Essence of Dispatch

Taking Pharo Booleans as example

S. Ducasse, L. Fabresse, G. Polito, and P. Tesone

http://www.pharo.org
Objectives

- Understand of message passing (late binding) for **real** this time
- The **heart of Object-Oriented Design**
- Look at a beautiful implementation in Pharo
In Pharo, Booleans have a superb implementation!
You get the classical messages:

- &,
- |,
- not (eager)
- or:, and: (lazy)

And some less traditional ones:

- ifTrue:ifFalse:, ifFalse:ifTrue:
  - Yes, conditionals are messages sent to boolean objects
Three exercises

- Exo 1: Implement $\text{not}$ (Not)
- Exo 2: Implement $\mid$ (Or)
- Exo 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 objects
- You only have objects and messages

How would you implement the message `not`?

```
false not  -> true
true not   -> false
```
Hint 1: No conditionals

The solution does not use explicit conditionals (i.e., no if)
Hint 2: How do we express choices in OOP?

In OOP, the choice is expressed

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

Let the receiver decide!
Hint 2: An example of choice in OOP

- x can be a file, a window, a tool,...
- The method is **selected** based on x’s class
Hint 3: With at least two classes

- true is the singleton instance of the class True
- false is the singleton instance of the class False

The Pharo implementation uses three classes:
- The class Boolean (abstract), True, and False
Hint 3: With at least 2 classes and 2 methods

The class Boolean is not needed per se but it improves reuse
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:
```

```
True
not
or:
/
ifTrue:ifFalse:
```

```
False
not
or:
/
ifTrue:ifFalse:
```

```
^ true
```

```
^ false
```

true
false
Message lookup chooses the right method

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

<<abstract>>

```
True
not
or:
/
ifTrue:ifFalse:
true
```

```
False
not
or:
/
ifTrue:ifFalse:
false
```

```
^ true
```

```
^ false
```

not
true
not
false
Boolean implementation

- The class `Boolean` is abstract
- The classes `True` and `False` implement
  - logical operations `&`, `not`
  - control structures `and:`, `or:`, `ifTrue:`, `ifFalse:`, `ifTrue:ifFalse:`, `ifFalse:ifTrue:`
  - reuse some logic from `Boolean`
Exercise 2: Implement Or

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</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></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>
# Implementation of Or in class False

<table>
<thead>
<tr>
<th></th>
<th>true</th>
<th>false</th>
<th>anything</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>anything</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>anything</td>
</tr>
<tr>
<td>false</td>
<td>anything</td>
<td>false</td>
<td>anything</td>
</tr>
</tbody>
</table>

False >> | aBoolean

"Evaluating Or — answer with the argument, aBoolean."

^ aBoolean
Implementation of Or in class True

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>
</tbody>
</table>
Implementation of Or in class True

| true | true   | true |
|      | true   | true |
| true | false  | true |
| true | anything | true |

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
Step back

- An example of Do not ask, tell principle application
- Here:
  - We **delegate** to the correct Boolean object
  - Each subclass implements its **own** logic
Summary

We saw:

- The solution to implement boolean operations does NOT use explicit conditionals (if)
- **Sending a message is making a choice**

Remember two important principles

- Do not ask, tell
- Let the receiver decide