**ACTIVITY in android**

The [Activity](https://developer.android.com/reference/android/app/Activity.html) class is a crucial component of an Android app. Android system initiates code in an [Activity](https://developer.android.com/reference/android/app/Activity.html) instance by invoking specific callback methods that correspond to specific stages of its lifecycle. We may implement an activity as a subclass of the [Activity](https://developer.android.com/reference/android/app/Activity.html) class.

An activity provides the window in which the app draws its UI.

Generally, one activity implements one screen in an app.

Most apps contain multiple screens, which means they comprise multiple activities. Typically, one activity in an app is specified as the main activity, which is the first screen to appear when the user launches the app. Each activity can then start another activity in order to perform different actions. For example, the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox. From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

To use activities in our app, we must register information about them in the app’s manifest, and we must manage activity lifecycles appropriately.

Configuring the manifest

### Declare activities

To declare your activity, open your manifest file and add an [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html) element as a child of the [<application>](https://developer.android.com/guide/topics/manifest/application-element.html)element. For example:

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

The only required attribute for this element is [android:name](https://developer.android.com/guide/topics/manifest/activity-element.html" \l "nm), which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme.

### Declare intent filters

[Intent filters](https://developer.android.com/guide/components/intents-filters.html) are a very powerful feature of the Android platform. They provide the ability to launch an activity based not only on an explicit request, but also an implicit one. For example, an explicit request might tell the system to “Start the Send Email activity in the Gmail app". By contrast, an implicit request tells the system to “Start a Send Email screen in any activity that can do the job." When the system UI asks a user which app to use in performing a task, that’s an intent filter at work.

<activity android:name=".ExampleActivity" android:icon="@drawable/app\_icon">  
    <intent-filter>  
        <action android:name="android.intent.action.SEND" />  
        <category android:name="android.intent.category.DEFAULT" />  
        <data android:mimeType="text/plain" />  
    </intent-filter>  
</activity>

In this example, the [<action>](https://developer.android.com/guide/topics/manifest/action-element.html) element specifies that this activity sends data. Declaring the [<category>](https://developer.android.com/guide/topics/manifest/category-element.html) element as DEFAULT enables the activity to receive launch requests. The [<data>](https://developer.android.com/guide/topics/manifest/data-element.html) element specifies the type of data that this activity can send. The following code snippet shows how to call the activity described above:

// Create the text message with a string  
Intent sendIntent = new Intent();  
sendIntent.setAction(Intent.ACTION\_SEND);  
sendIntent.setType("text/plain");  
sendIntent.putExtra(Intent.EXTRA\_TEXT, textMessage);  
// Start the activity  
startActivity(sendIntent);

### Declare permissions

You can use the manifest's [<activity>](https://developer.android.com/guide/topics/manifest/activity-element.html) tag to control which apps can start a particular activity. A parent activity cannot launch a child activity unless both activities have the same permissions in their manifest. If you declare a[<uses-permission>](https://developer.android.com/guide/topics/manifest/uses-permission-element.html) element for a particular activity, the calling activity must have a matching [<uses-permission>](https://developer.android.com/guide/topics/manifest/uses-permission-element.html) element.

For example, if your app wants to use a hypothetical app named SocialApp to share a post on social media, SocialApp itself must define the permission that an app calling it must have:

<manifest>  
<activity android:name="...."  
   android:permission=”com.google.socialapp.permission.SHARE\_POST”  
  
/>

Then, to be allowed to call SocialApp, your app must match the permission set in SocialApp's manifest:

<manifest>  
   <uses-permission android:name="com.google.socialapp.permission.SHARE\_POST" />  
</manifest>

## Managing the activity lifecycle

### onCreate()

We must implement this callback, which fires when the system creates a activity. We should initialize the essential components of our activity: For example, the app should create views and bind data to lists here. Most importantly, this is where you must call [setContentView()](https://developer.android.com/reference/android/app/Activity.html" \l "setContentView(android.view.View)) to define the layout for the activity's user interface.

When [onCreate()](https://developer.android.com/reference/android/app/Activity.html" \l "onCreate(android.os.Bundle)) finishes, the next callback is always [onStart()](https://developer.android.com/reference/android/app/Activity.html" \l "onStart()).

### onStart()

As [onCreate()](https://developer.android.com/reference/android/app/Activity.html" \l "onCreate(android.os.Bundle)) exits, the activity enters the Started state, and the activity becomes visible to the user. This callback contains what amounts to the activity’s final preparations for coming to the foreground and becoming interactive.

### onResume()

The system invokes this callback just before the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, and captures all user input. Most of an app’s core functionality is implemented in the [onResume()](https://developer.android.com/reference/android/app/Activity.html" \l "onResume()) method.

The [onPause()](https://developer.android.com/reference/android/app/Activity.html" \l "onPause()) callback always follows [onResume()](https://developer.android.com/reference/android/app/Activity.html" \l "onResume()).

### onPause()

The system calls [onPause()](https://developer.android.com/reference/android/app/Activity.html" \l "onPause()) when the activity loses focus and enters a Paused state. This state occurs when, for example, the user taps the Back or Recents button. When the system calls [onPause()](https://developer.android.com/reference/android/app/Activity.html" \l "onPause()) for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.

Once [onPause()](https://developer.android.com/reference/android/app/Activity.html" \l "onPause()) finishes executing, the next callback is either [onStop()](https://developer.android.com/reference/android/app/Activity.html" \l "onStop()) or [onResume()](https://developer.android.com/reference/android/app/Activity.html" \l "onResume()), depending on what happens after the activity enters the Paused state.

### onStop()

The system calls [onStop()](https://developer.android.com/reference/android/app/Activity.html" \l "onStop()) when the activity is no longer visible to the user. This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is entering a Resumed state and is covering the stopped activity. In all of these cases, the stopped activity is no longer visible at all.

The next callback that the system calls is either [onRestart()](https://developer.android.com/reference/android/app/Activity.html" \l "onRestart()), if the activity is coming back to interact with the user, or by [onDestroy()](https://developer.android.com/reference/android/app/Activity.html" \l "onDestroy()) if this activity is completely terminating.

### onRestart()

The system invokes this callback when an activity in the Stopped state is about to restart. [onRestart()](https://developer.android.com/reference/android/app/Activity.html" \l "onRestart()) restores the state of the activity from the time that it was stopped.

This callback is always followed by [onStart()](https://developer.android.com/reference/android/app/Activity.html" \l "onStart()).

### onDestroy()

The system invokes this callback before an activity is destroyed.

This callback is the final one that the activity receives. [onDestroy()](https://developer.android.com/reference/android/app/Activity.html" \l "onDestroy()) is usually implemented to ensure that all of an activity’s resources are released when the activity, or the process containing it, is destroyed.

