

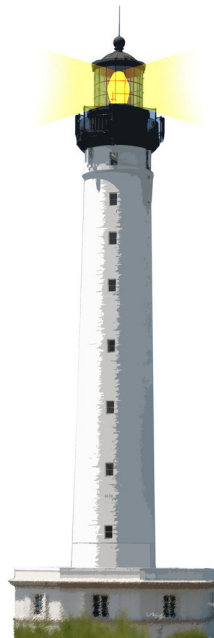
**Advanced Object-Oriented Design**

# Essence of Dispatch

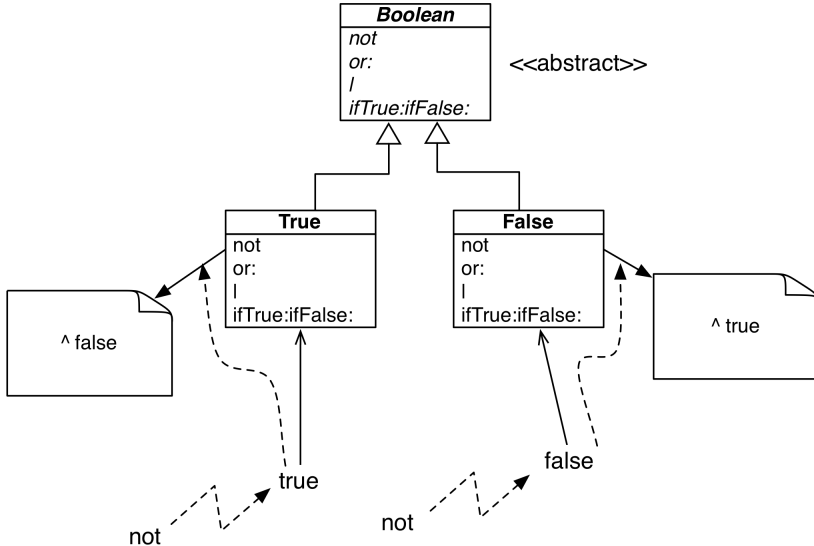
S. Ducasse



<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** useful?
- What are the lessons to learn?
- What are the properties of the solution?



# Imagine having more than two classes

MicAbstractBlock

MicAbstractAnnotatedBlock

MicAnnotatedBlock

MicContinuousMarkedBlock

MicCommentBlock

MicQuoteBlock

MicTableBlock

MicIntermediateBlock

MicListBlock

MicOrderedListBlock

MicUnorderedListBlock

MicListItemBlock

MicParagraphBlock

MacParagraphBlock

MacRawParagraphBlock

MicRootBlock

MicSectionBlock

MicSingleLineBlock

MicAnchorBlock

MicHeaderBlock

MicHorizontalLineBlock

MicStartStopMarkupBlock

MicEnvironmentBlock

...

MicMetaDataBlock

MicSameStartStopMarkupBlock

MicCodeBlock

MicMathBlock

MicMathBlockExtensionForTest

MicMultilineComment

Imagine just a method that would have to have one condition for each of such cases!



# A message send is an open conditional

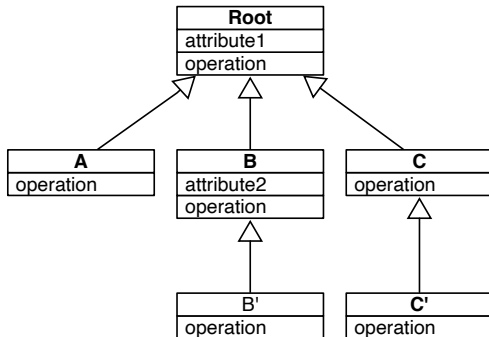
- Sending a message selects the right method
- It can be seen as a condition **without explicit ifs**
- The choice is dynamic
- It selects the method based on the receiver



# Select the right method



collectionOf do: [ :e |  
e operation ]

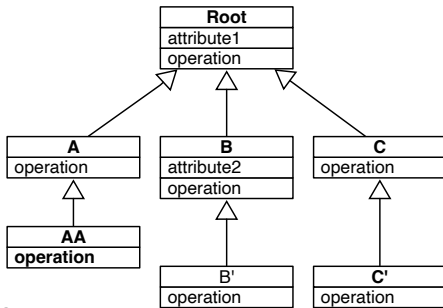


# But dynamically: new objects can be chosen



collectionOf do: [ :e |  
e operation ]

collectionOf add: AA new





# 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



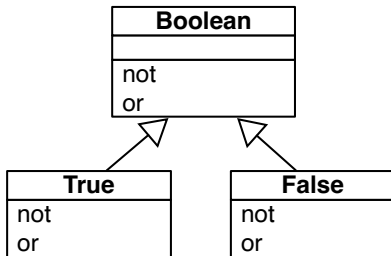
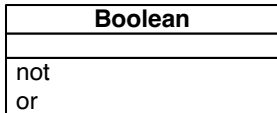
# How do we express choices?

- Ok we have a choice operation... then
- **How do we express choices?**



# How do we express choices?

Could we have the same solution with a **single** Boolean class?

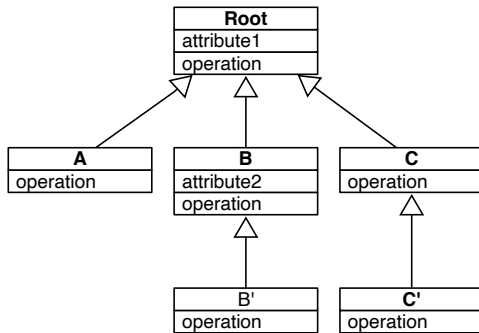
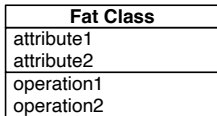


# Classes play case roles

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

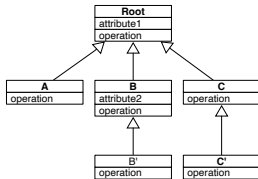
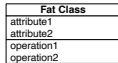


# One class vs. a hierarchy

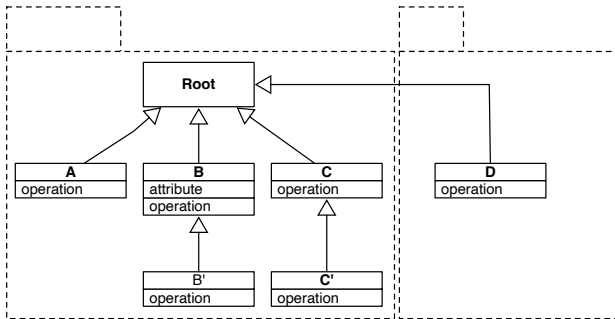


# Class hierarchy supports for dynamic dispatch

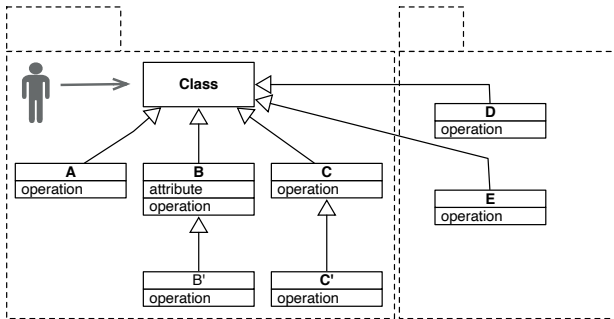
- More modular
- No need to recompile existing methods
- No need to introduce complex conditions
- An hierarchy provides a way to specialize behavior
- You only focus on one class at a time



# Message dispatch supports modularity



# Limit impact of changes





# Message sends are better than case statements

- Message sends are supporting **a choice**
- You could say: 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



# 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 substituted dynamically



# Avoid conditionals

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



# Summary: Cornerstone of OOP

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



A course by

S. Ducasse, G. Polito, and Pablo Tesone



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/>