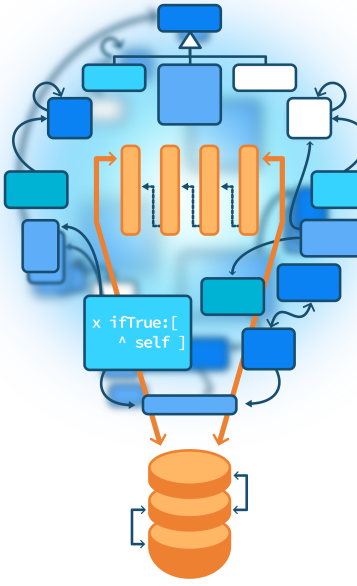
Flyweight

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



Goals

- Flyweight
- Symbols
- The case of color

Flyweight

Intent: Use sharing to support large numbers of fine-grained objects efficiently

Example: Symbol

- Ensure the uniqueness of symbols
- Reduce memory footprint
- Two symbols #unique are referring to the exact same object!

```
#unique == #unique
> true
```

Symbol creation

- At creation time check if there is not already a symbol object created for that surface syntax

```
Symbol class >> intern: aStringOrSymbol

  ^ (self findInterned: aStringOrSymbol) ifNil: [
    NewSymbols add: aStringOrSymbol createSymbol ]
```

Case Study: Color

UITheme

- creates literally thousands of color objects for nothing
- functional style

```
UITheme >> backgroundColor
  ^ Color white
```

```
UITheme >> textColor
  ^ Color black
```

A legitimate question

Should we turn Color into a flyweight?

- Cost of interning it
- Would a flyweight solve the spurious creation requests? No
- Do we need to create different colors or always the same?

When the domain should get into play

- Return colors without creating them endlessly
- A palette is a cache at the level of the domain

Palette limits

The case of implicit colors:

- Color red darker darker **vs** self selectedBackgroundColor
- Such pattern looks like a bad design practice

Conclusion

- Flyweight is useful to ensure uniqueness and limit memory footprint
- It does not avoid spurious object creation requests
- Better fix the cause than the consequences

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

Inria
LearningLab

2023