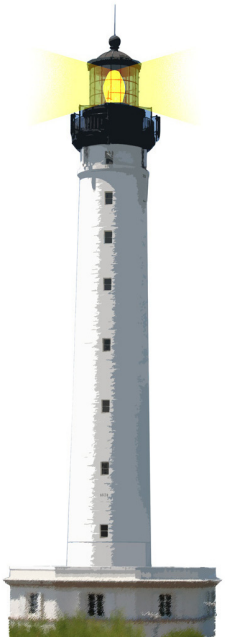
Essence of Dispatch

Damien Cassou, Stéphane Ducasse and Luc Fabresse

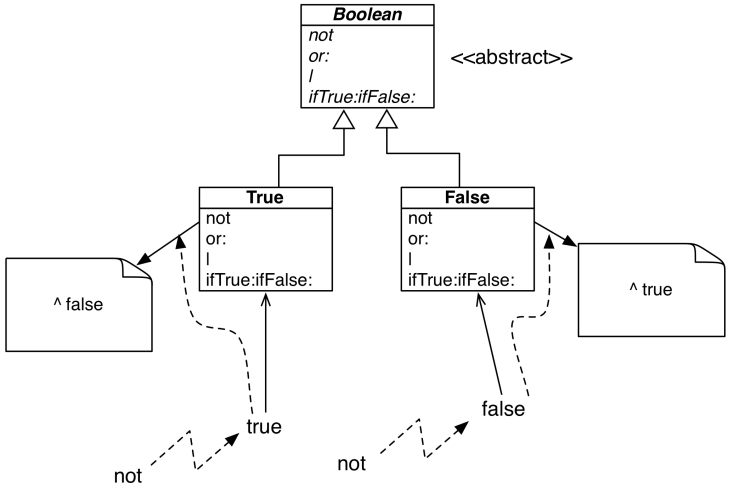
W3S02

Pharo

http://www.pharo.org



Remember: Implementing not in Two Methods



Stepping Back

- Let the receiver decide
- Do not ask, tell

Ok So What?

- You will probably never implement Booleans in the future
- So is it really that totally useless?
- What is the lesson to learn?

Message Sends Act as Case Statements

- Message sends act as case statements
- But with messages, the case statement is **dynamic** in the sense that it depends on the objects to which the message is sent

Sending a Message is Making a Choice

- Each time you send a message, the execution **selects the right** method depending on the class of the receiver
- Sending a message is a **choice** operator

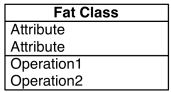
Classes Play Case Roles

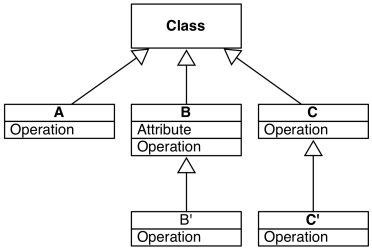
- To activate the choice operator we must have choices: classes
- A class represents a case

A Class Hierarchy is a Skeleton for Dynamic Dispatch

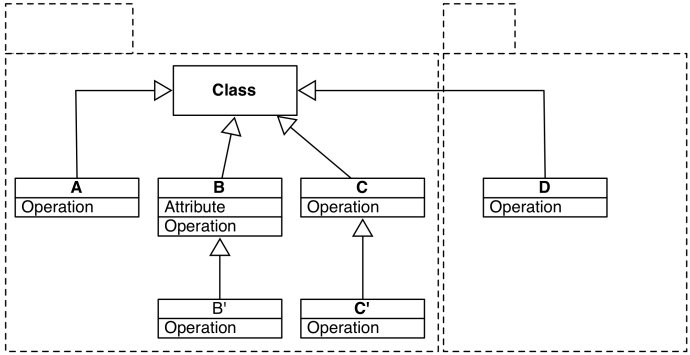
Compare the solution with one class vs. a hierarchy

- More modular
- Hierarchy provides a way to specialize behavior
- You only focus on one class at a time





Advantages of Class Hierarchy



More modular: We can package different classes in different packages

Let the Receiver Decide

- Sending a message lets the receiver decide
- Client does not have to decide
- Client code is more declarative: give orders
- Different receivers may be substitued dynamically

Avoid Conditionals

- Use objects and messages, when you can
- The execution engine acts as a conditional switch: Use it!
- Check the AntiIfCampaign

Summary: Cornerstone of OOP

- Let the receiver decide
- Message sends act as potential dynamic conditionals
- Class hierarchy: a skeleton for dynamic dispatch
- Avoid conditionals

A course by

and

in collaboration with