**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**GIÁO TRÌNH**

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

Hà Nội, 2.2025

MỤC LỤC

[CHƯƠNG 1. Làm quen 4](#_gjdgxs)

[Bài 1) Tạo ứng dụng đầu tiên 4](#_30j0zll)

[1.1) Android Studio và Hello World 4](#_1fob9te)

[1.2) Giao diện người dùng tương tác đầu tiên 5](#_3znysh7)

[1.3) Trình chỉnh sửa bố cục 5](#_2et92p0)

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

[1.5) Tài nguyên có sẵn 5](#_3dy6vkm)

[Bài 2) Activities 5](#_1t3h5sf)

[2.1) Activity và Intent 5](#_4d34og8)

[2.2) Vòng đời của Activity và trạng thái 5](#_2s8eyo1)

[2.3) Intent ngầm định 5](#_17dp8vu)

[Bài 3) Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ 5](#_3rdcrjn)

[3.1) Trình gỡ lỗi 5](#_26in1rg)

[3.2) Kiểm thử đơn vị 5](#_lnxbz9)

[3.3) Thư viện hỗ trợ 5](#_35nkun2)

[CHƯƠNG 2. Trải nghiệm người dùng 6](#_1ksv4uv)

[Bài 1) Tương tác người dùng 6](#_44sinio)

[1.1) Hình ảnh có thể chọn 6](#_2jxsxqh)

[1.2) Các điều khiển nhập liệu 6](#_z337ya)

[1.3) Menu và bộ chọn 6](#_3j2qqm3)

[1.4) Điều hướng người dùng 6](#_1y810tw)

[1.5) RecycleView 6](#_4i7ojhp)

[Bài 2) Trải nghiệm người dùng thú vị 6](#_2xcytpi)

[2.1) Hình vẽ, định kiểu và chủ đề 6](#_1ci93xb)

[2.2) Thẻ và màu sắc 6](#_3whwml4)

[2.3) Bố cục thích ứng 6](#_2bn6wsx)

[Bài 3) Kiểm thử giao diện người dùng 6](#_qsh70q)

[3.1) Espresso cho việc kiểm tra UI 6](#_3as4poj)

[CHƯƠNG 3. Làm việc trong nền 6](#_1pxezwc)

[Bài 1) Các tác vụ nền 6](#_49x2ik5)

[1.1) AsyncTask 6](#_2p2csry)

[1.2) AsyncTask và AsyncTaskLoader 6](#_147n2zr)

[1.3) Broadcast receivers 6](#_3o7alnk)

[Bài 2) Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền 6](#_23ckvvd)

[2.1) Thông báo 6](#_ihv636)

[2.2) Trình quản lý cảnh báo 6](#_32hioqz)

[2.3) JobScheduler 6](#_1hmsyys)

[CHƯƠNG 4. Lưu dữ liệu người dùng 7](#_41mghml)

[Bài 1) Tùy chọn và cài đặt 7](#_2grqrue)

[1.1) Shared preferences 7](#_vx1227)

[1.2) Cài đặt ứng dụng 7](#_3fwokq0)

[Bài 2) Lưu trữ dữ liệu với Room 7](#_1v1yuxt)

[2.1) Room, LiveData và ViewModel 7](#_4f1mdlm)

[2.2) Room, LiveData và ViewModel 7](#_2u6wntf)

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

# LÀM QUEN

## Tạo ứng dụng đầu tiên

### Android Studio và Hello World

Giới thiệu

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

Những gì Bạn nên biết

Bạn nên có khả năng:

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

A screenshot of a computer

AI-generated content may be incorrect.

Những gì Bạn sẽ cần:

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

Những gì bạn sẽ học

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

Những gì bạn sẽ làm

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

### Giao diện người dùng tương tác đầu tiên

### Trình chỉnh sửa bố cục

### Văn bản và các chế độ cuộn

### Tài nguyên có sẵn

## Activities

### Activity và Intent

### Vòng đời của Activity và trạng thái

### Intent ngầm định

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

### Trình gỡ lỗi

### Kiểm thử đơn vị

### Thư viện hỗ trợ

# TRẢI NGHIỆM NGƯỜI DÙNG

## Tương tác người dùng

### Hình ảnh có thể chọn

### Các điều khiển nhập liệu

### Menu và bộ chọn

### Điều hướng người dùng

### RecycleView

## Trải nghiệm người dùng thú vị

### Hình vẽ, định kiểu và chủ đề

### Thẻ và màu sắc

### Bố cục thích ứng

## Kiểm thử giao diện người dùng

### Espresso cho việc kiểm tra UI

# LÀM VIỆC TRONG NỀN

## Các tác vụ nền

### AsyncTask

### Giới thiệu Một luồng (thread) là một đường thực thi độc lập trong một chương trình đang chạy. Khi một chương trình Android được khởi chạy, hệ thống sẽ tạo ra một luồng chính, còn được gọi là UI thread. Luồng UI này là cách ứng dụng của bạn tương tác với các thành phần từ bộ công cụ giao diện người dùng của Android.

### Đôi khi, một ứng dụng cần thực hiện các tác vụ nặng như tải xuống tệp, truy vấn cơ sở dữ liệu, phát phương tiện hoặc tính toán phân tích phức tạp. Những công việc này có thể làm tắc nghẽn luồng UI, khiến ứng dụng không phản hồi với thao tác của người dùng hoặc không vẽ được giao diện trên màn hình. Người dùng có thể cảm thấy khó chịu và gỡ cài đặt ứng dụng của bạn.

### Để duy trì trải nghiệm người dùng (UX) mượt mà, Android cung cấp một lớp hỗ trợ có tên AsyncTask, giúp xử lý công việc bên ngoài luồng UI. Việc sử dụng AsyncTask để thực hiện các tác vụ nặng trên một luồng riêng biệt giúp UI thread luôn phản hồi nhanh.

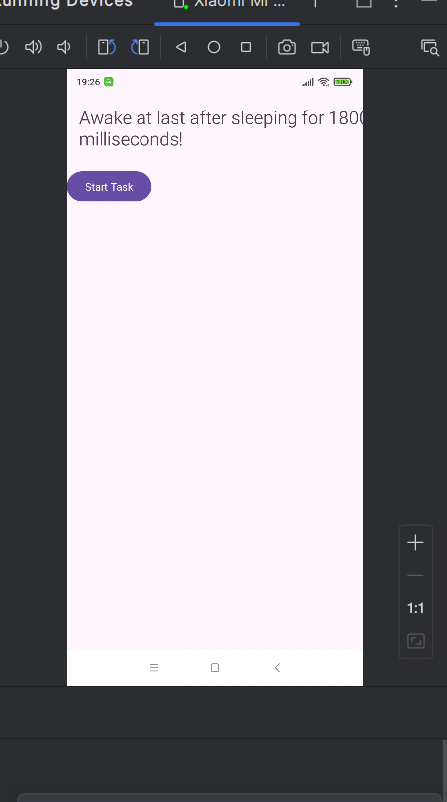
### Vì luồng riêng biệt này không đồng bộ với luồng gọi ban đầu, nó được gọi là luồng bất đồng bộ (asynchronous thread). AsyncTask cũng chứa một phương thức callback, cho phép bạn hiển thị kết quả của quá trình xử lý trở lại trong luồng UI.

### Trong bài thực hành này, bạn sẽ học cách thêm một tác vụ chạy nền vào ứng dụng Android của mình bằng cách sử dụng AsyncTask.

### Tóm tắt ứng dụng

### Bạn sẽ xây dựng một ứng dụng có một TextView và một Button. Khi người dùng nhấn vào Button, ứng dụng sẽ "ngủ" trong một khoảng thời gian ngẫu nhiên, sau đó hiển thị một thông báo trong TextView khi nó "thức dậy".

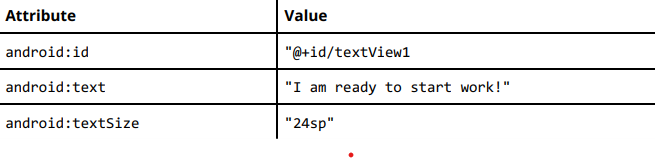
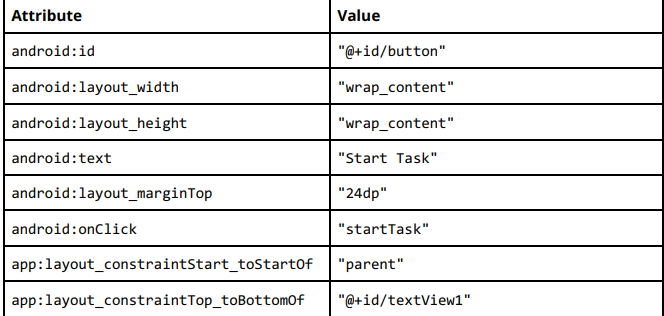
### Đây là giao diện của ứng dụng khi hoàn thành:



### Nhiệm vụ 1 : Cài đặt SimpleAsyncTask

* Tạo 1 dự án sử dụng Empty Activity . Chấp thuận những cài đặt mặc định
* Mở activity\_main.xml . Nhấn vào nhãn Text
* Thêm layout\_margin vào đầu ConstraintLayout



* Thêm hoặc chỉnh sửa các thuộc tính của TextView "Hello World!" với các giá trị sau
* 
* Xóa app:layout\_constraintRight\_toRightOf và app:layout\_constraintTop\_toTopOf
* Thêm một phần tử Button ngay dưới TextView và gán các thuộc tính sau  
  
* Thuộc tính onClick của Button sẽ được tô vàng do phương thức startTask() chưa được triển khai trong MainActivity. Đặt con trỏ vào vị trí được đánh dấu nhấn Alt + Enter (Option + Enter on a Mac) hoặc chọn tạo 'startTask(View) trong 'MainActivity' để tạo 1 phương thức trống trong MainActivity

**Nhiệm vụ 2 : Tạo lớp con AsyncTask**

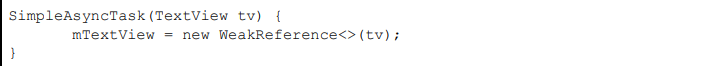
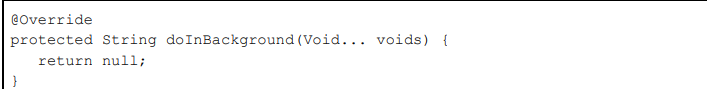
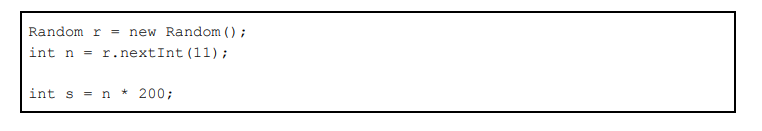
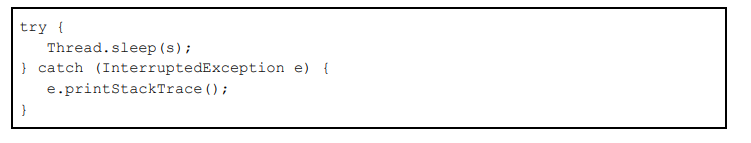
* AsyncTask là một lớp trừu tượng, có nghĩa là bạn phải kế thừa (subclass) nó để sử dụng.Trong ví dụ này, AsyncTask thực hiện một tác vụ nền rất đơn giản: ngủ trong một khoảng thời gian ngẫu nhiên.Trong một ứng dụng thực tế, tác vụ nền có thể thực hiện nhiều công việc khác nhau, chẳng hạn như truy vấn cơ sở dữ liệu, kết nối với Internet, tính toán nước đi tiếp theo trong cờ vây để đánh bại nhà vô địch hiện tại
* Khi bạn tạo một lớp con của AsyncTask, bạn có thể cần cung cấp thông tin về công việc mà nó sẽ thực hiện, cách báo cáo tiến trình (nếu có), và định dạng kết quả trả về.

Khi tạo một lớp con của AsyncTask, bạn có thể cấu hình nó bằng các tham số sau:

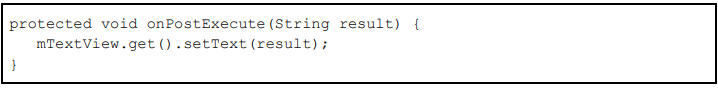
* Params: Kiểu dữ liệu của các tham số được gửi đến tác vụ khi thực thi phương thức doInBackground().
* Progress: Kiểu dữ liệu của các đơn vị tiến trình được cập nhật bằng phương thức onProgressUpdate().
* Result: Kiểu dữ liệu của kết quả được trả về bởi phương thức onPostExecute().
* Trong ứng dụng này, lớp con AsyncTask mà bạn tạo không yêu cầu tham số truy vấn hoặc cập nhật tiến trình. Bạn chỉ cần sử dụng hai phương thức: doInBackground() và onPostExecute().

Tạo một lớp Java mới có tên SimpleAsyncTask nhận ba tham số kiểu (generic type parameters).



* Ở đầu lớp, khai báo một biến thành viên có tên mTextView với kiểu dữ liệu WeakReference<TextView>.
* 
* Triển khai một constructor cho AsyncTask nhận một TextView làm tham số và Tạo một WeakReference mới cho TextView để tránh rò rỉ bộ nhớ .
* 
* Phương thức doInBackground() là bắt buộc đối với lớp con AsyncTask của bạn. Đặt con trỏ vào phần đánh dấu , nhấn Alt + Enter (Option + Enter on a Mac) và chọn Implement methods. Chọn doInBackground() và nhấn OK. Tiếp theo mẫu phương thức sẽ được thêm vào lớp của bạn  
  
* Thêm đoạn mã để tạo một số nguyên ngẫu nhiên từ 0 đến 10.Đây sẽ là **số mili-giây** mà tác vụ sẽ tạm dừng. Vì khoảng thời gian này khá ngắn, hãy **nhân số đó với 200** để kéo dài thời gian tạm dừng.
* 
* Thêm một khối try/catch và cho luồng (thread) tạm dừng.
* 
* Thay thế câu lệnh return hiện tại để trả về chuỗi:"Awake at last after sleeping for xx milliseconds", trong đó xx là số mili-giây mà ứng dụng đã ngủ.
* 
* Khi phương thức doInBackground() hoàn tất, giá trị trả về sẽ tự động được truyền đến phương thức onPostExecute().

Triển khai phương thức onPostExecute() ,nhận một chuỗi (String) làm đối số và hiển thị chuỗi đó trong TextView

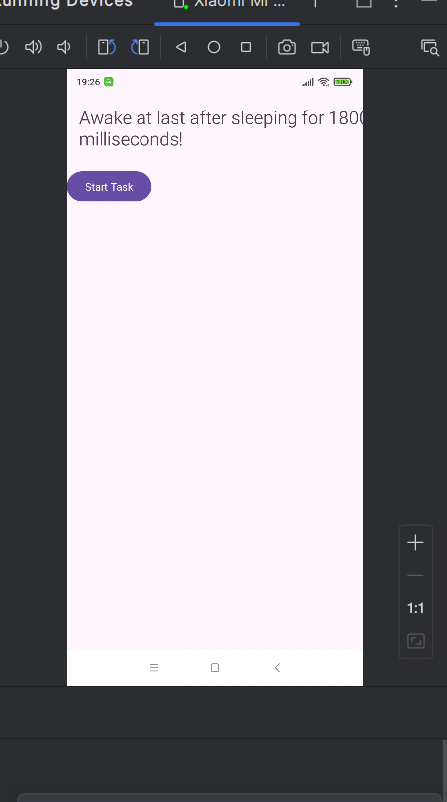


Tham số String của phương thức này chính là kiểu dữ liệu mà bạn đã định nghĩa ở tham số thứ ba trong khai báo lớp AsyncTask và đây cũng là giá trị được trả về từ phương thức doInBackground()

Vì mTextView là một WeakReference, bạn cần giải tham chiếu (dereference) nó bằng phương thức get().Sau đó, gọi setText() trên đối tượng TextView để hiển thị kết quả.

* Bây giờ ứng dụng của bạn đã có một lớp AsyncTask thực hiện công việc trong nền (hoặc ít nhất là giả lập công việc bằng cách gọi sleep()). Bạn có thể triển khai phương thức onClick cho nút "Start Task" để kích hoạt tác vụ nền (AsyncTask) khi người dùng nhấn vào.
* Trong tệp MainActivity.java, thêm một biến thành viên để lưu trữ TextView
* 
* Trong phương thức onCreate(), khởi tạo mTextView bằng TextView trong giao diện (layout).
* 
* Trong phương thức startTask(), cập nhật TextView để hiển thị dòng chữ "Napping…".Trích xuất chuỗi "Napping…" vào tài nguyên chuỗi (strings.xml).
* 
* Tạo một thể hiện (instance) của SimpleAsyncTask, truyền mTextView vào constructor.Gọi execute() trên thể hiện SimpleAsyncTask để bắt đầu tác vụ nền.
* 

### 



Trả lời câu hỏi

Câu hỏi 1

Đối với một ProgressBar:

1. Làm thế nào để xác định phạm vi giá trị mà một ProgressBar có thể hiển thị?
2. Làm thế nào để thay đổi mức độ của thanh tiến trình được lấp đầy?

Câu hỏi 2

Nếu một AsyncTask được định nghĩa như sau:

1. Kiểu của giá trị được truyền vào doInBackground() trong AsyncTask là gì?
2. Kiểu của giá trị được truyền vào callback báo cáo tiến trình của tác vụ là gì?
3. Kiểu của giá trị được truyền vào callback được thực hiện khi tác vụ hoàn tất là gì?

Câu hỏi 3

Để báo cáo tiến trình của công việc được thực hiện bởi một AsyncTask, bạn cần triển khai phương thức callback nào và gọi phương thức nào trong lớp con AsyncTask?

* Triển khai publishProgress(). Gọi publishProgress().
* Triển khai publishProgress(). Gọi onProgressUpdate().
* Triển khai onProgressUpdate(). Gọi publishProgress().
* Triển khai onProgressUpdate(). Gọi onProgressUpdate().

Hướng dẫn cho người chấm

Kiểm tra ứng dụng của bạn có các tính năng sau:

* Giao diện bao gồm một ProgressBar với các thuộc tính thích hợp để xác định phạm vi giá trị.
* AsyncTask phân chia tổng thời gian ngủ thành các đoạn và cập nhật thanh tiến trình sau mỗi đoạn.
* AsyncTask gọi phương thức thích hợp và triển khai callback thích hợp để cập nhật thanh tiến trình.
* AsyncTask cần biết các view nào để cập nhật. Tùy thuộc vào việc AsyncTask được triển khai như một lớp con hay không, các view có thể được truyền vào constructor của AsyncTask hoặc định nghĩa như các biến thành viên trong Activity.

### AsyncTask và AsyncTaskLoader

**Giới thiệu**

Trong bài thực hành này, bạn sẽ sử dụng AsyncTask để bắt đầu một tác vụ nền nhằm lấy dữ liệu từ internet thông qua một API REST đơn giản. Bạn sẽ sử dụng Google APIs Explorer để truy vấn Books API, triển khai truy vấn này trong một luồng làm việc bằng AsyncTask, và hiển thị kết quả trong giao diện người dùng. Sau đó, bạn sẽ triển khai lại cùng một tác vụ nền bằng AsyncTaskLoader, một cách hiệu quả hơn để cập nhật giao diện người dùng.

**Những gì bạn nên biết**

* Bạn nên có khả năng:
* Tạo một activity.
* Thêm một TextView vào layout của activity.
* Triển khai chức năng onClick cho một nút trong layout của bạn.
* Triển khai một AsyncTask và hiển thị kết quả trong giao diện người dùng.
* Truyền thông tin giữa các activity dưới dạng extras.

**Những gì bạn sẽ học**

* Cách sử dụng Google APIs Explorer để điều tra các Google APIs và xem các phản hồi JSON cho các yêu cầu HTTP.
* Cách sử dụng Google Books API để lấy dữ liệu qua internet và giữ cho giao diện người dùng nhanh và phản hồi tốt. Bạn sẽ không học chi tiết về Books API—ứng dụng của bạn chỉ sử dụng chức năng tìm kiếm sách đơn giản.
* Cách phân tích kết quả JSON từ truy vấn API của bạn.
* Cách triển khai một AsyncTaskLoader bảo tồn dữ liệu khi có sự thay đổi cấu hình.
* Cách cập nhật giao diện người dùng sử dụng các callback của loader.

Những gì bạn sẽ làm

* Sử dụng Google APIs Explorer để tìm hiểu về Books API.
* Tạo ứng dụng "Who Wrote It?" mà truy vấn Books API thông qua một luồng làm việc và hiển thị kết quả trong giao diện người dùng.
* Sửa đổi ứng dụng "Who Wrote It?" để sử dụng AsyncTaskLoader thay vì AsyncTask.

Tổng quan về ứng dụng

Bạn sẽ xây dựng một ứng dụng chứa một EditText và một Button.

* Người dùng nhập tên của cuốn sách vào EditText và nhấn nút.
* Nút sẽ thực thi một AsyncTask truy vấn Google Books API để tìm kiếm tác giả và tiêu đề của cuốn sách mà người dùng đang tìm kiếm.
* Kết quả sẽ được lấy và hiển thị trong một TextView bên dưới nút.

Khi ứng dụng hoạt động, bạn sẽ sửa đổi ứng dụng để sử dụng AsyncTaskLoader thay vì lớp AsyncTask.

**Nhiệm vụ 1. Khám phá Google Books API**

Trong bài thực hành này, bạn sẽ sử dụng Google Books API để tìm kiếm thông tin về một cuốn sách, chẳng hạn như tác giả và tiêu đề của cuốn sách. Books API cung cấp quyền truy cập chương trình vào dịch vụ tìm kiếm sách của Google thông qua các API REST. Đây là dịch vụ mà bạn sử dụng khi thực hiện tìm kiếm trên Google Books.

**1.1 Gửi yêu cầu đến Books API**

1. Truy cập Google APIs Explorer tại https://developers.google.com/apis-explorer/.
2. Nhấp vào Services trong menu bên trái, sau đó chọn Books API.
3. Tìm books.volumes.list và nhấp vào tên chức năng đó. Để tìm trong trang, bạn có thể nhấn Control+F (hoặc Command+F trên Mac).

Bạn sẽ thấy một trang web liệt kê các tham số của chức năng **Books API** thực hiện tìm kiếm sách.

1. Trong trường q, nhập tên một cuốn sách hoặc một phần tên cuốn sách, ví dụ "Romeo". Tham số q là trường duy nhất bắt buộc.
2. Trong trường maxResults, nhập 10 để giới hạn kết quả chỉ còn 10 cuốn sách phù hợp nhất.
3. Trong trường printType, nhập books để giới hạn kết quả chỉ cho các cuốn sách đang in.
4. Đảm bảo rằng công tắc Authorize requests using OAuth 2.0 ở đầu biểu mẫu tắt.
5. Nhấp vào liên kết Execute without OAuth ở dưới cùng của biểu mẫu.
6. Cuộn xuống để xem yêu cầu HTTP và phản hồi HTTP.

Yêu cầu HTTP là một uniform resource identifier (URI). URI là một chuỗi xác định một tài nguyên, và URL là một loại URI xác định một tài nguyên web. Đối với Books API, yêu cầu là một URL. Các tham số tìm kiếm mà bạn đã nhập vào biểu mẫu sẽ theo sau ? trong URL. Lưu ý trường API key ở cuối URL. Vì lý do bảo mật, khi bạn truy cập một API công cộng, bạn phải lấy một API key và đưa vào yêu cầu của mình. Books API không yêu cầu API key, vì vậy bạn có thể bỏ qua phần đó trong yêu cầu URI của ứng dụng.

**1.2 Phân tích phản hồi của Books API**

Phản hồi cho truy vấn nằm ở phía dưới của trang. Phản hồi sử dụng định dạng **JSON**, một định dạng phổ biến cho các phản hồi truy vấn API. Trong trang web của APIs Explorer, mã JSON được định dạng gọn gàng để dễ đọc. Trong ứng dụng của bạn, phản hồi JSON sẽ được trả về từ dịch vụ API dưới dạng một chuỗi đơn, và bạn sẽ cần phân tích chuỗi đó để trích xuất thông tin cần thiết.

Phản hồi bao gồm các cặp tên/giá trị được phân tách bằng dấu phẩy. Ví dụ, "kind": "books#volumes" là một cặp tên/giá trị, trong đó "kind" là tên và "books#volumes" là giá trị. Đây là định dạng JSON.

1. Tìm giá trị cho tên "title" của một cuốn sách. Lưu ý rằng kết quả này chứa một giá trị duy nhất.
2. Tìm giá trị cho tên "authors" của một cuốn sách. Lưu ý rằng kết quả này là một mảng có thể chứa nhiều giá trị hơn một.

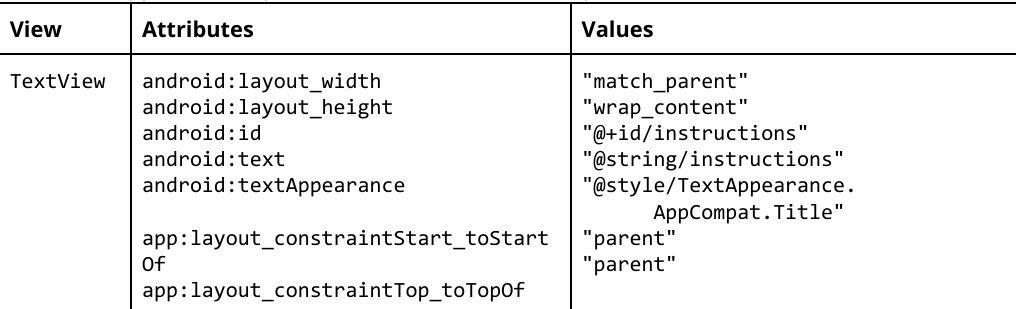
Tìm kiếm sách bao gồm tất cả các cuốn sách chứa chuỗi tìm kiếm, với nhiều đối tượng để đại diện cho từng cuốn sách. Trong bài thực hành này, bạn chỉ trả về tiêu đề và tác giả của mục đầu tiên trong phản hồi.

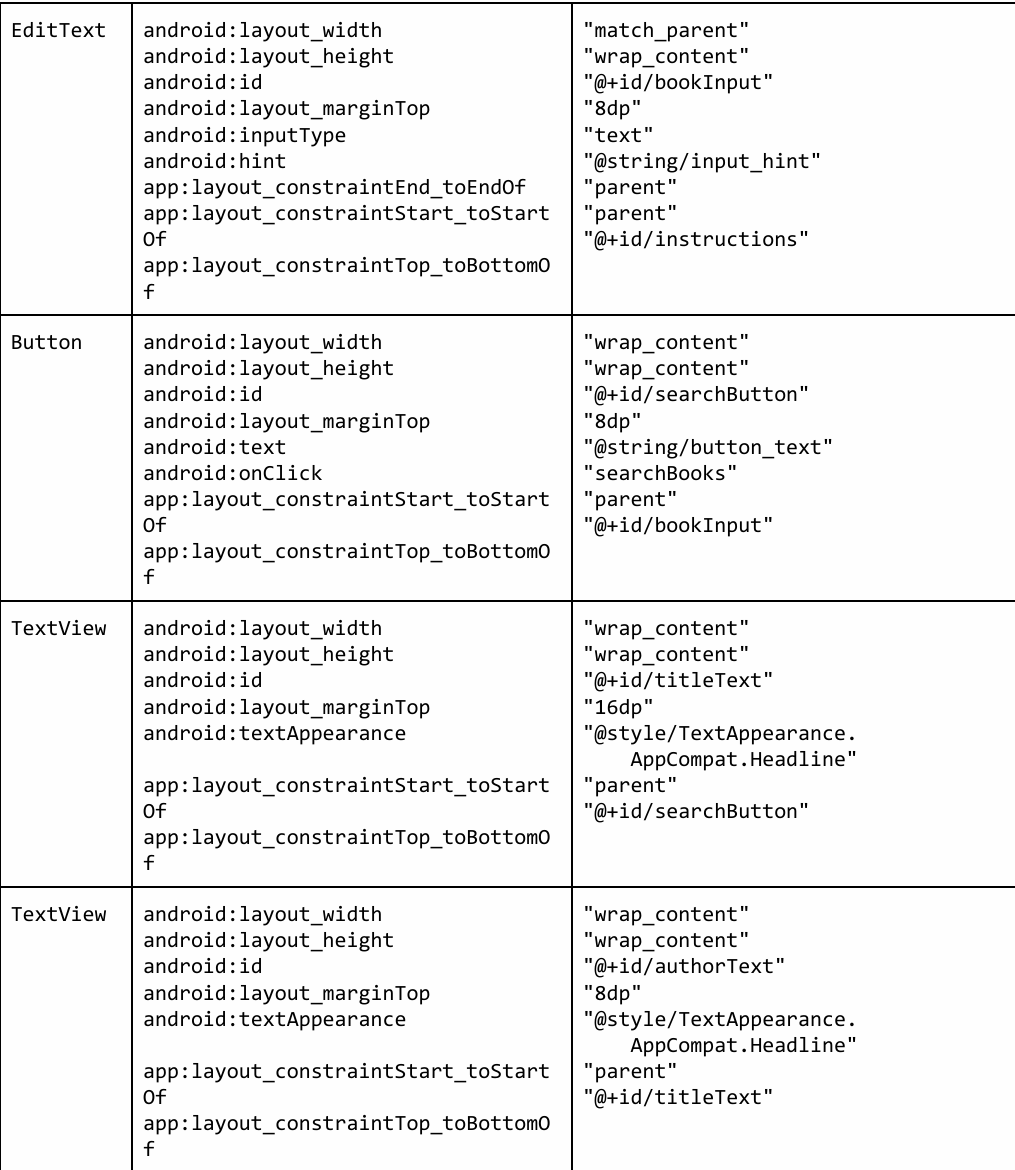
**Nhiệm vụ 2. Tạo ứng dụng "Who Wrote It?"**

Bây giờ bạn đã quen thuộc với Books API, đã đến lúc thiết lập giao diện của ứng dụng.

**2.1 Tạo dự án và giao diện người dùng (UI)**

1. Tạo một dự án mới có tên "WhoWroteIt", sử dụng mẫu Empty Activity. Chấp nhận mặc định cho tất cả các tùy chọn khác.
2. Mở tệp layout activity\_main.xml. Nhấp vào tab Text.
3. Thêm thuộc tính layout\_margin vào ConstraintLayout cấp cao nhất.
4. Xóa TextView hiện có.
5. Thêm các phần tử UI và thuộc tính sau vào tệp layout. Lưu ý rằng các tài nguyên chuỗi sẽ hiển thị bằng màu đỏ; bạn sẽ định nghĩa chúng ở bước tiếp theo.





1.Trong tệp strings.xml, thêm các tài nguyên chuỗi này:

1.Thuộc tính onClick cho nút sẽ được làm nổi bật bằng màu vàng, vì phương thức searchBooks() chưa được triển khai trong MainActivity. Để tạo stub cho phương thức trong MainActivity, đặt con trỏ vào văn bản được làm nổi bật, nhấn Alt+Enter (hoặc Option+Enter trên Mac) và chọn Create 'searchBooks(View)' in 'MainActivity'.

Mã giải pháp cho activity\_main.xml

**2.2 Lấy thông tin từ người dùng**

Để truy vấn Books API, bạn cần lấy thông tin từ người dùng từ EditText.

1. Trong MainActivity.java, tạo các biến thành viên cho EditText, TextView tác giả và TextView tiêu đề.
2. Khởi tạo các biến đó với các view trong phương thức onCreate().
3. Trong phương thức searchBooks(), lấy văn bản từ view EditText. Chuyển đổi văn bản thành String và gán nó cho một biến.

**2.3 Tạo lớp AsyncTask rỗng**

Bây giờ bạn đã sẵn sàng để kết nối internet và sử dụng Books API. Trong nhiệm vụ này, bạn sẽ tạo một lớp con AsyncTask mới có tên FetchBook để xử lý việc kết nối với mạng.

Kết nối mạng có thể chậm, điều này có thể làm cho ứng dụng của bạn không ổn định hoặc chậm. Vì lý do này, không nên thực hiện kết nối mạng trên luồng UI. Nếu bạn cố gắng thực hiện kết nối mạng trên luồng UI, thời gian chạy Android có thể phát sinh một NetworkOnMainThreadException để cảnh báo bạn rằng đây là một ý tưởng tồi.

Thay vào đó, hãy sử dụng một lớp con của AsyncTask để thực hiện các kết nối mạng. Một AsyncTask yêu cầu ba tham số kiểu: loại tham số đầu vào, loại chỉ báo tiến trình và loại kết quả.

1.Tạo một lớp Java trong ứng dụng của bạn có tên FetchBook, kế thừa từ AsyncTask. Các tham số kiểu tổng quát cho lớp sẽ là <String, Void, String>. (String vì truy vấn là một chuỗi, Void vì không có chỉ báo tiến trình, và String vì phản hồi JSON là một chuỗi.)

2.Triển khai phương thức yêu cầu, doInBackground(). Để làm điều này, đặt con trỏ vào văn bản gạch chân đỏ, nhấn Alt+Enter (hoặc Option+Enter trên Mac) và chọn **Implement methods.** Chọn **doInBackground()** và nhấp **OK**.

Đảm bảo rằng các tham số và kiểu trả về là chính xác. (Phương thức nhận một danh sách biến các đối tượng String và trả về một String.)

3.Chọn **Code > Override methods**, hoặc nhấn Ctrl+O. Chọn phương thức onPostExecute() để chèn định nghĩa phương thức vào lớp. Phương thức onPostExecute() nhận một String làm tham số và trả về void.

4.Để hiển thị kết quả trong các đối tượng TextView trong MainActivity, bạn phải có quyền truy cập vào các text views đó bên trong AsyncTask. Tạo các biến thành viên WeakReference để tham chiếu đến hai text views hiển thị kết quả.

**Lưu ý:** Như bạn đã học trong bài thực hành trước, bạn sử dụng các đối tượng WeakReference cho các text views này (thay vì các đối tượng TextView thực tế) để tránh rò rỉ ngữ cảnh từ Activity. Các tham chiếu yếu ngăn ngừa rò rỉ bộ nhớ bằng cách cho phép đối tượng được tham chiếu bởi tham chiếu đó được thu gom rác nếu cần thiết.

5.Tạo một constructor cho lớp FetchBook bao gồm các view TextView từ MainActivity, và khởi tạo các biến thành viên trong constructor đó.

Mã giải pháp cho **FetchBook**

### 2.4 Tạo lớp ****NetworkUtils**** và xây dựng URI

Bạn cần mở một kết nối internet và truy vấn **Books API**. Vì bạn có thể sử dụng chức năng này nhiều lần, bạn có thể muốn tạo một lớp tiện ích cho chức năng này hoặc phát triển một lớp con hữu ích để thuận tiện cho riêng bạn.

Trong nhiệm vụ này, bạn sẽ viết mã để kết nối internet trong một lớp trợ giúp có tên **NetworkUtils**.

1. Tạo một lớp Java mới trong ứng dụng của bạn có tên **NetworkUtils**. Lớp **NetworkUtils** không kế thừa từ bất kỳ lớp nào khác.
2. Để ghi lại, tạo một biến **LOG\_TAG** với tên của lớp:
3. Tạo một phương thức tĩnh có tên **getBookInfo()**. Phương thức **getBookInfo()** nhận tham số tìm kiếm dưới dạng **String** và trả về phản hồi **JSON** dưới dạng chuỗi từ API mà bạn đã xem xét trước đó.
4. Tạo các biến cục bộ sau trong phương thức **getBookInfo()**. Bạn sẽ cần những biến này để kết nối internet, đọc dữ liệu đến và lưu trữ chuỗi phản hồi.
5. Ở cuối phương thức getBookInfo(), trả về giá trị của bookJSONString.
6. Thêm một khung xương cho khối try/catch/finally trong getBookInfo(), sau các biến cục bộ và trước câu lệnh return.

Trong khối try, bạn sẽ xây dựng URI và thực hiện truy vấn. Trong khối catch, bạn sẽ xử lý các vấn đề với yêu cầu. Trong khối finally, bạn sẽ đóng kết nối mạng sau khi hoàn tất việc nhận dữ liệu JSON.

1. Tạo các hằng số thành viên sau ở đầu lớp NetworkUtils, dưới hằng số LOG\_TAG:

Như bạn đã thấy trong yêu cầu trên trang web Books API, tất cả các yêu cầu đều bắt đầu với cùng một URI. Để chỉ định loại tài nguyên, hãy thêm các tham số truy vấn vào URI cơ bản đó. Thông thường, các tham số truy vấn này được tách ra thành các hằng số và kết hợp chúng bằng cách sử dụng Uri.Builder để có thể tái sử dụng cho các URI khác nhau. Lớp Uri có một phương thức tiện lợi, Uri.buildUpon(), trả về một URI.Builder mà bạn có thể sử dụng.

Đối với ứng dụng này, bạn sẽ hạn chế số lượng và loại kết quả được trả về để tăng tốc độ truy vấn. Để hạn chế truy vấn, bạn chỉ tìm kiếm các cuốn sách đã được in.

1. Trong phương thức getBookInfo(), xây dựng URI yêu cầu trong khối try.
2. Cũng bên trong khối try, chuyển đổi URI của bạn thành một đối tượng URL.

**2.5 Thực hiện yêu cầu**

Yêu cầu API này sử dụng lớp HttpURLConnection kết hợp với InputStream, BufferedReader, và StringBuffer để lấy phản hồi JSON từ web. Nếu bất kỳ lúc nào trong quá trình này gặp lỗi và InputStream hoặc StringBuffer trống, yêu cầu sẽ trả về null, chỉ ra rằng truy vấn đã thất bại.

1. Trong khối try của phương thức getBookInfo(), mở kết nối URL và thực hiện yêu cầu:

1.Cũng bên trong khối try, thiết lập phản hồi từ kết nối bằng cách sử dụng InputStream, BufferedReader, và StringBuilder.

1.Đọc đầu vào từng dòng một vào chuỗi trong khi vẫn còn đầu vào:

**Lưu ý:** Vòng lặp while thêm dòng vào builder chuỗi trong hai bước: một bước cho dòng dữ liệu phản hồi và một bước để thêm ký tự xuống dòng ("\n").

Ký tự xuống dòng không ảnh hưởng đến việc phân tích JSON của phản hồi, nhưng giúp dễ dàng hơn khi gỡ lỗi phản hồi khi bạn xem nó trong log.

2.Ở cuối đầu vào, kiểm tra chuỗi để xem có nội dung phản hồi tồn tại không. Trả về null nếu phản hồi trống.

3.Chuyển đổi đối tượng StringBuilder thành một String và lưu trữ nó trong biến bookJSONString.

4.Trong khối finally, đóng cả kết nối và BufferedReader.

**Lưu ý**: Mỗi lần kết nối thất bại vì bất kỳ lý do nào, mã này sẽ trả về null. Điều này có nghĩa là phương thức onPostExecute() trong lớp FetchBook phải kiểm tra tham số đầu vào của nó cho chuỗi null và thông báo cho người dùng về sự cố.

Chiến lược xử lý lỗi này là đơn giản, vì người dùng không biết lý do tại sao kết nối thất bại. Một giải pháp tốt hơn cho ứng dụng sản xuất là xử lý từng điểm thất bại khác nhau để người dùng nhận được phản hồi hữu ích.

5.Ngay trước khi trả về cuối cùng, in giá trị của biến bookJSONString ra log.

6.Trong FetchBook, sửa đổi phương thức doInBackground() để gọi phương thức NetworkUtils.getBookInfo(), truyền vào tham số tìm kiếm mà bạn đã lấy từ đối số params được truyền vào bởi hệ thống. (Tham số tìm kiếm là giá trị đầu tiên trong mảng chuỗi.) Trả về kết quả của phương thức này. (Dòng này thay thế việc trả về null.)

1.Trong MainActivity, thêm dòng này vào cuối phương thức searchBooks() để khởi động tác vụ nền với phương thức execute() và chuỗi truy vấn.

1.Chạy ứng dụng của bạn và thực hiện tìm kiếm. Ứng dụng của bạn sẽ gặp lỗi. Trong Android Studio, nhấp vào Logcat để xem log và xem điều gì gây ra lỗi. Bạn sẽ thấy dòng sau:

Lỗi này cho biết rằng bạn chưa bao gồm quyền truy cập internet trong tệp manifest Android của bạn. Kết nối internet gây ra các vấn đề về bảo mật, đó là lý do tại sao các ứng dụng không có khả năng kết nối theo mặc định. Trong nhiệm vụ tiếp theo, bạn sẽ thêm quyền internet vào manifest.

**2.6 Thêm quyền truy cập internet**

1. Mở tệp AndroidManifest.xml.
2. Thêm mã sau ngay trước phần tử <application>:
3. Xây dựng và chạy lại ứng dụng của bạn. Trong Android Studio, nhấp vào Logcat để xem log. Lưu ý rằng lần này, truy vấn hoạt động đúng và chuỗi JSON kết quả được in ra log.

**2.7 Phân tích chuỗi JSON**

Bây giờ bạn đã có phản hồi JSON cho truy vấn của mình, bạn cần phân tích các kết quả để trích xuất thông tin mà bạn muốn hiển thị trong giao diện người dùng của ứng dụng. Java có các lớp trong API cốt lõi của nó giúp bạn phân tích và xử lý dữ liệu kiểu JSON. Quá trình này, cũng như cập nhật giao diện người dùng, diễn ra trong phương thức onPostExecute() của lớp FetchBook.

Có khả năng phương thức doInBackground() không trả về chuỗi JSON như mong đợi. Ví dụ, khối try/catch có thể gặp lỗi và ném ra ngoại lệ, kết nối mạng có thể hết thời gian, hoặc các lỗi không được xử lý khác có thể xảy ra. Trong những trường hợp đó, việc phân tích JSON sẽ thất bại và ném ra ngoại lệ. Để xử lý trường hợp này, hãy thực hiện phân tích JSON trong một khối try/catch, và xử lý trường hợp dữ liệu không chính xác hoặc không đầy đủ được trả về.

1. Trong lớp FetchBook, trong phương thức onPostExecute(), thêm một khối try/catch bên dưới lệnh gọi đến super.
2. Bên trong khối try, sử dụng các lớp JSONObject và JSONArray để lấy mảng JSON của các mục từ chuỗi kết quả.
3. Khởi tạo các biến được sử dụng cho vòng lặp phân tích.
4. Lặp qua mảng itemsArray, kiểm tra từng cuốn sách để lấy thông tin về tiêu đề và tác giả. Trong mỗi vòng lặp, kiểm tra xem có cả tác giả và tiêu đề không, và nếu có, thoát khỏi vòng lặp. Điều này đảm bảo rằng chỉ những mục có cả tiêu đề và tác giả sẽ được hiển thị.

**Lưu ý:** Vòng lặp kết thúc tại kết quả đầu tiên phù hợp trong phản hồi. Có thể có nhiều phản hồi hơn, nhưng ứng dụng này chỉ hiển thị kết quả đầu tiên.

1. Nếu tìm thấy phản hồi phù hợp, cập nhật giao diện người dùng với phản hồi đó. Vì các tham chiếu đến các đối tượng TextView là các đối tượng WeakReference, bạn phải giải tham chiếu chúng bằng cách sử dụng phương thức get().
2. Nếu vòng lặp đã dừng và kết quả không có mục nào có cả tác giả hợp lệ và tiêu đề hợp lệ, hãy đặt TextView tiêu đề thành một tài nguyên chuỗi "không có kết quả" và xóa TextView tác giả.
3. Trong khối catch, in lỗi ra log. Đặt TextView tiêu đề thành tài nguyên chuỗi "không có kết quả" và xóa TextView tác giả.
4. Thêm tài nguyên no\_results vào strings.xml.

Mã giải pháp cho phân tích JSON

**Nhiệm vụ 3: Triển khai các thực hành tốt về giao diện người dùng**

Bây giờ bạn đã có một ứng dụng hoạt động sử dụng Books API để thực hiện tìm kiếm sách. Tuy nhiên, một vài điều không hoạt động như mong đợi:

* Khi người dùng nhấn Tìm kiếm sách, bàn phím không biến mất. Người dùng không có dấu hiệu nào cho thấy truy vấn đang được thực hiện.
* Nếu không có kết nối mạng, hoặc nếu trường tìm kiếm trống, ứng dụng vẫn cố gắng truy vấn API và thất bại mà không cập nhật đúng giao diện người dùng.
* Nếu bạn xoay màn hình trong khi truy vấn, AsyncTask trở nên không còn kết nối với Activity, và không thể cập nhật giao diện người dùng với kết quả.

Bạn sẽ sửa hai vấn đề đầu tiên trong phần này, và vấn đề cuối cùng trong Nhiệm vụ 4.

**3.1 Ẩn bàn phím và cập nhật TextView**

Trải nghiệm người dùng khi tìm kiếm không trực quan. Khi người dùng nhấn nút, bàn phím vẫn hiển thị, và người dùng không có cách nào biết rằng truy vấn đang được thực hiện.

Một giải pháp là ẩn bàn phím một cách lập trình và cập nhật một trong các TextView kết quả để hiển thị "Đang tải…" trong khi truy vấn đang được thực hiện.

1.Trong MainActivity, thêm mã sau vào phương thức searchBooks(), sau khi định nghĩa queryString. Mã này ẩn bàn phím khi người dùng nhấn nút.

1. Ngay dưới lệnh gọi để thực hiện tác vụ FetchBook, thêm mã để thay đổi TextView tiêu đề thành thông điệp đang tải và xóa TextView tác giả.
2. Thêm tài nguyên loading vào strings.xml:

**3.2 Quản lý trạng thái mạng và trường tìm kiếm trống**

Bất cứ khi nào ứng dụng của bạn sử dụng mạng, nó cần xử lý khả năng kết nối mạng không khả dụng. Trước khi cố gắng kết nối với mạng, ứng dụng của bạn nên kiểm tra trạng thái kết nối mạng. Ngoài ra, nó cũng không nên cố gắng truy vấn Books API nếu người dùng chưa nhập chuỗi truy vấn.

1. Trong phương thức searchBooks(), sử dụng các lớp ConnectivityManager và NetworkInfo để kiểm tra kết nối mạng. Thêm mã sau vào sau mã quản lý đầu vào ẩn bàn phím:
2. Thêm một điều kiện xung quanh lệnh gọi tác vụ FetchBook và cập nhật TextView để đảm bảo rằng kết nối mạng tồn tại, rằng mạng đã được kết nối, và rằng một chuỗi truy vấn có sẵn.
3. Thêm một khối else vào điều kiện đó. Trong khối else, cập nhật giao diện người dùng với thông điệp lỗi no\_search\_term nếu không có từ để tìm kiếm, và thông điệp lỗi no\_network nếu không có kết nối mạng.
4. Thêm các tài nguyên no\_search\_term và no\_network vào strings.xml:

Mã giải pháp

Mã giải pháp cho thực hành này đến thời điểm này có trong dự án Android Studio WhoWroteIt.

**Nhiệm vụ 4: Chuyển đổi sang AsyncTaskLoader**

Khi bạn sử dụng AsyncTask để thực hiện các thao tác trong nền, luồng nền đó không thể cập nhật giao diện người dùng nếu có thay đổi cấu hình trong khi tác vụ nền đang chạy. Để giải quyết tình huống này, hãy sử dụng lớp AsyncTaskLoader.

AsyncTaskLoader tải dữ liệu trong nền và tái liên kết các tác vụ nền với Activity, ngay cả sau khi có thay đổi cấu hình. Với AsyncTaskLoader, nếu bạn xoay thiết bị trong khi tác vụ đang chạy, kết quả vẫn sẽ được hiển thị chính xác trong Activity.

Tại sao sử dụng AsyncTask nếu AsyncTaskLoader hữu ích hơn nhiều? Câu trả lời là tùy thuộc vào tình huống. Nếu tác vụ nền có khả năng hoàn thành trước khi có bất kỳ thay đổi cấu hình nào xảy ra và không quan trọng cho tác vụ đó để cập nhật giao diện người dùng, một AsyncTask có thể là đủ. Lớp AsyncTaskLoader thực tế sử dụng một AsyncTask ở phía sau để thực hiện chức năng của nó.

**Lưu ý:** Lớp AsyncTaskLoader là một phần của API Loader của nền tảng Android, đây là một khung để quản lý việc tải dữ liệu vào ứng dụng của bạn trong nền. Các loader đã bị ngừng sử dụng trong Android P (API 28) để thay thế bằng ViewModels và LiveData. Lớp AsyncTaskLoader vẫn còn khả dụng, nhưng để tương thích ngược đầy đủ, hãy đảm bảo sử dụng AsyncTaskLoader và các lớp liên quan khác từ Thư viện Hỗ trợ Android.

Trong bài tập này, bạn sẽ học cách sử dụng AsyncTaskLoader thay vì AsyncTask để thực hiện truy vấn Books API.

**4.1 Tạo lớp AsyncTaskLoader**

1.Để giữ lại kết quả từ bài thực hành trước, hãy sao chép dự án WhoWroteIt. Đổi tên dự án đã sao chép thành "WhoWroteItLoader".

2.Tạo một lớp có tên BookLoader mở rộng AsyncTaskLoader với kiểu tham số hóa <String>.

Đảm bảo nhập lớp AsyncTaskLoader từ Thư viện Hỗ trợ v4.

3.Triển khai phương thức bắt buộc loadInBackground(). Lưu ý sự tương đồng giữa phương thức này và phương thức doInBackground() ban đầu từ AsyncTask.

4.Tạo constructor cho lớp BookLoader. Với con trỏ văn bản của bạn trên dòng khai báo lớp, nhấn Alt+Enter (hoặc Option+Enter trên Mac) và chọn Create constructor matching super. Điều này sẽ tạo một constructor với tham số là Context.

**4.2 Triển khai các phương thức cần thiết**

1. Nhấn Ctrl+O để mở menu các phương thức ghi đè, và chọn onStartLoading. Hệ thống sẽ gọi phương thức này khi bạn khởi động loader.
2. Bên trong phần thân phương thức onStartLoading(), gọi forceLoad() để bắt đầu phương thức loadInBackground(). Loader sẽ không bắt đầu tải dữ liệu cho đến khi bạn gọi phương thức forceLoad().
3. Tạo một biến thành viên có tên mQueryString để lưu trữ chuỗi cho truy vấn Books API. Sửa đổi constructor để nhận một String làm tham số và gán nó cho biến mQueryString.
4. Trong phương thức loadInBackground(), thay thế câu lệnh return bằng mã sau, gọi phương thức NetworkUtils.getBookInfo() với chuỗi truy vấn và trả về kết quả:

**4.3 Sửa đổi MainActivity**

Kết nối giữa AsyncTaskLoader và Activity gọi nó được thực hiện với LoaderManager và giao diện LoaderCallbacks. Các callback loader này là một tập hợp các phương thức trong activity được gọi bởi LoaderManager khi loader đang được tạo, khi dữ liệu đã hoàn tất tải và khi loader được đặt lại. Các callback loader nhận kết quả của tác vụ và truyền chúng trở lại giao diện người dùng của activity.

1.Trong MainActivity, thêm triển khai LoaderManager.LoaderCallbacks vào khai báo lớp, với kiểu tham số hóa là String:

Đảm bảo nhập lớp LoaderManager.LoaderCallbacks từ Thư viện Hỗ trợ v4.

2.Triển khai tất cả các phương thức callback cần thiết từ giao diện. Bao gồm onCreateLoader(), onLoadFinished(), và onLoaderReset(). Đặt con trỏ của bạn trên dòng chữ ký lớp và nhấn Alt+Enter (hoặc Option+Enter trên Mac). Đảm bảo rằng tất cả các phương thức đều được chọn và nhấn OK.

* onCreateLoader() được gọi khi bạn khởi tạo loader.
* onLoadFinished() được gọi khi tác vụ của loader hoàn thành. Đây là nơi bạn thêm mã để cập nhật giao diện người dùng với kết quả.
* onLoaderReset() dọn dẹp bất kỳ tài nguyên còn lại nào.

Đối với ứng dụng này, bạn chỉ cần triển khai hai phương thức đầu tiên. Để trống onLoaderReset().

3.Phương thức searchBooks() là phương thức onClick cho nút. Trong searchBooks(), thay thế lệnh gọi để thực hiện tác vụ FetchBook bằng lệnh gọi đến restartLoader(). Truyền vào chuỗi truy vấn mà bạn đã lấy từ EditText trong đối tượng Bundle của loader:

Phương thức restartLoader() được định nghĩa bởi LoaderManager, quản lý tất cả các loader được sử dụng trong một activity hoặc fragment. Mỗi activity có đúng một thể hiện LoaderManager chịu trách nhiệm cho vòng đời của các Loader mà activity quản lý.

Phương thức restartLoader() nhận ba tham số:

* Một loader id, hữu ích nếu bạn triển khai nhiều loader trong activity của mình.
* Một Bundle cho bất kỳ dữ liệu nào mà loader cần.
* Thể hiện của LoaderCallbacks mà bạn đã triển khai trong activity của mình. Nếu bạn muốn loader gửi kết quả đến MainActivity, hãy chỉ định điều này làm tham số thứ ba.

**4.4 Triển khai các callback loader**

Trong nhiệm vụ này, bạn sẽ triển khai các phương thức callback onCreateLoader() và onLoadFinished() để xử lý tác vụ nền.

1. Trong onCreateLoader(), thay thế câu lệnh return bằng một câu lệnh trả về một thể hiện của lớp BookLoader. Truyền vào context (this) và queryString nhận được từ Bundle đã truyền vào.

1.Sao chép mã từ onPostExecute() trong lớp FetchBook của bạn vào onLoadFinished() trong MainActivity. Xóa lệnh gọi đến super.onPostExecute(). Đây là mã phân tích kết quả JSON để tìm kiếm một kết quả phù hợp với chuỗi truy vấn.

2.Xóa tất cả các lệnh gọi đến get() cho từng đối tượng TextView. Vì việc cập nhật giao diện người dùng xảy ra trong chính Activity, bạn không còn cần tham chiếu yếu đến các view gốc nữa.

3.Thay thế tham số cho constructor của JSONObject (biến s) bằng tham số data.

4.Chạy ứng dụng của bạn. Bạn nên có cùng một chức năng như trước, nhưng bây giờ trong một loader! Tuy nhiên, khi bạn xoay thiết bị, dữ liệu hiển thị sẽ bị mất. Điều này xảy ra vì khi activity được tạo (hoặc tái tạo), activity không biết rằng một loader đang chạy. Để kết nối lại với loader, bạn cần một phương thức initLoader() trong onCreate() của MainActivity.

5.Thêm mã sau vào onCreate() để kết nối lại với loader, nếu loader đã tồn tại:

Nếu loader tồn tại, hãy khởi tạo nó. Bạn chỉ muốn tái liên kết loader với activity nếu một truy vấn đã được thực hiện. Trong trạng thái ban đầu của ứng dụng, không có dữ liệu nào được tải, vì vậy không có dữ liệu nào để bảo tồn.

1. Chạy lại ứng dụng của bạn và xoay thiết bị. Bây giờ, loader manager giữ lại dữ liệu của bạn qua các thay đổi cấu hình thiết bị!
2. Xóa lớp FetchBook, vì nó không còn được sử dụng.

**Mã giải pháp**

Mã giải pháp cho nhiệm vụ này có trong dự án Android Studio WhoWroteItLoader.

**Thử thách lập trình**

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

**Thử thách 1:** Khám phá Books API chi tiết hơn và tìm một tham số tìm kiếm hạn chế kết quả chỉ cho các cuốn sách có thể tải xuống định dạng EPUB. Thêm tham số vào yêu cầu của bạn và xem các kết quả.

**Tóm tắt**

1. Các tác vụ kết nối với mạng không nên được thực hiện trên luồng UI. Thời gian chạy Android thường sẽ phát sinh một ngoại lệ nếu bạn cố gắng kết nối mạng hoặc truy cập tệp trên luồng UI.
2. Sử dụng Books Search API để truy cập Google Books một cách lập trình. Một yêu cầu API đến Google Books có dạng URL, và phản hồi là một chuỗi JSON.
3. Sử dụng Google APIs Explorer để khám phá các API của Google một cách tương tác.
4. Sử dụng getText() để lấy văn bản từ một view EditText. Để chuyển đổi văn bản thành một chuỗi đơn giản, sử dụng toString().
5. Phương thức Uri.buildUpon() trả về một URI.Builder mà bạn có thể sử dụng để xây dựng các chuỗi URI.
6. Để kết nối với internet, bạn phải cấu hình quyền mạng trong tệp manifest Android.

Lớp AsyncTask cho phép bạn chạy các tác vụ trong nền thay vì trên luồng UI:

* Để sử dụng AsyncTask, bạn phải tạo một lớp con. Lớp con ghi đè phương thức doInBackground(Params...). Thường thì lớp con cũng ghi đè phương thức onPostExecute(Result).
* Để bắt đầu một AsyncTask, sử dụng execute().
* Một AsyncTask không thể cập nhật giao diện người dùng nếu activity mà nó kiểm soát dừng lại, ví dụ như do có thay đổi cấu hình thiết bị.

Khi một AsyncTask được thực thi, nó trải qua bốn bước:

1. onPreExecute() chạy trên luồng UI trước khi tác vụ được thực hiện. Bước này thường được sử dụng để thiết lập tác vụ, ví dụ bằng cách hiển thị một thanh tiến trình trong giao diện người dùng.
2. doInBackground(Params...) chạy trên luồng nền ngay sau khi onPreExecute() hoàn thành. Bước này thực hiện các phép toán nền có thể mất nhiều thời gian.
3. onProgressUpdate(Progress...) chạy trên luồng UI sau khi bạn gọi publishProgress(Progress...).
4. onPostExecute(Result) chạy trên luồng UI sau khi phép toán nền hoàn thành. Kết quả của phép toán được truyền cho onPostExecute().

AsyncTaskLoader là phiên bản loader của AsyncTask.

1. AsyncTaskLoader cung cấp phương thức loadInBackground(), phương thức này chạy trên một luồng riêng biệt.
2. Kết quả của loadInBackground() được chuyển đến luồng UI thông qua callback onLoadFinished() của LoaderManager.
3. Để tạo và phân tích các chuỗi JSON, sử dụng các lớp JSON tích hợp sẵn của Java JSONObject và JSONArray.
4. Một AsyncTaskLoader sử dụng một lớp trợ giúp AsyncTask để thực hiện công việc trong nền, ngoài luồng chính.
5. Các thể hiện AsyncTaskLoader được quản lý bởi LoaderManager.
6. LoaderManager cho phép bạn liên kết một Activity mới được tạo với một loader bằng cách sử dụng getSupportLoaderManager().initLoader().

**Các khái niệm liên quan**

Tài liệu về các khái niệm liên quan có trong 7.2: Kết nối internet.

**Học thêm**

Tài liệu cho nhà phát triển Android:

● Connect to the network

● Manage network usage

● Loaders

● AsyncTask

● AsyncTaskLoader

**Bài tập về nhà**

Tạo và chạy một ứng dụng:

Tạo một ứng dụng lấy và hiển thị nội dung của một trang web nằm ở một URL. Ứng dụng hiển thị các thành phần sau:

* Một trường mà người dùng nhập vào một URL.
* Một trường như một menu hoặc spinner cho phép người dùng chọn giao thức (HTTP hoặc HTTPS).
* Một nút thực hiện tác vụ khi người dùng nhấn vào nó.
* Một hiển thị cuộn mã nguồn của trang web tại URL.

Sử dụng AsyncTaskLoader để lấy mã nguồn của trang web tại URL. Bạn cần triển khai một lớp con của AsyncTaskLoader.

Nếu không có kết nối internet khi người dùng nhấn nút, ứng dụng phải hiển thị cho người dùng một phản hồi thích hợp. Ví dụ, ứng dụng có thể hiển thị một thông điệp như "Kiểm tra kết nối internet của bạn và thử lại."

Hiển thị phải chứa một TextView trong một ScrollView hiển thị mã nguồn, nhưng giao diện cụ thể là tùy thuộc vào bạn. Màn hình của bạn có thể trông khác so với các ảnh chụp màn hình bên dưới. Bạn có thể sử dụng menu pop-up, spinner hoặc checkbox để cho phép người dùng chọn HTTP hoặc HTTPS.

**Câu hỏi**

**Câu hỏi 1**: Ứng dụng của bạn cần quyền gì để kết nối với internet?

● android.permission.CONNECTIVITY

● android.permission.INTERNET

● Nó không cần bất kỳ quyền đặc biệt nào, vì tất cả các ứng dụng Android đều được phép kết nối với internet.

**Câu hỏi 2**: Ứng dụng của bạn kiểm tra kết nối internet như thế nào?

Trong manifest:

● yêu cầu quyền ACCESS\_NETWORK\_STATE

● yêu cầu quyền ALL\_NETWORK\_STATE

● yêu cầu quyền NETWORK\_CONNECT

Trong mã:

● Bọc mã để kết nối với internet trong một khối try/catch và bắt lỗi NO\_NETWORK.

● Sử dụng ConnectivityManager để kiểm tra xem có mạng hoạt động trước khi kết nối với mạng.

● Hiển thị một hộp thoại nhắc nhở người dùng đảm bảo rằng kết nối internet có sẵn trước khi họ cố gắng kết nối.

**Câu hỏi 3:** Bạn triển khai phương thức callback loader nào được kích hoạt khi loader hoàn thành thực hiện tác vụ?

● Trong lớp con AsyncTaskLoader. AsyncTaskLoader phải triển khai LoaderManager.LoaderCallbacks.

● Trong Activity hiển thị kết quả của tác vụ. Activity phải triển khai LoaderManager.LoaderCallbacks.

● Trong một lớp Utility mở rộng Object và triển khai LoaderManager.LoaderCallbacks.

Câu hỏi 4: Khi người dùng xoay thiết bị, AsyncTask và AsyncTaskLoader cư xử khác nhau như thế nào nếu chúng đang trong quá trình thực hiện một tác vụ trong nền?

● Một AsyncTask đang chạy bị ngắt kết nối với activity, nhưng vẫn tiếp tục chạy. Một AsyncTaskLoader đang chạy bị ngắt kết nối với activity và dừng lại, bảo tồn tài nguyên hệ thống.

● Một AsyncTask đang chạy bị ngắt kết nối với activity và dừng lại, bảo tồn tài nguyên hệ thống. Một AsyncTaskLoader đang chạy tự động khởi động lại thực hiện tác vụ của nó từ đầu. Activity hiển thị kết quả.

● Một AsyncTask đang chạy bị ngắt kết nối với activity, nhưng vẫn tiếp tục chạy. Một AsyncTaskLoader đang chạy tự động kết nối lại với activity sau khi xoay thiết bị. Activity hiển thị kết quả.

**Câu hỏi 5:** Bạn khởi tạo một AsyncTaskLoader để thực hiện các bước, chẳng hạn như khởi tạo biến, cần phải được thực hiện trước khi loader bắt đầu thực hiện tác vụ nền như thế nào?

● Trong onCreateLoader() trong activity, tạo một thể hiện của lớp con AsyncTaskLoader. Trong constructor của loader, thực hiện các tác vụ khởi tạo.

● Trong onCreateLoader() trong activity, tạo một thể hiện của lớp con AsyncTaskLoader. Trong phương thức init() của loader, thực hiện các tác vụ khởi tạo.

● Trong Activity, triển khai initLoader() để khởi tạo loader.

● Thực hiện các tác vụ khởi tạo cho loader tại đầu của loadInBackground() trong Loader.

**Câu hỏi 6:** Các phương thức nào phải được triển khai bởi một AsyncTaskLoader?

**Nộp ứng dụng của bạn để chấm điểm**

**Hướng dẫn cho người chấm điểm:**

Kiểm tra xem ứng dụng có các tính năng sau không:

* Tệp manifest bao gồm các yêu cầu cho các quyền phù hợp.
* Sử dụng một lớp con của AsyncTaskLoader.
* Phản hồi thích hợp nếu thiết bị không thể kết nối với internet.
* Kết hợp giao thức và trang web để tạo một URL hợp lệ mà ứng dụng sử dụng để kết nối với internet.
* Triển khai các phương thức callback Loader cần thiết.
* Hiển thị kết quả của việc lấy mã nguồn của trang web trong một TextView trong một ScrollView. (Có thể thực hiện trong cùng một activity, hoặc bắt đầu một activity mới.)

### Broadcast receivers

**Giới thiệu**

Broadcasts là các thông điệp mà hệ thống Android và các ứng dụng Android gửi khi có sự kiện xảy ra có thể ảnh hưởng đến chức năng của các ứng dụng hoặc thành phần ứng dụng khác. Ví dụ, hệ thống Android gửi một broadcast hệ thống khi thiết bị khởi động lại, hoặc khi tai nghe được kết nối hoặc ngắt kết nối. Nếu tai nghe có dây bị rút ra, bạn có thể muốn ứng dụng media của mình tạm dừng nhạc. Ứng dụng Android của bạn cũng có thể phát đi các sự kiện, ví dụ khi dữ liệu mới được tải xuống mà có thể thu hút sự quan tâm của một ứng dụng khác. Các sự kiện mà ứng dụng của bạn gửi được gọi là broadcast tùy chỉnh.

Nói chung, bạn có thể sử dụng broadcasts như một hệ thống nhắn tin giữa các ứng dụng và bên ngoài quy trình người dùng thông thường.

Một broadcast được nhận bởi bất kỳ ứng dụng hoặc thành phần ứng dụng nào đã đăng ký một BroadcastReceiver cho hành động đó. BroadcastReceiver là lớp cơ sở cho mã nhận các intent broadcast. Để tìm hiểu thêm về các broadcast receiver, hãy xem tổng quan về Broadcasts và tham chiếu Intent.

**Lưu ý:** Mặc dù lớp Intent được sử dụng để gửi và nhận broadcasts, nhưng cơ chế broadcast của Intent hoàn toàn tách biệt với các intent được sử dụng để khởi động các hoạt động.

Trong thực hành này, bạn sẽ tạo một ứng dụng đáp ứng với sự thay đổi trạng thái sạc của thiết bị. Để làm điều này, ứng dụng của bạn sẽ nhận và phản hồi với một broadcast hệ thống, và nó cũng gửi và nhận một broadcast tùy chỉnh.

**Những gì bạn nên biết trước**

Bạn nên có khả năng:

* Xác định các phần chính của tệp AndroidManifest.xml.
* Tạo các intent ẩn danh.

**Những gì bạn sẽ học**

* Cách kế thừa một BroadcastReceiver và triển khai nó.
* Cách đăng ký nhận các intent broadcast hệ thống.
* Cách tạo và gửi các intent broadcast tùy chỉnh.

**Những gì bạn sẽ làm**

* Kế thừa một BroadcastReceiver để hiển thị một thông báo toast khi nhận được một broadcast.
* Đăng ký receiver của bạn để lắng nghe các broadcast hệ thống.
* Gửi và nhận một intent broadcast tùy chỉnh.

**Tổng quan về ứng dụng**

Ứng dụng PowerReceiver sẽ đăng ký một BroadcastReceiver hiển thị một thông báo toast khi thiết bị được kết nối hoặc ngắt kết nối với nguồn điện. Ứng dụng cũng sẽ gửi và nhận một broadcast tùy chỉnh để hiển thị một thông báo toast khác.

**Nhiệm vụ 1. Thiết lập dự án PowerReceiver**

**1.1 Tạo dự án**

1. Trong Android Studio, tạo một dự án Java mới có tên là PowerReceiver. Chấp nhận các tùy chọn mặc định và sử dụng mẫu Empty Activity.
2. Để tạo một broadcast receiver mới, chọn tên gói trong Android Project View và điều hướng đến File > New > Other > Broadcast Receiver.
3. Đặt tên cho lớp là CustomReceiver. Đảm bảo rằng Java được chọn làm ngôn ngữ nguồn, và rằng các tùy chọn Exported và Enabled đều được chọn. Exported cho phép broadcast receiver của bạn nhận các broadcast từ bên ngoài ứng dụng. Enabled cho phép hệ thống khởi tạo receiver.

**1.2 Đăng ký receiver của bạn cho các broadcast hệ thống**

Một broadcast hệ thống là một thông điệp mà hệ thống Android gửi khi có sự kiện hệ thống xảy ra. Mỗi broadcast hệ thống được bọc trong một đối tượng Intent:

* Trường action của intent chứa thông tin chi tiết về sự kiện như android.intent.action.HEADSET\_PLUG, được gửi khi một tai nghe có dây được kết nối hoặc ngắt kết nối.
* Intent có thể chứa dữ liệu khác về sự kiện trong trường extra, ví dụ như một giá trị boolean cho biết liệu một tai nghe có được kết nối hay không.

Các ứng dụng có thể đăng ký để nhận các broadcast cụ thể. Khi hệ thống gửi một broadcast, nó sẽ định tuyến broadcast đến các ứng dụng đã đăng ký để nhận loại broadcast cụ thể đó.

Một BroadcastReceiver có thể là một receiver tĩnh hoặc receiver động, tùy thuộc vào cách bạn đăng ký nó:

* Để đăng ký một receiver tĩnh, sử dụng phần tử <receiver> trong tệp AndroidManifest.xml. Các receiver tĩnh còn được gọi là receiver được khai báo trong manifest.
* Để đăng ký một receiver động, sử dụng ngữ cảnh ứng dụng hoặc ngữ cảnh hoạt động. Receiver nhận các broadcast miễn là ngữ cảnh đăng ký còn hợp lệ, nghĩa là miễn là ứng dụng hoặc hoạt động tương ứng đang chạy. Các receiver động còn được gọi là receiver được đăng ký theo ngữ cảnh.

Trong ứng dụng này, bạn quan tâm đến hai broadcast hệ thống, ACTION\_POWER\_CONNECTED và ACTION\_POWER\_DISCONNECTED. Hệ thống Android gửi các broadcast này khi nguồn điện của thiết bị được kết nối hoặc ngắt kết nối.

Bắt đầu từ Android 8.0 (API level 26 trở lên), bạn không thể sử dụng các receiver tĩnh để nhận hầu hết các broadcast hệ thống của Android, với một số ngoại lệ. Vì vậy, cho nhiệm vụ này, bạn sẽ sử dụng các receiver động:

1. (Tùy chọn) Điều hướng đến tệp AndroidManifest.xml. Android Studio đã tạo một phần tử <receiver>, nhưng bạn không cần nó, vì bạn không thể sử dụng receiver tĩnh để lắng nghe các broadcast hệ thống về kết nối nguồn. Xóa toàn bộ phần tử <receiver>.
2. Trong MainActivity.java, tạo một đối tượng CustomReceiver như một biến thành viên và khởi tạo nó.

**Tạo bộ lọc intent với các hành động Intent**

Bộ lọc intent xác định các loại intent mà một thành phần có thể nhận. Chúng được sử dụng để lọc các intent dựa trên các giá trị Intent như action và category.

1.Trong MainActivity.java, ở cuối phương thức onCreate(), tạo một đối tượng IntentFilter.

Khi hệ thống nhận một Intent dưới dạng broadcast, nó tìm kiếm các broadcast receiver dựa trên giá trị action được chỉ định trong đối tượng IntentFilter.

2.Trong MainActivity.java, ở cuối onCreate(), thêm các hành động ACTION\_POWER\_CONNECTED và ACTION\_POWER\_DISCONNECTED vào đối tượng IntentFilter.

**Đăng ký và hủy đăng ký receiver**

1. Trong MainActivity.java, ở cuối onCreate(), đăng ký receiver của bạn bằng ngữ cảnh của MainActivity. Receiver của bạn sẽ hoạt động và có khả năng nhận các broadcast miễn là MainActivity đang chạy.
2. Trong MainActivity.java, ghi đè phương thức onDestroy() và hủy đăng ký receiver của bạn. Để tiết kiệm tài nguyên hệ thống và tránh rò rỉ, các receiver động phải được hủy đăng ký khi không còn cần thiết hoặc trước khi hoạt động hoặc ứng dụng tương ứng bị hủy, tùy thuộc vào ngữ cảnh được sử dụng.

**1.3 Triển khai onReceive() trong BroadcastReceiver của bạn**

Khi một broadcast receiver chặn một broadcast mà nó đã đăng ký, đối tượng Intent sẽ được gửi đến phương thức onReceive() của receiver.

Trong CustomReceiver.java, bên trong phương thức onReceive(), bạn thực hiện các bước sau:

1. Xóa toàn bộ phần triển khai của phương thức onReceive(), bao gồm mã UnsupportedOperationException.
2. Lấy hành động Intent từ tham số intent và lưu trữ nó trong một biến kiểu String có tên là intentAction.
3. Tạo một câu lệnh switch với chuỗi intentAction. (Trước khi sử dụng intentAction, hãy kiểm tra xem nó có khác null không.) Hiển thị một thông báo toast khác nhau cho mỗi hành động mà receiver của bạn đã đăng ký.
4. Sau câu lệnh switch, thêm mã để hiển thị một thông báo toast trong một khoảng thời gian ngắn.
5. Chạy ứng dụng của bạn. Sau khi ứng dụng đang chạy, kết nối hoặc ngắt kết nối nguồn điện của thiết bị. Một thông báo Toast sẽ được hiển thị mỗi khi bạn kết nối hoặc ngắt kết nối nguồn điện, miễn là Activity của bạn đang chạy.

**Lưu ý:** Nếu bạn đang sử dụng trình giả lập, hãy thay đổi trạng thái kết nối nguồn bằng cách chọn biểu tượng ba chấm cho menu. Chọn **Battery** trong thanh bên trái, sau đó sử dụng cài đặt kết nối Charger.

**Nhiệm vụ 2. Gửi và nhận một broadcast tùy chỉnh**

Ngoài việc phản hồi các broadcast hệ thống, ứng dụng của bạn có thể gửi và nhận các broadcast tùy chỉnh. Sử dụng một broadcast tùy chỉnh khi bạn muốn ứng dụng của mình thực hiện một hành động mà không cần khởi động một hoạt động, ví dụ như khi bạn muốn thông báo cho các ứng dụng khác rằng dữ liệu đã được tải xuống thiết bị.

Android cung cấp ba cách để ứng dụng của bạn gửi các broadcast tùy chỉnh:

* Broadcast bình thường là không đồng bộ. Các receiver của broadcast bình thường chạy theo một thứ tự không xác định, thường là cùng một lúc. Để gửi một broadcast bình thường, tạo một broadcast intent và truyền nó cho sendBroadcast(Intent).
* Broadcast cục bộ được gửi đến các receiver nằm trong cùng một ứng dụng với người gửi. Để gửi một broadcast cục bộ, tạo một broadcast intent và truyền nó cho LocalBroadcastManager.sendBroadcast.
* Broadcast có thứ tự được gửi đến một receiver tại một thời điểm. Khi mỗi receiver thực thi, nó có thể truyền kết quả cho receiver tiếp theo hoặc có thể hủy broadcast để broadcast không được chuyển đến các receiver khác. Để gửi một broadcast có thứ tự, tạo một broadcast intent và truyền nó cho sendOrderedBroadcast(Intent, String).

Thực hành này không đề cập đến các broadcast có thứ tự, nhưng để biết thêm thông tin về chúng, hãy xem Gửi broadcasts.

Thông điệp broadcast được bọc trong một đối tượng Intent. Chuỗi hành động Intent phải cung cấp cú pháp tên gói Java của ứng dụng và xác định duy nhất sự kiện broadcast.

Đối với một broadcast tùy chỉnh, bạn định nghĩa hành động Intent của riêng mình (một chuỗi duy nhất). Bạn có thể tạo các đối tượng Intent với các hành động tùy chỉnh và phát chúng từ ứng dụng của mình bằng một trong các phương pháp trên. Các broadcast sẽ được nhận bởi các ứng dụng đã đăng ký một BroadcastReceiver cho hành động đó.

Trong nhiệm vụ này, bạn thêm một nút vào hoạt động của mình để gửi một broadcast intent cục bộ. Receiver của bạn sẽ đăng ký broadcast intent và hiển thị kết quả trong một thông báo toast.

**2.1 Định nghĩa chuỗi hành động broadcast tùy chỉnh của bạn**

Cả người gửi và người nhận của một broadcast tùy chỉnh phải đồng ý về một chuỗi hành động duy nhất cho Intent được phát. Một cách phổ biến để tạo ra một chuỗi hành động duy nhất là thêm tên hành động của bạn với tên gói ứng dụng.

Một trong những cách đơn giản nhất để lấy tên gói của ứng dụng là sử dụng BuildConfig.APPLICATION\_ID, trả về giá trị của thuộc tính applicationId từ tệp build.gradle cấp module của bạn.

1. Tạo một biến thành viên hằng số trong cả lớp MainActivity và CustomReceiver. Bạn sẽ sử dụng biến này làm hành động Intent cho broadcast.

**Quan trọng:** Mặc dù intents được sử dụng cả để gửi broadcasts và khởi động activities với startActivity(Intent), nhưng các hành động này hoàn toàn không liên quan. Các broadcast receiver không thể thấy hoặc chặn một Intent được sử dụng để khởi động một activity. Tương tự, khi bạn phát một Intent, bạn không thể sử dụng Intent đó để tìm hoặc khởi động một activity.

**2.2 Thêm nút "Gửi Broadcast Tùy Chỉnh"**

1. Trong tệp bố cục activity\_main.xml, thay thế TextView "Hello World" bằng một Button có các thuộc tính sau:
2. Trích xuất tài nguyên chuỗi.

Phương thức sendCustomBroadcast() sẽ là trình xử lý sự kiện nhấp cho nút. Để tạo một stub cho sendCustomBroadcast() trong Android Studio:

1. Nhấp vào tên phương thức sendCustomBroadcast được làm nổi bật màu vàng. Một bóng đèn đỏ xuất hiện bên trái.
2. Nhấp vào bóng đèn đỏ và chọn **Create 'sendCustomBroadcast(View)' in 'MainActivity'.**

**2.3 Triển khai sendCustomBroadcast()**

Vì broadcast này chỉ được sử dụng bởi ứng dụng của bạn, hãy sử dụng LocalBroadcastManager để quản lý broadcast. LocalBroadcastManager là một lớp cho phép bạn đăng ký và gửi các broadcast mà các thành phần trong ứng dụng của bạn quan tâm.

Bằng cách giữ các broadcast cục bộ, bạn đảm bảo rằng dữ liệu ứng dụng của bạn không được chia sẻ với các ứng dụng Android khác. Các broadcast cục bộ giữ thông tin của bạn an toàn hơn và duy trì hiệu quả hệ thống.

Trong MainActivity.java, bên trong phương thức sendCustomBroadcast(), thực hiện các bước sau:

1. Tạo một Intent mới, với chuỗi hành động tùy chỉnh của bạn làm đối số.
2. Sau khi khai báo Intent tùy chỉnh, gửi broadcast bằng lớp LocalBroadcastManager.

**2.4 Đăng ký và hủy đăng ký broadcast tùy chỉnh của bạn**

Việc đăng ký cho một broadcast cục bộ tương tự như việc đăng ký cho một broadcast hệ thống, điều này bạn thực hiện bằng cách sử dụng một receiver động. Đối với các broadcast được gửi bằng LocalBroadcastManager, việc đăng ký tĩnh trong manifest là không được phép.

Nếu bạn đăng ký một broadcast receiver động, bạn phải hủy đăng ký receiver khi không còn cần thiết. Trong ứng dụng của bạn, receiver chỉ cần phản hồi lại broadcast tùy chỉnh khi ứng dụng đang chạy, vì vậy bạn có thể đăng ký hành động trong onCreate() và hủy đăng ký trong onDestroy().

1. Trong MainActivity.java, bên trong phương thức onCreate(), lấy một thể hiện của LocalBroadcastManager và đăng ký receiver của bạn với hành động Intent tùy chỉnh:
2. Trong MainActivity.java, bên trong phương thức onDestroy(), hủy đăng ký receiver của bạn khỏi LocalBroadcastManager.

**2.5 Phản hồi lại broadcast tùy chỉnh**

1. Trong CustomReceiver.java, bên trong phương thức onReceive(), thêm một câu lệnh case khác trong khối switch cho hành động Intent tùy chỉnh. Sử dụng "Custom Broadcast Received" làm văn bản cho thông báo toast.
2. Trích xuất tài nguyên chuỗi.
3. Chạy ứng dụng của bạn và nhấn nút **Send Custom Broadcast** để gửi một broadcast tùy chỉnh. Receiver của bạn (CustomReceiver) sẽ hiển thị một thông báo toast.

Vậy là xong! Ứng dụng của bạn đã gửi một broadcast tùy chỉnh và có khả năng nhận cả các broadcast hệ thống và tùy chỉnh.

**Mã giải pháp**

Dự án Android Studio: PowerReceiver

**Thử thách lập trình**

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

**Thử thách:** Nếu bạn đang phát triển một ứng dụng phát nhạc, ứng dụng của bạn có thể cần phát hoặc tạm dừng nhạc khi người dùng kết nối hoặc ngắt kết nối tai nghe có dây. Để triển khai chức năng này, bạn cần một broadcast receiver phản hồi các sự kiện tai nghe có dây. Triển khai một broadcast receiver hiển thị thông báo toast khi tai nghe có dây được kết nối hoặc ngắt kết nối.

**Gợi ý**: Bạn cần đăng ký cho hành động ACTION\_HEADSET\_PLUG. Vì đây là một hành động broadcast hệ thống, bạn không thể đăng ký cho nó một cách tĩnh. Thay vào đó, hãy đăng ký receiver của bạn một cách động bằng cách sử dụng ngữ cảnh với Context.registerReceiver().

Bạn cũng phải hủy đăng ký receiver khi không còn cần thiết.

**Tóm tắt**

* Broadcast receiver là các thành phần cơ bản của một ứng dụng Android.
* Các broadcast receiver có thể nhận các broadcast được gửi bởi hệ thống hoặc bởi các ứng dụng.
* Intent được sử dụng trong cơ chế broadcast hoàn toàn khác với các intent được sử dụng để khởi động activities.
* Để xử lý Intent đến liên quan đến một broadcast, bạn kế thừa lớp BroadcastReceiver và triển khai onReceive().
* Bạn có thể đăng ký một broadcast receiver trong tệp manifest Android hoặc lập trình.
* Các broadcast cục bộ là riêng tư cho ứng dụng của bạn. Để đăng ký và gửi các broadcast cục bộ, hãy sử dụng LocalBroadcastManager. Các broadcast cục bộ không liên quan đến giao tiếp giữa các tiến trình, điều này làm cho chúng hiệu quả. Sử dụng các broadcast cục bộ cũng có thể bảo vệ ứng dụng của bạn khỏi một số vấn đề bảo mật, vì dữ liệu vẫn nằm trong ứng dụng của bạn.
* Để tạo các tên hành động Intent duy nhất cho các broadcast, một thực hành phổ biến là thêm tên hành động với tên gói của bạn.
* Nếu ứng dụng của bạn nhằm vào API level 26 trở lên, bạn không thể sử dụng manifest để khai báo một receiver cho hầu hết các broadcast ẩn danh. (Các broadcast ẩn danh, bao gồm hầu hết các broadcast hệ thống, là các broadcast không nhắm mục tiêu đến ứng dụng của bạn.) Một số broadcast ẩn danh là ngoại lệ. Tuy nhiên, bạn có thể sử dụng các receiver động để nhận tất cả các broadcast.

**Khái niệm liên quan**

Tài liệu về khái niệm liên quan có trong phần 7.3: Broadcasts.

**Tìm hiểu thêm**

Tài liệu dành cho nhà phát triển Android:

● Intents and Intent Filters

● Broadcasts overview

● BroadcastReceiver

● Implicit Broadcast Exceptions

**Bài tập về nhà**

**Cập nhật ứng dụng**

Mở rộng ứng dụng PowerReceiver mà bạn đã tạo trong thực hành.

1. Gửi dữ liệu bổ sung đến receiver broadcast tùy chỉnh cục bộ của bạn. Để làm điều này, tạo một số nguyên ngẫu nhiên giữa 1 và 20. Thêm số này vào trường extra của Intent trước khi gửi broadcast tùy chỉnh cục bộ.
2. Trong receiver của bạn, trích xuất dữ liệu số nguyên từ Intent. Trong thông báo toast, hiển thị bình phương của số ngẫu nhiên.

**Trả lời các câu hỏi**

**Câu hỏi 1**

Broadcast hệ thống là gì?

● Một thông điệp mà ứng dụng của bạn gửi và nhận khi một sự kiện quan tâm xảy ra trong ứng dụng.

● Một thông điệp được gửi từ một ứng dụng đến một thành phần khác của cùng một ứng dụng.

● Một thông điệp mà hệ thống Android gửi khi một sự kiện hệ thống xảy ra.

● Một thông điệp mà hệ thống Android nhận khi một sự kiện quan tâm xảy ra trong ứng dụng của bạn.

**Câu hỏi 2**

Cặp phương thức nào bạn sử dụng để đăng ký và hủy đăng ký receiver broadcast của bạn một cách động?

● registerBroadcast() và unRegisterBroadcast()

● registerComponentCallbacks() và unRegisterComponentCallbacks()

● registerBroadcastReceiver() và unRegisterBroadcastReceiver()

● registerReceiver() và unRegisterReceiver()

**Câu hỏi 3**

Cái nào trong số sau đây là đúng?

● Các broadcast receiver không thể thấy hoặc chặn các intent được sử dụng để khởi động một activity.

● Khi sử dụng một intent broadcast, bạn không thể tìm hoặc khởi động một activity.

● Bạn có thể sử dụng một intent broadcast để khởi động một activity.

● Bạn có thể nhận được intent được sử dụng để khởi động activity trong broadcast receiver của bạn.

**Câu hỏi 4**

Lớp nào được sử dụng để giảm thiểu các rủi ro bảo mật của các broadcast receiver khi các broadcast không vượt qua ứng dụng (tức là khi các broadcast được gửi và nhận bởi cùng một ứng dụng)?

● SecureBroadcast

● LocalBroadcastManager

● OrderedBroadcast

● SecureBroadcastManager

**Nộp ứng dụng của bạn để chấm điểm**

**Hướng dẫn cho người chấm điểm**

Kiểm tra xem ứng dụng có các tính năng sau:

● Ứng dụng tạo ra một số nguyên ngẫu nhiên và gửi số nguyên đó như một dữ liệu bổ sung trong Intent trong LocalBroadcast.

● Trong phương thức onReceive() của receiver, dữ liệu số nguyên ngẫu nhiên được trích xuất từ Intent, và bình phương của số nguyên được hiển thị trong một thông báo toast.

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

### Thông báo

**Giới thiệu**

Đôi khi bạn muốn ứng dụng của mình hiển thị thông tin cho người dùng ngay cả khi ứng dụng không đang chạy ở chế độ nền. Ví dụ, bạn có thể muốn thông báo cho người dùng rằng có nội dung mới hoặc đội thể thao yêu thích của họ vừa ghi bàn trong một trận đấu. Khung thông báo của Android cung cấp cách để ứng dụng thông báo cho người dùng ngay cả khi ứng dụng không ở chế độ foreground.

Một thông báo là một thông điệp mà ứng dụng của bạn hiển thị cho người dùng bên ngoài giao diện người dùng bình thường của ứng dụng. Các thông báo xuất hiện dưới dạng biểu tượng trong khu vực thông báo của thiết bị, nằm trong thanh trạng thái. Để xem chi tiết của một thông báo, người dùng mở thùng thông báo (notification drawer), ví dụ bằng cách kéo xuống thanh trạng thái. Khu vực thông báo và thùng thông báo là các khu vực do hệ thống kiểm soát mà người dùng có thể xem bất cứ lúc nào.

Trên các thiết bị chạy Android 8.0 trở lên, khi ứng dụng của bạn có một thông báo mới để hiển thị cho người dùng, biểu tượng ứng dụng của bạn sẽ tự động hiển thị một badge. (Badge cũng được gọi là notification dots). Khi người dùng nhấn và giữ biểu tượng ứng dụng, thông báo sẽ xuất hiện phía trên biểu tượng ứng dụng, như được hiển thị trong ảnh chụp màn hình dưới đây

Trong thực hành này, bạn sẽ tạo một ứng dụng kích hoạt một thông báo khi người dùng nhấn một nút trong ứng dụng của bạn. Người dùng có thể cập nhật thông báo hoặc hủy nó.

**Những gì bạn nên biết trước**

Bạn nên có khả năng:

* Triển khai phương thức onClick() cho các nút.
* Tạo các implicit intents.
* Gửi các custom broadcasts.
* Sử dụng các broadcast receivers.

**Những gì bạn sẽ học**

* Cách tạo một thông báo bằng cách sử dụng notification builder.
* Cách sử dụng pending intents để phản hồi các hành động thông báo.
* Cách cập nhật hoặc hủy bỏ các thông báo hiện có.

**Những gì bạn sẽ làm**

* Tạo một ứng dụng gửi thông báo khi người dùng nhấn nút trong ứng dụng.
* Cập nhật thông báo từ một nút trong ứng dụng và từ một nút hành động nằm trong thông báo.

**Tổng quan về ứng dụng**

Notify Me! là một ứng dụng cho phép người dùng kích hoạt, cập nhật và hủy bỏ một thông báo bằng cách sử dụng ba nút được hiển thị trong ảnh chụp màn hình dưới đây. Trong khi bạn tạo ứng dụng, bạn sẽ thực hành với các kiểu thông báo, hành động và độ ưu tiên.

**Nhiệm vụ 1: Tạo một thông báo cơ bản**

**1.1 Tạo dự án**

1. Trong Android Studio, tạo một dự án mới có tên "Notify Me!". Chấp nhận các tùy chọn mặc định và sử dụng mẫu Empty Activity.
2. Trong tệp activity\_main.xml, thay thế TextView mặc định bằng một nút có các thuộc tính sau:

Thực hiện các bước sau trong tệp MainActivity.java:

1.Tạo một biến thành viên cho nút **Notify Me!.**

1.Tạo một phương thức stub cho phương thức sendNotification().

2.Trong phương thức onCreate(), khởi tạo nút Notify Me! và tạo một onClickListener cho nó. Gọi sendNotification() từ phương thức onClick.

**1.2 Tạo một kênh thông báo**

Trong ứng dụng Cài đặt trên thiết bị chạy Android, người dùng có thể điều chỉnh các thông báo mà họ nhận. Bắt đầu từ Android 8.0 (API level 26), mã của bạn có thể gán mỗi thông báo của ứng dụng vào một kênh thông báo có thể tùy chỉnh bởi người dùng:

* Mỗi kênh thông báo đại diện cho một loại thông báo.
* Trong mã của bạn, bạn có thể nhóm nhiều thông báo trong mỗi kênh thông báo.
* Đối với mỗi kênh thông báo, ứng dụng của bạn thiết lập hành vi cho kênh, và hành vi này sẽ được áp dụng cho tất cả các thông báo trong kênh. Ví dụ, ứng dụng của bạn có thể thiết lập thông báo trong một kênh để phát âm thanh, nhấp nháy đèn hoặc rung.
* Bất kỳ hành vi nào mà ứng dụng của bạn thiết lập cho một kênh thông báo, người dùng đều có thể thay đổi hành vi đó, và người dùng có thể tắt thông báo của ứng dụng hoàn toàn.

Trên các thiết bị chạy Android 8.0 (API level 26) hoặc cao hơn, các kênh thông báo mà bạn tạo trong ứng dụng sẽ xuất hiện dưới dạng **Categories** trong mục Thông báo ứng dụng trong ứng dụng Cài đặt của thiết bị. Ví dụ, trong ảnh chụp màn hình dưới đây của một thiết bị chạy Android 8.0, ứng dụng Notify Me! có một kênh thông báo, Mascot Notification.

Khi ứng dụng của bạn nhắm đến Android 8.0 (API level 26), để hiển thị thông báo cho người dùng, bạn phải triển khai ít nhất một kênh thông báo. Để hiển thị thông báo trên các thiết bị thấp hơn, bạn không bắt buộc phải triển khai các kênh thông báo. Tuy nhiên, đây là thực hành tốt để luôn làm như sau:

* Nhắm đến SDK mới nhất có sẵn.
* Kiểm tra phiên bản SDK của thiết bị trong mã của bạn. Nếu phiên bản SDK là 26 hoặc cao hơn, hãy xây dựng các kênh thông báo.

Nếu targetSdkVersion của bạn được đặt ở mức 25 hoặc thấp hơn, khi ứng dụng của bạn chạy trên Android 8.0 (API level 26) hoặc cao hơn, nó sẽ hoạt động giống như trên các thiết bị chạy Android 7.1 (API level 25) hoặc thấp hơn.

Tạo một kênh thông báo:

1. Trong MainActivity, tạo một hằng số cho ID của kênh thông báo. Mỗi kênh thông báo phải được liên kết với một ID duy nhất trong gói của bạn. Bạn sẽ sử dụng ID kênh này sau này để gửi thông báo của bạn.
2. Hệ thống Android sử dụng lớp NotificationManager để gửi thông báo đến người dùng. Trong MainActivity.java, tạo một biến thành viên để lưu trữ đối tượng NotificationManager.
3. Trong MainActivity.java, tạo phương thức createNotificationChannel() và khởi tạo NotificationManager bên trong phương thức này.
4. Tạo một kênh thông báo trong phương thức createNotificationChannel(). Vì các kênh thông báo chỉ có sẵn trong API 26 và cao hơn, hãy thêm một điều kiện để kiểm tra phiên bản API của thiết bị.
5. Bên trong câu lệnh if, tạo một đối tượng NotificationChannel và sử dụng PRIMARY\_CHANNEL\_ID làm ID kênh.
6. Đặt tên cho kênh. Tên này sẽ được hiển thị dưới các **Categories** thông báo trong ứng dụng Cài đặt có thể nhìn thấy của thiết bị.
7. Đặt độ quan trọng thành IMPORTANCE\_HIGH. (Để xem danh sách đầy đủ các hằng số độ quan trọng thông báo, hãy tham khảo tài liệu của NotificationManager.)
8. Trong createNotificationChannel(), bên trong câu lệnh if, cấu hình các cài đặt ban đầu cho đối tượng notificationChannel. Ví dụ, bạn có thể thiết lập màu sắc đèn thông báo, kích hoạt rung, và thiết lập mô tả sẽ hiển thị trong ứng dụng Cài đặt của thiết bị. Bạn cũng có thể cấu hình âm thanh cảnh báo cho thông báo.

**1.3 Xây dựng thông báo đầu tiên của bạn**

Thông báo được tạo bằng cách sử dụng lớp NotificationCompat.Builder, cho phép bạn thiết lập nội dung và hành vi của thông báo. Một thông báo có thể chứa các thành phần sau:

* Biểu tượng (bắt buộc), bạn thiết lập trong mã bằng phương thức setSmallIcon().
* Tiêu đề (tùy chọn), bạn thiết lập bằng setContentTitle().
* Văn bản chi tiết (tùy chọn), bạn thiết lập bằng setContentText().

Để tạo biểu tượng thông báo cần thiết:

1. Trong Android Studio, đi đến **File > New > Image Asset.**
2. Từ danh sách thả xuống **Icon Type**, chọn **Notification Icons.**
3. Nhấp vào biểu tượng bên cạnh mục **Clip Art** để chọn một biểu tượng Material Design cho thông báo của bạn. Đối với ứng dụng này, sử dụng biểu tượng Android.
4. Đổi tên tài nguyên thành ic\_android và nhấp **Next** và **Finish**. Điều này tạo ra các tệp hình ảnh với các độ phân giải khác nhau cho các phiên bản API khác nhau.

Để xây dựng thông báo và hiển thị nó:

1. Bạn cần liên kết thông báo với một notification ID để mã của bạn có thể cập nhật hoặc hủy thông báo trong tương lai. Trong MainActivity.java, tạo một hằng số cho ID thông báo.
2. Trong MainActivity.java, ở cuối phương thức onCreate(), gọi createNotificationChannel(). Nếu bạn bỏ qua bước này, ứng dụng của bạn sẽ bị sập!
3. Trong MainActivity.java, tạo một phương thức trợ giúp có tên getNotificationBuilder(). Bạn sẽ sử dụng getNotificationBuilder() sau này trong đối tượng NotificationCompat.Builder. Android Studio sẽ hiển thị lỗi liên quan đến việc thiếu lệnh trả về, nhưng bạn sẽ sửa lỗi đó sớm.
4. Bên trong phương thức getNotificationBuilder(), tạo và khởi tạo trình xây dựng thông báo. Đối với ID kênh thông báo, sử dụng PRIMARY\_CHANNEL\_ID. Nếu có lỗi pop-up hiển thị, hãy đảm bảo rằng lớp NotificationCompat được nhập từ Thư viện hỗ trợ v4.
5. Bên trong phương thức getNotificationBuilder(), thêm tiêu đề, văn bản và biểu tượng vào trình xây dựng, như được hiển thị dưới đây. Cuối cùng, trả về đối tượng Builder.

Bây giờ bạn có thể hoàn thành phương thức sendNotification() để gửi thông báo:

1. Trong MainActivity.java, bên trong phương thức sendNotification(), sử dụng getNotificationBuilder() để lấy đối tượng Builder.
2. Gọi notify() trên NotificationManager.

1. Chạy ứng dụng của bạn. Nút **Notify Me!** sẽ phát ra một thông báo, và biểu tượng sẽ xuất hiện trong thanh trạng thái. Tuy nhiên, thông báo thiếu một tính năng thiết yếu: không có gì xảy ra khi bạn nhấn vào nó. Bạn sẽ thêm chức năng đó trong nhiệm vụ tiếp theo.

**1.4 Thêm một content intent và hủy thông báo**

Các content intent cho thông báo tương tự như các intent mà bạn đã sử dụng trong suốt khóa học này. Các content intent có thể là các explicit intent để khởi động một activity, implicit intent để thực hiện một hành động, hoặc broadcast intent để thông báo cho hệ thống về một sự kiện hệ thống hoặc sự kiện tùy chỉnh.

Sự khác biệt chính với một Intent được sử dụng cho một thông báo là Intent phải được bọc trong một PendingIntent. PendingIntent cho phép hệ thống thông báo Android thực hiện hành động được chỉ định thay mặt cho mã của bạn.

Trong bước này, bạn cập nhật ứng dụng của mình để khi người dùng nhấn vào thông báo, ứng dụng của bạn sẽ gửi một content intent khởi động MainActivity. (Nếu ứng dụng đang mở và hoạt động, việc nhấn vào thông báo sẽ không có tác dụng.)

1.Trong MainActivity.java, ở đầu phương thức getNotificationBuilder(), tạo một phương thức intent rõ ràng để khởi động MainActivity.

2.Bên trong getNotificationBuilder(), sau khi khai báo notificationIntent, sử dụng phương thức getActivity() để lấy một PendingIntent. Truyền hằng số ID thông báo cho requestCode và sử dụng cờ FLAG\_UPDATE\_CURRENT.

Bằng cách sử dụng PendingIntent để giao tiếp với một ứng dụng khác, bạn đang yêu cầu ứng dụng đó thực hiện một số mã đã được định nghĩa trước vào một thời điểm nào đó trong tương lai. Giống như ứng dụng khác có thể thực hiện một hành động thay mặt cho ứng dụng của bạn.

3.Sử dụng phương thức setContentIntent() từ lớp NotificationCompat.Builder để thiết lập content intent. Bên trong getNotificationBuilder(), gọi setContentIntent() trong mã đang xây dựng thông báo. Đồng thời, thiết lập auto-cancel thành true.

Thiết lập auto-cancel thành true sẽ đóng thông báo khi người dùng nhấn vào nó.

1. Chạy ứng dụng. Nhấn nút **Notify Me!** để gửi thông báo. Nhấn nút Home. Bây giờ xem thông báo và nhấn vào nó. Lưu ý rằng ứng dụng mở lại tại MainActivity.
2. Nếu bạn đang chạy ứng dụng trên một thiết bị hoặc trình giả lập có API 26 hoặc cao hơn, nhấn nút Home và mở trình khởi động ứng dụng. Lưu ý biểu tượng badge (dấu chấm thông báo) trên biểu tượng ứng dụng.

Trong ảnh chụp màn hình trên:

1. Thông báo trong thanh trạng thái.
2. Dấu chấm thông báo trên biểu tượng ứng dụng (chỉ trên API 26 hoặc cao hơn).

Khi người dùng chạm và giữ biểu tượng ứng dụng, một pop-up hiển thị thông báo cùng với biểu tượng.

Nếu bạn đang chạy trên một thiết bị hoặc trình giả lập có API 26 hoặc cao hơn, đây là cách xem kênh thông báo mà bạn đã tạo:

1. Mở ứng dụng Cài đặt của thiết bị.
2. Trong thanh tìm kiếm, nhập tên ứng dụng của bạn, "Notify Me!".
3. Mở **Notify Me! > App Notifications > Mascot Notifications**. Sử dụng cài đặt này để tùy chỉnh kênh thông báo. Mô tả của kênh thông báo được hiển thị ở dưới cùng của màn hình.

**1.5 Thêm độ ưu tiên và các tùy chọn mặc định cho thông báo của bạn để tương thích ngược**

**Lưu ý:** Nhiệm vụ này là cần thiết cho các thiết bị chạy Android 7.1 hoặc thấp hơn, mà hầu hết các thiết bị Android đều sử dụng. Đối với các thiết bị chạy Android 8.0 trở lên, bạn sử dụng các kênh thông báo để thêm độ ưu tiên và các tùy chọn mặc định cho thông báo, nhưng việc cung cấp tính tương thích ngược là thực hành tốt.

Khi người dùng nhấn nút **Notify Me!** trong ứng dụng của bạn, thông báo sẽ được phát ra, nhưng điều duy nhất mà người dùng thấy là biểu tượng trong thanh thông báo. Để thu hút sự chú ý của người dùng, hãy thiết lập các tùy chọn mặc định cho thông báo.

Độ ưu tiên là một giá trị số nguyên từ PRIORITY\_MIN (-2) đến PRIORITY\_MAX (2). Các thông báo có độ ưu tiên cao hơn sẽ được sắp xếp ở trên các thông báo có độ ưu tiên thấp hơn trong thùng thông báo. Các thông báo có độ ưu tiên HIGH hoặc MAX sẽ được gửi dưới dạng thông báo "heads up", rơi xuống trên màn hình hoạt động của người dùng. Không nên đặt tất cả các thông báo của bạn thành độ ưu tiên MAX, vì vậy hãy sử dụng MAX một cách tiết kiệm.

1.Bên trong phương thức getNotificationBuilder(), đặt độ ưu tiên của thông báo thành HIGH bằng cách thêm dòng sau vào đối tượng trình xây dựng thông báo:

1.Đặt âm thanh, rung, và mẫu màu LED cho thông báo của bạn (nếu thiết bị của người dùng có đèn LED) thành các giá trị mặc định.

Bên trong getNotificationBuilder(), thêm dòng sau vào đối tượng notifyBuilder:

1.Để xem các thay đổi, thoát ứng dụng và chạy lại từ Android Studio. Nếu bạn không thấy các thay đổi, hãy gỡ cài đặt ứng dụng và cài đặt lại.

**Lưu ý:** Thông báo có độ ưu tiên cao sẽ không rơi xuống trước màn hình hoạt động trừ khi cả độ ưu tiên và các tùy chọn mặc định đều được thiết lập. Chỉ đặt độ ưu tiên một mình là không đủ.

**Nhiệm vụ 2: Cập nhật hoặc hủy thông báo**

Sau khi ứng dụng của bạn phát ra một thông báo, thật hữu ích cho ứng dụng của bạn có thể cập nhật hoặc hủy thông báo nếu thông tin thay đổi hoặc trở nên không liên quan.

**2.1 Thêm nút cập nhật và nút hủy**

1. Trong tệp activity\_main.xml, tạo hai bản sao của nút **Notify Me!.** Trong trình chỉnh sửa thiết kế, ràng buộc các nút mới với nhau và với bố cục cha, để chúng không chồng lên nhau.
2. Thay đổi thuộc tính android:text trong các nút mới thành "Update Me!" và "Cancel Me!".
3. Thay đổi thuộc tính android:id cho các nút thành update và cancel.

1.Trích xuất tất cả các chuỗi văn bản vào strings.xml.

Thực hiện các bước sau trong tệp MainActivity.java:

1. 1.Thêm một biến thành viên cho mỗi nút mới.
2. Ở cuối phương thức onCreate(), khởi tạo các biến nút và thiết lập các onClick listener của chúng. Nếu Android Studio báo lỗi, hãy xây dựng lại dự án của bạn.
3. Tạo các phương thức để cập nhật và hủy thông báo. Các phương thức không nhận tham số và trả về void.
4. Trong phương thức onCreate(), gọi updateNotification() trong phương thức onClick của nút cập nhật. Trong phương thức onClick của nút hủy, gọi cancelNotification().

**2.2 Triển khai các phương thức hủy và cập nhật thông báo**

Để hủy một thông báo, gọi cancel() trên NotificationManager, truyền vào ID thông báo.

1. Trong MainActivity.java, bên trong phương thức cancelNotification(), thêm dòng sau:
2. Chạy ứng dụng.
3. Nhấn nút **Notify Me!** để gửi thông báo. Lưu ý rằng biểu tượng thông báo xuất hiện trong thanh trạng thái.
4. Nhấn nút **Cancel Me!.** Thông báo nên được hủy.

Cập nhật một thông báo phức tạp hơn so với việc hủy thông báo. Các thông báo Android đi kèm với các kiểu có thể cô đọng thông tin. Ví dụ, ứng dụng Gmail sử dụng thông báo InboxStyle nếu người dùng có nhiều tin nhắn chưa đọc, điều này cô đọng thông tin thành một thông báo duy nhất.

Trong ví dụ này, bạn sẽ cập nhật thông báo của mình để sử dụng BigPictureStyle, cho phép bạn bao gồm một hình ảnh trong thông báo.

1. Tải xuống hình ảnh này để sử dụng trong thông báo của bạn và đổi tên thành mascot\_1. Nếu bạn sử dụng hình ảnh của riêng mình, hãy đảm bảo rằng tỷ lệ khung hình của nó là 2:1 và chiều rộng không quá 450 dp.
2. Đặt hình ảnh mascot\_1 vào thư mục res/drawable.
3. Trong MainActivity.java, bên trong phương thức updateNotification(), chuyển đổi drawable của bạn thành bitmap.
4. Bên trong updateNotification(), sử dụng phương thức getNotificationBuilder() để lấy đối tượng NotificationCompat.Builder.
5. Bên trong updateNotification(), sau khi khai báo notifyBuilder, thay đổi kiểu của thông báo và thiết lập hình ảnh cùng tiêu đề:

**Lưu ý:** BigPictureStyle là một lớp con của NotificationCompat.Style cung cấp các bố cục thay thế cho thông báo. Để xem các lớp con đã được định nghĩa khác, hãy xem tài liệu.

1. Bên trong updateNotification(), sau khi thiết lập kiểu thông báo, xây dựng thông báo và gọi notify() trên NotificationManager. Truyền vào cùng một ID thông báo như trước.
2. Chạy ứng dụng của bạn. Nhấn nút cập nhật và kiểm tra lại thông báo—thông báo bây giờ có hình ảnh và tiêu đề đã cập nhật! Để thu nhỏ lại kiểu thông báo bình thường, hãy pinch thông báo mở rộng.

**2.3 Chuyển đổi trạng thái nút**

Trong ứng dụng này, người dùng có thể bị nhầm lẫn vì trạng thái của thông báo không được theo dõi trong hoạt động. Ví dụ, người dùng có thể nhấn vào nút Cancel Me! khi không có thông báo nào đang hiển thị.

Bạn có thể khắc phục điều này bằng cách bật và tắt các nút tùy thuộc vào trạng thái của thông báo:

* Khi ứng dụng lần đầu tiên được chạy, chỉ có nút Notify Me! được bật, vì chưa có thông báo nào để cập nhật hoặc hủy.
* Sau khi một thông báo được gửi, nút hủy và cập nhật sẽ được bật, và nút thông báo sẽ bị tắt, vì thông báo đã được gửi.
* Sau khi thông báo được cập nhật, nút cập nhật và nút thông báo sẽ bị tắt, chỉ còn nút hủy được bật.
* Nếu thông báo bị hủy, các nút sẽ trở lại trạng thái ban đầu, chỉ có nút thông báo được bật.

Để chuyển đổi trạng thái nút cho tất cả các nút, hãy thực hiện các bước sau trong MainActivity.java:

1.Thêm một phương thức tiện ích có tên setNotificationButtonState() để chuyển đổi trạng thái của các nút.

1.Ở cuối mỗi phương thức liên quan, thêm một lệnh gọi đến setNotificationButtonState() để bật và tắt các nút cho phù hợp:

onCreate()

sendNotification()

updateNotification()

cancelNotification()

**Nhiệm vụ 3: Thêm nút hành động cho thông báo**

Đôi khi, một thông báo yêu cầu sự tương tác từ người dùng. Ví dụ, người dùng có thể tạm dừng một báo thức hoặc trả lời một tin nhắn. Khi những loại thông báo này xảy ra, người dùng có thể nhấn vào thông báo để phản hồi sự kiện. Android sau đó sẽ tải hoạt động tương ứng trong ứng dụng của bạn.

Để tránh mở ứng dụng của bạn, khung thông báo cho phép bạn nhúng một nút hành động vào chính thông báo.

Một nút hành động cần các thành phần sau:

* Một biểu tượng, để được đặt trong thông báo.
* Một chuỗi nhãn, để được đặt bên cạnh biểu tượng.
* Một PendingIntent, để được gửi khi người dùng nhấn vào nút hành động của thông báo.

Trong nhiệm vụ này, bạn sẽ thêm một nút hành động vào thông báo của mình. Nút hành động này cho phép người dùng cập nhật thông báo từ bên trong thông báo, mà không cần mở ứng dụng. Nút Cập Nhật Thông Báo hoạt động ngay cả khi ứng dụng của bạn đang chạy ở chế độ nền hoặc foreground.

**3.1 Triển khai một BroadcastReceiver để gọi updateNotification()**

Trong bước này, bạn sẽ triển khai một BroadcastReceiver gọi phương thức updateNotification() khi người dùng nhấn vào nút hành động Cập Nhật Thông Báo trong thông báo.

1. Trong MainActivity.java, thêm một lớp con của BroadcastReceiver dưới dạng lớp bên trong. Ghi đè phương thức onReceive(). Đừng quên bao gồm một constructor rỗng:
2. Trong phương thức onReceive() của NotificationReceiver, gọi updateNotification().
3. Trong MainActivity.java, tạo một biến hằng số thành viên duy nhất để đại diện cho hành động cập nhật thông báo của bạn. Đảm bảo tiền tố giá trị biến bằng tên gói ứng dụng của bạn để đảm bảo tính duy nhất:
4. Trong MainActivity.java, tạo một biến thành viên cho receiver của bạn và khởi tạo nó bằng constructor mặc định:
5. Để nhận ý định ACTION\_UPDATE\_NOTIFICATION, đăng ký broadcast receiver trong phương thức onCreate():
6. Để hủy đăng ký receiver của bạn, ghi đè phương thức onDestroy() của Activity:

**Lưu ý:** Có vẻ như broadcast gửi bởi thông báo chỉ liên quan đến ứng dụng của bạn và nên được gửi qua LocalBroadcastManager. Tuy nhiên, việc sử dụng PendingIntent ủy thác trách nhiệm gửi thông báo cho khung Android. Vì runtime Android xử lý broadcast, bạn không thể sử dụng LocalBroadcastManager.

**3.2 Tạo biểu tượng cho hành động cập nhật**

Để tạo một biểu tượng cho nút hành động cập nhật:

1. Trong Android Studio, chọn **File > New > Image Asset.**
2. Trong danh sách thả xuống **Icon Type**, chọn **Action Bar and Tab Icons.**
3. Nhấn vào biểu tượng **Clip Art.**
4. Chọn biểu tượng cập nhật  và nhấn **OK.**
5. Trong trường **Name**, đặt tên biểu tượng là ic\_update.
6. Nhấn **Next**, sau đó **Finish.**

Bắt đầu từ Android 7.0, các biểu tượng không được hiển thị trong thông báo. Thay vào đó, có nhiều không gian hơn được cung cấp cho chính các nhãn. Tuy nhiên, các biểu tượng hành động thông báo vẫn được yêu cầu và tiếp tục được sử dụng trên các phiên bản Android cũ hơn và trên các thiết bị như Android Wear.

**3.3 Thêm hành động cập nhật vào thông báo**

Trong MainActivity.java, bên trong phương thức sendNotification(), thực hiện các bước sau:

1. Ở đầu phương thức, tạo một Intent sử dụng hành động cập nhật tùy chỉnh ACTION\_UPDATE\_NOTIFICATION.
2. Sử dụng getBroadcast() để lấy một PendingIntent. Để đảm bảo rằng PendingIntent này chỉ được gửi và sử dụng một lần, đặt FLAG\_ONE\_SHOT:
3. Sử dụng phương thức addAction() để thêm một hành động vào đối tượng NotificationCompat.Builder, sau khi định nghĩa notifyBuilder. Truyền vào biểu tượng, văn bản nhãn và PendingIntent:
4. Chạy ứng dụng của bạn. Nhấn nút **Notify Me!,** sau đó nhấn nút Home. Mở thông báo và nhấn vào nút Cập Nhật Thông Báo. Thông báo sẽ được cập nhật.

Người dùng hiện có thể cập nhật thông báo mà không cần mở ứng dụng!

**Giải pháp mã**

Dự án Android Studio: NotifyMe

**Thách thức lập trình**

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

Bật và tắt các nút là một cách phổ biến để đảm bảo rằng người dùng không thực hiện bất kỳ hành động nào không được hỗ trợ trong trạng thái hiện tại của ứng dụng. Ví dụ, bạn có thể tắt nút **Sync** khi không có mạng.

Trong ứng dụng NotifyMe, có một trường hợp mà trạng thái của các nút không khớp với trạng thái của ứng dụng: khi người dùng đóng thông báo bằng cách vuốt nó đi hoặc xóa toàn bộ ngăn thông báo. Trong trường hợp này, ứng dụng của bạn không có cách nào để biết rằng thông báo đã bị hủy và trạng thái của các nút cần phải thay đổi.

Tạo một PendingIntent khác để cho phép ứng dụng biết rằng người dùng đã đóng thông báo, và chuyển đổi trạng thái của các nút cho phù hợp.

**Gợi ý:** Kiểm tra lớp NotificationCompat.Builder để tìm một phương thức gửi một Intent nếu người dùng đóng thông báo.

**Tóm tắt**

Một thông báo là một tin nhắn mà bạn có thể hiển thị cho người dùng ngoài giao diện người dùng bình thường của ứng dụng.

* Thông báo cung cấp một cách để ứng dụng của bạn tương tác với người dùng ngay cả khi ứng dụng không đang chạy.
* Khi Android phát hành một thông báo, thông báo đầu tiên xuất hiện dưới dạng một biểu tượng trong khu vực thông báo của thiết bị.
* Để chỉ định UI và các hành động cho một thông báo, sử dụng NotificationCompat.Builder.
* Để tạo một thông báo, sử dụng NotificationCompat.Builder.build().
* Để phát hành một thông báo, sử dụng NotificationManager.notify() để truyền đối tượng thông báo cho hệ thống runtime Android.
* Để có thể cập nhật hoặc hủy một thông báo, liên kết một ID thông báo với thông báo đó.
* Thông báo có thể có nhiều thành phần, bao gồm một biểu tượng nhỏ (sử dụng setSmallIcon(), là bắt buộc); một tiêu đề (setContentTitle()); và văn bản chi tiết (setContentText()).
* Thông báo cũng có thể bao gồm các pending intents, kiểu mở rộng, độ ưu tiên, v.v. Để biết thêm chi tiết, xem NotificationCompat.Builder.

**Khái niệm liên quan**

Tài liệu khái niệm liên quan có trong 8.1: Thông báo.

**Tìm hiểu thêm**

Hướng dẫn:

● Notifications Overview

● Material Design spec for notifications

● Create and Manage Notification Channels

Tài liệu tham khảo:

● NotificationCompat.Builder

● NotificationCompat.Style

**Bài tập về nhà**

Xây dựng và chạy một ứng dụng

Mở mã giải pháp cho ứng dụng NotifyMe. Thay đổi thông báo đã cập nhật trong ứng dụng để sử dụng bố cục mở rộng InboxStyle thay vì BigPictureStyle. Sử dụng dữ liệu chuỗi giả cho từng dòng và cho văn bản tóm tắt.

**Lưu ý:** Thông báo có thể trông hơi khác nhau, tùy thuộc vào cấp API của thiết bị.

**Trả lời các câu hỏi**

**Câu hỏi 1**

Chọn tất cả những câu sau là đúng về các kênh thông báo:

* Bạn sử dụng các kênh thông báo để hiển thị thông báo cho người dùng trong thanh trạng thái của thiết bị.
* Bạn sử dụng các kênh thông báo để nhóm nhiều thông báo để người dùng có thể kiểm soát hành vi của các thông báo.
* Các kênh thông báo có sẵn trên các thiết bị cũ hơn, những thiết bị chạy Android 7.0 Nougat (API 24) và thấp hơn.
* Các kênh thông báo vẫn chưa có trong gói Android Support Library.

**Câu hỏi 2**

API nào bạn sử dụng để hiển thị một thông báo trong ngăn thông báo và trong thanh trạng thái của thiết bị?

* Notification.notify()
* NotificationManager.notify()
* NotificationCompact.notify()
* NotificationCompat.Builder.notify()

**Câu hỏi 3**

Thành phần nào không cần thiết khi bạn thêm một hành động thông báo?

* Biểu tượng đại diện cho hành động
* Tiêu đề mô tả hành động
* Trình lắng nghe nhấp chuột cho sự kiện nhấp vào nút hành động.
* PendingIntent được gửi khi người dùng nhấn vào nút hành động.

**Câu hỏi 4**

API nào bạn sử dụng để thêm một nút hành động vào thông báo?

* NotificationCompat.addActionButton()
* NotificationCompat.Builder.addAction()
* Notification.Builder.addActionButton()
* NotificationManager.addAction()

**Câu hỏi 5**

Giả sử bạn tạo một ứng dụng tải xuống một tác phẩm nghệ thuật trên thiết bị của người dùng mỗi ngày. Khi hình ảnh của ngày đã sẵn sàng, ứng dụng hiển thị một thông báo cho người dùng, và người dùng có thể tải xuống hình ảnh hoặc bỏ qua tải xuống. Phương thức PendingIntent nào bạn sẽ sử dụng để bắt đầu một dịch vụ tải xuống hình ảnh?

* Activity.startService()
* PendingIntent.getBroadcast()
* PendingIntent.getActivity()
* PendingIntent.getService()

**Nộp ứng dụng của bạn để chấm điểm**

**Hướng dẫn cho người chấm điểm**

Kiểm tra xem ứng dụng có các tính năng sau không:

Khi người dùng nhấn nút Cập Nhật Thông Báo, thông báo trở thành thông báo kiểu InboxStyle với nhiều dòng văn bản đại diện cho các mục dòng.

Màn hình có một tóm tắt và dòng văn bản tiêu đề, thay đổi vị trí của nó tùy thuộc vào cấp API.

Ứng dụng sử dụng lớp NotificationCompat.InboxStyle để tương thích ngược.

### Trình quản lý cảnh báo

### JobScheduler

# LƯU DỮ LIỆU NGƯỜI DÙNG

## Tùy chọn và cài đặt

### Shared preferences

### Cài đặt ứng dụng

## Lưu trữ dữ liệu với Room

### Room, LiveData và ViewModel

### Room, LiveData và ViewModel