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

• LÀM QUEN

• Tạo ứng dụng đầu tiên

• Android Studio và Hello World

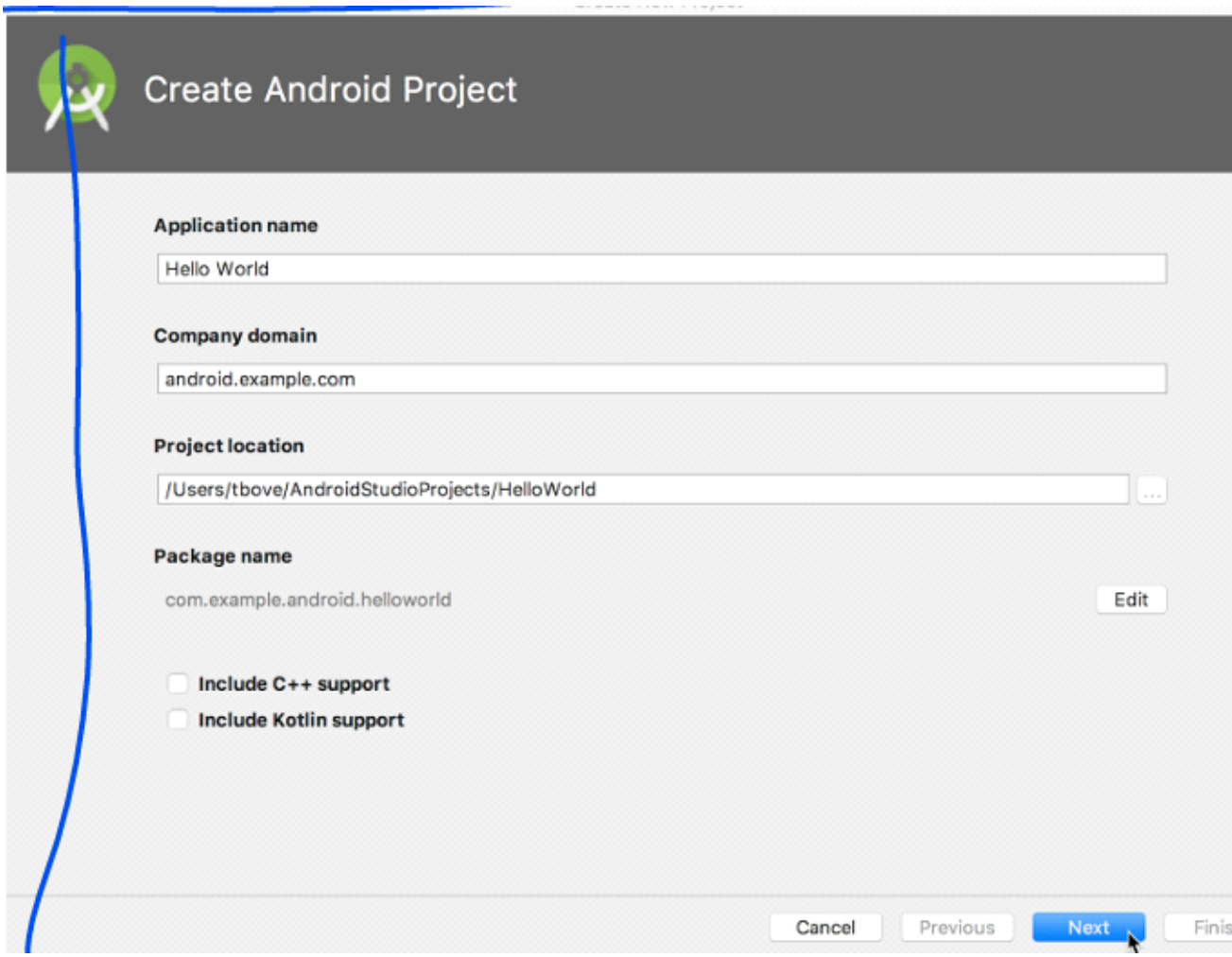
Giới thiệu

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.



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**
    - **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**

UNIT 3: Làm việc ở background

Lesson 7.1: AsyncTask

Giới thiệu

Một mối nguy là một đường thực thi độc lập trong một dự án đang chạy. Khi một chương trình Android được khởi tạo, hệ thống tạo ra một mối nguy chính, hay còn được gọi là mối nguy UI. Mối nguy UI là cách mà ứng dụng của bạn tương tác với các thành phần từ bộ công cụ UI của Android.

Thi thoảng một ứng dụng cần làm những tác vụ tiêu hao nhiều tài nguyên như tải thư mục, truy vấn cơ sở dữ liệu, chạy những bản ghi, hay thực hiện những phép tính phức tạp. Những tác vụ nặng như trên có thể chặn mối nguy UI nên ứng dụng sẽ không phản hồi với những tương tác của người dùng. Người dùng có thể bị khó chịu và gỡ cài đặt ứng dụng của bạn.

Để trải nghiệm người dùng được mượt mà, Android framework cung cấp một lớp hỗ trợ được gọi là AsyncTask, lớp hỗ trợ này sử dụng mối nguy UI để vận hành. Sử dụng AsyncTask để di chuyển sự xử lý chuyên xâu đến một mối nguy riêng biệt mà mối nguy UI có thể đáp ứng.

Vì mối nguy riêng biệt kia không được đồng bộ với mối nguy đang gọi, nên nó được gọi là một mối nguy không đồng bộ. Một cái AsynTask cũng bao gồm một lời gọi về cho phép bạn hiển thị kết quả của phép tính trong mối nguy UI

Trong bài giảng này, bạn học được cách thêm một tác vụ nền vào trong ứng dụng Android bằng việc sử dụng AsynTask.

Những thứ bạn nên biết trước:

Bạn nên biết cách:

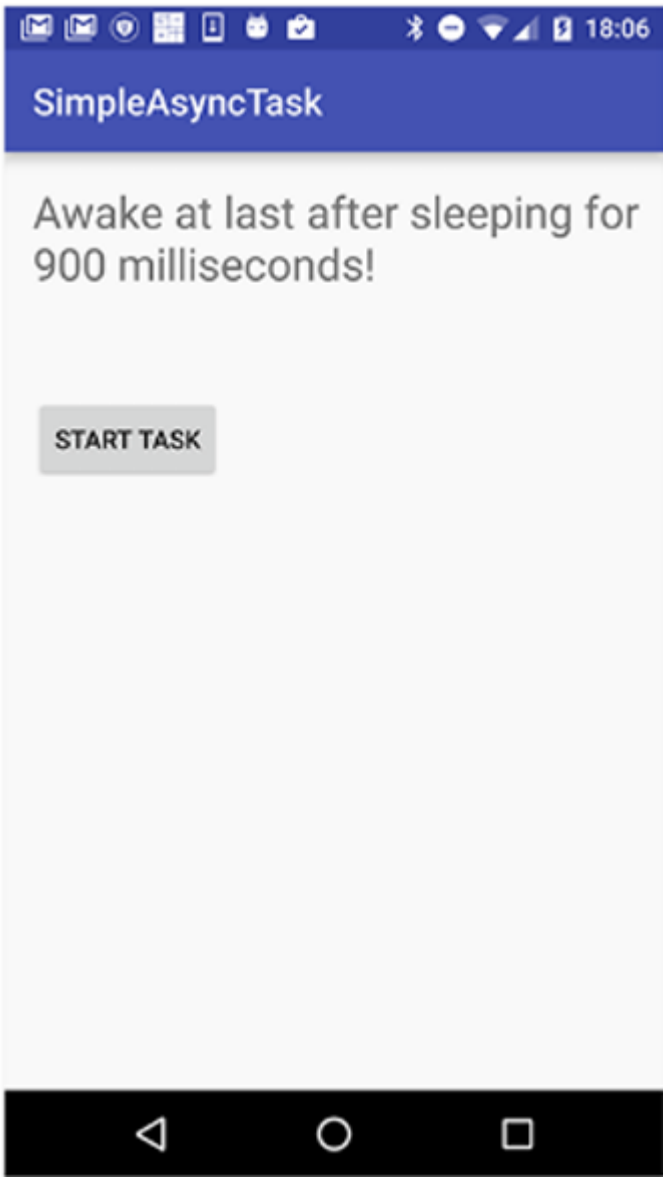
- Tạo một Activity
- Thêm một TextView vào layout cho Activity
- Lấy id và sửa chủ đề của TextView bằng code
- Sử dụng Button views và tính năng onClick của chúng

Bạn sẽ học được:

- Tạo ra một ứng dụng đơn giản có thể ngắt một tác vụ nền sử dụng AsynTask
- Chạy ứng dụng và xem điều gì sẽ xảy ra khi bạn xoay thiết bị
- Triển khai trạng thái hoạt động để giữ lại trạng thái của nội dung trong TextView

Tổng quan về ứng dụng:

Bạn sẽ xay dựng 1 ứng dụng có 1 TextView và 1 Button. Khi người dùng nhấn vào Button, ứng dụng sẽ ngủ trong 1 khoảng tgian ngẫu nhiên và rồi trình chiếu 1 lời nhắn trong TextView khi nó dậy lại. Ứng dụng khi được hoàn thành sẽ trông như này:



Bước 1: Cài đặt dự án SimpleAsynTask

SimpleAsynTask UI chứa một Button để chạy AsynTask và một TextView hiển thị trạng thái của ứng dụng.

1.1 Tạo dự án và layout

1. Tạo 1 dự án mới tên là SimpleAsyncTask sử dụng khung mẫu Empty Activity. Chấp nhận mọi lựa chọn mặc định.
2. Mở activity_main.xml. Nhấn vào Text.
3. Thêm thuộc tính layout_margin vào ConstraintLayout ở trên cùng.

```
android:layout_margin="16dp"
```

4. Thêm hoặc thay đổi những thuộc tính ở dưới của TextView “Hello World!” cho có những giá trị như sau. Trích xuất chuỗi thành một tài nguyên

Attribute	Value
android:id	"@+id/textView1
android:text	"I am ready to start work!"
android:textSize	"24sp"

5. Xóa ứng các thuộc tính: layout_constraintRight_toRightOf và app:layout_constraintTop_toTopOf
6. Thêm 1 Button dưới TextView và cho nó những thuộc tính sau. Xuất văn bản của nút ấy thành dạng string

Attribute	Value
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"

7. Thuộc tính onClick của nút sẽ được hiện thị với màu vàng, bởi vì phương pháp startTask() vẫn chưa được thêm vào MainActivity. Đặt trỏ chuột vào dòng chữ được ánh màu, nhấn Alt rồi Enter và chọn Create ‘startTask(View) in ‘MainActivity’ để tạo phương pháp sơ khai trong MainActivity.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_margin="16dp"
   tools:context=".MainActivity">

   <TextView
       android:id="@+id/textView1"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/ready_to_start"
       android:textSize="24sp"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toTopOf="parent"/>

   <Button
       android:id="@+id/button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="24dp"
       android:onClick="startTask"
       android:text="@string/start_task"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toBottomOf="@+id/textView1"/>

</android.support.constraint.ConstraintLayout>
```

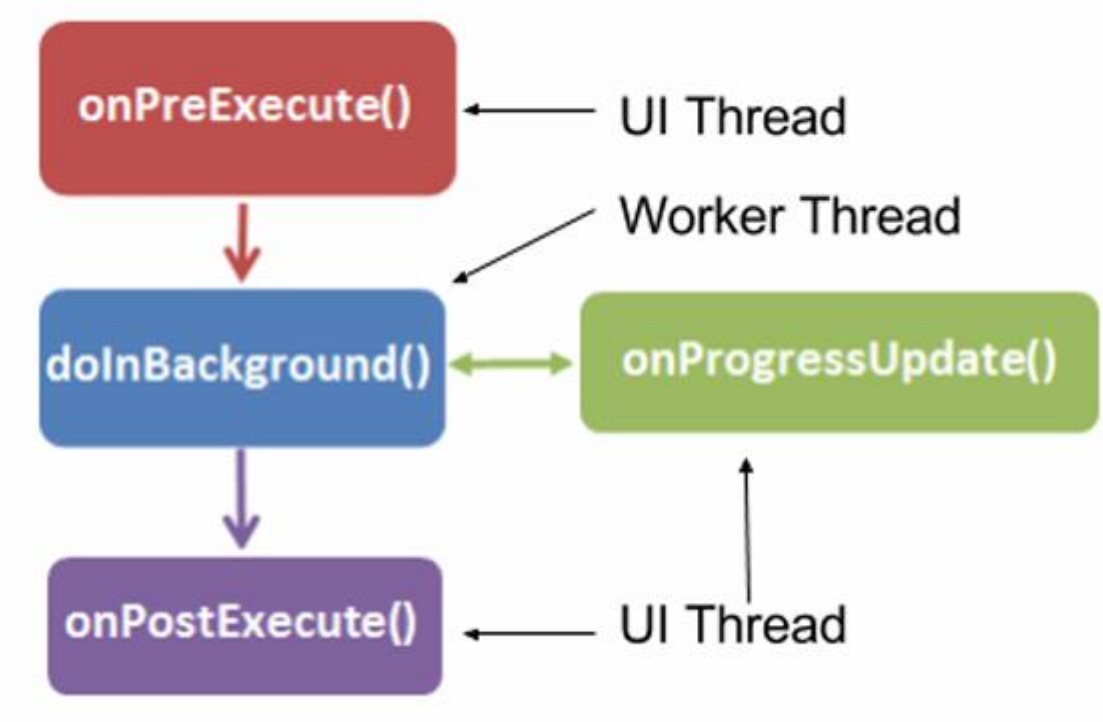
Phần 2: Tạo kế thừa của AsyncTask

AsyncTask là một lớp trừu tượng (*abstract class*), có nghĩa là bạn phải kế thừa (*subclass*) nó để sử dụng. Trong ví dụ này, **AsyncTask** thực hiện một tác vụ nền (*background task*) rất đơn giản: nó sẽ ngủ (*sleep*) 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, từ truy vấn cơ sở dữ liệu (*querying a database*), kết nối Internet (*connecting to the internet*), đến tính toán nước đi tiếp theo trong trò chơi cờ vây (*Go*) để đánh bại nhà vô địch hiện tại.

Một lớp con của **AsyncTask** sẽ có các phương thức sau để thực hiện công việc ngoài luồng chính (*main thread*):

- **onPreExecute()**: Chạy trên luồng giao diện người dùng (*UI thread*), được sử dụng để thiết lập tác vụ (chẳng hạn như hiển thị thanh tiến trình - *progress bar*).

- **doInBackground()**: Nơi bạn triển khai mã để thực thi công việc trên một luồng riêng biệt (*separate thread*).
- **onProgressUpdate()**: Được gọi trên luồng giao diện người dùng, dùng để cập nhật tiến trình trong UI (ví dụ như làm đầy thanh tiến trình).
- **onPostExecute()**: Cũng chạy trên luồng giao diện người dùng, được sử dụng để cập nhật kết quả lên UI sau khi **AsyncTask** hoàn thành xử lý.



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 thức và thời điểm báo cáo tiến trình, cũng như định dạng của kết quả trả về. Bạn có thể cấu hình **AsyncTask** bằng cách sử dụng các tham số sau:

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

Ví dụ, một lớp con của **AsyncTask** có tên **MyAsyncTask** với khai báo lớp như sau có thể nhận các tham số sau:

- Một **String** làm tham số trong **doInBackground()**, có thể dùng để thực hiện một truy vấn (*query*), chẳng hạn.
- Một **Integer** trong **onProgressUpdate()**, đại diện cho phần trăm công việc đã hoàn thành.
- Một **Bitmap** làm kết quả trong **onPostExecute()**, biểu thị kết quả của truy vấn.

```
public class MyAsyncTask
   extends AsyncTask <String, Integer, Bitmap>{}
```

Trong nhiệm vụ này, bạn sẽ sử dụng một lớp con của **AsyncTask** để định nghĩa công việc sẽ chạy trên một luồng khác với **UI thread**.

2.1 Kế thừa AsyncTask

Trong ứng dụng này, lớp con **AsyncTask** mà bạn tạo không yêu cầu tham số truy vấn (*query parameter*) hoặc cập nhật tiến trình. Bạn chỉ cần sử dụng hai phương thức **doInBackground()** và **onPostExecute()**.

1. Tạo lớp SimpleAsyncTask kế thừa AsyncTask

- Tạo một lớp Java mới có tên **SimpleAsyncTask** kế thừa **AsyncTask** và khai báo ba tham số kiểu tổng quát (*generic type parameters*).
- Sử dụng **Void** cho **Params**, vì **AsyncTask** này không cần đầu vào.
- Sử dụng **Void** cho **Progress**, vì tiến trình không được cập nhật.
- Sử dụng **String** cho **Result**, vì bạn sẽ cập nhật **TextView** với một chuỗi sau khi **AsyncTask** hoàn tất thực thi.

```
public class SimpleAsyncTask extends AsyncTask <Void, Void, String>{}
```

2. Định nghĩa biến thành viên mTextView

- Ở đầu lớp, khai báo một biến thành viên **mTextView** thuộc kiểu **WeakReference<TextView>**.

```
private WeakReference<TextView> mTextView;
```

3. Triển khai constructor của AsyncTask

- Viết một constructor nhận một **TextView** làm tham số và tạo một tham chiếu yếu (*weak reference*) đến **TextView** đó.

```
SimpleAsyncTask(TextView tv) {
       mTextView = new WeakReference<>(tv);
}
```

AsyncTask cần cập nhật **TextView** trong **Activity** sau khi hoàn thành nhiệm vụ (trong phương thức **onPostExecute()**), do đó, constructor phải có một tham chiếu đến **TextView** cần cập nhật.

Tại sao cần sử dụng WeakReference (WeakReference class)?

Nếu bạn truyền trực tiếp một **TextView** vào constructor của **AsyncTask** và lưu trữ nó như một biến thành viên, điều này sẽ tạo một tham chiếu mạnh (*strong reference*) đến **TextView**. Kết quả là, **Activity** chứa **TextView** đó sẽ không bao giờ được thu gom rác (*garbage collected*), gây ra rò rỉ bộ nhớ (*memory leak*), ngay cả khi **Activity** bị hủy và tạo lại do thay đổi cấu hình thiết bị.

Đây được gọi là **leaky context**, và **Android Studio** sẽ cảnh báo nếu bạn cố gắng thực hiện theo cách này.

Sử dụng **WeakReference** giúp ngăn chặn rò rỉ bộ nhớ bằng cách cho phép đối tượng được thu gom rác nếu cần thiết, tránh giữ tham chiếu mạnh đến **Activity**.

2.2 Triển khai doInBackground()

Phương thức **doInBackground()** là bắt buộc đối với lớp con của **AsyncTask**.

1. Triển khai phương thức doInBackground()

- Đặt con trỏ vào phần khai báo lớp đã được tô sáng, nhấn **Alt + Enter** (*Option + Enter* trên Mac) và chọn **Implement methods**. Chọn **doInBackground()** và nhấn **OK**. Một mẫu phương thức (**method template**) sẽ được tự động thêm vào lớp của bạn.

```
@Override
protected String doInBackground(Void... voids) {
    return null;
}
```

2. Tạo số nguyên ngẫu nhiên

- Thêm 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. Nhân giá trị này với **200** để kéo dài thời gian chờ.

```
Random r = new Random();
int n = r.nextInt(11);

int s = n * 200;
```

3. Sử dụng try/catch và đặt luồng vào trạng thái ngủ

- Bọc đoạn mã ngủ (**sleep**) trong một khối **try/catch** để xử lý ngoại lệ **InterruptedException**.

```
try {
    Thread.sleep(s);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

4. Thay đổi câu lệnh return

- Thay thế câu lệnh return hiện tại bằng một chuỗi có dạng: **"Awake at last after sleeping for xx milliseconds"**, trong đó **xx** là số mili-giây mà ứng dụng đã ngủ.

```
return "Awake at last after sleeping for " + s + " milliseconds!";
```

2.3 Triển khai onPostExecute()

Khi phương thức **doInBackground()** hoàn tất, giá trị trả về sẽ tự động được chuyển đến phương thức **onPostExecute()**.

1. Triển khai **onPostExecute()** để nhận một tham số **String** và hiển thị nó trong **TextView**.

```
protected void onPostExecute(String result) {
    mTextView.get().setText(result);
}
```

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 cũng là giá trị mà phương thức **doInBackground()** trả về.

Vì **mTextView** là một **WeakReference**, bạn cần giải tham chiếu (*dereference*) nó bằng phương thức **get()** để lấy đối tượng **TextView** thực tế, sau đó gọi **setText()** trên đối tượng đó.

Phần 3: Triển khai các bước cuối cùng

3.1 Triển khai phương thức khởi chạy AsyncTask

Ứng dụng của bạn hiện có một lớp **AsyncTask** thực thi công việc trong nền (*background task*) (hoặc sẽ thực thi nếu nó không chỉ gọi **sleep()** như một công việc mô phỏng). Bây giờ bạn có thể triển khai phương thức **onClick** cho nút **"Start Task"** để kích hoạt tác vụ nền.

Các bước triển khai:

1. Trong tệp **MainActivity.java**, khai báo một biến thành viên để lưu trữ đối tượng **TextView**.

```
private TextView mTextView;
```

2. Trong phương thức **onCreate()**, khởi tạo **mTextView** bằng cách liên kết nó với **TextView** trong tệp giao diện người dùng (*layout*).

```
mTextView = findViewById(R.id.textView1);
```

3. Trong phương thức **startTask()**, cập nhật **TextView** để hiển thị thông báo **"Napping…"**. Trích xuất chuỗi này vào **string resource** để quản lý tài nguyên tốt hơn.

```
mTextView.setText(R.string.napping);
```

4. Tạo một thể hiện của **SimpleAsyncTask**, truyền **mTextView** vào constructor. Gọi phương thức **execute()** trên thể hiện của **SimpleAsyncTask** để khởi chạy tác vụ bất đồng bộ.

```
new SimpleAsyncTask(mTextView).execute();
```

Code cho MainActivity:

```
package com.example.android.simpleasynctask;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

/**
* The SimpleAsyncTask app contains a button that launches an AsyncTask
```

```
* which sleeps in the asynchronous thread for a random amount of time.
*/
public class MainActivity extends AppCompatActivity {

   // The TextView where we will show results
   private TextView mTextView;

  @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);

       mTextView = findViewById(R.id.textView1);
   }

   public void startTask(View view) {
       // Put a message in the text view
       mTextView.setText(R.string.napping);

       // Start the AsyncTask.
       new SimpleAsyncTask(mTextView).execute();
   }
}
```

3.2 Triển khai onSaveInstanceState()

1. Chạy ứng dụng và nhấn nút Start Task. Ứng dụng ngủ trong bao lâu?

2. Nhấn nút Start Task một lần nữa và trong khi ứng dụng đang ngủ, xoay thiết bị. Nếu tác vụ nền hoàn tất trước khi bạn có thể xoay điện thoại, hãy thử lại.

Có một số vấn đề xảy ra ở đây:

- Khi bạn xoay thiết bị, hệ thống khởi động lại ứng dụng, gọi **onDestroy()** và sau đó **onCreate()**.
  AsyncTask sẽ tiếp tục chạy ngay cả khi **Activity** bị hủy, nhưng nó sẽ mất khả năng báo cáo lại cho UI của **Activity**. Nó sẽ không bao giờ có thể cập nhật **TextView** được truyền vào, vì **TextView** cụ thể đó cũng đã bị hủy.
- Khi **Activity** bị hủy, **AsyncTask** vẫn tiếp tục chạy đến khi hoàn tất trong nền, tiêu tốn tài nguyên hệ thống.
  Cuối cùng, hệ thống sẽ hết tài nguyên và **AsyncTask** sẽ thất bại.
- Ngay cả khi không có **AsyncTask**, việc xoay thiết bị cũng đặt lại tất cả các phần tử UI về trạng thái mặc định,
  mà đối với **TextView** là chuỗi mặc định bạn đã đặt trong tệp **layout**.

Vì những lý do này, **AsyncTask** không phù hợp cho các tác vụ có thể bị gián đoạn bởi việc hủy **Activity**.

3. Ở đầu lớp, thêm một hằng số làm khóa (key) để lưu trữ nội dung hiện tại của **TextView** trong **Bundle trạng thái**.

```
private static final String TEXT_STATE = "currentText";
```

4. Ghi đè (override) phương thức **onSaveInstanceState()** trong **MainActivity** để lưu trạng thái của **TextView** khi **Activity** bị hủy.

```
@Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        // Save the state of the TextView
```

```
        outState.putString(TEXT_STATE,
            mTextView.getText().toString());
    }
}
```

5. Trong **onCreate()**, truy xuất giá trị của **TextView** từ **Bundle trạng thái** khi **Activity** được khôi phục.

```
// Restore TextView if there is a savedInstanceState
if(savedInstanceState!=null){
  mTextView.setText(savedInstanceState.getString(TEXT_STATE));
```

Thử thách lập trình

Thử thách: Lớp **AsyncTask** cung cấp một phương thức ghi đè hữu ích khác: **onProgressUpdate()**, cho phép cập nhật giao diện trong khi **AsyncTask** đang chạy. Sử dụng phương thức này để cập nhật giao diện với thời gian ngủ hiện tại.
Tham khảo tài liệu về **AsyncTask** để xem cách **onProgressUpdate()** được triển khai đúng cách.
Lưu ý rằng trong phần khai báo lớp **AsyncTask**, cần chỉ định kiểu dữ liệu được sử dụng trong phương thức **onProgressUpdate()**.

Tóm tắt:

- **AsyncTask** là một lớp Java trừu tượng giúp chuyển xử lý chuyên sâu sang một luồng riêng biệt.
- **AsyncTask** phải được kế thừa để sử dụng.
- Tránh thực hiện các tác vụ tốn tài nguyên trong **UI thread**, vì có thể làm giao diện chậm hoặc không ổn định.
- Bất kỳ mã nào không liên quan đến vẽ giao diện hoặc phản hồi đầu vào người dùng nên được chuyển khỏi **UI thread** sang một luồng khác.
- **AsyncTask** có bốn phương thức chính: **onPreExecute()**, **doInBackground()**, **onPostExecute()**, và **onProgressUpdate()**.
- **doInBackground()** là phương thức duy nhất chạy trên **worker thread**. Không gọi phương thức giao diện trong **doInBackground()**.
- Các phương thức khác của **AsyncTask** chạy trên **UI thread**, cho phép gọi các phương thức của giao diện.

- Khi xoay màn hình thiết bị Android, **Activity** bị hủy và tạo lại. Điều này có thể làm ngắt kết nối giữa giao diện và **AsyncTask**, nhưng **AsyncTask** vẫn tiếp tục chạy nền.

Bài 7.2: AsyncTask và AsyncTaskLoader

Giới thiệu

Trong bài thực hành này, bạn sẽ sử dụng **AsyncTask** để khởi chạy một tác vụ nền nhằm lấy dữ liệu từ internet bằng cách sử dụng một **REST API** đơn giản. Bạn sẽ sử dụng **Google APIs Explorer** để truy vấn **Books API**, triển khai truy vấn này trong một **worker thread** bằng **AsyncTask**, và hiển thị kết quả trên giao diện người dùng (**UI**).

Sau đó, bạn sẽ triển khai lại cùng một tác vụ nền bằng **AsyncTaskLoader**, một cách tiếp cận hiệu quả hơn để cập nhật giao diện.

Những gì bạn cần biết trước

Bạn cần có khả năng:

- Tạo một **Activity**.
- Thêm một **TextView** vào **layout** của **Activity**.
- Triển khai chức năng **onClick** cho một **Button** trong **layout**.
- Triển khai **AsyncTask** và hiển thị kết quả lên **UI**.
- Truyền dữ liệu giữa các **Activity** bằng **extras**.

Những gì bạn sẽ học được

- Cách sử dụng **Google APIs Explorer** để tìm hiểu về **Google APIs** và xem phản hồi **JSON** từ các yêu cầu **HTTP**.
- Cách sử dụng **Google Books API** để lấy dữ liệu từ internet, giúp giao diện nhanh và mượt mà. Bạn sẽ không học chi tiết về **Books API**—ứng dụng chỉ sử dụng chức năng tìm kiếm sách đơn giản.
- Cách phân tích cú pháp kết quả **JSON** từ truy vấn **API**.
- Cách triển khai **AsyncTaskLoader** để duy trì dữ liệu khi thay đổi cấu hình.
- Cách cập nhật giao diện bằng **loader callbacks**.

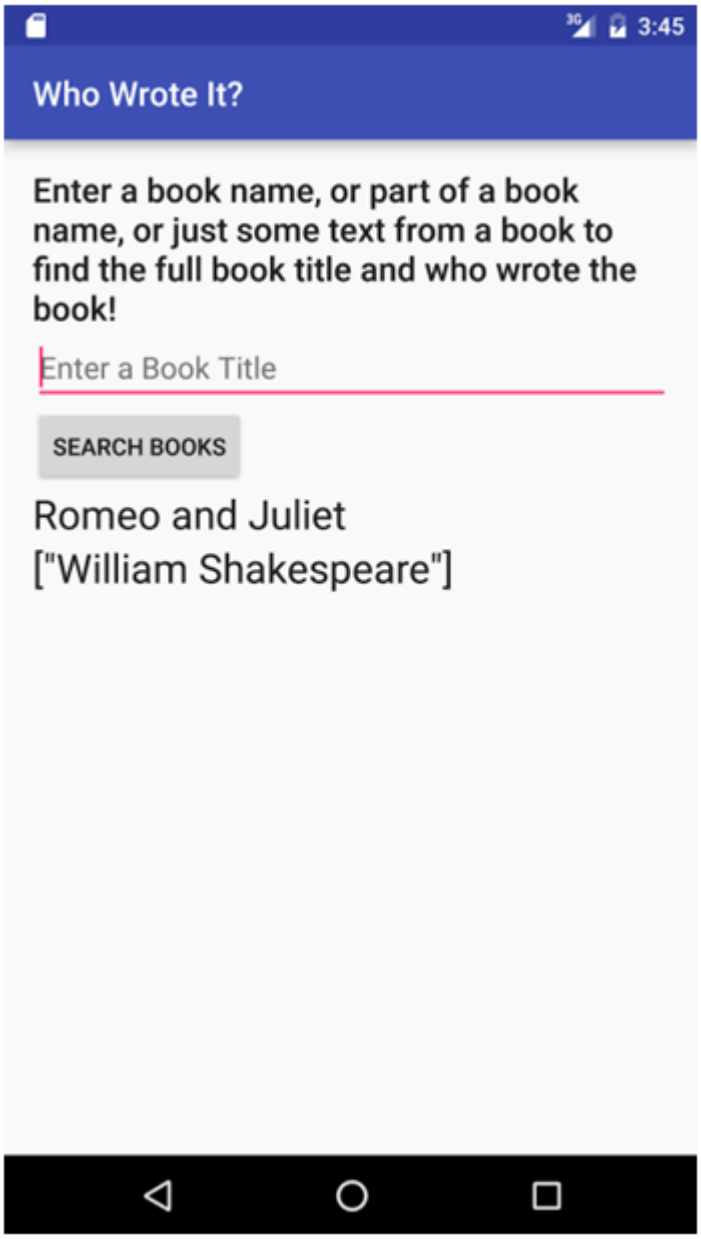
Những gì bạn sẽ làm

- Sử dụng **Google APIs Explorer** để tìm hiểu về **Books API**.
- Tạo ứng dụng **"Who Wrote It?"**, ứng dụng này truy vấn **Books API** bằng một **worker thread** và hiển thị kết quả lên giao diện.
- Chỉnh sửa ứng dụng **"Who Wrote It?"** để sử dụng **AsyncTaskLoader** thay vì **AsyncTask**.

Tổng quan ứng dụng

Bạn sẽ xây dựng một ứng dụng có **EditText** và **Button**.

- Người dùng nhập tên sách vào **EditText** và nhấn **Button**.
- **Button** thực thi một **AsyncTask**, truy vấn **Google Books API** để tìm **tên sách** và **tác giả**.
- Kết quả sẽ được truy xuất và hiển thị trong một **TextView** bên dưới **Button**.
- Khi ứng dụng hoạt động ổn định, bạn sẽ chỉnh sửa ứng dụng để sử dụng **AsyncTaskLoader** thay vì **AsyncTask**.


3:45
Who Wrote It?
Enter a book name, or part of a book name, or just some text from a book to find the full book title and who wrote the book!
Enter a Book Title
SEARCH BOOKS
Romeo and Juliet
["William Shakespeare"]