

TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN



GIÁO TRÌNH
THỰC HÀNH PHÁT TRIỂN ỨNG DỤNG CHO THIẾT BỊ DI ĐỘNG

Hà Nội, 2.2025

MỤC LỤC

CHƯƠNG 1. Làm quen.....	3
Bài 1) Tạo ứng dụng đầu tiên	3
1.1) Android Studio và Hello World	3
1.2) Giao diện người dùng tương tác đầu tiên	5
1.3) Trình chỉnh sửa bố cục	5
1.4) Văn bản và các chế độ cuộn	5
1.5) Tài nguyên có sẵn.....	5
Bài 2) Activities	5
2.1) Activity và Intent	5
2.2) Vòng đời của Activity và trạng thái	5
2.3) Intent ngầm định.....	5
Bài 3) Kiểm thử, gỡ lỗi và sử dụng thư viện hỗ trợ	5
3.1) Trình gỡ lỗi	5
3.2) Kiểm thử đơn vị	5
3.3) Thư viện hỗ trợ	5
CHƯƠNG 2. Trải nghiệm người dùng	6
Bài 1) Tương tác người dùng	6
1.1) Hình ảnh có thể chọn	6
1.2) Các điều khiển nhập liệu	6
1.3) Menu và bộ chọn	6
1.4) Điều hướng người dùng	6
1.5) RecyclerView	6
Bài 2) Trải nghiệm người dùng thú vị	6
2.1) Hình vẽ, định kiểu và chủ đề	6
2.2) Thẻ và màu sắc	6

2.3) Bố cục thích ứng	6
Bài 3) Kiểm thử giao diện người dùng	6
3.1) Espresso cho việc kiểm tra UI	6
CHƯƠNG 3. Làm việc trong nền	6
Bài 1) Các tác vụ nền	6
1.1) AsyncTask	6
1.2) AsyncTask và AsyncTaskLoader	6
1.3) Broadcast receivers	6
Bài 2) Kích hoạt, lập lịch và tối ưu hóa nhiệm vụ nền	6
2.1) Thông báo	6
2.2) Trình quản lý cảnh báo	6
2.3) JobScheduler	6
CHƯƠNG 4. Lưu trữ dữ liệu người dùng	7
Bài 1) Tùy chọn và cài đặt	7
1.1) Shared preferences	7
1.2) Cài đặt ứng dụng	7
Bài 2) Lưu trữ dữ liệu với Room	7
2.1) Room, LiveData và ViewModel	7
2.2) Room, LiveData và ViewModel	7
3.1) Trinhf gowx loi	

CHƯƠNG 1. LÀM QUEN

Bài 1) Tạo ứng dụng đầu tiên

1.1) Android Studio và Hello World

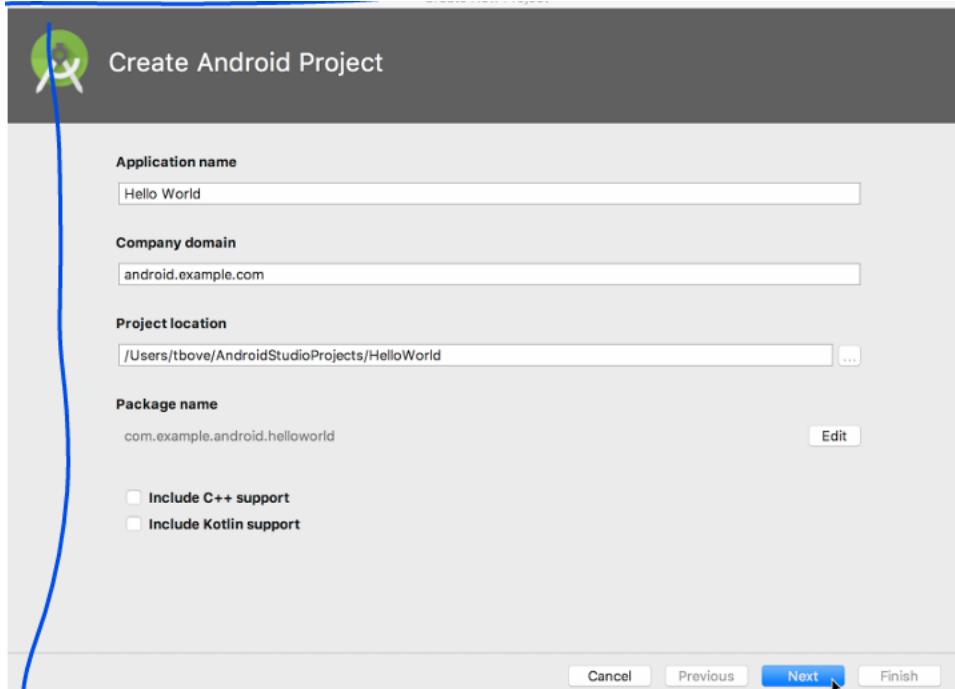
Giới thiệu

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

Những gì Bạn nên biết

Bạn nên có khả năng:

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



Những gì Bạn sẽ cần:

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

Những gì bạn sẽ học

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

Những gì bạn sẽ làm

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

1.2) Giao diện người dùng tương tác đầu tiên

1.3) Trình chỉnh sửa bố cục

1.4) Văn bản và các chế độ cuộn

1.5) Tài nguyên có sẵn

Bài 2) Activities

2.1) Activity và Intent

2.2) Vòng đời của Activity và trạng thái

2.3) Intent ngầm định

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

3.1) Trình gỡ lỗi

3.2) Kiểm thử đơn vị

3.3) Thư viện hỗ trợ

CHƯƠNG 2. TRẢI NGHIỆM NGƯỜI DÙNG

Bài 1) Tương tác người dùng

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

Bài 2) Trải nghiệm người dùng thú vị

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

Bài 3) Kiểm thử giao diện người dùng

- 3.1) Espresso cho việc kiểm tra UI**

CHƯƠNG 3. LÀM VIỆC TRONG NỀN

Bài 1) Các tác vụ nền

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

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

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

CHƯƠNG 4. LƯU DỮ LIỆU NGƯỜI DÙNG

Bài 1) Tùy chọn và cài đặt

1.1) Shared preferences

1.2) Cài đặt ứng dụng

Bài 2) Lưu trữ dữ liệu với Room

2.1) Room, LiveData và ViewModel

2.2) Room, LiveData và ViewModel

Bài 9.1: Share Preferences

Giới thiệu

Shared preferences cho phép bạn lưu trữ một lượng nhỏ dữ liệu nguyên thủy làm cặp khóa/giá trị trong một tệp trên thiết bị. Để có được xử lý tệp ưu tiên và để đọc, ghi và quản lý dữ liệu ưu tiên, sử dụng lớp SharedPreferences . Android framework quản lý chính tệp tùy chọn được chia sẻ. Tệp có thể truy cập được cho tất cả các thành phần của ứng dụng của bạn, nhưng không thể truy cập được cho các ứng dụng khác.

Dữ liệu bạn lưu vào các tùy chọn được chia sẻ khác với dữ liệu trong trạng thái hoạt động đã lưu, mà bạn đã học về một chương trước:

- Dữ liệu trong trạng thái hiện hoạt động đã lưu được giữ lại trong các trường hợp hoạt động trong cùng một người dùng phiên họp.
- Shared preferences vẫn tồn tại trong các phiên người dùng. Shared preferences vẫn tồn tại ngay cả khi ứng dụng của bạn dừng và khởi động lại, hoặc nếu thiết bị khởi động lại.

Chỉ sử dụng các Shared preferences khi bạn cần lưu một dữ liệu số lượng nhỏ dưới dạng các cặp khóa/giá trị đơn giản. Để quản lý số lượng lớn hơn của dữ liệu ứng dụng liên tục, hãy sử dụng phương pháp lưu trữ như thư viện phòng hoặc cơ sở dữ liệu SQL.

Những điều bạn nên biết

Bạn nên quen thuộc với:

- Tạo, xây dựng và chạy ứng dụng trong Android Studio.
- Thiết kế bố cục với các nút và chế độ xem văn bản.
- Sử dụng phong cách và chủ đề.
- Lưu và khôi phục trạng thái hiện hoạt động.

Những gì bạn sẽ học

Bạn sẽ học cách:

- Xác định sở thích chia sẻ là gì.
- Tạo một tệp tùy chọn được chia sẻ cho ứng dụng của bạn.
- Xóa dữ liệu trong các tùy chọn được chia sẻ.
- Cập nhật một ứng dụng để nó có thể lưu, truy xuất và đặt lại các tùy chọn được chia sẻ.

Tổng quan về ứng dụng

Ứng dụng HelloSharedPrefs là một biến thể khác của ứng dụng Hellotoast mà bạn đã tạo trong Bài 1. Nó bao gồm các nút để tăng số, để thay đổi màu nền và để đặt lại cả số lượng và màu sắc theo mặc định của họ. Ứng dụng cũng sử dụng các chủ đề và kiểu dáng để xác định các nút.

Bạn bắt đầu với ứng dụng khởi động và thêm các tùy chọn được chia sẻ vào mã hoạt động chính. Bạn cũng thêm một nút đặt lại đặt cả số đếm và màu nền cho mặc định và xóa tệp tùy chọn.

Nhiệm vụ 1: Khám phá Hellosharedprefs

Dự án ứng dụng khởi động hoàn chỉnh cho thực tế này có sẵn tại Hellosharedprefs Starter. Trong này

Nhiệm vụ bạn tải dự án vào Android Studio và khám phá một số tính năng chính của ứng dụng.

1.1 Mở và chạy dự án HelloSharedPrefs

1. Tải ứng dụng HelloSharedPrefs-Starter và giải nén tệp.
2. Mở dự án trong Android Studio, xây dựng và chạy ứng dụng.
 - Nhấp vào **Count** đếm để tăng số trong chế độ xem văn bản chính.

- Nhấp vào bất kỳ nút màu nào để thay đổi màu nền của chế độ xem văn bản chính.
 - Xoay thiết bị và lưu ý rằng cả màu nền và số lượng đều được bảo quản.
 - Nhấp vào nút **Reset** để đặt màu và đếm lại mặc định.
3. Buộc đóng ứng dụng bằng một trong những cách sau.
- Trong Android Studio, chọn **Run > Stop 'app'** hoặc nhấp vào **Biểu tượng Dừng** trên thanh công cụ.
 - Trên thiết bị, nhấn nút **Recents** (nút hình vuông ở góc dưới bên phải). Vuốt thẻ ứng dụng **HelloSharedPrefs** để thoát ứng dụng, hoặc nhấn vào dấu **X** ở góc phải của thẻ. Nếu bạn thoát ứng dụng theo cách này, hãy đợi vài giây trước khi khởi động lại để hệ thống có thời gian dọn dẹp.
4. Khởi động lại ứng dụng. Ứng dụng sẽ khởi động lại với giao diện mặc định—số đếm là 0 và màu nền là xám.

1.2 Khám phá code của Activity.

1. Mở MainActivity
2. Kiểm tra code và lưu ý những điều này:
 - Số đếm (**mCount**) được định nghĩa là một số nguyên. Phương thức **countUp()** trong sự kiện **onClick** tăng giá trị này lên và cập nhật **TextView** chính.
 - Màu sắc (**mColor**) cũng là một số nguyên, ban đầu được định nghĩa là màu xám trong tệp tài nguyên **colors.xml** với tên **default_background**.
 - Phương thức **changeBackground()** trong sự kiện **onClick** lấy màu nền của nút được nhấn, sau đó đặt màu đó cho **TextView** chính.
 - Cả hai số nguyên **mCount** và **mColor** đều được lưu vào gói trạng thái phiên bản trong phương thức **onSaveInstanceState()**, và được khôi phục trong **onCreate()**. Các khóa của gói cho số đếm và màu sắc được định nghĩa bởi các biến riêng (**COUNT_KEY**) và (**COLOR_KEY**).

Nhiệm vụ 2: Lưu và khôi phục dữ liệu vào tệp tùy chọn chia sẻ.

Trong nhiệm vụ này, bạn lưu trạng thái của ứng dụng vào một tệp tùy chọn chia sẻ và đọc lại dữ liệu đó khi ứng dụng được khởi động lại. Vì dữ liệu trạng thái mà bạn lưu vào tệp tùy chọn chia sẻ (số đếm hiện tại và màu sắc) là cùng dữ liệu mà bạn bảo toàn trong trạng thái phiên bản, bạn không cần phải thực hiện việc này hai lần. Bạn có thể thay thế hoàn toàn trạng thái phiên bản bằng trạng thái từ shared preference.

2

2.1 Khởi tạo preferences

- Để thêm các biến thành viên (member variables) vào lớp MainActivity nhằm lưu tên của tệp Shared Preferences và một tham chiếu đến đối tượng SharedPreferences,

```
private SharedPreferences mPreferences;  
private String sharedPrefFile =  
    "com.example.android.hellosharedprefs"; |
```

Bạn có thể đặt tên tệp Shared Preferences của mình bất kỳ như thế nào bạn muốn, nhưng theo thông lệ, nó thường có cùng tên với tên gói của ứng dụng của bạn.

- Trong phương thức onCreate(), hãy khởi tạo Shared Preferences. Chèn đoạn mã này trước câu lệnh if:

```
mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
```

Fương thức getSharedPreferences() (tù Context của hoạt động) mở tệp tại tên tệp đã cho (sharedPrefFile) với chế độ MODE_PRIVATE.

Ghi chú: Các phiên bản cũ của Android có các chế độ khác cho phép bạn tạo tệp Shared Preferences có thể đọc được hoặc ghi được bởi toàn bộ hệ thống. Những chế độ này đã bị deprecated (không khuyến khích sử dụng) từ API 17 và hiện nay bị phản đối mạnh mẽ vì lý do bảo mật. Nếu bạn cần chia sẻ dữ liệu với các ứng dụng khác, hãy cân nhắc sử dụng content URI được cung cấp bởi FileProvider.

Code giải pháp cho lớp MainActivity, một phần:

```
public class MainActivity extends AppCompatActivity {
    // Current count.
    no usages
    private int mCount = 0;
    no usages
    private int mColor;
    no usages
    private TextView mShowCountTextView;
    no usages
    private final String COUNT_KEY = "count";
    no usages
    private final String COLOR_KEY = "color";
    no usages
    private SharedPreferences mPreferences;
    no usages
    private String sharedPrefFile =
        "com.example.android.hellosharedprefs";
    no usages
    @Override
```

```

no usages
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Initialize views, color
    mShowCountTextView = (TextView) findViewById(R.id.count_textview);
    mColor = ContextCompat.getColor(this, R.color.default_background);
    mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
    // Restore the saved state. See onSaveInstanceState() for what gets saved
    if (savedInstanceState != null) {

        mCount = savedInstanceState.getInt(COUNT_KEY);
        if (mCount != 0) {
            mShowCountTextView.setText(String.format("%s", mCount));
        }

        mColor = savedInstanceState.getInt(COLOR_KEY);
        mShowCountTextView.setBackgroundColor(mColor);
    }
}

```

2.2 Lưu Lưu các tùy chọn trong onPause()

Việc lưu các tùy chọn rất giống với việc lưu trạng thái phiên bản -- cả hai thao tác đều lưu dữ liệu vào một đối tượng Bundle dưới dạng cặp khóa/giá trị. Tuy nhiên, đối với Shared Preferences, bạn lưu dữ liệu đó trong hàm gọi lại vòng đời onPause(), và bạn cần một đối tượng biên tập chia sẻ (SharedPreferences.Editor) để ghi vào đối tượng Shared Preferences.

1. Thêm phương thức vòng đời onPause() vào MainActivity

```

@Override
protected void onPause(){
    super.onPause();
    // ...
}

```

2. Trong onPause(), lấy một đối tượng biên tập (Editor) cho đối tượng SharedPreferences:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
// ...
```

Một trình biên tập Shared Preferences là cần thiết để ghi vào đối tượng Shared Preferences. Thêm dòng này vào `onPause()` sau lời gọi đến `super.onPause()`.

3. Sử dụng phương thức `putInt()` để lưu cả hai số nguyên `mCount` và `mColor` vào Shared Preferences với các khóa phù hợp:

```
preferencesEditor.putInt(COUNT_KEY, mCount);
preferencesEditor.putInt(COLOR_KEY, mColor);
// ...
```

4. Gọi apply() để lưu các tùy chọn:

```
preferencesEditor.apply();
```

Phương thức `apply()` lưu các tùy chọn một cách bất đồng bộ, ngoài luồng giao diện người dùng (UI thread). Trình biên tập Shared Preferences cũng có phương thức `commit()` để lưu các tùy chọn một cách đồng bộ. Phương thức `commit()` không được khuyến khích vì nó có thể chặn các hoạt động khác.

5. Xóa toàn bộ phương thức `onSaveInstanceState()`. Vì trạng thái phiên bản của hoạt động chứa cùng dữ liệu với Shared Preferences, bạn có thể thay thế hoàn toàn trạng thái phiên bản.

Mã giải pháp cho phương thức `onPause()` của `MainActivity` :

```
@Override
protected void onPause(){
    super.onPause();
    SharedPreferences.Editor preferencesEditor = mPreferences.edit();
    preferencesEditor.putInt(COUNT_KEY, mCount);
    preferencesEditor.putInt(COLOR_KEY, mColor);
    preferencesEditor.apply();
}
```

2.3 Khôi phục tùy chỉnh trong onCreate()

Cũng giống như trạng thái phiên làm việc, ứng dụng của bạn sẽ đọc bất kỳ tùy chỉnh đã lưu nào trong phương thức `onCreate()`. Một lần nữa, vì các tùy chỉnh dùng chung chứa cùng dữ liệu với trạng thái phiên làm việc, chúng ta có thể thay thế trạng thái bằng các tùy chỉnh ở đây. Mỗi khi `onCreate()` được gọi – khi ứng dụng khởi động hoặc khi có thay đổi cấu hình – các tùy chỉnh dùng chung sẽ được sử dụng để khôi phục trạng thái của giao diện.

- Xác định phần của phương thức `onCreate()` kiểm tra xem đối số `savedInstanceState` có bằng `null` hay không và khôi phục trạng thái phiên làm việc.

```
if (savedInstanceState != null) {  
  
    mCount = savedInstanceState.getInt(COUNT_KEY);  
    if (mCount != 0) {  
        mShowCountTextView.setText(String.format("%s", mCount));  
    }  
  
    mColor = savedInstanceState.getInt(COLOR_KEY);  
    mShowCountTextView.setBackgroundColor(mColor);  
}
```

- Xóa toàn bộ khối
- Trong phương thức `onCreate()`, tại cùng vị trí nơi mã trạng thái phiên (`instance state`) đã được sử dụng, hãy lấy giá trị đếm từ `SharedPreferences` bằng khóa `COUNT_KEY` và gán nó cho biến `mCount`.

```
mCount = mPreferences.getInt(COUNT_KEY, 0);  
// Replaces the saved state. See onSaveInstanceState() for
```

Khi bạn đọc dữ liệu từ `SharedPreferences`, bạn không cần lấy một `SharedPreferences.Editor`. Hãy sử dụng bất kỳ phương thức "get" nào trên đối tượng `SharedPreferences` (chẳng hạn như `getInt()` hoặc `getString()`) để truy xuất dữ liệu từ `SharedPreferences`.

Lưu ý rằng phương thức `getInt()` nhận hai đối số: một là khóa để truy xuất giá trị, và một là giá trị mặc định nếu không tìm thấy khóa đó. Trong trường hợp này, giá trị mặc định là 0, giống với giá trị khởi tạo của `mCount`.

4. Cập nhật giá trị của TextView chính với số đếm mới.

```
mShowCountTextView.setText(String.format("%s", mCount));
```

5. Lấy màu từ SharedPreferences bằng khóa COLOR_KEY và gán nó cho biến mColor.

```
mColor = mPreferences.getInt(COLOR_KEY, mColor);
```

Như trước đây, đối số thứ hai của getInt() là giá trị mặc định được sử dụng trong trường hợp khóa không tồn tại trong SharedPreferences. Trong trường hợp này, bạn có thể chỉ cần sử dụng lại giá trị của mColor, vì nó vừa được khởi tạo với màu nền mặc định ở phần trên của phương thức.

6. Cập nhật màu nền của TextView chính.

```
mShowCountTextView.setBackgroundColor(mColor);
```

7. Chạy ứng dụng. Nhấn nút **Count** và thay đổi màu nền để cập nhật trạng thái phiên (instance state) và SharedPreferences.
8. Xoay thiết bị hoặc trình giả lập để kiểm tra rằng số đếm và màu sắc được lưu lại sau khi thay đổi cấu hình.
9. Buộc đóng ứng dụng bằng một trong các phương pháp sau:
 - Trong Android Studio, chọn **Run > Stop 'app.'**
 - Trên thiết bị, nhấn nút **Gần đây** (nút hình vuông ở góc dưới bên phải). Vuốt thẻ ứng dụng **HelloSharedPrefs** để đóng ứng dụng hoặc nhấn **X** ở góc phải của thẻ.
10. Chạy lại ứng dụng. Ứng dụng sẽ khởi động lại và tải SharedPreferences, duy trì trạng thái.

Code giải pháp cho phương thức `onCreate()` trong `MainActivity`:

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // Initialize views, color  
    mShowCountTextView = (TextView) findViewById(R.id.count_textview);  
    mColor = ContextCompat.getColor(this, R.color.default_background);  
    mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);  
  
    mCount = mPreferences.getInt(COUNT_KEY, 0);  
    mShowCountTextView.setText(String.format("%s", mCount));  
    mColor = mPreferences.getInt(COLOR_KEY, mColor);  
    mShowCountTextView.setBackgroundColor(mColor);
```

2.4 Đặt lại SharedPreferences trong trình xử lý sự kiện nhấp reset().

Nút đặt lại trong ứng dụng ban đầu sẽ đặt lại cả số đếm và màu sắc của hoạt động về giá trị mặc định. Vì SharedPreferences lưu trạng thái của hoạt động, nên cũng cần xóa SharedPreferences cùng lúc.

- Trong phương thức reset() của sự kiện onClick, sau khi màu sắc và số đếm được đặt lại, hãy lấy một Editor cho đối tượng SharedPreferences:

```
SharedPreferences.Editor preferencesEditor = mPreferences.edit();
```

- Xóa tất cả dữ liệu trong SharedPreferences.

```
preferencesEditor.clear();
```

- Áp dụng các thay đổi.

Code giải pháp cho phương thức `reset()`:

```
no usages
public void reset(View view) {
    // Reset count
    mCount = 0;
    mShowCountTextView.setText(String.format("%s", mCount));

    // Reset color
    mColor = ContextCompat.getColor(this, R.color.default_background);
    mShowCountTextView.setBackgroundColor(mColor);

    SharedPreferences.Editor preferencesEditor = mPreferences.edit();
    preferencesEditor.clear();
    preferencesEditor.apply();

}
```

CODE GIẢI PHÁP

Dự án Android Studio: **HelloSharedPrefs**

Thử thách lập trình

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

Thử thách: Hãy chỉnh sửa ứng dụng **HelloSharedPrefs** sao cho thay vì tự động lưu trạng thái vào tệp `SharedPreferences`, bạn sẽ thêm một **Activity** thứ hai để thay đổi, đặt lại và lưu các tùy chọn. Thêm một nút có tên **Settings** để mở activity này. Trong activity mới, sử dụng các **ToggleButton** và **Spinner** để chỉnh sửa tùy chọn. Thêm hai nút **Save** và **Reset** để lưu hoặc xóa `SharedPreferences`.

Tóm tắt

- Lớp **SharedPreferences** cho phép ứng dụng lưu trữ một lượng nhỏ dữ liệu kiểu nguyên thủy dưới dạng cặp **key-value**.
- **SharedPreferences** được duy trì qua các phiên làm việc khác nhau của cùng một ứng dụng.
- Để ghi dữ liệu vào **SharedPreferences**, cần lấy một đối tượng **SharedPreferences.Editor**.
- Sử dụng các phương thức "put" trong **SharedPreferences.Editor**, chẳng hạn như `.putInt()` hoặc `putString()`, để lưu dữ liệu vào **SharedPreferences** với một **key** và một **value**.

- Sử dụng các phương thức "get" trong đối tượng **SharedPreferences**, chẳng hạn như `getInt()` hoặc `getString()`, để lấy dữ liệu từ **SharedPreferences** bằng một **key**.
- Sử dụng phương thức `clear()` trong đối tượng **SharedPreferences.Editor** để xóa toàn bộ dữ liệu được lưu trong **SharedPreferences**.
- Sử dụng phương thức `apply()` trong đối tượng **SharedPreferences.Editor** để lưu các thay đổi vào tệp **SharedPreferences**.

Khái niệm liên quan

Tài liệu về khái niệm liên quan có trong:

- **9.0: Lưu trữ dữ liệu**
- **9.1: SharedPreferences**
- **Tìm hiểu thêm**

Tài liệu dành cho nhà phát triển Android:

- **Tổng quan về lưu trữ dữ liệu và tệp**
- **Lưu dữ liệu dưới dạng cặp khóa-giá trị**
- **SharedPreferences**
- **SharedPreferences.Editor**

Stack Overflow:

- **Cách sử dụng SharedPreferences trong Android để lưu trữ, truy xuất và chỉnh sửa giá trị**
- **onSavedInstanceState vs. SharedPreferences**

Bài tập về nhà

Xây dựng và chạy ứng dụng

Mở ứng dụng **ScoreKeeper** mà bạn đã tạo trong bài học **Android Fundamentals 5.1: Drawables, styles, and themes**.

1. Thay thế trạng thái phiên đã lưu bằng **SharedPreferences** cho từng điểm số.
2. Để kiểm tra ứng dụng, xoay thiết bị để đảm bảo rằng khi cấu hình thay đổi, ứng dụng đọc dữ liệu từ **SharedPreferences** và cập nhật giao diện người dùng.
3. Dùng ứng dụng và khởi động lại để kiểm tra xem dữ liệu có được lưu trong **SharedPreferences** hay không.
4. Thêm nút **Reset** để đặt lại điểm số về **0** và xóa dữ liệu trong **SharedPreferences**.

- **Trả lời các câu hỏi**

Câu hỏi 1:

Trong phương thức vòng đời nào bạn lưu trạng thái ứng dụng vào **SharedPreferences**?

Câu hỏi 2:

Trong phương thức vòng đời nào bạn khôi phục trạng thái ứng dụng?

Câu hỏi 3:

Bạn có thể nghĩ ra trường hợp nào mà việc sử dụng cả **SharedPreferences** và **InstanceState** là hợp lý không?

- **Nộp ứng dụng để chấm điểm**

Hướng dẫn cho người chấm điểm

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

- Ứng dụng giữ nguyên điểm số khi thiết bị xoay.
- Ứng dụng giữ nguyên điểm số hiện tại sau khi bị dừng và khởi động lại.
- Ứng dụng lưu điểm số vào **SharedPreferences** trong phương thức onPause().
- Ứng dụng khôi phục **SharedPreferences** trong phương thức onCreate().
- Ứng dụng có một nút **Reset** để đặt lại điểm số về **0**.

Đảm bảo rằng phương thức xử lý sự kiện onClick của nút Reset thực hiện các tác vụ sau:

- Đặt cả hai biến điểm số về **0**.
- Cập nhật cả hai **TextView**.
- Xóa dữ liệu trong **SharedPreferences**.

Bài học 9.2: Cài đặt ứng dụng

Giới thiệu

Các ứng dụng thường bao gồm **cài đặt** để cho phép người dùng tùy chỉnh tính năng và hành vi của ứng dụng. Ví dụ:

Một số ứng dụng cho phép người dùng đặt **vị trí nhà**, Chọn **đơn vị đo lường mặc định**, Và các tùy chọn khác ảnh hưởng đến toàn bộ ứng dụng. Người dùng không thường xuyên truy cập cài đặt, vì sau khi thiết lập (chẳng hạn như vị trí nhà), họ hiếm khi cần thay đổi lại.

Người dùng mong đợi có thể truy cập **cài đặt ứng dụng** bằng cách: Nhấn vào mục **Settings** trong **thanh điều hướng bên** (navigation drawer) – như hình bên trái. Hoặc trong **menu tùy chọn** trên **thanh ứng dụng** (app bar) – như hình bên phải.

Trong hình trên:

1. **Settings** trong **thanh điều hướng bên** (navigation drawer).
2. **Settings** trong **menu tùy chọn** của **thanh ứng dụng** (app bar).

Trong bài thực hành này, bạn sẽ thêm một Settings Activity vào ứng dụng. Người dùng có thể điều hướng đến **cài đặt ứng dụng** bằng cách nhấn vào **Settings**, được đặt trong **menu tùy chọn** trên **thanh ứng dụng** (app bar).

Những kiến thức bạn cần biết trước

Bạn nên có khả năng:

- Tạo một **dự án Android Studio** từ mẫu và tạo bộ cục chính.
- Chạy ứng dụng trên **trình giả lập** hoặc **thiết bị kết nối**.
- Tạo và chỉnh sửa các phần tử giao diện người dùng (UI) bằng **layout editor** và mã XML.
- Trích xuất tài nguyên chuỗi và chỉnh sửa giá trị chuỗi.
- Truy cập các phần tử giao diện từ mã bằng **findViewById()**.
- Xử lý sự kiện **nhấn nút (Button click)**.
- Hiển thị **Toast message**.
- Thêm một **Activity** vào ứng dụng.
- Tạo **menu tùy chọn** trên **thanh ứng dụng** (app bar).
- Thêm và chỉnh sửa **các mục menu** trong **menu tùy chọn**.
- Sử dụng **styles và themes** trong dự án.
- Sử dụng **SharedPreferences**.

Bạn sẽ học được gì?

Bạn sẽ học cách:

- Thêm một **Fragment** để quản lý **cài đặt**.
- Tạo một **tệp tài nguyên XML** chứa các cài đặt và thuộc tính của chúng.
- Tạo điều hướng đến **Settings Activity**.
- Đặt **giá trị mặc định** cho cài đặt.
- Đọc **giá trị cài đặt** do người dùng thay đổi.
- Tùy chỉnh **mẫu Settings Activity**.

Những gì bạn sẽ làm

- Tạo một ứng dụng có mục **Settings** trong **menu tùy chọn**.

- Thêm một **công tắc bật/tắt** (toggle switch) cho tùy chọn **Settings**.
- Thêm mã để đặt giá trị mặc định cho cài đặt và truy cập giá trị cài đặt sau khi nó đã thay đổi.
- Sử dụng và tùy chỉnh mẫu **Settings Activity** của Android Studio.

Tổng quan về ứng dụng

Android Studio cung cấp một phím tắt để thiết lập menu tùy chọn với **Cài đặt**. Nếu bạn bắt đầu một dự án Android Studio cho điện thoại hoặc máy tính bảng bằng mẫu **Basic Activity**, ứng dụng mới sẽ bao gồm **Cài đặt**, như được hiển thị bên dưới:

 ảnh

Mẫu này cũng bao gồm một nút hành động nổi (**Floating Action Button**) ở góc dưới bên phải của màn hình với biểu tượng phong bì. Bạn có thể bỏ qua nút này trong bài thực hành này, vì bạn sẽ không sử dụng nó.

Bạn sẽ bắt đầu bằng cách tạo một ứng dụng có tên **AppWithSettings** bằng mẫu **Basic Activity**, sau đó thêm một **Settings Activity**, cung cấp một cài đặt công tắc bật/tắt mà người dùng có thể chuyển đổi giữa bật hoặc tắt.

Bạn sẽ thêm mã để đọc giá trị cài đặt và thực hiện một hành động dựa trên giá trị đó. Để đơn giản, hành động này sẽ là hiển thị một thông báo **Toast** với giá trị của cài đặt.

Trong nhiệm vụ thứ hai, bạn sẽ thêm mẫu **Settings Activity** tiêu chuẩn do Android Studio cung cấp vào ứng dụng **DroidCafeOptionsUp** mà bạn đã tạo trong bài học trước.

Mẫu **Settings Activity** được điền sẵn các cài đặt mà bạn có thể tùy chỉnh cho ứng dụng và cung cấp một bố cục khác nhau cho điện thoại và máy tính bảng.

- Điện thoại: Một màn hình **Cài đặt chính** với liên kết tiêu đề cho từng nhóm cài đặt, chẳng hạn như **General** cho cài đặt chung, như hiển thị bên dưới.

 ảnh

- Máy tính bảng: Một bố cục màn hình **chính/chi tiết**, với liên kết tiêu đề cho từng nhóm cài đặt ở bên trái (màn hình chính) và nhóm cài đặt tương ứng ở bên phải (màn hình chi tiết), như hiển thị trong hình bên dưới.

 ảnh

Để tùy chỉnh mẫu, bạn sẽ thay đổi tiêu đề, tiêu đề cài đặt, mô tả cài đặt và giá trị của các cài đặt.

Ứng dụng **DroidCafeOptionsUp** đã được tạo trong bài học trước từ mẫu **Basic Activity**, mẫu này cung cấp một menu tùy chọn trên thanh ứng dụng để đặt tùy chọn **Cài đặt**.

Bạn sẽ tùy chỉnh mẫu **Settings Activity** được cung cấp bằng cách thay đổi tiêu đề, mô tả, giá trị và giá trị mặc định của một cài đặt. Bạn cũng sẽ thêm mã để đọc giá trị cài đặt sau khi người dùng thay đổi nó và hiển thị giá trị đó.

Nhiệm vụ 1: Thêm một cài đặt công tắc (Switch Setting) vào ứng dụng

Trong nhiệm vụ này, bạn sẽ thực hiện các bước sau:

- Tạo một dự án mới dựa trên mẫu Basic Activity, mẫu này cung cấp một menu tùy chọn.
- Thêm một công tắc bật/tắt (SwitchPreference) với các thuộc tính trong tệp XML preference.
- Thêm một Activity cho cài đặt và một Fragment cho một cài đặt cụ thể. Để duy trì khả năng tương thích với AppCompatActivity, bạn sẽ sử dụng PreferenceFragmentCompat thay vì PreferenceFragment. Bạn cũng cần thêm thư viện android.support.v7.preference.
- Kết nối mục Cài đặt (Settings) trong menu tùy chọn với Settings Activity.

1.1 Tạo dự án và thêm thư mục XML cùng tệp tài nguyên

1. Trong **Android Studio**, tạo một dự án mới với các tham số sau:

Thuộc tính	Giá trị
Tên ứng dụng	AppWithSettings
Tên công ty	android.example.com (hoặc tên miền của bạn)
Vị trí dự án (Project location)	Đường dẫn đến thư mục chứa dự án của bạn
SDK tối thiểu cho điện thoại và máy tính bảng	API 15: Android 4.0.3 IceCreamSandwich
Mẫu	Basic Activity
Tên Activity	MainActivity
Tên Layout	activity_main
Tiêu đề	MainActivity

2. Chạy ứng dụng và nhấn vào biểu tượng **overflow** trên thanh ứng dụng để xem menu tùy chọn, như hiển thị trong hình bên dưới. Mục duy nhất trong menu tùy chọn là **Settings**.

anh

3. Bạn cần tạo một thư mục tài nguyên mới để chứa tệp XML cài đặt. Chọn thư mục **res** trong bảng **Project > Android**, sau đó chọn **File > New > Android Resource Directory**. Hộp thoại **New Resource Directory** sẽ xuất hiện.
4. Trong menu thả xuống **Resource type**, chọn **xml**. **Tên thư mục** sẽ tự động thay đổi thành **xml**. Nhấn **OK**.
5. Thư mục **xml** sẽ xuất hiện trong bảng **Project > Android** bên trong thư mục **res**. Chọn **xml** và chọn **File > New > XML resource file** (hoặc nhấp chuột phải vào **xml** và chọn **New > XML resource file**).
6. Nhập tên tệp XML là **preferences** vào trường **File name**, sau đó nhấn **OK**. Tệp **preferences.xml** sẽ xuất hiện trong thư mục **xml**, và trình chỉnh sửa giao diện (**layout editor**) sẽ hiển thị, như trong hình bên dưới.

Ảnh trang 23

Trong hình trên:

1. Tệp **preferences.xml** bên trong thư mục **xml**.
2. Trình chỉnh sửa giao diện (**layout editor**) hiển thị nội dung của **preferences.xml**.

1.2. Thêm Preference XML và các thuộc tính cho cài đặt.

1. Kéo một SwitchPreference từ bảng Palette ở bên trái vào đầu bố cục, như hiển thị trong hình bên dưới.

Ảnh

2. Thay đổi các giá trị trong bảng **Attributes** ở bên phải của trình chỉnh sửa giao diện như sau, và như hiển thị trong hình bên dưới:

- **defaultValue: true**
- **key: example_switch**
- **title: Settings option**
- **summary: Turn this option on or off**

Ảnh

3. Nhấp vào tab **Text** ở dưới cùng của trình chỉnh sửa giao diện để xem mã XML.

anh

4. Trích xuất tài nguyên chuỗi cho các giá trị thuộc tính **android:title** và **android:summary** thành: @string/switch_title cho **android:title** và @string/switch_summary cho **android:summary**

Các thuộc tính XML cho một **Preference** bao gồm:

- **android:defaultValue**: Giá trị mặc định của cài đặt khi ứng dụng khởi chạy lần đầu tiên.
- **android:title**: Tiêu đề của cài đặt. Đối với **SwitchPreference**, tiêu đề xuất hiện bên trái của công tắc bật/tắt.
- **android:key**: Khóa dùng để lưu trữ giá trị cài đặt. Mỗi cài đặt có một cặp khóa-giá trị tương ứng mà hệ thống sử dụng để lưu trữ cài đặt trong tệp **SharedPreferences** mặc định của ứng dụng.
- **android:summary**: Văn bản mô tả xuất hiện bên dưới cài đặt.

1.3 Sử dụng SwitchPreferenceCompat

Để sử dụng **PreferenceFragmentCompat** thay cho **PreferenceFragment**, bạn cũng phải sử dụng phiên bản **android.support.v7** của **SwitchPreference** (**SwitchPreferenceCompat**).

1. Trong bảng **Project > Android**, mở tệp **build.gradle (Module: app)** trong thư mục **Gradle Scripts**, và thêm dòng sau vào phần **dependencies**:

anh

Câu lệnh trên thêm thư viện **android.support.v7.preference** để sử dụng **PreferenceFragmentCompat** thay cho **PreferenceFragment**.

2. Trong tệp **preferences.xml** trong thư mục **xml**, thay đổi `<SwitchPreference` trong mã thành `<android.support.v7.preference.SwitchPreferenceCompat`.

anh

Dòng **SwitchPreferenceCompat** ở trên có thể hiển thị biểu tượng bóng đèn màu vàng với cảnh báo, nhưng bạn có thể bỏ qua nó ngay bây giờ.

3. Mở tệp **styles.xml** trong thư mục **values**, và thêm mục **preferenceTheme** sau vào phần khai báo **AppTheme**:

anh

Để sử dụng **PreferenceFragmentCompat**, bạn cũng phải khai báo **preferenceTheme** với kiểu **PreferenceThemeOverlay** trong chủ đề của ứng dụng.

1.4 Thêm một Activity cho cài đặt

Để tạo một **Settings Activity** cung cấp giao diện người dùng cho cài đặt, hãy thêm một **Empty Activity** vào ứng dụng. Thực hiện các bước sau:

1. Chọn **app** ở đầu bảng **Project > Android**, sau đó chọn **New > Activity > Empty Activity**.
2. Đặt tên **Activity** là **SettingsActivity**. Bỏ chọn tùy chọn **Generate Layout File** (vì bạn không cần tệp giao diện), và giữ nguyên tùy chọn **Launcher Activity** ở trạng thái bỏ chọn.
3. Giữ nguyên tùy chọn **Backwards Compatibility (AppCompat)** được chọn. **Package name** sẽ tự động được đặt thành **com.example.android.projectname**.
4. Chọn **kết thúc**.

1.5 Thêm một Fragment cho một cài đặt cụ thể

Một **Fragment** giống như một phần độc lập của **Activity** — nó có vòng đời riêng, nhận các sự kiện đầu vào riêng và có thể được thêm hoặc xóa trong khi **Activity** đang chạy.

Bạn sẽ sử dụng một lớp con chuyên biệt của **Fragment** để hiển thị danh sách cài đặt. Cách tốt nhất là sử dụng một **Activity** thông thường để chứa một **PreferenceFragment**, hiển thị cài đặt của ứng dụng. **PreferenceFragment** cung cấp một kiến trúc linh hoạt hơn so với việc sử dụng **Activity** để quản lý cài đặt.

Bạn sẽ sử dụng **PreferenceFragmentCompat** thay vì **PreferenceFragment** để đảm bảo khả năng tương thích với **AppCompatActivity**.

Trong bước này, bạn sẽ thêm một **Blank Fragment** để nhóm các cài đặt tương tự (không có layout, phương thức factory hoặc interface callbacks) vào ứng dụng và mở rộng từ **PreferenceFragmentCompat**.

Thực hiện các bước sau:

1. Chọn **app**, sau đó chọn **New > Fragment > Fragment (Blank)**.
2. Đặt tên **Fragment** là **SettingsFragment**. Bỏ chọn tùy chọn **Create layout XML?** (vì bạn không cần tệp giao diện).
3. Bỏ chọn các tùy chọn để thêm **fragment factory methods** và **interface callbacks**.
4. **Target Source Set** nên được đặt thành **main**.
5. Nhấn **Finish**. Kết quả là lớp **SettingsFragment** sau sẽ được tạo:

ảnh

- Chỉnh sửa phần định nghĩa lớp **SettingsFragment** để mở rộng từ **PreferenceFragmentCompat**.

ảnh

Khi bạn chỉnh sửa định nghĩa lớp để khớp với định nghĩa ở trên, một biểu tượng bóng đèn đỏ sẽ xuất hiện ở lề bên trái. Nhấp vào bóng đèn đỏ và chọn **Implement methods**, sau đó chọn **onCreatePreferences**. Android Studio sẽ tạo ra một phương thức **onCreatePreferences()** mẫu như sau:

ảnh

Để mở rộng **Fragment**, Android Studio sẽ thêm câu lệnh **import** sau:

ảnh

- Xóa toàn bộ phương thức **onCreateView()** trong **Fragment**.

Lý do bạn thay thế **onCreateView()** bằng **onCreatePreferences()** là vì bạn sẽ thêm **SettingsFragment** vào **SettingsActivity** hiện có để hiển thị cài đặt, thay vì hiển thị một màn hình **Fragment** riêng biệt. Việc thêm **Fragment** vào **Activity** giúp bạn dễ dàng thêm hoặc xóa **Fragment** trong khi **Activity** đang chạy. **PreferenceFragment** được gắn vào **PreferenceScreen** bằng **rootKey**.

Bạn cũng có thể xóa toàn constructor rỗng của **SettingsFragment**, vì **Fragment** này không được hiển thị độc lập.

ảnh

- Bạn cần liên kết **Fragment** này với tệp tài nguyên cài đặt **preferences.xml** mà bạn đã tạo ở bước trước. Thêm vào phương thức **onCreatePreferences()** vừa tạo một lệnh gọi đến **setPreferencesFromResource()**, truyền vào **ID của tệp XML** (`R.xml.preferences`) và **rootKey** để xác định **gốc của PreferenceScreen**.

ảnh

Phương thức **onCreatePreferences()** bây giờ sẽ trông như thế này:

ảnh

1.6 Hiển thị Fragment trong SettingsActivity

Để hiển thị **Fragment** trong **SettingsActivity**, hãy làm theo các bước sau:

1. Mở **SettingsActivity**.
2. Thêm đoạn mã sau vào cuối phương thức `onCreate()` để hiển thị **Fragment** làm nội dung chính:

ảnh

3.

Đoạn mã trên sử dụng mẫu thông thường để thêm **Fragment** vào **Activity**, giúp Fragment hiển thị làm nội dung chính của Activity:

- Sử dụng `getFragmentManager()` nếu lớp kế thừa từ **Activity** và **Fragment** kế thừa từ **PreferenceFragment**.
- Sử dụng `getSupportFragmentManager()` nếu lớp kế thừa từ **AppCompatActivity** và **Fragment** kế thừa từ **PreferenceFragmentCompat**.

Phương thức `onCreate()` hoàn chỉnh trong **SettingsActivity** bây giờ sẽ trông như sau:

ảnh

1.7 Kết nối mục Settings trong menu với SettingsActivity

Sử dụng **Intent** để khởi chạy **SettingsActivity** từ **MainActivity** khi người dùng chọn **Settings** trong menu tùy chọn.

1. Mở **MainActivity** và tìm khôi `if` trong phương thức `onOptionsItemSelected()`, nơi xử lý sự kiện nhấn vào **Settings** trong menu tùy chọn:

ảnh

2. Thêm một **Intent** vào khôi `if` để khởi chạy **SettingsActivity**.

ảnh

3. Để thêm nút **Up** trên **thanh ứng dụng** (app bar) vào **SettingsActivity**, bạn cần chỉnh sửa khai báo của nó trong tệp **AndroidManifest.xml** để xác định **MainActivity** là **Activity cha** của **SettingsActivity**. Mở **AndroidManifest.xml** và tìm khai báo của **SettingsActivity**:

ảnh

Thay đổi khai báo thành như sau:

ảnh

- Chạy ứng dụng. Nhấn vào **biểu tượng dấu ba chấm** để mở **menu tùy chọn**, như trong hình bên trái. Nhấn vào **Settings** để mở **SettingsActivity**, như trong hình ở giữa. Nhấn vào **nút Up** trên **thanh ứng dụng** của **SettingsActivity**, như trong hình bên phải, để quay lại **MainActivity**.

ảnh

1.8 Lưu giá trị mặc định vào SharedPreferences

Mặc dù giá trị mặc định cho **công tắc bật/tắt (toggle switch)** đã được đặt trong thuộc tính `android:defaultValue` (ở **Bước 1.2** của nhiệm vụ này), ứng dụng vẫn cần lưu giá trị mặc định vào tệp **SharedPreferences** cho mỗi cài đặt khi người dùng mở ứng dụng lần đầu tiên.

Thực hiện các bước sau để đặt giá trị mặc định cho **toggle switch**:

- Mở **MainActivity**.
- Thêm đoạn mã sau vào cuối phương thức `onCreate()`, sau phần mã của **FloatingActionButton**.

ảnh

Đoạn mã trên đảm bảo rằng các cài đặt được khởi tạo đúng với giá trị mặc định của chúng.

Phương thức `PreferenceManager.setDefaultValues()` nhận **ba đối số**:

- Context** của ứng dụng, chẳng hạn như `this`.
- ID tài nguyên (preferences)** của tệp XML chứa một hoặc nhiều cài đặt.
- Một **giá trị boolean** xác định liệu các giá trị mặc định có nên được thiết lập nhiều lần hay không: Nếu `false`, hệ thống chỉ thiết lập các giá trị mặc định nếu phương thức này chưa từng được gọi trước đó. Điều này có nghĩa là bạn có thể gọi phương thức này mỗi khi **MainActivity** khởi động mà không ghi đè lên các giá trị đã lưu của người dùng. Nếu `true`, phương thức sẽ **ghi đè** bất kỳ giá trị nào trước đó bằng các giá trị mặc định.

1.9 Đọc giá trị cài đặt đã thay đổi từ SharedPreferences

Khi ứng dụng khởi động, phương thức `onCreate()` của **MainActivity** có thể đọc các giá trị cài đặt đã thay đổi và sử dụng chúng thay vì các giá trị mặc định.

Mỗi cài đặt được xác định bằng **cặp khóa-giá trị**. Hệ thống Android sử dụng cặp khóa-giá trị này khi lưu hoặc truy xuất cài đặt từ tệp **SharedPreferences** của ứng dụng. Khi người dùng thay đổi một cài đặt, hệ thống sẽ cập nhật giá trị tương ứng trong tệp **SharedPreferences**.

Để sử dụng giá trị của cài đặt, ứng dụng có thể dùng **khóa (key)** để lấy cài đặt từ tệp **SharedPreferences** bằng phương thức

```
PreferenceManager.getDefaultSharedPreferences().
```

Thực hiện các bước sau để thêm đoạn mã này:

1. Mở **SettingsActivity** và tạo một biến **String** tĩnh để lưu **khóa** của giá trị.

ảnh

2. Mở **MainActivity** và thêm đoạn mã sau vào cuối phương thức **onCreate()**:

ảnh

3. Chạy ứng dụng và nhấn vào **Settings** để mở **SettingsActivity**.
4. Nhấn vào tùy chọn cài đặt để chuyển **toggle** từ **bật** sang **tắt**, như trong hình bên trái dưới đây.
5. Nhấn nút **Up** trong **SettingsActivity** để quay lại **MainActivity**. Một thông báo **Toast** sẽ xuất hiện trong **MainActivity** với giá trị của cài đặt, như trong hình bên phải dưới đây.
6. Lặp lại các bước này để xem thông báo **Toast** thay đổi khi bạn thay đổi cài đặt.

Đoạn mã trên sử dụng các thành phần sau:

- android.support.v7.preference.PreferenceManager.getDefaultSharedPreferences(this) để lấy cài đặt dưới dạng một đối tượng **SharedPreferences** (**sharedPref**).
- getBoolean() để lấy giá trị **Boolean** của cài đặt sử dụng khóa (**KEY_PREF_EXAMPLE_SWITCH** được định nghĩa trong **SettingsActivity**) và gán nó cho biến **switchPref**. Nếu không có giá trị nào cho khóa này, phương thức getBoolean() sẽ đặt giá trị của **switchPref** thành **false**. Đối với các kiểu dữ liệu khác như **chuỗi (String)**, **số nguyên (Integer)**, **hoặc số thực (Floating point)**, bạn có thể sử dụng tương ứng các phương thức getString(), getInt(), hoặc getFloat().
- Toast.makeText() và show() để hiển thị giá trị của cài đặt **switchPref**.

Mỗi khi **MainActivity** khởi động hoặc khởi động lại, phương thức `onCreate()` sẽ đọc các giá trị cài đặt để sử dụng trong ứng dụng. Phương thức `Toast.makeText()` sau này có thể được thay thế bằng một phương thức khởi tạo cài đặt.

Bây giờ, ứng dụng của bạn đã có một **SettingsActivity** hoạt động hoàn chỉnh.

Mã nguồn giải pháp Task 1

Dự án Android Studio: AppWithSettings

Nhiệm vụ 2: Sử dụng mẫu Settings Activity

Nếu bạn cần xây dựng nhiều màn hình con cho cài đặt và muốn tận dụng kích thước màn hình máy tính bảng, đồng thời duy trì khả năng tương thích với các phiên bản Android cũ hơn dành cho máy tính bảng, Android Studio cung cấp một lối tắt: mẫu Settings Activity.

Trong nhiệm vụ trước, bạn đã học cách sử dụng một Settings Activity trống và một Fragment trống để thêm cài đặt vào ứng dụng. Nhiệm vụ 2 sẽ hướng dẫn bạn cách sử dụng mẫu Settings Activity có sẵn trong Android Studio để:

- Chia nhiều cài đặt thành các nhóm.
- Tùy chỉnh cài đặt và giá trị của chúng.
- Hiển thị màn hình Settings chính, trong đó mỗi nhóm cài đặt sẽ có một tiêu đề liên kết, chẳng hạn như General cho cài đặt chung, như trong hình minh họa dưới đây.

ảnh

- Hiển thị bố cục màn hình **master/detail**, trong đó: **Phía bên trái (master)**: Chứa các liên kết tiêu đề cho từng nhóm cài đặt. **Phía bên phải (detail)**: Hiển thị nhóm cài đặt tương ứng khi được chọn, như trong hình minh họa dưới đây.

ảnh

- Trong bài thực hành trước, bạn đã tạo một ứng dụng có tên **DroidCafeOptionsUp** bằng cách sử dụng mẫu **Basic Activity**, mẫu này cung cấp một menu tùy chọn trên thanh ứng dụng, như minh họa bên dưới.

Chú thích cho hình minh họa ở trên:

ảnh

1. Thanh ứng dụng (App bar)
2. Biểu tượng hành động trong menu tùy chọn (Options menu action icons)
3. Nút tràn (Overflow button)
4. Menu tràn tùy chọn (Options overflow menu)

2.1 Khám phá mẫu Settings Activity

Để thêm mẫu **Settings Activity** vào một dự án ứng dụng trong Android Studio, hãy làm theo các bước sau:

1. Sao chép thư mục dự án **DroidCafeOptionsUp** và đổi tên thành **DroidCafeWithSettings**. Chạy ứng dụng để đảm bảo nó hoạt động bình thường.
2. Trong **Project > Android**, chọn **app** ở đầu danh sách, sau đó nhấp chuột phải và chọn **New > Activity > Settings Activity**.
3. Trong hộp thoại xuất hiện, chấp nhận tên mặc định của **Activity** (**SettingsActivity** là tên được đề xuất) và tiêu đề (**Settings**).
4. Nhấp vào biểu tượng ba chấm ở cuối menu **Hierarchical Parent**, sau đó chọn tab **Project** trong hộp thoại **Select Activity**.
5. Mở rộng thư mục **DroidCafeWithSettings > app > src > main > java > com.example.android.droidcafeinput** và chọn **MainActivity** làm **Parent Activity**, như minh họa trong hình bên dưới. Nhấn **OK**.

ảnh

Bạn chọn **MainActivity** làm **parent** để nút **Up** trên thanh ứng dụng trong **Settings Activity** đưa người dùng quay lại **MainActivity**. Việc chọn **parent Activity** sẽ tự động cập nhật tệp **AndroidManifest.xml** để hỗ trợ điều hướng bằng nút **Up**

6. Nhấn **Finish**.
7. Trong **Project > Android**, mở rộng thư mục **app > res > xml** để xem các tệp XML được tạo bởi mẫu **Settings Activity**.

ảnh

Bạn có thể mở và tùy chỉnh các tệp XML để thêm hoặc chỉnh sửa cài đặt theo ý muốn:

- **pref_data_sync.xml**: Bố cục **PreferenceScreen** cho cài đặt "Data & Sync".
- **pref_general.xml**: Bố cục **PreferenceScreen** cho cài đặt "General".
- **pref_headers.xml**: Bố cục tiêu đề cho màn hình chính của **Settings**.
- **pref_notification.xml**: Bố cục **PreferenceScreen** cho cài đặt "Notifications".

Các bố cục XML trên sử dụng nhiều lớp con khác nhau của lớp **Preference** thay vì **View**. Một số lớp con trực tiếp của **Preference** đóng vai trò làm container cho bố cục chứa nhiều cài đặt khác nhau. Ví dụ: **PreferenceScreen** đại diện cho một **Preference** cấp cao nhất, là gốc của một hệ thống **Preference**. Các tệp XML trên sử dụng **PreferenceScreen** ở đầu mỗi màn hình cài đặt. Các lớp con khác của **Preference** cung cấp giao diện phù hợp để người dùng thay đổi cài đặt. Ví dụ:

- **CheckBoxPreference**: Một hộp kiểm cho cài đặt có thể được bật hoặc tắt.
- **ListPreference**: Một hộp thoại chứa danh sách các nút radio.

- **SwitchPreference**: Một tùy chọn hai trạng thái có thể bật/tắt (ví dụ: On/Off hoặc True/False).
- **EditTextPreference**: Một hộp thoại chứa **EditText** để nhập văn bản.
- **RingtonePreference**: Một hộp thoại hiển thị danh sách nhạc chuông có sẵn trên thiết bị.

Mẹo: Bạn có thể chỉnh sửa các tệp XML để thay đổi cài đặt mặc định và tùy chỉnh văn bản hiển thị trong phần cài đặt. Tất cả chuỗi văn bản được sử dụng trong **Settings Activity**, bao gồm tiêu đề cài đặt, mảng chuỗi cho danh sách và mô tả cài đặt, đều được định nghĩa trong tệp **strings.xml**. Các chuỗi này được đánh dấu bằng các nhận xét sau:

```
<!-- Strings liên quan đến Settings -->
<!-- Ví dụ: Cài đặt chung -->
```

Mẫu **Settings Activity** cũng tạo ra các tệp sau:

- **SettingsActivity** trong thư mục **java/com.example.android.projectname**, có thể sử dụng ngay mà không cần chỉnh sửa. Đây là **Activity** hiển thị phần cài đặt. **SettingsActivity** kế thừa từ **AppCompatActivity** để duy trì khả năng tương thích với các phiên bản Android cũ hơn.
- **AppCompatPreferenceActivity** trong thư mục **java/com.example.android.projectname**, có thể sử dụng ngay mà không cần chỉnh sửa. Đây là một **Activity** trợ giúp mà **SettingsActivity** sử dụng để duy trì khả năng tương thích ngược với các phiên bản Android trước đó.

2.2 Thêm mục menu "Settings" và kết nối nó với Activity

Như bạn đã học trong một bài thực hành khác, bạn có thể chỉnh sửa tệp **menu_main.xml** để thêm hoặc xóa các mục trong menu tùy chọn.

1. Mở rộng thư mục **res** trong **Project > Android**, sau đó mở tệp **menu_main.xml**. Nhập vào tab **Text** để hiển thị mã XML.
2. Thêm một mục menu mới có tên **Settings** với **resource id** là **action_settings**:

ảnh

Bạn đặt giá trị "**never**" cho thuộc tính **app:showAsAction** để mục **Settings** chỉ xuất hiện trong menu tùy chọn tràn (overflow menu) và không hiển thị trực tiếp trên thanh ứng dụng, vì mục này không được sử dụng thường xuyên.

Bạn đặt giá trị "**50**" cho thuộc tính **android:orderInCategory** để **Settings** xuất hiện **bên dưới Favorites** (được đặt là "**30**") nhưng **bên trên Contact** (được đặt là "**100**").

3. Trích xuất chuỗi tài nguyên cho "**Settings**" trong thuộc tính **android:title** và đặt tên tài nguyên là **settings**.

4. Mở **MainActivity**, tìm khôi **switch-case** trong phương thức **onOptionsItemSelected()**, nơi xử lý sự kiện khi người dùng nhấn vào các mục trong menu tùy chọn. Dưới đây là một đoạn mã minh họa cách xử lý **case** đầu tiên (cho **action_order**):
ảnh
5. Lưu ý rằng trong đoạn mã trên, **case** đầu tiên sử dụng một **Intent** để khởi chạy **OrderActivity**. Hãy thêm một **case** mới cho **action_settings** vào khôi **switch-case**, sử dụng mã **Intent** tương tự để khởi chạy **SettingsActivity** (nhưng không cần sử dụng **intent.putExtra**):
ảnh
5. Chạy ứng dụng trên điện thoại hoặc trình giả lập để xem cách **Settings Activity template** xử lý kích thước màn hình điện thoại.
6. Nhấn vào biểu tượng **overflow** trong menu tùy chọn, sau đó chọn **Settings** để mở **Settings Activity**, như hiển thị ở phía bên trái của hình minh họa bên dưới.
7. Nhấn vào từng tiêu đề cài đặt (**General**, **Notifications**, và **Data & Sync**), như hiển thị ở giữa hình minh họa bên dưới, để xem nhóm cài đặt tương ứng trên từng màn hình con của **Settings screen**, được hiển thị ở phía bên phải của hình minh họa.
8. Nhấn nút **Up** trong **Settings Activity** để quay lại **MainActivity**.
ảnh

Bạn sử dụng mã của **Settings Activity template** mà không cần chỉnh sửa. Mẫu này không chỉ cung cấp bộ cục cho màn hình điện thoại và máy tính bảng, mà còn có chức năng **lắng nghe thay đổi cài đặt và cập nhật phần tóm tắt** để phản ánh thay đổi đó.

Ví dụ: Nếu bạn thay đổi cài đặt "**Add friends to messages**" (với các lựa chọn **Always**, **When possible**, hoặc **Never**), thì lựa chọn bạn đã chọn sẽ hiển thị trong phần tóm tắt bên dưới cài đặt.

ảnh

Nhìn chung, bạn không cần thay đổi mã của **Settings Activity template** để tùy chỉnh **Activity** theo các cài đặt mong muốn trong ứng dụng của mình.

Bạn có thể **tùy chỉnh** tiêu đề cài đặt, phần tóm tắt, các giá trị có thể chọn, giá trị mặc định mà không cần sửa đổi mã của mẫu, và thậm chí có thể **thêm nhiều cài đặt hơn** vào các nhóm có sẵn.

2.3 Tùy chỉnh cài đặt do mẫu cung cấp

Để tùy chỉnh các cài đặt được cung cấp bởi **Settings Activity template**, hãy chỉnh sửa **chuỗi văn bản** và **mảng chuỗi** trong tệp **strings.xml**, cũng như các thuộc tính bộ cục cho từng cài đặt trong các tệp trong thư mục **xml**.

Trong bước này, bạn sẽ thay đổi cài đặt "**Data & Sync**".

1. Mở rộng thư mục **res > values** và mở tệp **strings.xml**. Cuộn xuống phần nội dung có nhận xét `<!-- Example settings for Data & Sync -->`.
ảnh
2. Chính sửa **pref_header_data_sync**, chuỗi tài nguyên hiện đang được đặt là **Data & sync** (& là mã HTML cho dấu "&"). Thay đổi giá trị thành **Account** (không có dấu ngoặc kép).
3. Bạn nên **đổi tên tài nguyên** để giúp mã dễ hiểu hơn (ứng dụng vẫn hoạt động nếu không đổi tên, nhưng việc này giúp mã rõ ràng hơn). Nhấp chuột phải (**hoặc Control + click**) vào tài nguyên **pref_header_data_sync**, chọn **Refactor > Rename**. Đổi tên thành **pref_header_account**, chọn tùy chọn **search in comments and strings**, sau đó nhấn **Refactor**.
4. Bạn cũng nên **đổi tên tệp XML** để giúp mã dễ hiểu hơn. Nhấp chuột phải (**hoặc Control + click**) vào tài nguyên **pref_data_sync** trong **Project > Android**, chọn **Refactor > Rename**. Đổi tên thành **pref_account**, chọn tùy chọn **search in comments and strings**, sau đó nhấn **Refactor**.
5. Chính sửa chuỗi tài nguyên **pref_title_sync_frequency** (hiện đang đặt là **Sync frequency**) và thay đổi thành **Market**.
6. Thực hiện **Refactor > Rename** cho tài nguyên **pref_title_sync_frequency**, đổi tên thành **pref_title_account**, giống như các bước trước.
7. Thực hiện **Refactor > Rename** cho mảng chuỗi **pref_sync_frequency_titles**, đổi tên thành **pref_market_titles**.
8. Thay đổi từng giá trị trong mảng **pref_market_titles** (ví dụ: **15 minutes, 30 minutes, 1 hour**, v.v.) thành tên của các thị trường, chẳng hạn như **United States, Canada**, v.v., thay vì các khoảng thời gian.
ảnh
9. Thực hiện **Refactor > Rename** cho tên tài nguyên mảng chuỗi **pref_sync_frequency_values**, đổi tên thành **pref_market_values**.
10. Thay đổi từng giá trị trong mảng **pref_market_values** (ví dụ: **15, 30, 60**, v.v.) thành các giá trị tương ứng với thị trường—sử dụng các chữ viết tắt của quốc gia tương ứng, chẳng hạn như **US, CA**, v.v.
ảnh
11. Cuộn xuống tài nguyên chuỗi **pref_title_system_sync_settings** và chỉnh sửa giá trị (hiện đang đặt là **System sync settings**) thành **Account settings**.
12. Thực hiện **Refactor > Rename** cho tên tài nguyên chuỗi **pref_title_system_sync_settings**, đổi tên thành **pref_title_account_settings**.
13. Mở tệp **pref_account.xml**. Thành phần **ListPreference** trong bố cục này xác định cài đặt mà bạn vừa thay đổi. Lưu ý rằng các tài nguyên chuỗi cho thuộc tính **android:entries**, **android:entryValues**, và **android:title** hiện đã được thay đổi thành các giá trị bạn đã cập nhật trong các bước trước.
ảnh

14. Thay đổi thuộc tính **android:defaultValue** thành "US".

ảnh

Vì khóa của tùy chọn cài đặt này ("sync_frequency") đã được mã hóa cứng ở những nơi khác trong mã Java, **đừng thay đổi** thuộc tính **android:key**. Thay vào đó, hãy tiếp tục sử dụng "sync_frequency" làm khóa cho cài đặt này trong ví dụ này.

Nếu bạn đang **tùy chỉnh toàn diện** cài đặt cho một ứng dụng thực tế, bạn nên dành thời gian để thay đổi tất cả các khóa đã mã hóa cứng trong toàn bộ mã.

Lưu ý: Tại sao không sử dụng một tài nguyên chuỗi cho khóa?

Bởi vì tài nguyên chuỗi có thể được **bản địa hóa** (localized) cho các ngôn ngữ khác nhau bằng cách sử dụng các tệp XML đa ngôn ngữ. Nếu chuỗi khóa vô tình bị dịch cùng với các chuỗi khác, điều đó có thể khiến ứng dụng **bị lỗi**.

2.4 Thêm mã để đặt giá trị mặc định cho cài đặt

Để thêm mã để đặt giá trị mặc định cho cài đặt, hãy làm theo các bước sau:

1. Mở **MainActivity** và tìm phương thức **onCreate()**.
2. Thêm các câu lệnh **PreferenceManager.setDefaultValues** sau vào cuối phương thức **onCreate()**.

ảnh

Các giá trị mặc định đã được chỉ định trong tệp XML bằng thuộc tính **android:defaultValue**, nhưng các câu lệnh trên đảm bảo rằng tệp **SharedPreferences** được khởi tạo đúng cách với các giá trị mặc định.

Phương thức **setDefaultValues()** nhận ba đối số:

- **Ngữ cảnh của ứng dụng** (*context*), chẳng hạn như **this**.
- **ID tài nguyên** của tệp bố cục XML cài đặt, trong đó bao gồm các giá trị mặc định được đặt bằng thuộc tính **android:defaultValue**.
- **Một giá trị boolean** chỉ định liệu có nên đặt các giá trị mặc định nhiều lần hay không. Khi đặt là **false**, hệ thống sẽ chỉ đặt các giá trị mặc định **một lần duy nhất**, khi phương thức này được gọi lần đầu tiên. Miễn là bạn đặt đối số thứ ba này thành **false**, bạn có thể gọi phương thức này an toàn mỗi khi ứng dụng khởi động.

Hoạt động sẽ bắt đầu mà không ghi đè các giá trị cài đặt đã lưu của người dùng bằng cách đặt lại chúng về giá trị mặc định. Tuy nhiên, nếu bạn đặt giá trị này thành **true**, phương thức sẽ **ghi đè** mọi giá trị trước đó bằng các giá trị mặc định.

2.5 Thêm mã để đọc giá trị của cài đặt

1. Thêm đoạn mã sau vào cuối phương thức **onCreate()** trong **MainActivity**. Bạn có thể thêm ngay sau đoạn mã đã thêm ở bước trước để thiết lập các giá trị mặc định cho cài đặt.

ảnh

Như bạn đã học ở nhiệm vụ trước, bạn sử dụng **PreferenceManager.getDefaultSharedPreferences(this)** để lấy cài đặt dưới dạng đối tượng **SharedPreferences** (*marketPref*). Sau đó, bạn sử dụng **getString()** để lấy giá trị chuỗi của cài đặt sử dụng khóa (**sync_frequency**) và gán nó cho *marketPref*.

Nếu không có giá trị nào cho khóa này, phương thức **getString()** sẽ gán giá trị -1 cho *marketPref*, đây là giá trị của **Other** trong mảng **pref_market_values**.

2. Chạy ứng dụng. Khi màn hình chính của ứng dụng xuất hiện lần đầu tiên, bạn sẽ thấy một thông báo **Toast** hiển thị ở cuối màn hình.
 - o Lần đầu chạy ứng dụng, bạn sẽ thấy "-1" xuất hiện trong thông báo **Toast**, vì bạn chưa thay đổi cài đặt nào.
3. Nhấn vào **Settings** trong menu tùy chọn, sau đó nhấn **Account** trong màn hình **Settings**.
 - o Nhấn vào **Market** và chọn **Canada**, như minh họa bên dưới.
4. Nhấn vào nút **Up** trên thanh ứng dụng để quay lại màn hình **Settings**, sau đó nhấn lần nữa để quay lại màn hình chính.
5. Chạy lại ứng dụng từ **Android Studio**. Bạn sẽ thấy một thông báo **Toast** hiển thị "**CA**" (đại diện cho Canada), và cài đặt **Market** bây giờ đã được đặt thành **Canada**.

ảnh

6. Bây giờ hãy chạy ứng dụng trên máy tính bảng hoặc trình giả lập máy tính bảng. Vì màn hình của máy tính bảng có kích thước lớn hơn, Android runtime sẽ tận dụng không gian bổ sung này. Trên máy tính bảng, các cài đặt và chi tiết sẽ được hiển thị trên cùng một màn hình, giúp người dùng quản lý cài đặt dễ dàng hơn.

ảnh

Mã giải pháp cho **Nhiệm vụ 2**
Dự án **Android Studio: DroidCafeWithSettings**

Thử thách lập trình

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

Thử thách: Ứng dụng **DroidCafeWithSettings** hiển thị phần cài đặt đúng cách trên màn hình có kích thước máy tính bảng, nhưng nút **Up** trên thanh ứng dụng không đưa người dùng quay lại **MainActivity** như trên màn hình có kích thước điện thoại. Nguyên nhân là do có ba phương thức **onOptionsItemSelected()**—mỗi phương thức dành cho một **Fragment**—trong **SettingsActivity**. Nó sử dụng đoạn mã sau để khởi động lại **SettingsActivity** khi người dùng nhấn vào nút **Up**:

Hành động trên phù hợp với màn hình điện thoại, nơi các tiêu đề cài đặt (**General**, **Notifications**, và **Account**) xuất hiện trên một màn hình riêng (**SettingsActivity**). Sau khi thay đổi cài đặt, khi người dùng nhấn vào nút **Up**, họ sẽ quay lại màn hình tiêu đề cài đặt trong **SettingsActivity**. Nếu nhấn **Up** lần nữa, họ sẽ trở về **MainActivity**.

Tuy nhiên, trên máy tính bảng, các tiêu đề luôn hiển thị ở ngăn bên trái (trong khi phần cài đặt hiển thị ở ngăn bên phải). Do đó, nhấn vào nút **Up** không đưa người dùng trở lại **MainActivity**.

Hãy tìm cách làm cho nút **Up** hoạt động đúng trong **SettingsActivity** trên màn hình có kích thước máy tính bảng.

Gợi ý: Mặc dù có nhiều cách để khắc phục vấn đề này, bạn có thể tham khảo các bước sau:

1. **Thêm một tệp dimens.xml mới** để hỗ trợ các màn hình có kích thước lớn hơn 600dp.

Khi ứng dụng chạy trên một thiết bị cụ thể, hệ thống sẽ tự động chọn tệp dimens.xml phù hợp dựa trên các bộ định tính (**qualifiers**). Bạn có thể thêm một tệp dimens.xml mới với bộ định tính **Smallest Screen Width** được đặt thành **600dp** (**sw600dp**) như đã học trong bài thực hành về hỗ trợ chế độ xoay màn hình và kích thước màn hình. Điều này giúp xác định bất kỳ thiết bị nào có màn hình lớn, chẳng hạn như máy tính bảng.

2. **Thêm một tài nguyên bool** vào giữa các thẻ `<resources>` và `</resources>` trong tệp dimens.xml (**sw600dp**). Tệp này sẽ tự động được chọn cho các thiết bị có màn hình lớn (máy tính bảng).

anh

3. Thêm tài nguyên bool sau vào tệp dimens.xml tiêu chuẩn, tệp này sẽ được chọn khi ứng dụng chạy trên bất kỳ thiết bị nào **không** phải màn hình lớn.

anh

4. Trong **SettingsActivity**, bạn có thể thêm vào cả ba phương thức **onOptionsItemSelected()** (mỗi phương thức cho một Fragment) một khối **if else** để kiểm tra xem **isTablet** có bằng **true** hay không. Nếu đúng, mã của bạn có thể chuyển hướng hành động của nút **Up** về **MainActivity**.

Mã giải pháp cho thử thách

Dự án Android Studio: [DroidCafeWithSettingsChallenge](#)

Tóm tắt

Người dùng mong muốn có thể điều hướng đến cài đặt ứng dụng bằng cách nhấn vào **Settings** trong điều hướng bên (chẳng hạn như **navigation drawer**) hoặc trong **options menu** trên thanh ứng dụng (**app bar**).

Để cung cấp cài đặt cho ứng dụng của bạn, hãy tạo một **Activity** dành cho phần cài đặt:

- Sử dụng một **Activity** chứa **PreferenceFragment** để hiển thị cài đặt của ứng dụng.
- Để duy trì khả năng tương thích với **AppCompatActivity** và thư viện **android.support.v7.preference**, hãy sử dụng **PreferenceFragmentCompat** thay vì **PreferenceFragment**.

Hiển thị từng Fragment trong **SettingsActivity**:

- Nếu lớp **Activity** kế thừa từ **Activity** và lớp **Fragment** kế thừa từ **PreferenceFragment**, sử dụng **getFragmentManager()**.
- Nếu lớp **Activity** kế thừa từ **AppCompatActivity** và lớp **Fragment** kế thừa từ **PreferenceFragmentCompat**, sử dụng **getSupportFragmentManager()**.
- Để liên kết tài nguyên cài đặt **preferences.xml** với **Fragment**, sử dụng **setPreferencesFromResource()**.
- Để đặt giá trị mặc định cho cài đặt, sử dụng **PreferenceManager.setDefaultValues()**.
- Để kết nối mục **Settings** trong menu với **SettingsActivity**, sử dụng **Intent**.

Thêm tệp tài nguyên XML cho cài đặt:

1. Tạo một thư mục tài nguyên mới (**File > New > Android Resource Directory**).
2. Trong menu thả xuống **Resource type**, chọn **xml**. Thư mục **xml** sẽ xuất hiện bên trong thư mục **res**.
3. Nhập vào **xml**, sau đó chọn **File > New > XML resource file**.
4. Nhập **preferences** làm tên của tệp XML. Tệp **preferences.xml** sẽ xuất hiện bên trong thư mục **xml**.

Thêm các điều khiển giao diện người dùng như công tắc bật/tắt với các thuộc tính trong tệp XML **preferences**:

Để duy trì khả năng tương thích với **AppCompatActivity**, hãy sử dụng phiên bản thư viện **android.support.v7.preference**. Ví dụ, sử dụng **SwitchPreferenceCompat** cho các công tắc bật/tắt.

Sử dụng các thuộc tính với từng phần tử giao diện người dùng cho cài đặt:

- **android:defaultValue** là giá trị của cài đặt khi ứng dụng khởi động lần đầu tiên.
- **android:title** là tiêu đề cài đặt hiển thị cho người dùng.
- **android:key** là khóa được sử dụng để lưu trữ giá trị cài đặt.
- **android:summary** là văn bản hiển thị bên dưới cài đặt cho người dùng.

Lưu và đọc giá trị cài đặt:

- Khi ứng dụng khởi động, phương thức **onCreate()** của **MainActivity** có thể đọc các giá trị cài đặt đã thay đổi và sử dụng các giá trị đó thay vì giá trị mặc định.
- Mỗi cài đặt được xác định bằng một cặp khóa-giá trị (*key-value pair*). Hệ thống Android sử dụng cặp khóa-giá trị này khi lưu hoặc truy xuất cài đặt từ tệp **SharedPreferences** của ứng dụng. Khi người dùng thay đổi một cài đặt, hệ thống sẽ cập nhật giá trị tương ứng trong tệp **SharedPreferences**.
- Để sử dụng giá trị của cài đặt, ứng dụng có thể sử dụng khóa (*key*) để lấy giá trị cài đặt từ tệp **SharedPreferences**.
- Ứng dụng đọc giá trị cài đặt từ **SharedPreferences** bằng cách sử dụng **PreferenceManager.getDefaultSharedPreferences()**, và lấy từng giá trị cài đặt bằng các phương thức **.getString**, **.getBoolean**, v.v.

Các khái niệm liên quan

Tài liệu về các khái niệm liên quan có trong **9.2: Cài đặt ứng dụng**.

Tìm hiểu thêm

Tài liệu hướng dẫn sử dụng Android Studio: [Android Studio User Guide]

Tài liệu dành cho nhà phát triển Android: [Android Developer Documentation]

- **Cài đặt (Tổng quan)**
- **Preference (Tùy chọn)**
- **PreferenceFragment (Phân mảnh tùy chọn)**
- **PreferenceFragmentCompat (Phân mảnh tùy chọn tương thích)**
- **Fragment (Phân mảnh)**
- **SharedPreferences (Bộ lưu trữ dữ liệu chia sẻ)**
- **Lưu dữ liệu dưới dạng cặp key-value**
- **Hỗ trợ các kích thước màn hình khác nhau**

Tài liệu về Material Design:

- Thiết kế giao diện Android settings theo chuẩn Material Design

Thảo luận trên Stack Overflow:

- Làm thế nào để thêm tệp dimens.xml vào Android Studio?
- Cách xác định thiết bị là điện thoại hay máy tính bảng?

Bài tập về nhà

Xây dựng và chạy ứng dụng

Mở dự án **DroidCafeWithSettings** và thực hiện các yêu cầu sau:

1. Thêm một **ListPreference** (hộp thoại với các nút radio) vào phần cài đặt chung. Đặt hộp thoại này trong màn hình **General settings**, bên dưới **ListPreference** có tên "**Add friends to order messages**".
2. Chính sửa mảng chuỗi (string-array) sử dụng cho **ListPreference**. Bao gồm tiêu đề "**Choose a delivery method.**" Sử dụng cùng các phương thức giao hàng đã có trong các nút radio trong **OrderActivity**.
3. Hiển thị phương thức giao hàng được chọn trong thông báo **Toast**. Kết hợp nó với giá trị của **Market setting** đã chọn.
4. **Bài tập nâng cao (Extra Credit)**: Hiển thị phương thức giao hàng đã chọn như một phần văn bản tóm tắt (summary) bên dưới tiêu đề **ListPreference**. Cho phép văn bản này thay đổi khi người dùng cập nhật cài đặt.

anh

Trả lời các câu hỏi

Câu hỏi 1

Trong dự án **DroidCafeWithSettings**, bạn định nghĩa mảng các mục nhập (entries) và mảng giá trị (values) cho **ListPreference** trong tệp nào? Chọn một:

- **pref_general.xml**
- **strings.xml** ✓
- **menu_main.xml**
- **activity_main.xml**
- **content_main.xml**

Câu hỏi 2

Trong dự án **DroidCafeWithSettings**, bạn sử dụng mảng các mục nhập

(**entries**) và mảng giá trị (**values**) để thiết lập **ListPreference**, đồng thời đặt khóa (**key**) và giá trị mặc định (**default value**) trong tệp nào?

Chọn một:

- pref_general.xml
- strings.xml
- menu_main.xml
- content_main.xml
- SettingsActivity.java

Câu hỏi 3

Làm thế nào để thiết lập một **Settings Activity** và một **Fragment** với **SwitchPreference** cho giao diện người dùng, đồng thời vẫn tương thích với **thư viện v7 appcompat** để hỗ trợ các phiên bản Android cũ hơn?

Chọn một:

- Sử dụng một **Settings Activity** mở rộng từ Activity, một **Fragment** mở rộng từ PreferenceFragment, và một SwitchPreference cho giao diện người dùng.
- Thay đổi MainActivity để mở rộng từ Activity.
- **Sử dụng một Settings Activity mở rộng từ AppCompatActivity, một Fragment mở rộng từ PreferenceFragmentCompat, và một SwitchPreferenceCompat cho UI.**
- Không thể sử dụng một Fragment với SwitchPreference và vẫn giữ được tính tương thích với **v7 appcompat library**.

Nộp ứng dụng của bạn để chấm điểm

Hướng dẫn cho người chấm điểm

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

- Phương thức onCreate() đọc cài đặt deliveryPref bằng sharedPref.getString().
- Tệp pref_general.xml chứa một ListPreference sử dụng một mảng các lựa chọn phương thức giao hàng làm danh sách mục nhập.

- **Điểm cộng thêm:** Câu lệnh bindPreferenceSummaryToValue(findPreference("delivery")) đã được thêm vào phương thức onCreate() của lớp GeneralPreferenceFragment trong SettingsActivity để hiển thị lựa chọn phương thức giao hàng trong phần tóm tắt của tùy chọn.

Bài 10.1 Phần A: Room, LiveData, và ViewModel

Giới thiệu



Hệ điều hành Android cung cấp một nền tảng vững chắc để xây dựng các ứng dụng hoạt động tốt trên nhiều thiết bị và hình thức khác nhau. Tuy nhiên, các vấn đề như vòng đời phức tạp và việc thiếu một kiến trúc ứng dụng được khuyến nghị khiến việc viết ứng dụng mạnh mẽ trở nên khó khăn. **Android Architecture Components** cung cấp các thư viện cho các tác vụ phổ biến như quản lý vòng đời và lưu trữ dữ liệu, giúp dễ dàng triển khai **kiến trúc ứng dụng được khuyến nghị**.

Các Architecture Components giúp bạn cấu trúc ứng dụng theo cách mạnh mẽ, dễ kiểm thử và dễ bảo trì hơn, đồng thời giảm bớt mã dư thừa (boilerplate code).

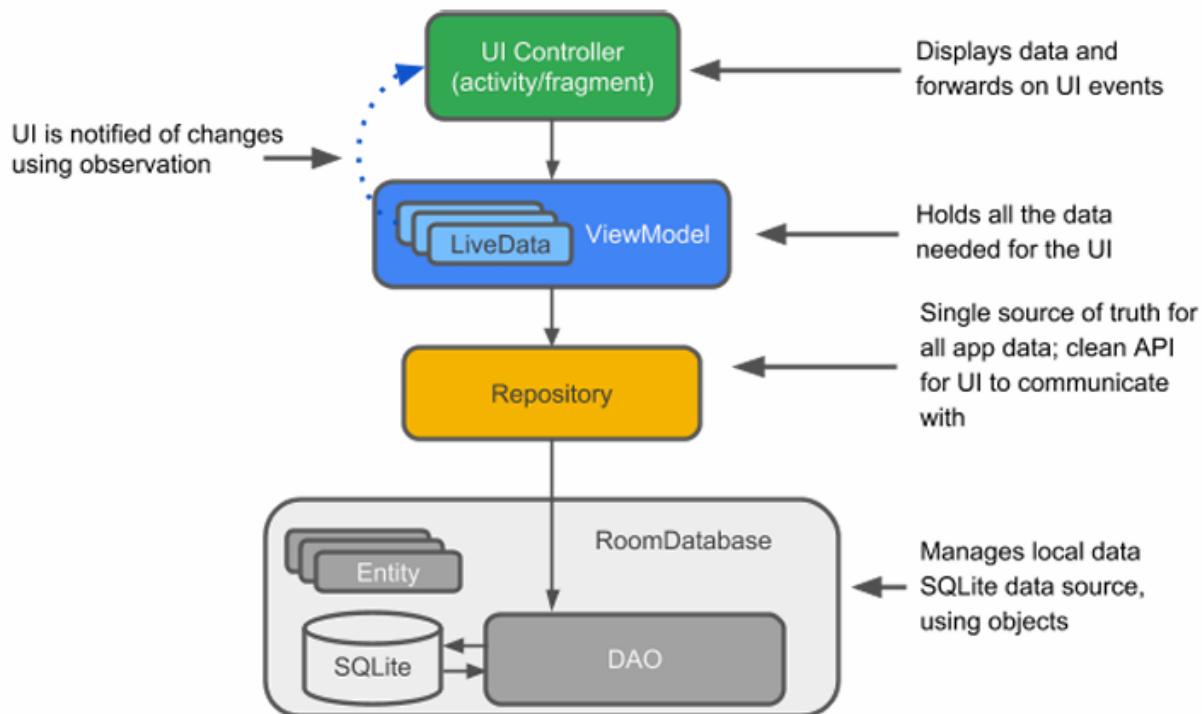
Các Thành Phần Kiến Trúc Được Khuyến Nghị?

Khi nói đến kiến trúc, việc nhìn thấy bức tranh tổng thể trước tiên sẽ rất hữu ích. Để giới thiệu các thuật ngữ, dưới đây là tổng quan ngắn gọn về các Thành Phần Kiến Trúc và cách chúng hoạt động cùng nhau. Mỗi thành phần sẽ được giải thích chi tiết hơn khi bạn sử dụng chúng trong thực hành này.

Sơ đồ dưới đây hiển thị một dạng cơ bản của kiến trúc được khuyến nghị cho các ứng dụng sử dụng các Thành Phần Kiến Trúc.

Kiến trúc bao gồm một bộ điều khiển giao diện người dùng (UI Controller), một **ViewModel**

phục vụ **LiveData**, một **Repository**, và một **cơ sở dữ liệu Room**. Cơ sở dữ liệu Room được hỗ trợ bởi một cơ sở dữ liệu **SQLite** và có thể truy cập thông qua **đối tượng truy cập dữ liệu (DAO - Data Access Object)**. Mỗi thành phần được mô tả ngắn gọn bên dưới và chi tiết hơn trong chương "Các khái niệm về Thành Phần Kiến Trúc", mục **10.1: Lưu trữ dữ liệu với Room**. Bạn sẽ triển khai các thành phần này trong phần thực hành này.



Vì tất cả các thành phần đều tương tác với nhau, bạn sẽ gặp các tham chiếu đến chúng trong suốt quá trình thực hành này. Dưới đây là phần giải thích ngắn gọn về từng thành phần:

Entity: Trong bối cảnh của các Thành Phần Kiến Trúc, **entity** là một lớp có chú thích (annotated class) mô tả một bảng trong cơ sở dữ liệu.

Cơ sở dữ liệu SQLite: Dữ liệu trên thiết bị được lưu trữ trong một cơ sở dữ liệu **SQLite**. Thư viện **Room persistence** sẽ tạo và duy trì cơ sở dữ liệu này cho bạn.

DAO (Data Access Object): Viết tắt của **đối tượng truy cập dữ liệu**. Đây là một ánh xạ từ các truy vấn SQL sang các hàm. Trước đây, bạn phải định nghĩa các truy vấn này trong một lớp hỗ trợ. Khi sử dụng **DAO**, mã của bạn chỉ cần gọi các hàm, và các thành phần sẽ xử lý phần còn lại.

Room database: Một lớp cơ sở dữ liệu hoạt động trên **SQLite**, giúp xử lý các tác vụ lặp đi lặp lại mà trước đây bạn phải làm thủ công bằng một lớp hỗ trợ. Cơ sở dữ liệu Room sử dụng **DAO** để gửi truy vấn đến cơ sở dữ liệu SQLite dựa trên các hàm được gọi.

Repository: Một lớp do bạn tạo ra để quản lý nhiều nguồn dữ liệu. Ngoài cơ sở dữ liệu Room, **Repository** có thể quản lý các nguồn dữ liệu từ xa, chẳng hạn như một máy chủ web.

ViewModel: Cung cấp dữ liệu cho giao diện người dùng (UI) và đóng vai trò là trung tâm giao tiếp giữa **Repository** và UI. Nó giúp UI không phải làm việc trực tiếp với backend. Các thể hiện của **ViewModel** có thể tồn tại ngay cả khi cấu hình thiết bị thay đổi.

LiveData: Một lớp chứa dữ liệu theo mô hình **observer pattern** (mô hình quan sát), có nghĩa là nó có thể được theo dõi. **LiveData** luôn lưu trữ hoặc bộ nhớ đệm phiên bản mới nhất của dữ liệu và thông báo cho các thành phần quan sát khi dữ liệu thay đổi. Thông thường, các thành phần UI sẽ quan sát dữ liệu liên quan. **LiveData** có khả năng nhận biết vòng đời (lifecycle-aware), nên nó tự động quản lý việc dừng và tiếp tục quan sát dựa trên trạng thái của **activity** hoặc **fragment** đang quan sát nó.

Những kiến thức bạn nên biết trước

Bạn cần có khả năng tạo và chạy ứng dụng trong **Android Studio 3.0** hoặc phiên bản cao hơn. Đặc biệt, bạn nên quen thuộc với:

- **RecyclerView** và **adapter**
- **Cơ sở dữ liệu SQLite** và ngôn ngữ truy vấn **SQLite**
- **Xử lý đa luồng (Threading)** nói chung, đặc biệt là **AsyncTask**

Sẽ hữu ích nếu bạn quen thuộc với:

- Các mẫu kiến trúc phần mềm giúp tách biệt dữ liệu khỏi giao diện người dùng (UI).

Mô hình quan sát (Observer Pattern). Tóm lại, **mô hình quan sát** xác định một mối quan hệ phụ thuộc **một-nhiều** giữa các đối tượng. Khi một đối tượng thay đổi trạng thái, tất cả các đối tượng phụ thuộc vào nó sẽ được **thông báo** và **cập nhật tự động**. Đối tượng chính được gọi là "**chủ thẻ**" (**subject**). Các đối tượng phụ thuộc vào nó được gọi là "**người quan sát**" (**observers**). Thông thường, **subject** sẽ thông báo cho **observers** bằng cách gọi một trong các phương thức của họ. **Subject** biết phương thức nào cần gọi vì **observers** đã được **đăng ký** (**registered**) với **subject** và chỉ định phương thức cần gọi.

Quan trọng: Phần thực hành này triển khai kiến trúc được định nghĩa trong **Hướng dẫn Kiến trúc Ứng dụng (Guide to App Architecture)** và được giải thích trong chương

Các khái niệm về Thành Phần Kiến Trúc, mục 10.1: **Lưu trữ dữ liệu với Room**. Bạn nên đọc chương khái niệm này để hiểu rõ hơn.

Những gì bạn sẽ học

- Cách thiết kế và xây dựng một ứng dụng bằng một số **Thành Phần Kiến Trúc Android**.
Bạn sẽ sử dụng **Room**, **ViewModel**, và **LiveData**.

Những gì bạn sẽ làm

- Tạo một ứng dụng với một **Activity** hiển thị danh sách từ trong **RecyclerView**.
- Tạo một **Entity** để biểu diễn các đối tượng từ.
- Xác định ánh xạ từ các truy vấn SQL sang các phương thức Java trong **DAO (Data Access Object)**.
- Sử dụng **LiveData** để hiển thị các thay đổi của dữ liệu lên giao diện người dùng thông qua cơ chế quan sát (observers).
- Thêm **Room database** vào ứng dụng để lưu trữ dữ liệu cục bộ và khởi tạo cơ sở dữ liệu.
- Trừu tượng hóa lớp backend dữ liệu bằng **Repository**, cung cấp một API không phụ thuộc vào cách dữ liệu được lưu trữ hoặc lấy.
- Sử dụng **ViewModel** để tách biệt toàn bộ thao tác dữ liệu khỏi giao diện người dùng (UI).
- Thêm một **Activity** thứ hai để cho phép người dùng thêm từ mới

Tổng quan về ứng dụng

Trong phần thực hành này, bạn sẽ xây dựng một ứng dụng sử dụng **Android Architecture Components**. Ứng dụng có tên **RoomWordsSample**, lưu trữ một danh sách các từ trong cơ sở dữ liệu **Room** và hiển thị danh sách này trong **RecyclerView**. Ứng dụng **RoomWordsSample** khá cơ bản, nhưng đầy đủ để bạn có thể sử dụng nó như một mẫu để xây dựng và phát triển thêm.

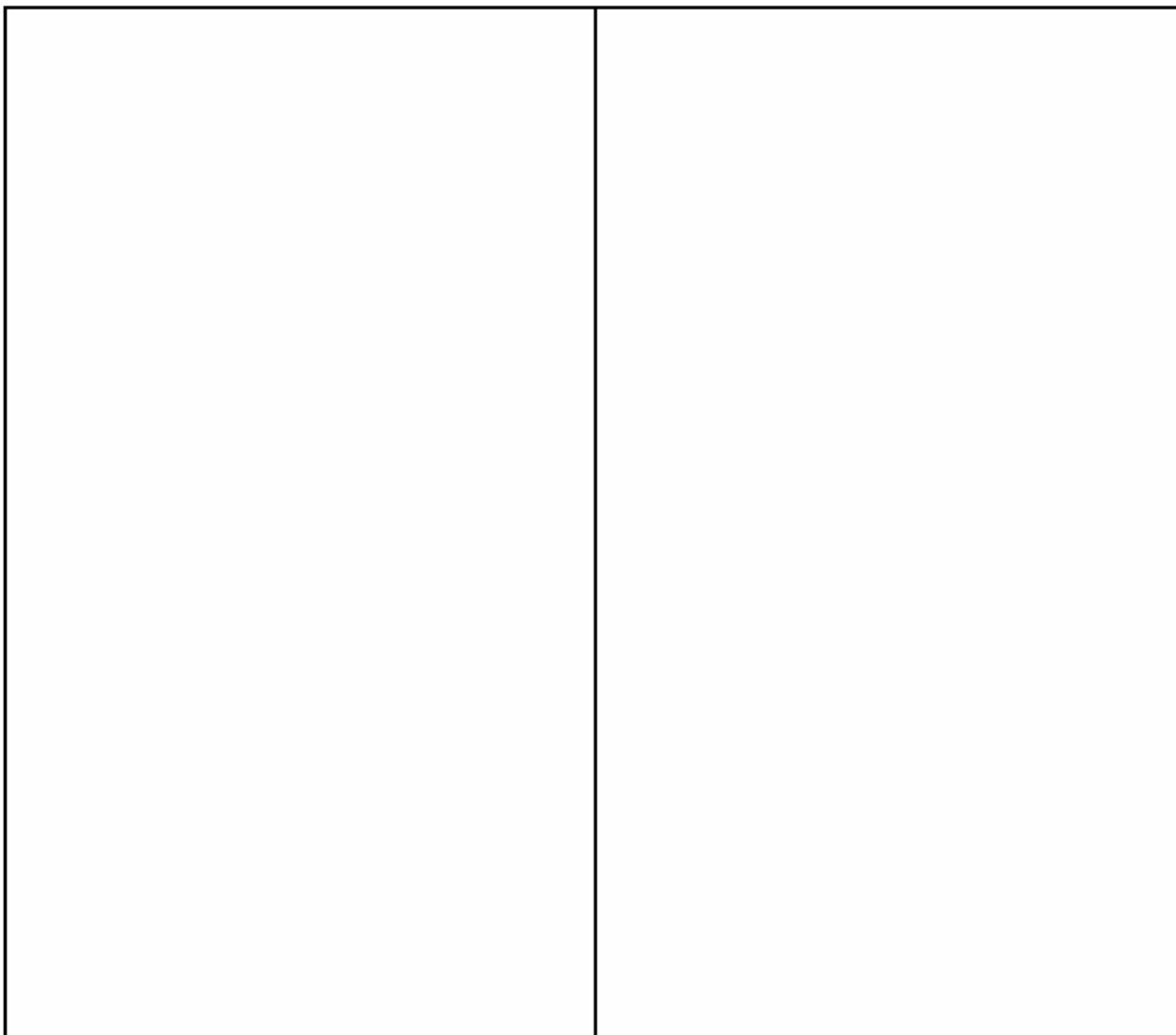
Ứng dụng **RoomWordsSample** thực hiện các chức năng sau:

- Làm việc với cơ sở dữ liệu để lấy và lưu trữ các từ, và tự động điền sẵn một số từ vào cơ sở dữ liệu.
- Hiển thị tất cả các từ trong **RecyclerView** trong **MainActivity**.

- Mở một **Activity** thứ hai khi người dùng nhấn nút + **FAB**. Khi người dùng nhập một từ, ứng dụng sẽ thêm từ đó vào cơ sở dữ liệu và danh sách sẽ được cập nhật tự động.

Các ảnh chụp màn hình dưới đây hiển thị những điều sau:

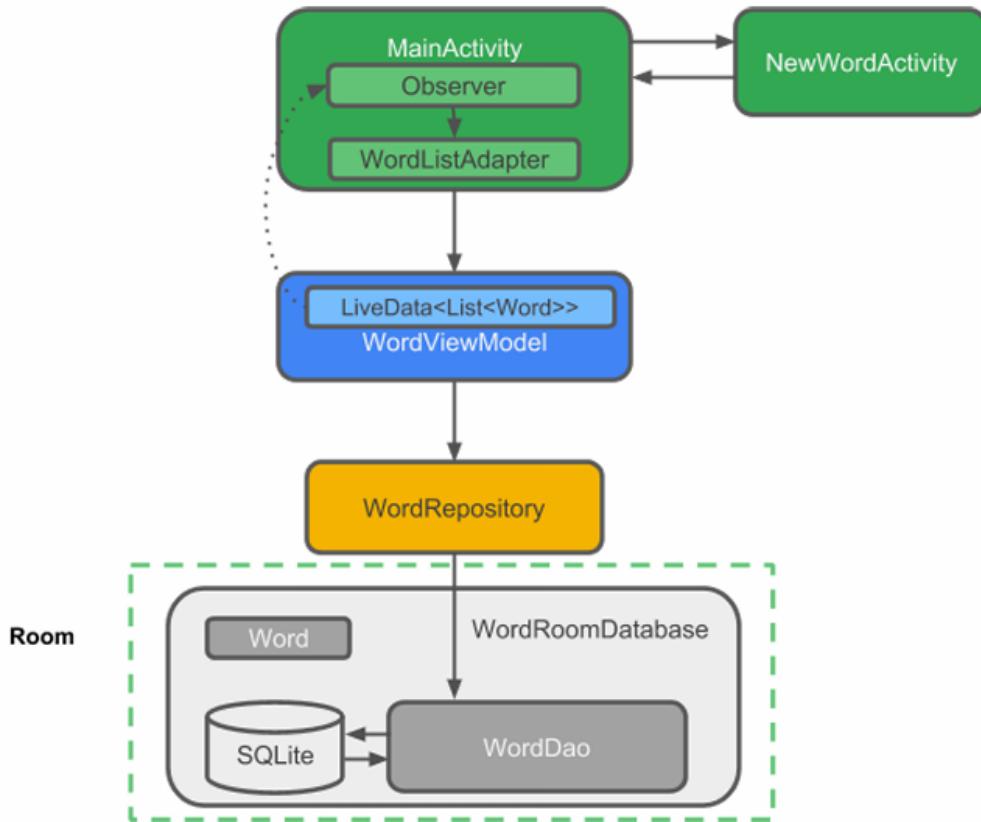
- Ứng dụng **RoomWordsSample** khi bắt đầu, với danh sách từ ban đầu.
- **Activity** để thêm một từ mới.



Tổng quan về kiến trúc RoomWordsSample

Sơ đồ dưới đây phản ánh sơ đồ tổng quan từ phần giới thiệu và hiển thị tất cả các thành phần của ứng dụng **RoomWordsSample**. Mỗi hộp bao quanh (ngoại trừ cơ sở dữ liệu SQLite) đại diện cho một lớp mà bạn sẽ tạo.

Mẹo: In hoặc mở sơ đồ này trong một tab riêng biệt để bạn có thể tham khảo khi xây dựng mã.



Nhiệm vụ 1: Tạo ứng dụng RoomWordsSample

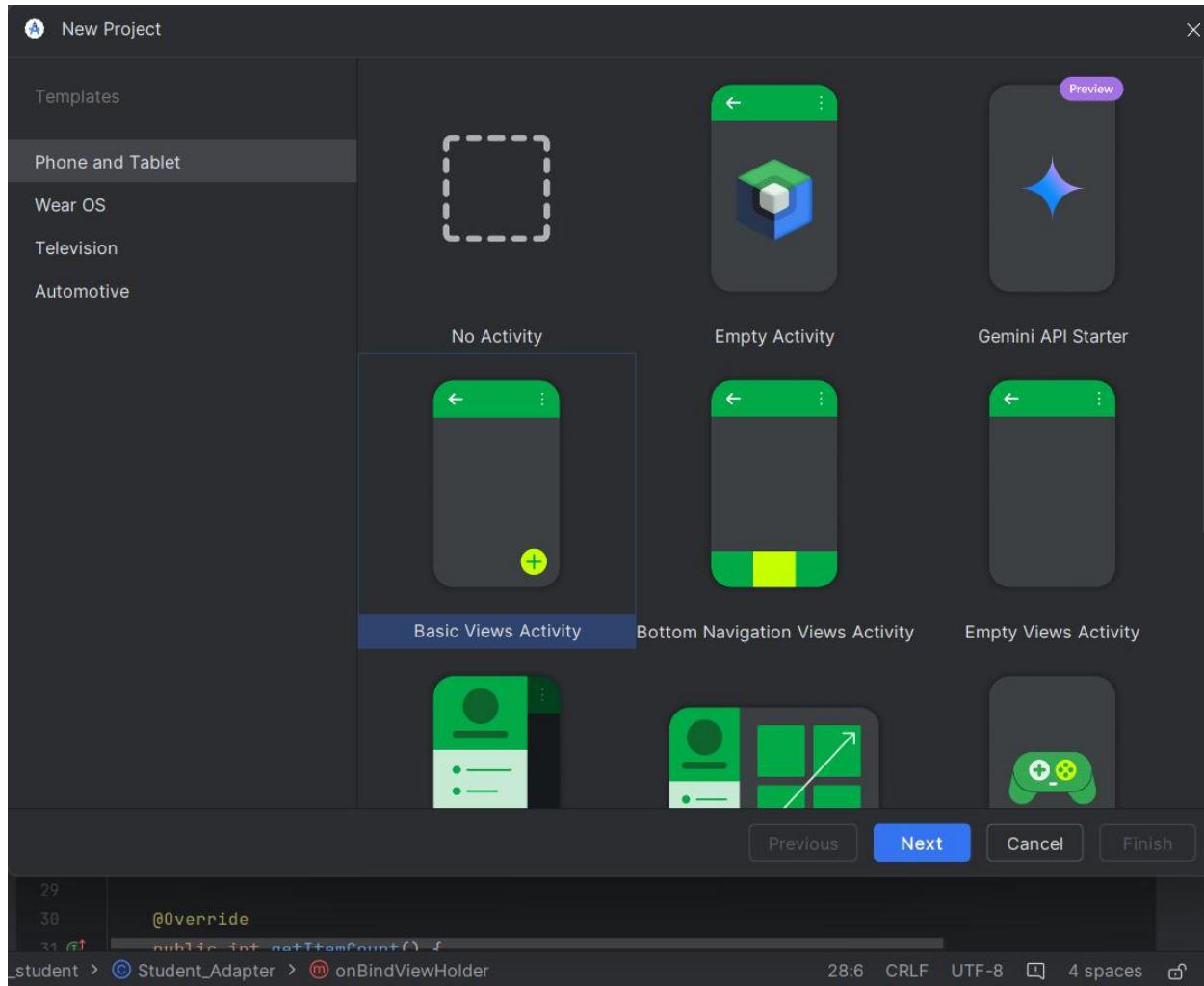
Lưu ý: Trong phần thực hành này, bạn sẽ cần tạo các biến thành viên, nhập các lớp và trích xuất các giá trị khi cần thiết. Mã mà bạn được yêu cầu làm quen sẽ được cung cấp nhưng không được giải thích chi tiết.

1.1 Tạo ứng dụng với một Activity

Mở **Android Studio** và tạo một ứng dụng mới. Trong các màn hình cài đặt, thực hiện các bước sau:

- Đặt tên ứng dụng là **RoomWordsSample**.
- Nếu bạn thấy các hộp kiểm cho **Include Kotlin support** và **Include C++ support**, hãy bỏ chọn cả hai.
- Chọn chỉ **Phone & Tablet** là kiểu thiết bị, và đặt **minimum SDK** là API 14 hoặc cao hơn.

- Chọn **Basic Activity**.



1.2 Cập nhật tệp Gradle

Trong **Android Studio**, thêm thủ công các thư viện **Architecture Component** vào các tệp Gradle của bạn.

1. Thêm đoạn mã sau vào tệp **build.gradle (Module: app)**, ở cuối khối **dependencies** (nhưng vẫn nằm trong nó).

```
// Room components
implementation("android.arch.persistence.room:runtime:$rootProject.roomVersion")
annotationProcessor("android.arch.persistence.room:compiler:$rootProject.roomVersion")
androidTestImplementation("android.arch.persistence.room:testing:$rootProject.roomVersion")

// Lifecycle components
implementation("android.arch.lifecycle:extensions:$rootProject.archLifecycleVersion")
annotationProcessor("android.arch.lifecycle:compiler:$rootProject.archLifecycleVersion")
```

Trong tệp build.gradle (Project: RoomWordsSample), thêm số phiên bản ở cuối tệp.

2. Trong tệp build.gradle (Project: RoomWordsSample), thêm số phiên bản ở cuối tệp.

```
ext {
    set("roomVersion", "1.0.0")
    set("archLifecycleVersion", "1.1.0")
}
```

Quan trọng: Hãy sử dụng số phiên bản mới nhất cho các thư viện Room và lifecycle. Để tìm số phiên bản mới nhất:

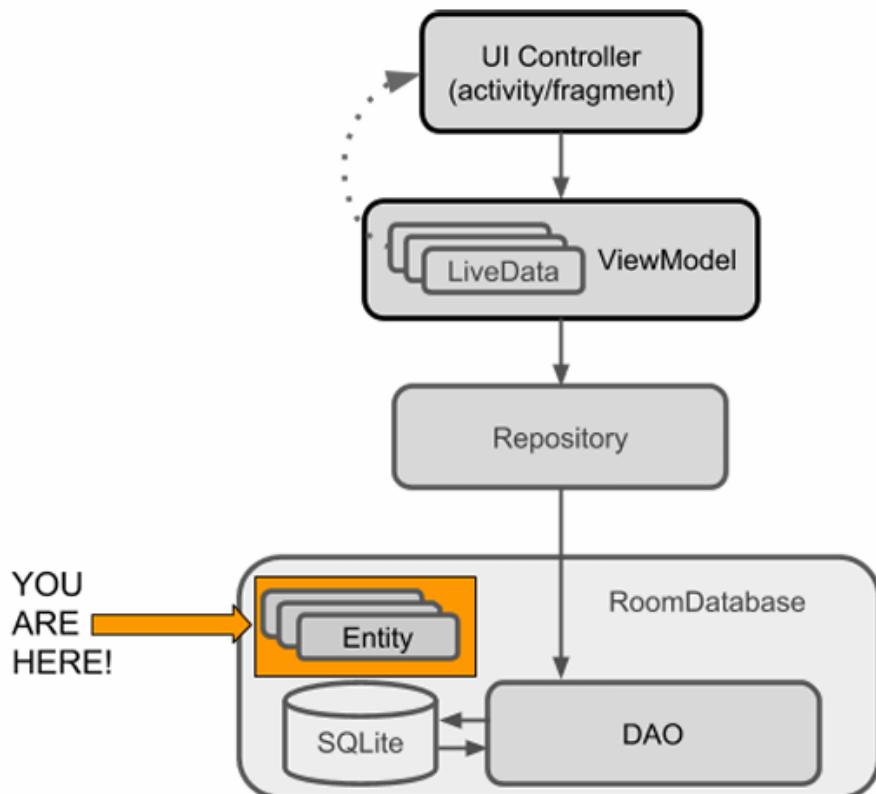
Trên trang Adding Components to your Project, tìm mục cho thành phần, ví dụ như Room. Số phiên bản được định nghĩa ở đầu phần dependencies của thành phần đó.

Ví dụ, số phiên bản Room trong định nghĩa dưới đây là 1.1.1:

```
set("roomVersion", "1.0.0")
```

Nhiệm vụ 2: Tạo Entity Word

Sơ đồ dưới đây là sơ đồ kiến trúc hoàn chỉnh với thành phần mà bạn sẽ triển khai trong nhiệm vụ này được làm nổi bật. Mỗi nhiệm vụ sẽ có một sơ đồ như vậy để giúp bạn hiểu rõ thành phần hiện tại phù hợp như thế nào trong cấu trúc tổng thể của ứng dụng, và cách các thành phần được kết nối với nhau.



Dữ liệu cho ứng dụng này là các từ, và mỗi từ được biểu diễn bởi một thực thể trong cơ sở dữ liệu. Trong nhiệm vụ này, bạn sẽ tạo lớp `Word` và chú thích nó để Room có thể tạo một bảng cơ sở dữ liệu từ đó. Sơ đồ dưới đây cho thấy một bảng cơ sở dữ liệu có tên `word_table`. Bảng này có một cột `word`, cột này cũng đồng thời là khóa chính, và hai hàng, mỗi hàng chứa lần lượt "Hello" và "World."

word_table table
word (primary key, string)
"Hello"
"World"

2.1 Tạo lớp Word

1. Tạo một lớp có tên là `Word` .
2. Thêm một hàm tạo (constructor) nhận một chuỗi `word` làm đối số. Thêm chú thích `@NonNull` để đảm bảo rằng tham số này không bao giờ được phép là `null` .
3. Thêm một phương thức "getter" có tên là `getWord()` trả về từ đó. Room yêu cầu các phương thức "getter" trong các lớp thực thể để nó có thể khởi tạo các đối tượng của bạn.

```
public class Word {  
    2 usages  
    @NonNull  
    private String mWord;  
    // Constructor  
    no usages  
    public Word(@NonNull String word) {  
        this.mWord = word;  
    }  
    // Getter for mWord  
    no usages  
    public String getWord() {  
        return this.mWord;  
    }  
}
```

2.2 Chú thích lớp Word

Để làm cho lớp `Word` có ý nghĩa đối với cơ sở dữ liệu Room, bạn phải chú thích nó. Các chú thích xác định cách từng phần của lớp `Word` liên kết với một mục trong cơ sở dữ liệu. Room sử dụng thông tin này để tạo mã code.

Bạn sử dụng các chú thích sau trong các bước dưới đây:

- `@Entity(tableName = "word_table")`

Mỗi lớp `@Entity` đại diện cho một thực thể trong một bảng. Chú thích khai báo lớp của bạn để chỉ ra rằng lớp đó là một thực thể. Chỉ định tên của bảng nếu bạn muốn nó khác với tên của lớp.

- `@PrimaryKey`

Mọi thực thể cần một khóa chính. Để đơn giản, mỗi từ trong ứng dụng RoomWordsSample đóng vai trò là khóa chính của chính nó. Để tìm hiểu cách tự động tạo các khóa duy nhất, xem mèo bên dưới.

- `@NonNull`

Biểu thị rằng một tham số, trường hoặc giá trị trả về của phương thức không bao giờ được phép là `null`. Khóa chính luôn nên sử dụng chú thích này. Sử dụng chú thích này cho bất kỳ trường bắt buộc nào trong các hàng của bạn.

- `@ColumnInfo(name = "word")`

Chỉ định tên của một cột trong bảng, nếu bạn muốn tên cột khác với tên của biến thành viên.

- Mọi trường được lưu trữ trong cơ sở dữ liệu phải là `public` hoặc có một phương thức "getter". Ứng dụng này cung cấp một phương thức "getter" `getWord()` thay vì để lộ trực tiếp các biến thành viên.

Để xem danh sách đầy đủ các chú thích, hãy tham khảo tài liệu tóm tắt gói Room.

Cập nhật lớp `Word` của bạn với các chú thích, như được hiển thị trong đoạn mã dưới đây.

1. Thêm chú thích `@Entity` vào khai báo lớp và đặt `tableName` thành `"word_table"`.
2. Chú thích biến thành viên `mWord` là `@PrimaryKey`. Yêu cầu `mWord` là `@NonNull` và đặt tên cột là `"word"`.

Lưu ý: Nếu bạn nhập các chú thích bằng tay, Android Studio sẽ tự động nhập (import) mọi thứ bạn cần.

Đây là mã hoàn chỉnh:

```
no usages
@Entity(tableName = "word_table")
public class Word {
    2 usages
    @PrimaryKey
    @NonNull
    @ColumnInfo(name = "word")
    private final String mWord;

    no usages
    public Word(@NonNull String word) {this.mWord = word;}

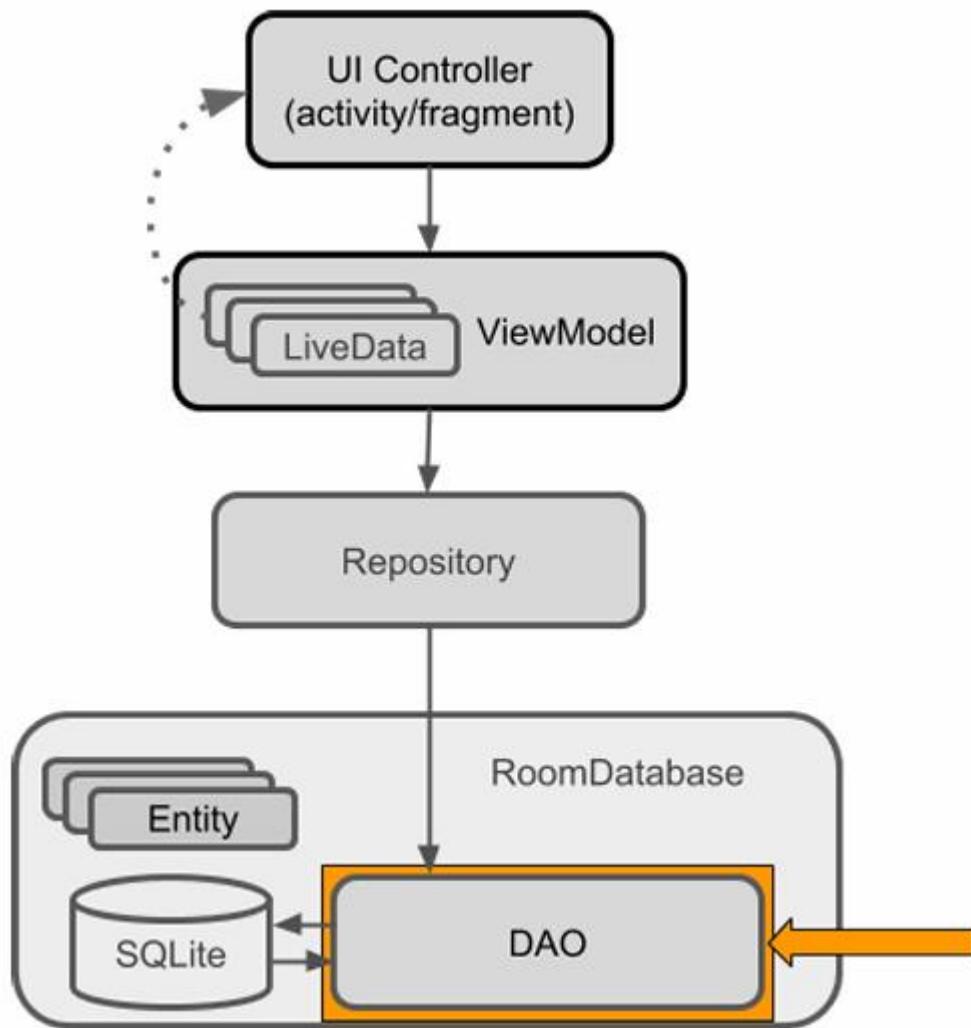
    no usages
    >     public String getWord() { return this.mWord; }
}
```

Nếu bạn gặp lỗi với các chú thích (annotations), bạn có thể nhập chúng theo cách thủ công như sau:

```
import android.arch.persistence.room.ColumnInfo;
import android.arch.persistence.room.Entity;
import android.arch.persistence.room.PrimaryKey;
import android.support.annotation.NonNull;
```

Mẹo về tự động tạo khóa: Để tự động tạo một khóa duy nhất cho mỗi thực thể, bạn cần thêm và chú thích một khóa chính kiểu số nguyên với autoGenerate=true. Xem **Định nghĩa dữ liệu bằng Room entities**.

Nhiệm vụ 3: Tạo DAO



Đối tượng truy cập dữ liệu (Data Access Object - DAO) là một lớp được chú thích, nơi bạn xác định các truy vấn SQL và liên kết chúng với các phương thức. Trình biên dịch sẽ kiểm tra lỗi trong SQL, sau đó tạo truy vấn từ các chú thích. Đối với các truy vấn phổ biến, thư viện cung cấp các chú thích tiện lợi như `@Insert`.

Lưu ý rằng:

- DAO phải là một interface hoặc abstract class.
- Room sử dụng DAO để tạo một API rõ ràng cho mã của bạn.
- Theo mặc định, tất cả các truy vấn (`@Query`) phải được thực thi trên một luồng khác ngoài luồng chính. (Bạn sẽ xử lý điều đó sau.)

- **Đối với các thao tác như chèn hoặc xóa, nếu bạn sử dụng các chú thích tiện lợi được cung cấp, Room sẽ tự động quản lý luồng cho bạn.**

3.1 Triển khai lớp DAO

DAO cho bài thực hành này đơn giản và chỉ cung cấp các truy vấn để lấy tất cả các từ, chèn từ và xóa tất cả các từ.

1. Tạo một interface mới và đặt tên là WordDao.
2. Chú thích khai báo lớp với @Dao để xác định lớp này là một DAO trong Room.
3. Khai báo một phương thức để chèn một từ:

```
void insert(Word word);
```

4. Chú thích phương thức insert() với @Insert. Bạn không cần phải viết bất kỳ SQL nào! (Cũng có các chú thích @Delete và @Update để xóa và cập nhật một hàng, nhưng bạn sẽ không sử dụng các thao tác này trong phiên bản đầu tiên của ứng dụng.)
5. Khai báo một phương thức để xóa tất cả các từ:

```
void deleteAll();
```

6. Không có chú thích tiện lợi để xóa nhiều thực thể, vì vậy hãy chú thích phương thức deleteAll() bằng @Query chung. Cung cấp truy vấn SQL dưới dạng tham số chuỗi cho @Query. Chú thích phương thức deleteAll() như sau:

```
no usages
@Query("DELETE FROM word_table")
```

7. Tạo một phương thức có tên getAllWords() trả về một danh sách (List) các đối tượng Word.

```
no usages
List<Word> getAllWords();
```

8. Chú thích phương thức getAllWords() với một truy vấn SQL để lấy tất cả các từ từ bảng word_table, sắp xếp theo thứ tự bảng chữ cái để tiện lợi:

```
@Query("SELECT * from word_table ORDER BY word ASC")
```

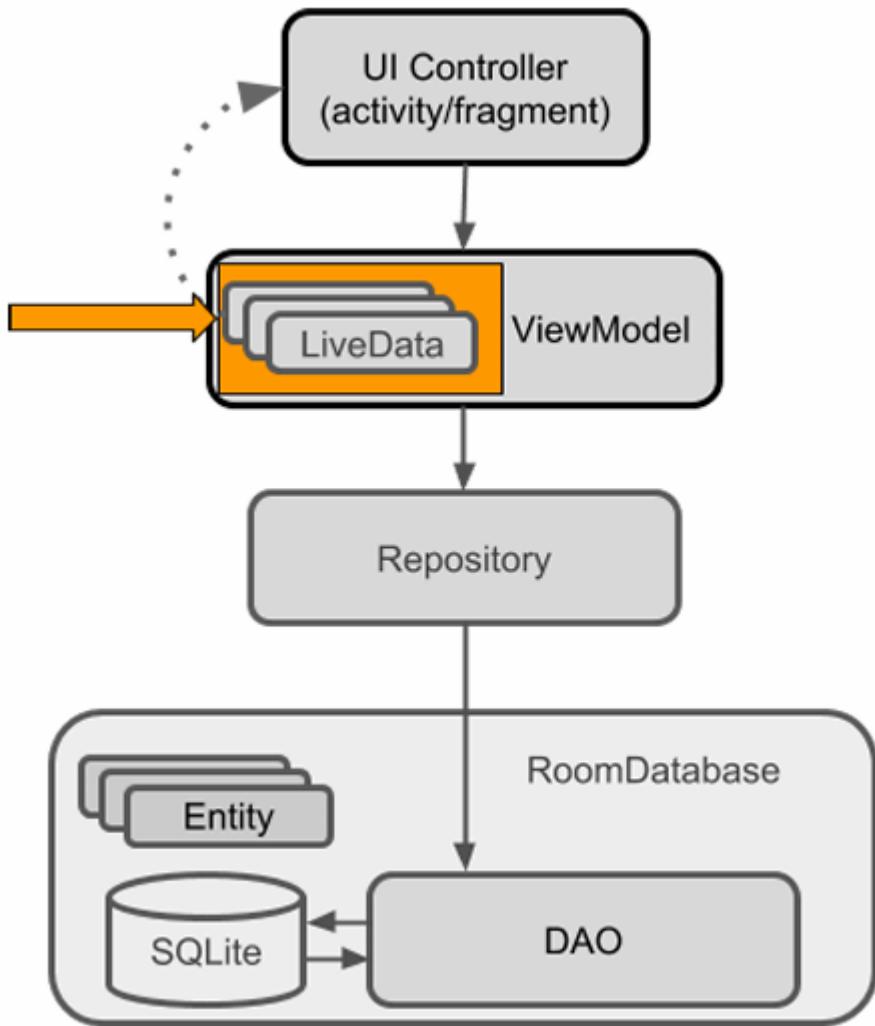
Dưới đây là mã hoàn chỉnh cho lớp WordDao:

```
no usages
@Dao
public interface DAO {
    no usages
    @Insert
    void insert(Word word);

   💡 no usages
    @Query("DELETE FROM word_table")
    void deleteAll();
    no usages
    List<Word> getAllWords();
}
```

Để tìm hiểu thêm về DAO, xem Truy cập dữ liệu bằng Room DAOs.

Task 4: Dùng LiveData



Khi bạn hiển thị dữ liệu hoặc sử dụng dữ liệu theo những cách khác, bạn thường muốn thực hiện một hành động khi dữ liệu thay đổi. Điều này có nghĩa là bạn phải quan sát dữ liệu để có thể phản ứng khi nó thay đổi.

LiveData, một lớp trong thư viện vòng đời (lifecycle library) dành cho việc quan sát dữ liệu, có thể giúp ứng dụng của bạn phản hồi khi dữ liệu thay đổi. Nếu bạn sử dụng kiểu trả về **LiveData** trong mô tả phương thức của mình, Room sẽ tự động tạo tất cả mã cần thiết để cập nhật **LiveData** khi cơ sở dữ liệu được cập nhật.

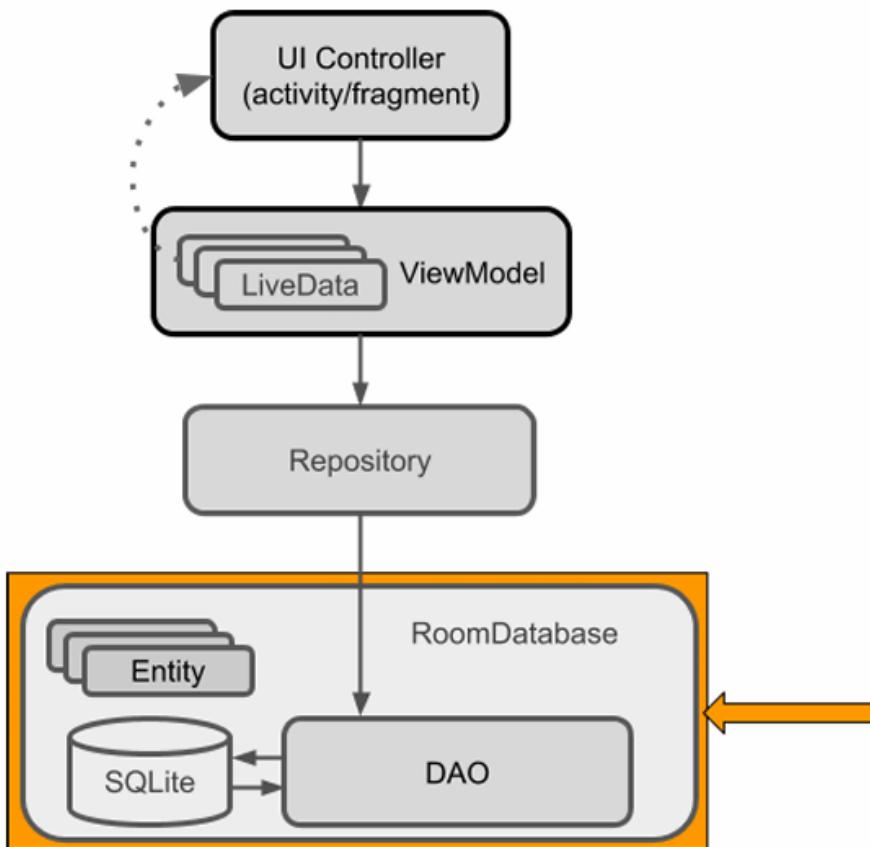
4.1 Trả về LiveData trong WordDao

- Trong giao diện **WordDao**, thay đổi phần khai báo phương thức `getAllWords()` để danh sách `List<Word>` được bọc trong `LiveData<>`.

```
@Query("SELECT * from word_table ORDER BY word ASC")
LiveData<List<Word>> getAllWords();
```

Xem tài liệu **LiveData** để tìm hiểu thêm về các cách khác để sử dụng **LiveData**, hoặc xem video **Architecture Components: LiveData and Lifecycle**.

Nhiệm vụ 5: Thêm một cơ sở dữ liệu Room



Room là một lớp cơ sở dữ liệu được xây dựng trên nền tảng cơ sở dữ liệu SQLite. Room xử lý các tác vụ thông thường mà trước đây bạn phải thực hiện bằng một lớp trợ giúp cơ sở dữ liệu như SQLiteOpenHelper.

- Room sử dụng DAO để thực hiện các truy vấn đến cơ sở dữ liệu của nó.

- Theo mặc định, để tránh hiệu suất UI kém, Room không cho phép thực hiện truy vấn cơ sở dữ liệu trên luồng chính. LiveData tuân theo quy tắc này bằng cách tự động chạy truy vấn bất đồng bộ trên một luồng nền khi cần.
- Room cung cấp kiểm tra cú pháp SQL trong thời gian biên dịch.
- Lớp Room của bạn phải là abstract và kế thừa từ RoomDatabase.
- Thông thường, bạn chỉ cần một instance của cơ sở dữ liệu Room cho toàn bộ ứng dụng.

5.1 Triển khai cơ sở dữ liệu Room

1. Tạo một lớp public abstract kế thừa từ RoomDatabase và đặt tên là WordRoomDatabase.

2.

```
public abstract class WordRoomDatabase extends RoomDatabase {
```

3. Chú thích lớp này là một cơ sở dữ liệu Room bằng cách sử dụng @Database. Xác định các entity thuộc về cơ sở dữ liệu – trong trường hợp này chỉ có Word. (Khai báo các entity sẽ tạo các bảng tương ứng trong database.) Đặt số phiên bản (version).

```
@Database(entities = {Word.class}, version = 1)
```

4. Định nghĩa các DAO làm việc với cơ sở dữ liệu. Cung cấp phương thức trừu tượng (abstract getter) cho mỗi @Dao.

```
no usages  
public abstract WordDao wordDao();
```

5. Tạo WordRoomDatabase dưới dạng một singleton để ngăn việc mở nhiều instance cùng lúc, điều này có thể gây lỗi. Dưới đây là đoạn mã để tạo singleton:

```
public abstract class WordRoomDatabase extends RoomDatabase {  
  
    4 usages  
    private static WordRoomDatabase INSTANCE;  
    no usages  
    static WordRoomDatabase getDatabase(final Context context) {  
        if (INSTANCE == null) {  
            synchronized (WordRoomDatabase.class) {  
                if (INSTANCE == null) {  
                    INSTANCE =  
                        Room.databaseBuilder(context.getApplicationContext(),  
                            WordRoomDatabase.class, "word_database")  
                            // Wipes and rebuilds instead of migrating  
                            // if no Migration object.  
                            // Migration is not part of this practical.  
                            .fallbackToDestructiveMigration().build();  
                }  
            }  
        }  
        return INSTANCE;  
    }  
}
```

6. Thêm đoạn mã để khởi tạo cơ sở dữ liệu vào vị trí có ghi chú Create database here. Đoạn mã này sử dụng Room's database builder để tạo một đối tượng RoomDatabase có tên "word_database" trong application context từ lớp WordRoomDatabase.

```
INSTANCE =  
    Room.databaseBuilder(context.getApplicationContext(),  
        WordRoomDatabase.class, "word_database")  
    build();
```

7. Thêm chiến lược di chuyển (migration) cho cơ sở dữ liệu. Trong bài thực hành này, bạn không cập nhật entity và số phiên bản. Tuy nhiên, nếu bạn thay đổi schema, bạn cần cập nhật version number và xác định cách xử lý migration. Đối với một ứng dụng mẫu như thế này, xóa và tạo lại cơ sở dữ liệu là một chiến lược migration hợp lý. Đối với một ứng dụng thực tế, bạn cần triển khai một chiến lược migration không phá hủy dữ liệu. Xem tài liệu "Understanding migrations with Room" để biết thêm chi tiết.

Thêm đoạn mã sau vào builder, trước khi gọi build()

```
// Wipes and rebuilds instead of migrating
// if no Migration object.
// Migration is not part of this practical.
.fallbackToDestructiveMigration().build();
```

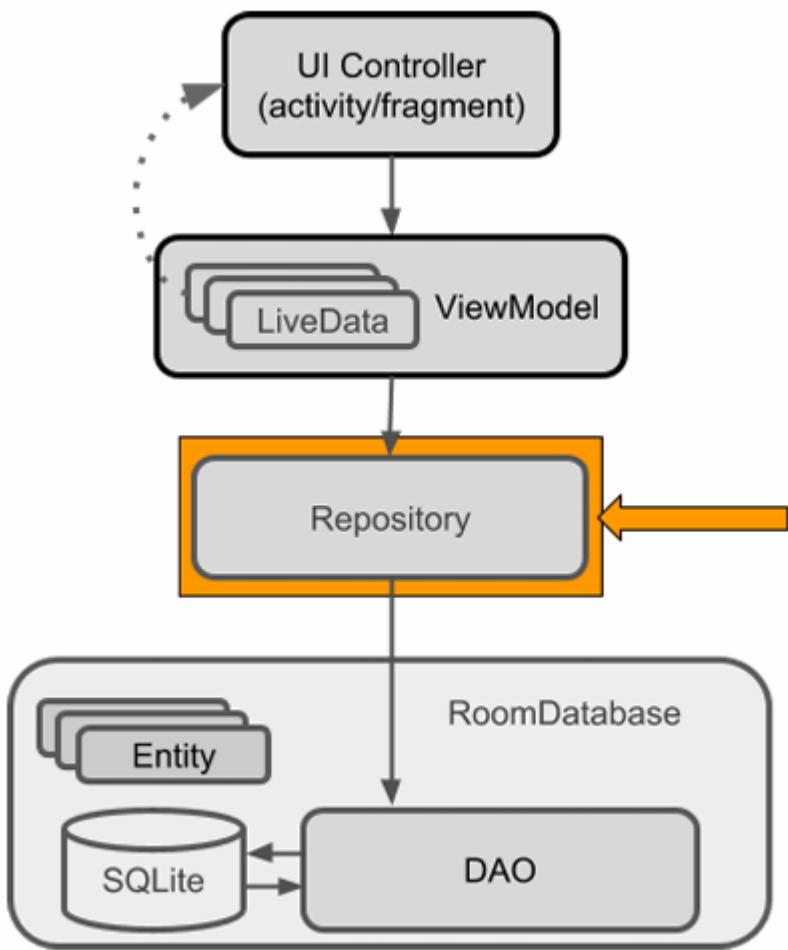
Dưới đây là mã hoàn chỉnh cho toàn bộ lớp **WordRoomDatabase**:

```
4 usages
@Database(entities = {Word.class}, version = 1)
public abstract class WordRoomDatabase extends RoomDatabase {

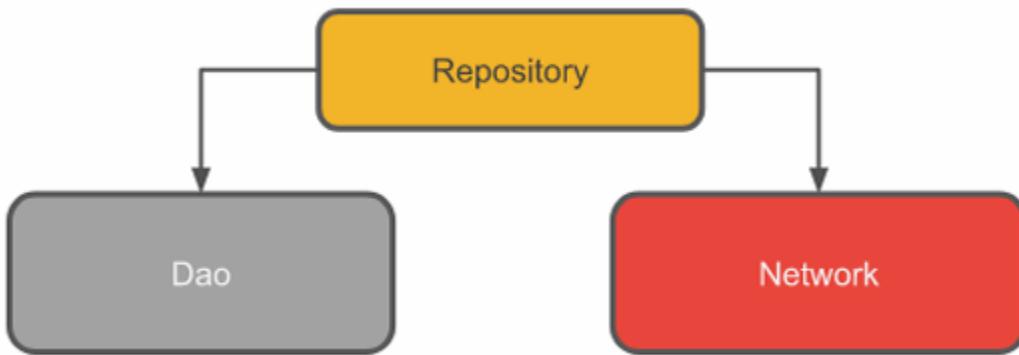
    4 usages
    private static WordRoomDatabase INSTANCE;
    no usages
    public abstract WordDao wordDao();
    no usages
    static WordRoomDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (WordRoomDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE =
                        Room.databaseBuilder(context.getApplicationContext(),
                            WordRoomDatabase.class, "word_database")
                            // Wipes and rebuilds instead of migrating
                            // if no Migration object.
                            // Migration is not part of this practical.
                            .fallbackToDestructiveMigration().build();
                }
            }
        }
        return INSTANCE;
    }
}
```

Quan trọng: Trong Android Studio, nếu bạn gặp lỗi khi dán mã hoặc trong quá trình build, hãy đảm bảo rằng bạn đang sử dụng **tên gói đầy đủ** cho các import. Xem **Thêm thành phần vào dự án (Adding Components to your Project)**. Sau đó, chọn **Build > Clean Project**. Tiếp theo, chọn **Build > Rebuild Project**, rồi build lại.

Nhiệm vụ 6: Tạo Repository



Repository là một lớp trừu tượng giúp truy cập nhiều nguồn dữ liệu khác nhau. **Repository** không phải là một phần của **Architecture Components**, nhưng được khuyến nghị như một **thực tiễn tốt nhất** để tách biệt mã và kiến trúc. Một lớp **Repository** xử lý các thao tác dữ liệu. Nó cung cấp một **API rõ ràng** cho phần còn lại của ứng dụng để làm việc với dữ liệu.



Repository quản lý các luồng truy vấn và cho phép bạn sử dụng nhiều nguồn dữ liệu khác nhau. Trong ví dụ phổ biến nhất, **Repository** triển khai logic để quyết định **nên lấy dữ liệu từ mạng hay sử dụng kết quả được lưu trong cơ sở dữ liệu cục bộ**.

6.1 Triển khai Repository

1. **Tạo một lớp public có tên là WordRepository.**
2. **Thêm các biến thành viên cho DAO và danh sách từ.**

```

private WordDao mWordDao ;
no usages
private LiveData<List<Word>> mAllWords ;

```

3. Thêm một **constructor** để lấy **đối tượng database** và khởi tạo các **biến thành viên**.

```

WordRepository(Application application) {
    WordRoomDatabase db = WordRoomDatabase.getDatabase(application);
    mWordDao = db.wordDao();
    mAllWords = mWordDao.getAllWords();
}

```

4. Thêm một phương thức bao bọc có tên `getAllWords()` để trả về danh sách từ đã lưu dưới dạng **LiveData**. **Room** thực thi tất cả truy vấn trên một **luồng riêng biệt**. **LiveData** sẽ **tự động thông báo** cho observer khi dữ liệu thay đổi.

```
LiveData<List<Word>> getAllWords() {  
    return mAllWords;  
}
```

5. Thêm một phương thức bao bọc cho phương thức `insert()`. Sử dụng **AsyncTask** để gọi `insert()` trên một **luồng không phải UI**, nếu không ứng dụng của bạn sẽ bị crash. **Room** đảm bảo rằng bạn không thực hiện các thao tác **tốn thời gian** trên **luồng chính**, vì điều đó có thể làm **đơ giao diện người dùng (UI)**.

```
public void insert (Word word) {  
    new insertAsyncTask( mWordDao ).execute(word);  
}
```

6. Tạo lớp `insertAsyncTask` như một **lớp nội bộ (inner class)**. Bạn đã quen với **AsyncTask**, vì vậy đây là mã `insertAsyncTask` để bạn có thể sao chép:

```
private static class insertAsyncTask extends AsyncTask<Word, Void, Void> {  
  
    2 usages  
    private WordDao mAsyncTaskDao;  
  
    1 usage  
    insertAsyncTask(WordDao dao) {  
        mAsyncTaskDao = dao;  
    }  
  
    @Override  
    protected Void doInBackground( final Word... params ) {  
        mAsyncTaskDao.insert(params[ 0 ]);  
        return null;  
    }  
}
```

Dưới đây là mã hoàn chỉnh cho lớp **WordRepository**:

```
no usages
public class WordRepository {
    3 usages
    private WordDao mWordDao ;
    2 usages
    private LiveData<List<Word>> mAllWords ;

    no usages
    WordRepository(Application application) {
        WordRoomDatabase db = WordRoomDatabase.getDatabase(application);
        mWordDao = db.wordDao();
        mAllWords = mWordDao.getAllWords();
    }
}
```

```
1 usage
private static class insertAsyncTask extends AsyncTask<Word, Void, Void> {

    2 usages
    private WordDao mAsyncTaskDao ;

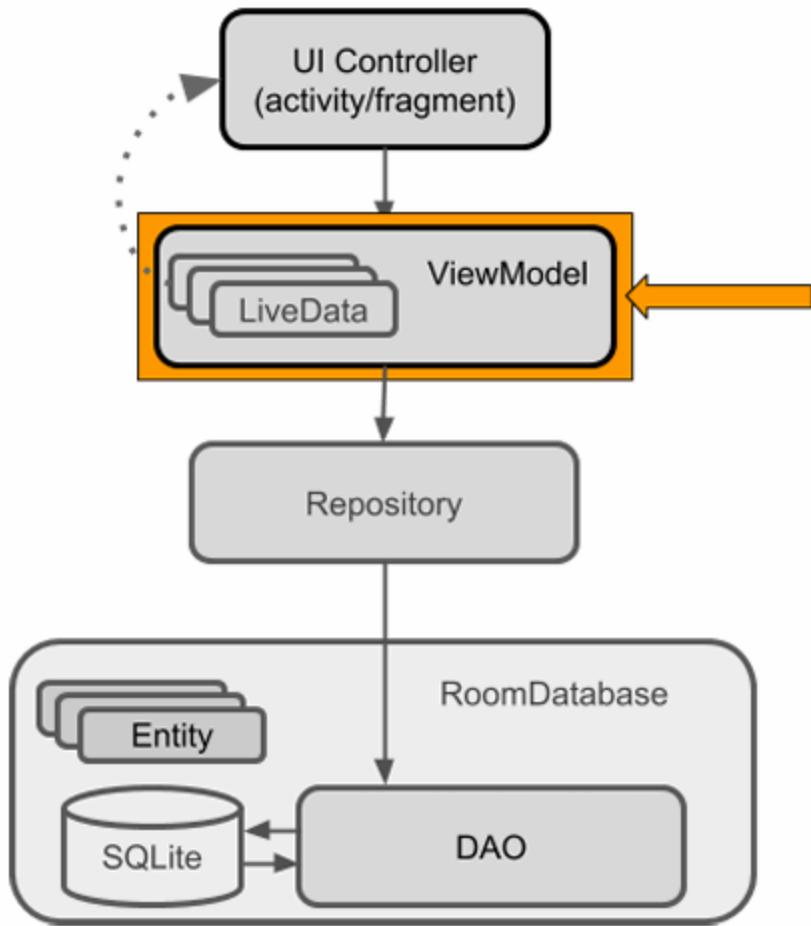
    1 usage
    insertAsyncTask(WordDao dao) {
        mAsyncTaskDao = dao;
    }

    @Override
    protected Void doInBackground( final Word... params) {
        mAsyncTaskDao.insert(params[ 0 ]);
        return null ;
    }
}
```

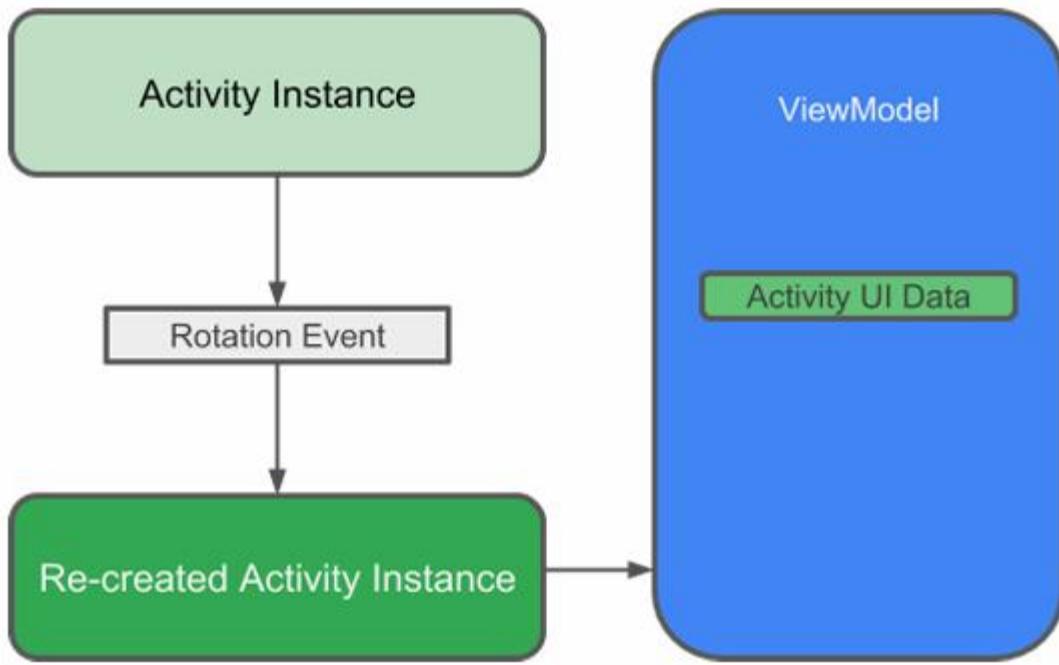
Lưu ý: Trong ví dụ đơn giản này, **Repository** không thực hiện quá nhiều tác vụ.

Để xem một triển khai phức tạp hơn, hãy tham khảo mã **BasicSample** trên **GitHub**.

Nhiệm vụ 7: Tạo ViewModel



ViewModel là một lớp có vai trò **cung cấp dữ liệu cho giao diện người dùng (UI)** và **duy trì dữ liệu khi cấu hình thay đổi**. **ViewModel** đóng vai trò là **trung tâm giao tiếp** giữa **Repository** và **UI**. **ViewModel** là một phần của **lifecycle library**. Để tìm hiểu tổng quan về chủ đề này, hãy xem tài liệu **ViewModel**.



ViewModel lưu trữ dữ liệu giao diện người dùng (UI) của ứng dụng theo cách giúp duy trì dữ liệu khi cấu hình thay đổi. Tách dữ liệu UI khỏi các lớp Activity và Fragment giúp bạn tuân theo nguyên tắc trách nhiệm đơn lẻ (Single Responsibility Principle): Activity và Fragment chịu trách nhiệm hiển thị dữ liệu lên màn hình. ViewModel chịu trách nhiệm lưu trữ và xử lý dữ liệu cần thiết cho UI.

Trong ViewModel, sử dụng LiveData cho dữ liệu có thể thay đổi mà UI sẽ sử dụng hoặc hiển thị.

7.1 Triển khai WordViewModel

- Tạo một lớp có tên **WordViewModel** kế thừa từ **AndroidViewModel**.

Cảnh báo:

- Không bao giờ truyền context vào các instance của ViewModel.
- Không lưu trữ Activity, Fragment, View hoặc Context của chúng trong ViewModel.

Một Activity có thể bị hủy và **tạo lại nhiều lần** trong vòng đời của **ViewModel**, chẳng hạn khi thiết bị **xoay màn hình**. Nếu bạn **lưu tham chiếu** đến Activity trong **ViewModel**, bạn sẽ gặp tình trạng **tham chiếu đến Activity đã bị hủy**, dẫn đến **rò rỉ bộ**

nhớ (memory leak). Nếu cần **application context**, hãy sử dụng **AndroidViewModel**, như được hướng dẫn trong bài thực hành này.

```
no usages
public class WordViewModel extends AndroidViewModel {
    no usages
```

- Thêm một biến thành viên **private** để lưu tham chiếu đến **Repository**.

```
private WordRepository mRepository ;|
```

- Thêm một biến thành viên **LiveData** kiểu **private** để lưu trữ danh sách từ được cache.

```
private LiveData<List<Word>> mAllWords;|
```

- Thêm một **constructor** để lấy tham chiếu đến **WordRepository** và lấy danh sách tất cả các từ từ **WordRepository**.

```
public WordViewModel (Application application) {
    super (application);
    mRepository = new WordRepository(application);
    mAllWords = mRepository .getAllWords();
}
```

- Thêm một phương thức "**getter**" để lấy danh sách tất cả các từ. Điều này giúp **hoàn toàn ẩn việc triển khai** khỏi UI.

```
LiveData<List<Word>> getAllWords() { return mAllWords ; }
```

- Tạo một phương thức bao bọc **insert()** gọi phương thức **insert()** của **Repository**. Bằng cách này, **việc triển khai insert()** được **hoàn toàn ẩn** khỏi UI.

```
public void insert(Word word) { mRepository .insert(word); }
```

Dưới đây là mã hoàn chỉnh cho **WordViewModel**.

```
public class WordViewModel extends AndroidViewModel {  
    3 usages  
    private WordRepository mRepository ;  
    2 usages  
    private LiveData<List<Word>> mAllWords;  
    no usages  
    public WordViewModel (Application application) {  
        super (application);  
        mRepository = new WordRepository(application);  
        mAllWords = mRepository .getAllWords();  
    }  
    no usages  
    LiveData<List<Word>> getAllWords() { return mAllWords ; }  
    no usages  
    public void insert(Word word) { mRepository .insert(word); }  
}
```

Nhiệm vụ 8: Thêm các tệp XML layout cho giao diện người dùng (UI)

Tiếp theo, thêm tệp **XML layout** cho danh sách và các mục sẽ được hiển thị trong **RecyclerView**.

Bài thực hành này giả định rằng bạn đã quen với việc tạo **layout bằng XML**, vì vậy chỉ cung cấp mã nguồn.

8.1 Thêm styles

1. Thay đổi màu sắc trong `colors.xml` thành các giá trị sau (**để sử dụng màu Material Design**):

```
<resources>
    <color name="colorPrimary">#2196F3</color>
    <color name="colorPrimaryLight">#64b5f6</color>
    <color name="colorPrimaryDark">#1976D2</color>
    <color name="colorAccent">#FFFF9800</color>
    <color name="colorTextPrimary">@android:color/white</color>
    <color name="colorScreenBackground">#fff3e0</color>
    <color name="colorTextHint">#E0E0E0</color>
</resources>
```

2. Thêm một style cho TextView trong tệp `values/styles.xml`.

```
<style name="text_view_style">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textAppearance">
        @android:style/TextAppearance.Large</item>
    <item name="android:background">@color/colorPrimaryLight</item>
    <item name="android:layout_marginTop">8dp</item>
    <item name="android:layout_gravity">center</item>
    <item name="android:padding">16dp</item>
        <item name="android:textColor">@color/colorTextPrimary</item>
    </style>
```

8.2 Thêm bố cục mục

Thêm một bố cục `layout/recyclerview_item.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/text_view_style"
        tools:text="placeholder text" />
</LinearLayout> |
```

8.3 Thêm RecyclerView

- Trong tệp **layout/content_main.xml**, thêm màu nền cho **ConstraintLayout**.

```
    android:background="@color/colorScreenBackground"
```

- Trong tệp **content_main.xml**, thay thế phần tử **TextView** bằng phần tử **RecyclerView**.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerview"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_margin="16dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />
```

8.4 Sửa biểu tượng trong FAB

Biểu tượng trong nút hành động nổi (FAB) nên phù hợp với hành động có sẵn.
Trong tệp `layout/activity_main.xml`, đặt biểu tượng dấu "+" cho **FloatingActionButton**:

1. Chọn **File > New > Vector Asset**.
2. Chọn **Material Icon**.
3. Nhấp vào biểu tượng robot Android trong trường **Icon**, sau đó chọn tài sản "+" ("add").
4. Trong tệp `layout/activity_main.xml`, trong **FloatingActionButton**, thay đổi thuộc tính **srcCompat** thành:

```
android :src= "@drawable/ic_add_black_24dp"
```

Nhiệm vụ 9: Tạo Adapter và thêm RecyclerView

Bạn sẽ hiển thị dữ liệu trong RecyclerView, cách này trông đẹp hơn so với chỉ đặt dữ liệu trong TextView. Thực hành này giả định rằng bạn đã hiểu cách hoạt động của RecyclerView, RecyclerView.LayoutManager, RecyclerView.ViewHolder, và RecyclerView.Adapter.

9.1 Tạo lớp WordListAdapter

● Thêm một lớp **WordListAdapter** kế thừa từ **RecyclerView.Adapter**. Adapter này sẽ lưu trữ dữ liệu và đổ dữ liệu vào **RecyclerView**. Lớp nội bộ **WordViewHolder** sẽ giữ và quản lý một view cho một mục trong danh sách.

Đây là mã nguồn:

```
public class WordListAdapter extends RecyclerView.Adapter<WordListAdapter.WordViewHolder> {
    2 usages
    private final LayoutInflater mInflater;
    5 usages
    private List<Word> mWords; // Cached copy of words

    no usages
    public WordListAdapter(Context context) {
        mInflater = LayoutInflater.from(context);
    }

    @NonNull
    @Override
    public WordViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View itemView = mInflater.inflate(R.layout.recyclerview_item, parent, attachToRoot: false);
        return new WordViewHolder(itemView);
    }

    @Override
    public void onBindViewHolder(@NonNull WordViewHolder holder, int position) {
        if (mWords != null) {
            Word current = mWords.get(position);
            holder.wordItemView.setText(current.getWord());
        } else {
            // Trường hợp dữ liệu chưa sẵn sàng
            holder.wordItemView.setText("No Word");
        }
    }
}
```

```
no usages
public void setWords(List<Word> words) {
    mWords = words;
    notifyDataSetChanged();
}

@Override
public int getItemCount() {
    return (mWords != null) ? mWords.size() : 0;
}

4 usages
public static class WordViewHolder extends RecyclerView.ViewHolder {
    3 usages
    private final TextView wordItemView;

    1 usage
    public WordViewHolder(View itemView) {
        super(itemView);
        wordItemView = itemView.findViewById(R.id.textView);
    }
}
}
```

Lưu ý: Biến mWords trong adapter lưu trữ dữ liệu tạm thời. Trong nhiệm vụ tiếp theo, bạn sẽ thêm mã để cập nhật dữ liệu tự động.

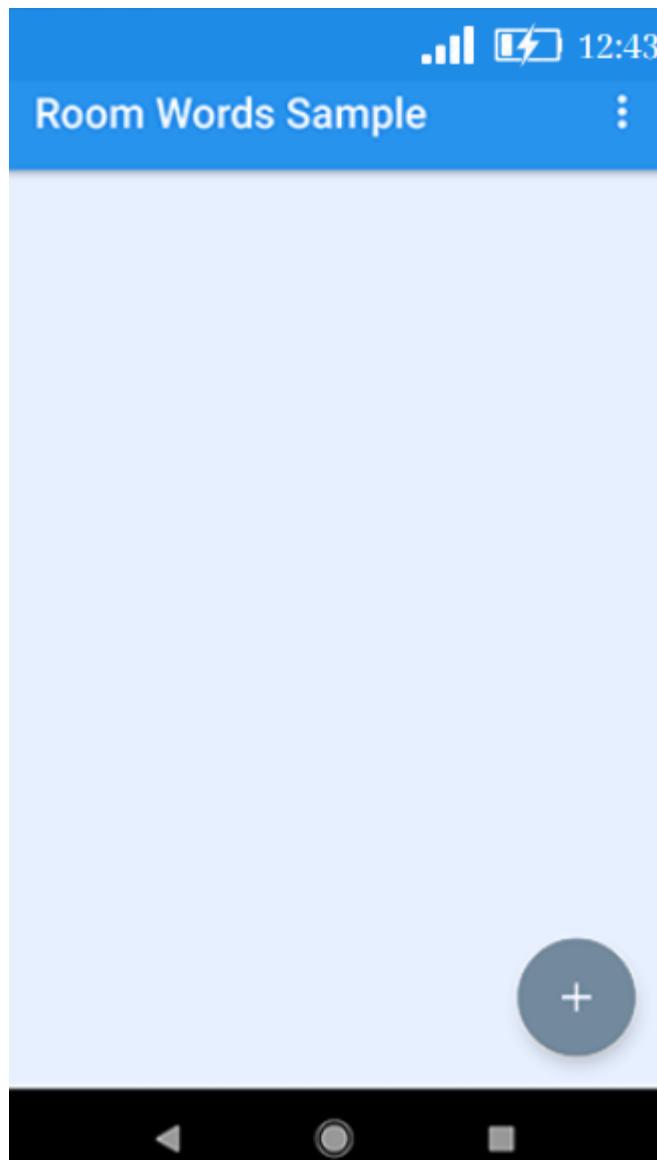
Lưu ý: Phương thức getItemCount() cần xử lý trường hợp dữ liệu chưa sẵn sàng và mWords vẫn còn null. Trong một ứng dụng phức tạp hơn, bạn có thể hiển thị dữ liệu tạm thời hoặc nội dung thay thế có ý nghĩa cho người dùng.

9.2 Thêm RecyclerView vào MainActivity

- Thêm **RecyclerView** vào phương thức **onCreate()** của **MainActivity**.

```
RecyclerView recyclerView = findViewById(R.id.recyclerview);
final WordListAdapter adapter = new WordListAdapter(mainActivity: this);
recyclerView.setAdapter(adapter);
recyclerView.setLayoutManager(new LinearLayoutManager(context: this));
```

- Chạy ứng dụng của bạn để đảm bảo rằng nó biên dịch và chạy thành công. Hiện tại chưa có mục nào vì bạn chưa kết nối dữ liệu. Ứng dụng sẽ hiển thị một **RecyclerView** trống.



Nhiệm vụ 10: Điền dữ liệu vào cơ sở dữ liệu

Hiện tại, cơ sở dữ liệu chưa có dữ liệu. Bạn sẽ thêm dữ liệu theo hai cách: **Thêm một số dữ liệu khi cơ sở dữ liệu được mở**. **Thêm một Activity để nhập từ mới**. Mỗi khi cơ sở dữ liệu được mở, toàn bộ nội dung sẽ bị xóa và điền lại. Đây là một giải pháp hợp lý cho một ứng dụng mẫu, nơi bạn thường muốn bắt đầu lại với dữ liệu sạch.

10.1 Tạo Callback để điền dữ liệu vào cơ sở dữ liệu

Để xóa toàn bộ nội dung và điền lại dữ liệu mỗi khi ứng dụng khởi động, bạn cần tạo một **RoomDatabase.Callback** và ghi đè phương thức **onOpen()**. Vì không thể thực hiện các thao tác với cơ sở dữ liệu Room trên luồng giao diện người dùng (UI thread), nên **onOpen()** sẽ tạo và thực thi một **AsyncTask** để thêm dữ liệu vào cơ sở dữ liệu.

1. Thêm callback **onOpen()** vào lớp **WordRoomDatabase**.

```
private static RoomDatabase.Callback sRoomDatabaseCallback =  
    new RoomDatabase.Callback(){  
        8 usages  
        @Override  
        public void onOpen ( @NonNull SupportSQLiteDatabase db){  
            super .onOpen(db);  
            new PopulateDbAsync( INSTANCE ).execute();  
        }  
    };
```

1. Tạo một lớp nội bộ **PopulateDbAsync** kế thừa từ **AsyncTask**. Triển khai phương thức **doInBackground()** để xóa tất cả các từ, sau đó tạo các từ mới. Dưới đây là mã **AsyncTask** để xóa nội dung trong cơ sở dữ liệu, sau đó điền vào một danh sách từ ban đầu. Bạn có thể sử dụng các từ của riêng mình!

```
public class PopulateDbAsync extends AsyncTask<Void, Void, Void> {
    3 usages
    private final WordDao mDao;
    2 usages
    String[] words = {"dolphin", "crocodile", "cobra"};
    no usages
    PopulateDbAsync(WordRoomDatabase db) {
        mDao = db.wordDao();
    }

    @Override
    protected Void doInBackground(Void... params) {
        // Start the app with a clean database every time.
        // Not needed if you only populate the database
        // when it is first created
        mDao.deleteAll();
        for (int i = 0; i <= words.length; i++) {
            Word word = new Word(words[i]);
            mDao.insert(word);
        }
        return null;
    }
}
```

1. Thêm **callback** vào chuỗi xây dựng cơ sở dữ liệu trong **WordRoomDatabase**, ngay trước khi gọi **.build()**.

```
.addCallback( sRoomDatabaseCallback )
```

Nhiệm vụ 11: Kết nối giao diện người dùng với dữ liệu

Bây giờ bạn đã tạo phương thức để điền dữ liệu ban đầu vào cơ sở dữ liệu, bước tiếp theo là thêm mã để hiển thị các từ đó trong **RecyclerView**.

Để hiển thị nội dung hiện tại của cơ sở dữ liệu, bạn cần thêm một **observer** để quan sát **LiveData** trong **ViewModel**. Mỗi khi dữ liệu thay đổi (bao gồm cả khi dữ liệu được khởi tạo), callback **onChanged()** sẽ được gọi. Trong trường hợp này, callback **onChanged()** sẽ gọi phương thức **setWords()** của adapter để cập nhật dữ liệu đã lưu trong bộ nhớ đệm của adapter và làm mới danh sách hiển thị.

11.1 Hiển thị danh sách từ

- Trong **MainActivity**, tạo một biến thành viên cho **ViewModel**, vì tất cả các tương tác của **Activity** đều thông qua **WordViewModel**.

```
private WordViewModel mWordViewModel;
```

- Trong phương thức **onCreate()**, lấy một **ViewModel** từ lớp **ViewModelProviders**.

```
mWordViewModel = Path.of(String.valueOf( obj: this )).getRoot(WordViewModel. class );
```

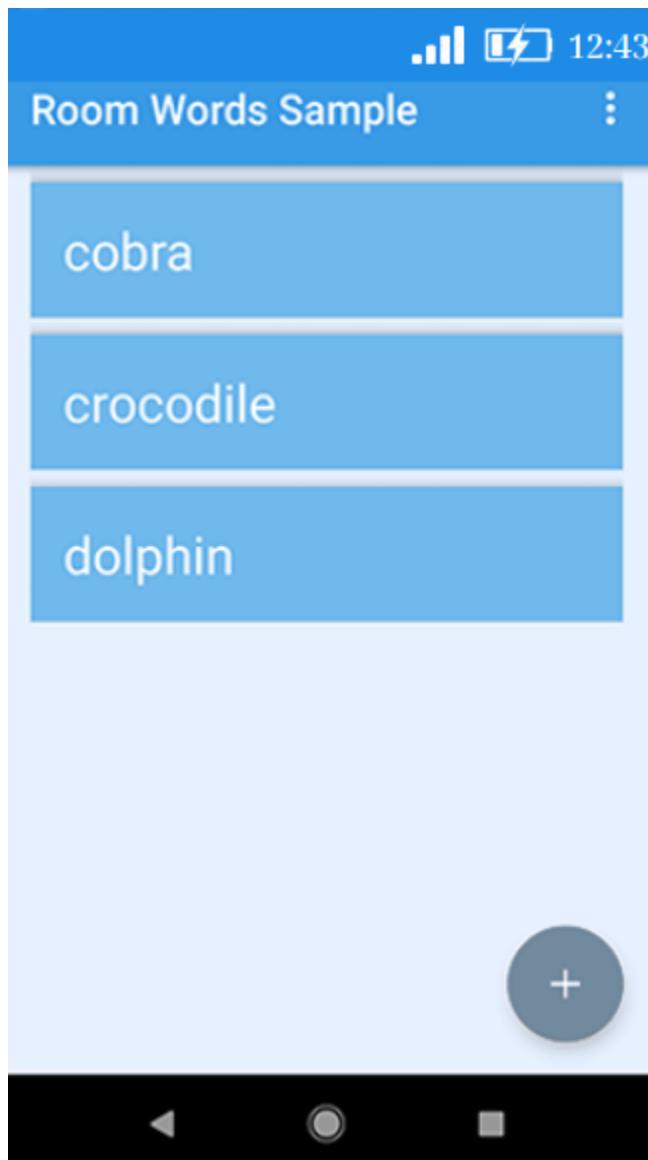
Sử dụng **ViewModelProviders** để liên kết **ViewModel** với bộ điều khiển giao diện người dùng (**UI controller**). Khi ứng dụng khởi chạy lần đầu, lớp **ViewModelProviders** sẽ tạo **ViewModel**. Nếu **Activity** bị hủy (ví dụ: do thay đổi cấu hình), **ViewModel** vẫn được giữ lại. Khi **Activity** được tạo lại, **ViewModelProviders** sẽ trả về **ViewModel** hiện có.

- Cũng trong phương thức **onCreate()**, thêm một **observer** cho **LiveData** được trả về bởi **getAllWords()**. Khi dữ liệu được quan sát thay đổi trong lúc **Activity** đang ở trạng thái foreground, phương thức **onChanged()** sẽ được gọi và cập nhật dữ liệu được lưu trong bộ nhớ đệm của adapter. Lưu ý rằng trong trường

hợp này, khi ứng dụng mở ra, dữ liệu ban đầu được thêm vào, vì vậy phương thức **onChanged()** sẽ được gọi.

```
mWordViewModel .getAllWords().observe( owner: this ,  new  Observer<List<Word>>() {  
    @Override  
    public void onChanged( @Nullable final List<Word> words) {  
        // Update the cached copy of the words in the adapter.  
        WordListAdapter adapter = null;  
        adapter .setWords(words);  
    }  
});
```

2. Chạy ứng dụng. Bộ dữ liệu ban đầu sẽ xuất hiện trong **RecyclerView**.



Nhiệm vụ 12: Tạo một Activity để thêm từ

Bây giờ, bạn sẽ thêm một **Activity** cho phép người dùng sử dụng **FAB** để nhập từ mới.
Đây là giao diện của **Activity** mới:



12:43

Room Words Sample

tiger

Save

