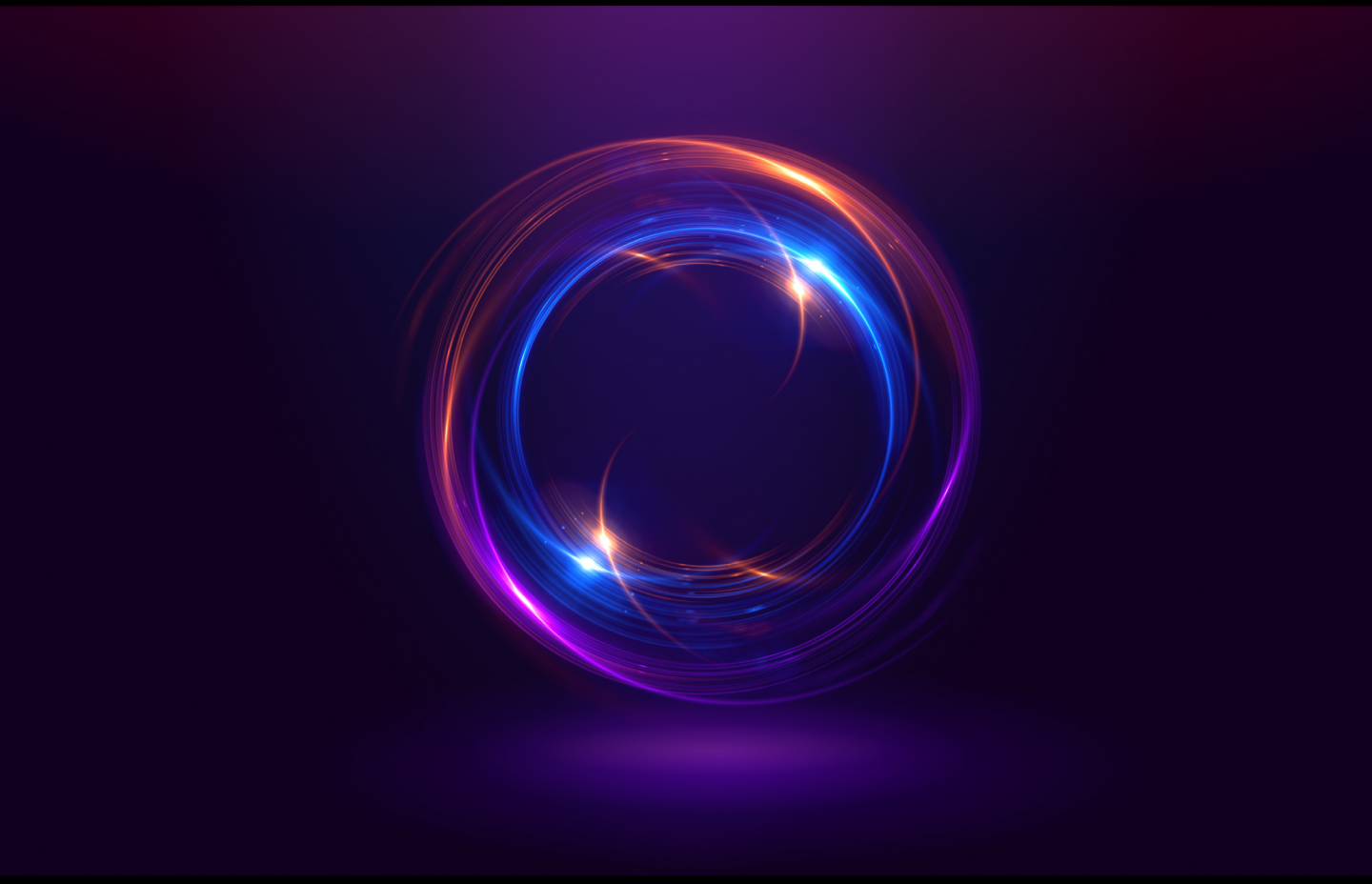
Android Studio Jellyfish Essentials



Java Edition

Neil Smyth

Payload publishing

Android Studio Jellyfish Essentials

Java Edition

Android Studio Jellyfish Essentials – Java Edition

ISBN: 978-1-951442-86-6

Rev: 1.0

https://www.payloadbooks.com

Chapter 72

72. An Android Room Database and Repository Tutorial

This chapter will combine the knowledge gained in *"The Android Room Persistence Library"* with the initial project created in the previous chapter to provide a detailed tutorial demonstrating how to implement SQLite-based database storage using the Room persistence library. In keeping with the Android architectural guidelines, the project will use a view model and repository. The tutorial will use all the elements covered in *"The Android Room Persistence Library"* including entities, a Data Access Object, a Room Database, and asynchronous database queries.

72.1 About the RoomDemo Project

The user interface layout created in the previous chapter was the first step in creating a rudimentary inventory app to store product names and quantities. When completed, the app will provide the ability to add, delete and search for database entries while displaying a scrollable list of all products currently stored in the database. This product list will update automatically as database entries are added or deleted.

72.2 Modifying the Build Configuration

Launch Android Studio and open the RoomDemo project started in the previous chapter. Before adding any new classes to the project, the first step is to add some additional libraries and plugins to the build configuration, including the Room persistence library. The first step is to add the *ksp* plugin and additional libraries to the Gradle build configuration. Using the Project tool window, locate and edit the *Gradle Scripts -> libs.versions.toml* file as follows (keeping in mind that a more recent version of the libraries may now be available):

```
[versions]
.
.
roomRuntime = "2.6.1"
fragment = "1.7.0"

[libraries]
.
.
androidx-room-compiler = { module = "androidx.room:room-compiler", version.ref =
"roomRuntime" }
androidx-room-runtime = { group = "androidx.room", name = "room-runtime",
version.ref = "roomRuntime" }
androidx-fragment = { group = "androidx.fragment", name = "fragment", version.ref
= "fragment" }
.
.
```

Click the Sync Now link to commit the changes. Next, make the following changes to the module level *build.gradle.kts* file (*app -> Gradle Scripts -> build.gradle.kts (Module :app)*):

```
.
.
dependencies {
.
.
    implementation(libs.androidx.room.runtime)
    implementation(libs.androidx.fragment)
    annotationProcessor(libs.androidx.room.compiler)
.
.
}
```

72.3 Building the Entity

This project will begin by creating the entity defining the database table schema. The entity will consist of an integer for the product id, a string column to hold the product name, and another integer value to store the quantity.

The product id column will serve as the primary key and will be auto-generated. Table 72-1 summarizes the structure of the entity:

Column	Data Type
productid	Integer / Primary Key / Auto Increment
productname	String
productquantity	Integer

Table 72-1

Add a class file for the entity by right-clicking on the *app -> java -> com.ebookfrenzy.roomdemo* entry in the Project tool window and selecting the *New -> Java Class* menu option. In the new class dialog, name the class *Product*, select the Class entry in the list, and press the keyboard return key to generate the file.

When the *Product.java* file opens in the editor, modify it so that it reads as follows:

```
package com.ebookfrenzy.roomdemo;

public class Product {

    private int id;
    private String name;
    private int quantity;

    public Product(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }
```

```
    public int getId() {
        return this.id;
    }
    public String getName() {
        return this.name;
    }

    public int getQuantity() {
        return this.quantity;
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

The class now has variables for the database table columns and matching getter and setter methods. Of course, this class does not become an entity until it has been annotated. With the class file still open in the editor, add annotations and corresponding import statements:

```
package com.ebookfrenzy.roomdemo;

import androidx.room.ColumnInfo;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "products")
public class Product {

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "productId")
    private int id;

    @ColumnInfo(name = "productName")
    private String name;
    private int quantity;
.
.
}
```

These annotations declare this as the entity for a table named *products* and assign column names for the *id* and *name* variables. The id column is also configured to be the primary key and auto-generated. Since it will not be necessary to reference the quantity column in SQL queries, a column name has not been assigned to the *quantity* variable.

72.4 Creating the Data Access Object

With the product entity defined, the next step is to create the DAO interface. Referring again to the Project tool window, right-click on the *app -> java -> com.ebookfrenzy.roomdemo* entry and select the *New -> Java Class* menu option. In the new class dialog, enter *ProductDao* into the Name field and select *Interface* from the list as highlighted in Figure 72-1:

Figure 72-1

Press the Return key to generate the new interface and, with the *ProductDao.java* file loaded into the code editor, make the following changes:

```
package com.ebookfrenzy.roomdemo;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.Query;

import java.util.List;

@Dao
public interface ProductDao {

    @Insert
    void insertProduct(Product product);

    @Query("SELECT * FROM products WHERE productName = :name")
    List<Product> findProduct(String name);

    @Query("DELETE FROM products WHERE productName = :name")
    void deleteProduct(String name);

    @Query("SELECT * FROM products")
    LiveData<List<Product>> getAllProducts();
}
```

The DAO implements methods to insert, find and delete records from the products database. The insertion method is passed a Product entity object containing the data to be stored, while the methods to find and delete records are passed a string containing the name of the product on which to perform the operation. The *getAllProducts()* method returns a LiveData object containing all of the records within the database. This method will be used to keep the RecyclerView product list in the user interface layout synchronized with the database.

72.5 Adding the Room Database

The last task before adding the repository to the project is implementing the Room Database instance. Add a new class to the project named *ProductRoomDatabase*, this time with the *Class* option selected.

Once the file has been generated, modify it as follows using the steps outlined in the "*The Android Room Persistence Library*" chapter:

```
package com.ebookfrenzy.roomdemo;

import android.content.Context;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;

@Database(entities = {Product.class}, version = 1)
public abstract class ProductRoomDatabase extends RoomDatabase {

    public abstract ProductDao productDao();
    private static ProductRoomDatabase INSTANCE;

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

72.6 Adding the Repository

Add a new class named *ProductRepository* to the project, with the *Class* option selected.

The repository class will be responsible for interacting with the Room database on behalf of the ViewModel. It must provide methods that use the DAO to insert, delete, and query product records. Except for the *getAllProducts()* DAO method (which returns a LiveData object), these database operations must be performed on separate threads from the main thread.

Remaining within the *ProductRepository.java* file, add the code for a handler to return the search results to the

repository thread:

```
package com.ebookfrenzy.roomdemo;

import android.os.Handler;
import android.os.Looper;
import android.os.Message;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import androidx.lifecycle.MutableLiveData;
import java.util.List;

public class ProductRepository {

    private final MutableLiveData<List<Product>> searchResults =
            new MutableLiveData<>();
            private List<Product> results;

    Handler handler = new Handler(Looper.getMainLooper()) {
        @Override public void handleMessage(Message msg) {
            searchResults.setValue(results);
        }
    };
}
```

The above declares a MutableLiveData variable named *searchResults* into which the results of a search operation are stored whenever an asynchronous search task completes (later in the tutorial, an observer within the ViewModel will monitor this live data object).

The repository class must now provide some methods the ViewModel can call to initiate these operations. However, the repository needs to obtain the DAO reference via a ProductRoomDatabase instance to do this. Add a constructor method to the ProductRepository class to perform these tasks:

```
.
.
import android.app.Application;
.
.
public class ProductRepository {

    private final MutableLiveData<List<Product>> searchResults =
                                                new MutableLiveData<>();
    private List<Product> results;
    private final ProductDao productDao;

    public ProductRepository(Application application) {
```

```
        ProductRoomDatabase db;
        db = ProductRoomDatabase.getDatabase(application);
        productDao = db.productDao();
    }
.
.
```

With a reference to DAO stored, the methods are ready to be added to the ProductRepository class file as follows:

```
public void insertProduct(Product newproduct) {
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(() -> productDao.insertProduct(newproduct));
    executor.shutdown();
}

public void deleteProduct(String name) {
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(() -> productDao.deleteProduct(name));
    executor.shutdown();
}

public void findProduct(String name) {
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(() -> {
        results = productDao.findProduct(name);
        handler.sendEmptyMessage(0);
    });
    executor.shutdown();
}
```

In the cases of the insertion and deletion methods, the appropriate new threads are created and used to perform the corresponding database operation. In the case of the *findProduct()* method, a message is sent to the handler indicating that new results are available.

One final task remains to complete the repository class. The RecyclerView in the user interface layout must keep up to date with the current list of products stored in the database. The ProductDao class already includes a method named *getAllProducts()* which uses a SQL query to select all of the database records and return them wrapped in a LiveData object. The repository needs to call this method once on initialization and store the result within a LiveData object that can be observed by the ViewModel and, in turn, by the UI controller. Once this has been set up, the UI controller observer will be notified each time a change occurs to the database table, and the RecyclerView can be updated with the latest product list. Remaining within the *ProductRepository.java* file, add a LiveData variable and call to the DAO *getAllProducts()* method within the constructor:

```
.
.
import androidx.lifecycle.LiveData;
.
.
public class ProductRepository {
```

```
    private final MutableLiveData<List<Product>> searchResults =
            new MutableLiveData<>();
    private List<Product> results;
    private final LiveData<List<Product>> allProducts;
    private final ProductDao productDao;

    public ProductRepository(Application application) {
        ProductRoomDatabase db;
        db = ProductRoomDatabase.getDatabase(application);
        productDao = db.productDao();
        allProducts = productDao.getAllProducts();
    }
.
.
}
```

To complete the repository, add methods that the ViewModel can call to obtain references to the *allProducts* and *searchResults* live data objects:

```
public LiveData<List<Product>> getAllProducts() {
    return allProducts;
}

public MutableLiveData<List<Product>> getSearchResults() {
    return searchResults;
}
```

72.7 Adding the ViewModel

The ViewModel is responsible for creating an instance of the repository and providing methods, and LiveData objects that the UI controller can utilize to keep the user interface synchronized with the underlying database. As implemented in *ProductRepository.java*, the repository constructor requires access to the application context to get a Room Database instance. To make the application context accessible within the ViewModel so it can be passed to the repository, the ViewModel needs to subclass AndroidViewModel instead of ViewModel.

Begin by locating the *com.ebookfrenzy.viewmodeldemo* entry in the Project tool window, right-clicking it, and selecting the *New -> Java Class* menu option. Next, name the new class MainViewModel and press the keyboard Enter key. Finally, edit the new class file to change the class to extend AndroidViewModel and implement the default constructor:

```
package com.ebookfrenzy.roomdemo.ui.main;

import android.app.Application;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import com.ebookfrenzy.roomdemo.Product;
import com.ebookfrenzy.roomdemo.ProductRepository;
import java.util.List;
```