



Swift Custom Operators: The Good, the Bad and the Ugly

**III. BUONO,
IL BRUTTO,
IL CATTIVO.**



W About the Author: Mike Gerasymenko

- Working on iOS since 2008 (iOS 2.2)
- Around 35 apps for the AppStore
- With Wire since March 2014



W Preface: Terminology

- Operator describes **operation**, which is a calculation from zero or more input values (operands) to an output value.
- **Arity** of operators:
 - unary: $!x$, $x++$, $x--$, $x\sim$, $--x$, $++x$
 - binary: $x+y$, $x-y$
 - ternary: $x? a : b$
 - n-ary
- **Placement**
 - Prefix: $!x$
 - Infix: $x+y$
 - Postfix: $x++$

W Preface: Precedence and Associativity

- During the evaluation of the expression, first the operators with the higher **precedence** are being evaluated, for example:
 - $x = a + b * c$ is $x = a + (b * c)$
- Given that the precedence is equal for the operators, the operators are evaluated according to their **associativity**, left, right or non-associative. Having left associativity means that the operations are performed from left to right, for example:
 - $x = a + b + c$ is $x = (a + b) + c$

- **Overloading** means that the same operator can work with different types:
 - operator+ can be applied to Int, Double, ...
- **Short-circuit evaluation** means that operand is only calculated when necessary:
 - evaluation of false && obj.isTrue() would not invoke obj.isTrue()



il buono

W Game: spot the operator

```
27 var result = [Cat]()
28
29 for i in 0..<100 {
30     do {
31         let cat = try CatFactory.generateCat()
32
33         if cat is PallasCat {
34             let anotherCat = try CatFactory.generateCat()
35             if let anotherPallasCat = anotherCat as? PallasCat {
36                 debugPrint("Oh boy")
37             }
38         }
39         else if let aCat = cat {
40             result = result + [aCat]
41         }
42     }
43     catch let error as NSError {
44
45     }
46 }
47
```



goo.gl/FIPNQ0

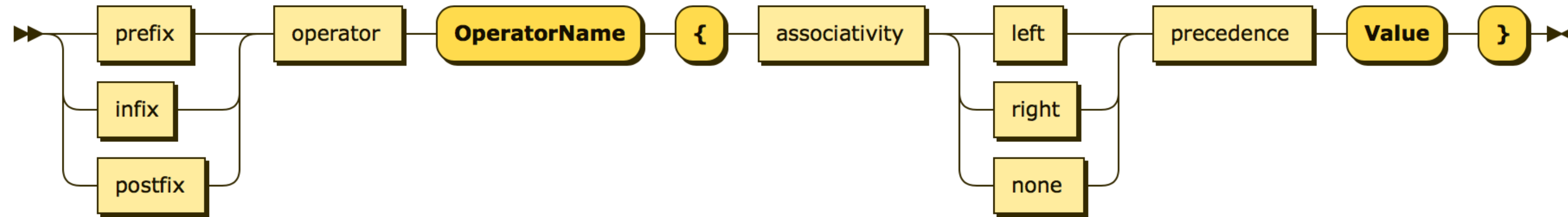
⌘ Those are also Operators

- postfix ?, !
- as, as?, as!, is
- ..., ..<
- in
- try, try?, try!

- Swift is really flexible with operator definition:
 - Operator definition consists of two parts: operator **declaration** and at least one **function** that gives the meaning to an operator
 - **Arity, placement, precedence and associativity** can be defined per operator
 - Rich **set** of operators that could be defined (standard operation characters + UTF8)
 - Swift gives the ability to overload **almost any existing operator** (except is, as, as?, =, ??, prefix &, postfix ?, ! and «? :»)

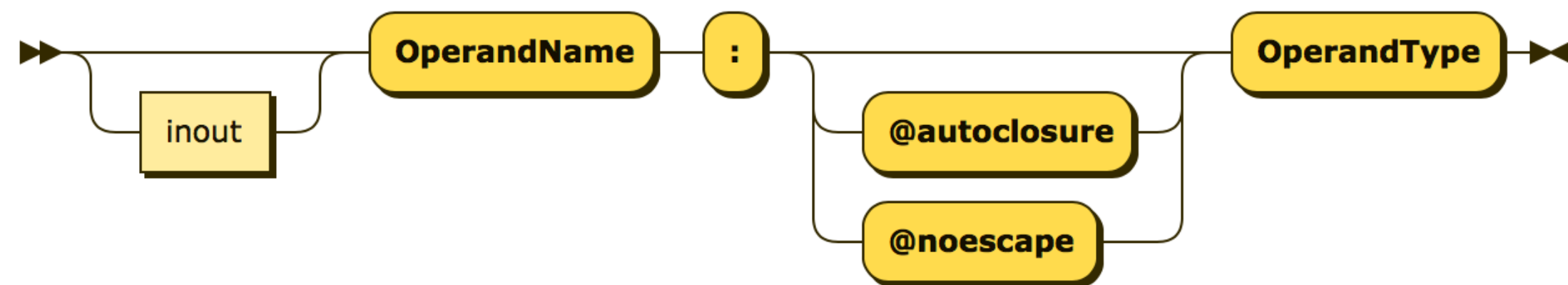
W The Good: Syntax: Operator Declaration

Operator:



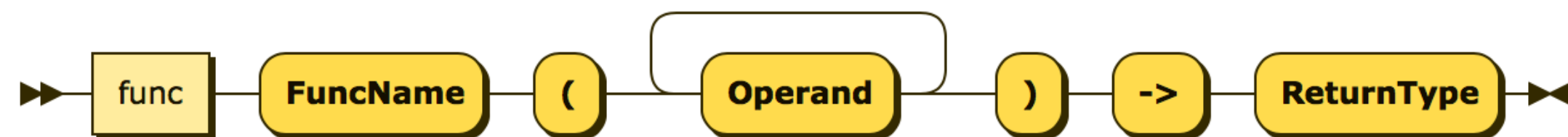
```
Operator ::= ( prefix | infix | postfix ) operator 'OperatorName' '{' associativity ( left | right | none ) precedence 'Value' '}'
```

Operand:



```
Operand ::= ( inout | ) 'OperandName' ':' ( '@autoclosure' | '@noescape' | ) 'OperandType'
```

Function:



```
Function ::= func 'FuncName' '(' 'Operand'+ ')' '->' 'ReturnType'
```

W The Good: Let's define an operator

- Playground time (page DefiningOperator)

W The Good: Being functional

- **Playground time (page BeingFunctional)**

W The Bad



W The Bad: Usefulness

- Why there's no such operator before?

W The Bad: Usefulness

- Playground time (page `AttributedString`)

W The Bad: Understanding of new operators

- New operator could be puzzling for other developers
- Overloading an existing operator could give a hard time debugging the code where it is not clear that new overloaded implementation is used

W The Bad: Global Scope

- Whatever you define, it would appear at the global scope.
- Operators from frameworks are visible in user code
- Being in global scope makes it easy to collide implementing same operator in the different way in framework and in user code


W The Ugly

il brutto



W The Ugly: Going Nuts

- **Playground time (page GoingNuts)**



**With great power
comes great responsibility.**



Curiosities

W Non-associative

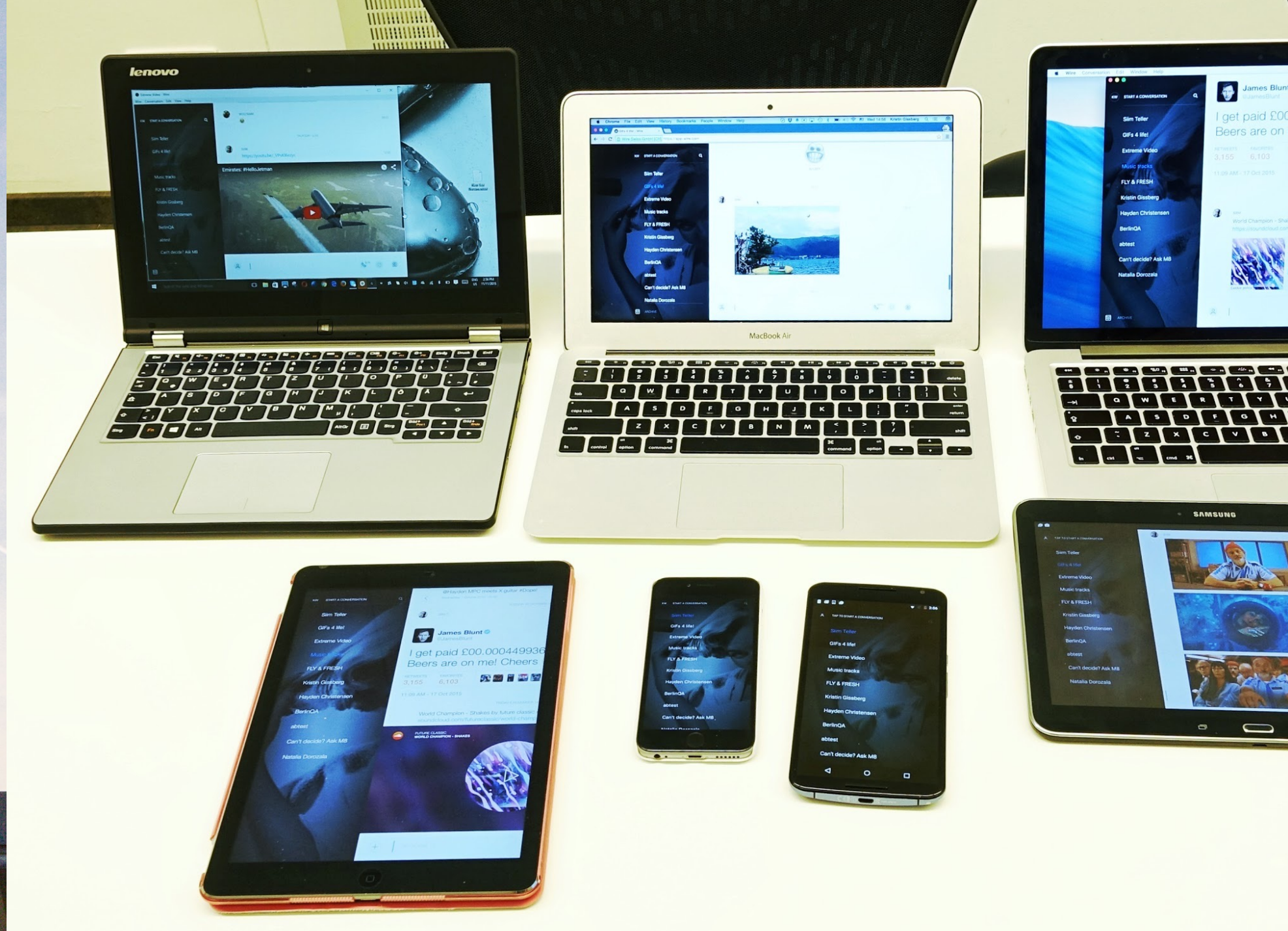
- operator== is **non-associative**, so it is harder to shoot your own foot:
- For example, $x == y == z$ in C would evaluate as $(x == y) == z$, and not to $x == y, y == z$

W Nice applications

- Libraries that are making use of custom operators
 - Cartography
 - Euler by mattt
 - Swift go
 - ...



Questions



W

Regards to my fellow colleagues



W References

- The Swift Programming Language https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/
- Facets of Swift, Part 5: Custom Operators <https://medium.com/swift-programming/facets-of-swift-part-5-custom-operators-1080bc78ccc#.l40cmmsy2>
- Functional Programming in Swift <http://five.agency/functional-programming-in-swift/>
- Railroad Diagram Generator <http://www.bottlecaps.de/rr/ui>
- 100 Most Popular Cat Names <https://www.cuteness.com/popular-cat-names>



Contact data

Mike Gerasymenko

`mihail@gerasimenko.me`

(find me on Wire using this email)

`mike@wire.com`

wire™