CHAPTER – 3

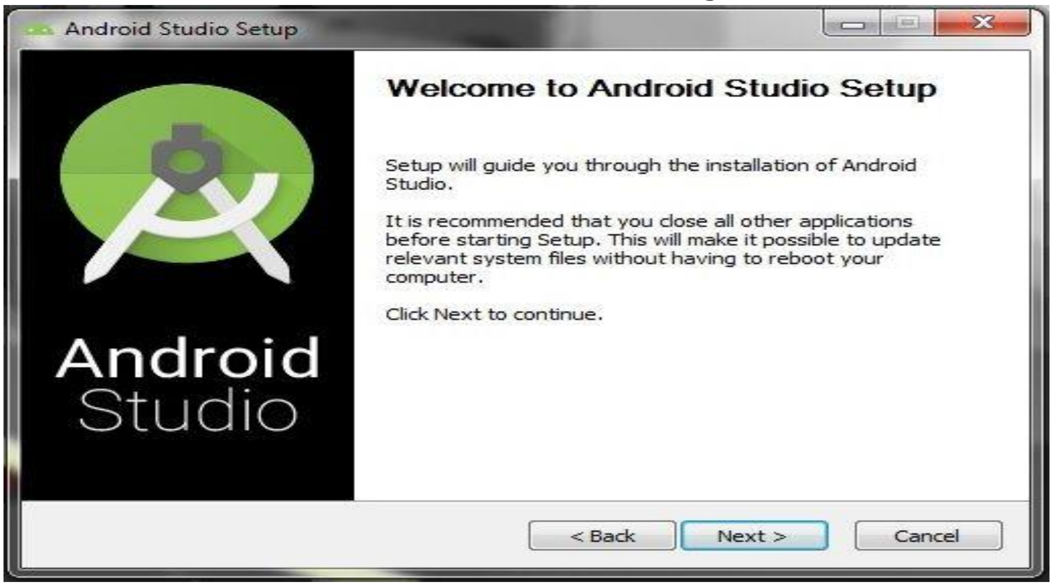
ANDROID CLASSES AND BASICS

ANDROID FUNDAMENTALS:

Creating an Android App:

Step 1: Install Android Studio:

1. Go to *http://developer.android.com/sdk/index.html* to download Android Studio.
2. Use the installer to install Android Studio following its instructions.



Welcome to Android Studio Setup

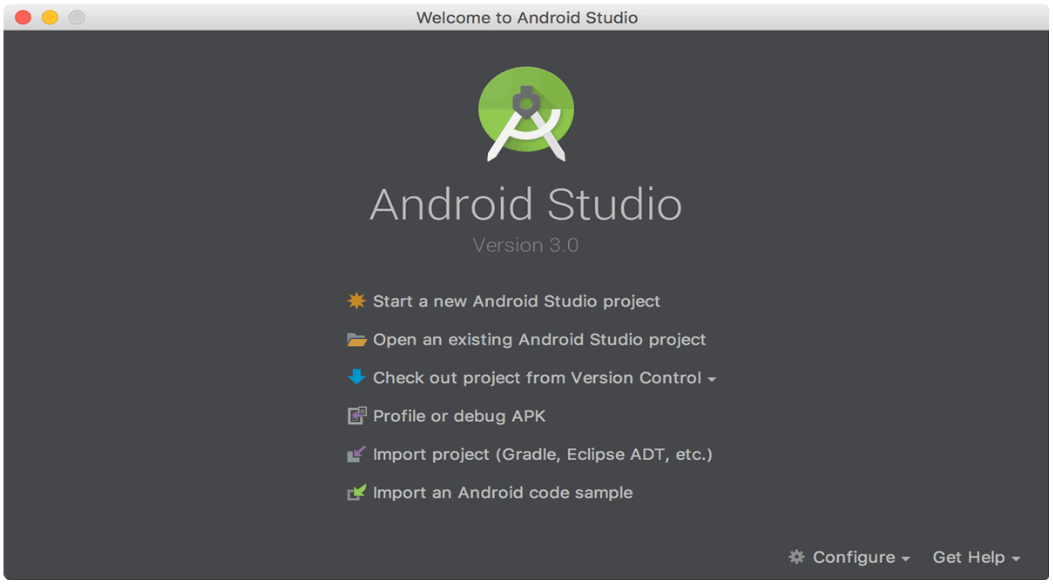
Setup will guide you through the installation of Android Studio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Step 2: Open a New Project:

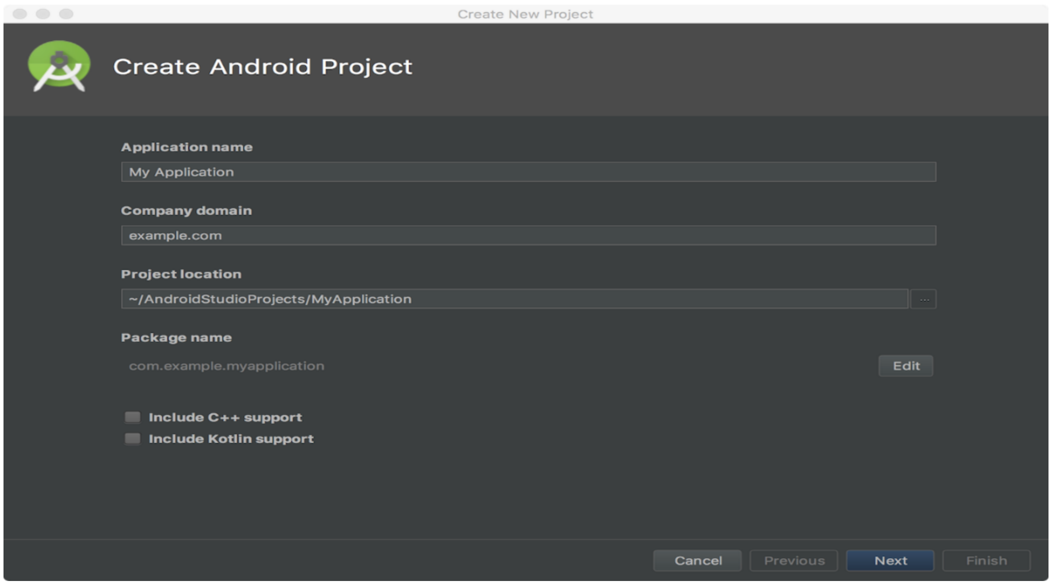
1. Open Android Studio.
2. Under the "Quick Start" menu, select "Start a new Android Studio project."



Welcome to Android Studio
Android Studio
Version 3.0
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

Step 3: Configure the Project:

1. Enter the values for the project then click **Next**.


Create New Project

Create Android Project

Application name
My Application

Company domain
example.com

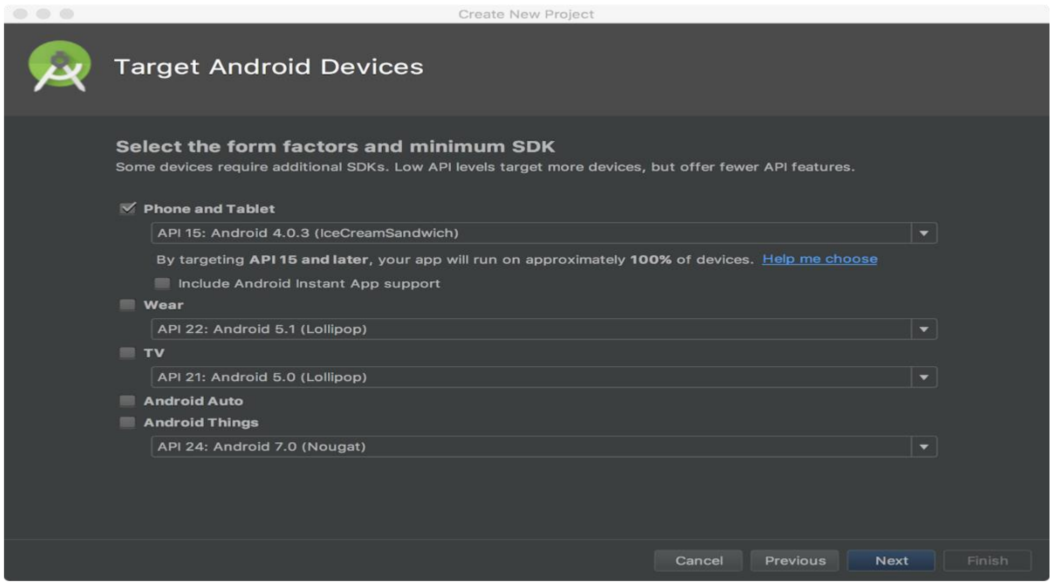
Project location
~/AndroidStudioProjects/MyApplication

Package name
com.example.myapplication

☐ Include C++ support
☐ Include Kotlin support

Step 4: Select Form Factors and API Level:

1. This opens a new window that shows the distribution of devices running each version of Android. Click on an API level to see a summary of top features introduced in that version. To return to the wizard, click **OK**.


Create New Project

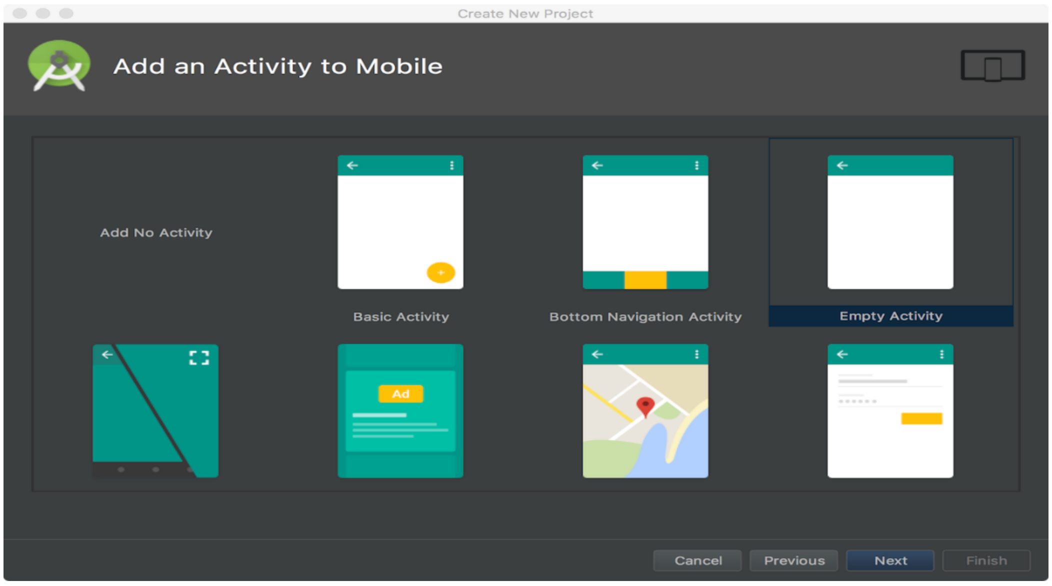
Target Android Devices

Select the form factors and minimum SDK
Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

☑ Phone and Tablet
API 15: Android 4.0.3 (IceCreamSandwich)
By targeting **API 15 and later**, your app will run on approximately **100%** of devices. Help me choose
☐ Include Android Instant App support
☐ Wear
API 22: Android 5.1 (Lollipop)
☐ TV
API 21: Android 5.0 (Lollipop)
☐ Android Auto
☐ Android Things
API 24: Android 7.0 (Nougat)

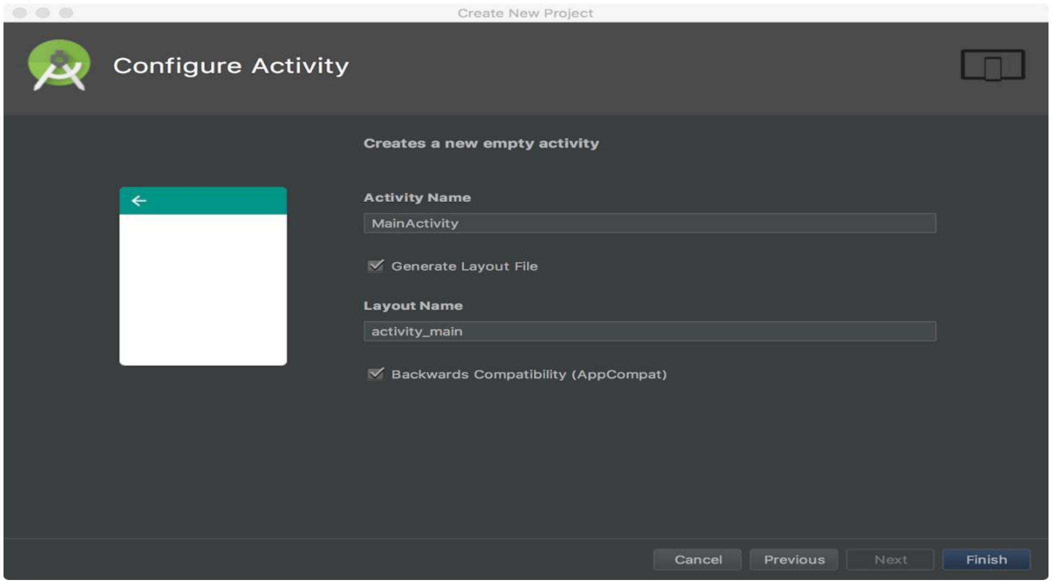
Step 5: Add An Activity:

1. Choose an activity type then click **Next**.



Step 6: Configure Activity:

1. Enter the activity name, the layout name, and the activity title. Then click **Finish**. Android Studio now sets up the project and opens the IDE.



Creates a new empty activity

Activity Name
MainActivity

☑ Generate Layout File

Layout Name
activity_main

☑ Backwards Compatibility (AppCompat)

Android Manifests File:

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

Double clicking on the AndroidManifest.xml file in the Android Studio project will open the Manifest editor. The manifest editor is the normal way of creating and modifying the manifest file (defining the app to the system).

Function of Manifests File:

- Names the package which becomes unique app ID

- ❖ Describes the components such as Activities, Services, Broadcast Receivers, Content Providers
- ❖ Publishes component capabilities example: which intent messages they can handle
- ❖ Determines which processes will host components
- ❖ Declares permissions that are needed to interact with components of other processes
- ❖ Declares permissions that other process components must have to interact with this application
- ❖ Lists libraries that the app will be linked against
- ❖ Declares minimum level Android API that is required

Example:

```
<? xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloandroid"
   android:versionCode="1"
   android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".HelloAndroid"
         android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Explanation of Manifests Tags:

1. <manifest>

The manifest tag has the following attributes:

- ❖ **xmlns**: The name of the namespace (android) and where the DTD for the xml parser is located
- ❖ **package**: The name of the java package for this application (must have at least two levels)
- ❖ **android:version:** The version code for this version of the app
- ❖ **android:versionName:** The version name (for publishing)

2. <activity>

- ❖ It is a child tag of <manifest>
- ❖ We need one <activity> tag for each activity of the application
- ❖ Its attributes are:

  - **android:name:** The name of the activity, this will be used as the name of the Java file and the resulting class
  - **android:label:** String that we will be able to programmatically retrieve the activity name at run time.

3. **<intent-filter>**

- It is child tag of <activity>
- An intent is a message sent from one program to another (message dispatcher) to tell the system what to do next. Typically an intent consists of two parts; an action and the data that that action is supposed to use to do it.
- When we select an icon on the main page the intent is to run the app associated with that icon.
- This tag is used to construct an android.content.IntentFilter object to handle a particular android.content.Intent

4. **<action>**

- It is child of <intent-filter>
- The action we want done: Predefined actions of the intent class of android.content

5. **<category>**

- It is child of <intent-filter>
- It is additional attributes that can be supplied
- LAUNCHER – indicates that it should appear in the launcher as a top level application

THE ACTIVITY CLASS:

Introduction:

An activity is a window that contains the user interface of the application. It is an application component that provides a screen with which users can interact in order to do something. Typically, applications have one or more activities; and the main purpose of an activity is to interact with the user.

Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. For a .java class to qualify as an activity it should extend the Activity class.

States of an Activity:

1. **Foreground State:**

If an activity is in the foreground of the screen, it is active or running.

2. **Paused State:**

If an activity has lost focus but is still visible, it is in paused state. A paused state is completely alive, but can be killed by the system in extreme low memory situations

3. **Background State:**

If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, however it is no longer visible to the user so its window is hidden and it will often killed by the system when the memory is needed elsewhere.

4. **Destroyed State:**

If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

Activity Life Cycle:

The steps through which an application goes from starting to finishing is known as activity life cycle. Activity life cycle is slightly different from normal Java life cycle due to following reasons:

- Difference in the way Android application are defined
- limited resources of the Android hardware platform
- Each application runs in its own process.
- Each activity of an app is run in the apps process
- Processes are started and stopped as needed to run an apps components.
- Processes may be killed to reclaim needed resources.
- Killed apps may be restored to their last state when requested by the user

Most management of the life cycle is done automatically by the system via the activity stack. The activity class has the following method callbacks to help us to manage the app:

- **onCreate():**

This is the first callback and called when the activity is first created.

- **onStart():**

This callback is called when the activity becomes visible to the user.

- **onResume():**

This is called when the user starts interacting with the application.

- **onPause():**

The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.

- **onStop():**

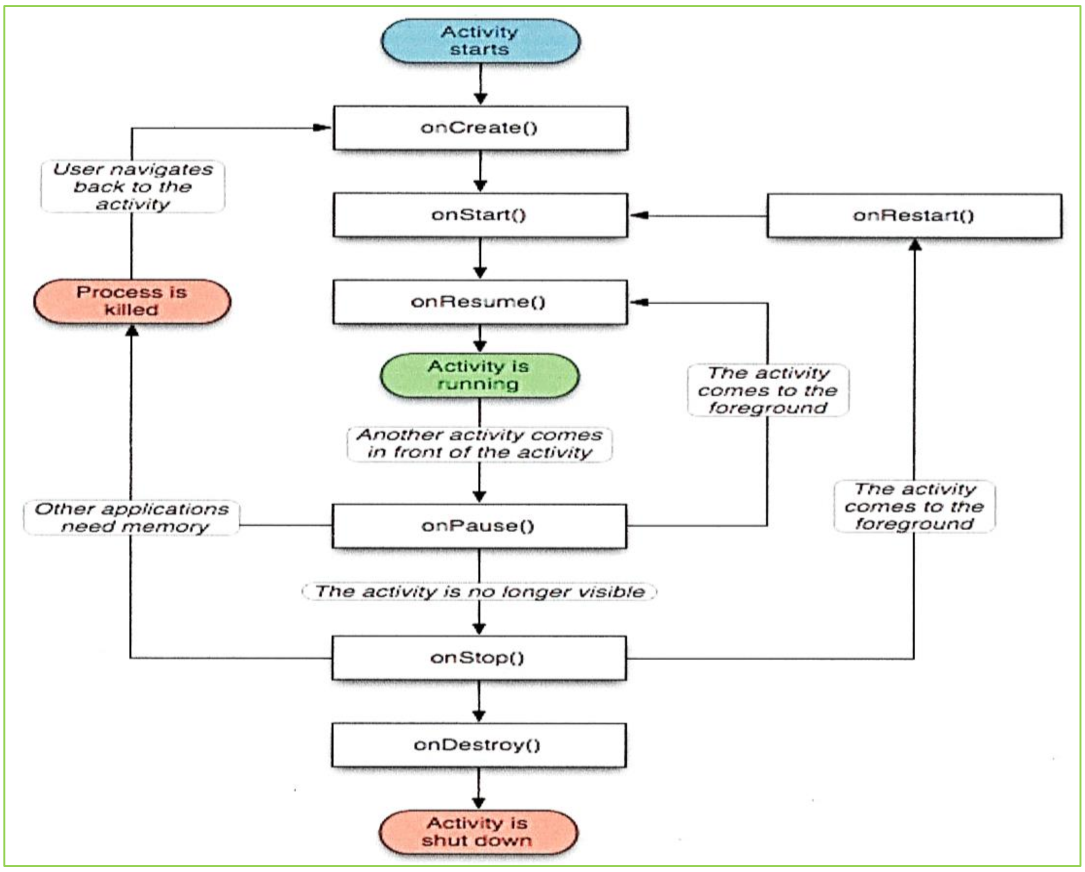
This callback is called when the activity is no longer visible.

- **onDestroy():**

This callback is called before the activity is destroyed by the system.

- **onRestart():**

This callback is called when the activity restarts after stopping it.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Extending the Activity Class:

To create an Android activity, a class needs to be written extending the Activity class and by writing the callback methods corresponding to create, stop, resume or destroy operations. These methods (listed above) will be called based on the various stages of the activity among the life cycle process.

Example Activity Which Has Only Two Callback Methods,

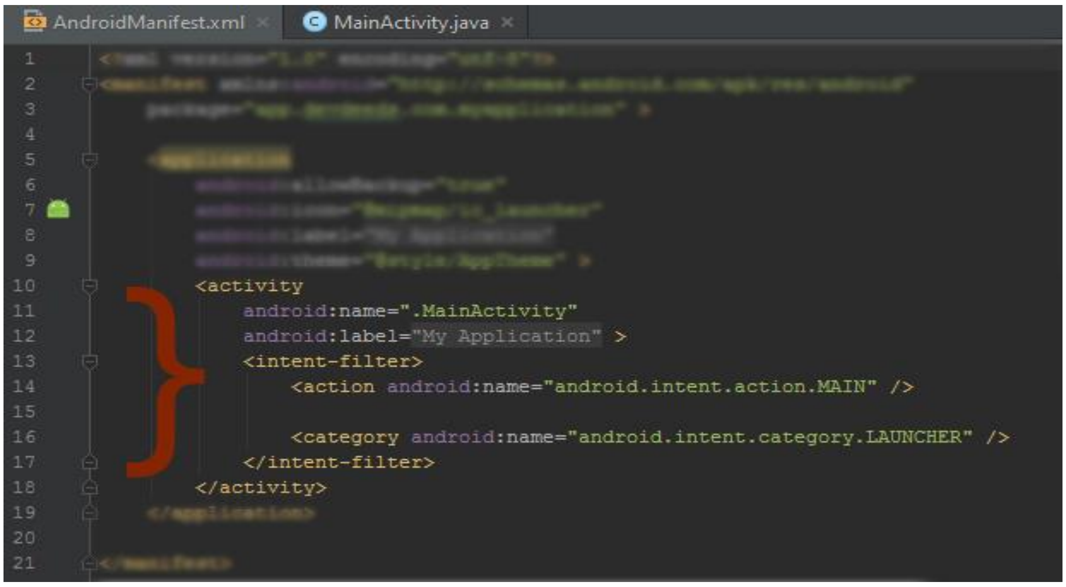
```
public class MainActivity extends Activity {
@Override
   public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      objStatus.updateStatusList("MainActivity","Created");
   }
 @Override
   protected void onPause() {
            super.onPause();
            objStatus.updateStatusList("MainActivity","Paused");
   }}
```

Creating Default Activity:

Every app has a landing page we often call it homepage/ default page where we show splash animation or intro page of our app. If our app has more than one activity then we have to know how to set default activity.
In Android, we can configure the starting activity (default activity) of our application via following “intent-filter” in “AndroidManifest.xml“.

Step 1:

First we should define our activity class in Android manifest file, which is at the root of our project directory.



```
[illegible]
        <activity
            android:name=".MainActivity"
            android:label="My Application" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
[illegible]
```

Step 2: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="app.devdeeds.com.myapplication" >
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
```

```
</manifest>
```

Please Note:

If no activity defined in AndroidManifest.xml has such a <intent-filter> tag then we will get error "Default Activity Not Found". This means application should at least have one default activity (home page).

Creating Splash and Login Activities:

Splash Screen:

Splash Screen is often referred to a welcome screen or user's first experience of an application. There are many ways for creating a Splash Screen for an Android Application. It is not recommended to put a splash screen in an app unless absolutely necessary such as in a game for pre-loading resources.

Step 1: Create SplashScreenActivity:

The SplashScreenActivity must be Launcher Activity.

```
public class SplashScreenActivity extends AppCompatActivity {
@Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_splash_screen);
      Intent intent = new Intent(getApplicationContext(), MainActivity.class);
      startActivity(intent);
      finish();
  }
}
```

Step 2: Create Background for Splash Screen Splash_Screen_Background.Xml

In res/drawable directory create splash_screen_background.xml file with the following code. Set a Background Image- It can be any image for example Product Logo.

```
<? xml version="1.0" encoding="utf-8"?>
  <layer-list
    xmlns:android=http://schemas.android.com/apk/res/android
    android:layout_height="match_parent" android:layout_width="match_parent">
    <item android:drawable="@color/colorPrimary"/>
      <!-- Set the Background Image-->
      <item android:gravity="center" android:width="500dp" android:height="700dp">
      <bitmap android:gravity="fill_horizontal|fill_vertical"
      android:src="@drawable/splash_screen"/>
    </item>
  </layer-list>
```

Step 3: Create Style for Splash Screen

In *res/values/***styles.xml**, define **SplashScreenTheme**

```
<resources>
  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
  </style>

  <!--Style for Splash Screen-->
  <style
    name="SplashScreenTheme"
    parent="Theme.AppCompat.Light.DarkActionBar">
    <item name ="android:windowBackground">
    @drawable/splash_screen_background</item>
  </style>
</resources>
```

Step 4: Set style for SplashScreenActivity in AndroidManifest.xml

Set **theme** for SplashScreenActivity

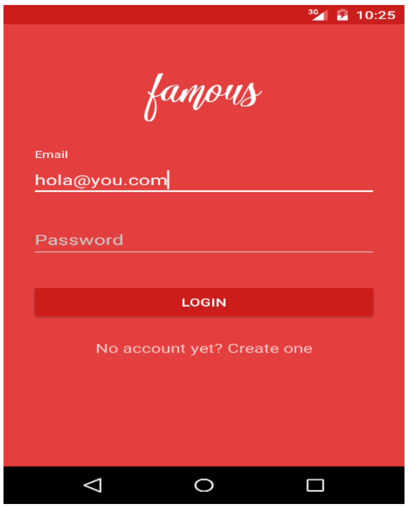
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.anuragdhunna.www.splashscreen">

  <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=".SplashScreenActivity"
       android:theme="@style/SplashScreenTheme">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".MainActivity"/>
  </application>
```

```
</manifest>
```

Login Activities:

Everyone loves a beautiful login screen, and since it's usually the very first impression people have about our app, it's super important to get it right.



First we have to define two TextView asking username and password of the user. The password TextView must have **inputType** set to password. Its syntax is given below:

```
<EditText
  android:id = "@+id/editText2"
  android:layout_width = "wrap_content"
  android:layout_height = "wrap_content"
  android:inputType = "textPassword" />

<EditText
  android:id = "@+id/editText1"
  android:layout_width = "wrap_content"
  android:layout_height = "wrap_content"/>
```

Define a button with login text and set its **onClick** Property. After that define the function mentioned in the onClick property in the java file.

```
<Button
  android:id = "@+id/button1"
  android:layout_width = "wrap_content"
  android:layout_height = "wrap_content"
  android:onClick = "login"
  android:text = "@string/Login"
/>
```

In the java file, inside the method of onClick get the username and passwords text using **getText()** and **toString()** method and match it with the text using **equals()** function.

```
EditText username = (EditText)findViewById(R.id.editText1);
EditText password = (EditText)findViewById(R.id.editText2);
public void login(View view){
if(username.getText().toString().equals("admin")
&& password.getText().toString().equals("admin")){
  //correcct password
  }
else{
  //wrong password
}
```

The last thing we need to do is to provide a security mechanism, so that unwanted attempts should be avoided. For this initialize a variable and on each false attempt, decrement it. And when it reaches to 0, disable the login button.

```
int counter = 3;
counter--;
if(counter==0){
  //disble the button, close the application etc.
}
```

THE INTENT CLASS:

Introduction:

Android Intent 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 **dictionary meaning** of intent is *intention or purpose*. So, it can be described as the intention to do action. The LabeledIntent is the subclass of android.content.Intent class.

An Intent is an object that provides runtime binding between separate components, such as two activities. The Intent represents an app’s "intent to do something."

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- 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. For example, we may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.javatpoint.com"));
startActivity(intent);
```

2. **Explicit Intent**

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

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

Creating Intent:

Start Activity:

To start Activities, we must make two steps as follow:

Step 1: Declare Activity in Manifest.

```
<activity
   android:name=".SecondActivity"
   android:label="@string/title_activity_second" >
     <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
</activity>
```

Step 2: Create Intent to call the Activity.

```
Intent secondActivity = new Intent(this, SecondActivity.class);
startActivity(secondActivity);
```

Start Service:

To start Services, we must make two steps as follow:

Step 1: Declare Service in Manifest.

```
<service android:name="MyService" >
</service>
```

Step 2: Create Intent to call the Service.

```
Intent myServiceIntent = new Intent(this, MyService.class);
```

```
startService(myServiceIntent);
```

Send BroadcastReceiver:

To start BroadcastReceiver, we must make two steps as follow:

Step 1: Declare BroadcastReceiver in Manifest.

```
<receiver android:name="MyBroadcastReceiver" >
    <intent-filter>
          <action android:name="com.example.intenttutorial.myrecevier" />
    </intent-filter>
</receiver>
```

Step 2: Create Intent to send the BroadcastReceiver.

```
Intent broadcastIntent = new Intent(broadcastFilter);
broadcastIntent.putExtra("9Android.net","MyBroadcastReciver is called!");
sendBroadcast(broadcastIntent);
```

Switching between Activities using Intent:

In Android user interface is displayed through an activity. In Android app development we might face situations where we need to switch between one Activity (Screen/View) to another.

Let's assume that our new Activity class name is **SecondScreen.java**

Step 1: Opening New Activity:

To open new activity following startActivity() or startActivityForResult() method will be used.

```
Intent i = new Intent(getApplicationContext(), SecondScreen.class);
StartActivity(i);
```

Step 2: Sending Parameters to New Activity:

To send parameter to newly created activity putExtra() methos will be used.

```
i.putExtra("key", "value");
// Example of sending email to next screen as
// Key = 'email'
// value = 'myemail@gmail.com'
i.putExtra("email", "myemail@gmail.com");
```

Step 3: Receiving Parameters on New Activity:

To receive parameters on newly created activity getStringExtra() method will be used.

```
Intent i = getIntent();
i.getStringExtra("key");
// Example of receiving parameter having key value as 'email'
// and storing the value in a variable named myemail
String myemail = i.getStringExtra("email");
```

Step 4: Opening New Activity and Expecting Result:

In some situations we might expect some data back from newly created activity. In that situations startActivityForResult() method is useful. And once new activity is closed we should use onActivityResult() method to read the returned result.

```
Intent i = new Intent(getApplicationContext(), SecondScreen.class);
startActivityForResult(i, 100); // 100 is some code to identify the returning result
// Function to read the result from newly created activity
@Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);
      if(resultCode == 100){
         // Storing result in a variable called myvar
         // get("website") 'website' is the key value result data
         String mywebsite = data.getExtras().get("result");
      }
   }
```

Step 5: Sending Result Back to Old Activity:

Sending result back to old activity when StartActivityForResult() is used.

```
Intent i = new Intent();
// Sending param key as 'website' and value as 'androidhive.info'
i.putExtra("website", "AndroidHive.info");
// Setting resultCode to 100 to identify on old activity
setResult(100,in);
```

Step 6: Closing Activity:

To close activity call finish() method

```
finish();
```

Step 7: Add Entry In Androidmanifest.Xml

To run our application we should enter our new activity in AndroidManifest.xml file. Add new activity between <application> tags

```
<activity android:name=".NewActivityClassName"></activity>
```

Permissions:

Every application developed for the Android platform has an AndroidManifest.xml file associated with it. When we build the application for the Android platform, the manifest file contains all the details that the Android platform needs to execute the application. It also provides information about the application components such as services and activities.

We can modify the Android manifest from the Android properties tab. The location of the AndroidManifest.xml file is as follows:

<WorkspaceName>/temp/<AppName>/build/luaandroid/dist/<AppName>

Allow APP Permissions in Android Manifest:

When we build the application, the AndroidManifest.xml file is generated either with the default permissions or with the permissions that we have set. The following permissions are set to *true* by default:

- ACCESS_NETWORK_STATE
- INTERNET
- READ_PHONE_STATE.

The following permissions are set to *false* by default:

- CAMERA
- ACCESS_FINE_LOCATION
- SET_ALARM
- MANAGE_ACCOUNTS, etc.

Set Android Manifest Permissions:

We can add user permissions to the Android project. Also we can add a Detail Activity class to the AndroidManifest.xml file. This declaration launches a customer detail screen where we can make changes when we test the application.

1. If needed, open the Android manifest file.
2. Select the AndroidManifest.xml tab.
3. Declare activity and add user permission in the AndroidManifest.xml file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="smp.tutorial.android"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET">
  </uses-permission>
  <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" >
   </uses-permission>
  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

```
        <activity
          android:name=".SplashScreenActivity"
          android:label="@string/title_activity_splash_screen" >
          <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
        </activity>
      </application>
    </manifest>
```

4. Select File > Save.

THE FRAGMENT CLASS AND ITS USAGE:

Introduction:

A fragment is a piece of an application's user interface or behavior that can be placed in an activity which enable more modular activity design. A fragment is a kind of sub-activity.

A fragment has its own layout and its own behavior with its own lifecycle callbacks. We can add or remove fragments in an activity while the activity is running. We can combine multiple fragments in a single activity to build multi-pane UI. A fragment can be used in multiple activities. Fragment lifecycle is closely related to the lifecycle of its host activity which means when the activity is paused, all the fragments available in the activity will stopped.

A fragment can implement a behavior that has no user interface components. Fragments were added to the android API in Honeycomb version of Android which is API version 11.

Creation of the Fragment:

We can create fragments by extending Fragment class. We insert a fragment into our activity layout by declaring the fragment in the activity's layout file, as a <fragment> element.

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. We were not able to divide device screen and control different parts separately. With the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, event and complete lifecycle.

Fragment Life Cycle:

Most applications should implement at least three methods (i.e onCreate(), onCreateView() & onPause()) for every fragment.

Phase I: When a fragment gets created, it goes through the following states:

- **onAttach():**

Called when the fragment has been associated with the activity (the Activity is passed in here).

- **onCreate():**

Fragment view is created, this method will return a view object which is generally the fragment's root layout view object.

- **onCreateView():**

The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for our fragment, we must return a View from this method that is the root of our fragment's layout. We can return null if the fragment does not provide a UI.

- **onActivityCreated():**

Activity is created. The activity's **OnCreate** method has been finished. If we need to do some work after activity create in our fragment, we can add those code here.

Phase II: When the fragment becomes visible,it goes through these states:

- **onStart():**

This method is invoked when the fragment is started until it is visible and ready to interact with user.

- **onResume():**

This method is called just after **OnStart** or when the fragment is popup from the backstack, it means the fragment is running again.

Phase III: When the fragment goes into the background mode, it goes through the following states:

- **onPause():**

This method is called when the fragment will be left. Such as when remove current fragment or replace current fragment with other fragment.

- **onStop():**

This fragment is going to be stopped. When remove current fragment or replace current fragment with other fragment.

Phase: IV: When the fragment is destroyed, it goes through the following states:

- **onDestroyView()**

This method is called before OnDestroy, it is used when we need to clean fragment view objects before destroy.

- **onDestroyed():**

This method is called when android OS destroy current fragment, we can do some resource release or state info store action in this method.

- **onDetach():**

When the fragment is destroyed, it will be detached from the activity, if we need to do some work in this state, we can put our code in this method.

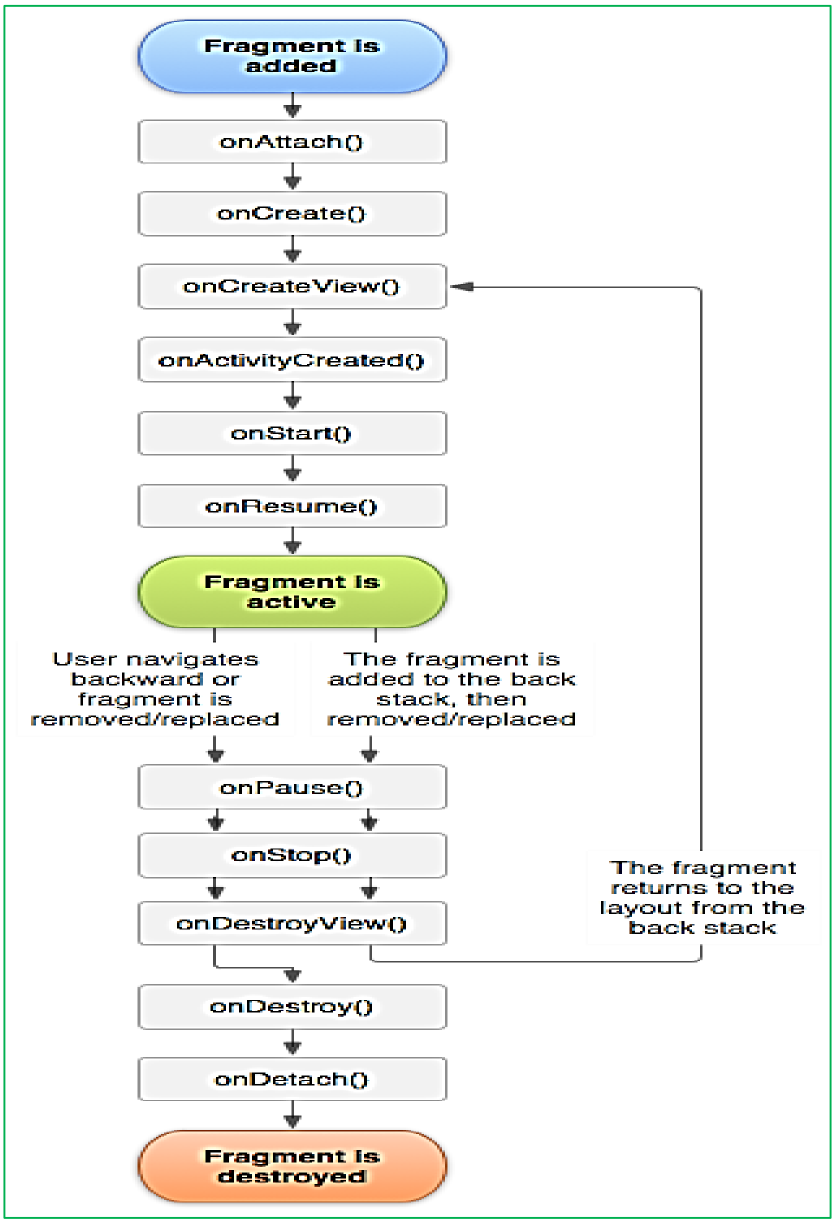


Fig: Life Cycle of Fragment

Activity v/s Fragment:

Activity	Fragment
An activity is a window with which the user interacts with.	A fragment is a part of an activity, which contributes its own UI to that Activity.

An activity may contain zero or multiple number of fragments.	A fragment can be reused in multiple activities, so it acts like a reusable component in activities.
An activity can exist without any fragment in it.	A fragment has to live inside the activity. It should always be the part of an activity.
Activity is needed to be declared in “AndroidManifest.xml”	Fragment is not needed to be declared in “AndroidManifest.xml”
We cannot have nested activities.	We can have nested fragments.