**Dawson recommendations: Adding a Gui**
**Project Specifications**
**Phase IV: The final chapter!**

**Due: December 6th: Worth 35% of final project mark.**

Phase IV involves primarily adding a graphical user interface to your previous phases. In addition to this, since it is the final phase, a portion of your grade for this phase will come from ensuring that your final version of code is as polished as possible (e.g. commenting, good variable/method names, appropriate use of helper methods, etc)

**Note:** Sometime before the due date, your teacher will provide feedback on the code you've submitted for phase 3, but you shouldn't wait to start phase 4.

First, one teammate should create a new branch labelled phase4 which will serve as your team's staging branch for this phase. You can branch this from the latest code in the master branch, or, if your code from phase 3 has not yet been merged into the master branch, from the latest phase3 code branch.

After that, all teammates should periodically pull phase4 (even if you aren't yet ready to push to it) to keep these protected branches up-to-date locally. Whenever you are about to start a new feature branch, first checkout phase4 and pull in updates of phase4 from origin.

**Make sure that you scroll to the bottom of this document to see the section about serialization, which is ungraded, but required. That is, you do not need to hand it in (so if you miss the deadline on that section, it is OK), but you still need to do it before your final exam.**

## Warning of a potential problem:

Some of you may get a bizarre error in Eclipse the first time you set up your project to use JavaFX:

The error message is Eclipse specific and can happen when you try to use any of the libraries within the JavaFX framework and will look something like:

Access restriction: The type 'Application' is not API (restriction on required library rt.jar)

If you get this error, please follow the step by step at

https://stackoverflow.com/questions/25222811/access-restriction-the-type-application-is-not-api-restriction-on-required-l

For the step at the link where you have to specify a pattern, type ** to allow everything

## Part 0: Getting started with this phase

Make sure that you allocate a proper amount of time to fix the issues you had during previous phases. You should start by reading over the comments given to you. In addition to this, make sure you have thoroughly tested your code to make sure that the fixes are there. And always

keep your eye out for other bugs, even if your teacher---who is human too 😊 -- didn't notice them.

One strong suggestion: It is possible to set everything up so that the two movie specific classes have no logic in them other than calling the super class and then filtering the results based on genre/ensuring that there are enough entries returned. If you do it this way, you will not need to store the Rating[] (or a HashMap) within these two classes. This will make things much simpler in terms of debugging prior parts, so if you have not coded it this way yet, it is strongly recommended you do so first.

## Part 1: Adding Save Functionality:

In previous phases, you wrote methods to load Rating[], Movie[], and Book[] from files. You will now write a generic method that can be used to *save* these things to files using the java.nio library.

- First, create an interface *Saveable* which has one method *toRawString()* which takes nothing as input and returns a String.
- Make the Rating, Movie, and Book classes implement this interface, and have the methods return an identical String to what was given in the first place to the constructor for these methods.

  In other words, after adding these methods, the Rating, Movie, and Book classes should all have the following functionality:

  ```
  String s = "10,1,2.0,1063502716"
  Rating r = new Rating(s);
  String u = r.toRawString();
  s.equals(u); -→ should now be true!
  ```

  There are a few ways to achieve this functionality! You should decide what is best as long as the public method does what it's supposed to! 😊
- Write a generic method *saveToFile* that takes as input an array of any type of any object type extending Saveable (the type should be the parameter) as well as a String representing a path to a file on the computer and a String representing the file *header* (remember the first line of the file contains information about the column names).

  The method should use a StringBuilder (because there are lots of concatenations!) to create a single String that starts with the contents of *header* then has each object in the array presented as a String (using the *toRawString()*) method, and with a new line character between them.

  The idea here is that we would like to be able to do:

  Rating[] ratings = MovieLensFileReader.loadRatings("ratings.csv");
  MovieLensFileReader.saveToFile(ratings, "ratings2.csv",
  "userId,movieId,rating,timestamp");

  After doing that, the file ratings2.csv should look identical to ratings.csv

  Hint: The method header for this method should be:

public static <T extends Saveable> saveToFile(T[] objects, String file, String header)

## Part 2: Adding a GUI

In this last part of the last phase (yay!), we will now add a GUI to the project. See the screenshot at the end of this document as a sample of what functionality you need to have. Although you will not be graded on the quality of the layout, you should be creative and design the GUI in a way that you think looks nice! You may also add additional features if you like (see below for some suggestions).

The gui should have the following (required) functionality.

1. The user should be able to specify whether they intend to load a Movie file or a Book file.
2. There should be two text boxes, where the user can specify the path to the Movie/Book file and also the Rating file.
3. Your program should have a load button, which when clicked will use the values in the textboxes to populate a Movie[] (or Book[]) as well as a Rating[]
4. Your program should have a save button, which when clicked will use the Movie[] (or Book[]) and Rating[] that is loaded into memory and save it into those two files (using the same text box as was used to choose the location to load the file from. Be careful when testing the program not to inadvertently delete the original files!).
5. There should be a section of the UI where the user can rate items. For this, the user should be given a textbox where they can enter a user id. There should be a select list where they can see a list of all the items in the system for them to choose from. And lastly there should be a textbox where they can enter a rating.
6. There should be a button *Add Rating* which when clicked will create a new Rating based on what the user has typed and add it to the stored Rating[] . Since arrays are not resizable, you will need to create a new Rating[] with a bigger size. (You may also choose to use ArrayList here if you like---but there is a lot of code to update) A useful method you can use is java.util.Arrays.copyOf(T[] original, int newSize). This method will return a new array with a size of *newSize*. The original values of *original* will be put into the first *original.length* positions of the array, and then the rest of the array will be "padded" with null values. (Note: if *newSize* is smaller than the original array, the returned array will be truncated)
7. There should be a radio button to allow the user to choose whether to provide personal or popular recommendations as well as a genre field where they can specify the genre. If they are getting Book recommendations, then the genre field will be ignored.
8. Lastly, there should be a "Recommend!" button which when clicked will present the user with the top 10 recommendations. Make sure that your Movie/Book classes have properly overridden the toString() method to only display their names.

After you've done this, try it out on yourself by finding an unused user id (scroll to the bottom of the ratings file) and rate 5-10 movies. Then see what recommendations you get.

**Suggestions for extra things:**

- Use a radio button for ratings (instead of a text field)
- Use images (for example of stars!) for ratings
- Allow users to save and load directly from the recommenders themselves. That is, since we made the recommenders serializable, let a user load a file that is already preprocessed with both ratings and items within it.
- The Books.csv file has a field in it for small_image_url (column W in excel). Modify the Book class to store this url and have a getImageUrl() method). Then, when recommendations are requested, download that image and display a *picture* of the book

instead of the book itself.

For an example of how to download and then display an image using JavaFX, see:

http://www.java2s.com/Code/Java/JavaFX/LoadImagefromURL.htm

- Instead of always creating new recommender objects each time, add functionality to the recommender objects to all you to *add* a rating to an existing recommender. This can lead to some performance improvement as you won't necessarily repeat all the work done in the constructor. (For example, the HashMap of ratings is already built) You will need to recalculate the most similar users and try to use as little code duplication as possible.
- ???

## Part 3: Using Serialization (UNGRADED!)

*This section is not going to be graded. It is in fact not required (although it is recommended!) that you hand it in with your assignment. It is expected that you complete this part before the exam. In other words, it has been removed as a required part of the assignment, due to time / workload management considerations----you'll now have a bit more time to get to this section---- but you are still responsible for knowing the material!*

Change the recommendation classes that you've written to be Serializable so that you do not need to constantly reload everything from scratch.

Remember that if you want a class to be serializable, you will need to do the following:

1. Add the serializeObject and serializeObject methods (verbatim) that we discuss in class/labs to a Utilities class so that you can access it in many places.
2. Add implements Serializable to your class header
3. Add a private static final long `serialVersionUID` to your class. The value should be generated by eclipse
4. Ensure that the super class (if there is one) is serializable (meaning you'll have to do all these steps on that class as well). In addition, the super class *must* have a constructor that takes nothing as input. The body of this constructor can/should be blank, but it has to exist for the serialization to work properly.
5. Ensure that all fields within your class are also serializable (meaning you'll have to do all these steps on that class as well)

Once this is all done, you can write, for example,

```
	Utilities.serializeObject(personalizedRecommender,
"TestSerialization.ser");
```

And it will produce a file TestSerialization.ser for you to use.

Alternatively, if this file already exists, you can write

```
		PersonalizedMovieRecommender p =
(PersonalizedMovieRecommender)Utilities.deserializeObject("TestSerialization.s
er");
```

Do this for both PersonalizedMovieRecommender and PopularMovieRecommender. (Note that since the super class of these is PersonalizedRecommender and PopularRecommender, this means you'll end up doing it for all 4 of them)

**Suggestion:** The serialization of PersonalizedMovieRecommender depends on the serialization of PersonalizedRecommender which in turn depends on the serialization of Rating and Movie/Book. Start by making the Movie and Book classes serializable and checking that you can properly serialize and then deserialize these (you can confirm this by running two different small programs: One should make the file, the other should load from it)

**Once you have done this, you should generate several .ser files (for various combinations of books/movies and popular/personal recommendations) so that you can properly test the GUI pieces later on.**

Then, add functionality to your GUI. The GUI should look at the extension of the file (.ser or .csv) and use that to determine which sort of saving/loading it will use.

**Submission:**

Same instructions as previous phases 😊

**Sample screenshot:**