**Using Kotlin for Android Development**

Kotlin is a great fit for developing Android applications, bringing all of the advantages of a modern language to the Android platform without introducing any new restrictions:

**Compatibility:** Kotlin is fully compatible with JDK 6, ensuring that Kotlin applications can run on older Android devices with no issues. The Kotlin tooling is fully supported in Android Studio and compatible with the Android build system.

**Performance:** A Kotlin application runs as fast as an equivalent Java one, thanks to very similar bytecode structure. With Kotlin's support for inline functions, code using lambdas often runs even faster than the same code written in Java.

**Interoperability:** Kotlin is 100% interoperable with Java, allowing to use all existing Android libraries in a Kotlin application. This includes annotation processing, so databinding and Dagger work too.

**Footprint:** Kotlin has a very compact runtime library, which can be further reduced through the use of ProGuard. In a real application, the Kotlin runtime adds only a few hundred methods and less than 100K to the size of the .apk file.

**Compilation Time:** Kotlin supports efficient incremental compilation, so while there's some additional overhead for clean builds, incremental builds are usually as fast or faster than with Java.

**Learning Curve:** For a Java developer, getting started with Kotlin is very easy. The automated Java to Kotlin converter included in the Kotlin plugin helps with the first steps. Kotlin Koans offer a guide through the key features of the language with a series of interactive exercises.

**Advantages and Disadvantages**

Following are some of the advantages of using Kotlin for your application development.

**Easy Language** − Kotlin is a functional language and very easy to learn. The syntax is pretty much similar to Java, hence it is very easy to remember. Kotlin is more expressive, which makes your code more readable and understandable.

**Concise** − Kotlin is based on JVM and it is a functional language. Thus, it reduce lots of boiler plate code used in other programming languages.

**Runtime and Performance** − Better performance and small runtime.

**Interoperability** − Kotlin is mature enough to build an interoperable application in a less complex manner.

**Brand New** − Kotlin is a brand new language that gives developers a fresh start. It is not a replacement of Java, though it is developed over JVM. It is accepted as the first official language of android development. Kotlin can be defined as - **Kotlin** = **JAVA** + extra updated new features.

**Following are some of the disadvantages of Kotlin.**

**Namespace declaration** − Kotlin allows developers to declare the functions at the top level. However, whenever the same function is declared in many places of your application, then it is hard to understand which function is being called.

**No Static Declaration** − Kotlin does not have usual static handling modifier like Java, which can cause some problem to the conventional Java developer.

**Kotlin - Environment Setup**

However, if you still want to use Kotlin offline in your local system, then you need to execute the following steps to configure your local workspace.

**Step 1** − Java 8 installation.

Kotlin runs on JVM, hence. it is really necessary to use JDK 8 for your local Kotlin development. Please refer to the official website of oracle to download and install JDK 8 or an above version. You might have to set the environment variable for JAVA such that it can work properly. To verify your installation in Windows operating system, hit “java –version” in the command prompt and as an output it will show you the java version installed in your system.

**Step 2** − IDE installation.

There are a number of IDE available over the internet. You can use any of your choice. You can find the download link of different IDE in the following table.

**IDE Name** **Installation Link**

1. NetBeans

https://netbeans.org/downloads/

1. Eclipse

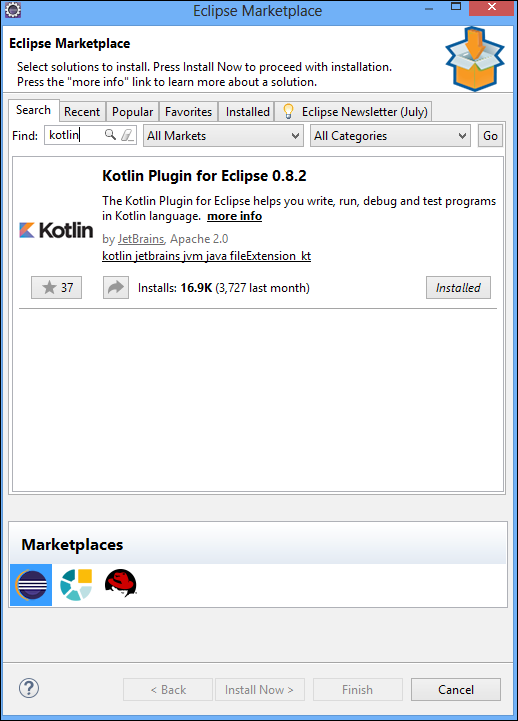
https://www.eclipse.org/downloads/

1. Intellij

https://www.jetbrains.com/idea/download/#section = windows

**Step 3** − Configuring Eclipse.

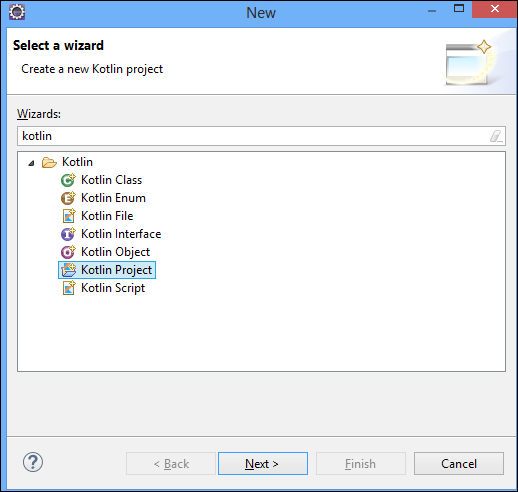
Open Eclipse and go to “Eclipse Market Place”. You will find the following screen.



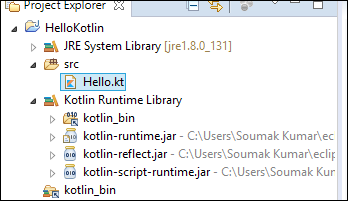
Search for Kotlin in the search box and install the same in your local system. It might take some time depending on the internet speed. You may have to restart your Eclipse, once it is successfully installed.

**Step 4** − **Kotlin Project**.

Once Eclipse is successfully restarted and Kotlin is installed, you will be able to create a Kotlin project on the fly. Go to **File** → **New** → **Others** and select “**Kotlin** **project**” from the list.



Once the project setup is done, you can create a Kotlin file under “SRC” folder. Left-click on the “Src” folder and hit “new”. You will get an option for Kotlin file, otherwise you may have to search from the “others”. Once the new file is created, your project directory will be look like the following.



Your development environment is ready now. Go ahead and add the following piece of code in the “Hello.kt” file.

fun main(args: Array<String>) {

println("Hello, World!")

}

**Working with the Command Line Compiler**

**For Linux – UBUNTU**

**1) Snap** package

If you’re on Ubuntu 16.04 or later, you can install the compiler from the command line:

**$** sudo snap install --classic kotlin

**2)** Create a simple application in Kotlin that displays **Hello, World!**.

Using our favorite editor, we create a new file called hello.kt with the following:

fun main(args: Array<String>) {

println("Hello, World!")

}

3) Compile the application using the Kotlin compiler

**$** kotlinc hello.kt -include-runtime **-d** hello.jar

4) Run the application.

**$** java -jar hello.jar

### **Running the REPL**

We can run the compiler without parameters to have an interactive shell.

We can type any valid Kotlin code and see the results.

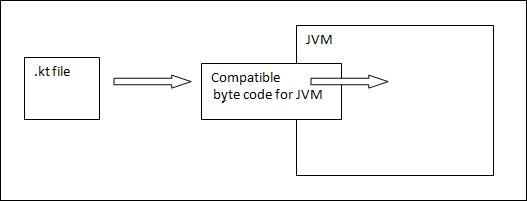


**Kotlin Architecture:**

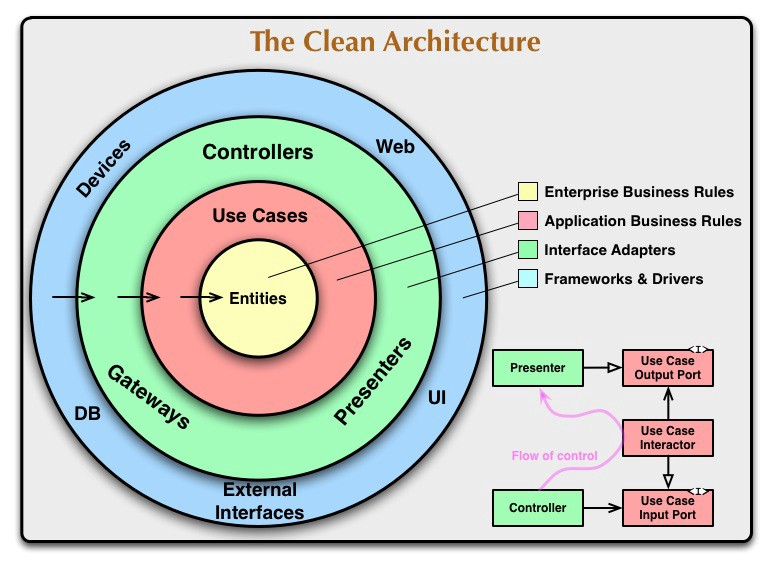
Kotlin is a programming language and has its ***own architecture*** to allocate memory and *produce a quality output* to the end user.

Following are the different scenarios where Kotlin compiler will work differently, whenever it is targeting different other kind of languages such as Java and JavaScript.

Kotlin compiler creates a byte code and that byte code can run on the JVM, which is exactly equal to the byte code generated by the Java .class file.

* Whenever two byte coded file runs on the JVM, they can communicate with each other and this is how an **interoperable feature** is established in Kotlin for Java.
* Whenever Kotlin targets JavaScript, the Kotlin compiler converts the .kt file into **ES5.1** and generates a compatible code for JavaScript.

Kotlin compiler is capable of creating platform basis compatible codes via LLVM.



**Basic Datatypes in Kotlin**

**Numbers**

Similar to Java.

Kotlin does not allow internal conversion of different data types.

|  |  |
| --- | --- |
| **Type** | **Size** |
| Double | 64 |
| Float | 32 |
| Long | 64 |
| Int | 32 |
| Short | 16 |
| Byte | 8 |

Example:

fun main(args: Array<String>) {

val a: Int = 10000

val d: Double = 100.00

val f: Float = 100.00f

val l: Long = 1000000004

val s: Short = 10

val b: Byte = 1

println("Your Int Value is "+a);

println("Your Double Value is "+d);

println("Your Float Value is "+f);

println("Your Long Value is "+l);

println("Your Short Value is "+s);

println("Your Byte Value is "+b);

}

**Characters**

Kotlin represents character using **char**.

Character should be declared ***in a single quote*** like ‘c’.

Character variable cannot be declared like number variables.

Kotlin variable can be declared in two ways - one using “**var**” and another using “**val**”.

fun main(args: Array<String>) {

val letter: Char // defining a variable

letter = 'A' // Assigning a value to it

println("$letter")

}

**Boolean**

Like other language, boolean is very simple.

There are two values for Boolean – ***true*** or false.

fun main(args: Array<String>) {

val letter: Boolean // defining a variable

letter = true // Assinging a value to it

println("Your character value is "+"$letter")

}

**Strings**

Strings are character arrays.

Like Java, they are **immutable** in nature.

There are two kinds of string available in Kotlin

- one is called raw String and

- another is called escaped String.

fun main(args: Array<String>) {

var rawString :String = "I am Raw String!"

val escapedString : String = "I am escaped String!\n"

println("Hello!"+escapedString)

println("Hey!!"+rawString)

}

## **Arrays**

Arrays are a collection of homogeneous data. Like Java, Kotlin supports arrays of different data types.

fun main(args: Array<String>) {

val numbers: IntArray = intArrayOf(1, 2, 3, 4, 5)

println("Hey!! I am array Example"+numbers[2])

}

**Collections**

Collection is a very important part of the data structure, which makes the software development easy for engineers.

Kotlin has two types of collection

- one is **immutable** collection (which means **lists**, maps and sets that cannot be editable) and

- **mutable** collection (this type of collection is editable).

It is very important to keep in mind the type of collection used in your application, as Kotlin system does not represent any specific difference in them.

fun main(args: Array<String>) {

val numbers: MutableList<Int> = mutableListOf(1, 2, 3) //mutable List

val readOnlyView: List<Int> = numbers // immutable list

println("my mutable list--"+numbers) // prints "[1, 2, 3]"

numbers.add(4)

println("my mutable list after addition --"+numbers) // prints "[1, 2, 3, 4]"

println(readOnlyView)

readOnlyView.clear() // ⇒ does not compile

// gives error

}

some useful methods such as first(), last(), filter(), etc.

fun main(args: Array<String>) {

val items = listOf(1, 2, 3, 4)

println("First Element of our list----"+items.first())

println("Last Element of our list----"+items.last())

println("Even Numbers of our List----"+items.

filter { it % 2 == 0 }) // returns [2, 4]

val readWriteMap = hashMapOf("foo" to 1, "bar" to 2)

println(readWriteMap["foo"]) // prints "1"

val strings = hashSetOf("a", "b", "c", "c")

println("My Set Values are"+strings)

}

**Ranges**

Ranges is another unique characteristic of Kotlin.

It provides an operator that helps you iterate through a range.

Internally, it is implemented using rangeTo() and its operator form is (..).

fun main(args: Array<String>) {

val i:Int = 2

for (j in 1..4)

print(j) // prints "1234"

if (i in 1..10) { // equivalent of 1 < = i && i < = 10

println("we found your number --"+i)

}

}

**Kotlin Control Flow:**

**If – Else**

fun main(args: Array<String>) {

val a:Int = 5

val b:Int = 2

var max: Int

if (a > b) {

max = a

} else {

max = b

}

print("Maximum of a or b is " +max)

// As expression

// val max = if (a > b) a else b

}

**Use of When**

If you are familiar with other programming languages, then you might have heard of the term **switch** statement, which is basically a conditional operator when multiple conditions can be applied on a particular variable. “when” operator matches the variable value against the branch conditions. If it is satisfying the branch condition then it will execute the statement inside that scope.

fun main(args: Array<String>) {

val x:Int = 5

when (x) {

1 -> print("x = = 1")

2 -> print("x = = 2")

// 1,2 -> print(" Value of X either 1,2") <<-- Valid for multiple matches

else -> { // Note the block

print("x is neither 1 nor 2")

}

}

}

**For Loop**

fun main(args: Array<String>) {

val items = listOf(1, 2, 3, 4)

for (i in items)

println("values of the array : "+i)

}

fun main(args: Array<String>) {

val items = listOf(1, 22, 83, 4)

for ((index, value) in items.withIndex()) {

println("the element at $index is $value")

}

}

**While Loop and Do-While Loop**

**//While loop**

fun main(args: Array<String>) {

var x:Int = 0

println("Example of While Loop--")

while(x< = 10) {

println(x)

x++

}

}

// **Do-while loop**

fun main(args: Array<String>) {

var x:Int = 0

do {

x = x + 10

println("I am inside Do block---"+x)

}while(x <= 50)

}

**Use of Return, Break, Continue**

fun main(args: Array<String>) {

var x:Int = 10

println("The value of X is--"+doubleMe(x))

}

fun doubleMe(x:Int):Int {

return 2\*x;

}

fun main(args: Array<String>) {

println("Example of Break and Continue")

myLabel@ for(x in 1..10) { // appling the custom label

if(x = = 5) {

println("I am inside if block with value"+x+"\n-- hence it will close the operation")

break@myLabel //specifing the label

} else {

println("I am inside else block with value"+x)

continue @myLabel

}

}

}

**Class & Object**

Basic Syntax:

class myClass { // class Header

// class Body

}

Example:

class myClass {

// property (data member)

private var name: String = "Anthony"

// member function

fun printMe() {

print("My Named is : "+name)

}

}

fun main(args: Array<String>) {

val obj = myClass() // create obj object of myClass class

obj.printMe()

}

**Nested Class**

By definition, when a class has been created inside another class, then it is called as a nested class. In Kotlin, nested class is by default static, hence, it can be accessed without creating any object of that class. In the following example, we will see how Kotlin interprets our nested class.

fun main(args: Array<String>) {

val demo = Outer.Nested().foo() // calling nested class method

print(demo)

}

class Outer {

class Nested {

fun foo() = "Welcome to Nested Class"

}

}

**Inner Class**

When a nested class is marked as a “inner”, then it will be called as an Inner class.

An inner class can be accessed by the data member of the outer class. In the following example, we will be accessing the data member of the outer class.

fun main(args: Array<String>) {

val demo = Outer().Nested().foo() // calling nested class method

println(demo)

}

class Outer {

private val welcomeMessage: String = "Welcome to the InnerClass Message"

inner class Nested {

fun foo() = welcomeMessage

}

}

**Anonymous Inner Class**

Anonymous inner class is a pretty good concept that makes the life of a programmer very easy. Whenever we are implementing an interface, the concept of anonymous inner block comes into picture. *The concept of creating an object of interface using runtime object reference is known as anonymous class*. In the following example, we will create an interface and we will create an object of that interface using Anonymous Inner class mechanism.

interface Human {

fun think()

}

fun main(args: Array<String>) {

var programmer : Human = object:Human {

// creating an instance of the interface

override fun think() {

// overriding the think method

print("I am an example of Anonymous Inner Class ")

}

}

programmer.think()

}

**Constructors**

* Kotlin has **two types** of constructor -
  + one is the primary constructor and
  + the other is the secondary constructor.
* One Kotlin class can have one primary constructor, and one or more secondary constructor.
* Java constructor initializes the member variables, however, in Kotlin the primary constructor initializes the class, whereas the secondary constructor helps to include some extra logic while initializing the same.
* The primary constructor can be declared at class header level as shown in the following example.

class Person(val firstName: String, var age: Int) {

// class body

}

fun main(args: Array<String>) {

val person1 = Person("Jainul", 35)

println("First Name = ${person1.firstName}")

println("Age = ${person1.age}")

}

class Person(val firstName: String, var age: Int) {

}

**Tools for Android Development**

The Kotlin team offers a set of tools for Android development that goes beyond the standard language features:

**Kotlin Android Extensions** is a compiler extension that allows you to get rid of findViewById() calls in your code and to replace them with synthetic compiler-generated properties.

**Anko** is a library providing a set of Kotlin-friendly wrappers around the Android APIs, as well as a DSL that lets you replace your layout .xml files with Kotlin code.

**Next Steps**

Download and install Android Studio which includes Kotlin support out of the box.

Follow the Getting Started with Android and Kotlin tutorial to create your first Kotlin application.

For a more in-depth introduction, check out the reference documentation on this site and Kotlin Koans.

Another great resource is Kotlin for Android Developers, a book that guides you step by step through the process of creating a real Android application in Kotlin.

Check out Google's sample projects written in Kotlin.

<https://kotlinlang.org/docs/reference/basic-syntax.html>

[https://codelabs.developers.google.com/codelabs/build-your-first-android-app-kotlin/index.html#0](https://codelabs.developers.google.com/codelabs/build-your-first-android-app-kotlin/index.html" \l "0)