CS – 31 Mobile Application Develop`ment in Android using Kotlin

BCA Semester – 6

H & H Kotak Institute of Science, Rajkot

BCA Department

Unit-1

Introduction to Kotlin Programming

Basics of Kotlin

Kotlin tutorial provides basic and advanced concepts of Kotlin programming language. Our Kotlin tutorial is designed for beginners and professionals both. Kotlin is a statically-typed, general-purpose programming language. It is widely used to develop android applications. Our Kotlin syllabus includes all topics of Kotlin such as introduction, architecture, class, object, inheritance, interface, generics, delegation, functions, mixing Java and Kotlin, Java vs. Kotlin, etc.

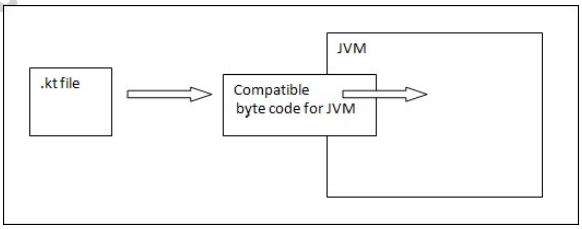
What is Kotlin? Kotlin is a general-purpose, statically typed, and open-source programming language. It runs on JVM and can be used anywhere Java is used today. It can be used to develop Android apps, server-side apps and much more.

History of Kotlin: Kotlin was developed by JetBrains team. A project was started in 2010 to develop the language and officially, first released in February 2016. Kotlin was developed under the Apache 2.0 license.

Features of Kotlin:

- **Concise:** Kotlin reduces writing the extra codes. This makes Kotlin more concise.
- **Null safety:** Kotlin is null safety language. Kotlin aimed to eliminate the NullPointerException (null reference) from the code.
- **Interoperable:** Kotlin easily calls the Java code in a natural way as well as Kotlin code can be used by Java.
- **Smart cast:** It explicitly typecasts the immutable values and inserts the value in its safe cast automatically.
- **Compilation Time**: It has better performance and fast compilation time.
- **Tool-friendly:** Kotlin programs are build using the command line as well as any of Java IDE.
- **Extension function:** Kotlin supports extension functions and extension properties which means it helps to extend the functionality of classes without touching their code.

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. Compile Kotlin into bytecode which can run on JVM. This bytecode is exactly equal to the byte code generated by the Java .class file. 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. Kotlin Multiplatform Mobile (KMM) is used to create multiplatform mobile applications with code shared between Android and iOS. Whenever two byte coded files (Two different programs from Kotlin and Java) runs on the JVM, they can communicate with each other and this is how an interoperable feature is established in Kotlin for Java.



Kotlin/Native is primarily designed to allow compilation for platforms where virtual machines are not desirable or possible, for example, embedded devices or iOS. It is easy to include a compiled Kotlin code into existing projects written in C, C++, Swift, Objective-C, and other languages.

Operations and Priorities

Kotlin Variable: Variables are an important part of any programming. They are the names you give to computer memory locations which are used to store values in a computer program and later you use those names to retrieve the stored values and use them in your program. Kotlin variables are created using either var or val keywords and then an equal sign = is used to assign a value to those created variables. Mutable means that the variable can be reassigned to a different value after initial assignment. To declare a mutable variable, we use the var keyword. A read-only variable can be declared using val (instead of var) and once a value is assigned, it can not be re-assigned.

Kotlin Variable Naming Rules

- There are certain rules to be followed while naming the Kotlin variables:
- Kotlin variable names can contain letters, digits, underscores, and dollar signs.
- Kotlin variable names should start with a letter, $ or underscores
- Kotlin variables are case sensitive which means Kotak and KOTAK are two different variables.
- Kotlin variable can not have any white space or other control characters.
- Kotlin variable can not have names like var, val, String, Int because they are reserved keywords in Kotlin.

Kotlin Data Types: Kotlin data type is a classification of data which tells the compiler how the programmer intends to use the data. For example, Kotlin data could be numeric, string, boolean etc. Kotlin treats everything as an object which means that we can call member functions and properties on any variable. Kotlin is a statically typed language, which means that the data type of every expression should be known at compile time. Kotlin built in data type can be categorized as: Number, Character, String, Boolean, and Array.

Kotlin number data types are used to define variables which hold numeric values and they are divided into two groups: (a) Integer types store whole numbers, positive or negative (b) Floating point types represent numbers with a fractional part, containing one or more decimals. Following table list down all the Kotlin number data types, keywords to define their variable types, size of the memory taken by the variables, and a value range which can be stored in those variables.

Data Type	Size (bits)	Data Range
Byte	8 bit	-128 to 127
Short	16 bit	-32768 to 32767
Int	32 bit	-2,147,483,648 to 2,147,483,647
Long	64 bit	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Float	32 bit	1.40129846432481707e-45 to 3.40282346638528860e+38
Double	64 bit	4.94065645841246544e-324 to 1.79769313486231570e+308

Kotlin character data type is used to store a single character and they are represented by the type **Char** keyword. A Char value must be surrounded by single quotes, like 'A' or '1'. The String data type is used to store a sequence of characters. String values must be surrounded by double quotes (" ") or triple quote (""" """). We have two kinds of string available in Kotlin - one is called Escaped String and another is called **Raw String**. Escaped string is declared within double quote (" ") and may contain escape characters like '\n', '\t', '\b' etc. Raw string is declared within triple quote (""" """) and may contain multiple lines of text without any escape characters. Boolean is very simple like other programming languages. We have only two values for **Boolean** data type - either true or false. **To convert a numeric data type to another type, Kotlin provides a set of functions: toByte(), toShort(), toInt(), toLong(), toFloat(), toDouble(), and toChar()**

Kotlin Arrays: Arrays are used to store multiple items of the same data-type in a single variable, such as an integer or string under a single variable name. For example, if we need to store names of 1000 employees, then instead of creating 1000 different string variables, we can simply define an array of string whose capacity will be 1000. Like any other modern programming languages, Kotlin also supports arrays and provide a wide range of array properties and support functions to manipulate arrays. To create an array in Kotlin, we use the arrayOf() function, and place the values in a comma-separated list inside it:

```
val fruits = arrayOf("Apple", "Mango", "Banana", "Orange")
```

To create arrays of primitive data type other factory methods available for creating arrays: intArrayOf(), byteArrayOf(), charArrayOf(), shortArrayOf(), and longArrayOf().

Kotlin Operators: An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Kotlin is rich in built-in operators and provide the following types of operators: Arithmetic Operators, Relational Operators, Assignment Operators, Unary Operators, Logical Operators, and Bitwise Operations.

(a) Kotlin Arithmetic Operators: Kotlin arithmetic operators are used to perform basic mathematical operations such as addition, subtraction, multiplication and division etc.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

(b) Kotlin Relational Operators: Kotlin relational (comparison) operators are used to compare two values, and returns a Boolean value: either true or false.

Operator	Name	Example
>	greater than	x > y
<	less than	x < y
>=	greater than or equal to	x >= y
<=	less than or equal to	x <= y
==	is equal to	x == y
!=	not equal to	x != y

(c) Kotlin Assignment Operators: Kotlin assignment operators are used to assign values to variables.

Operator	Example	Expanded Form
=	x = 10	x = 10
+=	x += 10	x = x - 10
-=	x -= 10	x = x - 10
*=	x *= 10	x = x * 10
/=	x /= 10	x = x / 10
%=	x %= 10	x = x % 10

(d) Kotlin Unary Operators: The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

Operator	Name	Example
+	unary plus	+x
-	unary minus	-x
++	increment by 1	++x
--	decrement by 1	--x
!	inverts the value of a boolean	!x

(e) Kotlin Logical Operators: Kotlin logical operators are used to determine the logic between two variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both operands are true	x && y
||	Logical or	Returns true if either of the operands is true	x || y
!	Logical not	Reverse the result, returns false if the operand is true	!x

(f) Kotlin Bitwise Operations: Kotlin does not have any bitwise operators but Kotlin provides a list of helper functions to perform bitwise operations.

Function	Description	Example
shl (bits)	signed shift left	x.shl(y)
shr (bits)	signed shift right	x.shr(y)
ushr (bits)	unsigned shift right	x.ushr(y)
and (bits)	bitwise and	x.and(y)
or (bits)	bitwise or	x.or(y)
xor (bits)	bitwise xor	x.xor(y)
inv()	bitwise inverse	x.inv()

Kotlin Ranges: Kotlin range is defined by its two endpoint values which are both included in the range. Kotlin ranges are created with rangeTo() function, or simply using downTo or (. .) operators. The main operation on ranges is contains, which is usually used in the form of in and !in operators.

Kotlin Control Flow

Kotlin flow control statements determine the next statement to be executed. For example, the statements if-else, if, and when are for **Decision Making**. And while, for, and do are for **Loop Control** flow statements. The following description describe each in details:

If…else: Kotlin if...else expressions works like an if...else expression in any other Modern Computer Programming. So let's start with our traditional if...else statement available in Kotlin. The syntax of a traditional if...else expression is as follows:

```
if (condition) {
   // code block A to be executed if condition is true
} else {
  // code block B to be executed if condition is false
}
```

Here if statement is executed and the given condition is checked. If this condition is evaluated to true then code block A is executed, otherwise program goes into else part and code block B is executed. Kotlin if...else can also be used as an expression which returns a value and this value can be assigned to a variable. Below is a simple syntax of Kotlin if...else expression:

```
val result = if (condition) {
   // code block A to be executed if condition is true
} else {
  // code block B to be executed if condition is false
}
```

If you're using if as an expression, for example, for returning its value or assigning it to a variable, the else branch is mandatory. We can use else if condition to specify more then one condition test known as **ladder if** expression. Kotlin allows to put an if expression inside another if expression. This is called **nested if** expression

When: Consider a situation when you have large number of conditions to check. Though you can use if..else if expression to handle the situation, but Kotlin provides when expression to handle the situation in nicer way. Using when expression is far easy and more clean in comparison to writing many if...else if expressions. Kotlin when expression evaluates a section of code among many alternatives as explained in below example. Kotlin when matches its argument against all branches sequentially until some branch condition is satisfied. Kotlin when expression is similar to the switch statement in C, C++ and Java. Let's see a simple example of when expression.

```
fun main(args: Array<String>){
    var number = 4
    var numberProvided = when(number) {
        1 -> "One"
        2 -> "Two"
        3 -> "Three"
        4 -> "Four"
        5 -> "Five"
        else -> "invalid number"
    }
    println("You provide $numberProvided")
}
```

For Loop: Kotlin for loop iterates through anything that provides an iterator ie. that contains a countable number of values, for example arrays, ranges, maps or any other collection available in Kotlin. Kotlin for loop is equivalent to the foreach loop in languages like C#. Kotlin does not provide a conventional for loop which is available in C, C++ and Java etc. The syntax of the Kotlin for loop is as follows:

```
for (item in collection) {
    // body of loop
}
```

We will study Kotlin Ranges in a separate chapter, for now you should know that Kotlin Ranges provide iterator, so we can iterate through a range using for loop. Following is an example where the loop iterates through the range and prints individual item. To iterate over a range of numbers, we will use a range expression:

```
fun main(args: Array<String>) {
   for (item in 1..5) {
      println(item)
   }
}
```

While Loop: Kotlin while loop executes its body continuously as long as the specified condition is true. Kotlin while loop is similar to Java while loop. The syntax of the Kotlin while loop is as follows:

```
while (condition) {
    // body of the loop
}
```

When Kotlin program reaches the while loop, it checks the given condition, if given condition is true then body of the loop gets executed, otherwise program starts executing code available after the body of the while loop.

do...while Loop: The do..while is similar to the while loop with a difference that the this loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested. The syntax of the Kotlin do...while loop is as follows:

```
do{
    // body of the loop
}while( condition )
```

When Kotlin program reaches the do...while loop, it directly enters the body of the loop and executes the available code before it checks for the given condition. If it finds given condition is true, then it repeats the execution of the loop body and continue as long as the given condition is true.

Break Statement: Kotlin break statement is used to come out of a loop once a certain condition is met. This loop could be a for, while or do...while loop.

Continue Statement: The Kotlin continue statement breaks the loop iteration in between (skips the part next to the continue statement till end of the loop) and continues with the next iteration in the loop.

Data Structures (Collections)

Collections are a common concept for most programming languages. A collection usually contains a number of objects of the same type and Objects in a collection are called elements or items. The Kotlin Standard Library provides a comprehensive set of tools for managing collections. The following collection types are relevant for Kotlin:

- **Kotlin List** - List is an ordered collection with access to elements by indices. Elements can occur more than once in a list.
- **Kotlin Set** - Set is a collection of unique elements which means a group of objects without repetitions.
- **Kotlin Map** - Map (or dictionary) is a set of key-value pairs. Keys are unique, and each of them maps to exactly one value. Kotlin provides two types of collection: Immutable Collection and Mutable Collection.

Immutable Collection or simply calling a Collection interface provides read-only methods which means once a collection is created, we can not change it because there is no method available to change the object created.

Collection Types	Methods of Immutable Collection
List	listOf() listOf<T>()
Map	mapOf()
Set	setOf()

Mutable collections provides both read and write methods.

Collection Types	Methods of Immutable Collection

List	ArrayList<T>() arrayListOf() mutableListOf()
Map	HashMap hashMapOf() mutableMapOf()
Set	hashSetOf() mutableSetOf()

Kotlin Functions

Kotlin is a statically typed language, hence, functions play a great role in it. We are pretty familiar with function, as we are using function throughout our examples in our last chapters. A function is a block of code which is written to perform a particular task. Functions are supported by all the modern programming languages and they are also known as methods or subroutines. At a broad level, a function takes some input which is called parameters, perform certain actions on these inputs and finally returns a value.

Kotlin Built-in Functions: Kotlin provides a number of built-in functions, we have used a number of buil-in functions in our examples. For example print() and println() are the most commonly used built-in function which we use to print an output to the screen.

User-Defined Functions: Kotlin allows us to create our own function using the keyword fun. A user defined function takes one or more parameters, perform an action and return the result of that action as a value. Syntax:

```
fun functionName([parameters])[:return type]{
   // body of function
   [return <value>]
}
```

Once we defined a function, we can call it any number of times whenever it is needed. **Recursion functions** are useful in many scenerios like calculating factorial of a number or generating fibonacci series. Kotlin supports recursion which means a Kotlin function can call itself. For example following is a Kotlin program to calculate the factorial of number 10:

```
fun main(args: Array<String>) {
   val a = 4
   val result = factorial(a)
   println( result )
}

fun factorial(a:Int):Int{
   val result:Int
   if( a <= 1){
      result = a
   }else{
      result = a*factorial(a-1)
   }
   return result
}
```

Object Oriented Programming

Kotlin supports both functional and object-oriented programming. While talking about functional features of Kotlin then we have concepts like functions, higher-order functions and lambdas etc. which represent Kotlin as a functional language. Kotlin also supports Object Oriented Programming (OOP) and provides features such as abstraction, encapsulation, inheritance, etc. This tutorial will teach you all the Kotlin OOP features in simple steps. Object oriented programming (OOP) allows us to solve the complex problems by using the objects.

Classes: A class is a blueprint for the objects which defines a template to be used to create the required objects. Classes are the main building blocks of any Object Oriented Programming language. A Kotlin class is defined using the class keyword. A Kotlin class declaration is similar to Java Programmig which consists of a class header and a class body surrounded by curly braces. Following is the syntax to create a Kotlin Class:

```
class ClassName { // Class Header
   // Variables or data members
   // Member functions or Methods
   ...
   ...
}
```

By default, Kotlin classes are public and we can control the visibility of the class members using different modifiers that we will learn in Visibility Modifiers.

Objects: The objects are created from the Kotlin class and they share the common properties and behaviours defined by a class in form of data members (properties) and member functions (behaviours) respectively. The syntax to declare an object of a class is:

```
var varName = ClassName()
```

We can access the properties and methods of a class using the . (dot) operator as shown below:

```
varName.property = <Value>
varName.functionName()
```

Method Over: Kotlin requires explicit modifiers for overridable members and overrides:

```
open class Shape {
    open fun draw() { /*...*/ }
    fun fill() { /*...*/ }
}
class Circle() : Shape() {
    override fun draw() { /*...*/ }
}
```

The override modifier is required for Circle.draw(). If it were missing, the compiler would complain. If there is no open modifier on a function, like Shape.fill(), declaring a method with the same signature in a subclass is not allowed, either with override or without it. The open modifier has no effect when added to members of a final class – a class without an open modifier.

Constructor: A Kotlin constructor is a special member function in a class that is invoked when an object is instantiated. Whenever an object is created, the defined constructor is called automatically which is used to initialize the properties of the class. Every Kotlin class needs to have a constructor and if we do not define it, then the compiler generates a default constructor. A Kotlin class can have following two constructors: Primary Constructor, and Second Constructors. A Kotlin class can have a primary constructor and one or more additional secondary constructors. The Kotlin primary constructor initializes the class, whereas the secondary constructor helps to include some extra logic while initializing the class.

Primary Constructor: The primary constructor can be declared at class header level as shown in the following example.

```
class Person constructor(val firstName: String, val age: Int) {
   // class body
}
```

The constructor keyword can be omitted if there is no annotations or access modifiers specified like public, private or protected. The primary constructor cannot contain any code. Initialization code can be placed in initializer blocks prefixed with the init keyword. Thee could be more than one init blocks and during the initialization of an instance, the initializer blocks are executed in the same order as they appear in the class body, interleaved with the property initializers.

Secondary Constructor: As mentioned earlier, Kotlin allows to create one or more secondary constructors for your class. This secondary constructor is created using the constructor keyword. It is required whenever you

want to create more than one constructor in Kotlin or whenever you want to include more logic in the primary constructor and you cannot do that because the primary constructor may be called by some other class.

Inheritance

Inheritance can be defined as the process where one class acquires the members (methods and properties) of another class. With the use of inheritance the information is made manageable in a hierarchical order. A class which inherits the members of other class is known as subclass (derived class or child class) and the class whose members are being inherited is known as superclass (base class or parent class).

Inheritance is one of the key features of object-oriented programming which allows user to create a new class from an existing class. Inheritance we can inherit all the features from the base class and can have additional features of its own as well. All classes in Kotlin have a common superclass called Any, which is the default superclass for a class with no supertypes declared:

```
class Example // Implicitly inherits from Any
```

Kotlin superclass Any has three methods: equals(), hashCode(), and toString(). Thus, these methods are defined for all Kotlin classes. Everything in Kotlin is by default final, hence, we need to use the keyword open in front of the class declaration to make it inheritable for other classes. Kotlin uses operator ":" to inherit a class. Take a look at the following example of inheritance.

```
open class ABC {
   fun think () {
      println("Hey!! i am thiking ")
   }
}
class BCD: ABC(){ // inheritence happend using default constructor
}

fun main(args: Array<String>) {
   var  a = BCD()
   a.think()
}
```

Abstract

A Kotlin abstract class is similar to Java abstract class which can not be instantiated. This means we cannot create objects of an abstract class. However, we can inherit subclasses from a Kotlin abstract class. A Kotlin abstract class is declared using the abstract keyword in front of class name. The properties and methods of an abstract class are non-abstract unless we explictly use abstract keyword to make them abstract. If we want to override these members in the child class then we just need to use override keyword infront of them in the child class.

```
abstract class Person {
   var age: Int = 40

   abstract fun setAge()    // Abstract Method

   fun getAge() {           // Non-Abstract Method
      return age
   }
}
```

Abstract classes are always open. You do not need to explicitly use open keyword to inherit subclasses from them. To summarise, A Kotlin class which contains the abstract keyword in its declaration is known as abstract class.

- Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.

- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Interface: Kotlin, the interface works exactly similar to Java 8, which means they can contain method implementation as well as abstract methods declaration. An interface can be implemented by a class in order to use its defined functionality. We have already introduced an example with an interface in Chapter 6 - section "anonymous inner class". In this chapter, we will learn more about it. The keyword "interface" is used to define an interface in Kotlin as shown in the following piece of code.

```
interface ExampleInterface {
   var myVar: String     // abstract property
   fun absMethod()       // abstract method
   fun sayHello() = "Hello there" // method with default
implementation
}
```

In the above example, we have created one interface named as "ExampleInterface" and inside that we have a couple of abstract properties and methods all together. Look at the function named "sayHello()", which is an implemented method. In the following example, we will be implementing the above interface in a class.

```
interface ExampleInterface  {
   var myVar: Int            // abstract property
   fun absMethod():String    // abstract method
   fun hello() {
      println("Hello there, Welcome to TutorialsPoint.Com!")
   }
}
class InterfaceImp : ExampleInterface {
   override var myVar: Int = 25
   override fun absMethod() = "Happy Learning "
}
fun main(args: Array<String>) {
   val obj = InterfaceImp()
   println("My Variable Value is = ${obj.myVar}")
   print("Calling hello(): ")
   obj.hello()
   print("Message from the Website-- ")
   println(obj.absMethod())
}
```

super and this keyword

super keyword is used to access methods of the parent class while this is used to access methods of the current class. this keyword is a reserved keyword in Kotlin i.e, we can't use it as an identifier. this is used to refer current-class's instance as well as static members. this can be used in various contexts as given below:

- to refer instance variable of current class
- to invoke or initiate current class constructor
- can be passed as an argument in the method call
- can be passed as argument in the constructor call
- can be used to return the current class instance

Visibility modifiers

Visibility **(Access) modifier** is used to restrict the usage of the variables, methods and class used in the application. Like other OOP programming language, this modifier is applicable at multiple places such as in the class header or method declaration. There are four access modifiers available in Kotlin.

Private: The classes, methods, and packages can be declared with a private modifier. Once anything is declared as private, then it will be accessible within its immediate scope. For instance, a private package can be accessible within that specific file. A private class or interface can be accessible only by its data members, etc.

Protected: Protected is another access modifier for Kotlin, which is currently not available for top level declaration like any package cannot be protected. A protected class or interface is visible to its subclass only.

Internal: Internal is a newly added modifier introduced in Kotlin. If anything is marked as internal, then that specific field will be in the internal field. An Internal package is visible only inside the module under which it is implemented. An internal class interface is visible only by other class present inside the same package or the module. In the following example, we will see how to implement an internal method.

Public: Public modifier is accessible from anywhere in the project workspace. If no access modifier is specified, then by default it will be in the public scope. In all our previous examples, we have not mentioned any modifier, hence, all of them are in the public scope. Following is an example to understand more on how to declare a public variable or method.

Important Questions from the Unit:

1. Explain Kotlin Architecture
2. Explain Data Types avaible in Kotlin
3. Explain Kotlin Collections
4. Explain Visibility Modifers of Kotlin
5. Write name rules of variable in Kotlin
6. Explain Abstract class in Kotlin with Example
7. Explain Interface of Kotlin with Example
8. Explain Exception Handling in Kotlin with Example

Unit – 2

Introduction to Android & Android Application Design

The Open Handset Alliance

Enter search advertising giant Google. Now a household name, Google has shown an interest in spreading its vision, its brand, its search and ad-revenue-based platform, and its suite of tools to the wireless marketplace. The company's business model has been amazingly successful on the Internet and, technically speaking, wireless isn't that different. With its user-centric, democratic design philosophies, Google has led a movement to turn the existing closely guarded wireless market into one where phone users can move between carriers easily and have unfettered access to applications and services. With its vast resources, Google has taken a broad approach, examining the wireless infrastructure from the FCC wireless spectrum policies to the handset manufacturers' requirements, application developer needs, and mobile operator desires. Next, Google joined with other like-minded members in the wireless community and posed the following question: What would it take to build a better mobile phone?

The **Open Handset Alliance (OHA)** was formed in November 2007 to answer that very question. The OHA is a business alliance comprised of many of the largest and most successful mobile companies on the planet. Its members include chip makers, handset manufacturers, software developers, and service providers. The entire mobile supply chain is well represented. Andy Rubin has been credited as the father of the Android platform. His company, Android Inc., was acquired by Google in 2005.Working together, OHA members, including Google, began developing a nonproprietary open standard platform based upon technology developed at Android Inc. that would aim to alleviate the aforementioned problems hindering the mobile community. The result is the Android project. To this day, most Android platform development is completed by Rubin's team at Google, where he acts as VP of Engineering and manages the Android platform roadmap.

The Android Platform

Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies. Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android. The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008. On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance. The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

Features of Android: After learning what is android, let's see the features of android. The important features of android are given below:

1) It is open-source.
2) Anyone can customize the Android Platform.
3) There are a lot of mobile applications that can be chosen by the consumer.
4) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.

It provides support for messaging services (SMS and MMS), web browser, storage (SQLite), connectivity (GSM, CDMA, Blue Tooth, Wi-Fi etc.), media, handset layout etc.

Android Versions

The development of the Android operating system was started in 2003 by Android, Inc. Later on, it was purchased by Google in 2005. The beta version of Android OS was released on November 5, 2007, while the software development kit (SDK) was released on November 12, 2007. The first Android mobile was publicly released with Android 1.0 of the T-Mobile G1 (aka HTC Dream) in October 2008. Google announced in August 2019 that they were ending the confectionery scheme, and they use numerical ordering for future Android versions. The first Android version which was released under the numerical order format was Android 1.0. The following table shows the Android versions, name, and API level:

Code name	Version numbers	API level	Release date
No codename	1.0	1	September 23, 2008
No codename	1.1	2	February 9, 2009
Cupcake	1.5	3	April 27, 2009
Donut	1.6	4	September 15, 2009
Eclair	2.0 - 2.1	5 - 7	October 26, 2009
Froyo	2.2 - 2.2.3	8	May 20, 2010
Gingerbread	2.3 - 2.3.7	9 - 10	December 6, 2010
Honeycomb	3.0 - 3.2.6	11 - 13	February 22, 2011
Ice Cream Sandwich	4.0 - 4.0.4	14 - 15	October 18, 2011
Jelly Bean	4.1 - 4.3.1	16 - 18	July 9, 2012
KitKat	4.4 - 4.4.4	19 - 20	October 31, 2013
Lollipop	5.0 - 5.1.1	21- 22	November 12, 2014
Marshmallow	6.0 - 6.0.1	23	October 5, 2015
Nougat	7.0	24	August 22, 2016
Nougat	7.1.0 - 7.1.2	25	October 4, 2016
Oreo	8.0	26	August 21, 2017
Oreo	8.1	27	December 5, 2017
Pie	9.0	28	August 6, 2018
Android 10	10.0	29	September 3, 2019
Android 11	11	30	September 8, 2020

Android SDK

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. The Android SDK includes the following:

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

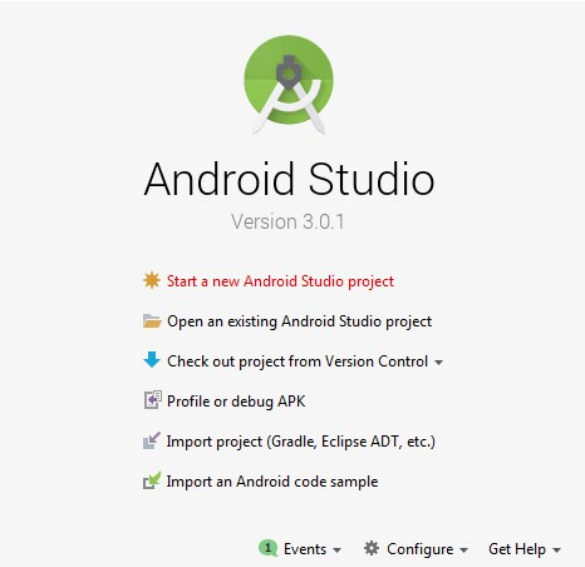
Before we begin building a sample android application, let's see some terminologies and building blocks of android.

Categories of Android applications: There are many android applications in the market. The top categories are: Entertainment, Tools, Communication, Productivity, Personalization, Music and Audio, Social, Media and Video, Travel and Local etc.

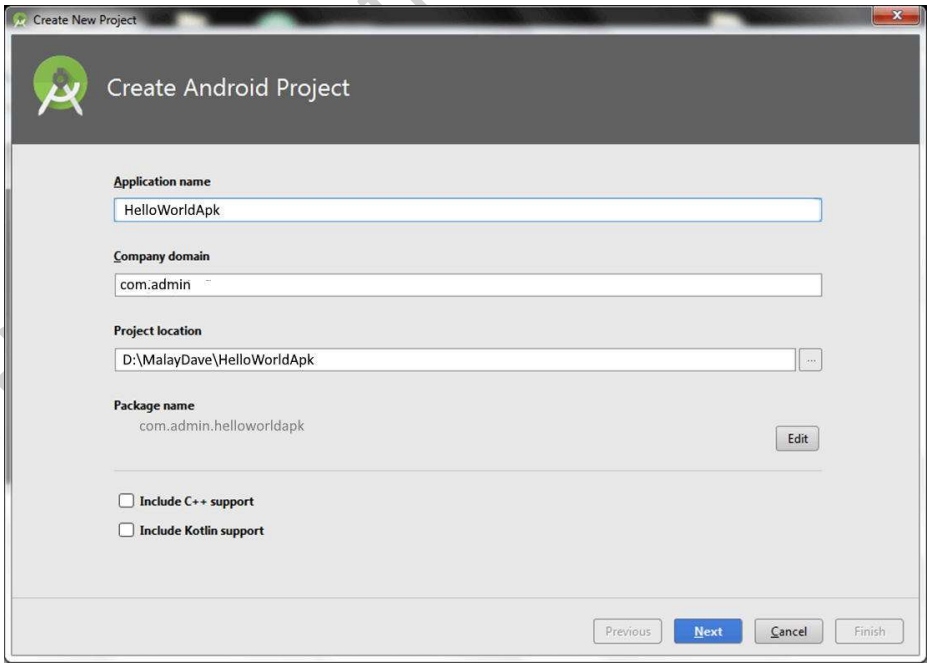
Building a sample Android application

1) Create the New Android project: For creating the new android studio project:

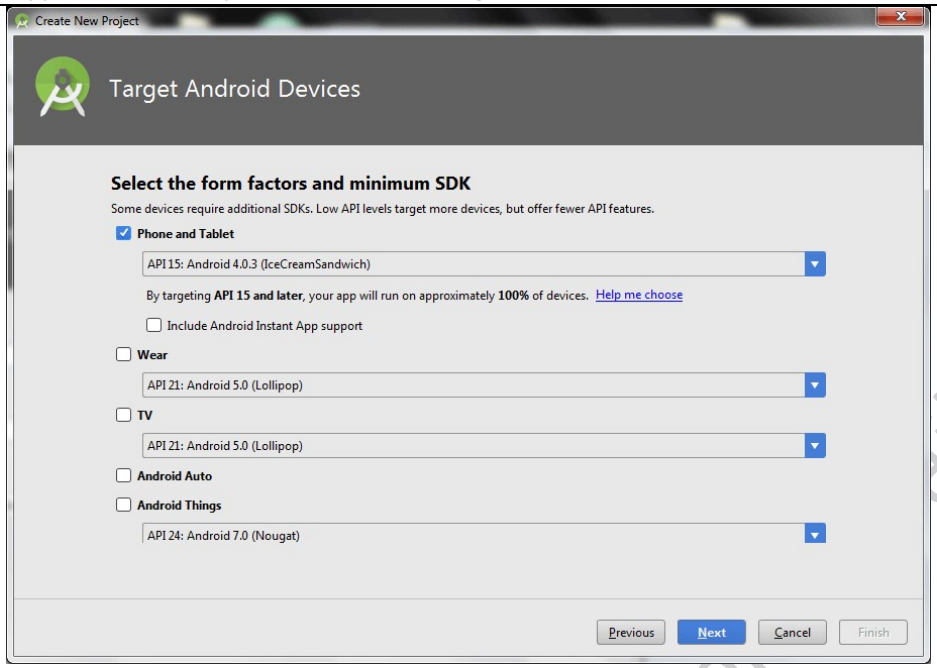
2) Select Start a new Android Studio project



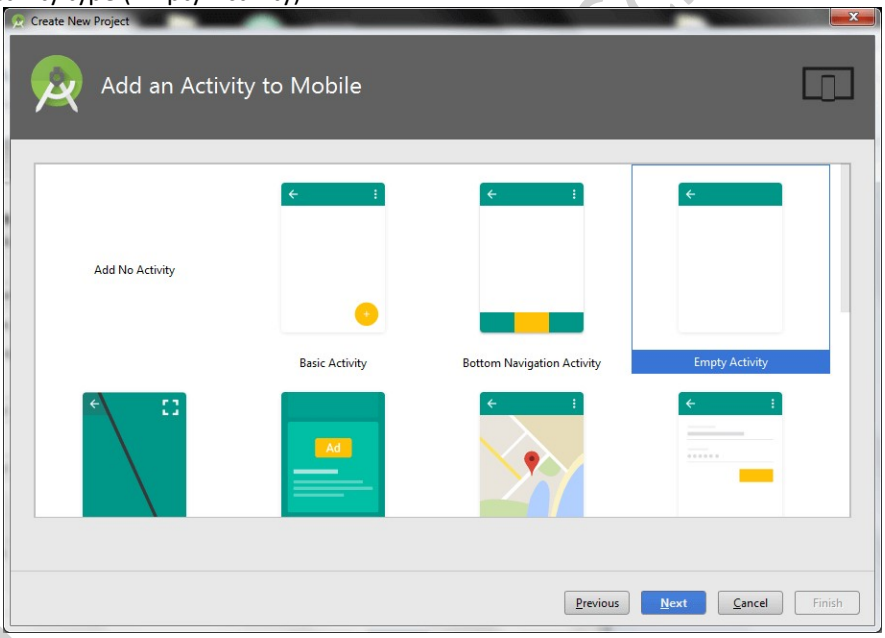
3) Provide the following information: Application name, Company domain, Project location and Package name of application and click next.



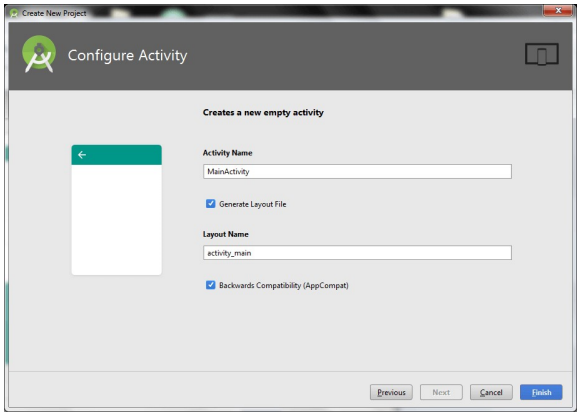
4) Select the API level of application and click next.



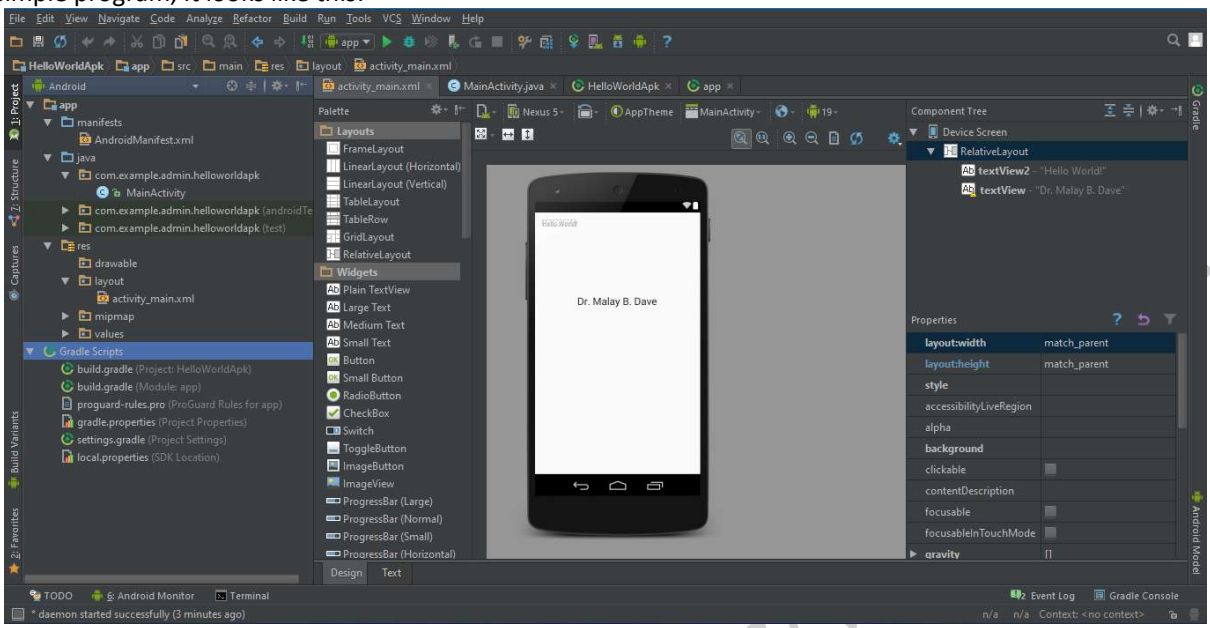
5) Select the Activity type (Empty Activity).



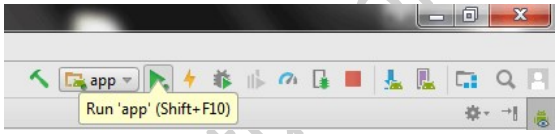
6) Provide the Activity Name and click finish.



After finishing the Activity configuration, Android Studio auto generates the activity class and other required configuration files. Now an android project has been created. You can explore the android project and see the simple program, it looks like this:



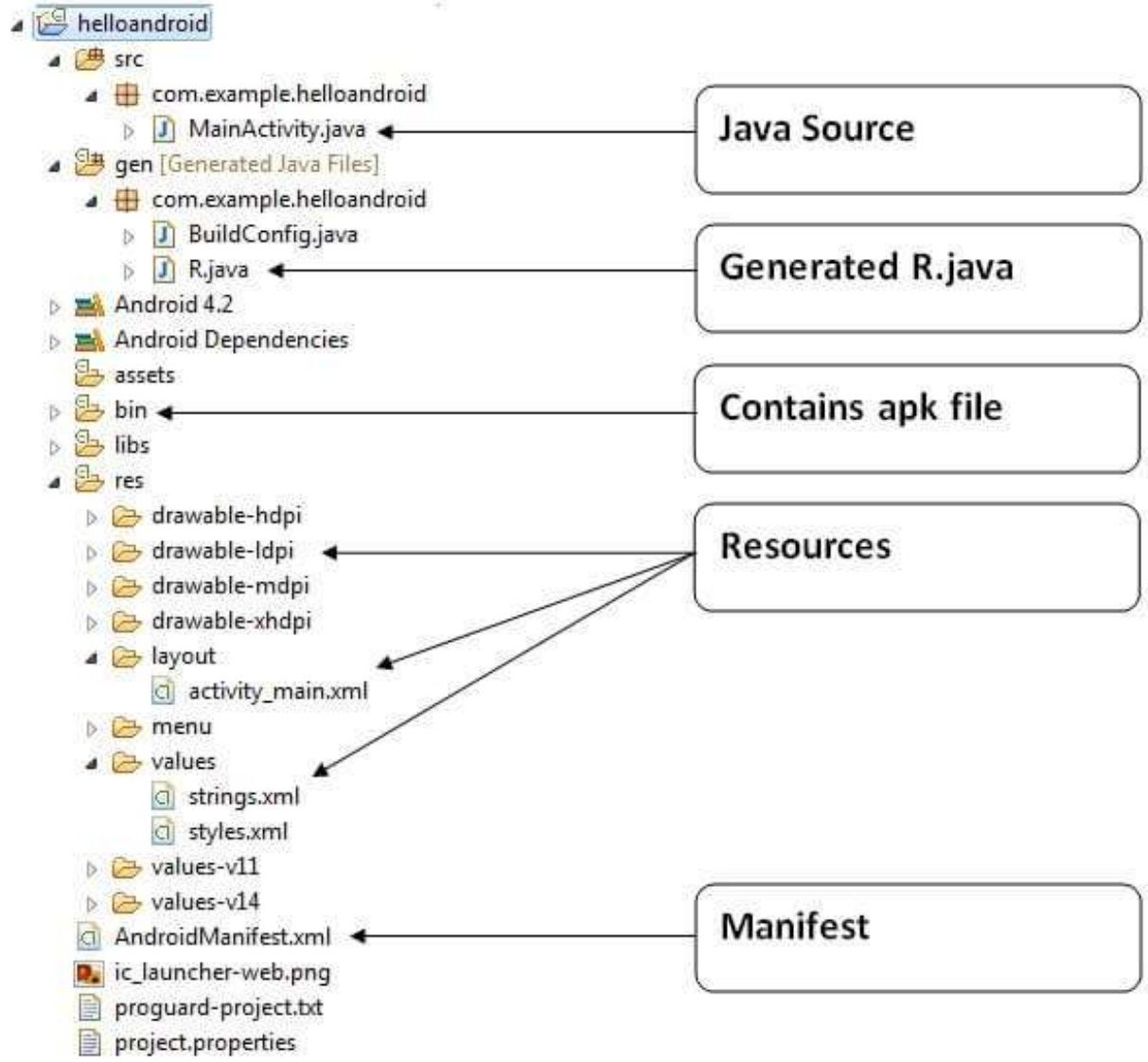
7) Run the android application, To run the android application, click the run icon on the toolbar or simply press Shift + F10.



The android emulator might take 2 or 3 minutes to boot. So please have patience. After booting the emulator, the android studio installs the application and launches the activity.

Anatomy of an Android applications

Before you run your app, you should be aware of a few directories and files in the Android project



Folder, File & Description
Java This contains the **.java** source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
res/drawable-hdpi This is a directory for drawable objects that are designed for high-density screens.
res/layout This is a directory for files that define your app's user interface.
res/values This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
AndroidManifest.xml

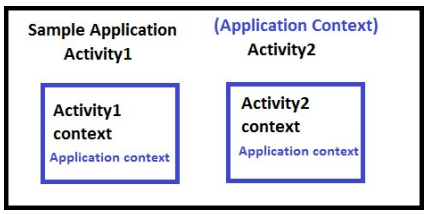
This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
Build.gradle This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

Android - Application Components

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact. There are following four main components that can be used within an Android application:

- **Activities:** They dictate the UI and handle the user interaction to the smart phone screen.
- **Services:** They handle background processing associated with an application.
- **Broadcast Receivers:** They handle communication between Android OS and applications.
- **Content Providers**: They handle data and database management issues.
- **Android Virtual Device (AVD):** It is used to test the android application without the need for mobile or tablet etc. It can be created in different configurations to emulate different types of real devices. Now that you've built your first Android app, let's talk about the different components that make up an Android app.

Android terminologies



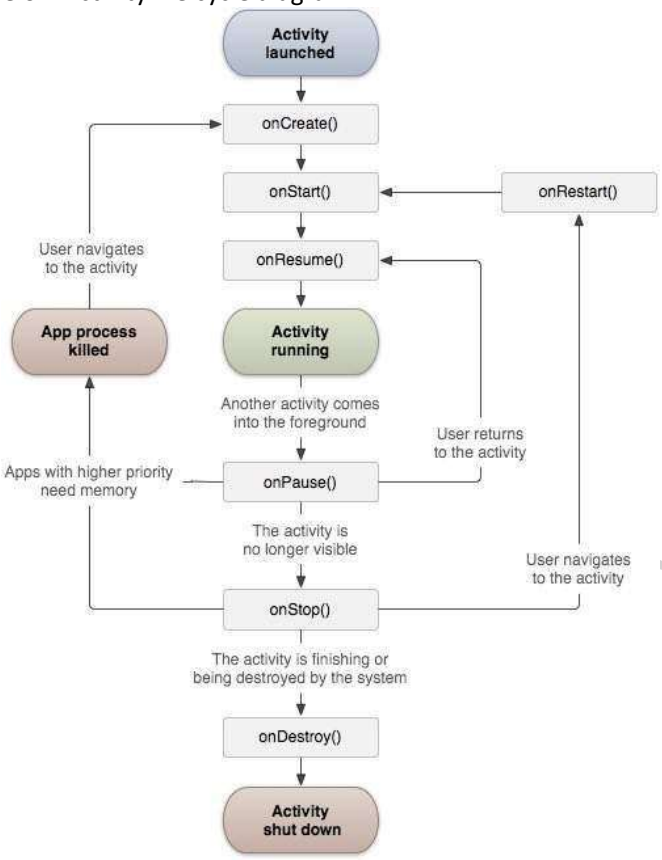
Context: There are mainly two types of context are available in Android, first is Application Context and Activity Context. It can be seen in the above image that in "Sample Application", nearest Context is Application Context. In "Activity1" and "Activity2", both Activity Context (Here it is Activity1 Context for Activity1 and Activity2 Context for Activity2) and Application Context.The nearest Context to both is their Activity Context only.

- **Application Context:** This context is tied to the life cycle of an application. Mainly it is an instance that is a singleton and can be accessed via getApplicationContext(). Some use cases of Application Context are: If it is necessary to create a singleton object, During the necessity of a library in an activity. getApplicationContext(): This method is generally used for the application level and can be used to refer to all the activities. For example, if we want to access a variable throughout the android app, one has to use it via getApplicationContext(). It is used to return the context which is linked to the Application which holds all activities running inside it. When we call a method or a constructor, we often have to pass a context and often we use "this" to pass the activity context or "getApplicationContext" to pass the application context.

- **Activity Context:** It is the activity context meaning each and every screen got an activity. For example, EnquiryActivity refers to EnquiryActivity only and AddActivity refers to AddActivity only. It is tied to the life cycle of activity. It is used for the current context. The method of invoking the Activity Context is getContext(). Some use cases of Activity Context are: The user is creating an object whose lifecycle is attached to an activity. Whenever inside an activity for UI related kind of operations like toast, dialogue, etc.,

Activities: An activity represents a single screen with a user interface just like window or frame of Java.Android activity is the subclass of ContextThemeWrapper class. If you have worked with C, C++ or Java programming language then you must have seen that your program starts from main() function. Very similar way, Android system initiates its program with in an Activity starting with a call on onCreate() callback method. There is a

sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram:



Services: A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states –

State & Description
Started A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
Bound A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

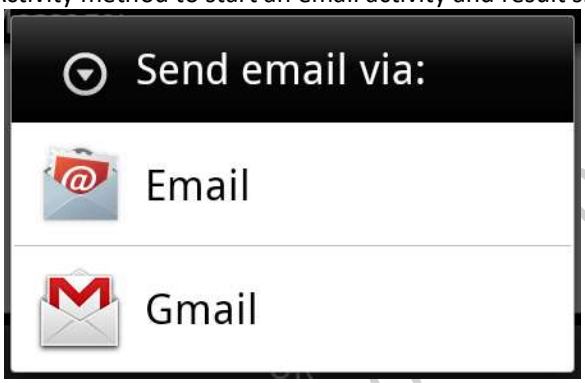
Intents: There are following two types of intents supported by Android: Implicit and Explicit. **Implicit Intents** do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. **Explicit intent** going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.

An Android Intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service.

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed. For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate chooser, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

```
Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));
email.putExtra(Intent.EXTRA_EMAIL, recipients);
email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
startActivity(Intent.createChooser(email, "Choose an email client
from..."));
```

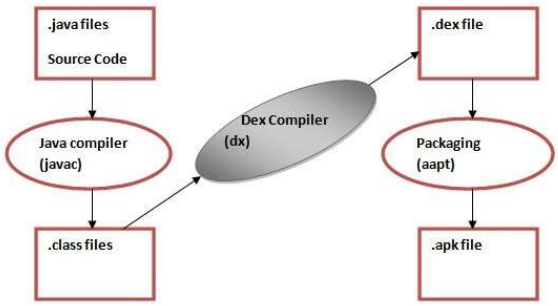
Above syntax is calling startActivity method to start an email activity and result should be as shown below –



Receiving and Broadcasting: Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action. There are following two important steps to make BroadcastReceiver works for the system broadcasted intents: Creating the Broadcast Receiver. And Registering Broadcast Receiver. There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

Dalvik Virtual Machine (DVM)

As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well. The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for memory, battery life and performance. Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein. The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file. Let's see the compiling and packaging process from the source file shown in the figure.



Android Manifest File and its common settings

Whatever component you develop as a part of your application, you must declare all its components in a manifest.xml which resides at the root of the application project directory. This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file –

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tutorialspoint7.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

      <activity android:name=".MainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category
android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Here <application>...</application> tags enclosed the components related to the application. Attribute android:icon will point to the application icon available under res/drawable-hdpi. The application uses the image named ic_launcher.png located in the drawable folders. The <activity> tag is used to specify an activity and android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

The action for the intent filter is named android.intent.action.MAIN to indicate that this activity serves as the entry point for the application. The category for the intent-filter is named android.intent.category.LAUNCHER to indicate that the application can be launched from the device's launcher icon. The @string refers to the strings.xml file explained below. Hence, @string/app_name refers to the app_name string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application. Following is the list of tags which you will use in your manifest file to specify different Android application components −

- <activity>elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

Android R.java file

Android R.java is an auto-generated file by aapt (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory. If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component. If you delete R.jar file, android creates it automatically. It includes a lot of static nested classes such as menu, id, layout, attr, drawable, string etc.

Using Intent Filter, Permissions

You have seen how an Intent has been used to call another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use <intent-filter> element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver. Following is an example of a part of AndroidManifest.xml file to specify an activity com.example.My Application.CustomActivity which can be invoked by either of the two mentioned actions, one category, and one data –

```
<activity android:name=".CustomActivity"
   android:label="@string/app_name">
```

```
        <intent-filter>
           <action android:name="android.intent.action.VIEW" />
           <action android:name="com.example.My Application.LAUNCH" />
           <category android:name="android.intent.category.DEFAULT" />
           <data android:scheme="http" />
        </intent-filter>
     </activity>
```

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the android.intent.action.VIEW, or using the com.example.My Application.LAUNCH action provided their category is android.intent.category.DEFAULT. The <data> element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"

Managing Application resources in a hierarchy

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under res/ directory of the project. This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications. Organize resource in android Directory and Resource Type:

anim/: XML files that define property animations. They are saved in res/anim/ folder and accessed from the R.anim class.

color/: XML files that define a state list of colors. They are saved in res/color/ and accessed from the R.color class.

drawable/: Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the R.drawable class.

layout/: XML files that define a user interface layout. They are saved in res/layout/ and accessed from the R.layout class.

menu/: XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the R.menu class.

```
MyProject/
   app/
      manifest/
         AndroidManifest.xml
   java/
      MyActivity.java
      res/
         drawable/
            icon.png
         layout/
            activity_main.xml
            info.xml
         values/
            strings.xml
```

raw/: Arbitrary files to save in their raw form. You need to call Resources.openRawResource() with the resource ID, which is R.raw.filename to open such raw files.

values/: XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory −

- arrays.xml for resource arrays, and accessed from the R.array class.
- integers.xml for resource integers, and accessed from the R.integer class.
- bools.xml for resource boolean, and accessed from the R.bool class.
- colors.xml for color values, and accessed from the R.color class.
- dimens.xml for dimension values, and accessed from the R.dimen class.
- strings.xml for string values, and accessed from the R.string class.
- styles.xml for styles, and accessed from the R.style class.

xml/: Arbitrary XML files that can be read at runtime by calling Resources.getXML(). You can save various configuration files here which will be used at run time.

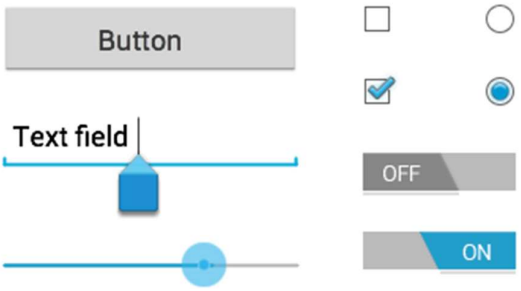
Important Questions from the Unit:

1. Write a Note on OHA
2. Explain Anatomy of Android Application
3. Explain Activity Life Cycle
4. Explain Context
5. Explain Application Component
6. Explain Android mainifest file
7. Explain Application Resource
8. Explain Intent and Intent Filter

Unit – 3

Android User Interface Design

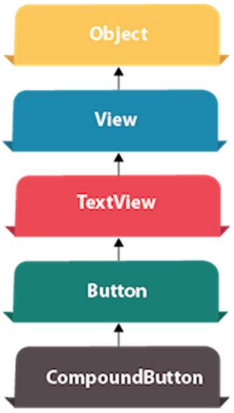
User Interface Screen elements



In android UI (android widgets) or input controls are the interactive or View components that are used to design the user interface of an application. In android we have a wide variety of UI or input controls available, those are TextView, EditText, Buttons, Checkbox, Progressbar, Spinners, etc. Following is the pictorial representation of user interface (UI) or input controls in android application. Generally, in android the user interface of an app is made with a collection of View and ViewGroup objects. The View is a base class for all UI components in android and it is used to create interactive UI components such as TextView, EditText, Checkbox, Radio Button, etc. and it is responsible for event handling and drawing. The ViewGroup is a subclass of View and it will act as a base class for layouts and layout parameters. The ViewGroup will provide invisible containers to hold other Views or ViewGroups and to define the layout properties. In android, we can define a UI or input controls in two ways, those are:

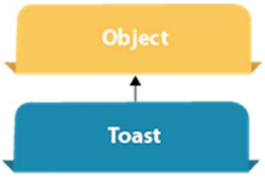
- Declare UI elements in XML
- Create UI elements at runtime

The android framework will allow us to use either or both of these methods to define our application's UI. Declare UI Elements in XML. In android, we can create layouts same as web pages in HTML by using default Views and ViewGroups in the XML file. The layout file must contain only one root element, which must be a View or ViewGroup object. Once we define the root element, then we can add additional layout objects or widgets as a child elements to build View hierarchy that defines our layout.



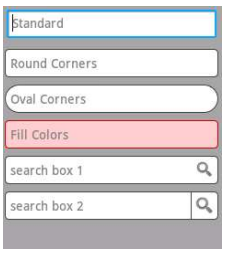
Button: Android Button represents a push-button. The **android.widget.Button** is subclass of TextView class and CompoundButton is the subclass of Button class. There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc. We can perform action on button using different types such as calling listener on button or adding onClick property of button in activity's xml file. The following is the sample code to create Button:

```
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="109dp"
        android:text="Click"
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="266dp" />
```



Toast: Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime. The **android.widget.Toast class** is the subclass of java.lang.Object class. You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast. Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction. For example:

```
Toast.makeText(getApplicationContext(),"Hello",Toast.LENGTH_SHORT).show();
```



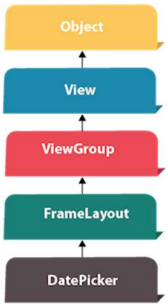
EditText: Android EditText is a subclass of TextView. To use EditText in Android application we need to import **android.widget.EditText class.** EditText is used for entering and modifying text. While using EditText width, we must specify its input type in inputType property of EditText which configures the keyboard according to input. Style of EditText shown in the figure. Following is the sample code to create EditText:

```
<EditText
      android:id="@+id/edittext"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/button"
      android:layout_below="@+id/textView1"
      android:layout_marginTop="61dp"
      android:ems="10"
      android:text="@string/enter_text" android:inputType="text" />
```

TextView: A TextView displays text to the user and optionally allows them to edit it. To use TextView in Android application we need to import **android.widget.TextView class**. A TextView is a complete text editor, however the basic class is configured to not allow editing. The following is the sample code to create TextView:

```
<TextView
      android:id="@+id/text_id"
      android:layout_width="300dp"
      android:layout_height="200dp"
      android:capitalize="characters"
      android:text="hello_world"
      android:textColor="@android:color/holo_blue_dark"
      android:textColorHighlight="@android:color/primary_text_dark"
      android:layout_centerVertical="true"
      android:layout_alignParentEnd="true"
      android:textSize="50dp"/>
```

DatePicker: Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality android provides DatePicker and DatePickerDialog components. DatePickerDialog is a simple dialog containing DatePicker. In order to show DatePickerDialog, you have to pass the DatePickerDialog id to showDialog(id_of_dialog) method. Its syntax is given below –




```
showDialog(999);
```

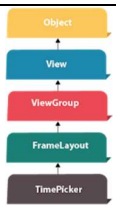
On calling this showDialog method, another method called onCreateDialog gets automatically called. So, we have to override that method too. Its syntax is given below –

```
@Override
protected Dialog onCreateDialog(int id) {
   // TODO Auto-generated method stub
   if (id == 999) {
      return new DatePickerDialog(this, myDateListener, year, month,
day);
   }
   return null;
}
```

The following is the sample code to create DatePicker:

```
<DatePicker
      android:id="@+id/datePicker"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_above="@+id/textView1"
      android:layout_centerHorizontal="true"
      android:layout_marginBottom="36dp" />
```

TimePicker: Android TimePicker widget is used to select date. It allows you to select time by hour and minute. You cannot select time by seconds. The **android.widget.TimePicker** is the subclass of FrameLayout class. The following is the sample code to create TimePicker:




```
<TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="36dp" />
```

ProgressBar: We can display the android progress bar dialog box to display the status of work being done e.g. downloading file, analyzing status of work etc. In this example, we are displaying the progress dialog for dummy file download operation. Here we are using **android.app.ProgressDialog class** to show the progress bar. Android ProgressDialog is the subclass of AlertDialog class. The ProgressDialog class provides methods to work on progress bar like setProgress(), setMessage(), setProgressStyle(), setMax(), show() etc. The progress range of Progress Dialog is 0 to 10000. The following is the sample code:


```
ProgressDialog progressBar = new ProgressDialog(this);
progressBar.setCancelable(true);//you can cancel it by pressing back button
progressBar.setMessage("File downloading ...");
progressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressBar.setProgress(0);//initially progress is 0
progressBar.setMax(100);//sets the maximum value 100
progressBar.show();//displays the progress bar
```

ListView: Android ListView is a view which contains the group of items and displays in a scrollable list. ListView is implemented by importing **android.widget.ListView class**. ListView is a default scrollable which does not use other scroll view. ListView uses Adapter classes which add the content from data source (such as string array, array, database etc) to ListView. Adapter bridges data between an AdapterViews and other Views (ListView, ScrollView etc). The following is the sample code:

```
<ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
         />
```

GridView: In android GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter. Users can then select any grid item by clicking on it. GridView is default scrollable so we don't need to use ScrollView or anything else with GridView. GridView is widely used in android applications. To use GridView in android application we have to import **android.widget.GridView class**. An example of GridView is your default Gallery, where you have number of images displayed using grid. Following is the sample code:

```
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"/>
```

RadioButton and RadioGroup: RadioButton is a two states button which is either checked or unchecked. If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user. RadioButton is generally used with RadioGroup. **RadioGroup** contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked. We need to import **android.widget.RadioButton and android.widget.RadioGroup class**. Following is the sample code:

```
<RadioGroup
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/radioGroup">

        <RadioButton
            android:id="@+id/radioMale"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="  Male"
            android:layout_marginTop="10dp"
            android:checked="false"
            android:textSize="20dp" />

        <RadioButton
            android:id="@+id/radioFemale"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="   Female"
            android:layout_marginTop="20dp"
            android:checked="false"

            android:textSize="20dp" />
    </RadioGroup>
```

ImageButton: In Android, ImageButton is used to display a normal button with a custom image in a button. In simple words we can say, ImageButton is a button with an image that can be pressed or clicked by the users. By default it looks like a normal button with the standard button background that changes the color during different button states. We need to import **android.widget.ImageButton class** to create ImageButton in Android application. The following is sample code:

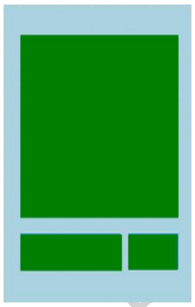
```
<ImageButton
android:id="@+id/simpleImageButton"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/home" />
```

Fragement: In Android, Fragment is a part of an activity which enable more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents a behaviour or a portion of user interface in an Activity. We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle. We need to import **android.app.Fragment class** in Android application to create Fragment. The following is the sample code:

```
<fragment
android:id="@+id/fragments"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

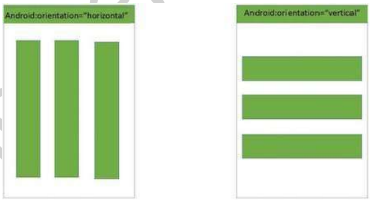
Designing User Interfaces with Layouts

The basic building block for user interface is a View object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc. The ViewGroup is a subclass of View and provides invisible container that hold other Views or other ViewGroups and define their layout properties. At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using View/ViewGroup objects or you can declare your layout using simple XML file main_layout.xml which is located in the res/layout folder of your project. There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.



Relative Layout: The Relative Layout is very flexible layout used in android for custom layout designing. It gives us the flexibility to position our component/view based on the relative or sibling component's position. Just because it allows us to position the component anywhere we want so it is considered as most flexible layout. For the same reason Relative layout is the most used layout after the Linear Layout in Android. It allow its child view to position relative to each other or relative to the container or another container. In Relative Layout, you can use "above, below, left and right" to arrange the component's position in relation to other component. Following is the sample code:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!—Design Other UI here-->

</RelativeLayout>
```



Linear Layout: Linear layout is a simple layout used in android for layout designing. In the Linear layout all the elements are displayed in linear fashion means all the childs/elements of a linear layout are displayed according to its orientation. The value for orientation property can be either horizontal or vertical. The following is the sample code:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <!—Design Other UI here-->

</LinearLayout>
```

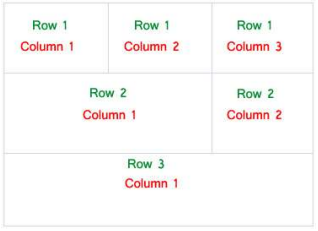
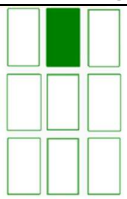


Table Layout: n Android, Table Layout is used to arrange the group of views into rows and columns. Table Layout containers do not display a border line for their columns, rows or cells. A Table will have as many columns as the row with the most cells. A table can also leave the cells empty but cells can't span the columns as they can in HTML(Hypertext markup language). The following is the sample code:

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:collapseColumns="0"> <!-- collapse the first column of the table
row-->
    <!-- first row of the table layout-->
    <TableRow
        android:id="@+id/row1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- Add elements/columns in the first row-->
    </TableRow>

</TableLayout>
```



Frame Layout: Frame Layout is one of the simplest layout to organize view controls. They are designed to block an area on the screen. Frame Layout should be used to hold child view, because it can be difficult to display single views at a specific area on the screen without overlapping each other. We can add multiple children to a FrameLayout and control their position by assigning gravity to each child, using the android:layout_gravity attribute.

Dialogs

Alter Dialog: Alert Dialog in an android UI prompts a small window to make decision on mobile screen. Sometimes before making a decision it is required to give an alert to the user without moving to next activity. To solve this problem alert dialog came into practise. For example you have seen this type of alert when you try to exit the App and App ask you to confirm exiting. Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue. Android AlertDialog is composed of three regions: title, content area and action buttons. Android AlertDialog is the subclass of Dialog class. We need to import **android.app.AlertDialog class** in Android application. Following is the sample code:


```
//Creating dialog box
AlertDialog alert = builder.create();
//Setting the title manually
alert.setTitle("AlertDialogExample");
alert.show();
```

ProgressDialog: Android Progress Dialog is a UI which shows the progress of a task like you want user to wait until the previous lined up task is completed and for that purpose you can use progress dialog. The best example is you see when downloading or uploading any file. We need to import **android.app.ProgressDialog class** in Android application. Following is the sample code:

```
ProgressDialog progressDialog = new ProgressDialog(this);
```

Drawing and Working with Animation

Animation is the process of adding a motion effect to any view, image, or text. With the help of an animation, you can add motion or can change the shape of a specific view. Animation in Android is generally used to give your UI a rich look and feel. The animations are basically of three types as follows:

- **Property Animation:** Property Animation is one of the robust frameworks which allows animating almost everything. This is one of the powerful and flexible animations which was introduced in Android 3.0. Property animation can be used to add any animation in the CheckBox, RadioButtons, and widgets other than any view.
- **View Animation (Twined Animation):** View Animation can be used to add animation to a specific view to perform tweened animation on views. Tweened animation calculates animation information such as size, rotation, start point, and endpoint. These animations are slower and less flexible. An example of View animation can be used if we want to expand a specific layout in that place we can use View Animation. The example of View Animation can be seen in Expandable RecyclerView.
- **Drawable Animation (Frame by Frame Animation):** Drawable Animation is used if you want to animate one image over another. The simple way to understand is to animate drawable is to load the series of drawable one after another to create an animation. A simple example of drawable animation can be seen in many apps Splash screen on apps logo animation.

Important Questions from the Unit:

- Explain Layouts available in Android
- Write a note on Animation in Android
- What is Dialog? Explain types of Dialog of Android.
- Explain Button and ImageButton
- Explain TextView and EditText

Unit – 4

Database Connectivity Using SQLite and Content Provider

Using Android Data and Storage APIs
Android use following three methods to store data on mobile device:

Shared Preference: Shared Preference in Android are used to save data based on key-value pair. If we go deep into understanding of word: shared means to distribute data within and preference means something important or preferable, so SharedPreferences data is shared and preferred data. Shared Preference can be used to save primitive data type: string, long, int, float and Boolean.

A preference file is actually a xml file saved in internal memory of device. Every application have some data stored in memory in a directory data/data/application package name so whenever `getSharedPreferences(String name,int mode)` function get called it validates the file that if it exists or it doesn't then a new xml is created with passed name. There are two different ways to save data in Android through Shared Preferences – One is using Activity based preferences and other is creating custom preferences.

Activity Preferences: For activity preferences developer have to call function getPreferences (int mode) available in Activity class. Use only when one preference file is needed in Activity It doesn't require name as it will be the only preference file for this activity. Developer doesn't usually prefer using this even if they need only one preference file in Activity. They prefer using custom getSharedPreferences(String name,int mode).
To use activity preferences developer have to call function getPreferences (int mode) available in Activity class. The function getPreferences(int mode) call the other function used to create custom preferences i.e getSharedPreferences(String name,int mode). Just because Activity contains only one preference file so getPreferences(int mode) function simply pass the name of Activity class to create a preference file.

Custom Preferences: Developer needs to use getSharedPreferences(String name,int mode) for custom preferences. Used in cases when more than one preference file required in Activity. Name of the preference file is passed in first parameter. Custom Preferences can be created by calling function getSharedPreferences(String name,int mode), it can be called from anywhere in the application with reference of Context. Here name is any preferred name for example: User, Book etc. and mode is used to set kind of privacy for file.

Internal Storage: Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection. In this unit we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory. By default these files are private and are accessed by only your application and get deleted, when user delete your application.

Writing file: In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private , public e.t.c. Its syntax is given below –

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

The method openFileOutput() returns an instance of FileOutputStream. So you receive it in the object of FileInputStream. After that you can call write method to write data on the file. Its syntax is given below –

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Reading file: In order to read from the file you just created, call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below −

```
FileInputStream fin = openFileInput(file);
```

After that, you can call read method to read one character at a time from the file and then you can print it. Its syntax is given below −

```
int c;
String temp="";
while( (c = fin.read()) != -1){
   temp = temp + Character.toString((char)c);
}

//string temp contains all the data of the file.
fin.close();
```

External Sorage: Like internal storage, we are able to save or read data from the device external memory such as **sdcard**. The FileInputStream and FileOutputStream classes are used to read and write data into the file.

Managing data using SQLite

SQLite is an opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation. SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC, ODBC etc. SQLite is a relational database i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database. It is embedded in android bydefault. So, there is no need to perform any database setup or administration task. Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat. For displaying data on the spinner or listview, move to the next page. SQLiteOpenHelper class provides the functionality to use the SQLite database. The main package is **android.database.sqlite** that contains the classes to manage your own databases.

In order to create a database, you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. **SQLiteDatabase class** contains methods to be performed on sqlite database such as create, update, delete, select etc. It returns an instance of SQLite database which you have to receive in your own object. Its syntax is given below

```
SQLiteDatabase mydatabase = openOrCreateDatabase"your database name", MODE_PRIVATE, null);
```

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below:

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");
```

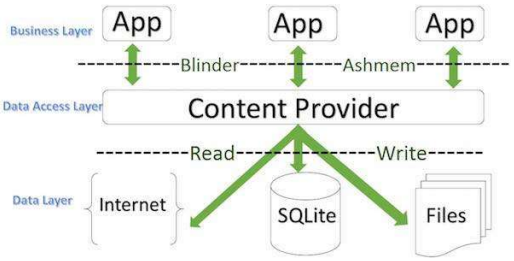
This will insert some values into our table in our database. We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

Database - Helper class: For managing all the operations related to the database, a helper class has been given and is called SQLiteOpenHelper. The **android.database.sqlite.SQLiteOpenHelper class** is used for database creation and version management. For performing any database operation, you have to provide the implementation of onCreate() and onUpgrade() methods of SQLiteOpenHelper class. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
   public DBHelper(){
      super(context,DATABASE_NAME,null,1);
   }
   public void onCreate(SQLiteDatabase db) {}
   public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Sharing Data Between Applications with Content Providers



A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network. sometimes it is required to share data across applications. This is where content providers become very useful. Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an SQlite database. A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

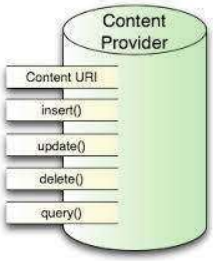
```
public class My Application extends ContentProvider {

}
```

Content URIs: To query a content provider, you specify the query string in the form of a URI which has following format –

```
<prefix>://<authority>/<data_type>/<id>
```

Here is the detail of various parts of the URI –

Part & Description
Prefix: This is always set to content://
Authority: This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
data_type: This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people*
Id: This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.



Create Content Provider: This involves number of simple steps to create your own content provider. First of all you need to create a Content Provider class that extends the ContentProviderbaseclass. Second, you need to define your content provider URI address which will be used to access the content. Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override onCreate() method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the onCreate() handler of each of its Content Providers is called on the main application thread. Next you will have to implement Content Provider queries to perform different database specific operations. Finally register your Content Provider in your activity file using <provider> tag. The figure shows the list of methods which you need to override in Content Provider class to have your Content Provider working.

- onCreate() This method is called when the provider is started.
- query() This method receives a request from a client. The result is returned as a Cursor object.
- insert()This method inserts a new record into the content provider.
- delete() This method deletes an existing record from the content provider.

- update() This method updates an existing record from the content provider.
- getType() This method returns the MIME type of the data at the given URI.

Important Questions form the Unit:

- Explain SharedPreference.
- Explain SQLiteOpenHelper Class.
- Explain SQLiteDatbase Class.
- Explain Content Provider.

Unit – 5

Location Based Services (LBS), Common Android API, Notifications, Services, Deployment of applications

Using Global Positioning Services (GPS)

Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology. Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc. We can also customize Google map according to our requirement.

Android allows us to integrate Google Maps in our application. For this Google provides us a library via Google Play Services for using maps. In order to use the Google Maps API, you must register your application on the Google Developer Console and enable the API.

Geocoding Locations

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

- **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
- **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
- **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
- **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
- **None:** This type displays an empty grid with no tiles loaded.

Syntax of different types of map

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Mapping Locations

Google map API provides several methods that help to customize Google map. These methods are as following:

Methods	Description
addCircle(CircleOptions options)	This method add circle to map.
addPolygon(PolygonOptions options)	This method add polygon to map.
addTileOverlay(TileOverlayOptions options)	This method add tile overlay to the map.
animateCamera(CameraUpdate update)	This method moves the map according to the update with an animation.
clear()	This method removes everything from the map.
getMyLocation()	This method returns the currently displayed user location.
moveCamera(CameraUpdate update)	This method reposition the camera according to the instructions defined in the update.
setTrafficEnabled(boolean enabled)	This method set the traffic layer on or off.

snapshot(GoogleMap.SnapshotReadyCallback callback)	This method takes a snapshot of the map.
stopAnimation()	This method stops the camera animation if there is any progress.

Many more with location based services

Geocoder Class: A class for handling geocoding and reverse geocoding. Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address. The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code. The Geocoder class requires a backend service that is not included in the core android framework. The Geocoder query methods will return an empty list if there no backend service in the platform. Use the isPresent() method to determine whether a Geocoder implementation exists. Constructors:

- **Geocoder(Context context):** Constructs a Geocoder localized for the default locale.
- **Geocoder(Context context, Locale locale):** Constructs a Geocoder localized for the given locale.

Methods:

- **void getFromLocationName(String locationName, int maxResults, Geocoder.GeocodeListener listener):** Provides an array of Addresses that attempt to describe the named location, which may be a place name such as "Dalvik, Iceland", an address such as "1600 Amphitheatre Parkway, Mountain View, CA", an airport code such as "SFO", and so forth.
- **List<Address> getFromLocationName(String locationName, int maxResults):** This method was deprecated in API level 33. Use getFromLocationName(java.lang.String, int, android.location.Geocoder.GeocodeListener) instead to avoid blocking a thread waiting for results.
- **static boolean isPresent():** Returns true if there is a geocoder implementation present that may return results.

LocationManager Class: This class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to be notified when the device enters the proximity of a given geographical location. Constatnts:

- **String KEY_LOCATIONS:** Key used for an extra holding a array of Locations when a location change is sent using a PendingIntent.
- **String KEY_LOCATION_CHANGED:** Key used for an extra holding a Location value when a location change is sent using a PendingIntent.
- **String MODE_CHANGED_ACTION:** Broadcast intent action when the device location enabled state changes.
- **String NETWORK_PROVIDER:** Standard name of the network location provider.

Android networking API

Android lets your application connect to the internet or any other local network and allows you to perform network operations. A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection. Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides ConnectivityManager class. You need to instantiate an object of this class by calling getSystemService() method. Its syntax is given below:

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

Once you instantiate the object of ConnectivityManager class, you can use getAllNetworkInfo method to get the information of all the networks. This method returns an array of NetworkInfo. So you have to receive it like this.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

The last thing you need to do is to check Connected State of the network. Its syntax is given below:

```
for (int i = 0; i<info.length; i++){
   if (info[i].getState() == NetworkInfo.State.CONNECTED){
      Toast.makeText(context, "Internet is connected
      Toast.LENGTH_SHORT).show();
   }
}
```

Android web API

A Web API is an online "application programming interface" that allows developers to interact with external services. These are the commands that the developer of the service has determined will be used to access certain features of their program. It is referred to as an interface because a good API should have commands that make it intuitive to interact with.

An example of this might be if we want to get information about a user from their social media account. That social media platform would likely have a web API for developers to use in order to request that data. Other commonly used APIs handle things like advertising (AdMob), machine learning (ML Kit), and cloud storage. It's easy to see how interacting with these types of services could extend the functionality of an app. In fact, the vast majority of successful apps on the Play Store will use at least one web API!

Most APIs work using either XML or JSON. These languages allow us to send and retrieve large amounts of useful information in the form of objects. XML is eXtensible Markup Language. If you are an Android developer, then you're probably already familiar with XML from building your layouts and saving variables. XML is easy to understand and generally places keys inside triangle brackets, followed by their values.

Android telephony API

The telephony API is used to among other things monitor phone information including the current states of the phone, connections, network etc. In this example, we want to utilize the telephone manager part of the Application Framework and use phone listeners (PhoneStateListener) to retrieve various phone states. This is a simple tutorial utilizing the telephony API and its associated listener which can be good before implementing other application functions related to a phone state like connecting the Call. This is a small example that can be used expanded with the various methods available, please see example:

Activity_Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="https://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Telephony Demo"
            android:textSize="22sp" />
      <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/textOut"
            android:text="Output"></TextView>
</LinearLayout>
```

Main_Activity.java

```
package com.TelephonyAPIs;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class TelephonyDemo extends Activity {
  TextView textOut;
  TelephonyManager telephonyManager;
  PhoneStateListener listener;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Get the UI
    textOut = (TextView) findViewById(R.id.textOut);

    // Get the telephony manager
    telephonyManager                    =                    (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);

    // Create a new PhoneStateListener
    listener = new PhoneStateListener() {
      @Override
      public void onCallStateChanged(int state, String incomingNumber) {
        String stateString = "N/A";
        switch (state) {
        case TelephonyManager.CALL_STATE_IDLE:
          stateString = "Idle";
          break;
        case TelephonyManager.CALL_STATE_OFFHOOK:
          stateString = "Off Hook";
          break;
        case TelephonyManager.CALL_STATE_RINGING:
          stateString = "Ringing";
          break;
        }
        textOut.append(String.format("\nonCallStateChanged:            %s",
stateString));
      }
    };

    // Register the listener wit the telephony manager
    telephonyManager.listen(listener,
PhoneStateListener.LISTEN_CALL_STATE);
  }
}
```

Notifying the user

A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Android Notification provides short, timely information about the action happened in the application, even it is not running. The notification displays the icon, title and some amount of the content text. The properties of Android notification are set using NotificationCompat.Builder object. Some of the notification properties are mention below:

- **setSmallIcon():** It sets the icon of notification.
- **setContentTitle():** It is used to set the title of notification.
- **setContentText():** It is used to set the text message.
- **setAutoCancel():** It sets the cancelable property of notification.
- **setPriority():** It sets the priority of notification.

You have simple way to create a notification. Follow the following steps in your application to create a notification:

Step 1 - Create Notification Builder: As a first step is to create a notification builder using NotificationCompat.Builder.build(). You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
```

Step 2 - Setting Notification Properties: Once you have Builder object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following:

A small icon, set by setSmallIcon()
A title, set by setContentTitle()
Detail text, set by setContentText()

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

You have plenty of optional properties which you can set for your notification. To learn more about them, see the reference documentation for NotificationCompat.Builder.

Step 3 - Attach Actions: This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events or do further work. The action is defined by a PendingIntent containing an Intent that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of NotificationCompat.Builder. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling setContentIntent(). A PendingIntent object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running. We take help of stack builder object which will contain an artificial back stack for the started Activity. This ensures that navigating backward from the Activity leads out of your application to the Home screen.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

Step 4 - Issue the notification: Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call NotificationCompat.Builder.build()

method on builder object before notifying it. This method combines all of the options that have been set and return a new Notification object.

```
NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

The NotificationCompat.Builder Class: The NotificationCompat.Builder class allows easier control over all the flags, as well as help constructing the typical notification layouts. Following are few important and most frequently used methods available as a part of NotificationCompat.Builder class.

Sr.	Constants & Description
1	**Notification build()** Combine all of the options that have been set and return a new Notification object.
2	**NotificationCompat.Builder setAutoCancel (boolean autoCancel)** Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel.
3	**NotificationCompat.Builder setContent (RemoteViews views)** Supply a custom RemoteViews to use instead of the standard one.
4	**NotificationCompat.Builder setContentInfo (CharSequence info)** Set the large text at the right-hand side of the notification.
5	**NotificationCompat.Builder setContentIntent (PendingIntent intent)** Supply a PendingIntent to send when the notification is clicked.
6	**NotificationCompat.Builder setContentText (CharSequence text)** Set the text (second row) of the notification, in a standard notification.
7	**NotificationCompat.Builder setContentTitle (CharSequence title)** Set the text (first row) of the notification, in a standard notification.
8	**NotificationCompat.Builder setDefaults (int defaults)** Set the default notification options that will be used.
9	**NotificationCompat.Builder setLargeIcon (Bitmap icon)** Set the large icon that is shown in the ticker and notification.
10	**NotificationCompat.Builder setNumber (int number)** Set the large number at the right-hand side of the notification.
11	**NotificationCompat.Builder setOngoing (boolean ongoing)** Set whether this is an ongoing notification.
12	**NotificationCompat.Builder setSmallIcon (int icon)** Set the small icon to use in the notification layouts.
13	**NotificationCompat.Builder setStyle (NotificationCompat.Style style)** Add a rich notification style to be applied at build time.
14	**NotificationCompat.Builder setTicker (CharSequence tickerText)** Set the text that is displayed in the status bar when the notification first arrives.

15	**NotificationCompat.Builder setVibrate (long[] pattern)** Set the vibration pattern to use.
16	**NotificationCompat.Builder setWhen (long when)** Set the time that the event occurred. Notifications in the panel are sorted by this time.

You can update the information in your status bar notification as events continue to occur in your application. For example, when a new SMS text message arrives before previous messages have been read, the Messaging application updates the existing notification to display the total number of new messages received. This practice of updating an existing Notification is much better than adding new Notifications to the NotificationManager because it avoids clutter in the Notifications window. Because each notification is uniquely identified by the NotificationManager with an integer ID, you can revise the notification by calling setLatestEventInfo() with new values, change some field values of the Notification, and then call notify() again.

You can revise each property with the object member fields (except for the Context and the expanded message title and text). You should always revise the text message when you update the notification by calling setLatestEventInfo() with new values for contentTitle and contentText. Then call notify() to update the notification. (Of course, if you've created a custom expanded view, then updating these title and text values has no effect.)

Adding a sound: You can alert the user with the default notification sound (which is defined by the user) or with a sound specified by your application. To use the user's default sound, add "DEFAULT_SOUND" to the defaults field:

```
notification.defaults |= Notification.DEFAULT_SOUND;
```

To use a different sound with your notifications, pass a Uri reference to the sound field. The following example uses a known audio file saved to the device SD card:

```
notification.sound =
Uri.parse("file:///sdcard/notification/ringer.mp3");
```

In the next example, the audio file is chosen from the internal MediaStore's ContentProvider:

```
notification.sound = Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

In this case, the exact ID of the media file ("6") is known and appended to the content Uri. If you don't know the exact ID, you must query all the media available in the MediaStore with a ContentResolver. See the Content Providers documentation for more information on using a ContentResolver. If you want the sound to continuously repeat until the user responds to the notification or the notification is cancelled, add "FLAG_INSISTENT" to the flags field. Note: If the defaults field includes "DEFAULT_SOUND", then the default sound overrides any sound defined by the sound field.

Adding vibration: You can alert the user with the the default vibration pattern or with a vibration pattern defined by your application. To use the default pattern, add "DEFAULT_VIBRATE" to the defaults field:

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
```

To define your own vibration pattern, pass an array of long values to the vibrate field:

```
long[] vibrate = {0,100,200,300};
notification.vibrate = vibrate;
```

The long array defines the alternating pattern for the length of vibration off and on (in milliseconds). The first value is how long to wait (off) before beginning, the second value is the length of the first vibration, the third is the next length off, and so on. The pattern can be as long as you like, but it can't be set to repeat. Note: If the defaults field includes "DEFAULT_VIBRATE", then the default vibration overrides any vibration defined by the vibrate field.

Adding flashing lights: To alert the user by flashing LED lights, you can implement the default light pattern (if available), or define your own color and pattern for the lights. To use the default light setting, add "DEFAULT_LIGHTS" to the defaults field:

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
```

To define your own color and pattern, define a value for the ledARGB field (for the color), the ledOffMS field (length of time, in milliseconds, to keep the light off), the ledOnMS (length of time, in milliseconds, to keep the light on), and also add "FLAG_SHOW_LIGHTS" to the flags field:

```
notification.ledARGB = 0xff00ff00;
notification.ledOnMS = 300;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

In this example, the green light repeatedly flashes on for 300 milliseconds and turns off for one second. Not every color in the spectrum is supported by the device LEDs, and not every device supports the same colors, so the hardware estimates to the best of its ability. Green is the most common notification color.

Application development using JSON in MySQL

JSON stands for JavaScript Object Notation.It is an independent data exchange format and is the best alternative for XML. This chapter explains how to parse the JSON file and extract necessary information from it. Android provides four different classes to manipulate JSON data. These classes are JSONArray, JSONObject, JSONStringer and JSONTokenizer. The first step is to identify the fields in the JSON data in which you are interested in. For example. In the JSON given below we interested in getting temperature only.

```
{
   "sys":
   {
      "country":"GB",
      "sunrise":1381107633,
      "sunset":1381149604
   },
   "weather":[
      {
         "id":711,
         "main":"Smoke",
         "description":"smoke",
         "icon":"50n"
      }
   ],
  "main":
   {
      "temp":304.15,
      "pressure":1009,
   }
}
```

JSON – Elements: An JSON file consist of many components. Here is the table defining the components of an JSON file and their description:

- **Array([):** In a JSON file , square bracket ([) represents a JSON array
- **Objects({):** In a JSON file, curly bracket ({) represents a JSON object
- **Key:** A JSON object contains a key that is just a string. Pairs of key/value make up a JSON object
- **Value:** Each key has a value that could be string, integer or double e.t.c

JSON – Parsing: For parsing a JSON object, we will create an object of class JSONObject and specify a string containing JSON data to it. Its syntax is −

```
String in;
```

```
JSONObject reader = new JSONObject(in);
```

The last step is to parse the JSON. A JSON file consist of different object with different key/value pair e.t.c. So JSONObject has a separate function for parsing each of the component of JSON file. Its syntax is given below:

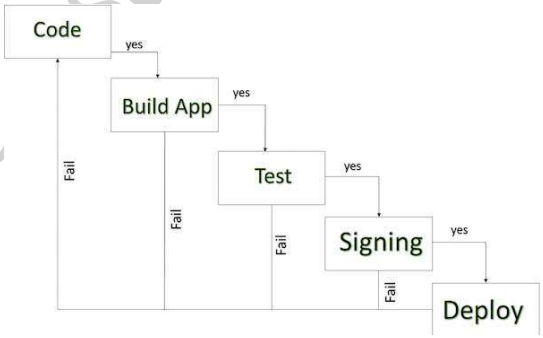
```
JSONObject sys  = reader.getJSONObject("sys");
country = sys.getString("country");

JSONObject main  = reader.getJSONObject("main");
temperature = main.getString("temp");
```

The method getJSONObject returns the JSON object. The method getString returns the string value of the specified key. Apart from these methods, there are other methods provided by this class for better parsing JSON files. These methods are listed below:

- **get(String name):** This method just Returns the value but in the form of Object type
- **getBoolean(String name):** This method returns the boolean value specified by the key
- **getDouble(String name):** This method returns the double value specified by the key
- **getInt(String name):** This method returns the integer value specified by the key
- **getLong(String name):** This method returns the long value specified by the key
- **length():** This method returns the number of name/value mappings in this object.
- **names():** This method returns an array containing the string names in this object.

Publish android application

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



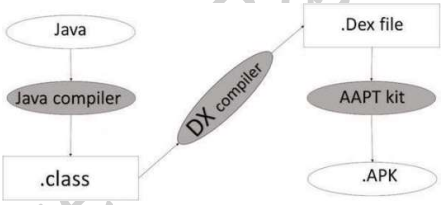
Android development life cycle: Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application:

- **Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
- **Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
- **Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly, you must take care of setting up time zone, localization or any other specific requirement as per the targeted region.
- **Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.
- **SDK and Screen Compatibility,** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.

- **Application Pricing** Deciding whether your app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.
- **Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
- **Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: Preparing for Release.
- **Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

Export Android Application Process: Apk development process, Before exporting the apps, you must some of tools



- Dx tools(Dalvik executable tools): It going to convert .class file to .dex file. it has useful for memory optimization and reduce the boot-up speed time
- AAPT(Android assistance packaging tool):it has useful to convert .Dex file to.Apk
- APK(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace. To export an application, just open that application project in Android studio and select Build → Generate Signed APK from your Android studio. Next select, Generate Signed APK option as shown in the above screen shot and then click it so that you get following screen where you will choose Create new keystore to store your application.

Enter your key store path, key store password, key alias and key password to protect your application and click on Next button once again. It will display following screen to let you create an application –

Once you filled up all the information, like app destination, build type and flavours click finish button While creating an application it will show as below

Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

Google Play Registration: The most important step is to register with Google Play using Google Play Marketplace. You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace.