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']

Lazy Logical Operators
false and: [1] error: 'crazy'

PrTc-> false and not an error

Booleans

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

Yes ifTrue:ifFalse: is a message!
Weather isRaining
ifTrue: [self takeMyUmbrella]
ifFalse: [self takeMySunglasses]

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

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

\[
\begin{align*}
\text{false not} & \rightarrow \text{true} \\
\text{true not} & \rightarrow \text{false}
\end{align*}
\]

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

\[
\begin{align*}
\text{false ifTrue: [3] ifFalse: [5]} \\
\text{true ifTrue: [3] ifFalse: [5]}
\end{align*}
\]

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 \rightarrow\text{true} \\
true not \rightarrow\text{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)

\[
\begin{align*}
\text{true} | \text{true} & \rightarrow \text{true} \\
\text{true} | \text{false} & \rightarrow \text{true} \\
\text{true} | \text{anything} & \rightarrow \text{true} \\
\text{false} | \text{true} & \rightarrow \text{true} \\
\text{false} | \text{false} & \rightarrow \text{false} \\
\text{false} | \text{anything} & \rightarrow \text{anything}
\end{align*}
\]

Boolean>> | aBoolean

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

self subclassResponsibility
**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.

---

**Ternary logic**

Boolean: true, false, unknown

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a OR b</th>
<th>a AND b</th>
<th>NOT a</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>U</td>
</tr>
<tr>
<td>T</td>
<td>U</td>
<td>T</td>
<td>F</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>T</td>
<td>F</td>
<td>U</td>
</tr>
</tbody>
</table>

---

**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!

Summary

Messages act as a dispatcher
Avoid conditional