**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**GIÁO TRÌNH**

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

Hà Nội, 2.2025

MỤC LỤC

[CHƯƠNG 1. Làm quen 4](#_Toc190855147)

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

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

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

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

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

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

[Bài 2) Activities 5](#_Toc190855154)

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

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

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

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

[3.1) Trình gỡ lỗi 5](#_Toc190855159)

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

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

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

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

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

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

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

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

[1.5) RecycleView 6](#_Toc190855168)

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

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

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

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

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

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

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

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

[1.1) AsyncTask 6](#_Toc190855177)

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

[1.3) Broadcast receivers 6](#_Toc190855179)

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

[2.1) Thông báo 6](#_Toc190855181)

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

[2.3) JobScheduler 6](#_Toc190855183)

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

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

[1.1) Shared preferences 7](#_Toc190855186)

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

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

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

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

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

# LÀM QUEN

## Tạo ứng dụng đầu tiên

### Android Studio và Hello World

Giới thiệu

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

Những gì Bạn nên biết

Bạn nên có khả năng:

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

A screenshot of a computer

AI-generated content may be incorrect.

Những gì Bạn sẽ cần:

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

Những gì bạn sẽ học

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

Những gì bạn sẽ làm

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

### 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

Luồng (thread) là một đường dẫn thực thi độc lập trong một chương trình đang chạy. Khi một chương trình Android được khởi chạy, hệ thống sẽ tạo một luồng chính, còn được gọi là luồng UI. Luồng UI này là cách ứng dụng của bạn tương tác với các thành phần từ UI Android Toolkit.

Tuy nhiên, đôi khi một ứng dụng cần thực hiện các tác vụ tốn nhiều tài nguyên chẳng hạn như tải xuống tệp, thực hiện truy vấn cơ sở dữ liệu, phát phương tiện hoặc tính toán phân tích phức tạp. Những công việc tiêu tốn nhiều tài nguyên này có thể làm tắc nghẽn luồng UI khiến ứng dụng không phản hồi với thao tác của người dùng hoặc không thể cập nhật giao diện. Người dùng có thể cảm thấy khó chịu và gỡ cài đặt ứng dụng của bạn.

Để đảm bảo cho trải nghiệm người dùng (UX) mượt mà, Android framework cung cấp một lớp hỗ trợ có tên AsyncTask, giúp xử lý các tác vụ nặng ngoài luồng UI. Bằng cách chuyển các công việc này sang một luồng riêng biệt, ứng dụng vẫn có thể duy trì khả năng phản hồi, tránh tình trạng “đơ” hay chậm trễ khi người dùng tương tác.

Vì luồng riêng biệt không được đồng bộ hóa với luồng gọi, nên nó được gọi là luồng không đồng bộ (asynchronous thread). AsyncTask cũng cung cấp các phương thức gọi lại (callback), cho phép bạn trả kết quả xử lý về luồng UI.

Trong phần thực hành này, bạn sẽ tìm hiểu cách tích hợp tác vụ nền vào ứng dụng Android bằng AsyncTask.

Những kiến thức bạn cần biết trước

Bạn cần có thể:

* Tạo một Activity.
* Thêm một TextView vào bố cục của Activity.
* Lấy id của TextView bằng mã lập trình và thiết lập nội dung cho nó.
* Sử dụng Button và xử lý sự kiện onClick của nó.

Những gì bạn sẽ tìm hiểu

* Cách thêm AsyncTask vào ứng dụng để chạy một tác vụ trong nền.
* Những hạn chế của AsyncTask khi sử dụng cho các tác vụ nền

Những gì bạn sẽ làm

* Tạo một ứng dụng đơn giản thực hiện tác vụ nền bằng AsyncTask.
* Chạy ứng dụng và quan sát điều gì xảy ra khi xoay màn hình thiết bị.
* Triển khai lưu trạng thái Activity để bào toàn nội dung của TextView khi thay đổi cấu hình

Tổng quan về ứng dụng

Bạn sẽ xây dựng một ứng dụng với 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 hoạt động trở lại.

Dưới đây là giao diện của ứng dụng sau khi hoàn thành:

\*\*\*Finished UI\*\*\*

Nhiệm vụ 1: Thiết lập dự án SimpleAsyncTask

Giao diện của SimpleAsyncTask bao gồm một Button để kích hoạt AsyncTask và một TextView để hiển thị trạng thái của ứng dụng.

* 1. Tạo dự án và bố cục giao diện

Tạo một dự án mới có tên SimpleAsyncTask sử dụng mẫu Empty Activity. Giữ nguyên các tùy chọn mặc định.

Mở tệp bố cục activity\_main.xml, sau đó chuyển sang tab Text.

Thêm thuộc tính layout\_margin vào ConstraintLayout cấp cao nhất

Thêm hoặc chỉnh sửa các thuộc tính sau của TextView chứa dòng chữ “Hello World!” để có giá trị sau (trích xuất chuỗi văn bản vào resource):

|  |  |
| --- | --- |
| Thuộc tính | Giá trị |
| android:id | “@+id/textView1 |
| android:text | “I am ready to start work!” |
| android:textSize | “24sp” |

Xóa các thuộc tính app:layout\_constraintRight\_toRightOf và app:layout\_constraintTop\_toTopOf.

Thêm một Button ngay bên dưới TextView, đồng thời thiết lập các thuộc tính sau (trích xuất văn bản của Button vào resource):

|  |  |
| --- | --- |
| Thuộc tính | Giá trị |
| android:id | “@+id/button” |
| android:layout\_width | “wrap\_content” |
| android:layout\_height | “wrap\_content” |
| android:text | “Start Task” |
| android:layout\_marginTop | “24dp” |
| android:onClick | “startTask” |
| app:layout\_constraintStart\_toStartOf | “parent” |
| app:layout\_constraintTop\_toBottomOf | “@+id/textView1” |

Thuộc tính onClick của Button sẽ được tô vàng vì phương thức startTask() chưa được triển khai trong MainActivity. Để tạo phương thức này, hãy đặt con trỏ vào đoạn văn bản được đánh dấu, nhấn Alt + Enter (hoặc Option + Enter trên Mac), sau đó chọn Create ‘startTask(View)’ in ‘MainActivity’. Điều này sẽ tự động tạo một phương thức khung (stub) trong MainActivity.

Nhiệm vụ 2: Tạo lớp con của AsyncTask

AsyncTask là một lớp trừu trượng, có nghĩa là bạn cần tạo một lớp con kế thừa từ nó để sử dụng. Trong ví dụ này, AsyncTask sẽ 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 bao gồm nhiều công việc phức tạp hơn, từ truy vấn cơ sở dữ liệu, kết nối internet cho đến tính toán nước đi tiếp theo để đánh bại nhà vô địch cờ vây hiện tại.

Một lớp con của AsyncTask có các phương thức sau để thực hiện công việc ngoài luồng chính:

* onPreExecute(): Chạy trên luồng UI, được sử dụng để chuẩn bị cho tác vụ (chẳng hạn như hiển thị thanh tiến trình).
* doInBackground(): Chứa mã thực thi tác vụ chính trên một luồng riêng biệt.
* onProgressUpdate(): Được gọi trên luồng UI để cập nhật tiến trình (ví dụ: hiển thị mức độ hoàn thành trên thanh tiến trình).
* onPostExecute(): Cũng chạy trên luồng UI, dùng để cập nhật kết quả lên giao diện sau khi AsyncTask hoàn tất.

Khi tạo một lớp con của AsyncTask, bạn có thể cần cung cấp thông tin về công việc cần thực hiện, cách báo cáo tiến trình (nếu có), cũng như định dạng của kết quả trả về.

Bạn có thể cấu hình lớp con AsyncTask bằng các tham số sau:

* Params: Kiểu dữ liệu của tham số được truyền vào doInBackground() khi tác vụ được thực thi.
* Progress: Kiểu dữ liệu của đơn vị tiến trình được cập nhật thông qua onProgressUpdate().
* Result: Kiểu dữ liệu của kết quả trả về từ onPostExecute().

Ví dụ, nếu bạn có một lớp con AsyncTask có tên MyAsyncTask, nó có thể sử dụng các tham số như sau”

* doInBackground() nhận một String làm tham số (chẳng hạn để thực hiện truy vấn).
* onProgressUpdate() sử dụng một Integer để biểu thị phần trăm công việc đã hoàn thành.
* onPostExecute() trả về một Bitmap làm kết quả của truy vấn.

\*\*\*code\*\*\*

Trong phần thực hành này, bạn sẽ sử dụng một lớp con AsyncTask để định nghĩa một công việc sẽ chạy trên một luồng riêng biệt, tách biệt với luồng UI.

2.1 Tạo lớp con từ AsyncTask

Trong ứng dụng này, lớp con AsyncTask mà bạn tạo sẽ không cần tham số truy vấn hay cập nhật tiến trình. Bạn chỉ sử dụng hai phương thức: doInBackground() và onPostExecute().

Các bước thực hiện:

1. Tạo một lớp Java mới có tên SimpleAsyncTask, kế thừa từ AsyncTask và sử dụng ba tham số kiểu dữ liệu tổng quát (generic type parameters). Đặt các tham số như sau:

\* Void cho Params, vì tác vụ này không cần dữ liệu đầu vào.

\* Void cho Progress, vì tiến trình không được cập nhật.

\* String cho Result, vì sau khi AsyncTask hoàn thành, bạn sẽ cập nhật TextView bằng một chuỗi.

\*\*\*code\*\*\*

2. Ở đầu lớp, khai báo một biến thành viên mTextView với kiểu WeakReference<TextView>:

\*\*\*code\*\*\*

3. Triển khai constructor cho AsyncTask, nhận một TextView làm tham số và tạo một tham chiếu yếu (weak reference) đến TextView đó:

\*\*\*code\*\*\*

Lớp AsyncTask cần cập nhật TextView trong Activity sau khi hoàn tất tác vụ ngủ (sleeping) trong phương thức onPostExecute(). Vì vậy, constructor của lớp sẽ cần một tham chiếu đến TextView để có thể cập nhật nội dung sau khi tác vụ hoàn tất.

Tại sao cần sử dụng WeakReference? Nếu bạn truyền trực tiếp một TextView vào constructor của AsyncTask và lưu nó vào một biến thành viên, thì tham thiếu đó sẽ khiến Activity không thể được garbage collected (thu hồi bộ nhớ), ngay cả khi Activity bị hủy và tạo lại (chẳng hạn như khi xoay màn hình. Điều này được gọi là leaky context (rò rỉ bộ nhớ), và Android Studio sẽ cảnh báo nếu bạn cố làm như vậy.

Việc sử dụng WeakReference giúp ngăn chặn rò rỉ bộ nhớ bằng cách cho phép đối tượng được tham chiếu có thể bị thu hồi bộ nhớ nếu cần thiết.

2.2 Triển khai phương thức doInBackground()

Phương thức doInBackground() là bắt buộc trong lớp con của AsyncTask.

1. Đặt con trỏ chuột vào phần khai báo lớp bị tô sáng, nhấn Alt + Enter (Option + Enter trên Mac), sau đó chọn Implement methods. Chọn doInBackground() và nhấn OK. Một mẫu phương thức sau sẽ được thêm vào lớp của bạn:

\*\*\*code\*\*\*

1. Thêm đoạn mã để tạo một số nguyên ngẫu nhiên trong khoảng 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:  
   \*\*\*code\*\*\*
2. Thêm một khối try / catch để cho luồng ngủ (sleep) trong khoảng thời gian đã tính toán:

\*\*\*code\*\*\*

1. Thay thế dòng return hiện hữu để trả về chuỗi “Thức dậy rồi! Sau khi ngủ trong xx mili-giây”. Trong đó xx là số mili-giây mà ứng dụng đã tạm dừng.

\*\*\*code\*\*\*

Phương thức doInBackground() hoàn chỉnh sẽ trông như sau:

\*\*\*code\*\*\*

2.3 Triển khai phương thức onPostExecute()

Sau khi phương thức doInBackground() hoàn thành, giá trị trả về sẽ tự động được truyền vào phương thức onPostExecute() để xử lý kết quả.

### AsyncTask và AsyncTaskLoader

### Broadcast receivers

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

### Thông báo

### Trình quản lý cảnh báo

### JobScheduler

# LƯU DỮ LIỆU NGƯỜI DÙNG

## Tùy chọn và cài đặt

### Shared preferences

### Cài đặt ứng dụng

## Lưu trữ dữ liệu với Room

### Room, LiveData và ViewModel

### Room, LiveData và ViewModel