Android Development Tool

Dr. Abraham (Abey) Campbell

Outline

Title: Android Development Tool	Module: COMP2007L	Number in sequence: 2

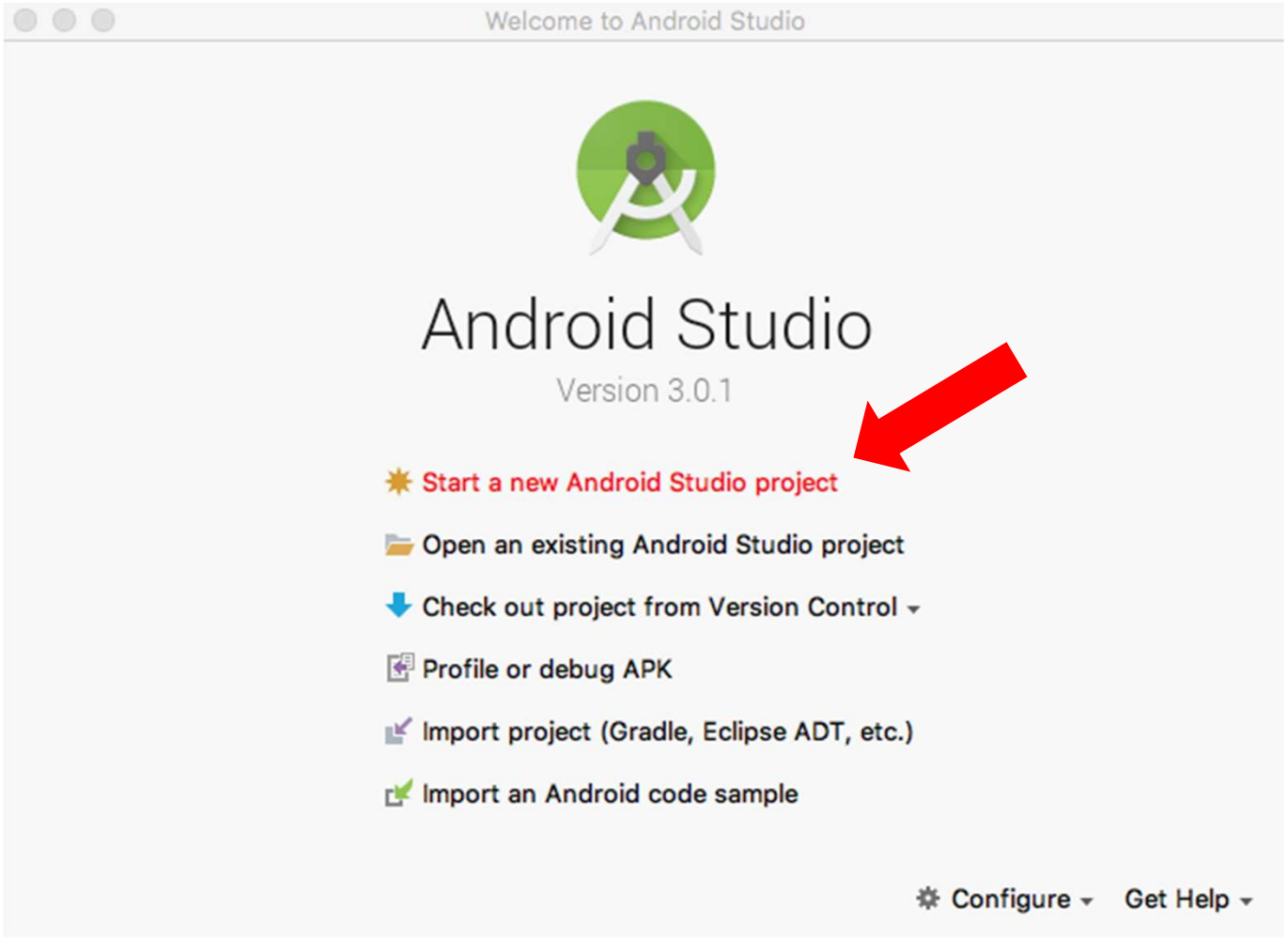
Aim(s): Present Android Studio and describe an Android studio project

Learning Objectives:

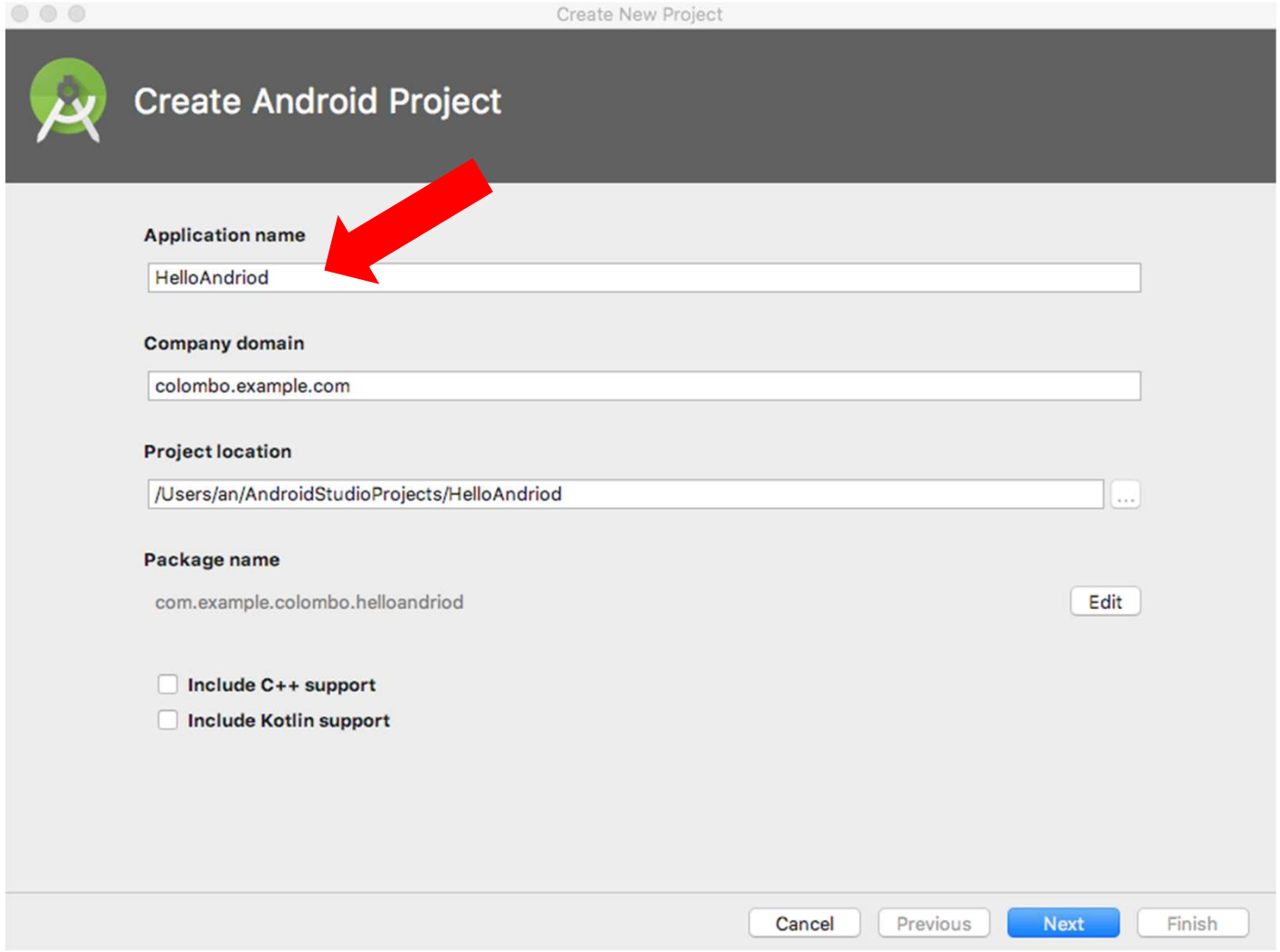
At the end of this lecture, you will be able to:

- Use Android Studio
- Create an Android project
- Explain Android project files
- Run Android projects
- Use Android tools

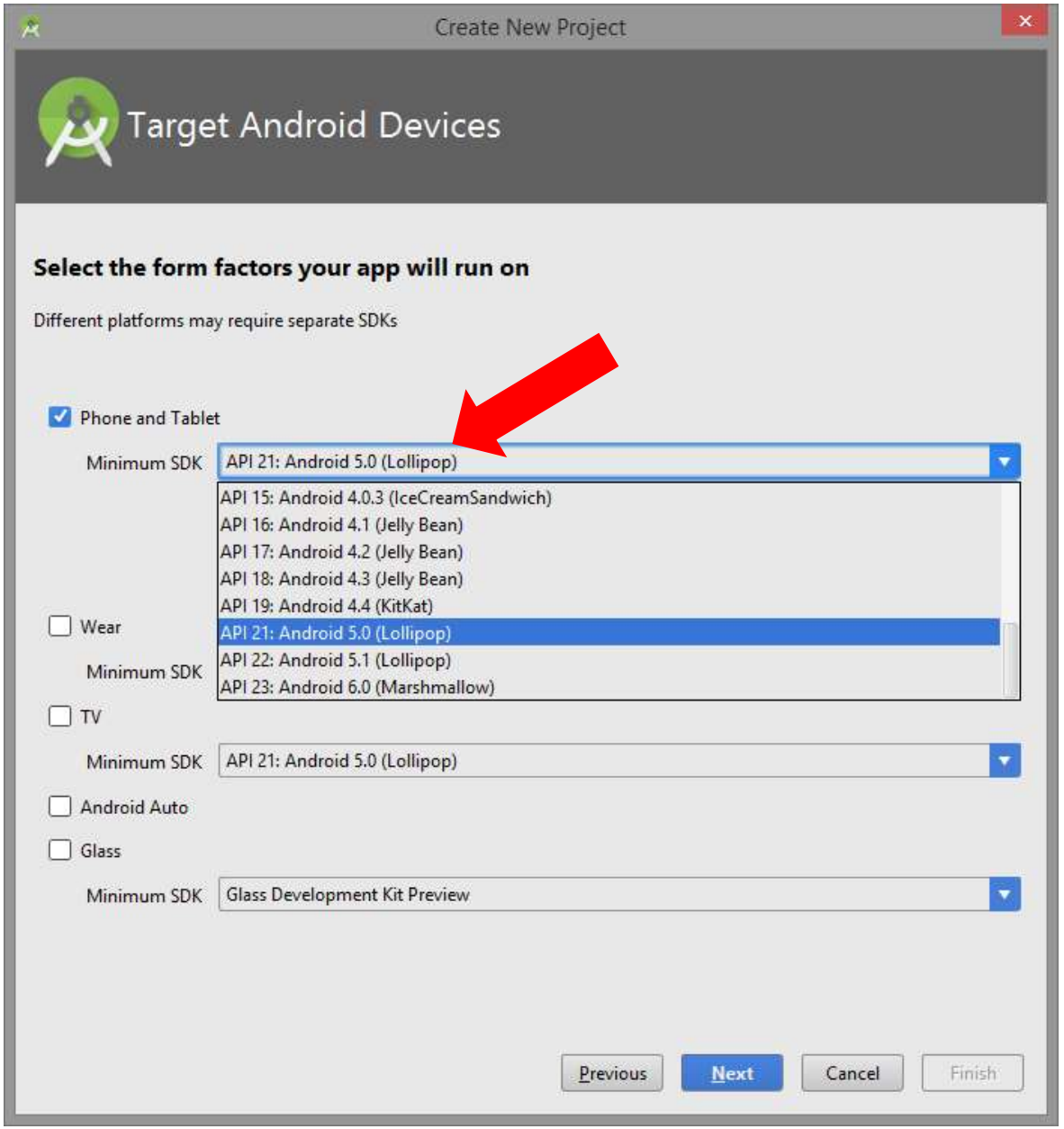
Creating an Android project Android Studio



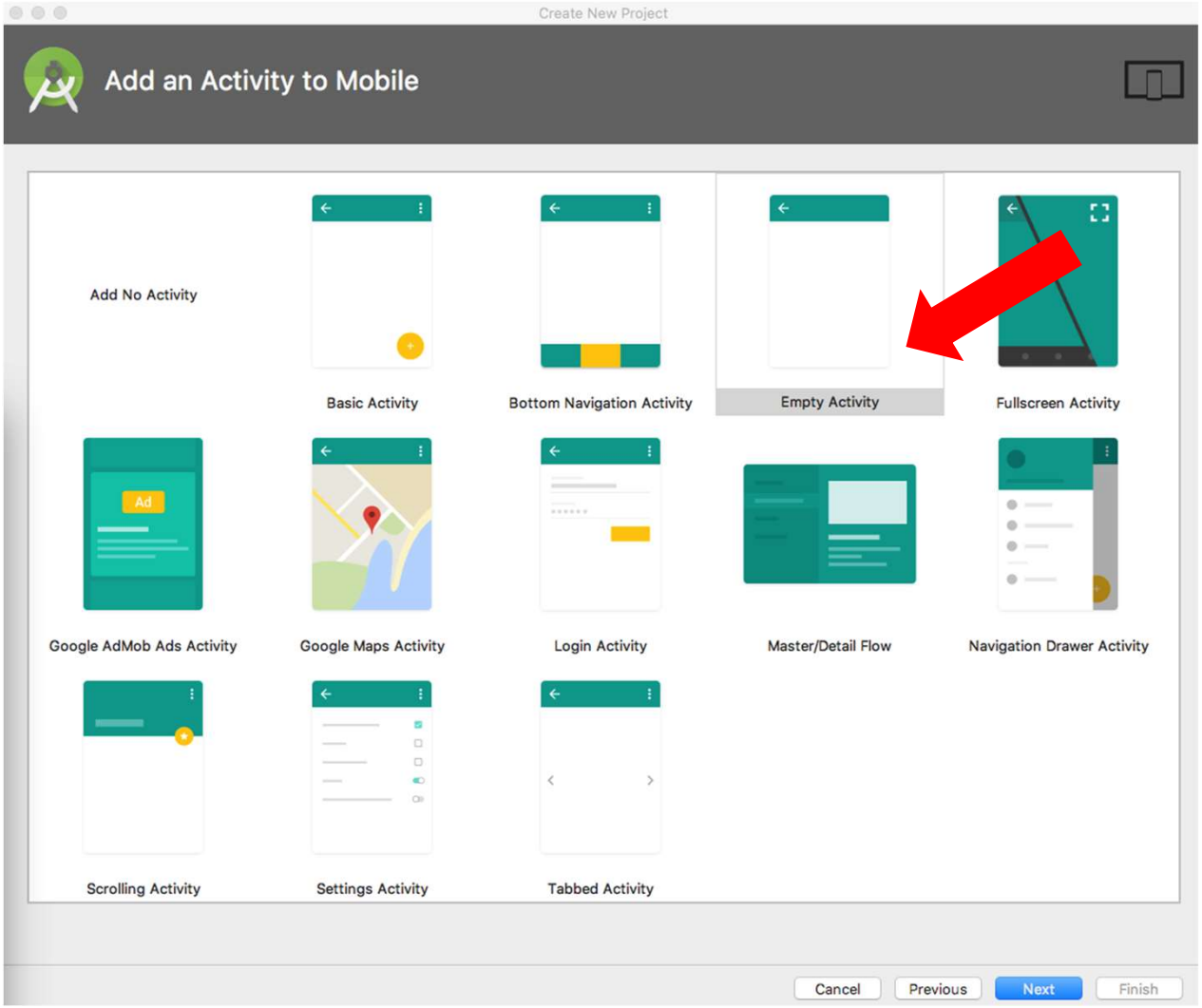
Creating an Android project Android Studio



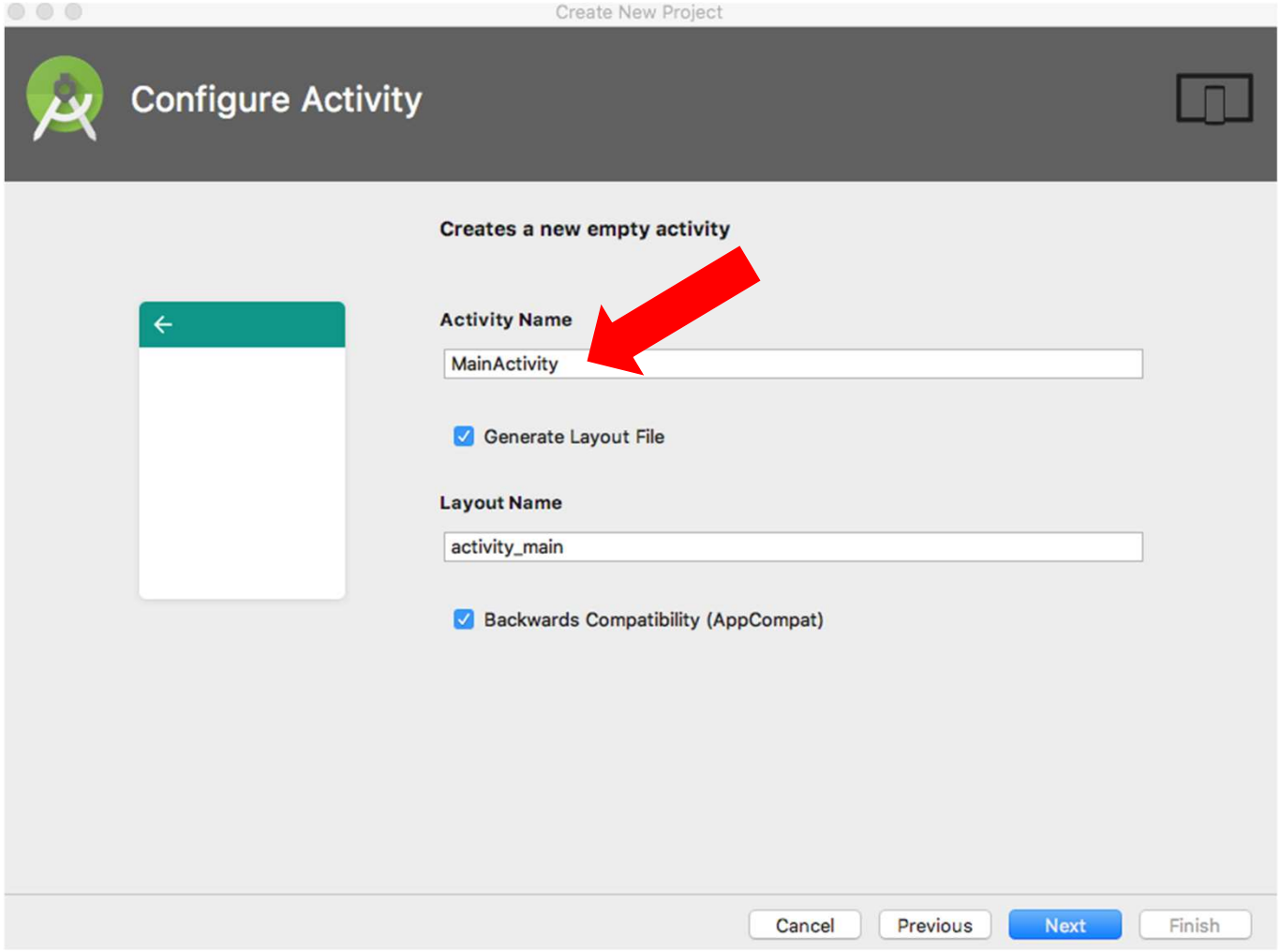
Creating an Android project



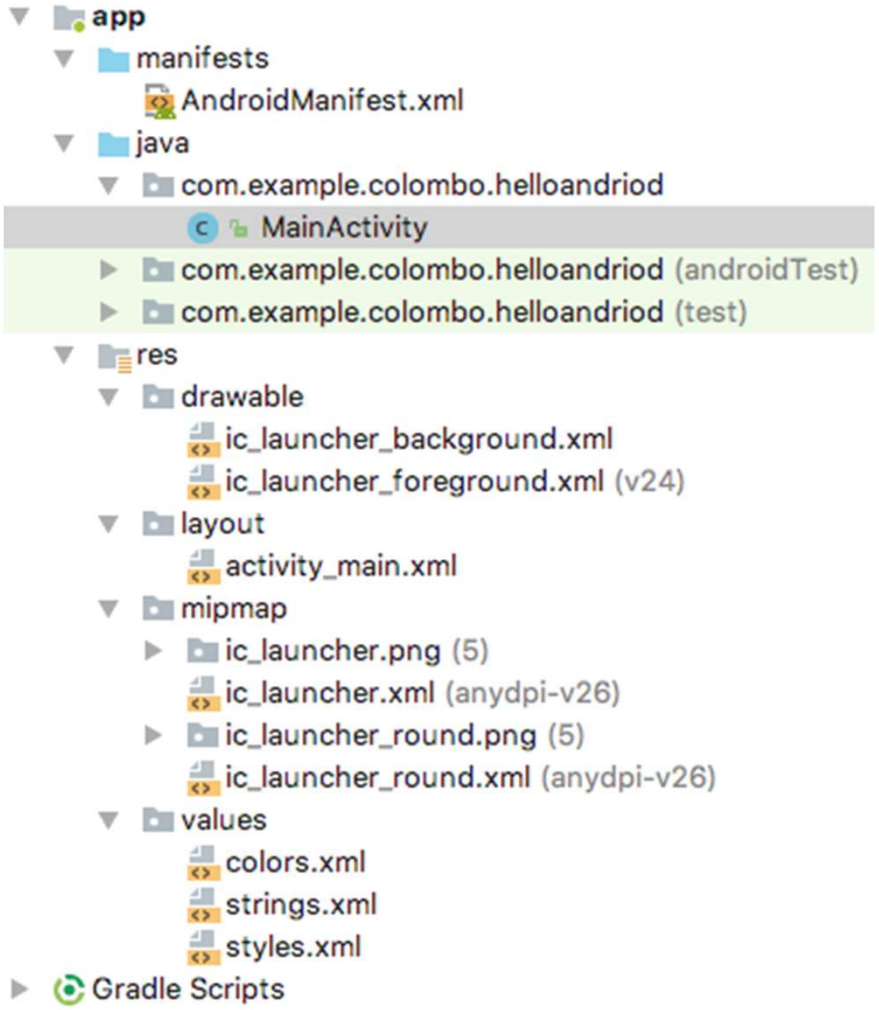
Creating an Android project Android Studio



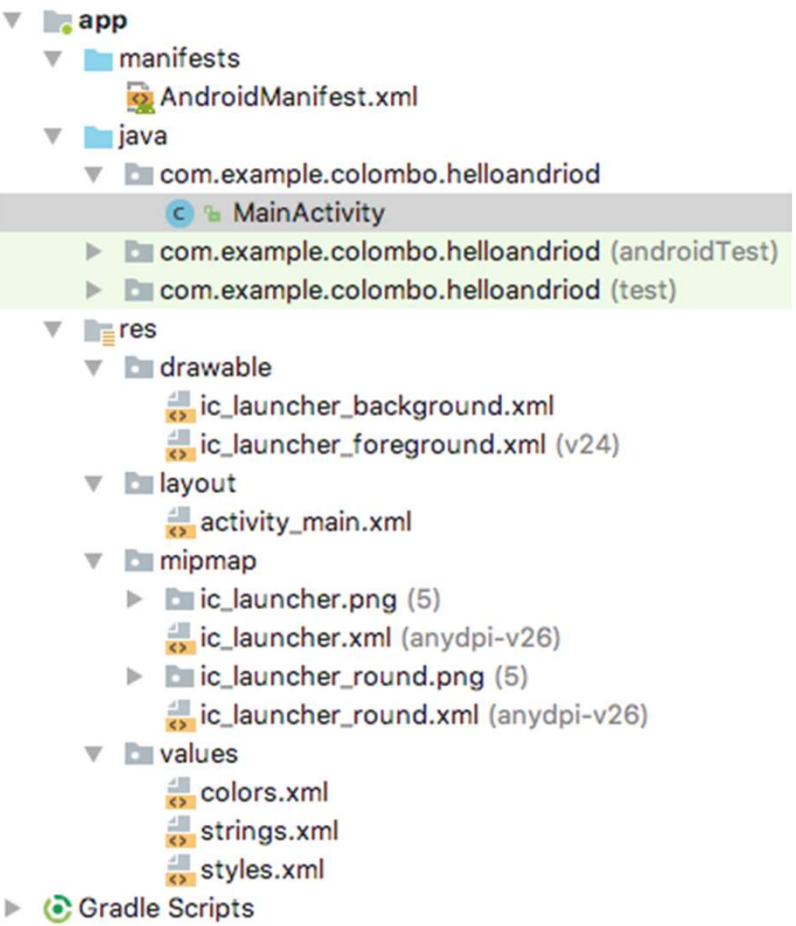
Creating an Android project Android Studio



Android project files



Android project files



- **AndroidManifest.xml**—The central configuration file for the application.
- **/java folder**—Required folder for all source code.
- **/java/com…./MainActivity.java**—Main entry point to this application. This activity has been defined as the default launch activity in the Android manifest file.
- **/res folder**—Required folder where all application resources are managed. Application resources include animations, drawable graphics, layout files, data-like strings and numbers, and raw files.
- **/res/drawable-***—Application icon graphic resources are included in several sizes for different device screen resolutions.
- **/res/layout**—Layout resource filed used by MainActivity to organize controls on the main application screen.
- **/res/values/strings.xml**—The resource file where string resources are define
- **/res/values/colors.xml**
- **/res/values/dimens.xml**
- **/res/values/styles.xml**

Activity – Class MainActivity

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- Class *MainActivity* (default name) is located in the src directory.
- Extends *AppCompatActivity* class (in *android.support.v7.app package*)
- *AppCompatActivity*: subclass of Activity.
- An Activity provides a screen with which users can interact.
- *MainActivity:* Used as the Controller for the app.

Activity – Class MainActivity

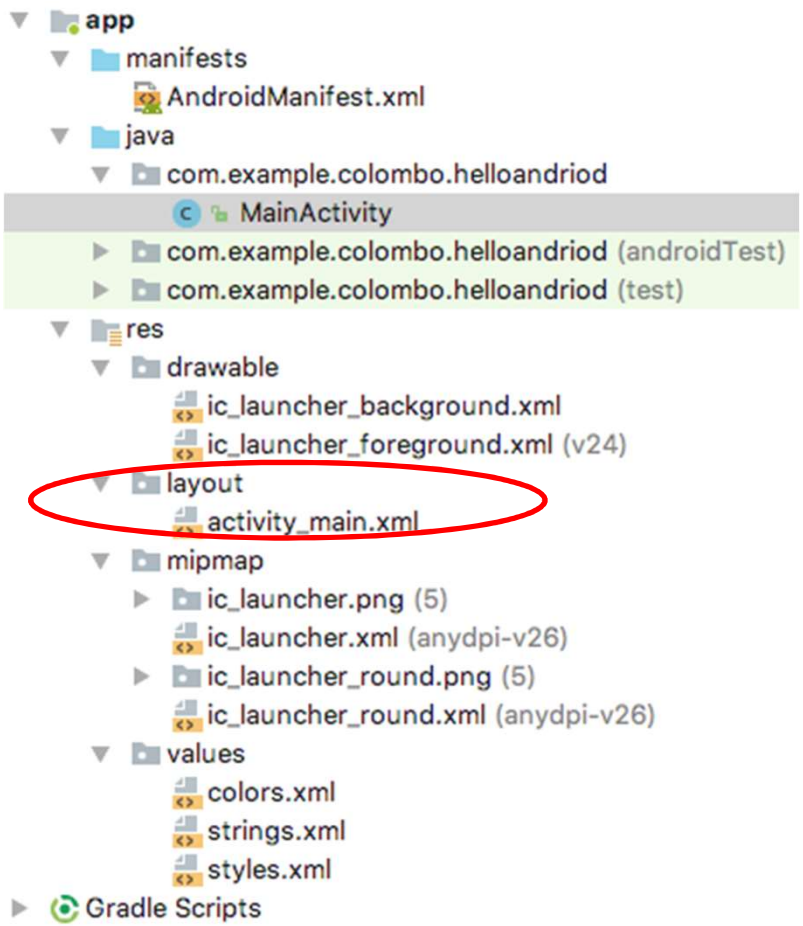
```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- *onCreate* method: called automatically.
- Inside the *onCreate* method:
  - create the initial View for the app, controlled by this Activity.
  - Inside *onCreate*, after calling the super method:
    - Set the View for this Activity by calling *setContentView* (inherited from the Activity class):

    void setContentView(int layoutResID)

    layoutResID is a Resource.
- The View for this Activity is set to be the one defined in the ***activity_main.xml***

activity_main.xml



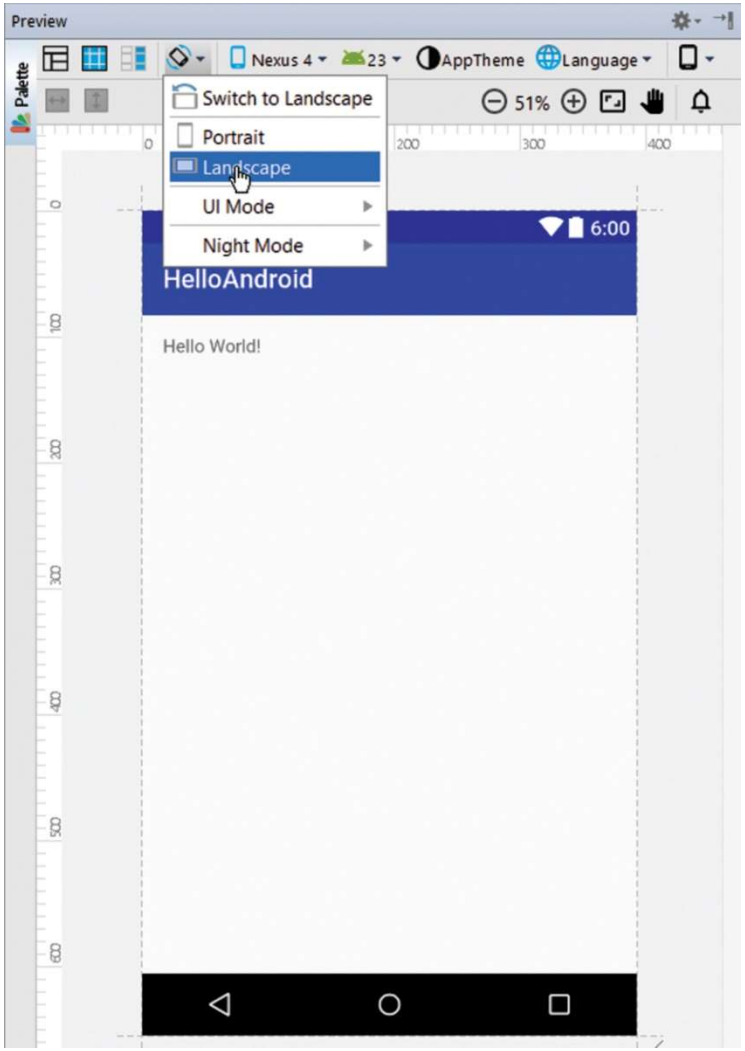
- xml file, a resource, located in the layout directory of the res directory.
- automatically created when we created the project.
- defines a simple layout for this app

activity_main

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

- Access *activity_main.xml* using: ***R.layout.activity_main***
- ***activity_main*** is a static constant of the static final class layout defined inside the R class (in the R.java file): automatically created when we created the project.
- ***R*** stands for resources.

Previewing the App



- Preview a GUI defined in a layout XML file.
- View/Tool Windows/Preview
- Customise the preview:
  - Orientation
- Choose device/theme

XML (1 of 2)

- XML: eXtensible Markup Language.
- Reference: http://www.w3.org/XML/
- As a markup language similar to HTML, but with user-defined tags.
- A non-empty element syntax:

```
<tagName attribute1 = "value1" attribute2
= "value2" … >Element Content</tagName>
```

- Example:

```
<app language = "Java" version =
"7.0">Hello Android</app>
```

XML (2 of 2)

- Element has no content (empty):

```
<tagName attribute1 = "value1" attribute2
= "value2" … />
```

- Ex:

```
<dev ide = "Studio" version = "1"/>
```

- Elements can be nested:

```
<a>
        <b>hello</b>
</a>
```

- Rules for naming tags: similar to naming identifiers in java.

activity_main.xml (1 of 8)

```
<?xml version="1.0" encoding="utf-8"?>
<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="100dp"
    android:paddingLeft="50dp"
    android:paddingRight="50dp"
    android:paddingTop="100dp"
    tools:context="com.jblearning.helloandroid.MainActivity">

  <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"/>
</RelativeLayout>
```

activity_main.xml (2 of 8)

- Two elements in activity_main.xml:
  - *RelativeLayout* and a *TextView*.
- *TextView*: nested inside the RelativeLayout element.
- *RelativeLayout*
  - Organizes elements in relation to each other and in relation to their parent View.

activity_main.xml (3 of 8)

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   …>
   <TextView
      …/>
</RelativeLayout>
```

activity_main.xml (4 of 8)

- *RelativeLayout* attributes:
  - *android:layout_width* and *android:layout_height* attributes define the size of the *RelativeLayout*.
- Value *match_parent*
  - As big as its parent element (e.g. screen).

activity_main.xml (5 of 8)

```
<RelativeLayout
    …
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom=
        "@dimen/activity_vertical_margin"
    …
>
```

activity_main.xml (6 of 8)

- *TextView* element has three attributes.
- A TextView element is an instance of the TextView class, which encapsulates the concept of a label.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"/>
```

activity_main.xml (7 of 8)

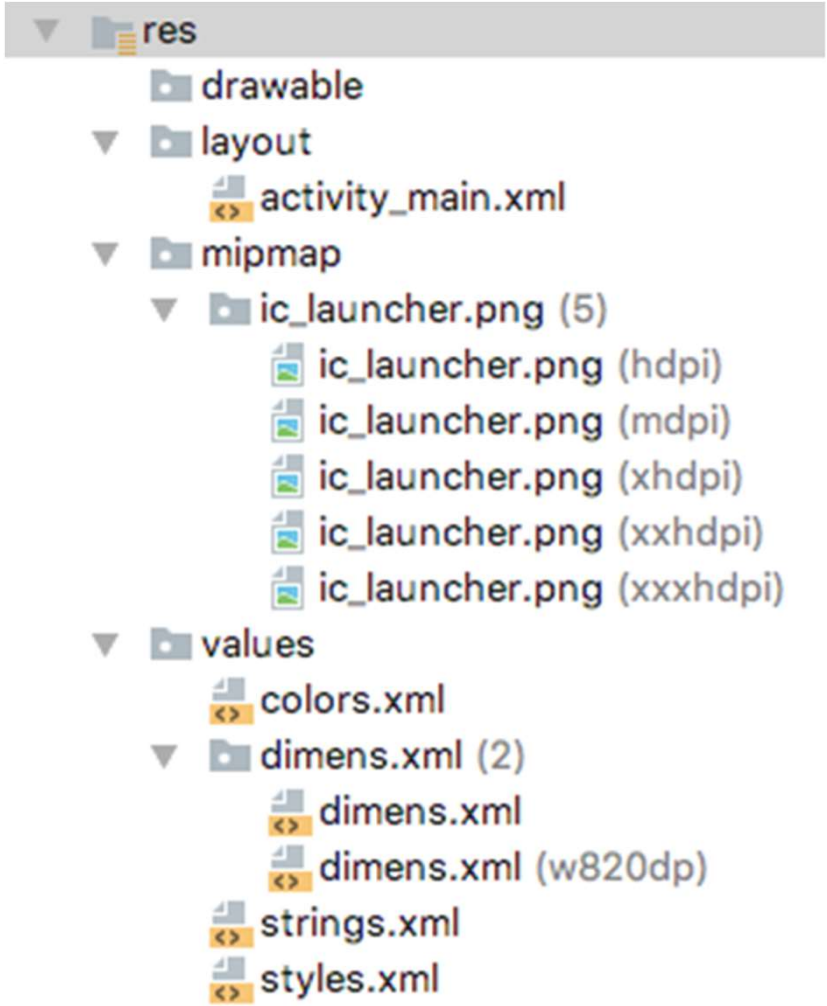
- Three attributes: *android:layout_width, android:layout_height and android:text*.
- android:layout_width and android:layout_height attributes define the size of the TextView.
- Value wrap_content: as small as possible so that their contents fit inside it.
- The element “wraps” around its content.
- By default, it appears at the top left corner of the screen.

activity_main.xml (8 of 8)

- The *android:text* attribute specifies the content of the *TextView* element.
- Its value is “Hello World!”.

```
<?xml version="1.0" encoding="utf-8"?>
<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="100dp"
    android:paddingLeft="50dp"
    android:paddingRight="50dp"
    android:paddingTop="100dp"
    tools:context="com.jblearning.helloandroid.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/>
</RelativeLayout>
```

Android Project Resources



dimens.xml (1 of 5)

- The dimens.xml file is another (automatically generated) XML file; its contents comply with XML syntax.
- It is used to define dimensions.

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines.
    <dimen name="activity_horizontal_margin">50dp</dimen>
    <dimen name="activity_vertical_margin">100dp</dimen>
</resources>
```

dimens.xml (2 of 5)

```
<resources>
   <!-- Default screen margins, per the Android Design
        guidelines. -->
  <dimen
name="activity_horizontal_margin">16dp</dimen>
  <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

- activity_horizontal_margin and activity_vertical_margin are used in activity_main.xml.

dimens.xml (3 of 5)

- The syntax for defining a dimension is

  <dimen name = "dimensionName">
  valueOfDimension</dimen>

dimens.xml (4 of 5)

- We can change the margin values as follows:

```
<resources>
<dimen name="activity_horizontal_margin">50dp</dimen>
  <dimen name="activity_vertical_margin">100dp</dimen>
</resources>
```

dimens.xml (5 of 5)

- The preview shows the new margins:

strings.xml (1 of 4)

- The strings.xml is another (automatically generated) XML file; its contents comply with XML syntax.
- One string is defined in the strings.xml file: app_name.

```
<resources>
    <string name="app_name">My First App</string>
</resources>
```

strings.xml (2 of 4)

```
<resources>
   <string name="app_name">HelloAndroid</string>
</resources>
```

strings.xml (3 of 4)

- The syntax for defining a string is

<string name =
"stringName">valueOfString</string>

- The value of the string app_name is

HelloAndroid

strings.xml (4 of 4)

- We can edit strings.xml, for example modify the value of hello_world as follows:

```
<string name="app_name">My First App</string>
```

- When we preview or run the app again, the title of the app is My First App.

activity_main.xml (1 of 2)

- We can also edit activity_main.xml, for example center the TextView as follows:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hi there"/>
```

activity_main.xml (2 of 2)

- When we run the app again, the text inside the TextView says “Hi there” instead of “Hello World!” as before.

styles.xml (1 of 5)

- The styles.xml file is another (automatically generated) XML file; its contents comply with XML syntax.
- It is used to define styles that the app uses.

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="android:textSize">40sp</item>
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

styles.xml (2 of 5)

```
<resources>
 <!-- Base application theme. -->
  <style name="AppTheme"
            parent="Theme.AppCompat.Light.DarkActionBar">
   <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
   </style>
</resources>
```

styles.xml (3 of 5)

- We can modify a style by adding an *item* element using this syntax:

```
<item
  name="styleAttribute">valueOfItem</item>
```

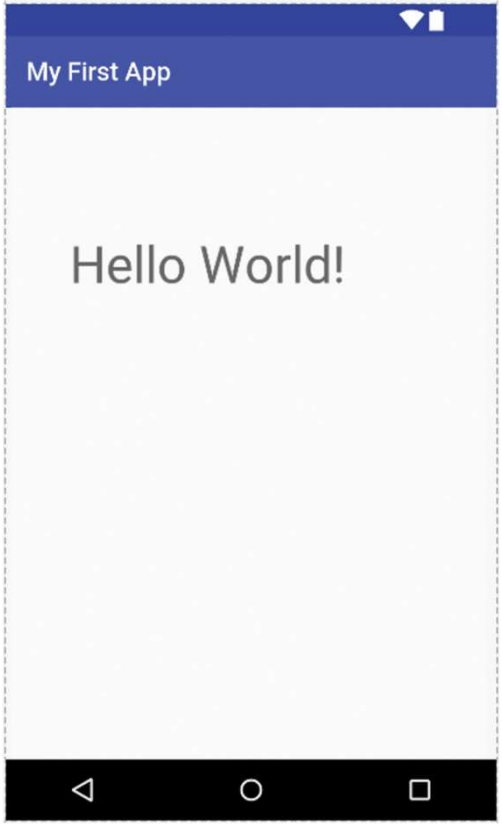
styles.xml (4 of 5)

- The name of the style attribute that specifies the text size inside a TextView is android:textSize. We change the default text size to 40.

```
<item name="android:textSize">40sp</item>
```

styles.xml (5 of 5)

- The preview shows the effect of the new text size:



colors.xml (1 of 3)

- The colors.xml file is another (automatically generated) XML file; its contents comply with XML syntax.
- It is used to define colors.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

colors.xml (2 of 3)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <color name="colorPrimary">#3F51B5</color>
 <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
</resources>
```

colors.xml (3 of 3)

- The syntax for defining a color is

  <color name = "colorName">
  valueOfColor</color>

- The color value is defined using hexadecimal notation (more on that later in the book).

Hexadecimal Color Value (1 of 2)

- #rrggbb (uses RGB color system).
- rr = amount of red in color
- gg = amount of green in color
- bb = amount of blue in color.

Hexadecimal Color Value (2 of 2)

- #rrggbb (uses RGB color system).
- Values vary from 00 (0) to FF (255)
- That means we can define 256 × 256 × 256 = 16.7 million colors.

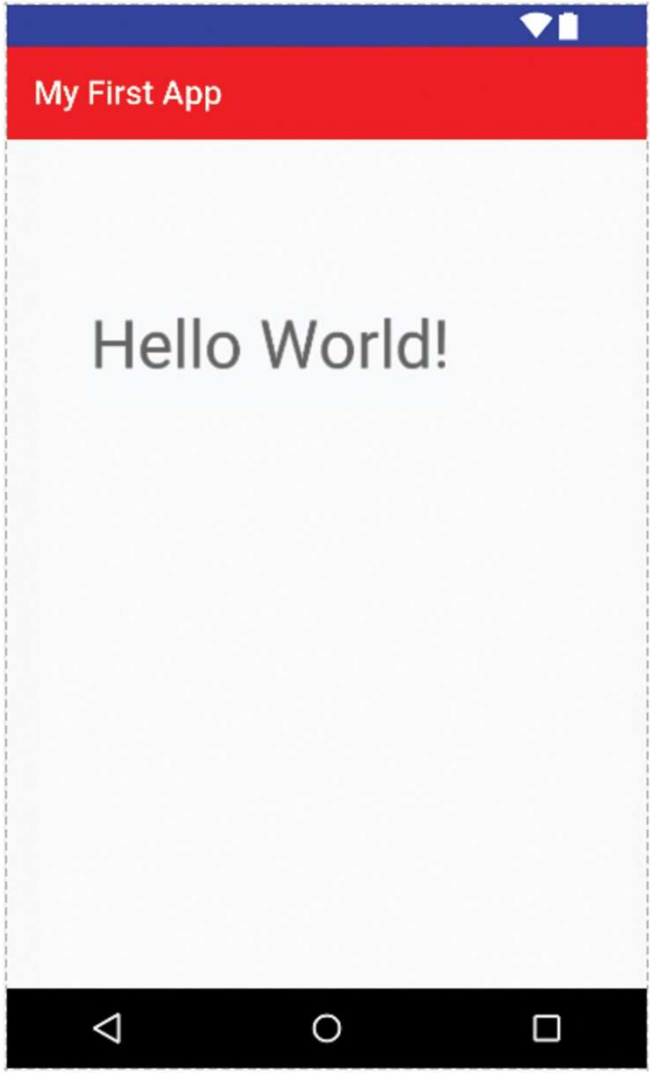
colors.xml (1 of 2)

- If we modify the value of colorPrimary to red (#FF0000):

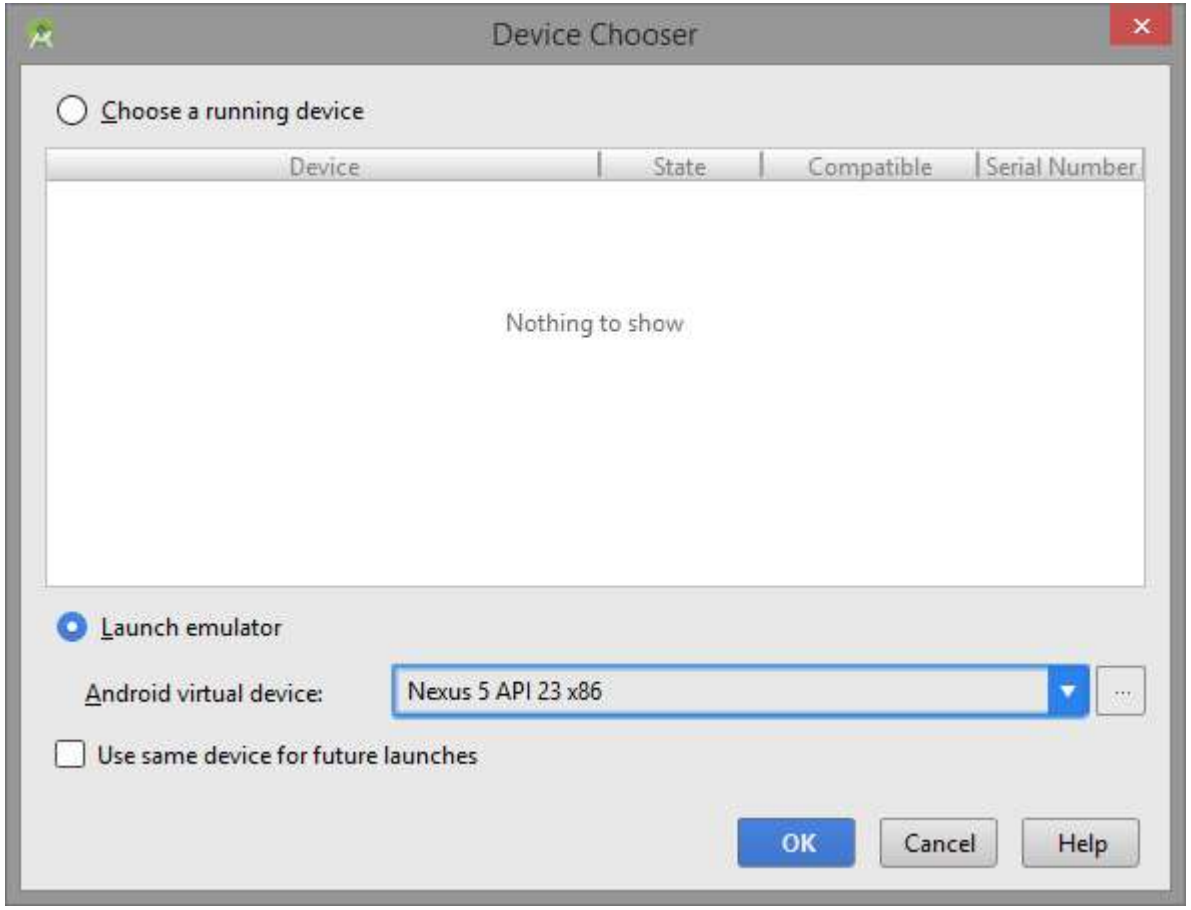
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <color name="colorPrimary">#FF0000</color>
 <color name="colorPrimaryDark">#303F9F</color>
  <color name="colorAccent">#FF4081</color>
</resources>
```

colors.xml (2 of 2)

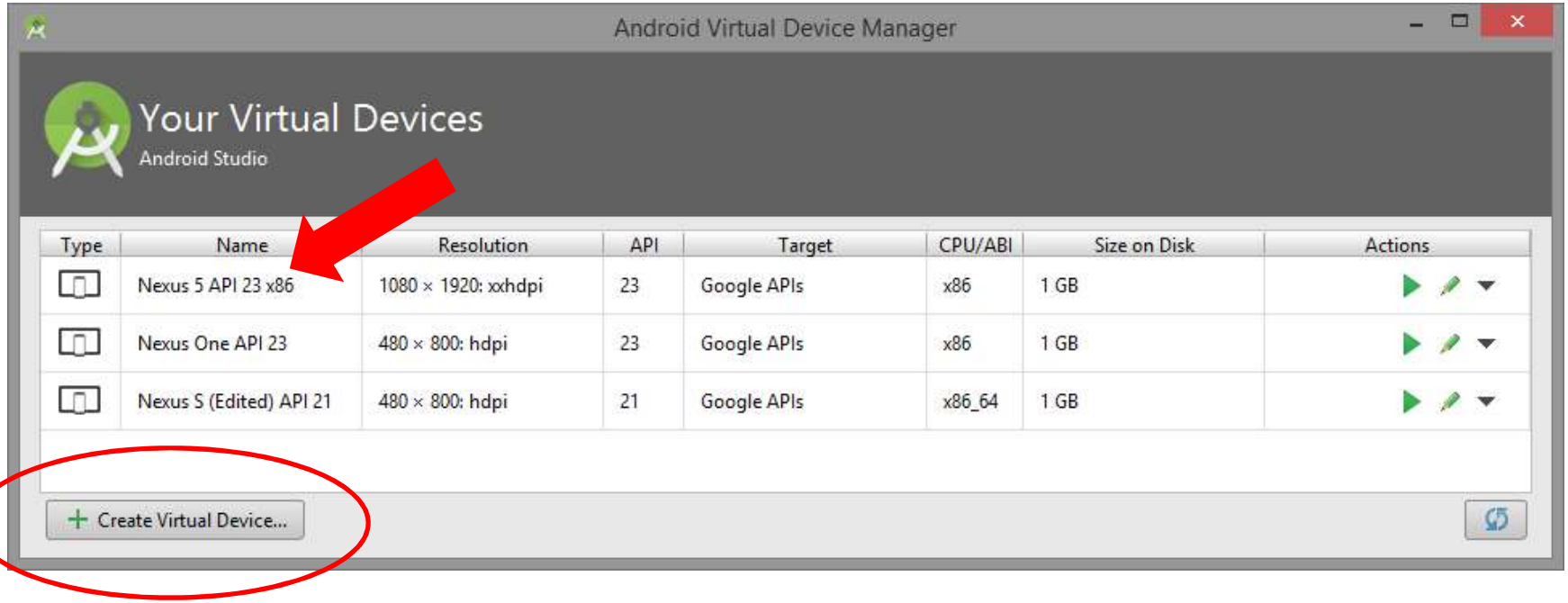
- Previewing the app after modifying the value of color Primary to red (#FF0000):



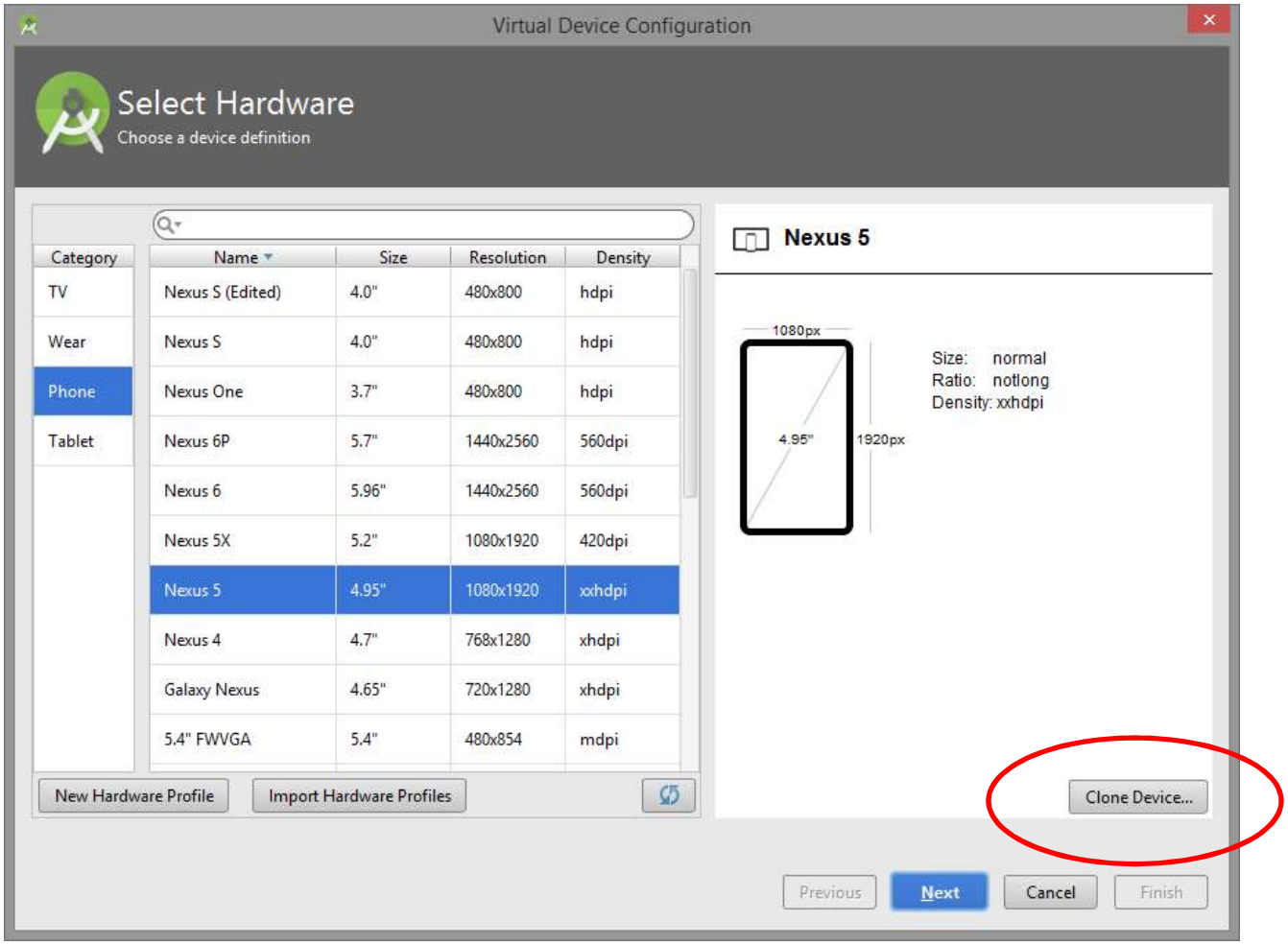
Running Android Project



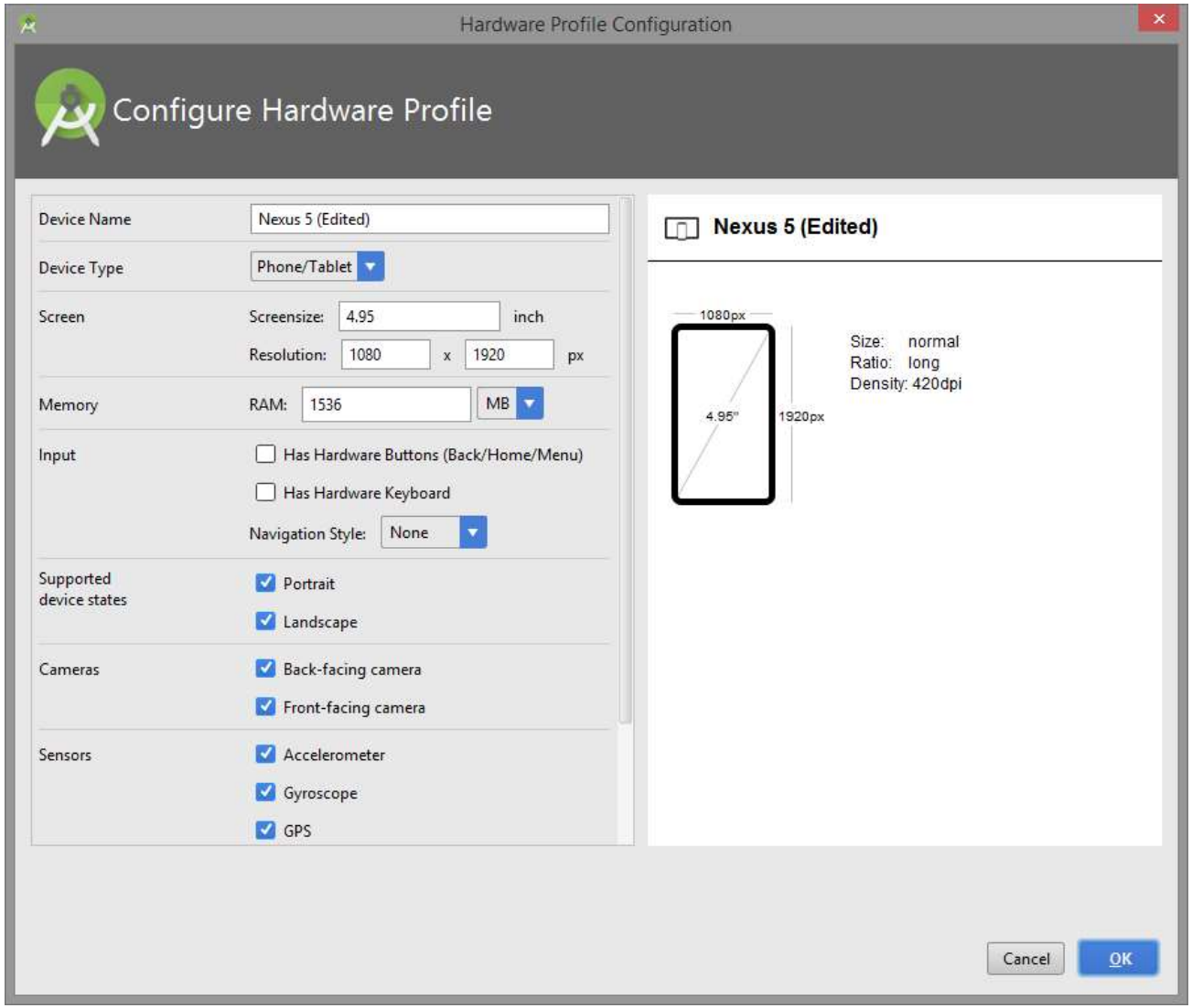
Creating Android Virtual Devices (AVD)



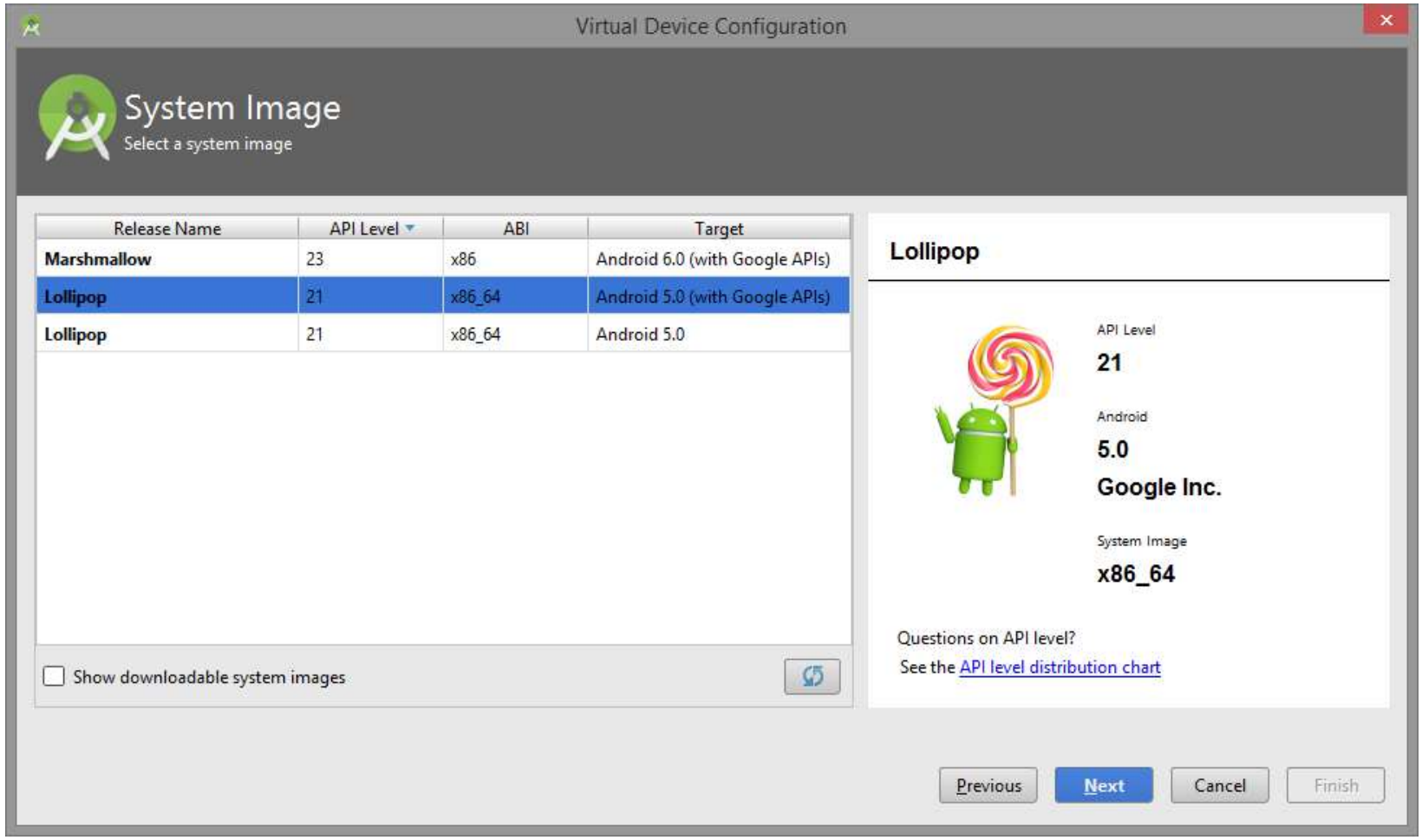
Creating Android Virtual Devices (AVD)



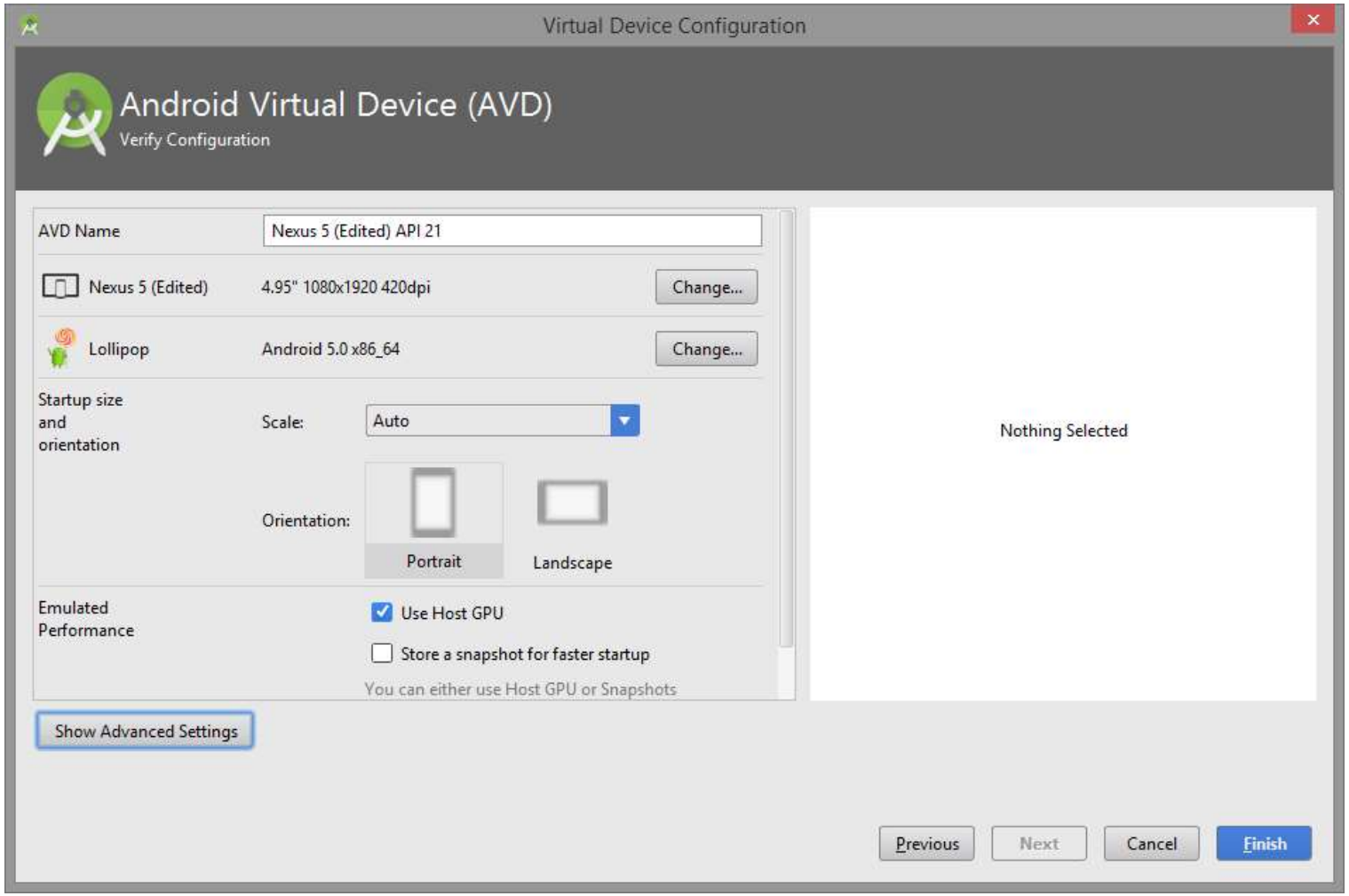
Creating Android Virtual Devices (AVD)



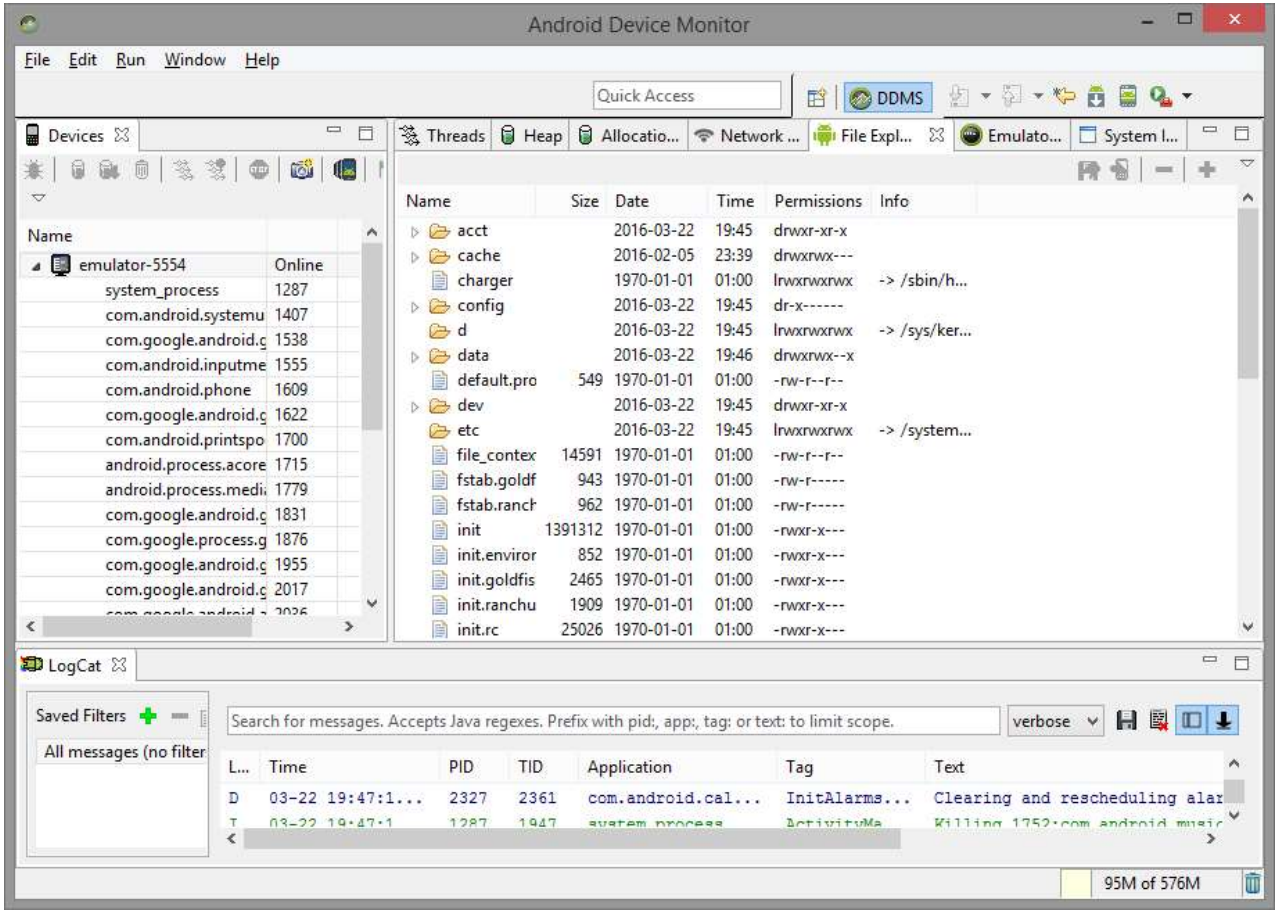
Creating Android Virtual Devices (AVD)



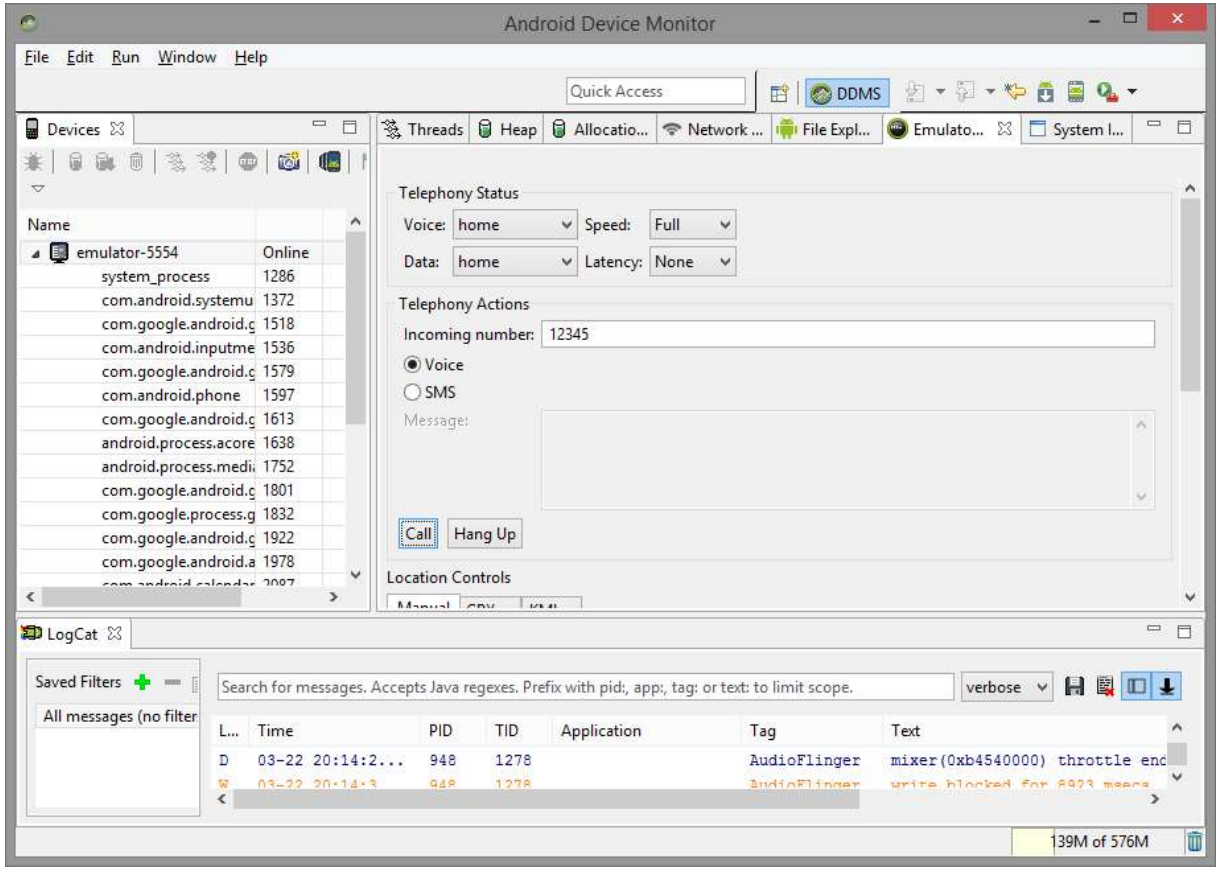
Creating Android Virtual Devices (AVD)

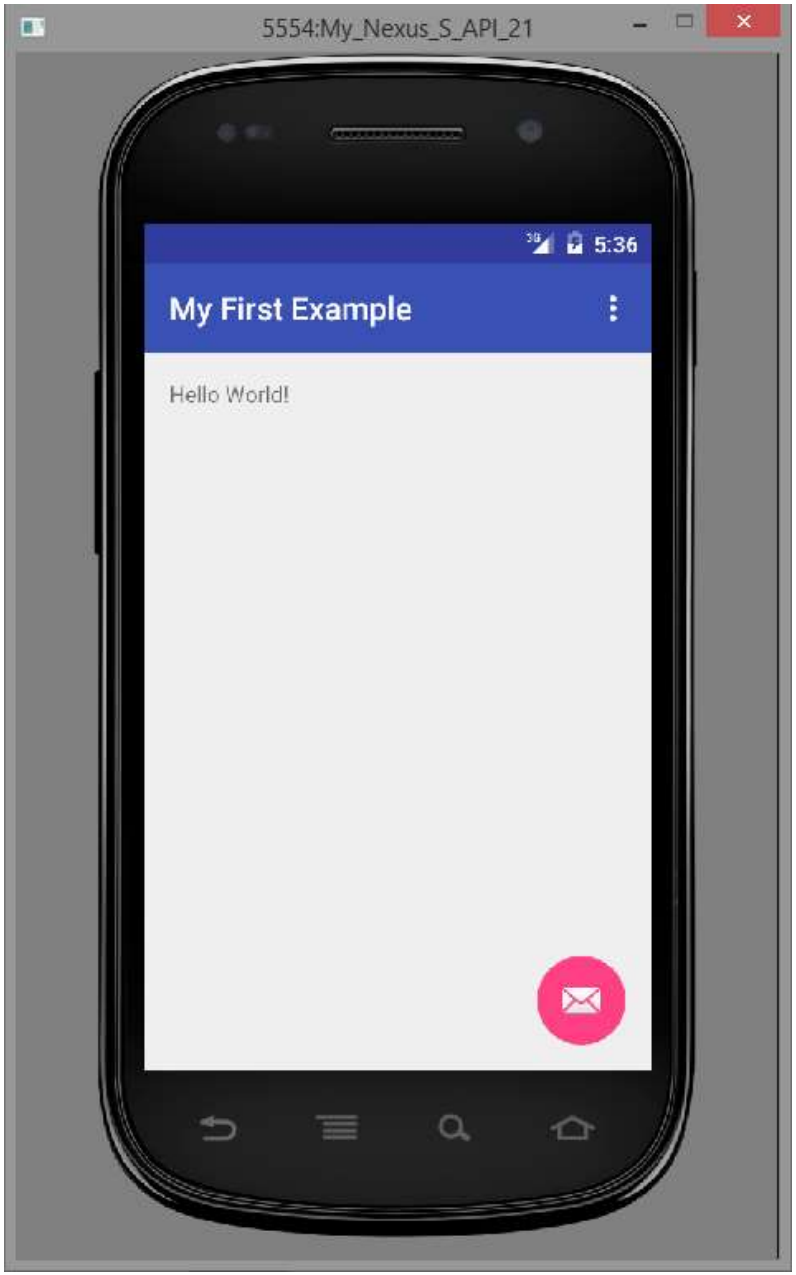


Android Device Monitor



Android Device Monitor



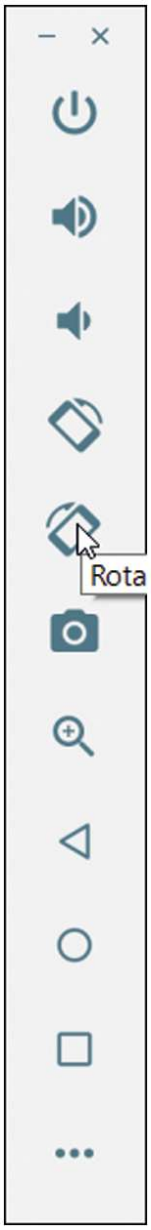

5554:My_Nexus_S_API_21
5:36
My First Example
Hello World!


5554:My_Nexus_S_API_21
5:40
APPS
WIDGETS
API Demos
Browser
Calculator
Calendar
Camera
Clock
Contacts
Custom Locale
Dev Settings
Dev Tools
Downloads
Email
Gallery
Gestures Builder
Google Settings
Maps
Messaging
Music
My First Example
My UI

Running Inside the Emulator

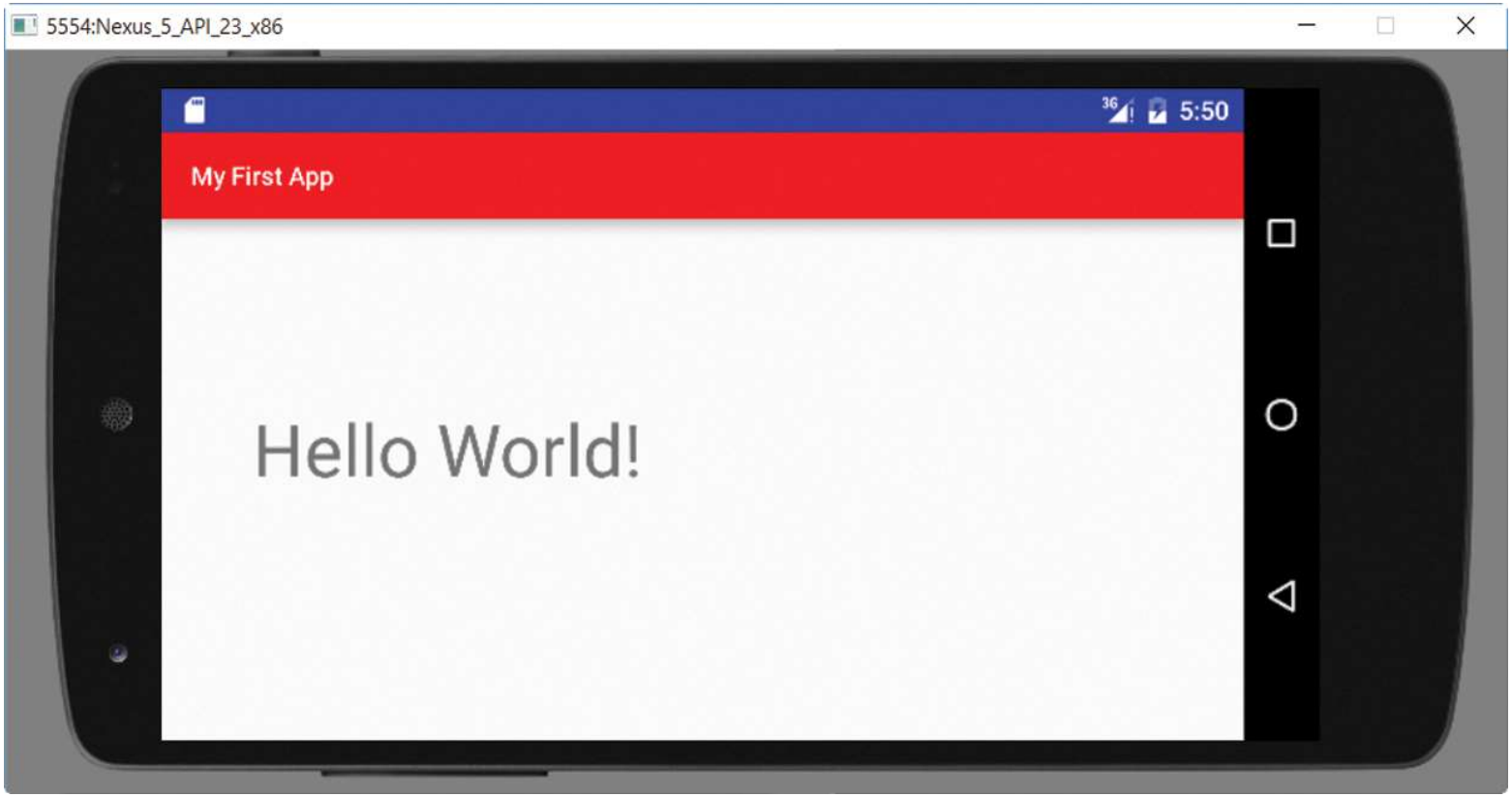
Running

- Rotate the app as it is running inside the emulator:
  - Click on the rotate button on the tool bar.



Running Inside the Emulator

- After rotating the app.



Instant Run (1 of 2)

- An interesting recent feature of Android Studio is Instant Run.
- With Instant Run, we can modify some selected components of an app, for example the strings.xml file.
- Click on the Run icon, and the emulator automatically updates the app on the fly. Instant Run is enabled by default.

Instant Run (2 of 2)

- If we want to disable Instant Run, we can choose File, Settings, and within the Build, Execution, Deployment section, select Instant Run, and then edit it.
- If we do not want to use Instant Run, we can uncheck it and check Restart activity on code changes.

Feedback and Debugging

- We can send output to the console in addition to displaying data on the screen.
- Using various static methods of the Log class, located in the android.util package.

Log Class (1 of 2)

- Selected methods include d, e, i, v, w. They all have the same parameter list and return type; for example, the API of d is

  public static void d(String tag,
  String message)

- d stands for debug, e for error, i for info, v for verbose, and w for warning.

Log Class (2 of 2)

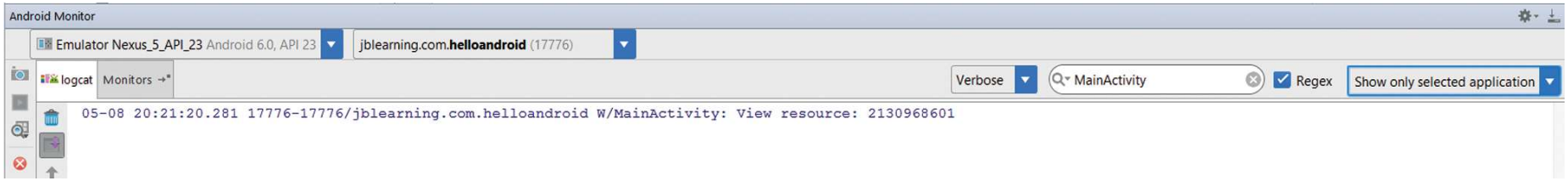
public static void d(String tag,
String message)

- tag identifies the source of the message and can be associated with a “filter”.
- message is the String to be output.

Logcat

- Output from logging statements will show in the lower panel, below the source code, if we select the Logcat tab.
- If the Logcat tab does not show, we should run the app in debug mode at least once (click on the bug icon, next to the run icon), then it will show when we run it.

Logcat Output

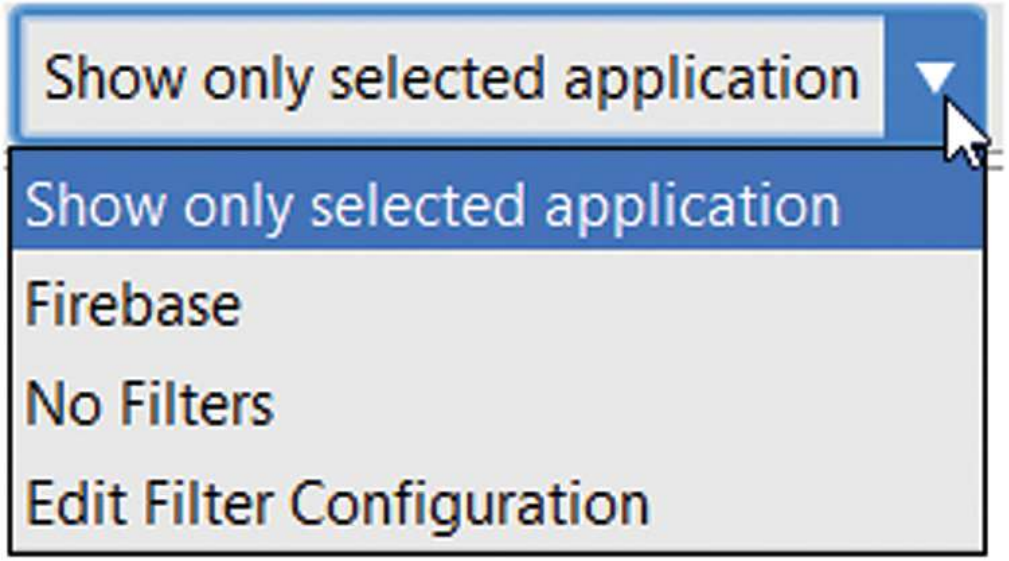


Logcat

- There can be many output messages, so we can filter them. When the filter is selected, only the output whose tag is associated with that filter is shown.

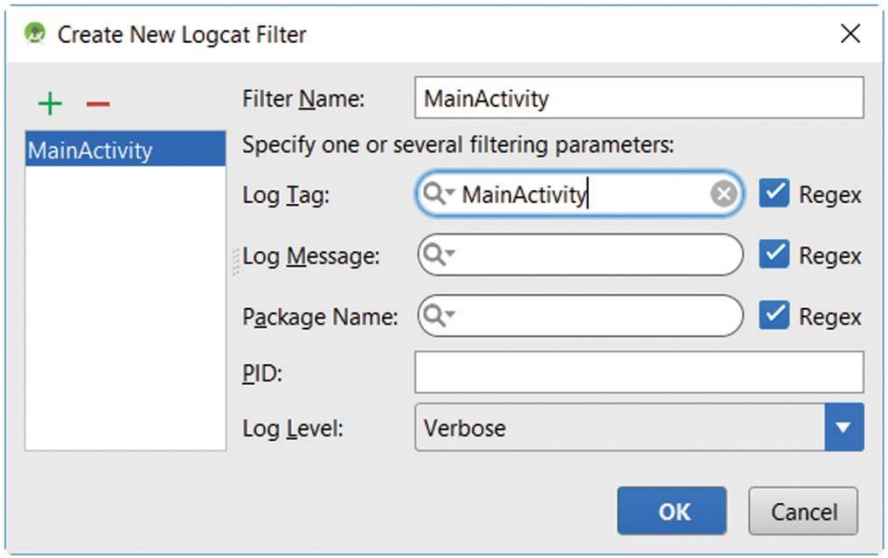
Setting Up a Filter (1 of 2)

- To create a filter, click on the combo box on the right of the lower panel, select Edit Filter Configuration.



Setting Up a Filter (2 of 2)

- Inside the dialog box, enter a name for the filter and a Log tag for it.
- For example, the filter name can be MainActivity_jbl, and the tag name can be MainActivity.



Using the Log Class (1 of 3)

- Now that a filter has been created along with its tag, we can output messages to Logcat like this:

Log.w(“MainActivity”, “Inside onCreate”);

- The tag name is MainActivity.
- The output is Inside onCreate.

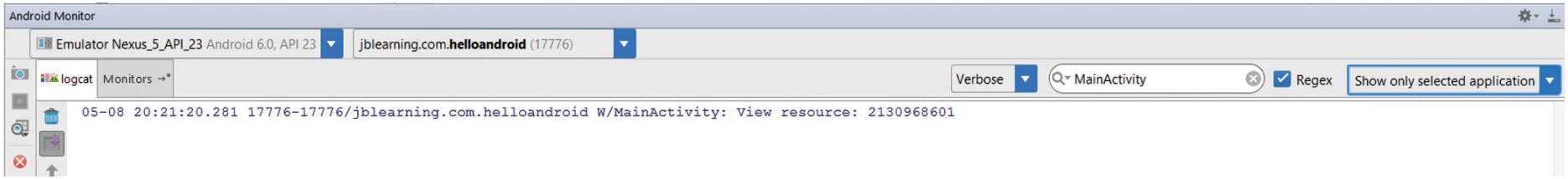
Using the Log Class (2 of 3)

- We can use logging statements at various places in MainActivity.java.
- We execute inside the onCreate method of the Activity class as well the value of the activity_main resource id.

Using the Log Class (3 of 3)

```
…
public static String MA = "MainActivity";
…
setContentView( R.layout.activity_main );
Log.w( MA, "View resource: " + R.layout.activity_main );
…
```

Logcat Output



Using the Debugger (1 of 6)

- Android Studio includes a debugger.
- We can set up breakpoints, check the values of variables or expressions, step over code line by line, check object memory allocation, …

Using the Debugger (2 of 6)

- To set up a breakpoint, we click on the left of a statement: an orange filled circle appears.

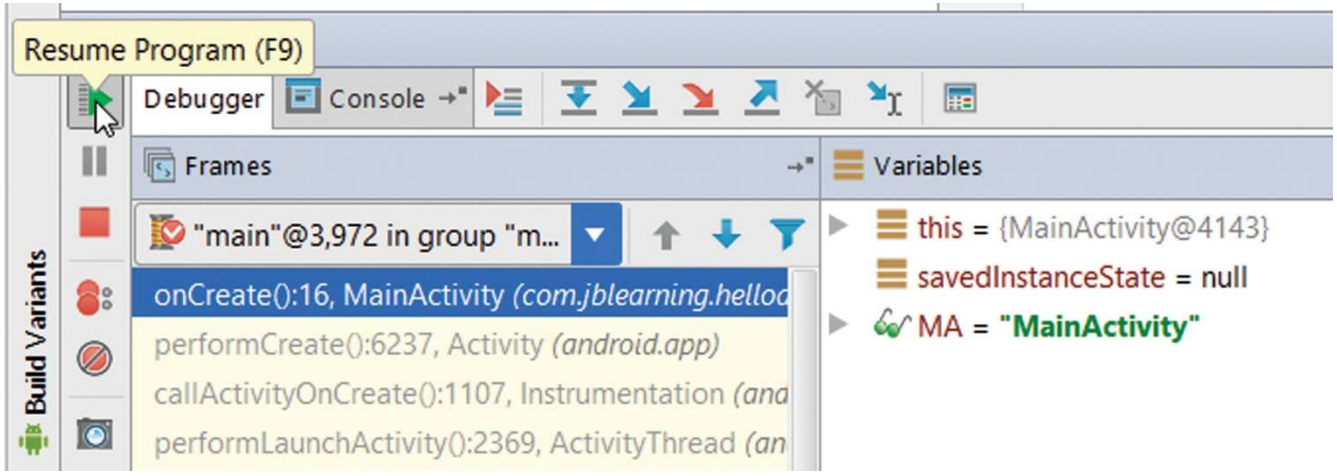
```
@Override
protected void onCreate( Bundle savedInstanceState ) {
  super.onCreate( savedInstanceState );
  setContentView( R.layout.activity_main );
  Log.w( MA, "View resource: " + R.layout.activity_main );

  int count = 0;
  for( int i = 0; i < 3; i++ )
    count++;
}
}
```

Using the Debugger (3 of 6)

- To run the app in debug mode, we click on the debug icon on the toolbar.
- The app runs and stops at the first breakpoint.
- The debugger tab is selected in the panel at the bottom of the screen and we can see some debugging information and tools.

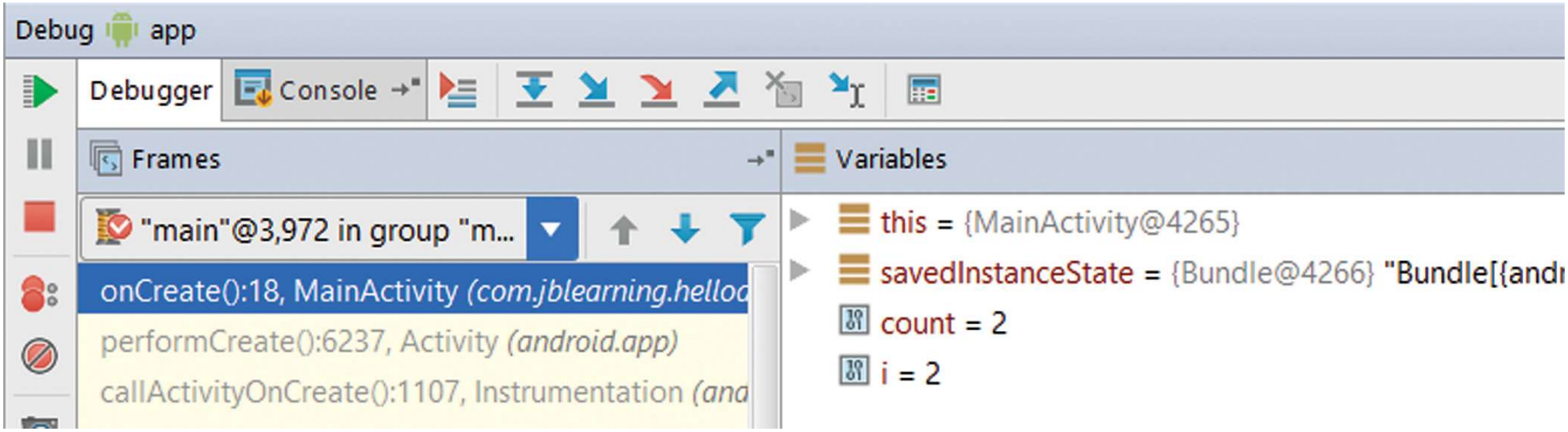
Using the Debugger (4 of 6)

- Under Frames, we can see where in the code we are currently executing.
- To resume the app, we click on the green Resume icon at the top left of the panel.



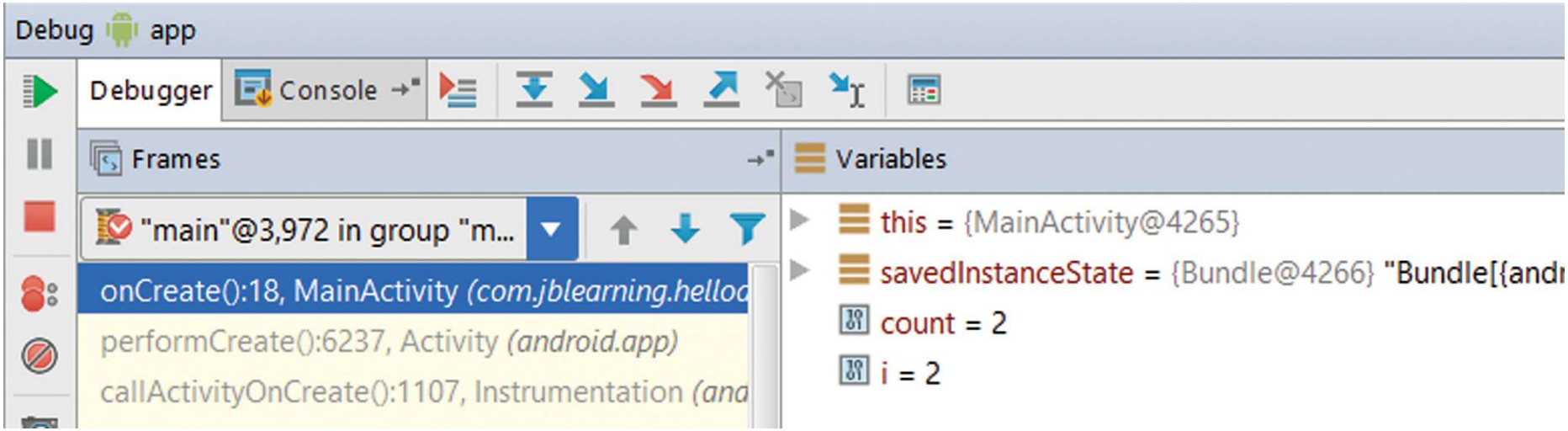
Using the Debugger (5 of 6)

- As we resume, stop at breakpoints, and resume the app a few times, the values of the various variables in our app are displayed under Variables.



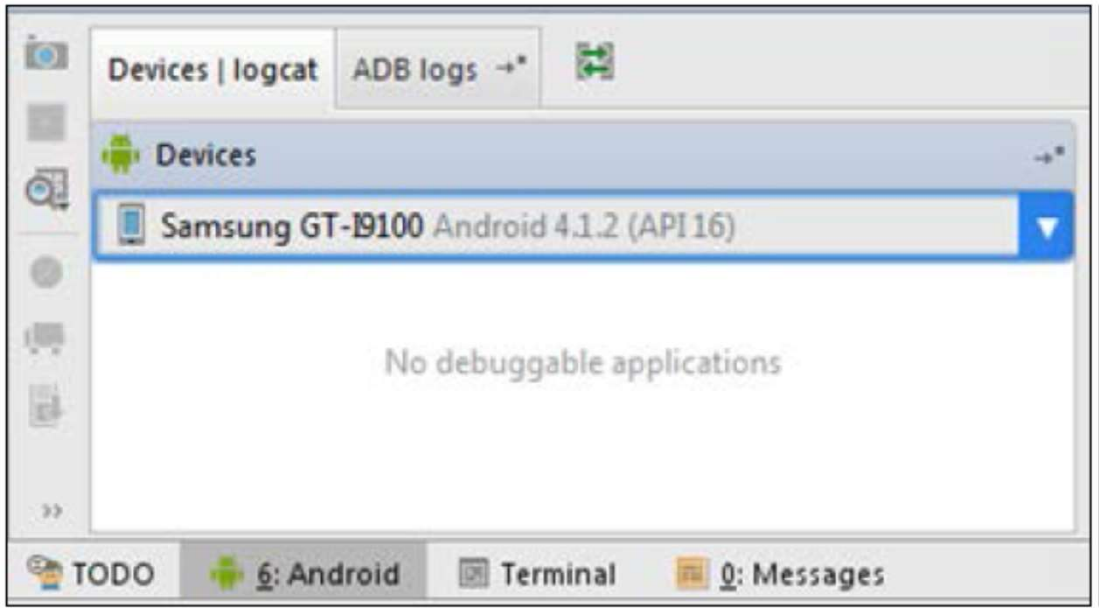
Using the Debugger (6 of 6)

- Under Variables, we can check the values of the various variables, for example MA, which has value MainActivity.



Deploy to a real Android Device Monitor

- Visit device manufacturer's website to download and install any drivers required
- http://developer.android.com/tools/device.html

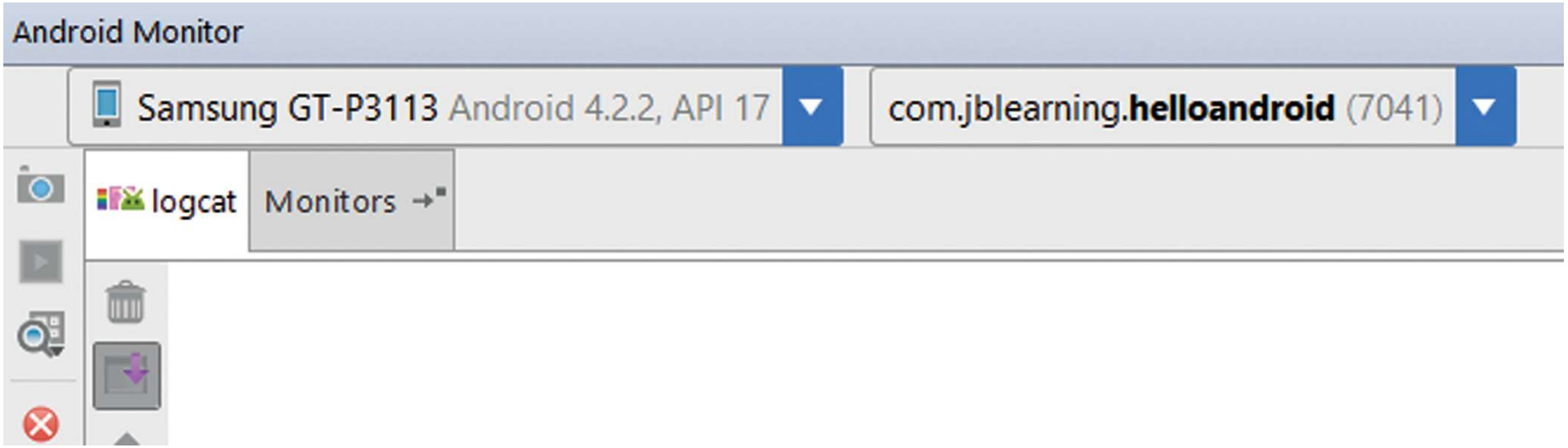


Running on an Actual Device (1 of 5)

- To run on an actual device, we need to do two things:
  - Download a driver for the device.
  - Connect the Android device to the computer.

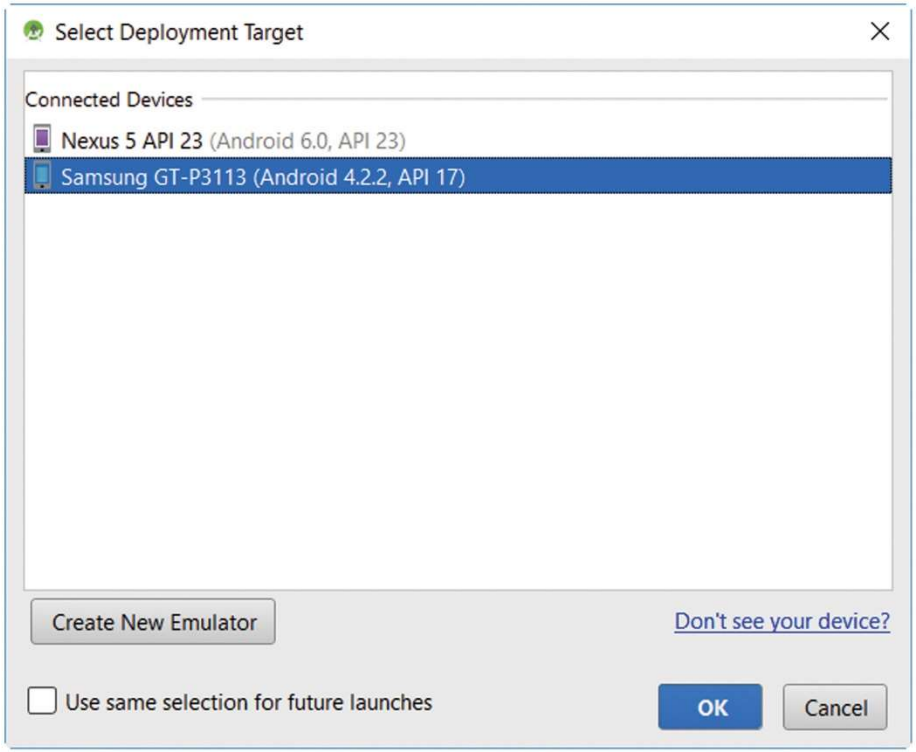
Running on an Actual Device (2 of 5)

- Once the device is connected, its name should appear on the left upper corner of the lower pane.



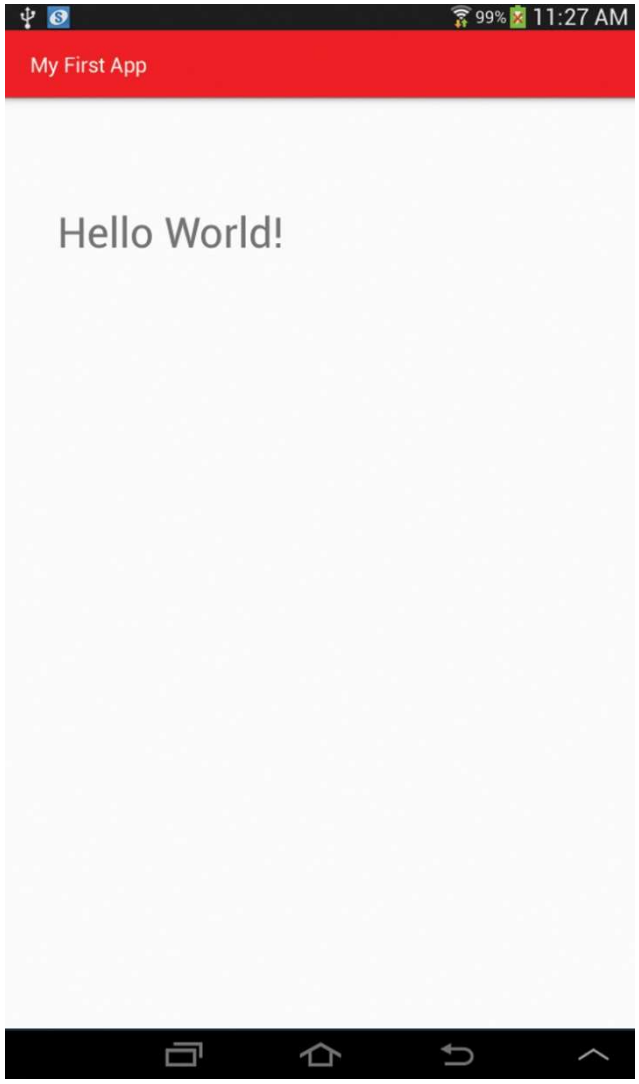
Running on an Actual Device (3 of 5)

- When we run, it should appear under "Choose a running device".
- Select "Choose a running device" and click on OK.



Running on an Actual Device (4 .. 5)

- The app running inside a tablet



.Running on an Actual Device (5 .. 5)

- If the app is running on a device, we can still log output statements in Logcat.
- This is much faster than starting the emulator.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="com.jblearning.helloandroid"
          xmlns:android="http://schemas.android.com/apk/res/android">

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="My First App"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>

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

</manifest>
```

The App Manifest

- The AndroidManifest.xml file, located in the manifests directory, specifies the resources that the app uses, such as activities, the file system, the Internet, and hardware resources.
- Before a user downloads an app on Google Play, the user is notified about these.

AndroidManifest.xml (1 of 2)

- AndroidManifest.xml's automatically generated versio: Among other things, it defines the icon and the label or title for the app.
- The text inside the label is the app_name String defined in strings.xml.

AndroidManifest.xml (2 of 2)

- We should supply a launcher icon for your app.
- This is the visual representation of our app on the home screen or the apps screen.
- A launcher icon for a mobile device should be 48 × 48 dp.

Launcher Icon (1 of 2)

- To set the launch icon for the app to hi.png, we assign the String @mipmap/hi to the android:icon attribute of the application element in the AmdroidManifest.xml file.
- The @mipmap/hi expression defines the resource in the mipmap directory (of the res directory) whose name is hi (note that we do not include the extension).

```
android:icon="@mipmap/hi"
```

Launcher Icon (2 of 2)

- When we run an app inside a device, the app is automatically installed.
- If we included an icon, we can see the app among the various app icons of the device.

Orientation (1 of 2)

- Sometimes, we want the app to run in only one orientation, vertical for example, and therefore we do not want the app to rotate when the user rotates the device.
- Inside the activity element, we can add the android:screenOrientation attribute and specify either portrait or landscape as its value.

Orientation (2 of 2)

- For example, if we want our app to run in vertical orientation only, we add:

  android:screenOrientation="portrait“

- Note that we can control the behavior of the app on a per activity basis.
- In this app, there is only one activity, but there could be several.

Gradle Build System (1 of 2)

- Android Application Package (APK), is the file format for distributing applications that run on the Android operating system.
- The file extension is .apk.
- To create an apk file, the project is compiled and its various parts are packaged into the apk file.
- apk file can be found in the projectName/app/build/outputs/apk directory.

Gradle Build System (2 of 2)

- Apk files are built using the gradle build system, which is integrated in the Android Studio environment.
- When we start an app, the gradle build scripts are automatically created.
- They can be modified to build apps that require custom building.

Summary

- Android Studio
  - Creating an Android project
  - Creating a virtual device
  - Running an Android project

Summary (cont.)

- The XML layout file: activity_main.xml
- Other XML files: strings.xml, styles.xml, dimens.xml, colors.xml.
- Running inside the emulator
- Running on a device
- The Activity (MainActivity) class
- Logcat output, Debugger.