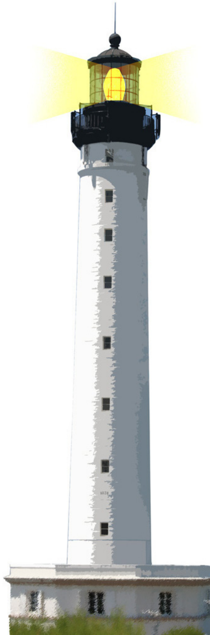
Advanced Object-Oriented Design

Message Sends are Plans for Reuse

S. Ducasse and L. Fabresse

http://www.pharo.org



About This Lecture

Another design lecture:

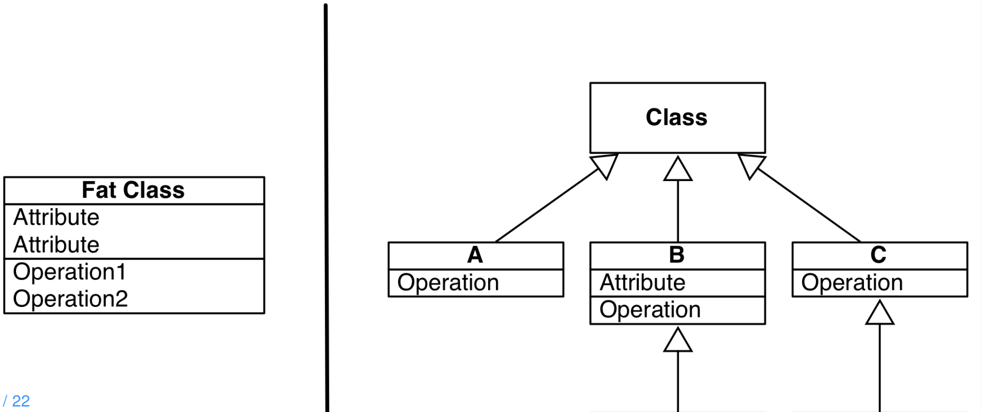
- Next step of the `not` implementation lecture
- Relevant to any object-oriented language
- May change your view on design

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

Sending A Message Leads to 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

```
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.
```

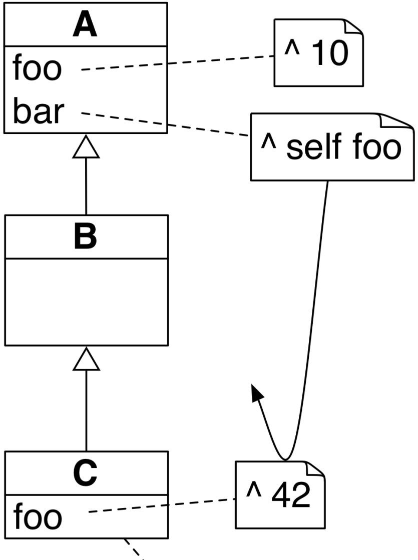
Avoid Duplication

- 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

Solution

- send messages
- define small methods

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.
```

Better Design

```
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

- Some developers complain about all these small methods
- They try to understand code line by line
- This does not scale

Small messages are a sign of good design

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

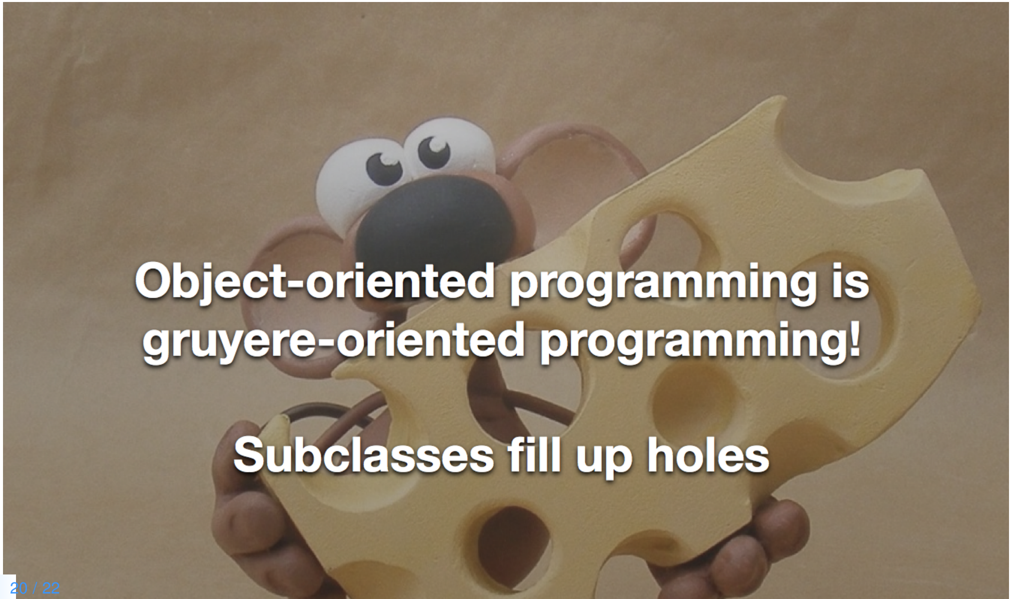
How to let the class users change the value?

Use an Instance Variable

```
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


Object-oriented programming is gruyere-oriented programming!
Subclasses fill up holes

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