ANDROID - APPLICATION COMPONENTS

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application: -

	Components & Description
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	**Content Providers** They handle data and database management issues.

1) Activities

An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of Activity class as follows: -

```
public class MainActivity extends Activity {
}
```

2) Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of Service class as follows: -

```
public class MyService extends Service {
}
```

3) Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

Example: Battery low notifier, Incoming & outgoing calls, Airplane mode status, mobile data status.

A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcaster as an Intent object.

```
public class MyReceiver extends BroadcastReceiver {
  public void onReceive(context,intent){}
}
```

4) Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. The data may be stored in the file system, the database or somewhere else entirely.

Example: Accessing contact list, media files, gallery and videos.

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

```
public class MyContentProvider extends ContentProvider {
  public void onCreate(){}
}
```

Additional Components

There are additional components which will be used in the construction of above-mentioned entities, their logic, and wiring between them. These components are: -

	Components & Description
1	**Fragments** Represents a portion of user interface in an Activity
2	**Views** UI elements that are drawn on-screen including buttons, lists forms etc.
3	**Layouts** View hierarchies that control screen format and appearance of the views.
4	**Intents** Messages wiring components together.
5	**Resources** External elements, such as strings, constants and drawable pictures.
6	**Manifest** Configuration file for the application.

CREATING ACTIVITIES

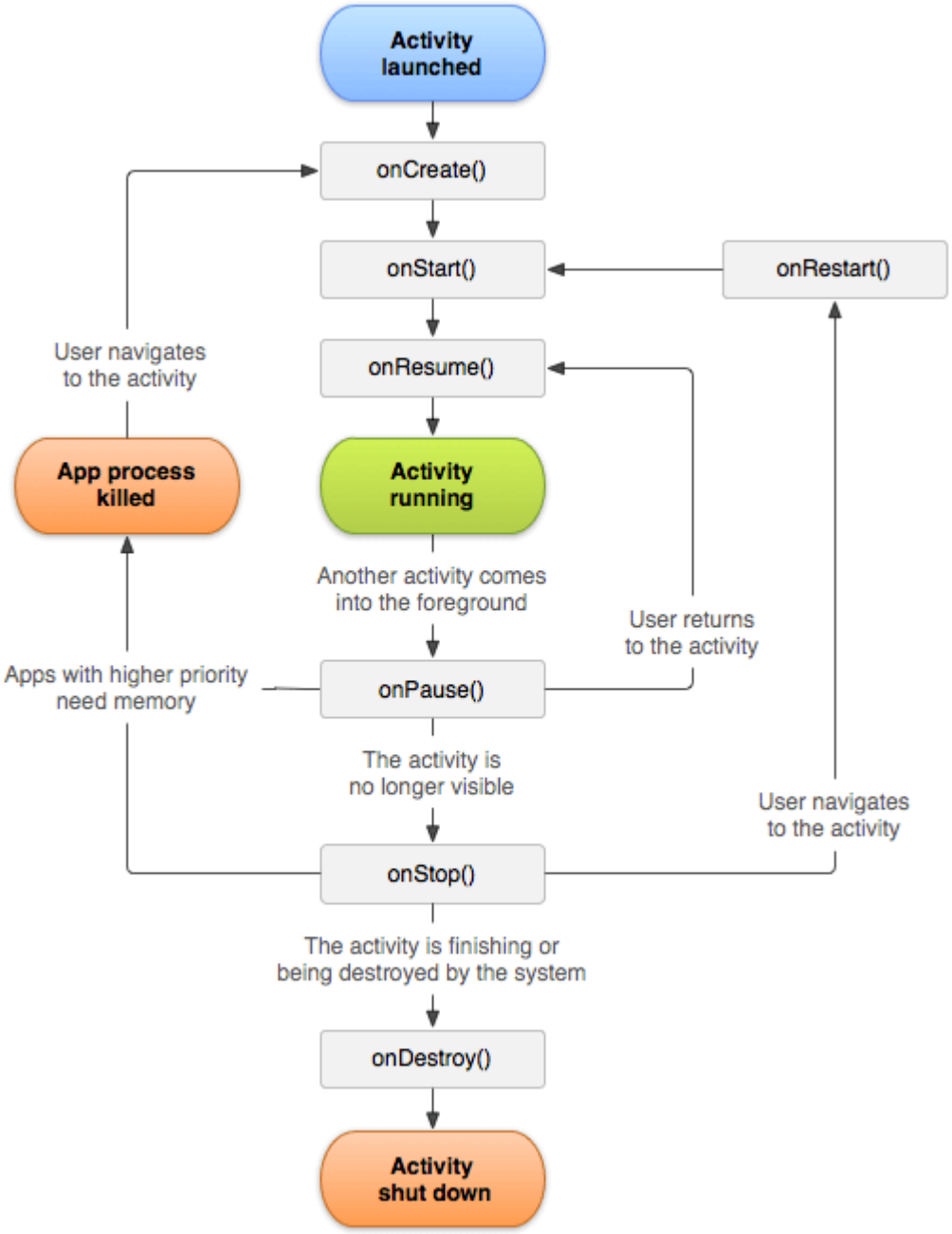
An activity is the single screen in android. It is like window or frame of Java.

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

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.

Android Activity Lifecycle

Let's see the 7 lifecycle methods of android activity.



The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

- **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 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.

Example

This example will take you through simple steps to show Android application activity life cycle. Follow the following steps to modify the Android application we created in Hello World Example.

Step 1

You will use Android studio to create an Android application and name it as HelloWorld under a package World Example chapter.

Step 2

Modify main activity file MainActivity.java as explained below. Keep rest of the files unchanged.

Step 3

Run the application to launch Android emulator and verify the result of the changes done in the application.

Android Activity Lifecycle Example

It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

```kotlin
package com.example.maina2

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets ^setOnApplyWindowInsetsListener
        }
    }
}
```

An activity class loads all the UI component using the XML file available in res/layout folder of the project. Following statement loads UI components from res/layout/activity_main.xml file:

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

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your AndroidManifest.xml file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

```xml
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Navigate to New Activity

I. Define the Button in activity_main.xml

Make sure you have a button defined in your activity_main.xml file:

The activity_main.xml file defines a button with an id of buttonNext.

```
<!-- activity_main.xml -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/buttonNext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Next"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```

II. Create the New Activity

Create a new activity that you want to navigate to. Let's call it NextActivity.

- Right-click on your java directory in Android Studio.
- Select New -> Activity -> Empty Activity.
- Name the activity NextActivity.

III. Define the Button's OnClickListener in MainActivity.java

Now, you need to handle the button click in your MainActivity.java file to navigate to NextActivity.

The MainActivity.java file sets an OnClickListener on the button to start NextActivity.

```
// MainActivity.java
package com.example.myapp;

import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button buttonNext = findViewById(R.id.buttonNext);
        buttonNext.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, NextActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

IV. Update the AndroidManifest.xml

Ensure that both activities are declared in the AndroidManifest.xml file.

The AndroidManifest.xml file declares both MainActivity and NextActivity.

```
<!-- AndroidManifest.xml -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

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

</manifest>
```

Purpose of Intent

1. **Navigation Between Activities**
    - An **Intent** is a messaging object that you can use to request an action from another app component. In this case, it is used to navigate from **MainActivity** to **NextActivity**.
    - When you call **startActivity(intent)**, the Android system looks for the component specified in the **Intent** and starts it. If the specified component is an activity, it brings that activity to the foreground.
2. **Data Transfer**
    - **Intent** can carry data between activities. You can use extras to attach additional information that the next activity might need.
3. **Component Invocation**
    - It can be used to invoke different app components such as activities, services, and broadcast receivers.