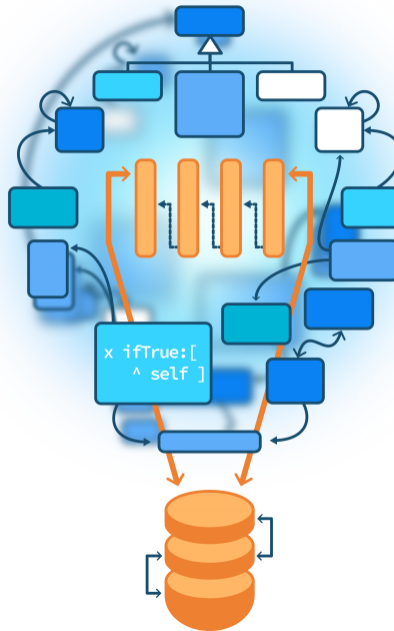
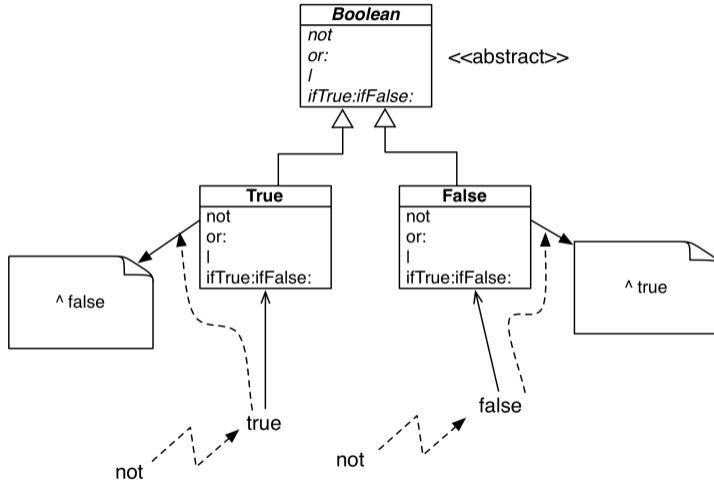


Essence of Dispatch

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



Remember: Implementing not in two methods



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 have one condition for each of such cases!



A message send is an open conditional

Sending a message

- selects the **right** method to execute based on the class of the receiver
- can be seen as a condition **without explicit ifs**
- is a dynamic choice

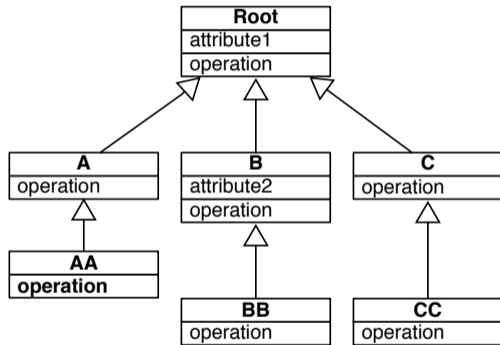


Select the right method

aCollection := {a.bb.c}.

...

aCollection do: [:e |
e operation]

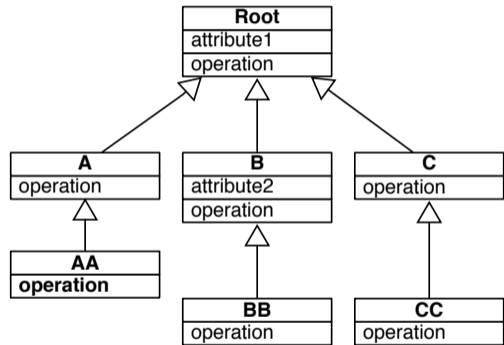


But dynamically: new objects can be chosen

aCollection := {a.bb.c.aa}.

...

aCollection do: [:e |
e operation]



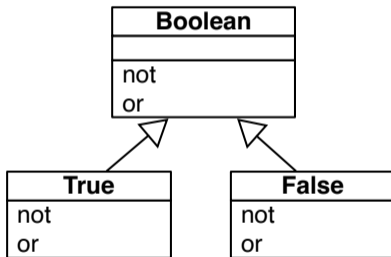
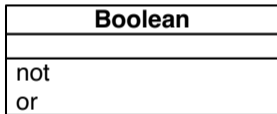
Sending a message is making a choice

- Message sending is a **choice** operator
- Each time you send a message, the execution engine **selects the right method** depending on the class of the receiver
- So, **How do we express choices?**



How do we express choices?

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

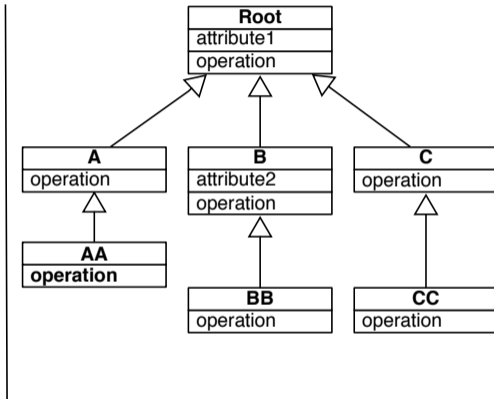
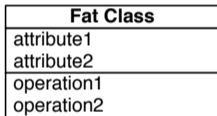


Classes play case roles

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

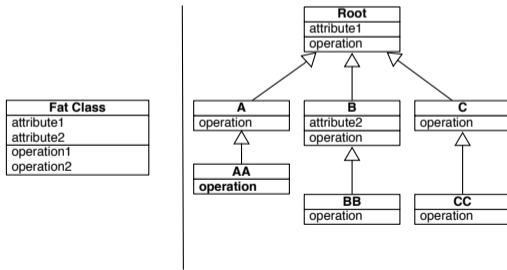


One class vs. a hierarchy

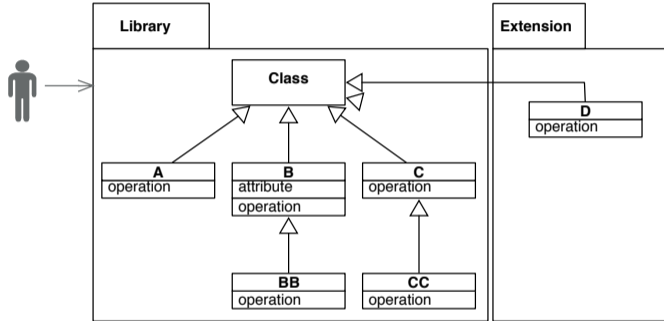


Class hierarchy supports for dynamic dispatch

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



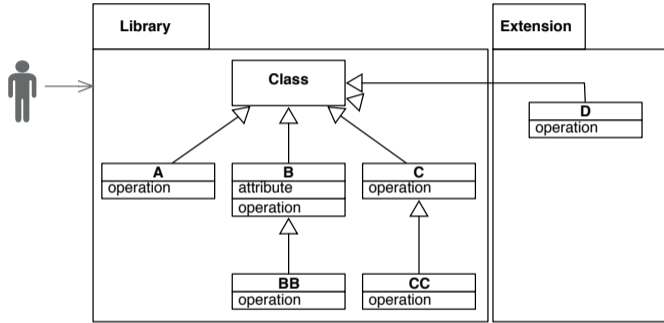
Message dispatch supports modularity



We can package different classes into different packages (better modularity)



Limit impact of changes



If a client receives instances of **D** (in addition to classes of first package), the client code does not have to change

Message Send is powerful

- Message sends are supporting **choices**
- The execution engine acts as a conditional switch: Use it!
- Classes act as "case statements"
- But with messages, the case statement is **extensible** by adding new classes without breaking client code



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



Summary: Cornerstone of OOP

- Avoid conditionals (cf. AntifCampaign)
- Use objects and messages whenever you can
- Let the receiver decide (Do not ask, tell)
- Class hierarchy supports for dynamic dispatch



Produced as part of the course on <http://www.fun-mooc.fr>

Advanced Object-Oriented Design and Development with Pharo

A course by

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



Inria
LearningLab



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/>