Kotlin SPL Term Project Report

Group Members
Chris Malang, Vishwas Viswanath Banjan, Karthik Raveendran, Sanchari Chatterjee, Maksim Mozhelyuk

Table of Contents

Introduction & Motivation	3
History of Kotlin	3 - 4
Paradigm	4
Advantages & Disadvantages	4 - 6
Elements of Kotlin & Syntax	6 - 10
Writability, Readability, Reliability	11
Project Overview	12 - 13
Citations	14

Introduction

Our selection for programming language choice was initially something that we had already set out to do. Choosing Kotlin as our language came fairly easily as our group was collectively made up of like minds who expressed an interest in learning a new language and Kotlin just so happened to be the most interesting one we all shared an emphasis on wanting to learn. We figured Kotlin to be interesting as it is designed to cohesively operate well with Java, a language that the University of North Carolina at Charlotte places a great deal of curriculum coverage specifically in the Computer Science concentration. So having already a familiarity with how Java operates we thought it would help make it a seamless approach to learning the new language. In addition, Kotlin has the ability to be fully interoperable with the Java Virtual Machine (JVM) which essentially allows for the leveraging of already existing libraries.

History of Kotlin

Kotlin at its inception is still a young language having been first introduced to the public in 2011 its name originated from the actual island Kotlin located near the Gulf of Finland. On February 15, 2016, the language was officially debuted with Kotlin version 1.0. Despite Kotlin being first unveiled as a project back in 2011, it has started to gain traction as the language is well versatile. As a result, Kotlin is capable of being used on the server-side, client-side web, and the Android platform. JetBrains are the creators of Kotlin programming language who is quite reputable in the software development industry. They are most notably recognized for their integrated development environment IntelliJ IDEA software, which can handle a wide variety of programming languages such as Java, Ruby, Python, and Kotlin just to name a few. With a lot of already powerful programming languages out on the market, one might ask what was the reason for the creation of another new language such as Kotlin. The main reason why Kotlin was invented as stated by JetBrains lead Dmitry Jemerov was, “that most of the languages out at the time didn’t have all the additional features” (Wikipedia).

Moving into the evolution of the functional programming language Kotlin we immerse ourselves in the various versions that have been created to date. In chronological order here are the various releases of Kotlin programming. On February 15, 2016, JetBrains officially announced there stable release of version 1.0 following this timeline a year later in 2017 Google I/O announced that they would begin providing support to the Kotlin language specifically for Android. On November 28, 2017, the functional programming released version 1.2 accompanying this release were quite significant features. Kotlin version 1.2 was now equipped to handle the reusing of code through multiple platforms such as JVM, Javascript, and Native. JetBrains on October 29, 2019, released there most current and up to date release version 1.3, which brought the addition of coroutines (lightweight threads) in support of asynchronous programming. As a result, the creators placed and emphasis on making the experience for the developer fluid as well as scalable when required. With the dedication and commitment that JetBrains has shown to Kotlin as a functional modern programming language, the results are starting to come into fruition. As a result, on May 7, 2019, Google I/O officially announced that their preferred language of choice in Android application development was in fact Kotlin. From Google announcing that they would be providing support to Kotlin in 2017 to announcing that Kotlin would become their preferred language in 2019, the language has truly made leaps and boundaries over a short timeframe. Kotlin is truly an interesting language and there is no doubt more in store in the foreseeable future.

Paradigm

Kotlin the programming language has a unique and versatile build design. It's syntax and patterns maintain the ability to utilize object-oriented programming such as Java's JVM while extending the additional feature of coding in a more functional way. Kotlin also provides similar syntax and concept to other languages such as C# and Scala, in fact, it has adopted the companion object paradigm that Scala utilizes. The beauty of Kotlin programming is in the design through the various releases Kotlin has continued to grow within the Android developer community. While object-oriented programming traces its foundational structure on the bases of object creation, functional programming provides a more transparent approach as it alleviates stress during evaluation. As a result, adopting the functional paradigm allows for numerous benefits such as debugging and comprehending code can become more simplistic while still maintaining a level of complexity. OOP (object-Oriented Programming) on the other hand takes a more stateful programming model. It's advantages span from having a clear modular structure to the ability to reuse objects that have already been created. This, as a result, drops overall costs during the development process. Kotlin utilizes both OOP as well as functional programming which ultimately, creates the ability for developers to have options when programming.

Advantages and Disadvantages

Since Kotlin was designed to improve upon the Java programming language a majority of the comparisons of advantages to disadvantages rely heavily on Kotlin versus Java. Some of the results were that Kotlin improved the way things were done by reducing the clutter in comparison to Java programming. Syntax in Kotlin is more concise, easier to comprehend, and makes evaluations simpler. The functional programming model was included in Kotlin as well to strengthen its abilities. A general overview of this section, we will observe the uniqueness that Kotlin encompasses by doing a comparison of advantages to disadvantages that along with some interpretation of how things differ between the two. From there we will elaborate on the elements of the language Kotlin such as primitive data types, structured types, and reserved words.

Advantages

- **Increases team efficiency-**Being rather clear and compact, the language increases team efficiency because of its succinct and intuitive syntax. More work can be done as it takes fewer lines to write and deploy working code that leads to fewer time requirements.

- **Easily maintainable-**Kotlin is supported by a vast majority of IDEs, including Android Studio, and other SDK tools. This helps increase developers' productivity, as they can continue to work with the toolkit they are used to.

- **Less buggy-**Kotlin makes the code in production more stable and consistent as it offers a much more clear and compact codebase. Bugs get detected at compile-time, so developers can fix errors before runtime.

- **Complies with existing Java code-**Kotlin is positioned as a total Java-interoperable programming language. It is consistent with Java and all related tools and frameworks, which makes it possible to switch to Kotlin step by step. In case your product cannot be written in Kotlin only, both *languages can be comfortably used at the same time.*

Disadvantages

- **Still, not Java-**Both java and Kotlin are pretty similar, but at their core, these are two different languages. Developers won't be able to quickly shift from one to another without taking some time to learn Kotlin.

- **Fluctuating compilation speed-**In some cases, Kotlin is faster than Java - mostly when performing incremental builds. But still, we should remember that Java is the winner when it comes to clear builds.

- **Fewer Kotlin experts for hire-**The demand for specialists has abruptly escalated after the announcement of Kotlin getting adopted as a first-class programming language at Google, but still, there are more Java programmers on the market.

Elements of Kotlin & Syntax

Null Safety

Kotlin programming utilizes unique syntax with regard to variables. The *null safety* feature within Kotlin syntax is designed to eliminate the danger of null referencing within the code as many issues can arise from this. Since Kotlin was designed to improve on Java the developers made this key feature as attempting to access a null reference can quickly prove problematic triggering a null reference exception in Java. As a result, by default, Kotlin is able to distinguish between references that can hold nullable references and those that aren't able to hold them (non-nullable references). This feat is possible by using the operator (***?)*** the example below provides further elaboration.

```
var x: String = “Tom"

assertEquals(x.length, 3)
```

The resulting code when generated would cause a compiler error as null cannot be assigned to x. However, by adding the following operator ***(?)*** to type String we would be able to assign null to variable x as shown below.

```
var x: String? = "Tom"

assertEquals(x.length, 3)
```

Finally, since we now have x associated with a null case we would have to be explicit when handling it in order to avoid compilation errors as Kotlin is aware that variable x can hold the value of null.

```
if (x != null) {
   println(x.length)
} else {
    assertNull(x)
```

Elvis Operator

The Elvis Operator defined with a (?:) is an additional feature to check whether an argument actually contains a nullable reference. If the first condition is a non-nullable it will be run and if the condition is null then the second condition to the right of the operator will execute ideally this feature is quite similar to an if-else statement. Ultimately, the below code will always execute the left-hand side condition *y?.length* if it is set to not null. However, if the left-hand condition is set to null then the right-hand condition *5* will execute.

Example:

```

val x = y?.length ?: 5

```

Variables in Kotlin

val & var

In Kotlin *val* and *var* in comparison are used to distinguish between what is mutable and what is immutable. To further elaborate *val* is considered to be immutable which means that once an object has been stored with *val* it no longer can be changed and becomes read-only. The opposite is *var* which is considered to be mutable and as a result, can be dynamically assigned and is capable of changing value over time.

Example of *val*:

```
1    fun main(args: Array<String>) {
2           val numberOfMinutesInHour = 60
3           print("Number of minutes in an hour: " + numberOfMinutesInHour)
4
5           numberOfMinutesInHour = 40
6           print("Number of minutes in an hour: " + numberOfMinutesInHour)
7    }
```

The results of the above code would generate a compilation error as *val* has been assigned the value of 60 initially at line 2 and later the user attempts to reassign the value of *val* to 40 at line 5 which is not allowed by the compiler.

Type, Type Inference

Unlike Java, Kotlin actually differs with regards to Type and Type Inference. While Java makes it mandatory for a programmer to explicitly declare the type of every function as well as variables Kotlin is quite the opposite. Since Kotlin is considered to be a strongly typed language it comes equipped with type inference or deduction. This additional feature is quite powerful as it gives the programmer the complete option of deciding whether they would like to implicitly or explicitly define a type to a variable. As a result, the code below while in Java would trigger a compiler error in Kotlin it would not. The compiler would be able to identify that country is a string and pin as an integer.

```
val country = “Germany”
val pin = 8849
```

Coroutines

With the introduction of Coroutines in Kotlin’s version 1.3 release, the language is now capable of asynchronous programming. A coroutine from a general perspective consists of a sequence of sub-task that will execute when running in a particular order. Generally, coroutines allow for computations to be suspended and executed at a later time. Essentially it allows for threading to occur within Kotlin and is quite considered to be lightweight. The Kotlin coroutine library actually contains a wide variety of construct builds so we will go over a few below:

- launch() Coroutine - This function is important as it is used to initially launch a coroutine via the launch coroutine builder. The below code demonstrates its ability which when run will allow the word “Kotlin” to be printed followed by a delay of five seconds and a two-second delay that later prints “is Great!”/

  Example:

  ```

  fun main() {
      GlobalScope.launch {
          delay(2000L)
          println(“is Great!”)
      }
      println(“Kotlin”)
      Thread.sleep(5000L)
  }

  ```
- runblocking() - Allows for another new coroutine to execute while simultaneously blocking the current thread until the runblocking() function has completed. As a result, runblocking allows for the calling of suspended functions that are outside of the coroutine. The below code would not be allowed to execute talk() without runblocking() as it is considered a suspended function.

  Example:

```
suspend fun talk() : Any =  TODO()

fun main() {

    runblocking {
        talk()
    }
}

```

Lambdas

Lambda expression is a simplified representation of a function. It can be passed as a parameter, stored in a variable or can also be returned as a value. Lambda expression is the shorter way of describing a function. It doesn't need a name or a return statement. We can store the lambda expressions in a variable and can execute them as regular functions. Lambdas can also be passed as parameters to other functions or be the return value.

❖ Lambda expressions

It doesn't need a name or a return statement. Lambda expressions are stored in a variable and are executed as regular functions. They can also be passed as parameters to other functions or be the return value.

Example

```
 fun main(args: Array<String>)
{
 val greeting = { println("Hello!")}
   // invoking function
   greeting()
 }
```

The output will be **Hello!**

Here, a lambda expression is assigned to the variable greeting. The expression doesn't accept any parameters and doesn't return any value in this program. Then, the function (lambda expression) is invoked as a greeting().

❖ Function Types

Functions as arguments can be passed to other functions. Also, a function can be returned from other functions. These functions are called higher-order functions.

Example-

```
fun callMe(greeting: () -> Unit)
 {
   greeting()
}
fun main(args: Array<String>)
{
   callMe({ println("Hello!") })
}
```

The output will be **Hello!**

Here, callMe()is a higher-order function (because it takes function as a parameter). The greeting parameter accepts the lambda passed to the callMe() function as greeting: () -> Unit

- Lambda With Parameters and Return Type

  The program here has a lambda expression accepting two integers as parameters and then returns the product of those two integers.
  Example-

  ```
  fun main(args: Array<String>)
  {
     val product = { a: Int, b: Int -> a * b }
     val result = product(9, 3)
     println(result)
  }
  ```

  The output will be **27**
  The lambda expression is-**{ a: Int, b: Int -> a * b }**
  a: Int, b: Int-Parameters
  a * b-Body
  Lambda expressions are enclosed inside curly braces.

Writability, Readability, and Reliability

Writability

In Kotlin the main focus was to take what Java brought to the programming scene and further improve on it. This led to an improved quality of language that makes life easier for programmers as code is written with a more simplistic approach. This enables a programmer who is coding in Kotlin capable of expressing complicated computation clearly, concisely, correctly and swift which is a direct representation of the readability that the language possesses.

Readability

With relation to Kotlin programming and readability, the main influence of the language is traced back to Java programming. Programmers coming from a Java background will feel at home with Kotlin with some slight variations that actually make it more simple to comprehend as a result of the ability that functional programming brings. Kotlin also encourages programmer efficiency as the language expressiveness makes complex processes and structure more simplistic. In other words, programmers are able to mindfully map the layout of the program design with ease.

Reliability

Kotlin's reliability we would have to say is quite significant as referencing back from historical events the language has continued to make leaps and bounds. Here's a brief refresher of the timeframe wise Kotlin's stable release of version 1.0 in February of 2016, to Google I/O announcing first-class support in 2017, and in May of 2019 Google, I/O announced that their preferred language for Android application development would be Kotlin. This just demonstrates the magnitude that Kotlin's reliability has as a reputable company such as Google I/O is able to place their trust and fate in the programming language. Some of the features that make Kotlin reliable are things such as the addition of null safety which ultimately allows for added checks to ensure no errors are generated during runtime. In addition, Kotlin has undergone numerous testing stages before its final release with regard to the timeframe expressed above as well as allowing for reverse compatibility on previous versions.

SPL Programming Project NewsNxt

Overview

For our semester term project with relation to Kotlin, we were a bit undecided about the type of project we initially set out to develop. Our first project proposal early in the semester was to attempt a Kotlin based chat application, which through this application would allow users to access the app and have a conversation with one another. In addition, we wanted to allow the application to have the functionality of allowing users to transfer messages either from a one on one context or a one to many group chat type feature. Ultimately, we decided to forego that project idea as we quickly realized the scope was out of reach in correlation with the timeline we've been given.

The results of that setback actually influenced us into the project that we ended up developing. Our realization of attempting to create a chat application actually resulted in us creating a news type application that would be fully mobile. The application would allow users to have every news source accessible from a central application at the palm of your hand. As a result, we developed News NXT through this idea.

Technology Used

As far as the technology used we went with Andriod Studio as our choice of the integrated development environment. Initially, we were all utilizing JetBrain's IntelliJ IDEA software, but we quickly realized that we wanted our application to be used mobile. Android Studio actually is built on JetBrains IntelliJ software so it actually worked for us. The added feature that comes prebuilt with the IDE actually made it better for us to use such as the ability to actually drag and drop graphical user interfaces as well as having a live preview that displays different layout according to resolutions. In addition, below are some of the sources and external libraries that we used to help us build the news application:

- Newsapi.org
- https://square.github.io/picasso/
- https://square.github.io/okhttp/
- https://www.ocpsoft.org/prettytime/

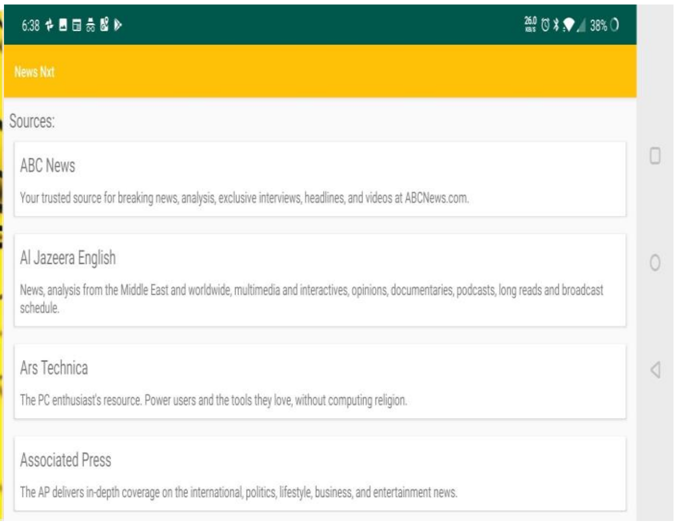
What Kotlin made smoother

With Kotlin being a strongly typed language we found the process smoother as the main point was that the language had familiarity with Java. Coding in Kotlin made it easier to an extent as we found code to be more concise in comparison to other languages such as Java. The driving point that we found smooth through the process was having the sheer ability to be able to code both in functional programming as well as object-oriented programming.

What Kotlin made tough

Surprisingly enough Kotlin wasn't quite tough to code in. The main issue we encountered through our project was the issue of time. The reason being is that we were initially as a group quite undecided on what we wanted to tackle from a project perspective. As stated earlier in the report during our project proposal we actually wanted to develop an application that would allow users to chat with one another. Essentially, we set out to create a chat application that we determined later into the semester that the entirely of getting it completed would not be quite realistic as thinking of the over the scope of the project turned out to be complex. We quickly switched to developing a news application, but overall the main hurdle of the project was time for us.

Application Screenshots



Sources:

ABC News
Your trusted source for breaking news, analysis, exclusive interviews, headlines, and videos at ABCNews.com.

Al Jazeera English
News, analysis from the Middle East and worldwide, multimedia and interactives, opinions, documentaries, podcasts, long reads and broadcast schedule.

Ars Technica
The PC enthusiast's resource. Power users and the tools they love, without computing religion.

Associated Press
The AP delivers in-depth coverage on the international, politics, lifestyle, business, and entertainment news.

Citations

Vasiliy, et al. "Why JetBrains Invented and Promotes Kotlin." *TechYourChance*, 22 Apr. 2018, https://www.techyourchance.com/jetbrains-invented-promotes-kotlin/.

"FAQ." *Kotlin*, https://kotlinlang.org/docs/reference/faq.html.

"Kotlin (Programming Language)." *Wikipedia*, Wikimedia Foundation, 6 Nov. 2019, https://en.wikipedia.org/wiki/Kotlin_(programming_language).

"What's New in Kotlin 1.2." *Kotlin*, https://kotlinlang.org/docs/reference/whatsnew12.html.

Lardinois, Frederic. "Kotlin Is Now Google's Preferred Language for Android App Development." *TechCrunch*, TechCrunch, 7 May 2019, https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/.

“Functional Programming vs OOP - Which One Is More Useful.” *EDUCBA*, 29 July 2019, https://www.educba.com/functional-programming-vs-oop/.

“Comparison to Java Programming Language.” *Kotlin*, https://kotlinlang.org/docs/reference/comparison-to-java.html.

https://developer.android.com/kotlin

Rusu, Dan. “Kotlin Avoids Entire Categories of Java Defects.” *Medium*, ProAndroidDev, 14 May 2018, https://proandroiddev.com/kotlin-avoids-entire-categories-of-java-defects-89f160ba4671.

Picasso, https://square.github.io/picasso/.

Prettytime.” *OCPsoft*, https://www.ocpsoft.org/prettytime/.

Square, Inc. “OkHttp.” *OkHttp*, https://square.github.io/okhttp/.