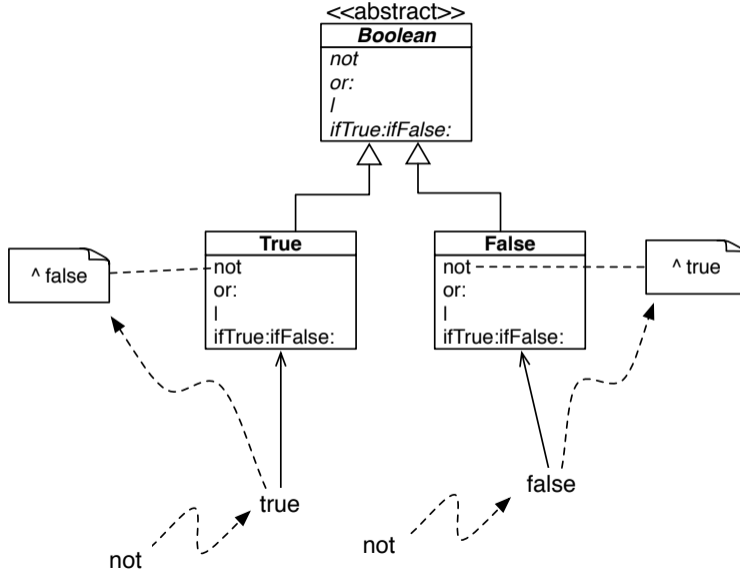# Essence of Dispatch

Let the receiver decide

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

Remember: Implementing not in two methods

What is the point?

- 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
    MicListBlock
        MicOrderedListBlock
        MicUnorderedListBlock
    MicParagraphBlock
        MacParagraphBlock
        MacRawParagraphBlock
    MicRootBlock
    MicSectionBlock

    MicSingleLineBlock
     MicAnchorBlock
     MicHeaderBlock
     MicHorizontalLineBlock
    MicStartStopMarkupBlock
     MicEnvironmentBlock

     ...
     MicMetaDataBlock
     MicSameStartStopMarkupBlock
        MicCodeBlock
        MicMathBlock
            MicMathBlockExtensionForTest
            MicMultilineComment

Imagine a method that has one condition for each of these 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**
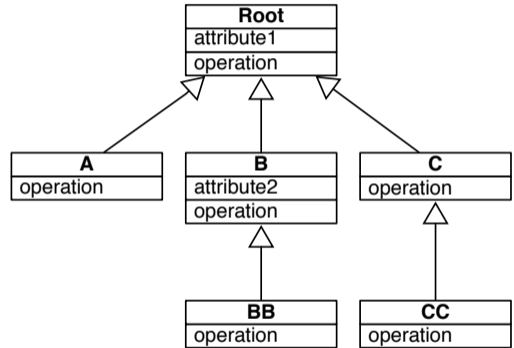- is a dynamic choice

Select the right method

```
aCollection := {a . bb . c}.
...
aCollection do: [ :e |
  e operation]
```
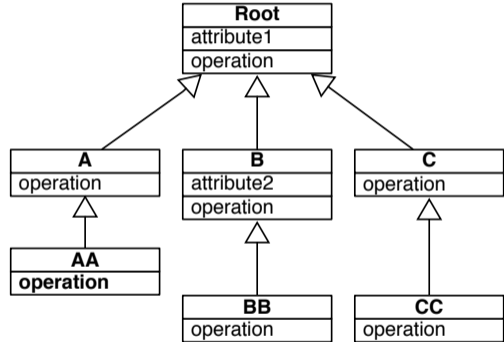
But dynamically: new objects can be chosen

```
aCollection := {a . bb . c . aa}.
...
aCollection do: [ :e |
   e operation]
```

```
          ┌─────────────────┐
          │      Root       │
          ├─────────────────┤
          │ attribute1      │
          ├─────────────────┤
          │ operation       │
          └─────────────────┘
```

Root
attribute1
operation

A		B		C
operation		attribute2		operation
		operation		

AA
operation

| BB		CC
operation		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, the next question is:
  - **How do we express choices**?

How do we express choices?

- Could we have the same solution for not with a **single** Boolean class?
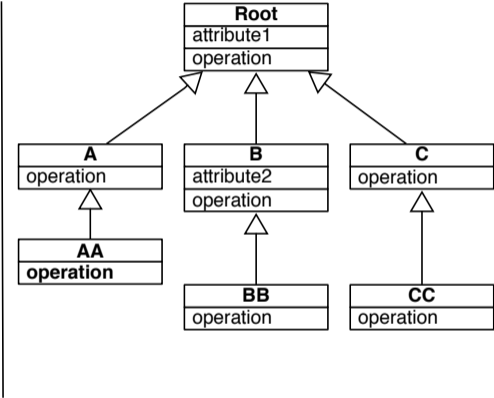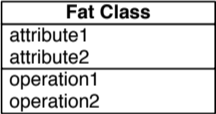- No! We would have conditionals in the not and or methods!
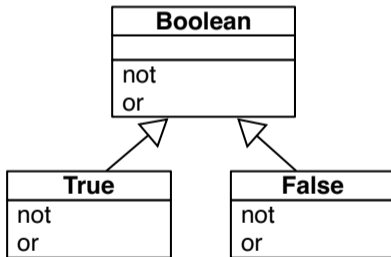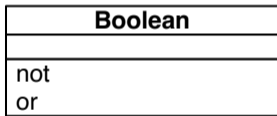
Classes play case distinct choices

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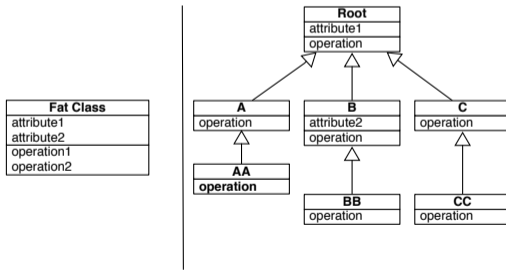- A **class** represents a choice (a case)
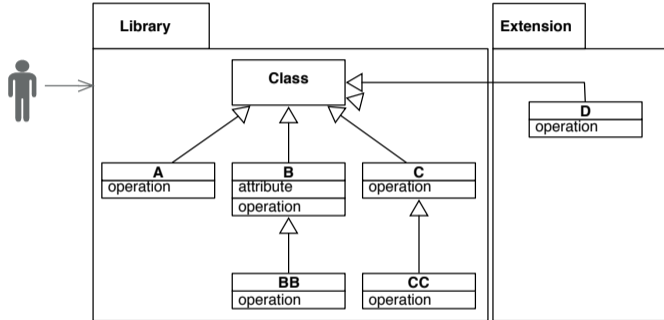
One class vs. a hierarchy

Class hierarchy supports dynamic dispatch

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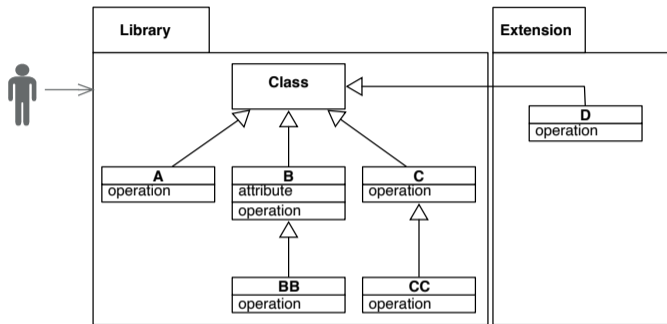- You only focus on one class at a time

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), its code does not have to change
- Method operation of D instances will be executed naturally

Message send is powerful

- Message sends are supporting **choices**
- The execution engine acts as a conditional switch: Use it!
- Classes act as "cases/choices"
- But with messages, the case statement is **extensible**:
  - 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: a cornerstone of OOP

- Avoid conditionals (see AntiIfCampaign)
- 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

2023