

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.....	7
Bài 1) Tạo ứng dụng đầu tiên .....	7
1.1) Android Studio và Hello World.....	7
2.2 Khám phá ngăn Project > Android.....	16
2.3 Khám phá thư mục Gradle Scripts.....	17
2.4 Khám phá thư mục app và res .....	19
2.5 Khám phá thư mục manifests .....	21
Nhiệm vụ 3: Sử dụng thiết bị ảo (trình giả lập) .....	22
3.1 Tạo một thiết bị ảo Android (AVD) .....	22
Nhiệm vụ 4: (Tùy chọn) Sử dụng thiết bị vật lý.....	27
4.1 Bật chế độ Gỡ lỗi USB (USB Debugging).....	27
4.2 Chạy ứng dụng trên thiết bị .....	28
Xử lý sự cố (Troubleshooting) .....	28
6.1 Xem Logcat pane .....	28
6.2 Thêm câu lệnh log vào ứng dụng của bạn .....	29
Các bước thực hiện: .....	30
Thử thách:.....	31
Tóm tắt .....	32
Các khái niệm liên quan.....	32
Tìm hiểu thêm .....	33
Bài tập về nhà .....	33
Trả lời các câu hỏi .....	33
Nộp bài kiểm tra ứng dụng .....	35
1.2) Giao diện người dùng tương tác đầu tiên .....	35
Giới thiệu .....	35

Những gì bạn cần biết trước.....	36
Những gì bạn sẽ học .....	36
Những gì bạn sẽ làm .....	36
Tổng quan về ứng dụng .....	37
Bài 1: Tạo và khám phá một dự án mới.....	38
Khám phá trình chỉnh sửa bố cục (Layout Editor).....	38
Khám phá trình chỉnh sửa bố cục.....	39
Bài 2: Thêm các phần tử giao diện vào trình chỉnh sửa bố cục.....	40
2.1 Kiểm tra ràng buộc của phần tử.....	40
Các nút điều khiển trong blueprint hoặc design pane:.....	41
2.2 Thêm một nút Button vào bố cục.....	41
Các bước thêm một Button: .....	42
2.3 Thêm một Button thứ hai vào bố cục.....	42
Xóa ràng buộc của phần tử UI.....	43
1.3) Trình chỉnh sửa bố cục .....	68
1.2 Tạo biến thể bố cục cho chế độ ngang.....	74
1.4 Thay đổi bố cục cho chế độ ngang .....	75
2.1 Thay đổi root view group thành LinearLayout .....	86
2.2 Thay đổi thuộc tính của các phần tử để phù hợp với LinearLayout .....	87
2.3 Thay đổi vị trí của các phần tử trong LinearLayout .....	90
2.4 Thêm thuộc tính weight cho phần tử TextView.....	92
Task 3: Thay đổi bố cục sang RelativeLayout .....	95
3.1 Thay đổi LinearLayout thành RelativeLayout .....	95
3.2 Sắp xếp lại các View trong RelativeLayout .....	96
Mẹo:.....	99
Mã giải pháp cho Task 3.....	99
Tóm tắt .....	101
Tài liệu liên quan.....	103

Bài tập về nhà .....	103
Trả lời các câu hỏi .....	104
1.4) Văn bản và các chế độ cuộn .....	105
Cách tổ chức bố cục cuộn hợp lý .....	106
Những kiến thức bạn cần biết trước.....	106
Những gì bạn sẽ học .....	106
Những gì bạn sẽ làm trong bài thực hành.....	106
Tổng quan về ứng dụng .....	107
1.5) Tài nguyên có sẵn.....	121
Giới thiệu .....	121
Tổng quan về ứng dụng .....	122
Nhiệm vụ 1: Thay đổi biểu tượng khởi chạy.....	122
1.1 Khám phá tài liệu chính thức của Android .....	124
1.2 Thêm tài nguyên hình ảnh cho biểu tượng khởi chạy.....	124
Nhiệm vụ 2: Sử dụng mẫu dự án .....	128
2.1 Khám phá kiến trúc của Basic Activity .....	128
2.2 Tùy chỉnh ứng dụng được tạo bởi mẫu .....	131
2.3 Khám phá cách thêm hoạt động bằng mẫu.....	134
3.1 Mã nguồn mẫu Android .....	134
3.2 Sử dụng SDK Manager để cài đặt tài liệu ngoại tuyến .....	134
4.1 Tìm kiếm trên Stack Overflow bằng thẻ (tags).....	135
Tóm tắt .....	136
Tìm hiểu thêm .....	136
Bài tập về nhà .....	137
Bài 2) Activities .....	138
2.1) Activity và Intent .....	138
Giới thiệu .....	138
Có hai loại Intent .....	138

Những kiến thức cần có trước .....	138
Những gì bạn sẽ học .....	139
Những gì bạn sẽ làm .....	139
Tổng quan về ứng dụng .....	139
Nhiệm vụ 1: Tạo dự án TwoActivities.....	139
1.1 Tạo dự án TwoActivities .....	140
1.2 Định nghĩa bố cục cho Main Activity .....	140
1.3     định nghĩa hành động của button .....	142
Nhiệm vụ 2: tạo và khởi chạy activity thứ hai .....	143
2.1 tạo activity thứ hai.....	144
2.2 chỉnh sửa tệp androidmanifest.xml.....	144
2.3 định nghĩa layout cho activity thứ hai .....	145
2.4 Thêm intent vào main activity.....	148
Nhiệm vụ 3: Gửi dữ liệu từ MainActivity sang SecondActivity .....	149
3.1 Thêm một EditText vào bố cục của MainActivity .....	149
3.2 Thêm một chuỗi vào Intent extras .....	151
3.3 Thêm một TextView vào SecondActivity để hiển thị thông điệp .....	153
3.4 Chính sửa SecondActivity để lấy extras và hiển thị thông điệp .....	154
Nhiệm vụ 4: Trả dữ liệu về MainActivity .....	155
4.1 Thêm một EditText và một Button vào giao diện của SecondActivity .....	155
4.2 Tạo một Intent phản hồi trong SecondActivity .....	158
4.3 Thêm các phần tử TextView để hiển thị phản hồi.....	159
4.4 Nhận phản hồi từ Intent extra và hiển thị nó .....	162
Mã giải pháp.....	165
Thử thách lập trình .....	165
Tóm tắt.....	165
Khái niệm liên quan .....	165
Tìm hiểu thêm .....	165

2.2) Vòng đời của Activity và trạng thái.....	166
Giới thiệu .....	166
Những kiến thức bạn cần có trước .....	167
Những gì bạn sẽ học .....	167
Những gì bạn sẽ làm.....	167
Tổng quan về ứng dụng .....	167
Nhiệm vụ 1: Thêm các phương thức gọi lại vòng đời vào TwoActivities .....	167
1.1 (Tùy chọn) Sao chép dự án TwoActivities .....	168
1.2 Triển khai các phương thức gọi lại vào MainActivity .....	168
2.3) Intent ngầm định Giới thiệu Trong một phần trước, bạn đã học về explicit intents (intents rõ ràng). Trong một explicit intent, bạn thực hiện một hoạt động trong ứng dụng của bạn, hoặc trong một ứng dụng khác, bằng cách gửi một intent với tên lớp đầy đủ của hoạt động. Trong bài học này, bạn sẽ tìm hiểu thêm về implicit intents (intents gián tiếp) và cách sử dụng chúng để thực hiện các hoạt động. ....	181
Bài 3) Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ.....	202
3.1) Trình gỡ lỗi.....	202
3.2) Kiểm thử đơn vị .....	213
3.3) Giải pháp mã cho nhiệm vụ 2: .....	223
3.4) Thư viện hỗ trợ .....	227
CHƯƠNG 2. TRẢI NGHIỆM NGƯỜI DÙNG.....	228
Bài 1) Tương tác người dùng.....	228
1.1) Hình ảnh có thể chọn.....	228
1.2) Các điều khiển nhập liệu .....	228
1.3) Menu và bộ chọn .....	228
1.4) Điều hướng người dùng.....	228
1.5) RecyclerView.....	228
Bài 2) Trải nghiệm người dùng thú vị .....	228
2.1) Hình vẽ, định kiểu và chủ đề .....	228
2.2) Thẻ và màu sắc .....	228

2.3) Bố cục thích ứng .....	228
Bài 3) Kiểm thử giao diện người dùng.....	228
3.1) Espresso cho việc kiểm tra UI.....	228
<b>CHƯƠNG 3. LÀM VIỆC TRONG NỀN.....</b>	<b>228</b>
Bài 1) Các tác vụ nền.....	228
1.1) AsyncTask .....	228
1.2) AsyncTask và AsyncTaskLoader.....	228
1.3) Broadcast receivers.....	228
Bài 2) Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền.....	228
2.1) Thông báo.....	228
2.2) Trình quản lý cảnh báo.....	228
2.3) JobScheduler .....	228
<b>CHƯƠNG 4. LƯU DỮ LIỆU NGƯỜI DÙNG.....</b>	<b>229</b>
Bài 1) Tùy chọn và cài đặt .....	229
1.1) Shared preferences .....	229
1.2) Cài đặt ứng dụng .....	241
Bài 2) Lưu trữ dữ liệu với Room .....	253
2.1) Room, LiveData và ViewModel.....	253
2.2) Room, LiveData và ViewModel.....	253
3.1) Trinhf gowx loi .....	

## **CHƯƠNG 1. LÀM QUEN**

### **Bài 1) Tạo ứng dụng đầu tiên**

#### **1.1) Android Studio và Hello World**

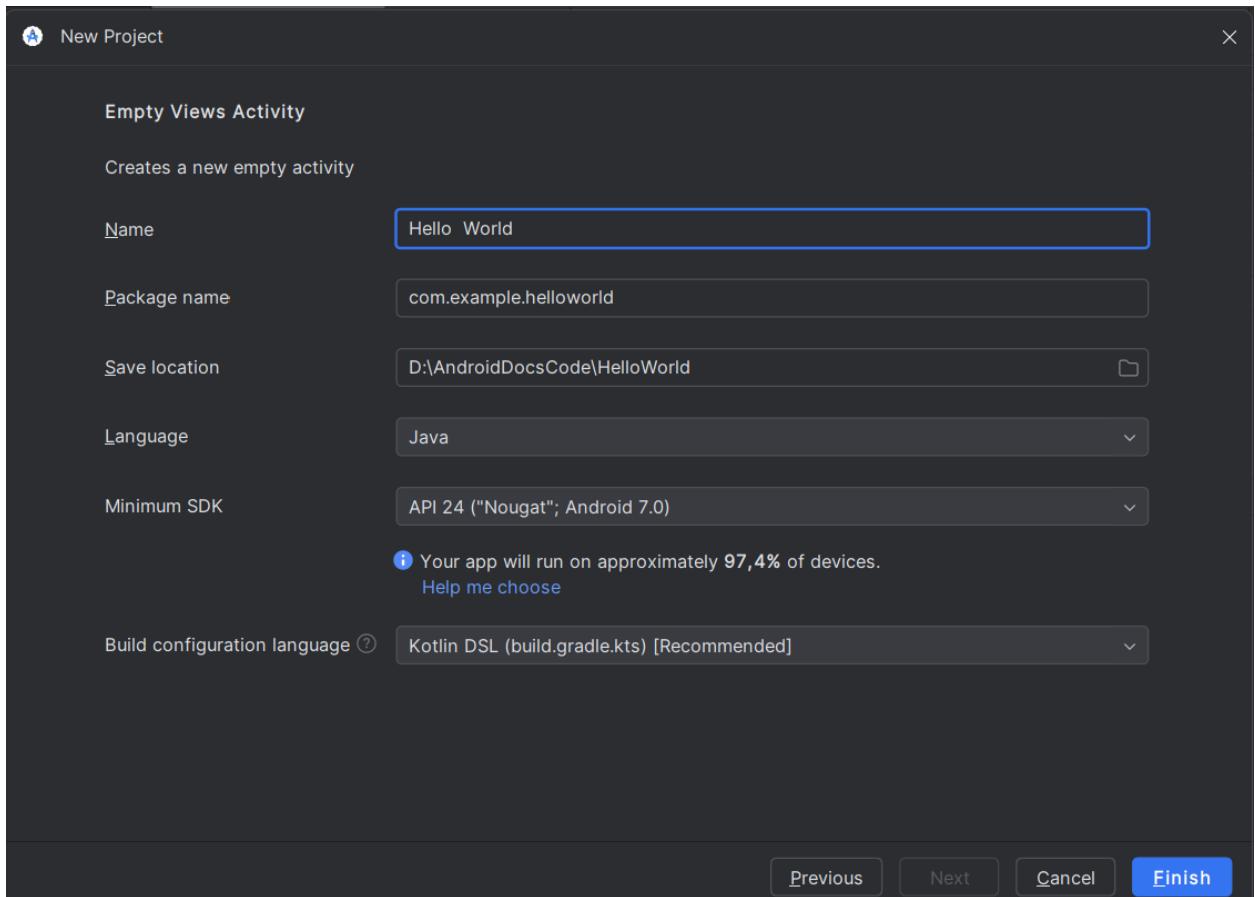
#### **Giới thiệu**

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

## Những gì Bạn nên biết

Bạn nên có khả năng:

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



## Những gì Bạn sẽ cần:

- Một máy tính chạy Windows hoặc Linux, hoặc một Mac chạy macOS. Xem trang tải xuống Android Studio để biết yêu cầu cấu 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.

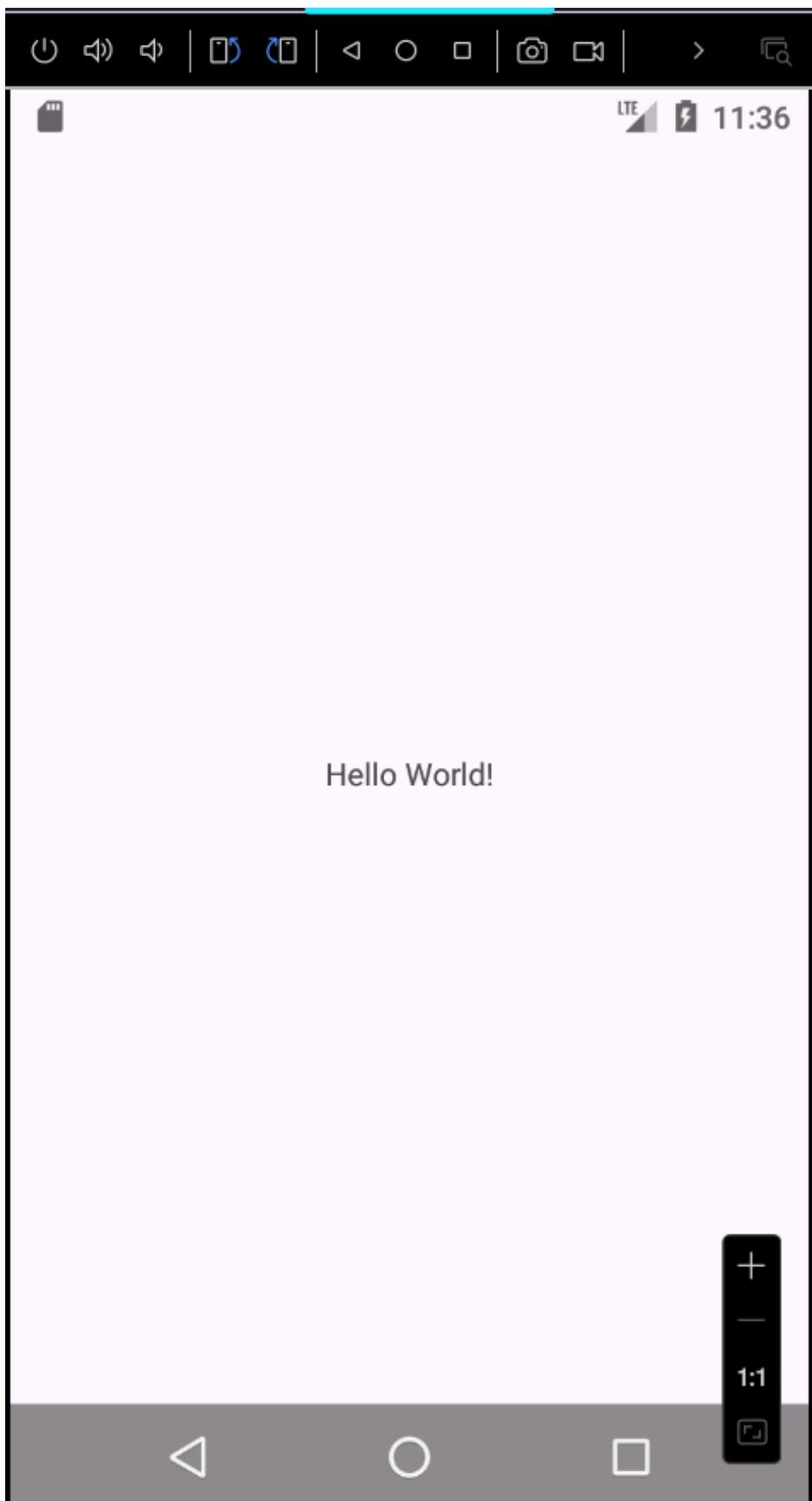
## Những gì bạn sẽ làm

- 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**

## Giới thiệu về ứng dụng

Sau khi cài đặt thành công Android Studio, bạn sẽ tạo một dự án mới từ mẫu có sẵn cho ứng dụng Hello World. Ứng dụng đơn giản này sẽ hiển thị dòng chữ “Hello World” trên màn hình của thiết bị Android (trên máy ảo và máy thật).

Hình ảnh của ứng dụng hoàn chỉnh sẽ trông như sau:



## Bài 1: Cài đặt Android Studio

Android Studio cung cấp một môi trường phát triển thích hợp (IDE) hoàn chỉnh, bao gồm một trình chỉnh sửa mã nâng cao 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 chóng và dễ dàng hơn. Bạn có thể kiểm thử ứng dụng trên nhiều trình giả lập được cấu hình sẵn hoặc trên thiết bị di động của chính mình, xây dựng ứng dụng hoàn chỉnh và xuất bản lên cửa hàng Google Play.

Lưu ý: Android Studio liên tục được cải tiến. Để biết thông tin mới nhất về yêu cầu hệ thống và hướng dẫn cài đặt, hãy xem tài liệu chính thức của Android Studio.

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

Để cài đặt và thiết lập Android Studio, trước tiên hãy kiểm tra yêu cầu hệ thống để đảm bảo thiết bị của bạn đáp ứng đủ điều kiện. Quy trình cài đặt tương tự trên tất cả các hệ điều hành, bất kỳ khác biệt nào sẽ được ghi chú riêng.

Các bước cài đặt:

1. Truy cập trang web chính thức của Android Developers và làm theo hướng dẫn để tải xuống và cài đặt Android Studio.
2. Chấp nhận các thiết lập mặc định trong suốt quá trình cài đặt, đồng thời đảm bảo rằng tất cả các thành phần cần thiết đều được chọn để cài đặt.
3. Sau khi quá trình cài đặt hoàn tất, Setup Wizard sẽ tiếp tục tải xuống và cài đặt một số thành phần bổ sung, bao gồm cả Android SDK. Hãy kiên nhẫn, vì quá trình này có thể mất thời gian tùy thuộc vào tốc độ Internet của bạn. Một số bước có thể trông giống nhau nhưng vẫn cần thiết.
4. Quá trình tải xuống hoàn tất, Android Studio sẽ khởi động và bạn đã sẵn sàng để tạo dự án đầu tiên của mình.

Xử lý sự cố:

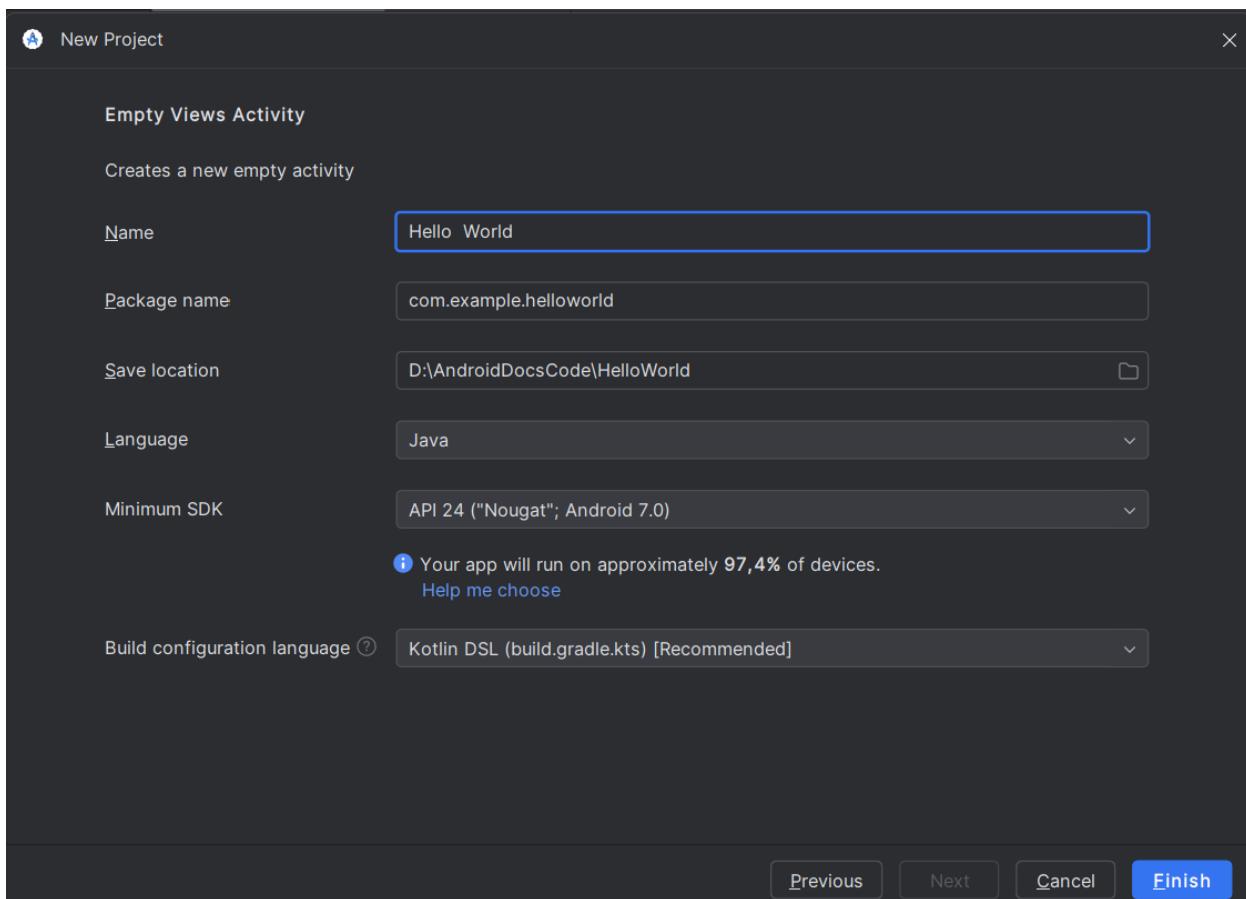
Nếu gặp sự cố khi cài đặt, hãy kiểm tra ghi chú phát hành của Android Studio hoặc tìm sự trợ giúp từ giảng viên của bạn.

## Bài 2: Tạo ứng dụng "Hello World"

Trong bài tập 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 phát triển ứng dụng trong Android Studio.

## 2.1 Tạo dự án ứng dụng

1. Mở **Android Studio** nếu nó chưa được mở.
2. Trong cửa sổ **Welcome to Android Studio**, nhấn 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).

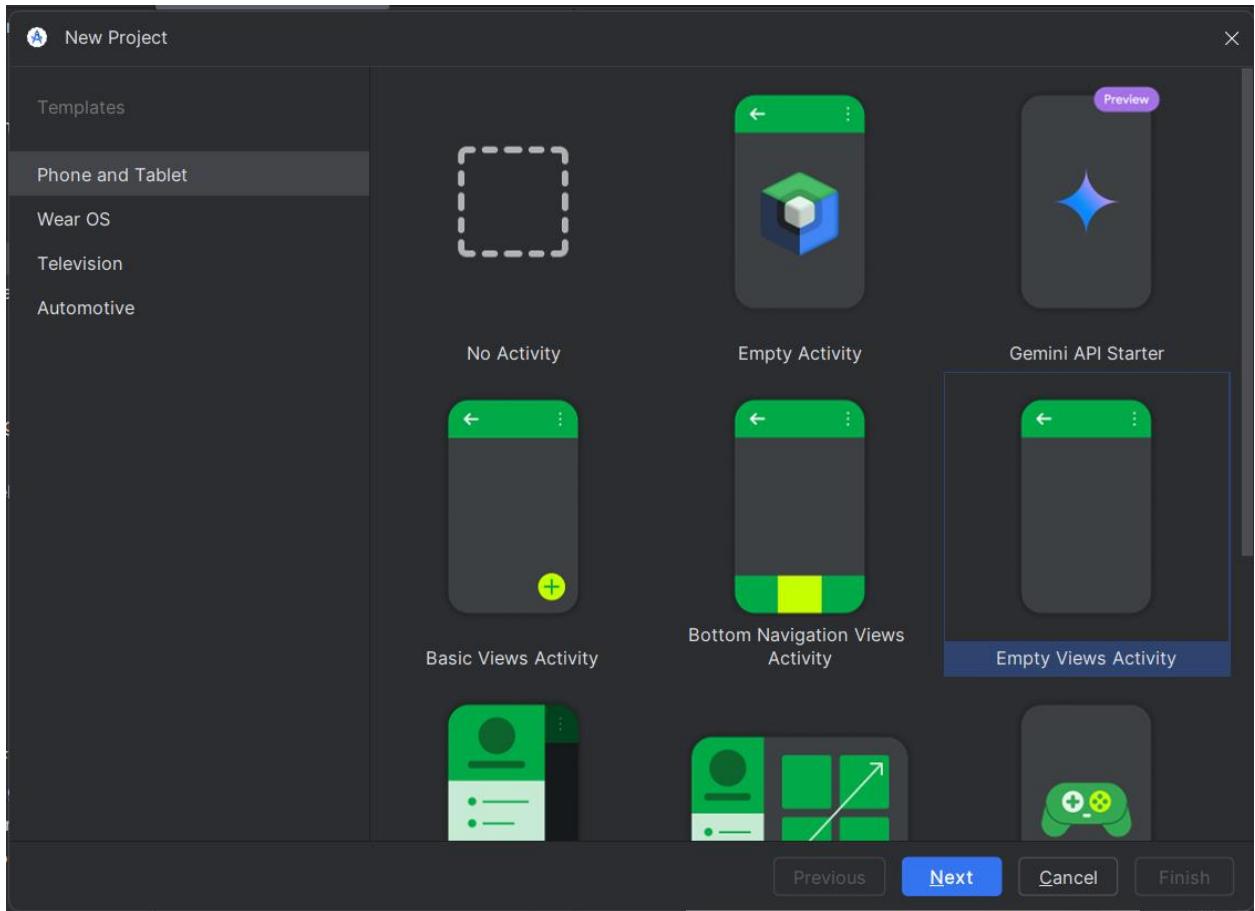


4. Kiểm tra xem Project location (vị trí lưu dự án) mặc định có đúng nơi bạn muốn lưu ứng dụng Hello World và các dự án Android Studio khác không. Nếu cần, hãy thay đổi sang thư mục bạn muốn.

5. Giữ nguyên android.example.com làm Company Domain (tên miền công ty) hoặc tạo một tên miền riêng.

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

6. Bỏ chọn các tùy chọn Include C++ support (Hỗ trợ C++) và Include Kotlin support (Hỗ trợ Kotlin), sau đó nhấn Next.
7. Ở màn hình Target Android Devices, đảm bảo rằng Phone and Tablet (Điện thoại và Máy tính bảng) được chọn. Kiểm tra xem API 15: Android 4.0.3 IceCreamSandwich có được đặt làm Minimum SDK (SDK tối thiểu) không; nếu không, hãy sử dụng menu thả xuống để đặt lại. Các bài học trong khóa học này sử dụng các cài đặt này, giúp ứng dụng Hello World tương thích với 97% thiết bị Android đang hoạt động trên Google Play Store.
8. Bỏ chọn Include Instant App support (Hỗ trợ Instant App) và tất cả các tùy chọn khác, sau đó nhấn Next. Nếu dự án của bạn yêu cầu cài đặt thêm thành phần cho SDK mục tiêu, Android Studio sẽ tự động cài đặt chúng.
9. Cửa sổ Add an Activity xuất hiện. Activity là một màn hình giao diện trong ứng dụng, nơi người dùng có thể thực hiện một hành động cụ thể. Mỗi Activity thường có một layout (bố cục giao diện) xác định cách hiển thị các thành phần UI trên màn hình. Android Studio cung cấp các mẫu Activity để giúp bạn bắt đầu nhanh hơn. Đối với dự án Hello World, hãy chọn Empty Activity (Hoạt động trống), sau đó nhấn Next.



10. Màn hình Configure Activity xuất hiện (có thể khác nhau tùy vào mẫu bạn chọn ở bước trước). Mặc định, Activity được đặt tên là MainActivity. Bạn có thể đổi nếu muốn, nhưng bài học này sử dụng MainActivity.
11. Đảm bảo rằng tùy chọn Generate Layout file (Tạo tệp bố cục) được chọn. Tên bố cục mặc định là activity\_main. Bạn có thể đổi nếu muốn, nhưng bài học này sử dụng activity\_main.
12. Đảm bảo rằng tùy chọn Backwards Compatibility (App Compat) (Hỗ trợ tương thích ngược) được chọn. Điều này giúp ứng dụng tương thích với các phiên bản Android cũ hơn.
13. Nhấn Finish để hoàn tất quá trình tạo dự án.

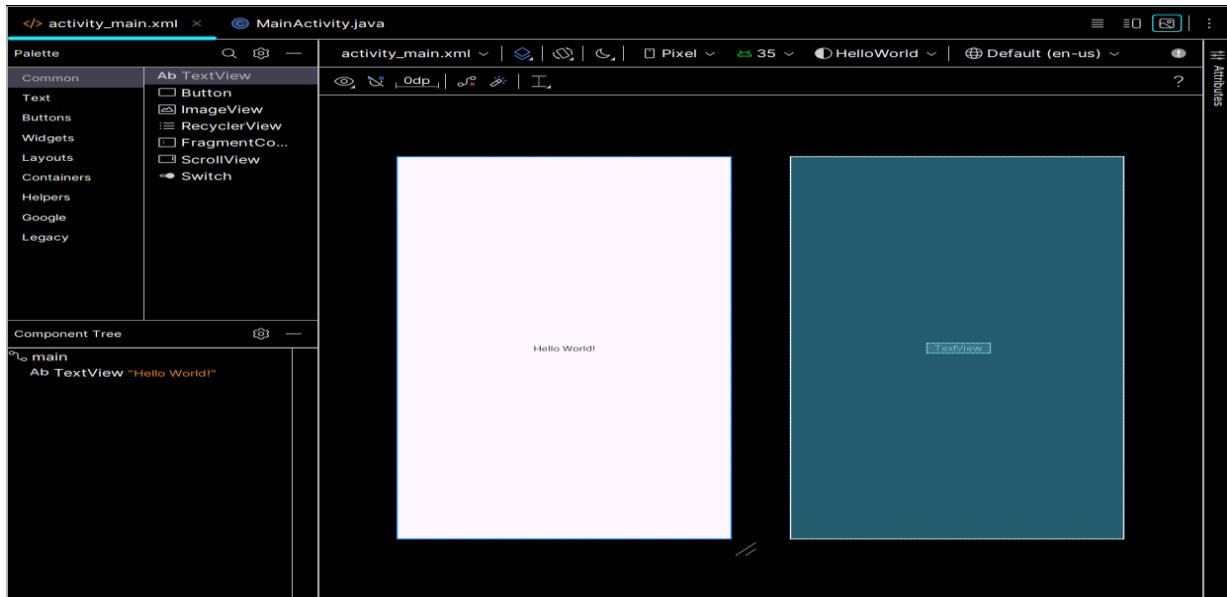
Android Studio tạo một thư mục cho các dự án của bạn và xây dựng dự án bằng Gradle (quá trình này có thể mất một vài phút).

**Mẹo:** Xem trang dành cho nhà phát triển **Cấu hình bản dựng (Configure your build)** để biết thông tin chi tiết.

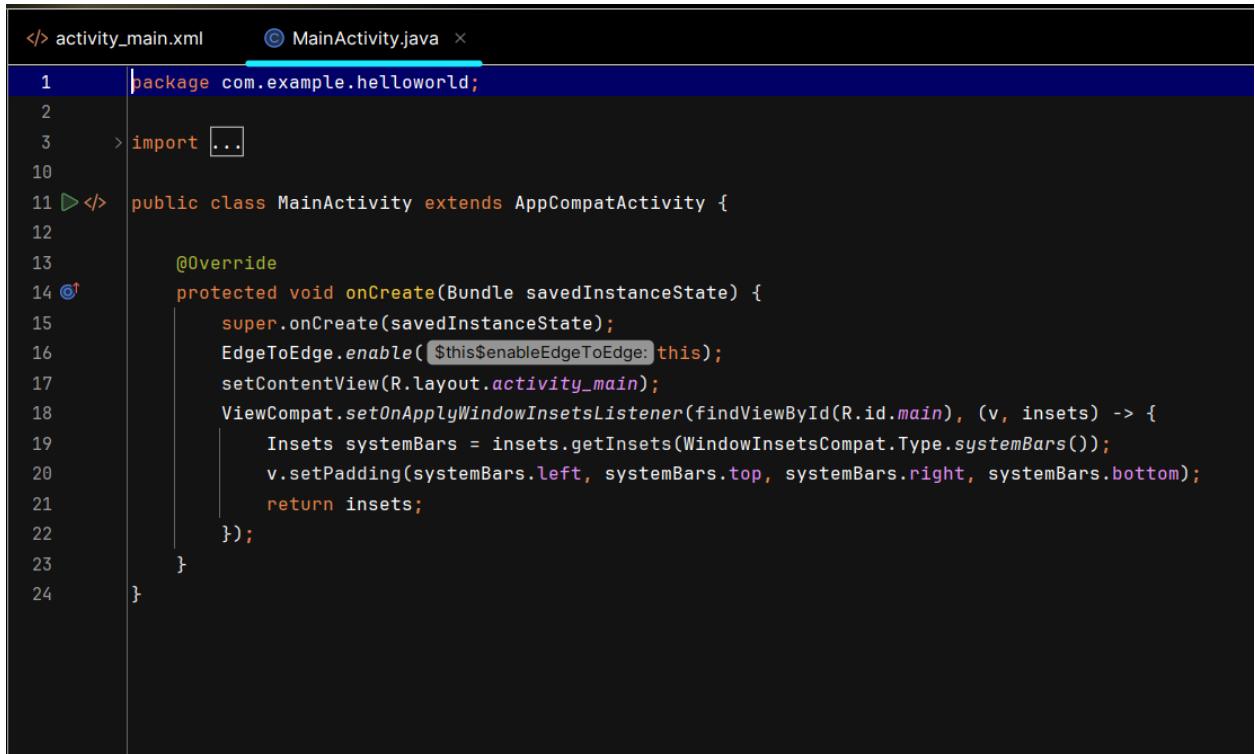
Bạn cũng có thể thấy một thông báo "Mẹo trong ngày" (Tip of the day) với các phím tắt và mẹo hữu ích khác. Nhấn **Close** để đóng thông báo.

Sau đó, trình soạn thảo của Android Studio sẽ xuất hiện. Hãy làm theo các bước sau:

1. Nhấp vào tab **activity\_main.xml** để xem trình chỉnh sửa bố cục (layout editor).
2. Nhấp vào tab **Design** trong trình chỉnh sửa bố cục (nếu chưa được chọn) để hiển thị bản xem trước đồ họa của bố cục như hình bên dưới.



3. Nhấp vào tab **MainActivity.java** để xem trình chỉnh sửa mã nguồn như hình bên dưới.

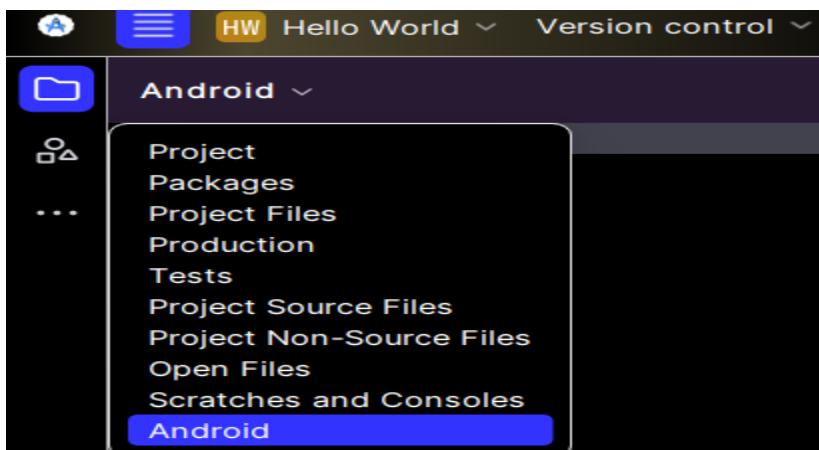


```
</> activity_main.xml      MainActivity.java x
1 package com.example.helloworld;
2
3 > import ...
10
11 </> public class MainActivity extends AppCompatActivity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         EdgeToEdge.enable($this$enableEdgeToEdge: this);
17         setContentView(R.layout.activity_main);
18         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
19             Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
20             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
21             return insets;
22         });
23     }
24 }
```

## 2.2 Khám phá ngăn Project > Android

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

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

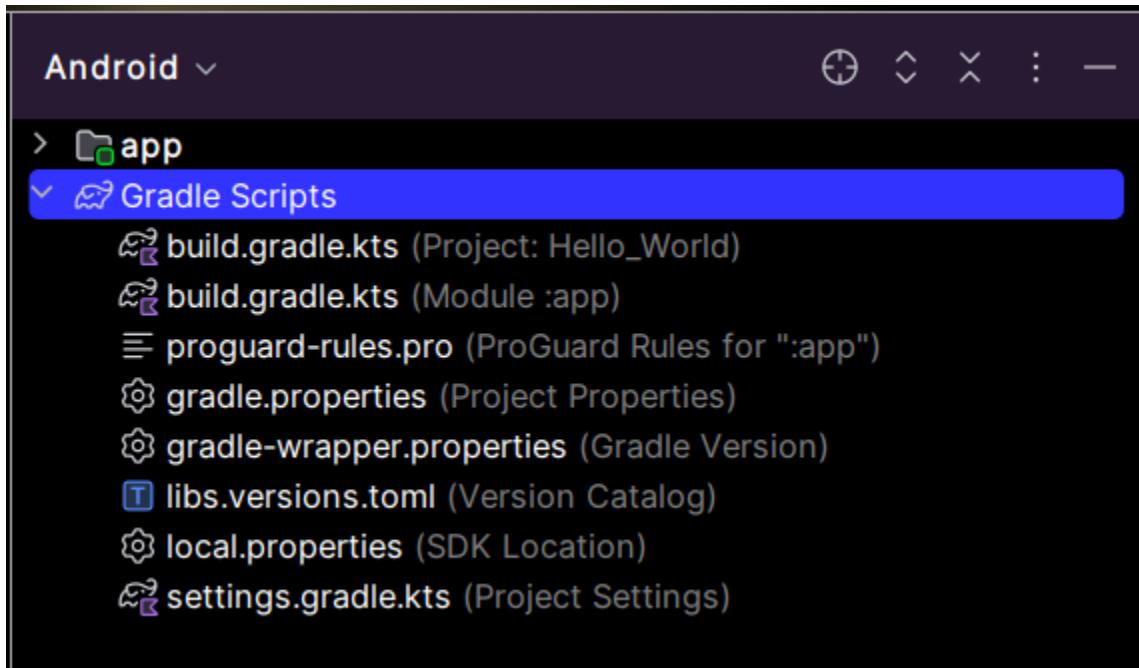


**Lưu ý:** Chương này và các chương khác sẽ gọi ngăn **Project** khi được đặt ở chế độ **Android** là **Project > Android**.

## 2.3 Khám phá thư mục Gradle Scripts

Hệ thống xây dựng Gradle trong Android Studio giúp bạn dễ dàng bao gồm các tệp nhị phân bên ngoài hoặc các mô-đun thư viện khác vào bản dựng của bạn dưới dạng dependencies.

Khi bạn lần đầu tiên tạo một dự án ứng dụng, ngăn **Project > Android** xuất hiện với thư mục **Gradle Scripts** được mở rộng như hình dưới đây.



Hãy làm theo các bước sau để khám phá hệ thống Gradle:

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

Đây là nơi bạn sẽ tìm thấy các tùy chọn cấu hình chung cho tất cả các mô-đun tạo nên dự án của bạn.

Mỗi dự án trong Android Studio chứa một tệp Gradle build cấp cao nhất duy nhất. Hầu hết thời gian, bạn sẽ không cần thực hiện bất kỳ thay đổi nào đối với tệp này, nhưng việc hiểu nội dung của nó vẫn 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 dependency của bạn không phải là một thư viện cục bộ hoặc một thư mục tệp, Gradle sẽ tìm kiếm các tệp trong bất kỳ kho lưu trữ trực tuyến nào được chỉ định trong khối **repositories** của tệp này. Theo mặc định, các dự án Android Studio mới khai báo **JCenter** và **Google** (bao gồm **Google Maven repository**) làm vị trí kho lưu trữ.

```
1 // Top-level build file where you can add configuration options common to all sub-projects/modules.
2 plugins {
3     alias(libs.plugins.android.application) apply false
4 }
```

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

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

Đây là tệp thường xuyên được chỉnh sửa nhất khi thay đổi các cấu hình cấp ứng dụng, chẳng hạn như khai báo dependencies trong phần **dependencies**. Bạn có thể khai báo một thư viện phụ thuộc bằng một trong số các cấu hình dependency khác nhau. Mỗi cấu hình dependency cung cấp cho Gradle các hướng dẫn khác nhau về cách sử dụng thư viện. Ví dụ, câu lệnh: implementation fileTree(dir: 'libs', include: ['\*.jar'])

thêm tất cả các tệp “.jar” bên trong thư mục **libs** làm dependency.

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

```

1  plugins {
2      alias(libs.plugins.android.application)
3  }
4
5  android {
6      namespace = "com.example.helloworld"
7      compileSdk = 35
8
9      defaultConfig {
10         applicationId = "com.example.helloworld"
11         minSdk = 24
12         targetSdk = 35
13         versionCode = 1
14         versionName = "1.0"
15
16         testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
17     }
18
19     buildTypes {
20         release {
21             isMinifyEnabled = false
22             proguardFiles(
23                 getDefaultProguardFile("proguard-android-optimize.txt"),
24                 "proguard-rules.pro"
25             )
26         }
27     }
28     compileOptions {
29         sourceCompatibility = JavaVersion.VERSION_11
30         targetCompatibility = JavaVersion.VERSION_11
31     }
32 }

```

```

33
34  dependencies {
35
36      implementation(libs.appcompat)
37      implementation(libs.material)
38      implementation(libs.activity)
39      implementation(libs.constraintlayout)
40      testImplementation(libs.junit)
41      androidTestImplementation(libs.ext.junit)
42      androidTestImplementation(libs.espresso.core)
43  }

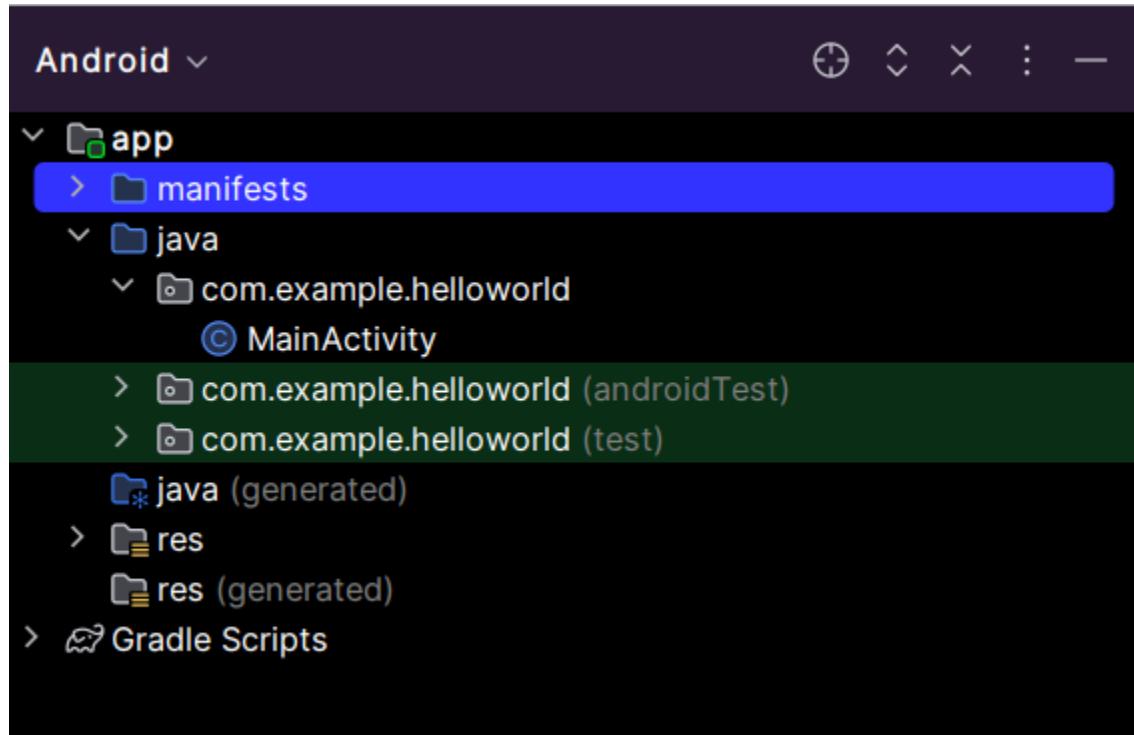
```

#### 4. Nhấp vào tam giác để đóng **Gradle Scripts**.

#### 2.4 Khám phá thư mục app và res

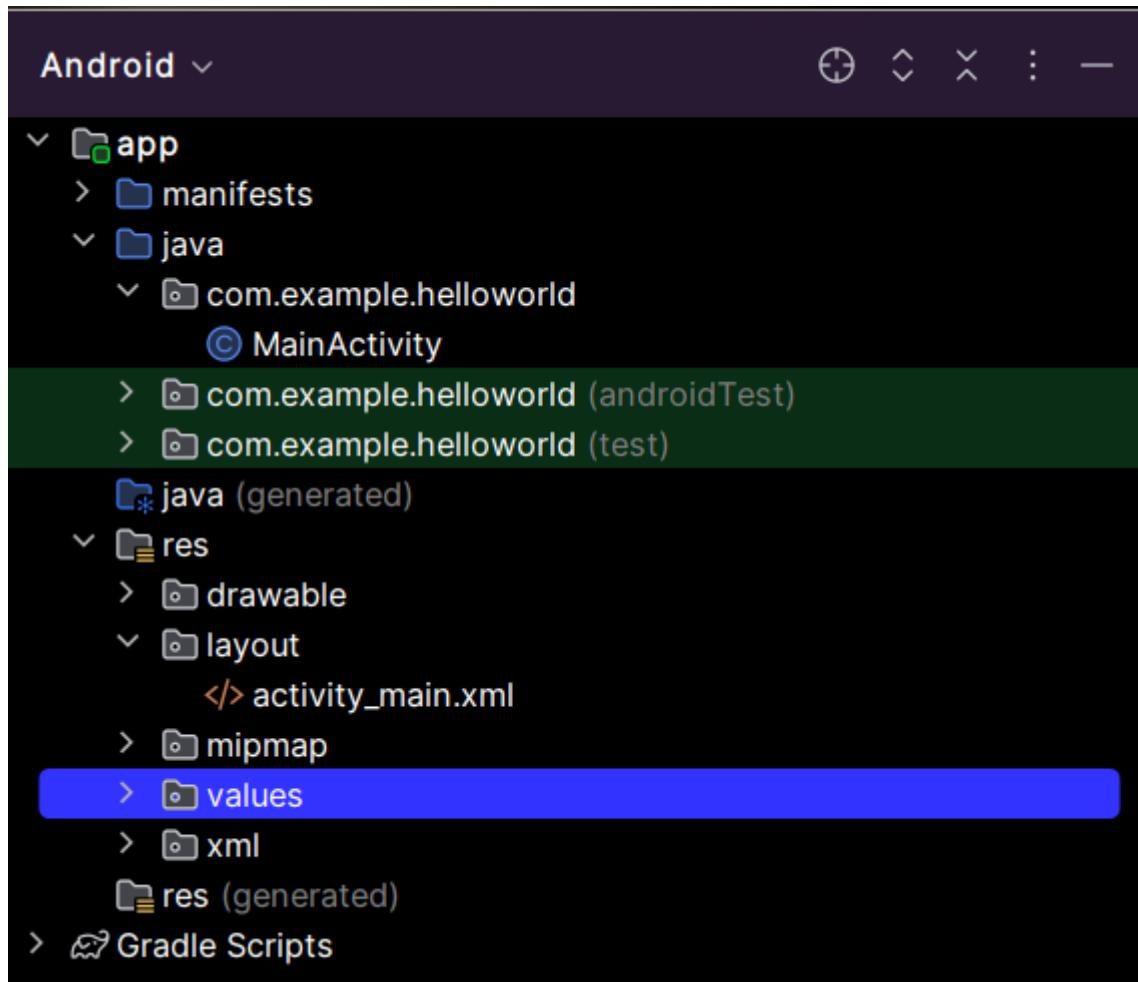
Tất cả mã nguồn và tài nguyên của ứng dụng đều nằm trong các thư mục **app** và **res**.

1. Mở rộng thư mục **app**, sau đó mở rộng thư mục **java** và thư mục **com.example.android.helloworld** để xem tệp **MainActivity.java**.
  - o Nhấp đúp vào tệp này để mở nó trong trình chỉnh sửa mã nguồn.



- o Thư mục **java** chứa các tệp lớp Java trong ba thư mục con, như hình minh họa bên trên.
- o Thư mục **com.example.hello.helloworld** (hoặc tên miền bạn đã chỉ định) chứa tất cả các tệp của một gói ứng dụng.
- o Hai thư mục còn lại được sử dụng để kiểm thử và sẽ được mô tả trong một bài học khác.
- o Đối với ứng dụng Hello World, chỉ có một gói và nó chứa tệp **MainActivity.java**.
- o Tên của **Activity** (màn hình) đầu tiên mà người dùng nhìn thấy, cũng là nơi khởi tạo các tài nguyên toàn cục của ứng dụng, thường được đặt là **MainActivity** (trong ngăn **Project > Android**, phần mở rộng tệp bị ẩn đi).

2. Mở rộng thư mục **res**, sau đó mở rộng 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 văn bản (strings) và hình ảnh (images).
- Một **Activity** thường được liên kết với một bố cục giao diện người dùng (UI views) được định nghĩa trong một tệp XML.
- Tệp này thường được đặt tên theo tên **Activity** của nó.

## 2.5 Khám phá thư mục **manifests**

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

- Mở rộng thư mục **manifests**.
- Mở tệp **AndroidManifest.xml**.

Tệp **AndroidManifest.xml** mô tả tất cả các thành phần của ứng dụng Android của bạn.

- Mọi thành phần của ứng dụng, chẳng hạn như mỗi **Activity**, phải được khai báo trong tệp XML này.
- Trong các bài học khác của khóa học, bạn sẽ chỉnh sửa tệp này để thêm các tính năng và quyền (permissions) cho ứng dụng.
- Để tìm hiểu tổng quan, hãy xem **App Manifest Overview**.

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

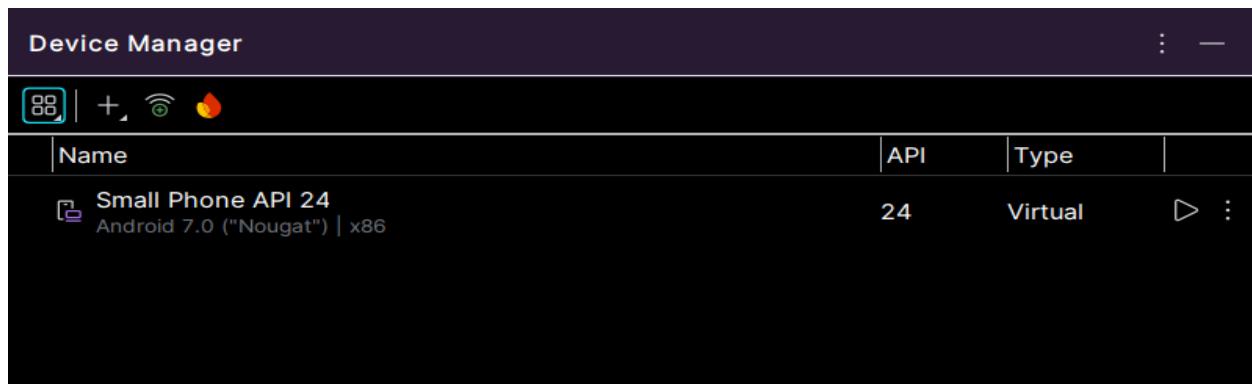
Trong nhiệm vụ này, bạn sẽ sử dụng **Android Virtual Device (AVD) Manager** để tạo một thiết bị ảo (còn gọi là trình giả lập) mô phỏng cấu hình của một loại thiết bị Android cụ thể và sử dụng thiết bị ảo đó để chạy ứng dụng. Lưu ý rằng **Android Emulator** có các yêu cầu bổ sung ngoài các yêu cầu hệ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

Sử dụng **AVD Manager**, bạn có thể xác định các đặc điểm phần cứng của thiết bị, cấp độ API, bộ nhớ, giao diện và các thuộc tính khác, sau đó lưu nó dưới dạng một thiết bị ảo. Với các thiết bị ảo, bạn có thể kiểm thử ứng dụng trên nhiều cấu hình thiết bị khác nhau (chẳng hạn như máy tính bảng và điện thoại) với các cấp độ API khác nhau mà không cần sử dụng thiết bị vật lý.

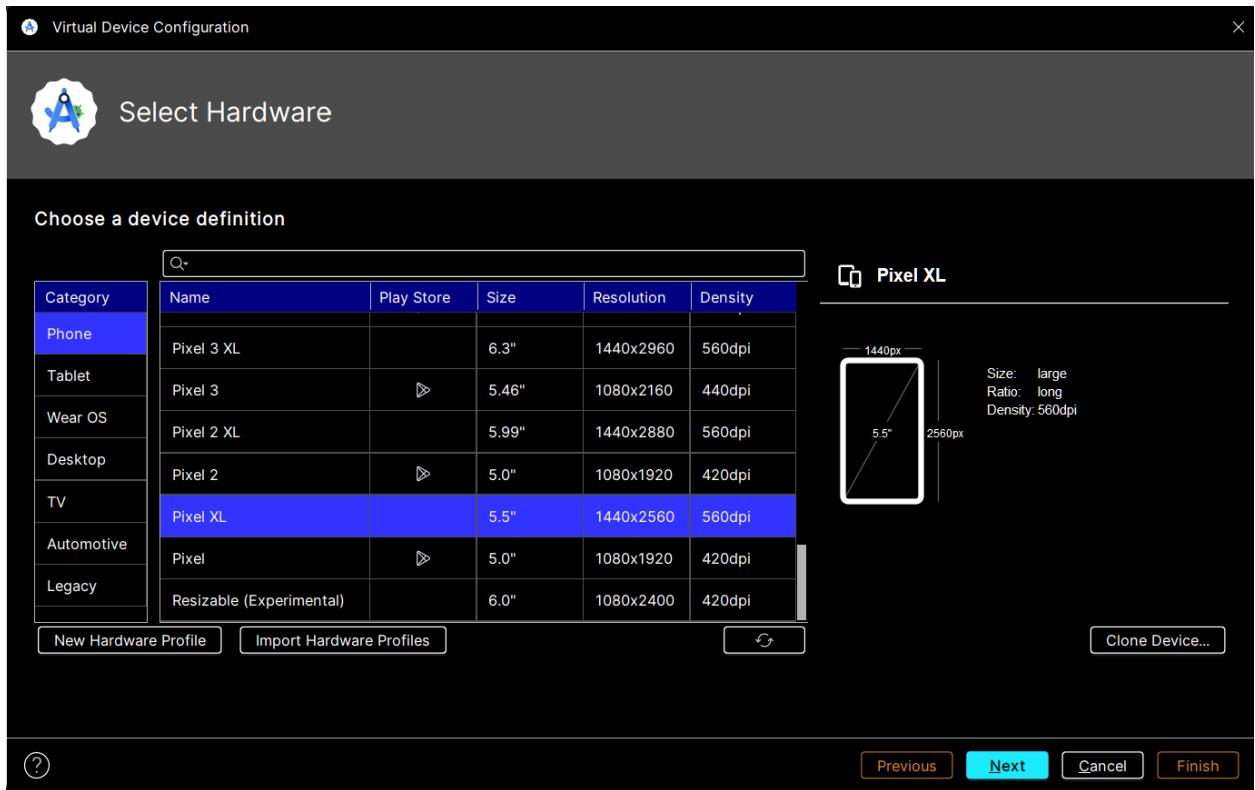
#### 3.1 Tạo một thiết bị ảo Android (AVD)

Để chạy trình giả lập trên máy tính của bạn, trước tiên bạn phải 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.



2. Nhấp vào + Create Virtual Device. Cửa sổ **Select Hardware** xuất hiện, hiển thị danh sách các thiết bị phần cứng được cấu hình sẵn. Mỗi thiết bị có các thông số như kích thước màn hình theo đường chéo (**Size**), độ phân giải màn hình tính theo pixel (**Resolution**), và mật độ điểm ảnh (**Density**).



3. Chọn một thiết bị, chẳng hạn như **Nexus 5X** hoặc **Pixel XL**, rồi nhấp vào **Next**. Màn hình **System Image** xuất hiện.
4. Nhấp vào tab **Recommended** (nếu chưa được chọn) và chọn phiên bản hệ điều hành Android mà bạn muốn chạy trên thiết bị ảo (chẳng hạn như **Oreo**).



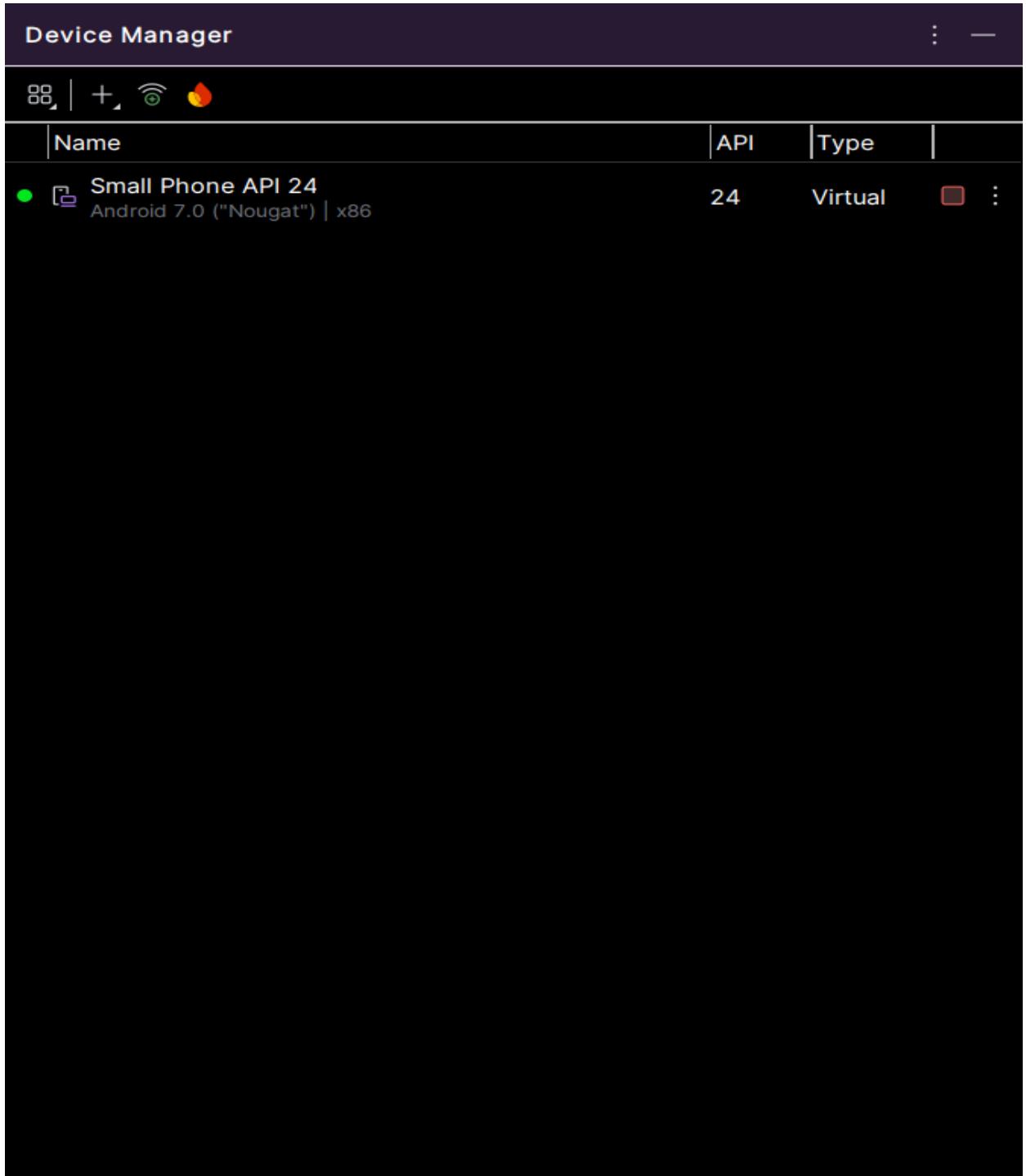
Có nhiều phiên bản hơn so với những gì được hiển thị trong tab **Recommended**. Bạn có thể xem thêm trong các tab **x86 Images** và **Other Images**. Nếu có liên kết **Download** bên cạnh hình ảnh hệ thống mà bạn muốn sử dụng, điều đó có nghĩa là nó chưa được cài đặt. Nhấp vào liên kết để bắt đầu tải xuống, sau đó nhấp **Finish** khi hoàn tất.

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

### 3.2 Chạy ứng dụng trên thiết bị ảo

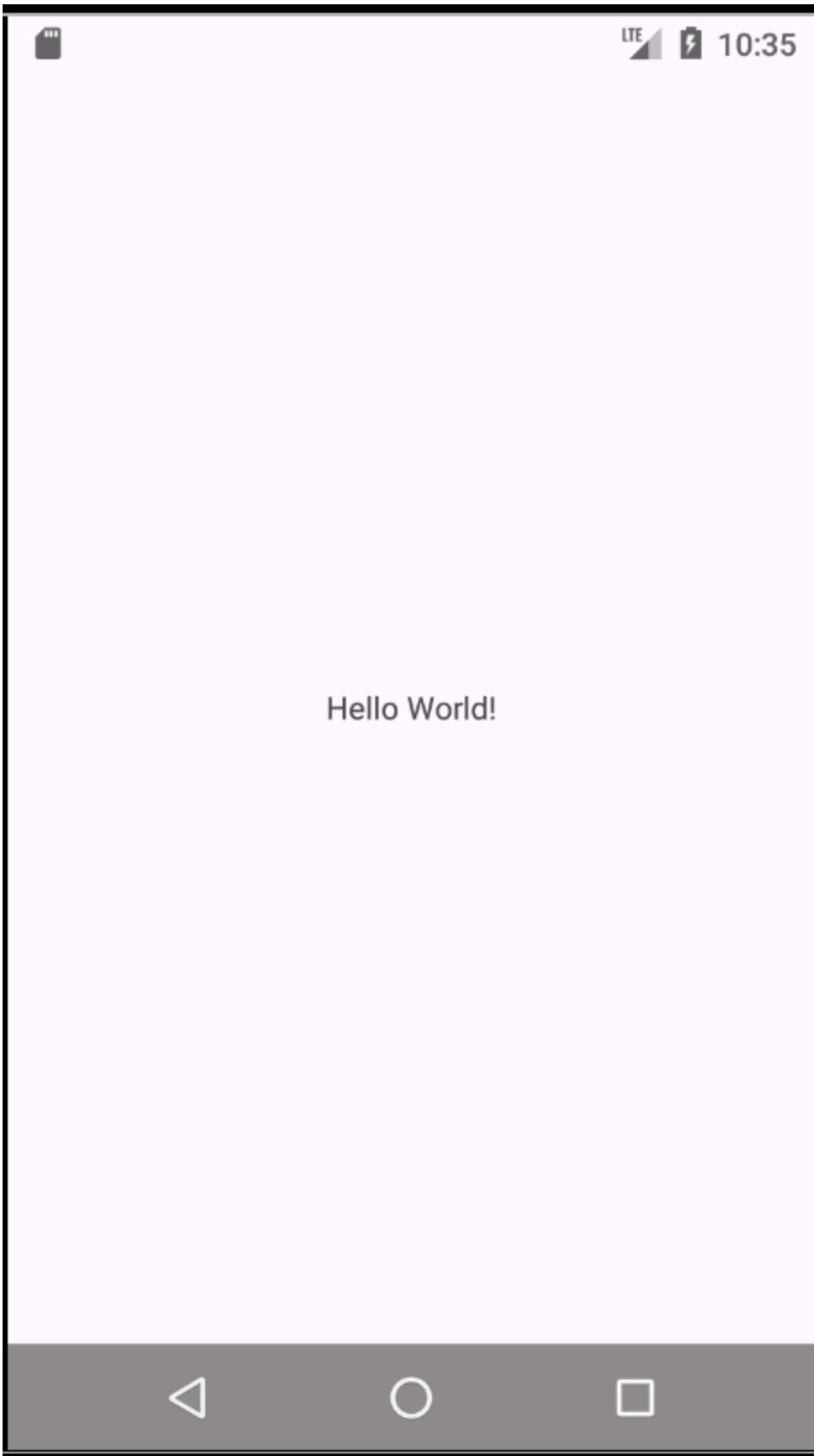
Trong nhiệm vụ này, bạn sẽ chạy ứng dụng **Hello World** của mình.

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



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

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



**Mẹo:** Khi kiểm thử trên thiết bị ảo, tốt nhất bạn nên khởi động nó một lần ngay từ đầu phiên làm việc và không đóng nó cho đến khi bạn hoàn thành việc kiểm thử. Điều này giúp tránh việc ứng dụng phải trải qua quá trình khởi động thiết bị nhiều lần.

Để đóng thiết bị ảo, bạn có thể:

- Nhấp vào nút **X** ở góc trên của trình giả lập.
- Chọn **Quit** từ menu.
- Nhấn **Control + Q** trên Windows hoặc **Command + Q** trên macOS.

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

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

### a) Bạn cần có:

- Một thiết bị Android, chẳng hạn như điện thoại hoặc máy tính bảng.
- Cáp dữ liệu để kết nối thiết bị Android với máy tính qua cổng USB.
- Nếu bạn đang sử dụng hệ thống **Linux** hoặc **Windows**, có thể cần thực hiện thêm một số bước để chạy ứng dụng trên thiết bị phần cứng.
  - Kiểm tra tài liệu **Using Hardware Devices**.
  - Bạn cũng có thể cần cài đặt **trình điều khiển USB** phù hợp với thiết bị của mình.
  - Đối với trình điều khiển USB trên Windows, hãy xem **OEM USB Drivers**.

---

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

Để **Android Studio** có thể giao tiếp với thiết bị của bạn, bạn phải bật **Gỡ lỗi USB** trên thiết bị Android. Tùy chọn này nằm trong phần **Developer options** của thiết bị.

Trên **Android 4.2 trở lên**, màn hình **Developer options** bị ẩn theo mặc định. Để hiển thị **Developer options** và bật **USB Debugging**, thực hiện như sau:

1. Trên thiết bị, mở **Cài đặt (Settings)**, tìm **Giới thiệu về điện thoại (About phone)**, nhấn vào đó và chạm vào **Số bản dựng (Build number) 7 lần liên tiếp**.
2. Quay lại màn hình trước đó (**Cài đặt / Hệ thống (Settings / System)**). Mục **Developer options** sẽ xuất hiện trong danh sách. Nhấn vào **Developer options**.

- 
3. Bật tùy chọn **USB Debugging**.

---

## 4.2 Chạy ứng dụng trên thiết bị

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

1. **Kết nối** thiết bị với máy tính bằng cáp USB.
2. Nhấp vào **Run** trên thanh công cụ.
  - o Cửa sổ **Select Deployment Target** sẽ mở ra, hiển thị danh sách trình giả lập và các thiết bị được kết nối.
3. **Chọn thiết bị** của bạn và nhấp **OK**.
  - o **Android Studio** sẽ cài đặt và chạy ứng dụng trên thiết bị của bạn.

---

## Xử lý sự cố (Troubleshooting)

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

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

Nếu máy tính vẫn không tìm thấy thiết bị hoặc hiển thị trạng thái "**unauthorized**", hãy làm như sau:

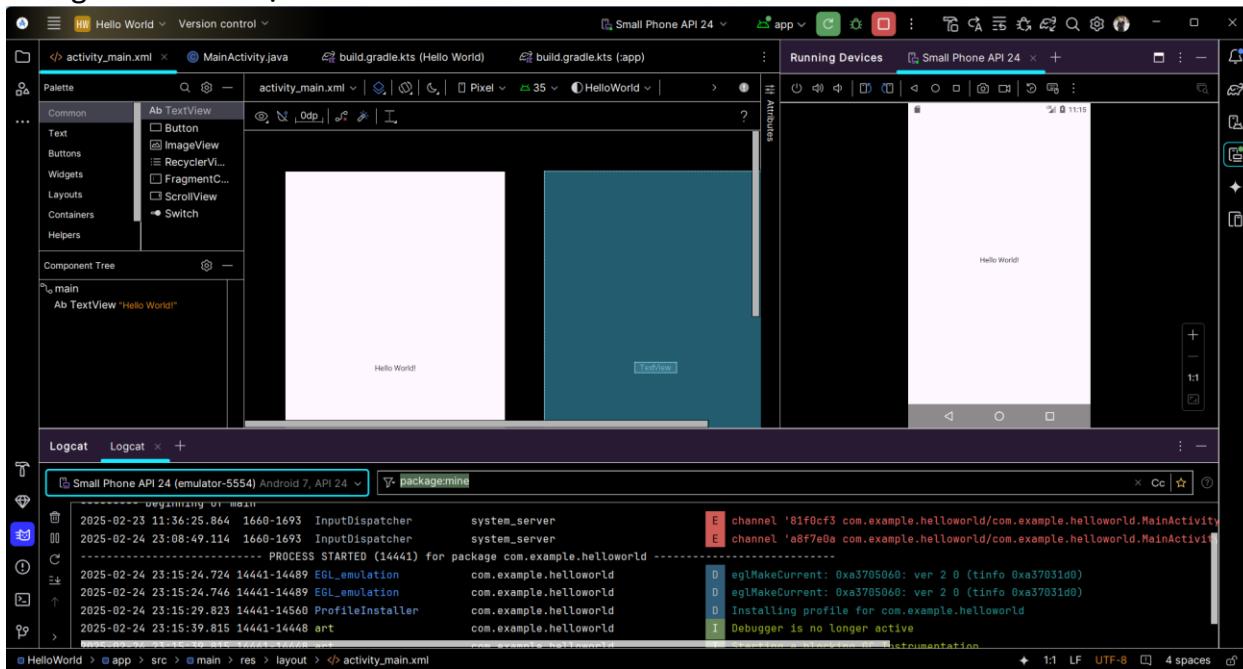
1. Rút kết nối thiết bị.
2. Trên thiết bị, mở **Developer Options** trong ứng dụng **Cài đặt (Settings)**.
3. Nhấn vào **Revoke USB Debugging authorizations**.
4. Kết nối lại thiết bị với máy tính.
5. Khi có thông báo yêu cầu cấp quyền, hãy chọn **Cho phép (Allow)**.

Bạn có thể cần cài đặt **trình điều khiển USB** phù hợp với thiết bị của mình. Xem tài liệu **Using Hardware Devices** để biết thêm chi tiết.

## 6.1 Xem Logcat pane

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

Trong hình minh họa:



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

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

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

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

### Các thành phần của thông báo:

- **Log:** Lớp **Log** dùng để gửi thông báo log đến **Logcat pane**.
- **d:** Cấp độ **Debug** của **Log**, dùng để lọc thông báo trong **Logcat pane**.
  - o Các cấp độ log khác:
    - **e:** **Error** (Lỗi)

- **w: Warn** (Cảnh báo)
- **i: Info** (Thông tin)
- "**MainActivity**": Đối số đầu tiên là **tag**, giúp lọc thông báo trong **Logcat pane**.
  - Thông thường, đây là tên của **Activity** chứa thông báo.
  - Tuy nhiên, bạn có thể đặt bất kỳ giá trị nào có ích cho việc gỡ lỗi.
  - Theo quy ước, **log tag** thường được định nghĩa là **hằng số** trong **Activity**.

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

- "**Hello world**": Đối số thứ hai là nội dung của thông báo.
- 

### Các bước thực hiện:

1. Mở ứng dụng **Hello World** trong **Android Studio** và mở tệp **MainActivity**.
2. Để tự động thêm các **import** cần thiết vào dự án (chẳng hạn như **android.util.Log** để sử dụng **Log**), thực hiện:
  - **Windows**: Chọn **File > Settings**
  - **macOS**: Chọn **Android Studio > Preferences**
3. Chọn **Editor > General > Auto Import**.
  - **Tích chọn tất cả các hộp kiểm** và đặt **Insert imports on paste** thành **All**.
4. Nhấp **Apply**, sau đó nhấp **OK**.
5. Trong phương thức **onCreate()** của **MainActivity**, thêm dòng lệnh sau:

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

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

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this );
    setContentView(R.layout.activity_main);
    Log.d( tag: "MainActivity", msg: "Hello World" );
    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;
    });
}
```

6. Nếu **Logcat pane** chưa mở, nhấp vào tab **Logcat** ở cuối cửa sổ **Android Studio** để mở.
  7. Kiểm tra xem **tên ứng dụng** và **tên gói (package name)** có chính xác không.
  8. Thay đổi **Log level** trong **Logcat pane** thành **Debug** (hoặc giữ nguyên **Verbose** nếu có ít thông báo log).
  9. Chạy ứng dụng của bạn.
- 

Sau khi chạy, bạn sẽ thấy thông báo xuất hiện trong **Logcat pane**.

```
2025-02-25 00:00:04.497 15139-15139 MainActivity com.example.helloworld D Hello World
```

## Thử thách lập trình

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

### Thử thách:

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

1. Tạo một dự án mới trong Android Studio.
  2. Thay đổi dòng chữ "Hello World" thành "Happy Birthday to " kèm theo tên của một người vừa có sinh nhật gần đây.
  3. (**Tùy chọn**) Chụp ảnh màn hình ứng dụng hoàn thành và gửi email cho ai đó mà bạn đã quên chúc mừng sinh nhật.
  4. Một cách sử dụng phổ biến của lớp Log là ghi lại các ngoại lệ (**exceptions**) trong chương trình Java khi chúng xảy ra. Có một số phương thức hữu ích như Log.e() mà bạn có thể dùng cho mục đích này. Hãy khám phá các phương thức cho phép bạn bao gồm một ngoại lệ trong một thông báo Log. Sau đó, viết mã trong ứng dụng của bạn để tạo ra và ghi lại một ngoại lệ.
-

## Tóm tắt

- Để cài đặt Android Studio, truy cập [Android Studio](#) và làm theo hướng dẫn để tải xuống và cài đặt.
- Khi tạo một ứng dụng mới, 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** ở cột dọc và chọn **Android** trong menu thả xuống ở trên cùng.
- Chỉnh sửa tệp **build.gradle(Module:app)** khi bạn cần thêm thư viện mới hoặc thay đổi phiên bản thư viện trong dự án.
- Tất cả mã nguồn và tài nguyên của ứng dụng đều nằm trong thư mục **app** và **res**. Thư mục **java** chứa các activity, test, và các thành phần khác trong mã nguồn Java. Thư mục **res** chứa các tài nguyên như layout, chuỗi ký tự (**strings**) và hình ảnh (**images**).
- Chỉnh sửa tệp **AndroidManifest.xml** để thêm các thành phần (**components**) và quyền truy cập (**permissions**) cho ứng dụng Android của bạn. Mọi thành phần của ứng dụng, chẳng hạn như nhiều **activity**, phải được khai báo trong tệp XML này.
- Sử dụng **Android Virtual Device (AVD) manager** để tạo một thiết bị ảo (**emulator**) để chạy ứng dụng.
- Thêm các câu lệnh **Log** vào ứng dụng để hiển thị thông báo trong **Logcat pane**, giúp bạn gỡ lỗi (**debugging**) dễ dàng hơn.
- Để chạy ứng dụng trên thiết bị Android thực tế bằng Android Studio, bật **USB Debugging** trên thiết bị.
  - Mở **Settings > About phone** và nhấn vào **Build number** bảy lần.
  - Quay lại màn hình trước (**Settings**) và chọn **Developer options**.
  - Bật **USB Debugging**.

---

## Các khái niệm liên quan

Tài liệu về các khái niệm liên quan có trong:

- **1.0: Giới thiệu về Android**
  - **1.1: Ứng dụng Android đầu tiên của bạn**
-

## Tìm hiểu thêm

Tài liệu về Android Studio:

- [Trang tải xuống Android Studio](#)
- [Ghi chú phát hành Android Studio](#)
- [Giới thiệu về Android Studio](#)
- [Công cụ dòng lệnh Logcat](#)
- [Trình quản lý thiết bị ảo Android \(AVD Manager\)](#)
- [Tổng quan về Android Manifest](#)
- [Cấu hình Gradle](#)
- [Lớp Log trong Android](#)
- [Tạo và quản lý thiết bị ảo](#)

Khác:

- Làm thế nào để cài đặt Java?
- [Cài đặt phần mềm JDK và thiết lập biến JAVA\\_HOME](#)
- [Trang chủ Gradle](#)
- Cú pháp Apache Groovy
- [Trang Wikipedia về Gradle](#)

---

## Bài tập về nhà

Xây dựng và chạy một ứng dụng

1. Tạo một dự án Android mới từ **Empty Template**.
2. Thêm các câu lệnh Log cho nhiều cấp độ nhật ký khác nhau (**log levels**) trong phương thức `onCreate()` của activity chính.
3. Tạo một trình giả lập (**emulator**) cho một thiết bị, chọn phiên bản Android tùy ý, và chạy ứng dụng trên đó.
4. Sử dụng bộ lọc trong **Logcat** để tìm các câu lệnh Log của bạn và điều chỉnh mức hiển thị chỉ để hiển thị **debug** hoặc **error**.

---

## Trả lời các câu hỏi

Câu hỏi 1

Tên của tệp layout cho activity chính là gì?

- MainActivity.java
- AndroidManifest.xml
- activity\_main.xml
- build.gradle

## Câu hỏi 2

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

- app\_name
- xmlns:app
- android:name
- applicationId

## Câu hỏi 3

Công cụ nào được sử dụng để tạo trình giả lập mới?

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

## Câu hỏi 4

Giả sử ứng dụng của bạn có câu lệnh log sau:

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

Bạn sẽ thấy câu lệnh "MainActivity layout is complete" trong **Logcat pane** nếu menu **Log level** được đặt ở mức nào? (Có thể chọn nhiều câu trả lời)

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

## Nộp bài kiểm tra ứng dụng

Kiểm tra để đảm bảo rằng ứng dụng có các yếu tố sau:

- Một **Activity** hiển thị "Hello World" trên màn hình.
- Các câu lệnh **Log** trong **onCreate()** của **MainActivity**.
- Mức **Log** trong **Logcat** chỉ hiển thị các thông báo **debug** hoặc **error**.

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

#### Giới thiệu

Giao diện người dùng (UI) hiển thị 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 các thành phần UI tương tác như nút bấm, hộp kiểm và trường nhập văn bản.

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

- **TextView** để hiển thị văn bản.
- **EditText** để cho phép người dùng nhập và chỉnh sửa văn bản.
- **Button** và các phần tử có thể nhấp khác (chẳng hạn như **RadioButton**, **CheckBox** và **Spinner**) để cung cấp hành vi tương tác.
- **ScrollView** và **RecyclerView** để hiển thị các mục có thể cuộn.
- **ImageView** để hiển thị hình ảnh.
- **ConstraintLayout** và **LinearLayout** để chứa các phần tử **View** khác và định vị chúng.

Mã Java hiển thị và điều khiển UI được chứa trong một lớp mở rộng từ **Activity**. Một **Activity** thường được liên kết với một bố cục UI được định nghĩa trong tệp XML (**eXtended Markup Language**).

Tệp XML này thường được đặt tên theo **Activity** của nó và xác định cách bố trí các phần tử **View** trên màn hình.

Ví dụ, mã **MainActivity** trong ứng dụng **Hello World** hiển thị một bố cục được định nghĩa trong tệp **activity\_main.xml**, trong đó bao gồm một **TextView** có nội dung "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 hoặc yêu cầu dữ liệu từ cơ sở dữ liệu hoặc internet. Bạn sẽ tìm hiểu thêm về lớp **Activity** trong một bài học khác.

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

## Những gì bạn cần biết trước

Bạn nên quen thuộc với:

- Cách cài đặt và mở Android Studio.
- Cách tạo ứng dụng **Hello World**.
- Cách chạy ứng dụng **Hello World**.

## Những gì bạn sẽ học

- Cách tạo một ứng dụng có hành vi tương tác.
- Cách sử dụng trình chỉnh sửa bố cục để thiết kế giao diện.
- Cách chỉnh sửa bố cục bằng XML.
- Nhiều thuật ngữ mới. Hãy tham khảo **bảng thuật ngữ và khái niệm** để có các định nghĩa dễ hiểu.

## Những gì bạn sẽ làm

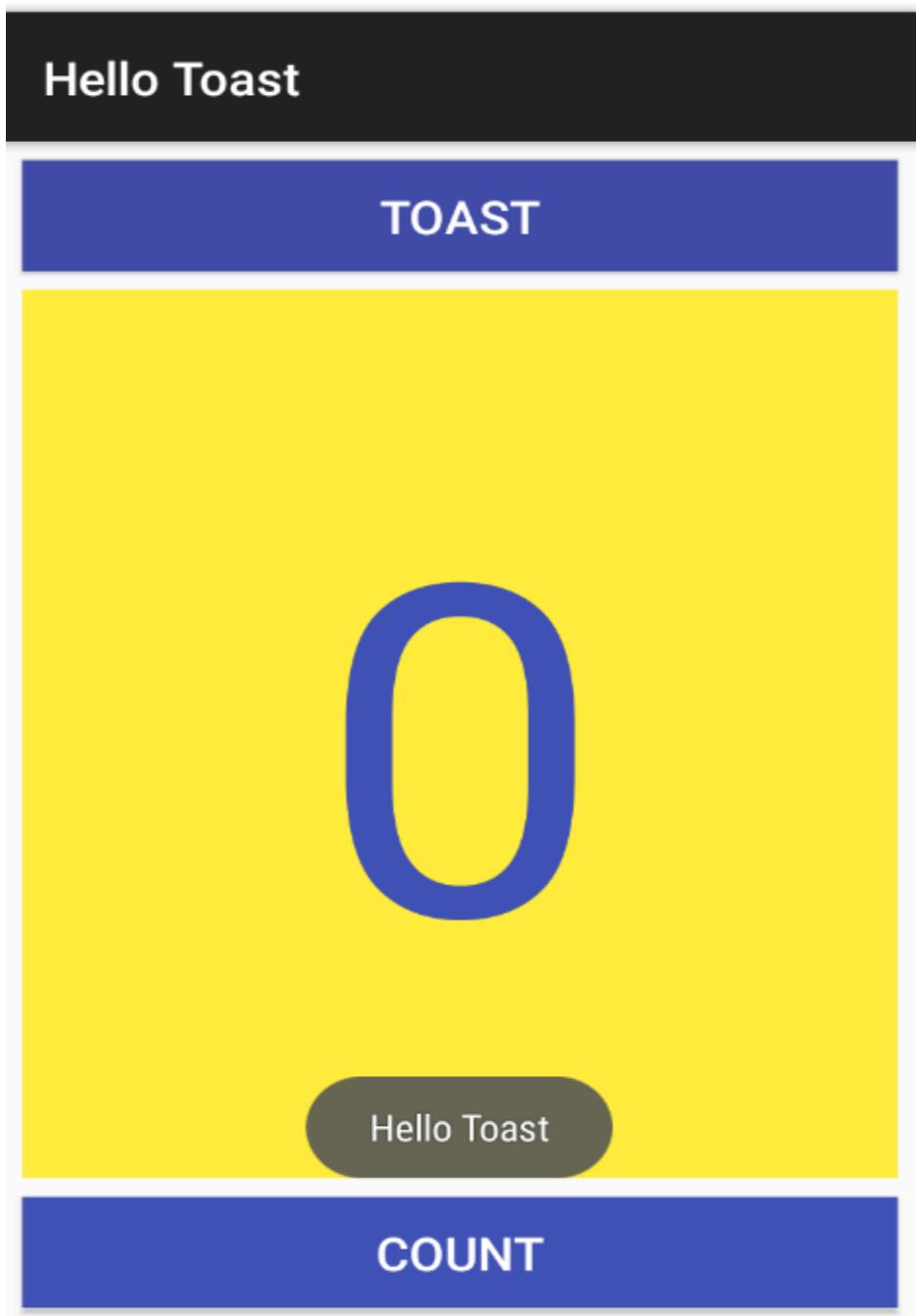
- Tạo một ứng dụng và thêm hai **Button** cùng một **TextView** vào bố cục.
- Điều chỉnh từng phần tử trong **ConstraintLayout** để ràng buộc chúng với lề và các phần tử khác.
- Thay đổi thuộc tính của các phần tử UI.
- Chỉnh sửa bố cục của ứng dụng trong XML.
- Trích xuất chuỗi văn bản cố định thành tài nguyên chuỗi (**string resources**).
- Triển khai phương thức xử lý sự kiện nhấp chuột (**click-handler methods**) để hiển thị thông báo trên màn hình khi người dùng chạm vào từng **Button**.



## Tổng quan về ứng dụng

Ứng dụng **HelloToast** bao gồm hai phần tử **Button** và một **TextView**. Khi người dùng chạm vào **Button** thứ nhất, một thông báo ngắn (**Toast**) sẽ xuất hiện trên màn hình. Khi chạm vào **Button** thứ hai, bộ đếm số lần nhấp (**click counter**) được hiển thị trong **TextView** sẽ tăng lên, bắt đầu từ số 0.

Đây là giao diện cuối cùng của ứng dụng:



## Bài 1: Tạo và khám phá một dự án mới

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

### 1.1 Tạo dự án Android Studio

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

Thuộc tính	Giá trị
Tên ứng dụng	Hello Toast
Tên công ty	com.example.android (hoặc miền của bạn)
Phone and Tablet Minimum SDK	API 15: Android 4.0.3 IceCreamSandwich
Mẫu (Template)	Empty Activity
Tùy chọn "Generate Layout file"	Được chọn
Tùy chọn "Backwards Compatibility"	Được chọn

2. Chạy ứng dụng:

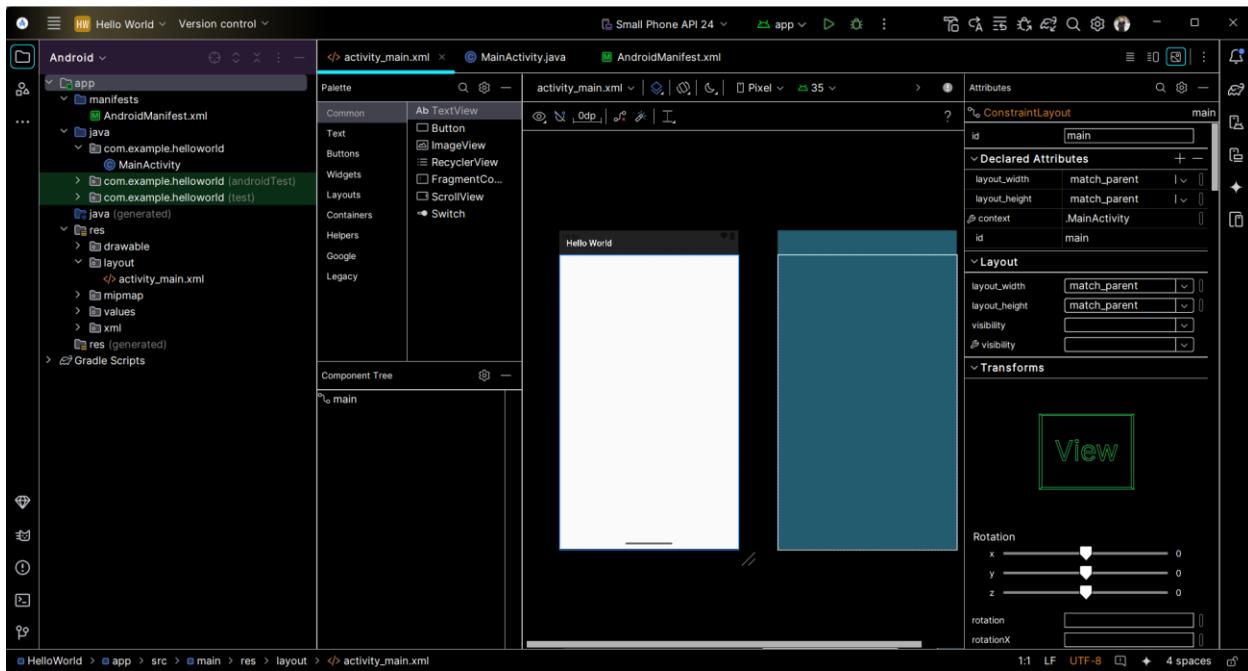
- Chọn **Run > Run app** hoặc nhấn vào **biểu tượng Run** trên thanh công cụ để **biên dịch và chạy ứng dụng** trên trình giả lập hoặc thiết bị thực tế.

## Khám phá trình chỉnh sửa bố cục (Layout Editor)

Android Studio cung cấp trình chỉnh sửa bố cục (layout editor) để giúp bạn xây dựng giao diện người dùng (UI) một cách nhanh chóng. Trình chỉnh sửa này cho phép bạn kéo thả các phần tử vào chế độ thiết kế trực quan (design view) và chế độ bản thiết kế (blueprint view), căn chỉnh vị trí, thêm ràng buộc (constraints) và thiết lập thuộc tính.

Ràng buộc (Constraints) giúp xác định vị trí của một phần tử UI trong bố cục. Một ràng buộc đại diện cho một kết nối hoặc căn chỉnh với một phần tử khác, bố cục cha, hoặc một đường hướng dẫn vô hình.

## Khám phá trình chỉnh sửa bố cục



Hãy làm theo các bước sau:

- Trong **khung Project > Android**, điều hướng đến thư mục **app > res > layout**, sau đó **nhấp đúp vào tệp** `activity_main.xml` để mở nó (nếu chưa mở).
- Chuyển sang **tab Design** nếu nó chưa được chọn.
  - Tab Design:** Dùng để thao tác với các phần tử UI và bố cục.
  - Tab Text:** Dùng để chỉnh sửa mã XML của bố cục.
- Pan Palettes** hiển thị danh sách các phần tử UI mà bạn có thể sử dụng trong giao diện ứng dụng.
- Pan Component Tree** hiển thị **cây phân cấp** của các phần tử UI.
  - Các phần tử UI được sắp xếp theo thứ bậc **cha – con**, trong đó phần tử con sẽ kế thừa thuộc tính từ phần tử cha.
  - Ví dụ: Trong hình minh họa, **TextView** là phần tử con của **ConstraintLayout**.
- Pan thiết kế và bản thiết kế** hiển thị các phần tử UI trong bố cục.
  - Trong hình minh họa, bố cục chỉ có một phần tử duy nhất là **TextView** hiển thị nội dung "Hello World".
- Tab Attributes** hiển thị **pane thuộc tính**, nơi bạn có thể thiết lập các thuộc tính cho một phần tử UI.

**Mẹo:**

- Xem tài liệu [Xây dựng giao diện UI với Layout Editor](#) để biết thêm chi tiết.

- Tham khảo tài liệu [Android Studio](#) để tìm hiểu đầy đủ về Android Studio.

## Bài 2: Thêm các phần tử giao diện vào trình chỉnh sửa bố cục

Trong bài này, bạn sẽ tạo giao diện người dùng (UI) cho ứng dụng **HelloToast** trong trình chỉnh sửa bố cục bằng cách sử dụng các tính năng của **ConstraintLayout**.

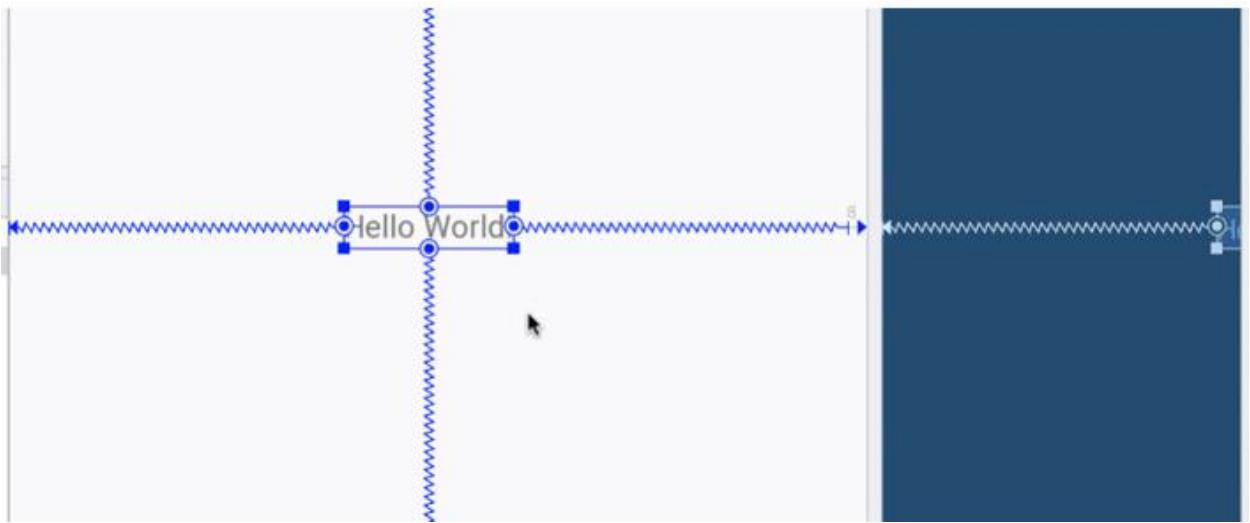
Bạn có thể tạo ràng buộc (constraints) **thủ công** (hướng dẫn sau) hoặc **tự động** bằng công cụ **Autoconnect**.

---

### 2.1 Kiểm tra ràng buộc của phần tử

Hãy làm theo các bước sau:

1. **Mở tệp activity\_main.xml** trong khung Project > Android, nếu nó chưa mở.
  - Nếu tab **Design** chưa được chọn, hãy nhấp vào nó.
  - Nếu không thấy bản thiết kế (**blueprint**), nhấp vào nút **Select Design Surface**  trên thanh công cụ và chọn **Design + Blueprint**.
2. **Kiểm tra công cụ Autoconnect** 
  - Công cụ **Autoconnect** nằm trên thanh công cụ và **được bật theo mặc định**.
  - Hãy kiểm tra và đảm bảo nó **không bị vô hiệu hóa**.
3. **Phóng to giao diện thiết kế**
  - Nhấp vào nút **zoom in**  để phóng to khu vực thiết kế và bản thiết kế.
4. **Chọn phần tử TextView**
  - Trong **Component Tree**, chọn **TextView** (có nội dung "Hello World").
  - Khi chọn, **TextView** sẽ được làm nổi bật trong **cửa sổ thiết kế** và **các ràng buộc (constraints)** của nó sẽ hiển thị.
5. **Xóa và thêm lại ràng buộc ngang**
  - Nhấp vào **chấm tròn (circular handle)** bên phải **TextView** để **xóa ràng buộc ngang**, khiến **TextView** nhảy sang trái.
  - Để thêm lại ràng buộc này, nhấp vào **cùng chấm tròn**, kéo một đường đến cạnh phải của bố cục.



---

### Các nút điều khiển trong blueprint hoặc design pane:

#### Constraint handle (tay nắm ràng buộc)

- Để tạo một **ràng buộc**, nhấp vào **chấm tròn** ở cạnh của phần tử.
- Kéo tay nắm này đến một **tay nắm khác** hoặc **đường biên** của phần tử cha.
- Đường zic-zac** hiển thị ràng buộc vừa tạo.



#### Resizing handle (tay nắm thay đổi kích thước)

- Để **thay đổi kích thước** phần tử, kéo **tay nắm vuông** ở góc.
- Khi kéo, tay nắm sẽ chuyển thành **góc xiên**.

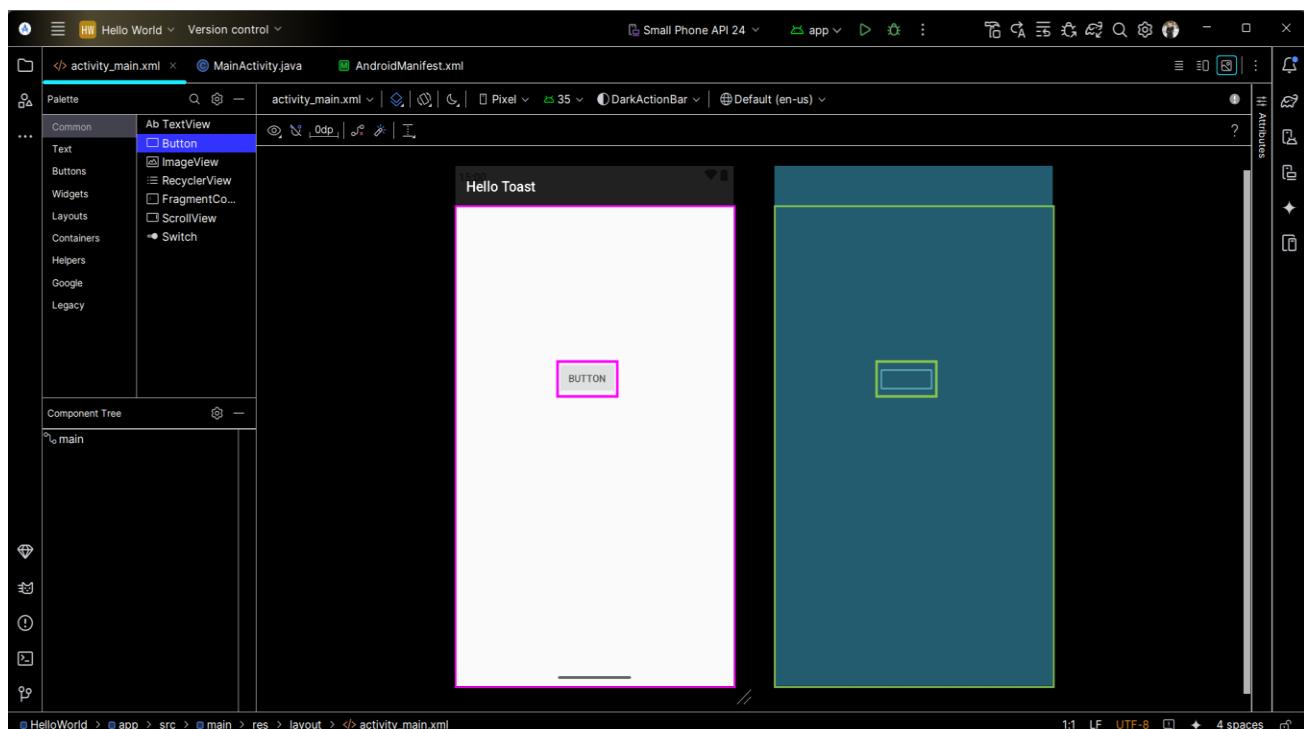


## 2.2 Thêm một nút Button vào bố cục

Khi được bật, công cụ **Autoconnect** sẽ tự động tạo **hai hoặc nhiều ràng buộc** cho một phần tử UI với bố cục cha. Sau khi bạn kéo phần tử vào bố cục, công cụ này sẽ tạo ràng buộc dựa trên vị trí của phần tử.

## Các bước thêm một Button:

1. Xóa mọi phần tử cũ
  - o Vì không cần **TextView**, hãy chọn nó và nhấn **phím Delete** hoặc chọn **Edit > Delete**.
  - o Bây giờ, bố cục của bạn hoàn toàn trống.
2. Thêm một Button vào bố cục
  - o Trong khung **Palette**, kéo một **Button** vào **bất kỳ vị trí nào** trong bố cục.
  - o Nếu bạn thả **Button** vào **khu vực giữa phía trên**, có thể **các ràng buộc sẽ tự động xuất hiện**.
  - o Nếu không, bạn có thể **kéo tay nắm ràng buộc** để kết nối **Button** với **cạnh trên, cạnh trái và cạnh phải** của **bố cục**, như minh họa trong hình động bên dưới.

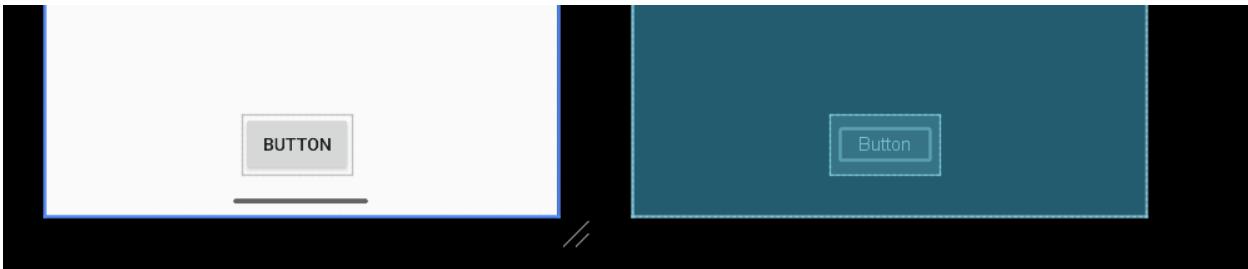


## 2.3 Thêm một Button thứ hai vào bố cục

Hãy làm theo các bước sau để thêm một **nút Button thứ hai** vào giao diện:

1. Thêm Button mới
  - o Trong khung **Palette**, kéo một **Button** khác vào **giữa bố cục** như minh họa.
  - o **Autoconnect** có thể tự động tạo **ràng buộc ngang** cho bạn.

- Nếu không, bạn có thể **kéo tay nắm ràng buộc** để tạo ràng buộc ngang thủ công.
2. **Thêm ràng buộc dọc**
- Kéo một **ràng buộc dọc** từ **Button thứ hai** đến **cạnh dưới** của **bối cảnh** để cố định vị trí.



## Xóa ràng buộc của phần tử UI

### Xóa tất cả ràng buộc của một phần tử

- Chọn phần tử và di chuyển chuột qua nó.
- **Nút "Clear Constraints"** sẽ xuất hiện. Nhấp vào để **xóa tất cả ràng buộc** của phần tử đó.

### Xóa một ràng buộc cụ thể

- Nhấp vào **tay nắm ràng buộc** đã đặt để xóa chỉ ràng buộc đó.

### Xóa tất cả ràng buộc trong bối cảnh

- Nhấp vào công cụ "**Clear All Constraints**" trên **thanh công cụ**.
- Công cụ này rất hữu ích khi bạn muốn **thiết lập lại toàn bộ ràng buộc** trong giao diện.

## Nhiệm vụ 3: Thay đổi thuộc tính của phần tử giao diện người dùng (UI)

Ngăn Attributes cung cấp quyền truy cập vào tất cả các thuộc tính XML mà bạn có thể gán cho một phần tử UI. Bạn có thể tìm thấy các thuộc tính (được gọi là "properties") chung cho tất cả các chế độ xem (views) trong tài liệu của lớp View.

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

### **3.1 Thay đổi kích thước nút**

Trình chỉnh sửa bộ cục cung cấp các bộ điều khiển thay đổi kích thước ở cả bốn góc của View để bạn có thể nhanh chóng thay đổi kích thước View.

Bạn có thể kéo các bộ điều khiển ở mỗi góc của View để thay đổi kích thước, nhưng làm như vậy sẽ cố định kích thước chiều rộng và chiều cao (hardcoded). Tránh mã hóa kích thước cố định cho hầu hết các phần tử View vì nó không thích nghi với nội dung và kích thước màn hình khác nhau.

Thay vào đó, sử dụng ngăn Attributes ở phía bên phải trình chỉnh sửa bộ cục để chọn chế độ kích thước không dùng kích thước cố định. Ngăn Attributes bao gồm một bảng điều chỉnh kích thước hình vuông được gọi là View Inspector ở trên cùng.

layout_marginBottom	8dp
background	#3F51B5
id	btnCount
text	@string/button_label_c...
textColor	@color/white
textSize	20dp

✓ Layout

Constraint Widget

✓ Constraints

- Start → StartOf parent (8dp)
- End → EndOf parent (8dp)
- Bottom → BottomOf parent (8dp)
- Horizontal Bias (0.0)

layout_width	0dp
layout_height	wrap_content
visibility	
↳ visibility	

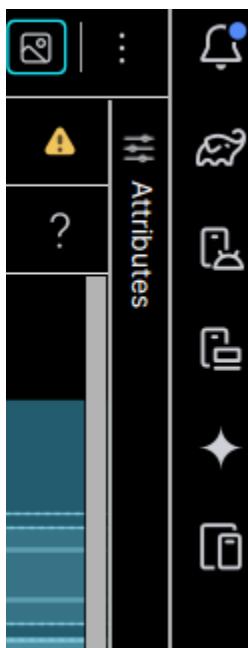
Ý nghĩa của các ký hiệu trong hình vuông:

- Điều khiển chiều cao (Height control):

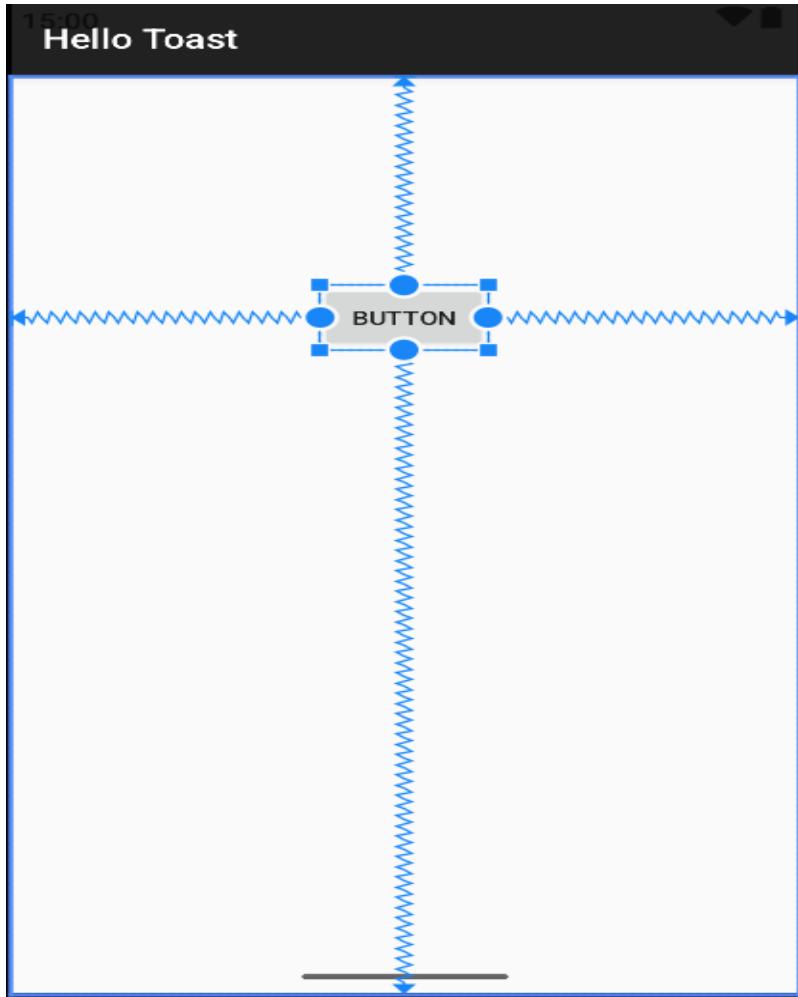
- Điều khiển này xác định thuộc tính layout\_height và xuất hiện ở hai đoạn trên và dưới của hình vuông.
  - Các góc xiên cho biết điều khiển này được đặt thành wrap\_content, có nghĩa là View sẽ mở rộng theo chiều dọc khi cần để vừa với nội dung.
  - Số "8" biểu thị khoảng cách margin tiêu chuẩn được đặt là 8dp.
2. **Điều khiển chiều rộng (Width control):**
- Điều khiển này xác định thuộc tính layout\_width và xuất hiện ở hai đoạn bên trái và bên phải của hình vuông.
  - Các góc xiên cho biết điều khiển này được đặt thành wrap\_content, có nghĩa là View sẽ mở rộng theo chiều ngang khi cần để vừa với nội dung, đến giới hạn margin 8dp.
3. **Nút đóng ngăn Attributes (Attributes pane close button):**
- Nhập vào nút này để đóng ngăn Attributes.

Làm theo các bước sau:

1. Chọn **Button** trên cùng trong ngăn **Component Tree**.
2. Nhấp vào tab **Attributes** ở phía bên phải cửa sổ trình chỉnh sửa bộ cục.

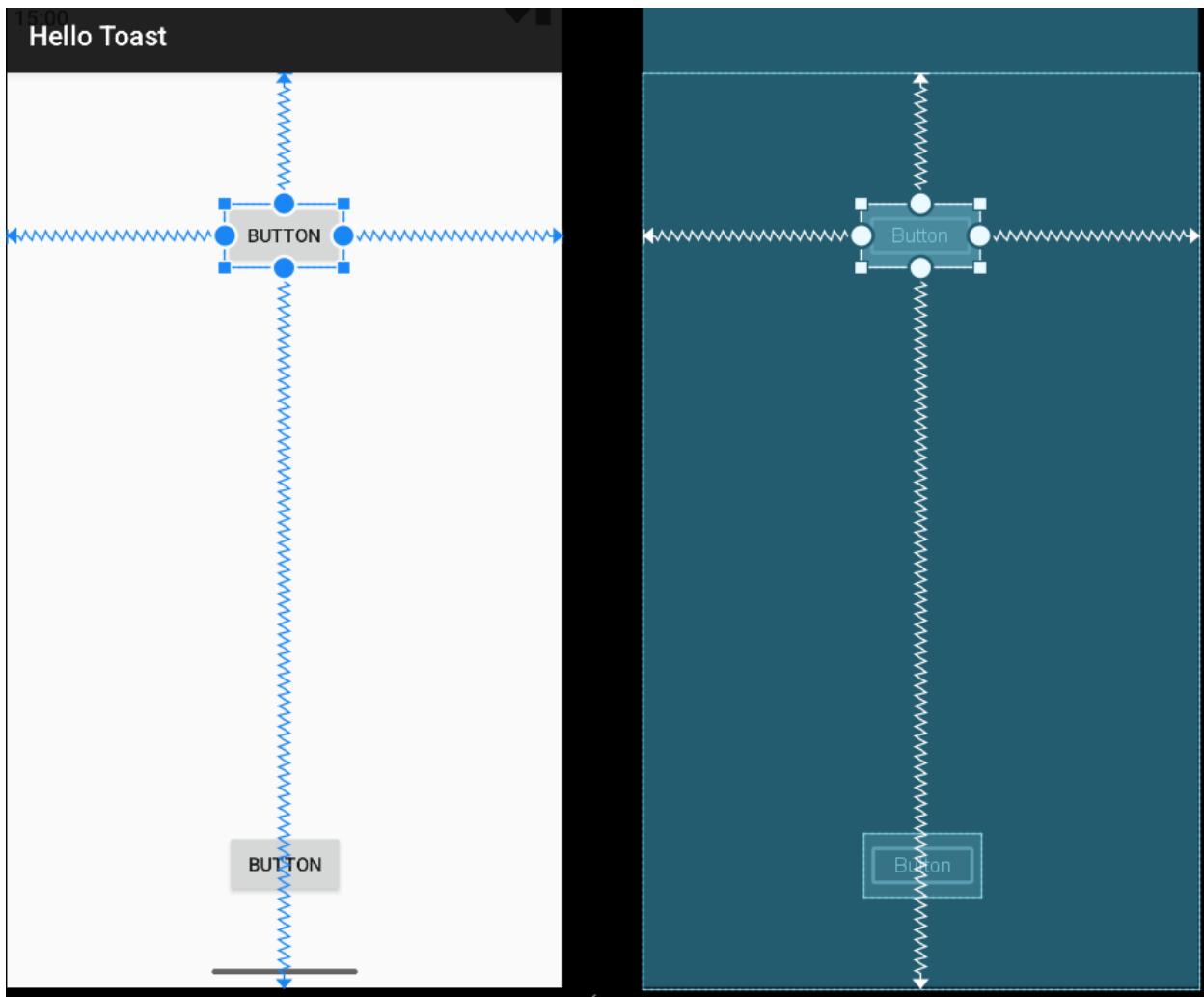


3. Nhấp vào **bộ điều khiển chiều rộng** hai lần — lần nhấp đầu tiên thay đổi thành **Fixed** với các đường thẳng, và lần nhấp thứ hai thay đổi thành **Match Constraints** với các cuộn lò xo, như được minh họa trong hình động bên dưới.



Kết quả của việc thay đổi **bộ điều khiển chiều rộng**, thuộc tính **layout\_width** trong ngăn **Attributes** hiển thị giá trị **match\_constraint**, và phần tử **Button** kéo dài theo chiều ngang để lấp đầy không gian giữa hai bên trái và phải của bố cục.

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



Như đã trình bày trong các bước trước, các thuộc tính **layout\_width** và **layout\_height** trong ngăn **Attributes** thay đổi khi bạn thay đổi các bộ điều khiển chiều cao và chiều rộng trong **view inspector**. Các thuộc tính này có thể nhận một trong ba giá trị đối với bố cục **ConstraintLayout**:

- **match\_constraint**: Thiết lập này mở rộng phần tử **View** để lấp đầy không gian của phần tử cha theo chiều rộng hoặc chiều cao—đến mức biên, nếu được đặt. Trong trường hợp này, phần tử cha là **ConstraintLayout**. Bạn sẽ tìm hiểu thêm về **ConstraintLayout** trong nhiệm vụ tiếp theo.
- **wrap\_content**: Thiết lập này thu nhỏ kích thước của phần tử **View** sao cho chỉ vừa đủ để bao bọc nội dung của nó. Nếu không có nội dung, phần tử **View** sẽ trở nên không hiển thị.

- Để chỉ định một kích thước cố định phù hợp với kích thước màn hình của thiết bị, hãy sử dụng một số cố định của đơn vị điểm ảnh độc lập với mật độ (**dp units**). Ví dụ, **16dp** nghĩa là 16 điểm ảnh độc lập với mật độ.

**Mẹo:** Nếu bạn thay đổi thuộc tính **layout\_width** bằng cách sử dụng menu bật lên của nó, thuộc tính **layout\_width** sẽ được đặt thành **zero (0)** vì không có kích thước nào được đặt cụ thể. Cài đặt này giống với **match\_constraint**—phần tử View có thể mở rộng hết mức có thể để đáp ứng các ràng buộc và cài đặt biên.

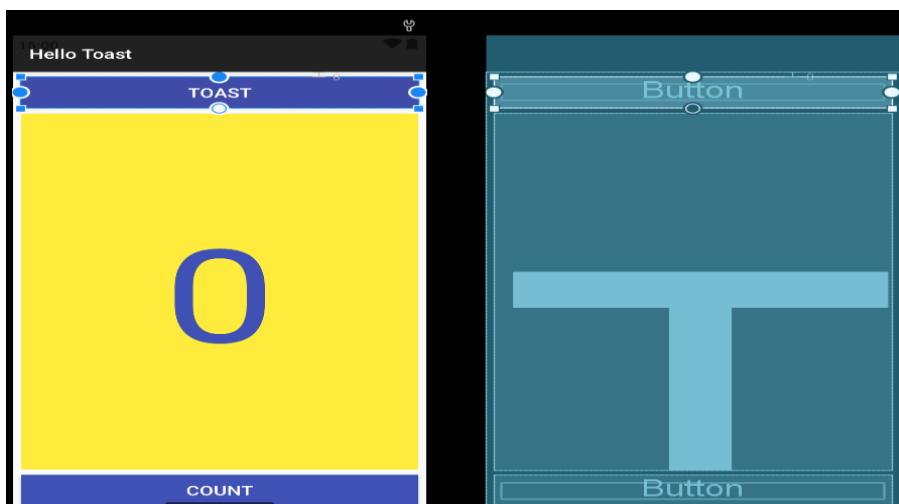
### 3.2 Thay đổi các thuộc tính của Button

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

Ngăn Attributes cung cấp quyền truy cập vào tất cả các thuộc tính 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 minh họa sau đây hướng dẫn cách thực hiện các bước sau:

- Sau khi chọn Button đầu tiên, chỉnh sửa trường ID ở đầu ngăn Attributes thành `button_toast` cho thuộc tính `android:id`, được dùng để xác định phần tử trong bố cục.
- Đặt thuộc tính `background` thành `@color/colorPrimary`. (Khi bạn nhập `@c`, các lựa chọn sẽ xuất hiện để dễ dàng chọn lựa.)
- Đặt thuộc tính `textColor` thành `@android:color/white`.
- Chỉnh sửa thuộc tính `text` thành `Toast`.



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

#### Lưu ý:

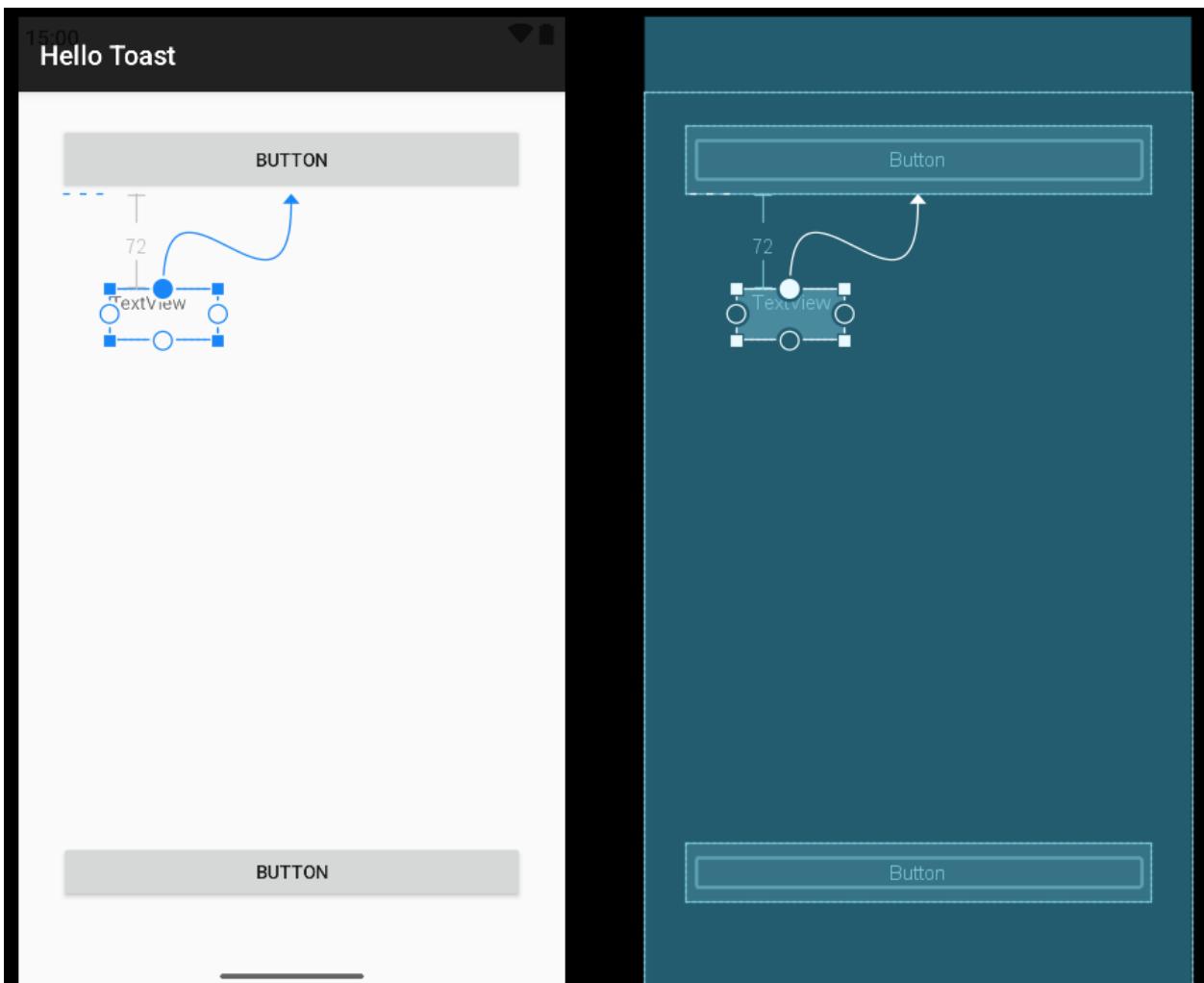
Màu colorPrimary là màu chủ đạo của theme, một trong những màu cơ sở được định nghĩa sẵn trong tệp tài nguyên colors.xml. Nó được sử dụng cho thanh ứng dụng (app bar). Việc sử dụng các màu cơ sở cho các phần tử UI khác sẽ tạo ra giao diện người dùng (UI) thống nhất. Bạn sẽ tìm hiểu thêm về theme ứng dụng và Material Design trong một bài học khác.

### Nhiệm vụ 4: Thêm một TextView và thiết lập các thuộc tính

Một trong những lợi ích của ConstraintLayout là khả năng căn chỉnh hoặc ràng buộc các phần tử tương đối 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 các lề và theo chiều dọc giữa hai nút Button. Sau đó, bạn sẽ thay đổi các thuộc tính của TextView trong ngăn Attributes.

#### 4.1 Thêm một TextView và thiết lập ràng buộc

- Như minh họa trong hình động dưới đây, kéo một TextView từ ngăn Palette vào phần trên của bố cục. Kéo một ràng buộc từ phía trên của TextView đến chốt ở phía dưới của nút Toast. Việc này sẽ ràng buộc TextView nằm ngay bên dưới nút Toast.
- Như minh họa trong hình động dưới đây, kéo một ràng buộc từ phía dưới của **TextView** đến chốt ở phía trên của nút **Count**. Sau đó, kéo các ràng buộc từ hai bên của **TextView** đến hai cạnh của bố cục. Việc này ràng buộc **TextView** nằm giữa bố cục, ở giữa hai nút **Button**.



## 4.2 Đặt các thuộc tính cho TextView

Với **TextView** được chọn, hãy mở bảng thuộc tính (**Attributes pane**) nếu nó chưa mở. Đặt các thuộc tính cho **TextView** như hướng dẫn dưới đây. Những thuộc tính mới sẽ được giải thích sau:

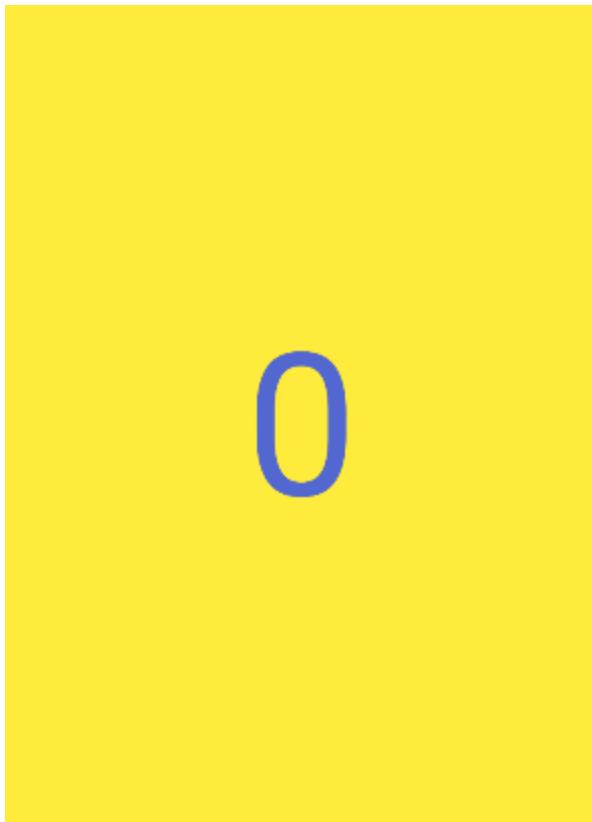
1. Đặt **ID** thành `show_count`.
2. Đặt **text** thành `0`.
3. Đặt **textSize** thành `160sp`.
4. Đặt **textStyle** thành **B** (in đậm) và **textAlignment** thành **ALIGNCENTER** (căn giữa đoạn văn bản).
5. Thay đổi ché độ kích thước ngang và dọc (**layout\_width** và **layout\_height**) thành **match\_constraint**.

6. Đặt **textColor** thành @color/colorPrimary.
  7. Cuộn xuống bảng thuộc tính và nhập vào "View all attributes", cuộn đến trang thứ hai của các thuộc tính, tìm thuộc tính "background" và nhập giá trị #FFF00 để chọn một sắc thái màu vàng.
  8. Cuộn xuống thuộc tính "gravity", mở rộng mục "gravity" và chọn "center\_ver" (để căn giữa theo chiều dọc).
- **textSize**: Kích thước văn bản của TextView. Trong bài học này, kích thước được đặt là **160sp**. **sp** là viết tắt của **scale-independent pixel** (pixel không phụ thuộc tỷ lệ) và giống như **dp**, là đơn vị điều chỉnh theo mật độ màn hình và tùy chọn kích thước phông chữ của người dùng. Hãy sử dụng đơn vị **dp** khi bạn xác định kích thước phông chữ để đảm bảo kích thước được điều chỉnh cho cả mật độ màn hình và sở thích của người dùng.
  - **textStyle** và **textAlignment**: Kiểu văn bản, được đặt là **B (bold)** (in đậm) trong bài học này. Căn chỉnh văn bản, được đặt là **ALIGNCENTER** (căn giữa đoạn văn bản).
  - **gravity**: Thuộc tính *gravity* xác định cách một *View* được căn chỉnh trong *View* hoặc *ViewGroup* cha của nó. Trong bước này, bạn căn chỉnh *TextView* theo chiều dọc ở giữa *ConstraintLayout* cha.

Bạn có thể nhận thấy rằng thuộc tính *background* nằm ở trang đầu tiên của bảng *Attributes* (Thuộc tính) đối với một *Button*, nhưng lại nằm ở trang thứ hai đối với một *TextView*. Bảng *Attributes* thay đổi tùy thuộc vào từng loại *View*: các thuộc tính phổ biến nhất của loại *View* sẽ xuất hiện trên trang đầu tiên, và các thuộc tính khác được liệt kê trên trang thứ hai. Để quay lại trang đầu tiên của bảng *Attributes*, nhấn vào biểu tượng trên thanh công cụ ở phía trên cùng của bảng.

#### Task5: Chính sửa bộ cục trong XML

Bộ cục của ứng dụng Hello Toast gần như đã hoàn thành! Tuy nhiên, bên cạnh mỗi phần tử giao diện người dùng trong bảng Component Tree lại xuất hiện một dấu chấm than. Di chuyển con trỏ chuột qua các dấu chấm than này để xem thông báo cảnh báo, như hình minh họa bên dưới. Cùng một cảnh báo xuất hiện cho cả ba phần tử: chuỗi được mã hóa cứng nên được thay thế bằng tài nguyên.



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

## 5.1 Mở mã XML của bố cục

Trong nhiệm vụ này, hãy mở tệp activity\_main.xml nếu nó chưa được mở, và nhấp vào tab Text ở dưới cùng của trình chỉnh sửa bố cục.

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

```
<?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">
```

```
        android:orientation="vertical"
        tools:context=".MainActivity" >

    <Button
        android:id="@+id/btnToast"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:background="#3F4BA7"
        android:text="@string/button_label_toast"
        android:textColor="@color/white"
        android:textSize="20dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textToast"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#FFEB3B"
        android:gravity="center"
        android:text="@string/count_initial_value"
        android:textAllCaps="true"
        android:textColor="#3F51B5"
        android:textSize="200dp"
        app:layout_constraintBottom_toTopOf="@+id/btnCount"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnToast"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/btnCount"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#3F51B5"
        android:text="@string/button_label_count"
        android:textColor="@color/white"
        android:textSize="20dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

## 5.2 Tách các chuỗi thành tài nguyên

Thay vì mã hóa cứng các chuỗi, việc sử dụng tài nguyên chuỗi là một thực hành tốt, vì các chuỗi được lưu trữ trong một tệp riêng giúp quản lý dễ dàng hơn, đặc biệt khi bạn sử dụng các chuỗi này nhiều lần. Ngoài ra, tài nguyên chuỗi là bắt buộc để dịch và địa phương hóa ứng dụng của bạn, vì bạn cần tạo một tệp tài nguyên chuỗi cho mỗi ngôn ngữ.

Các bước thực hiện:

1. Nhập một lần vào từ "Toast" (chuỗi cảnh báo đầu tiên được đánh dấu).
2. Nhấn Alt-Enter trên Windows hoặc Option-Enter trên macOS và chọn Extract string resource từ menu bật lên.
3. Nhập "button\_label\_toast" cho tên tài nguyên.
4. Nhập OK. Một tài nguyên chuỗi sẽ được tạo trong tệp values/res/values.xml, và chuỗi trong mã của bạn sẽ được thay thế bằng tham chiếu tới tài nguyên:  
@string/button\_label\_toast
5. Tách các chuỗi còn lại: sử dụng "button\_label\_count" cho "Count" và "count\_initial\_value" cho "0".
6. Trong ngăn Project > Android, mở rộng mục values trong thư mục res, sau đó nhấp đúp vào tệp strings.xml để xem các tài nguyên chuỗi của bạn trong tệp strings.xml.

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

7. Bạn cần một chuỗi khác để sử dụng trong một nhiệm vụ sau hiển thị thông báo. Thêm vào tệp strings.xml một tài nguyên chuỗi mới có tên là toast\_message cho cụm từ "Hello Toast!"

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

Mẹo: Các tài nguyên chuỗi bao gồm tên ứng dụng, xuất hiện trên thanh ứng dụng ở đầu màn hình nếu bạn khởi tạo dự án bằng Mẫu Trống. Bạn có thể thay đổi tên ứng dụng bằng cách chỉnh sửa tài nguyên app\_name.

## Nhiệm vụ 6: Thêm xử lý sự kiện onClick cho các nút

Trong nhiệm vụ này, bạn sẽ thêm một phương thức Java cho mỗi nút trong MainActivity, phương thức này sẽ được gọi khi người dùng nhấp vào nút.

6.1 Thêm thuộc tính onClick và phương thức xử lý cho mỗi nút  
Một click handler là phương thức được gọi khi người dùng nhấp hoặc chạm vào một phần tử UI có thể nhấp được. Trong Android Studio, bạn có thể chỉ định tên của phương thức trong trường onClick ở ngăn Attributes của tab Design. Bạn cũng có thể chỉ định tên của phương thức xử lý trong trình chỉnh sửa XML bằng cách thêm thuộc tính android:onClick vào nút. Ở đây, bạn sẽ sử dụng phương pháp thứ hai vì bạn chưa tạo các phương thức xử lý, và trình chỉnh sửa XML cung cấp cách tự động tạo các phương thức đó.

1. Với trình chỉnh sửa XML (tab Text) đang mở, tìm nút có android:id được đặt là button\_toast.

```
<Button
    android:id="@+id	btnToast"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:background="#3F4BA7"
    android:text="@string/button_label_toast"
    android:textColor="@color/white"
    android:textSize="20dp"
    />
```

2. Thêm thuộc tính android:onClick vào cuối phần tử button\_toast, sau thuộc tính cuối cùng và trước dấu kết thúc "/>"
3. Nhập vào biểu tượng bóng đèn đỏ xuất hiện bên cạnh thuộc tính. Chọn Create click handler, chọn MainActivity, và nhấp OK. Nếu biểu tượng bóng đèn đỏ không xuất hiện, hãy nhập vào tên phương thức ("showToast"). Nhấn Alt-Enter (trên Windows/Linux) hoặc Option-Enter (trên Mac), chọn Create 'showToast(view)' in MainActivity, và nhấp OK. Thao tác này sẽ tạo ra một phương thức mẫu (placeholder method stub) cho phương thức showToast() trong MainActivity, như được hiển thị ở cuối các bước này.
4. Lặp lại hai bước cuối với nút button\_count: Thêm thuộc tính android:onClick vào cuối phần tử, và tạo click handler

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

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

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id	btnToast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:background="#3F4BA7"
        android:text="@string/button_label_toast"
        android:textColor="@color/white"
        android:textSize="20dp"
        android:onClick="countUp"/>
    />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#FFEB3B"
        android:gravity="center"
        android:text="@string/count_initial_value"
        android:textAllCaps="true"
        android:textColor="#3F51B5"
        android:textSize="120dp"
        android:layout_below="@+id	btnToast"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"/>

    <Button
        android:id="@+id	btnCount"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#3F51B5"
        android:text="@string/button_label_count"
```

```
        android:textColor="@color/white"
        android:textSize="20dp"
        android:layout_below="@+id/show_count"
        android:layout_centerHorizontal="true"
    />

</RelativeLayout>
```

5. Nếu MainActivity.java chưa được mở, mở rộng mục java trong ngăn Project > Android, mở rộng com.example.android.hellotoast, sau đó nhấp đúp vào MainActivity. Trình soạn thảo mã sẽ xuất hiện với mã của MainActivity.

```
package com.example.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        Log.d("MainActivity", "Hello World");
        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;
        });
    }
    no usages
    public void showToast(View v){

    }
    no usages
    public void countUp(View v){

    }
}
```

## 6.2 Chính sửa xử lý sự kiện của nút Toast

Bây giờ, bạn sẽ chỉnh sửa phương thức showToast() – xử lý sự kiện click của nút Toast trong MainActivity – để hiển thị một thông báo. Một Toast cung cấp cách hiển thị thông báo đơn giản trong một cửa sổ popup nhỏ. Toast chỉ chiếm không gian cần thiết để hiển thị thông báo, và Activity hiện tại vẫn hiển thị và có thể

tương tác. 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 của thao tác nhấn nút hoặc thực hiện một hành động.

Thực hiện theo các bước sau để chỉnh sửa xử lý sự kiện nút Toast:

1. Tìm vị trí của phương thức showToast() mới được tạo trong MainActivity.

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

2. Để tạo một đối tượng Toast, hãy gọi phương thức makeText (factory method) trên lớp Toast.

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

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

3. Cung cấp context của Activity của ứng dụng. Vì một Toast được hiển thị phía trên giao diện người dùng của Activity, hệ thống cần thông tin về Activity hiện tại. Khi bạn đã ở trong context của Activity mà bạn cần, hãy sử dụng từ khóa "this" như một cách rút gọn.

```
public void showToast(View v) {  
    Toast toast = Toast.makeText(this);  
}
```

4. Cung cấp thông báo để hiển thị, chẳng hạn như một tài nguyên chuỗi (ví dụ: toast\_message mà bạn đã tạo ở bước trước). Tài nguyên chuỗi toast\_message được nhận diện bằng R.string.toast\_message.

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

5. Cung cấp thời lượng hiển thị. Ví dụ, Toast.LENGTH\_SHORT hiển thị Toast trong một khoảng thời gian tương đối ngắn.

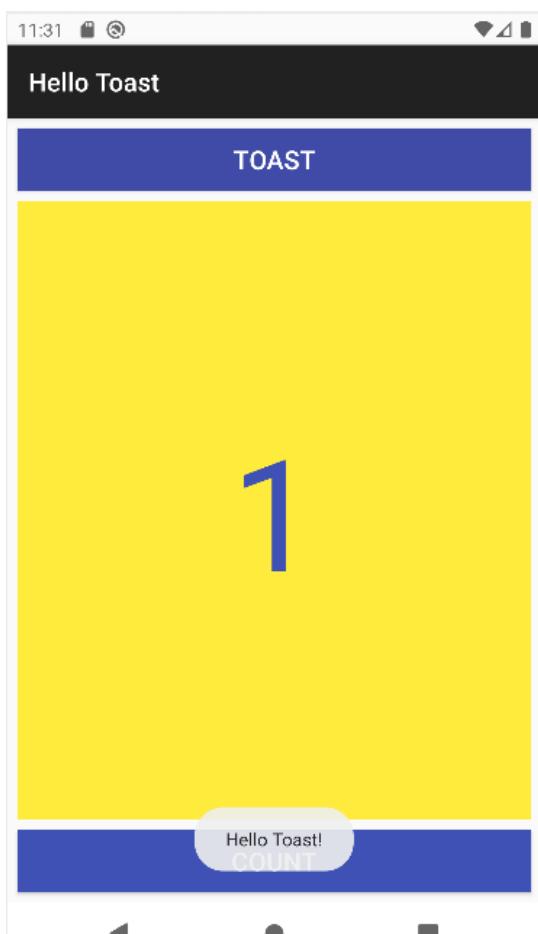
```
public void showToast(View v){  
    Toast toast = Toast.makeText(context: this,R.string.toast_message, Toast.LENGTH_LONG);  
}
```

Thời lượng hiển thị của Toast có thể là Toast.LENGTH\_LONG hoặc Toast.LENGTH\_SHORT. Thời gian thực tế là khoảng 3,5 giây cho Toast dài và 2 giây cho Toast ngắn.

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

```
public void showToast(View v){  
    Toast toast = Toast.makeText(context: this,R.string.toast_message, Toast.LENGTH_LONG);  
    toast.show();  
}
```

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



### 6.3 Chính sửa xử lý sự kiện của nút Count

Bây giờ, bạn sẽ chỉnh sửa phương thức countUp() – xử lý sự kiện click của nút Count trong MainActivity – sao cho nó hiển thị giá trị đếm hiện tại sau mỗi lần nhấn nút Count. Mỗi lần nhấn nút sẽ tăng giá trị đếm lên một đơn vị. Mã xử lý sự kiện cần thực hiện các điều sau:

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

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

1. Tìm vị trí của phương thức countUp() mới được tạo trong MainActivity, có dạng:

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

2. Để theo dõi giá trị đếm, bạn cần một biến thành viên riêng (private member variable). Mỗi lần nhấn nút Count sẽ tăng giá trị của biến này. Nhập đoạn mã sau (đoạn mã này sẽ được đánh dấu màu đỏ và xuất hiện biểu tượng bóng đèn đỏ):

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

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

3. Nhấp vào biểu tượng bóng đèn đỏ và chọn Create field 'mCount' từ menu bật lên. Thao tác này tạo ra một biến thành viên riêng tại đầu lớp MainActivity, và Android Studio mặc định kiểu của biến này là integer (int):
4. Thay đổi câu lệnh khai báo biến thành viên khởi tạo giá trị bằng 0:

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

5. Ngoài biến mCount, bạn cũng cần một biến thành viên riêng để lưu tham chiếu đến TextView hiển thị số đếm, gọi biến này là mShowCount:

```
public class MainActivity extends AppCompatActivity {  
    2 usages  
    private Button btnToast, btnCount;  
    2 usages  
    private TextView textToast;  
    2 usages  
    private int count = 0;
```

6. Nay, đã có mShowCount, bạn có thể lấy tham chiếu đến TextView thông qua ID đã đặt trong tệp bố cục. Để lấy tham chiếu này chỉ một lần, hãy thực hiện trong phương thức onCreate(). Như bạn đã học trong một bài học khác, phương thức onCreate() được dùng để inflate bố cục, tức là đặt content view của màn hình bằng tệp XML bố cục. Bạn cũng có thể sử dụng onCreate() để lấy tham chiếu đến các phần tử UI khác trong bố cục, chẳng hạn như TextView. Tìm phương thức onCreate() trong MainActivity:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    EdgeToEdge.enable(this);  
    setContentView(R.layout.activity_main);  
    Log.d("MainActivity", "Hello Toast");  
    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;  
        }) ;  
}
```

7. Thêm câu lệnh findViewById vào cuối phương thức onCreate() để gán giá trị cho mShowCount:

```

package com.example.helloworld;

import ...

public class MainActivity extends AppCompatActivity {
    private int mCount = 0;
    private TextView mShowCount;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable($this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main);
        Log.d(tag: "MainActivity", msg: "Hello Toast");
        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;
        });
        mShowCount = (TextView) findViewById(R.id.textToast);
    }
}

```

Một View, giống như một chuỗi, là một tài nguyên có thể có một ID. Gọi findViewById với ID của một view sẽ trả về đối tượng View. Vì phương thức này trả về một View, bạn cần ép kiểu kết quả về loại view mà bạn mong đợi, trong trường hợp này là (TextView).

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

```

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

```

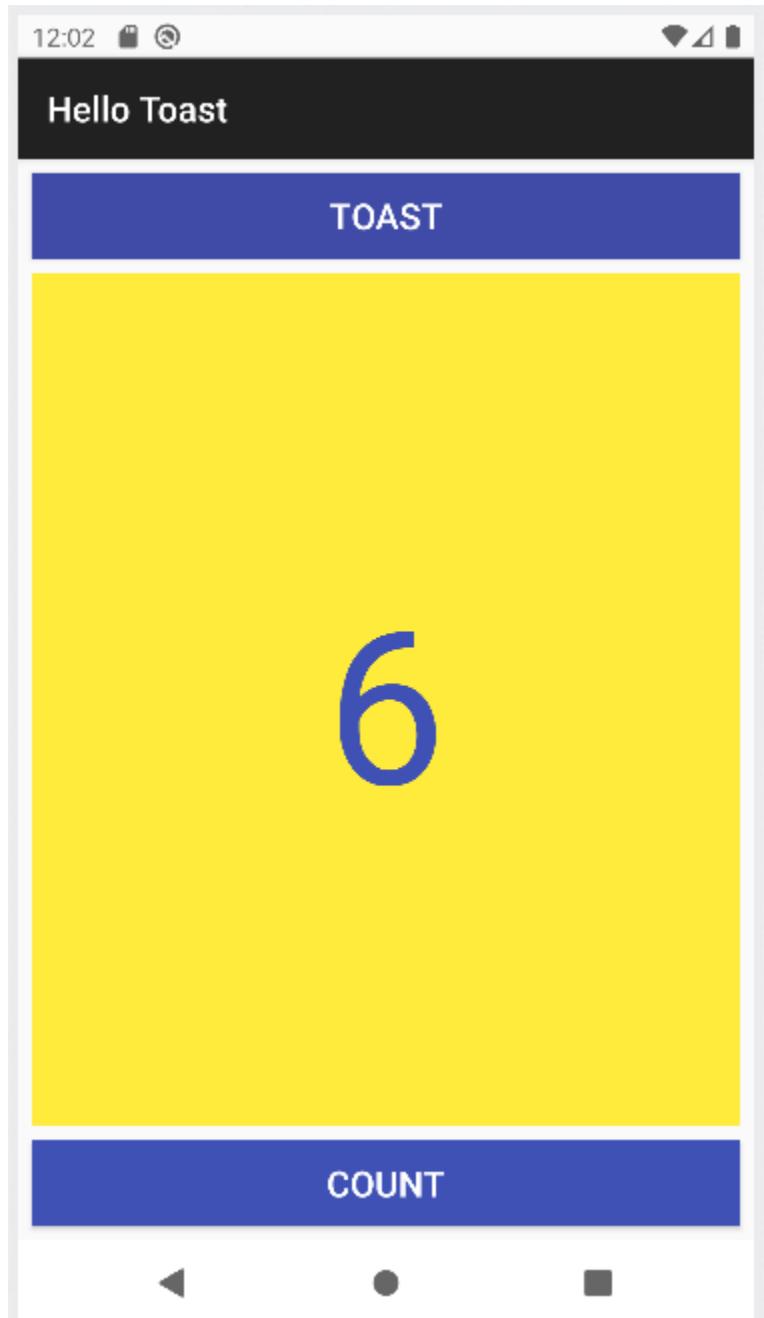
Toàn bộ phương thức countUp() bây giờ trông như sau:

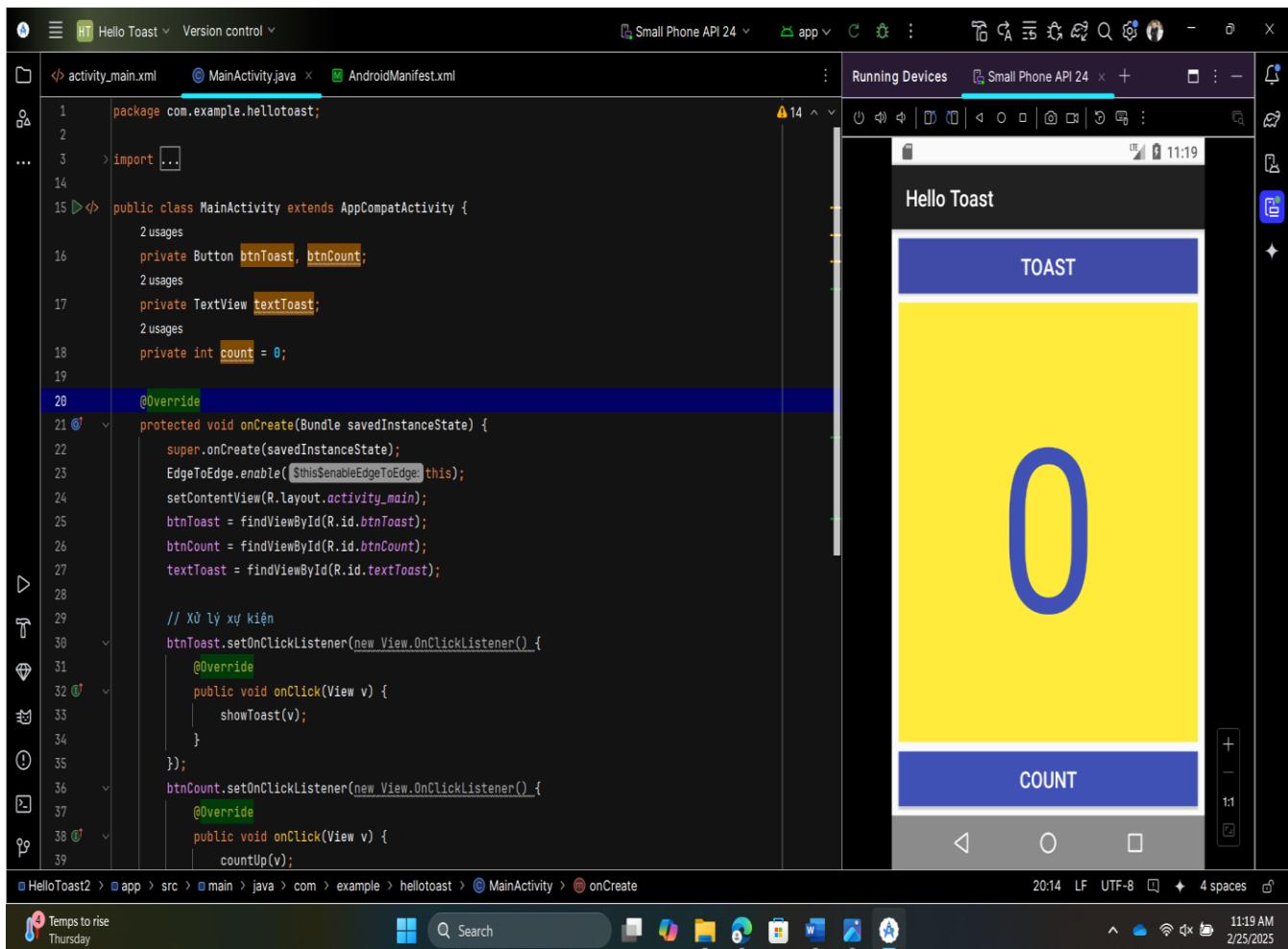
```

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

```

- Chạy ứng dụng để xác minh rằng giá trị đếm tăng lên mỗi khi bạn nhấn nút Count.





Mẹo: Để tìm hiểu chi tiết về cách sử dụng ConstraintLayout, hãy xem Codelab "Using ConstraintLayout to design your views".

**Thách thức** lập trình  
Lưu ý: Tất cả các thách thức lập trình đều không bắt buộc và không phải là điều kiện tiên quyết cho các bài học sau.

Ứng dụng HelloToast trông ổn khi thiết bị hoặc trình giả lập được đặt theo chiều dọc. Tuy nhiên, nếu bạn chuyển thiết bị hoặc trình giả lập sang chiều ngang, nút Count có thể chồng lên TextView ở phía dưới như trong hình minh họa bên dưới.

**Thử thách:** Thay đổi bố cục để nó trông đẹp cả khi ở chế độ ngang và dọc:

- Trên máy tính của bạn, sao chép thư mục dự án **HelloToast** và đổi tên nó thành **HelloToastChallenge**.

2. Mở **HelloToastChallenge** trong Android Studio và chỉnh sửa nó. (Xem Phụ lục: Tiện ích để biết hướng dẫn sao chép và chỉnh sửa một dự án.)
3. Thay đổi bố cục sao cho nút **Toast** và nút **Count** xuất hiện ở phía bên trái, như trong hình minh họa bên dưới. **TextView** xuất hiện bên cạnh chúng, nhưng chỉ rộng đủ để hiển thị nội dung của nó. (Gợi ý: Sử dụng `wrap_content`.)
4. Chạy ứng dụng ở cả chế độ ngang và dọc.

Tóm

tắt:

### **View, ViewGroup và layouts:**

- Tất cả các phần tử giao diện người dùng (UI) đều là các lớp con của lớp **View**, do đó kế thừa nhiều thuộc tính từ lớp **View**.
- Các phần tử **View** có thể được nhóm lại bên trong một **ViewGroup**, hoạt động như một container. Mỗi quan hệ này là mối quan hệ cha-con, trong đó **parent** là một **ViewGroup**, còn **child** là một **View** hoặc một **ViewGroup** khác.
- Phương thức **onCreate()** được sử dụng để **inflate layout**, nghĩa là thiết lập nội dung hiển thị của màn hình từ tệp XML layout. Bạn cũng có thể sử dụng nó để lấy tham chiếu đến các phần tử giao diện người dùng (UI) khác trong layout.
- Một **View**, giống như một chuỗi ký tự (string), là một tài nguyên có thể có ID. Phương thức **findViewById** nhận ID của một View làm tham số và trả về View đó.

### **Sử dụng trình chỉnh sửa bố cục:**

- Nhấp vào tab **Design** để thao tác các phần tử và bố cục, hoặc tab **Text** để chỉnh sửa mã XML của bố cục.
- Trong tab **Design**, bảng **Palettes** hiển thị các phần tử giao diện người dùng (UI) có thể sử dụng trong bố cục ứng dụng, và bảng **Component tree** hiển thị cây phân cấp của các phần tử UI.
- Các phần tử UI trong bố cục được hiển thị trong các bảng **design** và **blueprint**. Tab **Attributes** hiển thị bảng **Attributes** để cài đặt các thuộc tính cho phần tử UI.
- Constraint handle: Nhấp vào một constraint handle (vòng tròn ở mỗi cạnh của phần tử), kéo đến một constraint khác hoặc đường viền cha để tạo ràng buộc. Ràng buộc được hiển thị bằng một đường gấp khúc.

- Resizing handle: Kéo tay nắm (hình vuông) để thay đổi kích thước phần tử. Trong khi kéo, tay nắm sẽ chuyển thành góc xiên.
- Autoconnect tool: Khi được bật, công cụ này tự động tạo hai hoặc nhiều ràng buộc cho phần tử UI với bố cục cha, dựa trên vị trí của phần tử.
- Bạn có thể xóa ràng buộc của một phần tử bằng cách chọn phần tử, di chuột qua để hiển thị nút **Clear Constraints**, và nhấp để xóa tất cả các ràng buộc. Để xóa một ràng buộc cụ thể, nhấp vào constraint handle tương ứng.
- Bảng **Attributes** cung cấp truy cập vào tất cả các thuộc tính XML có thể gán cho một phần tử UI. Nó cũng bao gồm một bảng định cỡ hình vuông gọi là **view inspector** ở trên cùng. Các biểu tượng trong hình vuông đại diện cho cài đặt chiều cao và chiều rộng.

Thuộc tính **layout\_width** và **layout\_height** thay đổi khi bạn thay đổi chiều cao và chiều rộng trong bảng điều khiển. Các giá trị có thể là:

1. **match\_constraint**: Phần tử mở rộng để lấp đầy không gian bố cục cha (có tính đến lề nếu được đặt).
2. **wrap\_content**: Phần tử co lại vừa với nội dung của nó. Nếu không có nội dung, phần tử sẽ trở nên vô hình.
3. **dp cố định**: Chỉ định kích thước cố định, phù hợp với kích thước màn hình của thiết bị.

## Trích xuất tài nguyên chuỗi (String Resources):

Thay vì mã hóa cứng chuỗi, nên sử dụng tài nguyên chuỗi, đại diện cho các chuỗi. Thực hiện theo các bước sau:

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

## Xử lý sự kiện nhấp (Handling Clicks):

- **Click handler** là một phương thức được gọi khi người dùng nhấp hoặc chạm vào một phần tử UI.

- Để chỉ định một click handler cho một phần tử UI như **Button**, nhập tên phương thức trong trường **onClick** của bảng **Attributes** ở tab **Design**, hoặc trong trình chỉnh sửa XML bằng cách thêm thuộc tính android:onClick vào phần tử **Button**.
- Tạo click handler trong Activity chính bằng cách sử dụng tham số **View**. Ví dụ:

```
public void showToast(View view) {
    // Xử lý khi nhấp
}
```

- Có thể tìm thấy thông tin về tất cả các thuộc tính của **Button** trong tài liệu lớp **Button**, và tất cả các thuộc tính của **TextView** trong tài liệu lớp **TextView**.

Toast cung cấp cách hiển thị một thông báo đơn giản trong một cửa sổ popup nhỏ. Nó chỉ chiếm không gian đủ để chứa thông báo. Để tạo một instance của Toast, thực hiện các bước sau:

- Gọi phương thức **makeText()** từ lớp **Toast**.
- Cung cấp **context** của **Activity** ứng dụng và thông báo cần hiển thị (chẳng hạn như một tài nguyên chuỗi).
- Cung cấp thời lượng hiển thị, ví dụ: **Toast.LENGTH\_SHORT** cho thời gian ngắn. Thời lượng có thể là **Toast.LENGTH\_LONG** hoặc **Toast.LENGTH\_SHORT**.
- Hiển thị Toast bằng cách gọi phương thức **show()**.

```
Toast.makeText(context, R.string.message_text, Toast.LENGTH_SHORT).show();
```

### 1.3) Trình chỉnh sửa bối cảnh

#### Giới thiệu

Như đã học trong **1.2 Phần A: Giao diện tương tác đầu tiên**, bạn có thể xây dựng giao diện người dùng (UI) bằng **ConstraintLayout** trong trình chỉnh sửa bối cảnh. **ConstraintLayout** sắp xếp các phần tử UI bằng cách kết nối ràng buộc với các phần

tử khác hoặc với các cạnh của bố cục. ConstraintLayout được thiết kế để dễ dàng kéo thả các phần tử UI vào trình chỉnh sửa bố cục.

**ConstraintLayout** là một **ViewGroup**, một loại **View** đặc biệt có thể chứa các đối tượng **View** khác (được gọi là **children** hoặc **child views**). Phần thực hành này giới thiệu nhiều tính năng hơn 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 dọc.
- **RelativeLayout:** Một nhóm các phần tử **View** con, trong đó mỗi phần tử **View** được định vị và căn chỉnh tương đối với các phần tử **View** khác bên trong **ViewGroup**. Vị trí của các phần tử **View** con được mô tả dựa trên mối quan hệ giữa chúng với nhau hoặc với **ViewGroup** cha.

### Những gì bạn cần biết trước:

Bạn cần có khả năng:

- Tạo một ứng dụng **Hello World** với Android Studio.
- Chạy ứng dụng trên trình giả lập hoặc thiết bị thực.
- Tạo một 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 một biến thể bố cục cho màn hình ngang (landscape).
- Cách tạo một 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 gói (pack) và căn chỉnh (align) để căn chỉnh các phần tử trong bố cục.
- Cách định vị các view trong **LinearLayout**.
- Cách định vị các view trong **RelativeLayout**.

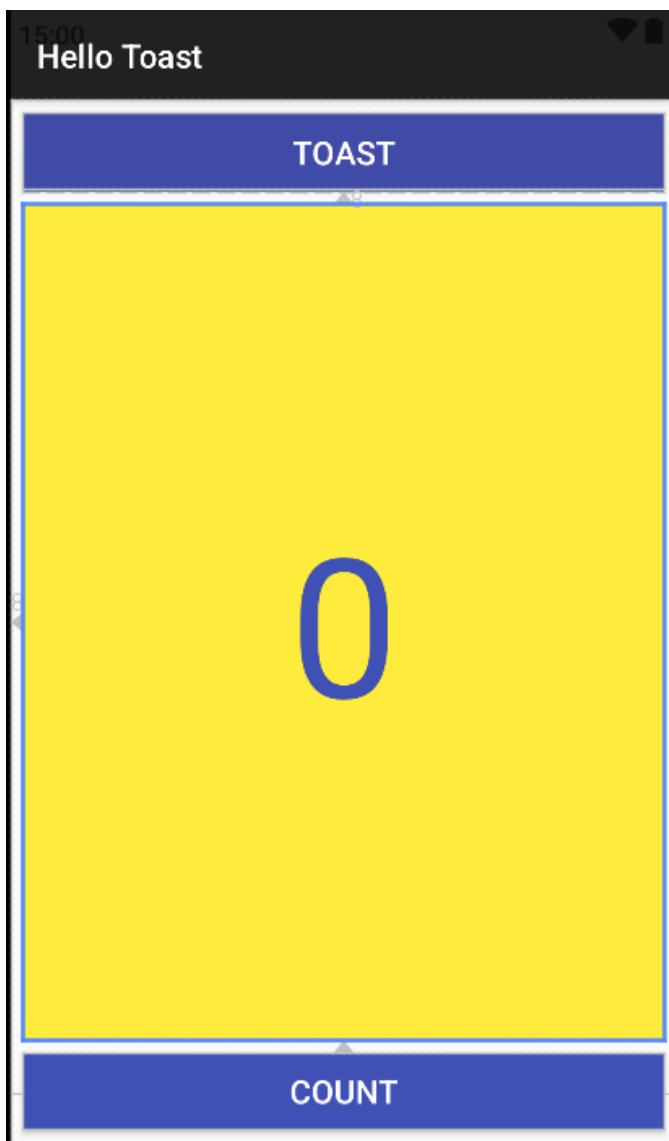
### Những gì bạn sẽ làm:

- Tạo một biến thể bố cục cho màn hình ngang.
- Tạo một biến thể bố cục cho máy tính bảng và màn hình lớn hơn.
- Sửa đổi bố cục để thêm các ràng buộc vào các phần tử UI.

- Sử dụng ràng buộc đường cơ sở của **ConstraintLayout** để căn chỉnh các phần tử với văn bản.
- Sử dụng các nút gói và căn chỉnh của **ConstraintLayout** để căn chỉnh các phần tử.
- Thay đổi bố cục để sử dụng **LinearLayout**.
- Định vị các phần tử trong **LinearLayout**.
- Thay đổi bố cục để sử dụng **RelativeLayout**.
- Sắp xếp lại các view trong bố cục chính để liên kết tương đối với nhau.

### Tổng quan về ứng dụng:

Ứng dụng **Hello Toast** từ bài học trước sử dụng **ConstraintLayout** để sắp xếp các phần tử UI trong bố cục của Activity, như minh họa trong hình dưới đây.



Để thực hành thêm với **ConstraintLayout**, bạn sẽ tạo một biến thể của bố cục này dành cho màn hình ngang (horizontal orientation), như minh họa trong hình dưới đây.

Bạn cũng sẽ học cách sử dụng ràng buộc đường cơ sở (baseline constraints) và một số tính năng căn chỉnh của **ConstraintLayout** bằng cách tạo một biến thể bố cục khác dành cho màn hình máy tính bảng.

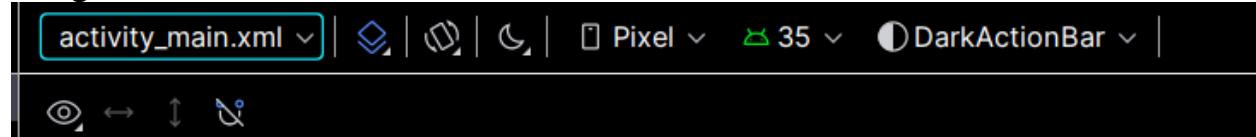
Bạn cũng sẽ tìm hiểu về các lớp con khác của **ViewGroup** như **LinearLayout** và **RelativeLayout**, đồng thời thay đổi bố cục của ứng dụng **Hello Toast** để sử dụng chúng.

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

Trong bài học trước, thử thách lập trình yêu cầu bạn thay đổi bố cục của ứng dụng **Hello Toast** để phù hợp với cả hai chế độ **dọc** và **ngang**. Trong nhiệm vụ này, bạn sẽ học cách dễ dàng hơn để tạo các biến thể bố cục cho các thiết bị di động ở chế độ **dọc** hoặc **ngang**, cũng như cho các thiết bị màn hình lớn như máy tính bảng.

Trong nhiệm vụ này, bạn sẽ sử dụng một số nút trên **hai thanh công cụ** trên cùng trong trình chỉnh sửa bố cục (**Layout Editor**).

Thanh công cụ trên cùng giúp bạn cấu hình cách hiển thị bản xem trước của bố cục trong trình chỉnh sửa bố cục:



#### Các bước chính:

1. **Chọn bề mặt thiết kế (Design Surface):** Chọn **Design** để hiển thị bản xem trước màu sắc của bố cục, hoặc **Blueprint** để chỉ hiển thị các đường viền của từng phần tử giao diện người dùng (UI). Để xem cả hai chế độ cạnh nhau, chọn **Design + Blueprint**.
2. **Thay đổi hướng hiển thị trong trình chỉnh sửa (Orientation in Editor):** Chọn **Portrait** (chế độ dọc) hoặc **Landscape** (chế độ ngang) để xem trước bố cục theo chiều dọc hoặc ngang. Điều này rất 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 cho từng hướng, chọn **Create Landscape Variation** hoặc các biến thể khác.

3. **Chọn thiết bị trong trình chỉnh sửa (Device in Editor):** Chọn loại thiết bị (ví dụ: điện thoại/máy tính bảng, Android TV hoặc Android Wear).
4. **Chọn phiên bản API trong trình chỉnh sửa (API Version in Editor):** Chọn phiên bản Android để hiển thị bản xem trước.
5. **Chọn chủ đề trong trình chỉnh sửa (Theme in Editor):** Chọn một chủ đề (ví dụ: AppTheme) để áp dụng cho bản xem trước.
6. **Chọn ngôn ngữ và khu vực (Locale in Editor):** Chọn ngôn ngữ và khu vực cho bản xem trước. Danh sách này chỉ hiển thị các ngôn ngữ có trong tài nguyên chuỗi văn bản (**string resources**). Bạn cũng có thể chọn **Preview as Right To Left** để xem bố cục như khi chọn một ngôn ngữ đọc từ phải sang trái (RTL).

Thanh công cụ thứ hai giúp bạn cấu hình cách hiển thị các phần tử giao diện người dùng trong **ConstraintLayout**, cũng như phóng to hoặc cuộn bản xem trước.



#### Trong hình trên:

1. **Hiển thị (Show):** Chọn **Show Constraints** và **Show Margins** để hiển thị hoặc ẩn các ràng buộc và khoảng cách trong bản xem trước.
2. **Autoconnect:** Bật hoặc tắt tính năng Autoconnect. Khi tính năng này được bật, bạn có thể kéo bất kỳ phần tử nào (như Button) đến bất kỳ phần nào trong bố cục để tự động tạo ràng buộc với bố cục cha.
3. **Xóa tất cả ràng buộc (Clear All Constraints):** Xóa toàn bộ ràng buộc trong bố cục.
4. **Suy luận ràng buộc (Infer Constraints):** Tạo ràng buộc bằng cách suy luận.

5. **Khoảng cách mặc định (Default Margins):** Thiết lập khoảng cách mặc định.
6. **Gói (Pack):** Gom nhóm hoặc mở rộng các phần tử được chọn.
7. **Căn chỉnh (Align):** Căn chỉnh các phần tử được chọn.
8. **Đường dẫn hướng (Guidelines):** Thêm đường dẫn hướng dọc hoặc ngang.
9. **Điều khiển phóng to/thu nhỏ (Zoom/pan controls):** Phóng to hoặc thu nhỏ bản xem trước.

## 1.1 Xem trước bố cục ở chế độ ngang

Để xem trước bố cục ứng dụng **Hello Toast** với chế độ ngang, hãy làm theo các bước sau:

1. Mở ứng dụng **Hello Toast** từ bài học trước.
  - **Lưu ý:** Nếu bạn đã tải xuống mã hoàn chỉnh cho Hello Toast, hãy xóa các bố cục chế độ ngang (**landscape**) và màn hình cực lớn (**extra-large**) đã hoàn thành mà bạn sẽ tạo trong nhiệm vụ này.
  - Chuyển từ chế độ hiển thị **Project > Android** sang **Project > Project Files** trong bảng **Project**. Sau đó, mở rộng đường dẫn app > src/main > res, chọn cả hai thư mục **layout-land** và **layout-xlarge**, sau đó chọn **Edit > Delete**.
  - Sau khi hoàn tất, chuyển lại bảng **Project** về **Project > Android**.
2. Mở tệp **activity\_main.xml** trong thư mục bố cục (**layout**). Nhập vào tab **Design** nếu nó chưa được chọn.
3. Nhấp vào nút **Orientation in Editor** trên thanh công cụ ở phía trên để thay đổi hướng hiển thị.
4. Trong menu thả xuống, chọn **Switch to Landscape**. Bố cục sẽ xuất hiện ở chế độ ngang, như minh họa bên dưới. Để quay lại chế độ dọc, chọn **Switch to Portrait**.



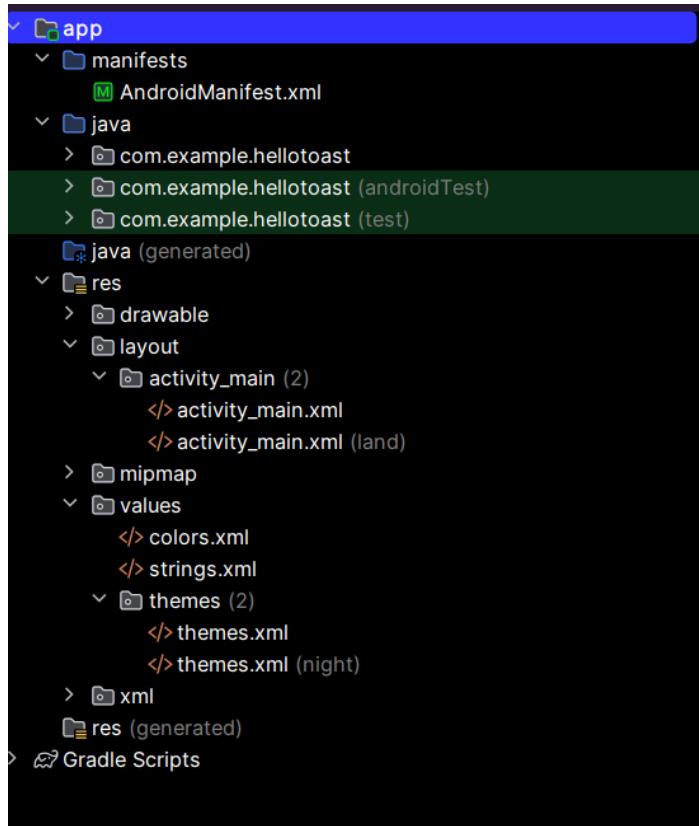
## 1.2 Tạo biến thể bố cục cho chế độ ngang

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.

3. 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)**.



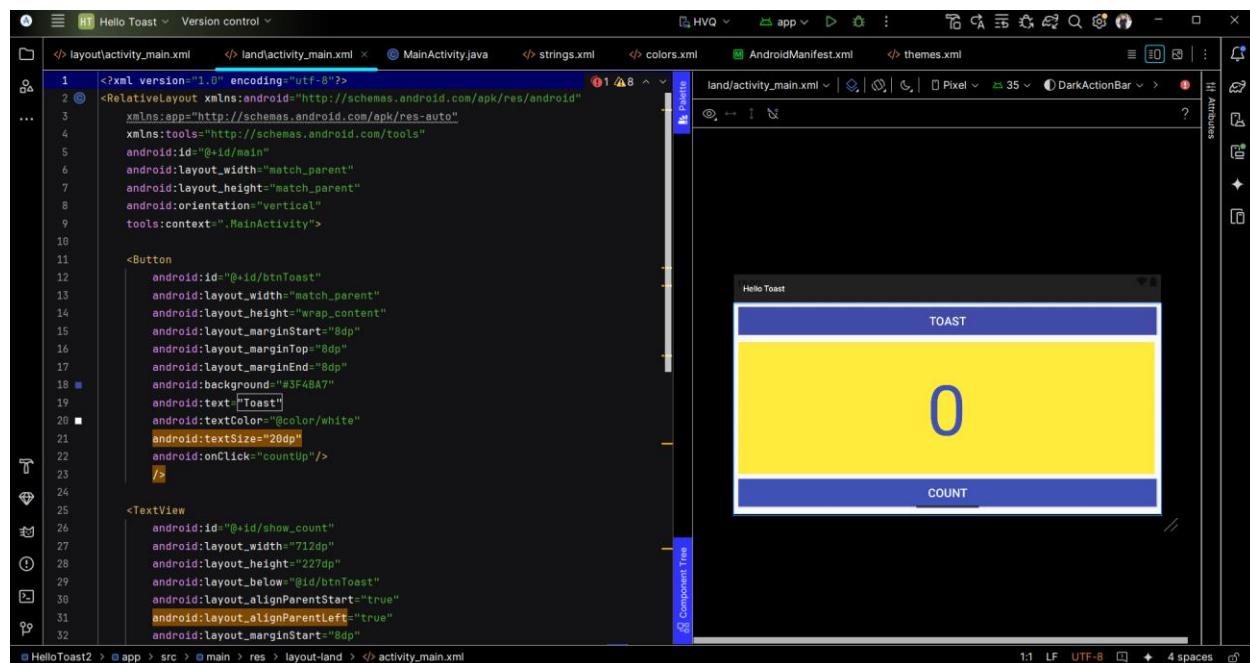
## 1.3 Xem trước bố cục trên các thiết bị khác nhau

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 nhập đúp vào tệp **activity\_main.xml (land)** trong thư mục **layout**.
2. Nhập 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, như minh họa trong hình bên dưới.



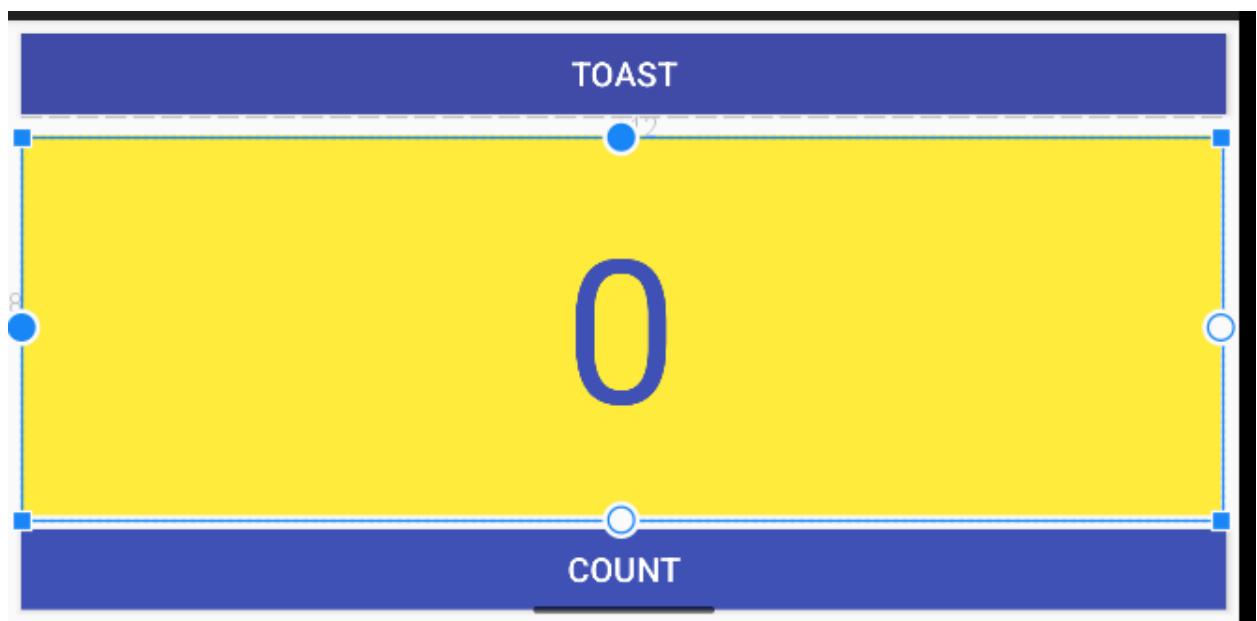
## Thay đổi bố cục với các bước chi tiết

## Hình minh họa:

1. **Tab Preview:** Sử dụng để hiển thị bảng xem trước bố cục.
2. **Preview Pane:** Hiển thị cách bố cục trông trên thiết bị.
3. **XML Code:** Phần mã nguồn của bố cục trong tab **Text**.

## Các bước thay đổi bố cục:

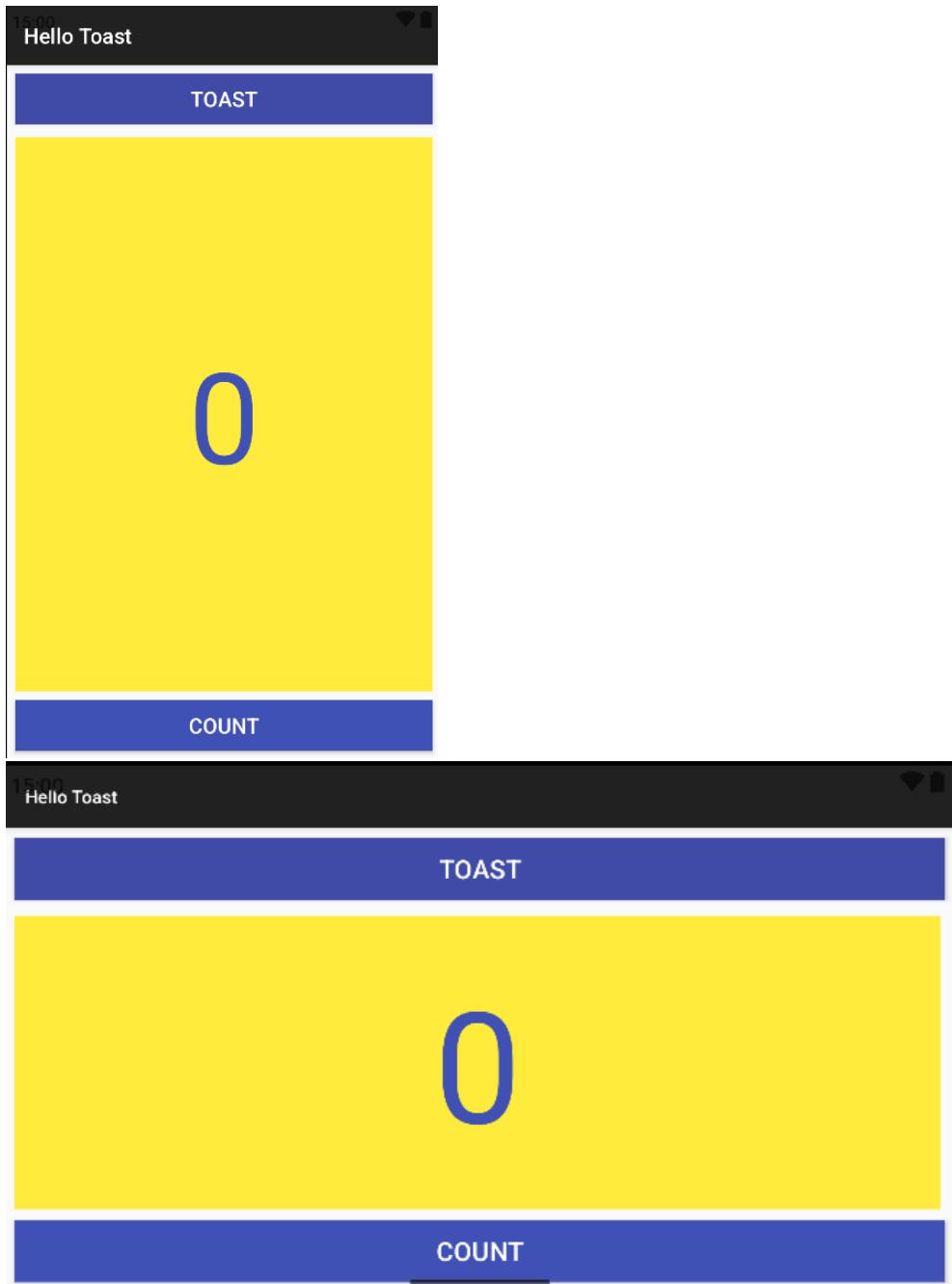
1. Đảm bảo tab **land/activity\_main.xml** vẫn mở trong trình chỉnh sửa bố cục. Nếu không, nhấp đúp vào tệp **activity\_main.xml (land)** trong thư mục **layout**.
2. Chuyển sang tab **Text** và bật tab **Preview** (nếu chưa được chọn).
3. Trong mã XML, tìm phần tử **TextView**.
4. Thay đổi thuộc tính `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.

### Lưu ý trong trình chỉnh sửa:

- Tab **land/activity\_main.xml** hiển thị bố cục cho chế độ ngang.
  - Tab **activity\_main.xml** hiển thị bố cục không thay đổi cho chế độ dọc. Bạn có thể chuyển đổi qua lại giữa hai tab để so sánh.
6. Chạy ứng dụng trên trình giả lập hoặc thiết bị thực. Chuyển hướng hiển thị từ dọc sang ngang để xem cách cả hai bố cục hoạt động.



## 1.5 Tạo một biến thể bố cục cho máy tính bảng

Như bạn đã học trước đây, bạn có thể xem trước bố cục trên các thiết bị khác nhau bằng cách nhấp vào nút **Device in Editor** trên thanh công cụ phía trên. Nếu bạn chọn một thiết bị như **Nexus 10** (máy tính bảng) từ menu, bạn sẽ nhận thấy rằng bố cục không phù hợp cho màn hình máy tính bảng lớn - văn bản của mỗi nút (**Button**) quá nhỏ và cách sắp xếp các nút (**Button**) ở trên và dưới không lý tưởng cho màn hình lớn của máy tính bảng.

Để điều chỉnh bố cục phù hợp với máy tính bảng mà không ảnh hưởng đến bố cục cho các thiết bị điện thoại ở chế độ ngang hoặc dọc, hãy thực hiện các bước sau:

1. Nhấp vào tab **Design** (nếu chưa được chọn) để hiển thị các bảng thiết kế và bản vẽ.
2. Nhấp vào nút **Orientation in Editor** trên thanh công cụ phía trên.
3. Chọn **Create layout x-large Variation**.

Sau khi thực hiện:

- Một cửa sổ chỉnh sửa mới sẽ mở ra với tab **xlarge/activity\_main.xml**, hiển thị bố cục dành riêng cho thiết bị có kích thước màn hình máy tính bảng.
- Trình chỉnh sửa cũng tự động chọn một thiết bị máy tính bảng, chẳng hạn như **Nexus 9** hoặc **Nexus 10**, để hiển thị bản xem trước.

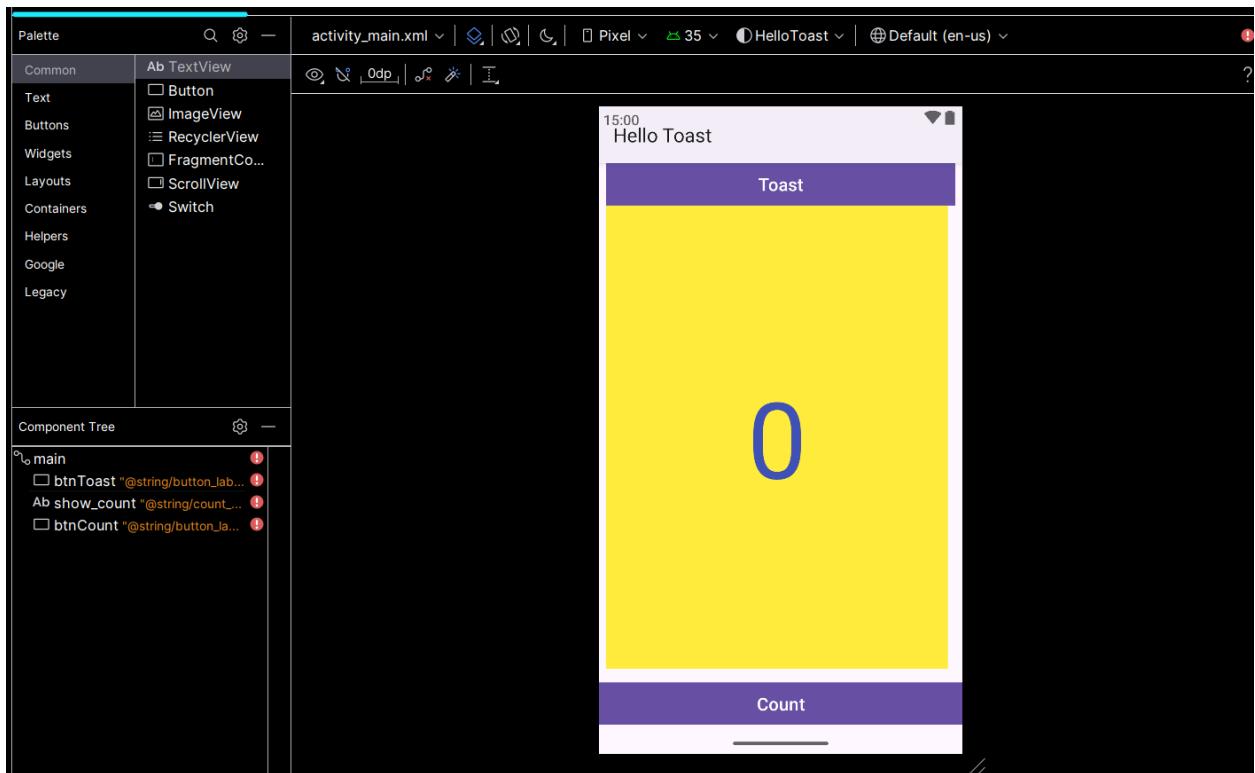
Bạn có thể thay đổi bố cục này, dành riêng cho máy tính bảng, mà không làm thay đổi các bố cục khác.

## 1.6 Thay đổi biến thể bố cục cho máy tính bảng

Bạn có thể sử dụng bảng thuộc tính (Attributes pane) trong tab **Design** để thay đổi các thuộc tính cho bố cục này.

1. **Tắt công cụ Autoconnect** trên thanh công cụ. Ở bước này, hãy đảm bảo rằng công cụ đã bị vô hiệu hóa.
2. Xóa tất cả các ràng buộc (constraints) trong bố cục bằng cách nhấp vào nút **Clear All Constraints** trên thanh công cụ.
  - Khi các ràng buộc đã bị xóa, bạn có thể tự do di chuyển và thay đổi kích thước các phần tử trên bố cục.

3. Trình chỉnh sửa bố cục cung cấp các tay cầm (handles) ở bốn góc của mỗi phần tử để thay đổi kích thước chúng. Trong **Component Tree**, chọn **TextView** có tên **show\_count**. Để dời **TextView** ra ngoài đường đi, giúp bạn dễ dàng kéo các phần tử **Button**, hãy kéo một góc của nó để thay đổi kích thước, như được minh họa trong hình động dưới đây.



Khi thay đổi kích thước một phần tử, nó sẽ cố định các giá trị chiều rộng và chiều cao. Tránh cố định kích thước cho hầu hết các phần tử, vì điều này có thể dẫn đến giao diện không nhất quán trên các màn hình có kích thước và mật độ khác nhau. Trong bước này, việc thay đổi kích thước chỉ được thực hiện để di chuyển phần tử tạm thời, và các kích thước sẽ được điều chỉnh lại trong bước khác.

#### Bước 4:

Chọn nút **button\_toast** trong **Component Tree**, nhấp vào tab **Attributes** để mở bảng thuộc tính, và thay đổi **textSize** thành **60sp** (#1 trong hình bên dưới), **layout\_width** thành **wrap\_content** (#2 trong hình bên dưới)

<b>id</b>	btnToast	+	-
<b>Declared Attributes</b>			
layout_width	match_parent	↴	0
layout_height	wrap_content	↴	0
background	#3F4BA7	0	
id	btnToast		
onClick	countUp	↴	0
text	@string/button_label_to...	0	
textColor	@color/white	0	
textSize	60sp	↴	0

Như được hiển thị ở phía bên phải của hình trên (2), bạn có thể nhấp vào điều khiển chiều rộng của trình kiểm tra chế độ xem, xuất hiện dưới dạng hai đoạn ở hai bên trái và phải của hình vuông, cho đến khi nó hiển thị **Wrap Content**. Ngoài ra, bạn có thể chọn **wrap\_content** từ menu **layout\_width**.

Bạn sử dụng **wrap\_content** để nếu văn bản của **Button** được bắn địa hóa sang một ngôn ngữ khác, nút **Button** sẽ hiển thị rộng hơn hoặc hẹp hơn để phù hợp với từ trong ngôn ngữ khác.

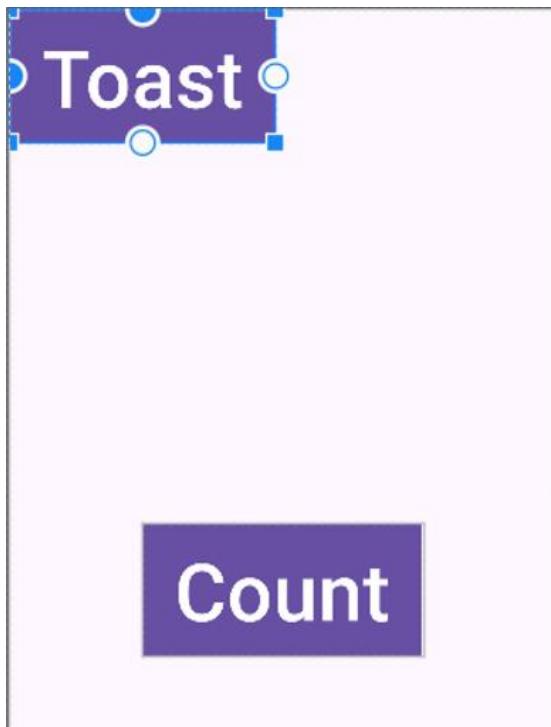
- Chọn nút **button\_count** trong **Component Tree**, thay đổi **textSize** thành **60sp** và **layout\_width** thành **wrap\_content**, sau đó kéo nút **Button** lên trên **TextView** vào một không gian trống trong bố cục.

## 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.

- 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.



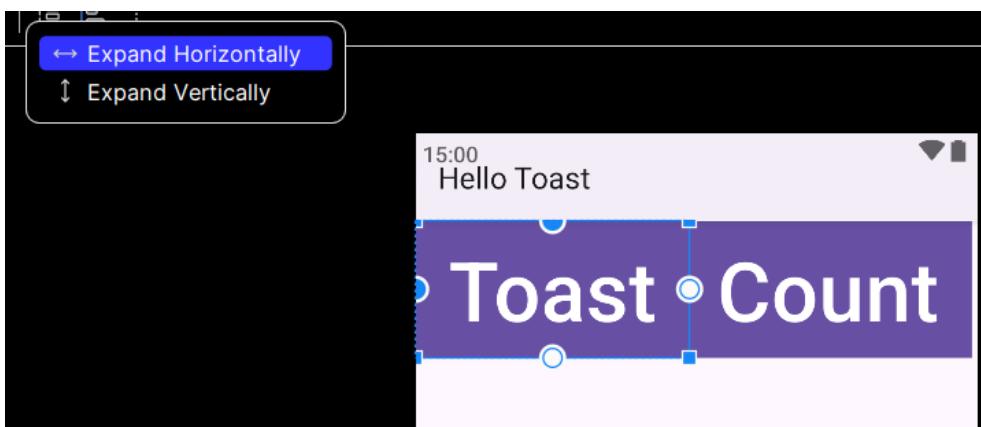
2. **Sử dụng ràng buộc đường cơ sở (baseline constraint)**, bạn có thể ràng buộc nút **button\_count** sao cho đường cơ sở văn bản của nó khớp với đường cơ sở văn bản của nút **button\_toast**. Chọn phần tử **button\_count**, sau đó di chuyển con trỏ chuột qua phần tử cho đến khi nút ràng buộc đường cơ sở xuất hiện bên dưới phần tử.
3. 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.

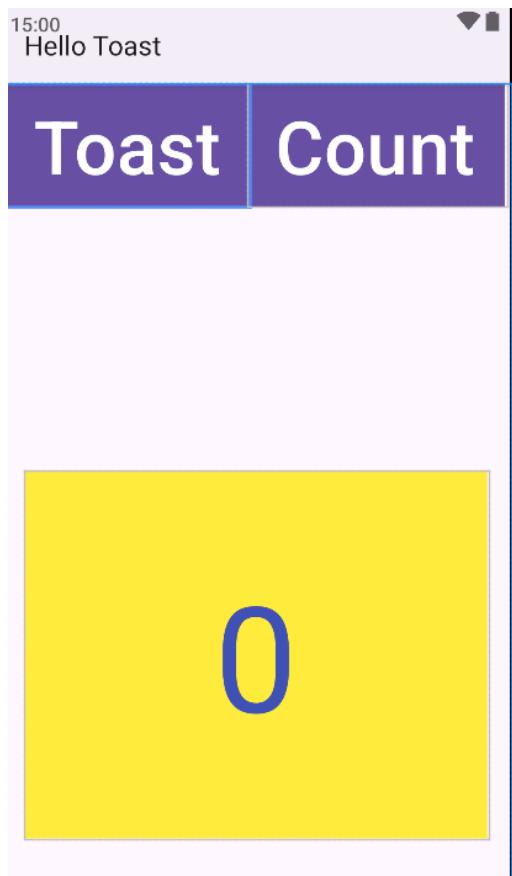
1. Chọn nút **button\_count** trong **Component Tree**, sau đó nhấn giữ **Shift** và chọn thêm nút **button\_toast** để cả hai nút đều được chọn.
2. Nhấp vào nút **pack** trên thanh công cụ, sau đó chọn tùy chọn **Expand Horizontally** như minh họa trong hình.



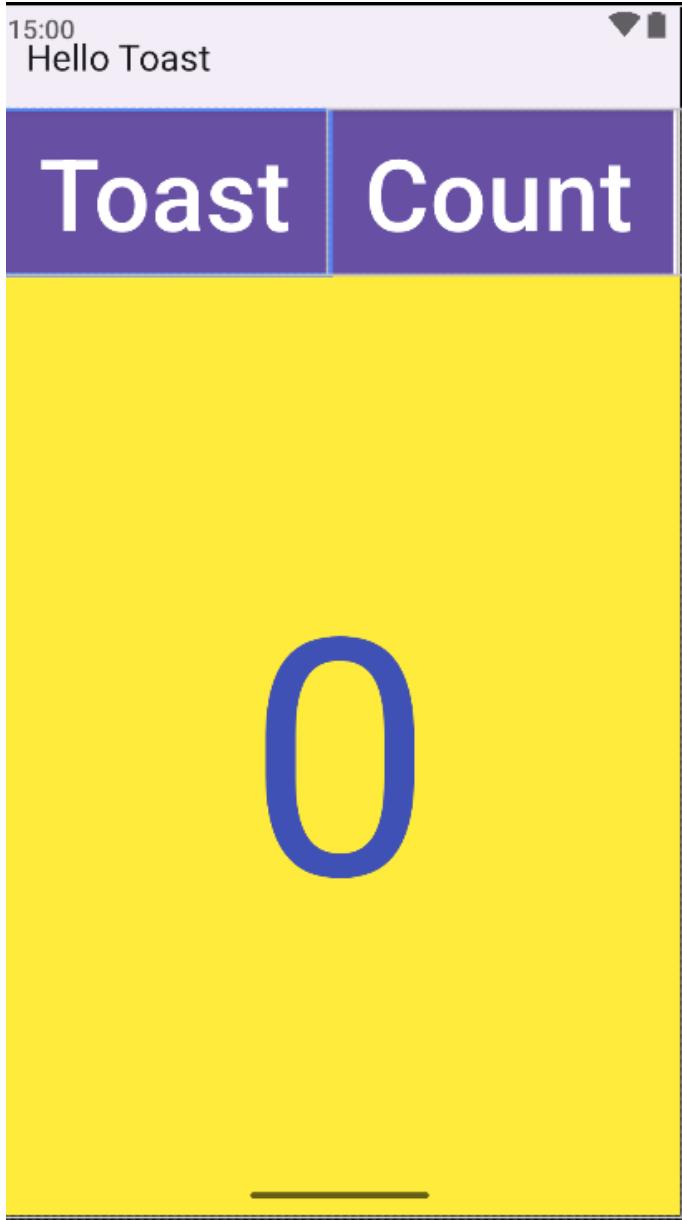
Các phần tử **Button** sẽ mở rộng theo chiều ngang để lấp đầy toàn bộ giao diện như minh họa bên dưới.



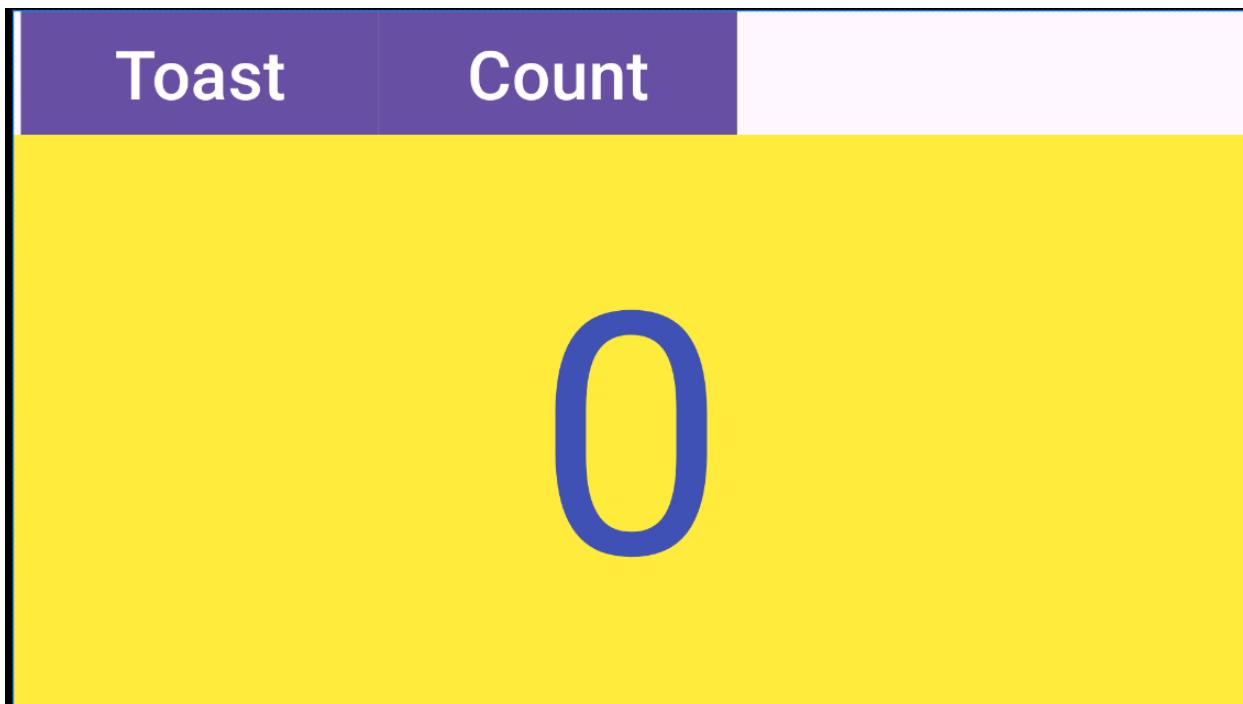
3. Để hoàn thiện bối cảnh, hãy ràng buộc **show\_count TextView** vào phía dưới của nút **button\_toast** và ràng buộc nó với các cạnh bên cũng như cạnh dưới của bối cảnh, như được minh họa trong hình động bên dưới.



4. 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.



5. 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).

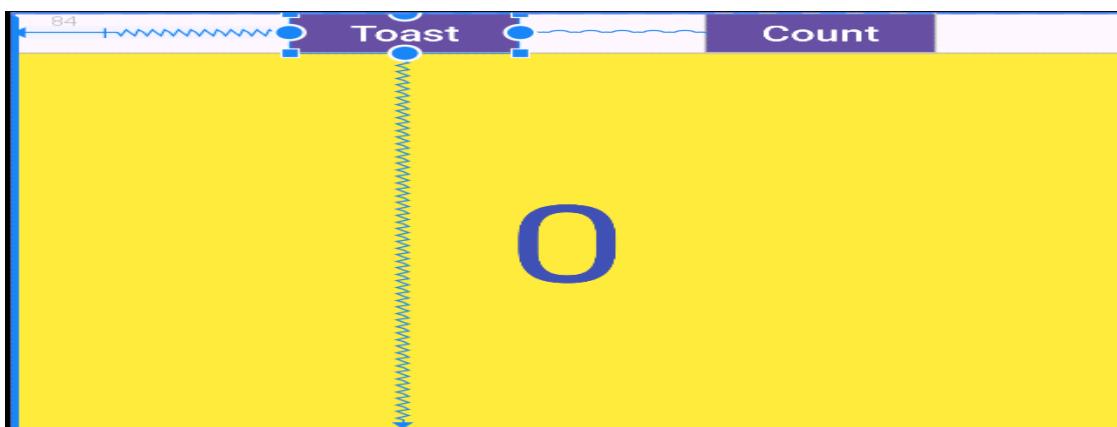


- Chạy ứng dụng trên các trình giả lập khác nhau và thay đổi hướng màn hình sau khi chạy ứng dụng để xem giao diện trông như thế nào trên các loại thiết bị khác nhau. Bạn đã thành công trong việc tạo một ứng dụng có giao diện người dùng (UI) hoạt động đúng trên điện thoại và máy tính bảng với các kích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

Mẹo: Để có hướng dẫn chi tiết về cách sử dụng ConstraintLayout, hãy xem **Using ConstraintLayout to design your views**.

**Thử thách:** Để hỗ trợ giao diện ngang (landscape) cho máy tính bảng, bạn có thể căn giữa các nút (Button) trong tệp activity\_main.xml (xlarge) để chúng xuất hiện như hình minh họa bên dưới.

**Gợi ý:** Chọn các phần tử, nhấp vào nút căn chỉnh trong thanh công cụ, và chọn **Căn giữa theo chiều ngang (Center Horizontally)**.



## Task 2: Thay đổi bố cục thành LinearLayout

LinearLayout là một **ViewGroup** sắp xếp tập hợp các **view** theo hàng ngang hoặc hàng dọc. LinearLayout là một trong những bố cục phổ biến nhất vì nó đơn giản và nhanh. Nó thường được sử dụng trong một **view group** khác để sắp xếp các phần tử giao diện người dùng theo chiều ngang hoặc dọc.

LinearLayout yêu cầu có các thuộc tính sau:

- **layout\_width**
- **layout\_height**
- **orientation**

**layout\_width** và **layout\_height** có thể nhận một trong các giá trị sau:

- **match\_parent**: Mở rộng view để lấp đầy **parent** theo chiều rộng hoặc chiều cao. Khi **LinearLayout** là **root view**, nó sẽ mở rộng theo kích thước của màn hình (**parent view**).
- **wrap\_content**: Thu nhỏ kích thước view sao cho nó vừa đủ chứa nội dung. Nếu không có nội dung, view sẽ trở nên vô hình.
- **Số dp cố định (density-independent pixels)**: Xác định kích thước cố định, được điều chỉnh theo mật độ màn hình của thiết bị. Ví dụ, **16dp** có nghĩa là 16 pixel độc lập với mật độ.

**orientation** có thể là:

- **horizontal**: Các **view** được sắp xếp từ trái sang phải.
- **vertical**: Các **view** được sắp xếp từ trên xuống dưới.

### 2.1 Thay đổi root view group thành LinearLayout

1. Mở ứng dụng **Hello Toast** từ bài trước.
2. Mở tệp **activity\_main.xml** (nếu chưa mở), và nhấp vào tab **Text** ở cuối khung chỉnh sửa để xem mã XML. Ở dòng đầu tiên của mã XML, bạn sẽ thấy dòng sau:

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

3. Thay đổi thẻ **<android.support.constraint.ConstraintLayout** thành **<LinearLayout** để mã XML trông như sau:

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

4. Đảm bảo rằng thẻ đóng ở cuối mã đã được thay đổi thành </LinearLayout> (Android Studio sẽ tự động thay đổi thẻ đóng nếu bạn thay đổi thẻ mở). Nếu nó không thay đổi tự động, hãy chỉnh sửa thủ công.
5. Ngay dưới dòng thẻ <LinearLayout>, thêm thuộc tính sau vào sau thuộc tính android:layout\_height:

```
    android:layout_height="match_parent"  
    android:orientation="vertical"
```

Sau khi thực hiện các thay đổi này, một số thuộc tính XML của các phần tử khác sẽ bị gạch chân màu đỏ vì chúng được sử dụng với **ConstraintLayout** và không phù hợp với **LinearLayout**.

## 2.2 Thay đổi thuộc tính của các phần tử để phù hợp với LinearLayout

Thực hiện các bước sau để thay đổi thuộc tính của các phần tử giao diện người dùng sao cho chúng hoạt động với **LinearLayout**:

1. Mở ứng dụng **Hello Toast** từ bài trước.
2. Mở tệp **activity\_main.xml** (nếu chưa mở), và nhấp vào tab **Text**.
3. Tìm phần tử **Button** có **id** là **button\_toast**, và thay đổi thuộc tính sau:

```
    android:id="@+id/btnToast"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"
```

4. Xóa các thuộc tính sau khỏi phần tử **button\_toast**.

```
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

5. Tìm phần tử **Button** có **id** là **button\_count**, và thay đổi thuộc tính sau.

```
    android:id="@+id/btnCount"  
    android:layout_width="match_parent"
```

6. Xóa các thuộc tính sau khỏi phần tử **button\_count**.

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent" />
```

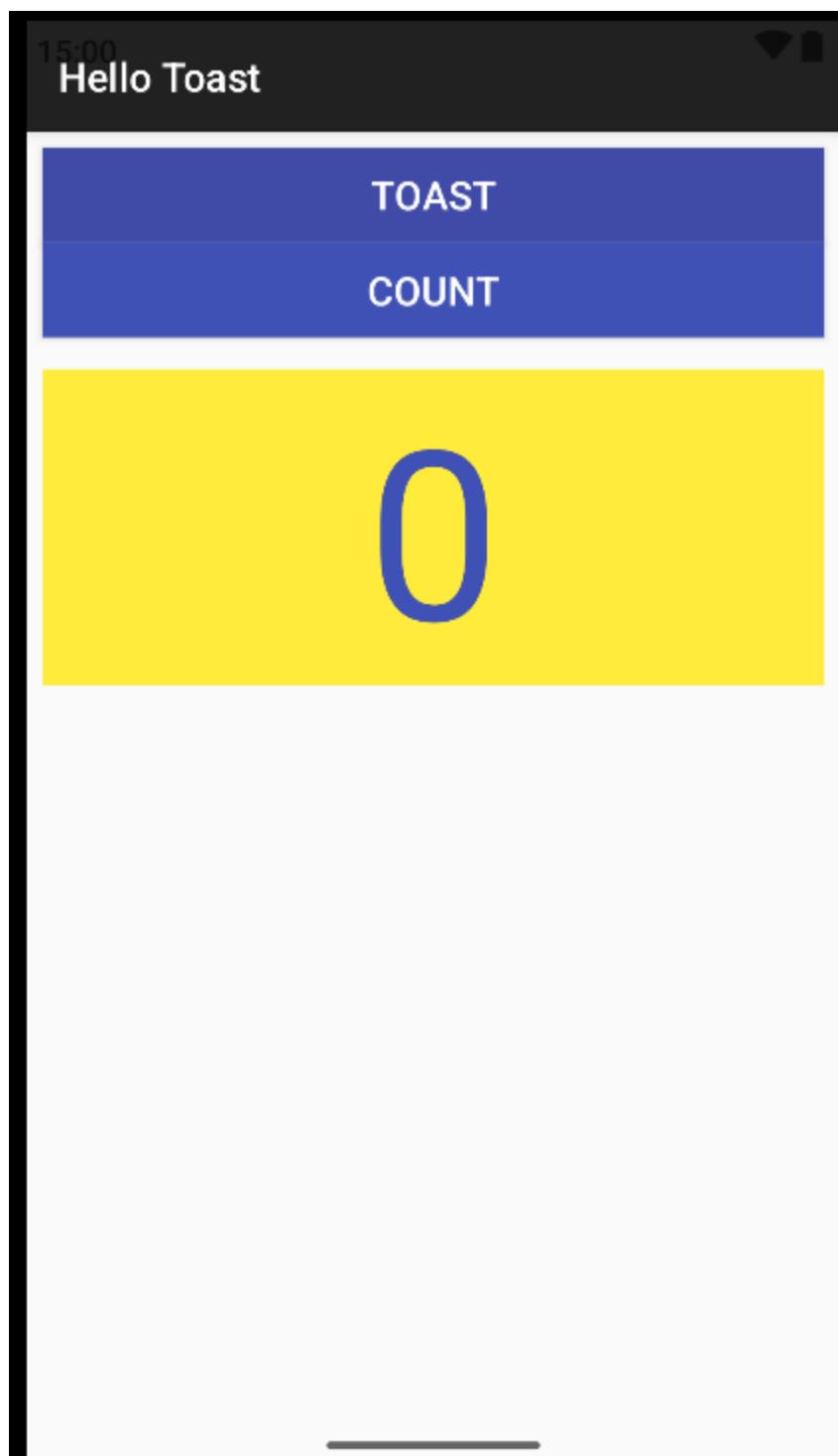
7. Tìm phần tử **TextView** có **id** là **show\_count**, và thay đổi các thuộc tính sau.

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

8. Xóa các thuộc tính sau khỏi phần tử **show\_count**.

```
app:layout_constraintBottom_toTopOf="@+id/btnCount"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/btnToast"
```

9. Nhập vào tab **Preview** ở phía bên phải cửa sổ Android Studio (nếu chưa được chọn) để xem trước bố cục hiện tại.



## 2.3 Thay đổi vị trí của các phần tử trong LinearLayout

LinearLayout sắp xếp các phần tử của nó theo một hàng ngang hoặc dọc. Bạn đã thêm thuộc tính `android:orientation="vertical"` cho LinearLayout, vì vậy các phần tử được xếp chồng lên nhau theo chiều dọc, như đã hiển thị trong hình trước đó.

Để thay đổi vị trí của chúng sao cho nút **Count** nằm ở phía dưới, hãy làm theo các bước sau:

1. Mở ứng dụng **Hello Toast** từ bài trước.
2. Mở tệp bố cục **activity\_main.xml** (nếu chưa mở), sau đó nhấp vào tab **Text**.
3. Chọn nút **button\_count** và tất cả các thuộc tính của nó, từ thẻ `<Button` đến hết dấu đóng `/>`, sau đó chọn **Edit > Cut (Cắt)**.
4. Nhấp chuột sau thẻ đóng `/>` của phần tử **TextView**, nhưng trước thẻ đóng `</LinearLayout>`, sau đó chọn **Edit > Paste (Dán)**.
5. (**Tùy chọn**) Để chỉnh sửa lại khoảng cách hoặc thụt dòng cho đẹp mắt, chọn **Code > Reformat Code** để định dạng lại mã XML với khoảng cách và thụt dòng phù hợp.

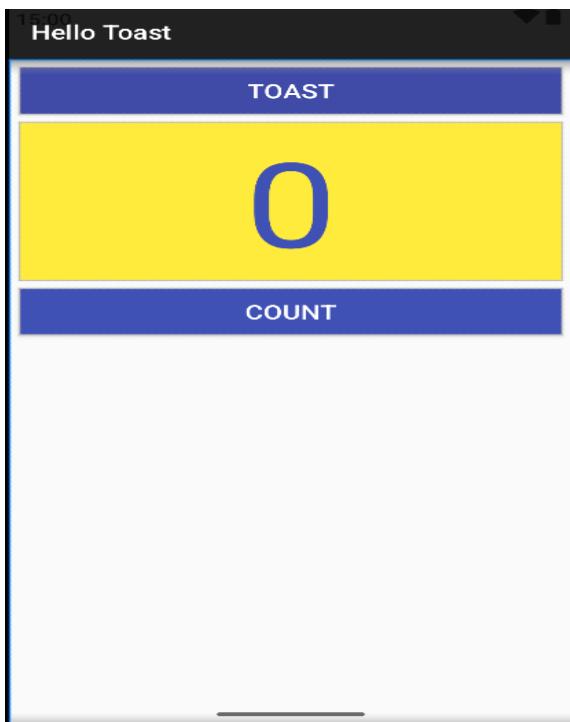
Bây giờ, mã XML cho các phần tử giao diện sẽ trông giống như sau:

```
<Button
    android:id="@+id/btnToast"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:background="#3F4BA7"
    android:text="@string/button_label_toast"
    android:textColor="@color/white"
    android:textSize="20dp"
    />

<TextView
    android:id="@+id/textToast"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
```

```
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:background="#FFEB3B"
    android:gravity="center"
    android:text="@string/count_initial_value"
    android:textAllCaps="true"
    android:textColor="#3F51B5"
    android:textSize="120dp" />
<Button
    android:id="@+id/btnCount"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:background="#3F51B5"
    android:text="Count"
    android:textColor="@color/white"
    android:textSize="20dp"
    />
```

Bằng cách di chuyển nút **button\_count** xuống dưới **TextView**, bố cục giờ đây gần giống với trước đó, với nút **Count** nằm ở phía dưới. Bản xem trước của bố cục bây giờ trông như sau:



## 2.4 Thêm thuộc tính weight cho phần tử TextView

Việc xác định các thuộc tính **gravity** và **weight** giúp bạn kiểm soát tốt hơn cách sắp xếp các **View** và nội dung trong **LinearLayout**.

- Thuộc tính `android:gravity` xác định cách căn chỉnh nội dung bên trong một **View**.  
Ở bài trước, bạn đã đặt thuộc tính này cho **show\_count (TextView)** để căn giữa nội dung (chữ số **0**) trong khung **TextView**.

```
    android:gravity="center"
```

- Thuộc tính `android:layout_weight` xác định lượng không gian thừa trong **LinearLayout** mà **View** đó sẽ chiếm. Nếu chỉ có một **View** có thuộc tính này, nó sẽ chiếm toàn bộ không gian trống. Nếu có nhiều **View** có **weight**, không gian sẽ được chia theo tỷ lệ.
  - Ví dụ: Nếu mỗi nút **Button** có `weight="1"` và **TextView** có `weight="2"`, tổng cộng là **4**, thì mỗi nút **Button** sẽ nhận  $\frac{1}{4}$  không gian, còn **TextView** sẽ nhận  $\frac{1}{2}$  không gian.

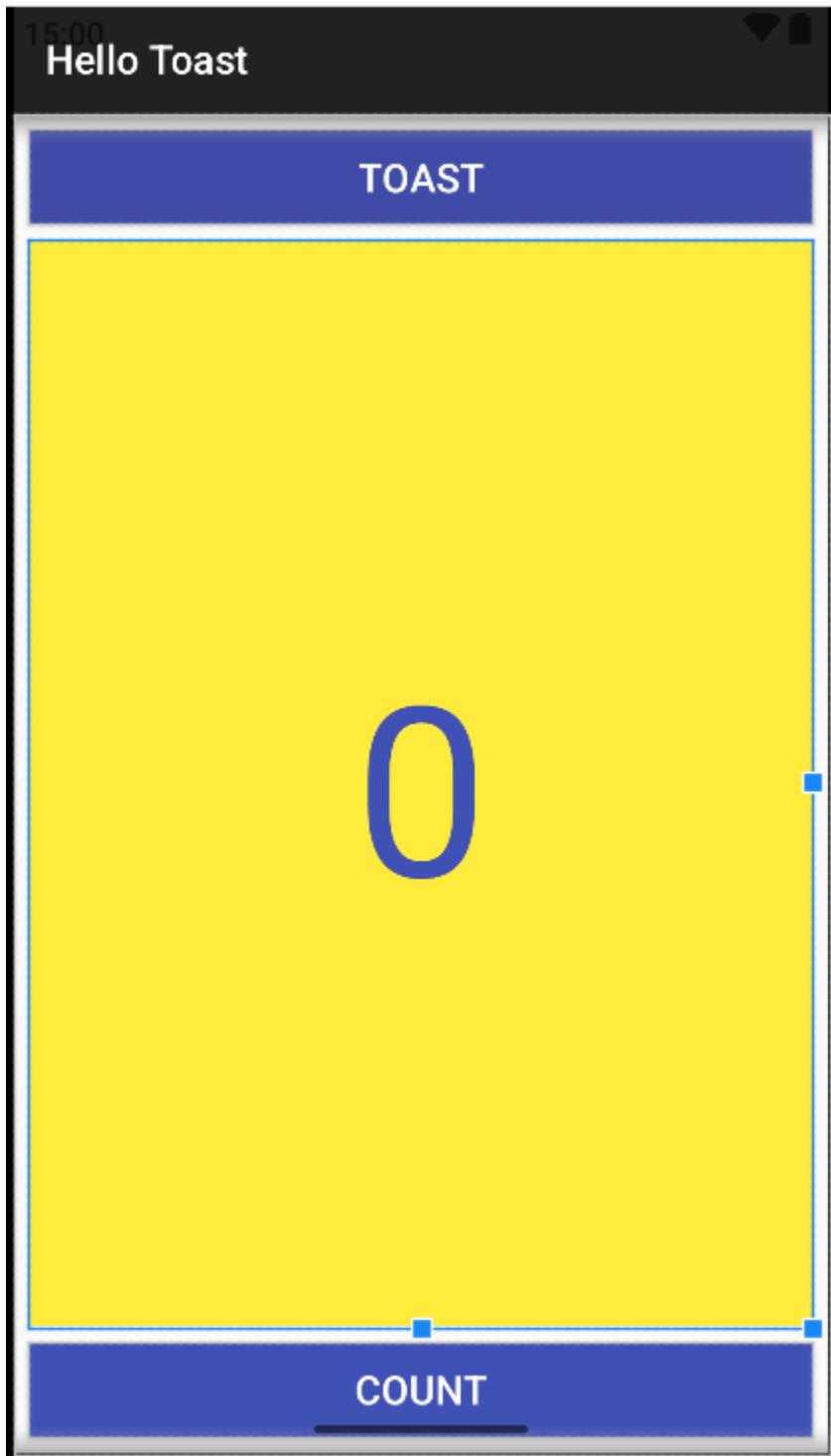
Trên các thiết bị khác nhau, **show\_count (TextView)** có thể chiếm một phần hoặc phần lớn không gian giữa hai nút **Toast** và **Count**. Để đảm bảo **TextView** mở rộng và lấp đầy không gian trống trên mọi thiết bị, hãy thêm thuộc tính `android:layout_weight`.

Các bước thực hiện

- Mở ứng dụng **Hello Toast** từ bài trước.
- Mở tệp bố cục **activity\_main.xml** (nếu chưa mở), sau đó nhấn vào tab **Text**.
- Tìm phần tử **show\_count (TextView)** và thêm thuộc tính sau:

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

Bản xem trước bây giờ trông như hình sau.



Bây giờ, phần tử **show\_count** (**TextView**) sẽ chiếm toàn bộ không gian giữa các nút bấm. Bạn có thể xem trước bố cục trên các thiết bị khác nhau bằng cách nhấn vào nút **Device in Editor** trên thanh công cụ của cửa sổ xem trước và chọn một thiết bị khác.

Dù chọn thiết bị nào, **show\_count** (**TextView**) vẫn sẽ lấp đầy không gian giữa các nút.

Mã giải pháp cho **Task 2**

Mã XML trong **activity\_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id	btnToast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:background="#3F4BA7"
        android:text="@string/button_label_toast"
        android:textColor="@color/white"
        android:textSize="20dp"
    />

    <TextView
        android:id="@+id/textToast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
    />
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:layout_weight="1"
        android:background="#FFEB3B"
        android:gravity="center"
        android:text="@string/count_initial_value"
        android:textAllCaps="true"
        android:textColor="#3F51B5"
        android:textSize="120dp" />

    <Button
        android:id="@+id/btnCount"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#3F51B5"
        android:text="Count"
        android:textColor="@color/white"
        android:textSize="20dp"
    />

</LinearLayout>
```

### Task 3: Thay đổi bố cục sang RelativeLayout

**RelativeLayout** là một nhóm **View** trong đó mỗi **View** được định vị và căn chỉnh dựa trên các **View** khác trong cùng nhóm. Trong nhiệm vụ này, bạn sẽ học cách xây dựng bố cục bằng **RelativeLayout**.

#### 3.1 Thay đổi LinearLayout thành RelativeLayout

Một cách dễ dàng để thay đổi **LinearLayout** thành **RelativeLayout** là chỉnh sửa trực tiếp trong tab **Text** của tệp XML.

1. Mở tệp **activity\_main.xml**, sau đó nhấp vào tab **Text** ở cuối trình chỉnh sửa để xem mã XML.
2. Thay đổi `<LinearLayout` ở đầu tệp thành `<RelativeLayout`, để câu lệnh trông như sau:

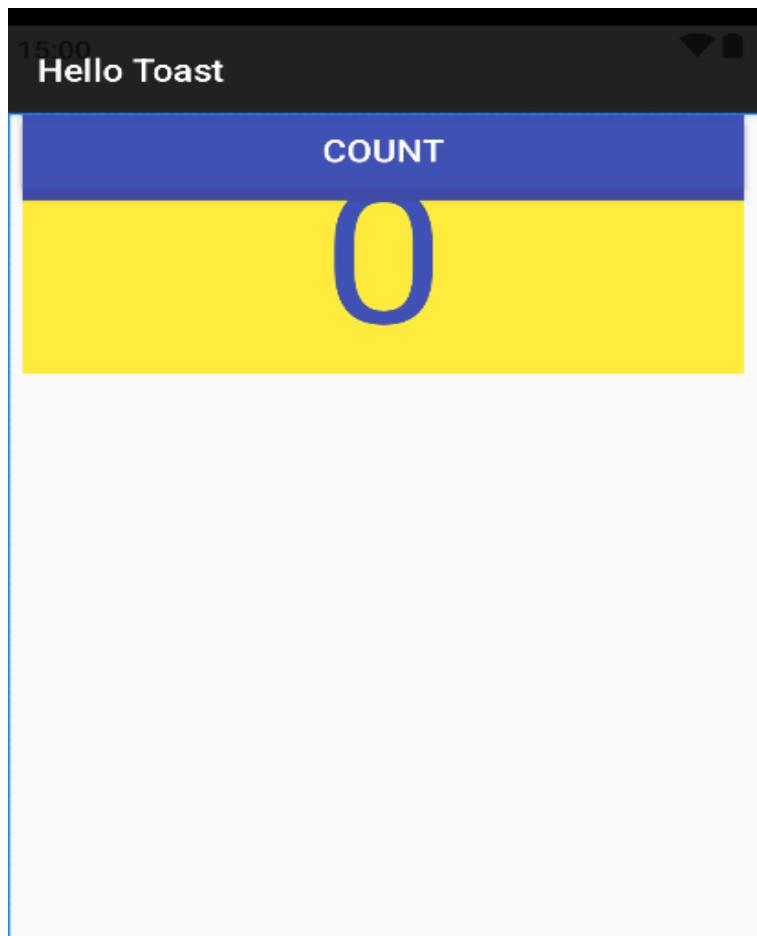
```
|<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

3. Cuộn xuống để đảm bảo thẻ kết thúc `</LinearLayout>` cũng được thay đổi thành `</RelativeLayout>`; nếu chưa, hãy chỉnh sửa thủ công.

### 3.2 Sắp xếp lại các View trong RelativeLayout

Một cách dễ dàng để sắp xếp và định vị các **View** trong **RelativeLayout** là thêm các thuộc tính XML trong tab **Text**.

1. Nhấp vào tab **Preview** bên cạnh trình chỉnh sửa (nếu chưa được chọn) để xem bản xem trước bối cảnh, bây giờ trông giống như hình bên dưới.



**Lưu ý:** Khi thay đổi sang **RelativeLayout**, trình chỉnh sửa bố cục cũng thay đổi một số thuộc tính của **View**. Ví dụ:

- Nút **Count** (button\_count) đè lên nút **Toast** (button\_toast), khiến bạn không nhìn thấy nút **Toast**.
  - Phần trên của **TextView** (show\_count) đè lên các nút **Button**.
2. Thêm thuộc tính android:layout\_below vào nút **button\_count** để đặt nút này ngay bên dưới **TextView** (show\_count). Đây là một trong nhiều thuộc tính giúp định vị **View** trong **RelativeLayout**, cho phép bạn đặt **View** dựa trên **View** khác.

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

3. Thêm thuộc tính android:layout\_centerHorizontal vào **button\_count** để căn giữa nút này theo chiều ngang trong **RelativeLayout** (là nhóm cha của **View** này).

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

→ Mã XML đầy đủ cho nút **button\_count** như sau:

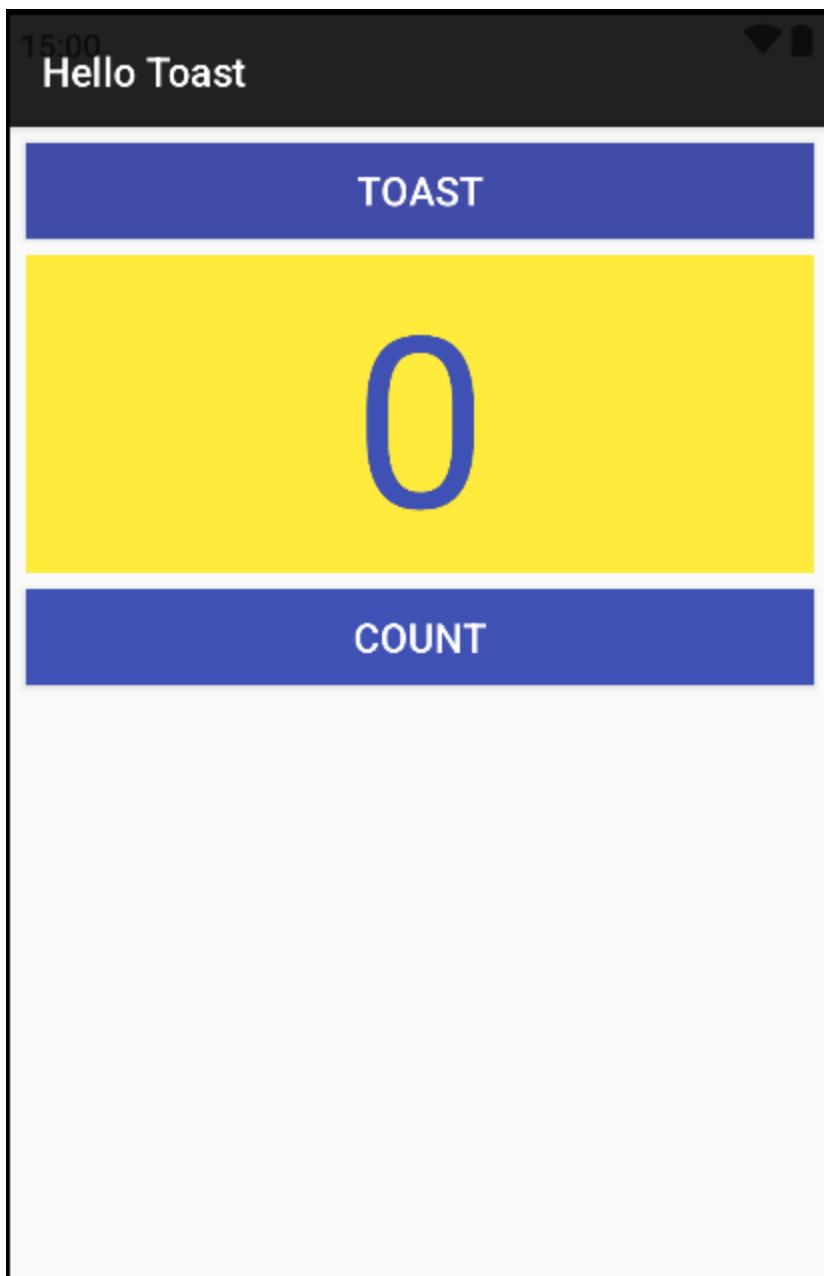
```
<Button  
    android:id="@+id/btnCount"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="8dp"  
    android:layout_marginEnd="8dp"  
    android:layout_marginBottom="8dp"  
    android:background="#3F51B5"  
    android:text="Count"  
    android:textColor="@color/white"  
    android:textSize="20dp"  
    android:layout_below="@+id/show_count"  
    android:layout_centerHorizontal="true"  
/>
```

4. Thêm các thuộc tính sau vào **show\_count** (**TextView**):

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

- android:layout\_alignParentLeft → Căn View về phía **trái** của **RelativeLayout** (nhóm cha).
  - android:layout\_alignParentStart → Căn View theo **cạnh bắt đầu** của nhóm cha.
    - Thuộc tính này giúp hỗ trợ các thiết bị sử dụng ngôn ngữ **phải sang trái (RTL)**.
    - Nếu ứng dụng chạy trên thiết bị có ngôn ngữ **trái sang phải (LTR)**, nó sẽ căn về bên **trái**.
    - Nếu chạy trên thiết bị có ngôn ngữ **phải sang trái (RTL)**, nó sẽ căn về bên **phải**.
5. Xóa thuộc tính `android:layout_weight="1"` của `show_count` (`TextView`), vì thuộc tính này không có tác dụng trong **RelativeLayout**.

Bản xem trước bố cục bây giờ trông giống như hình bên dưới.



**Mẹo:**

**RelativeLayout** giúp bạn dễ dàng sắp xếp nhanh chóng các phần tử UI trong bố cục. Để tìm hiểu thêm về cách định vị **View** trong **RelativeLayout**, hãy tham khảo tài liệu "**Positioning Views**" trong chủ đề "**RelativeLayout**" của **API Guide**.

### **Mã giải pháp cho Task 3**

Mã XML trong **activity\_main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btnToast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:background="#3F4BA7"
        android:text="@string/button_label_toast"
        android:textColor="@color/white"
        android:textSize="20dp"
    />

    <TextView
        android:id="@+id/show_count"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#FFEB3B"
        android:gravity="center"
        android:text="@string/count_initial_value"
        android:textAllCaps="true"
        android:textColor="#3F51B5"
        android:textSize="120dp"
        android:layout_below="@+id/btnToast"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"/>

    <Button
        android:id="@+id/btnCount"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:background="#3F51B5"
        android:text="@string/button_label_count"
        android:textColor="@color/white"
        android:textSize="20dp"
        android:layout_below="@+id/show_count"
        android:layout_centerHorizontal="true"
    />

</RelativeLayout>
```

## Tóm tắt

Sử dụng trình chỉnh sửa bố cục để xem trước và tạo các biến thể

- Để xem trước bố cục ứng dụng theo hướng **ngang**, nhấp vào nút **Orientation in Editor** trên thanh công cụ và chọn **Switch to Landscape**.  
→ Chọn **Switch to Portrait** để quay lại hướng dọc.
- Để tạo một biến thể bố cục khác cho **hướng ngang**, nhấp vào nút **Orientation in Editor** và chọn **Create Landscape Variation**.  
→ Một cửa sổ chỉnh sửa mới sẽ mở với tab **land/activity\_main.xml**, hiển thị bố cục theo hướng ngang.
- Để xem trước bố cục trên các thiết bị khác nhau mà không cần chạy ứng dụng trên thiết bị thật hoặc trình giả lập, nhấp vào nút **Device in Editor** trên thanh công cụ và chọn một thiết bị.
- Để tạo một biến thể bố cục khác dành cho **máy tính bảng (màn hình lớn hơn)**, nhấp vào nút **Orientation in Editor** và chọn **Create layout x-large Variation**.  
→ Một cửa sổ chỉnh sửa mới sẽ mở với tab **xlarge/activity\_main.xml**, hiển thị bố cục cho thiết bị có màn hình lớn.

Sử dụng ConstraintLayout

- Để xóa tất cả ràng buộc trong **ConstraintLayout**, nhấp vào nút **Clear All Constraints** trên thanh công cụ.
- Bạn có thể căn chỉnh một phần tử giao diện người dùng chứa văn bản (chẳng hạn như **TextView** hoặc **Button**) với một phần tử khác bằng **ràng buộc đường cơ sở (baseline constraint)**.
- Để tạo ràng buộc đường cơ sở, di chuột qua phần tử giao diện người dùng cho đến khi nút **baseline constraint** xuất hiện bên dưới phần tử.
- Nút **pack** trên thanh công cụ cung cấp tùy chọn để sắp xếp hoặc mở rộng các phần tử UI đã chọn.  
→ Bạn có thể sử dụng nó để **căn đều các nút (Button) theo chiều ngang trong bố cục**.

Sử dụng LinearLayout

- LinearLayout** là một **ViewGroup** sắp xếp các **View** theo hàng **ngang** hoặc **dọc**.
- Một **LinearLayout** cần có các thuộc tính sau:
  - layout\_width

- layout\_height
  - orientation
- **match\_parent** cho layout\_width hoặc layout\_height:
  - Mở rộng **View** để lấp đầy phần tử cha.
  - Nếu **LinearLayout** là phần tử gốc, nó sẽ mở rộng theo kích thước màn hình.
- **wrap\_content** cho layout\_width hoặc layout\_height:
  - Thu nhỏ **View** để vừa với nội dung của nó.
  - Nếu không có nội dung, **View** sẽ trở nên vô hình.
- **Số dp (density-independent pixels) cố định** cho layout\_width hoặc layout\_height:
  - Xác định kích thước cố định, được điều chỉnh theo mật độ màn hình.
  - Ví dụ: 16dp có nghĩa là 16 pixel độc lập với mật độ.
- **Hướng (orientation) của LinearLayout:**
  - horizontal → Sắp xếp phần tử từ **trái sang phải**.
  - vertical → Sắp xếp phần tử từ **trên xuống dưới**.
- **Sử dụng gravity và weight để kiểm soát bố cục:**
  - **android:gravity** → Căn chỉnh nội dung bên trong **View**.
  - **android:layout\_weight** → Xác định tỷ lệ không gian bổ sung được phân bổ cho **View**.
    - Nếu chỉ có một **View** có thuộc tính này, nó sẽ nhận toàn bộ không gian trống.
    - Nếu có nhiều **View**, không gian sẽ được chia theo tỷ lệ.
    - Ví dụ: Nếu hai **Button** có weight="1" và một **TextView** có weight="2" (tổng cộng 4), mỗi **Button** sẽ nhận  $\frac{1}{4}$  không gian và **TextView** sẽ nhận  $\frac{1}{2}$ .

## Sử dụng RelativeLayout

- **RelativeLayout** là một **ViewGroup**, trong đó mỗi **View** được căn chỉnh tương đối với các **View** khác trong cùng nhóm.
- Các thuộc tính quan trọng:
  - **android:layout\_alignParentTop** → Căn **View** lên **đỉnh** của phần tử cha.
  - **android:layout\_alignParentLeft** → Căn **View** về **bên trái** của phần tử cha.
  - **android:layout\_alignParentStart** → Căn **View** theo **cạnh bắt đầu** của phần tử cha.
    - **Start** là **bên trái** nếu ngôn ngữ từ **trái sang phải (LTR)**.
    - **Start** là **bên phải** nếu ngôn ngữ từ **phải sang trái (RTL)**.

## Tài liệu liên quan

Tài liệu về các khái niệm liên quan có trong:

**1.2: Bố cục và tài nguyên cho UI (Layouts and resources for the UI).**

Tìm hiểu thêm:

Tài liệu Android Studio:

- Giới thiệu về Android Studio
- Tạo biểu tượng ứng dụng với Image Asset Studio

Tài liệu Android Developer:

- Layouts
- Xây dựng UI với Layout Editor
- Xây dựng giao diện đáp ứng với ConstraintLayout
- LinearLayout
- RelativeLayout
- View
- Button
- TextView
- Hỗ trợ nhiều mật độ pixel khác nhau

Khác:

- Codelabs: **Sử dụng ConstraintLayout để thiết kế giao diện Android**
- **Từ vựng và khái niệm quan trọng**

## Bài tập về nhà

Thay đổi một ứng dụng

Mở ứng dụng **HelloToast**.

1. **Đổi tên** dự án thành **HelloConstraint**, sau đó **refactor** toàn bộ dự án thành **HelloConstraint**.  
→ (Hướng dẫn về cách sao chép và refactor một dự án có trong **Phụ lục: Tiện ích**.)

2. **Chỉnh sửa bố cục trong activity\_main.xml** để căn chỉnh các nút **Toast** và **Count** **đọc theo bên trái** của **TextView show\_count** (hiển thị số "0").  
→ Xem hình minh họa về bố cục.
3. **Thêm một nút thứ ba** có tên **Zero**, xuất hiện **giữa** các nút **Toast** và **Count**.
4. **Sắp xếp các nút theo chiều đọc**, phân bổ khoảng cách đều từ trên xuống dưới của **TextView show\_count**.
5. **Đặt nền ban đầu** của nút **Zero** thành **màu xám**.
6. **Đảm bảo rằng** nút **Zero** **xuất hiện** **trong** **các bố cục khác**, bao gồm:
  - **activity\_main.xml (land)** (giao diện ngang).
  - **activity\_main.xml (xlarge)** (giao diện trên màn hình máy tính bảng).
7. **Cập nhật xử lý sự kiện** của nút **Zero** để khi nhấn vào, nó sẽ đặt giá trị trong **show\_count** **về 0**.
8. **Cập nhật xử lý sự kiện** của nút **Count**:
  - Khi nhấn, nút **Count** sẽ đổi màu nền tùy thuộc vào số hiện tại là **chẵn** hay **lẻ**.
  - **Gợi ý:** Không sử dụng **findViewById** để tìm nút Count. Hãy tìm cách khác!
  - Có thể sử dụng các hằng số trong lớp **Color** để đặt màu nền.
9. **Cập nhật xử lý sự kiện** của nút **Count** để thay đổi màu nền của nút **Zero** thành một màu khác ngoài **xám**, nhằm thể hiện rằng nút Zero đã được kích hoạt.
  - **Gợi ý:** Lần này, có thể sử dụng **findViewById** để tìm nút Zero.
10. **Cập nhật xử lý sự kiện** của nút **Zero** để **đặt lại** màu nền của chính nó **về xám** khi số đếm bằng 0.

## Trả lời các câu hỏi

Câu hỏi 1:

Hai thuộc tính **ràng buộc bố cục (layout constraint)** nào trên **nút Zero** giúp nó **được căn giữa theo chiều đọc**, nằm cách đều hai nút còn lại? (Chọn 2 đáp án)  
**app:layout\_constraintBottom\_toTopOf="@+id/button\_count"**  
**app:layout\_constraintTop\_toBottomOf="@+id/button\_toast"**

Câu hỏi 2:

Thuộc tính ràng buộc bố cục nào trên **nút Zero** giúp nó **căn chỉnh ngang hàng** với hai nút còn lại?

**app:layout\_constraintLeft\_toLeftOf="parent"**

Câu hỏi 3:

Đâu là chữ ký (signature) đúng của một phương thức được dùng với thuộc tính **android:onClick** trong XML?

**public void callMethod(View view)**

Câu hỏi 4:

Trong trình xử lý sự kiện của nút **Count**, phương pháp nào **hiệu quả hơn** để thay đổi màu nền của nút?

**Sử dụng tham số view được truyền vào trình xử lý sự kiện, với setBackgroundColor():**

#### 1.4) Văn bản và các chế độ cuộn

Giới thiệu

Lớp **TextView** là một lớp con của **View** dùng để hiển thị văn bản trên màn hình. Bạn có thể kiểm soát cách hiển thị văn bản bằng cách sử dụng các thuộc tính **TextView** trong tệp **XML** của giao diện.

Trong bài thực hành này, bạn sẽ làm việc với nhiều phần tử **TextView**, bao gồm một phần tử có thể **cuộn nội dung theo chiều dọc**.

Nếu nội dung quá dài để hiển thị hết trên màn hình thiết bị, bạn có thể tạo một **chế độ xem cuộn (scrolling view)** để người dùng có thể **vuốt lên/xuống** để xem thêm nội dung. Ngoài ra, cũng có thể tạo cuộn **theo chiều ngang** bằng cách vuốt sang trái/phải.

Thông thường, chế độ xem cuộn được sử dụng cho:

- **Tin tức, bài báo, nội dung dài** không vừa trên màn hình.
- **Hộp nhập văn bản nhiều dòng**.
- **Kết hợp nhiều phần tử UI** (chẳng hạn như hộp văn bản và nút bấm) trong một chế độ cuộn.

Lớp **ScrollView** cung cấp bố cục cho chế độ xem cuộn. Đây là một lớp con của **FrameLayout**.

- **Chỉ chứa một phần tử con** trong nó.
- Phần tử con này có thể là một **ViewGroup** (chẳng hạn như **LinearLayout**) chứa nhiều phần tử UI bên trong.
- Nếu bố cục quá phức tạp (chứa nhiều hình ảnh), có thể gặp vấn đề về hiệu suất.

## Cách tổ chức bố cục cuộn hợp lý

- Sử dụng **LinearLayout** với **hướng dọc (vertical)** để chứa các phần tử có thể cuộn qua (chẳng hạn như **TextView**).
- Khi sử dụng **ScrollView**, tất cả các phần tử UI đều được **tải vào bộ nhớ** ngay cả khi không được hiển thị.
  - Điều này giúp **cuộn mượt mà đối với văn bản tĩnh**, nhưng có thể tiêu tốn nhiều bộ nhớ, gây ảnh hưởng đến hiệu suất ứng dụng.
  - Nếu cần hiển thị **danh sách dài và có thể chỉnh sửa** (thêm/xóa mục), nên sử dụng **RecyclerView** (được trình bày trong một bài học khác).

## Những kiến thức bạn cần biết trước

Bạn nên có kiến thức về:

Tạo ứng dụng "Hello World" với Android Studio.  
Chạy ứng dụng trên trình giả lập hoặc thiết bị thực tế.  
Sử dụng **TextView** trong bố cục ứng dụng.  
Tạo và sử dụng tài nguyên chuỗi (**string resources**).

## Những gì bạn sẽ học

Thêm nhiều **TextView** bằng XML.  
Định nghĩa chế độ xem cuộn (**ScrollView**) bằng XML.  
Hiển thị văn bản có định dạng HTML đơn giản (in đậm, in nghiêng, xuống dòng, v.v.).  
Tùy chỉnh màu nền và màu chữ của **TextView**.  
Tạo liên kết web có thể nhấp trong văn bản.

## Những gì bạn sẽ làm trong bài thực hành

Tạo ứng dụng "ScrollingText".  
Thay đổi **ConstraintLayout** thành **RelativeLayout**.  
Thêm hai **TextView** cho tiêu đề và tiêu đề phụ của bài viết.  
Sử dụng các thuộc tính **TextAppearance** và màu sắc để định dạng tiêu đề.  
Dùng thẻ HTML trong chuỗi văn bản để kiểm soát định dạng.  
Sử dụng thuộc tính **lineSpacingExtra** để tăng khoảng cách giữa các dòng, giúp dễ đọc hơn.

Thêm **ScrollView** để kích hoạt cuộn nội dung **TextView**.

Sử dụng thuộc tính **autoLink** để làm cho các URL trong văn bản có thể nhấp vào được.

---

## Tổng quan về ứng dụng

Ứng dụng **Scrolling Text** sẽ trình bày cách sử dụng **ScrollView** - một thành phần **UI ViewGroup** có thể chứa một **TextView** cuộn được.

Ứng dụng hiển thị một **bài đánh giá album nhạc** dài, cho phép người dùng **vượt lên/xuống** để đọc. Khi cuộn, một **thanh cuộn (scroll bar)** xuất hiện bên lề phải màn hình.

Ứng dụng này cũng minh họa cách sử dụng:

- **Thẻ HTML tối thiểu** để tạo văn bản **in đậm** hoặc **in nghiêng**.
- **Ký tự xuống dòng (\n)** để tách đoạn văn.
- **Liên kết web** có thể nhấp trong văn bản.

**Hình minh họa ứng dụng bao gồm:**

Một **liên kết web** có thể nhấp được nhúng trong văn bản.

Một **thanh cuộn** xuất hiện khi nội dung được cuộn.

## 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**.

- 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

- 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>`
- 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"`). Để lỏng—bạn sẽ xóa chúng trong bước tiếp theo.

- Xóa dòng mã XML sau đây, dòng này liên quan đến `ConstraintLayout`:

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

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

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context=".MainActivity" />
```

5. 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.
6. 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.

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

7. 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**.

8. 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.

TextView #2 Attribute	Value
layout_width	"match_parent"

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

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.

10. Trích xuất tài nguyên chuỗi cho chuỗi cứng "Article Subtitle" của thuộc tính android:text trong TextView thành article\_subtitle.

11. 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"
```

12. Thêm các thuộc tính TextView sau vào phần tử "Hello World" TextView và thay đổi thuộc tính android:text:

<b>TextView Attribute</b>	<b>Value</b>
android:id	"@+id/article"
android:layout_below	"@+id/article_subheading"
android:lineSpacingExtra	"5sp"
android:padding	"@dimen/padding_regular"
android:text	Change to "Article text"

13. Trích xuất tài nguyên chuỗi cho "Article text" thành article\_text và trích xuất tài nguyên kích thước cho "5sp" thành line\_spacing.

14. 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

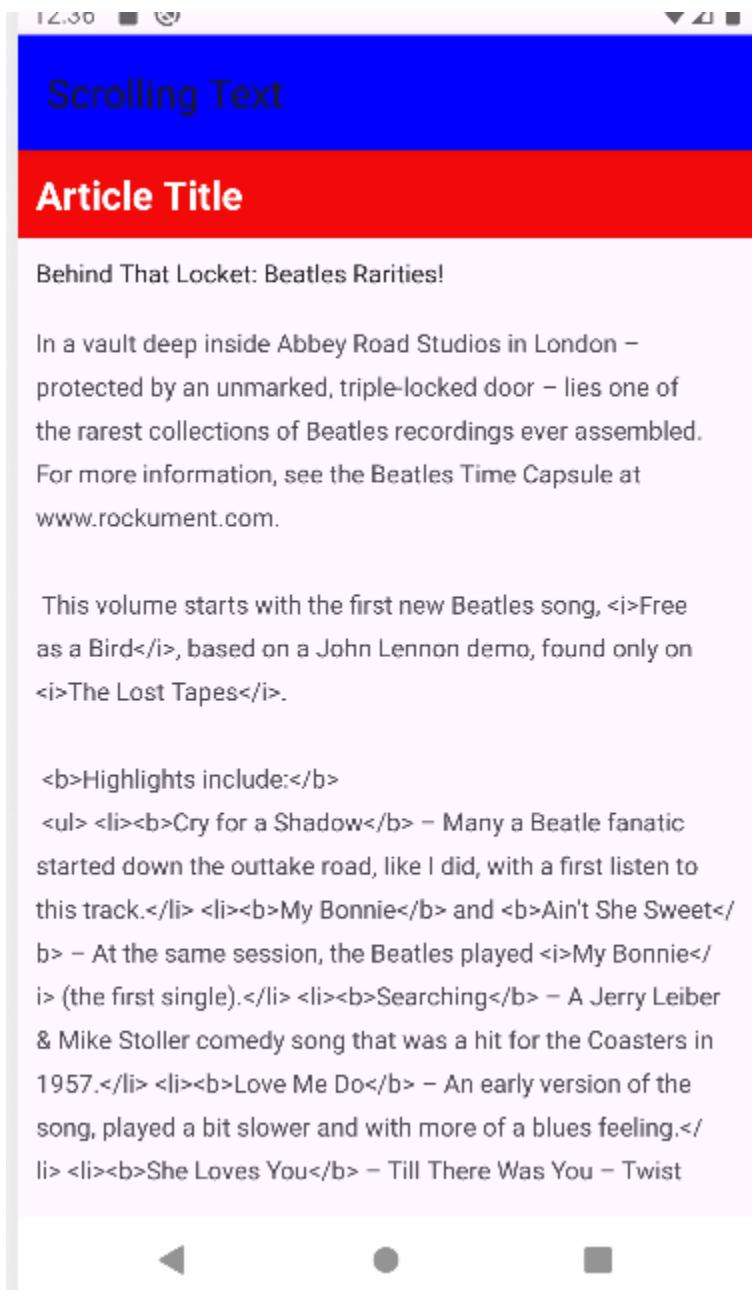
```

<resources>
    <string name="app_name">Scrolling Text</string>
    <string name="article_title">Article Title</string>
    <string name="article_subtitle">Behind That Locket: Beatles Rarities!</string>
    <string name="article_text"><![CDATA[
In a vault deep inside Abbey Road Studios in London - protected by an unmarked, triple-locked door - lies one of the rarest collections of Beatles recordings ever assembled
For more information, see the Beatles Time Capsule at www.rockument.com.<br>
This volume starts with the first new Beatles song, <i>Free as a Bird</i>, based on a John Lennon demo, found only on <i>The Lost Tapes</i>.<br>
<b>Highlights include:</b>
<ul>
<li><b>Cry for a Shadow</b> - Many a Beatle fanatic started down the outtake road, like I did, with a first listen to this track.</li>
<li><b>My Bonnie</b> and <b>Ain't She Sweet</b> - At the same session, the Beatles played <i>My Bonnie</i> (the first single).</li>
<li><b>Searching</b> - A Jerry Leiber & Mike Stoller comedy song that was a hit for the Coasters in 1957.</li>
<li><b>Love Me Do</b> - An early version of the song, played a bit slower and with more of a blues feeling.</li>
<li><b>She Loves You</b> - Till There Was You - Twist and Shout - Live at the Princess Wales Theatre by Leicester Square.</li>
<li><b>Leave My Kitten Alone</b> - One of the lost Beatle songs recorded during the <i>Beatles For Sale</i> sessions.</li>
<li><b>One After 909</b> - A song recorded for the <i>Let It Be</i> album but actually worked on way back in the beginning.</li>
</ul>
]]></string>
</resources>

```

### 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.



## Mã giải pháp nhiệm vụ 1

```

<?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.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" />

    <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 bộ cục XML để làm cho TextView có thể cuộn được.

### 2.1 Thêm thuộc tính autoLink để kích hoạt liên kết web

Thêm thuộc tính android:autoLink="web" vào TextView của bài viết. Mã XML cho TextView này bây giờ sẽ trông như sau:

```

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

```

```
    android:autoLink="web"
    android:layout_below="@+id/article_subheading"
    android:lineSpacingExtra="@dimen/line_spacing"
    android:padding="@dimen/padding_regular"
    android:text="@string/article_text" />
```

## 2.2 Thêm ScrollView vào bộ cục

Để làm cho Chế độ xem (chẳng hạn như TextView) có thể cuộn được, hãy nhúng Chế độ xem bên trong ScrollView.

- Thêm một **ScrollView** giữa TextView có ID article\_subheading và TextView có ID article. Khi bạn nhập <ScrollView, Android Studio sẽ tự động thêm </ScrollView> ở cuối, và hiển thị các gợi ý cho thuộc tính android:layout\_width và android:layout\_height.
- Chọn **wrap\_content** từ danh sách gợi ý cho cả hai thuộc tính.

Mã cho hai phần tử TextView và ScrollView bây giờ trông như sau:

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

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

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

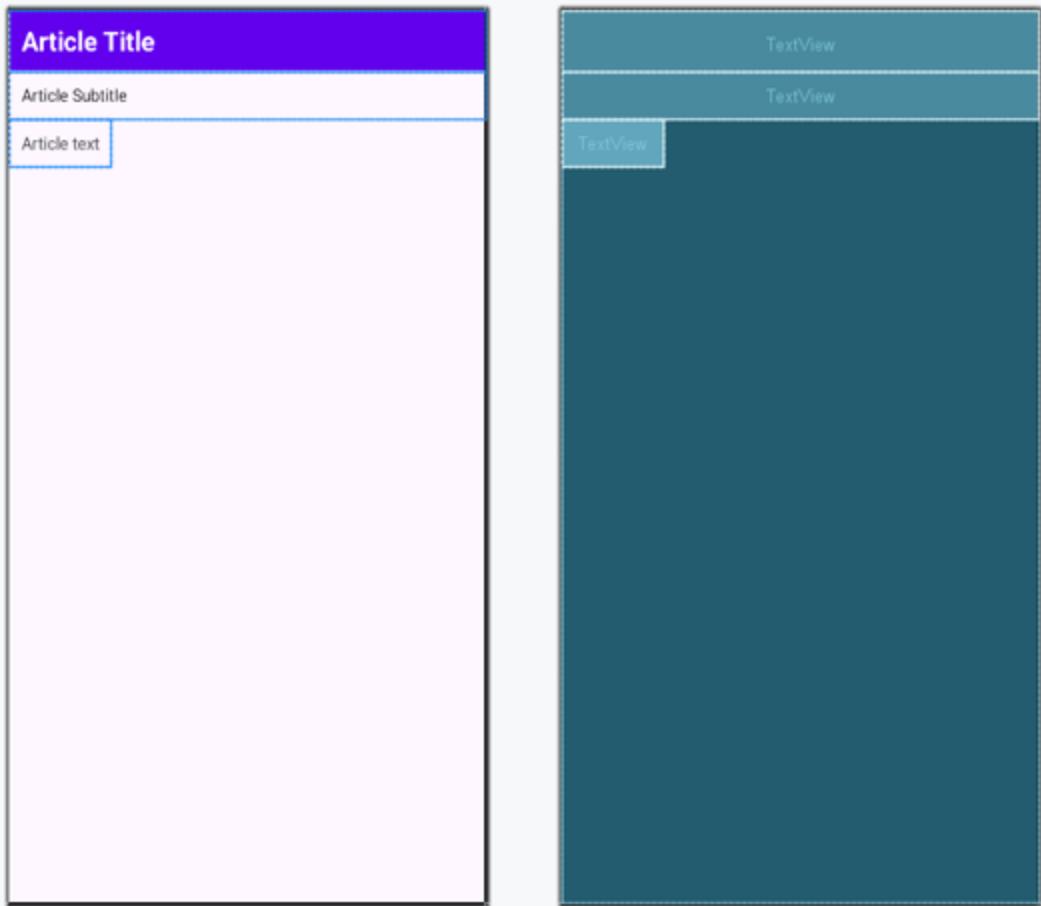
- Di chuyển đoạn mã kết thúc </ScrollView> xuống sau TextView có ID article để các thuộc tính của article nằm hoàn toàn bên trong **ScrollView**.
- Xóa thuộc tính sau từ TextView có ID article và thêm nó vào **ScrollView**:

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

Bài viết nằm bên trong phần tử ScrollView.

- 5.
- Nhấp vào tab **Preview** ở phía bên phải của trình chỉnh sửa bộ cục để xem trước bộ cục.

Bộ cục giờ đây trông giống như phần bên phải của hình minh họa sau:



### 2.3 Chạy ứng dụng

Để kiểm tra cách văn bản cuộn:

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

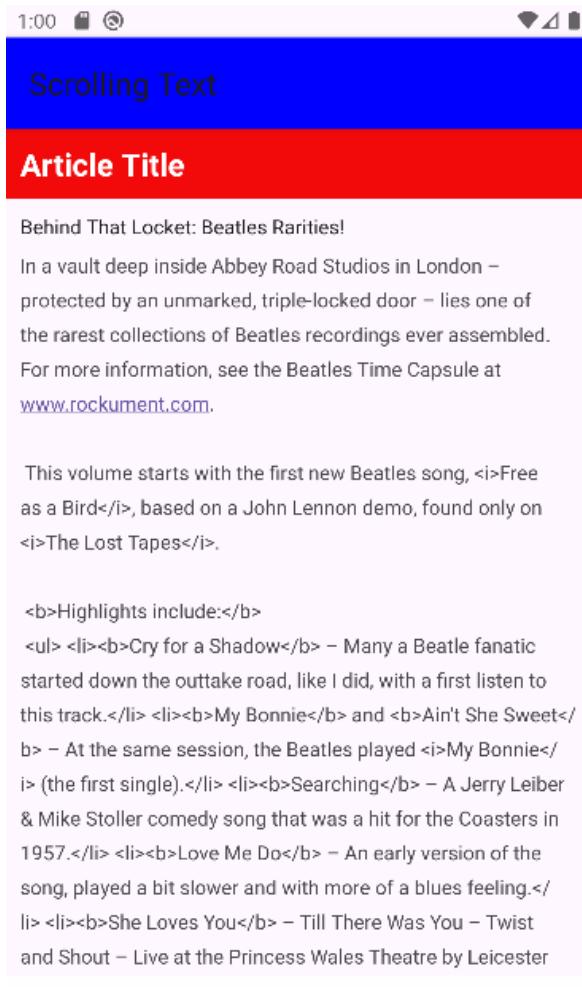
Vuốt lên và xuống để cuộn qua bài viết. Thanh cuộn xuất hiện ở lề phải khi bạn cuộn. Nhấn vào liên kết web để truy cập trang web. Thuộc tính android:autoLink tự động biến bất kỳ URL nào có thể nhận dạng được trong TextView (chẳng hạn như [www.rockument.com](http://www.rockument.com)) thành một liên kết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



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 bộ 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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.scrollingtext.MainActivity">

    <TextView
        android:id="@+id/article_heading"
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_title"

    android:textAppearance="@android:style/TextAppearance.DeviceDefault.Large"
        android:textColor="@android:color/white"
        android:textStyle="bold" />

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

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/article_subheading"
        android:layout_marginTop="5dp">

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

</RelativeLayout>

```

### 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:

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

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.

3. 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.
4. Thêm thuộc tính android:orientation="vertical" vào LinearLayout để thiết lập hướng của nó thành dọc.
5. Chọn **Code > Reformat Code** để thuần hóa mã chính xác.

LinearLayout bây giờ trông như thế này:

```
<ScrollView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/article_subheading">  
    <LinearLayout  
        android:orientation="vertical"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content">  
  
        <TextView  
            android:id="@+id/article"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:autoLink="web"
        android:lineSpacingExtra="@dimen/line_spacing"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_text" />
    </LinearLayout>
</ScrollView>
```

### 3.2 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.

- 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**.
- 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.
- Thay đổi thuộc tính bối cảnh 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:

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.scrollingtext.MainActivity">

    <TextView
        android:id="@+id/article_heading"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:padding="@dimen/padding_regular"
        android:text="@string/article_title"

        android:textAppearance="@android:style/TextAppearance.DeviceDefault.Large"
        android:textColor="@android:color/white"
        android:textStyle="bold" />
```

```
<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>

</RelativeLayout>
```

## 1.5) Tài nguyên có sẵn

### Giới thiệu

Những kiến thức bạn cần biết trước

Bạn nên có kiến thức về:

- Hiểu quy trình làm việc cơ bản của Android Studio.
- Tạo ứng dụng từ đầu bằng mẫu "Empty Activity".
- Sử dụng trình chỉnh sửa bố cục (Layout Editor).

## Những gì bạn sẽ học

- Nơi tìm thông tin và tài nguyên dành cho nhà phát triển Android.
- Cách thêm biểu tượng khởi chạy (launcher icon) vào ứng dụng.
- Cách tìm kiếm trợ giúp khi phát triển ứng dụng Android.

## Những gì bạn sẽ làm

- Khám phá các tài nguyên hỗ trợ dành cho nhà phát triển Android ở mọi cấp độ.
- Thêm biểu tượng khởi chạy cho ứng dụng.

## Tổng quan về ứng dụng

Bạn sẽ thêm một biểu tượng khởi chạy (launcher icon) vào ứng dụng HelloToast mà bạn đã tạo trước đó hoặc vào một ứng dụng mới.

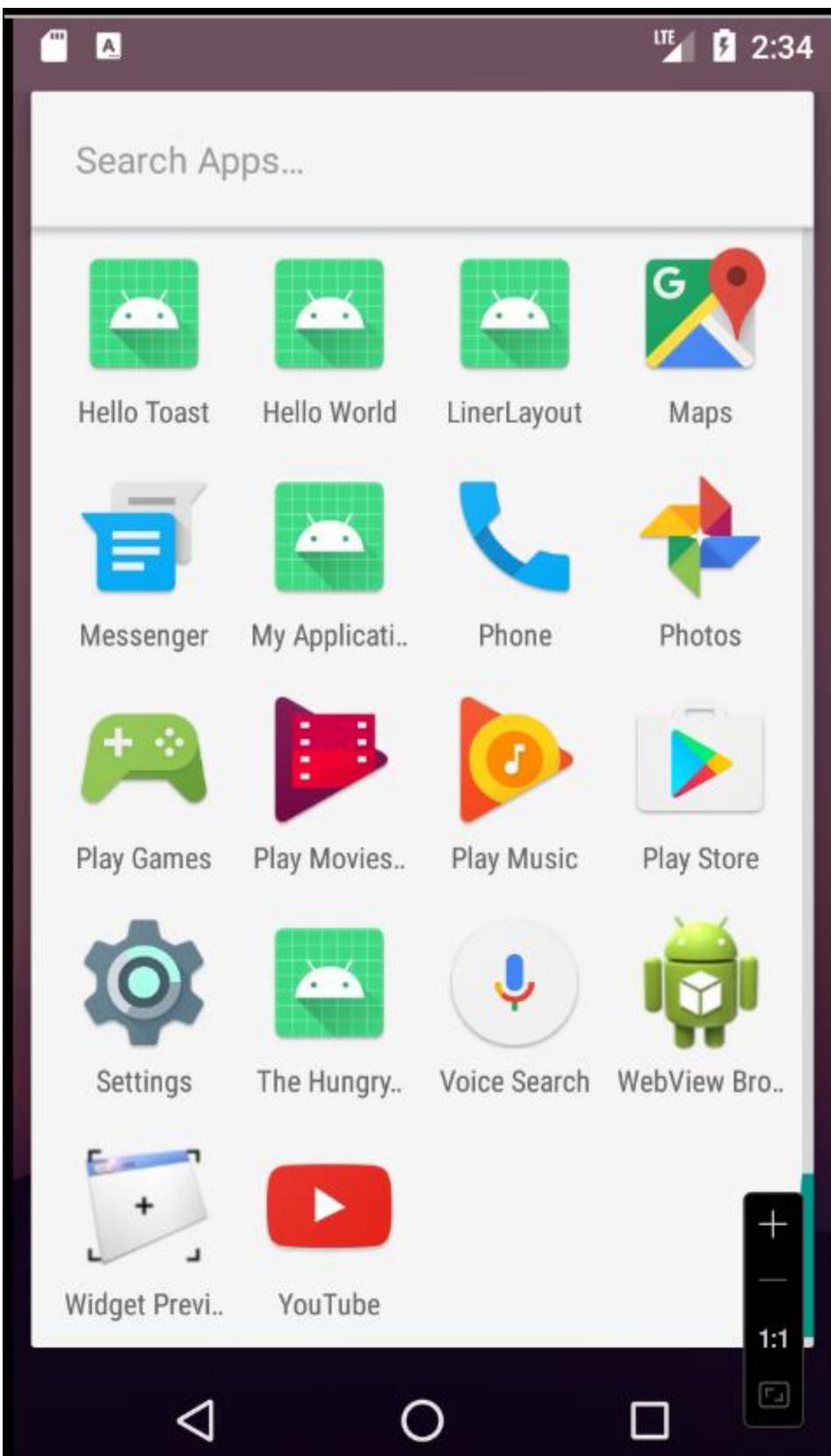
### Nhiệm vụ 1: Thay đổi biểu tượng khởi chạy

Mỗi ứng dụng mới bạn tạo bằng Android Studio sẽ bắt đầu với một biểu tượng khởi chạy tiêu chuẩn đại diện cho ứng dụng. Biểu tượng khởi chạy xuất hiện trong danh sách trên Google Play Store. Khi người dùng tìm kiếm trong Google Play Store, biểu tượng ứng dụng của bạn sẽ xuất hiện trong kết quả tìm kiếm.

Khi người dùng đã cài đặt ứng dụng, biểu tượng khởi chạy sẽ xuất hiện trên thiết bị ở nhiều vị trí khác nhau, bao gồm:

- Màn hình chính (Home screen)
- Màn hình tìm kiếm ứng dụng (Search Apps screen)

Ví dụ, ứng dụng HelloToast xuất hiện trong màn hình tìm kiếm ứng dụng của trình giả lập với biểu tượng mặc định của các dự án ứng dụng mới, như hình dưới đây.



Việc thay đổi biểu tượng khởi chạy là một quy trình từng bước đơn giản giúp bạn làm quen với các tính năng về tài sản hình ảnh (Image Asset) của Android Studio. Trong nhiệm vụ này, bạn cũng sẽ tìm hiểu cách truy cập tài liệu chính thức của Android.

## 1.1 Khám phá tài liệu chính thức của Android

Bạn có thể tìm thấy tài liệu chính thức dành cho nhà phát triển Android [tại developer.android.com](https://developer.android.com).

Tài liệu này chứa lượng thông tin phong phú và luôn được Google cập nhật thường xuyên.

16. Truy cập [developer.android.com/design/](https://developer.android.com/design/).

Đây là phần nói về Material Design, một triết lý thiết kế định nghĩa cách ứng dụng nên hiển thị và hoạt động trên thiết bị di động. Duyệt qua các liên kết để tìm hiểu thêm về Material Design. Ví dụ, truy cập phần Style để tìm hiểu về cách sử dụng màu sắc và các chủ đề khác.

17. Truy cập [developer.android.com/docs/](https://developer.android.com/docs/) để tìm: Thông tin về API, Tài liệu tham khảo, Hướng dẫn, Công cụ lập trình, Các ví dụ mã nguồn.

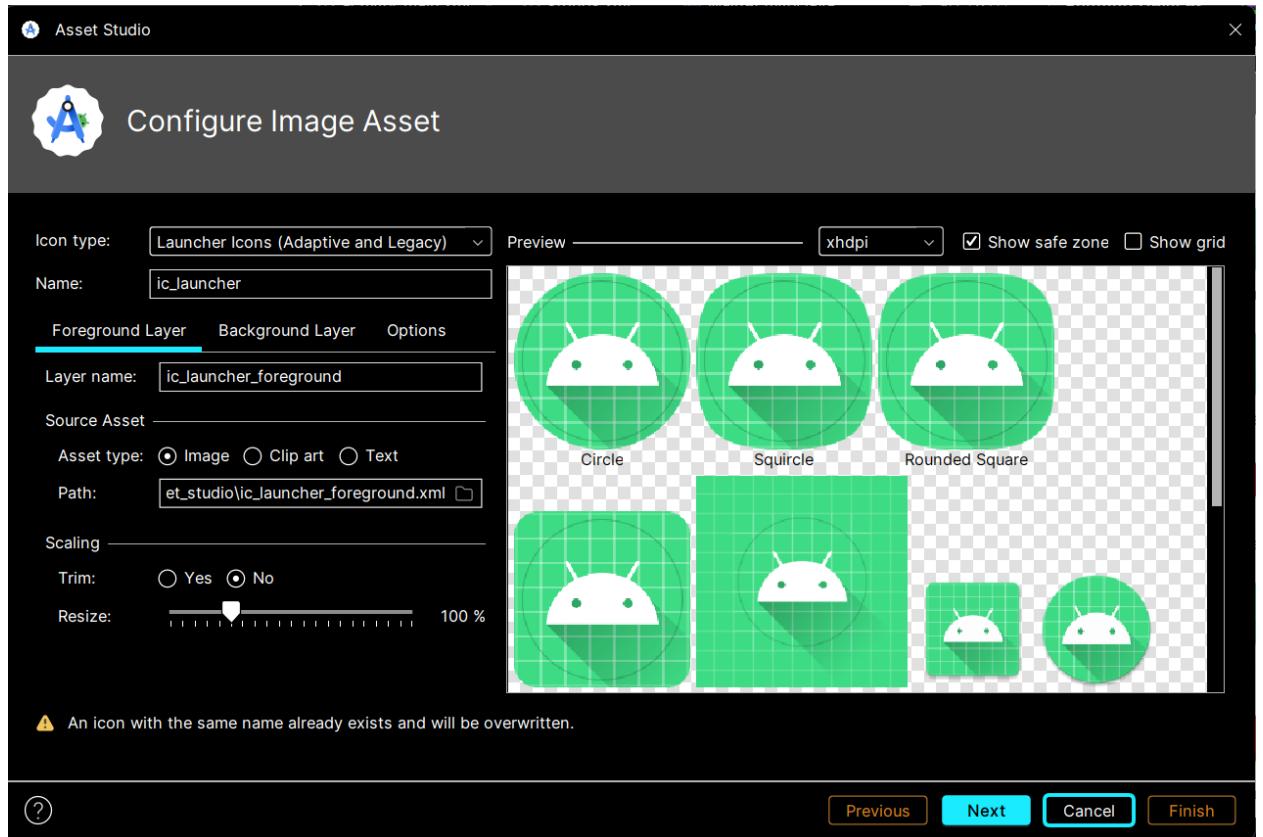
18. Truy cập [developer.android.com/distribute/](https://developer.android.com/distribute/) để tìm hiểu về cách đưa ứng dụng lên Google Play, hệ thống phân phối kỹ thuật số của Google dành cho các ứng dụng phát triển bằng Android SDK. Sử dụng Google Play Console để mở rộng lượng người dùng và bắt đầu kiếm tiền từ ứng dụng.

---

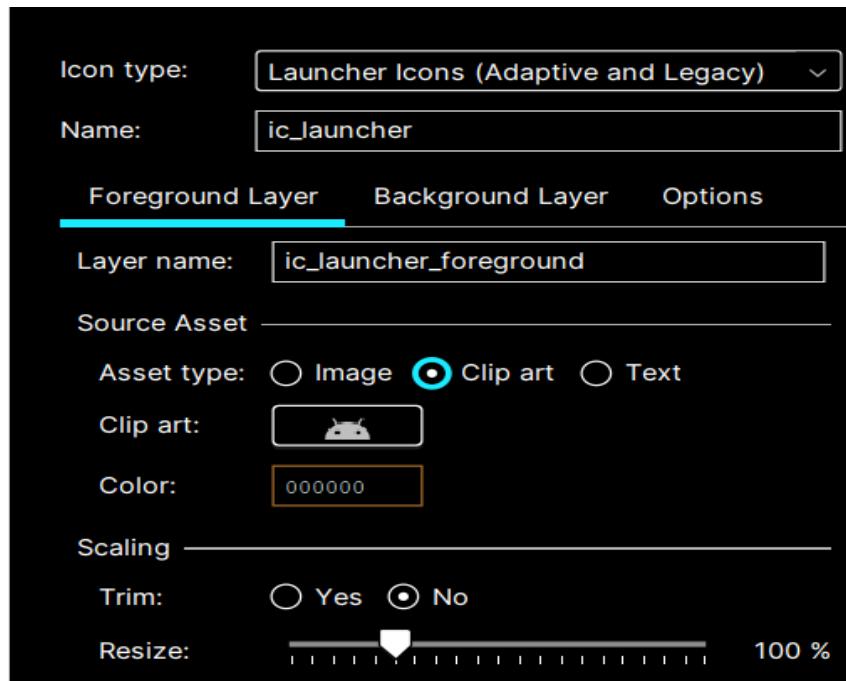
## 1.2 Thêm tài nguyên hình ảnh cho biểu tượng khởi chạy

Để thêm hình ảnh clip-art làm biểu tượng khởi chạy, hãy thực hiện các bước sau:

1. Mở dự án ứng dụng HelloToast từ bài học trước về trình chỉnh sửa bố cục (layout editor) hoặc tạo một dự án ứng dụng mới.
2. Trong Project > Android, nhấp chuột phải (hoặc Control-click trên macOS) vào thư mục res, sau đó chọn New > Image Asset. Cửa sổ Configure Image Asset sẽ xuất hiện.



3. Trong trường Icon Type, chọn Launcher Icons (Adaptive & Legacy) nếu nó chưa được chọn.
4. Nhấp vào tab Foreground Layer, chọn Clip Art trong mục Asset Type.



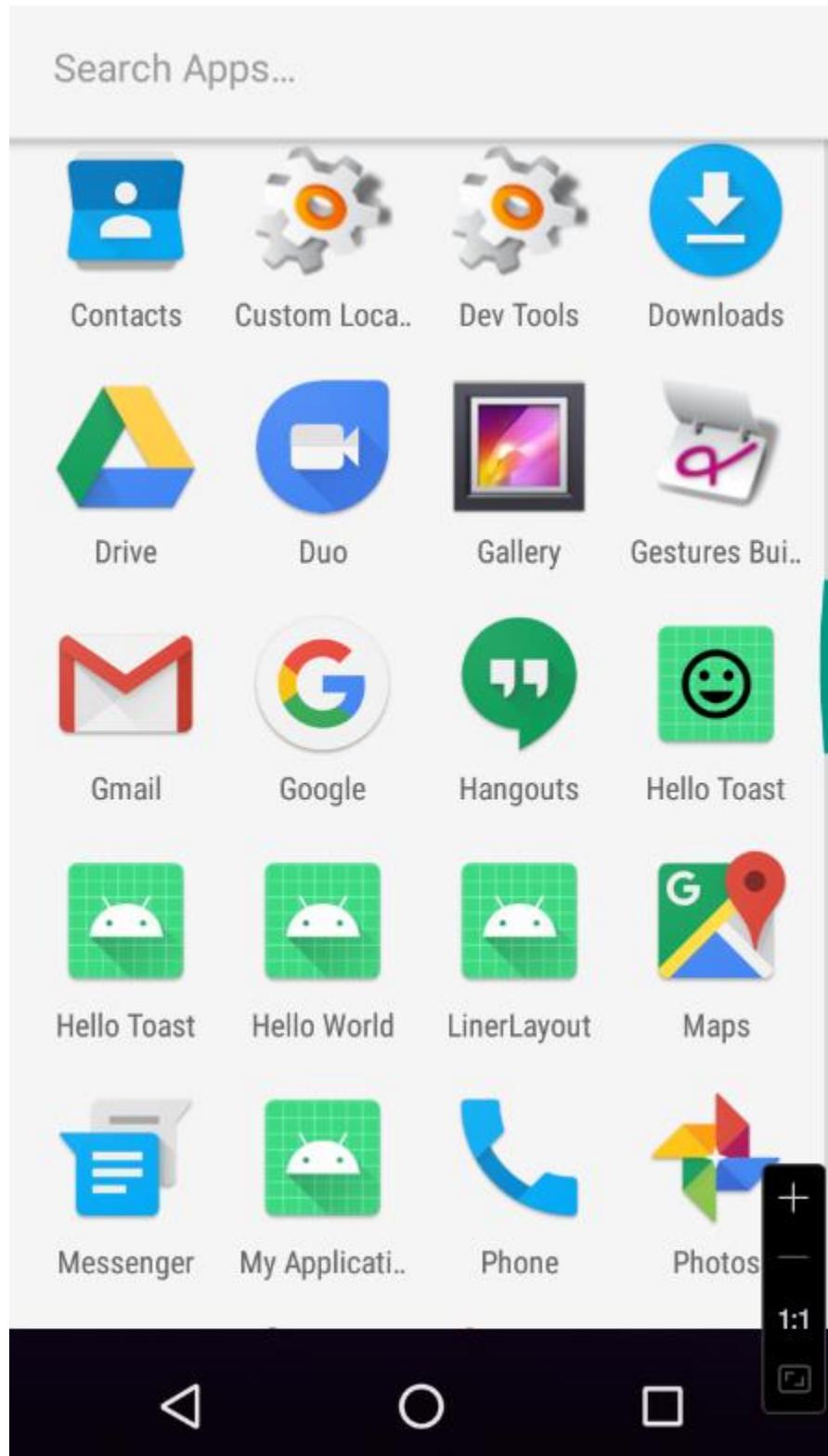
- Nhấp vào biểu tượng trong trường Clip Art. Các biểu tượng từ bộ biểu tượng Material Design sẽ xuất hiện.
- Duyệt qua cửa sổ Select Icon, chọn một biểu tượng phù hợp (chẳng hạn như biểu tượng mood để gợi ý tâm trạng vui vẻ), sau đó nhấp vào OK.



- Nhấp vào tab Background Layer, chọn Color làm Asset Type, sau đó nhấp vào chip màu để chọn một màu làm lớp nền.
- Nhấp vào tab Legacy và xem lại các cài đặt mặc định. Xác nhận rằng bạn muốn tạo các biểu tượng Legacy, Round và Google Play Store. Khi hoàn tất, nhấp vào Next.
- Chạy ứng dụng.

Android Studio tự động thêm hình ảnh biểu tượng khởi chạy vào thư mục mipmap cho các mật độ màn hình khác nhau. Kết quả là, biểu tượng khởi chạy của ứng dụng

sẽ thay đổi thành biểu tượng mới sau khi bạn chạy ứng dụng, như hình minh họa bên dưới.



Mẹo: Xem Launcher Icons để tìm hiểu thêm về cách thiết kế biểu tượng khởi chạy hiệu quả

## Nhiệm vụ 2: Sử dụng mẫu dự án

Android Studio cung cấp các mẫu thiết kế cho ứng dụng và hoạt động (activity) phổ biến, được khuyến nghị. Việc sử dụng các mẫu có sẵn giúp tiết kiệm thời gian và đảm bảo tuân theo các phương pháp thiết kế tốt nhất.

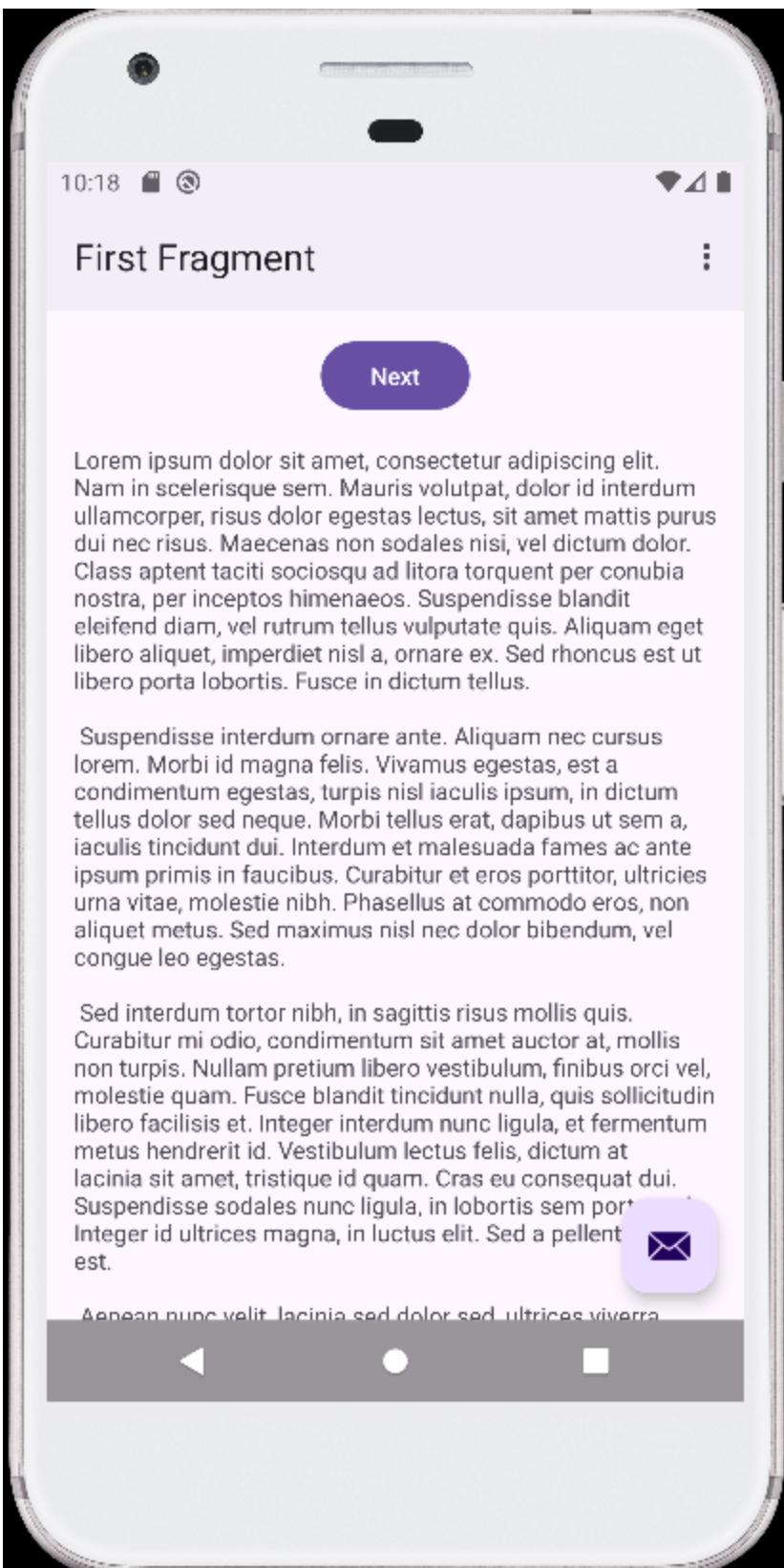
Mỗi mẫu bao gồm một activity và giao diện người dùng cơ bản. Bạn đã sử dụng mẫu Empty Activity. Mẫu Basic Activity có nhiều tính năng hơn và bao gồm các thành phần ứng dụng được khuyến nghị, chẳng hạn như menu tùy chọn xuất hiện trên thanh ứng dụng (app bar).

### 2.1 Khám phá kiến trúc của Basic Activity

Mẫu Basic Activity là một mẫu linh hoạt do Android Studio cung cấp, giúp bạn khởi động quá trình phát triển ứng dụng nhanh chóng.

1. Trong Android Studio, tạo một dự án mới với mẫu Basic Activity.
2. Xây dựng (build) và chạy ứng dụng.
3. Xác định các phần được gán nhãn trong hình và bảng bên dưới. Tìm các thành phần tương ứng trên màn hình thiết bị hoặc trình giả lập (emulator). Kiểm tra mã nguồn Java và tệp XML tương ứng được mô tả trong bảng.

Việc làm quen với mã nguồn Java và các tệp XML sẽ giúp bạn mở rộng và tùy chỉnh mẫu này theo nhu cầu của mình.



## Kiến trúc của mẫu Hoạt động cơ bản

#	Mô tả giao diện người dùng	Tham chiếu trong mã
1	Thanh trạng thái Hệ thống Android cung cấp và kiểm soát thanh trạng thái	Không hiển thị trong mã mẫu. Bạn có thể truy cập nó từ Activity của mình. Ví dụ, bạn có thể ẩn thanh trạng thái nếu cần thiết.
2	AppBarLayout > Toolbar Thanh ứng dụng (còn gọi là Action Bar) cung cấp cấu trúc giao diện, các thành phần giao diện tiêu chuẩn và điều hướng. Để đảm bảo tính tương thích ngược, AppBarLayout trong mẫu chứa một Toolbar với chức năng tương tự như ActionBar	Trong tệp <b>activity_main.xml</b> , tìm <b>android.support.v7.widget.Toolbar</b> bên trong <b>android.support.design.widget.AppBarLayout</b> . Thay đổi Toolbar để thay đổi giao diện của thành phần cha, App Bar. Để xem ví dụ, hãy tham khảo App Bar Tutorial.
3	Tên ứng dụng Tên này được lấy từ tên gói của bạn, nhưng có thể tùy chỉnh theo ý muốn	Trong tệp <b>AndroidManifest.xml</b> :  <code>android:label="@string/app_name"</code>
4	Nút menu tùy chọn (Options menu overflow button) Chứa các mục menu cho Activity, cũng như các tùy chọn chung như Tìm kiếm (Search) và Cài đặt (Settings) của ứng dụng. Các mục menu của ứng dụng sẽ được thêm vào menu này.	Trong <b>MainActivity.java</b> : <ul style="list-style-type: none"> <li>• <b>onOptionsItemSelected()</b> triển khai hành động xảy ra khi một mục menu được chọn.</li> </ul> Trong <b>res &gt; menu &gt; menu_main.xml</b> : <ul style="list-style-type: none"> <li>• Đây là tệp tài nguyên xác định các mục menu cho menu tùy chọn.</li> </ul>
5	Bố cục ViewGroup <b>CoordinatorLayout</b> là một <b>ViewGroup</b> giàu tính năng, cung cấp cơ chế để các phần tử <b>View (UI)</b> tương tác. Giao diện người dùng của ứng dụng được đặt trong tệp <b>content_main.xml</b> , tệp này được bao gồm bên trong <b>ViewGroup</b> này.	Trong <b>activity_main.xml</b> Không có <b>View</b> nào được chỉ định trực tiếp trong bố cục này; thay vào đó, nó bao gồm một bố cục khác bằng lệnh <b>include layout</b> , để đưa <b>@layout/content_main</b> vào, nơi chứa các <b>View</b> . Điều này giúp tách biệt các <b>View</b> của hệ thống khỏi các <b>View</b> riêng của ứng dụng.

6	<p><b>TextView</b></p> <p>Trong ví dụ, <b>TextView</b> được sử dụng để hiển thị "Hello World". Hãy thay thế nó bằng các <b>phần tử giao diện người dùng (UI elements)</b> phù hợp với ứng dụng của bạn.</p>	<p>Trong <b>content_main.xml</b> Tất cả các phần tử giao diện người dùng (UI elements) của ứng dụng đều được định nghĩa trong tệp này.</p>
7	<p>Nút hành động nổi (Floating Action Button – FAB)</p>	<p>Trong <b>activity_main.xml</b> được thêm vào dưới dạng một phần tử giao diện người dùng (UI element) sử dụng biểu tượng clip-art.</p> <p>Trong <b>MainActivity.java</b> bao gồm một đoạn mã mẫu (stub) trong <code>onCreate()</code>, thiết lập trình nghe sự kiện (<code>onClick()</code> listener) cho FAB.</p>

## 2.2 Tùy chỉnh ứng dụng được tạo bởi mẫu

Thay đổi giao diện của ứng dụng được tạo bởi mẫu Hoạt động Cơ bản.

Ví dụ, bạn có thể thay đổi màu của thanh ứng dụng để khớp với thanh trạng thái (trên một số thiết bị, thanh trạng thái có màu tối hơn của cùng một màu chính).

Bạn cũng có thể xóa nút hành động nổi nếu không sử dụng nó.

### 1. Thay đổi màu của thanh appBar(Toolbar) trong **activity\_main.xml**

- o Bằng cách thay đổi giá trị `android:background` thành:

```
    android:background="?attr/colorPrimary" />
```

- o Điều này sẽ đặt màu của thanh ứng dụng thành màu chính tối hơn, giúp khớp với thanh trạng thái.

### 2. Để xóa nút hành động nổi, bắt đầu bằng cách xóa đoạn mã trong `onCreate()`

- o Thiết lập trình nghe sự kiện (`onClick()` listener) cho nút.
- o Mở `MainActivity` và xóa khối mã sau.

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

3. Để xóa nút hành động nổi khỏi bố cục, xóa khối mã XML sau
  - o Trong activity\_main.xml.

```
<com.google.android.material.floatingactionbutton.FloatingActionButton  
    android:id="@+id/fab"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom|end"  
    android:layout_marginEnd="16dp"  
    android:layout_marginBottom="16dp"  
    app:srcCompat="@android:drawable/ic_dialog_email" />
```

4. Thay đổi tên của ứng dụng hiển thị trên thanh ứng dụng
  - o Bằng cách thay đổi giá trị chuỗi app\_name trong strings.xml thành nội dung sau.
5. Chạy ứng dụng
  - o Nút hành động nổi không còn xuất hiện, tên ứng dụng đã thay đổi, và màu nền của thanh ứng dụng đã thay đổi.



**Mẹo:** Xem Truy cập tài nguyên để biết chi tiết về cú pháp XML khi truy cập tài nguyên.

## 2.3 Khám phá cách thêm hoạt động bằng mẫu

Trong các bài thực hành trước, bạn đã sử dụng các mẫu Empty Activity và Basic Activity.

Trong các bài học sau, các mẫu bạn sử dụng sẽ thay đổi tùy theo nhiệm vụ.

Các mẫu Activity này cũng có sẵn trong dự án của bạn, cho phép bạn thêm Activity mới vào ứng dụng ngay cả sau khi thiết lập dự án ban đầu.

(Bạn sẽ tìm hiểu thêm về lớp Activity trong một chương khác.)

1. Tạo một dự án ứng dụng mới hoặc chọn một dự án hiện có.
2. Trong ngăn Project > **Android**, nhấp chuột phải vào thư mục **java**.
3. Chọn **New > Activity > Gallery**.
4. Thêm một **Activity**. Ví dụ: Nhấp vào Navigation Drawer Activity để thêm một Activity có ngăn điều hướng (navigation drawer) vào ứng dụng của bạn.
5. Nhấp đúp vào các tệp bố cục của **Activity** để hiển thị chúng trong trình chỉnh sửa bố cục (layout editor).

## Bài 3: Học từ mã nguồn mẫu

Android Studio và tài liệu Android cung cấp nhiều đoạn mã mẫu mà bạn có thể nghiên cứu, sao chép và tích hợp vào dự án của mình.

### 3.1 Mã nguồn mẫu Android

Bạn có thể khám phá hàng trăm mã mẫu trực tiếp trong Android Studio.

1. Trong Android Studio, chọn **File > New > Import Sample**.
2. Duyệt qua các mẫu có sẵn.
3. Chọn một mẫu và nhấp vào **Next**.
4. Chấp nhận các thiết lập mặc định và nhấp vào **Finish**.

**Lưu ý:** Các mẫu này chỉ là điểm khởi đầu để phát triển thêm. Chúng tôi khuyến khích bạn tự thiết kế và xây dựng ý tưởng của riêng mình.

### 3.2 Sử dụng SDK Manager để cài đặt tài liệu ngoại tuyến

Khi cài đặt Android Studio, các thành phần thiết yếu của Android SDK (Software Development Kit) cũng được cài đặt. Tuy nhiên, bạn có thể cài đặt thêm thư viện và tài liệu bằng SDK Manager.

1. Chọn Tools > Android > SDK Manager.
2. Ở cột bên trái, nhấp vào Android SDK.
3. Sao chép đường dẫn trong phần Android SDK Location (bạn sẽ cần nó để tìm tài liệu trên máy tính).



4. Nhấp vào tab SDK Platforms để cài đặt các phiên bản Android bổ sung.
5. Nhấp vào tab SDK Update Sites để xem các trang cập nhật mà Android Studio kiểm tra thường xuyên.
6. Nhấp vào tab SDK Tools để cài đặt các công cụ SDK bổ sung và tài liệu nhà phát triển Android ngoại tuyến.
7. Chọn hộp kiểm Documentation for Android SDK nếu nó chưa được cài đặt, rồi nhấp vào Apply.
8. Khi quá trình cài đặt hoàn tất, nhấp vào Finish.
9. Điều hướng đến thư mục sdk đã sao chép trước đó, mở thư mục docs.
10. Tìm tệp index.html và mở nó.

## Bài 4: Nhiều nguồn tài nguyên hơn

- Kênh YouTube Android Developer là một nguồn tuyệt vời để học các hướng dẫn và mẹo hay.
- Đội ngũ Android thường xuyên đăng tin tức và mẹo trên blog chính thức của Android.
- Stack Overflow là một cộng đồng lập trình viên hỗ trợ lẫn nhau. Nếu gặp vấn đề, rất có thể ai đó đã đăng câu trả lời. Bạn có thể thử đặt câu hỏi như:
  - "Làm thế nào để thiết lập và sử dụng ADB qua WiFi?"
  - "Những lỗi rò rỉ bộ nhớ phổ biến nhất trong lập trình Android là gì?"
- Cuối cùng nhưng không kém phần quan trọng, hãy tìm kiếm trên Google. Công cụ tìm kiếm sẽ thu thập kết quả từ tất cả các nguồn trên. Ví dụ: "Phiên bản Android OS phổ biến nhất ở Ấn Độ là gì?"

### 4.1 Tìm kiếm trên Stack Overflow bằng thẻ (tags)

Truy cập Stack Overflow và nhập [android] vào ô tìm kiếm. Dấu [...] giúp bạn tìm các bài viết có thẻ liên quan đến Android.

Bạn có thể kết hợp nhiều thẻ và từ khóa để tìm kiếm chính xác hơn. Hãy thử các tìm kiếm sau:

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

Để tìm hiểu thêm về cách tìm kiếm hiệu quả trên Stack Overflow, hãy xem trung tâm trợ giúp của Stack Overflow.

## Tóm tắt

- Tài liệu chính thức về Android Developer: [developer.android.com](http://developer.android.com)
- Material Design là một triết lý thiết kế giúp ứng dụng hoạt động và hiển thị tốt trên thiết bị di động.
- Google Play Store là nền tảng phân phối ứng dụng của Google dành cho các ứng dụng phát triển bằng Android SDK.
- **Android Studio** cung cấp các mẫu thiết kế cho ứng dụng và hoạt động (activity) phổ biến, được khuyến nghị. Những mẫu này bao gồm mã nguồn hoạt động cho các trường hợp sử dụng phổ biến.
- Khi tạo một dự án, bạn có thể chọn một mẫu cho activity đầu tiên của mình. Trong quá trình phát triển ứng dụng, bạn có thể tạo thêm các activity và thành phần khác từ các mẫu có sẵn.
- **Android Studio** chứa nhiều đoạn mã mẫu mà bạn có thể nghiên cứu, sao chép và tích hợp vào dự án của mình.

## Khái niệm liên quan

Tài liệu về khái niệm liên quan có trong **Mục 1.4: Các tài nguyên giúp bạn học tập**.

## Tìm hiểu thêm

### Tài liệu về Android Studio

- Giới thiệu về Android Studio
- Những bước cơ bản trong quy trình phát triển

### Tài liệu dành cho lập trình viên Android

- Trang web dành cho lập trình viên Android
- Khóa đào tạo của Google Developers
- Bố cục (Layouts)
- Tổng quan về tài nguyên ứng dụng

- Bố cục (Layouts)
- Menu
- TextView
- Tài nguyên chuỗi (String resources)
- Tệp khai báo ứng dụng (App Manifest)

## Mã nguồn mẫu

- Mã nguồn bài tập trên GitHub
- Mã mẫu dành cho lập trình viên Android

## Video

- Kênh YouTube Android Developer
- Các khóa học trực tuyến trên Udacity

## Khác

- Blog chính thức của Android
- Blog của Android Developers
- Google Developers Codelabs
- Stack Overflow
- Từ vựng Android

## Bài tập về nhà

Tải một ứng dụng mẫu và khám phá tài nguyên

1. Tải một ứng dụng mẫu vào Android Studio.
2. Mở một tệp Java activity trong ứng dụng. Tìm một lớp, kiểu dữ liệu hoặc thủ tục mà bạn chưa quen thuộc và tra cứu trong tài liệu Android Developer.
3. Truy cập Stack Overflow và tìm các câu hỏi về chủ đề tương tự.
4. Thay đổi biểu tượng khởi chạy. Sử dụng một biểu tượng có sẵn trong mục image assets của Android Studio.

## Bài 2) Activities

### 2.1) Activity và Intent

#### Giới thiệu

**Activity** đại diện cho một màn hình trong ứng dụng, nơi người dùng có thể thực hiện một tác vụ cụ thể như chụp ảnh, gửi email hoặc xem bản đồ. Một activity thường được hiển thị dưới dạng cửa sổ toàn màn hình.

Một ứng dụng thường bao gồm nhiều màn hình (activity) có liên kết lồng lěo với nhau. Trong đó, một activity sẽ được chỉ định là "**main**" activity (*MainActivity.java*), hiển thị cho người dùng khi ứng dụng được khởi chạy. Activity chính có thể khởi chạy các activity khác để thực hiện các tác vụ khác nhau.

Mỗi khi một activity mới bắt đầu, activity trước đó sẽ bị dừng nhưng vẫn được lưu trong ngăn xếp back stack. Ngăn xếp này hoạt động theo nguyên tắc "vào sau, ra trước" (Last In, First Out - LIFO). Khi người dùng nhấn nút Back, activity hiện tại sẽ bị xóa khỏi ngăn xếp và activity trước đó sẽ tiếp tục chạy.

Một activity được khởi chạy hoặc kích hoạt bằng một intent. Intent là một thông điệp bất đồng bộ được sử dụng để yêu cầu một hành động từ một activity khác hoặc từ một thành phần khác trong ứng dụng. Intent giúp khởi chạy activity từ một activity khác và truyền dữ liệu giữa các activity.

#### Có hai loại Intent

- **Explicit Intent (Intent tường minh):** Được sử dụng khi bạn biết rõ activity đích cần gọi, tức là bạn đã biết tên lớp đầy đủ của activity đó.
- **Implicit Intent (Intent ẩn danh):** Được sử dụng khi bạn không chỉ định activity cụ thể mà chỉ yêu cầu thực hiện một hành động chung.

#### Những kiến thức cần có trước

Bạn cần có khả năng:

- Tạo và chạy ứng dụng trong Android Studio.
- Sử dụng Layout Editor để tạo bố cục trong ConstraintLayout.
- Chỉnh sửa mã XML layout.
- Thêm chức năng onClick vào một Button.

## Những gì bạn sẽ học

- Cách tạo một Activity mới trong Android Studio.
- Cách xác định **parent activity** và **child activity** để thiết lập điều hướng **Up navigation**.
- Cách khởi chạy một Activity bằng **Explicit Intent**.
- Cách truyền dữ liệu giữa các Activity bằng **Explicit Intent**.

## Những gì bạn sẽ làm

- Tạo một ứng dụng Android mới với một **Main Activity** và một **Second Activity**.
- Truyền một chuỗi dữ liệu từ **Main Activity** đến **Second Activity** bằng **Intent** và hiển thị dữ liệu đó.
- Gửi một dữ liệu khác từ **Second Activity** trở lại **Main Activity**, cũng bằng **Intent**.

## Tổng quan về ứng dụng

Trong chương này, bạn sẽ tạo và xây dựng một ứng dụng có tên Two Activities, đúng như tên gọi, ứng dụng này sẽ chứa hai Activity. Bạn sẽ phát triển ứng dụng theo ba giai đoạn.

Ở giai đoạn đầu tiên, bạn sẽ tạo một ứng dụng trong đó Main Activity chứa một nút Send. Khi người dùng nhấn nút này, Main Activity sẽ sử dụng Intent để khởi chạy Second Activity.

Ở giai đoạn thứ hai, bạn sẽ thêm một EditText vào Main Activity. Người dùng sẽ nhập một tin nhắn và nhấn Send. Main Activity sẽ sử dụng Intent để khởi chạy Second Activity và gửi tin nhắn của người dùng sang Second Activity. Second Activity sẽ hiển thị tin nhắn mà nó nhận được.

Ở giai đoạn cuối, bạn sẽ thêm một EditText và một nút Reply vào Second Activity. Lúc này, người dùng có thể nhập một tin nhắn phản hồi và nhấn Reply. Tin nhắn phản hồi sẽ được hiển thị trên Main Activity. Tại thời điểm này, bạn sẽ sử dụng Intent để gửi phản hồi từ Second Activity trở lại Main Activity.

## Nhiệm vụ 1: Tạo dự án TwoActivities

Trong nhiệm vụ này, bạn sẽ thiết lập dự án ban đầu với một Main Activity, định nghĩa bố cục (layout) và tạo một phương thức khung (skeleton method) cho sự kiện onClick của nút bấm.

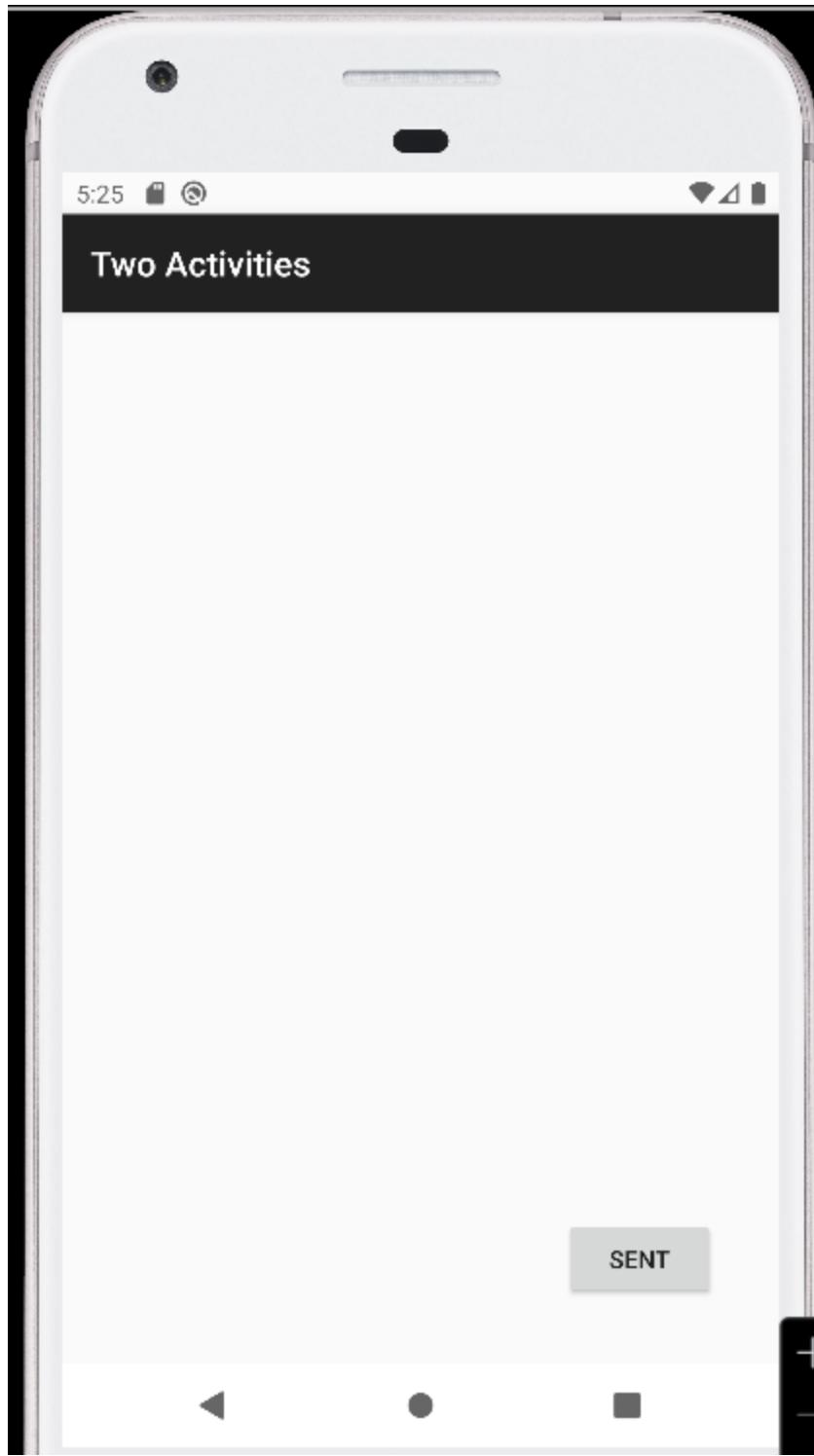
## 1.1 Tạo dự án TwoActivities

1. Mở Android Studio và tạo một dự án Android Studio mới.
  - o Đặt tên ứng dụng là Two Activities và chọn cài đặt Phone and Tablet giống như các bài thực hành trước.
  - o 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.
  - o 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:
  - o ID: button\_main
  - o layout\_width và layout\_height: wrap\_content
  - o Text: Send

Lúc này, bố cục sẽ trông như sau:



5. Nhấn vào tab Text để chỉnh sửa mã XML. Thêm thuộc tính sau vào Button:

```
        ...
        android:onClick="launchSecondActivity"/>
```

Lưu ý: Giá trị của thuộc tính này sẽ được gạch chân màu đỏ vì phương thức launchSecondActivity() chưa được tạo. Hãy bỏ qua lỗi này, bạn sẽ sửa nó trong nhiệm vụ tiếp theo.

- Trích xuất tài nguyên chuỗi (String Resource) cho chữ "Send" như đã hướng dẫn trong bài thực hành trước và đặt tên tài nguyên là button\_main.

Mã XML cho **Button** sẽ trông như sau:

```
<Button  
    android:id="@+id/button5"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginEnd="36dp"  
    android:layout_marginBottom="36dp"  
    android:text="SENT"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    android:onClick="launchSecondActivity"/>
```

### 1.3 định nghĩa hành động của button

Trong nhiệm vụ này, bạn sẽ triển khai phương thức launchSecondActivity() mà bạn đã tham chiếu trong thuộc tính android:onClick của tệp **activity\_main.xml**.

- nhấp vào "launchSecondActivity" trong mã xml của **activity\_main.xml**.
- nhấn **alt+enter** (hoặc **option+enter** trên mac) và chọn **create 'launchSecondActivity(View)' in 'MainActivity'**.
  - tệp **MainActivity** sẽ mở ra và android studio sẽ tạo một phương thức khung xương cho trình xử lý **launchSecondActivity()**.
- bên trong launchSecondActivity(), thêm một câu lệnh **log** để hiển thị thông báo "Button Clicked!".

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

- tại bước này, **LOG\_TAG** sẽ hiển thị màu đỏ vì chưa được khai báo. bạn sẽ thêm định nghĩa cho biến này ở bước sau.
- ở đầu lớp **MainActivity**, thêm một hằng số cho biến **LOG\_TAG**:
  - hằng số này sử dụng chính tên của lớp làm tag.

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

5. chạy ứng dụng. khi bạn nhấn nút **send**, thông báo "Button Clicked!" sẽ xuất hiện trong **logcat**.



2025-03-02 09:40:36.728 11084-11084 MainActivity com.example.twoactivities D | Button clicked!

- o nếu có quá nhiều đầu ra trong logcat, hãy nhập **MainActivity** vào hộp tìm kiếm để chỉ hiển thị các dòng có chứa tag này.

mã cho **MainActivity** sẽ trông như sau:

```
package com.example.twoactivities;

import android.os.Bundle;
import android.util.Log;
import android.view.View;

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

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

## Nhiệm vụ 2: tạo và khởi chạy activity thứ hai

Mỗi activity mới bạn thêm vào dự án sẽ có tệp layout và java riêng, tách biệt với main activity. chúng cũng có các phần tử `<activity>` riêng trong tệp AndroidManifest.xml. giống như main activity, các activity mới mà bạn tạo trong android studio cũng sẽ kế thừa từ lớp `AppCompatActivity`.

mỗi activity trong ứng dụng chỉ được kết nối lồng lěo với các activity khác. tuy nhiên, bạn có thể định nghĩa một activity là **parent** (cha) của một activity khác trong tệp **AndroidManifest.xml**. mỗi quan hệ cha-con này giúp android hiển thị các gợi ý điều hướng, chẳng hạn như **mũi tên quay lại** trong thanh tiêu đề của mỗi activity.

một activity có thể giao tiếp với các activity khác (trong cùng ứng dụng hoặc ứng dụng khác) bằng **intent**. có hai loại **intent**:

- **explicit intent**: bạn biết chính xác activity mục tiêu, tức là bạn đã biết tên lớp đầy đủ của activity đó.
- **implicit intent**: bạn không biết tên của component đích, nhưng chỉ có một hành động chung cần thực hiện.

## 2.1 tạo activity thứ hai

1. nhấp chuột vào thư mục **app** trong dự án, sau đó chọn **file > new > activity > empty activity**.
2. Đặt tên cho activity mới là **SecondActivity**. đảm bảo rằng **generate layout file** và **backwards compatibility (AppCompat)** được chọn.
  - tên tệp **layout** sẽ được đặt tự động là **activity\_second.xml**.
  - không chọn tùy chọn **launcher activity**.
3. nhấp vào **finish. android studio** sẽ:
  - thêm một tệp **layout** mới (**activity\_second.xml**).
  - thêm một tệp **java** mới (**SecondActivity.java**).
  - cập nhật tệp **AndroidManifest.xml** để bao gồm **SecondActivity**.

## 2.2 chỉnh sửa tệp androidmanifest.xml

1. mở thư mục **manifests > AndroidManifest.xml**.
2. tìm phần tử **<activity>** mà **android studio** đã tạo cho **SecondActivity**:

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

3. thay thế toàn bộ phần tử **<activity>** bằng đoạn mã sau:

```
<activity
    android:name=".launchSecondActivity"
    android:label="Second Activity"
    android:parentActivityName=".MainActivity">
```

```

<meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value="com.example.twoactivities.MainActivity"/>
</activity>

```

- thuộc tính label thêm tiêu đề của activity vào thanh ứng dụng (**app bar**).
  - thuộc tính parentActivityName xác định rằng **MainActivity** là **cha** của **SecondActivity**.
    - điều này giúp hỗ trợ **điều hướng ngược (up navigation)** trong ứng dụng.
    - khi người dùng ở **SecondActivity**, sẽ có **mũi tên quay lại** trên **app bar** để trở về **MainActivity**.
  - phần tử `<meta-data>` cung cấp thông tin bổ sung về activity dưới dạng **cặp key-value**.
    - trong trường hợp này, nó hoạt động giống như thuộc tính `android:parentActivityName`.
    - những thuộc tính metadata này **bắt buộc** đối với các phiên bản android cũ, vì `android:parentActivityName` chỉ có từ **API level 16 trở lên**.
4. trích xuất chuỗi "Second Activity" thành một **string resource**, sử dụng tên tài nguyên là **activity2\_name**.

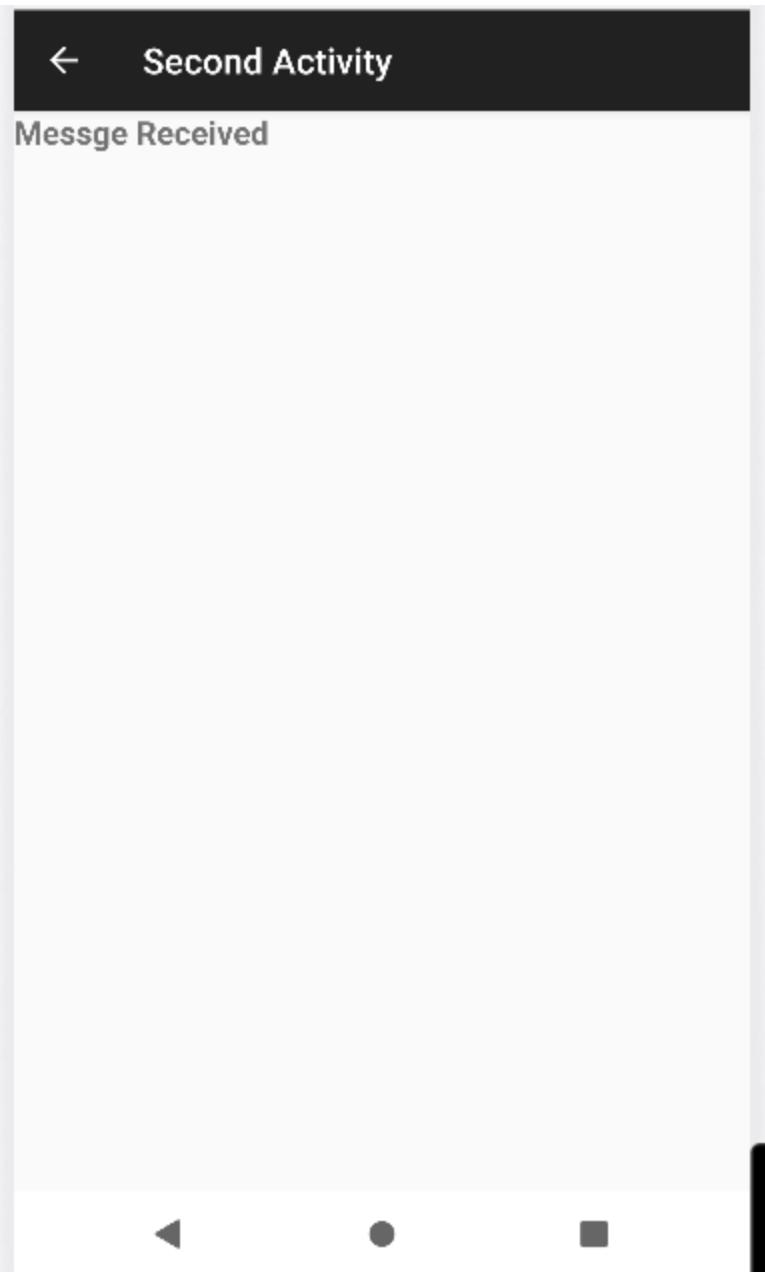
### 2.3 định nghĩa layout cho activity thứ hai

1. mở tệp **activity\_second.xml** và nhấp vào tab **design** (nếu chưa được chọn).
2. kéo một **TextView** từ bảng **palette** vào góc trên bên trái của layout, sau đó thêm **constraints** (ràng buộc) với cạnh trên và cạnh trái của layout.
  - đặt các thuộc tính trong bảng **attributes** như sau:

thuộc tính	giá trị
id	text_header
top margin	16dp
left margin	8dp
layout_width	wrap_content

thuộc tính	giá trị
layout_height	wrap_content
text	"Message Received"
textAppearance	AppCompat.Medium
textStyle	<b>B</b> (bold)

- **textAppearance** là một thuộc tính đặc biệt trong chủ đề (**theme**) của android, giúp định nghĩa kiểu chữ cơ bản. bạn sẽ tìm hiểu thêm về **theme** trong các bài học sau.



3. nhấp vào tab **text** để chỉnh sửa mã xml và trích xuất chuỗi "Message Received" thành một **string resource**, đặt tên là **text\_header**.
4. thêm thuộc tính `android:layout_marginLeft="8dp"` vào **TextView** để bổ sung cho `layout_marginStart`, giúp hỗ trợ các phiên bản android cũ hơn.

tệp **activity\_second.xml** sau khi chỉnh sửa sẽ trông 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=".activity_second">

    <TextView
        android:id="@+id/text_hear"

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_hear"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

## 2.4 Thêm intent vào main activity

Trong nhiệm vụ này, bạn sẽ thêm một **explicit intent** vào **MainActivity**. intent này sẽ được sử dụng để kích hoạt **SecondActivity** khi nút **send** được nhấn.

1. mở tệp **MainActivity**.
2. trong phương thức `launchSecondActivity()`, tạo một **intent** mới.
  - o constructor của Intent nhận **hai đối số** khi tạo một **explicit intent**:
    - **Context** của ứng dụng.
    - **component cụ thể** sẽ nhận intent.
  - o ở đây, sử dụng `this` làm **context** và `SecondActivity.class` làm **class đích**:

```
Intent myIntent = new Intent(packageContext: MainActivity.this, activity_second.class);
```

3. gọi phương thức `startActivity()` với **intent** vừa tạo làm đối số.

```
startActivity(myIntent);
```

4. chạy ứng dụng.
  - o khi bạn nhấn nút **send**, **MainActivity** sẽ gửi intent và hệ thống android sẽ khởi chạy **SecondActivity**.
  - o để quay lại **MainActivity**, bạn có thể nhấn nút **up** (mũi tên trái trên thanh ứng dụng) hoặc nút **back** ở cuối màn hình.

## Nhiệm vụ 3: Gửi dữ liệu từ MainActivity sang SecondActivity

Trong nhiệm vụ trước, bạn đã thêm một explicit intent vào MainActivity để khởi chạy SecondActivity. Bạn cũng có thể sử dụng một intent để gửi dữ liệu từ một activity sang activity khác trong quá trình khởi chạy.

Intent có thể truyền dữ liệu đến activity đích theo hai cách:

- Trong trường "data":
  - Dữ liệu intent là một URI chỉ định dữ liệu cụ thể cần được xử lý.
- Trong "intent extras":
  - Nếu thông tin bạn muốn truyền đến một activity thông qua intent không phải là một URI, hoặc bạn có nhiều hơn một thông tin cần gửi, bạn có thể đặt thông tin bổ sung vào extras.
  - Intent extras là các cặp key/value trong một Bundle.
  - Bundle là một tập hợp dữ liệu, được lưu dưới dạng cặp key/value.

Để truyền thông tin từ một activity sang activity khác, bạn đặt các key/value vào intent extra Bundle từ activity gửi, sau đó lấy chúng ra trong activity nhận.

Trong nhiệm vụ này, bạn sẽ chỉnh sửa explicit intent trong MainActivity để bao gồm dữ liệu bổ sung (cụ thể là một chuỗi do người dùng nhập vào) trong intent extra Bundle. Sau đó, bạn chỉnh sửa SecondActivity để lấy dữ liệu từ intent extra Bundle và hiển thị nó trên màn hình.

### 3.1 Thêm một EditText vào bố cục của MainActivity

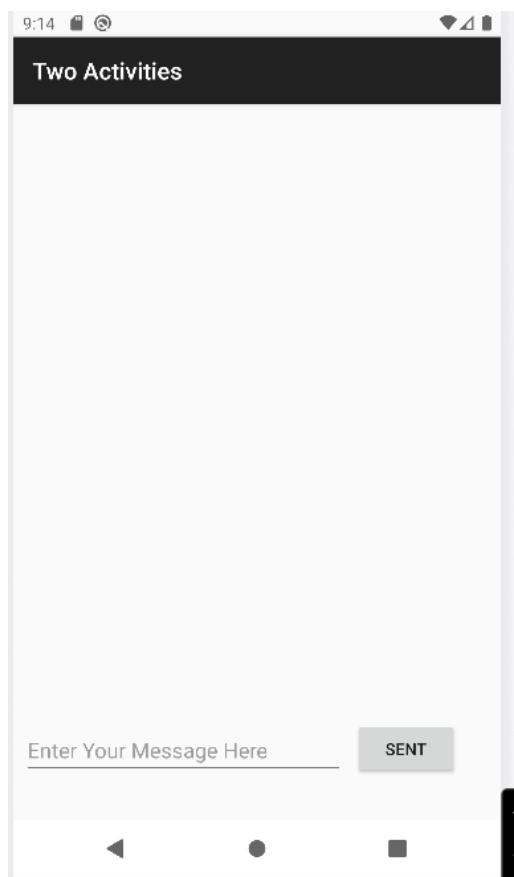
1. Mở activity\_main.xml.
2. Kéo một phần tử Plain Text (EditText) từ Palette vào phía dưới của bố cục, và thêm ràng buộc (constraints) vào:
  - Bên trái của bố cục.
  - Bên dưới của bố cục.
  - Bên trái của nút Send.

Thiết lập các thuộc tính trong **Attributes** như sau:

Thuộc tính	Giá trị
id	editText_main

Thuộc tính	Giá trị
Right margin	8
Left margin	8
Bottom margin	16
layout_width	match_constraint
layout_height	wrap_content
inputType	textLongMessage
hint	"Enter Your Message Here"
text	(Xóa bất kỳ văn bản nào trong trường này)

Sau khi thêm, bố cục trong activity\_main.xml sẽ trông như sau:



3. Nhấp vào tab Text để chỉnh sửa mã XML,  
và trích xuất chuỗi "Enter Your Message Here" vào resource có tên editText\_main.

Mã XML của bố cục sẽ 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	btnSent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="36dp"
        android:layout_marginBottom="36dp"
        android:text="SENT"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:onClick="launchSecondActivity"/>

    <EditText
        android:id="@+id/editText_main"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="36dp"
        android:ems="10"
        android:inputType="textLongMessage"
        android:text=""
        android:hint="Enter Your Message Here"
        android:layout_marginRight="8dp"
        android:layout_marginLeft="8dp"
        app:layout_goneMarginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id	btnSent"
        app:layout_constraintHorizontal_bias="0.506"
        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** trong một **Bundle**. Một **Bundle** là tập hợp dữ liệu, được lưu trữ dưới dạng cặp **key/value**. Để truyền thông tin từ **Activity** này sang **Activity** khác, bạn đặt các cặp **key/value** vào **Intent extra Bundle** từ **Activity** gửi và sau đó lấy chúng ra trong **Activity** nhận.

#### 1. Mở **MainActivity**.

- Thêm một hằng số **public** ở đầu lớp để xác định khóa (**key**) cho Intent extra.

```
no usages
public static final String EXTRA_MESSAGE = "com.example.android.twoactivities.extra.MESSAGE";
```

- Thêm một biến **private** ở đầu lớp để lưu trữ **EditText**.

```
private EditText mMessageEditText;
```

- Trong phương thức **onCreate()**, sử dụng **findViewById()** để lấy tham chiếu đến **EditText** và gán nó cho biến **private** vừa tạo.

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

- Trong phương thức **launchSecondActivity()**, ngay dưới dòng khởi tạo **Intent**, lấy văn bản từ **EditText** dưới dạng chuỗi (**string**).

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

- Thêm chuỗi đó vào **Intent** dưới dạng **extra**, với **EXTRA\_MESSAGE** là khóa (**key**) và chuỗi đó là giá trị (**value**).

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

Phương thức **onCreate()** trong **MainActivity** bây giờ sẽ trông giống như đoạn mã sau:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);
    mMessageEditText = (EditText) findViewById(R.id.editText_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;
        });
}
```

Phương thức **launchSecondActivity()** trong **MainActivity** bây giờ sẽ trông giống như đoạn mã sau:

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

### 3.3 Thêm một TextView vào SecondActivity để hiển thị thông điệp

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

Attribute	Value
id	text_message
Top margin	8
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	(Xóa bất kỳ văn bản nào trong trường này)
textAppearance	AppCompat.Medium

Giao diện mới trông vẫn giống như trước, vì **TextView** mới chưa chứa bất kỳ văn bản nào, do đó nó chưa hiển thị trên màn hình.

Mã XML cho **activity\_second.xml** nên trông giống như đoạn mã 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=".activity_second">  
  
    <TextView  
        android:id="@+id/text_hear"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_hear"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text=""
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_hear" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

### 3.4 Chính sửa SecondActivity để lấy extras và hiển thị thông điệp

1. Mở **SecondActivity** để thêm mã vào phương thức **onCreate()**.
2. Lấy Intent đã kích hoạt **Activity** này.

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

3. Lấy chuỗi chứa thông điệp từ **Intent extras**, sử dụng biến tĩnh **MainActivity.EXTRA\_MESSAGE** làm khóa (**key**).

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

4. Sử dụng **findViewById()** để lấy tham chiếu đến **TextView** trong giao diện.

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

5. Đặt nội dung của **TextView** thành chuỗi lấy từ **Intent extra**.

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

6. Chạy ứng dụng. Khi bạn nhập một thông điệp trong **MainActivity** và nhấn **Send**, **SecondActivity** sẽ mở ra và hiển thị thông điệp.

Phương thức **onCreate()** trong **SecondActivity** nên trông giống như đoạn mã sau:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```

        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_second);
        Intent intent = getIntent();
        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
        TextView textView = findViewById(R.id.text_message);
        textView.setText(message);
        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;
        });
    }
}

```

## Nhiệm vụ 4: Trả dữ liệu về MainActivity

Bây giờ bạn đã có một ứng dụng có thể mở một **Activity** mới và gửi dữ liệu đến nó, bước cuối cùng là trả dữ liệu từ **SecondActivity** về **MainActivity**. Bạn cũng sẽ sử dụng **Intent** và **Intent extras** cho nhiệm vụ này.

### 4.1 Thêm một EditText và một Button vào giao diện của SecondActivity

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

```

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

```

2. Mở **activity\_main.xml** và **activity\_second.xml**.
3. Sao chép **EditText** và **Button** từ tệp giao diện **activity\_main.xml** và Dán chúng vào **activity\_second.xml**.
4. Trong **activity\_second.xml**, chỉnh sửa các giá trị thuộc tính (**attribute values**) cho **Button** như sau:

Giá trị thuộc tính cũ	Giá trị thuộc tính mới
android:id="@+id/button_main"	android:id="@+id/button_second"
android:onClick= "launchSecondActivity"	android:onClick="returnReply"
android:text= "@string/button_main"	android:text= "@string/button_second"

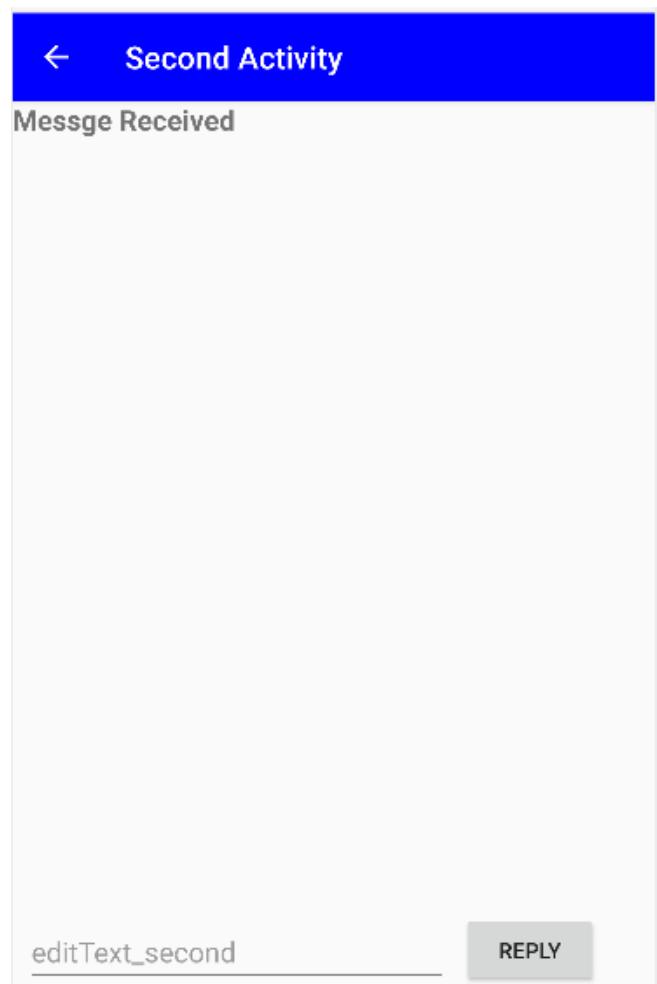
5. Trong **activity\_second.xml**, chỉnh sửa các giá trị thuộc tính cho **EditText** như sau:

Giá trị thuộc tính cũ	Giá trị thuộc tính mới
android:id="@+id/editText_main"	android:id="@+id/editText_second"
app:layout_constraintEnd_toStartOf="@+id/button"	app:layout_constraintEnd_toStartOf="@+id/button_second"
android:hint= "@string/editText_main"	android:hint= "@string/editText_second"

6. Trong trình chỉnh sửa giao diện XML, nhấp vào **returnReply**, nhấn **Alt + Enter** (**Option + Return** trên Mac), và chọn **Create 'returnReply(View)' in 'SecondActivity'**.

**Android Studio** sẽ tự động tạo một phương thức xử lý **returnReply()**, bạn sẽ triển khai phương thức này trong nhiệm vụ tiếp theo.

Giao diện mới của **activity\_second.xml** sẽ trông như sau:



Mã XML cho tệp giao diện **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=".activity_second">

    <Button
        android:id="@+id/button_second"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="36dp"
        android:layout_marginBottom="36dp"
        android:text="@string/button_second"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:onClick="returnReply"/>
    <EditText
        android:id="@+id/editText_second"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="36dp"
        android:ems="10"
        android:inputType="textLongMessage"
        android:text=""
        android:hint="editText_second"
        android:layout_marginRight="8dp"
        android:layout_marginLeft="8dp"
        app:layout_goneMarginBottom="16dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button_second"
        app:layout_constraintHorizontal_bias="0.506"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/text_hear"

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_hear"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

```
        android:text=""
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_hear" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

## 4.2 Tạo một Intent phản hồi trong SecondActivity

Dữ liệu phản hồi từ **SecondActivity** về **MainActivity** được gửi trong một **Intent extra**. Bạn sẽ tạo **Intent** phản hồi này và đặt dữ liệu vào đó theo cách tương tự như khi gửi dữ liệu ban đầu.

1. Mở **SecondActivity**.
2. Ở đầu lớp, thêm một hằng số **public** để định nghĩa khóa (**key**) cho **Intent extra**.

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

3. Thêm một biến **private** ở đầu lớp để lưu **EditText**.

```
private TextView mReply;
```

4. Trong phương thức **onCreate()**, trước khi viết mã cho **Intent**, sử dụng **findViewById()** để lấy tham chiếu đến **EditText** và gán nó vào biến **private** đã tạo.

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

5. Trong phương thức **returnReply()**, lấy văn bản từ **EditText** dưới dạng chuỗi (**string**).

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

6. Trong phương thức **returnReply()**, tạo một **Intent** mới để phản hồi—không tái sử dụng đối tượng **Intent** nhận được từ yêu cầu ban đầu.

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

7. Thêm chuỗi **reply** từ **EditText** vào **Intent** mới dưới dạng một **Intent extra**. Vì **extras** là cặp **key/value**, khóa ở đây là **EXTRA\_REPLY**, và giá trị là **reply**.

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

8. Đặt kết quả là **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\_CANCELED**.

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

9. Gọi **finish()** để đóng **Activity** và quay lại **MainActivity**.

```
finish();
```

Mã của **SecondActivity** sau khi chỉnh sửa sẽ trông như sau:

```
public class activity_second extends AppCompatActivity {
    public static final String EXTRA_REPLY =
"com.example.android.twoactivities.extra.REPLY";
    private TextView mReply;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_second);
        mReply = (TextView) findViewById(R.id.text_message);
        Intent intent = getIntent();
        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
        TextView textView = findViewById(R.id.text_message);
        textView.setText(message);
        ActionBar actionBar = getSupportActionBar();
        if(actionBar != null){
            actionBar.setBackgroundDrawable(new ColorDrawable(Color.BLUE));
        }
    }
    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 về. Trong nhiệm vụ này, bạn sẽ thêm các phần tử **TextView** vào tệp **activity\_main.xml** để hiển thị phản hồi trong **MainActivity**.

Để thực hiện nhanh hơn, bạn có thể sao chép các phần tử **TextView** đã dùng trong **SecondActivity**.

1. Mở **strings.xml** và thêm một tài nguyên chuỗi (**string resource**) cho tiêu đề phản hồi.

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

2. Mở **activity\_main.xml** và **activity\_second.xml**.

- Sao chép hai phần tử **TextView** từ tệp bô cục **activity\_second.xml** và dán chúng vào **activity\_main.xml**, đặt ở trên **Button**.
- Trong **activity\_main.xml**, chỉnh sửa giá trị thuộc tính cho **TextView** thứ nhất như sau:

Giá trị thuộc tính cũ	Giá trị thuộc tính mới
android:id="@+id/text_header"	android:id="@+id/text_header_reply"
android:text= "@string/text_header"	android:text= "@string/text_header_reply"

- Trong **activity\_main.xml**, chỉnh sửa giá trị thuộc tính cho **TextView** thứ hai như sau:

Giá trị thuộc tính cũ	Giá trị thuộc tính mới
android:id="@+id/text_message"	android:id= "@+id/text_message_reply"
app:layout_constraintTop_toBottomOf="@+id/text_header"	app:layout_constraintTop_toBottomOf= "@+id/text_header_reply"

- Thêm thuộc tính **android:visibility** vào từng **TextView** để làm cho chúng **ẩn đi ban đầu**. (Hiển thị chúng trên màn hình nhưng không có nội dung có thể gây nhầm lẫn cho người dùng).

```
    android:visibility="invisible"
```

Bạn sẽ làm cho các phần tử **TextView** này hiển thị sau khi dữ liệu phản hồi được truyền về từ **SecondActivity**.

Tệp **activity\_main.xml** vẫn trông giống như trong nhiệm vụ trước—mặc dù bạn đã thêm hai phần tử **TextView** vào bô cục. Vì bạn đã đặt chúng ở trạng thái **invisible**, nên chúng **không xuất hiện trên màn hình**.

Dưới đây là mã XML của 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:text="@string/text_header_reply"
    android:textAppearance="@style/TextAppearance.AppCompat.Medium"
    android:textStyle="bold"
    android:visibility="invisible"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

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

<Button
    android:id="@+id/btnSent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="36dp"
    android:layout_marginBottom="36dp"
    android:text="SENT"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:onClick="launchSecondActivity"/>

<EditText
    android:id="@+id/editText_main"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="36dp"
    android:ems="10"
    android:inputType="textLongMessage"
    android:text=""
    android:hint="Enter Your Message Here"
    android:layout_marginRight="8dp"
    android:layout_marginLeft="8dp"
    app:layout_goneMarginBottom="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/btnSent"
    app:layout_constraintHorizontal_bias="0.506"
    app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

## 4.4 Nhận phản hồi từ Intent extra và hiển thị nó

Khi bạn sử dụng một **Intent** tường minh để khởi chạy một **Activity** khác, có thể bạn không mong đợi nhận lại dữ liệu—you chỉ đơn thuần là kích hoạt **Activity** đó. Trong trường hợp này, bạn sử dụng **startActivity()** để khởi chạy **Activity** mới, giống như đã làm trước đó trong bài thực hành này.

Tuy nhiên, nếu bạn muốn nhận dữ liệu từ **Activity** được kích hoạt, bạn cần khởi chạy nó bằng **startActivityForResult()**.

Trong nhiệm vụ này, bạn sẽ chỉnh sửa ứng dụng để khởi chạy **SecondActivity**, mong đợi một kết quả phản hồi, trích xuất dữ liệu phản hồi 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 công khai ở đầu lớp để xác định khóa cho loại phản hồi cụ thể mà bạn quan tâm.

```
public static final int TEXT_REQUEST = 1;
```

3. **Thêm** hai biến riêng tư để giữ phần tiêu đề phản hồi và phần tử **TextView** phản hồi.

```
private TextView mReplyHeadTextView;  
private TextView mReplyTextView;
```

4. Trong phương thức **onCreate()**, sử dụng **findViewById()** để lấy tham chiếu từ bộ cục đến các phần tử **TextView** của tiêu đề phản hồi và phản hồi. Gán chúng vào các biến riêng tư.

```
mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);  
mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
```

Phương thức **onCreate()** đây đủ bây giờ sẽ trông như thế này:

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    EdgeToEdge.enable(this);  
    setContentView(R.layout.activity_main);  
    mMessageEditText = (EditText) findViewById(R.id.editText_main);  
    mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);  
    mReplyTextView = (TextView) findViewById(R.id.text_message_reply);  
}
```

5. Trong phương thức launchSecondActivity(), thay đổi lệnh gọi startActivity() thành startActivityForResult(), và truyền khóa **TEXT\_REQUEST** làm tham số.

```
startActivityForResult(intent, TEXT_REQUEST);
```

6. **Ghi đè** phương thức onActivityResult() với ba đối số:

```
@Override  
public void onActivityResult(int requestCode, int resultCode, Intent data) {  
}
```

- requestCode: Mã yêu cầu được đặt khi khởi chạy **Activity** bằng startActivityForResult().
  - resultCode: Mã kết quả do **Activity** được khởi chạy trả về (RESULT\_OK hoặc RESULT\_CANCELED).
  - Intent data: Chứa dữ liệu phản hồi từ **Activity**.
7. Bên trong onActivityResult(), gọi super.onActivityResult().

```
super.onActivityResult(requestCode, resultCode, data);
```

8. Kiểm tra TEXT\_REQUEST để đảm bảo xử lý đúng Intent phản hồi, và kiểm tra RESULT\_OK để xác nhận yêu cầu thành công.

```
if(requestCode == TEXT_REQUEST) {  
    if(resultCode == RESULT_OK) {  
        //  
    }  
}
```

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

9. Lấy Intent extra từ phản hồi Intent (data). Ở đây, khóa cho extra là hằng số EXTRA\_REPLY từ **SecondActivity**.

```
String reply = data.getStringExtra(activity_second.EXTRA_REPLY);
```

10. Thiết lập hiển thị của tiêu đề phản hồi thành true.

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

11. Cập nhật văn bản của **TextView** phản hồi với nội dung phản hồi và đặt thuộc tính hiển thị thành true.

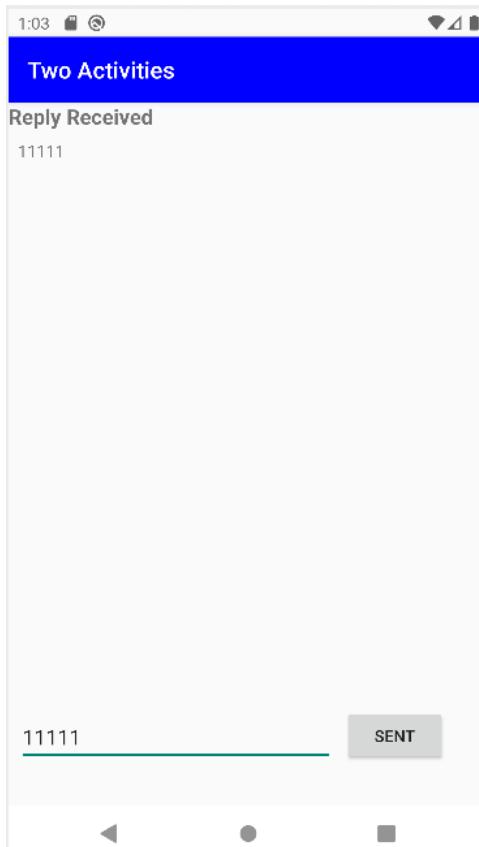
```
mReplyTextView.setText(reply);  
mReplyTextView.setVisibility(View.VISIBLE);
```

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

```
@Override  
public void onActivityResult(int requestCode, int resultCode, Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
    if(requestCode == TEXT_REQUEST){  
        if(resultCode == RESULT_OK){  
            String reply = data.getStringExtra(activity_second.EXTRA_REPLY);  
            mReplyHeadTextView.setVisibility(View.VISIBLE);  
            mReplyTextView.setText(reply);  
            mReplyTextView.setVisibility(View.VISIBLE);  
        }  
    }  
}
```

## 12. Chạy ứng dụng.

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



## Mã giải pháp

**Dự án Android Studio:** *TwoActivities*

## Thử thách lập trình

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

**Thử thách:** Tạo một ứng dụng với ba phần tử Button được gán nhãn **Text One**, **Text Two**, và **Text Three**.

Khi nhấp vào bất kỳ Button nào, hãy khởi chạy một Activity thứ hai. Activity thứ hai này sẽ chứa một ScrollView hiển thị một trong ba đoạn văn bản (bạn có thể chọn đoạn văn bản tùy ý). Sử dụng Intent để khởi chạy Activity thứ hai cùng với *extras* nhằm xác định đoạn văn nào sẽ được hiển thị.

## Tóm tắt

**Tổng quan:**

- Một Activity là một thành phần ứng dụng cung cấp một màn hình duy nhất tập trung vào một tác vụ của người dùng.
- Mỗi Activity có một tệp bố cục giao diện người dùng riêng.
- Bạn có thể thiết lập mối quan hệ cha/con giữa các Activity để bật tính năng điều hướng *Up* trong ứng dụng.
- Một View có thể được đặt thành hiển thị hoặc ẩn bằng thuộc tính `android:visibility`.

## Khái niệm liên quan

Tài liệu về khái niệm liên quan nằm trong **2.1: Activities and intents**.

## Tìm hiểu thêm

**Tài liệu về Android Studio:**

- Giới thiệu về Android Studio

**Tài liệu dành cho nhà phát triển Android:**

- Các nguyên tắc cơ bản của ứng dụng (*Application Fundamentals*)
- Activity
- Intent và *Intent Filters*

- Thiết kế điều hướng *Back* và *Up*
- Activity
- Intent
- ScrollView
- View
- Button
- TextView
- Tài nguyên chuỗi (String resources)

## 2.2) 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 cho đến khi bị hủy và hệ thống thu hồi tài nguyên của nó. Khi người dùng điều hướng giữa các Activity trong ứng dụng (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 Activity có một phương thức gọi lại tương ứng: `onCreate()`, `onStart()`, `onPause()` và các phương thức khác. Khi một Activity thay đổi trạng thái, phương thức gọi lại tương ứng sẽ được gọi. Bạn đã từng thấy một trong những phương thức này: `onCreate()`. Bằng cách ghi đè các phương thức gọi lại này trong lớp Activity, bạn có thể thay đổi hành vi mặc định của Activity để phản ứng với hành động của người dùng hoặc hệ thống.

Trạng thái của Activity cũng có thể thay đổi khi có sự thay đổi về cấu hình thiết bị, chẳng hạn như khi người dùng xoay thiết bị từ chế độ dọc sang chế độ ngang. Khi điều này xảy ra, Activity sẽ bị hủy và được tạo lại ở trạng thái mặc định, có thể khiến người dùng mất dữ liệu đã nhập trong Activity.

Để tránh gây nhầm lẫn cho người dùng, bạn cần phát triển ứng dụng sao cho dữ liệu không bị mất ngoài ý muốn. Trong bài thực hành này, bạn sẽ thử nghiệm với các thay đổi về cấu hình và tìm hiểu cách giữ lại trạng thái của Activity khi có thay đổi về cấu hình thiết bị hoặc các sự kiện vòng đời khác.

Trong bài thực hành này, bạn sẽ thêm các câu lệnh ghi nhật ký vào ứng dụng `TwoActivities` và quan sát các thay đổi trong vòng đời Activity khi sử dụng ứng dụng. Sau đó, bạn sẽ bắt đầu làm việc với những thay đổi này và tìm hiểu cách xử lý đầu vào của người dùng trong các tình huống này.

## Những kiến thức bạn cần có trước

Bạn nên có khả năng:

- Tạo và chạy một dự án ứng dụng trong Android Studio.
- Thêm các câu lệnh log vào ứng dụng và xem nhật ký trong Logcat.
- Hiểu và làm việc với Activity và Intent, cũng như thoái mái trong việc 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 gọi lại liên quan đến các thay đổi trong vòng đời Activity.
- Ảnh hưởng của các hành động (như thay đổi cấu hình) đối với các sự kiện vòng đời Activity.
- Cách giữ lại trạng thái Activity khi có sự kiện vòng đời xảy ra.

## Những gì bạn sẽ làm

- Thêm mã vào ứng dụng TwoActivities từ bài thực hành trước để triển khai các phương thức gọi lại vòng đời Activity, bao gồm các câu lệnh ghi nhật ký.
- Quan sát sự thay đổi trạng thái của Activity khi ứng dụng chạy và khi bạn tương tác với từng Activity trong ứng dụng.
- Chính sửa ứng dụng để giữ lại trạng thái của Activity trong trường hợp nó bị hủy ngoài ý muốn do hành động của người dùng hoặc thay đổi cấu hình thiết bị.

## Tổng quan về ứng dụng

Trong bài thực hành này, bạn sẽ mở rộng ứng dụng TwoActivities. Ứng dụng này vẫn có giao diện và hoạt động tương tự như trong bài thực hành trước. Nó chứa hai Activity và cho phép người dùng gửi dữ liệu giữa chúng.

Những thay đổi bạn thực hiện trong bài thực hành này sẽ không ảnh hưởng đến hành vi hiển thị của ứng dụng mà tập trung vào cách ứng dụng xử lý các sự kiện vòng đời Activity.

### Nhiệm vụ 1: Thêm các phương thức gọi lại vòng đời vào TwoActivities

Trong nhiệm vụ này, bạn sẽ triển khai tất cả các phương thức gọi lại vòng đời của **Activity** để in thông báo ra **Logcat** khi các phương thức này được gọi. Những thông báo

này sẽ giúp bạn quan sát khi nào trạng thái vòng đời của **Activity** thay đổi và cách những thay đổi đó ảnh hưởng đến ứng dụng khi chạy.

## 1.1 (Tùy chọn) Sao chép dự án TwoActivities

Đối với các nhiệm vụ trong bài thực hành này, bạn sẽ chỉnh sửa dự án **TwoActivities** 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 Triển khai các phương thức gọi lại vào MainActivity

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

```
Log.d(LOG_TAG, "-----");
Log.d(LOG_TAG, "onCreate");
```

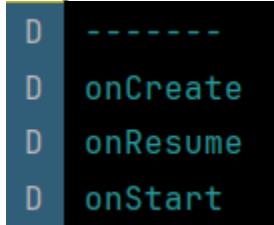
3. Ghi đè phương thức gọi lại `onStart()` và thêm một câu lệnh ghi log để theo dõi sự kiện này:

```
@Override
protected void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}
```

- Để thao tác nhanh hơn, hãy chọn **Code > Override Methods** trong **Android Studio**.
  - Một hộp thoại xuất hiện với danh sách tất cả các phương thức mà bạn có thể ghi đè trong lớp của mình.
  - Chọn một hoặc nhiều phương thức gọi lại từ danh sách này sẽ tự động chèn một mẫu hoàn chỉnh cho các phương thức đó, bao gồm cả lệnh gọi phương thức lớp cha (`super.methodName()`).
4. Sử dụng phương thức `onStart()` làm mẫu để triển khai các phương thức gọi lại vòng đời khác, bao gồm:
    - `onPause()`
    - `onRestart()`
    - `onResume()`
    - `onStop()`
    - `onDestroy()`

**Lưu ý:** Tất cả các phương thức gọi lại đều có cùng một mẫu (chỉ khác tên). Nếu bạn sao chép và dán phương thức onStart() để tạo các phương thức khác, hãy nhớ cập nhật nội dung để gọi đúng phương thức trong lớp cha và ghi log đúng tên phương thức.

5. Chạy ứng dụng.
6. Nhấn vào tab **Logcat** ở cuối Android Studio để hiển thị **Logcat**. Bạn sẽ thấy **ba thông báo log** hiển thị các trạng thái vòng đời mà **Activity** đã trải qua khi khởi động:



```
D -----
D onCreate
D onResume
D onStart
```

### 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:

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

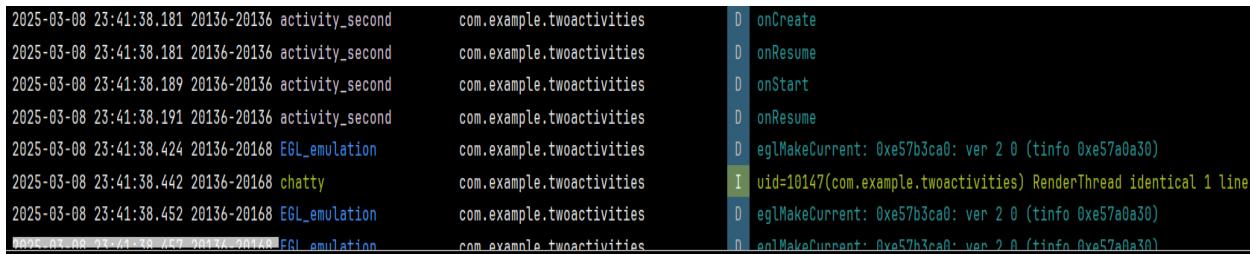
3. Thêm các phương thức callback vòng đời và câu lệnh log vào SecondActivity. (Bạn có thể sao chép và dán các phương thức callback từ MainActivity).
4. Thêm một câu lệnh log vào phương thức returnReply() ngay trước khi gọi finish().

```
Log.d(LOG_TAG, msg: "End SecondActivity");
```

### 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.



```
2025-03-08 23:41:38.181 20136-20136 activity_second com.example.twoactivities D onCreate
2025-03-08 23:41:38.181 20136-20136 activity_second com.example.twoactivities D onResume
2025-03-08 23:41:38.189 20136-20136 activity_second com.example.twoactivities D onStart
2025-03-08 23:41:38.191 20136-20136 activity_second com.example.twoactivities D onResume
2025-03-08 23:41:38.424 20136-20168 EGL_emulation com.example.twoactivities D eglGetCurrent: 0xe57b3ca0: ver 2 0 (tinfo 0xe57a0a30)
2025-03-08 23:41:38.442 20136-20168 chatty com.example.twoactivities I uid=10147(com.example.twoactivities) RenderThread identical 1 line
2025-03-08 23:41:38.452 20136-20168 EGL_emulation com.example.twoactivities D eglGetCurrent: 0xe57b3ca0: ver 2 0 (tinfo 0xe57a0a30)
2025-03-08 23:41:38.452 20136-20168 EGL_emulation com.example.twoactivities D eglGetCurrent: 0xe57b3ca0: ver 2 0 (tinfo 0xe57a0a30)
```

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()**:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);
    mMessageEditText = (EditText) findViewById(R.id.editText_main);
    ActionBar actionBar = getSupportActionBar();
    if(actionBar != null){
        actionBar.setBackgroundDrawable(new ColorDrawable(Color.BLUE));
    }
    mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
```

```

mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
Log.d(LOG_TAG, "-----");
Log.d(LOG_TAG, "onCreate");
Log.d(LOG_TAG, "onStart");
Log.d(LOG_TAG, "onResume");
Log.d(LOG_TAG, "onPause");
Log.d(LOG_TAG, "onStop");
Log.d(LOG_TAG, "onDestroy");
Log.d(LOG_TAG, "onRestart");
Log.d(LOG_TAG, "onCreate");
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;
});
}

```

Các phương thức vòng đời khác:

```

@Override
public void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "onStart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "onResume");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(LOG_TAG, "onPause");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "onStop");
}

@Override
protected void onRestart() {
    super.onRestart();
    Log.d(LOG_TAG, "onRestart");
}

```

```
}  
  
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    Log.d(LOG_TAG, "onDestroy");  
}
```

## SecondActivity

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

Ở đâu lớp **SecondActivity**:

```
private static final String LOG_TAG = activity_second.class.getSimpleName();  
public void returnReply(View view) {  
    String reply = mReply.getText().toString();  
    Intent replyIntent = new Intent();  
    replyIntent.putExtra(EXTRA_REPLY, reply);  
    setResult(RESULT_OK, replyIntent);  
    Log.d(LOG_TAG, "End SecondActivity");  
    finish();  
}
```

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 trả 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.

```
@Override  
public void onSaveInstanceState(@NonNull Bundle outState) {  
    super.onSaveInstanceState(outState);  
}
```

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**":

```
if(mReplyHeadTextView.getVisibility() == View.VISIBLE) {  
    outState.putBoolean("reply_visible", true);  
}
```

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.

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

```
@Override  
public void onSaveInstanceState(@NonNull Bundle outState) {  
    super.onSaveInstanceState(outState);  
    if(mReplyHeadTextView.getVisibility() == View.VISIBLE){  
        outState.putBoolean("reply_visible", true);  
        outState.putString("reply_visible", mReplyTextView.getText().toString());  
    }  
}
```

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.

- 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**.

```
mMessageEditText = (EditText) findViewById(R.id.editText_main);
mReplyHeaderView = (TextView) findViewById(R.id.text_header_reply);
mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
if (savedInstanceState != 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()**.

- Bên trong kiểm tra đó, lấy trạng thái hiển thị hiện tại (đúng hoặc sai) từ **Bundle** với khóa "**reply\_visible**".

```
if (savedInstanceState != null){  
    boolean isVisible = savedInstanceState.getBoolean(key: "reply_visible");  
}
```

3. Thêm một kiểm tra bên dưới dòng trước đó cho biến **isVisible**.

```
if (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.

4. Bên trong kiểm tra **isVisible**, làm cho tiêu đề hiển thị.

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

5. Lấy văn bản trả lời từ **Bundle** với khóa "reply\_text", và đặt **TextView** trả lời để hiển thị chuỗi đó.

```
mReplyTextView.setText(savedInstanceState.getString(key: "reply_text"));
```

6. Làm cho **TextView** trả lời hiển thị nữa:

```
mReplyTextView.setVisibility(View.VISIBLE);
```

1. Chạy ứng dụng. Thủ 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()**:

```

@Override
public void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    if(mReplyHeadTextView.getVisibility() == View.VISIBLE){
        outState.putBoolean("reply_visible", true);
        outState.putString("reply_visible", mReplyTextView.getText().toString());
    }
}

```

Phuong thuc onCreate():

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);
    ActionBar actionBar = getSupportActionBar();
    if(actionBar != null){
        actionBar.setBackgroundDrawable(new ColorDrawable(Color.BLUE));
    }
    mMessageEditText = (EditText) findViewById(R.id.editText_main);
    mReplyHeadTextView = (TextView) findViewById(R.id.text_header_reply);
    mReplyTextView = (TextView) findViewById(R.id.text_message_reply);
    if (savedInstanceState != null){
        boolean isVisible = savedInstanceState.getBoolean("reply_visible");
        if (isVisible){
            mReplyHeadTextView.setVisibility(View.VISIBLE);

            mReplyTextView.setText(savedInstanceState.getString("reply_text"));
            mReplyTextView.setVisibility(View.VISIBLE);
        }
    }
    Log.d(LOG_TAG, "-----");
    Log.d(LOG_TAG,"onCreate");
    Log.d(LOG_TAG,"onStart");
    Log.d(LOG_TAG,"onResume");
    Log.d(LOG_TAG,"onPause");
    Log.d(LOG_TAG,"onStop");
    Log.d(LOG_TAG,"onDestroy");
    Log.d(LOG_TAG,"onRestart");
    Log.d(LOG_TAG,"onCreate");
    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;
    });
}

```

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().

### **Khái niệm liên quan**

Tài liệu về khái niệm liên quan có trong mục 2.2: Vòng đời và trạng thái của Activity.

### **Tìm hiểu thêm**

Tài liệu của Android Studio:

- Làm quen với Android Studio

Tài liệu của nhà phát triển Android:

- Các nguyên lý cơ bản của ứng dụng
- Hoạt động (Activities)
- Hiểu về vòng đời của Activity
- Intents và Bộ lọc Intent
- Xử lý các thay đổi cấu hình
- Activity
- Intent

### **Bài tập về nhà**

Xây dựng và chạy một ứng dụng

1. Tạo một ứng dụng với một layout chứa một TextView đếm số, một nút Button để tăng giá trị của bộ đếm, và một EditText. Xem ảnh chụp màn hình dưới đây làm ví dụ. Bạn không cần phải sao chép chính xác layout.
2. Thêm một trình xử lý sự kiện khi nhấn nút Button để tăng bộ đếm.
3. Chạy ứng dụng và tăng bộ đếm. Nhập một số văn bản vào EditText.
4. Xoay thiết bị. Lưu ý rằng bộ đếm bị đặt lại, nhưng EditText thì không.
5. Triển khai phương thức onSaveInstanceState() để lưu lại trạng thái hiện tại của ứng dụng.
6. Cập nhật phương thức onCreate() để khôi phục trạng thái của ứng dụng.

7. Đảm bảo rằng khi bạn xoay thiết bị, trạng thái của ứng dụng vẫn được bảo lưu.

### Câu hỏi 1

Nếu bạn chạy ứng dụng bài tập trước khi triển khai phương thức onSaveInstanceState(), điều gì sẽ xảy ra khi bạn xoay thiết bị? Chọn một đáp án:

- EditText không còn chứa văn bản bạn đã nhập, nhưng bộ đếm vẫn được bảo lưu.
- Bộ đếm bị đặt lại về 0, và EditText không còn chứa văn bản bạn đã nhập.
- Bộ đếm bị đặt lại về 0, nhưng nội dung của EditText được bảo lưu.
- Bộ đếm và nội dung của EditText đều được bảo lưu.

### Câu hỏi 2

Các phương thức vòng đời của Activity nào được gọi khi có thay đổi cấu hình thiết bị (chẳng hạn như xoay màn hình)? Chọn một đáp án:

- Android ngay lập tức tắt Activity của bạn bằng cách gọi onStop(). Mã của bạn phải khởi động lại Activity.
- Android tắt Activity của bạn bằng cách gọi onPause(), onStop(), và onDestroy(). Mã của bạn phải khởi động lại Activity.
- Android tắt Activity của bạn bằng cách gọi onPause(), onStop(), và onDestroy(), sau đó khởi động lại Activity và gọi onCreate(), onStart(), và onResume().
- Android ngay lập tức gọi onResume().

### Câu hỏi 3

Khi nào phương thức onSaveInstanceState() được gọi trong vòng đời của Activity? Chọn một đáp án:

- onSaveInstanceState() được gọi trước phương thức onStop().
- onSaveInstanceState() được gọi trước phương thức onResume().
- onSaveInstanceState() được gọi trước phương thức onCreate().
- onSaveInstanceState() được gọi trước phương thức onDestroy().

### Câu hỏi 4

Các phương thức vòng đời của Activity nào là tốt nhất để lưu dữ liệu trước khi Activity kết thúc hoặc bị hủy? Chọn một đáp án:

- onPause() hoặc onStop()

- onResume() hoặc onCreate()
- onDestroy()
- onStart() hoặc onRestart()

### 2.3) Intent ngầm định

#### Giới thiệu

Trong một phần trước, bạn đã học về explicit intents (intents rõ ràng).

Trong một explicit intent, bạn thực hiện một hoạt động trong ứng dụng của bạn, hoặc trong một ứng dụng khác, bằng cách gửi một intent với tên lớp đầy đủ của hoạt động. Trong bài học này, bạn sẽ tìm hiểu thêm về implicit intents (intents gián tiếp) và cách sử dụng chúng để thực hiện các hoạt động.

Với một implicit intent, bạn khởi tạo một hoạt động mà không biết ứng dụng hoặc hoạt động nào sẽ xử lý tác vụ đó. Ví dụ, nếu bạn muốn ứng dụng của mình chụp một bức ảnh, gửi email, hoặc hiển thị một vị trí trên bản đồ, bạn thường không quan tâm ứng dụng hoặc hoạt động nào sẽ thực hiện tác vụ đó.

Ngược lại, hoạt động của bạn có thể khai báo một hoặc nhiều **intent filters** trong tệp AndroidManifest.xml để thông báo rằng hoạt động đó có thể chấp nhận implicit intents và định nghĩa các loại intents mà hoạt động sẽ chấp nhận.

Để khớp yêu cầu của bạn với một ứng dụng đã cài đặt trên thiết bị, hệ thống Android sẽ so khớp implicit intent của bạn với một hoạt động có intent filters chỉ ra rằng chúng có thể thực hiện hành động đó. Nếu có nhiều ứng dụng khớp, người dùng sẽ được hiển thị một trình chọn ứng dụng cho phép họ chọn ứng dụng mà họ muốn sử dụng để xử lý intent.

Trong bài thực hành này, bạn sẽ xây dựng một ứng dụng gửi một implicit intent để thực hiện các tác vụ sau:

- Mở một URL trong trình duyệt web.
- Mở một vị trí trên bản đồ.
- Chia sẻ văn bản.

Chia sẻ — gửi một mảnh thông tin cho những người khác qua email hoặc mạng xã hội — là một tính năng phổ biến trong nhiều ứng dụng. Để thực hiện hành động chia sẻ, bạn sử dụng lớp ShareCompat.IntentBuilder, giúp tạo một implicit intent để chia sẻ dữ liệu một cách dễ dàng.

Cuối cùng, bạn sẽ tạo một intent-receiver đơn giản để chấp nhận một implicit intent cho một hành động cụ thể.

## Những gì bạn đã biết

Bạn sẽ có khả năng:

- Sử dụng trình chỉnh sửa layout để thay đổi một layout.
- Chính sửa mã XML của một layout.
- Thêm một Button và một trình xử lý sự kiện nhấn nút.
- Tạo và sử dụng một Activity.
- Tạo và gửi một Intent giữa hai Activity.

## Những gì bạn sẽ học

- Cách tạo một Implicit Intent, và sử dụng các hành động và danh mục của nó.
- Cách sử dụng lớp trợ giúp ShareCompat.IntentBuilder để tạo một Implicit Intent cho việc chia sẻ dữ liệu.
- Cách quảng cáo ứng dụng của bạn có thể chấp nhận một Implicit Intent bằng cách khai báo các Intent filters trong tệp AndroidManifest.xml.

## Những gì bạn sẽ làm

- Tạo một ứng dụng mới để thử nghiệm với Implicit Intent.
- Triển khai một Implicit Intent để mở một trang web, và một Implicit Intent khác để mở một vị trí trên bản đồ.
- Triển khai một hành động để chia sẻ một đoạn văn bản.
- Tạo một ứng dụng mới có thể chấp nhận một Implicit Intent để mở một trang web.

## Tổng quan về ứng dụng

Trong phần này, bạn sẽ tạo một ứng dụng mới với một Activity và ba tùy chọn hành động: mở một trang web, mở một vị trí trên bản đồ, và chia sẻ một đoạn văn bản. Tất cả các trường văn bản đều có thể chỉnh sửa (EditText), nhưng chưa có giá trị mặc định.

### Nhiệm vụ 1: Tạo dự án và layout

Trong bài tập này, bạn sẽ tạo một dự án và ứng dụng mới có tên là **Implicit Intents**, với một layout mới.

#### 1.1 Tạo dự án

1. Mở Android Studio và tạo một dự án Android Studio mới. Đặt tên ứng dụng của bạn là **Implicit Intents**.
2. Chọn **Empty Activity** làm mẫu cho dự án. Nhấn **Next**.
3. Chấp nhận tên **Activity** mặc định là **MainActivity**. Đảm bảo ô **Generate Layout file** được chọn. Nhấn **Finish**.

## 1.2 Tạo layout

Trong nhiệm vụ này, bạn sẽ tạo layout cho ứng dụng. Sử dụng một **LinearLayout**, ba phần tử **Button**, và ba phần tử **EditText**, như sau:

1. Mở ứng dụng > res > values > strings.xml trong bảng **Project > Android**, và thêm các tài nguyên chuỗi sau:

```
<string name="app_name">Implicit Intents</string>
<string name="edittext_uri">http://deverloper.android.com</string>
<string name="edittext_log">Golden Gate Bridge</string>
<string name="button_uri">Open Website</string>
<string name="button_location">Open Location</string>
<string name="edittext_share">Twas brillig and the slithy toves</string>
<string name="button_share">Share This Text</string>
```

2. Mở res > layout > activity\_main.xml trong bảng **Project > Android**. Nhập vào tab **Text** để chuyển sang mã XML.
3. Thay đổi **android.support.constraint.ConstraintLayout** thành **LinearLayout**, như bạn đã học trong bài thực hành trước.
4. Thêm thuộc tính **android:orientation** với giá trị "vertical". Thêm thuộc tính **android:padding** với giá trị "16dp".

```
<?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"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">
```

5. Xóa **TextView** hiển thị "Hello World".

6. Thêm một bộ các phần tử giao diện người dùng cho nút **Open Website**. Bạn cần một phần tử EditText và một phần tử Button. Sử dụng các giá trị thuộc tính sau:

<b>EditText attribute</b>	<b>Value</b>
android:id	"@+id/website_edittext"
android:layout_width	"match_parent"
android:layout_height	"wrap_content"
android:text	"@string/edittext_uri"
<b>Button attribute</b>	<b>Value</b>
android:id	"@+id/open_website_button"
android:layout_width	"wrap_content"
android:layout_height	"wrap_content"
android:layout_marginBottom	"24dp"
android:text	"@string/button_uri"
android:onClick	"openWebsite"

Giá trị cho thuộc tính android:onClick sẽ vẫn được gạch chân đỏ cho đến khi bạn định nghĩa phương thức callback trong một nhiệm vụ sau.

7. **Thêm một bộ các phần tử giao diện người dùng** (EditText và Button) vào layout cho nút **Open Location**. Sử dụng các thuộc tính giống như trong bước trước, nhưng sửa đổi chúng như sau (Bạn có thể sao chép các giá trị từ nút **Open Website** và chỉnh sửa chúng)

<b>EditText attribute</b>	<b>Value</b>
android:id	"@+id/location_edittext"
android:text	"@string/edittext_loc"
<b>Button attribute</b>	<b>Value</b>
android:id	"@+id/open_location_button"
android:text	"@string/button_loc"
android:onClick	"openLocation"

Giá trị cho thuộc tính android:onClick sẽ vẫn được gạch chân đỏ cho đến khi bạn định nghĩa phương thức callback trong một nhiệm vụ sau.

8. **Thêm một bộ các phần tử giao diện người dùng (EditText và Button) vào layout cho nút Share This. Sử dụng các thuộc tính như dưới đây. (Bạn có thể sao chép các giá trị từ nút Open Website và chỉnh sửa chúng.)**

<b>EditText attribute</b>	<b>Value</b>
android:id	"@+id/share_edittext"
android:text	"@string/edittext_share"
<b>Button attribute</b>	<b>Value</b>
android:id	"@+id/share_text_button"
android:text	"@string/button_share"
android:onClick	"shareText"

Tùy thuộc vào phiên bản Android Studio của bạn, mã activity\_main.xml của bạn sẽ trông giống như sau. Các giá trị cho thuộc tính android:onClick sẽ vẫn được gạch chân đỏ cho đến khi bạn định nghĩa các phương thức callback trong một nhiệm vụ sau:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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">
```

```
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/website_edittext"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/edittext_uri" />
    <Button
        android:id="@+id/open_website_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_uri"
        android:layout_marginBottom="24dp"
        android:onClick="openWebsite"/>
    <EditText
        android:id="@+id/location_edittext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/edittext_log" />
    <Button
        android:id="@+id/open_location_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_location"
        android:layout_marginBottom="24dp"
        android:onClick="openLocation"/>
    <EditText
        android:id="@+id/share_edittext"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/edittext_share" />
    <Button
        android:id="@+id/share_text_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_share"
        android:layout_marginBottom="24dp"
        android:onClick="shareText"/>
</LinearLayout>
```

## Nhiệm vụ 2: Triển khai nút Open Website

Trong nhiệm vụ này, bạn triển khai phương thức xử lý sự kiện nhấn nút cho nút đầu tiên trong layout, **Open Website**. Hành động này sử dụng một **Implicit Intent** để gửi URI đã cho đến một **Activity** có thể xử lý **Implicit Intent** đó (chẳng hạn như trình duyệt web).

### 2.1 Định nghĩa openWebsite()

1. Nhập vào "openWebsite" trong mã XML activity\_main.xml.

2. Nhấn **Alt+Enter** (hoặc **Option+Enter** trên Mac) và chọn **Create 'openWebsite(View)' in 'MainActivity'**.

Tệp **MainActivity** sẽ mở, và Android Studio sẽ tạo một phương thức khung cho trình xử lý **openWebsite()**.

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

3. Trong **MainActivity**, thêm một biến private ở đầu lớp để lưu đối tượng **EditText** cho URI của trang web.

```
private EditText mWebsiteEditText;
```

4. Trong phương thức **onCreate()** của **MainActivity**, sử dụng **findViewById()** để lấy tham chiếu đến đối tượng **EditText** và gán nó cho biến private đó:

```
mWebsiteEditText = findViewById(R.id.website_edittext);
```

## 2.2 Thêm mã cho **openWebsite()**

- 1.Thêm một câu lệnh vào phương thức **openWebsite()** mới để lấy giá trị chuỗi từ **EditText**:

```
String url = mWebsiteEditText.getText().toString();
```

## 2.Mã hóa và phân tích chuỗi đó thành đối tượng Uri:

```
Uri webpage = Uri.parse(url);
```

3. Tạo một **Intent** mới với **Intent.ACTION\_VIEW** làm hành động và URI làm dữ liệu:

```
Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
```

Trình tạo **Intent** này khác với cái bạn đã sử dụng để tạo một **explicit Intent**. Trong trình tạo trước, bạn chỉ định ngữ cảnh hiện tại và một thành phần cụ thể (lớp **Activity**) để gửi **Intent**. Trong trình tạo này, bạn chỉ định một hành động và dữ liệu cho hành động đó. Các hành động được định nghĩa bởi lớp **Intent** và có thể bao gồm **ACTION\_VIEW** (để xem dữ liệu đã cho), **ACTION\_EDIT** (để chỉnh sửa dữ liệu đã cho), hoặc **ACTION\_DIAL** (để gọi một số điện thoại). Trong trường hợp này, hành động là **ACTION\_VIEW** vì bạn muốn hiển thị trang web được chỉ định bởi URI trong biến **webpage**.

4. Sử dụng phương thức **resolveActivity()** và trình quản lý gói Android để tìm một **Activity** có thể xử lý **Implicit Intent** của bạn. Đảm bảo rằng yêu cầu đã được giải quyết thành công:

```
if(intent.resolveActivity(getApplicationContext()) != null) {
```

Yêu cầu này sẽ so khớp hành động **Intent** và dữ liệu với các bộ lọc **Intent** của các ứng dụng đã cài đặt trên thiết bị. Bạn sử dụng nó để đảm bảo có ít nhất một **Activity** có thể xử lý yêu cầu của bạn.

5. Bên trong câu lệnh **if**, gọi **startActivity()** để gửi **Intent**:

```
startActivity(intent);
```

6. Thêm một khối **else** để in một thông báo Log nếu Intent không thể được giải quyết.

```
} else{
    Log.d( tag: "ImplicitIntents", msg: "Can't handle this!");
}
```

Phương thức **openWebsite()** bây giờ nên trông như sau. (Các chú thích đã được thêm vào để làm rõ.):

```
public void openWebsite(View view) {
    String url = mWebsiteEditText.getText().toString();
    Uri webpage = Uri.parse(url);
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);
    if(intent.resolveActivity(getApplicationContext()) != null) {
        startActivity(intent);
    } else{
        Log.d( tag: "ImplicitIntents", msg: "Can't handle this!");
    }
}
```

### Nhiệm vụ 3: Triển khai nút Mở Vị trí

Trong nhiệm vụ này, bạn sẽ triển khai phương thức xử lý sự kiện khi nhấp vào nút thứ hai trong giao diện người dùng, **Mở Vị trí**. Phương thức này gần như giống hệt với phương thức openWebsite(). Điểm khác biệt là việc sử dụng URI dạng geo để chỉ định một vị trí bản đồ. Bạn có thể sử dụng URI dạng geo với vĩ độ và kinh độ, hoặc sử dụng chuỗi truy vấn để chỉ định một vị trí chung. Trong ví dụ này, chúng tôi đã sử dụng cách thứ hai.

#### 3.1 Định nghĩa openLocation()

1. Nhập vào "openLocation" trong mã XML của **activity\_main.xml**.
2. Nhấn **Alt+Enter** (hoặc **Option+Enter** trên Mac) và chọn **Create 'openLocation(View)' in MainActivity.**
  - o Android Studio sẽ tự động tạo một phương thức khung xương trong **MainActivity** dành cho trình xử lý openLocation().

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

3. Thêm một biến riêng (private variable) ở đầu lớp **MainActivity** để lưu đối tượng **EditText** cho URI vị trí:

```
private EditText mLocationEditText;
```

4. Trong phương thức **onCreate()**, sử dụng **findViewById()** để lấy tham chiếu đến đối tượng **EditText** và gán nó vào biến riêng:

```
mLocationEditText = findViewById(R.id.location_edittext);
```

#### 3.2 Thêm mã vào openLocation()

1. Trong phương thức mới **openLocation()**, thêm một lệnh để lấy giá trị chuỗi từ đối tượng **mLocationEditText**:

```
String loc = mLocationEditText.getText().toString();
```

2. Phân tích chuỗi đó thành một đối tượng **Uri** với một truy vấn tìm kiếm geo:

```
Uri addressUri = Uri.parse(uriString: "geo: 0, 0?q=" + loc);
```

3. Tạo một đối tượng Intent mới với Intent.ACTION\_VIEW làm hành động và addressUri làm dữ liệu:

```
Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);
```

4. Giải quyết Intent và kiểm tra để đảm bảo rằng Intent được giải quyết thành công. Nếu có, gọi startActivity(), nếu không, ghi lại một thông báo lỗi:

```
if(intent.resolveActivity(getApplicationContext()) != null) {  
    startActivity(intent);  
} else{  
    Log.d(tag: "ImplicitIntents", msg: "Can't handle this intent!");  
}
```

Khi hoàn thành, phương thức **openLocation()** đây đủ sẽ trông như sau:

```
public void openLocation(View view) {  
    String loc = mLocationEditText.getText().toString();  
    Uri addressUri = Uri.parse(uriString: "geo: 0, 0?q=" + loc);  
    Intent intent = new Intent(Intent.ACTION_VIEW, addressUri);  
    if(intent.resolveActivity(getApplicationContext()) != null) {  
        startActivity(intent);  
    } else{  
        Log.d(tag: "ImplicitIntents", msg: "Can't handle this intent!");  
    }  
}
```

#### Nhiệm vụ 4: Triển khai nút Chia sẻ Văn bản này

Hành động chia sẻ là một cách đơn giản để người dùng chia sẻ nội dung trong ứng dụng của bạn lên mạng xã hội và các ứng dụng khác. Mặc dù bạn có thể tự xây dựng một hành động chia sẻ trong ứng dụng bằng cách sử dụng một Intent ngầm định, Android cung cấp lớp trợ giúp **ShareCompat.IntentBuilder** để việc triển khai chia sẻ trở nên dễ dàng hơn.

Bạn có thể sử dụng **ShareCompat.IntentBuilder** để tạo một **Intent** và khởi chạy trình chọn (chooser), cho phép người dùng chọn ứng dụng đích để chia sẻ.

Trong nhiệm vụ này, bạn sẽ triển khai chức năng chia sẻ một đoạn văn bản trong ô nhập văn bản (text edit), bằng cách sử dụng lớp **ShareCompat.IntentBuilder**.

#### 4.1 Định nghĩa shareText()

1. Nhấp vào "shareText" trong mã XML của **activity\_main.xml**.
2. Nhấn **Alt+Enter** (hoặc **Option+Enter** trên Mac) và chọn **Create 'shareText(View)' in MainActivity**.
  - o Android Studio sẽ tự động tạo một phương thức khung xương trong **MainActivity** cho trình xử lý **shareText()**.

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

3. Thêm một biến riêng (private variable) ở đầu lớp **MainActivity** để lưu đối tượng **EditText**:

```
private EditText mShareEditText;
```

4. Trong phương thức **onCreate()**, sử dụng **findViewById()** để lấy tham chiếu đến đối tượng **EditText** và gán nó vào biến riêng:

```
mShareEditText = findViewById(R.id.share_edittext);
```

#### 4.2 Thêm mã vào shareText()

1. Trong phương thức **shareText()**, thêm một lệnh để lấy giá trị chuỗi từ đối tượng **mShareEditText**:

```
String txt = mShareEditText.getText().toString();
```

2. Xác định loại MIME của văn bản cần chia sẻ:

```
String mimeType = "text/plain";
```

3. Gọi **ShareCompat.IntentBuilder** với các phương thức sau:

```

ShareCompat.IntentBuilder
    .from( launchingActivity: this)
    .setType(mimeType)
    .setChooserTitle("Share this text with: ")
    .setText(txt)
    .startChooser();

```

4. Trích xuất giá trị của .setChooserTitle thành một tài nguyên chuỗi (string resource).

Lời gọi đến ShareCompat.IntentBuilder sử dụng các phương thức sau:

Method	Description
from()	The Activity that launches this share Intent (this).
setType()	The MIME type of the item to be shared.
setChooserTitle()	The title that appears on the system app chooser.
setText()	The actual text to be shared
startChooser()	Show the system app chooser and send the Intent.

Định dạng này, với tất cả các phương thức setter của builder được kết nối với nhau trong một câu lệnh, là một cách viết tắt dễ dàng để tạo và khởi chạy Intent. Bạn có thể thêm bất kỳ phương thức bổ sung nào vào danh sách này.

Phương thức shareText() bây giờ nên trông như sau:

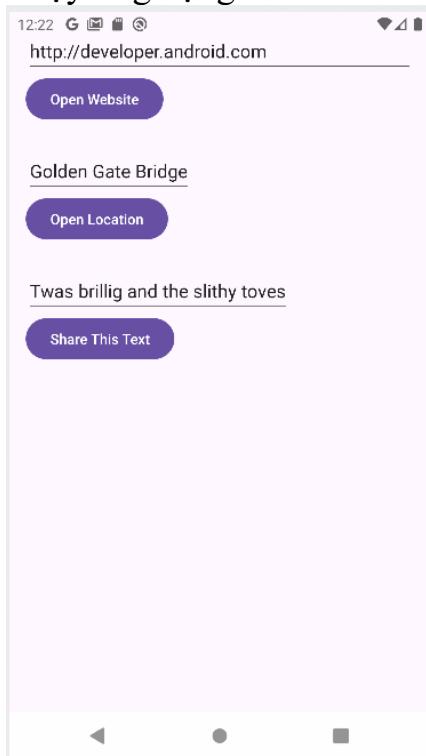
```

public void shareText(View view) {
    String txt = mShareEditText.getText().toString();
    String mimeType = "text/plain";
    ShareCompat.IntentBuilder
        .from( launchingActivity: this)
        .setType(mimeType)
        .setChooserTitle("Share this text with: ")
        .setText(txt)
        .startChooser();
}

```

## 4.2 Chạy ứng dụng

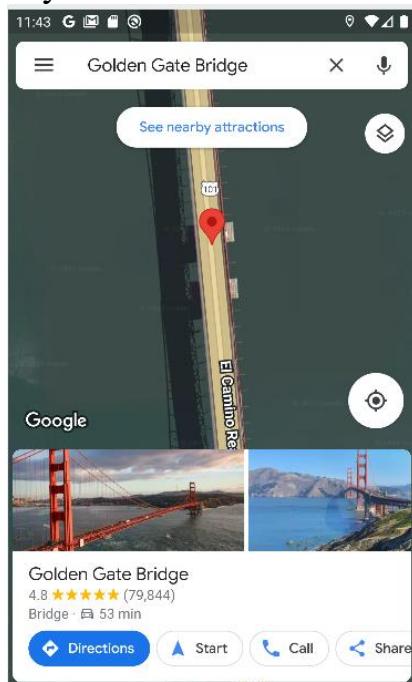
### 1. Chạy ứng dụng.



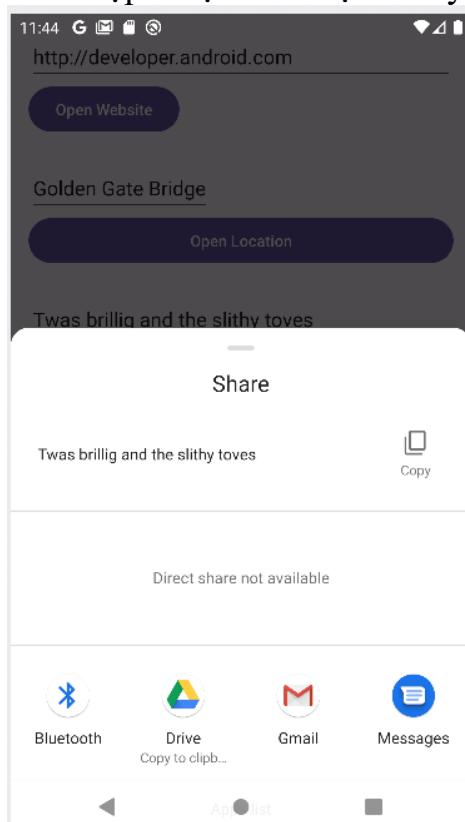
- Nhấp vào nút **Open Website** để mở trình duyệt với URL trang web được nhập trong ô **EditText** phía trên nút **Button**. Trình duyệt và trang web sẽ hiển thị như hình minh họa bên dưới.



3. Nhấn nút **Open Location** để mở bản đồ với vị trí được nhập trong **EditText** phía trên nút. Bản đồ hiển thị vị trí sẽ xuất hiện như hình dưới đây.



4. Nhấn nút **Share This Text** để mở hộp thoại với các tùy chọn chia sẻ văn bản. Hộp thoại hiển thị các tùy chọn sẽ xuất hiện như hình dưới đây.



## Nhiệm vụ 5: Nhận một Intent ngầm định

Đến nay, bạn đã tạo một ứng dụng sử dụng một **Intent ngầm định** để khởi chạy **Activity** của ứng dụng khác. Trong nhiệm vụ này, bạn sẽ xem xét vấn đề từ góc độ ngược lại: cho phép một **Activity** trong ứng dụng của bạn phản hồi một **Intent ngầm định** được gửi từ ứng dụng khác.

**Một Activity trong ứng dụng của bạn luôn có thể được kích hoạt từ bên trong hoặc bên ngoài ứng dụng bằng một Intent tường minh (explicit Intent).**

Để cho phép một Activity nhận một Intent ngầm định (implicit Intent), bạn cần định nghĩa một **Intent filter** trong tệp **AndroidManifest.xml** của ứng dụng để chỉ ra loại Intent ngầm định nào mà Activity của bạn quan tâm xử lý.

Để kết nối yêu cầu của bạn với một ứng dụng cụ thể đã cài đặt trên thiết bị, hệ thống Android sẽ khớp Intent ngầm định của bạn với một Activity có Intent filter cho biết rằng Activity đó có thể thực hiện hành động đó. Nếu có nhiều ứng dụng cài đặt phù hợp, hệ thống sẽ hiển thị một bộ chọn ứng dụng (app chooser) để người dùng chọn ứng dụng họ muốn sử dụng để xử lý Intent.

Khi một ứng dụng trên thiết bị gửi một Intent ngầm định, hệ thống Android sẽ khớp hành động (action) và dữ liệu (data) của Intent đó với bất kỳ Activity nào có Intent filter phù hợp. Khi các Intent filter của một Activity khớp với Intent:

- Nếu chỉ có một Activity phù hợp, Android sẽ để Activity đó xử lý Intent.
- Nếu có nhiều Activity phù hợp, Android sẽ hiển thị một bộ chọn ứng dụng để người dùng chọn ứng dụng họ muốn thực thi hành động đó.

Trong nhiệm vụ này, bạn sẽ tạo một ứng dụng rất đơn giản để nhận một Intent ngầm định mở URI của một trang web. Khi được kích hoạt bởi Intent ngầm định, ứng dụng đó sẽ hiển thị URI được yêu cầu dưới dạng một chuỗi trong một **TextView**.

### 5.1 Tạo dự án và bố cục

1. Tạo một dự án Android Studio mới với tên ứng dụng là **Implicit Intents Receiver** và chọn mẫu dự án **Empty Activity**.
2. Chấp nhận tên **Activity** mặc định (**MainActivity**). Nhấn **Next**.
3. Đảm bảo hộp kiểm **Generate Layout file** đã được chọn. Nhấn **Finish**.
4. Mở tệp **activity\_main.xml**.

- Trong **TextView** hiện có (hiển thị "Hello World"), xóa thuộc tính `android:text`. Mặc định **TextView** này sẽ không có văn bản, nhưng bạn sẽ thêm URI từ **Intent** trong phương thức `onCreate()`.
- Giữ nguyên các thuộc tính `layout_constraint`, nhưng thêm các thuộc tính sau:

Attribute	Value
<code>android:id</code>	"@+id/text_uri_message"
<code>android:textSize</code>	"18sp"
<code>android:textStyle</code>	"bold"

## 5.2 Sửa đổi AndroidManifest.xml để thêm một Intent filter

- Mở tệp **AndroidManifest.xml**.
- Lưu ý rằng **MainActivity** đã có Intent filter sau:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Intent filter này, là một phần của manifest mặc định trong dự án, cho biết rằng Activity này là điểm khởi đầu chính của ứng dụng (với Intent action là "android.intent.action.MAIN") và rằng Activity này sẽ xuất hiện dưới dạng một mục cấp cao nhất trong trình khởi chạy (category là "android.intent.category.LAUNCHER").

- Thêm một thẻ `<intent-filter>` thứ hai bên trong thẻ `<activity>`, và bao gồm các phần tử sau:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data
        android:host="developer.android.com"
        android:scheme="https" />
</intent-filter>
```

Các dòng này định nghĩa một Intent filter cho Activity, tức là loại Intent mà Activity có thể xử lý. Intent filter này khai báo các phần tử sau:

Loại (Type)	Giá trị (Value)	Khớp với (Matches)
action	"android.intent.action.VIEW"	Bất kỳ Intent nào có hành động là "view" (xem).
category	"android.intent.category.DEFAULT"	Bất kỳ Intent ngầm định nào. Category này phải được bao gồm để Activity của bạn nhận bất kỳ Intent ngầm định nào.
category	"android.intent.category.BROWSABLE"	Yêu cầu cho các liên kết duyệt web từ các trang web, email, hoặc nguồn khác.
data	android:scheme="http" android:host="developer.android.com"	Các URI chứa giao thức là http và tên máy chủ là developer.android.com .

Lưu ý rằng bộ lọc **data** có một giới hạn đối với cả loại liên kết mà nó sẽ chấp nhận và tên máy chủ cho các URI đó. Nếu bạn muốn bộ nhận (receiver) của mình có thể chấp nhận bất kỳ liên kết nào, bạn có thể bỏ qua phần tử <data>.

Phần **application** trong tệp **AndroidManifest.xml** bây giờ sẽ trông như sau:

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ImplicitIntentsReceiver"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
```

```

    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data
            android:scheme="https"
            android:host="developer.android.com"/>
    </intent-filter>
</activity>
</application>

</manifest>

```

### 5.3 Xử lý Intent

Trong phương thức `onCreate()` của Activity, xử lý Intent đến để lấy bất kỳ dữ liệu hoặc thông tin bổ sung (extras) nào mà nó chứa. Trong trường hợp này, Intent ngầm định đến sẽ có URI được lưu trữ trong dữ liệu của Intent.

1. Mở tệp **MainActivity**.
2. Trong phương thức `onCreate()`, lấy Intent đến được sử dụng để kích hoạt Activity:

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

3. Lấy dữ liệu từ Intent. Dữ liệu trong Intent luôn là một đối tượng URI:

```
Uri data = intent.getData();
```

4. Kiểm tra để đảm bảo biến `uri` không phải là null. Nếu kiểm tra đó thành công, tạo một chuỗi từ đối tượng URI:

```

if (data != null) {
    String uri_string = "URI: " + data.toString();
}

```

5. Trích xuất phần "URI: " vào một tài nguyên chuỗi (string resource) với tên `uri_label`.
6. Trong cùng khối if, lấy `TextView` để hiển thị thông báo:

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

7. Cũng trong khôi if, đặt nội dung văn bản cho TextView thành URI:

```
    textView.setText(uri_string);
```

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

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_main);
    Intent intent = getIntent();
    Uri data = intent.getData();
    if (data != null) {
        String uri_string = "URI: " + data.toString();
        TextView textView = findViewById(R.id.text_uri_message);
        textView.setText(uri_string);
    }
}
```

## 5.4 Chạy cả hai ứng dụng

Để hiển thị kết quả của việc nhận một Intent ngầm định, bạn sẽ chạy cả ứng dụng **Implicit Intents Receiver** và **Implicit Intents** trên trình giả lập hoặc thiết bị của mình.

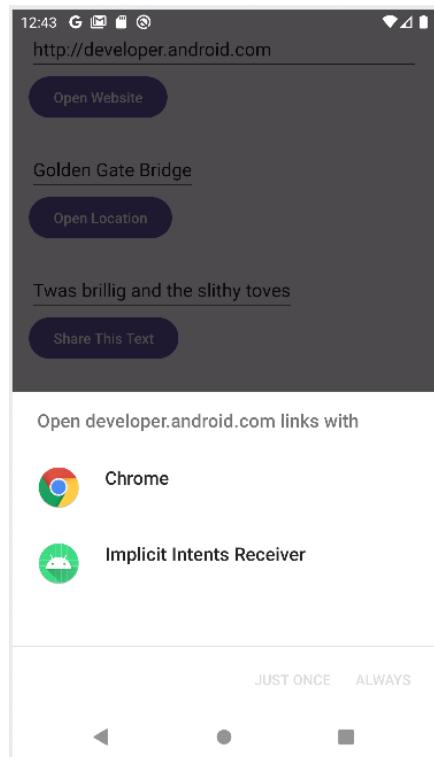
### 1. Chạy ứng dụng Implicit Intents Receiver.

Khi chạy ứng dụng này riêng lẻ, nó sẽ hiển thị một Activity trống mà không có văn bản. Điều này là do Activity được kích hoạt từ trình khởi chạy hệ thống, không phải bằng một Intent từ ứng dụng khác.

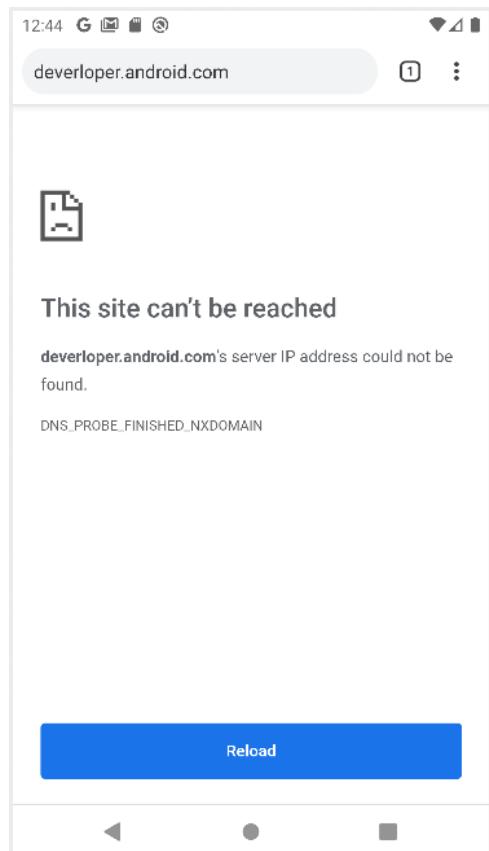
### 2. Chạy ứng dụng Implicit Intents và nhấn nút Open Website với URI mặc định.

Một hộp thoại chọn ứng dụng xuất hiện, hỏi bạn có muốn sử dụng trình duyệt mặc định (Chrome trong hình minh họa) hay ứng dụng **Implicit Intents Receiver**.

- o Chọn **Implicit Intents Receiver**, sau đó nhấn **Just Once**.
- o Ứng dụng **Implicit Intents Receiver** sẽ được khởi chạy, và thông báo sẽ hiển thị URI từ yêu cầu ban đầu



3. Nhấn nút Back và nhập một URI khác. Nhấn nút Open Website.



**Ứng dụng receiver có bộ lọc Intent rất hạn chế**, chỉ khớp với giao thức URI chính xác (`http`) và host (`developer.android.com`). Bất kỳ URI nào khác sẽ được mở bằng trình duyệt web mặc định.

## Mã giải pháp cho Task 5

Dự án Android Studio: **ImplicitIntentsReceiver**

### Thử thách lập trình

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

#### Thử thách:

Trong một thử thách thực hành trước, bạn đã tạo một ứng dụng danh sách mua sắm với một Activity để hiển thị danh sách và một Activity khác để chọn một mục.

Hãy thêm một **EditText** và một **Button** vào Activity danh sách mua sắm để định vị một cửa hàng cụ thể trên bản đồ.

### Tóm tắt

- **Intent ngầm định** cho phép bạn kích hoạt một Activity nếu bạn biết hành động nhưng không biết ứng dụng hoặc Activity cụ thể nào sẽ xử lý hành động đó.
- Một Activity có thể nhận Intent ngầm định phải định nghĩa bộ lọc Intent trong tệp **AndroidManifest.xml** để khớp với một hoặc nhiều hành động (action) và danh mục (category) của Intent.
- Hệ thống Android sẽ so khớp nội dung của Intent ngầm định và bộ lọc Intent của bất kỳ Activity có sẵn nào để xác định Activity nào sẽ được kích hoạt. Nếu có nhiều hơn một Activity có sẵn, hệ thống sẽ cung cấp một hộp chọn để người dùng chọn một.
- Lớp **ShareCompat.IntentBuilder** giúp việc tạo một Intent ngầm định để chia sẻ dữ liệu lên mạng xã hội hoặc email trở nên dễ dàng.

### Liên quan đến khái niệm

Tài liệu về khái niệm liên quan nằm trong **2.3: Implicit intents (Intent ngầm định)**.

### Tìm hiểu thêm

Tài liệu phát triển Android:

- **Application Fundamentals (Nguyên tắc cơ bản của ứng dụng)**
- **Activities (Hoạt động)**
- **Understand the Activity Lifecycle (Hiểu vòng đời của Activity)**
- **Intents and Intent Filters (Intent và Bộ lọc Intent)**
- **Allowing Other Apps to Start Your Activity (Cho phép ứng dụng khác khởi chạy Activity của bạn)**
- **Google Maps Intents for Android (Intent Google Maps cho Android)**
- **Activity**
- **Intent**
- **<intent-filter>**
- **<activity>**
- **Uri**
- **ShareCompat.IntentBuilder**

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

### 3.1) Trình gỡ lỗi

#### Giới thiệu

Trong các bài thực hành trước, bạn đã sử dụng lớp Log để in thông tin ra nhật ký hệ thống, thông tin này sẽ xuất hiện ở bảng Logcat trong Android Studio khi ứng dụng của bạn chạy. Thêm các câu lệnh ghi log vào ứng dụng là một cách để tìm lỗi và cải thiện hoạt động của ứng dụng. Một cách khác là sử dụng trình gỡ lỗi (debugger) được tích hợp trong Android Studio.

Trong bài thực hành này, bạn sẽ học cách gỡ lỗi ứng dụng trên trình giả lập hoặc thiết bị thật, đặt và xem điểm dừng (breakpoints), từng bước thực hiện mã lệnh và kiểm tra các biến.

#### Những gì bạn cần biết trước

Bạn cần có khả năng:

- Tạo một dự án Android Studio.
- Sử dụng trình chỉnh sửa giao diện (layout editor) để làm việc với các thành phần EditText và Button.

- Xây dựng và chạy ứng dụng của bạn trong Android Studio, trên trình giả lập và thiết bị thật.
- Đọc và phân tích dấu vết ngăn xếp (stack trace), bao gồm cách hiệu thứ tự **last on, first off**.
- Thêm các câu lệnh ghi log và xem nhật ký hệ thống (Logcat) trong Android Studio.

## Những gì bạn sẽ học

- Cách chạy ứng dụng ở chế độ gỡ lỗi trên trình giả lập hoặc thiết bị thật.
- Cách thực hiện từng bước mã lệnh khi ứng dụng chạy.
- Cách đặt và tổ chức các điểm dừng (breakpoints).
- Cách kiểm tra và thay đổi giá trị của các biến trong trình gỡ lỗi.

## Những gì bạn sẽ làm

- Xây dựng ứng dụng SimpleCalc.
- Đặt và xem các điểm dừng trong mã nguồn của ứng dụng SimpleCalc.
- Từng bước thực hiện mã lệnh khi ứng dụng chạy.
- Kiểm tra các biến và đánh giá các biểu thức.
- Xác định và sửa lỗi trong ứng dụng mẫu.

## Tổng quan ứng dụng

Ứng dụng SimpleCalc có hai thành phần EditText và bốn nút Button. Khi bạn nhập hai số và nhấn một nút, ứng dụng sẽ thực hiện phép tính tương ứng với nút đó và hiển thị kết quả.

### Nhiệm vụ 1: Khám phá dự án và ứng dụng SimpleCalc

Trong thực hành này, bạn sẽ không tự xây dựng ứng dụng SimpleCalc. Dự án hoàn chỉnh đã sẵn sàng tại **SimpleCalc**. Trong nhiệm vụ này, bạn sẽ mở dự án SimpleCalc trong Android Studio và khám phá một số tính năng chính của ứng dụng.

#### 1.1 Tải xuống và mở dự án SimpleCalc

1. **Tải xuống SimpleCalc** và giải nén file.
2. Mở **Android Studio** và chọn **File > Open**.
3. Điều hướng đến thư mục chứa SimpleCalc, chọn thư mục đó và nhấn **OK**.

Dự án SimpleCalc sẽ được xây dựng.

#### 4. Mở Project > Android nếu nó chưa được mở.

**Cảnh báo:** Ứng dụng này có chứa lỗi mà bạn sẽ tìm và sửa chữa. Nếu bạn chạy ứng dụng trên thiết bị hoặc trình giả lập, có thể gặp phải hành vi bất thường, bao gồm cả sự cố ứng dụng.

### 1.2 Khám phá bố cục (Layout)

1. Mở file **activity\_main.xml**.
2. Nhấp vào tab **Text** để xem mã XML.
3. Nhấp vào tab **Preview** để xem trước bố cục.

**Khám phá mã XML bố cục và thiết kế, lưu ý các điểm sau:**

- Bố cục chứa hai phần tử **EditText** để nhập liệu, bốn nút **Button** cho các phép tính và một **TextView** để hiển thị kết quả.
- Mỗi nút tính toán (**Button**) có trình xử lý sự kiện nhấp riêng thông qua thuộc tính **android:onClick** (như `onAdd`, `onSub`, v.v.).
- **TextView** dùng để hiển thị kết quả không có bất kỳ văn bản nào theo mặc định.
- Hai phần tử **EditText** có thuộc tính **android:inputType** với giá trị "`numberDecimal`". Thuộc tính này cho biết **EditText** chỉ chấp nhận đầu vào là số. Bàn phím hiển thị trên màn hình sẽ chỉ chứa các số. Bạn sẽ tìm hiểu thêm về kiểu nhập liệu (**input types**) cho phần tử **EditText** trong các thực hành sau.

### 1.3 Khám phá mã nguồn ứng dụng

1. Mở rộng thư mục **app > java** trong **Project > Android**. Ngoài lớp **MainActivity**, dự án này còn bao gồm lớp tiện ích **Calculator**.
2. Mở **Calculator** và xem xét mã nguồn. Lưu ý rằng các phép toán mà máy tính có thể thực hiện được định nghĩa bởi **enum Operator**, và tất cả các phương thức thực hiện phép toán đều có phạm vi **public**.
3. Mở **MainActivity** và xem xét mã nguồn cùng các chú thích.  
Lưu ý những điểm sau:

- Tất cả các trình xử lý sự kiện `android:onClick` được định nghĩa đều gọi phương thức riêng tư `compute()`, với tên phép toán là một trong các giá trị của `Calculator.Operator` enumeration.

- Phương thức compute() gọi phương thức riêng tư getOperand() (mà tiếp tục gọi getOperandText()) để lấy giá trị số từ các thành phần EditText.
- Phương thức compute() sau đó sử dụng câu lệnh switch trên tên phép toán để gọi phương thức thích hợp trong đối tượng Calculator (được biểu diễn bởi mCalculator).
- Các phương thức tính toán trong lớp Calculator thực hiện các phép toán số học thực tế và trả về một giá trị.
- Phần cuối cùng của phương thức compute() cập nhật TextView với kết quả của phép tính.

## 1.4 Chạy ứng dụng

1. Chạy ứng dụng và làm theo các bước sau:
  - Nhập cả giá trị số nguyên và số thực cho phép tính.
  - Nhập các giá trị số thực với các phần thập phân lớn (ví dụ: 1.6753456).
  - Thực hiện phép chia một số cho 0.
  - Để trống một hoặc cả hai thành phần EditText và thử bất kỳ phép tính nào.
2. Nhấp vào tab **Logcat** ở cuối cửa sổ Android Studio để mở khung Logcat (nếu chưa được mở).
3. Kiểm tra dấu vết ngăn xếp (stack trace) tại điểm ứng dụng báo lỗi.
  - Nếu một hoặc cả hai thành phần EditText trong SimpleCalc để trống, ứng dụng sẽ báo lỗi **exception**, như minh họa trong hình dưới đây.
  - Nhật ký hệ thống (log) sẽ hiển thị trạng thái ngăn xếp thực thi (execution stack) tại thời điểm ứng dụng tạo ra lỗi.
  - Stack trace thường cung cấp thông tin quan trọng về lý do xảy ra lỗi.

## 2.1 Bắt đầu và chạy ứng dụng trong chế độ gỡ lỗi

1. Trong Android Studio, chọn **Run > Debug app** hoặc nhấp vào biểu tượng **Debug** trên thanh công cụ.
2. Nếu ứng dụng của bạn đang chạy, hệ thống sẽ hỏi bạn có muốn khởi động lại ứng dụng trong chế độ gỡ lỗi hay không. Nhấp vào **Restart app** (Khởi động lại ứng dụng).

Android Studio sẽ biên dịch và chạy ứng dụng của bạn trên trình giả lập hoặc trên thiết bị.

- Gỡ lỗi (debugging) đều hoạt động giống nhau trên cả hai trường hợp.
  - Trong khi Android Studio đang khởi tạo debugger, bạn có thể thấy một thông báo trên thiết bị hoặc trình giả lập có nội dung: "**Waiting for debugger**" trước khi bạn có thể sử dụng ứng dụng.
3. Nhấp vào tab **Debug** ở cuối cửa sổ Android Studio để mở bảng điều khiển **Debug** (hoặc chọn **View > Tool Windows > Debug**). Tab **Debugger** sẽ tự động được chọn và hiển thị bảng điều khiển Debugger.

## 2.2 Đặt một điểm dừng (breakpoint)

Một điểm dừng (breakpoint) là một vị trí trong mã nguồn nơi bạn muốn tạm dừng việc thực thi bình thường của ứng dụng để thực hiện các hành động khác, chẳng hạn như: kiểm tra giá trị của các biến, đánh giá các biểu thức hoặc thực thi mã theo từng dòng để xác định nguyên nhân của các lỗi trong quá trình chạy. Bạn có thể đặt điểm dừng trên bất kỳ dòng mã nào có thể thực thi.

1. Mở tệp **MainActivity**, nhấp vào dòng thứ tư của phương thức `compute()` (ngay sau câu lệnh `try`).
2. Nhấp vào phần lề bên trái cạnh số dòng. Một chấm đỏ xuất hiện, biểu thị rằng một breakpoint đã được đặt. Nếu ứng dụng đang chạy ở chế độ debug, chấm đỏ sẽ có thêm dấu kiểm ().

Ngoài ra, bạn có thể chọn **Run > Toggle Line Breakpoint** hoặc nhấn **Control-F8** (hoặc **Command-F8** trên Mac) để đặt hoặc xóa breakpoint.

- Đặt nhầm có thể được sửa bằng cách nhấp lại vào chấm đỏ.
  - Nếu dòng mã không thể thực thi, chấm đỏ sẽ có thêm dấu "x" và hiển thị cảnh báo.
3. Trong ứng dụng **SimpleCalc**, nhập số vào các ô **EditText** và nhấn một nút tính toán.

Việc thực thi ứng dụng của bạn sẽ dừng lại khi đạt đến điểm dừng (breakpoint) mà bạn đã đặt, và trình gỡ lỗi (debugger) sẽ hiển thị trạng thái hiện tại của ứng dụng tại điểm dừng đó, như được minh họa trong hình dưới đây.

Hình minh họa hiển thị **ngăn Debug** với các tab **Debugger** và **Console**. Tab **Debugger** được chọn, hiển thị ngăn Debugger với các tính năng sau:

1. **Tab Frames:** Nhấp để hiển thị ngăn **Frames** với các khung ngăn xếp thực thi (execution stack frames) hiện tại cho một luồng (thread) cụ thể. Ngăn xếp thực thi hiển thị từng lớp (class) và phương thức (method) đã được gọi trong ứng dụng của bạn và trong thời gian chạy Android (Android runtime), với phương thức được gọi gần nhất ở trên cùng.
  - o Nhấp vào tab **Threads** để thay thế ngăn **Frames** bằng ngăn **Threads**. Ứng dụng của bạn hiện đang chạy trong luồng chính (main thread) và đang thực thi phương thức `compute()` trong lớp **MainActivity**.
2. **Nút Watches:** Nhấp để hiển thị ngăn **Watches** bên trong ngăn **Variables**, nơi hiển thị giá trị của bất kỳ biến nào mà bạn đã đặt để theo dõi (watches). Tính năng Watches cho phép bạn theo dõi một biến cụ thể trong chương trình và xem cách biến đó thay đổi khi chương trình chạy.
3. **Ngăn Variables:** Hiển thị các biến trong phạm vi hiện tại và giá trị của chúng. Tại giai đoạn này của việc thực thi ứng dụng, các biến có sẵn bao gồm: **this** (đại diện cho Activity), **operator** (tên toán tử từ **Calculator.Operator** mà phương thức được gọi từ đó), cũng như các biến toàn cục cho các phần tử **EditText** và **TextView**. Mỗi biến trong ngăn này có biểu tượng mở rộng để bạn có thể mở rộng danh sách các thuộc tính của đối tượng cho biến đó. Hãy thử mở rộng một biến để khám phá các thuộc tính của nó.

### 2.3 Tiếp tục thực thi ứng dụng của bạn

Tiếp tục thực thi ứng dụng của bạn bằng cách chọn **Run > Resume Program**, hoặc nhấp vào biểu tượng **Resume** ở phía bên trái của cửa sổ trình gỡ lỗi.

Ứng dụng **SimpleCalc** sẽ tiếp tục chạy, và bạn có thể tương tác với ứng dụng cho đến khi quá trình thực thi mã tiếp theo đạt đến điểm dừng (breakpoint).

### 2.4 Gỡ lỗi một ứng dụng đang chạy

Nếu ứng dụng của bạn đã chạy trên thiết bị hoặc trình giả lập, và bạn muốn gỡ lỗi ứng dụng đó, bạn có thể chuyển ứng dụng đang chạy sang chế độ gỡ lỗi.

1. Chạy ứng dụng **SimpleCalc** bình thường bằng cách nhấp vào biểu tượng **Run**.
2. Chọn **Run > Attach debugger to Android process**, hoặc nhấp vào biểu tượng **Attach** trên thanh công cụ.
3. Chọn tiến trình (process) của ứng dụng bạn từ hộp thoại xuất hiện (như hình minh họa bên dưới). Nhấp vào **OK**.

Cửa sổ Debug xuất hiện với phần Debugger được mở, và bây giờ bạn có thể gỡ lỗi ứng dụng của mình như thể bạn đã khởi chạy nó trong chế độ

## **debug.**

**Lưu ý:** Nếu cửa sổ Debug không tự động xuất hiện, nhấp vào tab **Debug** ở cuối màn hình. Nếu tab **Debugger** chưa được chọn, nhấp vào tab **Debugger** trong cửa sổ **Debug** để hiển thị phần Debugger.

### **Nhiệm vụ 3: Khám phá các tính năng của trình gỡ lỗi**

Trong nhiệm vụ này, chúng ta sẽ khám phá các tính năng khác nhau trong trình gỡ lỗi của Android Studio, bao gồm thực thi ứng dụng từng dòng, làm việc với các điểm dừng (breakpoints), và kiểm tra các biến.

#### **3.1 Thực hiện từng bước khi ứng dụng chạy**

Sau khi dừng tại điểm dừng (breakpoint), bạn có thể sử dụng trình gỡ lỗi để thực thi từng dòng mã trong ứng dụng và kiểm tra trạng thái của các biến khi ứng dụng chạy.

1. Gỡ lỗi ứng dụng trong Android Studio với điểm dừng mà bạn đã đặt ở nhiệm vụ trước.
  - 2. Trong ứng dụng, nhập số vào cả hai trường **EditText**, sau đó nhấp vào nút **Add**.
    - Việc thực thi ứng dụng sẽ dừng lại tại điểm dừng mà bạn đã đặt trước đó, và cửa sổ **Debugger** sẽ hiển thị trạng thái hiện tại của ứng dụng. Dòng mã hiện tại sẽ được đánh dấu nổi bật trong mã nguồn của bạn.
  - 3. Nhấp vào nút **Step Over** ở đầu cửa sổ trình gỡ lỗi.
    - Trình gỡ lỗi sẽ thực thi dòng mã hiện tại trong phương thức **compute()** (nơi bạn đặt điểm dừng, dòng gán giá trị cho biến **operandOne**), và điểm đánh dấu sẽ chuyển sang dòng tiếp theo trong mã nguồn (dòng gán giá trị cho biến **operandTwo**).
      - Cửa sổ **Variables** sẽ cập nhật để phản ánh trạng thái thực thi mới, và các giá trị hiện tại của biến sẽ xuất hiện dưới dạng chữ nghiêng sau mỗi dòng mã trong mã nguồn.
      - Bạn cũng có thể sử dụng **Run > Step Over** hoặc nhấn **F8** để thực hiện lệnh này.
  - 4. Tại dòng tiếp theo (dòng gán giá trị cho **operandTwo**), nhấp vào biểu tượng **Step Into**.
    - **Step Into** nhảy vào việc thực thi một lời gọi phương thức trong dòng hiện tại (so với việc chỉ thực thi phương thức đó và giữ nguyên tại dòng hiện tại). Trong trường hợp này, do dòng lệnh bao gồm lời gọi tới phương thức **getOperand()**, trình gỡ lỗi sẽ cuộn tới định nghĩa của phương thức **getOperand()** trong mã nguồn của **MainActivity**.

- Khi bạn nhảy vào một phương thức, cửa sổ **Frames** sẽ cập nhật để hiển thị khung mới trong ngăn xếp lệnh gọi (ở đây là **getOperand()**), và cửa sổ **Variables** hiển thị các biến có sẵn trong phạm vi phương thức mới. Bạn có thể nhấp vào bất kỳ dòng nào trong cửa sổ **Frames** để xem điểm trong khung lệnh gọi trước đó nơi phương thức được gọi.

Bạn cũng có thể sử dụng **Run > Step Into** hoặc phím F7 để nhảy vào một phương thức.

1. Nhấp vào **Step Over** để thực thi từng dòng trong phương thức **getOperand()**.
  - Lưu ý rằng khi phương thức hoàn thành, trình gõ lỗi sẽ đưa bạn quay lại điểm mà bạn đã nhảy vào phương thức ban đầu, và tất cả các bảng điều khiển sẽ được cập nhật để hiển thị thông tin mới.
2. Nhấp vào **Step Over** hai lần để di chuyển điểm thực thi đến dòng đầu tiên bên trong câu lệnh **case** cho **ADD**.
3. Nhấp vào **Step Into**.
  - Trình gõ lỗi thực thi phương thức phù hợp được định nghĩa trong lớp **Calculator**, mở tệp **Calculator.java**, và cuộn đến điểm thực thi trong lớp đó. Một lần nữa, các bảng điều khiển khác nhau sẽ được cập nhật để phản ánh trạng thái mới.
4. Sử dụng biểu tượng **Step Out** để thực thi phần còn lại của phương thức tính toán đó và quay trở lại phương thức **compute()** trong **MainActivity**. Bạn có thể tiếp tục gõ lỗi phương thức **compute()** từ điểm mà bạn đã dừng trước đó.
  - Bạn cũng có thể sử dụng **Run > Step Out** hoặc nhấn Shift-F8 để thoát khỏi việc thực thi một phương thức.

### 3.2 Làm việc với Breakpoint

Sử dụng breakpoint để chỉ định vị trí trong mã mà bạn muốn tạm dừng thực thi ứng dụng để gõ lỗi phần đó của ứng dụng.

1. Tìm breakpoint mà bạn đã đặt trong tác vụ trước—ở phần bắt đầu của phương thức **compute()** trong **MainActivity**.
2. Thêm một breakpoint vào dòng bắt đầu của câu lệnh **switch**.
3. Nhấp chuột phải vào breakpoint mới đó để nhập một điều kiện, như trong hình minh họa bên dưới, và nhập thử nghiệm sau vào trường **Condition**:

CopyEdit

`(operandOne == 42)|| (operandTwo == 42)`

4. Nhấn **Done** (Hoàn tất).

Breakpoint thứ hai này là một **breakpoint có điều kiện** (*conditional breakpoint*). Việc thực thi ứng dụng của bạn chỉ dừng lại tại breakpoint này nếu kiểm tra điều kiện trong biểu thức là **true**. Trong trường hợp này, biểu thức chỉ đúng nếu một trong hai toán hạng bạn nhập là **42**. Bạn có thể nhập bất kỳ biểu thức Java nào làm điều kiện miễn là nó trả về một giá trị **boolean**.

5. Chạy ứng dụng của bạn ở chế độ gỡ lỗi (**Run > Debug**) hoặc nhấn **Resume** nếu ứng dụng đã chạy. Trong ứng dụng, nhập hai số khác **42** và nhấp vào nút **Add**. Việc thực thi sẽ dừng lại tại breakpoint đầu tiên trong phương thức **compute()**.
6. Nhấn **Resume** để tiếp tục gỡ lỗi ứng dụng. Quan sát rằng việc thực thi không dừng lại ở breakpoint thứ hai của bạn vì điều kiện không được đáp ứng.
7. Trong ứng dụng, nhập **42** vào ô **EditText** đầu tiên và nhấn bất kỳ nút nào. Nhấn **Resume** để tiếp tục thực thi sau breakpoint đầu tiên. Quan sát rằng breakpoint thứ hai tại câu lệnh **switch**—breakpoint có điều kiện—dừng việc thực thi vì điều kiện đã được đáp ứng.
8. Nhấp chuột phải (hoặc **Control-click**) vào breakpoint đầu tiên trong phương thức **compute()** và bỏ chọn **Enabled**. Nhấn **Done**. Quan sát rằng biểu tượng breakpoint giờ có một chấm xanh với viền đỏ.  
Việc vô hiệu hóa một breakpoint cho phép bạn tạm thời "tắt tiếng" breakpoint đó mà không thực sự xóa nó khỏi mã của bạn. Nếu bạn xóa hoàn toàn một breakpoint, bạn cũng sẽ mất bất kỳ điều kiện nào đã tạo cho breakpoint đó, vì vậy việc vô hiệu hóa thường là lựa chọn tốt hơn.  
Bạn cũng có thể tắt tất cả các breakpoint trong ứng dụng cùng lúc bằng biểu tượng **Mute Breakpoints**.

9. Nhấp vào **View Breakpoints** ở cạnh trái của cửa sổ debugger. Cửa sổ **Breakpoints** xuất hiện.

Cửa sổ **Breakpoints** cho phép bạn xem tất cả các breakpoint trong ứng dụng, bật hoặc tắt từng breakpoint, và thêm các tính năng bổ sung cho breakpoint bao gồm các điều kiện, phụ thuộc vào các breakpoint khác và ghi nhật ký (*logging*).

Để đóng cửa sổ **Breakpoints**, nhấn **Done**.

### 3.3 Kiểm tra và thay đổi biến

Trình gỡ lỗi của Android Studio cho phép bạn kiểm tra trạng thái của các biến trong ứng dụng khi ứng dụng đang chạy.

- Chạy ứng dụng **SimpleCalc** ở chế độ debug nếu nó chưa chạy.
- Trong ứng dụng, nhập hai số, một trong số đó là **42**, sau đó nhấn nút **Add**. Breakpoint đầu tiên trong phương thức **compute()** hiện đang bị tắt. Việc thực thi dừng lại tại breakpoint thứ hai (breakpoint có điều kiện ở câu lệnh **switch**), và trình gõ lỗi sẽ xuất hiện.
- Quan sát trong bảng **Variables** rằng các biến **operandOne** và **operandTwo** đã nhận giá trị mà bạn đã nhập vào ứng dụng.
- Biến **this** là một đối tượng thuộc lớp **MainActivity**. Nhấp vào biểu tượng mở rộng để xem danh sách các biến thành viên của đối tượng đó. Sau đó, nhấp lại vào biểu tượng mở rộng để đóng danh sách.
- Nhấp chuột phải (hoặc **Control-click**) vào biến **operandOne** trong bảng **Variables**, và chọn **Set Value**.
- Thay đổi giá trị của biến **operandOne** thành **10** và nhấn **Return**.
- Thay đổi giá trị của biến **operandTwo** thành **10** theo cách tương tự và nhấn **Return**.
- Quan sát rằng kết quả trong ứng dụng bây giờ dựa trên các giá trị biến mà bạn đã thay đổi trong trình gõ lỗi; ví dụ, vì bạn đã nhấn nút **Add** ở Bước 2, kết quả trong ứng dụng giờ là **20**.
- Nhấp vào biểu tượng **Resume** để tiếp tục chạy ứng dụng của bạn.
- Trong ứng dụng, các mục nhập ban đầu (bao gồm **42**) vẫn được giữ nguyên trong các trường **EditText**. (Các giá trị này chỉ được thay đổi trong trình gõ lỗi.) Nhấn lại nút **Add**. Quá trình thực thi sẽ dừng lại tại điểm breakpoint một lần nữa.
- Nhấn vào biểu tượng **Evaluate Expression**, hoặc chọn **Run > Evaluate Expression**. Bạn cũng có thể nhấp chuột phải (hoặc **Control-click**) vào bất kỳ biến nào và chọn **Evaluate Expression**.

Cửa sổ **Evaluate Code Fragment** xuất hiện. Sử dụng nó để khám phá trạng thái của các biến và đối tượng trong ứng dụng của bạn, bao gồm cả việc gọi các phương thức trên các đối tượng đó. Bạn có thể nhập bất kỳ đoạn mã nào vào cửa sổ này.

- Nhập câu lệnh **mOperandOneEditText.getHint()**; vào trường phía trên của cửa sổ **Evaluate Code Fragment** (như minh họa trong hình trên) và nhấn **Evaluate**.

13. Trường **Result** hiển thị kết quả của biểu thức đó. Gợi ý (hint) cho trường **EditText** này là chuỗi "**Type Operand 1**", như đã được định nghĩa ban đầu trong tệp XML cho trường **EditText** này.

Kết quả bạn nhận được khi đánh giá một biểu thức dựa trên trạng thái hiện tại của ứng dụng. Tùy thuộc vào giá trị của các biến trong ứng dụng tại thời điểm bạn đánh giá biểu thức, bạn có thể nhận được kết quả khác nhau.

Cũng cần lưu ý rằng nếu bạn sử dụng **Evaluate Expression** để thay đổi giá trị của biến hoặc thuộc tính của đối tượng, bạn sẽ thay đổi trạng thái đang chạy của ứng dụng.

14. Nhấn **Close** để đóng cửa sổ **Evaluate Code Fragment**.

**Thách thức lập trình**

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

**Thách thức:** Ở cuối Bài 1, bạn đã thử chạy ứng dụng **SimpleCalc** mà không nhập giá trị vào một trong các trường **EditText**, dẫn đến lỗi. Sử dụng trình gỡ lỗi để từng bước thực thi mã và xác định chính xác lý do gây ra lỗi này. Khắc phục lỗi gây ra sự cố này.

**Tóm tắt:**

- Xem thông tin ghi log trong Android Studio bằng cách nhấp vào tab **Logcat**.
- Chạy ứng dụng của bạn trong chế độ gỡ lỗi bằng cách nhấp vào biểu tượng **Debug** hoặc chọn **Run > Debug app**.
- Nhấp vào tab **Debug** để hiển thị khung **Debug**. Nhấp vào tab **Debugger** trong khung **Debug** để hiển thị khung **Debugger** (nếu nó chưa được chọn).
- Khung **Debugger** hiển thị các **Frames** (ngăn xếp), **Variables** trong một frame cụ thể và **Watches** (theo dõi hoạt động của một biến khi chương trình chạy).
- **Breakpoint** là một điểm trong mã của bạn nơi bạn muốn tạm dừng quá trình thực thi bình thường của ứng dụng để thực hiện các hành động khác. Đặt hoặc xóa một điểm dừng gỡ lỗi bằng cách nhấp vào lề bên trái của cửa sổ trình soạn thảo ngay cạnh dòng mục tiêu.

**Khái niệm liên quan:**

Tài liệu khái niệm liên quan nằm trong mục **3.1: Trình gỡ lỗi của Android Studio**.

## Tìm hiểu thêm

### Tài liệu Android Studio:

- Hướng dẫn sử dụng Android Studio
- Gỡ lỗi ứng dụng của bạn
- Ghi và xem log
- Phân tích Stack Trace
- Cầu nối gỡ lỗi Android (ADB)
- Công cụ phân tích Android Profiler
- Trình phân tích mạng (Network Profiler)
- Trình phân tích CPU (CPU Profiler)
- Traceview

### Khác:

- Video: Gỡ lỗi và kiểm thử trong Android Studio

## 3.2) Kiểm thử đơn vị

### Giới thiệu

Kiểm thử mã của bạn có thể giúp bạn phát hiện lỗi sớm trong quá trình phát triển, khi chi phí xử lý lỗi là thấp nhất. Khi ứng dụng của bạn trở nên lớn hơn và phức tạp hơn, việc kiểm thử cải thiện độ tin cậy của mã.

Với các bài kiểm thử trong mã, bạn có thể kiểm tra từng phần nhỏ trong ứng dụng một cách độc lập, và bạn có thể kiểm thử theo cách tự động hóa và lặp lại được.

Android Studio và **Android Testing Support Library** hỗ trợ nhiều loại kiểm thử và khung kiểm thử khác nhau. Trong bài thực hành này, bạn sẽ khám phá chức năng kiểm thử tích hợp sẵn của Android Studio và học cách viết và chạy các bài kiểm thử đơn vị cục bộ.

**Kiểm thử đơn vị cục bộ** là các bài kiểm thử được biên dịch và chạy hoàn toàn trên máy cục bộ của bạn với **Java Virtual Machine (JVM)**. Bạn sử dụng kiểm thử đơn vị cục bộ để kiểm tra các phần của ứng dụng không cần truy cập vào **Android framework** hoặc thiết bị/emulator Android, chẳng hạn như logic nội bộ.

Bạn cũng có thể sử dụng kiểm thử đơn vị cục bộ để kiểm tra các phần của ứng dụng mà bạn có thể tạo ra các đối tượng giả ("mock" hoặc "stub") để bắt chước hành vi của các thành phần tương đương trong framework.

Kiểm thử đơn vị được viết bằng **JUnit**, một khung kiểm thử đơn vị phổ biến dành cho Java.

## Những gì bạn cần biết trước

Bạn cần phải:

- Tạo một dự án Android Studio.
- Xây dựng và chạy ứng dụng của bạn trong Android Studio, trên cả trình giả lập và thiết bị thực.
- Điều hướng trong **Project > Android** pane của Android Studio.

### • Tìm các thành phần chính của một dự án Android Studio, bao gồm:

- **AndroidManifest.xml**,
- Các tài nguyên (resources),
- Các tệp Java,
- Các tệp Gradle.

## Bạn sẽ học được gì?

- Cách tổ chức và chạy các bài kiểm thử trong Android Studio.
- Hiểu kiểm thử đơn vị là gì.
- Viết các bài kiểm thử đơn vị cho mã của bạn.

## Bạn sẽ làm gì?

- Chạy các bài kiểm thử ban đầu trong ứng dụng **SimpleCalc**.
- Thêm các bài kiểm thử mới vào ứng dụng **SimpleCalc**.
- Chạy các bài kiểm thử đơn vị để xem kết quả.

## Tổng quan về ứng dụng

Bài thực hành này sử dụng ứng dụng **SimpleCalc** từ bài codelab thực hành trước (**Android fundamentals 3.1: The debugger**). Bạn có thể chỉnh sửa ứng dụng này tại chỗ hoặc tạo một bản sao thư mục dự án trước khi tiếp tục.

## Nhiệm vụ 1: Khám phá và chạy CalculatorTest

Bạn sẽ viết và chạy các bài kiểm thử (bao gồm cả kiểm thử đơn vị và kiểm thử có công cụ) trong **Android Studio**, cùng với mã của ứng dụng.

Mỗi dự án Android mới đều bao gồm các lớp kiểm thử mẫu cơ bản mà bạn có thể mở rộng hoặc thay thế để sử dụng cho mục đích riêng của mình.

Trong nhiệm vụ này, bạn sẽ quay lại ứng dụng **SimpleCalc**, ứng dụng này đã bao gồm một lớp kiểm thử đơn vị cơ bản.

### 1.1 Khám phá bộ mã nguồn (source sets) và CalculatorTest

**Source sets** là các tập hợp mã trong dự án của bạn dùng cho các mục tiêu build khác nhau hoặc các "phiên bản" khác nhau của ứng dụng. Khi Android Studio tạo dự án của bạn, nó sẽ tạo ra ba bộ mã nguồn:

- **Bộ mã nguồn chính (main source set)**: Chứa mã và tài nguyên chính của ứng dụng.
- **Bộ mã nguồn (test)**: Chứa các bài kiểm thử đơn vị cục bộ của ứng dụng. Bộ mã nguồn này hiển thị (**test**) sau tên gói.
- **Bộ mã nguồn (androidTest)**: Chứa các bài kiểm thử có công cụ (instrumented tests) dành cho Android. Bộ mã nguồn này hiển thị (**androidTest**) sau tên gói.

Trong nhiệm vụ này, bạn sẽ khám phá cách các bộ mã nguồn được hiển thị trong **Android Studio**, kiểm tra cấu hình Gradle dành cho việc kiểm thử, và chạy các bài kiểm thử đơn vị cho ứng dụng **SimpleCalc**.

**Lưu ý:** Bộ mã nguồn (**androidTest**) đã được loại bỏ khỏi ví dụ này để đơn giản hóa. Nó sẽ được giải thích chi tiết hơn trong một bài học khác.

### Các bước thực hiện

1. Mở dự án **SimpleCalc** trong **Android Studio** nếu bạn chưa làm điều này.
2. Mở bảng điều hướng **Project > Android**, sau đó mở rộng các thư mục **app** và **java**.
  - Thư mục **java** trong chế độ xem **Android** hiển thị tất cả các bộ mã nguồn trong ứng dụng theo tên gói.
  - Trong trường hợp này (như minh họa bên dưới), mã của ứng dụng nằm trong bộ mã nguồn **com.android.example.SimpleCalc**.

- Mã kiểm thử nằm trong bộ mã nguồn có chữ **test** xuất hiện trong ngoặc đơn sau tên gói: **com.android.example.SimpleCalc (test)**.

## 2. Mở rộng thư mục com.android.example.SimpleCalc (test)

Thư mục này là nơi bạn đặt các bài kiểm thử đơn vị cục bộ của ứng dụng. Android Studio tạo sẵn một lớp kiểm thử mẫu trong thư mục này cho các dự án mới, nhưng với ứng dụng **SimpleCalc**, lớp kiểm thử được gọi là **CalculatorTest**.

### 1. Mở tệp CalculatorTest

Hãy kiểm tra mã nguồn và lưu ý những điểm sau:

- **Các thư viện được nhập:**  
Chỉ có các thư viện từ các gói **org.junit**, **org.hamcrest**, và **android.test** được nhập. Không có sự phụ thuộc nào vào các lớp của framework Android.
- **Chú thích @RunWith(Unit4.class):**  
Chú thích này chỉ định **runner** sẽ được sử dụng để chạy các bài kiểm thử trong lớp này. **Test runner** là một thư viện hoặc bộ công cụ cho phép thực hiện kiểm thử và in kết quả ra log. Với các bài kiểm thử cần thiết lập hoặc hạch toán phức tạp hơn (như Espresso), bạn sẽ sử dụng các test runner khác.  
Trong ví dụ này, chúng ta sử dụng **JUnit4 test runner** cơ bản.
- **Chú thích @SmallTest:**  
Chú thích này cho biết tất cả các bài kiểm thử trong lớp là kiểm thử đơn vị, không có phụ thuộc nào và chạy trong thời gian rất ngắn (tính bằng mili giây). Các chú thích **@SmallTest**, **@MediumTest**, và **@LargeTest** là các quy ước giúp dễ dàng nhóm các bài kiểm thử thành các bộ với chức năng tương tự nhau.
- **Phương thức setUp():**  
Phương thức này được dùng để thiết lập môi trường trước khi kiểm thử và có chú thích **@Before**. Trong trường hợp này, phương thức **setUp()** tạo một instance mới của lớp **Calculator** và gán nó vào biến thành viên **mCalculator**.
- **Phương thức addTwoNumbers():**  
Đây là một bài kiểm thử thực tế và được chú thích bằng **@Test**. Chỉ những phương thức trong lớp kiểm thử có chú thích **@Test** mới được trình kiểm thử (test runner) xem là bài kiểm thử. Theo quy ước, tên của phương thức kiểm thử không bao gồm từ "test."
- **Dòng lệnh đầu tiên của addTwoNumbers():**  
Dòng lệnh này gọi phương thức **add()** từ lớp **Calculator**. Bạn chỉ có thể

kiểm thử các phương thức có phạm vi truy cập là **public** hoặc **package-protected**. Trong trường hợp này, lớp **Calculator** là lớp **public** với các phương thức **public**, nên không có vấn đề gì.

- **Dòng lệnh thứ hai:**

Đây là một biểu thức khẳng định (assertion) cho bài kiểm thử. **Assertion** là các biểu thức phải đánh giá và trả về **true** để bài kiểm thử được thông qua. Trong trường hợp này, **assertion** kiểm tra xem kết quả nhận được từ phương thức **add()** ( $1 + 1$ ) có khớp với số được chỉ định là 2 hay không. Bạn sẽ tìm hiểu thêm về cách tạo các **assertion** sau trong bài thực hành này.

## 1.2 Chạy kiểm thử trong Android Studio

Trong phần này, bạn sẽ chạy các bài kiểm thử đơn vị trong thư mục kiểm thử và xem kết quả cho cả kiểm thử thành công và thất bại.

### Các bước thực hiện:

1. Trong ngăn **Project > Android**, nhấp chuột phải (hoặc Control-click) vào **CalculatorTest** và chọn **Run 'CalculatorTest'**.
  - o Dự án sẽ được xây dựng (nếu cần thiết), và ngăn **CalculatorTest** sẽ xuất hiện ở dưới cùng của màn hình.
  - o Ở đầu ngăn này, danh sách thả xuống cho các cấu hình thực thi khả dụng cũng thay đổi thành **CalculatorTest**.
2. Tất cả các bài kiểm thử trong lớp **CalculatorTest** sẽ được chạy.
  - o Nếu các bài kiểm thử thành công, thanh tiến trình ở đầu khung nhìn sẽ chuyển sang màu **xanh lá cây**.
  - o Một thông báo trạng thái ở cuối màn hình sẽ báo "**Tests Passed**".
3. Mở **CalculatorTest** nếu nó chưa được mở, và thay đổi biểu thức khẳng định trong phương thức **addTwoNumbers()** thành:

java

CopyEdit

```
assertThat(resultAdd, is(equalTo(3d)));
```

## 2. Chạy lại bài kiểm thử với cấu hình CalculatorTest

1. Trong danh sách thả xuống cấu hình chạy (run configurations) ở phía trên màn hình, chọn **CalculatorTest** (nếu nó chưa được chọn) và nhấn **Run**.

- Bài kiểm thử sẽ chạy lại như trước, nhưng lần này biểu thức khẳng định thất bại (**3 không bằng 1 + 1**).
  - Thanh tiến trình trong khung chạy kiểm thử chuyển sang màu **đỏ**, và nhật ký kiểm thử chỉ ra vị trí bài kiểm thử (biểu thức khẳng định) thất bại cũng như lý do tại sao.
2. Thay đổi biểu thức khẳng định trong phương thức **addTwoNumbers()** quay lại bài kiểm thử đúng, và chạy lại kiểm thử để đảm bảo tất cả bài kiểm thử đều thành công.
  3. Trong danh sách thả xuống cấu hình chạy, chọn **app** để chạy ứng dụng bình thường.

## Nhiệm vụ 2: Thêm các kiểm thử đơn vị khác vào CalculatorTest

Kiểm thử đơn vị là kiểm tra một phần nhỏ trong mã nguồn của bạn (chẳng hạn một phương thức hoặc một lớp), và tách phần đó khỏi phần còn lại của ứng dụng. Điều này giúp đảm bảo rằng phần mã nhỏ đó hoạt động đúng như mong đợi.

Thông thường, một kiểm thử đơn vị gọi một phương thức với nhiều đầu vào khác nhau, và kiểm tra rằng phương thức hoạt động như mong đợi và trả về kết quả đúng.

Trong nhiệm vụ này, bạn sẽ viết thêm các kiểm thử đơn vị cho các phương thức tiện ích của lớp **Calculator** trong ứng dụng **SimpleCalc**, và chạy các bài kiểm thử để đảm bảo rằng chúng tạo ra đầu ra như bạn mong đợi.

### 2.1 Thêm các bài kiểm thử khác cho phương thức add()

Mặc dù không thể kiểm thử mọi giá trị đầu vào mà phương thức **add()** có thể nhận, nhưng việc kiểm tra các trường hợp đầu vào đặc biệt là rất quan trọng. Ví dụ:

- Đầu vào có chứa **toán hạng âm**.
- Đầu vào có chứa **số thực (floating-point numbers)**.
- **Số rất lớn (exceptionally large numbers)**
- **Các loại toán hạng khác nhau (float và double)**
- **Toán hạng bằng 0 (zero)**
- **Toán hạng là vô cực (infinity)**

Trong nhiệm vụ này, chúng ta sẽ thêm các bài kiểm thử đơn vị (unit tests) mới cho phương thức add() để kiểm tra các loại đầu vào khác nhau.

1. **Thêm phương thức mới vào CalculatorTest có tên là addTwoNumbersNegative().**

Sử dụng khung mã như sau:

```
java
```

```
CopyEdit
```

```
@Test
```

```
public void addTwoNumbersNegative() {
```

```
}
```

Phương thức kiểm thử này có cấu trúc tương tự như addTwoNumbers():

- o Là phương thức **public**, không có tham số, và trả về kiểu **void**.
- o Được chú thích bằng **@Test**, cho biết đây là một bài kiểm thử đơn lẻ.

2. **Tại sao không thêm nhiều lệnh kiểm tra (assertions) vào addTwoNumbers()?**

- o Gom nhiều lệnh kiểm tra vào một phương thức có thể làm cho bài kiểm thử khó gỡ lỗi nếu chỉ một lệnh kiểm tra thất bại.
- o Ngoài ra, việc này cũng làm mờ các bài kiểm thử thành công khác.
- o Quy tắc chung cho các bài kiểm thử đơn vị là **tạo một phương thức kiểm thử cho từng lệnh kiểm tra riêng lẻ**.

3. **Chạy tất cả các bài kiểm thử trong CalculatorTest như trước.**

- o Trong cửa sổ kiểm thử, cả addTwoNumbers và addTwoNumbersNegative đều được liệt kê dưới dạng các bài kiểm thử khả dụng (và đang thành công) trong bảng điều khiển bên trái.
- o Bài kiểm thử addTwoNumbersNegative vẫn thành công ngay cả khi không chứa bất kỳ đoạn mã nào—một bài kiểm thử không làm gì vẫn được xem là thành công.

4. **Thêm một dòng mã vào addTwoNumbersNegative() để gọi phương thức add() trong lớp Calculator với một toán hạng âm.**

java

CopyEdit

```
double resultAdd = mCalculator.add(1d, 2d);
```

### **Ký hiệu d sau mỗi toán hạng cho biết đây là các số kiểu double.**

Do phương thức add() được định nghĩa với các tham số kiểu double, các kiểu dữ liệu khác như float hoặc int cũng có thể hoạt động. Việc chỉ rõ kiểu giúp bạn kiểm tra riêng biệt các kiểu khác nếu cần thiết.

### **4. Thêm một lệnh kiểm tra (assertion) với assertThat().**

java

CopyEdit

```
assertThat(resultAdd, is(equalTo(1d)));
```

- Phương thức assertThat() là một lệnh kiểm tra trong JUnit4, khẳng định biểu thức trong đối số đầu tiên bằng với biểu thức trong đối số thứ hai.
- Các phiên bản cũ của JUnit sử dụng các phương thức kiểm tra cụ thể hơn như assertEquals(), assertNull(), hoặc assertTrue(), nhưng assertThat() linh hoạt hơn, dễ đọc và dễ gỡ lỗi hơn.

### **assertThat() sử dụng matchers.**

- Matchers là các phương thức liên kết được gọi trong đối số thứ hai của lệnh kiểm tra này, chẳng hạn như is(equalTo()).
- Framework Hamcrest định nghĩa các matchers sẵn có để xây dựng các lệnh kiểm tra. (Tên "Hamcrest" là một phép đảo chữ từ "matchers.")
- Hamcrest cung cấp nhiều matchers cơ bản cho hầu hết các lệnh kiểm tra. Bạn cũng có thể tự định nghĩa matchers của riêng mình cho các lệnh kiểm tra phức tạp hơn.

### **Ví dụ:**

Trong trường hợp này, lệnh kiểm tra xác nhận rằng kết quả của phép cộng add() (-1 + 2) bằng với 1.

### **5. Thêm bài kiểm tra đơn vị mới vào CalculatorTest cho số dấu phẩy động:**

java

CopyEdit

@Test

```
public void addTwoNumbersFloats() {  
    double resultAdd = mCalculator.add(1.111f, 1.111d);  
    assertThat(resultAdd, is(equalTo(2.222d)));  
}
```

- Đây là một bài kiểm tra rất giống với phương thức kiểm tra trước, nhưng một đối số của phương thức add() được chỉ rõ là kiểu float thay vì double.
- Phương thức add() được định nghĩa với các tham số kiểu double, vì vậy bạn có thể gọi nó với kiểu float. Khi đó, số kiểu float sẽ được chuyển đổi thành kiểu double.

## 6. Nhấn vào Run để chạy lại tất cả các bài kiểm tra.

**Lần này bài kiểm tra thất bại và thanh tiến trình chuyển sang màu đỏ. Đây là phần quan trọng trong thông báo lỗi:**

makefile

CopyEdit

java.lang.AssertionError:

Expected: is <2.222>

but: was <2.2219999418258665>

**Tính toán với các số dấu phẩy động không chính xác, và việc chuyển đổi đã gây ra hiệu ứng phụ về độ chính xác.**

Lệnh kiểm tra trong bài kiểm tra này về mặt kỹ thuật là sai: giá trị mong đợi không bằng giá trị thực tế.

### Câu hỏi đặt ra là:

Khi gặp vấn đề về độ chính xác do việc chuyển đổi tham số kiểu float, đó có phải là vấn đề của mã nguồn hay bài kiểm tra?

- Trong trường hợp này, cả hai đối số đầu vào của phương thức add() từ ứng dụng SimpleCalc đều luôn thuộc kiểu double, do đó đây là một bài kiểm tra tùy ý và không thực tế.
- Tuy nhiên, nếu ứng dụng của bạn được viết sao cho đầu vào của phương thức add() có thể là kiểu double hoặc float, và bạn chỉ quan tâm đến một mức độ chính xác nhất định, thì bạn cần thêm một biên độ "gần đúng" để bài kiểm tra thành công.

### 7. Thay đổi phương thức assertThat() để sử dụng matcher closeTo():

java

CopyEdit

```
assertThat(resultAdd, is(closeTo(2.222, 0.01)));
```

### Hướng dẫn:

- Bạn cần chọn matcher thích hợp. Nhập vào closeTo hai lần (cho đến khi toàn bộ biểu thức được gạch chân) và nhấn **Alt+Enter (Option+Return trên Mac)**.
- Chọn **isCloseTo.closeTo (org.hamcrest.number)**.

### 8. Nhấn vào Run để chạy lại tất cả các bài kiểm tra.

Lần này bài kiểm tra đã thành công.

**Với matcher closeTo(), thay vì kiểm tra sự bằng nhau tuyệt đối, bạn có thể kiểm tra sự bằng nhau trong một phạm vi delta nhất định.**

- Trong trường hợp này, phương thức matcher closeTo() nhận hai tham số: giá trị mong đợi và giá trị delta.
- Trong ví dụ trên, delta là phạm vi gần đúng với độ chính xác hai chữ số thập phân.

## 2.2 Thêm các bài kiểm tra đơn vị cho các phương thức tính toán khác

Sử dụng những gì bạn đã học trong nhiệm vụ trước để hoàn thiện các bài kiểm tra đơn vị cho lớp Calculator.

1. Thêm một bài kiểm tra đơn vị có tên subTwoNumbers() để kiểm tra phương thức sub().
2. Thêm một bài kiểm tra đơn vị có tên subWorksWithNegativeResults() để kiểm tra phương thức sub() khi phép tính cho ra kết quả âm.
3. Thêm một bài kiểm tra đơn vị có tên mulTwoNumbers() để kiểm tra phương thức mul().
4. Thêm một bài kiểm tra đơn vị có tên mulTwoNumbersZero() để kiểm tra phương thức mul() khi ít nhất một tham số là số 0.
5. Thêm một bài kiểm tra đơn vị có tên divTwoNumbers() để kiểm tra phương thức div() với hai tham số khác 0.
6. Thêm một bài kiểm tra đơn vị có tên divTwoNumbersZero() để kiểm tra phương thức div() với một số chia kiểu double và số chia là 0.

Tất cả các bài kiểm tra trên đều phải thành công, ngoại trừ divTwoNumbersZero(), bài này sẽ gây ra ngoại lệ tham số không hợp lệ khi chia cho 0.

- Nếu bạn chạy ứng dụng, nhập 0 làm Số hạng 2 và nhấn Div để chia, kết quả sẽ là lỗi.

### 3.3) Giải pháp mã cho nhiệm vụ 2:

**Dự án Android Studio:** SimpleCalcTest

Đoạn mã sau đây minh họa các bài kiểm tra cho nhiệm vụ này:

```
public void addTwoNumbers() {
    double resultAdd = mCalculator.add(1d, 1d);
    assertThat(resultAdd, is(equalTo(2d)));
}

@Test
public void addTwoNumbersNegative() {
    double resultAdd = mCalculator.add(-1d, 2d);
    assertThat(resultAdd, is(equalTo(1d)));
}

@Test
public void addTwoNumbersFloats() {
    double resultAdd = mCalculator.add(1.111f, 1.111d);
    assertThat(resultAdd, is(closeTo(2.222, 0.01)));
}

@Test
public void subTwoNumbers() {
    double resultSub = mCalculator.sub(1d, 1d);
    assertThat(resultSub, is(equalTo(0d)));
}

@Test
public void subWorksWithNegativeResult() {
    double resultSub = mCalculator.sub(1d, 17d);
    assertThat(resultSub, is(equalTo(-16d)));
}

@Test
public void multTwoNumbers() {
    double resultMul = mCalculator.mul(32d, 2d);
    assertThat(resultMul, is(equalTo(64d)));
}

@Test
public void divTwoNumbers() {
    double resultDiv = mCalculator.div(32d, 2d);
    assertThat(resultDiv, is(equalTo(16d)));
}

@Test
public void divTwoNumbersZero() {
    double resultDiv = mCalculator.div(32d, 0);
    assertThat(resultDiv, is(equalTo(Double.POSITIVE_INFINITY)));
}
```

```

@Test
public void subWorksWithNegativeResult() {
    double resultSub = mCalculator.sub(1d, 17d);
    assertThat(resultSub, is(equalTo(-16d)));
}

@Test
public void multTwoNumbers() {
    double resultMul = mCalculator.mul(32d, 2d);
    assertThat(resultMul, is(equalTo(64d)));
}

@Test
public void divTwoNumbers() {
    double resultDiv = mCalculator.div(32d, 2d);
    assertThat(resultDiv, is(equalTo(16d)));
}

@Test
public void divTwoNumbersZero() {
    double resultDiv = mCalculator.div(32d, 0);
    assertThat(resultDiv, is(equalTo(Double.POSITIVE_INFINITY)));
}

```

---

## Thách thức lập trình

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

### Thách thức 1:

Chia cho 0 luôn là một trường hợp đặc biệt trong toán học và đáng để kiểm tra. Làm thế nào bạn có thể thay đổi ứng dụng để xử lý chia cho 0 một cách dễ chịu hơn?

Để thực hiện thách thức này, hãy bắt đầu bằng một bài kiểm tra cho thấy hành vi đúng là gì.

- Xóa phương thức divTwoNumbersZero() khỏi lớp CalculatorTest và thêm một bài kiểm tra đơn vị mới có tên là divByZeroThrows() để kiểm tra phương thức div() với đối số thứ hai bằng 0, với kết quả mong đợi là IllegalArgumentException.class.
- Bài kiểm tra này sẽ thành công và chứng minh rằng bất kỳ phép chia nào với số 0 sẽ dẫn đến ngoại lệ này.

Sau khi bạn học cách viết mã cho một trình xử lý Exception, ứng dụng của bạn có thể xử lý ngoại lệ này một cách dễ chịu, ví dụ: hiển thị một thông báo Toast yêu cầu người dùng thay đổi Operand 2 từ 0 thành một số khác.

---

### Thách thức 2:

Đôi khi rất khó để cài đặt một đơn vị mã khỏi tất cả các phụ thuộc bên ngoài của nó. Thay vì tổ chức mã của bạn theo những cách phức tạp chỉ để dễ dàng kiểm tra hơn, bạn có thể sử dụng một khung giả lập để tạo ra các đối tượng giả ("mock") giả vờ là các phụ thuộc.

- Nghiên cứu khung Mockito và tìm hiểu cách thiết lập nó trong Android Studio.
- Viết một lớp kiểm tra cho phương thức calcButton() trong ứng dụng SimpleCalc và sử dụng Mockito để mô phỏng ngữ cảnh Android mà các bài kiểm tra của bạn sẽ chạy.

### Tóm tắt

Android Studio có các tính năng tích hợp để chạy các bài kiểm tra đơn vị cục bộ:

- **Bài kiểm tra đơn vị cục bộ** sử dụng JVM trên máy cục bộ của bạn. Chúng không sử dụng khung Android.
- Các bài kiểm tra đơn vị được viết bằng **JUnit**, một khung kiểm tra đơn vị phổ biến cho Java.
- Các bài kiểm tra JUnit được đặt trong thư mục **(test)** trong mục **Project > Android** của Android Studio.
- Các bài kiểm tra đơn vị cục bộ chỉ cần các gói sau: **org.junit**, **org.hamcrest**, và **android.test**.
- Chú thích **@RunWith(JUnit4.class)** báo cho trình chạy kiểm tra thực thi các bài kiểm tra trong lớp này.
- Các chú thích **@SmallTest**, **@MediumTest**, và **@LargeTest** là các quy ước giúp dễ dàng nhóm các bài kiểm tra tương tự lại với nhau.
- Chú thích **@SmallTest** cho biết tất cả các bài kiểm tra trong một lớp là các bài kiểm tra đơn vị không có phụ thuộc và chạy trong vài mili-giây.
- **Bài kiểm tra có công cụ hỗ trợ (Instrumented tests)** là các bài kiểm tra chạy trên một thiết bị hoặc trình giả lập chạy Android. Các bài kiểm tra này có quyền truy cập vào khung Android.
- **Trình chạy kiểm tra (Test runner)** là một thư viện hoặc tập hợp công cụ cho phép thực hiện các bài kiểm tra và in kết quả vào nhật ký (log).

## **Khái niệm liên quan**

Tài liệu khái niệm liên quan nằm trong **3.2: App testing.**

### **Tìm hiểu thêm**

Tài liệu Android Studio:

- [Hướng dẫn sử dụng Android Studio \(Android Studio User Guide\)](#)
- [Ghi và xem nhật ký \(Write and View Logs\)](#)

Tài liệu nhà phát triển Android:

- [Thực hành tốt nhất cho kiểm tra \(Best Practices for Testing\)](#)
- [Bắt đầu với kiểm tra \(Getting Started with Testing\)](#)
- [Xây dựng kiểm tra đơn vị cục bộ \(Building Local Unit Tests\)](#)

Khác:

- [\*\*Trang chủ JUnit 4 \(JUnit 4 Home Page\)\*\*](#)
- [\*\*Tài liệu API JUnit 4 \(JUnit 4 API Reference\)\*\*](#)
- [java.lang.Math](#)
- [\*\*Java Hamcrest\*\*](#)
- [\*\*Trang chủ Mockito \(Mockito Home Page\)\*\*](#)
- [Video: Android Testing Support - Testing Patterns](#)
- [Hướng dẫn lập trình kiểm tra Android \(Android Testing Codelab\)](#)
- [Protip về chú thích kích thước kiểm tra Android \(Android Tools Protip: Test Size Annotations\)](#)
- [\*\*Lợi ích của việc sử dụng assertThat thay vì các phương thức Assert khác trong kiểm tra đơn vị \(The Benefits of Using assertThat over other Assert Methods in Unit Tests\)\*\*](#)

## **3.4) Thư viện hỗ trợ**

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

### **Bài 1) Tương tác người dùng**

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

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

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

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

- 3.1) Espresso cho việc kiểm tra UI**

## **CHƯƠNG 3. LÀM VIỆC TRONG NỀN**

### **Bài 1) Các tác vụ nền**

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

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

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

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

## **Bài 1) Tùy chọn và cài đặt**

### **1.1) Shared preferences**

Shared preferences cho phép bạn lưu trữ một lượng nhỏ dữ liệu nguyên thủy dưới dạng các cặp key/value trong một file trên thiết bị. Để lấy một đối tượng xử lý file preference, và để đọc, ghi và quản lý dữ liệu preference, hãy sử dụng lớp SharedPreferences. Android framework tự quản lý file shared preferences. File này có thể truy cập được đối với tất cả các thành phần của ứng dụng của bạn, nhưng không thể truy cập được đối với các ứng dụng khác.

Dữ liệu bạn lưu vào shared preferences khác với dữ liệu trong saved activity state, mà bạn đã học trong một chương trước:

- Dữ liệu trong saved activity instance state được giữ lại giữa các activity instance trong cùng một phiên người dùng.
- Shared preferences tồn tại dài dằng giữa các phiên người dùng. Shared preferences vẫn tồn tại ngay cả khi ứng dụng của bạn dừng và khởi động lại, hoặc nếu thiết bị khởi động lại.

Chỉ sử dụng shared preferences khi bạn cần lưu một lượng nhỏ dữ liệu dưới dạng các cặp key/value đơn giản. Để quản lý lượng lớn dữ liệu ứng dụng liên tục hơn, hãy sử dụng một phương pháp lưu trữ như thư viện Room hoặc cơ sở dữ liệu SQL.

### **Những điều bạn nên biết**

Bạn nên quen thuộc với:

- Tạo, xây dựng và chạy ứng dụng trong Android Studio
- Thiết kế bố cục với các nút và text view
- Sử dụng style và theme
- Lưu và khôi phục activity instance state

### **Những điều bạn sẽ học**

Bạn sẽ học cách:

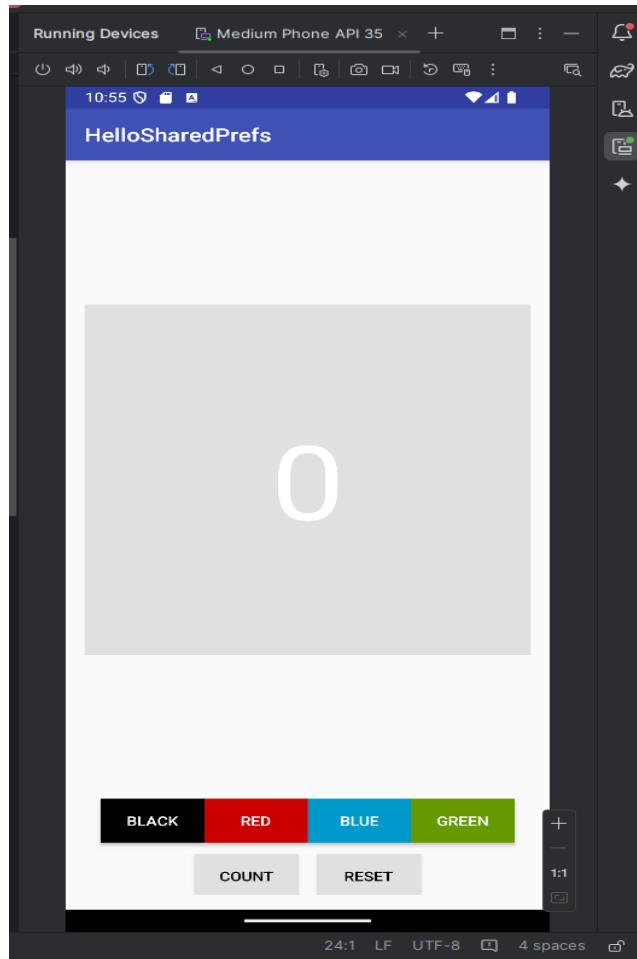
- Xác định shared preferences là gì
- Tạo một file shared preferences cho ứng dụng của bạn
- Lưu dữ liệu vào shared preferences và đọc lại các preference đó
- Xóa dữ liệu trong shared preferences

## Những điều bạn sẽ làm

- Cập nhật một ứng dụng để nó có thể lưu, truy xuất và đặt lại shared preferences

## Tổng quan về ứng dụng

Ứng dụng HelloSharedPrefs là một biến thể khác của ứng dụng HelloToast mà bạn đã tạo trong Bài 1. Nó bao gồm các nút để tăng số, thay đổi màu nền và đặt lại cả số và màu về mặc định của chúng. Ứng dụng này cũng sử dụng theme và style để xác định các nút.



Bạn bắt đầu với ứng dụng starter và thêm shared preferences vào mã activity chính. Bạn cũng thêm một nút reset để đặt cả số đếm và màu nền về mặc định, và xóa file preferences.

## Task 1: Khám phá HelloSharedPrefs

Dự án ứng dụng starter hoàn chỉnh cho bài thực hành này có sẵn tại HelloSharedPrefs-Starter. Trong task này, bạn tải dự án vào Android Studio và khám phá một số tính năng chính của ứng dụng.

### 1.1 Mở và chạy dự án HelloSharedPrefs

1. Tải xuống ứng dụng HelloSharedPrefs-Starter và giải nén file
2. Mở dự án trong Android Studio
3. Xây dựng và chạy ứng dụng. Hãy thử những điều sau:
  - o Nhấp vào nút **Count** để tăng số trong text view chính
  - o Nhấp vào bất kỳ nút màu nào để thay đổi màu nền của text view chính
  - o Xoay thiết bị và lưu ý rằng cả màu nền và số đếm đều được giữ lại
  - o Nhấp vào nút **Reset** để đặt màu và số đếm trở lại mặc định
4. Force-quit ứng dụng bằng một trong các phương pháp sau:
  - o Trong Android Studio, chọn **Run > Stop 'app'** hoặc nhấp vào biểu tượng Stop
  - o Trên thiết bị, nhấn nút Recents (nút hình vuông ở góc dưới bên phải). Vuốt thẻ ứng dụng HelloSharedPrefs để thoát ứng dụng, hoặc nhấp vào X ở góc bên phải của thẻ. Nếu bạn thoát ứng dụng theo cách này, hãy đợi một vài giây trước khi khởi động lại để hệ thống có thể dọn dẹp
5. Chạy lại ứng dụng

Ứng dụng khởi động lại với giao diện mặc định—số đếm là 0 và màu nền là xám.

### 1.2 Khám phá mã Activity

1. Mở MainActivity
2. Kiểm tra mã và lưu ý những điều sau:
  - o Số đếm (mCount) được định nghĩa là một số nguyên. Phương thức onClick countUp() tăng giá trị này và cập nhật TextView chính
  - o Màu (mColor) cũng là một số nguyên ban đầu được định nghĩa là màu xám trong file tài nguyên colors.xml dưới dạng default\_background
  - o Phương thức onClick changeBackground() lấy màu nền của nút đã được nhấp và sau đó đặt text view chính thành màu đó

- Cả số nguyên mCount và mColor đều được lưu vào bundle instance state trong onSaveInstanceState() và được khôi phục trong onCreate(). Các key bundle cho số đếm và màu được định nghĩa bởi các biến private (COUNT\_KEY) và (COLOR\_KEY)

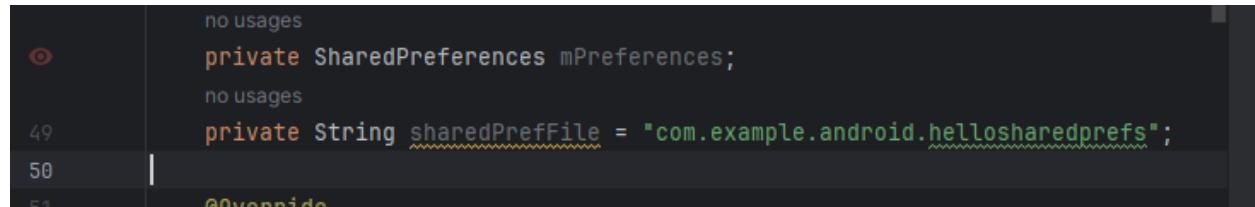
## Task 2: Lưu và khôi phục dữ liệu vào một file shared preferences

Trong task này, bạn lưu trạng thái của ứng dụng vào một file shared preferences và đọc lại dữ liệu đó khi ứng dụng được khởi động lại. Vì dữ liệu trạng thái mà bạn lưu vào shared preferences (số đếm và màu hiện tại) là **cùng** một dữ liệu mà bạn giữ lại trong instance state, bạn không phải thực hiện nó hai lần. Bạn có thể thay thế hoàn toàn instance state bằng shared preference state.

### 2.1 Khởi tạo preferences

1. Thêm các biến thành viên vào lớp MainActivity để giữ tên của file shared preferences và một tham chiếu đến đối tượng SharedPreferences.

```
private SharedPreferences mPreferences;
private String sharedPrefFile = "com.example.android.hellosharedprefs";
```



Bạn có thể đặt tên file shared preferences của bạn bất cứ điều gì bạn muốn, nhưng theo quy ước, nó có cùng tên với tên package của ứng dụng của bạn.

2. Trong phương thức onCreate(), khởi tạo shared preferences. Chèn mã này trước câu lệnh if:

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

Phương thức getSharedPreferences() (từ activity Context) mở file tại tên file đã cho (sharedPrefFile) với chế độ MODE\_PRIVATE.

*Lưu ý: Các phiên bản Android cũ hơn có các chế độ khác cho phép bạn tạo một file shared preferences có thể đọc hoặc ghi trên toàn thế giới. Các chế độ này đã bị deprecated trong*

API 17 và hiện không được khuyến khích vì lý do bảo mật. Nếu bạn cần chia sẻ dữ liệu với các ứng dụng khác, hãy cân nhắc sử dụng content URI được cung cấp bởi `FileProvider`.

```
private final String COLOR_KEY = "COLOR";  
1 usage  
private SharedPreferences mPreferences;  
1 usage  
private String sharedPrefFile = "com.example.android.hellosharedprefs";  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // Initialize views, color  
    mShowCountTextView = findViewById(R.id.count_textview);  
    mColor = ContextCompat.getColor(context: this,  
        R.color.default_background);  
  
    mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);  
  
    // Restore the saved instance state.  
    if (savedInstanceState != null) {  
  
        mCount = savedInstanceState.getInt(COUNT_KEY);  
        if (mCount != 0) {  
            mShowCountTextView.setText(String.format("%s", mCount));  
        }  
  
        mColor = savedInstanceState.getInt(COLOR_KEY);  
        mShowCountTextView.setBackgroundColor(mColor);  
    }  
}
```

## 2.2 Lưu preferences trong onPause()

Lưu preferences rất giống với việc lưu instance state -- cả hai thao tác đều đặt dữ liệu sang một đối tượng Bundle dưới dạng một cặp key/value. Tuy nhiên, đối với shared preferences, bạn lưu dữ liệu đó trong callback lifecycle `onPause()`, và bạn cần một đối tượng shared editor (`SharedPreferences.Editor`) để ghi vào đối tượng shared preferences.

1. Thêm phương thức lifecycle onPause() vào MainActivity.

```
@Override  
protected void onPause(){  
    super.onPause();  
    // ...  
}
```

2. Trong onPause(), lấy một editor cho đối tượng SharedPreferences:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

Cần có trình soạn thảo tùy chọn chia sẻ để ghi vào đối tượng tùy chọn chia sẻ. Thêm dòng này vào onPause() sau khi gọi super.onPause().

3. Sử dụng phương thức putInt() để đặt cả số nguyên mCount và mColor vào shared preferences với các key thích hợp:

```
preferencesEditor.putInt(COUNT_KEY, mCount);  
preferencesEditor.putInt(COLOR_KEY, mColor);
```

Lớp SharedPreferences.Editor bao gồm nhiều phương thức "put" cho các kiểu dữ liệu khác nhau, bao gồm putInt() và putString().

4. Gọi apply() để lưu preferences:

```
preferencesEditor.apply();
```

Phương thức apply() lưu preferences không đồng bộ, ngoài UI thread. Shared preferences editor cũng có một phương thức commit() để lưu preferences đồng bộ. Phương thức commit() không được khuyến khích vì nó có thể chặn các thao tác khác.

1. Xóa toàn bộ phương thức onSaveInstanceState(). Vì activity instance state chứa cùng dữ liệu như shared preferences, bạn có thể thay thế hoàn toàn instance state.

```
    @Override
    protected void onPause() {
        super.onPause();

        SharedPreferences.Editor preferencesEditor = mPreferences.edit();
        preferencesEditor.putInt(COUNT_KEY, mCount);
        preferencesEditor.putInt(COLOR_KEY, mColor);
        preferencesEditor.apply();
    }
}
```

## 2.3 Khôi phục preferences trong onCreate()

Như với instance state, ứng dụng của bạn đọc bất kỳ shared preferences đã lưu nào trong phương thức onCreate(). Một lần nữa, vì shared preferences chưa cùng dữ liệu như instance state, chúng ta có thể thay thế state bằng preferences ở đây. Mỗi khi onCreate() được gọi -- khi ứng dụng khởi động, trên các thay đổi cấu hình -- shared preferences được sử dụng để khôi phục state của view.

1. Xác định vị trí phần của phương thức onCreate() kiểm tra xem đối số savedInstanceState có null hay không và khôi phục instance state:

```
if (savedInstanceState != null) {

    mCount = savedInstanceState.getInt(COUNT_KEY);

    if (mCount != 0) {

        mShowCountTextView.setText(String.format("%s", mCount));

    }

    mColor = savedInstanceState.getInt(COLOR_KEY);

    mShowCountTextView.setBackgroundColor(mColor);

}
```

2. Xóa toàn bộ khối đó.
3. Trong phương thức onCreate(), ở cùng vị trí nơi có mã instance state, lấy số đếm từ preferences bằng key COUNT\_KEY và gán nó cho biến mCount.

```
mCount = mPreferences.getInt(COUNT_KEY, 0);
```

Khi bạn đọc dữ liệu từ preferences, bạn không cần phải lấy một shared preferences editor. Sử dụng bất kỳ phương thức "get" nào trên một đối tượng shared preferences (chẳng hạn như getInt() hoặc getString() để truy xuất dữ liệu preference).

Lưu ý rằng phương thức getInt() lấy hai đối số: một cho key và một cho giá trị mặc định nếu không tìm thấy key. Trong trường hợp này, giá trị mặc định là 0, giống như giá trị ban đầu của mCount.

4. Cập nhật giá trị của TextView chính bằng số đếm mới.

```
mShowCountTextView.setText(String.format("%s", mCount));
```

5. Lấy màu từ preferences bằng key COLOR\_KEY và gán nó cho biến mColor.

```
mColor = mPreferences.getInt(COLOR_KEY, mColor);
```

Như trước, đối số thứ hai cho getInt() là giá trị mặc định để sử dụng trong trường hợp key không tồn tại trong shared preferences. Trong trường hợp này, bạn có thể chỉ cần sử dụng lại giá trị của mColor, vừa được khởi tạo thành màu nền mặc định ở phía trên phương thức.

6. Cập nhật màu nền của text view chính.

```
mShowCountTextView.setBackgroundColor(mColor);
```

7. Chạy ứng dụng. Nhấn vào nút Count và thay đổi màu nền để cập nhật instance state và preferences.
8. Xoay thiết bị hoặc trình giả lập để xác minh rằng số đếm và màu được lưu trên các thay đổi cấu hình.
9. Force-quit ứng dụng bằng một trong các phương pháp sau:
  - Trong Android Studio, chọn Run > Stop 'app'.
  - Trên thiết bị, nhấn nút Recents (nút hình vuông ở góc dưới bên phải). Vuốt thẻ ứng dụng HelloSharedPrefs để thoát ứng dụng, hoặc nhấn vào X ở góc bên phải của thẻ.
10. Chạy lại ứng dụng. Ứng dụng khởi động lại và tải preferences, duy trì trạng thái.

Code cho phương thức onCreate() của MainActivity:

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

    // Initialize views, color
    mShowCountTextView = findViewById(R.id.count_textview);
    mColor = ContextCompat.getColor(context: this,
        R.color.default_background);

    mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

    // Restore the saved instance state.
    mCount = mPreferences.getInt(COUNT_KEY, i: 0);
    mShowCountTextView.setText(String.format("%s", mCount));
    mColor = mPreferences.getInt(COLOR_KEY, mColor);
    mShowCountTextView.setBackgroundColor(mColor);
}
```

## 2.4 Đặt lại preferences trong trình xử lý nhập reset()

Nút reset trong ứng dụng starter đặt lại cả số đếm và màu cho activity về các giá trị mặc định của chúng. Vì preferences giữ trạng thái của activity, điều quan trọng là phải xóa preferences cùng lúc.

- Trong phương thức onClick reset(), sau khi màu và số đếm được đặt lại, lấy một editor cho đối tượng SharedPreferences:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

- Xóa tất cả shared preferences:

```
preferencesEditor.clear();
```

- Áp dụng các thay đổi:

```
preferencesEditor.apply();
```

Code cho phương thức reset():

```
    }

    /**
     * 1 usage
     */
    public void reset(View view) {
        // Reset count
        mCount = 0;
        mShowCountTextView.setText(String.format("%s", mCount));

        // Reset color
        mColor = ContextCompat.getColor(context, R.color.default_background);
        mShowCountTextView.setBackgroundColor(mColor);
        // Clear preferences
        SharedPreferences.Editor preferencesEditor = mPreferences.edit();
        preferencesEditor.clear();
        preferencesEditor.apply();
    }

    @Override
```

## Tóm tắt

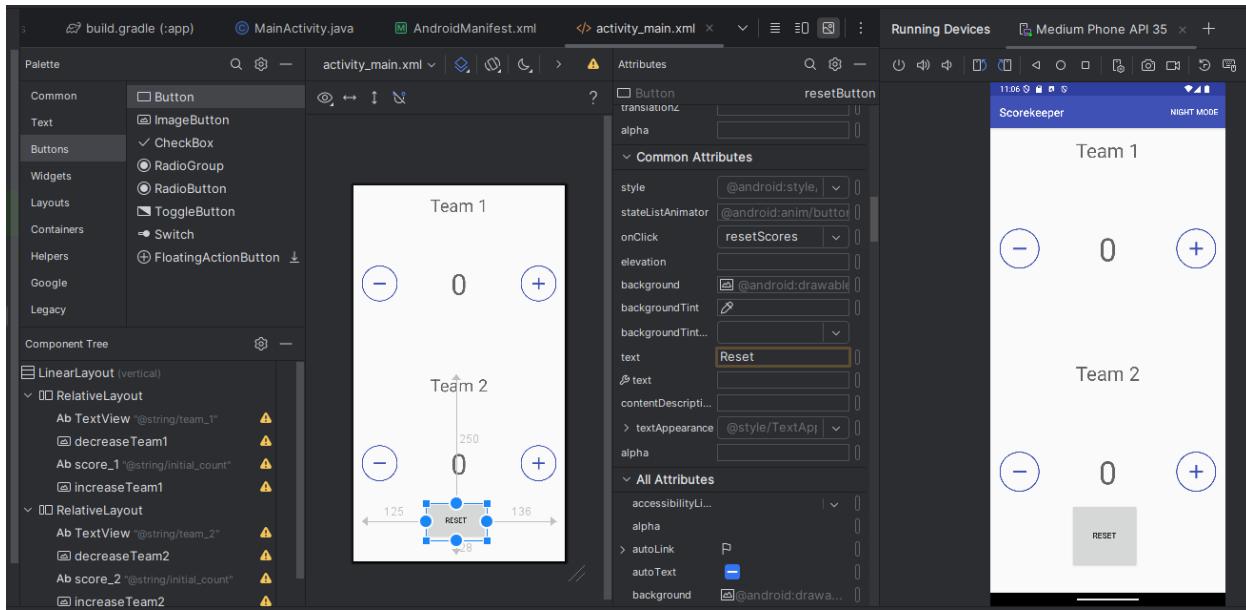
- Lớp SharedPreferences cho phép một ứng dụng lưu trữ một lượng nhỏ dữ liệu nguyên thủy dưới dạng các cặp key-value.
- Shared preferences tồn tại dài dằng giữa các phiên người dùng khác nhau của cùng một ứng dụng.
- Để ghi vào shared preferences, hãy lấy một đối tượng SharedPreferences.Editor.
- Sử dụng các phương thức "put" khác nhau trong một đối tượng SharedPreferences.Editor, chẳng hạn như putInt() hoặc putString(), để đặt dữ liệu vào shared preferences với một key và một value.
- Sử dụng các phương thức "get" khác nhau trong một đối tượng SharedPreferences, chẳng hạn như getInt() hoặc getString(), để lấy dữ liệu ra khỏi shared preferences bằng một key.
- Sử dụng phương thức clear() trong một đối tượng SharedPreferences.Editor để xóa tất cả dữ liệu được lưu trữ trong preferences.
- Sử dụng phương thức apply() trong một đối tượng SharedPreferences.Editor để lưu các thay đổi vào file preferences.

## Bài tập về nhà

Xây dựng và chạy một ứng dụng

Mở ứng dụng ScoreKeeper mà bạn đã tạo trong **bài Android fundamentals 5.1: Drawables, styles, and themes.**

2. Thay thế saved instance state bằng shared preferences cho mỗi điểm số.
3. Để kiểm tra ứng dụng, hãy xoay thiết bị để đảm bảo rằng các thay đổi cấu hình đọc các shared preferences đã lưu và cập nhật UI.
4. Dừng ứng dụng và khởi động lại để đảm bảo rằng các preferences đã được lưu.
5. Thêm một nút **Reset** để đặt lại các giá trị điểm về 0 và xóa shared preferences.



```
private SharedPreferences sharedPreferences;
1 usage
private static final String PREFS_NAME = "ScoreKeeperPrefs";
2 usages
private static final String SCORE_TEAM_1 = "score1";
2 usages
private static final String SCORE_TEAM_2 = "score2";

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

    mScoreText1 = findViewById(R.id.score_1);
    mScoreText2 = findViewById(R.id.score_2);

    sharedPreferences = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
    mScore1 = sharedPreferences.getInt(SCORE_TEAM_1, i: 0);
    mScore2 = sharedPreferences.getInt(SCORE_TEAM_2, i: 0);
    updateScoreUI();
}

4 usages
@Override
protected void onPause() {
    super.onPause();
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putInt(SCORE_TEAM_1, mScore1);
    editor.putInt(SCORE_TEAM_2, mScore2);
    editor.apply();
}
```

```
4 usages
private void updateScoreUI() {
    mScoreText1.setText(String.valueOf(mScore1));
    mScoreText2.setText(String.valueOf(mScore2));
}

2 usages
public void decreaseScore(View view) {
    if (view.getId() == R.id.decreaseTeam1) {
        mScore1--;
    } else if (view.getId() == R.id.decreaseTeam2) {...}
    updateScoreUI();
}

2 usages
public void increaseScore(View view) {
    if (view.getId() == R.id.increaseTeam1) {
        mScore1++;
    } else if (view.getId() == R.id.increaseTeam2) {
        mScore2++;
    }
    updateScoreUI();
}

1 usage
public void resetScores(View view) {
    mScore1 = 0;
    mScore2 = 0;
    sharedpreferences.edit().clear().apply();
    updateScoreUI();
}
```

## 1.2) Cài đặt ứng dụng

### Giới thiệu

Các ứng dụng thường bao gồm các cài đặt cho phép người dùng sửa đổi các tính năng và hành vi của ứng dụng. Ví dụ: một số ứng dụng cho phép người dùng đặt vị trí nhà của họ, các đơn vị mặc định cho các phép đo và các cài đặt khác áp dụng cho toàn bộ ứng dụng.

Người dùng không truy cập cài đặt thường xuyên, bởi vì một khi người dùng thay đổi một cài đặt, chẳng hạn như vị trí nhà, họ hiếm khi cần quay lại và thay đổi lại.

Người dùng mong đợi điều hướng đến cài đặt ứng dụng bằng cách nhấn vào **Settings** trong điều hướng bên, chẳng hạn như navigation drawer như được hiển thị ở phía bên trái của hình bên dưới, hoặc trong menu tùy chọn trong app bar, được hiển thị ở phía bên phải của hình bên dưới.

Trong hình trên:

1. Cài đặt trong điều hướng bên (navigation drawer)
2. Cài đặt trong menu tùy chọn của app bar

Trong bài thực hành này, bạn thêm một activity cài đặt vào một ứng dụng. Người dùng sẽ có thể điều hướng đến cài đặt ứng dụng bằng cách nhấn vào **Settings**, sẽ được đặt trong menu tùy chọn trong app bar.

## Những điều bạn nên biết

Bạn sẽ có thể:

- Tạo một dự án Android Studio từ một template và tạo bố cục chính.
- Chạy ứng dụng trên trình giả lập hoặc thiết bị đã kết nối.
- Tạo và chỉnh sửa các phần tử UI bằng trình chỉnh sửa bố cục và mã XML.
- Trích xuất tài nguyên chuỗi và chỉnh sửa các giá trị chuỗi.
- Truy cập các phần tử UI từ mã của bạn bằng findViewById().
- Xử lý một nhấp chuột Button.
- Hiển thị một tin nhắn Toast.
- Thêm một Activity vào một ứng dụng.
- Tạo một menu tùy chọn trong app bar.
- Thêm và chỉnh sửa các mục menu trong menu tùy chọn.
- Sử dụng style và theme trong một dự án.
- Sử dụng SharedPreferences.

## Những điều bạn sẽ học

Bạn sẽ học cách:

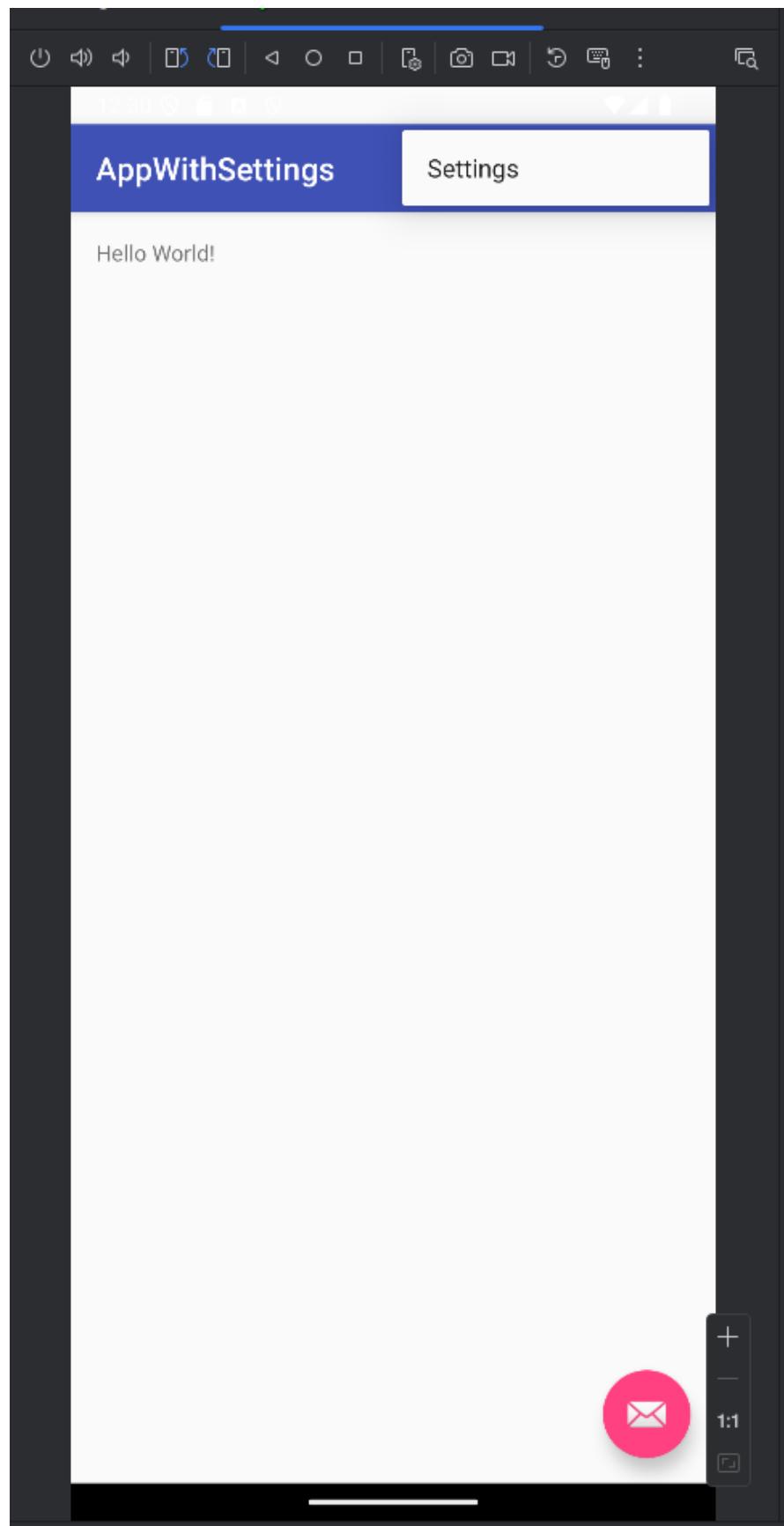
1. Thêm một Fragment để quản lý cài đặt.
2. Tạo một file tài nguyên XML của các cài đặt với các thuộc tính của chúng.
3. Tạo điều hướng đến Activity cài đặt.
4. Đặt các giá trị mặc định của cài đặt.
5. Đọc các giá trị cài đặt được thay đổi bởi người dùng.
6. Tùy chỉnh template Activity cài đặt.

## Những điều bạn sẽ làm

- Tạo một ứng dụng bao gồm Settings trong menu tùy chọn.
- Thêm một switch toggle tùy chọn Settings.
- Thêm mã để đặt giá trị mặc định cho cài đặt và truy cập giá trị cài đặt sau khi nó đã thay đổi.
- Sử dụng và tùy chỉnh template Activity cài đặt tiêu chuẩn được cung cấp bởi Android Studio.

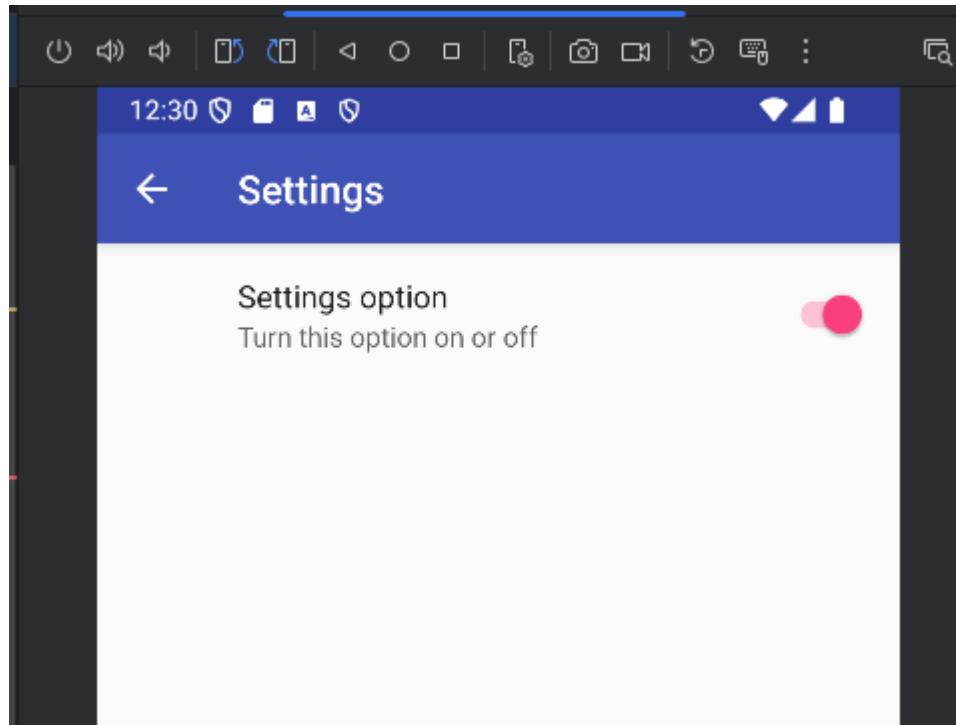
## Tổng quan về ứng dụng

Android Studio cung cấp một shortcut để thiết lập một menu tùy chọn với Settings. Nếu bạn bắt đầu một dự án Android Studio cho điện thoại hoặc máy tính bảng bằng template Basic Activity, ứng dụng mới bao gồm Settings như được hiển thị bên dưới:



Template cũng bao gồm một floating action button ở góc dưới bên phải của màn hình với một biểu tượng phong bì. Bạn có thể bỏ qua nút này cho bài thực hành này, vì bạn sẽ không sử dụng nó.

Bạn sẽ bắt đầu bằng cách tạo một ứng dụng có tên AppWithSettings bằng template Basic Activity, và bạn sẽ thêm một Activity cài đặt cung cấp một cài đặt switch toggle mà người dùng có thể bật hoặc tắt:

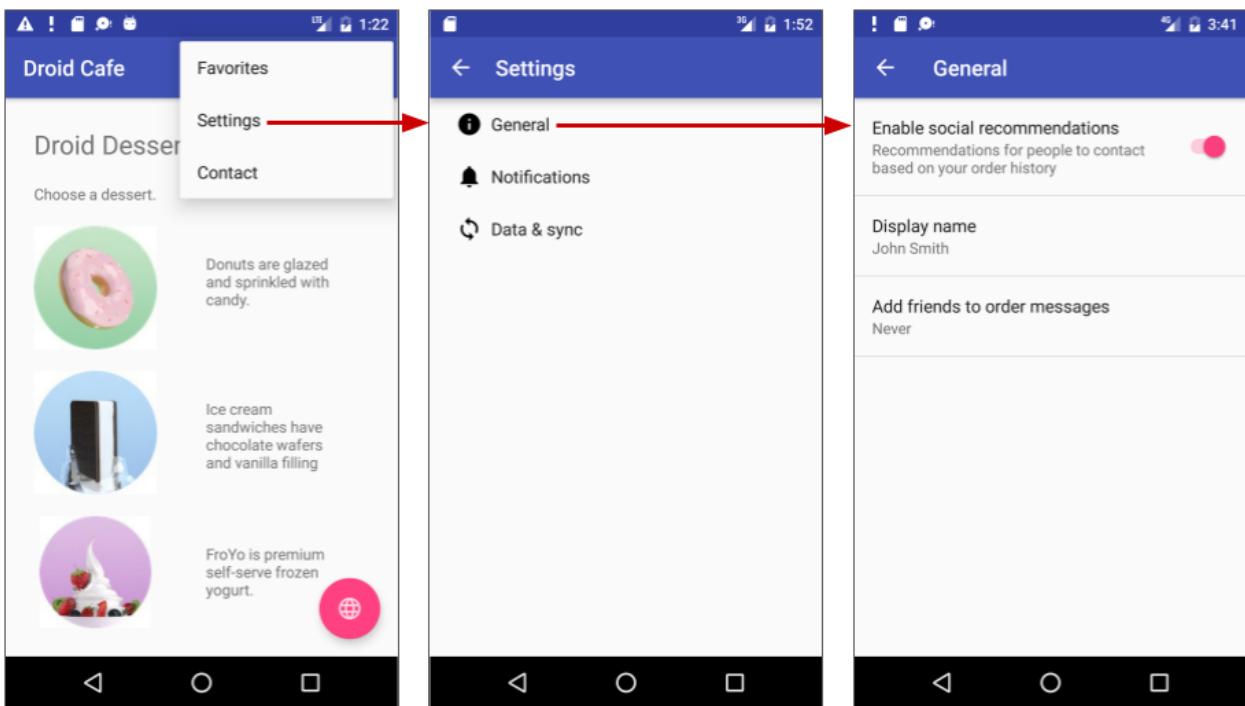


Bạn sẽ thêm mã để đọc cài đặt và thực hiện một hành động dựa trên giá trị của nó. Vì sự đơn giản, hành động sẽ là hiển thị một tin nhắn Toast với giá trị của cài đặt.

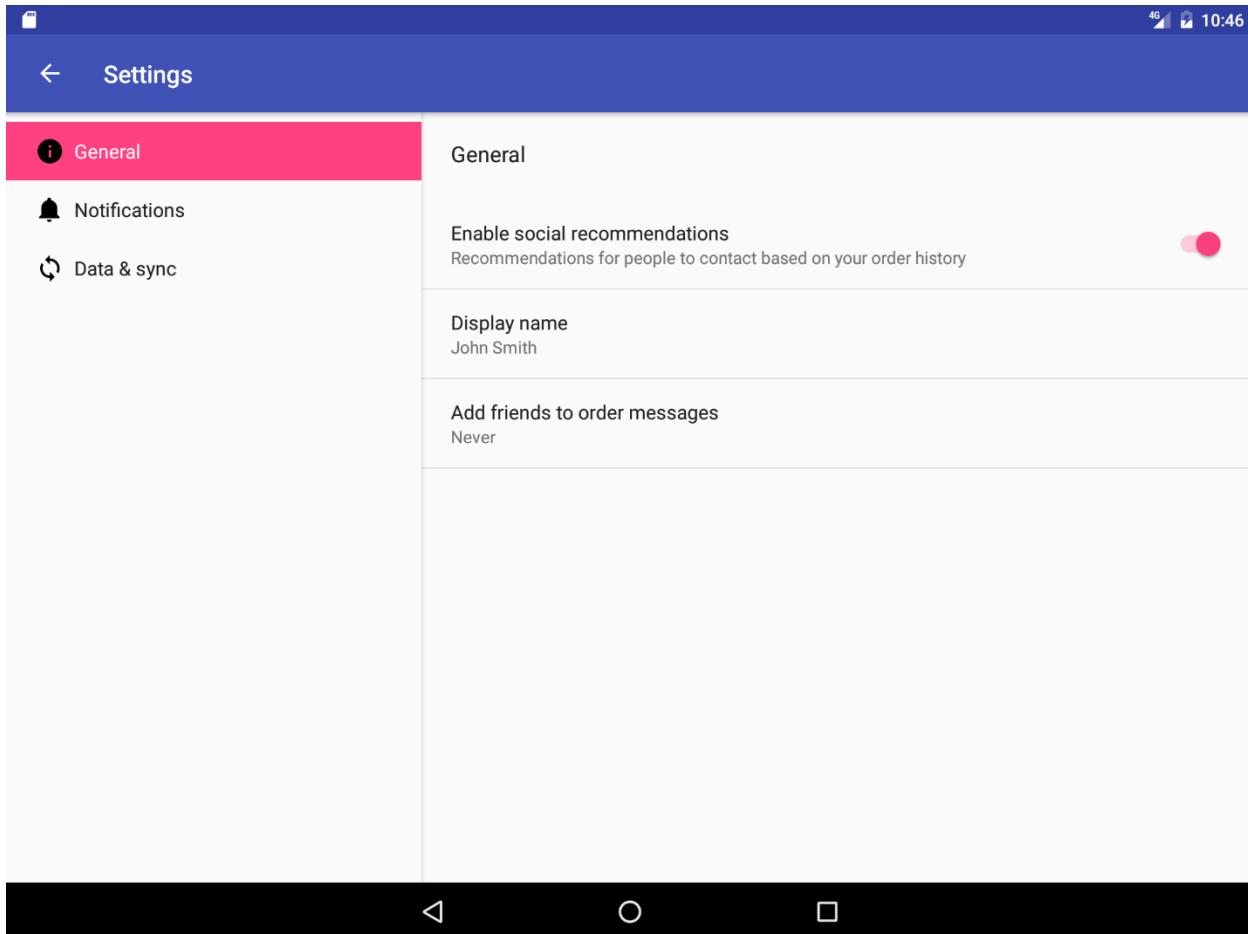
Trong task thứ hai, bạn sẽ thêm template Activity cài đặt tiêu chuẩn được cung cấp bởi Android Studio vào ứng dụng DroidCafeOptionsUp mà bạn đã tạo trong một bài học trước.

Template Activity cài đặt được điền sẵn với các cài đặt bạn có thể tùy chỉnh cho một ứng dụng và cung cấp một bố cục khác nhau cho điện thoại và máy tính bảng:

- **Điện thoại:** Một màn hình Settings chính với một liên kết tiêu đề cho mỗi nhóm cài đặt, chẳng hạn như General cho cài đặt chung, như được hiển thị bên dưới.



- **Máy tính bảng:** Một bố cục màn hình master/detail với một liên kết tiêu đề cho mỗi nhóm ở phía bên trái (master) và nhóm cài đặt ở phía bên phải (detail), như được hiển thị trong hình bên dưới.



Để tùy chỉnh template, bạn sẽ thay đổi các tiêu đề, tiêu đề cài đặt, mô tả cài đặt và giá trị cho các cài đặt.

Ứng dụng DroidCafeOptionsUp đã được tạo trong một bài học trước từ template Basic Activity, cung cấp một menu tùy chọn trong app bar để đặt tùy chọn Settings. Bạn sẽ tùy chỉnh template Activity cài đặt được cung cấp bằng cách thay đổi tiêu đề, mô tả, giá trị và giá trị mặc định của một cài đặt duy nhất. Bạn sẽ thêm mã để đọc giá trị của cài đặt sau khi người dùng thay đổi nó và hiển thị giá trị đó.

### Task 1: Thêm một cài đặt switch vào một ứng dụng

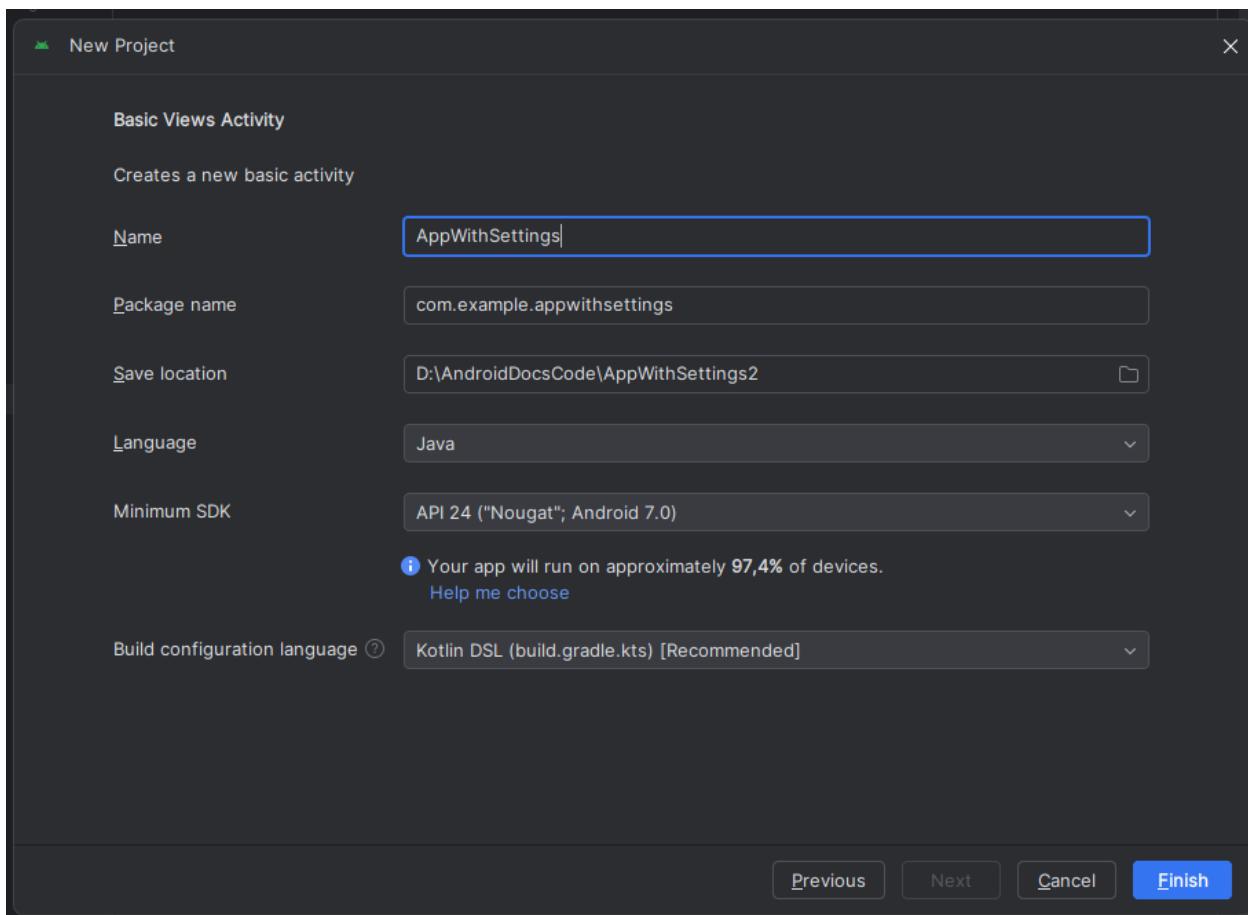
Trong task này, bạn thực hiện những điều sau:

- Tạo một dự án mới dựa trên template Basic Activity, cung cấp một menu tùy chọn.
- Thêm một switch toggle (SwitchPreference) với các thuộc tính trong một file XML preference.

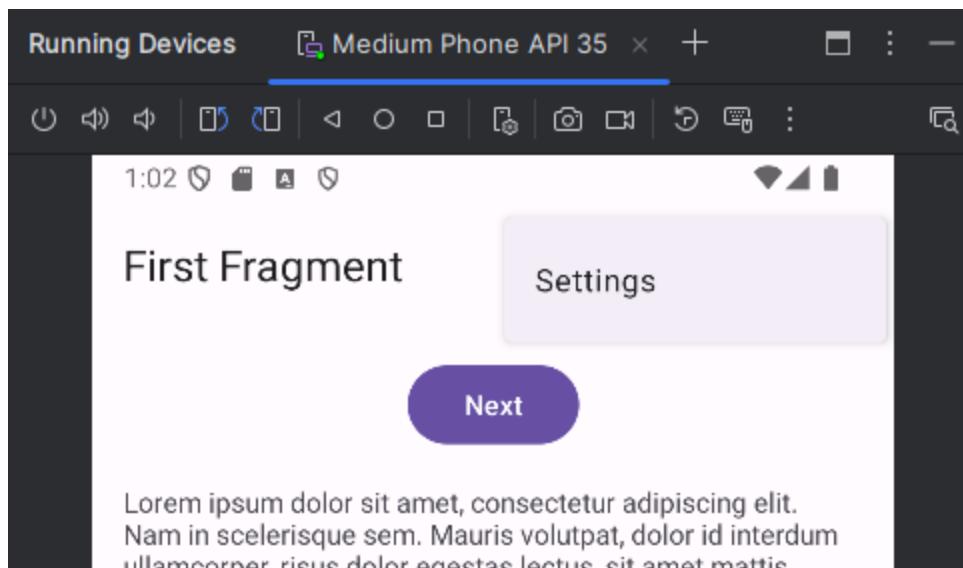
- Thêm một activity cho cài đặt và một fragment cho một cài đặt cụ thể. Để duy trì khả năng tương thích với AppCompatActivity, bạn sử dụng PreferenceFragmentCompat thay vì PreferenceFragment. Bạn cũng thêm thư viện android.support.v7.preference.
- Kết nối mục Settings trong menu tùy chọn với activity cài đặt.

## 1.1 Tạo dự án và thêm thư mục xml và file tài nguyên

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



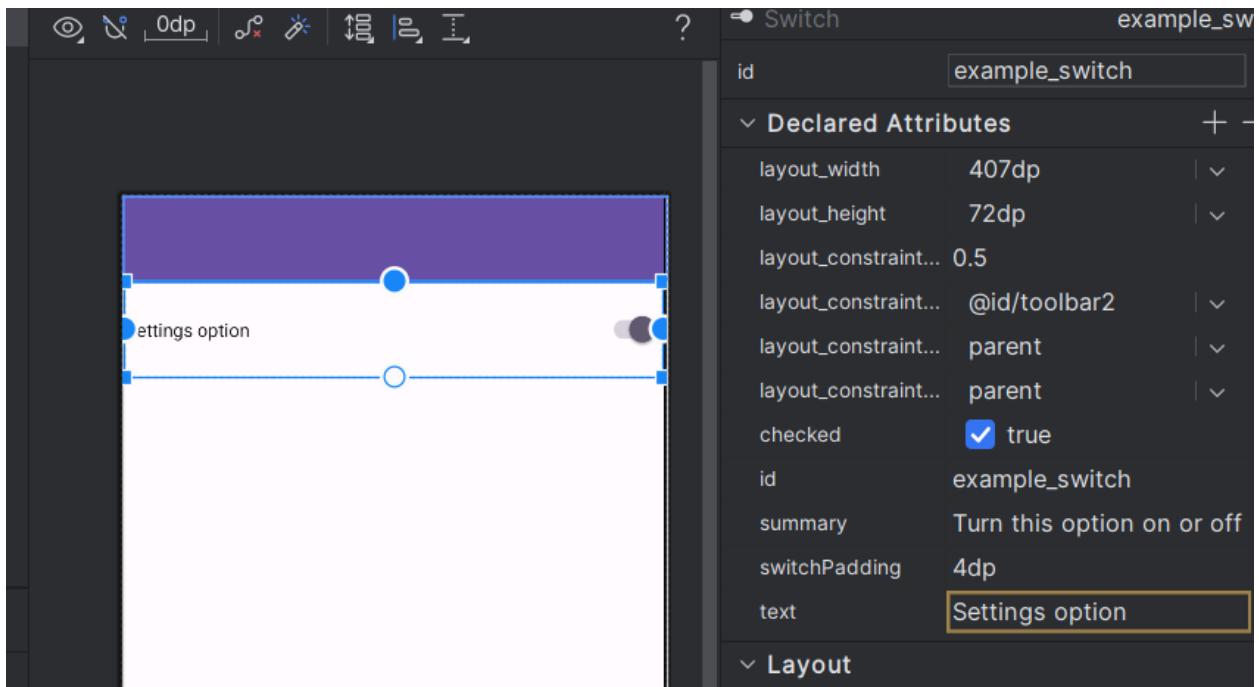
2. Chạy ứng dụng và nhấn vào biểu tượng overflow trong app bar để xem menu tùy chọn, như được hiển thị trong hình bên dưới. Mục duy nhất trong menu tùy chọn là **Settings**.



3. Bạn cần tạo một thư mục tài nguyên mới để giữ file XML chứa các cài đặt. Chọn thư mục res trong ngăn Project > Android, và chọn File > New > Android Resource Directory. Hộp thoại New Resource Directory xuất hiện.
4. Trong menu thả xuống Resource type, chọn xml. Directory name tự động thay đổi thành xml. Nhấp vào OK.
5. Thư mục xml xuất hiện trong ngăn Project > Android bên trong thư mục res. Chọn xml và chọn File > New > XML resource file (hoặc nhấp chuột phải vào xml và chọn New > XML resource file).
6. Nhập tên của file XML, preferences, vào trường File name và nhấp vào OK. File preferences.xml xuất hiện bên trong thư mục xml và trình chỉnh sửa bối cảnh xuất hiện, như được hiển thị trong hình bên dưới.

## 1.2 Thêm tùy chọn XML preference và thuộc tính cho cài đặt

Kéo một SwitchPreference từ bảng Palette ở phía bên trái lên phía trên cùng của bố cục, như được hiển thị trong hình bên dưới.



### 1.3 Thêm Activity cho cài đặt

Lưu và tải dữ liệu trong SettingsActivity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_settings);

    // Tìm Toolbar trong layout và đặt làm ActionBar
    Toolbar toolbar = findViewById(R.id.toolbar2);
    setSupportActionBar(toolbar); |

    // Hiển thị nút quay lại (Up button)
    Objects.requireNonNull(getSupportActionBar()).setDisplayHomeAsUpEnabled(true);

    // Lấy Switch từ layout
    switchNotifications = findViewById(R.id.example_switch);

    // Lấy SharedPreferences
    sharedPreferences = getSharedPreferences( name: "AppSettings", MODE_PRIVATE);

    // Tải giá trị đã lưu
    loadSettings();

    // Xử lý khi người dùng thay đổi trạng thái Switch
    switchNotifications.setOnCheckedChangeListener((buttonView, isChecked) -> saveSetting(isChecked));

    // Xử lý padding để hỗ trợ EdgeToEdge
    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));
    });
}
```

## 1.4 Cập nhật MainActivity cho cài đặt

```
@Override
protected void onResume() {
    super.onResume();
    updateSettings();
}

1 usage
private void updateSettings() {
    // Lấy SharedPreferences
    SharedPreferences sharedPreferences = getSharedPreferences( name: "AppSettings", MODE_
    boolean isNotificationsEnabled = sharedPreferences.getBoolean( key: "notifications", d

    // Hiển thị trạng thái mới nhất của Switch trong Snackbar
    Snackbar.make(binding.getRoot(),
        text: "Notifications: " + (isNotificationsEnabled ? "Enabled" : "Disabled"),
        Snackbar.LENGTH_LONG).show();
}
```

## Task 2: Sử dụng mẫu Activity Settings

Nếu bạn cần xây dựng một số màn hình con của cài đặt và bạn muốn tận dụng lợi thế của màn hình kích thước máy tính bảng cũng như duy trì khả năng tương thích với các phiên bản Android cũ hơn cho máy tính bảng, Android Studio cung cấp một lối tắt: mẫu Activity Settings.

Trong task trước đó, bạn đã học cách sử dụng một settings Activity trống và một Fragment trống để thêm một cài đặt vào một ứng dụng. Task 2 bây giờ sẽ cho bạn thấy cách sử dụng mẫu Settings Activity được cung cấp bởi Android Studio để:

- Chia nhiều cài đặt thành các nhóm.
- Tùy chỉnh các cài đặt và giá trị của chúng.
- Hiển thị màn hình chính Settings với một liên kết tiêu đề cho từng nhóm cài đặt, chẳng hạn như General cho các cài đặt chung, như hiển thị trong hình bên dưới.

### 2.1 Khám phá mẫu Activity Settings

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

**2.1) Room, LiveData và ViewModel**

**2.2) Room, LiveData và ViewModel**

## **Bài 3)**