MOBILE COMPUTING

Lecture Four

Android studio components

1. Activities

They dictate the UI and handle the user interaction to the smart phone screen.

2. Services

They handle background processing associated with an application.

3. Broadcast Receivers

They handle communication between Android OS and applications.

4. Layout

Represent what is seen on the scree

Structure of Android Studio project

app > manifests > AndroidManifest.xml

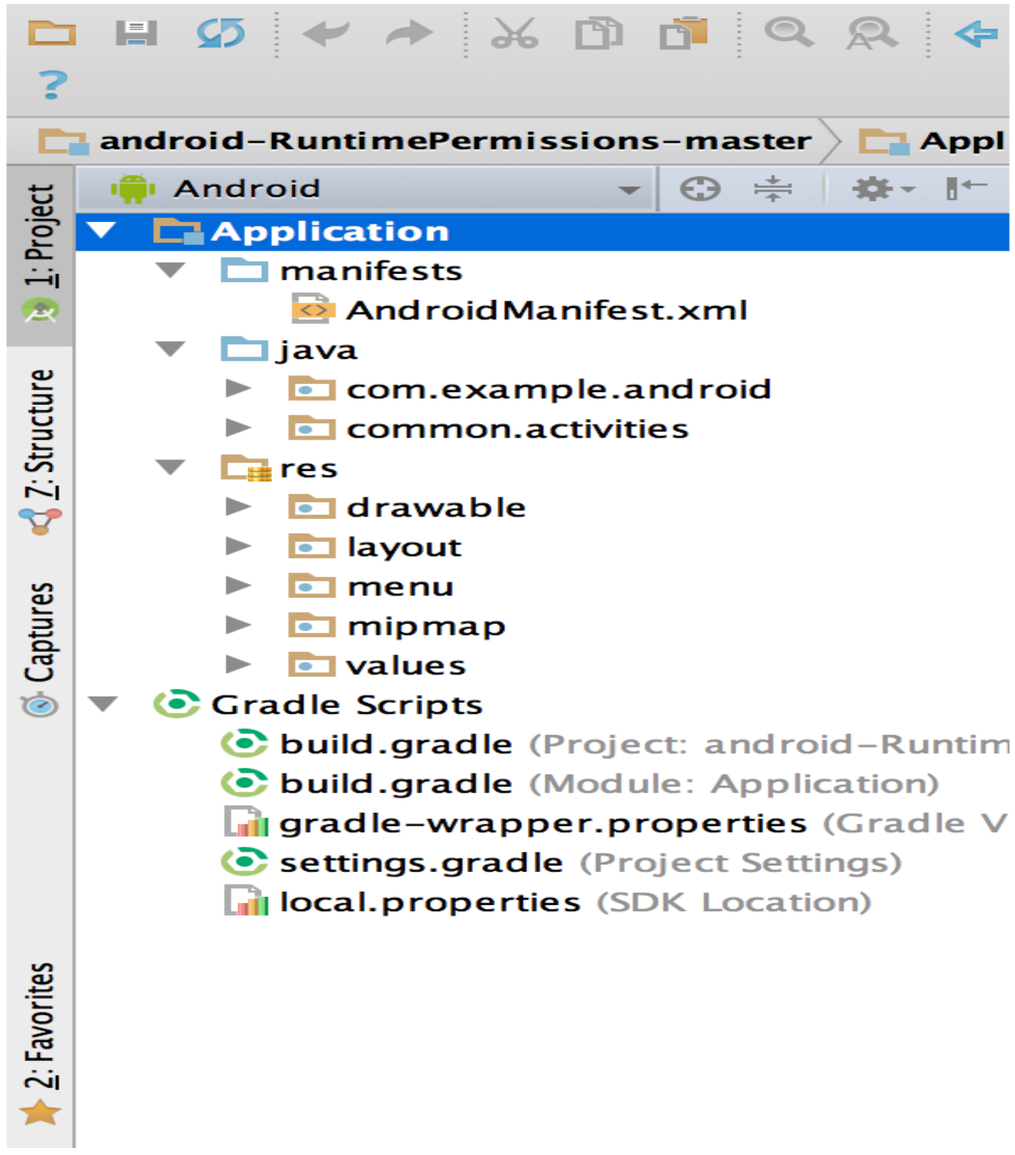
The manifest file defines each component included in the Android project (such as activities).

app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this Activity and loads its layout.

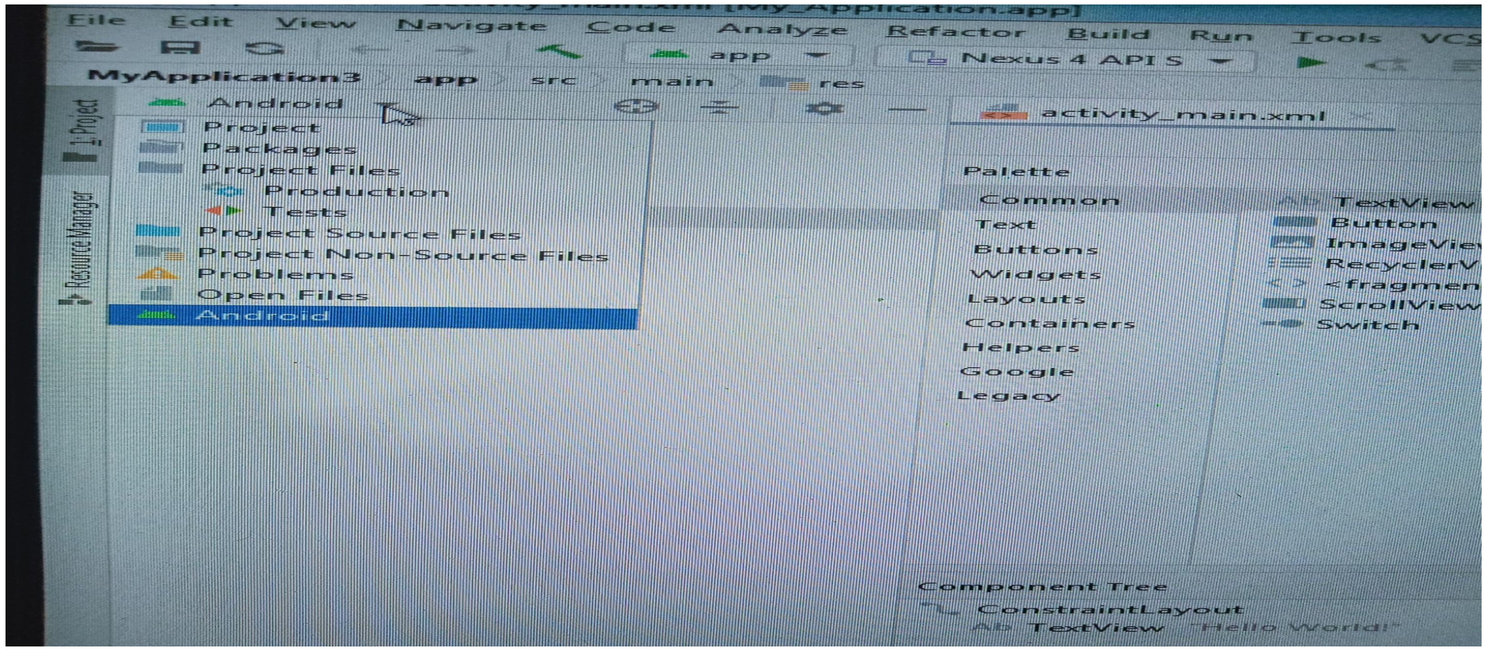
app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI).



Project Navigation

On the left side expand the menu to chose Android

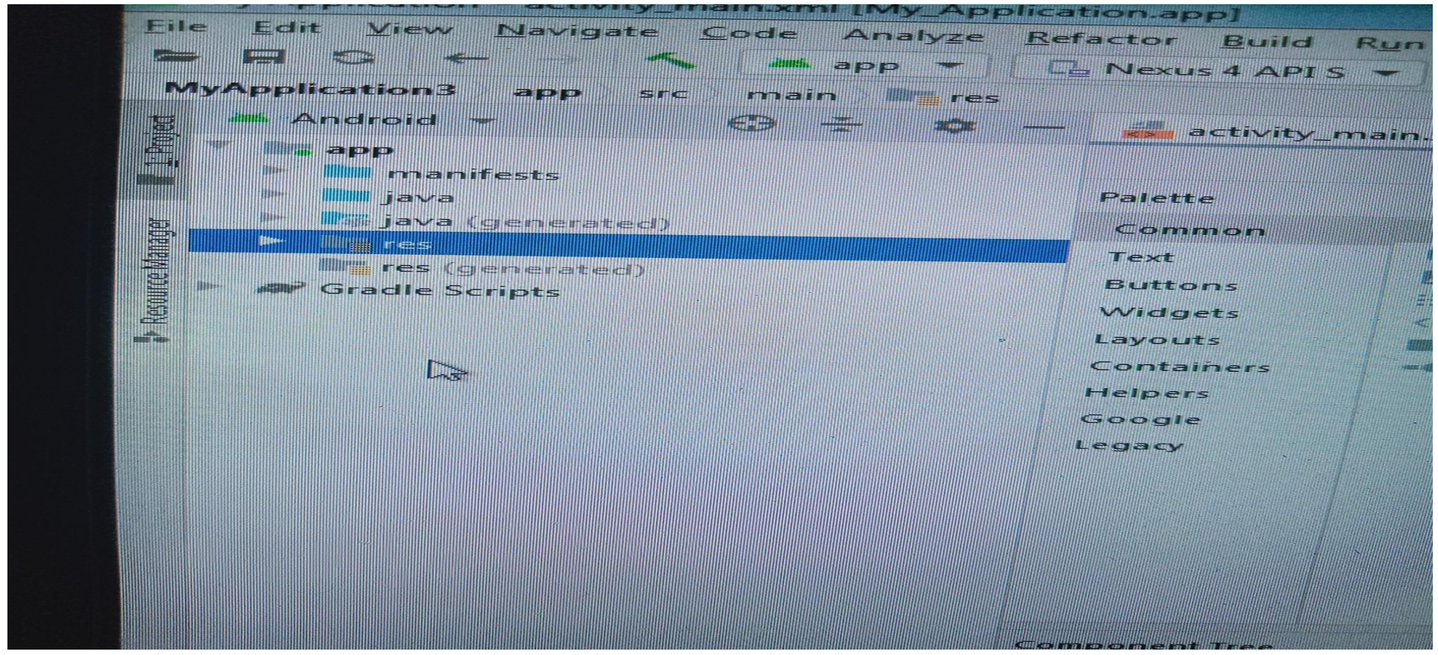


Three main parts of your project appear under app directory

manifests > AndroidManifest.xml

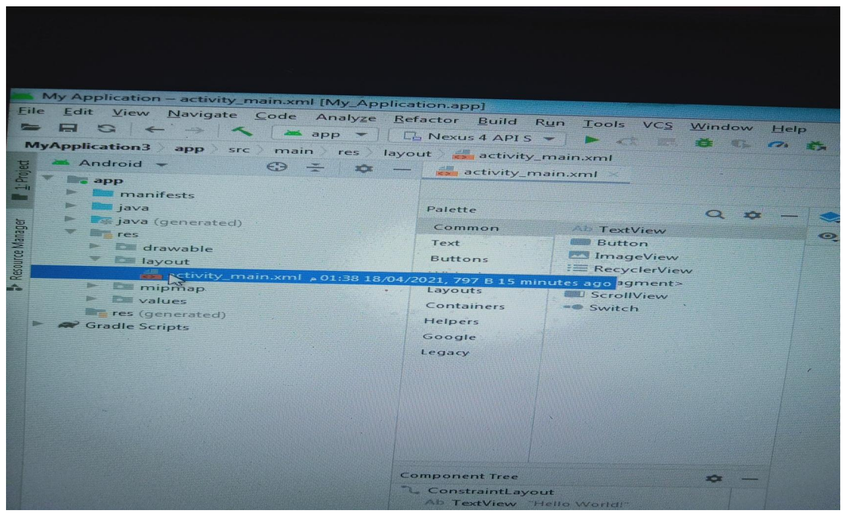
app > java > com.example.myfirstapp > MainActivity

app > res > layout > activity_main.xml

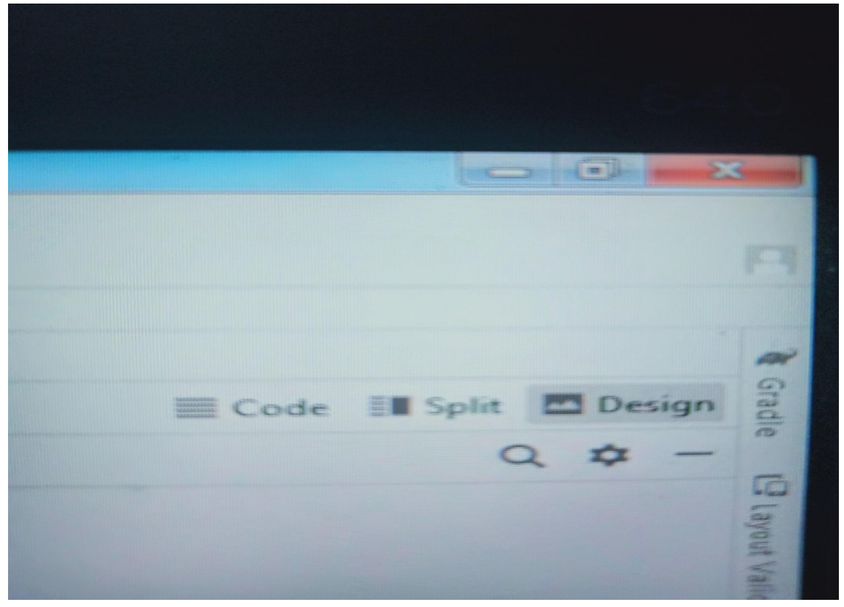


Activity layout and user interface(6)

Expand (res) then layout then click on **activity_main.xml**



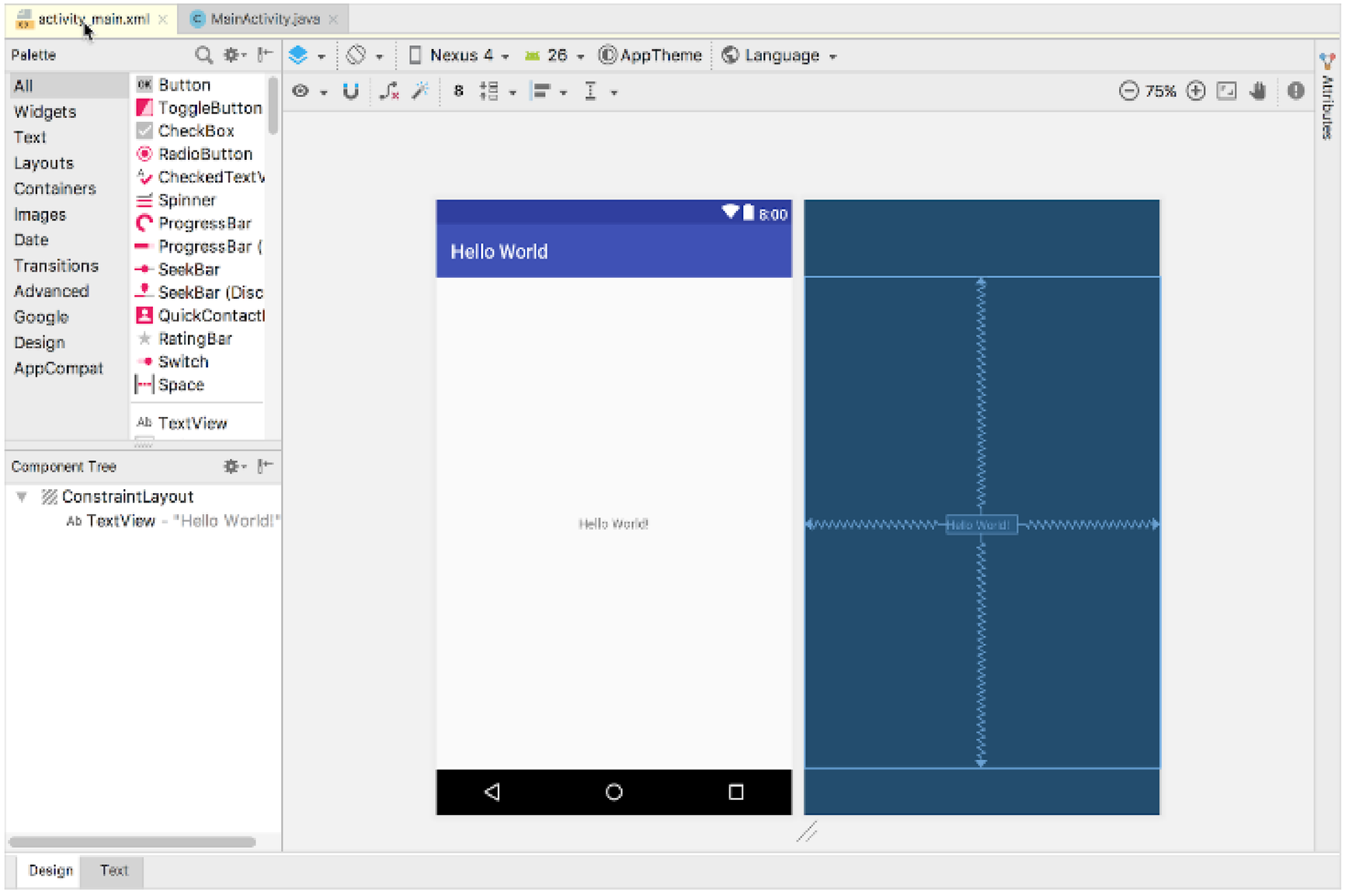
You can see the layout as XML code or as a graphical view,
3 options
Code: see xml code only
Design: see graphical interface only
Split: see both



Main Activity XML file

layout editor.

- On the right the scree and on the left set of tools that allows you to drag and drop “textbox” ,”image”,….



View class

The View class represents the basic building block for all UI components

A View has a location, expressed as a pair of left and top coordinates, and two dimensions, expressed as a width and a height.

TextView: for displaying text

EditText: to enable the user to enter and edit text

Button, RadioButton, CheckBox: to provide interactive behavior

ScrollView: to display scrollable items

ImageView: for displaying images

Example

```
<TextView
    android:layout_width="244dp"
    android:layout_height="91dp"
    android:text="Hello SIM"
    android:textSize="30sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.758"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />


<ImageView
   android:id="@+id/imageView"
   android:layout_width="274dp"
   android:layout_height="142dp"
   tools:layout_editor_absoluteX="68dp"
   tools:layout_editor_absoluteY="488dp"
   tools:srcCompat="@tools:sample/avatars" />
```

MainActivity

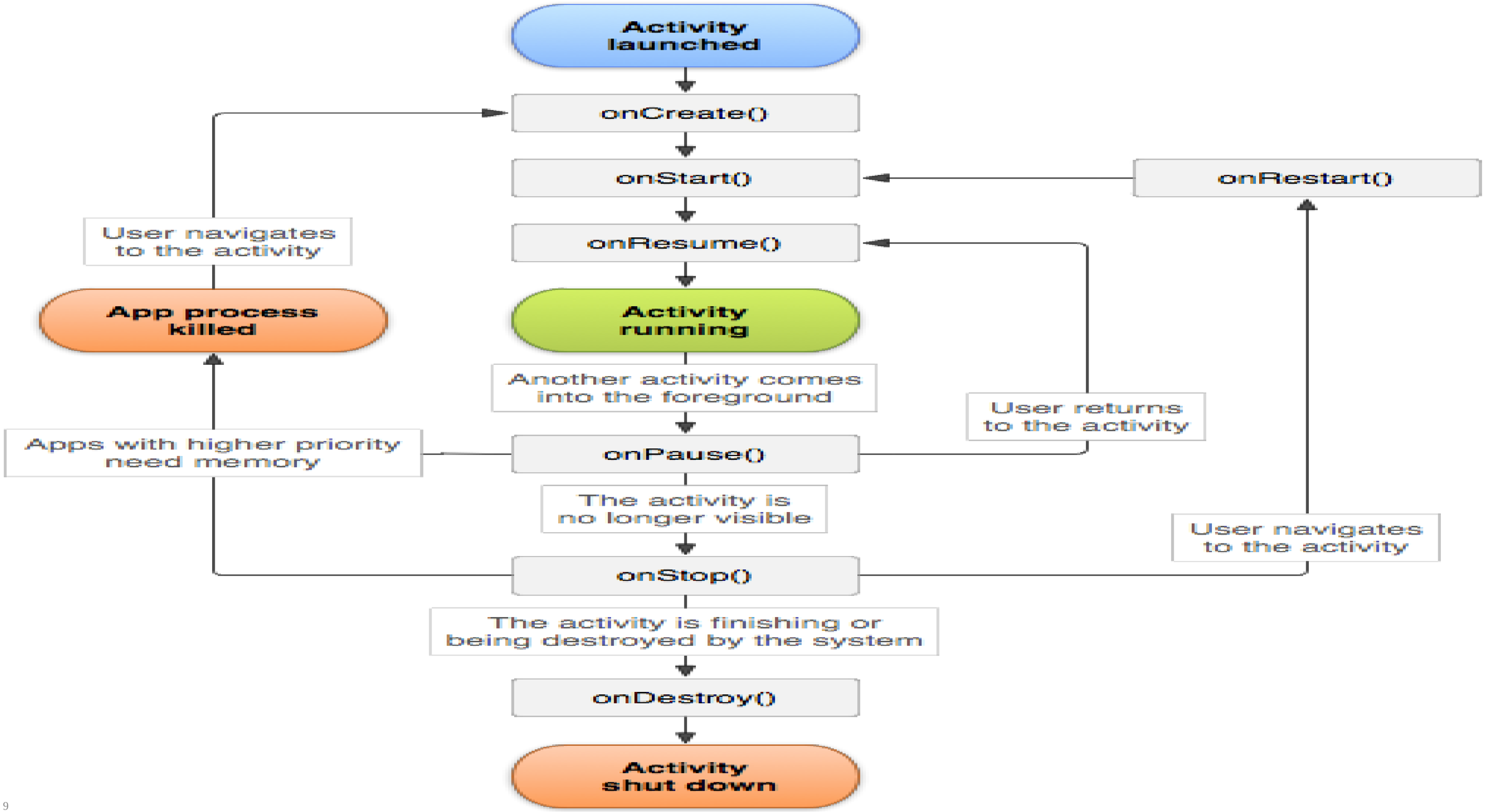
- **public class MainActivity extends AppCompatActivity** {

```
    @Override
    protected void  onCreate  (Bundle savedInstanceState) {
        super. onCreate (savedInstanceState);

        setContentView(R.layout.activity_main);
    }
}
```

Store information about activity layout to be loaded on recreating the activity

res layout XML file associated with UI

Activity Lifecycle



Essentially states of an activity:

1.Active and visible : If an activity is in the foreground of the screen, this is the activity that the user is currently interacting with.

2. Visible but not active : If an activity has lost focus but is still *visible*. It is possible if a new non-full-sized activity has focus. Such activity is completely alive (it maintains all state and information when user returns to it **(onResume())** is called.

3. Invisible and not active (resident in memory): If an activity is completely obscured by another activity, (navigation between application activates) when user returns to it **(onstart())** is called.

4. Invisible and not active (not resident in memory): If activity is killed by the system when memory is needed elsewhere. When user return to it **oncreate()** is called by saved instance.

5. Distorted or shut down by user (should be lunched by operating system

Activity methods onCreate() onStart()

```
public class MainActivity extends Activity {
  String msg = "Android : ";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d (msg, "The onCreate(  ) event");
}

/** Called when the activity is about to become visible. */
@Override
protected void onStart() {
  super.onStart();
  Log.d (msg, "The onStart( ) event");
}
```

Define a String variable with name msg

This code will be executed when the activity is created

This code will be executed when the activity starts

Activity methods

- **onCreate()** Called when the activity is first created. This method also provides you with a Bundle containing the activity's previously frozen state

- **onStart( )** Called when the activity is becoming visible to the user.
- (from resident in memory to be visible and interactive)

- **onResume()** Called when the activity will start interacting with the user.
- (from only visible to be visible and interactive)

- **onPause()** Called when the activity loses foreground state, is no longer focusable (from visible and interactive to be only visible)

- **onStop()** Called when the activity is no longer visible to the user. This may happen either because a new activity is being started on top.

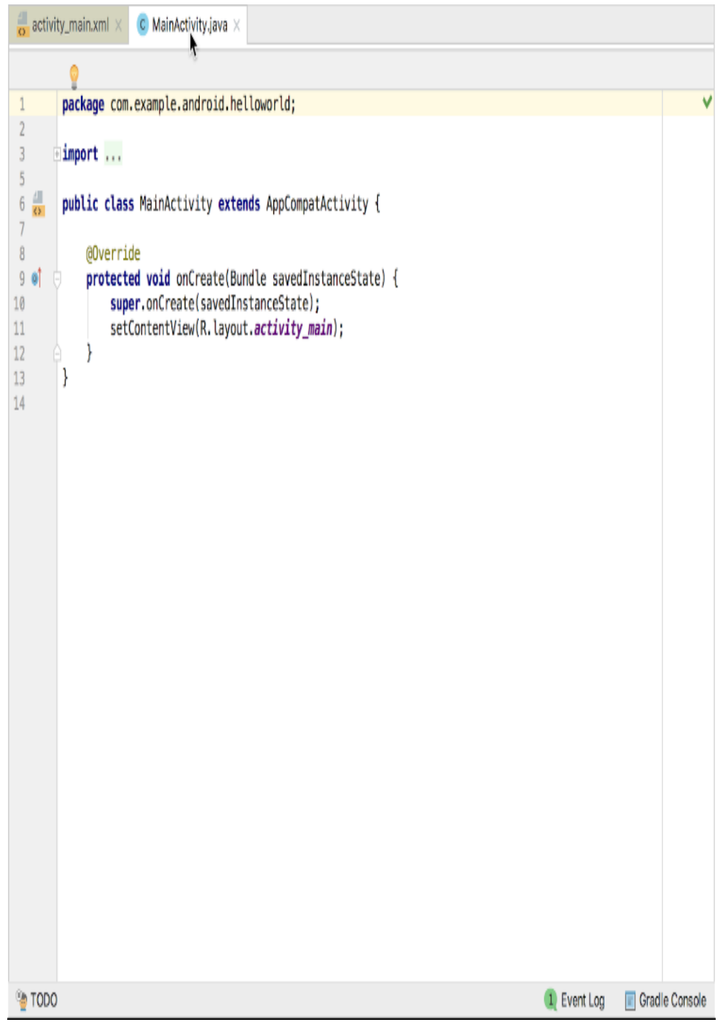
Main Activity

- Bundle class is used to stored the data of activity whenever above condition occur in app.

Bundle class

is used to store data about activities. The values that are to be passed are mapped to String keys which are later used in the next activity to retrieve the values.

```
public class MainActivity extends

AppCompatActivity {



    private static String TAG = "ActivityName";

public void onCreate(Bundle savedInstanceState)

{

    super.onCreate(savedInstanceState);

    setContentView(R.layout.content_main);

}
```



Bundle class

Bundle class is used to stored the data of activity whenever above condition occur in app.

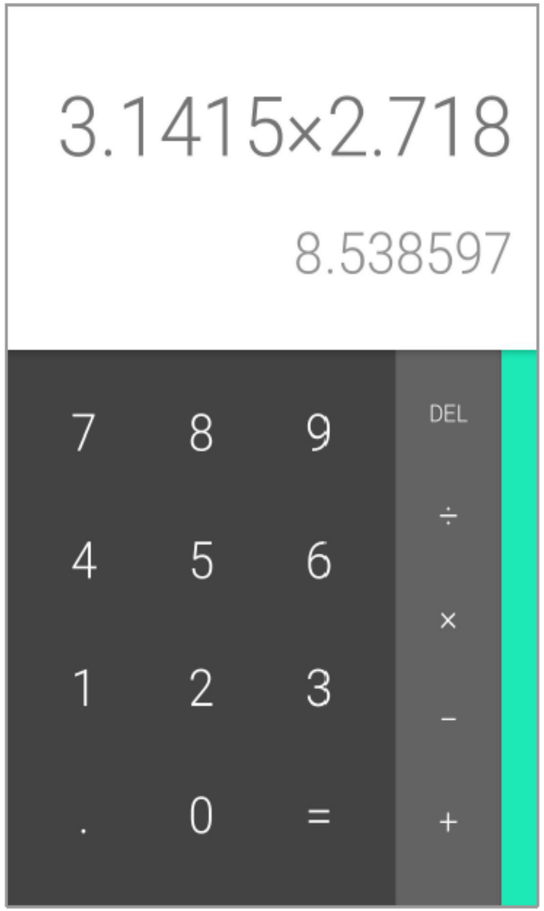
Bundles can stores all types of values and pass them to the new activity.

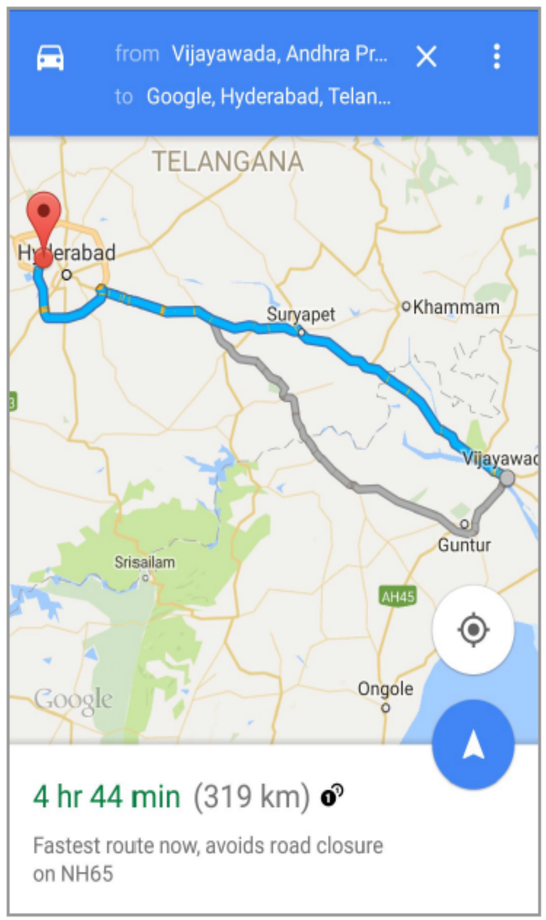
Activity in android studio

- **Activity** is a java class that creates and default window on the screen where we can place different components such as Button, EditText, TextView, Spinner etc.
- Unlike programming paradigms in which apps are launched with a main( ) method, the Android system initiates code in an Activity instance.
- The entry point of lunching (starting) application by OS is the Main Activity class, then the main activity can call another activity and so on.
-

Activities

- An Activity is an application component that represents one window, one hierarchy of views





Implementing an Activity

1. Define Activity Java class extends AppCompatActivity

2. Define layout in XML

3. Connect Activity with Layout Set content view in onCreate()

4. Declare Activity in the Android manifest

1. Define Activity Java class extends AppCompatActivity

public class MainActivity **extends AppCompatActivity** {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

{

{

2. Define layout in XML

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

android:orientation="vertical" >

<TextView **android:id="@+id/text"**
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a TextView" />

<Button **android:id="@+id/button"**
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a Button" />

</LinearLayout>

1.Any View object have an integer ID associated with it to uniquely identify.
It allow the view object to be used in Java code

2.wrap_content tells your view to size itself to the dimensions required by its content.

3. What will be written inside the object

3. Connect Activity with Layout

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Resource is layout in this XML file

4. Declare Activity in the Android manifest

Main Activity needs to include intent to start from launcher icon

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

The activity serves as the entry point for the application.

the activity can be launched from the device's launcher icon

Intent

- Intent objects allow android components to **communicate** “send messages around”

- Intents allow Android components to **request functionality** from other Android components.

- In other words : all components (applications and screens) of the Android device are **isolated**. The only way they communicate with each other is through intents.

- When you open up the **Instagram** app on your phone and use it to take a picture, you just made use of an intent.

  Intent object is responsible to pass the **picture** taken by **camera** to Instgram app.

- you can use intents to start different components: activities, services, and broadcast receivers.

Starting activities with intents object

- An Intent is a messaging object you can use to request an action from another app component.

Starting an activity

- You can start a new instance of an Activity by passing an Intent to **startActivity ( ) method**.

- The Intent describes the activity to start and carries any necessary data.

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

startActivity ( i );
```

Explicit intent and Implicit intent

An Intent is a messaging object you can use to request an action from another app component. It is mainly used to start an Android component

Starting an activity

Starting a service

Delivering a broadcast

Two types of Intent

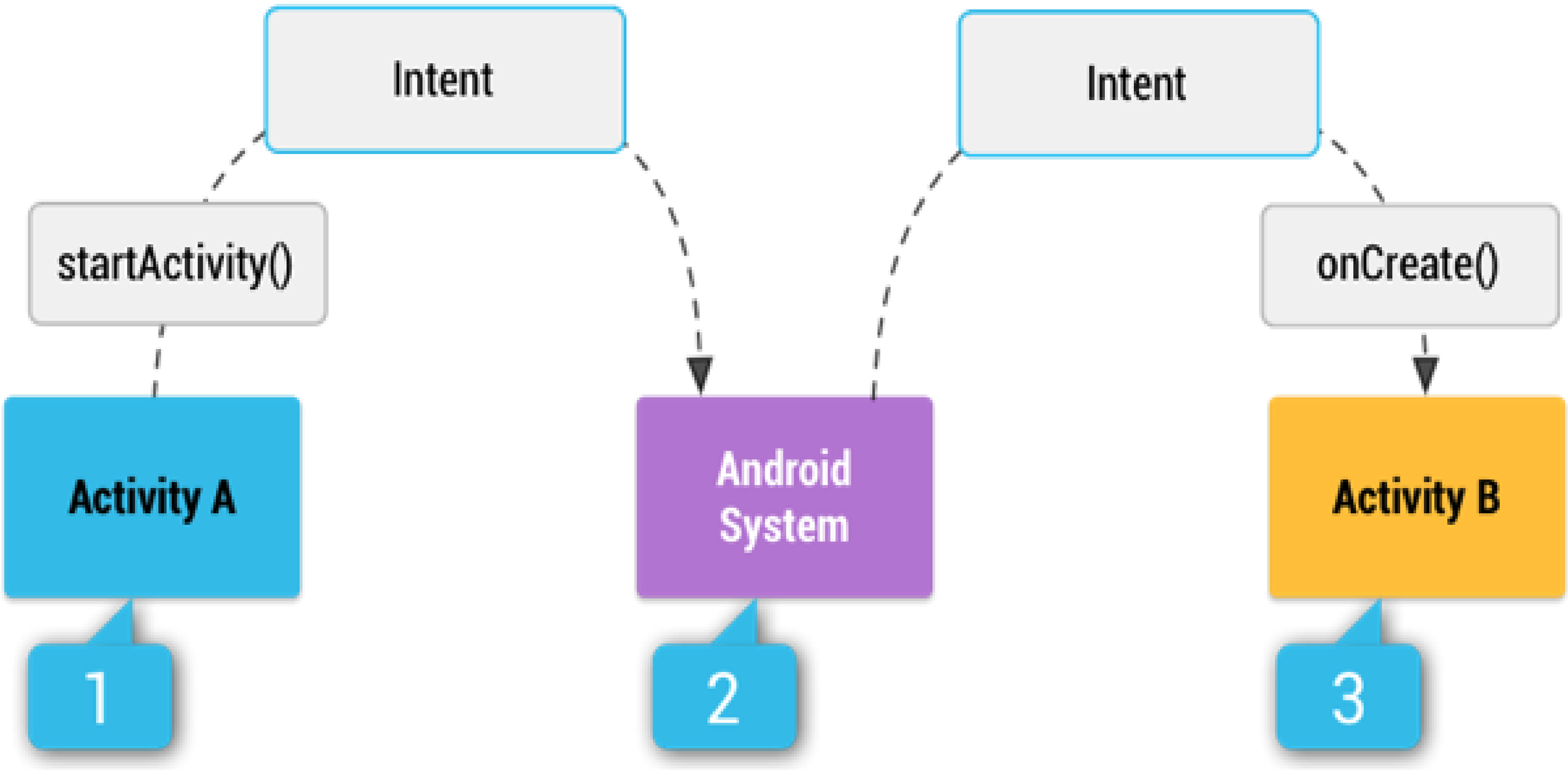
1. Explicit intents specify which application component will satisfy the intent, by supplying the component class name.

Use constructor that takes two **Intent(Context, Class)**

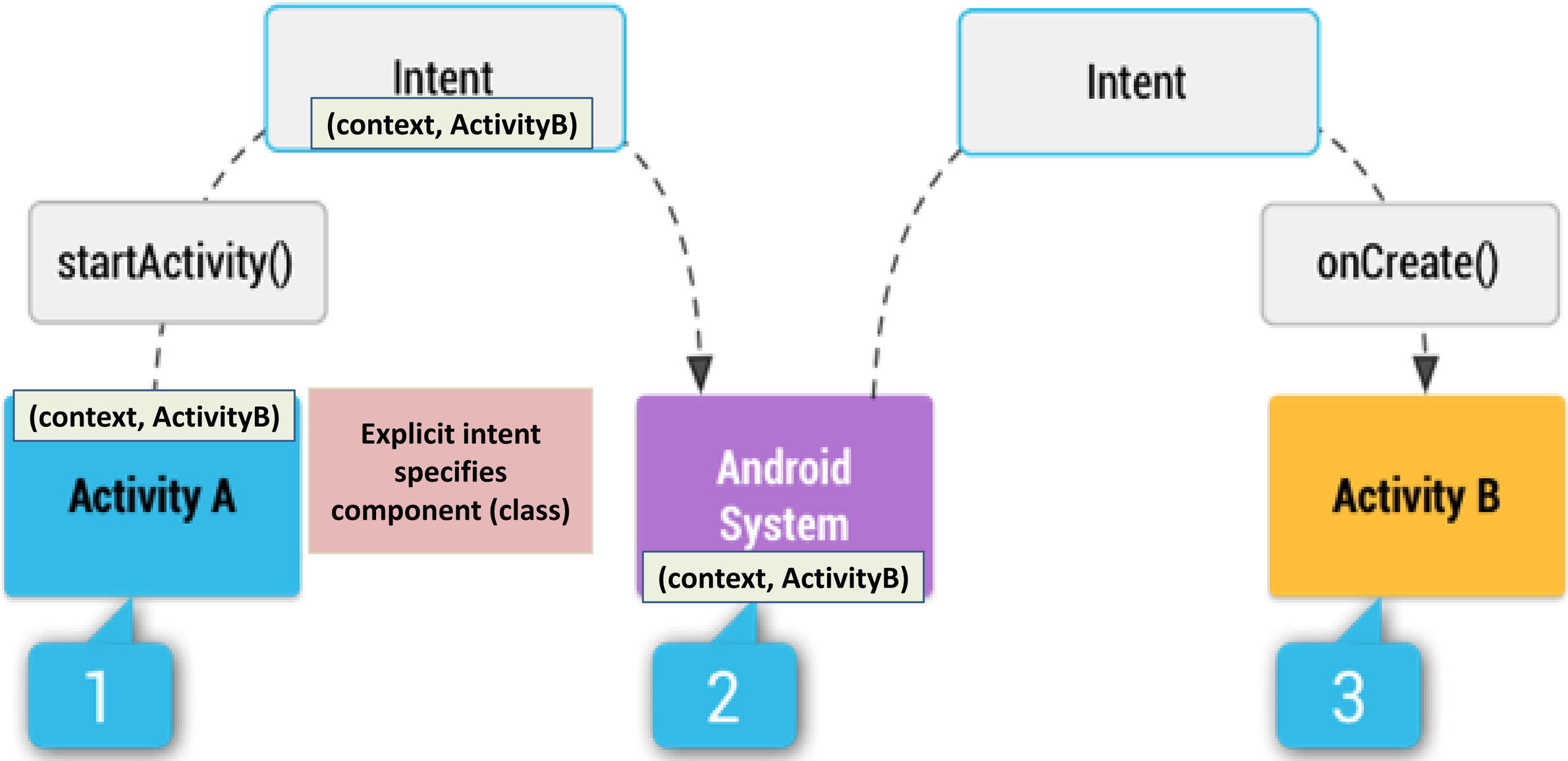
2. Implicit intents do not name a specific component, but instead declare a general action to perform

uses the non-argument constructor **Intent ( )**

The Intent Lifecycle



Explicit Intent



List of tags that describe components in manifest XML file
Mian components of Android studio

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

Methods of Intent

1.startActivity()

This is to launch a new activity or get an existing activity to be action.

2. startService()

This is to start a new service or deliver instructions for an existing service.

3. sendBroadcast()
This is to deliver the message to broadcast receivers.

Implicit intent and explicit intent

An Intent is a messaging object you can use to request an action from another app component. It is mainly used to start an Android component

Starting an activity

Starting a service

Delivering a broadcast

Two types of Intent

1. Explicit intents specify which application will satisfy the intent, by supplying the component class name.

Use constructor that takes two **Intent (Context, Class)**

- Intent **i** = new Intent (this, ActivityTwo.class);
- startActivity (**i**);

2. Implicit intents do not name a specific component, but instead declare a general action to perform

uses the non-argument constructor **Intent ( )**

Starting a service using Intent

In android studio a Service is a component that performs operations in the background without a user interface.

Consider that you have a service in your app that download a file from the web ad you want to start it.

This code is associated with starting the service.

```
Intent downloadIntent = new Intent  (this, DownloadService.class );

downloadIntent.setData(Uri.parse(fileUrl));


startService(downloadIntent);
```

1. Create an object of class Intent with two argument constructor (first the context , second the service name)
2. Set the data of intent object with file URL
3. Call the “startService” method by passing the intent object as parameter

Explicit intents

Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name.

Explicit intent is created using a constructor that takes two parameters. **Intent(Context, Class) constructor**

1. The context in which the intent will be used
2. The class object which will be invoked with the intent

Create an Intent to start a download service.

```
Intent   downloadIntent  =  new Intent (this, DownloadService.class);

downloadIntent.setData (Uri.parse(fileUrl));

startService (downloadIntent);
```

implicit intent

- An implicit intent specifies an action that can invoke any app on the device able to perform the action.

- For example, if you want the user to share content with other people, create an intent with the ACTION_SEND action and add extras that specify the content to share.

- **Intent sendIntent** = **new Intent** ( );

```
sendIntent.setAction (Intent.ACTION_SEND);

// Try to invoke the intent.

try {

    startActivity (sendIntent);

} catch (ActivityNotFoundException e) {

    // Define what your app should do if no activity can handle the intent.

}
```

Services in Android studio

- A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed.

- A service can essentially take two states

1.Started

– A service is **started** when an application component, such as an activity, starts it by calling *startService()*.

–Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

2.Bound

–A service is **bound** when an application component binds to it by calling *bindService()*.

–A bound service offers a client-server interface that allows components to interact with the service, send requests, and get results.

Service life cycle methods

A service has life cycle callback methods that can be implemented to perform work at the different stages and also to monitor its changes.