

**TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN**



**GIÁO TRÌNH
THỰC HÀNH PHÁT TRIỂN ỨNG DỤNG CHO
THIẾT BỊ DI ĐỘNG**

Hà Nội, 2.2025

Unit 1: Getting Started

Trong bài học này

Bài 1: Xây dựng ứng dụng đầu tiên của bạn

1.1 : Android Studio và Hello World

1.2 A: Giao diện người dùng tương tác đầu tiên của bạn

1.2 B: Trình chỉnh sửa bộ cục

1.3 : Chế độ xem văn bản và cuộn

1.4 : Tài nguyên có sẵn

Bài học 2: Activities

2.1 : hoạt động và ý nghĩa

2.2 : Vòng đời và trạng thái

2.3 : Nỗ lực ngầm

Lesson 3: Kiểm tra, gỡ lỗi và sử dụng thư viện hỗ trợ

3.1 : Trình gỡ lỗi

3.2 : Kiểm tra đơn vị

3.3 : Thư viện hỗ trợ

Bài 1.1: Android Studio và Hello World

Giới thiệu

Trong thực tế này, bạn sẽ tìm hiểu cách cài đặt Android Studio, môi trường phát triển Android. Bạn cũng tạo và chạy ứng dụng Android đầu tiên của mình, Hello World, trên trình giả lập và trên thiết bị thực.

Những điều bạn nên biết

Bạn sẽ có thể:

- Hiểu quy trình phát triển phần mềm chung cho các ứng dụng hướng đối tượng bằng IDE (môi trường phát triển tích hợp) chẳng hạn như Android Studio.
- Chứng minh rằng bạn có ít nhất 1-3 năm kinh nghiệm trong lập trình hướng đối tượng, với một số trong số đó tập trung vào ngôn ngữ lập trình Java. (Những thực tế này sẽ không giải thích lập trình hướng đối tượng hoặc ngôn ngữ Java.)

Những gì bạn cần

- Máy tính chạy Windows hoặc Linux hoặc máy Mac chạy macOS. Xem [trang tải xuống Android Studio](#) để biết các yêu cầu hệ thống cập nhật.
- Truy cập Internet hoặc một cách thay thế để tải các bản cài đặt Android Studio và Java mới nhất vào máy tính của bạn.

Những gì bạn sẽ học

1. Cách cài đặt và sử dụng Android Studio IDE.
2. Cách sử dụng quy trình phát triển để xây dựng ứng dụng Android.
3. Cách tạo dự án Android từ mẫu.
4. Cách thêm thông báo nhật ký vào ứng dụng của bạn cho mục đích gỡ lỗi.

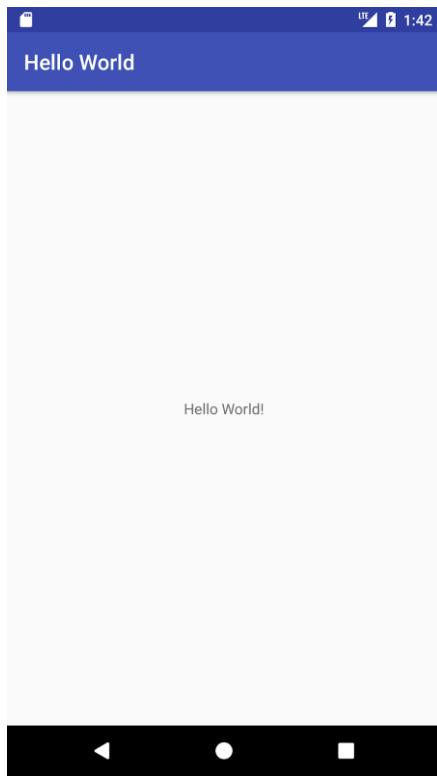
Bạn sẽ làm gì

1. Cài đặt môi trường phát triển Android Studio.
2. Tạo trình mô phỏng (thiết bị ảo) để chạy ứng dụng trên máy tính.
3. Tạo và chạy ứng dụng Hello World trên các thiết bị ảo và vật lý.
4. Khám phá bộ cục dự án.
5. Tạo và xem thông báo nhật ký từ ứng dụng của bạn.
6. Khám phá tệp AndroidManifest.xml .

Tổng quan về ứng dụng

Sau khi cài đặt thành công Android Studio, bạn sẽ tạo một dự án mới cho ứng dụng Hello World từ một mẫu. Ứng dụng đơn giản này hiển thị chuỗi "Hello World" trên màn hình của thiết bị ảo hoặc vật lý Android.

Đây là ứng dụng hoàn thành sẽ trông như thế nào:



Nhiệm vụ 1: Cài đặt Android Studio

Android Studio cung cấp một môi trường phát triển tích hợp (IDE) hoàn chỉnh bao gồm trình chỉnh sửa mã nâng cao và một bộ mẫu ứng dụng. Ngoài ra, nó còn chứa các công cụ để phát triển, gỡ lỗi, thử nghiệm và hiệu suất giúp phát triển ứng dụng nhanh hơn và dễ dàng hơn. Bạn có thể kiểm tra ứng dụng của mình bằng nhiều trình mô phỏng được định cấu hình sẵn hoặc trên thiết bị di động của riêng mình, tạo ứng dụng chính thức và phát hành trên cửa hàng Google Play.

Lưu ý: Android Studio liên tục được cải thiện. Để biết thông tin mới nhất về yêu cầu hệ thống và hướng dẫn cài đặt, hãy xem [Android Studio](#).

Android Studio có sẵn cho máy tính chạy Windows hoặc Linux và máy Mac chạy macOS. OpenJDK (Java Development Kit) mới nhất được đi kèm với Android Studio.

Để thiết lập và chạy Android Studio, trước tiên hãy kiểm tra các [yêu cầu hệ thống](#) để đảm bảo hệ thống của bạn đáp ứng các yêu cầu đó. Việc cài đặt tương tự cho tất cả các nền tảng. Bất kỳ sự khác biệt nào được lưu ý bên dưới.

1. Chuyển đến [trang web dành cho nhà phát triển Android](#) và làm theo hướng dẫn để tải xuống và [cài đặt Android Studio](#).
2. Chấp nhận cấu hình mặc định cho tất cả các bước và đảm bảo rằng tất cả các thành phần được chọn để cài đặt.
3. Sau khi cài đặt xong, Trình hướng dẫn cài đặt sẽ tải xuống và cài đặt một số thành phần bổ sung bao gồm SDK Android. Hãy kiên nhẫn, quá trình này có thể mất một chút thời gian tùy thuộc vào tốc độ Internet của bạn và một số bước có vẻ đú thura.
4. Khi quá trình tải xuống hoàn tất, Android Studio sẽ khởi động và bạn đã sẵn sàng tạo dự án đầu tiên của mình.

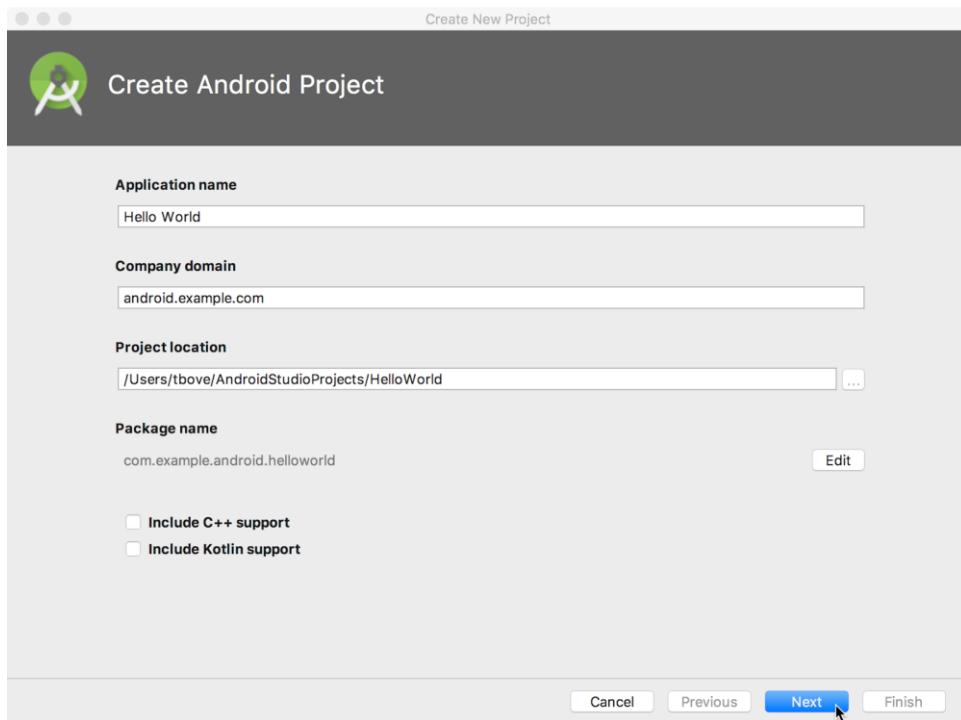
Khắc phục sự cố: Nếu bạn gặp sự cố khi cài đặt, hãy xem ghi [chú phát hành Android Studio](#) hoặc nhờ người hướng dẫn trợ giúp.

Nhiệm vụ 2: Tạo ứng dụng Hello World

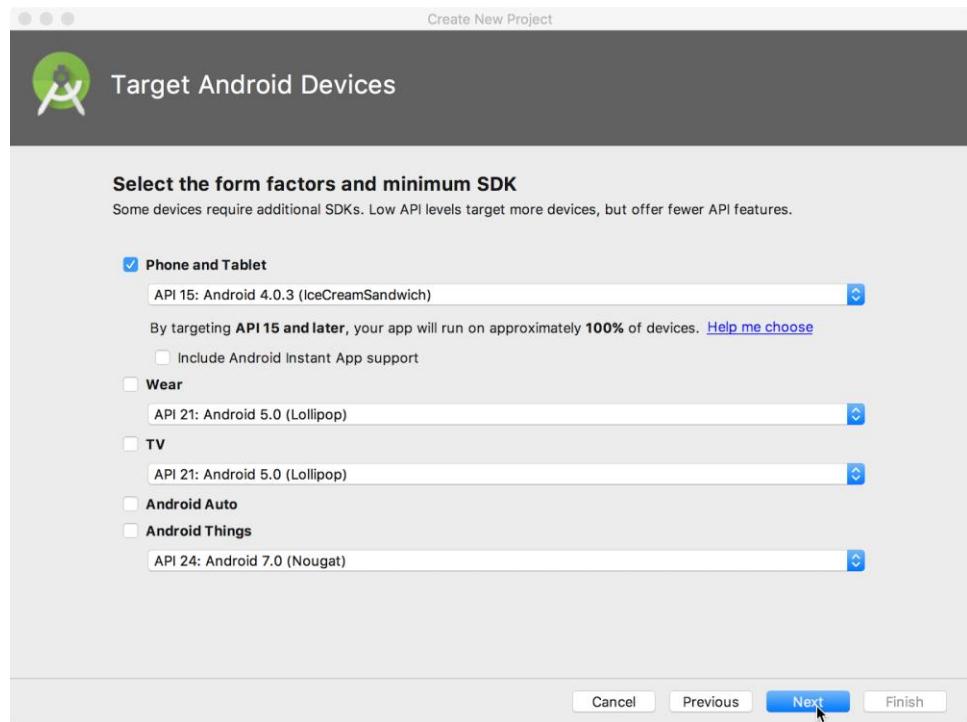
Trong tác vụ này, bạn sẽ tạo một ứng dụng hiển thị "Hello World" để xác minh rằng Android Studio đã được cài đặt chính xác và tìm hiểu những kiến thức cơ bản về phát triển bằng Android Studio.

2.1 Tạo dự án ứng dụng

1. Mở Android Studio nếu chưa mở.
2. Trong cửa sổ chính Welcome to Android Studio, nhập vào Start a new **Android Studio project** (Bắt đầu dự án Android Studio mới).
3. Trong cửa sổ **Create Android Project**, nhập **Hello World** cho tên Application.

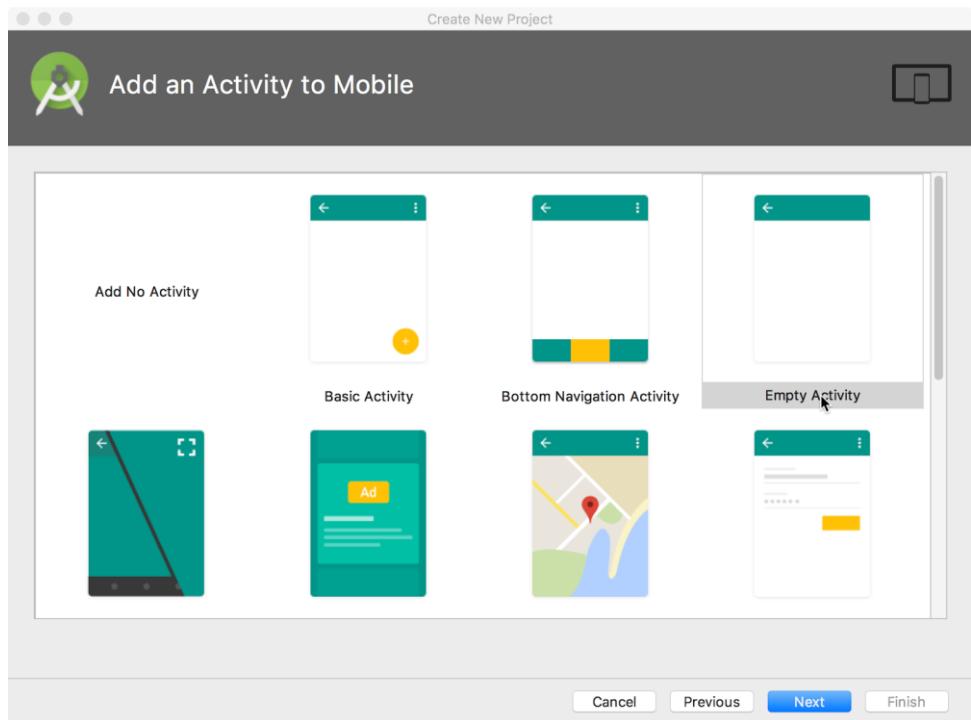


4. Xác minh rằng vị trí **Dự án mặc định** là nơi bạn muốn lưu trữ ứng dụng Hello World và các dự án Android Studio khác hoặc thay đổi vị trí đó thành thư mục ưa thích của bạn.
5. Chấp nhận android.example.com mặc định cho **Tên miền công ty** hoặc tạo một tên miền công ty. Nếu không định phát hành ứng dụng của mình, bạn có thể chấp nhận mặc định. Lưu ý rằng việc thay đổi tên gói của ứng dụng sau này là một công việc bổ sung.
6. Bỏ chọn các tùy chọn Bao gồm hỗ trợ C++ và Bao gồm hỗ trợ Kotlin, rồi nhấp vào **Next**
7. Trên **màn hình** Thiết bị Android mục tiêu, **Điện thoại và Máy tính bảng** sẽ được chọn. Đảm bảo rằng **API 15: Android 4.0.3 IceCreamSandwich** được đặt làm SDK tối thiểu; nếu không, hãy sử dụng menu bật lên để đặt nó.

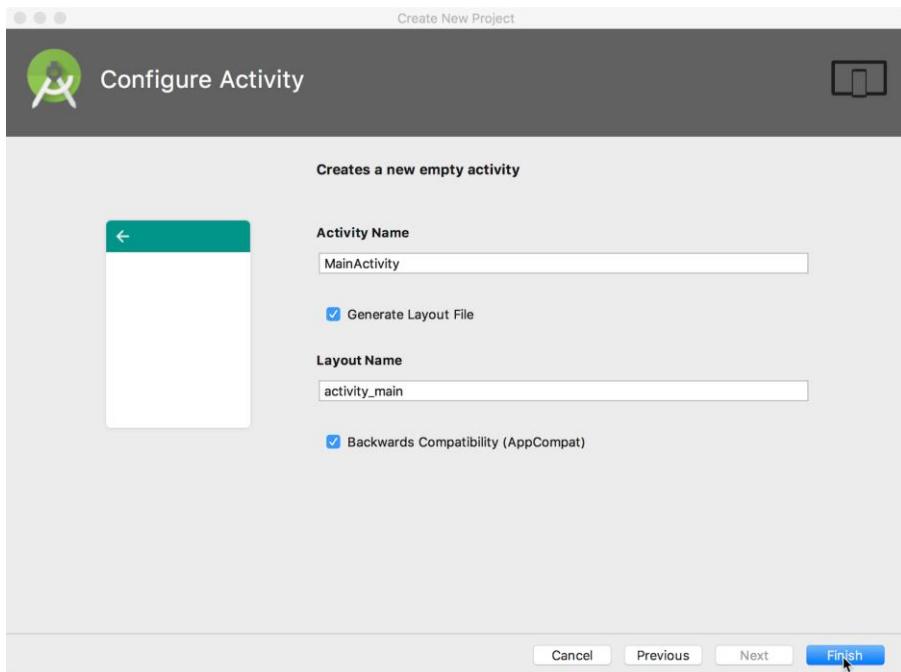


Đây là các cài đặt được sử dụng bởi các ví dụ trong các bài học cho khóa học này. Khi viết bài này, các cài đặt này làm cho ứng dụng Hello World của bạn tương thích với 97% thiết bị Android đang hoạt động trên Cửa hàng Google Play.

1. Bỏ chọn **Hỗ trợ bao gồm ứng dụng tức thì** và tất cả các tùy chọn khác. Sau đó nhấp vào **Tiếp theo**. Nếu dự án của bạn yêu cầu các thành phần bổ sung cho SDK mục tiêu bạn đã chọn, Android Studio sẽ tự động cài đặt các thành phần đó.
2. Cửa sổ **Thêm hoạt động** xuất hiện. Hoạt động là một thứ duy nhất, tập trung mà người dùng có thể làm. Nó là một thành phần quan trọng của bất kỳ ứng dụng Android nào. Hoạt động thường có bối cảnh được liên kết với nó xác định cách các thành phần giao diện người dùng xuất hiện trên màn hình. Android Studio cung cấp các mẫu Hoạt động để giúp bạn bắt đầu. Đối với dự án Hello World, chọn **Empty Activity** như hình dưới đây và nhấp vào **Next**.



1. The **Configure Activity** screen appears (which differs depending on which template you chose in the previous step). By default, the empty Activity provided by the template is named `MainActivity`. You can change this if you want, but this lesson uses `MainActivity`.



2. Đảm bảo rằng tùy chọn **Generate Layout** được chọn. Tên bộ cục theo mặc định là `activity_main`. Bạn có thể thay đổi điều này nếu muốn, nhưng bài học này sử dụng `activity_main`.
3. Đảm bảo rằng **Backwards Compatibility** được chọn. Điều này đảm bảo rằng ứng dụng của bạn sẽ tương thích ngược với các phiên bản Android trước đó.
4. Nhấn vào **Finish**.

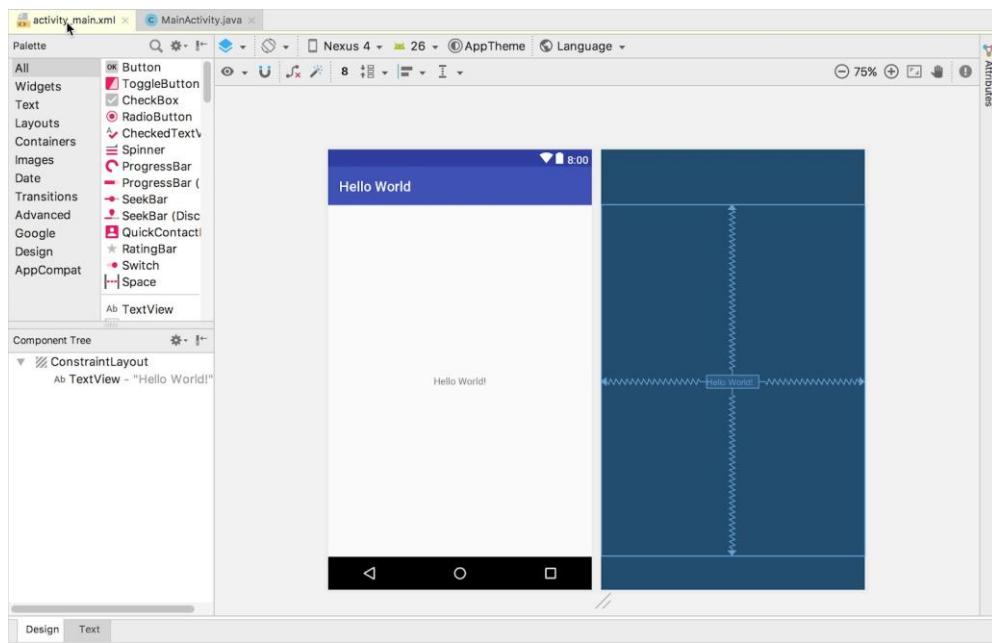
Android Studio sẽ tạo một thư mục cho các dự án của bạn và xây dựng dự án bằng [Gradle](#) (quá trình này có thể mất vài phút).

Mẹo: Xem trang [Định cấu hình nhà phát triển bản dựng của bạn](#) để biết thông tin chi tiết.

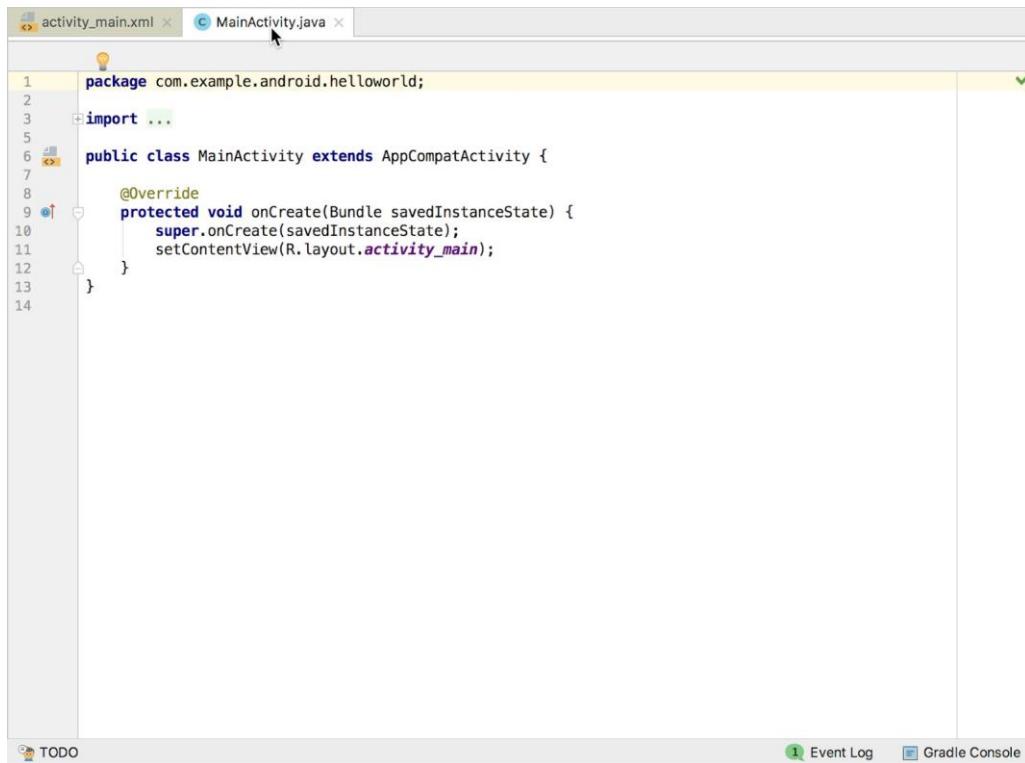
Bạn cũng có thể thấy thông báo "Mẹo trong ngày" với các phím tắt và các mẹo hữu ích khác. Bấm **Close** đóng tin nhắn.

Trình chỉnh sửa Android Studio sẽ xuất hiện. Làm theo các bước sau:

1. Nhấn vào **activity_main.xml** để xem trình chỉnh sửa bộ cục.
2. Nhập vào trình chỉnh sửa bộ cục **Design**, nếu chưa được chọn, để hiển thị một bản hiển thị đồ họa của bộ cục như hình dưới đây.



1. Nhấn vào **MainActivity.java** để xem trình chỉnh sửa mã như hình bên dưới.



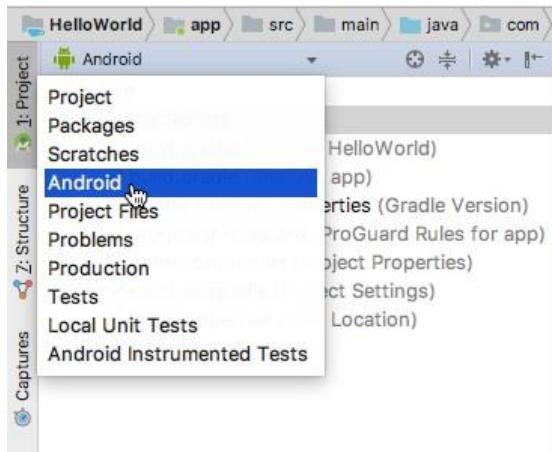
```
activity_main.xml MainActivity.java
```

```
1 package com.example.android.helloworld;
2
3 import ...
4
5 public class MainActivity extends AppCompatActivity {
6     @Override
7     protected void onCreate(Bundle savedInstanceState) {
8         super.onCreate(savedInstanceState);
9         setContentView(R.layout.activity_main);
10    }
11 }
12
13
14
```

2.2 Khám phá ngăn Project > Android

Trong thực tế này, bạn sẽ khám phá cách tổ chức dự án trong Android Studio.

1. Nếu chưa chọn, hãy nhấp vào biểu tượng **Project** trong cột tab dọc ở bên trái cửa sổ Android Studio. Ngăn dự án xuất hiện.
2. Cách xem dự án trong hệ thống phân cấp dự án Android tiêu chuẩn, chọn **Android** từ menu bật lên ở đầu ngăn Dự án, như được hiển thị bên dưới.

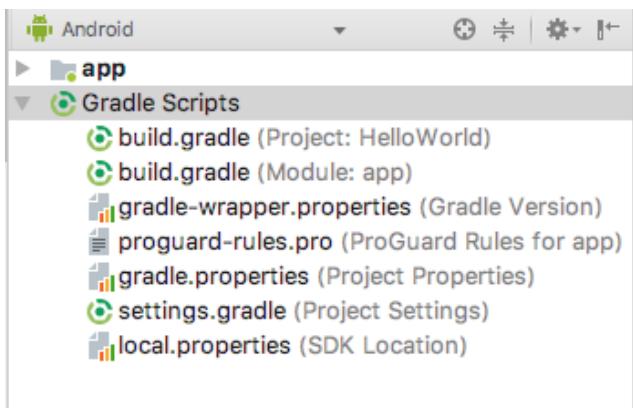


Note: Chương này và các chương khác đề cập đến ngăn Dự án, khi được đặt thành **Android**, như **Project > Android**

2.3 Khám phá thư mục Tập lệnh Gradle

Hệ thống bản dựng Gradle trong Android Studio giúp bạn dễ dàng đưa các tệp nhị phân bên ngoài hoặc các mô-đun thư viện khác vào bản dựng dưới dạng phần phụ thuộc.

Khi bạn tạo dự án ứng dụng lần đầu tiên, **Project > Android** xuất hiện cùng với thư mục **Gradle Scripts**. Hiện thị như hình dưới đây.



Làm theo các bước sau để xem hệ thống Gradle:

1. Nếu thư mục **Gradle Scripts** không được mở rộng, hãy nhấp vào hình tam giác để mở rộng thư mục đó. Thư mục này chứa tất cả các tệp cần thiết cho hệ thống xây dựng.
2. Tìm kiếm **build.gradle(Project: HelloWorld)**.

Đây là nơi bạn sẽ tìm thấy các tùy chọn cấu hình chung cho tất cả các mô-đun tạo nên dự án của bạn. Mỗi dự án Android Studio đều chứa một tệp bản dựng Gradle cấp cao nhất. Hầu hết thời gian, bạn sẽ không cần thực hiện bất kỳ thay đổi nào đối với tệp này, nhưng vẫn hữu ích khi hiểu nội dung của nó.

Theo mặc định, tệp bản dựng cấp cao nhất sử dụng khái niệm `buildscript` để xác định kho lưu trữ Gradle và các phần phụ thuộc chung cho tất cả các mô-đun trong dự án. Khi phần phụ thuộc của bạn không phải là thư viện cục bộ hoặc cây tệp, Gradle sẽ tìm kiếm các tệp trong bất kỳ kho lưu trữ trực tuyến nào được chỉ định trong khái niệm `kho lưu trữ` của tệp này. Theo mặc định, các dự án Android Studio mới khai báo JCenter và Google (bao gồm kho lưu trữ [Google Maven](#)) là vị trí kho lưu trữ:

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

3. Tìm kiếm thư mục **build.gradle(Module:app)**

Ngoài tệp build.gradle cấp dự án, mỗi mô-đun có một tệp build.gradle riêng cho phép bạn định cấu hình cài đặt bản dựng cho từng mô-đun cụ thể (ứng dụng HelloWorld chỉ có một mô-đun). Việc định cấu hình các tùy chọn cài đặt bản dựng này cho phép bạn cung cấp các tùy chọn đóng gói tùy chỉnh, chẳng hạn như các loại bản dựng bổ sung và hương vị sản phẩm. Bạn cũng có thể ghi đè các tùy chọn cài đặt trong tệp AndroidManifest.xml hoặc tệp build.gradle cấp cao nhất.

Tệp này thường là tệp cần chỉnh sửa khi thay đổi cấu hình cấp ứng dụng, chẳng hạn như khai báo các phần phụ thuộc trong phần phần phụ thuộc. Bạn có thể khai báo phần phụ thuộc thư viện bằng cách sử dụng một trong một số cấu hình phần phụ thuộc khác nhau. Mỗi cấu hình phần phụ thuộc cung cấp cho Gradle các hướng dẫn khác nhau về cách sử dụng thư viện. Ví dụ: thực hiện câu lệnh fileTree(dir: 'libs', include: ['*.jar']) thêm phần phụ thuộc của tất cả các tệp ".jar" bên trong thư mục libs.

Sau đây là thư mục **build.gradle(Module:app)** của ứng dụng HelloWorld:

```
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'
```

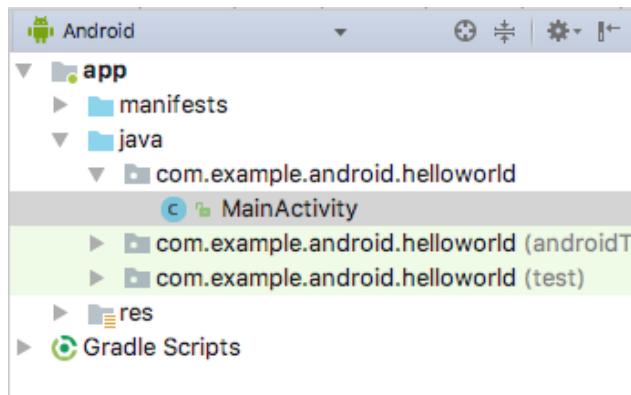
```
        androidTestImplementation 'com.android.support.test:runner:1.0.1'
        androidTestImplementation
            'com.android.support.test.espresso:espresso-core:3.0.1'
    }
```

- Nhập vào hình tam giác để đóng **Gradle Scripts**.

2.4 Khám phá ứng dụng và thư mục res

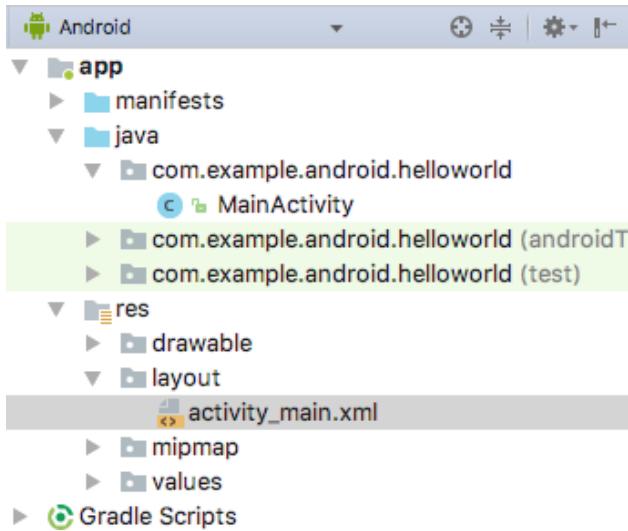
Tất cả mã và tài nguyên cho ứng dụng nằm trong thư mục ứng dụng và res.

- Mở rộng thư mục **app**, thư mục **java**, và thư mục **com.example.android.helloworld** để xem thư mục **MainActivity**. Nhấp đúp vào tệp sẽ mở tệp trong trình soạn thảo mã.



Thư mục **java** bao gồm lớp Java trong ba thư mục con, như trong hình trên. Mô hình **com.example.hello.helloworld** (hoặc tên miền bạn đã chỉ định) chứa tất cả các tệp cho một gói ứng dụng. Hai thư mục còn lại được sử dụng để kiểm tra và được mô tả trong một bài học khác. Đối với ứng dụng Hello World, chỉ có một gói và nó chứa **MainActivity.java**. Tên của Hoạt động (màn hình) đầu tiên mà người dùng nhìn thấy, cũng khởi tạo các tài nguyên trên toàn ứng dụng, thường được gọi là **MainActivity** (phần mở rộng tệp bị bỏ qua trong **Project > Android**).

-
2. Mở rộng thư mục **res** và thư mục **layout**, đồng thời nhấp đúp vào tệp **activity_main.xml** để mở nó trong trình chỉnh sửa bô cục.



Thư mục res chứa các tài nguyên, chẳng hạn như bô cục, chuỗi và hình ảnh. Hoạt động thường được liên kết với bô cục của các chế độ xem giao diện người dùng được xác định dưới dạng tệp XML. Tệp này thường được đặt tên theo Hoạt động của nó.

2.5 Khám phá thư mục tệp kê khai

Thư mục kê khai chứa các tệp cung cấp thông tin cần thiết về ứng dụng của bạn cho hệ thống Android, hệ thống phải có thông tin này trước khi có thể chạy bất kỳ mã nào của ứng dụng.

1. Mở rộng thư mục **manifests**.
2. Mở thư mục **AndroidManifest.xml**.

Tệp AndroidManifest.xml mô tả tất cả các thành phần của ứng dụng Android. Tất cả các thành phần của một ứng dụng, chẳng hạn như mỗi Hoạt động, phải được khai báo trong tệp XML này. Trong các

bài học khóa học khác, bạn sẽ sửa đổi tệp này để thêm tính năng và quyền tính năng. Để biết phần giới thiệu, hãy xem Tông [quan về tệp kê khai ứng dụng](#).

Task 3: Sử dụng thiết bị ảo (trình giả lập)

Trong tác vụ này, bạn sẽ sử dụng trình [quản lý Thiết bị ảo Android \(AVD\)](#) để tạo một thiết bị ảo (còn được gọi là trình mô phỏng) mô phỏng cấu hình cho một loại thiết bị Android cụ thể và sử dụng thiết bị ảo đó để chạy ứng dụng. Xin lưu ý rằng Trình mô phỏng Android có các [yêu cầu bổ sung](#) ngoài các yêu cầu hệ thống cơ bản đối với Android Studio.

Khi sử dụng Trình quản lý AVD, bạn xác định các đặc điểm phần cứng của thiết bị, cấp độ API, bộ nhớ, giao diện và các thuộc tính khác và lưu thiết bị đó dưới dạng thiết bị ảo. Với thiết bị ảo, bạn có thể kiểm tra ứng dụng trên các cấu hình thiết bị khác nhau (chẳng hạn như máy tính bảng và điện thoại) với các cấp độ API khác nhau mà không cần phải sử dụng thiết bị thực.

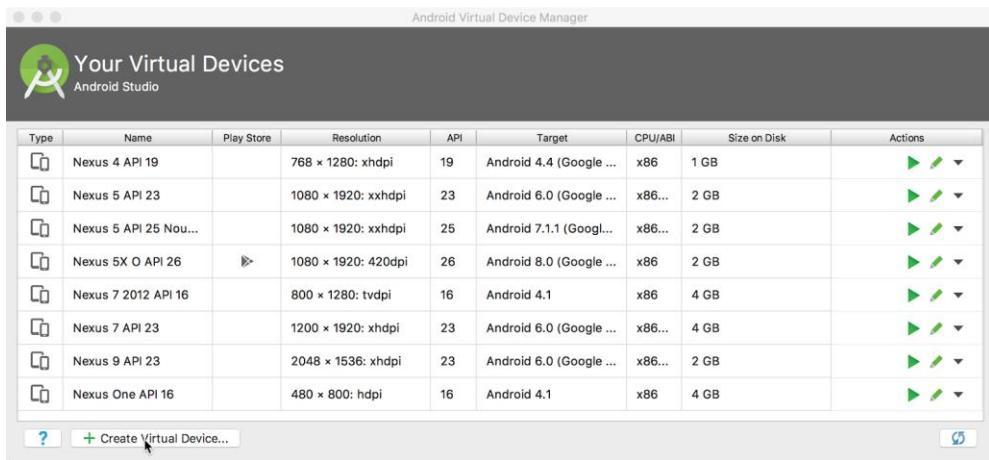
3.1 Tạo thiết bị ảo Android (AVD)

Để chạy trình mô phỏng trên máy tính, bạn phải tạo một cấu hình mô tả thiết bị ảo.

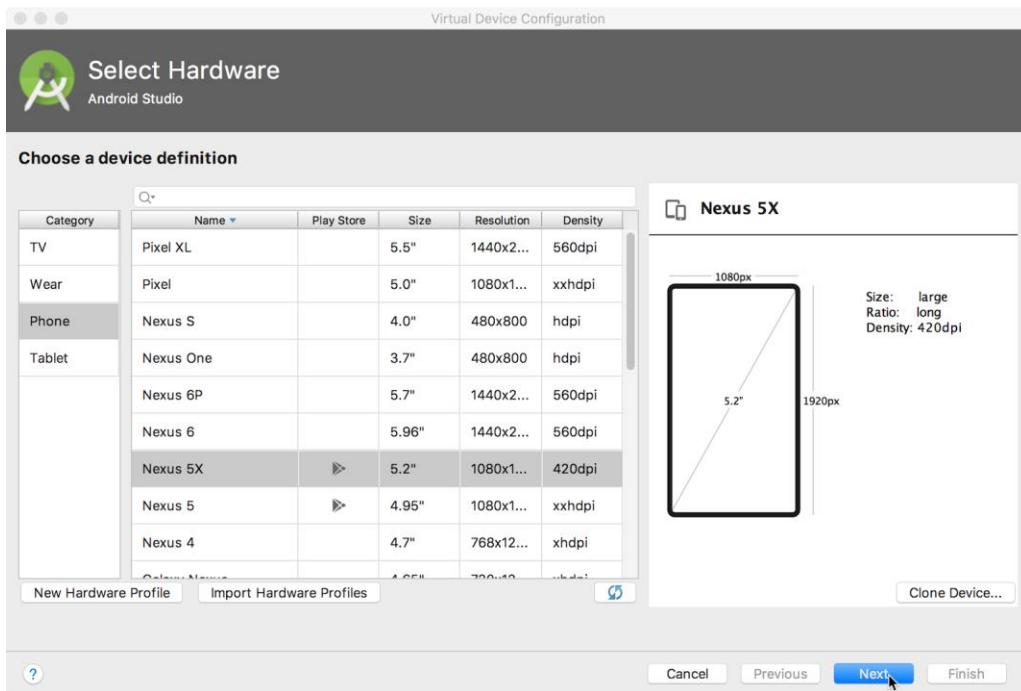
1. In Android Studio, select **Tools > Android > AVD Manager**, hoặc nhấp vào nút trình quản lý AVD



trên thanh công cụ. Màn hình thiết bị ảo xuất hiện **Your Virtual Devices**. Nếu bạn đã tạo thiết bị ảo, màn hình sẽ hiển thị chúng (như trong hình bên dưới); nếu không, bạn sẽ thấy một danh sách trống.

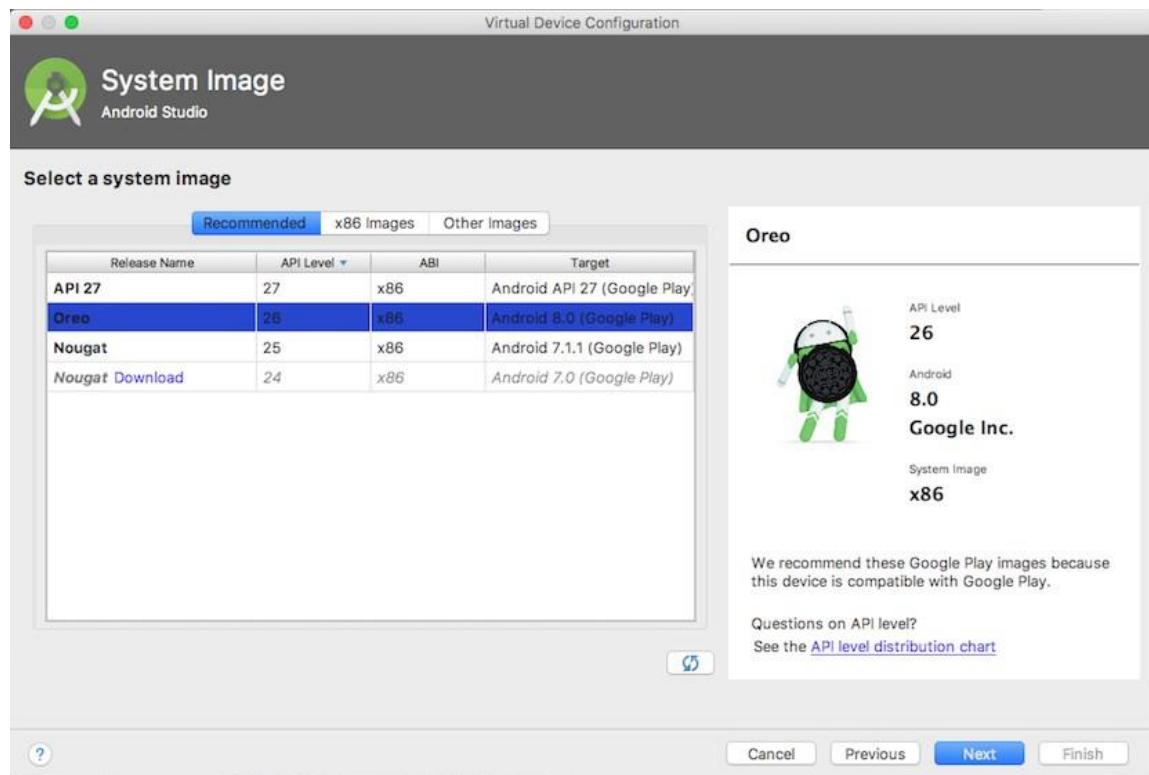


2. Nhấn vào **+Create Virtual Device**. Cửa sổ **Select Hardware** xuất hiện hiển thị danh sách các thiết bị phần cứng được định cấu hình sẵn. Đối với mỗi thiết bị, bảng cung cấp một cột cho kích thước hiển thị đường chéo của nó (**Size**), Độ phân giải màn hình tính bằng pixel (**Resolution**), và mật độ điểm ảnh (**Density**).



3. Chọn một thiết bị như **Nexus 5x** hoặc **Pixel XL**, và nhấn **Next**. Màn hình **System Image** xuất hiện.

-
- Nhấp vào biểu tượng **Recommended** nếu nó chưa được chọn và chọn phiên bản hệ thống Android để chạy trên thiết bị ảo.



Có nhiều phiên bản hơn được hiển thị trong mục **Recommended**. Nhìn vào **x86 Images** và **Other Images** tabs to see them.

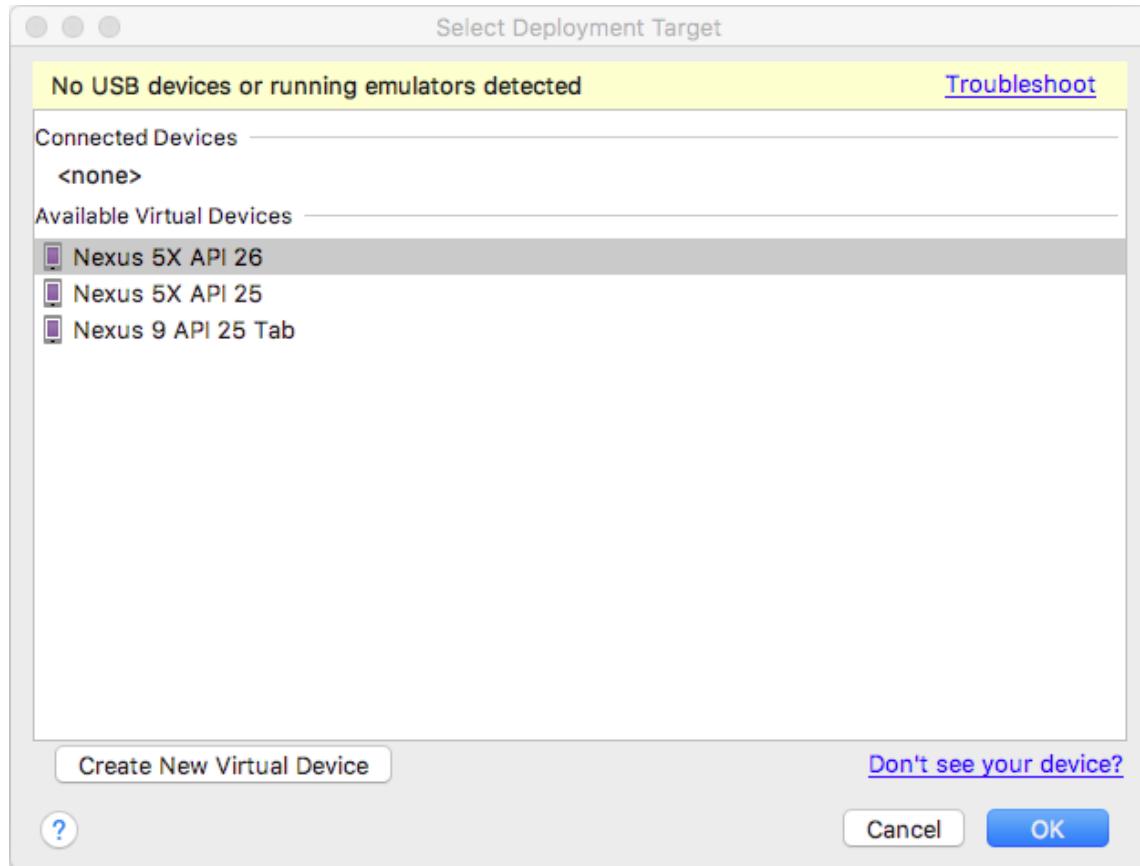
Nếu liên kết **Download** hiển thị bên cạnh hình ảnh hệ thống bạn muốn sử dụng, thì liên kết đó chưa được cài đặt. Nhấp vào liên kết để bắt đầu tải xuống, và nhấp vào **Finish** khi nó hoàn tất.

- Sau khi chọn hình ảnh hệ thống, nhấp vào **Next**. Cửa sổ thiết bị ảo **Android Virtual Device (AVD)** xuất hiện. Bạn cũng có thể thay đổi tên của AVD. Kiểm tra cấu hình của bạn và nhấp vào **Finish**.

3.2 Chạy ứng dụng trên thiết bị ảo

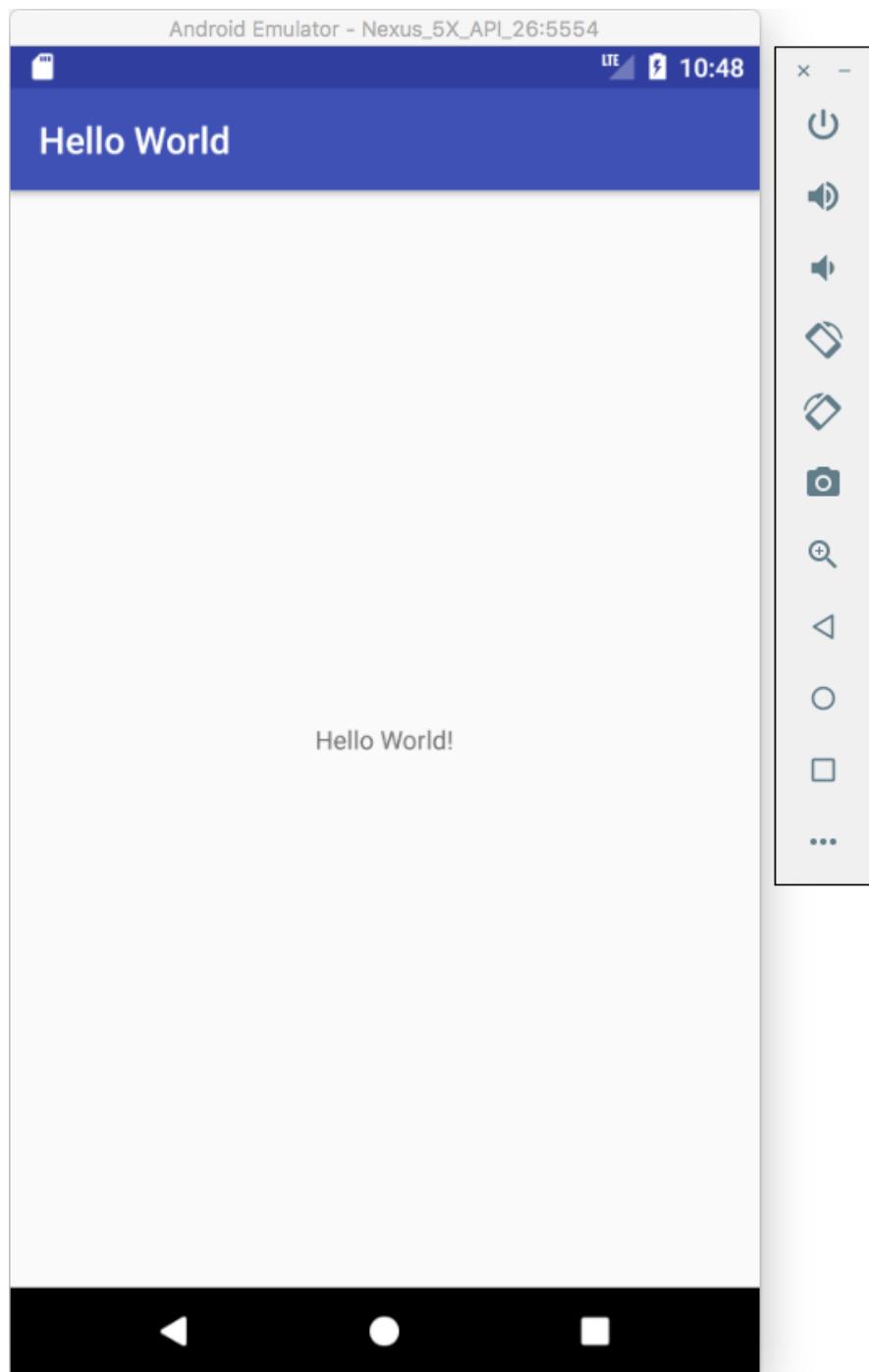
Trong nhiệm vụ này, cuối cùng bạn sẽ chạy ứng dụng Hello World của mình.

1. Trong Android Studio, hãy chọn **Run > Run app** hoặc nhập vào biểu tượng **Run**  trên thanh công cụ.
2. Cửa sổ **Select Deployment Target**, dưới **Available Virtual Devices**, chọn thiết bị ảo mà bạn vừa tạo và nhấp vào **OK**.



Trình giả lập khởi động và khởi động giống như một thiết bị vật lý. Tùy thuộc vào tốc độ máy tính của bạn, quá trình này có thể mất một lúc. Ứng dụng của bạn sẽ được xây dựng và sau khi trình mô phỏng đã sẵn sàng, Android Studio sẽ tải ứng dụng lên trình mô phỏng và chạy ứng dụng đó.

Bạn sẽ thấy ứng dụng Hello World như trong hình sau.



Mẹo: Khi thử nghiệm trên thiết bị ảo, bạn nên khởi động thiết bị một lần, ngay từ đầu phiên của bạn. Bạn không nên đóng ứng dụng cho đến khi hoàn tất việc kiểm tra ứng dụng của mình để ứng dụng của bạn không phải trải qua quá trình khởi động thiết bị một lần nữa. Để đóng thiết bị ảo, hãy nhấp vào nút X ở đầu trình giả lập, chọn **Thoát** từ menu hoặc nhấn **Control-Q** trong Windows hoặc **Command-Q** trong macOS.

Task 4: Nhiệm vụ 4: (Tùy chọn) Sử dụng thiết bị vật lý

Trong nhiệm vụ cuối cùng này, bạn sẽ chạy ứng dụng của mình trên thiết bị di động vật lý như điện thoại hoặc máy tính bảng. Bạn phải luôn kiểm tra ứng dụng của mình trên cả thiết bị ảo và thiết bị vật lý.

Những gì bạn cần:

- Thiết bị Android như điện thoại hoặc máy tính bảng.
- Cáp dữ liệu để kết nối thiết bị Android với máy tính qua cổng USB.
- Nếu bạn đang sử dụng hệ thống Linux hoặc Windows, bạn có thể cần thực hiện các bước bổ sung để chạy trên thiết bị phàn cứng. Kiểm tra [tài liệu Sử dụng thiết bị phàn cứng](#). Bạn cũng có thể cần cài đặt trình điều khiển USB thích hợp cho thiết bị của mình. Đối với trình điều khiển USB dựa trên Windows, hãy xem Trình [điều khiển OEM](#).

4.1 Bật gỡ lỗi USB

Để cho phép Android Studio giao tiếp với thiết bị của bạn, bạn phải bật tính năng Gỡ lỗi USB trên thiết bị Android của mình. Tính năng này được bật trong **Developer options** Cài đặt của thiết bị của bạn.

Trên Android 4.2 trở lên, màn hình **Tùy chọn nhà phát triển** bị ẩn theo mặc định. Để hiển thị các tùy chọn dành cho nhà phát triển và bật Gỡ lỗi USB:

- Trên thiết bị của bạn, mở **Settings**, tìm kiếm **About phone**, nhấn vào **About phone**, và nhấn vào **Build number 7** lần.
- Quay lại màn hình trước đó (**Settings / System**). **Developer options** xuất hiện trong danh

sách. Nhấn vào **Developer options**.

3. Chọn **USB Debugging**.

4.2 Chạy ứng dụng trên thiết bị

Giờ đây, bạn có thể kết nối thiết bị của mình và chạy ứng dụng từ Android Studio.

1. Kết nối thiết bị của bạn với máy phát triển bằng cáp USB.
2. Nhấn vào nút  trên thanh công cụ. Cửa sổ **Select Deployment Target** với danh sách các trình giả lập có sẵn và thiết bị được kết nối
3. Chọn thiết bị của bạn và nhấp vào **OK**.

Android Studio cài đặt và chạy ứng dụng trên thiết bị của bạn.

Troubleshooting

Nếu Android Studio không nhận dạng thiết bị của bạn, hãy thử các cách sau:

1. Rút phích cắm và cắm lại thiết bị của bạn.
2. Khởi động lại Android Studio.

Nếu máy tính của bạn vẫn không tìm thấy thiết bị hoặc tuyên bố thiết bị là "không được phép", hãy làm theo các bước sau:

1. Rút phích cắm của thiết bị.
2. Trên thiết bị, hãy mở **Developer Options** trong **Settings**.
3. Nhấn vào thu hồi ủy quyền **USB Debugging**
4. Kết nối lại thiết bị với máy tính của bạn.
5. Khi được nhắc, hãy cấp ủy quyền.

Bạn có thể cần cài đặt trình điều khiển USB thích hợp cho thiết bị của mình. Xem [Using Hardware Devices documentation](#).

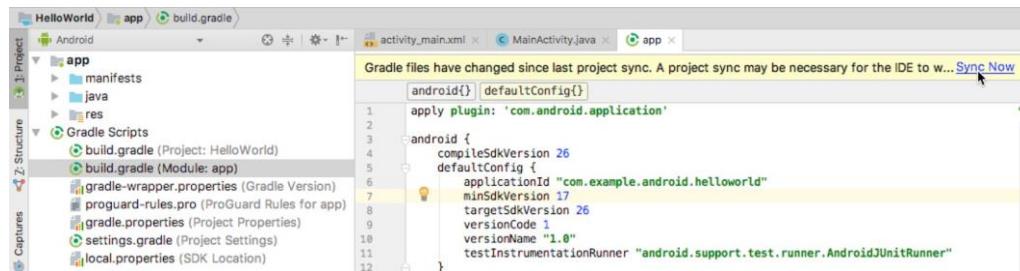
Task 5: Thay đổi cấu hình Gradle của ứng dụng

Trong tác vụ này, bạn sẽ thay đổi điều gì đó về cấu hình ứng dụng trong tệp **build.gradle(Module:app)** để tìm hiểu cách thực hiện các thay đổi và đồng bộ hóa chúng với dự án Android Studio của bạn.

5.1 Thay đổi phiên bản SDK tối thiểu cho ứng dụng

Làm theo các bước sau:

1. Mở rộng thư mục **Gradle Scripts** nếu nó chưa được mở và nhấp đúp vào **build.gradle(Module:app)**
Nội dung của tệp xuất hiện trong trình soạn thảo mã.
2. Trong khối defaultConfig, hãy thay đổi giá trị của minSdkVersion thành 17 như hình dưới đây
(ban đầu giá trị này được đặt thành 15).



Trình chỉnh sửa mã hiển thị một thanh thông báo ở trên cùng với **Sync Now**.

5.2 Đồng bộ hóa cấu hình Gradle mới

Khi bạn thực hiện các thay đổi đối với các tệp cấu hình bản dựng trong một dự án, Android Studio yêu cầu bạn đồng bộ hóa các tệp dự án để có thể nhập các thay đổi về cấu hình bản dựng và chạy một số kiểm tra để đảm bảo cấu hình sẽ không tạo ra lỗi bản dựng.

Để đồng bộ hóa các tệp dự án, hãy nhấp vào **Sync Now** trong thanh thông báo xuất hiện khi thực hiện thay đổi



(như trong hình trước) hoặc nhấp vào **Sync Project with Gradle Files** trên thanh công cụ.

Khi quá trình đồng bộ hóa Gradle kết thúc, thông báo Bản dựng Gradle đã hoàn tất sẽ xuất hiện ở góc dưới cùng bên trái của cửa sổ Android Studio.

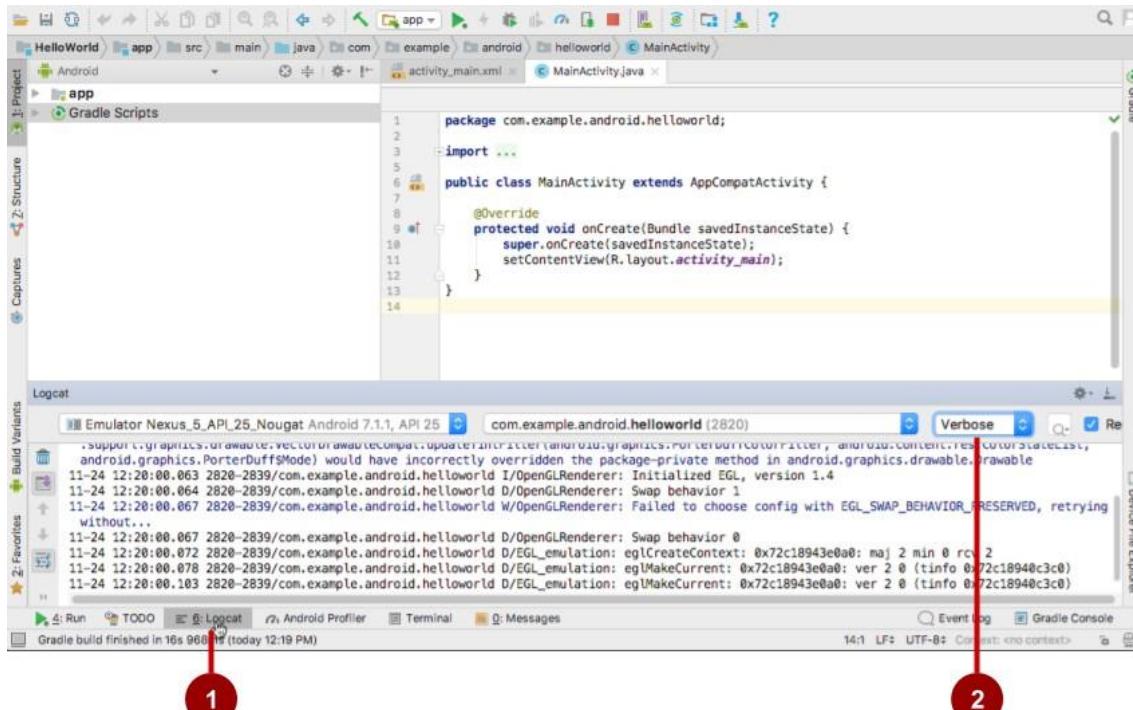
Để tìm hiểu sâu hơn về Gradle, hãy xem tài liệu Tổng [quan về hệ thống xây dựng](#) và Định [cấu hình bản dựng Gradle](#).

Nhiệm vụ 6: Thêm câu lệnh nhật ký vào ứng dụng của bạn

Trong nhiệm vụ này, bạn sẽ thêm [câu lệnh Nhật ký](#) vào ứng dụng của mình, cụm từ này hiển thị thông báo trong **Logcat**. Thông báo nhật ký là một công cụ gõ lỗi mạnh mẽ mà bạn có thể sử dụng để kiểm tra các giá trị, đường dẫn thực thi và báo cáo ngoại lệ.

6.1 Xem ngăn Logcat

Để xem mục **Logcat**, nhấn vào tab **Logcat** ở cuối cửa sổ Android Studio như trong hình bên dưới.



Trong hình trên:

1. Tab **Logcat** để mở và đóng ngăn **Logcat**, hiển thị thông tin về ứng dụng của bạn khi ứng dụng đang chạy. Nếu bạn thêm câu lệnh Nhật ký vào ứng dụng của mình, thông báo Nhật ký sẽ xuất hiện ở đây.
2. Menu Mức nhật ký được đặt thành **Verbose** (mặc định), hiển thị tất cả các thông báo Nhật ký. Các cài đặt khác bao gồm: **Debug**, **Error**, **Info**, và **Warn**.

1. Thêm câu lệnh nhật ký vào ứng dụng của bạn

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

Câu lệnh nhật ký trong mã ứng dụng của bạn sẽ hiển thị thông báo trong ngăn Logcat. Chẳng hạn:

Các phần của thông điệp là:

- **Log**: Lớp [Log](#) để gửi thông báo nhật ký đến ngăn Logcat.
- **d**: Cài đặt mức nhật ký **Debug** để lọc thông báo nhật ký hiển thị trong ngăn Logcat. Các cấp độ nhật ký khác là cho **Error**, cho **Warn**, và cho **Info**.
- "**MainActivity**": Đối số đầu tiên là một thẻ có thể được sử dụng để lọc các thông báo trong Ngăn Logcat. Đây thường là tên của Hoạt động mà từ đó tin nhắn originates. Tuy nhiên, bạn có thể làm cho điều này bất cứ thứ gì hữu ích cho bạn để gỡ lỗi. Theo quy ước, thẻ nhật ký được định nghĩa là hằng số cho Hoạt động:

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

- "**Hello world**": Lập luận thứ hai là thông điệp thực tế.

Làm theo các bước sau:

1. Mở ứng dụng Hello World của bạn trong Android studio và mở MainActivity.
2. Để tự động thêm các tính năng nhập rõ ràng vào dự án của bạn (chẳng hạn như android.util.Log bắt buộc để sử dụng Log), chọn **File > Settings** trong Windows, hoặc **Android Studio > Preferences** trong macOS.
3. Chọn **Editor > General >Auto Import**. Chọn tất cả các hộp kiểm và đặt **Insert imports on paste** cho All.
4. Nhấn vào **Apply** rồi nhấp vào **OK**.
5. Trong phương thức **onCreate()** của MainActivity, Thêm câu lệnh sau:

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

Phương thức **onCreate()** bây giờ sẽ giống như mã sau:

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

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

1. Nếu ngăn Logcat chưa mở, hãy nhấp vào biểu tượng **Logcat** ở cuối Android Studio để mở.
2. Kiểm tra xem tên của mục tiêu và tên gói của ứng dụng có chính xác không.
3. Thay đổi cấp độ Nhật ký trong **Logcat** thành **Debug**
4. Chạy ứng dụng của bạn.

Thông báo sau sẽ xuất hiện trong ngăn Logcat:

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

Thử thách mã hóa

Note: Tất cả các thử thách mã hóa là tùy chọn và không phải là điều kiện tiên quyết cho các bài học sau.

Challenge: Bây giờ bạn đã thiết lập và làm quen với quy trình phát triển cơ bản, hãy làm như sau:

1. Tạo một dự án mới trong Android Studio.
2. Thay đổi lời chào "Hello World" thành "Chúc mừng sinh nhật thành" và tên của ai đó có sinh nhật gần đây.
3. (Tùy chọn) Chụp ảnh màn hình ứng dụng đã hoàn thành của bạn và gửi email cho người có sinh nhật bạn quên.
4. Một cách sử dụng phổ biến của lớp [Log](#) là để [Java exceptions](#) khi chúng xảy ra trong chương trình của bạn. Có một số phương pháp hữu ích, chẳng hạn như [Log.e\(\)](#), mà bạn có thể sử dụng cho mục đích này. Khám phá

các phương thức bạn có thể sử dụng để bao gồm một ngoại lệ với thông báo Nhật ký. Sau đó, viết mã trong ứng dụng của bạn để kích hoạt và ghi lại một ngoại lệ.

Tóm tắt

1. Để cài đặt Android Studio, hãy truy cập vào Android [Studio](#) và làm theo hướng dẫn để tải xuống và cài đặt.
2. Khi tạo ứng dụng mới, hãy đảm bảo rằng **API 15: Android 4.0.3 IceCreamSandwich** được đặt là SDK tối thiểu.
3. Để xem hệ thống phân cấp Android của ứng dụng trong ngăn Project (Dự án), hãy nhấp vào biểu tượng **Project** trong cột tab dọc, sau đó chọn **Android** trong menu bật lên ở trên cùng.
4. Chính sửa tệp **build.gradle(Module:app)** khi bạn cần thêm thư viện mới vào dự án hoặc thay đổi phiên bản thư viện.
5. Tất cả mã và tài nguyên cho ứng dụng nằm trong thư mục ứng dụng và res. Thư mục java bao gồm các hoạt động, kiểm tra và các thành phần khác trong mã nguồn Java. Thư mục res chứa các tài nguyên, chẳng hạn như bố cục, chuỗi và hình ảnh.
6. Chính sửa tệp **AndroidManifest.xml** để thêm các thành phần tính năng và quyền vào ứng dụng Android của bạn. Tất cả các thành phần cho một ứng dụng, chẳng hạn như nhiều hoạt động, phải được khai báo trong tệp XML này.
 1. Sử dụng [Android Virtual Device \(AVD\) manager](#) để tạo một thiết bị ảo (còn được gọi là trình mô phỏng) để chạy ứng dụng của bạn.
 1. Thêm [Log](#) trong ứng dụng của bạn, hiển thị thông báo trong ngăn Logcat như một công cụ cơ bản để gỡ lỗi.
 1. Để chạy ứng dụng của bạn trên thiết bị Android vật lý bằng Android Studio, hãy bật tính năng Gỡ lỗi USB trên thiết bị. Mở **Settings > About phone** và nhấn vào **Build number** bảy lần. Quay lại màn hình trước đó (**Settings**), và nhấn **Developer options**. Chọn **USB Debugging**.

Các khái niệm liên quan

Tài liệu khái niệm liên quan có trong [1.0: Introduction to Android](#) và [1.1 Your first Android app](#).

Tìm hiểu thêm

Tài liệu Android Studio:

- [Android Studio download page](#)
- [Android Studio release notes](#)
- [Meet Android Studio](#)
- [Logcat command-line tool](#)
- [Android Virtual Device \(AVD\) managerApp Manifest Overview](#)
- [Configure your build](#)
- [Logclass](#)
- [Create and Manage Virtual Devices](#)

Khác:

- [How do I install Java?](#)
- [Installing the JDK Software and Setting JAVA_HOME](#)
- [Gradle site](#)
- [Apache Groovy syntax](#)
- [Gradle Wikipedia page](#)

Bài tập về nhà

Xây dựng và chạy ứng dụng

1. Tạo một dự án Android mới từ Empty Template.
2. Thêm câu lệnh ghi nhật ký cho các cấp độ nhật ký khác nhau trong onCreate() trong hoạt động chính.
3. Tạo trình mô phỏng cho thiết bị, nhắm mục tiêu bất kỳ phiên bản Android nào bạn thích và chạy ứng dụng.
4. Sử dụng tính năng lọc trong **Logcat** để tìm câu lệnh nhật ký của bạn và điều chỉnh các cấp độ để chỉ hiển thị các câu lệnh gỡ lỗi hoặc ghi nhật ký lỗi.

Trả lời những câu hỏi:

Câu hỏi 1

Tên của tệp bô cục cho hoạt động chính là gì?

- MainActivity.java
- AndroidManifest.xml
- activity_main.xml
- build.gradle

Câu hỏi 2

Tên của tài nguyên chuỗi chỉ định tên của ứng dụng là gì?

- app_name
- xmlns:app
- android:name
- applicationId

Câu hỏi 3

Bạn sử dụng công cụ nào để tạo trình giả lập mới?

- Android Device Monitor
- AVD Manager
- SDK Manager
- Theme Editor

Câu hỏi 4

Giả sử rằng ứng dụng của bạn bao gồm câu lệnh ghi nhật ký sau:

Log.i("MainActivity", "MainActivity layout is complete");	Page 35
---	---------

Bạn thấy câu lệnh "Bố cục MainActivity đã hoàn tất" trong ngăn **Logcat** nếu trình đơn Cấp nhật ký được đặt thành tùy chọn nào sau đây?

- Verbose
- Debug
- Info
- Warn
- Error
- Assert

Gửi ứng dụng của bạn để chấm điểm

Kiểm tra để đảm bảo ứng dụng có những điều sau:

Hoạt động hiển thị "Hello World" trên màn hình.

Câu lệnh nhật ký trong onCreate() trong hàm MainActivity.

Log level trong **Logcat** chỉ hiển thị các câu lệnh gỡ lỗi hoặc ghi nhật ký lỗi.

Bài 1.2 Phần A: Giao diện người dùng tương tác đầu tiên của bạn

Giới thiệu

Giao diện người dùng (UI) xuất hiện trên màn hình của thiết bị Android bao gồm một hệ thống phân cấp các đối tượng được gọi là *view* — mọi phần tử của màn hình đều là [view](#). Lớp View đại diện cho khung xây dựng cơ bản cho tất cả các thành phần giao diện người dùng và lớp cơ sở cho các lớp

cung cấp giao diện người dùng tương tác các thành phần như nút, hộp kiểm và trường nhập văn bản.
Các lớp con View thường được sử dụng được mô tả trong một số bài học bao gồm:

-
1. [TextView](#) để hiển thị văn bản.
 2. [EditText](#) để cho phép người dùng nhập và chỉnh sửa văn bản.
 3. [Button](#) và các yếu tố có thể nhấp khác (Ví dụ như: [RadioButton](#), [CheckBox](#), và [Spinner](#)) để cung cấp hành vi tương tác.
 4. [ScrollView](#) và [RecyclerView](#) để hiển thị các mục có thể cuộn.
 5. [ImageView](#) để hiển thị hình ảnh.
 6. [ConstraintLayout](#) và [LinearLayout](#) để chứa các phần tử View khác và định vị chúng .

Mã Java hiển thị và điều khiển giao diện người dùng được chứa trong một lớp mở rộng [Activity](#). Hoạt động thường được liên kết với bộ cục của chế độ xem giao diện người dùng được xác định dưới dạng tệp XML (Ngôn ngữ đánh dấu mở rộng). Tệp XML này thường được đặt tên theo Activity của nó và xác định bộ cục của các phần tử View trên màn hình.

Ví dụ: mã MainActivity trong ứng dụng Hello World hiển thị bộ cục được xác định trong activity_main.xml tệp bộ cục, bao gồm TextView với văn bản "Hello World".

Trong các ứng dụng phức tạp hơn, Hoạt động có thể triển khai các hành động để phản hồi các thao tác nhân của người dùng, vẽ nội dung đồ họa hoặc yêu cầu dữ liệu từ cơ sở dữ liệu hoặc internet. Bạn tìm hiểu thêm về lớp Sinh Hoạt trong một bài học khác.

Trong thực tế này, bạn sẽ tìm hiểu cách tạo ứng dụng tương tác đầu tiên của mình — một ứng dụng cho phép tương tác với người dùng. Bạn tạo một ứng dụng bằng cách sử dụng mẫu Hoạt động trống. Bạn cũng học cách sử dụng trình soạn thảo bộ cục để thiết kế bộ cục và cách chỉnh sửa bộ cục trong XML. Bạn cần phát triển những kỹ năng này để có thể hoàn thành các bài thực hành khác t

Những điều bạn nên biết

Bạn nên làm quen với:

1. Cách cài đặt và mở Android Studio.
1. Cách tạo ứng dụng HelloWorld.
1. How to run the HelloWorld app.

Những gì bạn sẽ học

- Cách tạo ứng dụng với hành vi tương tác.
- Cách sử dụng trình chỉnh sửa bộ cục để thiết kế bộ cục.
- Cách chỉnh sửa bộ cục trong XML.
- Rất nhiều thuật ngữ mới. Kiểm tra [bảng thuật ngữ từ vựng và khái niệm](#) để biết các định nghĩa thân thiện.

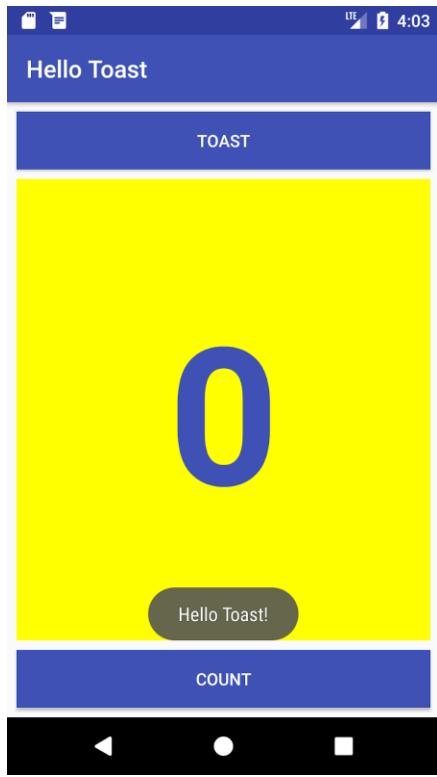
Bạn sẽ làm gì

7. Create an app and add two Button elements and a TextView to the layout.
8. Manipulate each element in the [ConstraintLayout](#) to constrain them to the margins and other elements.
9. Change UI element attributes.
10. Edit the app's layout in XML.
11. Extract hardcoded strings into string resources.
12. Implement click-handler methods to display messages on the screen when the user taps each Button.

Tổng quan về ứng dụng

Ứng dụng HelloToast bao gồm hai phần tử Button và một TextView. Khi người dùng nhấn vào Nút đầu tiên, nó sẽ hiển thị một thông báo ngắn (Toast) trên màn hình. Nhấn vào nút thứ hai sẽ tăng bộ đếm "nhấp chuột" được hiển thị trong TextView, bắt đầu từ không.

Đây là những gì ứng dụng đã hoàn thành trông như thế nào:



Nhiệm vụ 1: Tạo và khám phá một dự án mới

Trong thực tế này, bạn thiết kế và triển khai một dự án cho ứng dụng HelloToast. Một liên kết đến mã giải pháp được cung cấp ở cuối.

1.1 Tạo dự án Android Studio

Khởi động Android Studio và tạo một dự án mới với các thông số sau:

Thuộc tính	Giá trị
------------	---------

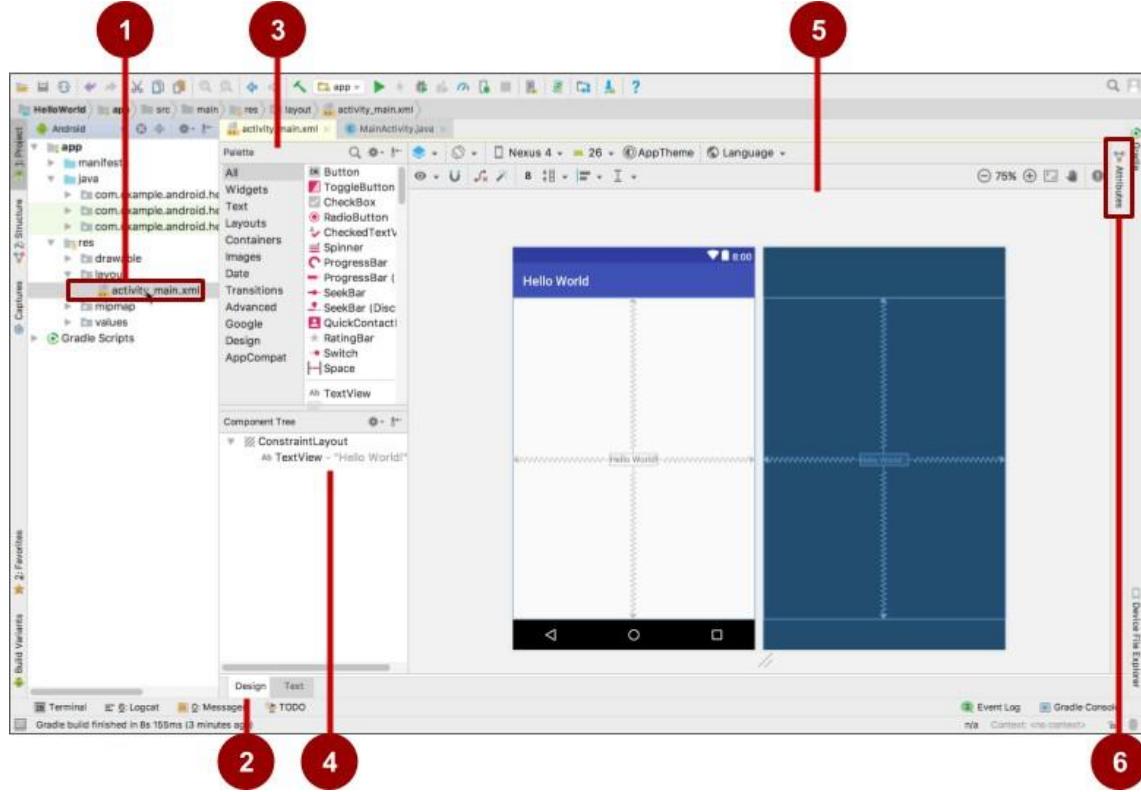
Tên ứng dụng	Hello Toast
Tên công ty	com.example.android (or your own domain)
SDK tối thiểu trên điện thoại và máy tính bảng	API15: Android 4.0.3 IceCreamSandwich
Mẫu	Empty Activity
Tạo hộp tệp bối cảnh	Selected
Hộp tương thích ngược	Selected

- Chọn **Run > Run app** hoặc nhấn vào Nút  trên thanh công cụ để tạo và thực thi ứng dụng trên trình mô phỏng hoặc thiết bị của bạn.

1.2 Khám phá trình chỉnh sửa bố cục

Android Studio cung cấp trình chỉnh sửa bố cục để nhanh chóng tạo bố cục của các thành phần giao diện người dùng (UI) của ứng dụng. Nó cho phép bạn kéo các phần tử vào chế độ xem thiết kế trực quan và bản thiết kế, định vị chúng trong bố cục, thêm ràng buộc và đặt thuộc tính. Các ràng buộc xác định vị trí của một phần tử giao diện người dùng trong bố cục. Một ràng buộc đại diện cho một kết nối hoặc căn chỉnh với một chế độ xem khác, bố cục cha hoặc một hướng dẫn vô hình.

Khám phá trình chỉnh sửa bố cục và tham khảo hình dưới đây khi bạn làm theo các bước được đánh số:



1. Trong ứng dụng **app > res > layout** trong thư mục **Project > Android**, Nhấp đúp vào **activity_main.xml** file để mở nó, nếu nó chưa được mở.
2. Nhấp vào tab **Thiết kế** nếu nó chưa được chọn. Bạn sử dụng tab **Thiết kế** để thao tác với các phần tử và bố cục, và tab **Văn bản** để chỉnh sửa mã XML cho bố cục.
3. Ngăn **Bảng màu** hiển thị các thành phần giao diện người dùng mà bạn có thể sử dụng trong bố cục của ứng dụng.
4. Ngăn **cây thành phần** hiển thị hệ thống phân cấp ché độ xem của các phần tử giao diện người dùng. Các phần tử View được tổ chức thành một hệ thống phân cấp cây gồm cha mẹ và con, trong đó con kế thừa các thuộc tính của cha mẹ của nó. Trong hình trên, TextView là con của [ConstraintLayout](#). Các em sẽ học về các yếu tố này ở phần sau của bài học này.
5. Các ngăn thiết kế và bản thiết kế của trình chỉnh sửa bố cục hiển thị các phần tử giao diện người dùng trong bố cục. Trong hình trên, bố cục chỉ hiển thị một phần tử: TextView hiển thị "Hello World".
6. Tab **Thuộc tính** hiển thị ngăn **Thuộc tính** để đặt thuộc tính cho phần tử giao diện người dùng.

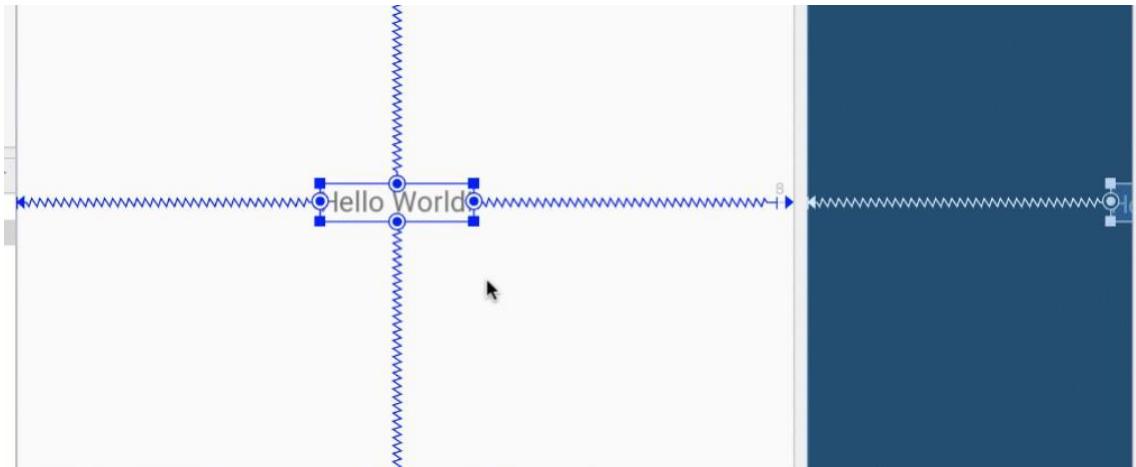
Nhiệm vụ 2: Thêm các thành phần View trong trình soạn thảo bố cục

Trong tác vụ này, bạn tạo bố cục giao diện người dùng cho ứng dụng HelloToast trong trình chỉnh sửa bố cục bằng cách sử dụng [các tính năng ConstraintLayout](#). Bạn có thể tạo các ràng buộc theo cách thủ công, như được hiển thị sau hoặc tự động bằng cách sử dụng **công cụ Tự động kết nối**.

Kiểm tra các ràng buộc của phần tử

Làm theo các bước sau:

2. Mở activity_main.xml từ ngăn Project > Android nếu chưa mở. Nếu Tab Thiết kế chưa được chọn, hãy nhấp vào nó. Nếu không có bản thiết kế, hãy nhấp vào nút Chọn bìa mặt thiết kế  trên thanh công cụ và chọn Thiết kế + Bản thiết kế.
3. Công cụ Tự động kết nối  cũng nằm trong thanh công cụ. Nó được bật theo mặc định. Đối với bước này, hãy đảm bảo rằng công cụ không bị tắt.
4. Nhấp vào nút phóng to  để phóng to các ngăn thiết kế và bản thiết kế để có cái nhìn cận cảnh.
5. Chọn TextView trong ngăn Component Tree. TextView "Hello World" được đánh dấu trong các ngăn thiết kế và bản thiết kế và các ràng buộc cho phần tử được hiển thị.
6. Tham khảo hình động bên dưới cho bước này. Nhấp vào bộ điều khiển hình tròn ở phía bên phải của TextView để xóa ràng buộc ngang liên kết chế độ xem với phía bên phải của bố cục. TextView nhảy sang bên trái vì nó không còn bị ràng buộc ở phía bên phải nữa. Để thêm lại ràng buộc ngang, hãy nhấp vào cùng một tay cầm và kéo một đường vào the right side of the layout.



Trong ngăn blueprint hoặc thiết kế, các bộ điều khiển sau xuất hiện trên phần tử TextView:

1. **Constraint handle:** Để tạo một ràng buộc như trong hình động ở trên, hãy nhấp vào một bộ điều khiển ràng buộc, được hiển thị dưới dạng một vòng tròn ở bên cạnh của một phần tử. Sau đó kéo tay cầm vào một tay cầm ràng buộc khác hoặc đến ranh giới mẹ. Một đường ngoằn ngoèo đại diện cho ràng buộc.



2. **Resizing handle:** Để thay đổi kích thước phần tử, hãy kéo các bộ điều khiển thay đổi kích thước hình vuông. Tay cầm thay đổi thành góc cạnh trong khi bạn đang kéo it.

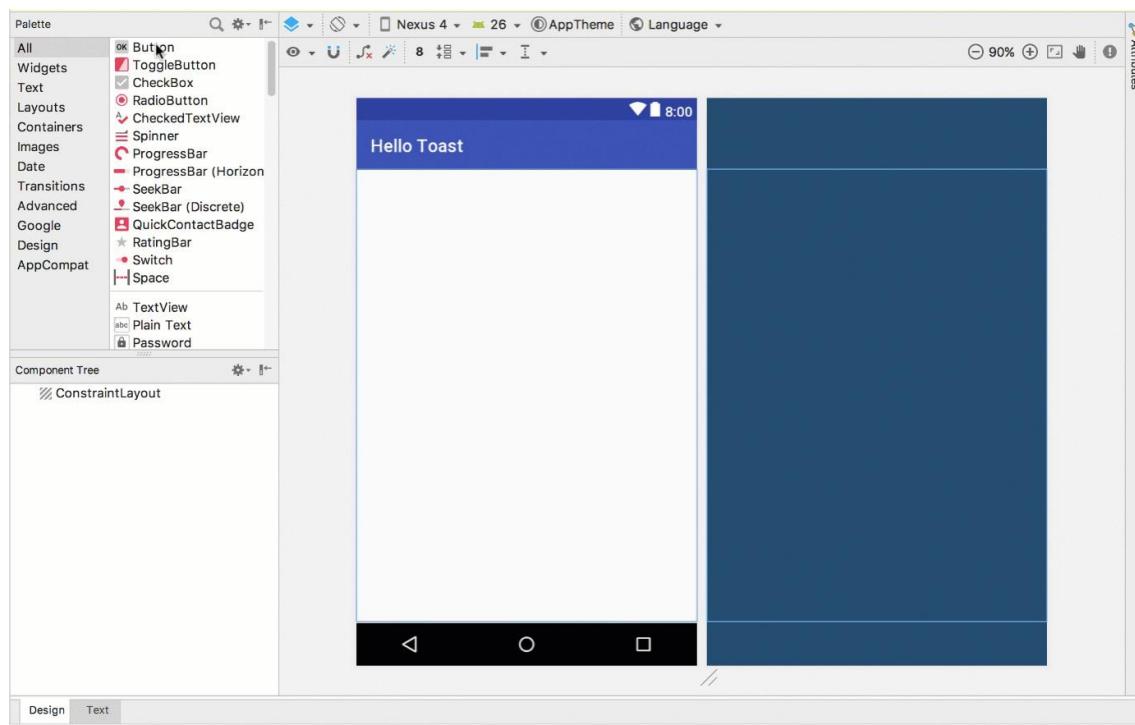


2.2 Kiểm tra các ràng buộc của phần tử

Khi được bật, công cụ **Tự động kết nối** sẽ tự động tạo hai hoặc nhiều ràng buộc cho một phần tử giao diện người dùng đối với bố cục mẹ. Sau khi bạn kéo phần tử vào bố cục, nó sẽ tạo ra các ràng buộc dựa trên vị trí của phần tử.

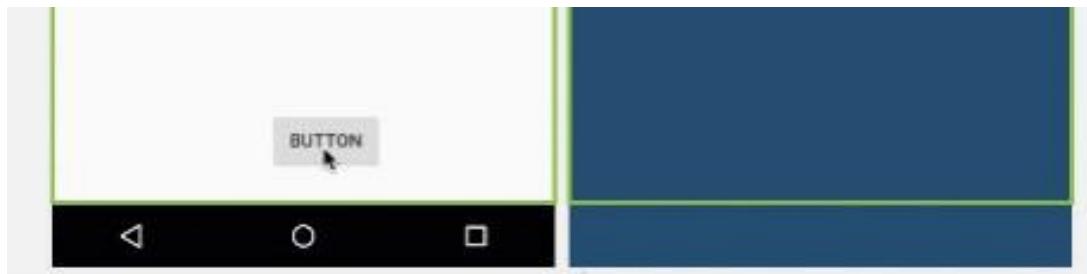
Làm theo các bước sau để thêm Nút:

1. Bắt đầu với một bảng sạch. Phần tử TextView không cần thiết, vì vậy trong khi nó vẫn được chọn, hãy nhấn phím Delete hoặc chọn **Edit > Delete**. Jetzt Sie haben eine leere Layout.
2. Kéo Nút từ ngăn **Bảng màu** đến bất kỳ vị trí nào trong bố cục. Nếu bạn thả Nút ở khu vực giữa trên cùng của bố cục, các ràng buộc có thể tự động xuất hiện. Nếu không, bạn có thể kéo các ràng buộc vào phía trên, bên trái và bên phải của bố cục như trong hình động bên dưới.



2.3 Thêm nút thứ hai vào bố cục

- 1 .Kéo một nút khác từ ngăn **Bảng màu** vào giữa bố cục như trong hình động bên dưới. Tự động kết nối có thể cung cấp các ràng buộc ngang cho bạn (nếu không, bạn có thể tự kéo chúng).
2. Kéo một ràng buộc dọc xuống cuối bố cục (tham khảo hình bên dưới).



3. Kéo một nút khác từ ngăn **Bảng màu** vào giữa bố cục như trong hình động bên dưới. Tự động kết nối có thể cung cấp các ràng buộc ngang cho bạn (nếu không, bạn có thể tự kéo chúng).
4. Kéo một ràng buộc dọc xuống cuối bố cục (tham khảo hình bên dưới).

Nhiệm vụ 3: Thay đổi thuộc tính phần tử giao diện người dùng

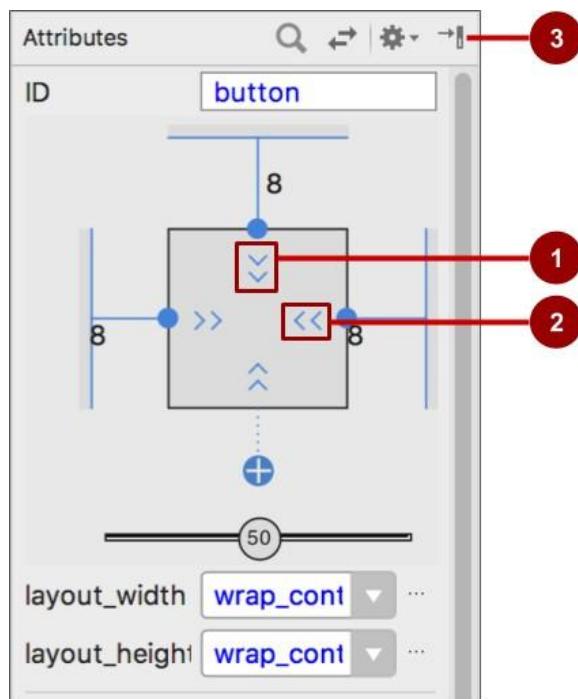
Ngăn **Attributes** cung cấp quyền truy cập vào tất cả các thuộc tính XML mà bạn có thể gán cho một phần tử giao diện người dùng. Bạn có thể tìm thấy các thuộc tính (được gọi là *thuộc tính*) chung cho tất cả các chế độ xem trong [tài liệu về lớp View](#).

Trong nhiệm vụ này, bạn nhập các giá trị mới và thay đổi giá trị cho các thuộc tính Button quan trọng, đó là applicable to most View types.

3.1 Thay đổi kích thước nút

Trình chỉnh sửa bối cảnh cung cấp các tay cầm thay đổi kích thước ở cả bốn góc của Chế độ xem để bạn có thể thay đổi kích thước Chế độ xem một cách nhanh chóng. Bạn có thể kéo các tay cầm trên mỗi góc của Chế độ xem để thay đổi kích thước, nhưng làm như vậy sẽ mã hóa cứng kích thước chiều rộng và chiều cao. Tránh mã hóa cứng kích thước cho hầu hết các thành phần Chế độ xem vì kích thước được mã hóa không thích ứng với các nội dung và kích thước màn hình khác nhau.

Thay vào đó, hãy sử dụng **ngăn Thuộc tính** ở phía bên phải của trình soạn thảo bối cảnh để chọn chế độ định cỡ không sử dụng kích thước được mã hóa cứng. Ngăn **Thuộc tính** bao gồm một bảng điều chỉnh kích thước hình vuông được gọi là trình *kiểm tra chế độ xem* ở trên cùng. Các ký hiệu bên trong hình vuông đại diện cho cài đặt chiều cao và chiều rộng như sau:



Trong hình trên:

1. **Kiểm soát độ cao.** Điều khiển này chỉ định thuộc tính layout_height và xuất hiện trong hai phân đoạn ở cạnh trên và dưới của hình vuông. Các góc chỉ ra rằng điều khiển này được đặt thành wrap_content, có nghĩa là View sẽ mở rộng theo chiều dọc khi cần thiết

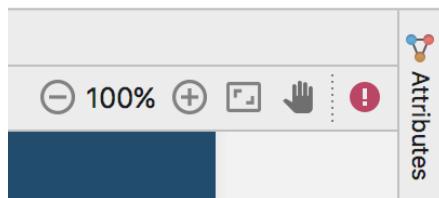
-
- để phù hợp với contents. Dấu "8" cho biết ký quỹ tiêu chuẩn được đặt thành 8dp.
2. **Kiểm soát chiều rộng.** Điều khiển này chỉ định layout_width và xuất hiện trong hai phân đoạn ở bên trái và bên phải của hình vuông. Các góc chỉ ra rằng điều khiển này được đặt thành wrap_content,

có nghĩa là Chế độ xem sẽ mở rộng theo chiều ngang khi cần thiết để phù hợp với nội dung của nó, lên đến biên độ 8dp.

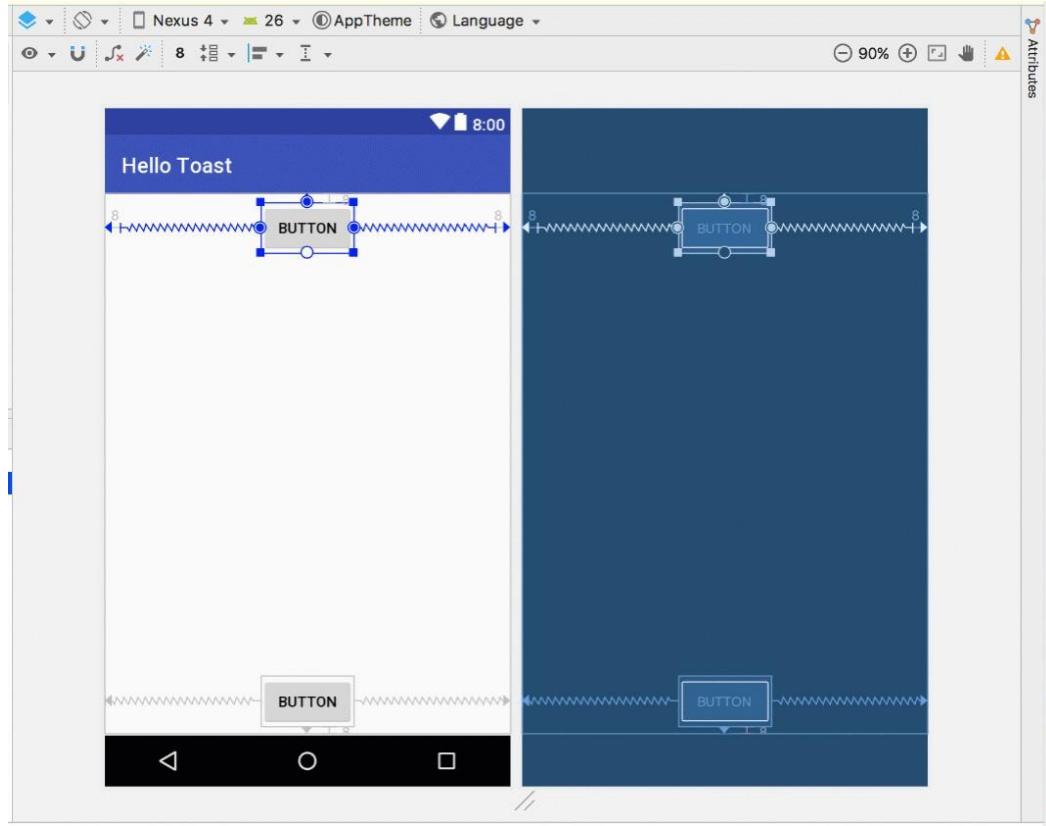
3. **Nút đóng ngăn Thuộc tính.** Bấm để đóng ngăn.

Làm theo các bước sau:

1. Chọn nút trên cùng trong ngăn **Cây thành phần**.
2. Nhấp vào tab **Attributes** ở phía bên phải của cửa sổ trình chỉnh sửa bố cục.

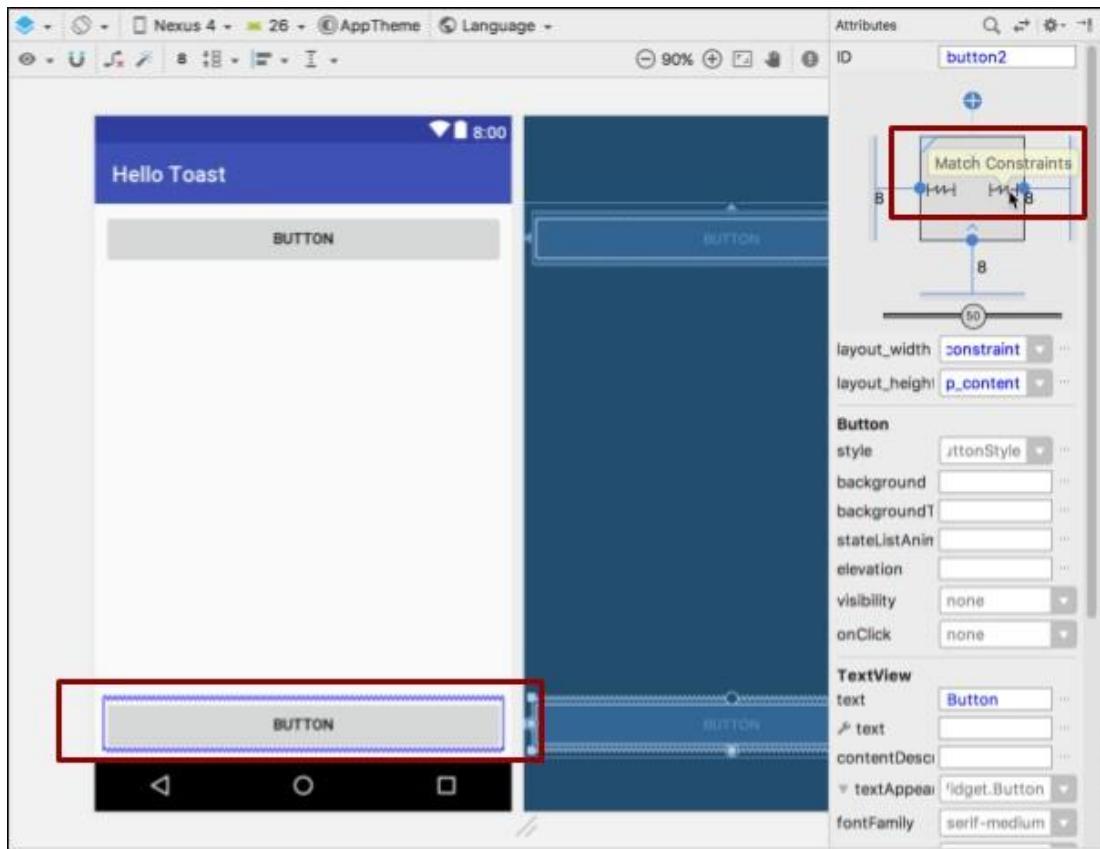


3. Nhấp vào điều khiển chiều rộng hai lần — lần nhấp đầu tiên thay đổi nó thành **Cố định** với các đường thẳng và lần nhấp thứ hai thay đổi nó thành **Match Constraints with spring coils**, như được hiển thị trong hình động bên dưới.



Kết quả của việc thay đổi điều khiển chiều rộng, thuộc tính `layout_width` trong **ngắn Thuộc tính** hiển thị giá trị `match_constraint` và phần tử Button kéo dài theo chiều ngang để lấp đầy khoảng trống giữa bên trái và bên phải của bố cục.

1. Chọn Nút thứ hai và thực hiện các thay đổi tương tự đối với `layout_width` như ở bước trước, như trong hình bên dưới.



Như được hiển thị trong các bước trước, các thuộc tính `layout_width` và `layout_height` trong **khung** **Thuộc tính** thay đổi khi bạn thay đổi các điều khiển chiều cao và chiều rộng trong trình kiểm tra. Các thuộc tính này có thể lấy một trong ba giá trị cho bối cảnh, đó là `ConstraintLayout`:

1. Cài đặt `match_constraint` mở rộng phần tử View để lấp đầy phần tử cha của nó theo chiều rộng hoặc chiều cao—lên đến lè, nếu được đặt. Cha mẹ trong trường hợp này là `ConstraintLayout`. Bạn tìm hiểu thêm về `ConstraintLayout` trong nhiệm vụ tiếp theo.
2. Cài đặt `wrap_content` thu nhỏ kích thước của phần tử View để nó vừa đủ lớn để bao bọc nội dung của nó. Nếu không có nội dung, phần tử View sẽ trở nên vô hình.
3. Để chỉ định kích thước cố định điều chỉnh theo kích thước màn hình của thiết bị, hãy sử dụng một số pixel cố định [không phụ thuộc vào mật độ](#) (đơn vị dp). Ví dụ: 16dp có nghĩa là 16 pixel không phụ thuộc vào mật độ .

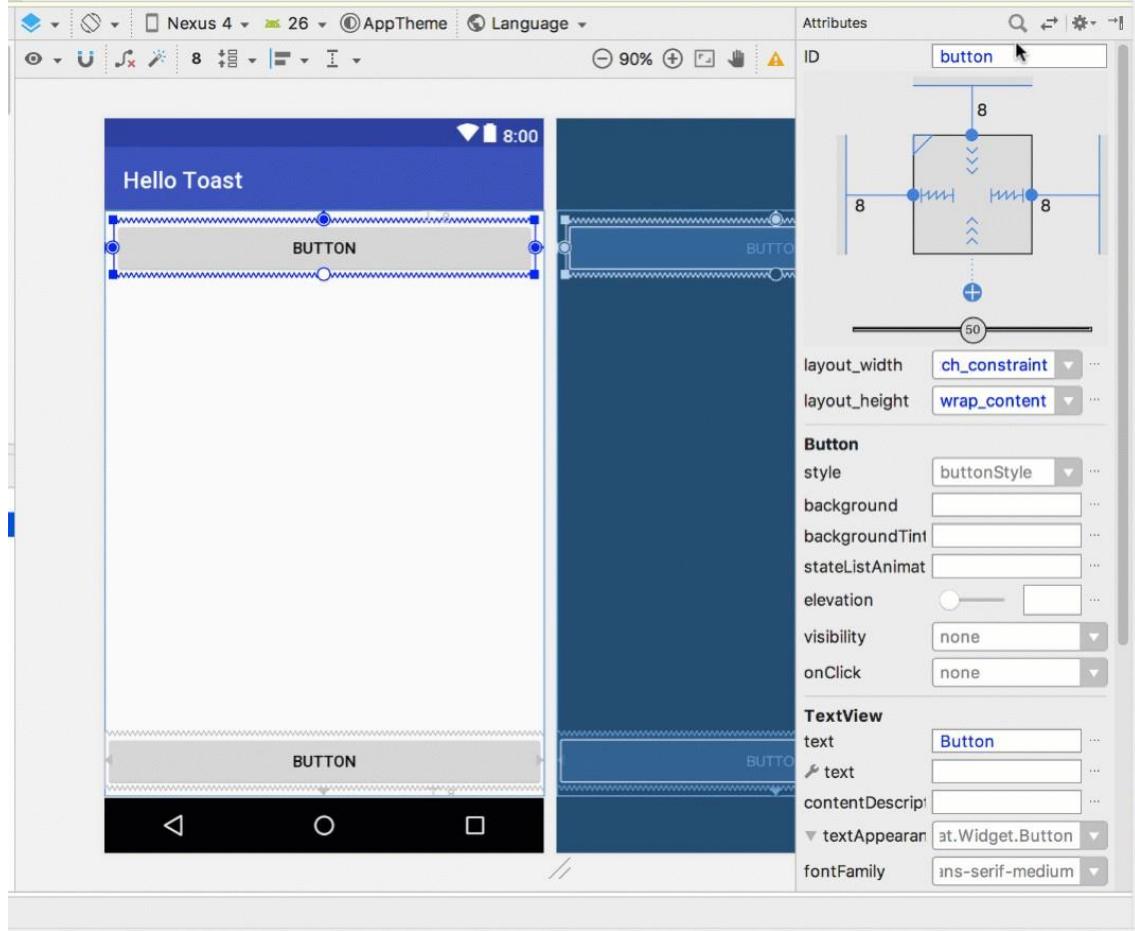
3.2 Thay đổi thuộc tính Nút

Để xác định mỗi Chế độ xem duy nhất trong bối cảnh Hoạt động, mỗi lớp con Chế độ xem hoặc Chế độ xem (chẳng hạn như Nút) cần có một ID duy nhất. Và để có bất kỳ công dụng nào, các phần tử Button cần văn bản. Các thành phần chế độ xem cũng có thể có hình nền có thể là màu sắc hoặc hình ảnh.

Ngăn Thuộc **tính** cung cấp quyền truy cập vào tất cả các thuộc tính mà bạn có thể gán cho phần tử Xem. Bạn có thể nhập giá trị cho từng thuộc tính, chẳng hạn như thuộc tính android:id, background, textColor và text.

Hình động sau đây minh họa cách thực hiện các bước này:

1. Sau khi chọn Nút đầu tiên, hãy chỉnh sửa trường ID ở đầu ngăn Thuộc **tính** để button_toast thuộc tính android:id, thuộc tính này được sử dụng để xác định phần tử trong bối cảnh.
2. Đặt thuộc tính background thành **@color/colorPrimary**. (Khi bạn nhập **@c**, các lựa chọn xuất hiện để dễ dàng lựa chọn.)
3. Đặt thuộc tính textColor thành **@android:color/white**.
4. Chính sửa thuộc tính text thành **Toast**.



1. Thực hiện các thay đổi thuộc tính tương tự cho Nút thứ hai, sử dụng **button_count** làm ID, Số **lượng** cho thuộc tính văn bản và cùng màu cho nền và văn bản như các bước trước.

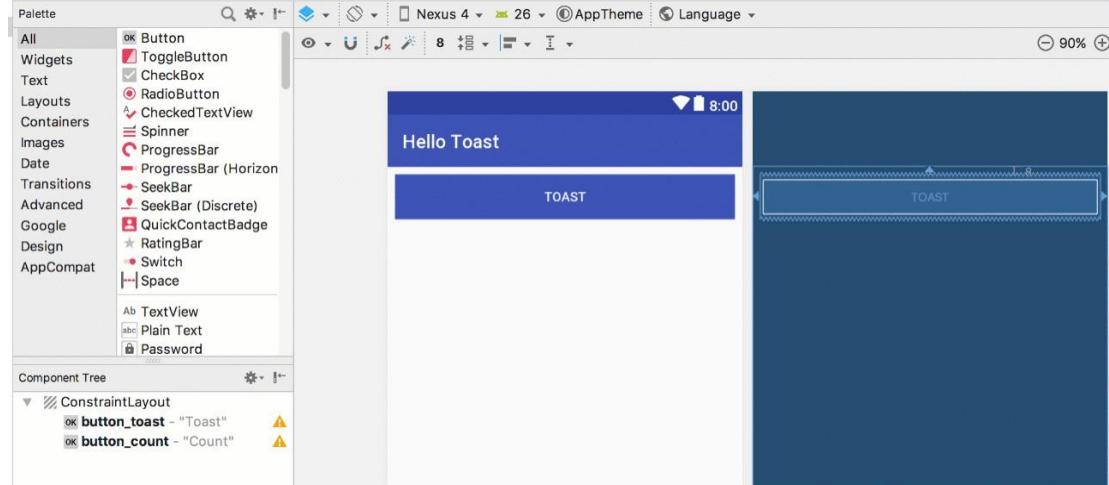
colorPrimary là màu chính của chủ đề, một trong những màu cơ sở chủ đề được xác định trước được xác định trong tệp tài nguyên colors.xml. Nó được sử dụng cho thanh ứng dụng. Sử dụng màu cơ bản cho các yếu tố giao diện người dùng khác sẽ tạo ra một giao diện người dùng thống nhất. Bạn sẽ tìm hiểu thêm về chủ đề ứng dụng và Material Design trong một bài học khác.

Nhiệm vụ 4: Thêm TextEdit và thiết lập các thuộc tính của nó

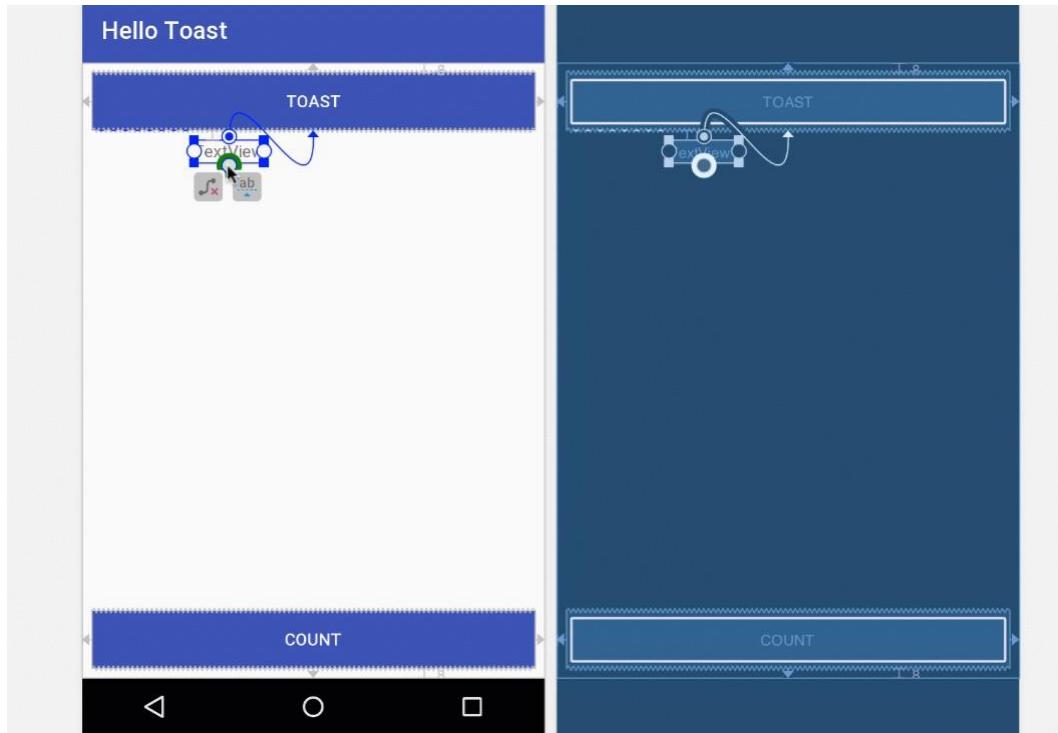
Một trong những lợi ích của [ConstraintLayout](#) là khả năng căn chỉnh hoặc hạn chế các phần tử liên quan đến các phần tử khác. Trong nhiệm vụ này, bạn sẽ thêm một TextView ở giữa bô cục và hạn chế nó theo chiều ngang vào lề và theo chiều dọc vào hai phần tử Button. Sau đó, bạn sẽ thay đổi các thuộc tính cho TextView trong ngăn Attributes (Thuộc tính).

4.1 Thêm TextView và các ràng buộc

1. Như được hiển thị trong hình động bên dưới, kéo TextView từ ngăn **Palette** vào phần trên của bô cục và kéo một ràng buộc từ đầu TextView đến tay cầm ở cuối Nút **Toast**. Điều này hạn chế TextView nằm bên dưới Nút.



2. Như trong hình động bên dưới, hãy kéo một ràng buộc từ cuối TextView đến tay cầm ở đầu **Nút đếm** và từ hai bên của TextView đến hai bên của bô cục. Điều này hạn chế TextView ở giữa bô cục giữa hai phần tử Button.

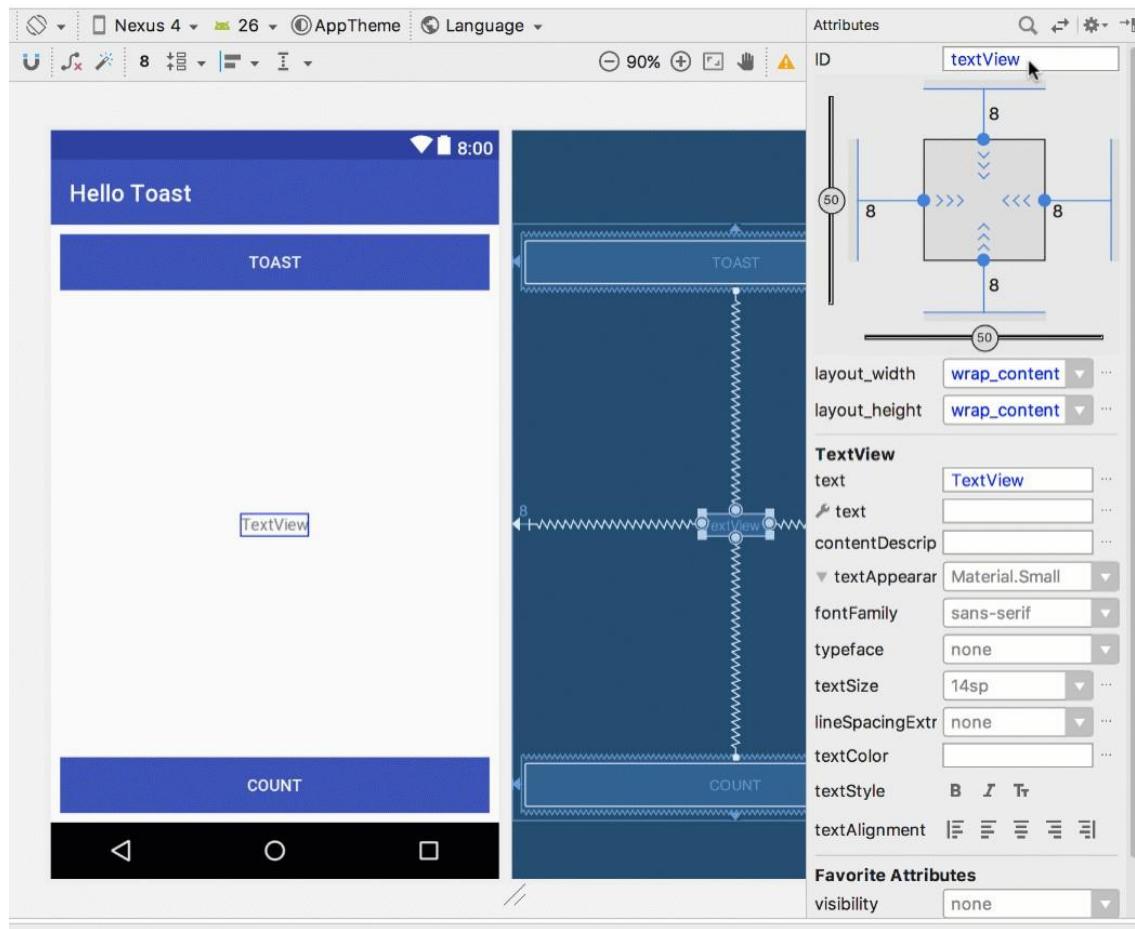


4.2 Đặt các thuộc tính TextView

Với TextView được chọn, hãy mở ngăn **Thuộc tính**, nếu nó chưa mở. Đặt thuộc tính cho TextView như trong hình động bên dưới. Các thuộc tính bạn chưa gấp được giải thích sau hình:

1. Đặt ID thành **show_count**.
2. Đặt văn bản thành **0**.
3. Đặt textSize thành **160sp**.
4. Đặt textStyle thành **B** (in đậm) và textAlign thành **ALIGNCENTER** (căn giữa đoạn văn).
5. Thay đổi các điều khiển kích thước ché độ xem ngang và dọc (**layout_width** và **layout_height**) thành **match_constraint**.
6. Đặt textColor thành **@color/colorPrimary**.

-
7. Cuộn xuống ngắn và bấm Xem **tất cả thuộc tính**, cuộn xuống trang thứ hai của các thuộc tính thành nền, sau đó nhập #FFF00 cho màu vàng.
 8. Cuộn xuống trọng lực, mở rộng trọng lực và chọn **center_ver** (đối với trung tâm-dọc).



9. **textSize**: Kích thước văn bản của TextView. Đối với bài học này, kích thước được đặt thành 160sp. sp là viết tắt của *pixel không phụ thuộc vào tỷ lệ và giống* như dp, là một đơn vị chia tỷ lệ theo mật độ màn hình và tùy chọn kích thước phông chữ của người dùng. Sử dụng đơn vị dp khi bạn chỉ định kích thước phông chữ để kích thước được điều chỉnh cho cả mật độ màn hình

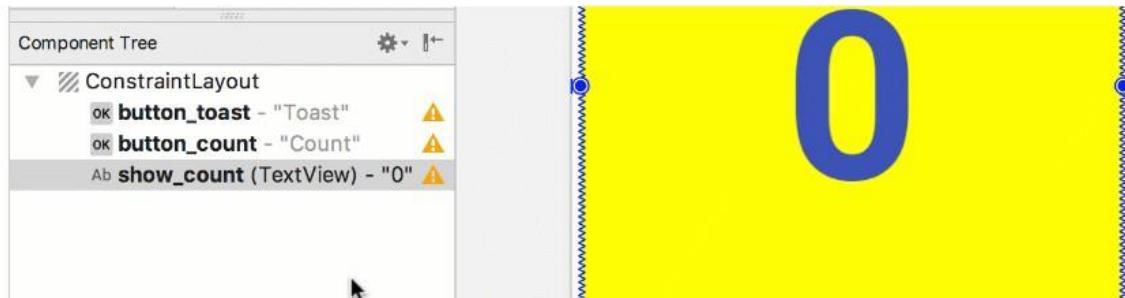
-
- và của người dùng preference.
10. **textStyle** và **textAlignment**: Kiểu văn bản, được đặt thành **B** (in đậm) trong bài học này và căn chỉnh văn bản, được đặt thành **ALIGNCENTER** (căn giữa đoạn văn).

-
11. trọng lực: Thuộc tính trọng lực chỉ định cách căn chỉnh Chế độ xem trong Chế độ xem hoặc Nhóm xem chính của nó. Trong bước này, bạn căn giữa TextView theo chiều dọc trong ConstraintLayout mẹ.

Bạn có thể nhận thấy rằng thuộc tính nền nằm trên trang đầu tiên của ngăn **Thuộc tính** cho Nút, nhưng trên trang thứ hai của ngăn **Thuộc tính** cho TextView. Ngăn **Thuộc tính** thay đổi cho từng loại Dạng xem: Các thuộc tính phổ biến nhất cho loại Dạng xem xuất hiện trên trang đầu tiên và phần còn lại được liệt kê trên trang thứ hai. Để quay lại trang đầu tiên của **ngăn Thuộc tính**, nhấn vào nút  trên thanh công cụ ở đầu ngăn.

Nhiệm vụ 5: Chính sửa bố cục trong XML

Bộ cục ứng dụng Hello Toast gần như đã hoàn thành! Tuy nhiên, một dấu chấm than xuất hiện bên cạnh mỗi phần tử giao diện người dùng trong Cây thành phần. Di con trỏ của bạn qua các dấu chấm than này để xem thông báo cảnh báo, như được hiển thị bên dưới. Cảnh báo tương tự xuất hiện cho cả ba phần tử: các chuỗi được mã hóa cứng nên sử dụng tài nguyên.



Cách dễ nhất để khắc phục sự cố bố cục là chỉnh sửa bố cục trong XML. Mặc dù trình soạn thảo bố cục là một công cụ mạnh mẽ, nhưng một số thay đổi dễ dàng thực hiện trực tiếp trong mã nguồn XML.

5.1 Mở mã XML cho bố cục

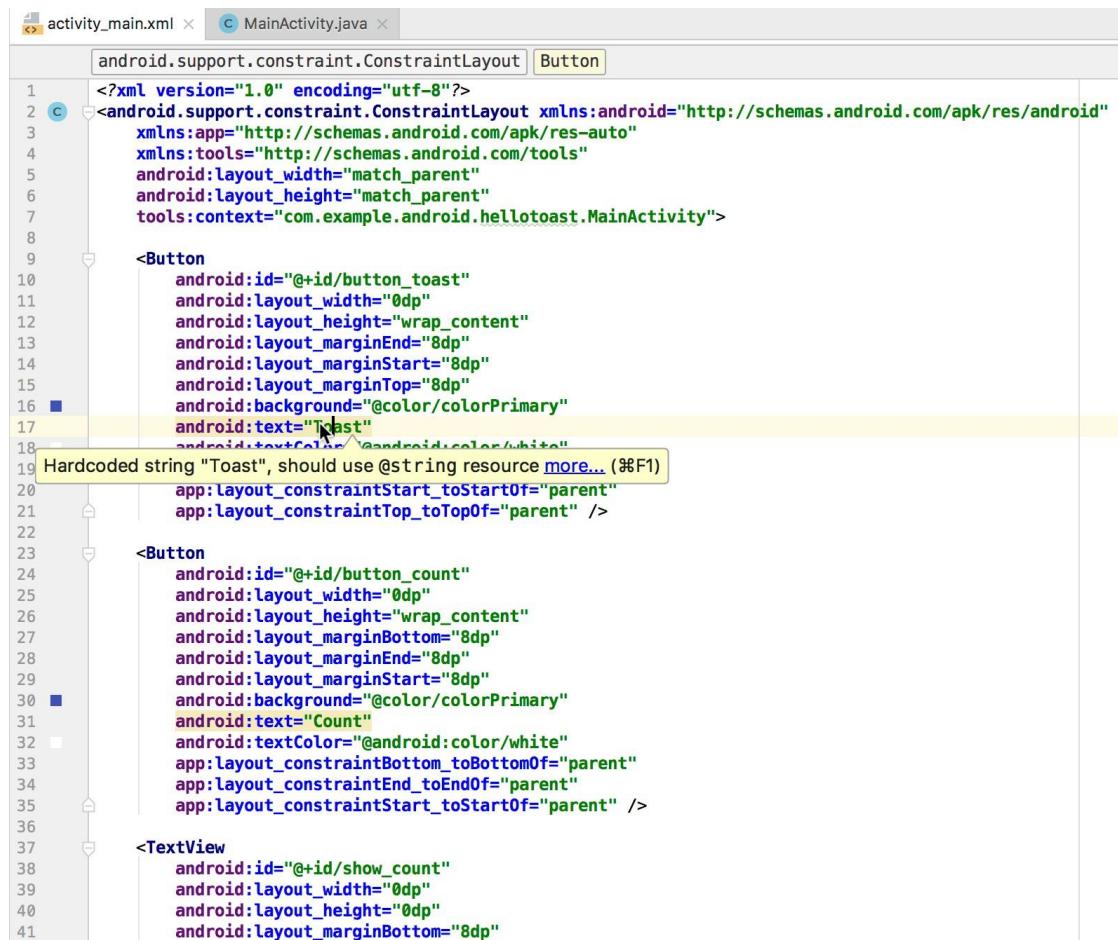
Đối với tác vụ này, hãy mở tệp activity_main.xml nếu nó chưa được mở và nhấp vào **tab Văn bản**

Design

Text

ở cuối trình chỉnh sửa bố cục.

Trình soạn thảo XML xuất hiện, thay thế các ngăn thiết kế và bản thiết kế. Như bạn có thể thấy trong hình bên dưới, cho thấy một phần của mã XML cho bố cục, các cảnh báo được đánh dấu - các chuỗi được mã hóa cứng "Toast" và "Count". (Mã hóa cứng "0" cũng được đánh dấu nhưng không được hiển thị trong con số.) Di con trỏ của bạn lên chuỗi được mã hóa cứng "Toast" để xem thông báo cảnh báo.



```
activity_main.xml x MainActivity.java x
[Android Support ConstraintLayout] [Button]

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout 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"
    tools:context="com.example.android.hellotoast.MainActivity">

    <Button
        android:id="@+id/button_toast"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:background="@color/colorPrimary"
        android:text="Toast"
        android:textColor="@android:color/white" />
    Hardcoded string "Toast", should use @string resource more... (#F1)
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button_count"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:background="@color/colorPrimary"
        android:text="Count"
        android:textColor="@android:color/white"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
```

5.2 Trích xuất tài nguyên chuỗi

Thay vì mã hóa cứng chuỗi, cách tốt nhất là sử dụng tài nguyên chuỗi, đại diện cho các chuỗi. Việc có các chuỗi trong một tệp riêng biệt giúp quản lý chúng dễ dàng hơn, đặc biệt nếu bạn sử dụng các chuỗi này nhiều lần. Ngoài ra, tài nguyên chuỗi là bắt buộc để dịch và bản địa hóa ứng dụng của bạn, vì bạn cần tạo tệp tài nguyên chuỗi cho từng ngôn ngữ.

1. Nhập một lần vào từ "Toast" (cảnh báo được đánh dấu đầu tiên).
2. Nhấn **Alt-Enter** trong Windows hoặc **Option-Enter** trong macOS và chọn **Trích xuất tài nguyên chuỗi** từ menu bật lên.
3. Nhập **button_label_toast** cho **Tên tài nguyên**.
4. Nhập vào OK. Tài nguyên chuỗi được tạo trong tệp values/res/string.xml và chuỗi trong mã của bạn được thay thế bằng tham chiếu đến tài nguyên: @string / button_label_toast
5. Trích xuất các chuỗi còn lại: button_label_count cho "Count" và count_initial_value cho "0".
6. Trong ngăn **Project > Android**, hãy mở rộng **các giá trị** trong **res**, sau đó nhấp đúp vào **strings.xml**. Để xem tài nguyên chuỗi của bạn trong tệp strings.xml:

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

7. Bạn cần một chuỗi khác để sử dụng trong tác vụ tiếp theo hiển thị thông báo. Thêm vào tệp strings.xml một tài nguyên chuỗi khác có tên toast_message cho cụm từ "Hello Toast!":

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

```
</resources>
```

Mẹo: Tài nguyên chuỗi bao gồm tên ứng dụng, tên này xuất hiện trong thanh ứng dụng ở đầu màn hình nếu bạn bắt đầu dự án ứng dụng của mình bằng Mẫu trống. Bạn có thể thay đổi tên ứng dụng bằng cách chỉnh sửa tài nguyên app_name.

Nhiệm vụ 6: Thêm trình xử lý onClick cho các nút

Trong tác vụ này, bạn thêm một phương thức Java cho mỗi Button trong MainActivity thực thi khi người dùng nhấp vào Button.

6.1 Thêm thuộc tính và trình xử lý onClick vào mỗi Button

Trình xử lý nhấp chuột là một phương thức được gọi khi người dùng nhấp hoặc nhấn vào phần tử giao diện người dùng có thể nhấp vào. Trong Android Studio, bạn có thể chỉ định tên của phương thức trong trường onClick trong **ngắn Attributes** của tab **Design**. Bạn cũng có thể chỉ định tên của phương thức xử lý trong trình soạn thảo XML bằng cách thêm thuộc tính android:onClick vào Button. Bạn sẽ sử dụng phương thức thứ hai vì bạn chưa tạo các phương thức xử lý và trình soạn thảo XML cung cấp một cách tự động để tạo các phương thức đó.

1. Với trình soạn thảo XML đang mở (tab Text), hãy tìm Button có android:id được đặt thành button_toast:

```
<Button  
    android:id="@+id/button_toast"  
    android:layout_width="0dp"  
    ...  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

-
2. Thêm thuộc tính android:onClick vào cuối phần tử button_toast sau thuộc tính cuối cùng và trước chỉ báo kết thúc />:

```
    android:onClick="showToast" />
```

3. Nhập vào biểu tượng bóng đèn màu đỏ xuất hiện bên cạnh thuộc tính. Chọn **Tạo trình xử lý nhấp chuột**, chọn **MainActivity** và bấm OK.

Nếu biểu tượng bóng đèn màu đỏ không xuất hiện, hãy nhập vào tên phương thức ("showToast"). Nhấn **Alt-Enter**

(Option-Enter trên máy Mac), chọn **Tạo 'showToast(view)' trong MainActivity** và nhấp vào **OK**.

Hành động này tạo sơ khai phương thức giữ chỗ cho phương thức showToast() trong MainActivity, như được hiển thị ở cuối các bước này.

4. Lặp lại hai bước cuối cùng với Nút button_count: Thêm thuộc tính android:onClick vào cuối và thêm trình xử lý nhấp chuột:

```
    android:onClick="countUp" />
```

Mã XML cho các phần tử giao diện người dùng trong ConstraintLayout bây giờ trông như sau:

```
<Button  
    android:id="@+id/button_toast"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginEnd="8dp"  
    android:layout_marginStart="8dp"  
    android:layout_marginTop="8dp"  
    android:background="@color/colorPrimary"  
    android:text="@string/button_label_toast"  
    android:textColor="@android:color/white"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintTop_toTopOf="parent"
    android:onClick="showToast"/>

<Button
    android:id="@+id/button_count"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:background="@color/colorPrimary"
    android:text="@string/button_label_count"
    android:textColor="@android:color/white"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:onClick="countUp" />

<TextView
    android:id="@+id/show_count"
    android:layout_width="0dp" android:layout_height="0dp"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:background="#FFFF00"
    android:gravity="center_vertical"
    android:text="@string/count_initial_value"
    android:textAlignment="center"
    android:textColor="@color/colorPrimary"
    android:textSize="160sp" android:textStyle="bold"
    app:layout_constraintBottom_toTopOf="@+id/button_count"
    app:layout_constraintEnd_toEndOf="parent" app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button_toast" />
```

5. Nếu MainActivity.java chưa mở, hãy mở rộng **java** trong chế độ xem Project > Android, mở rộng **com.example.android.hellotoast**, sau đó nhấp đúp vào **MainActivity**. Trình soạn thảo mã xuất hiện cùng với mã trong MainActivity:

```
package com.example.android.hellotoast;
```

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

public class MainActivity extends AppCompatActivity {

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

    public void showToast(View view) {
    }

    public void countUp(View view) {
    }
}
```

6.2 Chính sửa trình xử lý Nút Toast

Bây giờ bạn sẽ chỉnh sửa phương thức showToast() — trình xử lý nhấp chuột Toast Button trong MainActivity — để nó hiển thị một thông báo. Toast cung cấp một cách để hiển thị một thông báo đơn giản trong một cửa sổ bật lên nhỏ. Nó chỉ lấp đầy dung lượng cần thiết cho tin nhắn. Hoạt động hiện tại vẫn hiển thị và tương tác. Toast có thể hữu ích để kiểm tra tính tương tác trong ứng dụng của bạn—thêm thông báo Toast để hiển thị kết quả nhấn vào Nút hoặc thực hiện một hành động.

Làm theo các bước sau để chỉnh sửa trình xử lý nhấp chuột Nút Toast:

1. Xác định vị trí phương thức showToast() mới được tạo.

```
public void showToast(View view) {
```

-
- Để tạo một thực thể của Toast, hãy gọi [phương thức nhà máy makeText\(\)](#) trên lớp [Toast](#).

```
public void showToast(View view) {  
    Toast toast = Toast.makeText(  
}
```

Tuyên bố này không đầy đủ cho đến khi bạn hoàn thành tất cả các bước.

3. Cung cấp ngữ cảnh của Hoạt động ứng dụng. Vì Toast hiển thị trên đầu giao diện người dùng Hoạt động, hệ thống cần thông tin về Hoạt động hiện tại. Khi bạn đã ở trong ngữ cảnh của Hoạt động mà bạn cần, hãy sử dụng nó như một phím tắt.

```
Toast toast = Toast.makeText(this,
```

4. Cung cấp thông báo để hiển thị, chẳng hạn như tài nguyên chuỗi (toast_message bạn đã tạo ở bước trước). Tài nguyên chuỗi toast_message được xác định bởi R.string.

```
Toast toast = Toast.makeText(this, R.string.toast_message,
```

5. Cung cấp thời lượng cho màn hình. Ví dụ: Toast.LENGTH_SHORT hiển thị bánh mì nướng trong một thời gian tương đối ngắn.

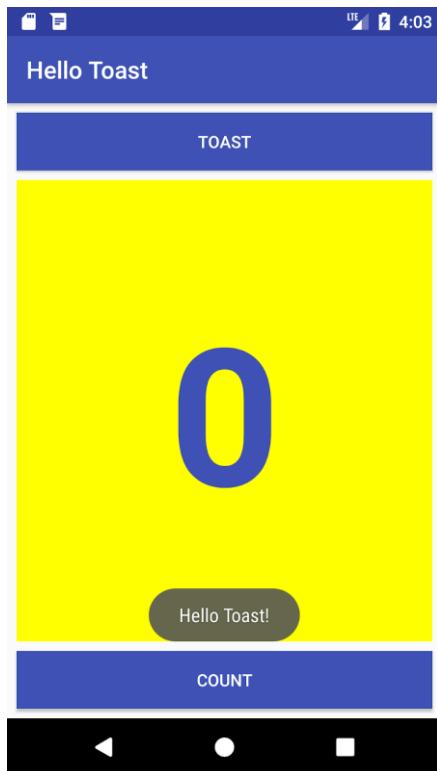
```
Toast toast = Toast.makeText(this, R.string.toast_message,  
                           Toast.LENGTH_SHORT);
```

Thời lượng của màn hình Toast có thể là Toast.LENGTH_LONG hoặc Toast.LENGTH_SHORT. Độ dài thực tế là khoảng 3.5 giây đối với Bánh mì nướng dài và 2 giây đối với Bánh mì nướng ngắn.

6. Hiển thị Toast bằng cách gọi show(). Sau đây là toàn bộ phương thức showToast():

```
public void showToast(View view) {  
    Toast toast = Toast.makeText(this, R.string.toast_message,  
                                Toast.LENGTH_SHORT);  
    toast.show();  
}
```

Chạy ứng dụng và xác minh rằng thông báo Toast xuất hiện khi **nhấn vào nút** Toast.



6.3 Chỉnh sửa trình xử lý Nút đếm

Bây giờ bạn sẽ chỉnh sửa phương thức countUp() — trình **xử lý nhấp chuột Nút đếm** trong **MainActivity** — để nó hiển thị số lượng hiện tại sau khi Đếm được chạm vào. Mỗi lần chạm sẽ tăng số lượng lên một.

Mã cho trình xử lý phải:

- Theo dõi số lượng khi nó thay đổi.
- Gửi số lượng đã cập nhật đến TextView để hiển thị nó.

Làm theo các bước sau để chỉnh sửa **trình xử lý nhấp chuột Nút đếm**:

1. Xác định vị trí phương thức countUp() mới tạo.

```
public void countUp(View view) {  
}
```

2. Để theo dõi số lượng, bạn cần một biến thành viên riêng. Mỗi lần nhấn vào nút **Đếm** sẽ tăng giá trị của biến này. Nhập thông tin sau, sẽ được đánh dấu bằng màu đỏ và hiển thị biểu tượng bóng đèn màu đỏ:

```
public void countUp(View view) {  
    mCount++;  
}
```

Nếu biểu tượng bóng đèn màu đỏ không xuất hiện, hãy chọn biểu thức mCount++. Bóng đèn màu đỏ cuối cùng cũng xuất hiện.

3. Nhấp vào biểu tượng bóng đèn màu đỏ và chọn **Tạo trường 'mCount'** từ menu bật lên. Thao tác này sẽ tạo ra một biến thành viên riêng tư ở đầu MainActivity và Android Studio giả định rằng bạn muốn biến đó là một số nguyên (int):

```
public class MainActivity extends AppCompatActivity {  
    private int mCount;
```

-
4. Thay đổi câu lệnh biến thành viên riêng để khởi tạo biến về không:

```
public class MainActivity extends AppCompatActivity {  
    private int mCount = 0;
```

5. Cùng với biến trên, bạn cũng cần một biến thành viên riêng để tham chiếu TextView show_count, bạn sẽ thêm vào trình xử lý nhấp chuột. Gọi biến này là mShowCount:

```
public class MainActivity extends AppCompatActivity {  
    private int mCount = 0;  
    private TextView mShowCount;
```

6. Nay bạn đã có mShowCount, bạn có thể lấy tham chiếu đến TextView bằng cách sử dụng ID bạn đã đặt trong tệp bố cục. Để có được tham chiếu này chỉ một lần, hãy chỉ định nó trong phương thức onCreate(). Như bạn đã học trong một bài học khác, phương thức [onCreate\(\)](#) được sử dụng để *thổi phồng bố cục*, có nghĩa là đặt chế độ xem nội dung của màn hình thành bố cục XML. Bạn cũng có thể sử dụng tính năng này để lấy tham chiếu đến các thành phần giao diện người dùng khác trong bố cục, chẳng hạn như TextView. Tìm phương thức onCreate() trong MainActivity:

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

7. Thêm câu lệnh [findViewById](#) vào cuối phương thức:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mShowCount = (TextView) findViewById(R.id.show_count);  
}
```

View, giống như chuỗi, là một tài nguyên có thể có id. Lệnh gọi [findViewById](#) lấy ID của ché độ xem làm tham số và trả về View. Vì phương thức trả về View, bạn phải truyền kết quả sang kiểu view mà bạn mong đợi, trong trường hợp này là (TextView).

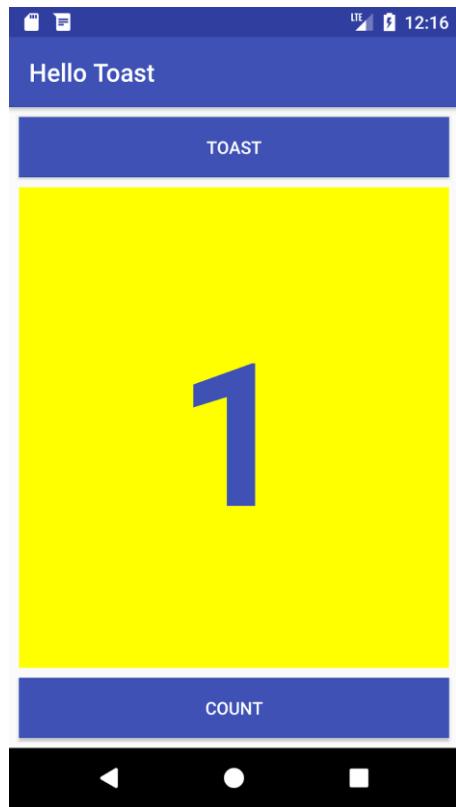
8. Bây giờ bạn đã gán cho mShowCount TextView, bạn có thể sử dụng biến để đặt văn bản trong TextView thành giá trị của biến mCount. Thêm thông tin sau vào phương thức countUp():

```
if (mShowCount != null)  
    mShowCount.setText(Integer.toString(mCount));
```

Toàn bộ phương thức countUp() bây giờ trông như thế này:

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

9. Chạy ứng dụng để xác minh rằng số lượng tăng lên khi bạn chạm vào **nút Đếm**.



Mẹo: Để biết hướng dẫn chuyên sâu về cách sử dụng ConstraintLayout, hãy xem Lớp học lập trình [Sử dụng ConstraintLayout để thiết kế chế độ xem](#).

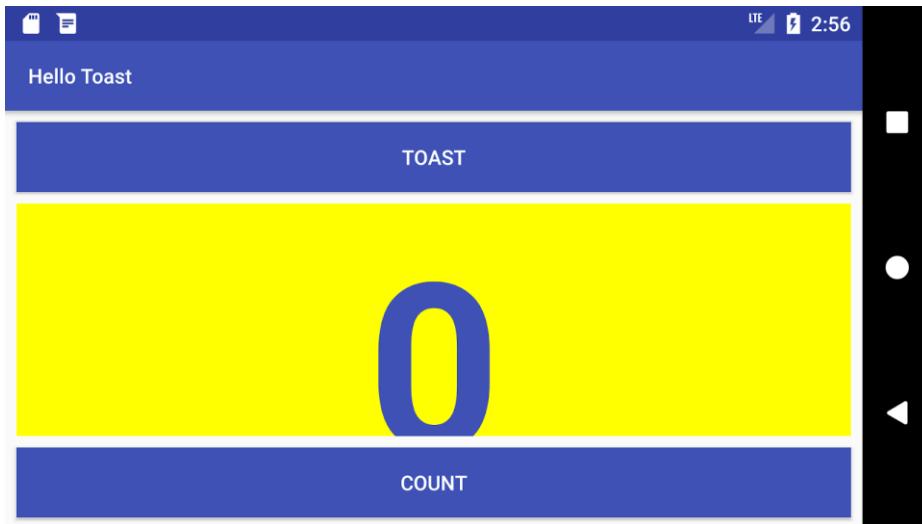
Mã giải pháp

Dự án Android Studio: [HelloToast](#)

Thử thách mã hóa

Lưu ý: Tất cả các thử thách mã hóa đều là tùy chọn và không phải là điều kiện tiên quyết cho các bài học sau này.

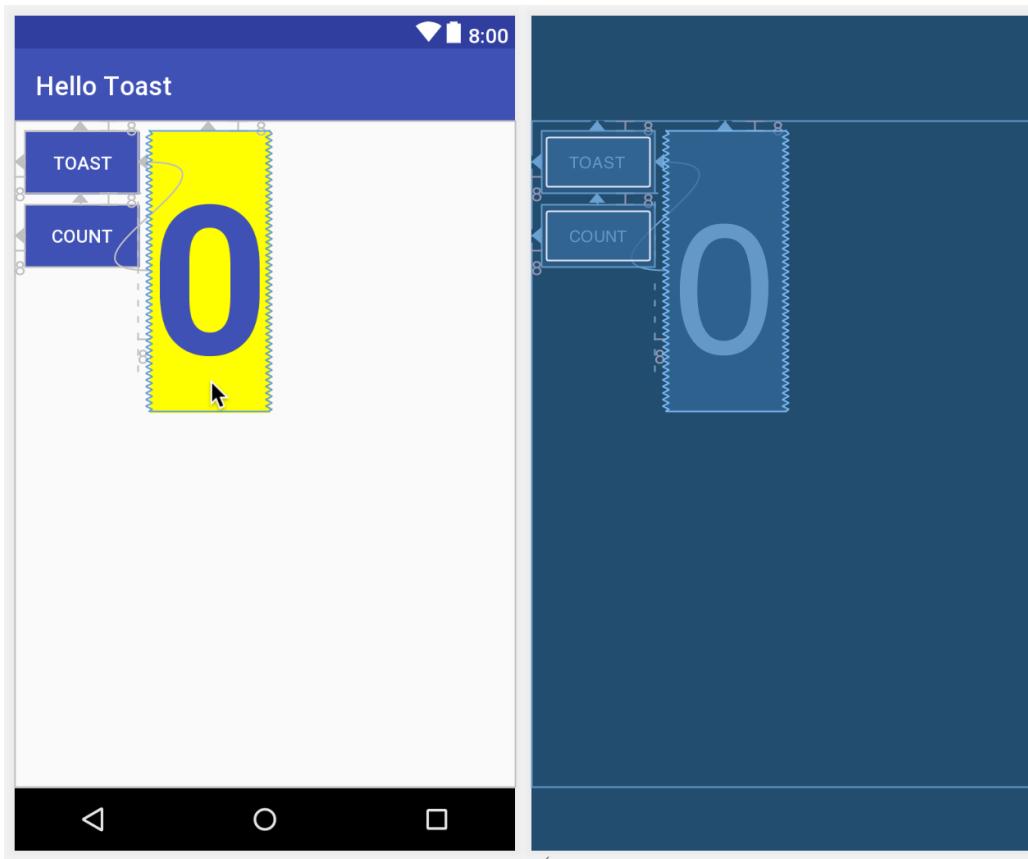
Ứng dụng HelloToast trông ổn khi thiết bị hoặc trình mô phỏng được định hướng theo chiều dọc. Tuy nhiên, nếu bạn chuyển thiết bị hoặc trình mô phỏng sang hướng ngang, Nút **đếm** có thể chòng lên TextView dọc theo phía dưới như trong hình bên dưới.



Thử thách: Thay đổi bố cục sao cho nó trông đẹp ở cả hướng ngang và dọc:

1. Trên máy tính của bạn, tạo một bản sao của **thư mục dự án HelloToast và đổi tên thành Xin chàoToastChallenge**.
2. Mở **HelloToastChallenge** trong Android Studio và tái cấu trúc nó. (Xem [Phu lục: Các tiện ích](#) để hướng dẫn sao chép và tái cấu trúc dự án.)
3. Thay đổi bố cục sao cho Nút **Toast** và Nút **đếm** xuất hiện ở phía bên trái, như trong hình bên dưới. TextView xuất hiện bên cạnh chúng, nhưng chỉ đủ rộng to show its nội

-
- dung. (Gợi ý: Sử dụng wrap_content.)
4. Chạy ứng dụng theo cả hướng ngang và dọc.





Mã giải pháp thách thức

Dự án Android Studio: [HelloToastChallenge](#)

Tóm tắt

View, ViewGroup và bố cục:

1. Tất cả các phần tử giao diện người dùng là các lớp con của [lớp View](#) và do đó kế thừa nhiều thuộc tính của lớp Xem siêu đăng cấp.
2. Các phần tử Ché độ xem có thể được nhóm lại bên trong ViewGroup, hoạt động như một vùng chứa. Mỗi quan hệ là cha-con, trong đó cha là ViewGroup và con là View hoặc ViewGroup khác.

-
1. Phương [thức onCreate\(\)](#) được sử dụng để *thổi phồng bố cục*, có nghĩa là đặt chế độ xem nội dung của màn hình thành bố cục XML. Bạn cũng có thể sử dụng nó để lấy tham chiếu đến các thành phần giao diện người dùng khác trong bố cục.
 2. View, giống như chuỗi, là một tài nguyên có thể có id. Lệnh gọi [findViewById](#) lấy ID của chế độ xem làm tham số và trả về View.

Sử dụng trình chỉnh sửa bố cục:

1. Nhập vào tab **Thiết kế** để thao tác với các phần tử và bố cục, và tab **Văn bản** để chỉnh sửa mã XML cho bố cục.
2. Trong tab **Thiết kế**, ngăn **Bảng màu** hiển thị các thành phần giao diện người dùng mà bạn có thể sử dụng trong bố cục của ứng dụng và ngăn **cây Thành phần** hiển thị hệ thống phân cấp chế độ xem của các phần tử giao diện người dùng.
3. Các ngăn thiết kế và bản thiết kế của trình chỉnh sửa bố cục hiển thị các yếu tố giao diện người dùng trong bố cục.
4. Tab **Thuộc tính** hiển thị ngăn **Thuộc tính** để đặt thuộc tính cho phần tử giao diện người dùng.
5. Bộ điều khiển ràng buộc: Nhập vào một bộ điều khiển ràng buộc, được hiển thị dưới dạng hình tròn ở mỗi bên của một phần tử, sau đó kéo đến một bộ điều khiển ràng buộc khác hoặc đến ranh giới cha để tạo một ràng buộc. Ràng buộc được biểu thị bằng đường zigzag.
6. Thay đổi kích thước bộ điều khiển: Bạn có thể kéo các bộ điều khiển thay đổi kích thước hình vuông để thay đổi kích thước phần tử. Trong khi kéo, tay cầm thay đổi thành một góc nghịch.
7. Khi được bật, công cụ **Tự động kết nối** sẽ tự động tạo hai hoặc nhiều ràng buộc cho một phần tử giao diện người dùng đối với bố cục mẹ. Sau khi bạn kéo phần tử vào bố cục, nó sẽ tạo ra các ràng buộc dựa trên vị trí của phần tử.
8. Bạn có thể xóa các ràng buộc khỏi một phần tử bằng cách chọn phần tử và di con trỏ lên phần tử đó để hiển thị nút Xóa ràng buộc . Nhấp vào nút này để loại bỏ tất cả các ràng buộc trên phần tử đã chọn. Để xóa một ràng buộc duy nhất, hãy nhấp vào tay cầm cụ thể đặt ràng buộc.
9. Ngăn **Attributes** cung cấp quyền truy cập vào tất cả các thuộc tính XML

mà bạn có thể gán cho một phần tử giao diện người dùng. Nó cũng bao gồm một bảng điều khiển kích thước hình vuông được gọi là view *inspector* ở trên cùng. Các ký hiệu bên trong hình vuông đại diện cho cài đặt chiều cao và chiều rộng.

Đặt chiều rộng và chiều cao bối cục:

The layout_width and layout_height attributes change as you change the height and width size controls in the view inspector. These attributes can take one of three values for a ConstraintLayout:

- The match_constraint setting expands the view to fill its parent by width or height—up to a margin, if one is set.
- The wrap_content setting shrinks the view dimensions so the view is just big enough to enclose its content. If there is no content, the view becomes invisible.
- Use a fixed number of dp ([density-independent pixels](#)) to specify a fixed size, adjusted for the screen size of the device.

Extracting string resources:

Instead of hard-coding strings, it is a best practice to use string resources, which represent the strings. Follow these steps:

1. Click once on the hardcoded string to extract, press **Alt-Enter** (**Option-Enter** on the Mac), and choose **Extract string resources** from the popup menu.
2. Set the **Resource name**.
3. Click **OK**. This creates a string resource in the values/res/string.xml file, and the string in your code is replaced with a reference to the resource: @string/button_label_toast

Handling clicks:

- A *click handler* is a method that is invoked when the user clicks or taps on a UI element.
- Specify a click handler for a UI element such as a Button by entering its name in the onClick field in the **Design** tab's **Attributes** pane, or in the XML editor by adding the android:onClick property to a UI element such as a Button.
- Create the click handler in the main Activity using the View parameter. Example: public void showToast(View view) { ... }.
- You can find information on all Button properties in the [Button class documentation](#), and all the TextView properties in the [TextView class documentation](#).

Displaying Toast messages:

A [Toast](#) provides a way to show a simple message in a small popup window. It fills only the amount of space required for the message. To create an instance of a Toast, follow these steps:

1. Call the [makeText\(\)](#) factory method on the [Toast](#) class.
2. Supply the [context](#) of the app Activity and the message to display (such as a string resource).
3. Supply the duration of the display, for example [Toast.LENGTH_SHORT](#) for a short period. The duration can be either `Toast.LENGTH_LONG` or `Toast.LENGTH_SHORT`.
4. Show the [Toast](#) by calling [show\(\)](#).

Related concept

The related concept documentation is in [1.2: Layouts and resources for the UI](#).

Learn more

Android developer documentation:

- [Android Studio](#)
- [Build a UI with Layout Editor](#)
- [Build a Responsive UI with ConstraintLayout](#)
- [Layouts](#)
- [View](#)
- [Button](#)
- [TextView](#)
- [Android resources](#)
- [Android standard R.color resources](#)
- [Supporting Different Densities](#)
- [Android Input Events](#)
- [Context](#)

Other:

- Codelabs: [Using ConstraintLayout to design your views](#)
- [Vocabulary words and concepts glossary](#)

The next codelab is [Android fundamentals 1.2 Part B: The layout editor](#).

Lesson 1.2 Part B: The layout editor

Introduction

As you learned in [1.2 Part A: Your first interactive UI](#), you can build a user interface (UI) using [ConstraintLayout](#) in the layout editor, which places UI elements in a layout using constraint connections to other elements and to the layout edges. ConstraintLayout was designed to make it easy to drag UI elements into the layout editor.

ConstraintLayout is a [ViewGroup](#), which is a special View that can contain other View objects (called *children* or *child views*). This practical shows more features of ConstraintLayout and the layout editor.

This practical also introduces two other [ViewGroup](#) subclasses:

- [LinearLayout](#): A group that aligns child View elements within it horizontally or vertically.
- [RelativeLayout](#): A group of child View elements in which each View element is positioned and aligned relative to other View element within the ViewGroup. Positions of the child View elements are described in relation to each other or to the parent ViewGroup.

What you should already know

You should be able to:

- Create a Hello World app with Android Studio.
- Run an app on the emulator or a device.
- Create a simple layout for an app with ConstraintLayout.
- Extract and use string resources.

What you'll learn

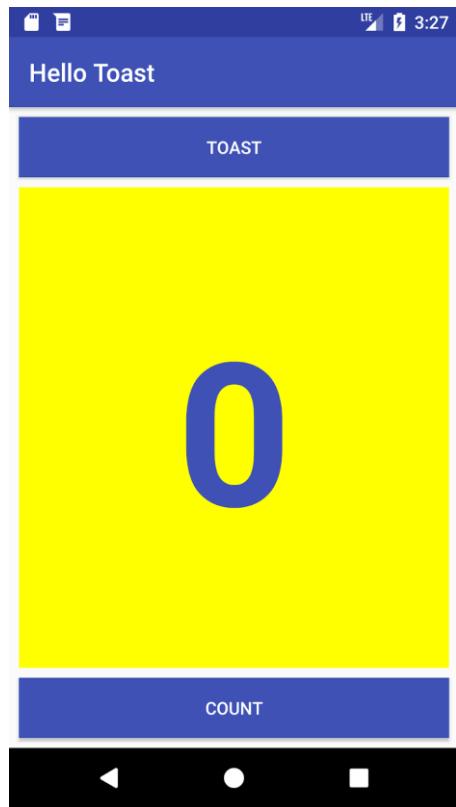
- How to create a layout variant for horizontal (landscape) orientation.
- How to create a layout variant for tablets and larger displays.
- How to use a baseline constraint to align UI elements with text.
- How to use the pack and align buttons to align elements in the layout.
- How to position views within a `LinearLayout`.
- How to position views within a `RelativeLayout`.

What you'll do

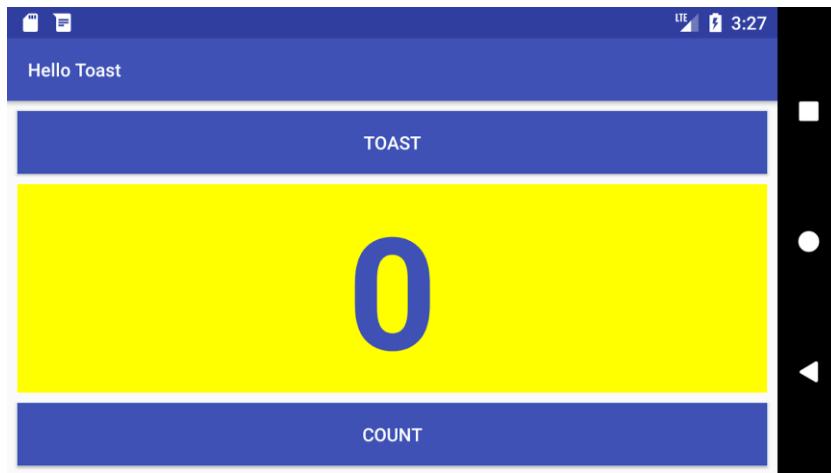
- Create a layout variant for a horizontal display orientation.
- Create a layout variant for tablets and larger displays.
- Modify the layout to add constraints to the UI elements.
- Use `ConstraintLayout` baseline constraints to align elements with text.
- Use `ConstraintLayout` pack and align buttons to align elements.
- Change the layout to use `LinearLayout`.
- Position elements in a `LinearLayout`.
- Change the layout to use `RelativeLayout`.
- Rearrange the views in the main layout to be relative to each other.

App overview

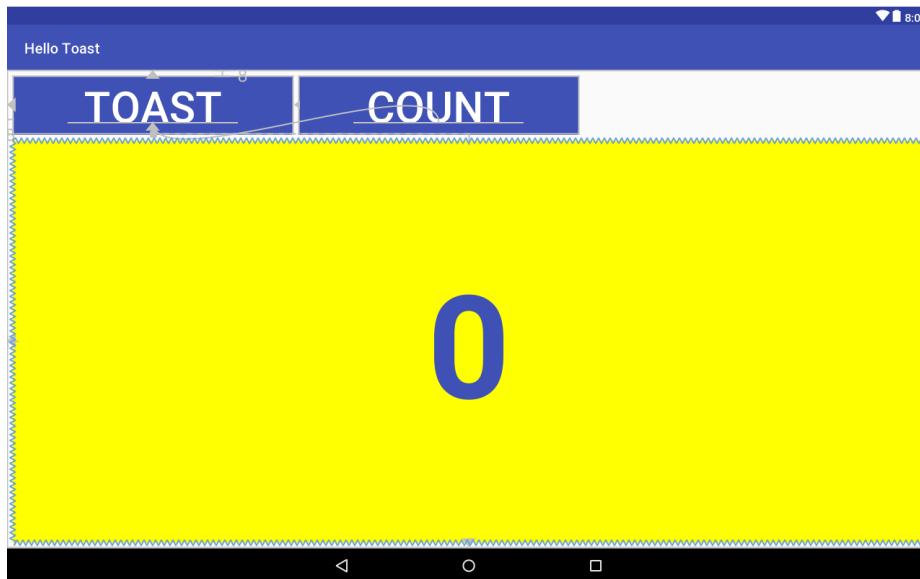
The Hello Toast app in a previous lesson uses [ConstraintLayout](#) to arrange the UI elements in the Activity layout, as shown in the figure below.



To gain more practice with ConstraintLayout, you will create a variant of this layout for horizontal orientation as shown in the figure below.



You will also learn how to use baseline constraints and some of the alignment features of ConstraintLayout by creating another layout variant for tablet displays.



You also learn about other ViewGroup subclasses such as [LinearLayout](#) and [RelativeLayout](#), and change the Hello Toast app layout to use them.

Task 1: Create layout variants

In the previous lesson, the coding challenge required changing the layout of the Hello Toast app so that it would fit properly in a horizontal or vertical orientation. In this task you will learn an easier way to create variants of your layout for horizontal (also known as *landscape*) and vertical (also known as *portrait*) orientations for phones, and for larger displays such as tablets.

In this task you will use some of the buttons in the top two toolbars of the layout editor. The top toolbar lets you configure the appearance of the layout preview in the layout editor:



In the figure above:

1. **Select Design Surface:** Select **Design** to display a color preview of your layout, or **Blueprint** to show only outlines for each UI element. 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. This is useful for previewing the layout without having to run 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 choose **Preview as Right To Left** to view the layout as if an RTL language had been chosen.

The second toolbar lets you configure the appearance of UI elements in a ConstraintLayout, and to zoom and pan the preview:



In the figure above:

1. **Show**: Choose **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/pan controls: Zoom in or out.

Tip: To learn more about using the layout editor, see [Build a UI with Layout Editor](#). To learn more about how to build a layout with ConstraintLayout, see [Build a Responsive UI with ConstraintLayout](#).

1.1 Preview the layout in a horizontal orientation

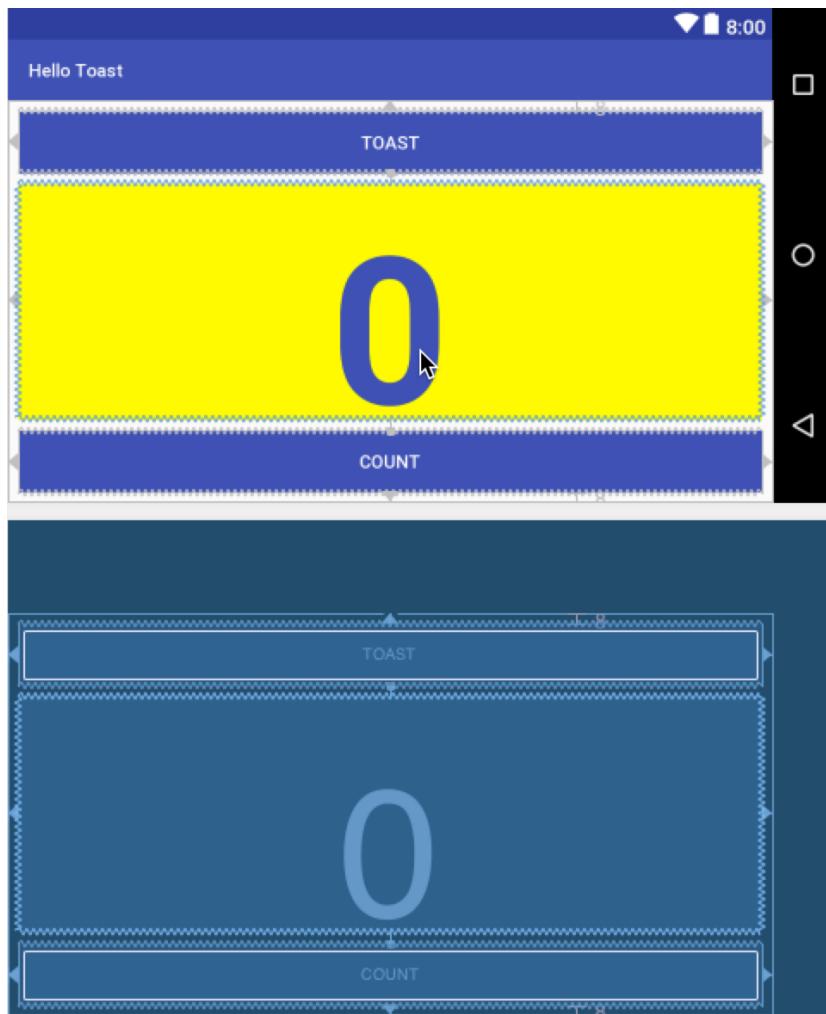
To preview the Hello Toast app layout with a horizontal orientation, follow these steps:

1. Open the Hello Toast app from the previous lesson.

Note: If you downloaded the [solution code for HelloToast](#), you need to delete the finished landscape and extra-large layouts that you will create in this task. Switch from **Project > Android** to **Project > Project Files** in the Project pane, expand **app > app > src/main > res**, select both the **layout-land** folder and the **layout-xlarge** folder, and choose **Edit > Delete**. Then switch the Project pane back to **Project > Android**.

-
2. Open the **activity_main.xml** layout file. Click the **Design** tab if it is not already selected.
 3. Click the **Orientation in Editor** button  in the top toolbar.

-
4. Select **Switch to Landscape** in the dropdown menu. The layout appears in horizontal orientation as shown below. To return to vertical orientation, select **Switch to Portrait**.



1.2 Create a layout variant for horizontal orientation

The visual difference between vertical and horizontal orientations for this layout is that the digit

(0) in the show_countTextViewelement is too low for the horizontal orientation—too close to the

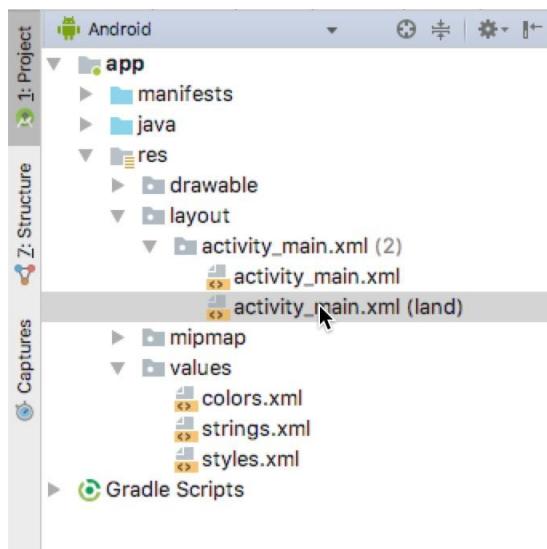
Count button. Depending on which device or emulator you use, the `TextView` element may appear too large or not centered because the text size is fixed to 160sp.

To fix this for horizontal orientations while leaving vertical orientations alone, you can create variant of the Hello Toast app layout that is different for a horizontal orientation. Follow these steps:

1. Click the **Orientation in Editor** button  in the top toolbar.
2. Choose **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.

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



1.3 Preview the layout for different devices

You can preview the layout for different devices without having to run the app on the device or

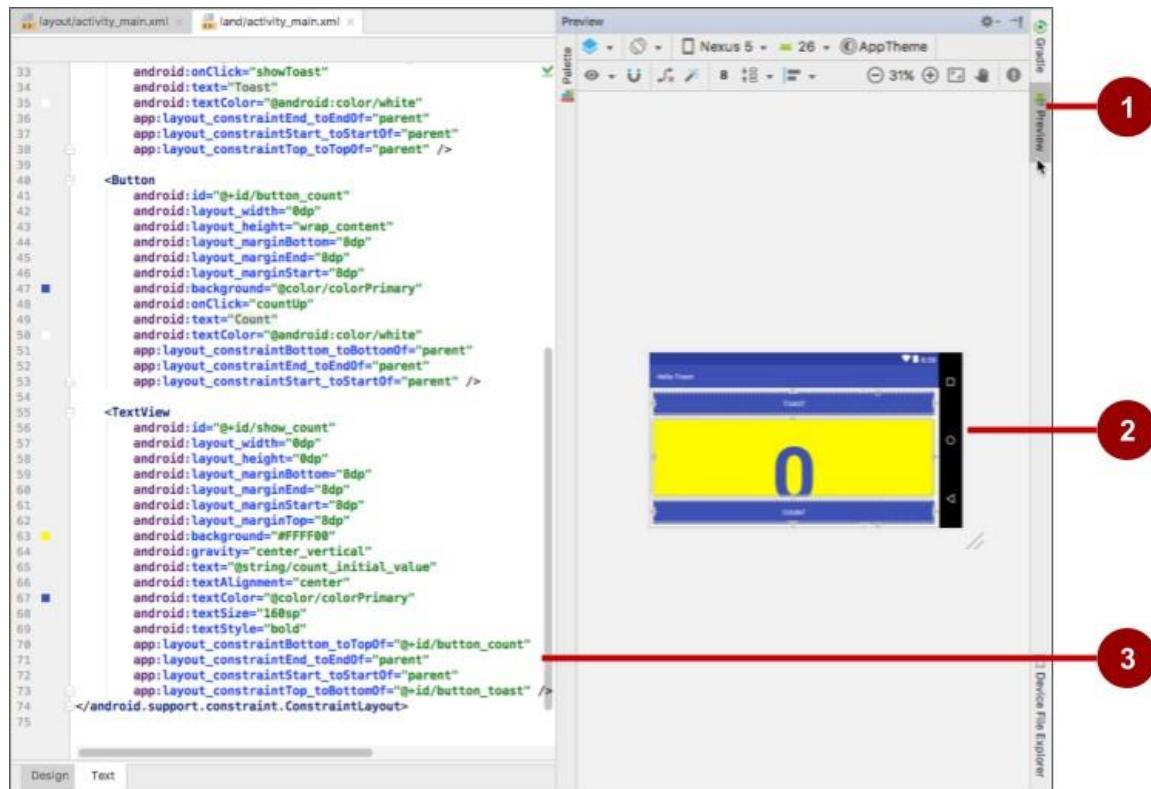
emulator. Follow these steps:

1. The **land/activity_main.xml** tab should still be open in the layout editor; if not, double-click the **activity_main.xml (land)** file in the **layout** directory.

-
2. Click the **Device in Editor** button  **Nexus 5** in the top toolbar.
 3. Choose a different device in the dropdown menu. For example, choose **Nexus 4**, **Nexus 5**, and then **Pixel** to see differences in the previews. These differences are due to the fixed text size for the **TextView**.

1.4 Change the layout for horizontal orientation

You can use the Attributes pane in the **Design** tab to set or change attributes, but it can sometimes be quicker to use the **Text** tab to edit the XML code directly. The **Text** tab shows the XML code and provides a **Preview** tab on the right side of the window to show the layout preview, as shown in the figure below.



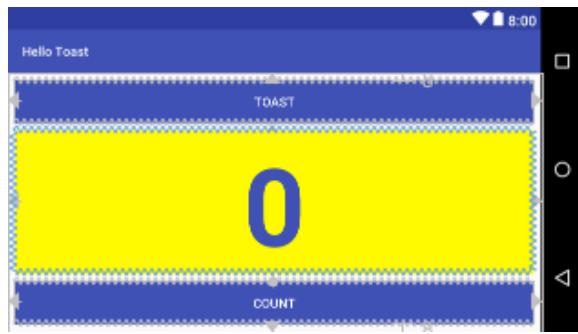
The figure above shows the following:

1. The **Preview** tab, which you use to show the preview pane

-
- 2. The preview pane
 - 3. The XML code

To change the layout, follow these steps:

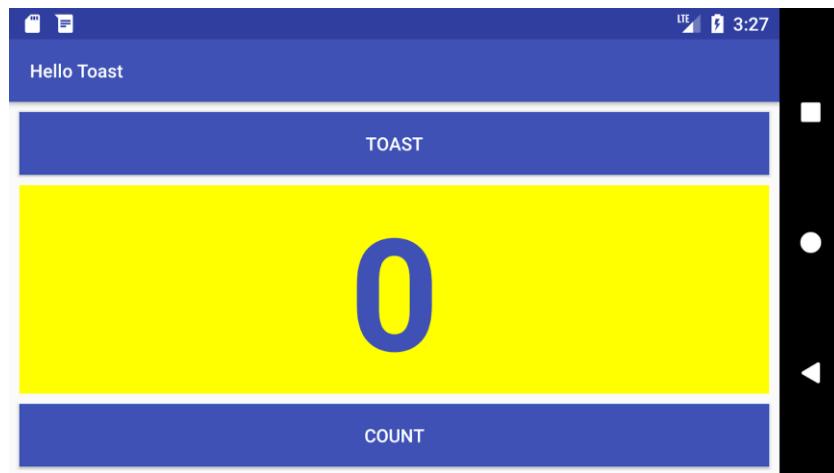
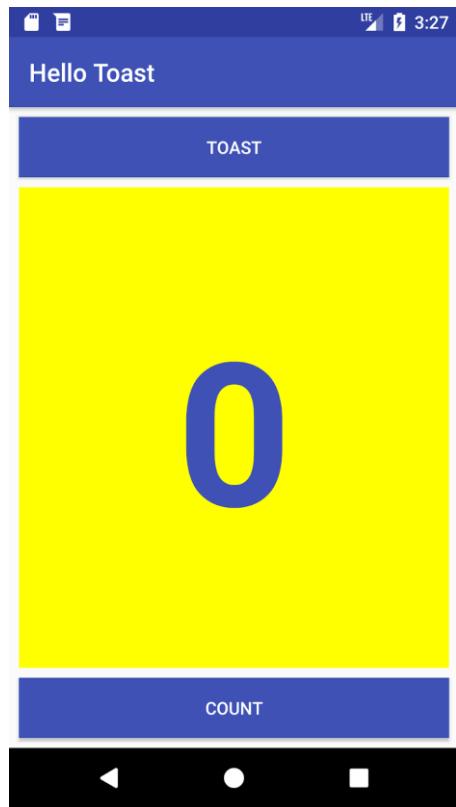
- 1. The **land/activity_main.xml** tab should still be open in the layout editor; if not, double-click the **activity_main.xml (land)** file in the **layout** directory.
- 2. Click the **Text** tab and the **Preview** tab (if not already selected).
- 3. Find the **TextView** element in the XML code.
- 4. Change the `android:textSize="160sp"` attribute to `android:textSize="120sp"`. The layout preview shows the result:



- 5. Choose different devices in the **Device in Editor** dropdown menu to see how the layout looks on different devices in horizontal orientation.

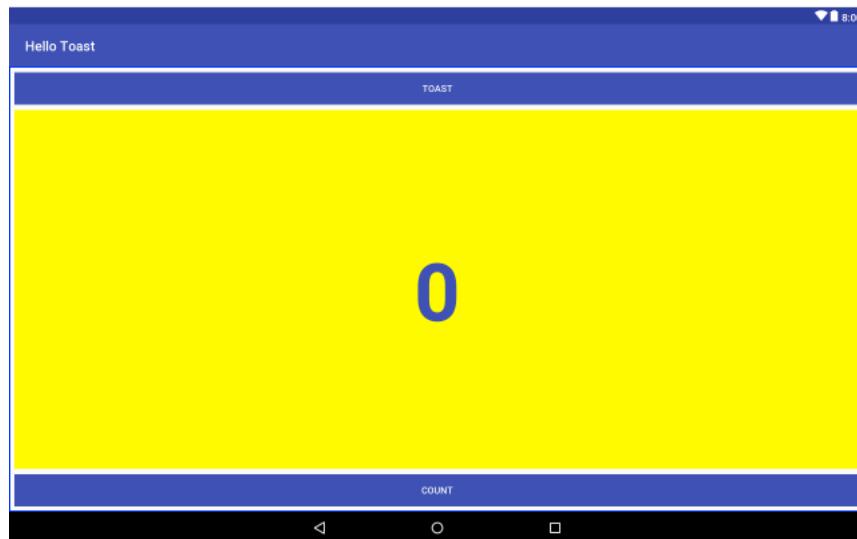
In the editor pane, the **land/activity_main.xml** tab shows the layout for horizontal orientation. The **activity_main.xml** tab shows the unchanged layout for vertical orientation. You can switch back and forth by clicking the tabs.

- 6. Run the app on an emulator or device, and switch the orientation from vertical to horizontal to see both layouts.



1.5 Create a layout variant for tablets

As you learned previously, you can preview the layout for different devices by clicking the **Device in Editor** button  **Nexus 5** in the top toolbar. If you pick a device such as **Nexus 10** (a tablet) from the menu, you can see that the layout is not ideal for a tablet screen—the text of each Button is too small, and the arrangement of the Button elements at the top and bottom is not ideal for a large-screen tablet.



To fix this for tablets while leaving the phone-size horizontal and vertical orientations alone, you can create variant of the layout that is completely different for tablets. Follow these steps:

1. Click the **Design** tab (if not already selected) to show the design and blueprint panes.
2. Click the **Orientation in Editor** button  in the top toolbar.
3. Choose **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. You can change this layout, which is specifically for tablets, without changing the other layouts.

1.6 Change the layout variant for tablets

You can use the Attributes pane in the **Design** tab to change attributes for this layout.

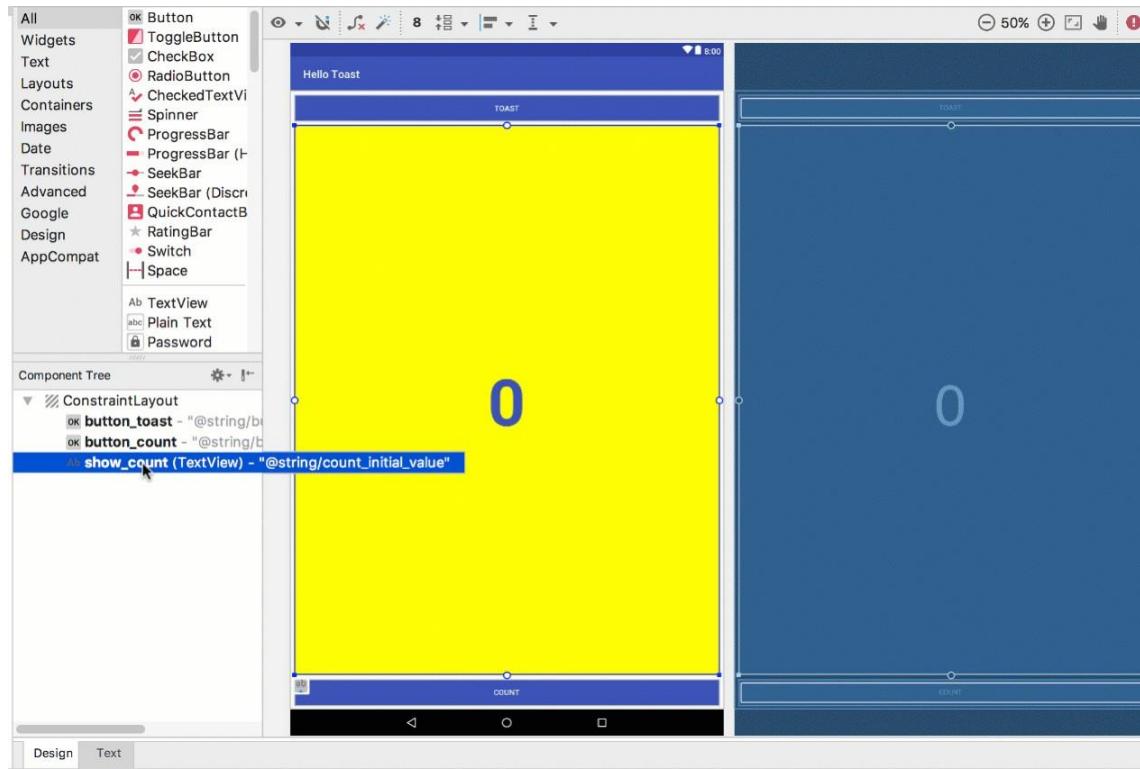
1. Turn off the Autoconnect tool in the toolbar. For this step, ensure that the tool is disabled:



2. Clear all constraints in the layout by clicking the **Clear All Constraints** button in the toolbar.

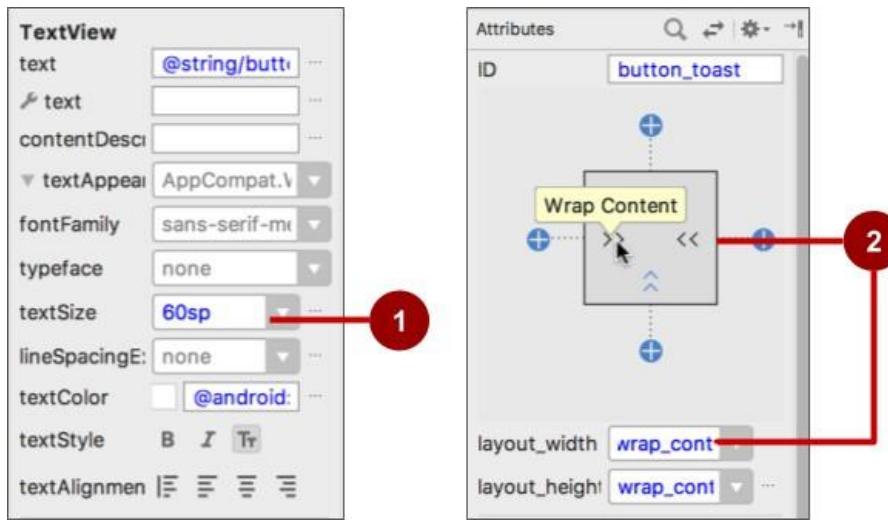
With constraints removed, you can move and resize the elements on the layout freely.

3. The layout editor offers resizing handles on all four corners of an element to resize it. In the **Component Tree**, select the TextView called `show_count`. To get the TextView out of the way so that you can freely drag the Button elements, drag a corner of it to resize it, as shown in the animated figure below.



Resizing an element hardcodes the width and height dimensions. Avoid hardcoding the size dimensions for most elements, because you can't predict how hardcoded dimensions will look on screens of different sizes and densities. You are doing this now just to move the element out of the way, and you will change the dimensions in another step.

4. Select the button_toast Button in the **Component Tree**, click the **Attributes** tab to open the **Attributes** pane, and change the textSize to **60sp** (#1 in the figure below) and the layout_width to **wrap_content** (#2 in the figure below).



As shown on the right side of the figure above (2), you can click the view inspector's width control, which appears in two segments on the left and right sides of the square, until it shows Wrap Content. As an alternative, you can select **wrap_content** from the `layout_width` menu.

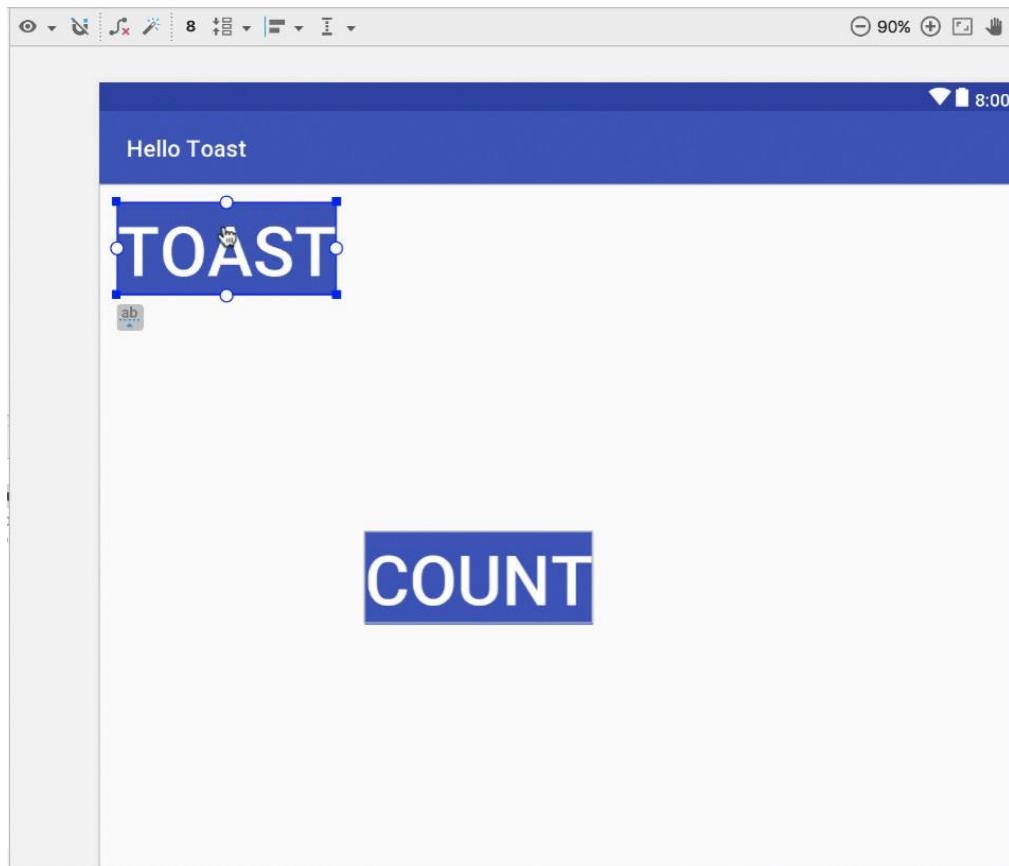
You use `wrap_content` so that if the Button text is localized into a different language, the Button will appear wider or thinner to accommodate the word in the different language.

5. Select the `button_count` Button in the **Component Tree**, change the `textSize` to **60sp** and the `layout_width` to **`wrap_content`**, and drag the Button above the `TextView` to an empty space in the layout.

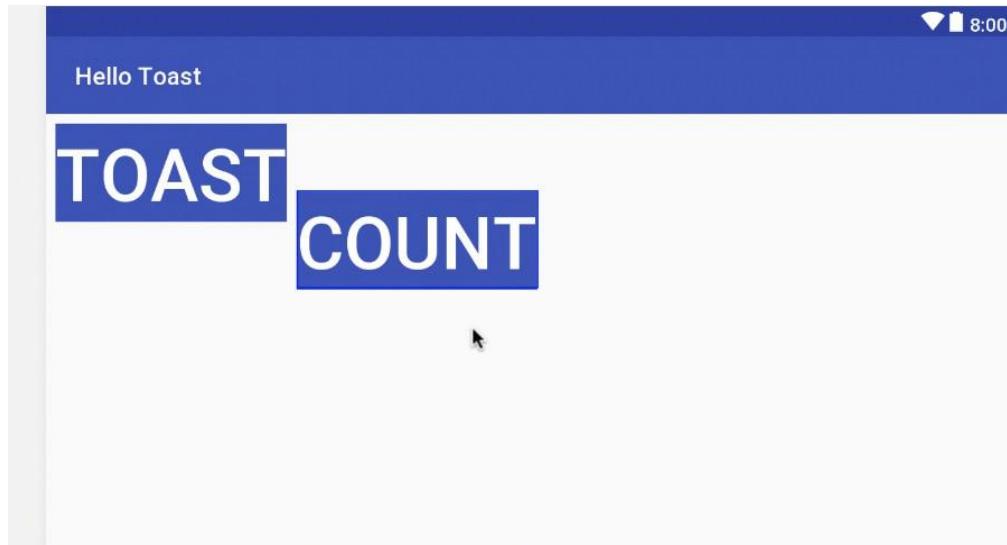
1.7 Use a baseline constraint

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.

1. Constrain the `button_toast` Button to the top and left side of the layout, drag the `button_count` Button to a space near the `button_toast` Button, and constrain the `button_count` Button to the left side of the `button_toast` Button, as shown in the animated figure:



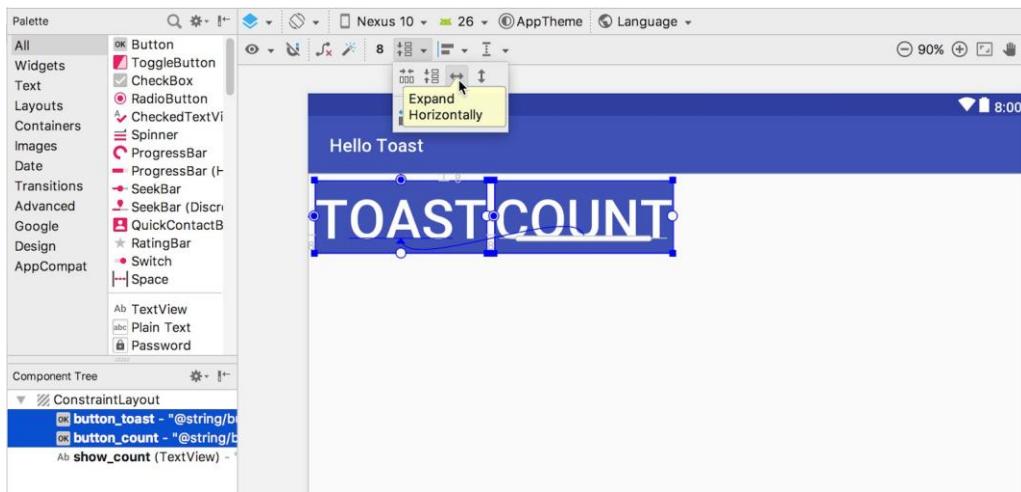
2. Using a *baseline constraint*, you can constrain the `button_count` Button so that its text baseline matches the text baseline of the `button_toast` Button. Select the `button_count` element, and then hover your pointer over the element until the baseline constraint button  appears underneath the element.
3. Click the baseline constraint button. The baseline handle appears, blinking in green as shown in the animated figure. Click and drag a baseline constraint line to the baseline of the `button_toast` element.



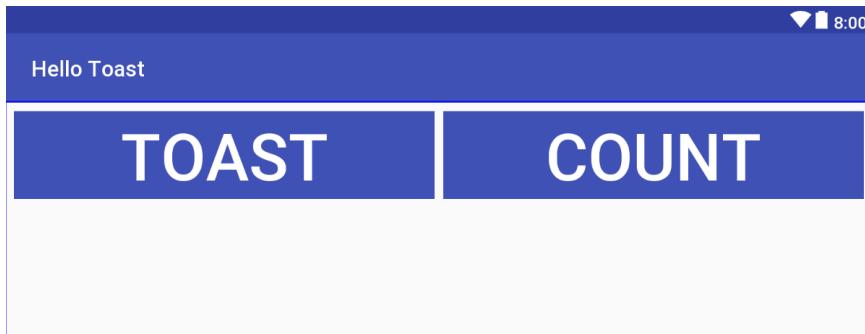
1.8 Expand the buttons horizontally

The pack button in the toolbar provides options for packing or expanding selected UI elements. You can use it to equally arrange the Button elements horizontally across the layout.

1. Select the button_countButton in the **Component Tree**, and Shift-select the button_toast Button so that both are selected.
2. Click the pack button in the toolbar, and choose **Expand Horizontally** as shown in the figure below.



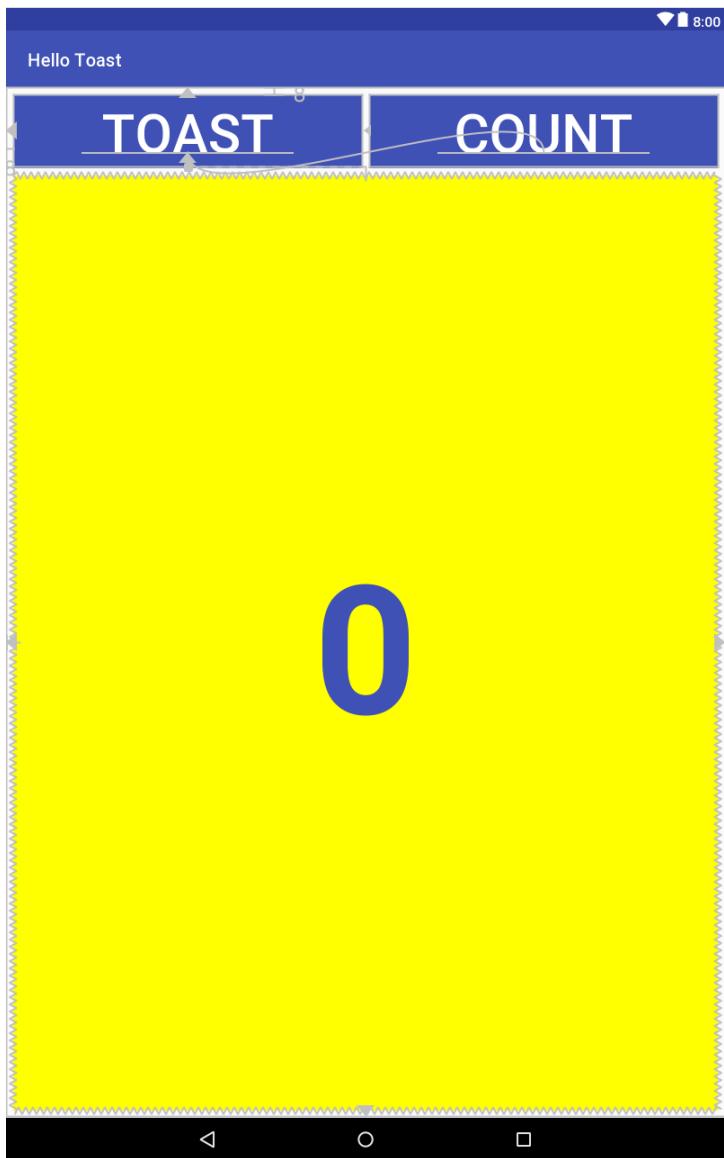
The Button elements expand horizontally to fill the layout as shown below.



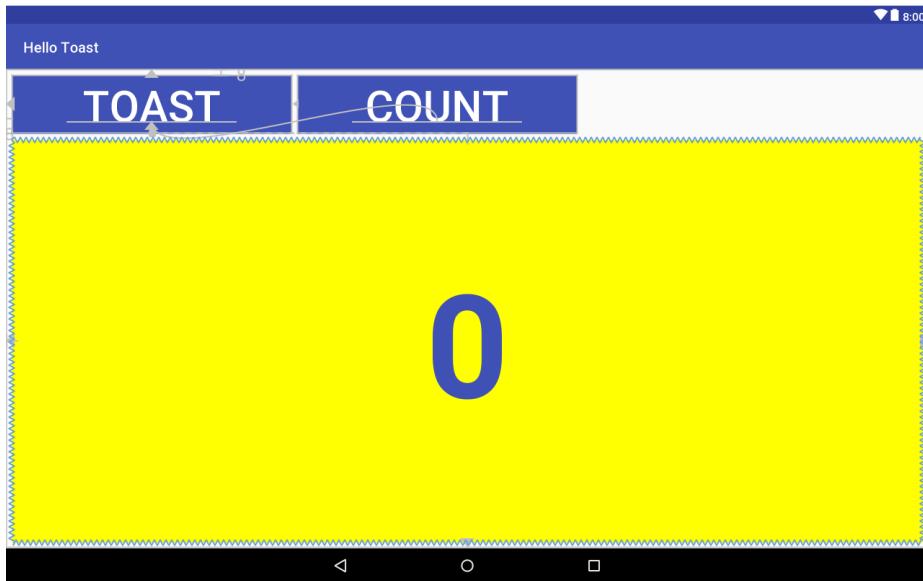
3. To finish the layout, constraint the show_count TextView to the bottom of the button_toast Button and to the sides and bottom of the layout, as shown in the animated figure below.



4. The final steps are to change the `show_countTextView.layout_width` and `layout_height` to **Match Constraints** and the `textSize` to **200sp**. The final layout looks like the figure below.



5. Click the **Orientation in Editor** button in the top toolbar and choose **Switch to Landscape**. The tablet layout appears in horizontal orientation as shown below. (You can choose **Switch to Portrait** to return to vertical orientation.).



6. Run the app on different emulators, and change the orientation after running the app, to see how it looks on different types of devices. You have successfully created an app that can run with a proper UI on phones and tablets that have different screen sizes and densities.

Tip: For an in-depth tutorial on using ConstraintLayout, see [Using ConstraintLayout to design your views](#).

Task 1 solution code

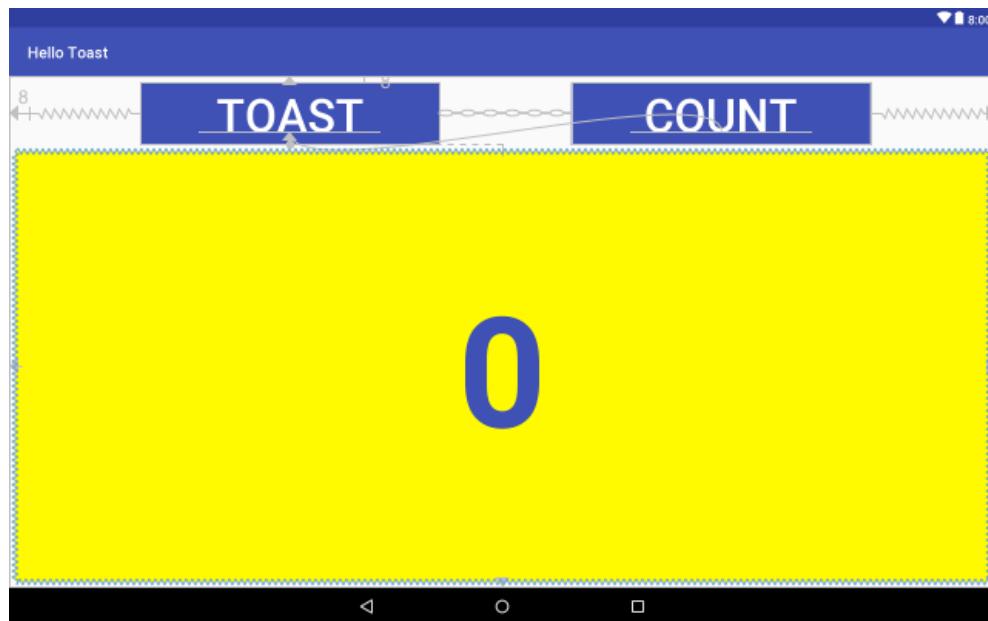
Android Studio project: [HelloToast](#)

Coding challenge 1

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: To accommodate horizontal (landscape) orientation for a tablet, you can center the Button elements in activity_main.xml (xlarge) so that they appear as shown in the figure below.

Hint: Select the elements, click the align button in the toolbar, and choose **Center Horizontally**.



Challenge 1 solution code

Android Studio project: [HelloToastChallenge2](#)

Task 2: Change the layout to LinearLayout

[LinearLayout](#) is a ViewGroup that arranges its collection of views in a horizontal or vertical row. A LinearLayout is one of the most common layouts because it is simple and fast. It is often used within another view group to arrange UI elements horizontally or vertically.

A LinearLayout is required to have these attributes:

- layout_width
- layout_height
- orientation

The layout_width and layout_height can take one of these values:

- match_parent: Expands the view to fill its parent by width or height. When the LinearLayout is the root view, it expands to the size of the screen (the parent view).
- wrap_content: Shrinks the view dimensions so the view is just big enough to enclose its content. If there is no content, the view becomes invisible.
- Fixed number of dp ([density-independent pixels](#)): Specify a fixed size, adjusted for the screen density of the device. For example, 16dp means 16 density-independent pixels.

The orientation can be:

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

In this task you will change the ConstraintLayout root view group for the Hello Toast app to LinearLayout so that you can gain practice in using LinearLayout.

2.1 Change the root view group to LinearLayout

1. Open the Hello Toast app from the previous task.
2. Open the activity_main.xml layout file (if it is not already open), and click the **Text** tab at the bottom of the editing pane to see the XML code. At the very top of the XML code is the following tag line:

```
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android">
```

3. Change the `<android.support.constraint.ConstraintLayout` tag to **<LinearLayout** so that the code looks like this:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">
```

4. Make sure the closing tag at the end of the code has changed to `</LinearLayout>`(Android Studio automatically changes the closing tag if you change the opening tag). If it hasn't changed automatically, change it manually.
5. Under the `<LinearLayout` tag line, add the following attribute after the `android:layout_height` attribute:

```
    android:orientation="vertical"
```

After making these changes, some of the XML attributes for other elements are underlined in red because they are used with `ConstraintLayout` and are not relevant for `LinearLayout`.

2.2 Change element attributes for the `LinearLayout`

Follow these steps to change UI element attributes so that they work with `LinearLayout`:

1. Open the Hello Toast app from the previous task.
2. Open the `activity_main.xml` layout file (if it is not already open), and click the **Text** tab.
3. Find the `button_toast` Button element, and change the following attribute:

Original	Change to
----------	-----------

android:layout_width="0dp"	android:layout_width="match_parent"
----------------------------	-------------------------------------

4. Delete the following attributes from the button_toast element:

app:layout_constraintEnd_toEndOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintTop_toTopOf="parent"

5. Find the button_count Button element, and change the following attribute:

Original	Change to
android:layout_width="0dp"	android:layout_width="match_parent"

6. Delete the following attributes from the button_count element:

app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintEnd_toEndOf="parent" app:layout_constraintStart_toStartOf="parent"

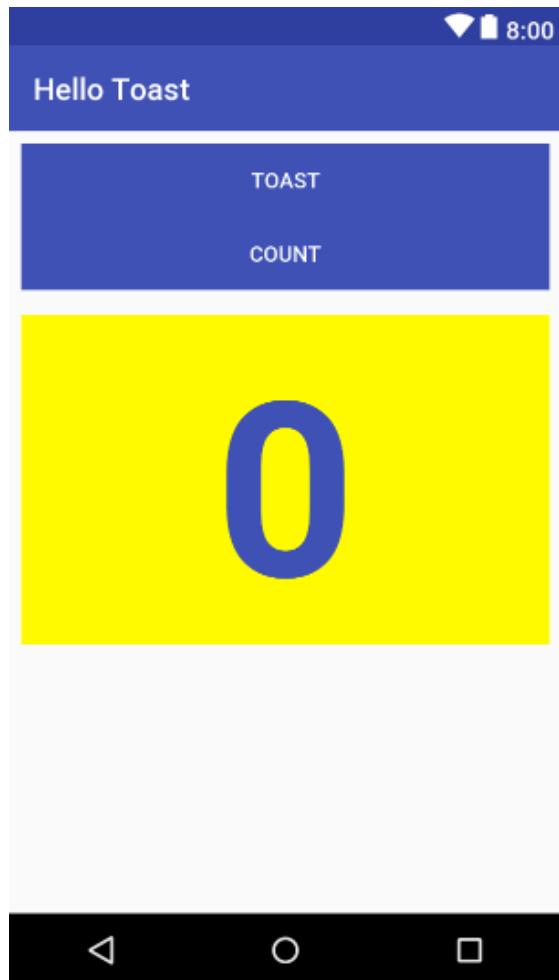
7. Find the show_count TextView element, and change the following attributes:

Original	Change to
android:layout_width="0dp"	android:layout_width="match_parent"
android:layout_width="0dp"	android:layout_height="wrap_content"

-
8. Delete the following attributes from the show_count element:

```
app:layout_constraintBottom_toTopOf="@+id/button_count"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button_toast"
```

9. Click the **Preview** tab on the right side of the Android Studio window (if it is not already selected) to see a preview of the layout thus far:



2.3 Change the positions of elements in the LinearLayout

[LinearLayout](#) arranges its elements in a horizontal or vertical row. You have already added the `android:orientation="vertical"` attribute for the `LinearLayout`, so the elements are stacked on top of each other vertically as shown in the previous figure.

To change their positions so that the **Count** button is on the bottom, follow these steps:

1. Open the Hello Toast app from the previous task.
2. Open the `activity_main.xml` layout file (if it is not already open), and click the **Text** tab.

-
3. Select the button_count Button and all of its attributes, from the <Button tag up to and including the closing />tag, and choose **Edit > Cut**.
 4. Click after the closing />tag of the TextView element but before the closing </LinearLayout>tag, and choose **Edit > Paste**.
 5. (Optional) To fix any indents or spacing issues for cosmetic purposes, choose **Code > Reformat Code** to reformat the XML code with proper spacing and indents.

The XML code for the UI elements now looks like the following:

```
<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:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:background="#FFFF00"
    android:gravity="center_vertical"
    android:text="@string/count_initial_value"
    android:textAlignment="center"
    android:textColor="@color/colorPrimary"
    android:textSize="160sp"
    android:textStyle="bold" />

<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" />
```

By moving the button_countButtonbelow the TextView, the layout is now close to what you had before, with the **Count** button on the bottom. The preview of the layout now looks like the following:



2.4 Add weight to the TextView element

Specifying gravity and weight attributes gives you additional control over arranging views and content in a LinearLayout.

The android:gravity attribute specifies the alignment of the content of a View within the View itself. In the previous lesson you set this attribute for the show_count TextView in order to center the content (the digit 0) in the middle of the TextView:

```
android:gravity="center_vertical"
```

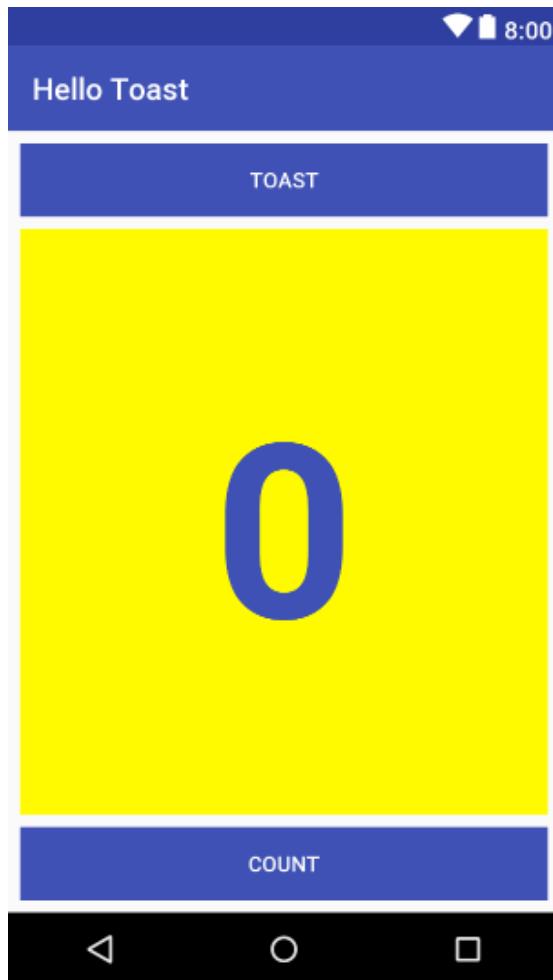
The android:layout_weight attribute indicates how much of the extra space in the LinearLayout will be allocated to the View. If only one View has this attribute, it gets all the extra screen space. For multiple View elements, the space is prorated. For example, if the Button elements each have a weight of 1 and the TextView2, totaling 4, the Button elements get $\frac{1}{4}$ of the space each, and the TextView half.

On different devices, the layout may show the show_count TextView element as filling part or most of the space between the **Toast** and **Count** buttons. In order to expand the TextView to fill the available space no matter which device is used, specify the android:gravity attribute for the TextView. Follow these steps:

1. Open the Hello Toast app from the previous task.
2. Open the activity_main.xml layout file (if it is not already open), and click the **Text** tab.
3. Find the show_count TextView element, and add the following attribute:

```
android:layout_weight="1"
```

The preview now looks like the following figure.



The `show_countTextView` element takes up all the space between the buttons. You can preview the layout for different devices, as you did in a previous task by clicking the **Device in Editor** button  **Nexus 5** in the top toolbar of the preview pane, and choosing a different device. No matter which device you choose for the preview, the `show_countTextView` element should take up all the space between the buttons.

Task 2 solution code

The XML code in activity_main.xml:

```
<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>
```

Task 3: Change the layout to RelativeLayout

A [RelativeLayout](#) is a view grouping in which each view is positioned and aligned relative to other views within the group. In this task you will learn how to build a layout with RelativeLayout.

3.1 Change LinearLayout to RelativeLayout

An easy way to change the LinearLayout to a RelativeLayout is to add XML attributes in the **Text** tab.

1. Open the **activity_main.xml** layout file, and click the **Text** tab at the bottom of the editing pane to see the XML code.
2. Change the **<LinearLayout** at the top to **<RelativeLayout** so that the statement looks like this:

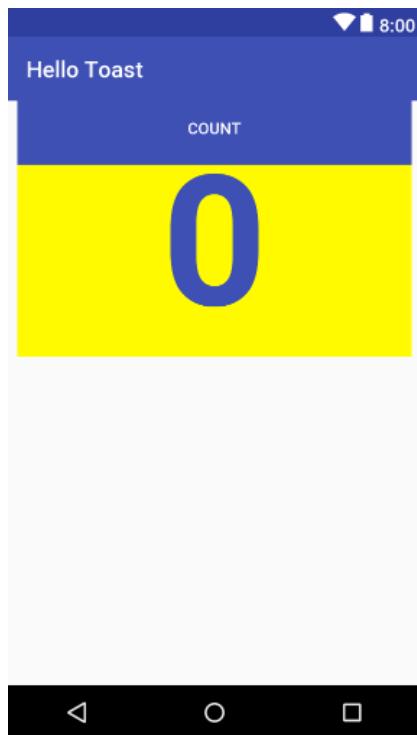
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

3. Scroll down to make sure that the ending tag **</LinearLayout>** has also changed to **</RelativeLayout>**; if it hasn't, change it manually.

3.2 Rearrange views in a RelativeLayout

An easy way to rearrange and position views in a RelativeLayout is to add XML attributes in the **Text** tab.

1. Click the **Preview** tab at the side of the editor (if it is not already selected) to see the layout preview, which now looks like the figure below.



With the change to RelativeLayout, the layout editor also changed some of the view attributes. For example:

- The **Count** Button(button_count) overlays the **Toast** Button, which is why you can't see the **Toast** Button(button_toast).
 - The top part of the TextView(show_count) overlays the Button elements.
2. Add the android:layout_below attribute to the button_count Button to position the Button directly below the show_count TextView. This attribute is one of several attributes for

positioning views within a `RelativeLayout`—you place views in relation to other views.

```
    android:layout_below="@+id/show_count"
```

3. Add the android:layout_centerHorizontal attribute to the same Button to center the view horizontally within its parent, which in this case is the RelativeLayoutview group.

```
    android:layout_centerHorizontal="true"
```

The full XML code for the button_countButtonis as follows:

```
<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"  
    android:layout_below="@+id/show_count"  
    android:layout_centerHorizontal="true"/>
```

4. Add the following attributes to the show_count TextView:

```
    android:layout_below="@+id/button_toast"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"
```

The android:layout_alignParentLeftaligns the view to the left side of the RelativeLayoutparent view group. While this attribute by itself is enough to align the view

to the left side, you may want the view to align to the right side *if* the app is running on a device that is using a right-to-left language. Thus, the android:layout_alignParentStart attribute makes the “start” edge of this view match the start edge of the parent. The *start* is the left edge of the screen if the preference is left-to-right, or it is the right edge of the screen if the preference is right-to-left.

5. Delete the android:layout_weight="1" attribute from the show_count TextView, which is not relevant for a RelativeLayout. The layout preview now looks like the following figure:



Tip: RelativeLayout makes it relatively easy to quickly place UI elements in a layout. To learn more about how to position views in a RelativeLayout, see "[Positioning Views](#)" in the “RelativeLayout” topic of the API Guide.

Task 3 solution code

The XML code in activity_main.xml:

```
<RelativeLayout 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_below="@+id/button_toast"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
```

```
<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"  
    android:layout_below="@+id/show_count"  
    android:layout_centerHorizontal="true"/>  
</RelativeLayout>
```

Summary

Using the layout editor to preview and create variants:

- To preview the app layout with a horizontal orientation in the layout editor, click the **Orientation in Editor** button  in the top toolbar and choose **Switch to Landscape**. Choose **Switch to Portrait** to return to vertical orientation.
- To create variant of the layout that is different for a horizontal orientation, click the **Orientation in Editor** button and choose **Create Landscape Variation**. A new editor window opens with the **land/activity_main.xml** tab showing the layout for the landscape (horizontal) orientation.
- To preview the layout for different devices without having to run the app on the device or emulator, click the **Device in Editor** button  **Nexus 5** in the top toolbar, and choose a device.
- To create variant of the layout that is different for a tablet (larger screen), click the **Orientation in Editor** button and choose **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.

Using ConstraintLayout:

- To clear all constraints in a layout with the ConstraintLayout root, click the **Clear All Constraints**  button in the toolbar.
- 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.
- To create a baseline constraint, hover your pointer over the UI element until the baseline constraint button  appears underneath the element.
- The pack button  in the toolbar provides options for packing or expanding selected UI elements. You can use it to equally arrange the Button elements horizontally across the layout.

Using LinearLayout:

- [LinearLayout](#) is a [ViewGroup](#) that arranges its collection of views in a horizontal or vertical row.
- A LinearLayout is required to have the layout_width, layout_height, and orientation attributes.
- match_parent for layout_width or layout_height: Expands the View to fill its parent by width or height. When the LinearLayout is the root View, it expands to the size of the screen (the parent View).
- Wrap_content for layout_width or layout_height: Shrinks the dimensions so the View is just big enough to enclose its content. If there is no content, the View becomes invisible.
- Fixed number of dp ([density-independent pixels](#)) for layout_width or layout_height: Specify a fixed size, adjusted for the screen density of the device. For example, 16dp means 16 density-independent pixels.
- The orientation for a LinearLayout can be horizontal to arrange elements from left to right, or vertical to arrange elements from top to bottom.

-
- Specifying gravity and weight attributes gives you additional control over arranging views and content in a LinearLayout.
 - The android:gravity attribute specifies the alignment of the content of a View within the View itself.
 - The android:layout_weight attribute indicates how much of the extra space in the LinearLayout will be allocated to the View. If only one View has this attribute, it gets all the extra screen space. For multiple View elements, the space is prorated. For example, if two Button elements each have a weight of 1 and a TextView, totaling 4, the Button elements get $\frac{1}{4}$ of the space each, and the TextView half.

Using RelativeLayout:

- A [RelativeLayout](#) is a ViewGroup in which each view is positioned and aligned relative to other views within the group.
- Use android:layout_alignParentTop to align the View to the top of the parent.
- Use android:layout_alignParentLeft to align the View to the left side of the parent.
- Use android:layout_alignParentStart to make the start edge of the View match the start edge of the parent. This attribute is useful if you want your app to work on devices that use different language or locale preferences. The *start* is the left edge of the screen if the preference is left-to-right, or it is the right edge of the screen if the preference is right-to-left.

Related concepts

The related concept documentation is in [1.2: Layouts and resources for the UI](#).

Learn more

-
- [Meet Android Studio](#)
 - [Create app icons with Image Asset Studio](#)

Android developer documentation:

- [Layouts](#)
- [Build a UI with Layout Editor](#)
- [Build a Responsive UI with ConstraintLayout](#)
- [Layouts](#)
- [LinearLayout](#)
- [RelativeLayout](#)
- [View](#)
- [Button](#)
- [TextView](#)
- [Supporting different pixel densities](#)

[densities](#) Other:

- Codelabs: [Using ConstraintLayout to design your Android views](#)
- [Vocabulary words and concepts glossary](#)

Homework

Change an app

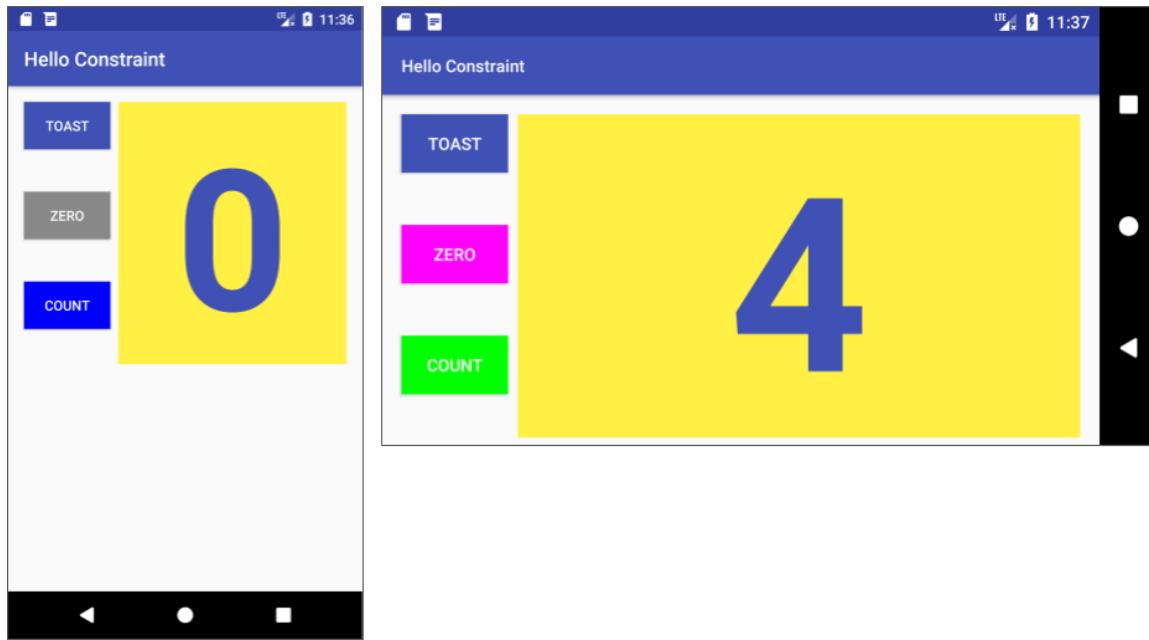
Open the [HelloToast](#) app.

1. Change the name of the project to **HelloConstraint**, and refactor the project to Hello Constraint. (For instructions on how to copy and refactor a project, see [Appendix: Utilities](#).)
2. Modify the activity_main.xml layout to align the **Toast** and **Count** Button elements along the left side of the show_count TextView that shows "0". Refer to the figures below for the layout.
3. Include a third Button called **Zero** that appears between the **Toast** and **Count** Button elements.
4. Distribute the Button elements vertically between the top and bottom of the show_count TextView.
5. Set the **Zero** Button to initially have a gray background.
6. Make sure that you include the **Zero** Button for the landscape orientation in activity_main.xml (land), and also for a tablet-sized screen in activity_main (xlarge).
7. Make the **Zero** Button change the value in the show_count TextView to 0.
8. Update the click handler for the **Count** Button so that it changes its *own* background color, depending on whether the new count is odd or even.

Hint: Don't use findViewById to find the **Count** Button. Is there something else you can use?

Feel free to use constants in the [Color](#) class for the two different background colors.

-
9. Update the click handler for the **Count** Button to set the background color for the **Zero** Button to something other than gray to show it is now active. Hint: You can use `findViewById` in this case.
 10. Update the click handler for the **Zero** Button to reset the color to gray, so that it is gray when the count is zero.



Answer these questions

Question 1

Which two layout constraint attributes on the **Zero** Button position it vertically equal distance between the other two Button elements? (Pick 2 answers.)

- `app:layout_constraintBottom_toTopOf="@+id/button_count"`
- `android:layout_marginBottom="8dp"`
- `android:layout_marginStart="16dp"`
- `app:layout_constraintTop_toBottomOf="@+id/button_toast"`

-
- android:layout_marginTop="8dp"

Question 2

Which layout constraint attribute on the **Zero** Button positions it horizontally in alignment with the other two Button elements?

- app:layout_constraintLeft_toLeftOf="parent"
- app:layout_constraintBottom_toTopOf="@+id/button_count"
- android:layout_marginBottom="8dp"
- app:layout_constraintTop_toBottomOf="@+id/button_toast"

Question 3

What is the correct signature for a method used with the android:onClick XML attribute?

- public void callMethod()
- public void callMethod(View view)
- private void callMethod(View view)
- public boolean callMethod(View view)

Question 4

The click handler for the **Count** Button starts with the following method signature:

```
public void countUp(View view)
```

Which of the following techniques is more efficient to use within this handler to change the Button element's background color? Choose one:

- Use findViewById to find the **Count** Button. Assign the result to a View variable, and then use [setBackgroundColor\(\)](#).
- Use the view parameter that is passed to the click handler with [setBackgroundColor\(\): view.setBackgroundColor\(\)](#)

Submit your app for grading

Guidance for graders

Check that the app has the following features:

- It displays the **Zero** button.
- The **Zero** button is between the **Toast** and **Count** buttons.
- The app includes an implementation of activity_main.xml, activity_main.xml (land), and activity_main.xml (xlarge).
- The app includes an implementation of the click handler method for the **Zero** button to reset the count to 0. The method must show the zero count in the show_count TextView. The click handler must also reset the **Zero** button's own background color to gray.
- The click handler method for the **Count** button has been updated to change its own background color depending on whether the new count is odd or even. This method must use the view parameter to access the button. This method must also change the background of the **Zero** button to a color other than gray.

Lesson 1.3: Text and scrolling views

Introduction

The [TextView](#) class is a subclass of the [View](#) class that displays text on the screen. You can control how the text appears with TextView attributes in the XML layout file. This practical shows how to work with multiple TextView elements, including one in which the user can scroll its contents vertically.

If you have more information than fits on the device's display, you can create a *scrolling view* so 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 enable users to enter multiple lines of text, or to combine UI elements (such as a text field and a button) within a scrolling view.

The [ScrollView](#) class provides the layout for the scrolling view. ScrollView is a subclass of [FrameLayout](#). Place only *one* view as a child within it—a child view contains the entire contents to scroll. This child view may itself be a [ViewGroup](#) (such as [LinearLayout](#)) containing UI elements.

Complex layouts may suffer performance issues with child views such as images. A good choice for a View within a ScrollView is a LinearLayout that is arranged in a vertical orientation, presenting items that the user can scroll through (such as TextView elements).

With a ScrollView, all of the UI elements are in memory and in the view hierarchy even if they aren't displayed on screen. This makes ScrollView ideal for scrolling pages of free-form text smoothly, because the text is already in memory. However, ScrollView can use up a lot of memory, which can affect the performance of the rest of your app. To display long lists of items that users can add to, delete from, or edit, consider using a [RecyclerView](#), which is described in a separate lesson.

What you should already know

You should be able to:

- Create a Hello World app with Android Studio.
- Run an app on an emulator or a device.
- Implement a TextView in a layout for an app.
- Create and use string resources.

What you'll learn

- How to use XML code to add multiple TextView elements.
- How to use XML code to define a scrolling View.
- How to display free-form text with some HTML formatting tags.
- How to style the TextView background color and text color.
- How to include a web link in the text.

What you'll do

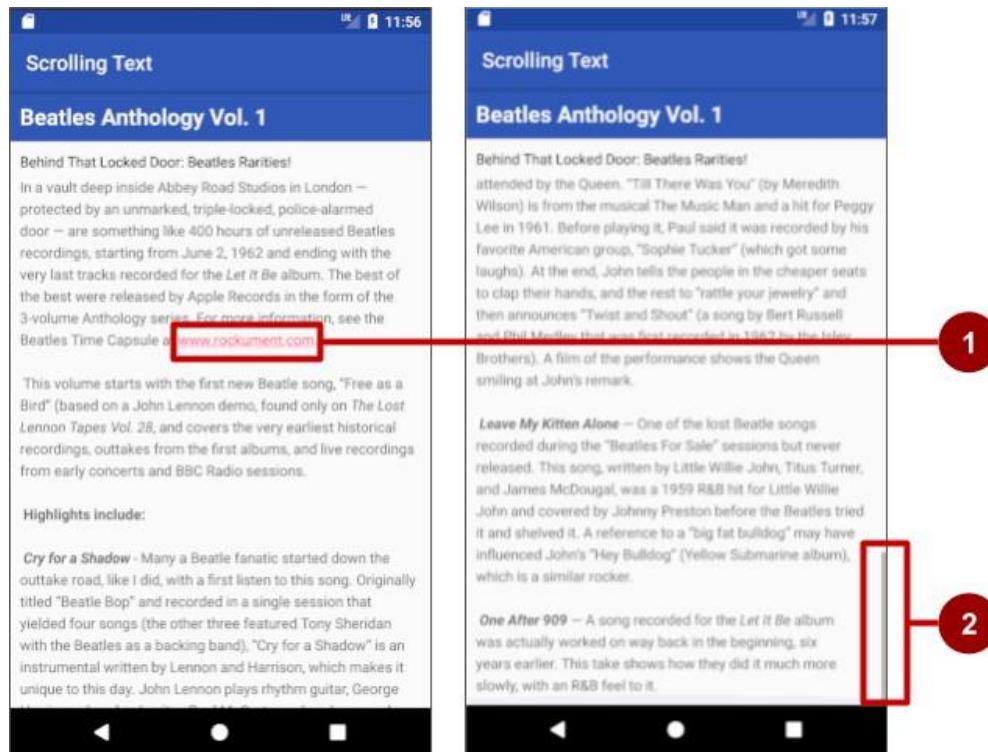
- Create the ScrollingText app.

-
- Change the ConstraintLayout ViewGroup to RelativeLayout.
 - Add two TextView elements for the article heading and subheading.
 - Use TextAppearance styles and colors for the article heading and subheading.

-
- Use HTML tags in the text string to control formatting.
 - Use the lineSpacingExtra attribute to add line spacing for readability.
 - Add a ScrollView to the layout to enable scrolling a TextView element.
 - Add the autoLink attribute to enable URLs in the text to be active and clickable.

App overview

The Scrolling Text app demonstrates the [ScrollView](#) UI component. ScrollView is a ViewGroup that in this example contains a TextView. It shows a lengthy page of text—in this case, a music album review—that the user can scroll vertically to read by swiping up and down. A scroll bar appears in the right margin. The app shows how you can use text formatted with minimal HTML tags for setting text to bold or italic, and with new-line characters to separate paragraphs. You can also include active web links in the text.

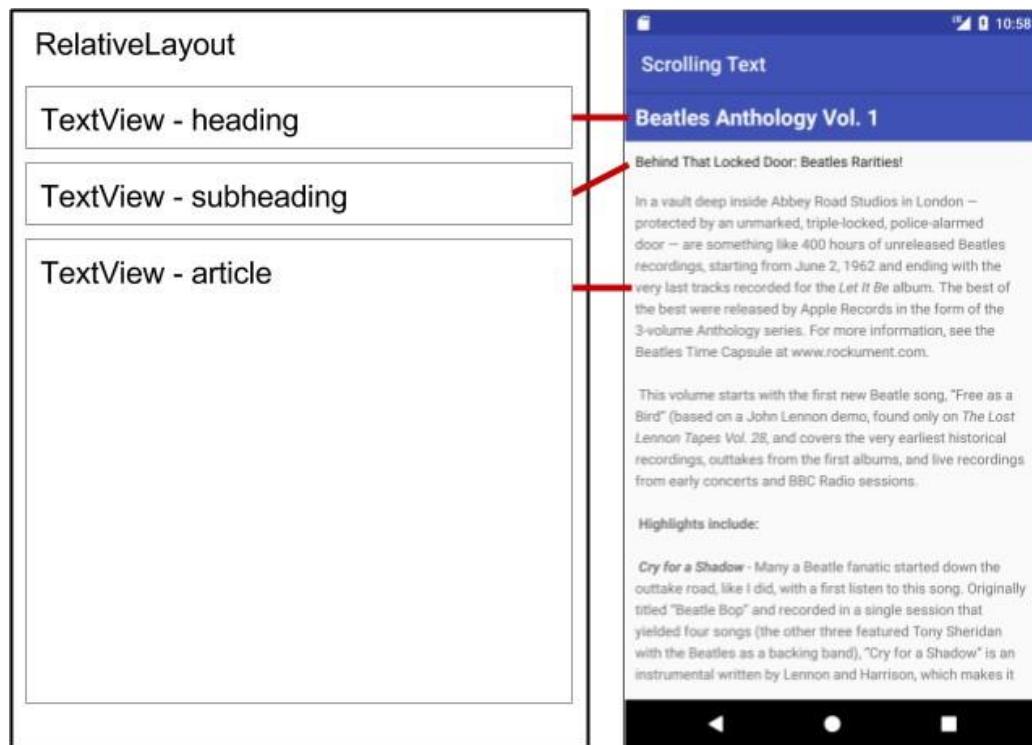


In the above figure, the following appear:

-
1. An active web link embedded in free-form text
 2. The scroll bar that appears when scrolling the text

Task 1: Add and edit TextView elements

In this practical, you will create an Android project for the ScrollingText app, add TextView elements to the layout for an article title and subtitle, and change the existing “Hello World” TextView element to show a lengthy article. The figure below is a diagram of the layout.



You will make all these changes in the XML code and in the strings.xml file. You will edit the XML code for the layout in the Text pane, which you show by clicking the **Text** tab, rather than clicking the **Design** tab for the Design pane. Some changes to UI elements and attributes are easier to make directly in the Text pane using XML source code.

1.1 Create the project and TextView elements

In this task you will create the project and the TextView elements, and use TextView attributes for styling the text and background.

Tip: To learn more about these attributes, see the [TextView](#) reference.

1. In Android Studio create a new project with the following parameters:

Attribute	Value
Application Name	Scrolling Text
Company Name	android.example.com (or your own domain)
Phone and Tablet Minimum SDK	API15: Android 4.0.3 IceCreamSandwich
Template	Empty Activity
Generate Layout File checkbox	Selected
Backwards Compatibility (AppCompat) checkbox	Selected

2. In the **app > res > layout** folder in the **Project > Android** pane, open the **activity_main.xml** file, and click the **Text** tab to see the XML code.

At the top, or *root*, of the Viewhierarchy is the [ConstraintLayout](#) ViewGroup:

```
android.support.constraint.ConstraintLayout
```

-
3. Change this ViewGroup to [RelativeLayout](#). The second line of code now looks something like this:

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

RelativeLayout lets you place UI elements relative to each other, or relative to the parent RelativeLayout itself.

The default “Hello World” TextView element created by the Empty Layout template still has constraint attributes (such as app:layout_constraintBottom_toBottomOf="parent"). Don't worry—you will remove them in a subsequent step.

4. Delete the following line of XML code, which is related to ConstraintLayout:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

The block of XML code at the top now looks like this:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.android.scrollingtext.MainActivity">
```

5. Add a TextView element above the “Hello World” TextView by entering <TextView. A TextView block appears that ends with /> and shows the layout_width and layout_height attributes, which are required for the TextView.
6. Enter the following attributes for the TextView. As you enter each attribute and value, suggestions appear to complete the attribute name or value.

TextView#1 attribute	Value
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:id	"@+id/article_heading"
android:background	"@color/colorPrimary"
android:textColor	"@android:color/white"
android:padding	"10dp"
android:textAppearance	"@android:style/TextAppearance.DeviceDefault.Large"
android:textStyle	"bold"
android:text	"Article Title"

7. Extract the string resource for the android:text attribute's hardcoded string "Article Title" in the TextView to create an entry for it in **strings.xml**.

Place the cursor on the hardcoded string, press Alt-Enter (Option-Enter on the Mac), and select **Extract string resource**. Make sure that the **Create the resource in directories** option is selected, and then edit the resource name for the string value to **article_title**.

String resources are described in detail in the [String Resources](#).

8. Extract the dimension resource for the android:padding attribute's hardcoded string "10dp" in the TextView to create dimens.xml and add an entry to it.

Place the cursor on the hardcoded string, press Alt-Enter (Option-Enter on the Mac), and select **Extract dimension resource**. Make sure that the **Create the resource in directories** option is selected, and then edit the Resource name to **padding_regular**.

9. Add another TextView element above the "Hello World" TextView and below the TextView you created in the previous steps. Add the following attributes to the TextView:

TextView#2 Attribute	Value
-----------------------------	--------------

layout_width	"match_parent"
--------------	----------------

layout_height	"wrap_content"
android:id	"@+id/article_subheading"
android:layout_below	"@+id/article_heading"
android:padding	"@dimen/padding_regular"
android:textAppearance	"@android:style/TextAppearance.DeviceDefault"
android:text	"Article Subtitle"

Because you extracted the dimension resource for the "10dp" string to padding_regular in the previously created TextView, you can use "@dimen/padding_regular" for the android:padding attribute in this TextView.

10. Extract the string resource for the android:text attribute's hardcoded string "Article Subtitle" in the TextView to **article_subtitle**.
11. In the "Hello World" TextView element, delete the layout_constraint attributes:

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

12. Add the following TextView attributes to the "Hello World" TextView element, and change the android:text attribute:

TextViewAttribute	Value
android:id	"@+id/article"
android:layout_below	"@+id/article_subheading"
android:lineSpacingExtra	"5sp"
android:padding	"@dimen/padding_regular"
android:text	Change to "Article text"

-
13. Extract the string resource for "Article text" to **article_text**, and extract the dimension resource for "5sp" to **line_spacing**.
 14. Reformat and align the code by choosing **Code > Reformat Code**. It is a good practice to reformat and align your code so that it is easier for you and others to understand.

1.2 Add the text of the article

In a real app that accesses magazine or newspaper articles, the articles that appear would probably come from an online source through a content provider, or might be saved in advance in a database on the device.

For this practical, you will create the article as a single long string in the strings.xml resource.

1. In the **app > res > values** folder, open **strings.xml**.
2. Open any text file with a large amount of text, or open the [strings.xml file of the finished ScrollingText app](#).
3. Enter the values for the strings `article_title` and `article_subtitle` with either a made-up title and subtitle, or use the values in the strings.xml file of the finished ScrollingText app. Make the string values single-line text without HTML tags or multiple lines.
4. Enter or copy and paste text for the `article_text` string.

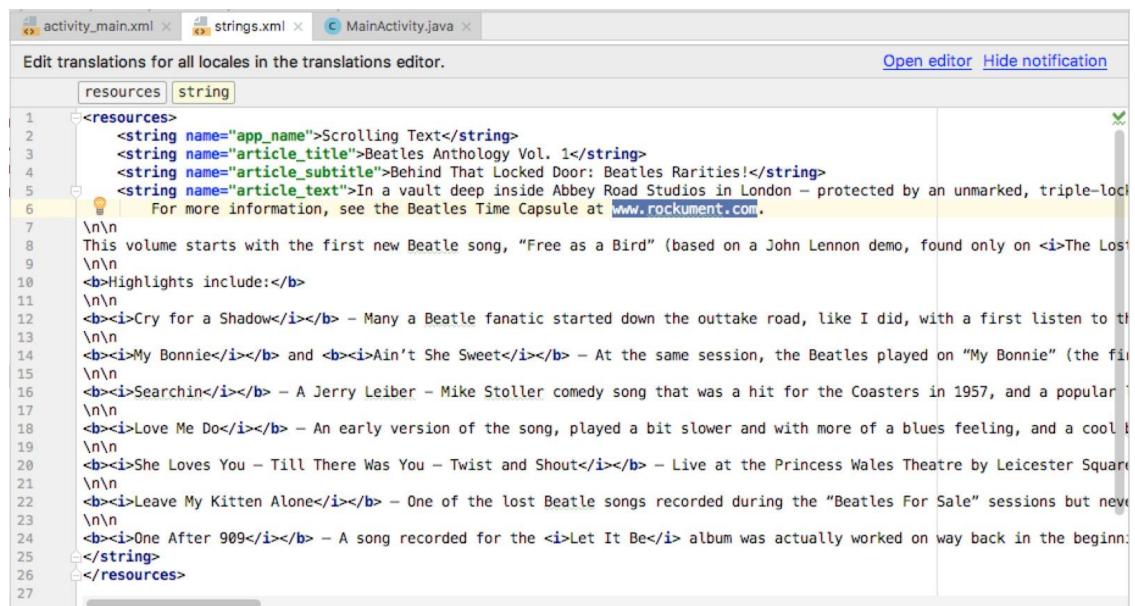
You can use the text in your text file, or use the text provided for the `article_text` string in the strings.xml file of the finished ScrollingText app. The only requirement for this task is that the text must be long enough so that it doesn't fit on the screen.

Keep in mind the following (refer to the figure below for an example):

- 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. If you want the text in strings.xml to be wrapped, you can press Return to enter hard line endings, or format the text first in a text editor with hard line endings.

-
- 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 **** tags around words that should be in bold.
- Enter the HTML **<i>** and **</i>** tags around words that should be in italics. If you use curled apostrophes within an italic phrase, replace them with straight apostrophes.
- You can combine bold and italics by combining the tags, as in **<i>...</i>**. Other HTML tags are ignored.
- Enclose The entire text within **<string name="article_text"> </string>**in the strings.xmlfile.
- Include a web link to test, such as **www.google.com**. (The example below uses www.rockument.com.) *Don't* use an HTML tag, because any HTML tags except the bold and italic tags are ignored and presented as text, which is not what you want.



The screenshot shows the Android Studio interface with three tabs at the top: activity_main.xml, strings.xml (which is selected), and MainActivity.java. Below the tabs, a notification bar says "Edit translations for all locales in the translations editor." and includes "Open editor" and "Hide notification" buttons. The main area is titled "Edit translations for all locales in the translations editor." and shows the XML code for the strings.xml file. A warning icon (a yellow exclamation mark) is positioned next to the string resource at line 6. The code is as follows:

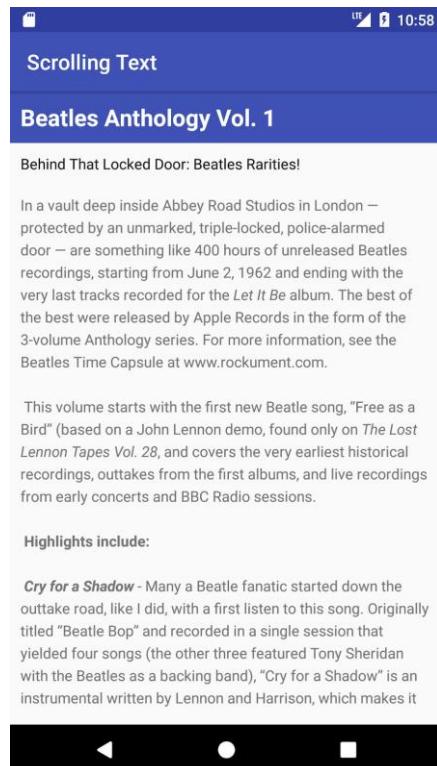
```

<resources>
    <string name="app_name">Scrolling Text</string>
    <string name="article_title">Beatles Anthology Vol. 1</string>
    <string name="article_subtitle">Behind That Locked Door: Beatles Rarities!</string>
    <string name="article_text">In a vault deep inside Abbey Road Studios in London – protected by an unmarked, triple-locked door – lies a time capsule containing a trove of rare Beatles recordings. This volume starts with the first new Beatle song, "Free as a Bird" (based on a John Lennon demo, found only on The Lost <u>www.rockument.com</u>). Don't use an HTML tag, because any HTML tags except the bold and italic tags are ignored and presented as text, which is not what you want.
    <b>Highlights include:</b>
    <b><i>Cry for a Shadow</i></b> – Many a Beatle fanatic started down the outtake road, like I did, with a first listen to this song.
    <b><i>My Bonnie</i></b> and <b><i>Ain't She Sweet</i></b> – At the same session, the Beatles played on "My Bonnie" (the first song ever recorded by the band).
    <b><i>Searchin</i></b> – A Jerry Leiber – Mike Stoller comedy song that was a hit for the Coasters in 1957, and a popular Beatles cover.
    <b><i>Love Me Do</i></b> – An early version of the song, played a bit slower and with more of a blues feeling, and a cool title.
    <b><i>She Loves You – Till There Was You – Twist and Shout</i></b> – Live at the Princess Wales Theatre by Leicester Square.
    <b><i>Leave My Kitten Alone</i></b> – One of the lost Beatle songs recorded during the "Beatles For Sale" sessions but never released.
    <b><i>One After 909</i></b> – A song recorded for the <i>Let It Be</i> album was actually worked on way back in the beginning.
</string>
</resources>

```

1.3 Run the app

Run the app. The article appears, but the user can't scroll the article because you haven't yet included a ScrollView(which you will do in the next task). Note also that tapping a web link does not currently do anything. You will also fix that in the next task.



Task 1 solution code

The activity_main.xml layout file looks like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        tools:context="com.example.android.scrollingtext.MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/article_heading"
        android:background="@color/colorPrimary"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_title"
        android:textAppearance=
            "@android:style/TextAppearance.DeviceDefault.Large"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

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

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

</RelativeLayout>
```

Task 2: Add a ScrollView and an active web link

In the previous task you created the ScrollingText app with TextView elements for an article title, subtitle, and lengthy article text. You also included a web link, but the link is not yet active. You will add the code to make it active.

Also, the TextView by itself can't enable users to scroll the article text to see all of it. You will add a new ViewGroup called ScrollView to the XML layout that will make the TextView scrollable.

2.1 Add the autoLink attribute for active web links

Add the android:autoLink="web" attribute to the articleTextView. The XML code for this TextView now looks like this:

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

2.2 Add a ScrollView to the layout

To make a View(such as a TextView) scrollable, embed the View *inside* a ScrollView.

-
1. Add a ScrollView between the article_subheadingTextView and the articleTextView. As you enter <ScrollView>, Android Studio automatically adds </ScrollView> at the end, and presents the android:layout_width and android:layout_height attributes with suggestions.
 2. Choose **wrap_content** from the suggestions for both attributes.

The code for the two TextView elements and the ScrollView now looks like this:

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

<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ScrollView>

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

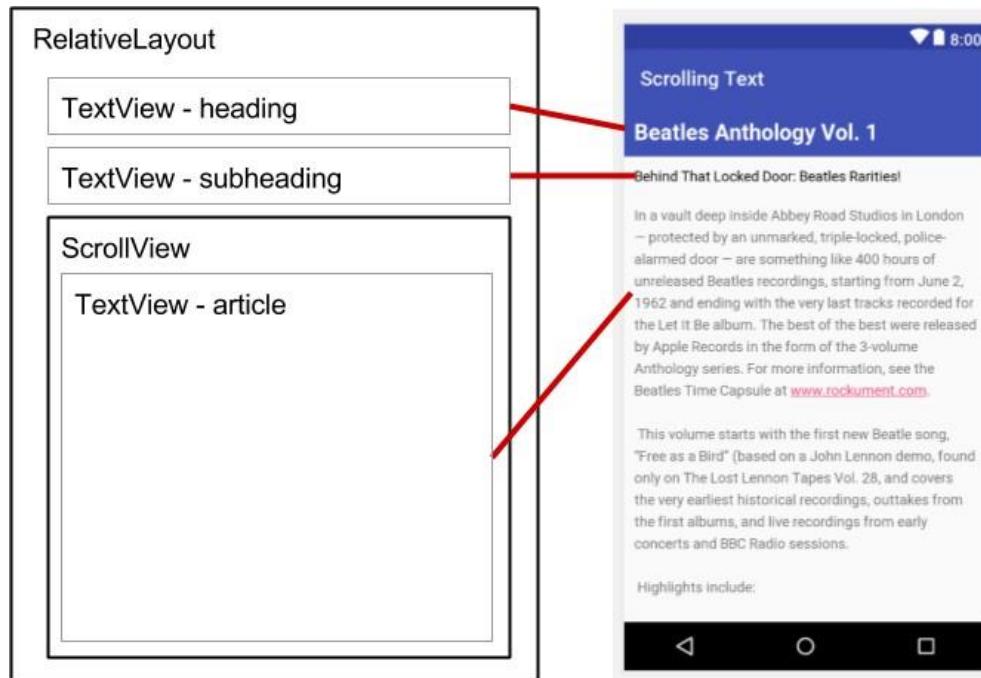
3. Move the ending </ScrollView> code *after* the articleTextView so that the article TextView attributes are entirely inside the ScrollView.
4. Remove the following attribute from the article TextView and add it to the ScrollView:

```
    android:layout_below="@+id/article_subheading"
```

With the above attribute, the ScrollView element will appear below the article subheading. The article is inside the ScrollView element.

5. Choose **Code > Reformat Code** to reformat the XML code so that the article TextView now appears indented inside the <ScrollView> code.
6. Click the **Preview** tab on the right side of the layout editor to see a preview of the layout.

The layout now looks like the right side of the following figure:



2.3 Run the app

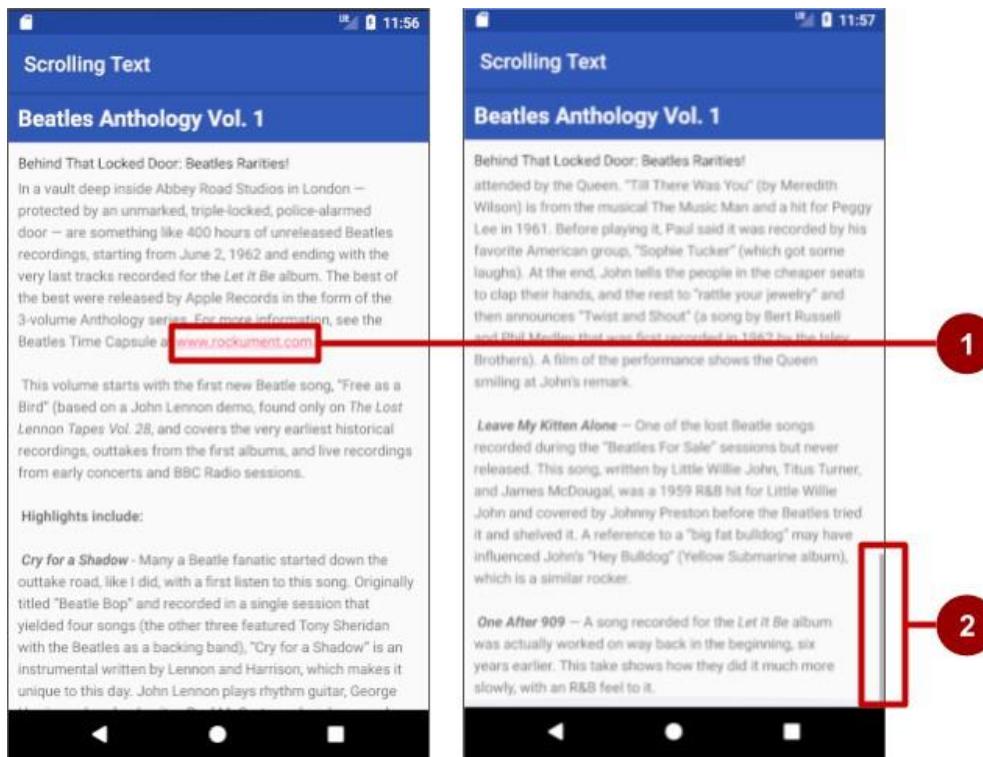
To examine how the text scrolls:

-
1. Run the app on a device or emulator.

Swipe up and down to scroll the article. The scroll bar appears in the right margin as you scroll.

Tap the web link to go to the web page. The android:autoLink attribute turns any recognizable URL in the TextView(such as www.rockument.com) into a web link.

2. Rotate your device or emulator while running the app. Notice how the scrolling view widens to use the full display and still scrolls properly.
3. Run the app on a tablet or tablet emulator. Notice how the scrolling view widens to use the full display and still scrolls properly.



In the above figure, the following appear:

1. An active web link embedded in free-form text
2. The scroll bar that appears when scrolling the text

Task 2 solution code

The XML code for the layout with the scroll view is as follows:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.android.scrollingtext.MainActivity">

    <TextView
        android:id="@+id/article_heading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_title"
        android:textAppearance=
            "@android:style/TextAppearance.DeviceDefault.Large"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

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

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/article_subheading">

        <TextView
            android:id="@+id/article"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:autoLink="web"
```

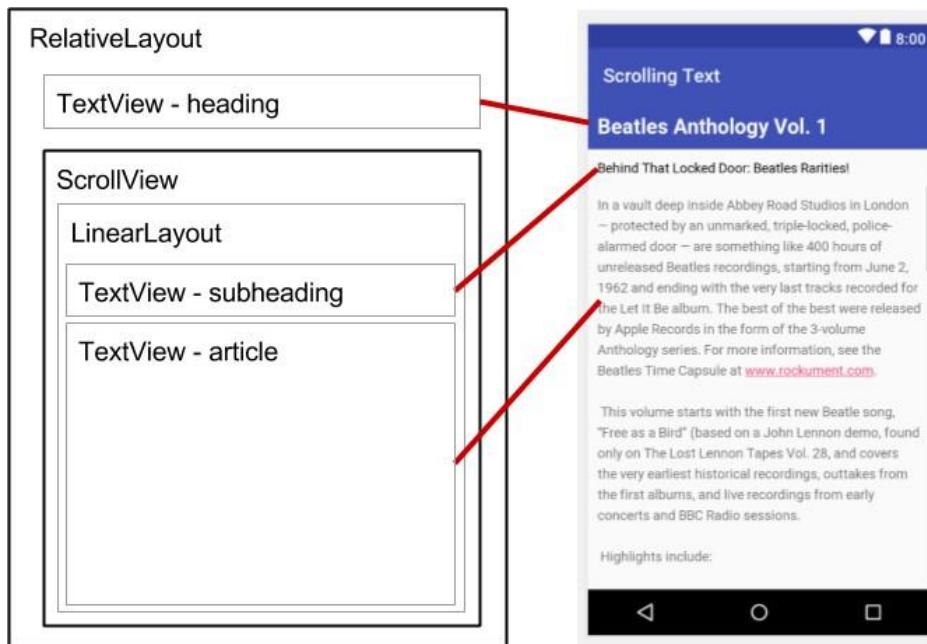
```
        android:lineSpacingExtra="@dimen/line_spacing"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_text" />

    </ScrollView>
</RelativeLayout>
```

Task 3: Scroll multiple elements

As noted before, a ScrollView can contain only one child View (such as the article TextView you created). However, that View can be another ViewGroup that contains 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 the article to scroll along with the article, add a LinearLayout within the ScrollView, and move the subheading and article into the LinearLayout. The LinearLayout becomes the single child View in the ScrollView as shown in the figure below, and the user can scroll the entire LinearLayout: the subheading and the article.



3.1 Add a LinearLayout to the ScrollView

1. Open the **activity_main.xml** file of the ScrollingText app project, and select the **Text** tab to edit the XML code (if it is not already selected).
2. Add a **LinearLayout** above the **articleTextView** within the **ScrollView**. As you enter **<LinearLayout**, Android Studio automatically adds **</LinearLayout>** to the end, and presents the **android:layout_width** and **android:layout_height** attributes with suggestions. Choose **match_parent** and **wrap_content** from the suggestions for its width and height, respectively. The code at the beginning of the **ScrollView** now looks like this:

```

<ScrollView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/article_subheading">

    <LinearLayout

```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content">></LinearLayout>

<TextView
    android:id="@+id/article"
```

You use match_parent to match the width of the parent ViewGroup. You use wrap_content to resize the LinearLayout so it is just big enough to enclose its contents.

3. Move the ending </LinearLayout> code *after* the articleTextView but *before* the closing </ScrollView>.

The LinearLayout now includes the articleTextView, and is completely inside the ScrollView.

4. Add the android:orientation="vertical" attribute to the LinearLayout to set its orientation to vertical.
5. Choose **Code > Reformat Code** to indent the code correctly.

The LinearLayout now looks like this:

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

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

        <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>
```

3.2 Move UI elements within the LinearLayout

The LinearLayout now has only one UI element—the article TextView. You want to include the article_subheading TextView in the LinearLayout so that both will scroll.

1. To move the article_subheading TextView, select the code, choose **Edit > Cut**, click above the article TextView inside the LinearLayout, and choose **Edit > Paste**.
2. Remove the android:layout_below="@+id/article_heading" attribute from the article_subheading TextView. Because this TextView is now within the LinearLayout, this attribute would conflict with the LinearLayout attributes.
3. Change the ScrollView layout attribute from
 android:layout_below="@+id/article_subheading" to
 android:layout_below="@+id/article_heading". Now that the subheading is part of the LinearLayout, the ScrollView must be placed below the heading, not the subheading.

The XML code for the ScrollView is now as follows:

```
<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>
```

4. Run the app.

Swipe up and down to scroll the article, and notice that the subheading now scrolls along with the article while the heading stays in place.

Solution code

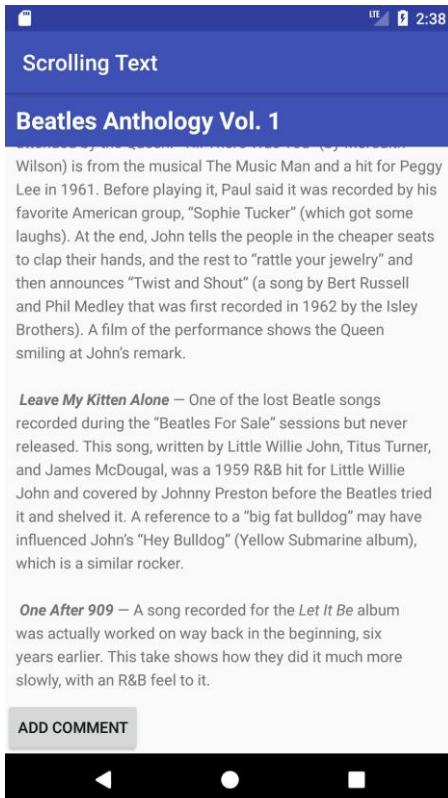
Android Studio project: [ScrollingText](#)

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: Add another UI element—a [Button](#)—to the LinearLayout inside the ScrollView so that it scrolls with the text.

-
- Make the Button appear below the article. The user scrolls to the end of the article to see the Button.
 - Use the text **Add Comment** for the Button. For this challenge, there is no need to create a button-handling method; all you have to do is put the Button element in the proper place in the layout.



Challenge solution code

Android Studio project: [ScrollingTextChallenge](#)

Summary

- Use a [ScrollView](#) to scroll a single child View (such as a [TextView](#)). A [ScrollView](#) can hold only one child View or [ViewGroup](#).
- Use a [ViewGroup](#) such as [LinearLayout](#) as a child View within a [ScrollView](#) to scroll more than one View element. Enclose the elements within the [LinearLayout](#).
- Display free-form text in a [TextView](#) with HTML formatting tags for bold and italics.
- Use \n as an end-of-line character in free-form text to keep a paragraph from running into the next paragraph.
- Use the `android:autoLink="web"` attribute to make web links in the text clickable.

Related concepts

The related concept documentation is in [1.3 Text and scrolling views](#).

Learn more

Android Studio documentation:

- [Android Studio download page](#)
- [Meet Android Studio](#)

Android developer documentation:

- [ScrollView](#)
- [LinearLayout](#)
- [RelativeLayout](#)

- [View](#)

-
- [Button](#)
 - [TextView](#)
 - [String resources](#)
 - [Relative Layout](#)

Other:

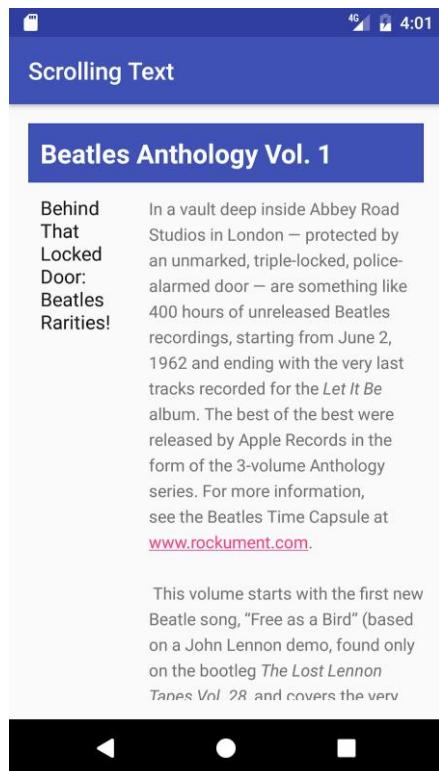
- Android Developers Blog: [Linkify your Text!](#)
- Codepath: [Working with a TextView](#)

Homework

Change an app

Open the [ScrollingText2 app](#) that you created in the [Working with TextView Elements](#) lesson.

1. Change the subheading so that it wraps within a column on the left that is 100 dp wide, as shown below.
2. Place the text of the article to the right of the subheading as shown below.



Answer these questions

Question 1

How many views can you use within a ScrollView? Choose one:

- One view only
- One view or one view group
- As many as you need

Question 2

Which XML attribute do you use in a LinearLayout to show views side by side? Choose one:

- android:orientation="horizontal"
- android:orientation="vertical"
- android:layout_width="wrap_content"

Question 3

Which XML attribute do you use to define the width of the LinearLayout inside the scrolling view?
Choose one:

- android:layout_width="wrap_content"
- android:layout_width="match_parent"
- android:layout_width="200dp"

Submit your app for grading

Guidance for graders

Check that the app has the following features:

- The layout shows the subheading in the left column and the article text in the right

column, as shown in the above figure.

-
- The ScrollView includes a LinearLayout with two TextView elements.
 - The LinearLayout orientation is set to horizontal.

Lesson 1.4: Learn to help yourself

Introduction

What you should already know

You should be able to:

- Understand the basic workflow of Android Studio.
- Create an app from scratch using the Empty Activity template.
- Use the layout editor.

What you'll learn

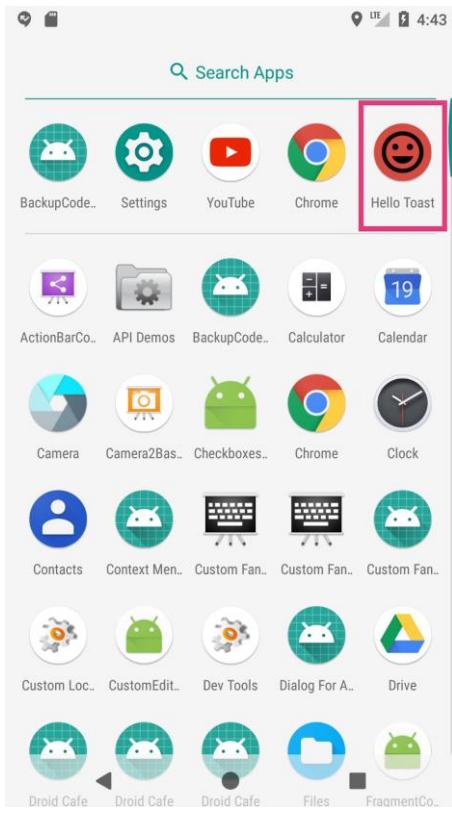
- Where to find developer information and resources.
- How to add a launcher icon to your app.
- How to look for help when you're developing your Android apps.

What you'll do

- Explore some of the many resources available to Android developers of all levels.
- Add a launcher icon for your app.

App overview

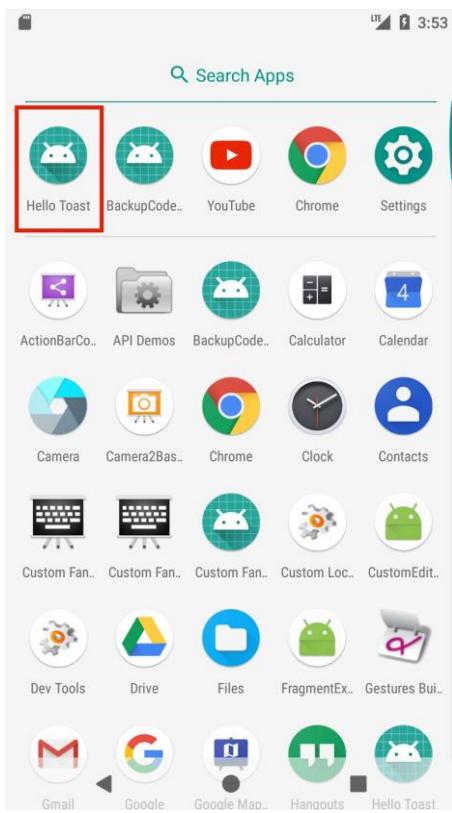
You will add a launcher icon to the HelloToast app you created previously or to a new app.



Task 1: Change the launcher icon

Each new app you create with Android Studio starts with a standard launcher icon that represents the app. The launcher icon appears in the Google Play store listing. When users search the Google Play store, the icon for your app appears in the search results.

When a user has installed the app, the launcher icon appears on the device in various places including the home screen and Search Apps screen. For example, the HelloToast app appears in the Search Apps screen of the emulator with the standard icon for new app projects, as shown below.



Changing the launcher icon is a simple step-by-step process that introduces you to Android Studio's image asset features. In this task you also learn more about accessing the official Android documentation.

1.1 Explore the official Android documentation

You can find the official Android developer documentation at developer.android.com.

This documentation contains a wealth of information that is kept current by Google.

16. Go to developer.android.com/design/.

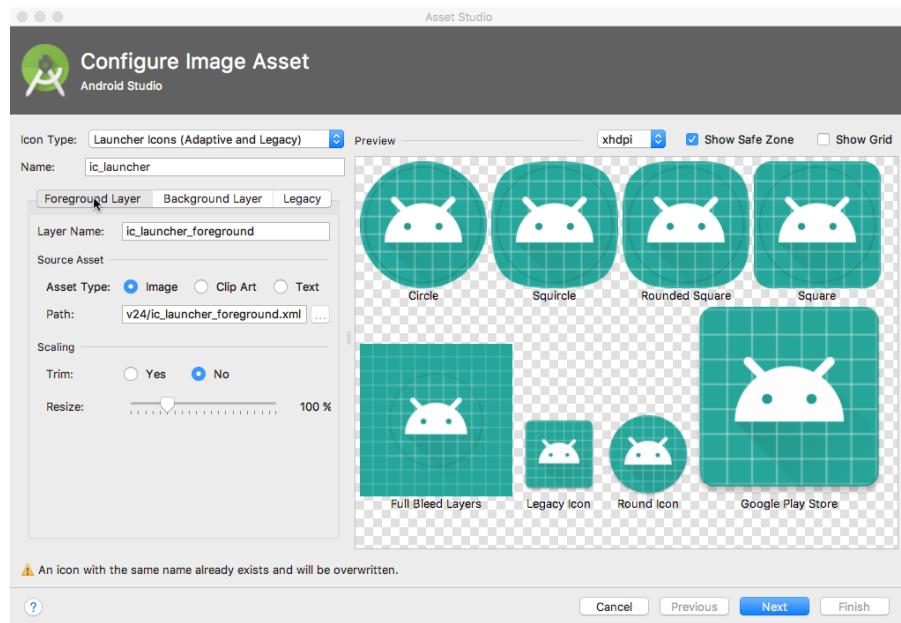
This section is about Material Design, which is a conceptual design philosophy that outlines how apps should look and work on mobile devices. Navigate the links to learn more about Material Design. For example, visit the [Style](#) section to learn more about the use of color and other topics.

17. Go to [developer.android.com/docs/](#) to find API information, reference documentation, tutorials, tool guides, and code samples.
18. Go to [developer.android.com/distribute/](#) to find information about putting an app 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](#).

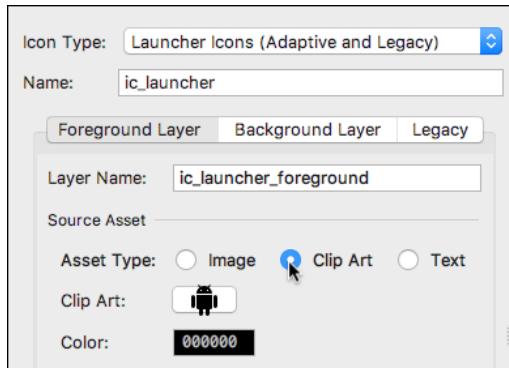
1.2 Add an image asset for the launcher icon

To add a clip-art image as the launcher icon, follow these steps:

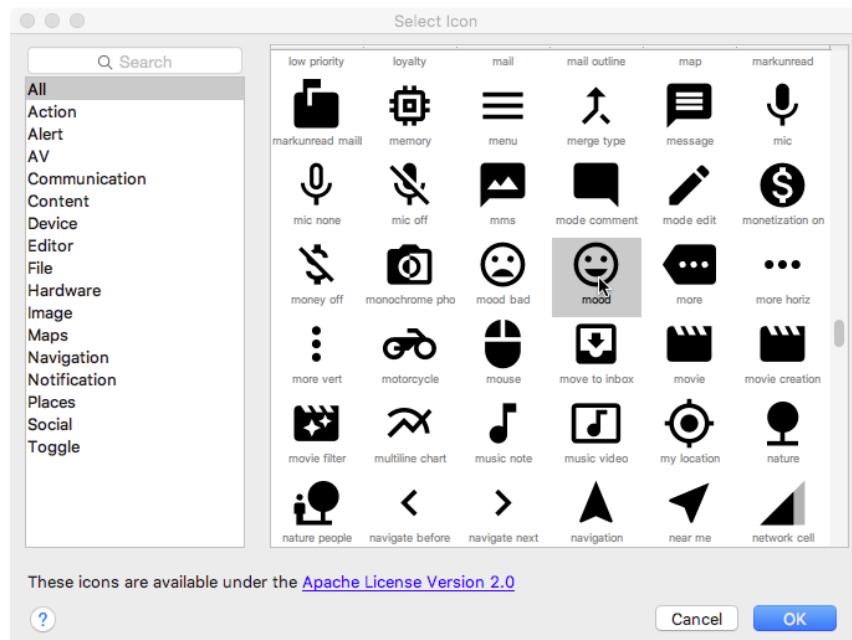
1. Open the HelloToast app project from the previous lesson on using the layout editor, or create a new app project.
2. In the **Project > Android** pane, **right-click** (or **Control-click**) the **res** folder and select **New > Image Asset**. The Configure Image Asset window appears.



3. In the **Icon Type** field, select **Launcher Icons (Adaptive & Legacy)** if it's not already selected.
4. Click the **Foreground Layer** tab, select **Clip Art** for the **Asset Type**.

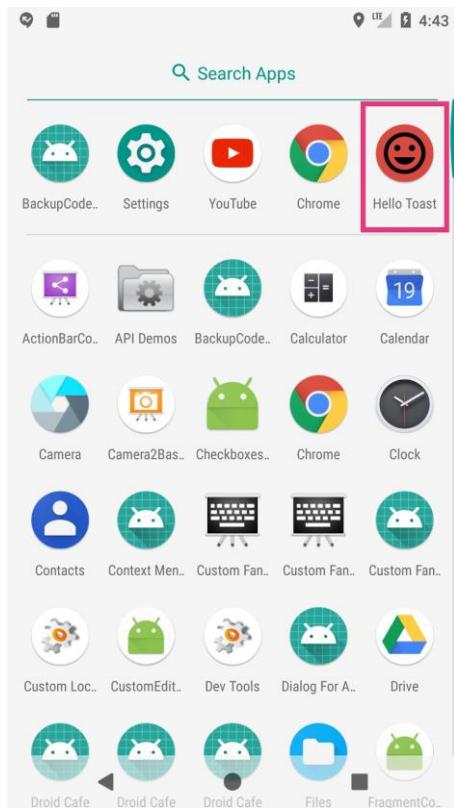


5. Click the icon in the **Clip Art** field. Icons appear from the material design icon set.
6. Browse the Select Icon window, choose an appropriate icon (such as the mood icon to suggest a good mood), and then click **OK**.



7. Click the **Background Layer** tab, choose **Color** as the **Asset Type**, and then click the color chip to select a color to use as the background layer.
8. Click the **Legacy** tab and review the default settings. Confirm that you want to generate legacy, round, and Google Play Store icons. Click **Next** when finished.
9. Run the app.

Android Studio automatically adds the launcher images to the **mipmap** directories for the different densities. As a result, the app launch icon changes to the new icon after you run the app, as shown below.



Tip: See [Launcher Icons](#) to learn more about how to design effective launcher icons.

Task 2: Use project templates

Android Studio provides templates for common and recommended app and activity designs. Using built-in templates saves time, and helps you follow design best practices.

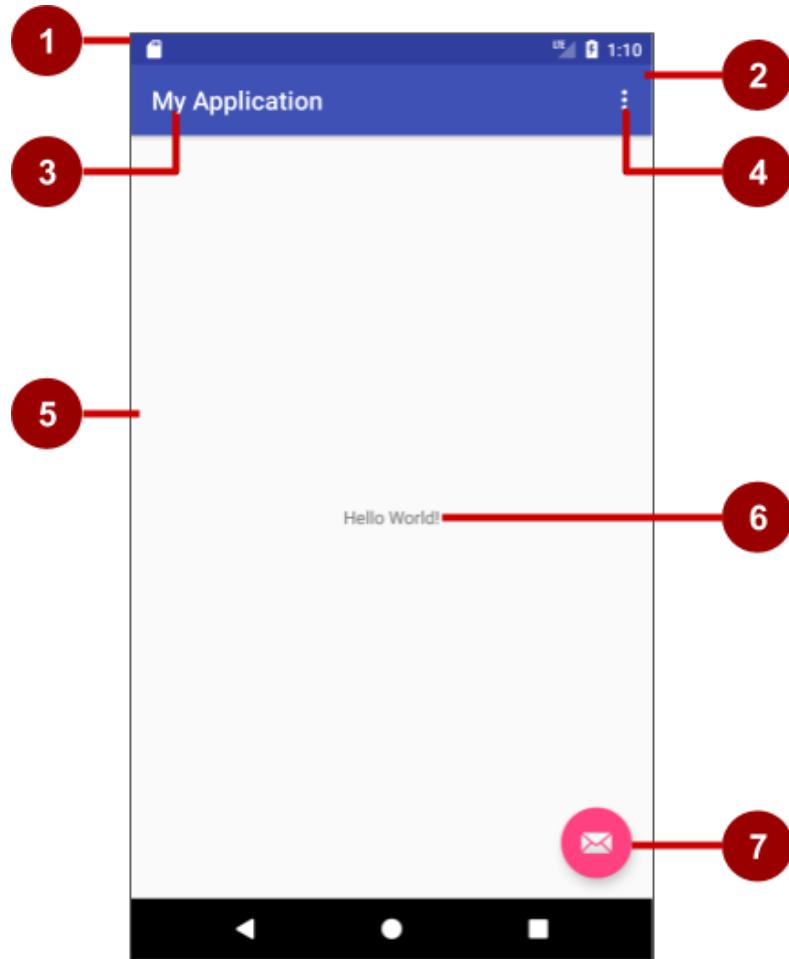
Each template incorporates a skeleton activity and user interface. You've already used the Empty Activity template. The Basic Activity template has more features and incorporates recommended app features, such as the options menu that appears in the app bar.

2.1 Explore the Basic Activity architecture

The Basic Activity template is a versatile template provided by Android Studio to assist you in jump-starting your app development.

1. In Android Studio, create a new project with the Basic Activity template.
2. Build and run the app.
3. Identify the labeled parts in the figure and table below. Find their equivalents on your device or emulator screen. Inspect the corresponding Java code and XML files described in the table.

Being familiar with the Java source code and XML files will help you extend and customize this template for your own needs.



Architecture of the Basic Activity template

#	UI Description	Code reference
1	Status bar The Android system provides and controls the status bar.	Not visible in the template code. It's possible to access it from your activity. For example, you can hide the status bar , if necessary.
2	AppBarLayout > Toolbar The app bar (also called the action bar) provides visual structure, standardized visual elements, and navigation. For backwards	In activity_main.xml, look for android.support.v7.widget.Toolbar

	compatibility, the AppBarLayout in the template embeds a Toolbar with the same functionality as an ActionBar .	inside android.support.design.widget.AppBarLayout. Change the toolbar to change the appearance of its parent, the app bar. For an example, see the App Bar Tutorial .
3	Application name This is derived from your package name, but can be anything you choose.	In AndroidManifest.xml: android:label="@string/app_name"
4	Options menu overflow button Menu items for the activity, as well as global options, such as Search and Settings for the app. Your app menu items go into this menu.	In MainActivity.java: onOptionsItemSelected() implements what happens when a menu item is selected. res > menu > menu_main.xml Resource that specifies the menu items for the options menu.
5	Layout ViewGroup The CoordinatorLayout ViewGroup is a feature-rich layout that provides mechanisms for View(UI) elements to interact. Your app's user interface goes inside the content_main.xml file included within this ViewGroup.	In activity_main.xml: There are no views specified in this layout; rather, it includes another layout with an include layout instruction to include @layout/content_main where the views are specified. This separates system views from the views unique to your app.
6	TextView In the example, used to display "Hello World". Replace this with the UI elements for your app.	In content_main.xml: All your app's UI elements are defined in this file.
7	Floating action button (FAB)	In activity_main.xml as a UI element using a clip-art icon. MainActivity.java includes a stub in onCreate() that sets an onClick() listener for the FAB.

2.2 Customizing the app produced by the template

Change the appearance of the app produced by the Basic Activity template. For example, you can change the color of the app bar to match the status bar (which on some devices is a darker shade of

the same primary color). You may also want to remove the floating action button if you are not going to use it.

1. Change the color of the app bar (Toolbar) in activity_main.xml by changing the android:background to "?attr/colorPrimaryDark", which sets the app bar color to a darker primary color that matches the status bar:

```
android:background="?attr/colorPrimaryDark"
```

2. To remove the floating action button, start by removing the stub code in onCreate() that sets an onClick() listener for the button. Open **MainActivity** and delete the following block of code:

```
FloatingActionButton fab = (FloatingActionButton)
    findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Replace with your own action",
            Snackbar.LENGTH_LONG)
            .setAction("Action", null).show();
    }
});
```

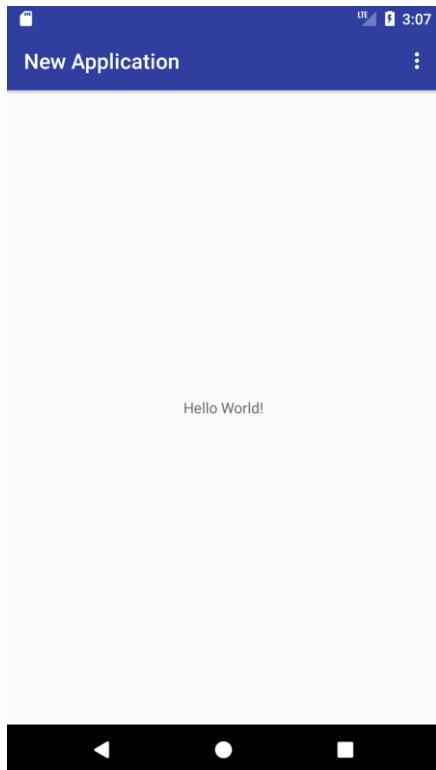
3. To remove the floating action button from the layout, delete the following block of XML code from activity_main.xml:

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    app:srcCompat="@android:drawable/ic_dialog_email" />
```

-
4. Change the name of the app that is displayed in the app bar by changing the app_name string resource in strings.xml to the following:

```
<string name="app_name">New Application</string>
```

5. Run the app. The floating action button no longer appears, the name has changed, and the app bar background color has changed.



Tip: See [Accessing Resources](#) for details on the XML syntax for accessing resources.

2.3 Explore how to add activities using templates

For the practicals so far, you've used the Empty Activity and Basic Activity templates. In later lessons, the templates you use vary, depending on the task.

These activity templates are also available from inside your project, so that you can add more activities to your app after the initial project setup. (You learn more about the Activity class in another chapter.)

1. Create a new app project or choose an existing project.
2. In the **Project > Android** pane, right-click the **java** folder.
3. Choose **New > Activity > Gallery**.
4. Add an Activity. For example, click **Navigation Drawer Activity** to add an Activity with a navigation drawer to your app.
5. Double-click the layout files for the Activity to display them in the layout editor.

Task 3: Learn from example code

Android Studio and the Android documentation provide many code samples that you can study, copy, and incorporate with your projects.

3.1 Android code samples

You can explore hundreds of code samples directly from within Android Studio.

1. In Android Studio, choose **File > New > Import Sample**.
2. Browse the samples.
3. Choose a sample and click **Next**.
4. Accept the defaults and click **Finish**.

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

3.2 Use the SDK Manager to install offline documentation

Installing Android Studio also installs essentials of the Android SDK (Software Development Kit). However, additional libraries and documentation are available, and you can install them using the SDK Manager.

1. Choose **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 Platforms** tab. You can install additional versions of the Android system from here.
5. Click the **SDK Update Sites** tab. Android Studio checks the listed and selected sites regularly for updates.
6. 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.
7. Select the checkbox for "Documentation for Android SDK" if it is not already installed, and click **Apply**.
8. When the installation finishes, click **Finish**.
9. Navigate to the **sdk** directory you copied above, and open the **docs** directory.
10. Find **index.html** and open it.

Task 4: Many more resources

- The [Android Developer YouTube channel](#) is a great source of tutorials and tips.
- The Android team posts news and tips in the [official Android blog](#).
- [Stack Overflow](#) is a community of programmers helping each other. If you run into a problem, chances are high that someone has already posted an answer. Try posting a question such as "How do I set up and use ADB over WiFi?" or "What are the most common memory leaks in Android development?"
- And last but not least, type your questions into Google search, and 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?"

4.1 Search on Stack Overflow using tags

Go to [Stack Overflow](#) and type [android] in the search box. The [] brackets indicate that you want to search for posts that have been tagged as being about Android.

You can combine tags and search terms to make your search more specific. Try these searches:

- [android] and [layout]
- [android] "hello world"

To learn more about the many ways in which you can search on Stack Overflow, see the [Stack Overflow help center](#).

Summary

- Official Android Developer Documentation: [developer.android.com](#)
- Material Design is a conceptual design philosophy that outlines how apps should look and work on mobile devices.
- The [Google Play](#) store is Google's digital distribution system for apps developed with

the Android SDK.

-
- Android Studio provides templates for common and recommended app and activity designs. These templates offer working code for common use cases.
 - When you create a project, you can choose a template for your first activity.
 - While you are further developing your app, activities and other app components can be created from built-in templates.
 - Android Studio contains many code samples that you can study, copy, and incorporate with your projects.

Related concept

The related concept documentation is in [1.4: Resources to help you learn](#).

Learn more

Android Studio documentation:

- [Meet Android Studio](#)
- [Developer workflow basics](#)

Android developer documentation:

- [Android developer site](#)
- [Google Developers Training](#)
- [Layouts](#)
- [App resources overview](#)
- [Layouts](#)
- [Menus](#)
- [TextView](#)
- [String resources](#)
- [App Manifest](#)

Code samples:

- [Source code for exercises on GitHub](#)
- [Android code samples for developers](#)

Videos:

- [Android Developer YouTube channel](#)
- [Udacity online courses](#)

Other:

- [Official Android blog](#)
- [Android Developers blog](#)
- [Google Developers Codelabs](#)
- [Stack Overflow](#)
- [Android vocabulary](#)

Homework

Load a sample app and explore resources

1. Load one of the sample apps into Android Studio.
2. Open one of the Java activity files in the app. Look for a class, type, or procedure that you're not familiar with and look it up in the Android Developer documentation.
3. Go to Stack Overflow and search for questions on the same topic.
4. Change the launcher icon. Use an icon that's available in the image assets section of Android Studio.

Answer these questions

Question 1

Within an Android Studio project, what menu command can you use to open the list of sample apps? Choose one:

-
- **File > Open**
 - **File > New > Import Sample**

-
- **File > New > Import Module**
 - **File > New > Import Project**

Question 2

Which buttons does the Basic Activity template provide as part of the UI? Choose two:

- Navigation buttons
- Options menu overflow button
- Floating action button
- Button class button with the text "Button"

Question 3

Which source of documentation is the official documentation for Android developers? Choose one:

- stackoverflow.com
- officialandroid.blogspot.com
- developer.android.com
- github.com

Submit your app for grading

Guidance for graders

The result is a new app, or a version of Hello Toast, with a new launcher icon that appears in the Search Apps screen of an Android device.

Lesson 2.1: Activities and intents

Introduction

An [Activity](#) represents a single screen in your app with which your user can perform a single, focused task such as taking a photo, sending an email, or viewing a map. An activity is usually presented to the user as a full-screen window.

An app usually consists of multiple screens that are loosely bound to each other. Each screen is an activity. Typically, one activity in an app is specified as the "main" activity (`MainActivity.java`), which is presented to the user when the app is launched. The main activity can then start other activities to perform different actions.

Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, that new activity is pushed onto the back stack and takes user focus. The back stack follows basic "last in, first out" stack logic. When the user is done with the current activity and presses the Back button, that activity is popped from the stack and destroyed, and the previous activity resumes.

An activity is started or activated with an *intent*. An [Intent](#) is an asynchronous message that you can use in your activity to request an action from another activity, or from some other app component. You use an intent to start one activity from another activity, and to pass data between activities.

An Intent can be *explicit* or *implicit*:

- An *explicit intent* is one in which you know the target of that intent. That is, you already know the fully qualified class name of that specific activity.
- An *implicit intent* is one in which you do not have the name of the target component, but you have a general action to perform.

In this practical you create explicit intents. You find out how to use implicit intents in a later practical.

What you should already know

You should be able to:

- Create and run apps in Android Studio.
- Use the layout editor to create a layout within a ConstraintLayout
- Edit the layout XML code.

-
- Add onClick functionality to a Button.

What you'll learn

- How to create a new Activity in Android Studio.
- How to define parent and child activities for Up navigation.
- How to start an Activity with an explicit Intent.
- How to pass data between each Activity with an explicit Intent.

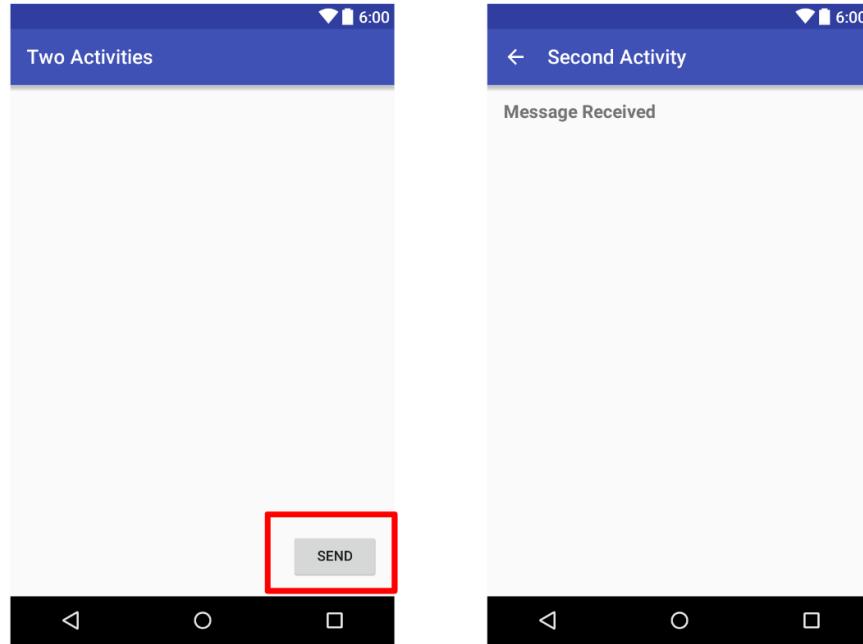
What you'll do

- Create a new Android app with a main Activity and a second Activity.
- Pass some data (a string) from the main Activity to the second using an Intent, and display that data in the second Activity.
- Send a second different bit of data back to the main Activity, also using an Intent.

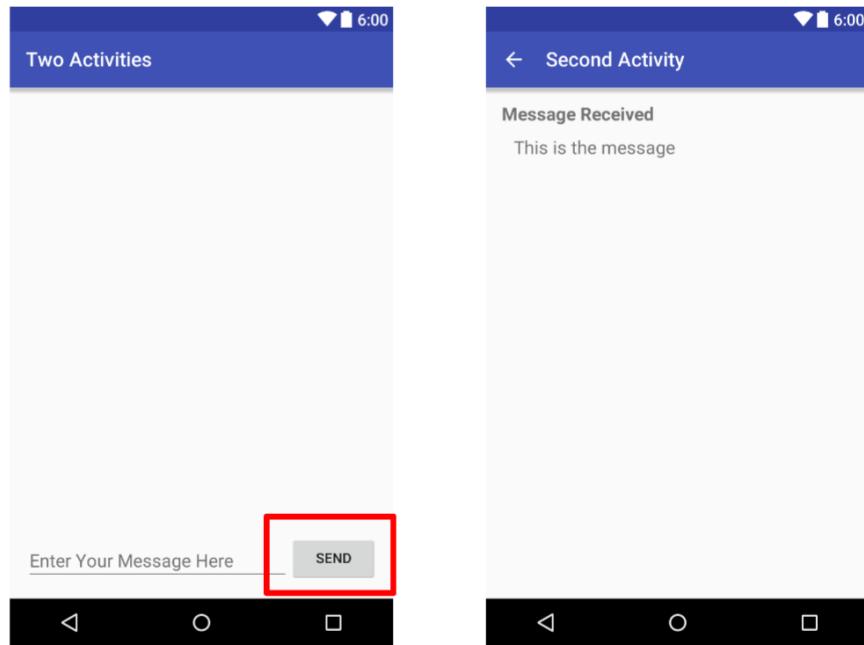
App overview

In this chapter you create and build an app called Two Activities that, unsurprisingly, contains two Activity implementations. You build the app in three stages.

In the first stage, you create an app whose main activity contains one button, **Send**. When the user clicks this button, your main activity uses an intent to start the second activity.

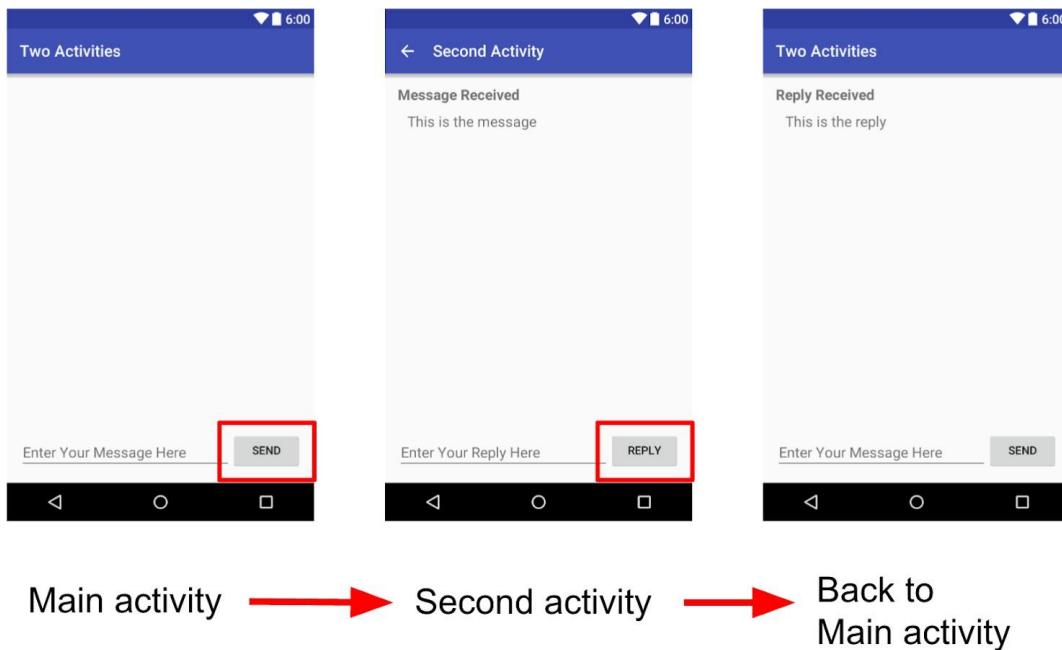


In the second stage, you add an EditTextview to the main activity. The user enters a message and clicks **Send**. The main activity uses an intent to start the second activity and send the user's message to the second activity. The second activity displays the message it received.



Main activity → Second activity

In the final stage of creating the Two Activities app, you add an `EditText` and a **Reply** button to the second activity. The user can now type a reply message and tap **Reply**, and the reply is displayed on the main activity. At this point, you use an intent to pass the reply back from the second activity to the main activity.



Task 1: Create the TwoActivities project

In this task you set up the initial project with a main Activity, define the layout, and define a skeleton method for the onClick button event.

1.1 Create the TwoActivities project

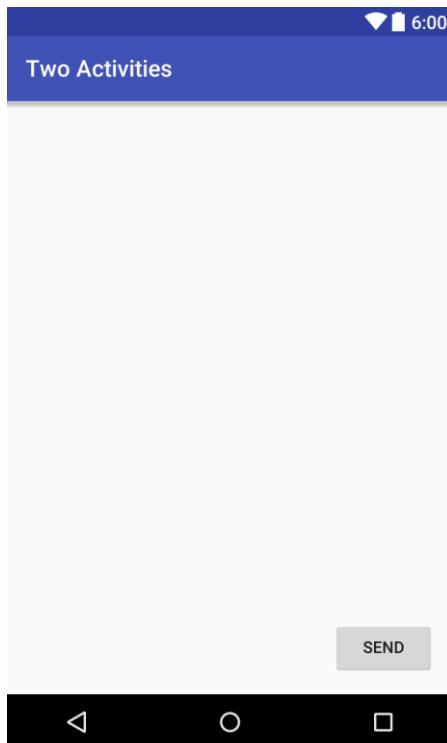
1. Start Android Studio and create a new Android Studio project.

Name your app **Two Activities** and choose the same **Phone and Tablet** settings that you used in previous practicals. The project folder is automatically named TwoActivities, and the app name that appears in the app bar will be "Two Activities".

2. Choose **Empty Activity** for the Activity template. Click **Next**.
3. Accept the default Activity name (MainActivity). Make sure the **Generate Layout file** and **Backwards Compatibility (AppCompat)** options are checked.
4. Click **Finish**.

1.2 Define the layout for the main Activity

1. Open **res > layout > activity_main.xml** in the **Project > Android** pane. The layout editor appears.
2. Click the **Design** tab if it is not already selected, and delete the **TextView**(the one that says "Hello World") in the **Component Tree** pane.
3. With Autoconnect turned on (the default setting), drag a **Button** from the **Palette** pane to the lower right corner of the layout. Autoconnect creates constraints for the **Button**.
4. In the **Attributes** pane, set the **ID** to **button_main**, the **layout_width** and **layout_height** to **wrap_content**, and enter **Send** for the Text field. The layout should now look like this:



5. Click the **Text** tab to edit the XML code. Add the following attribute to the Button:

```
android:onClick="launchSecondActivity"
```

The attribute value is underlined in red because the `launchSecondActivity()` method has not yet been created. Ignore this error for now; you fix it in the next task.

6. Extract the string resource, as described in a previous practical, for "Send" and use the name `button_main` for the resource.

The XML code for the Button should look like the following:

```
<Button
```

```
    android:id="@+id/button_main"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginRight="16dp"
    android:text="@string/button_main"
    android:onClick="launchSecondActivity"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toRightOf="parent" />
```

1.3 Define the Button action

In this task you implement the launchSecondActivity() method you referred to in the layout for the android:onClick attribute.

1. Click on "launchSecondActivity" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'launchSecondActivity(View)' in 'MainActivity'.**

The MainActivity file opens, and Android Studio generates a skeleton method for the launchSecondActivity() handler.

3. Inside launchSecondActivity(), add a Log statement that says "Button Clicked!"

```
Log.d(LOG_TAG, "Button clicked!");
```

LOG_TAG will show as red. You add the definition for that variable in a later step.

4. At the top of the MainActivity class, add a constant for the LOG_TAG variable:

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

This constant uses the name of the class itself as the tag.

5. Run your app. When you click the **Send** button you see the "Button Clicked!" message in the **Logcat** pane. If there's too much output in the monitor, type **MainActivity** into the search box, and the **Logcat** pane will only show lines that match that tag.

The code for **MainActivity** should look as follows:

```
package com.example.android.twoactivities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity {
    private static final String LOG_TAG =
            MainActivity.class.getSimpleName();

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

    public void launchSecondActivity(View view) {
        Log.d(LOG_TAG, "Button clicked!");
    }
}
```

Task 2: Create and launch the second Activity

Each new activity you add to your project has its own layout and Java files, separate from those of the main activity. They also have their own <activity> elements in the AndroidManifest.xml file. As with the main activity, new activity implementations that you create in Android Studio also extend from the AppCompatActivity class.

Each activity in your app is only loosely connected with other activities. However, you can define an activity as a parent of another activity in the AndroidManifest.xml file. This parent-child relationship enables Android to add navigation hints such as left-facing arrows in the title bar for each activity.

An activity communicates with other activities (in the same app and across different apps) with an intent. An Intent can be *explicit* or *implicit*:

- An *explicit intent* is one in which you know the target of that intent; that is, you already know the fully qualified class name of that specific activity.
- An *implicit intent* is one in which you do not have the name of the target component, but have a general action to perform.

In this task you add a second activity to our app, with its own layout. You modify the AndroidManifest.xml file to define the main activity as the parent of the second activity. Then you modify the launchSecondActivity() method in MainActivity to include an intent that launches the second activity when you click the button.

2.1 Create the second Activity

1. Click the **app** folder for your project and choose **File > New > Activity > Empty Activity**.
2. Name the new Activity **SecondActivity**. Make sure **Generate Layout File** and **Backwards Compatibility (AppCompat)** are checked. The layout name is filled in as `activity_second`. Do *not* check the **Launcher Activity** option.
3. Click **Finish**. Android Studio adds both a new Activity layout (`activity_second.xml`) and a new Java file (`SecondActivity.java`) to your project for the new Activity. It also updates the `AndroidManifest.xml` file to include the new Activity.

2.2 Modify the AndroidManifest.xml file

1. Open **manifests > AndroidManifest.xml**.
2. Find the `<activity>` element that Android Studio created for the second Activity.

```
<activity android:name=".SecondActivity"></activity>
```

3. Replace the entire `<activity>` element with the following:

```
<activity android:name=".SecondActivity"
    android:label = "Second Activity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=
            "com.example.android.twoactivities.MainActivity" />
</activity>
```

The `label` attribute adds the title of the Activity to the app bar.

With the `parentActivityName` attribute, you indicate that the main activity is the parent of the second activity. This relationship is used for Up navigation in your app: the app bar for the second activity will have a left-facing arrow so the user can navigate "upward" to the main activity.

With the `<meta-data>` element, you provide additional arbitrary information about the activity in the form of key-value pairs. In this case the metadata attributes do the same thing as the `android:parentActivityName` attribute—they define a relationship between two activities for upward navigation. These metadata attributes are required for older versions of Android, because the `android:parentActivityName` attribute is only available for API levels 16 and higher.

4. Extract a string resource for "Second Activity" in the code above, and use `activity2_name` as the resource name.

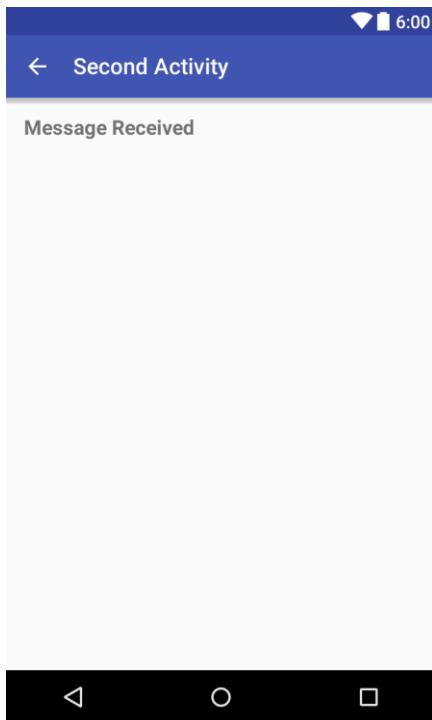
2.3 Define the layout for the second Activity

1. Open **activity_second.xml** and click the **Design** tab if it is not already selected.
2. Drag a **TextView** from the **Palette** pane to the top left corner of the layout, and add constraints to the top and left sides of the layout. Set its attributes in the **Attributes** pane as follows:

Attribute	Value
id	text_header
Top margin	16
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	Message Received
textAppearance	AppCompat.Medium
textStyle	B(bold)

The value of **textAppearance** is a special Android theme attribute that defines basic font styles. You learn more about themes in a later lesson.

The layout should now look like this:



3. Click the **Text** tab to edit the XML code, and extract the "Message Received" string into a resource named `text_header`.
4. Add the `android:layout_marginLeft="8dp"` attribute to the `TextView` to complement the `layout_marginStart` attribute for older versions of Android.

The XML code for `activity_second.xml` should be as follows:

```
<android.support.constraint.ConstraintLayout  
    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"  
        tools:context="com.example.android.twoactivities.SecondActivity">  
  
    <TextView  
        android:id="@+id/text_header"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header"
        android:textAppearance=
            "@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

2.4 Add an Intent to the main Activity

In this task you add an explicit Intent to the main Activity. This Intent is used to activate the second Activity when the **Send** button is clicked.

1. Open **MainActivity**.
2. Create a new Intent in the launchSecondActivity() method.

The Intent constructor takes two arguments for an explicit Intent: an application [Context](#) and the specific component that will receive that Intent. Here you should use this as the Context, and SecondActivity.class as the specific class:

```
Intent intent = new Intent(this, SecondActivity.class);
```

3. Call the startActivity() method with the new Intent as the argument.

```
startActivity(intent);
```

-
4. Run the app.

When you click the **Send** button, MainActivity sends the Intent and the Android system launches SecondActivity, which appears on the screen. To return to MainActivity, click the **Up** button (the left arrow in the app bar) or the Back button at the bottom of the screen.

Task 3: Send data from the main Activity to the second Activity

In the last task, you added an explicit intent to MainActivity that launched SecondActivity. You can also use an intent to *send data* from one activity to another while launching it.

Your intent object can pass data to the target activity in two ways: in the `data` field, or in the intent `extras`. The intent data is a URI indicating the specific data to be acted on. If the information you want to pass to an activity through an intent is not a URI, or you have more than one piece of information you want to send, you can put that additional information into the `extras` instead.

The intent `extras` are key/value pairs in a [Bundle](#). A Bundle is a collection of data, stored as key/value pairs. To pass information from one activity to another, you put keys and values into the intent extra Bundle from the sending activity, and then get them back out again in the receiving activity.

In this task, you modify the explicit intent in MainActivity to include additional data (in this case, a user-entered string) in the intent extra Bundle. You then modify SecondActivity to get that data back out of the intent extra Bundle and display it on the screen.

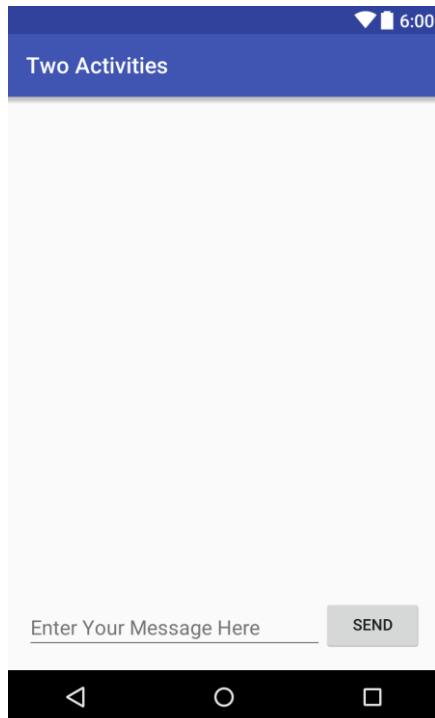
3.1 Add an EditText to the MainActivity layout

1. Open **activity_main.xml**.
2. Drag a **Plain Text** (EditText) element from the **Palette** pane to the bottom of the layout, and add constraints to the left side of the layout, the bottom of the layout, and the left side of the **Send** Button. Set its attributes in the **Attributes** pane as follows:

Attribute	Value
<code>id</code>	<code>editText_main</code>
Right margin	8

Left margin	8
Bottom margin	16
layout_width	match_constraint
layout_height	wrap_content
inputType	textLongMessage
hint	Enter Your Message Here
text	(Delete any text in this field)

The new layout in activity_main.xml looks like this:



3. Click the **Text** tab to edit the XML code, and extract the "Enter Your Message Here" string into a resource named editText_main.

The XML code for the layout should look something like the following.

```
<android.support.constraint.ConstraintLayout 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"
    tools:context="com.example.android.twoactivities.MainActivity">

    <Button
        android:id="@+id/button_main" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp" android:text="@string/button_main"
        android:onClick="launchSecondActivity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <EditText
        android:id="@+id/editText_main" android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp" android:ems="10"
        android:hint="@string/editText_main"
        android:inputType="textLongMessage"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button2" app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

3.2 Add a string to the Intent extras

The Intent *extras* are key/value pairs in a [Bundle](#). A Bundle is a collection of data, stored as key/value pairs. To pass information from one Activity to another, you put keys and values into the Intent extra Bundle from the sending Activity, and then get them back out again in the receiving Activity.

1. Open **MainActivity**.
2. Add a public constant at the top of the class to define the key for the Intent extra:

```
public static final String EXTRA_MESSAGE =  
    "com.example.android.twoactivities.extra.MESSAGE";
```

3. Add a private variable at the top of the class to hold the EditText:

```
private EditText mMessageEditText;
```

4. In the onCreate() method, use [findViewById\(\)](#) to get a reference to the EditText and assign it to that private variable:

```
mMessageEditText = findViewById(R.id.editText_main);
```

5. In the launchSecondActivity() method, just under the new Intent, get the text from the EditText as a string:

```
String message = mMessageEditText.getText().toString();
```

6. Add that string to the Intent as an extra with the EXTRA_MESSAGE constant as the key and the string as the value:

```
intent.putExtra(EXTRA_MESSAGE, message);
```

The onCreate() method in MainActivity should now look like the following:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mMessageEditText = findViewById(R.id.editText_main);  
}
```

The launchSecondActivity() method in MainActivity should now look like the following:

```
public void launchSecondActivity(View view) {  
    Log.d(LOG_TAG, "Button clicked!");  
    Intent intent = new Intent(this, SecondActivity.class);  
    String message = mMessageEditText.getText().toString();  
    intent.putExtra(EXTRA_MESSAGE, message);  
    startActivity(intent);  
}
```

3.3 Add a TextView to SecondActivity for the message

1. Open **activity_second.xml**.
2. Drag another **TextView** to the layout underneath the **text_header** **TextView**, and add constraints to the left side of the layout and to the bottom of **text_header**.
3. Set the new **TextView** attributes in the **Attributes** pane as follows:

Attribute	Value
id	text_message
Top margin	8
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	(Delete any text in this field)
textAppearance	AppCompat.Medium

The new layout looks the same as it did in the previous task, because the new **TextView** does not (yet) contain any text, and thus does not appear on the screen.

The XML code for the **activity_second.xml** layout should look something like the following:

```
<android.support.constraint.ConstraintLayout
    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"
    tools:context="com.example.android.twoactivities.SecondActivity">

    <TextView
        android:id="@+id/text_header"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
```

```
        android:layout_marginTop="16dp"
        android:text="@string/text_header"
        android:textAppearance=
            "@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header" />
</android.support.constraint.ConstraintLayout>
```

3.4 Modify SecondActivity to get the extras and display the message

1. Open **SecondActivity** to add code to the `onCreate()` method.
2. Get the Intent that activated this Activity:

```
Intent intent = getIntent();
```

3. Get the string containing the message from the Intent extras using the `MainActivity.EXTRA_MESSAGE` static variable as the key:

```
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

-
4. Use `findViewById()` to get a reference to the `TextView` for the message from the layout:

```
TextView textView = findViewById(R.id.text_message);
```

5. Set the text of the `TextView` to the string from the Intent extra:

```
textView.setText(message);
```

6. Run the app. When you type a message in `MainActivity` and click **Send**, `SecondActivity` launches and displays the message.

The `SecondActivity` `onCreate()` method should look as follows:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_second);  
    Intent intent = getIntent();  
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);  
    TextView textView = findViewById(R.id.text_message);  
    textView.setText(message);  
}
```

Task 4: Return data back to the main Activity

Now that you have an app that launches a new activity and sends data to it, the final step is to return data from the second activity back to the main activity. You also use an intent and intent *extras* for this task.

4.1 Add an EditText and a Button to the SecondActivity layout

1. Open **strings.xml** and add string resources for the Button text and the hint for the EditText that you will add to SecondActivity:

```
<string name="button_second">Reply</string>
<string name="editText_second">Enter Your Reply Here</string>
```

2. Open **activity_main.xml** and **activity_second.xml**.
3. **Copy** the EditText and Button from the **activity_main.xml** layout file and **Paste** them into the **activity_second.xml** layout.
4. In **activity_second.xml**, modify the attribute values for the Button as follows:

Old attribute value	New attribute value
android:id="@+id/button_main"	android:id="@+id/button_second"
android:onClick= "launchSecondActivity"	android:onClick="returnReply"
android:text= "@string/button_main"	android:text= "@string/button_second"

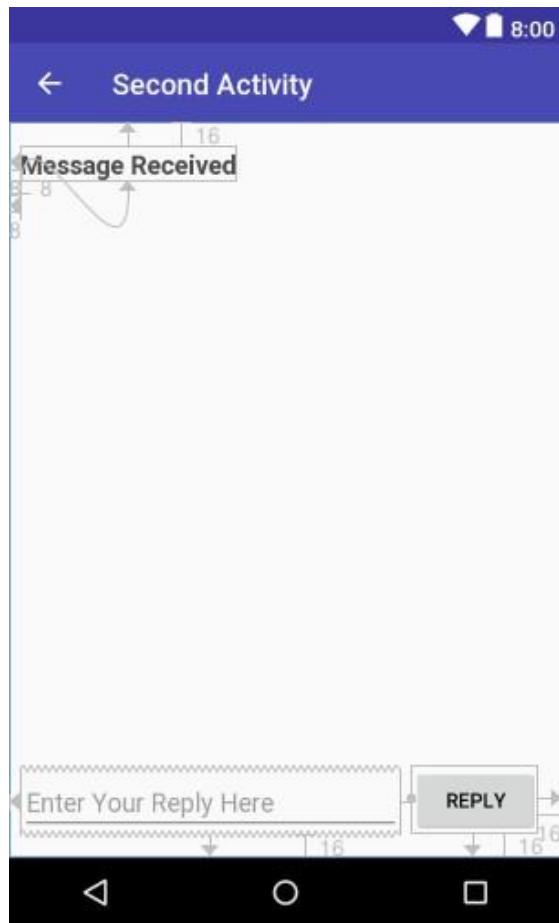
5. In activity_second.xml, modify the attribute values for the EditText as follows:

Old attribute value	New attribute value
android:id="@+id/editText_main"	android:id="@+id/editText_second"
app:layout_constraintEnd_toStartOf="@+id/button"	app:layout_constraintEnd_toStartOf="@+id/button_second"
android:hint="@string/editText_main"	android:hint="@string/editText_second"

6. In the XML layout editor, click on **returnReply**, press Alt+Enter (Option+Return on a Mac), and select **Create 'returnReply(View)' in 'SecondActivity'**.

Android Studio generates a skeleton method for the returnReply() handler. You implement this method in the next task.

The new layout for activity_second.xml looks like this:



The XML code for the activity_second.xml layout file is as follows:

```
<android.support.constraint.ConstraintLayout  
    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"  
        tools:context="com.example.android.twoactivities.SecondActivity">  
  
    <TextView
```

```
        android:id="@+id/text_header"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header" />

    <Button
        android:id="@+id/button_second"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/button_second"
        android:onClick="returnReply"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <EditText
        android:id="@+id/editText_second"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:ems="10"
        android:hint="@string/editText_second"
        android:inputType="textLongMessage"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintEnd_toStartOf="@+id/button_second"
    app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

4.2 Create a response Intent in the second Activity

The response data from the second Activity back to the main Activity is sent in an Intent extra. You construct this return Intent and put the data into it in much the same way you do for the sending Intent.

1. Open **SecondActivity**.
2. At the top of the class, add a public constant to define the key for the Intent extra:

```
public static final String EXTRA_REPLY =
    "com.example.android.twoactivities.extra.REPLY";
```

3. Add a private variable at the top of the class to hold the EditText.

```
private EditText mReply;
```

4. In the onCreate() method, before the Intent code, use findViewById() to get a reference to the EditText and assign it to that private variable:

```
mReply = findViewById(R.id.editText_second);
```

-
5. In the returnReply() method, get the text of the EditText as a string:

```
String reply = mReply.getText().toString();
```

6. In the returnReply() method, create a new intent for the response—*don't* reuse the Intent object that you received from the original request.

```
Intent replyIntent = new Intent();
```

7. Add the replystring from the EditText to the new intent as an Intent extra. Because *extras* are key/value pairs, here the key is EXTRA_REPLY, and the value is the reply:

```
replyIntent.putExtra(EXTRA_REPLY, reply);
```

8. Set the result to RESULT_OK to indicate that the response was successful. The [Activity](#) class defines the result codes, including RESULT_OK and RESULT_CANCELLED.

```
setResult(RESULT_OK, replyIntent);
```

9. Call finish() to close the Activity and return to MainActivity.

```
finish();
```

The code for SecondActivity should now be as follows:

```
public class SecondActivity extends AppCompatActivity {
    public static final String EXTRA_REPLY =
        "com.example.android.twoactivities.extra.REPLY";
    private EditText mReply;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        mReply = findViewById(R.id.editText_second);
        Intent intent = getIntent();
        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
        TextView textView = findViewById(R.id.text_message);
        textView.setText(message);
    }

    public void returnReply(View view) {
        String reply = mReply.getText().toString();
        Intent replyIntent = new Intent();
        replyIntent.putExtra(EXTRA_REPLY, reply);
        setResult(RESULT_OK, replyIntent);
        finish();
    }
}
```

4.3 Add TextView elements to display the reply

MainActivity needs a way to display the reply that SecondActivity sends. In this task you add TextView elements to the activity_main.xml layout to display the reply in MainActivity. To make this task easier, you copy the TextView elements you used in SecondActivity.

1. Open **strings.xml** and add a string resource for the reply header:

```
<string name="text_header_reply">Reply Received</string>
```

-
2. Open **activity_main.xml** and **activity_second.xml**.
 3. Copy the two TextView elements from the activity_second.xml layout file and paste them into the activity_main.xml layout above the Button.
 4. In activity_main.xml, modify the attribute values for the first TextView as follows:

Old attribute value	New attribute value
android:id="@+id/text_header"	android:id="@+id/text_header_reply"
android:text="@string/text_header"	android:text="@string/text_header_reply"

5. In activity_main.xml, modify the attribute values for the second TextView as follows:

Old attribute value	New attribute value
android:id="@+id/text_message"	android:id="@+id/text_message_reply"
app:layout_constraintTop_toBottomOf="@+id/text_header"	app:layout_constraintTop_toBottomOf="@+id/text_header_reply"

6. Add the android:visibility attribute to each TextView to make them initially invisible. (Having them visible on the screen, but without any content, can be confusing to the user.)

android:visibility="invisible"

You will make these TextView elements visible after the response data is passed back from the second Activity.

The activity_main.xml layout looks the same as it did in the previous task—although you have added two new TextView elements to the layout. Because you set these elements to invisible, they do not appear on the screen.

The following is the XML code for the activity_main.xml file:

```
<android.support.constraint.ConstraintLayout
    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"
    tools:context="com.example.android.twoactivities.MainActivity">

    <TextView
        android:id="@+id/text_header_reply"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header_reply"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        android:visibility="invisible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message_reply"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        android:visibility="invisible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header_reply" />

    <Button
        android:id="@+id/button2"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/button_main"
        android:onClick="launchSecondActivity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <EditText
        android:id="@+id/editText_main"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:ems="10"
        android:hint="@string/editText_main"
        android:inputType="textLongMessage"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button2"
        app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

4.4 Get the reply from the Intent extra and display it

When you use an explicit Intent to start another Activity, you may not expect to get any data back—you're just activating that Activity. In that case, you use `startActivity()` to start the new Activity, as you did earlier in this practical. If you want to get data back from the activated Activity, however, you need to start it with `startActivityForResult()`.

In this task you modify the app to start `SecondActivity` expecting a result, to extract that return data from the Intent, and to display that data in the `TextView` elements you created in the last task.

1. Open **MainActivity**.
2. Add a public constant at the top of the class to define the key for a particular type of response you're interested in:

```
public static final int TEXT_REQUEST = 1;
```

3. Add two private variables to hold the reply header and reply TextView elements:

```
private TextView mReplyHeadTextView;  
private TextView mReplyTextView;
```

4. In the onCreate() method, use findViewById() to get references from the layout to the reply header and reply TextView elements. Assign those view instances to the private variables:

```
mReplyHeadTextView = findViewById(R.id.text_header_reply);  
mReplyTextView = findViewById(R.id.text_message_reply);
```

The full onCreate() method should now look like this:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mMessageEditText = findViewById(R.id.editText_main);  
    mReplyHeadTextView = findViewById(R.id.text_header_reply);  
    mReplyTextView = findViewById(R.id.text_message_reply);  
}
```

5. In the launchSecondActivity() method, change the call to startActivityForResult() to be startActivityForResult(), and include the TEXT_REQUEST key as an argument:

```
startActivityForResult(intent, TEXT_REQUEST);
```

6. Override the onActivityResult() callback method with this signature:

```
@Override  
public void onActivityResult(int requestCode,  
                           int resultCode, Intent data) {  
}
```

The three arguments to onActivityResult() contain all the information you need to handle the return data: the requestCode you set when you launched the Activity with startActivityForResult(), the resultCode set in the launched Activity (usually one of RESULT_OK or RESULT_CANCELED), and the Intent data that contains the data returned from the launch Activity.

7. Inside onActivityResult(), call super.onActivityResult():

```
super.onActivityResult(requestCode, resultCode, data);
```

8. Add code to test for TEXT_REQUEST to make sure you process the right Intent result, in case there are several. Also test for RESULT_OK, to make sure that the request was successful:

```
if (requestCode == TEXT_REQUEST) {  
    if (resultCode == RESULT_OK) {  
    }  
}
```

The [Activity](#) class defines the result codes. The code can be RESULT_OK (the request was successful), RESULT_CANCELED (the user cancelled the operation), or

RESULT_FIRST_USER(for defining your own result codes).

-
9. Inside the inner if block, get the Intent extra from the response Intent(data). Here the key for the extra is the EXTRA_REPLY constant from SecondActivity:

```
String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);
```

10. Set the visibility of the reply header to true:

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

11. Set the reply TextView text to the reply, and set its visibility to true:

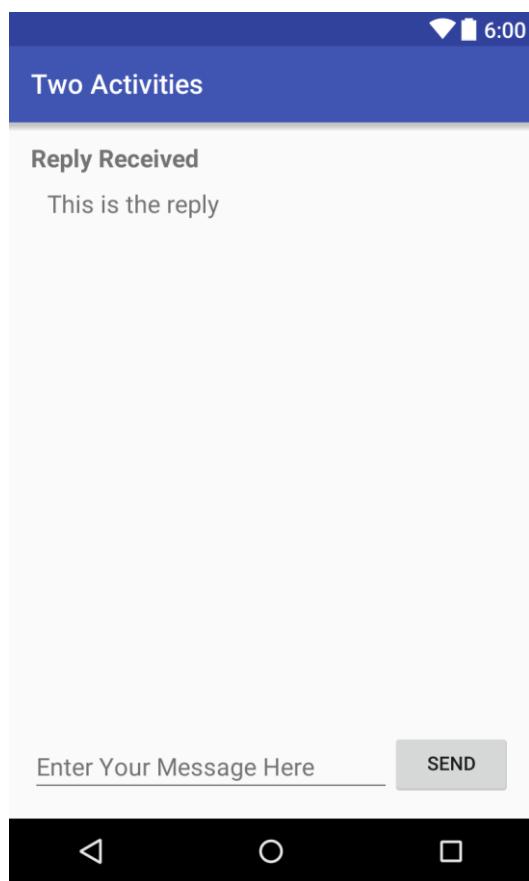
```
mReplyTextView.setText(reply);
mReplyTextView.setVisibility(View.VISIBLE);
```

The full onActivityResult() method should now look like this:

```
@Override
public void onActivityResult(int requestCode,
                             int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == TEXT_REQUEST) {
        if (resultCode == RESULT_OK) {
            String reply =
                data.getStringExtra(SecondActivity.EXTRA_REPLY);
            mReplyHeadTextView.setVisibility(View.VISIBLE);
            mReplyTextView.setText(reply);
            mReplyTextView.setVisibility(View.VISIBLE);
        }
    }
}
```

12. Run the app.

Now, when you send a message to the second Activity and get a reply, the main Activity updates to display the reply.



Solution code

Android Studio project: [TwoActivities](#)

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: Create an app with three Button elements labeled **Text One**, **Text Two**, and **Text Three**. When any of these Button elements are clicked, launch a second Activity. That second Activity should contain a ScrollView that displays one of three text passages (you can include your choice of passages). Use an Intent to launch the second Activity with extras to indicate which of the three passages to display.

Summary

Overview:

- An Activity is an app component that provides a single screen focused on a single user task.
- Each Activity has its own user interface layout file.
- You can assign your Activity implementations a parent/child relationship to enable Up navigation within your app.
- A View can be made visible or invisible with the android:visibility attribute.

To implement an Activity:

- Choose **File > New > Activity** to start from a template and do the following steps automatically.
- If not starting from a template, create an ActivityJava class, implement a basic UI for the Activity in an associated XML layout file, and declare the new Activity in `AndroidManifest.xml`.

Intent:

- An Intent lets you request an action from another component in your app, for example, to start one Activity from another. An Intent can be explicit or implicit.
- With an explicit Intent you indicate the specific target component to receive the data.
- With an implicit Intent you specify the functionality you want but not the target component.
- An Intent can include data on which to perform an action (as a URI) or additional information as Intent *extras*.
- Intent *extras* are key/value pairs in a Bundle that are sent along with the Intent.

Related concept

The related concept documentation is in [2.1: Activities and intents](#).

Learn more

Android Studio documentation:

- [Meet Android Studio](#)

Android developer documentation:

- [Application Fundamentals](#)

-
- [Activities](#)
 - [Intents and Intent Filters](#)

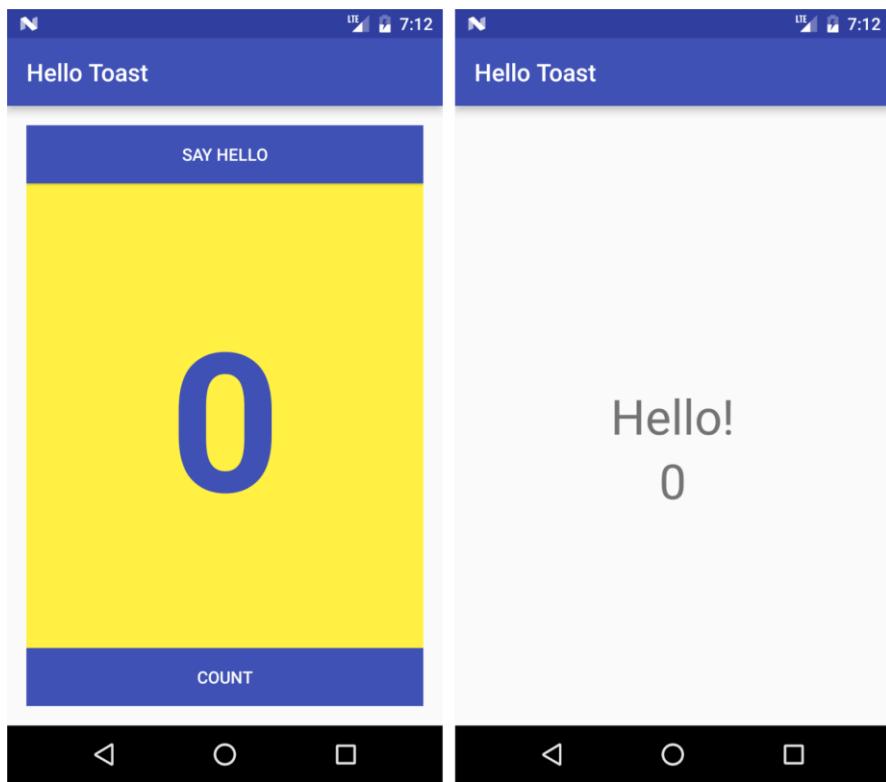
-
- [Designing Back and Up navigation](#)
 - [Activity](#)
 - [Intent](#)
 - [ScrollView](#)
 - [View](#)
 - [Button](#)
 - [TextView](#)
 - [String resources](#)

Homework

Build and run an app

Open the [HelloToast](#) app that you created in a previous practical codelab.

1. Modify the **Toast** button so that it launches a new Activityto display the word "Hello!" and the current count, as shown below.
2. Change the text on the Toastbutton to **Say Hello**.



Answer these questions

Question 1

What changes are made when you add a second Activity to your app by choosing **File > New > Activity** and an Activitytemplate? Choose one:

- The second Activity is added as a Java class. You still need to add the XML layout file.
- The second Activity XML layout file is created and a Java class added. You still need to define the class signature.
- The second Activity is added as a Java class, the XML layout file is created, and the `AndroidManifest.xml` file is changed to declare a second Activity.

-
- The second ActivityXML layout file is created, and the AndroidManifest.xmlfile is changed to declare a second Activity.

Question 2

What happens if you remove the android:parentActivityName and the <meta-data>elements from the second Activity declaration in the AndroidManifest.xml file? Choose one:

- The second Activity no longer appears when you try to start it with an explicit Intent.
- The second ActivityXML layout file is deleted.
- The Back button no longer works in the second Activityto send the user back to the main Activity.
- The Up button in the app bar no longer appears in the second Activityto send the user back to the parent Activity.

Question 3

Which constructor method do you use to create a new explicit Intent? Choose one:

- new Intent()
- new Intent(Context context, Class<?> class)
- new Intent(String action, Uri uri)
- new Intent(String action)

Question 4

In the HelloToast app homework, how do you add the current value of the count to the Intent? Choose one:

- As the Intentdata
- As the IntentTEXT_REQUEST
- As an Intentaction

-
- As an Intentextra

Question 5

In the HelloToast app homework, how do you display the current count in the second "Hello" Activity? Choose one:

- Get the Intent that the Activity was launched with.
- Get the current count value out of the Intent.
- Update the TextView for the count.
- All of the above.

Submit your app for grading

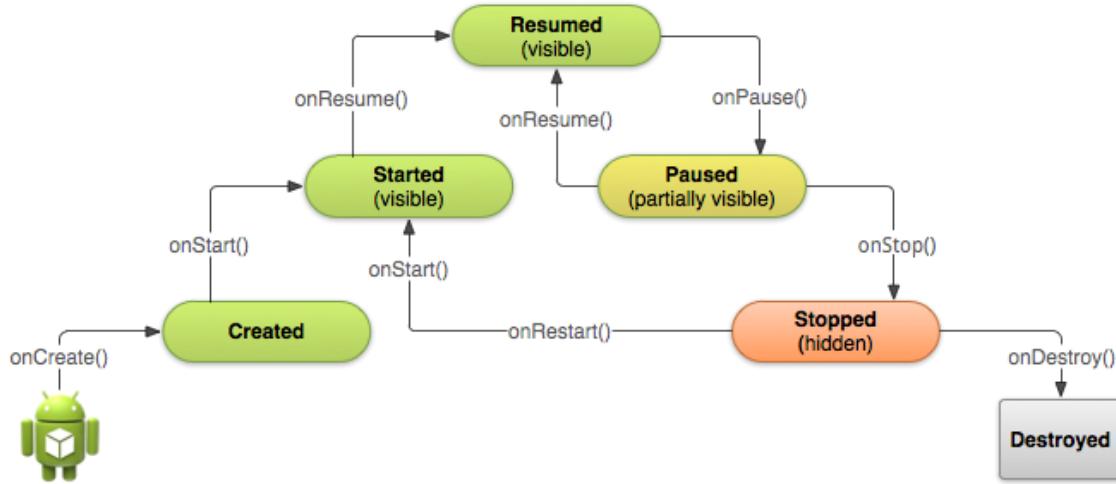
Guidance for graders

Check that the app has the following features:

- It displays the **Say Hello** button instead of the **Toast** button.
- The second Activity starts when the **Say Hello** button is pressed, and it displays the message "Hello!" and the current count from MainActivity.
- The second Activity Java and XML layout files have been added to the project.
- The XML layout file for the second Activity contains two TextView elements, one with the string "Hello!" and the second with the count.
- It includes an implementation of a click handler method for the **Say Hello** button (in MainActivity).
- It includes an implementation of the onCreate() method for the second Activity and updates the count TextView with the count from MainActivity.

Lesson 2.2: Activity lifecycle and state introduction

In this practical you learn more about the *activity lifecycle*. The lifecycle is the set of states an activity can be in during its entire lifetime, from when it's created to when it's destroyed and the system reclaims its resources. As a user navigates between activities in your app (as well as into and out of your app), activities transition between different states in their lifecycles.



Each stage in an activity's lifecycle has a corresponding callback method: `onCreate()`, `onStart()`, `onPause()`, and so on. When an activity changes state, the associated callback method is invoked. You've already seen one of these methods: `onCreate()`. By overriding any of the lifecycle callback methods in your Activity classes, you can change the activity's default behavior in response to user or system actions.

The activity state can also change in response to device-configuration changes, for example when the user rotates the device from portrait to landscape. When these configuration changes happen, the activity is destroyed and recreated in its default state, and the user might lose information that they've entered in the activity. To avoid confusing your users, it's important that you develop your app to prevent unexpected data loss. Later in this practical you experiment with configuration changes and learn how to preserve an activity's state in response to device configuration changes and other activity lifecycle events.

In this practical you add logging statements to the TwoActivities app and observe activity lifecycle changes as you use the app. You then begin working with these changes and exploring how to handle user input under these conditions.

What you should already know

You should be able to:

- Create and run an app project in Android Studio.
- Add log statements to your app and viewing those logs in the **Logcat** pane.
- Understand and work with an Activity and an Intent, and be comfortable interacting with them.

What you'll learn

- How the Activity lifecycle works.
- When an Activity starts, pauses, stops, and is destroyed.
- About the lifecycle callback methods associated with Activity changes.
- The effect of actions (such as configuration changes) that can result in Activity lifecycle events.
- How to retain Activity state across lifecycle events.

What you'll do

- Add code to the TwoActivities app from the previous practical to implement the various Activity lifecycle callbacks to include logging statements.
- Observe the state changes as your app runs and as you interact with each Activity in your app.
- Modify your app to retain the instance state of an Activity that is unexpectedly recreated in response to user behavior or configuration change on the device.

App overview

In this practical you add to the [TwoActivities](#) app. The app looks and behaves roughly the same as it did in the last codelab. It contains two Activity implementations and gives the user the ability to

send between them. The changes you make to the app in this practical will not affect its visible user behavior.

Task 1: Add lifecycle callbacks to TwoActivities

In this task you will implement all of the Activity lifecycle callback methods to print messages to logcat when those methods are invoked. These log messages will allow you to see when the Activity lifecycle changes state, and how those lifecycle state changes affect your app as it runs.

1.1 (Optional) Copy the TwoActivities project

For the tasks in this practical, you will modify the existing [TwoActivities](#) project you built in the last practical. If you'd prefer to keep the previous TwoActivities project intact, follow the steps in [Appendix: Utilities](#) to make a copy of the project.

1.2 Implement callbacks into MainActivity

1. Open the TwoActivities project in Android Studio, and open **MainActivity** in the **Project > Android** pane.
2. In the `onCreate()` method, add the following log statements:

```
Log.d(LOG_TAG, "-----");
Log.d(LOG_TAG, "onCreate");
```

3. Add an override for the `onStart()` callback, with a statement to the log for that event:

```
@Override
public void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "onStart");
```

```
}
```

For a shortcut, select **Code > Override Methods** in Android Studio. A dialog appears with all of the possible methods you can override in your class. Choosing one or more callback methods from the list inserts a complete template for those methods, including the required call to the superclass.

4. Use the `onStart()` method as a template to implement the `onPause()`, `onRestart()`, `onResume()`, `onStop()`, and `onDestroy()` lifecycle callbacks.

All the callback methods have the same signatures (except for the name). If you **Copy** and **Paste** `onStart()` to create these other callback methods, don't forget to update the contents to call the right method in the superclass, and to log the correct method.

1. Run your app.
2. Click the **Logcat** tab at the bottom of Android Studio to show the **Logcat** pane. You should see three log messages showing the three lifecycle states the Activity has transitioned through as it started:

```
D/MainActivity: -----  
D/MainActivity: onCreate  
D/MainActivity: onStart  
D/MainActivity: onResume
```

1.3 Implement lifecycle callbacks in SecondActivity

Now that you've implemented the lifecycle callback methods for `MainActivity`, do the same for `SecondActivity`.

1. Open `SecondActivity`.
2. At the top of the class, add a constant for the `LOG_TAG` variable:

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

3. Add the lifecycle callbacks and log statements to the second Activity. (You can **Copy** and **Paste** the callback methods from MainActivity.)
4. Add a log statement to the returnReply() method just before the finish() method:

```
Log.d(LOG_TAG, "End SecondActivity");
```

1.4 Observe the log as the app runs

1. Run your app.
2. Click the **Logcat** tab at the bottom of Android Studio to show the **Logcat** pane.
3. Enter **Activity** in the search box.

The Android logcat can be very long and cluttered. Because the LOG_TAG variable in each class contains either the words MainActivity or SecondActivity, this keyword lets you filter the log for only the things you're interested in.



Experiment using your app and note that the lifecycle events that occur in response to different actions. In particular, try these things:

- Use the app normally (send a message, reply with another message).
- Use the Back button to go back from the second Activity to the main Activity.
- Use the Up arrow in the app bar to go back from the second Activity to the main Activity.
- Rotate the device on both the main and second Activity at different times in your app and observe what happens in the log and on the screen.
- Press the overview button (the square button to the right of Home) and close the app (tap the X).
- Return to the home screen and restart your app.

TIP: If you're running your app in an emulator, you can simulate rotation with Control+F11 or Control+Function+F11.

Task 1 solution code

The following code snippets show the solution code for the first task.

MainActivity

The following code snippets show the added code in MainActivity, but not the entire class.

The onCreate() method:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // Log the start of the onCreate() method.  
    Log.d(LOG_TAG, "-----");  
    Log.d(LOG_TAG, "onCreate");
```

```
// Initialize all the view variables.  
mMessageEditText = findViewById(R.id.editText_main);  
mReplyHeadTextView = findViewById(R.id.text_header_reply);  
mReplyTextView = findViewById(R.id.text_message_reply);  
}
```

The other lifecycle methods:

```
@Override  
protected void onStart() {  
    super.onStart();  
    Log.d(LOG_TAG, "onStart");  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    Log.d(LOG_TAG, "onPause");  
}  
  
@Override  
protected void onRestart() {  
    super.onRestart();  
    Log.d(LOG_TAG, "onRestart");  
}  
  
@Override  
protected void onResume() {  
    super.onResume();  
    Log.d(LOG_TAG, "onResume");  
}  
  
@Override  
protected void onStop() {  
    super.onStop();  
    Log.d(LOG_TAG, "onStop");  
}  
  
@Override  
protected void onDestroy() {
```

```
super.onDestroy();
Log.d(LOG_TAG, "onDestroy");
}
```

SecondActivity

The following code snippets show the added code in SecondActivity, but not the entire class.

At the top of the SecondActivity class:

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

The returnReply() method:

```
public void returnReply(View view) {
    String reply = mReply.getText().toString();
    Intent replyIntent = new Intent();
    replyIntent.putExtra(EXTRA_REPLY, reply);
    setResult(RESULT_OK, replyIntent);
    Log.d(LOG_TAG, "End SecondActivity");
    finish();
}
```

The other lifecycle methods:

Same as for MainActivity, above.

Task 2: Save and restore the Activity instance state

Depending on system resources and user behavior, each Activity in your app may be destroyed and reconstructed far more frequently than you might think.

You may have noticed this behavior in the last section when you rotated the device or emulator. Rotating the device is one example of a device *configuration change*. Although rotation is the most common one, all configuration changes result in the current Activity being destroyed and recreated as if it were new. If you don't account for this behavior in your code, when a configuration change occurs, your Activity layout may revert to its default appearance and initial values, and your users may lose their place, their data, or the state of their progress in your app.

The state of each Activity is stored as a set of key/value pairs in a [Bundle](#) object called the *Activity instance state*. The system saves default state information to instance state Bundle just before the Activity is stopped, and passes that Bundle to the new Activity instance to restore.

To keep from losing data in an Activity when it is unexpectedly destroyed and recreated, you need to implement the `onSaveInstanceState()` method. The system calls this method on your Activity (between `onPause()` and `onStop()`) when there is a possibility the Activity may be destroyed and recreated.

The data you save in the instance state is specific to only this instance of this specific Activity during the current app session. When you stop and restart a new app session, the Activity instance state is lost and the Activity reverts to its default appearance. If you need to save user data between app sessions, use shared preferences or a database. You learn about both of these in a later practical.

2.1 Save the Activity instance state with `onSaveInstanceState()`

You may have noticed that rotating the device does not affect the state of the second Activity at all. This is because the second Activity layout and state are generated from the layout and the

Intent that activated it. Even if the Activity is recreated, the Intent is still there and the data in that Intent is still used each time the `onCreate()` method in the second Activity is called.

In addition, you may notice that in each Activity, any text you typed into message or reply EditTextelements is retained even when the device is rotated. This is because the state information of some of the Viewelements in your layout are automatically saved across configuration changes, and the current value of an EditTextis one of those cases.

So the only Activity state you're interested in are the TextViewelements for the reply header and the reply text in the main Activity. Both TextViewelements are invisible by default; they only appear once you send a message back to the main Activityfrom the second Activity.

In this task you add code to preserve the instance state of these two TextViewelements using onSaveInstanceState().

1. Open **MainActivity**.
2. Add this skeleton implementation of onSaveInstanceState() to the Activity, or use **Code > Override Methods** to insert a skeleton override.

```
@Override  
public void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
}
```

3. Check to see if the header is currently visible, and if so put that visibility state into the state Bundlewith the putBoolean()method and the key "reply_visible".

```
if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {  
    outState.putBoolean("reply_visible", true);  
}
```

Remember that the reply header and text are marked invisible until there is a reply from the second Activity. If the header is visible, then there is reply data that needs to be saved. Note that we're only interested in that visibility state — the actual text of the header doesn't need to be saved, because that text never changes.

4. Inside that same check, add the reply text into the Bundle.

```
outState.putString("reply_text",mReplyTextView.getText().toString());
```

If the header is visible you can assume that the reply message itself is also visible. You don't need to test for or save the current visibility state of the reply message. Only the actual text of the message goes into the state Bundlewith the key "reply_text".

You save the state of only those Viewelements that might change after the Activityis created. The other Viewelements in your app (the EditText, the Button) can be recreated from the default layout at any time.

Note that the system will save the state of some Viewelements, such as the contents of the EditText.

2.2 Restore the Activity instance state in onCreate()

Once you've saved the Activityinstance state, you also need to restore it when the Activityis recreated. You can do this either in onCreate(), or by implementing the onRestoreInstanceState() callback, which is called after onStart()after the Activityis created.

Most of the time the better place to restore the Activitystate is in onCreate(), to ensure that the UI, including the state, is available as soon as possible. It is sometimes convenient to do it in onRestoreInstanceState()after all of the initialization has been done, or to allow subclasses to decide whether to use your default implementation.

1. In the onCreate()method, after the Viewvariables are initialized with findViewById(), add a test to make sure that savedInstanceStateis not null.

```
// Initialize all the view variables.  
mMessageEditText = findViewById(R.id.editText_main);  
mReplyHeadTextView = findViewById(R.id.text_header_reply);  
mReplyTextView = findViewById(R.id.text_message_reply);  
  
// Restore the state.  
if (savedInstanceState != null) {  
}
```

When your Activity is created, the system passes the state Bundle to `onCreate()` as its only argument. The first time `onCreate()` is called and your app starts, the Bundle is null—there's no existing state the first time your app starts. Subsequent calls to `onCreate()` have a bundle populated with the data you stored in `onSaveInstanceState()`.

2. Inside that check, get the current visibility (true or false) out of the Bundle with the key "reply_visible".

```
if (savedInstanceState != null) {  
    boolean isVisible =  
        savedInstanceState.getBoolean("reply_visible");  
}
```

3. Add a test below that previous line for the isVisible variable.

```
if (isVisible) {  
}
```

If there's a `reply_visible` key in the state Bundle (and `isVisible` is therefore true), you will need to restore the state.

4. Inside the `isVisible` test, make the header visible.

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

5. Get the text reply message from the Bundle with the key "reply_text", and set the reply TextView to show that string.

```
mReplyTextView.setText(savedInstanceState.getString("reply_text"));
```

6. Make the reply TextViewvisible as well:

```
mReplyTextView.setVisibility(View.VISIBLE);
```

7. Run the app. Try rotating the device or the emulator to ensure that the reply message (if there is one) remains on the screen after the Activityis recreated.

Task 2 solution code

The following code snippets show the solution code for this task.

MainActivity

The following code snippets show the added code in MainActivity, but not the entire class.

The onSaveInstanceState()method:

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // If the heading is visible, message needs to be saved.
    // Otherwise we're still using default layout.
    if (mReplyHeadTextView.getVisibility() == View.VISIBLE) {
        outState.putBoolean("reply_visible", true);
        outState.putString("reply_text",
                           mReplyTextView.getText().toString());
    }
}
```

```
}
```

The onCreate() method:

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

    Log.d(LOG_TAG, "-----");
    Log.d(LOG_TAG, "onCreate");

    // Initialize all the view variables. mMessageEditText =
    findViewById(R.id.editText_main);
    mReplyHeadTextView = findViewById(R.id.text_header_reply); mReplyTextView =
    findViewById(R.id.text_message_reply);

    // Restore the saved state.
    // See onSaveInstanceState() for what gets saved. if
    (savedInstanceState != null) {
        boolean isVisible =
            savedInstanceState.getBoolean("reply_visible");
        // Show both the header and the message views. If isVisible is
        // false or missing from the bundle, use the default layout. if (isVisible) {
            mReplyHeadTextView.setVisibility(View.VISIBLE); mReplyTextView.setText(savedInstanceState
                .getString("reply_text"));
            mReplyTextView.setVisibility(View.VISIBLE);
        }
    }
}
```

The complete project:

Android Studio Project: [TwoActivitiesLifecycle](#)

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: Create a simple shopping-list app with a main activity for the list the user is building, and a second activity for a list of common shopping items.

- The main activity should contain the list to build, which should be made up of ten empty TextView elements.
- An **Add Item** button on the main activity launches a second activity that contains a list of common shopping items (**Cheese**, **Rice**, **Apples**, and so on). Use Button elements to display the items.
- Choosing an item returns the user to the main activity, and updates an empty TextView to include the chosen item.

Use an Intent to pass information from one Activity to another. Make sure that the current state of the shopping list is saved when the user rotates the device.

Summary

- The Activity lifecycle is a set of states an Activity migrates through, beginning when it is first created and ending when the Android system reclaims the resources for that Activity.
- As the user navigates from one Activity to another, and inside and outside of your app, each Activity moves between states in the Activity lifecycle.
- Each state in the Activity lifecycle has a corresponding callback method you can override in your Activity class.
- The lifecycle methods are `onCreate()`, `onStart()`, `onPause()`, `onRestart()`, `onResume()`, `onStop()`, `onDestroy()`.
- Overriding a lifecycle callback method allows you to add behavior that occurs when your Activity transitions into that state.
- You can add skeleton override methods to your classes in Android Studio with **Code**

> **Override.**

-
- Device configuration changes such as rotation results in the Activity being destroyed and recreated as if it were new.
 - A portion of the Activity state is preserved on a configuration change, including the current values of EditTextelements. For all other data, you must explicitly save that data yourself.
 - Save Activity instance state in the onSaveInstanceState() method.
 - Instance state data is stored as simple key/value pairs in a Bundle. Use the Bundle methods to put data into and get data back out of the Bundle.
 - Restore the instance state in onCreate(), which is the preferred way, or onRestoreInstanceState().

Related concept

The related concept documentation is in [2.2: Activity lifecycle and state](#).

Learn more

Android Studio documentation:

- [Meet Android Studio](#)

Android developer documentation:

- [Application Fundamentals](#)
- [Activities](#)
- [Understand the Activity Lifecycle](#)
- [Intents and Intent Filters](#)
- [Handle configuration changes](#)
- [Activity](#)
- [Intent](#)

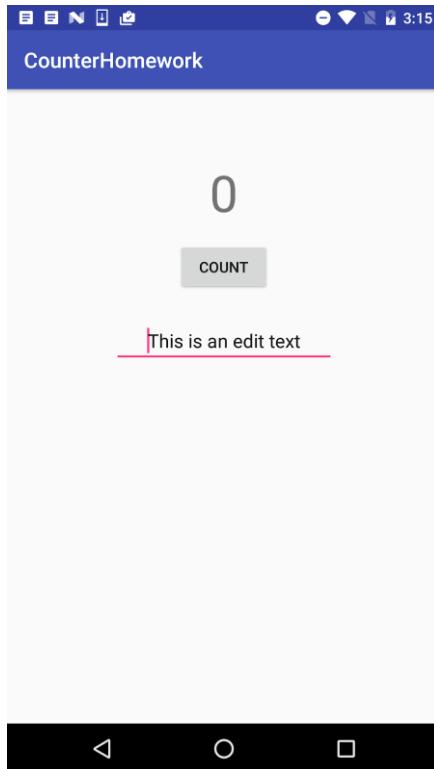
Homework

Build and run an app

1. Create an app with a layout that holds a counter TextView, a Button to increment the counter, and an EditText. See the screenshot below as an example. You don't have to precisely duplicate the layout.

2. Add a click handler for the Button that increments the counter.

-
3. Run the app and increment the counter. Enter some text into the EditText.
 4. Rotate the device. Note that the counter is reset, but the EditText is not.
 5. Implement `onSaveInstanceState()` to save the current state of the app.
 6. Update `onCreate()` to restore the state of the app.
 7. Make sure that when you rotate the device, the app state is preserved.



Answer these questions

Question 1

If you run the homework app before implementing `onSaveInstanceState()`, what happens if you rotate the device? Choose one:

- The EditText no longer contains the text you entered, but the counter is preserved.
- The counter is reset to 0, and the EditText no longer contains the text you entered.
- The counter is reset to 0, but the contents of the EditText is preserved.

-
- The counter and the contents of the EditText are preserved.

Question 2

What Activitylifecycle methods are called when a device-configuration change (such as rotation) occurs? Choose one:

- Android immediately shuts down your Activity by calling onStop(). Your code must restart the Activity.
- Android shuts down your Activityby calling onPause(), onStop(), and onDestroy(). Your code must restart the Activity.
- Android shuts down your Activity by calling onPause(), onStop(), and onDestroy(), and then starts it over again, calling onCreate(), onStart(), and onResume().
- Android immediately calls onResume().

Question 3

When in the Activitylifecycle is onSaveInstanceState()called? Choose one:

- onSaveInstanceState() is called before the onStop() method.
- onSaveInstanceState() is called before the onResume() method.
- onSaveInstanceState() is called before the onCreate() method.
- onSaveInstanceState() is called before the onDestroy() method.

Question 4

Which Activitylifecycle methods are best to use for saving data before the Activityis finished or destroyed? Choose one:

- onPause()or onStop()
- onResume()or onCreate()
- onDestroy()
- onStart()or onRestart()

Submit your app for grading

Guidance for graders

Check that the app has the following features:

- It displays a counter, a Button to increment that counter, and an EditText.
- Clicking the Button increments the counter by 1.
- When the device is rotated, both the counter and EditText states are retained.
- The implementation of MainActivity.java uses the onSaveInstanceState() method to store the counter value.
- The implementation of onCreate() tests for the existence of the outState Bundle. If that Bundle exists, the counter value is restored and saved to the TextView.

Lesson 2.3: Implicit intents

Introduction

In a previous section you learned about explicit intents. In an explicit intent, you carry out an activity in your app, or in a different app, by sending an intent with the fully qualified class name of the activity. In this section you learn more about *implicit* intents and how to use them to carry out activities.

With an implicit intent, you initiate an activity without knowing which app or activity will handle the task. For example, if you want your app to take a photo, send email, or display a location on a map, you typically don't care which app or activity performs the task.

Conversely, your activity can declare one or more intent filters in the AndroidManifest.xml file to advertise that the activity can accept implicit intents, and to define the types of intents that the activity will accept.

To match your request with an app installed on the device, the Android system matches your implicit intent with an activity whose intent filters indicate that they can perform the action. If multiple apps match, the user is presented with an app chooser that lets them select which app they want to use to handle the intent.

In this practical you build an app that sends an implicit intent to perform each of the following tasks:

- Open a URL in a web browser.

-
- Open a location on a map.
 - Share text.

Sharing—sending a piece of information to other people through email or social media—is a popular feature in many apps. For the sharing action you use the `ShareCompat.IntentBuilder` class, which makes it easy to build an implicit intent for sharing data.

Finally, you create a simple intent-receiver that accepts an implicit intent for a specific action.

What you should already know

You should be able to:

- Use the layout editor to modify a layout.
- Edit the XML code of a layout.
- Add a Button and a click handler.
- Create and use an Activity.
- Create and send an Intent between one Activity and another.

What you'll learn

- How to create an implicit Intent, and use its actions and categories.
- How to use the `ShareCompat.IntentBuilderHelper` class to create an implicit Intent for sharing data.
- How to advertise that your app can accept an implicit Intent by declaring Intent filters in the `AndroidManifest.xml` file.

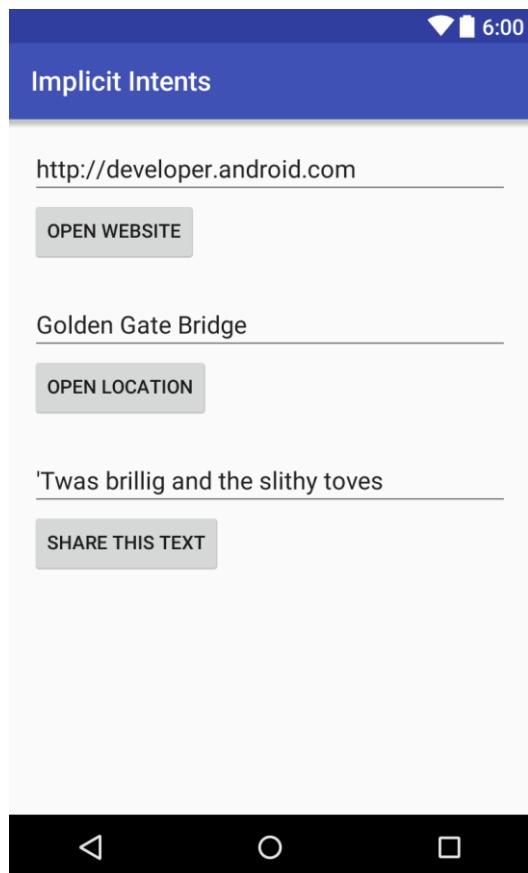
What you'll do

- Create a new app to experiment with implicit Intent.
- Implement an implicit Intent that opens a web page, and another that opens a location on a map.
- Implement an action to share a snippet of text.

-
- Create a new app that can accept an implicit Intent for opening a web page.

App overview

In this section you create a new app with one Activity and three options for actions: open a web site, open a location on a map, and share a snippet of text. All of the text fields are editable (`EditText`), but contain default values.



Task 1: Create the project and layout

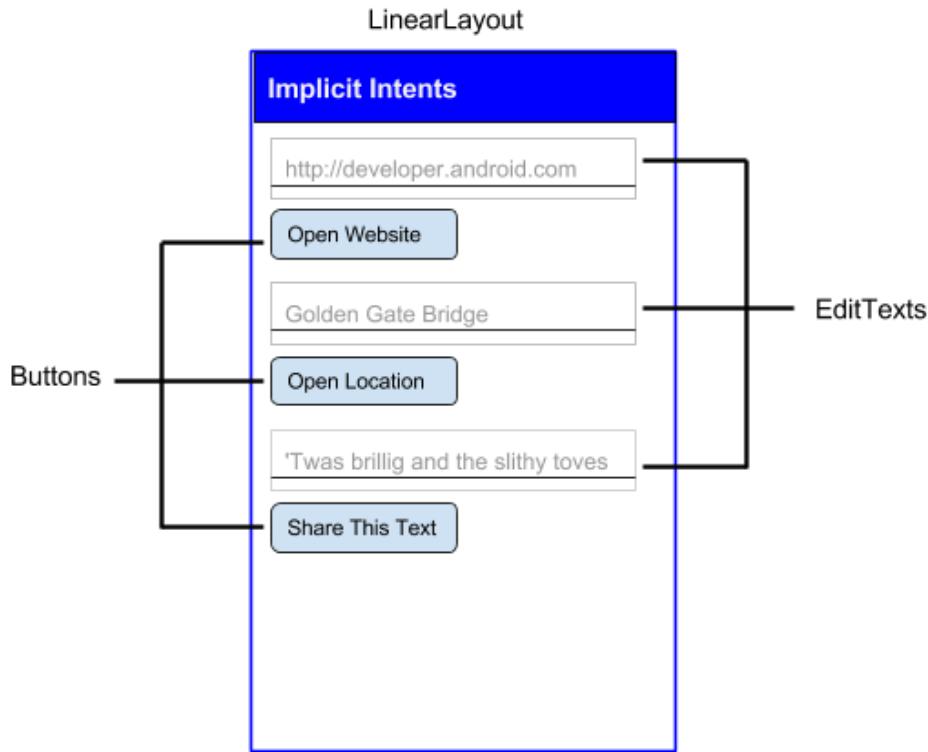
For this exercise, you create a new project and app called Implicit Intents, with a new layout.

1.1 Create the project

1. Start Android Studio and create a new Android Studio project. Name your app **Implicit Intents**.
2. Choose **Empty Activity** for the project template. Click **Next**.
3. Accept the default Activityname (MainActivity). Make sure the **Generate Layout file** box is checked. Click **Finish**.

1.2 Create the layout

In this task, create the layout for the app. Use a `LinearLayout`, three `Button` elements, and three `EditText` elements, like this:



1. Open **app > res > values > strings.xml** in the **Project > Android** pane, and add the following string resources:

```

<string name="edittext_uri">http://developer.android.com</string>
<string name="button_uri">Open Website</string>

<string name="edittext_loc">Golden Gate Bridge</string>
<string name="button_loc">Open Location</string>

<string name="edittext_share">\'Twas brillig and the slithy toves</string>
<string name="button_share">Share This Text</string>

```

-
2. Open **res > layout > activity_main.xml** in the **Project > Android** pane. Click the **Text** tab to switch to XML code.
 3. Change `android.support.constraint.ConstraintLayout` to `LinearLayout`, as you learned in a previous practical.
 4. Add the `android:orientation` attribute with the value "vertical". Add the `android:padding` attribute with the value "16dp".

```
<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"  
    android:padding="16dp"  
    tools:context="com.example.android.implicitintents.MainActivity">
```

5. Remove the `TextView` that displays "Hello World".
6. Add a set of UI elements to the layout for the **Open Website** button. You need an `EditText` element and a `Button` element. Use these attribute values:

EditText attribute	Value
<code>android:id</code>	"@+id/website_edittext"
<code>android:layout_width</code>	"match_parent"
<code>android:layout_height</code>	"wrap_content"
<code>android:text</code>	"@string/edittext_uri"
Button attribute	Value
<code>android:id</code>	"@+id/open_website_button"
<code>android:layout_width</code>	"wrap_content"
<code>android:layout_height</code>	"wrap_content"
<code>android:layout_marginBottom</code>	"24dp"
<code>android:text</code>	"@string/button_uri"

android:onClick	"openWebsite"
-----------------	---------------

The value for the android:onClickattribute will remain underlined in red until you define the callback method in a subsequent task.

7. Add a set of UI elements (EditTextand Button) to the layout for the **Open Location** button. Use the same attributes as in the previous step, but modify them as shown below. (You can copy the values from the **Open Website** button and modify them.)

EditText attribute	Value
android:id	"@+id/location_edittext"
android:text	"@string/edittext_loc"
Button attribute	Value
android:id	"@+id/open_location_button"
android:text	"@string/button_loc"
android:onClick	"openLocation"

The value for the android:onClickattribute will remain underlined in red until you define the callback method in a subsequent task.

8. Add a set of UI elements (EditTextand Button) to the layout for the **Share This** button. Use the attributes shown below. (You can copy the values from the **Open Website** button and modify them.)

EditText attribute	Value
android:id	"@+id/share_edittext"
android:text	"@string/edittext_share"
Button attribute	Value
android:id	"@+id/share_text_button"
android:text	"@string/button_share"

android:onClick	"shareText"
-----------------	-------------

Depending on your version of Android Studio, your activity_main.xml code should look something like the following. The values for the android:onClick attributes will remain underlined in red until you define the callback methods in a subsequent task.

```
<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" android:padding="16dp"
    tools:context="com.example.android.implicitintents.MainActivity">

    <EditText
        android:id="@+id/website_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_uri"/>

    <Button
        android:id="@+id/open_website_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:text="@string/button_uri"
        android:onClick="openWebsite"/>

    <EditText
        android:id="@+id/location_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_uri"/>

    <Button
```

```
    android:id="@+id/open_location_button"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp"
        android:text="@string/button_loc"
        android:onClick="openLocation"/>

<EditText
    android:id="@+id/share_edittext"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/edittext_share"/>

<Button
    android:id="@+id/share_text_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="24dp"
    android:text="@string/button_share"
    android:onClick="shareText"/>

</LinearLayout>
```

Task 2: Implement the Open Website button

In this task you implement the on-click handler method for the first button in the layout, **Open Website**. This action uses an implicit Intent to send the given URI to an Activity that can handle that implicit Intent(such as a web browser).

2.1 Define openWebsite()

1. Click "openWebsite" in the activity_main.xml XML code.
2. Press Alt+Enter(Option+Enter on a Mac) and select **Create 'openWebsite(View)' in 'MainActivity'**.

The MainActivity file opens, and Android Studio generates a skeleton method for the openWebsite() handler.

```
public void openWebsite(View view) {  
}
```

3. In MainActivity, add a private variable at the top of the class to hold the EditText object for the web site URI.

```
private EditText mWebsiteEditText;
```

4. In the onCreate() method for MainActivity, use findViewById() to get a reference to the EditText instance and assign it to that private variable:

```
mWebsiteEditText = findViewById(R.id.website_edittext);
```

2.2 Add code to openWebsite()

1. Add a statement to the new openWebsite() method that gets the string value of the EditText:

```
String url = mWebsiteEditText.getText().toString();
```

-
2. Encode and parse that string into a Uri object:

```
Uri webpage = Uri.parse(url);
```

3. Create a new Intentwith Intent.ACTION_VIEWas the action and the URI as the data:

```
Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
```

This Intentconstructor is different from the one you used to create an explicit Intent. In the previous constructor, you specified the current context and a specific component (Activityclass) to send the Intent. In this constructor you specify an action and the data for that action. Actions are defined by the Intentclass and can include ACTION_VIEW(to view the given data), ACTION_EDIT(to edit the given data), or ACTION_DIAL(to dial a phone number). In this case the action is ACTION_VIEWbecause you want to display the web page specified by the URI in the webpagevariable.

4. Use the resolveActivity() method and the Android package manager to find an Activity that can handle your implicit Intent. Make sure that the request resolved successfully.

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
}
```

This request that matches your Intent action and data with the Intent filters for installed apps on the device. You use it to make sure there is at least one Activitythat can handle your requests.

5. Inside the ifstatement, call startActivity()to send the Intent.

```
startActivity(intent);
```

-
6. Add an elseblock to print a Logmessage if the Intentcould not be resolved.

```
} else {
    Log.d("ImplicitIntents", "Can't handle this!");
}
```

The openWebsite() method should now look as follows. (Comments added for clarity.)

```
public void openWebsite(View view) {
    // Get the URL text.
    String url = mWebsiteEditText.getText().toString();

    // Parse the URI and create the intent.
    Uri webpage = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);

    // Find an activity to hand the intent and start that activity.
    if (intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    } else {
        Log.d("ImplicitIntents", "Can't handle this intent!");
    }
}
```

Task 3: Implement the Open Location button

In this task you implement the on-click handler method for the second button in the UI, **Open Location**. This method is almost identical to the openWebsite() method. The difference is the use of a geo URI to indicate a map location. You can use a geo URI with latitude and longitude, or use a query string for a general location. In this example we've used the latter.

3.1 Define openLocation()

1. Click "openLocation" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'openLocation(View)' in MainActivity.**

Android Studio generates a skeleton method in MainActivity for the openLocation() handler.

```
public void openLocation(View view) {  
}
```

3. Add a private variable at the top of MainActivity to hold the EditText object for the location URI.

```
private EditText mLocationEditText;
```

4. In the onCreate() method, use findViewById() to get a reference to the EditText instance and assign it to that private variable:

```
mLocationEditText = findViewById(R.id.location_edittext);
```

3.2 Add code to openLocation()

1. In the new openLocation() method, add a statement to get the string value of the mLocationEditTextEditText.

```
String loc = mLocationEditText.getText().toString();
```

2. Parse that string into a Uri object with a geo search query:

```
Uri addressUri = Uri.parse("geo:0,0?q=" + loc);
```

3. Create a new Intentwith Intent.ACTION_VIEWas the action and loc as the data.

```
Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);
```

4. Resolve the Intentand check to make sure that the Intentresolved successfully. If so, startActivity(), otherwise log an error message.

```
if (intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
} else {  
    Log.d("ImplicitIntents", "Can't handle this intent!");  
}
```

The openLocation() method should now look as follows (comments added for clarity):

```
public void openLocation(View view) {  
    // Get the string indicating a location. Input is not validated; it is  
    // passed to the location handler intact.  
    String loc = mLocationEditText.getText().toString();  
  
    // Parse the location and create the intent.  
    Uri addressUri = Uri.parse("geo:0,0?q=" + loc);  
    Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);  
  
    // Find an activity to handle the intent, and start that activity.  
    if (intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    } else {  
        Log.d("ImplicitIntents", "Can't handle this intent!");  
    }  
}
```

Task 4: Implement the Share This Text button

A share action is an easy way for users to share items in your app with social networks and other apps. Although you could build a share action in your own app using an implicit Intent, Android provides the [ShareCompat.IntentBuilder](#) helper class to make implementing sharing easy. You can use ShareCompat.IntentBuilder to build an Intent and launch a chooser to let the user choose the destination app for sharing.

In this task you implement sharing a bit of text in a text edit, using the ShareCompat.IntentBuilder class.

4.1 Define shareText()

1. Click "shareText" in the activity_main.xml XML code.
2. Press Alt+Enter (Option+Enter on a Mac) and select **Create 'shareText(View)' in MainActivity**.

Android Studio generates a skeleton method in MainActivity for the shareText() handler.

```
public void shareText(View view) {  
}
```

3. Add a private variable at the top of MainActivity to hold the EditText.

```
private EditText mShareTextEditText;
```

4. In onCreate(), use findViewById() to get a reference to the EditText instance and assign it to that private variable:

```
mShareTextEditText = findViewById(R.id.share_edittext);
```

4.2 Add code to shareText()

1. In the new shareText() method, add a statement to get the string value of the mShareTextEditText.

```
String txt = mShareTextEdit.getText().toString();
```

2. Define the mime type of the text to share:

```
String mimeType = "text/plain";
```

3. Call ShareCompat.IntentBuilder with these methods:

```
ShareCompat.IntentBuilder
    .from(this)
    .setType(mimeType)
    .setChooserTitle("Share this text with: ")
    .setText(txt)
    .startChooser();
```

4. Extract the value of .setChooserTitle to a string resource.

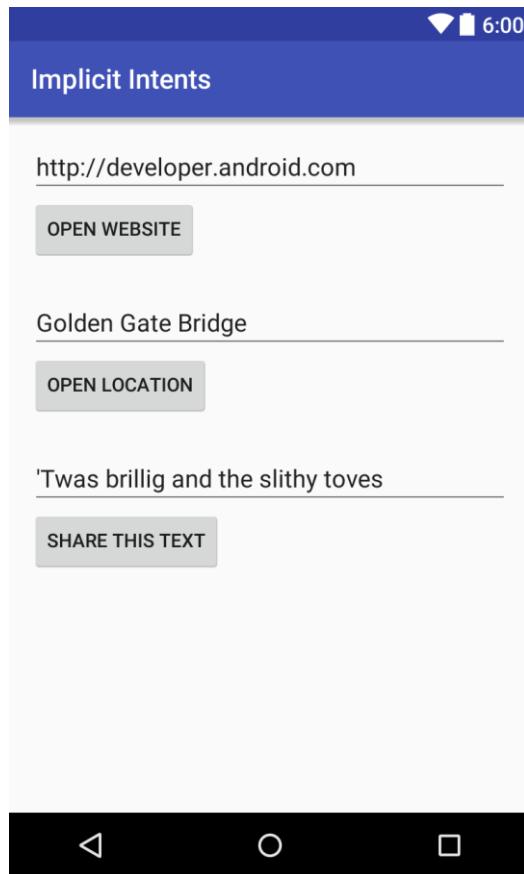
The call to ShareCompat.IntentBuilder uses these methods:

Method	Description
from()	The Activity that launches this share Intent(this).
setType()	The MIME type of the item to be shared.
setChooserTitle()	The title that appears on the system app chooser.
setText()	The actual text to be shared
startChooser()	Show the system app chooser and send the Intent.

This format, with all the builder's setter methods strung together in one statement, is an easy shorthand way to create and launch the Intent. You can add any of the additional methods to this list.

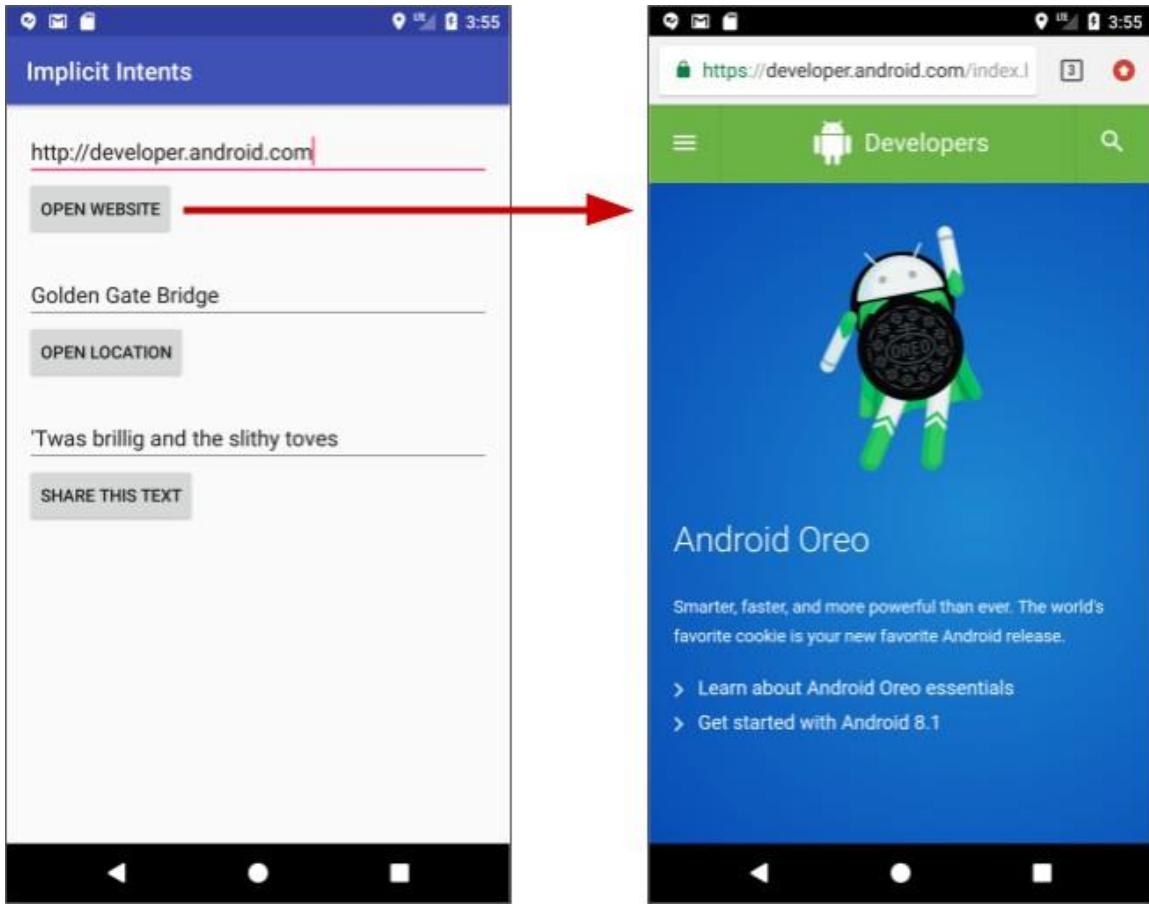
The shareText()method should now look as follows:

```
public void shareText(View view) {  
    String txt = mShareTextEdit.getText().toString();  
    String mimeType = "text/plain";  
    ShareCompat.IntentBuilder  
        .from(this)  
        .setType(mimeType)  
        .setChooserTitle(R.string.share_text_with)  
        .setText(txt)  
        .startChooser();  
}
```

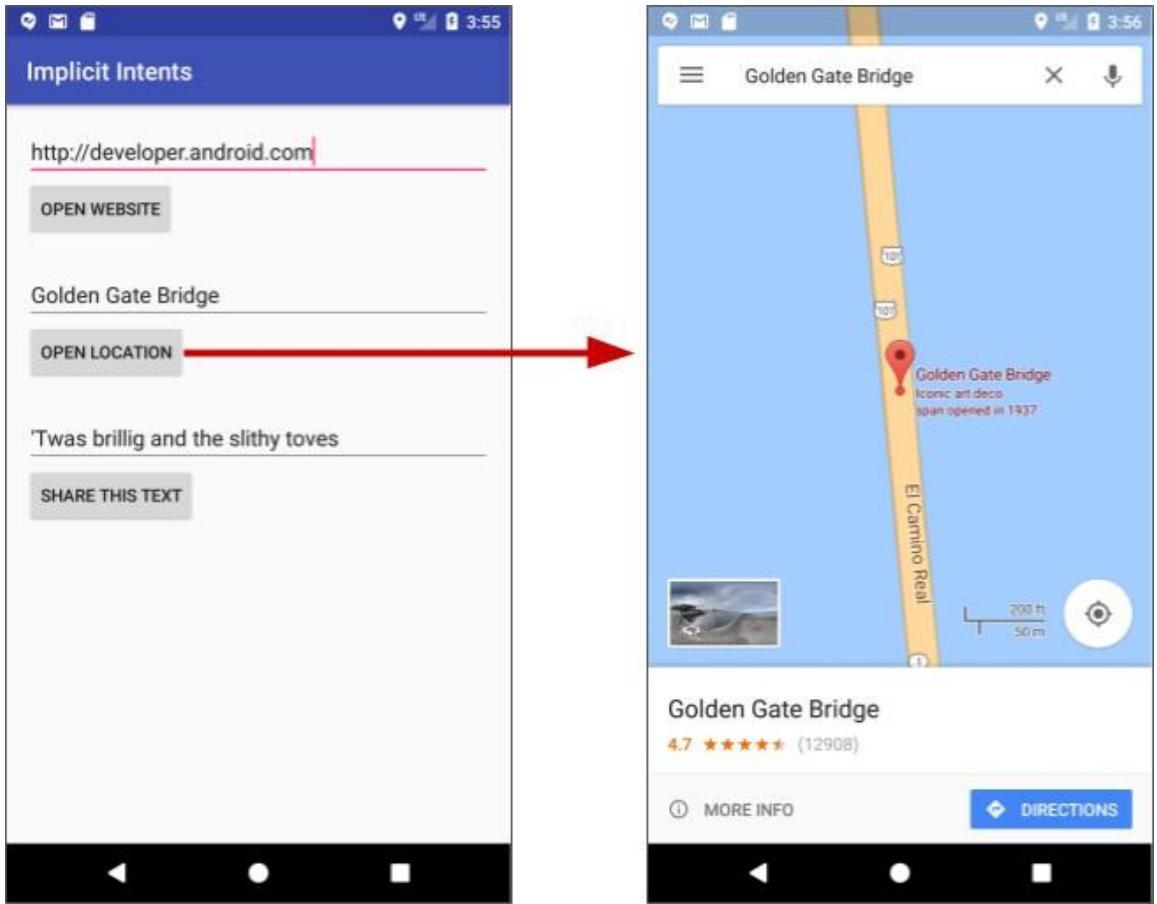


4.3 Run the app

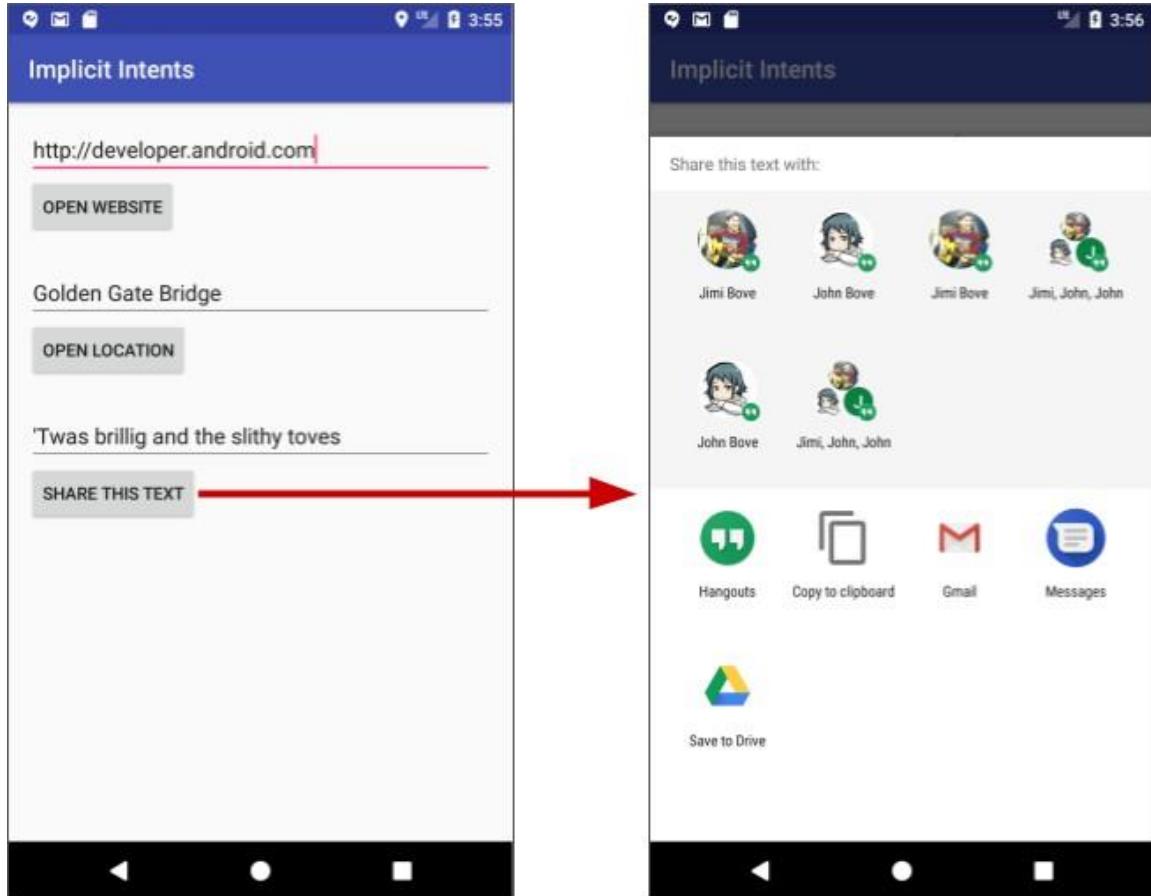
1. Run the app.
2. Click the **Open Website** button to launch a browser with the website URL in the EditText above the Button. The browser and website should appear as shown below.



3. Click the **Open Location** button to launch the map with the location in the EditText above the Button. The map with the location should appear as shown below.



4. Click the **Share This Text** button to launch a dialog with choices for sharing the text.
The dialog with choices should appear as shown below.



Task 4 solution code

Android Studio project: [ImplicitIntents](#)

Task 5: Receive an implicit Intent

So far, you've created an app that uses an implicit Intent in order to launch some other app's Activity. In this task you look at the problem from the other way around: allowing an Activity in your app to respond to an implicit Intent sent from some other app.

An Activity in your app can always be activated from inside or outside your app with an explicit Intent. To allow an Activity to receive an implicit Intent, you define an Intent filter in your app's `AndroidManifest.xml` file to indicate which types of implicit Intent your Activity is interested in handling.

To match your request with a specific app installed on the device, the Android system matches your implicit Intent with an Activity whose Intent filters indicate that they can perform that action. If there are multiple apps installed that match, the user is presented with an app chooser that lets them select which app they want to use to handle that Intent.

When an app on the device sends an implicit Intent, the Android system matches the action and data of that Intent with any available Activity that includes the right Intent filters. When the Intent filters for an Activity match the Intent:

- If there is only one matching Activity, Android lets the Activity handle the Intent itself.
- If there are multiple matches, Android displays an app chooser to allow the user to pick which app they'd prefer to execute that action.

In this task you create a very simple app that receives an implicit Intent to open the URI for a web page. When activated by an implicit Intent, that app displays the requested URI as a string in a `TextView`.

5.1 Create the project and layout

1. Create a new Android Studio project with the app name **Implicit Intents Receiver** and choose **Empty Activity** for the project template.
2. Accept the default Activity name (`MainActivity`). Click **Next**.
3. Make sure the **Generate Layout file** box is checked. Click **Finish**.
4. Open `activity_main.xml`.
5. In the existing ("Hello World") `TextView`, delete the `android:text` attribute. There's no text in this `TextView` by default, but you'll add the URI from the Intent in `onCreate()`.
6. Leave the `layout_constraint` attributes alone, but add the following attributes:

Attribute	Value
<code>android:id</code>	" <code>@+id/text_uri_message</code> "
<code>android:textSize</code>	"18sp"
<code>android:textStyle</code>	"bold"

5.2 Modify AndroidManifest.xml to add an Intent filter

1. Open the AndroidManifest.xml file.
2. Note that MainActivity already has this Intent filter:

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

This Intent filter, which is part of the default project manifest, indicates that this Activity is the main entry point for your app (it has an Intent action of "android.intent.action.MAIN"), and that this Activity should appear as a top-level item in the launcher (its category is "android.intent.category.LAUNCHER").

3. Add a second <intent-filter> tag inside <activity>, and include these elements :

```
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<data android:scheme="http" android:host="developer.android.com" />
```

These lines define an Intent filter for the Activity, that is, the kind of Intent that the Activity can handle. This Intent filter declares these elements:

Filter	Value	Matches
--------	-------	---------

type		
-------------	--	--

action	"android.intent.action.VIEW"	Any Intent with view actions.
category	"android.intent.category.DEFAULT"	Any implicit Intent. This category must be included for your Activity to receive any implicit Intent.
category	"android.intent.category.BROWSABLE"	Requests for browsable links from web pages, email, or other sources.
data	android:scheme="http" android:host="developer.android.com"	URIs that contain a scheme of http <i>and</i> a host name of developer.android.com.

Note that the `data` filter has a restriction on both the kind of links it will accept and the hostname for those URIs. If you'd prefer your receiver to be able to accept any links, you can leave out the `<data>` element.

The application section of AndroidManifest.xml should now look as follows:

```

<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>

        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="http"
                  android:host="developer.android.com" />
        </intent-filter>
    </activity>
</application>

```

```
</activity>  
</application>
```

5.3 Process the Intent

In the `onCreate()` method for your Activity, process the incoming Intent for any data or extras it includes. In this case, the incoming implicit Intent has the URI stored in the Intent data.

1. Open **MainActivity**.
2. In the `onCreate()` method, get the incoming Intent that was used to activate the Activity:

```
Intent intent = getIntent();
```

3. Get the Intent data. Intent data is always a URI object:

```
Uri uri = intent.getData();
```

4. Check to make sure that the `uri` variable is not null. If that check passes, create a string from that URI object:

```
if (uri != null) {  
    String uri_string = "URI: " + uri.toString();  
}
```

-
5. Extract the "URI: " portion of the above into a string resource (uri_label).
 6. Inside that same ifblock, get the TextViewfor the message:

```
TextView textView = findViewById(R.id.text_uri_message);
```

7. Also inside the ifblock, set the text of that TextViewto the URI:

```
textView.setText(uri_string);
```

The onCreate() method for MainActivityshould now look like the following:

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

    Intent intent = getIntent();
    Uri uri = intent.getData();
    if (uri != null) {
        String uri_string = getString(R.string.uri_label)
            + uri.toString();
        TextView textView = findViewById(R.id.text_uri_message);
        textView.setText(uri_string);
    }
}
```

5.4 Run both apps

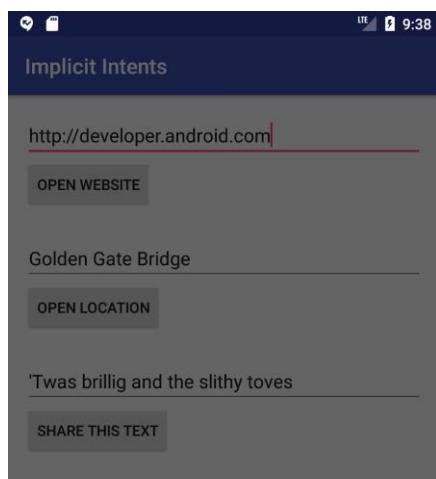
To show the result of receiving an implicit Intent, you will run both the Implicit Intents Receiver and Implicit Intents apps on the emulator or your device.

1. Run the Implicit Intents Receiver app.

Running the app on its own shows a blank Activity with no text. This is because the Activity was activated from the system launcher, and not with an Intent from another app.

2. Run the Implicit Intents app, and click **Open Website** with the default URI.

An app chooser appears asking if you want to use the default browser (Chrome in the figure below) or the Implicit Intents Receiver app. Select **Implicit Intents Receiver**, and click **Just Once**. The Implicit Intents Receiver app launches and the message shows the URI from the original request.



Open with

 Implicit Intents Receiver

 Chrome

JUST ONCE ALWAYS



3. Tap the Back button and enter a different URI. Click **Open Website**.

The receiver app has a very restrictive Intent filter that matches only exact URI protocol (http) and host (developer.android.com). Any other URI opens in the default web browser.

Task 5 solution code

Android Studio project: [ImplicitIntentsReceiver](#)

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: In a previous practical challenge you created a shopping list app builder with an Activity to display the list, and another Activity to pick an item. Add an EditText and a Button to the shopping list Activity to locate a particular store on a map.

Summary

- An implicit Intent allows you to activate an Activity if you know the action, but not the specific app or Activity that will handle that action.
- An Activity that can receive an implicit Intent must define Intent filters in the AndroidManifest.xml file that match one or more Intent actions and categories.
- The Android system matches the content of an implicit Intent and the Intent filters of any available Activity to determine which Activity to activate. If there is more than one available Activity, the system provides a chooser so the user can pick one.
- The ShareCompat.IntentBuilder class makes it easy to build an implicit Intent for sharing data to social media or email.

Related concept

The related concept documentation is in [2.3: Implicit intents](#).

Learn more

Android developer documentation:

- [Application Fundamentals](#)
- [Activities](#)
- [Understand the Activity Lifecycle](#)
- [Intents and Intent Filters](#)
- [Allowing Other Apps to Start Your Activity](#)
- [Google Maps Intents for Android](#)
- [Activity](#)
- [Intent](#)
- [<intent-filter>](#)
- [<activity>](#)
- [Uri](#)
- [ShareCompat.IntentBuilder](#)

Homework

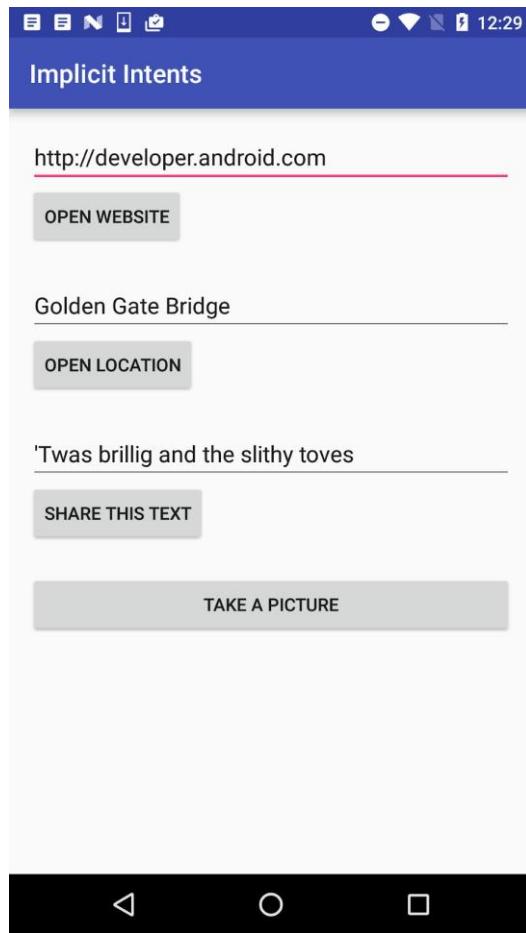
Build and run an app

Open the [ImplicitIntents](#) app that you created.

1. Add another button at the bottom of the screen.
2. When the Button is clicked, launch a camera app to take a picture. (You don't need to return the picture to the original app.)

Note:

If you use the Android emulator to test the camera, open the emulator configuration in the Android AVD manager, choose **Advanced Settings**, and then choose **Emulated** for both front and back cameras. Restart your emulator if necessary.



Answer these questions

Question 1

Which constructor method do you use to create an implicit Intent to launch a camera app?

- new Intent()
- new Intent(Context context, Class<?> class)

-
- new Intent(String action, Uri uri)
 - new Intent(String action)

Question 2

When you create an implicit Intent object, which of the following is true?

- Don't specify the specific Activity or other component to launch.
- Add an Intentaction or Intentcategories (or both).
- Resolve the Intent with the system before calling startActivity() or startActivityForResult().
- All of the above.

Question 3

Which Intentaction do you use to take a picture with a camera app?

- Intent takePicture = new Intent(Intent.ACTION_VIEW);
- Intent takePicture = new Intent(Intent.ACTION_MAIN);
- Intent takePicture = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
- Intent takePicture = new Intent(Intent.ACTION_GET_CONTENT);

Submit your app for grading

Guidance for graders

Check that the app has the following features:

- It displays a **Take a Picture** button at the bottom of the app.
- When clicked, the button launches a camera app on the device.

-
- Before sending the intent, the `onClick()` method for the **Take a Picture** Button ensures that an app is available on the device, using the `resolveActivity()` and `getPackageManager()` methods.

Lesson 3.1: The debugger

Introduction

In previous practicals you used the `Log` class to print information to the system log, which appears in the **Logcat** pane in Android Studio when your app runs. Adding logging statements to your app is one way to find errors and improve your app's operation. Another way is to use the debugger built into Android Studio.

In this practical you learn how to debug your app in an emulator and on the device, set and view breakpoints, step through your code, and examine variables.

What you should already know

You should be able to:

- Create an Android Studio project.
- Use the layout editor to work with `EditText` and `Button` elements.
- Build and run your app in Android Studio, on both an emulator and on a device.
- Read and analyze a stack trace, including last on, first off.
- Add log statements and view the system log (the **Logcat** pane) in Android Studio.

What you'll learn

- How to run your app in debug mode in an emulator or on a device.

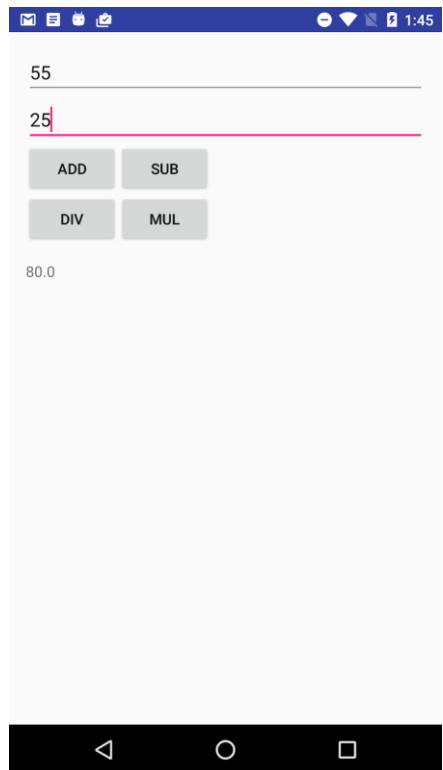
-
- How to step through the execution of your app.
 - How to set and organize breakpoints.
 - How to examine and modify variables in the debugger.

What you'll do

- Build the SimpleCalc app.
- Set and view breakpoints in the code for SimpleCalc.
- Step through your code as it runs.
- Examine variables and evaluate expressions.
- Identify and fix problems in the sample app.

App Overview

The SimpleCalc app has two `EditText` elements and four `Button` elements. When you enter two numbers and click a `Button`, the app performs the calculation for that `Button` and displays the result.



Task 1: Explore the SimpleCalc project and app

For this practical you won't build the SimpleCalc app yourself. The complete project is available at [SimpleCalc](#). In this task you open the SimpleCalc project in Android Studio and explore some of the app's key features.

1.1 Download and Open the SimpleCalc Project

1. Download [SimpleCalc](#) and unzip the file.

-
2. Start Android Studio and select **File > Open**.
 3. Navigate to the folder for SimpleCalc, select that folder file, and click **OK**. The SimpleCalc project builds.
 4. Open the **Project > Android** pane if it is not already open.

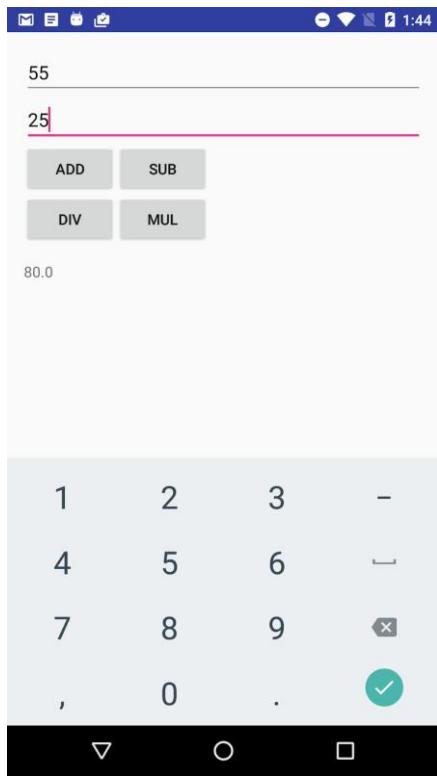
Warning: This app contains errors that you will find and fix. If you run the app on a device or emulator, you might run into unexpected behavior, including crashes in the app.

1.2 Explore the Layout

1. Open **activity_main.xml**.
2. Click the **Text** tab to see the XML code.
3. Click the **Preview** tab to see a preview of the layout.

Examine the layout XML code and design and note the following:

- The layout contains two `EditText` elements for the input, four `Button` elements for the calculations, and one `TextView` to display the result.
- Each calculation `Button` has its own `android:onClick` click handler (`onAdd`, `OnSub`, and so on.)
- The `TextView` for the result does not have any text in it by default.
- The two `EditText` elements have the `android:inputType` attribute and the value "numberDecimal". This attribute indicates that the `EditText` accepts only numbers as input. The keyboard that appears on-screen will only contain numbers. You will learn more about input types for `EditText` elements in a later practical.



1.3 Explore the app code

1. Expand the **app > java** folder in the **Project > Android** pane. In addition to the **MainActivity** class, this project also includes a utility **Calculator** class.
2. Open **Calculator** and examine the code. Note that the operations the calculator can perform are defined by the **Operatorenum**, and that all of the operation methods are public.
3. Open **MainActivity**, and examine the code and comments.

Note the following:

-
- All of the defined android:onClick click handlers call the private compute() method, with the operation name as one of the values from the Calculator.Operatorenumeration.
 - The compute()method calls the privatemethod getOperand()(which in turn calls getOperandText()) to retrieve the number values from the EditTextelements.
 - The compute()method then uses a switchon the operand name to call the appropriate method in the Calculatorinstance (mCalculator).
 - The calculation methods in the Calculator class perform the actual arithmetic and return a value.
 - The last part of the compute() method updates the TextViewwith the result of the calculation.

1.4 Run the app

Run the app and follow these steps:

1. Enter both integer and floating-point values for the calculation.
2. Enter floating-point values with large decimal fractions (for example, **1.6753456**)
3. Divide a number by zero.
4. Leave one or both of the EditTextelements empty, and try any calculation.
5. Click the **Logcat** tab at the bottom of the Android Studio window to open the **Logcat** pane (if it is not already open). Examine the stack trace at the point where the app reports an error.

If one or both of the EditText elements in SimpleCalc are empty, the app reports an exception, as shown in the figure below, and the system log displays the state of the execution stack at the time the app produced the error. The stack trace usually provides important information about why an error occurred.

```

public void onDiv(View view) {
    try {
        compute(Calculator.Operator.DIV);
    } catch (IllegalArgumentException iae) {
        Log.e(TAG, msg: "IllegalArgumentException", iae);
        mResultTextView.setText("Error");
    }
}

```

Stack Trace:

```

java.lang.NumberFormatException: empty String
at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1842)
at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:119)
at java.lang.Double.parseDouble(Double.java:539)
at java.lang.Double.valueOf(Double.java:583)
at com.example.android.SimpleCalc.MainActivity.getOperand(MainActivity.java:136)
at com.example.android.SimpleCalc.MainActivity.compute(MainActivity.java:102)
at com.example.android.SimpleCalc.MainActivity.onMul(MainActivity.java:94) <1 internal calls>
at android.view.View$DeclaredOnClickListener.onClick(View.java:5331)
at android.view.View.performClick(View.java:6256)
at android.view.View$PerformClick.run(View.java:24697)
at android.os.Handler.handleCallback(Handler.java:789)
at android.os.Handler.dispatchMessage(Handler.java:98)
at android.os.Looper.loop(Looper.java:164)
at android.app.ActivityThread.main(ActivityThread.java:6541) <1 internal calls>
at com.android.internal.os.Zygote$MethodAndArgsCaller.run(Zygote.java:240)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:767)

```

Task 2: Run SimpleCalc in the debugger

In this task you'll get an introduction to the debugger in Android Studio, and learn how to set a breakpoint and run your app in debug mode.

2.1 Start and run your app in debug mode

1. In Android Studio, select **Run > Debug app** or click the **Debug icon**  in the toolbar.
2. If your app is already running, you will be asked if you want to restart your app in debug mode. Click **Restart app**.

Android Studio builds and runs your app on the emulator or on the device. Debugging is the same in either case. While Android Studio is initializing the debugger, you may see a

message that says "Waiting for debugger" on the device before you can use your app.

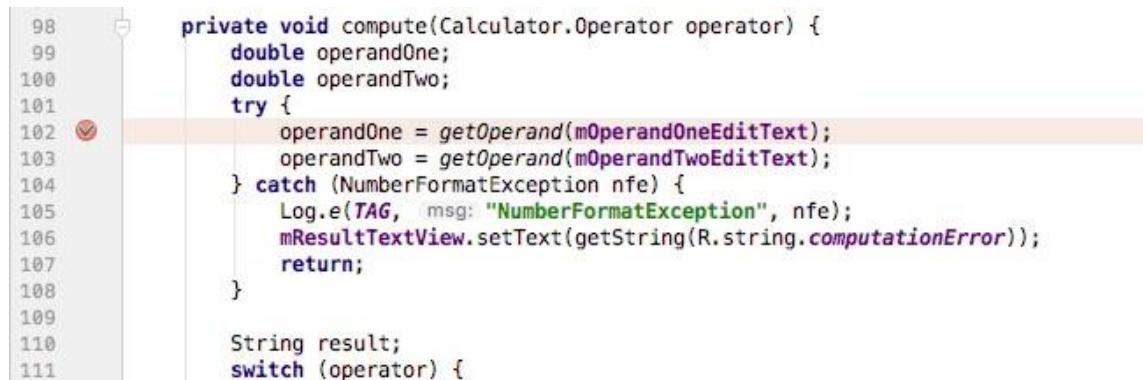
-
3. Click the **Debug** tab at the bottom of the Android Studio window to show the **Debug** pane (or choose **View > Tool Windows > Debug**). The **Debugger** tab in the pane should already be selected, showing the **Debugger** pane.

2.2 Set a breakpoint

A breakpoint is a place in your code where you want to pause the normal execution of your app to perform other actions such as examining variables or evaluating expressions, or executing your code line by line to determine the causes of runtime errors. You can set a breakpoint on any executable line of code.

1. Open **MainActivity**, and click in the fourth line of the `compute()` method (the line just after the `try` statement).
2. Click in the left gutter of the editor pane at that line, next to the line numbers. A red dot appears at that line, indicating a breakpoint. The red dot includes a check mark if the app is already running in debug mode.

As an alternative, you can choose **Run > Toggle Line Breakpoint** or press **Control-F8** (**Command-F8** on a Mac) to set or clear a breakpoint at a line.



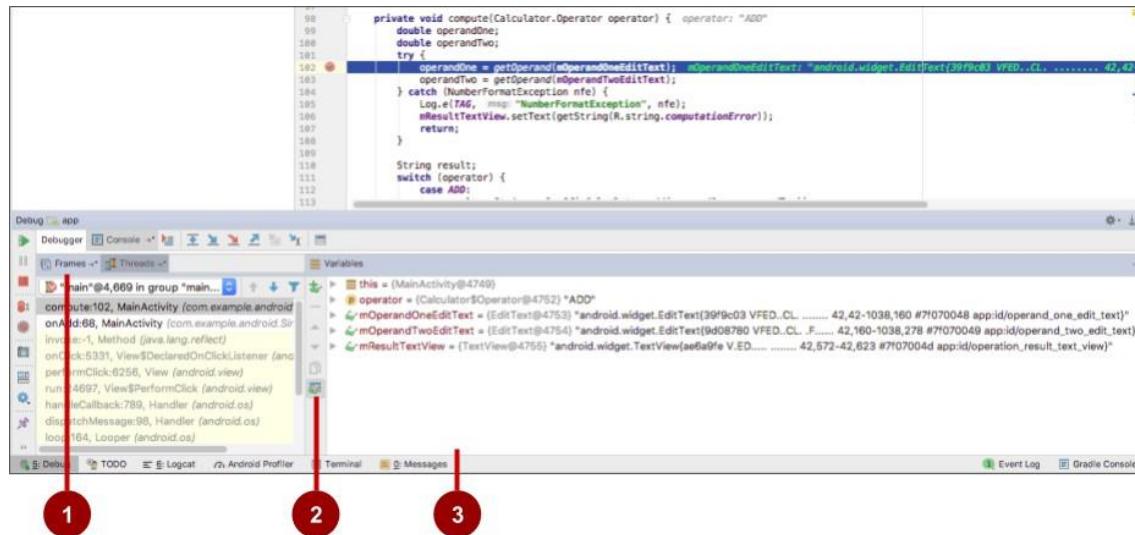
The screenshot shows the Java code for the `compute()` method. Line 102 contains a `try` block. A red dot with a checkmark is placed in the gutter next to line 102, indicating it is a valid breakpoint. The code is as follows:

```
98     private void compute(Calculator.Operator operator) {  
99         double operandOne;  
100        double operandTwo;  
101        try {  
102            operandOne = getOperand(mOperandOneEditText);  
103            operandTwo = getOperand(mOperandTwoEditText);  
104        } catch (NumberFormatException nfe) {  
105            Log.e(TAG, msg: "NumberFormatException", nfe);  
106            mResultTextView.setText(getString(R.string.computationError));  
107            return;  
108        }  
109        String result;  
110        switch (operator) {  
111    }
```

If you click a breakpoint by mistake, you can undo it by clicking the breakpoint. If you clicked a line of code that is not executable, the red dot includes an "x" and a warning appears that the line of code is not executable.

3. In the SimpleCalc app, enter numbers in the `EditText` elements and click one of the `calculate` `Button` elements.

The execution of your app stops when it reaches the breakpoint you set, and the debugger shows the current state of your app at that breakpoint as shown in the figure below.



The figure above shows the **Debug** pane with the **Debugger** and **Console** tabs. The **Debugger** tab is selected, showing the **Debugger** pane with the following features:

1. **Frames** tab: Click to show the **Frames** pane with the current execution stack frames for a given thread. The execution stack shows each class and method that have been called in your app and in the Android runtime, with the most recent method at the top.

Click the **Threads** tab to replace the **Frames** pane with the **Threads** pane. Your app is currently running in the main thread, and that the app is executing the `compute()` method in `MainActivity`.
2. **Watches** button: Click to show the **Watches** pane within the **Variables** pane, which shows the values for any variable watches you have set. Watches allow you to keep track of a specific variable in your program, and see how that variable changes as your program runs.
3. **Variables** pane: Shows the variables in the current scope and their values. At this stage of your app's execution, the available variables are: `this` (for the Activity), `operator` (the operator name from `Calculator.Operator` that the method was called from), as well as the global variables for the `EditText` elements and the `TextView`. Each variable in this pane has an expand icon to expand the list of object properties for the variable. Try expanding a variable to explore its properties.

2.3 Resume your app's execution

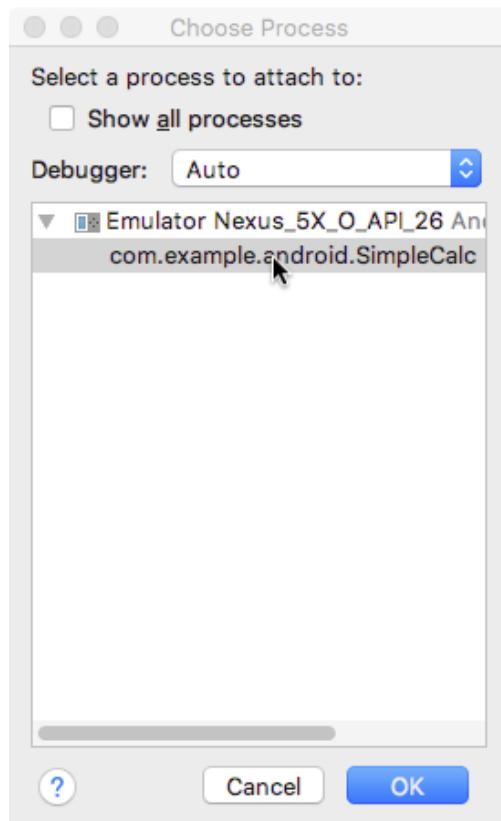
Resume your app's execution by choosing **Run > Resume Program**, or click the **Resume**  icon on the left side of the debugger window.

The SimpleCalc app continues running, and you can interact with the app until the next time the code execution arrives at the breakpoint.

2.4 Debug a running app

If your app is already running on a device or emulator, and you decide you want to debug that app, you can switch an already running app to debug mode.

1. Run the SimpleCalc app normally, with the **Run**  icon.
2. Select **Run > Attach debugger to Android process** or click the **Attach**  icon in the toolbar.
3. Select your app's process from the dialog that appears (shown below). Click **OK**.



The **Debug** pane appears with the **Debugger** pane open, and you can now debug your app as if you had started it in debug mode.

Note: If the **Debug** pane does not automatically appear, click the **Debug** tab at the bottom of the screen. If it is not already selected, click the **Debugger** tab in the **Debug** pane to show the **Debugger** pane.

Task 3: Explore debugger features

In this task we'll explore the various features in the Android Studio debugger, including executing your app line by line, working with breakpoints, and examining variables.

3.1 Step through your app's execution

After a breakpoint, you can use the debugger to execute each line of code in your app one at a time, and examine the state of variables as the app runs.

1. Debug your app in Android Studio, with the breakpoint you set in the last task.
2. In the app, enter numbers in both EditText elements, and click the **Add** button.

Your app's execution stops at the breakpoint that you set earlier, and the **Debugger** pane shows the current state of the app. The current line is highlighted in your code.

3. Click the **Step Over**  button at the top of the debugger window.

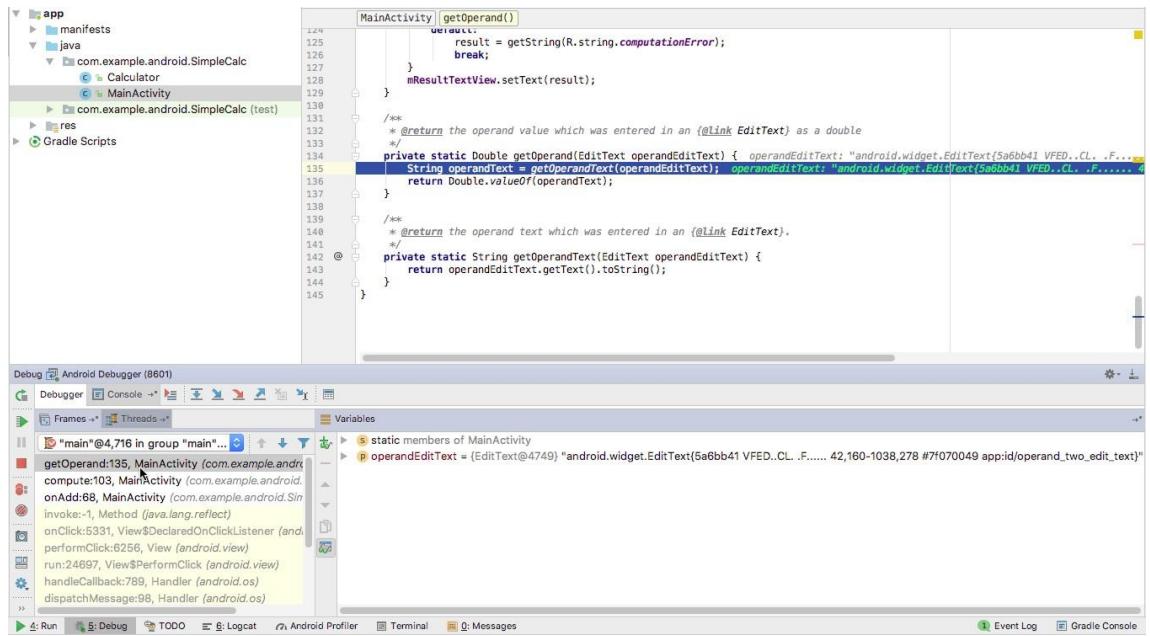
The debugger executes the current line in the `compute()` method (where the breakpoint is, the assignment for `operandOne`), and the highlight moves to the next line in the code (the assignment for `operandTwo`). The **Variables** pane updates to reflect the new execution state, and the current values of variables also appear after each line of your source code in italics.

You can also use **Run > Step Over**, or press **F8**, to step over your code.

4. At the next line (the assignment for `operandTwo`), click the **Step Into**  icon.

Step Into jumps into the execution of a method call in the current line (compared to just executing that method and remaining on the same line). In this case, because that assignment includes a call to `getOperand()`, the debugger scrolls the `MainActivity` code to that method definition.

When you step into a method, the **Frames** pane updates to indicate the new frame in the call stack (here, `getOperand()`), and the **Variables** pane shows the available variables in the new method scope. You can click any of the lines in the **Frames** pane to see the point in the previous stack frame where the method was invoked.



You can also use **Run > Step Into**, or F7, to step into a method.

1. Click **Step Over** to run each of the lines in `getOperand()`. Note that when the method completes the debugger returns you to the point where you first stepped into the method, and all the panels update to show the new information.
2. Click **Step Over** twice to move the execution point to the first line inside the case statement for ADD.
3. Click **Step Into** .

The debugger executes the appropriate method defined in the `Calculator` class, opens the `Calculator.java` file, and scrolls to the execution point in that class. Again, the various panes update to reflect the new state.

4. Use the **Step Out** icon to execute the remainder of that calculation method and pop back out to the `compute()` method in `MainActivity`. You can then continue debugging the `compute()` method from where you left off.

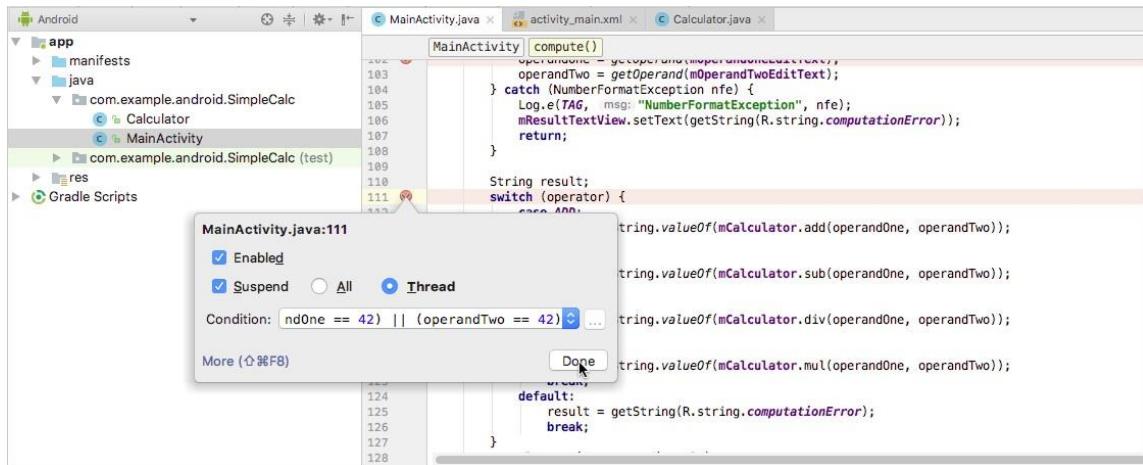
You can also use **Run > Step Out** or press **Shift-F8** to step out of a method execution.

3.2 Work with Breakpoints

Use breakpoints to indicate where in your code you want to interrupt your app's execution to debug that portion of that app.

1. Find the breakpoint you set in the last task—at the start of the compute() method in MainActivity.
2. Add a breakpoint to the start of the switchstatement.
3. Right-click on that new breakpoint to enter a condition, as shown in the figure below, and enter the following test in the **Condition** field:

(operandOne == 42)|| (operandTwo == 42)



4. Click **Done**.

This second breakpoint is a *conditional* breakpoint. The execution of your app will only stop at this breakpoint if the test in the condition is true. In this case, the expression is only true if one or the other operands you entered is **42**. You can enter any Java expression as a condition as long as it returns a boolean.

5. Run your app in debug mode (**Run > Debug**), or click **Resume** if it is already running. In the app, enter two numbers other than 42 and click the **Add** button. Execution halts at the first breakpoint in the compute() method.

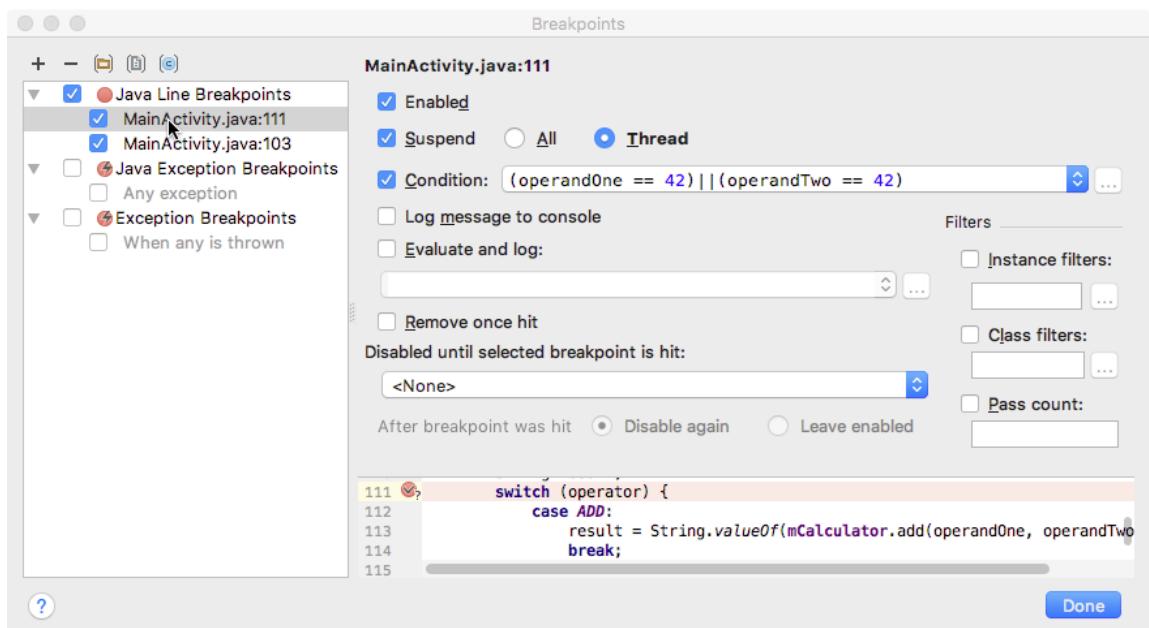
-
6. Click **Resume** to continue debugging the app. Observe that execution did not stop at your second breakpoint, because the condition was not met.
 7. In the app, enter **42** in the first EditText and click any Button. Click **Resume** to resume execution after the first breakpoint. Observe that the second breakpoint at the switch statement—the *conditional* breakpoint—halts execution because the condition was met.
 8. **Right-click** (or **Control-click**) the first breakpoint in compute() and uncheck **Enabled**. Click **Done**. Observe that the breakpoint icon now has a green dot with a red border.

Disabling a breakpoint enables you to temporarily "mute" that breakpoint without actually removing it from your code. If you remove a breakpoint altogether you also lose any conditions you created for that breakpoint, so disabling it is often a better choice.

You can also mute all breakpoints in your app at once with the **Mute Breakpoints**  icon.

9. Click **View Breakpoints**  on the left edge of the debugger window. The **Breakpoints** window appears.

The **Breakpoints** window enables you to see all the breakpoints in your app, enable or disable individual breakpoints, and add additional features of breakpoints including conditions, dependencies on other breakpoints, and logging.



To close the **Breakpoints** window, click **Done**.

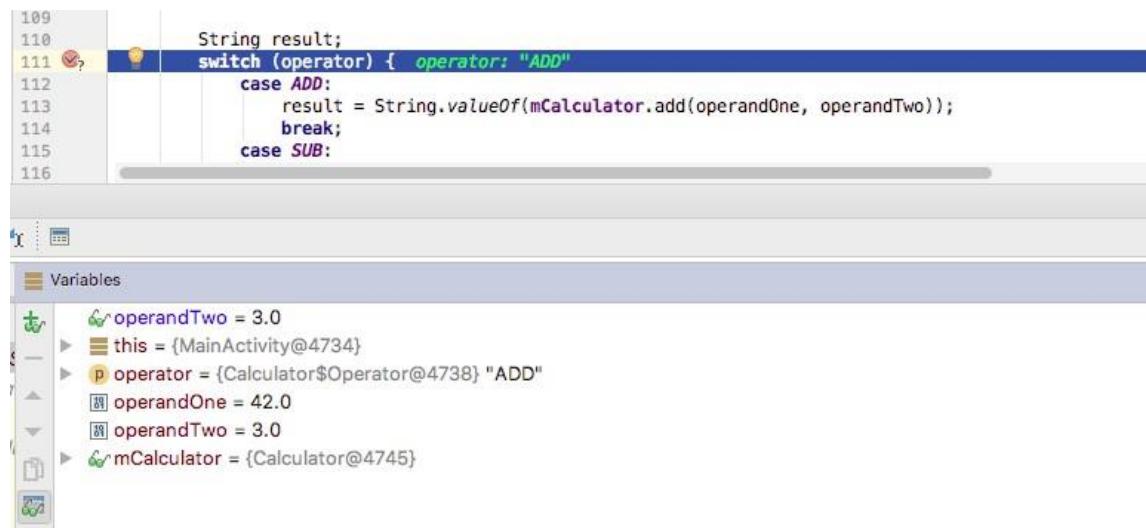
3.3 Examine and modify variables

The Android Studio debugger lets you examine the state of the variables in your app as that app runs.

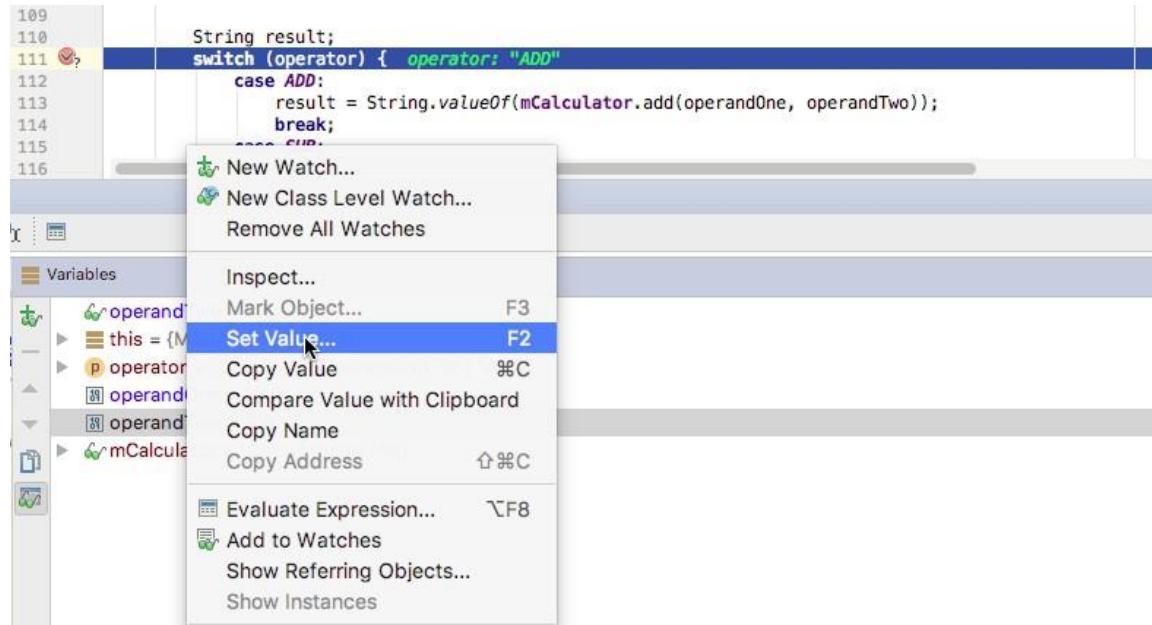
1. Run the SimpleCalc app in debug mode if it is not already running.
2. In the app, enter two numbers, one of them **42**, and click the **Add** button.

The first breakpoint in compute() is still muted. Execution stops at the second breakpoint (the conditional breakpoint at the switchstatement), and the debugger appears.

3. Observe in the **Variables** pane that the operandOne and operandTwo variables have the values you entered into the app.

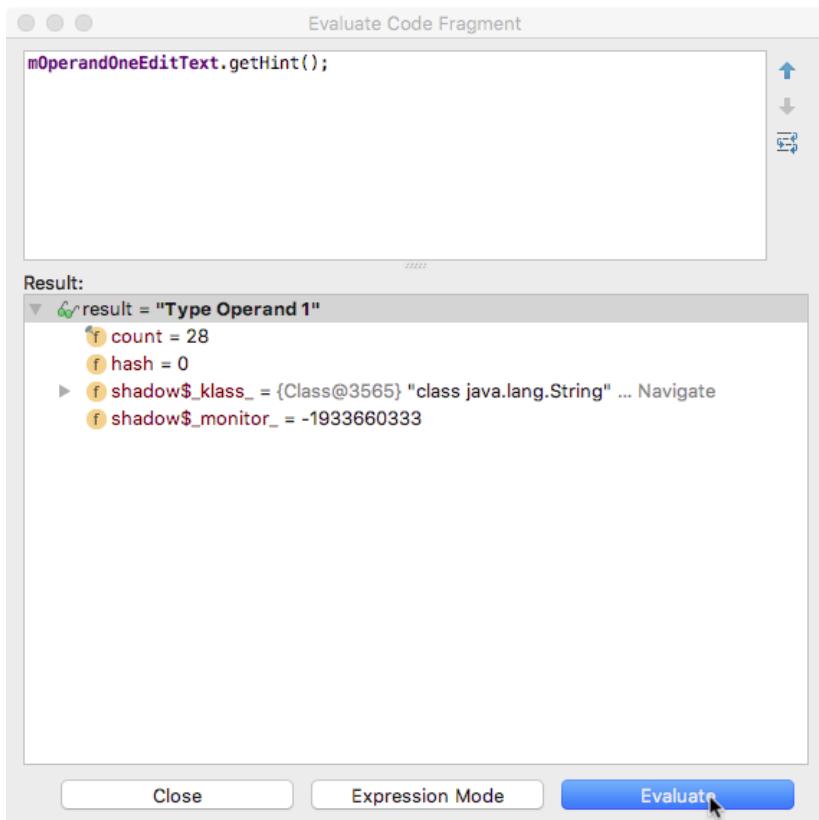


4. The this variable is a MainActivity object. Click the expand icon to see a list of member variables of that object. Click the expand icon again to close the list.
5. Right-click (or Control-click) the **operandOne** variable in the **Variables** pane, and select **Set Value**.



6. Change the value of `operandOne` to **10** and press **Return**.
7. Change the value of `operandTwo` to **10** in the same way and press **Return**.
8. Observe that the result in the app is now based on the variable values you changed in the debugger; for example, since you clicked the **Add** Button in Step 2, the result in the app is now **20**.
9. Click the **Resume** icon to continue running your app.
10. In the app, the original entries (including **42**) are preserved in the `EditText`s. (Their values were changed only in the debugger.) Click the **Add** button. Execution halts at the breakpoint again.
11. Click the **Evaluate Expression** icon, or select **Run > Evaluate Expression**. You can also **right-click** (or **Control-click**) any variable and choose **Evaluate Expression**.

The **Evaluate Code Fragment** window appears. Use it to explore the state of variables and objects in your app, including calling methods on those objects. You can enter any code into this window.



12. Type the statement **mOperandOneEditText.getHint();** into the top field of the **Evaluate Code Fragment** window (as shown in the figure above), and click **Evaluate**.
13. The Result field shows the result of that expression. The hint for this EditText is the string "Type Operand 1", as was originally defined in the XML for that EditText.

The result you get from evaluating an expression is based on the app's current state. Depending on the values of the variables in your app at the time you evaluate expressions, you may get different results.

Note also that if you use **Evaluate Expression** to change the values of variables or object properties, you change the running state of the app.

14. Click **Close** to close the **Evaluate Code Fragment** window.

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: At the end of Task 1, you tried running the SimpleCalc app with no value in one of the `EditText` elements, resulting in an error. Use the debugger to step through the execution of the code and determine precisely why this error occurs. Fix the bug that causes this error.

Summary

- View logging information in Android Studio by clicking the **Logcat** tab.
- Run your app in debug mode by clicking the Debug icon or choosing **Run > Debug app**.
- Click the **Debug** tab to show the **Debug** pane. Click the **Debugger** tab in the **Debug** pane to show the **Debugger** pane (if it is not already selected).
- The **Debugger** pane shows (stack) **Frames**, **Variables** in a specific frame, and **Watches** (active tracking of a variable while the program runs).
- A breakpoint is a place in your code where you want to pause normal execution of your app to perform other actions. Set or clear a debugging breakpoint by clicking in the left gutter of the editor window immediately next to the target line

Related concept

The related concept documentation is in [3.1: The Android Studio debugger](#).

Learn more

Android Studio documentation:

- [Android Studio User Guide](#)
- [Debug Your App](#)
- [Write and View Logs](#)
- [Analyze a Stack Trace](#)
- [Android Debug Bridge](#)
- [Android Profiler](#)
- [Network Profiler](#)
- [CPU Profiler](#)
- [Traceview](#)

Other:

- Video: [Debugging and Testing in Android Studio](#)

Homework

Build and run an app

Open the [SimpleCalc](#) app.

1. In MainActivity, place a breakpoint on the first line of the onAdd() method.
2. Run the app in the debugger. Perform an add operation in the app. The execution stops at the breakpoint.
3. Click **Step Into** to follow the execution of the app step by step. Note that **Step Into** opens and executes files from the Android framework, enabling you to see how Android itself operates on your code.
4. Examine how the **Debug** pane changes as you step through the code for the current stack frame and local variables.
5. Examine how the code itself in the editor pane is annotated as each line is executed.
6. Click **Step Out** to return back to your app if the execution stack gets too deep to understand.

Answer these questions

Question 1

Run the SimpleCalc app without the debugger. Leave one or both of the EditTextelements empty, and try any calculation. Why did the error occur?

- java.lang.NumberFormatException: empty String
- W/OpenGLO Renderer: Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED
- The app may be doing too much work on its main thread.
- The code cache capacity was increased to 128KB.

Question 2

Which function do you perform in the Debug pane in order to execute the current line where the breakpoint is, and then stop at the next line in the code? Choose one:

- **Step Into**
- **Step Over**
- **Step Out**
- **Resume**

Question 3

Which function do you perform in the Debug pane in order to jump to the execution of a method call from the current line where the breakpoint is? Choose one:

- **Step Into**
- **Step Over**
- **Step Out**

- **Resume**

Submit your app for grading

Guidance for graders

No app to submit for this homework assignment.

Lesson 3.2: Unit tests

Introduction

Testing your code can help you catch bugs early in development, when bugs are the least expensive to address. As your app gets larger and more complex, testing improves your code's robustness. With tests in your code, you can exercise small portions of your app in isolation, and you can test in ways that are automatable and repeatable.

Android Studio and the Android Testing Support Library support several different kinds of tests and testing frameworks. In this practical you explore Android Studio's built-in testing functionality, and you learn how to write and run local unit tests.

Local unit tests are tests that are compiled and run entirely on your local machine with the Java Virtual Machine (JVM). You use local unit tests to test the parts of your app that don't need access to the Android framework or an Android-powered device or emulator, for example the internal logic. You also use local unit tests to test parts of your app for which you can create fake ("mock" or stub) objects that pretend to behave like the framework equivalents.

Unit tests are written with JUnit, a common unit testing framework for Java.

What you should already know

You should be able to:

- Create an Android Studio project.
- Build and run your app in Android Studio, on both an emulator and on a device.

-
- Navigate the **Project > Android** pane in Android Studio.

-
- Find the major components of an Android Studio project, including `AndroidManifest.xml`, resources, Java files, and Gradle files.

What you'll learn

- How to organize and run tests in Android Studio.
- Understand what a unit test is.
- Write unit tests for your code.

What you'll do

- Run the initial tests in the SimpleCalc app.
- Add more tests to the SimpleCalc app.
- Run the unit tests to see the results.

App overview

This practical uses the [SimpleCalc](#) app from the previous practical codelab ([Android fundamentals 3.1: The debugger](#)). You can modify that app in place, or make a copy of your project folder before proceeding.

Task 1: Explore and run CalculatorTest

You write and run your tests (both unit tests and instrumented tests) inside Android Studio, alongside the code for your app. Every new Android project includes basic sample classes for testing that you can extend or replace for your own uses.

In this task you return to the SimpleCalc app, which includes a basic unit testing class.

1.1 Explore source sets and CalculatorTest

Source sets are collections of code in your project that are for different build targets or other "flavors" of your app. When Android Studio creates your project, it creates three source sets:

- The **main** source set, for your app's code and resources.
- The **(test)**source set, for your app's local unit tests. The source set shows **(test)** after the package name.
- The **(androidTest)**source set, for Android instrumented tests. The source set shows **(androidTest)** after the package name.

In this task you'll explore how source sets are displayed in Android Studio, examine the Gradle configuration for testing, and run the unit tests for the SimpleCalc app.

Note: The **(androidTest)** source set has been removed from this example for simplicity. It is explained in more detail in another lesson.

1. Open the [SimpleCalc](#) project in Android Studio, if you have not already done so.
1. Open the **Project > Android** pane, and expand the **app** and **java** folders.

The java folder in the Android view lists all the source sets in the app by package name. In this case (as shown below), the app code is in the `com.android.example.SimpleCalc` source set. The test code is in the source set with `test` appearing in parentheses after the package name:



com.android.example.SimpleCalc (test).

2. Expand the **com.android.example.SimpleCalc (test)** folder.

This folder is where you put your app's local unit tests. Android Studio creates a sample test class for you in this folder for new projects, but for SimpleCalc the test class is called `CalculatorTest`.

1. Open **CalculatorTest**.

Examine the code and note the following:

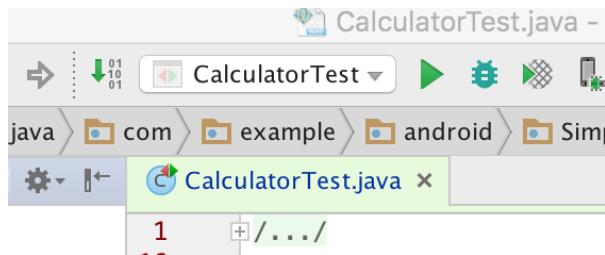
- The only imports are from the `org.junit`, `org.hamcrest`, and `android.testpackages`. There are no dependencies on the Android framework classes.
- The `@RunWith(JUnit4.class)` annotation indicates the runner that will be used to run the tests in this class. A test runner is a library or set of tools that enables testing to occur and the results to be printed to a log. For tests with more complicated setup or infrastructure requirements (such as Espresso) you'll use different test runners. For this example we're using the basic JUnit4 test runner.
- The `@SmallTest` annotation indicates that all the tests in this class are unit tests that have no dependencies, and run in milliseconds. The `@SmallTest`, `@MediumTest`, and `@LargeTest` annotations are conventions that make it easier to bundle groups of tests into suites of similar functionality.
- The `setUp()` method is used to set up the environment before testing, and includes the `@Before` annotation. In this case the setup creates a new instance of the `Calculator` class and assigns it to the `mCalculatorMember` variable.
- The `addTwoNumbers()` method is an actual test, and is annotated with `@Test`. Only methods in a test class that have an `@Test` annotation are considered tests to the test runner. Note that by convention test methods do not include the word "test."
- The first line of `addTwoNumbers()` calls the `add()` method from the `Calculator` class. You can only test methods that are public or package-protected. In this case the `Calculator` is a public class with public methods, so all is well.
- The second line is the assertion for the test. Assertions are expressions that must evaluate and result in true for the test to pass. In this case the assertion is that the result you got from the `add` method ($1 + 1$) matches the given number 2. You'll learn more about how to create assertions later in this practical.

1.2 Run tests in Android Studio

In this task you'll run the unit tests in the test folder and view the output for both successful and failed tests.

1. In the **Project > Android** pane, right-click (or Control-click) **CalculatorTest** and select **Run 'CalculatorTest'**.

The project builds, if necessary, and the **CalculatorTest** pane appears at the bottom of the screen. At the top of the pane, the drop-down list for available execution configurations also changes to **CalculatorTest**.



All the tests in the **CalculatorTest** class run, and if those tests are successful, the progress bar at the top of the view turns green. (In this case, there is currently only one test.) A status message in the footer also reports "Tests Passed."



1. Open **CalculatorTest** if it is not already open, and change the assertion in **addTwoNumbers()** to:

```
assertThat(resultAdd, is(equalTo(3d)));
```

-
2. In the run configurations dropdown menu at the top of the screen, select **CalculatorTest** (if it is not already selected) and click **Run** .

The test runs again as before, but this time the assertion fails (3 is not equal to 1 + 1.) The progress bar in the run view turns red, and the testing log indicates where the test (assertion) failed and why.

3. Change the assertion in `addTwoNumbers()` back to the correct test and run your tests again to ensure they pass.
4. In the run configurations dropdown, select **app** to run your app normally.

Task 2: Add more unit tests to CalculatorTest

With unit testing, you take a small bit of code in your app such as a method or a class, and isolate it from the rest of your app, so that the tests you write make sure that one small bit of the code works in the way you'd expect. Typically, a unit test calls a method with a variety of different inputs, and verifies that the method does what you expect and returns what you expect it to return.

In this task you learn more about how to construct unit tests. You'll write additional unit tests for the `CalculatorUtility` methods in the `SimpleCalc` app, and run those tests to make sure that they produce the output you expect.

Note: Unit testing, test-driven development, and the JUnit 4 API are all large and complex topics and outside the scope of this course.

2.1 Add more tests for the `add()` method

Although it is impossible to test every possible value that the `add()` method may ever see, it's a good idea to test for input that might be unusual. For example, consider what happens if the `add()`

method gets arguments:

- With negative operands
- With floating-point numbers

-
- With exceptionally large numbers
 - With operands of different types (a float and a double, for example)
 - With an operand that is zero
 - With an operand that is infinity

In this task we'll add more unit tests for the `add()` method to test different kinds of inputs.

1. Add a new method to `CalculatorTest` called `addTwoNumbersNegative()`. Use this skeleton:

```
@Test  
public void addTwoNumbersNegative() {  
}
```

This test method has a similar structure to `addTwoNumbers()`: it is a public method, with no parameters, that returns `void`. It is annotated with `@Test`, which indicates it is a single unit test.

Why not just add more assertions to `addTwoNumbers()`? Grouping more than one assertion into a single method can make your tests harder to debug if only one assertion fails, and obscures the tests that do succeed. The general rule for unit tests is to provide a test method for every individual assertion.

2. Run all tests in `CalculatorTest`, as before.

In the test window both `addTwoNumbers` and `addTwoNumbersNegative` are listed as available (and passing) tests in the left panel. The `addTwoNumbersNegative` test still passes even though it doesn't contain any code—a test that does nothing is still considered a successful test.

3. Add a line to `addTwoNumbersNegative()` to invoke the `add()` method in the `Calculator` class with a negative operand.

```
double resultAdd = mCalculator.add(-1d, 2d);
```

The dnotation after each operand indicates that these are numbers of type double. Because the add() method is defined with double parameters, a float or int will also work. Indicating the type explicitly enables you to test other types separately, if you need to.

4. Add an assertion with assertThat().

```
assertThat(resultAdd, is(equalTo(1d)));
```

The assertThat() method is a JUnit4 assertion that claims the expression in the first argument is equal to the one in the second argument. Older versions of JUnit used more specific assertion methods (assertEquals(), assertNull(), or assertTrue()), but assertThat() is a more flexible, more debuggable and often easier to read format.

The assertThat() method is used with *matchers*. Matchers are the chained method calls in the second operand of this assertion, is(equalTo()). The Hamcrest framework defines the available matchers you can use to build an assertion. ("Hamcrest" is an anagram for "matchers.") Hamcrest provides many basic matchers for most basic assertions. You can also define your own custom matchers for more complex assertions.

In this case the assertion is that the result of the add() operation (-1 + 2) equals 1.

5. Add a new unit test to CalculatorTest for floating-point numbers:

```
@Test
public void addTwoNumbersFloats() {
    double resultAdd = mCalculator.add(1.111f, 1.111d);
    assertThat(resultAdd, is(equalTo(2.222d)));
}
```

Again, a very similar test to the previous test method, but with one argument to add() that is explicitly type float rather than double. The add() method is defined with parameters of type double, so you can call it with a floattype, and that number is promoted to a double.

6. Click **Run**  to run all the tests again.

This time the test failed, and the progress bar is red. This is the important part of the error message:

```
java.lang.AssertionError:  
Expected: is <2.222>  
      but: was <2.2219999418258665>
```

Arithmetic with floating-point numbers is inexact, and the promotion resulted in a side effect of additional precision. The assertion in the test is technically false: the expected value is not equal to the actual value.

The question this raises is: When you have a precision problem with promoting float arguments, is that a problem with your code, or a problem with your test? In this particular case both input arguments to the add() method from the SimpleCalc app will always be type double, so this is an arbitrary and unrealistic test. However, if your app was written such that the input to the add() method could be either double or float, and you only care about *some* precision, you need to provide some wiggle room to the test so that "close enough" counts as a success.

7. Change the assertThat() method to use the closeTo() matcher:

```
assertThat(resultAdd, is(closeTo(2.222, 0.01)));
```

You need to make a choice for the matcher. Click on **closeTo** twice (until the entire expression is underlined), and press Alt+Enter (Option+Return on a Mac). Choose **isCloseTo.closeTo (org.hamcrest.number)**.

8. Click **Run**  to run all the tests again.

This time the test passes.

With the closeTo() matcher, rather than testing for exact equality you can test for equality within a specific delta. In this case the closeTo() matcher method takes two arguments: the expected value and the amount of delta. In the example above, that delta is just two decimal points of precision.

2.2 Add unit tests for the other calculation methods

Use what you learned in the previous task to fill out the unit tests for the Calculator class.

1. Add a unit test called `subTwoNumbers()` that tests the `sub()` method.
2. Add a unit test called `subWorksWithNegativeResults()` that tests the `sub()` method where the given calculation results in a negative number.
3. Add a unit test called `mulTwoNumbers()` that tests the `mul()` method.
4. Add a unit test called `mulTwoNumbersZero()` that tests the `mul()` method with at least one argument as zero.
5. Add a unit test called `divTwoNumbers()` that tests the `div()` method with two non-zero arguments.
6. Add a unit test called `divTwoNumbersZero()` that tests the `div()` method with a double dividend and zero as the divider.

All of these tests should pass, except `divTwoNumbersZero()` which causes an illegal argument exception for dividing by zero. If you run the app, enter zero as Operand 2, and click **Div** to divide, the result is an error.

Task 2 solution code

Android Studio project: [SimpleCalcTest](#)

The following code snippet shows the tests for this task:

```
@Test
```

Page 339

```
public void addTwoNumbers() {
    double resultAdd = mCalculator.add(1d, 1d);
    assertThat(resultAdd, is(equalTo(2d)));
}

@Test
public void addTwoNumbersNegative() {
    double resultAdd = mCalculator.add(-1d, 2d);
    assertThat(resultAdd, is(equalTo(1d)));
}

@Test
public void addTwoNumbersFloats() {
    double resultAdd = mCalculator.add(1.111f, 1.111d);
    assertThat(resultAdd, is(closeTo(2.222, 0.01)));
}

@Test
public void subTwoNumbers() {
    double resultSub = mCalculator.sub(1d, 1d);
    assertThat(resultSub, is(equalTo(0d)));
}

@Test
public void subWorksWithNegativeResult() {
    double resultSub = mCalculator.sub(1d, 17d);
    assertThat(resultSub, is(equalTo(-16d)));
}

@Test
public void mulTwoNumbers() {
    double resultMul = mCalculator.mul(32d, 2d);
    assertThat(resultMul, is(equalTo(64d)));
}

@Test
public void divTwoNumbers() {
    double resultDiv = mCalculator.div(32d, 2d);
    assertThat(resultDiv, is(equalTo(16d)));
}

@Test
public void divTwoNumbersZero() {
    double resultDiv = mCalculator.div(32d, 0);
    assertThat(resultDiv, is(equalTo(Double.POSITIVE_INFINITY)));
}
```

Coding challenges

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge 1: Dividing by zero is always worth testing for, because it is a special case in arithmetic. How might you change the app to more gracefully handle divide by zero? To accomplish this challenge, start with a test that shows what the right behavior should be.

Remove the `divTwoNumbersZero()` method from `CalculatorTest`, and add a new unit test called `divByZeroThrows()` that tests the `div()` method with a second argument of zero, with the expected result as `IllegalArgumentException.class`. This test will pass, and as a result it will demonstrate that any division by zero will result in this exception.

After you learn how to write code for an [Exception](#) handler, your app can handle this exception gracefully by, for example, displaying a [Toast](#) message to the user to change Operand 2 from zero to another number.

Challenge 2: Sometimes it's difficult to isolate a unit of code from all of its external dependencies. Rather than organize your code in complicated ways just so you can test it more easily, you can use a mock framework to create fake ("mock") objects that pretend to be dependencies. Research the [Mockito](#) framework, and learn how to set it up in Android Studio. Write a test class for the `calcButton()` method in `SimpleCalc`, and use Mockito to simulate the Android context in which your tests will run.

Summary

Android Studio has built-in features for running local unit tests:

- Local unit tests use the JVM of your local machine. They don't use the Android framework.
- Unit tests are written with JUnit, a common unit testing framework for Java.
- JUnit tests are located in the (test) folder in the Android Studio **Project > Android** pane.
- Local unit tests only need these packages: org.junit, org.hamcrest, and android.test.
- The `@RunWith(JUnit4.class)` annotation tells the test runner to run tests in this class.
- `@SmallTest`, `@MediumTest`, and `@LargeTest` annotations are conventions that make it easier to bundle similar groups of tests
- The `@SmallTest` annotation indicates all the tests in a class are unit tests that have no dependencies and run in milliseconds.
- Instrumented tests are tests that run on an Android-powered device or emulator. Instrumented tests have access to the Android framework.
- A test runner is a library or set of tools that enables testing to occur and the results to be printed to the log.

Related concept

The related concept documentation is in [3.2: App testing](#).

Learn more

Android Studio documentation:

- [Android Studio User Guide](#)
- [Write and View Logs](#)

Android developer documentation:

- [Best Practices for Testing](#)
- [Getting Started with Testing](#)
- [Building Local Unit Tests](#)

Other:

- [JUnit 4 Home Page](#)
- [JUnit 4 API Reference](#)
- [java.lang.Math](#)
- [Java Hamcrest](#)
- [Mockito Home Page](#)
- Video: [Android Testing Support - Testing Patterns](#)
- [Android Testing Codelab](#)
- [Android Tools Protip: Test Size Annotations](#)
- [The Benefits of Using assertThat over other Assert Methods in Unit Tests](#)

Homework

Build and run an app

Open the [SimpleCalc](#) app from the practical on using the debugger. You're going to add a **POW** button to the layout. The button calculates the first operand raised to the power of the second operand. For example, given operands of 5 and 4, the app calculates 5 raised to the power of 4, or 625.

Before you write the implementation of your power button, consider the kind of tests you might want to perform using this calculation. What unusual values may occur in this calculation?

1. Update the Calculator class in the app to include a pow() method. Hint: Consult the documentation for the [java.lang.Math](#) class.
2. Update the MainActivity class to connect the **POW** Button to the calculation.

Now write each of the following tests for your pow() method. Run your test suite each time you write a test, and fix the original calculation in your app if necessary:

- A test with positive integer operands.
- A test with a negative integer as the first operand.
- A test with a negative integer as the second operand.

-
- A test with 0 as the first operand and a positive integer as the second operand.

-
- A test with 0 as the second operand.
 - A test with 0 as the first operand and -1 as the second operand. (Hint: consult the documentation for Double.POSITIVE_INFINITY.)
 - A test with -0 as the first operand and any negative number as the second operand.

Answer these questions

Question 1

Which statement best describes a local unit test? Choose one:

- Tests that run on an Android-powered device or emulator and have access to the Android framework.
- Tests that enable you to write automated UI test methods.
- Tests that are compiled and run entirely on your local machine with the Java Virtual Machine (JVM).

Question 2

Source sets are collections of related code. In which source set are you likely to find unit tests?
Choose one:

- app/res
- com.example.android.SimpleCalcTest
- com.example.android.SimpleCalcTest (test)
- com.example.android.SimpleCalcTest (androidTest)

Question 3

Which annotation is used to mark a method as an actual test? Choose one:

- @RunWith(JUnit4.class)
- @SmallTest

-
- @Before
 - @Test

Submit your app for grading

Guidance for graders

Check that the app has the following features:

- It displays a **POW** Button that provides an exponential ("power of") calculation.
- The implementation of MainActivity includes a click handler for the **POW** Button.
- The implementation of Calculator includes a pow() method that performs the calculation.
- The CalculatorTest() method includes separate test methods for the pow() method in the Calculator class that perform tests for negative and 0 operands, and for the case of 0 and -1 as the operands.

Lesson 3.3: Support libraries

Introduction

The Android SDK includes the Android Support Library, which is a collection of several libraries. These libraries provide features that aren't built into the Android framework, including the following:

- Backward-compatible versions of framework components, so that apps running on older versions of the Android platform can support features made available in newer versions of the platform
- Additional layout and user interface elements
- Support for different device form factors, such as TV devices or wearables
- Components to support Material Design elements
- Other features, including palette support, annotations, percentage-based layout dimensions, and preferences

What you should already know

You should be able to:

- Create an Android Studio project.
- Use the layout editor to work with EditText and Button elements.
- Build and run your app in Android Studio, on both an emulator and on a device.
- Navigate the **Project > Android** pane in Android Studio.
- Find the major components of an Android Studio project, including `AndroidManifest.xml`, resources, Java files, and Gradle files.

What you'll learn

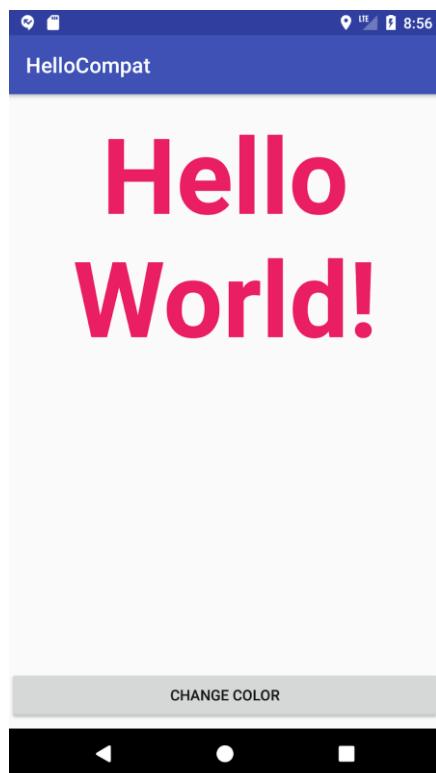
- How to verify that the Android Support Library is available in your Android Studio installation.
- How to indicate support library classes in your app.
- How to tell the difference between the values for `compileSdkVersion`, `targetSdkVersion`, and `minSdkVersion`.
- How to recognize deprecated or unavailable APIs in your code.
- More about the Android support libraries.

What you'll do

- Create a new app with one `TextView` and one `Button`.
- Verify that the Android Support Repository (containing the Android Support Library) is available in your Android Studio installation.
- Explore the `build.gradle` files for your app project.
- Manage class or method calls that are unavailable for the version of Android your app supports.
- Use a compatibility class from the support library to provide backward-compatibility for your app.

App overview

In this practical you'll create an app called HelloCompat with one TextView that displays "Hello World" on the screen, and one Button that changes the color of the text. There are 20 possible colors, defined as resources in the color.xml file, and each button click randomly picks one of those colors.



The methods to get a color value from the app's resources have changed with different versions for the Android framework. This example uses the ContextCompat class in the Android Support Library, which allows you to use a method that works for all versions.

Task 1: Set up your project to use support libraries

For this task you'll set up a new project for the HelloCompat app and implement the layout and basic behavior.

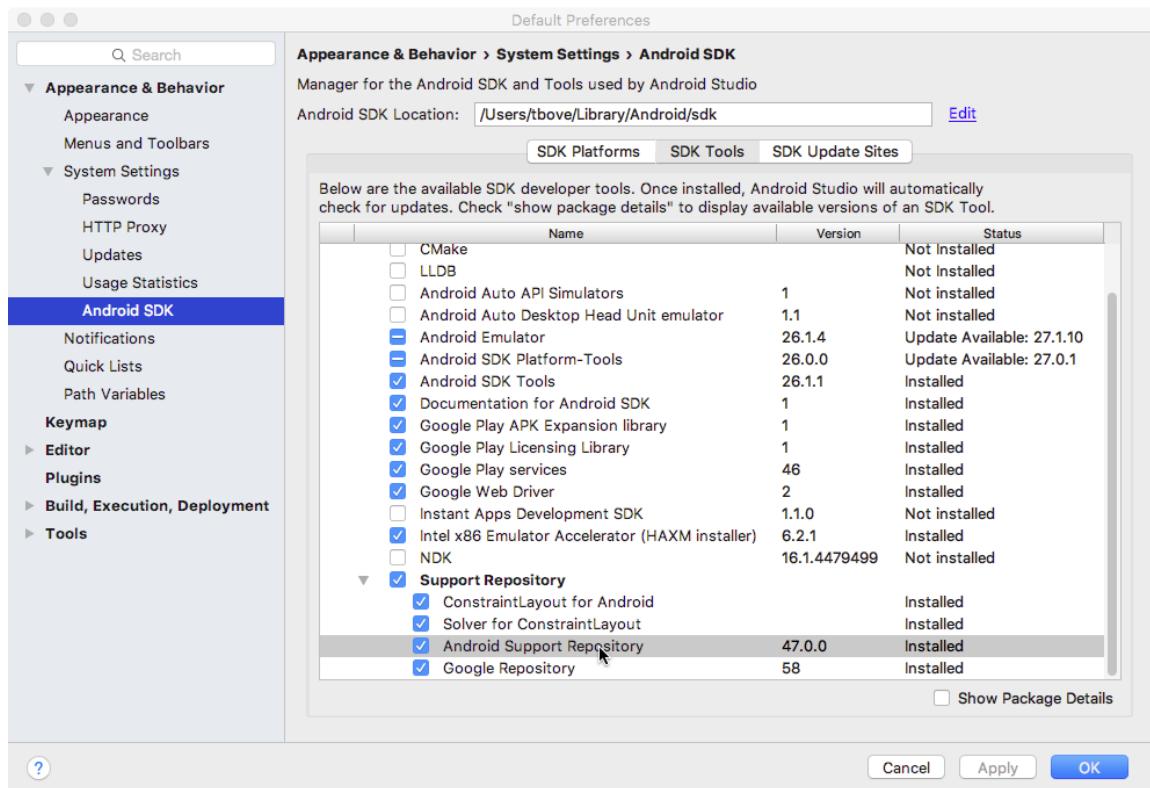
1.1 Verify that the Android Support Repository is available

The Android Support Library is downloaded as part of the Android SDK, and available in the Android SDK manager. In Android Studio, you'll use the Android Support Repository—the local repository for the support libraries—to get access to the libraries from within your Gradle build files. In this task you'll verify that the Android Support Repository is downloaded and available for your projects.

15. In Android Studio, select **Tools > Android > SDK Manager**, or click the **SDK Manager**  icon.

The Android SDK **Default Preferences** pane appears.

16. Click the **SDK Tools** tab and expand **Support Repository**, as shown in the figure below.



17. Look for **Android Support Repository** in the list.

If **Installed** appears in the Status column, you're all set. Click **Cancel**.

If **Not installed** or **Update Available** appears, click the checkbox next to **Android Support Repository**. A download icon should appear next to the checkbox. Click **OK**.

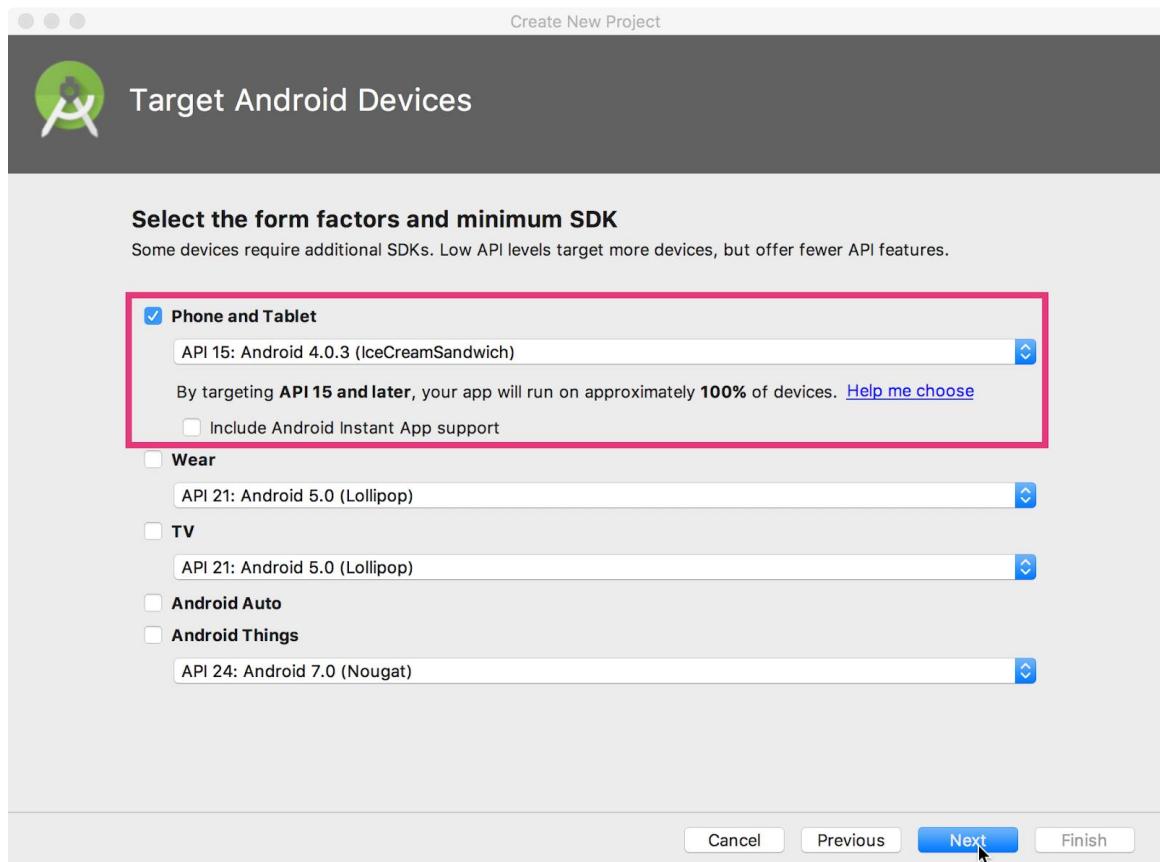
18. Click **OK** again, and then **Finish** when the support repository has been installed.

1.2 Set up the project and examine build.gradle

1. Create a new project called **HelloCompat**.

On the Target Android Devices page, **API 15: Android 4.0.3 (IceCreamSandwich)** is selected for the minimum SDK. As you've learned in previous lessons, this is the oldest

version of the Android platform your app will support.



1. Click **Next**, and choose the **Empty Activity** template.
2. Click **Next**, and ensure that the **Generate Layout file** and **Backwards Compatibility (App Compat)** options are checked. The latter option ensures that your app will be backwards-compatible with previous versions of Android.
3. Click **Finish**.

Explore build.gradle (Module:app)

1. In Android Studio, make sure the **Project > Android** pane is open.
2. Expand **Gradle Scripts** and open the **build.gradle (Module: app)** file.

Note that build.gradle for the overall project (build.gradle (Project: helpcompat)) is a different file from the build.gradle for the app module. In the build.gradle (Module: app) file.

3. Locate the compileSdkVersion line near the top of the file. For example:

```
compileSdkVersion 26
```

The *compile* version is the Android framework version your app is compiled with in Android Studio. For new projects the compile version is the most recent set of framework APIs you have installed. This value affects *only* Android Studio itself and the warnings or errors you get in Android Studio if you use older or newer APIs.

4. Locate the minSdkVersion line in the defaultConfig section a few lines down.

```
minSdkVersion 15
```

The *minimum* version is the oldest Android API version your app runs under. It's the same number you chose in Step 1 when you created your project. The Google Play store uses this number to make sure that your app can run on a given user's device. Android Studio also uses this number to warn you about using deprecated APIs.

5. Locate the targetSdkVersion line in the defaultConfig section. For example:

```
targetSdkVersion 26
```

The *target* version indicates the API version your app is designed and tested for. If the API of the Android platform is higher than this number (that is, your app is running on a newer device), the platform may enable compatibility behaviors to make sure that your app continues to work the way it was designed to. For example, Android 6.0 (API 23) provides a new runtime permissions model. If your app targets a lower API level, the platform falls back to the older install-time permissions model.

Although the target SDK can be the same number as the compile SDK, it is often a lower number that indicates the most recent version of the API for which you have tested

your app.

-
6. Locate the dependencies section of build.gradle, near the end of the file. For example:

```
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 dependencies section for a new project includes several dependencies to enable testing with Espresso and JUnit, as well as the v7 appcompat support library. The version numbers for these libraries in your project may be different than those shown here.

The v7 appcompat support library provides backward-compatibility for older versions of Android all the way back to API 9. It includes the v4 compat library as well, so you don't need to add both as a dependency.

7. Update the version numbers, if necessary.

If the current version number for a library is lower than the currently available library version number, Android Studio highlights the line and warns you that a new version is available ("A newer version of com.android.support:appcompat-v7 is available"). Edit the version number to the updated version.

Tip: You can also click anywhere in the highlight line and press Alt+Enter (Option+Return on a Mac). Select **Change to xx.xx.x** from the menu, where xx.xx.x is the most up-to-date version available.

8. Update the compileSdkVersion number, if necessary.

The major version number of the support library (the first number) must match your compileSdkVersion. When you update the support library version, you may also need to update compileSdkVersion to match.

9. Click **Sync Now** to sync your updated Gradle files with the project, if prompted.

-
10. Install missing SDK platform files, if necessary.

If you update compileSdkVersion, you may need to install the SDK platform components to match. Click **Install missing platform(s) and sync project** to start this process.

Task 2: Implement the layout and MainActivity

For this task you'll implement the layout and basic behavior for the MainActivity class.

2.1 Change the layout and colors

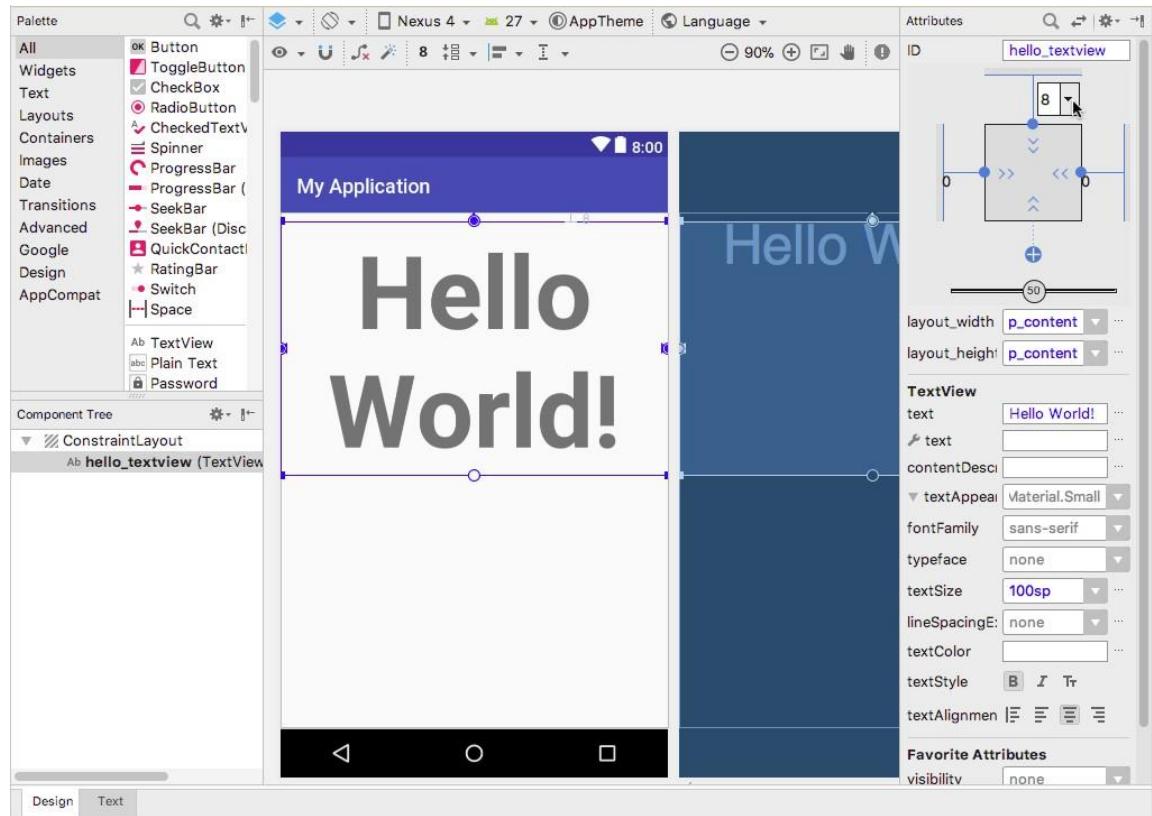
In this task you will modify the activity_main.xml layout for the app.

1. Open **activity_main.xml** in the **Project > Android pane**.
2. Click the **Design** tab (if it is not already selected) to show the layout editor.
3. Select the "Hello World" TextView in the layout and open the **Attributes** pane.
4. Change the TextView attributes as follows:

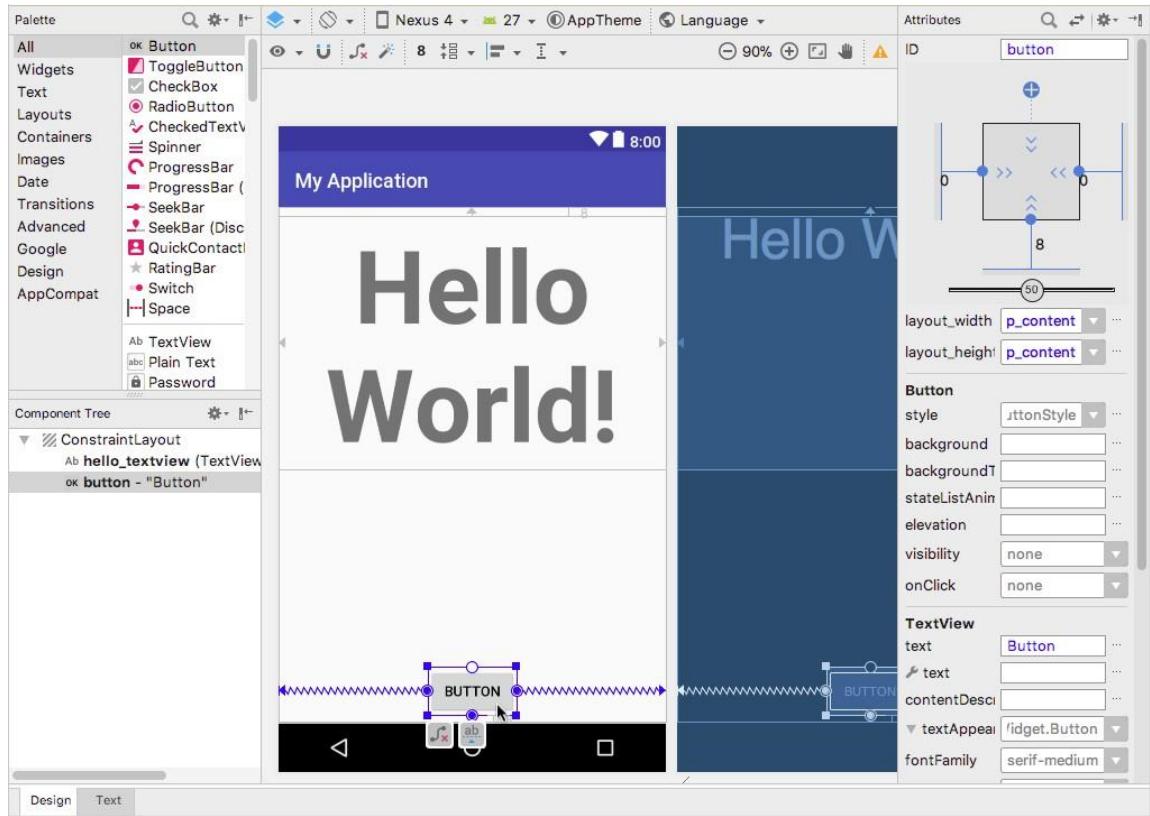
Attribute field	Enter the following:
ID	hello_textview
textStyle	B(bold)
textAlignment	Center the paragraph icon
textSize	100sp

This adds the android:id attribute to the TextView with the id set to hello_textview, changes the text alignment, makes the text bold, and sets a larger text size of 100sp.

-
5. Delete the constraint that stretches from the bottom of the hello_textview TextView to the bottom of the layout, so that the TextView snaps to the top of the layout, and choose **8** (8dp) for the top margin as shown below.



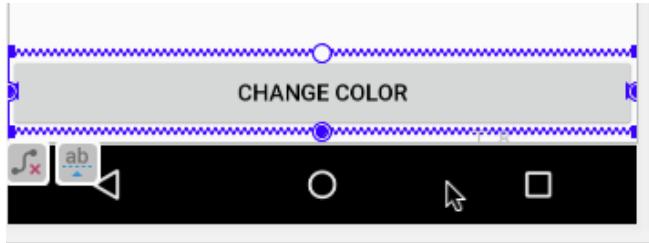
6. Drag a Button to the bottom of the layout, and add constraints to the left and right sides and bottom of the layout, as shown in the figure below.



7. Change the **layout_width** attribute in the **Attributes** pane for the Button to **match_constraint**.
8. Change the other attributes in the **Attributes** pane for the Button as follows:

Attribute field	Enter the following:
ID	color_button
text	"Change Color"

The Button should now appear in the layout as shown below:



9. In a previous lesson you learned how to extract a string resource from a literal text string. Click the **Text** tab to switch to XML code, and extract the "Hello Text!" and "Change Color" strings in the TextView and Button, and enter string resource names for them.
10. Add the following android:onClick attribute to the Button:

```
    android:onClick="changeColor"
```

11. To add colors, expand **res** and **values** in the **Project > Android** pane, and open **colors.xml**.
12. Add the following color resources to the file:

```
<color name="red">#F44336</color>
<color name="pink">#E91E63</color>
<color name="purple">#9C27B0</color>
<color name="deep_purple">#673AB7</color>
<color name="indigo">#3F51B5</color>
<color name="blue">#2196F3</color>
<color name="light_blue">#03A9F4</color>
<color name="cyan">#00BCD4</color>
<color name="teal">#009688</color>
<color name="green">#4CAF50</color>
<color name="light_green">#8BC34A</color>
<color name="lime">#CDDC39</color>
<color name="yellow">#FFEB3B</color>
<color name="amber">#FFC107</color>
<color name="orange">#FF9800</color>
<color name="deep_orange">#FF5722</color>
<color name="brown">#795548</color>
<color name="grey">#9E9E9E</color>
<color name="blue_grey">#607D8B</color>
<color name="black">#000000</color>
```

These color values and names come from the recommended color palettes for Android apps defined at [Material Design - Style - Color](#). The codes indicate color RGB values in hexadecimal.

2.2 Add behavior to MainActivity

In this task you'll finish setting up the project by adding private variables and implementing `onCreate()` and `onSaveInstanceState()`.

1. Open **MainActivity**.
2. Add a private variable at the top of the class to hold the `TextView` object.

```
private TextView mHelloTextView;
```

3. Add the following color array just after the private variable:

```
private String[] mColorArray = {"red", "pink", "purple", "deep_purple",
    "indigo", "blue", "light_blue", "cyan", "teal", "green",
    "light_green", "lime", "yellow", "amber", "orange", "deep_orange",
    "brown", "grey", "blue_grey", "black" };
```

Each color name corresponds to the name of a color resource in `color.xml`.

4. In the `onCreate()` method, use `findViewById()` to get a reference to the `TextView` instance and assign it to that private variable:

```
mHelloTextView = findViewById(R.id.hello_textview);
```

1. Also in onCreate(), restore the saved instance state, if any:

```
// restore saved instance state (the text color)
if (savedInstanceState != null) {
    mHelloTextView.setTextColor(savedInstanceState.getInt("color"));
}
```

2. Add the onSaveInstanceState() method to MainActivity to save the text color:

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    // save the current text color
    outState.putInt("color", mHelloTextView.getCurrentTextColor());
}
```

Task 2 solution code

The following is the solution code for the XML layout and a code snippet in the MainActivity class for the HelloCompat app so far.

XML layout

The XML layout for the activity_main.xml file is shown below. The changeColorclick handler for the android:onClickattribute for the Buttonis underlined in red because it hasn't yet been defined. You define it in the next task.

```
<android.support.constraint.ConstraintLayout 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"
    tools:context="com.example.android.myapplication.MainActivity">

    <TextView
        android:id="@+id/hello_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="@string/hello_world"
        android:textAlignment="center" android:textSize="100sp"
        android:textStyle="bold"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/color_button"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:text="@string/change_color"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:onClick="changeColor"/>

</android.support.constraint.ConstraintLayout>
```

MainActivity

The MainActivity class includes the following private variables at the top of the class:

```
// Text view for Hello World.  
private TextView mHelloTextView;  
// array of color names, these match the color resources in color.xml  
private String[] mColorArray = {"red", "pink", "purple", "deep_purple",  
    "indigo", "blue", "light_blue", "cyan", "teal", "green",  
    "light_green", "lime", "yellow", "amber", "orange", "deep_orange",  
    "brown", "grey", "blue_grey", "black" };
```

The MainActivity class includes the following onCreate() and onSaveInstanceState() methods:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mHelloTextView = findViewById(R.id.hello_textview);  
    // restore saved instance state (the text color)  
    if (savedInstanceState != null) {  
        mHelloTextView.setTextColor(savedInstanceState.getInt("color"));  
    }  
  
    @Override  
    public void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
        // save the current text color  
        outState.putInt("color", mHelloTextView.getCurrentTextColor());  
    }  
}
```

Task 3: Implement Button behavior

The **Change Color** button in the HelloCompat app picks one of the 20 colors from the color.xml resource file at random and sets the color of the text to that color. In this task you'll implement the behavior for Button click handler.

2.1 Add the changeButton() click handler

1. Open **activity_main.xml**, if it is not already open. Click the **Text** tab to show the XML code.
2. Click on "changeColor" in the android:onClick attribute inside the Button element.
3. Press Alt+Enter (Option+Enter on a Mac), and select **Create onClick event handler**.
4. Choose **MainActivity** and click **OK**.

This creates a placeholder method stub for the changeColor() method in MainActivity:

```
public void changeColor(View view) {  
}
```

2.2 Implement the Button action

1. Switch to **MainActivity**.
2. In the changeColor() method, create a random number object by using the Random class (a Java class) to generate simple random numbers.

```
Random random = new Random();
```

3. Use the random instance to pick a random color from the mColorArray array:

```
String colorName = mColorArray[random.nextInt(20)];
```

The nextInt() method with the argument 20 gets another random integer between 0 and 19. You use that integer as the index of the array to get a color name.

-
4. Get the resource identifier (an integer) for the color name from the resources:

```
int colorResourceName = getResources().getIdentifier(colorName,  
        "color", getApplicationContext().getPackageName());
```

When your app is compiled, the Android system converts the definitions in your XML files into resources with internal integer IDs. There are separate IDs for both the names and the values. This line matches the color strings from the colorName array with the corresponding color name IDs in the XML resource file. The getResources() method gets all the resources for your app. The getIdentifier() method looks up the color name (the string) in the color resources ("color") for the current package name.

5. Get the integer ID for the actual color from the resources and assign it to a colorRes variable, and use the getTheme() method to get the theme for the current application context.

```
int colorRes =  
    getResources().getColor(colorResourceName, this.getTheme());
```

The getResources() method gets the set of resources for your app, and the getColor() method retrieves a specific color from those resources by the ID of the color name. However, getColor() has a red underlined highlight.

If you point at getColor(), Android Studio reports: "Call requires API 23 (current min is 15)". Because your minSdkVersion is 15, you get this message if you try to use any APIs that were introduced after API 15. You can still compile your app, but because this version of getColor() is not available on devices prior to API 23, your app will crash when the user taps the **Change Color** button.

At this stage you could check for the platform version and use the right version of getColor() depending on where the app is running. A better way to support both older and newer Android APIs without warnings is to use one of the compatibility classes in the support library.

6. Change the colorResassignment line to use the ContextCompat class:

```
int colorRes = ContextCompat.getColor(this, colorResourceName);
```

ContextCompat provides many compatibility methods to address API differences in the application context and app resources. The getColor() method in ContextCompat takes two arguments: the current context (here, the Activity instance, this), and the name of the color.

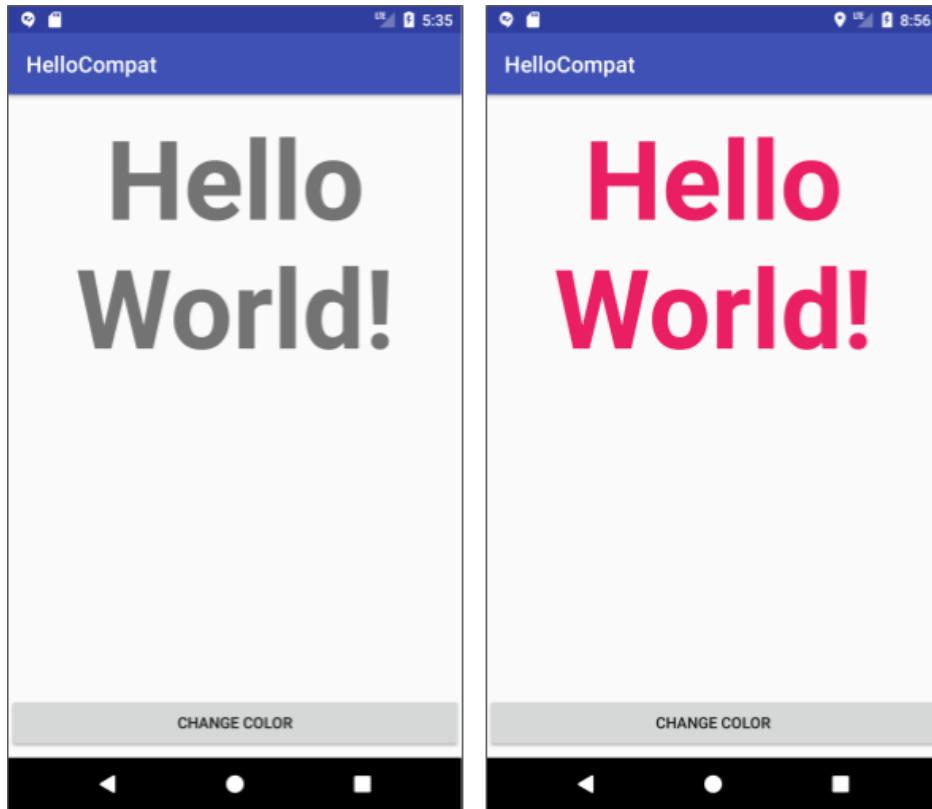
The implementation of this method in the support library hides the implementation differences in different versions of the API. You can call this method regardless of your compile SDK or minimum SDK versions with no warnings, errors, or crashes.

1. Set the color of the TextView to the color resource ID:

```
mHelloTextView.setTextColor(colorRes);
```

2. Run the app on a device or emulator, and click the **Change Color** Button.

The **Change Color** Button should now change the color of the text in the app, as shown below.



Solution code

MainActivity solution

The following is the changeColor() click handler in MainActivity:

```
/**  
 * This method handles the click of the Change Color button by  
 * picking a random color from a color array.  
 *  
 * @param view The view that was clicked
```

```
/*
public void changeColor(View view) {
    // Get a random color name from the color array (20 colors).
    Random random = new Random();
    String colorName = mColorArray[random.nextInt(20)];

    // Get the color identifier that matches the color name.
    int colorResourceName = getResources().getIdentifier(colorName,
            "color", getApplicationContext().getPackageName());

    // Get the color ID from the resources.
    int colorRes = ContextCompat.getColor(this, colorResourceName);

    // Set the text color.
    mHelloTextView.setTextColor(colorRes);
}
```

Android Studio project

Android Studio project: [HelloCompat](#)

Coding challenge

Note: All coding challenges are optional and are not prerequisites for later lessons.

Challenge: Rather than using ContextCompat for to get the color resource, use a test of the values in the [Build](#) class to perform a different operation if the app is running on a device that supports a version of Android older than API 23.

Summary

Installing the Android Support Library:

- Use the SDK Manager to install the Android Support Repository. Choose **Tools > Android**

> **SDK Manager**, click the **SDK Tools** tab, and expand **Support Repository**.

-
- If **Installed** appears in the Status column for **Android Support Repository**, click **Cancel**; if **Not installed** or **Update Available** appears, click the checkbox. A download icon should appear next to the checkbox. Click **OK**.

Android uses three directives to indicate how your app should behave for different API versions:

- `minSdkVersion`: the minimum API version your app supports.
- `compileSdkVersion`: the API version your app should be compiled with.
- `targetSdkVersion`: the API version your app was designed for.

To manage dependencies in your project:

- Expand **Gradle Scripts** in the **Project > Android** pane, and open the **build.gradle (Module: app)** file.
- You can add dependencies in the **dependencies** section.

The `ContextCompat` class provides methods for compatibility with context and resource-related methods for both old and new API levels.

Related concept

The related concept documentation is in [3.3: The Android Support Library](#).

Learn more

Android Studio documentation:

- [Android Studio User Guide](#)

Android developer documentation:

-
- [Android Support Library](#) (introduction)
 - [Support Library Setup](#)
 - [Support Library Features](#)
 - [Supporting Different Platform Versions](#)
 - [Package Index](#) (all API packages that start with android.support)

Other:

- [Picking your compileSdkVersion, minSdkVersion, and targetSdkVersion](#)
- [Understanding the Android Support Library](#)
- [All the Things Compat](#)

Homework

Run an app

Open the [HelloCompat](#) app you created in the practical on using support libraries.

1. Set a debugger breakpoint on the line in the changeColor() method that actually changes the color:

```
int colorRes = ContextCompat.getColor(this, colorResourceName);
```

2. Run the app in debug mode on a device or emulator that's running an API version 23 or newer. Click **Step Into** to step into the getColor() method and follow the method calls deeper into the stack. Examine how the ContextCompat class determines how to get the color from the resources, and which other framework classes it uses.

Some classes may produce a warning that the "source code does not match the bytecode." Click **Step Out** to return to a known source file, or keep clicking **Step Into** until the debugger returns on its own.

3. Repeat the previous step for a device or emulator running an API version older than 23. Note the different paths that the framework takes to accomplish getting the color.

Answer these questions

Question 1

Which class appears when you *first Step Into* the ContextCompat.getColor()method? Choose one:

- MainActivity
- ContextCompat
- AppCompatActivity
- Context

Question 2

In the class that appears, which statement is executed if the build version is API version 23 or newer? Choose one:

- return context.getColor(id);
- return context.getResources().getColor(id);
- throw new IllegalArgumentException("permission is null");
- return mResources == null ? super.getResources() : mResources;

Question 3

If you change the ContextCompat.getColor()method back to the getColor()method, what will happen when you run the app? Choose one:

- If your minSdkVersion is 15, the word getColor is underlined in red in the code editor. Hover your pointer over it, and Android Studio reports "Call requires API 23 (current min is 15)".
- The app will run without error on emulators and devices using API 23 or newer.
- The app will crash when the user taps **Change Color** if the emulator or device is using API 17.
- All of the above.

Submit your app for grading

Guidance for graders

No app to submit for this homework assignment.

End of Unit