ROOM DATABASE TUTORIAL

SQLITE	ROOM
• Deal with raw queries	• No raw queries
• No compile time verification	• Compile-time checks of SQLite statements
• Lots of boilerplate code to convert between sql queries and java data objects	• Maps database object and java objects without boilerplate code
• Sqlite api are low-level so more time, more effort to build apps	• Room when used with ViewModel and LiveData makes it easy

What is room?

Database layer on top of SQLite

Provides an abstract layer over SQLite to allow fluent database access

Object relation mapping(orm) library

Easy caching of relevant pieces of data

3 major components:

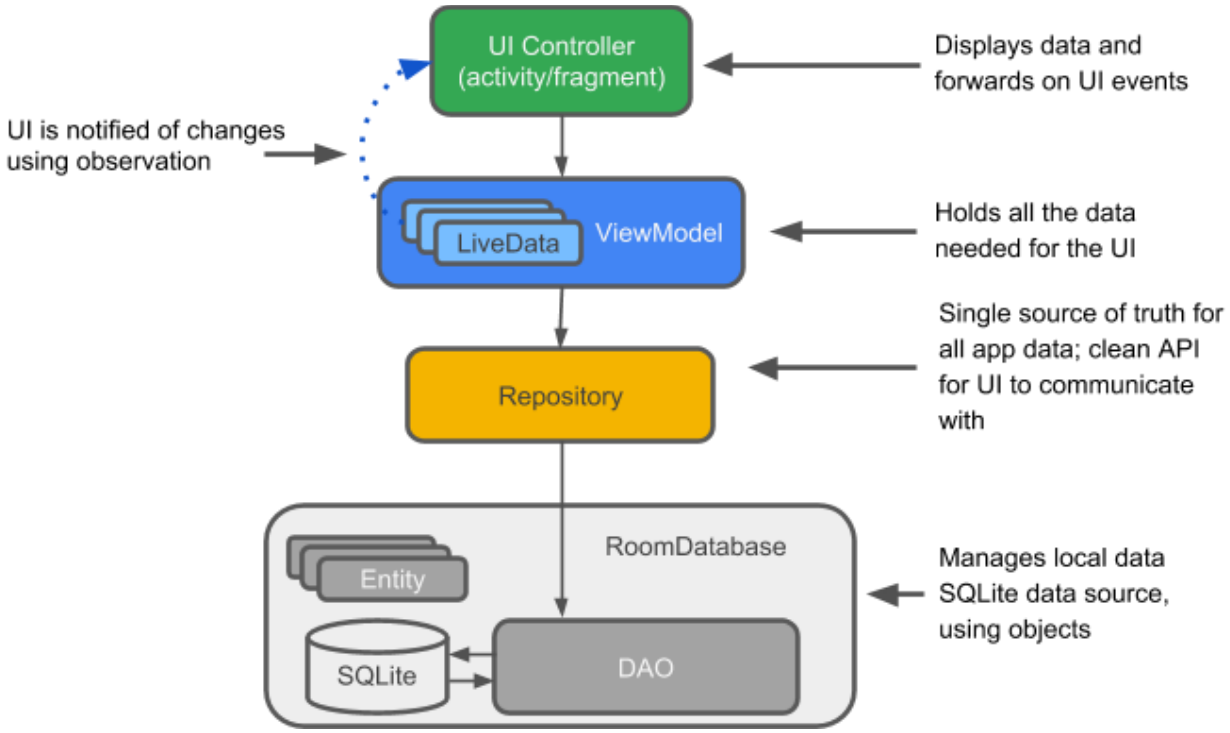
1) Database class: contains database holder and serves as the main access point in the app
2) Entity: table in your database
3) Data access object: to access your app's data, you work with Daos. Forms the main component of room. You can specify SQL queries and associate them with method calls.

ViewModel – holding and preparing the data for the UI so there is no direct communication with the room. It holds data in a way that survives configuration changes.

Repository – creates a clean API for the view model so that the view model doesn't have to know how the data is being generated or retrieved. It manages query threads and allows you to use multiple backends.

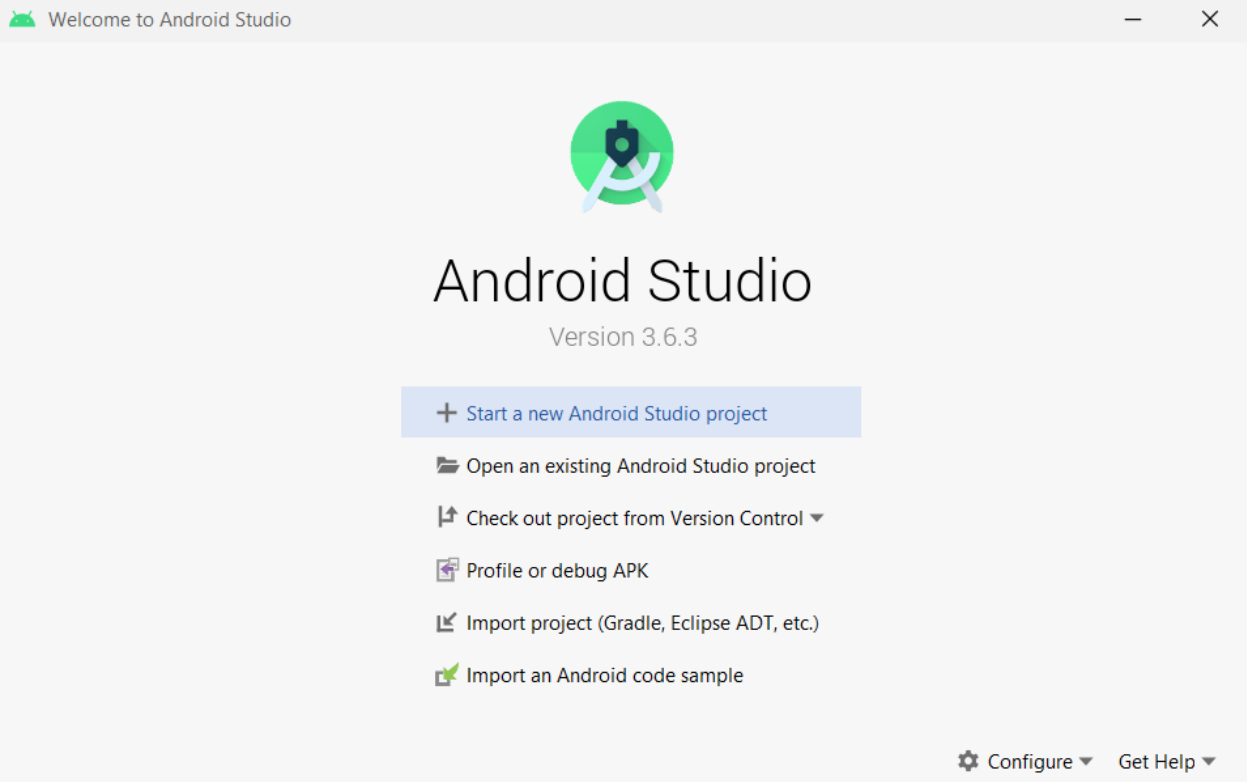
LiveData – hold any type of data and observed by UI observer for any changes. So instead of restarting the app, the changes are displayed immediately. It's in the ViewModel.

Below is the room Architecture:

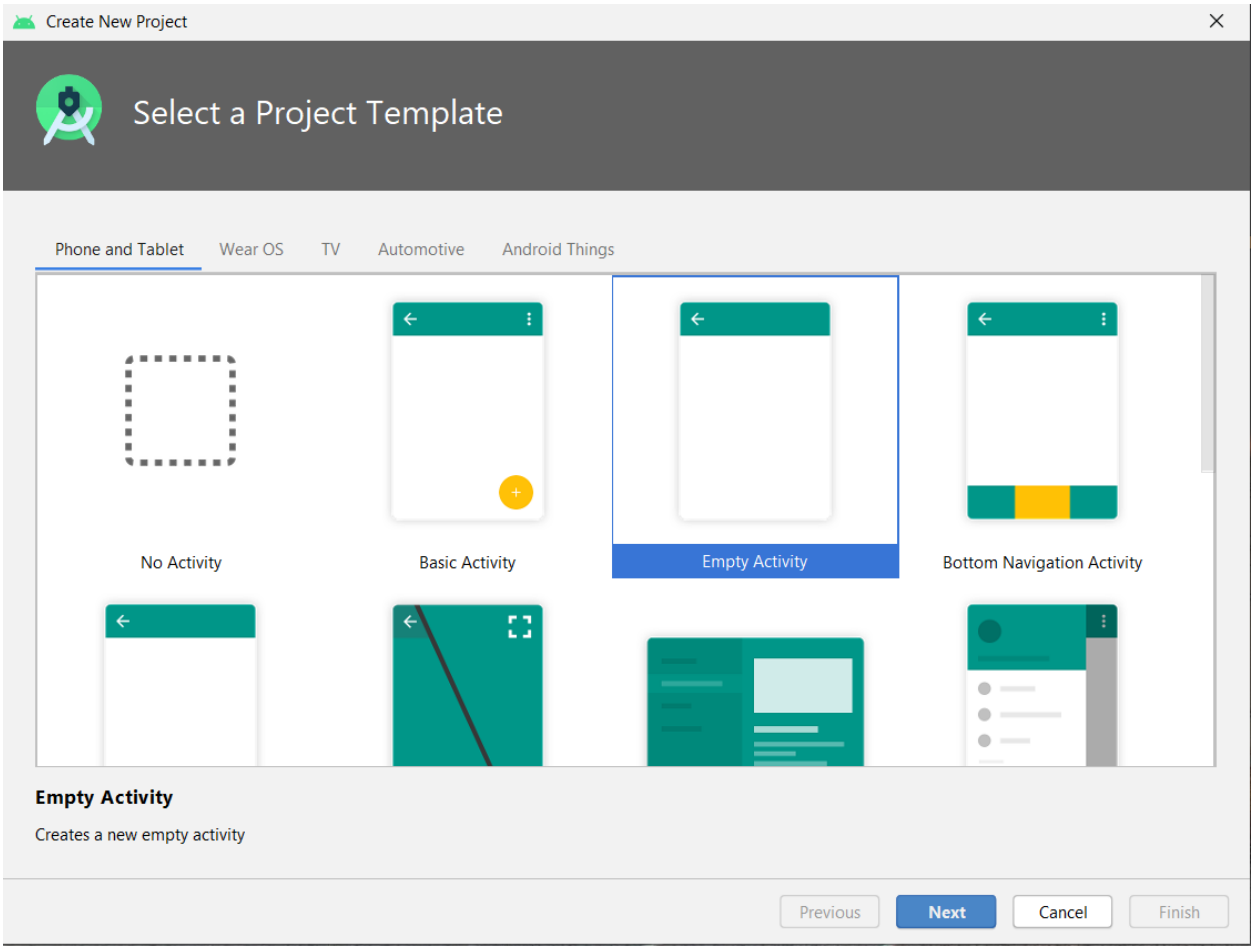


Create a new project by following the steps below:

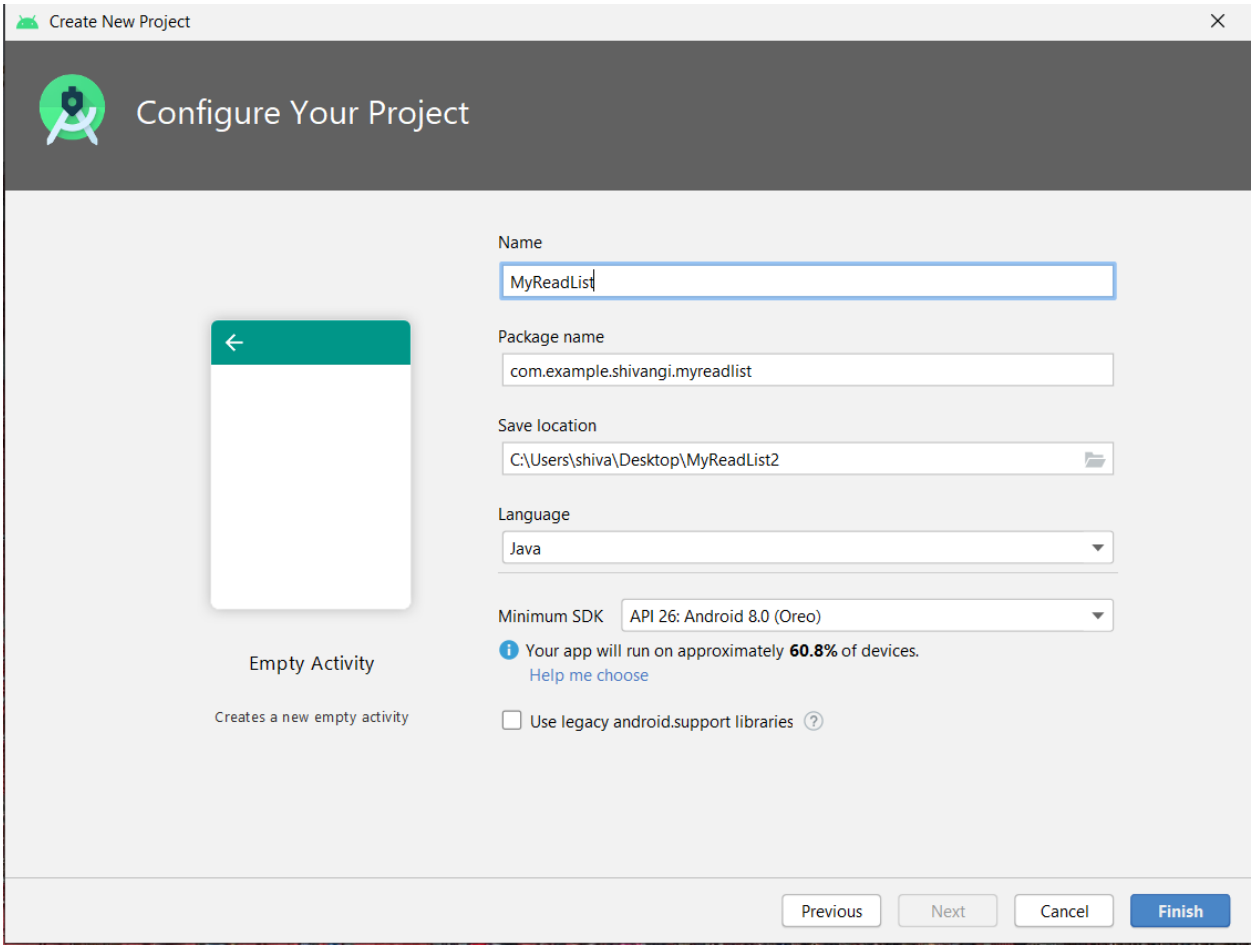
Select start a new Android Studio Project.

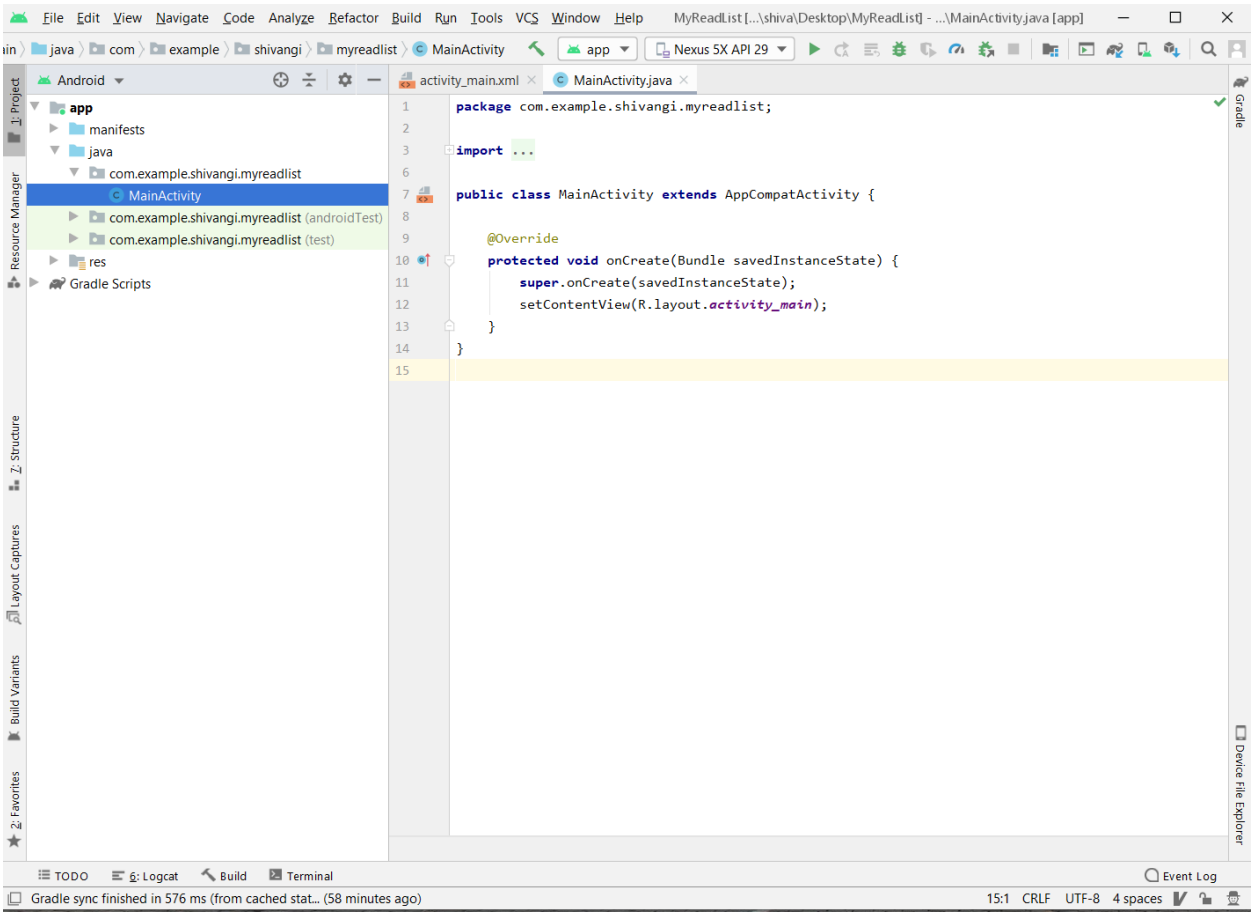

Welcome to Android Studio
Android Studio
Version 3.6.3
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

Select the Empty Activity project template.



Name the project MyReadList and click finish.




File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyReadList [...\shiva\Desktop\MyReadList] - ...\MainActivity.java [app]
java com example shivangi myreadlist MainActivity
app
Nexus 5X API 29
Android
activity_main.xml
MainActivity.java
app
manifests
java
com.example.shivangi.myreadlist
MainActivity
com.example.shivangi.myreadlist (androidTest)
com.example.shivangi.myreadlist (test)
res
Gradle Scripts
package com.example.shivangi.myreadlist;
import ...
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
}
1: Project
Resource Manager
Z: Structure
Layout Captures
Build Variants
2: Favorites
Gradle
Device File Explorer
TODO
6: Logcat
Build
Terminal
Event Log
Gradle sync finished in 576 ms (from cached stat... (58 minutes ago)
15:1 CRLF UTF-8 4 spaces

We declare strings and dimens in advance so that when we create our layout in the activity_main.xml file, we can add the strings and dimens to the object so we don't have to hardcode it. This is quite efficient as you can refer to the items in that file multiple times for example in the dimens.xml file, the declared dimensions can be used multiple times in the same layout file or in different layout files.

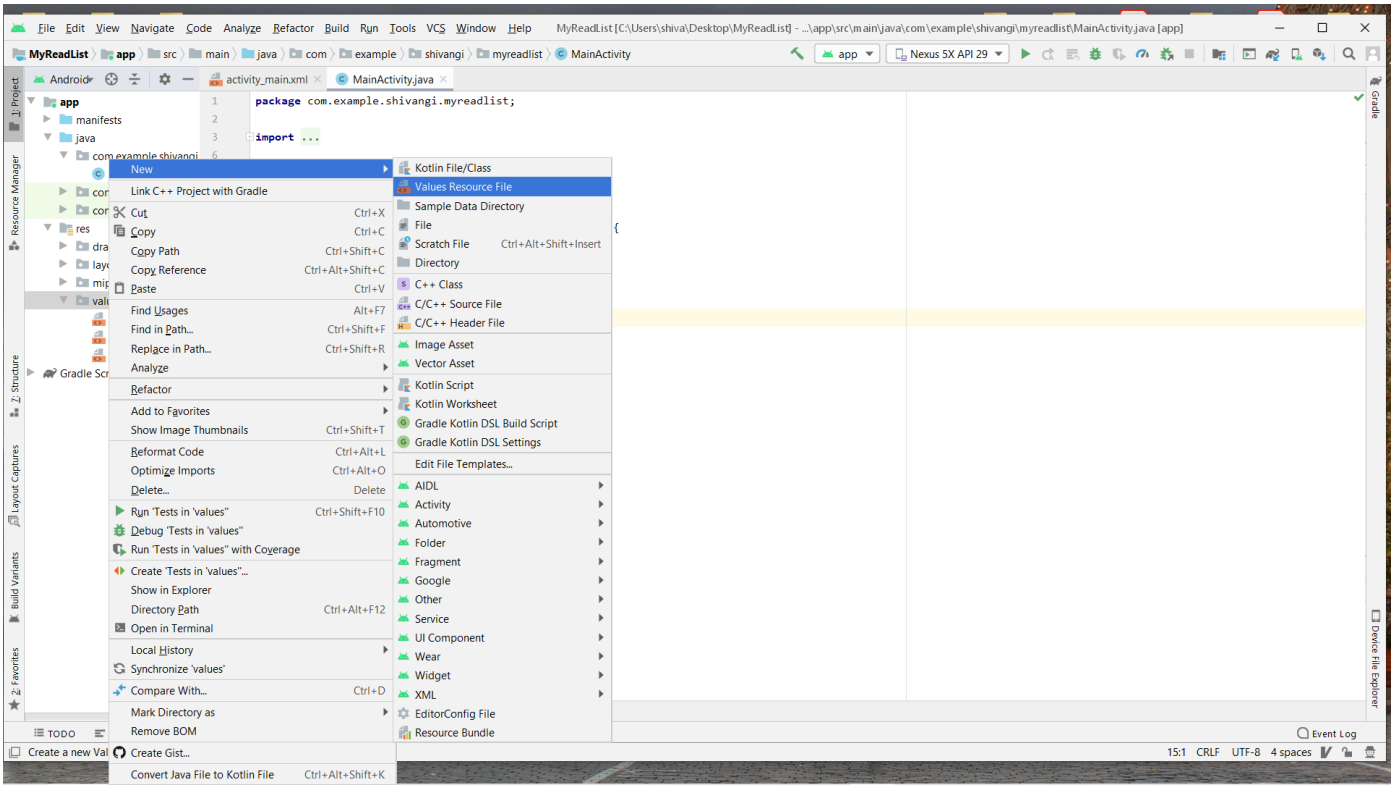
Add the following strings to the strings.xml file:

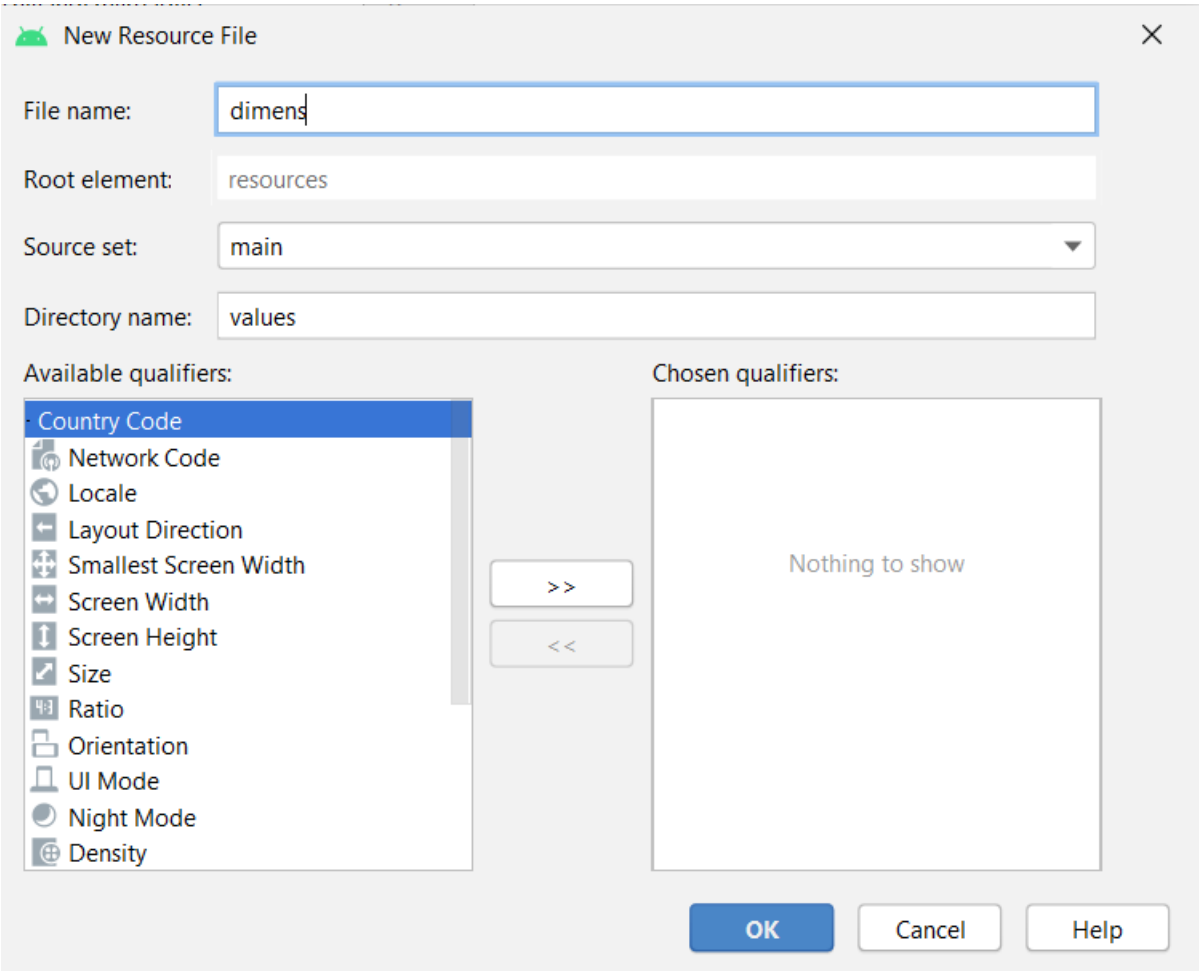
strings.xml

Edit translations for all locales in the translations editor. Open editor Hide notification

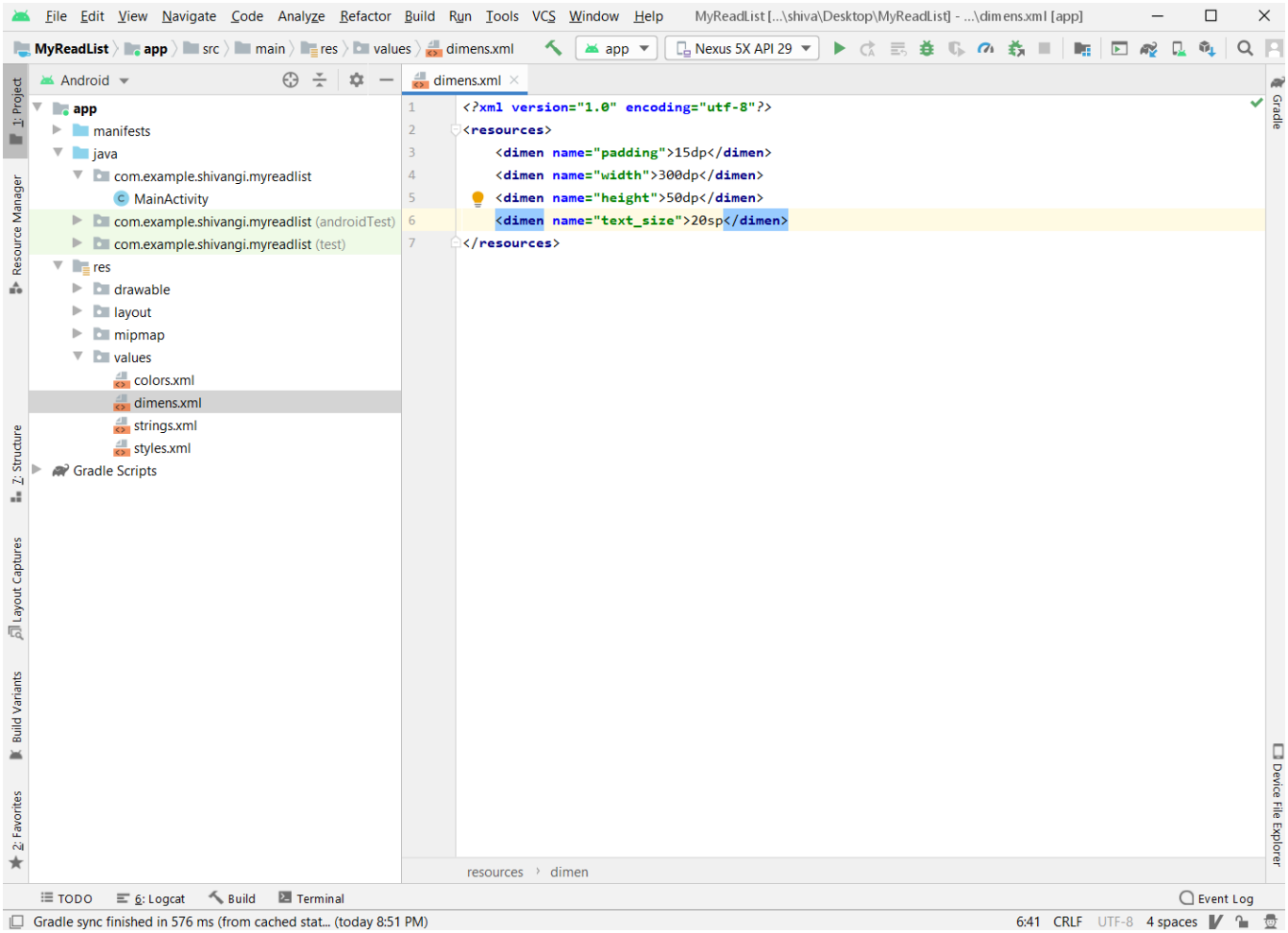
```xml
<resources>
    <string name="app_name">MyReadList</string>
    <string name="myListBtn">MY LIST</string>
    <string name="searchBtn">SEARCH</string>
    <string name="addBtn">ADD</string>
    <string name="searchHint">Search Book</string>
    <string name="bookNameHint">Enter Book Title</string>
    <string name="authorHint">Enter Author</string>
    <string-array name="genres">
        <item>Action</item>
        <item>Adventure</item>
        <item>Comic</item>
        <item>Educational</item>
        <item>Historical</item>
        <item>Horror</item>
        <item>Romance</item>
        <item>Sci-Fi</item>
        <item>Thriller</item>
    </string-array>
    <string name="queueRadioList">Add to Queue</string>
    <string name="readingRadioList">Reading</string>
    <string name="completeRadioList">Completed</string>
    <string name="categoryText">Select a category</string>
    <string name="readingStage">Select the progress</string>
    <string name="authorTextView">Author: </string>
    <string name="categoryTextView">Category: </string>
</resources>
```

Add a new xml file called dimens by right-clicking on the values directory:

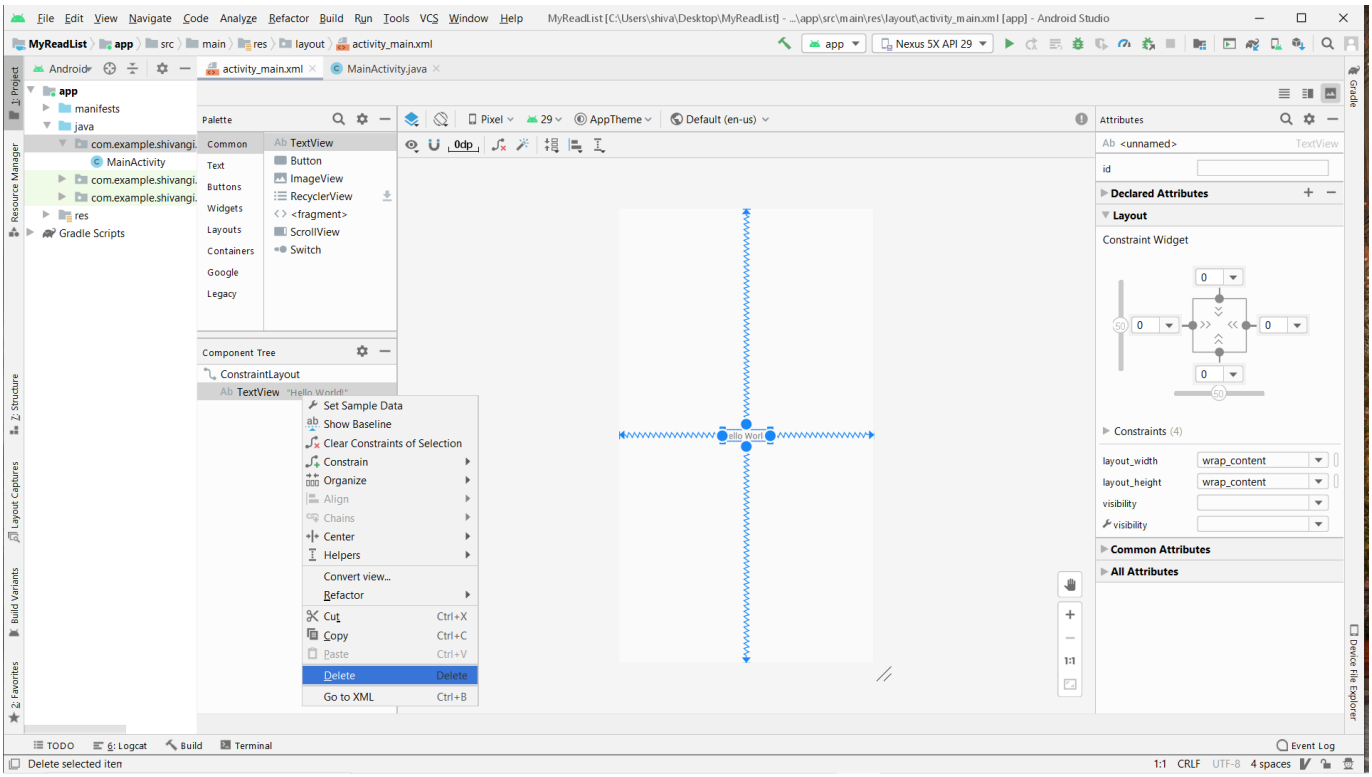



New Resource File
File name:
dimens
Root element:
resources
Source set:
main
Directory name:
values
Available qualifiers:
Country Code
Network Code
Locale
Layout Direction
Smallest Screen Width
Screen Width
Screen Height
Size
Ratio
Orientation
UI Mode
Night Mode
Density
>>
<<
Chosen qualifiers:
Nothing to show
OK
Cancel
Help

Add the following code in the dimens.xml file:

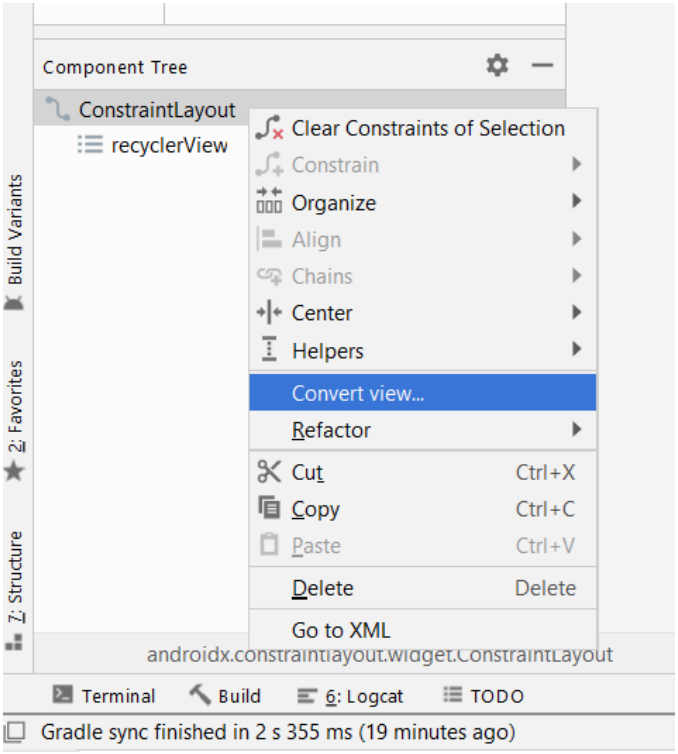


Delete the TextView in the activity_main.xml file by right-clicking on the TextView under the Component tree:

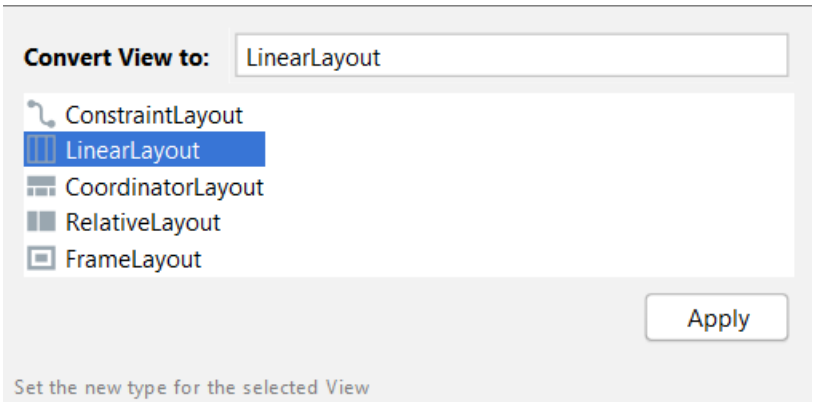


The layout for our activity_main.xml file will contain a recycler view that will reuse a layout for each item.

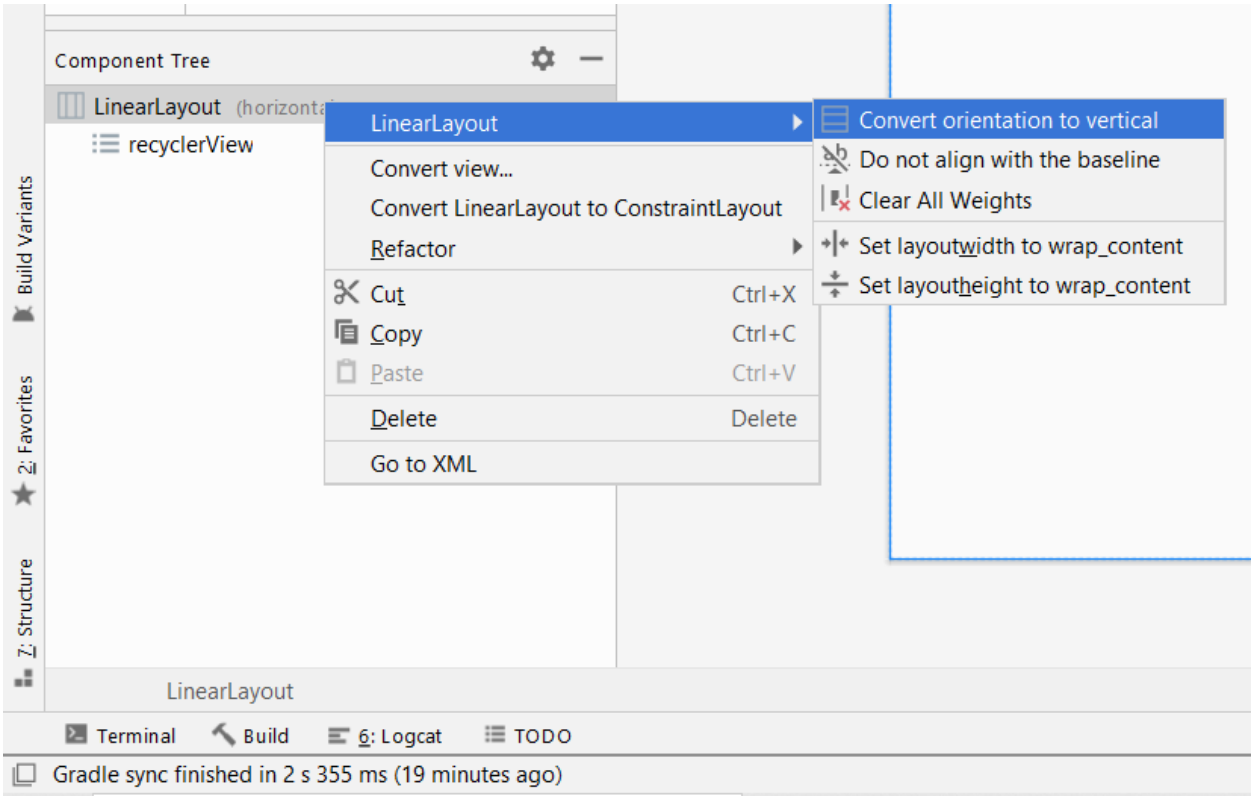
First, we will convert the layout to a vertical Linear Layout. To do this, right click on the constraint layout under the component tree and select convert view.



Select LinearLayout from the dialog box that appears.



Once again right-click on the LinearLayout(horizontal) under the component tree. Go to **LinearLayout** > **Convert orientation to vertical**.

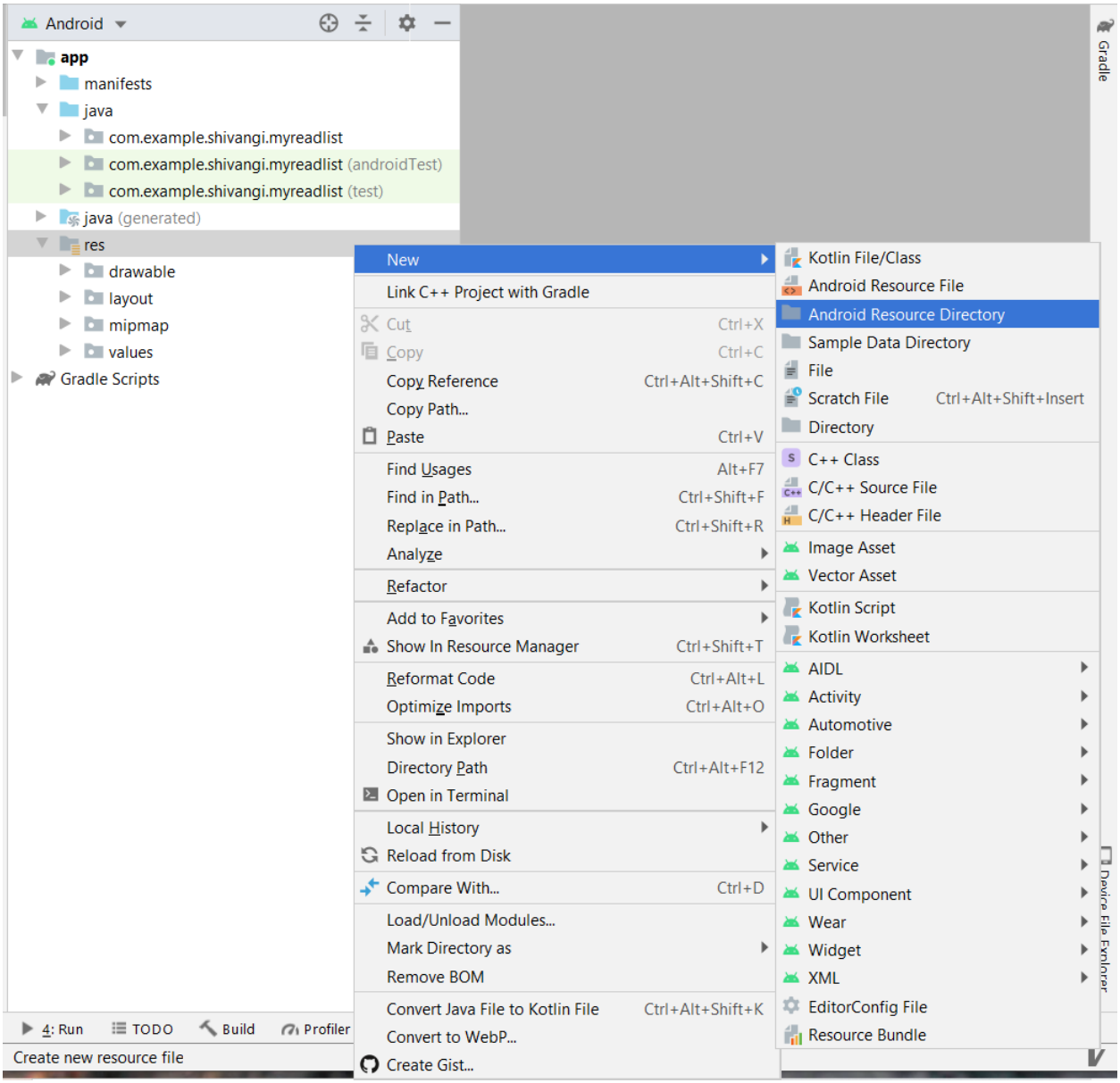


Add the following code to in the activity_main.xml file:

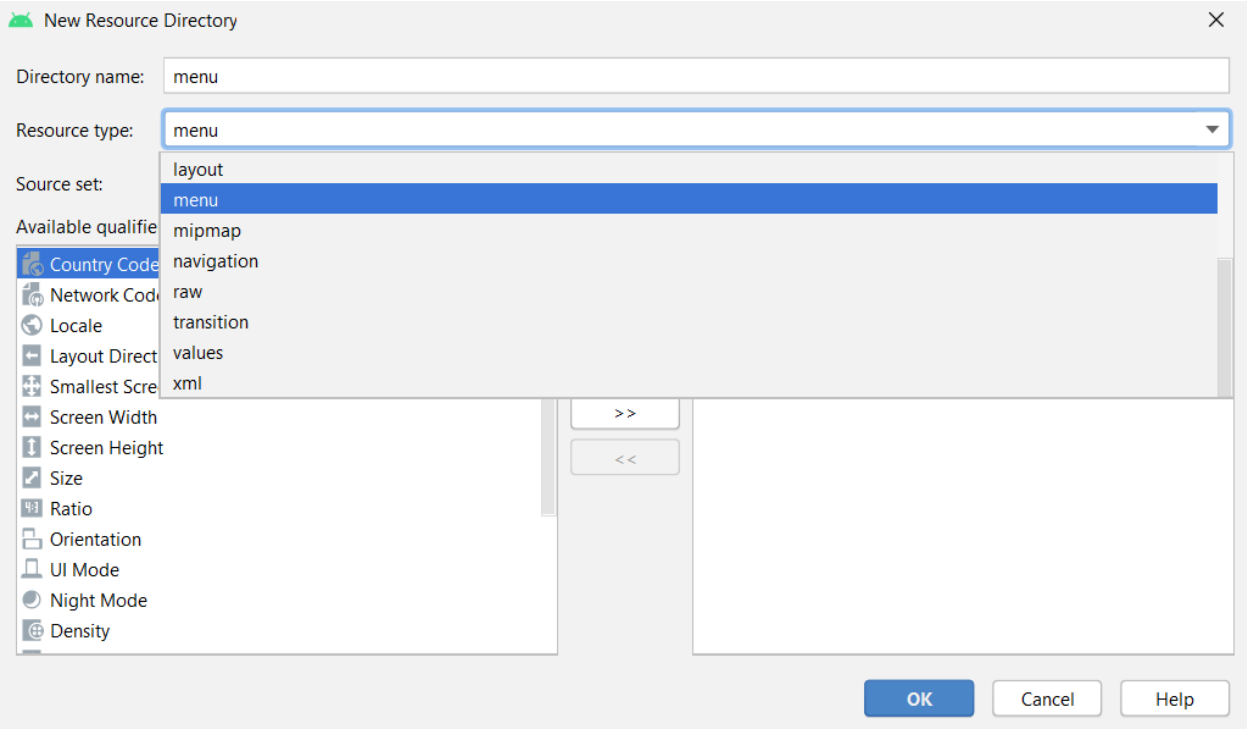
MainActivity.java | activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="10dp"/>

</LinearLayout>
```

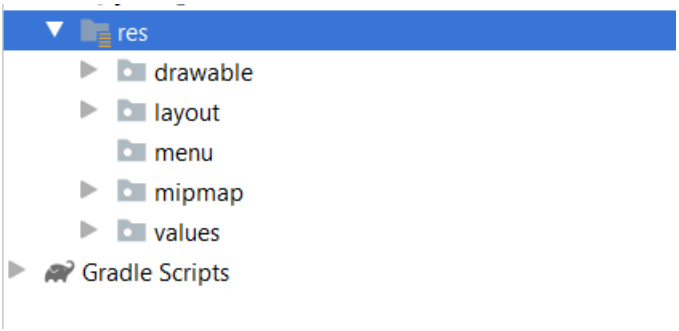
Now let's create a menu for the add, update and deleteAll operations. To add a menu, we first have to create a new directory in the res folder that has its resource as menu. Right click on the res folder, select **New** > **Android Resource Directory**.


Android
app
manifests
java
com.example.shivangi.myreadlist
com.example.shivangi.myreadlist (androidTest)
com.example.shivangi.myreadlist (test)
java (generated)
res
drawable
layout
mipmap
values
Gradle Scripts
New
Link C++ Project with Gradle
Cut Ctrl+X
Copy Ctrl+C
Copy Reference Ctrl+Alt+Shift+C
Copy Path...
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show In Resource Manager Ctrl+Shift+T
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Show in Explorer
Directory Path Ctrl+Alt+F12
Open in Terminal
Local History
Reload from Disk
Compare With... Ctrl+D
Load/Unload Modules...
Mark Directory as
Remove BOM
Convert Java File to Kotlin File Ctrl+Alt+Shift+K
Convert to WebP...
Create Gist...
Kotlin File/Class
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Directory
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Kotlin Script
Kotlin Worksheet
AIDL
Activity
Automotive
Folder
Fragment
Google
Other
Service
UI Component
Wear
Widget
XML
EditorConfig File
Resource Bundle
4: Run
TODO
Build
Profiler
Create new resource file
Gradle
Device File Explorer

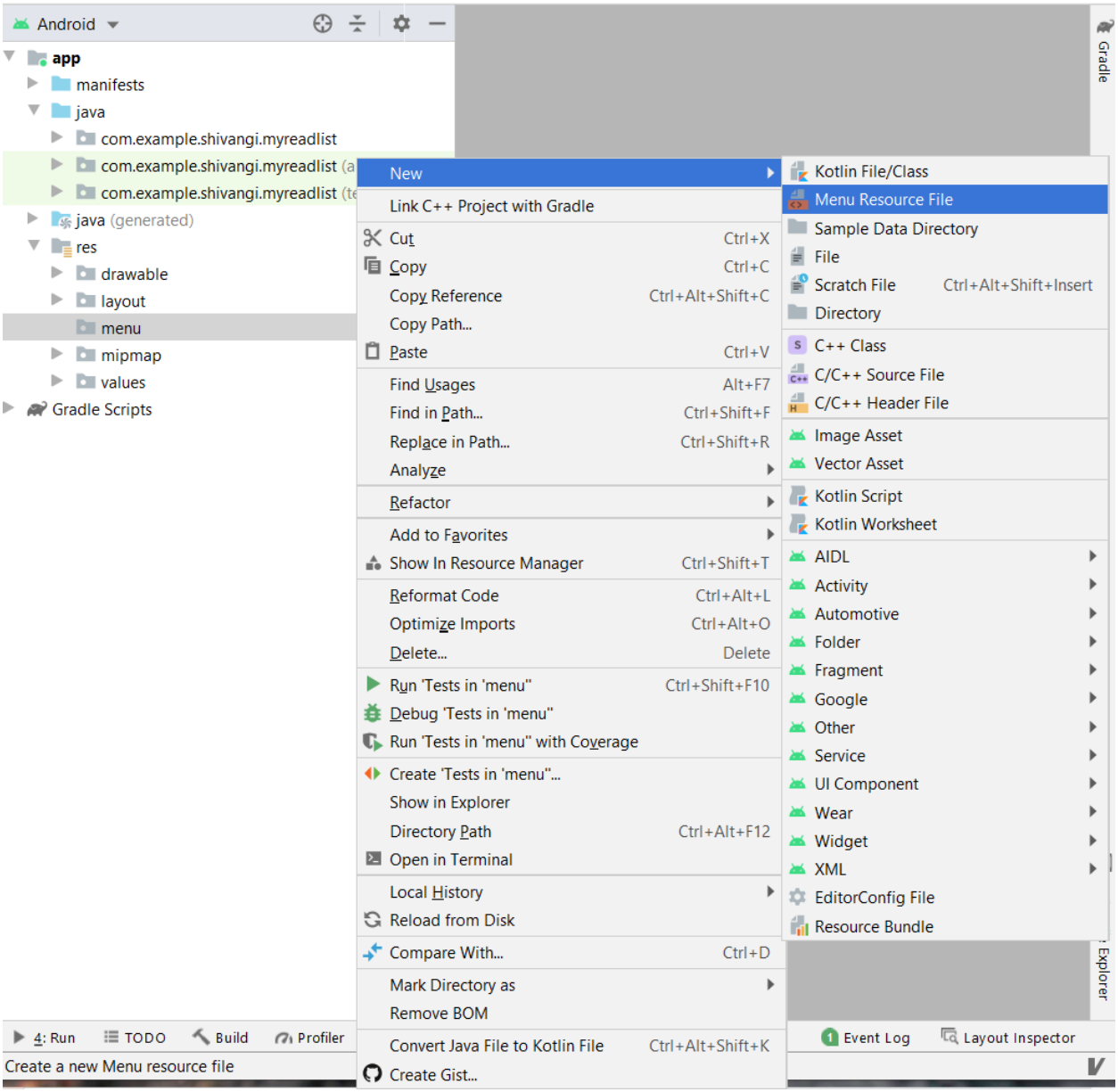
In the dialog box that appears, select the resource type as menu.



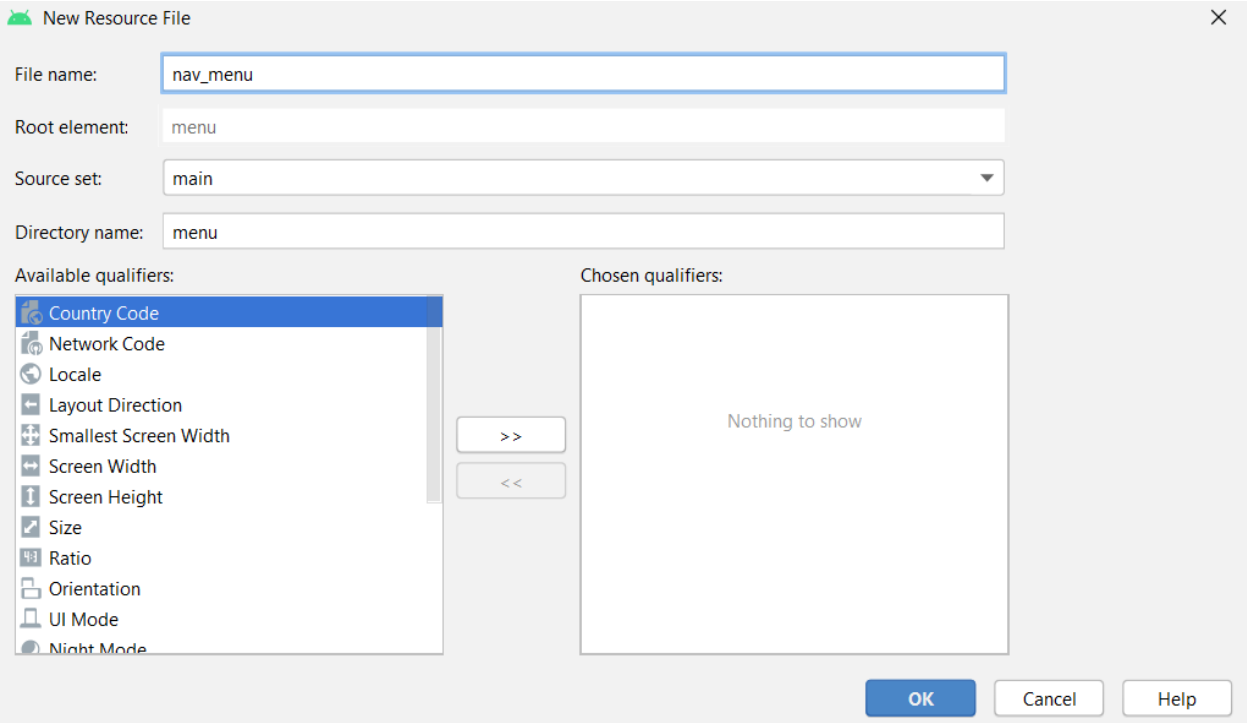
The menu folder will appear under your res folder.

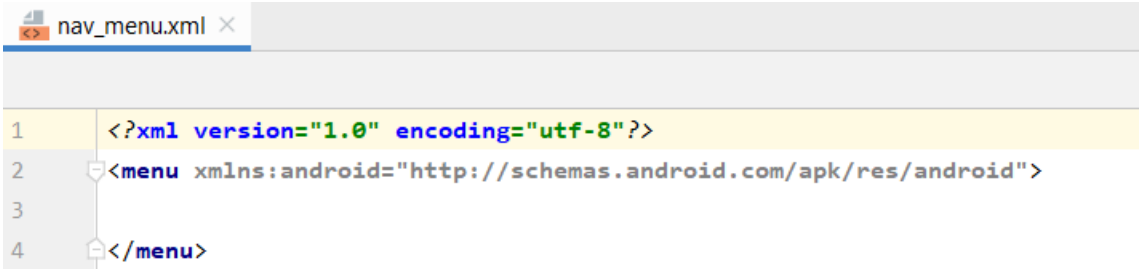


Now let’s create a new menu resource. Right-click on the menu and select **New** > **Menu Resource File**.

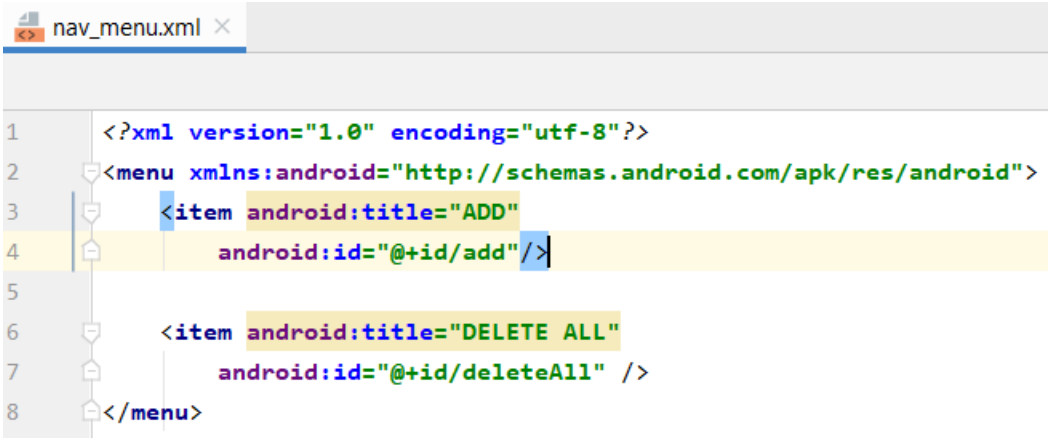


Name the file nav_menu ad click OK.

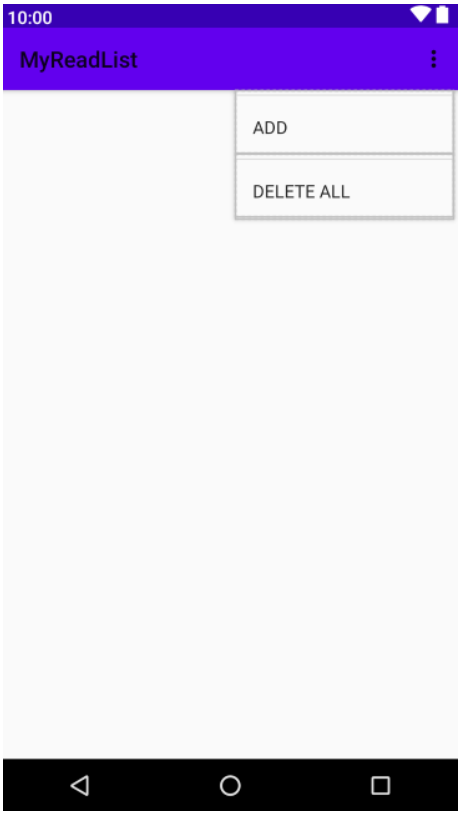




Let's add the following code in the nav_menu.xml file.

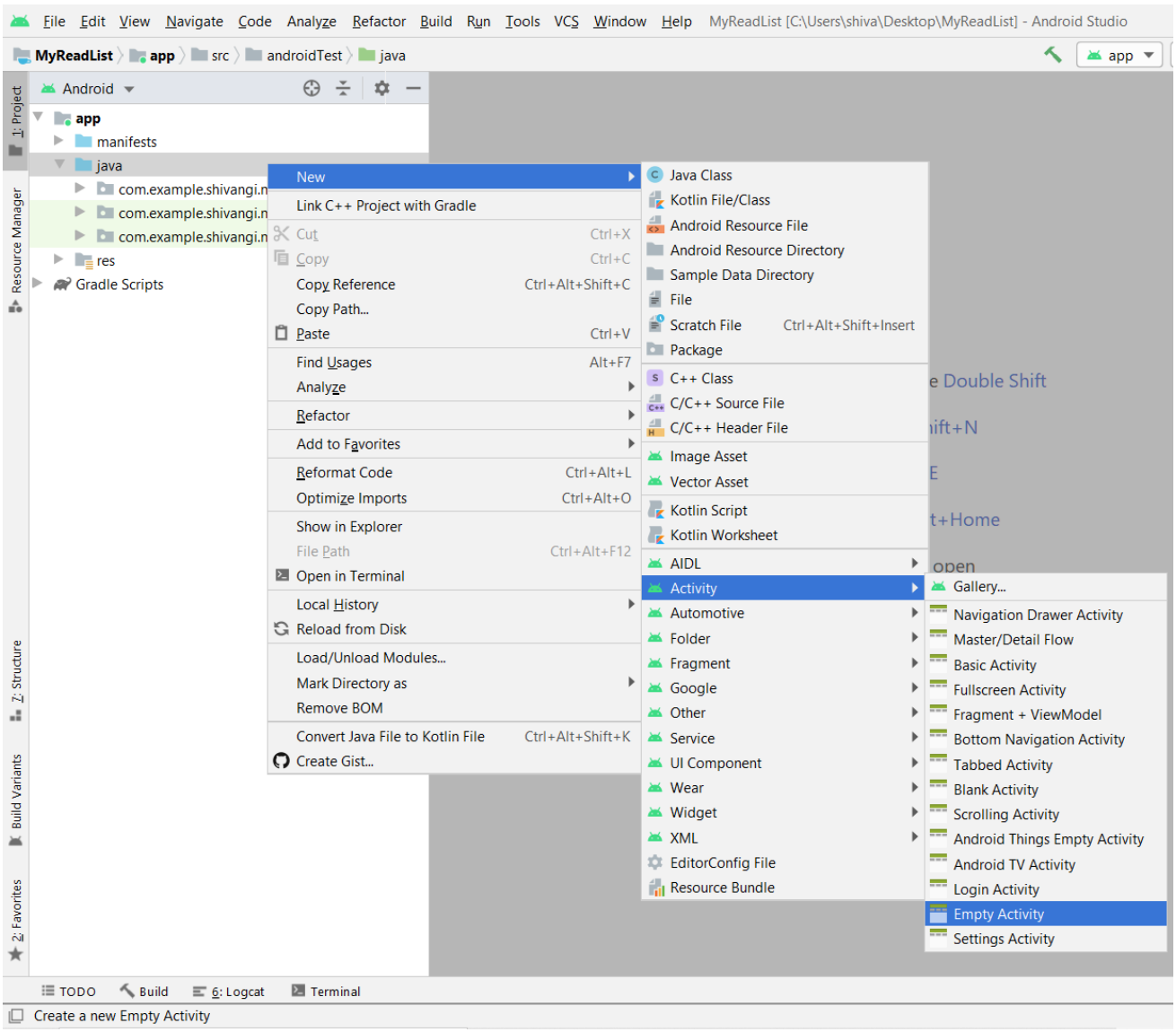


Your design view will look as follows:

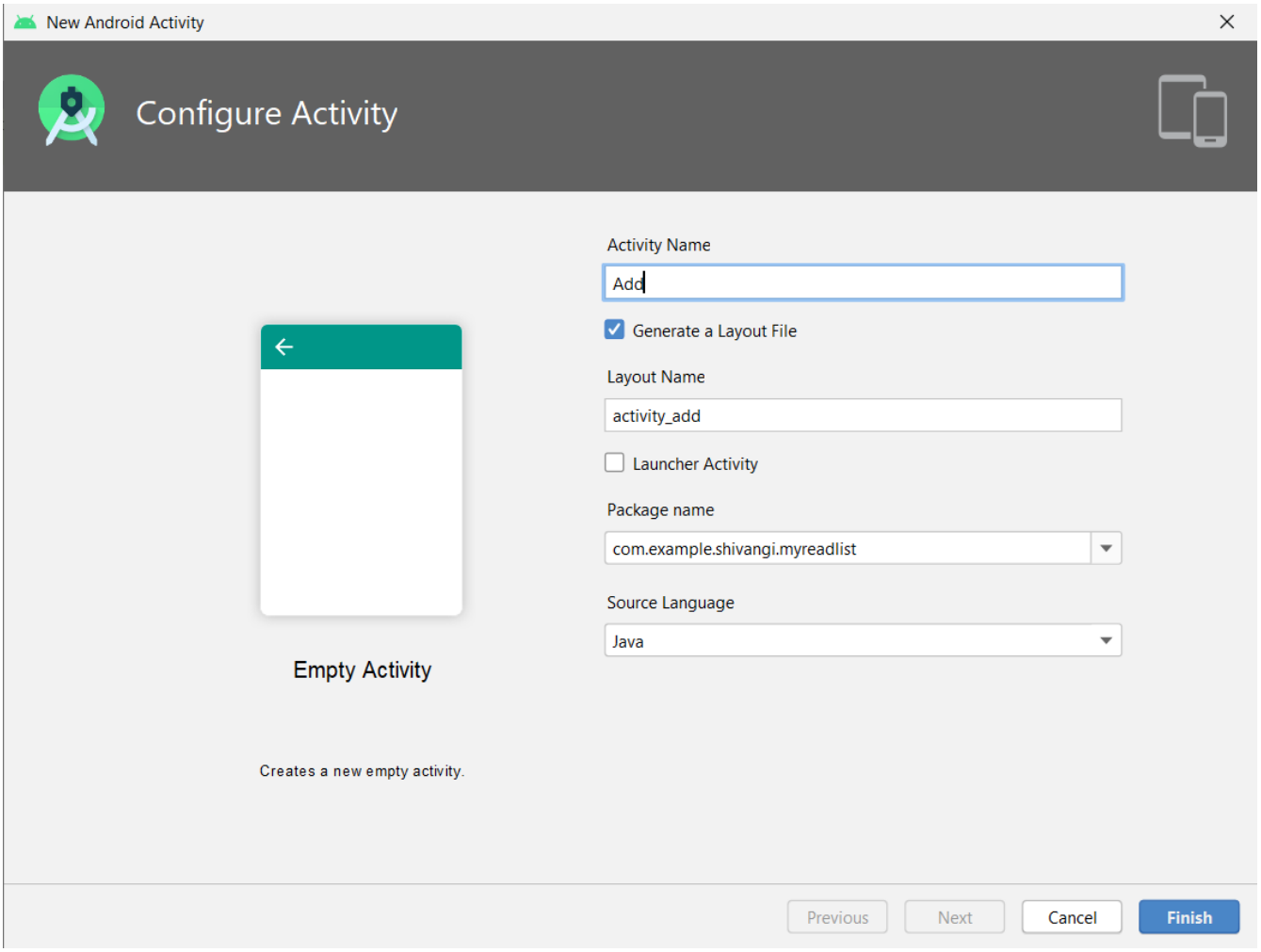


Now we will create an empty activity for our Add item in the menu created above.

To create a new activity, right- click on the java folder, go to **New** > **Activity** > **Empty Activity**.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help MyReadList [C:\Users\shiva\Desktop\MyReadList] - Android Studio
MyReadList app src androidTest java
app
Android
app
manifests
java
com.example.shivangi.n
com.example.shivangi.n
com.example.shivangi.n
res
Gradle Scripts
New
Link C++ Project with Gradle
Cut Ctrl+X
Copy Ctrl+C
Copy Reference Ctrl+Alt+Shift+C
Copy Path...
Paste Ctrl+V
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Show in Explorer
File Path Ctrl+Alt+F12
Open in Terminal
Local History
Reload from Disk
Load/Unload Modules...
Mark Directory as
Remove BOM
Convert Java File to Kotlin File Ctrl+Alt+Shift+K
Create Gist...
Java Class
Kotlin File/Class
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Kotlin Script
Kotlin Worksheet
AIDL
Activity
Automotive
Folder
Fragment
Google
Other
Service
UI Component
Wear
Widget
XML
EditorConfig File
Resource Bundle
Gallery...
Navigation Drawer Activity
Master/Detail Flow
Basic Activity
Fullscreen Activity
Fragment + ViewModel
Bottom Navigation Activity
Tabbed Activity
Blank Activity
Scrolling Activity
Android Things Empty Activity
Android TV Activity
Login Activity
Empty Activity
Settings Activity
e Double Shift
ift+N
E
t+Home
open
1: Project
Resource Manager
7: Structure
Build Variants
2: Favorites
TODO
Build
6: Logcat
Terminal
Create a new Empty Activity

Name the activity ADD and click finish.



We will now create a layout to add the book title, author, category and the reading status. We will use a spinner for the category and in the entries property, we will declare the genres array we created in our strings.xml file. We will also create a radio group for the user to select the status.

Switch to code view in your activity_add.xml file and add the following code:

activity_add.xml

Code Split

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Add">

    <EditText
        android:id="@+id/bookEditText"
        android:layout_width="300dp"
        android:layout_height="0dp"
        android:layout_marginTop="43dp"
        android:layout_marginBottom="25dp"
        android:ems="10"
        android:hint="Enter Book Title"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toTopOf="@+id/authorEditText"
        app:layout_constraintEnd_toEndOf="@+id/authorEditText"
        app:layout_constraintStart_toStartOf="@+id/authorEditText"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/authorEditText"
        android:layout_width="300dp"
        android:layout_height="0dp"
        android:layout_marginBottom="32dp"
        android:ems="10"
        android:hint="Enter Author"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toTopOf="@+id/textView2"
        app:layout_constraintEnd_toEndOf="@+id/spinner"
        app:layout_constraintStart_toStartOf="@+id/textView2"
        app:layout_constraintTop_toBottomOf="@+id/bookEditText" />
```

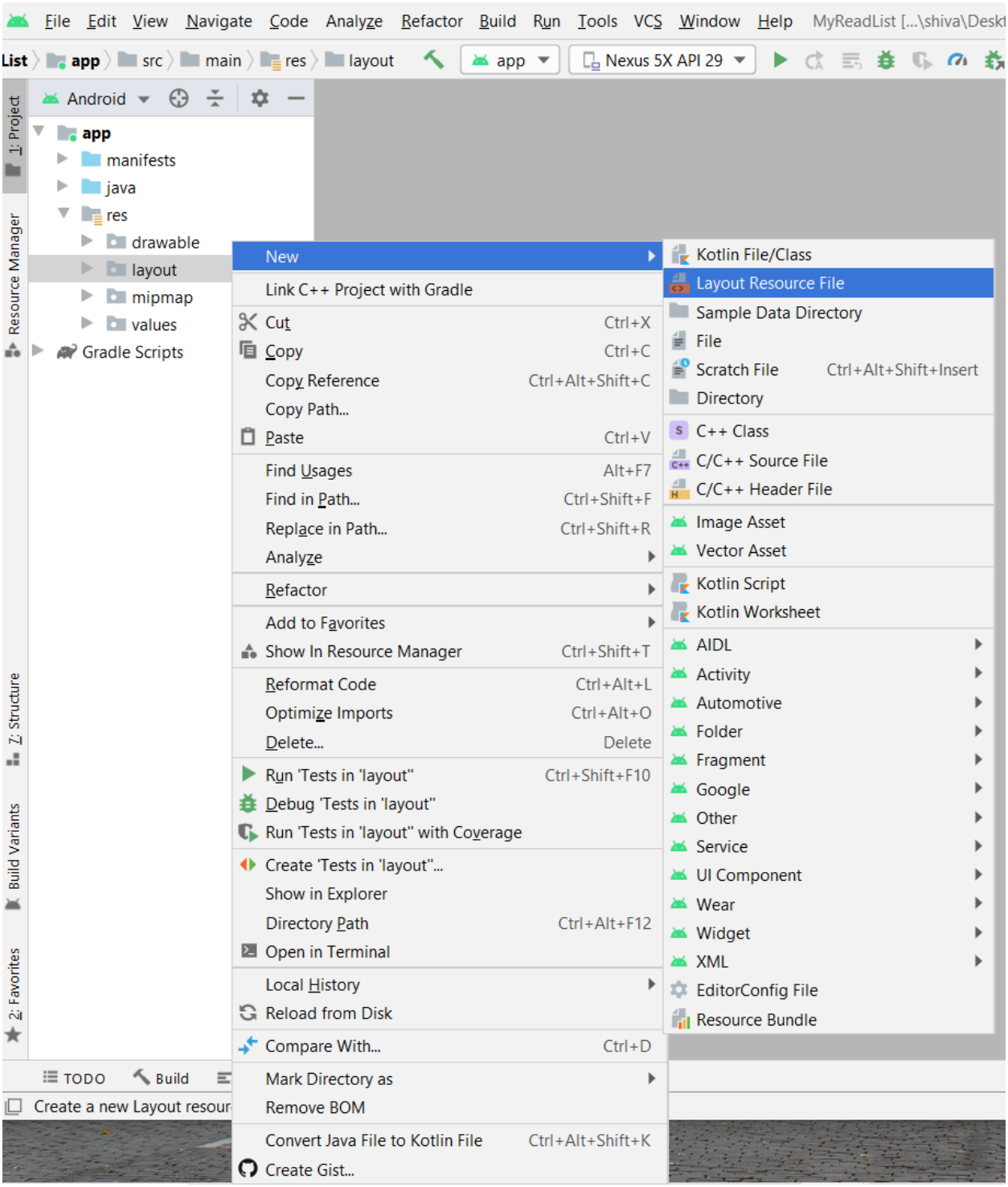
```
36      <Spinner
37          android:id="@+id/spinner"
38          android:layout_width="300dp"
39          android:layout_height="0dp"
40          android:layout_marginBottom="33dp"
41          android:entries="@array/genres"
42          app:layout_constraintBottom_toTopOf="@+id/textView"
43          app:layout_constraintEnd_toEndOf="@+id/radioGroup"
44          app:layout_constraintStart_toStartOf="@+id/textView2"
45          app:layout_constraintTop_toBottomOf="@+id/textView2" />
46
47      <RadioGroup
48          android:id="@+id/radioGroup"
49          android:layout_width="@dimen/width"
50          android:layout_height="wrap_content"
51          android:layout_marginStart="26dp"
52          android:layout_marginBottom="63dp"
53          app:layout_constraintBottom_toTopOf="@+id/btnSecondAdd"
54          app:layout_constraintStart_toStartOf="parent"
55          app:layout_constraintTop_toBottomOf="@+id/textView">
56
57          <RadioButton
58              android:id="@+id/radioQueue"
59              android:layout_width="match_parent"
60              android:layout_height="wrap_content"
61              android:layout_margin="10dp"
62              android:text="Add to Queue" />
```

```
64          <RadioButton
65              android:id="@+id/radioReading"
66              android:layout_width="match_parent"
67              android:layout_height="wrap_content"
68              android:layout_margin="10dp"
69              android:text="Reading" />
70
71          <RadioButton
72              android:id="@+id/radioCompleted"
73              android:layout_width="match_parent"
74              android:layout_height="wrap_content"
75              android:layout_margin="10dp"
76              android:text="Completed" />
77      </RadioGroup>
78
79      <TextView
80          android:id="@+id/textView"
81          android:layout_width="wrap_content"
82          android:layout_height="wrap_content"
83          android:layout_marginStart="2dp"
84          android:layout_marginBottom="17dp"
85          android:text="Select the progress"
86          android:textSize="18sp"
87          app:layout_constraintBottom_toTopOf="@+id/radioGroup"
88          app:layout_constraintStart_toStartOf="@+id/radioGroup"
89          app:layout_constraintTop_toBottomOf="@+id/spinner" />
90
```

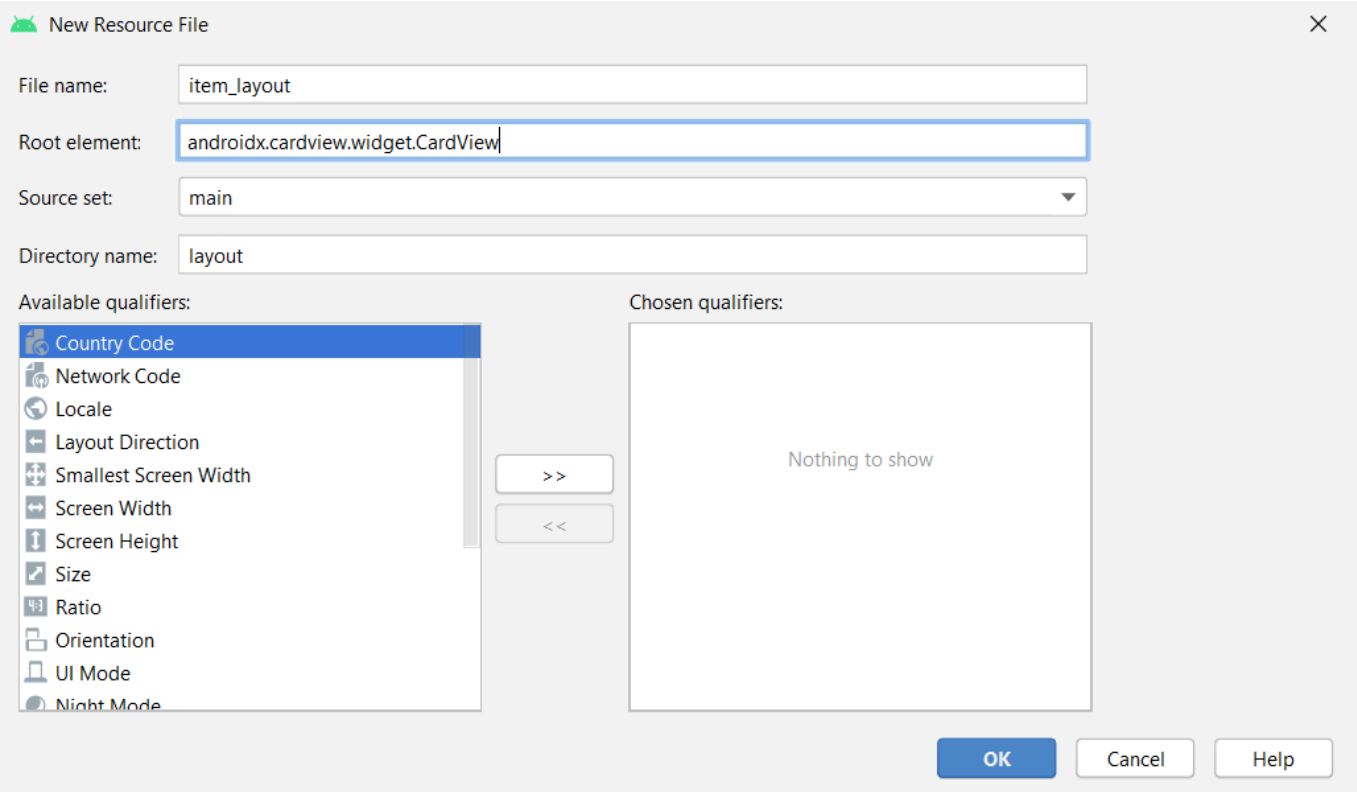
```
91      <TextView
92          android:id="@+id/textView2"
93          android:layout_width="wrap_content"
94          android:layout_height="wrap_content"
95          android:layout_marginStart="26dp"
96          android:layout_marginBottom="15dp"
97          android:text="Select a category"
98          android:textSize="18sp"
99          app:layout_constraintBottom_toTopOf="@+id/spinner"
100         app:layout_constraintStart_toStartOf="parent"
101         app:layout_constraintTop_toBottomOf="@+id/authorEditText" />
102
103     <Button
104         android:id="@+id/btnSecondAdd"
105         android:layout_width="300dp"
106         android:layout_height="0dp"
107         android:layout_marginBottom="100dp"
108         android:text="ADD"
109         app:layout_constraintBottom_toBottomOf="parent"
110         app:layout_constraintEnd_toEndOf="parent"
111         app:layout_constraintStart_toStartOf="parent"
112         app:layout_constraintTop_toBottomOf="@+id/radioGroup" />
113 </androidx.constraintlayout.widget.ConstraintLayout>
```

We will now create a new layout resource file which will be reused for each item in the list.

Right-click on the layout folder under the res folder. Select **New** > **Layout Resource File**.



Set the file name as item_layout and the root element as CardView in the dialog box that appears.



Your item_layout.xml file currently looks like this.

We will need 4 TextView to hold the data.

Add the following code to the file.

item_layout.xml

Code

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_margin="5dp"
    app:cardCornerRadius="20dp"
    android:id="@+id/cardView">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="10dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/bookName"
            android:textSize="24dp"
            android:textStyle="bold"
            android:textColor="@color/colorPrimaryDark"
            android:layout_marginBottom="10dp"/>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

```

item_layout.xml

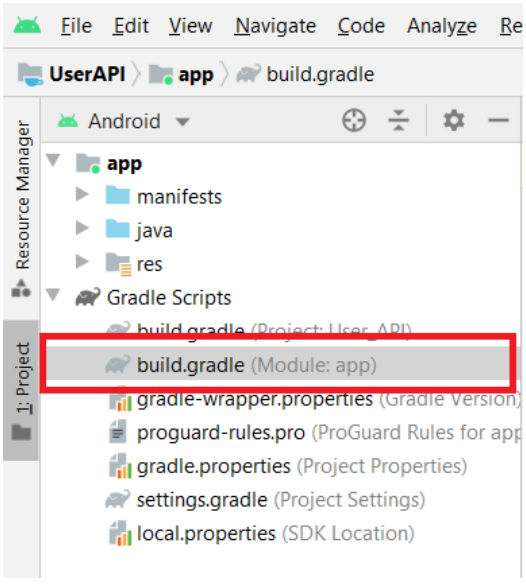
```
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Author:"
            android:textStyle="italic"
            android:layout_marginRight="5dp" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/authorName"
            android:textSize="14dp"
            android:textStyle="italic|bold"
            android:layout_marginRight="15dp"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Category:"
            android:textStyle="italic"
            android:layout_marginRight="5dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/bkCategory"
            android:textStyle="italic|bold"
            android:textSize="14dp"
            android:layout_marginBottom="10dp"/>

    </LinearLayout>
```

```
60
61          <TextView
62              android:layout_width="wrap_content"
63              android:layout_height="wrap_content"
64              android:id="@+id/bkStatus"
65              android:textStyle="bold"
66              android:textSize="14dp"/>
67
68      </LinearLayout>
69
70  </androidx.cardview.widget.CardView>
```

Now that we have created our layout files, let's add the dependencies. When creating a room, you will need room, lifecycle, UI and testing dependencies for this tutorial. The link below will provide you with the dependencies. We will also add the compileOptions to the android {} method. It is used to control different aspects of your application.

https://developer.android.com/topic/libraries/architecture/adding-components.html

Now go to the build.gradle(Module: app) file and add the dependencies under the dependencies method.



So your build.gradle file looks like this:

build.gradle (:app)

You can use the Project Structure dialog to view and edit your project configuration

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "com.example.shivangi.myreadlist"
        minSdkVersion 26
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

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.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

}
```

Add the following dependencies:

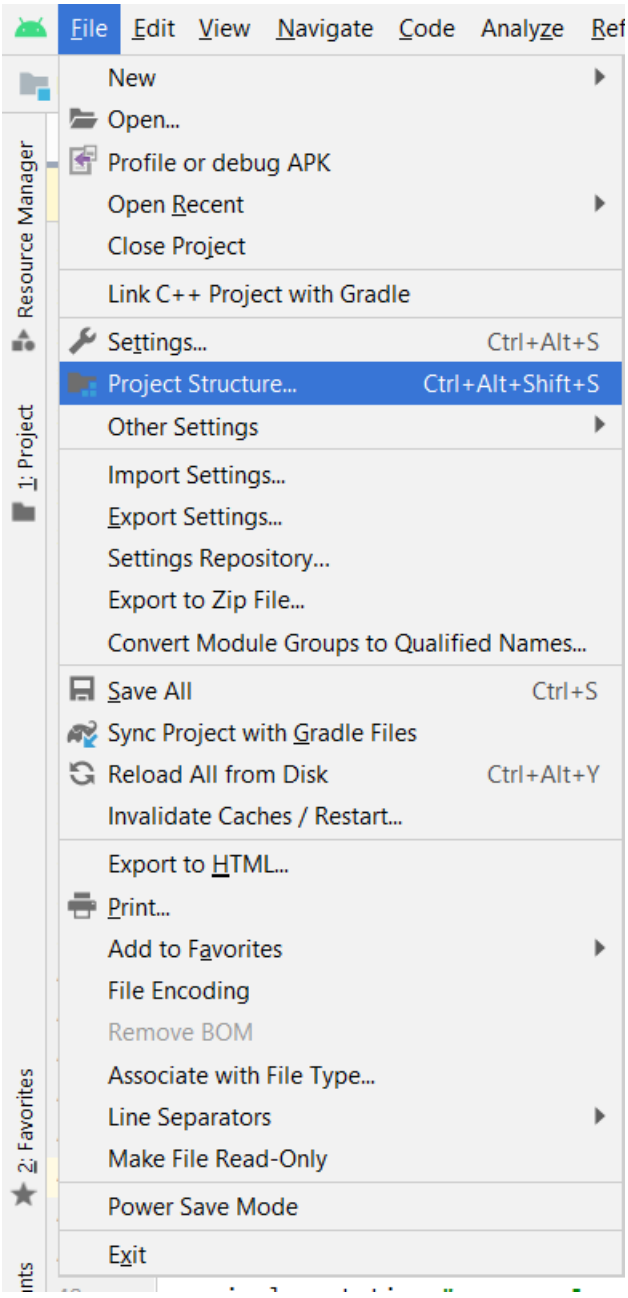
build.gradle (:app)

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

```
    }

    compileOptions {
        sourceCompatibility = 1.8
        targetCompatibility = 1.8
    }
}

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.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    // Room components
    implementation "androidx.room:room-runtime:$rootProject.roomVersion"
    annotationProcessor "androidx.room:room-compiler:$rootProject.roomVersion"
    androidTestImplementation "androidx.room:room-testing:$rootProject.roomVersion"

    // Lifecycle components
    implementation "androidx.lifecycle:lifecycle-extensions:$rootProject.archLifecycleVersion"
    implementation "androidx.lifecycle:lifecycle-common-java8:$lifecycle_version"

    // UI
    implementation "com.google.android.material:material:$rootProject.materialVersion"

    // Testing
    androidTestImplementation "androidx.arch.core:core-testing:$rootProject.coreTestingVersion"

}
```

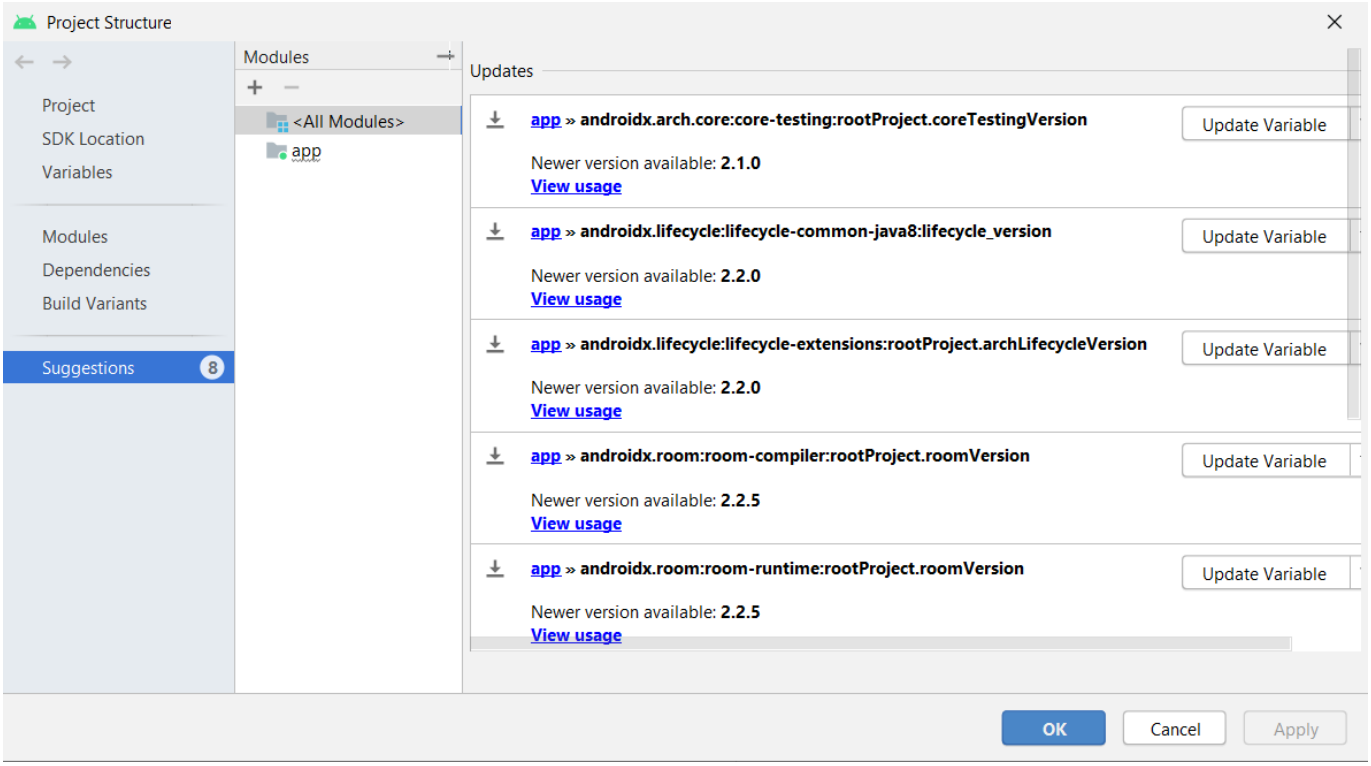
After adding the dependencies, we will download them from our project structure.

Go to **File** > **Project Structure**.

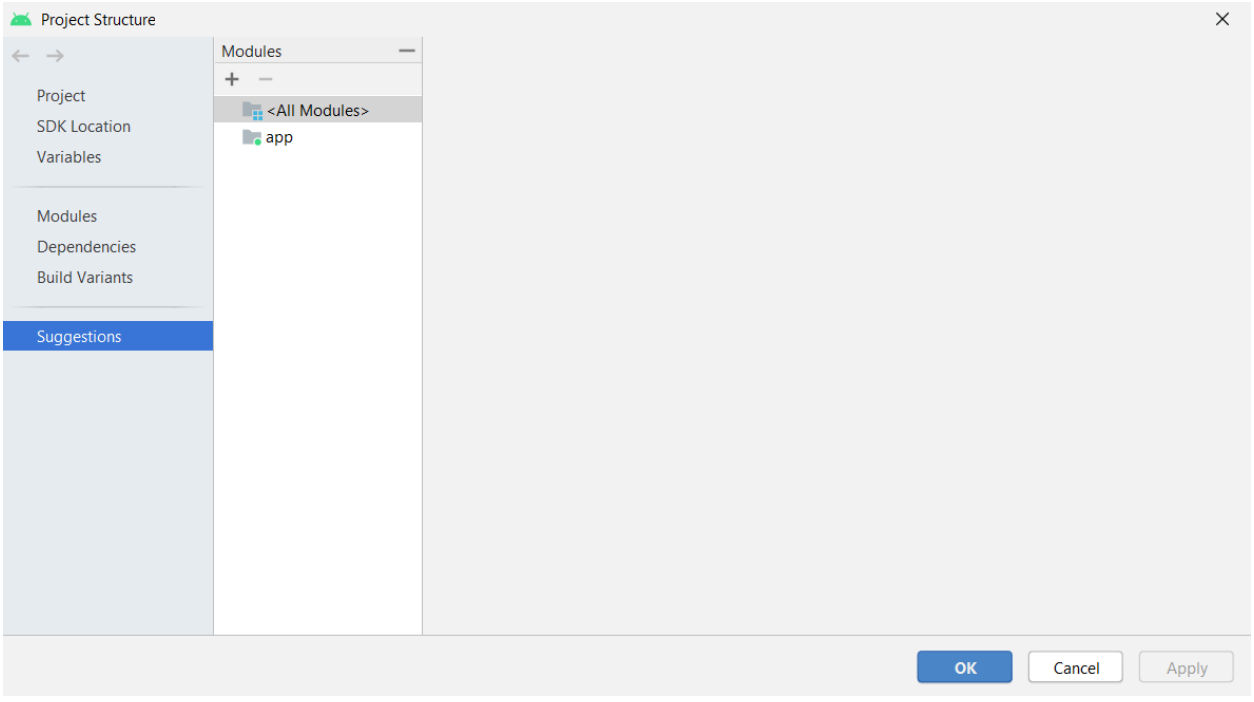


On the left tab, Click on the Suggestions.

You will see a list of updates. Click on the Update Variable button for all items in the list.



After all the items are updated, click OK.



If you look at your build.gradle file now, you will see the variables will change to the current version of the dependencies.

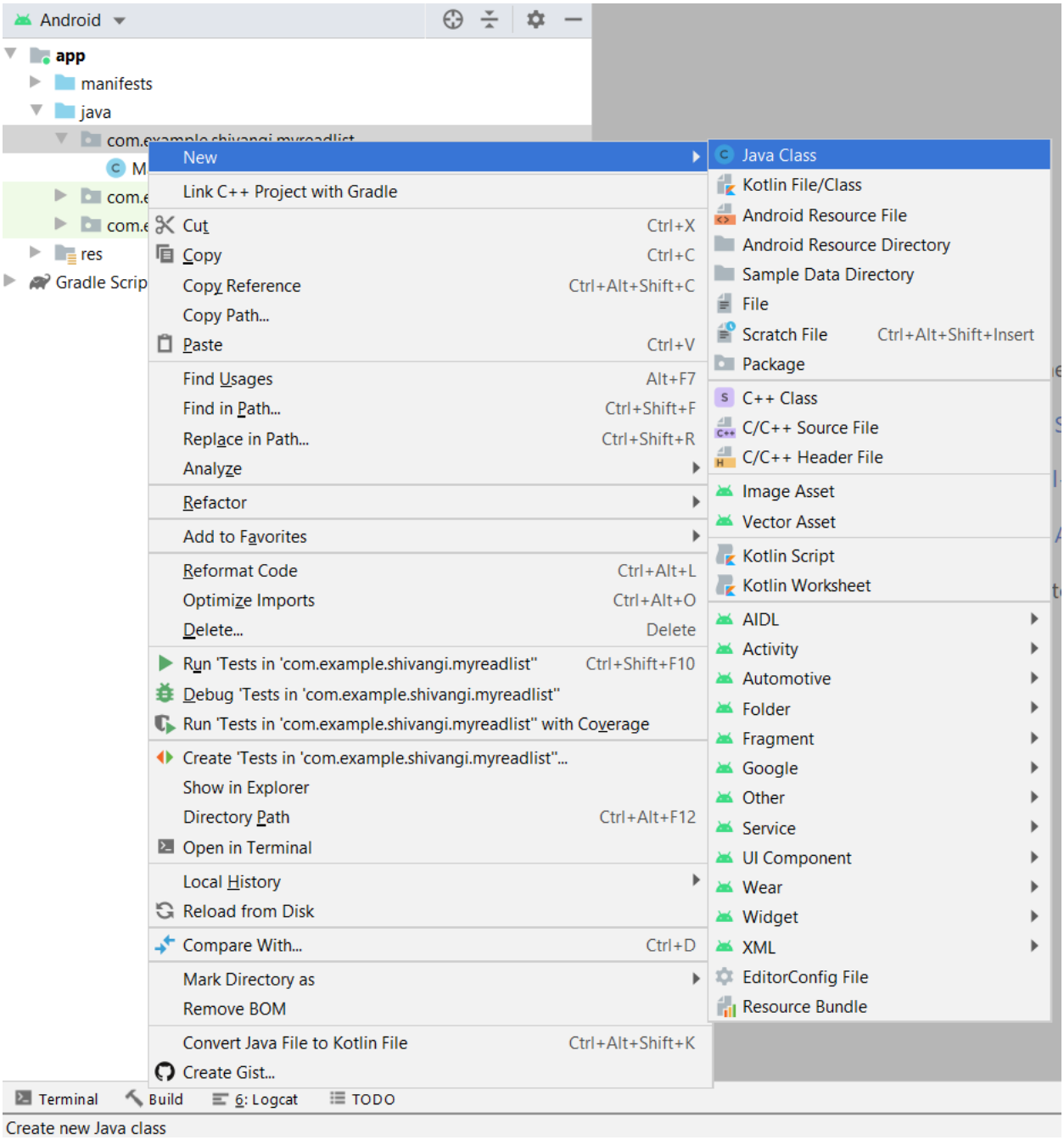
build.gradle (:app)

You can use the Project Structure dialog to view and edit your project configuration

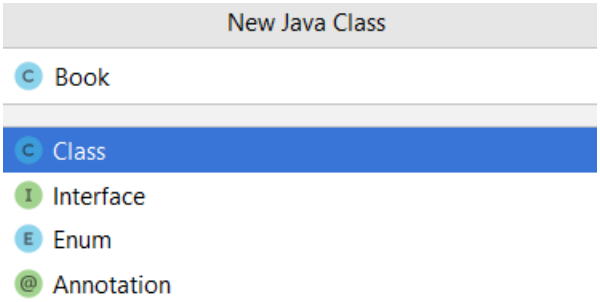
```
    }

    compileOptions {
        sourceCompatibility = 1.8
        targetCompatibility = 1.8
    }
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.13'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

    // Room components
    implementation "androidx.room:room-runtime:2.2.5"
    annotationProcessor "androidx.room:room-compiler:2.2.5"
    androidTestImplementation "androidx.room:room-testing:2.2.5"

    // Lifecycle components
    implementation "androidx.lifecycle:lifecycle-extensions:2.2.0"
    implementation "androidx.lifecycle:lifecycle-common-java8:2.2.0"

    // UI
    implementation "com.google.android.material:material:1.1.0"

    // Testing
    androidTestImplementation "androidx.arch.core:core-testing:2.1.0"

}
```

Now we will create a class that will create the attributes for our table as well as get and set the data from our room database.

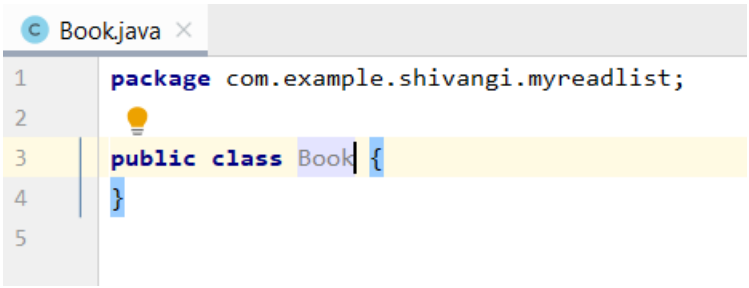
To create a java class, right-click on the package folder, select **New** > **Java class.**



Name the java class Book.



Below is the Book class we just created.



Now we will create attributes for our table. Room provides us with annotations that creates the table during compile time. This is efficient as we don't have to write all the boilerplate code.

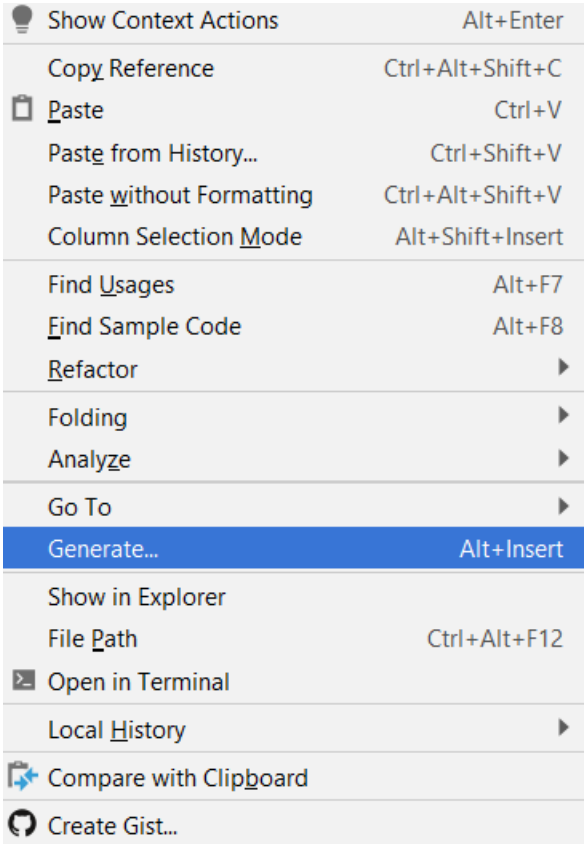
Add the following code to your book class.

In the code above, there is an @Entity annotation that creates a table with the table name book_table, by default, if the tableName is not set, the table name would be the same as the class name, in our case it would be Book. The @PrimaryKey that sets the field as primary key and provides an autogenerated value for every item in the list. You can change the name of a field by using the @ColumnInfo and setting its name parameter.

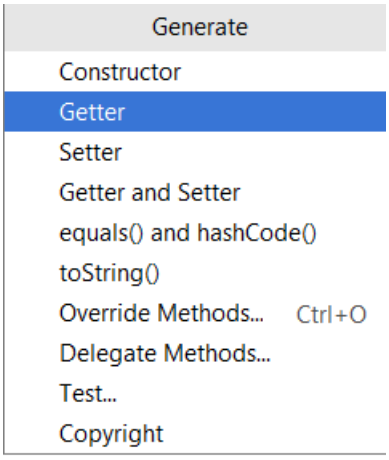
Apart from the @Entity, @PrimaryKey and @ColumnInfo, we have @NonNull annotation which is used to set a field that can never be null and @ForeignKey which is used to set a foreign key if you have more than one table in your database.

You can either make the objects / fields public, which is not a good idea as it allows direct access to the table. We therefore use getter methods to get the values from the objects.

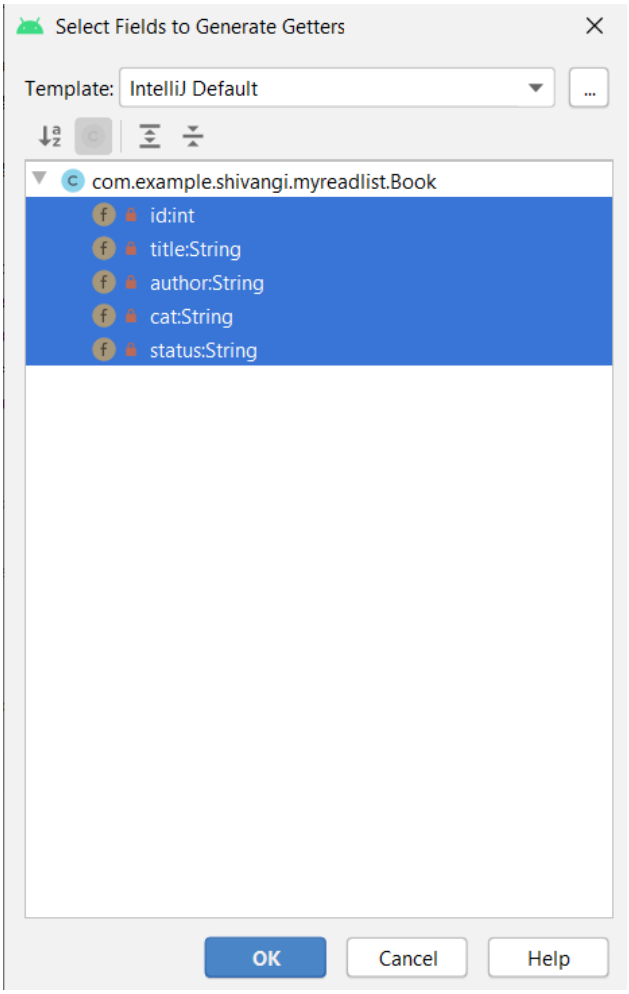
To add a getter method, right-click anywhere in the book class, and select Generate from the dropdown menu.



From the dialog box, select getter.



Select all the fields from the dialog box and click OK.

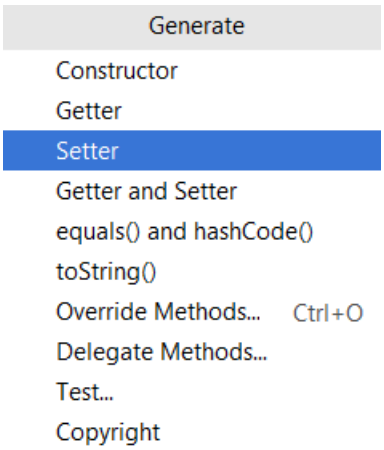


The following methods will be created.

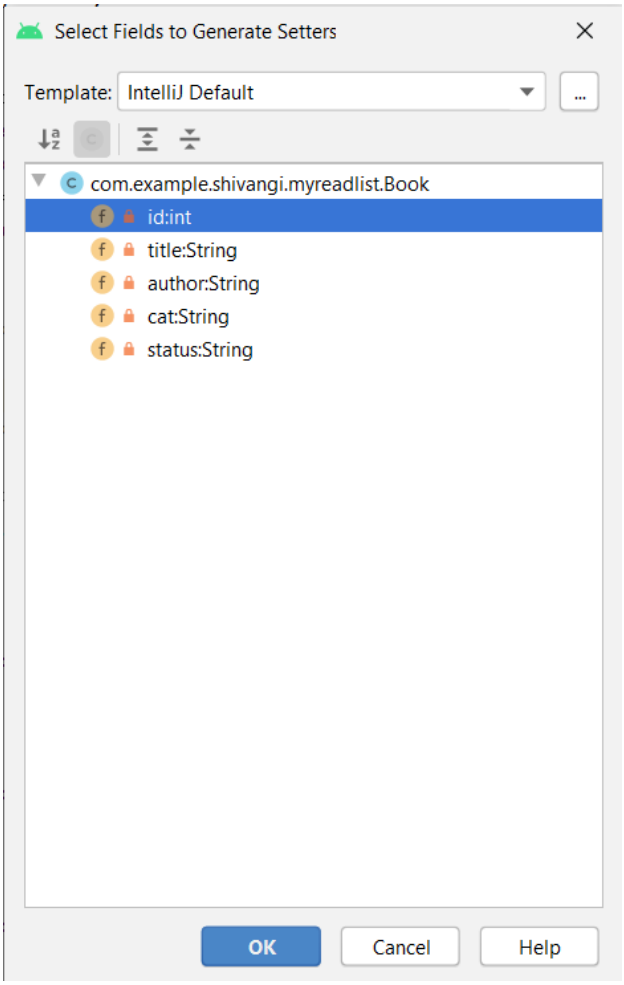
```
37      //Getter methods
38
39      public int getId() {
40          return id;
41      }
42
43      public String getTitle() {
44          return title;
45      }
46
47      public String getAuthor() {
48          return author;
49      }
50
51      public String getCat() {
52          return cat;
53      }
54
55      public String getStatus() {
56          return status;
57      }
```

We will need a setter method for our ID object because it is autogenerated value and so that the room can create this object.

To do this, right-click anywhere on the book class, select Generate and then select Setter from the dialog box.



From the dialog box that appears, just select the id field and click OK.

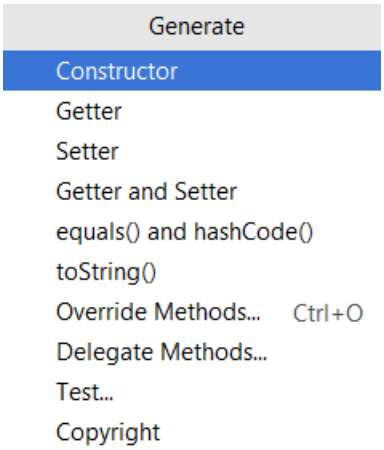


The following method will be created.

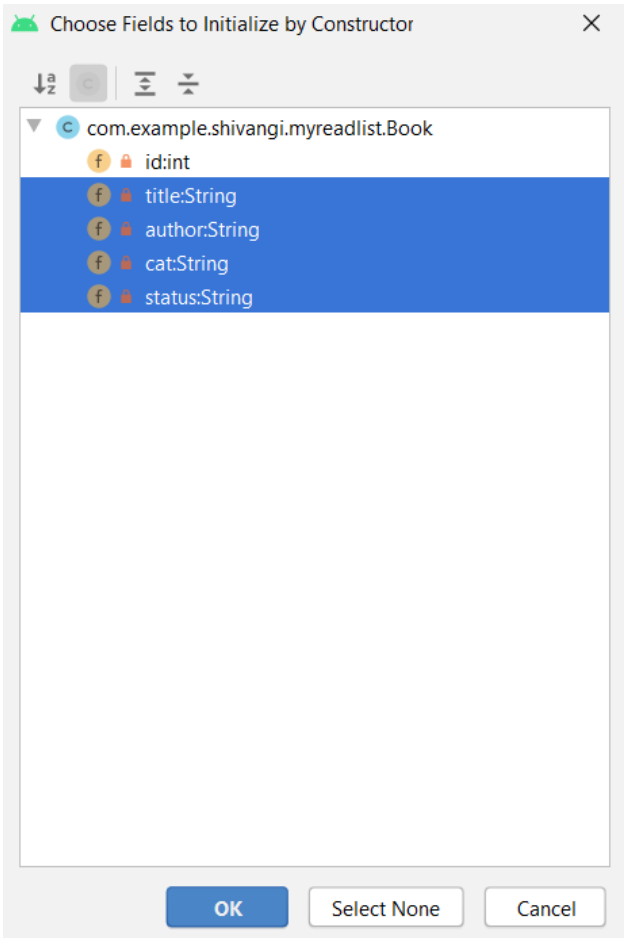
```
32
33      public void setId(int id) {
34          this.id = id;
35      }
```

We need a constructor to create the objects from the database.

To create a constructor, we follow the same steps as to creating a getter or a setter method. The difference is selecting Constructor from the dialog box that appears.



From the dialog box, select all the fields except ID as it is automatically generated.



The following code will be created.

```
    //Constructor
    public Book(String title, String author, String cat, String status) {
        this.title = title;
        this.author = author;
        this.cat = cat;
        this.status = status;
    }
```

Your final Book.java class will look as follows:

Book.java

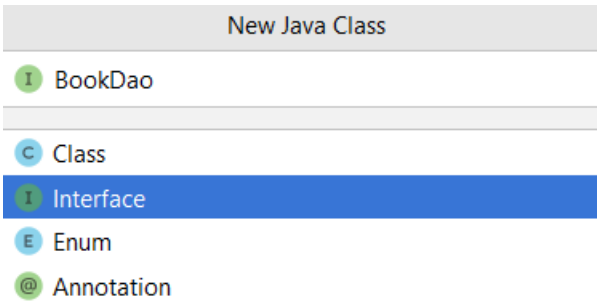
```
package com.example.shivangi.myreadlist;

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

@Entity(tableName = "book_table")
public class Book {

    //Attributes
    @PrimaryKey(autoGenerate = true)
    private int id;

    private String title;

    private String author;

    @ColumnInfo(name = "category")
    private String cat;

    private String status;

    //Constructor
    public Book(String title, String author, String cat, String status) {
        this.title = title;
        this.author = author;
        this.cat = cat;
        this.status = status;
    }
```

```
31      //Setter methods
32      public void setId(int id) {
33          this.id = id;
34      }
35
36      //Getter methods
37      public int getId() {
38          return id;
39      }
40
41      public String getTitle() {
42          return title;
43      }
44
45      public String getAuthor() {
46          return author;
47      }
48
49      public String getCat() {
50          return cat;
51      }
52
53      public String getStatus() {
54          return status;
55      }
56  }
```

Now we will create a DAO. Usually DAOs are Interfaces or abstract classes as we don't provide the method body. We just create the methods and room will automatically generate the necessary code for the methods.

To create a DAO, right-click on your package folder, select New > Java class.

Then select the Interface option from the dialog box and name the file BookDao.



The following Interface is created for you.

BookDao.java

```
1  package com.example.shivangi.myreadlist;
2
3  public interface BookDao {
4  }
5
```

Annotate the interface with the @Dao annotation provided by the room. One Dao is created per entity. Since we only have one Entity, we will only have one Interface.

BookDao.java

```
1  package com.example.shivangi.myreadlist;
2
3  import androidx.room.Dao;
4
5  @Dao
6  public interface BookDao {
7
8  }
9
```

Now we will create the necessary methods.

Add the following code to the Interface.

BookDao.java

```
3   import androidx.lifecycle.LiveData;
4   import androidx.room.Dao;
5   import androidx.room.Delete;
6   import androidx.room.Insert;
7   import androidx.room.Query;
8   import androidx.room.Update;
9
10  import java.util.List;
11
12  @Dao
13  public interface BookDao {
14
15      @Insert
16      void insert(Book book);
17
18      @Update
19      void update(Book book);
20
21      @Delete
22      void delete(Book book);
23
24      @Query("DELETE FROM book_table")
25      void deleteAll();
26
27      @Query("SELECT * FROM book_table ORDER BY title ASC")
28      LiveData<List<Book>> getBookList();
29  }
30
```

In the code above, we just created the method names, return types and the arguments for all the operations for our database. The @Insert, @Update, @ Delete and @Query annotations will allow the room to automatically generate the code for the corresponding annotation. We do not define what the annotations will do.

The @Query annotation allows us to define the database operations in form of a String. Any typos made in the String will immediately show us an error.

Before the List<>, we put the LiveData. This will observe our data and update any necessary changes.

Your final BookDao interface looks as follows:

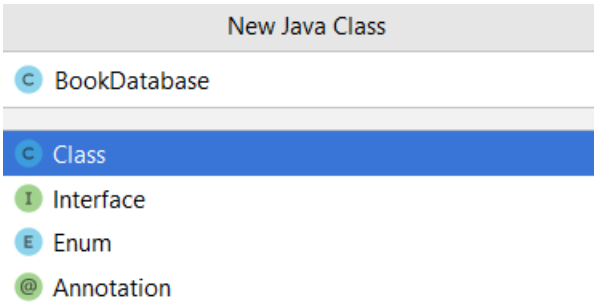
BookDao.java

```
3   import androidx.lifecycle.LiveData;
4   import androidx.room.Dao;
5   import androidx.room.Delete;
6   import androidx.room.Insert;
7   import androidx.room.Query;
8   import androidx.room.Update;
9
10  import java.util.List;
11
12  @Dao
13  public interface BookDao {
14
15      @Insert
16      void insert(Book book);
17
18      @Update
19      void update(Book book);
20
21      @Delete
22      void delete(Book book);
23
24      @Query("DELETE FROM book_table")
25      void deleteAll();
26
27      @Query("SELECT * FROM book_table ORDER BY title ASC")
28      LiveData<List<Book>> getBookList();
29  }
30
```

Now we create another class for our Room which will connect the Entity and the DAO and create an instance of the database. This class will be an abstract class.

Right click on your package folder and select **New** > **Java class**.

Name the class BookDatabase.



Your BookDatabase class looks as follows:

BookDatabase.java

```
1  package com.example.shivangi.myreadlist;
2
3  public abstract class BookDatabase{
4
5  }
```

Extend the BookDatabase to RoomDatabase:

BookDatabase.java

```
1  package com.example.shivangi.myreadlist;
2
3  import androidx.room.RoomDatabase;
4
5  public abstract class BookDatabase extends RoomDatabase {
6
7  }
```

Annotate the class using @Database. Define your entities and version.

NOTE: Once you instantiate a database, and then make changes to it, you have to increment the version of the database.

BookDatabase.java

```
1  package com.example.shivangi.myreadlist;
2
3  import androidx.room.Database;
4  import androidx.room.RoomDatabase;
5
6  @Database(entities = Book.class, version = 1)
7  public abstract class BookDatabase extends RoomDatabase {
8
9  }
```

Now we will create class variables. We create a BookDatabase instance to turn it into a singleton. Singleton means that you can't create multiple instances of this class and ensures you use the same instance in the app.

We create a method without the body as room provides the required code.

```
@Database(entities = Book.class, version = 1)
public abstract class BookDatabase extends RoomDatabase {

    private static BookDatabase instance;
    public abstract BookDao bookDao();

}
```

Now we will create a Singleton method for our BookDatabase class. Add the following code:

```
@Database(entities = Book.class, version = 1)
public abstract class BookDatabase extends RoomDatabase {

    private static BookDatabase instance;
    public abstract BookDao bookDao();

    public static synchronized BookDatabase getInstance(Context context){

    }
}
```

In the code above, we have stated the method to be synchronized. Synchronized means that this method will only allow one thread at a time to access it, preventing the creation of multiple instances.

We will now instantiate the database with the use of the Room.databaseBuilder. This method accepts a context, the database class, and name you'd want for the current file as parameters. Room provides you with a fallbackToDestructiveMigration() method to allow the room to migrate to a new version of the database. This method will delete the old database and create a new one with the incremented version.

Lastly, you call the build() to return an instance of the BookDatabase class.

Add the following code:

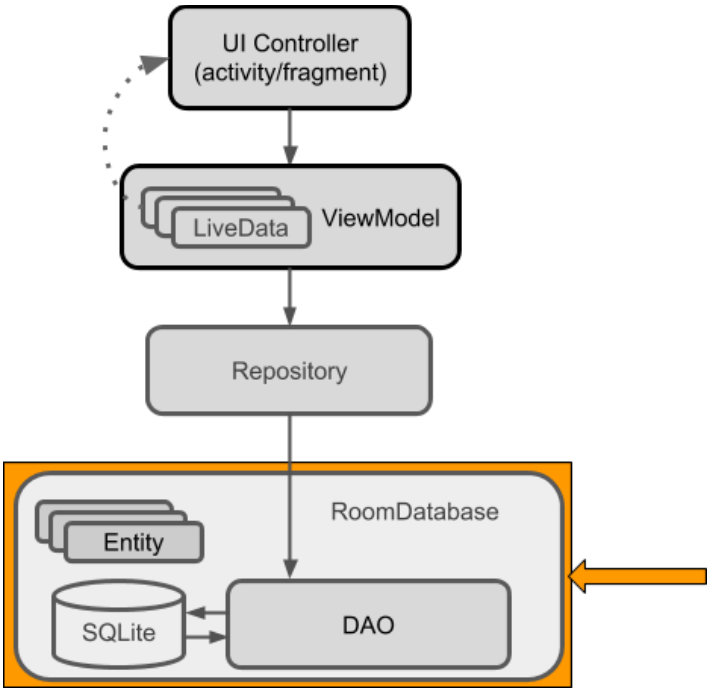
```
    public static synchronized BookDatabase getInstance(Context context){
        if(instance == null){
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    BookDatabase.class, name: "book_database")
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return instance;
    }
```

Your BookDatabase will look as follows:

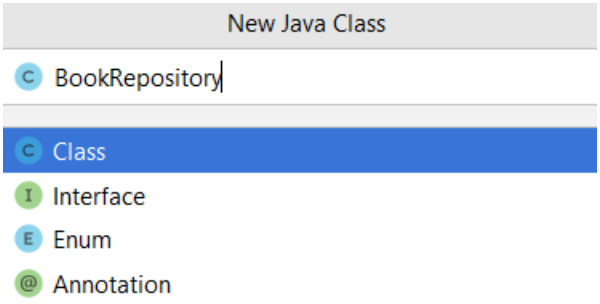
BookDatabase.java

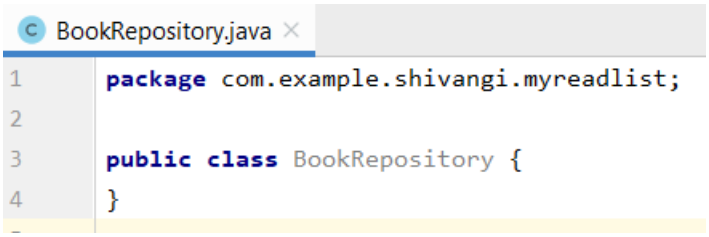
```java
package com.example.shivangi.myreadlist;

import android.content.Context;

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

@Database(entities = Book.class, version = 1)
public abstract class BookDatabase extends RoomDatabase {

    private static BookDatabase instance;
    public abstract BookDao bookDao();

    public static synchronized BookDatabase getInstance(Context context){
        if(instance == null){
            instance = Room.databaseBuilder(context.getApplicationContext(),
                    BookDatabase.class, name: "book_database")
                    .fallbackToDestructiveMigration()
                    .build();
        }
        return instance;
    }
}
```

We completed creating the room with the Entity and DAO that will communicate with the SQLite.



Now let's move to creating a repository class and name it BookRepository.





First we will add two class variables, one for BookDao and the second one for Book List LiveData.

Add the following code:

```
7   public class BookRepository {
8
9       private BookDao bookDao;
10      private LiveData<List<Book>> bookList;
11
12      |
13  }
```

We will now add a constructor where we assign the variables above. In this method, we will pass an Application parameter. Application is a subclass of Context and we will use it to create our context for database instance.

```
9   public class BookRepository {
10
11      private BookDao bookDao;
12      private LiveData<List<Book>> bookList;
13
14      BookRepository(Application application){
15          BookDatabase db = BookDatabase.getInstance(application);
16          bookDao = db.bookDao();
17          bookList = bookDao.getBookList();
18      }
```

We will now create methods for our database operations.

```
19      public void insert(Book book){ }
20
21      public void update(Book book){ }
22
23      public void delete(Book book){ }
24
25      public void deleteAll(){ }
26
27      public LiveData<List<Book>> getBookList(){
28          return bookList;
29      }
```

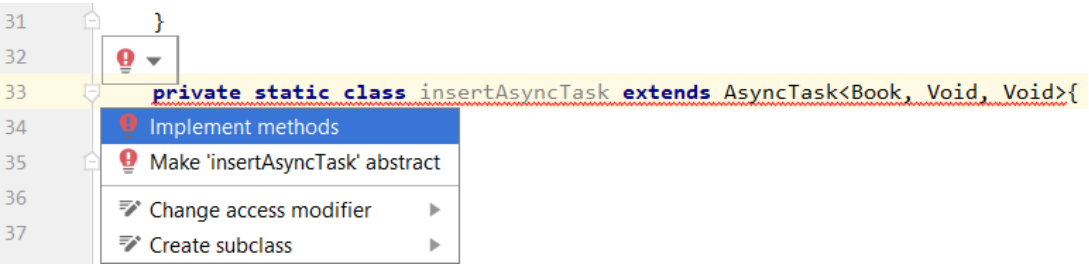
In the above code, Room will automatically execute the database operation for the LiveData on the background thread, so we don't need to add anymore operations for that method.

We will now create AsyncTasks for our other methods to perform the operations on the background thread as room doesn't allow database operations on the main thread to prevent the app from crashing.

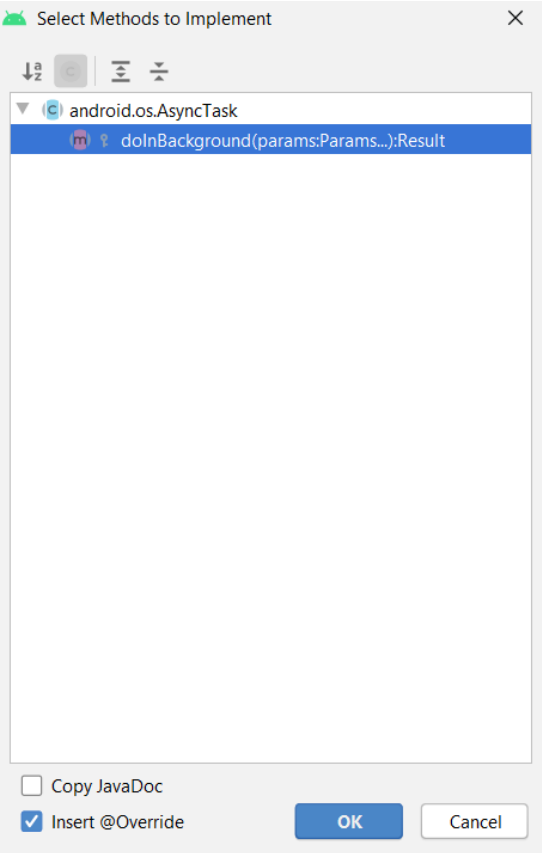
Add the following code below the methods we just created.

```
32
33      private static class insertAsyncTask extends AsyncTask<Book, Void, Void>{
34
35      }
```

This is an inner class that extends AsyncTask. Click on the red line, you will see a red bulb on the left. Click on the bulb and select implement methods.



Click ok on the dialog box that appears.



The following method will be added to the inner class.

```
32
33      private static class insertAsyncTask extends AsyncTask<Book, Void, Void>{
34
35          @Override
36          protected Void doInBackground(Book... books) {
37              return null;
38          }
39      }
```

We will create a member variable for BookDao as we need it to perform operations. Since the insertAsyncTask is a static class, we cannot access the BookDao directly. We will have to create a constructor to access it.

Add the following code in the insertAsyncTask class:

```
41      private static class insertAsyncTask extends AsyncTask<Book, Void, Void>{
42
43          private BookDao asyncDao;
44
45          insertAsyncTask(BookDao dao){
46              asyncDao = dao;
47          }
48
49          @Override
50          protected Void doInBackground(Book... books) {
51              return null;
52          }
53      }
```

Now we will add the following to our doInBackground() method. We are simply calling the insert method in the Dao and getting the book at that index.

```
41      private static class insertAsyncTask extends AsyncTask<Book, Void, Void>{
42
43          private BookDao asyncDao;
44
45          insertAsyncTask(BookDao dao){
46              asyncDao = dao;
47          }
48
49          @Override
50          protected Void doInBackground(Book... books) {
51              asyncDao.insert(books[0]);
52              return null;
53          }
54      }
```

We will do the same for our other database operations. We have to create AsyncTasks for each operation and add the corresponding operation in the doInBackground() method. Note that in the deleteAllAsyncTask class, we changed the Book parameter in the AsyncTask to Void as we don't have to pass a Book. Therefore, we don't have to pass any arguments in the asyncDao.deleteAll() method in doInBackground().

Below is the AsyncTask code for the other operations.

```
48      private static class updateAsyncTask extends AsyncTask<Book, Void, Void>{
49
50          private BookDao asyncDao;
51
52          updateAsyncTask(BookDao dao){
53              asyncDao = dao;
54          }
55
56          @Override
57  @      protected Void doInBackground(Book... books) {
58              asyncDao.update(books[0]);
59              return null;
60          }
61      }
62
63      private static class deleteAsyncTask extends AsyncTask<Book, Void, Void>{
64
65          private BookDao asyncDao;
66
67          deleteAsyncTask(BookDao dao){
68              asyncDao = dao;
69          }
70
71          @Override
72  @      protected Void doInBackground(Book... books) {
73              asyncDao.delete(books[0]);
74              return null;
75          }
76      }
```

```
86      private static class deleteAllAsyncTask extends AsyncTask<Void, Void, Void>{
87
88          private BookDao asyncDao;
89
90          deleteAllAsyncTask(BookDao dao){
91              asyncDao = dao;
92          }
93
94          @Override
95      protected Void doInBackground(Void... voids) {
96              asyncDao.deleteAll();
97              return null;
98          }
99      }
```

Let’s move to the methods we created for each operation. In the insert method, we will create an instance of the insertAsyncTask class and pass the BookDao variable and call the execute method on Book.

The code will look like below:

```
21      public void insert(Book book){
22          new insertAsyncTask(bookDao).execute(book);
23      }
```

We will do the same for the other methods and create an instance of the corresponding AsyncTask.

```
25      public void update(Book book){
26          new updateAsyncTask(bookDao).execute(book);
27      }
28
29      public void delete(Book book){
30          new deleteAsyncTask(bookDao).execute(book);
31      }
32
33      public void deleteAll(){
34          new deleteAllAsyncTask(bookDao).execute();
35      }
```

Your BookRepository class will look as follows:

BookRepository.java

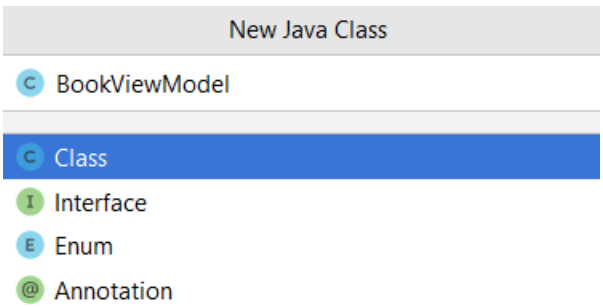
```java
package com.example.shivangi.myreadlist;

import android.app.Application;
import android.os.AsyncTask;
import androidx.lifecycle.LiveData;
import java.util.List;

public class BookRepository {

    private BookDao bookDao;
    private LiveData<List<Book>> bookList;

    BookRepository(Application application){
        BookDatabase db = BookDatabase.getInstance(application);
        bookDao = db.bookDao();
        bookList = bookDao.getBookList();
    }

    public void insert(Book book){
        new insertAsyncTask(bookDao).execute(book);
    }

    public void update(Book book){
        new updateAsyncTask(bookDao).execute(book);
    }

    public void delete(Book book){
        new deleteAsyncTask(bookDao).execute(book);
    }

    public void deleteAll(){
        new deleteAllAsyncTask(bookDao).execute();
    }
```

```
37      public LiveData<List<Book>> getBookList(){
38          return bookList;
39      }
40
41      private static class insertAsyncTask extends AsyncTask<Book, Void, Void>{
42
43          private BookDao asyncDao;
44
45          insertAsyncTask(BookDao dao){
46              asyncDao = dao;
47          }
48
49          @Override
50  @      protected Void doInBackground(Book... books) {
51              asyncDao.insert(books[0]);
52              return null;
53          }
54      }
55
56      private static class updateAsyncTask extends AsyncTask<Book, Void, Void>{
57
58          private BookDao asyncDao;
59
60          updateAsyncTask(BookDao dao){
61              asyncDao = dao;
62          }
63
64          @Override
65  @      protected Void doInBackground(Book... books) {
66              asyncDao.update(books[0]);
67              return null;
68          }
69      }
```

```
71      private static class deleteAsyncTask extends AsyncTask<Book, Void, Void>{
72
73          private BookDao asyncDao;
74
75          deleteAsyncTask(BookDao dao){
76              asyncDao = dao;
77          }
78
79          @Override
80  @       protected Void doInBackground(Book... books) {
81              asyncDao.delete(books[0]);
82              return null;
83          }
84      }
85
86      private static class deleteAllAsyncTask extends AsyncTask<Void, Void, Void>{
87
88          private BookDao asyncDao;
89
90          deleteAllAsyncTask(BookDao dao){
91              asyncDao = dao;
92          }
93
94          @Override
95          protected Void doInBackground(Void... voids) {
96              asyncDao.deleteAll();
97              return null;
98          }
99      }
100 }
```

Now we will create a ViewModel class for our database.

Name the class BookViewModel.



BookRepository.java | BookViewModel.java

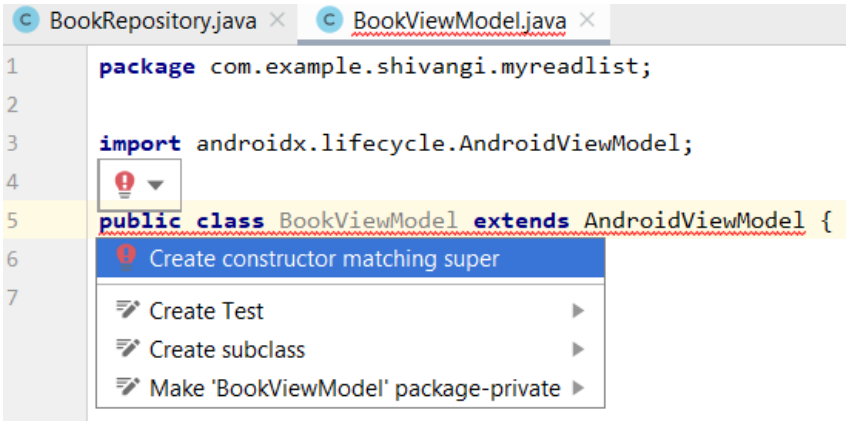
```
1  package com.example.shivangi.myreadlist;
2
3  public class BookViewModel{
4  }
```

The BookViewModel class will extend AndroidViewModel. AndroidViewModel is a subclass of ViewModel and we pass Application in the constructor so that there is no reference to context of the activity. This prevents memory leak.

BookRepository.java | BookViewModel.java

```
1  package com.example.shivangi.myreadlist;
2
3  import androidx.lifecycle.AndroidViewModel;
4
5  public class BookViewModel extends AndroidViewModel {
6  }
```

Click on the red line and a bulb will appear on the left. Click on it and add the Create constructor matching super.



The following constructor method will be added.

BookRepository.java | BookViewModel.java

```
1  package com.example.shivangi.myreadlist;
2
3  import android.app.Application;
4
5  import androidx.annotation.NonNull;
6  import androidx.lifecycle.AndroidViewModel;
7
8  public class BookViewModel extends AndroidViewModel {
9      public BookViewModel(@NonNull Application application) {
10         super(application);
11     }
12 }
```

We will create two class variable one for Dao and one for the BookList.

```
11 public class BookViewModel extends AndroidViewModel {
12
13     private BookRepository bookRepository;
14     private LiveData<List<Book>> bookList;
15
16     public BookViewModel(@NonNull Application application) {
17         super(application);
18     }
19 }
```

We instantiate the variable in the constructor. To do this, add the following code to the constructor:

```
11 public class BookViewModel extends AndroidViewModel {
12
13     private BookRepository bookRepository;
14     private LiveData<List<Book>> bookList;
15
16     public BookViewModel(@NonNull Application application) {
17         super(application);
18         bookRepository = new BookRepository(application);
19         bookList = bookRepository.getBookList();
20     }
21 }
```

Since our activity only has access to the ViewModel class, we will create methods to refer to the database operation methods from our repository.

Add the following code below the constructor:

```
22      public void insert(Book book){
23          bookRepository.insert(book);
24      }
```

In the code above, we call the insert method from the repository and pass Book.

We will do the same for the other methods.

```
26      public void update(Book book){
27          bookRepository.update(book);
28      }
29
30      public void delete(Book book){
31          bookRepository.delete(book);
32      }
33
34      public void deleteAll(){
35          bookRepository.deleteAll();
36      }
37
38      public LiveData<List<Book>> getBookList() {
39          return bookList;
40      }
```

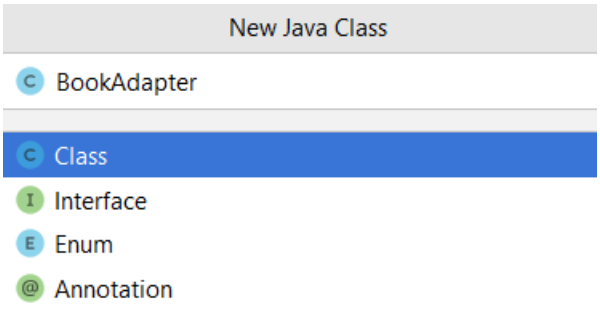
Your final BookViewModel will look as follows:

BookRepository.java | BookViewModel.java

```java
package com.example.shivangi.myreadlist;

import ...

public class BookViewModel extends AndroidViewModel {

    private BookRepository bookRepository;
    private LiveData<List<Book>> bookList;

    public BookViewModel(@NonNull Application application) {
        super(application);
        bookRepository = new BookRepository(application);
        bookList = bookRepository.getBookList();
    }

    public void insert(Book book){
        bookRepository.insert(book);
    }

    public void update(Book book){
        bookRepository.update(book);
    }

    public void delete(Book book){
        bookRepository.delete(book);
    }

    public void deleteAll(){
        bookRepository.deleteAll();
    }

    public LiveData<List<Book>> getBookList() {
        return bookList;
    }
}
```

We will now create an adapter to display our data. A recycler view is responsible for creating a view for each item in the data set.

Create a new class and name it BookAdapter.



BookRepository.java | BookViewModel.java | BookAdapter.java

```
1  package com.example.shivangi.myreadlist;
2
3  public class BookAdapter {
4  }
```

We will extend our BookAdapter class to extend RecyclerView.Adapter.

BookAdapter.java

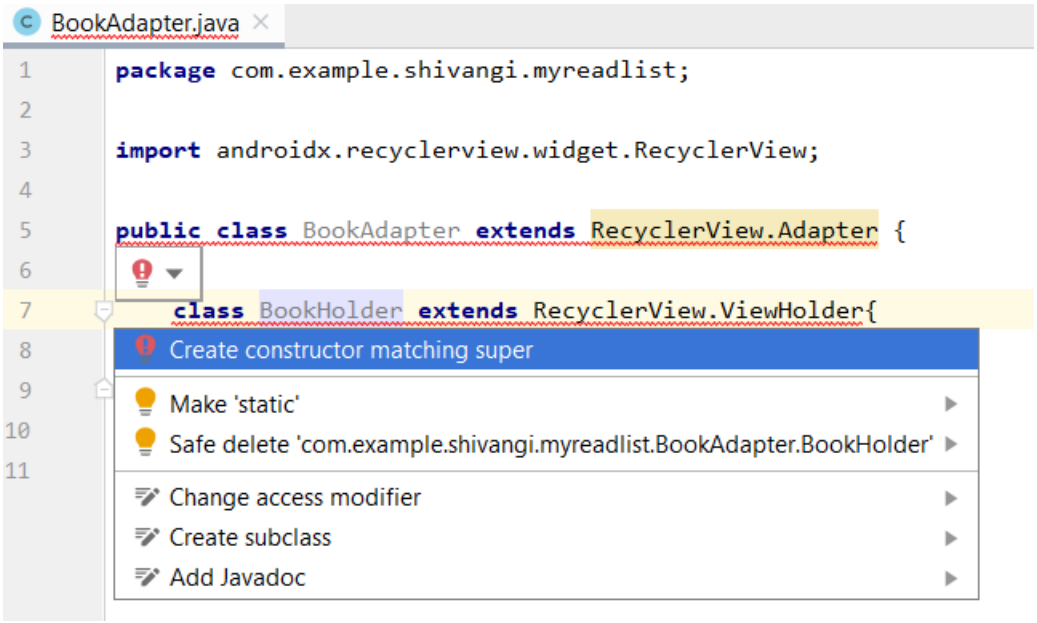
```
1  package com.example.shivangi.myreadlist;
2
3  import androidx.recyclerview.widget.RecyclerView;
4
5  public class BookAdapter extends RecyclerView.Adapter {
6  }
```

We will now create an inner ViewHolder class and extend it RecyclerView.ViewHolder.

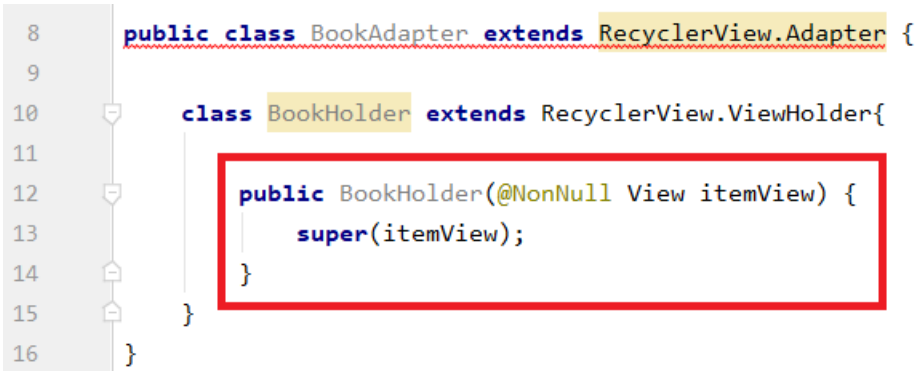
BookAdapter.java

```
1  package com.example.shivangi.myreadlist;
2
3  import androidx.recyclerview.widget.RecyclerView;
4
5  public class BookAdapter extends RecyclerView.Adapter {
6
7      class BookHolder extends RecyclerView.ViewHolder{}
8  }
```

The red lines are due to methods that are not implemented. To implement these methods and constructor, click on the BookHolder class, a red bulb icon will appear on the left. Click on it and select Create constructor matching super.



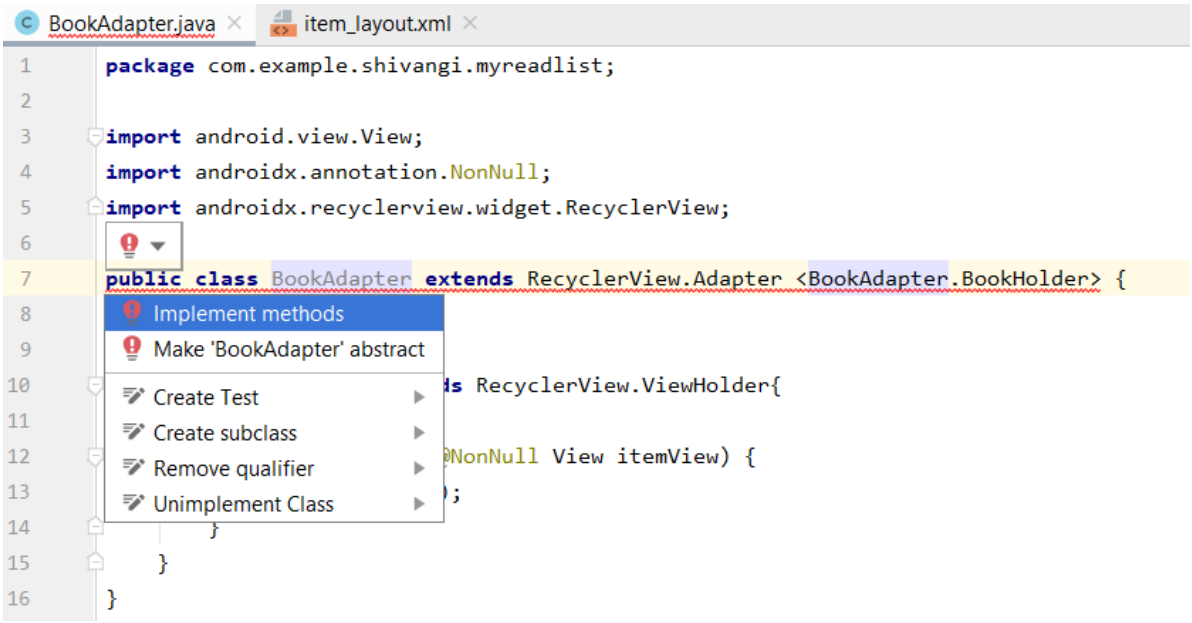
The following constructor will be created.

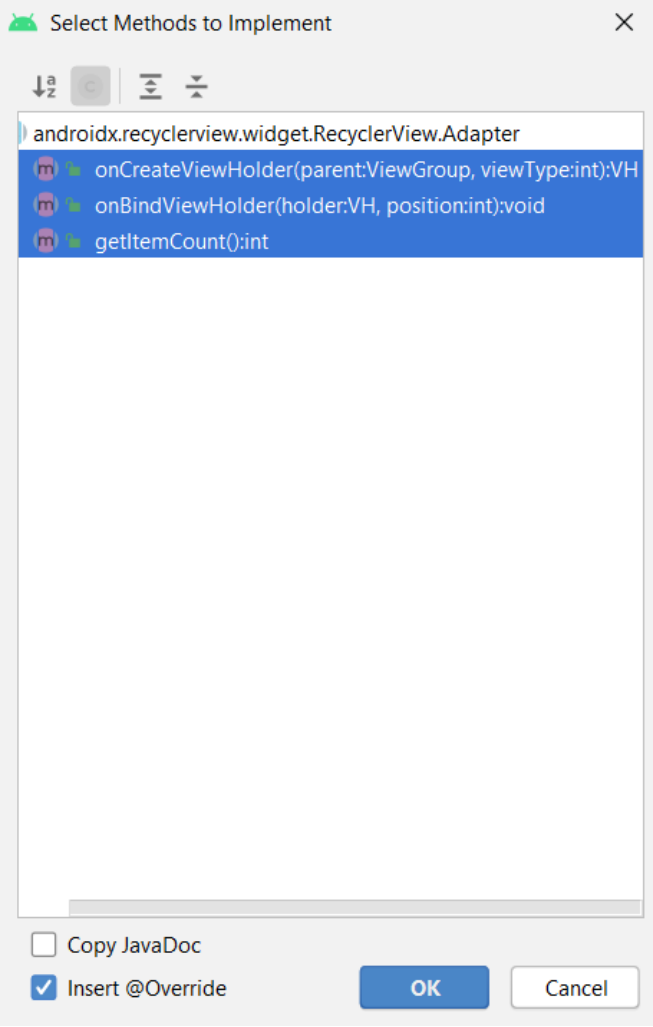


In our BookAdapter class, we will add the view holder for the adapter.

```
7   public class BookAdapter extends RecyclerView.Adapter <BookAdapter.BookHolder> {
8
9
10      class BookHolder extends RecyclerView.ViewHolder{
11
12          public BookHolder(@NonNull View itemView) {
13              super(itemView);
14          }
15      }
16  }
```

We will implement the BookAdapter methods. Click on the BookAdapter, a red bulb will appear on the left. From the dropdown list, select Implement methods. A dialog box will appear. You don't have to make any changes, just click OK.




Select Methods to Implement
androidx.recyclerview.widget.RecyclerView.Adapter
onCreateViewHolder(parent:ViewGroup, viewType:int):VH
onBindViewHolder(holder:VH, position:int):void
getItemCount():int
Copy JavaDoc
Insert @Override
OK
Cancel

The following methods are added:

```
public class BookAdapter extends RecyclerView.Adapter <BookAdapter.BookHolder> {


    @NonNull
    @Override
    public BookHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        return null;
    }

    @Override
    public void onBindViewHolder(@NonNull BookHolder holder, int position) {

    }

    @Override
    public int getItemCount() {
        return 0;
    }

    class BookHolder extends RecyclerView.ViewHolder{

        public BookHolder(@NonNull View itemView) {
            super(itemView);
        }
    }
}
```

In our ViewHolder, we will add variables. These variables will be assigned to refer to the components in our item_layout.xml. We use the itemView parameter provided in the BookHolder constructor to access the components.

Add the following variables in your BookHolder.

```
    class BookHolder extends RecyclerView.ViewHolder{

        private TextView bookName;
        private TextView author;
        private TextView category;
        private TextView status;

        public BookHolder(@NonNull View itemView) {
            super(itemView);
        }
    }
```

Add the following code in your BookHolder constructor:

```
35          public BookHolder(@NonNull View itemView) {
36              super(itemView);
37              bookName = itemView.findViewById(R.id.bookName);
38              author = itemView.findViewById(R.id.authorName);
39              category = itemView.findViewById(R.id.bkCategory);
40              status = itemView.findViewById(R.id.bkStatus);
41          }
42      }
```

We will add a class variable for our BookAdapter class that will get the list of Books. We will initialize it as a new ArrayList. Add the following code:

```
12  public class BookAdapter extends RecyclerView.Adapter <BookAdapter.BookHolder> {
13
14      private List<Book> books = new ArrayList<>();
15
```

onCreateViewHolder method will inflate the item_layout for each data in the ArrayList. Add the following code in the onCreateViewHolder method:

```
17      @NonNull
18      @Override
19      public BookHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
20          View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_layout,
21                  parent, attachToRoot: false);
22          return new BookHolder(view);
23      }
```

onBindViewHolder - This is where we will get data from our java object into our BookHolder view. We will create a reference to the Book object in that position.

We will use the holder and get the reference to the component in our layout and use .setText() method to set the value from the data.

```
25      @Override
26      public void onBindViewHolder(@NonNull BookHolder holder, int position) {
27          Book current = books.get(position);
28
29          holder.bookName.setText(current.getTitle());
30          holder.author.setText(current.getAuthor());
31          holder.category.setText(current.getCat());
32          holder.status.setText(current.getStatus());
33      }
```

Change the return value to return the size of the ArrayList.

```
35      @Override
36      public int getItemCount() {
37          return books.size();
38      }
```

We will create a method to get the list of Books. Add the following code below your getItemCount() method.

```
40      public void setBooks(List<Book> book){
41          this.books = book;
42          notifyDataSetChanged();
43      }
```

Your final BookAdapter class will look as follows:

BookAdapter.java | item_layout.xml

```
1  package com.example.shivangi.myreadlist;
2
3  import ...
12
13 public class BookAdapter extends RecyclerView.Adapter <BookAdapter.BookHolder> {
14
15     private List<Book> books = new ArrayList<>();
16
17     @NonNull
18     @Override
19     public BookHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
20         View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_layout,
21                 parent, attachToRoot: false);
22         return new BookHolder(view);
23     }
24
25     @Override
26     public void onBindViewHolder(@NonNull BookHolder holder, int position) {
27         Book current = books.get(position);
28
29         holder.bookName.setText(current.getTitle());
30         holder.author.setText(current.getAuthor());
31         holder.category.setText(current.getCat());
32         holder.status.setText(current.getStatus());
33     }
34
35     @Override
36     public int getItemCount() {
37         return books.size();
38     }
39
```

```
40      public void setBooks(List<Book> book){
41          this.books = book;
42          notifyDataSetChanged();
43      }
44
45      class BookHolder extends RecyclerView.ViewHolder{
46
47          private TextView bookName;
48          private TextView author;
49          private TextView category;
50          private TextView status;
51
52          public BookHolder(@NonNull View itemView) {
53              super(itemView);
54              bookName = itemView.findViewById(R.id.bookName);
55              author = itemView.findViewById(R.id.authorName);
56              category = itemView.findViewById(R.id.bkCategory);
57              status = itemView.findViewById(R.id.bkStatus);
58          }
59      }
60  }
61
```

Now let's setup the adapter in our MainActivity.

Add the following code in your onCreate method.

```
23      @Override
24      protected void onCreate(Bundle savedInstanceState) {
25          super.onCreate(savedInstanceState);
26          setContentView(R.layout.activity_main);
27
28          RecyclerView recyclerView = findViewById(R.id.recyclerView);
29          recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
30          recyclerView.setHasFixedSize(true);
31
32          BookAdapter adapter = new BookAdapter();
33          recyclerView.setAdapter(adapter);
34
35      }
```

Now let's link our ViewModel to our Activity.

Add a member variable for BookViewModel.

```
18  public class MainActivity extends AppCompatActivity {
19
20      private BookViewModel viewModel;
21
```

Now we assign the viewModel variable. We ask the android system for a view model instead of assigning a new BookViewModel because we will create new instances of it. To do this add the following code in the onCreate method.

```
35          viewModel = ViewModelProviders.of( activity: this).get(BookViewModel.class);
36          viewModel.getBookList().observe( owner: this, new Observer<List<Book>>() {
37              @Override
38              public void onChanged(List<Book> books) {
39                  adapter.setBooks(books);
40              }
41          });
```

In the above code, we use .observe() as we are using LiveData. It creates a new observer and triggers every time our data is changed.

Now let's override the onCreateOptionMenu() method. When you initially create this method, it will look as follows:

```
54      @Override
55      public boolean onCreateOptionsMenu(Menu menu) {
56          return super.onCreateOptionsMenu(menu);
57      }
```

Add the following code to the method:

```
55      @Override
56      public boolean onCreateOptionsMenu(Menu menu) {
57          MenuInflater inflater = getMenuInflater();
58          inflater.inflate(R.menu.nav_menu,menu);
59          return true;
60      }
```

In the code above, the MenuInflater will inflate the nav_menu.xml file we created.

Now we will override the onOptionsItemSelected() method. This method will provide the operations for the selected item.

```
64      @Override
65      public boolean onOptionsItemSelected(@NonNull MenuItem item) {
66          return super.onOptionsItemSelected(item);
67      }
```

Add the following code:

```
84      @Override
85      public boolean onOptionsItemSelected(@NonNull MenuItem item) {
86          switch (item.getItemId()){
87              case R.id.add:
88                  startActivity(new Intent( packageContext: MainActivity.this,Add.class));
89                  break;
90              case R.id.deleteAll:
91                  break;
92          }
93          return super.onOptionsItemSelected(item);
94      }
```

Let's move to our Add.java class.

Firstly, let's call get access to our components in activity_add.xml from our activity.

Add the following variables as class variables in your Add class.

```
14  public class Add extends AppCompatActivity {
15
16      private EditText bookName;
17      private EditText author;
18      private Spinner spinner;
19      private RadioGroup radioGroup;
20
```

Now let's get the components by using findViewById();

```
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28          super.onCreate(savedInstanceState);
29          setContentView(R.layout.activity_add);
30
31          bookName = findViewById(R.id.bookEditText);
32          author = findViewById(R.id.authorEditText);
33          spinner = findViewById(R.id.spinner);
34          radioGroup = findViewById(R.id.radioGroup);
35      }
```

We will now create a method that will get the input from our layout and save it in Intent. The if statement above ensures that the user has filled all the required fields. Below the if statement, we will accept the input and save it in our database. We us startActivityForResult() method, means start activity from MainActivity then we can send data back to our MainActivity once the Add activity is closed. The MainActivity will receive this data through intent.

We will create constants for the Intent in our Add class. Add the following code:

```
14  public class Add extends AppCompatActivity {
15
16      public static final String EXTRA_BOOK = "com.example.shivangi.myreadlist.EXTRA_BOOK";
17      public static final String EXTRA_AUTHOR = "com.example.shivangi.myreadlist.EXTRA_AUTHOR";
18      public static final String EXTRA_CATEGORY = "com.example.shivangi.myreadlist.EXTRA_CATEGORY";
19      public static final String EXTRA_STATUS = "com.example.shivangi.myreadlist.EXTRA_STATUS";
20
```

Now add the following code in the add method.

```
37      public void add(){
38          String bkName = bookName.getText().toString();
39          String bkAuthor = author.getText().toString();
40          String category = spinner.getSelectedItem().toString();
41
42          int radBtnID = radioGroup.getCheckedRadioButtonId();
43          RadioButton radioButton = findViewById(radBtnID);
44          String status = radioButton.getText().toString();
45
46          if(bkName.isEmpty() || bkAuthor.isEmpty() || category.isEmpty() || status.isEmpty()){
47              Toast.makeText( context: this, text: "Please insert all fields!",Toast.LENGTH_SHORT).show();
48              return;
49          }
50
51          Intent data = new Intent();
52          data.putExtra(EXTRA_BOOK,bkName);
53          data.putExtra(EXTRA_AUTHOR,bkAuthor);
54          data.putExtra(EXTRA_CATEGORY,category);
55          data.putExtra(EXTRA_STATUS,status);
56
57          setResult(RESULT_OK,data);
58          finish();
59      }
```

We want to save the data when the add button is clicked. Add the following button for the onClick function of the button.

```
37          Button addBtn = findViewById(R.id.btnSecondAdd);
38          addBtn.setOnClickListener(new View.OnClickListener() {
39              @Override
40              public void onClick(View v) {
41                  add();
42              }
43          });
```

Your final Add class looks as follows:

MainActivity.java | Add.java

```java
package com.example.shivangi.myreadlist;

import ...

public class Add extends AppCompatActivity {

    public static final String EXTRA_BOOK = "com.example.shivangi.myreadlist.EXTRA_BOOK";
    public static final String EXTRA_AUTHOR = "com.example.shivangi.myreadlist.EXTRA_AUTHOR";
    public static final String EXTRA_CATEGORY = "com.example.shivangi.myreadlist.EXTRA_CATEGORY";
    public static final String EXTRA_STATUS = "com.example.shivangi.myreadlist.EXTRA_STATUS";

    private EditText bookName;
    private EditText author;
    private Spinner spinner;
    private RadioGroup radioGroup;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add);

        bookName = findViewById(R.id.bookEditText);
        author = findViewById(R.id.authorEditText);
        spinner = findViewById(R.id.spinner);
        radioGroup = findViewById(R.id.radioGroup);

        Button addBtn = findViewById(R.id.btnSecondAdd);
        addBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                add();
            }
        });
    }
```

```
46      public void add(){
47          String bkName = bookName.getText().toString();
48          String bkAuthor = author.getText().toString();
49          String category = spinner.getSelectedItem().toString();
50
51          int radBtnID = radioGroup.getCheckedRadioButtonId();
52          RadioButton radioButton = findViewById(radBtnID);
53          String status = radioButton.getText().toString();
54
55          if(bkName.isEmpty() || bkAuthor.isEmpty() || category.isEmpty() || status.isEmpty()){
56              Toast.makeText( context: this, text: "Please insert all fields!",Toast.LENGTH_SHORT).show();
57              return;
58          }
59
60          Intent data = new Intent();
61          data.putExtra(EXTRA_BOOK,bkName);
62          data.putExtra(EXTRA_AUTHOR,bkAuthor);
63          data.putExtra(EXTRA_CATEGORY,category);
64          data.putExtra(EXTRA_STATUS,status);
65
66          setResult(RESULT_OK,data);
67          finish();
68      }
69  }
```

We will create a constant for the second parameter in our startActivityForResult() method and change our operation for add item in the menu.

```
18  public class MainActivity extends AppCompatActivity {
19
20      private BookViewModel viewModel;
21      public static final int REQUEST_CODE = 1;
22
```

```
57          switch (item.getItemId()){
58              case R.id.add:
59                  Intent intent = new Intent( packageContext: MainActivity.this, Add.class);
60                  startActivityForResult(intent,REQUEST_CODE);
61                  break;
```

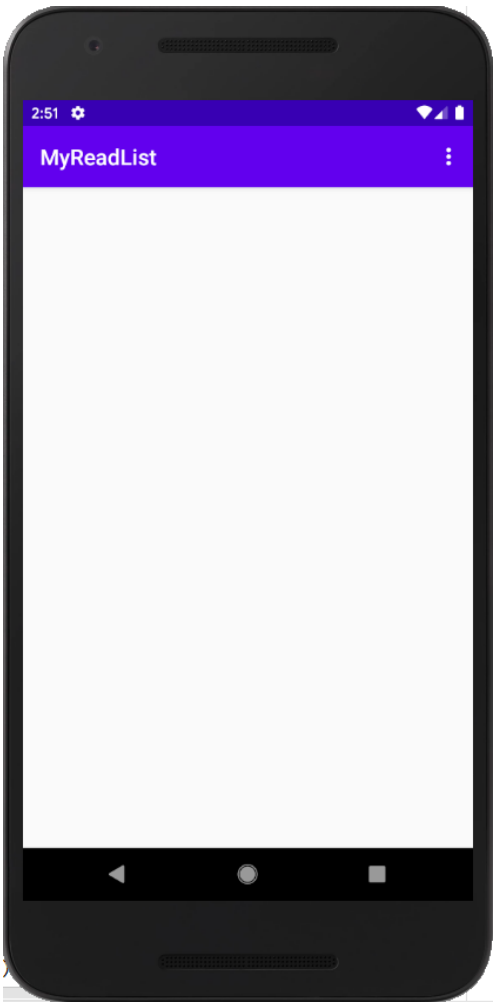
We override the onActivityResult() method to get the result back from the startActivityForResult().

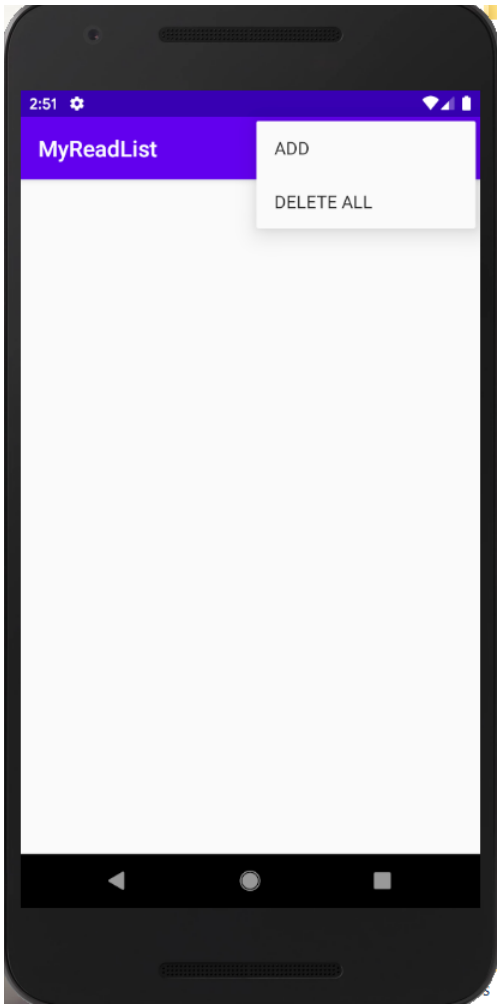
In the code below, we use if statement to determines if everything was performed correctly and is known with the use of a toast message. We get the strings from our intent and save it to out database.

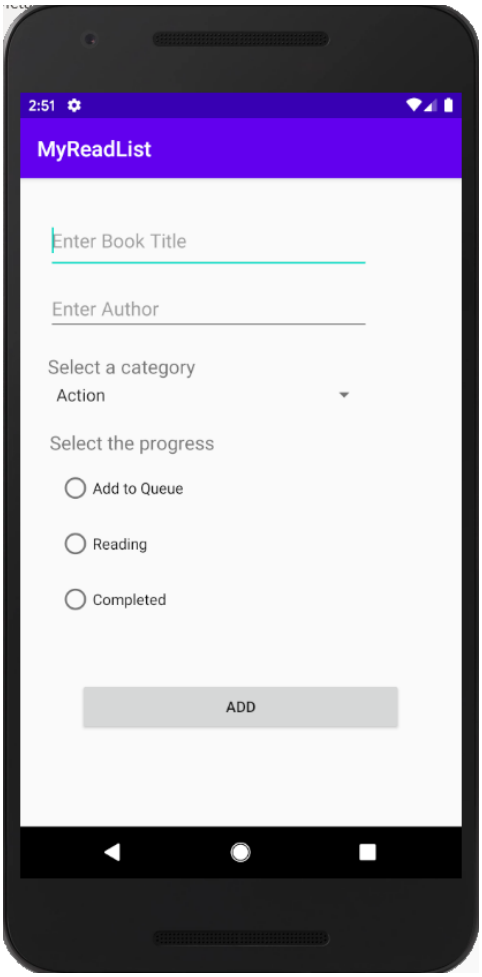
Add the following code below your onCreate method in the MainActivity.

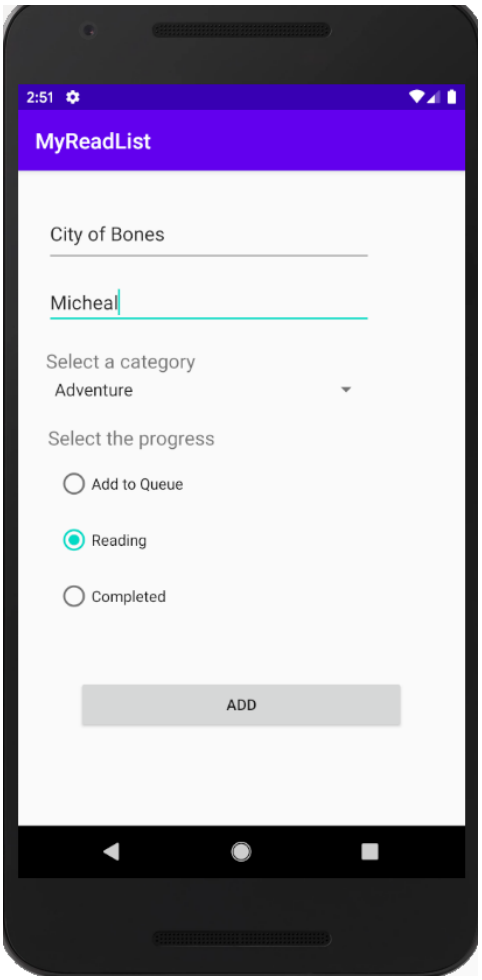
```
53      public void onActivityResult(int requestCode, int resultCode, Intent data) {
54          super.onActivityResult(requestCode, resultCode, data);
55
56          if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
57              String book = data.getStringExtra(Add.EXTRA_BOOK);
58              String author = data.getStringExtra(Add.EXTRA_AUTHOR);
59              String cat = data.getStringExtra(Add.EXTRA_CATEGORY);
60              String status = data.getStringExtra(Add.EXTRA_STATUS);
61
62              Book books = new Book(book,author,cat,status);
63              viewModel.insert(books);
64
65              Toast.makeText( context: this, text: "Book Saved",Toast.LENGTH_LONG).show();
66          } else {
67              Toast.makeText( context: this, text: "Book not Saved",Toast.LENGTH_LONG).show();
68          }
69      }
```

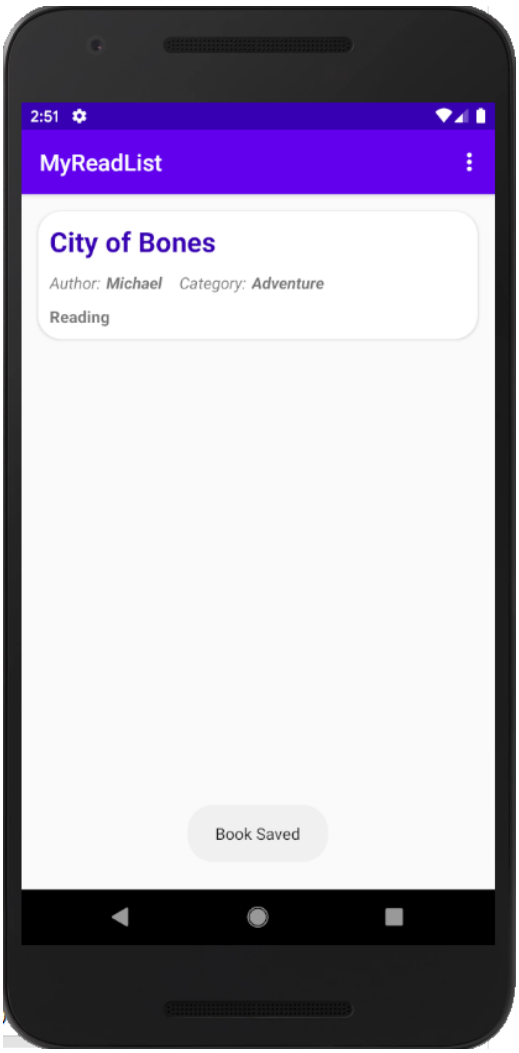
Let’s run the app.



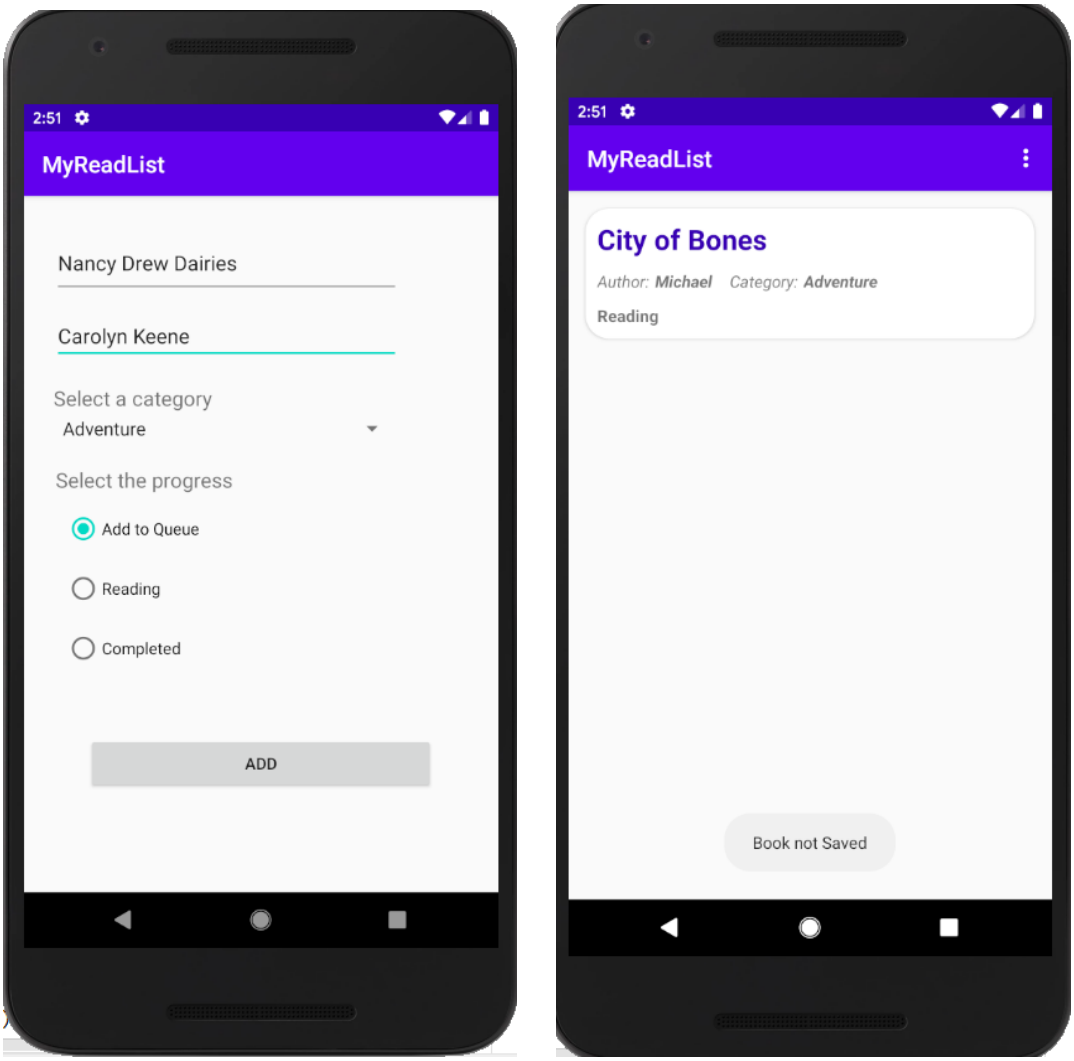



2:51
MyReadList
Enter Book Title
Enter Author
Select a category
Action
Select the progress
Add to Queue
Reading
Completed
ADD


2:51
MyReadList
City of Bones
Micheal
Select a category
Adventure
Select the progress
Add to Queue
Reading
Completed
ADD


2:51
MyReadList
City of Bones
Author: Michael Category: Adventure
Reading
Book Saved

When you click the back button, the data will not be saved and a toast will appear informing the user.



We will now move on to creating swipe function to delete data.

Let's add a method in our adapter to get the position of an item. Add the following code in the BookAdapter class below the setBooks method.

```
45      public  Book getBookAt(int pos){
46          return books.get(pos);
47      }
```

Now go to your MainActivity class. A ItemTouchHelper class will help our items in recycler view swipeable. Add the following code in the onCreate method.

```
48      new ItemTouchHelper(new ItemTouchHelper.SimpleCallback() {
49          @Override
50          public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder, @NonNull RecyclerView.ViewHolder target) {
51              return false;
52          }
53
54          @Override
55          public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
56
57          }
58      });
```

The SimpleCallBack method takes an integer for drag and drop as its first parameter and the direction of swipe as second parameter. Add the following parameters.

```
48      new ItemTouchHelper(new ItemTouchHelper.SimpleCallback( dragDirs: 0,ItemTouchHelper.LEFT) {
49          @Override
50          public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder,
51                                @NonNull RecyclerView.ViewHolder target) {
52              return false;
53          }
```

We won't make any changes in our onMove method as we won't be using this method.

We will call the delete method from our viewModel and use the adapter to get the position of the item in order to perform the delete operation. A toast message will appear when the data is deleted.

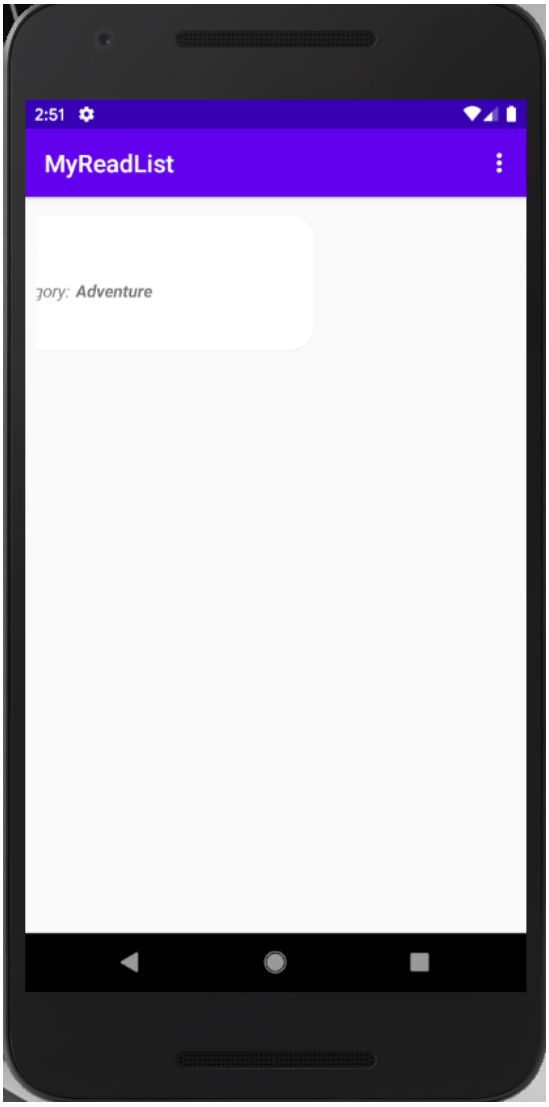
```
55          @Override
56          public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
57              viewModel.delete(adapter.getBookAt(viewHolder.getAdapterPosition()));
58              Toast.makeText( context: MainActivity.this, text: "Book Deleted",Toast.LENGTH_SHORT).show();
59          }
```

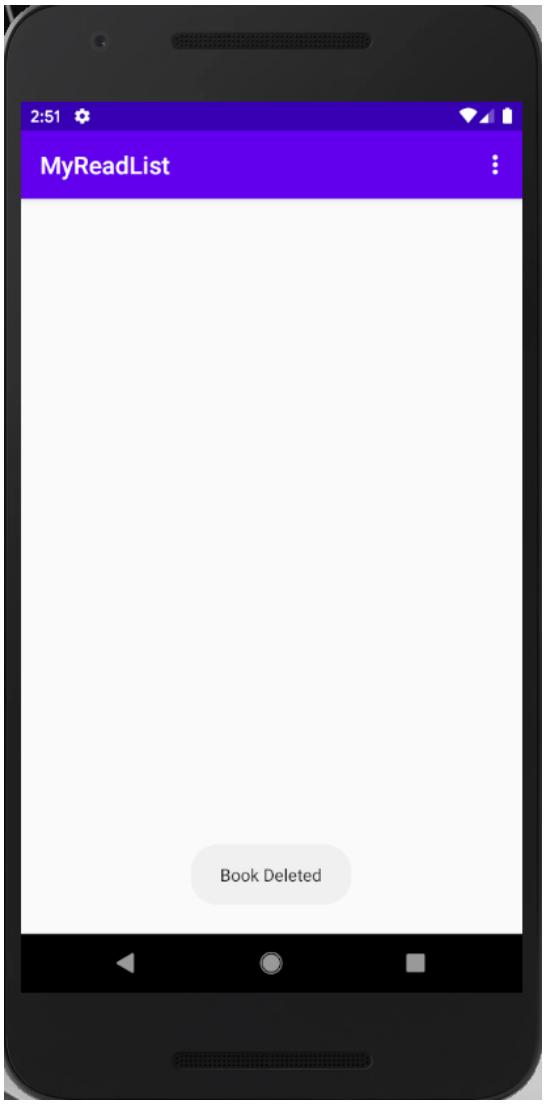
Then we will attach the ItemTouchHelper to our recycler view.

```
48      new ItemTouchHelper(new ItemTouchHelper.SimpleCallback( dragDirs: 0,ItemTouchHelper.LEFT) {
49          @Override
50          public boolean onMove(@NonNull RecyclerView recyclerView, @NonNull RecyclerView.ViewHolder viewHolder,
51                                @NonNull RecyclerView.ViewHolder target) {
52              return false;
53          }
54
55          @Override
56          public void onSwiped(@NonNull RecyclerView.ViewHolder viewHolder, int direction) {
57              viewModel.delete(adapter.getBookAt(viewHolder.getAdapterPosition()));
58              Toast.makeText( context: MainActivity.this, text: "Book Deleted",Toast.LENGTH_SHORT).show();
59          }
60      }).attachToRecyclerView(recyclerView);
```

Let’s run the app.

Swipe the data to the left. A toast message will appear at the bottom.



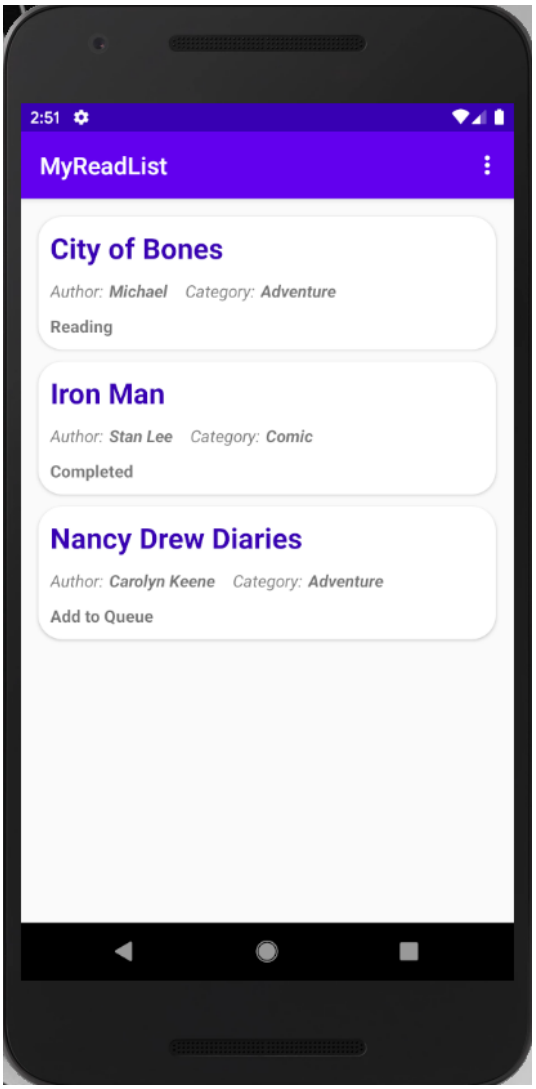


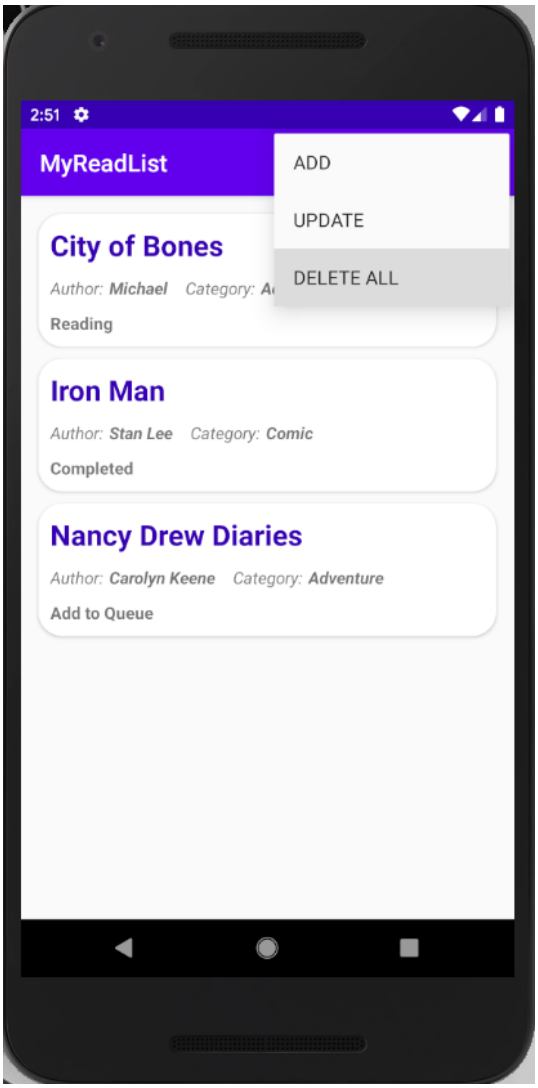
We also need to add an action to delete all data.

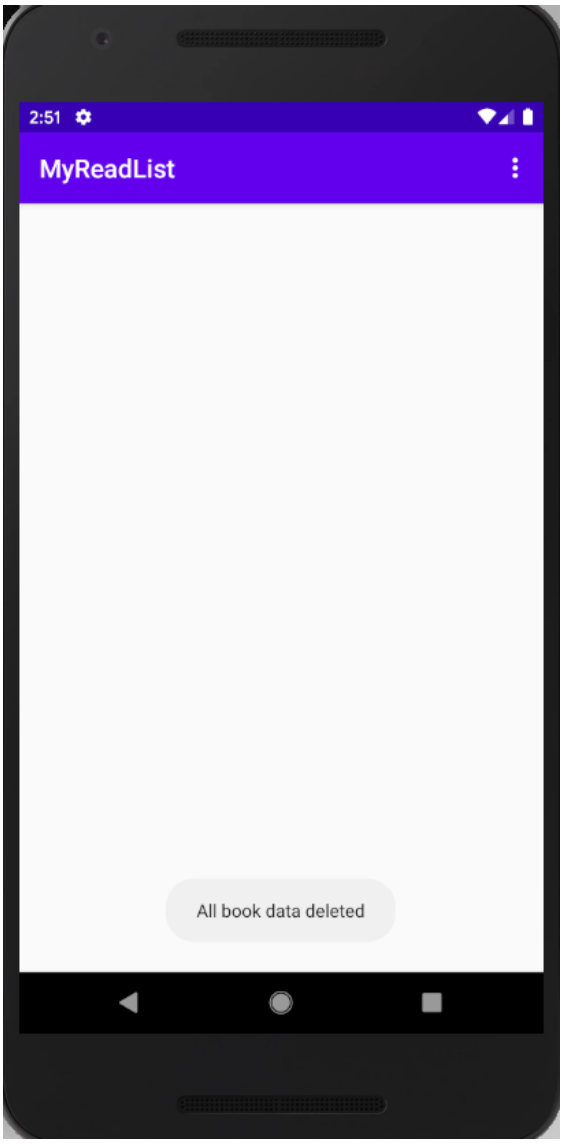
Add the following code in the onOptionsItemSelected method. We are just calling the deleteAll method from our viewModel to perform the operation.

```
84      @Override
85      public boolean onOptionsItemSelected(@NonNull MenuItem item) {
86          switch (item.getItemId()){
87              case R.id.add:
88                  Intent intent = new Intent( packageContext: MainActivity.this, Add.class);
89                  startActivityForResult(intent,REQUEST_CODE);
90                  break;
91              case R.id.deleteAll:
92                  viewModel.deleteAll();
93                  Toast.makeText( context: this, text: "All book data deleted",Toast.LENGTH_SHORT).show();
94                  break;
95          }
96          return super.onOptionsItemSelected(item);
97      }
```

Run your app and add a list of data.






2:51
MyReadList
All book data deleted

Let's create update operations.

We will need to create an onClick event for the items. In our BookAdapter, we will create an interface and define a method. Therefore, every time you implement the interface you also implement the method.

To do this add the following code at the end of the BookAdapter class.

```
73      public interface onItemClickListener{
74          void onItemClick(Book book);
75      }
```

We also create a setOnClickListener() method and pass our ItemClickListener in it. This method is used to call the adapter in the onItemClickListener interface. We will need a class variable for onItemClickListener.

Add the following member variable in the BookAdapter.

```
13  public class BookAdapter extends RecyclerView.Adapter <BookAdapter.BookHolder> {
14
15      private List<Book> books = new ArrayList<>();
16      private onItemClickListener listener;
```

Add the following code at the bottom of the BookAdapter class.

```
80      public void setOnClickListener(onItemClickListener listener){
81          this.listener = listener;
82      }
```

To catch the onclick, we set the onClickListener to our view.

Add the following code in your BookHolder constructor.

```
57      public BookHolder(@NonNull View itemView) {
58          super(itemView);
59          bookName = itemView.findViewById(R.id.bookName);
60          author = itemView.findViewById(R.id.authorName);
61          category = itemView.findViewById(R.id.bkCategory);
62          status = itemView.findViewById(R.id.bkStatus);
63
64          itemView.setOnClickListener(new View.OnClickListener() {
65              @Override
66              public void onClick(View v) {
67                  int pos = getAdapterPosition();
68                  if(listener != null && pos != RecyclerView.NO_POSITION){
69                      listener.onItemClick(books.get(pos));
70                  }
71              }
72          });
73      }
```

In the above code, we are setting the onClick listener to our view and getting the position of the item clicked.

Now we will call the setOnClickListener method in the MainActivity and put data into the intent. In this intent, we will also send the ID so that room knows which item is being updated. First, let's create another constant for ID. Add the following constant in your Add class.

```
15  public class Add extends AppCompatActivity {
16
17      public static final String EXTRA_ID = "com.example.shivangi.myreadlist.EXTRA_ID";
18      public static final String EXTRA_BOOK = "com.example.shivangi.myreadlist.EXTRA_BOOK";
19      public static final String EXTRA_AUTHOR = "com.example.shivangi.myreadlist.EXTRA_AUTHOR";
20      public static final String EXTRA_CATEGORY = "com.example.shivangi.myreadlist.EXTRA_CATEGORY";
21      public static final String EXTRA_STATUS = "com.example.shivangi.myreadlist.EXTRA_STATUS";
```

Now let's store the data in the intent. We will create a new request code variable for this operation. Go to MainActivity file and add the following member variable:

```
23  public class MainActivity extends AppCompatActivity {
24
25      private BookViewModel viewModel;
26      public static final int REQUEST_CODE = 1;
27      public static final int EDIT_REQUEST_CODE = 2;
28
```

Add the following code at the end of onCreate method.

```
63          adapter.setOnClickListener(new BookAdapter.onItemClickListener() {
64              @Override
65              public void onItemClick(Book book) {
66                  Intent intent = new Intent( packageContext: MainActivity.this, Add.class);
67                  intent.putExtra(Add.EXTRA_ID,book.getId());
68                  intent.putExtra(Add.EXTRA_BOOK,book.getTitle());
69                  intent.putExtra(Add.EXTRA_AUTHOR,book.getAuthor());
70                  intent.putExtra(Add.EXTRA_CATEGORY,book.getCat());
71                  intent.putExtra(Add.EXTRA_STATUS,book.getStatus());
72                  startActivityForResult(intent,EDIT_REQUEST_CODE);
73              }
74          });
```

Now we will get the intent data in our Add class and display in the edit texts of our activity_add.xml.

So in your Add class, add the following code to get the data. Note that we are using the same layout for both add and update.

```
47          Intent intent = getIntent();
48
49          if(intent.hasExtra(EXTRA_ID)){
50              addBtn.setText("UPDATE");
51              bookName.setText(intent.getStringExtra(EXTRA_BOOK));
52              author.setText(intent.getStringExtra(EXTRA_AUTHOR));
53              String spin = intent.getStringExtra(EXTRA_CATEGORY);
54
55              int count = spinner.getCount();
56              for(int i = 0; i < count; i++){
57                  String string = spinner.toString();
58                  if(string == spin){
59                      spinner.setSelection(i);
60                      return;
61                  }
62              }
63
64              String value = intent.getStringExtra(EXTRA_STATUS);
65
66              int rad = radioGroup.getChildCount();
67              for(int i = 0; i < rad; i++){
68                  radioButton = (RadioButton) radioGroup.getChildAt(i);
69                  String radioBtn = radioButton.toString();
70                  if(radioBtn == value){
71                      radioGroup.check(i);
72                  }
73              }
74          }
75          else {
76              addBtn.setText("ADD");
77          }
```

Now we want to save the updated data. We will create a variable in our add() method to get the ID. The getIntExtra() method will take two parameters, one is the name of the key and the other is a default value. We set the default value as -1 because we wont have an ID of the value -1.

The if statement will verify that the ID is valid.

Add the following code in the add() method.

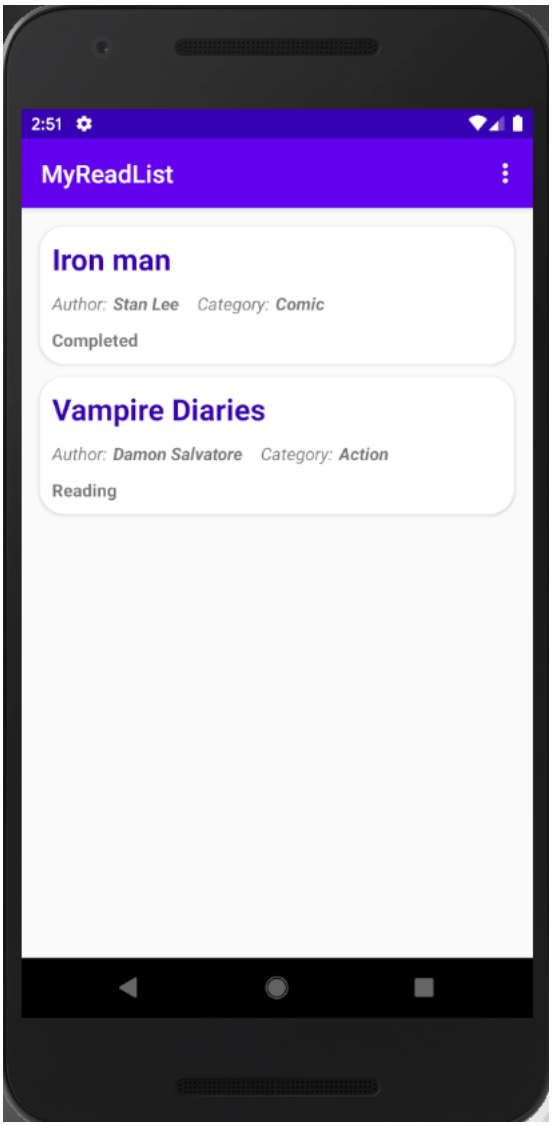
```
62      public void add(){
63          String bkName = bookName.getText().toString();
64          String bkAuthor = author.getText().toString();
65          String category = spinner.getSelectedItem().toString();
66
67          int radBtnID = radioGroup.getCheckedRadioButtonId();
68          RadioButton radioButton = findViewById(radBtnID);
69          String status = radioButton.getText().toString();
70
71          if(bkName.isEmpty() || bkAuthor.isEmpty() || category.isEmpty() || status.isEmpty()){
72              Toast.makeText( context: this, text: "Please insert all fields!",Toast.LENGTH_SHORT).show();
73              return;
74          }
75
76          Intent data = new Intent();
77          data.putExtra(EXTRA_BOOK,bkName);
78          data.putExtra(EXTRA_AUTHOR,bkAuthor);
79          data.putExtra(EXTRA_CATEGORY,category);
80          data.putExtra(EXTRA_STATUS,status);
81
82          int id = getIntent().getIntExtra(EXTRA_ID,  defaultValue: -1);
83
84          if(id != -1) {
85              data.putExtra(EXTRA_ID,id);
86          }
87
88          setResult(RESULT_OK,data);
89          finish();
90      }
```

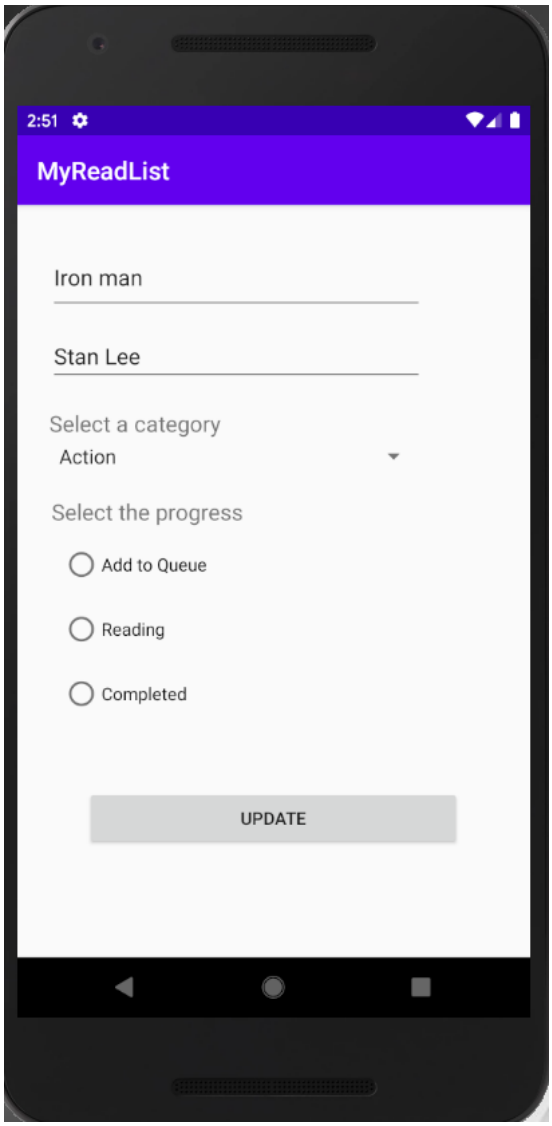
In our onActivityResult() method, we will add another if statement that will get the ID and verify it. If the ID is valid, we will get the other data and save it to the Book object. We will set the ID to identify the data for update.

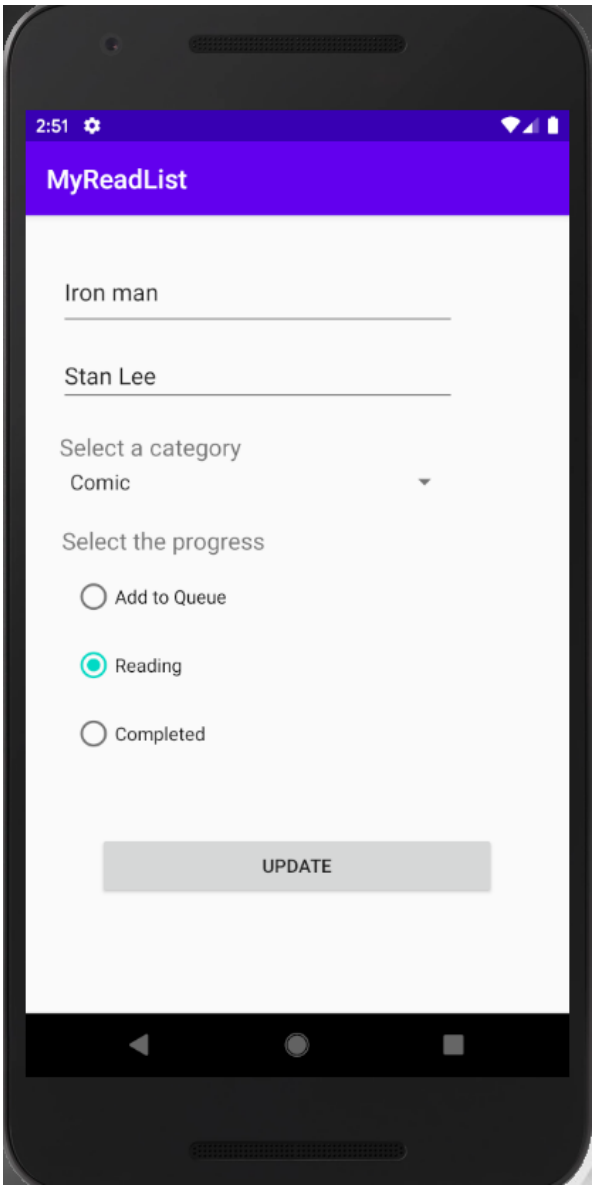
Go to MainActivity class and the following code.

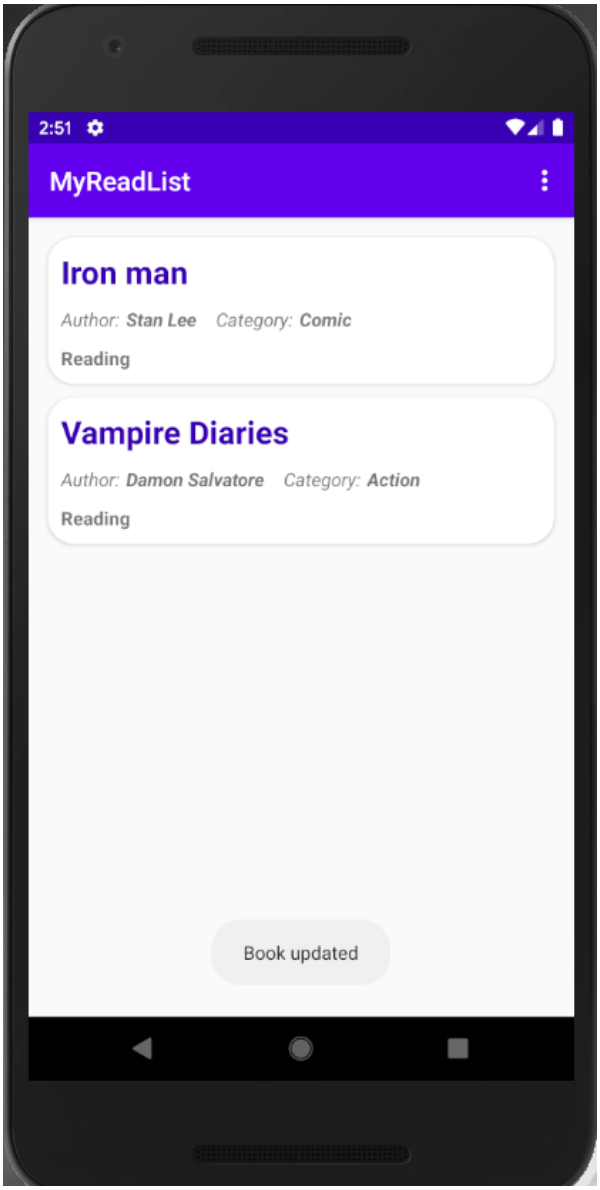
```
101    public void onActivityResult(int requestCode, int resultCode, Intent data) {
102        super.onActivityResult(requestCode, resultCode, data);
103
104        if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
105            String book = data.getStringExtra(Add.EXTRA_BOOK);
106            String author = data.getStringExtra(Add.EXTRA_AUTHOR);
107            String cat = data.getStringExtra(Add.EXTRA_CATEGORY);
108            String status = data.getStringExtra(Add.EXTRA_STATUS);
109
110            Book books = new Book(book,author,cat,status);
111            viewModel.insert(books);
112
113            Toast.makeText( context: this, text: "Book Saved",Toast.LENGTH_LONG).show();
114        } else if(requestCode == EDIT_REQUEST_CODE && resultCode == RESULT_OK){
115            int id = data.getIntExtra(Add.EXTRA_ID, defaultValue: -1);
116
117            if(id == -1){
118                Toast.makeText( context: this, text: "Book not updated",Toast.LENGTH_LONG).show();
119                return;
120            }
121
122            String book = data.getStringExtra(Add.EXTRA_BOOK);
123            String author = data.getStringExtra(Add.EXTRA_AUTHOR);
124            String cat = data.getStringExtra(Add.EXTRA_CATEGORY);
125            String status = data.getStringExtra(Add.EXTRA_STATUS);
126
127            Book books = new Book(book,author,cat,status);
128            books.setId(id);
129            viewModel.update(books);
130
131            Toast.makeText( context: this, text: "Book updated",Toast.LENGTH_LONG).show();
132        }
133        else {
134            Toast.makeText( context: this, text: "Book not Saved",Toast.LENGTH_LONG).show();
```

Now run the app.






2:51
MyReadList
Iron man
Stan Lee
Select a category
Comic
Select the progress
Add to Queue
Reading
Completed
UPDATE


2:51
MyReadList
Iron man
Author: Stan Lee Category: Comic
Reading
Vampire Diaries
Author: Damon Salvatore Category: Action
Reading
Book updated

RESOURCES

https://developer.android.com/topic/libraries/architecture/adding-components.html

https://codelabs.developers.google.com/codelabs/android-training-room-delete-data/#2

https://codelabs.developers.google.com/codelabs/android-training-room-delete-data/index.html?index=..%2F..android-training#13

https://www.youtube.com/watch?v=dYbbTGiZ2sA