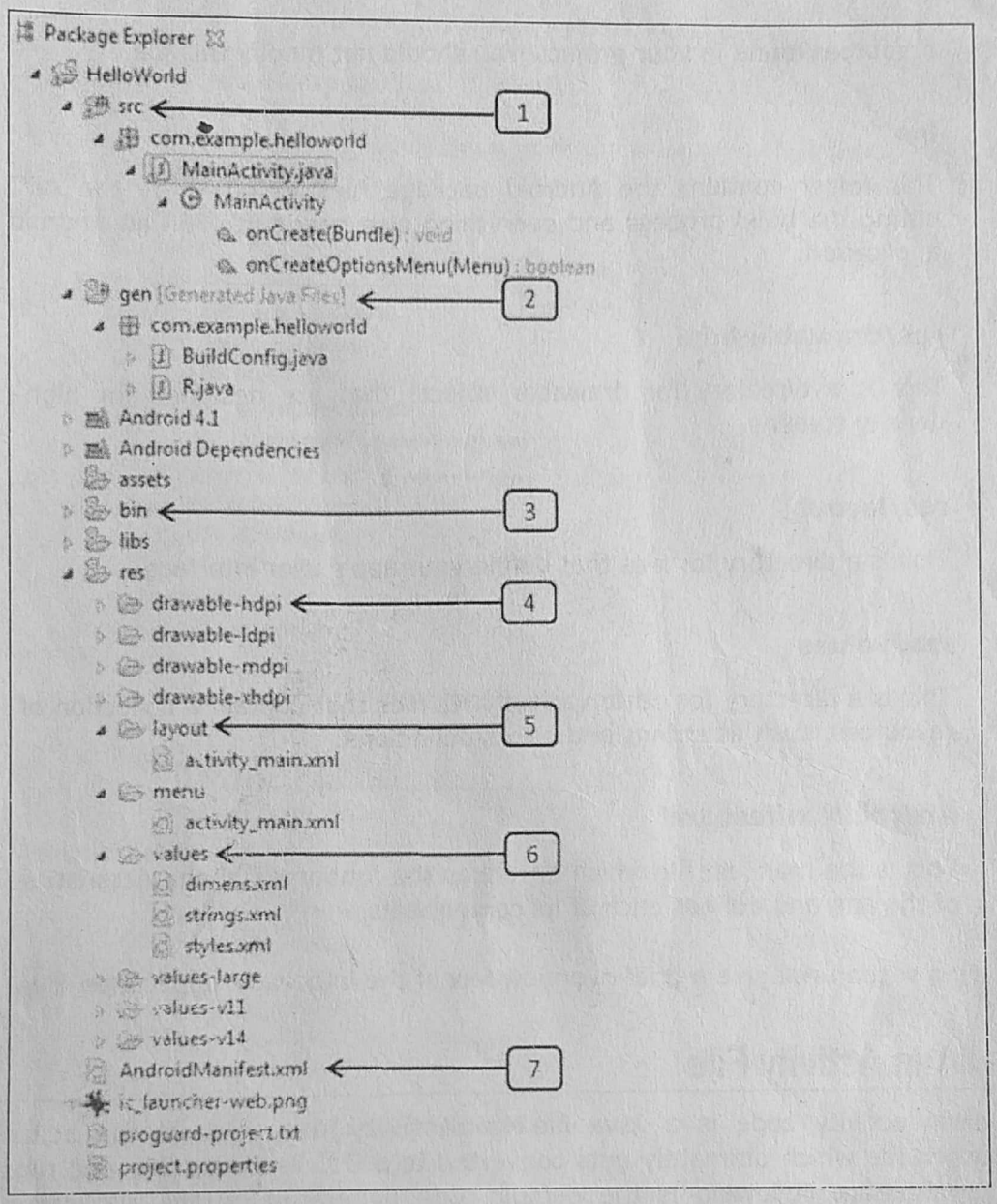
chapter3-2 Android Application Design

Anatomy Android Application



S.N.	Folder, File & Description
1	**src** This contains the **.java** source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
2	**gen** This contains the **.R** file, a compiler-generated file that references all the

	resources found in your project. You should not modify this file.
3	**bin** This folder contains the Android package files **.apk** built by the ADT during the build process and everything else needed to run an Android application.
4	**res/drawable-hdpi** This is a directory for drawable objects that are designed for high-density screens.
5	**res/layout** This is a directory for files that define your app's user interface.
6	**res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
7	**AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Following section will give a brief overview few of the important application files.

The Main Activity File

The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application:

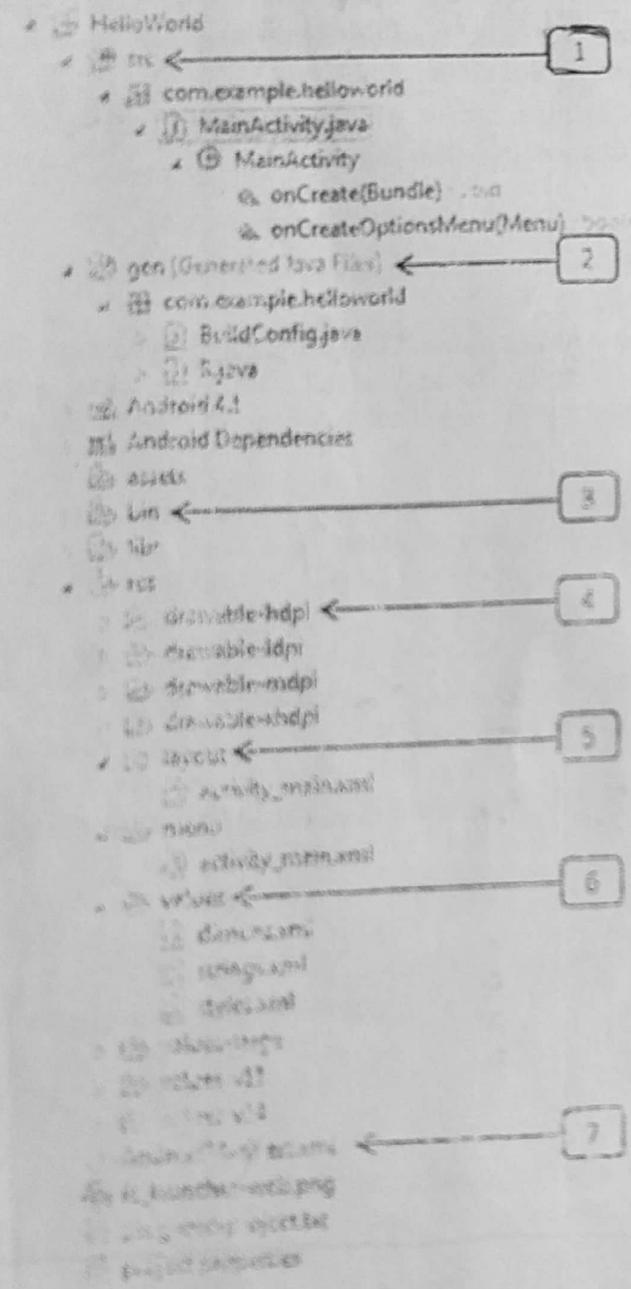
```
package com.example.helloworld;


import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v4.app.NavUtils;
```

Chapter :-2 Android Application Design

Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Andriod project:



S.N. Folder, File & Description

1 src :- This contains the .java source files for your project. By default, it includes an MainActivity.javasource file having an activity class that runs when your app is launched using the app icon.

2 **gen :-** This contains the .R file, a compiler-generated file that references all the resources found in your project. You should not modify this file.

3 **Bin :-** This folder contains the Android package files .apk built by the ADT during the build process and everything else needed to run an Android application.

4 **Res/drawable-hdpi :-** This is a directory for drawable objects that are designed for high-density screens.

5 **Res/layout :-** This is a directory for files that define your app's user interface.

6 **Res/values :-** This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.

7 **AndroidManifest.xml:-** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Following section will give a brief overview few of the important application files.

➢ **The Main Activity File**

The main activity code is a Java file MainActivity.java. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for Hello World! application:

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.MenuItem;
import android.support.v4.app.NavUtils;

public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Here, R.layout.activity_main refers to the activity_main.xml file located in the res/layout folder. TheonCreate() method is one of many methods that are fi red when an activity is loaded.

➢ **The Manifest File**

⇒ Whatever component you develop as a part of your application, you must declare all its components in a manifest file called AndroidManifest.xml which ressides at the root of the application project directory.

⇒ This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
   <application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name=".MainActivity"
           android:label="@string/title_activity_main" >
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
   </application>
</manifest>
```

⇒ Here **<application>...</application>** tags enclosed the components related to the application. Attributeandroid:icon will point to the application icon available under res/drawable-hdpi. The application uses the image named ic_launcher.png located in the drawable folders The <activity> tag is used to specify an activity and android:name attribute specifies the fully qualified class name of the Activity subclass and the android:label attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

⇒ The action for the intent filter is named android.intent.action.MAIN to indicate that this activity serves as the entry point for the application. The category for the intent-filter is namedandroid.intent.category.LAUNCHER to indicate that the application can be launched from the device's launcher icon.

⇒ The @string refers to the strings.xml file explained below. Hence, @string/app_name refers to theapp_name string defined in the strings.xml fi le, which is "HelloWorld". Similar way, other strings get populated in the application.

⇒ Following is the list of tags which you will use in your manifest file to specify different Android application components:

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

➢ **The Strings File**

⇒ The strings.xml file is located in the res/values folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file.

HDPI High Dots per Inch

PNG Portable Network Graphics

⇒ This file is responsible for their textual content. For example, a default strings file will look like as following file:

```
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
</resources>
```

➢ **The R File**

The gen/com.example.helloworld/R.java file is the glue between the activity Java files likeMainActivity.java and the resources like strings.xml. It is an automatically generated file and you should not modify the content of the R.java file. Following is a sample of R.java file:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.example.helloworld;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f070000;
    }
    public static final class string {
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050001;
        public static final int menu_settings=0x7f050002;
        public static final int title_activity_main=0x7f050003;
    }
    public static final class style {
        public static final int AppTheme=0x7f060000;
    }
}
```

➢ **The Layout File**

⇒ The activity_main.xml is a layout file available in res/layout directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout:

```
<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" >

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:padding="@dimen/padding_medium"
        android:text="@string/hello_world"
        tools:context=".MainActivity" />

</RelativeLayout>
```

⇒ This is an example of simple RelativeLayout which we will study in a separate chapter. The TextView is an Android control used to build the GUI and it have various attribuites like android:layout_width,android:layout_height etc which are being used to set its width and height etc.

⇒ The @string refers to the strings.xml file located in the res/values folder. Hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

(2) Android Terminologies

1. **.apk file**

⇒ Android application package file. Each Android application is compiled and packaged in a single file that includes all of the application's code (.dex files), resources, assets, and manifest file. The application package file can have any name but must use the .apk extension. For example: myExampleAppname.apk. For convenience, an application package file is often referred to as an ".apk".

Related: Application.

2. **.dex file**

⇒ Android programs are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. .dex files can be created by automatically translating compiled applications written in the Java programming language.

3. **Action**

⇒ A description of something that an Intent sender wants done. An action is a string value assigned to an Intent. Action strings can be defined by Android or by a third-party developer. For example, android.intent.action.VIEW for a Web URL, or com.example.rumbler.SHAKE_PHONE for a custom application to vibrate the phone.

Related: Intent.

4. **Activity**

⇒ A single screen in an application, with supporting Java code, derived from the Activity class. Most commonly, an activity is visibly represented by a full screen window that can receive and handle UI events and perform complex tasks, because of the Window it uses to render its window. Though an Activity is typically full screen, it can also be floating or transparent.

6. **adb**

⇒ Android Debug Bridge, a command-line debugging application included with the SDK. It provides tools to browse the device, copy tools on the device, and forward ports for debugging. If you are developing in Eclipse using the ADT Plugin, adb is

integrated into your development environment. See Android Debug Bridge for more information.

6. **Application**

⇒ From a component perspective, an Android application consists of one or more activities, services, listeners, and intent receivers. From a source file perspective, an Android application consists of code, resources, assets, and a single manifest. During compilation, these files are packaged in a single file called an application package file (.apk).

Related: .apk, Activity

7. **Content Provider**

⇒ A data-abstraction layer that you can use to safely expose your application's data to other applications. A content provider is built on the ContentProvider class, which handles content query strings of a specific format to return data in a specific format. See Content Providers topic for more information.

Related: URI Usage in Android

8. **Dalvik**

⇒ The Android platform's virtual machine. The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-mappable execution.

⇒ The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The VM runs on top of Posix-compliant operating systems, which it relies on for underlying functionality (such as threading and low level memory management).

⇒ The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device.

9. **Drawable**

⇒ A compiled visual resource that can be used as a background, title, or other part of the screen. A drawable is typically loaded into another UI element, for example as a background image. A drawable is not able to receive events, but does assign various other properties such as "state" and scheduling, to enable subclasses such as animation objects or image libraries. Many drawable objects are loaded from drawable resource files — xml or bitmap files that describe the image. Drawable resources are compiled into subclasses of android.graphics.drawable. For more information about drawables and other resources, see Resources.

Related: Resources, Canvas

10. **Intent**

⇒ An message object that you can use to launch or communicate with other applications/activities asynchronously. An Intent object is an instance of Intent. It includes several criteria fields that you can supply, to determine what application/activity receives the Intent and what the receiver does when handling the Intent. The application can send the Intent to a single target application or it can send it as a broadcast, which can in turn be handled by multiple applications sequentially. For more information, see Intents and Intent Filters.

⇒ Related: Intent Filter, Broadcast Receiver.

19. A widget is a simple application extension that is often part of a larger application already installed on the device. Widget come in all shapes and size, are customisable, and reside on any available Home screen for quick

11. **Intent Filter**
⇒ A filter object that an application declares in its manifest file, to tell the system what types of Intents each of its components is willing to accept and with what criteria. Through an intent filter, an application can express interest in specific data types, Intent actions, URI formats,

12. **Broadcast Receiver**
⇒ An application class that listens for Intents that are broadcast, rather than being sent to a single target application/activity. The system delivers a broadcast Intent to all interested broadcast receivers, which handle the Intent sequentially.

13. **Manifest File**
⇒ An XML file that each application must define, to describe the application's package name, version, components (activities, intent filters, services), imported libraries, and describes the various activities, and so on. See The AndroidManifest.xml File for complete information.

14. **Service**

An object of class Service that runs in the background (without any UI presence) to perform various persistent actions, such as playing music or monitoring network activity.

~~Related: Activity~~

15 **Theme**

A set of properties (text size, background color, and so on) bundled together to define various default display settings. Android provides a few standard themes, listed in R.style (starting with "Theme_").

16. **View**

An object that draws to a rectangular area on the screen and handles click, keystroke, and other interaction events. A View is a base class for most layout components of an Activity or Dialog screen (text boxes, windows, and so on). It receives calls from its parent object (see viewgroup, below)to draw itself, and informs its parent object about where and how big it would like to be (which may or may not be respected by the parent). ~~For more information, see~~ View.

Related: ~~Viewgroup, Widget~~

17. **Window**
⇒ In an Android application, an object derived from the abstract class Window that specifies the elements of a generic window, such as the look and feel (title bar text, location and content of menus, and so on). Dialog and Activity use an implementation of this class to render a window. You do not need to implement this class or use windows in your application.

9 **What is Application context**
⇒ Context objects are so common, and get passed around so frequently. it can be easy to create a situation you didn't intend. Loading resources, launching a new Activity,

18 ViewGroup: A ViewGroup is a special View that can contain other views (called children). The View group is the base class for layouts and views containers. 8 This class also defines the ViewGroup. LayoutParams class which serves as the base class for layouts parameters.

what is context context objects are so common and get passed around so frequently. It can be easy to create a situation, you didn't intend. Loading resources, inflating layouts, finding system services, getting internal file paths and creating new Activities, obtaining a system service, getting internal file paths, and creating views all require a Context (and that's not even getting started on the full list!) to accomplish the task. What like to do is provide for you some insights on how Context works alongside some tips that will (hopefully) allow you to leverage it more effectively in your applications.

Context Types all require a context to accomplish the task.

Not all Context instances are created equal. Depending on the Android application component, the Context you have access to varies slightly:

1. Application – is a singleton instance running in your application process. It can be accessed via methods like getApplication() from an Activity or Service, and getApplicationContext() from any other object that inherits from Context. Regardless of where or how it is accessed, you will always receive the same instance from within your process.

2. Activity/Service – inherit from ContextWrapper which implements the same API, but proxies all of its method calls to a hidden internal Context instance, also known as its base context. Whenever the framework creates a new Activity or Service instance, it also creates a new ContextImpl instance to do all of the heavy lifting that either component will wrap. Each Activity or Service, and their corresponding base context, are unique per-instance.

3. **BroadcastReceiver** – is not a Context in and of itself, but the framework passes a Context to it in onReceive() each time a new broadcast event comes in. This instance is a ReceiverRestrictedContext with two main functions disabled; calling registerReceiver() and bindService(). These two functions are not allowed from within an existing BroadcastReceiver.onReceive(). Each time a receiver processes a broadcast, the Context handed to it is a new instance.

4. **ContentProvider** – is also not a Context but is given one when created that can be accessed via getContext(). If the ContentProvider is running local to the caller (i.e. same application process), then this will actually return the same Application singleton. However, if the two are in separate processes, this will be a newly created instance representing the package the provider is running in

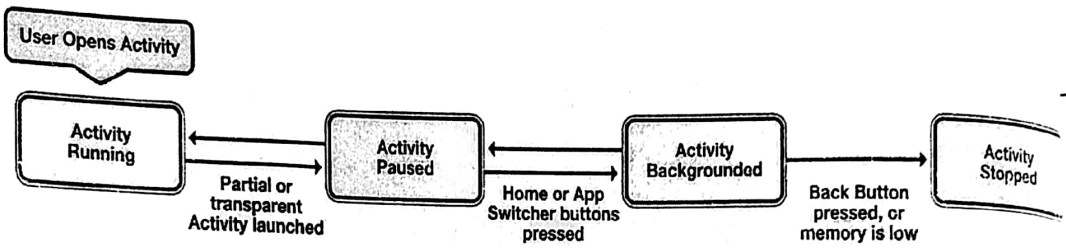
❖ **Introduction to activity**

- Activities are a fundamental building block of android application and they can exist in a number of different states
- Traditional application development :- static main method
- Android :- any registered activity within an application

5. saved References : The first issue we need to address comes from saving a reference to a Context in an object or class that has a lifecycle that extends beyond that of the instance you saved. For e.g., creating a custom singleton that requires a context to load resources or access a ContentProvider, and passing a reference to current Activity or service in that singleton

- Android maintains an history stack of all the activities which are spawned in an application .
- An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.
- Each activity can then start another activity in order to perform different actions.

- **Activity Life Cycle**
  - The Android OS arbitrates Activities based on their state.
  - This helps Android identify activities that are no longer in use, allowing the OS to reclaim memory and resources.
  - The following diagram illustrates the states an Activity can go through during its lifetime:



The state can be broken in to 4 main groups as follows:
(1)Active or Running
(2)Paused
(3)Stopped or Backgrounded
(4)Restarted

- **Activity or Running**
  - Activites are considered active or running if they are in the foreground, also known as the top of the activity stack
  - The highest priority activity in android

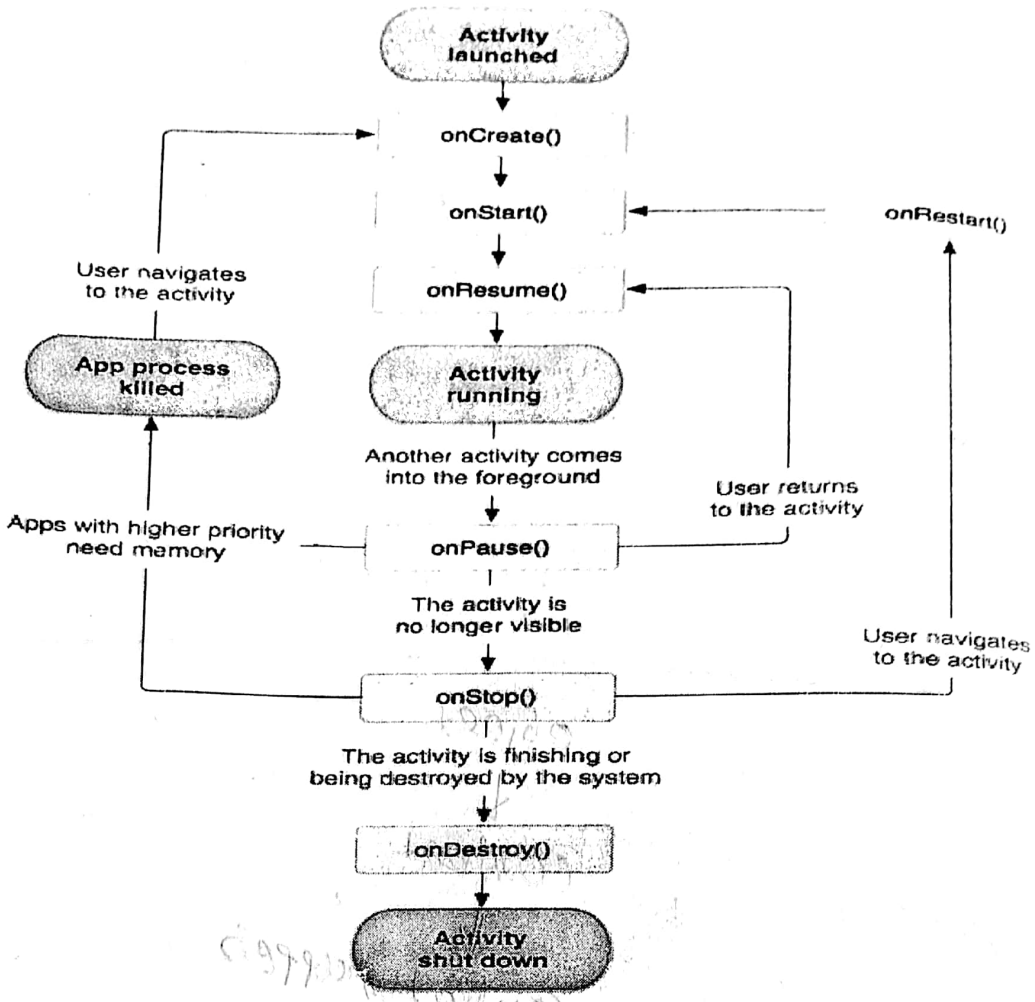
- **Paused**
  - When the device goes to sleep, or an activity is still visible but partially hidden by a new, non- full-sized or transparent activity, the activity is considered paused.

- They maintain all state and member information, and remain attached to the window manager
- The second highest priority activity in android

- **Stopped/ backgrounded**
  - Activities that are completely hidden by another activity are considered stopped or in the background
  - Stopped activates are considered to be the lowest priority of the three states

- **Restarted**
  - It is possible for an activity that is anywhere from paused to stopped in the lifecycle to be removed from memory by android.
  - If the user navigates back to the activity it must be restarted, restored to its previously saved state, and then displayed to the user.

object
↓
context
↓
ContextWrapper
↓
ContextThemeWrapper
↓
Activity



onCreate()	Called then the activity is created. Used to initialize the activity, for exam create the user interface.
onResume()	Called if the activity get visible again and the user starts interacting with activity again. Used to initialize fields, register listeners, bind to services, etc.
onPause()	Called once another activity gets into the foreground. Always called bef the activity is not visible anymore. Used to release resources or save applicat data. For example you unregister listeners, intent receivers, unbind from servi or remove system service listeners.
onStop()	Called once the activity is no longer visible. Time or CPU intensive shut-do operations, such as writing information to a database should be down the onStop() method. This method is guaranteed to be called as of API 11.

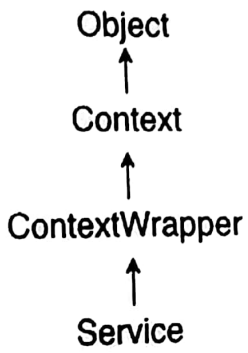
A service is a component that runs in background to perform long-running operations without needing to interact with the user and it works even if application is destroyed.

❖ **What is Android Service**

⇒ **Android service** is a component that is *used to perform operations on the background* such as playing music, handle network transactions, interacting content providers etc. It doesn't has any UI (user interface).

The service runs in the background indefinitely even if application is destroyed.

Moreover, service can be bounded by a component to perform interactivity and inter process communication (IPC).



The android.app.Service is subclass of ContextWrapper class.

Note: Android service is not a thread or separate process.

Life Cycle of Android Service

There can be two forms of a service.The lifecycle of service can follow two different paths: started or bound.

1. Started
2. Bound

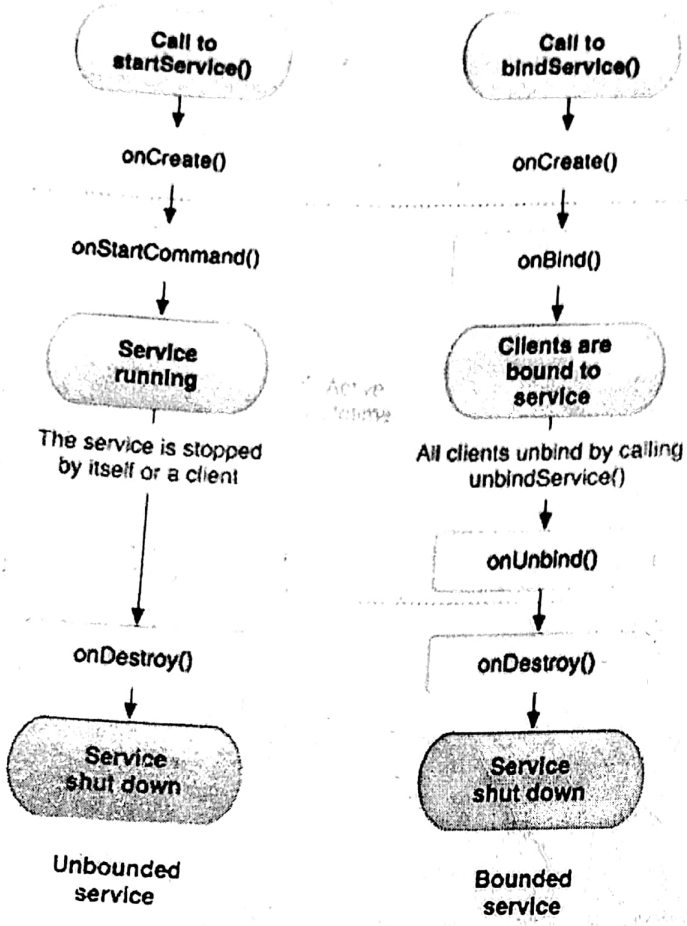
1) Started Service

A service is started when component (like activity) calls **startService()** method, now it runs in the background indefinitely. It is stopped by **stopService()** method. The service can stop itself by calling the **stopSelf()** method.

2) Bound Service

A service is bound when another component (e.g. client) calls **bindService()** method. The client can unbind the service by calling the **unbindService()** method.

The service cannot be stopped until all clients unbind the service.



To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. 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.

Callback	Description
onStartCommand()	The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods.
onBind()	The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow

	binding, then you should return *null*.
onUnbind()	The system calls this method when all clients have disconnected from a particular interface published by the service.
onRebind()	The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*.
onCreate()	The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time setup.
onDestroy()	The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

❖ **What is Intents?**

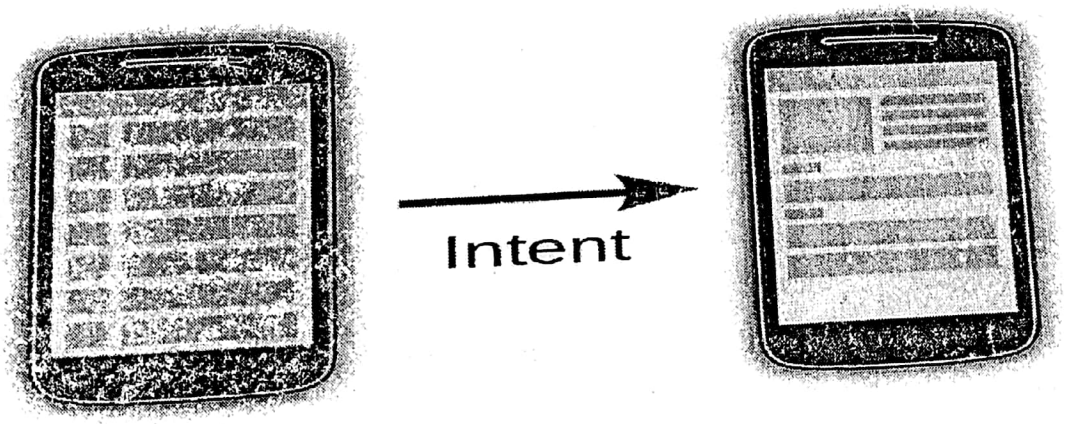
⇒ *Intents* are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

⇒ Intents are objects of the `android.content.Intent` type. Your code can send them to the Android system defining the components you are targeting. For example, via the `startActivity()` method you can define that the intent should be used to start an activity.

An intent can contain data via a `Bundle`. This data can be used by the receiving component.

1.2. Starting activities

To start an activity, use the method `startActivity(intent)`. This method is defined on the `Context` object which `Activity` extends.



The following code demonstrates how you can start another activity via an intent.

```
# Start the activity connect to the
# specified class

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

1.3. Sub-activities

Activities which are started by other Android activities are called *sub-activities*. This wording makes it easier to describe which activity is meant.

1.4. Starting services

You can also start services via intents. Use the `startService(Intent)` method call for that.

2. Intents types

2.1. Different types of intents

Android supports explicit and implicit intents.

An application can define the target component directly in the intent (*explicit intent*) or ask the Android system to evaluate registered components based on the intent data (*implicit intents*).

2.2. Explicit Intents

Explicit intents explicitly define the component which should be called by the Android system, by using the Java class as identifier.

The following shows how to create an explicit intent and send it to the Android system. If the class specified in the intent represents an activity, the Android system starts it.

```
Intent i = new Intent(this, ActivityTwo.class);
i.putExtra("Value1", "This value one for ActivityTwo ");
i.putExtra("Value2", "This value two ActivityTwo");
```

Explicit intents are typically used within on application as the classes in an application are controlled by the application developer.

2.3. Implicit Intents

Implicit intents specify the action which should be performed and optionally data which provides content for the action.

For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter.

```
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.vogella.com"));
startActivity(i);
```

If an implicit intent is sent to the Android system, it searches for all components which are registered for the specific action and the fitting data type.

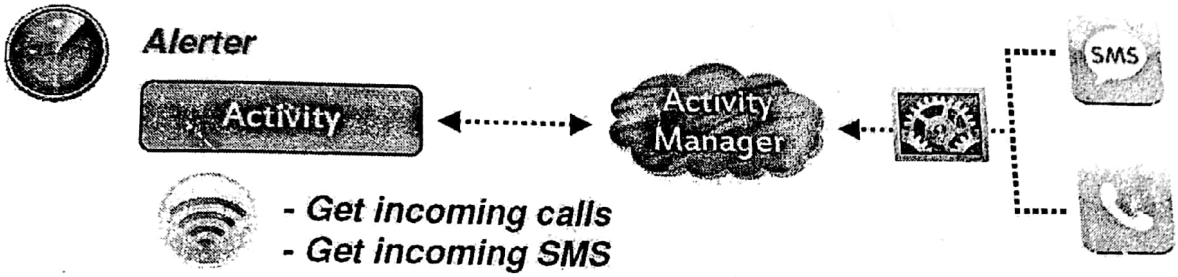
If only one component is found, Android starts this component directly. If several components are identified by the Android system, the user will get a selection dialog and can decide which component should be used for the intent.

What is Receiving and Broadcasting Intents?

⇒ When an application desires to receive and respond to a global event..

⇒ In order to be triggered when an event occurs,, application does not have to be running

⇒ By default,, Android includes some built-in "Intents Receiver"



⇒ Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself These messages are sometime called events or intents.

⇒ 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.

⇒ There are following two important steps to make BroadcastReceiver works for the systen broadcasted intents:

1. Creating the Broadcast Receiver.
2. Registering Broadcast Receiver

⇒ There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

⇒ Android system initiates many system-wide broadcast announcements such as screen turned off and battery is low broadcasts. Also, apps can initiate custom "defined by developers" broadcast. A broadcast is a way to send a message within an application or another application.

⇒ A `BroadcastReceiver` is a component that listens and responds to the initiated broadcasts. In this post, we will see how to send custom broadcast and how to declare

a broadcast receiver programmatically and using Manifest file to listen to this broadcast.

What is Manifest File and common settings ?

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

⇒ It performs some other tasks also:

- 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.

A simple AndroidManifest.xml file looks like this:

```
1.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.    package="com.javatpoint.hello"
3.    android:versionCode="1"
4.    android:versionName="1.0" >
5.    <uses-sdk
6.      android:minSdkVersion="8"
7.      android:targetSdkVersion="15" />
8.
9.    <application
10.     android:icon="@drawable/ic_launcher"
11.     android:label="@string/app_name"
12.     android:theme="@style/AppTheme" >
13.     <activity
14.       android:name=".MainActivity"
15.       android:label="@string/title_activity_main" >
16.       <intent-filter>
17.         <action android:name="android.intent.action.MAIN" />
18.
19.         <category android:name="android.intent.category.LAUNCHER" />
20.       </intent-filter>
21.     </activity>
22.   </application>
23. </manifest>
```

Elements of the AndroidManifest.xml file

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

<manifest>

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

<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.

<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.

<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.

<action>

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

<category>

It adds a category name to an intent-filter.

description:

Declares a security permission that can be used to limit access to specific components or features of this or other applications. See the Permissions section in the introduction, and the Security and Permissions document for more information on how permissions work.

attributes:

`android:description`

A user-readable description of the permission, longer and more informative than the label. It may be displayed to explain the permission to the user — for example, when the user is asked whether to grant the permission to another application.

This attribute must be set as a reference to a string resource; unlike the `label` attribute, it cannot be a raw string.

`android:icon`

A reference to a drawable resource for an icon that represents the permission.

`android:label`

A name for the permission, one that can be displayed to users.

As a convenience, the label can be directly set as a raw string while you're developing the application. However, when the application is ready to be published, it should be set as a reference to a string resource, so that it can be localized like other strings in the user interface.

`android:name`

The name of the permission. This is the name that will be used in code to refer to the permission — for example, in a `<uses-permission>` element and the `permission` attributes of application components.

The name must be unique, so it should use Java-style scoping — for example, `"com.example.project.PERMITTED_ACTION"`.

`android:permissionGroup`

Assigns this permission to a group. The value of this attribute is the name of the group, which must be declared with the `<permission-group>` element in this or another application. If this attribute is not set, the permission does not belong to a group.

`android:protectionLevel`

Characterizes the potential risk implied in the permission and indicates the procedure the system should follow when determining whether or not to grant the permission to an application requesting it. The value can be set to one of the following strings:

Value	Meaning
`"normal"`	The default value. A lower-risk permission that gives requesting applications access to isolated application-

	level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval (though the user always has the option to review these permissions before installing).
`"dangerous"`	A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities.
`"signature"`	A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.
`"signatureOrSystem"`	A permission that the system grants only to applications that are in the Android system image *or* that are signed with the same certificate as the application that declared the permission. Please avoid using this option, as the `signature` protection level should be sufficient for most needs and works regardless of exactly where applications are installed. The `"signatureOrSystem"` permission is used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.

Android manifest file:

When system starts app. Manifest helps system to know all the components that exits in a app.

The components are:

1 User permissions: Internet, Camera, Database, Phone contacts access etc.

2 Declaring Activities, services, content providers .. etc.

3 Declaring API Level

4 Declare Hardware and software used by the App for example camera, Bluetooth or multi touchscreen

In short manifest is the summary of your App.

The list below defines some of the basic terminology of the Android platform.

1. **.apk file :** Android application package file. Each Android application is compiled and packaged in a single file that includes all of the application's code (.dex files), resources, assets, and manifest file. The application package file can have any name but *must* use the `.apk` extension. For example: `myExampleAppname.apk`. For convenience, an application package file is often referred to as an ".apk".
2. **.dex file:**Compiled Android application code file. Android programs are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. .dex files can be created by automatically translating compiled applications written in the Java programming language.
3. **Action:** A description of something that an Intent sender wants done. An action is a string value assigned to an Intent. Action strings can be defined by Android or by a third-party developer. For example, android.intent.action.VIEW for a Web URL, or com.example.rumbler.SHAKE_PHONE for a custom application to vibrate the phone.
4. **Activity:** A single screen in an application, with supporting Java code, derived from the `Activity` class. Most commonly, an activity is visibly represented by a full screen window that can receive and handle UI events and perform complex tasks, because of the Window it uses to render its window. Though an Activity is typically full screen, it can also be floating or transparent.
5. **adb:** Android Debug Bridge, a command-line debugging application included with the SDK. It provides tools to browse the device, copy tools on the device, and forward ports for debugging. If you are developing in Android Studio, adb is integrated into your development environment.
6. **Application:** From a component perspective, an Android application consists of one or more activities, services, listeners, and intent receivers. From a source file perspective, an Android application consists of code, resources, assets, and a single manifest. During compilation, these files are packaged in a single file called an application package file (.apk).
7. **Canvas:** A drawing surface that handles compositing of the actual bits against a Bitmap or Surface object. It has methods for standard computer drawing of bitmaps, lines, circles, rectangles, text, and so on, and is bound to a Bitmap or Surface. Canvas is the simplest, easiest way to draw 2D objects on the screen. However, it does not support hardware acceleration, as OpenGL ES does. The base class is `Canvas`.
8. **Content Provider:** A data-abstraction layer that you can use to safely expose your application's data to other applications. A content provider is built on the `ContentProvider` class, which handles content query strings of a specific format to return data in a specific format. See Content Providers topic for more information.
9. **Dalvik:** The Android platform's virtual machine. The Dalvik VM is an interpreter-only virtual machine that executes files in the Dalvik Executable (.dex) format, a format that is optimized for efficient storage and memory-mappable execution. The virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The VM runs on top of Posix-compliant operating systems, which it relies on for underlying functionality (such as threading and low level memory management). The Dalvik core class library is intended to provide a familiar development base for those used to

programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device.

10. **Dialog:** A floating window that acts as a lightweight form. A dialog can have button controls only and is intended to perform a simple action (such as button choice) and perhaps return a value. A dialog is not intended to persist in the history stack, contain complex layout, or perform complex actions. Android provides a default simple dialog for you with optional buttons, though you can define your own dialog layout. The base class for dialogs is `Dialog`.
11. **Drawable :** A compiled visual resource that can be used as a background, title, or other part of the screen. A drawable is typically loaded into another UI element, for example as a background image. A drawable is not able to receive events, but does assign various other properties such as "state" and scheduling, to enable subclasses such as animation objects or image libraries. Many drawable objects are loaded from drawable resource files — xml or bitmap files that describe the image. Drawable resources are compiled into subclasses of `android.graphics.drawable`. For more information about drawables and other resources, see Resources.
12. **Intent:** An message object that you can use to launch or communicate with other applications/activities asynchronously. An Intent object is an instance of `Intent`. It includes several criteria fields that you can supply, to determine what application/activity receives the Intent and what the receiver does when handling the Intent. Available criteria include include the desired action, a category, a data string, the MIME type of the data, a handling class, and others. An application sends an Intent to the Android system, rather than sending it directly to another application/activity. The application can send the Intent to a single target application or it can send it as a broadcast, which can in turn be handled by multiple applications sequentially. The Android system is responsible for resolving the best-available receiver for each Intent, based on the criteria supplied in the Intent and the Intent Filters defined by other applications.
13. **Intent Filter:** A filter object that an application declares in its manifest file, to tell the system what types of Intents each of its components is willing to accept and with what criteria. Through an intent filter, an application can express interest in specific data types, Intent actions, URI formats, and so on. When resolving an Intent, the system evaluates all of the available intent filters in all applications and passes the Intent to the application/activity that best matches the Intent and criteria.
14. **Broadcast Receiver:** An application class that listens for Intents that are broadcast, rather than being sent to a single target application/activity. The system delivers a broadcast Intent to all interested broadcast receivers, which handle the Intent sequentially.
15. **Layout Resource :**An XML file that describes the layout of an Activity screen.
16. **Manifest File:** An XML file that each application must define, to describe the application's package name, version, components (activities, intent filters, services), imported libraries, and describes the various activities, and so on.
17. **Nine-patch / 9-patch / Ninepatch image:** A resizeable bitmap resource that can be used for backgrounds or other images on the device.

18. **OpenGL ES :**Android provides OpenGL ES libraries that you can use for fast, complex 3D images. It is harder to use than a Canvas object, but better for 3D objects. The `android.opengl` and`javax.microedition.khronos.opengles` packages expose OpenGL ES functionality.
19. **Resources:** Nonprogrammatic application components that are external to the compiled application code, but which can be loaded from application code using a well-known reference format. Android supports a variety of resource types, but a typical application's resources would consist of UI strings, UI layout components, graphics or other media files, and so on. An application uses resources to efficiently support localization and varied device profiles and states. For example, an application would include a separate set of resources for each supported local or device type, and it could include layout resources that are specific to the current screen orientation (landscape or portrait).
20. **Service:** An object of class `Service` that runs in the background (without any UI presence) to perform various persistent actions, such as playing music or monitoring network activity.
21. **Surface:** An object of type `Surface` representing a block of memory that gets composited to the screen. A Surface holds a Canvas object for drawing, and provides various helper methods to draw layers and resize the surface. You should not use this class directly; use `SurfaceView` instead.
22. **Surface View:** A View object that wraps a Surface for drawing, and exposes methods to specify its size and format dynamically. A SurfaceView provides a way to draw independently of the UI thread for resource-intensive operations (such as games or camera previews), but it uses extra memory as a result. SurfaceView supports both Canvas and OpenGL ES graphics. The base class is `SurfaceView`.
23. **Theme:** A set of properties (text size, background color, and so on) bundled together to define various default display settings. Android provides a few standard themes, listed in `R.style` (starting with "Theme_").
24. **URIs in Android:** Android uses URI (uniform resource identifier) strings as the basis for requesting data in a content provider (such as to retrieve a list of contacts) and for requesting actions in an Intent (such as opening a Web page in a browser). The URI scheme and format is specialized according to the type of use, and an application can handle specific URI schemes and strings in any way it wants. Some URI schemes are reserved by system components. For example, requests for data from a content provider must use the `content://`. In an Intent, a URI using an `http://` scheme will be handled by the browser.
25. **View:**An object that draws to a rectangular area on the screen and handles click, keystroke, and other interaction events. A view is a base class for most layout components of an Activity or Dialog screen (text boxes, windows, and so on). It receives calls from its parent object (see ViewGroup) to draw itself, and informs its parent object about where and how big it would like to be (which may or may not be respected by the parent).

26. **View Hierarchy:** An arrangement of View and ViewGroup objects that defines the user interface for each component of an app. The hierarchy consists of view groups that contain one or more child views or view groups. You can obtain a visual representation of a view hierarchy for debugging and optimization by using the Hierarchy Viewer that is supplied with the Android SDK.
27. **ViewGroup:**A container object that groups a set of child views. The view group is responsible for deciding where child views are positioned and how large they can be, as well as for calling each to draw itself when appropriate. Some view groups are invisible and are for layout only, while others have an intrinsic UI (for instance, a scrolling list box). View groups are all in the `widget` package, but extend `ViewGroup`.
28. **Widget:** One of a set of fully implemented View subclasses that render form elements and other UI components, such as a text box or popup menu. Because a widget is fully implemented, it handles measuring and drawing itself and responding to screen events. Widgets are all in the `android.widget` package.
29. **Window:** In an Android application, an object derived from the abstract class `Window` that specifies the elements of a generic window, such as the look and feel (title bar text, location and content of menus, and so on). Dialog and Activity use an implementation of this class to render a window. You do not need to implement this class or use windows in your application.

Android Activity Lifecycle

Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

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

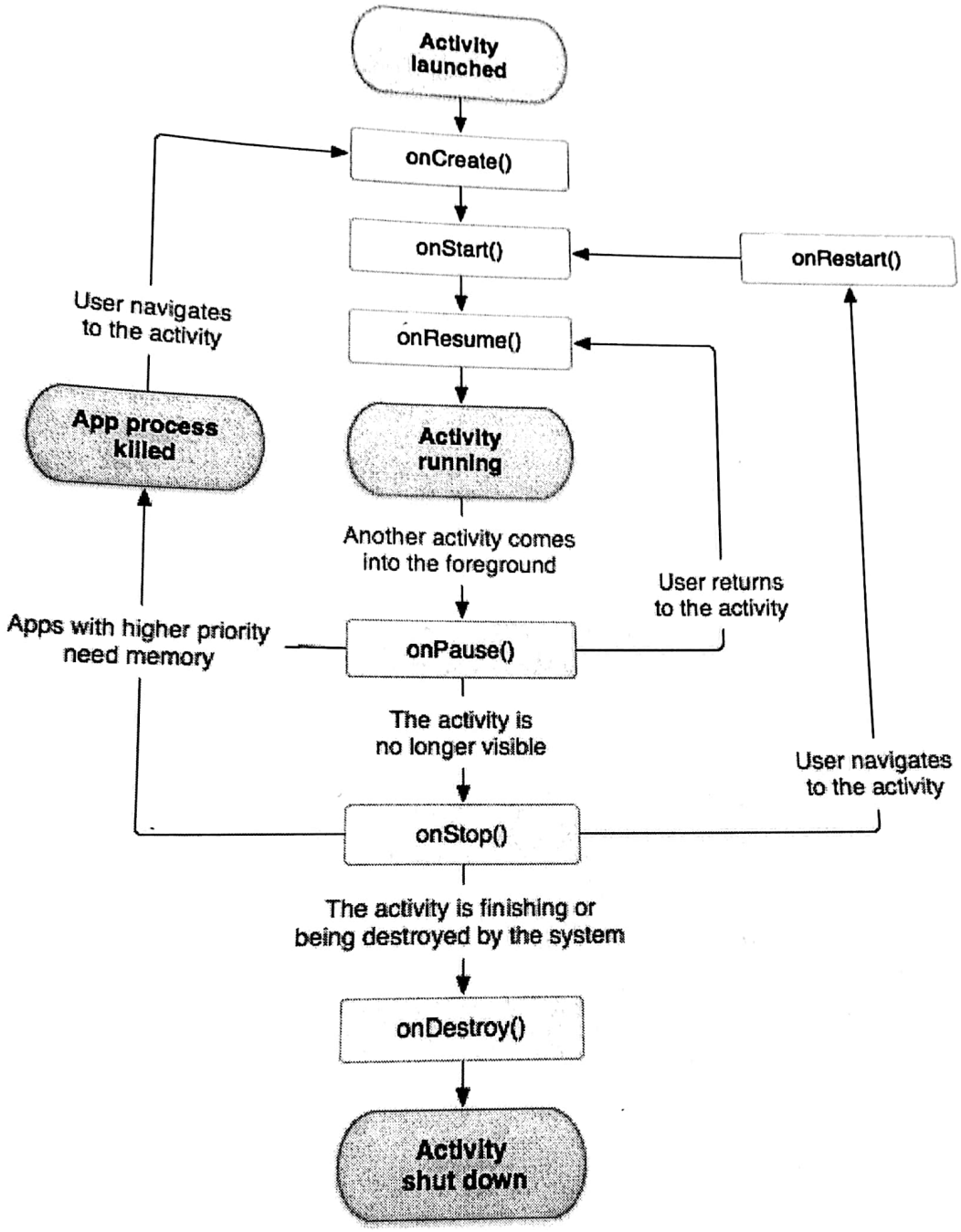
By the help of activity, you can place all your UI components or widgets in a single screen.

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

Android Activity Lifecycle methods

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

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.



File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.co
   m/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent"
7.     tools:context="example.javatpoint.com.activitylifecycle.MainActivity">
```

```
8.
9.     <TextView
10.        android:layout_width="wrap_content"
11.        android:layout_height="wrap_content"
12.        android:text="Hello World!"
13.        app:layout_constraintBottom_toBottomOf="parent"
14.        app:layout_constraintLeft_toLeftOf="parent"
15.        app:layout_constraintRight_toRightOf="parent"
16.        app:layout_constraintTop_toTopOf="parent" />
17.
18. </android.support.constraint.ConstraintLayout>
```

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.

File: MainActivity.java

```
1.  package example.javatpoint.com.activitylifecycle;
2.
3.  import android.app.Activity;
4.  import android.os.Bundle;
5.  import android.util.Log;
6.
7.  public class MainActivity extends Activity {
8.
9.      @Override
10.     protected void onCreate(Bundle savedInstanceState) {
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.activity_main);
13.         Log.d("lifecycle","onCreate invoked");
14.     }
15.     @Override
16.     protected void onStart() {
17.         super.onStart();
18.         Log.d("lifecycle","onStart invoked");
19.     }
20.     @Override
21.     protected void onResume() {
22.         super.onResume();
23.         Log.d("lifecycle","onResume invoked");
24.     }
25.     @Override
26.     protected void onPause() {
27.         super.onPause();
```

```
28.         Log.d("lifecycle","onPause invoked");
29.     }
30.     @Override
31.     protected void onStop() {
32.         super.onStop();
33.         Log.d("lifecycle","onStop invoked");
34.     }
35.     @Override
36.     protected void onRestart() {
37.         super.onRestart();
38.         Log.d("lifecycle","onRestart invoked");
39.     }
40.     @Override
41.     protected void onDestroy() {
42.         super.onDestroy();
43.         Log.d("lifecycle","onDestroy invoked");
44.     }
45. }
```