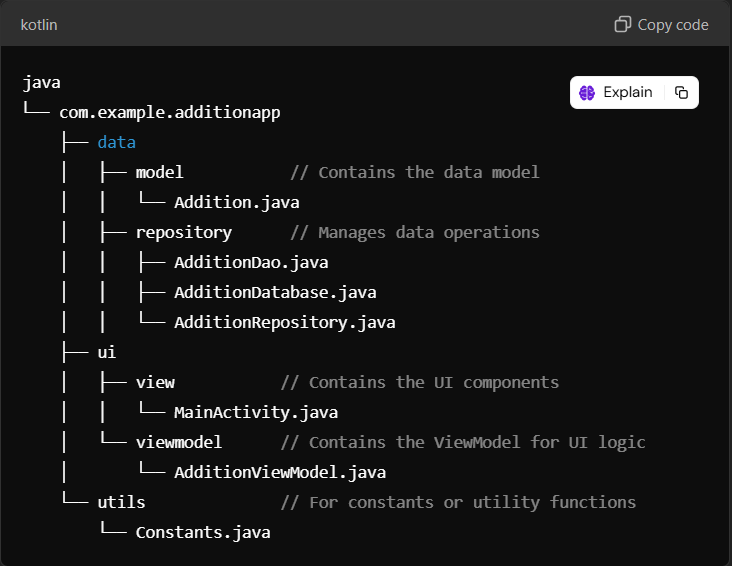
**Data flow of MVVM architecture:**

****

**Explanation:**

The provided code structure represents a typical layout for an Android application named "**AdditionApp**" that performs **addition** operations. Here’s a concise explanation of each part of the structure:

1. **Package Directory (`com.example.additionapp`):**

- This is the root package for the application, following standard Java package naming conventions.

2. **Data Layer (`data`):**

- This part of the application manages **data-related** functionalities.

- **Model (`model`):**

- Contains the data model class.

- \*\***Addition.java**\*\*: This class likely defines the properties and behaviors of the addition operation, such as operands and the result.

- \*\***Repository (`repository`)**\*\*:

- Responsible for handling data operations, possibly from local storage or remote sources.

- \*\***AdditionDao.java**\*\*: This Data Access Object (DAO) interface might define methods for database operations related to additions.

- \*\***AdditionDatabase.java**\*\*: This class probably manages the database setup and configuration.

- \*\***AdditionRepository.java**\*\*: This class acts as a mediator between the data sources (like databases) and the rest of the app, abstracting data operations.

3. **UI Layer (`ui`):**

- This section contains components related to the user interface.

- \*\***View (`view`)**\*\*:

- Contains the main user interface components.

- \*\***MainActivity.java**\*\*: The entry point of the app, where the user interface is initialized and displayed.

- \*\***ViewModel (`viewmodel`)**\*\*:

- Holds UI-related data in a lifecycle-conscious way.

- \*\*AdditionViewModel.java\*\*: This class likely contains the business logic for handling data operations and communicates between the UI and the repository.

4. \*\***Utilities (`utils`)**\*\*:

- This folder is for defining constants or utility functions that can be reused throughout the application.

- \*\***Constants.java**\*\*: Contains constant values used across the app, promoting maintainability and preventing the use of magic numbers or strings.

Overall, this code structure follows the Model-View-ViewModel (MVVM) architectural pattern, promoting separation of concerns, making the app easier to manage and scale.

**Dependencies for adding MVVM:**

dependencies {

implementation(libs.appcompat)

implementation(libs.material)

implementation(libs.activity)

implementation(libs.constraintlayout)

testImplementation(libs.junit)

androidTestImplementation(libs.ext.junit)

androidTestImplementation(libs.espresso.core)

// ViewModel and LiveData

implementation("androidx.lifecycle:lifecycle-viewmodel:2.6.1")

implementation("androidx.lifecycle:lifecycle-livedata:2.6.1")

// Room (for database)

implementation("androidx.room:room-runtime:2.6.0")

annotationProcessor("androidx.room:room-compiler:2.6.0")

// Coroutine support (optional for later)

implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.4")

implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4")

}