CS3013 Spring 2025 – Program 3 – MovieList App

1. OVERALL REQUIREMENTS

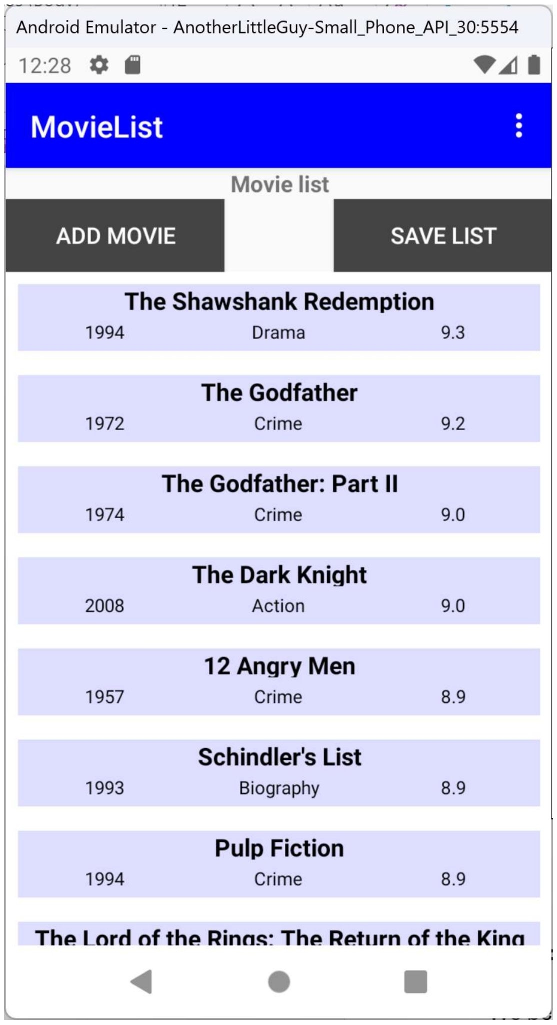
Write, test and deliver an app named **MovieList** – which tracks a list of movies and their ratings – on a scale zero up to 10. The app should follow all the specifications below. In doing this, you are NOT to use AI mechanisms at all. By turning in this assignment, you are asserting/confirming that the work presented in 100% your own work and not that of any AI mechanism. To do this assignment, you are to use only mechanisms presented in our class lectures. Points will be taken off for the inclusion of materials from outside our lectures. All work on this program can be done with mechanisms covered in class. *Do this assignment inspired by and with the guidance of the apps we did in class - especially the BookRecycle app.*

We want an app that starts with an initial list of movies and does all the following:

A. Visually **display the list** in a vertical scrolling alignment
B. For each movie depicted in the list: **the movie title, year of release, movie genre, and rating** must be displayed together as one unit in the visually scrollable list.
C. The app must support the ability to **add a movie to the list**. This capability is supplied by a button which brings up a second screen that allows a user to enter all four fields of data (title, year, genre, rating) for an additional movie and then allows submission of that data into the list to add the new movie to the end of the list. The button should be labeled: "ADD MOVIE"
D. The app must support the ability to **delete a movie from the list** by swiping to the RIGHT the view item of that movie in the visual list. Left swipes and other moves of a view are not responded to.
E. The app must support the ability to **save the current list content to a file named MOVIELIST.csv** – from which it can read in the data at its next initialization. This file is the official version of the movie list data and reflects all changes in the list made in the run of the app. Note that whatever the state of the visible list – that is the list that is saved to the file – in exactly the same order as depicted on the screen. When the app first comes up, it must read the file MOVIELIST.csv and populate its internal data and the list on the screen accordingly. The save action should be driven by a button on the main screen – the button is labeled "SAVE LIST"
F. **[OPTIONAL]** The app must support the ability to **sort the list** and change the visible list according to the desired sort. It is required that the list can be sorted according to year, or according to rating, or according to genre. At the right end of the app bar there should be an options icon (AKA overflow icon) of three vertical dots. Clicking

on this should present a menu of three options: Sort by Rating, Sort by Year, Sort by Genre. Depending on which of these is clicked, the app should then sort the list according to the chosen property and the visible displayed list should change accordingly. The "official list" is thereby changed.

One *possible* initial appearance of the app is shown below.


Android Emulator - AnotherLittleGuy-Small_Phone_API_30:5554

12:28

MovieList

Movie list

ADD MOVIE

SAVE LIST

The Shawshank Redemption		
1994	Drama	9.3
The Godfather		
1972	Crime	9.2
The Godfather: Part II		
1974	Crime	9.0
The Dark Knight		
2008	Action	9.0
12 Angry Men		
1957	Crime	8.9
Schindler's List		
1993	Biography	8.9
Pulp Fiction		
1994	Crime	8.9
The Lord of the Rings: The Return of the King		

---

2. The File Data for this app

We begin with a csv file – **Top20.csv** – which you are provided - listing the top 20 rated movies – this file lists the title, the release-year, the genre of each movie – as well as a rating for each movie. The rating **r** of each movie is in the range: 0 ≤ **r** < 10. In this file, the movie list is sorted initially according to *rating* . Keep a safe copy of this file under its name.

The file your program will actually use is named **MOVIELIST.csv** – and, initially, MOVIELIST.csv should be populated with the contents of Top20.csv. so – to start – make a copy of Top20.csv into the file MOVIELIST.csv. Note that both Top20.csv and MOVIELIST.csv are comma separated. Thus, each line consists of four Strings separated by commas: title, year, genre, rating. Note that a rating vale is never 10 – so that the ratings can be sorted as Strings and come out in the proper order.

When your program first comes up, it must read in this file and populate its internal storage for the data. This provides the initial movie list to be displayed. Your program may alter the ordering and content of the list as it runs. And, later saves of the list will alter the content of the file MOVIELIST.csv.

File Storage needed: For simplicity, name the package for this app **com.example.movielist**. In the **Device Explorer**, the file storage area for your app will be in the folder: **/data/data/com.example.movielist/files.** Refer to this as the "target directory." That is where your file MOVIELIST.csv should reside. Once you have created an initial MOVIELIST.csv on your computer by copying Top20.csv, then you want to put it in the proper location. Do this by right-clicking on the files folder, and then choosing the Upload option. Follow the prompts to find your MOVIELIST.csv file and move it into the proper target directory.

---

3. Internal Data for this App

Each movie is internally stored as a Movie type object – based on a Kotlin class **Movie** that you define in the file Movie.kt. For simplicity and ease of coding – use only String type attributes/properties for Movie objects – name them: title, year, genre, and rating. Note that rating is a String type field. Each time a new movie is encountered, we create a Movie type object for it. These Movie type objects are stored in an ArrayList (carrying generic type Movie) called **movieList**. It is important to note that, while the app is running, there is the list of movies the user sees on the screen – *and* there is the ArrayList of all the list's internal Movie data objects. You will create a **MovieAdapter** class whose job it is to keep the list on the screen aligned and faithful to the current contents of the internal ArrayList. You will have an XML RecyclerView in your main XML – and it is this RecyclerView that presents the visual list. So, the RecyclerView and the Adapter and the actual ArrayList **movieList** are to be kept aligned at all times. When the user presses the "SAVE LIST" button, the external file MOVIELIST.csv is then forced into agreement with the current content in the ArrayList.

You will also have a class **MovieViewHolder** that works at the level of an individual movie – binding the data of a given movie to an actual View depicted on the screen. The data of an individual movie is saved in a Movie object that is in the ArrayList movieList. When that movie is actually rendered on the screen and visible there - then **a MovieViewHolder** object has been bound (bind-ed?) to the data of that particular movie – and there is a visible View on the screen laid out according to the XML you have given in the file **movie_item.xml**. As we did in class, make the **MovieViewHolder** class an inner class within the **MovieAdapter** class.

---

4. Component Naming

Use the naming conventions listed below:

Component Name	**Description**
Movie	The name of the class that models any given movie
movieList	The arraylist that stores the Movie objects for all the movies in the current list.
Class name: MovieAdapter Instance name: movieAdapter	The overall adaptor needed for recycler view on the list
MovieViewHolder	The ViewHolder class that holds/binds the datafor an individual Move object.
titleTextView yearTextView genreTextView ratingTextView	IDs of the 4 TextViews in movie_item.xml
recyclerView	The ID of the RecyclerView element/tag ni the file activity_main.xml
readFile()	Function in MainActivity class that reads the file MOVIELIST.csv in and uses its data to populate the ArrayList movieList.
writeFile()	Function in MainActivity class that writes the Movie items in the ArrayList movieList into the file MOVIELIST.csv
fun saveList(v : View)	Function in MainActivity class that is called when the “SAVE LIST” button is clicked – it simply calls the function writeFile()
fun startSecond	Function in MainActivity class that is called when the “ADD MOVIE” button is clicked – it does a startForResult launch of the second activity

AddMovieActivity	Name of the second activity that does the work of adding a new movie

5. Basic Main XML and the RecyclerView

The **activity_main.xml** file should present all buttons needed and text views for labels. Most importantly, it should contain a **RecyclerView** element/tag which will host the actual adapted list. Linear layouts can be used to order and lay out elements of the screen. A horizontal linear layout can be used to set the buttons properly laterally.

6. Adapter and Holder for the basic list handling

Once you have defined the class **Movie**, you create a layout file **movie_item.xml** that expresses exactly how an individual movie – and its 4 pieces of data – are to be laid out on the screen. You can choose to lay this out as done in the above picture. But here you are free to design and lay things out differently. But you must render all 4 pieces of data. This XML file movie_item.xml should give IDs to all the text views according to component naming table above.

Then create a MovieAdapter class in a file **MovieAdapter.kt**. Model it after what we did for BookAdapter.kt. Inside this MovieAdapter class is the MovieViewHolder class which basically encompasses the internal View objects for the 4 fields of the movie_item.xml layout. So, for example, you create a variable for the title view and initialize it:

var titleTextView: TextView

and, in an init { } section:

titleTextView = itemView.findViewById<TextView?>(R.id.titleTextView)

You do the same for the other three fields: year, genre, and rating.

Once the holder is defined, you create the rest of the MovieAdapter class by defining the three functions you need to override:

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MovieViewHolder

override fun onBindViewHolder(holder: MovieViewHolder, position: Int)

override fun getItemCount(): Int

Do these by following closely the analogy with the BookRecycle app and its adapter.

➔➔ Additional Changes to MovieAdapter class for item deletion

There are other functions you will add to the Adapter Kotlin class and file. One is simply for ability to remove an item from the list via the adapter:

fun removeItem(position: Int)

Also, you'll need an inner class inside the adapter class to support swiping to the right. This class has three functions you override – outlined below. Code this exactly as coded for ther BookRecycle app.

```
inner class SwipeToDeleteCallback :
    ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.RIGHT) {

    override fun onMove(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder,
        target: RecyclerView.ViewHolder
    ): Boolean = false

    override fun getMovementFlags(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder
    ) =
        if (viewHolder is MovieViewHolder) {
            makeMovementFlags(
                ItemTouchHelper.ACTION_STATE_IDLE,
                ItemTouchHelper.RIGHT
            ) or makeMovementFlags(
                ItemTouchHelper.ACTION_STATE_SWIPE,
                ItemTouchHelper.RIGHT
            )
        } else {
            0
        }

    override fun onSwiped(viewHolder: RecyclerView.ViewHolder,
direction: Int) {
        val position = viewHolder.adapterPosition
        removeItem(position)
    }
}
```

(***) As a class variable (property) of your MovieAdapter, create a object instance of this inner class and save it in the variable **swipeToDeleteCallback**.

```
// added to enable swipe delete

val swipeToDeleteCallback = SwipeToDeleteCallback()
```

This will allow the main activity to access this when it need to activate the touch helper for swiping. See the line in the next section marked **<TOUCH>.**

Note that there is no special coding needed in the Adapter class to support movie addition.

---

7. MainActivity.kt Development Work

The **MainActivity** class needs to be coded to accomplish the following:

- ⇨ Create and support the ArrayList movieList
- ⇨ Create and track an object instance of the MovieAdapter class – named movieAdapter – instantiated of the ArrayList movieList.
- ⇨ In its **onCreate()** method it must
  - o Define a variable **recyclerView** as the object version of the XML **RecyclerView** element (do this with **findViewById()**)
  - o Set the layout manager for this **RecyclerView** object to something like: LinearLayoutManager(this)
  - o Set the adapter of this **RecyclerView** object to **movieAdapter**.

- ⇨ Support work to allow adding a new Movie to the list. It creates a variable **startForResult** via a call **to registerForActivityResult** to set things up to allow a second activity to pass data back to it. It the MainActivity function startSecond(), it uses the startForResult object to launch the second activity - named **AddMovieActivity -** to do the work of getting the data about a new Movie being added.

- ⇨ **<TOUCH>** Support for deletion of a movie by swiping: Near the bottom of the onCreate() method, add lines below to allow swipe delete

```
    // add lines below to allow swipe delete
    val itemTouchHelper =
ItemTouchHelper(movieAdapter.swipeToDeleteCallback)
```

itemTouchHelper.attachToRecyclerView(recyclerView)

⇨ Support for file reading and writing for the file MOVIELIST.csv.

---

8. Adding a Movie to the List with AddMovieActivity

Just like the BookRecycle case, create a second activity named **AddMovieActivity** to allow users to enter the 4 data values for a new movie they want to put in the list. This class has variables that track the the fieldsof data, It is important to note that this activity's Kotlin code does not in any way use the Movie class. It does not create any movie objects and it only deals in the four String fields title, year, genre, rating and allows the user to enter 4 text values.

Create second activity – **AddMovieActivity** – and design and lay out its XML to allow users to enter the 4 Strings title, year, genre, rating. You need four EditText fields and a submit button. Then code the Kotlin file AddMovieActivity.kt to gather the data from the user and send it back to the calling Main activity. Do this exactly like we did for the BookRecycle app. There we used two functions backToFirst() and setBookInfo() to do this work. So we'd have two functions:

backToFirst() – function retrieves the four text Strings that the user entered out of the four EditText elements of the **activity_add_movie.xml** file - and then pass them in order to the setMovieInfo function.

setMovieInfo() – this function creates a Intent object and packages in it – as key-value pairs – the four Strings title, year, genre, rating. Then it uses that Intent object to do a setResult() call back to the MainActivity. It then calls finish() to end the second activity life.

Here it is best to follow the analogy of the BookRecycle app.

The **MainActivity** then regains control and enters at the lambda code it originally set up at

```
private val startForResult =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { … }
```

That code gets the activity result passed/returned by the second activity and extracts the **Intent** data bundle from it – and then pulls out the four key-value pairs to capture the four pieces of data it needs to make a new **Movie** object. It then makes that new object and adds it into the ArrayList **movieList**. With the underlying ArrayList now changed, it has to alert the adapter to the change with the call: **movieAdapter.notifyDataSetChanged()**.

Again (!) - here it is best to follow the analogy of the BookRecycle app.

---

9. Deleting a Movie from the list

Deletion of a movie occurs via rightward swipe on the screen view of that particular movie. This is supported by mechanisms we reviewed in class for the **BookRecycle** app. The actual deletion of the movie object from the ArrayList occurs by using the function **removeItem(position: Int)** in the Adapter class. What is more interesting is how the swipe is detected and how the determination is made as to which movie's view got swiped. These latter things are handled by the two mechanisms:

A. An instance of the **SwipeToDeleteCallback** inner class defined inside the MovieAdapter class – this is described above in the section:
    i. **➔➔ Additional Changes to MovieAdapter class for item deletion**
B. Work in **MainActivity.kt** to create a "touch helper" object for the swipe – and link it to the **swipeToDeleteCallback** object in the adapter and then to attach the **RecyclerView** object to the touch helper.
    i. val itemTouchHelper = ItemTouchHelper(bookAdapter.swipeToDeleteCallback)
  b. itemTouchHelper.attachToRecyclerView(recyclerView)

---

10. Reading and writing the file MOVIELIST.csv

Review closely the in-class example code for the app **PersistSimple** and the coverage of it in our lecture notes. That code showed how to create two functions **readFile()** and **writeFile()** (both in **MainActivity.kt**) to support the reading in from a file – and the writing out to the same file – for an internal ArrayList of Employee objects. It alsl showed how the **MainActivity**, in its **onCreate()** code, determined where it could find the file it needed and set up a variable pointing to the directory where the file way – so that the read and write functions could access the directory and file.

In our **MovieList** app we want to do exactly the same things for an internal ArrayList of Movie objects and an external file **MOVIELIST.csv**. Follow tihe same paradigm to create functions **readFile()** and **writeFile()** for the movie file and to initlaize things analogously in **onCreate()** of **MainActivity**. It is important to realize that the File-I/O work here in these two functions does NOT in any way interact with the complexities of the Adapter or the ViewHolder. The file I/O to be done here is straight between the array list of Movie objects and the file MOVIELIST.csv.

Remember – once the app is up and has determined the directory for the file and stored that is a variable **myPlace** – it should call **readFile()** to initially populate the ArrayList. Further, any time the "SAVE LIST" button is pushed, the function **writeFile()** should be called.

---

11. Gettting an App Bar (Action Bar) to appear

You do not need do any special/extra work to get an app bar (action bar) for your app. Simply let the themes file do that as default. If you do the optional work of putting an overflow menu at the right end of the action bar for the purposes of sorting, the mechanisms discussed here will work for the themes created app bar. I used the simple themes.xml file below. You can modify it to change appearance.

```xml
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.MovieList" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <item name="colorPrimary">#0000ff</item>
    <item name="colorOnPrimary">#ffff00</item>
    <item name="colorSurface">#555555</item>
  </style>

  <style name="Theme.MovieList.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
  </style>

  <style name="Theme.MovieList.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />
  <style name="Theme.MovieList.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />
</resources>
```

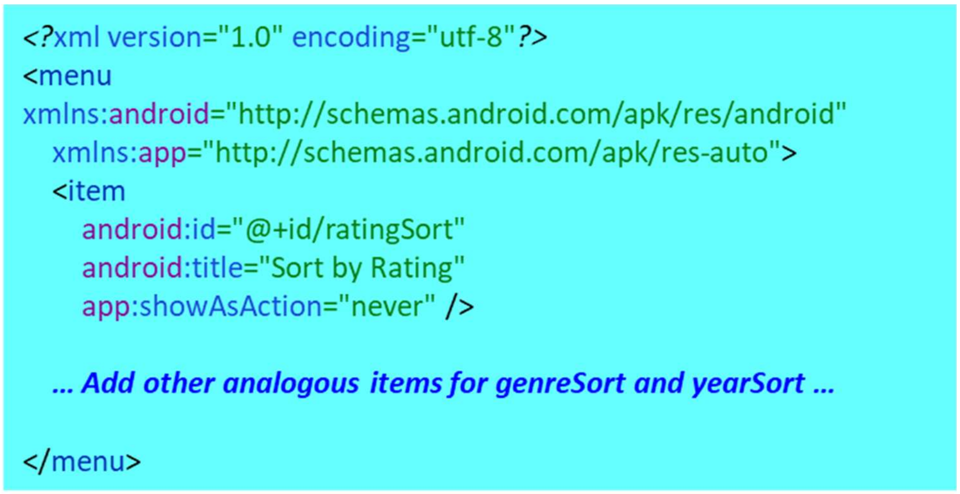
---

12. Getting sorting function in options/overflow menu

To sort the movieList: To sort an ArrayList of objects in Kotlin using **sortedBy**, you can call the sortedBy function and provide a selector function that specifies the property to sort by. For example, if you have a list of objects with a property called *name*, you can sort them like this: val sortedList = myList.sortedBy { it.name }.

To get an options menu to show up at the right end of the app bar:

A. Need to create a menu XML file res/menu – call it **main.xml** – and put into it the items that you want to appear in the menu – the **Allright** program does this. Some possible menu material is given below:

   a. 

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">
   <item
      android:id="@+id/ratingSort"
      android:title="Sort by Rating"
      app:showAsAction="never" />

   ... Add other analogous items for genreSort and yearSort ...

</menu>
```

B. In **MainActivity.kt** – make sure you put in the right imports:

   i. **import android.view.Menu**
      import android.view.MenuItem

C. In **MainActivity.kt** – add the code for two functions:

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
   menuInflater.inflate(R.menu.main, menu)
   return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
   Log.d("MOVIELIST", "options menu")
   when (item.itemId) {
      R.id.ratingSort -> {
         Log.d("MOVIELIST", "onOptions: rating sort")
```

```
            movieList?.sortBy{ it?.rating }
            movieAdapter.notifyDataSetChanged()
          }
          R.id.yearSort -> {
            Log.d("MOVIELIST", "onOptions: year sort")
            movieList?.sortBy{ it?.year }
            movieAdapter.notifyDataSetChanged()
          }
          R.id.genreSort -> {
            Log.d("MOVIELIST", "onOptions: genre sort")
            movieList?.sortBy{ it?.genre }
            movieAdapter.notifyDataSetChanged()
          }
        }
        return super.onOptionsItemSelected(item)
      }
```

Note that lines like: **movieList?.*sortBy*{ it?.rating }**

actually achieve the sorting of the underlying ArrayList **movieList.** But, every time we change the underlying ArrayList, we have to notify the adapter that the "data set" has changed. Hence the line: **movieAdapter.notifyDataSetChanged()**

13. Advice on how to develop this app

Develop the app incrementally – building up and expanding functionality as you go. One possible order is listed below. Develop each step of functionality and test it at that level. Only then go forward to developing the aspects of the next step.

1. **GET TO SEE A BASIC LIST**
   a. Foundations: define your Movie data class in a file Movie.kt - also define how you want each individual movie's view to look/appear in the list display – define a visual layout for a given movie in a layout file movie_item.xml. Set up other foundational elements like strings.xml, or colors.xml, or themes.xml as you need them. Go into MainActvity.kt and declare the movieList ArrayList – follow how we did bookList in the app BookRecycle. Basically it starts as an empty ArrayList of Movie type objects.

   b. Now develop the basic RecyclerView and adapter mechanisms to visually display a basic list. Here you define the **MovieAdapter** and **MovieViewHolder** classes in same named Kotlin files. Follow the analogy to what we did in the BookRecycle app. Then you lay out your XML file activity_main.xml with the textviews and buttons you need – and the critical **RecyclerView** element/tag. It is this RecylcerView that will host the properly adapted list.
   c. Now change the **onCreate()** function in the **MainActivity.kt** Kotlin file to create the necessary adapter object – connect to the ArrayList and then get the RecyclerView object (find by ID) and set its adapter to the adapter object you have created. You need some actual Movies at this point – so create s couple of Movie objects to store in the ArrayList. Then run the app and test and adjust how things look. Test to make sure you see the list of movies you put in when you bring up the app. Artificially grow the list by just ad hoc adding in more created Movie objects yourself in the code.
   d. Once you see the list and how your depiction of it looks – then go on to next step.
2. **DEVELOP THE ABILTY TO ADD A MOVIE TO THE LIST**
   a. Create second activity – **AddMovieActivity** – and design and lay out its XML to allow users to enter the 4 Strings title, year, genre, rating. You need 4 EditText fields and a submit button. Then code the Kotlin file AddMovieActivity.kt to gather the data from the user and send it back to the calling Main activity. Do this exactly like we did for the BookRecycle app.
3. **PUTTING IN SWIPE TO DELETE**
   a. Review the section above: **Deleting a Movie from the list**
   b. Make the changes in the Adapter Kotlin file: **MovieAdapter.kt**, adding in the Inner class **SwipeToDeleteCallbac**k and create an instance of it under the variable **swipeToDeleteCallback**.
   c. Make the changes to **MainActivity.kt** to create a "touch helper" object for the swipe – and link it to the **swipeToDeleteCallback** object in the adapter and then to attach the **RecyclerView** object to the touch helper.

TEST THE APP WITH DUMMY DATA HARD CODED INTO THE APP – put on a few Movie objects into the ArrayList movieList. For example:

```
movieList.add(Movie("The Godfather", "1972", "Crime", 9.2))

movieList.add(Movie("The Dark Knight", "2008", "Action", 9.0))

movieList.add(Movie("The Matrix", "1999", "Science Fiction", 8.7))
```

4. **PUTTING WRITING TO FILE AND READING FROM FILE – SAVE COMMAND**

   To this point, you have the app running with list activities but no file saving. Now follow the example of the **PersistSimple** app and do for the array list movieList what that app does for the emplList. Follow the guidance of the section above:

   Reading and writing the file MOVIELIST.csv

Now build and run the app – testing it for reading and writing the file. Can it initially read in the initial file (which is a copy of Top20.csv)?? Iterate until you fully see the original 20 movies in the list correctly in the RecyclerView. Then explore how well file writing works? Make changes to the list and save them – see if those change get into the file. Use the Device Explorer to display the file in raw text form.

5. **PUTTNG IN OPTIONS MENU CAPABILITY AND SORTING**

Follow the guidance of section 12 above.

6. **Now test the fully integrated app** – mixing all the possible actions on it: read in list on init, display list, add movie, delete movie, sort by any of the 3 criteria, save to the file. Mix these up in various orders and check for correctness!