**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 4](#_Toc190855147)

[Bài 1) Tạo ứng dụng đầu tiên 4](#_Toc190855148)

[1.1) Android Studio và Hello World 4](#_Toc190855149)

[1.2) Giao diện người dùng tương tác đầu tiên 5](#_Toc190855150)

[1.3) Trình chỉnh sửa bố cục 5](#_Toc190855151)

[1.4) Văn bản và các chế độ cuộn 5](#_Toc190855152)

[1.5) Tài nguyên có sẵn 5](#_Toc190855153)

[Bài 2) Activities 5](#_Toc190855154)

[2.1) Activity và Intent 5](#_Toc190855155)

[2.2) Vòng đời của Activity và trạng thái 5](#_Toc190855156)

[2.3) Intent ngầm định 5](#_Toc190855157)

[Bài 3) Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ 5](#_Toc190855158)

[3.1) Trình gỡ lỗi 5](#_Toc190855159)

[3.2) Kiểm thử đơn vị 5](#_Toc190855160)

[3.3) Thư viện hỗ trợ 5](#_Toc190855161)

[CHƯƠNG 2. Trải nghiệm người dùng 6](#_Toc190855162)

[Bài 1) Tương tác người dùng 6](#_Toc190855163)

[1.1) Hình ảnh có thể chọn 6](#_Toc190855164)

[1.2) Các điều khiển nhập liệu 6](#_Toc190855165)

[1.3) Menu và bộ chọn 6](#_Toc190855166)

[1.4) Điều hướng người dùng 6](#_Toc190855167)

[1.5) RecycleView 6](#_Toc190855168)

[Bài 2) Trải nghiệm người dùng thú vị 6](#_Toc190855169)

[2.1) Hình vẽ, định kiểu và chủ đề 6](#_Toc190855170)

[2.2) Thẻ và màu sắc 6](#_Toc190855171)

[2.3) Bố cục thích ứng 6](#_Toc190855172)

[Bài 3) Kiểm thử giao diện người dùng 6](#_Toc190855173)

[3.1) Espresso cho việc kiểm tra UI 6](#_Toc190855174)

[CHƯƠNG 3. Làm việc trong nền 6](#_Toc190855175)

[Bài 1) Các tác vụ nền 6](#_Toc190855176)

[1.1) AsyncTask 6](#_Toc190855177)

[1.2) AsyncTask và AsyncTaskLoader 6](#_Toc190855178)

[1.3) Broadcast receivers 6](#_Toc190855179)

[Bài 2) Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền 6](#_Toc190855180)

[2.1) Thông báo 6](#_Toc190855181)

[2.2) Trình quản lý cảnh báo 6](#_Toc190855182)

[2.3) JobScheduler 6](#_Toc190855183)

[CHƯƠNG 4. Lưu dữ liệu người dùng 7](#_Toc190855184)

[Bài 1) Tùy chọn và cài đặt 7](#_Toc190855185)

[1.1) Shared preferences 7](#_Toc190855186)

[1.2) Cài đặt ứng dụng 7](#_Toc190855187)

[Bài 2) Lưu trữ dữ liệu với Room 7](#_Toc190855188)

[2.1) Room, LiveData và ViewModel 7](#_Toc190855189)

[2.2) Room, LiveData và ViewModel 7](#_Toc190855190)

3.1) Trinfh gowx loi ……………………………………………………………………...

# LÀM QUEN

## Tạo ứng dụng đầu tiên

### 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.

A screenshot of a computer

AI-generated content may be incorrect.

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.

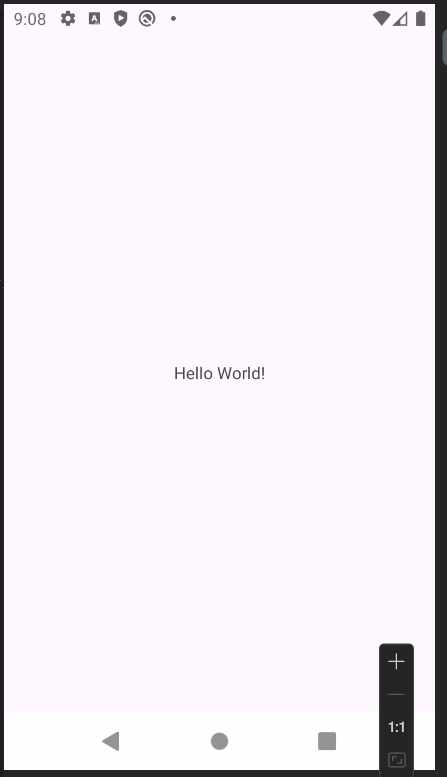
Những gì bạn sẽ làm

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

Đây là giao diện của 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 một code editor hiện đại và một bộ mẫu ứng dụng. Ngoài ra, nó còn chứa các công cụ hỗ trợ phát triển, gỡ lỗi, kiểm thử và tối ưu hiệu suất, giúp việc phát triển ứng dụng trở nên nhanh và dễ hơn. Bạn có thể thử ứng dụng của mình trên nhiều trình giả lập được cấu hình sẵn hoặc trên thiết bị di động của mình, xây dựng ứng dụng hoàn chỉnh và đăng lên cửa hàng Google Play.

Android Studio có sẵn cho các máy tính chạy Windows, Linux và macOS. Phiên bản mới nhất của OpenJDK (Java Development Kit) được tích hợp sẵn trong Android Studio.

Để bắt đầu với Android Studio, trước tiên hãy kiểm tra các yêu cầu hệ thống để đảm bảo thiết bị của bạn đáp ứng đủ điều kiện. Quá trình cài đặt tương tự trên tất cả các nền tảng, chỉ có một số điểm khác biệt nhỏ được lưu ý dưới đây.

1. Truy cập trang web của Android Developers và làm theo hướng dẫn để tải xuống và cài đặt Android Studio.
2. Chấp nhận các cấu hình mặc định trong tất cả các bước và đảm bảo rằng tất cả các thành phần cần thiết đều được chọn để cài đặt.
3. Sau khi cài đặt xong, Setup Wizard sẽ tải xuống và cài đặt thêm một số thành phần khác, bao gồm Android SDK. Hãy kiên nhẫn vì quá trình này có thể mất thời gian tùy thuộc vào tốc độ Internet của bạn, và một số bước có thể lặp lại.
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.

**Nhiệm vụ 2: Tạo ứng dụng Hello World**

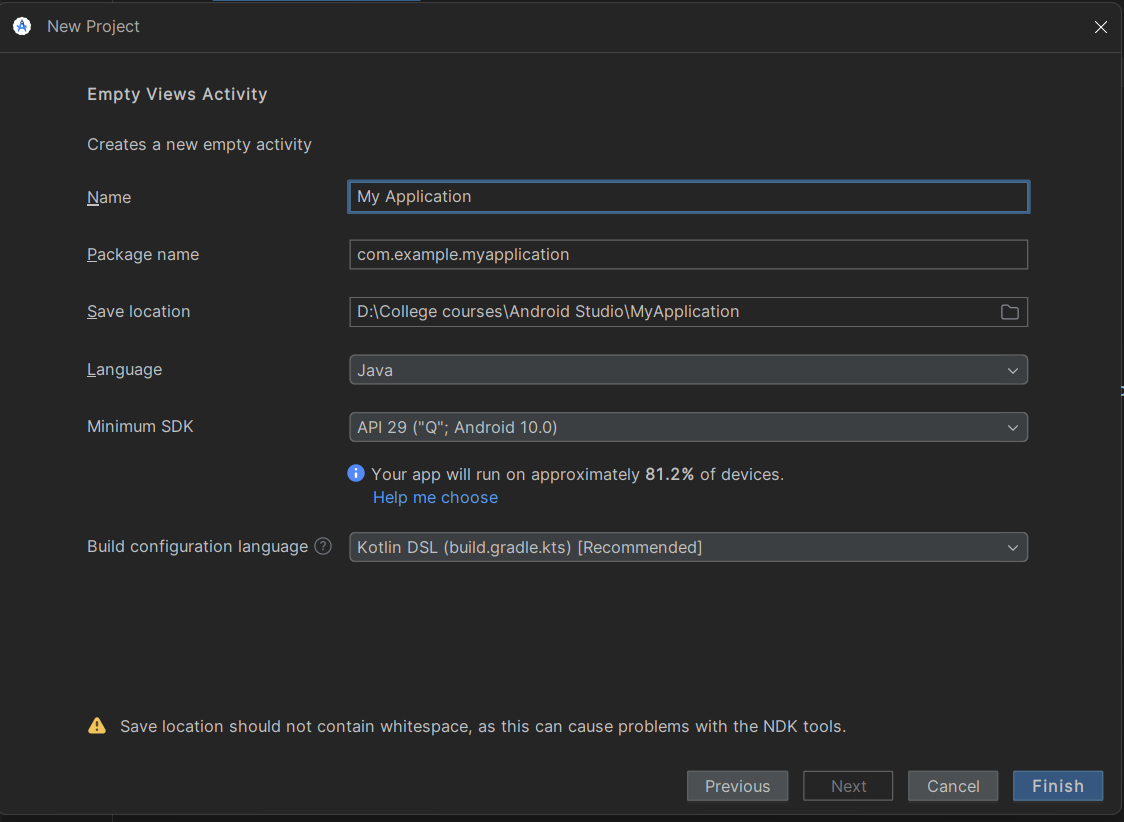
Trong nhiệm vụ này, bạn sẽ tạo một ứng dụng hiển thị dòng chữ **"Hello World"** để kiểm tra xem Android Studio đã được cài đặt đúng chưa, đồng thời làm quen với các bước cơ bản trong quá trình phát triển ứng dụng với Android Studio.

**2.1 Tạo dự án ứng dụng**

1. Mở **Android Studio** nếu chưa mở.

2. Ở cửa sổ chào mừng **Welcome to Android Studio**, nhấp vào **Start a new Android Studio project** (Bắt đầu một dự án Android Studio mới).

3. Trong cửa sổ **Create Android Project**, nhập **Hello World** vào ô **Application name** (Tên ứng dụng).



4. Xác nhận rằng thư mục mặc định trong **Project location** là nơi bạn muốn lưu trữ ứng dụng **Hello World** và các dự án khác trong Android Studio. Nếu cần, bạn có thể thay đổi sang thư mục mong muốn.

5. Chấp nhận giá trị mặc định **android.example.com** cho **Company Domain**, hoặc nhập một tên miền tùy chỉnh.

Nếu bạn không có kế hoạch xuất bản ứng dụng, bạn có thể giữ nguyên giá trị mặc định. Lưu ý: Việc thay đổi tên gói (package name) sau này sẽ tốn thêm công sức.

6. Bỏ chọn các tùy chọn **Include C++ support** và **Include Kotlin support**, sau đó nhấp **Next**.

7. Ở màn hình **Target Android Devices**, đảm bảo rằng **Phone and Tablet** đã được chọn. Kiểm tra rằng **API 29: Android 10.0 “Q”** được đặt làm **Minimum SDK**. Nếu không, hãy sử dụng menu thả xuống để chọn đúng phiên bản này.

8. Đảm bảo rằng tùy chọn **Generate Layout file** được chọn. Mặc định, tên tệp bố cục sẽ là **activity\_main**. Bạn có thể thay đổi nếu muốn, nhưng bài học này sẽ sử dụng **activity\_main**.

9. Đảm bảo rằng tùy chọn **Backwards Compatibility (App Compat)** được chọn.Tùy chọn này giúp ứng dụng của bạn tương thích với các phiên bản Android cũ hơn.

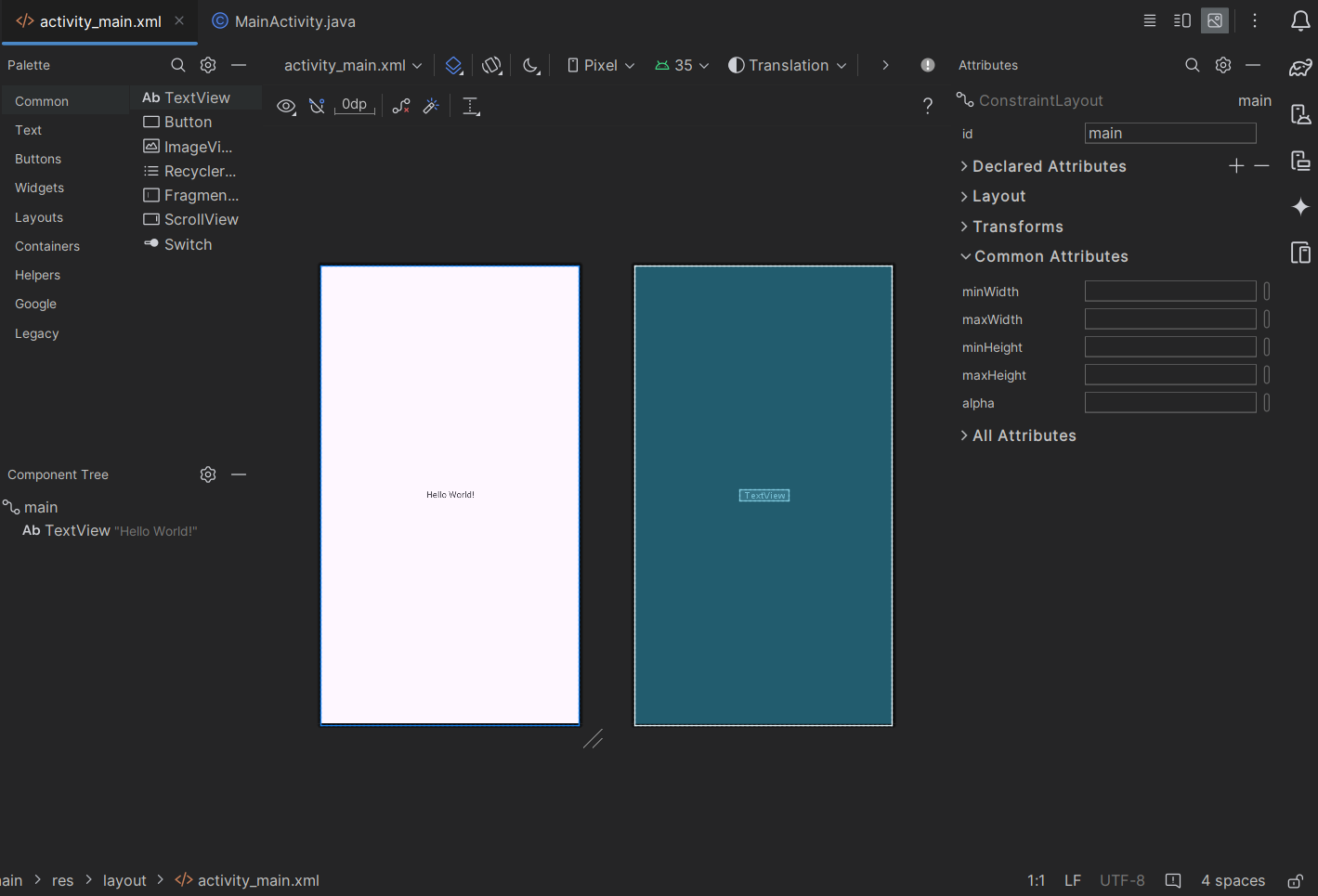
10. Nhấp **Finish**

Android Studio 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).

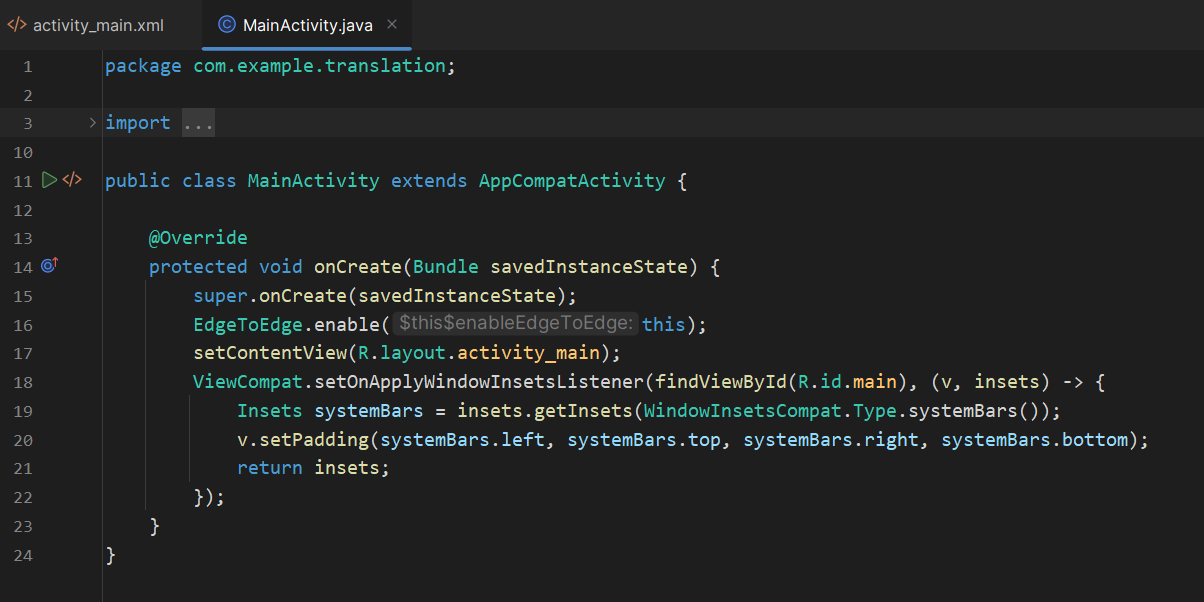
Mẹo: Xem trang tài liệu "Configure your build" dành cho nhà phát triển để biết thông tin chi tiết. Bạn cũng có thể thấy một thông báo "Tip of the day" với các phím tắt và mẹo hữu ích khác. Nhấp **Close** để đóng thông báo này.

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

1. Nhấp vào tab **activity\_main.xml** để mở trình chỉnh sửa bố cục.
2. Nhấp vào tab **Design** trong trình chỉnh sửa bố cục, nếu chưa được chọn, để hiển thị bản xem trước giao diện như hình minh họa dưới đây.



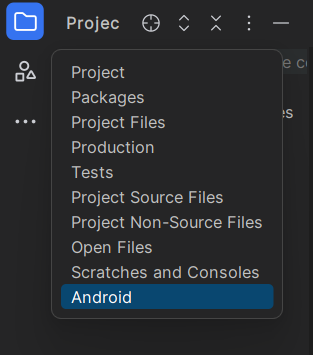
1. Nhấp vào tab **MainActivity.java** để mở trình chỉnh sửa mã như hình minh họa dưới đây.



**2.2 Khám phá Project > Android pane**

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

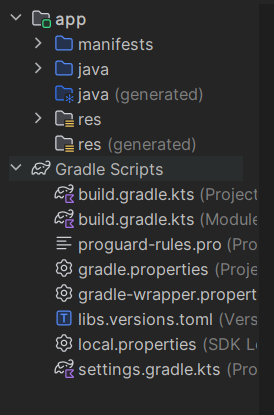
1. Nếu chưa được chọn, nhấp vào tab **Project** trong cột tab dọc bên trái của cửa sổ Android Studio. Ngăn **Project** sẽ xuất hiện.
2. Để xem dự án theo cấu trúc chuẩn của Android, chọn **Android** từ menu thả xuống ở đầu ngăn **Project**, như hình minh họa bên dưới.

****

**2.3 Khám phá thư mục Gradle Scripts**

Hệ thống xây dựng Gradle trong Android Studio giúp bạn dễ dàng thêm các tệp nhị phân bên ngoài hoặc các mô-đun thư viện khác vào dự án 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 hình minh họa bên dưới.



Thực hiện các bước sau để tìm hiểu về hệ thống Gradle:

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

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

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

Đây là nơi chứa các tùy chọn cấu hình chung cho tất cả các mô-đun trong dự án. Mọi dự án trong Android Studio đều có một tệp Gradle build cấp cao nhất. Thông thường, bạn không cần thay đổi tệp này, nhưng hiểu nội dung của nó vẫn rất hữu ích.

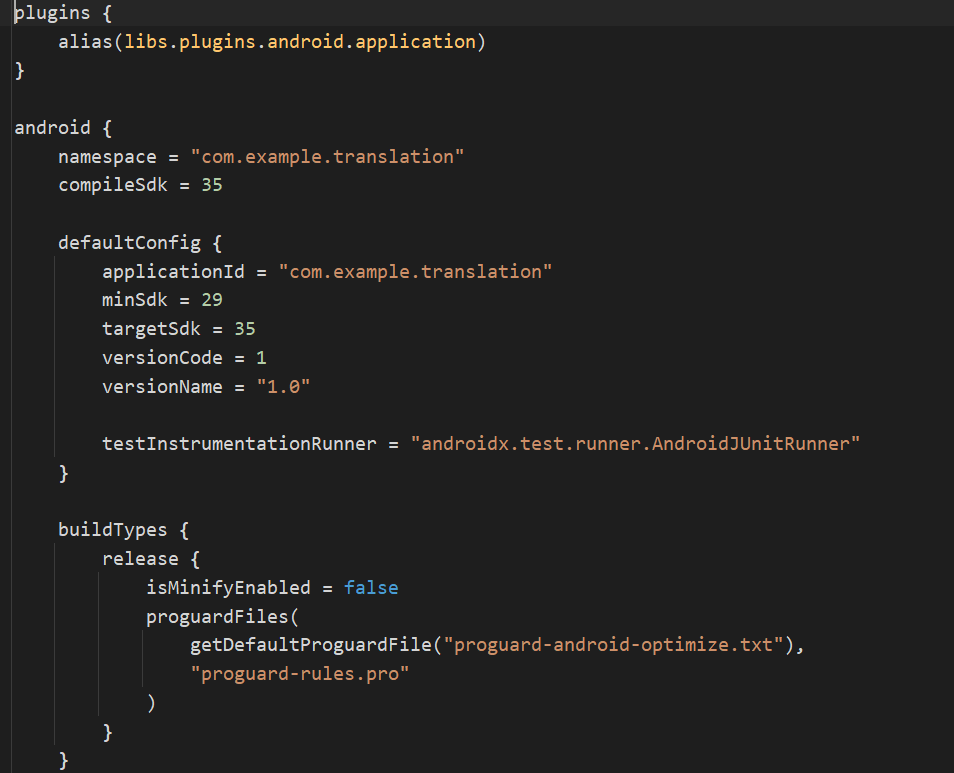
Theo mặc định, tệp build cấp cao nhất sử dụng khối **buildscript** để xác định các kho lưu trữ Gradle và dependencies chung cho tất cả các mô-đun trong dự án. Khi một dependency không phải là thư viện cục bộ hoặc tệp trong cây thư mục, Gradle sẽ tìm kiếm nó trong các kho lưu trữ trực tuyến được chỉ định trong khối **repositories** của tệp này. Mặc định, các dự án Android Studio mới sẽ khai báo **JCenter** và **Google** (bao gồm cả kho lưu trữ Google Maven) làm vị trí kho lưu trữ.

1. Tìm tệp **build.gradle (Module: app)**

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

Tệp này thường được chỉnh sửa nhiều nhất khi thay đổi các cấu hình ở cấp ứng dụng, chẳng hạn như khai báo dependencies trong phần **dependencies**. Bạn có thể khai báo một thư viện dependency bằng nhiều cách khác nhau. Mỗi cách cung cấp cho Gradle một hướng dẫn khác nhau về cách sử dụng thư viện đó. Ví dụ, câu lệnh: implementation fileTree(dir: 'libs', include: ['\*.jar']) sẽ thêm tất cả các tệp **".jar"** bên trong thư mục **libs** làm dependencies.

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



1. Nhấn vào hình tam giác để tắt Gradle Scripts.

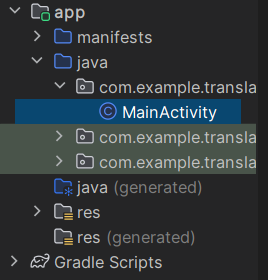
**2.4 Khám phá thư mục app và res**

Tất cả mã nguồn và tài nguyên (resources) của ứng dụng được lưu trữ trong các thư mục **app** và **res**.

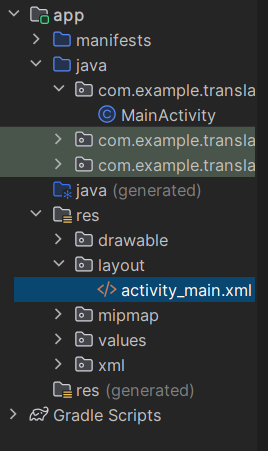
1. Mở rộng thư mục **app**, sau đó mở tiếp thư mục **java** và thư mục con **com.example.android.helloworld** để tìm tệp **MainActivity.java**. Nhấp đúp vào tệp **MainActivity.java** để mở nó trong trình chỉnh sửa mã (code editor).

Thư mục **java** chứa các tệp lớp Java trong ba thư mục con, như minh họa trong hình trên.Thư mục **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 thử (testing) và sẽ được mô tả trong một bài học khác.Đối với ứng dụng **Hello World**, chỉ có một gói duy nhất và nó chứa tệp **MainActivity.java**.Tên của Activity đầu tiên (màn hình đầu tiên mà người dùng nhìn thấy), đồng thời khởi tạo các tài nguyên ở cấp ứng dụng, thường được đặt là **MainActivity** (phần mở rộng tệp bị ẩn trong ngăn **Project > Android**).

1. Mở rộng thư mục **res**, sau đó mở tiếp thư mục **layout**, và nhấp đúp vào tệp **activity\_main.xml** để mở nó trong trình chỉnh sửa bố cục (layout editor).



Thư mục **res** chứa các tài nguyên như bố cục (layouts), chuỗi ký tự (strings), và hình ảnh (images).Một **Activity** thường được liên kết với một bố cục giao diện người dùng (UI) được định nghĩa dưới dạng tệp XML. Thông thường, tệp này được đặt tên theo tên của **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 những thông tin này trước khi có thể chạy bất kỳ 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 sau, bạn sẽ chỉnh sửa tệp này để thêm các tính năng và quyền truy cập (permissions). Để tìm hiểu thêm, hãy xem phần giới thiệu về **App Manifest Overview**.

**Nhiệm vụ 3: Sử dụng thiết bị ảo (emulator)**

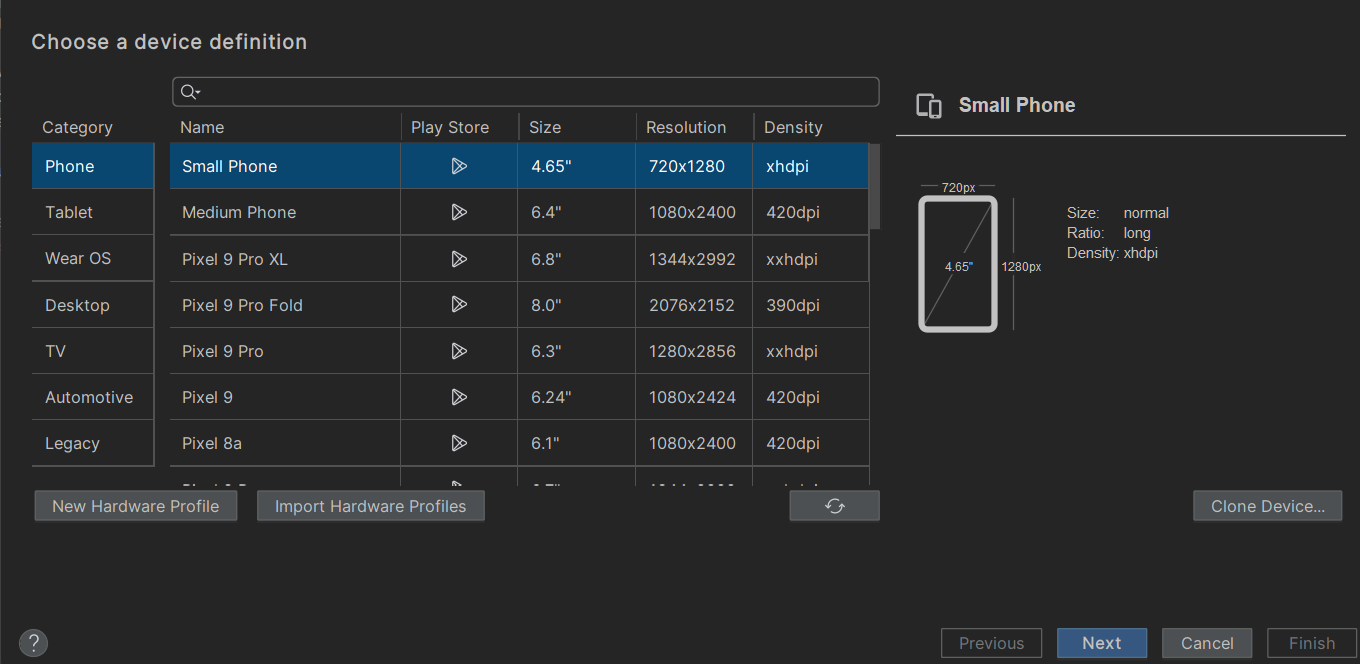
Trong nhiệm vụ này, bạn sẽ sử dụng **Android Virtual Device (AVD) Manager** để tạo một thiết bị ảo (còn gọi là **emulator**) mô phỏng cấu hình của một thiết bị Android cụ thể, sau đó chạy ứng dụng trên thiết bị ảo đó. Lưu ý rằng **Android Emulator** có các yêu cầu bổ sung so với các yêu cầu hệ thống cơ bản của **Android Studio**.

Sử dụng **AVD Manager**, bạn có thể định nghĩa các đặc điểm phần cứng của thiết bị, thiết lập mức API, dung lượng lưu trữ, giao diện (skin) và các thuộc tính khác, lưu cấu hình đó dưới dạng một thiết bị ảo. Với thiết bị ảo, bạn có thể kiểm thử ứng dụng trên nhiều cấu hình thiết bị khác nhau (ví dụ: điện thoại và máy tính bảng) với các mức API khác nhau, mà không cần sử dụng thiết bị vật lý.

3.1 Tạo một thiết bị ảo Android (AVD)

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

1. Trong **Android Studio**, chọn **Tools > Android > AVD Manager**, hoặc nhấp vào biểu tượng **AVD Manager** trên thanh công cụ. Màn hình **Your Virtual Devices** sẽ xuất hiện. Nếu bạn đã tạo các thiết bị ảo trước đó, danh sách sẽ hiển thị chúng. Nếu chưa, danh sách sẽ trống.



1. Nhấp vào **+Create Virtual Device**. Cửa sổ **Select Hardware** sẽ xuất hiện, hiển thị danh sách các thiết bị phần cứng được cấu hình sẵn. Đối với mỗi thiết bị, bảng hiển thị các thông số sau: **Size**: Kích thước màn hình theo đường chéo (inch), **Resolution**: Độ phân giải màn hình (pixel), **Density**: Mật độ điểm ảnh (**dpi** - dots per inch).
2. Chọn một thiết bị như **Small Phone**, sau đó nhấp vào **Next**. Cửa sổ **System Image** sẽ xuất hiện.
3. Nhấp vào tab **Recommended** (nếu chưa được chọn), sau đó chọn phiên bản hệ điều hành Android mà bạn muốn chạy trên thiết bị ảo (ví dụ: **Oreo**).

Có nhiều phiên bản Android hơn những gì hiển thị trong tab **Recommended**. Bạn có thể xem thêm trong các tab **x86 Images** và **Other Images**.

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

1. Sau khi chọn phiên bản hệ điều hành, nhấp vào **Next**.Cửa sổ **Android Virtual Device (AVD)** sẽ xuất hiện. Tại đây, bạn có thể thay đổi tên của thiết bị ảo (**AVD**) nếu muốn.Kiểm tra lại cấu hình của bạn, sau đó nhấp vào **Finish** để hoàn tất.

3.2 Chạy ừng dụng trên máy ảo

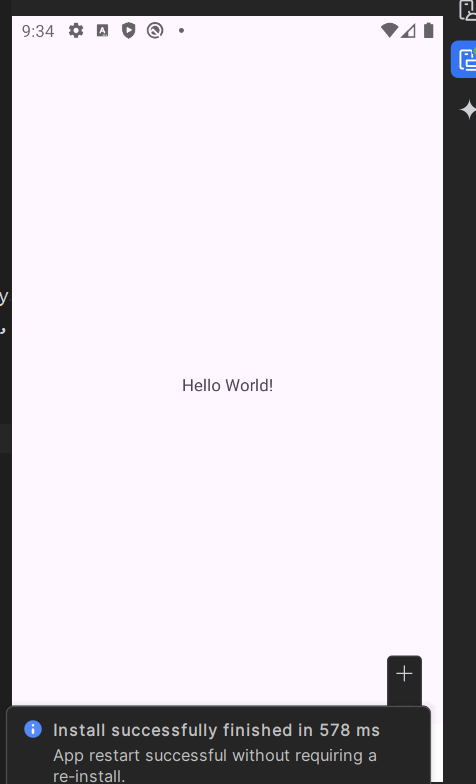
Ở nhiệm vụ này, bạn sẽ chạy ứng dụng Hello World

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 thuộc vào tốc độ của máy tính, quá trình này có thể mất một chút thời gian.

Bạn sẽ thấy ứng dụng **Hello World** hiển thị trên màn hình trình giả lập.



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

Trong nhiệm vụ này, bạn sẽ thay đổi một số cấu hình trong tệp **build.gradle(Module:app)** để học cách chỉnh sửa và đồng bộ chúng với dự án **Android Studio** của mình.

5.1 Thay đổi phiên bản SDK tối thiểu cho ứng dụng

Thực hiện các bước sau:

1. Mở rộng thư mục **Gradle Scripts** (nếu chưa mở), sau đó nhấp đúp vào tệp **build.gradle(Module:app)**.
   * Nội dung tệp sẽ xuất hiện trong trình chỉnh sửa mã (**code editor**).
2. Trong khối **defaultConfig**, thay đổi giá trị của **minSdkVersion** thành **17**, như ví dụ dưới đây (ban đầu được đặt là 15).



5.2 Đồng bộ cấu hình Gradle mới

Khi bạn thực hiện thay đổi trong các tệp cấu hình **build** của dự án, **Android Studio** yêu cầu bạn đồng bộ các tệp dự án để nhập các thay đổi cấu hình và kiểm tra xem chúng có gây ra lỗi biên dịch hay không.

Để đồng bộ tệp dự án, bạn có thể: Nhấp vào **Sync Now** trên thanh thông báo xuất hiện sau khi chỉnh sửa (như trong hình trước đó).Hoặc nhấp vào biểu tượng **Sync Project with Gradle Files** trên thanh công cụ.

Sau khi quá trình đồng bộ Gradle hoàn tất, bạn sẽ thấy thông báo **Gradle build finished** xuất hiện ở góc dưới bên trái của cửa sổ **Android Studio**.

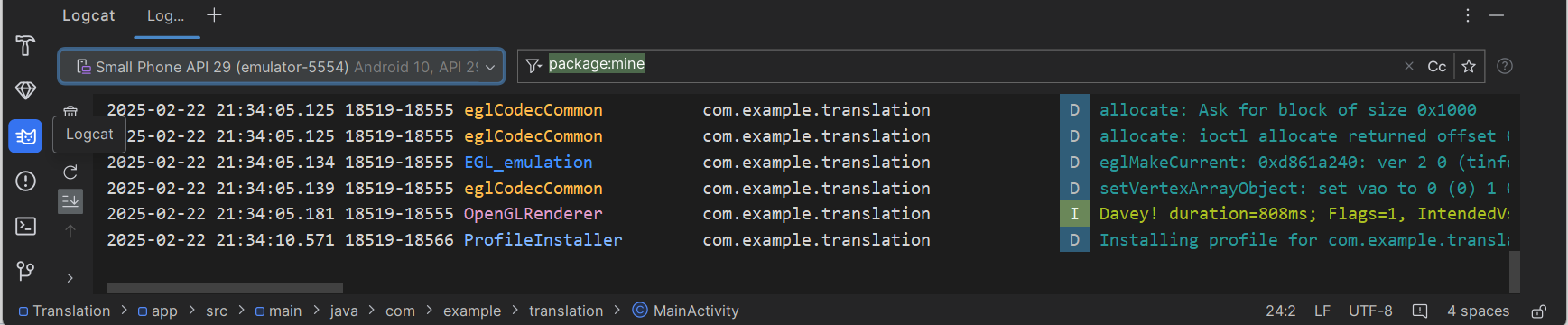
Để tìm hiểu sâu hơn về **Gradle**, hãy tham khảo tài liệu **Build System Overview** và **Configuring Gradle Builds**.

**Nhiệm vụ 6: Thêm câu lệnh log vào ứng dụng của bạn**

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

6.1 Quan sát Logcat pane

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



Trong hình trên:

1. **Tab Logcat** để mở và đóng **Logcat pane**, hiển thị thông tin về ứng dụng của bạn khi nó đang chạy. Nếu bạn thêm các câu lệnh **Log** vào ứng dụng, các thông báo **Log** sẽ xuất hiện tại đây.
2. **Menu cấp độ Log** được đặt thành **Verbose** (mặc định), hiển thị tất cả các thông báo **Log**. Các tùy chọn khác bao gồm **Debug, Error, Info** và **Warn**.

**6.2 Thêm câu lệnh log vào ứng dụng của bạn**

Bạn có thể thêm các câu lệnh **Log** vào mã của mình để hiển thị thông tin trong Logcat. Ví dụ:

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

Các phần của thông điệp bao gồm:  
● **Log**: Lớp Log dùng để gửi thông điệp log đến Logcat.  
● **d**: Cấp độ log Debug để lọc và hiển thị thông điệp log trong Logcat. Các cấp độ log khác bao gồm **e** cho Error, **w** cho Warn và **i** cho Info.  
● **"MainActivity"**: Đối số đầu tiên là một thẻ (tag) có thể được sử dụng để lọc thông điệp trong Logcat. Thông thường, đây là tên của Activity mà thông điệp được gửi từ đó. Tuy nhiên, bạn có thể đặt bất kỳ tên nào hữu ích cho quá trình gỡ lỗi.

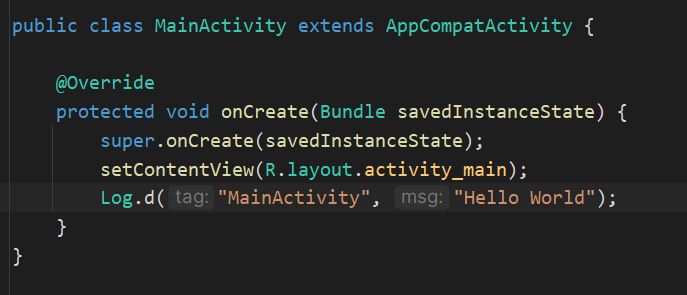
Theo quy ước, các thẻ log được định nghĩa là hằng số cho Activity:

private static final String LOG\_TAG = MainActivity.class.getSimpleName();

Làm theo các bước sau:

1. Mở ứng dụng **Hello World** trong Android Studio và mở tệp **MainActivity**.
2. Để tự động thêm các thư viện nhập khẩu (imports) không mơ hồ vào dự án của bạn (chẳng hạn như android.util.Log cần thiết để sử dụng Log), thực hiện:
   * Trên **Windows**: Chọn **File > Settings**.
   * Trên **macOS**: Chọn **Android Studio > Preferences**.
3. Chọn **Editor > General > Auto Import**.
   * Tích chọn tất cả các ô
   * Đặt **Insert imports on paste** thành **All**.
4. Nhấn **Apply**, sau đó nhấn **OK**.
5. Trong phương thức onCreate() của **MainActivity**, thêm dòng lệnh sau:

Log.d("MainActivity", "Hello World");



Phương thức onCreate() bâyh sẽ trông như sau

1. Nếu ngăn Logcat chưa mở, hãy nhấp vào tab **Logcat** ở cuối Android Studio để mở nó.
2. Kiểm tra xem tên của mục tiêu và tên gói của ứng dụng có đúng không.
3. Thay đổi **Log level** trong ngăn Logcat thành **Debug** (hoặc giữ nguyên **Verbose** vì có rất ít thông báo log).
4. Chạy ứng dụng của bạn.

Result



**Tóm tắt**

 Để cài đặt **Android Studio**, hãy truy cập trang Android Studio và làm theo hướng dẫn để tải xuống và cài đặt.

 Khi tạo một ứng dụng mới, hãy đảm bảo rằng **API 15: Android 4.0.3 IceCreamSandwich** được đặt làm **Minimum SDK**.

 Để xem cấu trúc Android của ứng dụng trong **Project pane**, nhấp vào tab **Project** trong cột tab dọc, sau đó chọn **Android** trong menu thả xuống ở trên cùng.

 Chỉnh sửa tệp **build.gradle(Module:app)** khi cần thêm thư viện mới vào dự án hoặc thay đổi phiên bản thư viện.

 Mã nguồn và tài nguyên của ứng dụng được đặt trong thư mục **app** và **res**. Thư mục **java** chứa các **Activity**, các bài kiểm tra và các thành phần khác được viết bằng mã nguồn Java. Thư mục **res** chứa tài nguyên như **layout, chuỗi văn bản, hình ảnh**.

 Chỉnh sửa tệp **AndroidManifest.xml** để thêm tính năng, thành phần và quyền truy cập vào ứng dụng Android. Mọi thành phần trong ứng dụng, chẳng hạn như nhiều **Activity**, đều phải được khai báo trong tệp XML này.

 Sử dụng **Android Virtual Device (AVD) Manager** để tạo **thiết bị ảo (emulator)** nhằm chạy ứng dụng của bạn.

 Thêm các câu lệnh **Log** vào ứng dụng để hiển thị thông báo trong **Logcat pane**, đây là một công cụ cơ bản để **debug**.

 Để chạy ứng dụng trên thiết bị Android thực bằng **Android Studio**, hãy bật **USB Debugging** trên thiết bị: Mở **Settings > About phone** và nhấn **Build number** **bảy lần**. Quay lại màn hình trước (**Settings**) và nhấn vào **Developer options**.và chọn **USB Debugging**.

### Giao diện người dùng tương tác đầu tiên

#### Phần A: Giao diện người dùng tương tác đầu tiên

Giới thiệu

Giao diện người dùng (**UI**) xuất hiện trên màn hình của một thiết bị Android bao gồm một **hệ thống phân cấp** các đối tượng được gọi là **View** — 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 UI và là **lớp cơ sở** cho các lớp cung cấp thành phần UI tương tác như **nút bấm (Button), hộp kiểm (Checkbox), ô nhập văn bản (Text Entry Field),**...Một số **lớp con phổ biến** của **View** được mô tả trong nhiều bài học bao gồm:

 **TextView**: Dùng để 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 hành vi tương tác.

 **ScrollView** và **RecyclerView**: Dùng để hiển thị nội dung có thể cuộn.

 **ImageView**: Dùng để hiển thị hình ảnh.

 **ConstraintLayout** và **LinearLayout**: Dùng để chứa các phần tử **View** khác và sắp xếp chúng trên giao diện.

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

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

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

Trong bài thực hành này, bạn sẽ học cách tạo ứng dụng tương tác đầu tiên của mình—một ứng dụng cho phép người dùng tương tác.Tạo một ứng dụng bằng mẫu **Empty Activity**.Sử dụng trình chỉnh sửa bố cục (**Layout Editor**) để thiết kế giao diện. Chỉnh sửa bố cục trong XML. Những kỹ năng này sẽ cần thiết để bạn hoàn thành các bài thực hành khác trong khóa học này.

Những thứ bạn đã biết

Bạn nên nắm vững các kiến thức sau:

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

Những thứ bạn sẽ học

 Cách tạo một ứng dụng có **tương tác với người dùng**.

 Cách sử dụng **layout editor** để thiết kế giao diện.

 Cách chỉnh sửa **layout bằng XML**.

 Rất nhiều thuật ngữ mới. Hãy xem **glossary (bảng thuật ngữ và khái niệm)** để có định nghĩa dễ hiểu hơn.

Những thứ bạn sẽ làm

 Tạo một ứng dụng và thêm **hai nút Button** cùng một **TextView** vào layout.

 Điều chỉnh từng phần tử trong **ConstraintLayout**, ràng buộc chúng với lề và các phần tử khác.

 Thay đổi **thuộc tính của các phần tử giao diện**.

 Chỉnh sửa **layout trong XML**.

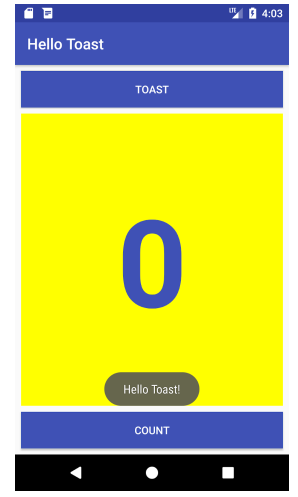
 Trích xuất **chuỗi văn bản cứng (hardcoded strings)** vào **tài nguyên chuỗi (string resources)**.

 Triển khai **phương thức xử lý sự kiện khi nhấn nút (click-handler methods)** để hiển thị thông báo trên màn hình khi người dùng nhấn vào từng nút.

Tổng quan ứng dụng

The HelloToast app consists of two Button elements and one TextView. When the user taps the first Button, it displays a short message (a Toast) on the screen. Tapping the second Button increases a "click" counter displayed in the TextView, which starts at zero.

Here's what the finished app looks like:



Nhiệm vụ 1: Tạp và khám phá dự án mới

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

* 1. Tạo 1 dự án Android Studio
  2. Khám phá trình chỉnh sửa Layout

1 Trong **Project > Android** pane, điều hướng đến thư mục **app > res > layout** và nhấp đúp vào tệp **activity\_main.xml** để mở nó (nếu chưa mở).

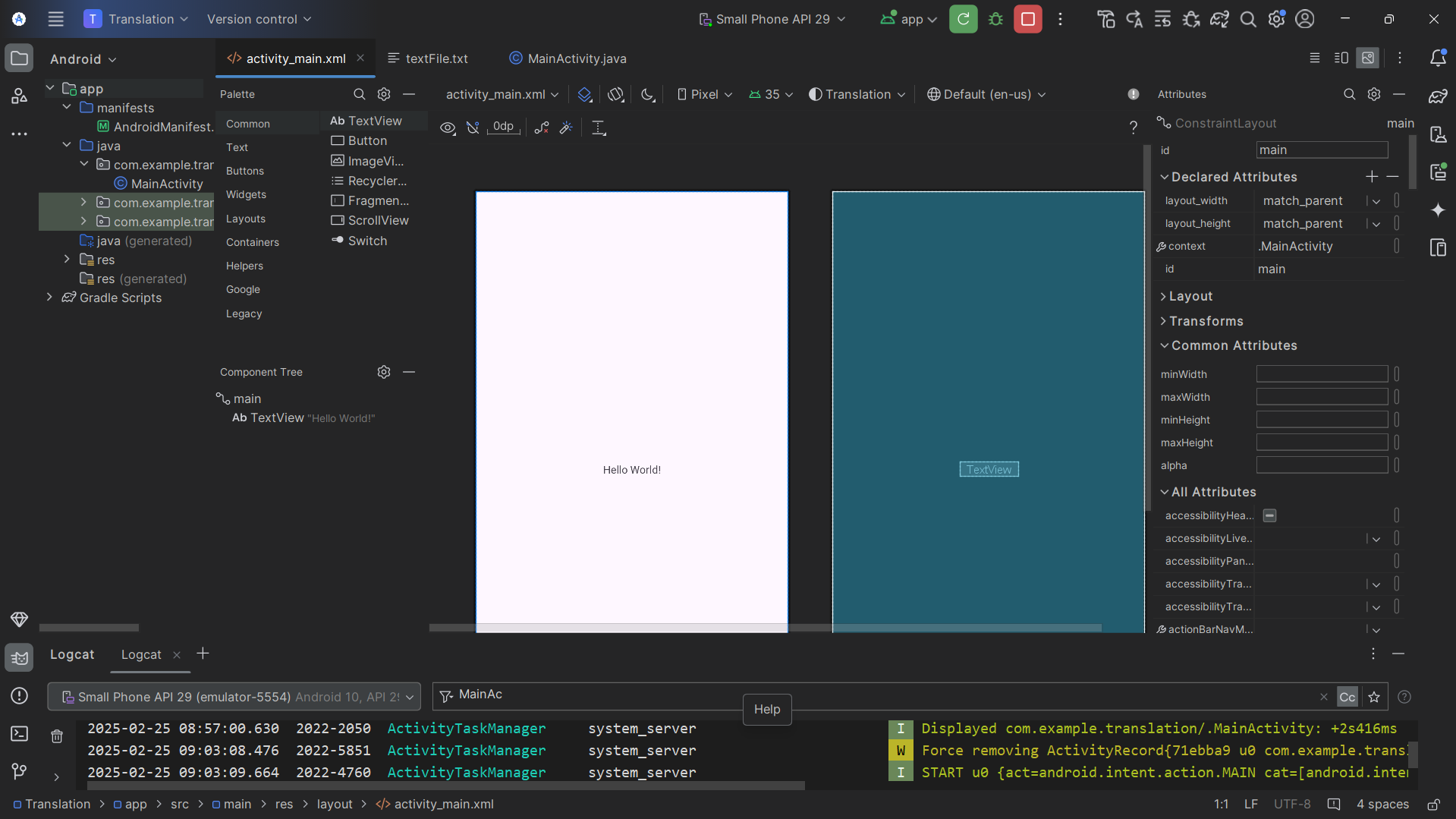
2 Nhấp vào tab **Design** nếu nó chưa được chọn. Bạn sử dụng tab này để thao tác các phần tử và bố cục, còn tab **Text** để chỉnh sửa mã XML của bố cục.

3 **Palettes pane** hiển thị các phần tử giao diện người dùng (UI elements) mà bạn có thể sử dụng trong bố cục của ứng dụng.

4 **Component tree pane** hiển thị cây phân cấp của các phần tử UI. Các phần tử này được tổ chức theo dạng cây với quan hệ cha - con, trong đó một 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 về các phần tử này trong bài học sau.

5 **Design và Blueprint panes** của trình chỉnh sửa bố cục (layout editor) hiển thị các phần tử giao diện. Trong hình minh họa, bố cục chỉ có một phần tử: **TextView** với nội dung "Hello World".

6 **Attributes tab** hiển thị bảng thuộc tính (**Attributes pane**) để thiết lập các thuộc tính cho một phần tử UI.

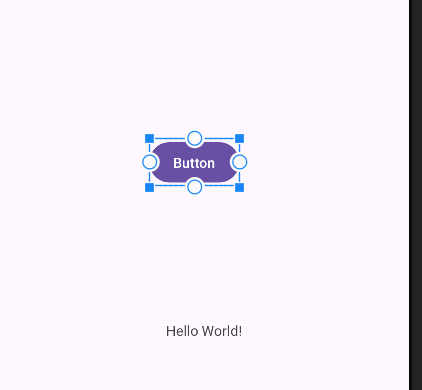


Task 2: Thêm các phần tử View trong trình chỉnh sửa bố cục (layout editor)

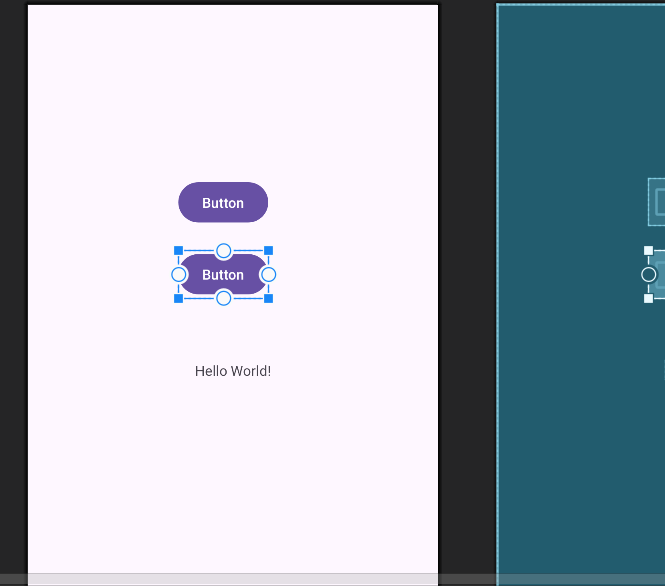
Trong nhiệm vụ này, bạn sẽ tạo bố cục giao diện người dùng (UI) cho ứng dụng HelloToast trong trình chỉnh sửa bố cục bằng cách sử dụng các tính năng của **ConstraintLayout**. Bạn có thể tạo các ràng buộc (constraints) một cách thủ công, như sẽ được hướng dẫn sau, hoặc tự động bằng công cụ **Autoconnect**.

2.1 Kiểm tra rằng buộc của phần tử

2.2 Thêm các nút bấm



2.3 Thêm nút thứ vào layout



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

Bảng **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 (UI). Các thuộc tính này thường được gọi là **properties (thuộc tính)**, và bạn có thể tìm thấy danh sách đầy đủ trong tài liệu của lớp **View**.

Trong nhiệm vụ này, bạn sẽ nhập các giá trị mới và thay đổi các thuộc tính quan trọng của **Button**, có thể áp dụng cho hầu hết các phần tử **View**.

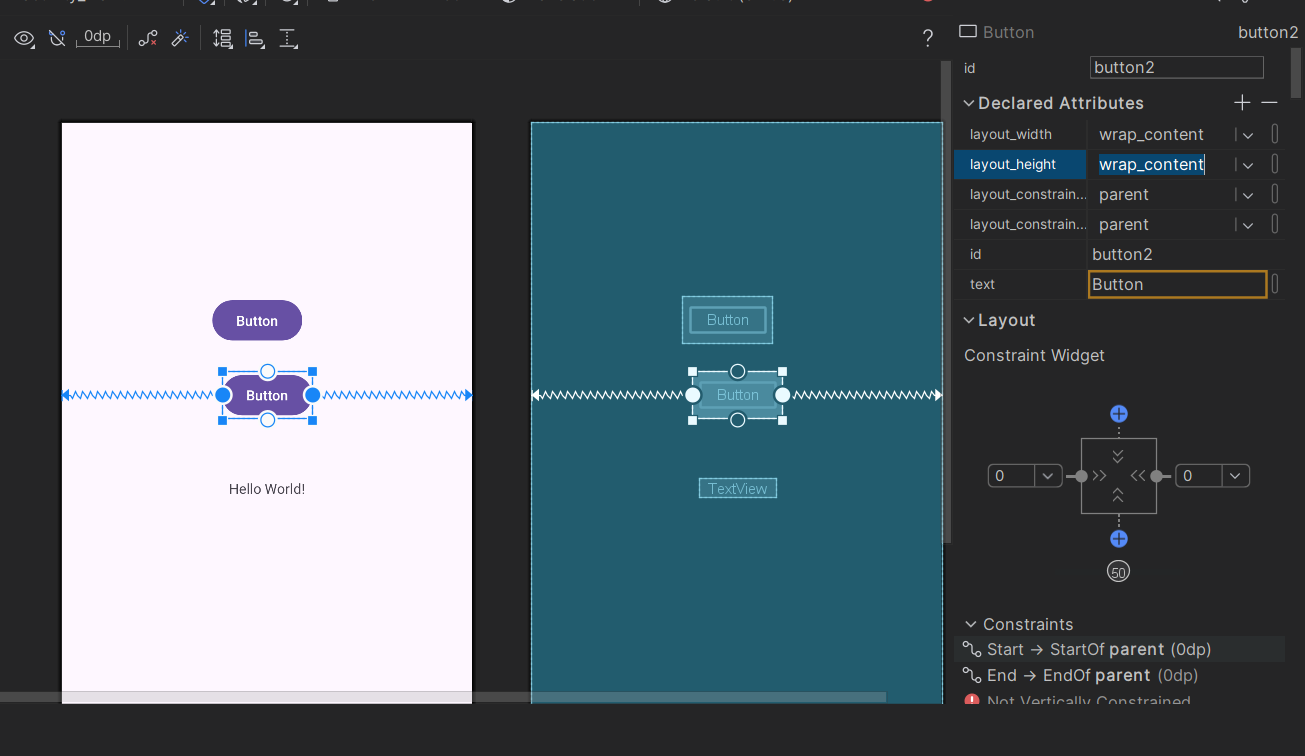
3.1 Thay đổi kích thước Button

Trình chỉnh sửa bố cục cung cấp các tay cầm thay đổi kích thước ở bốn góc của một View để bạn có thể thay đổi kích thước nhanh chóng. Bạn có thể kéo các tay cầm ở mỗi góc của View để thay đổi kích thước của nó, nhưng làm như vậy sẽ cố định các kích thước chiều rộng và chiều cao. Tránh cố định kích thước cho hầu hết các phần tử View, vì các kích thước cố định không thể thích ứng với nội dung và kích thước màn hình khác nhau.

Thay vào đó, hãy sử dụng bảng **Attributes** ở bên phải trình chỉnh sửa bố cục để chọn một chế độ kích thước không sử dụng kích thước cố định. Bảng **Attributes** bao gồm một bảng định cỡ hình vuông ở trên cùng, được gọi là **view inspector**.

Thuộc tính layout\_width và layout\_height trong bảng Thuộc tính thay đổi khi bạn điều chỉnh các điều khiển chiều cao và chiều rộng trong trình kiểm tra. Những thuộc tính này có thể nhận một trong ba giá trị đối với bố cục, là ConstraintLayout:

● Cài đặt match\_constraint mở rộng phần tử View để lấp đầy phần tử cha theo chiều rộng hoặc chiều cao—tới một lề nếu có đặt lề. Trong trường hợp này, phần tử cha là ConstraintLayout. Bạn sẽ tìm hiểu thêm về ConstraintLayout trong nhiệm vụ tiếp theo.  
● Cài đặt wrap\_content thu nhỏ kích thước của phần tử View sao cho vừa đủ để chứa nội dung của nó. Nếu không có nội dung, phần tử View sẽ trở nên vô hình.  
● Để chỉ định một kích thước cố định có thể điều chỉnh theo kích thước màn hình của thiết bị, hãy sử dụng một số cố định của pixel độc lập với mật độ (dp). Ví dụ, 16dp có nghĩa là 16 pixel độc lập với mật độ.



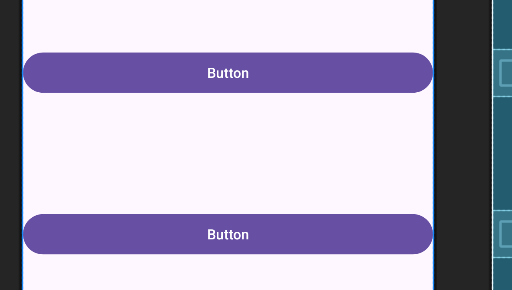
3.2 Thay đổi thuộc tính Button

Để xác định từng View một cách duy nhất trong bố cục của một Activity, mỗi View hoặc lớp con của View (chẳng hạn như Button) cần một ID duy nhất. Và để có thể sử dụng, các phần tử Button cần có văn bản. Các phần tử View cũng có thể có nền là màu hoặc hình ảnh.

Bảng Thuộc tính cung cấp quyền truy cập vào tất cả các thuộc tính mà bạn 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.

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

1. Sau khi chọn Button đầu tiên, chỉnh sửa trường ID ở đầu bảng Thuộc tính thành button\_toast cho thuộc tính android:id, được sử dụng để xác định phần tử trong bố cục.
2. Đặt thuộc tính background thành @color/colorPrimary. (Khi bạn nhập @c, các lựa chọn sẽ xuất hiện để dễ dàng chọn.)
3. Đặt thuộc tính textColor thành @android:color/white.
4. Chỉnh sửa thuộc tính text thành Toast.
5. Thực hiện các thay đổi thuộc tính tương tự cho Button thứ hai, sử dụng button\_count làm ID, Count cho thuộc tính text, và cùng màu nền và màu chữ như các bước trước.



Nhiệm vụ 4: Thêm TextEdit và sửa thuộc tính

Một trong những lợi ích của ConstraintLayout là khả năng căn chỉnh hoặc ràng buộc các phần tử so với 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 phần tử Button. Sau đó, bạn sẽ thay đổi các thuộc tính cho TextView trong bảng Attributes.

4.1 Thêm TextView và rằng buộc

4.2 Chỉnh thuộc tính TextView

 Đặt ID thành show\_count.

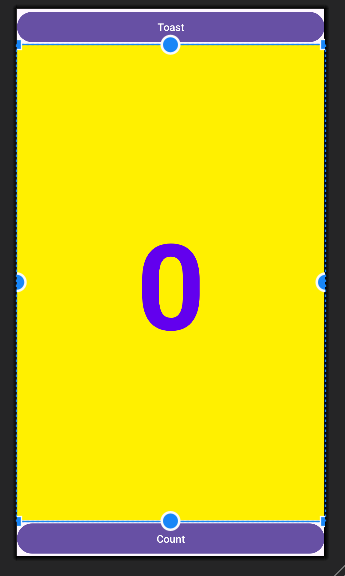
 Đặt văn bản thành 0.

 Đặt textSize thành 160sp.

 Đặt textStyle thành B (in đậm) và textAlignment thành ALIGNCENTER (căn giữa đoạn văn).

 Thay đổi điều khiển kích thước chiều ngang và chiều dọc (layout\_width và layout\_height) thành match\_constraint.

 Đặt textColor thành @color/colorPrimary.

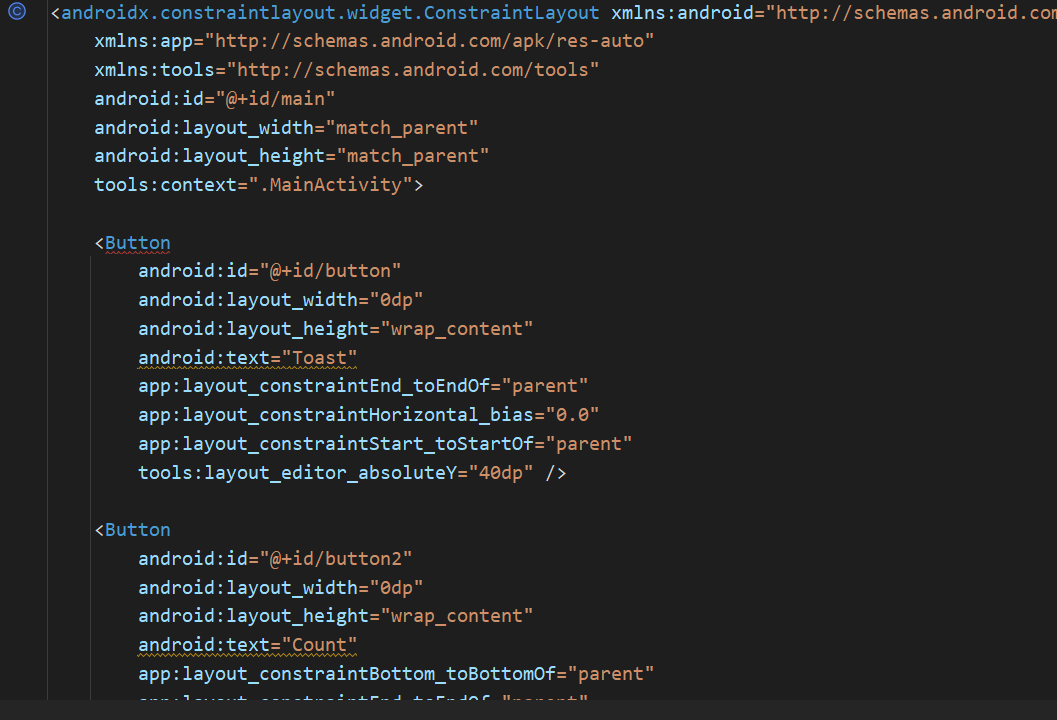


**Nhiệm vụ 5: Chỉnh sửa giao diện trong XML**

Bố cục của ứng dụng Hello Toast gần như đã hoàn thành! Tuy nhiên, một dấu chấm than xuất hiện bên cạnh mỗi phần tử UI trong Cây Thành Phần (Component Tree). Di chuột qua các dấu chấm than này để xem thông báo cảnh báo, như hình dưới đây. Cùng một cảnh báo xuất hiện cho cả ba phần tử: các chuỗi được mã hóa cứng (hardcoded strings) nên sử dụng tài nguyên (resources).

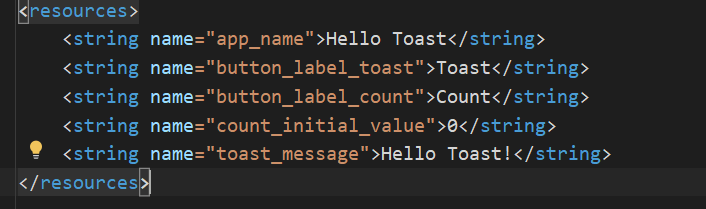
Cách dễ nhất để khắc phục các vấn đề về bố cục là chỉnh sửa bố cục trong XML. Mặc dù trình chỉnh sửa bố cục là một công cụ mạnh mẽ, nhưng một số thay đổi sẽ dễ thực hiện hơn trực tiếp trong mã nguồn XML.

5.1 Mở XML code cho giao diện



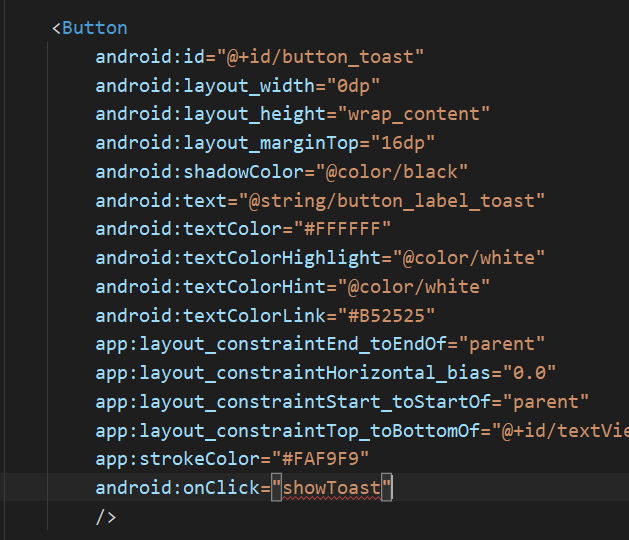
5.2 Trích xuất chuỗi

Thay vì mã hóa cứng chuỗi, một phương pháp tốt nhất là sử dụng tài nguyên chuỗi, đại diện cho các chuỗi. Việc đặt chuỗi trong một tệp riêng biệt giúp dễ dàng quản lý 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 một tệp tài nguyên chuỗi cho từng ngôn ngữ.



Task 6: Thêm xử lí onClick cho Button

6.1 Thêm thuộc tính onClick và xử lí vào mỗi Button



6.2 Chỉnh sửa xử lí Toast Button

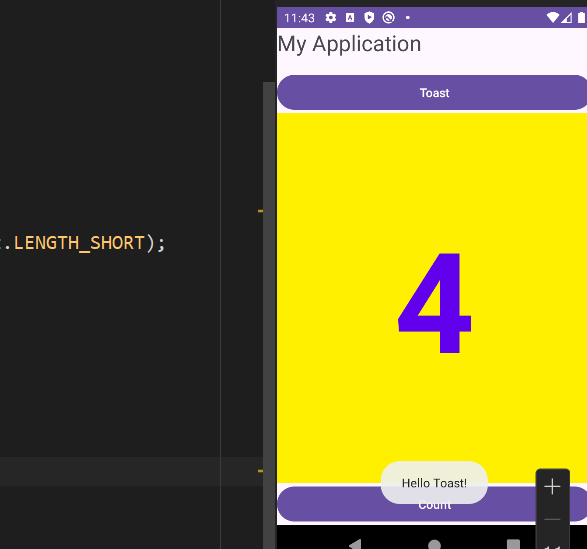
Bây giờ, bạn sẽ chỉnh sửa phương thức showToast() — trình xử lý nhấp của Button Toast trong MainActivity — sao cho nó hiển thị một thông báo. Một Toast cung cấp cách hiển thị một thông báo đơn giản trong một cửa sổ popup nhỏ. Nó chỉ chiếm không gian cần thiết cho thông báo. Activity hiện tại vẫn hiển thị và tương tác được. Một Toast có thể hữu ích để kiểm tra tính tương tác trong ứng dụng của bạn — hãy thêm một thông báo Toast để hiển thị kết quả khi người dùng nhấn vào Button hoặc thực hiện một hành động.



6.3 Chỉnh sửa xử lí nút Count

Bây giờ, bạn sẽ chỉnh sửa phương thức countUp() — trình xử lý nhấp của Button Count trong MainActivity — để nó hiển thị số đếm hiện tại sau khi nhấn vào Count. Mỗi lần nhấn sẽ tăng số đếm lên một đơn vị.

Mã cho trình xử lý phải:  
● Theo dõi số đếm khi nó thay đổi.  
● Gửi số đếm đã cập nhật đến TextView để hiển thị.



#### Trình chỉnh sửa bố cục

Như bạn đã học trong Bài 1.2 Phần A: Giao diện người dùng tương tác đầu tiên của bạn, bạn có thể xây dựng giao diện người dùng (UI) bằng cách sử dụng **ConstraintLayout** trong trình chỉnh sửa bố cục. **ConstraintLayout** sắp xếp các phần tử giao diện người dùng trong một bố cục bằng cách sử dụng các ràng buộc với các phần tử khác và với các cạnh của bố cục. **ConstraintLayout** được thiết kế để giúp bạn dễ dàng kéo các phần tử giao diện người dùng vào trình chỉnh sửa bố cục.

**ConstraintLayout** là một **ViewGroup**, một loại **View** đặc biệt có thể chứa các **View** khác (gọi là **children** hoặc **child views**). Trong phần thực hành này, bạn sẽ tìm hiểu thêm về các tính năng của **ConstraintLayout** và trình chỉnh sửa bố cục.

Phần thực hành này cũng giới thiệu hai lớp con khác của **ViewGroup**:

* **LinearLayout**: Một nhóm căn chỉnh các phần tử **View** con bên trong theo chiều ngang hoặc chiều dọc.
* **RelativeLayout**: Một nhóm các phần tử **View** con, trong đó mỗi phần tử **View** được định vị và căn chỉnh dựa trên các phần tử **View** khác trong **ViewGroup**. Vị trí của các phần tử **View** con được mô tả theo mối quan hệ với nhau hoặc với **ViewGroup** cha.

Những gì bạn nên biết trước  
Bạn nên có khả năng:

* Tạo ứng dụng **Hello World** bằng **Android Studio**.
* Chạy ứng dụng trên **trình giả lập** hoặc **thiết bị thực**.
* Tạo bố cục đơn giản cho ứng dụng bằng **ConstraintLayout**.
* Trích xuất và sử dụng **tài nguyên chuỗi (string resources)**.

Những gì bạn sẽ học

* Cách tạo **biến thể bố cục** cho **chế độ ngang (landscape)**.
* Cách tạo **biến thể bố cục** cho **máy tính bảng** và **màn hình lớn hơn**.
* Cách sử dụng **ràng buộc đường cơ sở (baseline constraint)** để căn chỉnh **các phần tử UI với văn bản**.
* Cách sử dụng **các nút pack và align** để căn chỉnh các phần tử trong bố cục.
* Cách định vị các **View** trong **LinearLayout**.
* Cách định vị các **View** trong **RelativeLayout**.

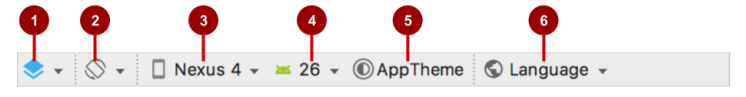
Những gì bạn sẽ làm

* Tạo **biến thể bố cục** cho **màn hình ngang**.
* Tạo **biến thể bố cục** cho **máy tính bảng** và **màn hình lớn**.
* **Chỉnh sửa bố cục** để thêm **các ràng buộc (constraints)** cho các phần tử UI.
* **Sử dụng ràng buộc đường cơ sở (baseline constraint)** để căn chỉnh phần tử với văn bản.
* **Sử dụng các nút pack và align** để căn chỉnh phần tử.
* **Chuyển đổi bố cục** sang **LinearLayout**.
* **Định vị các phần tử** trong **LinearLayout**.
* **Chuyển đổi bố cục** sang **RelativeLayout**.
* **Sắp xếp lại các View** trong bố cục chính sao cho liên kết với nhau.

**Nhiệm vụ 1: Tạo các biến thể bố cục**

Trong bài học trước, thử thách lập trình yêu cầu thay đổi bố cục của ứng dụng Hello Toast để phù hợp với chế độ hiển thị ngang hoặc dọc. Trong nhiệm vụ này, bạn sẽ học một cách dễ dàng hơn để tạo các biến thể của bố cục cho chế độ ngang (còn gọi là landscape) và chế độ dọc (còn gọi là portrait) trên điện thoại, cũng như cho các màn hình lớn hơn như máy tính bảng.

Trong nhiệm vụ này, bạn sẽ sử dụng một số nút trong hai thanh công cụ trên cùng của trình chỉnh sửa bố cục. Thanh công cụ trên cùng cho phép bạn cấu hình cách hiển thị bản xem trước bố cục trong trình chỉnh sửa bố cục.

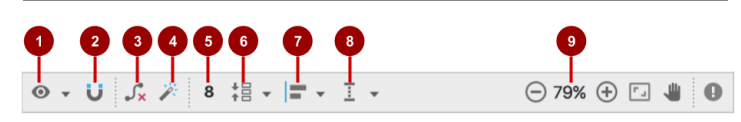




Trong hình trên:

1. **Chọn Bề mặt Thiết kế**: Chọn **Design** để hiển thị bản xem trước có màu của bố cục hoặc **Blueprint** để chỉ hiển thị đường viền của từng phần tử UI. Để xem cả hai chế độ cạnh nhau, chọn **Design + Blueprint**.
2. **Chế độ hiển thị trong Trình chỉnh sửa**: Chọn **Portrait** hoặc **Landscape** để xem trước bố cục theo chiều dọc hoặc ngang. Điều này hữu ích để xem trước bố cục mà không cần chạy ứng dụng trên trình giả lập hoặc thiết bị. Để tạo bố cục thay thế, chọn **Create Landscape Variation** hoặc các biến thể khác.
3. **Thiết bị trong Trình chỉnh sửa**: Chọn loại thiết bị (**điện thoại/máy tính bảng, Android TV hoặc Android Wear**).
4. **Phiên bản API trong Trình chỉnh sửa**: Chọn phiên bản Android để hiển thị bản xem trước.
5. **Chủ đề trong Trình chỉnh sửa**: Chọn một chủ đề (**ví dụ: AppTheme**) để áp dụng cho bản xem trước.
6. **Ngôn ngữ trong Trình chỉnh sửa**: Chọn ngôn ngữ và khu vực để xem trước. Danh sách này chỉ hiển thị các ngôn ngữ có trong tài nguyên chuỗi (**string resources**). Bạn cũng có thể chọn **Preview as Right To Left** để xem bố cục dưới dạng ngôn ngữ viết từ phải sang trái (**RTL**).

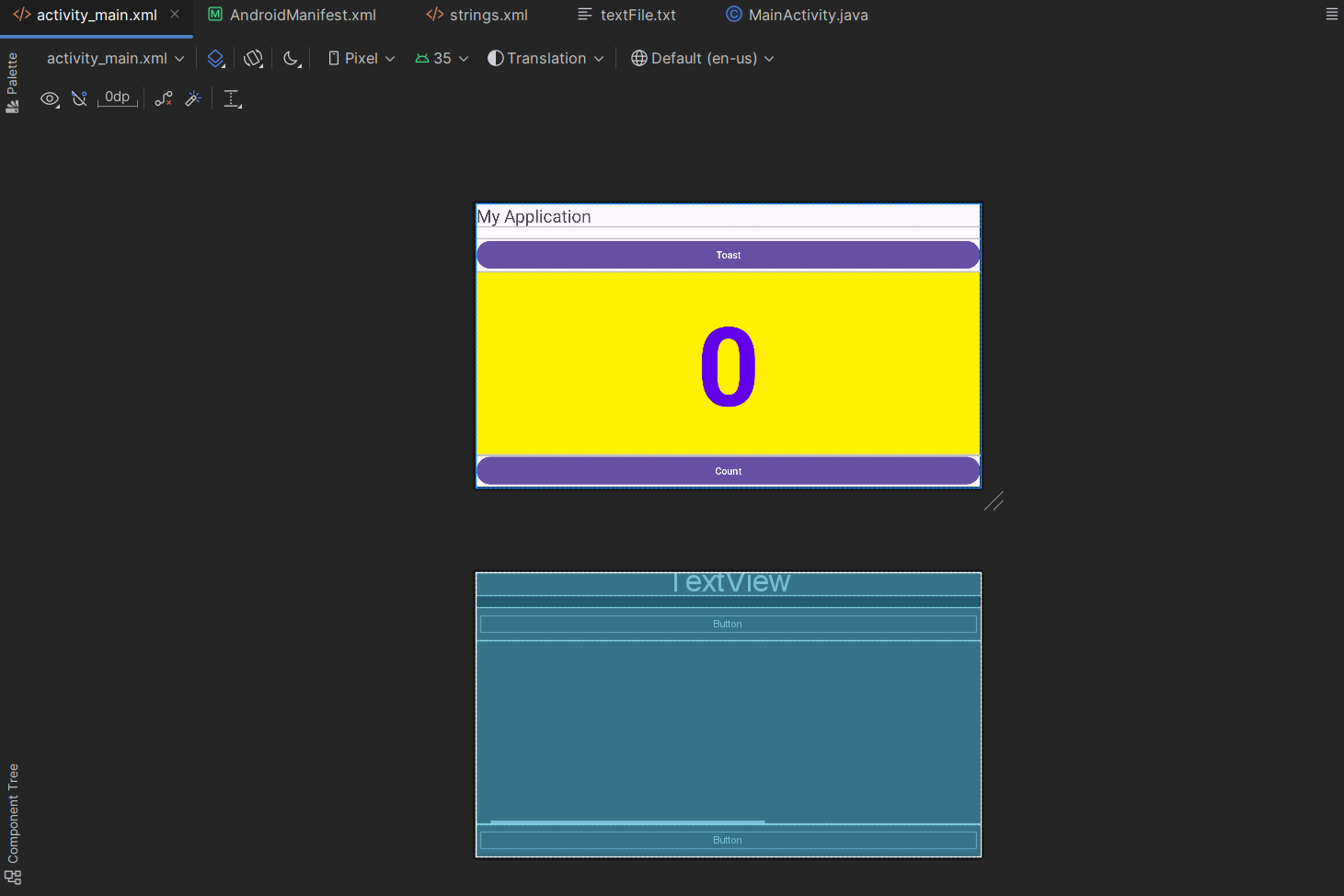




Trong hình trên:

1. **Hiển thị**: Chọn **Show Constraints** và **Show Margins** để hiển thị hoặc ẩn chúng trong bản xem trước.
2. **Tự động kết nối (Autoconnect)**: Bật hoặc tắt **Autoconnect**. Khi bật, bạn có thể kéo bất kỳ phần tử nào (**ví dụ: Button**) vào bố cục để tự động tạo ràng buộc với bố cục cha.
3. **Xóa tất cả ràng buộc**: Xóa toàn bộ ràng buộc trong bố cục.
4. **Suy luận ràng buộc (Infer Constraints)**: Tạo ràng buộc bằng cách suy luận.
5. **Lề mặc định (Default Margins)**: Đặt lề mặc định.
6. **Gói (Pack)**: Gom nhóm hoặc mở rộng các phần tử đã chọn.
7. **Căn chỉnh (Align)**: Căn chỉnh các phần tử đã chọn.
8. **Đường hướng dẫn (Guidelines)**: Thêm đường hướng dẫn dọc hoặc ngang.
9. **Điều khiển thu phóng/di chuyển**: Phóng to hoặc thu nhỏ

1.1 Xem trước bố cục trong chế độ ngang (horizontal orientation).



1.2 Tạo một biến thể bố cục cho chế độ ngang (horizontal orientation).

Sự khác biệt trực quan giữa chế độ dọc và chế độ ngang đối với bố cục này là chữ số (0) trong phần tử **show\_count** (TextView) nằm quá thấp trong chế độ ngang—quá gần với nút **Count**. Tùy thuộc vào thiết bị hoặc trình giả lập bạn sử dụng, phần tử **TextView** có thể trông quá lớn hoặc không căn giữa do kích thước văn bản được cố định ở **160sp**.

Sự khác biệt về mặt hình ảnh giữa chế độ dọc và ngang trong bố cục này là chữ số **0** trong phần tử **show\_count** (**TextView**) nằm quá thấp đối với chế độ ngang, quá gần với nút **Count** và kích thước cố định của **TextView** (160sp) có thể khiến bố cục trong chế độ ngang trông không cân đối, với chữ số **0** quá lớn hoặc không được căn giữa. Để giải quyết vấn đề này trong chế độ ngang mà vẫn giữ nguyên bố cục cho chế độ dọc, bạn có thể tạo một biến thể bố cục cho ứng dụng **Hello Toast**. Thực hiện theo các bước sau:

1. Nhấp vào nút **Orientation in Editor** trên thanh công cụ phía trên.
2. Chọn **Create Landscape Variation**.

Khi đó, một cửa sổ chỉnh sửa mới sẽ mở ra với tab **land/activity\_main.xml**, hiển thị bố cục dành riêng cho chế độ ngang. Bạn có thể thay đổi bố cục này mà không ảnh hưởng đến bố cục gốc cho chế độ dọc.

1. Trong bảng **Project > Android**, mở thư mục **res > layout**, bạn sẽ thấy Android Studio đã tự động tạo biến thể mới có tên **activity\_main.xml (land)**.

1.3 **Xem trước bố cục trên các thiết bị khác nhau**

Bạn có thể xem trước bố cục trên các thiết bị khác nhau mà không cần chạy ứng dụng trên thiết bị thực hoặc trình giả lập. Thực hiện các bước sau:

1. Tab **land/activity\_main.xml** vẫn đang mở trong trình chỉnh sửa bố cục. Nếu không, hãy nhấp đúp vào tệp **activity\_main.xml (land)** trong thư mục **layout**.
2. Nhấp vào nút **Device in Editor** trên thanh công cụ phía trên.
3. Trong menu thả xuống, chọn một thiết bị khác. Ví dụ, chọn **Nexus 4**, **Nexus 5**, sau đó **Pixel** để xem sự khác biệt trong các bản xem trước. Những khác biệt này xảy ra do kích thước văn bản cố định của **TextView**.

1.4 **Thay đổi bố cục cho chế độ ngang**

Bạn có thể sử dụng bảng **Attributes** trong tab **Design** để thiết lập hoặc thay đổi các thuộc tính. Tuy nhiên, đôi khi việc chỉnh sửa trực tiếp mã XML trong tab **Text** sẽ nhanh hơn. Tab **Text** hiển thị mã XML và cung cấp một tab **Preview** ở phía bên phải cửa sổ để hiển thị bản xem trước bố cục

**Các bước thay đổi bố cục:**

1. Đảm bảo tab **land/activity\_main.xml** vẫn mở trong trình chỉnh sửa bố cục. Nếu không, nhấp đúp vào tệp **activity\_main.xml (land)** trong thư mục **layout**.
2. Chuyển sang tab **Text** và bật tab **Preview** (nếu chưa được chọn).
3. Trong mã XML, tìm phần tử **TextView**.
4. Thay đổi thuộc tính android:textSize="100sp" thành android:textSize="120sp". Kết quả thay đổi sẽ hiển thị ngay lập tức trong bản xem trước
5. Sử dụng menu thả xuống **Device in Editor** để chọn các thiết bị khác nhau và xem cách bố cục hiển thị ở chế độ ngang trên các thiết bị đó.

A yellow and purple rectangle with a number 0

AI-generated content may be incorrect.

**1.7 Sử dụng ràng buộc đường cơ sở (baseline constraint)**

Bạn có thể căn chỉnh một phần tử giao diện người dùng (UI) chứa văn bản, như **TextView** hoặc **Button**, với một phần tử giao diện khác cũng chứa văn bản. Ràng buộc đường cơ sở (**baseline constraint**) cho phép bạn ràng buộc các phần tử sao cho các đường cơ sở của văn bản được căn thẳng hàng.

1. Ràng buộc nút **button\_toast** vào phía trên và bên trái của bố cục. Kéo nút **button\_count** đến một vị trí gần nút **button\_toast**. Sau đó, ràng buộc nút **button\_count** vào phía bên trái của nút **button\_toast**, như minh họa trong hình động.



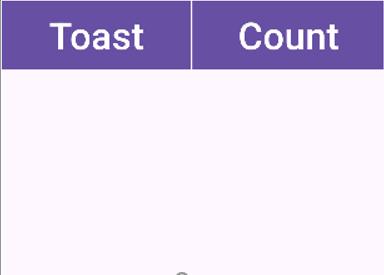
1. **Sử dụng ràng buộc đường cơ sở (baseline constraint)**, bạn có thể ràng buộc nút **button\_count** sao cho đường cơ sở văn bản của nó khớp với đường cơ sở văn bản của nút **button\_toast**. Chọn phần tử **button\_count**, sau đó di chuyển con trỏ chuột qua phần tử cho đến khi nút ràng buộc đường cơ sở xuất hiện bên dưới phần tử.
2. Nhấp vào nút ràng buộc đường cơ sở. Tay cầm ràng buộc đường cơ sở sẽ xuất hiện, nhấp nháy màu xanh lá. Nhấp và kéo một đường ràng buộc đường cơ sở đến đường cơ sở của phần tử **button\_toast** để hoàn tất.

**1.8 Mở rộng các nút theo chiều ngang**  
Nút **pack** trên thanh công cụ cung cấp các tùy chọn để sắp xếp hoặc mở rộng các phần tử giao diện người dùng đã chọn. Bạn có thể sử dụng nó để sắp xếp đều các nút **Button** theo chiều ngang trên giao diện.

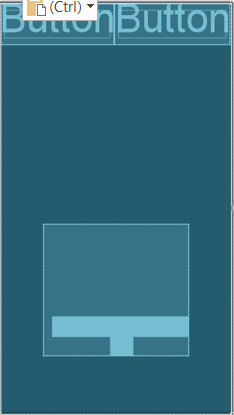
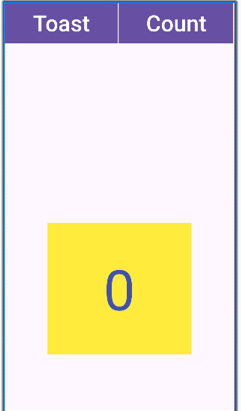
1. Chọn nút **button\_count** trong **Component Tree**, sau đó nhấn giữ **Shift** và chọn thêm nút **button\_toast** để cả hai nút đều được chọn.
2. Nhấp vào nút **pack** trên thanh công cụ, sau đó chọn tùy chọn **Expand Horizontally** như minh họa trong hình.



Các phần tử **Button** sẽ mở rộng theo chiều ngang để lấp đầy toàn bộ giao diện như minh họa bên dưới.



1. Để hoàn thiện bố cục, hãy ràng buộc **show\_count TextView** vào phía dưới của nút **button\_toast** và ràng buộc nó với các cạnh bên cũng như cạnh dưới của bố cục, như được minh họa trong hình động bên dưới.

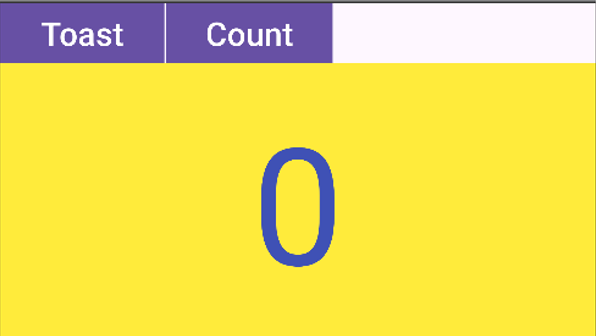


1. Các bước cuối cùng là thay đổi **layout\_width** và **layout\_height** của **show\_count TextView** thành **Match Constraints** và đặt **textSize** thành **200sp**. Bố cục cuối cùng sẽ trông như hình minh họa bên dưới.

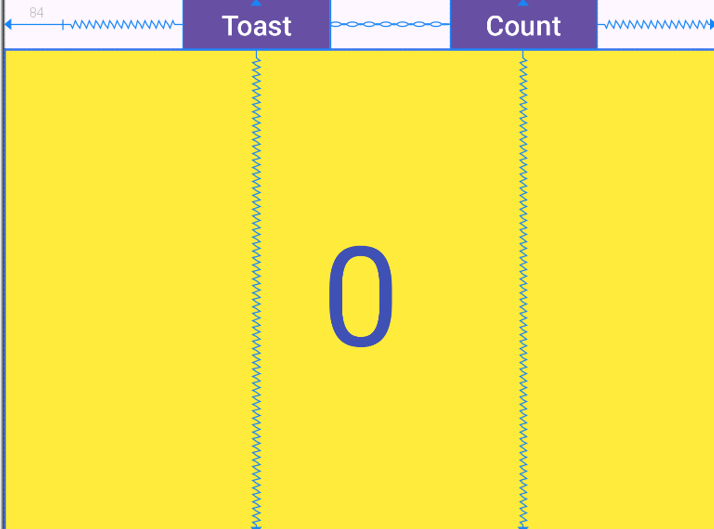
A yellow and blue number

AI-generated content may be incorrect.

1. Nhấn vào nút **Orientation in Editor** trên thanh công cụ ở phía trên và chọn **Switch to Landscape** (Chuyển sang chế độ ngang). Giao diện của bố cục trên máy tính bảng sẽ xuất hiện với hướng nằm ngang như hình dưới đây. (Bạn có thể chọn **Switch to Portrait** để quay lại chế độ dọc).



Chạy ứng dụng trên các trình giả lập khác nhau và thay đổi hướng màn hình sau khi chạy ứng dụng để xem giao diện trông như thế nào trên các loại thiết bị khác nhau. Bạn đã thành công trong việc tạo một ứng dụng có giao diện người dùng (UI) hoạt động đúng trên điện thoại và máy tính bảng với các kích thước và mật độ màn hình khác nhau.  
Mẹo: Để có hướng dẫn chi tiết về cách sử dụng ConstraintLayout



### Trình chỉnh sửa bố cục

### Văn bản và các chế độ cuộn

**Giới thiệu**  
Lớp **TextView** là một lớp con của lớp **View**, dùng để hiển thị văn bản trên màn hình. Bạn có thể kiểm soát cách văn bản xuất hiện bằng cách sử dụng các thuộc tính của **TextView** trong tệp bố cục XML. Bài thực hành này hướng dẫn cách làm việc với nhiều phần tử **TextView**, bao gồm cả một phần tử mà người dùng có thể cuộn nội dung của nó theo chiều dọc.

Nếu có nhiều thông tin hơn so với khả năng hiển thị của màn hình thiết bị, bạn có thể tạo một **chế độ xem cuộn** để người dùng có thể cuộn theo chiều dọc bằng cách vuốt lên hoặc xuống, hoặc cuộn ngang bằng cách vuốt sang phải hoặc trái.  
Thông thường, bạn sẽ sử dụng chế độ xem cuộn cho các bài báo, tin tức hoặc bất kỳ đoạn văn bản dài nào không thể hiển thị hoàn toàn trên màn hình. Bạn cũng có thể sử dụng chế độ xem cuộn để cho phép người dùng nhập nhiều dòng văn bản hoặc kết hợp các thành phần giao diện người dùng (chẳng hạn như trường nhập văn bản và nút) trong một chế độ xem cuộn.

Lớp **ScrollView** cung cấp bố cục cho chế độ xem cuộn. **ScrollView** là một lớp con của **FrameLayout**. Hãy chỉ đặt **một** phần tử con bên trong nó—phần tử con này chứa toàn bộ nội dung cần cuộn. Phần tử con này có thể là một **ViewGroup** (chẳng hạn như **LinearLayout**) chứa các thành phần giao diện người dùng.

Các bố cục phức tạp có thể gặp vấn đề về hiệu suất với các phần tử con như hình ảnh. Một lựa chọn tốt cho một **View** bên trong **ScrollView** là **LinearLayout** được sắp xếp theo chiều dọc, hiển thị các mục mà người dùng có thể cuộn qua (chẳng hạn như các phần tử **TextView**).

Với **ScrollView**, tất cả các thành phần giao diện người dùng đều được tải vào bộ nhớ và có trong cây cấu trúc xem ngay cả khi chúng không hiển thị trên màn hình. Điều này làm cho **ScrollView** lý tưởng để cuộn qua các trang văn bản tự do một cách mượt mà, vì văn bản đã được tải sẵn vào bộ nhớ. Tuy nhiên, **ScrollView** có thể tiêu tốn nhiều bộ nhớ, ảnh hưởng đến hiệu suất của phần còn lại của ứng dụng. Để hiển thị danh sách dài các mục mà người dùng có thể thêm, xóa hoặc chỉnh sửa, hãy cân nhắc sử dụng **RecyclerView**, được mô tả trong bài học khác.

**Những gì bạn cần biết trước**  
Bạn cần có khả năng:

* Tạo ứng dụng Hello World bằng Android Studio.
* Chạy ứng dụng trên trình giả lập hoặc thiết bị.
* Triển khai **TextView** trong bố cục của ứng dụng.
* Tạo và sử dụng tài nguyên chuỗi văn bản.

**Những gì bạn sẽ học**

* Cách sử dụng mã XML để thêm nhiều phần tử **TextView**.
* Cách sử dụng mã XML để định nghĩa một **View** cuộn.
* Cách hiển thị văn bản tự do với một số thẻ định dạng HTML.
* Cách tạo kiểu cho màu nền và màu văn bản của **TextView**.
* Cách thêm liên kết web vào văn bản.

**Những gì bạn sẽ làm**

* Tạo ứng dụng ScrollingText.
* Thay đổi **ConstraintLayout ViewGroup** thành **RelativeLayout**.
* Thêm hai phần tử **TextView** cho tiêu đề bài viết và tiêu đề phụ.
* Sử dụng kiểu dáng và màu sắc **TextAppearance** cho tiêu đề bài viết và tiêu đề phụ.

**Sử dụng thẻ HTML trong chuỗi văn bản để kiểm soát định dạng:**

* Sử dụng thuộc tính **lineSpacingExtra** để thêm khoảng cách giữa các dòng, cải thiện khả năng đọc.
* Thêm **ScrollView** vào bố cục để cho phép cuộn qua phần tử **TextView**.
* Sử dụng thuộc tính **autoLink** để các URL trong văn bản trở nên hoạt động và có thể nhấp vào.

**Tổng quan về ứng dụng**  
Ứng dụng **Scrolling Text** minh họa cách sử dụng thành phần giao diện người dùng **ScrollView**. **ScrollView** là một **ViewGroup**, trong ví dụ này, chứa một **TextView**. Nó hiển thị một trang văn bản dài—trong trường hợp này là một bài đánh giá album nhạc—mà người dùng có thể cuộn theo chiều dọc để đọc bằng cách vuốt lên hoặc xuống. Thanh cuộn xuất hiện ở lề bên phải. Ứng dụng này cho thấy cách bạn có thể sử dụng văn bản được định dạng với các thẻ HTML tối thiểu để thiết lập văn bản in đậm hoặc in nghiêng, và sử dụng ký tự xuống dòng để ngăn cách các đoạn văn. Bạn cũng có thể thêm các liên kết web hoạt động trong văn bản.

Trong hình trên, các yếu tố sau xuất hiện:

1. Một liên kết web hoạt động được nhúng trong văn bản tự do.
2. Thanh cuộn xuất hiện khi cuộn văn bản.

**Nhiệm vụ 1: Thêm và chỉnh sửa các phần tử TextView**

Trong bài thực hành này, bạn sẽ tạo một dự án Android cho ứng dụng **ScrollingText**, thêm các phần tử **TextView** vào bố cục để hiển thị tiêu đề và tiêu đề phụ của bài viết, đồng thời thay đổi phần tử **TextView** "Hello World" hiện tại để hiển thị một bài viết dài. Hình minh họa bên dưới là sơ đồ của bố cục.

Bạn sẽ thực hiện tất cả các thay đổi này trong mã XML và tệp strings.xml. Bạn sẽ chỉnh sửa mã XML cho bố cục trong tab **Text**, mà bạn có thể hiển thị bằng cách nhấp vào tab **Text**, thay vì nhấp vào tab **Design** để mở giao diện thiết kế. Một số thay đổi đối với các phần tử giao diện người dùng (UI) và thuộc tính sẽ dễ dàng thực hiện hơn trực tiếp trong tab **Text** bằng cách sử dụng mã nguồn XML.

**1.1 Tạo dự án và các phần tử TextView**

Trong nhiệm vụ này, bạn sẽ tạo dự án và thêm các phần tử **TextView**, đồng thời sử dụng các thuộc tính **TextView** để tạo kiểu cho văn bản và nền.

**Mẹo**: Để tìm hiểu thêm về các thuộc tính này, xem tài liệu tham khảo về **TextView**.

1. Trong Android Studio, tạo một dự án mới với các thông số sau:

| **Thuộc tính** | **Giá trị** |
| --- | --- |
| **Application Name** | Scrolling Text |
| **Company Name** | android.example.com (hoặc tên miền của bạn) |
| **Phone and Tablet Minimum SDK** | API15: Android 4.0.3 IceCreamSandwich |
| **Template** | Empty Activity |
| **Generate Layout File checkbox** | Selected |
| **Backwards Compatibility (AppCompat) checkbox** | Selected |

1. Trong thư mục **app > res > layout** ở ngăn **Project > Android**, mở tệp **activity\_main.xml** và nhấp vào tab **Text** để xem mã XML.

Ở phần trên cùng, hoặc **gốc**, của cấu trúc phân cấp **View** là **ViewGroup ConstraintLayout**:

<androidx.constraintlayout.widget.ConstraintLayout>

1. Thay đổi **ViewGroup** này thành **RelativeLayout**. Dòng mã thứ hai bây giờ sẽ trông giống như sau:

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

**RelativeLayout** cho phép bạn đặt các phần tử giao diện người dùng (UI) tương đối với nhau hoặc tương đối với chính **RelativeLayout** cha.  
Phần tử **TextView** "Hello World" mặc định được tạo bởi mẫu Bố cục Trống (Empty Layout) vẫn có các thuộc tính ràng buộc (chẳng hạn như app:layout\_constraintBottom\_toBottomOf="parent"). Đừng lo lắng—bạn sẽ xóa chúng trong bước tiếp theo.

1. Xóa dòng mã XML sau đây, dòng này liên quan đến ConstraintLayout:

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

Khối mã XML ở phần đầu bây giờ trông như thế này:

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">

1. Thêm một phần tử TextView phía trên TextView "Hello World" bằng cách nhập <TextView. Một khối TextView sẽ xuất hiện, kết thúc bằng />, và hiển thị các thuộc tính layout\_width và layout\_height, là những thuộc tính bắt buộc đối với TextView.
2. Nhập các thuộc tính sau cho TextView. Khi bạn nhập từng thuộc tính và giá trị, các gợi ý sẽ xuất hiện để hoàn thành tên thuộc tính hoặc giá trị.

A screenshot of a computer

AI-generated content may be incorrect.

1. Trích xuất tài nguyên chuỗi cho thuộc tính android:text với chuỗi cứng "Article Title" trong TextView để tạo một mục trong **strings.xml**.

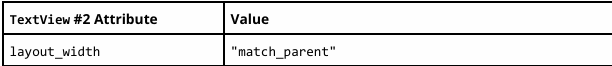
Đặt con trỏ vào chuỗi cứng, nhấn Alt-Enter (hoặc Option-Enter trên Mac), chọn **Extract string resource,** đảm bảo tùy chọn **Create the resource in directories** được chọn, sau đó chỉnh sửa tên tài nguyên thành **article\_title**.

Tài nguyên chuỗi được mô tả chi tiết trong mục **String Resources**.

1. Trích xuất tài nguyên kích thước cho thuộc tính android:padding với chuỗi cứng "10dp" trong TextView để tạo tệp dimens.xml và thêm một mục vào đó.

Đặt con trỏ vào chuỗi cứng, nhấn Alt-Enter (hoặc Option-Enter trên Mac), chọn Extract dimension resource, đảm bảo tùy chọn Create the resource in directories được chọn, sau đó chỉnh sửa tên tài nguyên thành padding\_regular.

9. Thêm một phần tử TextView khác phía trên TextView "Hello World" và bên dưới TextView bạn đã tạo trong các bước trước đó. Thêm các thuộc tính sau cho TextView.



A screenshot of a computer program

AI-generated content may be incorrect.

Vì bạn đã trích xuất tài nguyên kích thước cho chuỗi "10dp" thành padding\_regular trong TextView được tạo trước đó, bạn có thể sử dụng @dimen/padding\_regular cho thuộc tính android:padding trong TextView này.

1. Trích xuất tài nguyên chuỗi cho chuỗi cứng "Article Subtitle" của thuộc tính android:text trong TextView thành article\_subtitle.
2. Trong phần tử TextView "Hello World", xóa các thuộc tính layout\_constraint:

app:layout\_constraintBottom\_toBottomOf="parent"  
app:layout\_constraintLeft\_toLeftOf="parent"  
app:layout\_constraintRight\_toRightOf="parent"  
app:layout\_constraintTop\_toTopOf="parent"

1. Thêm các thuộc tính TextView sau vào phần tử “Hello World” TextView và thay đổi thuộc tính android:text:

A list of words and symbols

AI-generated content may be incorrect.

1. Trích xuất tài nguyên chuỗi cho "Article text" thành article\_text và trích xuất tài nguyên kích thước cho "5sp" thành line\_spacing.
2. Định dạng lại và căn chỉnh mã bằng cách chọn **Code > Reformat Code**. Đây là một thực hành tốt để định dạng và căn chỉnh mã của bạn sao cho dễ hiểu hơn đối với bạn và người khác.

**1.2 Thêm nội dung bài viết**  
Trong một ứng dụng thực tế truy cập các bài báo từ tạp chí hoặc báo, các bài viết hiển thị có thể sẽ đến từ một nguồn trực tuyến thông qua nhà cung cấp nội dung hoặc có thể được lưu sẵn trong cơ sở dữ liệu trên thiết bị.

Trong bài thực hành này, bạn sẽ tạo bài viết dưới dạng một chuỗi dài trong tệp tài nguyên strings.xml.

1. Trong thư mục **app > res > values**, mở **strings.xml**.
2. Mở bất kỳ tệp văn bản nào có một lượng lớn văn bản, hoặc mở tệp **strings.xml** của ứng dụng **ScrollingText** đã hoàn thiện.
3. Nhập giá trị cho các chuỗi article\_title và article\_subtitle với tiêu đề và phụ đề tùy ý, hoặc sử dụng các giá trị trong tệp strings.xml của ứng dụng **ScrollingText** đã hoàn thiện. Đảm bảo rằng các giá trị chuỗi là văn bản trên một dòng và không có thẻ HTML hoặc nhiều dòng.
4. Nhập hoặc sao chép-dán văn bản cho chuỗi article\_text.

Bạn có thể sử dụng văn bản trong tệp văn bản của bạn, hoặc sử dụng văn bản được cung cấp cho chuỗi article\_text trong tệp strings.xml của ứng dụng **ScrollingText** đã hoàn thiện. Yêu cầu duy nhất cho nhiệm vụ này là văn bản phải đủ dài để không hiển thị hết trên màn hình.

Lưu ý các điểm sau (tham khảo hình minh họa bên dưới để có ví dụ):

* Khi bạn nhập hoặc dán văn bản vào tệp strings.xml, các dòng văn bản sẽ không tự động xuống dòng mà sẽ kéo dài vượt ra ngoài lề phải. Đây là hành vi đúng—mỗi dòng văn bản mới bắt đầu từ lề trái sẽ đại diện cho toàn bộ một đoạn văn. Nếu bạn muốn văn bản trong strings.xml được xuống dòng, bạn có thể nhấn **Return** để thêm dấu kết thúc dòng cứng, hoặc định dạng văn bản trước trong một trình soạn thảo văn bản với các dấu kết thúc dòng cứng.
* Nhập \n để đại diện cho kết thúc một dòng và thêm một \n khác để đại diện cho một dòng trống. Bạn cần thêm ký tự kết thúc dòng để các đoạn văn không nối liền với nhau.
* Nếu bạn có dấu nháy đơn (') trong văn bản, bạn phải thoát nó bằng cách thêm một dấu gạch chéo ngược (\') phía trước. Nếu bạn có dấu nháy kép trong văn bản, bạn cũng phải thoát nó (\"). Bạn cũng phải thoát bất kỳ ký tự không thuộc ASCII nào khác. Xem phần **Định dạng và phong cách** của **Tài nguyên chuỗi** để biết thêm chi tiết.
* Thêm thẻ HTML <b> và </b> xung quanh các từ cần in đậm.
* Thêm thẻ HTML <i> và </i> xung quanh các từ cần in nghiêng. Nếu bạn sử dụng dấu nháy đơn cong trong một cụm từ in nghiêng, hãy thay chúng bằng dấu nháy đơn thẳng.
* Bạn có thể kết hợp in đậm và in nghiêng bằng cách kết hợp các thẻ, như trong <b><i>... từ ...</i></b>.
* Các thẻ HTML khác sẽ bị bỏ qua.
* Bao toàn bộ văn bản trong <string name="article\_text"> </string> trong tệp strings.xml.
* Bao gồm một liên kết web để kiểm tra, chẳng hạn như [www.google.com](http://www.google.com). (Ví dụ dưới đây sử dụng [www.rockument.com](http://www.rockument.com).) Đừng sử dụng thẻ HTML, vì bất kỳ thẻ HTML nào ngoài thẻ in đậm và in nghiêng sẽ bị bỏ qua và hiển thị dưới dạng văn bản, điều này không phải là điều bạn muốn.

**1.3 Chạy ứng dụng**

Chạy ứng dụng. Bài viết xuất hiện, nhưng người dùng không thể cuộn bài viết vì bạn chưa thêm một ScrollView (việc này bạn sẽ thực hiện trong nhiệm vụ tiếp theo). Lưu ý rằng khi nhấn vào một liên kết web, hiện tại sẽ không có tác dụng gì. Bạn cũng sẽ sửa điều đó trong nhiệm vụ tiếp theo.

A screenshot of a phone

AI-generated content may be incorrect.

**Mã giải pháp nhiệm vụ 1**  
Tệp bố cục activity\_main.xml trông như sau:

*<?*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>

**Nhiệm vụ 2: Thêm một ScrollView và một liên kết web hoạt động**  
Trong nhiệm vụ trước, bạn đã tạo ứng dụng ScrollingText với các phần tử TextView cho tiêu đề bài viết, phụ đề và nội dung bài viết dài. Bạn cũng đã thêm một liên kết web, nhưng liên kết đó chưa hoạt động. Bạn sẽ thêm mã để làm cho nó hoạt động.

Ngoài ra, TextView tự nó không thể cho phép người dùng cuộn văn bản bài viết để xem toàn bộ nội dung. Bạn sẽ thêm một ViewGroup mới gọi là ScrollView vào bố cục XML để làm cho TextView có thể cuộn được.

1. Di chuyển đoạn mã kết thúc </ScrollView> xuống sau TextView có ID article để các thuộc tính của article nằm hoàn toàn bên trong **ScrollView**.
2. Xóa thuộc tính sau từ TextView có ID article và thêm nó vào **ScrollView**:

android:layout\_below="@id/article\_subheading">

Bài viết nằm bên trong phần tử ScrollView.

1. Chọn **Code > Reformat Code** để định dạng lại mã XML, sao cho TextView của bài viết xuất hiện được thụt vào bên trong đoạn mã <ScrollView.
2. Nhấp vào tab **Preview** ở phía bên phải của trình chỉnh sửa bố cục để xem trước bố cục.

**2.3 Chạy ứng dụng**

Để kiểm tra cách văn bản cuộn:

1. Chạy ứng dụng trên thiết bị hoặc trình giả lập.

Vuốt lên và xuống để cuộn qua bài viết. Thanh cuộn xuất hiện ở lề phải khi bạn cuộn.  
Nhấn vào liên kết web để truy cập trang web. Thuộc tính android:autoLink tự động biến bất kỳ URL nào có thể nhận dạng được trong TextView (chẳng hạn như [www.rockument.com](http://www.rockument.com)) thành một liên kết web.

2. Xoay thiết bị hoặc trình giả lập khi ứng dụng đang chạy.

Quan sát cách chế độ cuộn mở rộng để sử dụng toàn bộ màn hình và vẫn cuộn đúng cách.

3. Chạy ứng dụng trên máy tính bảng hoặc trình giả lập máy tính bảng.

Quan sát cách chế độ cuộn mở rộng để sử dụng toàn bộ màn hình và vẫn cuộn đúng cách.

A screenshot of a phone

AI-generated content may be incorrect.

Trong hình trên, các yếu tố sau đây xuất hiện:

1. Một liên kết web hoạt động được nhúng trong văn bản tự do.
2. Thanh cuộn xuất hiện khi cuộn qua văn bản.

Mã giải pháp nhiệm vụ 2  
Mã XML cho bố cục với chế độ xem cuộn như sau:

*<?*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"  
 android:id="@+id/main">  
  
 <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" />  
 <ScrollView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/article\_subheading">  
 <TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/article"  
 android:autoLink="web"  
 android:lineSpacingExtra="@dimen/line\_spacing"  
 android:padding="@dimen/padding\_regular"  
 android:text="@string/article\_text" />  
 </ScrollView>  
  
  
</RelativeLayout>

**Nhiệm vụ 3: Cuộn nhiều phần tử**

Như đã đề cập trước đó, một ScrollView chỉ có thể chứa một View con (ví dụ như TextView bạn đã tạo cho bài viết). Tuy nhiên, View đó có thể là một ViewGroup khác chứa các phần tử View, chẳng hạn như LinearLayout.

Bạn có thể *lồng* một ViewGroup, chẳng hạn như LinearLayout, vào trong ScrollView, từ đó cho phép cuộn toàn bộ nội dung bên trong LinearLayout.

Ví dụ, nếu bạn muốn phần tiêu đề phụ của bài viết cùng cuộn với nội dung bài viết, bạn cần thêm một LinearLayout vào bên trong ScrollView, sau đó di chuyển tiêu đề phụ và bài viết vào trong LinearLayout.

LinearLayout sẽ trở thành phần tử con duy nhất của ScrollView, như minh họa trong hình dưới đây. Người dùng sẽ có thể cuộn toàn bộ LinearLayout, bao gồm cả tiêu đề phụ và bài viết.

**3.1 Thêm một LinearLayout vào ScrollView**

1. Mở tệp activity\_main.xml trong dự án ứng dụng ScrollingText và chọn tab **Text** để chỉnh sửa mã XML (nếu tab này chưa được chọn).
2. Thêm một LinearLayout phía trên TextView bài viết (article) trong ScrollView. Khi bạn nhập <LinearLayout, Android Studio sẽ tự động thêm </LinearLayout> ở cuối, đồng thời gợi ý các thuộc tính android:layout\_width và android:layout\_height.Chọn match\_parent cho chiều rộng và wrap\_content cho chiều cao từ các gợi ý.Mã ở phần đầu của ScrollView bây giờ sẽ trông như sau:

Bạn sử dụng match\_parent để chiều rộng khớp với ViewGroup cha. Bạn sử dụng wrap\_content để thay đổi kích thước LinearLayout sao cho vừa đủ để bao bọc nội dung bên trong.

1. Di chuyển mã kết thúc </LinearLayout> xuống sau TextView bài viết (article) nhưng trước thẻ đóng </ScrollView>.LinearLayout bây giờ sẽ bao gồm TextView bài viết (article) và hoàn toàn nằm trong ScrollView.
2. Thêm thuộc tính android:orientation="vertical" vào LinearLayout để thiết lập hướng của nó thành dọc.
3. Chọn **Code > Reformat Code** để thụt lề mã chính xác.

LinearLayout bây giờ trông như thế này:

**3.2 Di chuyển các thành phần giao diện vào trong LinearLayout**

Hiện tại, LinearLayout chỉ chứa một thành phần giao diện là TextView bài viết (article). Bạn muốn đưa TextView tiêu đề phụ (article\_subheading) vào LinearLayout để cả hai có thể cuộn.

1. Để di chuyển TextView tiêu đề phụ (article\_subheading), hãy chọn đoạn mã của nó, chọn **Edit > Cut**, nhấp vào phía trên TextView bài viết (article) bên trong LinearLayout, và chọn **Edit > Paste**.
2. Xóa thuộc tính android:layout\_below="@id/article\_heading" khỏi TextView tiêu đề phụ (article\_subheading). Vì TextView này hiện đang nằm trong LinearLayout, thuộc tính này sẽ xung đột với các thuộc tính của LinearLayout.
3. Thay đổi thuộc tính bố cục của ScrollView từ  
   android:layout\_below="@id/article\_subheading" thành  
   android:layout\_below="@id/article\_heading".  
   Bây giờ, vì tiêu đề phụ là một phần của LinearLayout, ScrollView phải được đặt bên dưới tiêu đề, không phải tiêu đề phụ.

Mã XML cho ScrollView bây giờ sẽ như sau:

<ScrollView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/article\_heading">  
  
 <LinearLayout  
 android:orientation="vertical"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
 <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>

1. Chạy ứng dụng.

Vuốt lên và xuống để cuộn bài viết, và hãy chú ý rằng phần tiêu đề phụ giờ đây di chuyển cùng bài viết, trong khi phần tiêu đề chính vẫn cố định tại chỗ.

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.

Thử thách: Thêm một thành phần UI khác—một Nút —vào LinearLayout bên trong ScrollView để nó cuộn cùng với văn bản.tử giao diện người dùng khác—một **nút bấm (Button)**—vào **LinearLayout** bên trong **ScrollView** sao cho nút bấm này cuộn cùng với văn bản.

● Làm cho **nút bấm (Button)** xuất hiện ở phía dưới bài viết. Người dùng sẽ cuộn đến cuối bài viết để thấy **nút bấm**.  
● Sử dụng dòng chữ **Thêm bình luận** làm nội dung cho **nút bấm**. Trong thử thách này, không cần tạo phương thức xử lý nút bấm; bạn chỉ cần đặt phần tử **Button** vào đúng vị trí trong bố cục.

1. Thay đổi phần tiêu đề phụ (**subheading**) sao cho nó được gói gọn trong một cột bên trái có chiều rộng 100 dp, như minh họa bên dưới.
2. Đặt nội dung bài viết ở bên phải của phần tiêu đề phụ, như minh họa bên dưới.

### Tài nguyên có sẵn

## Activities

### Activity và Intent

**Giới thiệu**  
Một **Activity** đại diện cho một màn hình đơn lẻ trong ứng dụng của bạn, nơi người dùng có thể thực hiện một tác vụ cụ thể, chẳng hạn như chụp ảnh, gửi email hoặc xem bản đồ. Một hoạt động (activity) thường được hiển thị cho người dùng dưới dạng một cửa sổ toàn màn hình.

Một ứng dụng thường bao gồm nhiều màn hình được kết nối lỏng lẻo với nhau. Mỗi màn hình là một activity. Thông thường, một activity trong ứng dụng được chỉ định là **"main activity"** (**MainActivity.java**), và đây là activity được hiển thị khi ứng dụng được khởi chạy. Activity chính có thể khởi động các activity khác để thực hiện các hành động khác nhau.

Mỗi khi một activity mới được khởi động, activity trước đó sẽ dừng lại, nhưng hệ thống sẽ lưu activity đó trong một ngăn xếp (ngăn xếp "back stack"). Khi một activity mới được khởi động, activity mới này sẽ được đẩy vào ngăn xếp và nhận quyền điều khiển từ người dùng. Ngăn xếp "back stack" tuân theo nguyên tắc cơ bản "vào sau, ra trước" (**last in, first out**). Khi người dùng hoàn thành với activity hiện tại và nhấn nút **Back**, activity đó sẽ được xóa khỏi ngăn xếp và bị hủy, và activity trước đó sẽ được tiếp tục.

Một activity được khởi động hoặc kích hoạt bằng cách sử dụng một **intent**. Một **Intent** là một thông điệp không đồng bộ mà bạn có thể sử dụng trong activity của mình để yêu cầu một hành động từ một activity khác hoặc từ một thành phần khác trong ứng dụng. Bạn sử dụng intent để khởi động một activity từ một activity khác và truyền dữ liệu giữa các activity.

**Intent** có thể là **explicit** hoặc **implicit**:

* Một **explicit intent** (intent rõ ràng) là intent trong đó bạn biết rõ mục tiêu của intent đó. Nghĩa là, bạn đã biết tên lớp đầy đủ (fully qualified class name) của activity cụ thể.
* Một **implicit intent** (intent không rõ ràng) là intent trong đó bạn không có tên của thành phần mục tiêu, nhưng bạn có một hành động chung để thực hiện.

Trong bài thực hành này, bạn sẽ tạo các **explicit intents**. Bạn sẽ tìm hiểu cách sử dụng **implicit intents** trong bài thực hành sau.

**Những gì bạn nên biết trước:**  
Bạn cần có khả năng:

* Tạo và chạy các ứng dụng trong Android Studio.
* Sử dụng trình chỉnh sửa bố cục (**layout editor**) để tạo bố cục trong một **ConstraintLayout**.
* Chỉnh sửa mã XML của bố cục.
* Thêm chức năng **onClick** cho một **Button**.

**Những gì bạn sẽ học được:**

* Cách tạo một **Activity** mới trong Android Studio.
* Cách định nghĩa **parent** (cha) và **child activities** (hoạt động con) để sử dụng điều hướng **Up navigation**.
* Cách khởi động một **Activity** bằng một **explicit Intent**.
* Cách truyền dữ liệu giữa các **Activity** bằng một **explicit Intent**.

**Những gì bạn sẽ làm:**

* Tạo một ứng dụng Android mới với một **main Activity** (activity chính) và một **second Activity** (activity thứ hai).
* Truyền một số dữ liệu (chuỗi) từ **main Activity** sang **second Activity** bằng một **Intent**, và hiển thị dữ liệu đó trong **second Activity**.
* Gửi một dữ liệu khác (dạng chuỗi) ngược lại từ **second Activity** về **main Activity**, cũng bằng một **Intent**.

**Tổng quan ứng dụng:**  
Trong chương này, bạn sẽ tạo và xây dựng một ứng dụng gọi là **Two Activities**, đúng như tên gọi, ứng dụng này chứa hai **Activity**. Bạn sẽ xây dựng ứng dụng qua ba giai đoạn:

* Ở giai đoạn đầu tiên, bạn tạo một ứng dụng với **main activity** chứa một nút **Send**. Khi người dùng nhấn nút này, **main activity** sẽ sử dụng một **intent** để khởi động **second activity**.
* Ở giai đoạn thứ hai, bạn thêm một thành phần **EditText** vào **main activity**. Người dùng nhập một thông điệp và nhấn nút **Send**. **Main activity** sử dụng một **intent** để khởi động **second activity** và gửi thông điệp của người dùng đến **second activity**. **Second activity** hiển thị thông điệp mà nó nhận được.
* Ở giai đoạn cuối cùng của việc tạo ứng dụng **Two Activities**, bạn thêm một thành phần **EditText** và một nút **Reply** vào **second activity**. Bây giờ, người dùng có thể nhập một thông điệp phản hồi và nhấn nút **Reply**, và thông điệp phản hồi sẽ được hiển thị trên **main activity**. Trong giai đoạn này, bạn sử dụng một **intent** để chuyển thông điệp phản hồi từ **second activity** trở lại **main activity**.

**Nhiệm vụ 1: Tạo dự án TwoActivities**

Trong nhiệm vụ này, bạn thiết lập dự án ban đầu với một **main Activity**, định nghĩa bố cục, và tạo một phương thức cơ bản cho sự kiện nhấn nút **onClick**.

### ****1.1** Tạo dự án TwoActivities**

1. Mở **Android Studio** và tạo một **dự án Android Studio mới**.
   * Đặt tên ứng dụng là **Two Activities** và chọn cài đặt **Phone and Tablet** giống như các bài thực hành trước.
   * Thư mục dự án sẽ tự động được đặt tên là **TwoActivities**, và tên ứng dụng hiển thị trên **App Bar** sẽ là **"Two Activities".**
2. Chọn **Empty Activity** làm mẫu **Activity**. Nhấn **Next**.
3. Chấp nhận tên **Activity** mặc định là **MainActivity**.
   * Đảm bảo rằng **Generate Layout file** và **Backwards Compatibility (AppCompat)** đã được chọn.
4. Nhấn **Finish** để hoàn tất tạo dự án.

### ****1.2** Định nghĩa bố cục cho Main Activity**

1. Mở **res > layout > activity\_main.xml** trong **Project > Android**. Trình chỉnh sửa bố cục **(Layout Editor)** sẽ xuất hiện.
2. Nhấn vào tab **Design** (nếu chưa được chọn) và xóa **TextView** mặc định (có nội dung **"Hello World"**) trong **Component Tree.**
3. Khi chế độ **Autoconnect** đang bật (mặc định), kéo một **Button** từ **Palette** vào góc dưới bên phải của bố cục. **Autoconnect** sẽ tự động tạo các ràng buộc **(constraints)** cho **Button**.
4. Trong **Attributes**, đặt các thuộc tính sau:
   * **ID**: button\_main
   * **layout\_width** và **layout\_height**: wrap\_content
   * **Text**: Send

Lúc này, bố cục sẽ trông như sau:

A white background with pink and black lines

AI-generated content may be incorrect.

5. Nhấp vào tab **Text** để chỉnh sửa mã XML. Thêm thuộc tính sau vào Button:  
android:onClick="launchSecondActivity"  
Giá trị của thuộc tính được gạch dưới màu đỏ vì phương thức launchSecondActivity() chưa được tạo. Bỏ qua lỗi này tạm thời; bạn sẽ sửa trong nhiệm vụ tiếp theo.

**6. Trích xuất tài nguyên chuỗi**, như đã mô tả trong bài thực hành trước, cho văn bản "Send" và sử dụng tên tài nguyên là button\_main.

Mã XML cho nút Button sẽ trông như sau:

1.3 Xác định hành động của Button

Trong tác vụ này, bạn sẽ triển khai phương thức launchSecondActivity() mà bạn đã tham chiếu trong bố cục thông qua thuộc tính android:onClick.

1. Nhấp vào "launchSecondActivity" trong mã XML của activity\_main.xml.
2. Nhấn Alt+Enter (hoặc Option+Enter trên Mac) và chọn **Create 'launchSecondActivity(View)' in 'MainActivity'**.
   * File MainActivity sẽ mở ra, và Android Studio sẽ tự động tạo một phương thức khung cho trình xử lý launchSecondActivity().
3. Bên trong launchSecondActivity(), thêm một câu lệnh Log với nội dung "Button Clicked!".

LOG\_TAG sẽ hiển thị màu đỏ. Bạn sẽ thêm định nghĩa cho biến đó trong một bước tiếp theo

1. Ở đầu lớp MainActivity, thêm hằng số cho biến LOG\_TAG:

Hằng số này sử dụng tên của chính lớp làm thẻ.

1. Chạy ứng dụng của bạn. Khi bạn nhấn nút **Send**, bạn sẽ thấy thông báo "Button Clicked!" trong bảng **Logcat**. Nếu có quá nhiều đầu ra trong màn hình, hãy nhập **MainActivity** vào hộp tìm kiếm, và bảng **Logcat** sẽ chỉ hiển thị các dòng khớp với thẻ đó.

Mã cho MainActivity sẽ trông như sau:

**Nhiệm vụ 2: Tạo và khởi chạy Activity thứ hai**  
Mỗi activity mới mà bạn thêm vào dự án sẽ có tệp layout và tệp Java riêng, tách biệt với những tệp của activity chính. Các activity này cũng có các phần tử <activity> riêng trong tệp **AndroidManifest.xml**.

Giống như activity chính, các activity mới mà bạn tạo trong Android Studio cũng mở rộng từ lớp **AppCompatActivity**.

Mỗi activity trong ứng dụng của bạn chỉ được kết nối lỏng lẻo với các activity khác. Tuy nhiên, bạn có thể định nghĩa một activity là **parent** (cha) của một activity khác trong tệp **AndroidManifest.xml**. Mối quan hệ cha-con này cho phép Android thêm các gợi ý điều hướng, chẳng hạn như mũi tên quay lại hướng trái trong thanh tiêu đề của mỗi activity.

Một activity giao tiếp với các activity khác (trong cùng một ứng dụng hoặc giữa các ứng dụng khác nhau) bằng một **intent**. Một **Intent** có thể là:

* **Intent rõ ràng (explicit intent):** Là loại mà bạn biết rõ mục tiêu của intent đó; nghĩa là bạn đã biết tên lớp đầy đủ của activity cụ thể đó.
* **Intent không rõ ràng (implicit intent):** Là loại mà bạn không có tên của thành phần mục tiêu, nhưng có một hành động tổng quát để thực hiện.

Trong nhiệm vụ này, bạn thêm một activity thứ hai vào ứng dụng, với layout riêng của nó. Bạn chỉnh sửa tệp **AndroidManifest.xml** để định nghĩa activity chính là **parent** của activity thứ hai. Sau đó, bạn chỉnh sửa phương thức **launchSecondActivity()** trong **MainActivity** để bao gồm một intent giúp khởi chạy activity thứ hai khi bạn nhấn nút.

**2.1 Tạo Activity thứ hai**

1. Nhấp vào thư mục **app** của dự án và chọn **File > New > Activity > Empty Activity**.
2. Đặt tên cho Activity mới là **SecondActivity**. Đảm bảo rằng các tùy chọn **Generate Layout File** và **Backwards Compatibility (AppCompat)** được chọn. Tên layout sẽ được tự động điền là **activity\_second**.  
   **Không chọn** tùy chọn **Launcher Activity**.
3. Nhấn **Finish**. Android Studio sẽ thêm cả một tệp layout mới (**activity\_second.xml**) và một tệp Java mới (**SecondActivity.java**) vào dự án của bạn cho Activity mới. Đồng thời, nó cũng cập nhật tệp **AndroidManifest.xml** để bao gồm Activity mới.

**2.2 Sửa đổi tệp AndroidManifest.xml**

1. Mở **manifests > AndroidManifest.xml**.
2. Tìm phần tử <activity> mà Android Studio đã tạo cho **SecondActivity**:

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

1. Thay thế toàn bộ phần tử <activity> bằng nội dung sau:

**Thuộc tính label** thêm tiêu đề của **Activity** vào thanh ứng dụng (app bar).  
Với thuộc tính parentActivityName, bạn chỉ định rằng **MainActivity** là cha (parent) của **SecondActivity**. Mối quan hệ này được sử dụng cho điều hướng **Up navigation** trong ứng dụng của bạn: thanh ứng dụng của **SecondActivity** sẽ có mũi tên quay trái để người dùng có thể điều hướng "lên trên" (quay lại) **MainActivity**.

Với phần tử <meta‑data>, bạn cung cấp thông tin tùy ý bổ sung về Activity dưới dạng các cặp key-value. Trong trường hợp này, các thuộc tính metadata thực hiện cùng một chức năng như thuộc tính android:parentActivityName—chúng định nghĩa một mối quan hệ giữa hai Activity cho điều hướng "lên trên". Các thuộc tính metadata này là bắt buộc đối với các phiên bản Android cũ hơn, vì thuộc tính android:parentActivityName chỉ khả dụng cho API từ cấp 16 trở lên.

1. Trích xuất tài nguyên chuỗi (string resource) cho "Second Activity" trong đoạn mã trên, và sử dụng tên tài nguyên là activity2\_name.

**2.3 Định nghĩa bố cục cho Activity thứ hai**

1. Mở tệp **activity\_second.xml** và nhấp vào tab **Design** nếu nó chưa được chọn.
2. Kéo một **TextView** từ bảng **Palette** vào góc trên bên trái của bố cục và thêm các ràng buộc (**constraints**) vào các cạnh trên và trái của bố cục.
   * Thiết lập các thuộc tính của nó trong bảng **Attributes** như sau:

A screenshot of a computer

AI-generated content may be incorrect.

Giá trị của **textAppearance** là một thuộc tính đặc biệt của Android theme, định nghĩa các kiểu phông chữ cơ bản. Bạn sẽ tìm hiểu thêm về các theme trong một bài học sau.

Bố cục bây giờ sẽ trông như thế này:

A screen shot of a phone

AI-generated content may be incorrect.

1. Nhấp vào tab **Text** để chỉnh sửa mã XML, sau đó trích xuất chuỗi **"Message Received"** thành một tài nguyên có tên là **text\_header**.
2. Thêm thuộc tính **android:layout\_marginLeft="8dp"** vào **TextView** để bổ sung cho thuộc tính **layout\_marginStart** dành cho các phiên bản Android cũ hơn.

Mã XML cho tệp **activity\_second.xml** nên như sau:

2.4 Thêm một Intent vào Main Activity  
Trong nhiệm vụ này, bạn sẽ thêm một **Intent** tường minh (explicit Intent) vào **MainActivity**. Intent này được sử dụng để kích hoạt **SecondActivity** khi nút **Send** được nhấn.

**Các bước thực hiện:**

1. **Mở MainActivity.java**.
2. **Tạo một Intent mới** trong phương thức launchSecondActivity().

Constructor của **Intent** yêu cầu hai đối số cho một **explicit Intent**:

* **Context**: Ngữ cảnh ứng dụng (sử dụng this để tham chiếu đến **MainActivity**).
* **Component**: Thành phần cụ thể sẽ nhận Intent (sử dụng SecondActivity.class).

1. Gọi phương thức startActivity() với đối tượng **Intent** mới làm đối số.
2. Chạy ứng dụng.  
   Khi bạn nhấn nút **Send**, **MainActivity** sẽ gửi **Intent** và hệ thống Android sẽ khởi chạy **SecondActivity**, hiển thị màn hình của nó. Để quay lại **MainActivity**, bạn có thể nhấp vào nút **Up** (mũi tên quay lại ở thanh ứng dụng) hoặc nút **Back** ở dưới cùng của màn hình.

**Task 3: Gửi dữ liệu từ MainActivity sang SecondActivity**  
Trong nhiệm vụ trước, bạn đã thêm một intent rõ ràng vào **MainActivity** để khởi chạy **SecondActivity**. Bạn cũng có thể sử dụng một intent để **gửi dữ liệu** từ một activity này sang một activity khác trong quá trình khởi chạy.

Đối tượng intent có thể truyền dữ liệu tới activity mục tiêu theo hai cách: trong trường **data**, hoặc trong phần **intent extras**.

* Dữ liệu trong intent là một **URI** chỉ định **dữ liệu cụ thể cần xử lý**.
* Nếu thông tin bạn muốn truyền tới activity thông qua intent không phải là một URI, hoặc nếu bạn muốn gửi nhiều thông tin, bạn có thể đưa thông tin bổ sung vào phần **extras**.

**Extras của intent** là các cặp key/value được lưu trữ trong một **Bundle**. **Bundle** là một tập hợp dữ liệu được lưu trữ dưới dạng các cặp key/value.  
Để truyền thông tin từ một activity này sang một activity khác, bạn đưa các key và value vào phần **extras** của intent từ activity gửi và sau đó lấy chúng ra trong activity nhận.

Trong nhiệm vụ này, bạn sẽ chỉnh sửa intent rõ ràng trong **MainActivity** để thêm dữ liệu bổ sung (trong trường hợp này là một chuỗi do người dùng nhập vào) vào **extras** của intent. Sau đó, bạn chỉnh sửa **SecondActivity** để lấy dữ liệu này từ **extras** của intent và hiển thị trên màn hình.

**Thêm một EditText vào bố cục của MainActivity**

1. Mở tệp **activity\_main.xml**.
2. Kéo một thành phần **Plain Text (EditText)** từ **Palette** vào phía dưới của bố cục:
   * Thêm ràng buộc (constraints) vào cạnh trái của bố cục, cạnh dưới của bố cục, và cạnh trái của nút **Send**.
   * Đặt các thuộc tính trong bảng **Attributes** như sau:

A white box with black text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Bố cục mới trong tệp activity\_main.xml sẽ trông như thế này:

A white screen with a pink background

AI-generated content may be incorrect.

1. Nhấn vào tab **Text** để chỉnh sửa mã XML và trích xuất chuỗi **"Enter Your Message Here"** vào một tài nguyên (resource) với tên là **editText\_main**.

Mã XML cho bố cục sẽ trông giống như sau.

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.constraintlayout.widget.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:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <Button  
 android:id="@+id/button\_main"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginRight="16dp"  
 android:onClick="launchSecondActivity"  
 android:text="@string/button\_main"  
 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\_marginRight="8dp"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginBottom="16dp"  
 android:ems="10"  
 android:inputType="textLongMessage"  
 android:hint="@string/editText\_main"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/button\_main"  
 app:layout\_constraintStart\_toStartOf="parent" />  
</androidx.constraintlayout.widget.ConstraintLayout>

3.2 Thêm một chuỗi vào **Intent extras**  
Các **Intent extras** là các cặp key/value được lưu trong một **Bundle**. Một **Bundle** là một tập hợp dữ liệu được lưu trữ dưới dạng cặp key/value. Để truyền thông tin từ một **Activity** sang một **Activity** khác, bạn cần đưa các cặp key/value vào **Intent extras** từ **Activity** gửi, và sau đó lấy chúng ra từ **Intent extras** trong **Activity** nhận.

1. Mở **MainActivity**.
2. Thêm một hằng số **public** ở đầu lớp để định nghĩa key cho **Intent extra**:
3. Thêm một biến riêng tư ở đầu lớp để chứa EditText.
4. Trong phương thức onCreate(), sử dụng findViewById() để lấy tham chiếu đến EditText và gán nó cho biến riêng tư đó
5. Trong phương thức launchSecondActivity(), ngay dưới Intent mới, lấy văn bản từ EditText dưới dạng chuỗi.
6. Thêm chuỗi đó vào Intent dưới dạng một extra với hằng số EXTRA\_MESSAGE làm khóa và chuỗi làm giá trị



Phương thức onCreate() trong MainActivity bây giờ sẽ trông như sau:

Phương thức launchSecondActivity() trong MainActivity bây giờ sẽ trông như sau:

3.3 Thêm một TextView vào SecondActivity để hiển thị thông điệp

1. Mở tệp activity\_second.xml.
2. Kéo một TextView khác vào giao diện bên dưới TextView có ID là text\_header, và thêm các ràng buộc (constraints) vào cạnh trái của giao diện và cạnh dưới của text\_header.
3. Đặt các thuộc tính cho TextView mới trong bảng Attributes như sau:

A screenshot of a computer

AI-generated content may be incorrect.

Giao diện mới trông giống như trong bài tập trước, vì TextView mới chưa (chưa có) chứa bất kỳ văn bản nào, vì vậy nó không xuất hiện trên màn hình. Mã XML cho giao diện activity\_second.xml sẽ trông giống như sau

3.4 Sửa đổi SecondActivity để lấy dữ liệu extras và hiển thị thông điệp

1. Mở SecondActivity để thêm mã vào phương thức onCreate().
2. Lấy Intent kích hoạt Activity này:
3. Lấy chuỗi chứa thông điệp từ extras của Intent bằng cách sử dụng biến tĩnh MainActivity.EXTRA\_MESSAGE làm khóa:  
   
4. Sử dụng findViewById() để tham chiếu đến TextView cho thông điệp từ giao diện:  
   
5. Đặt văn bản của TextView thành chuỗi từ extra của Intent:
6. Chạy ứng dụng. Khi bạn nhập thông điệp trong MainActivity và nhấn Send, SecondActivity sẽ được mở và hiển thị thông điệp.  
   Phương thức onCreate() của SecondActivity sẽ trông như sau:

Nhiệm vụ 4: Trả lại dữ liệu cho Activity chính  
Bây giờ bạn đã có một ứng dụng khởi chạy một Activity mới và gửi dữ liệu đến đó, bước cuối cùng là trả lại dữ liệu từ Activity thứ hai về Activity chính. Bạn cũng sẽ sử dụng Intent và intent extras cho nhiệm vụ này.  
4.1 Thêm một EditText và một Button vào giao diện SecondActivity

1. Mở tệp strings.xml và thêm các tài nguyên chuỗi cho văn bản Button và gợi ý (hint) cho EditText mà bạn sẽ thêm vào SecondActivity:

2. Mở tệp activity\_main.xml và activity\_second.xml.  
3. Sao chép EditText và Button từ tệp layout activity\_main.xml và dán chúng vào layout activity\_second.xml.  
4. Trong activity\_second.xml, sửa đổi giá trị thuộc tính cho Button như sau:

A white rectangular box with black text

AI-generated content may be incorrect.

5. Trong activity\_second.xml, sửa đổi giá trị thuộc tính cho EditText như sau:

A white rectangular box with black text

AI-generated content may be incorrect.

6. Trong trình chỉnh sửa XML layout, nhấp vào returnReply, nhấn Alt+Enter (Option+Return trên Mac), và chọn Tạo 'returnReply(View)' trong 'SecondActivity'. Android Studio sẽ tạo ra một phương thức khung cho trình xử lý returnReply(). Bạn sẽ triển khai phương thức này trong nhiệm vụ tiếp theo.  
Giao diện mới cho activity\_second.xml trông như sau:

A white rectangular object with black border

AI-generated content may be incorrect.

Mã XML cho tệp layout activity\_second.xml như sau:

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.constraintlayout.widget.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:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".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\_marginLeft="8dp"  
 android:layout\_marginTop="8dp"  
 android:textAppearance="App.AppCompat.Medium"  
 app:layout\_constraintBottom\_toBottomOf="@+id/text\_header"  
 app:layout\_constraintStart\_toStartOf="parent" />  
 <Button  
 android:id="@+id/button\_second"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginRight="16dp"  
 android:onClick="returnReply"  
 android:text="@string/button\_second"  
 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\_marginRight="8dp"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginBottom="16dp"  
 android:ems="10"  
 android:inputType="textLongMessage"  
 android:hint="@string/editText\_second"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/button\_second"  
 app:layout\_constraintStart\_toStartOf="parent" />  
</androidx.constraintlayout.widget.ConstraintLayout>

**4.2 Tạo một Intent phản hồi trong Activity thứ hai**  
Dữ liệu phản hồi từ Activity thứ hai về Activity chính được gửi trong một Intent extra.  
Bạn tạo ra Intent trả lại này và đưa dữ liệu vào đó theo cách tương tự như cách bạn làm với Intent gửi đi.

1. Mở SecondActivity.
2. Ở đầu lớp, thêm một hằng số public để định nghĩa khóa cho Intent extra:
3. Thêm một biến riêng tư ở đầu lớp để chứa EditText.

4. Trong phương thức onCreate(), trước mã Intent, sử dụng findViewById() để lấy tham chiếu đến EditText và gán nó cho biến riêng tư đó:

5. Trong phương thức returnReply(), lấy văn bản của EditText dưới dạng chuỗi:

1. Trong phương thức returnReply(), tạo một intent mới cho phản hồi — không tái sử dụng đối tượng Intent mà bạn nhận được từ yêu cầu ban đầu.  
     
   7. Thêm chuỗi phản hồi từ EditText vào intent mới như một Intent extra. Vì extras là cặp khóa/giá trị, ở đây khóa là EXTRA\_REPLY và giá trị là reply:  
     
   8. Đặt kết quả thành RESULT\_OK để chỉ ra rằng phản hồi đã thành công. Lớp Activity định nghĩa các mã kết quả, bao gồm RESULT\_OK và RESULT\_CANCELLED.  
     
   9. Gọi finish() để đóng Activity và quay lại MainActivity.  
   

Mã cho SecondActivity bây giờ sẽ trông như sau:

package com.example.twoactivities;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
  
import androidx.activity.EdgeToEdge;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.graphics.Insets;  
import androidx.core.view.ViewCompat;  
import androidx.core.view.WindowInsetsCompat;  
  
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);  
 EdgeToEdge.*enable*(this);  
 setContentView(R.layout.*activity\_second*);  
 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;  
 });  
 Intent intent = getIntent();  
 String message = intent.getStringExtra(MainActivity.*EXTRA\_MESSAGE*);  
 TextView textView = findViewById(R.id.*text\_message*);  
 textView.setText(message);  
 mReply = findViewById(R.id.*editText\_second*);  
 }  
  
 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 Thêm các phần tử TextView để hiển thị phản hồi  
MainActivity cần một cách để hiển thị phản hồi mà SecondActivity gửi. Trong nhiệm vụ này, bạn sẽ thêm các phần tử TextView vào giao diện activity\_main.xml để hiển thị phản hồi trong MainActivity.  
Để làm cho nhiệm vụ này dễ dàng hơn, bạn sao chép các phần tử TextView mà bạn đã sử dụng trong SecondActivity.

1. Mở tệp strings.xml và thêm một tài nguyên chuỗi cho tiêu đề phản hồi:
2. Mở tệp activity\_main.xml và activity\_second.xml.
3. Sao chép hai phần tử TextView từ tệp layout activity\_second.xml và dán chúng vào giao diện activity\_main.xml phía trên Button.
4. Trong activity\_main.xml, sửa đổi giá trị thuộc tính cho TextView đầu tiên như sau:

A white rectangular box with black text

AI-generated content may be incorrect.

1. Trong activity\_main.xml, sửa đổi giá trị thuộc tính cho TextView thứ hai như sau:

A white rectangular box with black text

AI-generated content may be incorrect.

1. Thêm thuộc tính android:visibility vào mỗi TextView để làm chúng ban đầu vô hình. (Việc để chúng hiển thị trên màn hình mà không có nội dung có thể gây nhầm lẫn cho người dùng.)



Bạn sẽ làm cho các phần tử TextView này hiển thị sau khi dữ liệu phản hồi được truyền lại từ Activity thứ hai.

Giao diện activity\_main.xml trông giống như trong nhiệm vụ trước — mặc dù bạn đã thêm hai phần tử TextView mới vào giao diện. Vì bạn đã đặt các phần tử này thành vô hình, nên chúng không hiển thị trên màn hình.  
Dưới đây là mã XML cho tệp activity\_main.xml:

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.constraintlayout.widget.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:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
 <TextView  
 android:id="@+id/text\_header\_reply"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="8dp"  
 android:layout\_marginTop="16dp"  
 android:text="@string/text\_header\_reply"  
 android:textAppearance="@style/TextAppearance.AppCompat.Medium"  
 android:textStyle="bold"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 android:visibility="invisible"/>  
  
 <TextView  
 android:id="@+id/text\_message\_reply"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginTop="8dp"  
 android:textAppearance="@style/TextAppearance.AppCompat.Medium"  
 app:layout\_constraintTop\_toBottomOf="@+id/text\_header\_reply"  
 app:layout\_constraintStart\_toStartOf="parent"  
 android:visibility="invisible"/>  
  
 <Button  
 android:id="@+id/button\_main"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginRight="16dp"  
 android:onClick="launchSecondActivity"  
 android:text="@string/button\_main"  
 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\_marginRight="8dp"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginBottom="16dp"  
 android:ems="10"  
 android:inputType="textLongMessage"  
 android:hint="@string/editText\_main"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toStartOf="@+id/button\_main"  
 app:layout\_constraintStart\_toStartOf="parent" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>

4.4 Lấy phản hồi từ Intent extra và hiển thị nó  
Khi bạn sử dụng một Intent rõ ràng để bắt đầu một Activity khác, bạn có thể không mong đợi nhận lại bất kỳ dữ liệu nào — bạn chỉ đang kích hoạt Activity đó. Trong trường hợp này, bạn sử dụng startActivity() để bắt đầu Activity mới, như bạn đã làm trước đó trong bài thực hành này. Tuy nhiên, nếu bạn muốn lấy dữ liệu từ Activity đã được kích hoạt, bạn cần bắt đầu nó với startActivityForResult().  
Trong nhiệm vụ này, bạn sẽ chỉnh sửa ứng dụng để bắt đầu SecondActivity và mong đợi một kết quả, lấy dữ liệu trả về từ Intent và hiển thị dữ liệu đó trong các phần tử TextView mà bạn đã tạo trong nhiệm vụ trước.

1. Mở MainActivity.
2. Thêm một hằng số public ở đầu lớp để định nghĩa khóa cho một loại phản hồi mà bạn quan tâm:
3. Thêm hai biến riêng tư để chứa phần tử tiêu đề phản hồi và phần tử TextView phản hồi:
4. Trong phương thức onCreate(), sử dụng findViewById() để lấy tham chiếu từ giao diện đến phần tử tiêu đề phản hồi và phần tử TextView phản hồi. Gán các đối tượng view này cho các biến riêng tư:

Phương thức onCreate() đầy đủ bây giờ sẽ trông như sau:

1. Trong phương thức launchSecondActivity(), thay đổi lời gọi từ startActivity() thành startActivityForResult(), và bao gồm khóa TEXT\_REQUEST làm đối số:

startActivityForResult(intent, *TEXT\_REQUEST*);

1. **Ghi đè phương thức callback onActivityResult() với chữ ký sau:**



Ba tham số trong onActivityResult() chứa tất cả thông tin mà bạn cần để xử lý dữ liệu trả về: requestCode là mã yêu cầu mà bạn đã đặt khi khởi động Activity với startActivityForResult(), resultCode là mã kết quả được đặt trong Activity đã được khởi động (thường là một trong RESULT\_OK hoặc RESULT\_CANCELED), và Intent data chứa dữ liệu trả về từ Activity đã được khởi động.

1. **Bên trong onActivityResult(), gọi super.onActivityResult():**
2. **Thêm mã để kiểm tra TEXT\_REQUEST để đảm bảo bạn xử lý đúng kết quả của Intent, trong trường hợp có nhiều kết quả. Cũng kiểm tra RESULT\_OK để đảm bảo rằng yêu cầu đã thành công:**

Lớp Activity định nghĩa các mã kết quả. Mã có thể là RESULT\_OK (yêu cầu thành công), RESULT\_CANCELED (người dùng hủy bỏ thao tác), hoặc RESULT\_FIRST\_USER (dành cho mã kết quả do người dùng tự định nghĩa).

1. **Bên trong khối if con, lấy Intent extra từ phản hồi Intent (data)**: Ở đây, khóa cho extra là hằng số EXTRA\_REPLY từ SecondActivity. Bạn sẽ sử dụng data.getStringExtra() để lấy giá trị trả về:
2. **Đặt độ hiển thị của tiêu đề phản hồi (reply header) thành true**:
3. **Đặt nội dung cho TextView phản hồi (reply TextView) với reply và đặt độ hiển thị của nó thành true**:

**Phương thức onActivityResult() đầy đủ:**

1. **Chạy ứng dụng.**  
   Bây giờ, khi bạn gửi một tin nhắn đến SecondActivity và nhận được phản hồi, MainActivity sẽ được cập nhật để hiển thị phản hồi.

A white rectangular object with buttons and symbols

AI-generated content may be incorrect.

### Vòng đời của Activity và trạng thái

**Giới thiệu**  
Trong bài thực hành này, bạn sẽ tìm hiểu thêm về **vòng đời của Activity**. Vòng đời là tập hợp các trạng thái mà một Activity có thể trải qua trong suốt thời gian tồn tại của nó, từ khi được tạo ra đến khi bị hủy và tài nguyên của nó được hệ thống thu hồi. Khi người dùng điều hướng giữa các Activity trong ứng dụng của bạn (cũng như khi vào hoặc thoát khỏi ứng dụng), các Activity sẽ chuyển đổi giữa các trạng thái khác nhau trong vòng đời của chúng.

Mỗi giai đoạn trong vòng đời của một Activity có một phương thức callback tương ứng: **onCreate()**, **onStart()**, **onPause()**, và nhiều hơn nữa. Khi một Activity thay đổi trạng thái, phương thức callback tương ứng sẽ được gọi. Bạn đã từng thấy một trong những phương thức này: **onCreate()**. Bằng cách ghi đè bất kỳ phương thức callback nào của vòng đời trong các lớp **Activity** của bạn, bạn có thể thay đổi hành vi mặc định của Activity để phản hồi các hành động của người dùng hoặc hệ thống.

Trạng thái của Activity cũng có thể thay đổi do các thay đổi cấu hình của thiết bị, ví dụ như khi người dùng xoay thiết bị từ chế độ dọc sang ngang. Khi những thay đổi cấu hình này xảy ra, Activity sẽ bị hủy và được tạo lại trong trạng thái mặc định, và người dùng có thể mất thông tin mà họ đã nhập vào Activity. Để tránh làm người dùng bối rối, điều quan trọng là bạn cần phát triển ứng dụng của mình để ngăn ngừa việc mất dữ liệu ngoài ý muốn. Trong phần sau của bài thực hành này, bạn sẽ thử nghiệm với các thay đổi cấu hình và học cách bảo toàn trạng thái của Activity để phản hồi những thay đổi cấu hình thiết bị và các sự kiện khác trong vòng đời của Activity.

Trong bài thực hành này, bạn sẽ thêm các câu lệnh ghi nhật ký (logging statements) vào ứng dụng **TwoActivities** và quan sát các thay đổi vòng đời của Activity khi bạn sử dụng ứng dụng. Sau đó, bạn sẽ bắt đầu làm việc với những thay đổi này và khám phá cách xử lý đầu vào của người dùng trong những điều kiện đó.

**Những gì bạn cần biết trước**  
Bạn cần có khả năng:  
● Tạo và chạy một dự án ứng dụng trong Android Studio.  
● Thêm các câu lệnh ghi nhật ký (log statements) vào ứng dụng của bạn và xem các nhật ký đó trong bảng **Logcat**.  
● Hiểu và làm việc với **Activity** và **Intent**, đồng thời thoải mái khi tương tác với chúng.

**Những gì bạn sẽ học**  
● Cách hoạt động của vòng đời **Activity**.  
● Khi nào một **Activity** bắt đầu, tạm dừng, dừng lại và bị hủy.  
● Các phương thức callback của vòng đời liên quan đến những thay đổi của **Activity**.  
● Tác động của các hành động (chẳng hạn như thay đổi cấu hình) có thể dẫn đến các sự kiện trong vòng đời **Activity**.  
● Cách giữ trạng thái của **Activity** qua các sự kiện vòng đời.

**Những gì bạn sẽ làm**  
● Thêm mã vào ứng dụng **TwoActivities** từ bài thực hành trước để triển khai các phương thức callback vòng đời khác nhau của **Activity**, bao gồm cả các câu lệnh ghi nhật ký.  
● Quan sát các thay đổi trạng thái khi ứng dụng của bạn chạy và khi bạn tương tác với từng **Activity** trong ứng dụng.  
● Sửa đổi ứng dụng của bạn để giữ trạng thái phiên bản của một **Activity** khi nó được tạo lại một cách không mong muốn do hành vi của người dùng hoặc thay đổi cấu hình trên thiết bị.

**Tổng quan về ứng dụng**  
Trong bài thực hành này, bạn sẽ bổ sung vào ứng dụng **TwoActivities**. Ứng dụng sẽ trông và hoạt động gần giống như trong bài codelab trước. Nó chứa hai triển khai **Activity** và cung cấp cho người dùng khả năng gửi dữ liệu giữa chúng. Những thay đổi bạn thực hiện trong bài thực hành này sẽ không ảnh hưởng đến hành vi giao diện người dùng có thể thấy được của ứng dụng.

**Nhiệm vụ 1: Thêm các callback vòng đời vào TwoActivities**  
Trong nhiệm vụ này, bạn sẽ triển khai tất cả các phương thức callback vòng đời của **Activity** để in thông báo vào **logcat** khi các phương thức này được gọi. Các thông báo log này sẽ cho phép bạn thấy khi nào vòng đời **Activity** thay đổi trạng thái và cách những thay đổi trạng thái vòng đời này ảnh hưởng đến ứng dụng khi nó chạy.

**1.1 (Tùy chọn) Sao chép dự án TwoActivities**  
Đối với các nhiệm vụ trong bài thực hành này, bạn sẽ sửa đổi dự án **TwoActivities** hiện có mà bạn đã xây dựng trong bài thực hành trước. Nếu bạn muốn giữ nguyên dự án TwoActivities trước đó, hãy làm theo các bước trong **Phụ lục: Tiện ích** để tạo một bản sao của dự án.

**1.2 Thêm các callback vào MainActivity**

1. Mở dự án **TwoActivities** trong Android Studio, sau đó mở **MainActivity** từ **Project > Android** pane.
2. Trong phương thức **onCreate()**, thêm các câu lệnh log sau đây:

Log.d(LOG\_TAG, " ");

Log.d(LOG\_TAG, "onCreate");

1. Thêm một phương thức ghi đè cho callback onStart(), với một câu lệnh ghi nhật ký cho sự kiện đó:

Để tiết kiệm thời gian, chọn Code > Override Methods trong Android Studio. Một hộp thoại sẽ xuất hiện với tất cả các phương thức có thể ghi đè trong lớp của bạn. Chọn một hoặc nhiều phương thức callback từ danh sách sẽ chèn một mẫu hoàn chỉnh cho các phương thức đó, bao gồm cả lời gọi bắt buộc đến lớp cha (superclass).

1. Sử dụng phương thức onStart() làm mẫu để triển khai các callback vòng đời onPause(), onRestart(), onResume(), onStop(), và onDestroy().  
   Tất cả các phương thức callback có chữ ký (signature) giống nhau (ngoại trừ tên). Nếu bạn Copy và Paste phương thức onStart() để tạo các phương thức callback khác, đừng quên cập nhật nội dung để gọi đúng phương thức trong lớp cha, và ghi nhật ký đúng phương thức.

1. Chạy ứng dụng của bạn.  
2. Nhấp vào tab Logcat ở dưới cùng của Android Studio để hiển thị bảng Logcat. Bạn sẽ thấy ba thông báo nhật ký hiển thị ba trạng thái vòng đời mà Activity đã chuyển qua khi nó bắt đầu:

**1.3 Triển khai các phương thức callback vòng đời trong SecondActivity**

1. Mở SecondActivity.
2. Ở đầu lớp, thêm một hằng số cho biến LOG\_TAG:
3. **Thêm các phương thức callback vòng đời và câu lệnh log vào SecondActivity.** (Bạn có thể sao chép và dán các phương thức callback từ MainActivity).
4. **Thêm một câu lệnh log vào phương thức returnReply() ngay trước khi gọi finish().**

1.4 **Quan sát log khi ứng dụng chạy**

1. Chạy ứng dụng của bạn.
2. Nhấp vào tab **Logcat** ở dưới cùng trong Android Studio để hiển thị cửa sổ Logcat.
3. Nhập **Activity** vào ô tìm kiếm.

Logcat của Android có thể rất dài và lộn xộn. Vì biến **LOG\_TAG** trong mỗi lớp chứa từ **MainActivity** hoặc **SecondActivity**, từ khóa này giúp bạn lọc log chỉ hiển thị những thông tin bạn quan tâm.

Thử nghiệm sử dụng ứng dụng của bạn và ghi chú lại các sự kiện vòng đời xảy ra khi thực hiện các hành động khác nhau. Cụ thể, thử các điều sau:

* Sử dụng ứng dụng bình thường (gửi tin nhắn, trả lời bằng tin nhắn khác).
* Dùng nút **Back** để quay lại từ **SecondActivity** về **MainActivity**.
* Dùng nút **Up arrow** trong thanh công cụ ứng dụng để quay lại từ **SecondActivity** về **MainActivity**.
* Xoay thiết bị trên cả **MainActivity** và **SecondActivity** vào các thời điểm khác nhau trong ứng dụng và quan sát những gì xảy ra trong log và trên màn hình.
* Nhấn nút **Overview** (nút vuông bên phải nút Home) và đóng ứng dụng (chạm vào nút **X**).
* Quay lại màn hình chính và khởi động lại ứng dụng của bạn.

**MẸO**: Nếu bạn đang chạy ứng dụng trong giả lập, bạn có thể mô phỏng việc xoay màn hình bằng cách nhấn **Control+F11** hoặc **Control+Function+F11**.

**Mã giải pháp cho tác vụ 1**

Dưới đây là đoạn mã thêm vào **MainActivity**, nhưng không phải là toàn bộ lớp.

Phương thức onCreate():

Các phương thức vòng đời khác:

**SecondActivity**

Dưới đây là đoạn mã thêm vào **SecondActivity**, nhưng không phải là toàn bộ lớp.

Ở đầu lớp **SecondActivity**:

Phương thức **returnReply()**:

Các phương thức vòng đời khác:  
Giống như đối với **MainActivity**, ở trên.

**Tác vụ 2: Lưu và phục hồi trạng thái của Activity**

Tùy thuộc vào tài nguyên hệ thống và hành vi người dùng, mỗi **Activity** trong ứng dụng của bạn có thể bị hủy và tái tạo lại nhiều hơn bạn nghĩ.

Bạn có thể đã nhận thấy hành vi này trong phần trước khi bạn xoay thiết bị hoặc giả lập. Xoay thiết bị là một ví dụ của thay đổi cấu hình thiết bị. Mặc dù xoay màn hình là thay đổi cấu hình phổ biến nhất, tất cả các thay đổi cấu hình đều dẫn đến việc **Activity** hiện tại bị hủy và tái tạo lại như thể nó là mới. Nếu bạn không xử lý hành vi này trong mã của mình, khi xảy ra thay đổi cấu hình, giao diện **Activity** có thể trở lại giao diện mặc định và các giá trị ban đầu, và người dùng có thể mất vị trí, dữ liệu hoặc trạng thái tiến trình trong ứng dụng của bạn.

Trạng thái của mỗi **Activity** được lưu dưới dạng một tập hợp các cặp khóa/giá trị trong một đối tượng **Bundle** gọi là trạng thái phiên bản **Activity**. Hệ thống lưu thông tin trạng thái mặc định vào **Bundle** của trạng thái phiên bản ngay trước khi **Activity** bị dừng và chuyển **Bundle** đó đến phiên bản **Activity** mới để phục hồi.

Để tránh mất dữ liệu trong **Activity** khi nó bị hủy và tái tạo lại một cách bất ngờ, bạn cần triển khai phương thức **onSaveInstanceState()**. Hệ thống gọi phương thức này trên **Activity** của bạn (giữa **onPause()** và **onStop()**) khi có khả năng **Activity** có thể bị hủy và tái tạo lại.

Dữ liệu bạn lưu trong trạng thái phiên bản chỉ áp dụng cho chính phiên bản **Activity** này trong phiên làm việc hiện tại của ứng dụng. Khi bạn dừng và khởi động lại một phiên làm việc mới, trạng thái phiên bản **Activity** bị mất và **Activity** trở lại giao diện mặc định. Nếu bạn cần lưu trữ dữ liệu người dùng giữa các phiên làm việc của ứng dụng, hãy sử dụng **shared preferences** hoặc cơ sở dữ liệu. Bạn sẽ học về cả hai trong một bài thực hành sau.

**2.1 Lưu trạng thái phiên bản Activity với onSaveInstanceState()**

Bạn có thể đã nhận thấy rằng việc xoay thiết bị không ảnh hưởng đến trạng thái của **SecondActivity** chút nào. Điều này là vì giao diện và trạng thái của **SecondActivity** được tạo ra từ giao diện và **Intent** kích hoạt nó. Ngay cả khi **Activity** bị tái tạo, **Intent** vẫn còn và dữ liệu trong **Intent** đó vẫn được sử dụng mỗi khi phương thức **onCreate()** trong **SecondActivity** được gọi.

Ngoài ra, bạn có thể nhận thấy rằng trong mỗi **Activity**, bất kỳ văn bản nào bạn đã nhập vào các phần tử **EditText** của tin nhắn hoặc trả lời đều được giữ lại ngay cả khi thiết bị bị xoay. Điều này là vì thông tin trạng thái của một số phần tử **View** trong giao diện của bạn được tự động lưu trữ qua các thay đổi cấu hình, và giá trị hiện tại của **EditText** là một trong những trường hợp đó.

Vì vậy, trạng thái **Activity** mà bạn quan tâm chỉ là các phần tử **TextView** cho tiêu đề trả lời và văn bản trả lời trong **MainActivity**. Cả hai phần tử **TextView** này đều vô hình mặc định; chúng chỉ hiển thị khi bạn gửi tin nhắn trở lại **MainActivity** từ **SecondActivity**.

Trong tác vụ này, bạn sẽ thêm mã để bảo vệ trạng thái phiên bản của hai phần tử **TextView** này bằng cách sử dụng **onSaveInstanceState()**.

1. Mở **MainActivity**.
2. Thêm cấu trúc triển khai của **onSaveInstanceState()** vào **Activity**, hoặc sử dụng **Code > Override Methods** để chèn một phương thức ghi đè cấu trúc.
3. Kiểm tra xem tiêu đề có đang hiển thị không, và nếu có, hãy đưa trạng thái hiển thị đó vào trong **Bundle** trạng thái bằng phương thức **putBoolean()** với khóa **"reply\_visible"**:

Hãy nhớ rằng tiêu đề và văn bản trả lời sẽ được đánh dấu là vô hình cho đến khi có một câu trả lời từ **SecondActivity**. Nếu tiêu đề đang hiển thị, điều này có nghĩa là có dữ liệu trả lời cần được lưu lại.

Lưu ý rằng chúng ta chỉ quan tâm đến trạng thái hiển thị đó — văn bản thực tế của tiêu đề không cần phải lưu lại, vì văn bản đó không bao giờ thay đổi.

1. Trong cùng một kiểm tra đó, thêm văn bản trả lời vào trong **Bundle**:

Ở đây, **mReplyTextView** là phần tử **TextView** chứa văn bản trả lời. Phương thức **putString()** sẽ lưu lại văn bản trả lời vào trong **Bundle**.

Nếu tiêu đề đang hiển thị, bạn có thể giả định rằng tin nhắn trả lời cũng đang hiển thị. Bạn không cần phải kiểm tra hay lưu trạng thái hiển thị hiện tại của tin nhắn trả lời. Chỉ cần lưu lại văn bản thực tế của tin nhắn vào trong **Bundle** trạng thái với khóa **"reply\_text"**.

Bạn chỉ lưu trạng thái của những phần tử **View** có thể thay đổi sau khi **Activity** được tạo. Các phần tử **View** khác trong ứng dụng của bạn (như **EditText**, **Button**) có thể được tái tạo từ giao diện mặc định bất kỳ lúc nào.

Lưu ý rằng hệ thống sẽ tự động lưu trạng thái của một số phần tử **View**, chẳng hạn như nội dung của **EditText**.

**2.2 Khôi phục trạng thái phiên bản Activity trong onCreate()**

Sau khi bạn đã lưu trạng thái phiên bản **Activity**, bạn cũng cần phải phục hồi nó khi **Activity** được tái tạo. Bạn có thể làm điều này trong **onCreate()**, hoặc bằng cách triển khai phương thức **onRestoreInstanceState()**, phương thức này sẽ được gọi sau **onStart()** sau khi **Activity** được tạo.

Hầu hết thời gian, việc phục hồi trạng thái **Activity** trong **onCreate()** là lựa chọn tốt hơn, để đảm bảo rằng giao diện người dùng, bao gồm cả trạng thái, có sẵn càng sớm càng tốt. Tuy nhiên, đôi khi cũng tiện lợi khi làm điều này trong **onRestoreInstanceState()** sau khi tất cả việc khởi tạo đã hoàn tất, hoặc để cho các lớp con quyết định có sử dụng triển khai mặc định của bạn hay không.

1. Trong phương thức **onCreate()**, sau khi các biến **View** được khởi tạo với **findViewById()**, thêm một kiểm tra để đảm bảo rằng **savedInstanceState** không phải là null.

Khi **Activity** của bạn được tạo, hệ thống sẽ truyền **Bundle** trạng thái vào **onCreate()** như là đối số duy nhất. Lần đầu tiên **onCreate()** được gọi và ứng dụng của bạn khởi động, **Bundle** sẽ là null — không có trạng thái tồn tại khi ứng dụng của bạn khởi động lần đầu tiên. Những lần gọi sau của **onCreate()** sẽ có một **Bundle** được điền dữ liệu từ những gì bạn đã lưu trong **onSaveInstanceState()**.

1. Bên trong kiểm tra đó, lấy trạng thái hiển thị hiện tại (đúng hoặc sai) từ **Bundle** với khóa **"reply\_visible"**.
2. Thêm một kiểm tra bên dưới dòng trước đó cho biến **isVisible**.

Nếu có một khóa **reply\_visible** trong **Bundle** trạng thái (và do đó **isVisible** là true), bạn sẽ cần phải khôi phục trạng thái.

1. Bên trong kiểm tra **isVisible**, làm cho tiêu đề hiển thị.
2. Lấy văn bản trả lời từ **Bundle** với khóa **"reply\_text"**, và đặt **TextView** trả lời để hiển thị chuỗi đó.
3. Làm cho **TextView** trả lời hiển thị nữa:
4. Chạy ứng dụng. Thử xoay thiết bị hoặc trình giả lập để đảm bảo rằng tin nhắn trả lời (nếu có) vẫn hiển thị trên màn hình sau khi **Activity** được tái tạo.

Mã giải pháp cho nhiệm vụ 2

**MainActivity**

Các đoạn mã sau đây hiển thị mã đã thêm vào **MainActivity**, nhưng không phải là toàn bộ lớp.

Phương thức **onSaveInstanceState()**:

Phuong thuc onCreat():

protected void onSaveInstanceState(@NonNull Bundle outState) {  
 super.onSaveInstanceState(outState);  
 if(mReplyHeadTextView.getVisibility() == View.*VISIBLE*)  
 {  
  
 outState.putBoolean("reply\_visible", true);  
 outState.putString("reply\_text", mReplyTextView.getText().toString());  
 }  
}  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 EdgeToEdge.*enable*(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;  
 });  
 mMessageEditText = findViewById(R.id.*editText\_main*);  
 mReplyHeadTextView = findViewById(R.id.*text\_header\_reply*);  
 mReplyTextView = findViewById(R.id.*text\_message\_reply*);  
 if(savedInstanceState != null){  
 boolean isVisisble = savedInstanceState.getBoolean("reply\_visible");  
 if (isVisisble){  
 mReplyHeadTextView.setVisibility(View.*VISIBLE*);  
 mReplyTextView.setText(savedInstanceState.getString("reply\_text"));  
 mReplyTextView.setVisibility(View.*VISIBLE*);  
 }  
 }

Dự án hoàn chỉnh:

Dự án Android Studio: TwoActivitiesLifecycle

**Thách thức lập trình** *Lưu ý: Tất cả các thách thức lập trình là tùy chọn và không phải là yêu cầu bắt buộc cho các bài học sau.*

**Thách thức**: Tạo một ứng dụng danh sách mua sắm đơn giản với một hoạt động chính cho danh sách mà người dùng đang xây dựng, và một hoạt động thứ hai cho danh sách các mặt hàng mua sắm thông thường.  
● Hoạt động chính nên chứa danh sách để xây dựng, bao gồm mười phần tử TextView trống.  
● Một nút **Thêm mục** trên hoạt động chính sẽ mở một hoạt động thứ hai chứa danh sách các mặt hàng mua sắm thông thường (Phô mai, Gạo, Táo, v.v.). Sử dụng các phần tử Button để hiển thị các mục.  
● Việc chọn một mục sẽ trả người dùng về hoạt động chính và cập nhật một TextView trống để bao gồm mục đã chọn.  
Sử dụng một Intent để truyền thông tin từ một hoạt động sang hoạt động khác. Đảm bảo rằng trạng thái hiện tại của danh sách mua sắm được lưu lại khi người dùng xoay thiết bị.

**Tóm tắt**  
● Vòng đời của Activity là một tập hợp các trạng thái mà một Activity chuyển qua, bắt đầu khi nó được tạo ra và kết thúc khi hệ thống Android thu hồi tài nguyên cho Activity đó.  
● Khi người dùng điều hướng từ một Activity sang một Activity khác, và trong và ngoài ứng dụng của bạn, mỗi Activity di chuyển giữa các trạng thái trong vòng đời Activity.  
● Mỗi trạng thái trong vòng đời Activity có một phương thức callback tương ứng mà bạn có thể ghi đè trong lớp Activity của bạn.  
● Các phương thức vòng đời bao gồm: onCreate(), onStart(), onPause(), onRestart(), onResume(), onStop(), onDestroy().  
● Việc ghi đè một phương thức callback của vòng đời cho phép bạn thêm hành vi xảy ra khi Activity của bạn chuyển sang trạng thái đó.  
● Bạn có thể thêm các phương thức ghi đè khung vào các lớp của mình trong Android Studio bằng cách vào **Code > Override**.

Dưới đây là bản dịch tiếng Việt của đoạn mô tả:

● Các thay đổi cấu hình thiết bị như xoay màn hình dẫn đến Activity bị hủy và tái tạo lại như thể nó là mới.  
● Một phần trạng thái của Activity được lưu lại khi có sự thay đổi cấu hình, bao gồm các giá trị hiện tại của các phần tử EditText. Đối với tất cả các dữ liệu khác, bạn phải tự lưu trữ dữ liệu đó.  
● Lưu trạng thái của Activity trong phương thức onSaveInstanceState().  
● Dữ liệu trạng thái của instance được lưu trữ dưới dạng các cặp khóa/giá trị đơn giản trong một Bundle. Sử dụng các phương thức của Bundle để đưa dữ liệu vào và lấy dữ liệu ra khỏi Bundle.  
● Khôi phục trạng thái của instance trong onCreate(), đó là cách được ưu tiên, hoặc onRestoreInstanceState().

### Intent ngầm định

## Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ

### Trình gỡ lỗi

### Kiểm thử đơn vị

### Thư viện hỗ trợ

# TRẢI NGHIỆM NGƯỜI DÙNG

## Tương tác người dùng

### Hình ảnh có thể chọn

### Các điều khiển nhập liệu

### Menu và bộ chọn

### Điều hướng người dùng

### RecycleView

## Trải nghiệm người dùng thú vị

### Hình vẽ, định kiểu và chủ đề

### Thẻ và màu sắc

### Bố cục thích ứng

## Kiểm thử giao diện người dùng

### Espresso cho việc kiểm tra UI

# LÀM VIỆC TRONG NỀN

## Các tác vụ nền

### AsyncTask

### AsyncTask và AsyncTaskLoader

### Broadcast receivers

## Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền

### Thông báo

### Trình quản lý cảnh báo

### JobScheduler

# LƯU DỮ LIỆU NGƯỜI DÙNG

## Tùy chọn và cài đặt

### Shared preferences

### Cài đặt ứng dụng

## Lưu trữ dữ liệu với Room

### Room, LiveData và ViewModel

### Room, LiveData và ViewModel