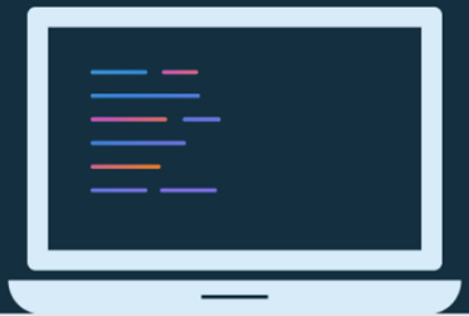
Bài học 8: Cấu trúc ứng dụng (lớp giao diện người dùng)



Giới thiệu về bài học này

Bài học 8: Cấu trúc ứng dụng (lớp giao diện người dùng)

- Cấu trúc ứng dụng Android
- ViewModel
- Liên kết dữ liệu
- LiveData
- Biến đổi LiveData
- Tóm tắt

Cấu trúc ứng dụng Android

Tránh các quyết định ngắn hạn

- Các yếu tố bên ngoài, chẳng hạn như thời hạn gấp, có thể dẫn đến những quyết định thiếu sáng suốt về thiết kế và cấu trúc ứng dụng.
- Các quyết định gây ra những hậu quả đối với công việc sau này (việc duy trì lâu dài ứng dụng có thể khó khăn hơn).
- Cần phải đảm bảo cả giao sản phẩm đúng hạn lẫn nhiệm vụ bảo trì sau này.

Ví dụ về các quyết định ngắn hạn

- Điều chỉnh ứng dụng của bạn theo một thiết bị cụ thể
- Sao chép và dán mã vào tệp của bạn một cách thiếu sáng suốt
- Đặt mọi logic nghiệp vụ vào tệp hoạt động
- Mã hóa cứng các chuỗi mà người dùng thấy được trong mã của bạn

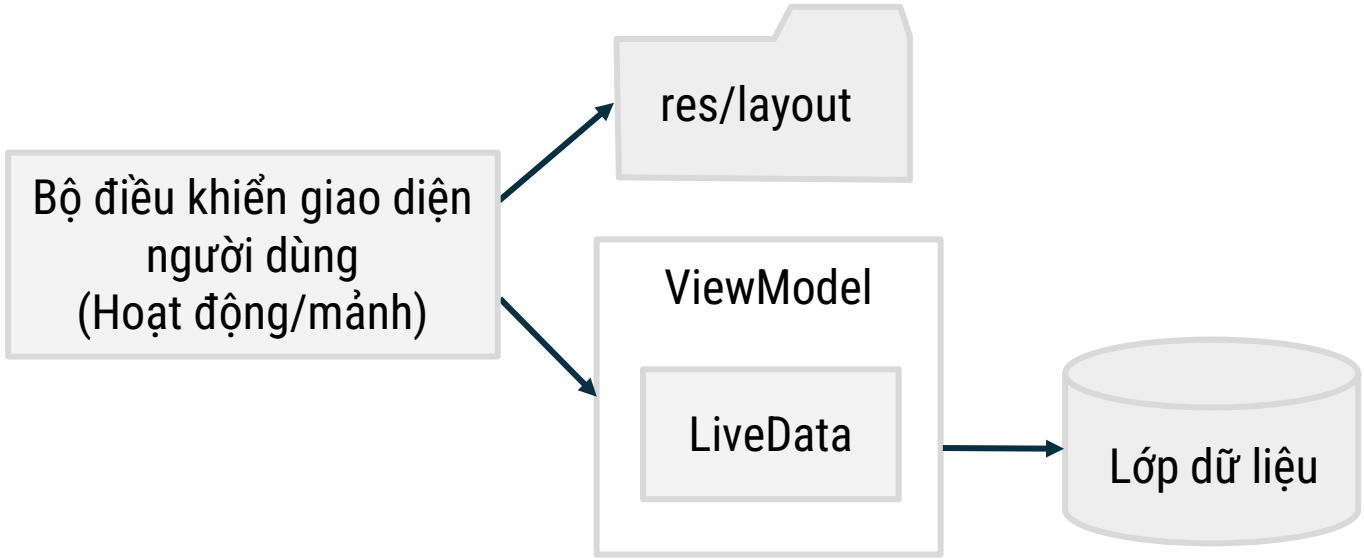
Tại sao bạn cần có cấu trúc ứng dụng tốt

- Xác định rõ nơi cần đặt logic nghiệp vụ cụ thể
- Giúp các nhà phát triển dễ dàng cộng tác hơn
- Giúp mã của bạn dễ kiểm tra hơn
- Cho phép bạn hưởng lợi từ những vấn đề đã được giải quyết
- Tiết kiệm thời gian và giảm món nợ kỹ thuật khi bạn mở rộng ứng dụng của mình

Android Jetpack

- Thư viện Android kết hợp các phương pháp hay nhất và cung cấp khả năng tương thích ngược trong các ứng dụng của bạn
- Jetpack bao gồm các thư viện gói `androidx.*`

Tách biệt vấn đề



Các thành phần cấu trúc

- Các mẫu thiết kế cấu trúc, như MVVM và MVI, mô tả một mẫu chung chung cho cấu trúc ứng dụng mà bạn cần xây dựng.
- Các thành phần cấu trúc của Jetpack giúp bạn thiết kế những ứng dụng mạnh mẽ, có thể kiểm tra và bảo trì.

ViewModel

Gradle: hàm mở rộng vòng đời

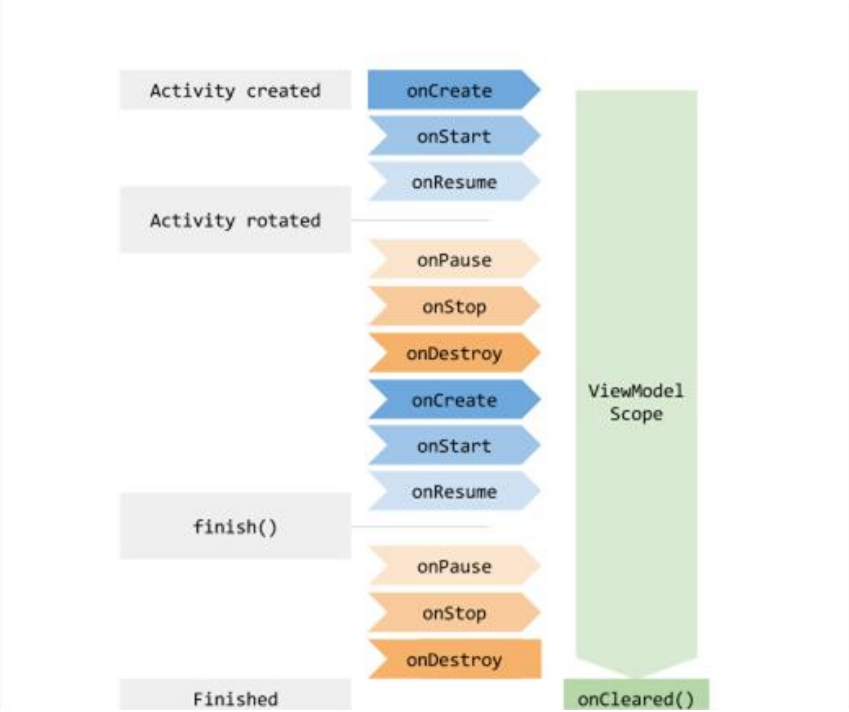
Trong tệp `app/build.gradle`:

```
dependencies {
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
    implementation "androidx.activity:activity-ktx:$activity_version"
}
```

ViewModel

- Chuẩn bị dữ liệu cho giao diện người dùng
- Không được tham chiếu đến hoạt động, mảnh hoặc chế độ xem trong hệ phân cấp chế độ xem
- Trong phạm vi một vòng đời (hoạt động và mảnh có vòng đời)
- Cho phép giữ lại dữ liệu trong các thay đổi về cấu hình
- Tiếp tục giữ lại miễn sao phạm vi vẫn còn

Thời gian tồn tại của ViewModel



Kabaddi Kounter

Lớp ViewModel

lớp trừu tượng ViewModel

Summary

Public constructors	
`<init>()`	
ViewModel is a class that is responsible for preparing and managing the data for an `Activity` or a `Fragment`.	

Protected methods	
open Unit	`onCleared()` This method will be called when this ViewModel is no longer used and will be destroyed.

Extension properties	
From androidx.lifecycle	
CoroutineScope	`viewModelScope` CoroutineScope tied to this ViewModel.

Triển khai ViewModel

```
class ScoreViewModel : ViewModel() {
    var scoreA : Int = 0
    var scoreB : Int = 0
    fun incrementScore(isTeamA: Boolean) {
        if (isTeamA) {
            scoreA++
        }
        else {
            scoreB++
        }
    }
}
```

Tải và sử dụng ViewModel

```
class MainActivity : AppCompatActivity() {
    // Delegate provided by androidx.activity.viewModels
    val viewModel: ScoreViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        val scoreViewA: TextView = findViewById(R.id.scoreA)
        scoreViewA.text = viewModel.scoreA.toString()
    }
```

Dùng ViewModel

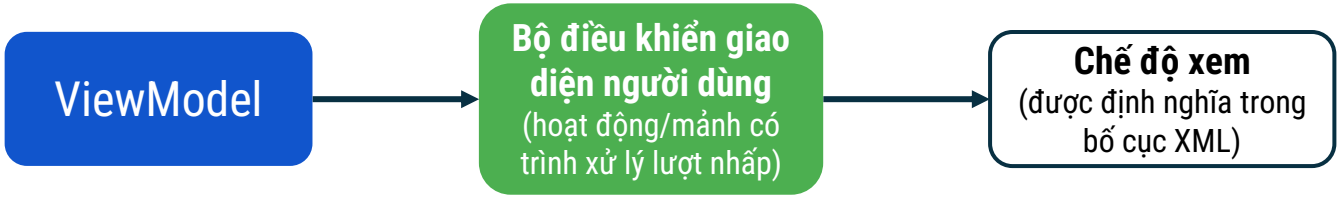
Trong `onCreate()` của `MainActivity`:

```
val scoreViewA: TextView = findViewById(R.id.scoreA)
val plusOneButtonA: Button = findViewById(R.id.plusOne_teamA)

plusOneButtonA.setOnClickListener {
    viewModel.incrementScore(true)
    scoreViewA.text = viewModel.scoreA.toString()
}
```

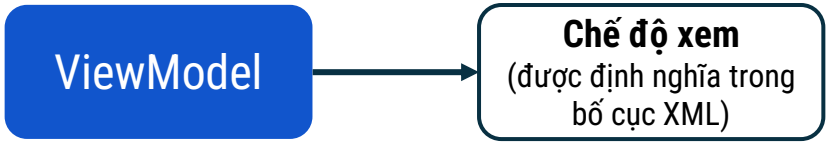
Liên kết dữ liệu

ViewModel và liên kết dữ liệu

- Cấu trúc ứng dụng không có liên kết dữ liệu



- ViewModel có thể hoạt động đồng thời với liên kết dữ liệu



Liên kết dữ liệu trong XML được xem lại

Chỉ định ViewModel trong thẻ `data` của liên kết.

```
<layout>
    <data>
        <variable>
            name="viewModel"
            type="com.example.kabaddikounter.ScoreViewModel" />
    </data>
    <ConstraintLayout ../>
</layout>
```

Đính kèm ViewModel vào một liên kết dữ liệu

```
class MainActivity : AppCompatActivity() {

    val viewModel: ScoreViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding: ActivityMainBinding = DataBindingUtil.setContentView(this,
            R.layout.activity_main)

        binding.viewModel = viewModel

        ...
```

Dùng ViewModel trên một liên kết dữ liệu

Trong tệp `activity_main.xml`:

```
<TextView
    android:id="@+id/scoreViewA"
    android:text="@{viewModel.scoreA.toString()}" />
        ...
```

ViewModel và liên kết dữ liệu

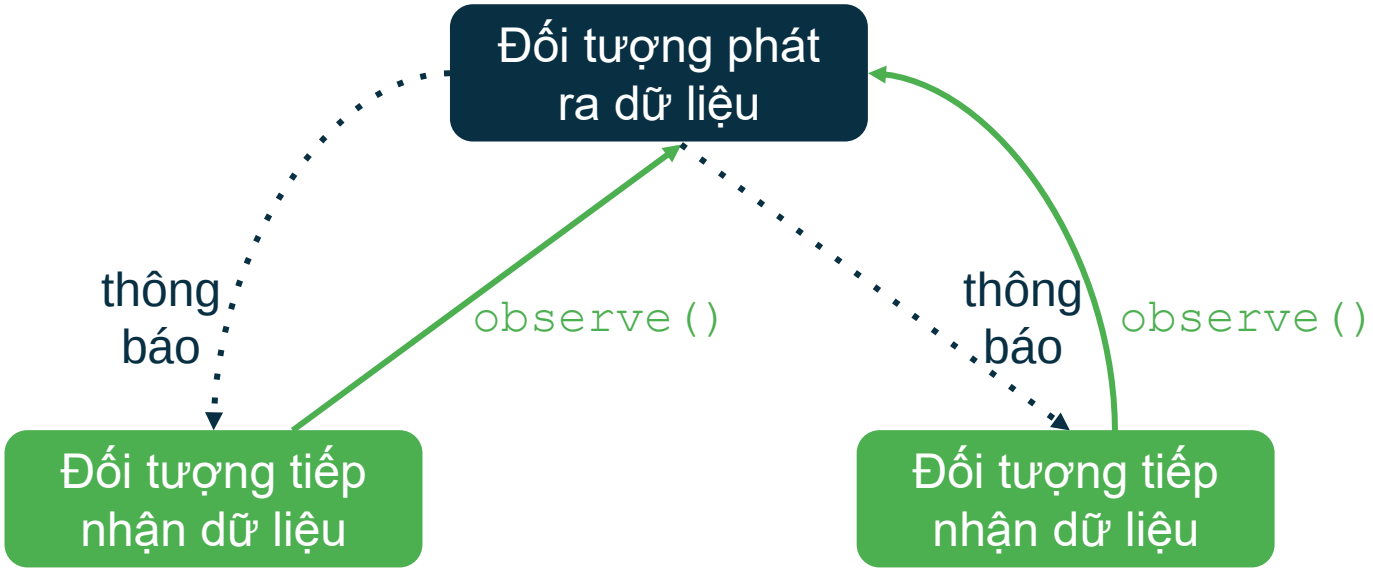
```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    val binding: ActivityMainBinding = DataBindingUtil.setContentView(this,
        R.layout.activity_main)

    binding.plusOneButtonA.setOnClickListener {
        viewModel.incrementScore(true)
        binding.scoreViewA.text = viewModel.scoreA.toString()
    }
}
```

LiveData

Mẫu thiết kế đối tượng tiếp nhận dữ liệu

- Chủ thể duy trì danh sách các đối tượng tiếp nhận dữ liệu để thông báo khi trạng thái thay đổi.
- Đối tượng tiếp nhận dữ liệu nhận được các thay đổi về trạng thái từ chủ thể và thực thi mã thích hợp.
- Có thể thêm hoặc xóa đối tượng tiếp nhận dữ liệu bất cứ lúc nào.

Sơ đồ mẫu thiết kế đối tượng tiếp nhận dữ liệu



LiveData

- Một lớp lưu giữ dữ liệu nhận biết vòng đời có thể quan sát được
- Trình bao bọc có thể được dùng với bất kỳ dữ liệu nào bao gồm cả danh sách (ví dụ: `LiveData<Int>` lưu giữ `Int`)
- Thường được ViewModel sử dụng để lưu giữ các trường dữ liệu riêng lẻ
- Có thể thêm hoặc xóa đối tượng tiếp nhận dữ liệu (hoạt động hoặc mảnh) bất cứ lúc nào.
  - `observe(owner: LifecycleOwner, observer: Observer)`
    `removeObserver(observer: Observer)`

LiveData và MutableLiveData

`LiveData<T>`	`MutableLiveData<T>`
• `getValue()`	• `getValue()` • `postValue(value: T)` • `setValue(value: T)`

Đây là loại dữ liệu được lưu trữ trong `LiveData` hoặc `MutableLiveData`.

Dùng LiveData trong ViewModel

```
class ScoreViewModel : ViewModel() {

    private val _scoreA = MutableLiveData<Int>(0)
    val scoreA: LiveData<Int>
        get() = _scoreA

    fun incrementScore(isTeamA: Boolean) {
        if (isTeamA) {
            _scoreA.value = _scoreA.value!! + 1
        }
        ...
```

Thêm đối tượng tiếp nhận dữ liệu vào LiveData

Thiết lập trình xử lý lượt nhấp để gia tăng điểm số trong `ViewModel`:

```
binding.plusOneButtonA.setOnClickListener {
    viewModel.incrementScore(true)
}
```

Tạo đối tượng tiếp nhận dữ liệu để cập nhật điểm số của đội A trên màn hình:

```
val scoreA_Observer = Observer<Int> { newValue ->
    binding.scoreViewA.text = newValue.toString()
}
```

Thêm đối tượng tiếp nhận dữ liệu vào `LiveData scoreA` trong `ViewModel`:

```
viewModel.scoreA.observe(this, scoreA_Observer)
```

Liên kết dữ liệu hai chiều

- Chúng ta đã có liên kết dữ liệu hai chiều với `ViewModel` và `LiveData`.
- Liên kết với `LiveData` trong tệp XML sẽ loại bỏ nhu cầu thêm đối tượng tiếp nhận dữ liệu trong mã.

Ví dụ về tệp XML của bố cục

```
<layout>
    <data>
        <variable>
            name="viewModel"
            type="com.example.kabaddikounter.ScoreViewModel" />
    </data>

    <ConstraintLayout ..>
        <TextView ...
            android:id="@+id/scoreViewA"
            android:text="@{viewModel.scoreA.toString()}" />
        ...
    </ConstraintLayout>
</layout>
```

Ví dụ về hoạt động

```
class MainActivity : AppCompatActivity() {
    val viewModel: ScoreViewModel by viewModels()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding: ActivityMainBinding = DataBindingUtil
             .setContentView(this, R.layout.activity_main)

        binding.viewModel = viewModel
        binding.lifecycleOwner = this

        binding.plusOneButtonA.setOnClickListener {
            viewModel.incrementScore(true)
        }
        ...
```

Ví dụ về ViewModel

```
class ScoreViewModel : ViewModel() {
    private val _scoreA = MutableLiveData<Int>(0)
    val scoreA : LiveData<Int>
        get() = _scoreA
    private val _scoreB = MutableLiveData<Int>(0)
    val scoreB : LiveData<Int>
        get() = _scoreB
    fun incrementScore(isTeamA: Boolean) {
        if (isTeamA) {
            _scoreA.value = _scoreA.value!! + 1
        } else {
            _scoreB.value = _scoreB.value!! + 1
        }
    }
}
```

Biến đổi LiveData

Điều chỉnh LiveData có phép biến đổi

`LiveData` có thể được biến đổi thành đối tượng mới của `LiveData`.

- `map()`
- `switchMap()`

Ví dụ về LiveData có phép biến đổi

```
val result: LiveData<String> = Transformations.map(viewModel.scoreA) {
    x -> if (x > 10) "A Wins" else ""
}
```

Tóm tắt

Tóm tắt

Trong Bài học 8, bạn đã tìm hiểu cách:

- Tuân theo thiết kế cấu trúc ứng dụng tốt, cũng như nguyên tắc tách biệt vấn đề để làm cho các ứng dụng dễ bảo trì hơn và giảm món nợ kỹ thuật
- Tạo `ViewModel` để lưu giữ dữ liệu riêng biệt với bộ điều khiển giao diện người dùng
- Dùng `ViewModel` với liên kết dữ liệu để tạo giao diện người dùng phản hồi nhanh hơn bằng ít mã hơn
- Dùng đối tượng tiếp nhận dữ liệu để tự động nhận thông tin cập nhật từ `LiveData`

Tìm hiểu thêm

- Hướng dẫn về cấu trúc ứng dụng
- Android Jetpack
- Tổng quan về ViewModel
- Ứng dụng mẫu theo cấu trúc Android
- ViewModelProvider
- Tải dữ liệu nhận biết vòng đời với các thành phần cấu trúc
- ViewModel và LiveData: Mẫu + Đi ngược lại mẫu

Lộ trình

Thực hành những gì bạn đã học được bằng cách hoàn thành lộ trình này:

Bài học 8: Cấu trúc ứng dụng (lớp giao diện người dùng)