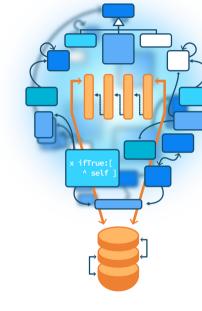
Advanced Object-Oriented Design

About global variables

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





Goals/Roadmap

- Understand that globals (but also Singleton) are not nice because globally shared
- Difficult to test
- See that globals may take different forms
- Study some cases
- Think modular
- Polymorphic dispatch requires instances of different classes

Autopsy of an error

```
... >> menu
...
icon: (Smalltalk icons iconNamed: #window)
...
```

- Smalltalk class namespace of Pharo
- Smalltalk icons refers to an icon manager

Case 1: Smalltalk icons

- Smalltalk icons acts as a global variable
- What if we want to have icons specific to an specific application
- We cannot have two icon sets used by widgets side by side at the same time to compare them

Case 2: A disguised global variable

Since in Pharo we can extend core libraries we could think this is any better.

```
MyApp >> menu
...
icon: #window aslcon
...
```

Here we extend Symbol class

```
Symbol >> aslcon
^ Smalltalk icons iconNamed: self
```

Case 2: A disguised global variable

```
MyApp >> menu
...
icon: #window aslcon
...
```

- Does not duplicate Smalltalk icons iconNamed:
- This is already something!
- But still a global

Case 2: A disguised global variable

- One global variable but disguised: only one place to edit but still fundamentally one global variable
- There is **only one** icon table
- We cannot dispatch to a different object (there is only one Symbol class)
- MyApp cannot extend or slightly change icons for my application only!
- I cannot simply have two icon sets at the same time to compare them

A better approach

^ self iconProvider at: aSymbol

```
MyApp >> menu
...
icon: (self iconNamed: #window)
...

MyAppSuperclass >> iconNamed: aSymbol
```

Why is this better?

- Modular
- Each receiver may do something different
- Each user may be configured differently
- Still we may share the common behavior

Case 3: asClass

Accessing programmatically a class is usually done as:

Smalltalk globals at: #Point

People wanted a shorter version for scripting

#Point asClass

Symbol >> asClass
^ Smalltalk globals at: self

• But there is a difference!

Case 3: asClass analysis

Same limits as before:

- Another global entry point
- What if we want to remotely access a class in another system
- We can only have one namespace
- We cannot inject a special namespace for test for example
- No way to dispatch to a different object

Case 3: Possible solution

Delegate to the class to get its environment

self class environment at: #Point

This supports different environments

Case 4: Smalltalk tools - The ugly

browseMethodFull

"Create and schedule a full Browser and then select the current class and message."

self currentClassOrMetaClass ifNotNil: [
Smalltalk tools browser
openOnClass: self currentClassOrMetaClass
selector: self currentMessageName]

Case 4: Smalltalk tools Analysis

browseMethodFull

"Create and schedule a full Browser and then select the current class and message."

```
self currentClassOrMetaClass ifNotNil: [
Smalltalk tools browser
openOnClass: self currentClassOrMetaClass
selector: self currentMessageName ]
```

- One global entry point
- Everybody refers to this single point!
- Yes this is called monolithic thinking
- Only one toolset possible at the same time (could be ok)

Case 4: Smalltalk tools possible solution

- Objects should refer to instance variables and messages
- Avoid direct reference to a global

MyApp >> initialize toolEnvironment := ToolEnvironment new

MyApp >> browseMethodFull self toolEnvironment browser openOnClass: self currentClassOrMetaClass selector: self currentMessageName

Points to consider

- With a global, when it changes, all its users are updated for free
- How to manage the fact that a tool may change?
- Browsers may register to a ToolEnvironment to be notified and update its instance

Conclusion

- Avoid global
- Think modular
- Give a chance to objects to specialize messages

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









Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France https://creativecommons.org/licenses/by-nc-nd/3.0/fr/