Más clases

Bloques (BlockClosure)

Boolean

Motivación: #between:and:

```
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]
```

aBoolean

mensaje

argumento del **#and:**

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 al menos 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
```

- 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:

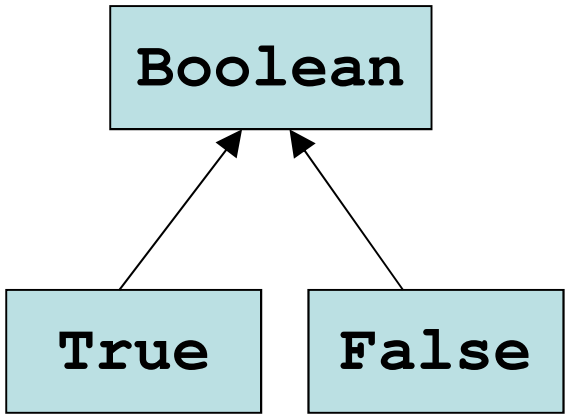
```
| amount |
  amount:=0.
  4 timesRepeat: [amount :=amount + 1]
```

```
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**

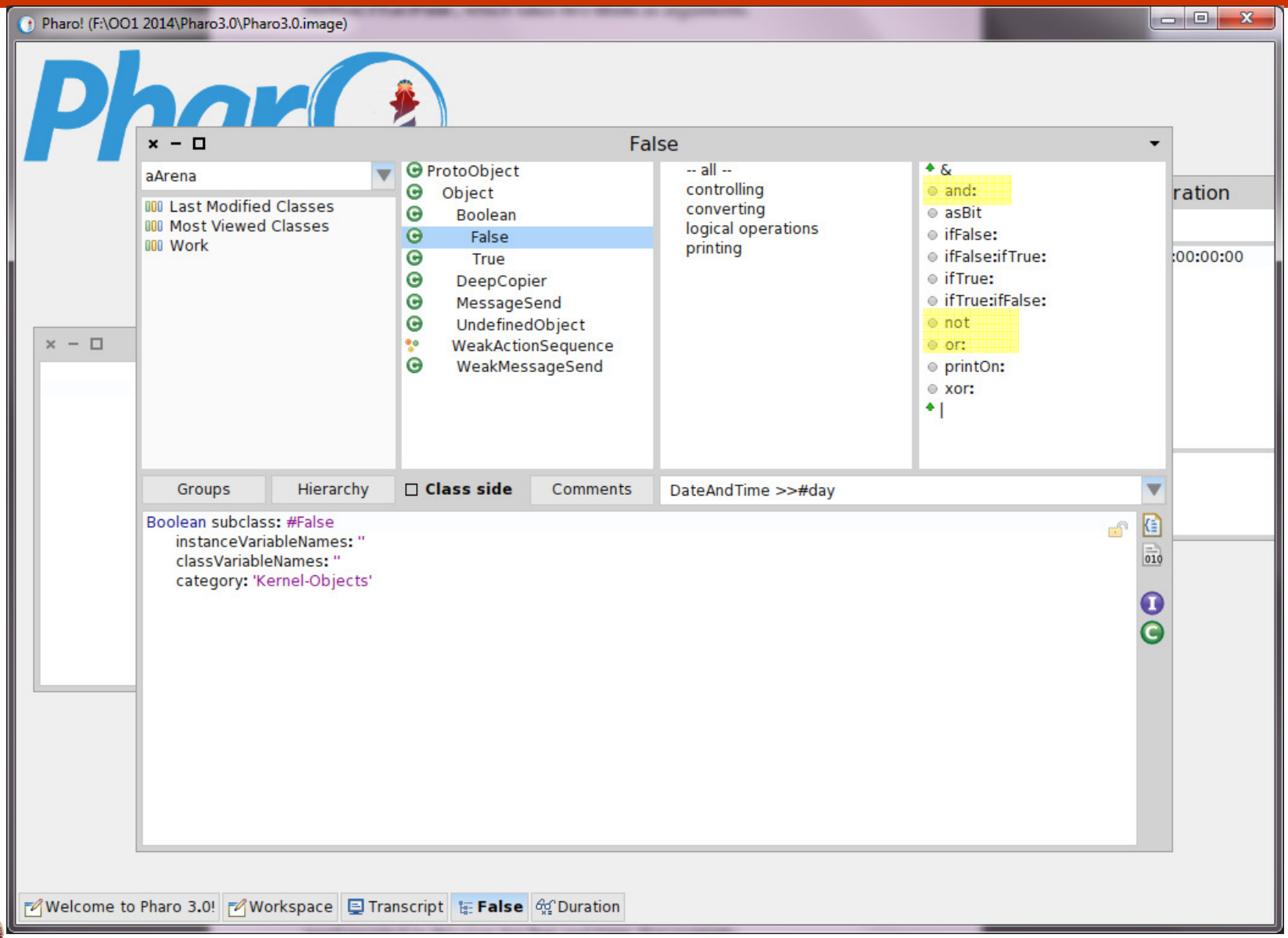
- es una clase abstracta cuya clases concretas son **True** y **False**



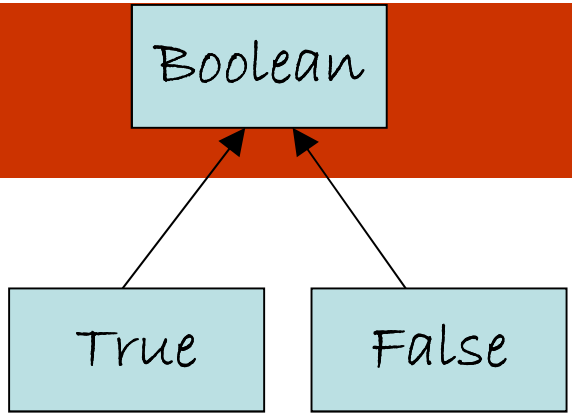
True ---> true

False ---> false

```
a<b ---> aBoolean

a<b and: [b<c]

a<b or: [b<c]

a<b not
```


Pharo! (F:\OO1 2014\Pharo3.0\Pharo3.0.image)
False
aArena
Last Modified Classes
Most Viewed Classes
Work
ProtoObject
Object
Boolean
False
True
DeepCopier
MessageSend
UndefinedObject
WeakActionSequence
WeakMessageSend
-- all --
controlling
converting
logical operations
printing
&
and:
asBit
ifFalse:
ifFalse:ifTrue:
ifTrue:
ifTrue:ifFalse:
not
or:
printOn:
xor:
Groups
Hierarchy
Class side
Comments
DateAndTime >>#day
Boolean subclass: #False
instanceVariableNames: ''
classVariableNames: ''
category: 'Kernel-Objects'
Welcome to Pharo 3.0!
Workspace
Transcript
False
Duration

Boolean: #and:



¿Dónde se implementa el #and: ?

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

si el receptor es `false` ⇒ `^false`

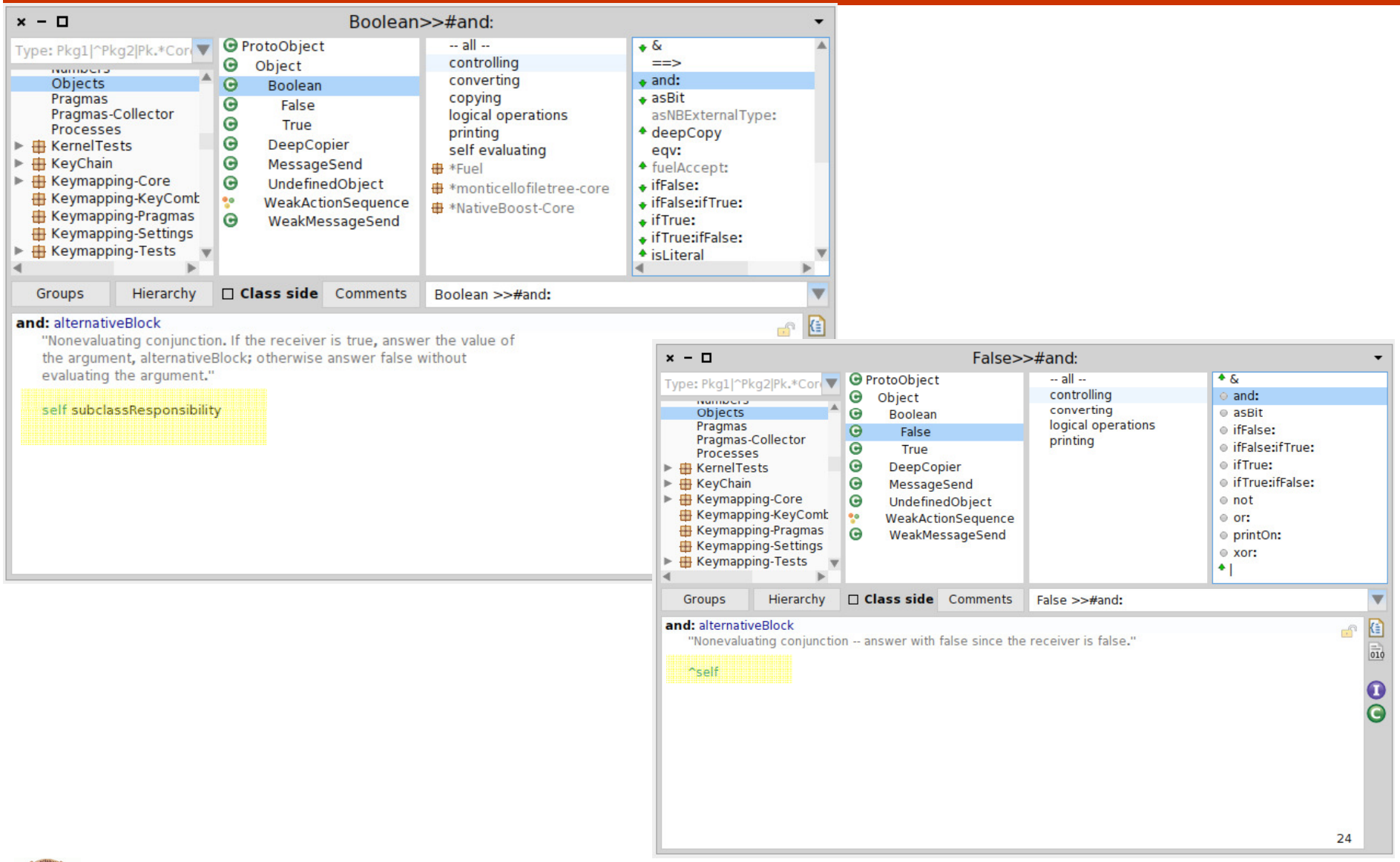
sino ⇒ `^aBlock value`

¿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
        "Evaluate the receiver block repeatedly
        and each time it evaluates to true,
        evaluate the argument block. Continue
        until the receiver block does not
        evaluate to true."

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