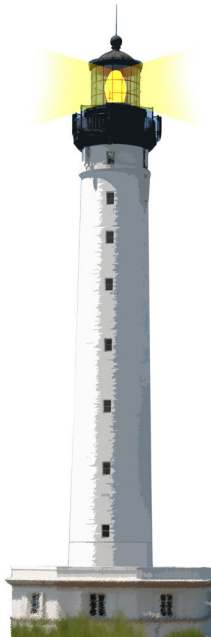
Message Sends are Plans for Reuse

S. Ducasse and L. Fabresse



Pharo

http://www.pharo.org

About this lecture

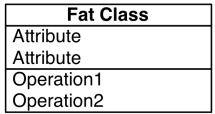
- Next step of the not implementation lecture
- Relevant to any object-oriented language
- Another essential aspect of OOP

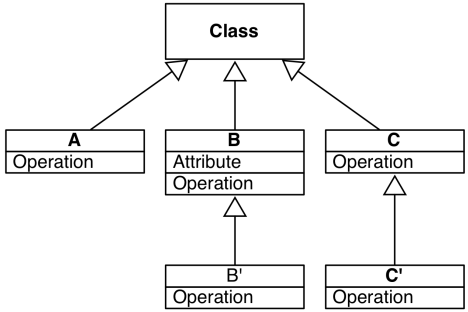
What you will learn

- Message sends are hooks for subclasses
- *"I like big methods because I can see all the code"* leads to bad design
- Why writing small methods is a sign of good design?

Remember: Sending a message makes a choice

- a message send leads to a choice
- a class hierarchy defines the choices
- self always represents the receiver
- method lookup starts in the class of the receiver





An example

Imagine

```
Node >> setWindowWithRatioForDisplay
  | defaultNodeSize |
  defaultNodeSize := mainCoordinate / maximizeViewRatio.
  self window add:
    (UINode new
      with: bandWidth * 55 / defaultWindowSize).
  previousNodeSize := defaultNodeSize.
```

We want to change the defaultNodeSize formula in a subclass

Duplication

Duplicate the code in a subclass

```
Node subclass: OurSpecificNode
  ...
```

```
OurSpecificNode >> setWindowWithRatioForDisplay
  | defaultNodeSize |
  defaultNodeSize :=
    (mainCoordinate / maximizeViewRatio) + 10.
  self window add:
    (UINode new
      with: bandWidth * 55 / defaultWindowSize).
  previousNodeSize := defaultNodeSize.
```

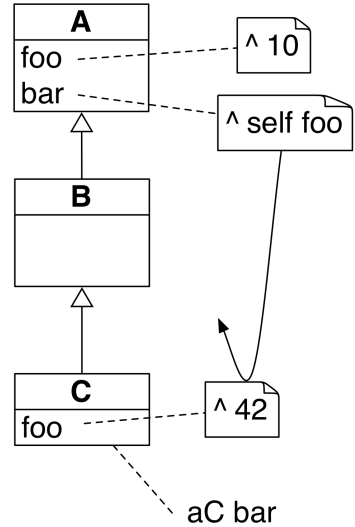
Avoid duplication

- in Java-like languages, using `private` attributes makes duplication in subclasses impossible
- duplication is not a good practice:
  - duplication copies bugs
  - changing one copy requires changing others

A much better solution

- Define small methods
- Send messages

Subclasses can override such methods



We can refactor this

```
Node >> setWindowWithRatioForDisplay
 | defaultNodeSize |
 defaultNodeSize := (mainCoordinate / maximizeViewRatio).
 self window add:
    (UINode new
       with: bandWidth * 55 / defaultWindowSize).
 previousNodeSize := defaultNodeSize.
```

Into ...

```
Node >> setWindowWithRatioForDisplay
  | defaultNodeSize |
  defaultNodeSize := self ratio.
  self window add:
    (UINode new
      with: bandWidth * 55 / defaultWindowSize).
  previousNodeSize := defaultNodeSize.

Node >> ratio
  ^ mainCoordinate / maximizeViewRatio
```

Subclasses reuse superclass logic

```
Node >> ratio
  ^ mainCoordinate / maximizeViewRatio
```

A subclass can refine the behavior

```
OurSpecificNode >> ratio
  ^ super ratio + 10
```

Another step

```
Node >> setWindowWithRatioForDisplay
  | defaultNodeSize |
  defaultNodeSize := self ratio.
  self window add:
    (UINode new
      with: bandWidth * 55 / defaultWindowSize).
  previousNodeSize := defaultNodeSize.
```

We can also extract the UINode instantiation.

Another step

```
Node >> setWindowWithRatioForDisplay
  | defaultNodeSize |
  defaultNodeSize := self ratio.
  self window add: self uiNode.
  previousNodeSize := defaultNodeSize.
```

```
Node >> uiNode
  ^ UINode new
    with: bandWidth * 55 / defaultWindowSize
```

Do not hardcode class use

```
Node >> uiNode
  ^ UINode new
    with: bandWidth * 55 / defaultWindowSize
```

Define methods returning classes

```
Node >> uiNode
  ^ self uiNodeClass new
    with: bandWidth * 55 / defaultWindowSize.
```

```
Node >> uiNodeClass
  ^ UINode
```

Many small messages

Small messages are a sign of good design, because:

- each little method is a potential hook for customization
- gives name to expressions

Some developers complain about all these small methods

- They try to understand code line by line
- This does not scale

Avoid magic numbers

```
Node >> uiNode
  ^ self uiNodeClass new
    with: bandWidth * 55 / defaultWindowSize.
```

Subclasses may want to change values

- do not hardcode magic numbers (55)

Use a message send

```
Node >> uiNode
  ^ self uiNodeClass new
    with: bandWidth * self averageRatio / defaultWindowSize.

Node >> averageRatio
  ^ 55
```

- This gives a name to a value
- Subclasses can override the value

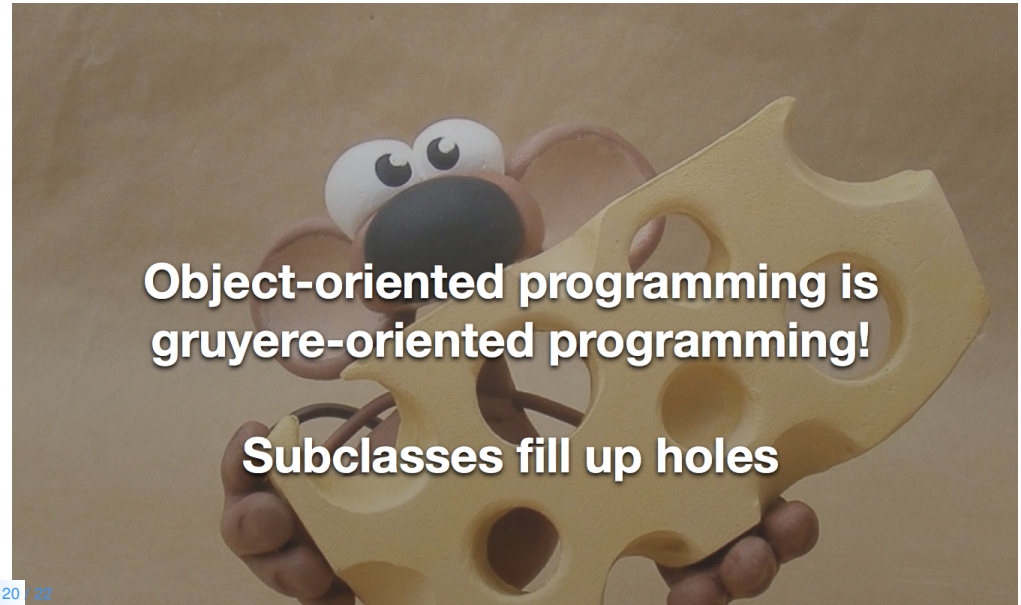
How to let the class users change the value?

Use an instance variable if needed

```
Node >> averageRatio
  ^ averageRatio ifNil: [ self defaultAverageRatio ]
Node >> defaultAverageRatio
  ^ 55
Node >> averageRatio: aNumber
  averageRatio := aNumber
```

- Subclasses can override the value
- Class users can set the value

Gruyere-oriented programming



Conclusion

- Code can be reused and refined in subclasses
- Sending a message in a class defines a **hook**:
  - i.e., a place where subclasses can **inject variations**
- Prefer small methods because:
  - this gives names to expressions
  - this gives freedom to subclasses

A course by

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