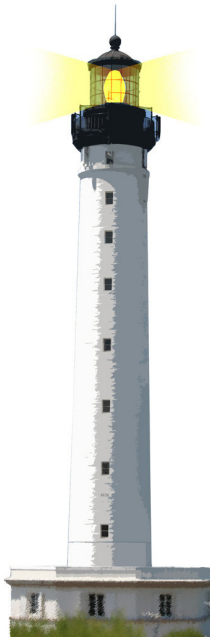
About magic literals

S. Ducasse and G. Polito



What you will learn

- Think about setters
- Think about customization

Remember

Imagine

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

How programmers can change 55 to 65?

Solution 1: introduce a variable

```
Object << Node
  slots: {percent};
  ...
```

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

Initialize and setter

```
Node >> initialize
  super initialize.
  percent := 55
```

Now how can the user change it?

```
Node >> percent: aZeroToHundred
  percent := aZeroToHundred
```

Now clients can decide

```
Node new percent: 65
Node new percent: 70
```

How can we make that we can have subclass encapsulating certain configurations?

Defining a hook

```
Node >> defaultPercent
  ^ 55
```

```
Node >> initialize
  super initialize
  percent := self defaultPercent.
```

Customizing a hook

```
MyNode >> defaultPercent
  ^ 65
```

- Subclasses can override the value
- Users can still set the value but can reuse the default value

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, G. Polito, and Pablo Tesone