

**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

# MỤC LỤC

CHƯƠNG 1.	Làm quen.....	3
Bài 1)	Tạo ứng dụng đầu tiên .....	3
1.1)	Android Studio và Hello World .....	3
1.2)	Giao diện người dùng tương tác đầu tiên .....	27
1.3)	Trình chỉnh sửa bố cục .....	59
1.4)	Văn bản và các chế độ cuộn .....	59
1.5)	Tài nguyên có sẵn.....	59
Bài 2)	Activities .....	59
2.1)	Activity và Intent .....	59
2.2)	Vòng đời của Activity và trạng thái.....	59
2.3)	Intent ngầm định .....	59
Bài 3)	Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ.....	59
3.1)	Trình gỡ lỗi.....	59
3.2)	Kiểm thử đơn vị .....	59
3.3)	Thư viện hỗ trợ .....	59
CHƯƠNG 2.	Trải nghiệm người dùng.....	60
Bài 1)	Tương tác người dùng.....	60
1.1)	Hình ảnh có thể chọn .....	60
1.2)	Các điều khiển nhập liệu .....	60
1.3)	Menu và bộ chọn .....	60
1.4)	Điều hướng người dùng.....	60
1.5)	RecyclerView.....	60
Bài 2)	Trải nghiệm người dùng thú vị .....	60
2.1)	Hình vẽ, định kiểu và chủ đề .....	60
2.2)	Thẻ và màu sắc .....	60

2.3)	Bố cục thích ứng .....	60
Bài 3)	Kiểm thử giao diện người dùng.....	60
3.1)	Espresso cho việc kiểm tra UI.....	60
CHƯƠNG 3.	Làm việc trong nền .....	60
Bài 1)	Các tác vụ nền.....	60
1.1)	AsyncTask .....	60
1.2)	AsyncTask và AsyncTaskLoader.....	60
1.3)	Broadcast receivers .....	60
Bài 2)	Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền.....	60
2.1)	Thông báo.....	60
2.2)	Trình quản lý cảnh báo.....	60
2.3)	JobScheduler .....	60
CHƯƠNG 4.	Lưu dữ liệu người dùng .....	61
Bài 1)	Tùy chọn và cài đặt .....	61
1.1)	Shared preferences .....	61
1.2)	Cài đặt ứng dụng .....	61
Bài 2)	Lưu trữ dữ liệu với Room .....	61
2.1)	Room, LiveData và ViewModel.....	61
2.2)	Room, LiveData và ViewModel.....	61

3.1) Trình gowx loi .....

## CHƯƠNG 1. LÀM QUEN

### Bài 1) Tạo ứng dụng đầu tiên

#### 1.1) Android Studio và Hello World

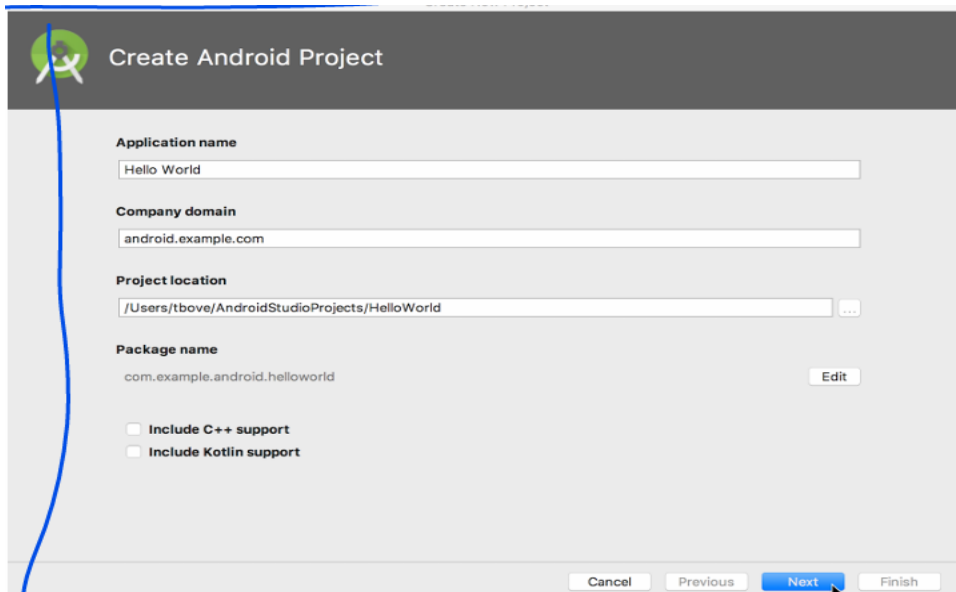
### Giới thiệu

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

## Những gì Bạn nên biết

Bạn nên có khả năng:

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



## Những gì Bạn sẽ cần:

- Một máy tính chạy Windows hoặc Linux, hoặc một Mac chạy macOS. Xem trang tải xuống Android Studio để biết yêu cầu hệ thống cập nhật.
- Truy cập Internet hoặc một phương pháp thay thế để tải các cài đặt mới nhất của Android Studio và Java lên máy tính của bạn.

## Những gì bạn sẽ học

- Cách cài đặt và sử dụng IDE Android Studio.
- Cách sử dụng quy trình phát triển để xây dựng ứng dụng Android.

- Cách tạo một dự án Android từ một mẫu.
- Cách thêm thông điệp ghi lại vào ứng dụng của bạn để phục vụ mục đích gỡ lỗi.

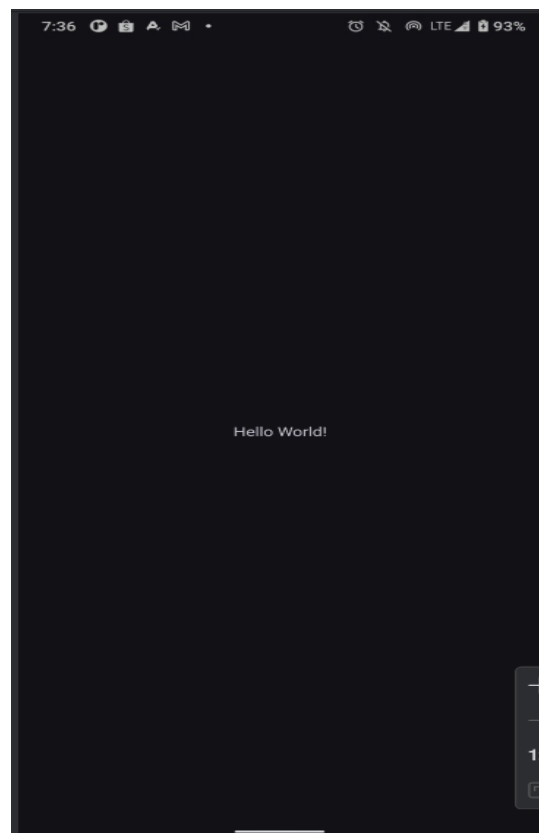
## Những gì bạn sẽ làm

- Cài đặt môi trường phát triển **Android Studio**.
- Tạo một trình giả lập (thiết bị ảo) để chạy ứng dụng của bạn trên máy tính.
- Tạo và chạy ứng dụng **Hello World** trên các thiết bị ảo và vật lý.
- Khám phá cấu trúc dự án.
- Tạo và xem các thông điệp ghi lại từ ứng dụng của bạn.
- Khám phá tệp **AndroidManifest.xml**

## Tổng quan 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” lên màn hình của thiết bị Android ảo hoặc thiết bị vật lý.

Đây là ứng dụng sau khi hoàn thành:



## 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, kiểm thử và tối ưu hiệu suất, giúp việc phát triển ứng dụng trở nên nhanh chóng và dễ dàng hơn. Bạn có thể kiểm thử ứng dụng của mình trên nhiều trình giả lập được cấu hình sẵn hoặc trên thiết bị di động của chính mình, xây dựng ứng dụng để đưa vào sản xuất và xuất bản trên cửa hàng Google Play.

**Lưu ý:** Android Studio liên tục được cải tiến. 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, cũng như cho Mac chạy macOS. Bộ OpenJDK (Java Development Kit) mới nhất được tích hợp sẵn trong Android Studio.

Để bắt đầu sử dụng Android Studio, trước tiên hãy kiểm tra **yêu cầu hệ thống** để đảm bảo rằng hệ thống của bạn đáp ứng được các yêu cầu này. Quá trình cài đặt tương tự trên tất cả các nền tảng. Mọi điểm khác biệt sẽ được ghi chú bên dưới.

1. Truy cập trang Android developers và làm theo hướng dẫn tải và cài đặt Android Studio.
2. Chấp nhận các cấu hình mặc định cho tất cả các bước và đảm bảo rằng tất cả các thành phần đều được chọn để cài đặt.
3. Sau khi hoàn tất cài đặt, **Setup Wizard** sẽ tải xuống và cài đặt một số thành phần bổ sung, bao gồm **Android SDK**. Hãy kiên nhẫn, vì quá trình này có thể mất một chút thời gian tùy thuộc vào tốc độ Internet của bạn, và một số bước có thể trông giống nhau.
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.

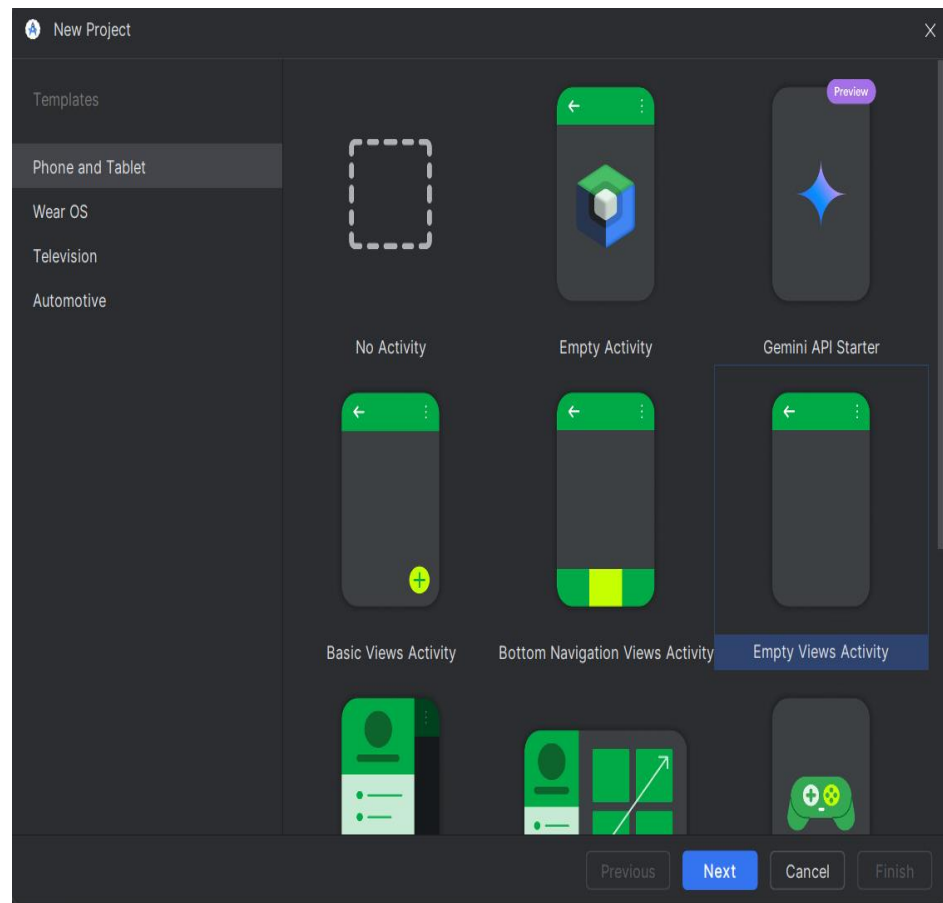
**Xử lý sự cố:** Nếu bạn gặp vấn đề trong quá trình cài đặt, hãy kiểm tra **ghi chú phát hành của Android Studio** hoặc nhờ sự trợ giúp từ giảng viên của bạn.

## Nhiệm vụ 2: Tạo ứng dụng Hello World

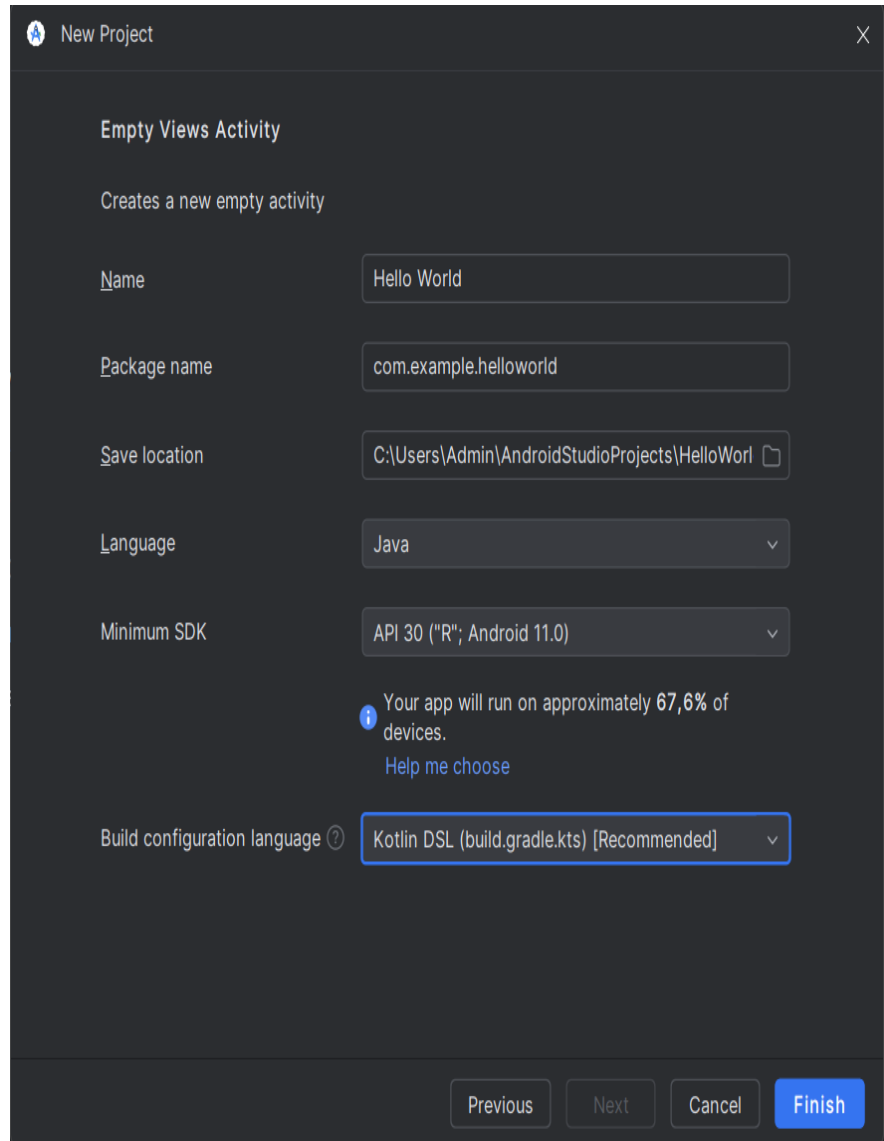
Trong nhiệm vụ này, bạn sẽ tạo một ứng dụng hiển thị **"Hello World"** để xác minh rằng **Android Studio** đã được cài đặt đúng cách và tìm hiểu những kiến thức cơ bản về phát triển ứng dụng với **Android Studio**.

## 2.1 Tạo dự án ứng dụng

1. Mở Android Studio nếu chưa mở.
2. Trong cửa sổ chính **Welcome to Android Studio**, nhấp vào **Start a new Android Studio project**.
3. Trong cửa sổ **New Project** ta chọn **Phone and Tablet** và chọn mẫu **Empty Views Activity**, nhấn **Next**.



4. Nhập **Hello World** vào ô **Name**, chọn ngôn ngữ lập trình ở ô **Language**.
5. Xác minh rằng **Save location** mặc định là nơi bạn muốn lưu ứng dụng **Hello World** và các dự án Android Studio khác, hoặc đổi nó đến thư mục mà bạn muốn.
6. Chấp nhận **Package name** mặc định **com.example.helloworld** hoặc tạo mới nếu bạn muốn. Nếu bạn không có kế hoạch phát hành ứng dụng của mình, hãy chấp nhận giá trị mặc định. Lưu ý rằng việc thay đổi **tên package** của ứng dụng sau này sẽ tốn thêm công sức.



## 7. Nhấn **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 **Configure your build** dành cho nhà phát triển để biết thông tin chi tiết.

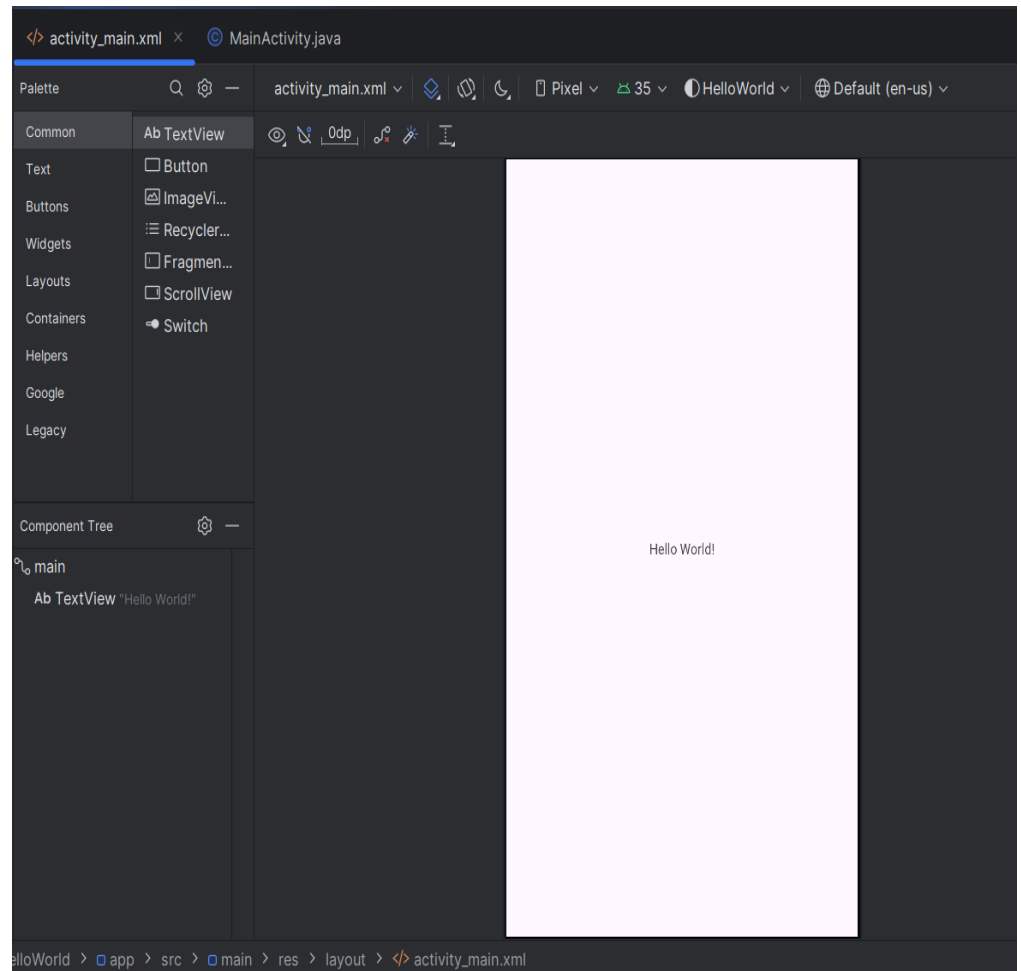
Bạn cũng có thể thấy thông báo **"Tip of the Day"** hiển thị các phím tắt và mẹo hữu ích khác. Nhấp vào **Close** để đóng thông báo.

Sau đó, trình chỉnh sửa của **Android Studio** sẽ xuất hiện. Hãy thực hiện các bước tiếp theo:

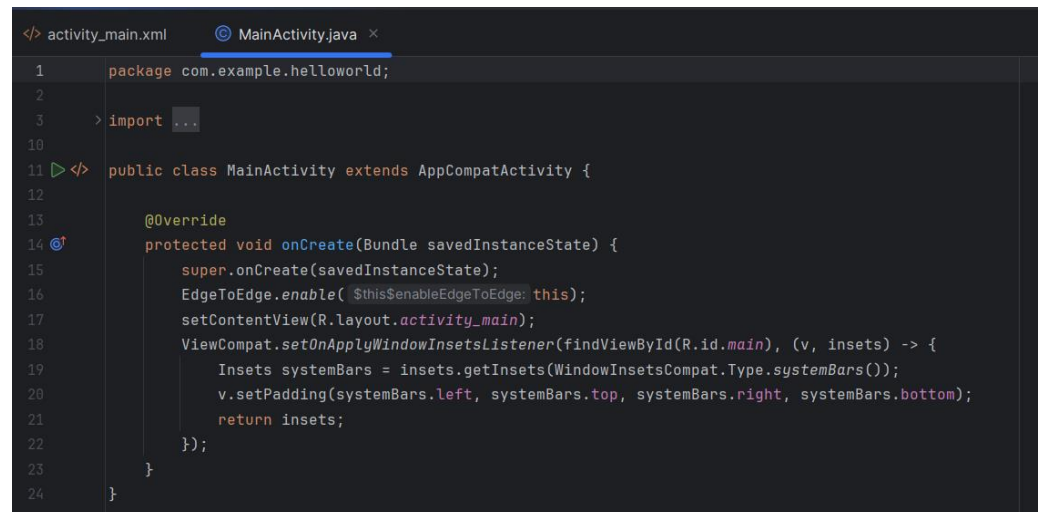
1. Nhấp vào tab **activity\_main.xml** để mở **layout editor**.



2. Nhấp vào tab **Design** (nếu chưa được chọn) để hiển thị giao diện đồ họa của **layout**, giống như hình minh họa bên dưới.



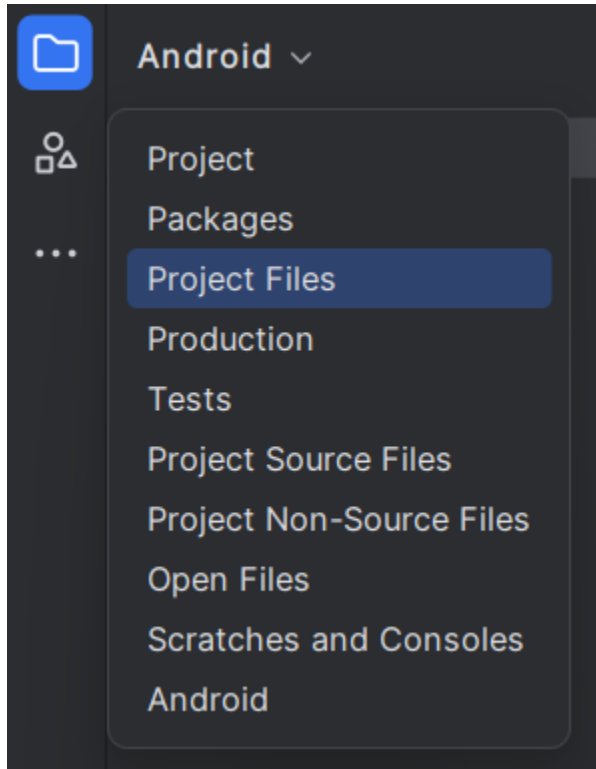
3. Nhấp vào tab **MainActivity.java** để mở **code editor** và xem mã nguồn của ứng dụng.



## 2.2 Khám phá Project > Android Pane

Trong phần thực hành này, bạn sẽ tìm hiểu cách dự án được tổ chức trong **Android Studio**.

1. Nếu **Project pane** chưa hiển thị, hãy nhấp vào tab **Project** trong cột tab dọc bên trái của cửa sổ **Android Studio**.
2. Để xem dự án theo cấu trúc tiêu chuẩn của **Android**, hãy chọn **Android** từ menu thả xuống ở đầu **Project pane**, như hình minh họa bên dưới.

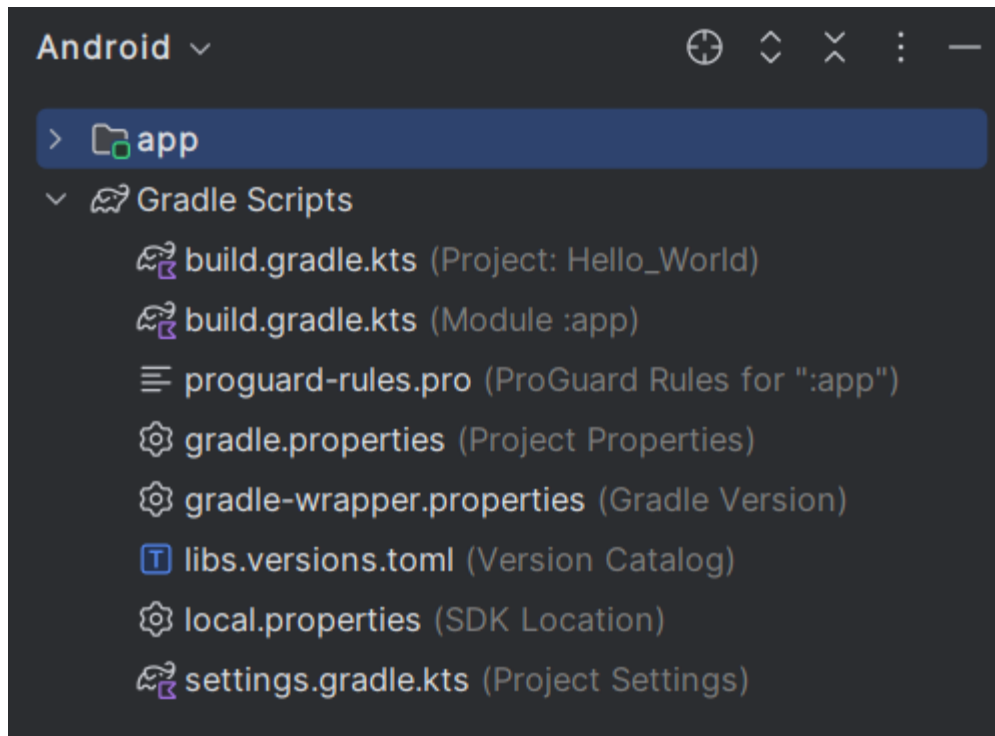


**Lưu ý:** Chương này và các chương khác sẽ gọi **Project pane** (khi được đặt ở chế độ **Android**) là **Project > Android pane**.

## 2.3 Khám phá thư mục Gradle Scripts

Hệ thống **Gradle build** trong **Android Studio** giúp bạn dễ dàng thêm **các thư viện bên ngoài** hoặc **các module khác** vào dự án của mình dưới dạng **dependencies**.

Khi bạn tạo một dự án ứng dụng mới, **Project > Android pane** sẽ hiển thị với thư mục **Gradle Scripts** được mở rộng, như minh họa bên dưới.



Hãy làm theo các bước sau để tìm hiểu về hệ thống **Gradle** trong **Android Studio**:

1. Nếu thư mục **Gradle Scripts** chưa được mở rộng, hãy nhấp vào biểu tượng **tam giác** để mở nó.

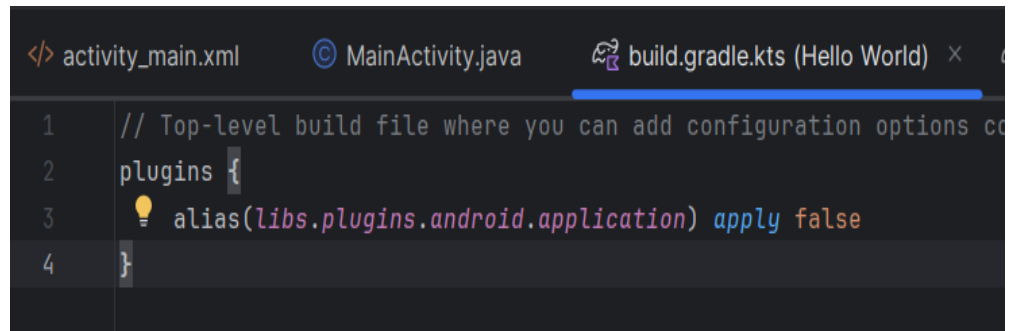
Thư mục này chứa tất cả các tệp cần thiết cho hệ thống build.

2. Tìm tệp **build.gradle (Project: HelloWorld)**.

Đây là nơi bạn sẽ tìm thấy các **tùy chọn cấu hình chung** cho tất cả các **module** trong dự án của mình. Mỗi dự án trong **Android Studio** đều có một tệp **Gradle build cấp cao nhất (top-level Gradle build file)**. Thông thường, bạn sẽ không cần thay đổi tệp này, nhưng hiểu cách nó hoạt động sẽ giúp bạn dễ dàng quản lý dependencies và cấu hình build.

Mặc định, tệp **build.gradle** cấp cao nhất sử dụng khối **buildscript** để khai báo **repositories** và **dependencies** chung cho tất cả các **module** trong dự án. Khi một **dependency** không phải là thư viện cục bộ hoặc **file tree**, Gradle sẽ tìm kiếm các tệp trong **các kho lưu trữ trực tuyến** được chỉ định trong khối **repositories** của tệp này.

Mặc định, các dự án mới trong **Android Studio** sử dụng **JCenter** và **Google Maven Repository** làm nguồn tìm kiếm thư viện.



```
</> activity_main.xml    MainActivity.java    build.gradle.kts (Hello World) x
1 // Top-level build file where you can add configuration options co
2 plugins {
3     alias(libs.plugins.android.application) apply false
4 }
```

### 3. Tìm tệp **build.gradle(Module:app)**

Ngoài tệp **build.gradle** cấp dự án, mỗi **module** cũng có một tệp **build.gradle** riêng, cho phép bạn cấu hình các cài đặt build cho từng module cụ thể (ứng dụng **HelloWorld** chỉ có một module). Việc cấu hình các cài đặt build này giúp bạn tùy chỉnh **tùy chọn đóng gói**, chẳng hạn như **thêm các loại build (build types)** và các **phiên bản sản phẩm (product flavors)**. Bạn cũng có thể **ghi đè** các cài đặt trong tệp **AndroidManifest.xml** hoặc tệp **build.gradle** cấp cao nhất.

Tệp này thường là nơi bạn chỉnh sửa khi thay đổi **cấu hình cấp ứng dụng**, chẳng hạn như khai báo thư viện trong phần **dependencies**. Bạn có thể khai báo **thư viện phụ thuộc (library dependency)** bằng nhiều cách khác nhau, tùy vào **cấu hình phụ thuộc (dependency configuration)** bạn sử dụng. Mỗi cấu hình sẽ cung cấp cho **Gradle** các hướng dẫn khác nhau về cách sử dụng thư viện. Ví dụ: Dòng lệnh sau trong **dependencies**:  
`implementation fileTree(dir: 'libs', include: ['*.jar'])` sẽ thêm tất cả các tệp **“.jar”** nằm trong thư mục **libs** vào ứng dụng dưới dạng dependency.

Dưới đây là nội dung của tệp **build.gradle (Module: app)** cho ứng dụng HelloWorld:

```

1  plugins {
2      alias(libs.plugins.android.application)
3  }
4
5  android {
6      namespace = "com.example.helloworld"
7      compileSdk = 35
8
9      defaultConfig {
10         applicationId = "com.example.helloworld"
11         minSdk = 30
12         targetSdk = 35
13         versionCode = 1
14         versionName = "1.0"
15
16         testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
17     }
18
19     buildTypes {
20         release {
21             isMinifyEnabled = false
22             proguardFiles(
23                 getDefaultProguardFile("name: "proguard-android-optimize.txt"),
24                 "proguard-rules.pro"
25             )
26         }
27     }

```

```

28         compileOptions {
29             sourceCompatibility = JavaVersion.VERSION_11
30             targetCompatibility = JavaVersion.VERSION_11
31         }
32     }
33
34     dependencies {
35
36         implementation(libs.appcompat)
37         implementation(libs.material)
38         implementation(libs.activity)
39         implementation(libs.constraintlayout)
40         testImplementation(libs.junit)
41         androidTestImplementation(libs.ext.junit)
42         androidTestImplementation(libs.espresso.core)
43     }

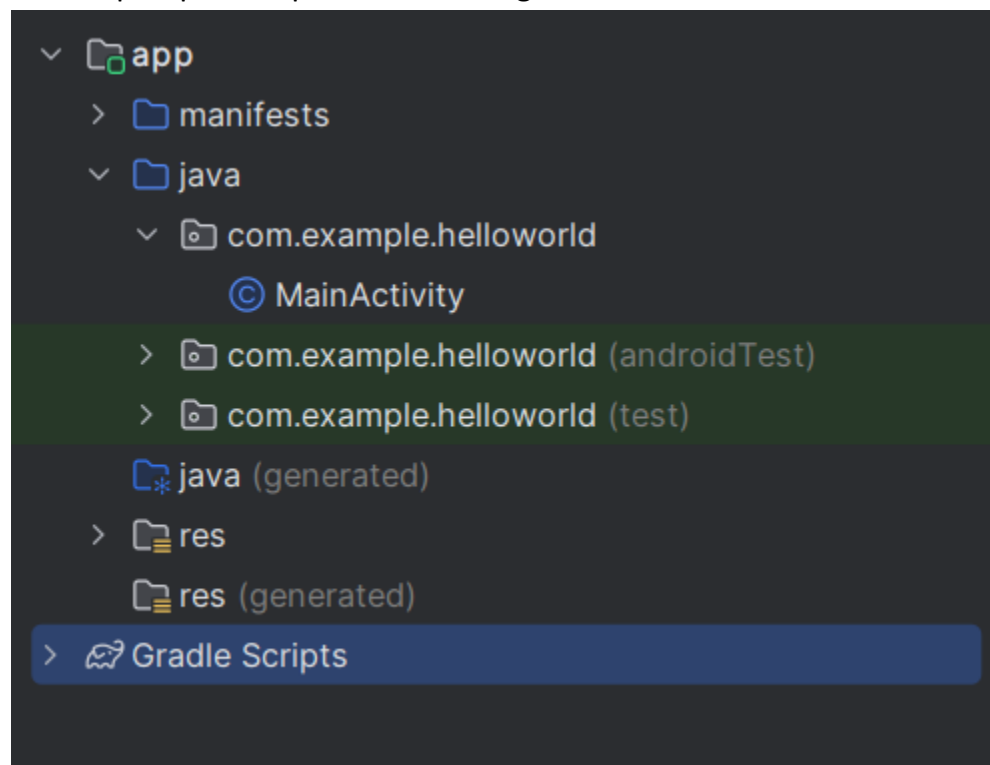
```

4. Nhấp vào **biểu tượng tam giác** bên cạnh **Gradle Scripts** để đóng thư mục này.

## 2.4 Khám phá thư mục app và res

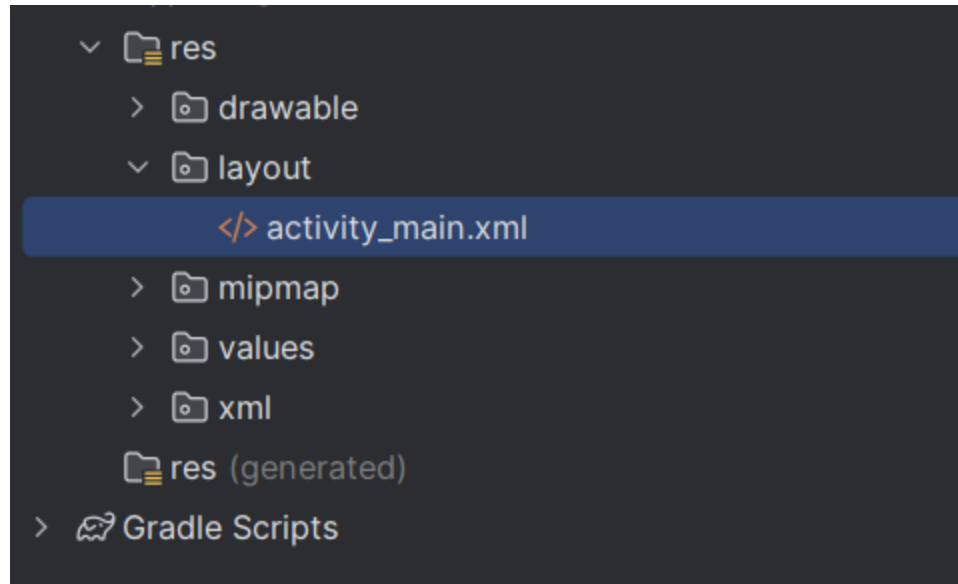
Tất cả các mã và tài nguyên cho ứng dụng đều nằm trong các thư mục ứng dụng và res.

1. Mở rộng thư mục **app**, thư mục **Java** và thư mục **com.example.android.helloworld** để xem tệp java **MainActivity**. Nhấp đúp vào tệp sẽ mở nó trong trình chỉnh sửa mã.



Thư mục **java** chứa các tệp lớp Java, được sắp xếp trong ba thư mục con, như minh họa trong hình trên. Thư mục **com.example.hello.helloworld** (hoặc tên miền mà bạn đã chỉ định) chứa **tất cả các tệp của một gói ứng dụng**. Hai thư mục còn lại được sử dụng cho **kiểm thử (testing)** và sẽ được đề cập trong một bài học khác. Đối với ứng dụng **Hello World**, chỉ có một gói (**package**) duy nhất và nó chứa tệp **MainActivity.java**. Màn hình đầu tiên mà người dùng nhìn thấy khi khởi chạy ứng dụng – hay còn gọi là **Activity đầu tiên** – thường được đặt tên là **MainActivity** (trong **Project > Android pane**, phần mở rộng tệp **.java** có thể bị ẩn đi).

2. Mở rộng thư mục **res** và thư mục **layout** và nhấp đúp vào tệp **activity\_main.xml** để mở nó trong trình soạn thảo bố cục.



Thư mục **res** chứa các tài nguyên như bố cục (layouts), chuỗi (strings) và hình ảnh (images). Một **Activity** thường được liên kết với một bố cục giao diện người dùng (UI views) được định nghĩa dưới dạng tệp XML. Tệp này thường được đặt tên theo **Activity** tương ứng.

## 2.5 Khám phá thư mục manifests

Thư mục **manifests** chứa các tệp cung cấp thông tin quan trọng về ứng dụng của bạn cho hệ thống Android. Hệ thống cần có những thông tin này trước khi có thể chạy bất kỳ đoạn mã nào của ứng dụng.

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

Tệp **AndroidManifest.xml** mô tả tất cả các thành phần của ứng dụng Android. Mọi thành phần trong ứng dụng, chẳng hạn như mỗi **Activity**, đều phải được khai báo trong tệp XML này. Trong các bài học tiếp theo, bạn sẽ chỉnh sửa tệp này để thêm các tính năng và quyền truy cập cho ứng dụng. Để tìm hiểu thêm, hãy xem **App Manifest Overview**.

## Nhiệm vụ 3: Sử dụng thiết bị ảo(Trình giả lập)

Trong nhiệm vụ này, bạn sẽ sử dụng **Android Virtual Device (AVD) Manager** để tạo một thiết bị ảo (còn gọi là **emulator**) nhằm mô phỏng cấu hình của một thiết bị Android cụ

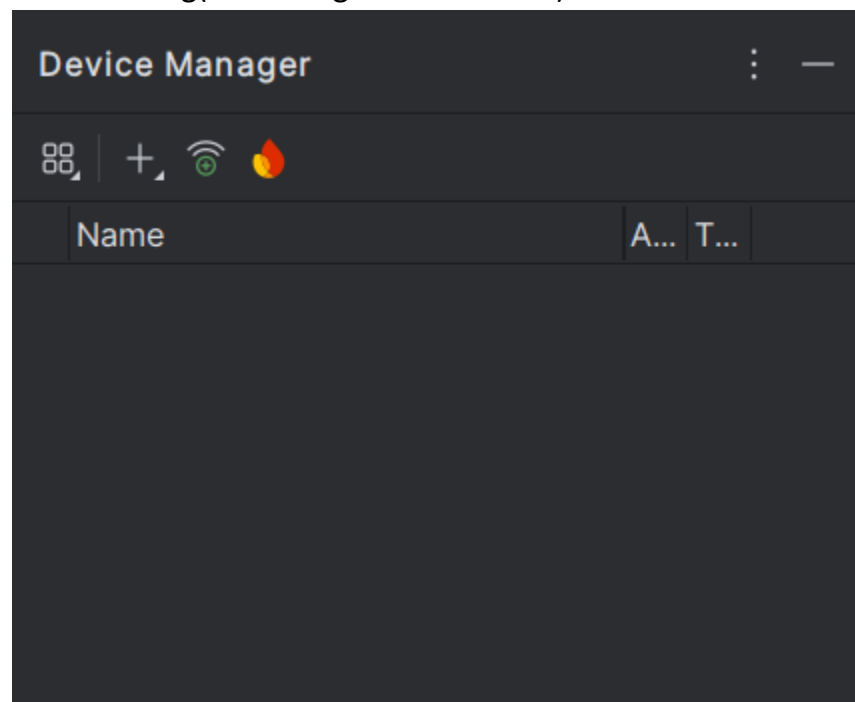
thể. Sau đó, bạn sẽ chạy ứng dụng trên thiết bị ảo này. Lưu ý rằng **Android Emulator** có các yêu cầu bổ sung so với yêu cầu hệ thống cơ bản của **Android Studio**.

Khi sử dụng AVD Manager, bạn xác định các đặc điểm phần cứng của thiết bị, API level, 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 một thiết bị ảo Android(AVD)

Để chạy **emulator** trên máy tính, bạn cần tạo một cấu hình mô tả thiết bị ảo.

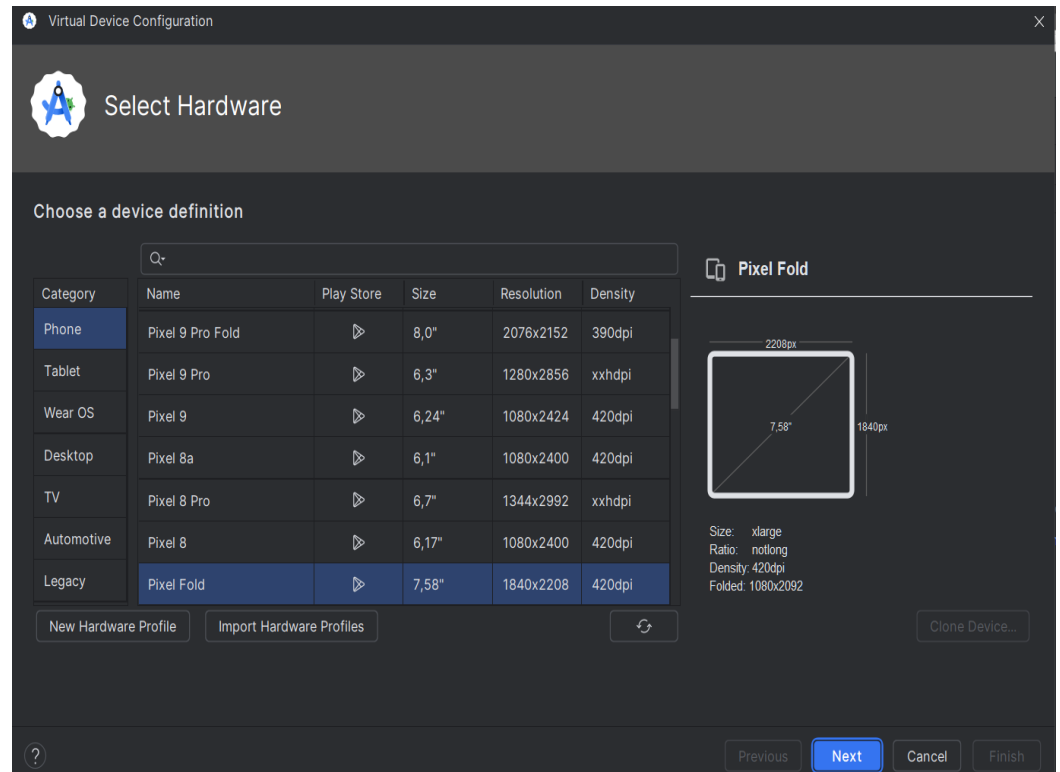
1. Trong **Android Studio**, chọn **Tools >Device Manager**, hoặc nhấp vào biểu tượng **Device Manager** trên thanh công cụ. Màn hình **Device Manager** sẽ xuất hiện. Nếu bạn đã tạo các thiết bị ảo trước đó, chúng sẽ được hiển thị. Nếu chưa có thiết bị nào, danh sách sẽ trống (như trong hình bên dưới).



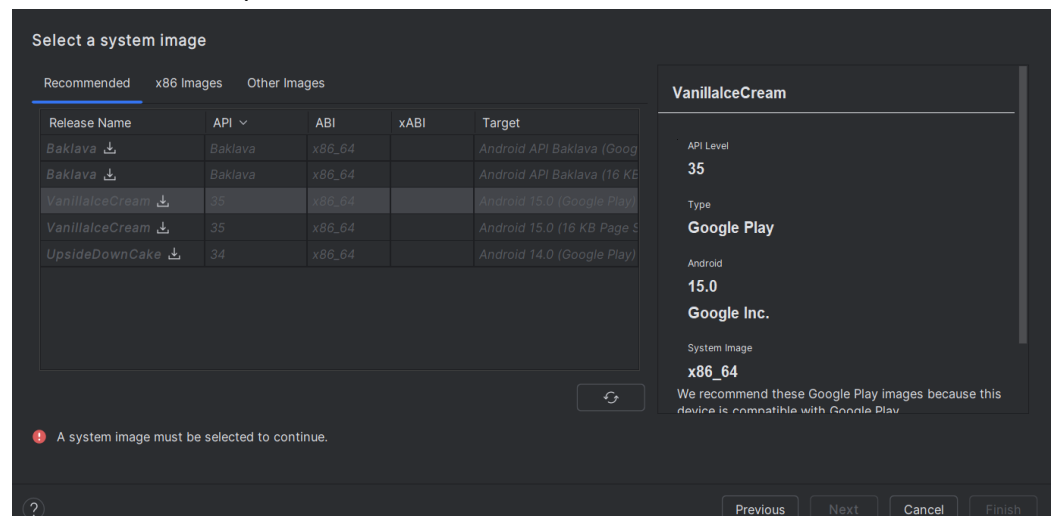
2. Nhấp vào dấu +. Chọn **Create Virtual Device**. Cửa sổ **Select Hardware** xuất hiện, hiển thị danh sách các thiết bị phần cứng được cấu hình sẵn. Đối với mỗi thiết bị, bảng sẽ hiển thị các cột thông tin sau: Kích thước đường chéo màn hình(**Size**), Độ phân



giải màn hình tính bằng pixel(**Resolution**), Mật độ điểm ảnh (**Density**).



3. Chọn một thiết bị, chẳng hạn như **Medium Phone** hoặc **Pixel Fold**, sau đó nhấp vào **Next**. Màn hình **System Image** sẽ xuất hiện.
4. Nhấp vào tab **Recommended** nếu nó chưa được chọn, sau đó chọn phiên bản Android để chạy trên thiết bị ảo (chẳng hạn như **VanillaIceCream**).



Có nhiều phiên bản Android hơn so với những phiên bản hiển thị trong tab **Recommended**. Hãy kiểm tra thêm trong các tab **x86 Images** và **Other Images** để xem danh sách đầy đủ.

Nếu có liên kết **Download** bên cạnh một phiên bản hệ điều hành mà bạn muốn sử dụng, điều đó có nghĩa là nó chưa được cài đặt. Nhấp vào liên kết **Download** để bắt đầu tải xuống. Khi quá trình tải xuống hoàn tất, nhấp vào **Finish**.

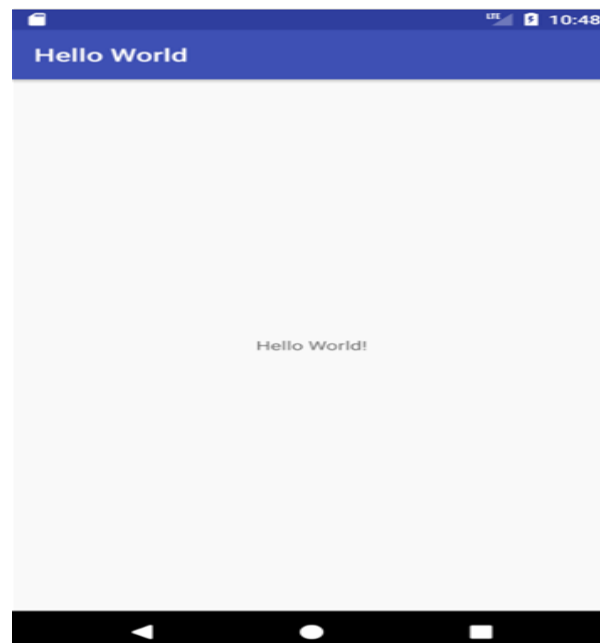
### 3.2 Chạy ứng dụng trên thiết bị ảo

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

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

Trình giả lập (**emulator**) sẽ khởi động giống như một thiết bị thật. Tùy vào tốc độ của máy tính, quá trình này có thể mất một chút thời gian. Sau khi ứng dụng được biên dịch, **Android Studio** sẽ tải ứng dụng lên trình giả lập và chạy nó.

Bạn sẽ thấy ứng dụng **Hello World** hiển thị như trong hình minh họa bên dưới.



**Mẹo:** Khi kiểm thử trên thiết bị ảo, bạn nên khởi động nó một lần ngay từ đầu phiên làm việc. Không nên đóng trình giả lập cho đến khi hoàn tất kiểm thử ứng dụng, để tránh mất thời gian khởi động lại thiết bị. Để đóng thiết bị ảo, nhấp vào nút **X** ở góc trên của trình giả lập, chọn **Quit** từ menu, hoặc nhấn **Control + Q** trên Windows hoặc **Command + Q** trên macOS.

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

Trong nhiệm vụ cuối cùng này, bạn sẽ chạy ứng dụng trên một thiết bị di động thực tế, chẳng hạn như điện thoại hoặc máy tính bảng. Bạn nên luôn kiểm thử ứng dụng trên cả thiết bị ảo và thiết bị thực.

Bạn cần chuẩn bị:

- Một thiết bị **Android** (điện thoại hoặc máy tính bảng).
- Cáp dữ liệu để kết nối thiết bị Android với máy tính qua cổng **USB**.
- Nếu sử dụng **Linux hoặc Windows**, bạn có thể cần thực hiện một số bước bổ sung để chạy ứng dụng trên thiết bị phần cứng. Xem tài liệu **Using Hardware Devices** để biết hướng dẫn chi tiết. Nếu sử dụng **Windows**, có thể cần cài đặt driver USB phù hợp. Xem tài liệu **OEM USB Drivers** để tải driver cho thiết bị của bạn.

### 4.1 Bật chế độ gỡ lỗi USB (USB Debugging)


Để **Android Studio** có thể giao tiếp với thiết bị của bạn, bạn cần bật **USB Debugging** trong phần **Developer options** trên thiết bị Android.

**Trên Android 4.2 trở lên**, mục **Developer options** bị ẩn theo mặc định. Để hiển thị và bật **USB Debugging**, làm theo các bước sau:

1. **Mở cài đặt (Settings)** trên thiết bị của bạn. Tìm kiếm và chọn **About phone** (Giới thiệu về điện thoại). Nhấn vào **Build number** (Số bản dựng) **7 lần liên tiếp** để kích hoạt chế độ nhà phát triển.
2. Quay lại màn hình trước (**Settings / System**), mục **Developer options** (Tùy chọn nhà phát triển) sẽ xuất hiện trong danh sách. Nhấn vào **Developer options**.
3. Chọn **USB Debugging**.

### 4.2 Chạy ứng dụng trên thiết bị thực

Bây giờ bạn có thể kết nối thiết bị và chạy ứng dụng trực tiếp từ **Android Studio**.

1. **Kết nối** điện thoại hoặc máy tính bảng của bạn với máy tính bằng **cáp USB**.
2. Nhấn nút **Run**  trên thanh công cụ của **Android Studio**. Cửa sổ **Select Deployment Target** xuất hiện, hiển thị danh sách các thiết bị ảo và thiết bị thực đã kết nối.
3. Chọn **thiết bị của bạn**, sau đó nhấn **OK**.

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

## Xử lý sự cố

Nếu **Android Studio** không nhận diện được thiết bị của bạn, hãy thử các bước sau:

1. **Rút cáp USB** và cắm lại thiết bị.
2. **Khởi động lại Android Studio**.

Nếu máy tính vẫn không nhận thiết bị hoặc báo lỗi "unauthorized" (chưa được ủy quyền), làm như sau:

1. Rút **cáp USB**.
2. Trên điện thoại, mở **Settings** (Cài đặt) > **Developer Options** (Tùy chọn nhà phát triển).
3. Nhấn vào **Revoke USB Debugging authorizations** (Thu hồi quyền gỡ lỗi USB).
4. Kết nối lại thiết bị với máy tính bằng cáp USB.
5. Khi xuất hiện thông báo trên điện thoại, hãy **chấp nhận quyền gỡ lỗi USB**.

Bạn có thể cần cài đặt **driver USB** phù hợp cho thiết bị của mình. Xem tài liệu **Using Hardware Devices** để biết hướng dẫn chi tiết.

## Nhiệm vụ 5: Thay đổi cấu hình Gradle của ứng dụng

Trong nhiệm vụ này, bạn sẽ thực hiện một số thay đổi trong tệp **build.gradle(Module:app)** để hiểu cách chỉnh sửa cấu hình ứng dụng và đồng bộ chúng với **Android Studio**.

### 5.1 Thay đổi phiên bản SDK tối thiểu của ứng dụng

Làm theo các bước sau:

1. **Mở rộng thư mục Gradle Scripts** trong **Android Studio** (nếu nó chưa mở). Nhấp đúp vào tệp **build.gradle(Module:app)** để mở trong trình chỉnh sửa mã (**code editor**).

2. Tìm khối **defaultConfig** và thay đổi giá trị của **minSdkVersion** từ 15 thành 17, như ví dụ sau:

```
android {  
    namespace = "com.example.helloworld"  
    compileSdk = 35  
  
    defaultConfig {  
        applicationId = "com.example.helloworld"  
        minSdk = 30  
        targetSdk = 35  
        versionCode = 1  
        versionName = "1.0"  
  
        testInstrumentationRunner = "androidx.test.runner."  
    }  
}
```

Trình chỉnh sửa mã (**code editor**) sẽ hiển thị một thanh thông báo ở phía trên với liên kết **Sync Now**.

## 5.2 Đồng bộ cấu hình Gradle mới

Khi bạn thay đổi các tệp cấu hình **Gradle** trong dự án, **Android Studio** yêu cầu bạn **đồng bộ hóa (sync)** các tệp dự án để nhập các thay đổi trong cấu hình xây dựng (**build configuration**) và kiểm tra lỗi có thể xảy ra khi biên dịch.

Để đồng bộ hóa các tệp dự án, nhấp vào **Sync Now** trong thanh thông báo xuất hiện sau khi bạn thực hiện thay đổi hoặc nhấp vào biểu tượng **Sync Project with Gradle Files** trên thanh công cụ.

**Khi hoàn tất đồng bộ hóa**, bạn sẽ thấy thông báo "**Gradle build finished**" ở góc dưới bên trái của cửa sổ **Android Studio**.

Để tìm hiểu chi tiết hơn về Gradle, hãy tham khảo tài liệu **Build System Overview** (Tổng quan về hệ thống build) và **Configuring Gradle Builds** (Cấu hình Gradle).

## Nhiệm vụ 6: Thêm câu lệnh Log vào ứng dụng

Trong nhiệm vụ này, bạn sẽ thêm các câu lệnh **Log** vào ứng dụng của mình, giúp hiển thị thông báo trong bảng **Logcat**. Các thông báo **Log** là một công cụ gỡ lỗi mạnh mẽ, cho phép bạn kiểm tra giá trị của biến, theo dõi luồng thực thi của chương trình và báo cáo lỗi (**exception**).

### 6.1 Xem bảng Logcat

Để xem bảng **Logcat**, nhấp vào tab **Logcat** ở phía dưới cửa sổ **Android Studio**, như minh họa trong hình bên dưới.



### 6.2 Thêm câu lệnh Log vào ứng dụng

Các câu lệnh **Log** trong mã nguồn ứng dụng sẽ hiển thị thông báo trong bảng **Logcat**. Ví dụ:

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

Các phần của một câu lệnh Log:

- **Log**: Lớp Log dùng để gửi thông điệp đến bảng **Logcat**.
- **d**: Mức **Debug** giúp lọc thông điệp hiển thị trong **Logcat**. Các mức khác gồm: **e** (**Error**) – Lỗi nghiêm trọng, **w** (**Warn**) – Cảnh báo có thể gây lỗi, **i** (**Info**) – Thông tin chung.
- **"MainActivity"**: **TAG** giúp phân loại và lọc thông điệp trong **Logcat**. Thông thường, TAG sẽ là tên của **Activity** nơi log được gọi. Bạn có thể đặt TAG thành bất cứ thứ gì hữu ích cho việc debug.

**Quy ước:** Nên khai báo TAG dưới dạng hằng số trong **Activity**:

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

- **"Hello world"**: Đối số thứ hai là **thông điệp thực tế** sẽ được hiển thị trong **Logcat**.

## Các bước thực hiện:

1. **Mở ứng dụng Hello World trong Android Studio**, sau đó mở tệp MainActivity.java (hoặc MainActivity.kt).
2. **Bật tính năng tự động nhập thư viện (Auto Import):**
  - ? **Windows:** Vào File > Settings.
  - ? **macOS:** Vào Android Studio > Preferences.
  - ? Chọn Editor > General > Auto Import.
  - ? **Chọn tất cả các checkbox** và đặt Insert imports on paste thành All
3. Nhấp **Apply**, sau đó nhấp **OK** để lưu cài đặt.
4. Trong phương thức **onCreate()** của MainActivity, thêm dòng sau:

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

Phương thức onCreate() sau khi thêm Log

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
    Log.d( tag: "MainActivity", msg: "Hello World");
}
```

5. Nếu bảng **Logcat** chưa mở, nhấp vào tab **Logcat** ở phía dưới **Android Studio**.
6. Kiểm tra **tên thiết bị** (target device) và **tên gói ứng dụng** (package name) trong Logcat để đảm bảo đúng với ứng dụng của bạn.
7. Thay đổi mức **Log Level** trong Logcat: Chọn **Debug** (hoặc để **Verbose** nếu có ít thông điệp log).
8. Chạy ứng dụng.

Bạn sẽ thấy thông điệp Log xuất hiện trong **Logcat** với mức **Debug (D)**

```
2025-03-03 21:49:40.932 3929-3929 MainActivity com.example.helloworld D Hello World
```

## Coding challenge

**Lưu ý:** Tất cả các thử thách lập trình (**coding challenges**) đều là **tùy chọn** và **không bắt buộc** để tiếp tục các bài học sau.

**Thử thách:** Bây giờ bạn đã thiết lập xong môi trường và quen thuộc với quy trình phát triển cơ bản, hãy thực hiện các bước sau:

1. Tạo dự án mới trong Android Studio.
2. Hãy thay đổi dòng chữ "**Hello World**" thành "**Happy Birthday to [Tên người có sinh nhật gần đây]**".
3. (Tùy chọn) Chụp màn hình và gửi lời chúc mừng sinh nhật
4. Một cách sử dụng phổ biến của lớp **Log** là ghi lại các ngoại lệ (**Java exceptions**) khi chúng xảy ra trong chương trình của bạn. Có một số phương thức 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 trong thông điệp **Log**. Sau đó, viết mã trong ứng dụng của bạn để kích hoạt và ghi log khi xảy ra ngoại lệ.

## Tóm tắt

- Để cài đặt **Android Studio**, truy cập trang **Android Studio** và làm theo hướng dẫn để tải xuống và cài đặt.
- Khi tạo ứng dụng mới, hãy đảm bảo đặt **API 15: Android 4.0.3 IceCreamSandwich** làm **Minimum SDK**.
- Để xem cấu trúc thư mục Android của ứng dụng, nhấp vào tab **Project** ở cột bên trái, sau đó chọn **Android** trong menu thả xuống.
- Chỉnh sửa tệp **build.gradle(Module:app)** khi cần thêm thư viện mới hoặc thay đổi phiên bản thư viện trong dự án.
- Tất cả mã nguồn và tài nguyên của ứng dụng nằm trong thư mục **app** và **res**. Thư mục **java** chứa các **Activity**, bài kiểm thử và các thành phần khác viết bằng Java/Kotlin. Thư mục **res** chứa các tài nguyên như bố cục (**layouts**), chuỗi văn bản (**strings**) và hình ảnh (**images**).
- Chỉnh sửa tệp **AndroidManifest.xml** để thêm các thành phần (**components**) và quyền truy cập (**permissions**) cho ứng dụng Android. Mọi thành phần như **Activity** đều phải được khai báo trong tệp này.
- Sử dụng **Android Virtual Device (AVD) Manager** để tạo thiết bị ảo (**emulator**) và chạy ứng dụng.



- Thêm các câu lệnh **Log** vào ứng dụng để hiển thị thông báo trong **Logcat**, giúp dễ dàng gỡ lỗi.
- Để chạy ứng dụng trên thiết bị Android thực bằng **Android Studio**, bật **USB Debugging** bằng cách: Mở **Settings** > **About phone**, nhấn **Build number 7** lần liên tiếp. Quay lại màn hình **Settings**, vào **Developer options** và bật **USB Debugging**.

## Nội dung liên quan

Tài liệu liên quan có trong **1.0: Giới thiệu về Android** và **1.1: Ứng dụng Android đầu tiên của bạn**.

## 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 \(AVD\) manager](#)
- [App Manifest Overview](#)
- [Configure your build](#)
- [Log class](#)
- [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

Xây dựng và chạy ứng dụng

- ✓ Tạo một dự án Android mới từ mẫu **Empty Template**.
- ✓ Thêm các câu lệnh logging với nhiều mức log khác nhau trong phương thức **onCreate()** của **MainActivity**.
- ✓ Tạo một trình giả lập (emulator) với bất kỳ phiên bản Android nào và chạy ứng dụng trên đó.
- ✓ Sử dụng bộ lọc trong **Logcat** để tìm các câu lệnh log của bạn và điều chỉnh mức log để chỉ hiển thị **debug** hoặc **error**.

Trả lời câu hỏi sau:

**Câu 1:** Tên của tệp layout cho **MainActivity** là ?

activity\_main.xml

**Câu 2:** Tên của **string resource** xác định tên ứng dụng là?

app\_name

**Câu 3:** Bạn dùng công cụ nào để tạo trình giả lập mới?

AVD Manager

**Câu 4:** Giả sử ứng dụng của bạn bao gồm câu lệnh ghi log sau:

```
Log.i("MainActivity", "MainActivity layout is complete");
```

Bạn sẽ thấy dòng "MainActivity layout is complete" trong bảng **Logcat** nếu **Log level** được đặt thành một trong các mức sau:

- Verbose
- Info

Nộp ứng dụng để đánh giá

Đảm bảo ứng dụng của bạn có các yếu tố sau:

- **Một Activity** hiển thị dòng chữ **"Hello World"** trên màn hình.
- **Các câu lệnh Log** trong phương thức **onCreate()** của **MainActivity**.

- **Mức Log trong Logcat** được đặt để chỉ hiển thị **debug** hoặc **error**.

## 1.2) Giao diện người dùng tương tác đầu tiên.

### Part A

### Giới thiệu

Giao diện người dùng (**UI**) hiển thị trên màn hình của một thiết bị Android bao gồm một **cây phân cấp các đối tượng gọi là views** — mỗi phần tử trên màn hình đều là một **View**.

Lớp **View** đại diện cho **khối xây dựng cơ bản** của tất cả các thành phần giao diện người dùng và là **lớp cơ sở** cho các lớp cung cấp thành phần giao diện tương tác như **nút bấm (Button)**, **hộp kiểm (Checkbox)**, và **trường nhập văn bản (Text Entry Field)**.

Các lớp con của **View** được sử dụng phổ biến sẽ được mô tả trong nhiều bài học, bao gồm:

- ✓ **TextView** – Hiển thị văn bản.
- ✓ **EditText** – Cho phép người dùng nhập và chỉnh sửa văn bản.
- ✓ **Button** và các phần tử có thể nhấp khác (**RadioButton**, **CheckBox**, **Spinner**) – Cung cấp khả năng tương tác.
- ✓ **ScrollView** và **RecyclerView** – Hiển thị danh sách có thể cuộn.
- ✓ **ImageView** – Hiển thị hình ảnh.
- ✓ **ConstraintLayout** và **LinearLayout** – Chứa các phần tử **View** khác và sắp xếp vị trí của chúng.

Mã Java hiển thị và điều khiển giao diện người dùng (**UI**) được chứa trong một lớp mở rộng từ **Activity**. Một **Activity** thường được liên kết với một bố cục giao diện người dùng được định nghĩa trong tệp **XML (eXtended Markup Language)**. Tệp XML này thường được đặt tên theo **Activity** và xác định bố cục của các phần tử **View** trên màn hình.

Ví dụ, mã **MainActivity** trong ứng dụng **Hello World** hiển thị một bố cục được định nghĩa trong tệp **activity\_main.xml**, trong đó có một **TextView** hiển thị dòng chữ "**Hello World**".

Trong các ứng dụng phức tạp hơn, một **Activity** có thể thực hiện các hành động như phản hồi thao tác chạm của người dùng, vẽ nội dung đồ họa hoặc yêu cầu dữ liệu từ cơ sở dữ liệu hoặc internet. Bạn sẽ tìm hiểu thêm về lớp **Activity** trong bài học khác.

Trong bài thực hành này, bạn sẽ học cách tạo ứng dụng tương tác đầu tiên—một ứng dụng cho phép người dùng tương tác. Bạn sẽ tạo một ứng dụng sử dụng mẫu **Empty Activity**, đồng thời học cách sử dụng **layout editor** để thiết kế bố cục cũng như chỉnh sửa bố cục trong XML. Bạn cần phát triển các kỹ năng này để có thể hoàn thành các bài thực hành khác trong khóa học.

## Những gì bạn nên biết trước

Bạn nên quen thuộc với:

- ✓ Cách cài đặt và mở **Android Studio**.
- ✓ Cách tạo ứng dụng **HelloWorld**.
- ✓ Cách chạy ứng dụng **HelloWorld**.

## Những gì bạn sẽ học

- ✓ Cách tạo một ứng dụng có hành vi tương tác.
- ✓ Cách sử dụng **layout editor** để thiết kế bố cục.
- ✓ Cách chỉnh sửa bố cục trong **XML**.
- ✓ Nhiều thuật ngữ mới. Hãy xem **Bảng thuật ngữ và khái niệm** để có định nghĩa dễ hiểu.

## Những gì bạn sẽ làm

- ✓ Tạo một ứng dụng và thêm hai **Button** cùng một **TextView** vào bố cục.
- ✓ Điều chỉnh các phần tử trong **ConstraintLayout** để căn chỉnh chúng với lề và các phần tử khác.
- ✓ Thay đổi thuộc tính của các phần tử UI.
- ✓ Chỉnh sửa bố cục ứng dụng trong **XML**.
- ✓ Trích xuất các chuỗi cứng thành tài nguyên chuỗi (**string resources**).
- ✓ Triển khai phương thức xử lý sự kiện khi nhấn **Button**, hiển thị thông báo trên màn hình khi người dùng nhấn vào mỗi nút.

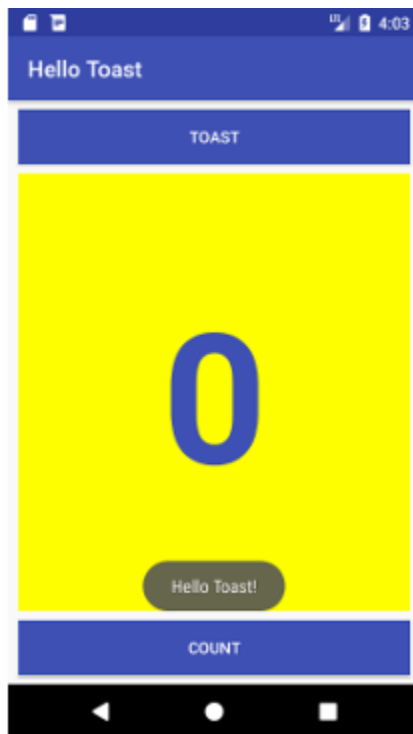
## Tổng quan ứng dụng

Ứng dụng **HelloToast** bao gồm hai **Button** và một **TextView**.

- ✓ Khi người dùng nhấn vào **Button đầu tiên**, ứng dụng sẽ hiển thị một thông báo ngắn (**Toast**) trên màn hình.

✓ Khi nhấn vào **Button thứ hai**, giá trị của **bộ đếm số lần nhấn** trong **TextView** sẽ tăng lên, bắt đầu từ **0**.

✎ Dưới đây là hình ảnh của ứng dụng sau khi hoàn thành: (Hình minh họa ứng dụng).



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

Trong bài thực hành này, bạn sẽ thiết kế và triển khai một dự án cho ứng dụng **HelloToast**.


✎ Liên kết đến mã nguồn giải pháp sẽ được cung cấp ở cuối bài.

### 1.1 Tạo dự án Android Studio

Bắt đầu Android Studio và tạo một dự án mới với các thông số sau:

Attribute	Value
-----------	-------

Application Name	<b>Hello Toast</b>
Company Name	<b>com.example.android</b> (or your own domain)
Phone and Tablet Minimum SDK	<b>API15: Android 4.0.3 IceCreamSandwich</b>
Template	<b>Empty Activity</b>
Generate Layout file box	Selected
Backwards Compatibility box	Selected

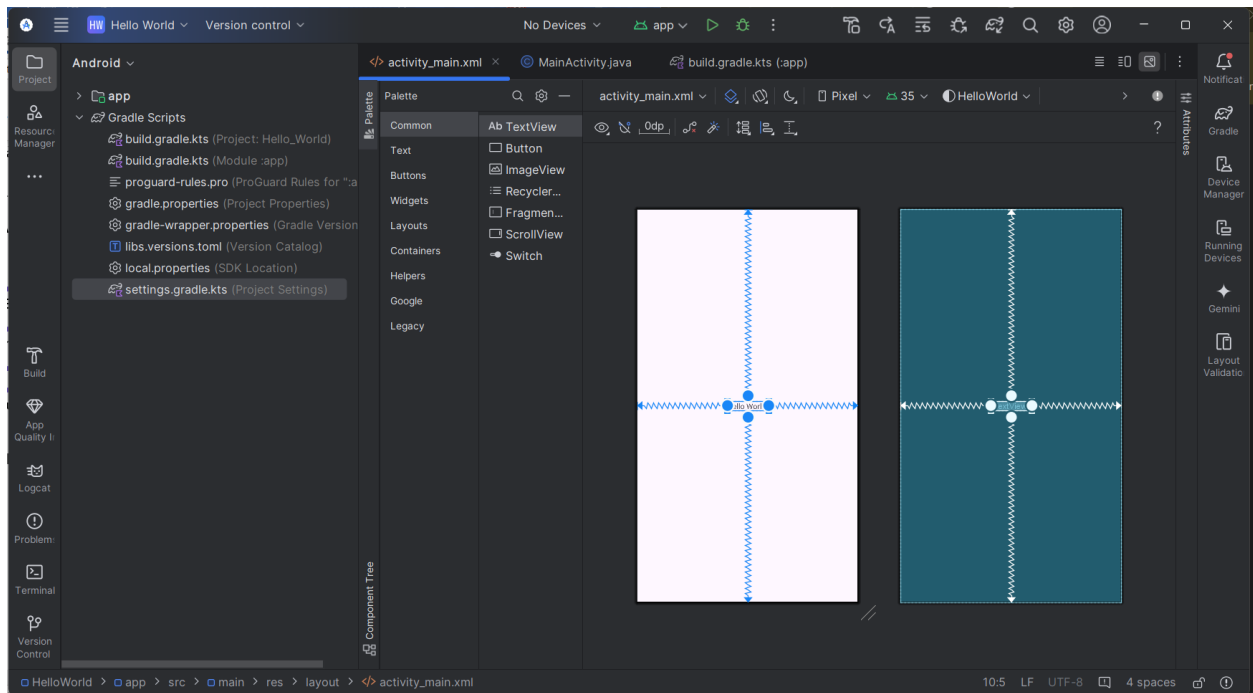
Chọn **Run > Run app** hoặc nhấp vào **biểu tượng Run**  trên thanh công cụ để biên dịch và chạy ứng dụng trên trình giả lập (**emulator**) hoặc thiết bị thực của bạn.

## 1.2 Khám phá trình chỉnh sửa bố cục

**Android Studio** cung cấp **trình chỉnh sửa bố cục (Layout Editor)** để nhanh chóng xây dựng bố cục giao diện người dùng (**UI**) của ứng dụng.

- ✓ Cho phép **kéo thả các phần tử** vào chế độ xem thiết kế trực quan và bản thiết kế (**Blueprint View**).
- ✓ Hỗ trợ **định vị các phần tử** trong bố cục, thêm **ràng buộc (constraints)** và thiết lập thuộc tính.
- ✓ **Ràng buộc (Constraints)** xác định vị trí của một phần tử UI trong bố cục.
- ✓ Một **constraint** đại diện cho **mối liên kết hoặc căn chỉnh** với một phần tử khác, bố cục cha hoặc một đường hướng dẫn vô hình.

Khám phá **trình chỉnh sửa bố cục (Layout Editor)** và tham khảo hình minh họa bên dưới khi thực hiện các bước theo số thứ tự.



1. Trong bảng **Project > Android**, mở thư mục **app > res > layout**, sau đó nhấp đúp vào tệp **activity\_main.xml** để mở nó (nếu chưa mở).
2. Nhấp vào tab **Design** nếu nó chưa được chọn. Tab **Design** được dùng để thao tác với các phần tử và bố cục, trong khi tab **Text** dùng để chỉnh sửa mã XML của bố cục.
3. **Pane Palettes** hiển thị các phần tử UI mà bạn có thể sử dụng trong bố cục ứng dụng.
4. **Pane Component Tree** hiển thị cây phân cấp của các phần tử UI. Các phần tử **View** được tổ chức theo cấu trúc cây gồm **cha và con**, trong đó phần tử con kế thừa thuộc tính của phần tử cha. Trong hình minh họa, **TextView** là phần tử con của **ConstraintLayout**. Bạn sẽ tìm hiểu thêm về các phần tử này trong bài học.
5. **Pane thiết kế và bản thiết kế (Blueprint)** của **Layout Editor** hiển thị các phần tử UI trong bố cục. Trong hình minh họa, bố cục chỉ có một phần tử: **TextView**, hiển thị dòng chữ "**Hello World**".
6. Tab **Attributes** hiển thị bảng **Thuộc tính (Attributes Pane)**, nơi bạn có thể thiết lập các thuộc tính cho một phần tử UI.

**Mẹo:** Xem **Building a UI with Layout Editor** để biết chi tiết về cách sử dụng **Layout Editor**, và **Meet Android Studio** để xem tài liệu đầy đủ về **Android Studio**.

## Nhiệm vụ 2: Thêm các phần tử View trong Layout Editor

Trong nhiệm vụ này, bạn sẽ tạo bố cục giao diện người dùng (**UI layout**) cho ứng dụng **HelloToast** bằng cách sử dụng các tính năng của **ConstraintLayout**.

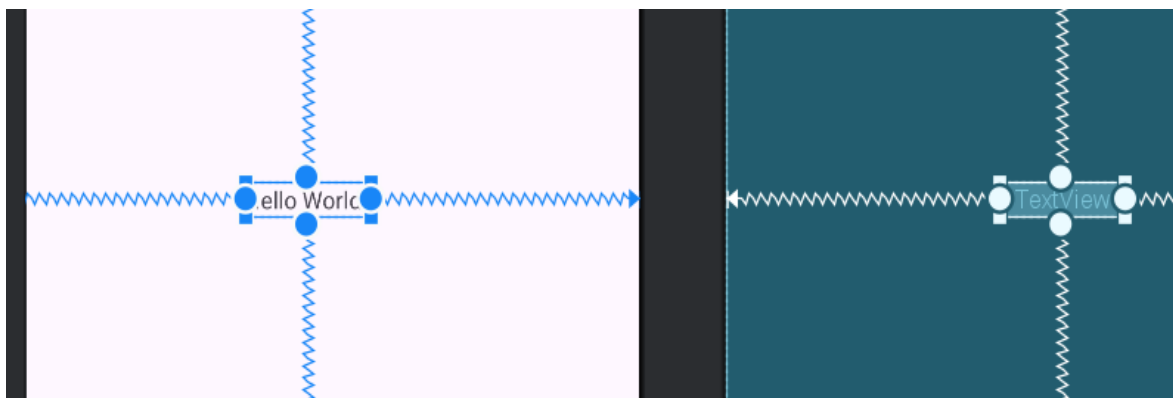
Bạn có thể tạo **constraints (ràng buộc)** theo hai cách:

- ✓ **Thủ công** – Tạo ràng buộc bằng cách kéo thả và chỉnh sửa trong **Attributes**.
- ✓ **Tự động** – Sử dụng công cụ **Autoconnect** để tạo ràng buộc tự động.

### 2.1 Kiểm tra các ràng buộc (constraints) của phần tử

Thực hiện các bước sau:

- Mở tệp **activity\_main.xml** trong bảng **Project > Android** (nếu chưa mở). Nếu tab **Design** chưa được chọn, hãy nhấp vào nó.
  - Nếu không thấy **Blueprint**, nhấp vào nút **Select Design Surface** trên thanh công cụ và chọn **Design + Blueprint**.
- Công cụ **Autoconnect** cũng nằm trên thanh công cụ và được bật theo mặc định. Ở bước này, hãy đảm bảo rằng công cụ không bị tắt.
- Nhấp vào nút **Zoom In**  để phóng to khu vực thiết kế và bảng **Blueprint** để quan sát kỹ hơn.
- Chọn **TextView** trong bảng **Component Tree**. Dòng chữ "**Hello World**" trong **TextView** sẽ được làm nổi bật trong bảng thiết kế và bảng **Blueprint**, đồng thời các **ràng buộc (constraints)** của phần tử sẽ hiển thị.
- Thực hiện theo hình minh họa động bên dưới:
  - Nhấp vào **nút tròn (circular handle)** bên phải của **TextView** để xóa ràng buộc ngang đang liên kết phần tử với cạnh phải của bố cục.
  - Sau khi xóa ràng buộc, **TextView** sẽ nhảy sang bên trái vì nó không còn bị ràng buộc về phía bên phải.
  - Để thêm lại ràng buộc ngang, nhấp vào cùng một **nút tròn** và kéo một đường nối đến cạnh phải của bố cục.





Trong bảng **Blueprint** hoặc **Design**, các nút điều khiển sau sẽ xuất hiện trên phần tử **TextView**:

✓ **Constraint handle**: Dùng để tạo ràng buộc.

- Nhấp vào **nút tròn (constraint handle)** ở cạnh của phần tử.
- Kéo và thả vào một **constraint handle khác** hoặc **ranh giới của phần tử cha**.
- Một **đường zic-zac** sẽ xuất hiện để biểu thị ràng buộc.

✓ **Resizing handle**: Dùng để thay đổi kích thước phần tử.

- Kéo các **nút vuông (resizing handles)** ở góc phần tử để thay đổi kích thước.
- Khi kéo, nút điều khiển sẽ đổi thành **góc xiên** để hiển thị trạng thái điều chỉnh kích thước.

## 2.2 Thêm Nút (Button) vào Giao Diện

Khi **Autoconnect** được bật, công cụ này sẽ **tự động tạo ràng buộc** (constraints) giữa một phần tử UI với bố cục cha (**parent layout**). Sau khi kéo thả phần tử vào bố cục, hệ thống sẽ tạo ràng buộc dựa trên **vị trí của phần tử**.

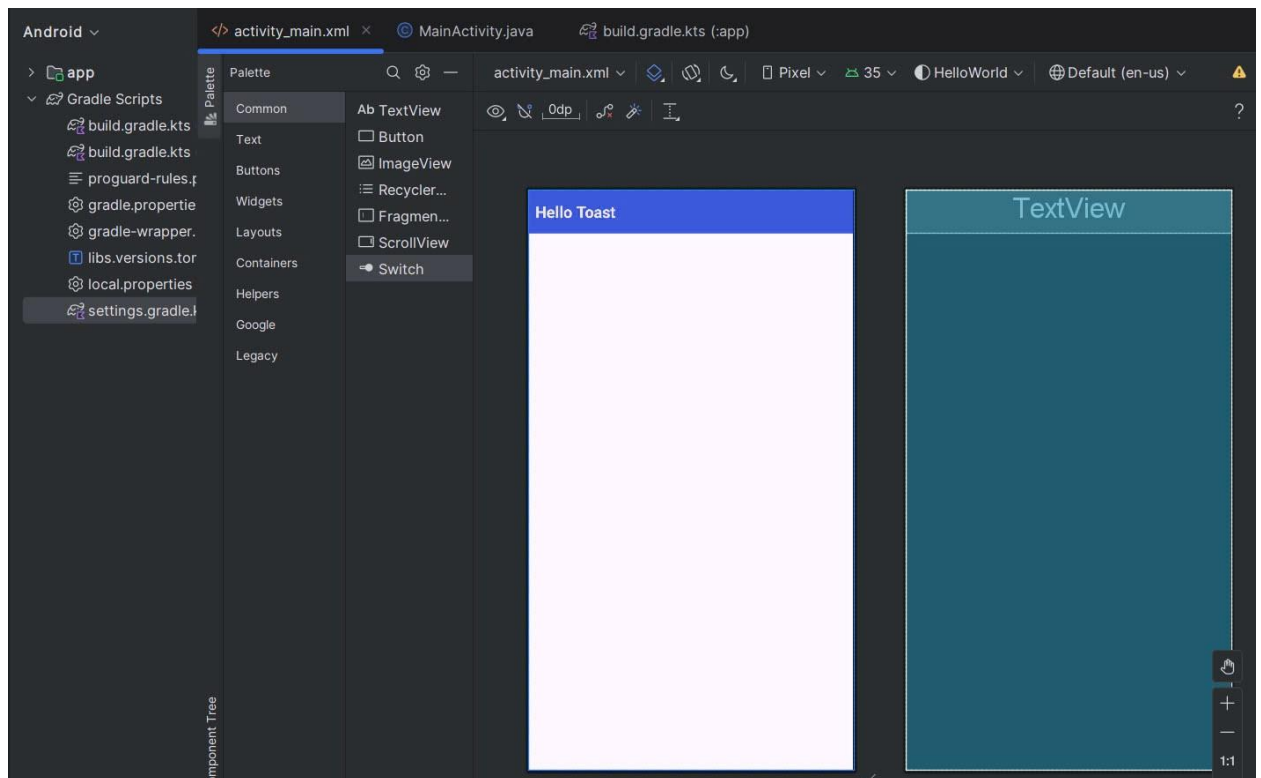
◆ **Thực hiện các bước sau để thêm một Button:**

### 1. Xóa TextView mặc định

- Nếu **TextView** vẫn đang được chọn, nhấn **phím Delete** hoặc chọn **Edit > Delete** để xóa nó.
- Lúc này, bố cục (**layout**) sẽ **trống hoàn toàn**.

### 2. Thêm Button vào giao diện

- Kéo thả một **Button** từ bảng **Palette** vào bất kỳ vị trí nào trong **layout**.
- Nếu thả **Button** vào **giữa phía trên** của layout, các ràng buộc (**constraints**) có thể được tạo tự động.
- Nếu không, bạn có thể **kéo các ràng buộc** thủ công để căn chỉnh **Button** với **đỉnh, trái, và phải** của bố cục như trong hình minh họa.



## 2.3 Thêm Nút (Button) thứ hai

### 1. Kéo thả Button thứ hai

- Kéo một **Button** khác từ bảng **Palette** vào **giữa layout**, như trong hình minh họa.
- **Autoconnect** có thể tự động tạo ràng buộc ngang (**horizontal constraints**). Nếu không, bạn có thể **kéo ràng buộc ngang** thủ công.

### 2. Thêm ràng buộc dọc (vertical constraint)

- Kéo một **ràng buộc dọc** từ Button thứ hai xuống **đáy của layout** để cố định vị trí của nó.



Bạn có thể xóa ràng buộc của một phần tử bằng cách chọn phần tử đó và di chuột lên để hiển thị nút **Clear Constraints**. Nhấp vào nút này để xóa **tất cả** ràng buộc của phần tử đã chọn. Nếu bạn chỉ muốn xóa một ràng buộc cụ thể, hãy nhấp vào **điểm neo** (handle) của ràng buộc đó.

Để xóa toàn bộ ràng buộc trong giao diện, nhấp vào công cụ **Clear All Constraints** trên thanh công cụ. Công cụ này đặc biệt hữu ích khi bạn muốn thiết lập lại toàn bộ ràng buộc trong bố cục của mình.

## Nhiệm vụ 3: Thay đổi thuộc tính của phần tử UI

Ngăn **Attributes** cung cấp quyền truy cập vào tất cả các thuộc tính XML mà bạn có thể gán cho một phần tử UI. Bạn có thể tìm thấy các thuộc tính chung cho tất cả các **View** trong tài liệu của **View class**.

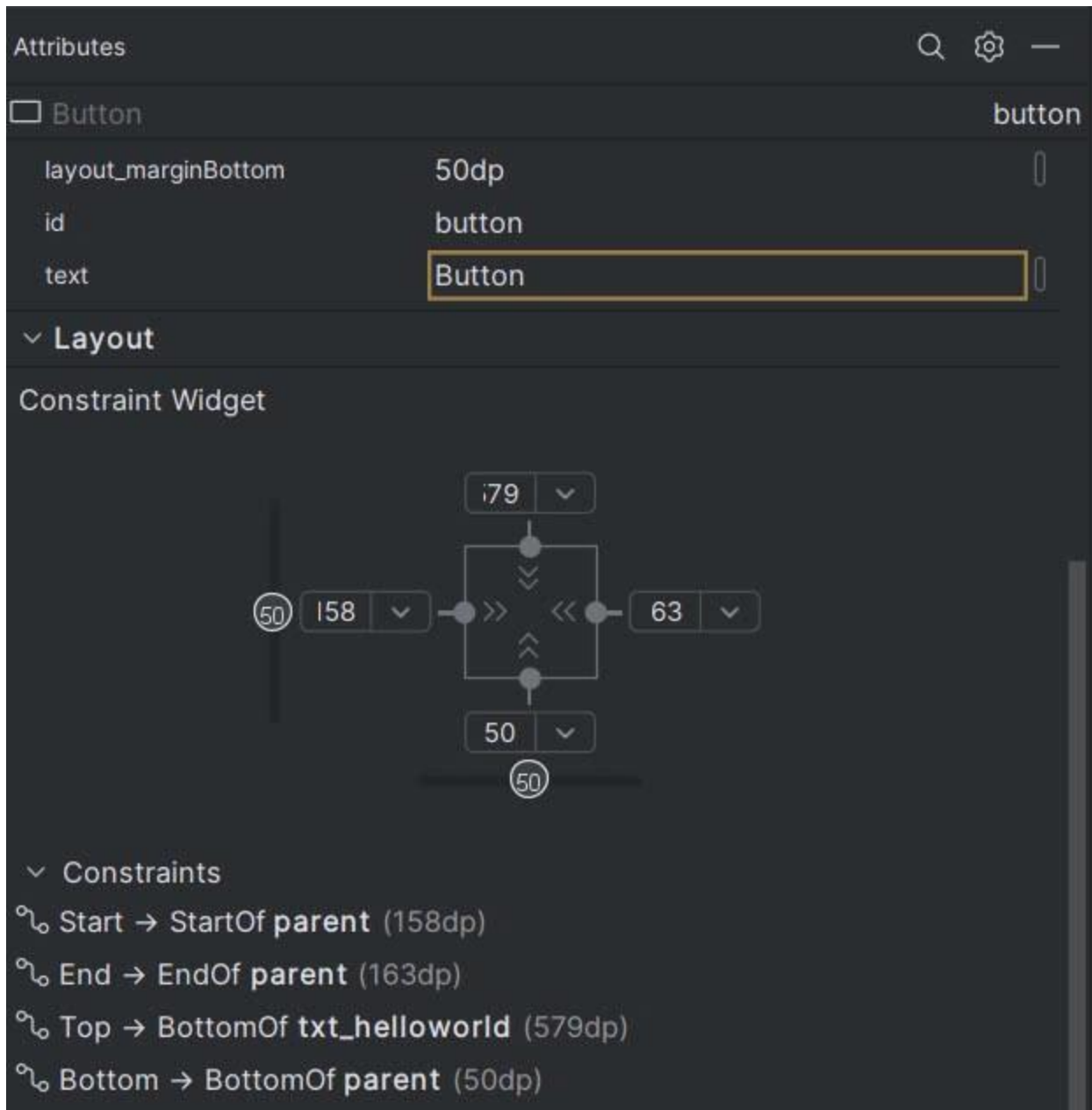
Trong bài này, bạn sẽ nhập các giá trị mới và thay đổi các thuộc tính quan trọng của **Button**, những thuộc tính này cũng áp dụng cho hầu hết các loại **View** khác.

### 3.1 Thay đổi kích cỡ nút

Trình chỉnh sửa bố cục (**Layout Editor**) cung cấp các **tay cầm điều chỉnh kích thước** ở bốn góc của **View**, giúp bạn thay đổi kích thước nhanh chóng. Bạn có thể kéo các tay cầm này để điều chỉnh kích thước của **View**, nhưng cách này sẽ **mã hóa cứng (hardcode)** các giá trị chiều rộng và chiều cao.

Tuy nhiên, bạn nên **tránh** mã hóa cứng kích thước cho hầu hết các phần tử **View**, vì chúng không thể thích ứng với nội dung và kích thước màn hình khác nhau.

Thay vào đó, hãy sử dụng ngăn **Attributes** bên phải **Layout Editor** để chọn chế độ kích thước không sử dụng giá trị cố định. Trong **Attributes**, có một **bảng điều chỉnh kích thước (view inspector)** hình vuông ở trên cùng. Các biểu tượng trong hình vuông này biểu thị các cài đặt chiều rộng và chiều cao như sau:



Trong hình minh họa trên:

1. **Điều khiển chiều cao (Height control)**

- Xác định thuộc tính **layout\_height** và xuất hiện ở hai cạnh trên và dưới của hình vuông.
- Các góc nghiêng cho thấy chế độ **wrap\_content** đang được sử dụng, nghĩa là **View** sẽ mở rộng theo chiều dọc để phù hợp với nội dung.
- Số "8" biểu thị phần lẻ **tiêu chuẩn** được đặt là **8dp**.

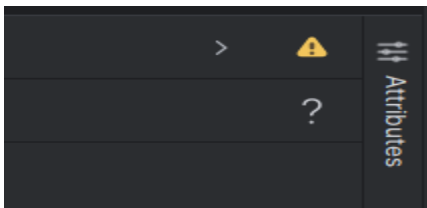
2. **Điều khiển chiều rộng (Width control)**

- Xác định thuộc tính **layout\_width** và xuất hiện ở hai cạnh trái và phải của hình vuông.

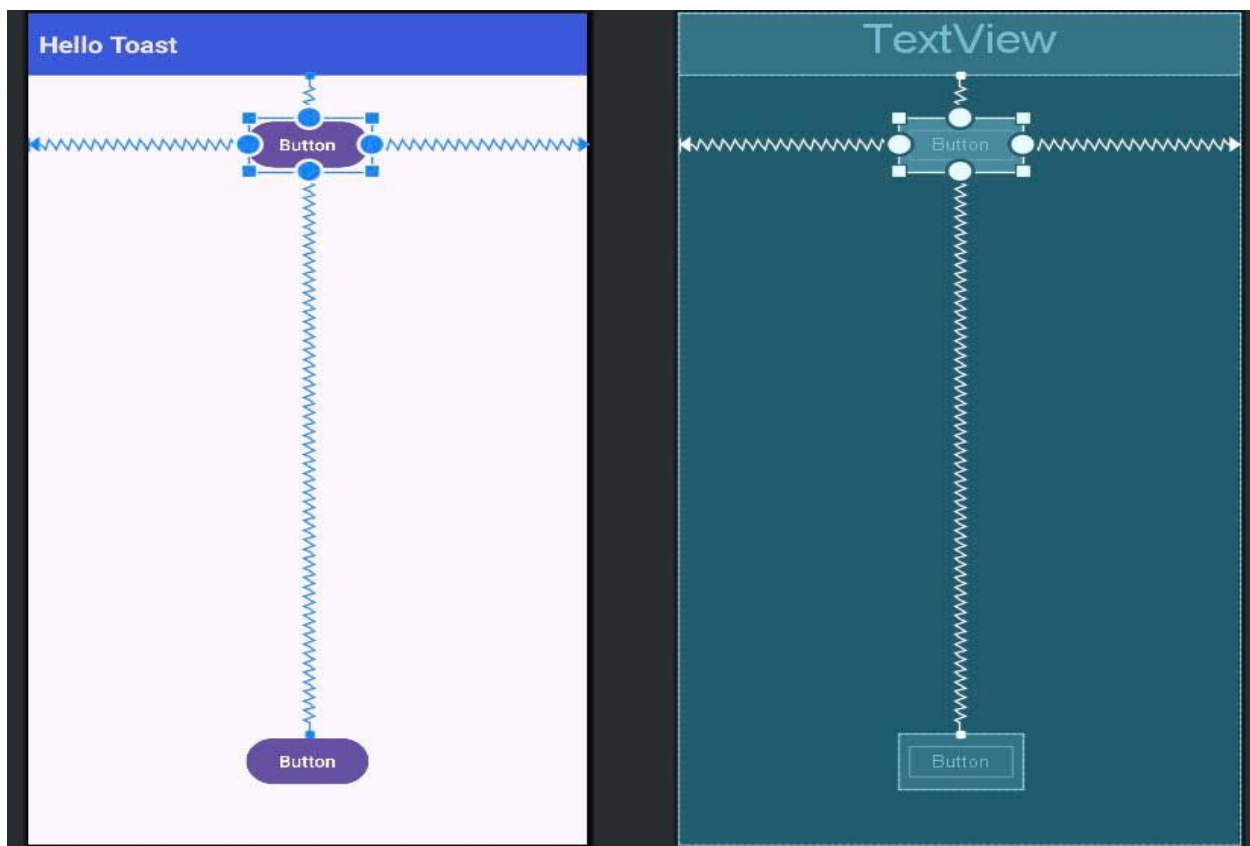
- Các góc nghiêng cho thấy chế độ **wrap\_content** đang được sử dụng, nghĩa là **View** sẽ mở rộng theo chiều ngang để phù hợp với nội dung, nhưng không vượt quá lề **8dp**.
3. **Nút đóng ngăn Attributes**
- Nhấp vào đây để đóng ngăn **Attributes** nếu bạn không cần sử dụng nữa.

Thực hiện bước sau:

1. Chọn **Button** ở phía trên trong ngăn **Component Tree**.
2. Nhấp vào tab **Attributes** ở bên phải cửa sổ **Layout Editor**.

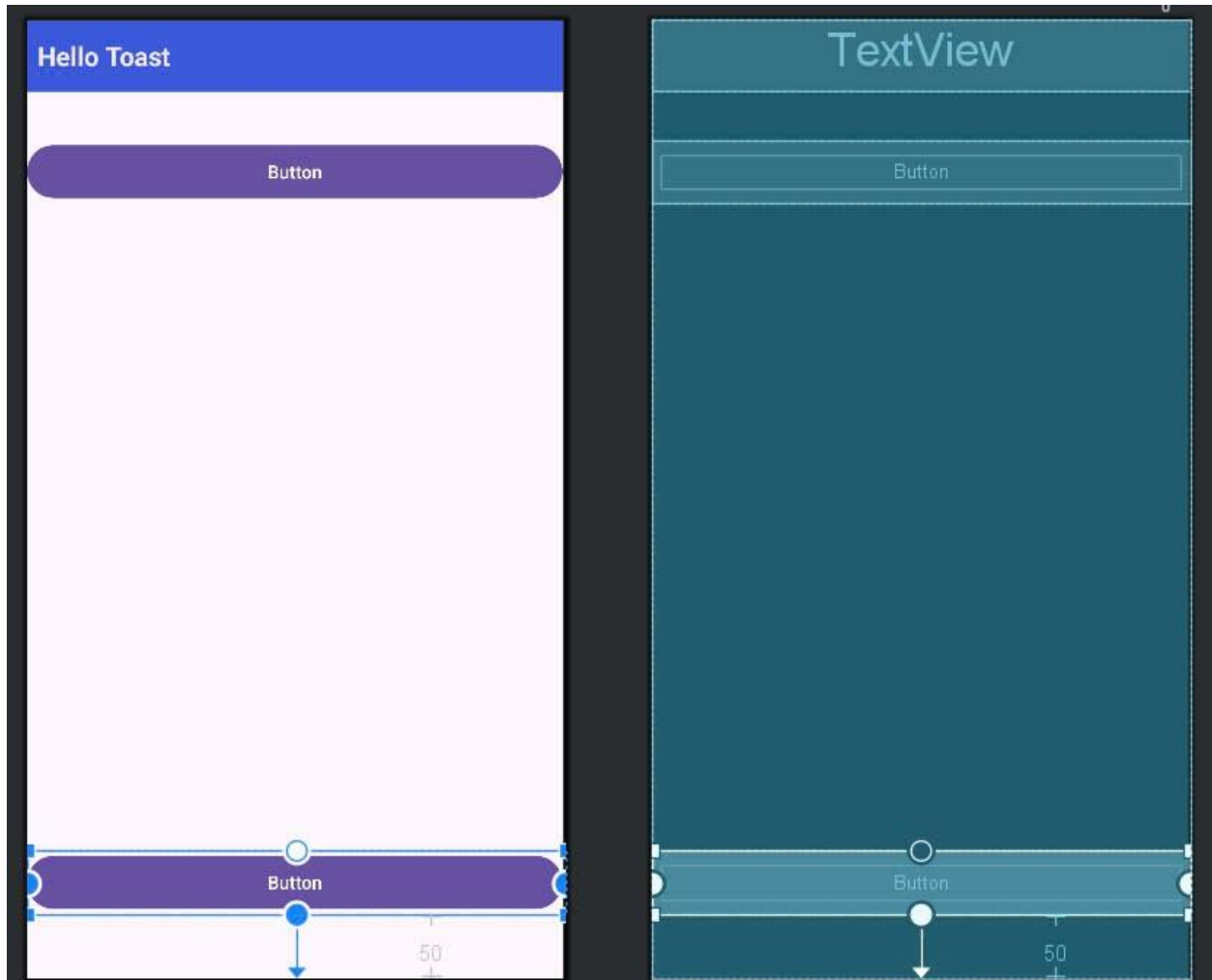


3. Nhấp vào **điều khiển chiều rộng** hai lần. Lần nhấp đầu tiên sẽ chuyển sang chế độ Fixed, hiển thị dưới dạng các đường thẳng. Lần nhấp thứ hai sẽ chuyển sang chế độ Match Constraints, hiển thị dưới dạng các lò xo. Bạn có thể quan sát sự thay đổi này trong hình minh họa bên dưới.



Sau khi thay đổi **điều khiển chiều rộng**, thuộc tính **layout\_width** trong ngăn **Attributes** sẽ hiển thị giá trị **match\_constraint**, và phần tử **Button** sẽ mở rộng theo chiều ngang để lấp đầy không gian giữa hai cạnh trái và phải của bố cục.

4. Chọn **Button** thứ hai và thực hiện các thay đổi tương tự cho **layout\_width** như ở bước trước, như minh họa trong hình bên dưới.



Như đã thấy trong các bước trước, các thuộc tính **layout\_width** và **layout\_height** trong ngăn **Attributes** sẽ thay đổi khi bạn điều chỉnh các điều khiển chiều rộng và chiều cao trong **inspector**. Các thuộc tính này có thể nhận một trong ba giá trị trong **ConstraintLayout**:

- **match\_constraint**: Mở rộng phần tử **View** để lấp đầy không gian của **parent** theo chiều rộng hoặc chiều cao, giới hạn trong phạm vi của **margin** (nếu có). Trong trường hợp này, **parent** chính là **ConstraintLayout**. Bạn sẽ tìm hiểu thêm về **ConstraintLayout** trong nhiệm vụ tiếp theo.
- **wrap\_content**: Thu nhỏ phần tử **View** sao cho nó vừa đủ chứa nội dung bên trong. Nếu không có nội dung, phần tử **View** sẽ trở nên vô hình.

- **Fixed size:** Để đặt kích thước cố định nhưng vẫn thích ứng với kích thước màn hình của thiết bị, sử dụng đơn vị **dp (density-independent pixels)**. Ví dụ, **16dp** tương ứng với 16 pixel độc lập với mật độ màn hình.

**Mẹo:** Nếu bạn thay đổi thuộc tính **layout\_width** bằng cách sử dụng menu bật lên, giá trị của **layout\_width** sẽ được đặt về **0** vì không có kích thước cố định. Cài đặt này tương đương với **match\_constraint**—phần tử có thể mở rộng tối đa theo các ràng buộc và thiết lập **margin**.

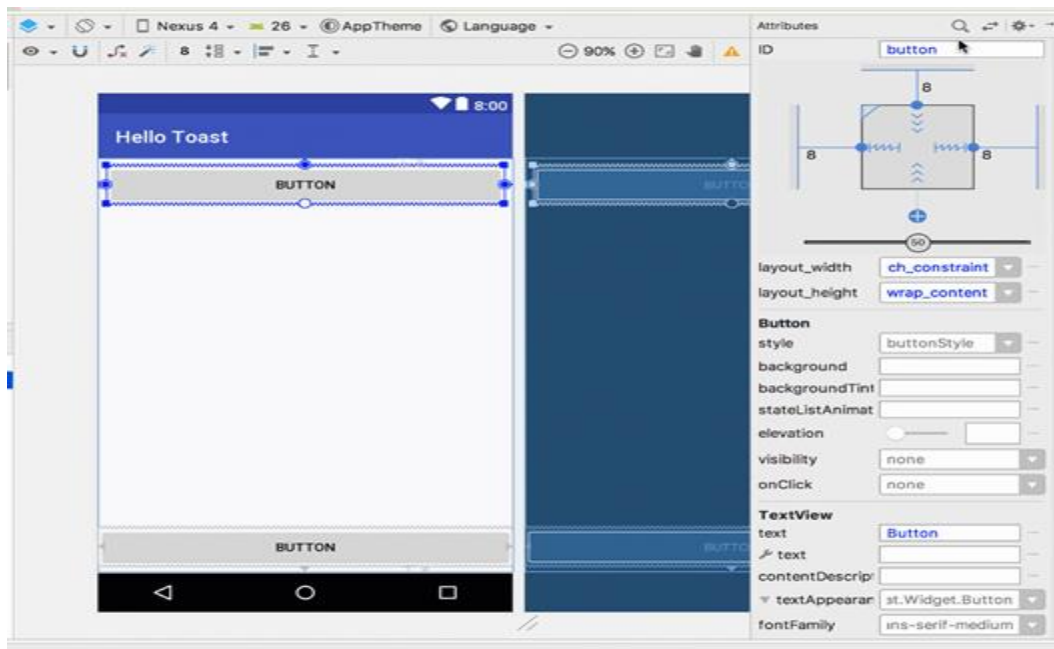
## 3.2 Thay đổi thuộc tính của Button

Để xác định duy nhất từng **View** trong một **Activity** layout, mỗi **View** hoặc các lớp con của **View** (như **Button**) cần có một **ID** duy nhất. Ngoài ra, để có chức năng cụ thể, các phần tử **Button** cần có văn bản hiển thị. Các **View** cũng có thể có nền là màu hoặc hình ảnh.

Ngăn **Attributes** cho phép truy cập vào tất cả các thuộc tính có thể gán cho một phần tử **View**. Bạn có thể nhập giá trị cho từng thuộc tính, chẳng hạn như **android:id**, **background**, **textColor**, và **text**.

Dưới đây là các bước thực hiện:

1. Sau khi chọn **Button** đầu tiên, chỉnh sửa trường **ID** ở đầu ngăn **Attributes** thành **button\_toast** để đặt thuộc tính **android:id**, giúp xác định phần tử trong layout.
2. Đặt thuộc tính **background** thành **@color/colorPrimary**. (Khi nhập **@c**, các lựa chọn sẽ xuất hiện để bạn dễ dàng chọn.)
3. Đặt thuộc tính **textColor** thành **@android:color/white**.
4. Chỉnh sửa thuộc tính **text** thành **Toast**.



- Thực hiện các thay đổi thuộc tính tương tự cho **Button** thứ hai, sử dụng **button\_count** làm **ID**, đặt **Count** cho thuộc tính **text**, và sử dụng cùng màu nền và màu chữ như các bước trước.

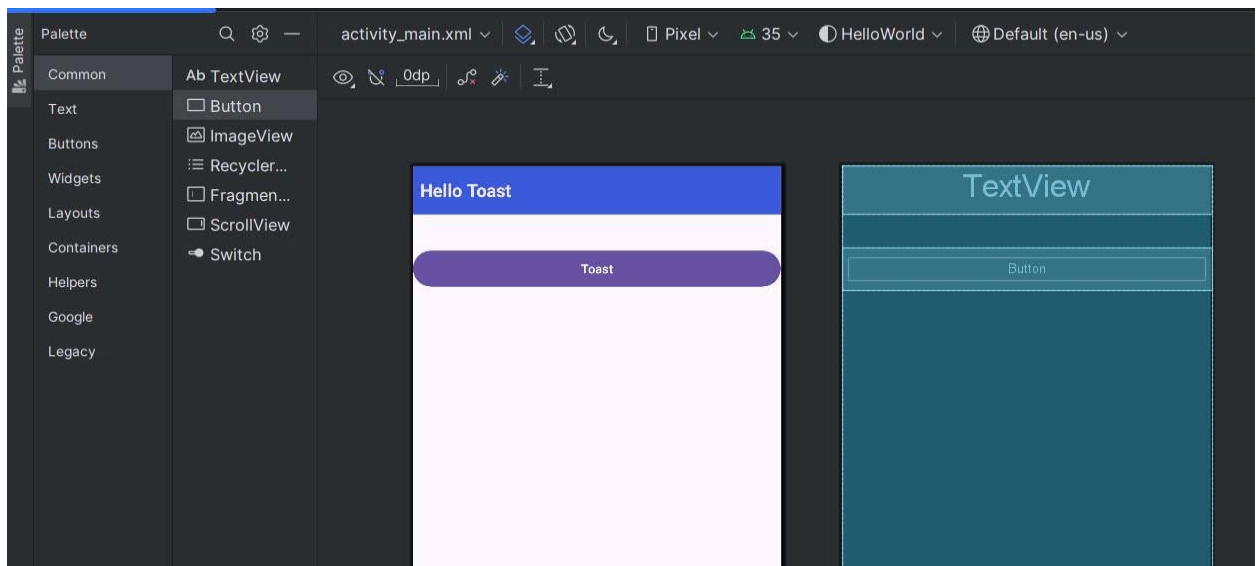
Màu **colorPrimary** là màu chính của giao diện ứng dụng, một trong các màu cơ bản được định nghĩa trước trong tệp tài nguyên **colors.xml**. Màu này thường được sử dụng cho thanh ứng dụng (**app bar**). Việc sử dụng các màu cơ bản này cho các phần tử giao diện khác giúp tạo ra một giao diện đồng nhất. Bạn sẽ tìm hiểu thêm về chủ đề ứng dụng (**app themes**) và **Material Design** trong bài học khác.

## Nhiệm vụ 4: Thêm TextyEdit và thiết lập thuộc tính

Một trong những lợi ích của **ConstraintLayout** là khả năng căn chỉnh hoặc ràng buộc các phần tử dựa trên các phần tử khác. Trong nhiệm vụ này, bạn sẽ thêm một **TextView** vào giữa bố cục và ràng buộc nó theo chiều ngang với lề và theo chiều dọc với hai **Button**. Sau đó, bạn sẽ thay đổi các thuộc tính cho **TextView** trong bảng **Attributes**.

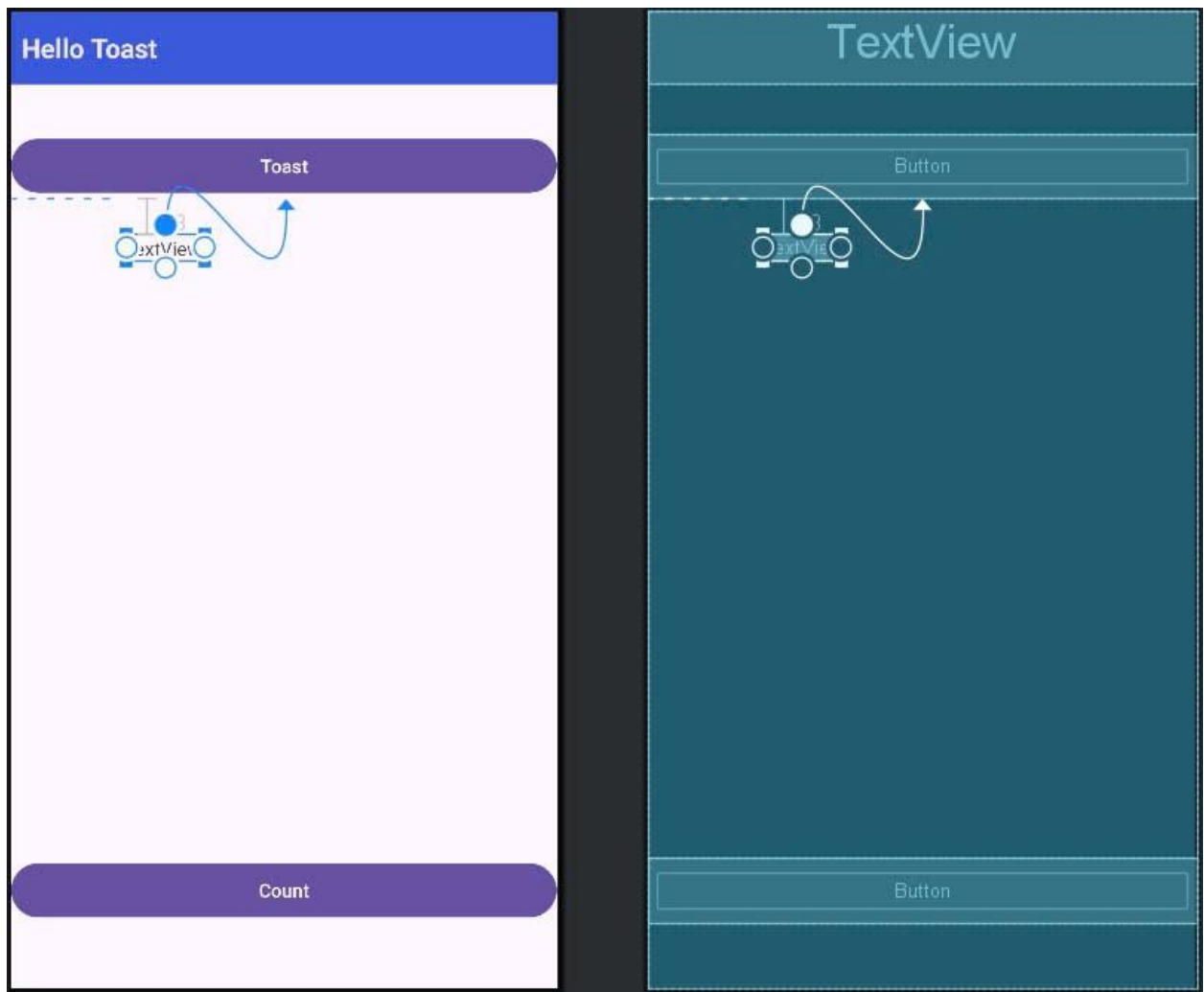
### 4.1 Thêm TextView và ràng buộc

- Như hình minh họa bên dưới, kéo một **TextView** từ bảng **Palette** vào phần trên của bố cục, sau đó kéo một ràng buộc từ mép trên của **TextView** đến tay cầm ở mép dưới của nút **Toast**. Điều này sẽ ràng buộc **TextView** nằm ngay bên dưới nút **Toast**.



- Như hình minh họa bên dưới, kéo một ràng buộc từ mép dưới của **TextView** đến tay cầm ở mép trên của nút **Count**, và kéo các ràng buộc từ hai bên của **TextView** đến hai cạnh của bố cục. Điều này sẽ giúp **TextView** nằm giữa bố cục, giữa hai nút **Toast** và **Count**.

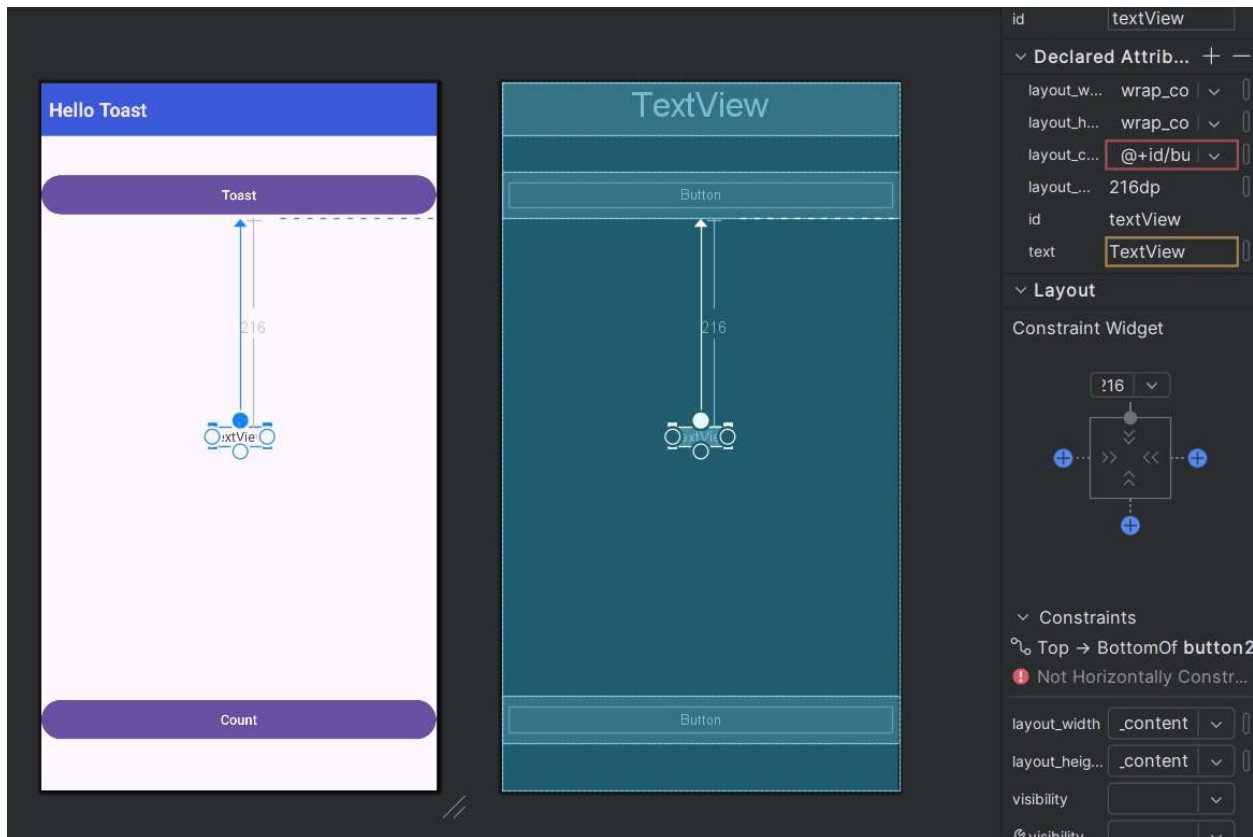




## 4.2 Thiết lập thuộc tính TextView

Với **TextView** đã chọn, mở bảng **Attributes** nếu chưa mở sẵn. Thiết lập các thuộc tính cho **TextView** như hình minh họa bên dưới. Các thuộc tính mới sẽ được giải thích sau:

1. Đặt **ID** thành `show_count`.
2. Đặt **text** thành 0.
3. Đặt **textSize** thành `160sp`.
4. Đặt **textStyle** thành **B** (đậm) và **textAlignment** thành **ALIGNCENTER** (căn giữa đoạn văn).
5. Thay đổi chế độ kích thước ngang và dọc (**layout\_width** và **layout\_height**) thành **match\_constraint**.
6. Đặt **textColor** thành `@color/colorPrimary`.
7. Cuộn xuống bảng thuộc tính, nhấp vào **View all attributes**, tiếp tục cuộn xuống mục **background** và nhập `#FFF00` để đặt màu vàng.
8. Cuộn xuống **gravity**, mở rộng **gravity**, và chọn **center\_ver** (căn giữa theo chiều dọc).



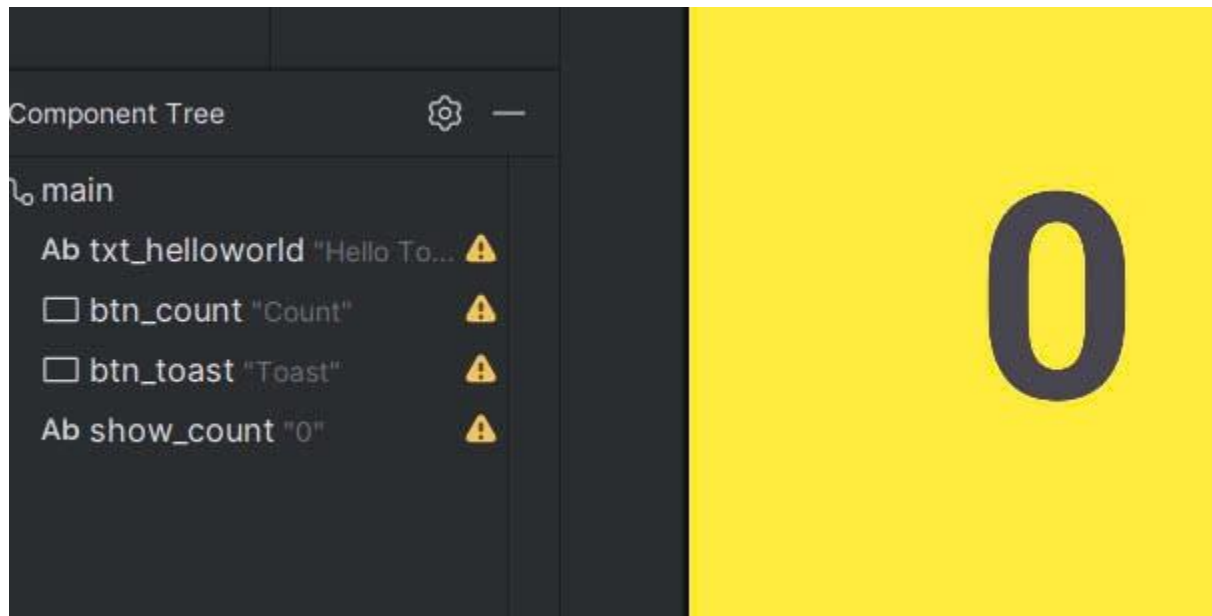
- **textSize**: Kích thước chữ của **TextView**. Trong bài học này, kích thước được đặt là **160sp**. Đơn vị **sp** (scale-independent pixel) giống như **dp**, giúp văn bản tự động điều chỉnh theo mật độ màn hình và cài đặt kích thước chữ của người dùng. Nên sử dụng **sp** khi chỉ định kích thước chữ để đảm bảo tính thích ứng.
- **textStyle** và **textAlignment**: Kiểu chữ được đặt thành **B** (đậm), và căn chỉnh văn bản được đặt thành **ALIGNCENTER** (căn giữa đoạn văn).
- **gravity**: Thuộc tính **gravity** xác định cách **View** được căn chỉnh trong **parent View** hoặc **ViewGroup**. Trong bước này, **TextView** được căn giữa theo chiều dọc trong **ConstraintLayout**.

Bạn có thể nhận thấy rằng thuộc tính **background** xuất hiện trên trang đầu tiên của bảng **Attributes** đối với **Button**, nhưng lại nằm trên trang thứ hai đối với **TextView**. Lý do là bảng **Attributes** thay đổi theo từng loại **View**: Những thuộc tính phổ biến nhất sẽ xuất hiện ở trang đầu tiên, còn các thuộc tính khác sẽ nằm ở trang thứ hai. Để quay lại trang đầu tiên của bảng **Attributes**, hãy nhấp vào biểu tượng trong thanh công cụ ở đầu bảng.

## Nhiệm vụ 5: Chỉnh sửa layout trong XML

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 **Component Tree**. Khi di chuột qua các dấu chấm than này, bạn sẽ thấy thông báo cảnh báo như hình bên dưới.

Tất cả ba phần tử đều có cùng một cảnh báo: **Chuỗi ký tự được mã hóa cứng (hardcoded strings) nên được lưu trong tài nguyên thay vì viết trực tiếp trong mã.**



Cách dễ nhất để khắc phục vấn đề này là chỉnh sửa layout trong XML. Mặc dù **layout editor** là một công cụ mạnh mẽ, nhưng một số thay đổi sẽ dễ dàng thực hiện hơn khi chỉnh sửa trực tiếp trong mã nguồn XML.

## 5.1 Mở mã XML của layout

Trong bước này, hãy mở tệp **activity\_main.xml** (nếu chưa mở) và nhấp vào tab **Text** ở phía dưới **layout editor** để chuyển sang chế độ chỉnh sửa XML trực tiếp.

Trình chỉnh sửa XML sẽ xuất hiện, thay thế các bảng thiết kế và bản vẽ. Như bạn có thể thấy trong hình bên dưới, một phần mã XML của layout hiển thị các cảnh báo được đánh dấu—các chuỗi được mã hóa cứng **"Toast"** và **"Count"**. (Chuỗi được mã hóa cứng **"0"** cũng bị đánh dấu nhưng không hiển thị trong hình.) Di chuột qua chuỗi **"Toast"** để xem thông báo cảnh báo.

```
activity_main.xml x strings.xml MainActivity.java
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
26     app:layout_constraintVertical_bias="0.0" />
27
28     <Button
29         android:id="@+id/btn_count"
30         android:layout_width="0dp"
31         android:layout_height="80dp"
32         android:layout_marginBottom="4dp"
33         android:onClick="countUp"
34         android:text="@string/button_count"
35         app:layout_constraintBottom_toBottomOf="parent"
36         app:layout_constraintEnd_toEndOf="parent"
37         app:layout_constraintHorizontal_bias="0.0"
38         app:layout_constraintStart_toStartOf="parent" />
39
40     <Button
41         android:id="@+id/btn_toast"
42         android:layout_width="0dp"
43         android:layout_height="80dp"
44         android:layout_marginTop="10dp"
45         android:onClick="showToast"
46         android:text="@string/button_toast"
47         app:layout_constraintEnd_toEndOf="parent"
48         app:layout_constraintHorizontal_bias="0.0"
49         app:layout_constraintStart_toStartOf="parent"
50         app:layout_constraintTop_toBottomOf="@+id/txt_helloworld" />
51
52     <TextView
53         android:id="@+id/show_count"
```

## 5.2 Trích xuất tài nguyên chuỗi

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

1. Nhấp vào từ **"Toast"** (cảnh báo đầu tiên được đánh dấu).
2. Nhấn **Alt + Enter** trên Windows hoặc **Option + Enter** trên macOS và chọn **Extract string resource** từ menu bật lên.

3. Nhập **button\_label\_toast** vào trường **Resource name**.
4. Nhấn **OK**. Một tài nguyên chuỗi sẽ được tạo trong tệp **values/res/strings.xml**, và chuỗi trong mã sẽ 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"**
  - **count\_initial\_value** cho **"0"**
6. Trong bảng **Project > Android**, mở rộng thư mục **values** 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 World</string>
    <string name="button_toast">Toast</string>
    <string name="count_initial_value">0</string>
    <string name="button_count">Count</string>
    <string name="toast_message">Hello Toast</string>
</resources>
```

7. Bạn cần thêm một chuỗi khác để sử dụng trong một nhiệm vụ tiếp theo nhằm hiển thị thông báo. Hãy thêm vào tệp **strings.xml** một tài nguyên chuỗi mới có tên **toast\_message** với nội dung **"Hello Toast!"**.

```
<resources>
    <string name="app_name">Hello World</string>
    <string name="button_toast">Toast</string>
    <string name="count_initial_value">0</string>
    <string name="button_count">Count</string>
    <string name="toast_message">Hello Toast</string>
</resources>
```

💡 **Mẹo:** Các tài nguyên chuỗi cũng bao gồm tên ứng dụng, xuất hiện trên thanh ứng dụng (app bar) ở đầu màn hình nếu bạn tạo dự án ứng dụng bằng mẫu **Empty Template**. 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 Button

Trong bài này, bạn sẽ thêm một phương thức Java cho từng **Button** trong **MainActivity** để thực thi khi người dùng nhấn vào **Button**.

### 6.1 Thêm thuộc tính onClick và trình xử lý cho từng Button

**Trình xử lý sự kiện click** là một phương thức được gọi khi người dùng nhấp hoặc chạm vào một phần tử UI có thể nhấp. Trong Android Studio, bạn có thể chỉ định tên của phương thức trong trường **onClick** của bảng **Attributes** trong tab **Design**.

Bạn cũng có thể thêm tên của phương thức trình xử lý trong **trình chỉnh sửa XML** bằng cách thêm thuộc tính **android:onClick** vào **Button**. Bạn sẽ sử dụng cách thứ hai vì chưa tạo các phương thức xử lý, và trình chỉnh sửa XML cung cấp một cách tự động để tạo các phương thức này.

**Các bước thực hiện:**

1. Mở trình chỉnh sửa XML (tab **Text**) và 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**, ngay trước ký hiệu kết thúc **/>**.

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

3. Nhấp vào biểu tượng bóng đèn màu đỏ xuất hiện bên cạnh thuộc tính này. Chọn **Create click handler**, chọn **MainActivity**, rồi nhấn **OK**.
  - Nếu biểu tượng bóng đèn không xuất hiện, nhấp vào tên phương thức ("showToast"), sau đó nhấn **Alt + Enter** (hoặc **Option + Enter** trên Mac), chọn **Create 'showToast(View)' in MainActivity**, rồi nhấn **OK**.
  - Hành động này sẽ tạo một phương thức trống (stub method) cho **showToast()** trong **MainActivity**, như được hiển thị ở cuối các bước này.
4. Lặp lại hai bước trên với **button\_count**:
  - Thêm thuộc tính **android:onClick**.
  - Thêm trình xử lý sự kiện

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

Bây giờ, mã XML của các phần tử UI trong **ConstraintLayout** sẽ bao gồm các phương thức **onClick**.

```
<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="16sp"
    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 **Project > Android**, mở rộng thư mục **java**, sau đó mở **com.example.android.hellotoast** và nhấp đúp vào **MainActivity**.
  - Trình chỉnh sửa mã sẽ hiển thị 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 Chỉnh sửa trình xử lý sự kiện cho nút Toast

Bây giờ, bạn sẽ chỉnh sửa phương thức `showToast()`—trình xử lý sự kiện cho nút **Toast** trong **MainActivity**—để hiển thị một thông báo.

**Toast** là một cách hiển thị thông báo đơn giản trong một cửa sổ pop-up nhỏ. Nó chỉ chiếm không gian cần thiết để hiển thị nội dung và không làm gián đoạn hoạt động của ứng dụng.

Bạn có thể sử dụng **Toast** để kiểm tra tính tương tác của ứng dụng, chẳng hạn như hiển thị một thông báo khi người dùng nhấn vào một **Button**.

Hãy làm theo các bước sau để chỉnh sửa trình xử lý sự kiện của nút **Toast**:

1. Tìm phương thức `showToast()` mới được tạo trong **MainActivity**:

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

2. Để tạo một **Toast**, hãy gọi phương thức `makeText()` của lớp **Toast**.

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

Câu lệnh này chưa hoàn chỉnh cho đến khi bạn hoàn thành tất cả các bước sau.



### 3. Cung cấp context của Activity

Vì **Toast** hiển thị trên giao diện của **Activity**, hệ thống cần biết thông tin về **Activity** hiện tại. Khi bạn đang ở trong chính **Activity** đó, có thể dùng `this` làm context.

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

### 4. Cung cấp thông điệp hiển thị

Bạn có thể sử dụng một chuỗi trực tiếp hoặc một tài nguyên chuỗi (như **toast\_message** đã tạo trước đó). Chuỗi tài nguyên **toast\_message** được xác định bởi `R.string.toast_message`.

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

### 5. Cung cấp thời gian hiển thị

Ví dụ, `Toast.LENGTH_SHORT` sẽ hiển thị **Toast** trong khoảng thời gian ngắn (~2 giây).

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

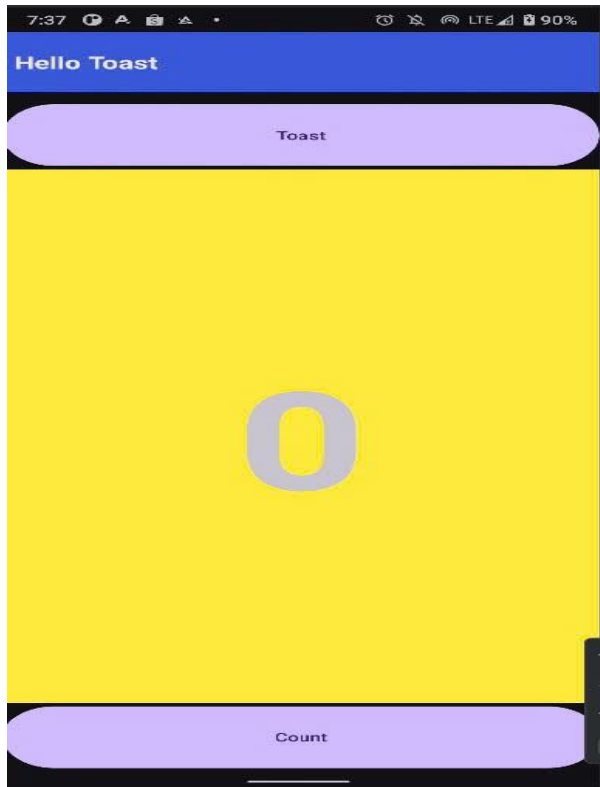
Thời gian hiển thị có thể là `Toast.LENGTH_LONG` (~3.5 giây) hoặc `Toast.LENGTH_SHORT` (~2 giây).

### 6. Hiển thị Toast bằng cách gọi `show()`

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 kiểm tra xem thông báo **Toast** có xuất hiện khi nhấn nút **Toast** hay không.



## 6.3 Chỉnh sửa trình xử lý nút **Count**

Bây giờ bạn sẽ chỉnh sửa phương thức `countUp()`—trình xử lý sự kiện của nút **Count** trong `MainActivity`—để hiển thị số đếm hiện tại sau khi nút **Count** được nhấn. Mỗi lần nhấn sẽ tăng giá trị đếm lên một đơn vị.

Mã cho trình xử lý này cần:

- Theo dõi giá trị số đếm khi nó thay đổi.
- Gửi giá trị số đếm đã cập nhật đến **TextView** để hiển thị.

Thực hiện các bước sau để chỉnh sửa trình xử lý sự kiện của nút **Count**:

1. Xác định vị trí phương thức `countUp()` đã được tạo:

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

2. Để theo dõi số đếm, bạn cần một biến thành viên `private`. Mỗi lần nhấn nút **Count** sẽ tăng giá trị của biến này. Thêm dòng sau vào phương thức `countUp()`, dòng này sẽ được tô đỏ và hiển thị biểu tượng bóng đèn màu đỏ:

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

Nếu biểu tượng bóng đèn đỏ không xuất hiện, hãy chọn biểu thức `mCount++`. Biểu tượng bóng đèn sẽ xuất hiện sau một lúc.

3. Nhấp vào biểu tượng bóng đèn đỏ và chọn **Create field 'mCount'** từ menu bật lên. Thao tác này sẽ tạo một biến thành viên `private` ở đầu `MainActivity`, và Android Studio sẽ tự động nhận diện kiểu dữ liệu là `int`:

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

4. Thay đổi khai báo biến thành viên **private** để khởi tạo giá trị ban đầu là 0:

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

5. Bên cạnh biến trên, bạn cũng cần một biến thành viên **private** để tham chiếu đến **TextView** có ID `show_count`, biến này sẽ được sử dụng trong trình xử lý sự kiện khi nhấn nút. Gọi biến này là `mShowCount`:

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

6. Sau khi khai báo `mShowCount`, bạn cần lấy tham chiếu đến **TextView** bằng cách sử dụng ID đã đặt trong tệp **layout**. Để chỉ lấy tham chiếu một lần duy nhất, hãy thực hiện trong phương thức `onCreate()`.
  - Trong một bài học khác, bạn sẽ tìm hiểu rằng phương thức `onCreate()` được sử dụng để **inflate layout**, có nghĩa là thiết lập nội dung hiển thị của màn hình dựa trên tệp XML layout.
  - Bạn cũng có thể sử dụng phương thức này để lấy tham chiếu đến các phần tử UI khác trong bố cục, chẳng hạn như **TextView**.

Xác định vị trí phương thức `onCreate()` trong `MainActivity`:

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

7. Thêm dòng `findViewById` vào cuối phương thức để lấy tham chiếu đến **TextView**:

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

Một **View**, giống như một chuỗi văn bản, là một tài nguyên có thể có **ID**. Lệnh gọi `findViewById` nhận ID của một **View** làm tham số và trả về đối tượng **View** tương ứng. Vì phương thức này trả về một đối tượng kiểu **View**, bạn cần ép kiểu kết quả về đúng loại **View** mà bạn mong đợi, trong trường hợp này là (**TextView**).

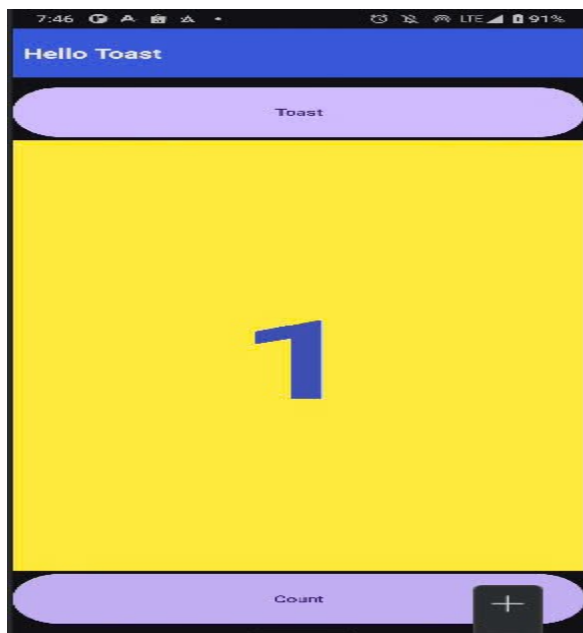
8. Sau khi đã gán `mShowCount` bằng tham chiếu đến **TextView**, bạn có thể sử dụng biến này để cập nhật văn bản hiển thị thành giá trị của `mCount`. Thêm đoạn mã sau vào phương thức `countUp()`:

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

Toàn bộ phương thức `countUp()` sau khi chỉnh sửa:

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

9. Chạy ứng dụng để xác minh rằng số đếm sẽ tăng lên mỗi khi bạn nhấn nút **Count**.



💡 **Mẹo:** Để tìm hiểu chuyên sâu về cách sử dụng **ConstraintLayout**, hãy xem hướng dẫn thực hành (Codelab) [Using ConstraintLayout to design your views](#).

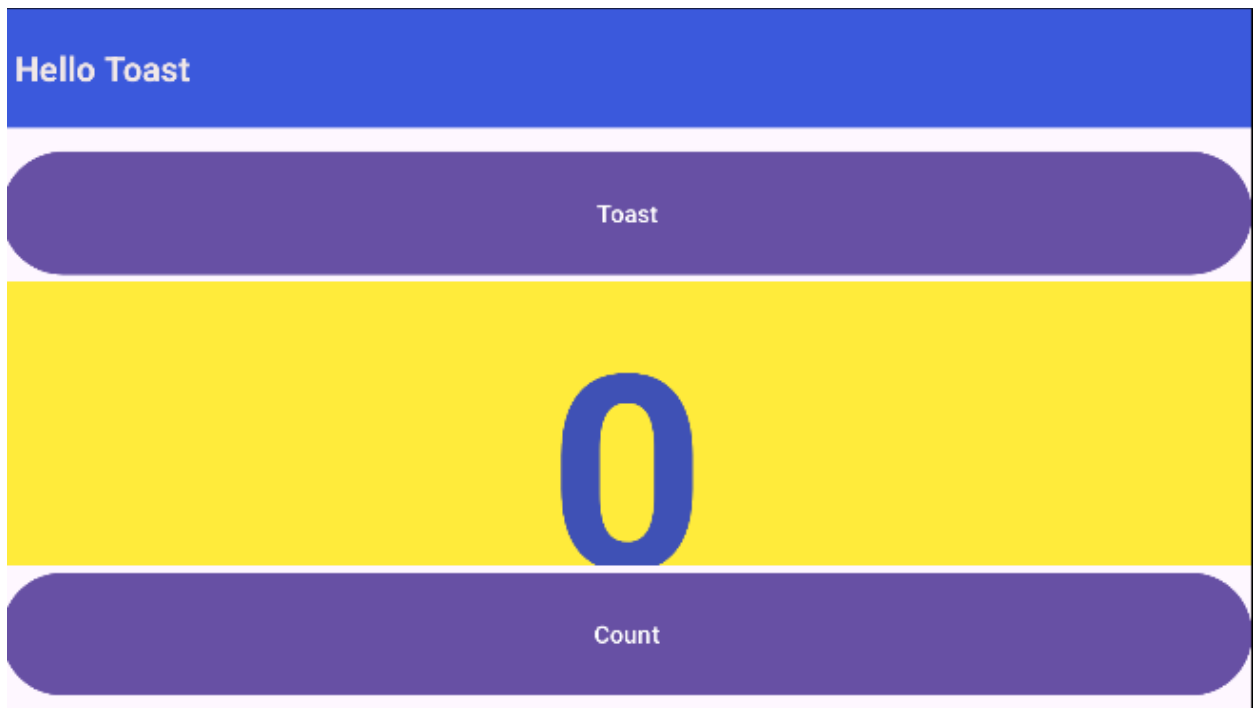
## Mã nguồn giải pháp

Dự án Android Studio: **HelloToast**.

## Thử thách lập trình

🔗 **Lưu ý:** Tất cả các thử thách lập trình đều là tùy chọn và không phải là điều kiện tiên quyết cho các bài học sau.

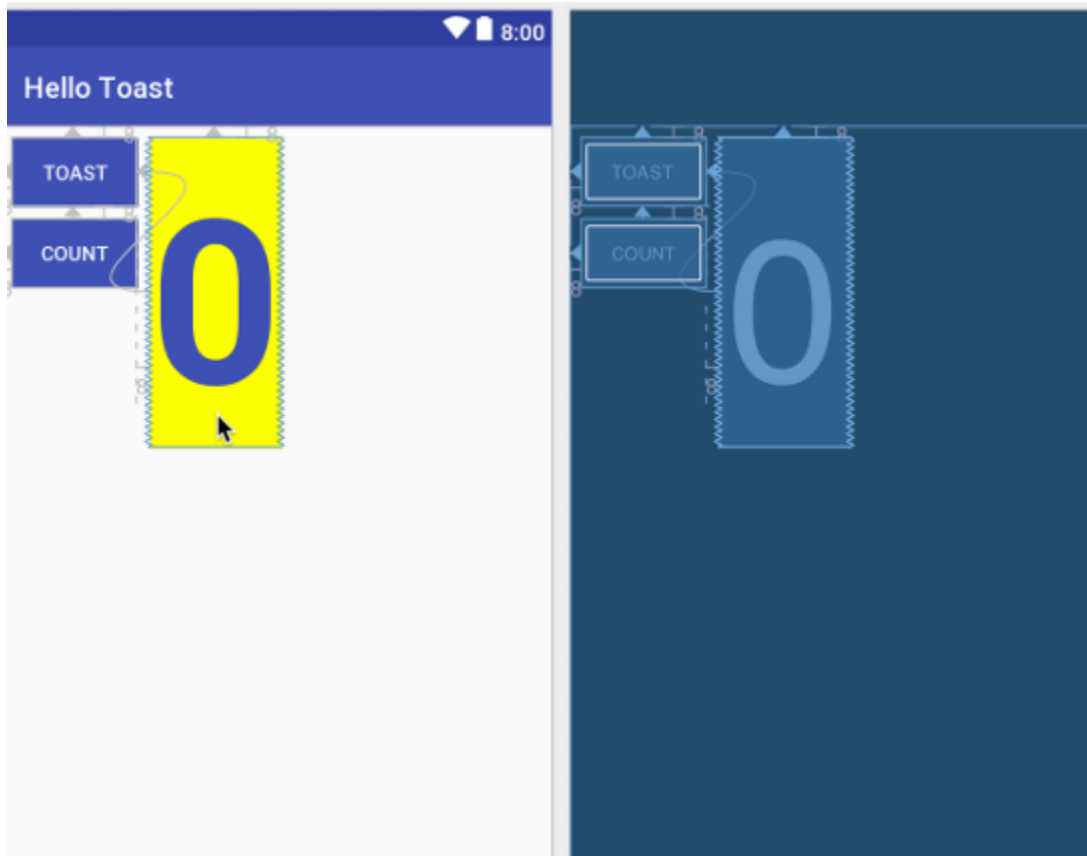
Ứng dụng **HelloToast** hiển thị tốt khi thiết bị hoặc trình giả lập ở chế độ dọc. Tuy nhiên, nếu bạn xoay thiết bị hoặc trình giả lập sang chế độ ngang, nút **Count** có thể bị chồng lên **TextView** ở phía dưới, như minh họa trong hình bên dưới.

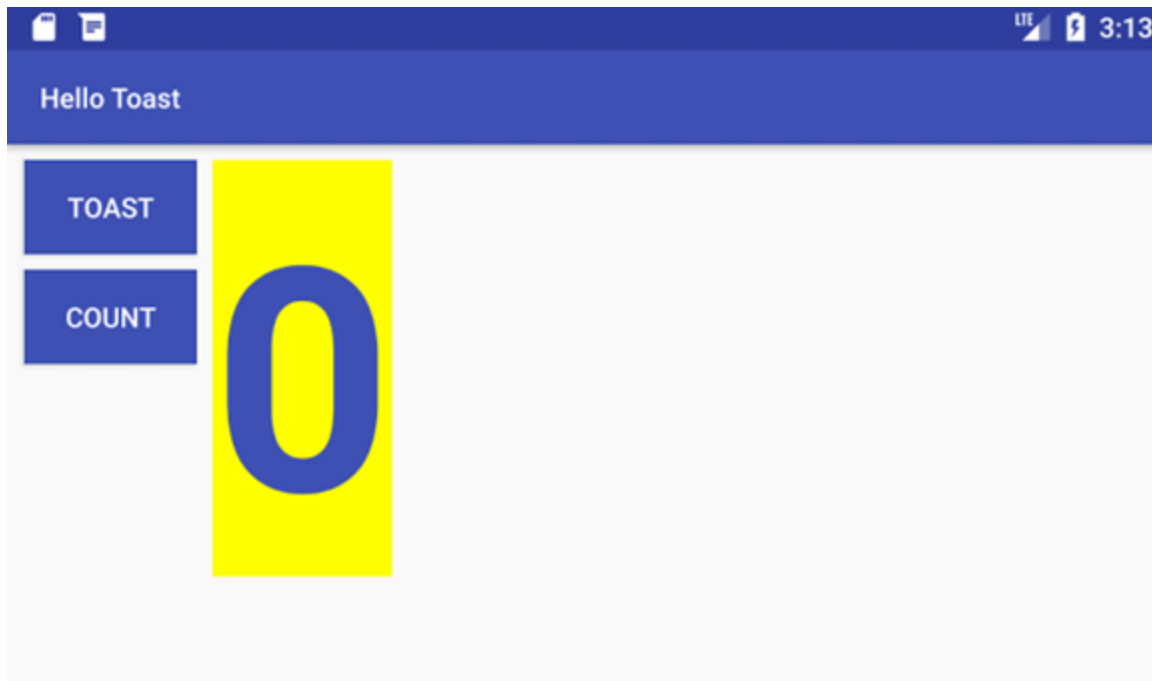


🔗 **Thử thách:** Hãy chỉnh sửa bố cục để ứng dụng hiển thị tốt ở cả chế độ ngang và dọc:

1. Trên máy tính, tạo một bản sao của thư mục dự án **HelloToast** và đổi tên thành **HelloToastChallenge**.
2. Mở **HelloToastChallenge** trong **Android Studio** và tiến hành refactor. (Xem Phụ lục: Tiện ích để biết hướng dẫn sao chép và refactor dự án.)

3. Chỉnh sửa bố cục sao cho nút **Toast** và **Count** xuất hiện ở bên trái màn hình, như trong hình minh họa. **TextView** xuất hiện bên cạnh nhưng chỉ đủ rộng để hiển thị nội dung của nó. (Gợi ý: Sử dụng `wrap_content`.)
4. Chạy ứng dụng ở cả hai chế độ ngang và dọc để kiểm tra kết quả. 🚀





Mã nguồn giải pháp cho thử thách

📁 Dự án Android Studio: *HelloToastChallenge*

## Tóm tắt

### ◆ View, ViewGroup và Layouts:

- Tất cả các phần tử giao diện người dùng (UI elements) đều là các lớp con của lớp `View`, do đó chúng kế thừa nhiều thuộc tính từ lớp cha `View`.
- Các phần tử `View` có thể được nhóm lại bên trong một `ViewGroup`, đóng vai trò như một container.
- Mỗi quan hệ giữa chúng là **cha - con**, trong đó **cha** là một `ViewGroup`, còn **con** có thể là một `View` hoặc một `ViewGroup` khác.
- **Phương thức `onCreate()`** được sử dụng để **inflate layout**, tức là thiết lập giao diện của màn hình theo tệp XML. Bạn cũng có thể sử dụng nó để lấy tham chiếu đến các phần tử giao diện trong layout.
- Một **View**, giống như một chuỗi văn bản, là một tài nguyên có thể có **ID**. Lệnh `findViewById` nhận ID của một `View` làm tham số và trả về **đối tượng View** tương ứng.

Sử dụng trình chỉnh sửa layout:

- Nhấn vào tab **Design** để thao tác với các thành phần và bố cục, hoặc vào tab **Text** để chỉnh sửa mã XML của layout.

- Trong tab **Design**, bảng **Palette** hiển thị các thành phần UI mà bạn có thể sử dụng trong layout, còn bảng **Component Tree** hiển thị cấu trúc phân cấp của các thành phần UI.
- **Bảng thiết kế (Design pane)** và **bảng bố cục (Blueprint pane)** hiển thị các thành phần UI trong layout.

UI.

- Tab **Attributes** hiển thị bảng **Attributes** để thiết lập thuộc tính cho các thành phần UI.
- **Tay nắm ràng buộc (Constraint handle):** Nhấp vào tay nắm ràng buộc, được hiển thị dưới dạng **vòng tròn** ở mỗi cạnh của một phần tử, sau đó kéo nó đến một tay nắm ràng buộc khác hoặc đến ranh giới của phần tử cha để tạo ràng buộc. Ràng buộc này được biểu thị bằng **đường zigzag**.
- **Tay nắm thay đổi kích thước (Resizing handle):** Bạn có thể kéo các **tay nắm hình vuông** để thay đổi kích thước của phần tử. Khi kéo, tay nắm sẽ chuyển thành **góc xiên**.
- Khi được bật, công cụ **Autoconnect** sẽ tự động tạo **hai hoặc nhiều ràng buộc (constraints)** cho một phần tử giao diện với **bố cục cha (parent layout)**. Sau khi bạn kéo phần tử vào bố cục, nó sẽ tạo các ràng buộc dựa trên **vị trí của phần tử**.
- Bạn có thể **xóa ràng buộc (constraints)** khỏi một phần tử bằng cách **chọn phần tử đó** và di chuột qua để hiển thị nút **Clear Constraints**. Nhấp vào nút này để **xóa tất cả ràng buộc** trên phần tử đã chọn. Để xóa một ràng buộc cụ thể, hãy nhấp vào **tay cầm (handle)** của ràng buộc đó.
- **Bảng Thuộc tính (Attributes pane)** 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 (UI). Ngoài ra, nó còn bao gồm một **bảng định kích thước hình vuông** gọi là **trình kiểm tra chế độ xem (view inspector)** ở phía trên. Các biểu tượng bên trong hình vuông này biểu thị **cài đặt chiều cao và chiều rộng** của phần tử.

Thiết lập chiều rộng và chiều cao của bố cục:

### Thuộc tính chiều rộng và chiều cao của bố cục:

Các thuộc tính `layout_width` và `layout_height` sẽ thay đổi khi bạn điều chỉnh kích thước chiều cao và chiều rộng trong **view inspector**. Trong `ConstraintLayout`, các thuộc tính này có thể nhận một trong ba giá trị sau:

- **match\_constraint:** Mở rộng phần tử để lấp đầy phần tử cha theo chiều rộng hoặc chiều cao—tối đa đến khoảng lề (nếu có thiết lập).
- **wrap\_content:** Thu nhỏ phần tử sao cho vừa đủ để chứa nội dung bên trong. Nếu không có nội dung, phần tử sẽ trở nên vô hình.
- **\*\*Sử dụng một số cố định (đơn vị dp - density-independent pixels)** để chỉ định kích thước cố định, có thể điều chỉnh theo kích thước màn hình của thiết bị.



Trích xuất tài nguyên chuỗi:

Thay vì mã hóa cứng chuỗi văn bản, một thực hành tốt là sử dụng tài nguyên chuỗi (`string resources`). Thực hiện theo các bước sau:

1. Nhấp một lần vào chuỗi văn bản cần trích xuất, nhấn `Alt-Enter` (`Option-Enter` trên Mac), và chọn **Extract string resources** từ menu bật lên.
2. Đặt **tên tài nguyên** (`Resource name`).
3. Nhấp **OK**. Điều này sẽ tạo một tài nguyên chuỗi trong tệp `values/res/string.xml`, và chuỗi trong mã nguồn sẽ được thay thế bằng một tham chiếu đến tài nguyên đó: `@string/button_label_toast`.

Xử lý sự kiện nhấp:

- **Trình xử lý nhấp (Click handler)** là một phương thức được gọi khi người dùng nhấp hoặc chạm vào một phần tử UI.
- Để chỉ định trình xử lý nhấp cho một phần tử UI như `Button`, bạn có thể:
  - Nhập tên phương thức vào trường `onClick` trong **tab Design** của **Attributes pane**.
  - Hoặc thêm thuộc tính `android:onClick` vào phần tử UI trong **XML editor**.
- Tạo trình xử lý nhấp trong `MainActivity` bằng cách sử dụng tham số `View`. Ví dụ:
- ```
public void showToast(View view) {
```
- ```
    // Xử lý sự kiện nhấp
```
- ```
}
```
- Bạn có thể tìm hiểu thêm về tất cả các thuộc tính của **Button** trong **Button class documentation**, và các thuộc tính của **TextView** trong **TextView class documentation**.

Hiển thị thông báo Toast:

**Toast** cung cấp một cách để hiển thị một thông báo đơn giản trong một cửa sổ bật lên nhỏ. Nó chỉ chiếm một lượng không gian vừa đủ cho nội dung thông báo. Để tạo một **Toast**, thực hiện theo các bước sau:

1. Gọi phương thức `makeText()` của lớp `Toast`.
2. Cung cấp **context của Activity** trong ứng dụng và thông điệp cần hiển thị (chẳng hạn như một tài nguyên chuỗi).
3. Cung cấp **thời gian hiển thị**, ví dụ: `Toast.LENGTH_SHORT` để hiển thị trong thời gian ngắn. Thời gian hiển thị có thể là `Toast.LENGTH_LONG` hoặc `Toast.LENGTH_SHORT`.
4. Hiển thị **Toast** bằng cách gọi phương thức `show()`.

**Khái niệm liên quan**

Tài liệu liên quan đến khái niệm này có trong **1.2: Layouts and resources for the UI**.

# **Tìm hiểu thêm**

Tài liệu dành cho nhà phát triển Android:

- 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

## **Khác:**

- Codelabs: Using ConstraintLayout to design your views
- Vocabulary words and concepts glossary

Codelab tiếp là Android fundamentals 1.2 Part B: The layout editor .

## **Part B: Trình chỉnh sửa layout (The layout editor)**

**1.3) Trình chỉnh sửa bố cục**

**1.4) Văn bản và các chế độ cuộn**

**1.5) Tài nguyên có sẵn**

## **Bài 2) Activities**

**2.1) Activity và Intent**

**2.2) Vòng đời của Activity và trạng thái**

**2.3) Intent ngầm định**

## **Bài 3) Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ**

**3.1) Trình gỡ lỗi**

**3.2) Kiểm thử đơn vị**

**3.3) Thư viện hỗ trợ**

## **CHƯƠNG 2. TRẢI NGHIỆM NGƯỜI DÙNG**

### **Bài 1) Tương tác người dùng**

- 1.1) Hình ảnh có thể chọn**
- 1.2) Các điều khiển nhập liệu**
- 1.3) Menu và bộ chọn**
- 1.4) Điều hướng người dùng**
- 1.5) RecyclerView**

### **Bài 2) Trải nghiệm người dùng thú vị**

- 2.1) Hình vẽ, định kiểu và chủ đề**
- 2.2) Thẻ và màu sắc**
- 2.3) Bố cục thích ứng**

### **Bài 3) Kiểm thử giao diện người dùng**

- 3.1) Espresso cho việc kiểm tra UI**

## **CHƯƠNG 3. LÀM VIỆC TRONG NỀN**

### **Bài 1) Các tác vụ nền**

- 1.1) AsyncTask**
- 1.2) AsyncTask và AsyncTaskLoader**
- 1.3) Broadcast receivers**

### **Bài 2) Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền**

- 2.1) Thông báo**
- 2.2) Trình quản lý cảnh báo**
- 2.3) JobScheduler**

## **CHƯƠNG 4. LƯU DỮ LIỆU NGƯỜI DÙNG**

### **Bài 1) Tùy chọn và cài đặt**

**1.1) Shared preferences**

**1.2) Cài đặt ứng dụng**

### **Bài 2) Lưu trữ dữ liệu với Room**

**2.1) Room, LiveData và ViewModel**

**2.2) Room, LiveData và ViewModel**