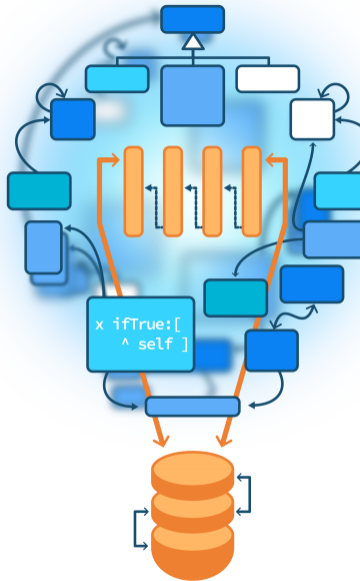# About magic literals

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

What you will learn

- Think about setters
- Think about customization

Case study

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

How programmers can change 55 to 65?

Introduce an instance variable

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

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

Initialize it and add a setter

Initialize the value:

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

and add a setter:

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

Now clients decide

Clients can customize this value:

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

But, how subclasses can encapsulate certain configurations (values)?

Defining a hook

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

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

Customizing a hook

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

- Subclasses can:
  - override the value (initialize)
  - override the default value (defaultPercent)
- Clients can:
  - set the value (percent:)
  - reuse the default value (defaultPercent)

Conclusion

- Magic numbers are specific values that may be constant
- Do not hide them in code
- Let clients customize these values (setters) if applicable
- Use hooks to define and reuse magic numbers (meaningful names)
- Use shared pools (a.k.a. enums) that can be shared among hierarchies

Advanced Object-Oriented Design and Development with Pharo

A course by
S.Ducasse, L. Fabresse, G. Polito, and P. Tesone

2023