Essence of Dispatch

Taking Pharo Booleans as Example

Damien Cassou, Stéphane Ducasse and Luc Fabresse
Objectives

- Understanding of message passing (late binding)
  - the heart of OOP
  - more an OOP lecture than a Pharo one
- Insight at how beautiful Pharo’s implementation is
Context: Booleans

In Pharo, Booleans have a superb implementation!

- &, |, not (eager)
- or:, and: (lazy)
- ifTrue:ifFalse:, ifFalse:ifTrue:
Three Exercises

1. Implement not (Not)
2. Implement | (Or)
3. What is the goal of these exercises?
Exercise 1: Implement Not

Propose an implementation of Not in a world where:

- You have: true, false
- You only have objects and messages
- How would you implement the message not?

\[
\begin{align*}
\text{false not} & \rightarrow \text{true} \\
\text{true not} & \rightarrow \text{false}
\end{align*}
\]
Hint 1: No conditionals

The solution does not use conditionals (i.e., no if)
Hint 2: With Three Classes

- The solution uses three classes:
  - Boolean (abstract), True and False
- true is the singleton instance of True
- false is the singleton instance of False
Hint 2: Three Classes

```
<table>
<thead>
<tr>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
</tr>
<tr>
<td>or:</td>
</tr>
<tr>
<td>l</td>
</tr>
<tr>
<td>ifTrue:ifFalse:</td>
</tr>
</tbody>
</table>

<<abstract>>

```

```
<table>
<thead>
<tr>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
</tr>
<tr>
<td>or:</td>
</tr>
<tr>
<td>l</td>
</tr>
<tr>
<td>ifTrue:ifFalse:</td>
</tr>
</tbody>
</table>

true

```

```
<table>
<thead>
<tr>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
</tr>
<tr>
<td>or:</td>
</tr>
<tr>
<td>l</td>
</tr>
<tr>
<td>ifTrue:ifFalse:</td>
</tr>
</tbody>
</table>

false

```
Hint 3: How do We Express Choice in OOP?

In OOP, choice is expressed

- By defining classes with compatible methods
- By sending a message to an instance of such class

Example

```plaintext
x open
```

- `x` can be a file, a window, a tool,...
- The method is **selected** based on `x`'s class
Implementation of Not in Two Methods

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

True >> not
"Negation — answer false since the receiver is true."
^ false
Implementation Hierarchy

```
Boolean
  not
  or:
  /
  ifTrue:ifFalse:

<<abstract>>

True
  not
  or:
  /
  ifTrue:ifFalse:

^ false

False
  not
  or:
  /
  ifTrue:ifFalse:

true
false

^ true
```
Message Lookup is Choosing the Right Method

```
Boolean
not
or:
/
ifTrue:ifFalse:

<<abstract>>

True
not
or:
/
ifTrue:ifFalse:

False
not
or:
/
ifTrue:ifFalse:

^false

true

false

not

not

^true
```
Boolean Implementation

- Boolean is abstract
- Subclasses are True and False and implement
  - logical operations & and not
  - control structures and:, or:, ifTrue:, ifFalse:, ifTrue:ifFalse:, ifFalse:ifTrue:

```ruby
Boolean>>not
"Abstract method. Negation: Answer true if the receiver is false, answer false if the receiver is true."
self subclassResponsibility
```
## Behavior of Or

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>anything</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>anything</td>
<td>anything</td>
</tr>
</tbody>
</table>
Implementation of Or in Boolean

```
Boolean >> | aBoolean
  "Abstract method. Evaluating Or: Evaluate the argument. Answer true if either the receiver or the argument is true."
  self subclassResponsibility
```
Implementation of Or in Class False

<table>
<thead>
<tr>
<th>false</th>
<th>true  --&gt; true</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false  --&gt; false</td>
</tr>
<tr>
<td>false</td>
<td>anything  --&gt; anything</td>
</tr>
</tbody>
</table>

False >> | aBoolean
"Evaluating Or  --> answer with the argument, aBoolean."
^ aBoolean
# Implementation of Or in Class True

<table>
<thead>
<tr>
<th>true</th>
<th>true  -&gt;  true</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false  -&gt;  true</td>
</tr>
<tr>
<td>true</td>
<td>anything  -&gt;  true</td>
</tr>
</tbody>
</table>

**True** >> | aBoolean

"Evaluating Or — answer true since the receiver is true."

^ true
The object true is the receiver of the message!

```
True>> | aBoolean
    "Evaluating disjunction (Or) — answer true since the receiver is true."
    ^ true
```

So we can write it like the following:

```
True >> | aBoolean
    "Evaluating disjunction (Or) — answer true since the receiver is true."
    ^ self
```
Or Implementation in Two Methods

```
Boolean
not
or:
/
ifTrue:ifFalse:
```

```
True
not
or:
I arg
ifTrue:ifFalse:
```

```
False
not
or:
I arg
ifTrue:ifFalse:
```

```
^ self
```

```
true
```

```
false
```

```
^ arg
```
Summary

- The solution to implement booleans’ operations:
  - does NOT use conditionals (if)
  - lets the receiver decide
- Do not ask, tell
A course by

Inria

and

UTOP

Université de Technologie
Ouverte Pluripartenaire

in collaboration with

unisci

Fondation unit

Université de Lille

INSTITUT
Mines-Télécom

MINES
Douai

Inria 2016

Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
https://creativecommons.org/licenses/by-nc-nd/3.0/fr/