Background Report : Wake Up Swifty (Alarm Clock Project in Swift)

RICHARD CHIANG, z6w8@ugrad.cs.ubc.ca
MICHELLE CHEN, i5c9@ugrad.cs.ubc.ca
HEE SU KIM, k0b0b@ugrad.cs.ubc.ca
SHU (OLIVER) ZHANG, d5m8@ugrad.cs.ubc.ca

0. Overview of Project Topic

Apple has been developing in iOS with Objective-C since the acquisition of NeXT in 1996 ("Cocoa Core Competencies: Objective-C", 2015). Due to the major influence of the C programming language on Objective-C, it suffers many disadvantages and suffers from aesthetically unpleasant coding syntax. We believe the emergence of Swift marks a significant milestone in iOS development.

Since we will be completing Project Type 4 up to the 100% commitment level, we plan to build an iOS alarm clock with the Swift programming language. To turn off the alarm, the user will need to physically shake their phone, or perform other physically/mentally challenging tasks. These tasks will help the user to wake up swiftly. We believe that will be a substantial program since it will allow us to utilize a variety of different features that are offered in Swift, such as: guards, multiple return types, closures, OOP semantics, and optional types. Although Swift is not necessarily esoteric, most developers in industry still use Objective-C when developing iOS applications and the language itself is fairly new (it was only introduced three years ago at the 2014 WWDC). Moreover, no one in the group has studied or programmed in Swift before.

Why Swift?
We are using Swift to implement this application because we want to learn more about the Swift programming language, and how certain design choices allow it to be a valuable addition to the field of Computer Sciences. In addition to comparing Swift to Objective-C, we will think critically about how it differs from other programming languages we've previously studied. However, from a more practical standpoint, Swift has many rich and powerful features that comes prepackaged with the language. A primary reason for this is due to the fact that this language is relatively young and developed by experienced individuals with the backing of a powerful company, which allowed for the developers of this language to learn from the pros and cons of a plethora of other programming languages including Apple's own Objective-C ("The Complete Guide To Swift Programming", 2017). Swift earned first place in the Most Loved Programming Language category from the Stack Overflow Developer Survey in 2015, and second and third place in 2016 and 2017 respectively ("Stack Overflow Developer Survey 2017", 2017). To explore the rich features that this language has to offer, we will be discussing a few key items and how these items are an improvement upon Objective-C, the language that has been historically used for iOS development.

1. Valuable Features of Swift

1A Readability in Implementation of OOP Semantics

Emulating many high level programming languages, Swift features many syntactical elements that makes the language's OOP features more readable and inline with industry standard syntax conventions. These improvements on readability are very clear when compared to Objective-C.

The table below compares the way a class is declared in Java, TypeScript, Objective-C, and Swift. All four languages are popular with mobile developers.

Typescript	`class Counter {`
	`  val: int;`
	`  constructor() {`
	`  this.val = 0;`
	`  }`
	`  function incrementBy(by: int): void {`
	`  this.val += by ;`
	`  }`
	`}`
	`let aCounter = new Counter(); aCounter.incrementBy(5);`

Java	`public class Counter {`
	`  int val;`
	`  Counter() {`
	`  this.val = 0;`
	`  }`
	`  void incrementBy(int by): void {`
	`  this.val += by ;`
	`  }`
	`}`
	`Counter aCounter = new Counter(); aCounter.incrementBy(5);`

Swift	`class Counter {`
	`  val: Int;`
	`  init() {`
	`  val = 0;`
	`  }`
	`  func incrementBy(by: int) {`
	`  this.val += by ;`
	`  }`
	`}`
	`let aCounter = new Counter(); aCounter.incrementBy(5);`

Objective-C	`@interface Counter:NSObject {`
	`  int val;`
	`}`
	`- (void) incrementBy: (int) by;`
	`@end`
	`@ implementation Counter`
	`- (id) init {`
	`  …`
	`  ...`
	`@ end`
	`Counter aCounter = [[Counter alloc] init]; [aCounter incrementBy by:5]`

Additionally, in programming, numerous string manipulation tasks are often required. Thus, having a clear, understandable, easy syntax for this task brings great benefits.

In Objective-C, mutable 'NSMutableString' class must be used instead of immutable 'NSString' class. Then String Format Specifiers must be used for transformations, which causes difficulty in intuitively recognizing what is intended. However, Swift offers undemanding syntax through simple mathematical notations which are more comprehensible to programmers that are new to the language.

	Objective-C	**Swift**
Creating a String	`NSMutableString *hello = [NSMutableString stringWithString: @"Hello"];`	`var hello = "Hello"`
Adding two Strings	`[hello appendString: @" World"];`	`Hello += " World"`

Evidently, Swift's syntax is much closer to other popular programming languages used in mobile development, compared to Objective-C. As a result, this makes Swift an easier language to learn and pickup for mobile developers coming from other platforms.

1B Null Pointer Exceptions

A pointer enables reference to a particular data by storing its address. However, pointers may make that data vulnerable by exposing and allowing direct access. In addition, pointers frequently hinder users in locating bugs. Objective-C often encounters the problems stated above. Although Swift uses pointers as well, Swift allows the user to immediately locate and fix bugs if the pointer has a `nil` value (missing a value) by crashing the program when it encounters them. This allows users to save effort and time required to fix a bugs related to pointers, which can be exhausting.

1C Creating an Object Only when Needed (Lazy Variables)

Sometimes, a user might want to create an object only when an access is required. This could benefit the user in two ways: by saving memory in the case where object requires a significant space in memory, and by reducing the delay in load time. For example, in cases where an object is required for a user interface, object with long loading time would cause a significant issue. As a result, Swift offers the 'lazy' keyword associated with the declaration of a variable (e.g. `lazy cat = new Cat()`) ("About Swift", 2017). This gives developers the ability to easily generate space saving efficiencies in the language.

Although Objective-C allows implementation of this feature through overriding the getter method and uses of '@property modifier' and 'self notation,' it is undeniable that extensive work is required. Compared to Objective-C, Swift is remarkably undemanding. Simply adding the 'lazy' modifier when declaring a variable would enable this feature (Dejeu, 2017).

1D Tracking Modification of Variables

Tracking points in time when variables are, or have been, modified often provides useful information. In Objective-C, this modification is notified only when the values have changed. On the other hand, Swift also notifies about when the change will take place in the future. To monitor alteration in Objective-C, KVO (Key Value Observation) must be set to the variable and destroyed upon deletion of the containing class. In contrast, Swift offers a straightforward solution by simply defining observers on a property (i.e., Int) or a variable (Dejeu, 2017).

1E Multiple Return Values

Swift allows returning of multiple variables from a method unlike Objective-C ("About Swift", 2017). There are few advantages achievable through this feature. For instance, less writing of codes might be required. For example, consider the case where there is a method which modifies the colour of a pencil object. This method can return a boolean value that indicates the colour changing was successful and also return the actual modified pencil object. Thus, two separate methods (one for checking whether the colour changing took place and one for returning the altered object) can be condensed into one function, requiring less lines of code.

1F Unified Files

Objective-C requires two code files: the header (.h) and the implementation (.m). Previously, programmers would need to make edits in two different files, which creates an additional point of divergence. One common example of this is ensuring that method names are synchronized across the header and implementation files. This creates additional bookkeeping work that may easily be forgotten about, especially when the header is located in a different file altogether. However with Swift, programmers can utilize the more cohesive file system (.swift) which ensures that these related changes are found in a single unified file.

1G Automatic Reference Counting

Automatic Reference Counting is an automatic memory management tool that adds retain/release/autorelease calls to code, based on calculations performed at runtime. ARC is

present in Objective-C, however it is unavailable for certain APIs (most notably the Core Graphics API) and procedural C code. Swift includes ARC for all APIs, allowing programmers to focus more on the functionality of their code rather than memory management.

1H Null Pointers and Optionals

Swift's optional types (value, or no value) allows functions to return values that wrapped in an optional or nil ("About Swift"). This features comes in handy when we need to implement functions that doesn't always need to return a value. For instance, consider a function that finds the largest number in an array. Logically, this function should either return nil in the event an empty array is passed to that function or an actual number if there is numbers in that array. The syntax for declaring an optional type in Swift is simply suffixing a type with a question mark. As an example, the function signature for a function that returns the largest integer in an array of integers, or nil if the array is empty, should look like this:

```
func findGreatest(list: [Int]) -> Int?
```

Objective-C does not have this. This is in contrast to Objective-C, where nil can be returned from any function similar to C. However, if we do assign the return value of a function that returns nil in Objective-C, we may cause a dangling pointer bug that can be cumbersome for developers to debug since no explicit error is thrown. However, Swift's nil-pointers cause applications to crash, which help the programmer to debug and locate the source of the error as soon as it occurs (Redwerk, 2016).

1I Type Inference and Generics

The compiler for Swift is smart enough to infer the type of a variable without having to explicitly declare its type. For example, developers can declare a string like so: `var foo = "Test String"` ("About Swift"). Notice that we did not specify that foo is a String - nevertheless, this line will be executed without any errors. This syntax is partially similar to languages like Javascript where variables are also declared without type. However, a very important distinction lies in the fact that Swift is still statically typed. Once we assign `foo` a String value we cannot assign `foo` an Integer value. This is not the case in dynamically typed languages like Javascript where we can assign and reassign variables to whatever value we want. Objective-C is also statically typed like Swift, but programmers must explicitly declare the type of every variable they create (Redwerk, 2016). This type inference ability of Swift allows the language to maintain the easy-to-read and non-verbose syntax styles of dynamically typed languages while also keeping the type safety features of statically typed languages.

We can combine the power of type inference with Swift's Generics, which can allow us to write generic functions that operates on multiple different types. For example, if we are asking the user to type in an answer, we could perform operations on their answer even if we don't know the type beforehand (String, Number, etc…).

1J Guards

The Swift language features the keyword *guard* that allows a program to cease execution if a given condition is not met. The syntax of this feature follows the format: `guard [condition] else { … }`. This enables Swift to have a very elegant error checking code pattern which can only be achieved through nested if statements in Objective-C which does not have guards (Redwerk, 2016).

2. Features we can Incorporate into our Alarm Clock

2.1 Argument Labels and Parameter Names

Swift introduces argument labels in function definitions, which refer to a parameter. Most other programming languages only utilize parameter names, making the code very awkward to read from a literacy perspective. Swift's argument labels allow the programmer to pass in parameters while using the argument labels, which results in prose-like code that is more readable and expressive. In this way, Apple is allowing the function call to flow like a sentence.

On the other hand, argument labels can also be omitted if the programmer so chooses. In this case, an underscore can be used in place of the label. We can use this in combination with subclasses to maintain clarity in our code. Take, for example, the UIButton class. If we want to make some changes to the draw function, so we will need to overwrite it. In this case, using an underscore in place of our own custom argument label helps to keep the code clean and maintain clarity.

2.2 Multiple Return Types

Our Alarm Clock project will include features to help users turn the alarm clock off, without the use of a snooze feature. This might include asking the user to answer a challenging math question or perform some physical exercise.

Once the user sets the alarm time, we can initialize the alarm by first calling a function which returns the following two values: a randomly generated mission to be completed, and the condition required to turn off the alarm. For example, this function might generate then return a math question such as, '23 x 11 + 2' and also the answer, '255.' Or in another case, the function might return a mission that asks the user to shake the phone and the condition that tells the phone should be shaken x times. Thus since Swift allows a function to return multiple values, we can have the program return values mentioned above without having to execute two different methods.

2.3 Powerful Error Handling

Having a powerful error handling makes the user to easily identify the causes of issues. As more detailed and specific the error handling is, it is easier to diagnose.

One simple way to handle errors is to intentionally return `nil` in a method where a value is expected, but not provided. By doing so, the user can ignore the error case by doing nothing, but this procedure can only be effective when no action should be taken when reaching an error state. Nonetheless, when designing a program, the best way to handle errors are to initially avoid their possibilities.

Swift includes features such as ‘guard’ and ‘failable initializers.’ Guard identifies whether a particular condition is met, but if the condition fails, a block of code can be executed. Thus guard allows the user to identify which conditions must be present before executing rest of the lines. In addition, failable initializers in Swift does not allow an object to be created unless sufficient information has been provided.

Within our app, we could make use of the ‘guard’ feature by checking if a valid type of answer was returned by the user. For example, if the user returned a string value when a numerical answer was expected, a block of code that prints out a statement such as, “Try again” and resetting the answer box could be executed. As for a different example, a user might return a blank answer. In that case, no action would be required, so the program could handle this error by returning a `nil` value.

3. Conclusion

Evident from it’s name, Swift was designed to be fast. It’s performance has since been measured by many, including Joseph Lord (2015), who noted that there are a number of adjustments programmers can make to optimize the speed of performance. Since he began his research, Swift has progressively updated to include some of these adjustments automatically - resulting in a speed difference of roughly 20% between Swift and Objective-C (Lord, 2015).

As another testament to Apple’s and focus on interactivity, Apple developed Playgrounds, a “revolutionary” (as described by Apple) app tailored to beginner coders. The app offers a game-like tutorial alongside multiple coding challenges. Playgrounds enables visual display of code snippets in real time as it is written, which makes the coding and testing process a lot more intuitive. The visual debugger, for one, allows coders to step through the code at various speeds. Overall, Playgrounds clearly highlights Apple’s focus on Swift’s friendliness and inclusivity towards beginner programmers.

With Swift’s numerous features aimed at improving readability and error handling, we are certain that our process of implementing an Alarm Clock app will be relatively straightforward and effortless, despite our lack of experience with programming in Swift.

4. References

About Swift. (2017, September 19). Retrieved November 09, 2017, from https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html

A Guide to Error Handling in Swift Programming Language. (2017). *Mindbowser*. Retrieved 10 November 2017, from http://mindbowser.com/error-handling-in-swift/

Cocoa Core Competencies: Objective-C. (2015, October 21). Retrieved November 09, 2017, from https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/ObjectiveC.html

Dejeu, A. (2017, January 27). Swift 3.0: Lazily Stored Properties, Variable Observers, KVO ( ). Retrieved November 09, 2017, from https://medium.com/ios-seminar/swift-3-0-lazily-stored-properties-variable-observers-kvo-ba877fed063e

Dua, K. (2017, February 14). 8 Advantages of Using Swift for iOS Development. Retrieved November 09, 2017, from https://clearbridgemobile.com/8-advantages-choosing-swift-objective-c-ios/

Lord, J. (2017). *How Swift is Swift?*. *Academy.realm.io*. Retrieved 10 November 2017, from https://academy.realm.io/posts/swift-summit-joseph-lord-performance/

Redwerk. (2016, June 10). Difference Between Swift And Objective-C. Titans Of iOS Development | Redwerk. Retrieved November 09, 2017, from https://redwerk.com/blog/10-differences-objective-c-swift

Rizos, Y. (2012, June 4). What is the benefit of multiple return values. Retrieved November 09, 2017, from https://softwareengineering.stackexchange.com/questions/96168/what-is-the-benefit-of-multiple-return-values

Solt, P. (2017). *Swift vs. Objective-C: 10 reasons the future favors Swift*. *InfoWorld*. Retrieved 10 November 2017, from https://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html

Stack Overflow Developer Survey 2017. (2017). Retrieved November 09, 2017, from https://insights.stackoverflow.com/survey/2017

Team, A. E. (2017, October 18). 9 Reasons to Choose Swift for iOS App Development. Retrieved November 09, 2017, from https://www.upwork.com/hiring/for-clients/9-reasons-to-choose-swift-for-ios-app-development/

The Complete Guide To Swift Programming. (2017). Retrieved November 09, 2017, from https://www.liveedu.tv/guides/programming/swift/history/

Wenderlich, R. (2017). *Magical Error Handling in Swift*. *Ray Wenderlich*. Retrieved 10 November 2017, from https://www.raywenderlich.com/130197/magical-error-handling-swift

Wodehouse, C. (2017, October 09). Swift vs. Objective-C | What Is The Difference? Retrieved November 09, 2017, from https://www.upwork.com/hiring/mobile/swift-vs-objective-c-a-look-at-ios-programming-languages/