MOBILE APPLICATION DEVELOPMENT LABORATORY

PRACTICAL REPORT FILE

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

BACHELOR OF TECHNOLOGY
(Information Technology)

Submitted By: **Submitted To:**

Vinay Kumar   Prof. Ranjodh Kaur
URN: 1905278   Assistant Professor
CRN: 1921149   Deptt. of Information Technology

Department of Information Technology,
Guru Nanak Dev Engineering College,
Ludhiana-141006

Contents

1. To study design aspects of development environment like Android, iOS.

2. To setup Android studio2 and study its basic components.

3. To study various XML files needed for interface design.

4. To implement different types of layout like relative, grid, linear and table.

5. To incorporate element of interactivity using Android Fragment and Intent Class.

6. To perform database connectivity of android app using SQLite.

7. To implement the concept of multithreading using Android Service class.

8. To implement concept of permission and perform request for permission to access different hardware components of mobile.

9. To perform debugging and testing of android app using tools like Logcat, Android debug bridge, DDMS.

1. To study design aspects of development environment like Android, iOS.

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.
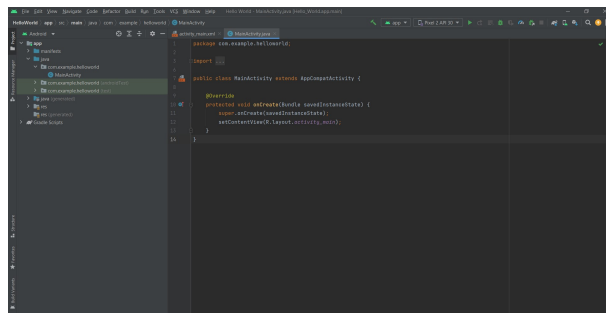
To start coding an Android App, you need an Android development environment. Android Studio is an IDE dedicated to creating and maintaining Android apps Fortunately, Alex did a great job explaining the ins and outs of how it works. As a designer, you're going to want to get comfortable working in the following files:

colors.xml

Much like Sass variables or CSS Custom Properties, you're going to want to abstract as much of your code as possible. Regardless of coding language or medium, this kind of practice helps keep your code consistent, flexible, and easy to maintain.In Android, colors.xml is used to specify all your app's color values. For example, a color called sunset would described like this: #FA5A41.After turning their palette into something Android-friendly, we can now use it in the various pieces that make up the app. Access the contents of colors.xml by first typing @color/, then the color's name:

dimens.xml

We want to abstract our measurements, much like with our color values. In Android, dimens.xml is where we place that code.

Density-independent pixels

1

Android has a unit called density-independent pixels (dp). It's a technical solution that plays to the strengths of their many devices having different display sizes and densities. Unlike declaring a regular static pixel value, a device pixel declaration tells the Android operating system to draw the pixels making up a UI proportionately based on a device's reported display properties.

Scale-independent pixels

Scale-independent pixels (sp) are much like density-independent pixels, only they are used for type. The main difference is that they scale along with a user's font size preference. This is great for helping to make your app accessible for those who benefit from a larger type size.

Abstraction

The client's design system used a suite of standardized spacing values (10px, 20px, 30px, etc.). Since Sketch pixels don't directly translate to Android density-independent pixels, the trick is to abstract these Sketch measurements and codify them as a series of distances:

2dp

4dp

8dp

10dp

12dp

Start from the top

There's a trick in web design where you consistently apply spacing to only the top or bottom of all your interface elements. The idea is it makes a consistent appearance easier, as you don't have to constantly undo and redo your margin declarations and wreck the cascade. Personally, I prefer applying spacing from the top, as an infinite vertically-scrolling canvas means there's always going to be more room below.

styles.xml

Styling components in Android is a highly declarative process. There isn't a real cascade, like what you'd get with CSS, meaning that you're going to spend a lot of time re-describing things

like color and fonts between different interface components.

strings.xml

strings.xml is used to contain the content of your app. If you're picking up on a pattern here, it's that this is yet another centralizing file that controls things. Strings are declared in a fashion similar to colors and measurement: Login as an admin.

They are declared by typing android:text="", then @string/ inside the double quotes, then the string's name:

Commenting and prefixing

Keeping your app code organized helps you quickly locate things and avoid writing duplicate code. This is especially important as the scope of your app grows.As more code is added to the repo, what we can do is liberally apply comments to help break up the walls of code. I personally like to use multiline comments to separate the larger sections, to more quickly identify what's what when scrolling through the page.

Get a phone

Sketch and Android Studio can't directly communicate, meaning that there's the opportunity for drift to occur between your design files and the actual implementation. Android studio has two tools to help combat this: a Design view and a virtual device emulator.

Design view and emulated Android devices

The Design view gives you an in-pane preview of the code you're writing for a specific view. It's good for a quick check, to see if the layout you coded appears properly. However, I've found that when it comes to reviewing fine details, Android Studio simply can't hack it.

Physical hardware

As the saying goes, nothing beats the real thing. On a decent computer, the build time for compiling to an emulator or a physical device was basically the same.

Wrapping up

The client met their deadline, and thanks to Alex's help, I got to learn some new skills. Excel-

lent all around! If you're a designer finding yourself undertaking a similar challenge, hopefully this advice can help you out as well.

IOS

IOS is a mobile operating system for Apple-manufactured devices. iOS runs on the iPhone, iPad, iPod Touch and Apple TV.IOS is best known for serving as the underlying software that allows iPhone users to interact with their phones using gestures such as swiping, tapping and pinching. These finger actions are typically performed on multitouch capacitive touch screen displays, which provide fast response and accept inputs from multiple fingers.

The iPhone X safe area

The iPhone X requires some special attention from designers. The width of the screen is the same as on the iPhone 6, 7 and 8, but it's also 145pt taller. When designing apps for the iPhone X you have to be aware of the safe zone and layout margins to ensure your app user interface doesn't get clipped by the device's sensors or corners.App icons assets are generally added to the application package as plain, squared PNG files in various dimensions. When rendered on a device, iOS applies various effects to app icons.

Rounded Corners

The old simple radii values for rounded corners are gone. Since iOS 7, app icons have been using the shape of a superellipse. When designing app icons for iOS, I recommend to use the official app icon templates provided by Apple.

Border stroke (in some situations)

If the app icon you are using has a white background, a 1 pixel gray border stroke will be applied to make it easier to recognize the edges of the icon. This is only done in the settings app (if your application is listed there) and the AppStore.

Legacy effects (iOS 6 and previous versions)

On older iOS versions, these effects are applied automatically: rounded corners (not the same shape as iOS 7+ icons are using), drop shadows on the home screen and a gloss effect that can be disabled.

Keyboards

There are various keyboard types available to provide the best possible keyboard for a specific text input. While it is possible to build your own completely custom keyboard, default keyboards cannot be customized in style or size.

Grid system
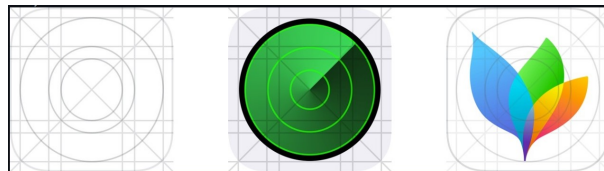
Apple developed a golden ratio grid system that can be used to size and align elements on your icon correctly. Nevertheless, even Apple designers are not following the grid system very strictly with the native apps' icons. Feel free to break the rules if your icon simply works better without aligning all elements strictly to the grid.

Typography

The default system font on all iOS versions previous iOS 9 is Helvetica Neue. With the release of iOS 9, Apple introduced a brand new font called San Francisco, which replaced Helvetica Neue as the default font. San Francisco comes in two shapes: SF UI Display" and SF UI Text", while Display" is primarily used for UI components, Text" features a wider letter spacing and should be used for longer texts. You can download the San Francisco fonts here if you are a member

of Apple's Developer program. In addition to the default font, many alternative font faces are available to use. You can find a complete list of pre-installed typefaces here.

Toolbar

A toolbar contains a set of actions for managing or manipulating the content of the current view. On the iPhone, it will always appear aligned at the bottom edge of the screen, while on the iPad, it can also be displayed aligned at the top of the screen.Similarly to the navigation bar, the background fill of toolbars can be modified, is translucent and blurs the underlaying content by default.

Search Bar

Search bars come in two different styles by default: prominent and minimal. Both versions do have the same functionality. As long as no text was entered by the user, a placeholder text is shown inside the bar, and, optionally, a bookmarks icon that can be used to access recent or saved searches. Once a search term is entered, the placeholder disappears, and a clear button to delete the entered value appears on the right edge.

Table View

Table views are used to display small to large amounts of list style information in a single or multiple columns and with the option to divide several rows into separate sections or to group them. There are two basic table view types that should be used, depending on the type of data you are presenting.

Alerts

The purpose of alerts is to inform the user about critical information and optionally to force the user to make a decision about some action. An alert view does always contain a title text, which should not be longer than one line and one (for pure informational alerts, e.g., OK") or two (for alerts that require a decision, e.g., Send" and Cancel") buttons.

Modals

Modals are a useful view for tasks that require multiple commands or inputs by the user. They

appear on top of everything else, and, while open, block interaction with any other interactive elements underneath. The typical modal usually provides: a title to describe the task; a button to close the modal without saving or performing any other actions; a button to save or submit any entered information.
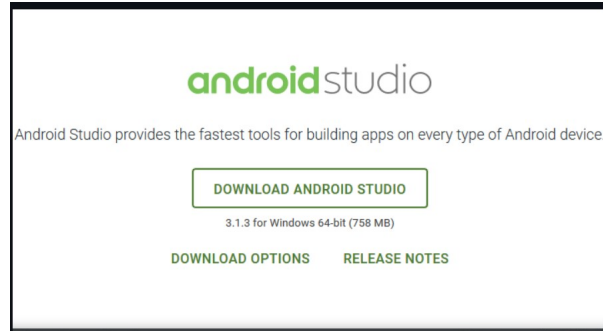
2. To setup Android studio2 and study its basic components.
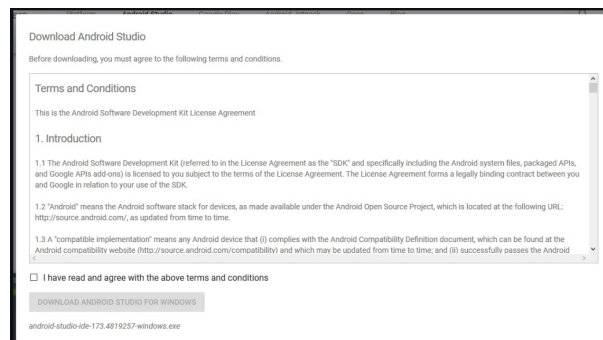
Step – 1 :

Head over to this link https://developer.android.com/studio/downloads to get the Android Studio executable or zip file .

Step – 2 :

Click on the download android studio button.

Click on the "I have read and agree with the above terms and conditions" checkbox followed by the download button.

Click on Save file button in the appeared prompt box and the file will start downloading .

Step – 3:

After the downloading has finished, open the file from downloads and run it. It will prompt the following dialogue box .

Click on next . In the next prompt it'll ask for a path for installation. Choose a path and hit next. Note :The installation path should have the required minimum space.

Step – 4 :

It will start the installation, and once it is completed, it will be like the image shown below .
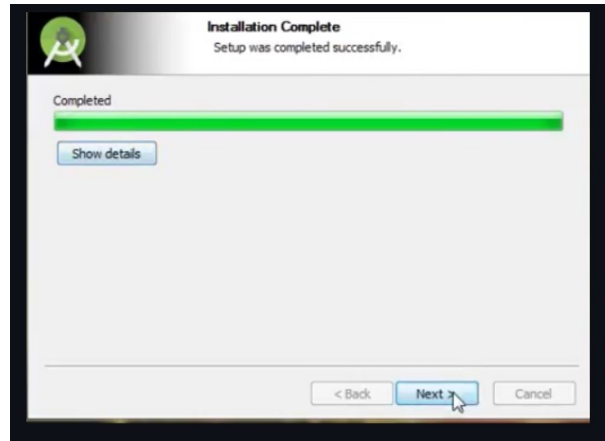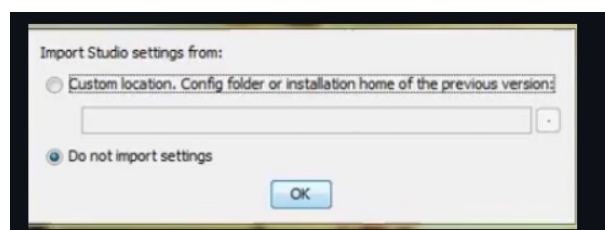

Click on next .

Step – 5 :

Once "Finish" is clicked, it will ask whether the previous settings needs to be impoerted [if android studio had been installed earlier], or not. It is better to choose the 'Don't import Settings option'.
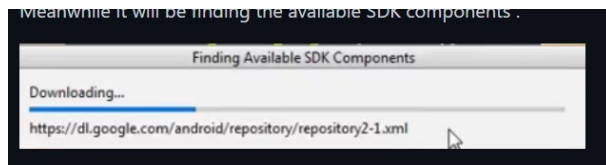

Click the OK button.

Step – 6 :

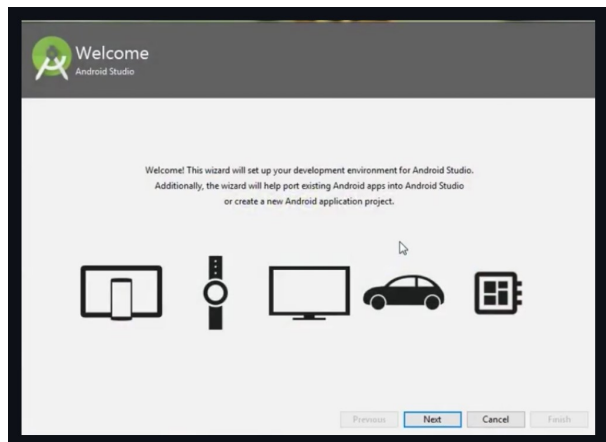This will start the Android Studio.

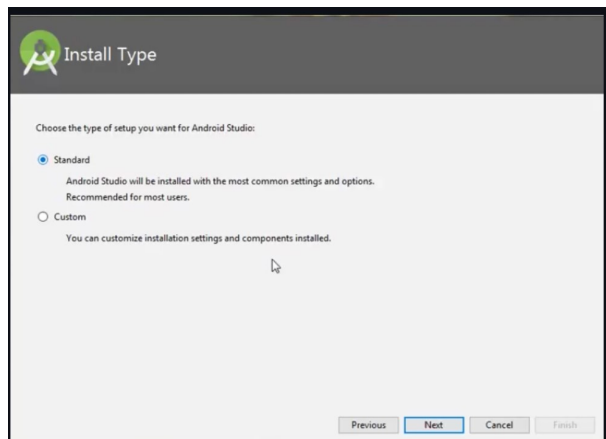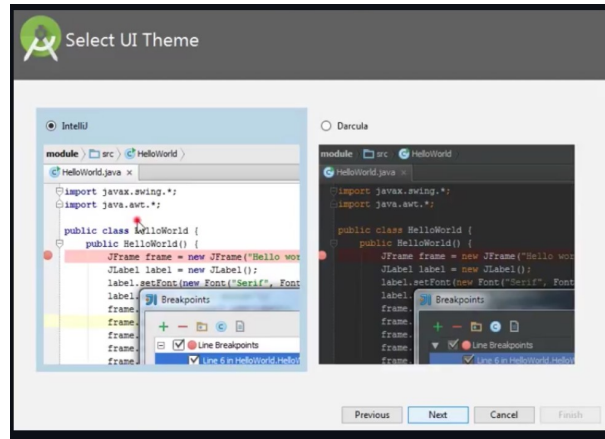Meanwhile it will be finding the available SDK components .

Step – 7:

After it has found the SDK components, it will redirect to the Welcome dialog box .
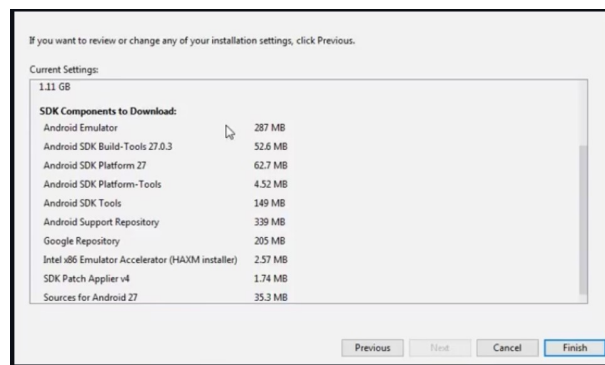

Click on next .

Choose Standard and click on Next. Now choose the theme, whether Light theme or the Dark one. The light one is called the IntelliJ theme whereas the dark theme is called Darcula. Choose as required.
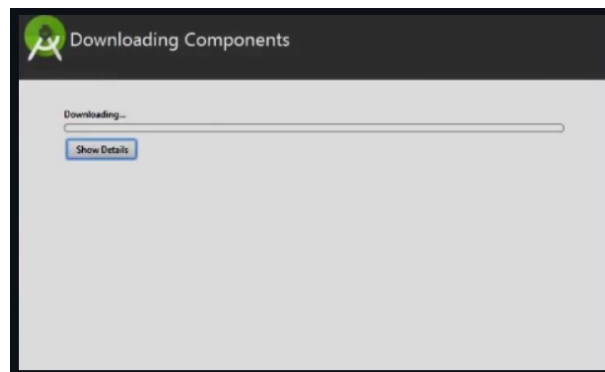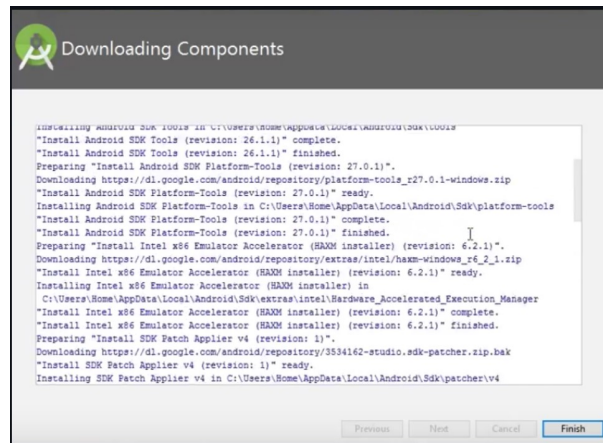

Click on the Next button

Step – 8 :
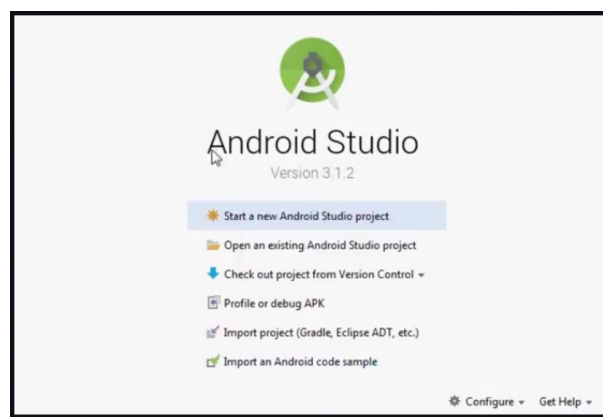Now it is time to download the SDK components .

Click on Finish .

It has started downloading the components

The Android Studio has been successfully configured. Now it's time to launch and build apps. Click on the Finish button to launch it.

Step – 9 :
Click on 'Start new android project' to build a new app.

Components of an Android Application

There are some necessary building blocks that an Android application consists of. These loosely coupled components are bound by the application manifest file which contains description of each component and how they interact. The manifest file also contains the app's metadata, its hardware configuration and platform requirements, external libraries and required permissions. There are the following main components of an android app:-

1. Activities-

Activities are said to be the presentation layer of our applications. The UI of our application is build around one or more extensions of the Activity class. By using Fragments and Views, activities set the layout and display the output and also respond to the users actions.

2. Services-

These are like invisible workers of our app. These components run at backend, updating your data sources and Activities, triggering Notification and also broadcast Intents. They also perform some tasks when applications are not active. A service can be used as a subclass of class Service:

3. Content Providers-

It is used to manage and persist the application data also typically interact with SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured. A content provider should be a sub class of the class ContentProvider.

4. Intents-

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request an action be performed on a particular piece of data.

5. Broadcast Receivers-

They are known to be intent listeners as they enable your application to listen the Intents that satisfy the matching criteria specified by us. Broadcast Receivers make our application to react to any received Intent thereby making them perfect for creating event driven applications.

6. Widgets-

These are the small visual application components that you can find on the home screen of the devices. They are special variation of Broadcast Receivers that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

7. Notifications-

Notifications are the application alerts that are used to draw user's attention to some particular app event without stealing focus or interrupting the current Activity of the user. They are generally used to grab user's attention when the application is not visible or active, particularly from within a Service or Broadcast Receiver. Examples: E-mail popups, Messenger popups etc.

3. To study various XML files needed for interface design.

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. XML tags are not predefined in XML. We must define our own Tags. Xml as itself is well readable both by human and machine. Also, it is scalable and simple to develop. In Android we use xml for designing our layouts because xml is lightweight language so it doesn't make our layout heavy.

Different XML Files Used in Android:

Layout XML Files:

Layout XML files are used to define the actual UI(User interface) of our application. It holds all the elements(views) or the tools that we want to use in our application. Like the TextView's, Button's and other UI elements.

Manifest xml File(Mainfest.xml):

This xml is used to define all the components of our application. It includes the names of our application packages, our Activities, receivers, services and the permissions that our application needs. For Example – Suppose we need to use internet in our app then we need to define Internet permission in this file.

Strings xml File(strings.xml):

This xml file is used to replace the Hard-coded strings with a single string. We define all the strings in this xml file and then access them in our app(Activity or in Layout XML files) from this file. This file enhance the reusability of the code.

Styles xml File(styles.xml):

This xml is used to define different styles and looks for the UI(User Interface) of application. We define our custom themes and styles in this file.

Color xml File (colors.xml):

This file is used to define the color codes that we used in our app. We simply define the color's in this file and used them in our app from this file.

4. To implement different types of layout like relative, grid, linear and table.

1. Linear Layout

The LinearLayout arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally.
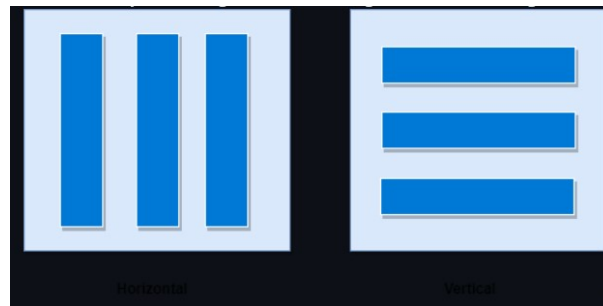
activity-main.xml
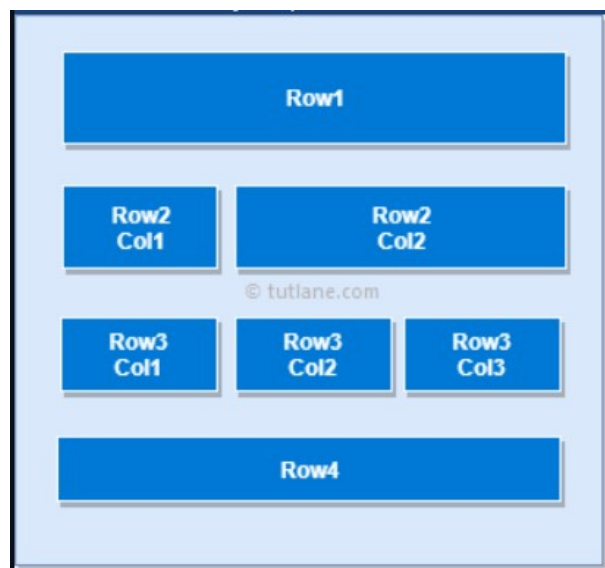
```
1      <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingLeft="16dp"
6      android:paddingRight="16dp"
7      android:orientation="vertical" >
8      <EditText
9          android:layout_width="match_parent"
10         android:layout_height="wrap_content"
11         android:hint="@string/to" />
12     <EditText
13         android:layout_width="match_parent"
14         android:layout_height="wrap_content"
15         android:hint="@string/subject" />
16     <EditText
17         android:layout_width="match_parent"
18         android:layout_height="0dp"
19         android:layout_weight="1"
20         android:gravity="top"
21         android:hint="@string/message" />
22     <Button
23         android:layout_width="100dp"
24         android:layout_height="wrap_content"
25         android:layout_gravity="right"
26         android:text="@string/send" />
27  </LinearLayout>
```

Output

15
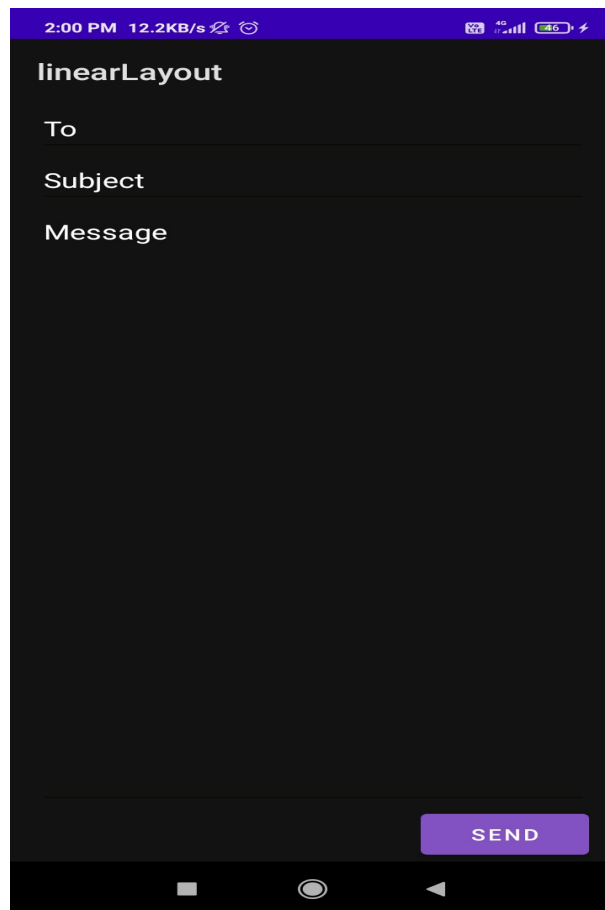
2. Table Layout

The TableLayout groups views into rows and columns. You use the element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.

activity-main.xml

16

```xml
1        <?xml version="1.0" encoding="utf-8"?>
2    <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
3        android:layout_width="match_parent"
4        android:layout_height="match_parent"
5        android:stretchColumns="1">
6
7        <TableRow>
8            <TextView
9                android:layout_column="1"
10                android:text="Open..."
11                android:padding="3dip" />
12            <TextView
13                android:text="Ctrl-O"
14                android:gravity="right"
15                android:padding="3dip" />
16        </TableRow>
17
18        <TableRow>
19            <TextView
20                android:layout_column="1"
21                android:text="Save..."
22                android:padding="3dip" />
23            <TextView
24                android:text="Ctrl-S"
25                android:gravity="right"
26                android:padding="3dip" />
27        </TableRow>
28
29        <TableRow>
30            <TextView
31                android:layout_column="1"
32                android:text="Save As..."
33                android:padding="3dip" />
34            <TextView
35                android:text="Ctrl-Shift-S"
36                android:gravity="right"
37                android:padding="3dip" />
38        </TableRow>
39
40        <View
41            android:layout_height="2dip"
42            android:background="#FF909090" />
43
44        <TableRow>
45            <TextView
46                android:text="X"
47                android:padding="3dip" />
48            <TextView
49                android:text="Import..."
50                android:padding="3dip" />
51        </TableRow>
52
53        <TableRow>
54            <TextView
55                android:text="X"
56                android:padding="3dip" />
57            <TextView
58                android:text="Export..."
59                android:padding="3dip" />
```

```
60          <TextView
61              android:text="Ctrl -E"
62              android:gravity="right"
63              android:padding="3dip" />
64      </TableRow>
65
66      <View
67          android:layout_height="2dip"
68          android:background="#FF909090" />
69
70      <TableRow>
71          <TextView
72              android:layout_column="1"
73              android:text="Quit"
74              android:padding="3dip" />
75      </TableRow>
76  </TableLayout>
```

Output
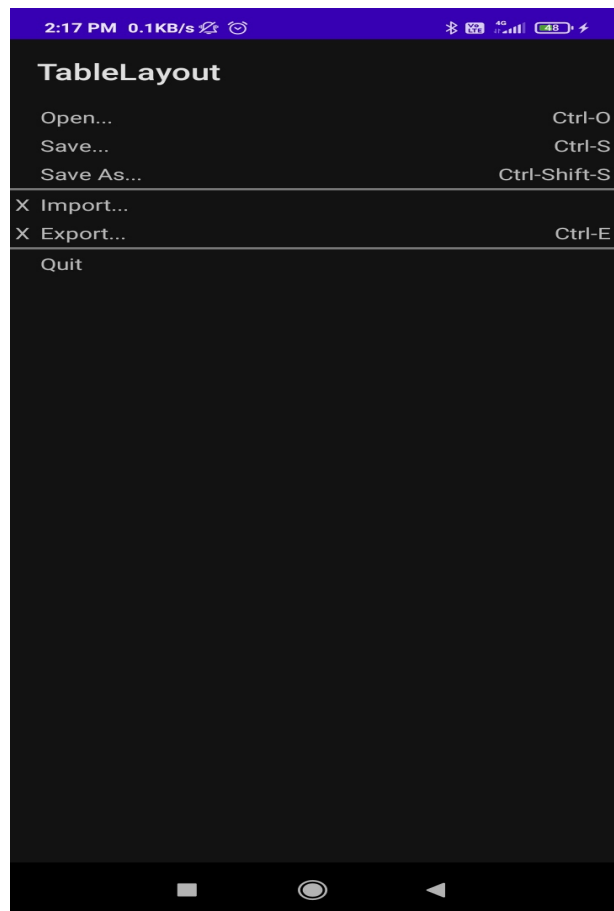
3. Relative Layout

The RelativeLayout enables you to specify how child views are positioned relative to each other.

activity-main.xml

```
1    <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      xmlns:android="http://schemas.android.com/apk/res/android">
6
7      <Button
8          android:id="@+id/button1"
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content"
11         android:text="Top Left Button"
12         android:layout_alignParentLeft="true"
13         android:layout_alignParentTop="true"/>
14     <Button
15         android:id="@+id/button2"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:text="Top Right Button"
19         android:layout_alignParentTop="true"
20         android:layout_alignParentRight="true"/>
21
22     <Button
23         android:id="@+id/button3"
24         android:layout_width="174dp"
25         android:layout_height="55dp"
26         android:layout_alignParentLeft="true"
27         android:layout_alignParentBottom="true"
28         android:layout_marginBottom="0dp"
29         android:text="Bottom Left Button" />
30
31     <Button
32         android:id="@+id/button4"
33         android:layout_width="171dp"
34         android:layout_height="wrap_content"
35         android:layout_alignParentRight="true"
36         android:layout_alignParentBottom="true"
37         android:layout_marginBottom="0dp"
38         android:text="Bottom Right Button" />
39     <Button
```
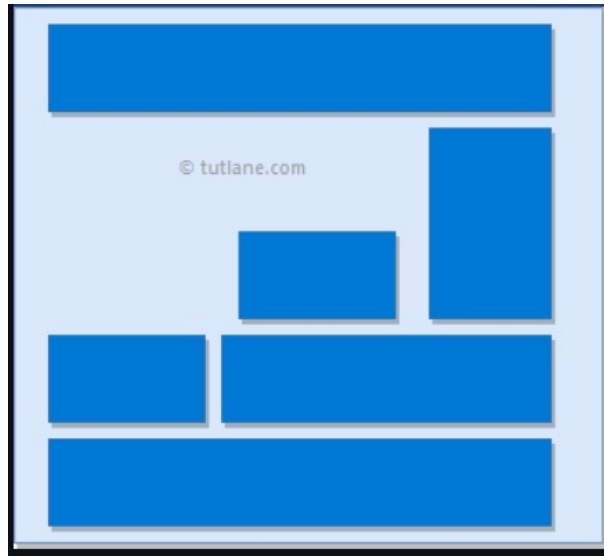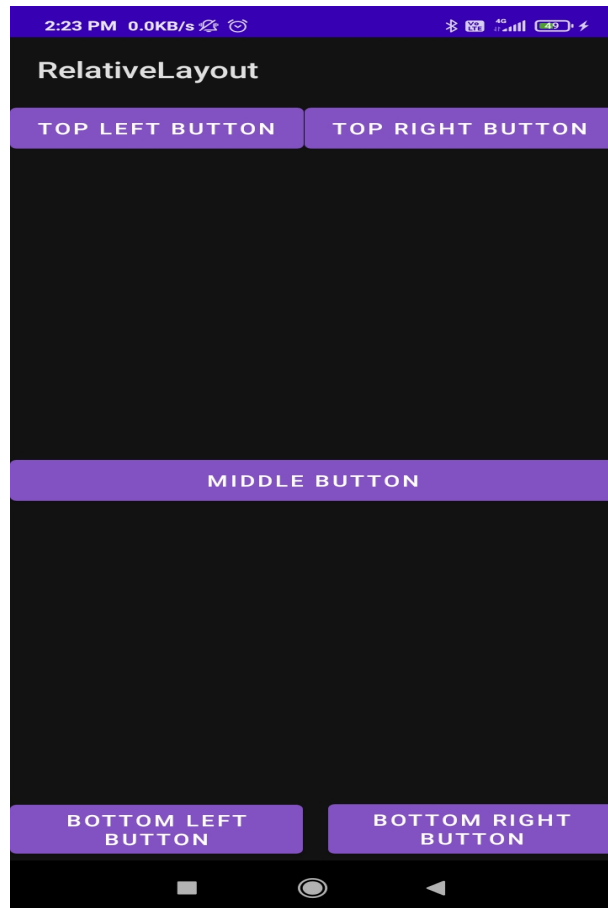
19

```
40              android:id="@+id/button5"
41              android:layout_width="fill_parent"
42              android:layout_height="wrap_content"
43              android:text="Middle Button"
44              android:layout_centerVertical="true"
45              android:layout_centerHorizontal="true"/>
46
47  </RelativeLayout>
```

Output

5. To incorporate element of interactivity using Android Fragment and Intent Class.

Android Intent

It is the message that is passed between components such as activities, content providers, broadcast receivers, services etc. It is generally used with startActivity() method to invoke activity, broadcast receivers etc. The LabeledIntent is the subclass of android.content.Intent class. Android intents are mainly used to: o Start the service o Launch an activity o Display a web page o Display a list of contacts o Broadcast a message o Dial a phone call etc.

Types of Android Intents There are two types of intents in android:

1. Implicit Intent: Implicit Intent doesn't specify the component. In such case, intent provides information of available components provided by the system that is to be invoked.

code

```
1    String url=editText.getText().toString();
2    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
3    startActivity(intent);
```

Output
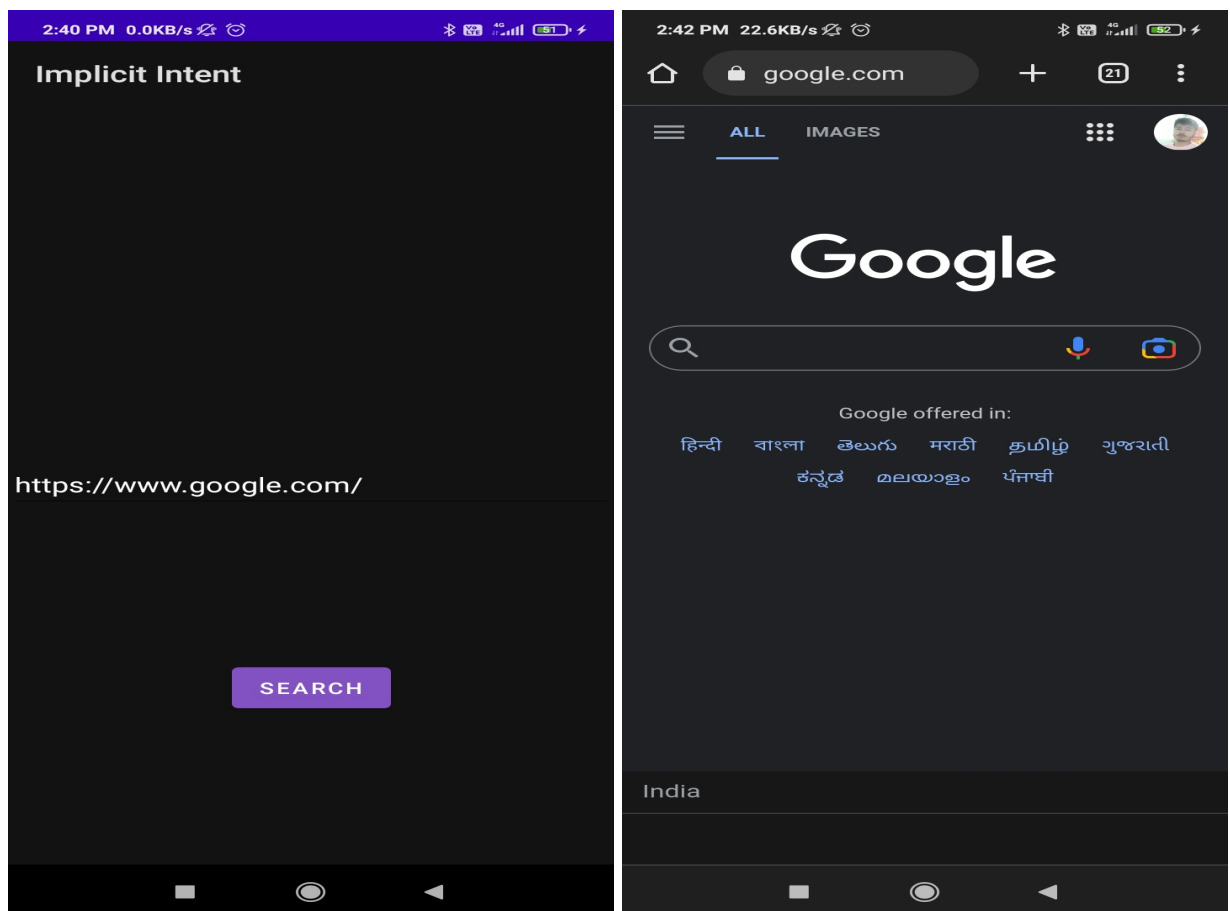
2.Explicit Intent Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

code

```
1   public void callFirstActivity(View view) {
2       Intent i = new Intent(getApplicationContext(), MainActivity.class);
3       startActivity(i);
4   }
5   public void callSecondActivity(View view){
6       Intent i = new Intent(getApplicationContext(), MainActivityTwo.class);
7       startActivity(i);
8   }
```

Output
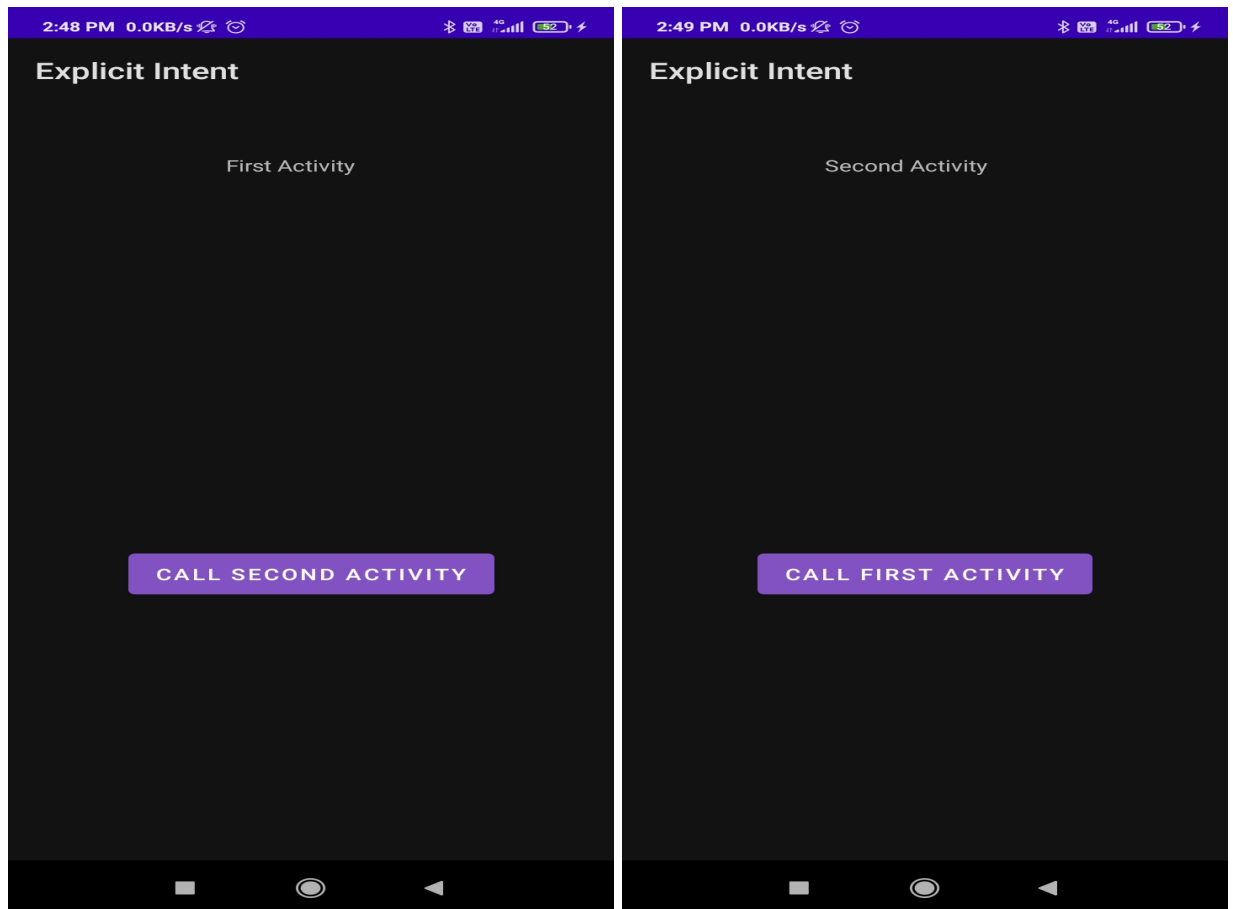
Android Fragments

Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity. Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity. Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.

Output
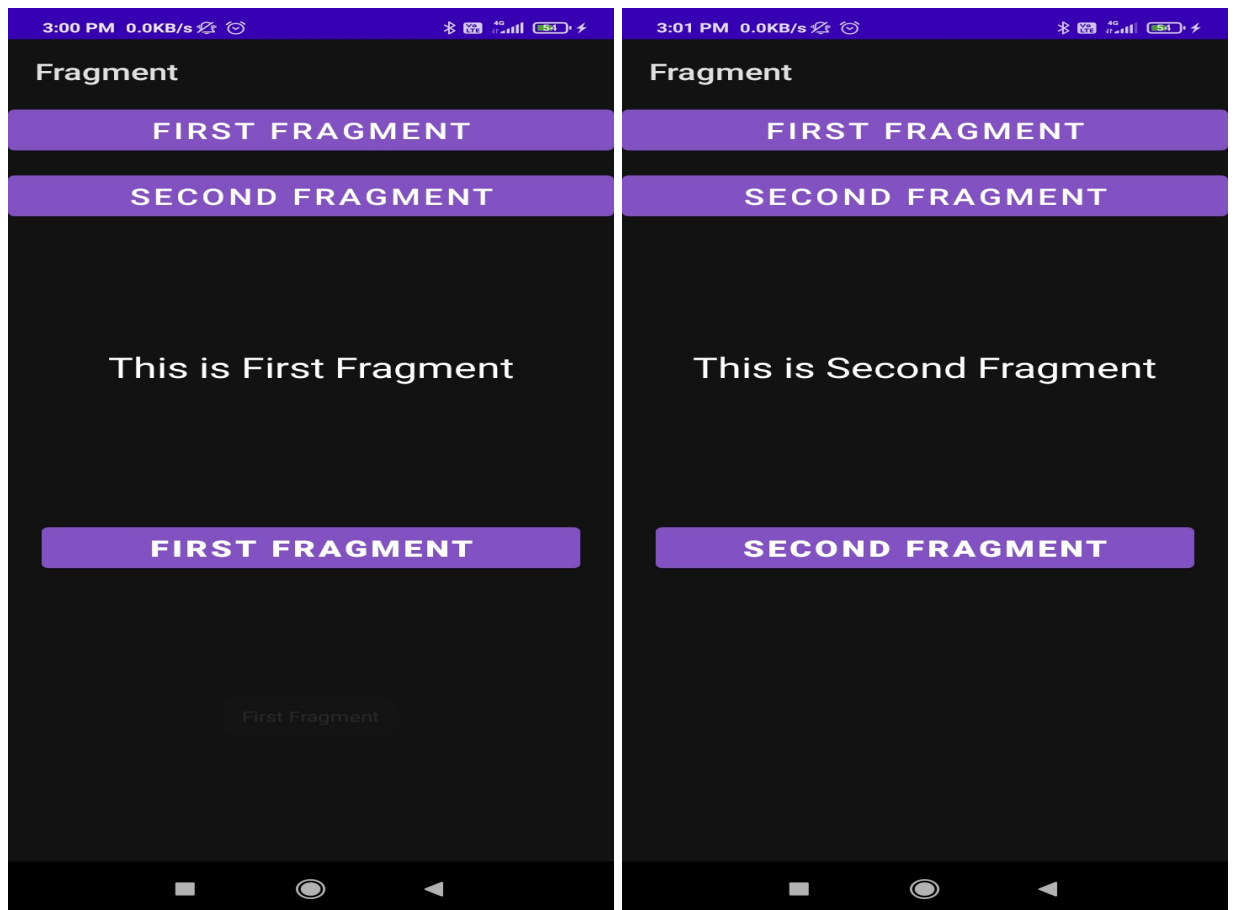
6. To perform database connectivity of android app using SQLite.

Android has built in SQLite database implementation. It is available locally over the device(mobile & tablet) and contain data in text format. It carry light weight data and suitable with many languages. So, it doesn't required any administration or setup procedure of the database.

Creating And Updating Database In Android

For creating, updating and other operations you need to create a subclass or SQLiteOpenHelper class. SQLiteOpenHelper is a helper class to manage database creation and version management. It provides two methods onCreate(SQLiteDatabase db), onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion).

The SQLiteOpenHelper is responsible for opening database if exist, creating database if it does not exists and upgrading if required. The SQLiteOpenHelper only require the DATABASE-NAME to create database. After extending SQLiteOpenHelper you will need to implement its methods onCreate, onUpgrade and constructor.

onCreate(SQLiteDatabase sqLiteDatabase)

This method is called only once throughout the application lifecycle. It will be called whenever there is a first call to getReadableDatabase() or getWritableDatabase() function available in super SQLiteOpenHelper class. So SQLiteOpenHelper class call the onCreate() method after creating database and instantiate SQLiteDatabase object. Database name is passed in constructor call.

onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion)

This is only called whenever there is a updation in existing version. So to update a version we have to increment the value of version variable passed in the superclass constructor.

In onUpgrade method we can write queries to perform whatever action is required. In most example you will see that existing table(s) are being dropped and again onCreate() method is being called to create tables again. But it's not mandatory to do so and it all depends upon your requirements.

We have to change database version if we have added a new row in the database table. If we have requirement that we don't want to lose existing data in the table then we can write alter table query in the onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion) method.

activity-main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:id="@+id/activity_main"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:background="@android:color/holo_blue_dark">
8
9      <TextView
10         android:text="USERNAME"
```

```
11          android:layout_width="match_parent"
12          android:layout_height="wrap_content"
13          android:layout_alignParentTop="true"
14          android:layout_marginTop="12dp"
15          android:id="@+id/textView"
16          android:textSize="18sp"
17          android:textStyle="bold|italic"
18          android:layout_alignParentLeft="true"
19          android:layout_alignParentStart="true"
20          android:gravity="center" />
21
22      <EditText
23          android:layout_width="match_parent"
24          android:layout_height="wrap_content"
25          android:inputType="textPersonName"
26          android:ems="10"
27          android:id="@+id/editName"
28          android:textStyle="bold|italic"
29          android:layout_below="@+id/textView"
30          android:layout_alignParentRight="true"
31          android:layout_alignParentEnd="true"
32          android:hint="Enter Name"
33          android:gravity="center_vertical|center" />
34
35      <TextView
36          android:text="PASSWORD"
37          android:layout_width="match_parent"
38          android:layout_height="wrap_content"
39          android:layout_marginTop="13dp"
40          android:id="@+id/textView2"
41          android:textStyle="bold|italic"
42          android:textSize="18sp"
43          android:layout_below="@+id/editName"
44          android:layout_alignParentRight="true"
45          android:layout_alignParentEnd="true"
46          android:gravity="center"
47          android:hint="Enter Password" />
48
49      <Button
50          android:text="VIEW DATA"
51          android:layout_width="wrap_content"
52          android:layout_height="wrap_content"
53          android:id="@+id/button2"
54          android:textSize="18sp"
55          android:onClick="viewdata"
56          android:textStyle="bold|italic"
57          android:layout_alignBaseline="@+id/button"
58          android:layout_alignBottom="@+id/button"
59          android:layout_alignRight="@+id/button4"
60          android:layout_alignEnd="@+id/button4" />
61
62      <Button
63          android:text="ADD USER"
64          android:layout_width="wrap_content"
65          android:layout_height="wrap_content"
66          android:id="@+id/button"
67          android:textStyle="bold|italic"
68          android:textSize="18sp"
69          android:onClick="addUser"
```

```
70          android:layout_marginLeft="28dp"
71          android:layout_marginStart="28dp"
72          android:layout_below="@+id/editPass"
73          android:layout_alignParentLeft="true"
74          android:layout_alignParentStart="true"
75          android:layout_marginTop="23dp" />
76
77      <Button
78          android:text="UPDATE"
79          android:layout_width="wrap_content"
80          android:layout_height="wrap_content"
81          android:id="@+id/button3"
82          android:onClick="update"
83          android:textStyle="normal|bold"
84          android:layout_below="@+id/editText3"
85          android:layout_alignLeft="@+id/button4"
86          android:layout_alignStart="@+id/button4"
87          android:layout_marginTop="13dp" />
88
89      <EditText
90          android:layout_width="wrap_content"
91          android:layout_height="wrap_content"
92          android:inputType="textPersonName"
93          android:ems="10"
94          android:id="@+id/editText6"
95          android:layout_alignTop="@+id/button4"
96          android:layout_alignParentLeft="true"
97          android:layout_alignParentStart="true"
98          android:freezesText="false"
99          android:hint="Enter Name to Delete Data"
100         android:layout_toLeftOf="@+id/button2"
101         android:layout_toStartOf="@+id/button2" />
102
103     <Button
104         android:text="DELETE"
105         android:layout_width="wrap_content"
106         android:layout_height="wrap_content"
107         android:layout_marginRight="21dp"
108         android:layout_marginEnd="21dp"
109         android:id="@+id/button4"
110         android:onClick="delete"
111         android:textStyle="normal|bold"
112         tools:ignore="RelativeOverlap"
113         android:layout_marginBottom="41dp"
114         android:layout_alignParentBottom="true"
115         android:layout_alignParentRight="true"
116         android:layout_alignParentEnd="true" />
117
118     <EditText
119         android:layout_width="wrap_content"
120         android:layout_height="wrap_content"
121         android:inputType="textPersonName"
122         android:ems="10"
123         android:layout_marginTop="47dp"
124         android:id="@+id/editText3"
125         android:textStyle="bold|italic"
126         android:textSize="14sp"
127         android:layout_below="@+id/button"
128         android:layout_alignParentLeft="true"
```

```
129            android:layout_alignParentStart="true"
130            android:layout_marginLeft="7dp"
131            android:layout_marginStart="7dp"
132            android:hint="Current_Name" />
133
134      <EditText
135            android:layout_width="match_parent"
136            android:layout_height="wrap_content"
137            android:inputType="textPassword"
138            android:ems="10"
139            android:layout_marginTop="11dp"
140            android:id="@+id/editPass"
141            android:hint="Enter_Password"
142            android:gravity="center_vertical|center"
143            android:textSize="18sp"
144            android:layout_below="@+id/textView2"
145            android:layout_alignParentLeft="true"
146            android:layout_alignParentStart="true"
147            android:textAllCaps="false"
148            android:textStyle="normal|bold" />
149
150      <EditText
151            android:layout_width="wrap_content"
152            android:layout_height="wrap_content"
153            android:inputType="textPersonName"
154            android:ems="10"
155            android:id="@+id/editText5"
156            android:textStyle="bold|italic"
157            android:textSize="14sp"
158            android:hint="New_Name"
159            android:layout_alignTop="@+id/button3"
160            android:layout_alignLeft="@+id/editText3"
161            android:layout_alignStart="@+id/editText3"
162            android:layout_marginTop="32dp" />
163  </RelativeLayout>
```

Output

7. To implement the concept of multithreading using Android Service class.

Introduction:

Multi-threading is defined as a feature through which we can run two or more concurrent threads of a process. In this a process, the common data is shared among all these threads also known as sub-processes exclusively. In android there are many ways through which multi-threading can be established in the application.

Objective:

- Understanding the basic concept of multithreading.

- Understanding of Handler class in android.

- Understanding of Runnable Interface.

Multi-Threading In Android:

Multi-Threading in Android is a unique feature through which more than one threads execute together without hindering the execution of other threads. Multi-Threading in Android is not different from conventional multi-Threading. A class can be thought of as a process having its method as it's sub-processes or threads. All these methods can run concurrently by using feature of Multi-Threading. In android, multi-Threading can be achieved through the use of many in-built classes. Out of them, Handler class is most commonly used.

Handler Class In Android:

Handler class come from the Package android.os.Handler package and is most commonly used for multi-threading in android. Handler class provide sending and receiving feature for messages between different threads and handle the thread execution which is associated with that instance of Handler class. In android class, every thread is associated with an instance of Handler class and it allows the thread to run along with other threads and communicate with them through messages.

Runnable Interface:

Runnable interface is used in multi-threading to be called in a loop when the thread starts. It is a type of thread that executes the statement in its body or calls other methods for a specified or infinite number of times.This runable interface is used by the Handler class to execute the multi-threading, i.e., to execute one or more thread in specified time.Runnable is an interface which is implemented by the class desired to support multithreading and that class must implements it's abstract method public void run().Run() method is the core of multithreading as it includes the statement or calls to other methods that the thread needs to be made for multithreading.

Steps Involved in making project on Multi-Threading: Step 1:

Create a new project in Android Studio, go to File New Project and fill all required details to create a new project.

Step 2:

Add the following code to res layout activity-main.xml.

activity-main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <?xml version="1.0" encoding="utf-8"?>
3  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:orientation="vertical"
8      android:gravity="center_horizontal"
9      android:layout_marginTop="100dp"
10     tools:context=".MainActivity">
11     <EditText
12         android:id="@+id/edit_query"
13         android:layout_width="match_parent"
14         android:layout_height="wrap_content"
15         android:hint="Enter string" />
16     <Button
17         android:id="@+id/click"
18         android:layout_marginTop="50dp"
19         style="@style/Base.TextAppearance.AppCompat.Widget.
20  Button.Borderless.Colored"
21         android:layout_width="wrap_content"
22         android:background="#c1c1c1"
23         android:textColor="#FFF"
24         android:layout_height="wrap_content"
25         android:text="Button" />
26     <TextView
27         android:id="@+id/text"
28         android:layout_width="wrap_content"
29         android:layout_height="wrap_content" />
30     <TextView
31         android:id="@+id/text1"
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content" />
34  </LinearLayout>
```

Step 3:

Add the following code to src MainActivity.java

code

```
1      import android.os.Bundle;
2   import android.support.v7.app.AppCompatActivity;
3   import android.view.View;
4   import android.widget.EditText;
5   import android.widget.TextView;
6
7   public class MainActivity extends AppCompatActivity {
8       EditText edit_query;
9       TextView textView;
10      TextView text1;
11      boolean twice = false;
12      Thread t = null;
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
```

```
16            super.onCreate(savedInstanceState);
17            setContentView(R.layout.activity_main);
18            edit_query = findViewById(R.id.edit_query);
19            textView = findViewById(R.id.text);
20            text1 = findViewById(R.id.text1);
21            findViewById(R.id.click).setOnClickListener
22    (new View.OnClickListener() {
23                @Override
24                public void onClick(View v) {
25                    runthread();
26                    runthread1();
27                }
28            });
29        }
30
31        private void runthread1() {
32            runOnUiThread(new Runnable() {
33                @Override
34                public void run() {
35                    try {
36                        Thread.sleep(5000);
37                    } catch (InterruptedException e) {
38                        e.printStackTrace();
39                    }
40                    text1.setText("tutorialspoint.com");
41                }
42            });
43        }
44
45        private void runthread() {
46            twice = true;
47                if (twice) {
48                    final String s1 = edit_query.getText().toString();
49                    t = new Thread(new Runnable() {
50                    @Override
51                    public void run() {
52                        runOnUiThread(new Runnable() {
53                            @Override
54                            public void run() {
55                                textView.setText(t.getName());
56                                twice = false;
57                            }
58                        });
59                    }
60                });
61                t.start();
62                t.setName(s1);
63                t.setPriority(Thread.MAX_PRIORITY);
64            }
65        }
66    }
```

Output
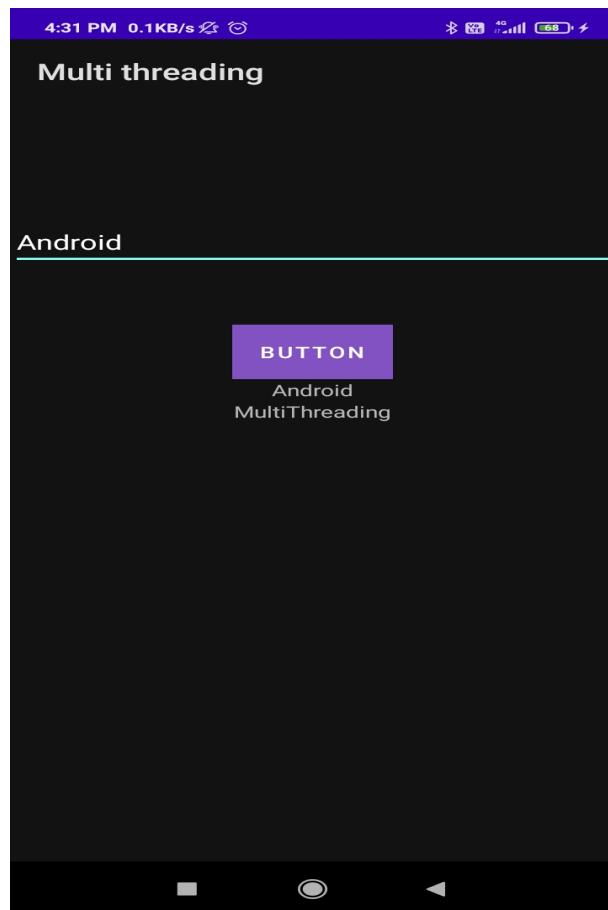
8. To implement concept of permission and perform request for permission to access different hardware components of mobile.

Permission:

The purpose of a permission is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

Permission approval :

An app must publicize the permissions it requires by including tags in the app manifest. For example, an app that needs to access location would have this line in the manifest:

Example:

manifest.xml

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

<application ...>
    ...
</application>
```

Code:

```java
1  public class MainActivity extends AppCompatActivity {
2
3      // Defining Buttons
4      private Button storage, camera;
5
6      // Defining Permission codes.
7      // We can give any value
8      // but unique for each permission.
9      private static final int CAMERA_PERMISSION_CODE = 100;
10     private static final int STORAGE_PERMISSION_CODE = 101;
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState)
14     {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         storage = findViewById(R.id.storage);
19         camera = findViewById(R.id.camera);
20
21         // Set Buttons on Click Listeners
22         storage.setOnClickListener(new View.OnClickListener() {
23             @Override
24             public void onClick(View v)
25             {
26                 checkPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE,
                        STORAGE_PERMISSION_CODE);
27             }
28         });
29
30         camera.setOnClickListener(new View.OnClickListener() {
```

33

```
31              @Override
32              public void onClick(View v)
33              {
34                  checkPermission(Manifest.permission.CAMERA,
                        CAMERA_PERMISSION_CODE);
35              }
36          });
37      }
38
39      // Function to check and request permission.
40      public void checkPermission(String permission, int requestCode)
41      {
42          if (ContextCompat.checkSelfPermission(MainActivity.this, permission)
                == PackageManager.PERMISSION_DENIED) {
43
44              // Requesting the permission
45              ActivityCompat.requestPermissions(MainActivity.this, new String[]
                    { permission }, requestCode);
46          }
47          else {
48              Toast.makeText(MainActivity.this, "Permission already granted",
                    Toast.LENGTH_SHORT).show();
49          }
50      }
51
52      // This function is called when the user accepts or decline the
            permission.
53      // Request Code is used to check which permission called this function.
54      // This request code is provided when the user is prompt for permission.
55
56      @Override
57      public void onRequestPermissionsResult(int requestCode, @NonNull String[]
            permissions, @NonNull int[] grantResults)
58      {
59          super.onRequestPermissionsResult(requestCode,
60                  permissions,
61                  grantResults);
62
63          if (requestCode == CAMERA_PERMISSION_CODE) {
64              if (grantResults.length > 0 && grantResults[0] == PackageManager.
                    PERMISSION_GRANTED) {
65                  Toast.makeText(MainActivity.this, "Camera Permission Granted"
                        , Toast.LENGTH_SHORT).show();
66              }
67              else {
68                  Toast.makeText(MainActivity.this, "Camera Permission Denied",
                        Toast.LENGTH_SHORT).show();
69              }
70          }
71          else if (requestCode == STORAGE_PERMISSION_CODE) {
72              if (grantResults.length > 0
73                      && grantResults[0] == PackageManager.PERMISSION_GRANTED)
                        {
74                  Toast.makeText(MainActivity.this, "Storage Permission Granted
                        ", Toast.LENGTH_SHORT).show();
75              } else {
76                  Toast.makeText(MainActivity.this, "Storage Permission Denied"
                        , Toast.LENGTH_SHORT).show();
77              }
```
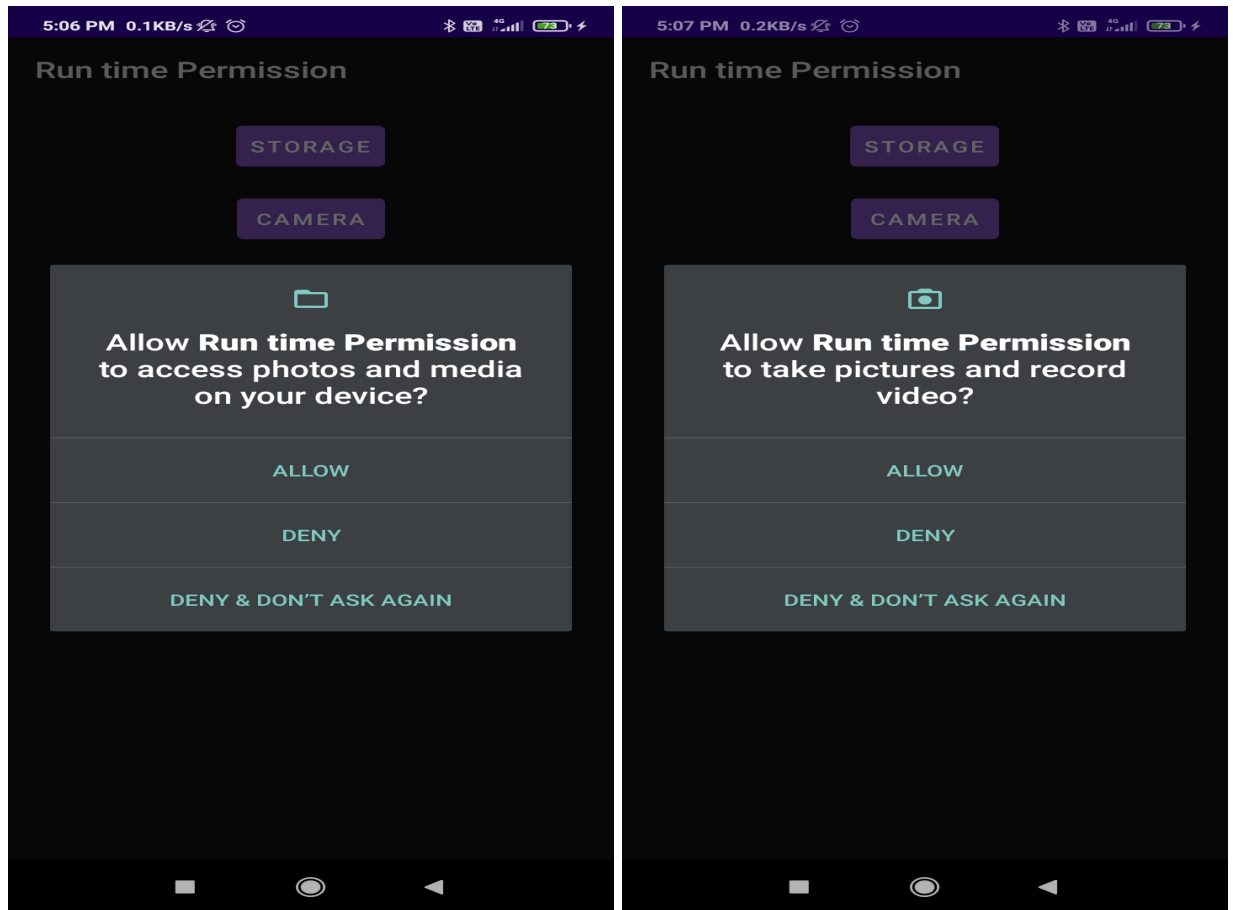
```
78                }
79            }
80  }
```

Output

9. To perform debugging and testing of android app using tools like Logcat, Android debug bridge, DDMS.

Debugging and Testing:

Debugging is the process of finding and fixing errors (bugs) or unexpected behaviour in our code. All code has bugs, from incorrect behaviour in our app, to behaviour that excessively consumes memory or network resources, to actual app freezing or crashing.

Objective:

- Errors in design or implementation

- Android framework limitations (or bugs).

- Missing requirements or assumptions for how the app should work

- Device limitations (or bugs)

Use the debugging, testing, and profiling capabilities in Android Studio to help us reproduce, find, and resolve all of these problems. Those capabilities include:

- Logcat

- Android Debug Bridge

- DDMS (Dalvik Debug Monitor Server)

LOGCAT:

We can use log class to send messages to the Android system log, and view those messages in Android Studio in the Logcat pane. The Android SDK includes a useful logging utility class called android.util.Log. The class allows us to log messages categorized based severity; each type of logging message has its own message. Here is a listing of the message types, and their respective method calls, ordered from lowest to highest priority:

- The Log.v() method is used to log verbose messages.

- The Log.d() method is used to log debug messages.

- The Log.i() method is used to log informational messages.

- The Log.w() method is used to log warnings.

- The Log.e() method is used to log errors.

- The Log.wtf() method is used to log events that should never happen
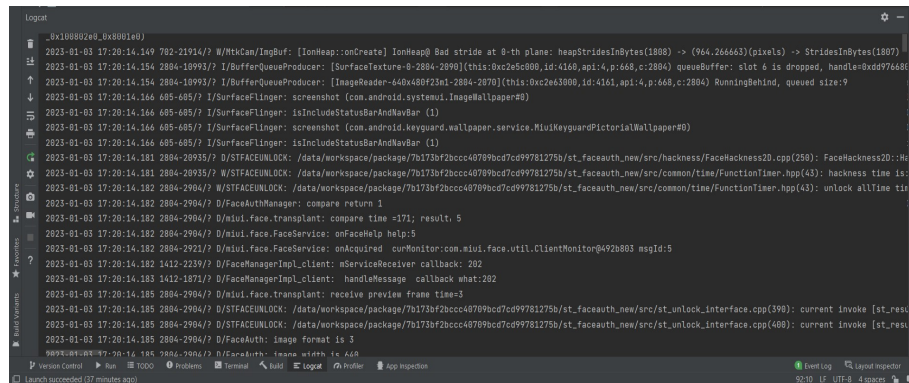
Android Debug Bridge:

Android Debug Bridge (adb) is a versatile command-line tool that lets us communicate with a device. ADB is a part of Android SDK The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that we can use to run a variety of commands on a device. It is a client-server program that includes three components:

- A client, which sends commands. The client runs on development machine. We can invoke a client from a command-line terminal by issuing an adb command.

- A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device.

- A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.
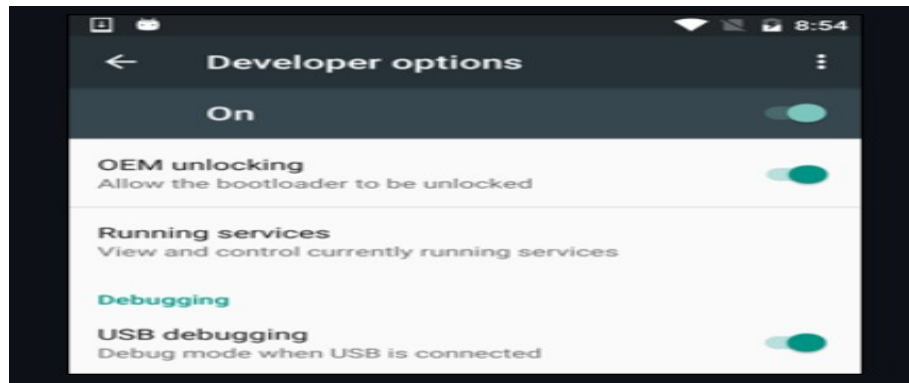
Enable adb debugging:

We have to use USB debugging under Developer Options for using adb with a device. We can now connect our device with USB. We can verify that our device is connected by executing adb devices from the android-sdk platform-tools directory.
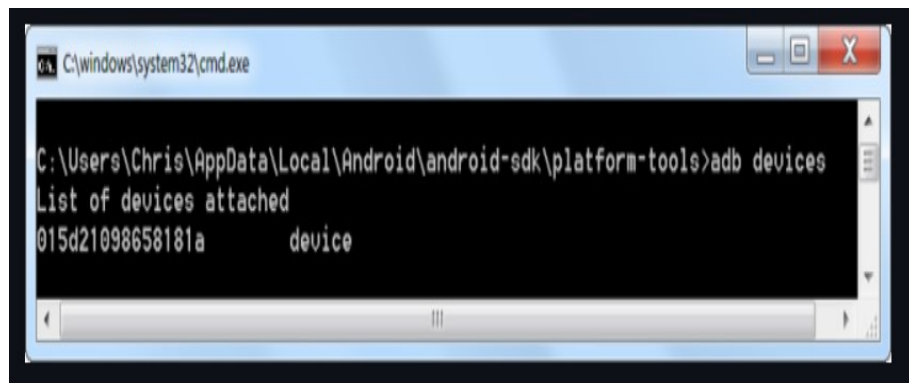
Enable adb debugging:
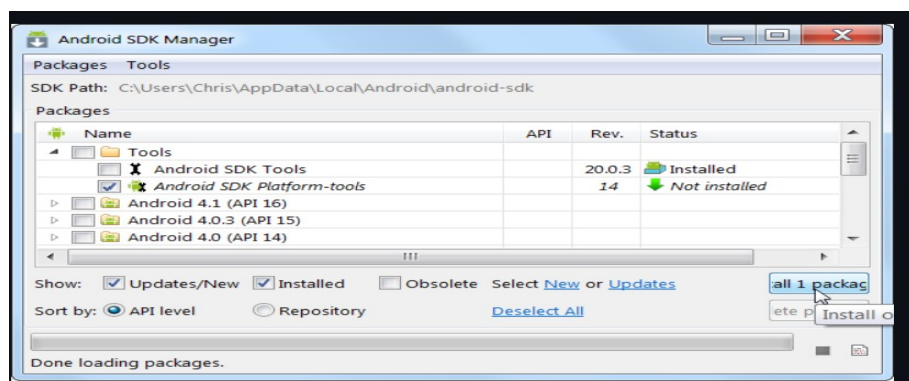
Enable USB debugging:

Test ADB connection:

Now, we can run ADB commands to use the tool in console for both Debugging and Testing of the application.

DDMS (Dalvik Debug Monitor Server):

The DDMS is a debugging tool used in the Android platform. The Dalvik Debug Monitor Service is downloaded as part of the Android SDK. Some of the services provided by the DDMS are port forwarding, on-device screen capture, on-device thread and heap monitoring, and radio state information. It allows developers to spot bugs in applications running on either an emulator or an actual Android device. Its feature, known as Emulator Control allows developers to simulate phone states and activities. For example, it can simulate different types of networks which can have different network characteristics such as speed and latency. This debugging tool can be integrated into the Eclipse IDE by adding the ADT (Android Development Tools) plug-in. Otherwise, it can be accessed from the command line and will automatically connect to any running emulator.