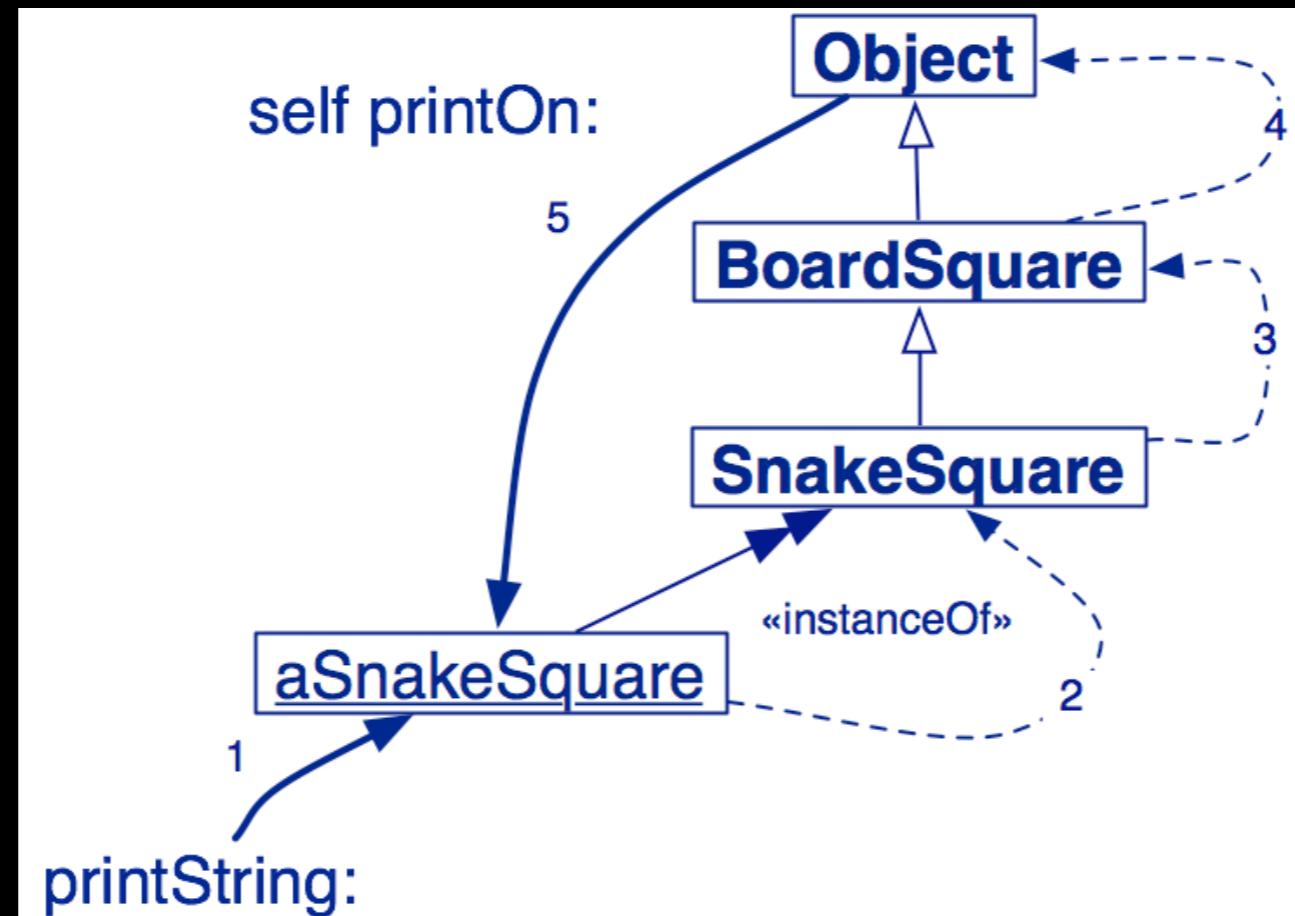
La classe de cada objecte
hereta d'Object



aSnakeSquare és-un SnakeSquare i és-un BoardSquare i és-un Object

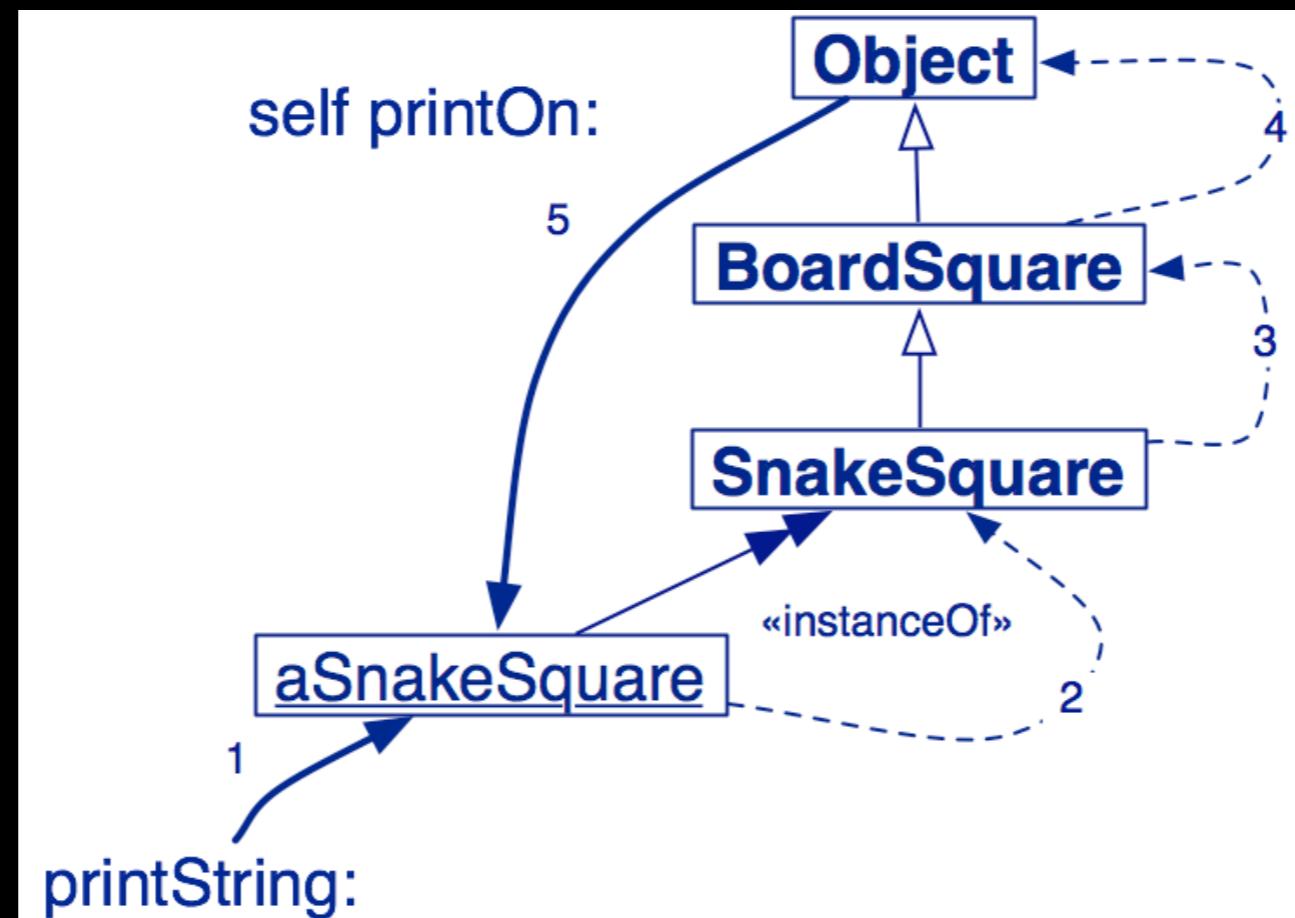


Quan un objecte rep un missatge, el mètode es busca al diccionari de mètodes de la seva classe, i, si cal, a les seves superclasses, fins arribar a **Object**.



Responsabilitats d'object:

- representa el comportament comú a tots els objectes (gestió d'errors, etc.).
- totes les classes haurien d'heretar d'Object

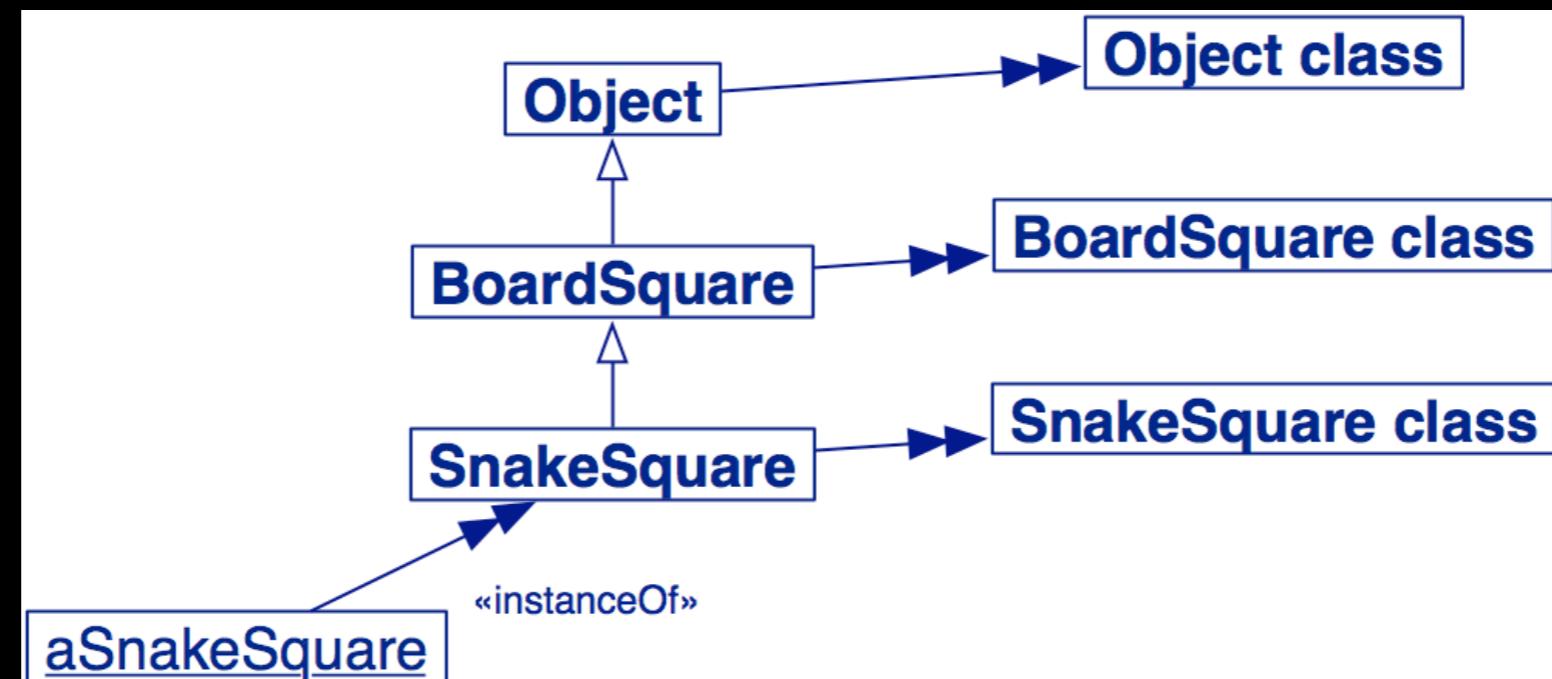


Les metaclasses en 7 parts:

- 1- Tot objecte és instància d'una classe
- 2- Tota classe hereta eventualment d'**Object**
- 3- **Tota classe és instància d'una metaclass**
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

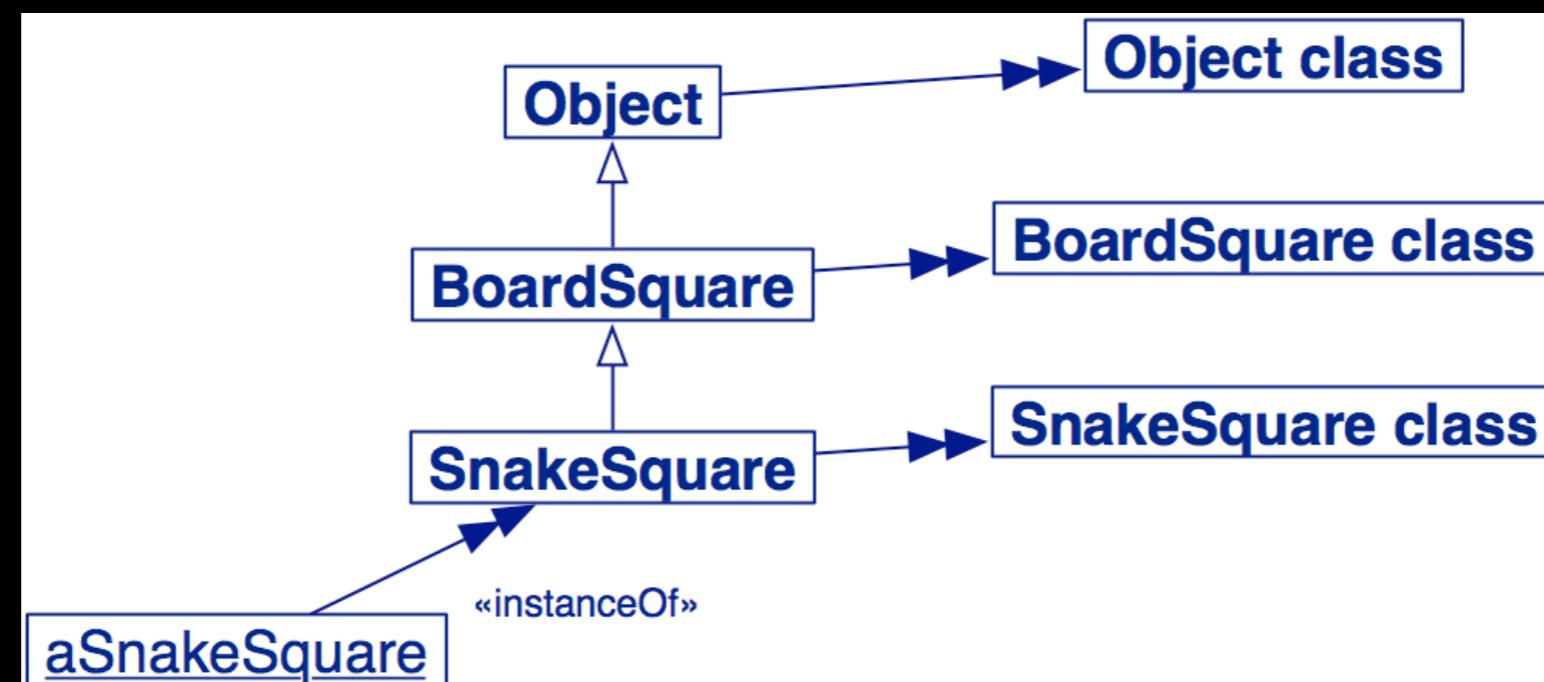
Les classes també són objectes!!

Cada classe, diguem-ne **C**, és l'única instància de la seva **metaclasses**, anomenada **C class**



No hi ha metaclasses explícites

- Les metaclases es construeixen *implicitament* quan es crean les classes
- No hi ha compartició de metaclases, cada classe és *instància única* de la seva metaclasse



Exemples:

```
BoardSquare allSubclasses →  
    an OrderedCollection(LadderSquare FirstSquare SnakeSquare)
```

```
SnakeSquare allSubclasses → an OrderedCollection()
```

```
SnakeSquare allInstances → an Array(<-6[11])
```

```
SnakeSquare instVarNames → #( 'back' )
```

```
SnakeSquare back: 5 → <-5[nil]
```

```
SnakeSquare selectors → #( #setBack: #printOn: #destination )
```

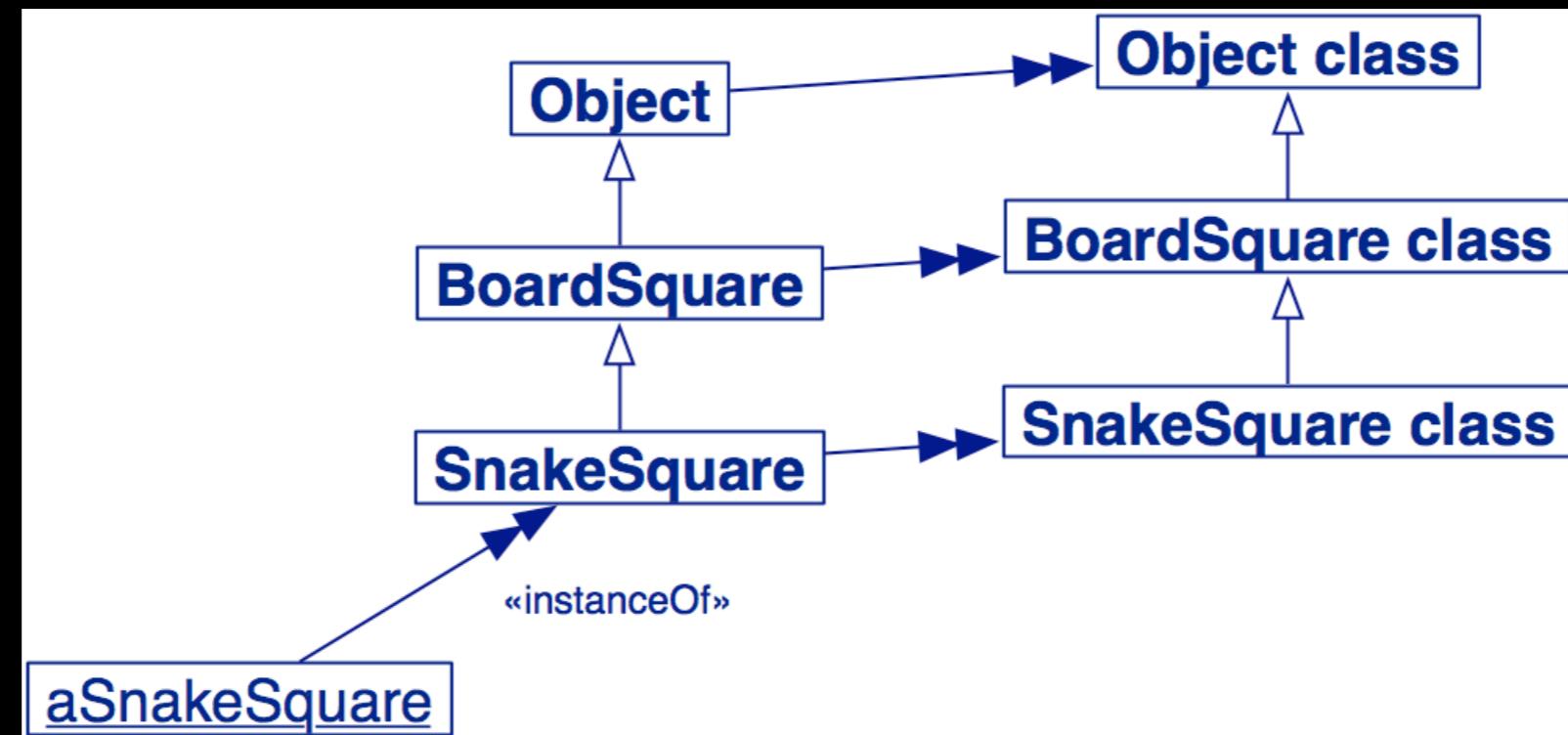
```
SnakeSquare canUnderstand: #new → false
```

```
SnakeSquare canUnderstand: #setBack: → true
```

Les metaclasses en 7 parts:

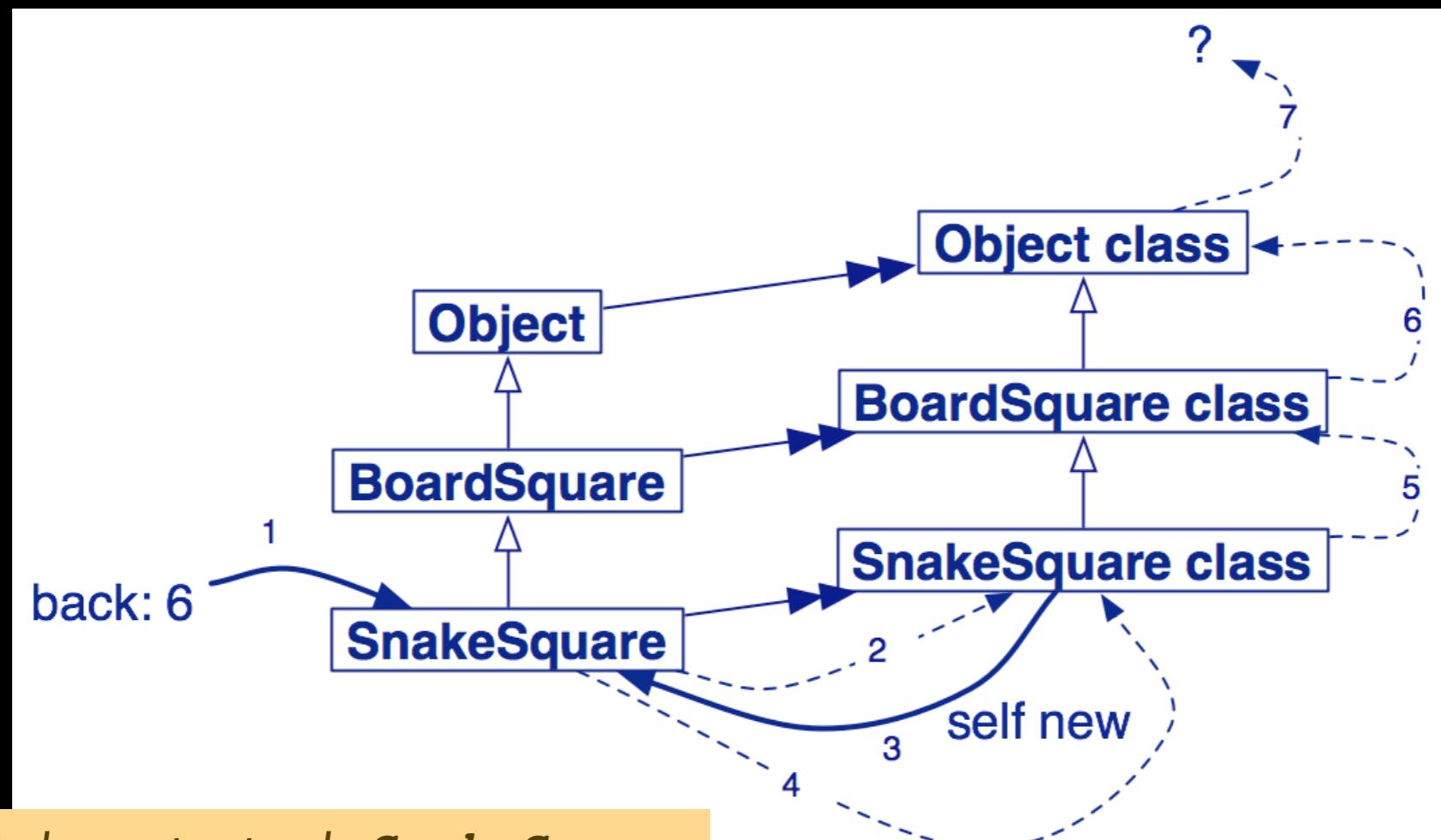
- 1- Tot objecte és instància d'una classe
- 2- Tota classe hereta eventualment d'**Object**
- 3- Tota classe és instància d'una metaclass
- 4- ***La jerarquia de metaclasses és equivalent a la jerarquia de classes***
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

La jerarquia de metaclasses és equivalent a la jerarquia de classes



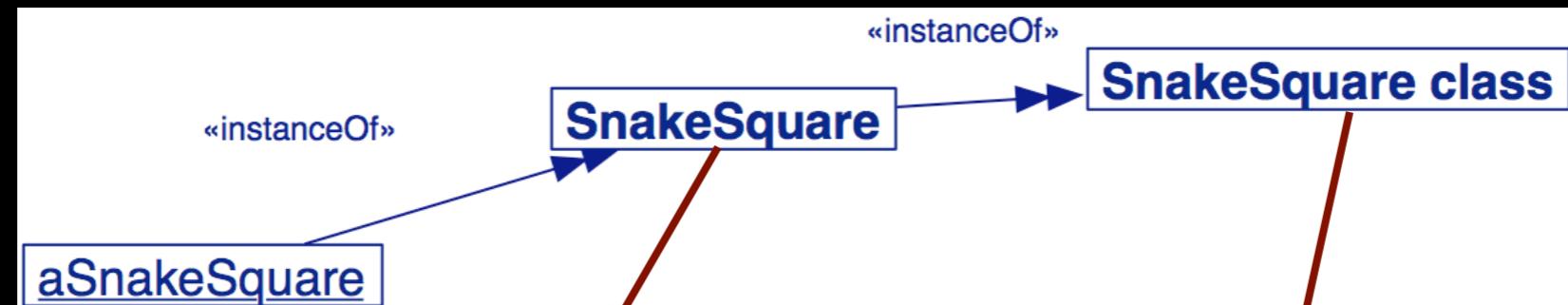
CAP: Reflexió en Smalltalk

- Les classes també són objectes, per tant tot el que hem après pels objectes és vàlid també per a les classes.
- Un enviament de missatge a una classe farà que es comenci a buscar el mètode corresponent al diccionari de mètodes de la metaclasse.



#back : és un mètode constructor de **SnakeSquare**

CAP: Reflexió en Smalltalk



The screenshot shows the Smalltalk IDE interface with two open windows illustrating reflection.

Top Window (Left):

Code:

```

x - □
Scoped Variables
Type: Pkg1|^Pkg2|Pk.*Core$ ▾
  Examples-SnakesAndLadders
  FFI-Kernel
  FFI-Pools
  FileSystem-Core
  FileSystem-Disk
  FileSystem-Memory
  FileSystem-Tests-Core
  FileSystem-Tests-Disk
  FileSystem-Tests-Memory
  SnakesAndLadders
  ...
setBack: aNumber
back := aNumber.
  
```

History Navigator:

SnakesAndLadders>>#setBack:

- all --
- initialize-release
- playing
- destination
- printOn:
- setBack:

Bottom Window (Right):

Code:

```

x - □
Scoped Variables
Type: Pkg1|^Pkg2|Pk.*Core$ ▾
  Examples-SnakesAndLadders
  FFI-Kernel
  FFI-Pools
  FileSystem-Core
  FileSystem-Disk
  FileSystem-Memory
  FileSystem-Tests-Core
  FileSystem-Tests-Disk
  FileSystem-Tests-Memory
  SnakesAndLadders
  ...
back: number
^ self new setBack: number
  
```

History Navigator:

SnakesAndLadders class>>#back:

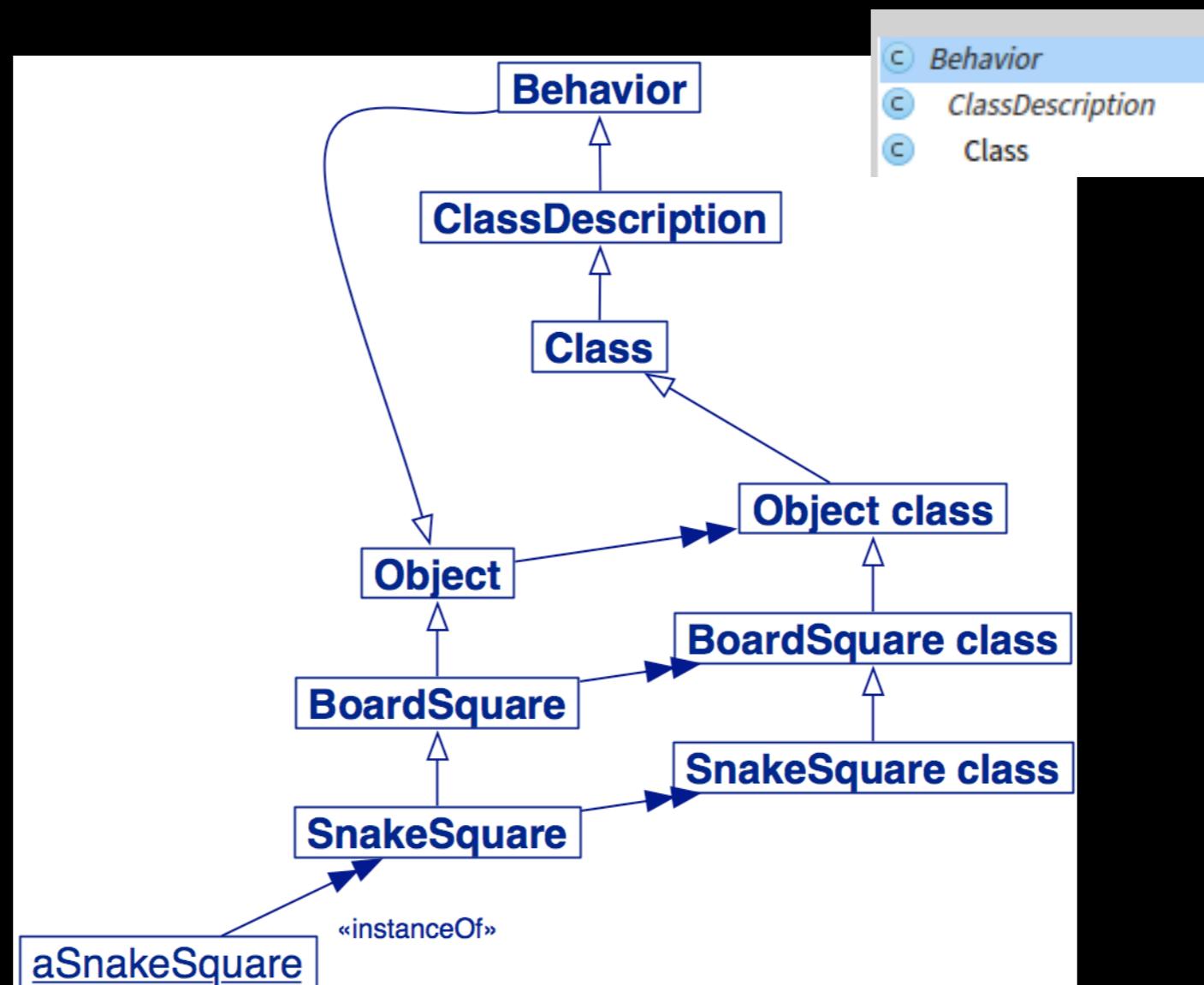
- all --
- instance creation

Two red arrows point from the **destination** and **printOn:** items in the History Navigator of the left window to the corresponding **setBack:** and **back:** methods in the bottom window, demonstrating how reflection allows access to class-level methods.

Les metaclasses en 7 parts:

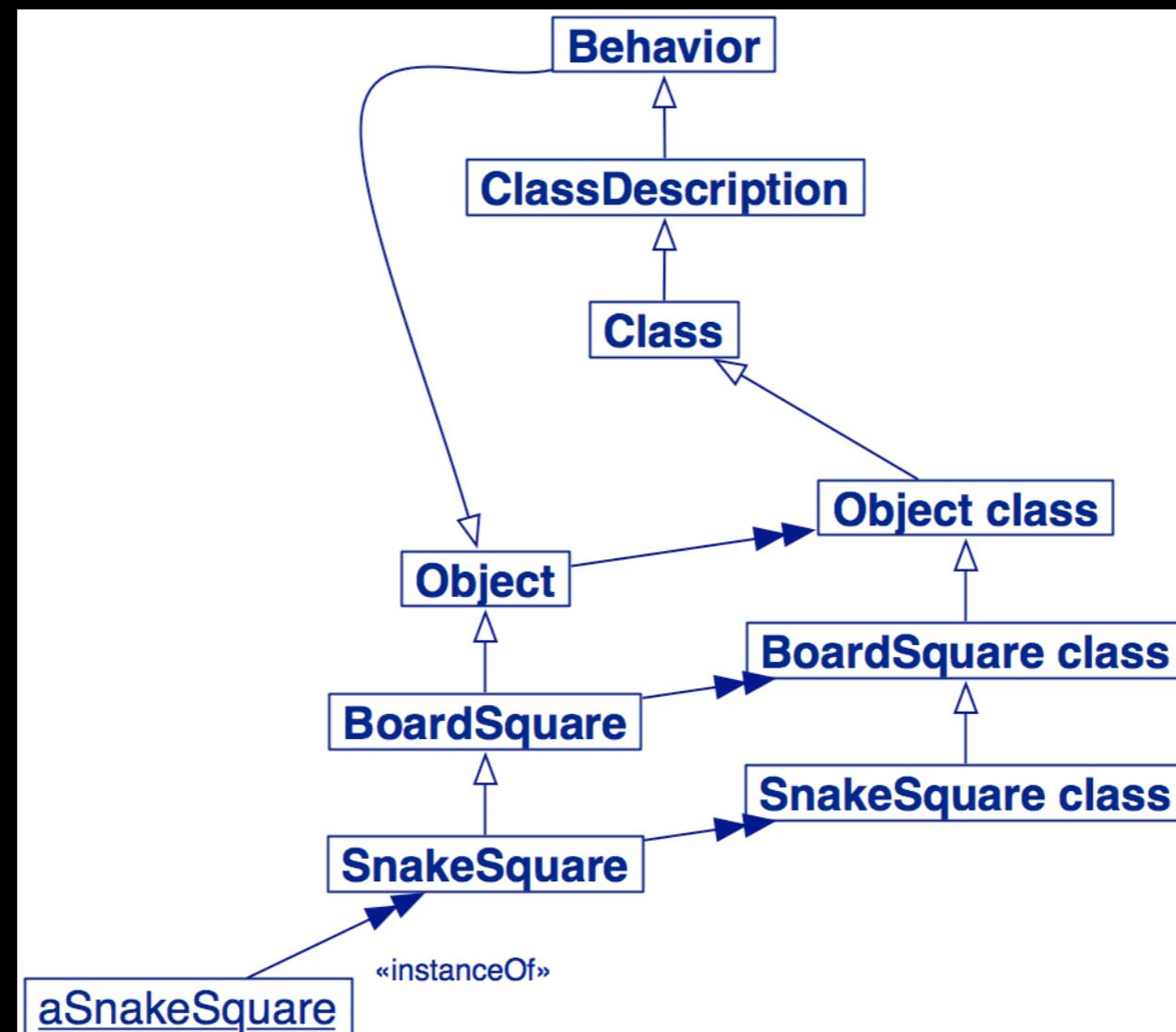
- 1- Tot objecte és instància d'una classe
- 2- Tota classe hereta eventualment d'**Object**
- 3- Tota classe és instància d'una metaclass
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- **Tota metaclass hereta de Class i Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

Cada classe és-una Class =
La metaclass de cada classe hereta de Class



CAP: Reflexió en Smalltalk

On està definit new?



Behavior (responsabilitats)

- Mínim estat necessari pels objectes que tenen instàncies
- Interfície bàsica pel compilador.

Estat:

- enllaç amb la jerarquia de classes, diccionari de mètodes, descripció de les instàncies (representació i nombre).

Mètodes:

- Crear un diccionari de mètodes, compilar mètodes.
- Crear instàncies (`#new`, `#basicNew`, `#new:`, `#basicNew:`)
- Manipular la jerarquia de classes (`#superclass:`)
- Accés (`#selectors`, `#allSelectors`, `#compiledMethodAt:`)
- Accés a instàncies i a variables (`#allInstances`, `#instVarNames`)
- Accés a la jerarquia de classes (`#superclass`, `#subclassesDo:`)
- *Testing* (`#hasMethods`, `#includesSelector:`, `#canUnderstand:`,
`#inheritsFrom:`, `#isVariable`)

ClassDescription (responsabilitats)

Afegeix un cert nombre d'utilitats a **Behavior**

- Variables d'instància amb nom (*named instance variables*).
- Organització dels mètodes en categories.
- La noció de nom (abstracte).
- Manteniment dels canvis en els *Change Sets* i el *Logging*.
- La majoria dels mecanismes necessaris per fer **fileOut**.

ClassDescription és una classe abstracte: Les utilitats que proporciona estan pensades per ser heretades per **Class** i **Metaclass**

Class (responsabilitats)

Representa el comportament comú a totes les classes

- nom, compilació, emmagatzematge de mètodes, variables d'instància...

Representació pels noms de variables de classes.

- `#addClassVarName:`, `#classVarNamed:`.

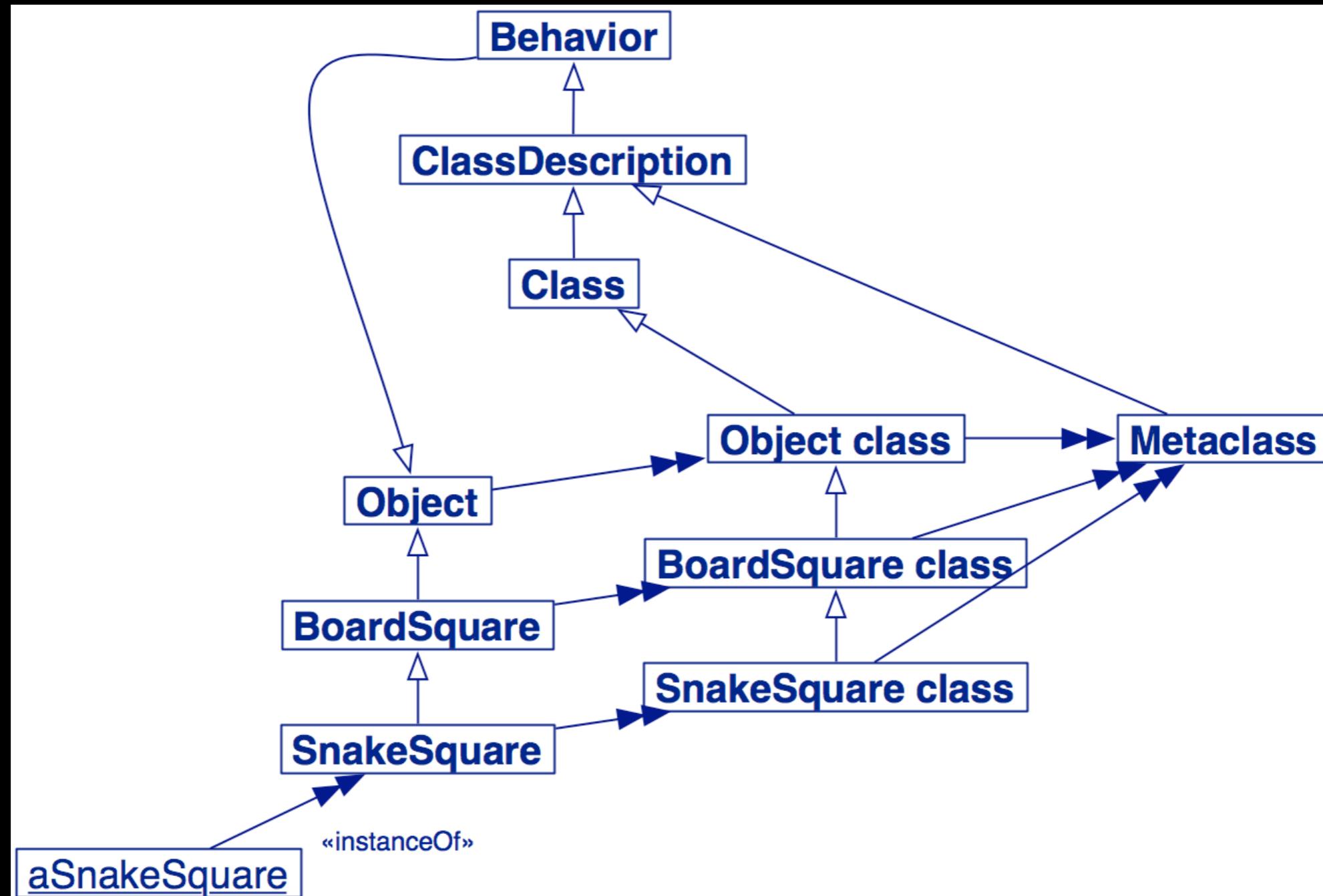
Class hereta d'Object perque Class és un Object.

Class sap com crear instàncies, per tant totes les metaclasses han d'heretar de **Class**.

Les metaclasses en 7 parts:

- 1- Tot objecte és instància d'una classe
- 2- Tota classe hereta eventualment d'**Object**
- 3- Tota classe és instància d'una metaclass
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- **Tota metaclass és instància de Metaclass**
- 7- La metaclass de **Metaclass** és instància de **Metaclass**

CAP: Reflexió en Smalltalk



Metaclass (responsabilitats)

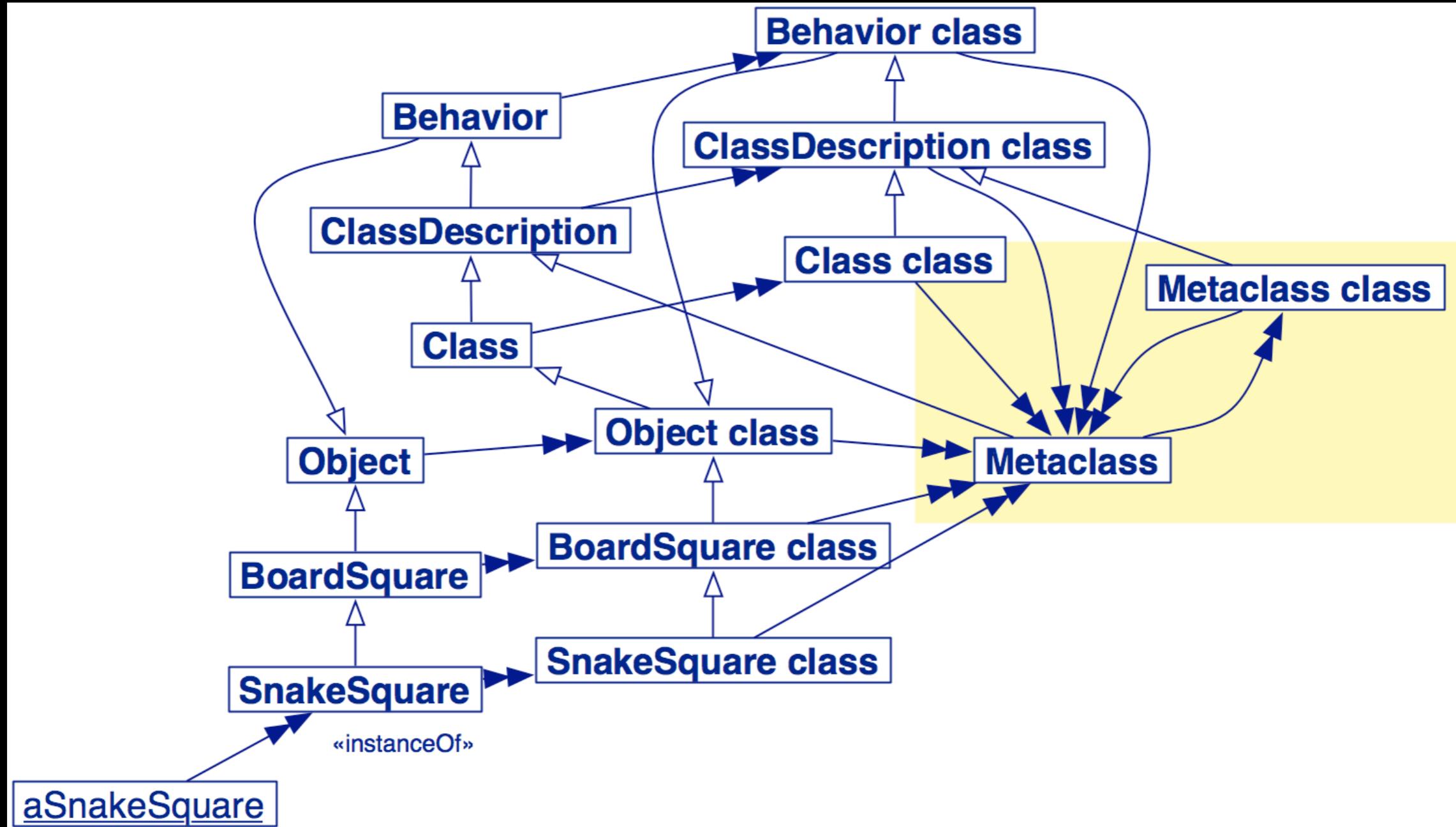
Representa el comportament comú a les metaclasses.

- Crear instàncies (`#new`, `#instanceVariableNames:`)
- Crear instàncies inicialitzades de l'única instància de la metaclass.
- Compilació de mètodes (`#subClasses`)
- Informació de la classe (*enllaços d'herència, variables d'instància...*)

Les metaclasses en 7 parts:

- 1- Tot objecte és instància d'una classe
- 2- Tota classe hereta eventualment d'**Object**
- 3- Tota classe és instància d'una metaclass
- 4- La jerarquia de metaclasses és equivalent a la jerarquia de classes
- 5- Tota metaclass hereta de **Class** i **Behavior**
- 6- Tota metaclass és instància de **Metaclass**
- 7- ***La metaclass de Metaclass és instància de Metaclass***

CAP: Reflexión en Smalltalk



```
MetaclassHierarchyTest >> testHierarchy
    "The class hierarchy"
    self assert: SnakeSquare superclass = BoardSquare.
    self assert: BoardSquare superclass = Object.
    self assert: Object superclass superclass = nil.
    "The parallel metaclass hierarchy"
    self assert: SnakeSquare class name = 'SnakeSquare class'.
    self assert: SnakeSquare class superclass = BoardSquare class.
    self assert: BoardSquare class superclass = Object class.
    self assert: Object class superclass superclass = Class.
    self assert: Class superclass = ClassDescription.
    self assert: ClassDescription superclass = Behavior.
    self assert: Behavior superclass = Object.
    "The Metaclass hierarchy"
    self assert: SnakeSquare class class = Metaclass.
    self assert: BoardSquare class class = Metaclass.
    self assert: Object class class = Metaclass.
    self assert: Class class class = Metaclass.
    self assert: ClassDescription class class = Metaclass.
    self assert: Behavior class class = Metaclass.
    self assert: Metaclass superclass = ClassDescription.
    "The fixpoint"
    self assert: Metaclass class class = Metaclass
```

Pròleg:

- Les Metaclasses en 7 parts
- *Classes indexades*
- Variables de classe-instància
- Variables de classe

Dues maneres de representar objectes

Variables d'Instància amb nom o *indexades*

- Amb nom `name` de `GamePlayer`
- Indexada: `#(Jack Jill) at: 1`

O, mirant-ho d'una altra manera

- Objectes amb referències a altres objectes
- Objectes amb *arrays* de *bytes* (*word*, *long*)
- Fem la diferència per raons d'eficiència: emmagatzemar *arrays* de *bytes* (com les *strings* de C) és més eficient que emmagatzemar un *array* de referències, cada una d'elles apuntant a un sol *byte*.

Diferentes maneres de crear classes

Indexat	Amb nom	Mètode de definició
No	Yes	#subclass: ...
Yes	Yes	#variableSubclass: ...
Yes	No	#variableByteSubclass: ...

Veieu el protocol **subclass creation** de Class

Restriccions:

- **classes amb referències (*pointer classes*):** definides amb `#subclass:` poden tenir subclasses de qualsevol mena.
- **classes amb bytes (*byte classes*):** definides amb `#variableSubclass:` només poden tenir subclasses definides amb `#variableSubclass:` o amb `#variableByteSubclass:`

Mètodes de *Testing*

The screenshot shows the Smalltalk Behavior browser interface. The left pane displays a list of categories under 'Type: Pkg1|^Pkg2|Pk.*Core\$. Classes'.

The right pane lists various metaclasses under the 'Behavior' category, which is selected. The list includes:

- obsolete subclasses
- printing
- private
- queries
- reflective operations
- system startup
- testing
- testing class hierarchy
- testing method dictionary
- traits
- user interface

Below this list are two groups of metaclasses:

- *ast-core
- *Deprecated60

Two specific metaclasses are highlighted with blue circles:

- `T isBytes (TBehavior)`
- `T isPointers (TBehavior)`

The bottom pane shows the source code for the `messageSelectorAndArgumentNames` method:messageSelectorAndArgumentNames
"comment stating purpose of message"

| temporary variable names |
statements

Page navigation: 1/5 [1] Format as you read W +L

Mètodes de *Testing*

The screenshot shows the Smalltalk Behavior browser interface. The left pane displays a list of categories under 'Type: Pkg1|^Pkg2|Pk.*Core\$'. The 'Classes' category is selected. The right pane lists various reflective operations, with the 'testing' category expanded. A specific method, 'kindOfSubclass (TBehavior)', is highlighted with a blue rounded rectangle.

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

Classes

Behavior

History Navigator

obsolete subclasses
printing
private
queries
reflective operations
system startup
testing
testing class hierarchy
testing method dictionary
traits
user interface

*ast-core
*Deprecated60

T includesBehavior: (TBehavior)
T inheritsFrom: (TBehavior)
isRootInEnvironment (TBehavior)
T kindOfSubclass (TBehavior)

messageSelectorAndArgumentNames

"comment stating purpose of message"

| temporary variable names |
statements

1/5 [1]

Format as you read W +L

Mètodes de *Testing*

The screenshot shows the Smalltalk Behavior browser interface. The top window is titled "Behavior" and contains a list of metaclasses: Behavior, ClassDescription, and Class. Below it is a "History Navigator" pane with buttons for navigating through history. The bottom window is titled "Playground" and contains a text input field with the following code:

```
Object allSubclasses select: [ :class | class isBytes ]
```

A tooltip is displayed over the code, listing various subclasses of Object. One method, `kindOfSubclass`, is highlighted with a blue rounded rectangle.

The tooltip content is as follows:

an OrderedCollection(MCMockClassH LargeInteger ByteArray LargeNegativeInteger
LargePositiveInteger UUID SocketAddress ExternalAddress Alien CompiledCode
ByteString EnumFontFamilyProc EnumWindowsProc FFICallbackReturnValue
FFICallbackThunk IntegerFlagAlien LOGFONTA OPENFILENAME
VMCallbackContext32 VMCallbackContext64 CompiledMethod CompiledBlock
ByteSymbol ENUMLOGFONTEXA)

The playground window also contains some comments and temporary variable names:

"comment stating purpose of message"
| temporary variable names |
statements

At the bottom of the playground window, there is a status bar with the text "1/5 [1]" and "Format as you read W +L".

Exemple

The screenshot shows the Smalltalk playground interface with several windows open:

- Playground Window:** Shows an array with four nil elements: `an Array [4 items] (nil nil nil nil)`. The array is displayed in a table with columns for Index and Item.
- Object Browser Window:** Shows the `Array` class selected from the `Collections-Seque` category. The browser lists various methods for the `Array` class, such as `accessing`, `comparing`, `converting`, `copying`, `printing`, `private`, and `self evaluating`.
- Code Editor Window:** Displays the source code for the `Array` class:

```
ArrayedCollection variableSubclass: #Array
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'Collections-Sequenceable'
```
- Playground Window (Bottom Right):** Shows the expression `#(1 2 3 4) class isVariable` evaluated to `true`.

Creació de classes indexades: Exemple

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

- IndexedObject Class Definition Window:** The title bar says "IndexedObject". The left pane shows the class hierarchy with "IndexedObject" selected. The right pane shows the "History Navigator" with "no messages". The code pane contains the following definition:

```
Object variableSubclass: #IndexedObject
instanceVariableNames: ''
classVariableNames: ''
package: 'ProvesCAP2019'
```

The status bar at the bottom left shows "1/4 [1]".
- Playground Window:** The title bar says "Playground". It has a "Page" tab selected. The code pane contains:

```
(IndexedObject new: 2)
at: 1 put: 'Jack';
at: 2 put: 'Jill';
at: 1
```

A yellow callout box highlights the value "'Jack'" at index 1. The status bar at the bottom right shows "Page 1 of 1" and "1/4 [1]".

Classes Indexades / Variables d'Instància

Una variable indexada s'afegeix implícitament a la llista de variables d'Instància

- Només **UNA** variable indexada (d'instància) per classe
- Accés amb **#at:** i amb **#at:put:**.
(recordeu que aquests mètodes retornen el valor, no el receptor)

Les subclasses han de ser també indexades

Pròleg:

- Les Metaclasses en 7 parts
- Classes indexades
- *Variables de classe-instància*
- Variables de classe

Variables de classe-instància

Les classes també són objectes

- Instàncies de la seva *metacasse*
(els mètodes es busquen en el diccionari de mètodes de la seva metacasse)
- Així doncs, poden tenir variables d'instància

Quan la metacasse defineix una variable d'instància nova, aleshores la seva instància (la classe) té una variable nova

- Que s'afegeix a **subclasses**, **superclass**, **methodDict**, etc.

Utilizeu variables de classe-instància per representar l'estat privat d'una classe

- Per exemple, nombre d'instàncies, superclasse, etc.
- *No serveixen per representar informació que han de compartir les instàncies*

Exemple: El patró **Singleton**

Una classe amb una sola instància:

La guardem en una variable de classe instància.

```
WebServer class
  instanceVariableNames: 'uniqueInstance'

WebServer class >> new
  self error: 'Use uniqueInstance to get the unique instance'

WebServer class >> uniqueInstance
  uniqueInstance isNil
  ifTrue: [uniqueInstance := self basicNew initialize].
  ^ uniqueInstance
```

Pròleg:

- Les Metaclasses en 7 parts
- Classes indexades
- Variables de classe-instància
- *Variables de classe*

Variables de classe = Variables compartides

Per compartir informació entre les instàncies d'una classe feu servir una *variable de classe*

- Compartida i directament accessible per totes les instàncies de la classe i subclasses
- Accessible tant a mètodes de classe com a mètodes d'instància.
- Comença amb una lletra majúscula.

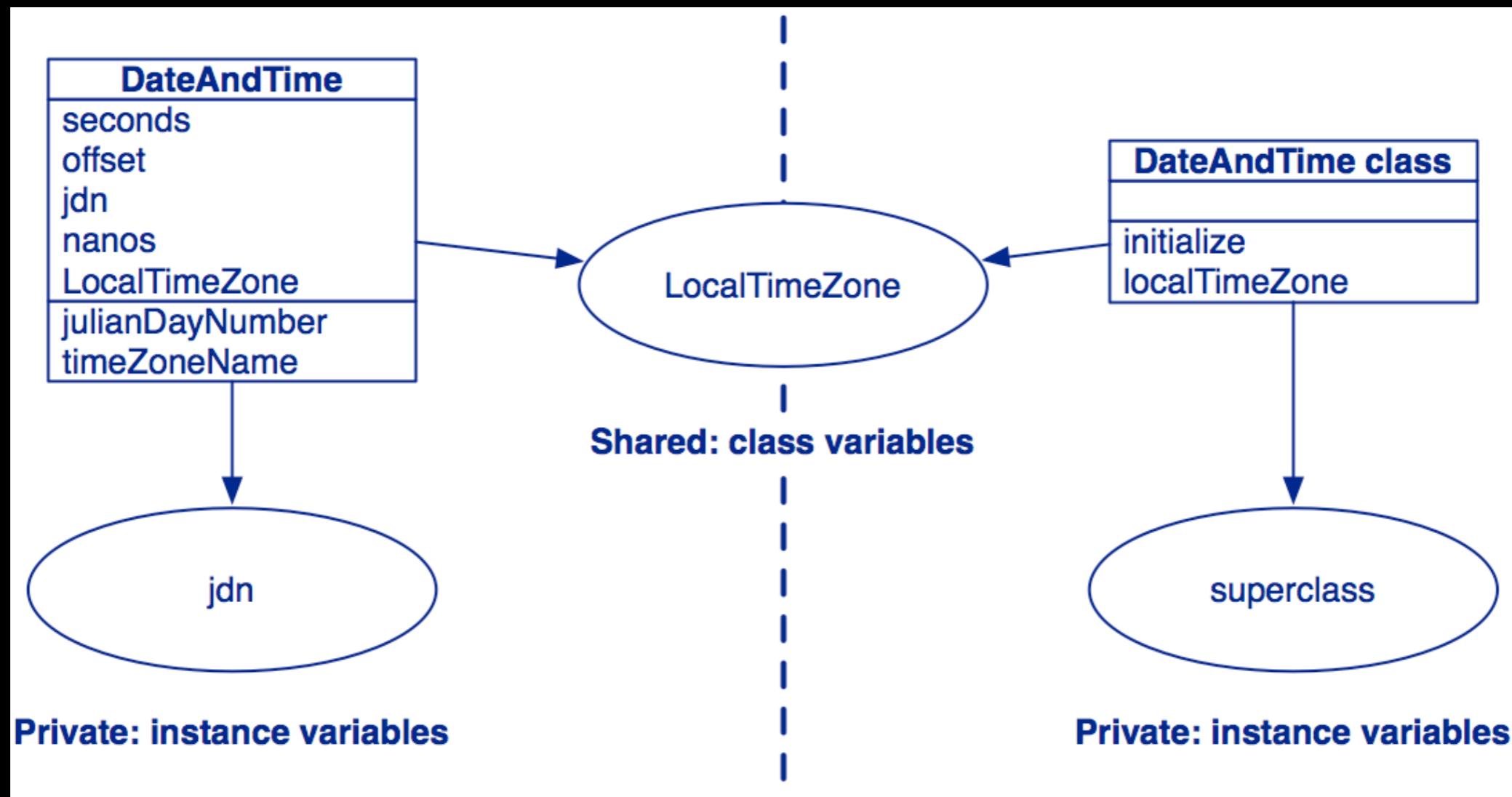
Inicialitzant Variables de Classe

Les variables de classe s'haurien d'inicialitzar en un mètode **initialize de classe**, o per inicialització *mandrosa (lazy)*

```
Magnitude subclass: #DateAndTime
instanceVariableNames: 'seconds offset jdn nanos'
classVariableNames: 'LocalTimeZone ...'
package: 'Kernel-Chronology'

DateAndTime class >> localTimeZone
"Answer the local time zone"
^ LocalTimeZone ifNil: [ LocalTimeZone := TimeZone default ]
```

Variables de Classe vs. Variables d'Instància



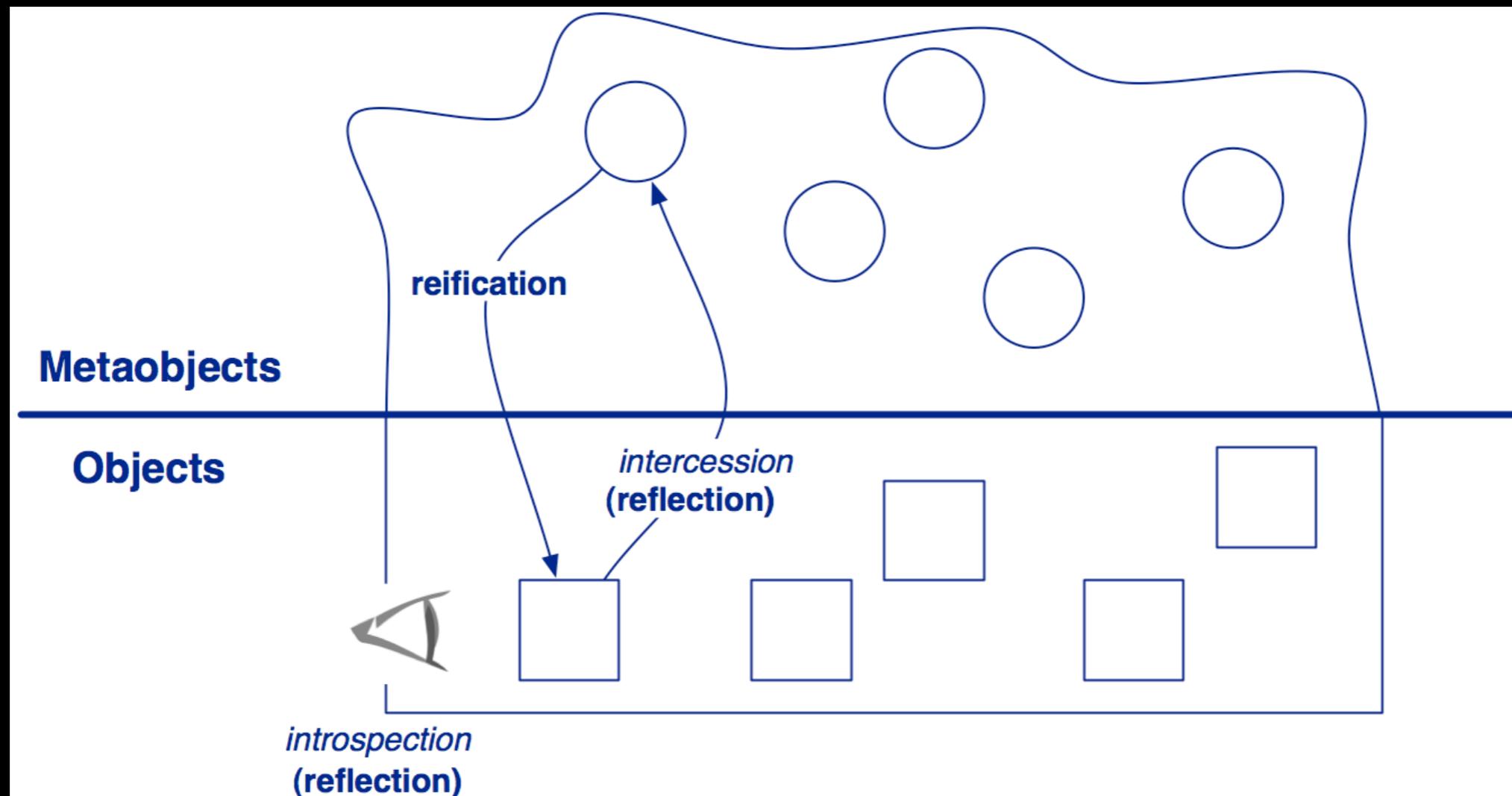
And Now for Something Completely Different...

Reflexió

LUCHILOVIO

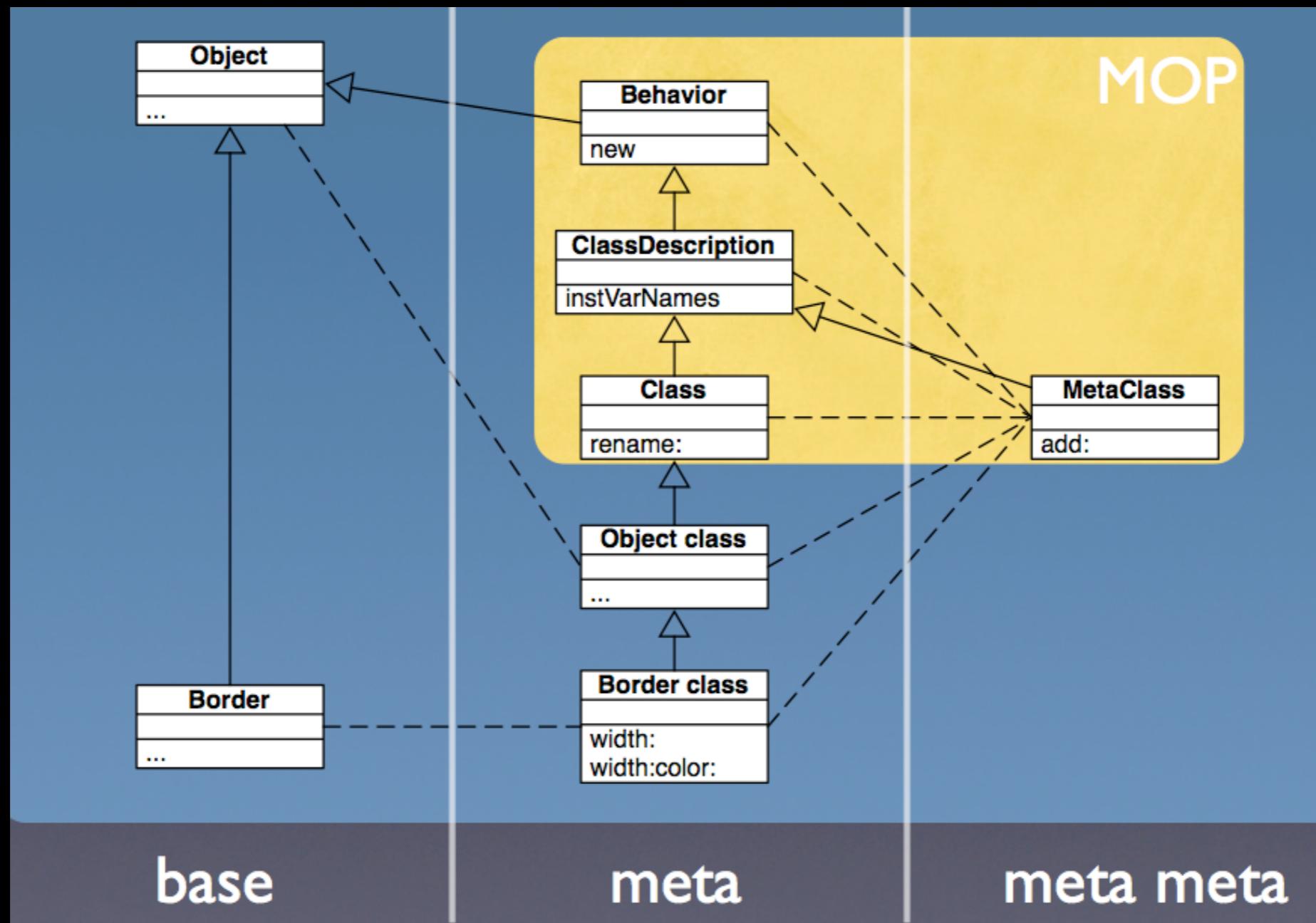
Part del material és cortesia d'Stéphane Ducasse i Marcus Denker

Recordem: *Intercessió, Introspecció, Reificació*



CAP: Reflexió en Smalltalk

Recordem: *Intercessió, Introspecció, Reificació*



CAP: Reflexió en Smalltalk



La Reflexió ens permet *examinar* i a l'hora *alterar* els meta-objectes d'un sistema

Utilitzar la reflexió per modificar un sistema que s'està executant requereix tenir una mica de cura...



Introspecció:

- Inspeccionar objectes
- Consultar el codi
- Accedir els contexts d'execució

Intercessió:

- Sobreescrivir **#doesNotUnderstand:**
- Classes Anònimes
- Method Wrappers
- Continuacions

Introspecció:

- *Inspecionar objectes*
- Consultar el codi
- Accedir els contexts d'execució

Intercessió:

- Sobreescrivir **#doesNotUnderstand**:
- Classes Anònimes
- Method Wrappers
- Continuacions

L'essència d'una classe

- **Un format** (nombre de variables d'instància, etc.)
- **Una superclasse**
- **Un diccionari de mètodes**

Exercici: On es crea la metaclass, i la classe com a instància?

Pista: començar per **subclass:**...

L'essència d'una classe

SlotClassBuilder>>#buildNewClass

History Navigator

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

- Slot
- Slot
- ClassBuilder
- Examples
- Examples-Associations
- Exception
- Layout
- Variables
- Slot-Tests
- SmartSuggestions

AddedField
ModifiedField
RemovedField
ShiftedField
UnmodifiedField
AbstractMethodUpdateStrategy
MethodRecompileStrategy
AbstractModification
InstanceModification
MethodModification
DangerousClassNotifier
SlotClassBuilder

-- all --
accessing
initialization
initialize-release
private
private validating

applyFormatChange:
applySharedVariableOrPoolChange:
applySlotChange:
applySuperclassChange:
build
buildNewClass
category:
classSlots:
classTraitComposition:
cleanUp
comment:
comment:stamp:
copyClassSlotsFromExistingClass
environment

```
buildNewClass
| metaclass newClass |
metaclass := Metaclass new.
metaclass superclass: self superMetaclass withLayoutType: FixedLayout slots: classSlots.
newClass := metaclass new.
newClass setName: name.
newClass superclass: superclass withLayoutType: self layoutClass slots: slots.
newClass declareClassVariables: sharedVariables.
newClass sharing: sharedPoolsString.
comment ifNotNil: [ newClass classComment: comment stamp: commentStamp ].
installer classAdded: newClass inCategory: category.
installer installTraitComposition: traitComposition on: newClass.
installer installTraitComposition: classTraitComposition on: metaclass.
newClass classLayout slots do: [ :each | each installingIn: newClass ].
newClass classLayout slots do: [ :each | each layoutChanged: newClass ].
^ newClass
```

1/16 [1] Format as you read W +L

L'essència d'un objecte

- Una referència a una classe
- Valors
- Pot ser especial:
 - **SmallInteger**
 - Indexat
 - Classes compactes (**CompiledMethod**, **Array** ...)

Metaobjectes vs. metaclasses

Cal distingir entre metaclasses i metaobjectes!

- Una **metaclass** és una classe les instàncies de la qual són classes
- Un **metaobjecte** és un objecte que descriu o manipula altres objectes (*diferents metaobjectes poden controlar aspectes diferents dels objectes*)

Alguns Metaobjectes

Estructura:

**Behavior, ClassDescription, Class, Metaclass,
ClassBuilder**

Semántica:

Compiler, Decompile, IRBuilder

Comportament:

CompiledMethod, BlockContext, Message, Exception

Control de l'estat:

BlockContext, Process, ProcessorScheduler

Recursos:

WeakArray

Noms:

SystemDictionary

Llibreries:

MethodDictionary, ClassOrganizer

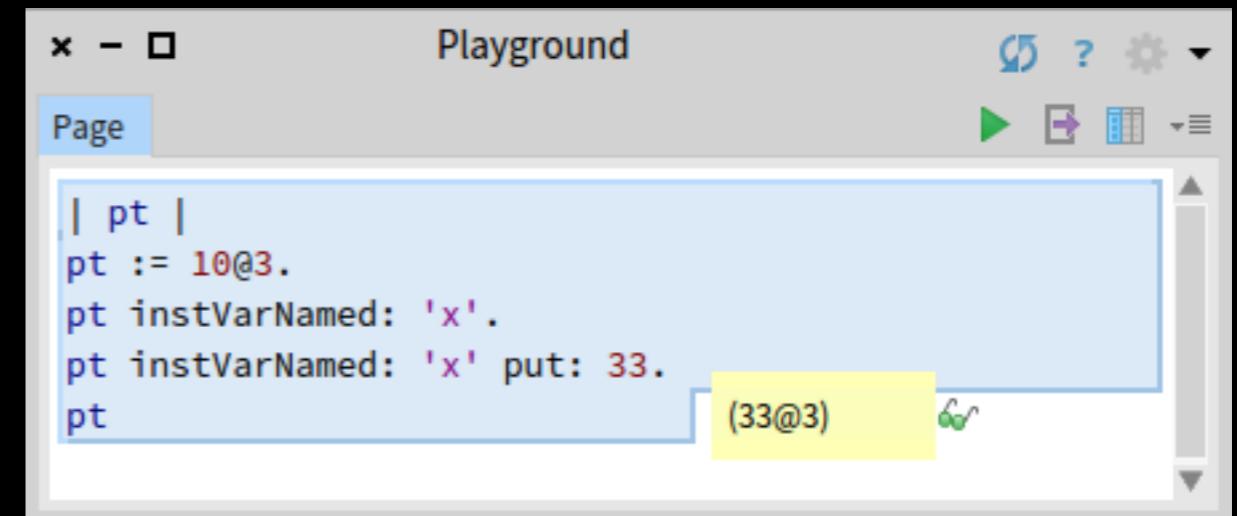
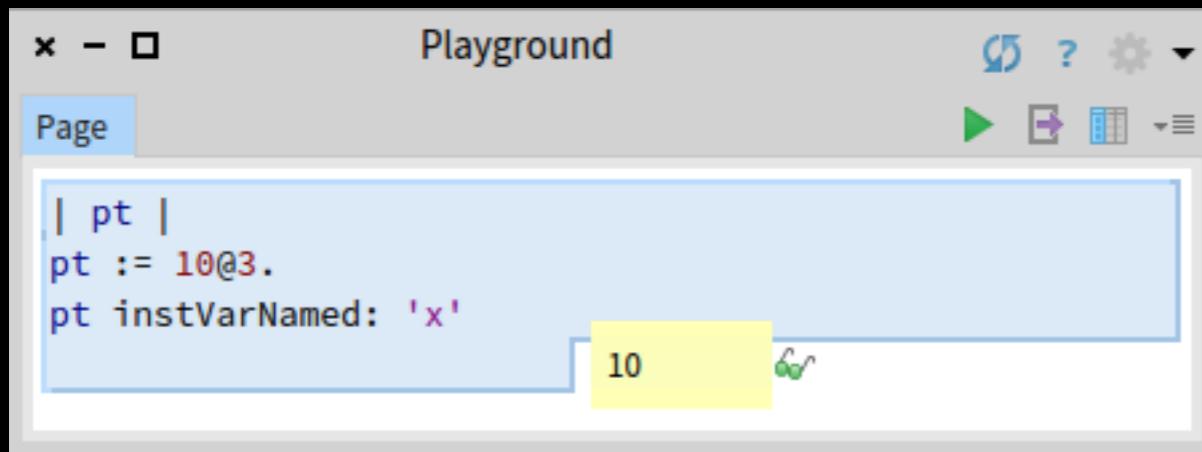
Metaoperaciones

“Meta-operations are operations that provide information about an object as opposed to information directly contained by the object ...They permit things to be done that are not normally possible”

Inside Smalltalk

Accés a l'estat

```
Object >> #instVarNamed:  
Object >> #instVarNamed:put:  
Object >> #instVarAt:  
Object >> #instVarAt:put:
```



Accés a la meta-informació

```
Object >> #class
```

```
Object >> #identityHash
```

```
'hello' class → ByteString
```

```
(10@3) class → Point
```

```
Smalltalk class → SystemDictionary
```

```
Class class → Class class
```

```
Class class class → Metaclass
```

```
Class class class class → Metaclass class
```

```
'hello' identityHash → 843767808
```

```
Object identityHash → 657408
```

```
5 identityHash → 8322625
```

Canvis

Object >> #primitiveChangeClassTo:

- canvia la classe de l'objecte receptor per la classe de l'objecte argument. Les dues classes haurien de tenir el mateix format, és a dir, la mateixa estructura a les seves instàncies.

ProtoObject >> #become:

- Intercanvia *les referències* al receptor per les de l'argument, i a l'inrevés.
- Totes les variables del sistema (*tot ell*) que referenciaven el receptor ara apunten a l'argument i vice-versa.
- Falla si cap dels dos objectes és un **SmallInteger**

Object >> #becomeForward:

- Com **become** però només en una direcció

Implementar mètodes específics de la instància*

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

- Top Bar:** Shows the current workspace title: `ReflectionTest>>#testPrimitiveChangeClassTo`.
- Left Sidebar:** A tree view of packages and classes. The `Reflection-Complete` package is selected, and its class `ReflectionTest` is highlighted.
- Middle Area:** The `History Navigator` pane shows a list of running tests: `testBecome`, `testBecomeForward`, and `testPrimitiveChangeClassTo`.
- Bottom Area:** The code editor displays the implementation of the `testPrimitiveChangeClassTo` method. The code uses reflection to change the superclass of a behavior object.

```
testPrimitiveChangeClassTo
| behavior model |
behavior := Behavior new.
behavior superclass: Model.
behavior setFormat: Model format.
model := Model new.
model primitiveChangeClassTo: behavior new.
behavior compile: 'thisIsATest ^ 2'.
self assert: model thisIsATest = 2.
self should: [ Model new thisIsATest ] raise: MessageNotUnderstood
```

At the bottom right of the code editor, there are checkboxes for "Format as you read" and "W +L".

#become:

Intercanvia totes les referències d'un objecte a l'altre i vice-versa

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

- Top Bar:** Shows the title "ReflectionTest>>#testBecome".
- Left Panel (Variables):** Displays a tree view of packages and classes. The "Reflection-Complete" package is selected. Under it, the "ReflectionTest" class is highlighted.
- Middle Panel (History Navigator):** Shows a list of running tests: "testBecome", "testBecomeForward", and "testPrimitiveChangeClassTo".
- Bottom Panel (Code Editor):** Displays the source code for the "testBecome" method:

```
testBecome
| pt1 pt2 pt3 |
pt1 := 0 @ 0.
pt2 := pt1.
pt3 := 100 @ 100.
pt1 become: pt3.
self assert: pt1 = (100 @ 100).
self assert: pt1 == pt2.
self assert: pt3 = (0 @ 0)
```
- Bottom Right:** Includes checkboxes for "Format as you read" and "W +L".

#becomeForward:

Intercanvia totes les referències d'un objecte a l'altre

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

- Title Bar:** ReflectionTest>>#testBecomeForward
- Toolbars:** Standard toolbar with icons for New, Open, Save, etc.
- Left Panel (Type Browser):** Shows the current package selection: Pkg1|Pkg2|Pk.*Core\$. It lists several classes and packages:
 - Refactoring-Tests-Core
 - Refactoring-Tests-Critics
 - Refactoring-Tests-Enviro
 - Reflection-Complete (selected)
 - ReflectionMirrors-Primiti
 - Reflectivity
 - Reflectivity-Examples
- Middle Panel (Code Editor):** Displays the source code for the `testBecomeForward` method:

```
testBecomeForward
| pt1 pt2 pt3 |
pt1 := 0 @ 0.
pt2 := pt1.
pt3 := 100 @ 100.
pt1 becomeForward: pt3.
self assert: pt1 = (100 @ 100).
self assert: pt1 == pt2.
self assert: pt2 == pt3
```
- Right Panel (History Navigator):** Shows a history of recent methods:
 - all --
 - running
 - testBecome
 - testBecomeForward (highlighted)
 - testPrimitiveChangeClassTo
- Status Bar:** Shows "1/9 [1]" at the bottom left and "Format as you read" with checkboxes for "W" and "+L" at the bottom right.

Introspecció:

- Inspecionar objectes
- ***Consultar el codi***
- Accedir els contexts d'execució

Intercessió:

- Sobreescrivir **#doesNotUnderstand:**
- Classes Anònimes
- Method Wrappers
- Continuacions

Code Metrics

```
Collection allSuperclasses size → 2
    Collection allSelectors size → 636
Collection allInstVarNames size → 0
    Collection selectors size → 200
Collection instVarNames size → 0
    Collection subclasses size → 14
Collection allSubclasses size → 101
    Collection linesOfCode → 1088
```

(en una imatge nova)

System Navigation

The screenshot shows the System Navigation interface. At the top, there's a toolbar with icons for search, help, and settings. Below it, a tab bar has 'Page' selected. A code input field contains the expression: `SystemNavigation default browseAllImplementorsOf: #,`. The main window displays a table titled 'Implementors of , [22]'. The table lists various classes and their implementors, grouped by category. The first few rows are:

Category	Class	Implementor
[FileSystem-Core]	AbstractFileReference (copying)	,
[FileSystem-Core]	FileReference (navigating)	,
[ClassAnnotation]	AnnotationContext (converting)	,
[ClassAnnotation]	CompositeAnnotationContext (converting)	,
[Announcements-Core]	Announcement class (public)	,
[Collections-Abstract]	Collection (copying)	,
[Announcements-Core]	AnnouncementSet (adding)	,
[Collections-Unordered]	Matrix (copying)	,

Below the table is a 'Filter...' button. At the bottom of the window, there's a navigation bar with tabs: 'Browse', 'Users', 'Senders', 'Implementors' (which is selected), 'Version', and 'Source'. The bottom-most part of the window shows some code snippets, including:
`, extension`
`^ self resolve, extension`

Recapitulem: Les classes també són objectes

Object:

- Arrel de l'herència
- Comportament per defecte.
- Comportament mínim

Behavior:

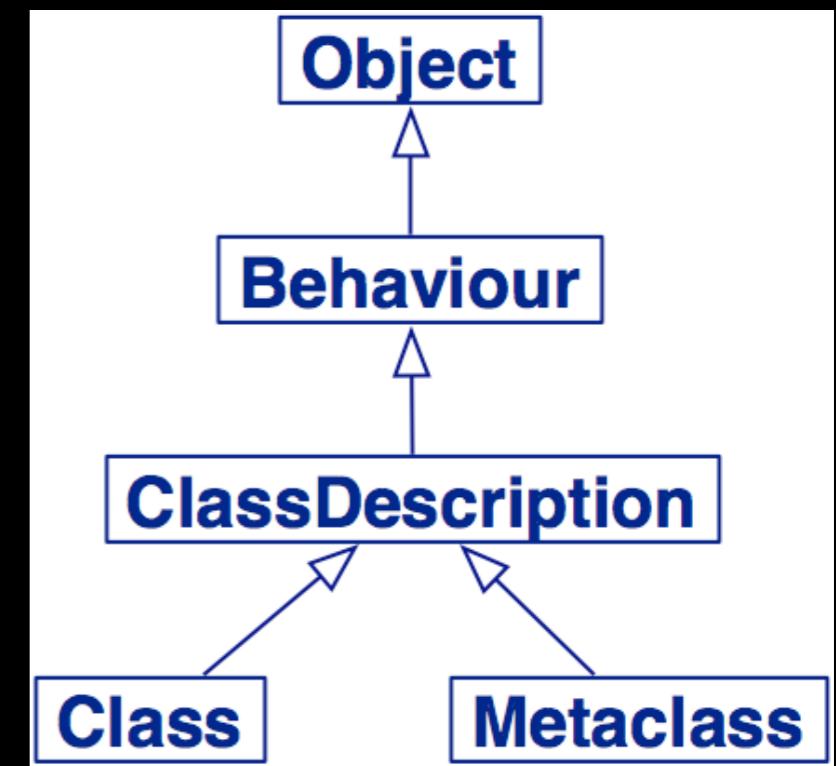
- Essència d'una classe
- Classe anònima.
- Format, diccionari de mètodes, superclasse

ClassDescription:

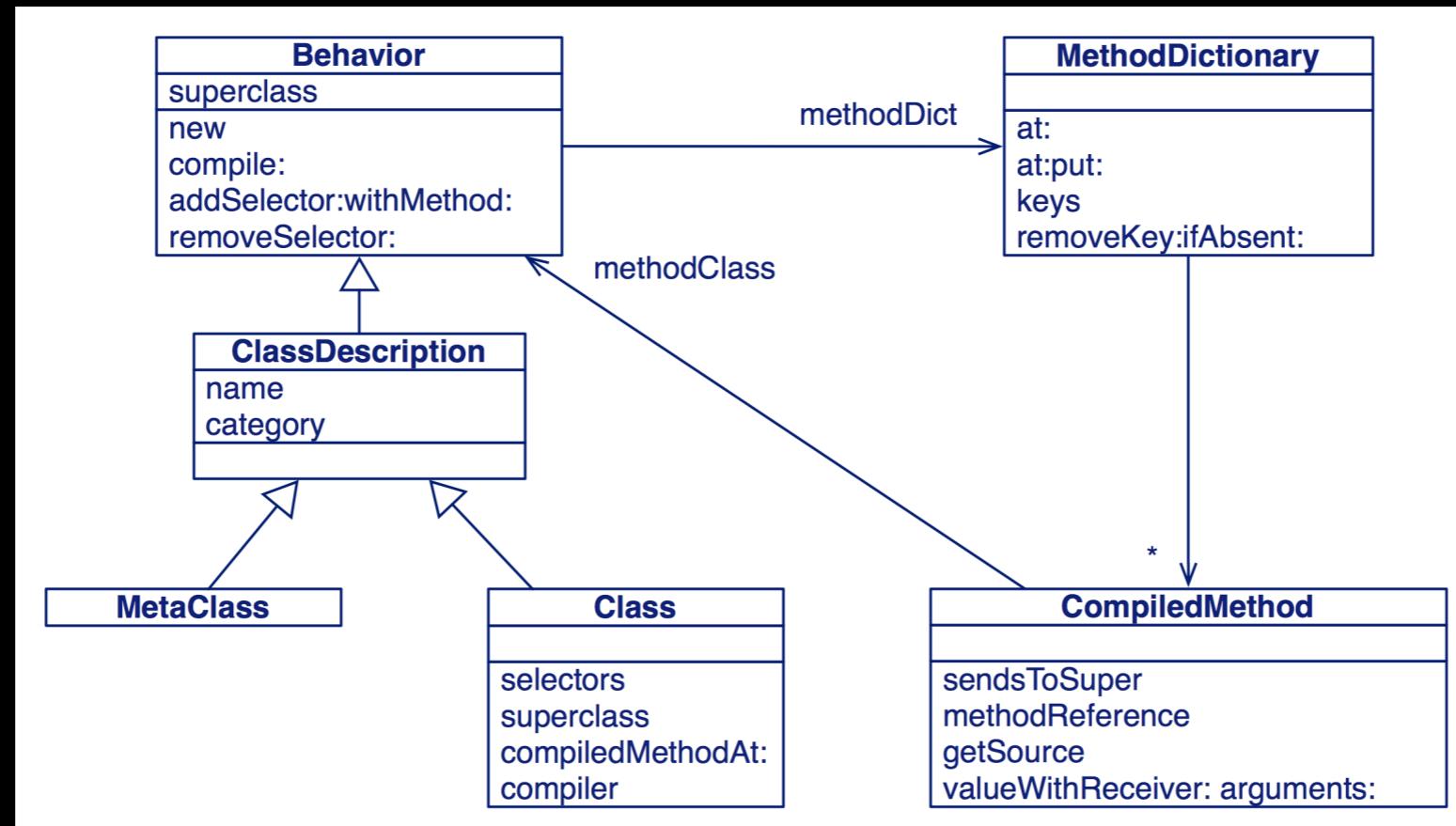
- Representació i organització

Metaclass:

- Única instància



Les classes són contenidors de CompiledMethods



Les classes són contenidors de CompiledMethods

Inspector on an OrderedCollection class (OrderedCollection)

an OrderedCollection class (OrderedCollection)

Raw	Defin...	Comi...	All Ref	All Re...	Meth...	InstV...	Meta																						
<table border="1"><thead><tr><th>Variable</th><th>Value</th></tr></thead><tbody><tr><td>self</td><td>OrderedCollection</td></tr><tr><td>superclass</td><td>SequenceableCollection</td></tr><tr><td>methodDict</td><td>a MethodDictionary [61 items] (#add:->OrderedCollection>>#a...)</td></tr><tr><td>format</td><td>65539</td></tr><tr><td>layout</td><td>a FixedLayout</td></tr><tr><td>instanceVariables</td><td>nil</td></tr><tr><td>organization</td><td>a ClassOrganization</td></tr><tr><td>subclasses</td><td>an Array [5 items] (SortedCollection)</td></tr><tr><td>name</td><td>#OrderedCollection</td></tr><tr><td>classPool</td><td>a Dictionary [0 items] ()</td></tr></tbody></table>								Variable	Value	self	OrderedCollection	superclass	SequenceableCollection	methodDict	a MethodDictionary [61 items] (#add:->OrderedCollection>>#a...)	format	65539	layout	a FixedLayout	instanceVariables	nil	organization	a ClassOrganization	subclasses	an Array [5 items] (SortedCollection)	name	#OrderedCollection	classPool	a Dictionary [0 items] ()
Variable	Value																												
self	OrderedCollection																												
superclass	SequenceableCollection																												
methodDict	a MethodDictionary [61 items] (#add:->OrderedCollection>>#a...)																												
format	65539																												
layout	a FixedLayout																												
instanceVariables	nil																												
organization	a ClassOrganization																												
subclasses	an Array [5 items] (SortedCollection)																												
name	#OrderedCollection																												
classPool	a Dictionary [0 items] ()																												

"OrderedCollection"
self

a MethodDictionary [61 items] (#add:->OrderedCollection>>#a...)

Items	Keys	Raw	Meta																																												
<table border="1"><thead><tr><th>Key</th><th>Value</th></tr></thead><tbody><tr><td>#errorConditionNotSatisfied</td><td>OrderedCollection>>#errorConditionNotSatisfied</td></tr><tr><td>#postCopyFrom:to:</td><td>OrderedCollection>>#postCopyFrom:to:</td></tr><tr><td>#with:collect:</td><td>OrderedCollection>>#with:collect:</td></tr><tr><td>#sort:</td><td>OrderedCollection>>#sort:</td></tr><tr><td>#reset</td><td>OrderedCollection>>#reset</td></tr><tr><td>#addAllFirstUnlessAlreadyPresent</td><td>OrderedCollection>>#addAllFirstUnlessAlreadyPresent</td></tr><tr><td>#do:</td><td>OrderedCollection>>#do:</td></tr><tr><td>#at:put:</td><td>OrderedCollection>>#at:put:</td></tr><tr><td>#removeAllSuchThat:</td><td>OrderedCollection>>#removeAllSuchThat:</td></tr><tr><td>#size</td><td>OrderedCollection>>#size</td></tr><tr><td>#collector</td><td>OrderedCollection>>#collector</td></tr><tr><td>#removeFirst:</td><td>OrderedCollection>>#removeFirst:</td></tr><tr><td>#find:</td><td>OrderedCollection>>#find:</td></tr><tr><td>#reverseDo:</td><td>OrderedCollection>>#reverseDo:</td></tr><tr><td>#reject:</td><td>OrderedCollection>>#reject:</td></tr><tr><td>#removeFirst</td><td>OrderedCollection>>#removeFirst</td></tr><tr><td>#asArray</td><td>OrderedCollection>>#asArray</td></tr><tr><td>#collect:</td><td>OrderedCollection>>#collect:</td></tr><tr><td>#remove:ifAbsent:</td><td>OrderedCollection>>#remove:ifAbsent:</td></tr><tr><td>#join:</td><td>OrderedCollection>>#join:</td></tr><tr><td>#ensureBoundsFrom:to:</td><td>OrderedCollection>>#ensureBoundsFrom:to:</td></tr></tbody></table>				Key	Value	#errorConditionNotSatisfied	OrderedCollection>>#errorConditionNotSatisfied	#postCopyFrom:to:	OrderedCollection>>#postCopyFrom:to:	#with:collect:	OrderedCollection>>#with:collect:	#sort:	OrderedCollection>>#sort:	#reset	OrderedCollection>>#reset	#addAllFirstUnlessAlreadyPresent	OrderedCollection>>#addAllFirstUnlessAlreadyPresent	#do:	OrderedCollection>>#do:	#at:put:	OrderedCollection>>#at:put:	#removeAllSuchThat:	OrderedCollection>>#removeAllSuchThat:	#size	OrderedCollection>>#size	#collector	OrderedCollection>>#collector	#removeFirst:	OrderedCollection>>#removeFirst:	#find:	OrderedCollection>>#find:	#reverseDo:	OrderedCollection>>#reverseDo:	#reject:	OrderedCollection>>#reject:	#removeFirst	OrderedCollection>>#removeFirst	#asArray	OrderedCollection>>#asArray	#collect:	OrderedCollection>>#collect:	#remove:ifAbsent:	OrderedCollection>>#remove:ifAbsent:	#join:	OrderedCollection>>#join:	#ensureBoundsFrom:to:	OrderedCollection>>#ensureBoundsFrom:to:
Key	Value																																														
#errorConditionNotSatisfied	OrderedCollection>>#errorConditionNotSatisfied																																														
#postCopyFrom:to:	OrderedCollection>>#postCopyFrom:to:																																														
#with:collect:	OrderedCollection>>#with:collect:																																														
#sort:	OrderedCollection>>#sort:																																														
#reset	OrderedCollection>>#reset																																														
#addAllFirstUnlessAlreadyPresent	OrderedCollection>>#addAllFirstUnlessAlreadyPresent																																														
#do:	OrderedCollection>>#do:																																														
#at:put:	OrderedCollection>>#at:put:																																														
#removeAllSuchThat:	OrderedCollection>>#removeAllSuchThat:																																														
#size	OrderedCollection>>#size																																														
#collector	OrderedCollection>>#collector																																														
#removeFirst:	OrderedCollection>>#removeFirst:																																														
#find:	OrderedCollection>>#find:																																														
#reverseDo:	OrderedCollection>>#reverseDo:																																														
#reject:	OrderedCollection>>#reject:																																														
#removeFirst	OrderedCollection>>#removeFirst																																														
#asArray	OrderedCollection>>#asArray																																														
#collect:	OrderedCollection>>#collect:																																														
#remove:ifAbsent:	OrderedCollection>>#remove:ifAbsent:																																														
#join:	OrderedCollection>>#join:																																														
#ensureBoundsFrom:to:	OrderedCollection>>#ensureBoundsFrom:to:																																														

Invocar un missatge pel nom

```
Object >> #perform:
```

```
Object >> #perform:withArguments:
```

Demanar un objecte que executi un missatge.

La cerca de mètodes és l'habitual

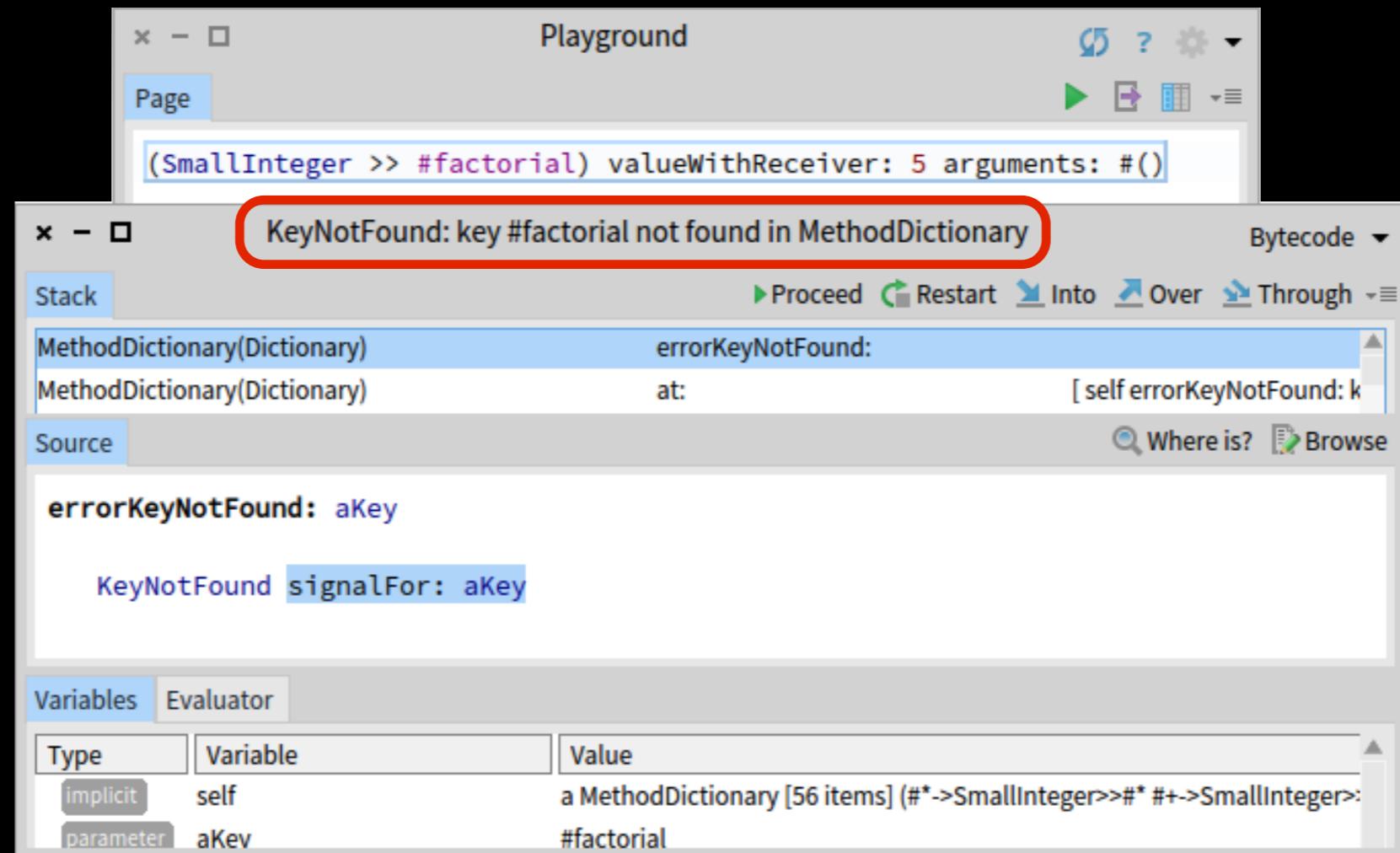
```
5 factorial → 120
```

```
5 perform: #factorial → 120
```

Executar un mètode compilat

CompiledMethod >> #valueWithReceiver:arguments:

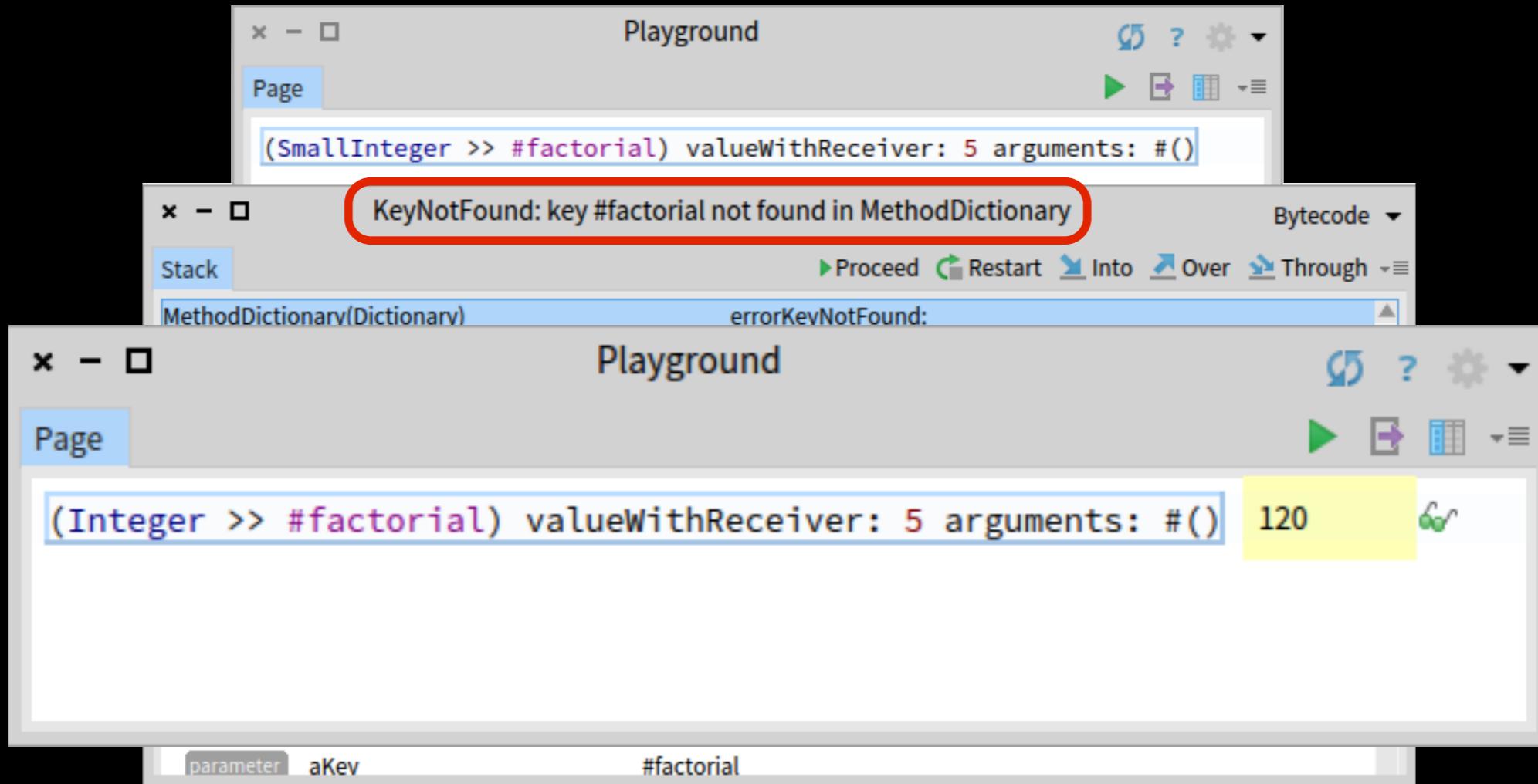
No es fa cap cerca de mètodes!!



Executar un mètode compilat

CompiledMethod >> #valueWithReceiver:arguments:

No es fa cap cerca de mètodes!!



**Exemple: Trobar els mètodes que fan referència
a `thisContext` dins el codi font.**

Utilitzant RBBrowserEnvironment

The screenshot shows the RBBrowserEnvironment interface with the following details:

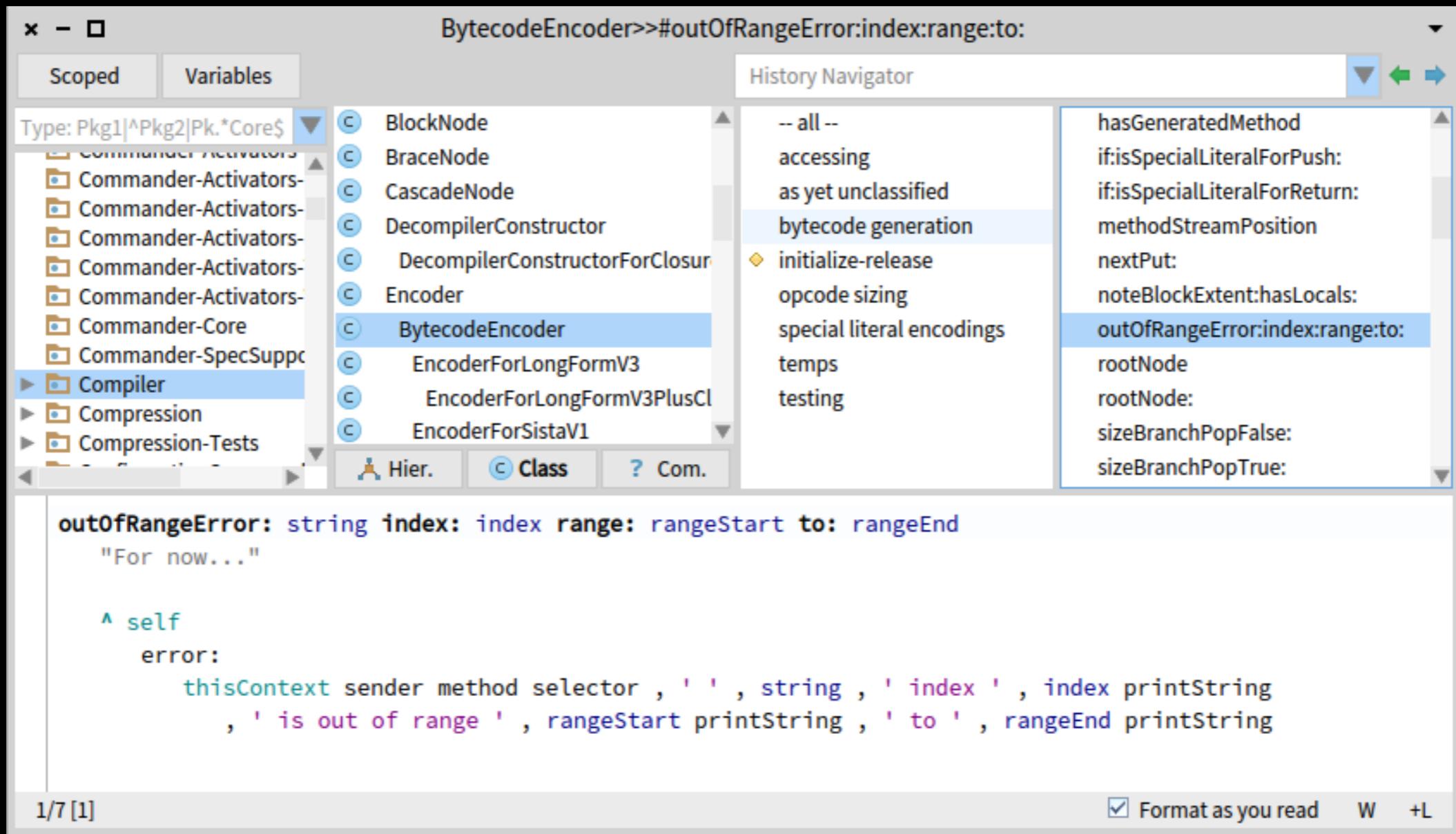
- Playground Tab:** Contains the code: `Nautilus openInEnvironment: (RBBrowserEnvironment new selectMethods: [:each | (each sourceCode findString: 'thisContext') > 0])`.
- History Navigator:** Shows the method `outOfRangeError:index:range:to:` under the category `bytecode generation`.
- Scopes Tab:** Shows the package tree:
 - Type: Pkg1|^Pkg2|Pk.*Core
 - Alien
 - Collections-Streams
 - Collections-Tests
 - Compiler (selected)
 - FFI-Kernel
 - Flashback-Decompile
- Code View:** Displays the implementation of `outOfRangeError:`:

```
outOfRangeError: string index: index range: rangeStart to: rangeEnd
    "For now..."

    ^ self
    error:
        thisContext sender method selector , ' ', string , ' index ' , index printString
        , ' is out of range ' , rangeStart printString , ' to ' , rangeEnd printString
```
- Bottom Status:** Shows page 1/7 [1] and options for reading mode (Format as you read), width (W), and line length (+L).

CAP: Reflexió en Smalltalk

Exemple: Trobar els mètodes que fan referència a `thisContext` dins el codi font.
Utilitzant RBBrowserEnvironment



The screenshot shows the RBBrowserEnvironment interface with the following details:

- Title Bar:** BytecodeEncoder>#outOfRangeError:index:range:to:
- Toolbar:** Includes buttons for 'x - □' (close), 'History Navigator' (with back and forward arrows), and other standard browser controls.
- Left Panel (Scope):** Shows the current scope with the type filter set to "Pkg1|^Pkg2/Pk.*Core\$". It lists several packages and classes, with "Compiler" currently selected.
- Center Panel (History Navigator):** A tree view of method categories:
 - all --
 - accessing
 - as yet unclassified
 - bytecode generation** (selected)
 - initialize-release
 - opcode sizing
 - special literal encodings
 - temp
 - testing
- Right Panel (List):** A list of methods:
 - hasGeneratedMethod
 - if:isSpecialLiteralForPush:
 - if:isSpecialLiteralForReturn:
 - methodStreamPosition
 - nextPut:
 - noteBlockExtent:hasLocals:
 - outOfRangeError:index:range:to:** (selected)
 - rootNode
 - rootNode:
 - sizeBranchPopFalse:
 - sizeBranchPopTrue:
- Bottom Panel (Code View):** Displays the source code for the `outOfRangeError` method:

```
outOfRangeError: string index: index range: rangeStart to: rangeEnd
    "For now..."
```

^ self

```
error:
    thisContext sender method selector , ' ', string , ' index ' , index printString
    , ' is out of range ' , rangeStart printString , ' to ' , rangeEnd printString
```
- Page Number:** 1/7 [1]
- Page Footer:** Format as you read, W, +L

Introspecció:

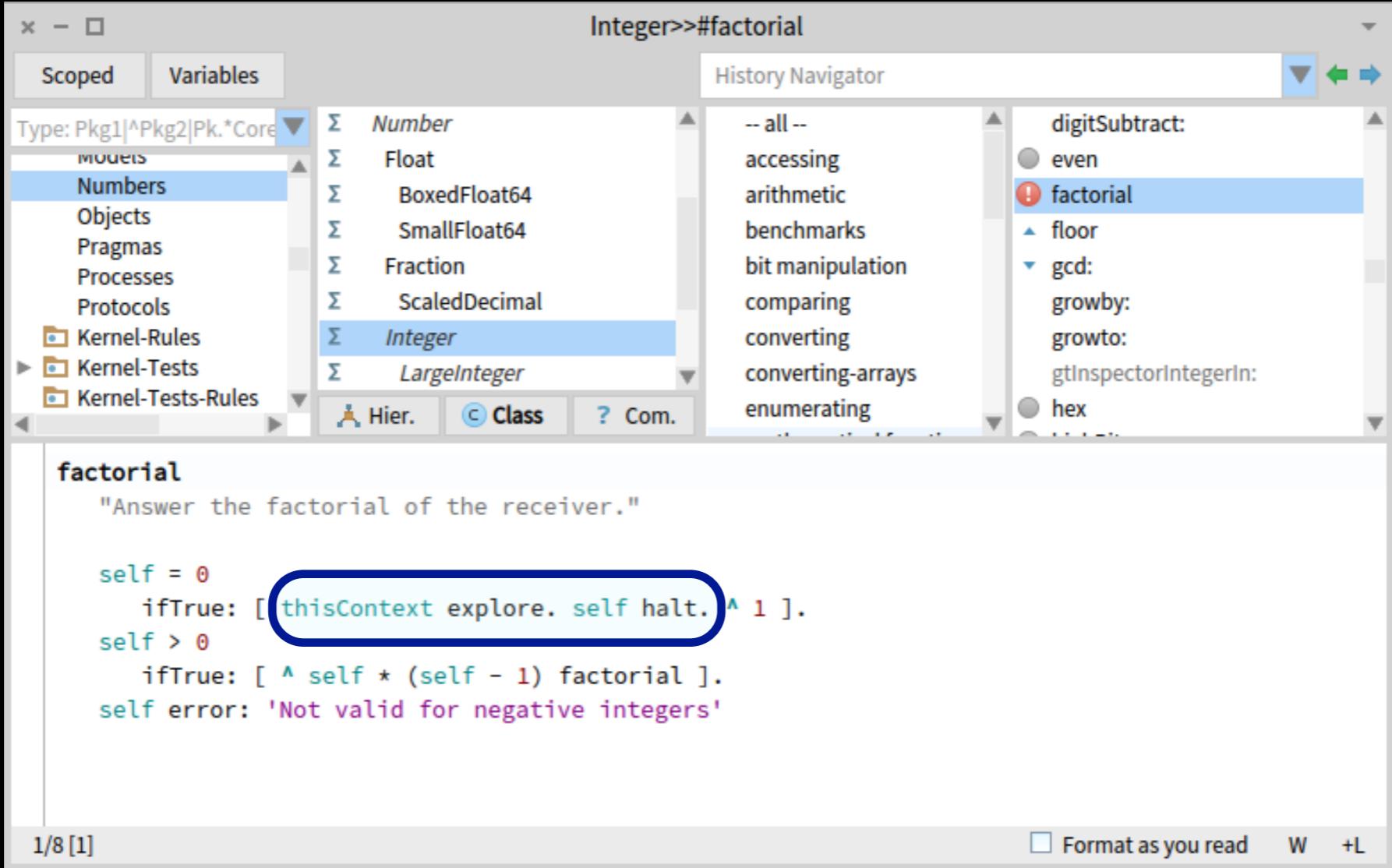
- Inspeccionar objectes
- Consultar el codi
- ***Accedir els contexts d'execució***

Intercessió:

- Sobreescrivir **#doesNotUnderstand:**
- Classes Anònimes
- Method Wrappers
- Continuacions

Accedir a la pila d'execució

La pila d'execució pot ser *reificada* i *manipulada* quan desitjem. **thisContext** és una pseudo-variable que ens dóna accés a la pila



The screenshot shows the Smalltalk IDE interface. The top navigation bar has tabs for 'Scoped' and 'Variables'. The main area is titled 'Integer>>#factorial'. On the left, there's a 'History Navigator' pane containing a list of methods like 'digitSubtract:', 'even', 'factorial', 'floor', 'gcd:', 'growby:', 'growto:', 'gtInspectorIntegerIn:', and 'hex'. Below the navigator is a code editor window. The code for the factorial method is:

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

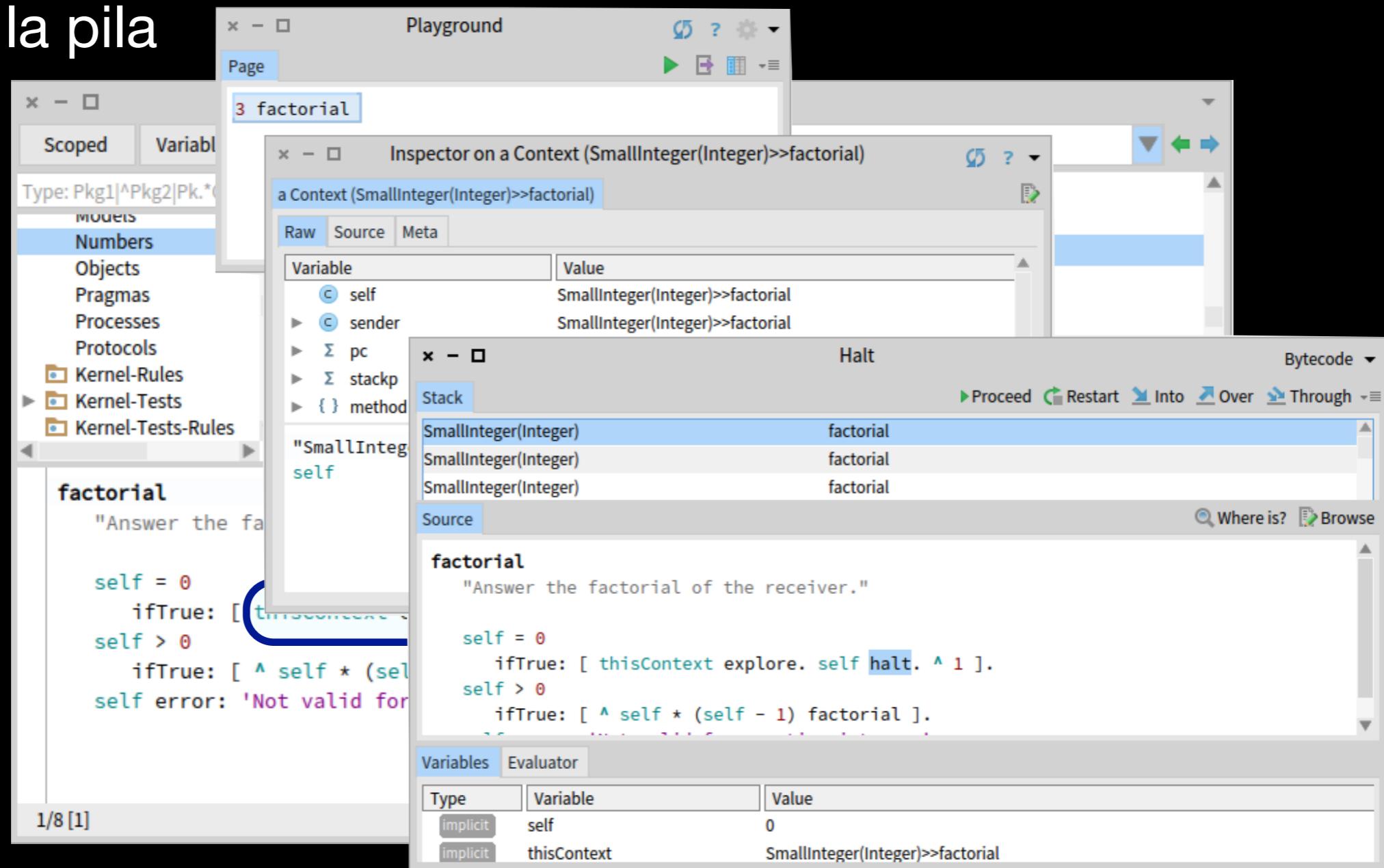
    self = 0
        ifTrue: [ thisContext explore. self halt. ^ 1 ].
    self > 0
        ifTrue: [ ^ self * (self - 1) factorial ].
    self error: 'Not valid for negative integers'
```

The line `thisContext explore. self halt. ^ 1` is highlighted with a blue oval.

At the bottom of the code editor, there are status icons for 'Format as you read', 'W', and '+L'. The bottom left corner shows '1/8 [1]'.

Accedir a la pila d'execució

La pila d'execució pot ser *reificada* i *manipulada* quan desitjem. **thisContext** és una pseudo-variable que ens dóna accés a la pila



Què passa quan s'executa un mètode?

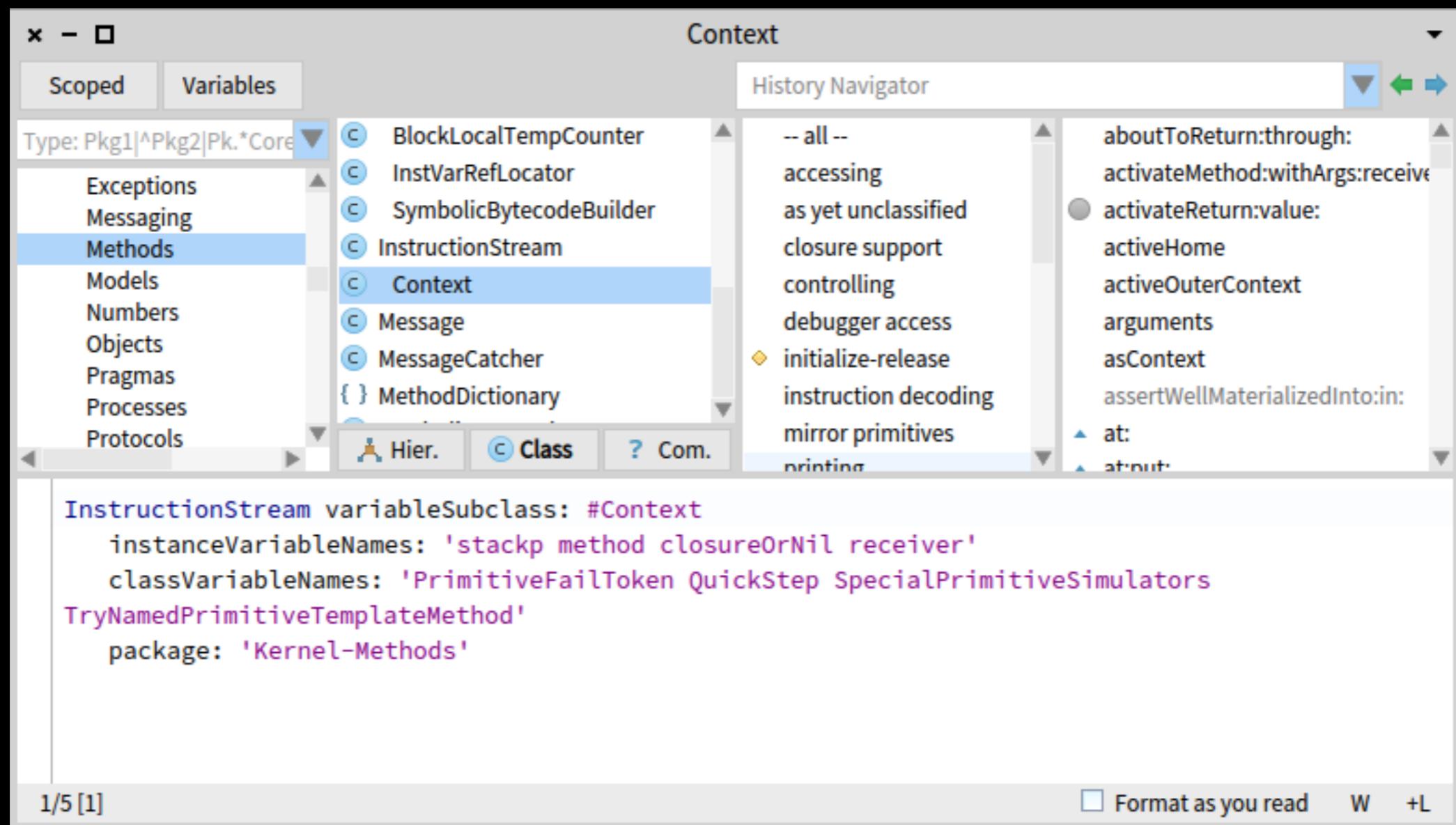
Cal espai per:

- variables temporals
- recordar on hem de retornar

TOT és un objecte!

- Modelitzem aquest espai amb objectes
- Classe **Context**

Què passa quan s'executa un mètode?



Context

Context gestiona l'espai associat a l'execució d'un **CompiledMethod**

- *Program counter* (pc)
- el mètode en sí
- *sender* i *receiver*

El *sender* és el previ **Context**

- La cadena de *senders* és una pila
- creix i decreix amb les invocacions i els retorns

Exemple: Aturada contextual

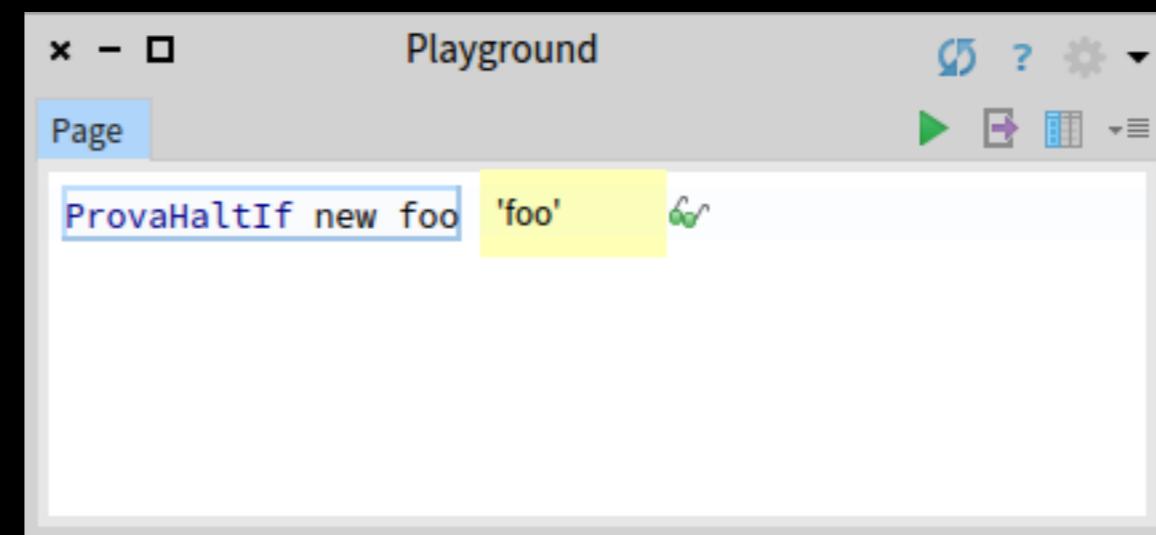
No podem posar **halt** en mètodes que s'utilitzen sovint
(p.ex. **OrderedCollection>>#add:**)

Idea: aturar-se només si el mètode ha estat invocat des d'algun altre mètode amb un *selector* determinat.
Podriem implementar-lo així

```
haltIf: aSelector
  | context |
  context := thisContext.
  [context sender isNil]
    whileFalse:
      [context := context sender.
       (context selector = aSelector)
         ifTrue: [ Halt signal ] ].
```

Object >> haltIf:

```
ProvaHaltIf >> foo
    self haltIf: #bar.
^ 'foo'
```



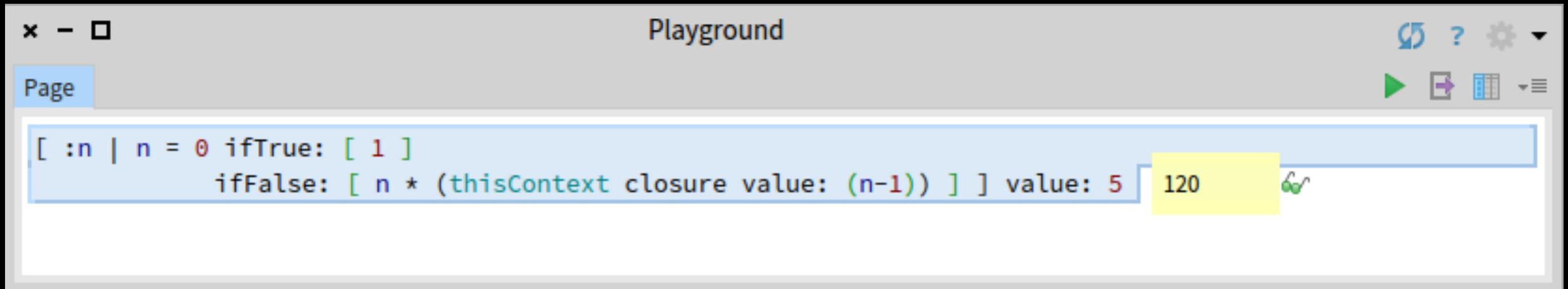
```
ProvaHaltIf >> bar
^ (self foo), 'bar'
```

A screenshot showing two windows. The top window is a "Playground" with the code "ProvaHaltIf new bar" entered. The bottom window is a "Halt" dialog box with tabs for "Proceed", "Abandon", "Debug", and "Report". The "Report" tab is selected, displaying a stack trace:

ProvaHaltif	foo
ProvaHaltif	bar
UndefinedObject	Dolt
OpalCompiler	evaluate
RubSmalltalkEditor	evaluate:andDo:
RubSmalltalkEditor	highlightEvaluateAn

Exemple: Recursivitat en blocs anònims

Com un bloc no referenciat per cap variable pot cridar-se ell mateix?



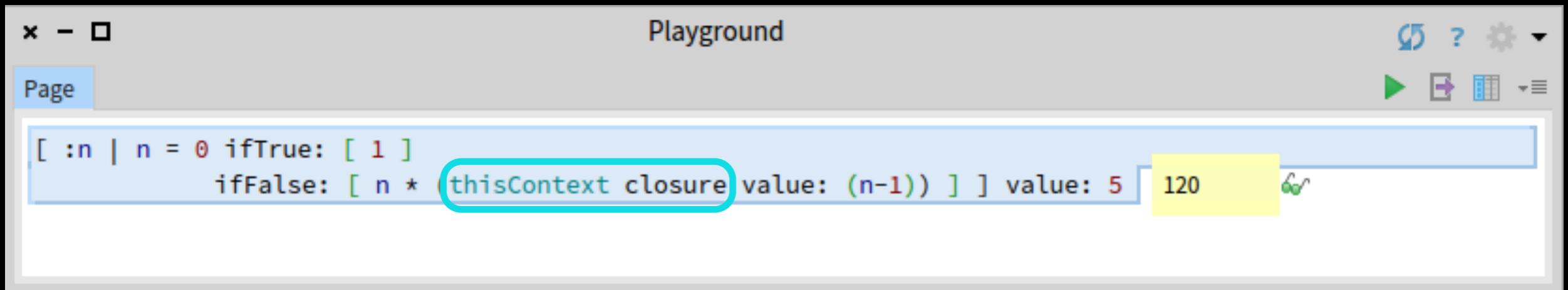
The screenshot shows a Smalltalk playground window titled "Playground". The toolbar includes icons for back, forward, and help. The code editor contains the following Smalltalk code:

```
[ :n | n = 0 ifTrue: [ 1 ]
           ifFalse: [ n * (thisContext closure value: (n-1)) ] ] value: 5
```

The result of the evaluation is displayed in a yellow box: 120. A green circular arrow icon is positioned next to the result.

Exemple: Recursivitat en blocs anònims

Com un bloc no referenciat per cap variable pot cridar-se ell mateix?



The screenshot shows a Smalltalk playground window titled "Playground". The code input field contains the following Smalltalk code:

```
[ :n | n = 0 ifTrue: [ 1 ]
           ifFalse: [ n * (thisContext closure value: (n-1)) ] ] value: 5
```

A cyan oval highlights the word "closure" in the line `thisContext closure value: (n-1)`. The result of the evaluation, "120", is displayed in a yellow box to the right of the code input field.

Exemple: Cal **self** si tinc **thisContext**?

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

- History Navigator**: A pane on the right displaying a history of objects. It shows a list of objects with their types and status. The object `provaReceiver` is highlighted in blue, indicating it is the current receiver. The status is listed as "as yet unclassified".
- Playground**: A pane at the bottom showing a transcript of interactions. The transcript shows the creation of a `ProvaReceiver` object and a message sent to it. The message is `ProvaReceiver new provaReceiver`, and the response is `true`.
- Variables**: A pane on the left showing the current scope. It lists variables and their types. The variable `receiver` is shown with the type `ProvaReceiver`.

The code in the playground pane is:

```
provaReceiver
^ (thisContext receiver == self)
```

Què es guarda en un **Context**?

The screenshot shows a Smalltalk environment with two windows:

- Playground Window:** Contains the following code:

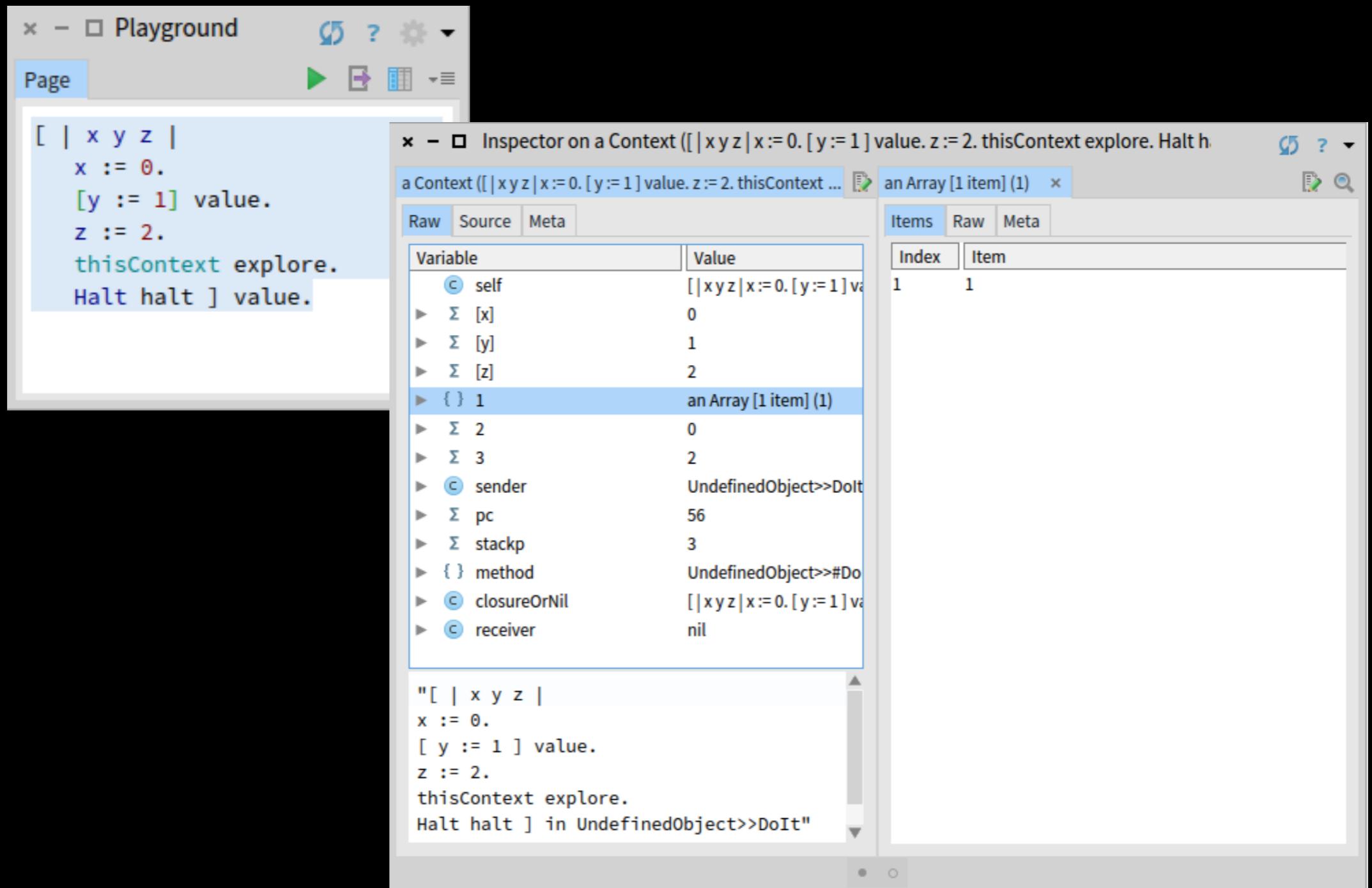
```
[ | x y z |
  x := 0.
  y := 1.
  z := 2.
  thisContext explore.
  Halt halt ] value.
```
- Inspector Window:** Titled "Inspector on a Context ([| xyz | x:=0. y:=1. z:=2. thisContext explore. Halt halt] in UndefinedObject>>DoIt)". It displays a table of variables and their values:

Variable	Value
self	[xyz x:=0.y:=1.z:=2.thisContext explore.Halt halt] in UndefinedObject>>DoIt
x	0
y	1
z	2
1	0
2	1
3	2
sender	UndefinedObject>>DoIt
pc	43
stackp	3
method	UndefinedObject>>#DoIt
closureOrNil	[xyz x:=0.y:=1.z:=2.thisContext explore.Halt halt]
receiver	nil

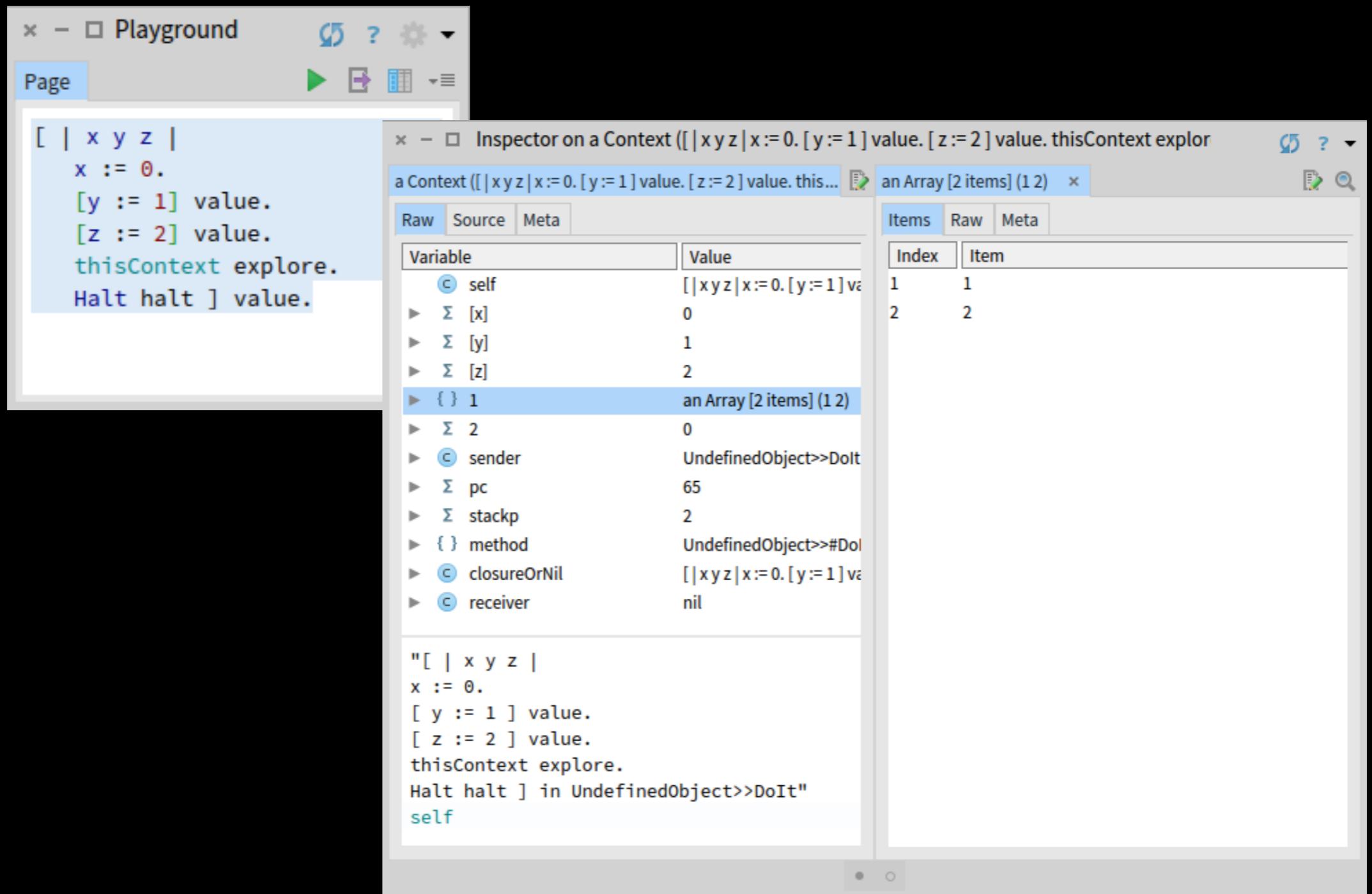
At the bottom of the Inspector window, the source code is shown again:

```
"[ | x y z |
  x := 0.
  y := 1.
  z := 2.
  thisContext explore.
  Halt halt ] in UndefinedObject>>DoIt"
self
```

Què es guarda en un Context?



Què es guarda en un Context?



Exemples: BlockWithExit & Binding

The screenshot shows the Smalltalk-80 environment with two windows open:

- Binding class>>#exampleSuccessful**: This window displays the following code:

```
exampleSuccessful
"Binding exampleSuccessful"

#testingDynamicBinding bindTo: 'This is a test' in:
[ Transcript show: (#testingDynamicBinding binding)asString;
```
- BlockWithExit class>>#example**: This window displays the following code:

```
example
"BlockWithExit example"

| theLoop coll |
"Transcript open."
coll := OrderedCollection new.
1000 timesRepeat: [ coll add: 1000 atRandom ].
theLoop := [coll do: [:each | Transcript show: each asString; cr.
(each < 100) ifTrue: [theLoop exit]]] withExit.
theLoop value.
```

Both windows have "History Navigator" tabs at the top, which show the execution history for each method. The "exampleSuccessful" history includes "exampleSuccessful" and "exampleUnsuccessful". The "example" history includes "example" and "with:".

Building Control Structures in the Smalltalk-80 System

L. Peter Deutsch

Byte, Vol.6 No.8 (Agost 1981), pp. 322-346

<https://archive.org/details/byte-magazine-1981-08/page/n335>

Exemple: Puc tenir un **BlockWithExit** anònim?

The screenshot shows a Smalltalk environment with two windows:

- Playground**: A code editor window containing the following Smalltalk code:

```
| theLoop coll |
coll := OrderedCollection new.
1000 timesRepeat: [ coll add: 1000 atRandom ].
theLoop := [coll do: [:each | each traceCr.
    (each < 100) ifTrue: [theLoop exit]]] withExit.
theLoop value.
```
- Transcript**: A log window showing the output of the executed code:

```
543
749
465
650
138
759
676
483
106
459
292
575
917
505
19
```

Building Control Structures in the Smalltalk-80 System

L. Peter Deutsch

Byte, Vol.6 No.8 (Agost 1981), pp. 322-346

<https://archive.org/details/byte-magazine-1981-08/page/n335>

Exemple: Puc tenir un **BlockWithExit** anònim?



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

```
| theLoop coll |
coll := OrderedCollection new.
1000 timesRepeat: [ coll add: 1000 atRandom ].
([coll do: [:each | each traceCr.
           (each < 100) ifTrue: [thisContext explore. Object halt "exit"]]] withExit) value
```

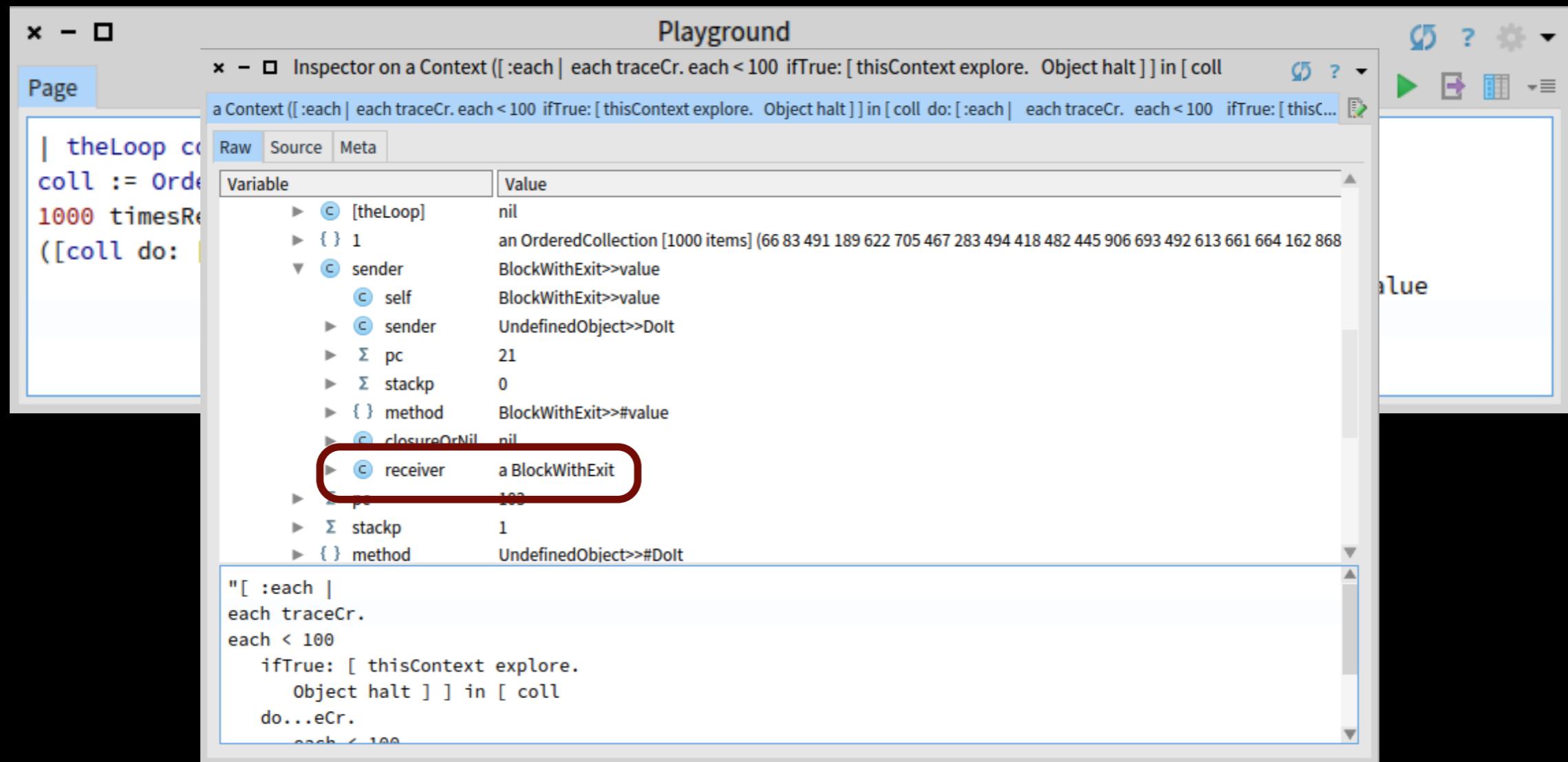
Building Control Structures in the Smalltalk-80 System

L. Peter Deutsch

Byte, Vol.6 No.8 (Agost 1981), pp. 322-346

<https://archive.org/details/byte-magazine-1981-08/page/n335>

Exemple: Puc tenir un **BlockWithExit** anònim?



Building Control Structures in the Smalltalk-80 System

L. Peter Deutsch

Byte, Vol.6 No.8 (Agost 1981), pp. 322-346

<https://archive.org/details/byte-magazine-1981-08/page/n335>

Exemple: Puc tenir un **BlockWithExit** anònim?

The screenshot shows a Smalltalk environment with two windows: a 'Playground' window and a 'Transcript' window.

In the 'Playground' window, the following code is displayed:

```
| theLoop coll |
coll := OrderedCollection new.
1000 timesRepeat: [ coll add: 1000 atRandom ].  
([coll do: [:each | each traceCr.  
        (each < 100) ifTrue: [thisContext sender sender sender receiver exit]]] withExit) value
```

The 'Transcript' window shows the following output:

```
109  
440  
585  
208  
941  
285  
34
```

Building Control Structures in the Smalltalk-80 System

L. Peter Deutsch

Byte, Vol.6 No.8 (Agost 1981), pp. 322-346

<https://archive.org/details/byte-magazine-1981-08/page/n335>

Introspecció:

- Inspeccionar objectes
- Consultar el codi
- Accedir els contexts d'execució

Intercessió:

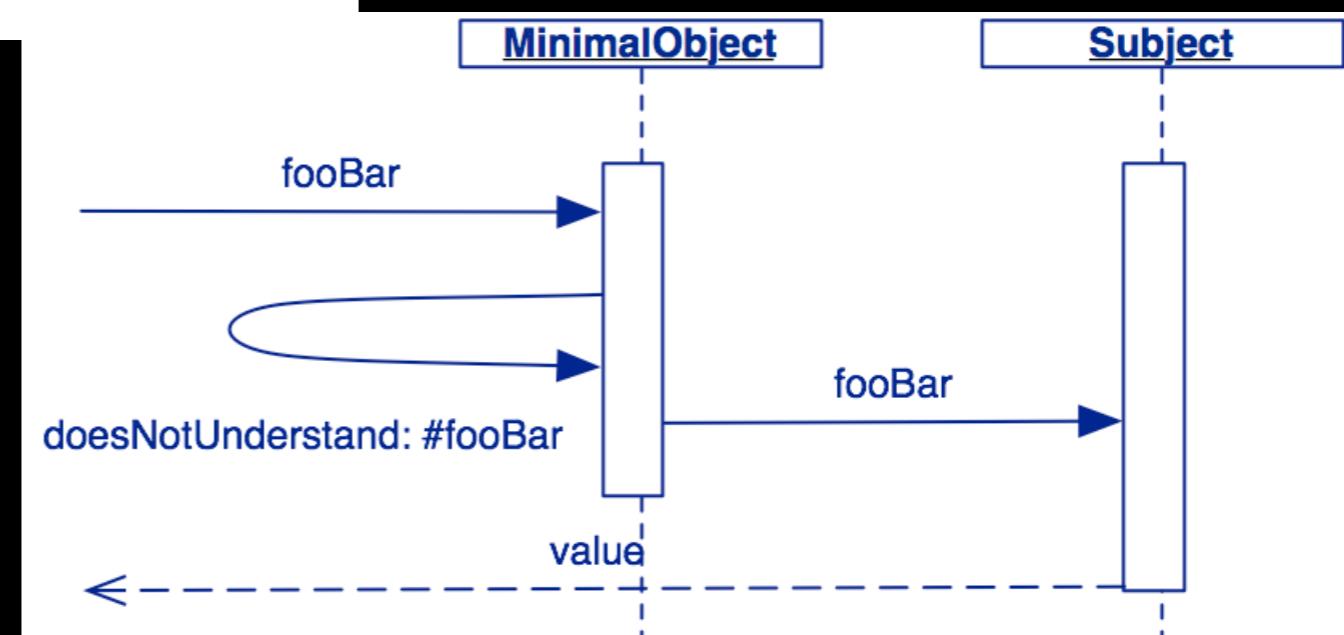
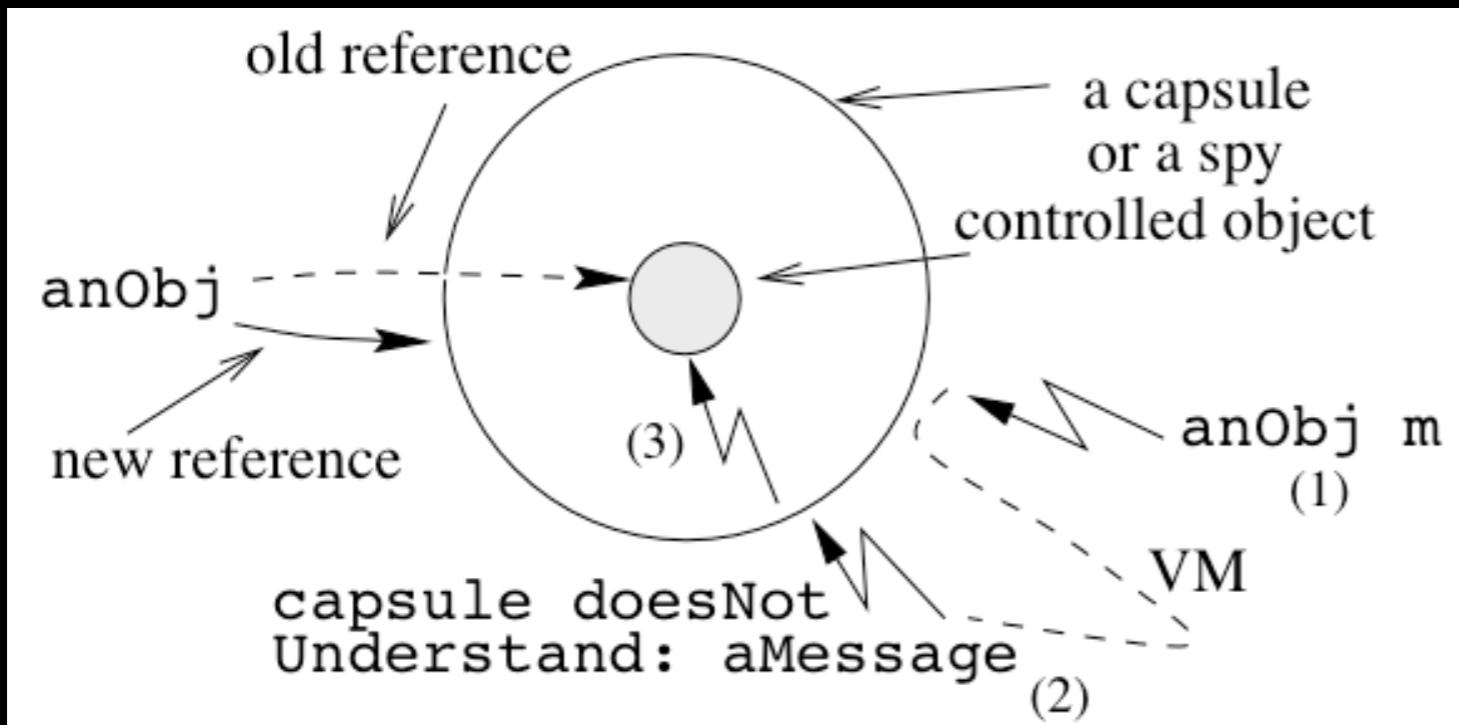
- **Sobreescriri `#doesNotUnderstand`:**
- Classes Anònimes
- Method Wrappers
- Continuacions

Sobreescrivir **#doesNotUnderstand:**

Introduirem un objecte mínim

- embolica (*wraps*) un objecte normal
- no enten (quasi bé) res
- redefineix **#doesNotUnderstand:**
- superclase és **nil** o **ProtoObject**
- utilitza **#become:** per substituir l'objecte a controlar

L'objecte mínim en acció



Logging d'enviaments de missatge utilitzant l'objecte mínim

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

- Code View:** A code editor window titled "LoggingProxy>>#initialize" containing the following code:

```
ProtoObject subclass: #LoggingProxy
instanceVariableNames: 'subject invocationCount'
classVariableNames: ''
package: 'Reflection-Complete'
```

Below this, the "initialize" method is defined:initialize
invocationCount := 0.
subject := self.The line "subject := self." is highlighted with a red oval.
- History Navigator:** A panel on the right showing a navigation history with items like "doesNotUnderstand:", "initialize", "invocationCount", and "initialize-release".
- Project Browser:** A left-hand pane showing a tree of packages and classes, with "Reflection-Complete" selected.
- Toolbars and Status:** Standard IDE toolbars and status bars at the bottom.

CAP: Reflexió en Smalltalk

Logging d'enviaments de missatge utilitzant l'objecte mínim

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

- LoggingProxy>>#doesNotUnderstand:** This window displays the implementation of the `#doesNotUnderstand:` message. The code is as follows:

```
doesNotUnderstand: aMessage
    Transcript show: 'performing ', aMessage printString.
    invocationCount := invocationCount + 1.
    ^ aMessage sendTo: subject
```

The line `^ aMessage sendTo: subject` is highlighted with a red oval.
- Message>>#sendTo:** This window shows the `#sendTo:` message definition. The code is:

```
sendTo: receiver
    "answer the result of sending this message to receiver"

    ^ receiver perform: selector withArguments: args
```

Both windows include a "History Navigator" pane on the right side, which lists various message categories and their implementations.

Utilitzar **#become:** per instal.lar un proxy

LoggingProxyTest>>#testDelegation

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

History Navigator

- DynamicAccessorsTest
- HaltDemo
- LoggingMethodWrapper
- LoggingProxy
- LoggingProxyTest
- MethodWrapperTest
- MinimalObject
- MinimalObjectTest
- ReflectionTest

-- all --
running

- testDelegation
- testSelf

testDelegation

```
| point |
point := 1@2.
LoggingProxy new become: point.
self assert: point invocationCount = 0.
self assert: point + (3@4) = (4@6).
self assert: point invocationCount = 1.
```

1/7 [1] Format as you read W +L

Com funciona això?

x - □ LoggingProxyTest>>#testDelegation

Scoped Variables

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

Refactoring-Tests-Critics
Refactoring-Tests-Environ
Reflection-Complete
ReflectionMirrors-Primitiv
ReflectionMirrors-Primitiv
Reflectivity
Reflectivity-Examples
Reflectivity-Tests
Reflectivity-Tools
Reflectivity-Tools-Tests

DynamicAccessorsTest
HaltDemo
LoggingMethodWrapper
LoggingProxy
LoggingProxyTest
MethodWrapperTest
MinimalObject
MinimalObjectTest
ReflectionTest

-- all --
running

testDelegation
testSelf

History Navigator

testDelegation

```
| point |
point := 1@2.
LoggingProxy new become: point.
self assert: point invocationCount = 0.
self assert: point + (3@4) = (4@6).
self assert: point invocationCount = 1.
```

1/7 [1] Format as you read W +L

Limitacions

- El problema de **self**

*Missatges enviats per l'objecte a ell mateix
no són interceptats*

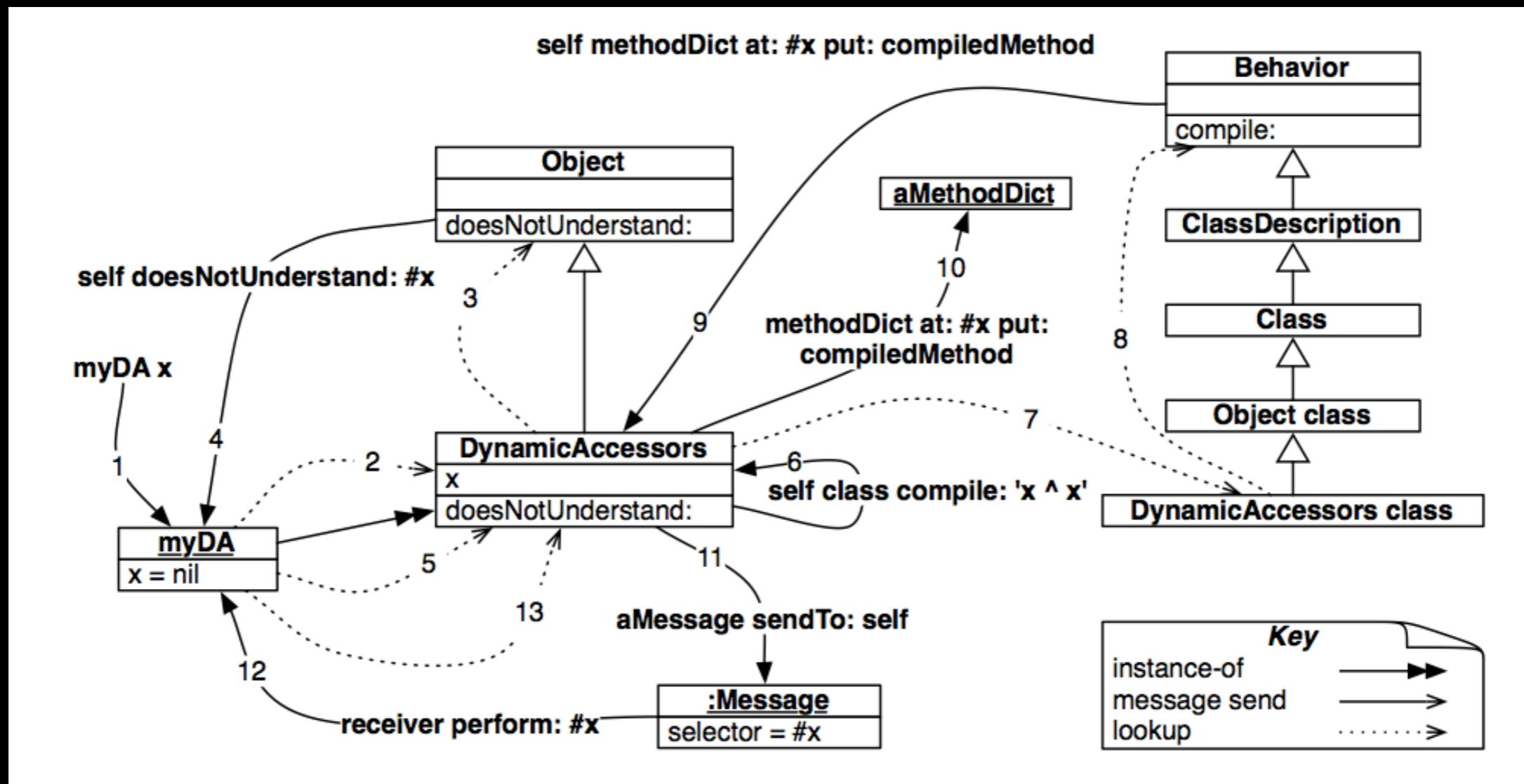
- El control de les classes és impossible

No puc fer swap de classes

- Interpretació del protocol mínim

*Què fem amb els missatges que poden ser enviats
tant a l'objecte embolicat com a l'objecte mínim?*

Redefinir **#doesNotUnderstand:** per generar codi dinàmicament:
Exemple: Els *getters* “*on demand*”



Redefinir **#doesNotUnderstand:** per generar codi dinàmicament:
Exemple: Els *getters* “*on demand*”

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

- Title Bar:** DynamicAccessors>>#doesNotUnderstand:
- Toolbars:** History Navigator, with buttons for back, forward, and search.
- Left Panel (Type Browser):** Shows the current type is Pkg1|^Pkg2|Pk.*Core\$. A list of packages is shown, with "Reflection-Complete" selected. Other packages include Refactoring-Tests-Critics, Refactoring-Tests-Enviro, ReflectionMrors-Primitiv, Reflectivity, Reflectivity-Examples, Reflectivity-Tools, and Reflectivity-Tools-Tests.
- Middle Panel (Code Editor):** Displays the source code for the #doesNotUnderstand: method in DynamicAccessors. The code is as follows:

```
doesNotUnderstand: aMessage
| messageName |
messageName := aMessage selector asString.
(self class instVarNames includes: messageName)
ifTrue: [self class compile: messageName , String cr , ' ^ ', messageName.
^ aMessage sendTo: self].
super doesNotUnderstand: aMessage
```

A yellow callout box points to the "doesNotUnderstand:" message in the History Navigator and contains the text: "Un objecte mínim es pot utilitzar per generar codi dinàmicament o carregar codi de manera *lazy*".

Bottom Status Bar: 1/7 [1] Format as you read W +L

Introspecció:

- Inspeccionar objectes
- Consultar el codi
- Accedir els contexts d'execució

Intercessió:

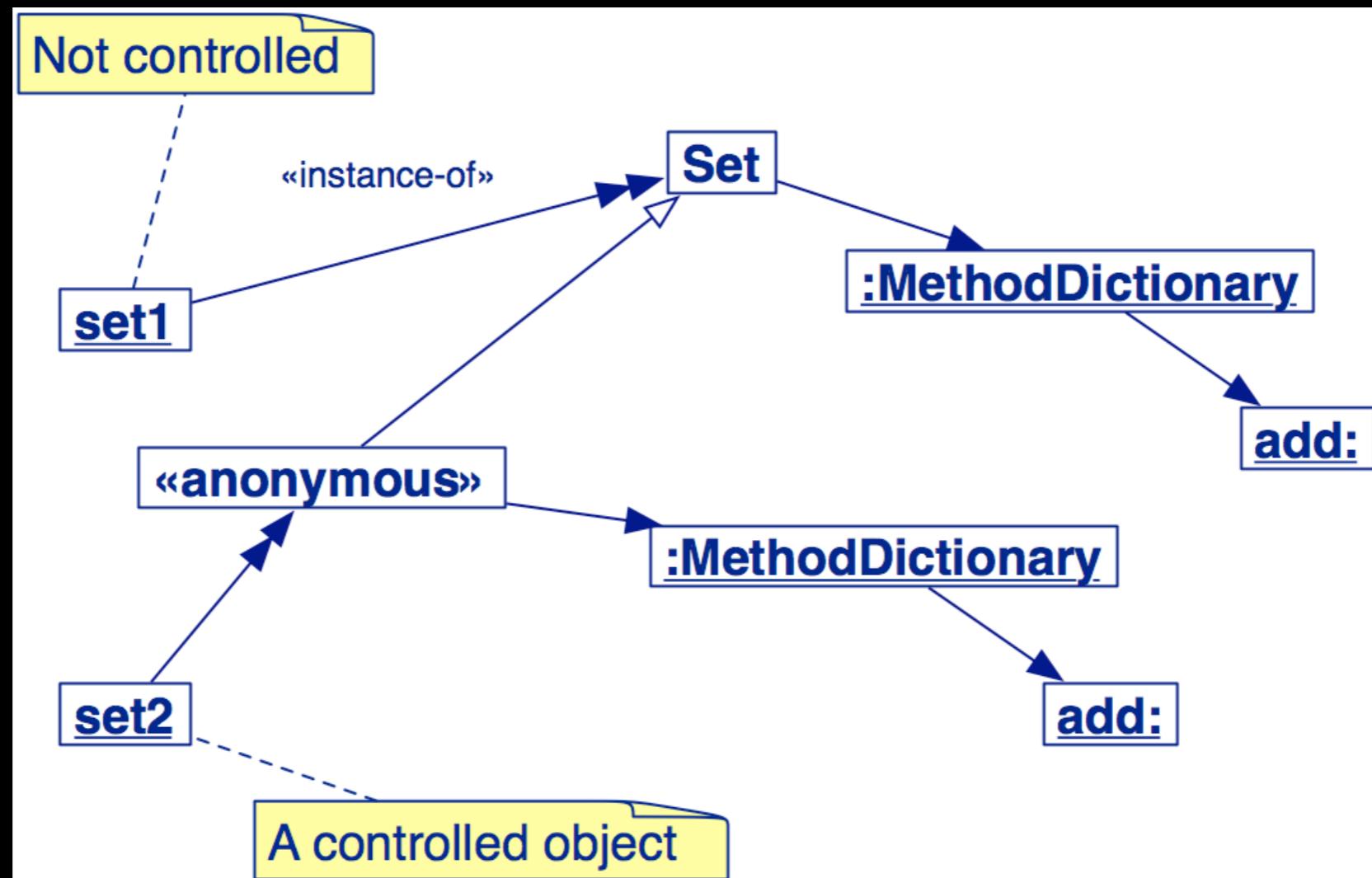
- Sobreescrivir **#doesNotUnderstand:**
- **Classes Anònimes**
- Method Wrappers
- Continuacions

Control de missatges amb classes anònimes

Crear una classe *anònim*a:

- Instància de **Behavior**
- Definir mètodes de control
- Interposar-la entre la instància i la classe

Control sel.selectiu



Classes anònimes a Pharo

The screenshot shows the Pharo IDE interface with two main windows: 'Playground' and 'Transcript'.

Playground Window:

```
| set anonClass |
set := Set new.
anonClass := Behavior new.
anonClass superclass: Set;
    setFormat: Set format;
    methodDictionary: MethodDictionary new.

anonClass compile:
    'add: anObject
        Transcript show: ''adding ''', anObject printString; cr.
        ^ super add: anObject'.

set add: 1.
set primitiveChangeClassTo: anonClass new.
set add: 2.
('set size = ' , (set size) asString) traceCr.
```

A red oval highlights the line `anonClass := Behavior new.`

A green oval highlights the line `anonClass new.`

Transcript Window:

```
adding 2
'set size = 2'
```

Avaluació

- Control sel.lectiu
- El problema de **self** no hi és
- Eficient
- Transparent a l'usuari

Introspecció:

- Inspeccionar objectes
- Consultar el codi
- Accedir els contexts d'execució

Intercessió:

- Sobreescrivir **#doesNotUnderstand:**
- Classes Anònimes
- **Method Wrappers**
- Continuacions

Substitució de mètodes

Primera aproximació:

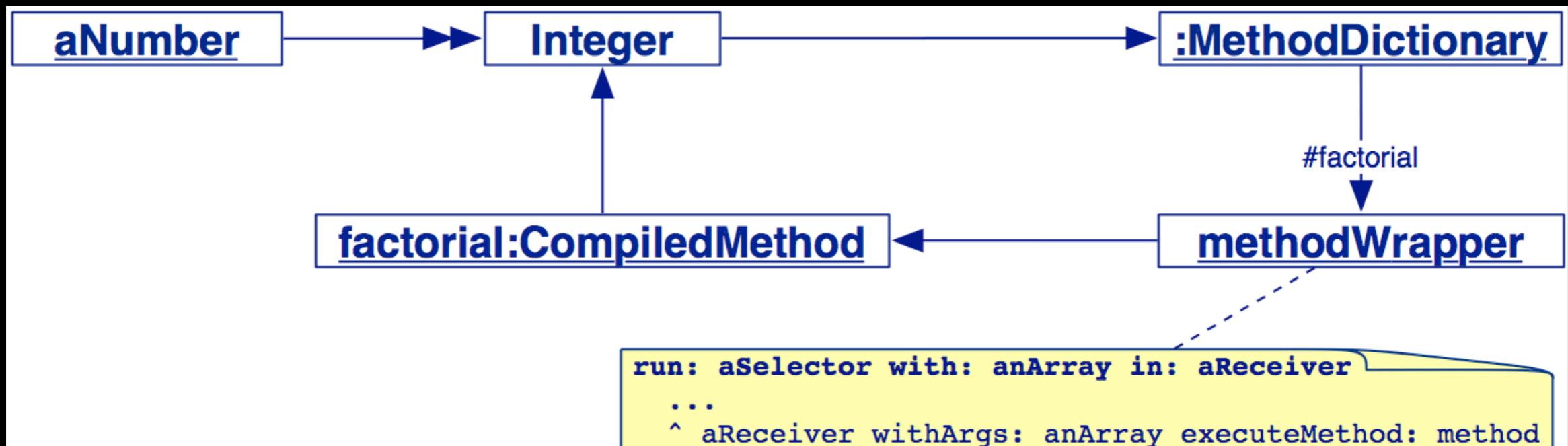
- Afegir mètodes amb *mangled names*
L'usuari, però, els pot veure.

Segona aproximació:

- Embolicar els mètodes sense afegir res a l'interfície
Substítueix el mètode per un objecte que implementi #run:with:in:

Embolicar mètodes per fer accions abans i després

Un methodWrapper substitueix l'original CompiledMethod en el diccionari de mètodes d'una classe i l'embolica realitzant algunes accions “before” i “after”, per exemple



LoggingMethodWrapper

```
LoggingMethodWrapper>>initializeOn: aCompiledMethod
    method := aCompiledMethod.
    reference := aCompiledMethod methodReference.
    invocationCount := 0
```

```
LoggingMethodWrapper>>install
    reference actualClass methodDictionary
        at: reference selector
        put: self
```

uninstall és similar ...

```
LoggingMethodWrapper>>run: aSelector with: anArray in: aReceiver
    invocationCount := invocationCount + 1.
    ^ aReceiver withArgs: anArray executeMethod: method
```

Instalar un **LoggingMethodWrapper**

```
logger := LoggingMethodWrapper on: Integer>>#factorial.
```

```
logger invocationCount. 0  
5 factorial.  
logger invocationCount. 0
```

```
logger install.  
[ 5 factorial ] ensure: [ logger uninstall ].  
logger invocationCount. 6
```

```
10 factorial.  
logger invocationCount. 6
```

Avaluació

- Basat en classes
totes les instàncies queden controlades
- Només s'intercepten els missatges coneguts
- Un sol mètode pot ser controlat
- No cal compilar per instal.lar

Introspecció:

- Inspeccionar objectes
- Consultar el codi
- Accedir els contexts d'execució

Intercessió:

- Sobreescrivir **#doesNotUnderstand**:
- Classes Anònimes
- Method Wrappers
- **Continuacions**

Continuation: abstract representation of the control state of a computer program. A continuation reifies the program control state, i.e. the continuation is a data structure that represents the computational process at a given point in the process's execution; the created data structure can be accessed by the programming language, instead of being hidden in the runtime environment.

<http://en.wikipedia.org/wiki/Continuation>

Per a aquesta part del curs teniu un document al Racó: **Continuaciones.pdf**

En aquest document trobareu explicacions molt més detallades del que segueix. A més, recomano ***molt*** que us llegiu el capítol 14, *Blocks: A Detailed Analysis*, del llibre ***Deep into Pharo***

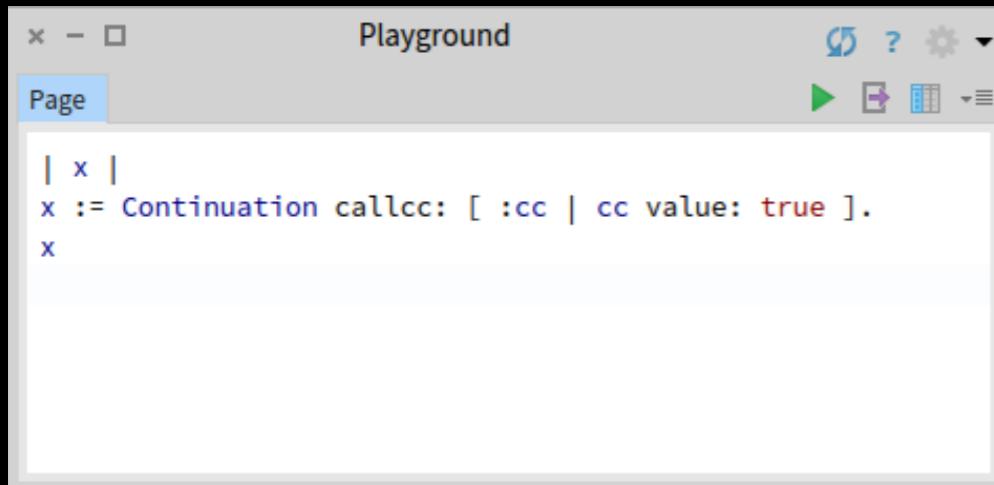
(<http://files.pharo.org/books-pdfs/deep-into-pharo/2013-DeepIntoPharo-EN.pdf>)

En el capítol esmentat es fan referències a les classes **MethodContext** i **ContextPart**. **MethodContext** era la classe de la que eren instàncies els *stack frames*. **MethodContext** era subclasse de **ContextPart**, i aquesta era subclasse de **InstructionStream**. En Pharo 6.1 la classe **Context** és una fusió de **MethodContext** i **ContextPart**, substituint a aquestes dues classes. Aleshores, és **Context** la que és subclasse d'**InstructionStream**.

Tingueu-ho en compte en estudiar el capítol *Blocks: A Detailed Analysis*

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

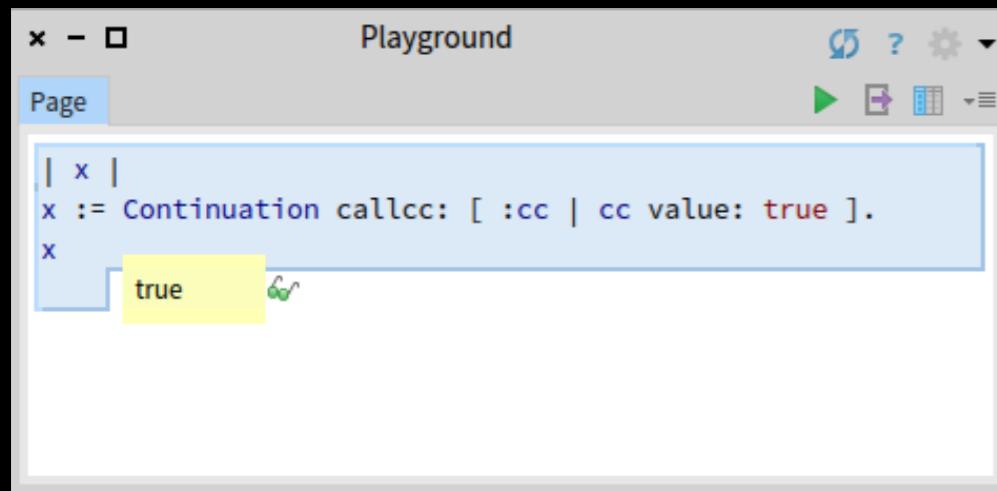
Exemples (senzills): Quin és el resultat d'avaluar aquests codis?



```
| x |
x := Continuation callcc: [ :cc | cc value: true ].
x
```

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

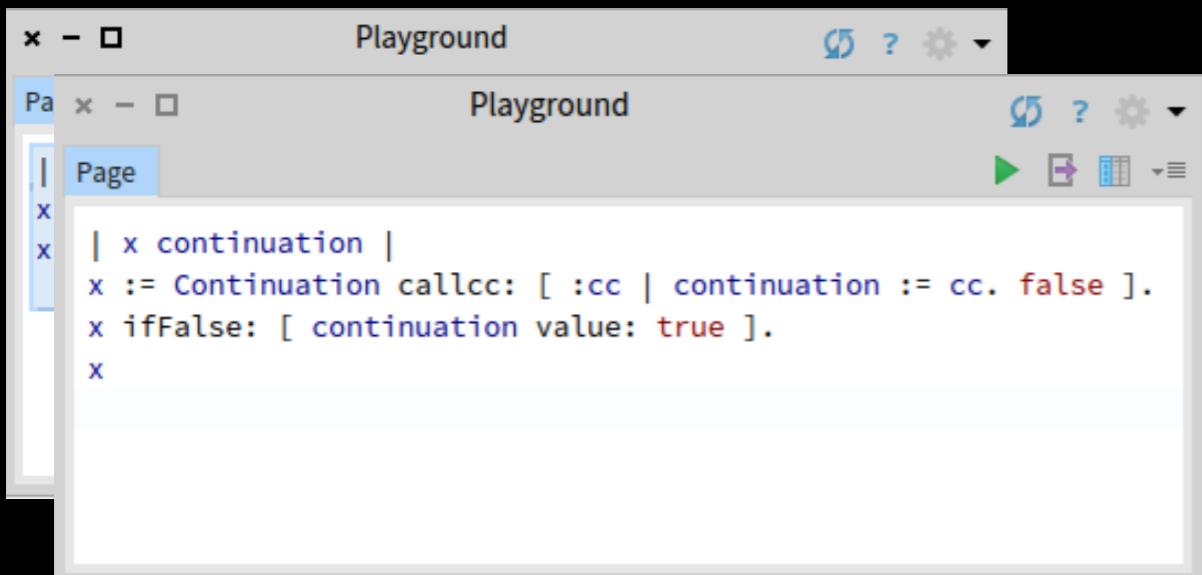
Exemples (senzills): Quin és el resultat d'avaluar aquests codis?



```
| x |
x := Continuation callcc: [ :cc | cc value: true ].
x
```

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

Exemples (senzills): Quin és el resultat d'avaluar aquests codis?



The screenshot shows two overlapping windows of a Smalltalk playground. Both windows have a title bar labeled "Playground". The left window has a sidebar on the left with tabs for "Page" (which is selected) and "x". The main area contains the following Smalltalk code:

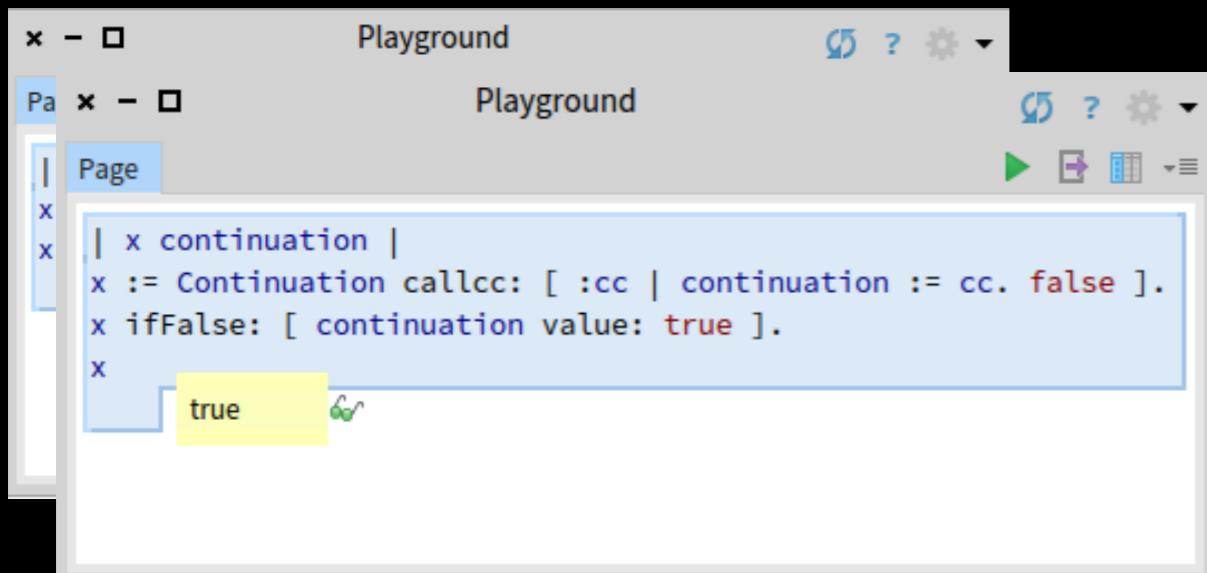
```
| x continuation |
x := Continuation callcc: [ :cc | continuation := cc. false ].
x ifFalse: [ continuation value: true ].
```

The right window also has a title bar labeled "Playground" and a sidebar with tabs for "Page" and "x". It contains a single line of code:

```
x
```

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

Exemples (senzills): Quin és el resultat d'avaluar aquests codis?



The image shows two overlapping Smalltalk playground windows. Both windows have a title bar labeled "Playground". The top window has a toolbar with icons for help, settings, and execution. The bottom window also has a toolbar with similar icons. In the bottom window's code editor, there is a continuation-based code snippet:

```
| x continuation |
x := Continuation callcc: [ :cc | continuation := cc. false ].
x ifFalse: [ continuation value: true ].
```

The word "true" is highlighted with a yellow background in the code editor. The bottom window's status bar shows the text "true" and a small icon.

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

Exemples (senzills): Quin és el resultat d'avaluar aquests codis?

```
| x cc |
x := 0.
x ifFalse: [ assoc key value: assoc ].
assoc := Continuation callcc: [ :cc | cc -> 0 ].
assoc value: assoc value + 1.
self assert: assoc value ~= 5.
assoc value = 4
ifFalse: [ assoc key value: assoc ].
assoc value.
```

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

Exemples (senzills): Quin és el resultat d'avaluar aquests codis?

```
| x cc |
x := 0.
x ifFalse: [ assoc | 
assoc := Continuation callcc: [ :cc | cc -> 0 ].
assoc value: assoc value + 1.
self assert: assoc value ~= 5.
assoc value = 4
    ifFalse: [ assoc key value: assoc ].
assoc value.
```

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

Exemples (senzills): Quin és el resultat d'avaluar aquests codis?

The screenshot shows a stack of four Smalltalk windows, each titled "Playground". The windows are arranged vertically, with the top one being the largest and the others becoming progressively smaller as they descend. Each window contains a text area with Smalltalk code. The code is a continuation-passing style (CPS) transformation of a traditional imperative loop. It involves several temporary variables and continuations:

```
| tmp tmp2 y |
 #(1 2 3) do: [ :i |
    | x |
    x := i.
    tmp ifNil: [ tmp2 := (Continuation callcc: [ :cc | tmp := cc. [ :q | ] ]) ].  
    tmp2 value: x.  
    x := 17 ].  
y := (Continuation callcc: [ :cc | tmp value: cc. 42 ]).  
y
```

Amb aquesta infraestructura ja podem guardar i recuperar qualsevol moment de l'execució d'un programa (d'un *thread* en realitat, però no entrarem en aquests detalls).

Exemples (senzills): Quin és el resultat d'avaluar aquests codis?

The screenshot shows a desktop environment with four Smalltalk playground windows open. The bottom-right window is the active one, displaying the following code:

```
| tmp tmp2 y |
 #(1 2 3) do: [ :i |
 | x |
 x := i.
 tmp ifNil: [ tmp2 := (Continuation callcc: [ :cc | tmp := cc. [ :q | ] ]) ].  
tmp2 value: x.  
x := 17 ].  
y := (Continuation callcc: [ :cc | tmp value: cc. 42 ]).  
y
```

The code uses continuation passing style (CPS) to implement a delayed assignment for variable 'x'. It creates a continuation for each value in the sequence (1, 2, 3), setting 'x' to the current value and then calling the continuation. Finally, it creates another continuation to return the value of 'x' (which is now 17) and set 'y' to that value.

Exemple: Implementar **BlockClosure>>#whileTrueCC:**, l'equivalent del **#whileTrue:**, sense cap construcció iterativa, només fent servir **#callcc**:

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

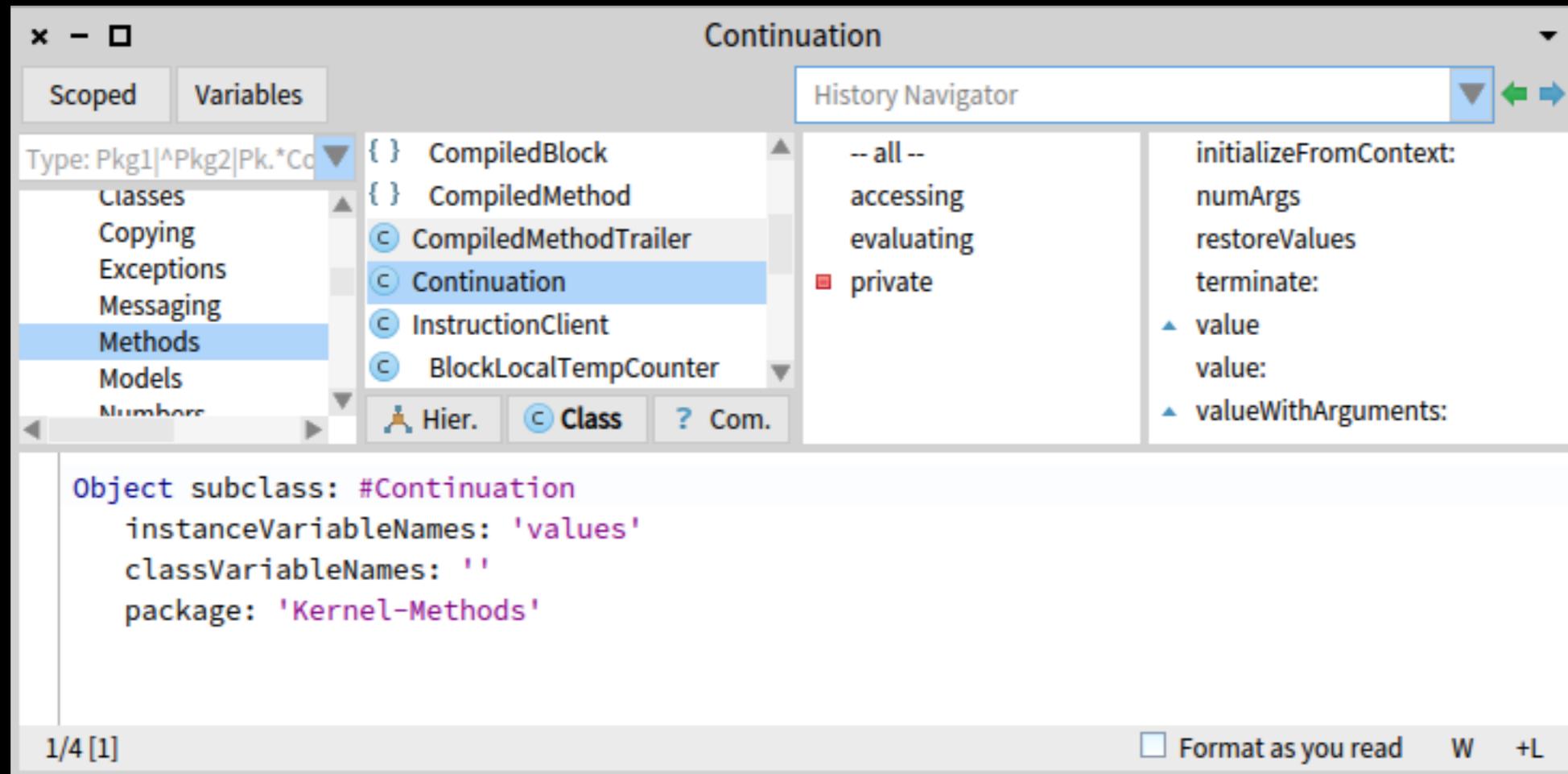
- Top Bar:** Shows the class name **BlockClosure>>#whileTrueCC:**.
- History Navigator:** A sidebar on the right showing a list of loaded packages: *metacello-core, *metacello-core-scripting, *metacello-mc, *OpalCompiler-Core, *Reflection-Complete, *Reflectivity, *Slot, and *SortFunctions-Core-control. The package ***Reflection-Complete** is highlighted.
- Scope Browser:** On the left, it shows the current type is **Pkg1|^APkg2|Pk.*Core\$**. It lists several packages under Refactoring-Tests and Reflection-Complete, with **Reflection-Complete** currently selected. Below this is a list of classes: MethodWrapperTest, MinimalObject, MinimalObjectTest, ReflectionTest, BlockClosure (which is also selected), and Continuation.
- Code Editor:** The main area contains the implementation of the **whileTrueCC:** method. The code is as follows:

```
whileTrueCC: aBlock
    "versió de whileTrue: implementada amb callcc"

    | cont |
    cont := Continuation callcc: [ :cc | cc ].
    self value
        ifTrue: [ aBlock value.
            cont value: cont ]
        ifFalse: [ ^ nil ]
```

At the bottom of the code editor, there are status bars for page number (9/9 [22]), reading mode (Format as you read), and window controls (W +L).

A Pharo 6.1 existeix la classe **Continuation**. Serveix per guardar el context, és a dir, la pila d'execució



S'instancia amb **Continuation class>>#current**
o amb **Continuation class>>#currentDo:**

The image shows two overlapping Smalltalk environments. The top environment is titled "Continuation class>>#current". Its left pane lists "Methods" under "Type: Pkg1|^Pkg2|Pk.*". The bottom pane displays the source code for the "current" method:

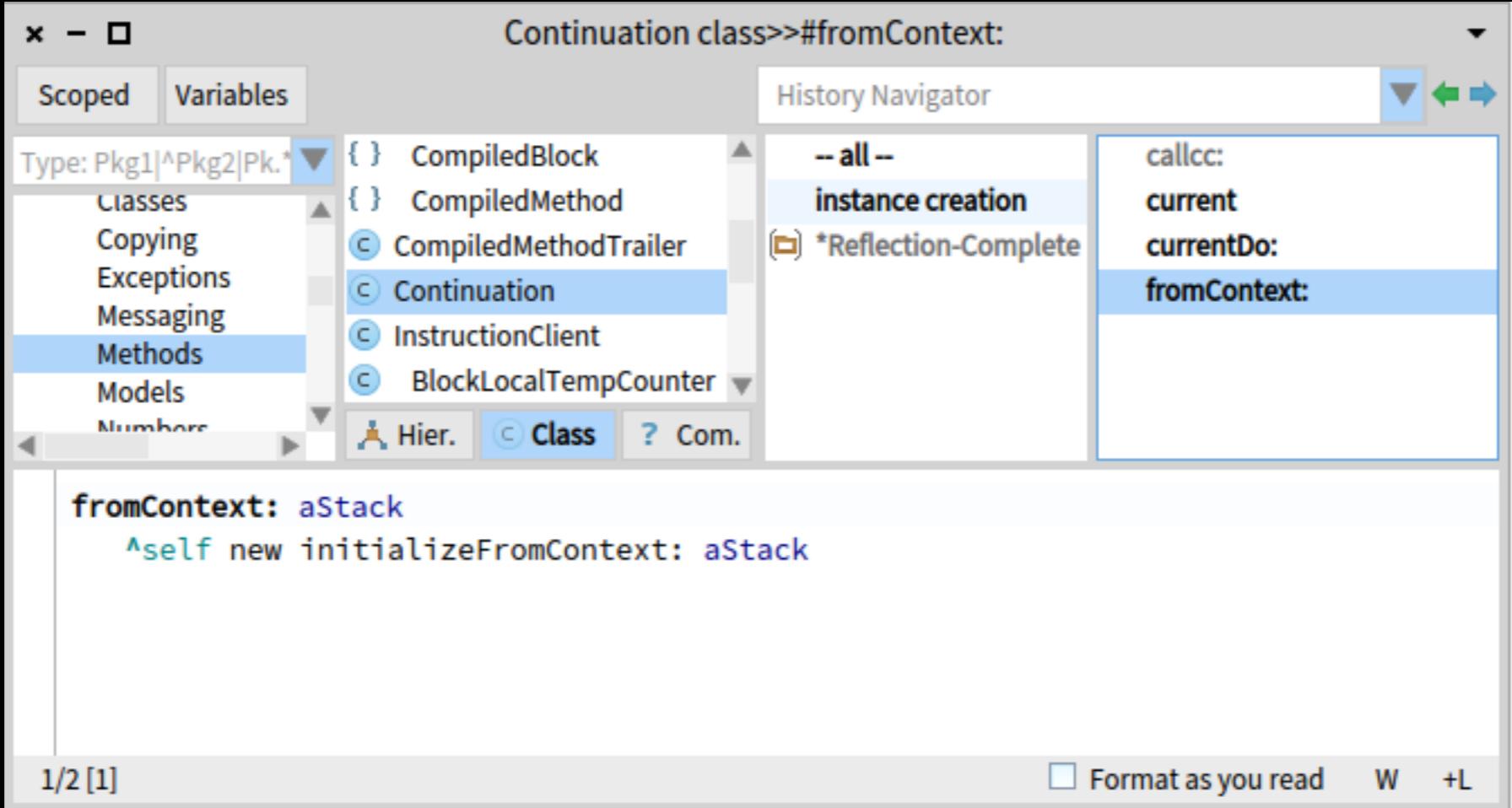
```
current
  ^ self fromContext: thisContext sender
```

The bottom environment is titled "Continuation class>>#currentDo:". Its left pane also lists "Methods". The bottom pane displays the source code for the "currentDo:" method:

```
currentDo: aBlock
  ^ aBlock value: (self fromContext: thisContext sender)
```

Both environments feature a "History Navigator" at the top right, which includes buttons for navigating between methods and a list of recent history items: "callcc:", "current", "currentDo:", and "fromContext:".

Així doncs, és clar que el mètode que crea la instància és **Continuation class>>#fromContext:**



The screenshot shows a Smalltalk development environment window titled "Continuation class>>#fromContext:". The interface includes a top navigation bar with tabs for "Scoped" and "Variables". Below this is a "History Navigator" pane containing a list of recent items: "-- all --", "instance creation", and "*Reflection-Complete". To the right of the navigator is a list of objects under "Type: Pkg1|^Pkg2|Pk.*": CompiledBlock, CompiledMethod, CompiledMethodTrailer, Continuation (which is selected), InstructionClient, and BlockLocalTempCounter. At the bottom of the window, the code for the "fromContext:" method is displayed:

```
fromContext: aStack
^self new initializeFromContext: aStack
```

The status bar at the bottom indicates "1/2 [1]" and includes buttons for "Format as you read", "W", and "+L".

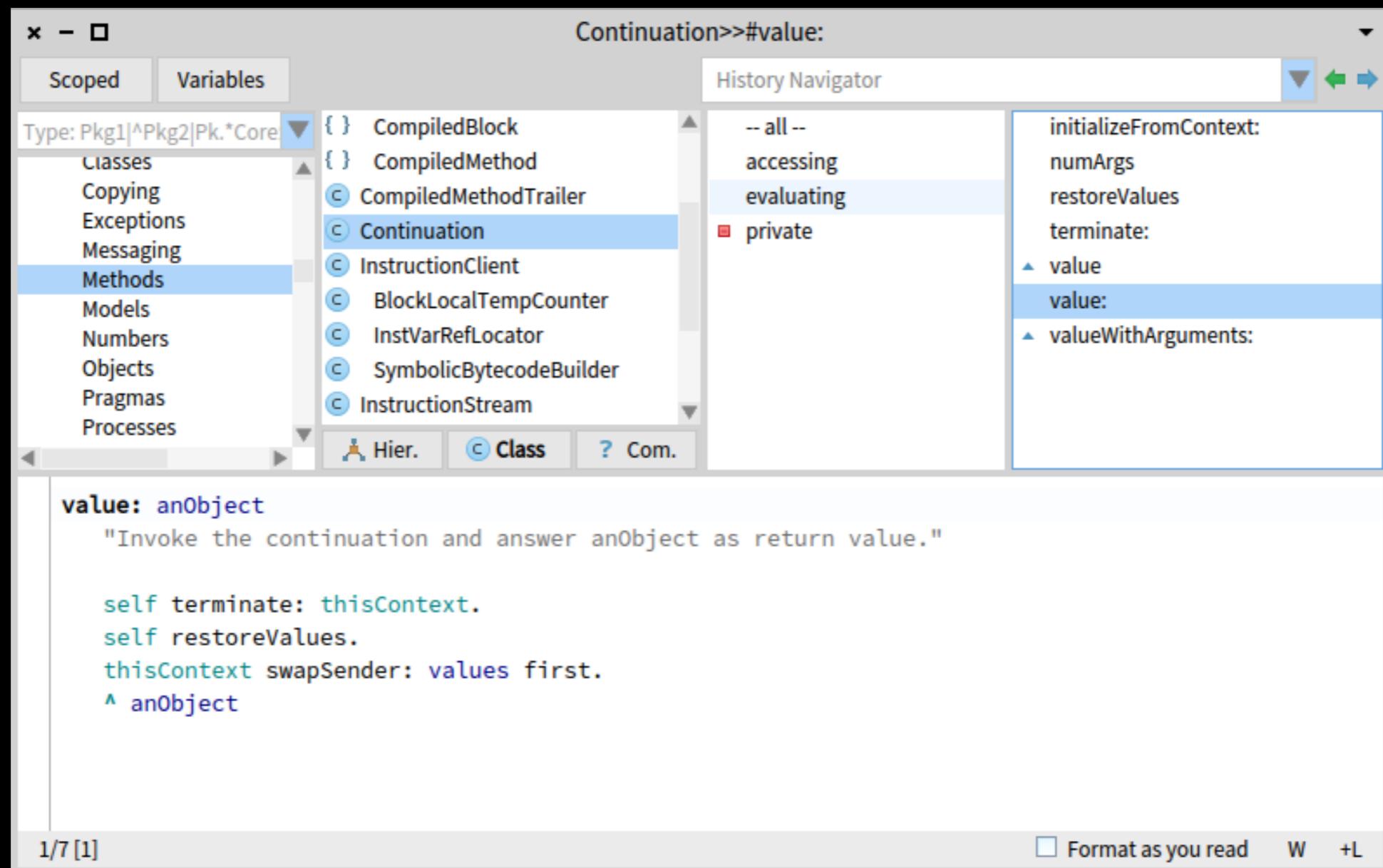
CAP: Reflexió en Smalltalk

Cal guardar tota la informació que tenim a la pila d'execució, però considerant que Pharo 6.1 internament recicla les instàncies de **Context**, el que cal guardar va més enllà dels elements enllaçats des del context argument **aContext**

The screenshot shows the Pharo Smalltalk IDE interface. The title bar reads "Continuation>>#initializeFromContext:". The left pane is a "History Navigator" with tabs for "Scoped" and "Variables". The "Variables" tab is selected, showing a list of objects: CompiledBlock, CompiledMethod, CompiledMethodTrailer, Continuation (which is highlighted), InstructionClient, BlockLocalTempCounter, InstVarRefLocator, SymbolicBytecodeBuilder, and InstructionStream. Below this is a "Type" dropdown set to "Pkg1|^Pkg2|Pk.*Core" and a "Methods" list containing: Classes, Copying, Exceptions, Messaging, Methods (selected), Models, Numbers, Objects, Pragmas, and Processes. At the bottom of the left pane are buttons for "Hier.", "Class" (selected), and "? Com.". The right pane contains the source code for the `#initializeFromContext:` method of the `Continuation` class:initializeFromContext: aContext
| valueStream context |
valueStream := WriteStream on: (Array new: 20).
context := aContext.
[context notNil] whileTrue:
[valueStream nextPut: context.
1 to: context class instSize do: [:i | valueStream nextPut: (context instVarAt: i)].
1 to: context size do: [:i | valueStream nextPut: (context at: i)].
context := context sender].
values := valueStream contents

The bottom status bar shows "1/10 [1]" on the left and "Format as you read W +L" on the right.

Un cop la pila d'execució està guardada a la variable d'instància **values** de la instància de **Continuation**, cal poder reiniciar-la quan desitjem fer-ho. Per a això tenim **Continuation>>#value**:



CAP: Reflexió en Smalltalk

Continuation>>#terminate:

Type: Pkg1|^Pkg2|Pk.*Core

Scoped Variables

History Navigator

Methods

Continuation

initializeFromContext:
numArgs
restoreValues
terminate:
value
value:
valueWithArguments:

terminate: aContext

```
| context |
context := aContext.
[context notNil] whileTrue: [context := context swapSender: nil]
```

1/4 [1]

Context de #terminate:

closureOrNil
receiver
method
stackp
pc
sender
aContext

Context de #value:

closureOrNil
receiver
method
stackp
pc
sender

<aContext>

closureOrNil
receiver
method
stackp
pc
sender

<aContext>

closureOrNil
receiver
method
stackp
pc
sender

```
graph LR
    C1[closureOrNil  
receiver  
method  
stackp  
pc  
sender  
aContext] --> C2[closureOrNil  
receiver  
method  
stackp  
pc  
sender]
    C2 --> C3[closureOrNil  
receiver  
method  
stackp  
pc  
sender]
    C3 --> C4[closureOrNil  
receiver  
method  
stackp  
pc  
sender]
    C4 -.-> C1
```

CAP: Reflexió en Smalltalk

Continuation>>#restoreValues

History Navigator

Type: Pkg1|^Pkg2|Pk.*Core

Variables

Scoped

Methods

CompiledBlock
CompiledMethod
CompiledMethodTrailer
Continuation
InstructionClient
BlockLocalTempCounter
InstVarRefLocator
SymbolicBytecodeBuilder

-- all --
accessing
evaluating
private

initializeFromContext:
numArgs
restoreValues
terminate:
value
value:
valueWithArguments:

restoreValues

```
| valueStream context |
valueStream := values readStream.
[valueStream atEnd] whileFalse:
[context := valueStream next.
1 to: context class instSize do: [:i | context instVarAt: i put: valueStream next].
1 to: context size do: [:i | context at: i put: valueStream next]]
```

1/7 [1]

Context d'#restoreValues

closureOrNil
receiver
method
stackp
pc
sender

Context de #value:

closureOrNil
receiver
method
stackp
pc
sender

<aContext>

closureOrNil
receiver
method
stackp
pc
sender

<aContext>

closureOrNil
receiver
method
stackp
pc
sender

values

The diagram illustrates the continuation flow in Smalltalk. It shows four stack frames representing contexts. The first frame is labeled 'Context d'#restoreValues' and contains variables: closureOrNil, receiver, method, stackp, pc, and sender. An arrow points from this frame to the second frame, labeled 'Context de #value:', which also contains closureOrNil, receiver, method, stackp, pc, and sender. This pattern repeats for two more frames labeled '<aContext>' before ending with an ellipsis. A variable named 'values' is shown at the bottom, with an arrow pointing to the 'closureOrNil' slot of the first context frame. The code in the 'restoreValues' method shows it reads values from a stream and writes them back into the 'values' variable.

CAP: Reflexió en Smalltalk

Continuation>>#value:

History Navigator

Type: Pkg1|^Pkg2|Pk.*Core

Scoped Variables

Methods

Continuation

value: anObject
"Invoke the continuation and answer anObject as return value."
self terminate: thisContext.
self restoreValues.
thisContext swapSender: values first.
^ anObject

1/7 [1]

Context de #value:
closureOrNil
receiver
method
stackp
pc
sender

<aContext>
closureOrNil
receiver
method
stackp
pc
sender

<aContext>
closureOrNil
receiver
method
stackp
pc
sender

values

initializeFromContext:
numArgs
restoreValues
terminate:
value
value:
valueWithArguments:

Finalment, volem crear instàncies de **Continuation** de manera controlada. Afegirem un mètode a **Continuation class** anomenat **#callcc:** que espera un bloc d'un paràmetre [:k | . . .] com a argument

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

- Title Bar:** Continuation class>#callcc:
- Toolbars:** History Navigator (with buttons for back, forward, and search).
- Left Panel (Type Selector):** Shows categories: Classes, Copying, Exceptions, Messaging, Methods (selected), Models, Numbers, Objects.
- Middle Panel (Object Browser):** Shows objects: CompiledBlock, CompiledMethod, CompiledMethodTrailer, Continuation (selected), InstructionClient, BlockLocalTempCounter, InstVarRefLocator.
- Right Panel (History Navigator):** Shows history entries: -- all --, instance creation, *Reflection-Complete, callcc: (highlighted), current, currentDo:, fromContext:.
- Bottom Panel (Code Editor):** Displays the source code for #callcc: method:

```
callcc: aBlock
  ^ self currentDo: aBlock
```
- Status Bar:** 1/2 [1] Format as you read W +L

Exemple: Interacció entre Closures i Continuacions

(recordeu allò que explicàvem sobre què es guarda en un Context?)

The screenshot shows a Smalltalk playground window with the title "Playground". The code area contains the following Smalltalk code:

```
| x y c |
y := 3.
x := Continuation callcc: [ :k | c := k. false ].
y := y + 5.
x ifFalse: [ c value: true ].
{ x . y }.
```

The output pane at the bottom shows the result of running the code: `#(true 8)`. The number 8 is highlighted with a yellow background.

Exemple: Interacció entre Closures i Continuacions

(recordeu allò que explicàvem sobre què es guarda en un Context?)

The image shows two side-by-side Smalltalk playground windows. Both windows have a title bar labeled "Playground" and a toolbar with standard icons.

Left Window:

```
| x y c |
y := 3.
x := Continuation callcc: [ :k | c := k. false ].
y := y + 5.
x ifFalse: [ c value: true ].
{ x . y }.
```

Output:

```
#(true 8)
```

Right Window:

```
| x y c |
[y := 3.
x := Continuation callcc: [ :k | c := k. false ].
y := y + 5.
x ifFalse: [ c value: true ].
{ x . y } ] value
```

Output:

```
#(true 13)
```

In both windows, the code is identical up to the final closing brace of the block. In the left window, the output is a list containing the value of 'c' (true) and the result of the block (8). In the right window, the output is a list containing the value of 'c' (true) and the result of the block (13), which is the sum of 3 and 5. This demonstrates that the continuation captures the state of the variable 'y' at the time it was created, even though it is modified later in the code.

Finalment, les conseqüències de tenir continuacions són força subtils...

Playground

Page

```
" Es cert que: false = ([ :x | false ] value: <qualsevol cosa>) ?!?"  
" Es a dir, per a tot programa P amb un 'forat' P = P[[forat]] on es pugui"  
" posar un boolea, no importa si posem 'false' o [ :x | false ] value: <qualsevol cosa>"  
" P[[ false ]] = P[[ ([ :x | false ] value: <qualsevol cosa>) ]]"  
"  
" Si, si no tenim continuacions!"  
"  
" En altre cas..."  
" Si P[[ ]] es Continuation callcc: [ :k | [[ ]] ] "  
" i <qualsevol cosa> = (k value: true) "  
  
(Continuation callcc: [ :k | false ]) == (Continuation callcc: [ :k | ([ :x | false] value: (k value: true)) ]) false 60
```

<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.