Android VI

Persistence with Room
Object relational mapping: Converts between objects in an OO programming language to relational data in a relational database.

For Android, we will use Google's Room ORM.

We will also be following Google's best practices for structuring our app with persistence.

Eventually, we will have our Github search app cache results in an SQLite database local to the phone, and populate our RecyclerViews with this information.
Recommended App Architecture: MVVM
### Adding Room Component Libraries

In `build.gradle(Module:app)`:  

```gradle
// Room components
implementation "android.arch.persistence.room:runtime:$rootProject.roomVersion"
annotationProcessor "android.arch.persistence.room:compiler:$rootProject.roomVersion"
androidTestImplementation "android.arch.persistence.room:testing:$rootProject.roomVersion"

// Lifecycle components
implementation "android.arch.lifecycle:extensions:$rootProject.archLifecycleVersion"
annotationProcessor "android.arch.lifecycle:compiler:$rootProject.archLifecycleVersion"
```

In `build.gradle((Project: ...))`:  

```gradle
ext {
    roomVersion = '1.1.1'
    archLifecycleVersion = '1.1.1'
}
```
Create an Entity

<table>
<thead>
<tr>
<th>word_table table</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
</tr>
<tr>
<td>(Primary Key, String)</td>
</tr>
<tr>
<td>&quot;Hello&quot;</td>
</tr>
<tr>
<td>&quot;World&quot;</td>
</tr>
</tbody>
</table>

@Entity(tableName = "word_table")
public class Word {

    @PrimaryKey(autoGenerate = true)
    @NonNull
    @ColumnInfo(name = "word")
    private String mWord;

    public Word(String word) {this.mWord = word;}

    public String getWord() {return this.mWord;}
}
The DAO (data access object)

- Specify SQL queries and associate them with method calls
- With Room, the compiler generates queries from convenience annotations
- Must be an interface or abstract class
- Simple SQL commands like INSERT have built in implementations

```java
@Dao
public interface WordDao {

    // @Insert: can add a conflict strategy
    // with (onConflict = OnConflictStrategy.REPLACE)
    @Insert
    void insert(Word word);

    @Query("DELETE FROM word_table")
    void deleteAll();

    @Query("SELECT * from word_table ORDER BY word ASC")
    List<Word> getAllWords();
}
```
The Observer Pattern

The Dog, Cat, and Mouse objects are subscribers, Duck isn’t. Each can decide at any time to subscribe or unsubscribe.

Publishers + subscribers = subscriber pattern
LiveData class

- Wrapper class used to observe data changes
- Implements the observer pattern
  - It does it for you, just use it
  - It also does it on a background thread for you
- To use it, just change the getAllWords() method in DAO to:

```java
@Query("SELECT * from word_table ORDER BY word ASC")
LiveData<List<Word>> getAllWords();
```
Adding a Room Database

- RoomDatabase is a database layer on top of an SQLite database
- Replaces SQLiteOpenHelper
- Forces queries to be on a background thread by default
@Database(entities = {Word.class}, version = 1)
public abstract class WordRoomDatabase extends RoomDatabase {

    public abstract WordDao wordDao();
    private static volatile WordRoomDatabase INSTANCE;

    static WordRoomDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (WordRoomDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            WordRoomDatabase.class, "word_database")
                            .build();
                }
            }
        }
        return INSTANCE;
    }
}
Repository

- Abstracts access to multiple data sources
- Not required, but a best practice for a clean data API for the rest of your app
- Can open an AsyncTask here for your insert without worry
- We will just implement the DAO part, but you could also add a network part too
- See code from demo
ViewModel

Provides data for the UI and avoids issues with configuration changes (sudden Activity death)

Do not store context or any UI components in the ViewModel, or else we may find a space for you in our notebook!

See Demo for code