UNIT II: Android Studio and User Interface Design

2.1 Android Studio and its Features

2.2 Introduction to Activities and Activity Lifecycle

2.3 Working with the AndroidManifest.xml

2.4 Using the log system

2.5 Views and View Groups

2.6 Linear Layout, Relative Layout,

2.7 TableLayout, Constraint Layout, Frame Layout, Scroll Layout, Scroll View

2.1 Android Studio and its Features

Android Studio is Android's official IDE. It is purpose-built for Android to accelerate your development and help you build the highest-quality apps for every Android device.

Android Studio was announced on 16th May 2013 at the Google I/O conference as an official IDE for Android app development. It started its early access preview from version 0.1 in May 2013. The first stable built version was released in December 2014, starts from version 1.0.

Features of Android Studio

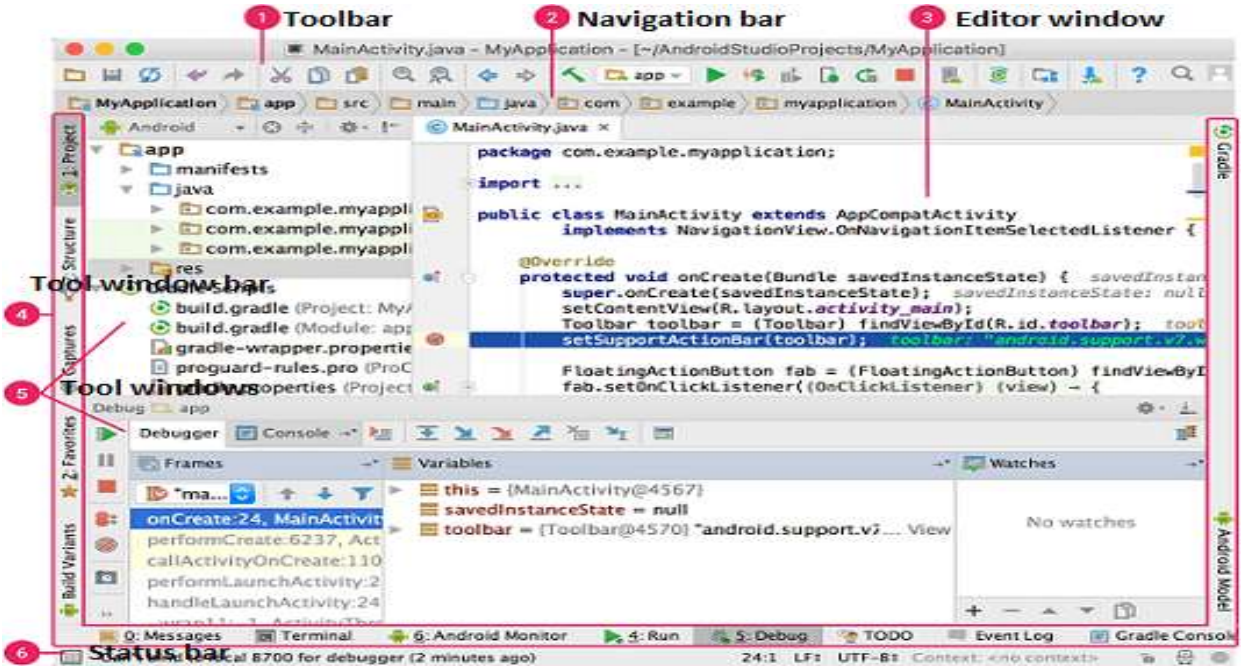
It has a flexible Gradle-based build system.

It has a fast and feature-rich emulator for app testing.

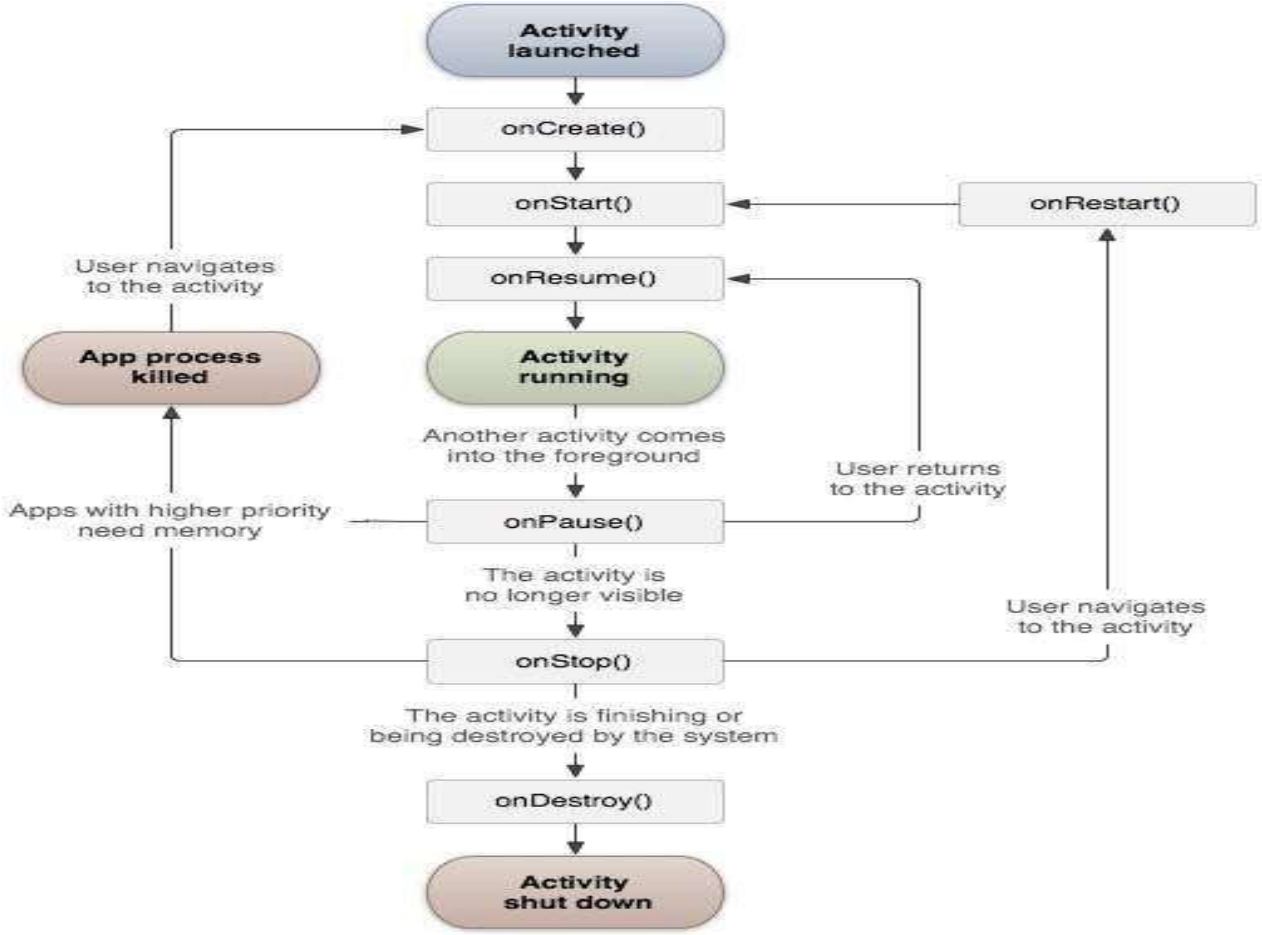
Android Studio has a consolidated environment where we can develop for all Android devices.

Apply changes to the resource code of our running app without restarting the app.

Android Studio provides extensive testing tools and frameworks.



2.2 Introduction to Activities and Activity Lifecycle



An activity represents a single screen with a user interface just like window or frame of Java.Android activity is the subclass of ContextThemeWrapper class.

By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different states.

1. **onCreate :** called when activity is first created.
2. **onStart :** called when activity is becoming visible to the user.
3. **onResume :** called when activity will start interacting with the user.
4. **onPause :** called when activity is not visible to the user.
5. **onStop :** called when activity is no longer visible to the user.
6. **onRestart :** called after your activity is stopped, prior to start.
7. **onDestroy :** called before the activity is destroyed.

2.3 Working with the AndroidManifest.xml

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.
- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

1. ***<manifest>***

manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

2. **<application>**

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

3. ***<activity>***

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

android:label represents a label i.e. displayed on the screen.

android:name represents a name for the activity class. It is required attribute.

4. <intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

4.1 <action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

4.2 <category>

It adds a category name to an intent-filter.

```
<manifest>
  <application>
<activity android:name=".MainActivity" >
    </activity>
  </application>
</manifest>
```

5. <uses-permission>: This element specifies the Android Manifest permissions that are requested for the purpose of security.

2.4 Using the log system

The Android system uses a centralized system for all logs. The application programmer can also write custom log messages. The tooling to develop Android applications allows you to define filters for the log statements you are interested in.

1.2. Create log statements

To write log statements, you use use the android.util.Log class with the following methods:

Log.e(String, String) (error)

Log.w(String, String) (warning)

Log.i(String, String) (information)

Log.d(String, String) (debug)

Log.v(String, String) (verbose)

The priority is one of the following values:

- **V:** Verbose (lowest priority)
- **D:** Debug
- **I:** Info
- **W:** Warning
- **E:** Error
- **A:** Assert

Verbose: Show all log messages (the default).

Debug: Show debug log messages that are useful during development only, as well as the message levels lower in this list.

Info: Show expected log messages for regular usage, as well as the message levels lower in this list.

Warn: Show possible issues that are not yet errors, as well as the message levels lower in this list.

Error: Show issues that have caused errors, as well as the message level lower in this list.

Assert: Show issues that the developer expects should never happen.

2.5 Views and View Groups

View

The View class is the base class or we can say that it is the superclass for all the GUI components in android. For example, the EditText class is used to accept the input from users in android apps, which is a subclass of View, and another example of the TextView class which is used to display text labels in Android apps is also a subclass of View.
Or the other definition,

View refer to the **android.view.View** class, which is the base class of all UI classes. android.view.View class is the root of the UI class hierarchy. So from an object point of view, all UI objects are View objects. Following are some of the common View subclasses that will be used in android applications.

- TextView
- EditText
- ImageView
- RadioButton
- Button
- CheckBox
- DatePicker
- Spinner

These are some of the view subclass available in android.

ViewGroup

The ViewGroup class is a subclass of the View class. And also it will act as a base class for layouts and layouts parameters. The ViewGroup will provide an invisible container to hold other Views or ViewGroups and to define the layout properties. For example, Linear Layout is the ViewGroup that contains UI controls like Button, TextView, etc., and other layouts also. **ViewGroup** Refer to the **android.view.ViewGroup** class, which is the base class of some special UI classes that can contain other View objects as children. Since ViewGroup objects are also View objects, multiple ViewGroup objects and View objects can be organized into an object tree to build a complex UI structure. Following are the commonly used ViewGroup subclasses used in android applications.

- FrameLayout
- WebView
- ListView
- GridView
- LinearLayout
- RelativeLayout
- TableLayout

2.6 Linear Layout, Relative Layout

Linear Layout

LinearLayout is the most basic layout in android studio, that aligns all the children sequentially either in a horizontal manner or a vertical manner by specifying the **android:orientation** attribute.

If one applies **android:orientation=”vertical”** then elements will be arranged one after another in a vertical manner and

If you apply **android:orientation=”horizontal”** then elements will be arranged one after another in a horizontal manner.

```
<? 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="match_parent"
   android:orientation="vertical">

</LinearLayout>
```

android:id

This is the ID which uniquely identifies the layout.

android:Layout_Gravity

This specifies how a **control** should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center

android:gravity

This specifies how a **text** should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.

android:orientation

This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.

android:weightSum

Sum up of child weight

android:LayoutWeight

It is use for Divide the weight sum according to the design.

Relative Layout

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.

android:layout_alignParentBottom.

If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentRight

If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_centerHorizontal

If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".

android:layout_centerVertical

If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".

android:layout_centerInParent

If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".

android:layout_above

Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form

android:layout_above=”@id/text”

android:layout_below

Positions the top edge of this view below the given anchor view ID and must be a reference to another resource

android:layout_below=”@id/text”

android:layout_toLeftOf

Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource

android:layout_ toLeftOf=”@id/text”

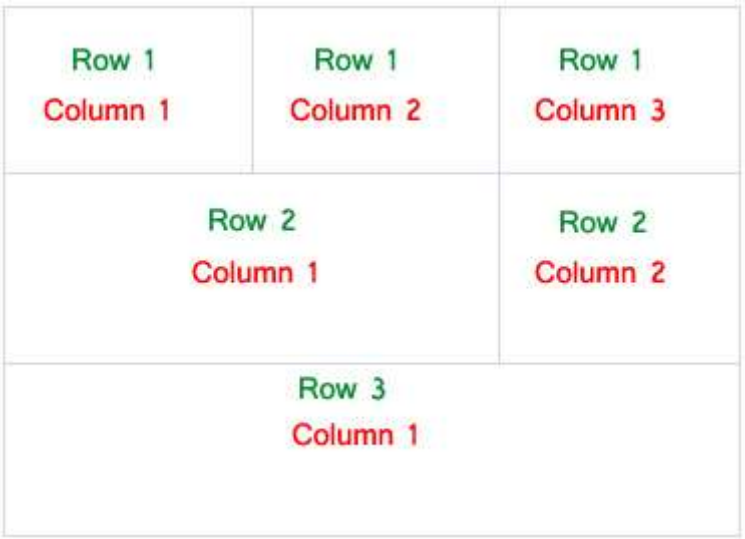
android:layout_toRightOf

Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource

android:layout_ toRightOf=”@id/text”

Table Layout

Android Table Layout going to be arranged groups of views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.



android:id

This is the ID which uniquely identifies the layout.

android:collapseColumns

Collapse columns attribute is used to collapse or invisible the columns of a table layout. These columns are the part of the table information but are invisible.

If the values is 0 then the first column appears collapsed, i.e. it is the part of table but it is invisible.

android:shrinkColumns

Shrink column attribute is used to shrink or reduce the width of the column‘s. We can specify either a single column or a comma delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

If the value is 0 then the first column’s width shrinks or reduces by word wrapping its content.

If the value is 0, 1 then both first and second columns are shrinks or reduced by word wrapping its content.

If the value is ‘*’ then the content of all columns is word wrapped to shrink their widths.

android:stretchColumns

Stretch column attribute is used in Table Layout to change the default width of a column which is set equal to the width of the widest column but we can also

stretch the columns to take up available free space by using this attribute. The value that assigned to this attribute can be a single column number or a comma delimited list of column numbers (1, 2, and 3…n).

If the value is 1 then the second column is stretched to take up any available space in the row, because of the column numbers are started from 0.

If the value is 0, 1 then both the first and second columns of table are stretched to take up the available space in the row.

Constraint Layout

Advantages of using ConstraintLayout in Android

- ConstraintLayout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.
- It helps to improve the UI performance over other layouts.
- With the help of ConstraintLayout, we can control the group of widgets through a single line of code.
- With the help of ConstraintLayout, we can easily add animations to the UI components which we used in our app.

Disadvantages of using ConstraintLayout

- When we use the Constraint Layout in our app, the XML code generated becomes a bit difficult to understand.
- In most of the cases, the result obtain will not be the same as we got to see in the design editor.
- Sometimes we have to create a separate layout file for handling the UI for the landscape mode.

android:id

This is used to give a unique id to the layout.

app:layout_constraintBottom_toBottomOf

This is used to constrain the view with respect to the bottom position.

app:layout_constraintLeft_toLeftOf

This attribute is used to constrain the view with respect to the left position.

app:layout_constraintRight_toRightOf

This attribute is used to constrain the view with respect to the right position.

app:layout_constraintTop_toTopOf

This attribute is used to constrain the view with respect to the top position.

Frame Layout

Android Framelayout is a ViewGroup subclass that is used to specify the position of multiple views placed on top of each other to represent a single view screen.

Syntax:

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

   // Add items or widgets here
</FrameLayout>
```

Scroll Layout

ScrollView is a kind of layout that is useful to add vertical or horizontal scroll bars to the content which is larger than the actual size of layouts such as linearlayout, relativelayout, framelayout.

The ScrollView will enable a scroll to the content which is exceeding the screen layout and allow users to see the complete content by scrolling.

ScrollView supports only vertical scrolling.

1. Scroll View

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


</ScrollView>
```

2. HorizontalScrollView

Horizontal ScrollView is a FrameLayout, used to provide the child View element horizontal scrolling property.

The ChildView in itself can be a layout manager like the linear layout. The TextView class takes care of its own scrolling, But it can be placed inside a HorizontalScrollView to create more complex UI designs.

Syntax:

```
<HorizontalScrollView
      android:id="@+id/horizontalScrollView"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
     android:foregroundGravity="center_vertical">


</HorizontalScrollView>
```

Thank You