

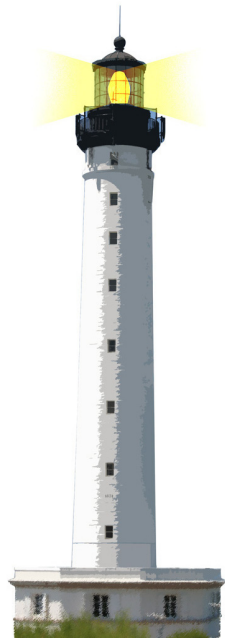
# Avoid Null Checks

Damien Cassou, Stéphane Ducasse and Luc Fabresse

W7S07



<http://www.pharo.org>



# Anti If Campaign

```
Main >> showHappiness: animal  
animal isDog  
  ifTrue: [ animal shakeTail ].  
animal isDuck  
  ifTrue: [ animal quack ].  
animal isCat: [ ... ].
```

Branching (with if) based on the type of an object is bad:

- adding a new type requires modifying all such code
- methods will become very long and full of details

Send messages instead



**Anti-IF Campaign**



# Anti If Campaign

```
Dog >> showHappiness  
  self shakeTail  
Duck >> showHappiness  
  self quack  
Cat >> showHappiness  
...
```

Branching (with if) based on the type of an object is bad

- adding a new type requires modifying all such code
- methods will become very long and full of details

Send messages instead



**Anti-IF Campaign**

# Do Not Return Nil

```
Inferencer >> rulesForFact: aFact  
  self noRule ifTrue: [ ^ nil ]  
  ^ self rulesAppliedTo: aFact
```

ifTrue: [ ^ nil ] forces every client to check for nil:

```
(inferencer rulesForFact: 'a')  
  ifNotNil: [ :rules |  
    rules do: [ :each | ... ]
```

# Return Polymorphic Objects

When possible, replace if by polymorphic objects:

- when returning a collection, return an empty one
- when returning a number, return 0

```
Inferencer >> rulesForFact: aFact  
self noRule ifTrue: [ ^ #() ]  
^ self rulesAppliedTo: aFact
```

Your clients can just iterate and manipulate the returned value

```
(inferencer rulesForFact: 'a')  
do: [:each | ... ]
```

# For Exceptional Cases, Use Exceptions

For exceptional cases, replace `nil` by exceptions:

- avoid error codes because they require `if` in clients
- exceptions may be handled by the client, or the client's client, or ...

```
FileStream >> nextPutAll: aByteArray  
  canWrite ifFalse: [ self cantWriteError ].
```

```
...
```

```
FileStream >> cantWriteError  
(CantWriteError file: file) signal
```

# Initialize Your Object State

Avoid nil checks by initializing your variables

- by default instance variables are initialized with nil

```
Archive >> initialize  
super initialize.  
members := OrderedCollection new
```

# Use Lazy Initialization if Necessary

You can defer initialization of a variable to its first use:

```
FreeTypeFont >> descent
^ cachedDescent ifNil: [
    cachedDescent := (self face descender * self pixelSize //
        self face unitsPerEm) negated ]
```



# Sometimes you have to check...

Sometimes you have to check before doing an action

- if you can, turn the default case into an object

```
ToolPalette >> nextAction  
self selectedTool  
  ifNotNil: [ :tool | tool attachHandles ]
```

```
ToolPalette >> previousAction  
self selectedTool  
  ifNotNil: [ :tool | tool detachHandles ]
```

# Use NullObject

```
NoTool >> attachHandles
```

```
  ^ self
```

```
NoTool >> detachHandles
```

```
  ^ self
```

```
ToolPalette >> initialize
```

```
  self selectedTool: NoTool new
```

```
ToolPalette >> nextAction
```

```
  self selectedTool attachHandles
```

```
ToolPalette >> previousAction
```

```
  self selectedTool detachHandles
```

- a null object proposes a polymorphic API and embeds default actions/values
- Woolf, Bobby (1998). "Null Object". In Pattern Languages of Program Design 3. Addison-Wesley.

# Conclusion

- A message acts as a better if
- Avoid null checks, return polymorphic objects instead
- Initialize your variables
- If you can, create objects representing default behavior



**Anti-IF Campaign**

A course by



and



in collaboration with



Inria 2016

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/>