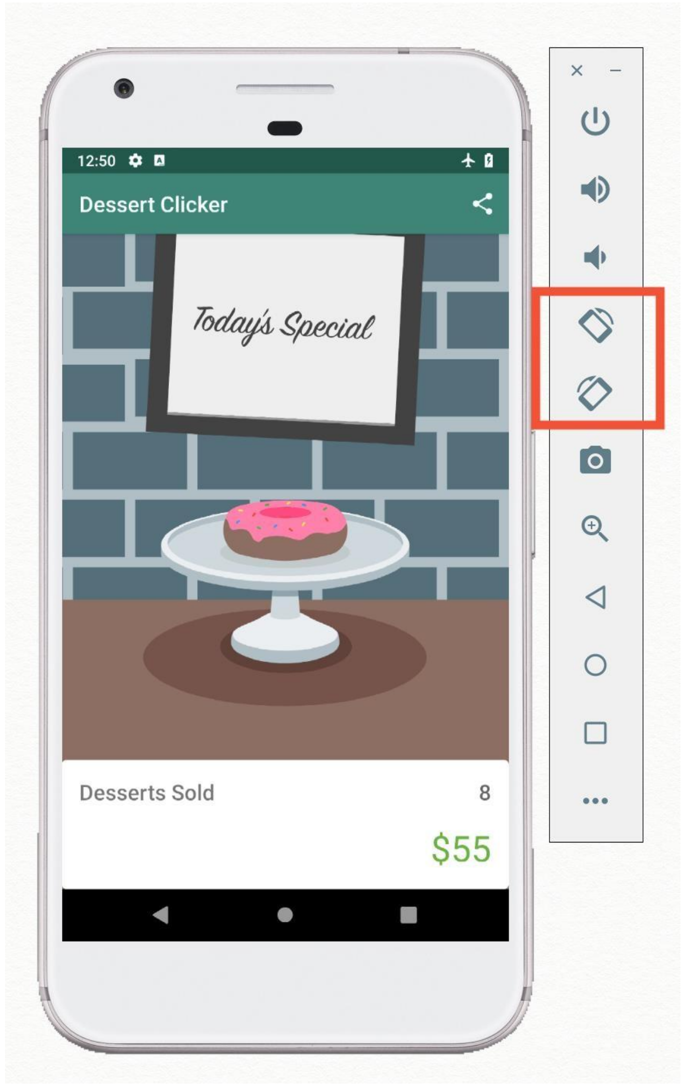
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Mobilné výpočty

Ing. Maroš Čavojský, PhD.

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Rotácie zariadenia



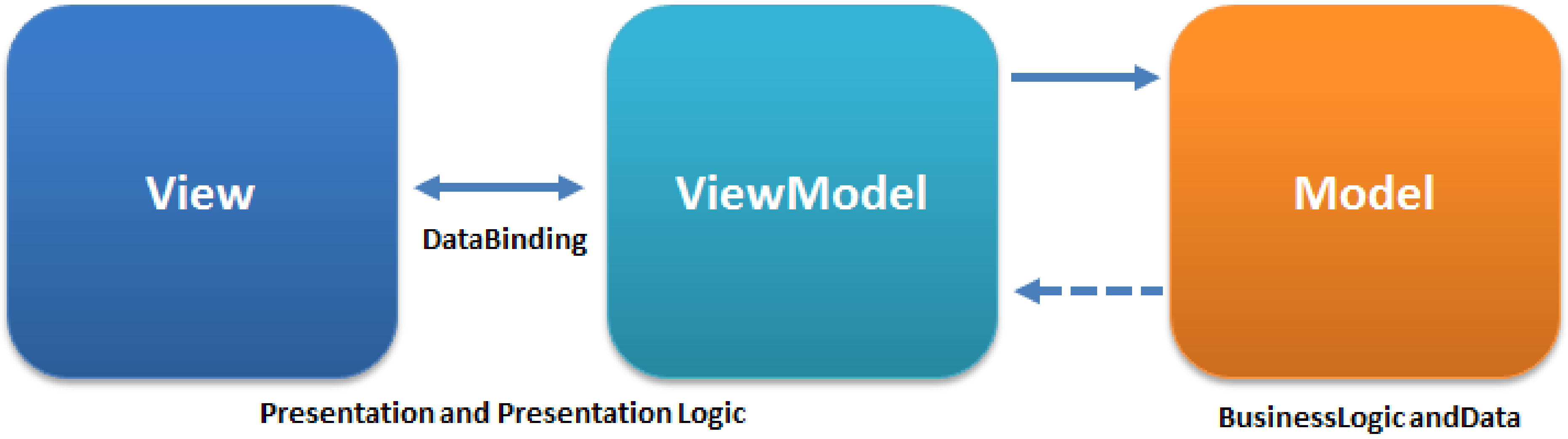
Riešenie

- Bundle
- Shared Preferences
- SQLite Databáza
- Zápis do súboru

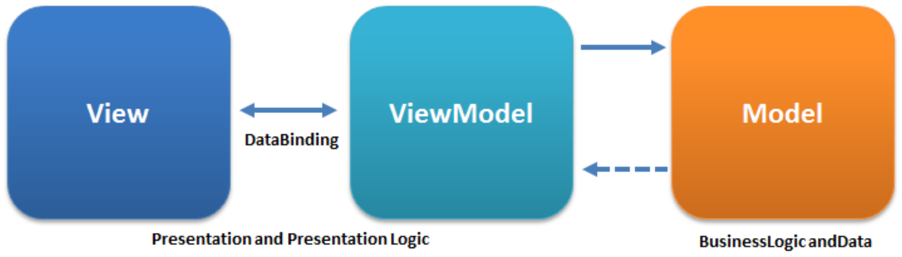
Riešenie

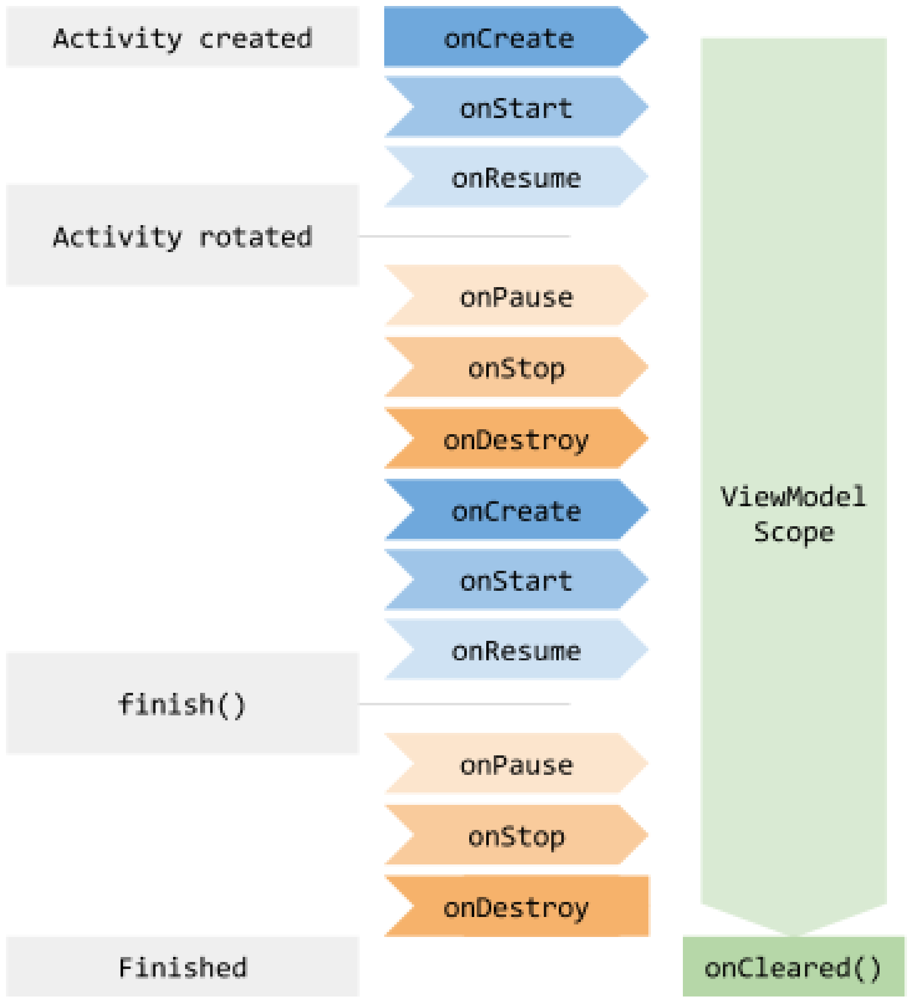
- Bundle
- Shared Preferences
- SQLite Databáza
- Zápis do súboru

Model-view-viewmodel

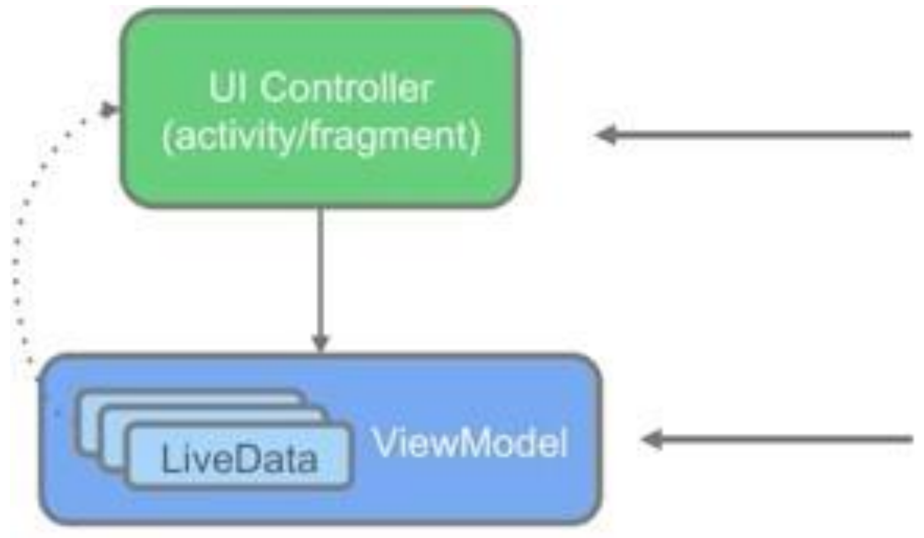


ViewModel


View
DataBinding
ViewModel
Model
Presentation and Presentation Logic
BusinessLogic andData


Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish()
onPause
onStop
onDestroy
Finished
ViewModel Scope
onCleared()

Zodpovednosť



Display data and capture OS and user events

Hold all of the data needed for the UI and prepare it for display

LiveData

- Zabezpečia, že UI bude zodpovedať aktuálnym dátam - Observer
- Žiadne úniky pamäte (Memory leaks) - Lifecycle
- Žiadne pády aplikácie, keď je aplikácia minimalizovaná
- Žiadne ošetrovanie životného cyklu – LiveData sú už Lifecycle aware
- Vždy aktuálne – po obnovení aktivity sa údaje obnovia
- Správne ošetrenie pri zmene konfigurácii
- Zdieľanie dát

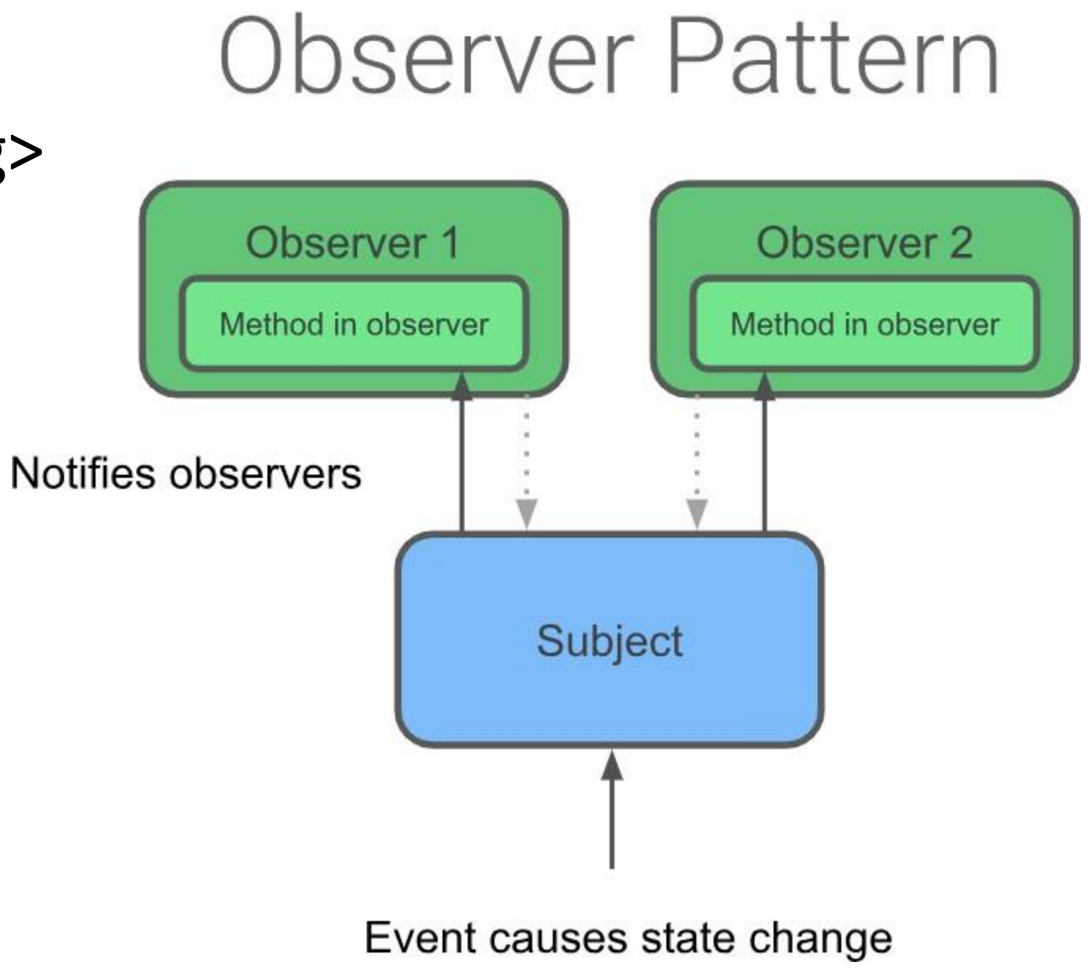
Obsah vo ViewModel

- Dáta a ich načítavanie a zapisovanie
- LiveData … String … LiveData<String>
- MutableLiveData<String>

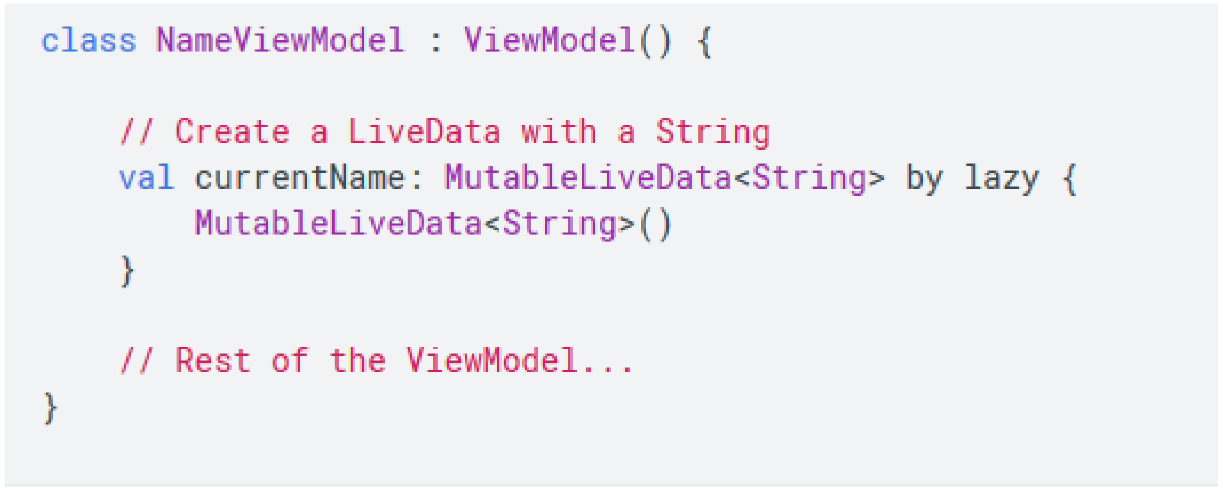
Obsah vo ViewModel

- Dáta a ich načítavanie a zapisovanie
- LiveData … String … LiveData<String>
- MutableLiveData<String>

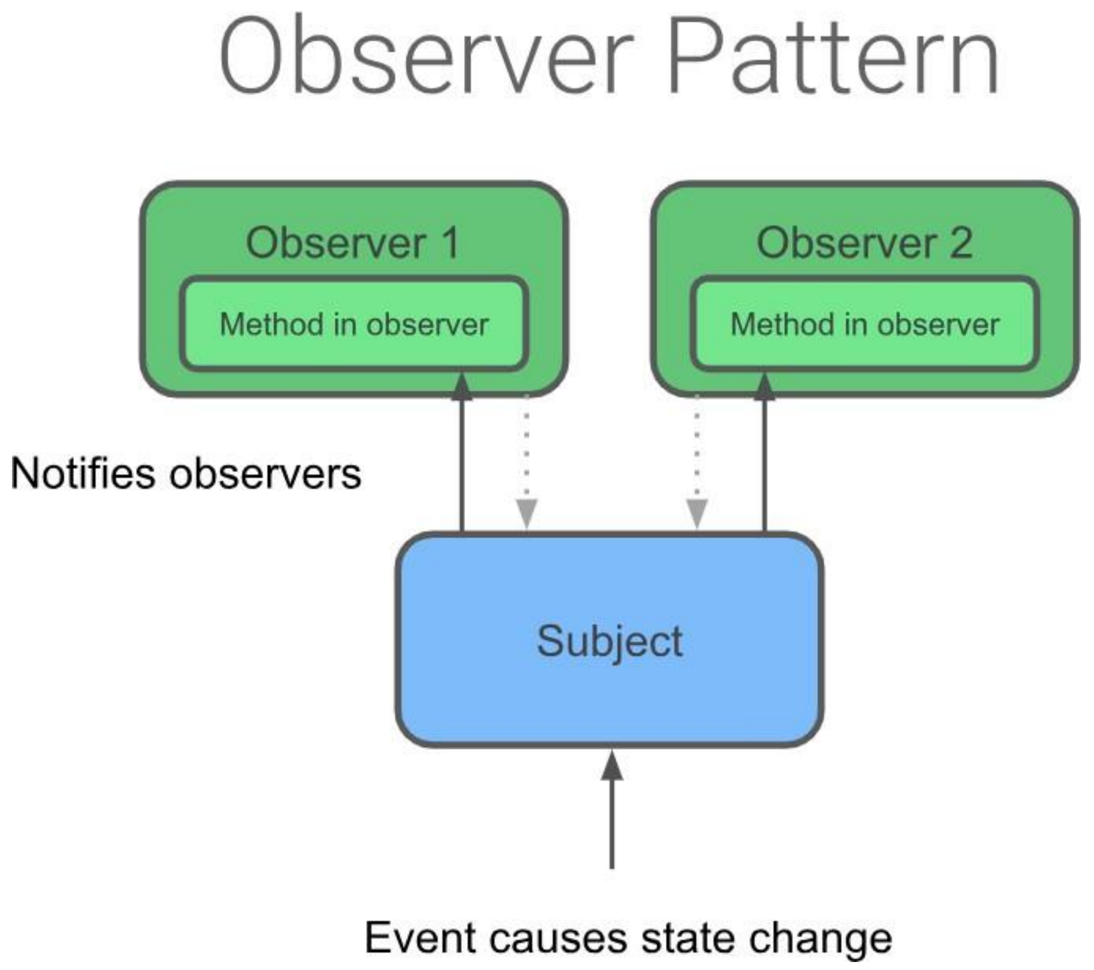
- setValue() - v hlavnom vlákne
- postValue() – v pracovnom vlákne



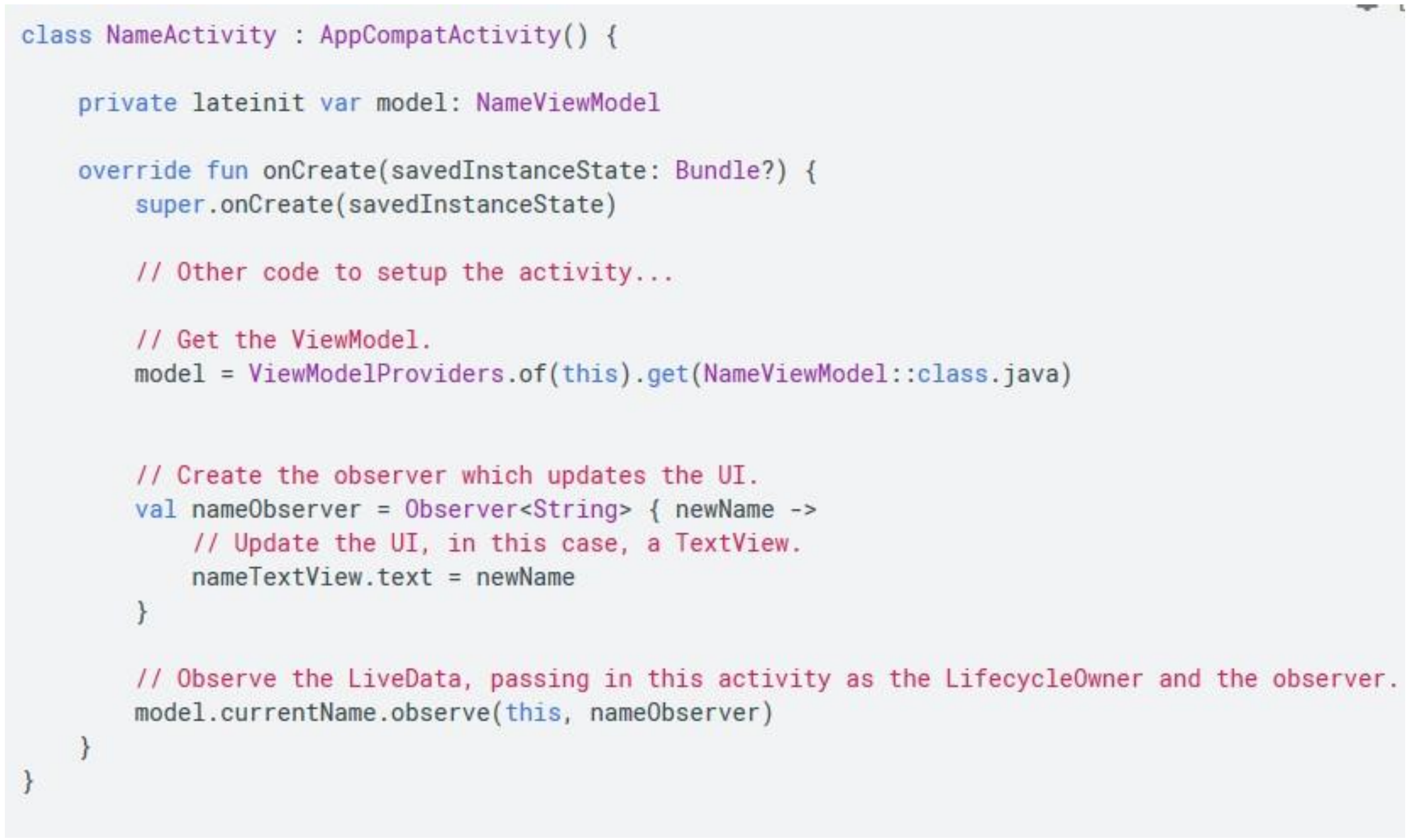
Obsah vo ViewModel



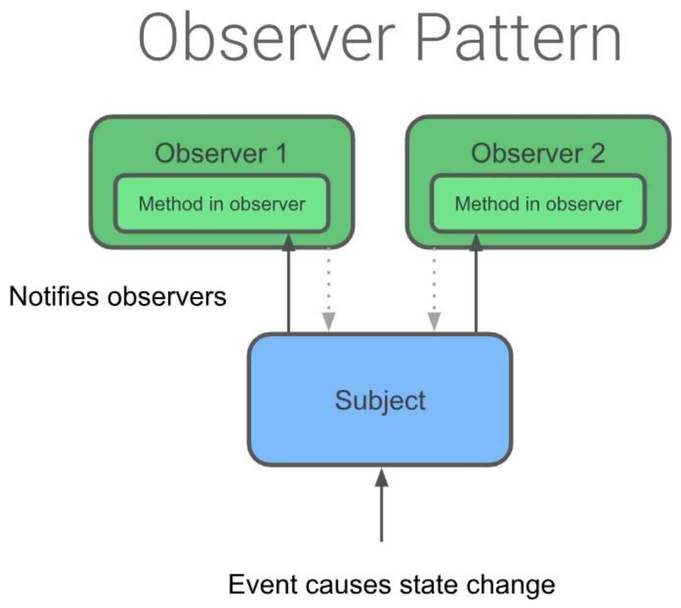
```
class NameViewModel : ViewModel() {

    // Create a LiveData with a String
    val currentName: MutableLiveData<String> by lazy {
        MutableLiveData<String>()
    }

    // Rest of the ViewModel...
}
```



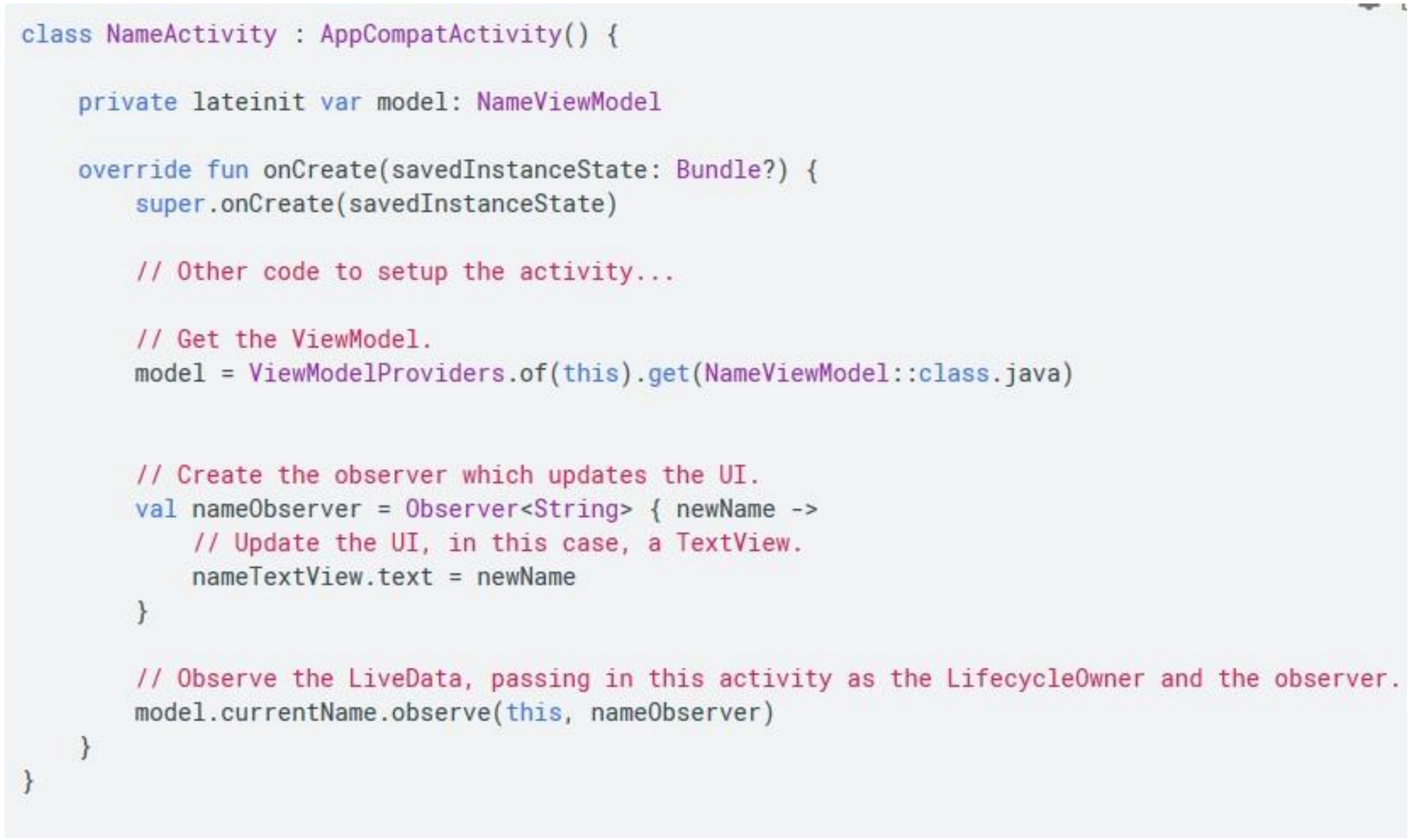
Obsah vo ViewModel



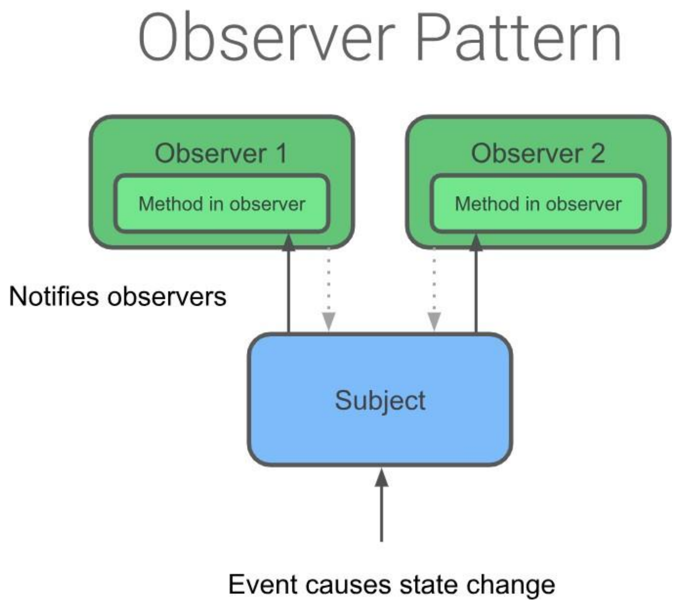
```kotlin
class NameActivity : AppCompatActivity() {

    private lateinit var model: NameViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Other code to setup the activity...

        // Get the ViewModel.
        model = ViewModelProviders.of(this).get(NameViewModel::class.java)


        // Create the observer which updates the UI.
        val nameObserver = Observer<String> { newName ->
            // Update the UI, in this case, a TextView.
            nameTextView.text = newName
        }

        // Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
        model.currentName.observe(this, nameObserver)
    }
}
```



Obsah vo ViewModel



```
class NameActivity : AppCompatActivity() {

    private lateinit var model: NameViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Other code to setup the activity...

        // Get the ViewModel.
        model = ViewModelProviders.of(this).get(NameViewModel::class.java)


        // Create the observer which updates the UI.
        val nameObserver = Observer<String> { newName ->
            // Update the UI, in this case, a TextView.
            nameTextView.text = newName
        }

        // Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
        model.currentName.observe(this, nameObserver)
    }
}
```

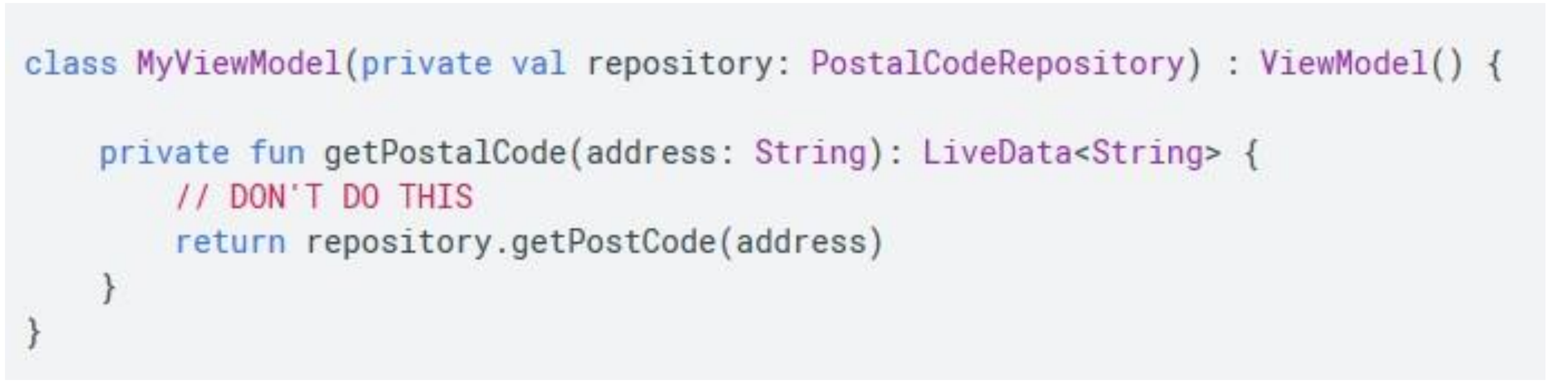


Transformácie

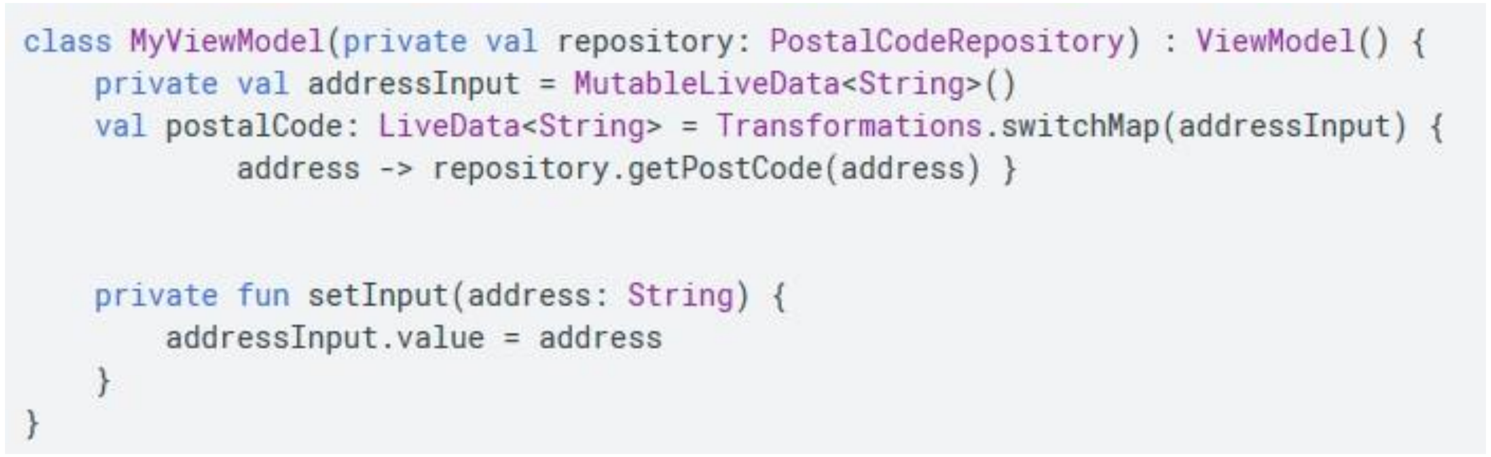
```
val userLiveData: LiveData<User> = UserLiveData()
val userName: LiveData<String> = Transformations.map(userLiveData) {
    user -> "${user.name} ${user.lastName}"
}
```

```
private fun getUser(id: String): LiveData<User> {
  ...
}
val userId: LiveData<String> = ...
val user = Transformations.switchMap(userId) { id -> getUser(id) }
```

Transformácie

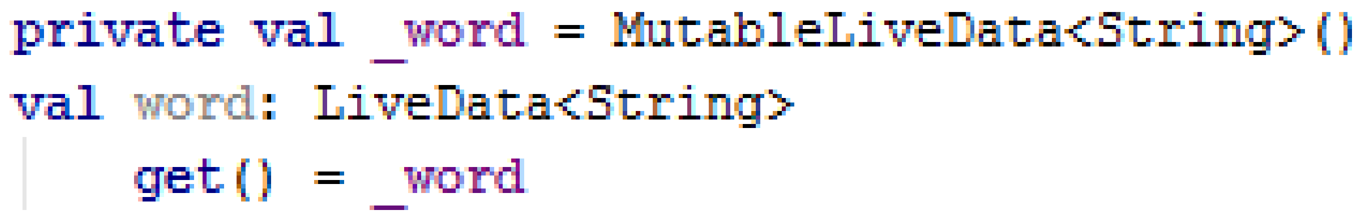


```kotlin
class MyViewModel(private val repository: PostalCodeRepository) : ViewModel() {

    private fun getPostalCode(address: String): LiveData<String> {
        // DON'T DO THIS
        return repository.getPostCode(address)
    }
}
```



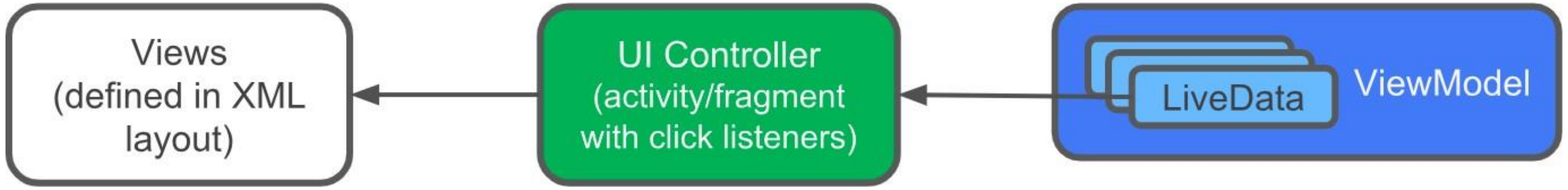
```kotlin
class MyViewModel(private val repository: PostalCodeRepository) : ViewModel() {
    private val addressInput = MutableLiveData<String>()
    val postalCode: LiveData<String> = Transformations.switchMap(addressInput) {
            address -> repository.getPostCode(address) }


    private fun setInput(address: String) {
        addressInput.value = address
    }
}
```

Enkapsulácia (zabalenie) LiveData



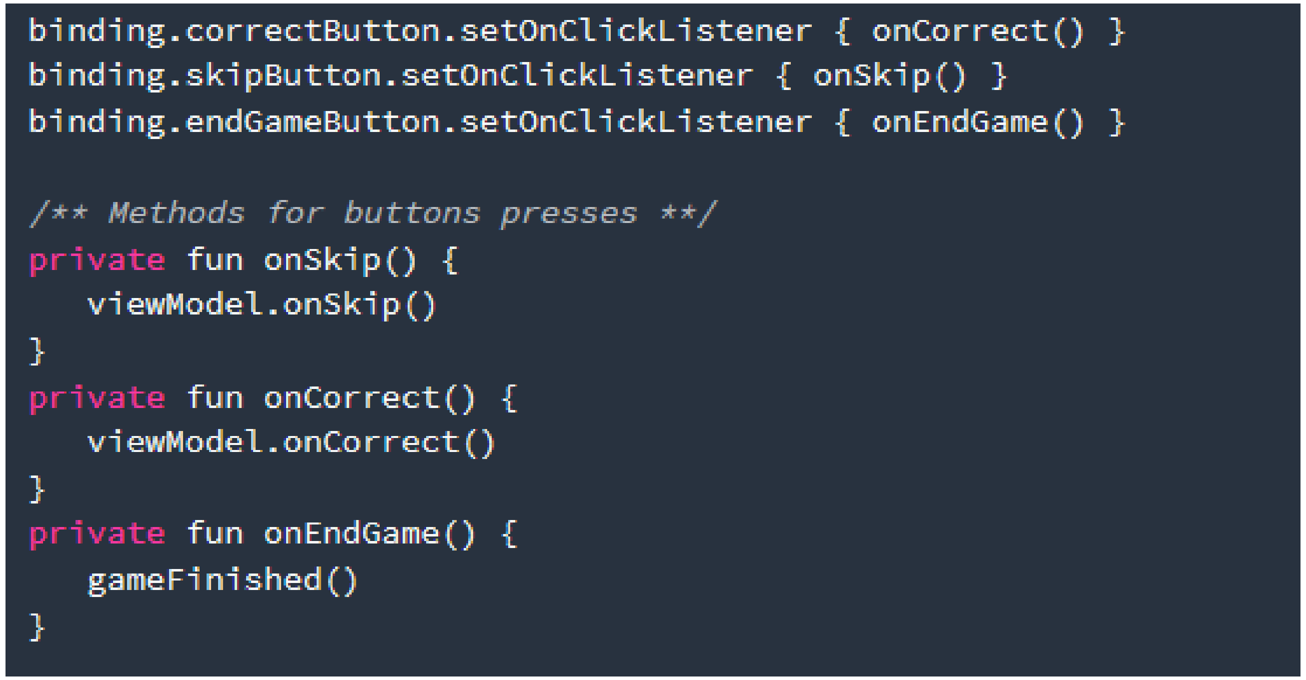
```kotlin
private val _word = MutableLiveData<String>()
val word: LiveData<String>
    get() = _word
```

ViewModel data binding



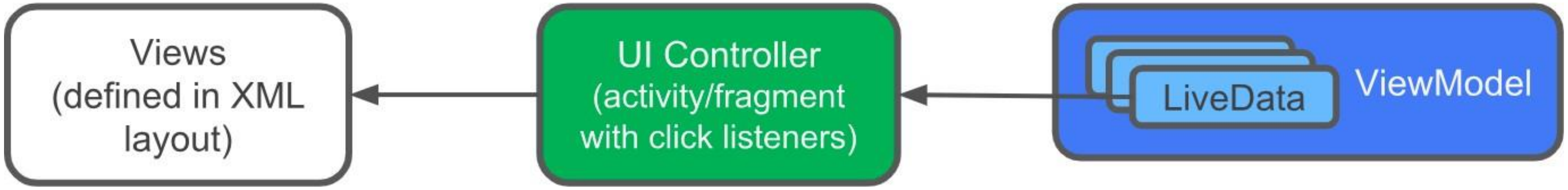
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE FAKULTA ELEKTROTECHNIKY A INFORMATIKY

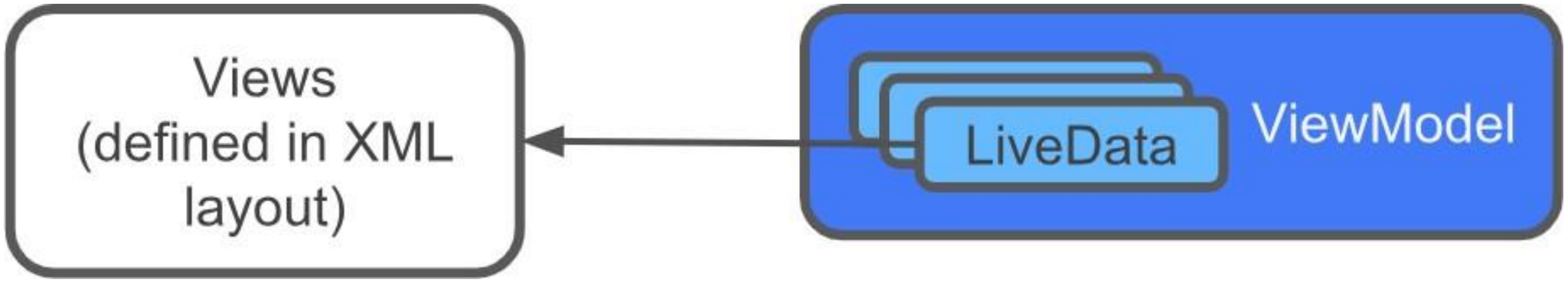
ViewModel data binding



```
binding.correctButton.setOnClickListener { onCorrect() }
binding.skipButton.setOnClickListener { onSkip() }
binding.endGameButton.setOnClickListener { onEndGame() }

/** Methods for buttons presses **/
private fun onSkip() {
   viewModel.onSkip()
}
private fun onCorrect() {
   viewModel.onCorrect()
}
private fun onEndGame() {
   gameFinished()
}
```

ViewModel data binding





ViewModel data binding

```xml
<layout ...>

   <data>

       <variable
           name="gameViewModel"
           type="com.example.android.guesstheword.screens.game.GameViewModel" />
   </data>
```

```kotlin
// Set the viewmodel for databinding - this allows the bound layout access
// to all the data in the ViewModel
binding.gameViewModel = viewModel
```

```xml
<Button
   android:id="@+id/skip_button"
   ...
   android:onClick="@{() -> gameViewModel.onSkip()}"
   ... />
```

ViewModel data binding

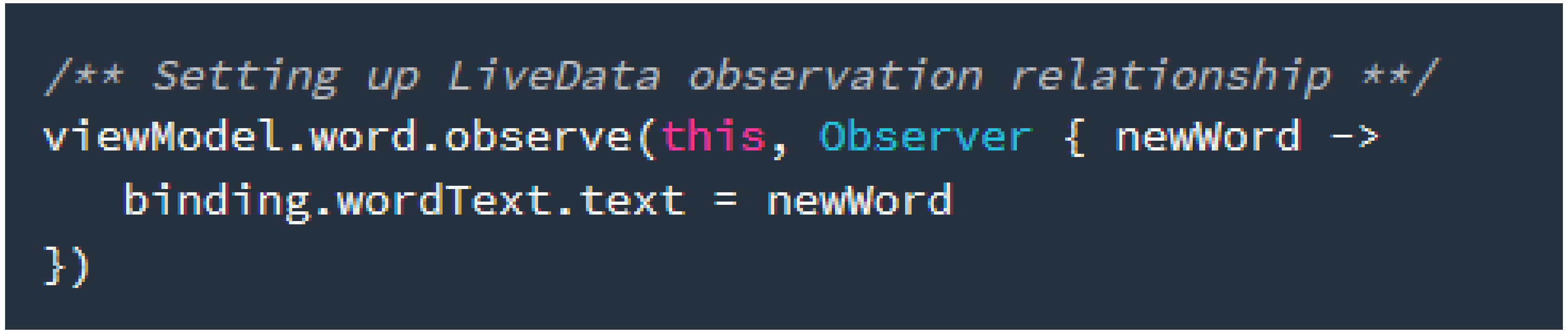
```
<layout ...>

   <data>

       <variable
           name="gameViewModel"
           type="com.example.android.guesstheword.screens.game.GameViewModel" />
   </data>
```

```
<TextView
    android:id="@+id/word_text"
    ...
    android:text="@{gameViewModel.word}"
    ... />
```

```
<TextView
    android:id="@+id/score_text"
    ...
    android:text="@{String.valueOf(scoreViewModel.score)}"
    ... />
```

```
binding.gameViewModel = ...
// Specify the current activity as the lifecycle owner of the binding.
// This is used so that the binding can observe LiveData updates
binding.lifecycleOwner = this
```

ViewModel data binding

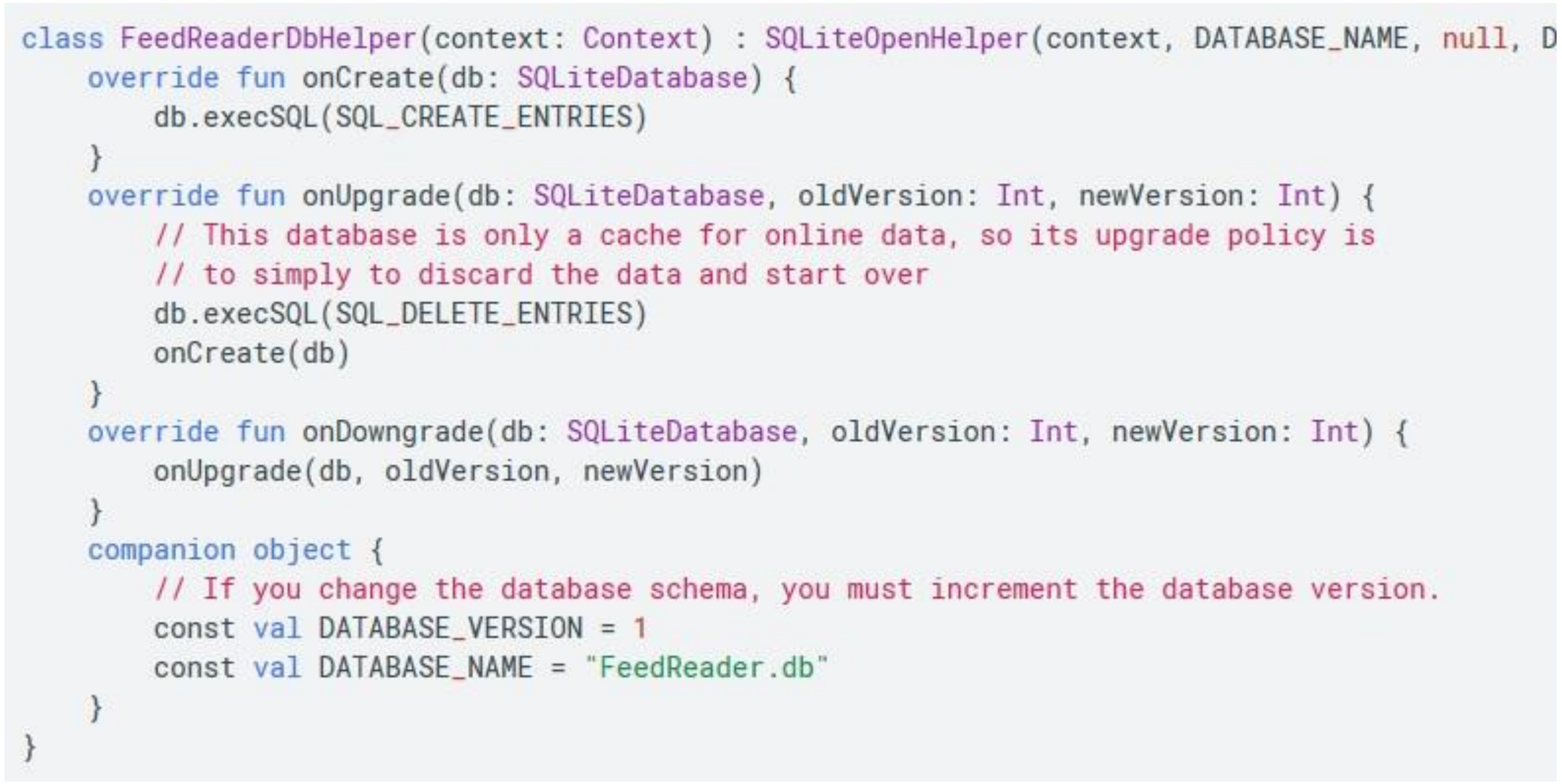


```
/** Setting up LiveData observation relationship **/
viewModel.word.observe(this, Observer { newWord ->
    binding.wordText.text = newWord
})
```

Ukladanie dát

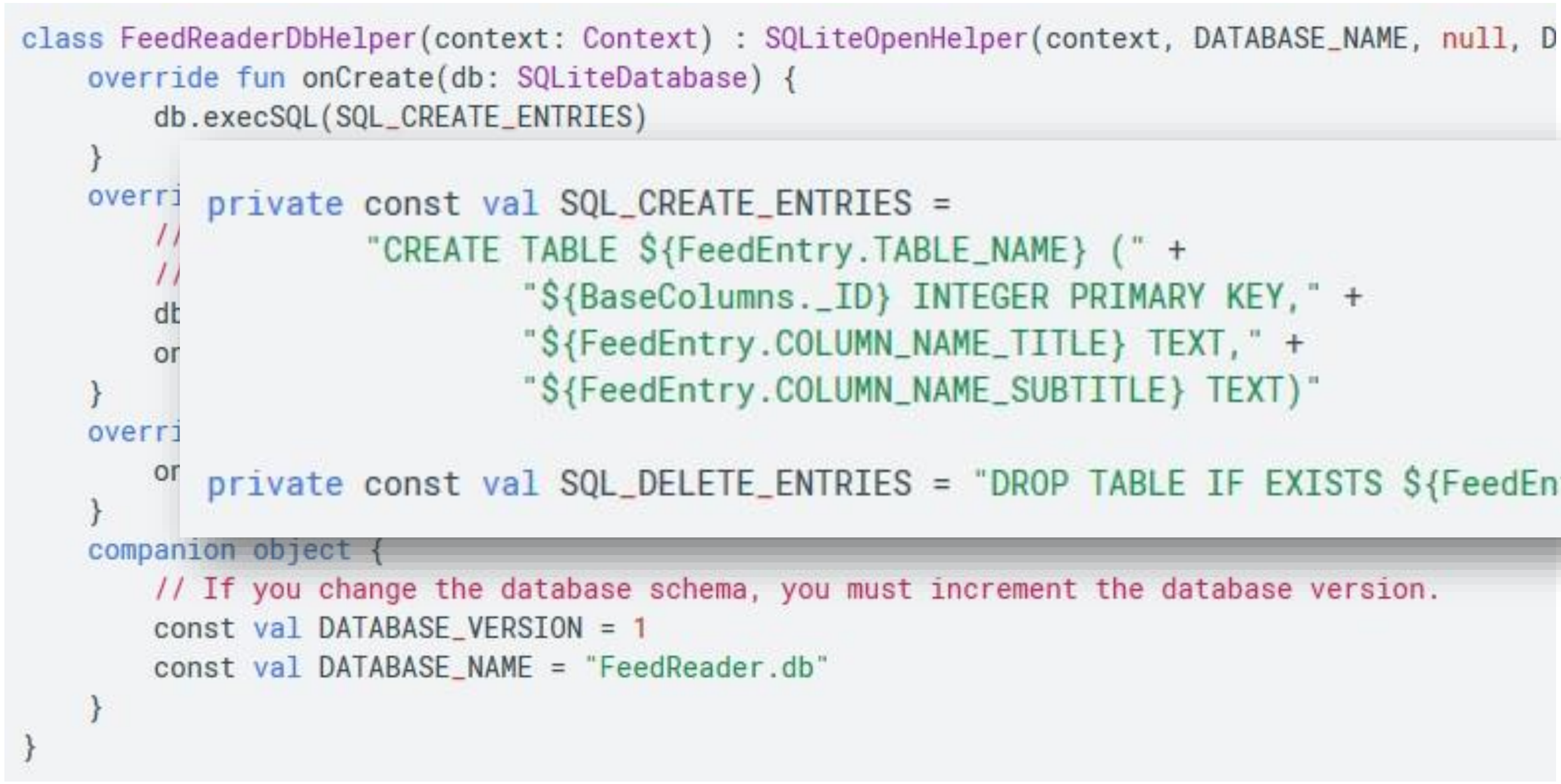
- Shared Preferences
- SQLite Databáza
- Zápis do súboru

SQLite databáza

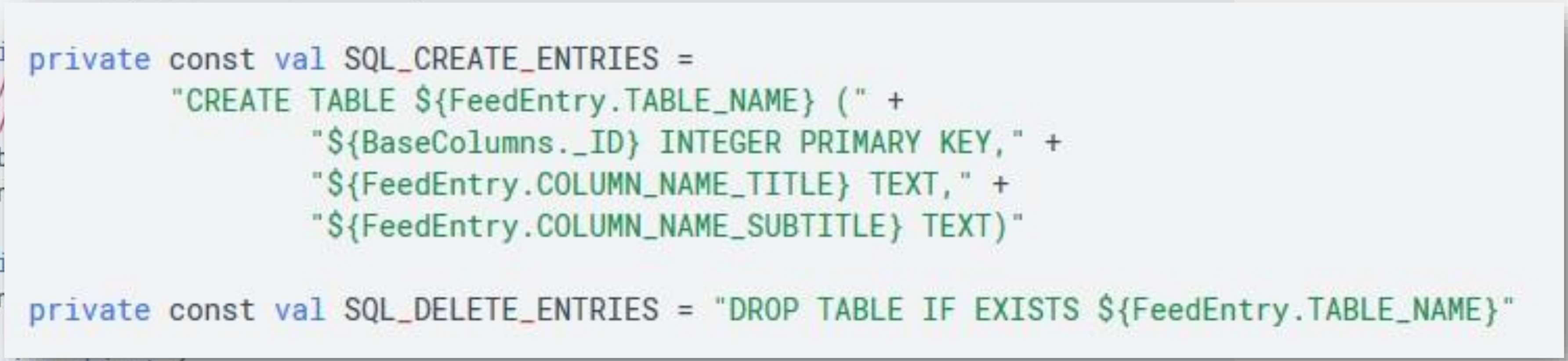


```kotlin
class FeedReaderDbHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, D
    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(SQL_CREATE_ENTRIES)
    }
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES)
        onCreate(db)
    }
    override fun onDowngrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        onUpgrade(db, oldVersion, newVersion)
    }
    companion object {
        // If you change the database schema, you must increment the database version.
        const val DATABASE_VERSION = 1
        const val DATABASE_NAME = "FeedReader.db"
    }
}
```

SQLite databáza

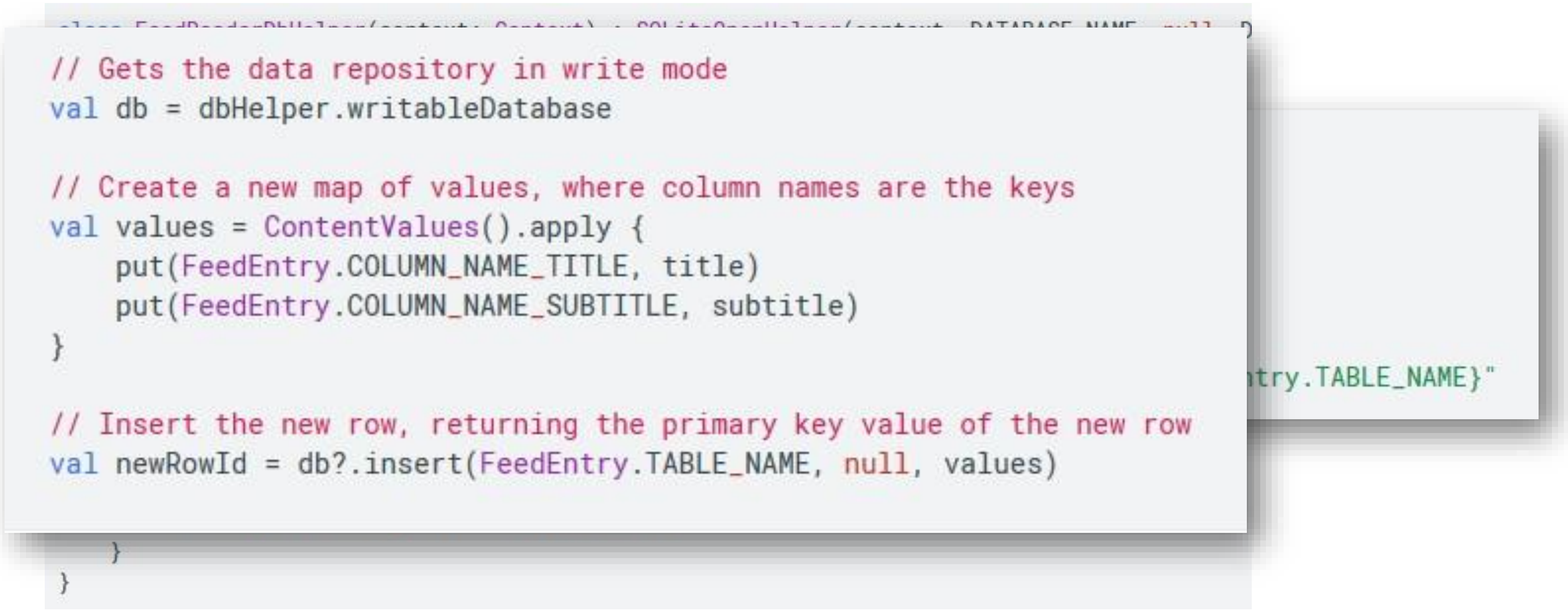


```
class FeedReaderDbHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, D
    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(SQL_CREATE_ENTRIES)
    }
    overri
        //
        //
        db
        or
    }
    overri
        or
    }
    companion object {
        // If you change the database schema, you must increment the database version.
        const val DATABASE_VERSION = 1
        const val DATABASE_NAME = "FeedReader.db"
    }
}
```



```
private const val SQL_CREATE_ENTRIES =
        "CREATE TABLE ${FeedEntry.TABLE_NAME} (" +
                "${BaseColumns._ID} INTEGER PRIMARY KEY," +
                "${FeedEntry.COLUMN_NAME_TITLE} TEXT," +
                "${FeedEntry.COLUMN_NAME_SUBTITLE} TEXT)"

private const val SQL_DELETE_ENTRIES = "DROP TABLE IF EXISTS ${FeedEntry.TABLE_NAME}"
```

SQLite databáza



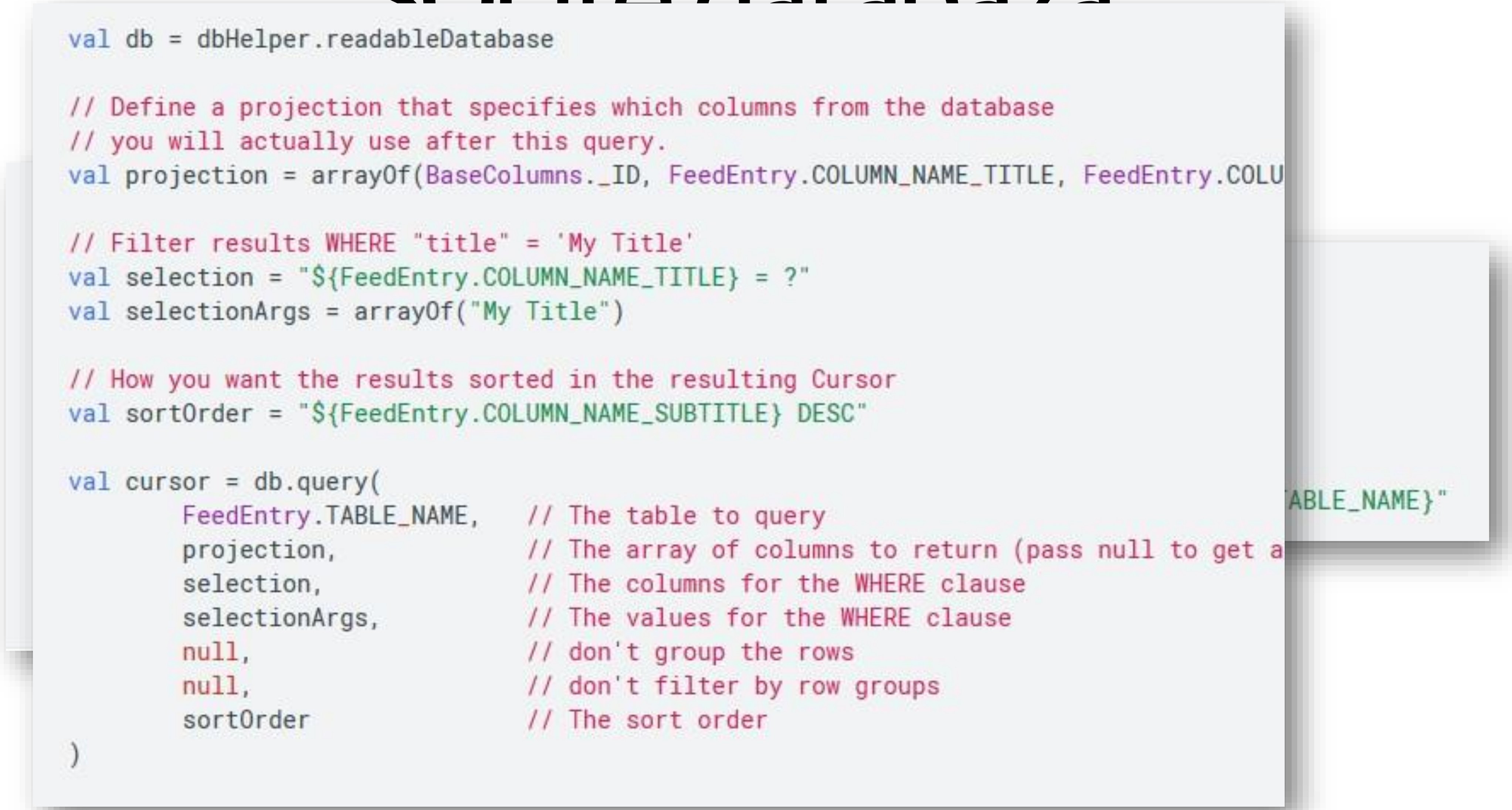
```
class FeedReaderDbHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, D
```

```
// Gets the data repository in write mode
val db = dbHelper.writableDatabase

// Create a new map of values, where column names are the keys
val values = ContentValues().apply {
    put(FeedEntry.COLUMN_NAME_TITLE, title)
    put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle)
}

// Insert the new row, returning the primary key value of the new row
val newRowId = db?.insert(FeedEntry.TABLE_NAME, null, values)
```

```
try.TABLE_NAME}"
```

```
    }
}
```

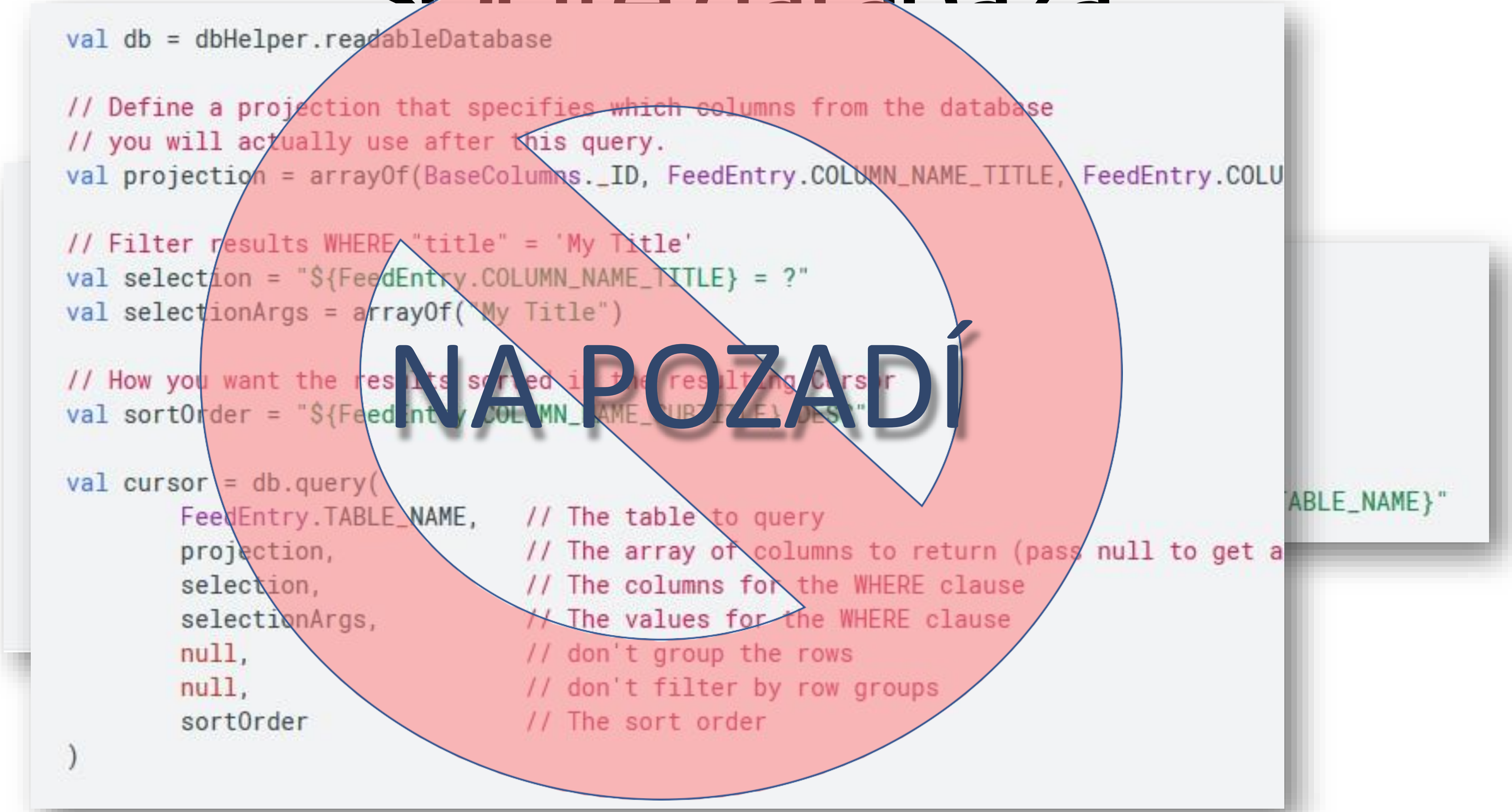
SQLite databáza



```
val db = dbHelper.readableDatabase

// Define a projection that specifies which columns from the database
// you will actually use after this query.
val projection = arrayOf(BaseColumns._ID, FeedEntry.COLUMN_NAME_TITLE, FeedEntry.COLU

// Filter results WHERE "title" = 'My Title'
val selection = "${FeedEntry.COLUMN_NAME_TITLE} = ?"
val selectionArgs = arrayOf("My Title")

// How you want the results sorted in the resulting Cursor
val sortOrder = "${FeedEntry.COLUMN_NAME_SUBTITLE} DESC"

val cursor = db.query(
        FeedEntry.TABLE_NAME,   // The table to query
        projection,             // The array of columns to return (pass null to get a
        selection,              // The columns for the WHERE clause
        selectionArgs,          // The values for the WHERE clause
        null,                   // don't group the rows
        null,                   // don't filter by row groups
        sortOrder               // The sort order
)
```

```
ABLE_NAME}"
```

SQLite databáza

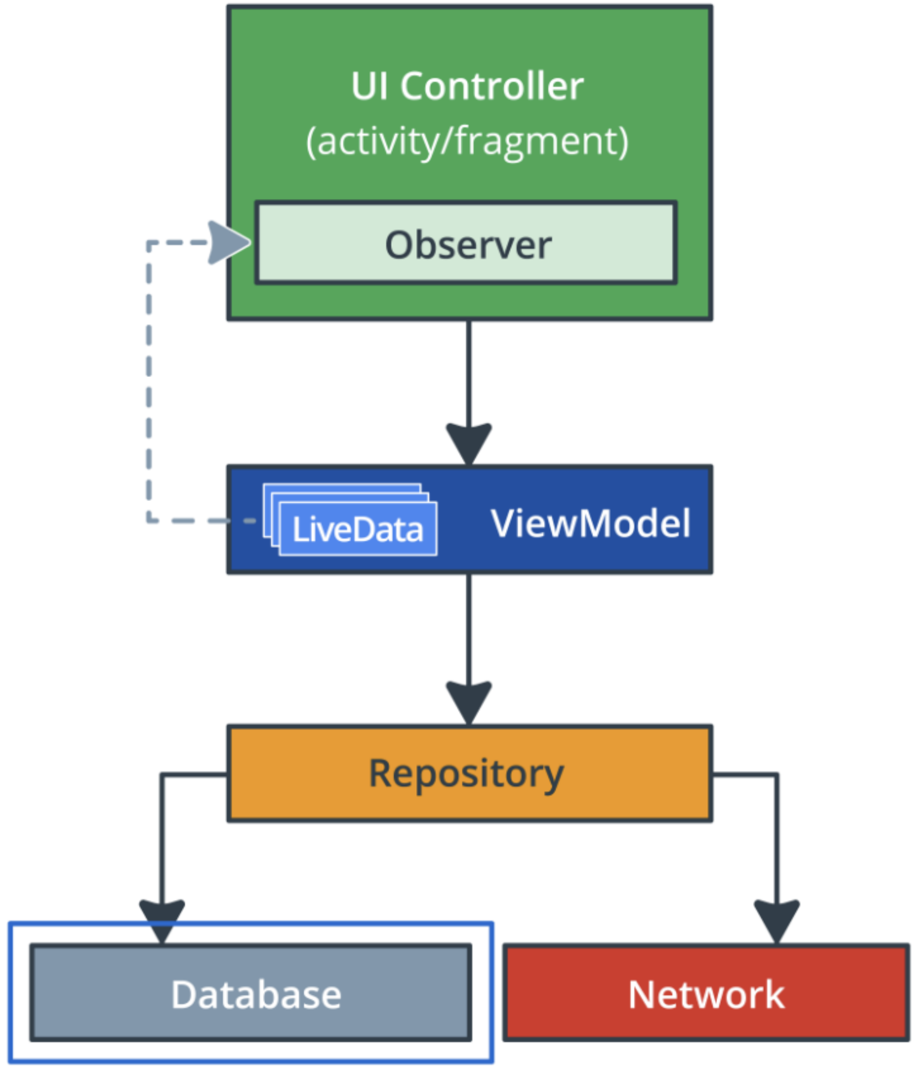


```
val db = dbHelper.readableDatabase

// Define a projection that specifies which columns from the database
// you will actually use after this query.
val projection = arrayOf(BaseColumns._ID, FeedEntry.COLUMN_NAME_TITLE, FeedEntry.COLU

// Filter results WHERE "title" = 'My Title'
val selection = "${FeedEntry.COLUMN_NAME_TITLE} = ?"
val selectionArgs = arrayOf("My Title")

// How you want the results sorted in the resulting Cursor
val sortOrder = "${FeedEntry.COLUMN_NAME_SUBTITLE} DESC"

val cursor = db.query(
        FeedEntry.TABLE_NAME,   // The table to query
        projection,             // The array of columns to return (pass null to get a
        selection,              // The columns for the WHERE clause
        selectionArgs,          // The values for the WHERE clause
        null,                   // don't group the rows
        null,                   // don't filter by row groups
        sortOrder               // The sort order
)
```

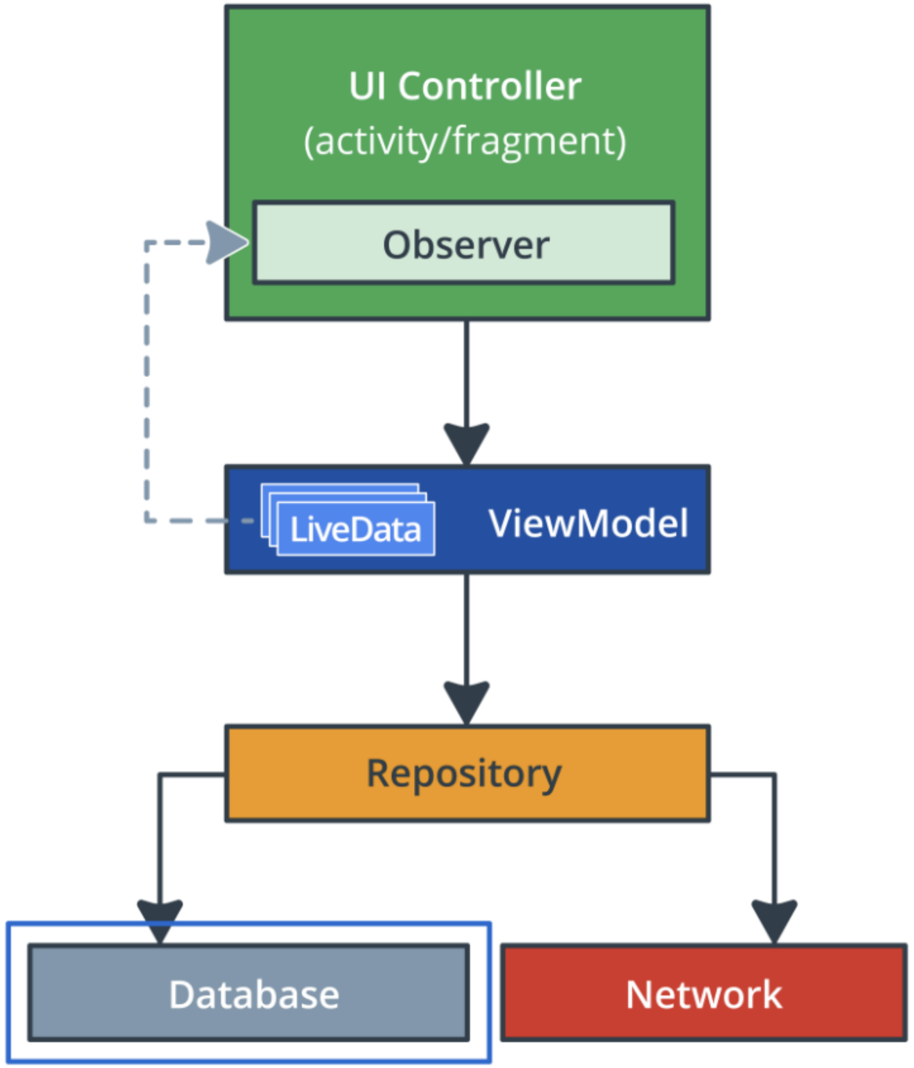
```
ABLE_NAME}"
```

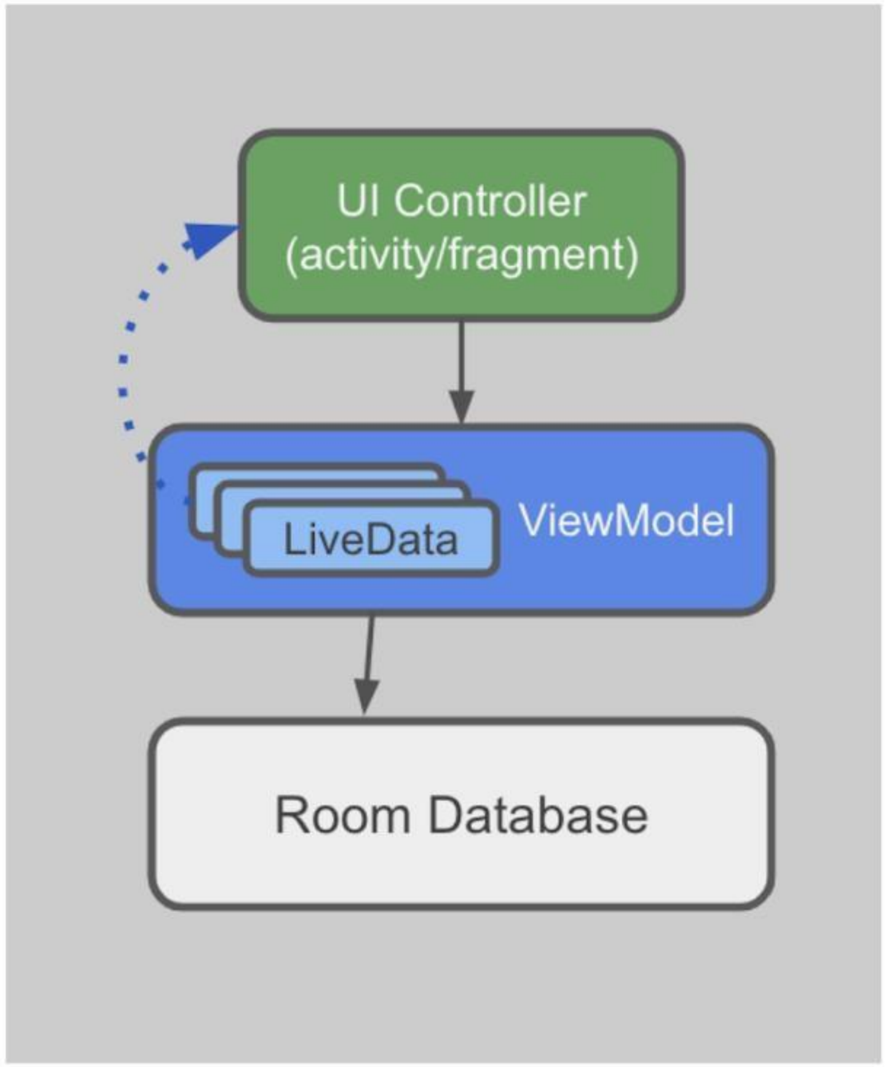
NA POZADÍ

SQLite databáza - Room

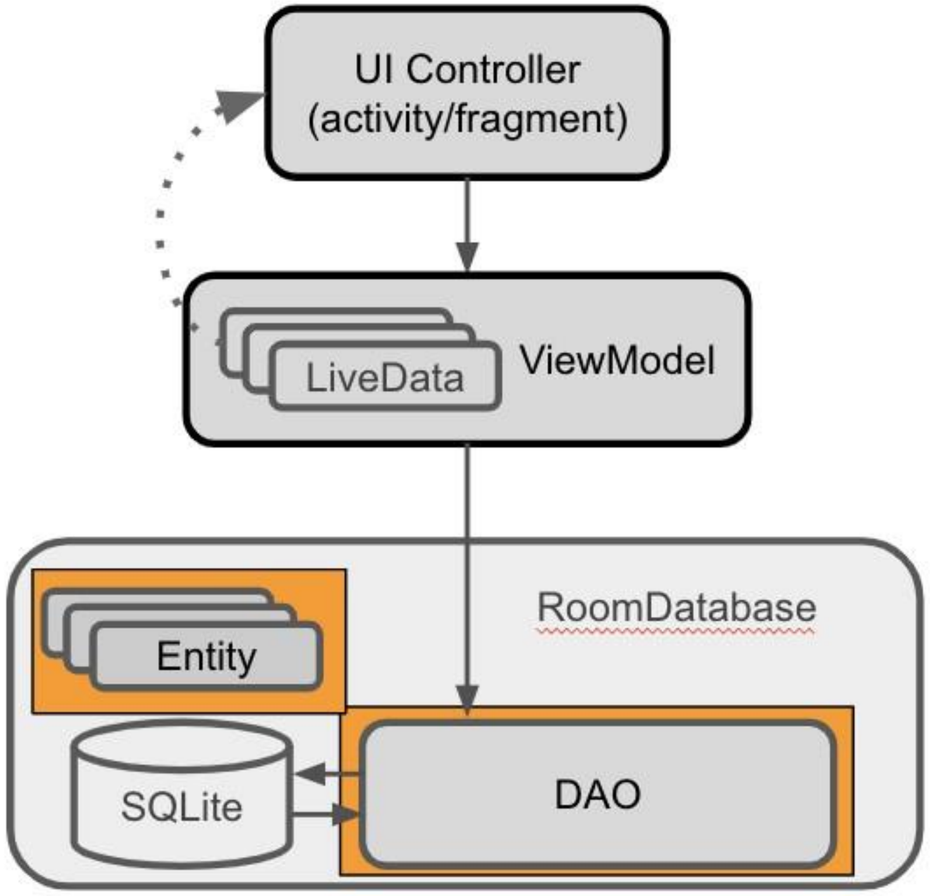


SQLite databáza - Room



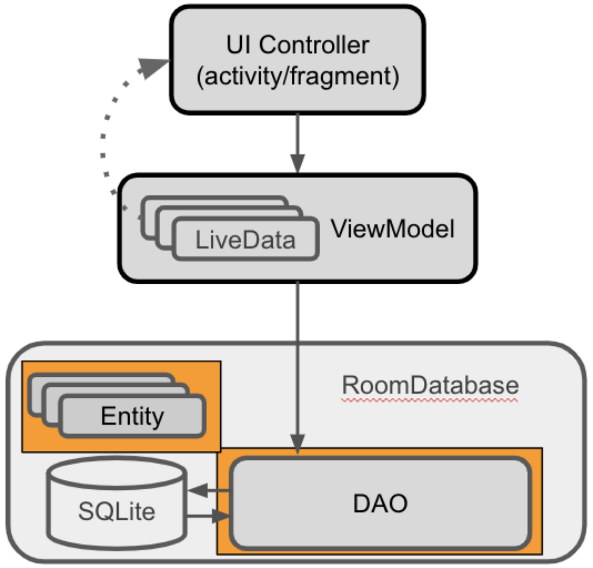


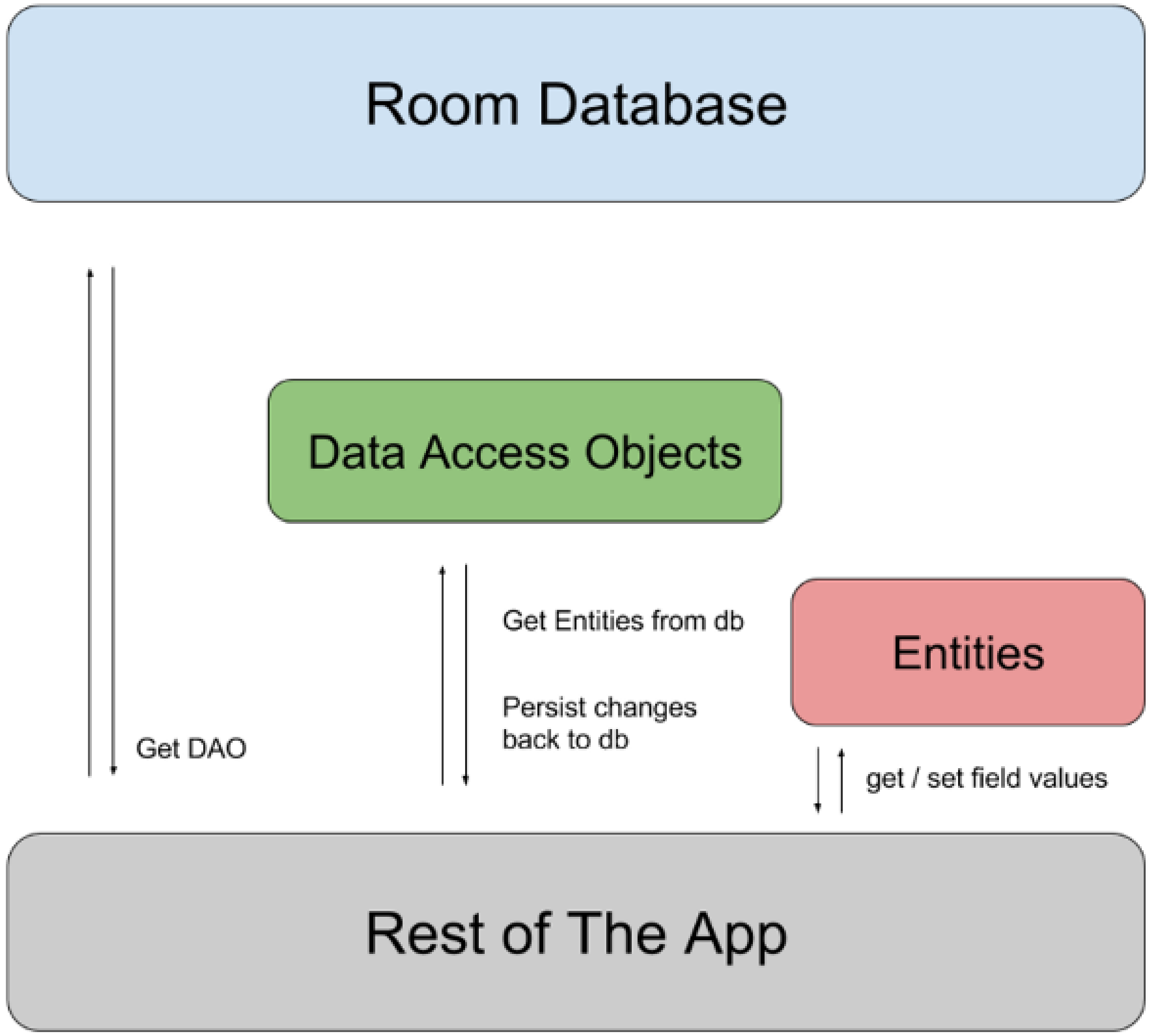
SQLite databáza - Room



SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SQLite databáza - Room

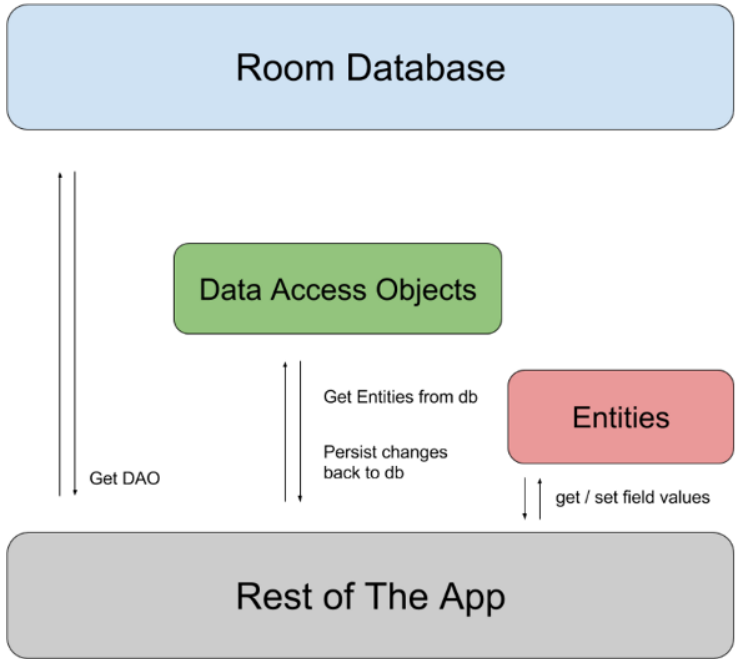




SQLite databáza - Room

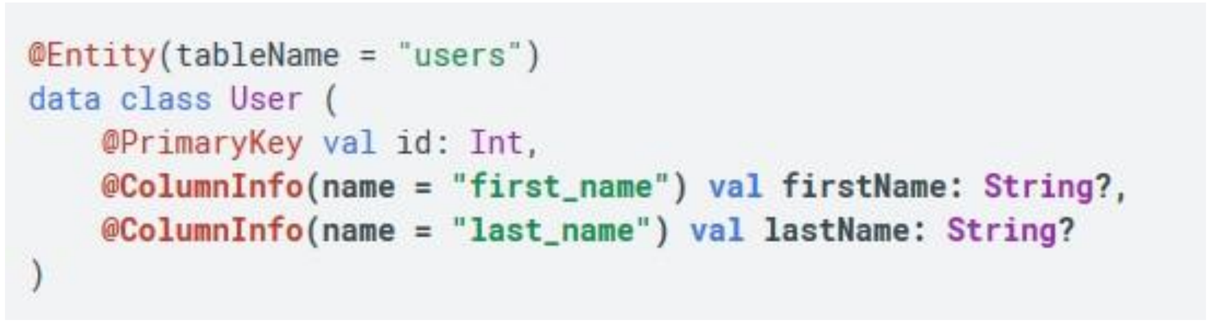
```kotlin
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

```kotlin
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

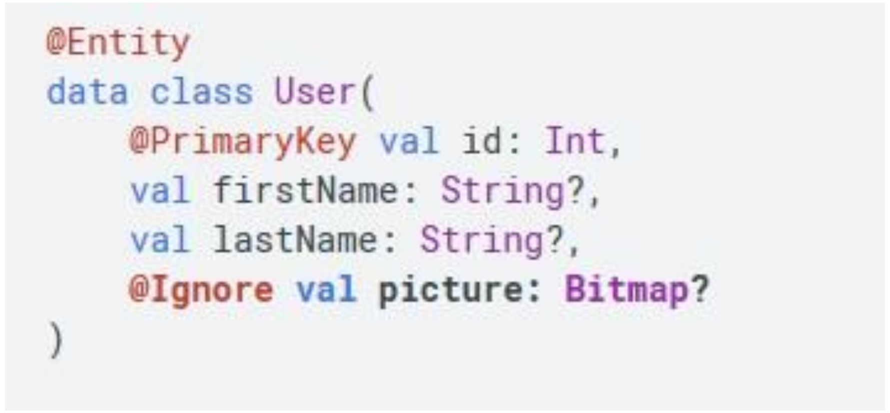


```kotlin
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

SQLite databáza - Room



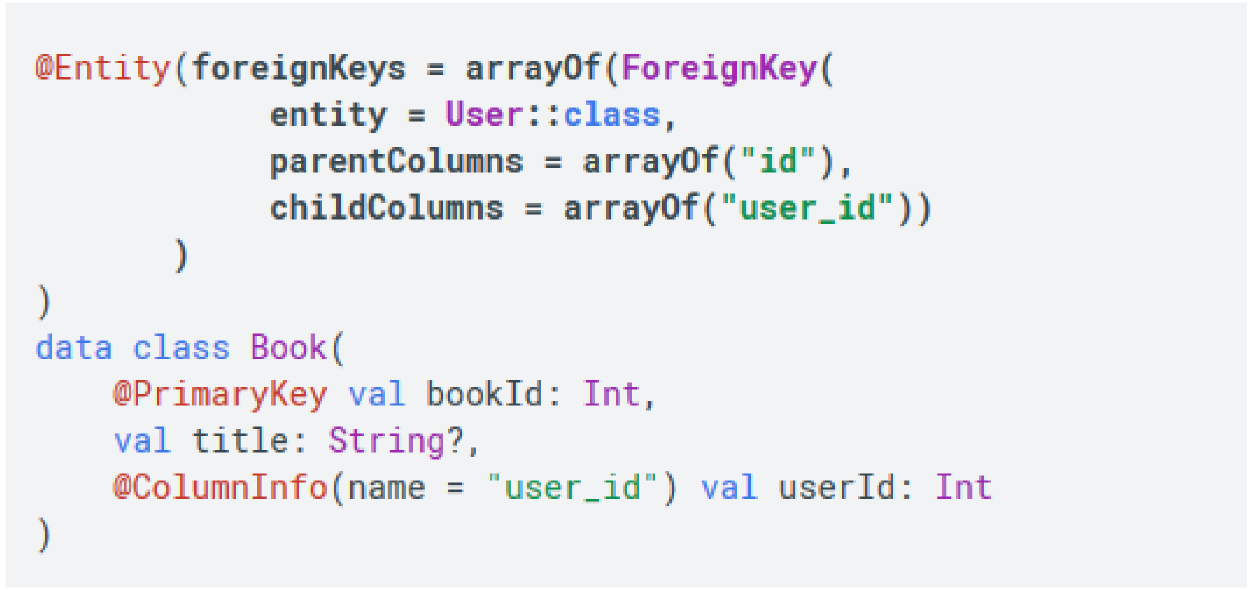
```kotlin
@Entity(tableName = "users")
data class User (
    @PrimaryKey val id: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```



```kotlin
@Entity
data class User(
    @PrimaryKey val id: Int,
    val firstName: String?,
    val lastName: String?,
    @Ignore val picture: Bitmap?
)
```

- https://developer.android.com/training/data-storage/room/defining-data

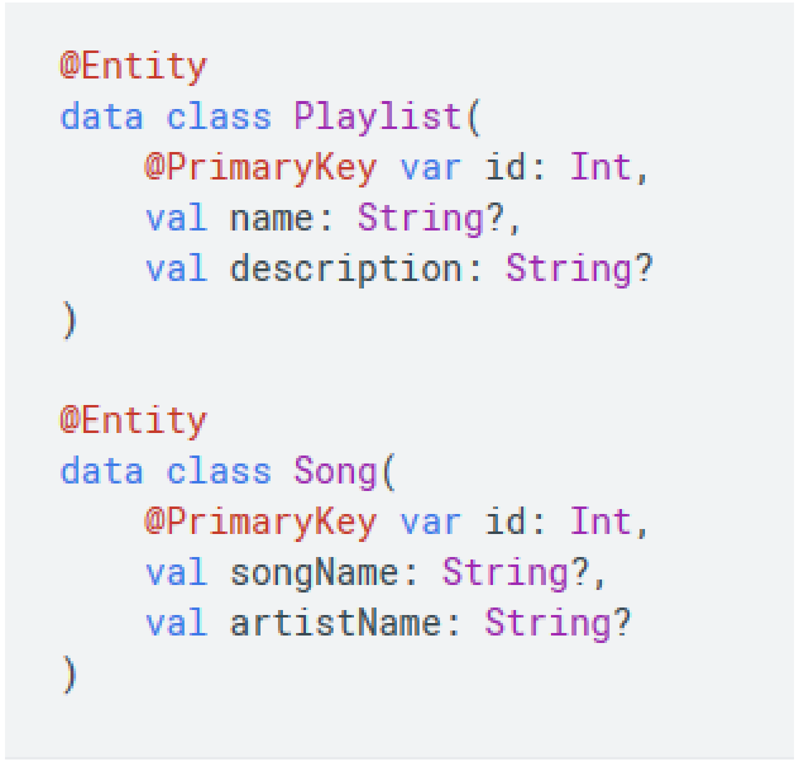
SQLite databáza - Room



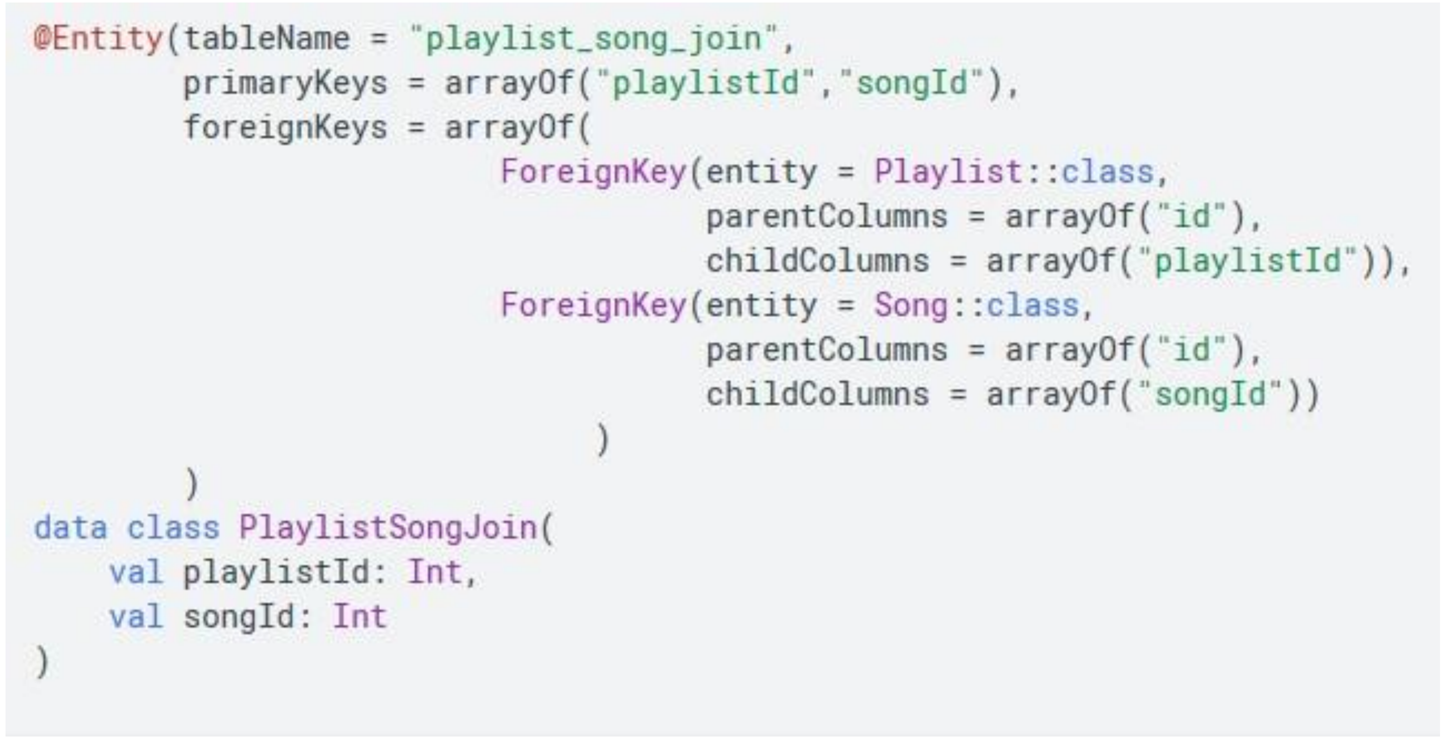
```kotlin
@Entity(foreignKeys = arrayOf(ForeignKey(
            entity = User::class,
            parentColumns = arrayOf("id"),
            childColumns = arrayOf("user_id"))
       )
)
data class Book(
    @PrimaryKey val bookId: Int,
    val title: String?,
    @ColumnInfo(name = "user_id") val userId: Int
)
```

- https://developer.android.com/training/data-storage/room/relationships

Room – M:N vzťah



```
@Entity
data class Playlist(
    @PrimaryKey var id: Int,
    val name: String?,
    val description: String?
)

@Entity
data class Song(
    @PrimaryKey var id: Int,
    val songName: String?,
    val artistName: String?
)
```



```
@Entity(tableName = "playlist_song_join",
        primaryKeys = arrayOf("playlistId","songId"),
        foreignKeys = arrayOf(
                         ForeignKey(entity = Playlist::class,
                                    parentColumns = arrayOf("id"),
                                    childColumns = arrayOf("playlistId")),
                         ForeignKey(entity = Song::class,
                                    parentColumns = arrayOf("id"),
                                    childColumns = arrayOf("songId"))
                         )
        )
data class PlaylistSongJoin(
    val playlistId: Int,
    val songId: Int
)
```

- https://developer.android.com/training/data-storage/room/relationships

SQLite databáza - Room

```
@Dao
interface MyDao {
    @Query("SELECT * FROM user WHERE age BETWEEN :minAge AND :maxAge")
    fun loadAllUsersBetweenAges(minAge: Int, maxAge: Int): Array<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :search " +
           "OR last_name LIKE :search")
    fun findUserWithName(search: String): List<User>
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
fun insertUsers(vararg users: User)
```

```
@Update
fun updateUsers(vararg users: User)
```

```
@Delete
fun deleteUsers(vararg users: User)
```

SQLite databáza - Room

```
@Dao
interface MyDao {
    @Query("SELECT * FROM user WHERE age BETWEEN :minAge AND :maxAge")
    fun loadAllUsersBetweenAges(minAge: Int, maxAge: Int): Array<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :search " +
           "OR last_name LIKE :search")
    fun findUserWithName(search: String): List<User>
```

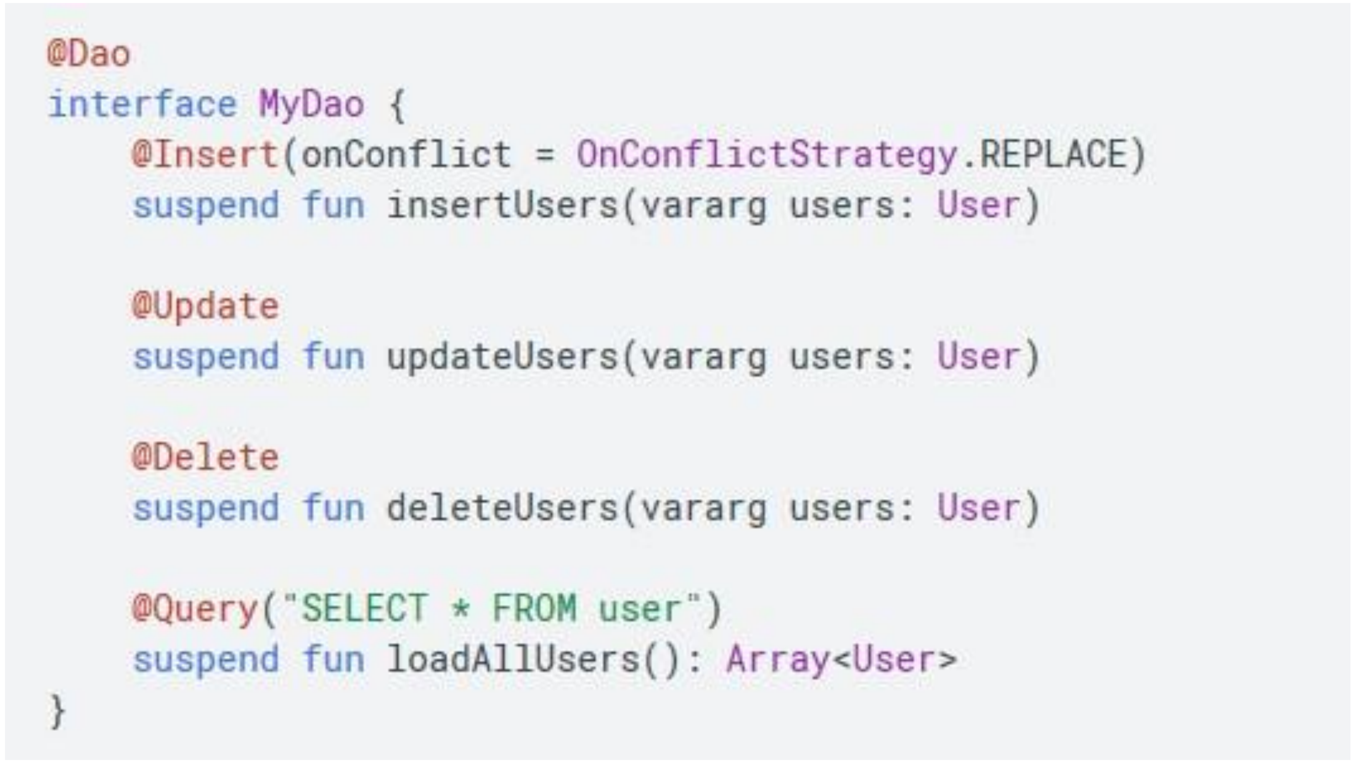
NA POZADÍ

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
fun insertUsers(vararg users: User)
```

```
@Update
fun updateUsers(vararg users: User)
```

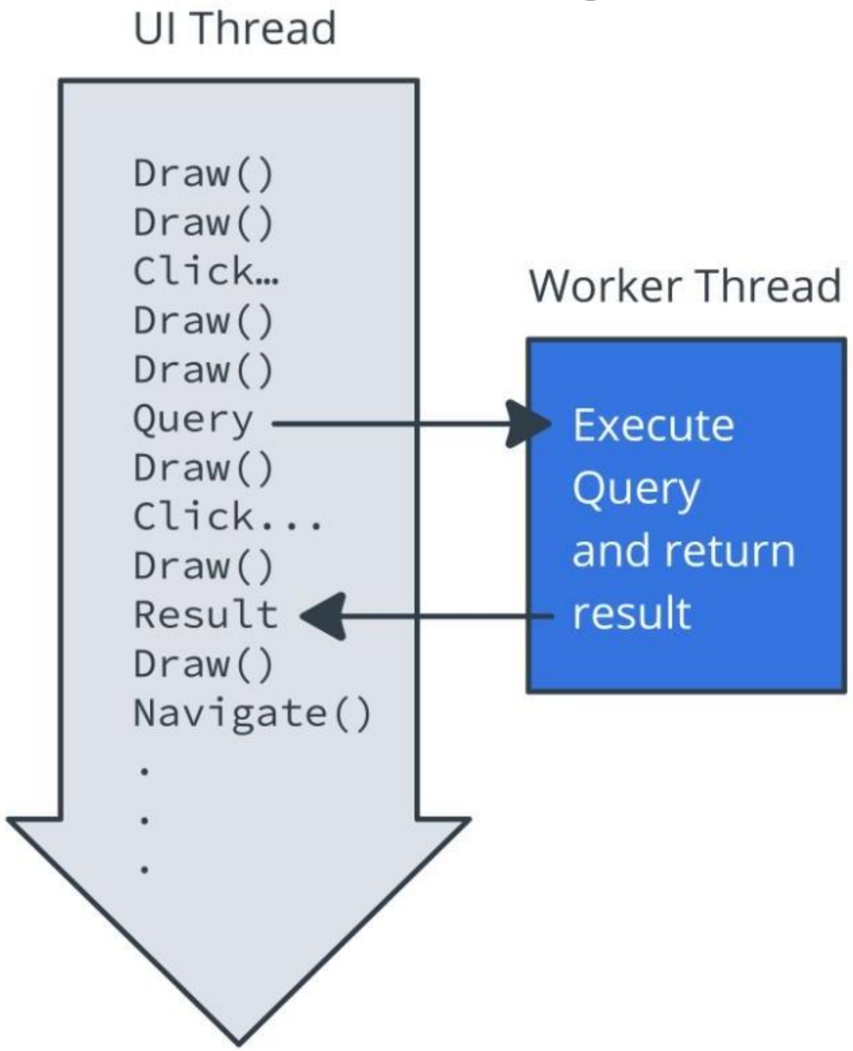
```
@Delete
fun deleteUsers(vararg users: User)
```

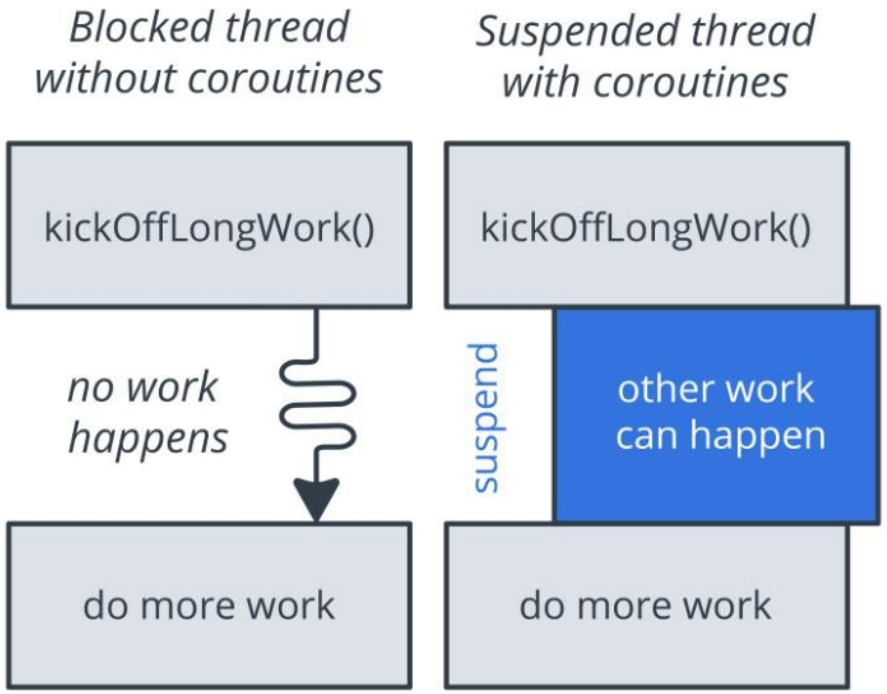
Coroutines + Room



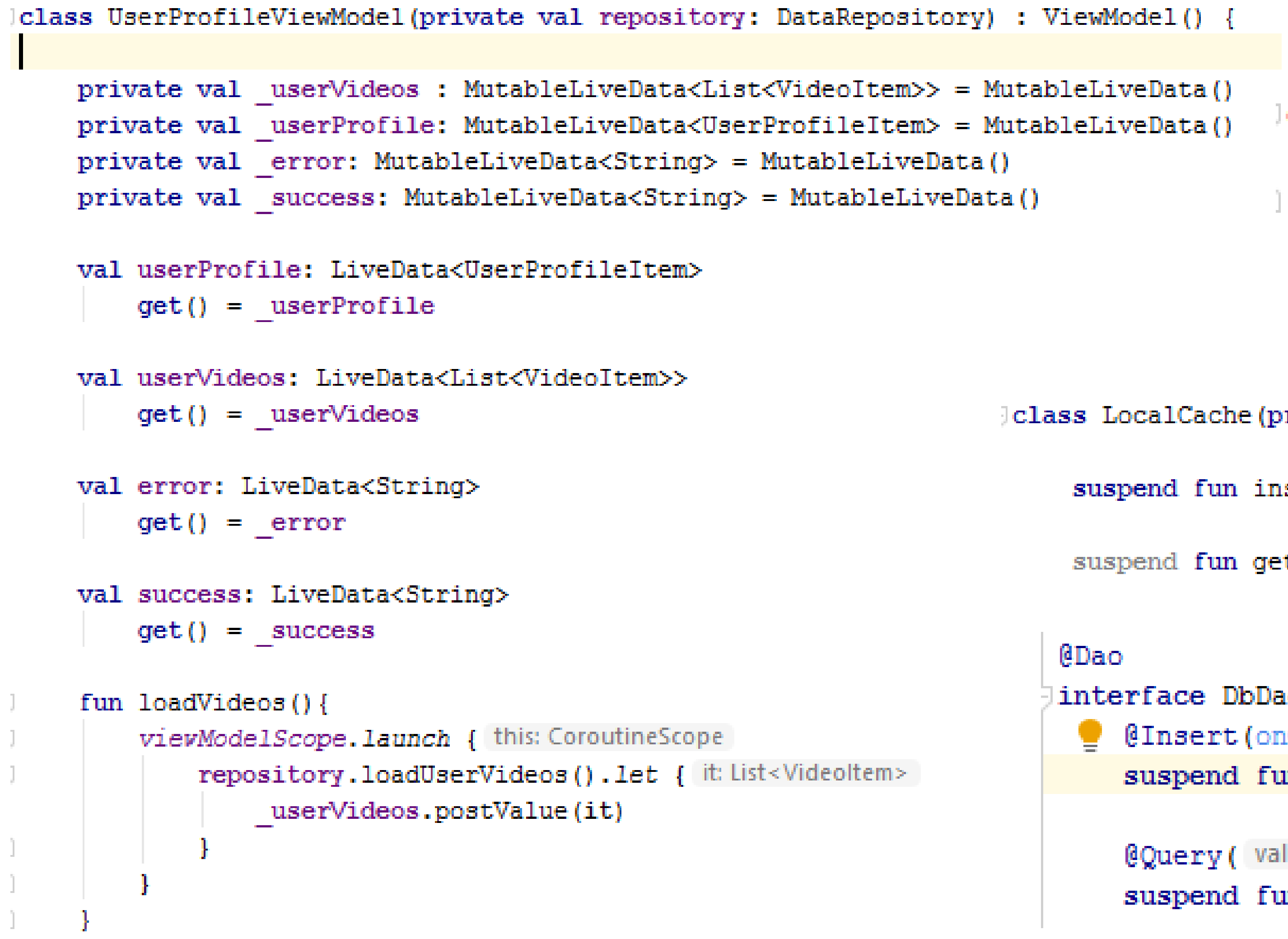
```kotlin
@Dao
interface MyDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUsers(vararg users: User)

    @Update
    suspend fun updateUsers(vararg users: User)

    @Delete
    suspend fun deleteUsers(vararg users: User)

    @Query("SELECT * FROM user")
    suspend fun loadAllUsers(): Array<User>
}
```

Coroutines + Room

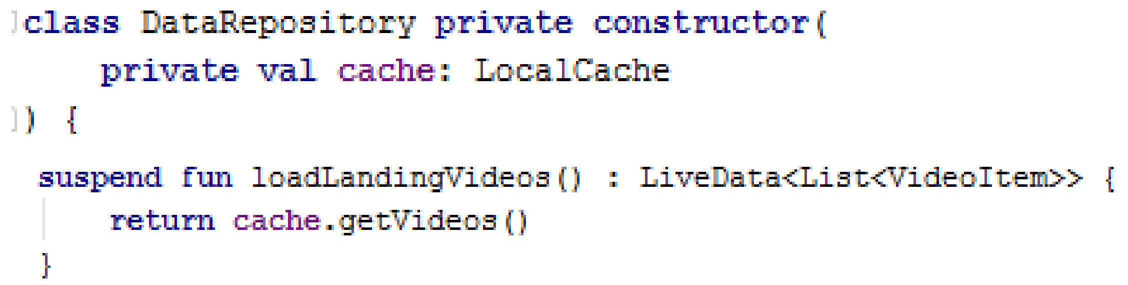




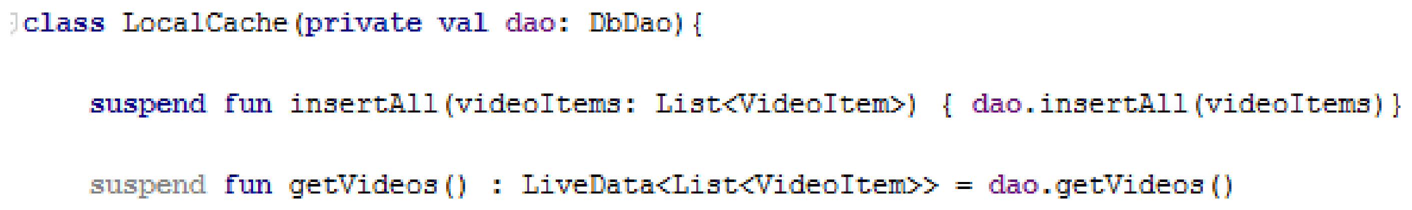
Coroutines + Room



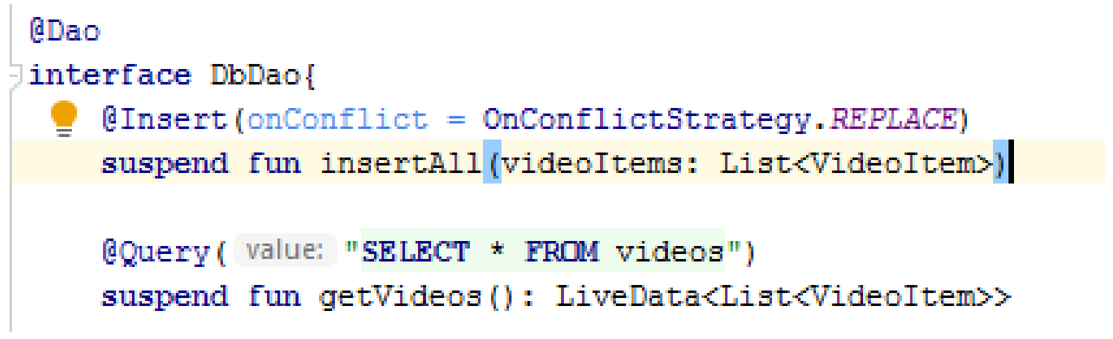
```
class UserProfileViewModel(private val repository: DataRepository) : ViewModel() {

    private val _userVideos : MutableLiveData<List<VideoItem>> = MutableLiveData()
    private val _userProfile: MutableLiveData<UserProfileItem> = MutableLiveData()
    private val _error: MutableLiveData<String> = MutableLiveData()
    private val _success: MutableLiveData<String> = MutableLiveData()

    val userProfile: LiveData<UserProfileItem>
        get() = _userProfile

    val userVideos: LiveData<List<VideoItem>>
        get() = _userVideos

    val error: LiveData<String>
        get() = _error

    val success: LiveData<String>
        get() = _success

    fun loadVideos(){
        viewModelScope.launch { this: CoroutineScope
            repository.loadUserVideos().let { it: List<VideoItem>
                _userVideos.postValue(it)
            }
        }
    }
```



```
class DataRepository private constructor(
    private val cache: LocalCache
) {

  suspend fun loadLandingVideos() : LiveData<List<VideoItem>> {
      return cache.getVideos()
  }
```

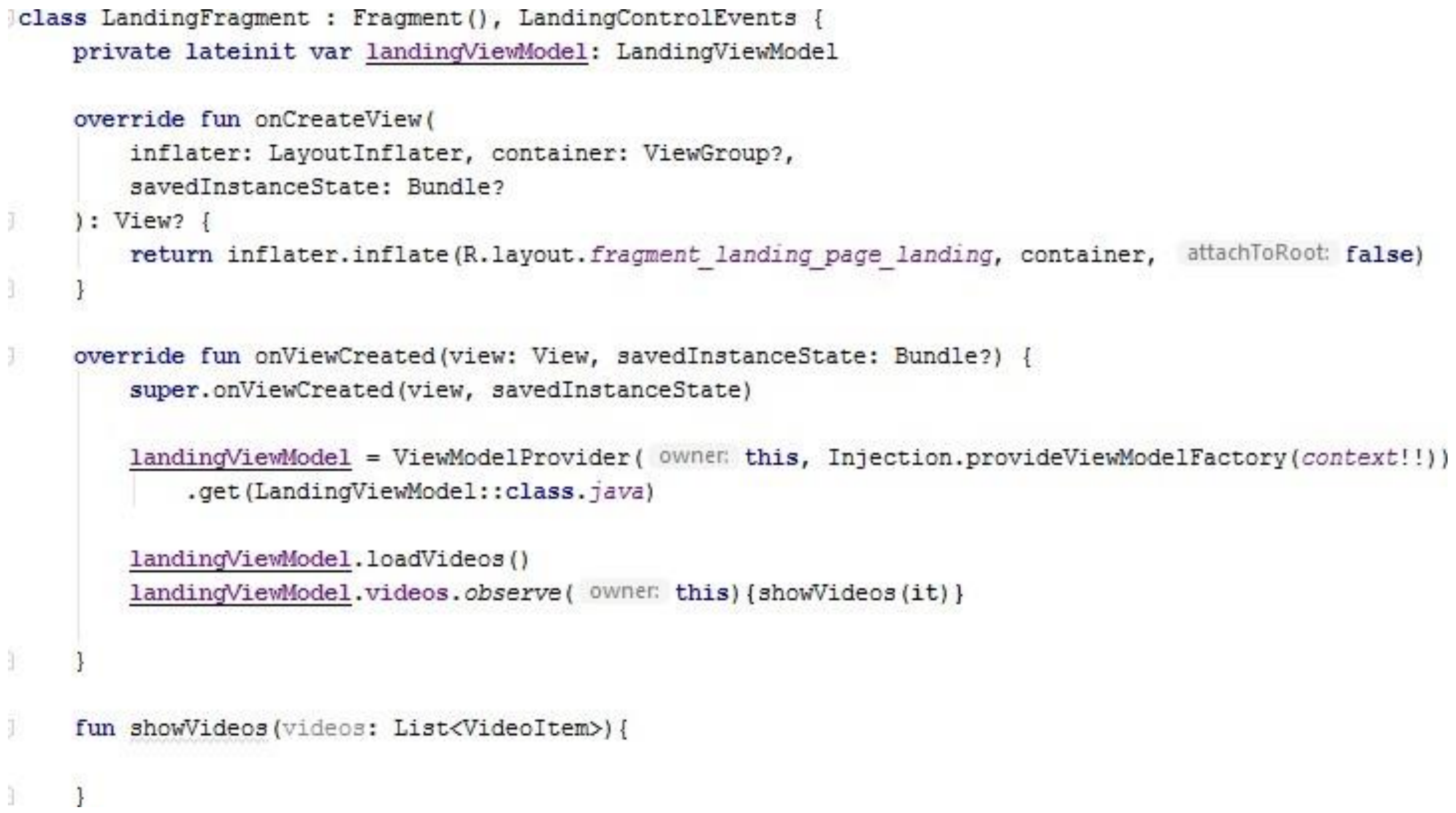


```
class LocalCache(private val dao: DbDao){

    suspend fun insertAll(videoItems: List<VideoItem>) { dao.insertAll(videoItems)}

    suspend fun getVideos() : LiveData<List<VideoItem>> = dao.getVideos()
```



```
@Dao
interface DbDao{
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(videoItems: List<VideoItem>)

    @Query( value: "SELECT * FROM videos")
    suspend fun getVideos(): LiveData<List<VideoItem>>
```

Coroutines + Room



```kotlin
class LandingFragment : Fragment(), LandingControlEvents {
    private lateinit var landingViewModel: LandingViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_landing_page_landing, container,  attachToRoot: false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        landingViewModel = ViewModelProvider( owner: this, Injection.provideViewModelFactory(context!!))
            .get(LandingViewModel::class.java)

        landingViewModel.loadVideos()
        landingViewModel.videos.observe( owner: this){showVideos(it)}

    }

    fun showVideos(videos: List<VideoItem>){

    }
```

Mobilné výpočty

Ing. Maroš Čavojský, PhD.

maros.cavojsky@stuba.sk
C606