Mobile solutions

Kocsis Gergely

2019.01.05.

Room

The most professional way of storing data in a persistent way is to save data into a database. Handling databases in ordinary ways of Java is also possible under Android.

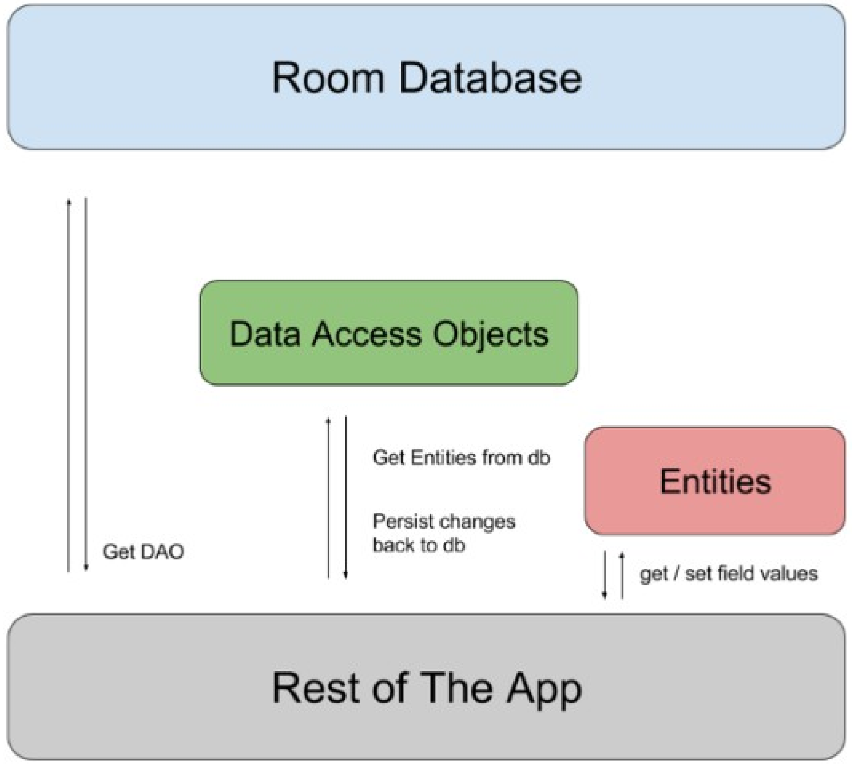
However the SDK also provides a native way for this. This is called Room.

Create an application that is similar to the previously created shopping list but is stores data in a database.

Room

A minimal Room application has to contain three components.:

- Entity: A class representing the
- DAO (Data Access Object): An interface describing the database connection. This interface is automatically implemented by Room.
- Database: An abstracet class extending the RoomDatabase class. Its task is to manage the database connection.



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

Room

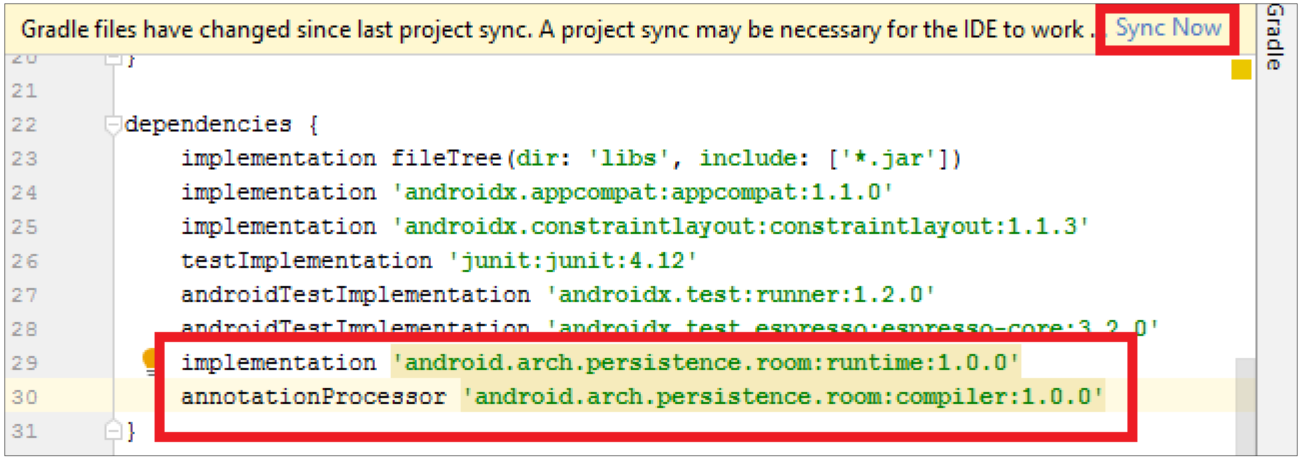
Create an empty application and an Entity class in it.

The name of the class should be ShoppingListItem. This is a simple bean class. What make it special are the annotations similar to those used in JPA.

These annotation are not processed by our application by default. We have to add the following dependencies to the gradle.build file (to the end of dependenciies). The latest available version may be used. (https://developer.android.com/jetpack/androidx/releases/room)

```
implementation 'android.arch.persistence.room:runtime:2.2.1'
annotationProcessor 'android.arch.persistence.room:compiler:2.2.1'
```

Synchronize the project before proceeding.



Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work . Sync Now

```
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'android.arch.persistence.room:runtime:1.0.0'
    annotationProcessor 'android.arch.persistence.room:compiler:1.0.0'
}
```

Room

Create an empty application and an Entity class in it.

The name of the class should be ShoppingListItem. This is a simple bean class. What make it special are the annotations similar to those used in JPA.

```java
import androidx.annotation.NonNull;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "ShoppingList")
public class ShoppingListItem {
    @PrimaryKey
    @NonNull
    private int itemID;

    private String itemName;

    //... getters and setters

    @Override
    public String toString() {
        return itemID + " - " + itemName;
    }
```

Room

The second task is to create the DAO inteface. The method signatures of this interface define the operations of the database. Here we can implement the basic CRUD operations.

For now let us just define a simple insert and a listing operation.

```java
@Dao
public interface ShoppingListDAO {
    @Insert
    void insertListItem(ShoppingListItem sli);

    @Query("SELECT * FROM ShoppingList")
    List<ShoppingListItem> getAllItems();
}
```

Table name

@Update and @Delete annotations are also defined. We will extend our project with these later.

Room

The third task is to create the database. The database is an abstract class extending the RoomDatabase class. In the class we define a “getter” method for all the entities to be handled.

```
@Database(entities = {ShoppingListItem.class}, version = 1, exportSchema = false)
public abstract class ShoppingListDatabase extends RoomDatabase {
    public abstract ShoppingListDAO shoppingListDAO();
}
```

In the @Database annotation all the entities to be handled have to be listed. The version number is to be increased whenever the scheme is changed.

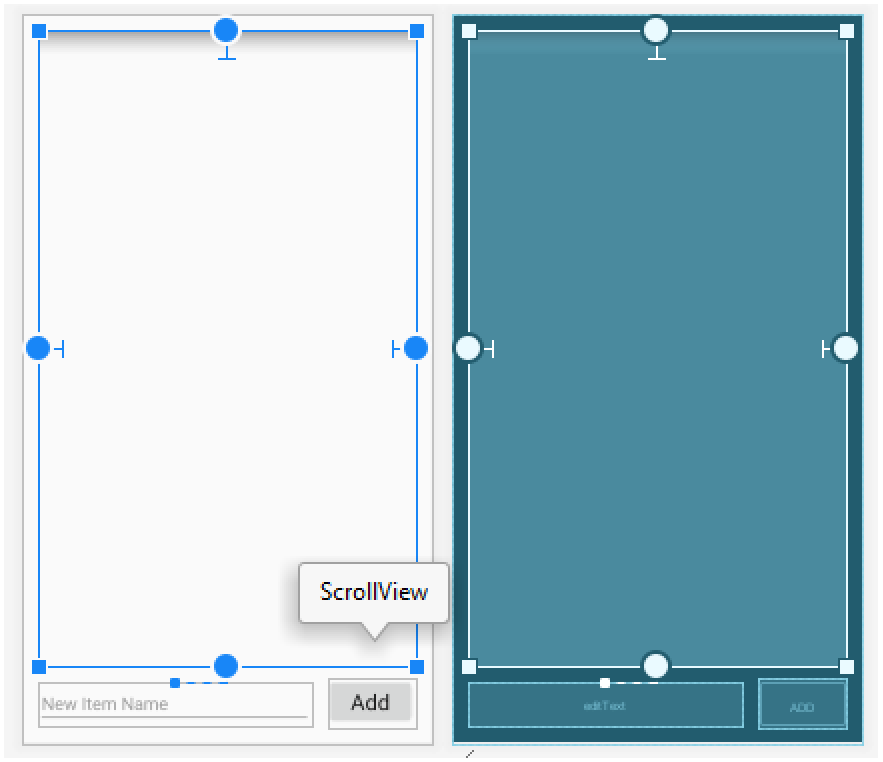
Creating the database is an expensive task. In the followings try to do it only once per run.

Room

Create the application's UI. On the UI by the use of a button we can add new items to the list. The full list is presented in a ScrollView that containds a TextView.

Pushing the Add button launches the addItem method of the MainActivity.

The hint attribute of the EditText used to give the name of the new item is "New item name".



Room

Test our database. For this we will need a reference in the MainActivity for which we also do the instantiation in the onCreate method. (Right at this point we have to note that we are following an antipattern with this!).

```
private ShoppingListDatabase shoppingListDatabase = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    shoppingListDatabase = Room.databaseBuilder(
            this, ShoppingListDatabase.class, "shoppingList_db")
            .fallbackToDestructiveMigration()
            .build();
```

The problem is that whenever the onCreate method runs the database is reinstantiated.

Room

Test our database.

Execute some simple database operations. Be careful to run these tasks not in the main UI thread. To do this we may know two different ways.

We can use AsyncTask, the native Android solution:

```
\\onCtreate
new AsyncTask<String, Void, Void>() {
    @Override
    protected Void doInBackground(String... itemData) {
        ShoppingListItem sli = new ShoppingListItem();
        sli.setItemID(Integer.parseInt(itemData[0]));
        sli.setItemName(itemData[1]);
        shoppingListDatabase.shoppingListDAO().insertListItem(sli);
        return null;
    }
}.execute("1", "Apple");
```

Room

Test our database.

Execute some simple database operations. Be careful to run these tasks not in the main UI thread. To do this we may know two different ways.

Or we can use the classical Java solution by Thread:

```
\\onCtreate
new Thread(new Runnable() {
    @Override
    public void run() {
        Log.d("CHECKdb",
          shoppingListDatabase.shoppingListDAO().getAllItems().toString());
    }
}).start();
```

Room

Using the sample query extend the MainActivity by a refrehScrollView method that presents the list in the TextView contained by the ScrollView.

Also using the samples implement the addItem method in order not to have to set the id of the item by hand..Annotate the ID field of the ShoppingListItem Entityby the followings.

```
@Entity(tableName = "ShoppingList")
public class ShoppingListItem {
    @PrimaryKey(autoGenerate = true)
    @NonNull
    private int itemID;
```

So that the Iddoes not have to be given at the time of adding a new item. Since tha schema has been modified change the version number of the database!

```
@Database(entities = {ShoppingListItem.class}, version = 2, exportSchema = false)
```

.

Room

It is good to knkow that the components of the UI can be modified only by the UI thread. Otherwise we get an exception.

```
private void refreshScrollView() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            List<ShoppingListItem> shoppingList;
            final String listText =
shoppingListDatabase.shoppingListDAO().getAllItems().toString();
            runOnUiThread(new Runnable(){
                @Override
                public void run() {
                    itemsView.setText(listText);
                }
            });
        }
    }).start();
}
```

Room

Solve the mistake we made at the time of instantiating the database..

Change the database class so that it can be instantiated only as a Singleton.