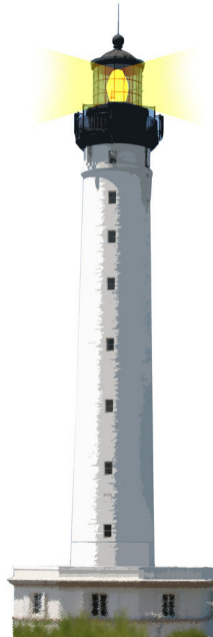


About defensive programming

S. Ducasse



Goal

- Thinking about spurious checks
- Dynamically-typed languages do not need explicit type checks
- Favor test



Preamble

```
Object >> assert: aBlock description: aStringOrBlock  
"Throw an assertion error if aBlock does not evaluates to true."  
<debuggerCompleteToSender>  
aBlock value  
  ifFalse: [ AssertionFailure signal: aStringOrBlock value ]
```

- assert:description: is checking and in addition raises and error.
- It changes the program execution



Defensive Example

```
BLLayoutCommonConstraints >> padding: aBIPadding
"Change element's margin to a BIMargin. aBIPadding must not be nil."
self
  assert: [ aBIPadding isNotNil ]
  description: [ 'Padding must not be nil' ].

padding := aBIPadding
```

Drawbacks of the approach

- Runtime cost
- Assertions can be optional so we should not consider that they are executed.
- Assertions can be a good help to track problems and stabilize



Defensive Example 2

```
BLLayoutCommonConstraints >> padding: aBIPadding
  "Change element's margin to aBIMargin. aBIPadding must not be nil."

  aBIPadding isNil
    ifTrue: [ 'Padding must not be nil' ].

  padding := aBIPadding
```

- What is the goal here? That padding does not break
- But I can still write `x padding: aJunkObject`
- So the test is not good and worth



Better setter

`BLLayoutCommonConstraints >> padding: aBIPadding`

"Change element's margin to a BLMargin. aBIPadding must not be nil."

`padding := aBIPadding`



Defensive Example 3

```
BlEvent >> source
```

```
"Return an event target that plays a role of a source of this event"
```

```
self
```

```
assert: [ self hasSource ]
```

```
description: [ 'Can not access a source if there is no one' ].
```

```
^ source
```

- Assertions are conceptually optional
- Tell look like leftover from debugging



Defensive Example Alternative 2

```
BLEvent >> source
```

```
"Return an event target that plays a role of a source of this event"
```

```
self hasSource
```

```
  ifFalse: [ self error: 'Can not access a source if there is no one' ].
```

```
  ^ source
```

- We could catch the error if needed.
- At least the reader knows that there is a check for real
- Now would be better to have a well initialized source



About explicit type checks

```
BLLayoutCommonConstraints >> padding: aBIPadding  
  "Change element's margin to a BLMargin. aBIPadding must not be nil."  
  
  (aBIPadding isKindOf: BIPadding)  
  ifTrue: [ self error ].  
  
  padding := aBIPadding
```

- It is slow
- It prevents to pass polymorphic objects



Conclusion

- Avoid optional checks that are only for debugging purpose
- Avoid explicit type-checks
- Favor tests



A course by

S. Ducasse, L. Fabresse, G. Polito, and Pablo 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/>