Unit -II

Application Structure (**AndroidManifest.xml** , **uses-permission & uses-sdk** , Resources & R.java , Assets & Layouts &Drawable Resources , **Activities and Activity lifecycle.**

Android uses packages not only to arrange the code in an application but to manage the application themselves.

The above diagram shows the basic building blocks of an Android application. Android application in Eclipse or in any development tool have a pre-defined structure with code and resource organized into a number of folders

Every Android project contains several folders, like:

Folder Name	Description
src	The 'src' stands for **Source Code.** It contains the Java Source files.
gen	The 'gen' stands for **Generated Java Library.** This library is for Android internal use only.
Android 2.2	The Android Framework Library is stored here.
assets	It is used to store raw asset files.
libs	It contains private libraries.
res	The 'res' stands for **Resource file.** It can store resource files such as pictures, XML files, etc. It contains some additional folders such as Drawable, Layout and Values. **anim:** It is used for XML files that are compiled into animation objects. **color:** It is used for XML files that describe colors. **drawable:** It is used to store various graphics files. In Android project structure, **there are three types of drawable folders,** 1. drawable-mdpi 2. drawable-hdpi 3. drawable-ldpi The above drawable folders are required in order to adapt to different screen resolutions. **layout:** It is used for placing the XML layout files, which defines how various Android objects such as textbox, buttons, etc. are organized on the screen.

	menu: It is used for defining the XML files in the application menu. **raw:** The 'raw' stands for **Raw Asset Files.** These files are referenced from the application using a resource identifier in the R class. **For example,** good place for media is MP3 or Ogg files. **values:** It is used for XML files which stores various string values, such as titles, labels, etc. **xml:** It is used for configuring the application components.
AndroidManifest .xml	This file indicates the Android definition file. This file contains the information about the Android application such as minimum Android version, permission to access Android device capabilities such as Internet access permission, phone permission etc.
default.properti es	This file contains the project settings, such as build the target. Do not edit this file manually. It should be maintained in a Source Revision Control System.
Proguard.cfg	This file defines how ProGuard optimizes and makes your code unclear.
MainLayout.xml	This file describes the layout of the page. So all the components such as textboxes, labels, radio buttons, etc. are displaying on the application screen.
Activity class	The application occupies the entire device screen which needs at least one class inherits from the Activity class. OnCreate() method initiates the application and loads the layout page.

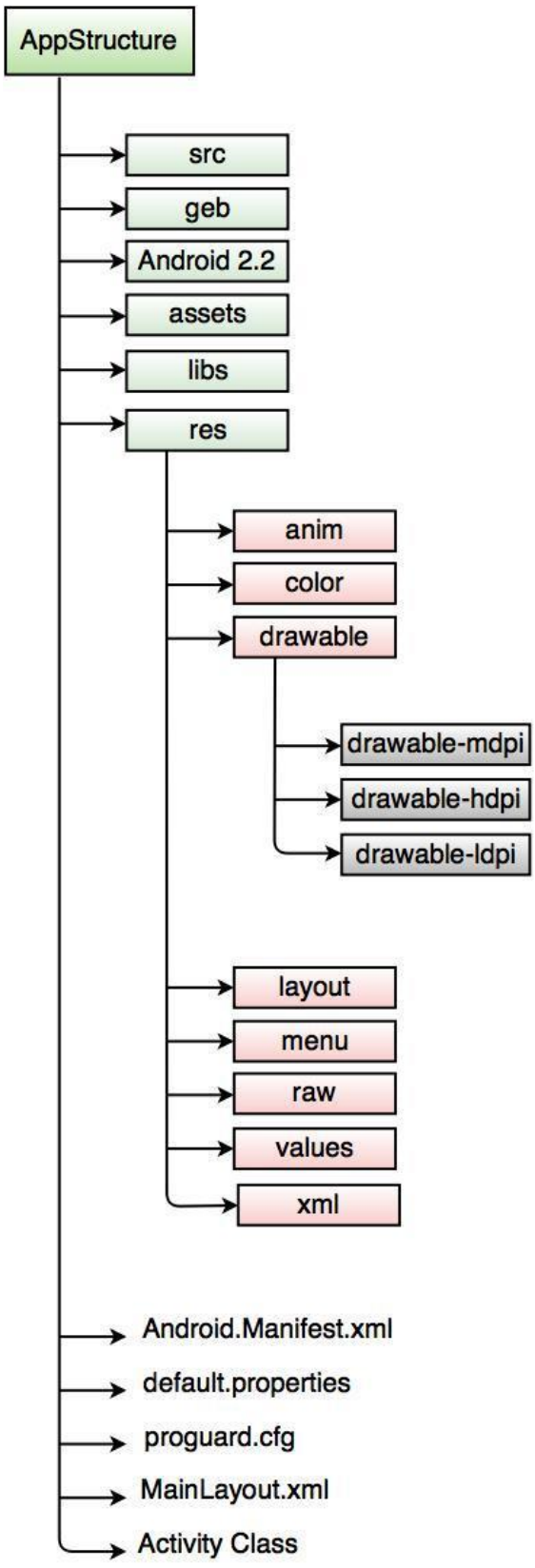


Fig. Structure of Android Application

AndroidManifest.xml file in android

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:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.javatpoint.hello"
  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>
```

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.

Android R.java file

Android R.java is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.

> *Note: If you delete R.jar file, android creates it automatically*

Android Activities and Activity Lifecycle

Activities

- Activity is also known as Widgets.
- Activity represents a single screen with a user interface.
- It is an individual user interface screen in an Android Application where visual elements called Views.
- It interacts with the user to do only one thing, such as unlock screen, dial a phone, etc.
- If new activity starts, then previous activity is stopped, but the data is preserved.
- An application consists of multiple activities.
  For example, an email application has one activity to display a list of new emails, another activity is to compose email, reading email and so on.

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.

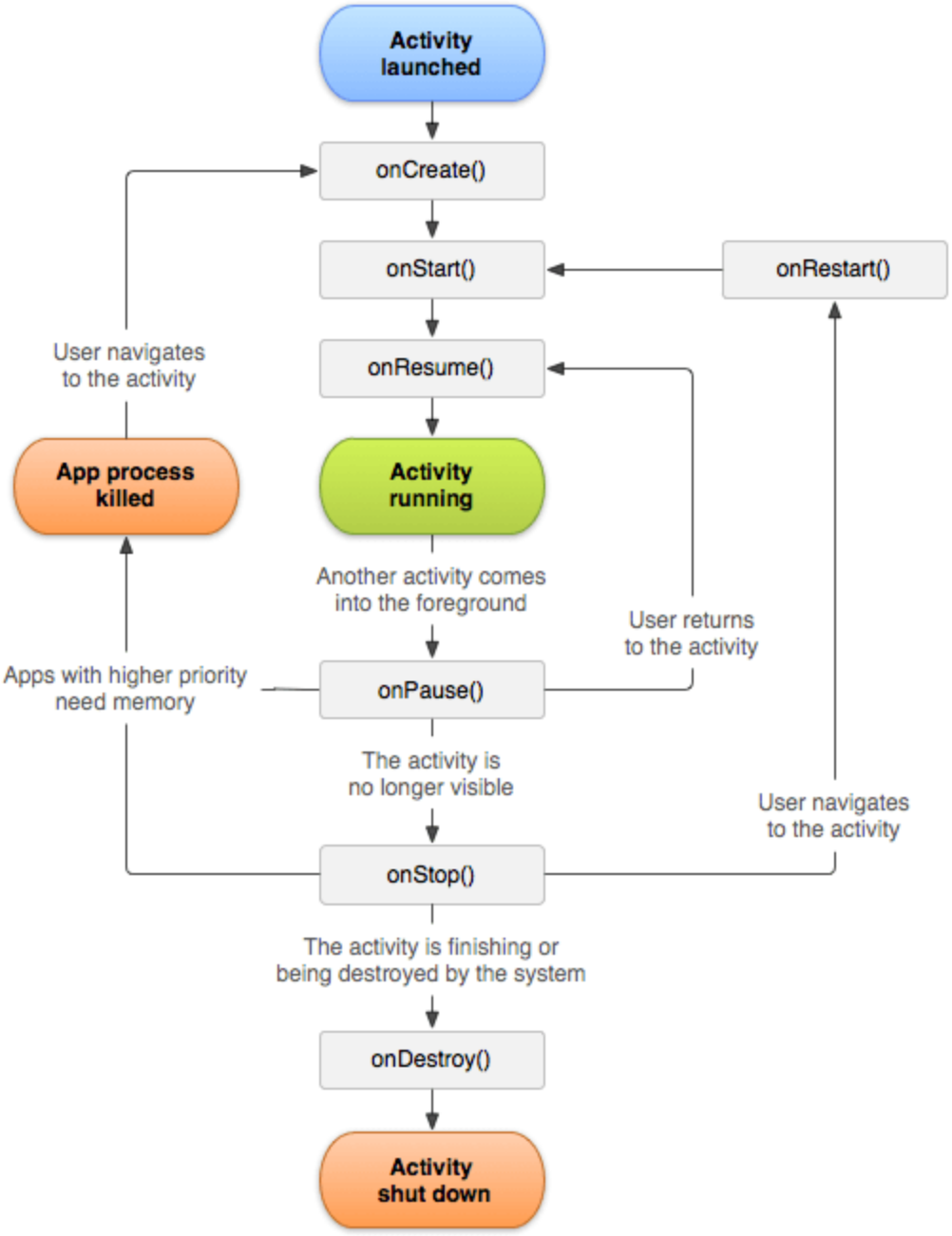
By the help of activity, you can place all your UI components or widgets in a single screen.

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.


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

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

```
package example.javatpoint.com.activitylifecycle;

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

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
```

```
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}
```

Uses - Permission

To request a permission an app must include a `<uses-permission>` tag in its **`AndroidManifest.xml`** file.

Android has many built-in permissions that an app can request. For example, an app can request access to the internet as follows:

```
<manifest …>

  <uses-permission android:name="android.permission.INTERNET">

  …

</manifest>
```

Specifies a system permission that the user must grant in order for the app to operate correctly. Permissions are granted by the user when the application is installed (on devices running Android 5.1 and lower) or while the app is running (on devices running Android 6.0 and higher).
For more information on permissions, see the Permissions section in the introduction and the separate System Permissions API guide. A list of permissions defined by the base platform can be found at `android.Manifest.permission`.

attributes:

`android:name`

The name of the permission. It can be a permission defined by the application with the `<permission>` element, a permission defined by another application, or one of the standard system permissions (such as `"android.permission.CAMERA"` or `"android.permission.READ_CONTACTS"`).

uses-sdk

Lets you express an application's compatibility with one or more versions of the Android platform, by means of an API Level integer. The API Level expressed by an application will be compared to the API Level of a given Android system, which may vary among different Android devices.

Despite its name, this element is used to specify the API Level, *not* the version number of the SDK (software development kit) or Android platform. The API Level is always a single integer. You cannot derive the API Level from its associated Android version number (for example, it is not the same as the major version or the sum of the major and minor versions).

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

attributes:

`android:minSdkVersion`

An integer designating the minimum API Level required for the application to run. The Android system will prevent the user from installing the application if the system's API Level is lower than the value specified in this attribute. You should always declare this attribute.

`android:targetSdkVersion`

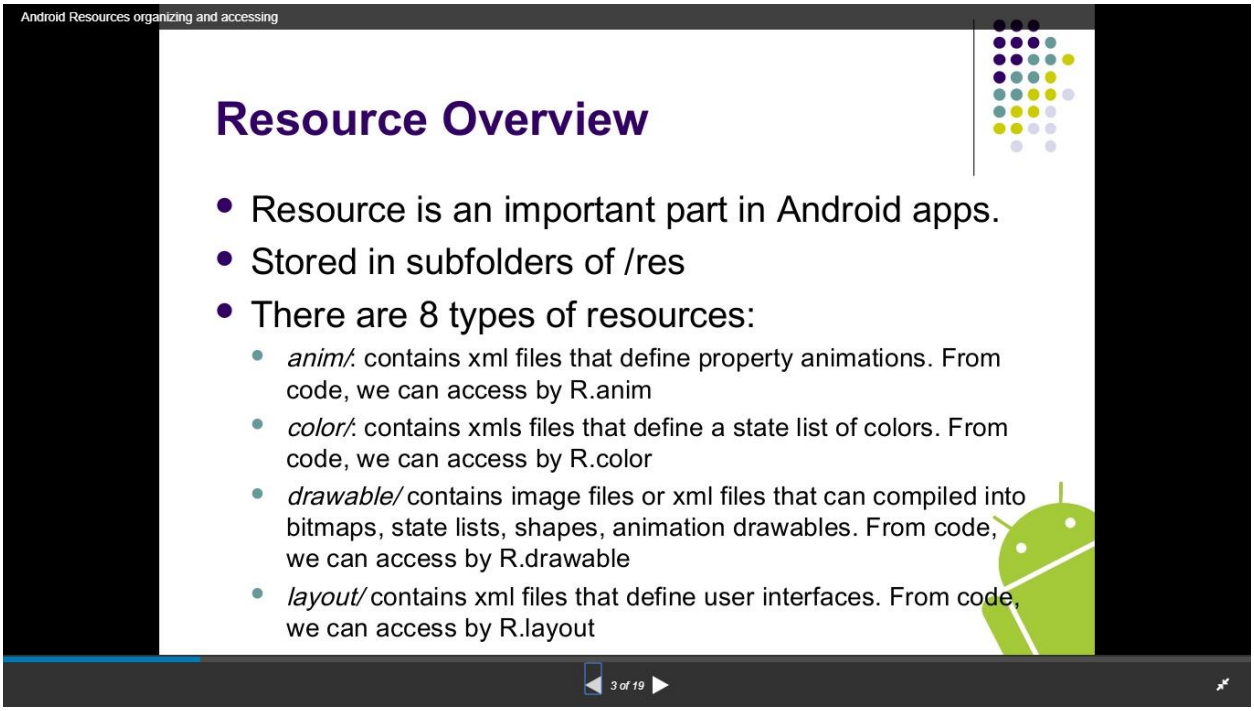
An integer designating the API Level that the application targets. If not set, the default value equals that given to `minSdkVersion`.

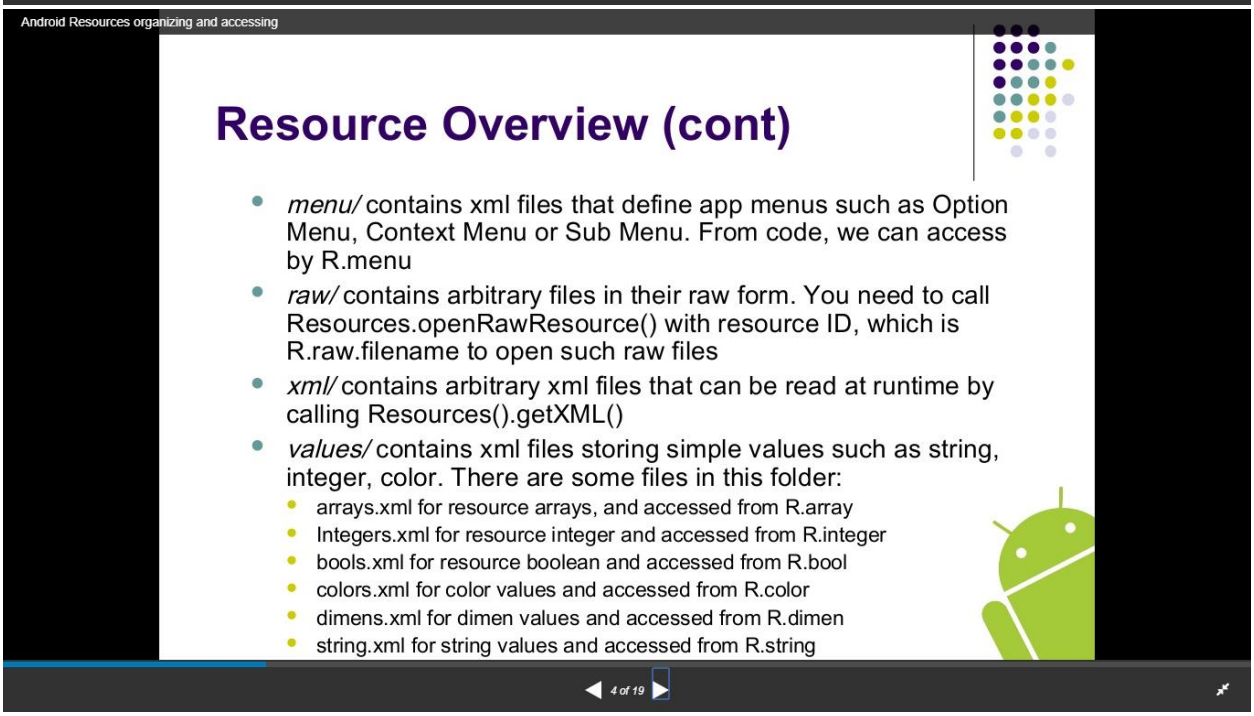
This attribute informs the system that you have tested against the target version and the system should not enable any compatibility behaviors to maintain your app's forward-compatibility with the target version. The application is still able to run on older versions (down to `minSdkVersion`).

`android:maxSdkVersion`

An integer designating the maximum API Level on which the application is designed to run.

In Android 1.5, 1.6, 2.0, and 2.0.1, the system checks the value of this attribute when installing an application and when re-validating the application after a system update. In either case, if the application's `maxSdkVersion` attribute is lower than the API Level used by the system itself, then the system will not allow the application to be installed.





There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other **resources** like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

Organize resource in Android Studio

```
MyProject/
   app/
      manifest/
         AndroidManifest.xml
   java/
      MyActivity.java
      res/
         drawable/
            icon.png
         layout/
            activity_main.xml
            info.xml
         values/
            strings.xml
```