

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



# **GIÁO TRÌNH**

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

Hà Nội, 2.2025

# MỤC LỤC

CHƯƠNG 1.	Làm quen.....	3
Bài 1)	Tạo ứng dụng đầu tiên .....	3
1.1)	Android Studio và Hello World .....	3
1.2)	Giao diện người dùng tương tác đầu tiên .....	5
1.3)	Trình chỉnh sửa bố cục .....	9
1.4)	Văn bản và các chế độ cuộn .....	42
1.5)	Tài nguyên có sẵn.....	42
Bài 2)	Activities .....	42
2.1)	Activity và Intent .....	42
2.2)	Vòng đời của Activity và trạng thái .....	42
2.3)	Intent ngầm định.....	42
Bài 3)	Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ .....	42
3.1)	Trình gỡ lỗi .....	42
3.2)	Kiểm thử đơn vị.....	42
3.3)	Thư viện hỗ trợ.....	42
CHƯƠNG 2.	Trải nghiệm người dùng .....	43
Bài 1)	Tương tác người dùng .....	43
1.1)	Hình ảnh có thể chọn .....	43
1.2)	Các điều khiển nhập liệu .....	43
1.3)	Menu và bộ chọn .....	43
1.4)	Điều hướng người dùng .....	43
1.5)	RecyclerView .....	43
Bài 2)	Trải nghiệm người dùng thú vị.....	43
2.1)	Hình vẽ, định kiểu và chủ đề .....	43
2.2)	Thẻ và màu sắc .....	43

2.3)	Bố cục thích ứng.....	43
Bài 3)	Kiểm thử giao diện người dùng.....	43
3.1)	Espresso cho việc kiểm tra UI .....	43
CHƯƠNG 3. Làm việc trong nền .....		43
Bài 1)	Các tác vụ nền.....	43
1.1)	AsyncTask .....	43
1.2)	AsyncTask và AsyncTaskLoader .....	43
1.3)	Broadcast receivers .....	43
Bài 2)	Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền.....	43
2.1)	Thông báo .....	43
2.2)	Trình quản lý cảnh báo .....	43
2.3)	JobScheduler.....	43
CHƯƠNG 4. Lưu dữ liệu người dùng .....		44
Bài 1)	Tùy chọn và cài đặt.....	44
1.1)	Shared preferences.....	44
1.2)	Cài đặt ứng dụng.....	44
Bài 2)	Lưu trữ dữ liệu với Room .....	44
2.1)	Room, LiveData và ViewModel.....	44
2.2)	Room, LiveData và ViewModel.....	44
3.1)	Trình gỡ lỗi .....	

## CHƯƠNG 1. LÀM QUEN

### Bài 1) Tạo ứng dụng đầu tiên

#### 1.1) Android Studio và Hello World

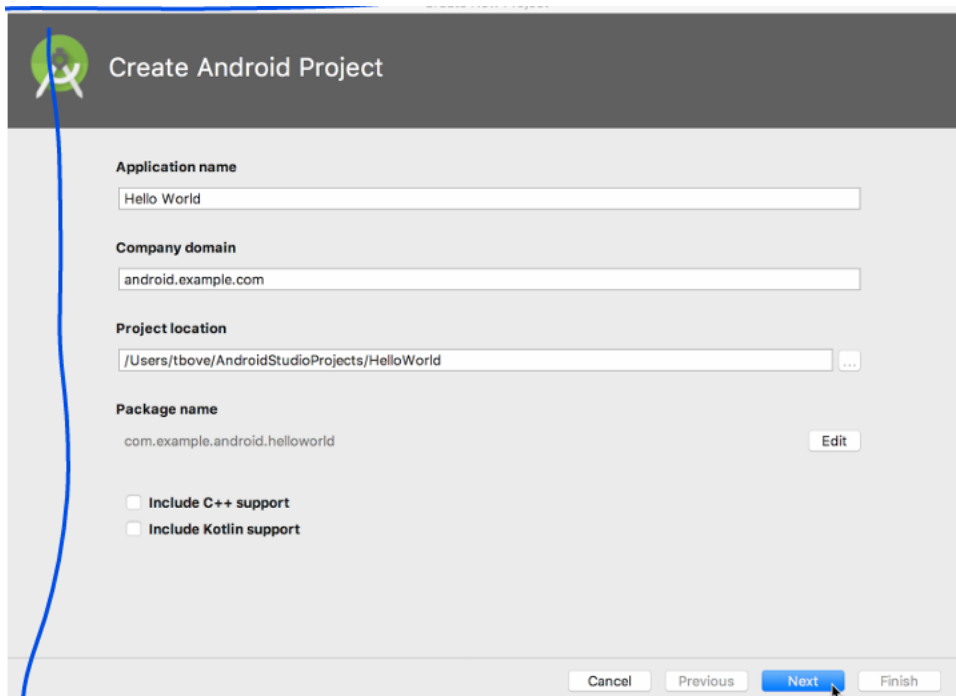
### Giới thiệu

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

## Những gì Bạn nên biết

Bạn nên có khả năng:

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



## Những gì Bạn sẽ cần:

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

## Những gì bạn sẽ học

- Cách cài đặt và sử dụng IDE Android Studio.
- Cách sử dụng quy trình phát triển để xây dựng ứng dụng Android.
- Cách tạo một dự án Android từ một mẫu.
- Cách thêm thông điệp ghi lại vào ứng dụng của bạn để phục vụ mục đích gỡ lỗi.

## Những gì bạn sẽ làm

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

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

## Giao diện

Giao diện người dùng (UI) xuất hiện trên màn hình của thiết bị Android bao gồm một hệ thống phân cấp các đối tượng gọi là views – mọi phần tử của màn hình đều là 1 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 như nút bấm, checkboxes, và trường nhập văn bản. Các lớp của View con phổ biến sẽ được mô tả ở bài tập sau:

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

Mã java điều khiển và hiển thị giao diện người dùng được chứa trong 1 lớp kế thừa Activity. 1 Activity thường được liên kết với 1 bố cục của UI views đc định nghĩa trong tập tin xml. Tập tin XML thường được đặt tên theo Activity tương ứng và định nghĩa bố cục của phần tử View trên màn hình.

Ví dụ, trong ứng dụng HelloWorld mã trong MainActivity hiển thị bố cục được định nghĩa trong tập tin activity\_main.xml, trong đó có 1 TextView với nội dung “Hello World”.

Trong các ứng dụng phức tạp hơn, 1 Activity phải thực thi nhiều 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 Activity trong 1 bài học khác.

Trong bài thực hành này, bạn học được cách tạo ứng dụng tương tác đầu tiên- 1 ứng dụng có thể cho phép người dùng tương tác. Bạn tạo 1 ứng dụng sử dụng mẫu Empty Activity. Bạn cũng học cách sử dụng trình chỉnh sửa bố cục, và làm thế nào để chỉnh sửa bố cục trong XML. Bạn cần phát triển các kỹ năng này để bạn có thể hoàn thành các bài thực hành khác trong khóa học này.

### Những kiến thức bạn cần biết trước

- Cách cài đặt và mở Android Studio.
- Cách tạo 1 ứng dụng HelloWorld.
- Cách chạy ứng dụng HelloWorld.

### Những kiến thức bạn sẽ tìm hiểu

- Làm thế nào để tạo 1 ứng dụng với giao diện tương tác.
- Làm thế nào sử dụng trình chỉnh sửa bố cục để thiết kế 1 bố cục.
- Làm thế nào để chỉnh sửa bố cục trong XML.
- Nhiều thuật ngữ mới. Kiểm tra trong [Vocabulary and concepts glossary](#) để hiểu rõ hơn về các định nghĩa

### Những cái bạn sẽ làm

- 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 giao diện người dùng (UI).
- Chỉnh sửa bố cục của ứng dụng bằng **XML**.
- Trích xuất các chuỗi mã cứng (**hardcoded strings**) vào tài nguyên chuỗi (**string resources**).
- Triển khai phương thức xử lý sự kiện nhấp (**click-handler methods**) để hiển thị thông báo trên màn hình khi người dùng nhấn vào **Button**.

# Tổng quan về ứng dụng

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

- Khi người dùng nhấn vào **Button** đầu tiên, một thông báo ngắn (**Toast**) sẽ hiển thị trên màn hình.
- Khi nhấn vào **Button** thứ hai, bộ đếm số lần nhấn (**click counter**) sẽ tăng lên. Giá trị của bộ đếm được hiển thị trong **TextView**, bắt đầu từ 0.

Đây là giao diện của ứng dụng sau khi hoàn thành:

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

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

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

### 1.1 Tạo dự án Android Studio

*a) Mở Android Studio và tạo một dự án mới với các thông số sau:*

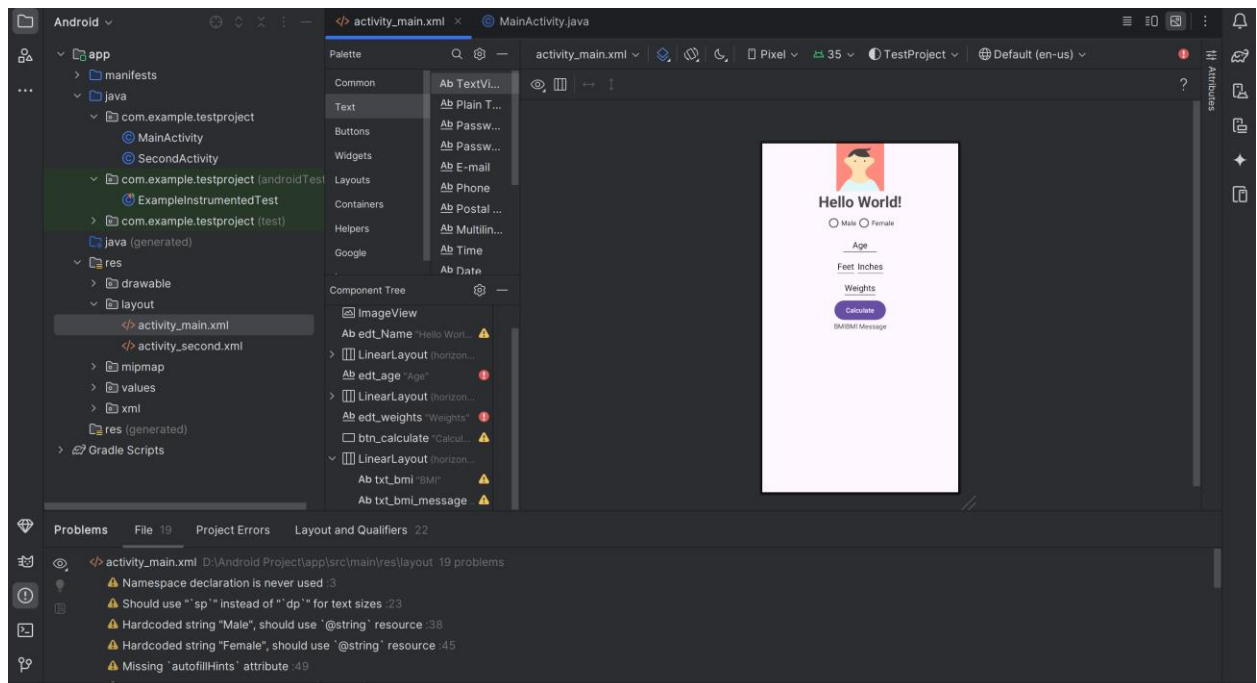
Attribute	Value
Application Name	<b>Hello Toast</b>
Company Name	<b>com.example.android</b> (or your own domain)
Phone and Tablet Minimum SDK	<b>API15: Android 4.0.3 IceCreamSandwich</b>
Template	<b>Empty Activity</b>
Generate Layout file box	Selected
Backwards Compatibility box	Selected

15. 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 của bạn.

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

**Android Studio** cung cấp **layout editor** để giúp bạn nhanh chóng xây dựng bố cục giao diện người dùng (**UI**). Công cụ này cho phép bạn:

- Kéo thả các phần tử vào **chế độ thiết kế trực quan** và **chế độ bản vẽ**.
- Sắp xếp vị trí các phần tử trong bố cục.
- Thêm **constraints** (ràng buộc) để xác định vị trí của phần tử trong giao diện.



1. Trong **app > res > thư mục layout** trong **Project > Android** panel, điều hướng đến **app > res > layout**, sau đó **double-click** vào tệp **activity\_main.xml** để mở (nếu chưa mở).
2. Nhấn vào tab **Design** (nếu chưa được chọn).
  - Tab **Design** dùng để thao tác với các phần tử và bố cục.
  - Tab **Text** dùng để chỉnh sửa mã XML của bố cục.
3. **Palettes pane** hiển thị các phần tử UI mà bạn có thể sử dụng trong bố cục ứng dụng.
4. **Component tree pane** hiển thị **cây phân cấp của các phần tử UI**.
  - Các **View** được tổ chức theo cấu trúc cha - con, trong đó **con** kế thừa thuộc tính từ **cha**.



- Ví dụ trong hình minh họa, **TextView** là **con** của **ConstraintLayout**.
5. **Design & Blueprint panes** hiển thị giao diện bố cục với các phần tử UI.
- Trong hình minh họa, bố cục chỉ có một phần tử: **TextView** hiển thị dòng chữ "Hello World".
6. **Attributes tab** hiển thị các thuộc tính của phần tử UI và cho phép chỉnh sửa trực tiếp.

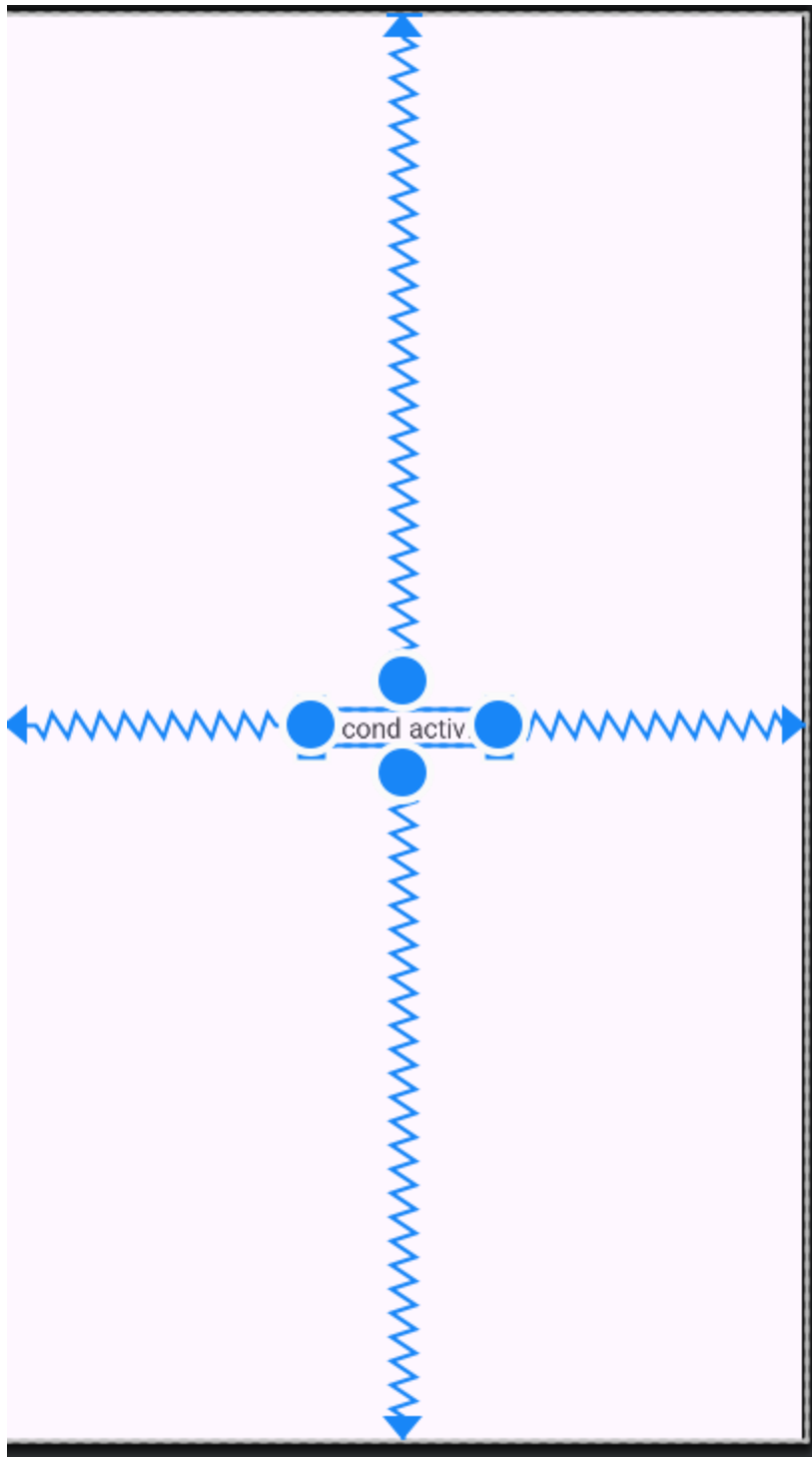
## Task 2: Thêm các phần tử View trong Layout Editor

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

### 2.1 Kiểm tra các ràng buộc của phần tử

**Thực hiện các bước sau:**

1. Mở activity\_main.xml từ Project > Android pane nếu nó chưa được mở. Nếu tab Design chưa được chọn, nhấp vào nó. Nếu không có blueprint, nhấp vào nút Select Design Surface trong thanh công cụ và chọn Design + Blueprint.
2. Công cụ Autoconnect cũng nằm trong thanh công cụ. Nó được bật theo mặc định. Đối với bước này, đảm bảo rằng công cụ không bị tắt.
3. Nhấp vào nút zoom in để phóng to khu vực thiết kế và blueprint để có cái nhìn cận cảnh.
4. Chọn TextView trong Component Tree pane. TextView "Hello World" sẽ được làm nổi bật trong các pane thiết kế và blueprint, và các ràng buộc của phần tử sẽ hiển thị.
5. Tham khảo hình động bên dưới cho bước này. Nhấp vào tay nắm tròn ở phía bên phải của TextView để xóa ràng buộc ngang đang liên kết view với phía bên phải của layout. TextView sẽ nhảy sang bên trái vì nó không còn bị ràng buộc với phía bên phải. Để thêm lại ràng buộc ngang, nhấp vào cùng tay nắm đó và kéo một đường đến phía bên phải của layout.



Trong các pane blueprint hoặc design, các tay nắm sau xuất hiện trên phần tử TextView:

- Tay nắm ràng buộc: Để tạo một ràng buộc như trong hình động ở trên, nhấp vào tay nắm ràng buộc, được hiển thị dưới dạng một vòng tròn ở

cạnh của một phần tử. Sau đó, kéo tay nắm đến một tay nắm ràng buộc khác hoặc đến ranh giới của phần tử cha. Một đường zic-zac sẽ biểu thị ràng buộc.



- Tay nắm thay đổi kích thước: Để thay đổi kích thước phần tử, kéo các tay nắm hình vuông. Tay nắm sẽ chuyển thành một góc xiên khi bạn kéo nó.



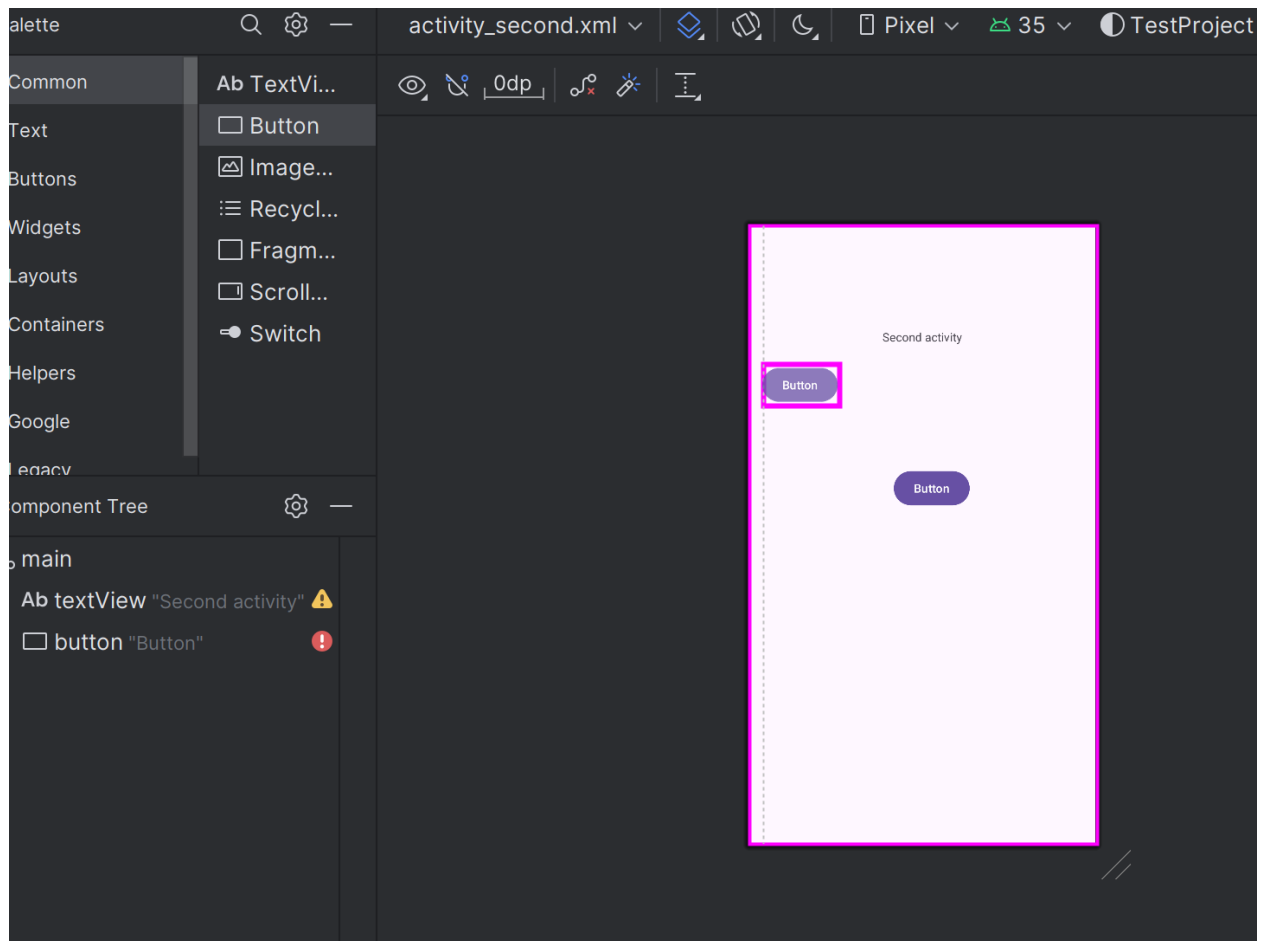
## 2.2 Thêm Button vào Layout

Khi được bật, công cụ Autoconnect tự động tạo hai hoặc nhiều ràng buộc cho một phần tử giao diện người dùng với layout cha. Sau khi bạn kéo phần tử vào layout, nó sẽ tạo ràng buộc dựa trên vị trí của phần tử.

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

1. Bắt đầu với một bố cục trống. Phần tử **TextView** không cần thiết, vì vậy khi nó vẫn được chọn, nhấn phím **Delete** hoặc chọn **Edit > Delete**. Bây giờ bạn có một layout hoàn toàn trống.

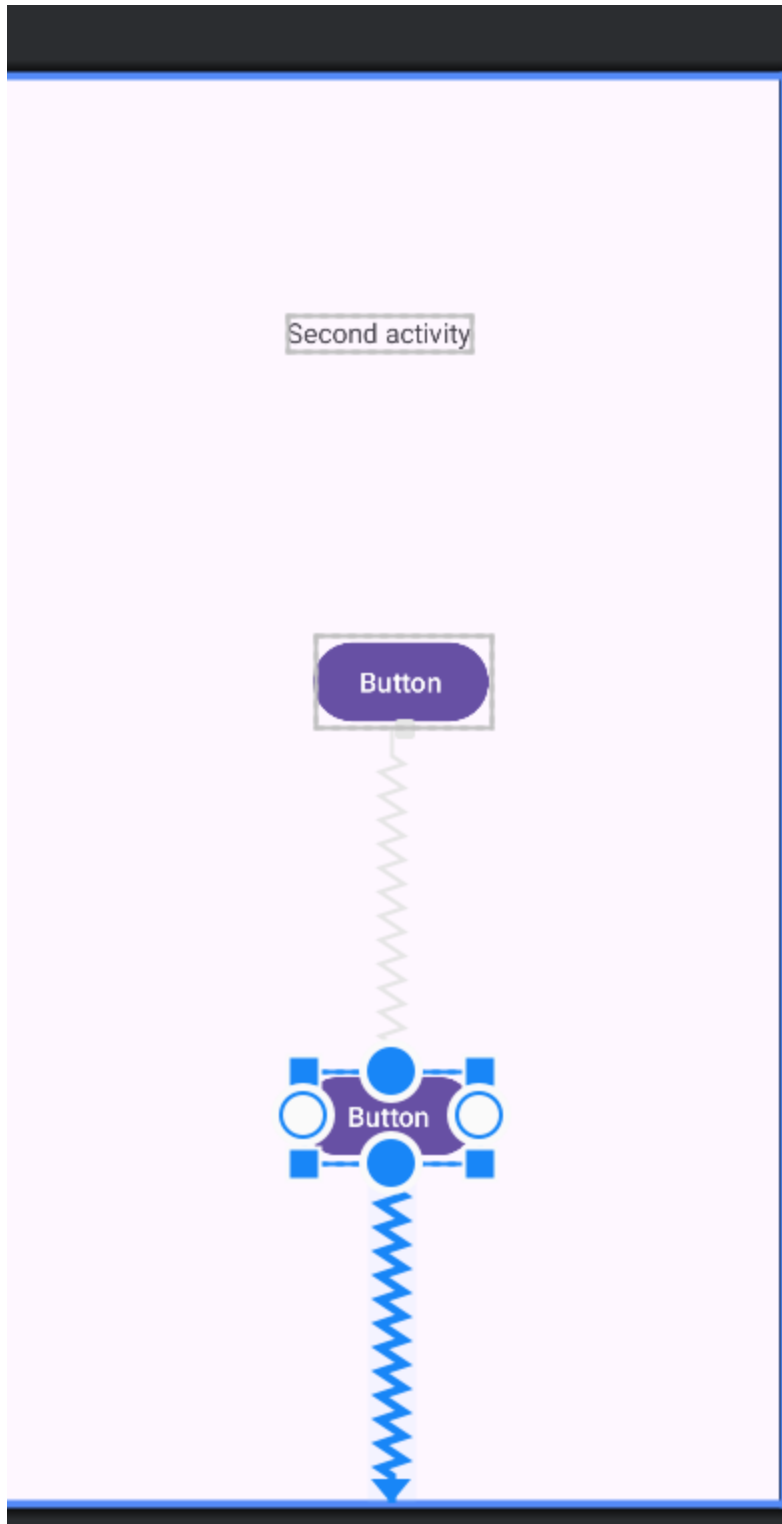
2. Kéo một **Button** từ **Palette** pane đến bất kỳ vị trí nào trong layout. Nếu bạn thả **Button** vào khu vực giữa phía trên của layout, các ràng buộc có thể tự động xuất hiện. Nếu không, bạn có thể kéo các ràng buộc đến cạnh trên, cạnh trái và cạnh phải của layout như trong hình động bên dưới.



## 2.3 Thêm một Button thứ hai vào layout

1. Kéo một **Button** khác từ **Palette** pane vào giữa layout như trong hình động bên dưới. **Autoconnect** có thể tự động tạo các ràng buộc ngang cho bạn (nếu không, bạn có thể tự kéo chúng).

2. Kéo một ràng buộc dọc đến cạnh dưới của layout (tham khảo hình bên dưới)



Bạn có thể xóa ràng buộc khỏi một phần tử bằng cách chọn phần tử và di chuột qua nó để hiển thị nút **Clear Constraints**. Nhấp vào nút này để xóa **tất cả** ràng buộc trên phần tử đã chọn. Để xóa một ràng buộc cụ thể, nhấp vào tay nắm ràng buộc tương

ứng.

Để xóa tất cả ràng buộc trong toàn bộ layout, nhấp vào công cụ **Clear All Constraints** trên thanh công cụ. Công cụ này hữu ích nếu bạn muốn thiết lập lại toàn bộ ràng buộc trong layout.

## Task 3: Thay đổi thuộc tính của phần tử UI

Pane **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 **View** 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 của **Button**, những thuộc tính này cũng áp dụng cho hầu hết các loại **View**.

### 3.1 Thay đổi kích thước Button

**Layout editor** cung cấp các tay nắm thay đổi kích thước ở cả bốn góc của một **View**, cho phép bạn thay đổi kích thước nhanh chóng. Bạn có thể kéo các tay nắm ở mỗi góc của **View** để thay đổi kích thước, nhưng cách này sẽ cố định (hardcode) các giá trị **width** và **height**.

Tránh hardcode kích thước cho hầu hết các phần tử **View**, vì kích thước cố định không thể thích ứng với nội dung và kích thước màn hình khác nhau.

Thay vào đó, sử dụng **Attributes pane** ở phía bên phải của **layout editor** để chọn chế độ kích thước không sử dụng giá trị cố định. **Attributes pane** bao gồm một **view inspector** (bảng điều khiển kích thước) hình vuông ở trên cùng. Các biểu tượng bên trong hình vuông này đại diện cho các thiết lập **height** và **width** như sau:

## Attributes



☐ Button

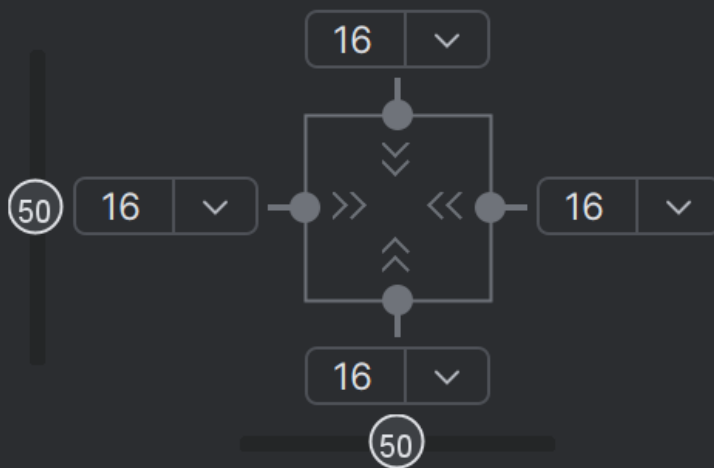
button2

id

button2

### Layout

#### Constraint Widget



### Constraints

Start → StartOf **parent** (16dp)

End → EndOf **parent** (16dp)

Top → TopOf **parent** (16dp)

Bottom → BottomOf **parent** (16dp)

layout\_width

wrap\_content



layout\_height

wrap\_content



visibility

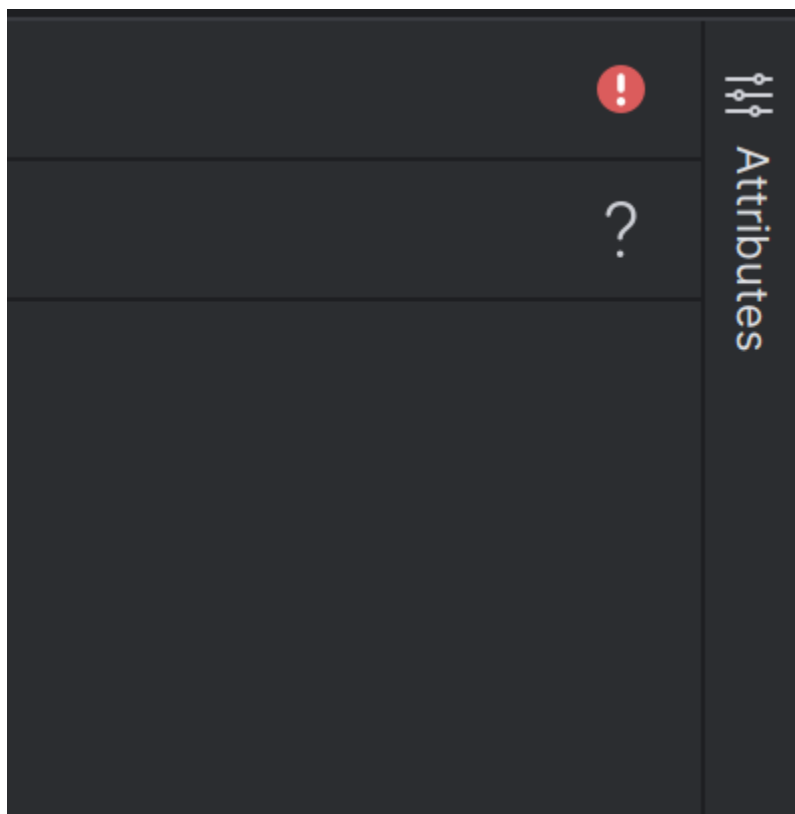


Trong hình trên:

1. **Height control:** Điều khiển này xác định thuộc tính `layout_height` và xuất hiện dưới dạng hai đoạn ở phía trên và dưới của hình vuông. Các góc nghiêng cho thấy điều khiển này được đặt thành `wrap_content`, nghĩa là **View** sẽ mở rộng theo chiều dọc để phù hợp với nội dung. Số "8" biểu thị một **margin tiêu chuẩn** được đặt thành **8dp**.
2. **Width control:** Điều khiển này xác định thuộc tính `layout_width` và xuất hiện dưới dạng hai đoạn ở phía trái và phải của hình vuông. Các góc nghiêng cho thấy điều khiển này được đặt thành `wrap_content`. điều này có nghĩa là **View** sẽ mở rộng theo chiều ngang để phù hợp với nội dung, tối đa đến **margin 8dp**.
3. **Nút đóng Attributes pane:** Nhấp vào để đóng pane.

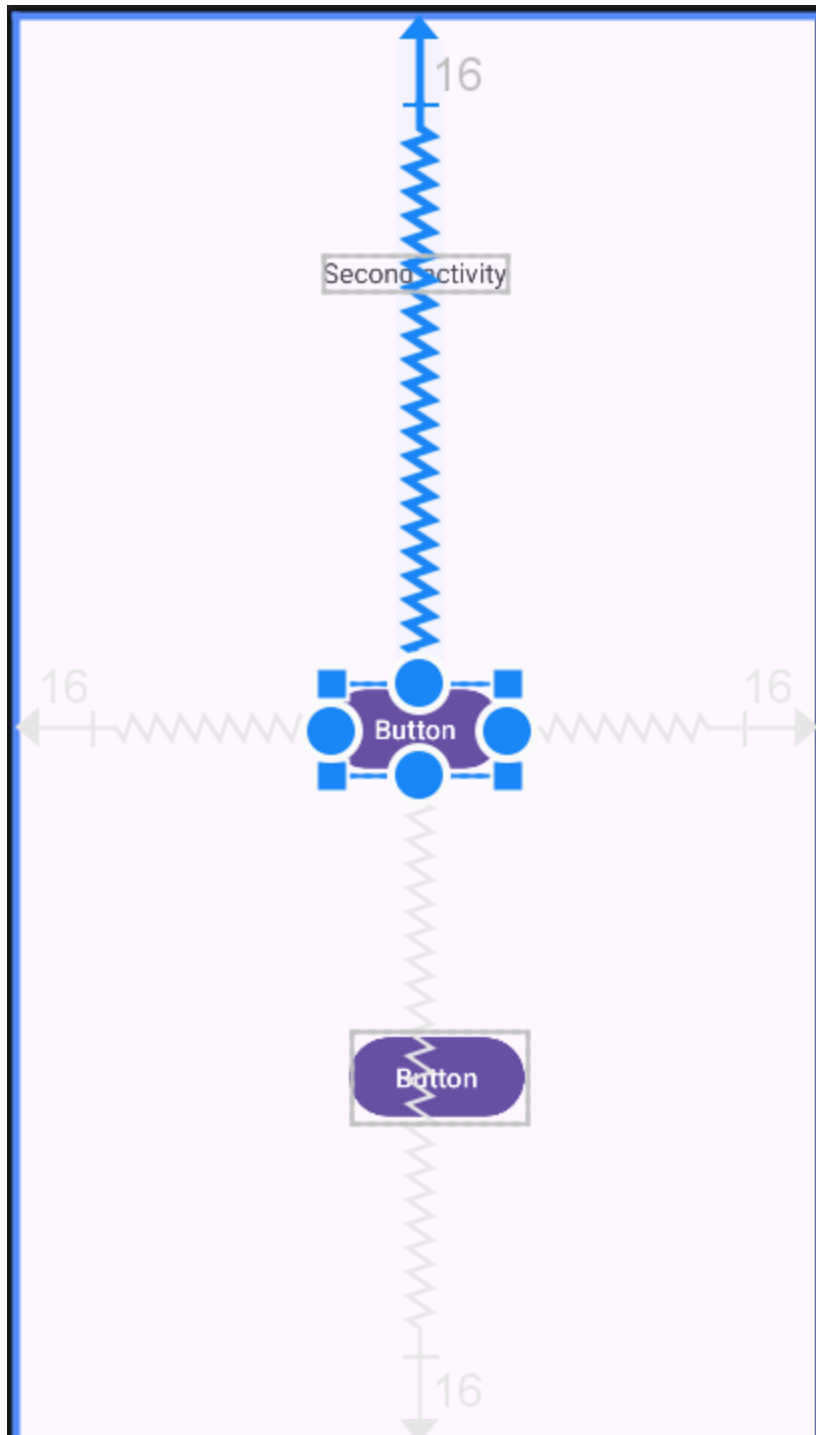
Thực hiện các bước sau:

1. Chọn **Button** phía trên trong **Component Tree** pane.
2. Nhấp vào tab **Attributes** ở phía bên phải của cửa sổ **layout editor**.



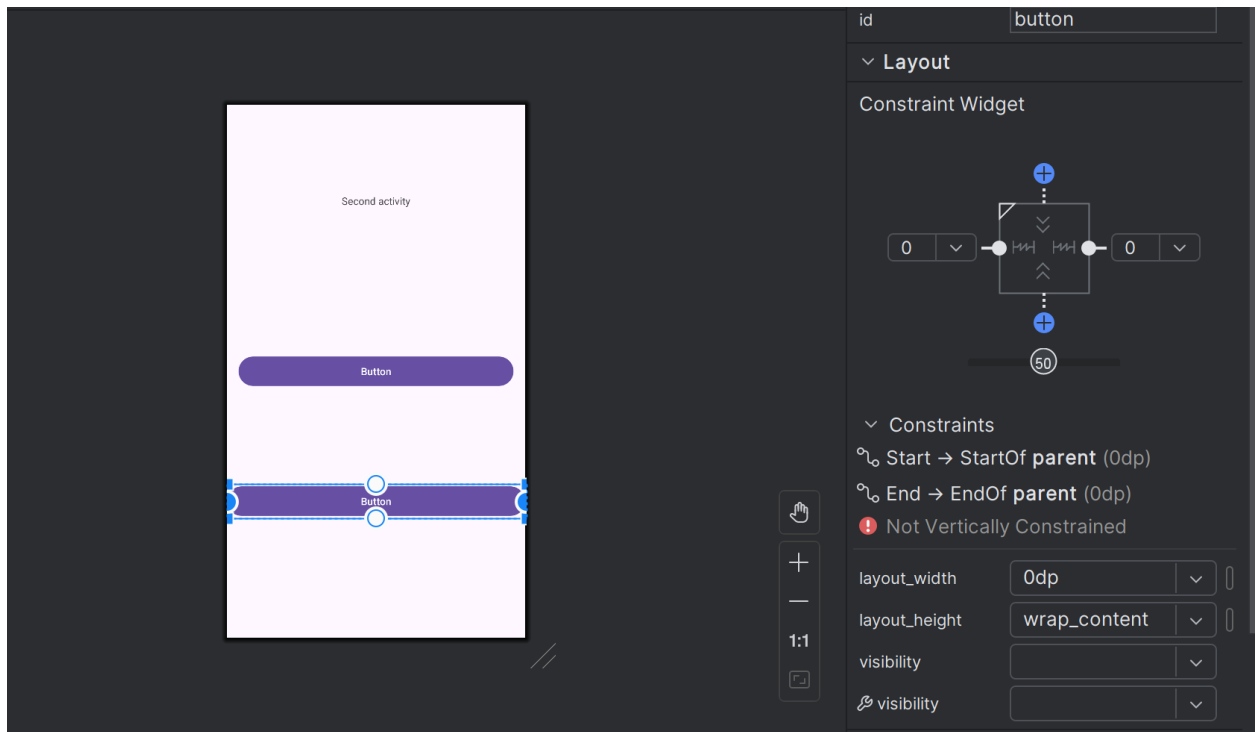
3. Nhấp vào **width control** hai lần—lần nhấp đầu tiên thay đổi thành **Fixed** với các đường thẳng, và lần nhấp thứ hai thay đổi thành **Match Constraints** với các lò xo, như minh họa trong hình động bên dưới.





Do thay đổi **width control**, thuộc tính `layout_width` trong **Attributes pane** hiển thị giá trị **match\_constraint**, và phần tử **Button** sẽ kéo giãn theo chiều ngang để lấp đầy khoảng trống giữa hai bên trái và phải của **layout**.

4. Chọn **Button** thứ hai và thực hiện các thay đổi tương tự cho `layout_width` như bước trước, như minh họa trong hình dưới đây.



Như đã thấy trong các bước trước, các thuộc tính **layout\_width** và **layout\_height** trong **Attributes pane** sẽ thay đổi khi bạn điều chỉnh các **height** và **width controls** trong **inspector**. Các thuộc tính này có thể nhận một trong ba giá trị cho **layout**, là **ConstraintLayout**:

- **match\_constraint**: Mở rộng phần tử **View** để lấp đầy **parent** theo chiều rộng hoặc chiều cao—tối đa đến **margin** nếu có. **Parent** trong trường hợp này là **ConstraintLayout**. Bạn sẽ tìm hiểu thêm về **ConstraintLayout** trong nhiệm vụ tiếp theo.
- **wrap\_content**: Thu nhỏ kích thước của phần tử **View** sao cho vừa đủ để chứa nội dung của nó. Nếu không có nội dung, phần tử **View** sẽ trở nên vô hình.
- Để chỉ định một kích thước cố định và điều chỉnh theo kích thước màn hình thiết bị, sử dụng một số cố định với đơn vị **density-independent pixels (dp)**. Ví dụ, **16dp** có nghĩa là 16 pixel độc lập với mật độ.

**Mẹo:** Nếu bạn thay đổi thuộc tính **layout\_width** bằng menu bật lên của nó, giá trị **layout\_width** sẽ được đặt thành **0** vì không có kích thước cố định nào được thiết lập. Cài đặt này tương đương với **match\_constraint**—phần tử **View** có thể mở rộng tối đa để đáp ứng các ràng buộc (**constraints**) và thiết lập lề (**margin**).

### 3.2 Thay đổi thuộc tính của Button

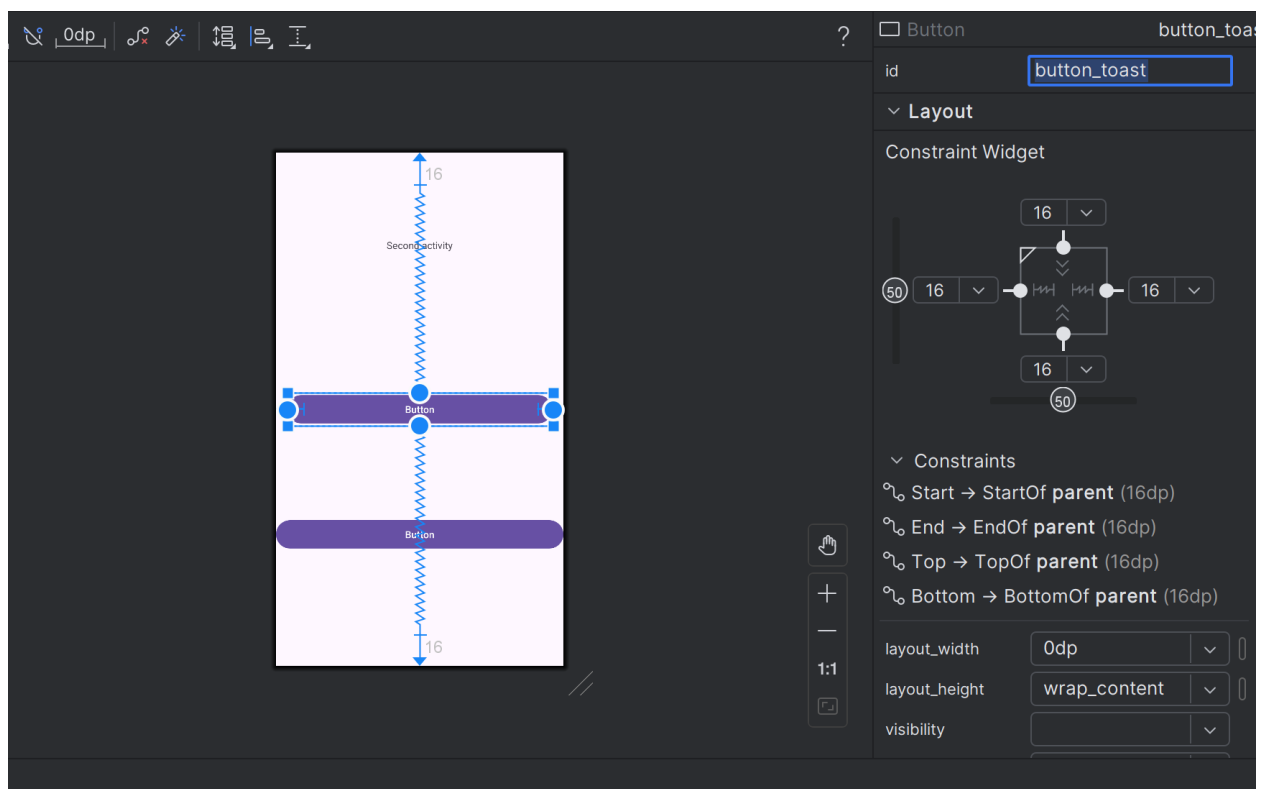
Để xác định từng **View** một cách duy nhất trong bố cục **Activity**, mỗi **View** hoặc lớp con của **View** (chẳng hạn như **Button**) cần có một **ID** duy nhất. Ngoài ra, các phần

từ **Button** cần có nội dung văn bản để có thể sử dụng được. Các phần tử **View** cũng có thể có nền là màu sắc hoặc hình ảnh.

**Bảng thuộc tính (Attributes pane)** cung cấp quyền truy cập vào tất cả các thuộc tính mà bạn có thể gán cho một phần tử **View**. Bạn có thể nhập giá trị cho từng thuộc tính như **android:id**, **background**, **textColor**, và **text**.

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

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



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

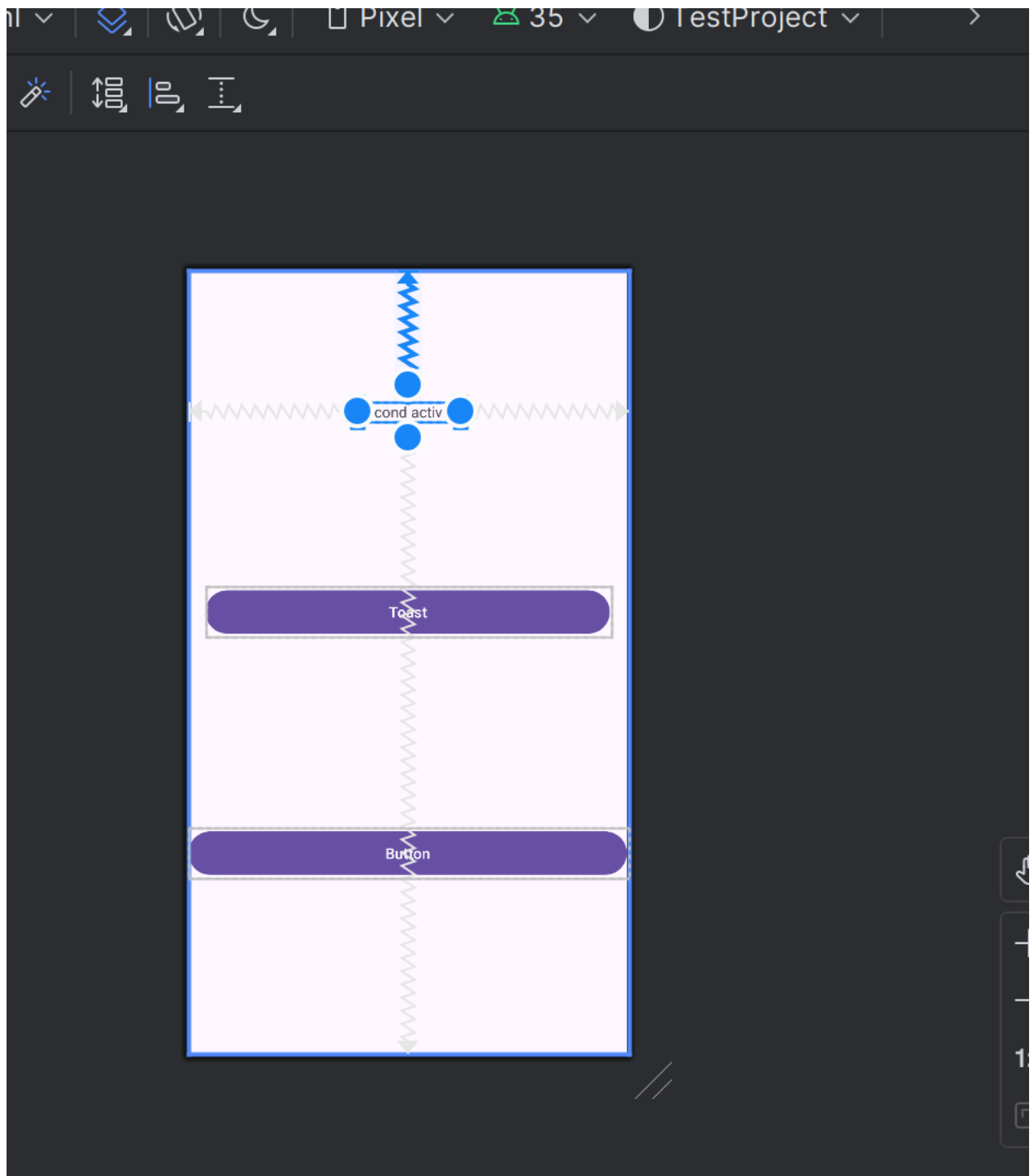
colorPrimary là màu chính của chủ đề, một trong các màu cơ bản được định nghĩa sẵn trong tệp tài nguyên **colors.xml**. Nó được sử dụng cho thanh ứng dụng (app bar). Việc sử dụng các màu cơ bản cho các thành phần giao diện người dùng khác giúp tạo ra một giao diện đồng nhất. Bạn sẽ tìm hiểu thêm về chủ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

## Task 4: Thêm một **TextEdit** và đặt thuộc tính của nó

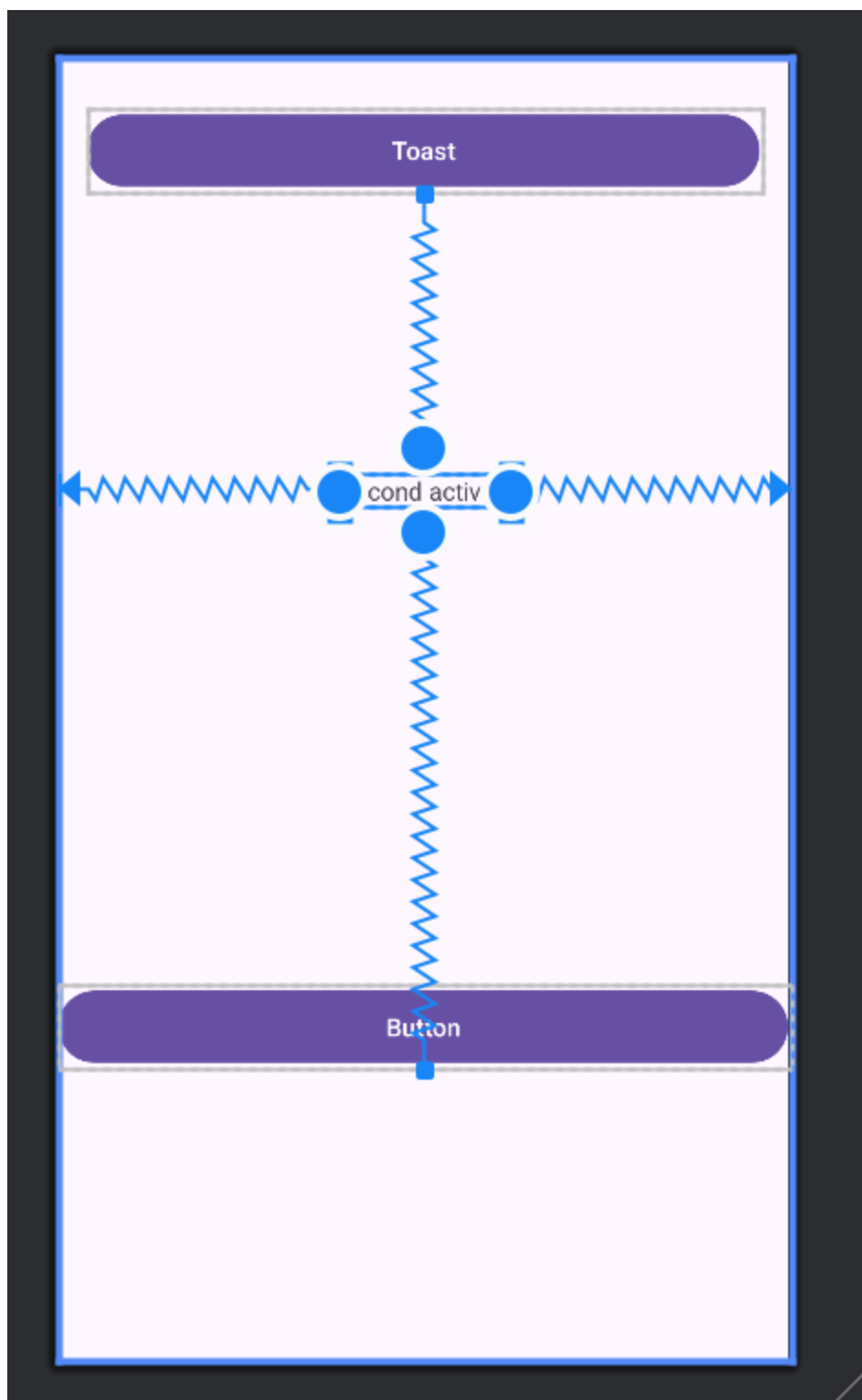
Một trong những lợi ích của **ConstraintLayout** là khả năng căn chỉnh hoặc ràng buộc các phần tử so với các phần tử khác. Trong nhiệm vụ này, bạn sẽ thêm một **TextView** vào giữa bố cục, ràng buộc nó theo chiều ngang với các lề và theo chiều dọc với hai phần tử **Button**. Sau đó, bạn sẽ thay đổi các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

### 4.1 Thêm một **TextView** và ràng buộc

1. Như hình minh họa bên dưới, kéo một **TextView** từ bảng **Palette** vào phần trên của bố cục, rồi kéo một ràng buộc từ cạnh trên của **TextView** đến tay cầm ở cạnh dưới của nút **Toast**. Điều này ràng buộc **TextView** nằm bên dưới **Button**.



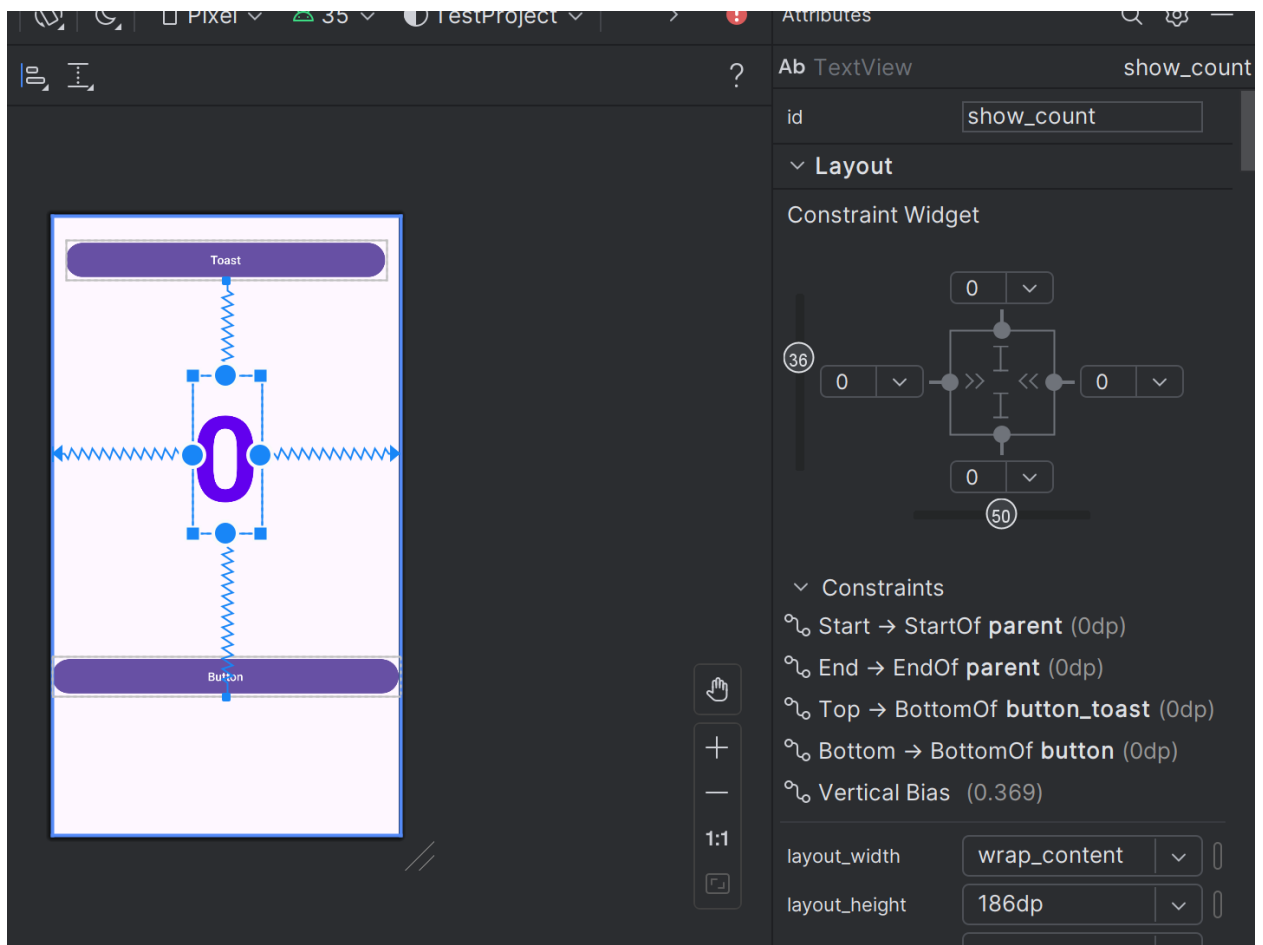
2. Như hình minh họa bên dưới, kéo một ràng buộc từ cạnh dưới của **TextView** đến tay cầm ở cạnh trên của nút **Count**, và từ hai cạnh của **TextView** đến hai cạnh của bố cục. Điều này ràng buộc **TextView** nằm ở giữa bố cục giữa hai nút **Button**



## 4.2 Đặt thuộc tính cho TextView

Với **TextView** được chọn, mở bảng **Attributes** nếu nó chưa được mở. Đặt các thuộc tính cho **TextView** như hình minh họa bên dưới. Các thuộc tính bạn chưa gặp sẽ được giải thích sau hình:

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



- **textSize:** Kích thước chữ của **TextView**. Trong bài học này, kích thước được đặt là **160sp**. **sp** là viết tắt của **scale-independent pixel** (pixel độc lập theo tỷ lệ), tương tự như **dp**, đây là đơn vị tự động điều chỉnh theo mật độ màn hình và cài đặt kích thước phông chữ của người dùng. Khi chỉ định kích thước phông chữ, hãy sử dụng đơn vị **sp** để đảm bảo kích thước được điều chỉnh phù hợp với cả mật độ màn hình và sở thích của người dùng.
- **textStyle** và **textAlignment:** Kiểu chữ (**text style**) được đặt thành **B** (đậm) trong bài học này, và căn chỉnh văn bản (**text alignment**) được đặt thành **ALIGNCENTER** (căn giữa đoạn văn bản).
- **gravity:** Thuộc tính **gravity** xác định cách một **View** được căn chỉnh bên trong **View** hoặc **ViewGroup** cha của nó. Trong bước này, bạn căn giữa

**TextView** theo chiều dọc bên trong **ConstraintLayout** cha.

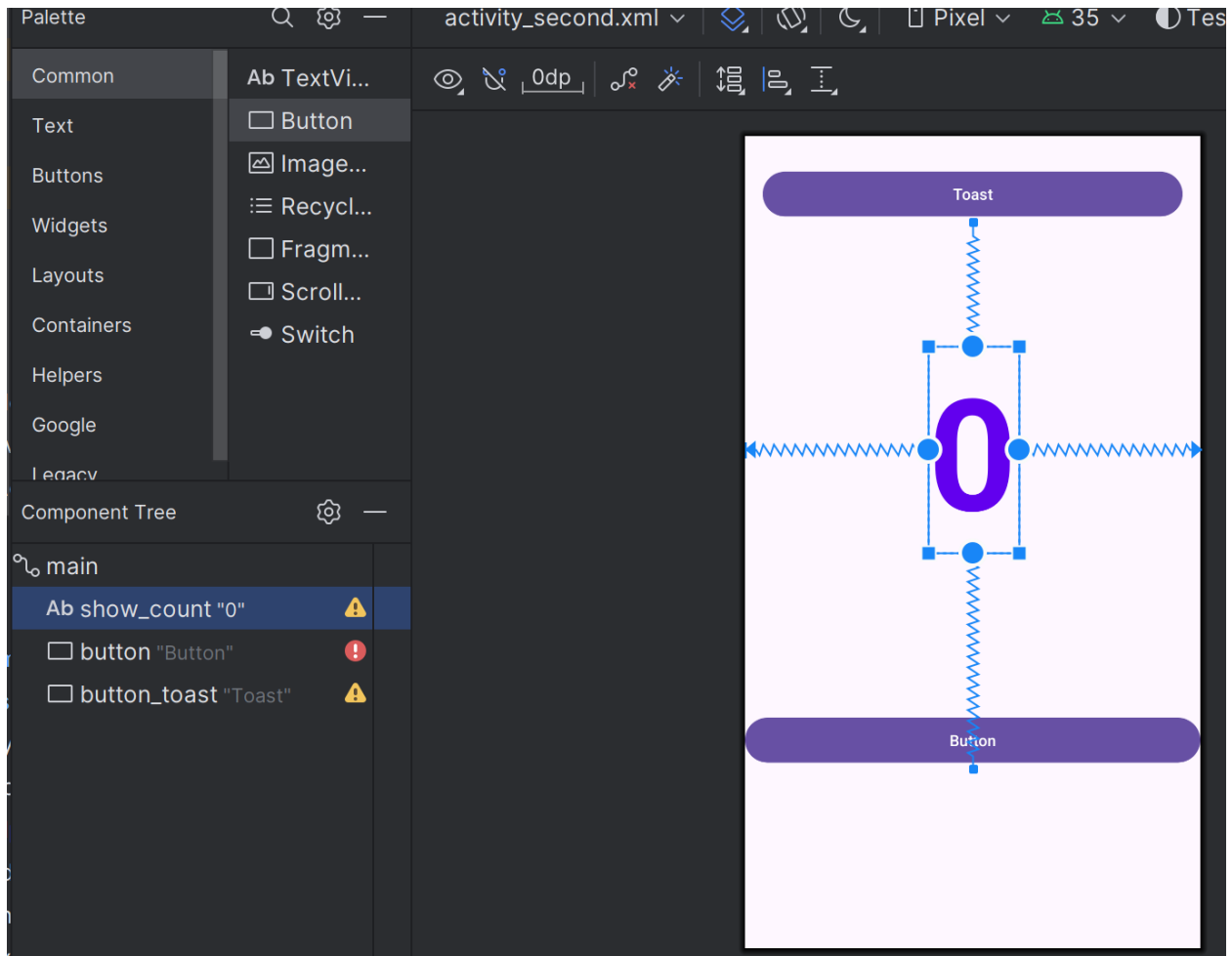
Bạn có thể nhận thấy rằng thuộc tính **background** xuất hiện trên trang đầu tiên của **Attributes** pane đối với **Button**, nhưng lại nằm trên trang thứ hai đối với **TextView**. **Attributes** pane thay đổi theo từng loại **View**: Các thuộc tính phổ biến nhất của loại **View** sẽ xuất hiện trên trang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

## Task 5: Chỉnh sửa layout trong XML

Bố cục của ứng dụng **Hello Toast** gần như đã hoàn thành! Tuy nhiên, một dấu chấm than xuất hiện bên cạnh mỗi phần tử giao diện người dùng trong **Component Tree**. Di chuột qua các dấu chấm than này để xem thông báo cảnh báo, như hình minh họa bên dưới.

Cảnh báo giống nhau xuất hiện cho cả ba phần tử: **các chuỗi được mã hóa cứng (hardcoded strings) nên sử dụng tài nguyên (resources)**.

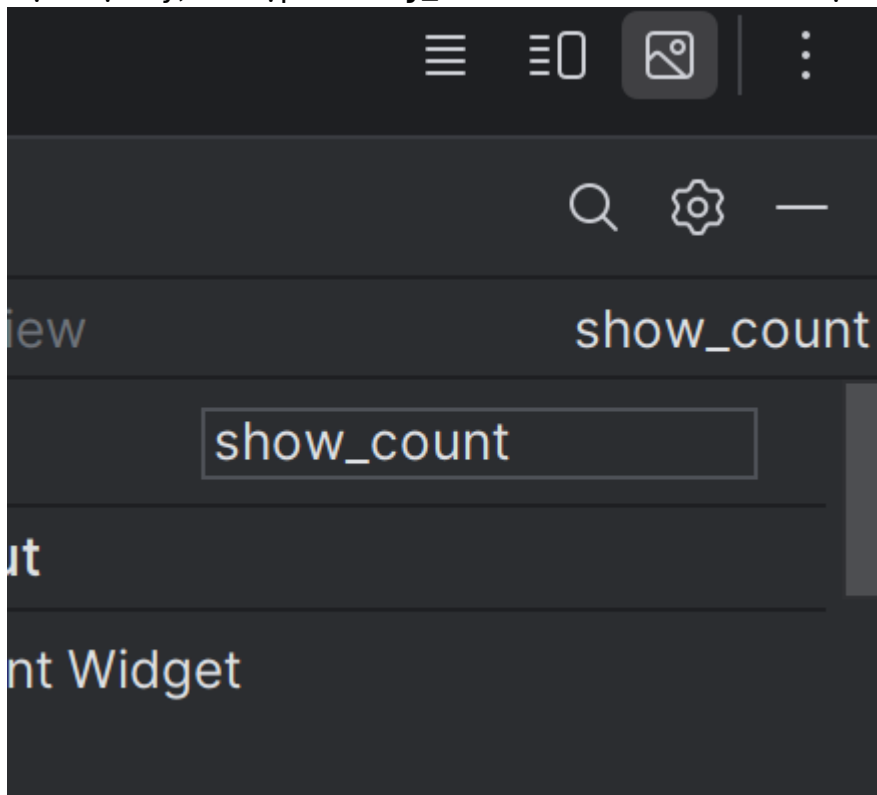




Cách dễ nhất để khắc phục các vấn đề về bố cục là chỉnh sửa bố cục trong XML. Mặc dù **layout editor** là một công cụ mạnh mẽ, nhưng một số thay đổi sẽ dễ thực hiện hơn khi chỉnh sửa trực tiếp trong mã nguồn XML.

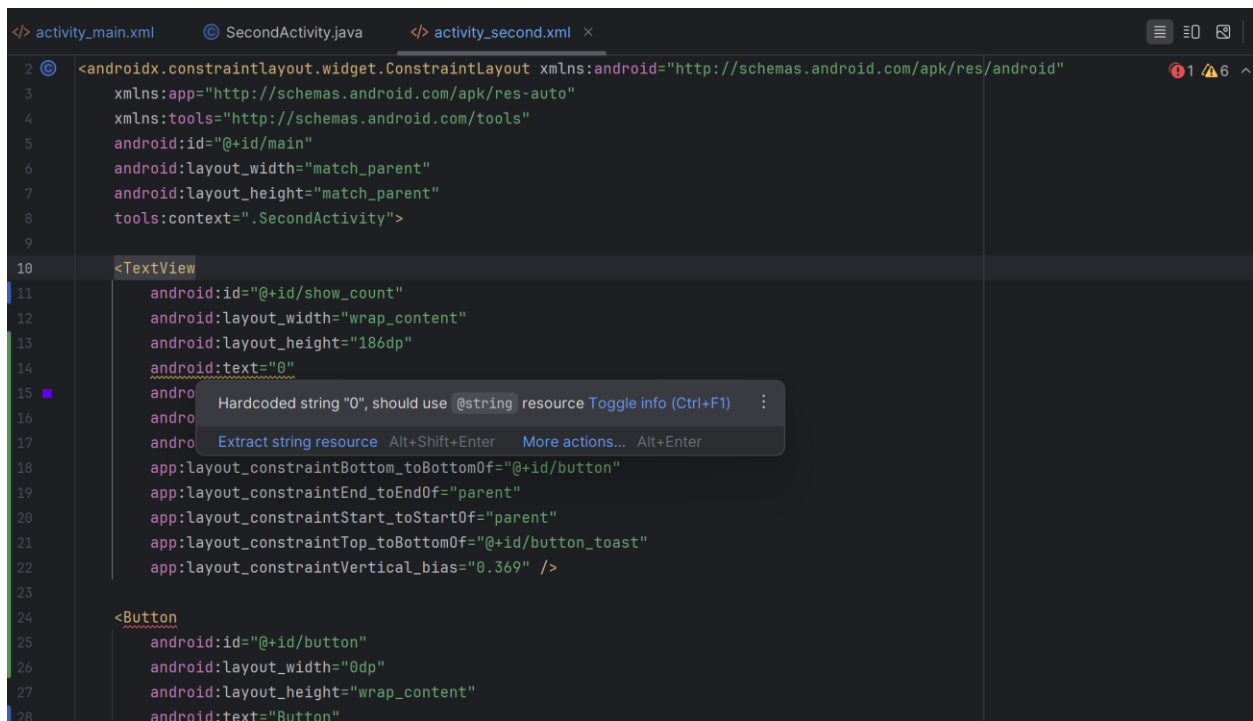
## 5.1 Mở mã XML cho bố cục

Trong nhiệm vụ này, mở tệp **activity\_main.xml** nếu nó chưa được mở, sau đó nhấp



vào tab **ở phía dưới**  
**của trình chỉnh sửa bố cục.**

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



```
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:id="@+id/main"
6   android:layout_width="match_parent"
7   android:layout_height="match_parent"
8   tools:context=".SecondActivity">
9
10  <TextView
11      android:id="@+id/show_count"
12      android:layout_width="wrap_content"
13      android:layout_height="186dp"
14      android:text="0"
15      android:layout_constraintBottom_toBottomOf="@+id/button"
16      android:layout_constraintEnd_toEndOf="parent"
17      android:layout_constraintStart_toStartOf="parent"
18      android:layout_constraintTop_toBottomOf="@+id/button_toast"
19      android:layout_constraintVertical_bias="0.369" />
20
21  <Button
22      android:id="@+id/button"
23      android:layout_width="80dp"
24      android:layout_height="wrap_content"
25      android:text="Button"
```

## 5.2 Trích xuất tài nguyên chuỗi

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

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

```
<resources>
  ⚡ <string name="app_name">Test Project</string>
</resources>
```

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

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

## Task 6: Thêm trình xử lý onClick cho các Button

Trong bước này, bạn sẽ viết các phương thức trong **MainActivity.java** để xử lý sự kiện khi người dùng nhấn vào các Button.

### 6.1 Thêm thuộc tính onClick và xử lý sự kiện cho mỗi Button

Một **click handler** là một phương thức được gọi khi người dùng nhấp chuột 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** trong bảng **Attributes** của tab **Design**. Bạn cũng có thể chỉ định tên của phương thức handler trong trình chỉnh sửa XML bằng cách thêm thuộc tính **android:onClick** vào **Button**. Bạn sẽ sử dụng phương pháp sau vì bạn chưa tạo các phương thức handler, và trình chỉnh sửa XML cung cấp một cách tự động để tạo những phương thức đó.

1. Với trình chỉnh sửa XML mở (tab **Text**), tìm **Button** có **android:id** được thiết lập là button\_label\_toast:

```
<Button
    android:id="@+id/button_label_toast"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Toast"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    tools:layout_editor_absoluteY="520dp" />
```

2. Thêm thuộc tính **android:onClick** vào cuối phần tử **button\_toast**, sau thuộc tính cuối cùng và trước chỉ thị kết thúc **/>**

```
android:onClick="showToast"
```

3. Nhấp 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, nhấp vào tên phương thức ("showToast"). Nhấn **Alt-Enter** (hoặc **Option-Enter** trên Mac), chọn **Create 'showToast(view)' in MainActivity**, và nhấp **OK**.

Hành động này tạo một phương thức placeholder (stub) cho phương thức **showToast()** trong **MainActivity**, như được hiển thị ở cuối các bước này.

4. Lặp lại hai bước cuối cùng với nút **button\_count**: Thêm thuộc tính **android:onClick** vào cuối, và thêm handler cho sự kiện click

```
app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.019"
android:onClick="countUp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Mã XML cho các phần tử giao diện người dùng trong ConstraintLayout hiện tại trông như thế này

```
<Button
    android:id="@+id/button_label_toast"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Toast"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    tools:layout_editor_absoluteY="520dp"
    android:onClick="showToast" />
```

```
<TextView
    android:id="@+id/count_initial_value"
    android:layout_width="wrap_content"
    android:layout_height="186dp"
    android:text="0"
    android:textColor="@color/design_default_color_primary"
    android:textSize="140sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="@+id/button_label_toast"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button_label_count"
    app:layout_constraintVertical_bias="0.369" />
```

```
<Button
```

```
    android:id="@+id/button_label_count"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="16dp"  
    android:layout_marginTop="16dp"  
    android:layout_marginEnd="16dp"  
    android:layout_marginBottom="16dp"  
    android:text="Count"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintHorizontal_bias="1.0"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    app:layout_constraintVertical_bias="0.019"  
    android:onClick="countUp"/>
```

5. Nếu tệp MainActivity.java chưa được mở, mở rộng mục java trong chế độ xem Project > Android, sau đó mở rộng com.example.android.hellotoast, và cuối cùng nhấp đúp vào MainActivity. Trình chỉnh sửa mã sẽ xuất hiện với mã trong MainActivity.

```

> public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_second);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }

    1 usage
    public void showToast(View view) {
    }

    1 usage
    public void countUp(View view) {
    }
}

```

## 6.2 Chỉnh sửa trình xử lý nút Toast

Bây giờ, bạn sẽ chỉnh sửa phương thức `showToast()`—trình xử lý sự kiện nhấn nút Toast trong `MainActivity`—để nó hiển thị một thông báo. Một Toast cung cấp cách thức để hiển thị một thông báo đơn giản trong một cửa sổ pop-up nhỏ. Nó chỉ chiếm không gian cần thiết cho thông điệp. Hoạt động hiện tại vẫn hiển thị và có thể tương tác. Một Toast có thể hữu ích để kiểm tra tính tương tác trong ứng dụng của bạn—thêm một thông báo Toast để hiển thị kết quả khi nhấn nút hoặc thực hiện một hành động.

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

1. Tìm phương thức `showToast()` mới được tạo.

```

1 usage
public void showToast(View view) {
}

```



2. Để tạo một thể hiện của Toast, gọi phương thức chế tạo `makeText()` trên lớp `Toast`

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

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

```
1 usage Context  
public void showToast(View view) {  
    Toast toast = Toast.makeText(this,);  
}
```

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

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

5. Cung cấp thời gian hiển thị. Ví dụ, `Toast.LENGTH_SHORT` hiển thị toast trong một khoảng thời gian khá ngắn

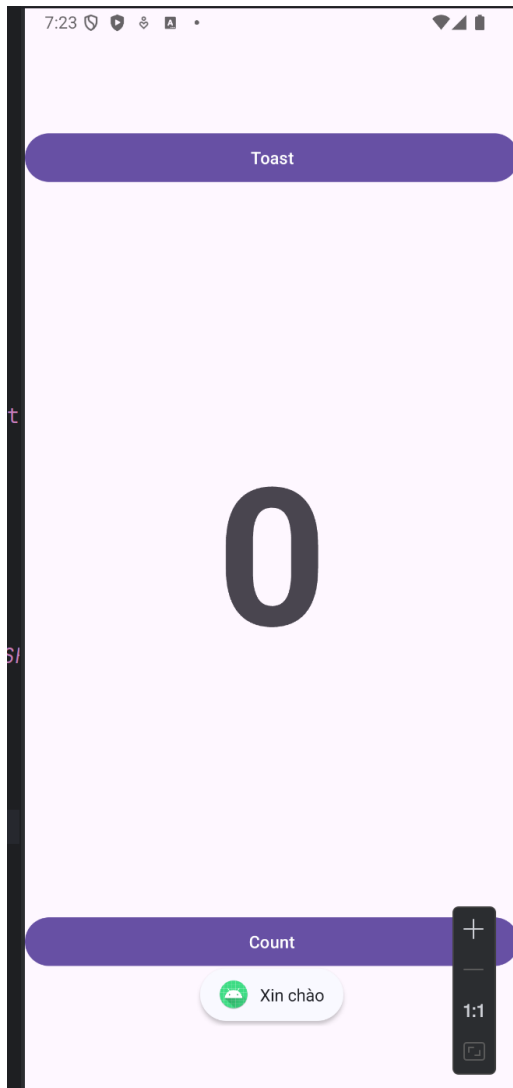
```
public void showToast(View view) {  
    Toast toast = Toast.makeText(context: this, R.string.toast_message, Toast.LENGTH_SHORT);  
}
```

Thời gian hiển thị của một Toast có thể là `Toast.LENGTH_LONG` hoặc `Toast.LENGTH_SHORT`. Thời gian thực tế lần lượt là khoảng 3.5 giây cho Toast dài và 2 giây cho Toast ngắn.

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

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

Chạy ứng dụng và xác nhận rằng thông báo Toast xuất hiện khi nút Toast được nhấn.



### 6.3 Xử lý hàm Count Button

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

Yêu cầu của trình xử lý sự kiện bao gồm:

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

Thực hiện các bước sau để chỉnh sửa trình xử lý sự kiện khi nhấn nút "Count":

1. Xác định vị trí của phương thức `countUp()` vừa được tạo.

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

2. Để theo dõi số đếm, bạn cần một biến thành viên riêng (**private**). Mỗi lần nhấn nút "Count" sẽ tăng giá trị của biến này. Hãy nhập đoạn mã sau, phần này sẽ được đánh dấu màu đỏ và hiển thị biểu tượng bóng đèn đỏ:

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

Nếu biểu tượng bóng đèn đỏ không xuất hiện, hãy chọn biểu thức `mCount++`. Biểu tượng bóng đèn đỏ cuối cùng sẽ xuất hiện.

3. Nhấp vào biểu tượng bóng đèn đỏ và chọn "Create field 'mCount'" từ menu xuất hiện. Điều này sẽ tạo một biến thành viên riêng ở đầu `MainActivity`, và Android Studio sẽ tự động giả định rằng bạn muốn nó là kiểu số nguyên (`int`):
4. Thay đổi câu lệnh khai báo biến thành viên riêng (`private`) để khởi tạo biến với giá trị bằng 0:

```
public class MainActivity extends AppCompatActivity {  
    private int mCount = 0;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {
```

5. Cùng với biến trên, bạn cũng cần một biến thành viên (private) để tham chiếu đến TextView show\_count, biến này sẽ được sử dụng trong trình xử lý sự kiện khi nhấn nút. Đặt tên cho biến này là mShowCount.

```
public class MainActivity extends AppCompatActivity {  
    1 usage  
    private int mCount = 0;  
    no usages  
    private TextView mShowCount;  
    @Override
```

6. Bây giờ bạn đã có **mShowCount**, bạn có thể lấy tham chiếu đến **TextView** bằng ID mà bạn đã đặt trong tệp layout. Để lấy tham chiếu này chỉ một lần, hãy khai báo nó trong phương thức **onCreate()**. Như bạn sẽ học trong bài học khác, phương thức **onCreate()** được sử dụng để "inflate" layout, có nghĩa là thiết lập giao diện màn hình dựa trên tệp XML. Bạn cũng có thể sử dụng nó để lấy tham chiếu đến các phần tử giao diện người dùng khác trong layout, chẳng hạn như **TextView**. Xác định vị trí phương thức **onCreate()** trong **MainActivity**:

```
@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;  
    });  
}
```

7. Thêm câu lệnh findViewById vào cuối phương thức:

```
@SuppressWarnings("MissingInflatedId")  
@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;  
    });  
    mShowCount = (TextView) findViewById(R.id.txt_showCount);  
}
```

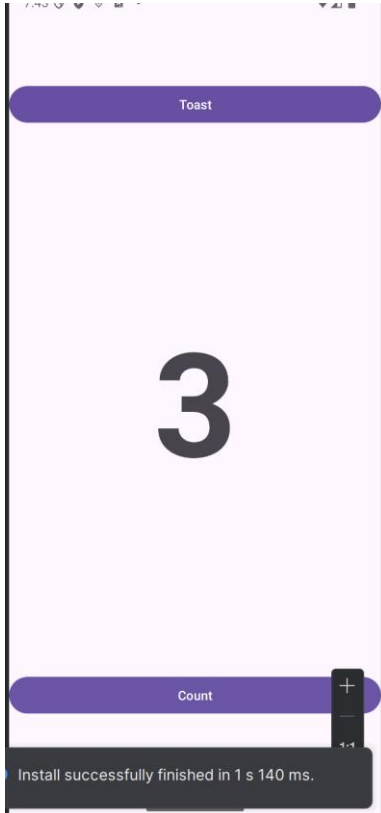
8. Bây giờ bạn đã gán TextView cho biến mShowCount, bạn có thể sử dụng biến này để đặt văn bản trong TextView thành giá trị của biến mCount. Thêm đoạn mã sau vào phương thức countUp():

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

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

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

9. Chạy ứng dụng để kiểm tra rằng số đếm tăng lên khi bạn nhấn nút "Count":

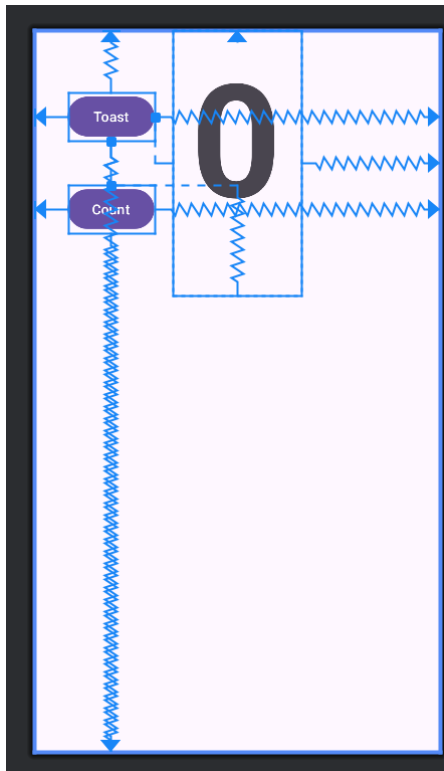


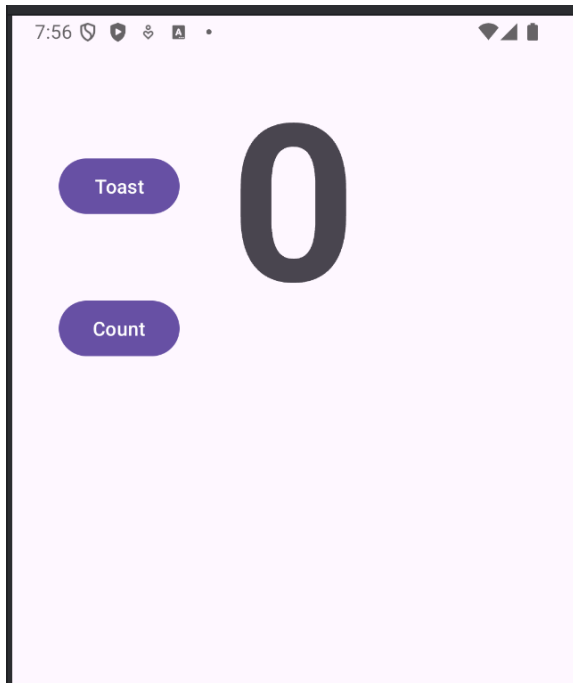
## Mã giải pháp Dự án Android Studio: HelloToast

Ứng dụng HelloToast hiển thị tốt khi thiết bị hoặc trình giả lập được đặt theo chiều dọc. Tuy nhiên, nếu bạn xoay thiết bị hoặc trình giả lập sang chiều ngang, nút "Count" có thể chồng lên TextView ở phía dưới, như được hiển thị trong hình bên dưới.

**Thử thách:** Thay đổi bố cục để ứng dụng hiển thị đẹp ở cả chế độ ngang và dọc

1. Trên máy tính của bạn, tạo một bản sao của thư mục dự án **HelloToast** và đổi tên thành **HelloToastChallenge**.
2. Mở **HelloToastChallenge** trong Android Studio và thực hiện refactor. (Xem **Phụ lục: Tiện ích** để biết hướng dẫn về cách sao chép và refactor một dự án.)
3. Thay đổi bố cục sao cho nút **Toast** và **Count** xuất hiện ở phía bên trái, như hình minh họa. **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 và kiểm tra trong cả hai chế độ ngang và dọc





## Tóm tắt

View, ViewGroup và layouts:

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

Sử dụng trình chỉnh sửa bố cục (Layout Editor):

- Nhấn vào tab **Design** để thao tác với 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) mà bạn có thể sử dụng trong bố cục ứng dụng, còn bảng **Component tree** hiển thị cây phân cấp của các phần tử UI.
- Các bảng thiết kế (**design**) và bản vẽ (**blueprint**) trong trình chỉnh sửa bố cục hiển thị các phần tử UI trong bố cục.
- Tab **Attributes** hiển thị bảng thuộc tính (**Attributes pane**) để thiết lập các thuộc tính cho một phần tử UI.
- **Các thao tác với bố cục:**
  - **Tay nắm ràng buộc (Constraint handle):** Nhấn vào tay nắm ràng buộc, được hiển thị dưới dạng vòng tròn ở mỗi cạnh của một phần tử, sau đó kéo đến một tay nắm ràng buộc khác hoặc đến ranh giới cha để tạo ràng buộc. Ràng buộc sẽ được biểu diễn bằng một đường gấp khúc.
  - **Tay nắm thay đổi kích thước (Resizing handle):** Bạn có thể kéo các tay nắm vuông để thay đổi kích thước của phần tử. Khi kéo, tay nắm sẽ thay đổi thành một góc xiên.
  - **Công cụ Autoconnect:** Khi được bật, công cụ này 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, nó sẽ tạo các ràng buộc dựa trên vị trí của phần tử.
  - **Xóa ràng buộc:** Bạn có thể xóa ràng buộc của một phần tử bằng cách chọn phần tử đó và di chuột qua để hiển thị nút **Clear Constraints**. Nhấn vào nút này để xóa tất cả ràng buộc của phần tử đã chọn. Để xóa một ràng buộc cụ thể, nhấn vào tay nắm đặt ràng buộc đó.
  - **Bảng thuộc tính (Attributes pane):** Cung cấp quyền truy cập vào tất cả các thuộc tính XML mà bạn có thể gán cho một phần tử UI. Nó cũng bao gồm một bảng kích thước hình vuông có tên là **view inspector** ở trên cùng. Các biểu tượng bên trong hình vuông này đại diện cho các cài đặt chiều cao và chiều rộng.

Chỉnh sửa chiều rộng và chiều cao:



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

- **match\_constraint**: Mở rộng phần tử để lấp đầy không gian của phần tử cha theo chiều rộng hoặc chiều cao, giới hạn trong phần lề (nếu có thiết lập).
- **wrap\_content**: Thu nhỏ kích thước phần tử sao cho nó chỉ vừa đủ chứa nội dung bên trong. Nếu không có nội dung, phần tử sẽ trở nên vô hình.
- **Kích thước cố định (dp - density-independent pixels)**: Xác định một kích thước cố định bằng đơn vị dp, giúp điều chỉnh phù hợp với kích thước màn hình của thiết bị.

Trích xuất tài nguyên chuỗi (String Resources):

Thay vì mã hóa cứng (hard-code) các chuỗi văn bản, bạn nên sử dụng tài nguyên chuỗi (string resources), giúp quản lý chuỗi tốt hơn. Thực hiện theo các bước sau:

1. Nhấp vào chuỗi văn bản được mã hóa cứng mà bạn muốn trích xuất, nhấn **Alt + Enter** (**Option + Enter** trên Mac), sau đó chọn **Extract string resources** từ menu xuất hiện.
2. Đặt tên cho tài nguyên (**Resource name**).
3. Nhấn **OK**. Hệ thống sẽ tạo một tài nguyên chuỗi trong tệp `values/res/string.xml`, và chuỗi trong mã của bạn sẽ được thay thế bằng một tham chiếu đến tài nguyên: `@string/button_label_toast`.

Xử lý sự kiện nhấn:

- **Trình xử lý nhấn (click handler)** là một phương thức được gọi khi người dùng nhấp hoặc chạm vào một phần tử UI.
- Chỉ định một **click handler** cho một phần tử UI như **Button** bằng cách nhập tên phương thức vào trường **onClick** trong bảng **Attributes** của tab **Design**, hoặc trong trình chỉnh sửa XML bằng cách thêm thuộc tính `android:onClick` vào phần tử UI như **Button**.
- Tạo trình xử lý nhấn trong **Activity chính** bằng cách sử dụng tham số **View**. Ví dụ:

- Bạn có thể tìm thấy thông tin về tất cả các thuộc tính của **Button** trong **Button class documentation**, và tất cả các thuộc tính của **TextView** trong **TextView class documentation**.

Hiển thị tin nhắn Toast:

**Toast** cung cấp một cách để hiển thị một thông báo đơn giản trong một cửa sổ popup nhỏ. Nó chỉ chiếm đúng lượng không gian cần thiết cho thông báo. Để tạo một **Toast**, thực hiện các bước sau:

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

## Trình chỉnh sửa bố cục

### 1.3) Văn bản và các chế độ cuộn

### 1.4) Tài nguyên có sẵn

## Bài 2) Activities

### 2.1) Activity và Intent

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

### 2.3) Intent ngầm định

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

### 3.1) Trình gỡ lỗi

### 3.2) Kiểm thử đơn vị

### 3.3) Thư viện hỗ trợ

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

### **Bài 1) Tương tác người dùng**

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

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

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

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

- 3.1) Espresso cho việc kiểm tra UI

## **CHƯƠNG 3. LÀM VIỆC TRONG NỀN**

### **Bài 1) Các tác vụ nền**

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

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

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

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

### **Bài 1) Tùy chọn và cài đặt**

#### **1.1) Shared preferences**

#### **1.2) Cài đặt ứng dụng**

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

#### **2.1) Room, LiveData và ViewModel**

#### **2.2) Room, LiveData và ViewModel**