

MODULE – 2:

GETTING STARTED WITH ANDROID PROGRAMMING

Syllabus: What is Android? Obtaining the required tools, Anatomy of an Android Application, Components of Android Applications: Activities, Fragments, Utilizing the Action Bar

2.1 WHAT IS ANDROID?

2.1.1 Introduction

Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work. The main advantage of adopting Android:

- It offers a unified approach to application development.
- Developers need only develop for Android, and their applications can run on numerous different devices based on Android OS.

2.1.2 Features of Android

As Android is open source and freely available to manufacturers for customization, there are no fixed hardware and software configurations. However, Android itself supports the following features:

- **Storage** : Uses SQLite, a lightweight relational database, for data storage.
- **Connectivity** : Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), WiFi, LTE, and WiMAX.
- **Messaging**: Supports both SMS and MMS.
- **Web browser**: Based on the open-source WebKit, together with Chrome's V8 JavaScript engine
- **Media support**: Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
- **Hardware support**: Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
- **Multi-touch**: Supports multi-touch screens
- **Multi-tasking**: Supports multi-tasking applications
- **Flash support**: Android 2.3 supports Flash 10.1.
- **Tethering**: Supports sharing of Internet connections as a wired/wireless hotspot

2.1.3 Architecture of Android

The Android OS is roughly divided into five sections in four main layers as shown in Figure 2.1.

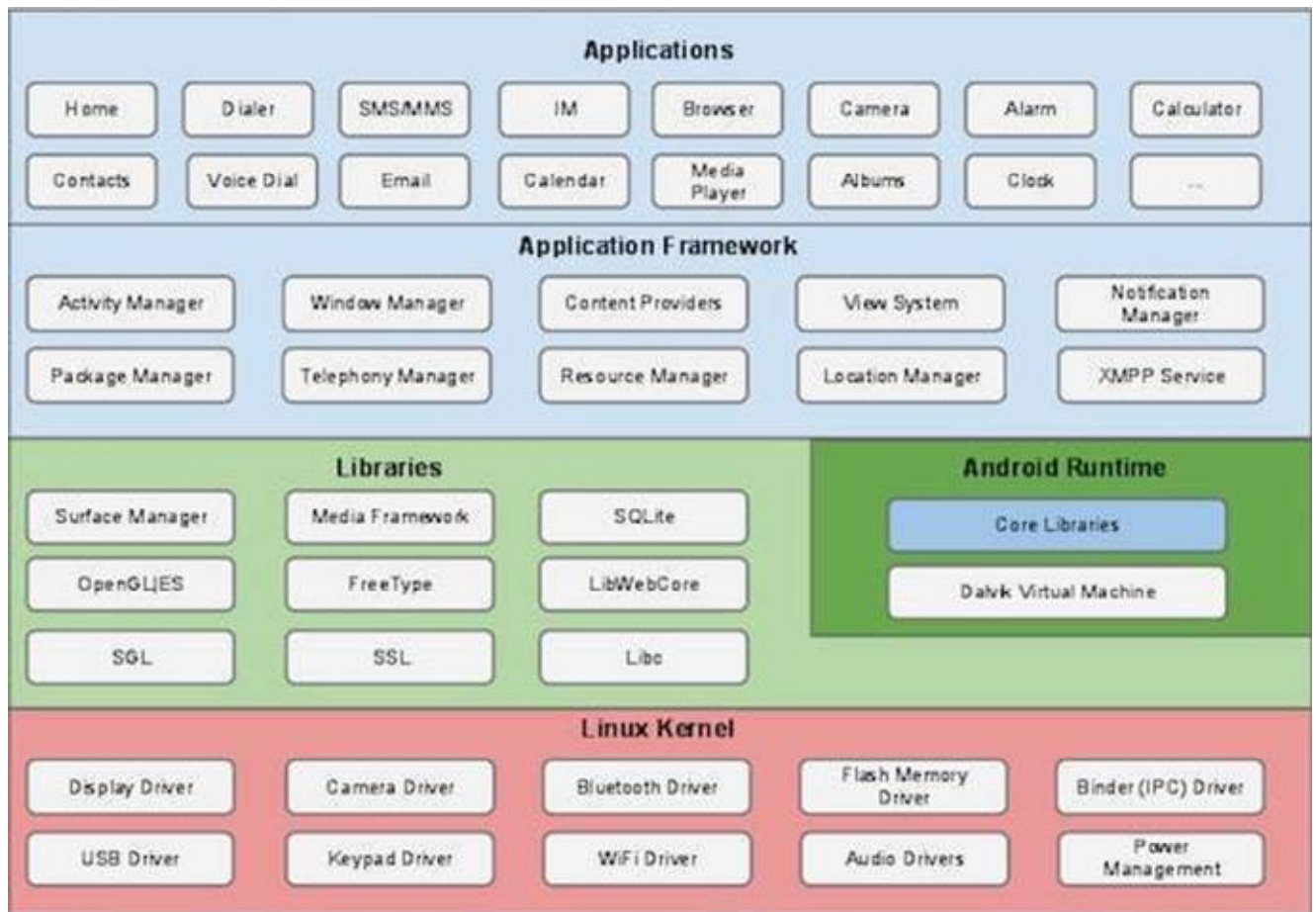


Figure 2.1 Various Layers of Android OS

- **Linux kernel:** This is the kernel on which Android is based. This layer contains all the low level device drivers for the various hardware components of an Android device.
- **Libraries:**
 - These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage.
 - The WebKit library provides functionalities for web browsing.
- **Android runtime :**
 - At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language.
 - The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executables).
 - Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

- **Application Framework:** Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.
- **Applications:**
 - At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market.
 - Any applications that you write are located at this layer.

2.1.4 Android Devices in Market

Android devices come in all shapes and sizes. Few of the Android OS can be seen powering the following types of devices: Smartphones, Tablets, E-reader devices, Netbooks, MP4 players, Internet TVs etc.

2.1.5 The Android Market

One of the main factors determining the success of a smartphone platform is the applications that support it. Applications play a very vital role in determining whether a new platform swims or sinks. Making these applications accessible to the general user is extremely important. In August 2008, Google announced the Android Market, an online application store for Android devices, and made it available to users in October 2008. Using the Market application that is preinstalled on their Android device, users can simply download third-party applications directly onto their devices. Both paid and free applications are supported on the Android Market, though paid applications are available only to users in certain countries due to legal issues. **Google Playstore** is the current name of Android Market!!

2.2 OBTAINING THE REQUIRED TOOLS

To develop an android application, we need to install certain softwares. It involves installing Eclipse, Android SDK, Android Development Tools (ADT) and creating Android Virtual Devices (AVD).

Note that, The Android SDK makes use of the Java SE Development Kit (JDK). Hence, before installing Eclipse etc, install JDK from www.oracle.com/technetwork/java/javase/downloads/index.html

2.2.1 Eclipse

The first step towards developing any applications is obtaining the integrated development environment (IDE). For Android, the recommended IDE is Eclipse, a multi-language software development environment featuring an extensible plug-in system. It can be used to develop various types of applications, using languages such as Java, Ada, C, C++, COBOL, Python, etc. For Android development, you should download the Eclipse IDE for Java EE Developers (www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr1). Once the Eclipse IDE is downloaded, unzip its content (the eclipse folder) into a folder, say C:\Android\.

2.2.2 Android SDK

The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials. You can download the Android SDK from <http://developer.android.com/sdk/index.html>. Once the SDK is downloaded, unzip its content (the android-sdk-windows folder) into the C:\Android\ folder, or whatever name you have given to the folder you just created.

2.2.3 Android Development Tools (ADT)

The ADT plug-in for Eclipse is an extension to the Eclipse IDE that supports the creation and debugging of Android applications. Using the ADT, you will be able to do the following in Eclipse:

- Create new Android application projects.
- Access the tools for accessing your Android emulators and devices.
- Compile and debug Android applications.
- Export Android applications into Android Packages (APK).
- Create digital certificates for code-signing your APK.

To install the ADT, first launch Eclipse by double-clicking on the eclipse.exe file located in the eclipse folder.

2.2.4 Creating Android Virtual Devices (AVD)

An AVD is an emulator instance that enables you to model an actual device. Each AVD consists of a hardware profile, a mapping to a system image, as well as emulated storage, such as a secure digital (SD) card. One can create many AVDs in order to test your applications with several different configurations. This testing is important to confirm the behavior of your application when it is run on different devices with varying capabilities. To create an AVD, go to Windows Menu and choose “Android Virtual Device Manager”. Then give appropriate name, device, memory required etc. for the application. Once the new AVD is created, it can be used for different applications further.

2.3 Creating Your First Android Applications

Following are the steps involved in creating any Android application:

1. Using Eclipse, create a new project by selecting File ⇨ New ⇨ Android Application Project.
2. Name the Android project suitably, say *HelloWorld*.
3. In the Package Explorer (located on the left of the Eclipse IDE), expand the HelloWorld project by clicking on the various arrows displayed to the left of each item in the project. In the **res/layout** folder, double-click the **activity_main.xml** file. The **activity_main.xml** file defines the user interface (UI) of your application. The default view is the **Layout view**, which lays out the activity graphically. To modify the UI, click the **activity_main.xml** tab located at the bottom.
4. Add the following code in bold to the **activity_main.xml** file.

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

```
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent" >

<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello" />

</LinearLayout>
```

5. To save the changes made to your project, press Ctrl+s.
6. You are now ready to test your application on the Android Emulator. Select the project name in Eclipse and press F11. You will be asked to select a way to debug the application. Select required Android Application and click OK.
7. The Android Emulator will now be started (if the emulator is locked, you need to slide the unlock button to unlock it first).
8. Click the Home button (the house icon in the lower-left corner above the keyboard) so that it now shows the Home screen.
9. Click the application Launcher icon to display the list of applications installed on the device. Note that the HelloWorld application is now installed in the application launcher.

(Note: In Step 6, include the steps of creating AVD).

In Android, an Activity is a window that contains the user interface of your applications. An application can have zero or more activities; in this example, the application contains one activity: **MainActivity**. This **MainActivity** is the entry point of the application, which is displayed when the application is started. When you debug the application on the Android Emulator, the application is automatically installed on the emulator.

2.4 Anatomy of Android Application

The various folders and their files are as follows:

- **src** — Contains the file, **MainActivity.java**. It is the source file for your activity. You will write the code for your application in this file.
- **Android 4.4.2** — This item contains one file, **android.jar**, which contains all the class libraries needed for an Android application.
- **gen** — Contains the **R.java** file, a compiler-generated file that references all the resources found in your project. **You should not modify this file.**
- **assets** — This folder contains all the assets used by your application, such as HTML, text files, databases, etc.
- **res** — This folder contains all the resources used in your application. It also contains a few other subfolders:
 - **drawable - <resolution>**: All the image files to be used by the Android application must be stored here.

- **layout** - contains **activity_main.xml** file, which is the GUI of the application.
- **values** - contains files like **strings.xml**, **styles.xml** that are needed for storing the string variables used in the applications, creating style-sheets etc.
- **AndroidManifest.xml** — This is the manifest file for your Android application. Here you specify the permissions needed by your application, as well as other features (such as intent-filters, receivers, etc.).

Details of some of the important files are given hereunder:

- **strings.xml File:** The activity_main.xml file defines the user interface for your activity. Observe the following in bold:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Hello World!</string>
<string name="app_name">HelloWorld</string>
</resources>
```

It is recommended that you store all the string constants in your application in this **strings.xml** file and reference these strings using the **@string** identifier. That way, if you ever need to localize your application to another language, all you need to do is replace the strings stored in the **strings.xml** file with the targeted language and recompile your application.

- **AndroidManifest.xml File:** This file contains detailed information about the application. Observe the code in this file:

```
<?xml version="1.0" encoding="utf-8"?>
<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="19"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
```

```

    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

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

```

Key points about this file are as below :

- It defines the package name of the application as ***net.learn2develop.HelloWorld***.
 - The version code of the application is 1. This value is used to identify the version number of your application. It can be used to programmatically determine whether an application needs to be upgraded.
 - The version name of the application is 1.0. This string value is mainly used for display to the user.
 - The application uses the image named ***ic_launcher.png*** located in the ***drawable*** folder.
 - The name of this application is the string named ***app_name*** defined in the ***strings.xml*** file.
 - There is one activity in the application represented by the ***MainActivity.java*** file. The label displayed for this activity is the same as the application name.
 - Within the definition for this activity, there is an element named ***<intent-filter>***:
 - 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 named ***android.intent.category.LAUNCHER*** to indicate that the application can be launched from the device's Launcher icon.
 - Finally, the ***android:minSdkVersion*** attribute of the ***<uses-sdk>*** element specifies the minimum version of the OS on which the application will run.
- **R.java File:** As you add more files and folders to your project, Eclipse will automatically generate the content of **R.java**, which at the moment contains the following:

```

package net.learn2develop.HelloWorld;

public final class R {
    public static final class attr {
    }
}

```

```
public static final class drawable {
    public static final int icon=0x7f020000;
}
public static final class layout {
    public static final int main=0x7f030000;
}
public static final class string {
    public static final int app_name=0x7f040001;
    public static final int hello=0x7f040000;
}
}
```

You are not supposed to modify the content of the R.java file; Eclipse automatically generates the content for you when you modify your project.

- **MainActivity.java File:** The code that connects the activity to the UI (activity_main.xml) is the setContentView() method, which is in the MainActivity.java file:

```
package net.learn2develop.HelloWorld;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Here, **R.layout.main** refers to the **activity_main.xml** file located in the **res/layout** folder. As you add additional XML files to the **res/layout** folder, the filenames will automatically be generated in the **R.java** file. The **onCreate()** method is one of many methods that are fired when an activity is loaded.

2.5 Components of Android Application

Android applications consist of loosely coupled components, bound by the application manifest that describes each component and how they interact. The manifest is also used to specify the application's metadata, its hardware and platform requirements, external libraries, and required permissions. The following components comprise the building blocks for all your Android applications:

- **Activities** — It is the presentation layer of application. The UI of your application is built around one or more extensions of the Activity class. Activities use Fragments and Views to layout and display information, and to respond to user actions. Compared to desktop development, Activities are equivalent to Forms.
- **Services** — These are the invisible workers of your application. Service components run without a UI, updating your data sources and Activities, triggering Notifications, and broadcasting Intents. They are used to perform long running tasks, or those that require no user interaction (such as network lookups or tasks that need to continue even when your application's Activities aren't active or visible.)
- **Content Providers** — Shareable persistent data storage. Content Providers manage and persist application data and typically interact with SQL databases. They are also the preferred means to share data across application boundaries. You can configure your application's Content Providers to allow access from other applications, and you can access the Content Providers exposed by others. Android devices include several native Content Providers that expose useful databases such as the media store and contacts.
- **Intents** — A powerful inter-application message-passing framework. Intents are used extensively throughout Android. You can use Intents to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service, or Broadcast Receiver, or to request an action be performed on a particular piece of data.
- **Broadcast Receivers** — Intent listeners. Broadcast Receivers enable your application to listen for Intents that match the criteria you specify. Broadcast Receivers start your application to react to any received Intent, making them perfect for creating event-driven applications.
- **Widgets** — Visual application components that are typically added to the device home screen. A special variation of a Broadcast Receiver, widgets enable you to create dynamic, interactive application components for users to embed on their home screens.
- **Notifications** — Notifications enable you to alert users to application events without stealing focus or interrupting their current Activity. They're the preferred technique for getting a user's attention when your application is not visible or active, particularly from within a Service or Broadcast Receiver. For example, when a device receives a text message or an email, the messaging and Gmail applications use Notifications to alert you by flashing lights, playing sounds, displaying icons, and scrolling a text summary. You can trigger these notifications from your applications

By decoupling the dependencies between application components, you can share and use individual Content Providers, Services, and even Activities with other applications — both your own and those of third parties.

2.6 Activities

An activity is a window that contains the user interface of your applications. An application can have zero or more activities. Typically, applications have one or more activities, and the main aim of an activity is to interact with the user. From the moment an activity appears

on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's life cycle. One has to understand the life cycle of an activity to ensure that the application works correctly. To create an activity, you create a Java class that extends the Activity base class:

```
package net.learn2develop.Activities;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Your activity class would then load its UI component using the XML file defined in your res/layout folder. In this example, you would load the UI from the activity_main.xml file:

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

Every activity you have in your application must be declared in your AndroidManifest.xml file, like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.program4"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

```

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

The Activity base class defines a series of events that governs the life cycle of an activity. Figure 2.2 shows the life cycle of an activity and the various stages it goes through — from when the activity is started until it ends.

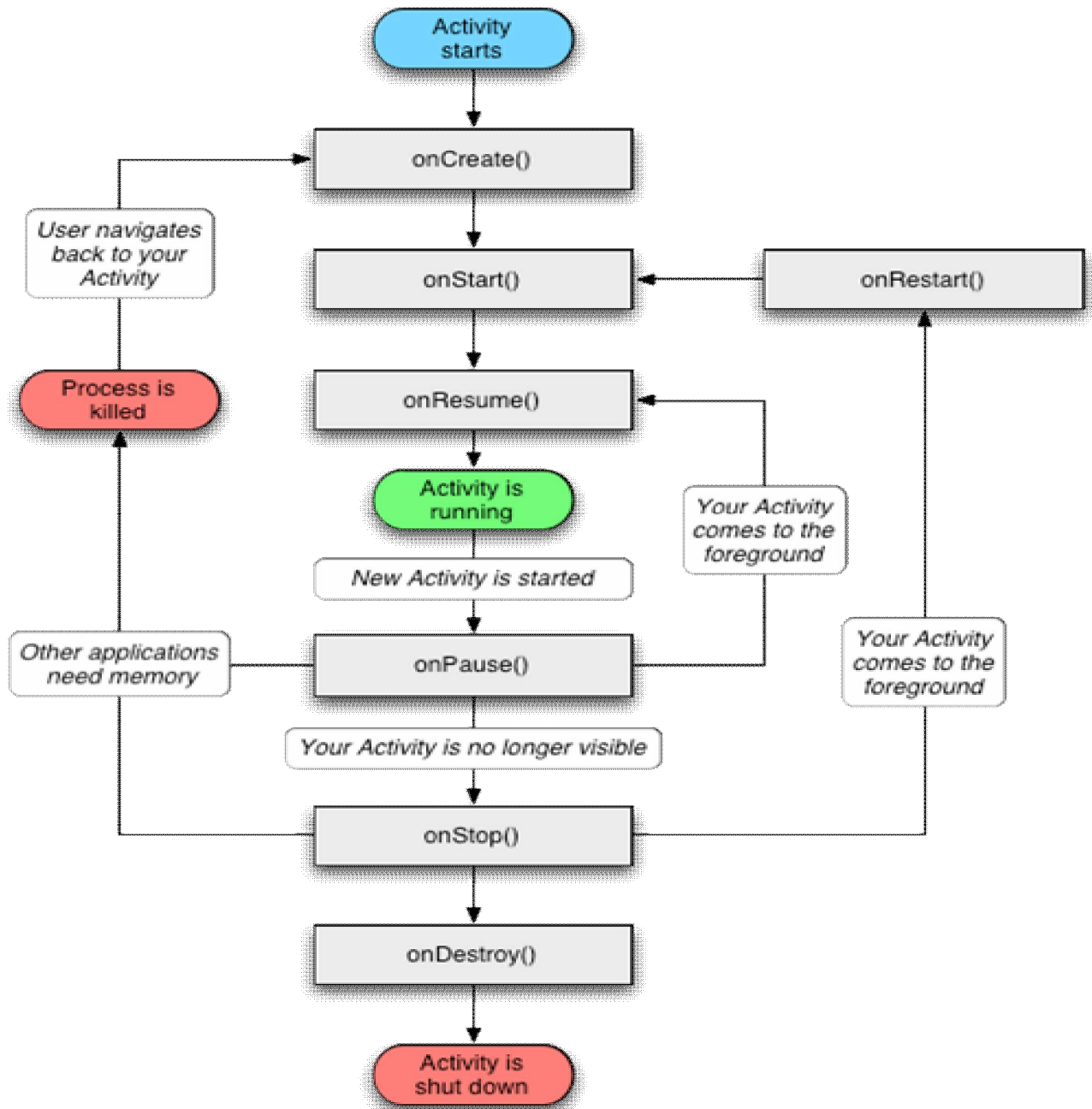


Figure 2.2 Life Cycle of Activity

The Activity class defines the following events:

- onCreate() — Called when the activity is first created
- onStart() — Called when the activity becomes visible to the user
- onResume() — Called when the activity starts interacting with the user
- onPause() — Called when the current activity is being paused and the previous activity is being resumed
- onStop() — Called when the activity is no longer visible to the user
- onDestroy() — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- onRestart() — Called when the activity has been stopped and is restarting again

By default, the activity created for you contains the onCreate() event. This event handler contains the code that helps to display the UI elements of your screen.

2.6.1 Understanding Life Cycle of an Activity using an Example

Perform the following steps to understand the working of life cycle of an activity:

- a) Using Eclipse, create a new Android project and name it.
- b) In the MainActivity.java file, add the following statements (**Do not remove existing statements**):

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity
{
    String tag = "Events";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(tag, "In the onCreate() event");
    }

    public void onStart()
    {
        super.onStart();
        Log.d(tag, "In the onStart() event");
    }
    public void onRestart()
    {
        super.onRestart();
        Log.d(tag, "In the onRestart() event");
    }
}
```

```
public void onResume()
{
    super.onResume();
    Log.d(tag, "In the onResume() event");
}
public void onPause()
{
    super.onPause();
    Log.d(tag, "In the onPause() event");
}
public void onStop()
{
    super.onStop();
    Log.d(tag, "In the onStop() event");
}
public void onDestroy()
{
    super.onDestroy();
    Log.d(tag, "In the onDestroy() event");
}

/* keep the remaining code as it is */
}
```

- c) Start the AVD and Press F11 to debug the application on the Android Emulator.
- d) When the activity is first loaded, you should see the following in the **LogCat** window
- ```
12-28 13:45:28.115: DEBUG/Events(334): In the onCreate() event
12-28 13:45:28.115: DEBUG/Events(334): In the onStart() event
12-28 13:45:28.115: DEBUG/Events(334): In the onResume() event
```
- e) When you now press the back button on the Android Emulator, observe that the following is printed:
- ```
12-28 13:59:46.266: DEBUG/Events(334): In the onPause() event
12-28 13:59:46.806: DEBUG/Events(334): In the onStop() event
12-28 13:59:46.806: DEBUG/Events(334): In the onDestroy() event
```
- f) Click the Home button and hold it there. Click the Activities icon and observe the following:
- ```
12-28 14:00:54.115: DEBUG/Events(334): In the onCreate() event
12-28 14:00:54.156: DEBUG/Events(334): In the onStart() event
12-28 14:00:54.156: DEBUG/Events(334): In the onResume() event
```
- g) Press the Phone button on the Android Emulator so that the activity is pushed to the background. Observe the output in the LogCat window:
- ```
12-28 14:01:16.515: DEBUG/Events(334): In the onPause() event
12-28 14:01:17.135: DEBUG/Events(334): In the onStop() event
```

- h) Notice that the `onDestroy()` event is not called, indicating that the activity is still in memory. Exit the phone dialer by pressing the Back button. The activity is now visible again. Observe the output in the LogCat window:

```
12-28 14:02:17.255: DEBUG/Events(334): In the onRestart() event
12-28 14:02:17.255: DEBUG/Events(334): In the onStart() event
12-28 14:02:17.255: DEBUG/Events(334): In the onResume() event
```

The `onRestart()` event is now fired, followed by the `onStart()` and `onResume()` events.

As we can see from this simple experiment, an activity is destroyed when you press the Back button. This is crucial to know, as whatever state the activity is currently in will be lost; hence, you need to write additional code in your activity to preserve its state when it is destroyed. At this point, note that the `onPause()` event is called in both scenarios — when an activity is sent to the background, as well as when it is killed when the user presses the Back button. When an activity is started, the `onStart()` and `onResume()` events are always called, regardless of whether the activity is restored from the background or newly created.

2.7 Fragments

Fragments enable you to divide your Activities into fully encapsulated reusable components, each with its own lifecycle and UI. The primary advantage of Fragments is the ease with which you can create dynamic and flexible UI designs that can be adapted to various screen sizes.

Each Fragment is an independent module that is tightly bound to the Activity into which it is placed. Fragments can be reused within multiple activities, as well as laid out in a variety of combinations to suit multi-pane tablet UIs and added to, removed from, and exchanged within a running Activity to help build dynamic UIs. Fragments provide a way to present a consistent UI optimized for a wide variety of Android device types, screen sizes, and device densities.

Although it is not necessary to divide your Activities (and their corresponding layouts) into Fragments, doing so will drastically improve the flexibility of your UI and make it easier for you to adapt your user experience for new device configurations.

2.7.1 Creating New Fragment

Extend the Fragment class to create a new Fragment, (optionally) defining the UI and implementing the functionality it encapsulates. In most circumstances you'll want to assign a UI to your Fragment. It is possible to create a Fragment that doesn't include a UI but instead provides background behavior for an Activity. If your Fragment does require a UI, override the `onCreateView` handler to inflate and return the required View hierarchy, as shown in the Fragment skeleton code shown below:

```
package com.paad.fragments;
import android.app.Fragment;
```

```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class MySkeletonFragment extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState)
    {
        // Create, or inflate the Fragment's UI, and return it.
        // If this Fragment has no UI then return null.
        return inflater.inflate(R.layout.my_fragment, container, false);
    }
}
```

Unlike Activities, Fragments don't need to be registered in your manifest. This is because Fragments can exist only when embedded into an Activity, with their lifecycles dependent on that of the Activity to which they've been added.

2.7.2 Lifecycle of Fragments

The lifecycle events of a Fragment reflect those of its parent Activity. But, when the container Activity is in its active and resumed state by adding or removing a fragment, it will affect the lifecycle independently. Figure 2.3 shows various events in lifecycle of fragments.

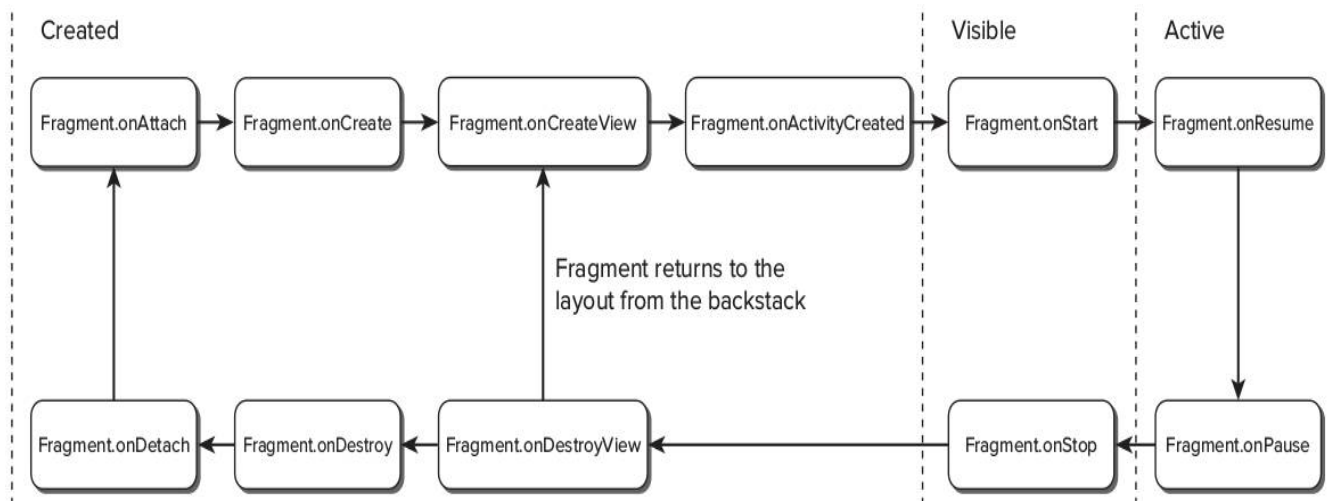


Figure 2.3 Lifecycle of Fragments

Most of the Fragment lifecycle events correspond to their equivalents in the Activity class. Those that remain are specific to Fragments and the way in which they're inserted into their parent Activity are explained here:

- **Attaching and Detaching Fragments from the Parent Activity:** The full lifetime of your Fragment begins when it is bound to its parent Activity and ends when it has been detached. These events are represented by the calls to `onAttach` and `onDetach`, respectively. As with any handler called after a Fragment/Activity has become paused, it's possible that `onDetach` will not be called if the parent Activity's process is terminated without completing its full lifecycle. The `onAttach` event is triggered before the Fragment's UI has been created, before the Fragment itself or its parent Activity have finished their initialization. Typically, the `onAttach` event is used to gain a reference to the parent Activity in preparation for further initialization tasks.
- **Creating and Destroying Fragments:** The created lifetime of your Fragment occurs between the first call to `onCreate` and the final call to `onDestroy`. As it's not uncommon for an Activity's process to be terminated without the corresponding `onDestroy` method being called, so a Fragment can't rely on its `onDestroy` handler being triggered. As with Activities, you should use the `onCreate` method to initialize your Fragment. It's good practice to create any class scoped objects here to ensure they're created only once in the Fragment's lifetime.
- **Creating and Destroying User Interfaces:** A Fragment's UI is initialized (and destroyed) within a new set of event handlers: `onCreateView` and `onDestroyView`, respectively. Use the `onCreateView` method to initialize your Fragment:
 - Inflate the UI,
 - get references (and bind data to) the Views it contains,
 - and then create any required Services and Timers.

Once you have inflated your View hierarchy, it should be returned from this handler:

```
return inflater.inflate(R.layout.my_fragment, container, false);
```

If your Fragment needs to interact with the UI of its parent Activity, wait until the `onActivityCreated` event has been triggered. This signifies that the containing Activity has completed its initialization and its UI has been fully constructed.

2.7.3 Introducing Fragment Manager

Each Activity includes a Fragment Manager to manage the Fragments it contains. You can access the Fragment Manager using the `getFragmentManager` method:

```
FragmentManager fragmentManager = getFragmentManager();
```

The Fragment Manager provides the methods used to access the Fragments currently added to the Activity, and to perform Fragment Transaction to add, remove, and replace Fragments.

2.7.4 Adding Fragments to Activities

The simplest way to add a Fragment to an Activity is by including it within the Activity's layout using the fragment tag, as shown below –


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.paad.weatherstation.MyListFragment"
        android:id="@+id/my_list_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
    />
    <fragment android:name="com.paad.weatherstation.DetailsFragment"
        android:id="@+id/details_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="3"
    />
</LinearLayout>
```

Once the Fragment has been inflated, it becomes a View Group, laying out and managing its UI within the Activity. This technique works well when you use Fragments to define a set of static layouts based on various screen sizes. If you plan to dynamically modify your layouts by adding, removing, and replacing Fragments at run time, a better approach is to create layouts that use container Views into which Fragments can be placed at runtime, based on the current application state.

2.8 Utilizing the Action Bar

The Action Bar component was introduced in Android 3.0 (API level 11). Few facts about action bar:

- It is a navigation panel that replaces the title bar at the top of every Activity and that formalizes a common Android design pattern.
- It is possible to hide the Action Bar, but best practice is to keep it and customize it to suit the style and navigation requirements of your application.
- A simple action bar is as shown in Figure 2.4.



Figure 2.4 Action Bar

- The Action Bar can be added to each Activity within your application and is designed to provide a consistent UI between applications and within a particular application's Activities.
- The Action Bar provides a consistent framework for providing branding, navigation, and surfacing the key actions to be performed within an Activity.

- The Action Bar is enabled by default in any Activity that uses the (default) Theme.Holo theme and whose application has a target (or minimum) SDK version of 11 or higher.

Following code snippet shows how to enable the Action Bar by setting the target SDK to Android 4.0.3 (API level 15) and not modifying the default theme.

```
<uses-sdk android:targetSdkVersion="15" /> (inside manifest file)
```

To toggle the visibility of the Action Bar at run time, you can use its show and hide methods:

```
ActionBar actionBar = getActionBar();  
// Hide the Action Bar  
actionBar.hide();  
// Show the Action Bar  
actionBar.show();
```

Alternatively, you can apply a theme that doesn't include the Action Bar, such as the Theme.Holo.NoActionBar theme, as shown below:

```
<activity  
android:name=".MyNonActionBarActivity"  
android:theme="@android:style/Theme.Holo.NoActionBar">  
                                                    (inside manifest file)
```

You can create or customize your own theme that removes the Action Bar by setting the android:windowActionBar style property to false:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
<style name="NoActionBar" parent="@style/ActivityTheme">  
<item name="android:windowActionBar">false</item>  
</style>  
</resources>
```

When you apply a theme that excludes the Action Bar from an Activity, you can't programmatically display it at run time. A call to `getActionBar` will return null.

Question Bank:

1. Explain the features of Android.
2. Explain the architecture of Android (OR – Explain the various layers in android OS.)
3. Discuss the tools required for developing Android application.
4. What do you mean by an AVD? How do you create it?
5. Explain the various steps involved in creating an Android application.
6. Briefly discuss the anatomy of an android application.

7. Explain the content and working of Manifest file.
8. Bring out the significance of R.java file.
9. Discuss the usage of MainActivity.java file with suitable code.
10. Explain the components of android application.
11. What is an Activity? Explain lifecycle of an activity with a neat diagram.
12. Discuss lifecycle of an activity with an example.
13. Define a fragment. What are the events associated with a fragment? Explain with a neat diagram.
14. What is a fragment Manager? How do you add a fragment to an Activity? Explain with suitable code.
15. Explain action bar with suitable code segment.