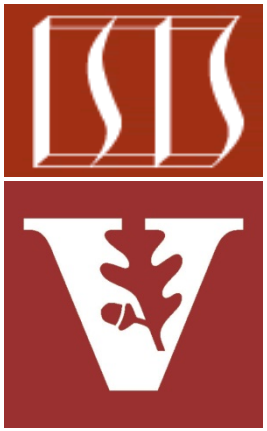
Developing Android Apps: Part 1



Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

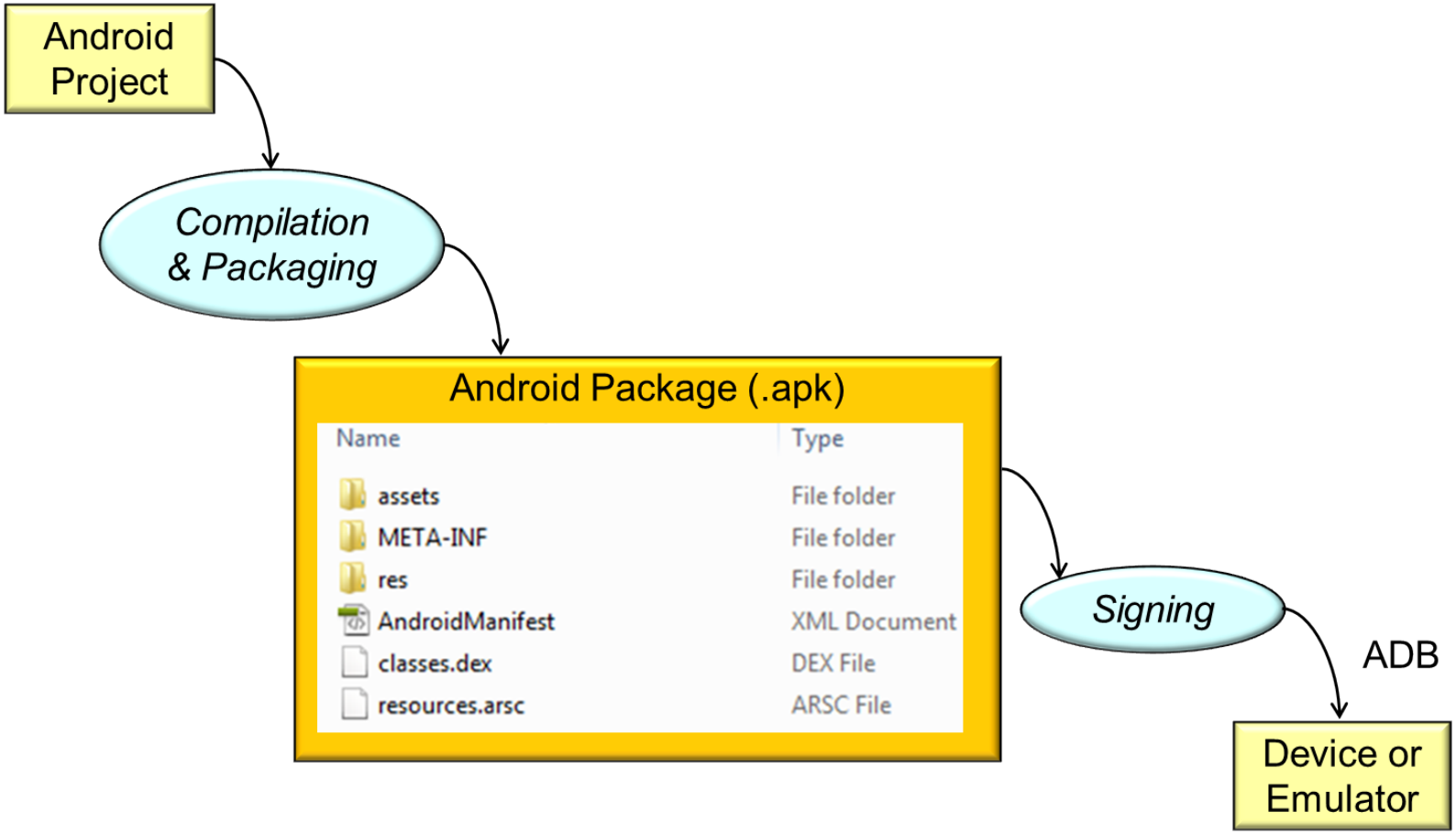
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

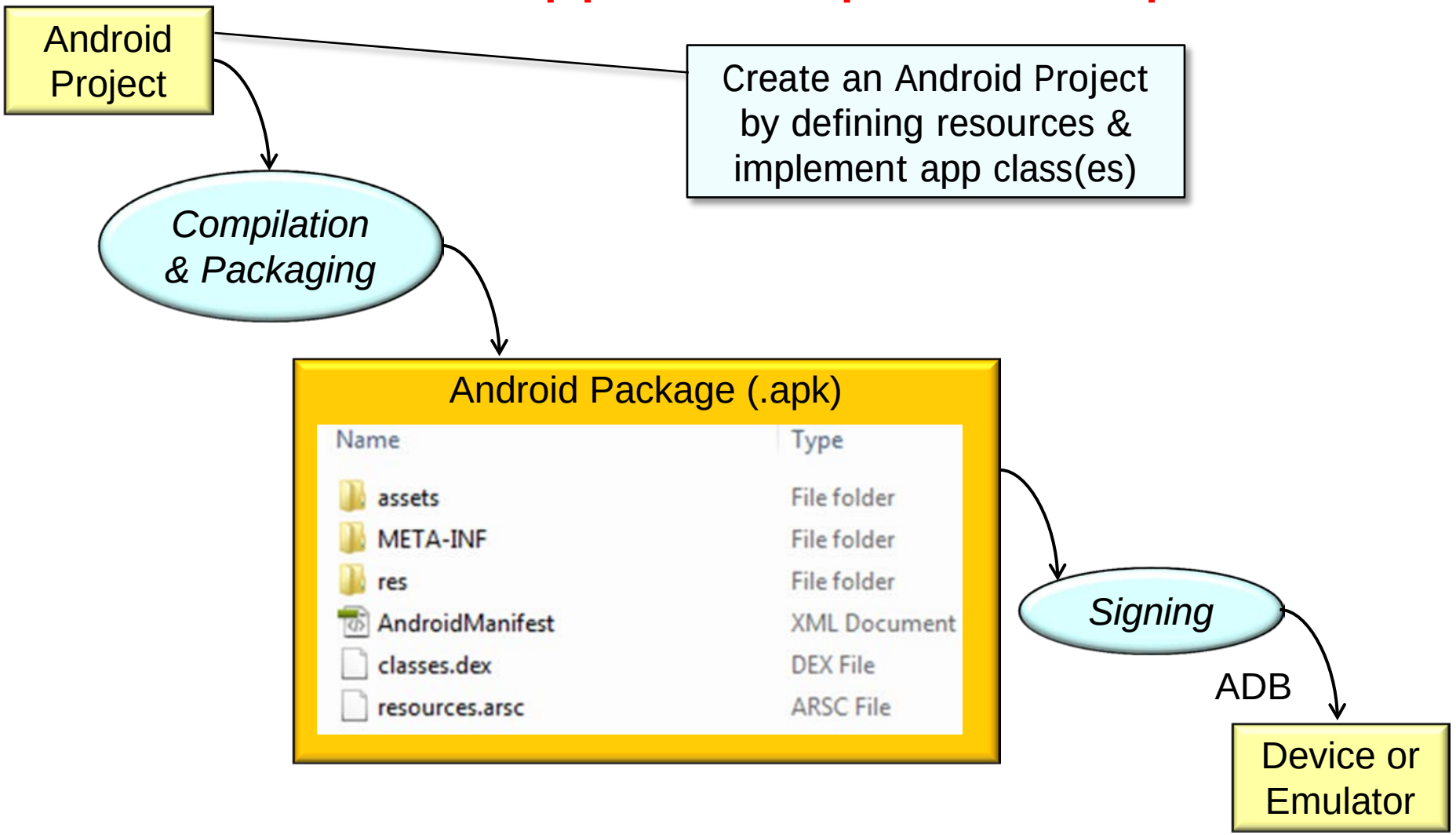
Systems Programming for Android

Learning Objectives in this Part of the Module

- Understand the key steps in developing an Android app

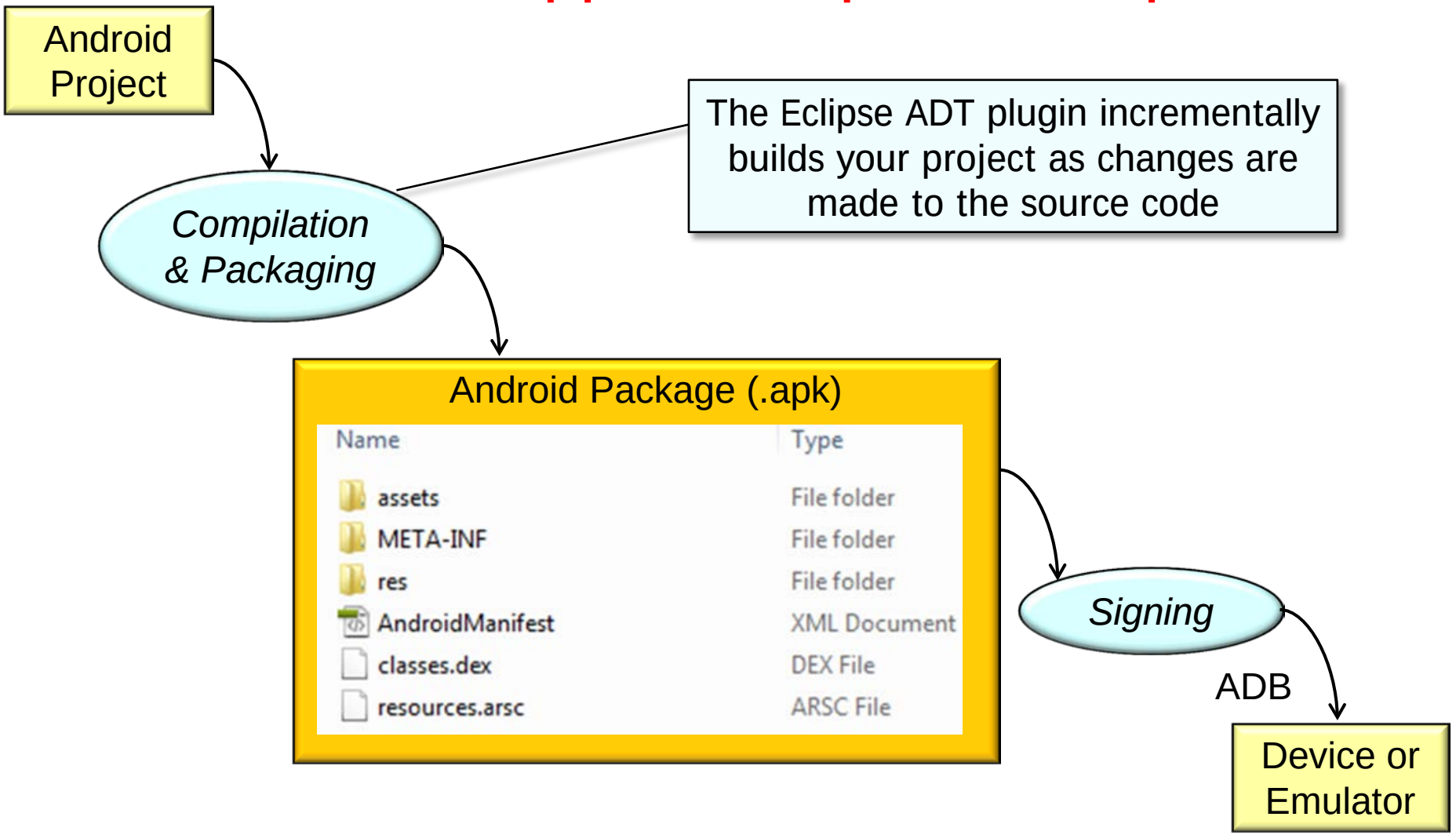


Android App Development Steps



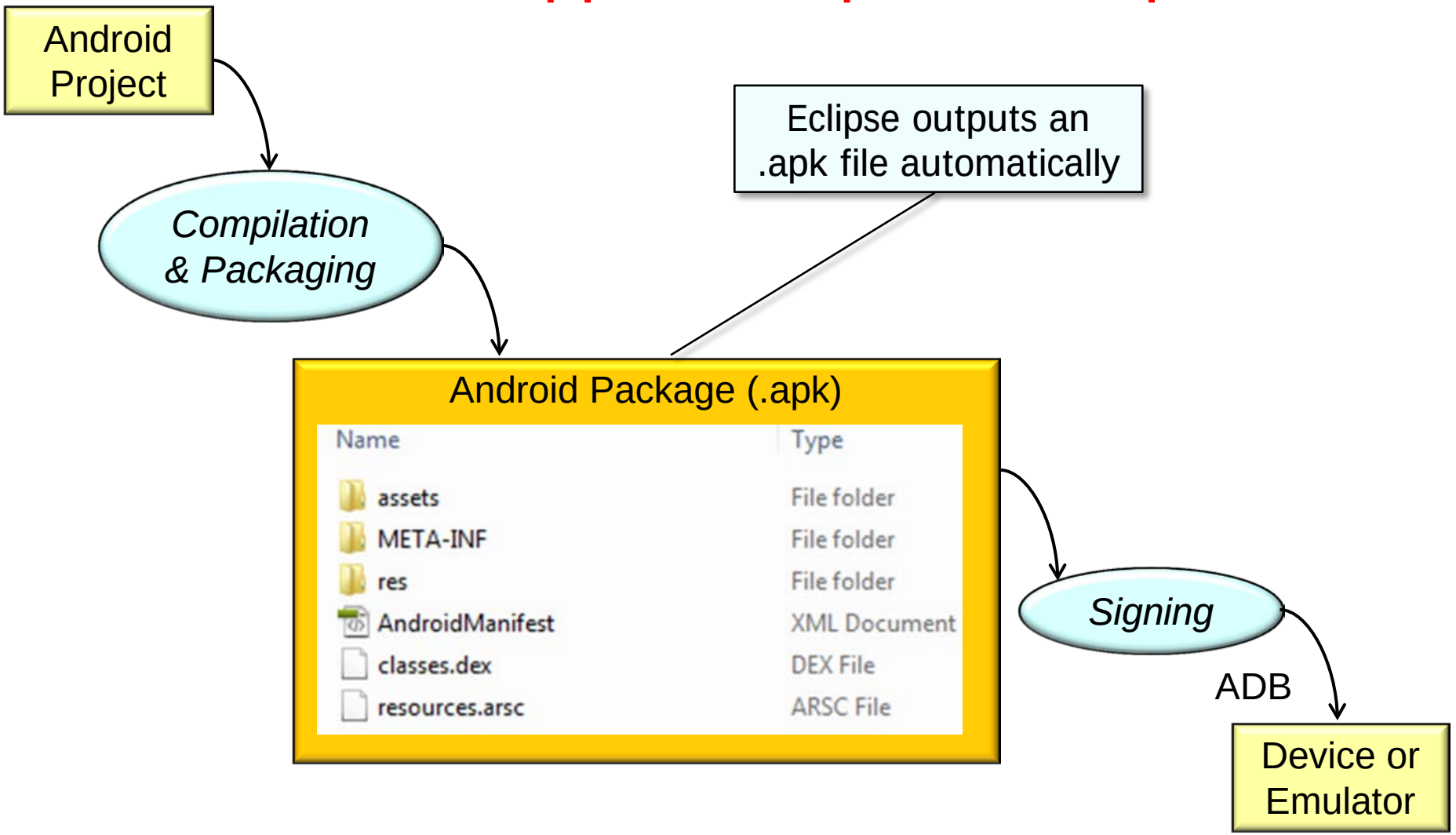
developer.android.com/guide/developing/building has more info on this process

Android App Development Steps



developer.android.com/tools/sdk/eclipse-adt.html has more on ADT plugin

Android App Development Steps



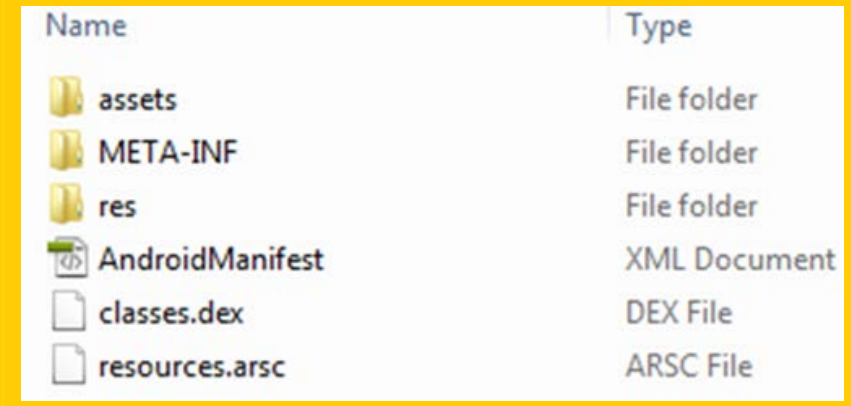
Android App Development Steps

Android Project

Compilation & Packaging

Android system uses signed certificates to identify the author of an app & establish trust relationships between apps

Android Package (.apk)



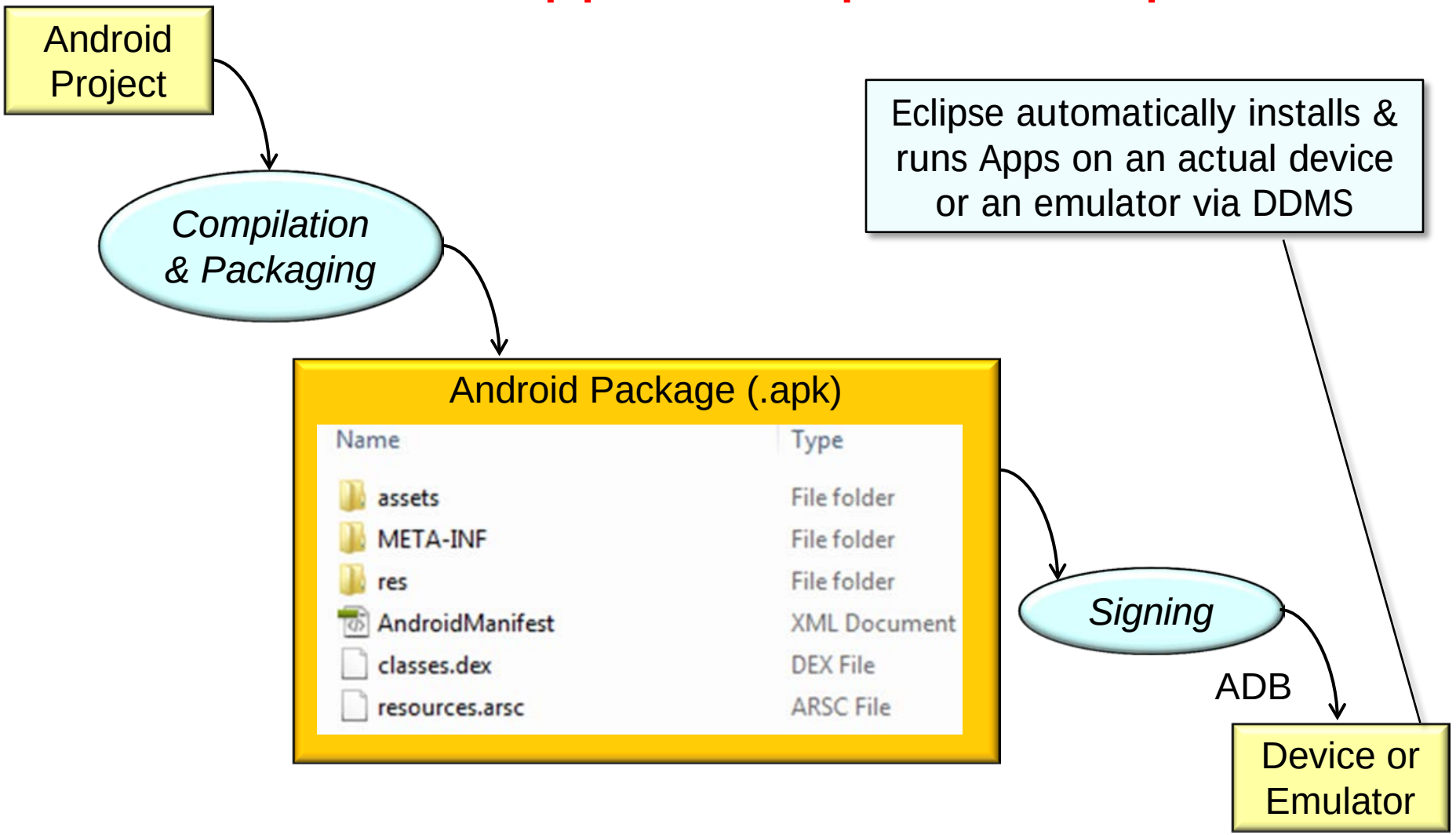
Signing

ADB

Device or Emulator

developer.android.com/tools/publishing/app-signing.html has more on signing

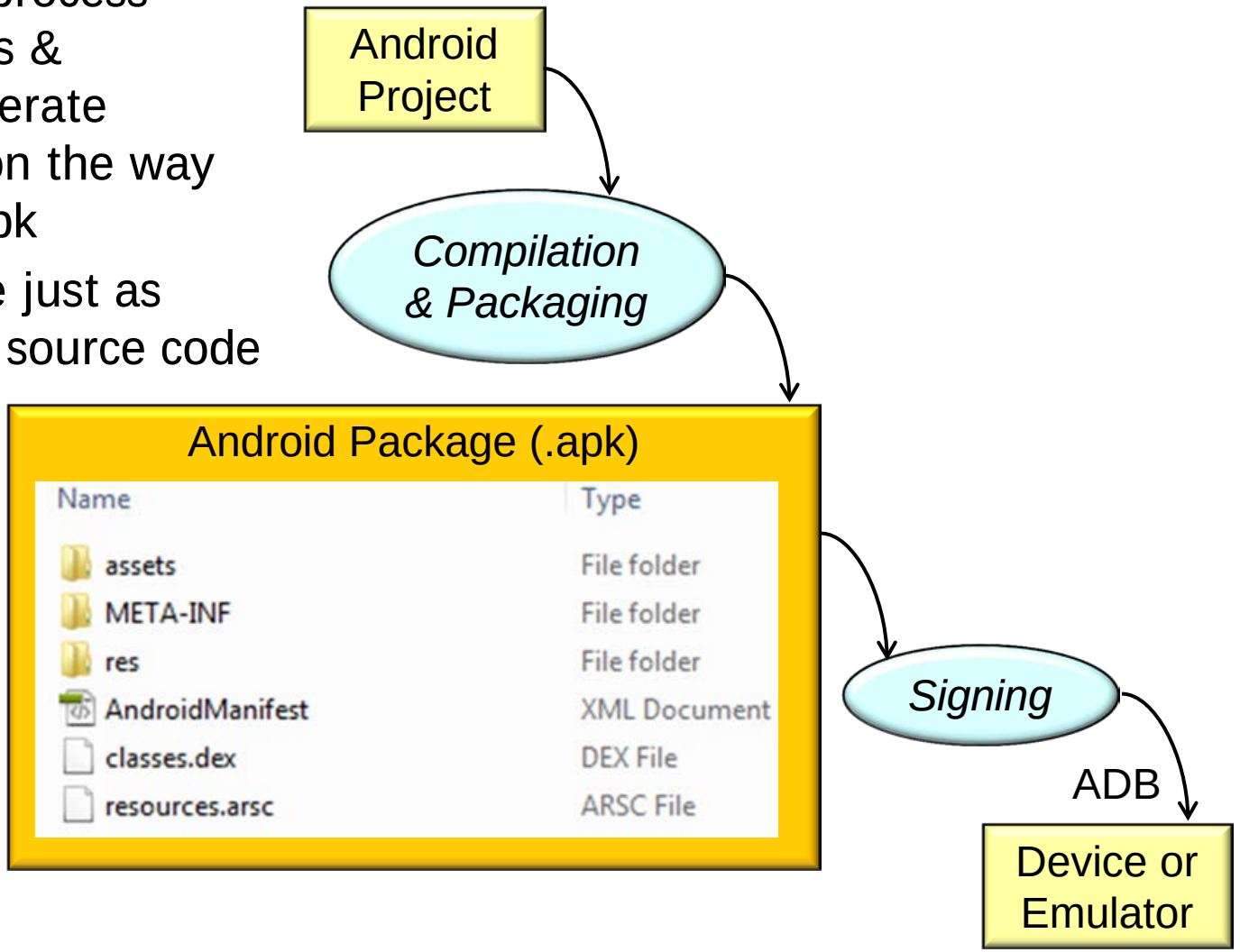
Android App Development Steps



developer.android.com/tools/debugging/ddms.html has more on DDMS

Summary

- The Android build process involves many tools & processes that generate intermediate files on the way to producing an .apk
  - The XML files are just as important as the source code

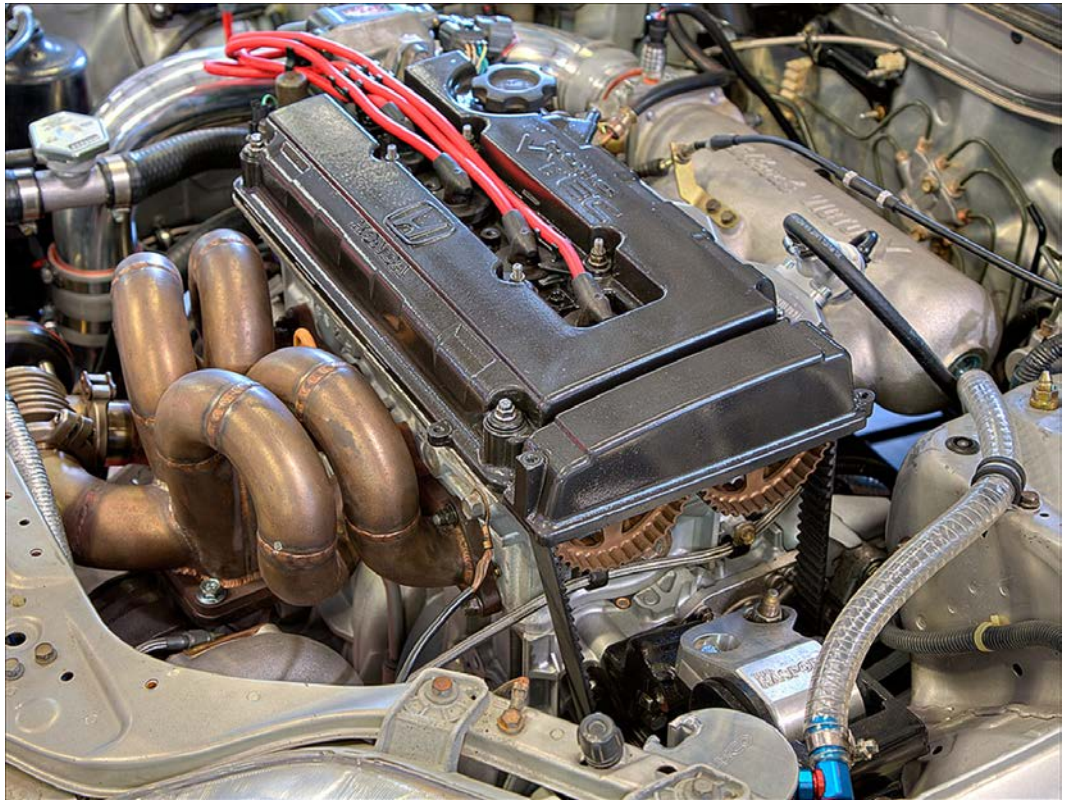


Summary

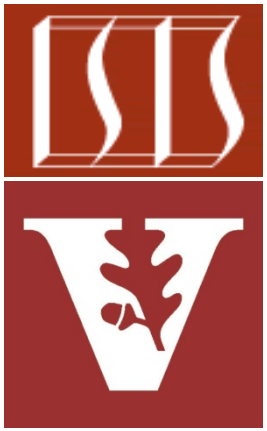
- The Android build process involves many tools & processes that generate intermediate files on the way to producing an .apk
- If you use Eclipse, the complete build process is automatically done periodically as you develop & save your changes

Summary

- The Android build process involves many tools & processes that generate intermediate files on the way to producing an .apk
- If you use Eclipse, the complete build process is automatically done periodically as you develop & save your changes
- It is useful, however, to understand what's happening under the hood since much of the tools & processes are masked from you



Developing Android Apps: Part 2



Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

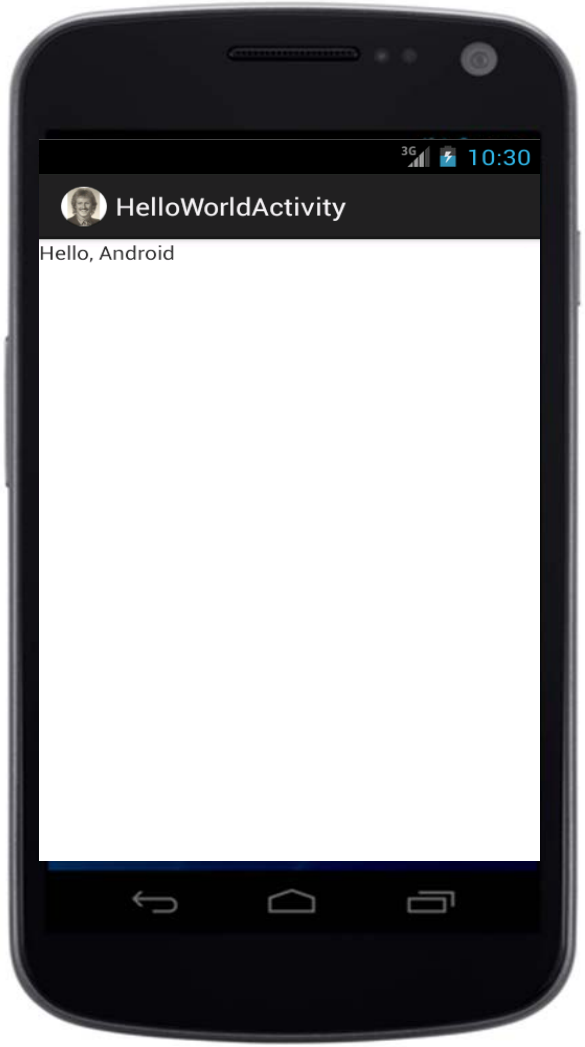
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

Systems Programming for Android

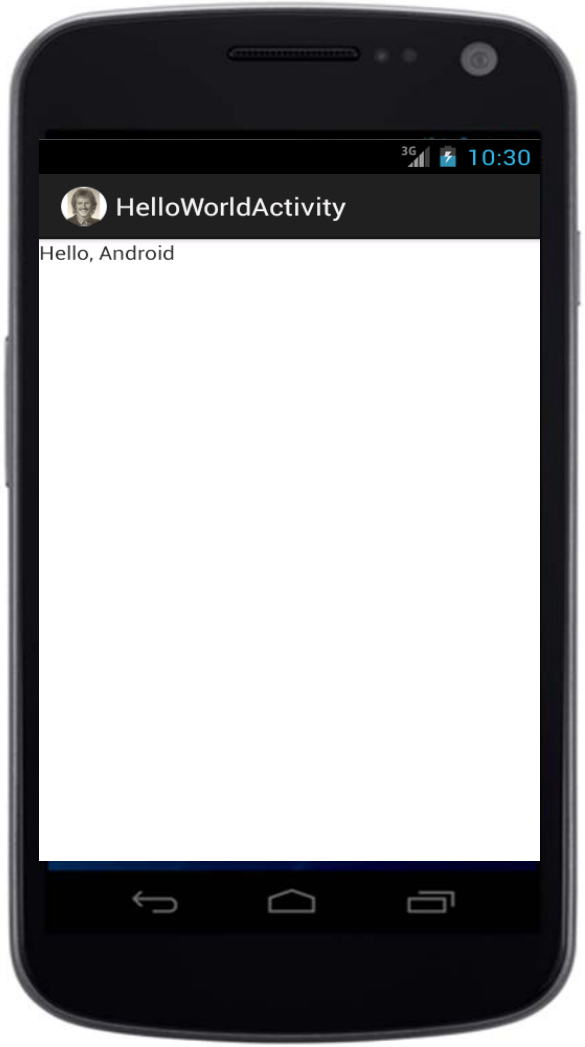
Learning Objectives in this Part of the Module

- Understand how to use Eclipse to create a simple Android app



A Simple "Hello Android" Example

- Project name: HelloAndroid
- Application name: Hello, Android
- Package name: course.examples.helloandroid
- Create activity: HelloAndroidActivity
- Min SDK: 8 (Froyo)

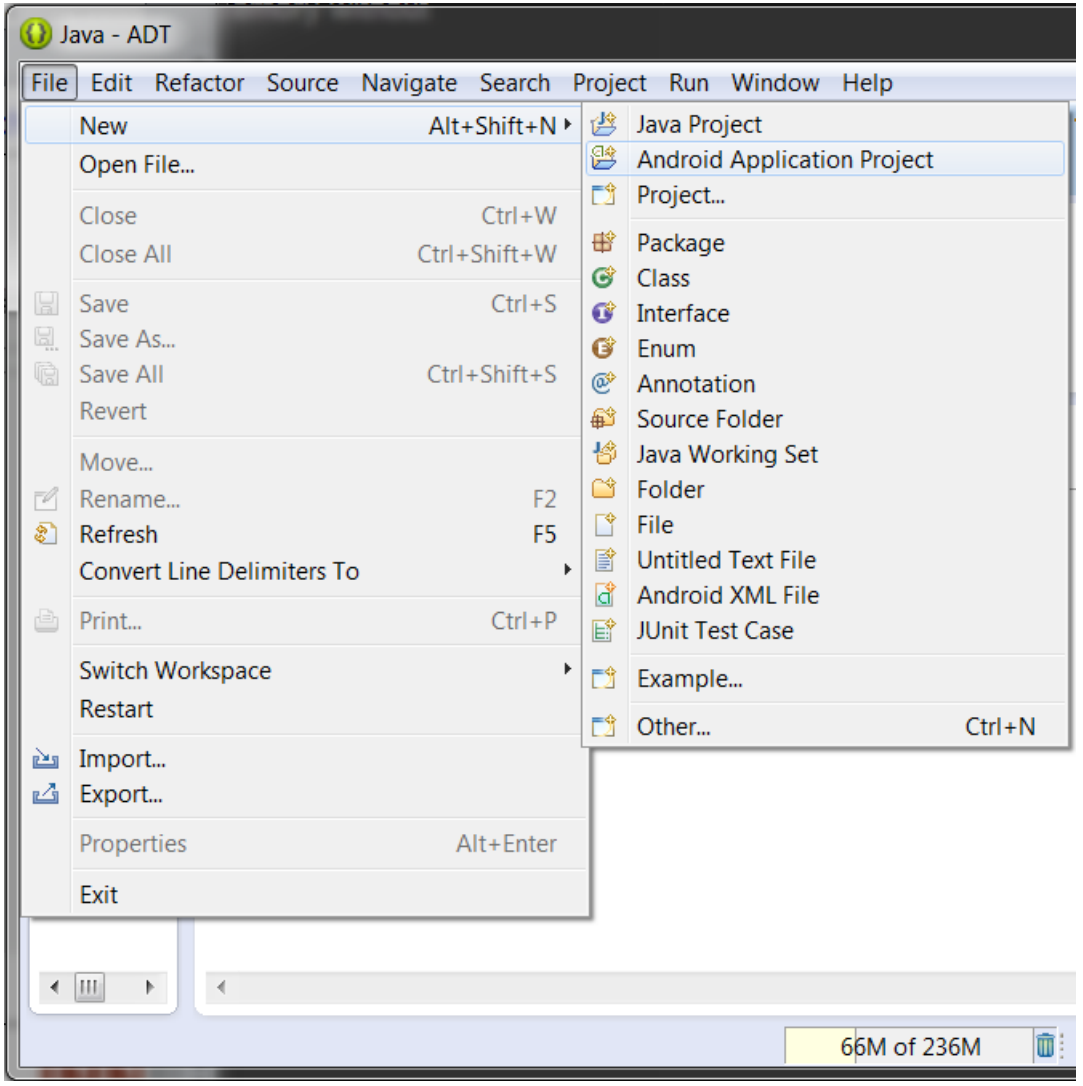


Creating the “Hello Android” Android Project

- Start Eclipse
  - BTW, if you can afford extra memory & a solid-state drive I highly recommend it!

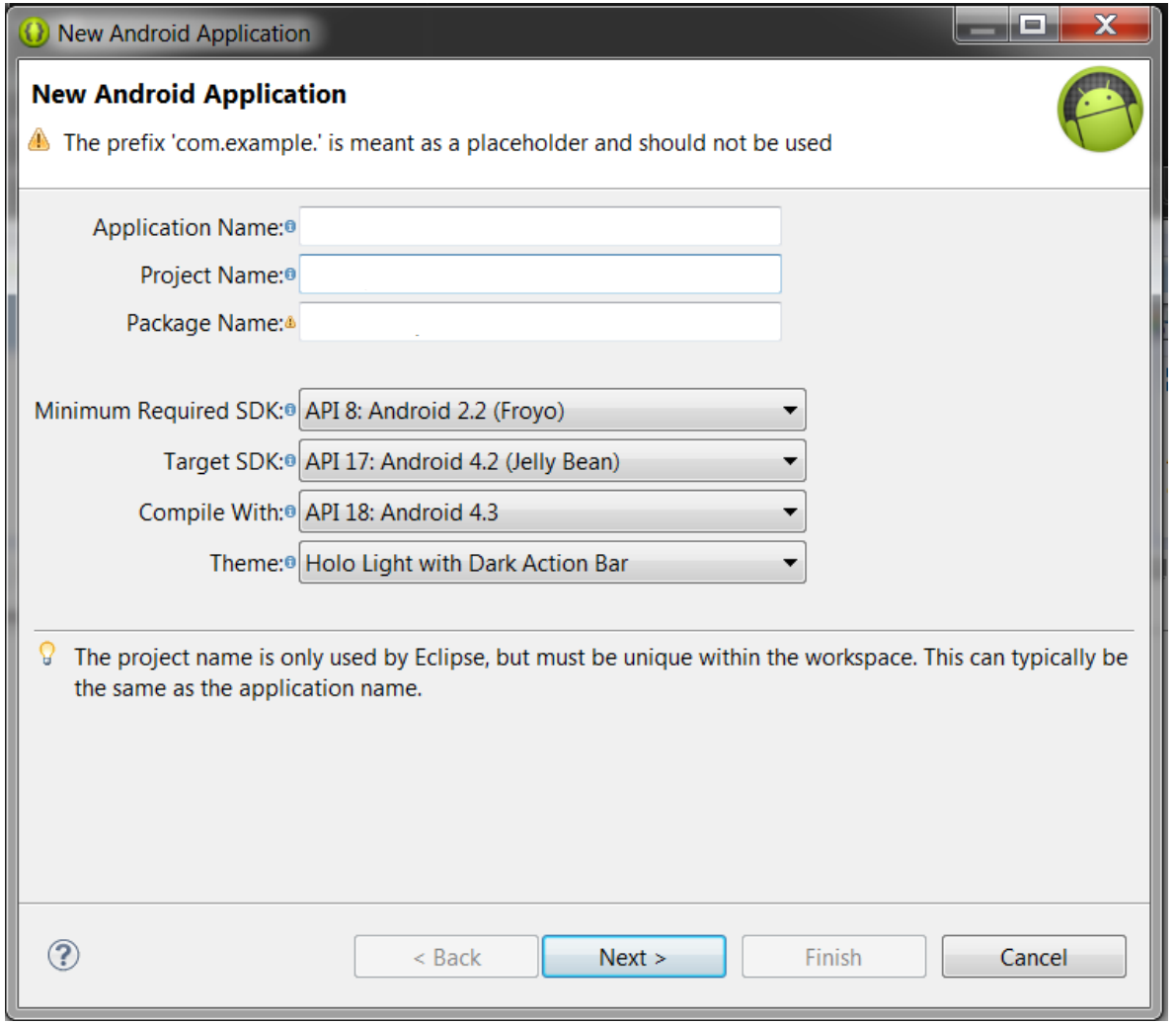
Creating the "Hello Android" Android Project

- Start Eclipse
- Create a new Android project from the file menu:
  - File->New->Android Application Project



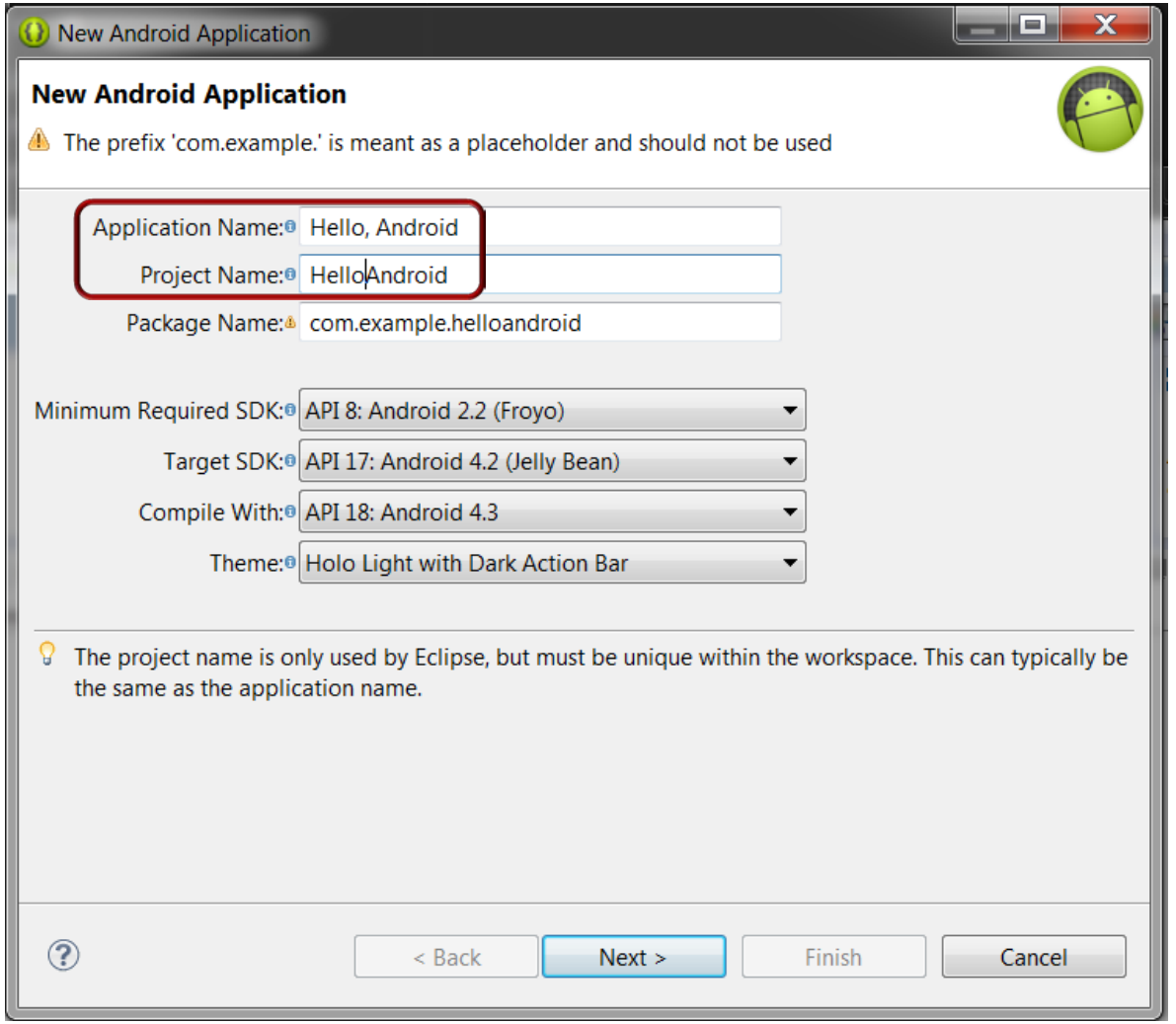
Configuring the Project Settings Via a Wizard

- The new project wizard is where you setup critical information for an app:



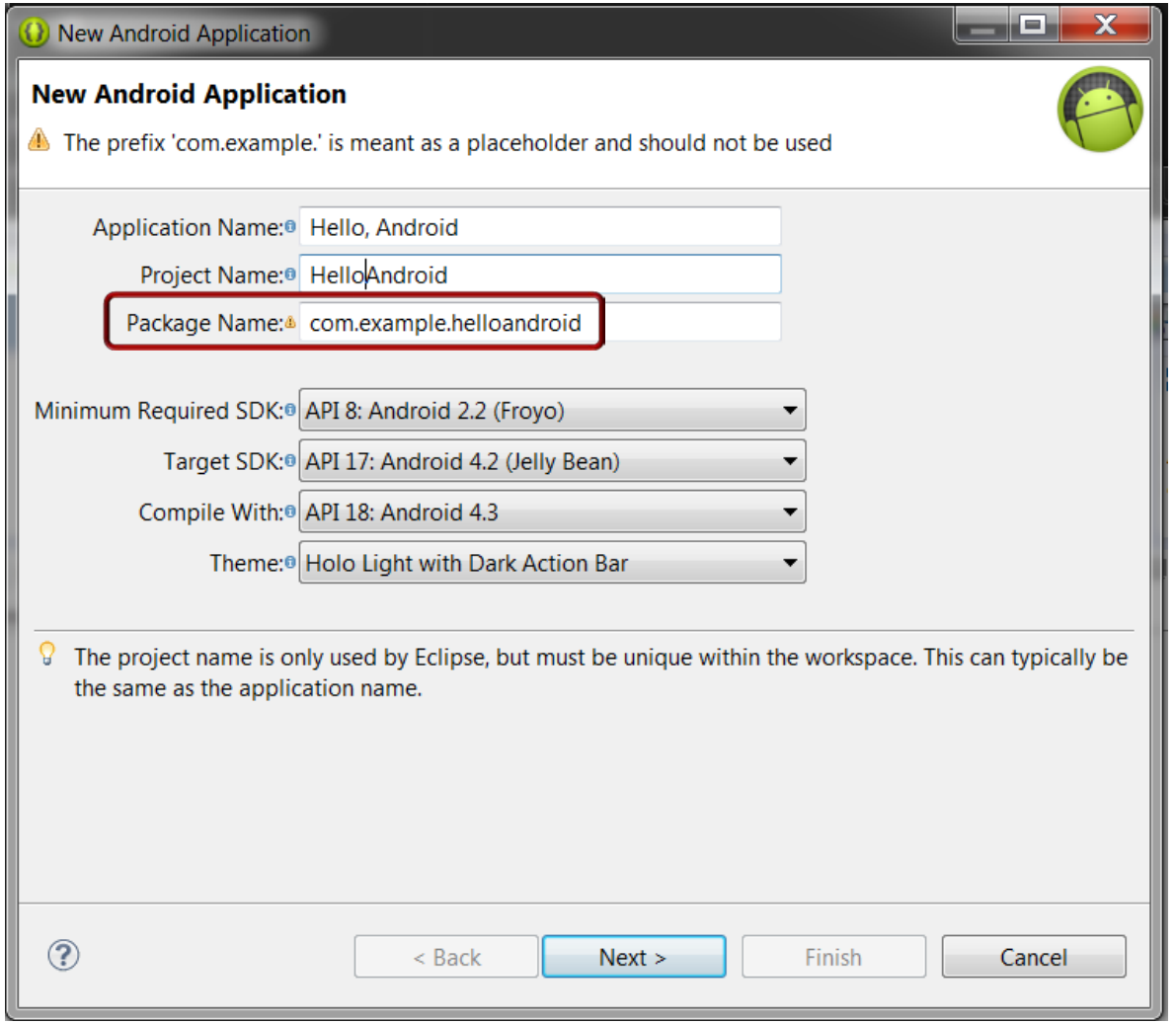
Configuring the Project Settings Via a Wizard

- The new project wizard is where you setup critical information for an app:
  - You specify the name of the app & the project



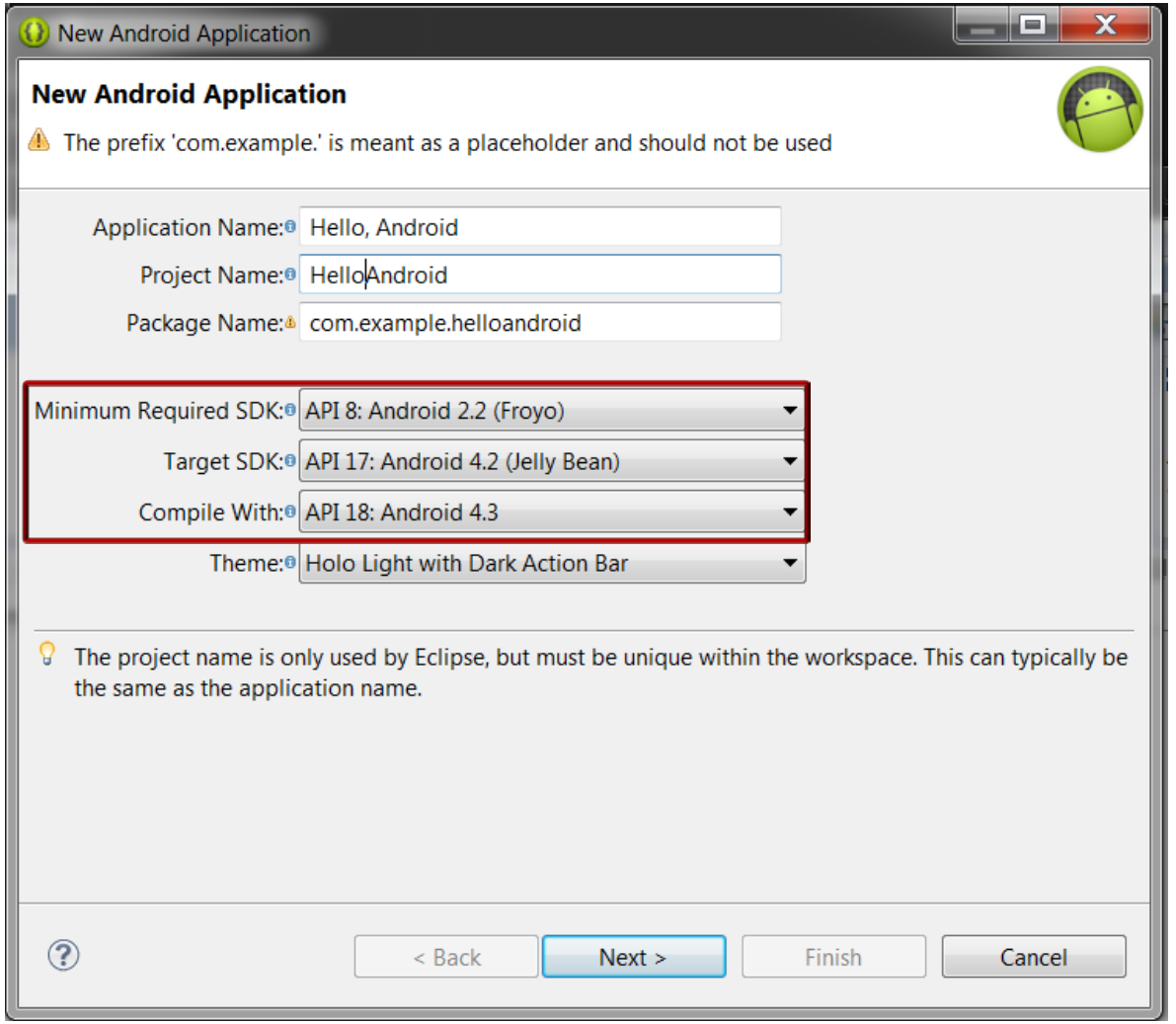
Configuring the Project Settings Via a Wizard

- The new project wizard is where you setup critical information for an app:
  - You specify the name of the app & the project
  - You provide a Java package that will hold all of your app's code



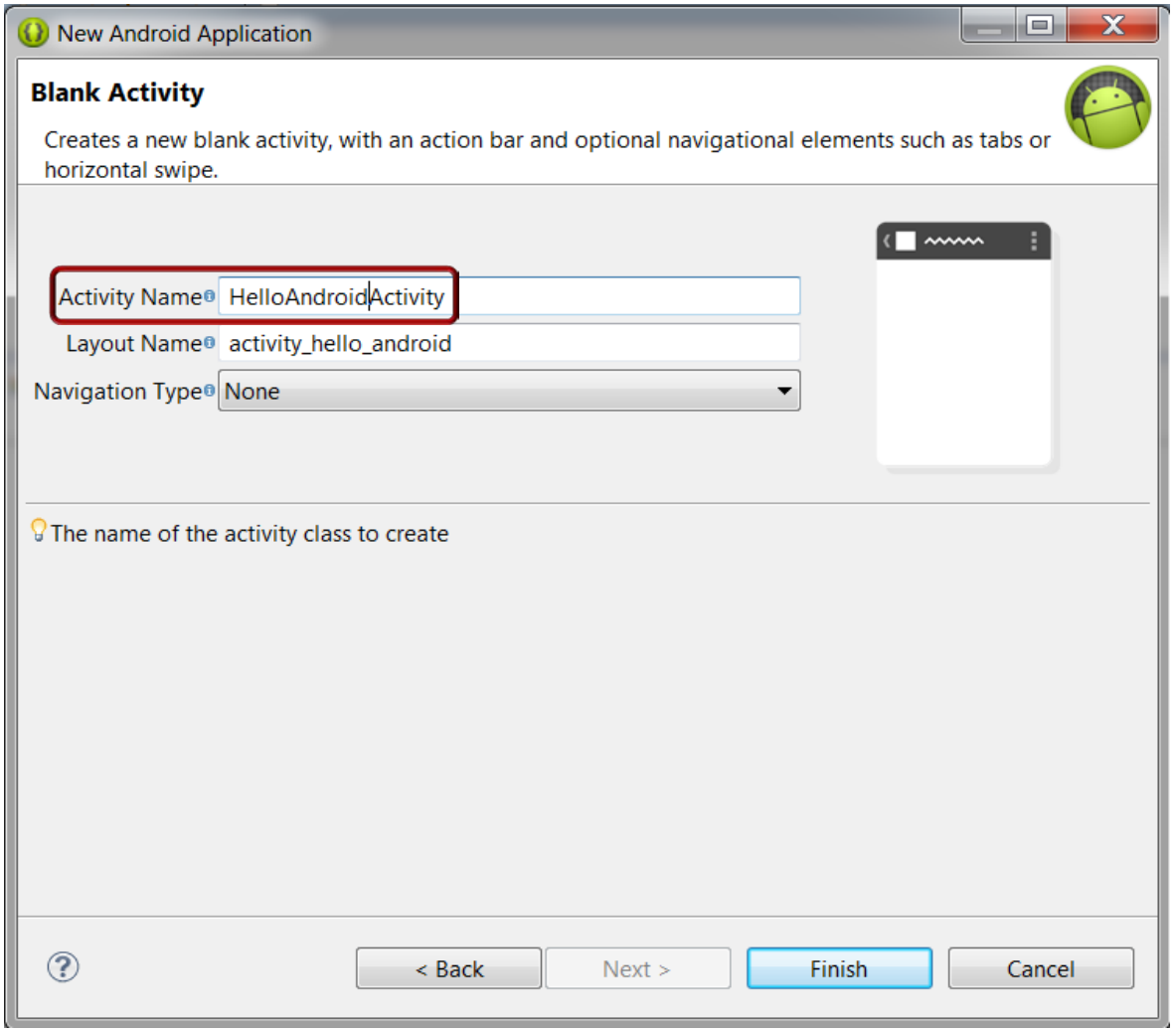
Configuring the Project Settings Via a Wizard

- The new project wizard is where you setup critical information for an app:
  - You specify the name of the app & the project
  - You provide a Java package that will hold all of your app's code
  - You need to set the version info for the Android platform that you will be targeting



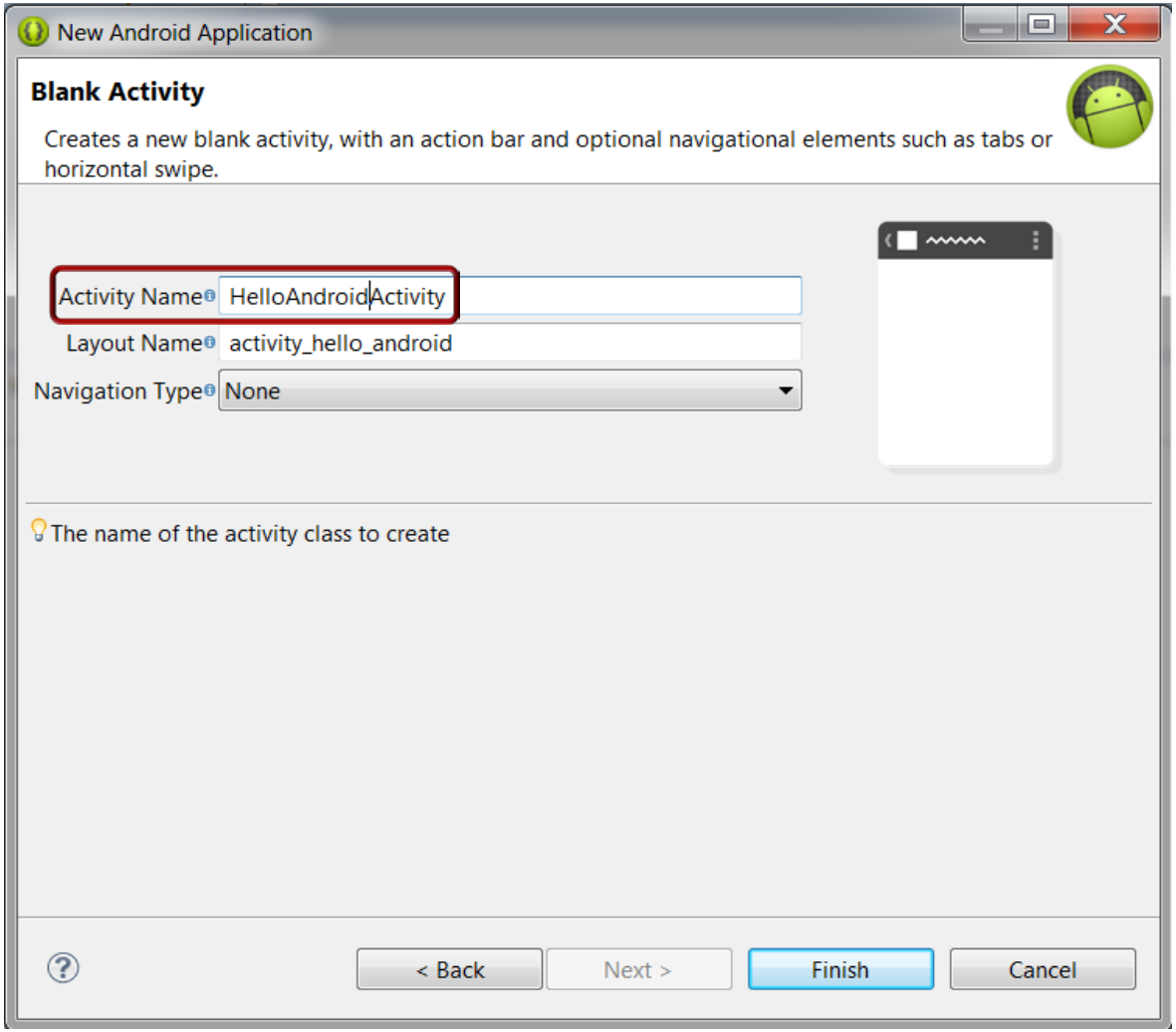
Configuring the Project Settings Via a Wizard

- The new project wizard is where you setup critical information for an app:
  - You specify the name of the app & the project
  - You provide a Java package that will hold all of your app's code
  - You need to set the version info for the Android platform that you will be targeting
  - You specify a default Activity for your project



Configuring the Project Settings Via a Wizard

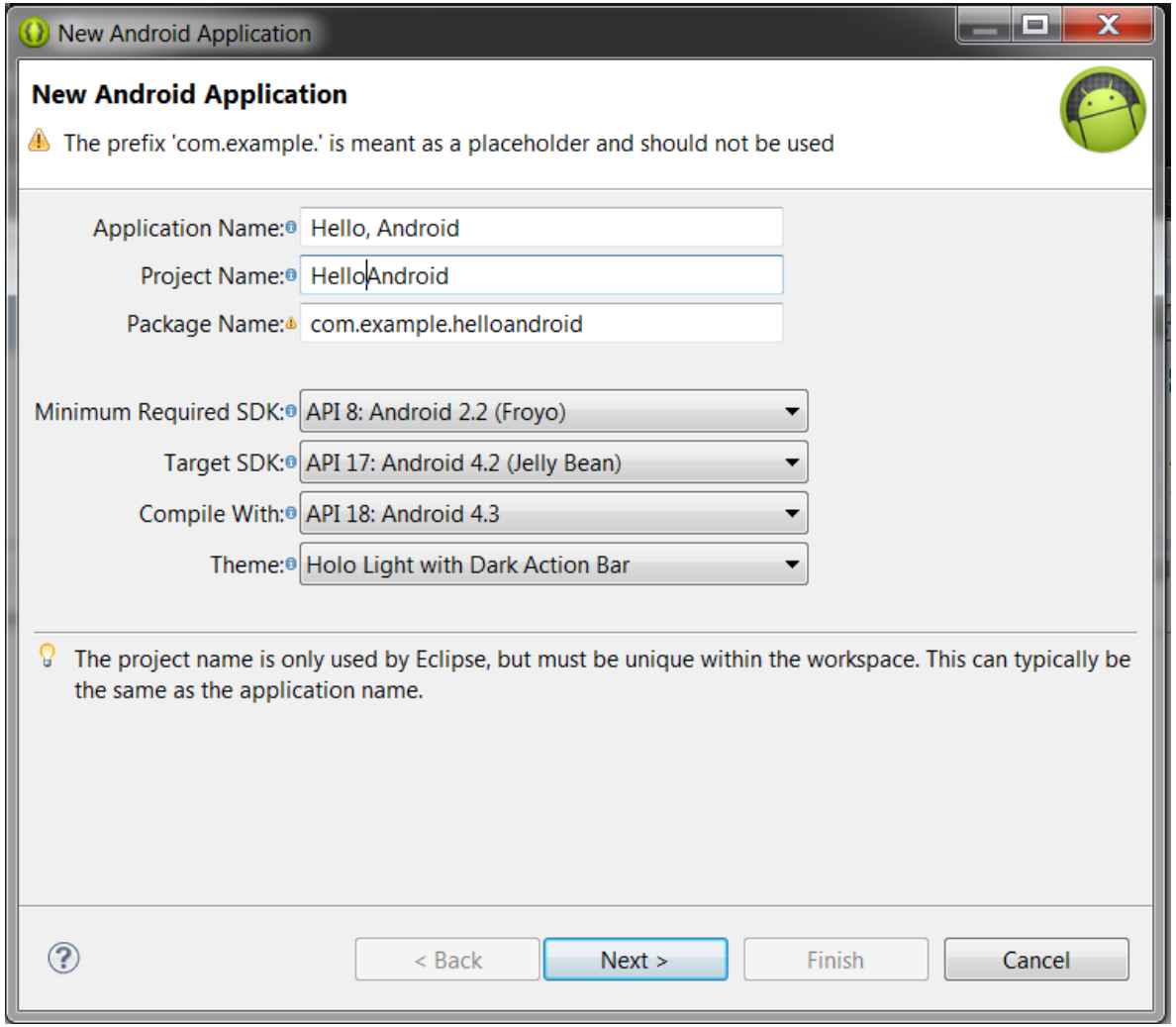
- The new project wizard is where you setup critical information for an app:
  - You specify the name of the app & the project
  - You provide a Java package that will hold all of your app's code
  - You need to set the version info for the Android platform that you will be targeting
  - You specify a default Activity for your project



You can configure this info via the wizard or via the XML directly

Summary

- Eclipse provides various tools & wizards that simplify the creation of apps

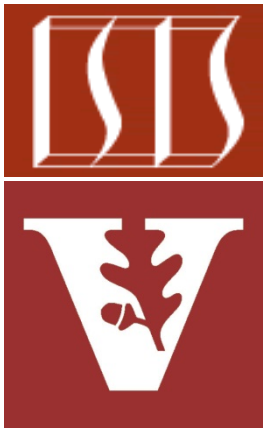


Summary

- Eclipse provides various tools & wizards that simplify the creation of apps
- It's not mandatory to use these wizards if you're comfortable working with XML directly
  - However, good luck debugging hand-written XML files ;-)

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:background="@android:color/transparent">

    <ListView android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="0px"
        android:layout_weight="1"
        android:clipToPadding="false"
        android:drawSelectorOnTop="false"
        android:cacheColorHint=
            "@android:color/transparent"
        android:scrollbarAlwaysDrawVerticalTrack="true" />

    <Button android:id="@+id/clear_all_button"
            android:layout_width="150dip"
            android:layout_height="wrap_content"
            android:layout_margin="5dip"
            android:text=
               "@string/website_settings_clear_all"
            android:visibility="gone" />
</LinearLayout>
```

Developing Android Apps: Part 3



Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

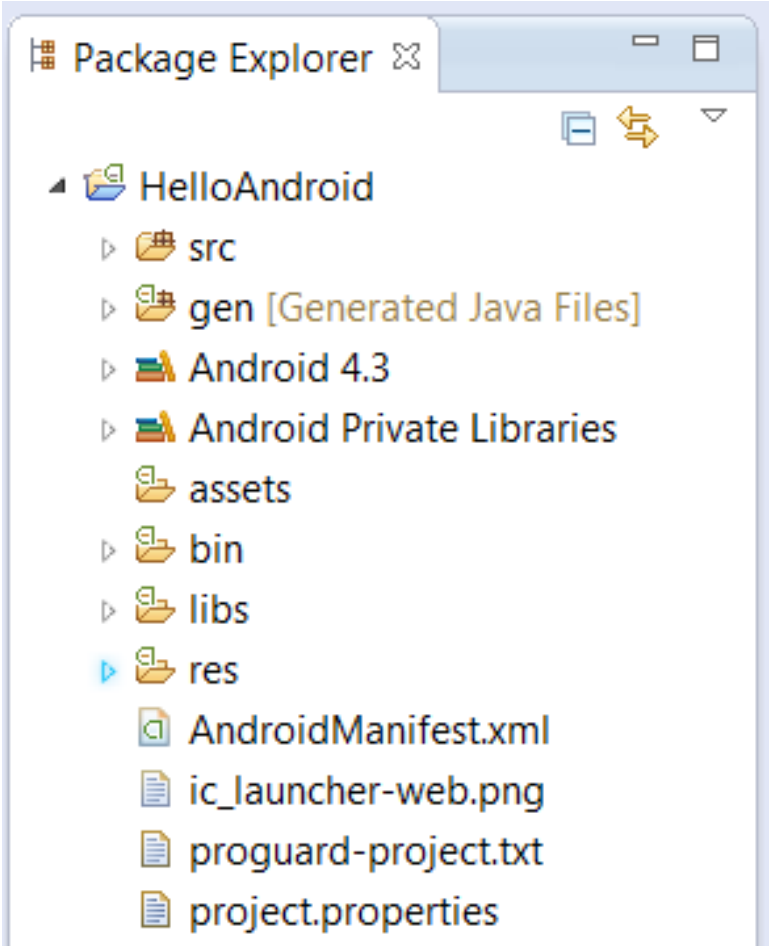
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

Systems Programming for Android

Learning Objectives in this Part of the Module

- Understand the main parts of an Android project



Overview of Android Projects

- Projects are containers that storing things like code & resource files

src/
Contains your stub Activity file. All other source code files (such as `.java` or `.aidl` files) go here as well.

bin/
Output directory of the build. This is where you can find the final `.apk` file and other compiled resources.

jni/
Contains native code sources developed using the Android NDK.

gen/
Contains the Java files generated by ADT, such as your `R.java` file and interfaces created from AIDL files.

assets/
You can use this to store raw asset files, such as textures and game data.

res/
Contains application resources, such as drawable files, layout files, and string values..

libs/
Contains private libraries

...

Overview of Android Projects

- Projects are containers that storing things like code & resource files
- Android projects eventually get built into an .apk file that can be installed onto a device

src/
Contains your stub Activity file. All other source code files (such as `.java` or `.aidl` files) go here as well.

bin/
Output directory of the build. This is where you can find the final `.apk` file and other compiled resources.

jni/
Contains native code sources developed using the Android NDK.

gen/
Contains the Java files generated by ADT, such as your `R.java` file and interfaces created from AIDL files.

assets/
You can use this to store raw asset files, such as textures and game data.

res/
Contains application resources, such as drawable files, layout files, and string values..

libs/
Contains private libraries

...

Overview of Android Projects

- Projects are containers that storing things like code & resource files
- Android projects eventually get built into an .apk file that can be installed onto a device
- Some are generated for you by default, while others should be created if required

src/
Contains your stub Activity file. All other source code files (such as `.java` or `.aidl` files) go here as well.

bin/
Output directory of the build. This is where you can find the final `.apk` file and other compiled resources.

jni/
Contains native code sources developed using the Android NDK.

gen/
Contains the Java files generated by ADT, such as your `R.java` file and interfaces created from AIDL files.

assets/
You can use this to store raw asset files, such as textures and game data.

res/
Contains application resources, such as drawable files, layout files, and string values..

libs/
Contains private libraries

...

developer.android.com/tools/projects has more on Android projects

Three Key Elements in an Android Project

- Each Android project contains three key elements
  - Java source code

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.widget.TextView;

public class HelloAndroidActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
```

This part of the app is typically the most “free-form” & creative

Three Key Elements in an Android Project

- Each Android project contains three key elements
  - Java source code
  - XML-based GUI metadata to manage layouts, etc.

```
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_height="match_parent"
    android:layout_width="match_parent">
   <Button android:id="@+id/mapButton"
           android:layout_gravity="bottom"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Find Address">
   </Button>
</LinearLayout>
```

You can write this metadata manually or generate it via a layout editor

Three Key Elements in an Android Project

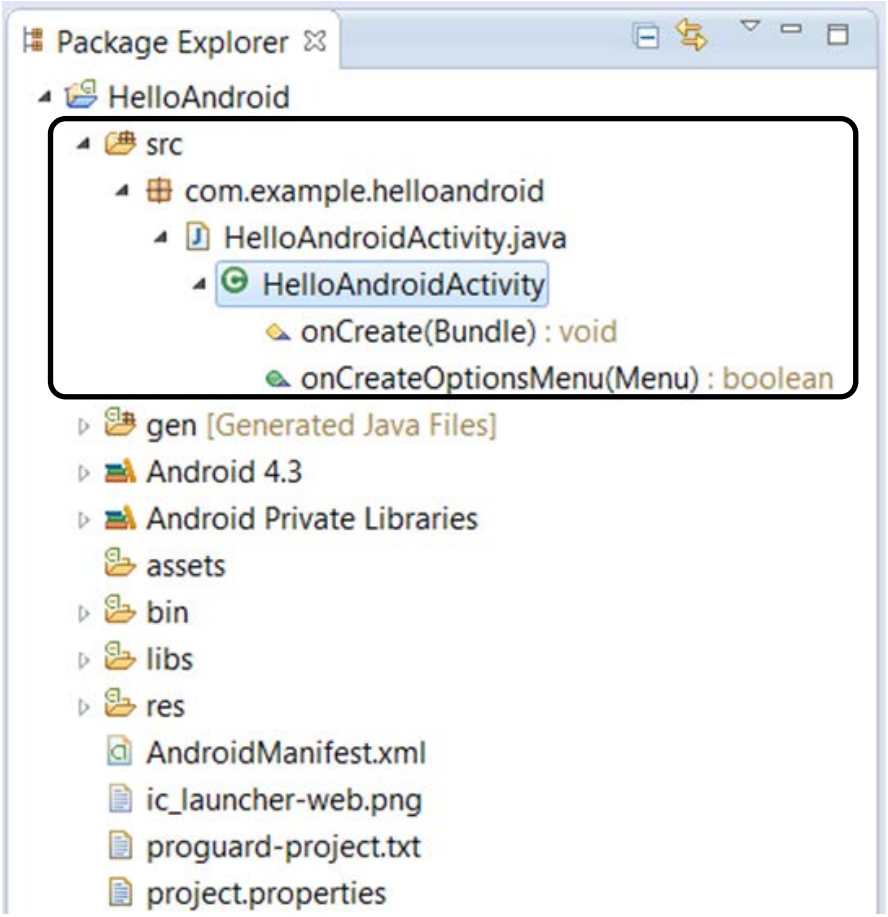
- Each Android project contains three key elements
  - Java source code
  - XML-based GUI metadata to manage layouts, etc.
  - An XML Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest> ...
    <application>
        <activity>
            <intent-filter> <action /> ... <data /> </intent-filter> ...
        </activity>
        <service> <intent-filter> .... </intent-filter> </service>
        <receiver> <intent-filter> ... </intent-filter> </receiver>
        <provider> <grant-uri-permission /> </provider> ...
    </application>
</manifest>
```

The manifest presents essential information about the app to Android

Android Project Anatomy: "src"

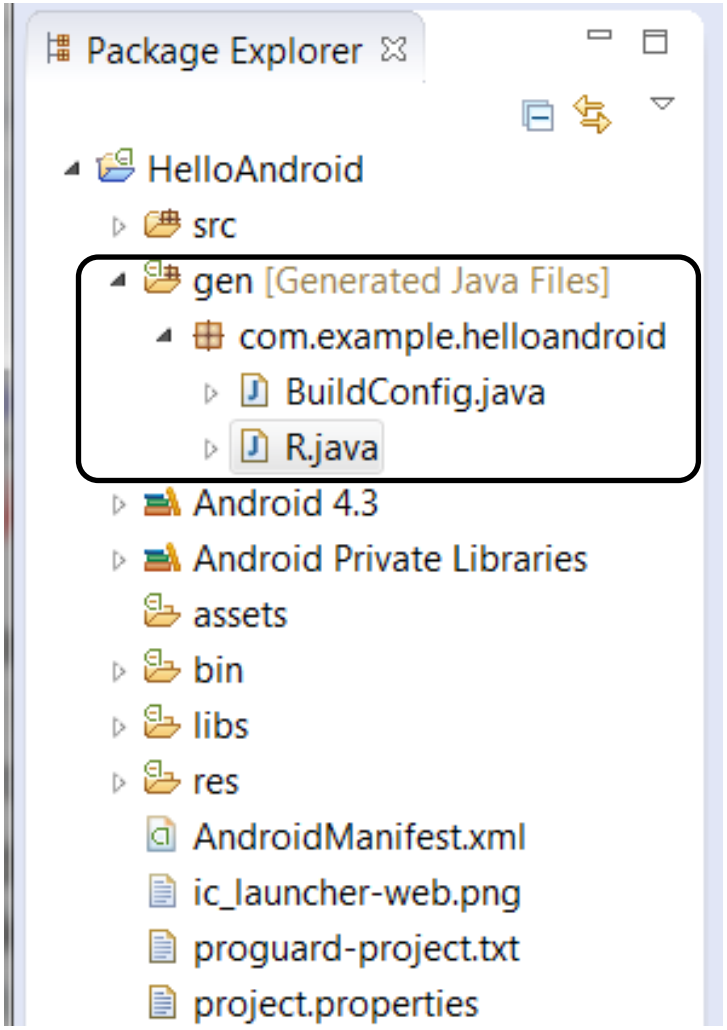
- The App source code resides inside the "src" folder in the package that you specified in the new Android project wizard



Eclipse usually (re)builds files properly, but it can sometimes do weird things..

Android Project Anatomy: "gen"

- Android generates some files that make it easier for you to build GUIs & fetch resources
  - The "gen" folder contains the generated code produced by the Android plugin
    - e.g., stubs generated by AIDL compiler, R.java file generated by the Android resource compiler (aapt.exe), etc.



Never put any code in this folder or modify any generated code

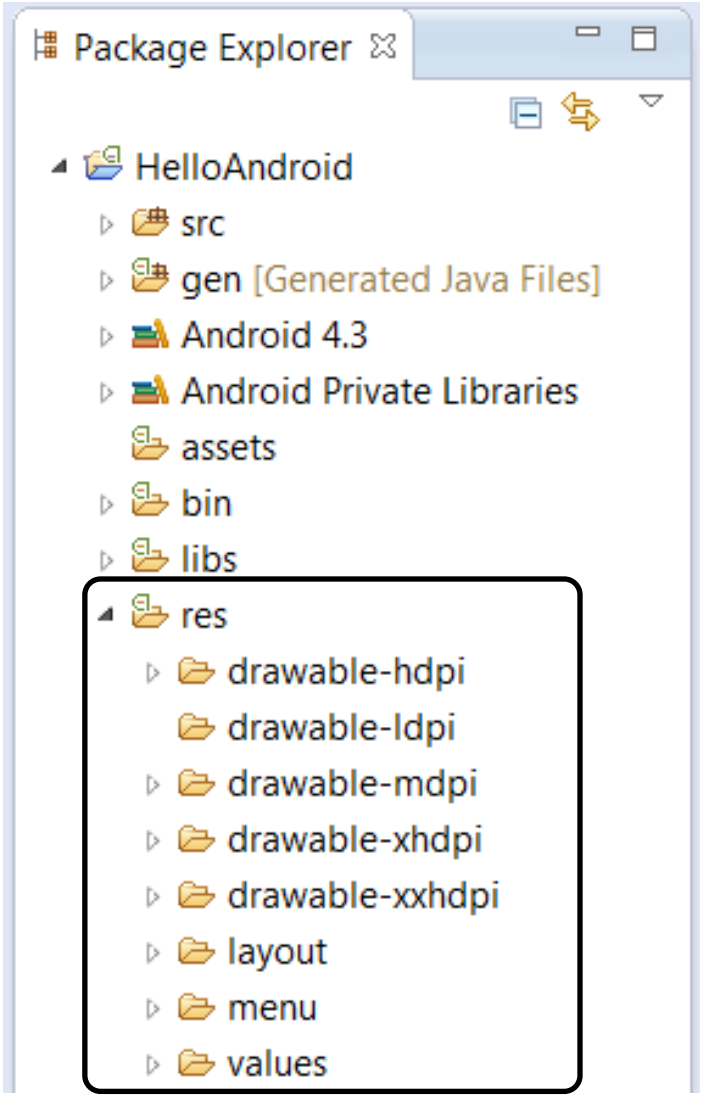
Android Project Anatomy: "gen"

- Android generates some files that make it easier for you to build GUIs & fetch resources
- An app uses the "R.java" file to fetch GUI resources & widgets
  - We'll discuss this file later

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.

package com.example.helloandroid;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        /**  Default screen margins, per the Android Design guidelin

         Customize dimensions originally defined in res/values/dimen
         screen margins) for sw720dp devices (e.g. 10" tablets) in l

         */
        public static final int activity_horizontal_margin=0x7f04000
        public static final int activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_hello_android=0x7f030000;
    }
    public static final class menu {
        public static final int hello_android=0x7f070000;
    }
    public static final class string {
        public static final int action_settings=0x7f050001;
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050002;
    }
```

Android Project Anatomy: "res"

- The "res" folder contains non-code resources (e.g., layouts, menus, images, etc.) used by your app
  - You can use image resolutions for different screen sizes based on contents in "drawable-*" folders



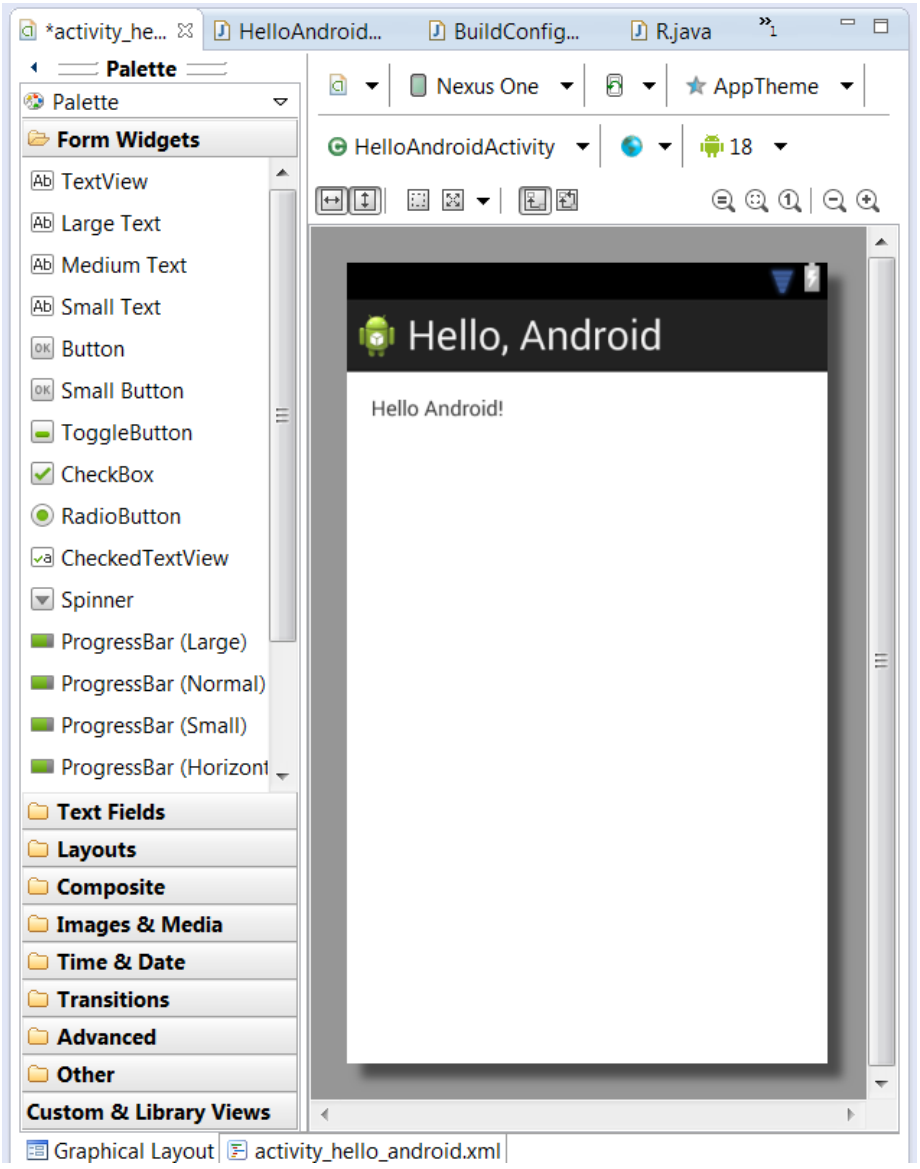
developer.android.com/guide/practices/screens_support.html has more info

Android Project Anatomy: "res"

- The "res" folder contains non-code resources (e.g., layouts, menus, images, etc.) used by your app
  - You can use image resolutions for different screen sizes based on contents in "drawable-*" folders
- You can define your GUI using XML or the Android layout editor

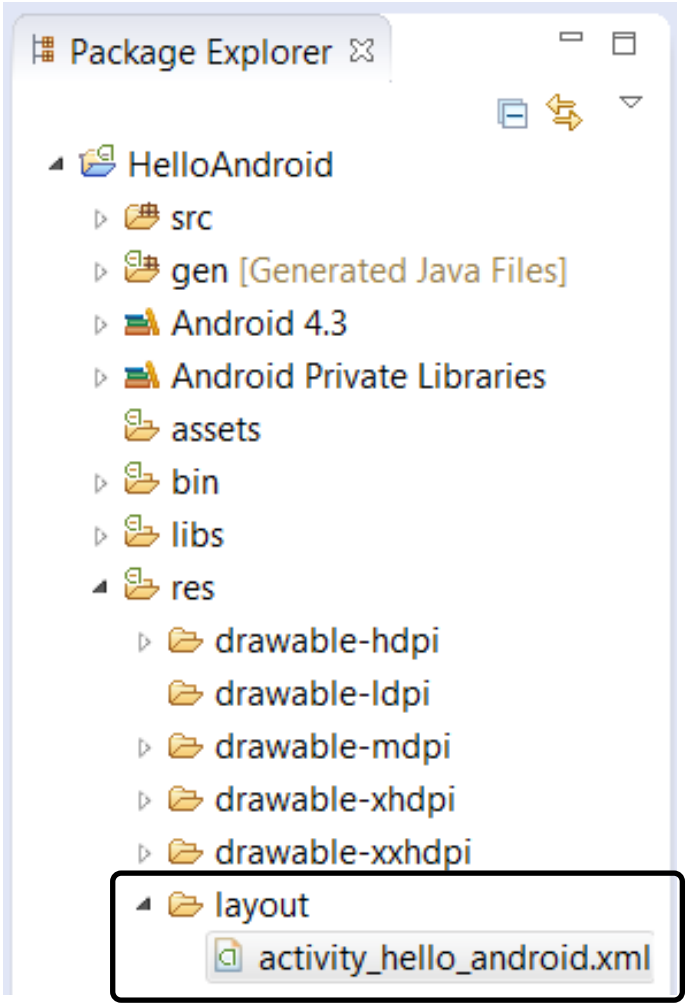
activity_hel... | HelloAndroid... | strings.xml | styles.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".HelloAndroidActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_android" />
</RelativeLayout>
```



Android Project Anatomy: "res"

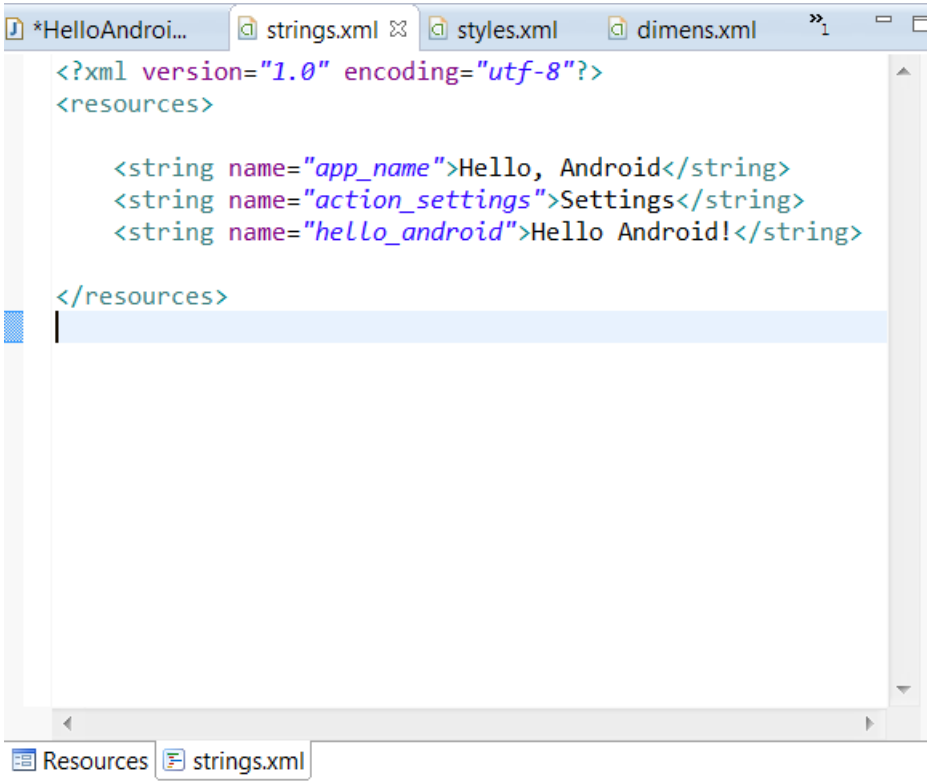
- The "res" folder contains non-code resources (e.g., layouts, menus, images, etc.) used by your app
  - You can use image resolutions for different screen sizes based on contents in "drawable-*" folders
- You can define your GUI using XML or the Android layout editor
  - The various XML files are located beneath the layout folder



developer.android.com/tools/help/adt.html#graphical-editor has more info

Android Project Anatomy: "res"

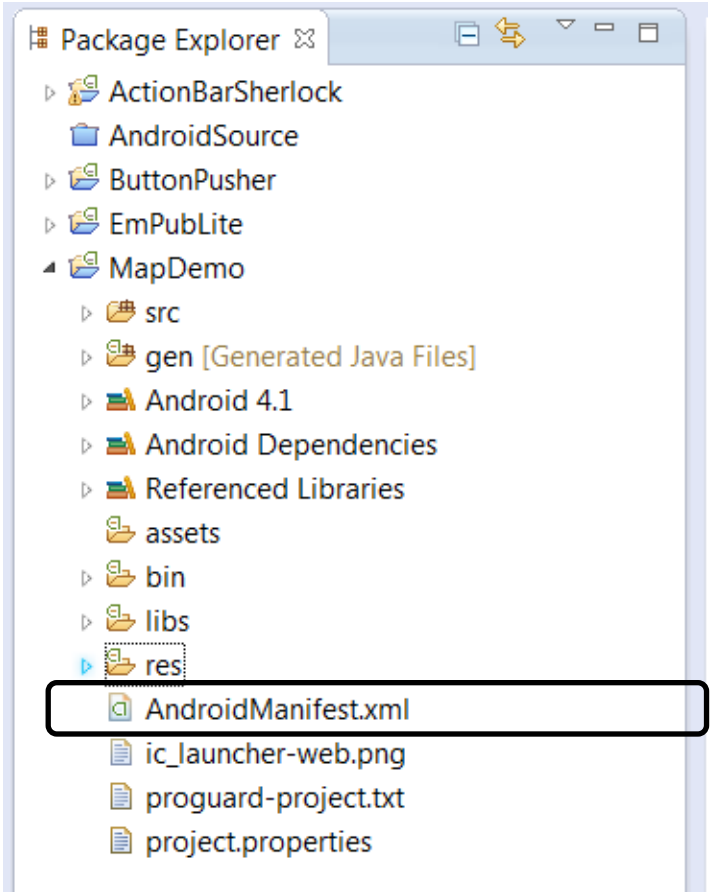
- The "res" folder contains non-code resources (e.g., layouts, menus, images, etc.) used by your app
  - You can use image resolutions for different screen sizes based on contents in "drawable-*" folders
- You can define your GUI using XML or the Android layout editor
  - The various XML files are located beneath the layout folder
- The values/strings.xml file contains text strings for your app
  - Can optionally include text styling & formatting via HTML tags



developer.android.com/guide/topics/resources/string-resource.html has more

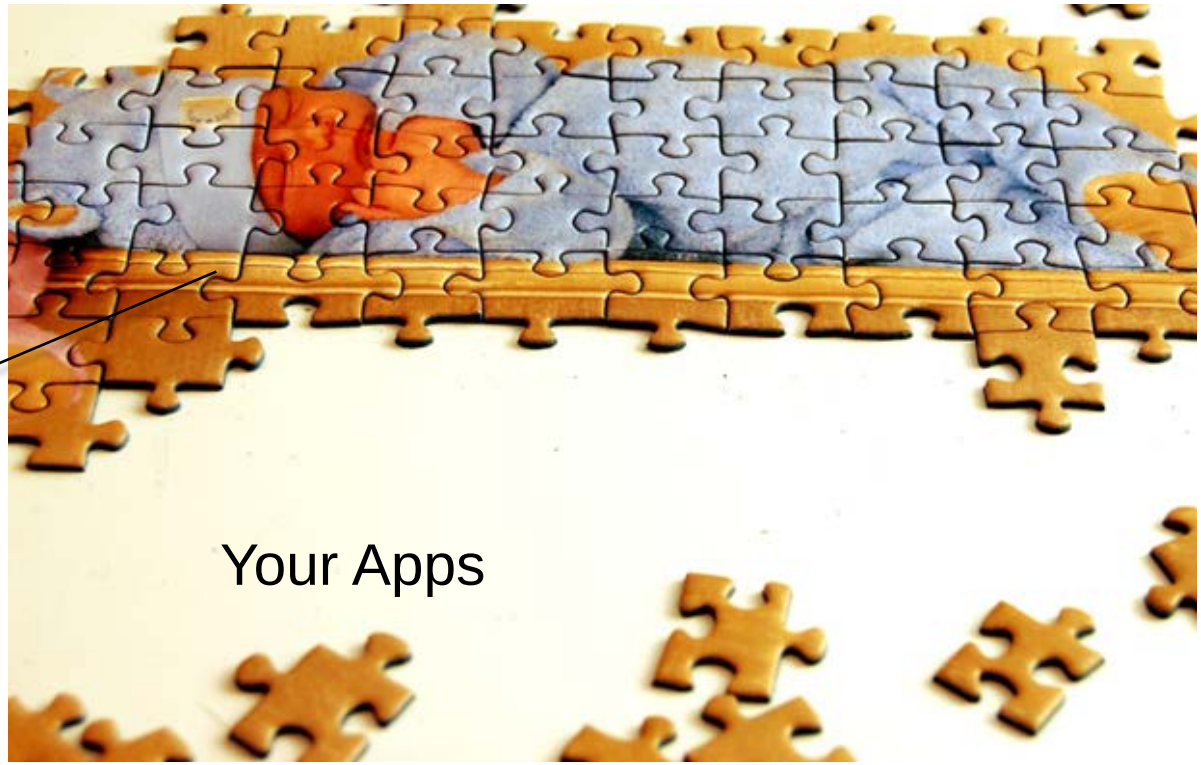
Android Project Anatomy: Manifest File

- The AndroidManifest.xml file contains information Android needs to execute your app



Android Project Anatomy: Manifest File

- The AndroidManifest.xml file contains information Android needs to execute your app
- Since Android provides an App framework it has to know how to plug your App's components into the framework

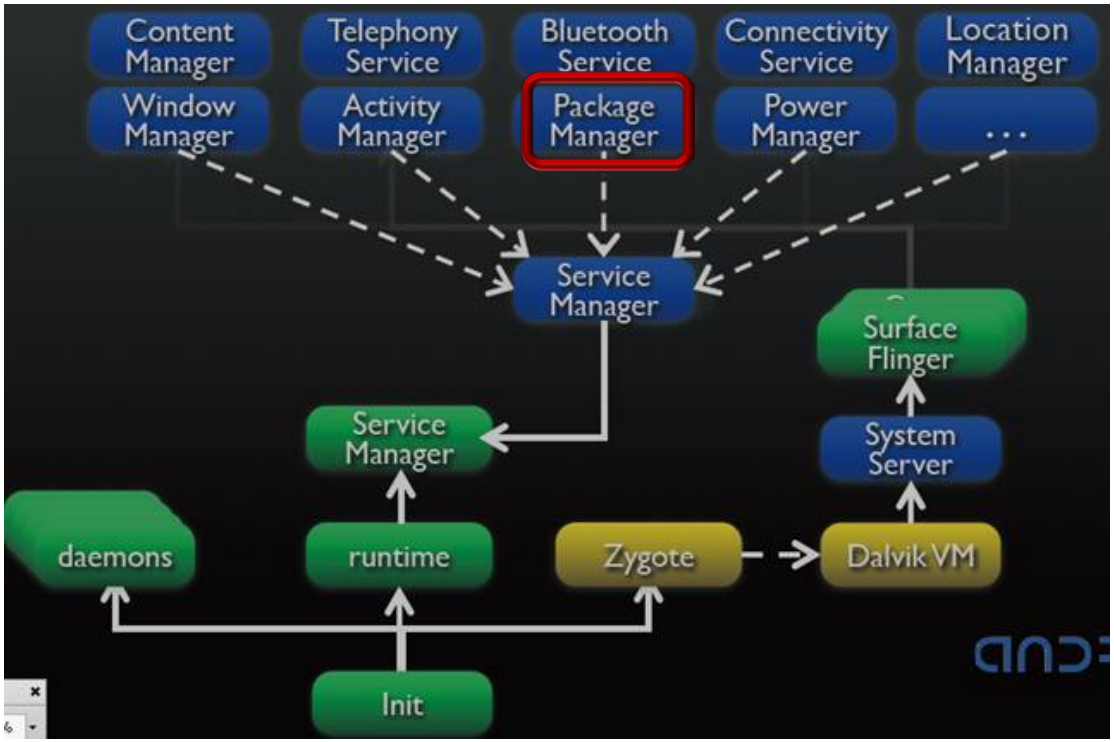


The manifest file tells Android how your app "plugs-in" to the framework

developer.android.com/guide/topics/manifest/manifest-intro.html has more

Android Project Anatomy: Manifest File

- The AndroidManifest.xml file contains information Android needs to execute your app
- Since Android provides an App framework it has to know how to plug your App's components into the framework



- When you install an App, the PackageManager reads your manifest file & populates various internal data structures

```
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(intent, 0);
boolean isIntentSafe = activities.size() > 0;
```

developer.android.com/reference/android/content/pm/PackageManager.html

Android Project Anatomy: Manifest File

- The manifest file contains various important sections, including:
  - App name/info, required platform version & minimum API level
  - The list of Activity, Service, Content Provider, & Broadcast Receiver components defined by your App & events that your App cares about
  - The security permissions your App is requesting
  - Whether or not your App can be debugged when deployed

```
<manifest … package="com.example.helloandroid"
    android:versionCode="1" android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8“ android:targetSdkVersion="17" />
    <application … android:label="@string/app_name">
        <activity
            android:name="com.example.helloandroid.HelloAndroidActivity"
            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>
```

developer.android.com/guide/topics/fundamentals.html#Manifest has more

Summary

- Projects act as containers for storing things such as code & resource files

`src/`
Contains your stub Activity file. All other source code files (such as `.java` or `.aidl` files) go here as well.

`bin/`
Output directory of the build. This is where you can find the final `.apk` file and other compiled resources.

`jni/`
Contains native code sources developed using the Android NDK.

`gen/`
Contains the Java files generated by ADT, such as your `R.java` file and interfaces created from AIDL files.

`assets/`
You can use this to store raw asset files, such as textures and game data.

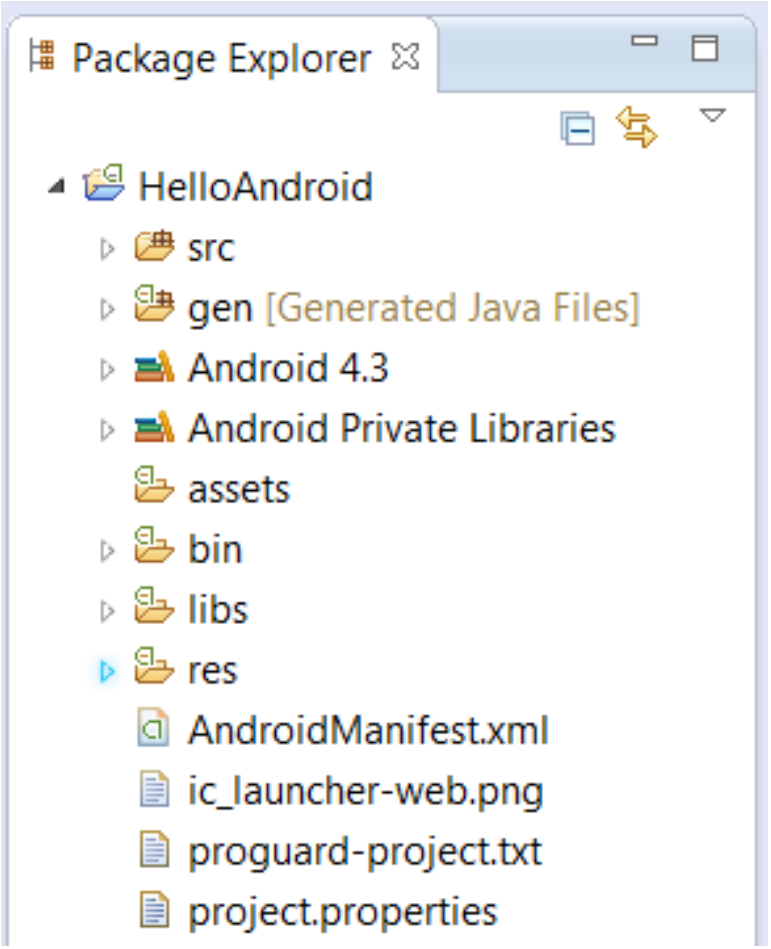
`res/`
Contains application resources, such as drawable files, layout files, and string values..

`libs/`
Contains private libraries

...

Summary

- Projects act as containers for storing things such as code & resource files
- The SDK tools expect your projects to follow a specific structure so it can compile & package your application correctly
  - It is highly recommended that you create them with Eclipse & ADT or with the android tool on the command line



Summary

- Projects act as containers for storing things such as code & resource files
- The SDK tools expect your projects to follow a specific structure so it can compile & package your application correctly
- There are two other types of projects

Test Projects

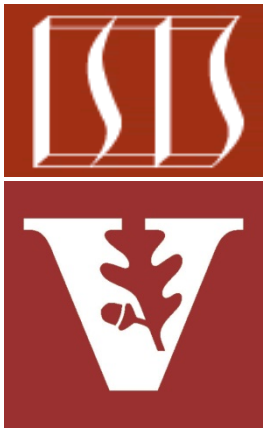
These projects contain code to test your application projects and are built into applications that run on a device.

Library Projects

These projects contain shareable Android source code and resources that you can reference in Android projects. This is useful when you have common code that you want to reuse. Library projects cannot be installed onto a device, however, they are pulled into the `.apk` file at build time.

developer.android.com/tools/projects has more on Android projects

Developing Android Apps: Part 4



Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

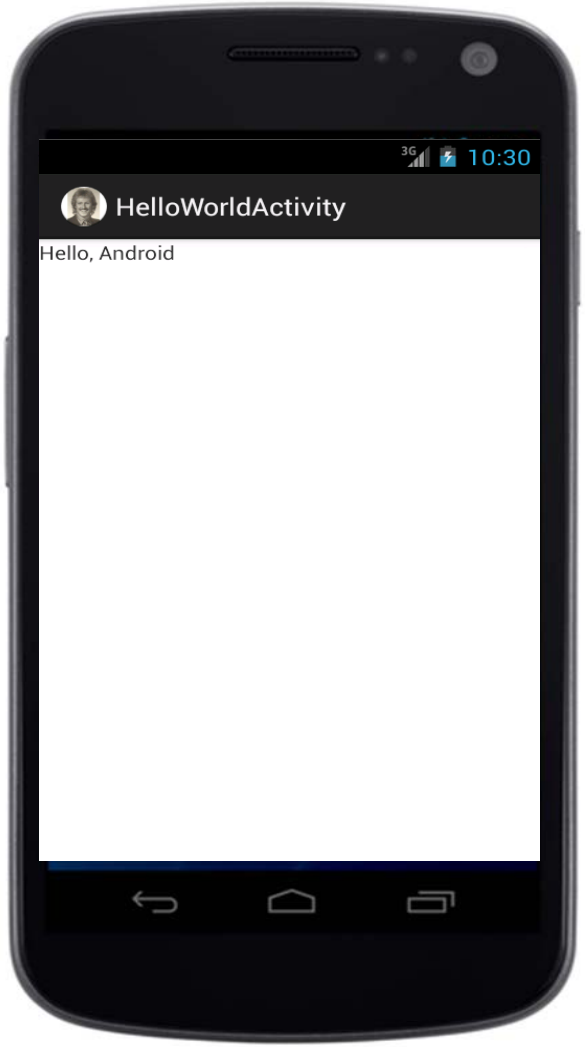
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

Systems Programming for Android

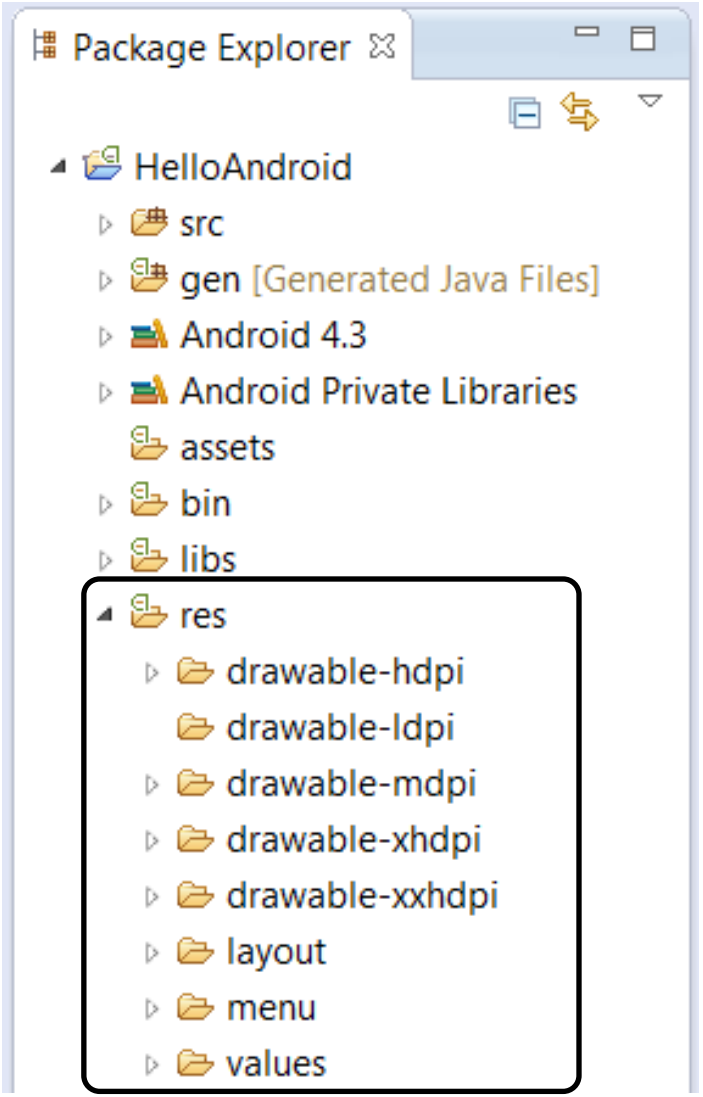
Learning Objectives in this Part of the Module

- Understand how to use Eclipse to create the simple “Hello Android” app by
  - Defining resources
  - Implementing user-defined classes



Define Resources for "Hello Android" App

- Several types of resources can be defined
  - Layout
  - Strings
  - Images
  - Menus
  - etc.



developer.android.com/guide/topics/resources has more info

Defining & Using App Layout Resources

- User interface layout specified in XML file stored in res/layout/<filename>.xml
  - With Eclipse can also do layout visually (but beware of limitations)

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
     android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".HelloAndroidActivity" >

     <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/hello_android" />
</RelativeLayout>
```

Defining & Using App Layout Resources

- User interface layout specified in XML file stored in res/layout/<filename>.xml
- Accessed from R.layout class

```
public class MyActivity extends Activity
{
   public void onCreate(Bundle savedInstanceState)
   {
        super.onCreate(savedInstanceState);
        ...
        setContentView(R.layout.main);
   }
   ...
}
```

Defining & Using App String Resources

- Types
  - String
  - String Array
  - Plurals
- Can include style & formatting
  - Stored in res/values/<filename>.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">
         Hello, Android</string>
    <string name="action_settings">
         Settings</string>
    <string name="hello_android">
        Hello Android!</string>
</resources>
```

Defining & Using App String Resources

- Types
  - String
  - String Array
  - Plurals
- Can include style & formatting
  - Stored in res/values/<filename>.xml
- Each string references as @string/string_name in other *.xml files

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">
         Hello, Android</string>
    <string name="action_settings">
         Settings</string>
    <string name="hello_android">
        Hello Android!</string>
</resources>
```

res/layout/activity_hello_android.xml

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

Defining & Using App String Resources

- Types
  - String
  - String Array
  - Plurals
- Can include style & formatting
  - Stored in res/values/<filename>.xml
- Each string specified as @string/string_name
- Accessed as R.string.string_name

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">
         Hello, Android</string>
    <string name="action_settings">
         Settings</string>
    <string name="hello_android">
        Hello Android!</string>
</resources>
```

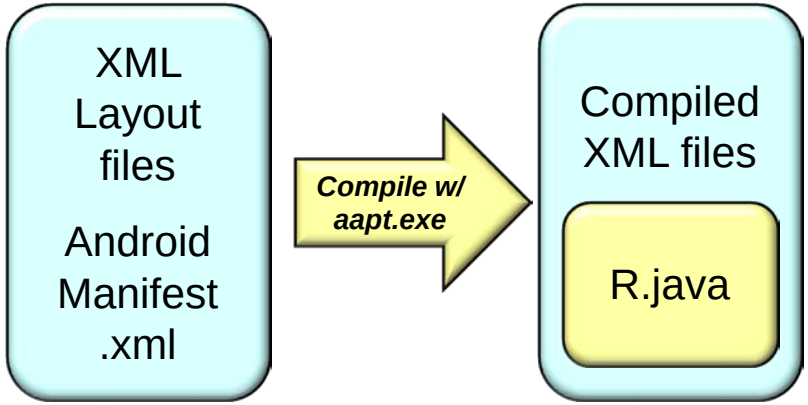
res/layout/activity_hello_android.xml

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

```
tv.setText(R.string.hello_android);
```

Defining & Using R.java Resources

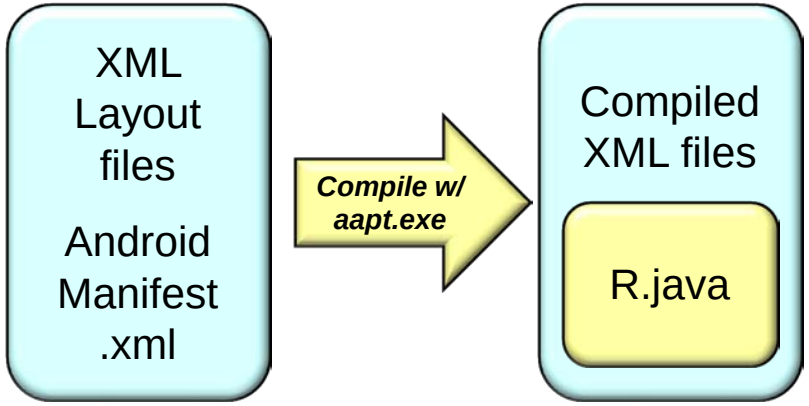
- At compilation time, resources are used to generate the R.java class




```
public final class R {
  public static final class layout {
    public static final int
      activity_hello_android
                  =0x7f030000;
  }
  public static final class string {
    public static final int
      action_settings=0x7f050001;
    public static final int
      app_name=0x7f050000;
    public static final int
      hello_android=0x7f050002;
  }
  ...
```

Defining & Using R.java Resources

- At compilation time, resources are used to generate the R.java class



- App access resources through the R class

```
public final class R {
  public static final class layout {
    public static final int
      activity_hello_android
                  =0x7f030000;
  }
  public static final class string {
    public static final int
      action_settings=0x7f050001;
    public static final int
      app_name=0x7f050000;
    public static final int
      hello_android=0x7f050002;
  }
  ...
```

```
setContentView(R.layout.main);
tv.setText(R.string.hello_android);
```

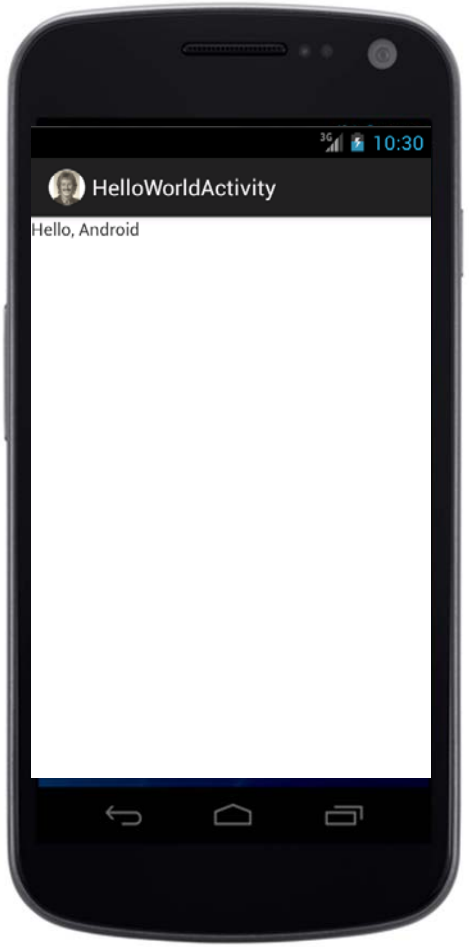
Implement User-defined Classes

- "HelloAndroid.java" implements the main Activity for the app
  - All App Activities inherit from com.android.Activity

```
package com.example.helloworld;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.widget.TextView;

public class HelloAndroidActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      TextView tv = new TextView(this);
      tv.setText(R.string.hello_android);
      setContentView(tv);
   }
}
```

Implement User-defined Classes

- "HelloAndroid.java" implements the main Activity for the app
  - All App Activities inherit from com.android.Activity
- Each Activity manages one screen in an app
  - When you change screens, you typically change the currently active Activity



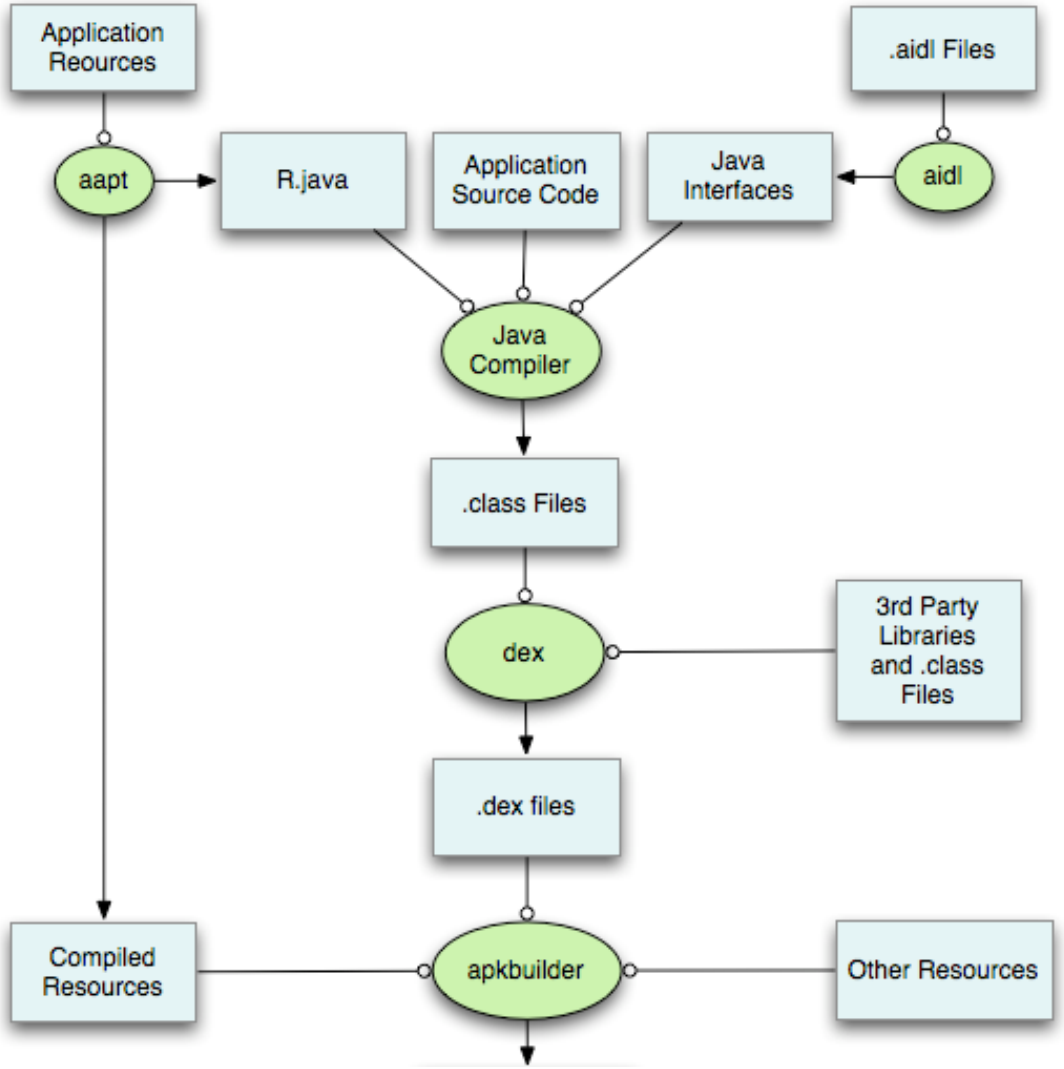
Implement User-defined Classes

- "HelloAndroid.java" implements the main Activity for the app
  - All App Activities inherit from com.android.Activity
- Each Activity manages one screen in an app
- Most apps have a default Activity that serves as the entry point to the app

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

Summary

- The Java source code is combined with others files generated by various tools in the Android/Eclipse tool chain to create a signed *.apk package



developer.android.com/guide/developing/building has more info on this process

Summary

- The Java source code is combined with others files generated by various tools in the Android/Eclipse tool chain to create a signed *.apk package