ITEC 399
MOBILE APPLICATION DEVELOPMENT

CHAPTER 4 –ANDROID PLATFORM

Subject Teacher: Mobina Beheshti

OBJECTIVES

- Description of Android OP
- Android studio software application
- Android application marketplace
- Android studio installation process
- Android studio default files
- How to run app in android studio

WHAT IS ANDROID?

- Android is an operating system based on **Linux** and is an application platform for mobile devices using **Java**.
- Android was initially developed by **Google** and is now run by the **Open Handset Alliance** (OHA), which is a consortium of around 50 hardware, software and telecom companies led by Google.
- Android is the largest installed base of any mobile platform and is growing fast (Figure 1).
- The Android operating system currently powers more than one **billion** smartphones and tablets.
- Each Android version is named after a dessert and the current version is 6.0 Marshmallow.

2009
2010
2011

Figure 1: Android growth in device activations

ANDROID STUDIO

- Android provides developers with everything they need to build very advanced app experiences.
- It facilitates a single application model that lets developers deploy their apps broadly for a wide range of devices, from phones to tablets and beyond, whilst simultaneously considering the hardware capabilities available on each device.
- Also, the **Android Studio** offers a comprehensive development tool with advanced features for **developing**, **debugging** and **packaging** Android apps.
- Using Android Studio IDE, they can develop for any available Android device or create virtual devices that emulate any hardware configuration.

ANDROID APP MARKETPLACE

- **Google Play** (Figure 2) is the premier marketplace for selling and distributing Android apps.
- When developers publish an app on **Google Play**, they reach the huge installed base of Android. As an open marketplace, Google Play puts them in control of how they sell their products.
- They can **publish** whenever they want, as often as they want, and to the customers they want. They can **distribute** broadly to all markets and devices or focus on specific segments, devices, or ranges of hardware capabilities.

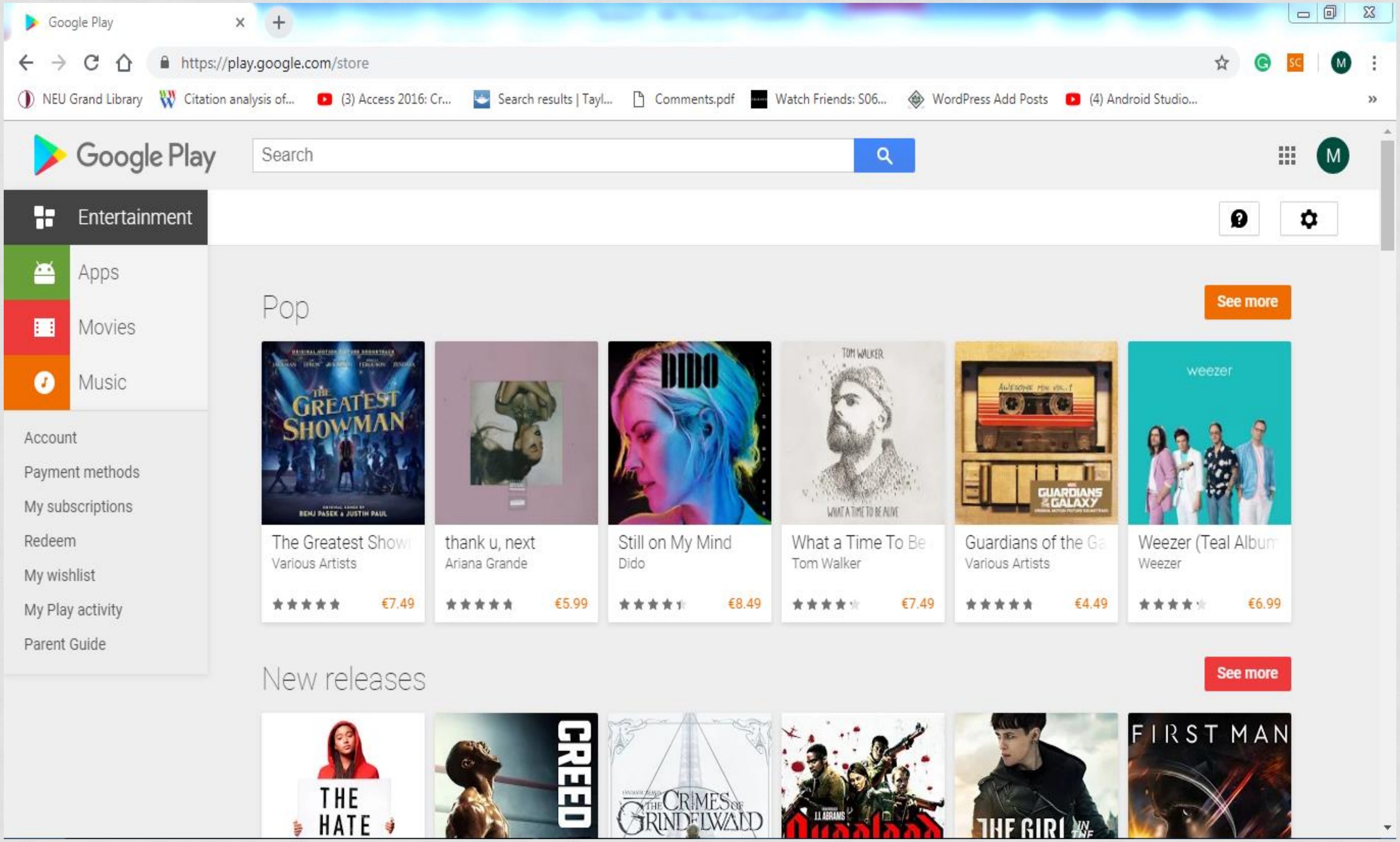


Figure 2: Google Play

ANDROID APP DEVELOPMENT SOFTWARE

In order to develop apps for the Android platform, you need:

- The Android System Development Kit (**SDK**), which provides you with the Application Programming Interface (**API)** libraries and developer tools necessary to **build**, **test** and **debug** apps for Android.

Before you start the development of your first Android app:

- Make sure that you have Java Platform (JDK) installed on your computer. It can be downloaded from:

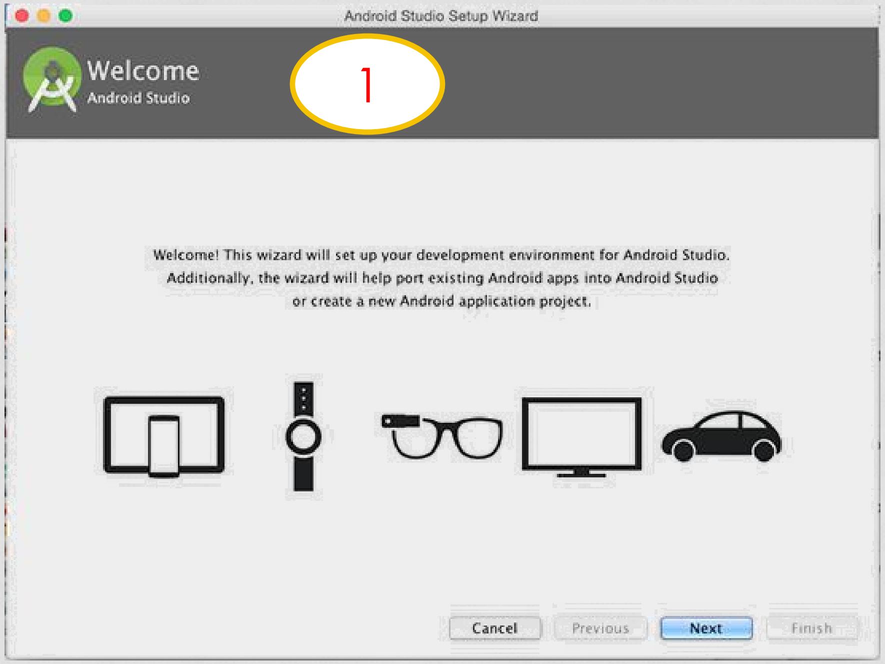
www.oracle.com

- Always select the **latest** edition.

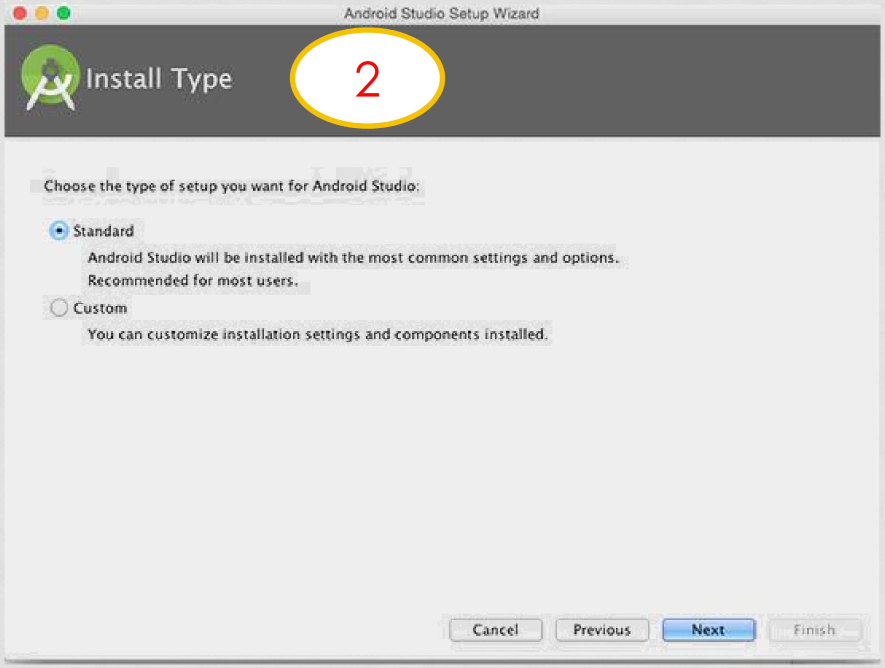
ANDROID APP DEVELOPMENT SOFTWARE

Installation Process

- You need to download the Android Studio, which includes the essential Android SDK components and intelligent code editor that contains advanced code completion, refactoring and code analysis. The powerful code editor helps you to be a more productive Android app developer.
- After you download Android Studio, unpack the ZIP file, click the *Installation* icon, keep all the default settings and click *Next* until installation is completed on your computer. Then click *Finish* (see images below).



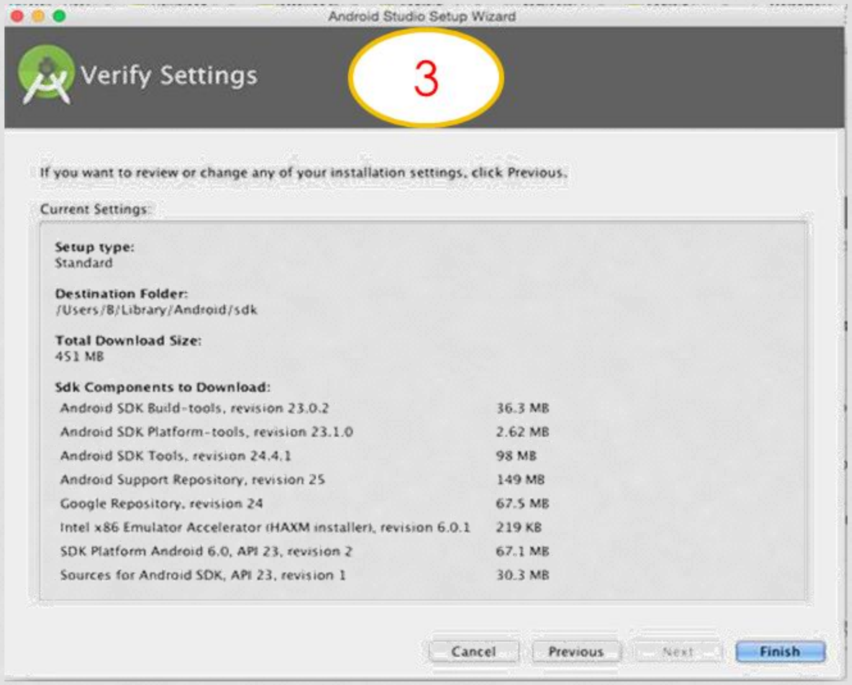
Welcome! This wizard will set up your development environment for Android Studio. Additionally, the wizard will help port existing Android apps into Android Studio or create a new Android application project.



Choose the type of setup you want for Android Studio:

[✓ Standard]
Android Studio will be installed with the most common settings and options. Recommended for most users.

Custom
You can customize installation settings and components installed.



If you want to review or change any of your installation settings, click Previous.

Current Settings:

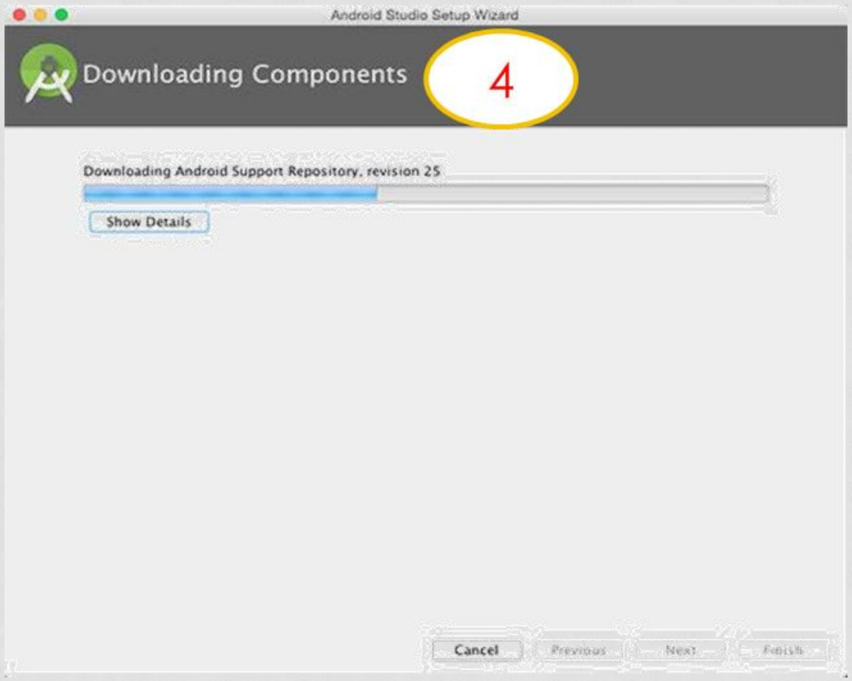
Setup type:
Standard

Destination Folder:
/Users/B/Library/Android/sdk

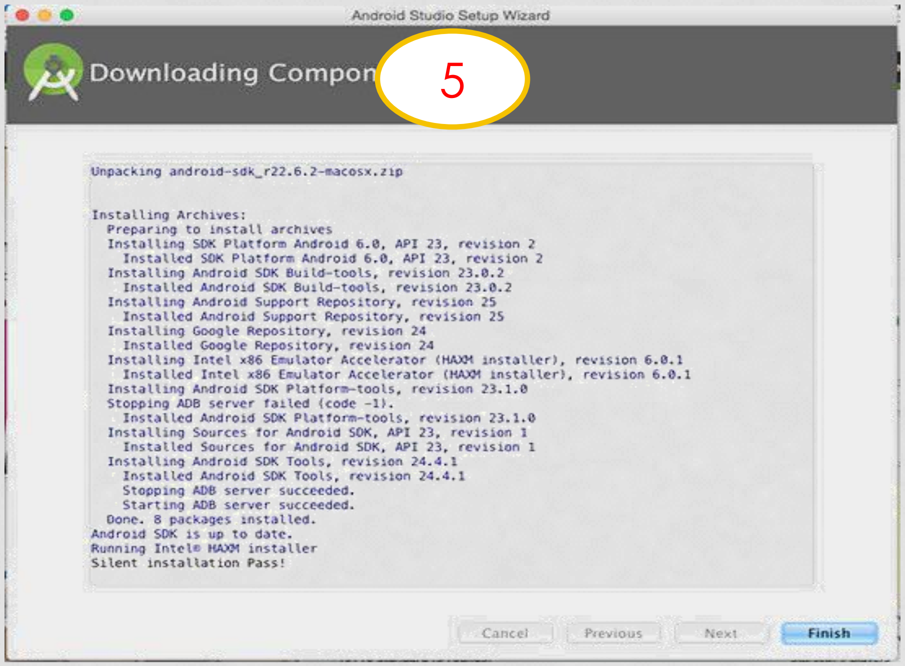
Total Download Size:
451 MB

Sdk Components to Download:

Android SDK Build-tools, revision 23.0.2	36.3 MB
Android SDK Platform-tools, revision 23.1.0	2.62 MB
Android SDK Tools, revision 24.4.1	98 MB
Android Support Repository, revision 25	149 MB
Google Repository, revision 24	67.5 MB
Intel x86 Emulator Accelerator (HAXM installer), revision 6.0.1	219 KB
SDK Platform Android 6.0, API 23, revision 2	67.1 MB
Sources for Android SDK, API 23, revision 1	30.3 MB

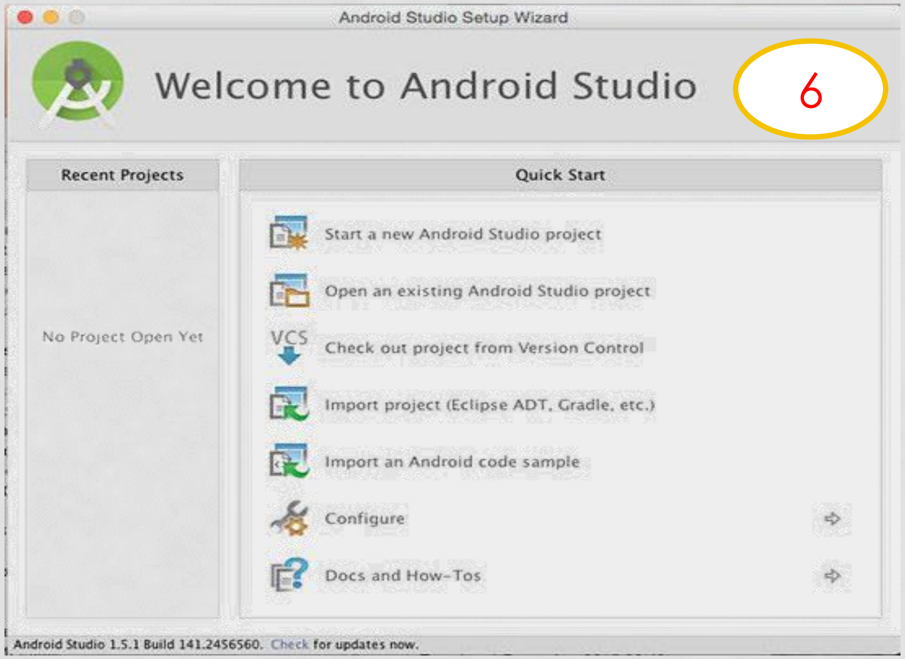


Downloading Android Support Repository, revision 25



```
Unpacking android-sdk_r22.6.2-macosx.zip


Installing Archives:
  Preparing to install archives
  Installing SDK Platform Android 6.0, API 23, revision 2
    Installed SDK Platform Android 6.0, API 23, revision 2
  Installing Android SDK Build-tools, revision 23.0.2
    Installed Android SDK Build-tools, revision 23.0.2
  Installing Android Support Repository, revision 25
    Installed Android Support Repository, revision 25
  Installing Google Repository, revision 24
    Installed Google Repository, revision 24
  Installing Intel x86 Emulator Accelerator (HAXM installer), revision 6.0.1
    Installed Intel x86 Emulator Accelerator (HAXM installer), revision 6.0.1
  Installing Android SDK Platform-tools, revision 23.1.0
  Stopping ADB server failed (code -1).
    Installed Android SDK Platform-tools, revision 23.1.0
  Installing Sources for Android SDK, API 23, revision 1
    Installed Sources for Android SDK, API 23, revision 1
  Installing Android SDK Tools, revision 24.4.1
    Installed Android SDK Tools, revision 24.4.1
    Stopping ADB server succeeded.
    Starting ADB server succeeded.
  Done. 8 packages installed.
Android SDK is up to date.
Running Intel® HAXM installer
Silent installation Pass!
```


Android Studio Setup Wizard
Welcome to Android Studio
6
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
Configure
Docs and How-Tos

Android Studio 1.5.1 Build 141.2456560. Check for updates now.

HOW TO INSTALL AND RUN YOUR APP ON A REAL DEVICE

- If you have a device running Android, here's how to install and run your app.
- **Set Up Your Device**
- Plug in your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device. For help installing drivers, see the OEM USB Drivers document.
- Enable **USB debugging** on your device.
- On most devices running Android 3.2 or older, you can find the option under **Settings > Applications > Development**.
- On Android 4.0 and newer, it's in **Settings > Developer options**.

HOW TO INSTALL AND RUN YOUR APP ON A REAL DEVICE

- On Android 4.2 and newer, **Developer options** is hidden by default. To make it available, go to **Settings > About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.
- Select one of your project's files and click **Run**▶ from the toolbar.
- In the **Choose Device** window that appears, select the **Choose a running device** radio button, select your device, and click **OK**.
- Android Studio installs the app on your connected device and starts it.

HOW TO INSTALL AND RUN YOUR APP ON THE ANDROID EMULATOR

To run your app on the **emulator**, you need to first create an Android Virtual Device (**AVD**).

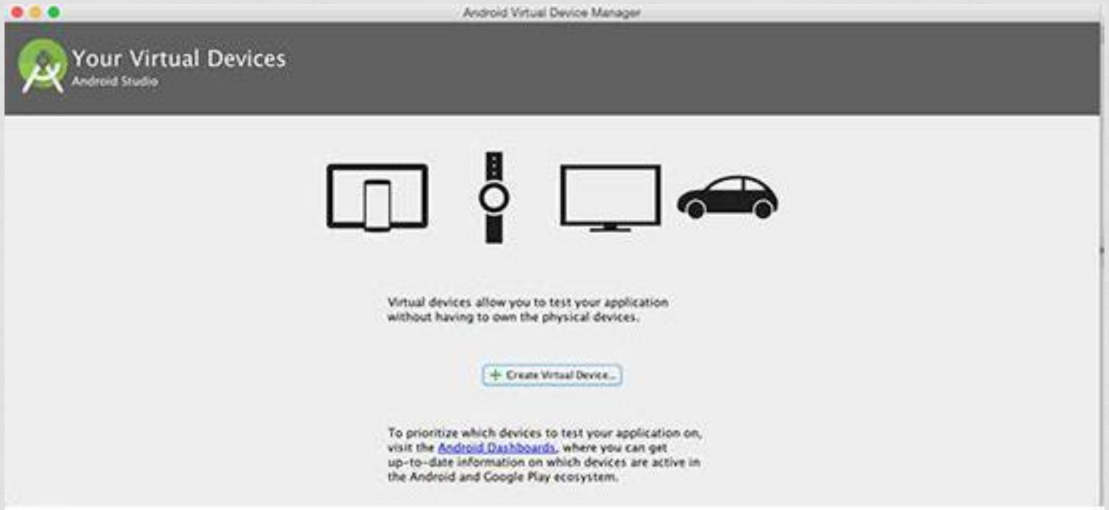
- An AVD is a device configuration for the Android emulator that allows you to model a specific device.

Create an AVD

- 1. Launch the Android Virtual Device Manager:
- In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon in the toolbar. The *AVD Manager* screen appears.
- 2. On the AVD Manager main screen, click **Create Virtual Device**.

HOW TO INSTALL AND RUN YOUR APP ON THE ANDROID EMULATOR

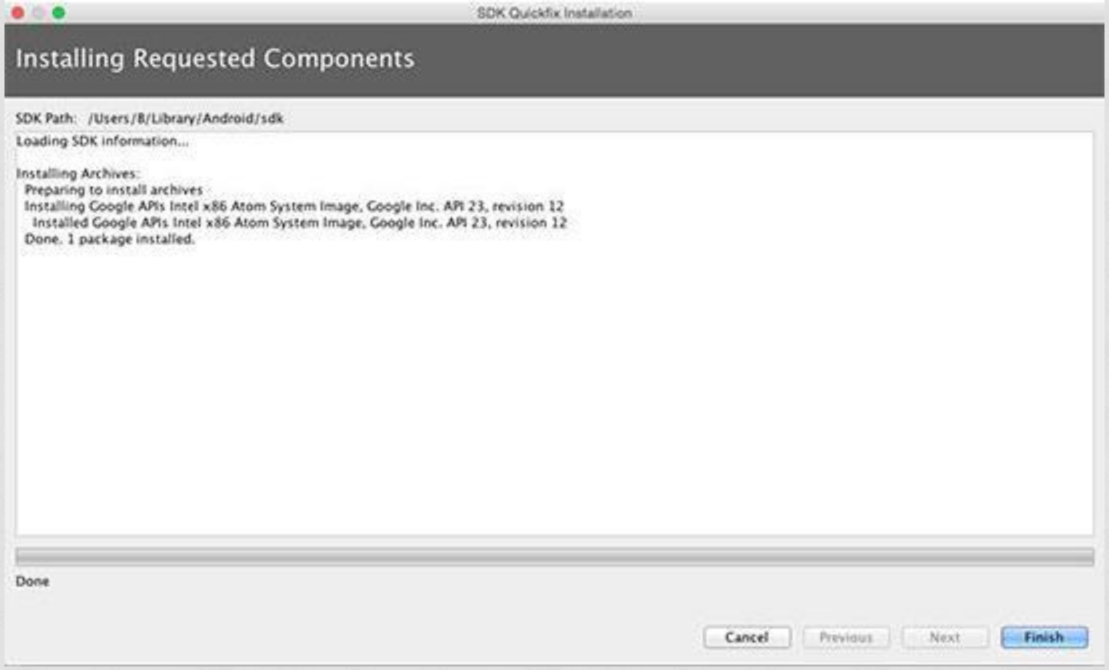
- In the Select Hardware window, select a device configuration, such as Nexus 6, then click **Next**.
- Select the desired system version for the AVD and click **Next**.
- Verify the configuration settings, then click **Finish**.



Your Virtual Devices
Android Studio

Virtual devices allow you to test your application without having to own the physical devices.

To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.



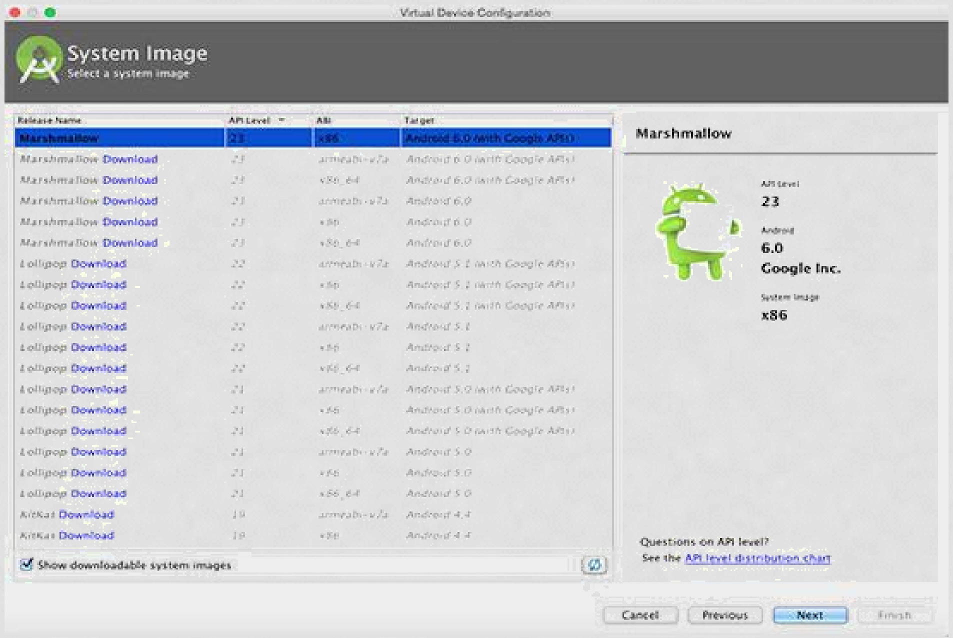
Installing Requested Components

SDK Path: /Users/B/Library/Android/sdk

```
Loading SDK information...

Installing Archives:
  Preparing to install archives
  Installing Google APIs Intel x86 Atom System Image, Google Inc. API 23, revision 12
    Installed Google APIs Intel x86 Atom System Image, Google Inc. API 23, revision 12
  Done. 1 package installed.
```

Done



Virtual Device Configuration

System Image

Select a system image

Release Name	API Level	ABI	Target
Marshmallow	23	x86	Android 6.0 (with Google APIs)
Marshmallow Download	23	armeabi-v7a	Android 6.0 (with Google APIs)
Marshmallow Download	23	x86_64	Android 6.0 (with Google APIs)
Marshmallow Download	23	armeabi-v7a	Android 6.0
Marshmallow Download	23	x86	Android 6.0
Marshmallow Download	23	x86_64	Android 6.0
Lollipop Download	22	armeabi-v7a	Android 5.1 (with Google APIs)
Lollipop Download	22	x86	Android 5.1 (with Google APIs)
Lollipop Download	22	x86_64	Android 5.1 (with Google APIs)
Lollipop Download	22	armeabi-v7a	Android 5.1
Lollipop Download	22	x86	Android 5.1
Lollipop Download	22	x86_64	Android 5.1
Lollipop Download	21	armeabi-v7a	Android 5.0 (with Google APIs)
Lollipop Download	21	x86	Android 5.0 (with Google APIs)
Lollipop Download	21	x86_64	Android 5.0 (with Google APIs)
Lollipop Download	21	armeabi-v7a	Android 5.0
Lollipop Download	21	x86	Android 5.0
Lollipop Download	21	x86_64	Android 5.0
KitKat Download	19	armeabi-v7a	Android 4.4
KitKat Download	19	x86	Android 4.4

✓ Show downloadable system images

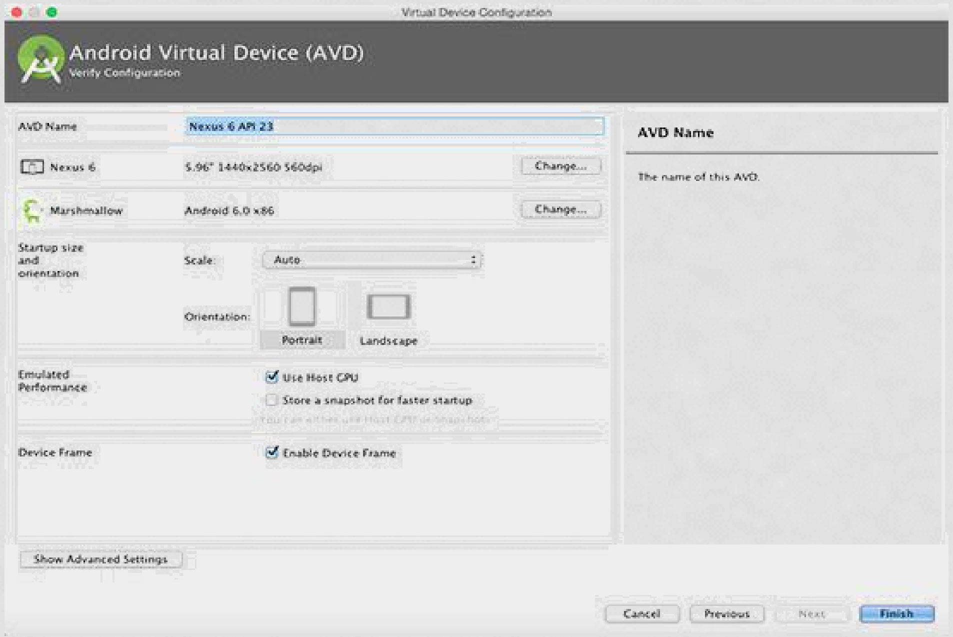
Marshmallow

API Level: 23

Android: 6.0, Google Inc.

System Image: x86

Questions on API level?
See the API level distribution chart



Virtual Device Configuration

Android Virtual Device (AVD)

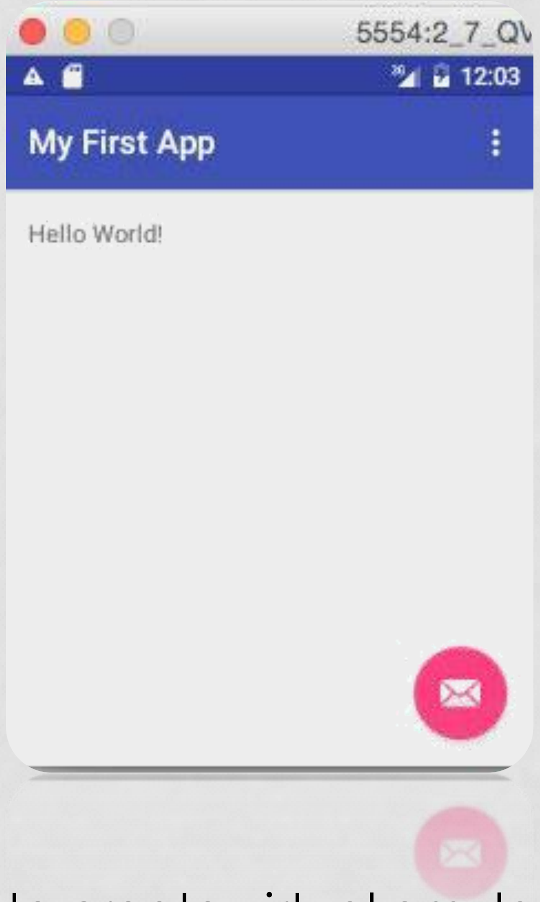
Verify Configuration

AVD Name	Nexus 6 API 23
Nexus 6	5.96" 1440x2560 560dpi
Marshmallow	Android 6.0 x86
Startup size and orientation	Scale: Auto; Orientation: [✓ Portrait] Landscape
Emulated Performance	✓ Use Host CPU; ☐ Store a snapshot for faster startup
Device Frame	✓ Enable Device Frame

You can either use Host CPU or snapshots

AVD Name

The name of this AVD.



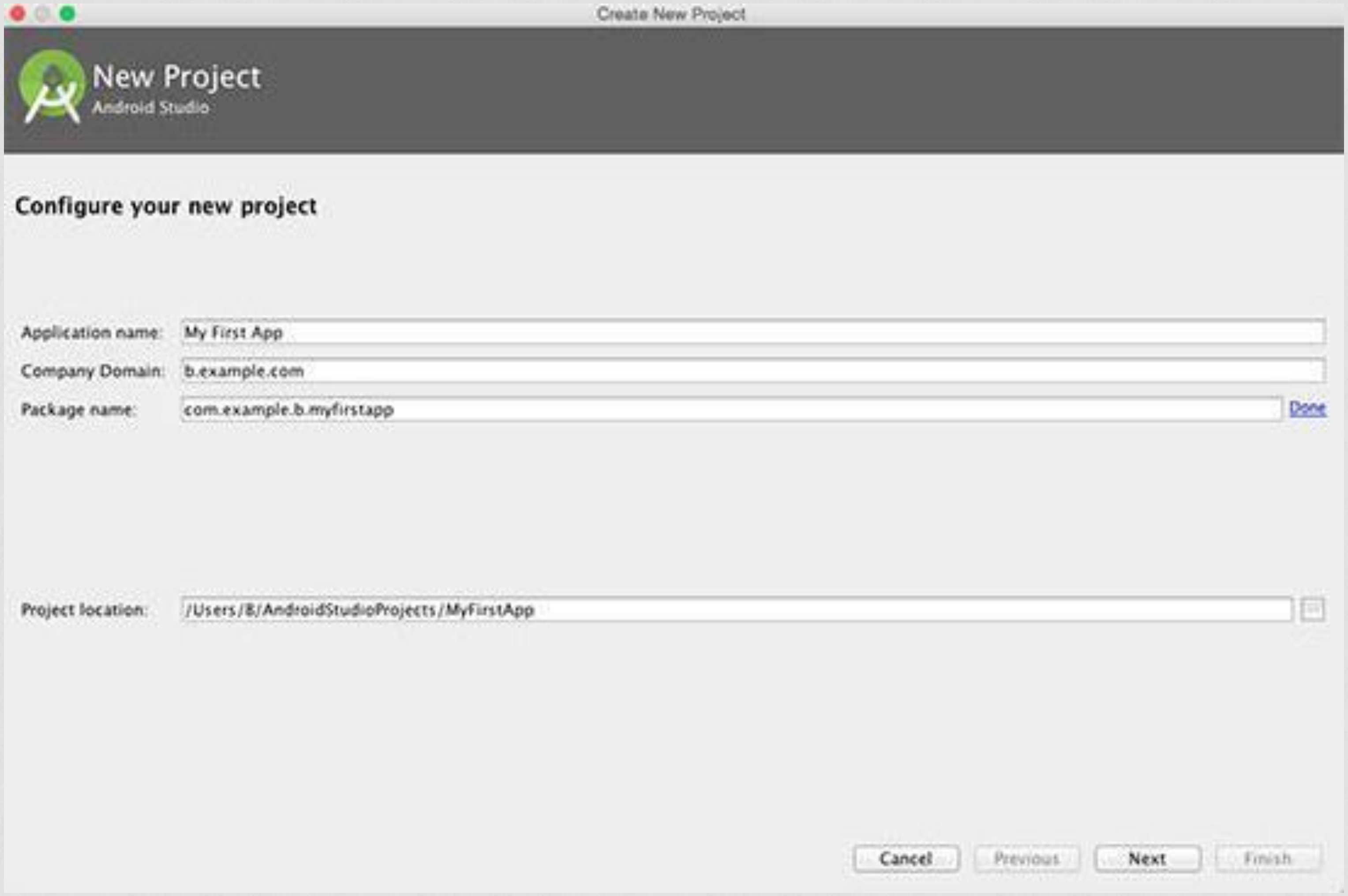
List of images to create virtual emulator and running the app

CREATE NEW PROJECT

Create your first app:

1. In Android Studio, create a new project:

- If you don't have a project opened, in the **Welcome** screen, click **New Project**.
- If you have a project opened, from the **File** menu, select **New Project**. The *Create New Project* screen appears.



Configure your new project

Application name: My First App
Company Domain: b.example.com
Package name: com.example.b.myfirstapp

Project location: /Users/B/AndroidStudioProjects/MyFirstApp

CREATE NEW PROJECT

2. Fill out the fields on the screen, and click **Next**.

- **Application Name:** is the app name that appears to users. For this project, use ‘My First App.’
- **Company domain:** provides a qualifier that will be appended to the package name: Android Studio will remember this qualifier for each new project you create (leave default values for now).
- **Package name:** is the fully qualified name for the project (following the same rules as those for naming packages in the Java programming language). Your package name must be **unique** across all packages installed on the Android system.
    - You can **Edit** this value independently from the application name or the company domain (leave default values for now).
- **Project location:** is the directory on your system that holds the project files.

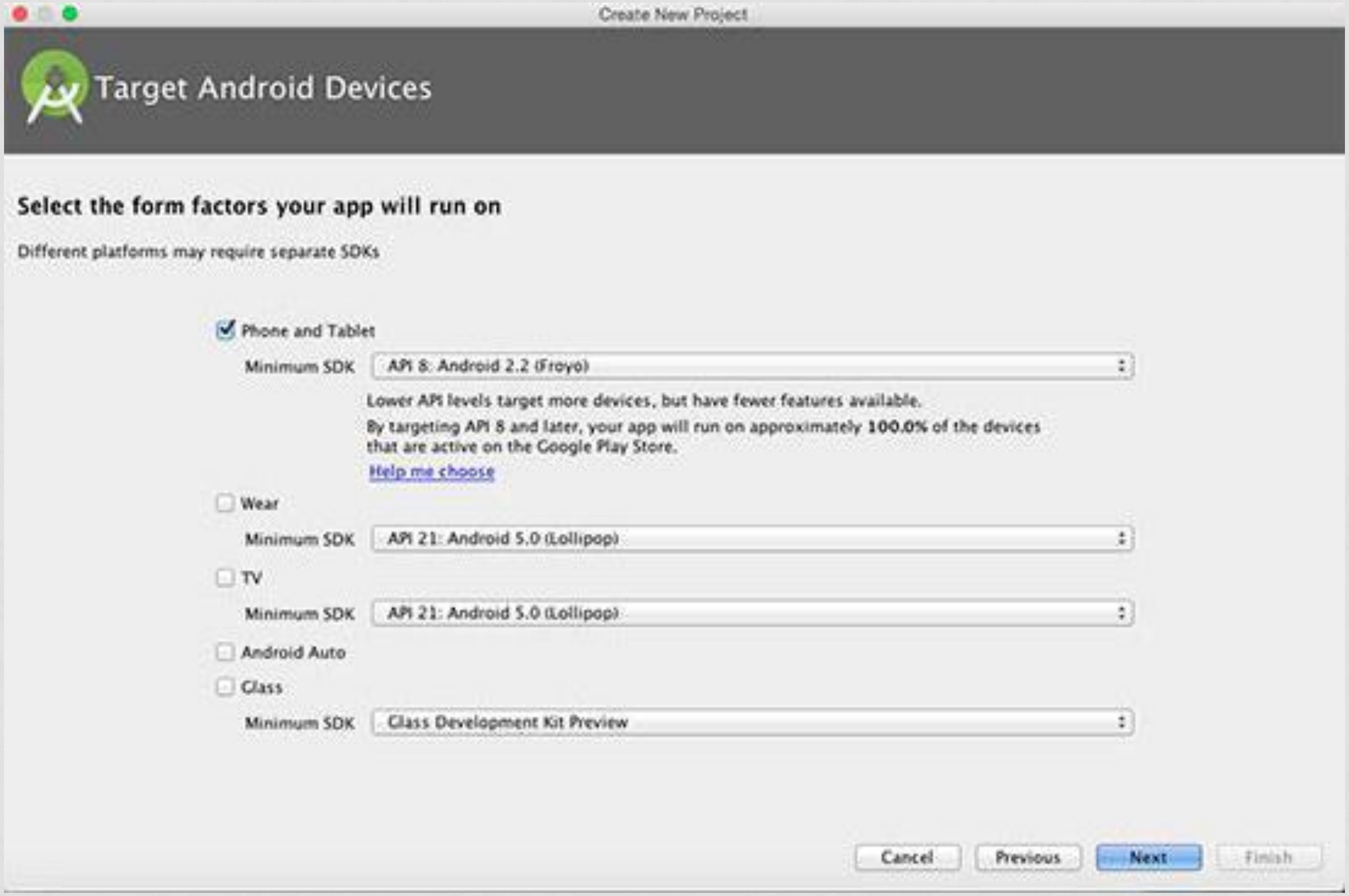
CREATE NEW PROJECT

3. Under **Select the form factors your app will run on**, check the box for **Phone and Tablet**.

For **Minimum SDK**, select **API 8: Android 2.2 (Froyo)**.

- The Minimum Required SDK is the earliest version of Android that your app supports, indicated using the API level.
- To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set.
- If any feature of your app is possible only on newer versions of Android, and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it.

5. Leave all of the other options (TV, Wear and Glass) unchecked and click **Next.**



Select the form factors your app will run on

Different platforms may require separate SDKs

[✓] Phone and Tablet

Minimum SDK: API 8: Android 2.2 (Froyo)

Lower API levels target more devices, but have fewer features available.

By targeting API 8 and later, your app will run on approximately **100.0%** of the devices that are active on the Google Play Store.

Help me choose

[ ] Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] Android Auto

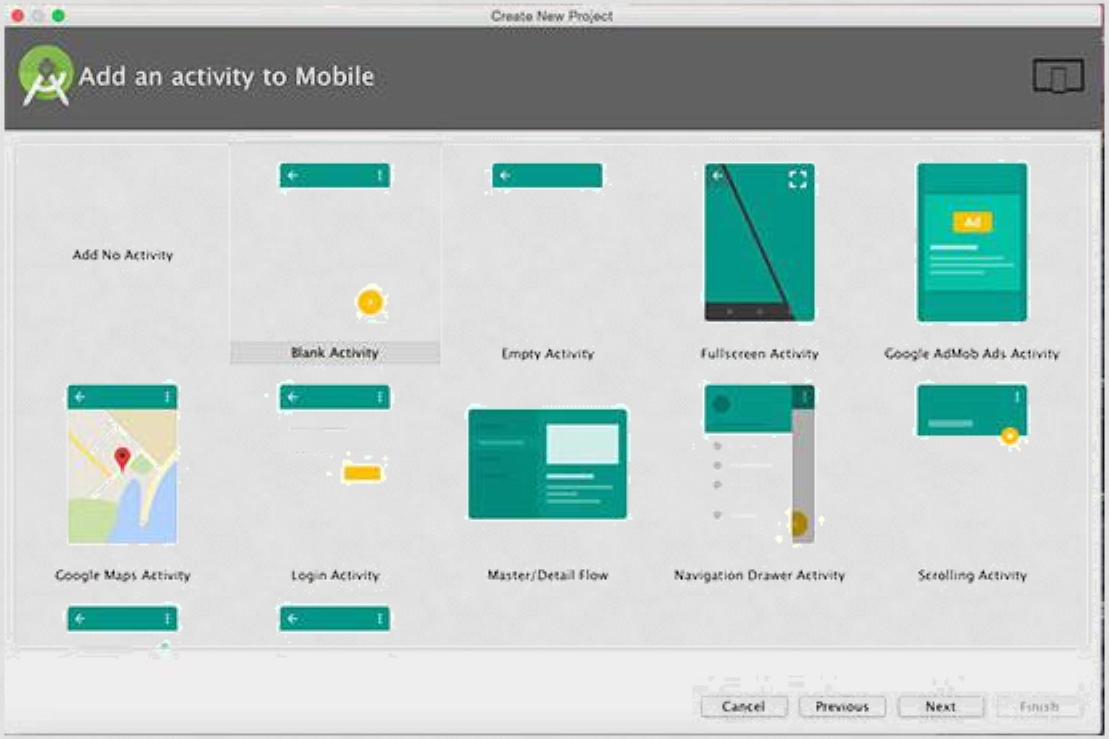
[ ] Glass

Minimum SDK: Glass Development Kit Preview

Configuring Your App

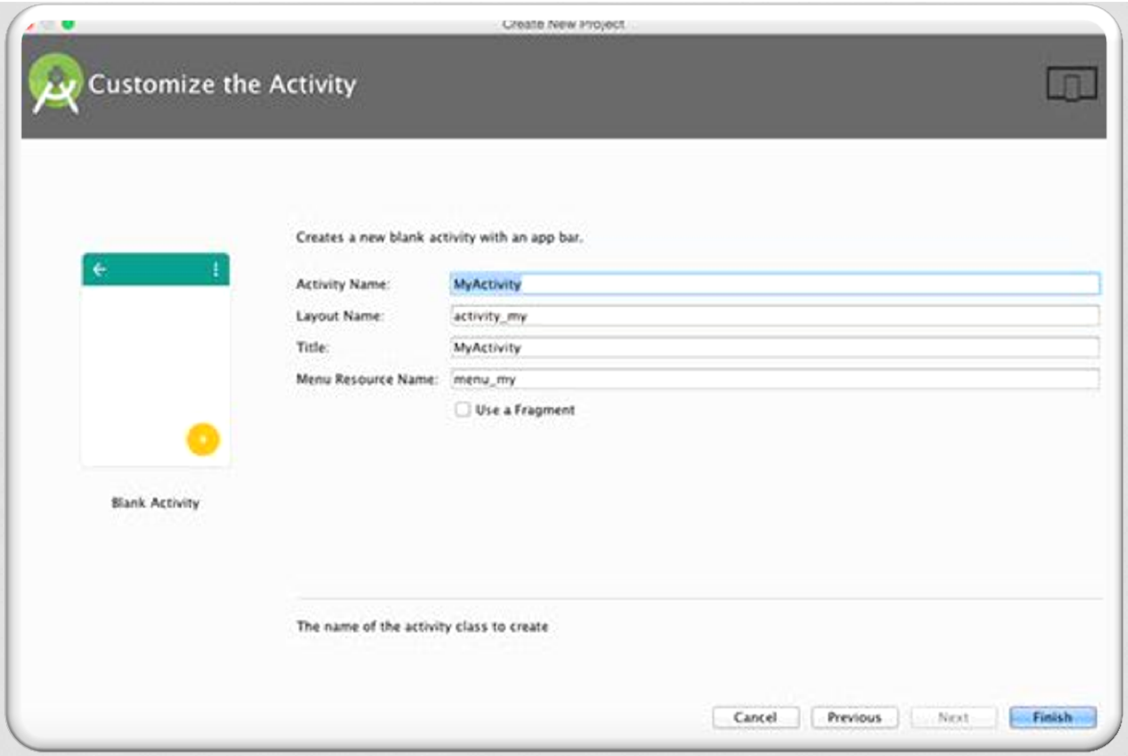
CREATE NEW PROJECT

6. Under **Add an activity to <*template*>**, select **Blank Activity** and click **Next**.



CREATE NEW PROJECT

- Under **Customize the Activity**, change the **Activity Name** to *MyActivity*.
- The **Layout Name** changes to *activity_my*, and the **Title** to *MyActivity*.
- The **Menu Resource Name** is *menu_my*.
- Then click the **Finish** button to create the project as shown in figures 8 and 9.


Create New Project

Customize the Activity

Creates a new blank activity with an app bar.

Activity Name: MyActivity

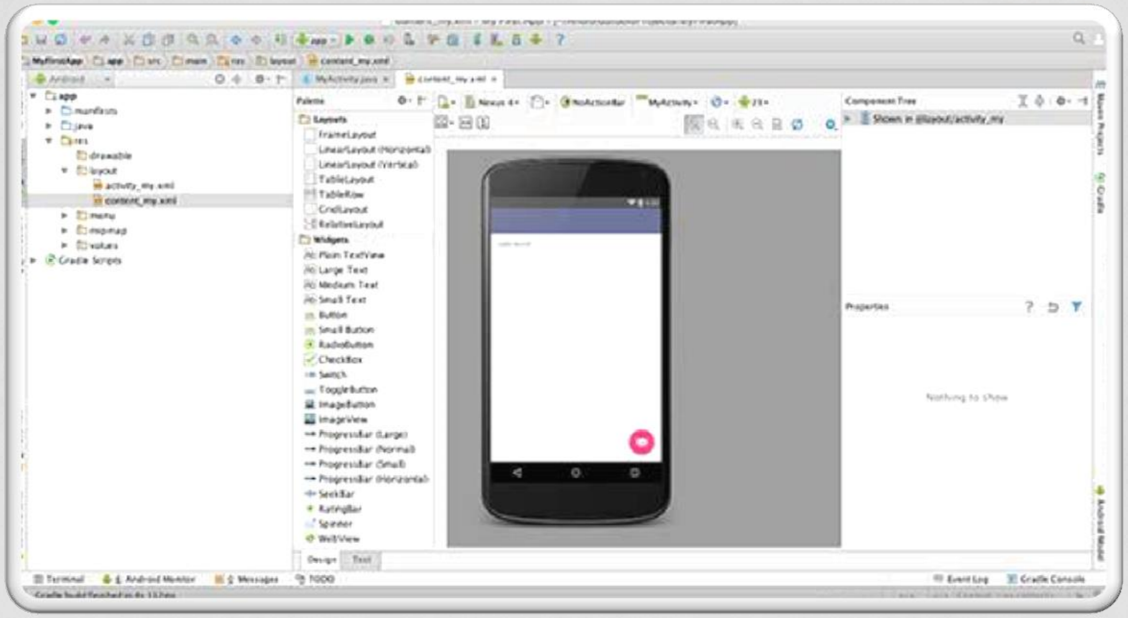
Layout Name: activity_my

Title: MyActivity

Menu Resource Name: menu_my

☐ Use a Fragment

The name of the activity class to create



ANDROID STUDIO DEFAULT FILES

Your Android project is now a basic ‘Hello World’ app that contains some **default files**. Take a moment to review the most important of these:

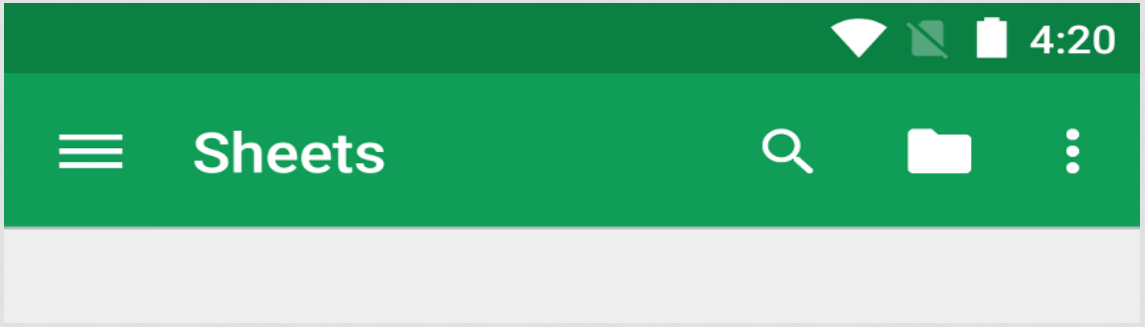
1. app/src/main/res/layout/activity_my.xml

- This XML layout file is for the activity you added when you created the project with Android Studio.
- Following the New Project workflow, Android Studio presents this file with both a **text view** and a **preview** of the **screen** UI.
- The file contains some default interface elements from the material design library, including the app bar and a floating action button.
- It also includes a separate layout file with the main content.

ANDROID STUDIO DEFAULT FILES

The *app bar*, also known as the ***action bar***, is one of the most important design elements in your app's activities, because it provides a visual structure and interactive elements that are familiar to users. Using the app bar makes your app consistent with other Android apps, allowing users to quickly understand how to operate your app and have a great experience. The key functions of the app bar are as follows:

- A dedicated space for giving your app an identity and indicating the user's location in the app.
- Access to important actions in a predictable way, such as search.
- Support for navigation and view switching (with tabs or drop-down lists).



ANDROID STUDIO DEFAULT FILES

2. app/src/main/res/layout/content_my.xml

- This XML layout file resides in activity_my.xml, and contains some settings and a TextView element that displays the message, "Hello world!".

3.app/src/main/java/com.mycompany.myfirstapp/MyActivity.java

- A tab for this file appears in Android Studio when the New Project workflow finishes. When you select the file, you see the class definition for the activity you created. When you build and run the app, the Activity class starts the activity and loads the layout file that says "Hello World!"

ANDROID STUDIO DEFAULT FILES

4.app/src/main/AndroidManifest.xml

- The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll revisit this file as you follow these lessons and add more components to your app.

APP MANIFEST OVERVIEW

- Every app project must have an **AndroidManifest.xml** file at the root of the project source set.
- The manifest file describes **essential** information about:

❑The app's package name, which usually matches your code's namespace.

❑The components of the app, which include all activities, services, broadcast receivers, and content providers.

❑The permissions that the app needs in order to access protected parts of the system or other apps.

❑The hardware and software features the app requires, which affects which devices can install the app from Google Play.

ANDROID STUDIO DEFAULT FILES

5. app/build.gradle

- Android Studio uses Gradle to compile and build your app. There is a build.gradle file for each module of your project, as well as a build.gradle file for the entire project. Usually, you're only interested in the build.gradle file for the module, in this case the app or application module. This is where your app's build dependencies are set, including the default Config settings:

❑ **compiledSdkVersion:** is the platform version against which you will compile your app. By default, this is set to the latest version of Android available in your SDK (it should be Android 4.1 or greater; if you don't have such a version available, you must install one using the SDK Manager.). You can still build your app to support older versions, but setting this to the latest version allows you to enable new features and optimize your app for a great user experience on the latest devices.

❑ **applicationId:** is the fully qualified package name for your application that you specified during the New Project workflow.

ANDROID STUDIO DEFAULT FILES

- **minSdkVersion:** is the Minimum SDK version you specified during the New Project workflow. This is the earliest version of the Android SDK that your app supports.
- **targetSdkVersion:** indicates the highest version of Android with which you have tested your application. As new versions of Android become available, you should test your app on the new version and update this value to match the latest API level, and thereby take advantage of new platform features. For more information,
- read Supporting Different Platform Versions. See Building Your Project with Gradle for more information about Gradle.

ANDROID STUDIO DEFAULT FILES

6.drawable-<*density*>/

- Directories for drawable resources, other than launcher icons, designed for various densities.

7.layout/

- Directory for files that define your app's user interface, like activity_my.xml, discussed above, which describes a basic layout for the MyActivity class.

8.menu/

- Directory for files that define your app's menu items.

ANDROID STUDIO DEFAULT FILES

8. mipmap/

- Launcher icons reside in the mipmap/ folder rather than the drawable/ folders. This folder contains the ic_launcher.png image that appears when you run the default app.

9.values/

- Directory for other XML files that contain a collection of resources, such as string and color definitions.

ANDROID APP

- The graphical user interface for an Android app is built using a hierarchy of **View** and **ViewGroup** objects.
- View objects are usually UI widgets such as **buttons** or **text fields**.
- ViewGroup objects are **invisible** view containers that define how the child views are laid out, such as in a grid or a vertical list.
- Android provides an XML vocabulary that corresponds to the subclasses of View and ViewGroup, so you can define your UI in XML using a hierarchy of UI elements, as illustrated in Figure 11.

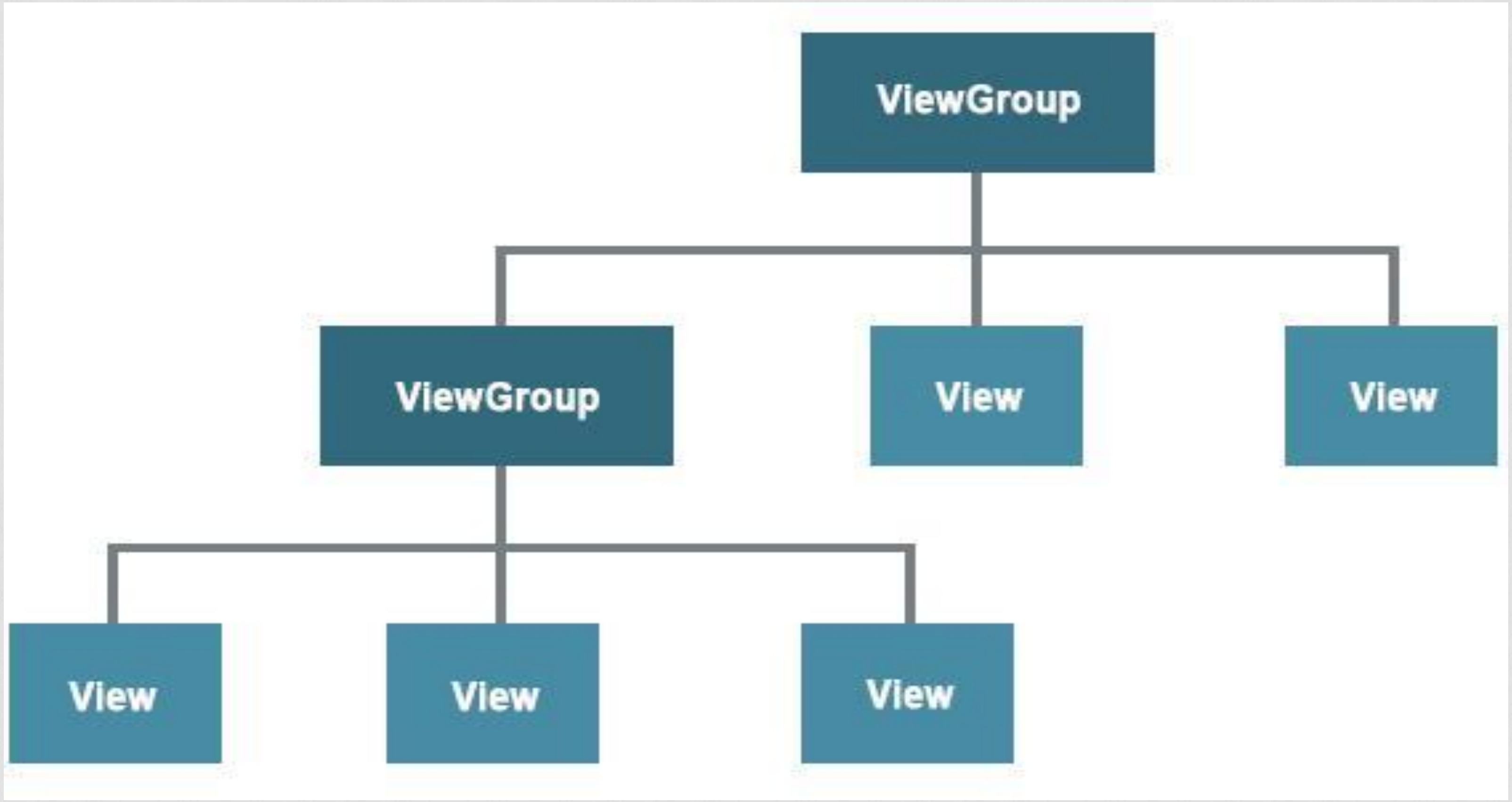
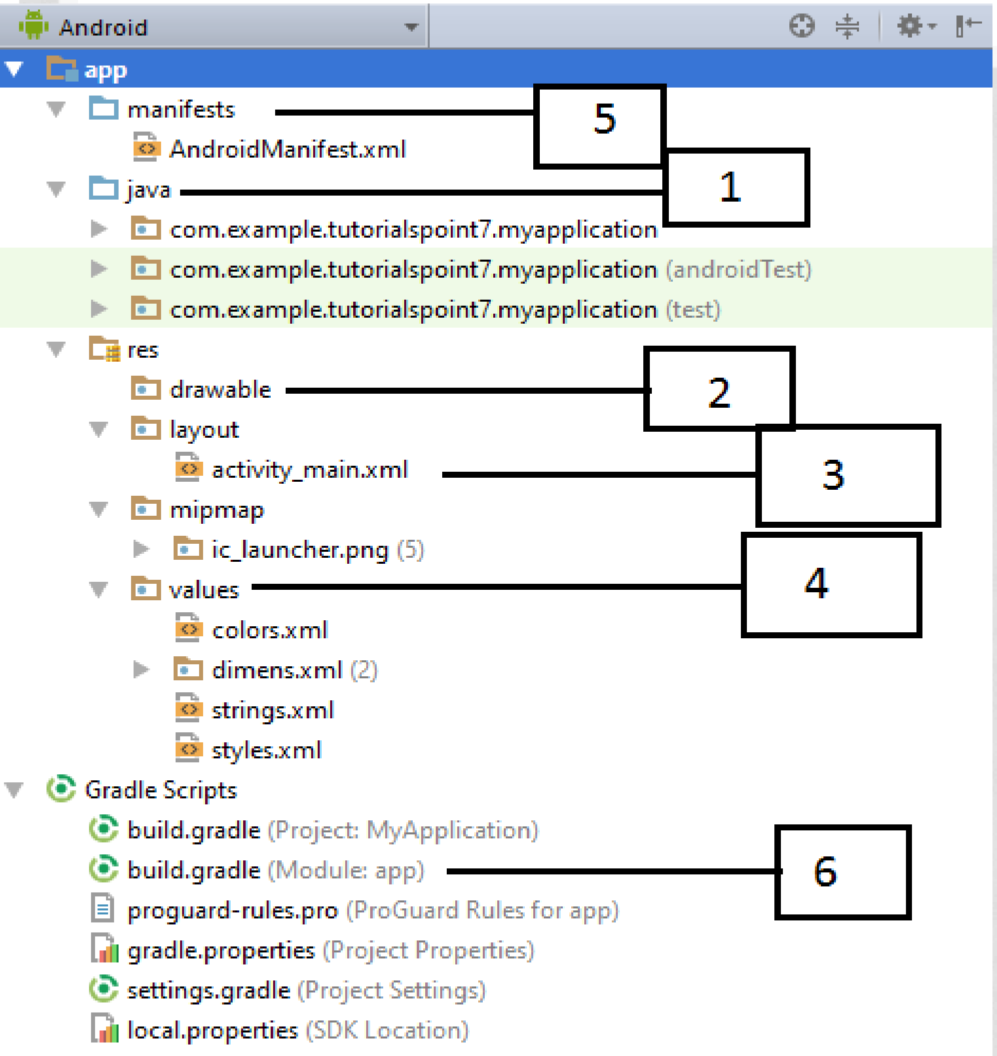


Illustration of ViewGroup objects form branches in the layout that contain other View objects

ANATOMY OF ANDROID APPLICATION



Android

app
- manifests
  - AndroidManifest.xml
- java
  - com.example.tutorialspoint7.myapplication
  - com.example.tutorialspoint7.myapplication (androidTest)
  - com.example.tutorialspoint7.myapplication (test)
- res
  - drawable
  - layout
    - activity_main.xml
  - mipmap
    - ic_launcher.png (5)
  - values
    - colors.xml
    - dimens.xml (2)
    - strings.xml
    - styles.xml

Gradle Scripts
- build.gradle (Project: MyApplication)
- build.gradle (Module: app)
- proguard-rules.pro (ProGuard Rules for app)
- gradle.properties (Project Properties)
- settings.gradle (Project Settings)
- local.properties (SDK Location)

Before you run your app, you should be aware of a few directories and files in the Android project.

ANATOMY OF ANDROID APPLICATION

1	Java This contains the .java source files for your project. By default, it includes an MainActivity.java source file having an activity class that runs when your app is launched using the app icon.
2	res/drawable-hdpi This is a directory for drawable objects that are designed for high-density screens.
3	res/layout This is a directory for files that define your app's user interface.

ANATOMY OF ANDROID APPLICATION

4	res/values This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
5	AndroidManifest.xml This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
6	Build.gradle This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

THE MAIN ACTIVITY FILE

The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application

```
package com.example.helloworld;


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);
  }
}
```

- Here, **R.layout.activity_main** refers to the **activity_main.xml** file located in the **res/layout** folder.
- The **onCreate()** method is one of many methods that are figured when an activity is loaded.

THE MANIFEST FILE

Whatever component you develop as a part of your application, you must declare all its components in a ***manifest.xml*** which resides at the root of the application project directory.

This file works as an interface between **Android OS** and **your application**, so if you do not declare your component in this file, then it will not be considered by the OS.

For example, a default manifest file will look like as following file :

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    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 MANIFEST FILE

- Here **<application>...</application>** tags enclosed the components related to the application.
- Attribute ***android:icon*** will point to the application icon available under ***res/drawable-hdpi***.
- The application uses the **image** named **ic_launcher.png** located in the **drawable** folders
- The **<activity>** tag is used to specify an activity and ***android:name*** attribute specifies the fully qualified class name of the *Activity* subclass and the ***android:label*** attributes specifies a string to use as the label for the activity.
- You can specify multiple activities using **<activity>** tags.

THE MANIFEST FILE

- The **action** for the intent filter is named ***android.intent.action.MAIN*** to indicate that this activity serves as the entry point for the application.
- The **category** for the intent-filter is named ***android.intent.category.LAUNCHER*** to indicate that the application can be launched from the device's launcher icon.
- The ***@string*** refers to the ***strings.xml*** file explained below. Hence, ***@string/app_name*** refers to the ***app_name*** string defined in the **strings.xml** file, which is "HelloWorld". Similar way, other strings get populated in the application.

THE MANIFEST FILE

Following is the list of tags which you will use in your **manifest** file to specify different Android application components :

- <activity>elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

THE STRINGS FILE

The **strings.xml** file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as following file

```
<resources>

  <string name="app_name">HelloWorld</string>

  <string name="hello_world">Hello world!</string>

  <string name="menu_settings">Settings</string>

  <string name="title_activity_main">MainActivity</string>

</resources>
```

THE LAYOUT FILE

The **activity_main.xml** is a layout file available in ***res/layout*** directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout

```
<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" >
     <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:padding="@dimen/padding_medium"
      android:text="@string/hello_world"
      tools:context=".MainActivity" />

</RelativeLayout>
```

- This is an example of simple ***RelativeLayout*** .
- The ***TextView*** is an Android control used to build the GUI and it have various attributes like ***android:layout_width***, ***android:layout_height*** etc which are being used to set its width and height etc…
- The ***@string*** refers to the strings.xml file located in the **res/values** folder. Hence, **@string/hello_world** refers to the hello string defined in the **strings.xm**l file, which is "Hello World!".

EXAMPLE OF LAYOUT FILE

- In Android Studio, from the **res/layout** directory, edit the content_my.xml file.
- Within the <LinearLayout> element, define a <Button> element immediately following the <EditText> element.
- Set the button's **width** and **height** attributes to "**wrap_content**", so the button is only as big as necessary to **fit the button's text label**.
- Define the button's text label with the android:text attribute; set its value to the button_send string resource you defined in the previous section.

YOUR <LINEARLAYOUT> SHOULD LOOK LIKE THIS:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="match_parent"
app:layout_behavior="@string/appbar_scrolling_view_behavior"
tools:showIn="@layout/activity_my">
<EditText
android:id="@+id/edit_message"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="@string/edit_message" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_send" />
</LinearLayout>
```

ATTRIBUTES OF OBJECTS

- This button doesn't need the android:id attribute because it won't be referenced from the activity code.
- The layout is currently designed so that both the EditText and Button widgets are only as big as necessary to fit their content, as Figure 12 shows.

ATTRIBUTES OF OBJECTS

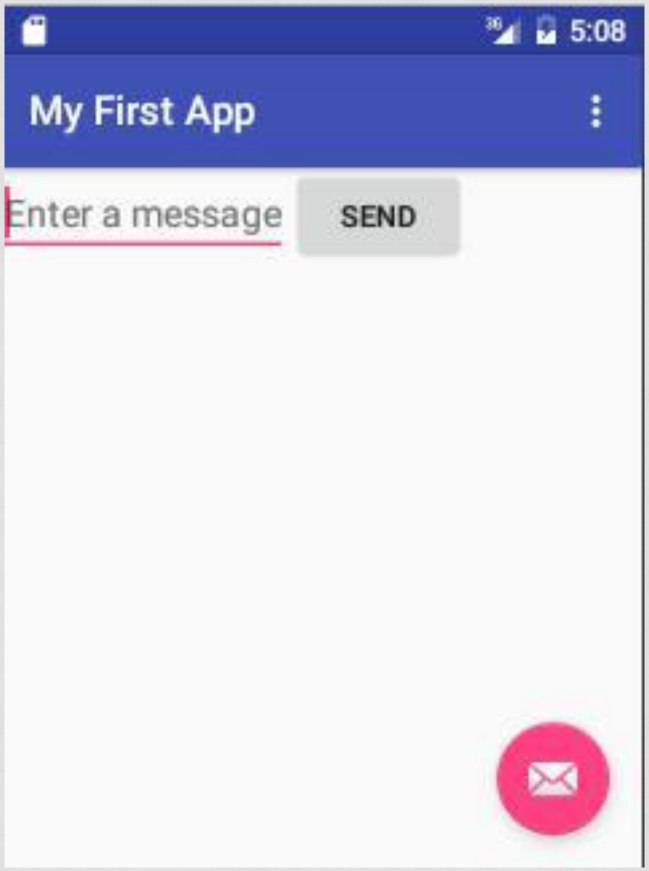
- It would be nice to fill the unused screen width with the text field. You can do this inside a LinearLayout with the *weight* property, which you can specify using the android:layout_weight attribute.

- The weight value is a number that specifies the amount of remaining space each view should consume, relative to the amount consumed by sibling views. The way this works is similar to the amount of ingredients in a drink recipe: "2 parts soda, 1 part syrup" means two-thirds of the drink is soda. For example, if you give one view a weight of 2 and another one a weight of 1, the sum is 3, so the first view fills 2/3 of the remaining space and the second view fills the rest. If you add a third view and give it a weight of 1, then the first view (with a weight of 2) now gets 1/2 the remaining space, while the remaining two each get 1/4.

- The default weight for all views is 0, so if you specify any weight value greater than 0 to only one view, then that view fills whatever space remains after all views are given the space they require.

ANDROID APP

- This layout is applied by the default Activity class that the SDK tools generated when you created the project. Run the app to see the results:
- In Android Studio, from the toolbar, click **Run** ▶


5:08
My First App
Enter a message
SEND