Más clases

Bloques (BlockClosure)

Boolean

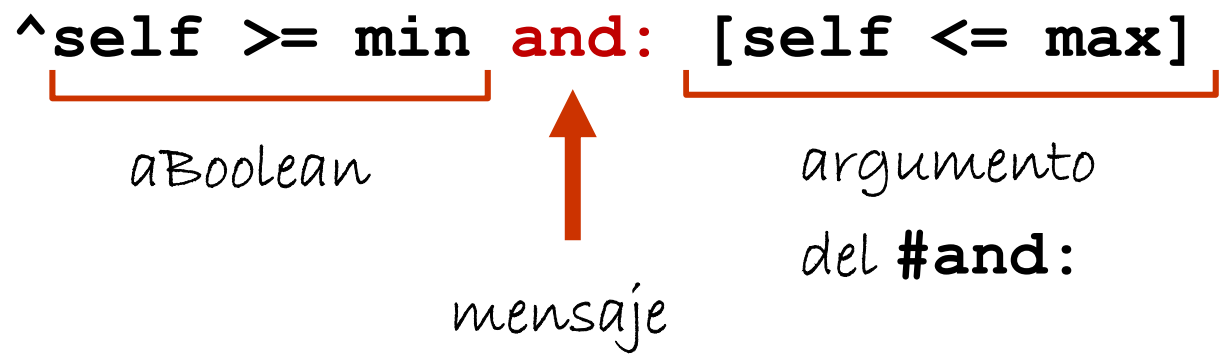
Motivación

‘dibujar un cuadrado’

```
robotech brushDown.
4 timesRepeat: [robotech move:10; direction:90].
robotech brushUp
```

Motivación

```
Magnitude>>between: min and: max
    "Answer whether the receiver is less than or
     equal to the argument, max, and greater than
     or equal to the argument, min."

    ^self >= min and: [self <= max]
```



Bloques

Clase: BlockClosure

Motivación

- Pensemos en definir una función:

$$f(x) = x * x + x$$

$f(2) = 6$

$f(20) = 420$

- En Smalltalk

```
|f|
f:= [:x | x * x + x].

f value: 2   ------ >   6
f value:20   ------ >  420
```

Otros ejemplos

```
[2 + 3 + 4 + 5] value
```

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

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

Bloques: clase BlockClosure

- Los *bloques* son instancias de la clase *BlockClosure* que contienen una secuencia de sentencias

- es esencialmente una función anónima.

- se evalúa mediante el envío del mensaje *#value*
  - devuelve el valor de la última expresión en su cuerpo
    - Salvo que haya una sentencia de retorno *^anObject*

- son definidos como literales usando los *[...]*

```
» [30 squared]
» [Transcript clear. Transcript show: 'Hello']
```

Bloques: clase BlockClosure (2)

- Representan una ejecución retardada, las sentencias serán ejecutadas sólo si el programa lo requiere explícitamente cuando se le envía el mensaje #value

```
[30 squared]  value

[30 squared. 2 + 5]  value

[ Transcript clear.
  Transcript show: 'Hello'] value
```

Bloques: clase BlockClosure (3)

- como los bloques son objetos, entonces
  - pueden ser ligados a variables

```
incrementBlock := [index := index + 1.
                   anArray at: index]

. . .
incrementBlock value
```

Bloques: clase BlockClosure (3)

- como los bloques son objetos, entonces
  - puede tener argumentos, en ese caso se usa el mensaje ***#value:*** que liga su argumento al argumento del bloque antes de su ejecución

```
sizer := [ :array | array size < 3].
sizer value: #(a b c d)    →  4
sizer value: #(1 2)        →  2
```

Bloques: clase BlockClosure (3)

- Un bloque puede tener variables temporales:

```
[:x :y | | z | z := x+ y. z ] value: 1 value: 2
```

- Pueden contener variables definidas afuera del bloque

```
| x |
x := 1.
[ :y | x + y ] value: 2    -> 3
```

Uso de bloques para iterar: #timesRepeat:

```
| amount |
  amount:=0.
  4 timesRepeat: [amount :=amount + 1]
```

```
Integer>>timesRepeat: aBlock
      "Evaluate the argument, aBlock, the number
      of times represented by the receiver."

      | count |
      count := 1.
      [count <= self]
            whileTrue:
                  [aBlock value.
                  count := count + 1]
```

La Clase `Boolean`

Clase **Boolean**

- es una clase abstracta cuya clases concretas son **`True`** y **`False`**

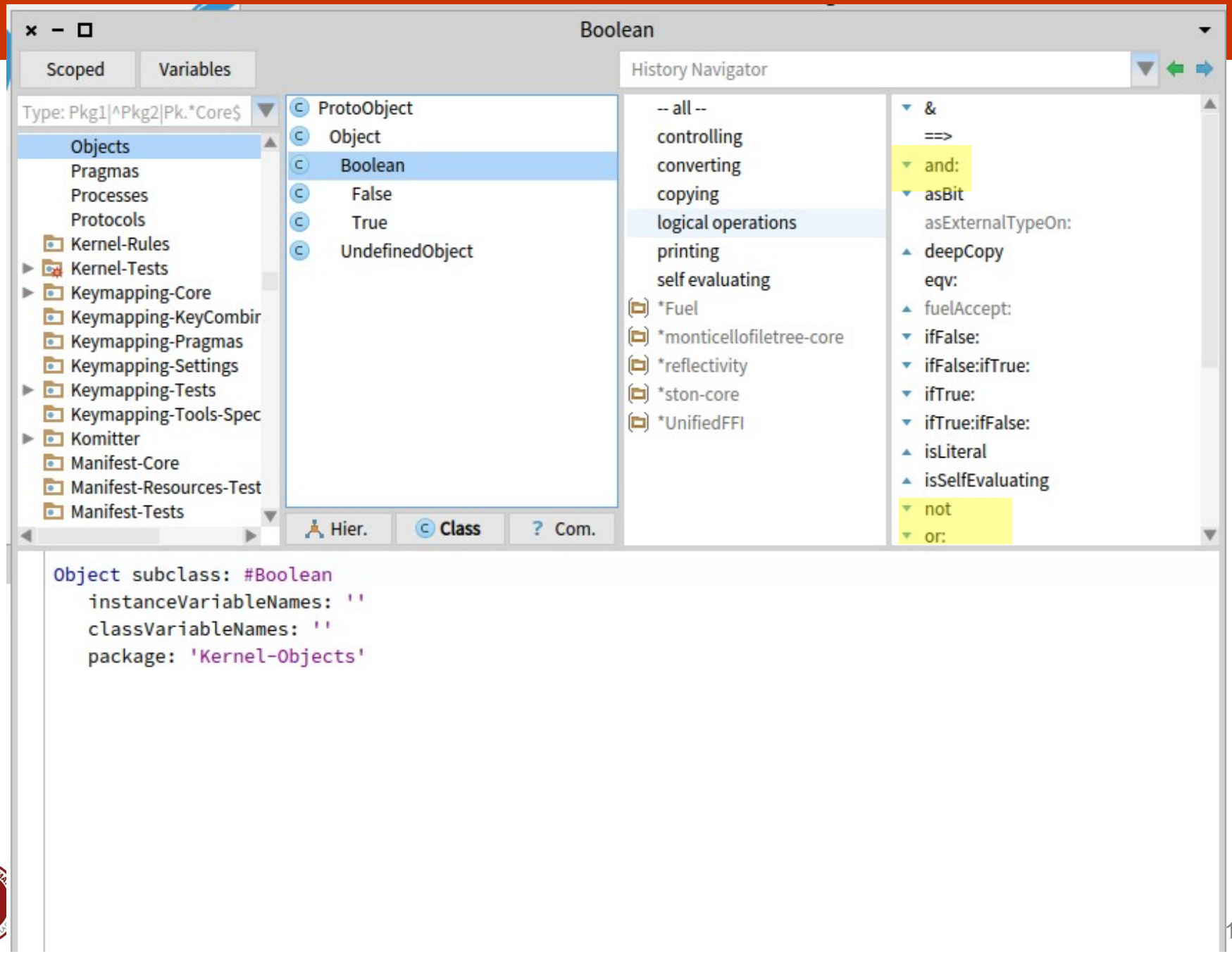
- Existen sólo 2 instancias:

  `True` ---> *true*

  `False` ---> *false*

- Las siguientes expresiones devuelven un booleano

```
a<b
a<b and:[b<c]
a<b or:[b<c]
a<b not
```


Boolean
Scoped
Variables
History Navigator
Type: Pkg1|^Pkg2|Pk.*Core$
Objects
Pragmas
Processes
Protocols
Kernel-Rules
Kernel-Tests
Keymapping-Core
Keymapping-KeyCombir
Keymapping-Pragmas
Keymapping-Settings
Keymapping-Tests
Keymapping-Tools-Spec
Komitter
Manifest-Core
Manifest-Resources-Test
Manifest-Tests
ProtoObject
Object
Boolean
False
True
UndefinedObject
Hier.
Class
Com.
-- all --
controlling
converting
copying
logical operations
printing
self evaluating
*Fuel
*monticellofiletree-core
*reflectivity
*ston-core
*UnifiedFFI
&
==>
and:
asBit
asExternalTypeOn:
deepCopy
eqv:
fuelAccept:
ifFalse:
ifFalse:ifTrue:
ifTrue:
ifTrue:ifFalse:
isLiteral
isSelfEvaluating
not
or:
Object subclass: #Boolean
instanceVariableNames: ''
classVariableNames: ''
package: 'Kernel-Objects'

Boolean: #and:

¿Dónde se implementa el #and: ?

```
Boolean>> and:aBlock
```

si el receptor es `false` ⇒ `^false`

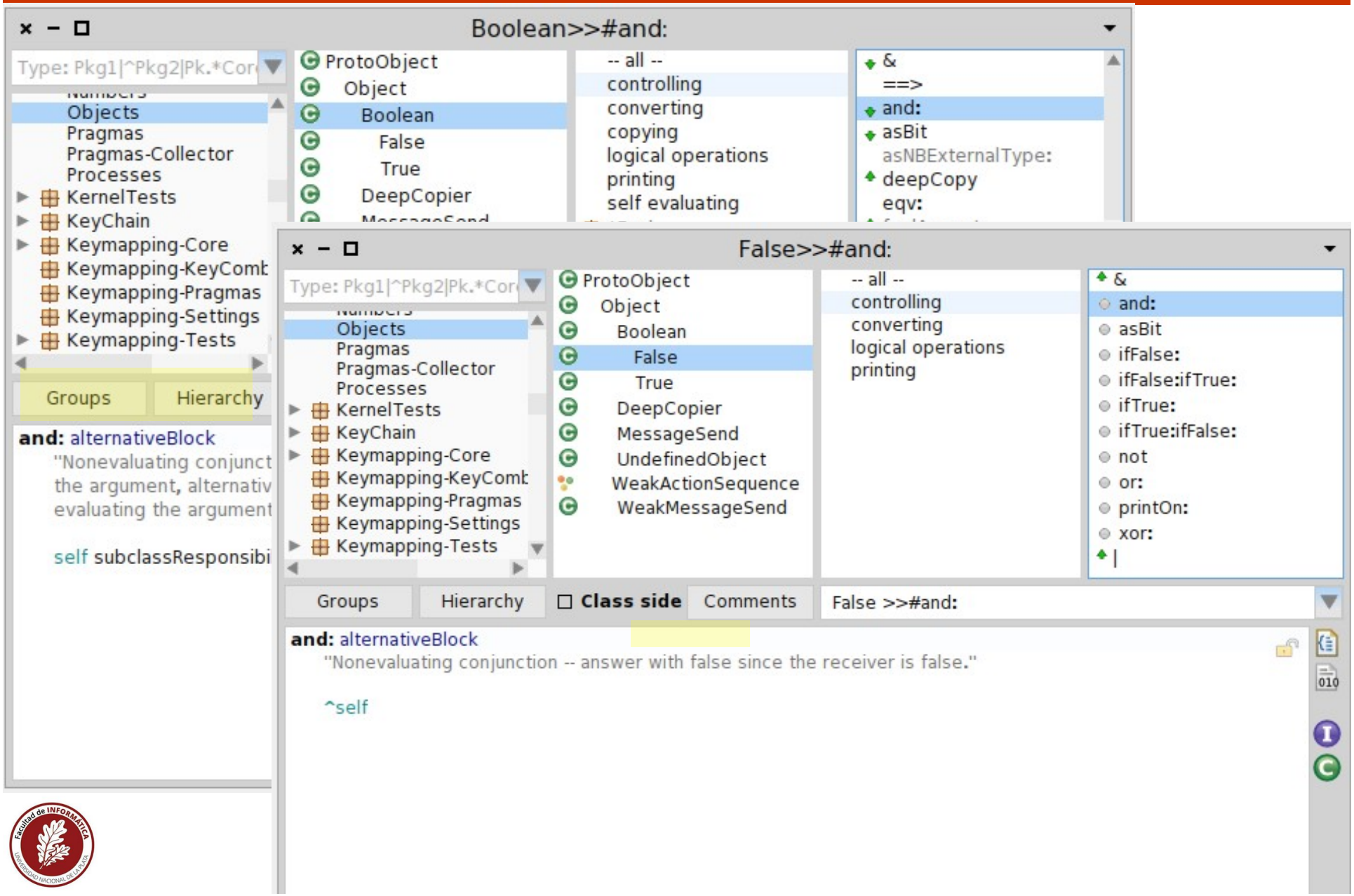
sino ⇒ `^aBlock value`

¿Hace falta preguntar si es `false` o `true`?

```
True>> and:aBlock
        ^aBlock value
```

```
False>> and:aBlock
        ^false
```

La clase **Boolean** es abstracta



Boolean: más comportamiento

```
(3 < 4) & (5 < 6)          "logical AND"
(3 < 4) | (5 < 6)          "logical OR"

#ifTrue:
   a < b ifTrue:[Transcript clear;
                 show: ` a is less than b'].

#ifFalse:
   a < b ifFalse:[Transcript clear;
                  show: ` a is NOT less than b'].

#ifTrue:ifFalse:
   a < b ifTrue: [Transcript clear;
                  show: ` a is less than b']
         ifFalse:[Transcript clear;
                  show: ` a is NOT less than b'].

#ifFalse:ifTrue:
```

Implementación del `#ifTrue:aBlock`

```
True>> ifTrue: aBlock
    "Evaluate the given Block if I am an instance of
        class True. Otherwise, do nothing."

    ^aBlock value
```

Hay que implementarlo en la clase False *?*

```
False>> ifTrue: aBlock
    "Evaluate the given Block if I am an instance of
        class True. Otherwise, do nothing."
```

Ejercicio: implemetar el `#ifTrue:ifFalse:`

Boolean

#whileTrue: aBlock

```
index := 1.
[index <= list size] whileTrue:
                        [list at: index put: 0.
                         index := index + 1]
```

En qué clase esta implementado?

BlockClosure>> whileTrue: aBlock

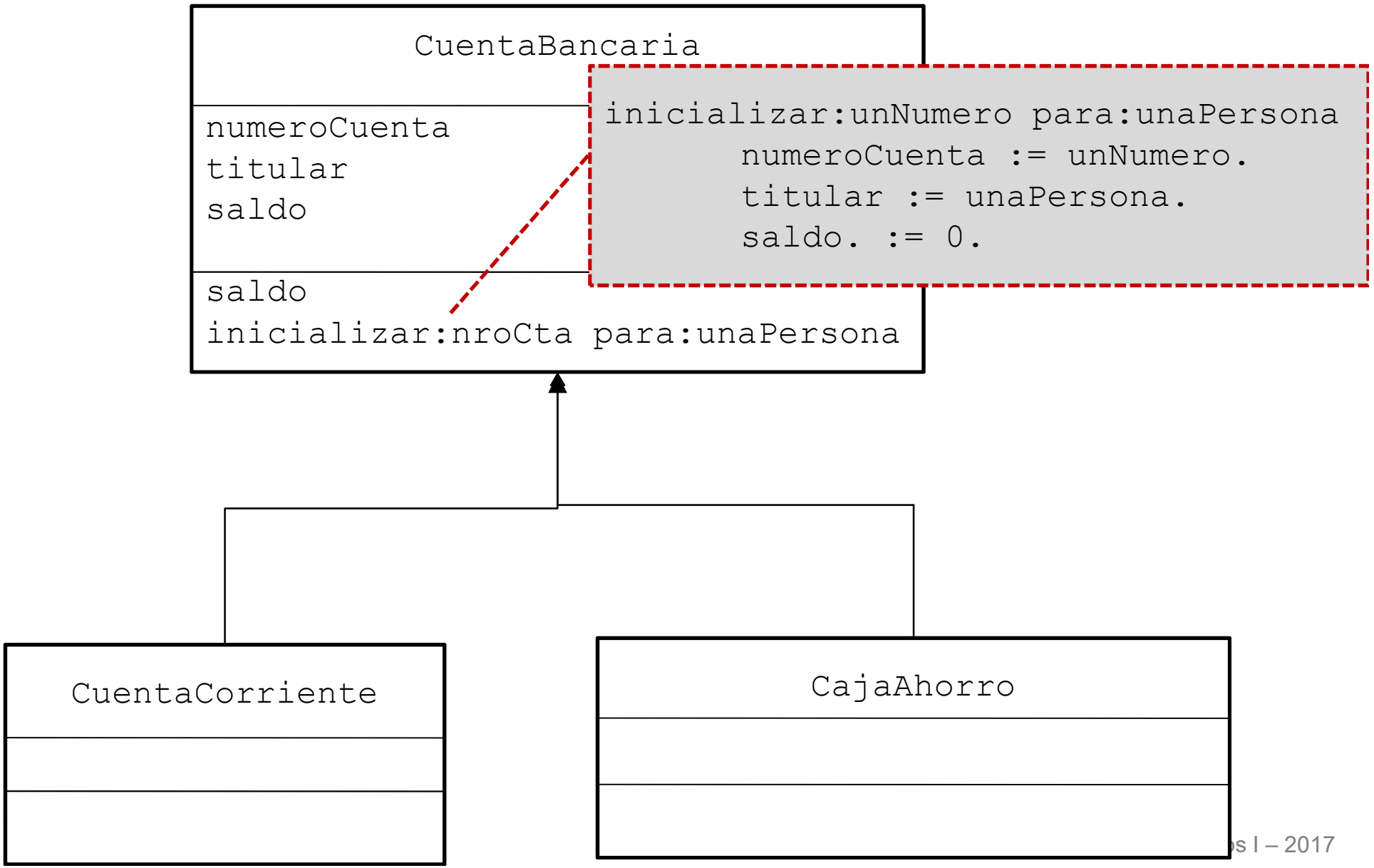
```
"Ordinarily compiled in-line, and therefore not overridable.
 This is in case the message is sent to other than a literal block.
 Evaluate the argument, aBlock, as long as the value of the receiver
 is true."

self value ifTrue: [aBlock value.
                    self whileTrue: aBlock]
```

Self + Super

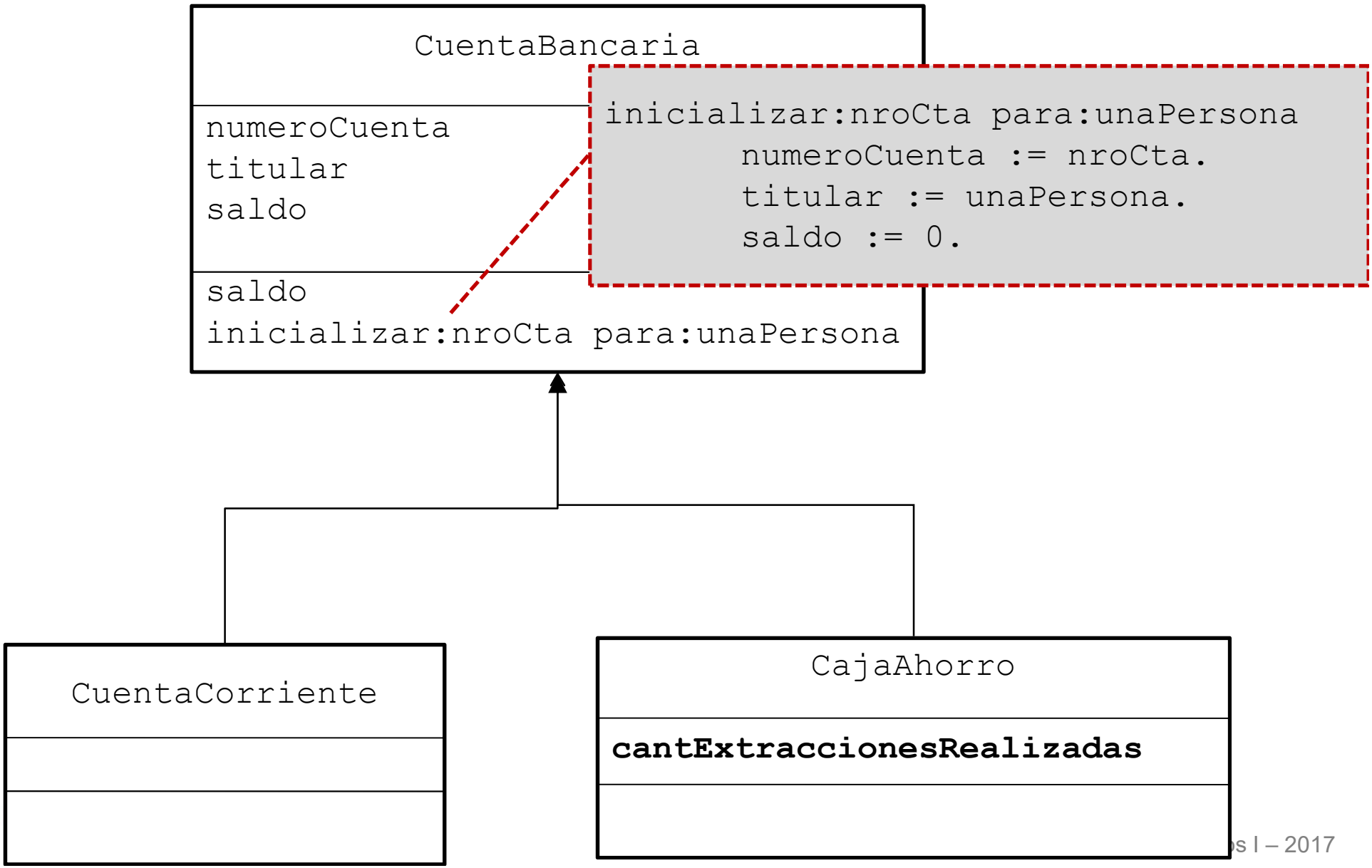
La pseudo variale **super**

- Sirve para alterar la búsqueda de un método en la jerarquía de clases
  - Indica: empezar por la superclase
  - pero, cuál superclase?

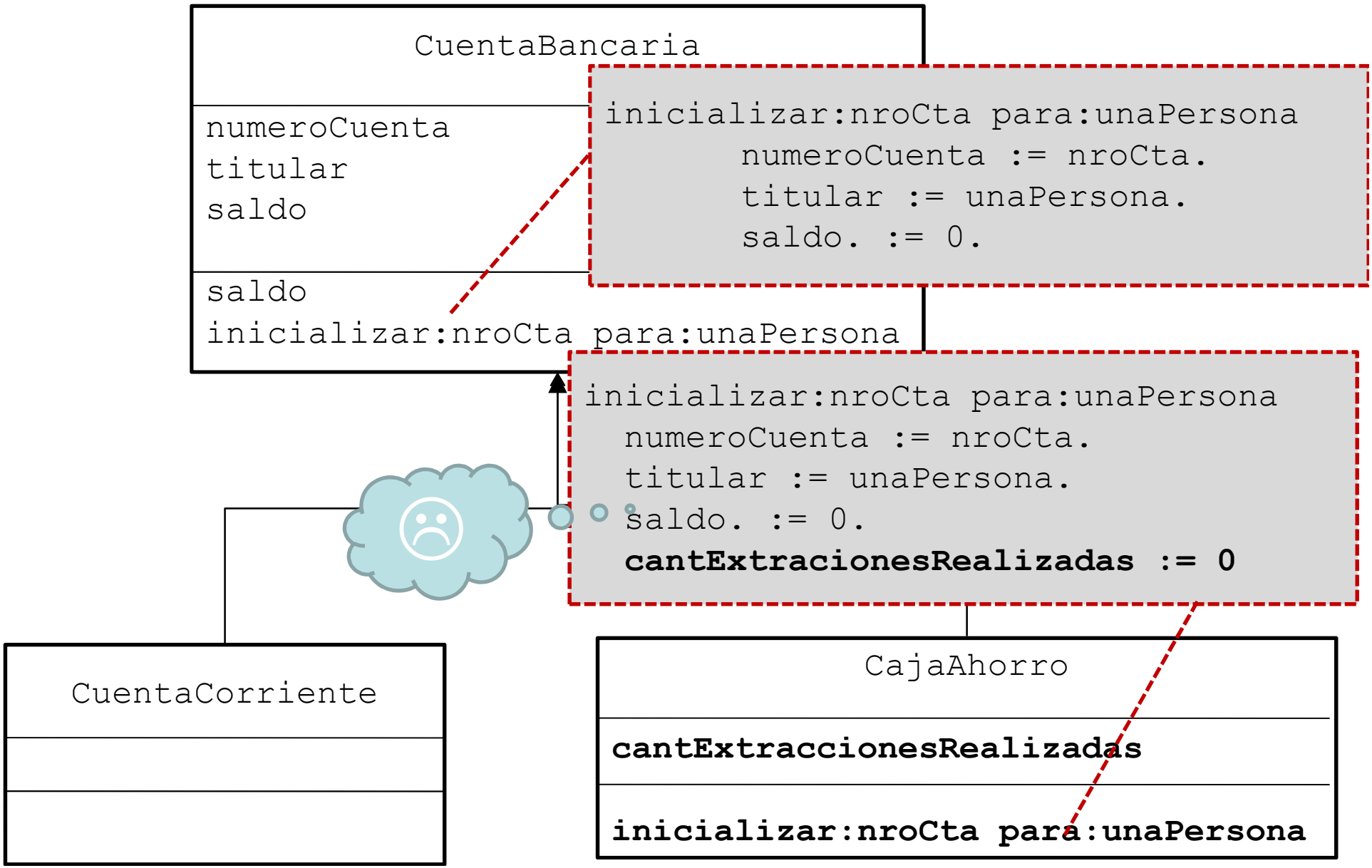
Supongamos que Inicializar una Caja de Ahorro es lo mismo que inicializar una cuenta bancaria, entonces...



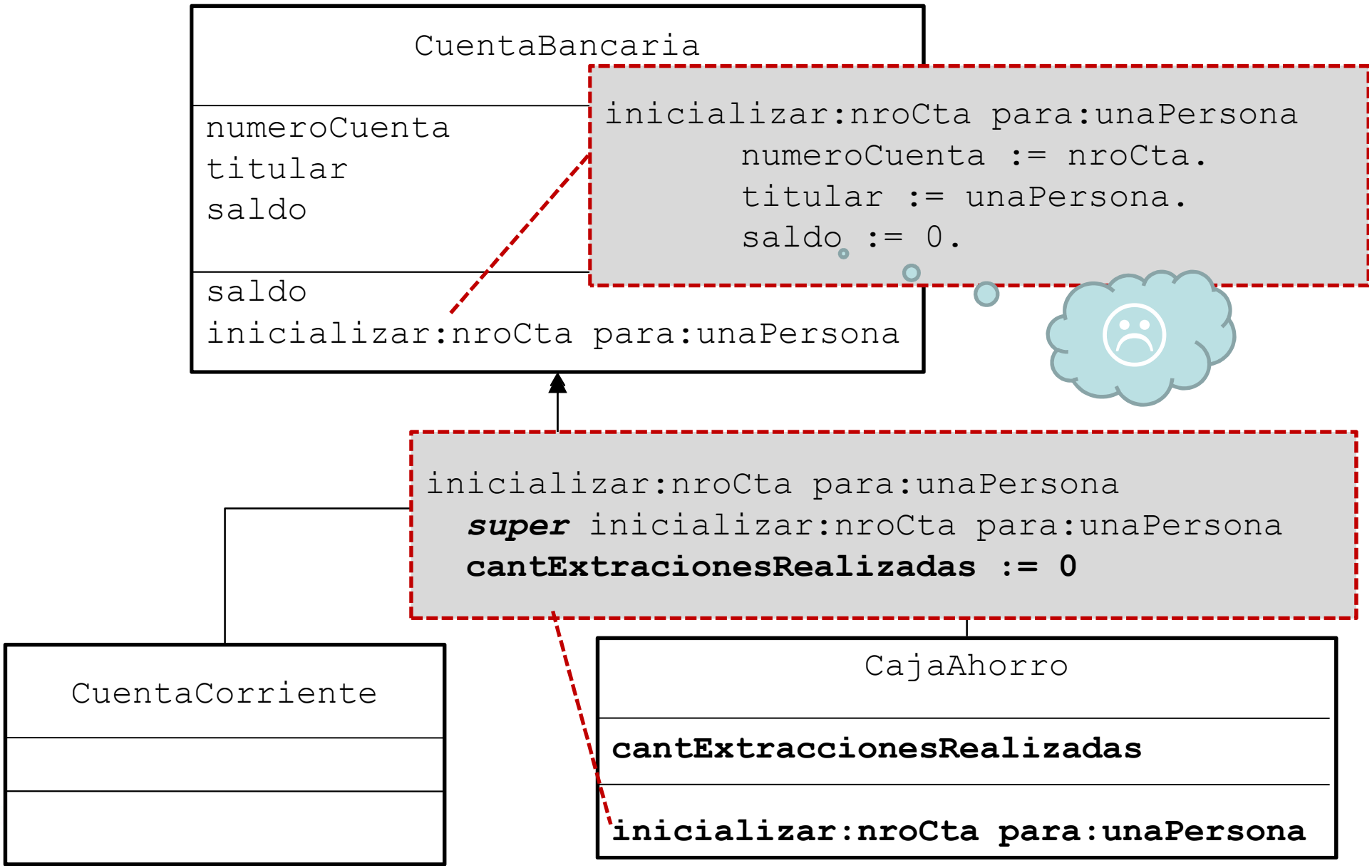
Pero sí ahora... **CajaAhorro** controla la cantidad de extracciones realizadas...



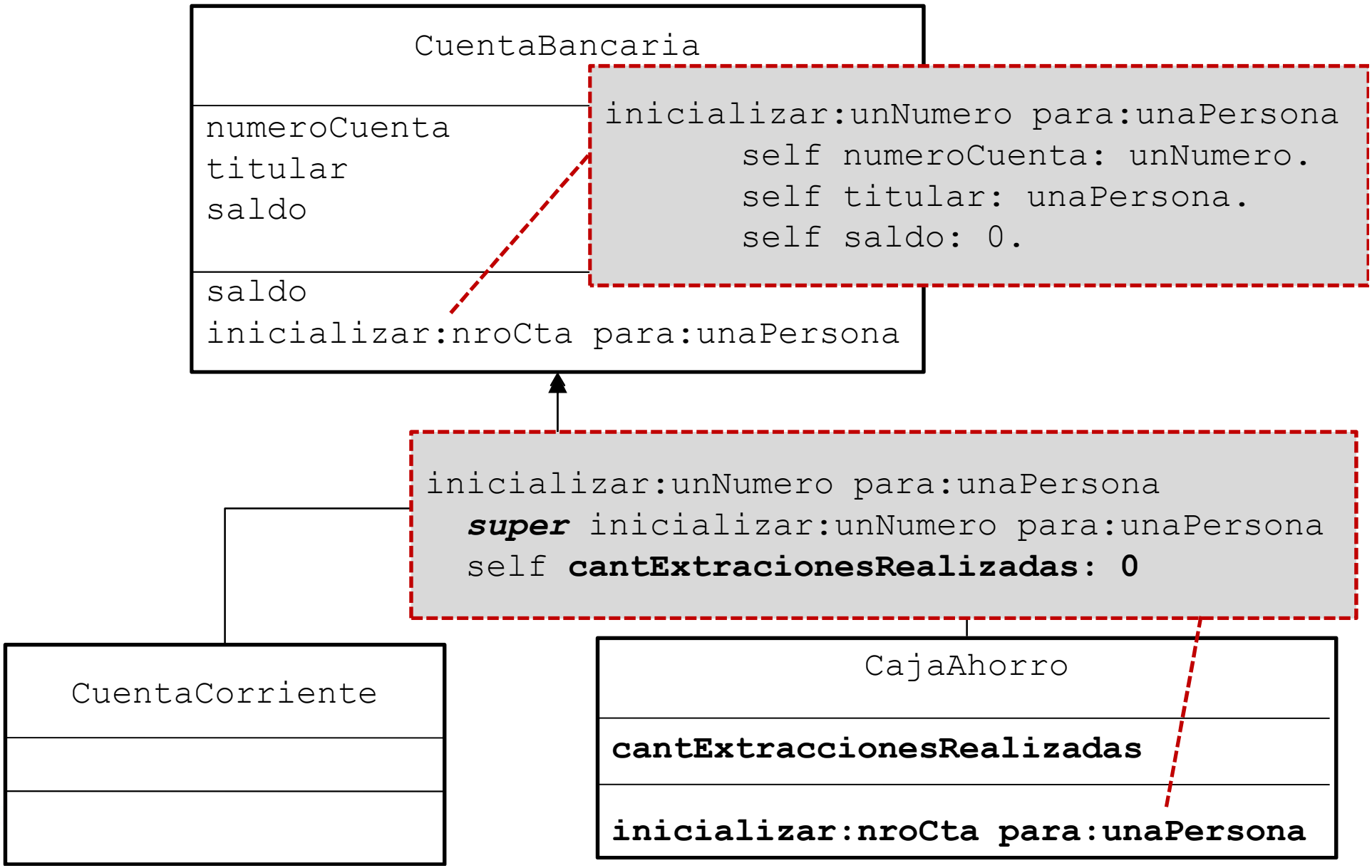
Ahora la inicialización sería....



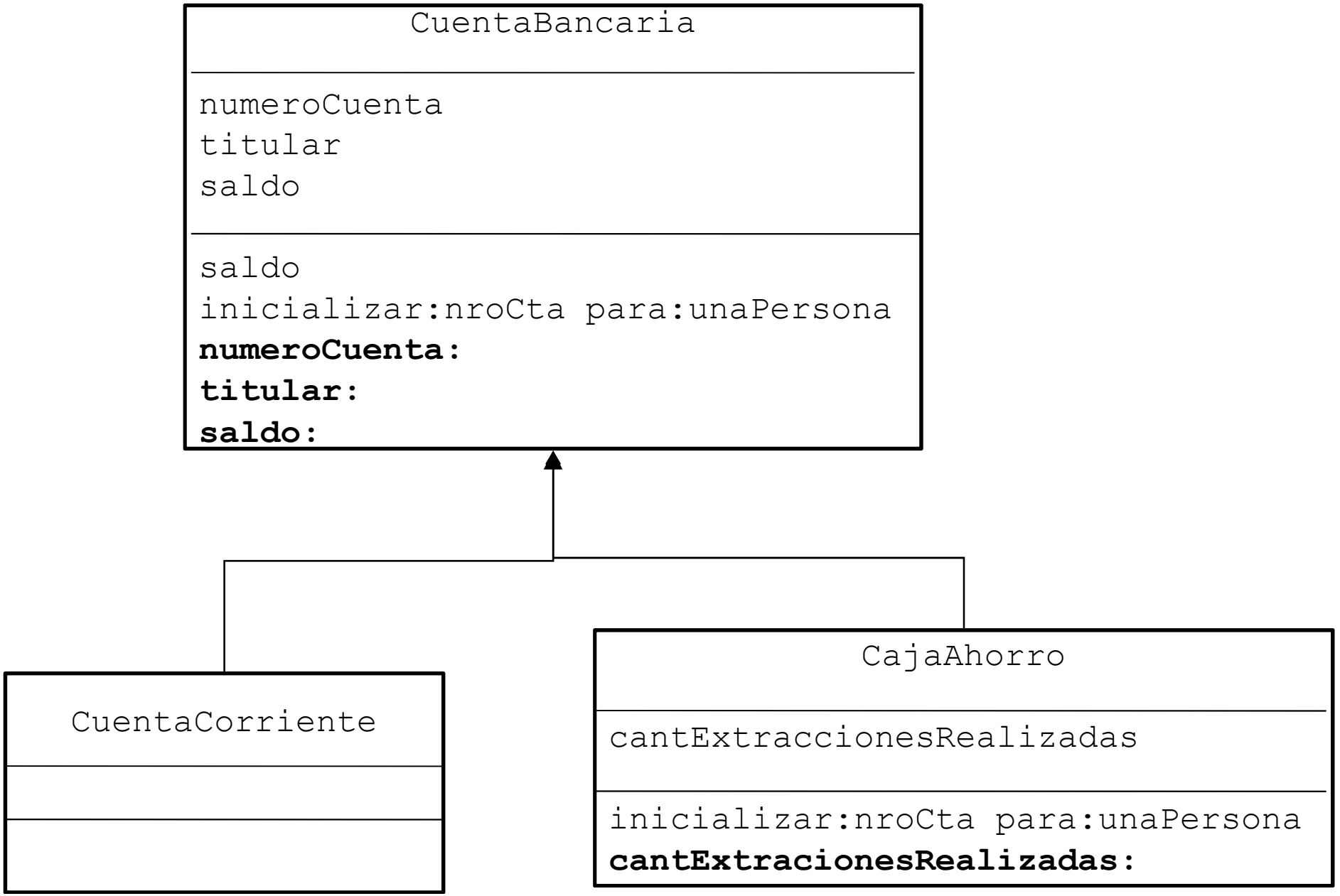
Como evitamos el código redundante?



Las variable de isntancia solo las modificamos a través de métodos setters ....



Finalmente, nos queda....



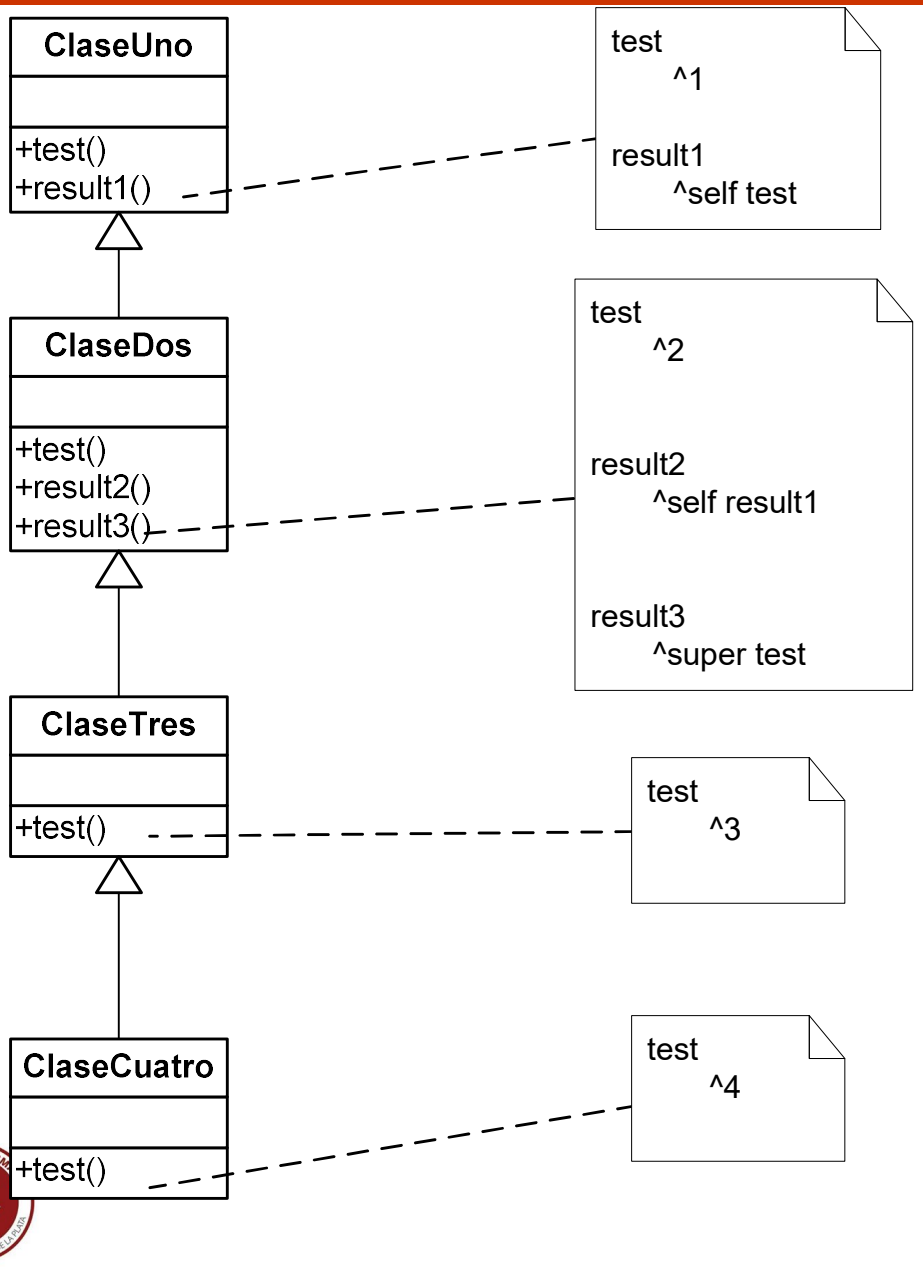
Diferencia entre **self** y **super**

super también como **self**, representa el receptor del mensaje

Pero…

cambia el mecanismo de lookup del método

- la búsqueda comienza en la superclase de la clase del método donde esta escrita la sentencia que refiere al **super**




```
|exp1|
exp1 := ClaseUno new.
exp1  test.

|exp1|
exp1 := ClaseUno new.
exp1 result1.

|exp2|
exp2 := ClaseDos new.
exp2 test

|exp2|
exp2 := ClaseDos new.
exp2 result1

|exp3 exp4|
exp3 := ClaseTres new.
exp4 := ClaseCuatro new.
exp3  test
exp4 result1
exp3 result2
exp4 result2
exp3 result3
exp4 result3
```