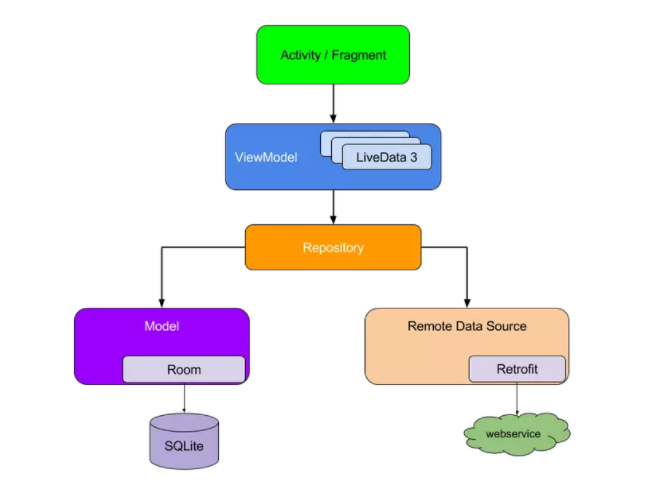
là bộ thư viện giup code đơn giản hơn : MVVM, Architecture component, View Model, Live data, Data binding, Room, Navigation, KTX, Work Manager, Paging, Hilt, Lifecycle Aware

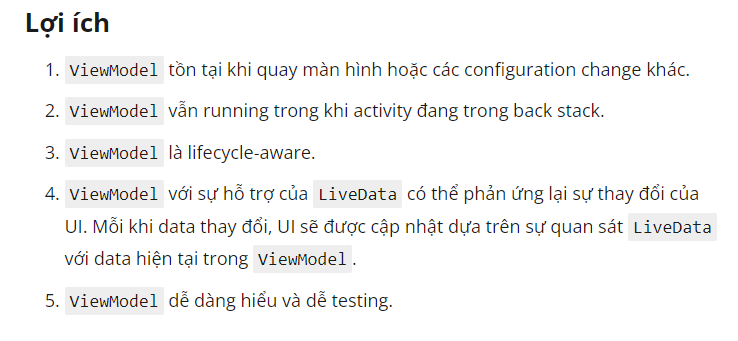
MVVM



* Model: đối tượng
* View: giao diện người dung được cập nhật bởi view model
* View Model: Xử lý logic, sử dụng observer để thông báo sự thay đổi dữ liệu trong view model đến view

View Model

* Quản lý dữ liệu cho 1 UI
* Luôn được tạo cùng phạm vi(activity hoặc fragment), view model sẽ không bị destroy khi activity hoặc fragment bị destroy bởi 1 configuration change(thay đổi cấu hình như kiểu xoay màn hình). Instance mới của các owner sẽ liên kết lại với viewmodel hiện có của nó
* Activity, fragment quan sát sự thay đổi thông qua live data hoặc data binding



View Model Provider

* Khi View Model có tham số thì phải xác định ViewModel Provider.Factory rồi chuyển 1 thể hiện của nó làm tham số thứ 2 của View Model Provider

Live Data

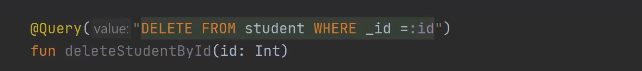
* Tạo 1 instance live data chứa kiểu dữ liệu nhất định
* Là lớp dữ liệu có thể quan sát được, chỉ cập nhật dữ liệu khi activity hoạt động
* Khi data thay đổi nó sẽ thông báo cho observer để cập nhật lên view
* Lợi ích:
* Không bị rò rỉ bộ nhớ(memory leak)
* Không có sự cố khi activity dừng
* Không sử dụng vòng đời thủ công
* Luôn cập nhật dữ liệu

Data Binding

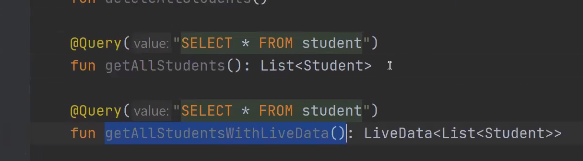
* One way:
* Two way
* Bin adapter: đặt lại tên 1 thuộc tính nào đó

Room database

1. Là 1 abstract layer cung cấp cách thức thao tác với cơ sở dữ liệu SQLite.
2. 3 thành phần: Database, entity, Dao
3. Entity: đại diện cho 1 class chứa 1 cột của database, room sẽ tạo 1 bảng trong database tên là tableName. Nếu ko muốn được lưu trữ thì đánh dấu @IgNore
4. Dao: Thành phần có thể truy cập CSDL

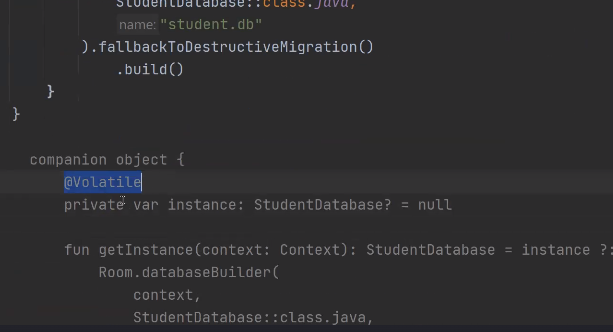




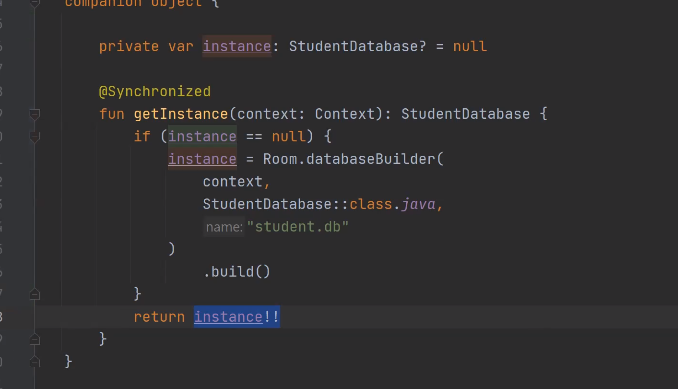


* getAllStudents: phải gọi nhiều lần
* Dùng live data khi có thay đổi chỉ cần thông báo cho observer để cập nhật

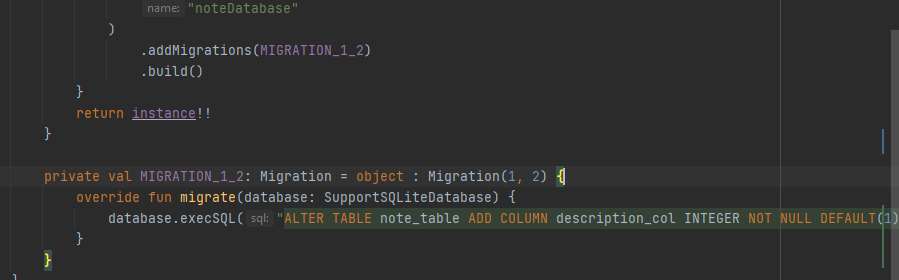
1. Database



* @Volatile: có 2 thread cùng gọi, thằng nào chạy xong trước thì notify cho thằng kia biết là ko tạo nữa, biến instance nhận đc kết quả khi các thread cùng truy cập đến



* @ Synchronized: được gọi trên 1 thread, không gọi được 2 thread cùng 1 lúc, nếu null thì khởi tạo, lần sau trả về instance
* Fallback to destructive migration: cho phép room tạo lại 1 cách triệt để bảng cơ sở dữ liệu nếu migration không tìm thấy CSDL thì sẽ chuyển sang phiên bản mới nhất (khi nâng version)
* Export Schema: nếu true nó import ra file json : cấu trúc các version để theo dõi, version này có cột này, version kia lại them cột kia
* Update version:



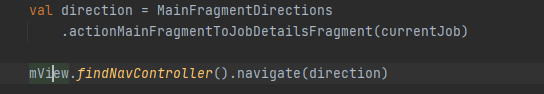
Room insert 1 lần



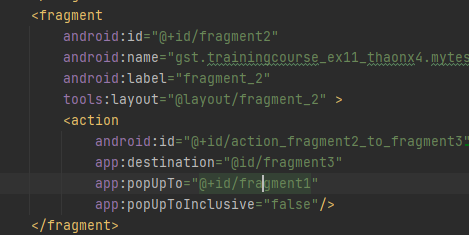
Navigation Components



* App:defaultNavHost: quyết định backTrack hay ko
* Giups đơn giản việc điều hướng trong ứng dụng
* Nav\_graph: Đồ thị kết nối của các Navigation Destination (1 màn hình hoặc 1 vài view)
* Navigation Action: Đường dẫn
* Navigation Host: 1 container để hiển thị 1 màn hình
* Pass data giữa 2 destination: bundle và safe args



* Navigation component có thể kết hợp với bottom navigation
* back về bất kì màn hình nào mjh muốn khi nhấn nút back



KTX

* Là các thành phần kotlin mở rộng giúp code ngắn gọn và sạch
* Tận dụng các tính năng có sẵn của kotlin:
* Extension Functions
* Lambdas
* Named parameters
* Coroutines

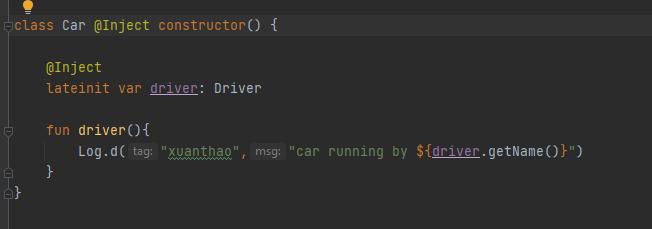
Architecture component

Có 4 thành phần:

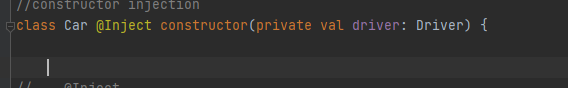
* Room
* Live Data
* Lifecycle
* View Model

Hilt and Di

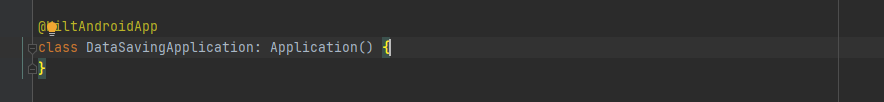
* di : là 1 kĩ thuật trong đó 1 object cung cấp các di cho 1 object khác
* Đơn giản hóa dagger, khả năng mã hóa cao và dễ test
* Tập hợp các thành phần và phạm vị tiêu chuẩn, dễ đọc, dễ thiết lập
* Hạn chế việc tiêm phụ thuộc, tạo 1 lần tái sử dụng được nhiều lần
* Field injection



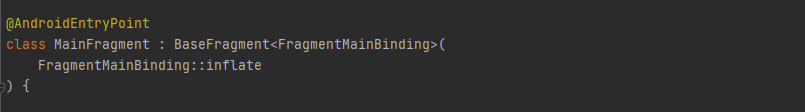
* Constructor Injection



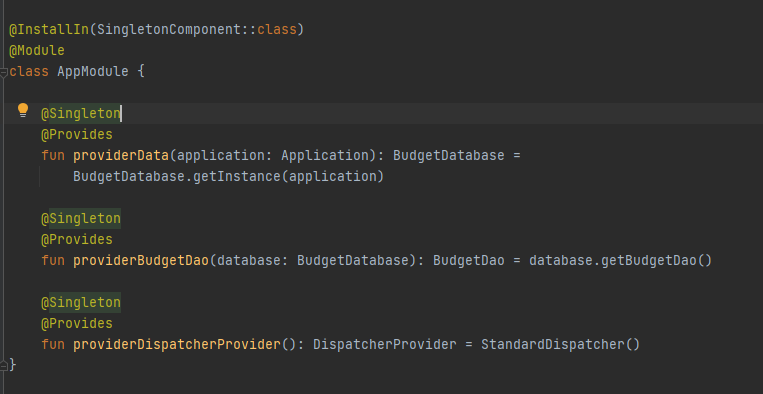
* Application: Định nghĩa các công việc, vùng chứa phụ thuộc cấp ứng dụng



* Hilt cung cấp các phụ thuộc cho các lớp android bằng @AndroidEntryPoint

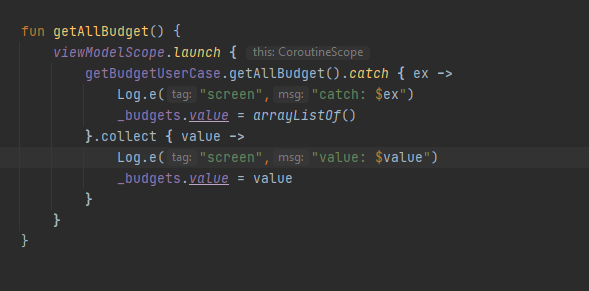


* @InstallIn: nơi mà hilt muốn đưa phụ thuộc này vào
* @Singleton: là 1 scope annotation, mỗi hilt sẽ có 1 scope annotation tương ứng
* @Provides: Kiểu instance mà hilt cung cấp vd như: Budget Database, Budget Dao,…



Flow

* Chỉ chạy khi collect
* Flow có thể chạy bất đồng bộ
* Catch cho phép emit giá trị nguồn thu

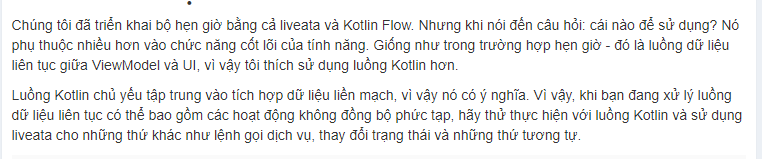


User case

* View model gọn gàng hơn
* có logic data flow tương tự, có thể tái sử dụng ở view model khác
* Implement data flow logic

Live data và Flow

* Live data để quan sát dữ liệu mà không có bất kì phần mềm nào để xử lý vòng đời
* Luồng kotlin được sử dụng để tích hợp dữ liệu liên tục và đơn giản hóa lập trình không đồng bộ



RequireActivity và RequireContext

* fragment năm trong 1 fragment  khác thì require Context sẽ tham chiếu đến fragment cha, nếu fragment năm trong 1 activity thì require Context sẽ tham chiếu đến activity cha còn require Activity tham chiếu đến activity cha

Workmanager

* Quản lý công việc, đảm bảo vẫn đc chạy khi thoát ứng dụng hoặc khởi động lại
* Work Request: yêu cầu công việc, Xác định cách thức và thời điểm bắt đầu công việc, có 2 loại work request:

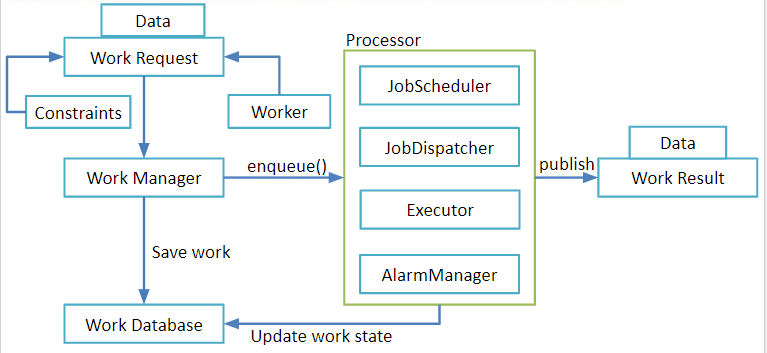
1. One time work request: work request chỉ thực thi 1 lần

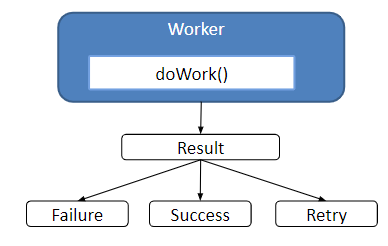
2. Periodic work request: work request lặp lại theo chu kì, sau khi có 1 instance của work request nó sẽ được xếp hàng trong workmanager

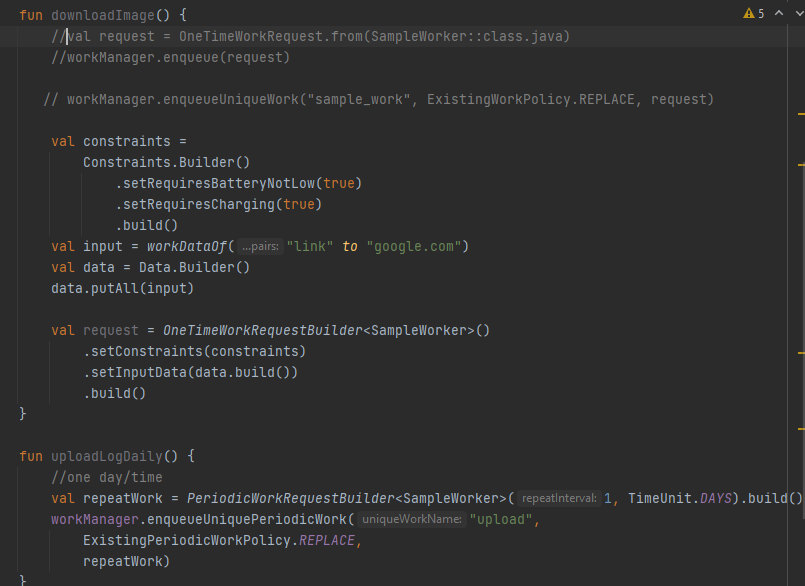
* Worker xác định các bước thực hiện công việc
* Constraint 1 số ràng buộc, thỏa mãn các điều kiện thì mới build
* 1 số loại work constraints:

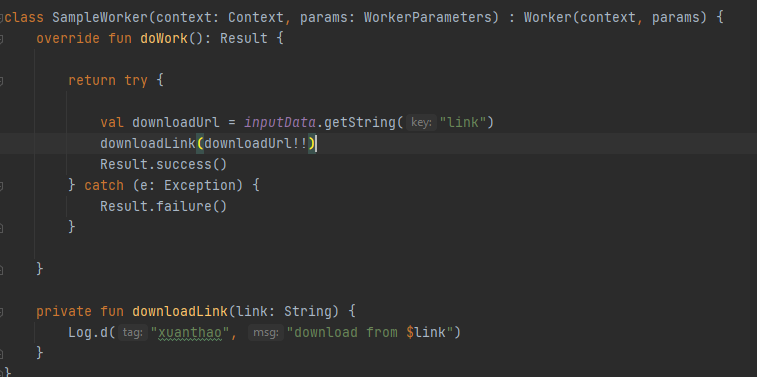
1. requiresStorageNotLow(): yêu câu bộ nhớ ko quá thấp
2. requiresBatteryNotLow() : pin ko quá yếu
3. requiresCharging() : đag sac pin

* processor sẽ đọc worker để xử lý
* work database để lưu trữ công việc
* b1 tao class doWork()





* Retry: thực hiện lại
* B2: Tạo workmanager trong viewmodel
* B3: work constraint:
* B4: quay trở lại class worker inputdata:
* 



* lấy data theo phương thức getKey

Lifecycle\_aware

* Giups tạo ra các đoạn mã có tổ chức hơn, dễ bảo trì
* có 3 lớp chính: Lifecycle, LifeCycleOwner, LifecycleObserver
* Lifecycle: là object có thể truy xuất thông tin vòng đời của activity hay fragment
* LifecycleOwner: Đó là giao diện đánh dấu cho biết rằng giao diện triển khai là LifeCycleObserver. Chúng ta có thể sử dụng chú thích OnLifecycleEvent để xác định hành động tương ứng với sự kiện vòng đời