INRIA

Objects to the Roots:
Learning from beauty

Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

Really?!

No primitive types
No hardcoded constructs for conditional
Only messages
Only objects

and this works?
I mean really?
Not even slow?
Can't be real!

Motto

Let's open our eyes, look, understand, and deeply understand the underlying design aspects of object-oriented programming...

Booleans

```
3 > 0
    ifTrue: ['positive']
    ifFalse: ['negative']
```


Booleans

```
3 > 0
    ifTrue: ['positive']
    ifFalse: ['negative']

'positive'
```


Yes ifTrue:ifFalse: is a message!

```
Weather isRaining
    ifTrue: [self takeMyUmbrella]
    ifFalse: [self takeMySunglasses]
```

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

Booleans

& | not
or: and: (lazy)
xor:
ifTrue:ifFalse:
ifFalse:ifTrue:
...

Lazy Logical Operators

```
false and: [1 error: 'crazy']
```

PrIt-> false and not an error

Yes! ifTrue:ifFalse: is a message send to a Boolean.

But optimized by the compiler :)

Implementing not

Now you are good and you should implement it

Propose an implementation of not in a world where you do not have Booleans

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

Implementing ifTrue:ifFalse:

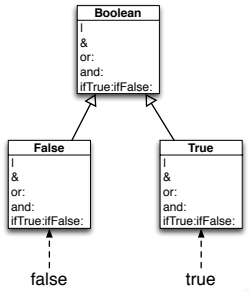
Now you are good and you should implement it

Propose an implementation of not in a world where you do not have Booleans

false ifTrue: [3] ifFalse: [5]
true ifTrue: [3] ifFalse: [5]

Boolean Objects

false and true are objects described by classes Boolean, True and False



Let's the receiver decide!

Boolean>>not

"Class Boolean is an abstract class that implements behavior common to true and false. Its subclasses are True and False. Subclasses must implement methods for logical operations &, not, controlling and:, or:, ifTrue:, ifFalse:, ifTrue:ifFalse:, ifFalse:ifTrue:"

Boolean>>not
"Negation. Answer true if the receiver is false, answer false if the receiver is true."
self subclassResponsibility

Not

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

Boolean>>not
"Negation. Answer true if the receiver is false, answer false if the receiver is true."
self subclassResponsibility

False>>not
"Negation -- answer true since the receiver is false."
^true
True>>not
"Negation--answer false since the receiver is true."
^false

| (Or)

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

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

Boolean>> | aBoolean

Boolean>> | aBoolean
"Evaluating disjunction (OR). Evaluate the argument. Answer true if either the receiver or the argument is true."

self subclassResponsibility

False>> | aBoolean

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

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

^ aBoolean

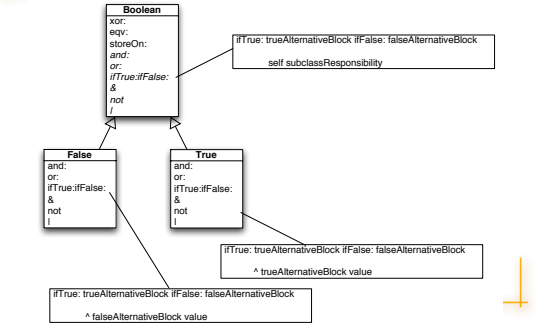
True>> | aBoolean

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

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

^ self

Boolean, True and False



Implementation Note

Note that the Virtual Machine shortcuts calls to boolean such as condition for speed reason.

Virtual machines such as VisualWorks introduced a kind of macro expansion, an optimisation for essential methods and Just In Time (JIT) compilation. A method is executed once and afterwards it is compiled into native code. So the second time it is invoked, the native code will be executed.

Ok so what?

You will probably not implement another Boolean classes

So is it really that totally useless?

Ternary logic

Boolean: true, false, unknown

A	*B*	*A* OR *B*	*A* AND *B*	NOT *A*
True	True	True	True	False
True	Unknown	True	Unknown	False
True	False	True	False	False
Unknown	True	True	Unknown	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
False	True	True	False	True
False	Unknown	Unknown	False	True
False	False	False	False	True

More important...

Message sends act as case statements

OOP: the art of dispatching

Subclasses create your vocabulary

Avoid Conditional

Use objects and messages

VM dispatch is a conditional switch: Use it!

AntiIfCampaign

Summary

Messages act as a dispatcher
Avoid conditional