Swift or Objective-C-Which One is Better?

Mr. Ariz Ansari
Research Student
Department of Computer Engineering
College of Engineering, Bharati Vidyapeeth, Pune, India

Abstract

The introduction of a new programming language gives the necessity to differentiation its features with the existing programming languages to study the improvements that the new programming language offers over the existing ones. These studies can show the efficiency of the newly introduced programming language with comparison with the old. Also these studies can show the plus-points and drawbacks offered by the existing programming languages. If we compare the languages we can find out whether the language is offering improvements or relapses. In this research, comparison is made between the new programming language of Apple, Swift, with the main programming language of Apple before Swift, Objective-C. This research focusses on the characteristics of Swift programming language and differences between Swift and Objective-C.

Keywords: Programming, Functional programming, Object oriented programming, Software, Datatypes, Programming profession

I. INTRODUCTION

SWIFT is the new programming language created by Apple and it was presented to the public on the 9th of September, 2014 but developers could use it since June 6, 2014. It allows for developing applications for the new version of operating systems of Apple: iOS and OS X. The Apple's intention is to offer a new programming language easier, simpler, more-flexible, quicker, funnier and user-friendly to program than Objective-C to facilitate the applications development for platforms of Apple.

Swift was launched to offer an alternative to Objective-C because this has a syntax which barely evolved from it was created and has a great difference with other programming languages that have appeared in the latest years, because these have based on the C++ syntax. For this, Swift is implemented in new programming languages like C++11, C#, F#, Go, Haskell, Java, JavaScript, Python, Ruby, or Scala. Then his syntax is totally different than its predecessor. The Swift's syntax is more simplified because it does not use pointers and includes improvements in its data structures and in its syntax. As we will see, Swift has an easier syntax which helps to developers to have less mistakes and incorporates new functionalities and a new programming paradigm with in itself.

II. VERSIONS

Swift have had different versions with changes in it syntax and functionality since the release of the first version to developers appeared on June 6, 2014. In the present situation , Swift is in its third version, Swift 1.2.The first public version, Swift 1.0 GM, was presented released on June 06 2014. It was a Golden Master (GM) version because Apple announced that it will continue adding changes and improvements in the world of programming language. Swift 1.0 GM presented a lot of changes in its syntax, native libraries and the value type of some function, variable to use the new type "optional" and the syntax of some reserved words like arrays, dictionaries and open range operators.

Swift 1.1, the second version, released on October 22, 2014. This update added the "failures initializers", which changed some "protocols" and some internal functionalities of Swift. Swift 1.2 appeared on April 8, 2015 and was a major update. It arrived with the new version of Xcode 6.3 embedded in it. It introduced different improvements in the compiler: the compiler started to create incremental builds; better compilation velocity; improved the error and warnings message.

III. CHANGES WITH RESPECT TO OBJECTIVE-C

Swift and Objective-C use the same compiler, i.e the Low Level Virtual Machine (LLVM). LLVM was introduced in the late months of 2000, created for University of Illinois and it is programmed in C++. LLVM main focus is to transform the Swift (xcode) source code into an optimize native source code for input in elected hardware for their product oriented market such as Mac, iPhone, or iPad. Swift provides full compatibility with Objective-C and old projects because it allows the same use of libraries, primitive types, control flow and other functions that is included in Objective-C's library. However, Swift has a very diverse combination of libraries translated to Swift's native or mother code.

A. Pre-processor

Swift does not have pre-processor which is preinstalled as in C or Objective-C. To acquire the same functionality in Swift, users need to use constants instead of the simple macros which are defined as a constant variable or in the case of complex macros or functions definations.

B. Syntax

As in other programming languages (such JavaScript, Ruby), Swift grant the option of using the semicolon character (";") by the end of the line. Besides, Swift uses an access operator the called as the point character (".”). Like many other programming languages use the square brackets ("[", "]") or curly brackets ("{", "}") like in Objective-C. This allows more understandable code because it has a syntax more common to the most used programming languages. As well in the future context the contain changes regarding to the flow of structures syntax .The braces ("{" and "}") are used to enclose the scope so as to avoid programming problems. For example, in conditional flow structure (such as the keyworld "if"), the braces are used to keep the logic with the line of code, the first sentence is the structure will do when the condition will be true and the other sentences will execute in the other cases ("else"). If & else keyword work in pair like a "True or False" statement.

C. Variables

One of the major adjustment in Swift in comparison to Objective-C is the performance of the word "goto". Objective-C allows it to be used as this reserved word to "goto" any part of the current scope. On the other hand, Swift removed this specific reserved keyword and created the "Labeled" Statements for the same purpose. The "Labeled" Statements have a similar functionality as the "goto". But the of this "Labeled" keyword works on a much smaller scope than the "goto" in Objective-C. Exactly, they have the same scope as other programming languages like C#, Java, or PHP. Which allows for a "nested loop" or "switch case" in the programing language. Swift's, Boolean types are used in a great extent. Now they only exist as variables such as "true" or "false". In Objective-C it exists "true"/"false", 0/1, and "TRUE"/"FALSE".

D. Classes and Structures

In Swift, the syntax for creation of a class or structure is very similar as in C++, C#, or Java. Besides, Swift only uses one file (".swift") to define a class, contrary to Objective-C that uses two files (".h" and ".m"). For that, in Swift, you have to define all the class or structure in the same file.

```
// Class with inheritance
class Child: Person {

    override init(name:String){
        super.init(name: name)
    }

    override func description() -> String {
        return "I'm a child. My name is \(name)."
    }
}

var child = Child(name:"Cris Jr.");
child.name = "Cristian González García Jr.";
var descriptionJr = child.description();
```

Fig. 1: (Class showing Inheritance)

The Fig. 1 contains an example about a definition of a class in Swift. In the first part, it uses a default method "init", to define the constructor. The same words are kept as in Objective-C.

IV. LANGUAGE CHARACTERISTICS

Swift introduced a significant amount of changes as how to program and it has added new characteristics: it has added changes in variables, its datatypes and values, it has modified functions and methods to associate multiple return and diverse functionality in program and its characteristics. All this will be explained with more details in this section below.

A. Variables

Swift is more opposing characters than Objective-C because Swift has a strong typing to avoid insecure code. Swift forces to initialize the variables before their first use in the program. Moreover, you must too specify if the variable is a variable ("var") or a constant ("let") using these reserved word before the name. Besides, Swift checks are possible in arrays and integer overflow and auto-manages the memory stack using the Automatic Reference Counting (ARC).

```
// Variable
var variable:String = "I'm a variable"

// Constant
let constant:String = "I'm a constant"

// Static variable
var string:String = "Hello World Swift"

// Inflicted variable
var newString = "Hello World Swift"
```

Fig. 2: (Variable Example)

Swift allows the developer to determine the explicitly of the value type or let the compiler infers the type. Swift is strongly typed, for that, when you set a variable for the first time, the compiler assigns it to different types and you cannot assign a new value with different type later but you could do an explicit type conversion for changing the type of the value that you want to assign to the type of the variable.

B. Functions

Swift allows the programmer to send functions as parameter of other functions as well as to other scope operators. Also known as Lambda function, Anonymous function, Function literal or Lambda abstraction in Functional Programming. This is one of the aspects of the Functional programming that is embedded in Swift. The under mentioned example shows an overview how this function receives a function, it has multiple return; embedded in it to return values in a function.

```
func add(array:[Int]) -> Int{
    var aux = 0;
    for i in array{
        aux += i
    }
    return aux
}

func addWithFunction(array:[Int], f:([Int]) -> Int)
    -> (a:Int, b:Int){
    return (f(array), array.count)
}

var array = [Int](0...4)
var result = addWithFunction(array, add)
println(result.a)
println(result.b)
```


Fig. 3: (A function that receives a function and returns multiple parameters)

C. Classes

Classes integrates two changes as of the new regulation , one is a new type of constructor, the “Convenience Initializer” and the other change in destructors now known as “Deinitialization”. Now let’s talk about both this concepts.

D. Convenience Initializer

The “Convenience Initializer” is an optional constructor that, in case of it exists, is always called before the main constructor. Thus, Apple the creation of constructors is pretended to be done clearly and in an easy way because developers would use different Convenience Initializers as alternative constructors. So, normal constructors would be having the generic code to all possible cases. To create a “Convenience Initializer” it is sufficient to use the reserved word “convenience” before the constructor.

E. Deinitialization

Due that Swift incorporates ARC, destructors are not needed to release memory as occur in C, C++ and Objective-C but Swift incorporates “Deinitialization” .

A “Deinitialization” is an instance at which program is called immediately after the time it has released and it cannot be called explicitly. Using “Deinitialization”, developers have a mechanism to do a special cleaning of different resources or to do actions in case the object dies. For example when the object have to work with files.

```
class Fruit {
    var name: String
    var weight: Int

    init() {
        name = "Unknown"
        weight = 0
    }

    convenience init(name:String) {
        self.init()
        self.name = name
    }

    deinit {
        println("\(name) has disappeared")
    }
}

var fruit = Food()
var fruit1 = Food(name: "Apple")
var fruit2 = Food(weight: 50)
```

Fig. 4: (Class example with Convenience Initializers and a Deinit.)

F. Operators

Swift has all the unary operators such as (++, --,!), binary operators such as(+, -, *, /), ternary operators such as (a:b?c), logical operators such as (!, &&,||, true, false), and the assignment operators of C. Even, Apple added a Range of Operators, Overflow Operators and Custom operators to its programing list. We are going to explain these three operators in the next sections.

1) Range operators

The "Close-Range Operator" uses the ellipsis (from a…b) as we show in Fig. 5. It defines a range from "a" to "b" in which both the extreme's are included. It is used to repeat on a range where both limits are included.

```
// Closed Range Operator
var closedExample = [0...5]
for index in 1...5 {
    println("\(closedExample)")
}
```

Fig. 5: (Closed Range Operator example)

2) Overflow Operators

In Swift the arithmetic operators such as +, -, %,*, / do not have "overflow" functionally by default . In case that we want it, we need to add the "ampersand" sign (&) before the arithmetic operator as it can be seen in Fig. below. For example, to apply for overflow to the addition, we must use the combination: "&+". If we want to have underflow in case of subtraction, we have to use "&-". Swift also allows us to control the division by zero with the combination of "&/" and "&%".And in case of multiplication we can use the "&*".

```
var max = Int8.max
//var maxPlus = max + 1 // Error
var maxPlus = max &+ 1

var min = Int8.min
//var minxPlus = max - 1 // Error
var minPlus = max &- 1

//var div = 1 / 0 // Error
var div = 1 &/ 0
var div2 = 1 &% 0

var mul = 1 &* 0
```

Fig. 6: (Overflow Operators examples)

3) Custom operators

Swift allows the developer to define a new operators using the existing arithmetic signs for his own convenience. This is impossible in Objective-C while C++ affords this functionality to a very small extent.

To create a new "Custom Operator", we must need to declare its header and do his implementation. There are three ways to do a "Custom Operator". They are: prefix, infix, and postfix. The below example gives an overview for the above types.

```
prefix operator ++++ {}

prefix func ++++ (inout newValue: Int) -> Int {
    newValue += 4
    return newValue
}

var five = 5
var nine = ++++five
```

Fig. 7: (Custom Operator)

G. Variables

Swift has many new data variable types. They are: tuples and "optionals" . Some programming languages are already incorporated with some of these variable types in the compiler. Then, with this new data types, Swift offers a new way to work with more flexible and facilities for developers to work on.

1) Tuples

Tuples allow us to group together multiple values in a unique component for the execution. Therefore, it allows one of its uses is to return the value of multiple values in a function.

```
let contact = (13, "Cristian")
var office = contact.0
var name = contact.1
```

Fig. 8: (Tuples in Swift)

2) Optionals

The "Optional" is a new concept value type in Swift which neither exist in C nor Objective-C. It is used and described to assign a type or variable when the value could be of different type or nil values exist in the variable. In this aspect, if a conversion cannot be done in the system, the variable would take a "nil" value. To define an "optional" variable you must put a question mark (?) after the end of the type statement. In the example of the Fig. 9, the variable "convertedNumberImplicit" can never be "nil" and if in the case that the assignment of the value is "nil", the program will break & all the memory will deallocate. In the next variable, "convertedNumberExplicit", the type is an Optional Integer .In this case, the value could be "nil".

```
// Optional
let possibleNumber: String = "123"
let convertedNumberImplicit = possibleNumber.toInt()
let convertedNumberExplicit:Int? = possibleNumber.toInt()

let possibleNotNumber = "Hello Swift"
let convertedNotNumberImplicit = possibleNotNumber.toInt()
let convertedNotNumberExplicit:Int? = possibleNotNumber.toInt()
```

Fig. 9: (Example that assign value to Optional Variables)

This allows us to use an optional value in a conditional of overflow because the "nil" value is the same as a "false or the garbage" value. However, an exclamation mark (!) can for the optional to read the value if it is defined at the end of it. As shown in the figure below.

```
if convertedNumberImplicit != nil {
    println("integer value: \(convertedNumberImplicit!)")
} else {
    println("Could not be converted to an integer")
}
```

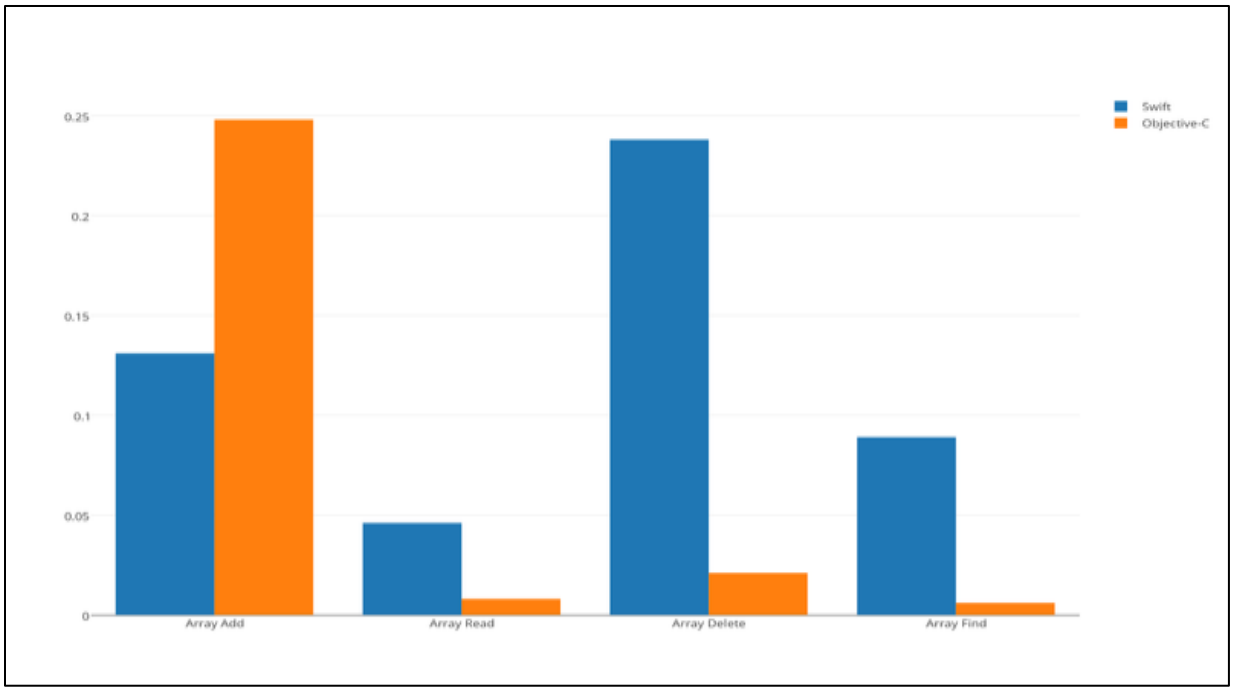
Fig. 10: (nil comparision)

H. Switch

In Swift the "Switch" keyword has improvements in relation with Objective-C. Now, you can use the "break" sentence as an optional keyword because the compiler will automatically breaks the "switch" when the "cases" are finished. Through this way, Apple helps in reducing the mistakes while programming. Due to this, it is next to impossible to have empty "cases" except in the program, but only if you use the reserved word "fallthrough". "Fallthrough" is a reserved word that does the execution continuously to the next "case". The same functionality is used by the "Switch" compiler.

V. PARAMETER COMPARISON BETWEEN SWIFT & OBJECTIVE C

Testing environment -
Tests were run for Simulator iPhone 5 iOS 8.3.
The result:



This shows the speed comparison between swift & objective c in the array operations such as add read delete and find.

VI. CONCLUSION

Apple has introduced a modern programming language called Swift with same of the best functionalities of many programming languages like C#, Java, JavaScript, PHP, Python and Ruby, among money other programming languages . Swift provides many more possibilities to developers and many new features for doing easier and more effective applications development because they have done change in the syntax, classes, variables, functions, operators, and data structures; they have improved the "Switch" statement and they have removed the concept of pointers as well.

Furthermore, by using Swift, developers need fewer characters to program the same code as compared to other programming languages because Swift has simplified the syntax but they need the same numbers of lines and words to program as other programming languages.

Overall, Apple has created a programming language with all the necessary functionalities of this age. The biggest advantage of introducing Swift is that Apple now has a programming language that improves their ecosystem due to the fact that Swift is not a competitor of Objective-C. In fact, Swift is a programming language that is capable of co-existing with Objective-C and it gives another possibility for development to developers.

One of its most upcoming features is what Apple calls Protocol-Oriented Programming (POP), one of the many models which will have a huge impact on software development in the future. In Swift, **protocols** (or "Interfaces" in languages such as objective-C) can be implemented by not only classes, but also enums and structs as well; furthermore, they provide an elegant and confusion-free alternative to multiple inheritance, as a single object can implement multiple protocols.

REFERENCE

[1] Ios apprentice : swift 2
[2] Objective-C Programmer's Reference by Carlos Oliveira (Author)
[3] Salunkhe, R. and Jaykumar, N., 2016, June. Query Bound Application Offloading: Approach Towards Increase Performance of Big Data Computing. In Journal of Emerging Technologies and Innovative Research (Vol. 3, No. 6 (June-2016)). JETIR.
[4] https://en.wikipedia.org/wiki/Swif(programming_language)
[5] Naveenkumar, J., Makwana, R., Joshi, S.D. and Thakore, D.M., 2015. Offloading Compression and Decompression Logic Closer to Video Files Using Remote Procedure Call. Journal Impact Factor, 6(3), pp.37-45.
[6] Parameter comparison https://yalantis.com/blog/is-swift-faster-than-objective-c/
[7] Stolk, J., Mann, I., Mohais, A. and Michalewicz, Z., 2013. Combining vehicle routing and packing for optimal delivery schedules of water tanks. OR Insight, 26(3), pp.167-190.
[8] Swift Programming by Matthew Mathias (Author), John Gallagher (Author)

[9] Jayakumar, D. T., and Raj Naveenkumar. "SDjoshi,"." International Journal of Advanced Research in Computer Science and Software Engineering," Int. J 2.9 (2012): 62-70.
[10] www.whoishostingthis.com › Resources
[11] https://developer.apple.com/library/ProgrammingWithObjectiveC/.
[12] https://www.swift.com/our-solutions/compliance-and-shared-services/swiftref
[13] https://transferwise.com/swift-codes/
[14] Naveenkumar, J., Makwana, R., Joshi, S.D. and Thakore, D.M., 2015. Performance Impact Analysis of Application Implemented on Active Storage Framework. International Journal, 5(2).
[15] https://www.swift.com/our-solutions/compliance-and-shared-services/swiftref
[16] https://www.jpmorgan.com/pdfdoc/fundstransfer/fieldtags/standard_text_mapping.pdf
[17] https://developer.apple.com/library/Swift/Swift/AboutTheLanguageReference
[18] Naveenkumar, J., Makwana, R., Joshi, S.D. and Thakore, D.M., 2015. Performance Impact Analysis of Application Implemented on Active Storage Framework. International Journal, 5(2).