UNIT-5

Events Database and Connectivity

5.1 Handling UI Events

Events are the actions performed by the user in order to interact with the application, for e.g. pressing a button or touching the screen. The events are managed by the android framework in the FIFO manner i.e. First In – First Out. Handling such actions or events by performing the desired task is called Event Handling.Android provides a huge set of 2D-drawing APIs that allow you to create graphics.

Overview of the input event management

- **Event Listeners:** It is an interface in the View class. It contains a single callback method. Once the view to which the listener is associated is triggered due to user interaction, the callback methods are called.
- **Event Handlers:** It is responsible for dealing with the event that the event listeners registered for and performing the desired action for that respective event.
- **Event Listeners Registration:** Event Registration is the process in which an Event Handler gets associated with an Event Listener so that this handler is called when the respective Event Listener fires the event.
- **Touch Mode:** When using an app with physical keys it becomes necessary to give focus to buttons on which the user wants to perform the action but if the device is touch-enabled and the user interacts with the interface by touching it, then it is no longer necessary to highlight items or give focus to particular View. In such cases, the device enters touch mode and in such scenarios, only those views for which the isFocusableInTouchMode() is true will be focusable, e.g. plain text widget.

For e.g. if a button is pressed then this action or event gets registered by the event listener and then the task to be performed by that button press is handled by the event handler, it can be anything like changing the color of the text on a button press or changing the text itself, etc.

Event Listeners and their respective event handlers

1. **OnClickListener():** This method is called when the user clicks, touches, or focuses on any view (widget) like Button, ImageButton, Image, etc. Event handler used for this is onClick().
2. **OnLongClickListener():** This method is called when the user presses and holds a particular widget for one or more seconds. Event handler used for this is onLongClick().
3. **OnMenuItemClickListener():** This method is called when the user selects a menu item. Event handler used for this is onMenuItemClick().
4. **OnMenuItemClickListener():** This method is called when the user selects a menu item. Event handler used for this is onMenuItemClick().

There are various other event listeners available which can be used for different requirements and can be found in the official documentation.

Example:

Step1: Create a New Project in Android Studio

Step2: Working with the activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <ImageView
      android:id="@+id/image_view_1"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:ignore="ContentDescription"
      android:background="@color/black"/>

</RelativeLayout>
```

Step3: Working with the MainActivity.kt file

```
package com.latihan.darmanto.radiobutton
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.RadioButton
import android.widget.TextView
class MainActivity : AppCompatActivity() {
          override fun onCreate(savedInstanceState: Bundle?) {
          super.onCreate(savedInstanceState)
          setContentView(R.layout.activity_main)
 }
 fun onRadioButtonClicked(view: View) {
          var textView: TextView = findViewById(R.id.textView)
          if (view is RadioButton) {
          // Is the button now checked?
          val checked = view.isChecked
          // Check which radio button was clicked
          when (view.getId()) {
          R.id.radio_pirates ->
          if (checked) {
          // Pirates are the best
         textView.setText("Pirates")
          }
          R.id.radio_ninjas ->
          if (checked) {
          // Ninjas rule
         textView.setText("Ninjas")
              }
        }
    }
 }
}
```

5.2 Building data with adapter view class

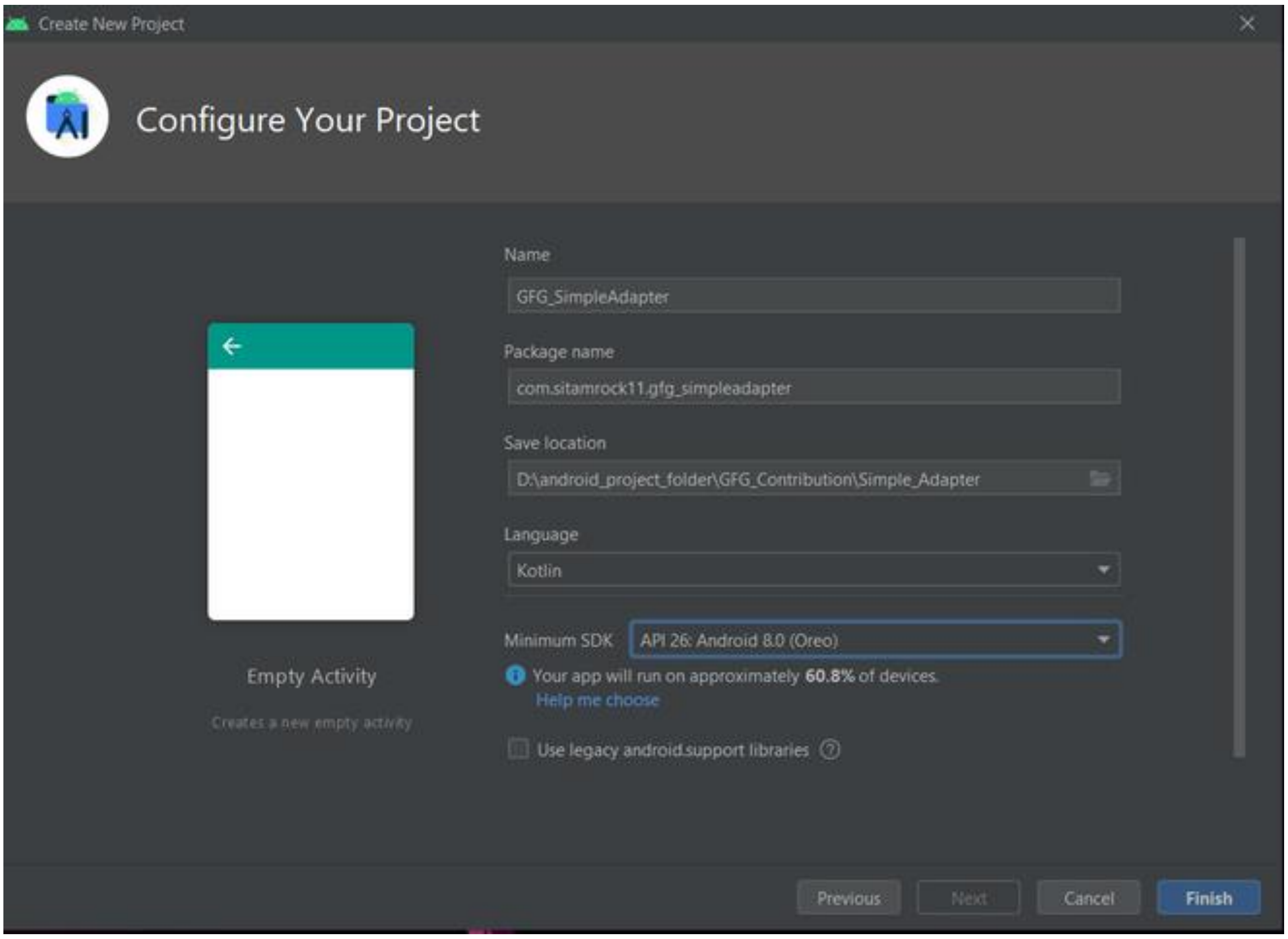
In Android, whenever we want to bind some data which we get from any data source (e.g. ArrayList, HashMap, SQLite, etc.) with a UI component(e.g. ListView, GridView, etc.) then **Adapter** comes into the picture. Basically **Adapter** acts as a bridge between the UI component and data sources. Here **Simple Adapter** is one type of **Adapter**. It is basically an easy adapter to map static data to views defined in our XML file(UI component) and is used for customization of List or Grid items. Here we use an ArrayList of Map (e.g. hashmap, mutable map, etc.) for data-backing. Each entry in an ArrayList is corresponding to one row of a list. The Map contains the data for each row. Now to display the row we need a view for which we used to specify a custom list item file (an XML file).

Example:

A sample image is given below to get an idea about what we are going to do in this article. In this project, we are going to make this application which has a list of some fruits and in each row of the list has a fruit image and name. Note that we are going to implement this same project in both Kotlin and Java languages. Now you choose your preferred language.

Step1: Create New Project

1. Open Android Studio
2. Go to File => New => New Project.
3. Then, select Empty Activity and click on next



Configure Your Project

Name: GFG_SimpleAdapter

Package name: com.sitamrock11.gfg_simpleadapter

Save location: D:\android_project_folder\GFG_Contribution\Simple_Adapter

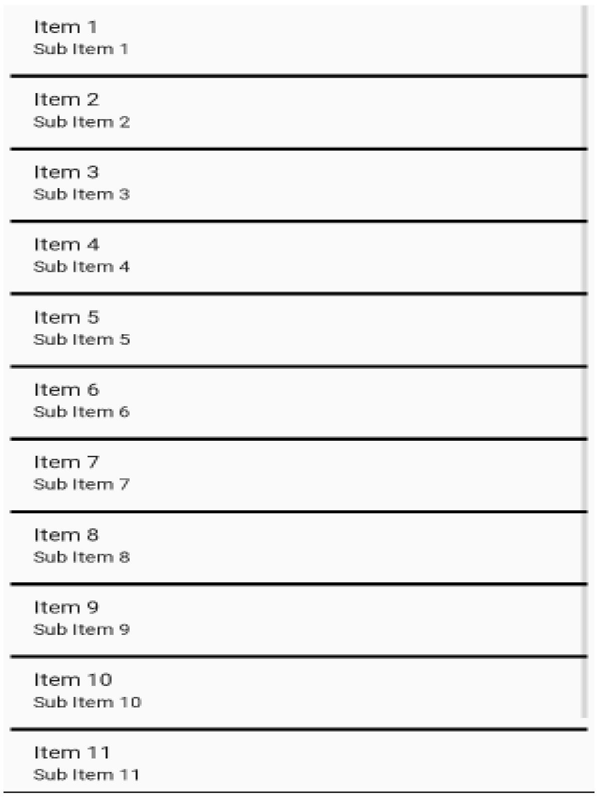
Language: Kotlin

Minimum SDK: API 26: Android 8.0 (Oreo)

Your app will run on approximately **60.8%** of devices.

Step2: Modify activity_main.xml file

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

<RelativeLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">


    <!--Creating a ListView-->
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:divider="#000000"
        android:dividerHeight="3dp"
        android:padding="5dp"  />

</RelativeLayout>
```

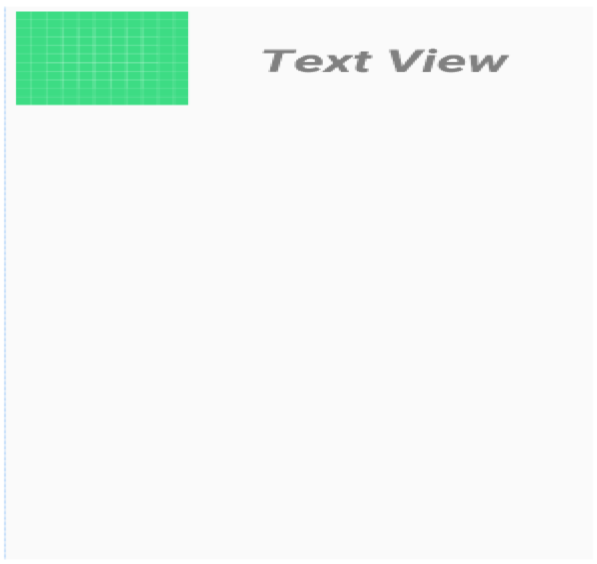

Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Item 8
Sub Item 8
Item 9
Sub Item 9
Item 10
Sub Item 10
Item 11
Sub Item 11

Create another XML file (named list_row_items) and create UI for each row of the ListView:

Below is the code for the list_row_items.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!--Creating a ImageView-->
    <ImageView
```

```
        android:id="@+id/imageView"
        android:layout_width="120dp"
        android:layout_height="120dp"
        android:layout_margin="10dp"
        android:scaleType="fitCenter"
        android:src="@drawable/ic_launcher_background"  />

    <!--Creating a TextView-->
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:layout_marginRight="20dp"
        android:layout_toRightOf="@+id/imageView"
        android:gravity="center"
        android:padding="5dp"
        android:text="Text View"
        android:textColor="#808080"
        android:textSize="40sp"
        android:textStyle="bold|italic" />

</RelativeLayout>
```



Step3: Create MainActivity.kt file

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ListView
import android.widget.SimpleAdapter
import java.util.ArrayList
import java.util.HashMap

class MainActivity : AppCompatActivity() {

    private lateinit var listView:ListView
    // creating  a String type array
      // (fruitNames) which contains
    // names of different fruits' images
    private val fruitNames=arrayOf("Banana","Grape","Guava","Mango","Orange","Watermelon")

    // creating an Integer type array (fruitImageIds) which
    // contains IDs of different fruits' images
    private val fruitImageIds=arrayOf(R.drawable.banana,
                            R.drawable.grape,
                            R.drawable.guava,
                            R.drawable.mango,
                            R.drawable.orange,
                            R.drawable.watermelon)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // ViewBinding the ListView  of activity_main.xml file
        // with this kotlin code in MainActivity.kt
        listView=findViewById(R.id.listView)
        // creating an ArrayList of HashMap.The kEY of the HashMap is
        // a String and VALUE is of any datatype(Any)
        val list=ArrayList<HashMap<String,Any>>()

        // By a for loop, entering different types of data in HashMap,
        // and adding this map including it's datas into the ArrayList
        // as list item and this list is the second parameter of the SimpleAdapter
        for(i in fruitNames.indices){
            val map=HashMap<String,Any>()
            // Data entry in HashMap
            map["fruitName"] = fruitNames[i]
            map["fruitImage"]=fruitImageIds[i]
            // adding the HashMap to the ArrayList
            list.add(map)
        }

        // creating A string type array(from) which contains
        // column names for each View in each row of the list
        // and this array(form) is the fourth parameter of the SimpleAdapter
        val from=arrayOf("fruitName","fruitImage")
        // creating an integer type array(to) which contains
        id of each View in each row of the list
        and this array(form) is the fifth parameter of the SimpleAdapter*/
        val to= intArrayOf(R.id.textView,R.id.imageView)
        // creating an Object of SimpleAdapter
          // class and passing
        // all the required parameters
        val simpleAdapter=SimpleAdapter(this,list,R.layout.list_row_items,from,to)
        // now setting the simpleAdapter
          // to the ListView
        listView.adapter = simpleAdapter
    }
}
```

5.3 Introducing the data storage option

We employ some form of storage in Android to retain the data permanently (until destroyed) for future reference. Android Storage System is the name given to these storage systems. Internal storage, external storage, shared preferences, database, and shared storage are some of the storage options offered by Android. However, many users are unsure when to use which storage. So, in this blog, we'll discover when to use which storage unit. Let's begin with the internal storage system. We create several variables for storing various data that are used in an Android application when developing it. For example, we can utilize a variable to save data from a remote database and then use that variable throughout the application. These variables, however, are in-app storage, which means they will be visible to you while the app is operating. When the application is ended, all of the data in the variable is wiped, and you are left with nothing. Those variables will be created again when you start the application, and new values can be saved in those variables.

1. **Storage on the Inside:** When you install an app on your phone, the Android operating system will give you some form of secret internal storage where the app can store its private data. No other application has access to this information. When you uninstall an application, all of the data associated with it is also removed.To save a file to the internal storage, you must first obtain it from the internal directory. You can do this by calling the getFilesDir() or getCacheDir() methods. The getFilesDir() method returns the absolute path to the directory where files are created on the filesystem. getCacheDir() returns the absolute path to the filesystem's application-specific cache directory.

   When Should Internal Storage Be Used?

   The internal storage can be used when you need some confidential data for your application. Another thing to keep in mind is that if your app is storing data that may be utilized by other apps, you should avoid using internal storage since when you remove the app, all of your data will be gone, and other apps will never have access to that data. For instance, if your app is downloading a pdf or storing an image or video that might be used by other apps, you shouldn't use internal storage.

2. **External Hard Drives:** Most Android devices have relatively low internal storage. As a result, we keep our data on an external storage device. These storage units are accessible to everyone, which means they can be accessed by all of your device's applications. You can also access the storage by connecting your mobile device to a computer. You must obtain the READ EXTERNAL STORAGE permission from the user in order to gain access to the external storage. As a result, any application with this permission has access to your app's data.

   When is it appropriate to use external storage?

   You can use external storage if the data that your application stores can be used by other applications. Additionally, if the file stored by your application is huge, such as a video, you can save it to external storage. You can use external storage to keep the data even after uninstalling the application.

3. **Using the Shared Preferences:** You can use the shared preferences if you only have a little amount of data to keep and don't want to use the internal storage. Shared Preferences are used to store data in a key-value format, which means you'll have one key and the associated data or value will be stored depending on that key. The data saved in the shared preferences will remain with the application until you delete it from your phone. All shared preferences will be deleted from the device if you uninstall the application.

   When Should Shared Preferences Be Used?

You can utilize the shared preference in your application when the data you want to store is relatively little. It's not a good idea to save more than 100 kilobytes of data in shared preferences. In addition, if you wish to keep tiny and private data, you can use Android's shared preferences.

4. **Using Android Database:** Databases are collections of data that are organized and saved for future use. Using a Database Management System, you can store any type of data in your database. All you have to do is establish the database and use one query to perform all of the operati ons, such as insertion, deletion, and searching. The query will be passed to the database, which will return the desired output. In Android, an SQLite database is an example of a database.

   When Should you utilize a database?

   A database is useful when you need to keep track of structured data. A database can hold any type of information. So, if your data is large and you want to retrieve it quickly, you can use a database and store it in a structured style.

5.4 Internal and External Storage

External Storage: Android External Storage is the memory space in which we perform read and write operation. Files in the external storage are stored in /sdcard or /storage folder etc. The files which are saved in the external storage is readable and can be modified by the user.

Before accessing the file in external storage in our application, we should check the availability of the external storage in our device.

Example: In this example, we will write the data to file inside the external storage and read the same file content from same external storage.

Step1: Create activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="example.javatpoint.com.kotlinexternalstoragereadwrite.MainActivity">
   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignLeft="@+id/textView2"
      android:layout_alignParentTop="true"
      android:layout_alignStart="@+id/textView2"
      android:layout_marginTop="68dp"
```

```
    android:text="File Name"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.027"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.065" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextData"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="36dp"
    android:layout_marginLeft="50dp"
    android:layout_marginStart="50dp"
    android:text="File Data"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.027"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    app:layout_constraintVertical_bias="0.167" />

<EditText
    android:id="@+id/editTextFile"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editTextData"
    android:layout_alignStart="@+id/editTextData"
    android:layout_alignTop="@+id/textView"
    android:ems="10"
    android:inputType="none" />

<EditText
    android:id="@+id/editTextData"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/editTextFile"
    android:layout_marginEnd="37dp"
    android:layout_marginRight="37dp"
    android:layout_marginTop="30dp"
    android:ems="10"
    android:inputType="none"
    android:lines="5" />

<Button
    android:id="@+id/button_save"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="68dp"
    android:layout_toLeftOf="@+id/editTextData"
    android:layout_toStartOf="@+id/editTextData"
    android:text="Save" />

<Button
    android:id="@+id/button_view"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/button_save"
        android:layout_alignEnd="@+id/editTextData"
        android:layout_alignRight="@+id/editTextData"
        android:layout_marginEnd="43dp"
        android:layout_marginRight="43dp"
        android:text="View" />

</RelativeLayout>
```

Step2: Create MainActivity.kt

```
package example.javatpoint.com.kotlinexternalstoragereadwrite

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import android.os.Environment
import java.io.*

class MainActivity : AppCompatActivity() {
    private val filepath = "MyFileStorage"
    internal var myExternalFile: File?=null
    private val isExternalStorageReadOnly: Boolean get() {
        val extStorageState = Environment.getExternalStorageState()
        return if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState))  {
                true
        } else {
                false
        }
```

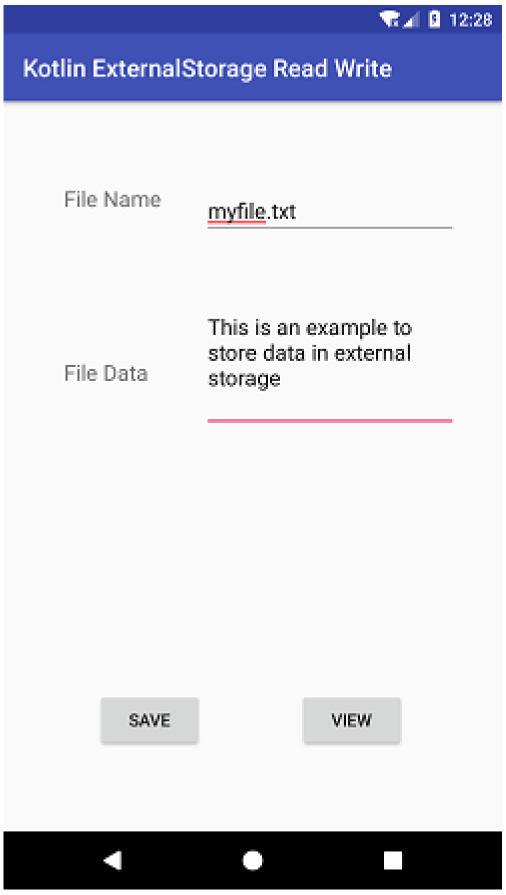
```
}
private val isExternalStorageAvailable: Boolean get() {
   val extStorageState = Environment.getExternalStorageState()
   return if (Environment.MEDIA_MOUNTED.equals(extStorageState))  {
      true
   } else{
      false
   }
}

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)
   val fileName = findViewById(R.id.editTextFile)  as EditText
   val fileData = findViewById(R.id.editTextData)  as EditText
   val saveButton = findViewById<Button>(R.id.button_save)  as Button
   val viewButton = findViewById(R.id.button_view)  as Button

   saveButton.setOnClickListener(View.OnClickListener  {
     myExternalFile = File(getExternalFilesDir(filepath), fileName.text.toString())
      try {
          val fileOutPutStream = FileOutputStream(myExternalFile)
          fileOutPutStream.write(fileData.text.toString().toByteArray())
          fileOutPutStream.close()
      } catch (e: IOException) {
          e.printStackTrace()
      }
      Toast.makeText(applicationContext,"data save",Toast.LENGTH_SHORT).show()
   })
   viewButton.setOnClickListener(View.OnClickListener  {
     myExternalFile = File(getExternalFilesDir(filepath), fileName.text.toString())
```

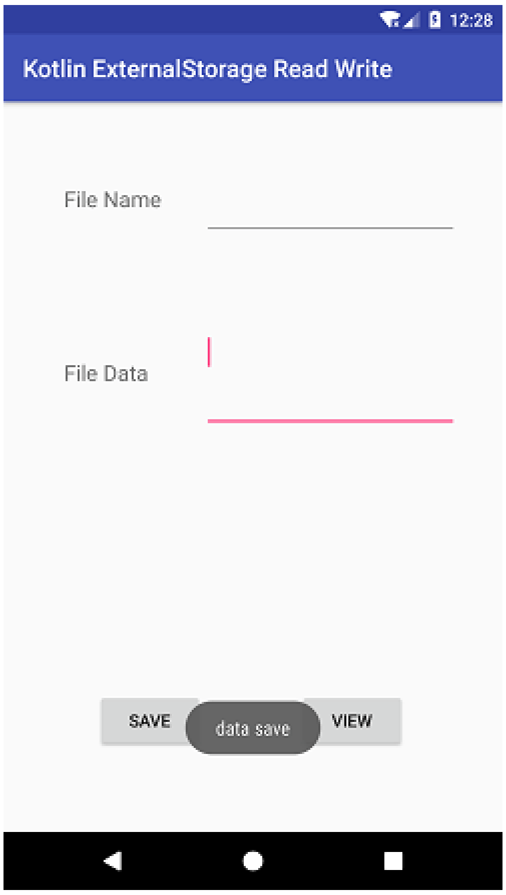
```
        val filename = fileName.text.toString()
        myExternalFile = File(getExternalFilesDir(filepath),filename)
        if(filename.toString()!=null && filename.toString().trim()!=""){
            var fileInputStream =FileInputStream(myExternalFile)
            var inputStreamReader: InputStreamReader = InputStreamReader(fileInputStream)
            val bufferedReader: BufferedReader = BufferedReader(inputStreamReader)
            val stringBuilder: StringBuilder = StringBuilder()
            var text: String? = null
            while ({ text = bufferedReader.readLine(); text }() != null) {
                stringBuilder.append(text)
            }
            fileInputStream.close()
            //Displaying data on EditText
            Toast.makeText(applicationContext,stringBuilder.toString(),Toast.LENGTH_SHORT).show()
        }
      })

      if (!isExternalStorageAvailable || isExternalStorageReadOnly) {
          saveButton.isEnabled = false
      }
   }
}
```

Step3: Create AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="example.javatpoint.com.kotlinexternalstoragereadwrite">
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        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>
```

OutPut:


12:28
Kotlin ExternalStorage Read Write
File Name
myfile.txt
File Data
This is an example to store data in external storage
SAVE
VIEW



5.5 Using SQLite Database

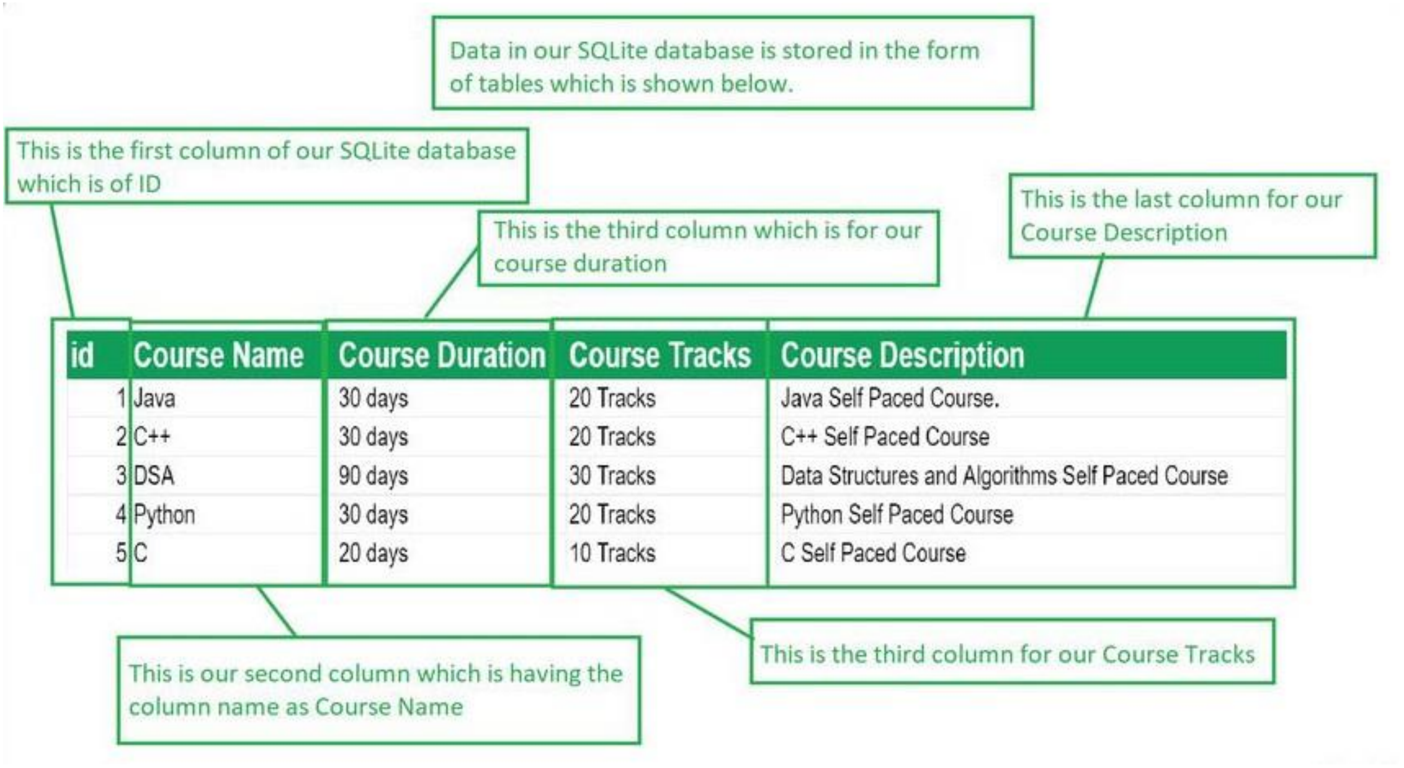
SQLite is another data storage available in Android where we can store data in the user's device and can use it any time when required. In this article, we will take a look at creating an SQLite database in the Android app and adding data to that database in the Android app. This is a series of 4 articles in which we are going to perform the basic CRUD (Create, Read, Update, and Delete) operation with SQLite Database in Android.

What is SQLite Database?

SQLite Database is an open-source database provided in Android which is used to store data inside the user's device in the form of a Text file. We can perform so many operations on this data such as adding new data, updating, reading, and deleting this data. SQLite is an offline database that is locally stored in the user's device and we do not have to create any connection to connect to this database.

How Data is Being Stored in the SQLite Database?

Data is stored in the SQLite database in the form of tables. When we stored this data in our SQLite database it is arranged in the form of tables that are similar to that of an excel sheet. Below is the representation of our SQLite database which we are storing in our SQLite database.


Data in our SQLite database is stored in the form of tables which is shown below.
This is the first column of our SQLite database which is of ID
This is the third column which is for our course duration
This is the last column for our Course Description
This is our second column which is having the column name as Course Name
This is the third column for our Course Tracks

id	Course Name	Course Duration	Course Tracks	Course Description
1	Java	30 days	20 Tracks	Java Self Paced Course.
2	C++	30 days	20 Tracks	C++ Self Paced Course
3	DSA	90 days	30 Tracks	Data Structures and Algorithms Self Paced Course
4	Python	30 days	20 Tracks	Python Self Paced Course
5	C	20 days	10 Tracks	C Self Paced Course

Step1: Giving permission to access the storage in the AndroidManifest.xml file

Navigate to app > AndroidManifest.xml and add the below code to it.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"  />
```

Step2: Working with the activity_main.xml file

Navigate to app > res > layout > activity_main.xml. Add the below code to your file. Below is the code for activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!-- Edit text to enter name -->
    <EditText
        android:id="@+id/enterName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Name"
        android:textSize="22sp"
        android:layout_margin="20sp"/>

    <!-- Edit text to enter age -->
    <EditText
        android:id="@+id/enterAge"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20sp"
        android:textSize="22sp"
        android:hint="Enter Age"/>

    <!-- Button to add Name -->
    <Button
        android:id="@+id/addName"
        android:layout_width="150sp"
        android:layout_gravity="center"
        android:background="@color/colorPrimary"
        android:text="Add Name"
        android:textColor="#ffffff"
        android:textSize="20sp"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"/>

    <!-- Button to print Name -->
    <Button
        android:id="@+id/printName"
        android:layout_width="150sp"
        android:layout_gravity="center"
        android:background="@color/colorPrimary"
```

```
        android:text="Print Name"
        android:textColor="#ffffff"
        android:textSize="20sp"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <!-- Text view to get all name -->
        <TextView
            android:id="@+id/Name"
            android:textAlignment="center"
            android:layout_weight="1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="20sp"
            android:text="Name\n\n"
            android:textSize="22sp"
            android:padding="10sp"
            android:textColor="#000000"/>

        <!-- Text view to get all ages -->
        <TextView
            android:layout_weight="1"
            android:id="@+id/Age"
            android:textAlignment="center"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="20sp"
            android:text="Age\n\n"
        android:textSize="22sp"
        android:padding="10sp"
        android:textColor="#000000"/>

    </LinearLayout>

</LinearLayout>
```

Step4: Creating a new class for SQLite operations

```
package com.release.database

import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
    SQLiteOpenHelper(context, DATABASE_NAME, factory, DATABASE_VERSION) {

    // below is the method for creating a database by a sqlite query
    override fun onCreate(db: SQLiteDatabase) {
        // below is a sqlite query, where column names
        // along with their data types is given
        val query = ("CREATE TABLE " + TABLE_NAME + " ("
                + ID_COL + " INTEGER PRIMARY KEY, " +
                NAME_COl + " TEXT," +
                AGE_COL + " TEXT" + ")")
        // we are calling sqlite
        // method for executing our query
        db.execSQL(query)
    }
```

```
override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
    // this method is to check if table already exists
    db.execSQL("DROP  TABLE IF EXISTS " + TABLE_NAME)
    onCreate(db)
}
// This method is for adding data in our database
fun addName(name : String, age : String ){
    // below we are creating
    // a content values variable
    val values = ContentValues()
    // we are inserting our values
    // in the form of key-value pair
    values.put(NAME_COl, name)
    values.put(AGE_COL, age)
    // here we are creating a
    // writable variable of
    // our database as we want to
    // insert value in our database
    val db = this.writableDatabase
    // all values are inserted into database
    db.insert(TABLE_NAME, null, values)
    // at last we are
    // closing our database
    db.close()
}
// below method is to get
// all data from our database
fun getName(): Cursor? {
    // here we are creating a readable
    // variable of our database
    // as we want to read value from it
    val db = this.readableDatabase
```

```
        // below code returns a cursor to
        // read data from the database
        return db.rawQuery("SELECT  * FROM " + TABLE_NAME,  null)
    }
    companion object{
        // here we have defined variables for our database
        // below is variable for database name
        private val DATABASE_NAME  = "Practice_Database"
        // below is the variable for database version
        private val DATABASE_VERSION  = 1
        // below is the variable for table name
        val TABLE_NAME  = "Main_table"
        // below is the variable for id column
        val ID_COL  = "id"
        // below is the variable for name column
        val NAME_COl  = "name"
        // below is the variable for age column
        val AGE_COL  = "age"
    }
}
```

Step5: Working with MainActivity.kt file

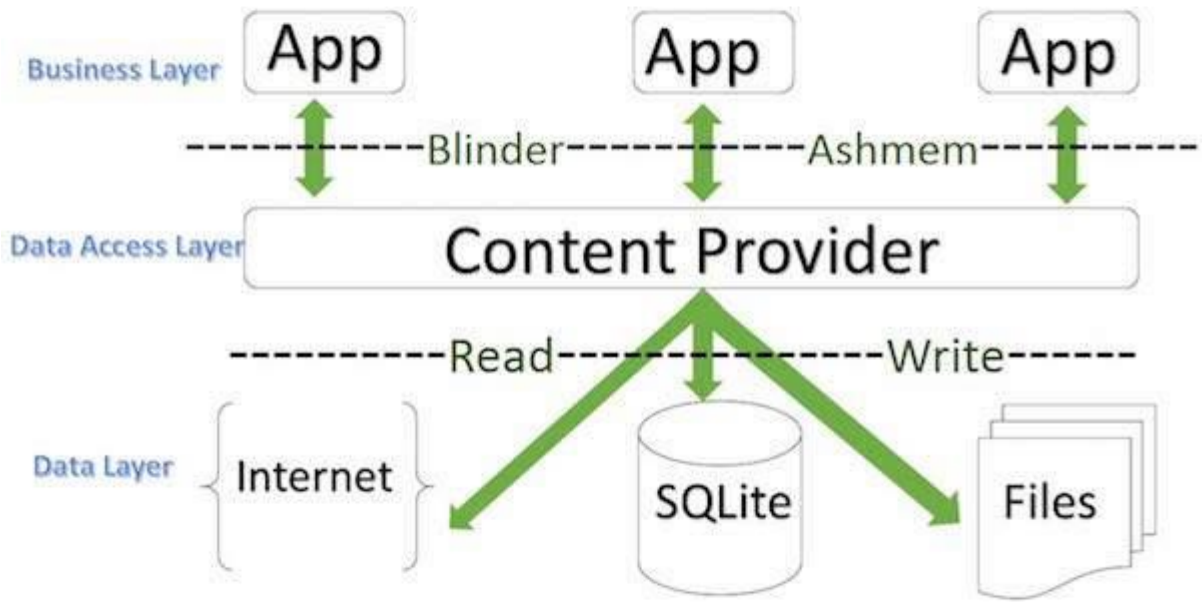
```
package com.release.main

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
setContentView(R.layout.activity_main)
// below code is to add on click
// listener to our add name button
addName.setOnClickListener{
    // below we have created
    // a new DBHelper class,
    // and passed context to it
    val db = DBHelper(this, null)
    // creating variables for values
    // in name and age edit texts
    val name = enterName.text.toString()
    val age = enterAge.text.toString()
    // calling method to add
    // name to our database
    db.addName(name, age)
    // Toast to message on the screen
    Toast.makeText(this, name + " added to database", Toast.LENGTH_LONG).show()
    // at last, clearing edit texts
    enterName.text.clear()
    enterAge.text.clear()
}
// below code is to add on  click
// listener to our print name button
printName.setOnClickListener{
    // creating a DBHelper class
    // and passing context to it
    val db = DBHelper(this, null)
    // below is the variable for cursor
    // we have called method to get
    // all names from our database
    // and add to name text view
    val cursor = db.getName()
```

```
            // moving the cursor to first position and
            // appending value in the text view
            cursor!!.moveToFirst()
            Name.append(cursor.getString(cursor.getColumnIndex(DBHelper.NAME_COl)) + "\n")
            Age.append(cursor.getString(cursor.getColumnIndex(DBHelper.AGE_COL)) + "\n")
            // moving our cursor to next
            // position and appending values
            while(cursor.moveToNext()){
                Name.append(cursor.getString(cursor.getColumnIndex(DBHelper.NAME_COl)) + "\n")
                Age.append(cursor.getString(cursor.getColumnIndex(DBHelper.AGE_COL)) + "\n")
            }
            // at last we close our cursor
            cursor.close()
        }
    }
}
```

5.6 Working with Content Provider

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.



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.

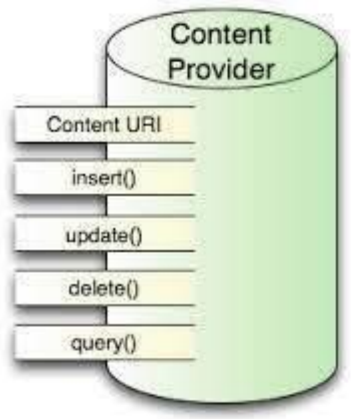
A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions

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.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working –



ContentProvider

- **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.

5.7 Web Service and JSON Parsing

JSON Parsing: JSON (Javascript Object Notation) is a programming language . It is minimal, textual, and a subset of JavaScript. It is an alternative to XML.

Android provides support to parse the JSON object and array.

Advantage of JSON over XML:

1. JSON is faster and easier than xml for AJAX applications.
2. Unlike XML, it is shorter and quicker to read and write.
3. It uses array.

JSON Object: A JSON object contains key/value pairs like map. The keys are strings and the values are the JSON types. Keys and values are separated by comma. The { (curly brace) represents the json object.

```
{
   "employee": {
      "name":       "sachin",
      "salary":     56000,
      "married":    true
   }
}
```

JSON array: The [(square bracket) represents the json array.

```
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

Let's take another example of JSON array.

```
{ "Employee" :
   [
    {"id":"101","name":"Sonoo Jaiswal","salary":"50000"},
    {"id":"102","name":"Vimal Jaiswal","salary":"60000"}
   ]
}
```

Example:

Step1: Create a JSON file index.html.

```
{"info":[
    {"name":"Ajay","id":"111","email":"ajay@gmail.com"},
    {"name":"John","id":"112","email":"john@gmail.com"},
    {"name":"Rahul","id":"113","email":"rahul@gmail.com"},
    {"name":"Maich","id":"114","email":"maich@gmail.com"},
    {"name":"Vikash","id":"115","email":"vikash@gmail.com"},
    {"name":"Mayank","id":"116","email":"mayank@gmail.com"},
    {"name":"Prem","id":"117","email":"prem@gmail.com"},
    {"name":"Chandan","id":"118","email":"chandan@gmail.com"},
    {"name":"Soham","id":"119","email":"soham@gmail.com"},
    {"name":"Mukesh","id":"120","email":"mukesh@gmail.com"},
    {"name":"Ajad","id":"121","email":"ajad@gmail.com"}
    ]
}
```

Step2: Add the ListView in the activity_main.xml layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="example.javatpoint.com.kotlinjsonparsing.MainActivity">

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

  </ListView>

</android.support.constraint.ConstraintLayout>
```

Step3: Add the following okhttp dependency in the build.gradle file.

```
compile 'com.squareup.okhttp3:okhttp:3.8.1'
```

Step4: Create a data model class Model.kt which includes the information String "id", String "name" and String "email".

```
package example.javatpoint.com.kotlinjsonparsing


public class Model{
    lateinit var id:String
    lateinit var name:String
    lateinit var email:String


    constructor(id: String,name:String,email:String) {
        this.id = id
        this.name = name
        this.email = email
    }
    constructor()
}
```

Step5: Create an adapter_layout.xml file in the layout directory which contains the row items for ListView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/linearLayout"
    android:padding="5dp"
    android:orientation="vertical">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/tvId"
        android:layout_margin="5dp"
        android:textSize="16dp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/tvName"
        android:textSize="16dp"
        android:layout_margin="5dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/tvEmail"
        android:layout_margin="5dp"
        android:textSize="16dp"/>

</LinearLayout>
```

Step6: Create a custom adapter class CustomAdapter.kt and extend BaseAdapter to handle the custom ListView.

```
package example.javatpoint.com.kotlinjsonparsing


import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.BaseAdapter
import android.widget.LinearLayout
import android.widget.TextView


class CustomAdapter(context: Context,arrayListDetails:ArrayList<Model>) : BaseAdapter(){


    private val layoutInflater: LayoutInflater
    private val arrayListDetails:ArrayList<Model>
```

```
  init {
      this.layoutInflater = LayoutInflater.from(context)
      this.arrayListDetails=arrayListDetails
  }
  override fun getCount(): Int {
      return arrayListDetails.size
  }
  override fun getItem(position: Int): Any {
      return arrayListDetails.get(position)
  }
  override fun getItemId(position: Int): Long {
      return position.toLong()
  }
  override fun getView(position: Int, convertView: View?, parent: ViewGroup): View? {
      val view: View?
      val listRowHolder: ListRowHolder
      if (convertView == null) {
          view = this.layoutInflater.inflate(R.layout.adapter_layout, parent, false)
          listRowHolder = ListRowHolder(view)
          view.tag = listRowHolder
      } else {
          view = convertView
          listRowHolder = view.tag as ListRowHolder
      }
      listRowHolder.tvName.text = arrayListDetails.get(position).name
      listRowHolder.tvEmail.text = arrayListDetails.get(position).email
      listRowHolder.tvId.text = arrayListDetails.get(position).id
      return view
  }
}

private class ListRowHolder(row: View?) {
  public val tvName: TextView
  public val tvEmail: TextView
  public val tvId: TextView
  public val linearLayout: LinearLayout

  init {
      this.tvId = row?.findViewById<TextView>(R.id.tvId) as TextView
      this.tvName = row?.findViewById<TextView>(R.id.tvName) as TextView
      this.tvEmail = row?.findViewById<TextView>(R.id.tvEmail) as TextView
      this.linearLayout = row?.findViewById<LinearLayout>(R.id.linearLayout) as LinearLayout
  }
}
```

Step7: Add the following code in MainActivity.kt class file. This class read the JSON data in the form of a JSON object. Using the JSON object, we read the JSON array data. The JSON data are bind in ArrayList.

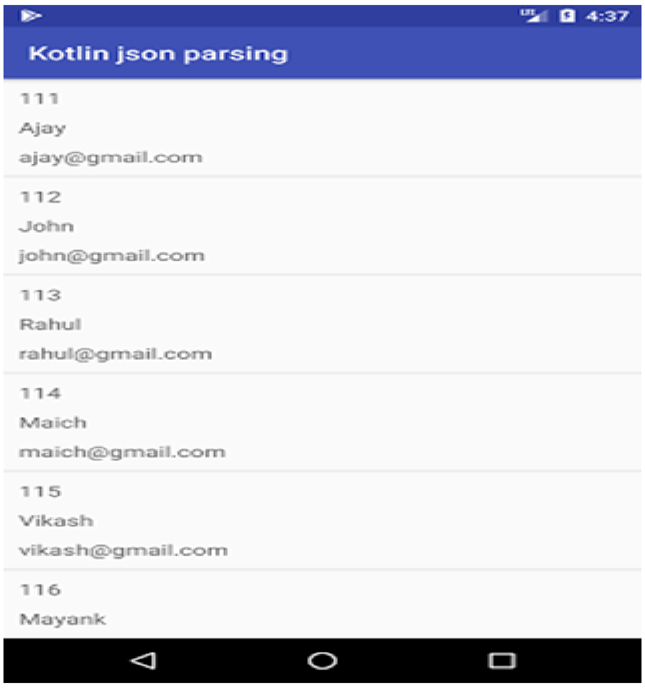
```
package example.javatpoint.com.kotlinjsonparsing


import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.ListView
import android.widget.ProgressBar
import okhttp3.*
import org.json.JSONArray
import org.json.JSONObject
import java.io.IOException
import kotlin.collections.ArrayList

class MainActivity : AppCompatActivity() {
    lateinit var progress:ProgressBar
    lateinit var listView_details: ListView
    var arrayList_details:ArrayList<Model> = ArrayList();
```

```
//OkHttpClient creates connection pool between client and server
val client = OkHttpClient()
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    progress = findViewById(R.id.progressBar)
    progress.visibility = View.VISIBLE
    listView_details = findViewById<ListView>(R.id.listView) as ListView
    run("http://10.0.0.7:8080/jsondata/index.html")
}

fun run(url: String) {
    progress.visibility = View.VISIBLE
    val request = Request.Builder()
            .url(url)
            .build()
    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            progress.visibility = View.GONE
        }

        override fun onResponse(call: Call, response: Response) {
            var str_response = response.body()!!.string()
            //creating json object
            val json_contact:JSONObject = JSONObject(str_response)
            //creating json array
            var jsonarray_info:JSONArray= json_contact.getJSONArray("info")
            var i:Int = 0
            var size:Int = jsonarray_info.length()
            arrayList_details= ArrayList();
            for (i in 0.. size-1) {
                var json_objectdetail:JSONObject=jsonarray_info.getJSONObject(i)
                var model:Model= Model();
                model.id=json_objectdetail.getString("id")
                model.name=json_objectdetail.getString("name")
                model.email=json_objectdetail.getString("email")
                arrayList_details.add(model)
            }

            runOnUiThread {
                //stuff that updates ui
                val obj_adapter : CustomAdapter
                obj_adapter = CustomAdapter(applicationContext,arrayList_details)
                listView_details.adapter=obj_adapter
            }
            progress.visibility = View.GONE
        }
    })
  }
}
```

Step8: Add the Internet permission in AndroidManifest.xml file.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

OutPut:



```
111
Ajay
ajay@gmail.com
112
John
john@gmail.com
113
Rahul
rahul@gmail.com
114
Maich
maich@gmail.com
115
Vikash
vikash@gmail.com
116
Mayank
```