# Android Developer Fundamentals (Version 2) — Concepts

Android Developer Fundamentals (V2) is an instructor-led course created by the Google Developers Training team. In this course, you learn basic Android programming concepts and build a variety of apps, starting with Hello World and working your way up to apps that schedule jobs, update settings, and use Architecture Components.

Version 2 of the course is available as of September 2018. The course has been updated to reflect best practices for more recent versions of the Android framework and Android Studio. The original Android Developer Fundamentals (V1) course is still available, if you need to refer to it.

Android Developer Fundamentals prepares you to take the exam for the Associate Android Developer certification.

This course is intended to be taught in a classroom, but all the materials are online, so if you like to learn by yourself, go ahead!

## Prerequisites

Android Developer Fundamentals is intended for new and experienced developers who already have Java programming experience and now want to learn to build Android apps.

## Course materials

The course materials include:

- This concept reference, which teaches subjects you need to learn to complete the exercises in the practical workbook. Some lessons are purely conceptual and do not have an accompanying practical.
- The practical codelabs: Codelabs for Android Developer Fundamentals (V2).
- Slide decks (for optional use by instructors)

## What topics are covered?

Android Developer Fundamentals includes four teaching units, which are described in What does the course cover?

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

*Developed by the Google Developers Training Team*

*Last updated: September 2018*

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Unit 1: Get started

**Lesson 1: Build your first app**

# 1.0: Introduction to Android

**Contents:**



## What is Android?

Android is an operating system and programming platform developed by Google for mobile phones and other mobile devices, such as tablets. It can run on many different devices from many different manufacturers. Android includes a software development kit (SDK) that helps you write original code and assemble software modules to create apps for Android users. Android also provides a marketplace to distribute apps. All together, Android represents an *ecosystem* for mobile apps.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/


# Why develop apps for Android?

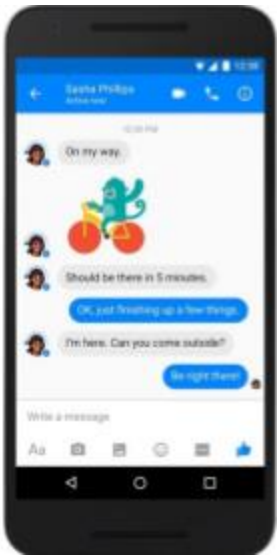
Developers create apps for a variety of reasons. They may need to address business requirements or build new services or businesses, or they may want to offer games and other types of content for users. Developers choose to develop for Android in order to reach the majority of mobile device users.

## Most popular platform for mobile apps

As the world's most popular mobile platform, Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It has the largest installed base of any mobile platform and is still growing fast. Every day another million users power up their Android-powered devices for the first time and start looking for apps, games, and other digital content.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

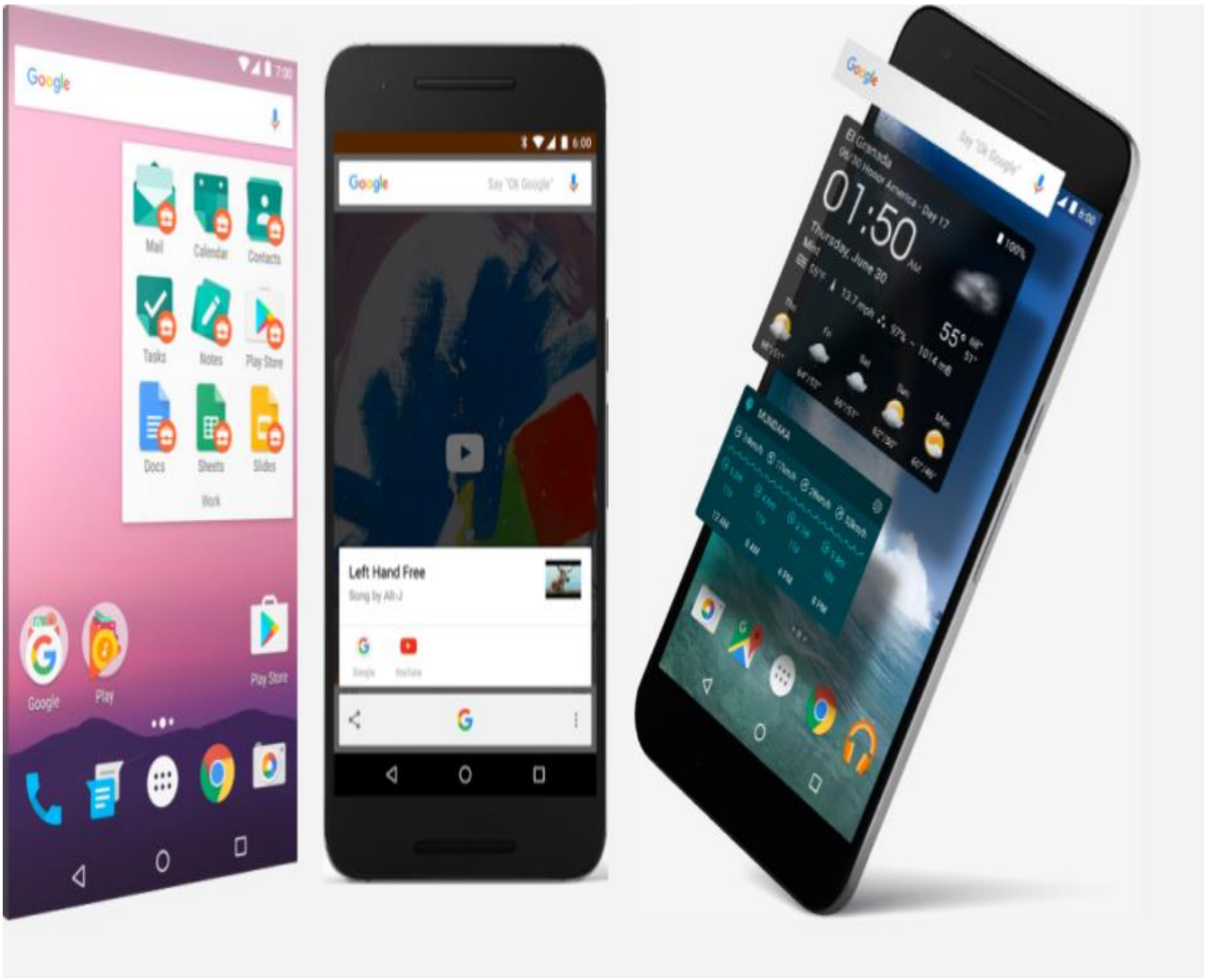




https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## Best experience for app users

Android provides a touchscreen user interface (UI) for interacting with apps. Android's UI is mainly based on direct manipulation. People use touch gestures such as swiping, tapping, and pinching to manipulate on-screen objects. In addition to the keyboard, there's a customizable on-screen keyboard for text input. Android can also support game controllers and full-size physical keyboards connected by Bluetooth or USB.



The Android home screen can contain several panes of *app icons*, which launch their associated apps. Home screen panes can also contain *app widgets*, which display live, auto-updating content such as the weather, the user's email inbox, or a news ticker. Android can also play multimedia content such as music, animation, and video. The figure above shows app icons on the home screen (left), playing music (center), and displaying app widgets (right). Along the top of the screen is a status bar, showing information about the device and its connectivity. The Android home screen may be made up of several panes, and the user swipes back and forth between the panes.

Android is designed to provide immediate response to user input. Besides a dynamic interface that responds immediately to touch, an Android-powered device can vibrate to provide haptic feedback. Many apps take advantage of internal hardware such as accelerometers, gyroscopes, and proximity sensors to respond to additional user actions. These sensors can also detect screen rotation. For example, you could design a racing game where the user rotates the device as if it were a steering wheel.

The Android platform, based on the Linux kernel, is designed primarily for touchscreen mobile devices such as mobile phones and tablets. Because Android-powered devices are usually battery-powered, Android is designed to manage processes to keep power consumption at a minimum, providing longer battery use.

## It's easy to develop apps

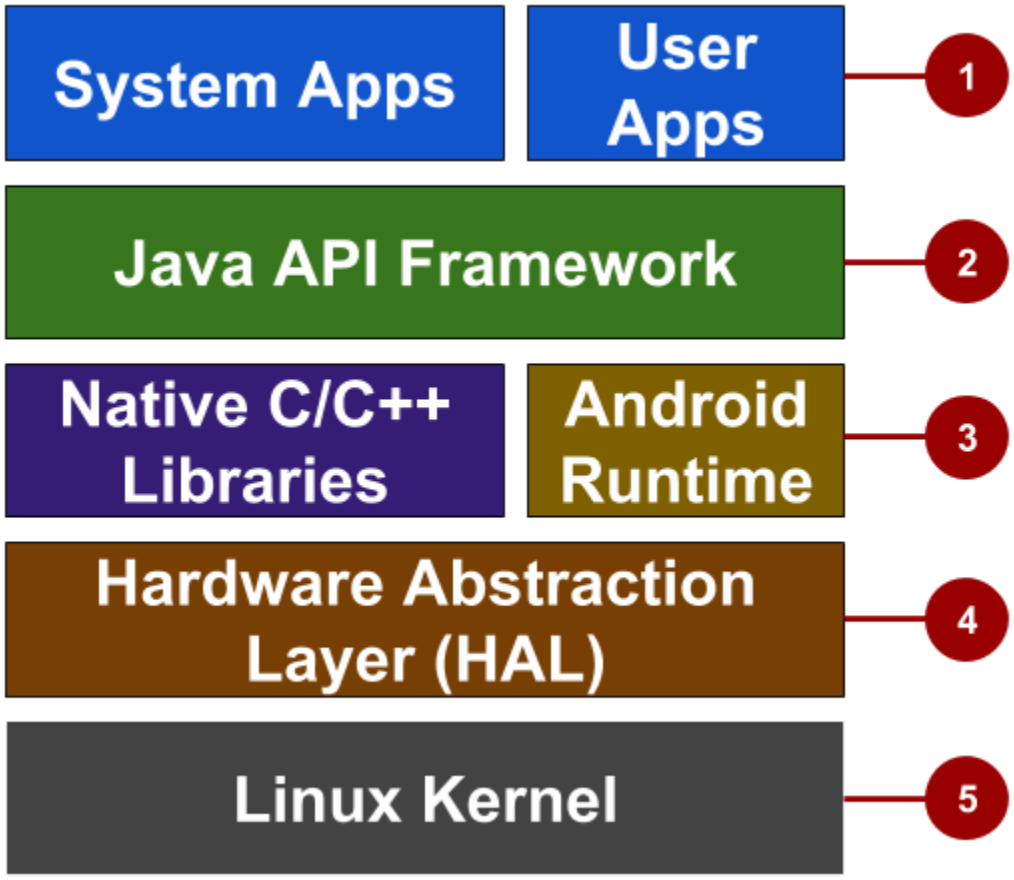
To develop apps that take advantage of the Android operating system and UI, use the Android software development kit (SDK). The SDK includes software libraries of prewritten code, a debugger, a device emulator, documentation, sample code, and tutorials. Use the SDK to create apps that look great and take advantage of the hardware capabilities available on each Android-powered device.

To develop apps using the SDK, you use the Java programming language to develop the app and Extensible Markup Language (XML) files to describe data resources. By writing the code in Java and creating a single app binary, you create an app that can run on both phone and tablet form factors. You can declare your UI in lightweight sets of XML resources. For example, create one set for parts of the UI that are common to all form factors, and other sets for features specific to phones or tablets. At runtime, Android applies the correct resource sets based on the device's screen size, screen density, locale, and so on.

To help you develop your apps efficiently, Google offers an integrated development environment (IDE) called Android Studio. It offers advanced features for developing, debugging, and packaging Android apps. Using Android Studio, you can develop for any Android-powered device, or create virtual devices that emulate any hardware configuration.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

Android provides a rich development architecture. You don't need to know much about the components of this architecture, but it is useful to know what is available in the system for your app to use. The following diagram shows the major components of the Android *stack*—the operating system and development architecture.




In the figure above:

1. *Apps:* Your apps live at this level, along with core system apps for email, SMS messaging, calendars, internet browsing, and contacts.
2. *Java API framework:* All features for Android development, such as UI components, resource management, and lifecycle management, are available through application programming interfaces (APIs). You don't need to know the details of how the APIs work. You only need to learn how to use them.
3. *Libraries and Android runtime:* Each app runs in its own process, with its own instance of the Android runtime. Android includes a set of core runtime libraries that provide most of the functionality of the Java programming language. Many core Android system components and services are built from native code that require native libraries written in C and C++. These native libraries are available to apps through the Java API framework.
4. *Hardware abstraction layer (HAL):* This layer provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module.

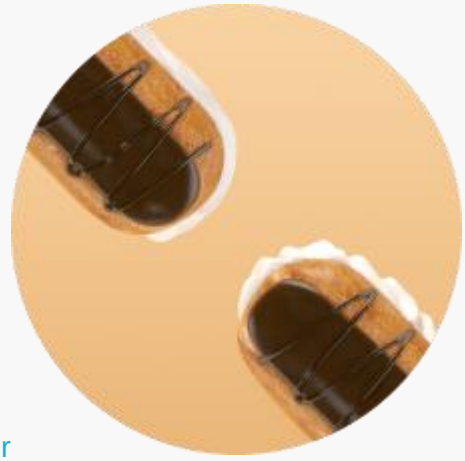
https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

5. *Linux kernel:* The foundation of the Android platform is the Linux kernel. The layers above the Linux kernel rely on the Linux kernel for threading, low-level memory management, and other underlying functionality. Using a Linux kernel enables Android to take advantage of Linux-based security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

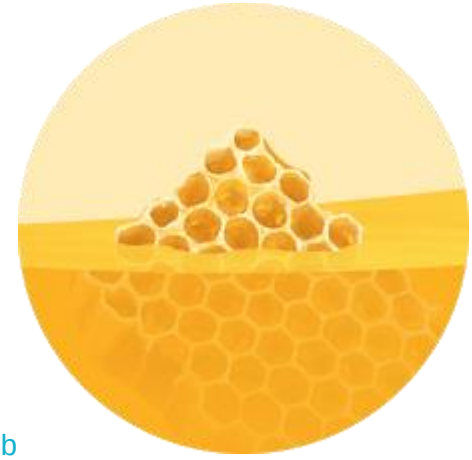
## Many distribution options

You can distribute your Android app in many different ways: email, website, or an app marketplace such as Google Play. Android users download billions of apps and games from the Google Play store each month. Google Play is a digital distribution service, operated and developed by Google, that serves as the official app store for Android. Google Play lets consumers to browse and download apps developed with

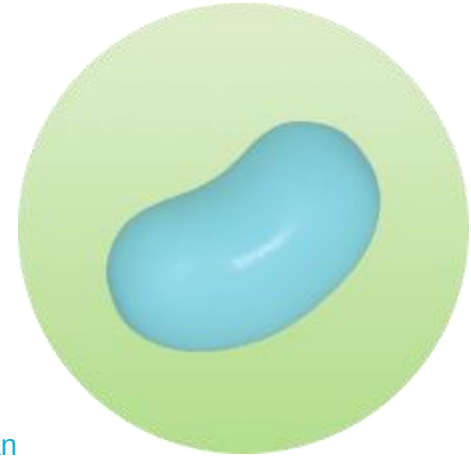

the Android SDK.

| Code name | Version number | Initial release date | API level |
| --- | --- | --- | --- |
| N/A | 1.0 | 23 September 2008 | 1 |
| 1.1 | 9 February 2009 | 2 | |
| Cupcake | 1.5 | 27 April 2009 | 3 |
|  Donut | 1.6 | 15 September 2009 | 4 |
|  Eclair | 2.0 – 2.1 | 26 October 2009 | 5–7 |

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

| | | | |
|---|---|---|---|
|  Froyo | 2.2 – 2.2.3 | 20 May 2010 | 8 |
|  Gingerbread | 2.3 – 2.3.7 | 6 December 2010 | 9–10 |
|  Honeycomb | 3.0 – 3.2.6 | 22 February 2011 | 11–13 |

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

| | | | |
|---|---|---|---|
| Ice Cream Sandwich  | 4.0 – 4.0.4 | 18 October 2011 | 14–15 |
|  Jelly Bean | 4.1 – 4.3.1 | 9 July 2012 | 16–18 |
|  KitKat | 4.4 – 4.4.4 | 31 October 2013 | 19–20 |

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

| | | | |
|---|---|---|---|
|  Lollipop | 5.0 – 5.1.1 | 12 November 2014 | 21–22 |
|  Marshmallow | 6.0 – 6.0.1 | 5 October 2015 | 23 |
|  Nougat | 7.0 | 22 August 2016 | 24 |

| | | | |
|---|---|---|---|
|  Oreo | 8.0 | August 21, 2017 | 26 |

# Android versions

Google provides major incremental upgrades to the Android operating system using confectionery-themed names. The latest major release is Android 8.0 "Oreo".

**Tip**: See previous versions and their features at The Android Story. The dashboard for platform versions shows the distribution of active devices running each version of Android, based on the number of devices that visit the Google Play store. It's a good practice to support about 90% of the active devices, while targeting your app to the latest version.

**Note:** To provide the best features and functionality across Android versions, use the Android Support Library in your app. This library allows your app to use recent Android platform APIs on older devices.

# The challenges of Android app development

While the Android platform provides rich functionality for app development, there are still a number of challenges you need to address, such as:

- Building for a multiscreen world
- Getting performance right
- Keeping your code and your users more secure
- Making sure your app is compatible with older platform versions
- Understanding the market and the user

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Building for a multi-screen world

Android runs on billions of handheld devices around the world and supports various form factors including wearable devices and televisions. Devices come in different sizes and shapes, which affects how you design the screens and UI elements in your apps.

In addition, device manufacturers may add their own UI elements, styles, and colors to differentiate their products. Each manufacturer offers different features with respect to keyboard forms, screen size, or camera buttons. An app running on one device may look a bit different on another. Your challenge, as a developer, is to design UI elements that work on all devices.

# Maximizing app performance

An app's *performance* is determined by how fast it runs, how easily it connects to the network, and how well it manages battery and memory usage. Performance is affected by factors such as battery life, multimedia content, and internet access. Be aware that some features you design for your app may cause performance problems for users. For example, to save the user's battery power, enable background services only when they are necessary.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## Keeping your code and your users more secure

You need to take precautions to make your code, and the user's experience when they use your app, as secure as possible.

- Use tools such as ProGuard, which is provided in Android Studio. ProGuard detects and removes unused classes, fields, methods, and attributes.
- Encrypt all of your app's code and resources while packaging the app.
- To protect critical user information such as logins and passwords, secure your communication channel to protect data in transit across the internet, as well as data at rest on the device.

## Remaining compatible with older versions of Android

The Android platform continues to improve and provide new features you can add to your apps. However, you should ensure that your app can still run on devices with older versions of Android. It is impractical to focus only on the most recent Android version, as not all users may have upgraded or may be able to upgrade their devices. Fortunately Android Studio provides options for developers to more easily remain compatible with older versions.

# Learn more

Introductory Android developer documentation:

- Developer Guides
- Platform Architecture
- Layouts
- Supporting different platform versions

Other:

- Distribution dashboard
- Meet Android Studio
- Wikipedia: Android version history

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# 1.1: Your first Android app

**Contents:**

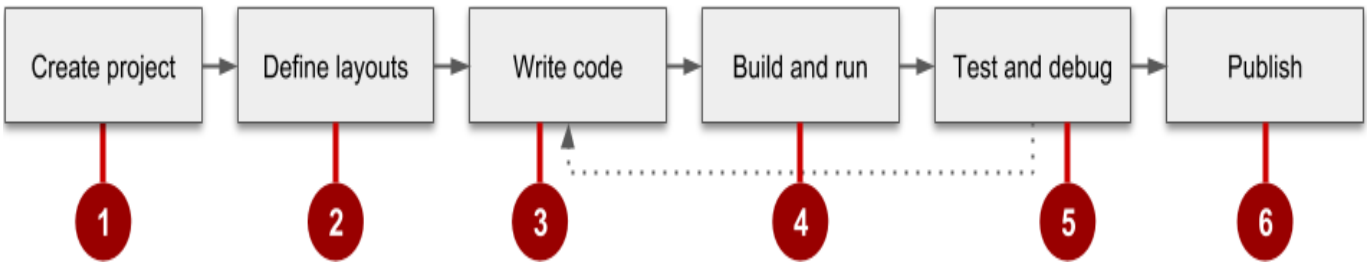


This chapter describes how to develop applications using Android Studio, which is an integrated development environment (IDE) for Android.

## The development process

An Android app project begins with an idea and a definition of the requirements necessary to realize that idea. You may want to sketch user interfaces (UIs) for the various app functions. To show what a UI would look like and how it would work, use drawings, mockups, and prototypes.

When you are ready to start coding, you use Android Studio to go through the following steps:




1. Create the project in Android Studio and choose an appropriate template.
2. Define a layout for each screen that has UI elements. You can place UI elements on the screen using the layout editor, or you can write code directly in the Extensible Markup Language (XML).
3. Write code using the Java programming language. Create source code for all of the app's components.
4. Build and run the app on real and virtual devices. Use the default build configuration or create custom builds for different versions of your app.©
5. Test and debug the app's logic and UI.
6. Publish the app by assembling the final APK (package file) and distributing it through channels such as Google Play.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Using Android Studio

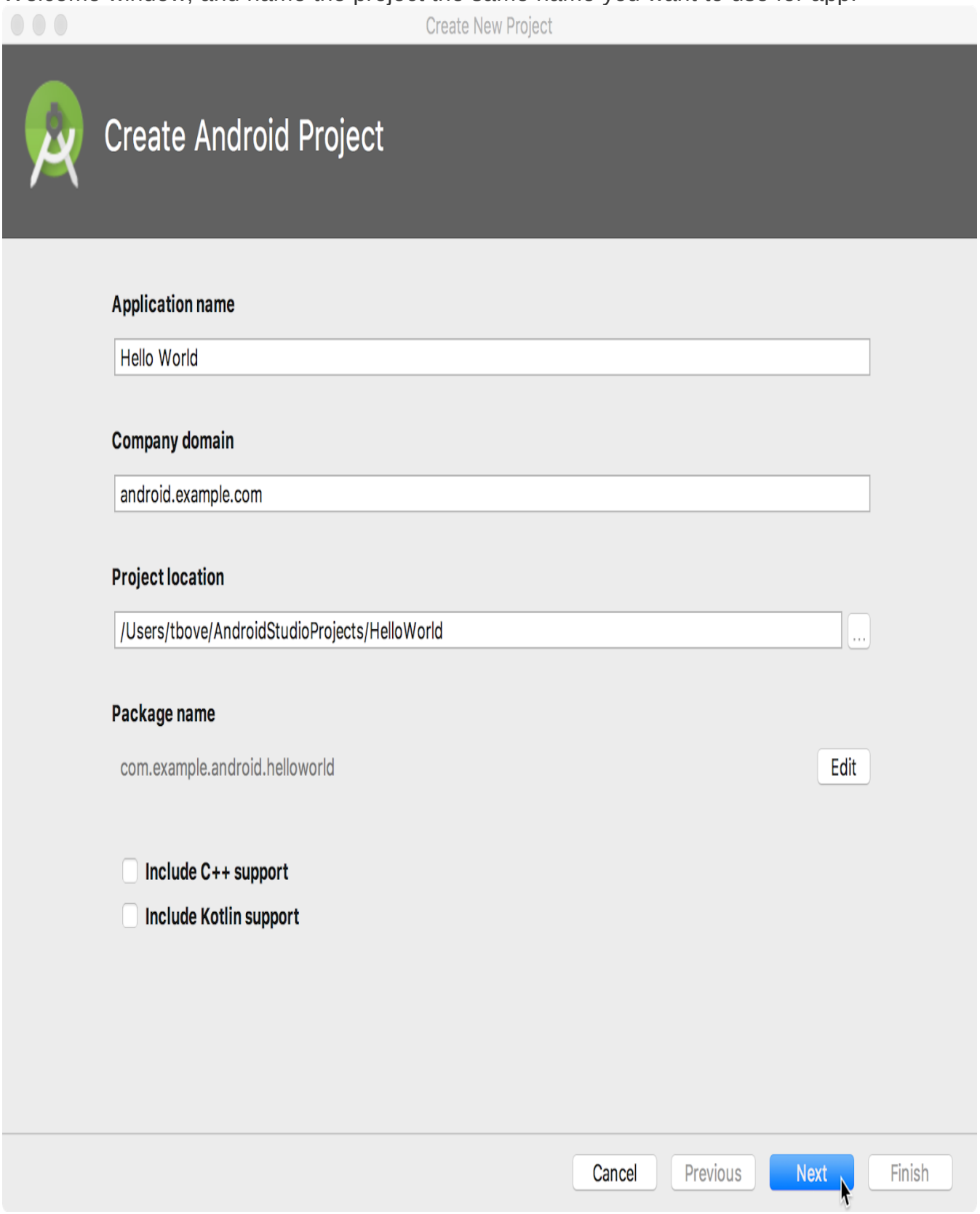
Android Studio provides a unified development environment for creating apps for all Android-powered devices. Android Studio includes code templates with sample code for common app features, extensive testing tools and frameworks, and a flexible build system.

## Starting an Android Studio project

After you have successfully installed the Android Studio IDE, double-click the Android Studio application icon to start it. Click **Start a new Android Studio project** in the

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

Welcome window, and name the project the same name you want to use for app.



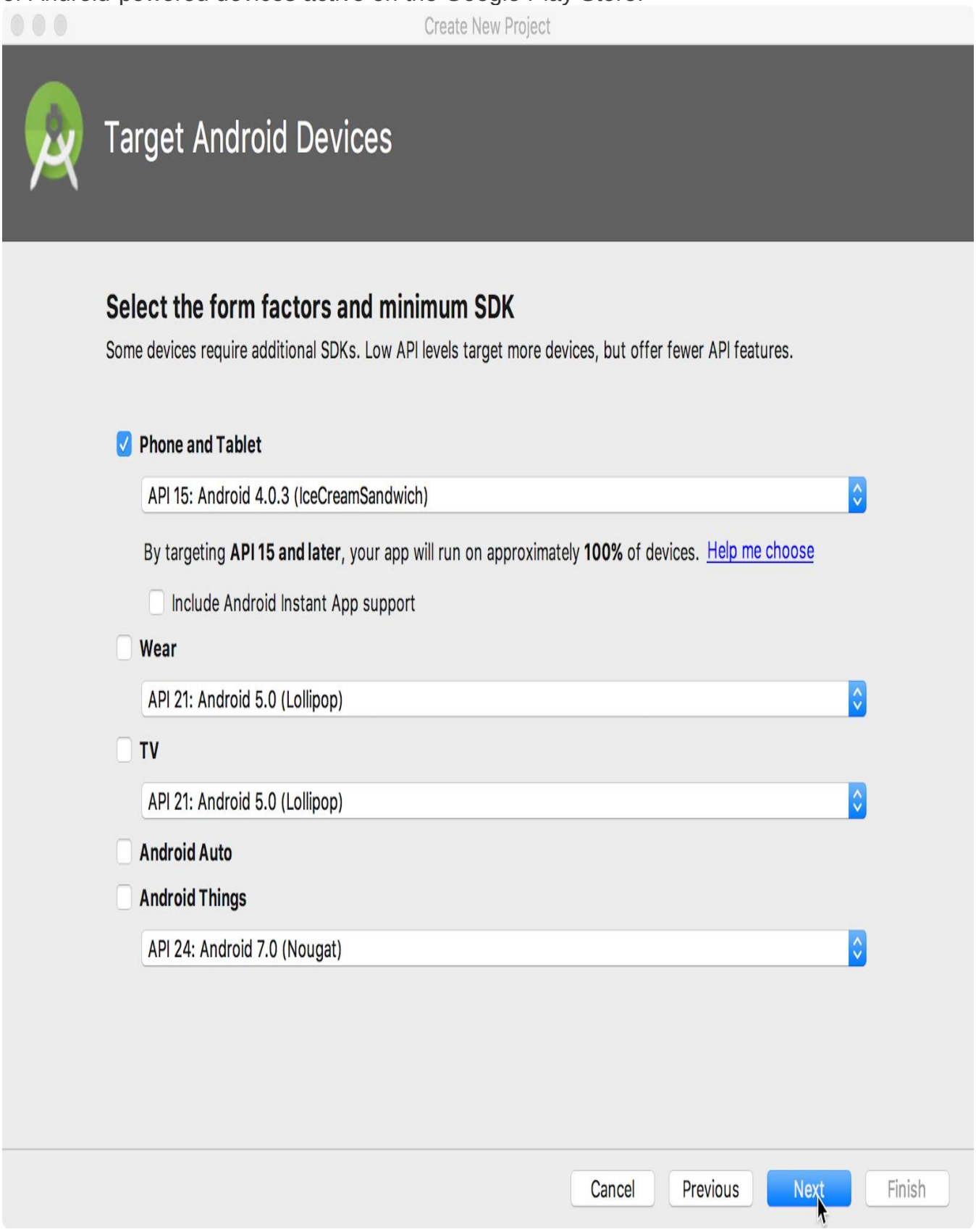
https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

When choosing a unique **Company domain**, keep in mind that apps published to Google Play must have a unique package name. Because domains are unique, prepending the app's name with your name, or your company's domain name, should provide an adequately unique package name. If you don't plan to publish the app, you can accept the default example domain. Be aware that changing the package name later is extra work.

## Choosing target devices and the minimum SDK

When choosing Target Android Devices, **Phone and Tablet** are selected by default, as shown in the figure below. The choice shown in the figure for the Minimum SDK—**API 15: Android 4.0.3 (IceCreamSandwich)**—makes your app compatible with 97%

of Android-powered devices active on the Google Play Store.



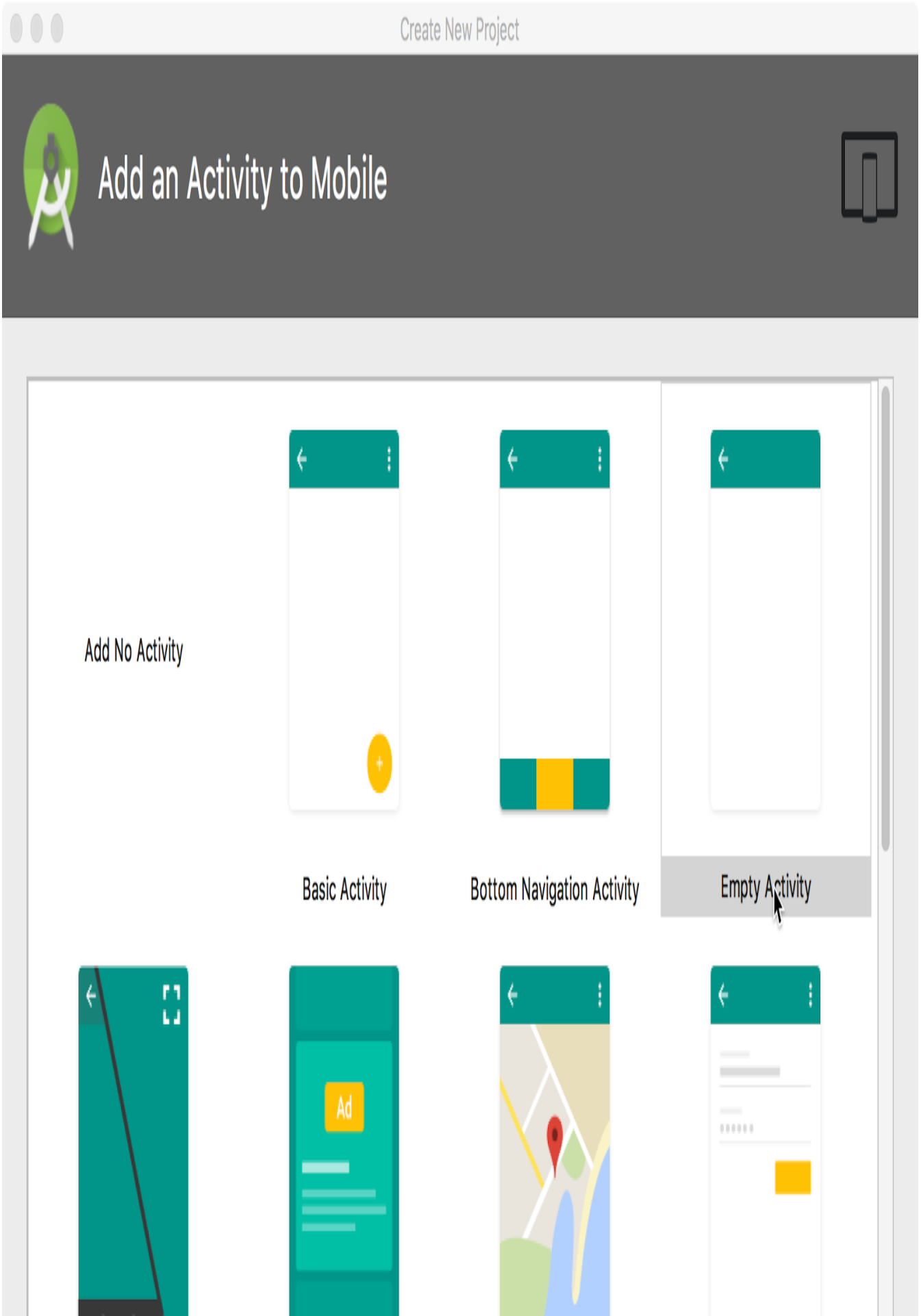
Different devices run different versions of the Android system, such as Android 4.0.3 or Android 4.4. Each successive version often adds new APIs not available in the previous version. To indicate which set of APIs are available, each version specifies an API level. For instance, Android 1.0 is API level 1 and Android 4.0.3 is API level 15.

The Minimum SDK declares the minimum Android version for your app. Each successive version of Android provides compatibility for apps that were built using the APIs from previous versions. That means your app should *always* be compatible with future versions of Android, if you use the documented Android APIs.

## Choosing an Activity template

An `Activity` is a single, focused thing that the user can do. It is a crucial component of any Android app. An `Activity` typically has a layout associated with it that defines how UI elements appear on a screen.
Android Studio pre-populates your project with minimal code for an `Activity` and layout based on a *template*. Available `Activity` templates range from a virtually blank template (**Add No Activity**) to an `Activity` that includes navigation and an options menu.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

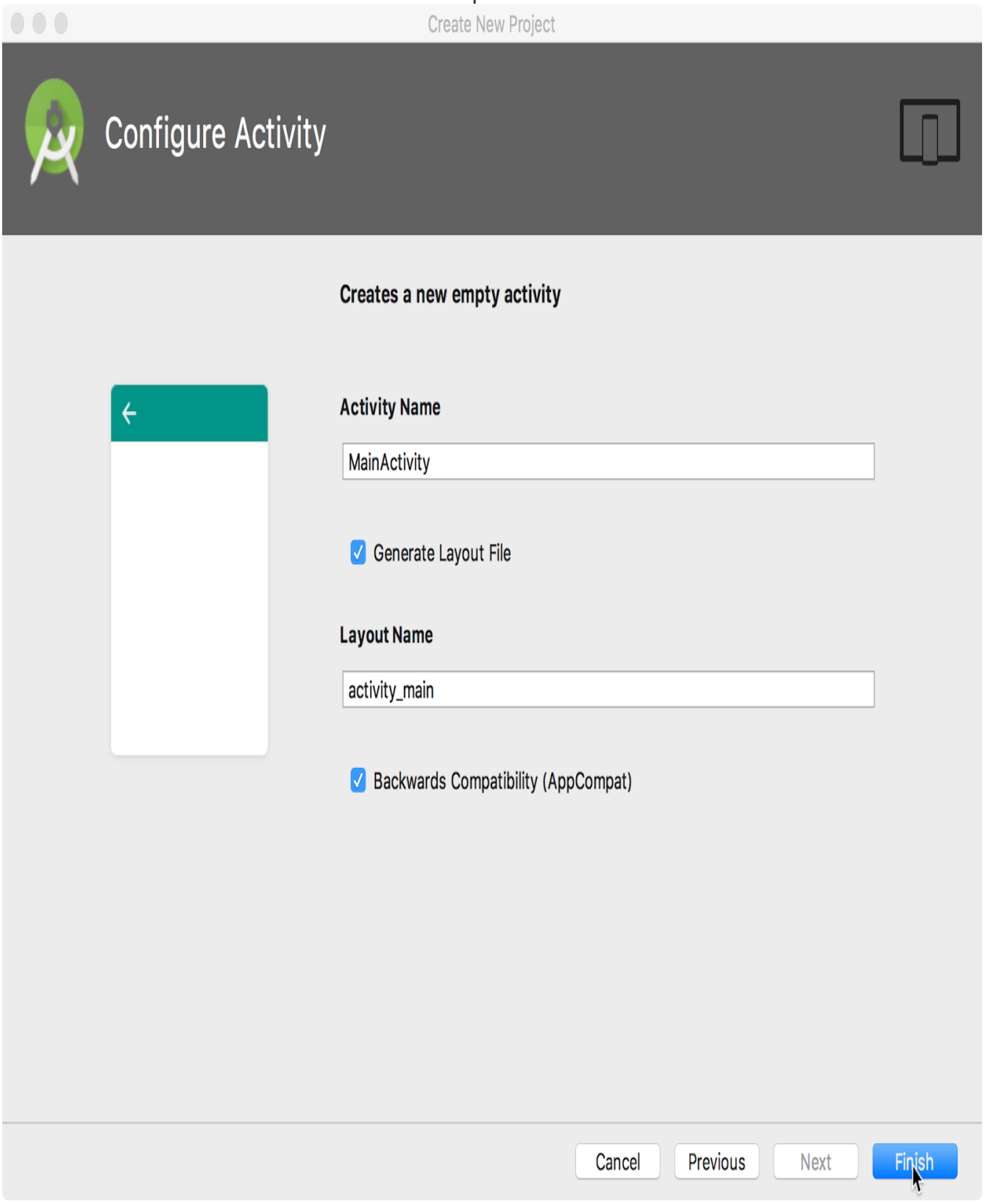


https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

You can customize the `Activity` after you select your template. For example, the **Empty Activity** choice provides a single `Activity` with a single layout resource for the screen. The **Configure Activity** screen appears after you click **Next**. On the **Configure Activity** screen you can accept the commonly used name for the `Activity` (such as `MainActivity`), or you can change the name.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

**Tip**: This course covers the `Activity` class in more detail in another practical. You can also read Introduction to Activities for a comprehensive introduction.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

The **Configure Activity** screen differs depending on which template you chose. In most cases you can select the following options, if they are not already selected:

- **Generate Layout file**: Leave this checkbox selected to create the layout resource connected to this `Activity`, which is usually named `activity_main`. The layout defines the UI for the `Activity`.
- **Backwards Compatibility (AppCompat):** Leave this checkbox selected to include the `AppCompat` library. Use the `AppCompat` library to make sure that the app is compatible with previous versions of Android, even if the app uses features found only in newer Android versions.

Android Studio creates a folder for your projects, and builds the project with Gradle.

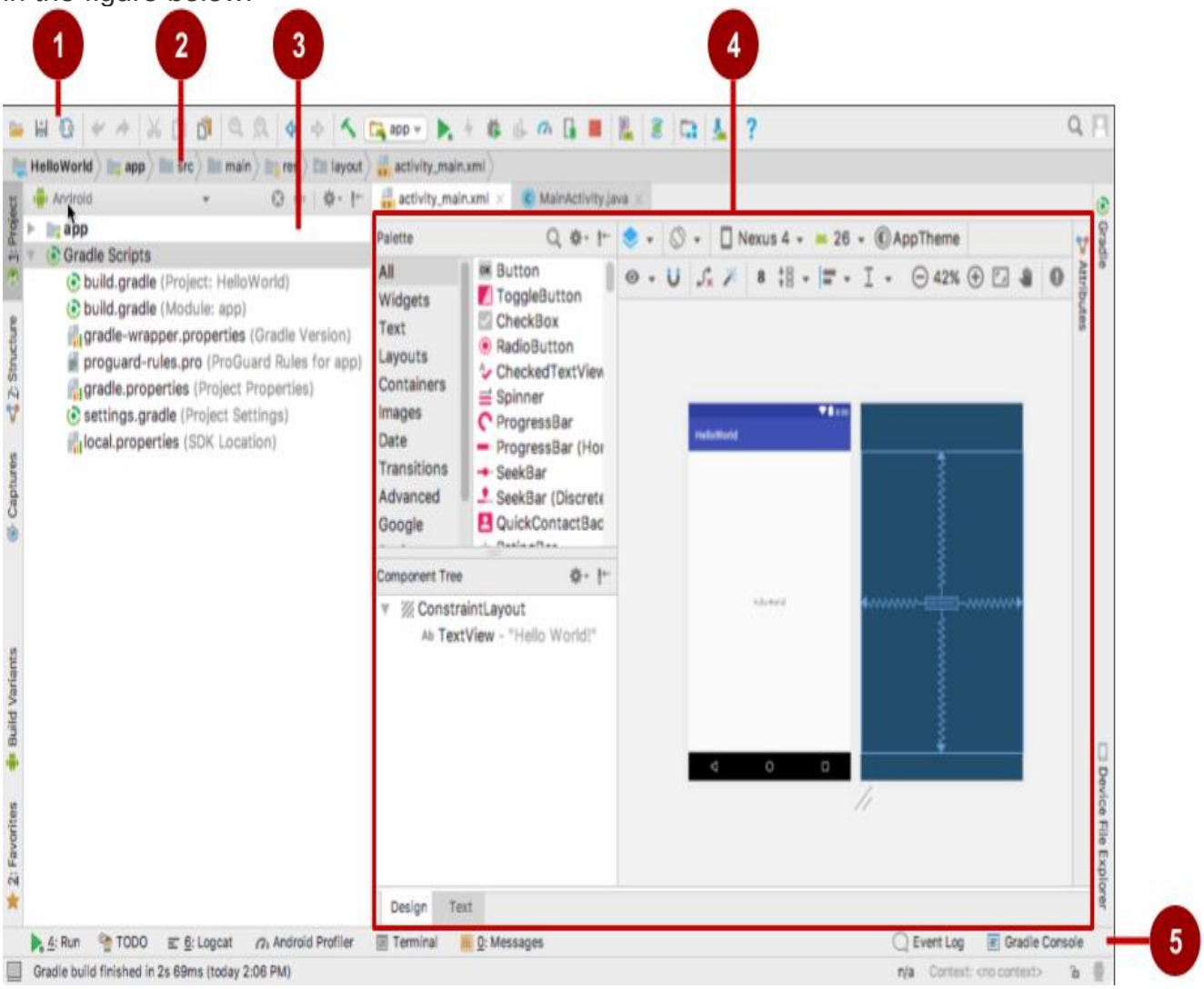
**Tip**: See the Configure your build developer page for detailed information.

# Exploring a project

An Android Studio project contains all of the source code and all resources for an app. The resources include layouts, strings, colors, dimensions, and images. The Android Studio main window is made up of several logical areas, or *panes*, as shown

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

in the figure below.




In the figure above:

1. Toolbar: Provides a wide range of actions, including running the Android app and launching Android tools.
2. Navigation bar: Navigate through the project and open files for editing.
3. **Project** pane: Displays project files in a hierarchy. The selected hierarchy in the figure above is **Android**.
4. Editor: The contents of a selected file in the project. For example, after you select a layout (as shown in the figure above), the editor pane shows the layout editor with tools to edit the layout. After you select a Java code file, the editor pane shows the Java code with tools for editing the code.
5. Tabs along the left, right, and bottom of the window: You can click tabs to open other panes, such as **Logcat** to open the **Logcat** pane with log messages, or **TODO** to manage tasks.

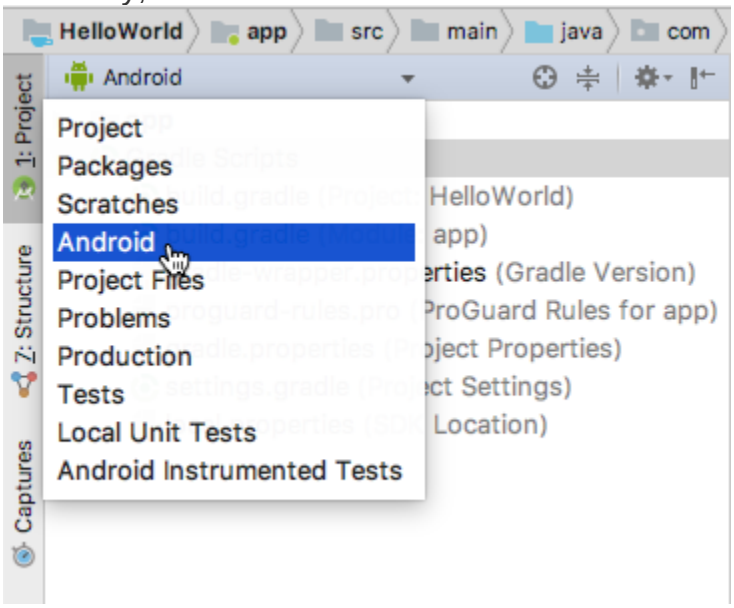
The status bar at the bottom of the Android Studio window displays the status of the project and Android Studio itself, as well as any warnings or messages. You can watch the build progress in the status bar.

**Tip**: You can organize the main window to give yourself more screen space by hiding or moving panes. You can also use keyboard shortcuts to access most features. See Keyboard Shortcuts for a complete list.

## Using the Project pane

You can view the project organization in several ways in the Project pane. If it is not already selected, click the **Project** tab. (The **Project** tab is in the vertical tab column on the left side of the Android Studio window.)

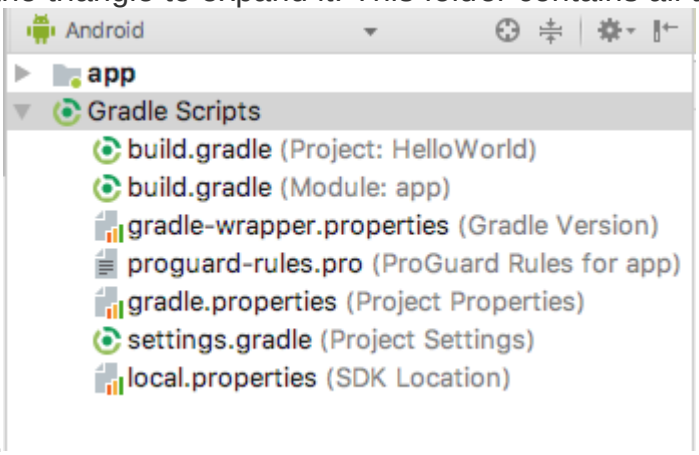
The Project pane appears. To view the project in the standard Android project hierarchy, select **Android** from the Down arrow at the top of the **Project** pane.




**Note:** This chapter and other chapters refer to the Project pane, when set to **Android**, as the **Project > Android** pane.

## Gradle files

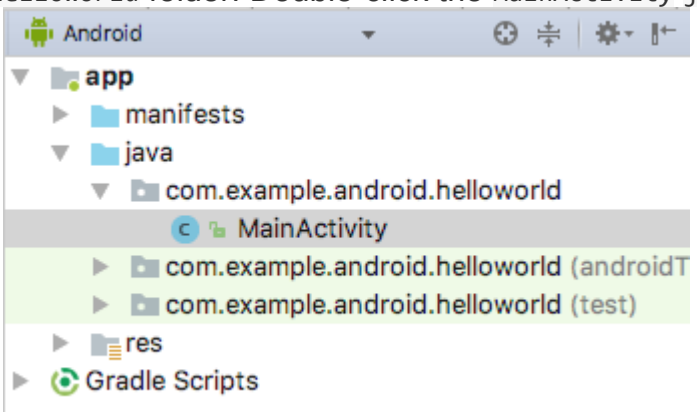
When you first create an app project, the **Project > Android** pane appears with the `Gradle Scripts` folder expanded as shown below. If the `Gradle Scripts` folder is not expanded, click the triangle to expand it. This folder contains all the files needed by the build system.




The `build.gradle(Module:app)` file specifies additional libraries and the module's build configuration. The `Activity` template that you select creates this file. The file includes the `minSdkVersion` attribute that declares the minimum version for the app, and the `targetSdkVersion` attribute that declares the highest (newest) version for which the app has been optimized.
This file also includes a list of *dependencies*, which are libraries required by the code—such as the `AppCompat` library for supporting a wide range of Android versions.

## App code

To view and edit the Java code, expand the `app` folder, the `java` folder, and the `com.example.android.helloworld` folder. Double-click the `MainActivity` `java` file to open it in the code editor.

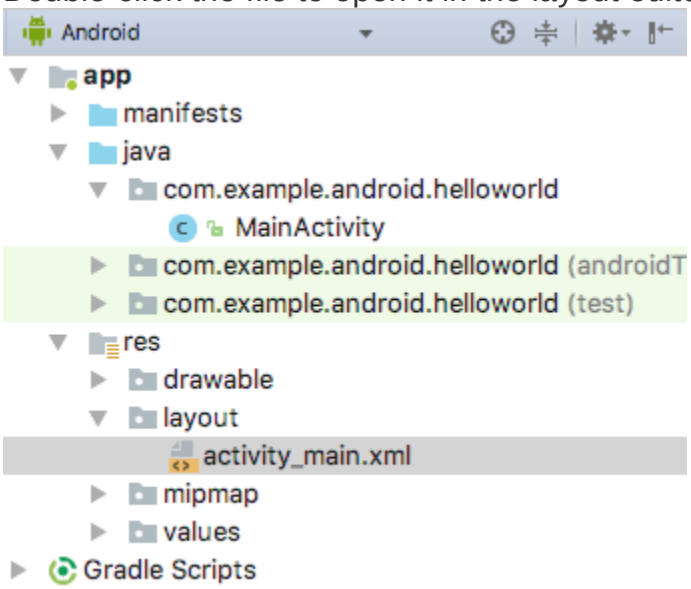



The `java` folder includes Java class files. Each `Activity`, `Service`, or other component (such as a `Fragment`) is defined as a Java class, usually in its own file. Tests and other Java class files are also located here.
The `java` folder contains three subfolders:

- `com.example.hello.helloworld` (or the domain name you have specified)**:** All the files for a package are in a folder named after the package. For your Hello World app, there is one package, and it contains only `MainActivity.java`. The first `Activity` (screen) that the user sees, which also initializes app-wide resources, is customarily called `MainActivity`. (The file extension is omitted in the **Project > Android** pane.)
- `com.example.hello.helloworld(androidTest)`: This folder is for your instrumented tests, and starts out with a skeleton test file.
- `com.example.hello.helloworld(test)`: This folder is for your unit tests and starts out with an automatically created skeleton unit test file.

## *Layout files*

To view and edit a layout file, expand the `res` folder and the `layout` folder to see the layout file. In the figure below, the layout file is called `activity_main.xml`. Double-click the file to open it in the layout editor. Layout files are written in XML.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## *Resource files*

The `res` folder holds resources, such as layouts, strings, and images. An `Activity` is usually associated with a layout of UI views that are defined as an XML file. This XML file is usually named after its `Activity`. The `res` folder includes these subfolders:

- `drawable`: Store all your app's images in this folder.
- `layout`: Every `Activity` has at least one XML layout file that describes the UI. For Hello World, this folder contains `activity_main.xml`.
- `mipmap`: The launcher icons are stored in this folder. There is a subfolder for each supported screen density. Android uses the screen density (the number of pixels per inch) to determine the required image resolution. Android groups all actual screen densities into generalized densities, such as medium (mdpi), high (hdpi), or extra-extra-extra-high (xxxhdpi). The `ic_launcher.png` folder contains the default launcher icons for all the densities supported by your app.
- `values`: Instead of hardcoding values like strings, dimensions, and colors in your XML and Java files, it is best practice to define them in their respective `values` files. This practice makes it easier to change the values and keep the values consistent across your app.

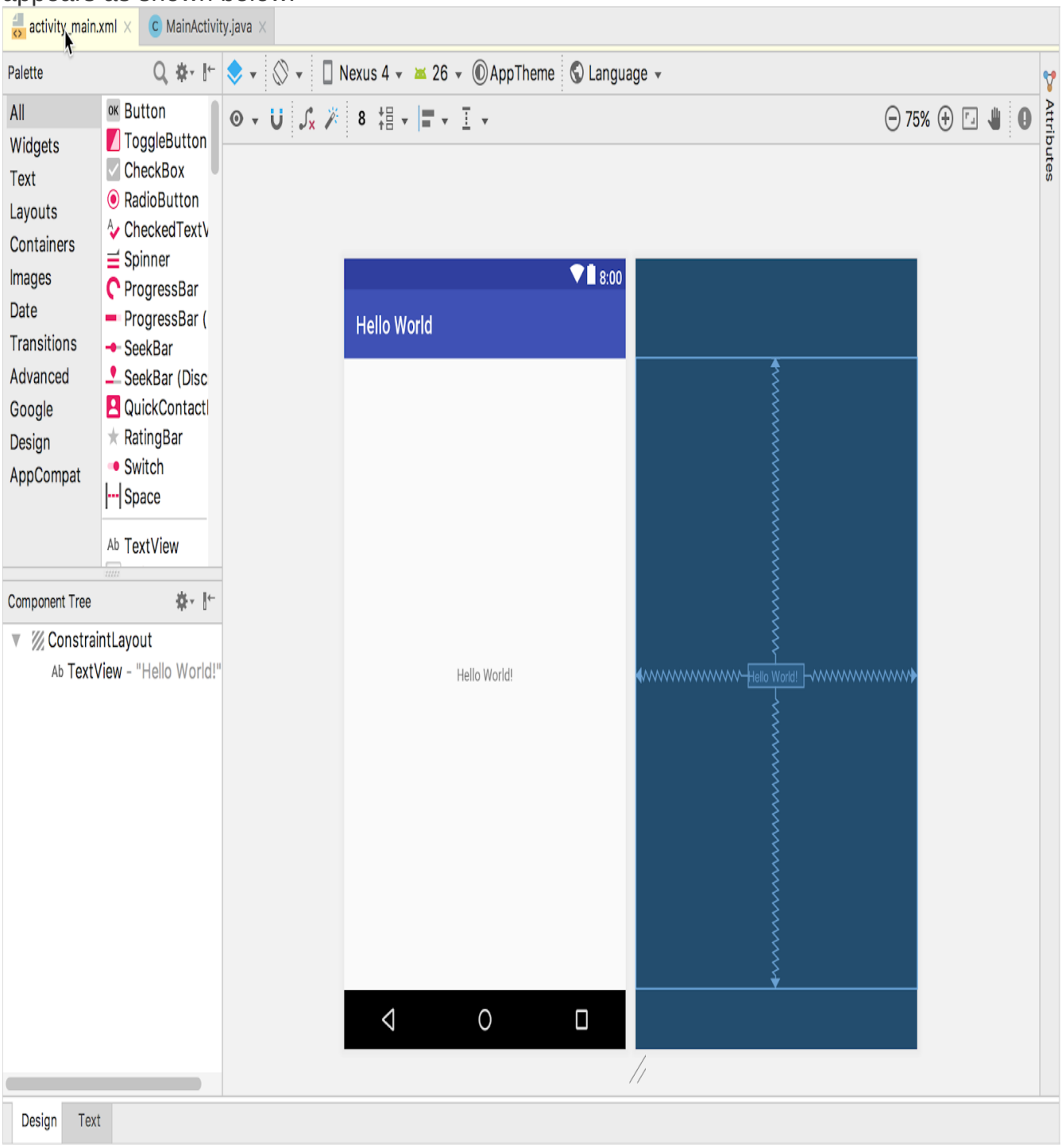
The `values` subfolder includes these subfolders:

- `colors.xml`: Shows the default colors for your chosen theme. You can add your own colors or change the colors based on your app's requirements.
- `dimens.xml`: Store the sizes of views and objects for different resolutions.
- `strings.xml`: Create resources for all your strings. Doing this makes it easy to translate the strings to other languages.
- `styles.xml`**:** All the styles for your app and theme go here. Styles help give your app a consistent look for all UI elements.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

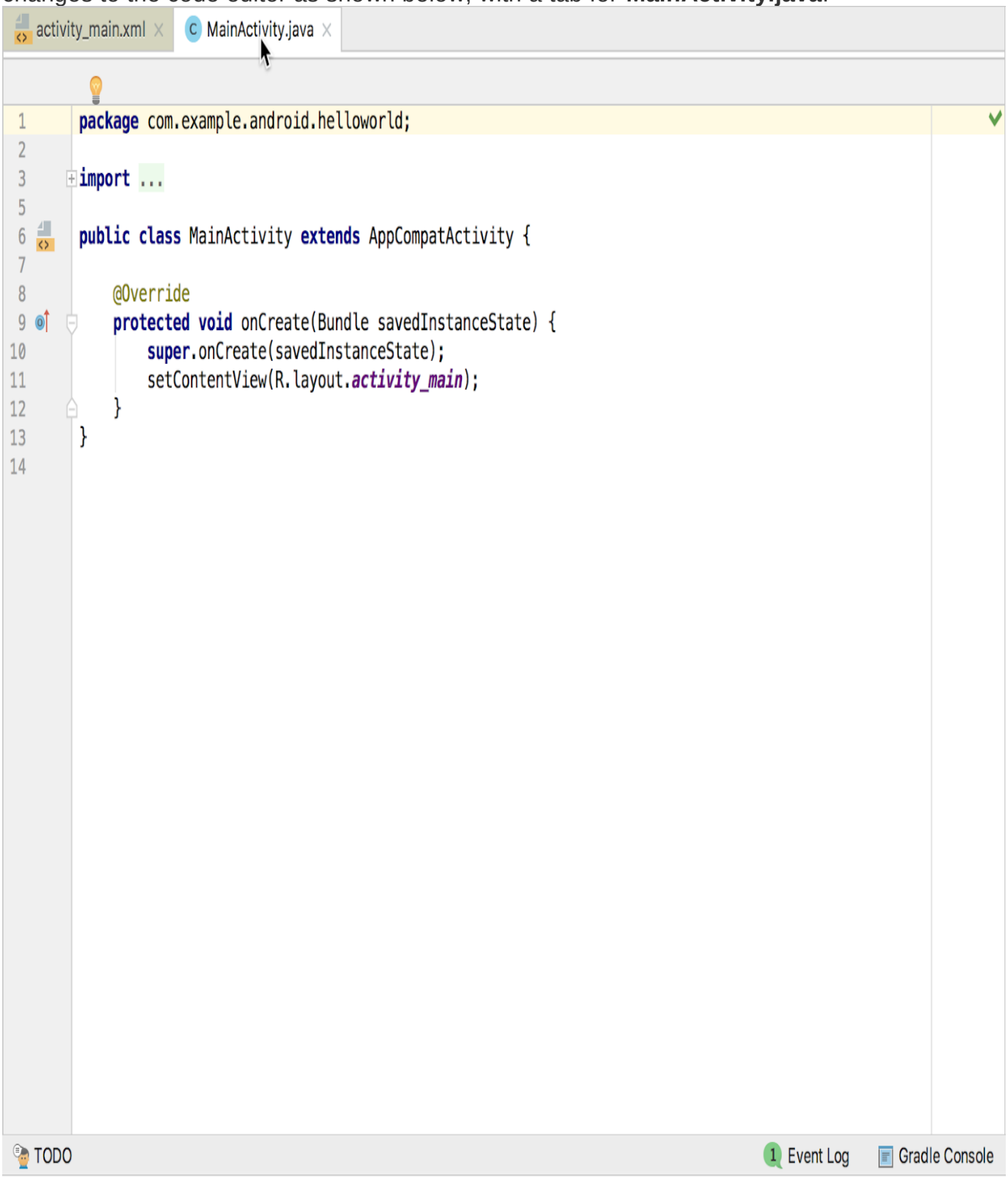
# Using the editor pane

If you select a file, the editor pane appears. A tab appears for the file so that you can open multiple files and switch between them. For example, if you double-click the **activity_main.xml** layout file in the **Project > Android** pane, the layout editor appears as shown below.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

If you double-click the **MainActivity** file in the **Project > Android** pane, the editor changes to the code editor as shown below, with a tab for **MainActivity.java**:



At the top of the `MainActivity.java` file is a `package` statement that defines the app package. This package statement is followed by an `import` block condensed with `...`, as shown in the figure above. Click the dots to expand the block to view it.
The `import` statements import libraries needed for the app. For example, the following statement imports the `AppCompatActivity` library:

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

Each `Activity` in an app is implemented as a Java class. The following class declaration extends the `AppCompatActivity` class to implement features in a way that is backward-compatible with previous versions of Android:

```
public class MainActivity extends AppCompatActivity {
    // ... Rest of the code for the class.
}
```

# Understanding the Android manifest

Before the Android system can start an app component such as an `Activity`, the system must know that the `Activity` exists. It does so by reading the app's `AndroidManifest.xml` file, which describes all of the components of your Android app. Each `Activity` must be listed in this XML file, along with all components for the app.
To view and edit the `AndroidManifest.xml` file, expand the `manifests` folder in the **Project > Android** pane, and double-click `AndroidManifest.xml`. Its contents appear in the editing pane:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        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>
```

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Android namespace and application tag

The Android Manifest is coded in XML and always uses the Android namespace:

```
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.helloworld">
```

The `package` expression shows the unique package name of the new app. Do not change the package expression after the app is published.
The `<application>` tag, with its closing `</application>` tag, defines the manifest settings for the entire app.

# Automatic backup

The `android:allowBackup` attribute enables automatic app data backup:

```
android:allowBackup="true"
```

Setting the `android:allowBackup` attribute to `true` enables the app to be backed up automatically and restored as needed. Users invest time and effort to configure apps. Switching to a new device can cancel out all that careful configuration. The system performs this automatic backup for nearly all app data by default, and does so without the developer having to write any additional app code.
For apps whose target SDK version is Android 6.0 (API level 23) and higher, devices running Android 6.0 and higher automatically create backups of app data to the cloud because the `android:allowBackup` attribute defaults to `true` if omitted. For apps < API level 22 you have to explicitly add the `android:allowBackup` attribute and set it to `true`.
**Tip**: To learn more about the automatic backup for apps, see Configuring Auto Backup for Apps.

# The app icon

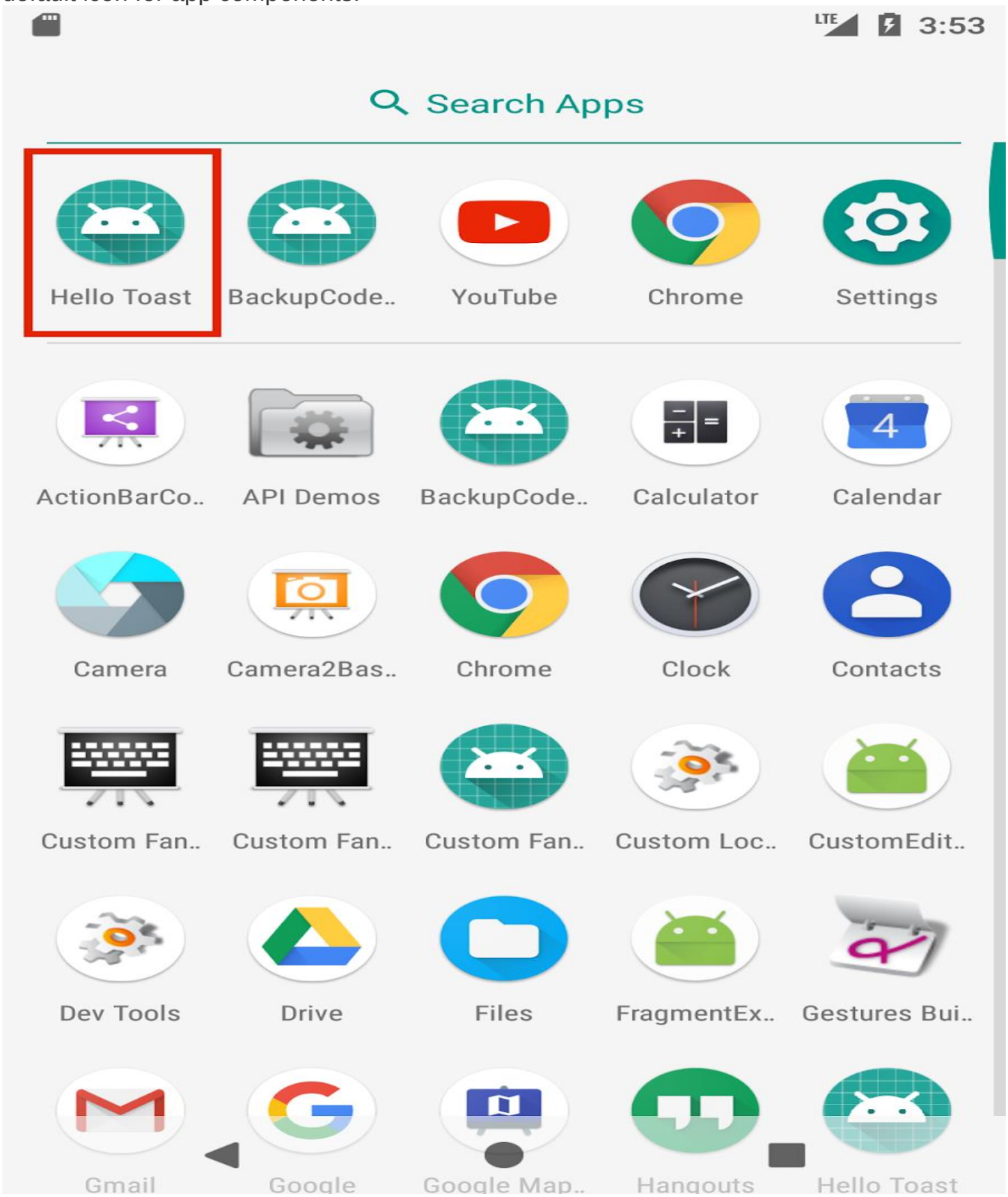
The `android:icon` attribute sets the icon for the app:

```
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
```

The `android:icon` attribute assigns to the app an icon in the `mipmap` folder (inside the `res` folder in the **Project > Android** pane). The icon appears on the home screen or in the Search Apps screen for launching the app. The icon is also used as the

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

default icon for app components.



## App label and string resources

The `android:label` attribute shows the string `"Hello World"` highlighted. If you click the string, it changes to show the string resource `@string/app_name`:

```
android:label="@string/app_name"
```

**Tip**: To see the context menu, ctrl-click or right-click `app_name` in the editor pane. Select **Go To > Declaration** to see where the string resource is declared: in the `strings.xml` file. When you select **Go To > Declaration** or open the file by double-clicking `strings.xml` inside the `values` folder in the **Project > Android** pane, the file's contents appear in the editor pane.

After opening the `strings.xml` file, you can see that the string name `app_name` is set to `Hello World`. You can change the app name by changing the `Hello World` string to something else. String resources are described in a separate lesson.

## App theme

The `android:theme` attribute sets the app's theme, which defines the appearance of UI elements such as text:

```
android:theme="@style/AppTheme">
```

The `theme` attribute is set to the standard theme `AppTheme`. Themes are described in a separate lesson.

## Declaring the Android version

Different devices may run different versions of the Android system, such as Android 4.0 or Android 4.4. Each successive version can add new APIs not available in the previous version. To indicate which set of APIs are available, each version specifies an API level. For instance, Android 1.0 is API level 1 and Android 4.4 is API level 19.

The API level allows a developer to declare the minimum version with which the app is compatible, using the `<uses-sdk>` manifest tag and its `minSdkVersion` attribute. For example, the Calendar Provider APIs were added in Android 4.0 (API level 14). If your app can't function without these APIs, declare API level 14 as the app's minimum supported version like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.helloworld">
    <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="19" />
    // ... Rest of manifest information
</manifest>
```

The `minSdkVersion` attribute declares the minimum version for the app, and the `targetSdkVersion` attribute declares the highest (newest) version which has been optimized within the app. Each successive version of Android provides compatibility for apps that were built using the APIs from previous versions, so the app should *always* be compatible with future versions of Android while using the documented Android APIs.

The `targetSdkVersion` attribute does *not* prevent an app from being installed on Android versions that are higher (newer) than the specified value. Even so, the target attribute is important, because it indicates to the system whether the app should inherit behavior changes in newer versions.
If you don't update the `targetSdkVersion` to the latest version, the system assumes that your app requires backward-compatible behaviors when it runs on the latest version. For example, among the behavior changes in Android 4.4, alarms created with the `AlarmManager` APIs are now inexact by default so that the system can batch app alarms and preserve system power. If your target API level is lower than `"19"`, the system retains the previous API's behavior for your app.

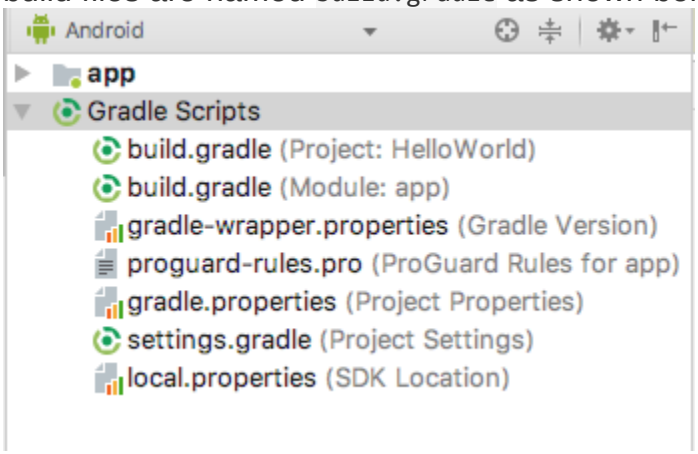
# Understanding the build process

The Android application package (APK) is the package file format for distributing and installing Android mobile apps. The build process involves tools and processes that automatically convert each project into an APK.

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android Plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu.

## Understanding build.gradle files

When you create a project, Android Studio automatically generates the necessary build files in the `Gradle Scripts` folder in the **Project > Android** pane. Android Studio build files are named `build.gradle` as shown below:




## build.gradle (Project: *apptitle*)

This file is the top-level build file for the entire project, located in the root project folder, which defines build configurations that apply to all modules in your project. This file, generated by Android Studio, should not be edited to include app dependencies.

If a dependency is something other than a local library or file tree, Gradle looks for the files in whichever online repositories are specified in the repositories block of this file. By default, new Android Studio projects declare JCenter and Google (which includes the Google Maven repository) as the repository locations:

```
allprojects {
    repositories {
        google()
        jcenter()
    }
}
```

## build.gradle (Module: app)

Android Studio creates separate `build.gradle (Module: app)` files for each module. You can edit the build settings to provide custom packaging options for each module, such as additional build types and product flavors, and to override settings in the manifest or top-level build.gradle file. This file is most often the file to edit when changing app-level configurations, such as declaring dependencies in the `dependencies` section. The following shows the contents of a project's `build.gradle (Module: app)` file:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.example.android.helloworld"
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
                        "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                                                    'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation
                    'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

The `build.gradle` files use Gradle syntax. Gradle is a Domain Specific Language (DSL) for describing and manipulating the build logic using Groovy, which is a dynamic language for the Java Virtual Machine (JVM). You don't need to learn Groovy to make changes, because the Android Plugin for Gradle introduces most of the DSL elements you need.
**Tip**: To learn more about the Android plugin DSL, read the DSL reference documentation.

## *Plugin and Android blocks*

In the `build.gradle (Module: app)` file above, the first statement applies the Android-specific Gradle plug-in build tasks:

```
apply plugin: 'com.android.application'

android {
   compileSdkVersion 26
   // ... Rest of android block.
}
```

The `android { }` block specifies the target SDK version for compiling the app code ( `compileSdkVersion 26`) and several blocks of information.

## *The defaultConfig block*

Core settings and entries for the app are specified in the `defaultConfig { }` block within the `android { } block:`

```
defaultConfig {
    applicationId "com.example.android.helloworld"
    minSdkVersion 15
    targetSdkVersion 26
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner
                "android.support.test.runner.AndroidJUnitRunner"
}
```

The `minSdkVersion` and `targetSdkVersion` settings override any `AndroidManifest.xml` settings for the minimum SDK version and the target SDK version. See "Declaring the Android version" previously in this chapter for background information on these settings.
The `testInstrumentationRunner` statement adds the instrumentation support for testing the UI using Espresso and UIAutomator. These tools are described in a separate lesson.

### *Build types*

Build types for the app are specified in a `buildTypes { }` block, which controls how the app is built and packaged.

```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'),
                                                        'proguard-rules.pro'
    }
}
```

The build type specified is `release` for the app's release. Another common build type is `debug`. Configuring build types is described in a separate lesson.

### *Dependencies*

Dependencies for the app are defined in the `dependencies { }` block, which is the part of the `build.gradle` file that is most likely to change as you start developing code that depends on other libraries. The block is part of the standard Gradle API and belongs *outside* the `android { }` block.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation
                    'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

In the snippet above, the statement `implementation fileTree(dir: 'libs', include: ['*.jar'])` adds a dependency of all ".jar" files inside the `libs` folder.

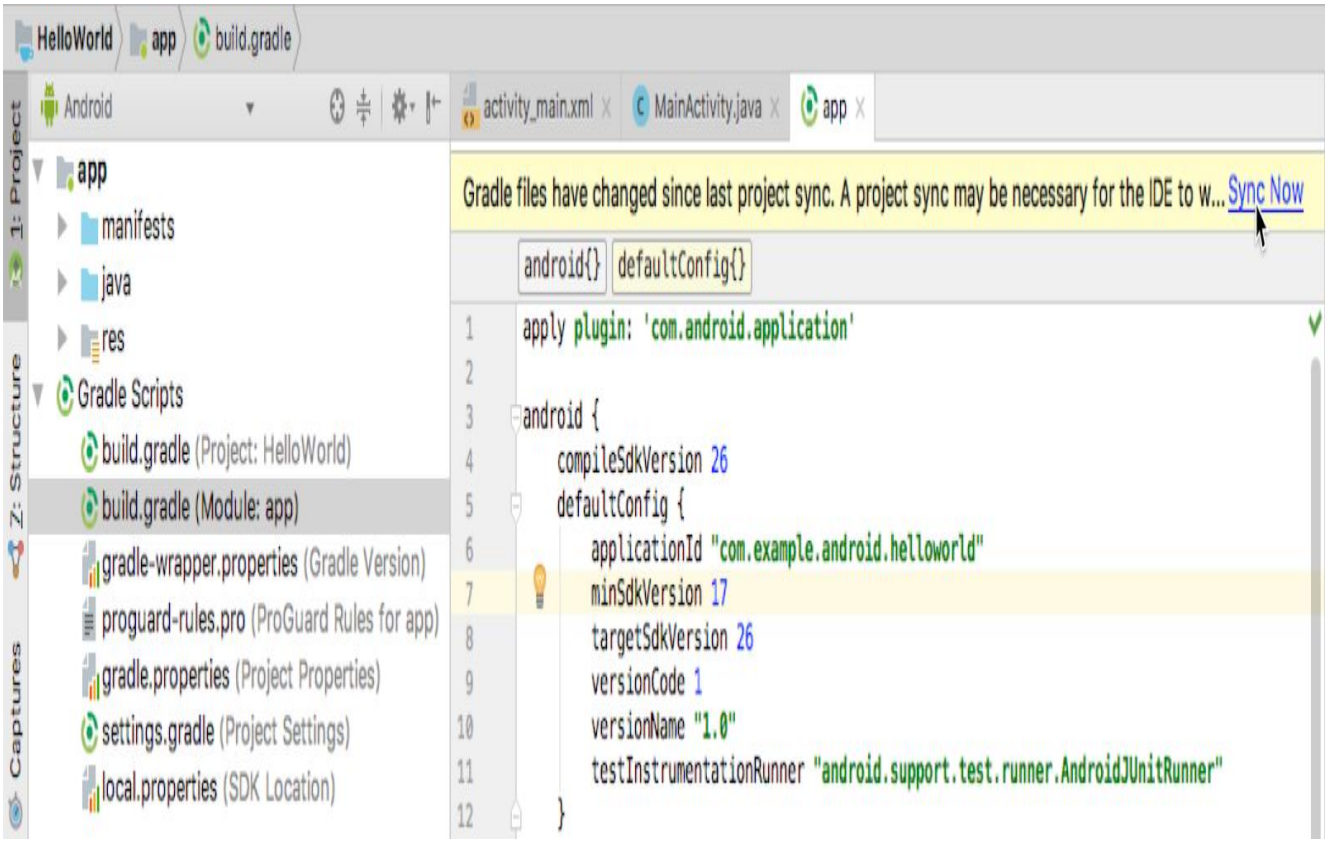
## Syncing your project

When you make changes to the build configuration files in a project, Android Studio requires that you *sync* the project files. During the sync, Android Studio imports the build configuration changes and runs checks to make sure the configuration won't create build errors.

To sync the project files, click **Sync Now** in the notification bar that appears when making a change (as shown in the figure below), or click the **Sync Project with**

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

**Gradle Files** button  in the menu bar.




If Android Studio notices any errors with the configuration — for example, if the source code uses API features that are only available in an API level higher than the `compileSdkVersion`—the **Messages** window appears to describe the issue.

# Running the app on an emulator or a device

With virtual device emulators, you can test an app on different devices such as tablets or smartphones—with different API levels for different Android versions—to make sure it looks good and works for most users. You don't have to depend on having a physical device available for app development.

The Android Virtual Device (AVD) manager creates a virtual device or emulator that simulates the configuration for a particular type of Android-powered device. Use the AVD Manager to define the hardware characteristics of a device and its API level, and to save it as a virtual device configuration. When you start the Android emulator, it reads a specified configuration and creates an emulated device on your computer that behaves exactly like a physical version of that device.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Creating a virtual device

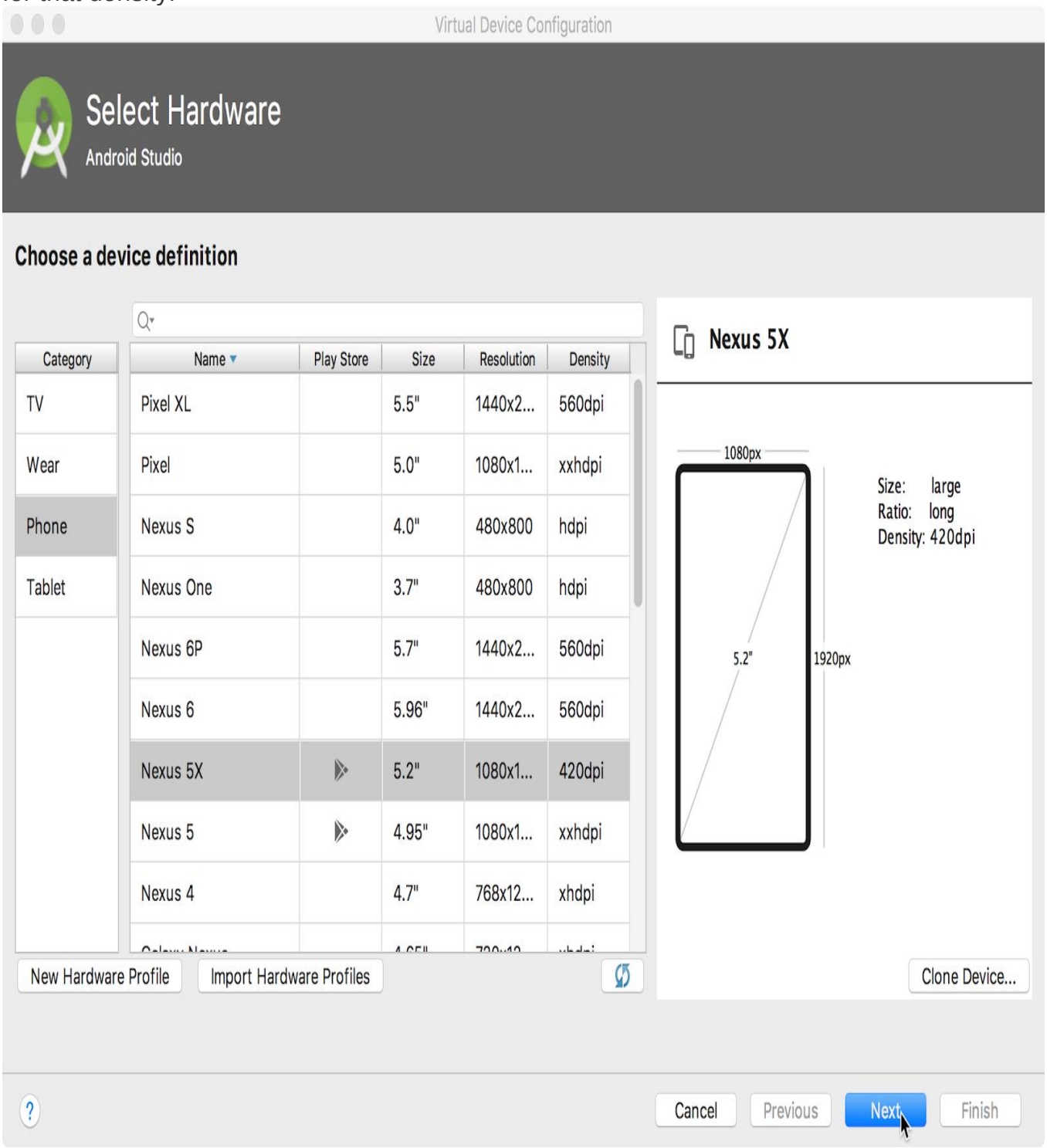
To run an emulator on your computer, use the AVD Manager to create a configuration that describes the virtual device. Select **Tools > Android > AVD Manager**, or click the **AVD Manager** icon  in the toolbar.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

The **Your Virtual Devices** screen appears showing all of the virtual devices created previously. Click the **+Create Virtual Device** button to create a new virtual device.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

You can select a device from a list of predefined hardware devices. For each device, the table provides a column for its diagonal display size (**Size**), screen resolution in pixels (**Resolution**), and pixel density (**Density**). For example, the pixel density of the Nexus 5 device is `xxhdpi`, which means the app uses the icons in the `xxhdpi` folder of the `mipmap` folder. Likewise, the app uses layouts and drawables from folders defined for that density.



After you click **Next**, the **System Image** screen appears for choosing the version of the Android system for the device. The **Recommended** tab shows the recommended systems for the device. More versions are available under the **x86 Images** and **Other Images** tabs. If a **Download** link is visible next to a system image version, it is not installed yet. Click the link to start the download, and click **Finish** when it's done.

## Running the app on the virtual device

To run the app on the virtual device you created in the previous section, follow these steps:

1. In Android Studio, select **Run > Run app** or click the  **Run icon** in the toolbar.

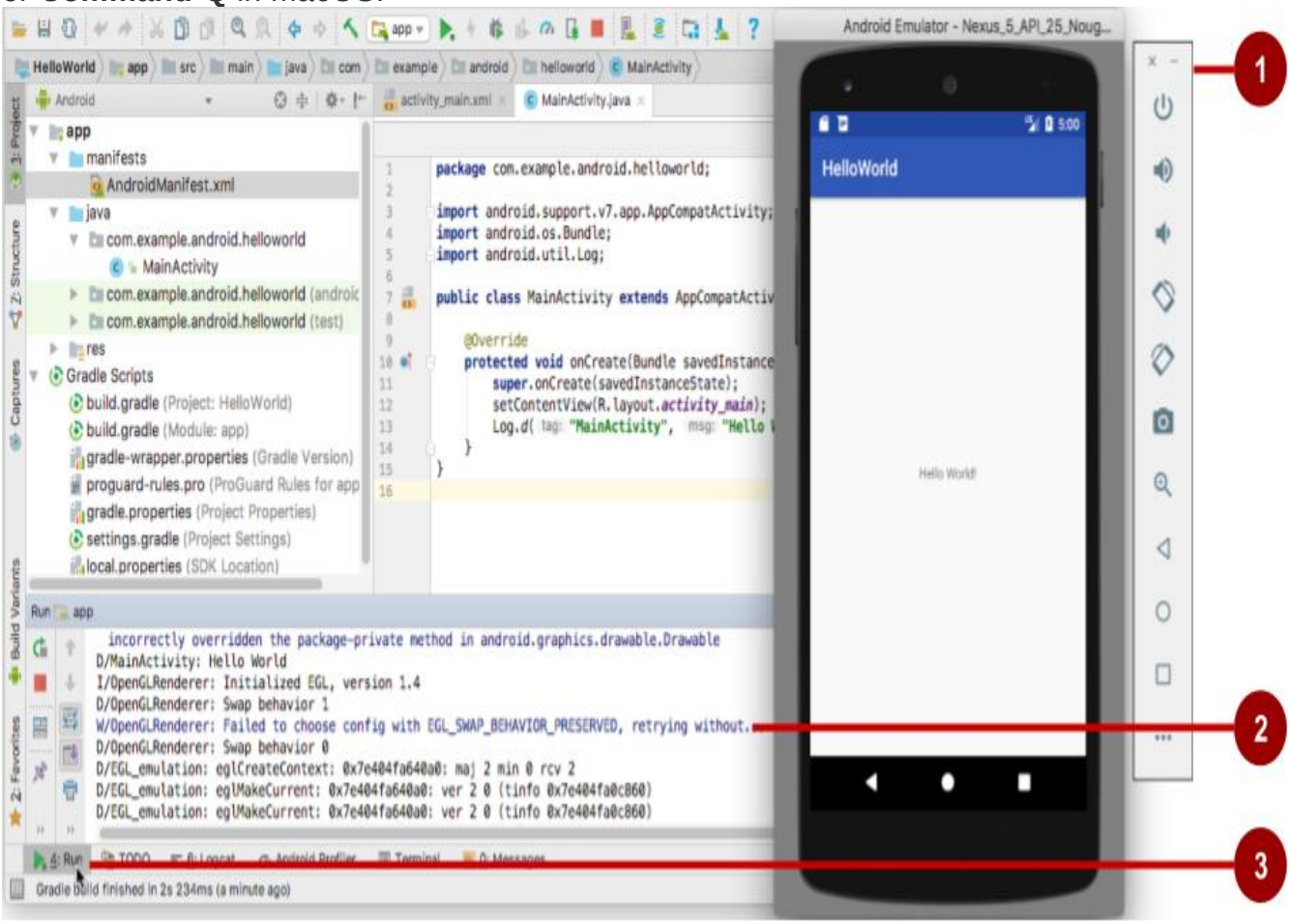
2. In the Select Deployment Target window, under Available Emulators, select the virtual device you created, and click **OK**.

The emulator starts and boots just like a physical device. Depending on the speed of your computer, the startup process might take a while. The app builds, and once the emulator is ready, Android Studio uploads the app to the emulator and runs it.

You should see the app created from the Empty Activity template ("Hello World") as shown in the following figure, which also shows Android Studio's **Run** pane that displays the actions performed to run the app on the emulator.

**Tip**: When testing on a virtual device, it is a good practice to start it up once, at the very beginning of your session. Do not close it until you are done testing your app, so that your app doesn't have to go through the device startup process again. To close the virtual device, select **Quit** from the menu or press **Control-Q** in Windows

or **Command-Q** in macOS.




The figure above shows the emulator and the run log:

1. The Emulator running the app.
2. The **Run** pane, which shows the actions taken to install and run the app.
3. The **Run** tab, which you click to open or close the **Run** pane.

## Running the app on a physical device

Always test your apps on a physical device. While emulators are useful, they can't show all possible device states, such as what happens if an incoming call occurs while the app is running. To run the app on a physical device, you need the following:

- An Android-powered device such as a phone or tablet.
- A data cable to connect your Android-powered device to your computer via the USB port.
- If you are using a Linux or Windows system, you may need to perform additional steps to run on a hardware device. Check the Using Hardware Devices documentation. You may also need to install the appropriate USB driver for your device. See OEM USB Drivers.

To let Android Studio communicate with your Android-powered device, you must turn on USB Debugging on the device. You enable USB Debugging in the device's **Developer options** settings. (Note that enabling USB Debugging is not the same as rooting your device.)

On Android 4.2 and higher, the **Developer options** screen is hidden by default. To show developer options and enable USB Debugging:

1. On your device, open **Settings > About phone** and tap **Build number** seven times.
2. Return to the previous screen (**Settings**). **Developer options** appears at the bottom of the list. Tap **Developer options**.
3. Select **USB Debugging**.
4. Connect the device and run the app from Android Studio.

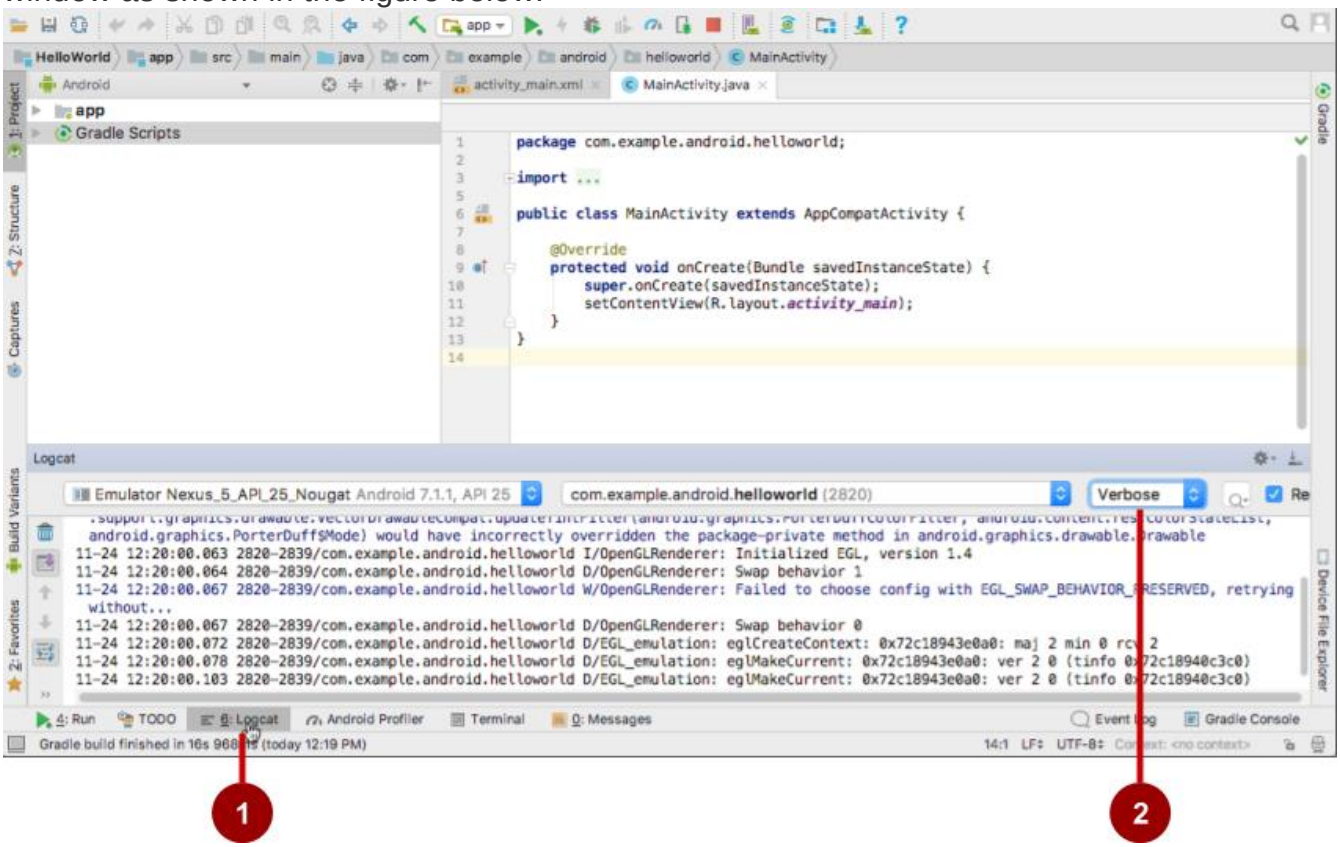
# Using the log

The log is a powerful debugging tool you can use to look at values, execution paths, and exceptions. After you add logging statements to an app, your log messages appear along with general log messages in the **Logcat** pane.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Viewing log messages

To see the **Logcat** pane, click the **Logcat** tab at the bottom of the Android Studio window as shown in the figure below.




In the figure above:

1. The **Logcat** tab for opening and closing the **Logcat** pane, which displays information about your app as it is running. If you add `Log` statements to your app, `Log` messages appear here.
2. The `Log` level menu set to **Verbose** (the default), which shows all `Log` messages. Other settings include **Debug**, **Error**, **Info**, and **Warn**.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Adding logging statements to your app

Logging statements add whatever messages you specify to the log. Adding logging statements at certain points in the code allows the developer to look at values, execution paths, and exceptions. For example, the following logging statement adds `"MainActivity"` and `"Hello World"` to the log:

```
Log.d("MainActivity", "Hello World");
```

The following are the elements of this statement:

- `Log`: The `Log` class for sending log messages to the **Logcat** pane.
- `d`: The **Debug** `Log` level setting to filter log message display in the **Logcat** pane. Other log levels are `e` for **Error**, `w` for **Warn**, and `i` for **Info**. You assign a log level so that you can filter the log messages using the drop-down menu in the center of the **Logcat** pane.
- `"MainActivity"`: The first argument is a tag which can be used to filter messages in the **Logcat** pane. This tag is commonly the name of the `Activity` from which the message originates. However, you can name the tag anything that is useful to you for debugging.
- `"Hello world"`: The second argument is the actual message.

By convention, log tags are defined as constants for the `Activity`:

```
private static final String LOG_TAG = MainActivity.class.getSimpleName();
```

Use the constant in the logging statements:

```
Log.d(LOG_TAG, "Hello World");
```

After you add the `Log.d` statement shown above, follow these steps to see the log message:

1. If the **Logcat** pane is not already open, click the **Logcat** tab at the bottom of Android Studio to open it.
2. Change the `Log` level in the **Logcat** pane to **Debug**. (You can also leave the `Log` level as **Verbose**, because there are so few log messages.)
3. Run your app on a virtual device.

The following message should appear in the **Logcat** pane:

```
11-24 14:06:59.001 4696-4696/? D/MainActivity: Hello World
```

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Related practical

The related practical is 1.1 Android Studio and Hello World.

# Learn more

Android Studio documentation:

- Android Studio download page
- Meet Android Studio
- Reading and writing logs
- Android Virtual Device (AVD) manager
- App Manifest
- Configure Your Build
- `Log` class
- Configure Build Variants
- Create and Manage Virtual Devices
- Sign Your App
- Shrink Your Code and Resources

Android API Guide, "Develop" section:

- Introduction to Android
- Platform Architecture
- UI Overview
- Platform versions
- Supporting Different Platform Versions
- Supporting Multiple Screens

Other:

- Wikipedia: Summary of Android version history
- Groovy syntax
- How do I install Java?
- Installing the JDK Software and Setting JAVA_HOME
- Gradle site
- Gradle Wikipedia page

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# 1.2: Layouts and resources for the UI

**Contents:**



This chapter describes the screen's user interface (UI) layout and other resources you create for your app, and the code you would use to respond to a user's tap of a UI element.

## Views

The UI consists of a hierarchy of objects called *views* — every element of the screen is a `View`. The `View` class represents the basic building block for all UI components, and the base class for classes that provide interactive UI components such as buttons, checkboxes, and text entry fields.
A `View` has a location, expressed as a pair of left and top coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the density-independent pixel (dp).
The Android system provides hundreds of predefined `View` subclasses. Commonly used `View` subclasses described over several lessons include:

- `TextView` for displaying text
- `EditText` to enable the user to enter and edit text
- `Button` and other clickable elements (such as `RadioButton`, `CheckBox`, and `Spinner`) to provide interactive behavior
- `ScrollView` and `RecyclerView` to display scrollable items
- `ImageView` for displaying images
- `ConstraintLayout` and `LinearLayout` for containing other views and positioning them

You can define a `View` to appear on the screen and respond to a user tap. A `View` can also be defined to accept text input, or to be invisible until needed.
You can specify `View` elements in layout resource files. Layout resources are written in XML and listed within the **layout** folder in the **res** folder in the **Project > Android** pane.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

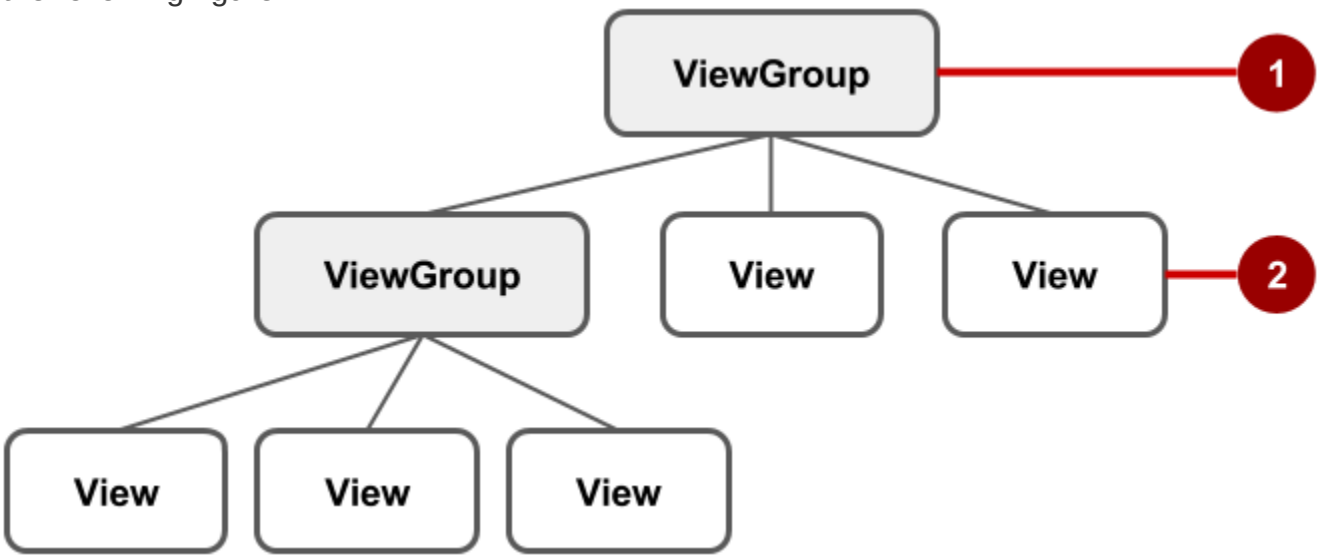
# ViewGroup groups

`View` elements can be grouped inside a `ViewGroup`, which acts as a container. The relationship is parent-child, in which the *parent* is a `ViewGroup`, and the *child* is a `View` or another `ViewGroup`. The following are commonly used `ViewGroup` groups:

- `ConstraintLayout`: A group that places UI elements (child `View` elements) using constraint connections to other elements and to the layout edges (parent `View`).
- `ScrollView`: A group that contains one other child `View` element and enables scrolling the child `View` element.
- `RecyclerView`: A group that contains a list of other `View` elements or `ViewGroup` groups and enables scrolling them by adding and removing `View` elements dynamically from the screen.

# Layout ViewGroup groups

The `View` elements for a screen are organized in a hierarchy. At the *root* of this hierarchy is a `ViewGroup` that contains the layout of the entire screen.
The `ViewGroup` can contain child `View` elements or other `ViewGroup` groups as shown in the following figure.

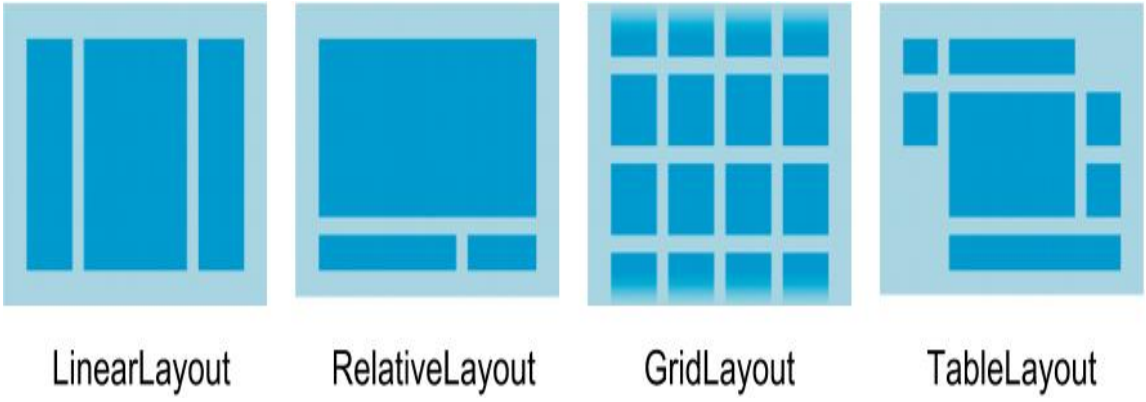


In the figure above:

1. The *root* `ViewGroup`.
2. The first set of child `View` elements and `ViewGroup` groups whose parent is the root.

Some `ViewGroup` groups are designated as *layouts* because they organize child `View` elements in a specific way and are typically used as the root `ViewGroup`. Some examples of layouts are:

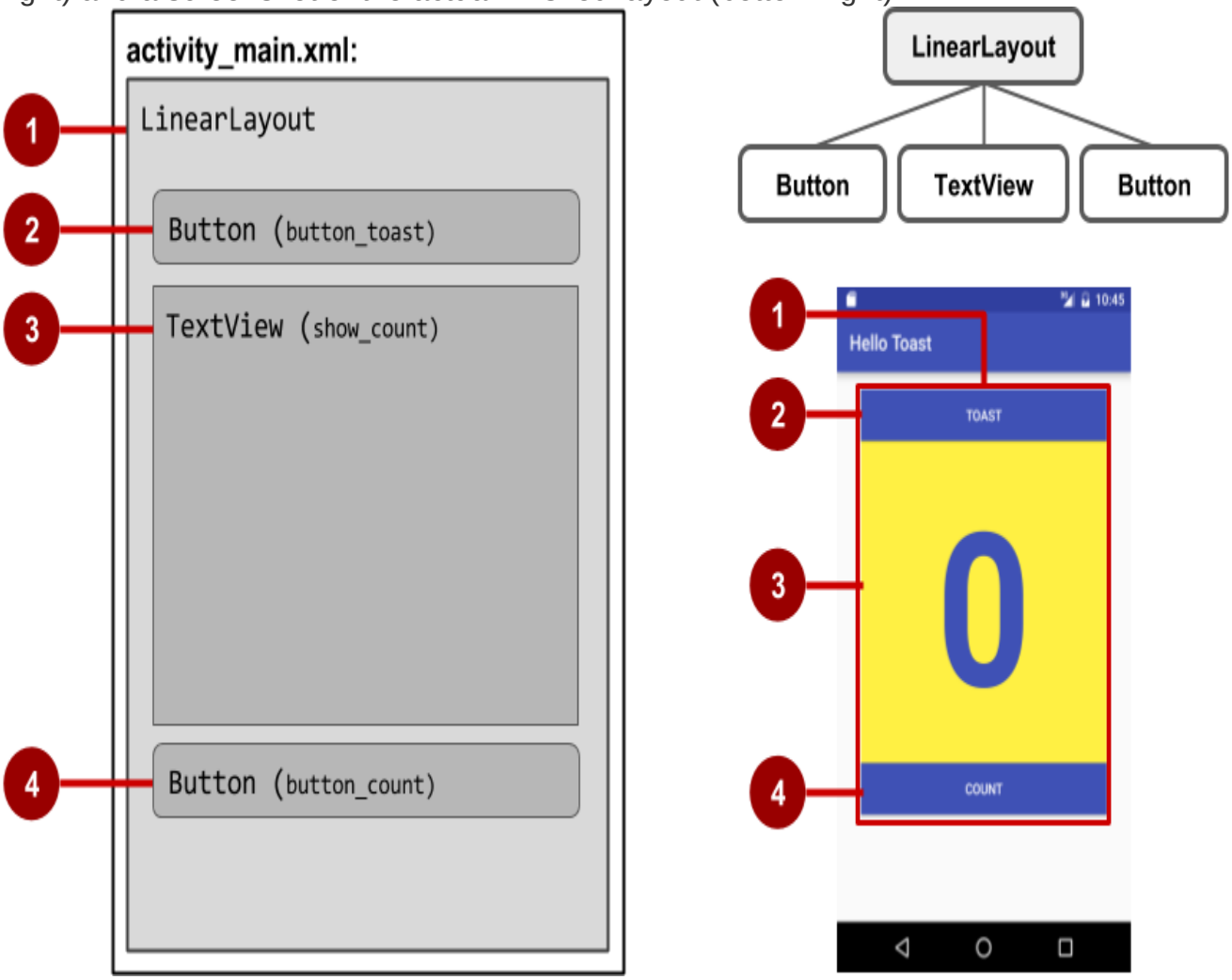
- `ConstraintLayout`: A group of child `View` elements using constraints, edges, and guidelines to control how the elements are positioned relative to other elements in the layout. `ConstraintLayout` was designed to make it easy to click and drag `View` elements in the layout editor.
- `LinearLayout`: A group of child `View` elements positioned and aligned horizontally or vertically.
- `RelativeLayout`: A group of child `View` elements in which each element is positioned and aligned relative to other elements within the `ViewGroup`. In other words, the positions of the child `View` elements can be described in relation to each other or to the parent `ViewGroup`.
- `TableLayout`: A group of child `View` elements arranged into rows and columns.
- `FrameLayout`: A group of child `View` elements in a stack. `FrameLayout` is designed to block out an area on the screen to display one `View`. Child `View` elements are drawn in a stack, with the most recently added child on top. The size of the `FrameLayout` is the size of its largest child `View` element.
- `GridLayout`: A group that places its child `View` elements in a rectangular grid that can be scrolled.




**Tip**: Learn more about different layout types in Common Layout Objects.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

A simple example of a `LinearLayout` with child `View` elements is shown below as a diagram of the layout file (`activity_main.xml`), along with a hierarchy diagram (top right) and a screenshot of the actual finished layout (bottom right).




In the figure above:

1. `LinearLayout`, the root `ViewGroup`, contains all the child `View` elements in a vertical orientation.
2. `Button` (`button_toast`). The first child `View` element appears at the top in the `LinearLayout`.
3. `TextView` (`show_count`). The second child `View` element appears under the first child `View` element in the `LinearLayout`.
4. `Button` (`button_count`). The third child `View` element appears under the second child `View` element in the `LinearLayout`.

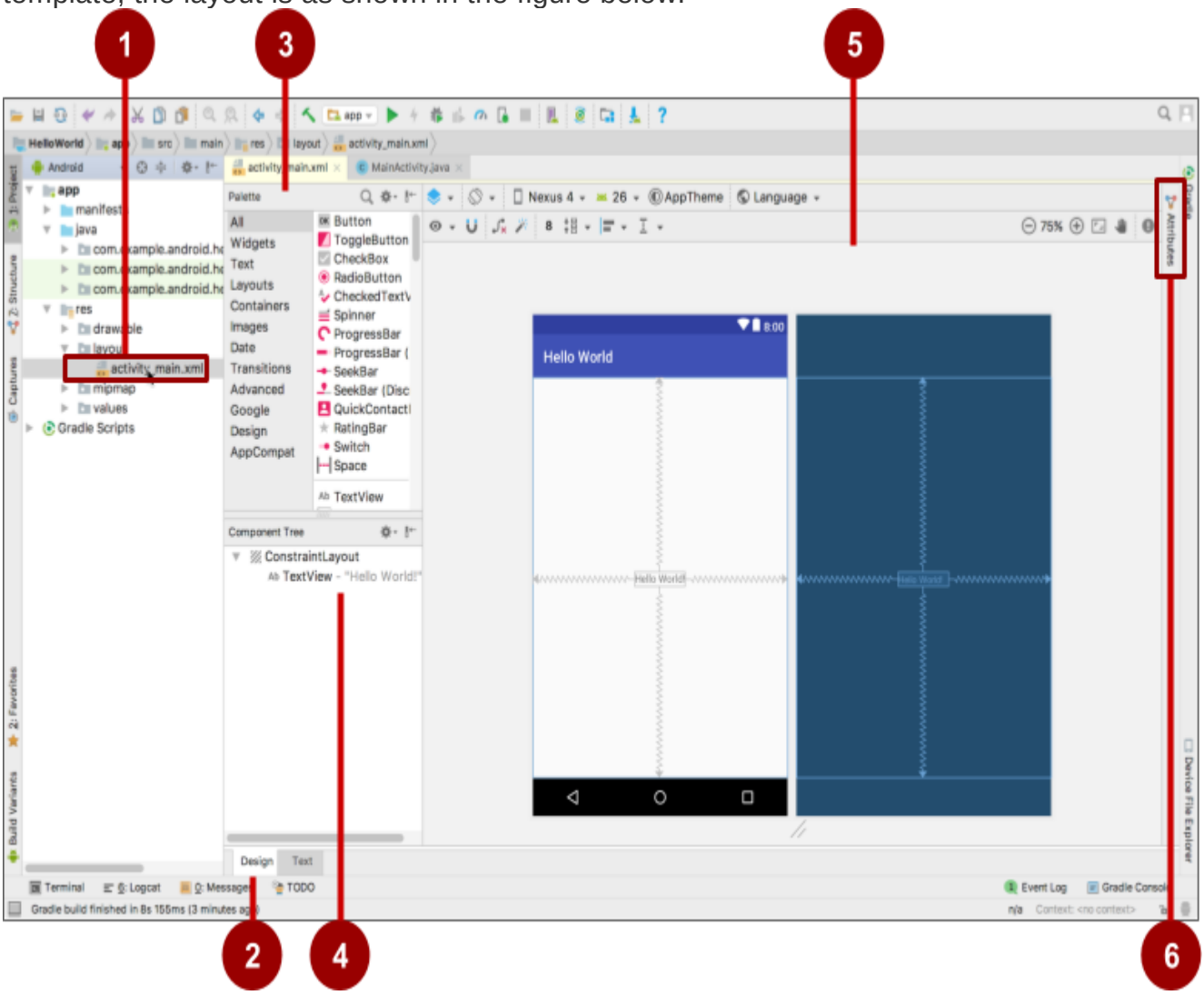
The layout hierarchy can grow to be complex for an app that shows many `View` elements on a screen. It's important to understand the hierarchy, as it affects whether `View` elements are visible and how efficiently they are drawn.
**Tip**: You can explore the layout hierarchy of your app using Hierarchy Viewer. It shows a tree view of the hierarchy and lets you analyze the performance of `View` elements on an Android-powered device. Performance issues are covered in a subsequent chapter.

# The layout editor

You define layouts in the layout editor, or by entering XML code.

The layout editor shows a visual representation of XML code. You can drag `View` elements into the design or blueprint pane and arrange, resize, and specify attributes for them. You immediately see the effect of changes you make.
To use the layout editor, double-click the XML layout file (**activity_main.xml**). The layout editor appears with the **Design** tab at the bottom highlighted. (If the **Text** tab is highlighted and you see XML code, click the **Design** tab.) For the Empty Activity

template, the layout is as shown in the figure below.

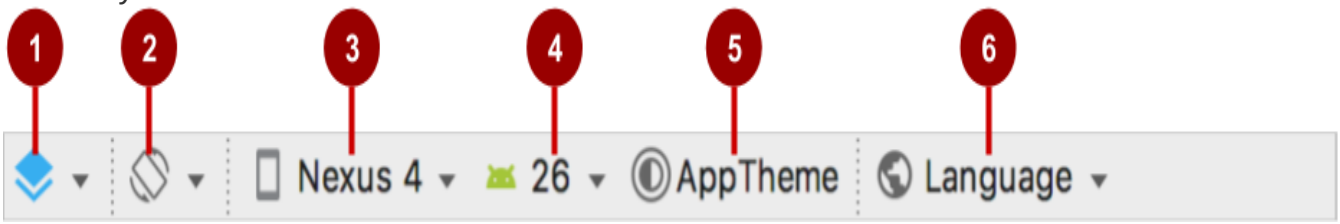



1. XML layout file (**activity_main.xml**).
2. **Design** and **Text** tabs. Click **Design** to see the layout editor, or **Text** to see XML code.
3. **Palette** pane. The Palette pane provides a list of UI elements and layouts. Add an element or layout to the UI by dragging it into the design pane.
4. **Component Tree**. The Component Tree pane shows the layout hierarchy. Click a `View` element or `ViewGroup` in this pane to select it. `View` elements are organized into a tree hierarchy of parents and children, in which a child inherits the attributes of its parent. In the figure above, the `TextView` is a child of the `ConstraintLayout`.
5. Design and blueprint panes. Drag `View` elements from the **Palette** pane to the design or blueprint pane to position them in the layout. In the figure above, the layout shows only one element: a `TextView` that displays "Hello World".
6. **Attributes** tab. Click **Attributes** to display the **Attributes** pane for setting attributes for a `View` element.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Layout editor toolbars

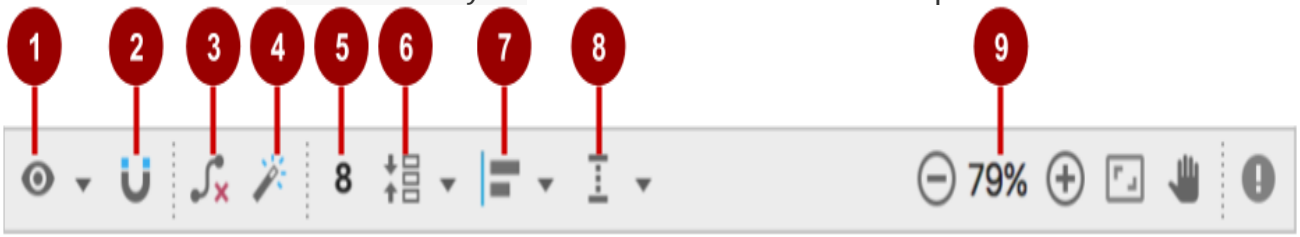
The layout editor toolbars provide buttons to configure your layout and change its appearance. The top toolbar lets you configure the appearance of the layout preview in the layout editor:




The figure above shows the top toolbar of the layout editor:

1. **Select Design Surface**: Select **Design** to display a color preview of the UI elements in your layout, or **Blueprint** to show only outlines of the elements. To see *both* panes side by side, select **Design + Blueprint**.
2. **Orientation in Editor**: Select **Portrait** or **Landscape** to show the preview in a vertical or horizontal orientation. The orientation setting lets you preview the layout orientations without running the app on an emulator or device. To create alternative layouts, select **Create Landscape Variation** or other variations.
3. **Device in Editor**: Select the device type (phone/tablet, Android TV, or Android Wear).
4. **API Version in Editor**: Select the version of Android to use to show the preview.
5. **Theme in Editor**: Select a theme (such as **AppTheme**) to apply to the preview.
6. **Locale in Editor**: Select the language and locale for the preview. This list displays only the languages available in the string resources (see the lesson on localization for details on how to add languages). You can also select **Preview as Right To Left** to see the layout as if an RTL language had been chosen.

The layout editor also offers a second toolbar that lets you configure the appearance of UI elements in a `ConstraintLayout` and zoom in and out of the preview:



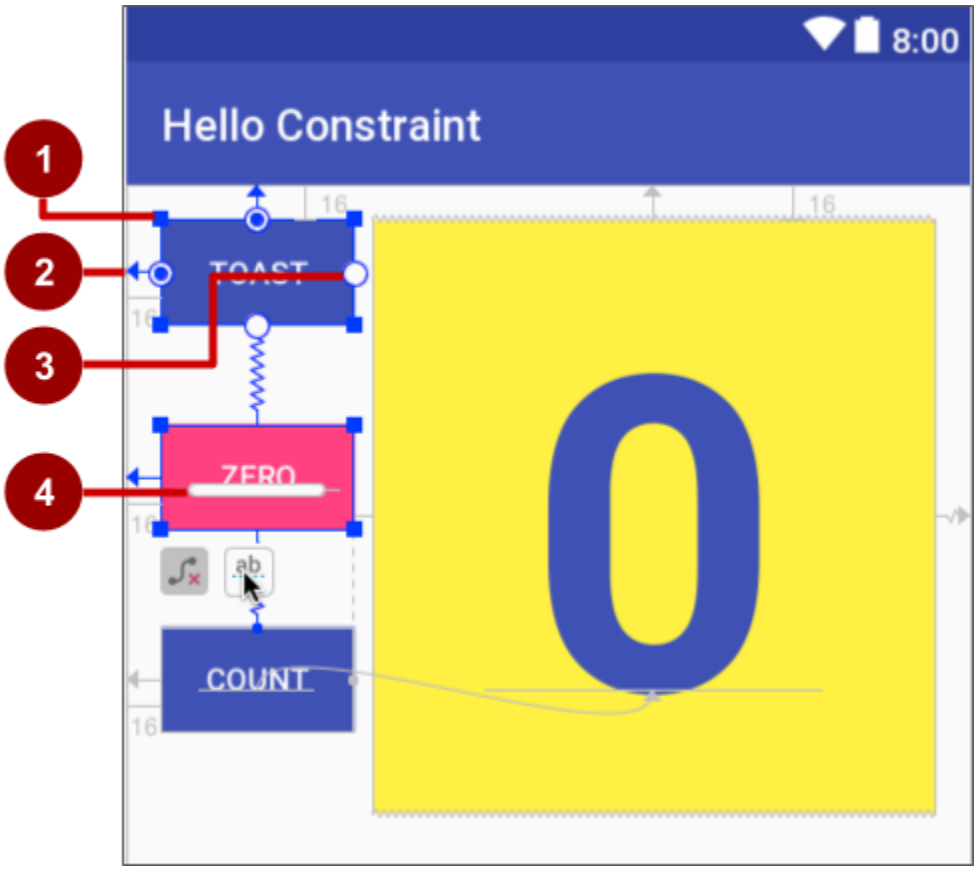
https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

The figure above shows the `ConstraintLayout` editing toolbar:

1. **Show**: Select **Show Constraints** and **Show Margins** to show them in the preview, or to stop showing them.
2. **Autoconnect**: Enable or disable Autoconnect. With Autoconnect enabled, you can drag any element (such as a `Button`) to any part of a layout to generate constraints against the parent layout.
3. **Clear All Constraints**: Clear all constraints in the entire layout.
4. **Infer Constraints**: Create constraints by inference.
5. **Default Margins**: Set the default margins.
6. **Pack**: Pack or expand the selected elements.
7. **Align**: Align the selected elements.
8. **Guidelines**: Add vertical or horizontal guidelines.
9. Zoom controls: Zoom in or out.

## Using ConstraintLayout

The layout editor offers more features in the **Design** tab when you use a `ConstraintLayout`, including handles for defining constraints.
A *constraint* is a connection or alignment to another UI element, to the parent layout, or to an invisible guideline. Each constraint appears as a line extending from a circular handle. After you select a UI element in the **Component Tree** pane or click it in the layout editor, the element shows a resizing handle on each corner and a circular constraint handle in the middle of each side.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

The figure above shows the constraint and resizing handles on `View` elements in a layout:

1. **Resizing handle**.
2. **Constraint line and handle**. In the figure, the constraint aligns the left side of the **Toast** `Button` to the left side of the layout.
3. **Constraint handle** without a constraint line.
4. **Baseline handle**. The baseline handle aligns the text baseline of an element to the text baseline of another element.

In the blueprint or design panes, the following handles appear on the `TextView` element:

- **Constraint handle**: To create a constraint, click a constraint handle, shown as a circle on each side of an element. Then drag the circle to another constraint handle or to a parent boundary. A zigzag line represents the constraint.

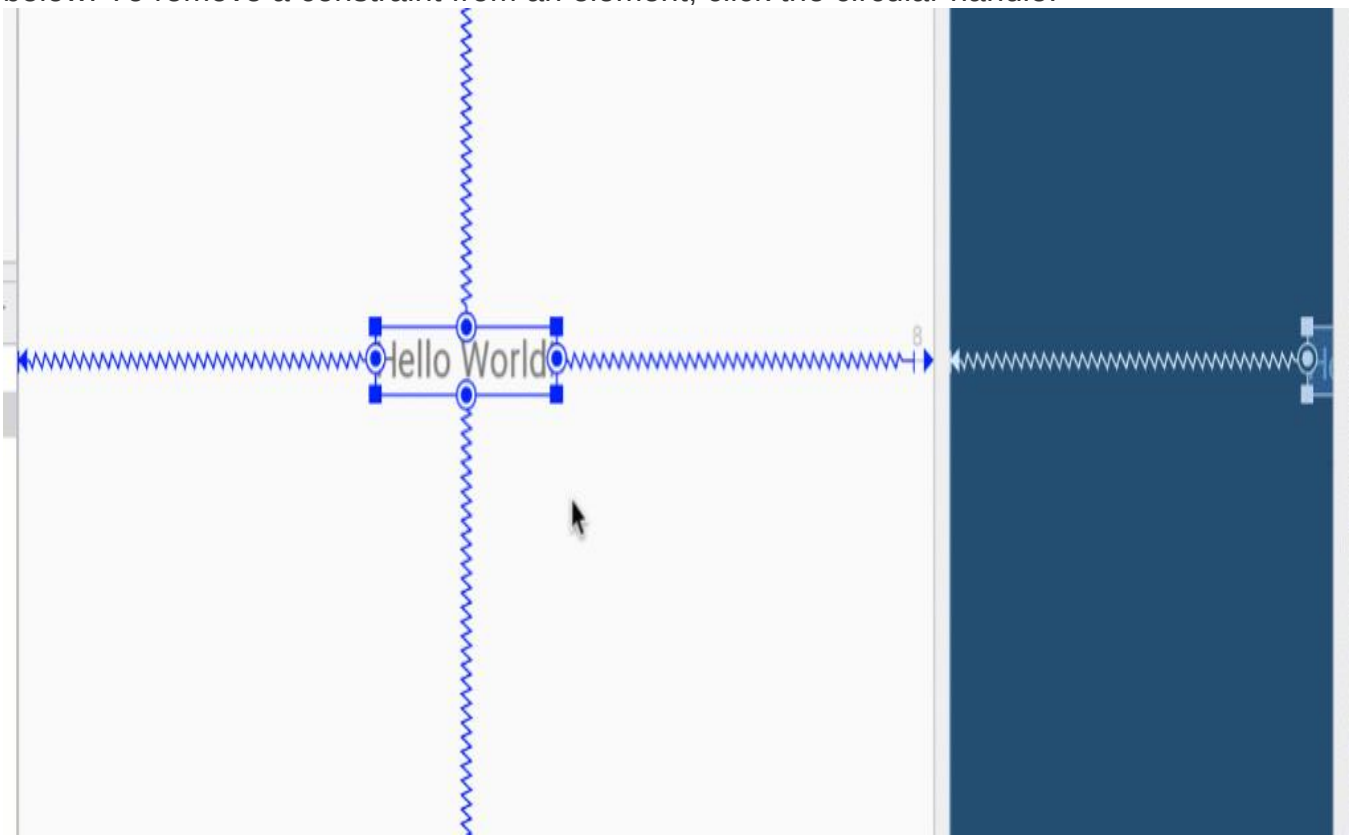
- **Resizing handle**: You can drag the square resizing handles to resize the element. While dragging, the handle changes to an angled corner.

You can drag the resizing handles on each corner of the UI element to resize it, but doing so hard-codes the width and height dimensions, which you should avoid for most elements because hard-coded dimensions don't adapt to different screen densities.

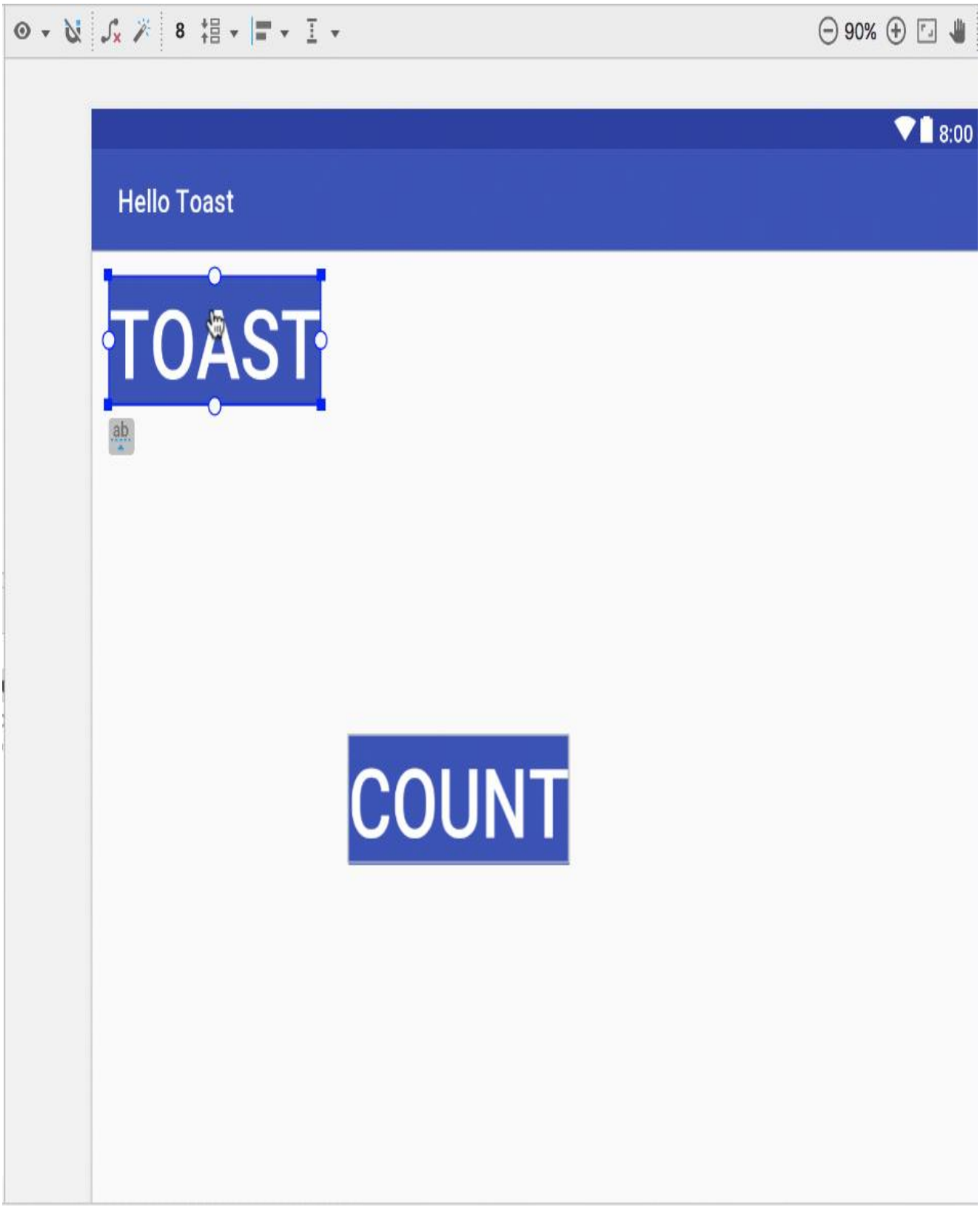
https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## *Constraining a UI element*

To add a constraint to a UI element, click the circular handle and drag a line to another element or to the side of a layout, as shown in the two animated figures below. To remove a constraint from an element, click the circular handle.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

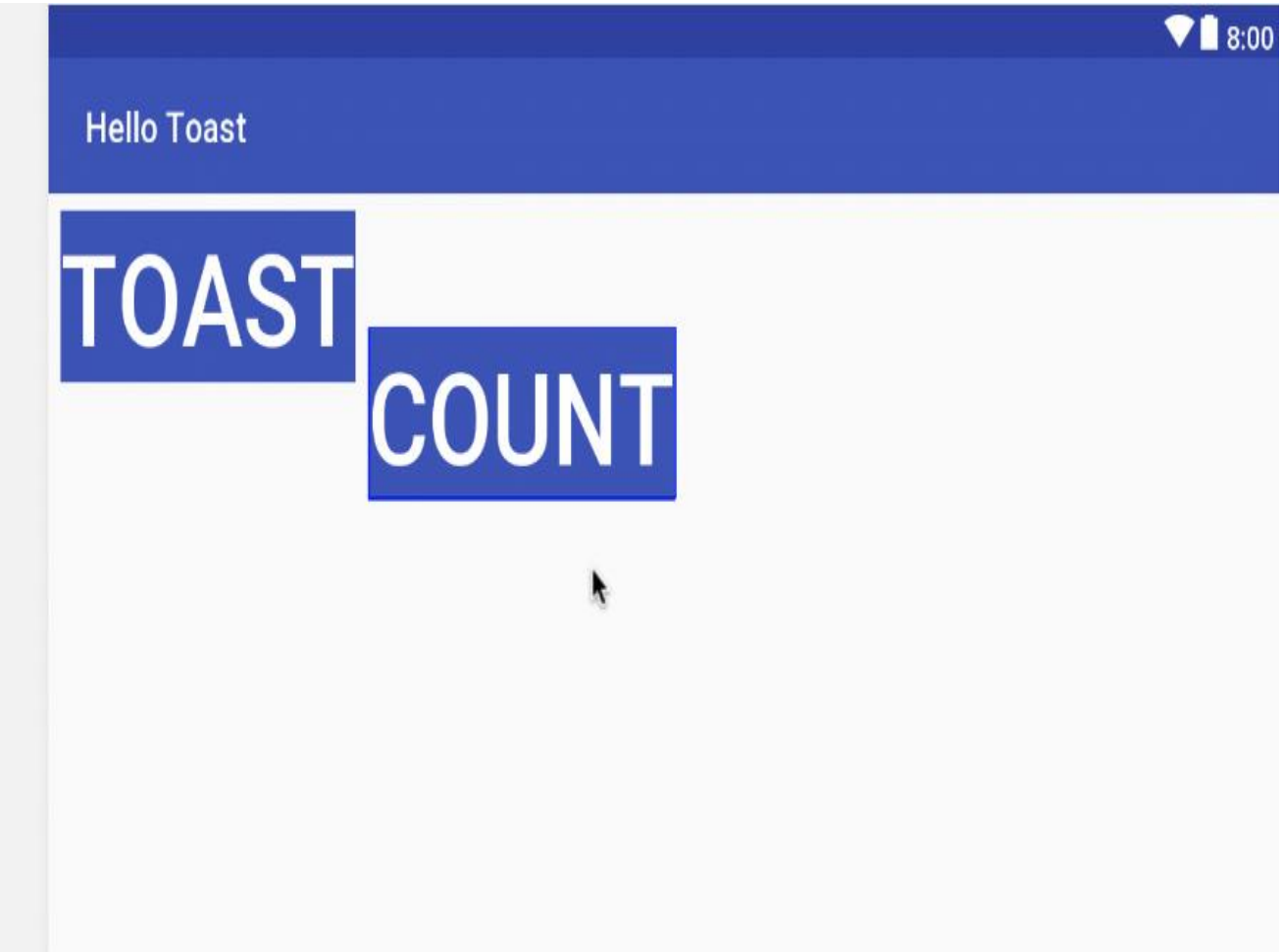


The constraints you define in the layout editor are created as XML attributes, which you can see in the **Text** tab as described in "Editing XML directly" in this chapter. For example, the following XML code is created constraining the top of an element to the top of its parent:

```
app:layout_constraintTop_toTopOf="parent"
```

## *Using a baseline constraint*

You can align one UI element that contains text, such as a `TextView` or `Button`, with another UI element that contains text. A *baseline constraint* lets you constrain the elements so that the text baselines match. Select the UI element that has text, and then hover your pointer over the element until the baseline constraint button  appears underneath the element.
Click the baseline constraint button. The baseline handle appears, blinking in green as shown in the animated figure. Drag a baseline constraint line to the baseline of the other UI element.

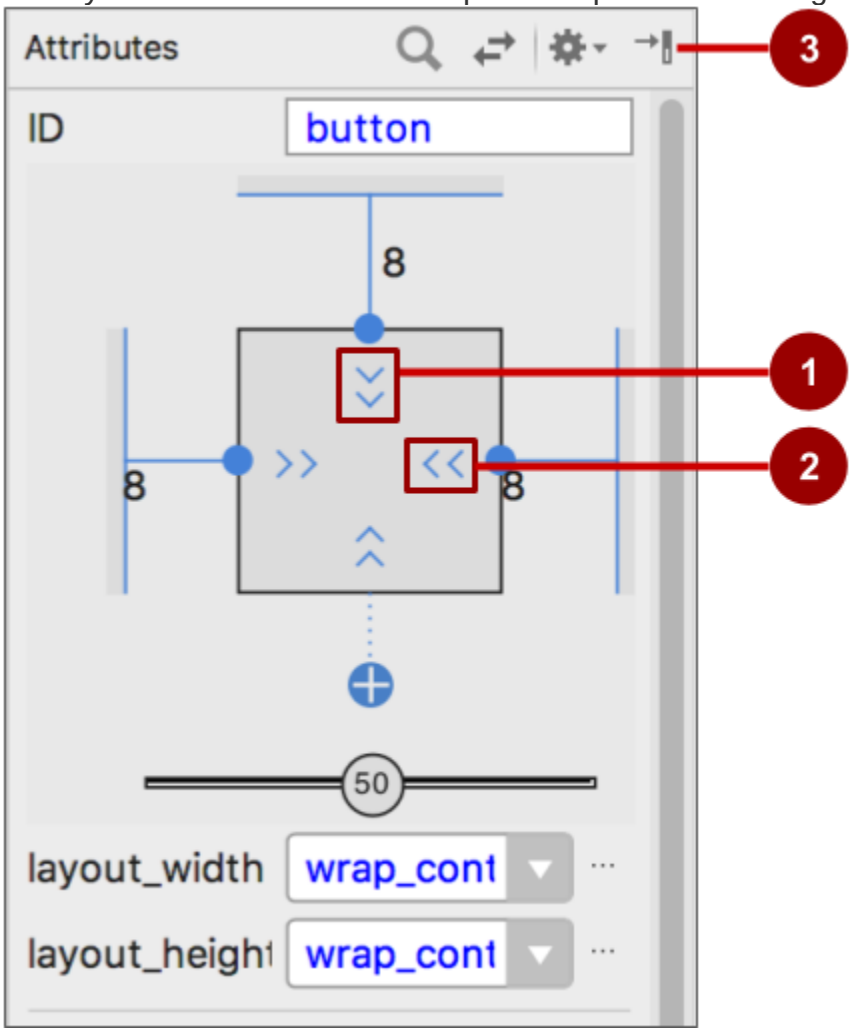



**Tip**: For an in-depth tutorial on using `ConstraintLayout`, see Using ConstraintLayout to design your views.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Using the Attributes pane

The **Attributes** pane offers access to all of the XML attributes you can assign to a UI element. You can find the attributes (known as *properties*) common to all views in the `View` class documentation.
To show the **Attributes** pane, click the **Attributes** tab on the right side of the layout editor. The **Attributes** pane includes a square sizing panel called the *view inspector*. The symbols inside the view inspector represent the height and width settings.



The figure above shows the **Attributes** pane:

1. **Vertical view size control**. The vertical size control, which appears on the top and bottom of the view inspector, specifies the `layout_height` property. The angles indicate that this size control is set to `wrap_content`, which means the UI element expands vertically as needed to fit its contents. The "8" indicates a standard margin set to 8 dp.
2. **Horizontal view size control**. The horizontal size control, which appears on the left and right of the view inspector, specifies the `layout_width`. The angles indicate that this size control is set to `wrap_content`, which means the UI element expands horizontally as needed to fit its contents, up to a margin of 8 dp.
3. **Attributes** pane close button. Click to close the pane.

The `layout_width` and `layout_height` attributes in the **Attributes** pane change as you change the inspector's horizontal and vertical size controls. These attributes can take one of three values for a `ConstraintLayout`:

- The `match_constraint` setting expands the UI element to fill its parent by width or height up to a margin, if a margin is set. The parent in this case is the `ConstraintLayout`.
- The `wrap_content` setting shrinks the UI element to the size of its content. If there is no content, the element becomes invisible.
- To specify a fixed size that's adjusted for the screen size of the device, set a number of `dp` (density-independent pixels). For example, `16dp` means 16 density-independent pixels.

**Tip**: If you change the `layout_width` attribute using its popup menu, the `layout_width` attribute is set to zero because there is no set dimension. This setting is the same as `match_constraint`—the UI element can expand as much as possible to meet constraints and margin settings.
The **Attributes** pane offers access to all of the attributes you can assign to a `View` element. You can enter values for each attribute, such as the `android:id`, `background`, `textColor`, and `text` attributes.

# Creating layout variants for orientations and devices

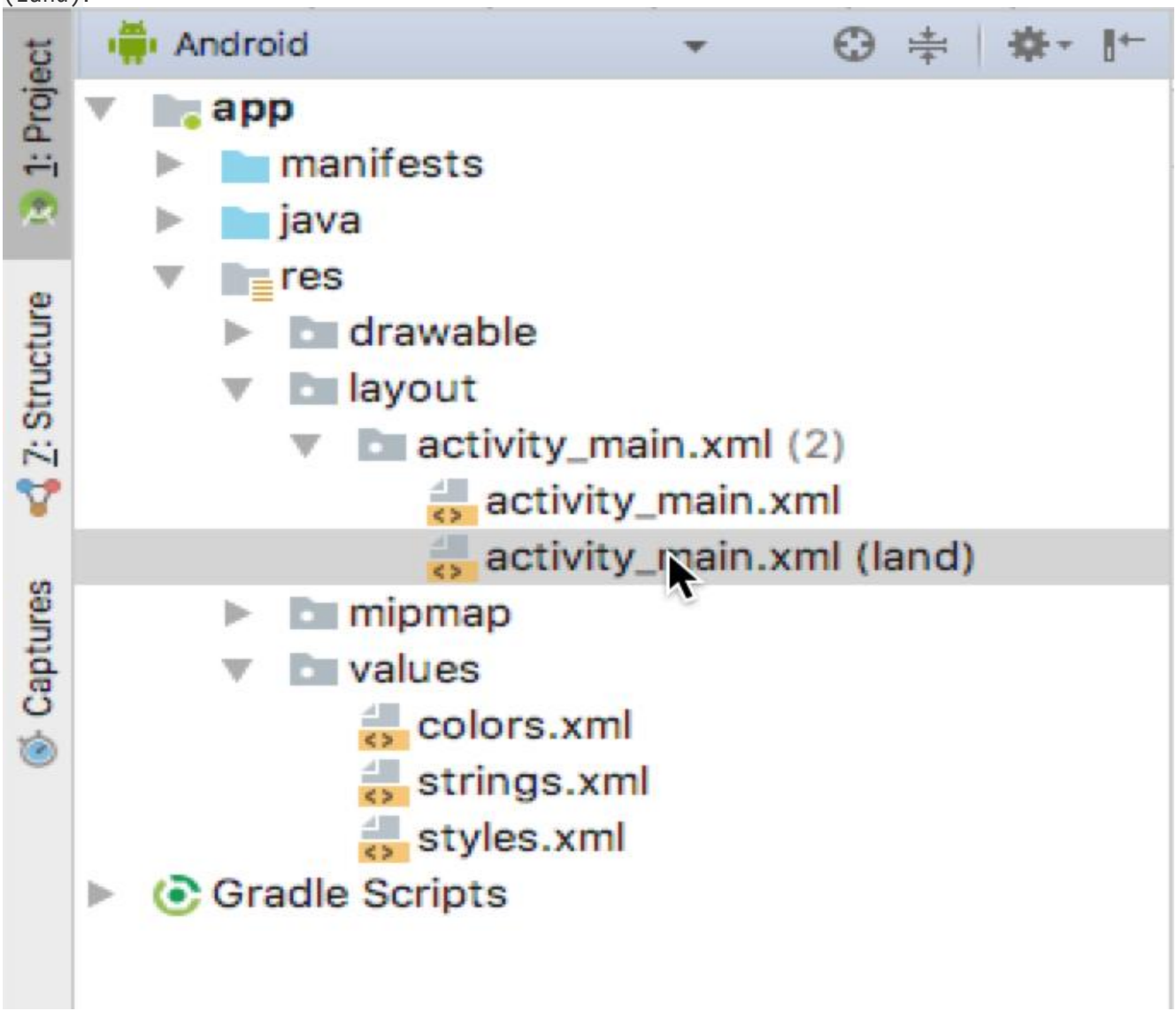
You can preview an app's layout with a horizontal orientation, and with different devices, without having to run the app on an emulator or device.

To preview the layout for a different orientation, click the **Orientation in Editor** button  in the top toolbar. To show the layout in a horizontal orientation, select **Switch to Landscape**. To return to vertical orientation, select **Switch to Portrait**.

You can also preview the layout for different devices. Click the **Device in Editor** button Nexus 5 in the top toolbar, and select a different device in the drop-down menu. For example, select **Nexus 4**, **Nexus 5**, and then **Pixel** to see differences in the previews.

To create a variant of the layout strictly for the horizontal orientation, leaving the vertical orientation layout alone: click the **Orientation in Editor** button and select **Create Landscape Variation**. A new editor window opens with the **land/activity_main.xml** tab showing the layout for the landscape (horizontal) orientation. You can change this layout, which is specifically for horizontal orientation, without changing the original portrait (vertical) orientation.

In the **Project > Android** pane, look inside the **res > layout** directory. You see that Android Studio automatically creates the variant for you, called `activity_main.xml (land)`.



To create a layout variant for tablet-sized screens, click the **Orientation in Editor** button and select **Create layout x-large Variation**. A new editor window opens with the **xlarge/activity_main.xml** tab showing the layout for a tablet-sized device. The editor also picks a tablet device, such as the Nexus 9 or Nexus 10, for the preview. In the **Project > Android** pane, look inside the **res > layout** directory. You see that Android Studio automatically creates the variant for you, called `activity_main.xml (xlarge)`. You can change this layout, which is specifically for tablets, without changing the other layouts.

# Editing XML directly

It is sometimes quicker and easier to edit the XML code directly, especially when copying and pasting the code for similar views.

To view and edit the XML code, open the XML layout file. The layout editor appears with the **Design** tab at the bottom highlighted. Click the **Text** tab to see the XML code. The following shows the XML code for a `LinearLayout` with two `Button` elements with a `TextView` in the middle:

```
<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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.android.hellotoast.MainActivity">

    <Button
        android:id="@+id/button_toast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="@color/colorPrimary"
        android:onClick="showToast"
        android:text="@string/button_label_toast"
        android:textColor="@android:color/white" />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_vertical"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="#FFFF00"
        android:text="@string/count_initial_value"
        android:textAlignment="center"
        android:textColor="@color/colorPrimary"
```

```
        android:textSize="160sp"
        android:textStyle="bold"
        android:layout_weight="1"/>

    <Button
        android:id="@+id/button_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:background="@color/colorPrimary"
        android:onClick="countUp"
        android:text="@string/button_label_count"
        android:textColor="@android:color/white" />
</LinearLayout>
```

# XML attributes (view properties)

Views have *properties* that define where a view appears on the screen, its size, how the view relates to other views, and how it responds to user input. When defining views in XML or in the layout editor's **Attributes** pane, the properties are referred to as *attributes*.

For example, in the following XML description of a `TextView`, the `android:id`, `android:layout_width`, `android:layout_height`, `android:background`, are XML attributes that are translated automatically into the `TextView` properties:

```
<TextView
       android:id="@+id/show_count"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:background="@color/myBackgroundColor"
       android:textStyle="bold"
       android:text="@string/count_initial_value" />
```

Attributes generally take this form:

`android:`*attribute_name*`="`*value*`"`

The *attribute_name* is the name of the attribute. The *value* is a string with the value for the attribute. For example:

```
android:textStyle="bold"
```

If the *value* is a resource, such as a color, the `@` symbol specifies what kind of resource. For example:

```
android:background="@color/myBackgroundColor"
```

The background attribute is set to the color resource identified as `myBackgroundColor`, which is declared to be `#FFF043`. Color resources are described in Style-related attributes in this chapter.

Every `View` and `ViewGroup` supports its own variety of XML attributes:

- Some attributes are specific to a `View` subclass. For example, the `TextView` subclass supports the `textSize` attribute. Any elements that extend the `TextView` subclass inherit these subclass-specific attributes.
- Some attributes are common to all `View` elements, because they are inherited from the root `View` class. The `android:id` attribute is one example.

For descriptions of specific attributes, see the overview section of the `View` class documentation.

# Identifying a View

To uniquely identify a `View` and reference it from your code, you must give it an `id`. The `android:id` attribute lets you specify a unique `id`—a resource identifier for a `View`. For example:

```
android:id="@+id/button_count"
```

The `@+id/button_count` part of the attribute creates an `id` called `button_count` for a `Button` (a subclass of `View`). You use the plus (`+`) symbol to indicate that you are creating a new `id`.

# Referencing a View

To refer to an existing resource identifier, omit the plus (`+`) symbol. For example, to refer to a `View` by its `id` in *another* attribute, such as `android:layout_toLeftOf` (described in the next section) to control the position of a `View`, you would use:

```
android:layout_toLeftOf="@id/show_count"
```

In the attribute above, `@id/show_count` refers to the `View` with the resource identifier `show_count`. The attribute positions the element to be "to the left of" the `show_count` `View`.

# Positioning a View

Some layout-related positioning attributes are required for a `View` or a `ViewGroup`, and automatically appear when you add the `View` or `ViewGroup` to the XML layout.

## *LinearLayout positioning*

`LinearLayout` is required to have these attributes set:

- `android:layout_width`
- `android:layout_height`
- `android:orientation`

The `android:layout_width` and `android:layout_height` attributes can take one of three values:

- `match_parent` expands the UI element to fill its parent by width or height. When the LinearLayout is the root `ViewGroup`, it expands to the size of the device screen. For a UI element within a root `ViewGroup`, it expands to the size of the parent `ViewGroup`.
- `wrap_content` shrinks the UI element to the size of its content. If there is no content, the element becomes invisible.
- Use a fixed number of `dp` (density-independent pixels) to specify a fixed size, adjusted for the screen size of the device. For example, `16dp` means 16 density-independent pixels. Density-independent pixels and other dimensions are described in "Dimensions" in this chapter.

The `android:orientation` can be:

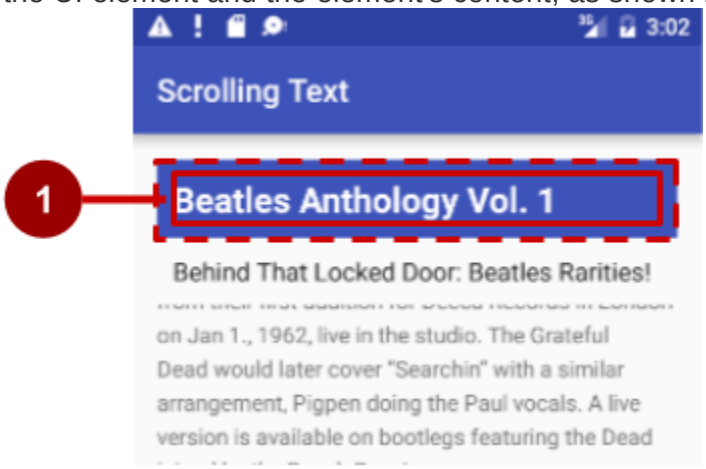
- `horizontal`: Views are arranged from left to right.
- `vertical`: Views are arranged from top to bottom.

Other layout-related attributes include:

- `android:layout_gravity`: This attribute is used with a UI element to control where the element is arranged within its parent. For example, the following attribute centers the UI element horizontally within the parent `ViewGroup`:

```
android:layout_gravity="center_horizontal"
```

- Padding is the space, measured in density-independent pixels, between the edges of the UI element and the element's content, as shown in the figure below.




In the figure above: (1) *Padding* is the space between the edges of the `TextView` (dashed lines) and the content of the `TextView` (solid line). Padding is not the same as *margin*, which is the space from the edge of the `View` to its parent. The size of a `View` includes its padding. The following are commonly used padding attributes:

- `android:padding`: Sets the padding of all four edges.
- `android:paddingTop`: Sets the padding of the top edge.
- `android:paddingBottom`: Sets the padding of the bottom edge.
- `android:paddingLeft`: Sets the padding of the left edge.
- `android:paddingRight`: Sets the padding of the right edge.
- `android:paddingStart`: Sets the padding of the start of the view, in pixels. Used in place of the padding attributes listed above, especially with views that are long and narrow.
- `android:paddingEnd`: Sets the padding of the end edge of the view, in pixels. Used along with `android:paddingStart`.

**Tip**: To see all of the XML attributes for a `LinearLayout`, see the Summary section of the `LinearLayout` class definition. Other root layouts, such as `RelativeLayout` and `AbsoluteLayout`, also list their XML attributes in the Summary sections.

### *RelativeLayout Positioning*

Another useful `Viewgroup` for layout is `RelativeLayout`, which you can use to position child `View` elements relative to each other or to the parent. The attributes you can use with `RelativeLayout` include the following:

- `android:layout_toLeftOf`: Positions the right edge of this `View` to the left of another `View` (identified by its `ID`).
- `android:layout_toRightOf`: Positions the left edge of this `View` to the right of another `View` (identified by its `ID`).
- `android:layout_centerHorizontal`: Centers this `View` horizontally within its parent.
- `android:layout_centerVertical`: Centers this `View` vertically within its parent.
- `android:layout_alignParentTop`: Positions the top edge of this `View` to match the top edge of the parent.
- `android:layout_alignParentBottom`: Positions the bottom edge of this `View` to match the bottom edge of the parent.

For a complete list of attributes for `View` and `View` subclass elements in a `RelativeLayout`, see `RelativeLayout.LayoutParams`.

## Style-related attributes

You specify style attributes to customize the appearance of a `View`.
A `View` that *doesn't* have style attributes, such as `android:textColor`, `android:textSize`, and `android:background`, takes on the styles defined in the app's theme.
The following are style-related attributes used in lesson on using the layout editor:

- `android:background`: Specifies a color or drawable resource to use as the background.
- `android:text`: Specifies text to display in the view.
- `android:textColor`: Specifies the text color.
- `android:textSize`: Specifies the text size.
- `android:textStyle`: Specifies the text style, such as `bold`.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Resource files

Resource files are a way of separating static values from code so that you don't have to change the code itself to change the values. You can store all the strings, layouts, dimensions, colors, styles, and menu text separately in resource files.

Resource files are stored in folders located in the `res` folder when viewing the Project > Android pane. These folders include:

- `drawable`: For images and icons
- `layout`: For layout resource files
- `menu`: For menu items
- `mipmap`: For pre-calculated, optimized collections of app icons used by the Launcher
- `values`: For colors, dimensions, strings, and styles (theme attributes)

The syntax to reference a resource in an XML layout is as follows:

`@`*package_name*`:`*resource_type*`/`*resource_name*

- *package_name* is the name of the package in which the resource is located. The package name is not required when you reference resources that are stored in the `res` folder of your project, because these resources are from the same package.
- *resource_type* is the `R` subclass for the resource type. See Resource Types for more about the resource types and how to reference them.
- *resource_name* is either the resource filename without the extension, or the `android:name` attribute value in the XML element.

For example, the following XML layout statement sets the `android:text` attribute to a `string` resource:

```
android:text="@string/button_label_toast"
```

- No *package_name* is included, because the resource is stored in the `strings.xml` file in the project.
- The *resource_type* is `string`.
- The *resource_name* is `button_label_toast.`

Another example: this XML layout statement sets the `android:background` attribute to a `color` resource, and since the resource is defined in the project (in the `colors.xml` file), the *package_name* is not specified:

```
android:background="@color/colorPrimary"
```

In the following example, the XML layout statement sets the `android:textColor` attribute to a `color` resource. However, the resource is not defined in the project but supplied by Android, so you need to specify the *package_name*, which is `android`, followed by a colon:

```
android:textColor="@android:color/white"
```

**Tip**: For more about accessing resources from code, see Accessing Resources. For Android color constants, see the Android standard R.color resources.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Values resource files

Keeping values such as strings and colors in separate resource files makes it easier to manage them, especially if you use them more than once in your layouts.

For example, it is essential to keep strings in a separate resource file for translating and localizing your app, so that you can create a string resource file for each language without changing your code. Resource files for images, colors, dimensions, and other attributes are handy for developing an app for different device screen sizes and orientations.

# Strings

String resources are located in the `strings.xml` file (inside **res > values** in the **Project > Android** pane). You can edit this file directly by opening it in the editor pane:

```
<resources>
    <string name="app_name">Hello Toast</string>
    <string name="button_label_count">Count</string>
    <string name="button_label_toast">Toast</string>
    <string name="count_initial_value">0</string>
</resources>
```

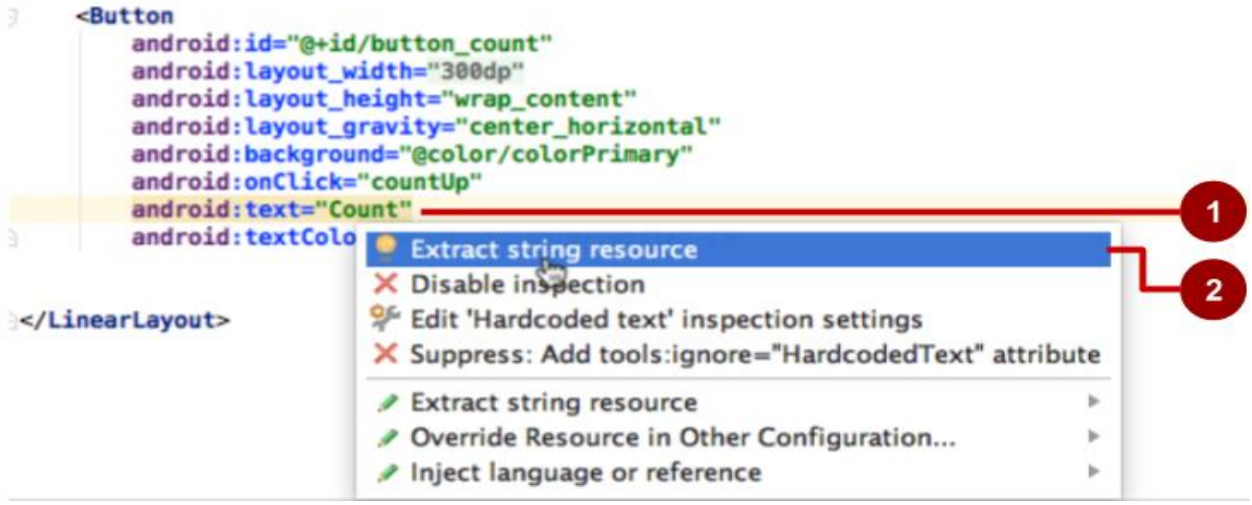
The `name` (for example, `button_label_count`) is the resource name you use in your XML code, as in the following attribute:

```
android:text="@string/button_label_count"
```

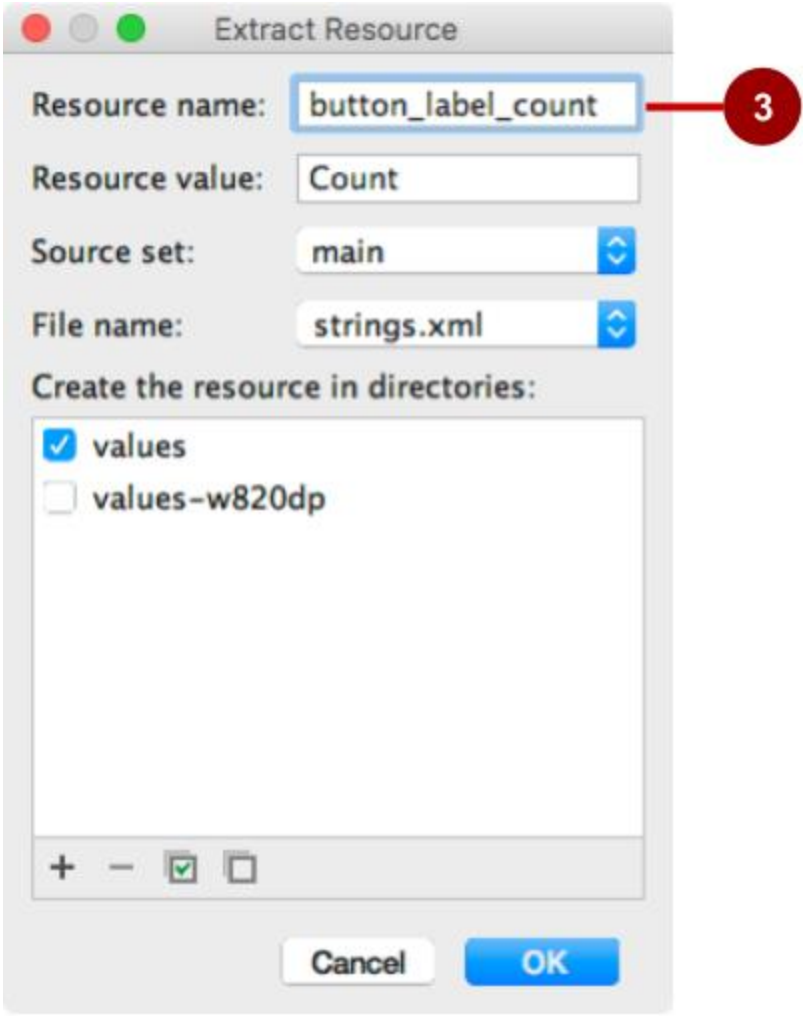
The string value of this `name` is the word (`Count`) enclosed within the `<string></string>` tags. (You don't use quotation marks unless the quotation marks are part of the string value.)

# Extracting strings to resources

You should also *extract* hard-coded strings in an XML layout file to string resources.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/




To extract a hard-coded string in an XML layout, follow these steps, as shown in the figure above:

1. Click the hard-coded string and press **Alt-Enter** in Windows, or **Option-Return** in Mac OS X.
2. Select **Extract string resource**.
3. Edit the **Resource name** for the string value.

You can then use the resource name in your XML code. Use the expression `"@string/resource_name"` (including quotation marks) to refer to the string resource:

```
android:text="@string/button_label_count"
```

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Colors

Color resources are located in the `colors.xml` file (inside **res > values** in the **Project > Android** pane). You can edit this file directly in the editor pane:

```
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="myBackgroundColor">#FFF043</color>
</resources>
```

The `name` (for example, `colorPrimary`) is the resource name you use in your XML code:

```
android:textColor="@color/colorPrimary"
```

The color value of this `name` is the hexadecimal color value (`#3F51B5`) enclosed within the `<color></color>` tags. The hexadecimal value specifies red, green, and blue (RGB) values. The value always begins with a pound (`#`) character, followed by the Alpha-Red-Green-Blue information. For example, the hexadecimal value for black is #000000, while the hexadecimal value for a variant of sky blue is #559fe3. Base color values are listed in the Color class documentation.

The `colorPrimary` color is one of the predefined base colors and is used for the app bar. In a production app, you could, for example, customize this to fit your brand. Using the base colors for other UI elements creates a uniform UI.

**Tip**: For the Material Design specification for Android colors, see Style and Using the Material Theme. For common color hexadecimal values, see Color Hex Color Codes. For Android color constants, see the Android standard R.color resources.

You can see a small block of the color choice in the left margin next to the color resource declaration in `colors.xml`, and also in the left margin next to the attribute that uses the resource name in the layout XML file.


**Tip**: To see the color in a popup, turn on the Autopopup documentation feature. Select **Preferences > Editor > General > Code Completion**, and select the "Autopopup documentation in (ms)" option. You can then hover your cursor over a color resource name to see the color.

# Dimensions

To make dimensions easier to manage, you should separate the dimensions from your code, especially if you need to adjust your layout for devices with different screen densities. Keeping dimensions separate from code also makes it easy to have consistent sizing for UI elements, and to change the size of multiple elements by changing one dimension resource.

Dimension resources are located in the `dimens.xml` file (inside **res > values** in the **Project > Android** pane). The `dimens.xml` file can actually be a folder holding more than one `dimens.xml` file—one for each device screen resolution. You can edit each `dimens.xml` file directly:

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="my_view_width">300dp</dimen>
    <dimen name="count_text_size">200sp</dimen>
    <dimen name="counter_height">300dp</dimen>
</resources>
```

The `name` (for example, `activity_horizontal_margin`) is the resource name you use in the XML code:

```
android:paddingLeft="@dimen/activity_horizontal_margin"
```

The value of this `name` is the measurement (`16dp`) enclosed within the `<dimen></dimen>` tags.
You can extract dimensions in the same way as strings:

1. Click the hard-coded dimension, and press **Alt-Enter** in Windows, or press **Option-Return** in Mac OS X.
2. Select **Extract dimension resource**.
3. Edit the **Resource name** for the dimension value.

Density-independent pixels (`dp`) are independent of screen resolution. For example, `10px` (10 fixed pixels) look a lot smaller on a higher resolution screen, but Android scales `10dp` (10 device-independent pixels) to look right on different resolution screens. Text sizes can also be set to look right on different resolution screens using *scaled-pixel* (`sp`) sizes.
**Tip**: For more information about `dp` and `sp` units, see Supporting Different Densities.

# Styles

A style is a resource that specifies common attributes such as height, padding, font color, font size, background color. Styles are meant for attributes that modify the look of the view.

Styles are defined in the `styles.xml` file (inside **res > values** in the **Project > Android** pane). You can edit this file directly. Styles are covered in a later chapter, along with the Material Design Specification.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## Other resource files

Android Studio defines other resources that are covered in other chapters:

- Images and icons: The `drawable` folder provides icon and image resources. If your app does not have a `drawable` folder, you can manually create it inside the `res` folder. For more information about drawable resources, see Drawable Resources in the App Resources section of the Android Developer's Guide.
- Optimized icons: The `mipmap` folder typically contains pre-calculated, optimized collections of app icons used by the Launcher. Expand the folder to see that versions of icons are stored as resources for different screen densities.
- Menus: You can use an XML resource file to define menu items and store them in your project in the `menu` folder. Menus are described in a later chapter.

# Responding to View clicks

A *click event* occurs when the user taps or clicks a clickable `View`, such as a `Button`, `ImageButton`, `ImageView`, or `FloatingActionButton`. When such an event occurs, your code performs an action. In order to make this pattern work, you have to:

- Write a Java method that performs the specific action you want the app to do when this event occurs. This method is typically referred to as an *event handler*.
- Associate this event-handler method to the `View`, so that the method executes when the event occurs.

## The onClick attribute

Android Studio provides a shortcut for setting up a clickable `View`, and for associating an event handler with the `View`: use the `android:onClick` attribute in the XML layout. For example, the following XML attribute sets a `Button` to be clickable, and sets `showToast()` as the event handler:

```
<Button
    android:id="@+id/button_toast"
    android:onClick="showToast"
```

When the user taps the `button_toast` `Button`, the button's `android:onClick` attribute calls the `showToast()` method. In order to work with the `android:onClick` attribute, the `showToast()` method must be `public` and return `void`. To know which `View` called the method, the `showToast()` method must require a `view` parameter.
Android Studio provides a shortcut for creating an event handler *stub* (a placeholder for a method that you can fill in later) in the code for the `Activity` associated with the XML layout.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

Follow these steps:

1. Inside the XML layout file (such as `activity_main.xml`), click the method name in the `android:onClick` attribute statement (`showToast` in the XML snippet above).
2. Press **Alt-Enter** in Windows or **Option-Return** in Mac OS X, and select **Create onClick event handler**.
3. Select the `Activity` associated with the layout file (such as **MainActivity**) and click **OK**. Android Studio creates a placeholder method stub in `MainActivity.java` as shown below.

```
public void showToast(View view) {
        // Do something in response to the button click.
}
```

## Updating a View

To update a `View`, for example to replace the text in a `TextView`, your code must first instantiate an object from the `View`. Your code can then update the object, which updates the screen.
To refer to the `View` in your code, use the `findViewById()` method of the `View` class, which looks for a `View` based on the resource `id`. For example, the following statement sets `mShowCount` to be the `TextView` in the layout with the resource id `show_count`:

```
mShowCount = (TextView) findViewById(R.id.show_count);
```

From this point on, your code can use `mShowCount` to represent the `TextView`, so that when you update `mShowCount`, the `TextView` is updated.
For example, when the following `Button` with the `android:onClick` attribute is tapped, `onClick` calls the `countUp()` method:

```
android:onClick="countUp"
```

You can implement `countUp()` to increment the count, convert the count to a string, and set the string as the text for the `mShowCount` object:

```
public void countUp(View view) {
        mCount++;
        if (mShowCount != null)
            mShowCount.setText(Integer.toString(mCount));
}
```

Since you had already associated `mShowCount` with the `TextView` for displaying the count, the `mShowCount.setText()` method updates the `TextView` on the screen.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Related practicals

The related practical lessons are:

- 1.2 Part A: Your first interactive UI
- 1.2 Part B: The layout editor

# Learn more

Android Studio documentation:

- Android Studio User Guide
- Image Asset Studio

Android developer documentation:

- UI Overview
- Build a UI with Layout Editor
- Build a Responsive UI with ConstraintLayout
- Layouts
- `View`
- `Button`
- `TextView`
- Android Input Events
- Context
- Common Layout Objects
- `Color`
- Android resources
- Android standard R.color resources
- Supporting Different Densities

Material Design:

- Style
- Using the Material Theme

Other:

- Codelabs: Using ConstraintLayout to design your views
- Hierarchy Viewer
- Color Hex Color Codes
- Vocabulary words and concepts glossary

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# 1.3: Text and scrolling views

**Contents:**



This chapter describes one of the most often used `View` subclasses in apps: the `TextView`, which shows textual content on the screen. A `TextView` can be used to show a message, a response from a database, or even entire magazine-style articles that users can scroll. This chapter also shows how you can create a scrolling view of text and other elements.

## TextView

One `View` subclass you may use often is the `TextView` class, which displays text on the screen. You can use `TextView` for a `View` of any size, from a single character or word to a full screen of text. You can add a resource `id` to the `TextView` in the layout, and control how the text appears using attributes in the layout file.
You can refer to a `TextView` in your Java code by using its resource `id` in order to update the text or its attributes from your code. If you want to allow users to edit the text, use `EditText`, a subclass of `TextView` that allows text input and editing. You learn all about `EditText` in another lesson.

### TextView attributes

You can use XML attributes for a `TextView` to control:

- Where the `TextView` is positioned in a layout (like any other view)
- How the `TextView` itself appears, such as with a background color
- What the text looks like within the `TextView`, such as the initial text and its style, size, and color

For example, to set the width, height, and initial text value of the view:

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

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

You can extract the text string into a string resource (perhaps called `hello_world`) that's easier to maintain for multiple-language versions of the app, or if you need to change the string in the future. After extracting the string, use the string resource name with `@string/` to specify the text:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello_world"
   <!-- more attributes -->
/>
```

In addition to `android:layout_width` and `android:layout_height` (which are required for a `TextView`), the most often used attributes with `TextView` are the following:

- `android:text`: Set the text to display.
- `android:textColor`: Set the color of the text. You can set the attribute to a color value, a predefined resource, or a theme. Color resources and themes are described in other chapters.
- `android:textAppearance`: The appearance of the text, including its color, typeface, style, and size. You set this attribute to a predefined style resource or theme that already defines these values.
- `android:textSize`: Set the text size (if not already set by `android:textAppearance`). Use `sp` (scaled-pixel) sizes such as `20sp` or `14.5sp`, or set the attribute to a predefined resource or theme.
- `android:textStyle`: Set the text style (if not already set by `android:textAppearance`). Use `normal`, `bold`, `italic`, or `bold|italic`.
- `android:typeface`: Set the text typeface (if not already set by `android:textAppearance`). Use `normal`, `sans`, `serif`, or `monospace`.
- `android:lineSpacingExtra`: Set extra spacing between lines of text. Use `sp` (scaled-pixel) or dp (device-independent pixel) sizes, or set the attribute to a predefined resource or theme.
- `android:autoLink`: Controls whether links such as URLs and email addresses are automatically found and converted to clickable (touchable) links.

Use one of the following with `android:autoLink`:

- `none`: Match no patterns (default).
- `web`: Match web URLs.
- `email`: Match email addresses.
- `phone`: Match phone numbers.
- `map`: Match map addresses.
- `all`: Match all patterns (equivalent to web|email|phone|map).

For example, to set the attribute to match web URLs, use `android:autoLink="web"`.

# Using embedded tags in text

In an app that accesses magazine or newspaper articles, the articles that appear would probably come from an online source or might be saved in advance in a database on the device. You can also create text as a single long string in the strings.xml resource.

In either case, the text may contain embedded HTML tags or other text formatting codes. To properly display in a text view, text must be formatted following these rules:

- Enter **\n** to represent the end of a line, and another **\n** to represent a blank line. You need to add end-of-line characters to keep paragraphs from running into each other.
- If you have an apostrophe (`'`) in your text, you must *escape* it by preceding it with a backslash (**\'**). If you have a double-quote in your text, you must also escape it (**\"**). You must also escape any other non-ASCII characters. See the "Formatting and Styling" section of String Resources for more details.
- Enter the HTML and **</b>** tags around words that should be in bold.
- Enter the HTML and **</i>** tags around words that should be in italics. Note, however, that if you use curled apostrophes within an italic phrase, you should replace them with straight apostrophes.
- You can combine bold and italics by combining the tags, as in ***...*** words...**</i></b>**. Other HTML tags are ignored.
- To create a long string of text in the `strings.xml` file, enclose the entire text within `<string name="`*your_string_name*`"></string>` (*your_string_name* is the name you provide the string resource, such as `article_text`).
- As you enter or paste text in the `strings.xml` file, the text lines don't wrap around to the next line—they extend beyond the right margin. This is the correct behavior—each new line of text starting at the left margin represents an entire paragraph.

**Tip:** If you want to see the text wrapped in strings.xml, you can press Return to enter hard line endings, or format the text first in a text editor with hard line endings. The endings will not be displayed on the screen.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## Referring to a TextView in code

To refer to a `TextView` in your Java code, use its resource `id`. For example, to update a `TextView` with new text, you would:

1. Find the `TextView` and assign it to a variable. You use the `findViewById()` method of the `View` class, and refer to the view you want to find using this format:

```
R.id.view_id
```

In which *view_id* is the resource identifier for the view (such as `show_count`) :

```
mShowCount = (TextView) findViewById(R.id.show_count);
```

3. After retrieving the `View` as a `TextView` member variable, you can then set the text to new text (in this case, `mCount_text`) using the `setText()` method of the `TextView` class:

```
mShowCount.setText(mCount_text);
```

# Scrolling views

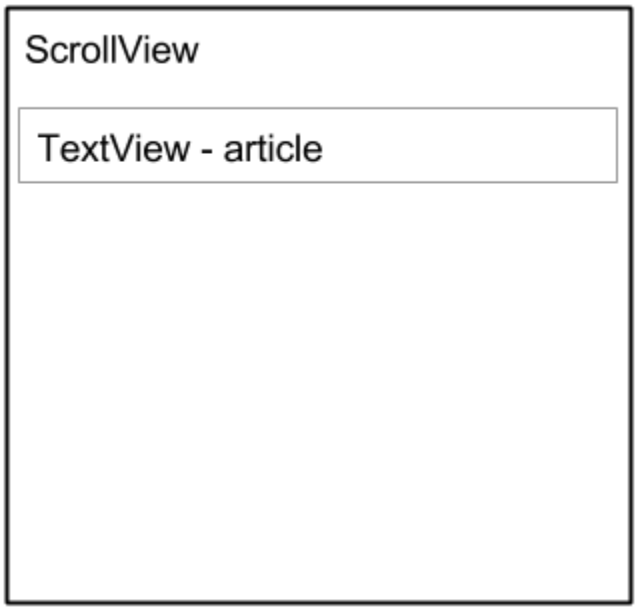
If the information you want to show in your app is larger than the device's display, you can create a *scrolling view* that the user can scroll vertically by swiping up or down, or horizontally by swiping right or left.

You would typically use a scrolling view for news stories, articles, or any lengthy text that doesn't completely fit on the display. You can also use a scrolling view to combine views (such as a `TextView` and a `Button`) within a scrolling view.

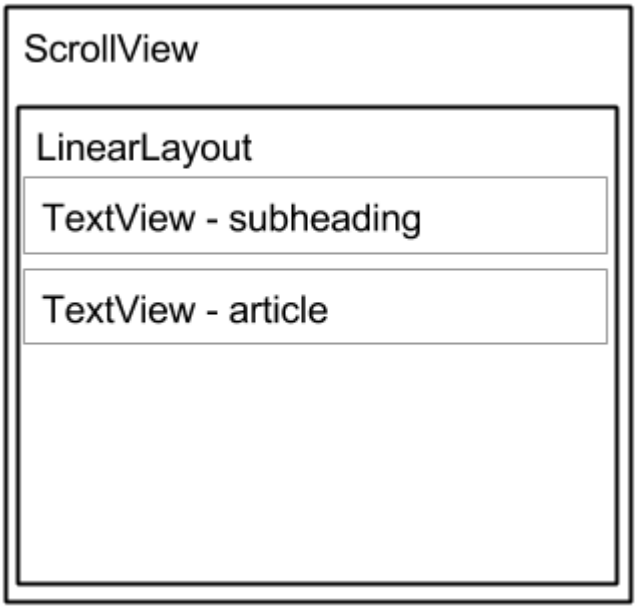
## Creating a layout with a ScrollView

The `ScrollView` class provides the layout for a vertical scrolling view. (For horizontal scrolling, you would use `HorizontalScrollView`.) `ScrollView` is a subclass of `FrameLayout`, which means that you can place only *one* `View` as a child within it; that

child contains the entire contents to scroll.




Even though you can place only one child `View` inside a `ScrollView`, the child `View` can be a `ViewGroup` with a hierarchy of child `View` elements, such as a `LinearLayout`. A good choice for a `View` within a `ScrollView` is a `LinearLayout` that is arranged in a vertical orientation.



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

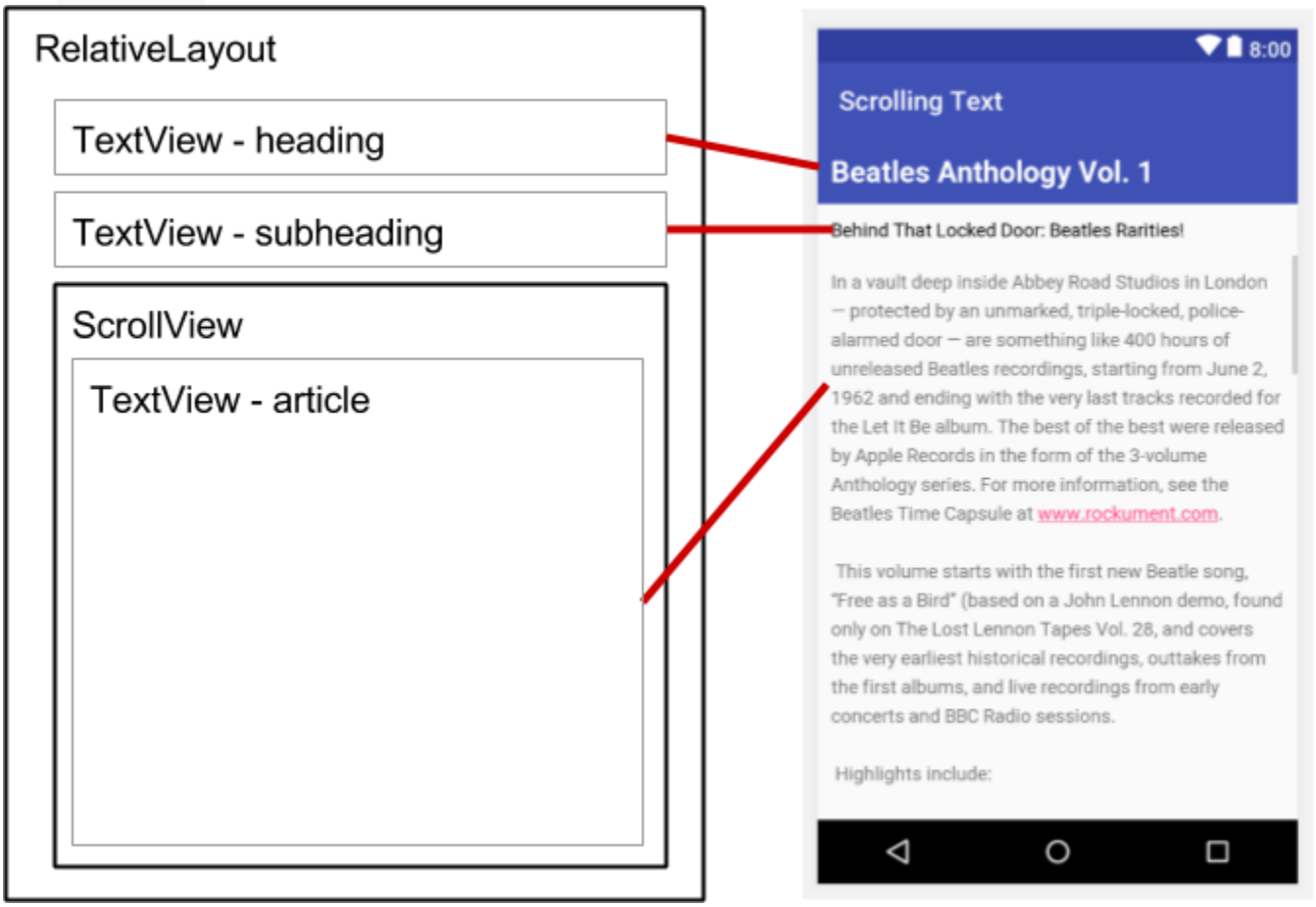
## ScrollView and performance

All of the contents of a `ScrollView` (such as a `ViewGroup` with `View` elements) occupy memory and the view hierarchy even if portions are not displayed on screen. This makes `ScrollView` useful for smoothly scrolling pages of free-form text, because the text is already in memory. However, a ScrollView with a `ViewGroup` with `View` elements can use up a lot of memory, which can affect the performance of the rest of your app. Using nested instances of `LinearLayout` can also lead to an excessively deep view hierarchy, which can slow down performance. Nesting several instances of `LinearLayout` that use the `android:layout_weight` attribute can be especially expensive as each child `View` needs to be measured twice. Consider using flatter layouts such as `RelativeLayout` or `GridLayout` to improve performance.
Complex layouts with `ScrollView` may suffer performance issues, especially with images. We recommend that you *not* use images within a `ScrollView`. To display long lists of items, or images, consider using a `RecyclerView`, which is covered in another lesson.

## ScrollView with a TextView

To display a scrollable magazine article on the screen, you might use a `RelativeLayout` that includes a separate `TextView` for the article heading, another for the article subheading, and a third `TextView` for the scrolling article text (see the figure below), set within a `ScrollView`. The only part of the screen that would scroll would be

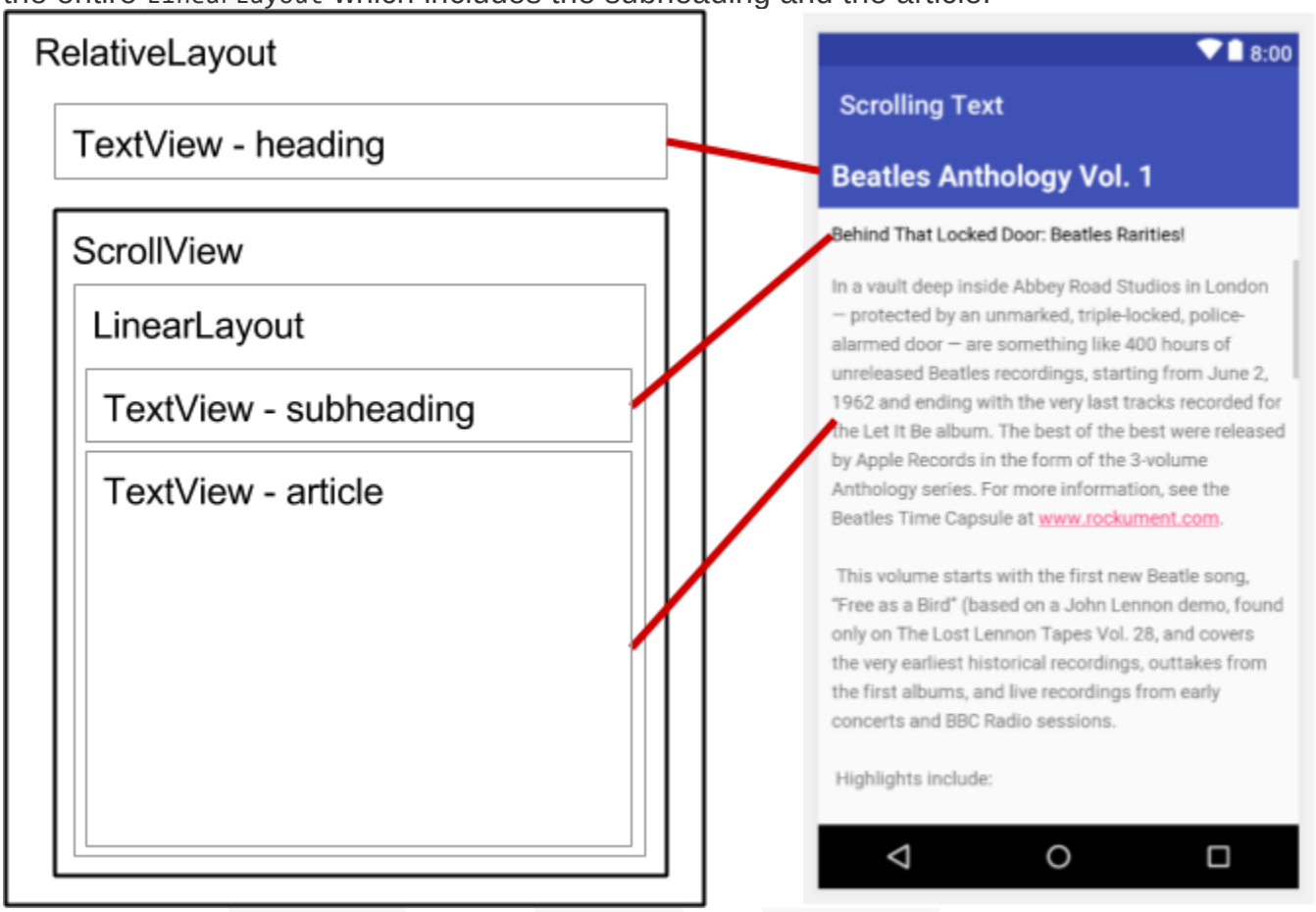
the `ScrollView` with the article text.




## ScrollView with a LinearLayout

A `ScrollView` can contain only one child `View`; however, that `View` can be a `ViewGroup` that contains several `View` elements, such as `LinearLayout`. You can *nest* a `ViewGroup` such as `LinearLayout` *within* the `ScrollView`, thereby scrolling everything that is inside the `LinearLayout`.
For example, if you want the subheading of an article to scroll along with the article even if they are separate `TextView` elements, add a `LinearLayout` to the `ScrollView` as a single child `View` as shown in the figure below, and then move the `TextView` subheading and article elements into the `LinearLayout`. The user scrolls

the entire `LinearLayout` which includes the subheading and the article.




When adding a `LinearLayout` inside a `ScrollView`, use `match_parent` for the `LinearLayout` `android:layout_width` attribute to match the width of the parent `ScrollView`, and use `wrap_content` for the `LinearLayout` `android:layout_height` attribute to make it only large enough to enclose its contents.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

Since `ScrollView` only supports vertical scrolling, you must set the `LinearLayout` orientation attribute to vertical (`android:orientation="vertical"`), so that the entire `LinearLayout` will scroll vertically. For example, the following XML layout scrolls the `article TextView` along with the `article_subheading TextView`:

```
<ScrollView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@id/article_heading">

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">

      <TextView
         android:id="@+id/article_subheading"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:padding="@dimen/padding_regular"
         android:text="@string/article_subtitle"
         android:textAppearance=
                      "@android:style/TextAppearance.DeviceDefault" />

      <TextView
         android:id="@+id/article"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:autoLink="web"
         android:lineSpacingExtra="@dimen/line_spacing"
         android:padding="@dimen/padding_regular"
         android:text="@string/article_text" />
   </LinearLayout>

</ScrollView>
```

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Related practical

The related practical for this concept chapter is 1.3: Text and scrolling views.

# Learn more

Android Studio documentation:

- Android Studio User Guide

Android developer documentation:

- `ScrollView`
- `LinearLayout`
- `RelativeLayout`
- `View`
- `Button`
- `TextView`
- String Resources
- Relative Layout

**Other:**

- Android Developers Blog: Linkify your Text!
- Codepath: Working with a TextView

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# 1.4: Resources to help you learn

**Contents:**



This chapter describes resources available for Android developers, and how to use them.

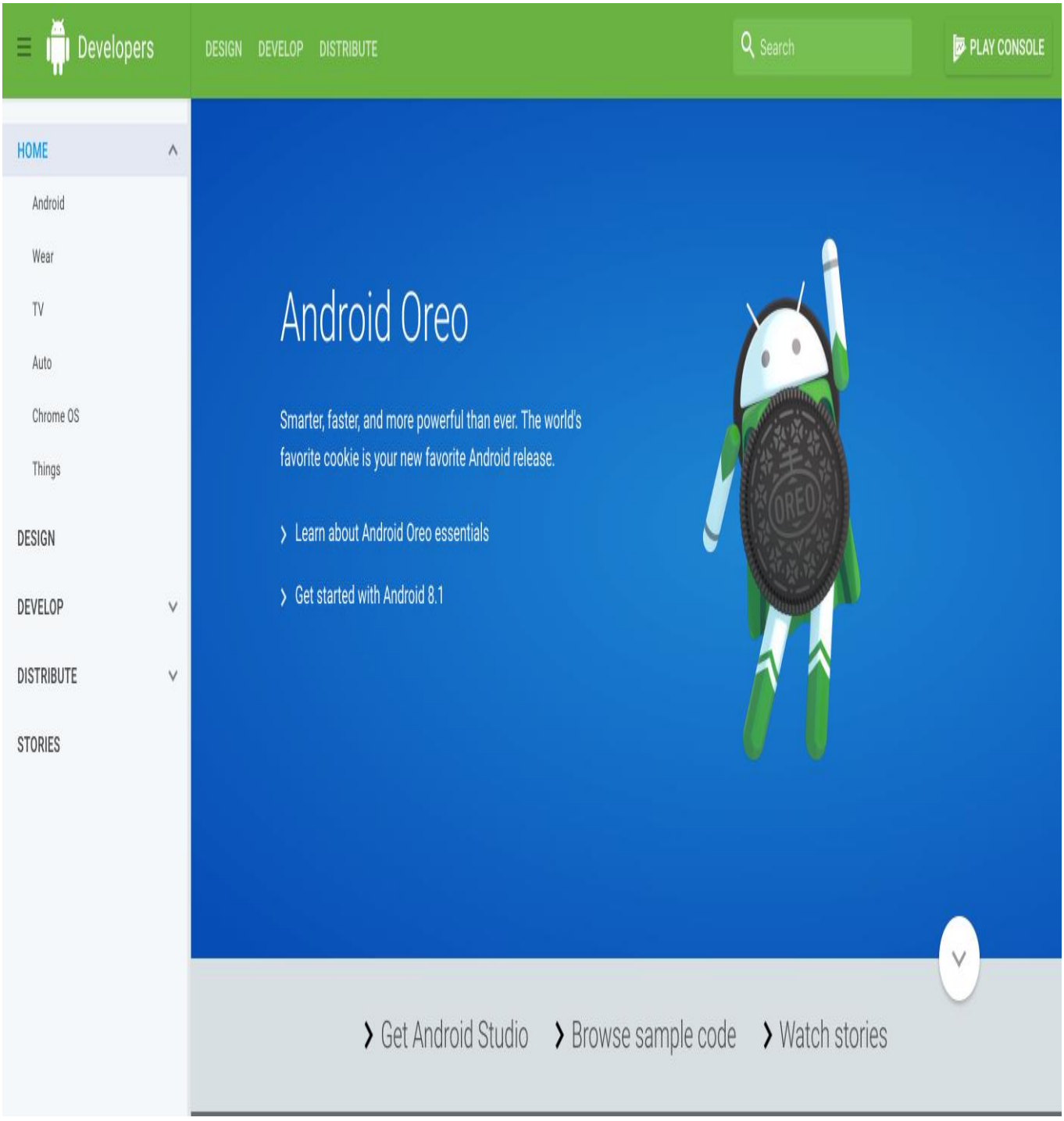
## Exploring Android developer documentation

The best place to learn about Android development and to keep informed about the newest Android development tools is to browse the official Android developer documentation.

developer.android.com

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## The home page



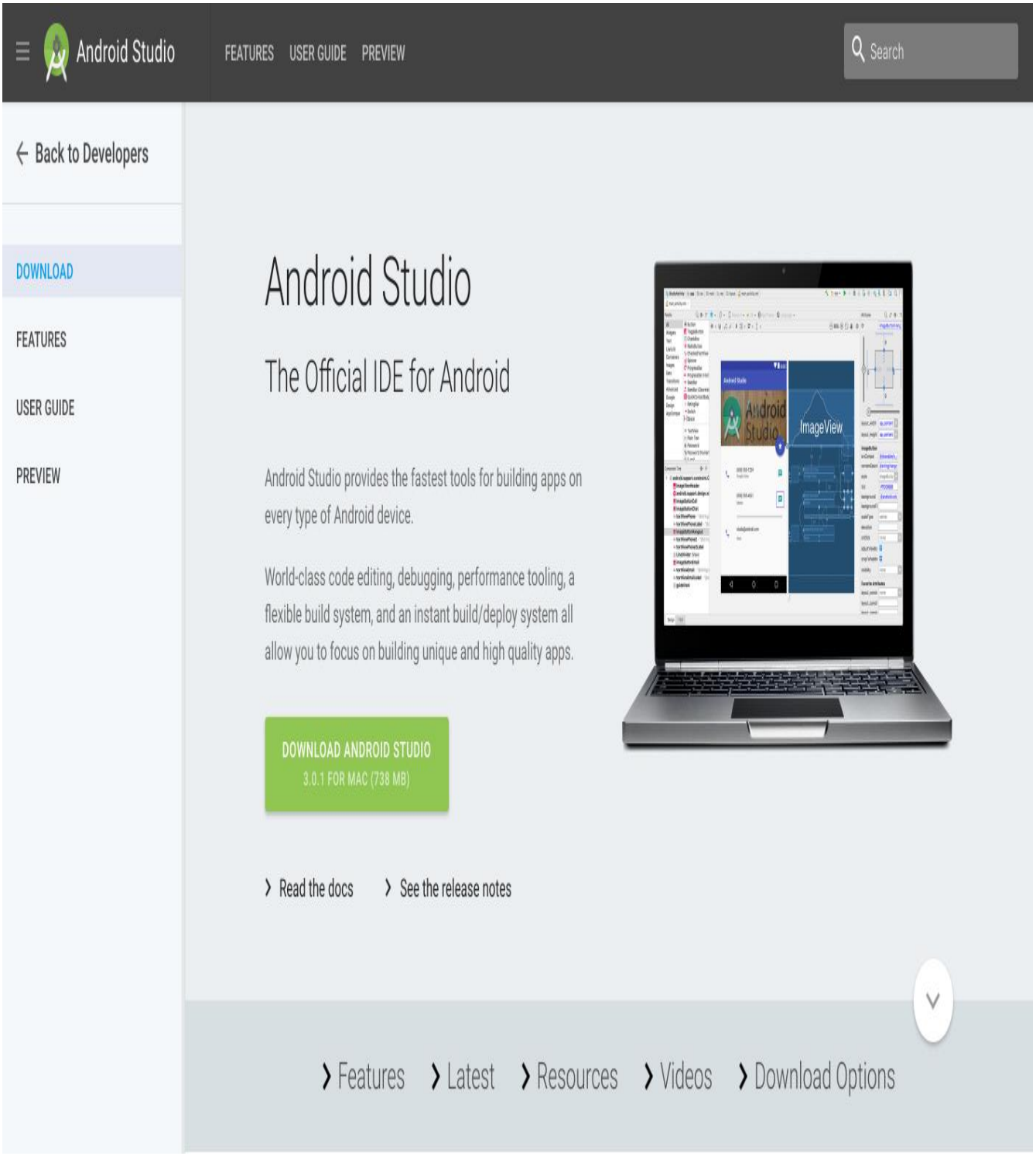
https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

This page contains a wealth of information kept current by Google. To start exploring, click the following links along the bottom of the page:

- **Get Android Studio**: Download Android Studio, the official integrated development environment (IDE) for building Android apps.
- **Browse sample code**: Browse the sample code library in GitHub to learn how to build different components for your apps. Each sample is a fully functioning Android app. The GitHub page lets you browse the resources and source files, and clone or download the app project. For more sample code, see "Exploring code samples in the Android SDK" in this chapter.
- **Watch stories**: Learn about other Android developers, their apps, and their successes with Android and Google Play. The page offers videos and articles with the newest stories about Android development, such as how developers improved their users' experiences, and how to increase user engagement with apps.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

## Android Studio page



https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

After clicking **Get Android Studio** on the home page, the Android Studio page, shown above, appears with the following useful links:

- **Download Android Studio**: Download Android Studio for the computer operating system you are currently using.
- **Read the docs**: Browse the Android Studio documentation.
- **See the release notes**: Read the release notes for the newest version of Android Studio.
- **Features**: Learn about the features of the newest version of Android Studio.
- **Latest**: Read news about Android Studio.
- **Resources**: Read articles about using Android Studio, including a basic introduction.
- **Videos**: Watch video tutorials about using Android Studio.
- **Download Options**: Download a version of Android Studio for a different operating system than the one you are using.

# Android Studio documentation

The following are links into the Android Studio documentation that are useful for this training:

- Meet Android Studio
- Developer Workflow Basics
- Projects Overview
- Create App Icons with Image Asset Studio
- Add Multi-Density Vector Graphics
- Create and Manage Virtual Devices
- Measure App Performance with Android Profiler
- Debug Your App
- Configure Your Build
- Sign Your App

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Design

The **Design** link in the navigation drawer (left column) of the home page gives you access to Material Design, which is a conceptual design philosophy that outlines how apps should look and work on mobile devices. Use the following links to learn more:

- **Introducing material design**: An introduction to the material design philosophy.
- **Downloads for designers**: Download color palettes for compatibility with the material design specification.
- **Articles**: Read articles and news about Android design.

Scroll down the Design page for links to resources such as videos, templates, font, and color palettes. The following are links into the Design section that are useful for this training:

- Material Design Guidelines
- Style
- Using the Material Theme
- Components - Buttons
- Dialogs design guide
- Gestures design guide
- Notification Design Guide
- Icons and other downloadable resources
- Design - Patterns - Navigation
- Drawable Resource Guide
- Styles and Themes Guide
- Settings
- Material Palette Generator
- Android standard R.color resources

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Develop

The **Develop** link in the navigation drawer (left column) of the home page provides a wealth of application programming interface (API) information, reference documentation, tutorials, tool guides, and code samples.

The following are popular links into the Develop section that are useful for this training:

- Introduction to Android
- Vocabulary Glossary
- Platform Architecture
- Android Application Fundamentals
- UI Overview
- Platform Versions
- Android Support Library
- Working with System Permissions

The following are articles that describe best development practices:

- Supporting Different Platform Versions
- Supporting Multiple Screens
- Supporting Different Densities
- Best Practices for Interaction and Engagement
- Best Practices for User Interface
- Best Practices for Testing
- Providing Resources
- Optimizing Downloads for Efficient Network Access Guide
- Best Practices for App Permissions

The following are useful articles and training guides:

- Starting Another Activity
- Specifying the Input Method Type
- Handling Keyboard Input
- Adding the App Bar
- Using Touch Gestures
- Creating Lists and Cards
- Getting Started with Testing
- Managing the Activity Lifecycle
- Connecting to the Network
- Managing Network Usage
- Manipulating Broadcast Receivers On Demand
- Scheduling Repeating Alarms
- Transferring Data Without Draining the Battery
- Saving Files
- Saving Key-Value Sets
- Saving Data in SQL Databases
- Configuring Auto Backup for Apps
- Working with System Permissions

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

The following are general topics covered in this training:

- App Resources
- Styles and Themes
- Layouts
- Menus
- Intents and Intent Filters
- Processes and threads
- Loaders
- Services
- Notifications
- Storage Options
- Localizing with Resources
- Content Providers
- Cursors
- Backing up App Data to the Cloud
- Settings
- System Permissions

# Distribute

The **Distribute** link in the navigation drawer (left column) provides information about everything that happens *after* you've written your app: putting it on Google Play, Google's digital distribution system for apps developed with the Android SDK. Use the Google Play Console to grow your user base and start earning money. You can accept the Developer Agreement, pay the registration fee, and complete your account details in order to join the Google Play developer program.

See also:

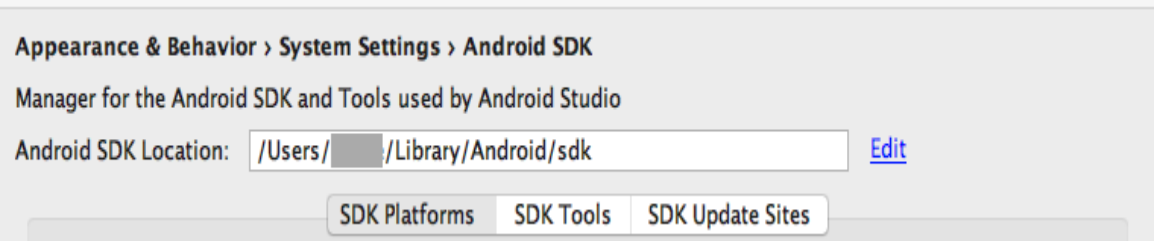
- Essentials for a Successful App
- Launch Checklist

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Installing offline documentation

To access to documentation even when you are not connected to the internet, install the Software Development Kit (SDK) documentation using the SDK Manager. Follow these steps:

1. Select **Tools > Android > SDK Manager**.
2. In the left column, click **Android SDK**.
3. Select and copy the path for the Android SDK Location at the top of the screen, as you will need it to locate the documentation on your computer:



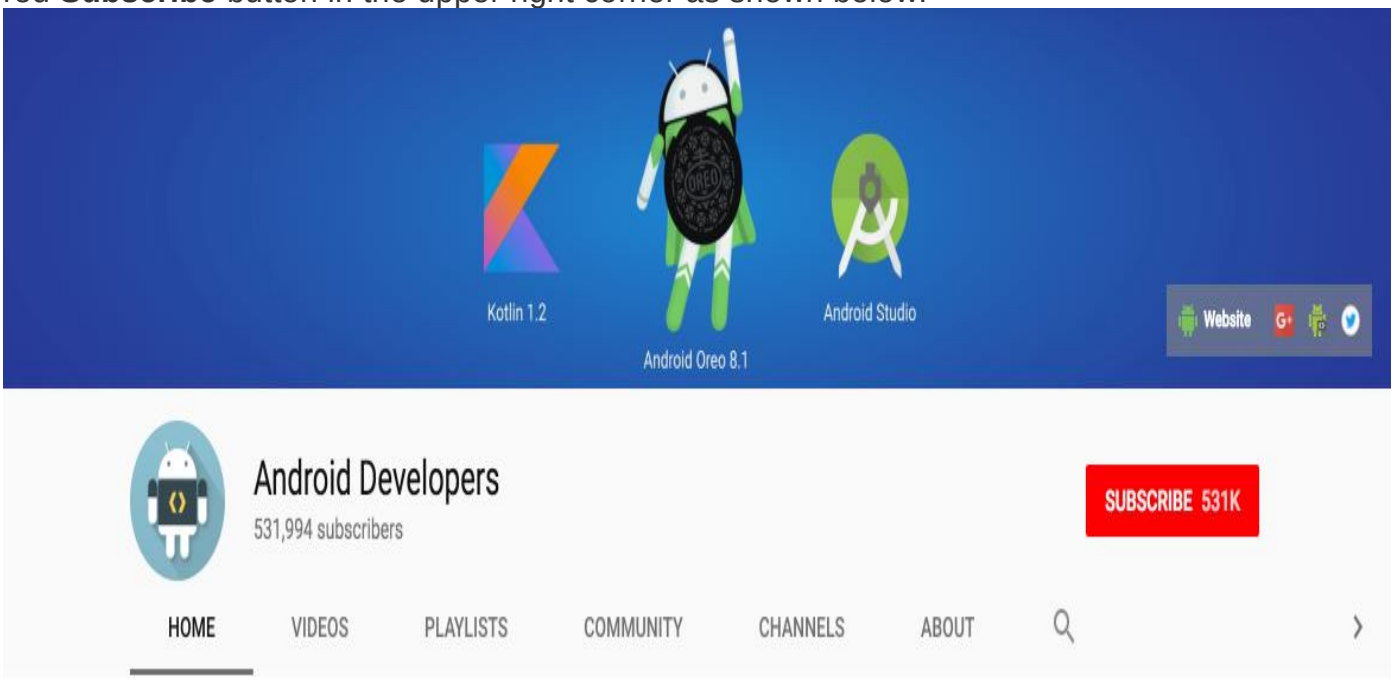

4. Click the **SDK Tools** tab. You can install additional SDK Tools that are not installed by default, as well as an offline version of the Android developer documentation.

5. Click the checkbox for "Documentation for Android SDK" if it is not already installed, and click **Apply**.
6. When the installation finishes, click **Finish**.
7. Navigate to the **sdk** directory you copied above, and open the **docs** directory.
8. Find **index.html** and open it.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Watching developer videos

In addition to the Android documentation, the Android Developers YouTube channel is a great source of tutorials and tips. You can subscribe to the channel to receive notifications of new videos by email. To subscribe, click the red **Subscribe** button in the upper right corner as shown below.




The following are popular videos referred to in this training:

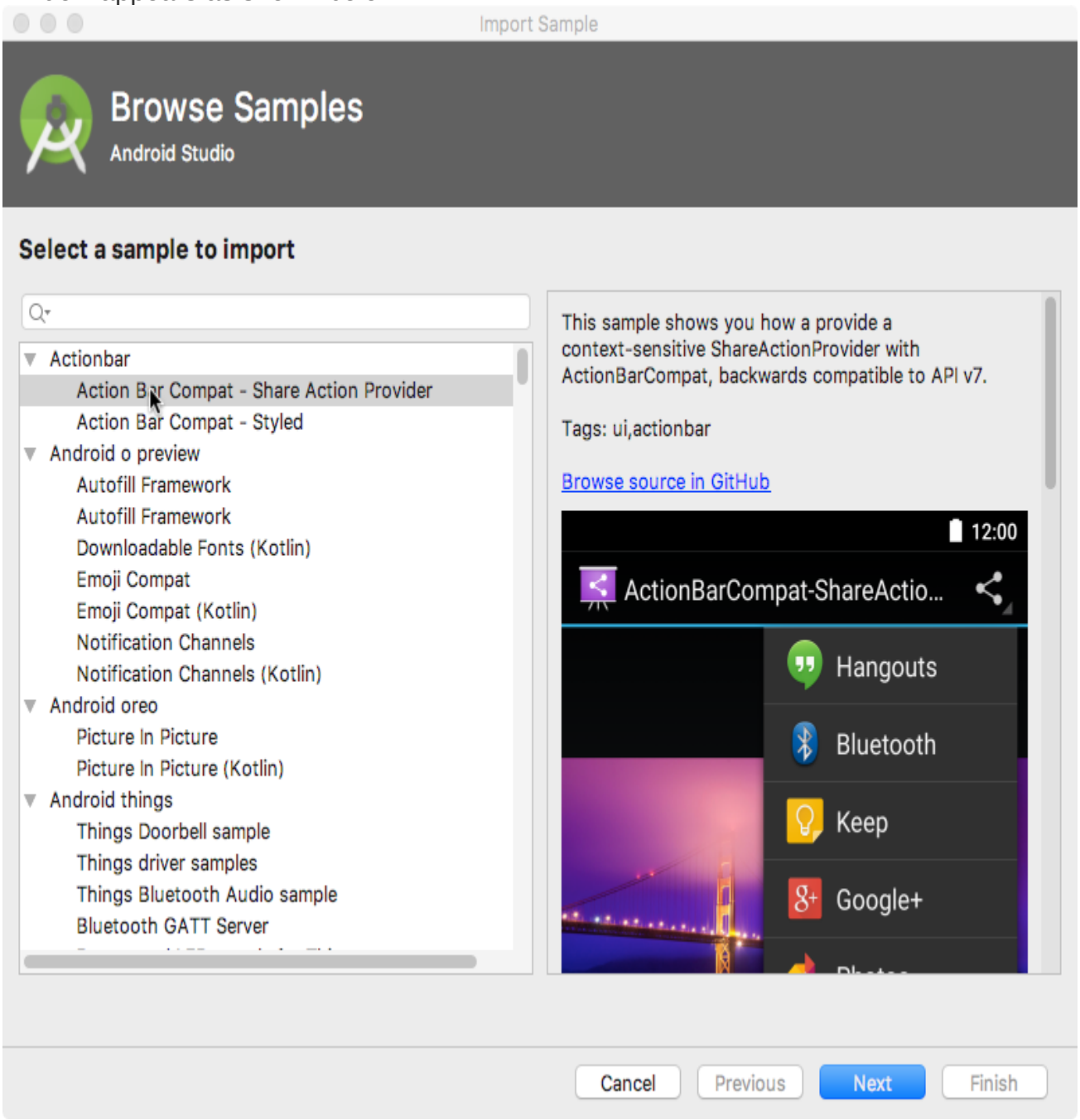
- Debugging and Testing in Android Studio
- Android Testing Support - Android Testing Patterns #1
- Android Testing Support - Android Testing Patterns #2
- Android Testing Support - Android Testing Patterns #3
- Threading Performance 101
- Good AsyncTask Hunting
- Scheduling Alarms Presentation
- RecyclerView Animations and Behind the Scenes (Android Dev Summit 2015)
- Android Application Architecture: The Next Billion Users
- Android Performance Patterns Playlist

In addition, Udacity offers online Android development courses.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Exploring code samples in the Android SDK

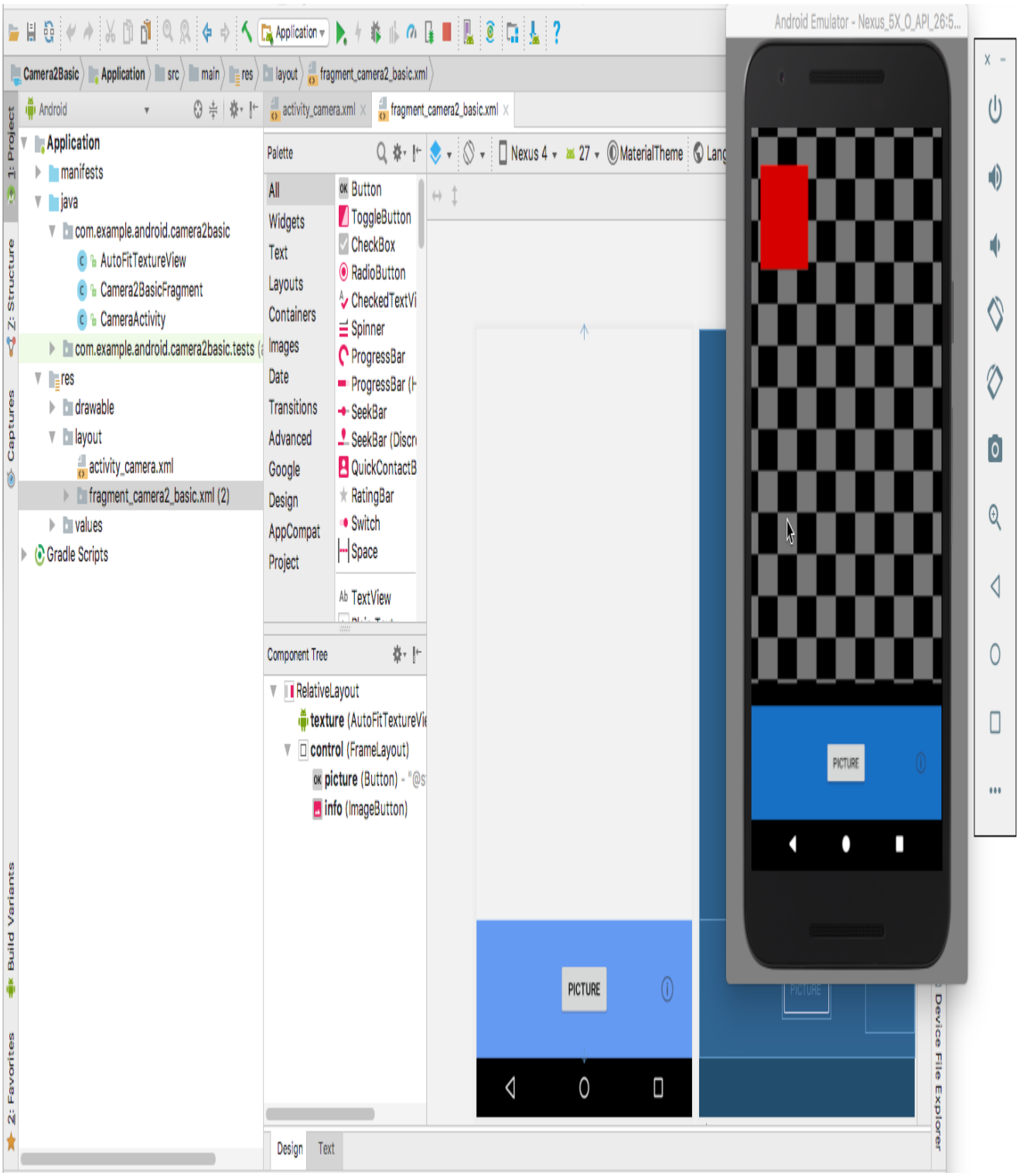
You can explore hundreds of code samples directly in Android Studio. Select **Import an Android code sample** from the Android Studio welcome screen, or select **File > New > Import Sample** if you have already opened a project. The Browse Samples window appears as shown below.




Select a sample (such as **Camera2Basic**) and click **Next**. Accept or edit the Application name and Project location, and click **Finish**. The app project appears as shown below, and you can run the app in the emulator provided with Android Studio,

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

or on a connected device.



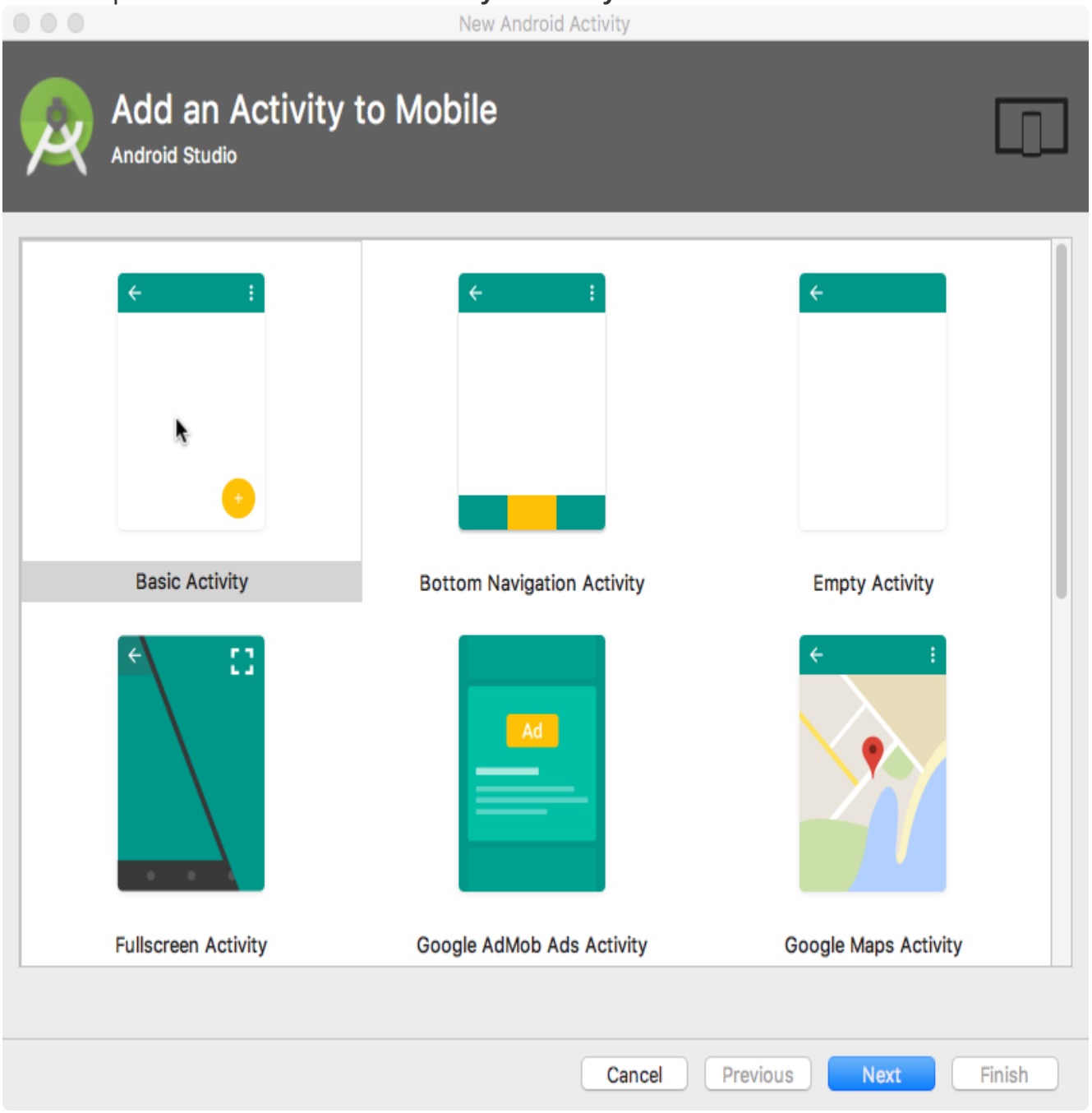

**Note:** The samples are meant to be a starting point for further development. We encourage you to design and build your own ideas into them.

https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/

# Using Activity templates

Android Studio provides templates for common and recommended activity designs. Using templates saves time, and helps you follow best practices for developing activities.

Each template incorporates an skeleton activity and user interface. You select an activity template for the main activity when starting an app project. You can also add an activity template to an existing project. Right-click the **java** folder in **Project > Android** pane and select **New > Activity > Gallery**.



# Browsing the Android developer blog

The Android Developers Blog provides a wealth of articles on Android development.

The following are popular blog posts:

- Welcoming Android 8.1 Oreo and Android Oreo (Go edition)
- Android Studio 3.0
- Quick Boot & the Top Features in the Android Emulator
- Double Stuffed Security in Android Oreo
- Making it safer to get apps on Android O
- Connecting your App to a Wi-Fi Device
- Linkify your Text!
- Holo Everywhere
- 5 Tips to help you improve game-as-a-service monetization

# Other sources of information

Google and third parties offer a wide variety of helpful tips and techniques for Android development. The following are sources of information referenced by this training:

**Google Developer Training**: Whether you're new to programming or an experienced developer, Google offers a range of online courses to teach you Android development, from getting started to optimizing app performance. Click the **Android** tab at the top of the page.

**Google I/O Codelabs**: Google Developers Codelabs provide a guided hands-on coding experience on a number of topics. Most codelabs will step you through the process of building a small app, or adding a new feature to an existing app. Select **Android** from the **Category** drop-down menu on the right side of the page.

**Android Testing Codelab**: This codelab shows you how to get started with testing for Android, including testing integration in Android Studio, unit testing, hermetic testing, functional user interface testing, and the Espresso testing framework.

**Google Testing Blog**: This blog is focused on testing code. Blog posts referred to in the training include:

- Android UI Automated Testing
- Test Sizes

**Stack Overflow**: Stack Overflow is a community of millions of programmers helping each other. If you run into a problem, chances are someone else has already posted an answer on this forum. Examples referred to in the training include:

- How to assert inside a RecyclerView in Espresso?
- How do I Add A Fragment to a Custom Navigation Drawer Template?
- How do you create Preference Activity and Preference Fragment on Android?
- How to use SharedPreferences in Android to store, fetch and edit values
- How to populate AlertDialog from Arraylist?
- onSavedInstanceState vs. SharedPreferences

**Google on GitHub**: GitHub is a Git repository hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. Git is a widely used version control system for software development. The following are hosted within GitHub and referred to in this training:

- Android Testing Samples
- Espresso cheat sheet
- Roman Nurik's Android Asset Studio
- Source code for exercises on GitHub

Miscellaneous sources of information referred to in this training:

- Codepath: Working with a TextView
- SQLite.org: Full description of the Query Language
- Atomic Object: "Espresso – Testing RecyclerViews at Specific Positions"

**Google search**: Enter a question into the Google search box, prefaced by "Android" to narrow your search. The Google search engine will collect relevant results from all of these resources. For example:

- *"What is the most popular Android OS version in India?"* This question collects results about Android market share, including the Dashboards page that provides an overview of device characteristics and platform versions that are active in the Android ecosystem.
- *"Android Settings Activity"* collects various articles about the Settings Activity including the Settings topic page, the `PreferenceActivity` class, and Stack Overflow's How do you create Preference Activity and Preference Fragment on Android?
- *"Android TextView"* collects information about text views including the `TextView` class, the `View` class, the Layouts topic page, and code samples from various sources.
- Preface any search with *"Android"* to narrow your search to Android-related topics. For example, you can search for any Android class description, such as *"Android TextView"* or *"Android activity"*.

# Related practical

The related practical for this concept chapter is 1.4: Available resources.