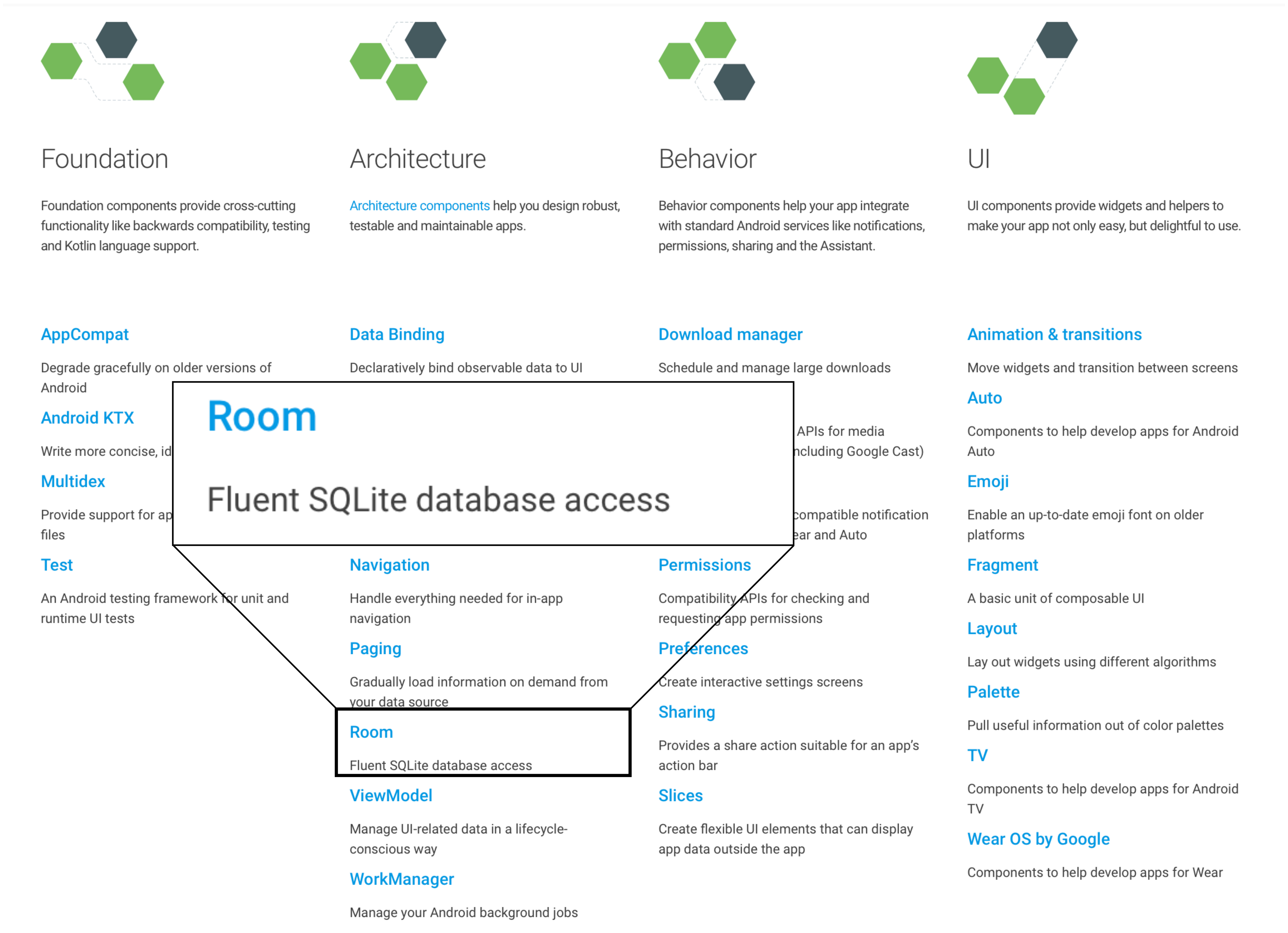
Opening doors to Room

Exploring Room Persistence Library

Annsh Singh

SO **Delhi**

Android Jetpack Components



Foundation

Foundation components provide cross-cutting functionality like backwards compatibility, testing and Kotlin language support.

AppCompat

Degrade gracefully on older versions of Android

Android KTX

Write more concise, id

Multidex

Provide support for ap
files

Test

An Android testing framework for unit and runtime UI tests

Architecture

Architecture components help you design robust, testable and maintainable apps.

Data Binding

Declaratively bind observable data to UI

Room

Fluent SQLite database access

Navigation

Handle everything needed for in-app navigation

Paging

Gradually load information on demand from your data source

Room

Fluent SQLite database access

ViewModel

Manage UI-related data in a lifecycle-conscious way

WorkManager

Manage your Android background jobs

Behavior

Behavior components help your app integrate with standard Android services like notifications, permissions, sharing and the Assistant.

Download manager

Schedule and manage large downloads

APIs for media
ncluding Google Cast)

compatible notification
ear and Auto

Permissions

Compatibility APIs for checking and requesting app permissions

Preferences

Create interactive settings screens

Sharing

Provides a share action suitable for an app's action bar

Slices

Create flexible UI elements that can display app data outside the app

UI

UI components provide widgets and helpers to make your app not only easy, but delightful to use.

Animation & transitions

Move widgets and transition between screens

Auto

Components to help develop apps for Android Auto

Emoji

Enable an up-to-date emoji font on older platforms

Fragment

A basic unit of composable UI

Layout

Lay out widgets using different algorithms

Palette

Pull useful information out of color palettes

TV

Components to help develop apps for Android TV

Wear OS by Google

Components to help develop apps for Wear

Outline

What?

Why?

How?

What?

- Room is a persistence library provides an abstraction layer over SQLite to allow **fluent database access** while harnessing the full power of SQLite.
- It’s basically a **wrapper above SQLite.**

Why?

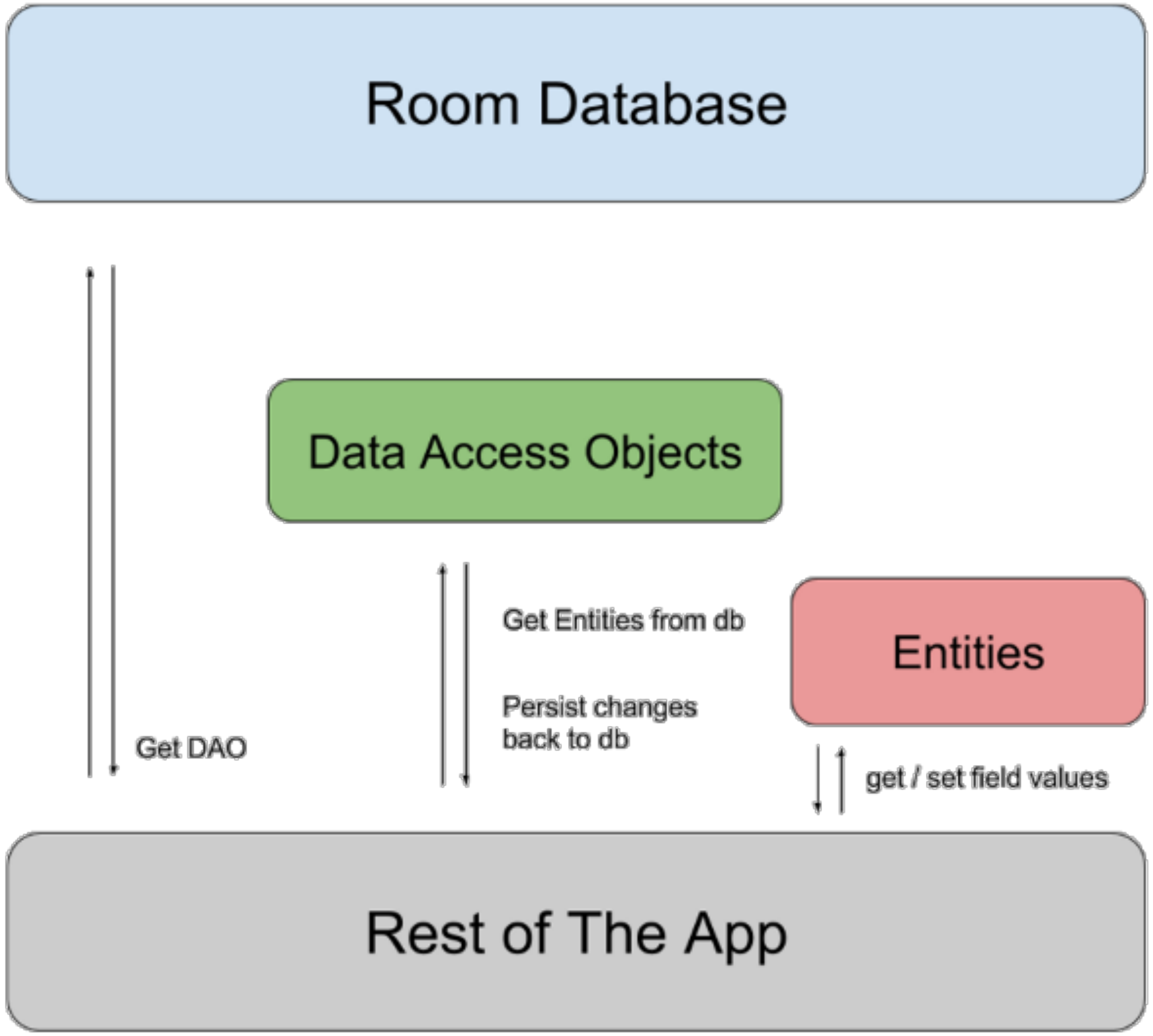
- Offers **compile time check** - each @Query and @Entity is checked at the compile time, so there’s no risk of runtime error that might crash your app (and it doesn’t check only syntax, but also e.g. missing tables)
- Works really well with **LiveData** (live monitoring)
- Decrease the amount of **boilerplate code**

How?

- Add the required Room dependencies in **build.grade** file

```
implementation "android.arch.persistence.room:runtime:2.1.0-alpha04"
kapt "android.arch.persistence.room:compiler:2.1.0-alpha04"
```

- Understanding Room components



Room Components

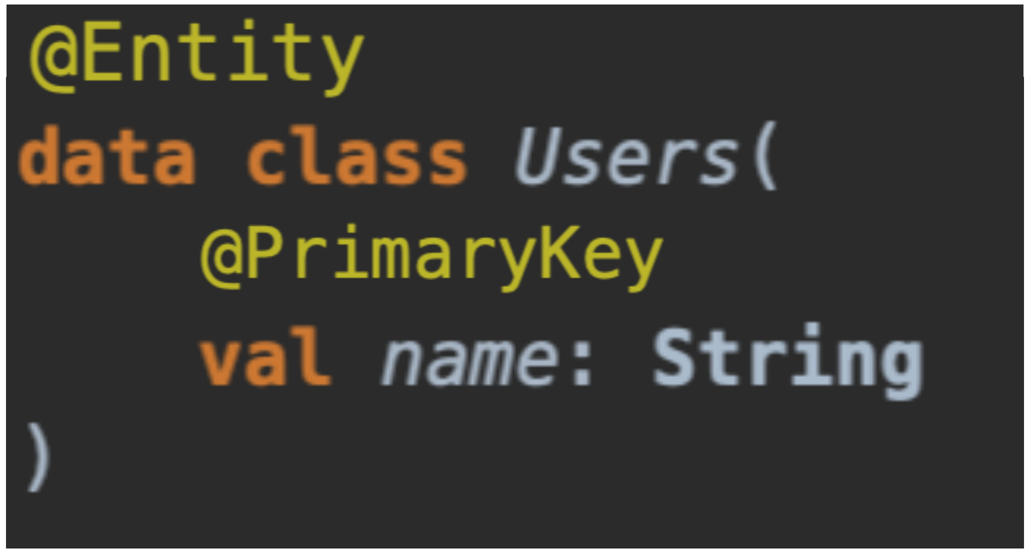
1. Entity

A Java or a Kotlin class which represents a table within the database.

For **each entity** you create, **a table** is created with the associated Database.

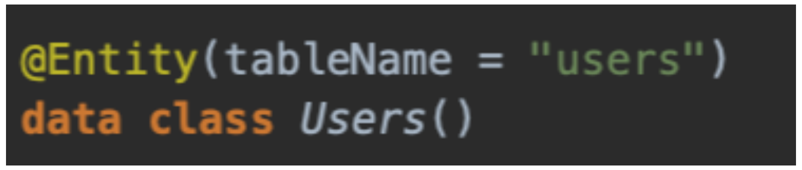
By default, Room creates a **column for each field.**

How to create it?



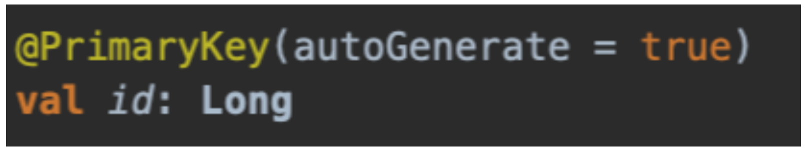
```kotlin
@Entity
data class Users(
    @PrimaryKey
    val name: String
)
```

Entity Annotations



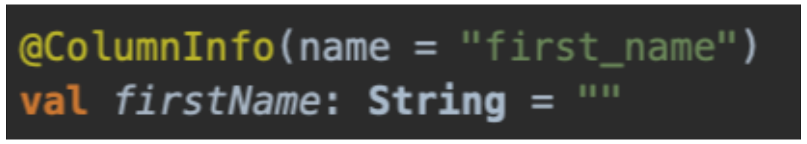
```
@Entity(tableName = "users")
data class Users()
```

Specify the name of the table if you want it to be different from the name of the class



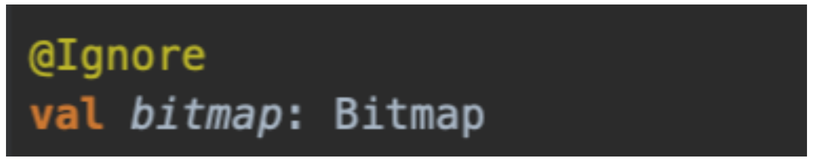
```
@PrimaryKey(autoGenerate = true)
val id: Long
```

Every entity needs a primary key. Allow the database to auto increment using **autoGenerate = true**



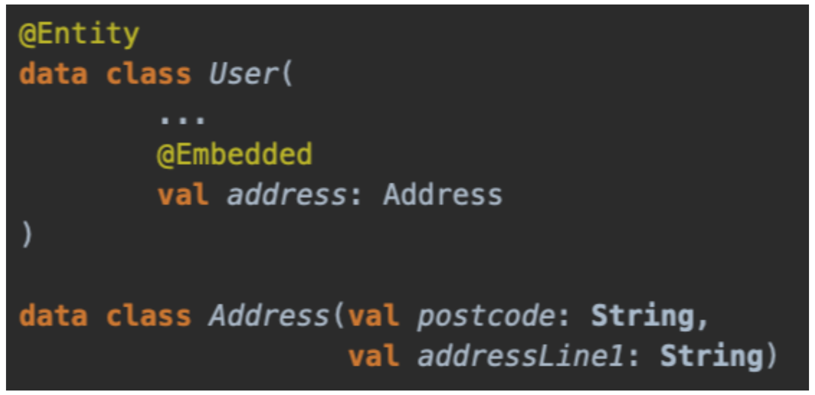
```
@ColumnInfo(name = "first_name")
val firstName: String = ""
```

Specify the name of the column in the table if you want it to be different from the name of the member variable.



```
@Ignore
val bitmap: Bitmap
```

If we have something in our Pojo that doesn't need to go into the database, just add this annotation.



```
@Entity
data class User(
        ...
        @Embedded
        val address: Address
)

data class Address(val postcode: String,
                   val addressLine1: String)
```

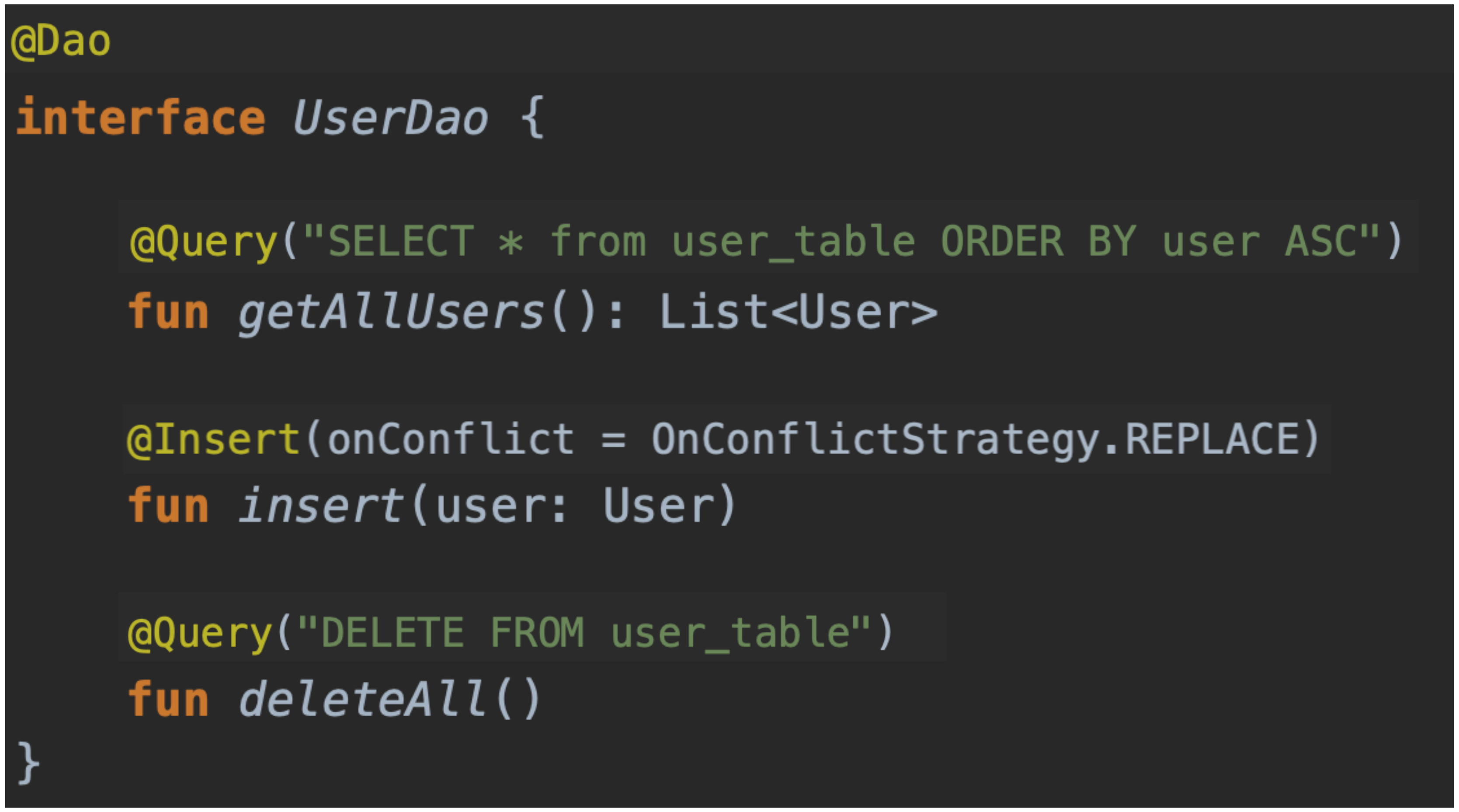
If we embed an Address object, it will save as fields, but map back to an Address object. You'll be able to query parts of the address.

Room Components

2. DAO - Data Access Object

Here you specify SQL queries and associate them with method calls.
The compiler checks the SQL and generates queries from convenience **annotations** for **common queries, such as @Insert.**
Room creates each DAO implementation at **compile** time.

How to create it?



```
@Dao
interface UserDao {

    @Query("SELECT * from user_table ORDER BY user ASC")
    fun getAllUsers(): List<User>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(user: User)

    @Query("DELETE FROM user_table")
    fun deleteAll()
}
```

Observing Changes

How to get automatic updates whenever the data changes to make sure your UI reflects the latest values from your database?

LiveData

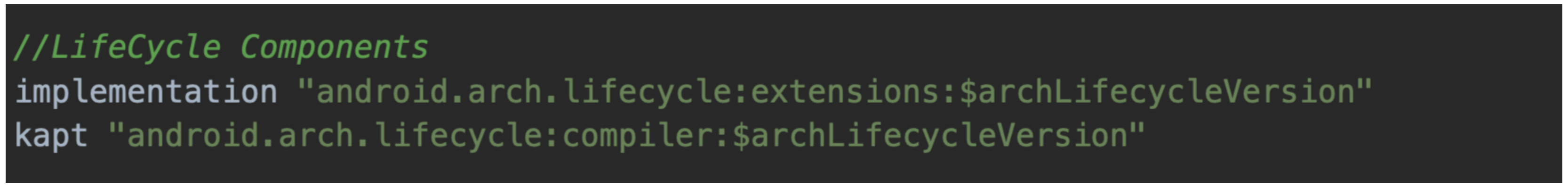
RxJava

Coroutines

LiveData

- It is an observable **data holder** class
- **Lifecycle Aware** - It respects the lifecycle of other app components, such as activities, fragments, or services.
- **No memory leaks** - Bound to Lifecycle objects and clean up after themselves when their associated lifecycle is destroyed
- **No crashes due to stopped activities** - Does’t receive updates when activity is in back stack.
- **Always up to date data** - an activity that was in the background receives the latest data right after it returns to the foreground.
- **Proper configuration changes** - If an activity or fragment is recreated due to a configuration change, like device rotation, it immediately receives the latest available data.

- Add the required Lifecycle dependencies in **build.grade** file

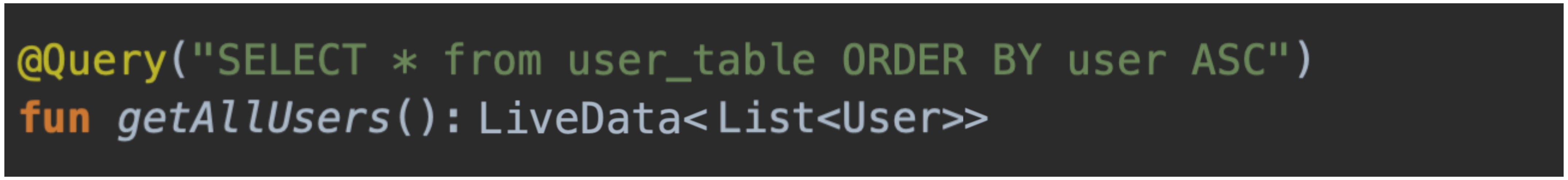


```
//LifeCycle Components
implementation "android.arch.lifecycle:extensions:$archLifecycleVersion"
kapt "android.arch.lifecycle:compiler:$archLifecycleVersion"
```

- Implementation with Room -

In the **UserDao** we saw earlier, change the **getAllUsers()** method signature so that the returned List<User> is **wrapped with LiveData**.

How?

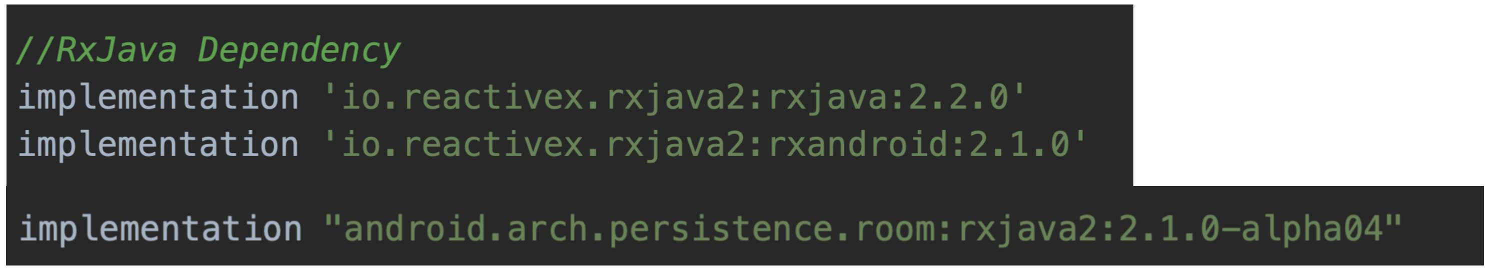


```
@Query("SELECT * from user_table ORDER BY user ASC")
fun getAllUsers(): LiveData<List<User>>
```

RxJava

- Starting with Room 2.1.0-alpha01, DAO methods annotated with **@Insert, @Delete or @Update** support Rx return types **Completable, Single<T> and Maybe<T>**
- Return types for **@Insert** -
  - **Completable** - where *onComplete* is called as soon as the insertion was done
  - **Single<Long>** or **Maybe<Long>** - where the value emitted on *onSuccess* is the row id of the item inserted
  - **Single<List<Long>>** or **Maybe<List<Long>>** - where the value emitted on *onSuccess* is the list of row ids of the items inserted
  - In case of error inserting the data, *Completable*, *Single* and *Maybe* will emit the exception in **onError**.
- Return types for **@Update/Delete** -
  - **Completable** - where *onComplete* is called as soon as the update/delete was done.
  - **Single<Long>** or **Maybe<Long>** - where the value emitted on *onSuccess* is the number of rows affected by update/delete
- To ensure that observable queries are done **off the main thread** - Use the **observeOn** operator to specify the *Scheduler* on which an *Observer* will observe the *Observable* and **subscribeOn** to specify the *Scheduler* on which the *Observable* will operate

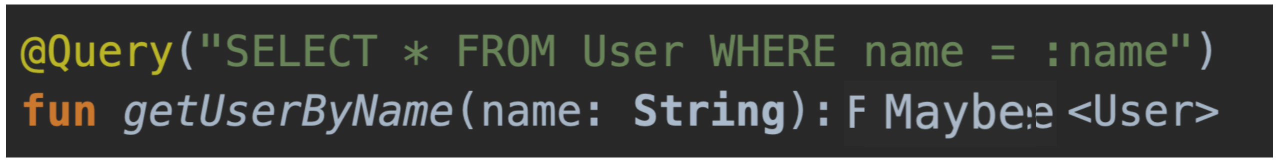
- Add the required dependencies for RxJava and its support for Room in **build.grade** file



```
//RxJava Dependency
implementation 'io.reactivex.rxjava2:rxjava:2.2.0'
implementation 'io.reactivex.rxjava2:rxandroid:2.1.0'

implementation "android.arch.persistence.room:rxjava2:2.1.0-alpha04"
```

- Return types for **@Query** -



```
@Query("SELECT * FROM User WHERE name = :name")
fun getUserByName(name: String): F Maybee <User>
```

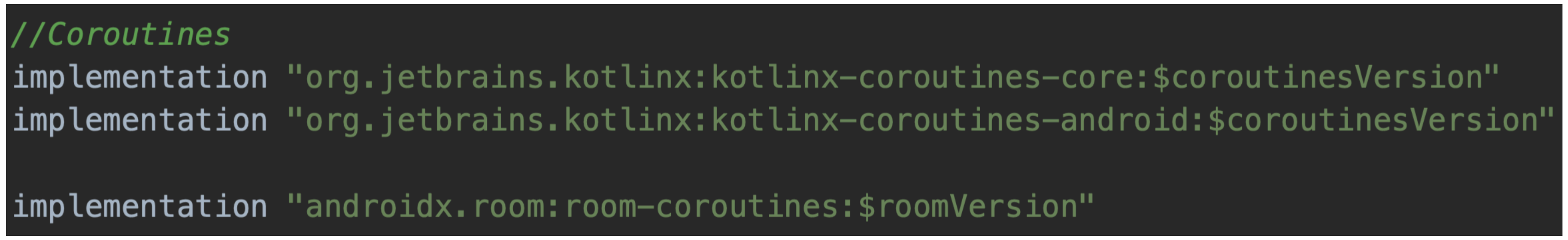
- **Maybe** -
    - When there is **no user in the database** and the query returns no rows, *Maybe* will **complete**.
    - When there is a user in the database, *Maybe* will trigger *onSuccess* and it will **complete**.
    - If the **user is updated** after *Maybe* was completed, **nothing happens.**
- **Single** -
    - When there is no user in the database and the query returns no rows, *Single* will trigger **onError(EmptyResultSetException.class)**
    - When there is a user in the database, *Single* will trigger *onSuccess*.
    - If the **user is updated** after *Single* was completed, **nothing happens.**
- **Flowable** -
    - When there is **no user in the database** and the query returns no rows, the **Flowable will not emit**, neither *onNext*, nor *onError*
    - When there is a user in the database, the *Flowable* will trigger *onNext*.
    - Every time the **user data is updated**, the *Flowable* object will **emit automatically**, allowing you to update the UI based on the latest data.

Coroutines

(Room integration still in development)

- A new way of managing **background threads** that can simplify code by reducing the need for callbacks.
- Coroutines are a Kotlin feature that convert async callbacks for long-running tasks, such as database or network access, into ***sequential*** code.
- They wait until a result is available from a long-running task and continue execution.
- **Suspend** modifier -
  - Kotlin's way of marking a function, or function type, available to coroutines
  - Instead of blocking until that function returns like a normal function call, it **suspends** execution until the result is ready then it **resumes** where it left off with the result.
- **Coroutine Scope** -
  - In Kotlin, all coroutines run inside a CoroutineScope.
  - A scope controls the **lifetime of coroutines** through its job - When you cancel the job of a scope, it cancels all coroutines started in that scope.
  - On **Android**, you can use a scope to cancel all running coroutines when, for example, the user navigates away from an Activity or Fragment.
  - Scopes also allow you to **specify a default dispatcher** - A dispatcher controls which **thread** runs a coroutine.

- Add the required dependencies for Coroutines and its support for Room in **build.grade** file

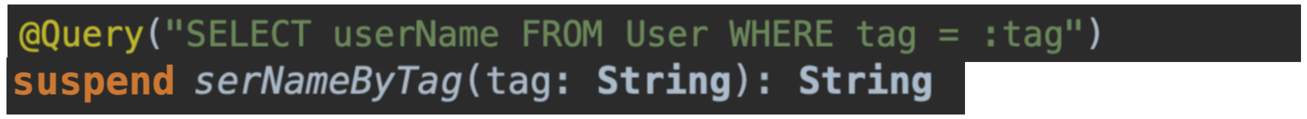


```
//Coroutines
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:$coroutinesVersion"
implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:$coroutinesVersion"

implementation "androidx.room:room-coroutines:$roomVersion"
```

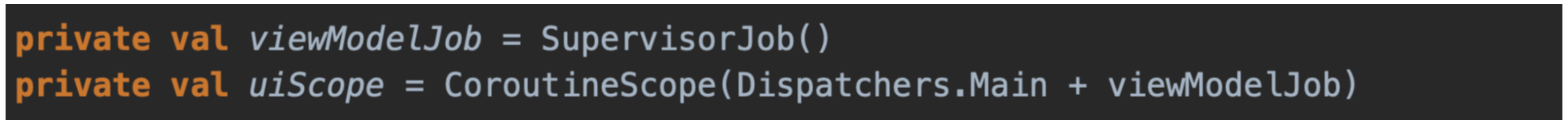
- How it works -

UserDao

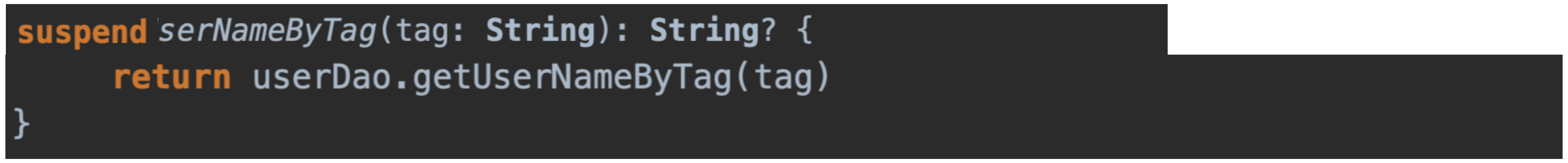


```
@Query("SELECT userName FROM User WHERE tag = :tag")
suspend serNameByTag(tag: String): String
```

ViewModel

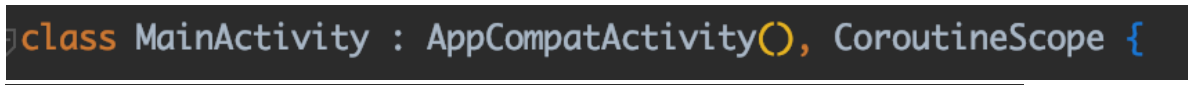


```
private val viewModelJob = SupervisorJob()
private val uiScope = CoroutineScope(Dispatchers.Main + viewModelJob)
```

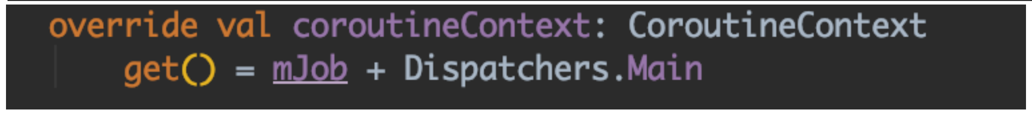


```
suspend serNameByTag(tag: String): String? {
    return userDao.getUserNameByTag(tag)
}
```

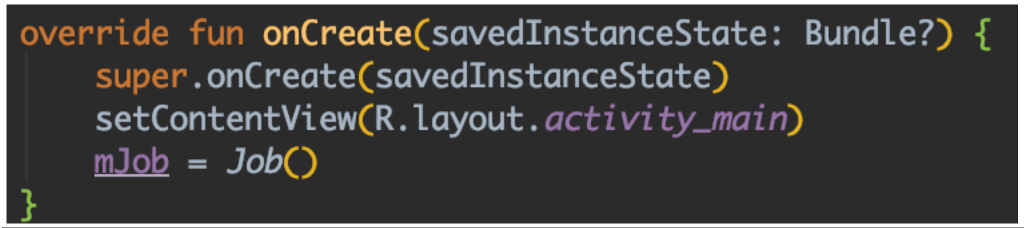
Activity (View)



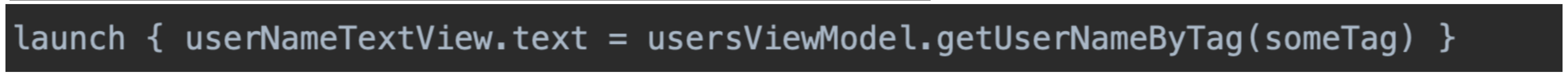
```
class MainActivity : AppCompatActivity(), CoroutineScope {
```



```
  override val coroutineContext: CoroutineContext
      get() = mJob + Dispatchers.Main
```

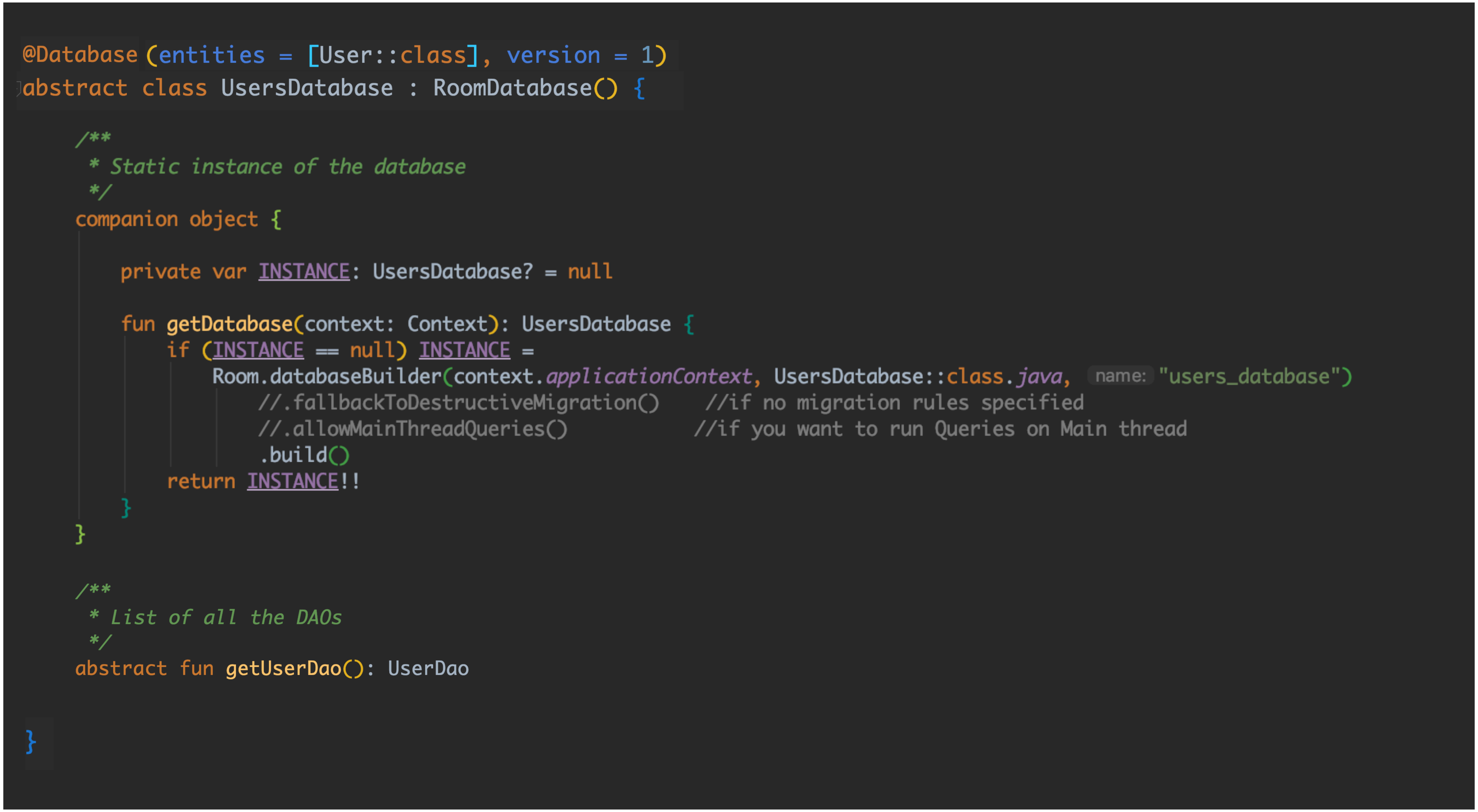


```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    mJob = Job()
}
```



```
launch { userNameTextView.text = usersViewModel.getUserNameByTag(someTag) }
```

Creating Database



```
@Database (entities = [User::class], version = 1)
abstract class UsersDatabase : RoomDatabase() {

    /**
     * Static instance of the database
     */
    companion object {

        private var INSTANCE: UsersDatabase? = null

        fun getDatabase(context: Context): UsersDatabase {
            if (INSTANCE == null) INSTANCE =
                Room.databaseBuilder(context.applicationContext, UsersDatabase::class.java,  name: "users_database")
                    //.fallbackToDestructiveMigration()    //if no migration rules specified
                    //.allowMainThreadQueries()           //if you want to run Queries on Main thread
                    .build()
            return INSTANCE!!
        }
    }

    /**
     * List of all the DAOs
     */
    abstract fun getUserDao(): UserDao

}
```

More Resources

- Video on Room by **Yigit** at **Google I/O ’17** - (Link)
- Video on Room by **Yigit** and **Daniel** at **Android Dev Summit '18** - (Link)
- Medium posts by **Florina Muntenescu** (Link) to learn more about Room
- Video on Coroutines by **Venkat Subramaniam** - (Link)
- Medium posts by **Roman Elizarov** (Link) to learn more about coroutines.
- Interesting post on coroutines by **Dmytro Danylyk** - (Link)
- Subscribe to **Android Developers** Youtube channel to keep yourself upto date with the latest and greatest in Android - (Link)

Any Questions?

Get in touch -

Annsh Singh

@annsh2013

annsh29@gmail.com

https://www.linkedin.com/in/annsh/

https://github.com/annshsingh