WaffleStudio Android Seminar - 2

이승민 (안드로이드 세미나장)

2021.09.11.(토) 11:30 ~

What we will learn in Seminar 2

- Room DB
  - Database
  - Repository Pattern (MVVM)
- Recycler View
  - View Holder

Room DB - Database

- 데이터베이스는 여러 사람이 공유하여 사용할 목적으로 체계화해 통합, 관리하는 데이터의 집합 - Wikipedia
- 작성된 목록으로써 여러 응용 시스템들의 통합된 정보들을 저장하여 운영할 수 있는 공용 데이터들의 묶음 – Wikipedia
- 그 중 가장 널리 사용되는 DB는 RDB (Relational Database)

Room DB - Database

- 사람의 정보를 저장하는 표 (table) 생성
- 사람의 정보의 종류를 생성 (schema 설계)
- 사람의 정보(row)를 추가/읽기/변경/삭제 (instance에 대한 CRUD)
- 연도의 정보를 저장하는 표 (table) 과 연도의 정보의 종류 (schema) 생성
- 특정 사람의 연도와 각 연도의 정보를 연결 - 관계(relation) 생성

CREATE
READ
UPDATE
DELETE

사람 table

이름	나이	성별	생년
이승민	22	남	2000
김익명	21	여	2001
박무명	24	남	1998

연도 table

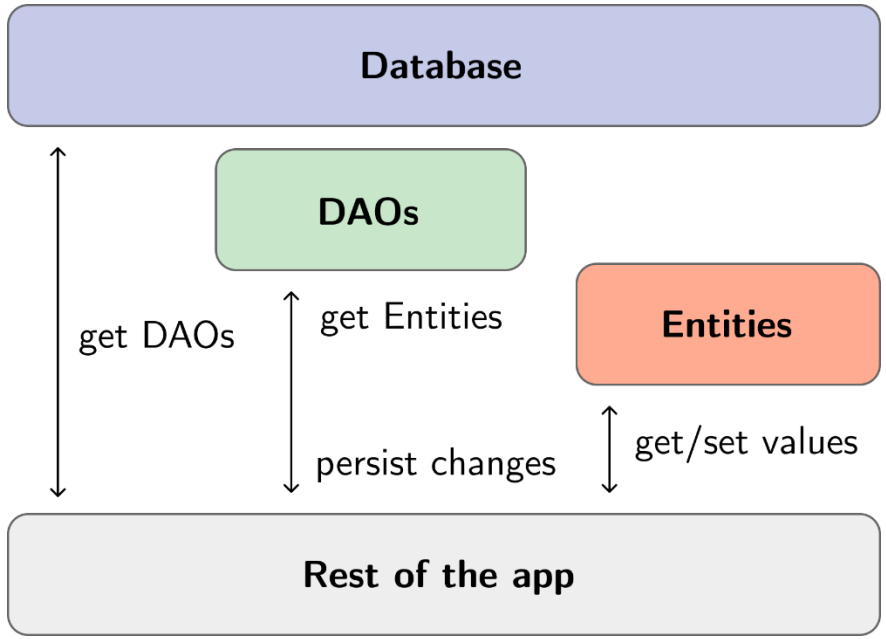
연도	인구
1998	100000
1999	200000
2000	400000
2001	300000

Room DB

- Android OS 에 내장되어 있는 로컬 데이터 베이스가 존재 (SQLite)
  - 우리는 앱에서 영원히 저장되어야 하는 값들을 local database 에 저장함
  - 저장된 정보는 앱이 꺼져도 유지되고 다시 시작할 때 읽어올 수 있음
  - 유저 정보나 데이터가 없을 때도 값을 읽고 싶다면 꼭 local db를 사용해야 함
- 이 내장 local DB 를 쉽게 사용하기 위한 Library : Room

Room DB

- Database
  - 아까 배운 그거
- DAO
  - Data Access Object
  - DB의 table에 대한 접근을 담당
- Entities (Model)
  - table
  - DB에 자동으로 저장됨



Room DB

- Entities (Model)
  - DB의 table을 정의 (schema)
  - 원하는 정보의 타입과 이름 명시

```kotlin
@Entity(tableName = "memos")
data class Memo(
    @ColumnInfo(name = "title")
    val title: String,
    @ColumnInfo(name = "detail")
    val detail: String
) {
    @PrimaryKey(autoGenerate = true)
    var id: Int = 0
}
```

Room DB

suspend fun 은 일단 신경쓰지 말자

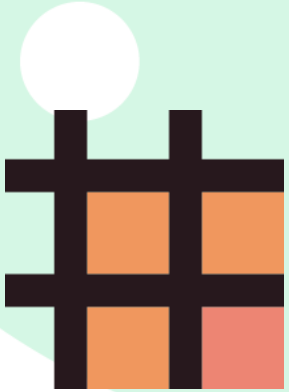
```kotlin
@Dao
interface MemoDao {

    @Query( value: "SELECT * FROM memos")
    fun getAllMemos(): LiveData<List<Memo>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertMemo(memo: Memo)

    @Delete
    suspend fun deleteMemo(memo: Memo)

    @Query( value: "DELETE FROM memos WHERE id=:id")
    suspend fun deleteMemoById(id: Int)
}
```

- DAO
  - DB의 table과 소통하기 위한 방법 명시
  - 함수가 어떤 작업을 수행할지를 annotation으로 명시
  - 명시된 annotation에 따라 DB와 소통

Room DB

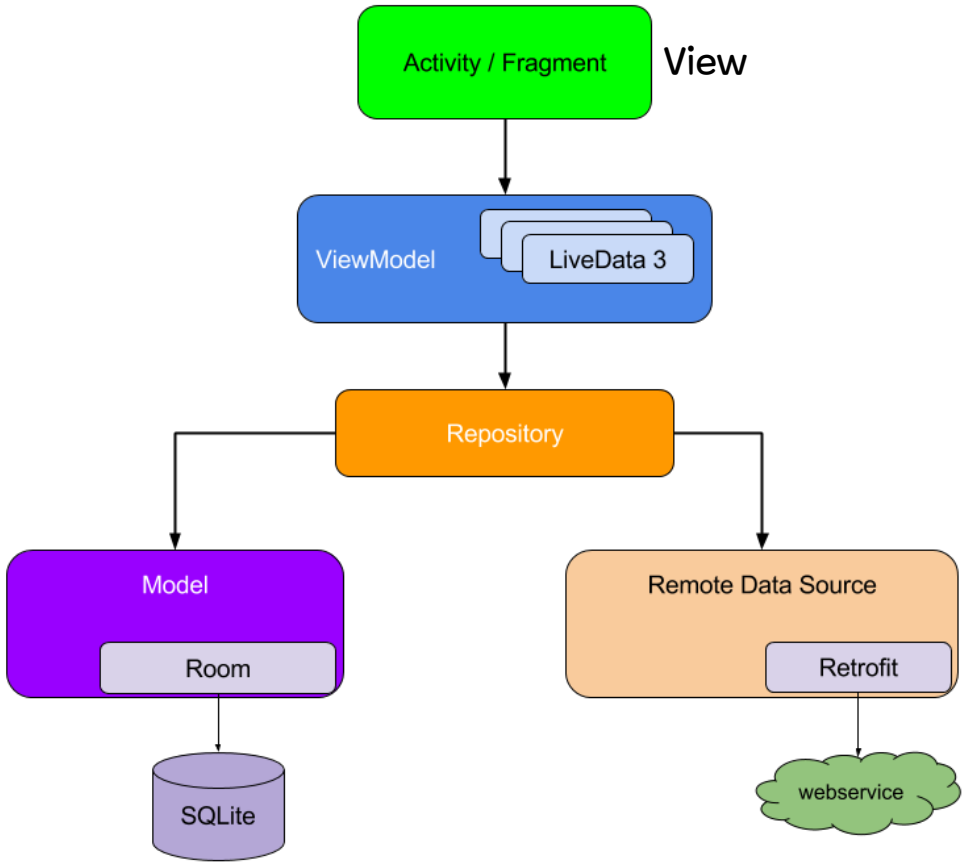
```kotlin
@Database(entities = [Memo::class], version = 1)
abstract class MemoDatabase : RoomDatabase() {
    abstract fun memoDao(): MemoDao

    companion object {
        @Volatile
        private var INSTANCE: MemoDatabase? = null

        @JvmStatic
        fun getInstance(context: Context): MemoDatabase = INSTANCE ?: synchronized( lock: this) {
            INSTANCE ?: Room.databaseBuilder(
                context.applicationContext,
                MemoDatabase::class.java,
                name: "memo_db"
            ).build().also {  it: MemoDatabase
                INSTANCE = it
            }
        }
    }
}
```

- Database
  - Singleton 한 DB를 생성 및 유지
  - 어떤 Entity(table)를 관리하는지
  - 어떤 Dao를 사용할 수 있는지
- Singleton Pattern
  - 프로그램 전체에서 객체를 단 하나만 생성하도록 하는 디자인 패턴



Room DB - Repository Pattern (MVVM)

- 다시 만난 MVVM
  - View는 ViewModel에서
  - ViewModel은 Repository에서
  - Repository는
    - Local DB에서
    - Network에서
  - 정보를 가져온다!



Room DB - Repository Pattern (MVVM)

- 역시 Singleton 하게

```
class MemoRepository(private val memoDao: MemoDao) {

    fun getMemos() = memoDao.getAllMemos()
    suspend fun addMemo(memo: Memo) = memoDao.insertMemo(memo)

    companion object {
        @Volatile
        private var INSTANCE: MemoRepository? = null

        @JvmStatic
        fun getInstance(memoDao: MemoDao): MemoRepository = INSTANCE ?: synchronized( lock: this) {
            INSTANCE ?: MemoRepository(memoDao).also {  it: MemoRepository
                INSTANCE = it
            }
        }
    }
}
```

Room DB - Repository Pattern (MVVM)

- 어떻게 연결? -> 일단은 Application 이 가지고 있자 (다음 세미나에 진행 예정)

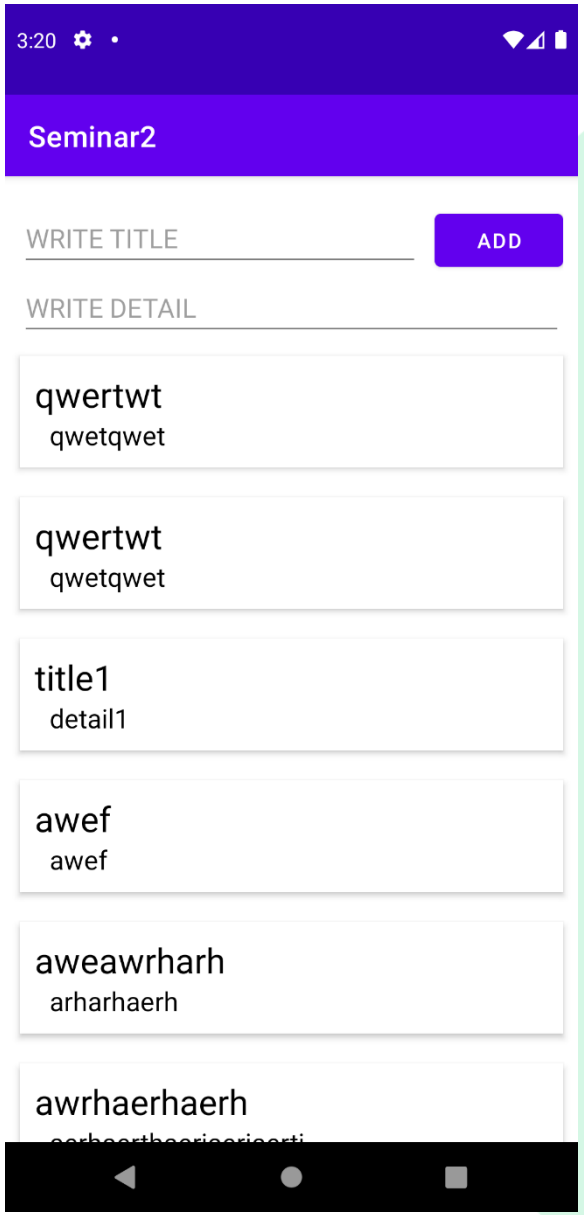
```
class App : Application() {

    private val memoDatabase by lazy { MemoDatabase.getInstance( context: this) }
    val memoRepository by lazy { MemoRepository.getInstance(memoDatabase.memoDao()) }

    override fun onCreate() {
        super.onCreate()

        if (BuildConfig.DEBUG) {
            Timber.plant(Timber.DebugTree())
        }
    }
}
```

```
class MainViewModel(application: Application) : AndroidViewModel(application) {

    private val memoRepository by lazy { (application as App).memoRepository }

}
```

AndroidViewModel : application context 를 사용할 수 있도록 해주는 Class

Recyclerview

- 일반적인 UI 이자, 매우 많이 쓰이는 UI
- 여러 아이템을 보여주는 List 형태의 UI
- List -> 수많은 아이템들이 render 될 수 있다
  - 따라서 성능상의 이슈가 발생할 수 있다
- Ex) 10000개의 Item이 있는 List라면 10000개의 View를 렌더링해서 앱이 가지고 있어야할까?
  - View의 생성 및 유지는 상당히 무거운 작업



Recyclerview

- 해결법
  - 사용자에게 보이는 View만 만들어놓고 데이터만 바꾸면서 재활용(Recycle)하자!
  - 스크롤을 내리면 -> 올라가서 보이지 않는 View를 재활용 -> 아래에서 올라오기
- 그럼 어떻게 재활용하지?
  - 어디에 끼우지? - ViewHolder
  - 어떻게 끼우지? - viewBinding
  - 누가 끼우지? - Adapter
  - 뭘 끼우지? - data list
  - ~~언제?~~
  - ~~왜?~~

Recyclerview

```
class MemoAdapter : RecyclerView.Adapter<MemoAdapter.MemoViewHolder>() {

    private var memos: List<Memo> = listOf()

    inner class MemoViewHolder(val binding: ItemMemoBinding)
        : RecyclerView.ViewHolder(binding.root)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MemoViewHolder {
        val binding = ItemMemoBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            attachToParent: false
        )
        return MemoViewHolder(binding)
    }

    override fun onBindViewHolder(holder: MemoViewHolder, position: Int) {
        val data = memos[position]
        holder.binding.apply { this: ItemMemoBinding
            textTitle.text = data.title
            textDetail.text = data.detail
        }
    }

    override fun getItemCount() = memos.size

    fun setMemos(memos: List<Memo>) {
        this.memos = memos
        this.notifyDataSetChanged()
    }
}
```

```
memoAdapter = MemoAdapter()
memoLayoutManager = LinearLayoutManager( context: this)
binding.recyclerViewMemo.apply { this: RecyclerView
    adapter = memoAdapter
    layoutManager = memoLayoutManager
}
```

QA