Course: 305-02: Mobile Application Development – 1

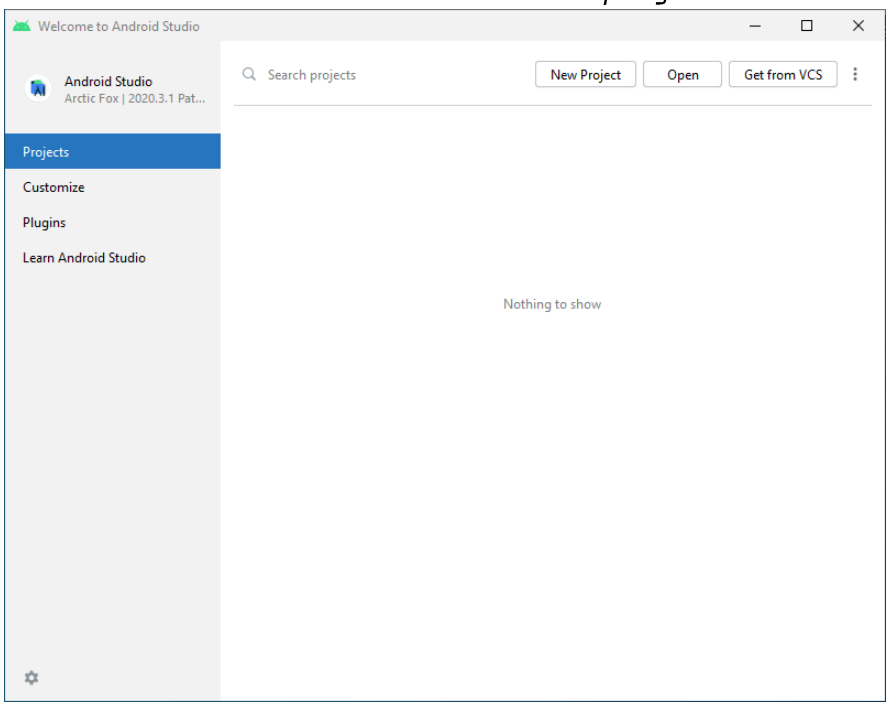
Unit-4: Creating basic App

4. Creating basic App
   4.1. Basic App using Android studio
      4.1.1. Create new android project
      4.1.2. Write message and run

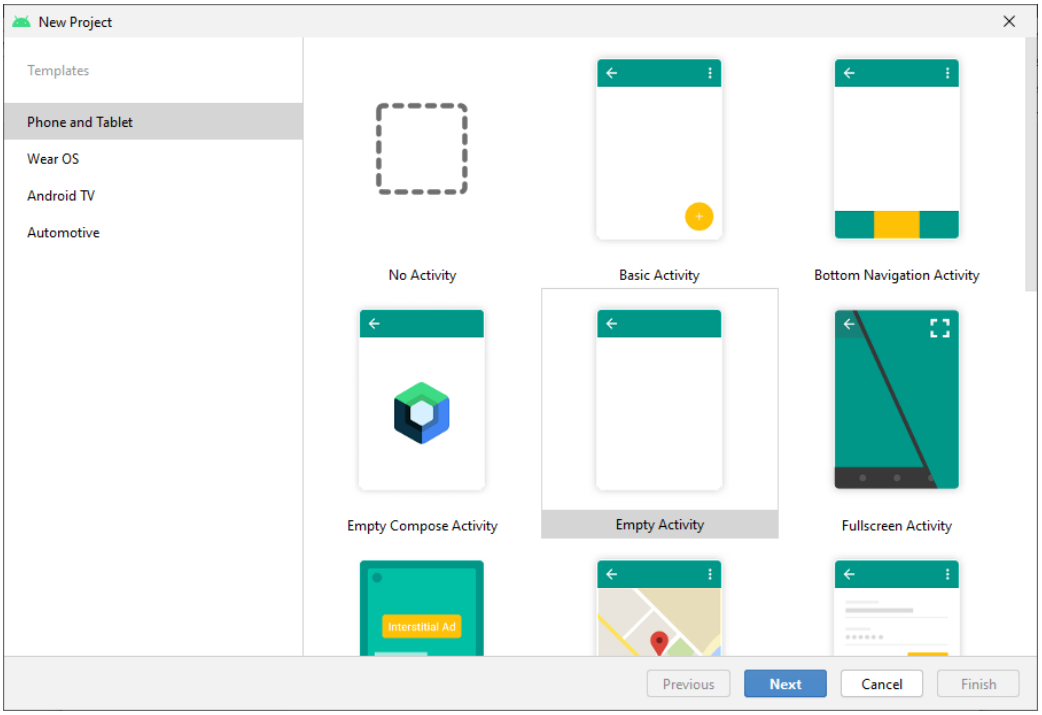
<u>Create the New Android project</u>

For creating the new android studio project:

1) Select Start a new Android Studio project

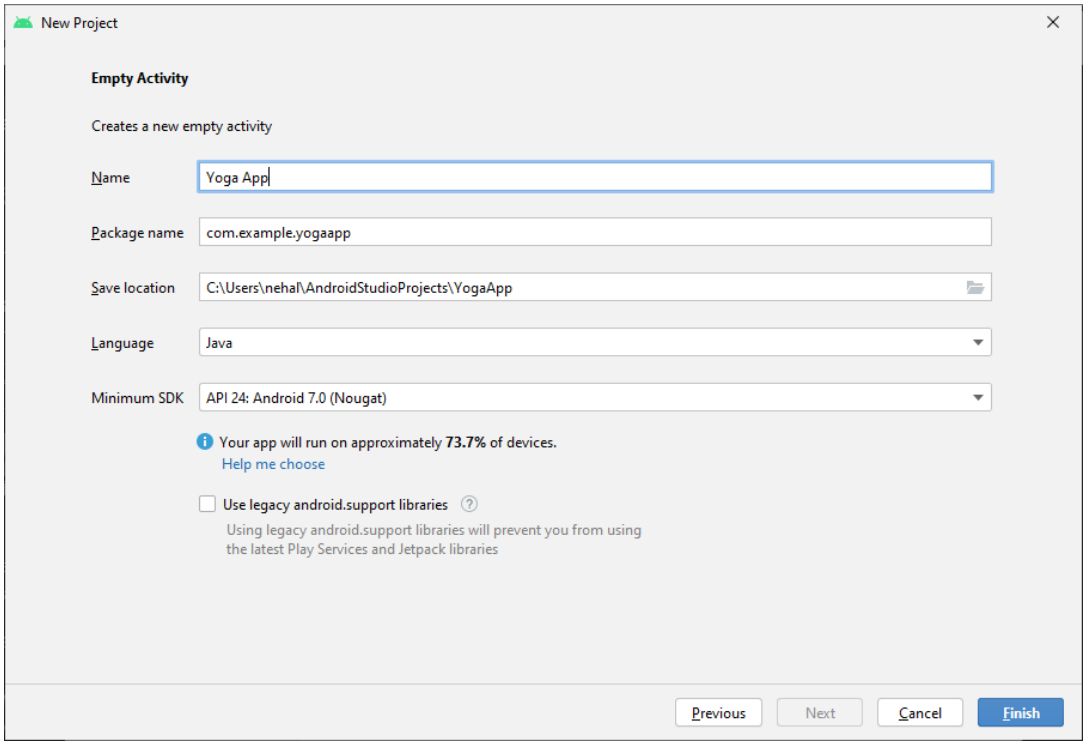


- Click on "New Project" Button, or click "Open" button for existing project.



- *Select a layout from existing templates or select "Empty Activity" for blank template.*

2) *Provide the following information: Application name, Company domain, Project location and Package name of application and click next.*



- ***Name***: *Provide your application name, this name will be shown default in your application's title bar.*
- ***Package Name***: *Provide your application package name. generally its follows a domain name in reverse. This package name should be unique while publishing application on Google Play Store.*
- ***Save Location***: *Provide the Location path of your local drive.*
- ***Language***: *Select "Java" as project language*
- ***Minimum SDK***: *Select Minimum SDK version your application requires. Your application will work only in devices which have minimum SDK version or newer android version.*
    - *For example if you select "API 24 Android 7.0 Nougat", your application will work in devices which have android minimum version 7.0 Nougat or newer than that.*

3) *Click Finish, and android studio will build a new project with provided information.*

After finishing the Activity configuration, Android Studio auto generates the activity class and other required configuration files.

Now an android project has been created. You can explore the android project and see the simple program; it looks like this:

4.1.3. Understanding different components.

Android Core Building Blocks

android components

An android component is simply a piece of code that has a well-defined life cycle e.g., Activity, Receiver, Service etc.

The core building blocks or fundamental components of android are activities, views, intents, services, content providers, fragments and AndroidManifest.xml.

Activity

An activity is a class that represents a single screen. It is like a Frame in AWT.

View

A view is the UI element such as button, label, text field etc. Anything that you see is a view.

Intent

Intent is used to invoke components. It is mainly used to:

- *Start the service*
- *Launch an activity*
- *Display a web page*
- *Display a list of contacts*
- *Broadcast a message*
- *Dial a phone call etc.*

For example, you may write the following code to view the webpage.

```
1. Intent intent=new Intent(Intent.ACTION_VIEW);
2. intent.setData(Uri.parse("http://www.google.com"));
3. startActivity(intent);
```

Service

Service is a background process that can run for a long time.

There are two types of services local and remote. Local service is accessed from within the application whereas remote service is accessed remotely from other applications running on the same device.

Content Provider

Content Providers are used to share data between the applications.

Fragment

Fragments are like parts of activity. An activity can display one or more fragments on the screen at the same time.

AndroidManifest.xml

It contains information about activities, content providers, permissions etc. It is like the web.xml file in Java EE.

Android Virtual Device (AVD)

It is used to test the android application without the need for mobile or tablet etc. It can be created in different configurations to emulate different types of real devices.

4.2. Dalvik Virtual Machine (DVM)

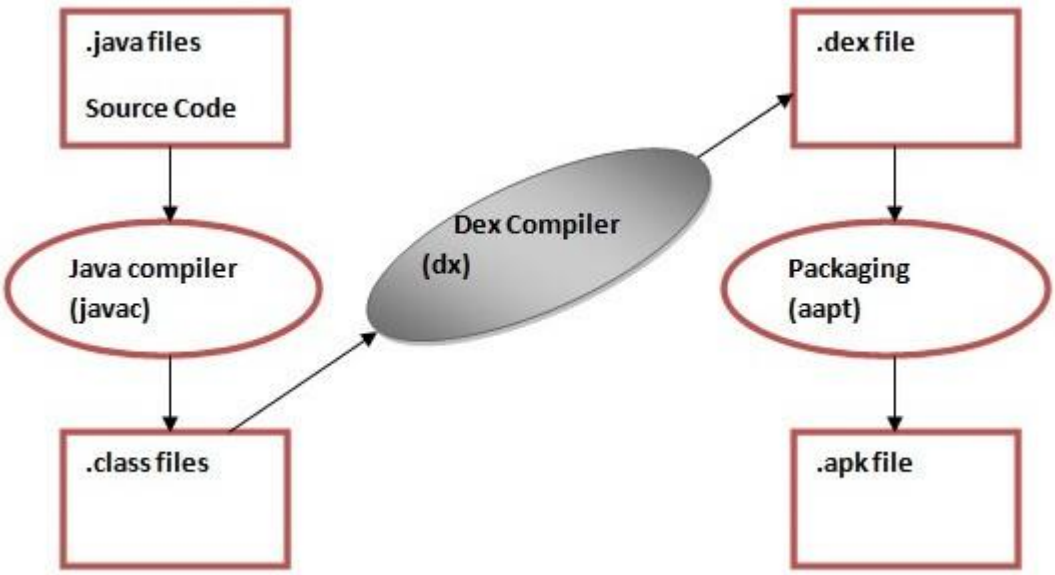
As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well.

The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for memory, battery life and performance.

Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.

The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

Let's see the compiling and packaging process from the source file:



The **javac** tool compiles the java source file into the class file.

The **dx tool** takes all the class files of your application and generates a single .dex file. It is a platform-specific tool.

The **Android Assets Packaging Tool (aapt)** handles the packaging process.

4.3. Understanding AndroidManifest.xml

*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 provide profiling and other information. This information is 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.example.my_application"
3.    android:versionCode="1"
4.    android:versionName="1.0" >
5.
6.    <uses-sdk
7.      android:minSdkVersion="8"
8.      android:targetSdkVersion="15" />
9.
10.   <application
11.     android:icon="@drawable/ic_launcher"
12.     android:label="@string/app_name"
13.     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

<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.

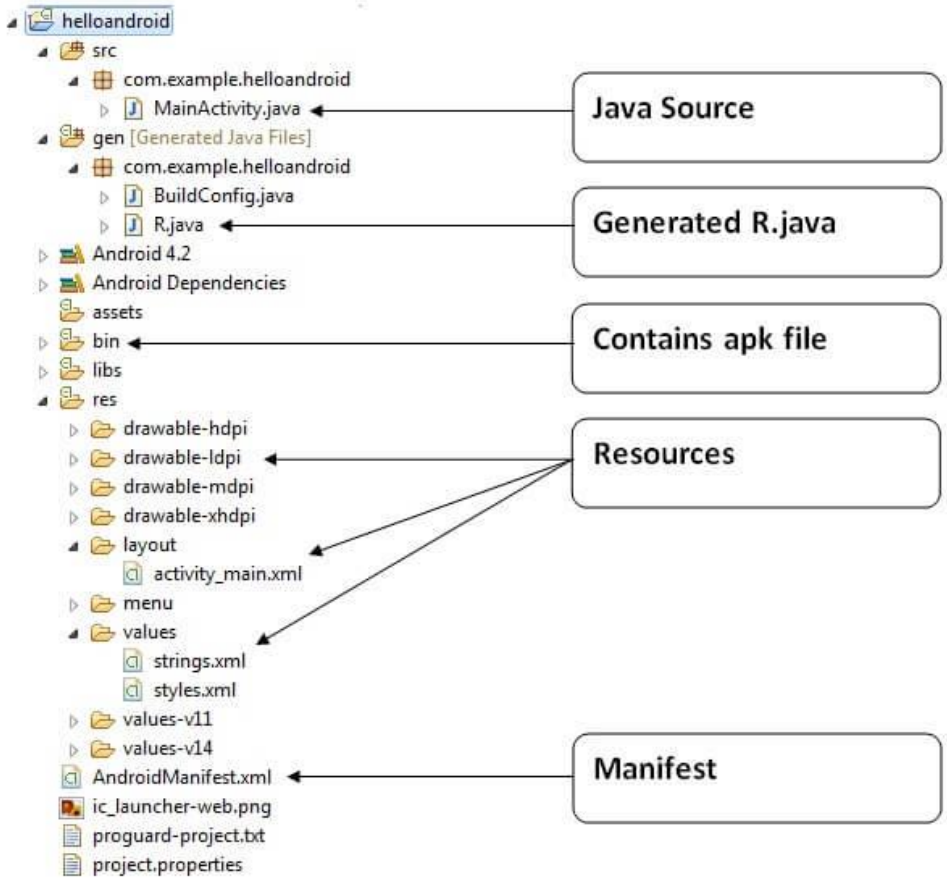
<category>

It adds a category name to an intent-filter.

Internal Details of Hello Android Example

Here, we are going to learn the internal details or working of hello android example.

Android application contains different components such as java source code, string resources, images, manifest file, apk file etc. Let's understand the project structure of android application.



Java Source Code

Let's see the java source file created by the Eclipse IDE:

File: MainActivity.java

```
1. package com.example.helloandroid;
2. import android.os.Bundle;
3. import android.app.Activity;
4. import android.view.Menu;
```

```
import android.widget.TextView;
public class MainActivity extends Activity {//(1)
    @Override
    protected void onCreate(Bundle savedInstanceState) {//(2)
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);//(3)
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {//(4)
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

(1) Activity is a java class that creates and default window on the screen where we can place different components such as Button, EditText, TextView, Spinner etc. It is like the Frame of Java AWT.

It provides life cycle methods for activity such as onCreate, onStop, OnResume etc.

(2) The onCreate method is called when Activity class is first created.

(3) The setContentView(R.layout.activity_main) gives information about our layout resource. Here, our layout resources are defined in activity_main.xml file.

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/androi
d"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />
</RelativeLayout>
```

As you can see, a textview is created by the framework automatically. But the message for this string is defined in the strings.xml file. The @string/hello_world provides information about the textview message. The value of the attribute hello_world is defined in the strings.xml file.

File: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">helloandroid</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
</resources>
```

You can change the value of the hello_world attribute from this file.

Generated R.java file

It is the auto-generated file that contains IDs for all the resources of res directory. It is generated by aapt(Android Asset Packaging Tool). Whenever you create any component on activity_main, a corresponding ID

is created in the R.java file which can be used in the Java Source file later.

APK File

An apk file is created by the framework automatically. If you want to run the android application on the mobile, transfer and install it.

Resources

It contains resource files including activity_main, strings, styles etc.

Manifest file

It contains information about package including components such as activities, services, content providers etc.

For more information about manifest file visit here: AndroidManifest.xml file.

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.

Let's see the android R.java file. It includes a lot of static nested classes such as menu, id, layout, attr, drawable, string etc.

```
1.  /* AUTO-GENERATED FILE.  DO NOT MODIFY.
2.   *
3.   * This class was automatically generated by the
4.   * aapt tool from the resource data it found.  It
5.   * should not be modified by hand.
6.   */
7.
8.  package com.example.helloandroid;
9.
10. public final class R {
11.     public static final class attr {
12.     }
13.     public static final class drawable {
14.         public static final int ic_launcher=0x7f020000;
15.     }
16.     public static final class id {
```

```
17.         public static final int menu_settings=0x7f070000;
18.     }
19.     public static final class layout {
20.         public static final int activity_main=0x7f030000;
21.     }
22.     public static final class menu {
23.         public static final int activity_main=0x7f060000;
24.     }
25.     public static final class string {
26.         public static final int app_name=0x7f040000;
27.         public static final int hello_world=0x7f040001;
28.         public static final int menu_settings=0x7f040002;
29.     }
30.     public static final class style {
31.         /**
32.         Base application theme, dependent on API level. This theme is replaced
33.         by AppBaseTheme from res/values-vXX/styles.xml on newer devices.
34.
35.
36.             Theme customizations available in newer API levels can go in
37.             res/values-vXX/styles.xml, while customizations related to
38.             backward-compatibility can go here.
39.
40.
41.         Base application theme for API 11+. This theme completely replaces
42.         AppBaseTheme from res/values/styles.xml on API 11+ devices.
43.
44.  API 11 theme customizations can go here.
45.
46.         Base application theme for API 14+. This theme completely replaces
47.         AppBaseTheme from BOTH res/values/styles.xml and
48.         res/values-v11/styles.xml on API 14+ devices.
49.
50.  API 14 theme customizations can go here.
```

```
51.         */
52.        public static final int AppBaseTheme=0x7f050000;
53.        /**  Application theme.
54.  All customizations that are NOT specific to a particular API-level can go here.
55.         */
56.        public static final int AppTheme=0x7f050001;
57.    }
58. }
```