SWIFT VS OBJECTIVE -C

Presented by -

Shipra Behera
Vandana Sridhar

WHAT IS SWIFT AND OBJECTIVE-C?

- **Objective -C :**
  - A general-purpose object oriented programming language that adds Smalltalk-style (dynamically typed language) messaging to C.
  - It was the main programming language supported by Apple for MacOS and iOS operating systems.
  - It provided support to Cocoa and Cocoa touch until the introduction of Swift. [1]

- **Swift :**
  - A general-purpose, multi paradigm, object oriented programming language developed by Apple for iOS, macOS, watchOS, tvOS and beyond.
  - Swift is designed to work with Apple's Cocoa and Cocoa touch frameworks and the large body of the existing Objective -C code. [2]

HISTORY OF OBJECTIVE-C

- Foundation of Apple's desktop operating system, and was also the basis for NEXTSTEP — OS X's immediate ancestor — created by Steve Jobs' NeXT Computer Inc.
- Objective-C was created primarily by Brad Cox and Tom Love in the early 1980s at their company Stepstone.
- They realized that a language like Smalltalk would be invaluable in building development environments for system developers. Cox began writing a preprocessor for C to add some of the abilities of Smalltalk.[1]
- In 1988, NeXT licensed Objective-C from StepStone (the new name of PPI, the owner of the Objective-C trademark) and extended the GCC compiler to support Objective-C.
- In 1995, Stepstone sold the total rights for Objective-C to NeXT, which then went to Apple when it acquired NeXT in late 1996.[6]
- Apple added a number of features to the Objective-C language, extending its functionality to parallel that of other languages that were beginning to arise. This major update was labeled Objective-C 2.0, and remains the language of choice for both OS X and iOS.

HISTORY OF SWIFT

- Chris Lattner began the development of Swift in the year 2010 and collaborated with other programmers at Apple.
- The language ideas for Swift were taken from Rust, Objective- C, Ruby, Haskell, C#, CLU and python.
- Apple initially released a prototype of the language and made it available to registered Apple developers.
- Apple also created a manual of 500 pages called the Swift Programming Language
- Swift 1.0 was released on 9th September 2014 and Swift 1.1 was introduced on 22nd October 2014.
- The company also introduced a rough framework for the development of Swift 3.0 and released several intermittent versions prior to 3.0 and this new version introduced new features in syntax.
- According to Stack Overflow Developer Survey 2015, Swift gained first stand as the Most Loved Programming Language. [2]

FEATURES INTRODUCED IN OBJECTIVE-C

- Add more object-oriented features to C. Derives its object syntax from Smalltalk.
- Classes are objects: Each class is an instance of a meta-class automatically created and managed by the run-time.
- Dynamic typing: an object can be sent a message that is not specified in its interface.
- Optional static typing.
- Automatic Reference Counting: combines best of manual memory management and automatic garbage collection..[1]
- Categories: can add methods to built-in classes without subclassing.
- Posing: permits a class to wholly replace another class within a program.
- Forwarding: Objective-C permits the sending of a message to an object that may not respond.
- Method Swizzling: at runtime you can swap out one implementation of a method with another

FEATURES INTRODUCED IN SWIFT

- Variable initialization : Variables are initialized before use
- Out of bound Errors : Arrays indices are checked for out of bound errors.
- Optional : Optional ensures that nil values are handled explicitly.
- Memory management: Done automatically
- Error Handling : Allows control recovery from unexpected failures
- Type Safe : It is a type safe language.
- Concise and fast iteration over a range or collection.
- Advanced control flow with do, guard, defer and repeat keywords.
- Tuples and multiple return values.
- Swift is managed as a collection of projects, each with its own repositories.
- Open source and easy to learn [3]

NEED FOR OBJECTIVE-C

- Everything in the iOS SDK has been built in Objective-C and works best with the Objective-C programming model in mind.
- Since Objective-C has been around much longer, third party libraries and frameworks (code written by other people/teams that you can use) are primarily in Objective-C as well
- To support older operating systems: iOS8 and below.
- To support old hardware & devices: Swift supports devices iPhone 4, iPad 2, 5th generation onwards.
- Every small update in Swift brings adjustments to paradigms (such as how to do type casting) that can be a little difficult to absorb.
- You can't do everything in Swift. I you want to use a library of C++ code in your application, you will need to talk to the C++ objects from Objective-C.

HOW SWIFT GAINS EDGE OVER OBJECTIVE-C

- Swift is faster.
- Safer
- More readable.
- Less code
- Less error prone
- Integrates with memory management.
- Open source language
- Interactive coding
- Closer to other platform
- Apple’s ongoing focus
- More mature, solves issues in early stages.
- More social networks are moving towards Swift
- Interoperable language - high compatibility with other languages.

NOVELTIES SWIFT INTRODUCED WITH RELATION TO OBJECTIVE-C

- Swift and Objective C use the same compiler ie LLVM (low level virtual machine).
- LLVM transforms swift source code in optimise native source code for the required hardware.
- Swift provides full compatibility with Objective C and old projects since it allows usage of same libraries, primitive types, control flow and other functions of Objective C.
- Swift has renovated the old syntax of objective C. This includes classes, protocols, control flows and variables.
- Swift to objective C interoperability includes Generics, Tuples, enumerations, structures, global variables, type aliases, nested types and curried function. [4]

COMPARATIVE STUDY - TOPIC LIST

1) Pre Processor availability
2) Syntax of Variables
3) Execution Time
4) Functional Programming
5) Code Maintenance
6) Readability
7) Control flow
8) Memory management
9) Safety [4] [5]

PREPROCESSOR

Objective-C	Swift
● Preprocessor is not a part of the Objective-C compiler, but is a separate step in the compilation process. ● Simply a text substitution tool which instructs compiler to do the required pre-processing before the actual compilation.	● Doesn’t have a preprocessor. ● Users must use constants instead of macros. ● Users can define constant variables. ● In the case of complex macros, define a function

SYNTAX OF VARIABLES

Objective-C	Swift
• "goto" - uses this reserved word to go to any part of the current scope. • To identify where in the program the branch is to be made, a label is needed.	**Labeled Statements:** • Instead of using the reserved word "goto", swift used labeled statements. • Have the same functionality of "goto" but operate in a smaller scope.
• "YES/NO", "true"/"false", 0/1, and "TRUE"/"FALSE".	**Boolean type:** • Simplified boolean type • Uses variables such as "true"/"false"
• All properties are, in a sense, computed. • When a property is accessed through dot notation, the call is translated into a getter or setter method invocation. • Can cause side effects.	**Property Observers:** • Adds observers to variables using two reserved words 'willset' and 'didset'. • Willset - before the value is assigned to the variable • Didset- called after value is assigned to variable

CLASSES & STRUCTURES

Objective-C	Swift
• Convention is to have two files containing the interface and the implementation. • Uses the default method “init” to define the constructor. • Accesses the properties of objects using (.) dot • Has blocks that uses special syntax to create closures. • Methods can be overridden, although it doesn't have any way to annotate a method as being an explicit override of a base class method.	• Syntax to create a create is very similar to C++, C# and Java. • Need to define all the classes and structures in the same file in a .swift file. • Uses the default method “init” to define the constructor. • To access an object, needs to use reserved word “self” instead of “->” • Accesses the properties of objects using (.) dot • Doesn’t have pointers and memory blocks needn’t be allocated. • Uses keywords such as “override “ and “super” for overriding and inheritance.

EXECUTION TIME

Objective-C	Swift
• Guarantees every method to be dynamically dispatched. There is no static dispatch involved, which makes it impossible to optimize an Objective-C program further. • Complete code is built when you compile after making any changes to the code.	• Swift doesn't guarantee dynamic dispatch (dynamic dispatch - process of selecting which implementation of a polymorphic operation to call at run time) • Source files that haven't been changed will no longer be recompiled by default thereby improving build time. • Larger structural changes to code may still require multiple files to be rebuilt. • Swift is 2-3 times faster than Objective-C

FUNCTIONAL PROGRAMMING

Objective-C	Swift
• Borrows some functional aspects using *blocks*. • Supports some functional operators like filter, map, reduce, forEach..[4]	• For mathematical operations such as sorting, array filter etc, swift provides an easier way with the help of a single line of code. • Makes concurrency and parallel processing easier to work • Uses a function as a parameter while calling other functions. [8]

CODE MAINTENANCE

Objective-C	Swift
• Complex code structure since it's built on the C language. • Maintain two separate files of code in Objective-C to improve the efficiency and the developing time of an application.	• Code closely resembles the English language. • Codes are easy to maintain • Drops the two-file requirement as in objective C(.m & .h) • Combined into a single code .swift file. • Xcode and LLVM compiler work behind the scene to reduce the workload of the programmer.

READABILITY

Objective-C	Swift
• @ symbol is introduced in Objective-C as a new keyword. • Enables the "Objective-C" part of the language to freely intermix with the C or C++ part. • Verbose when it comes to string manipulation. • Over thirty years old, and that means it has a more clunky syntax.	• Swift isn't built on C. • It unifies all keywords • Removes @ symbol in front of every objective C keyword. • Improves readability and errors can be prevented easily.

CONTROL FLOW STATEMENTS

Objective-C	Swift

• Control flow statements have similar syntax with C and C++.
• Use of curly braces to delimit code blocks is not necessary unlike Swift.
 	
• The infamous iOS “goto fail” error could have been avoided if Objective-C enforced curly braces.[7]
 	
`if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)`
`goto fail;`
`goto fail;`
	• Syntax differences of control flow statements are easy to identify and understand
	• Switch statement is more powerful than objective C.
	• A value passed to the switch statement is compared against matching patterns.
	• Switch statements don’t include “break” statements.
	• The “if let” statements includes boolean conditions, unwraps multiple optionals at a time.
	• Can express conditions without unnecessary nesting [9]

MEMORY MANAGEMENT

Objective-C	Swift
• In more recent versions of macOS and iOS, garbage collection has been deprecated in favor of Automatic Reference Counting (ARC), introduced in 2011.[1] • Compiler inserts retain and release calls automatically into Objective-C code based on static code analysis. • Objective-C uses the ARC supported within the Cocoa API.	• Programmer doesn't have to worry about memory for every object. • Memory management is handled by the ARC(automatic reference counting). • ARC in swift works across both object oriented and procedural codes. • Requires no conceptual switching of contexts even when using low level APIs

SAFETY

Objective-C	Swift
• Nothing happens when you call a method with a nil pointer nothing happens. • Then expressions and a line of code become a no-operation (NOP). • It doesn't cause a crash, but NOP causes unpredictable results that complicate the process of finding and fixing bugs.	• Swift is safer • The possible of nil optional value is made clear by optional types, which may generate a compile error. • This gives a short feedback loop and allow programmers to code with intention. • Problems can be fixed as code is written. • Reduces time and money on fixing bugs that are related to pointer issues.

OBJECT PRINCIPLES IN OBJECTIVE-C AND SWIFT

- Classes/ objects
- Properties and Methods
- Access Control
- Encapsulation
- Abstraction
- Inheritance
- Method overloading
- Method Overriding
- Polymorphism

CLASSES & OBJECTS

Objective-C	Swift

Class:
`@interface Person: NSObject {`
`  //class members;`
`}`
`//class methods`
`@end`
`@implementation Person`
`  //ClassMethods`
`@end`
	Class:
	`Class person {`
	`// attributes and functions`
	`}`

Object: `Person *person1 = [[Person alloc] init]; // allocates a memory instance and initializes it`	Object: `let man = person() // created an object of a person`

PROPERTIES AND METHODS

Objective-C	Swift

`@interface Person : NSObject {`
`  @property int age;`
`  @property NSString *name;`
`}`
`-(id)initWithNameAndAge: (NSString *)name`
`  age:(int) age;`
`@end`
	Properties/Attributes:
	`Class Person {`
	`var age = Int!`
	`var name = String!`
	`Init age(age:Int, name: String) {`
	`}`
	`}`

`@implementation Person`
`-(void) play: (NSString *) sport {`
`//function`
`}`
`@end`
	`func play (sport: String)`
	`{`
	`// mention how the person instance will play the sport`
	
	`}`

ACCESS CONTROL

Objective-C	Swift
• Public: no access restrictions. • Private: accessible within the defining class. • Protected: accessible within the defining class and by derived classes (default). • Can also add property attributes such as readonly, atomic, readwrite, etc.	• Swift provides several control levels which makes it handy to write code • Open access and public access : Entities with this access level can be accessed within the module that they are defined as well as outside their module. • Internal access: Entities with this access level can be accessed by any files within the same module but not outside of it. • File - private access - Entities with this access level can only be accessed within the defining source file. • Private access - Entities with this access level can be accessed only within the defining enclosure.

ENCAPSULATION

Objective-C

```
@interface Maths: NSObject {
 int a;
 int b;
 private:
  int result;
 -(id) initWithAandB: (int) a andB: (int) b;
 -(void) add;
 -(void) displayResult;
}

@implementation Maths {
 -(id) initWithAandB: (int) a andB: (int) b {
  self = [super init];
  if (self) {
   self.a = a;
   self.b = b;
  }
  return self;
 }
 -(void) add {
  result = a + b;
 }
 -(void) displayResult {
  NSLOG(result);
 }
}
```

Swift

```
Class Maths{
     let a: Int!
     Let b: Int!
     Private var result : Int!

     init(a: Int, b: Int) {
        Self.a = a
        Self.b = b }
     func add() {
     result = a+b }

     func displayResult() {
     print(" Result - \(result)") } }

     Let calculation = Maths(a :2, b:3)
     calculation.add()
     calculation.display()
```

In this example, the variable result is encapsulated using the access specifier “private” [10]

ABSTRACTION

Objective-C

```
@interface Maths: NSObject {
 int a;
 int b;
 private:
  int result;
 -(id) initWithAandB: (int) a andB: (int) b;
 -(void) add;
 -(void) displayResult;
}

@implementation Maths {
 -(id) initWithAandB: (int) a andB: (int) b {
  self = [super init];
  if (self) {
   self.a = a;
   self.b = b;
  }
  return self;
 }
 -(void) add {
  result = a + b;
 }
 -(void) displayResult {
  NSLOG(result);
 }
} 11
```

Swift

```
Class Maths{
let a: Int!
Let b: Int!
Private var result : Int!

init(a: Int, b: Int) {
       Self.a = a
       Self.b = b }
func add() {
result = a+b }

func displayResult() {
print(" Result - \(result)") } }

Let calculation = Maths(a :2,
 b:3)
calculation.add()
calculation.display()
```

Exposing only relevant data and methods of the object and hiding their internal implementation.

In this example, we are exposing displayResult() and add() method to the user to perform the calculations, but hiding the internal calculations.

INHERITANCE

Objective-C

```
@interface Person : NSObject {
   NSString *personName;
   NSInteger personAge;
}
- (id)initWithName:(NSString *)name
andAge:(NSInteger)age;
@end
@interface Employee : Person {
}
- (id)initWithName:(NSString *)name
andAge:(NSInteger)age
  andEducation:(NSString *)education;
@end
```

- All classes in Objective-C is derived from the superclass NSObject.
- Employee derives from Person and accesses all properties of Person.

Swift

```
Class Person {
var age = Int!
var name = String!
Init age(age:Int, name: String) {
} }
class Men: Person{  }

let andy = Men(age: 20, name= "Matt")
print (andy.age)
```

- Declared the derived class Men which inherits from parent class Person. Men accesses all of the properties of Person.

METHOD OVERLOADING

Objective-C

- Method overloading does not exist in the same way as it does in other programming languages.[8]
- Need to change the *number of parameters* or the *name* of (at least) one parameter that each method accepts.
- Method name includes the method signature keywords (the parameter names that come before the ":"s), so the following are two *different* methods, even though they both begin "writeToFile"

```
-(void) writeToFile:(NSString *)path
fromInt:(int)anInt;

-(void) writeToFile:(NSString *)path
fromString:(NSString *)aString;
```

- The names of the two methods are "writeToFile:fromInt:" and "writeToFile:fromString:")

Swift

Method overloading is the process where the class has two methods with the same name but different parameters.

```
func play (sport: String)
 {
// mention how the person instance with play
the sport
}
func play (instrument : String)
{
// mention how the person instance with play
the instrument
}
```

Now there are two play functions where the person
Plays two different types of activities.

METHOD OVERRIDING

Objective-C	Swift

`@implementation MyClass: NSObject {`
`}`
`-(int) myNumber {`
`return 1;`
`}`
`@end`
 	
`@interface MySubClass: MyClass {`
`}`
`-(int) myNumber;`
`@end`
 	
`@implementation MySubClass: NSObject {`
`}`
`-(int) myNumber {`
`return 2;`
`}`
`@end`
	Overriding is the process by which two methods have the same method name and parameters. One of the methods is in the base class and the other is in the derived class
	
	`class Men {`
	`Override init(age : Int , gender : String, color : String, maritalStatus : String ) {`
	`// initialize men`
	`}`
	`}`
	
	Since class Men inherits from Person , the init() method in the Men class overrides the one from the Person class. This init() method behaves differently for objects of Men class.

POLYMORPHISM

Objective-C

```
@interface Shape : NSObject {
   CGFloat area;
}
- (void)calculateArea;
@end
@interface Square : Shape {
   CGFloat length;
}
- (id)initWithSide:(CGFloat)side;
- (void)calculateArea;
@end
@interface Rectangle : Shape {
 CGFloat length; CGFloat breadth;
}
- (id)initWithLength:(CGFloat)rLength
andBreadth:(CGFloat)rBreadth;
- (void)calculateArea;
@end
```

- Shape and Rectangle derive from Base class Shape and provide different implementations of function calculateArea.

Swift

Objects of the same class can behave independently within the same interface.

```
class Men : Person  {
Override init(age : Int , gender : String,
color : String, maritalStatus : String ) {
        Gender = "male"
} }
class Football {
  let team: [Men]!
  init(team: [Men]) {
              Self.team = team}
Func play() {
for player in team{
// define each player based on his ability
} } }
```

- Here a football class is created and a men's team instance is created. We iterate through the team to check and configure each player to play according to their abilities.[12]

SUMMARY & CONCLUSION

We saw several comparisons and properties of Swift and Objective C such as:

- The importance and history of both languages.
- Features introduced in both.
- A comparative study based on several properties.
- Object oriented principles and how they are constructed.

- All in all, both languages have their advantages and disadvantages. Apple seems to be pushing Swift as its primary language of choice for iOS application development. This doesn't mean that objective C will become obsolete anytime soon. [13]
- Several larger companies seem to be using objective C heavily because it provides specialized knowledge .
- Both Swift and objective C are intuitive user friendly languages created by the trillion dollar brand and they will continue to dominate the mobile application space.

REFERENCES

Websites:

[1] https://en.wikipedia.org/wiki/Objective-C

[2] https://en.wikipedia.org/wiki/Swift_(programming_language)

[3] https://www.geeksforgeeks.org/introduction-to-swift-programming/

[4]https://www.altexsoft.com/blog/engineering/swift-vs-objective-c-out-with-the-old-in-with-the-new/

[5]https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-swift-programming-language/

[6]https://medium.com/chmcore/a-short-history-of-objective-c-aff9d2bde8dd

[7]https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/

[8] https://www.oreilly.com/library/view/ios-6-programming/9781449342746/ch01s15.html

REFERENCES

Publications:

[9] CG Garcia, JP Espada, BCPG Bustelo, JMC Lovelle, (2015), Swift vs. Objective- C , A New Programming Language, IJIMAI, ISSN-e 1989-1660, Vol. 3, Nº. 3, 2015

[10] M Reboucas, G Pinto, F Ebert,W Torres, A Serebrenik, F Castor, 2016, An empirical study on the usage of the swift programming language, IEEE 23rd international conference on software analysis, evolution and reengineering 1, 634-638

[11] B.J Cox ,1988, The objective-C environment : past, present, and future,Digest of Papers. COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference,10.1109/CMPCON.1988.4852

[12] C. Eidhof, F. Kugler, W.Swierstra, 2014, Functional Programming in Swift, ACM digital library, ISBN:3000480056 9783000480058.

[13] C. Adamson, B. Dudney, 2012, iOS SDK Development, ACM digital library, Pragmatic bookshelf, ISBN:1934356948 9781934356944