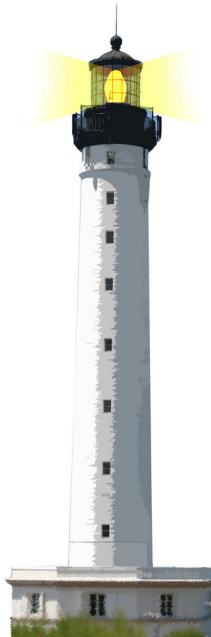
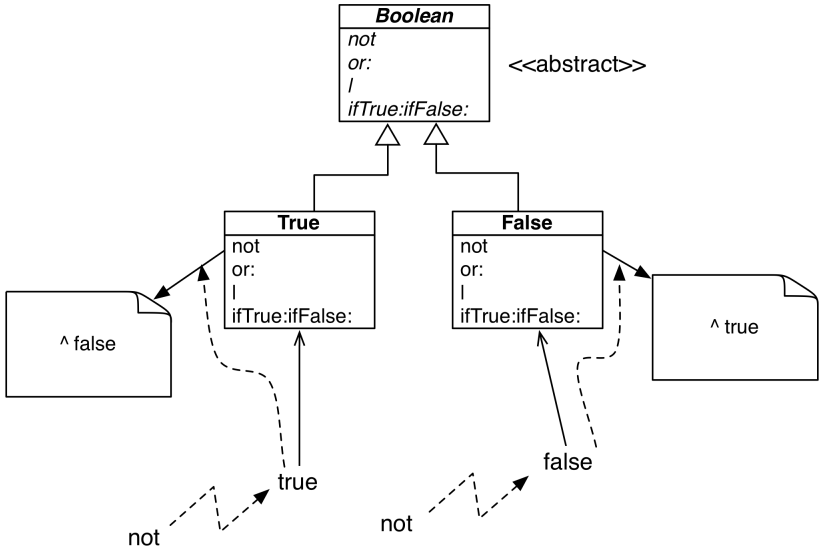
Advanced Object-Oriented Design

Essence of Dispatch

S. Ducasse and L. Fabresse



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 are supporting a choice
- They 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

How do we express choices?

- Ok we have a choice operation... then
- Could we have the same solution with a single Boolean class?

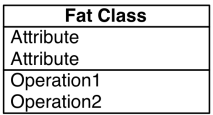
Classes play case roles

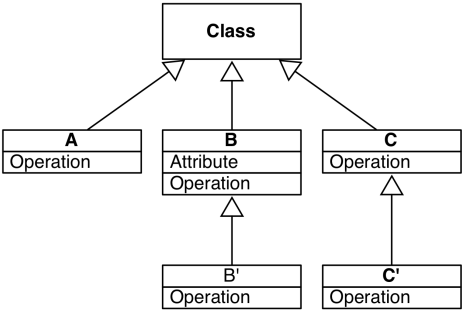
- To activate the choice operator we must have choices: classes
- We can see that a class represents a case

Class hierarchy supports for dynamic dispatch

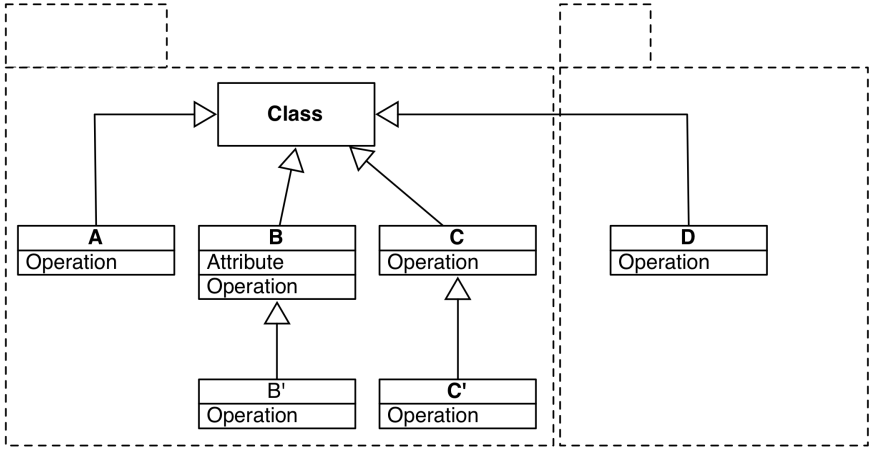
Compare the solution with one class vs. a hierarchy

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





Message dispatch supports modularity



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 support for dynamic dispatch
- Avoid conditionals

A course by

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