JAVA PROGRAMMING:
Build a Recommendation System

JIAN MA

2nd October 2021

Table of Contents

Step One: Introducing the Recommender

Video 1: Introduction and Motivation

https://www.coursera.org/lecture/java-programming-recommender/introduction-and-motivation-gIBt5

Video 2: Reading and Storing Data

https://www.coursera.org/lecture/java-programming-recommender/reading-and-storing-data-o3Lyd

Assignment 1: First Ratings

Provided starter files can be found at:

http://www.dukelearntoprogram.com/course5/index.php

In this exercise, you will use the provided classes Movie.java, Rating.java, and Rater.java and read in and store information about movies and ratings of movies by different movie
raters to answer simple questions about both movies and ratings.

For this assignment you will be given three starter files.

The class Movie is a Plain Old Java Object (POJO) class for storing the data about one movie. It includes the following items:

Eight private variables to represent information about a movie including:

-id - a String variable representing the IMDB ID of the movie

-title - a String variable for the movie's title

-year - an integer representing the year

-genres - one String of one or more genres separated by commas

-director - one String of one or more directors of the movie separated by commas

-country - one String of one or more countries the film was made in, separated by commas

-minutes - an integer for the length of the movie

-poster - a String that is a link to an image of the movie poster if one exists, or "N/A" if no poster exists

A constructor with eight parameters to initialize the private variables

Eight getter methods to return the private information such as the method getGenres that returns a String of all the genres for this movie.

A toString method for representing movie information as a String so it can easily be printed.

The class Rating is also a POJO class for storing the data about one rating of an item. It includes

Two private variables to represent information about a rating:

-item - a String description of the item being rated (for this assignment you should use the IMDB ID of the movie being rated)

-value - a double of the actual rating

A constructor with two parameters to initialize the private variables.

Two getter methods getItem and getValue.

A toString method to represent rating information as a String.

A compareTo method to compare this rating with another rating.

The class Rater keeps track of one rater and all their ratings. This class includes:

Two private variables:

-myID - a unique String ID for this rater
-myRatings - an ArrayList of Ratings
A constructor with one parameter of the ID for the rater.

A method addRating that has two parameters, a String named item and a double named rating. A new Rating is created and added to myRatings.

A method getID with no parameters to get the ID of the rater.

A method getRating that has one parameter item. This method returns the double rating of this item if it is in myRatings. Otherwise this method returns -1.

A method numRatings that returns the number of ratings this rater has.

A method getItemsRated that has no parameters. This method returns an ArrayList of Strings representing a list of all the items that have been rated.

---Data files

You'll use several data files for this project. There are multiple versions of each type of file or different sizes.
In creating recommendations you'll need two data files: one of movies, and one of ratings of these movies by different raters.

First there is a CSV file with movie data, the smallest of which is called ratedmovies_short.csv. The other similar files are larger,

so we have created this small file with just a few entries in it that you can use for testing. Each file of this type has a header row as the first line,

first followed by one line for each movie. Shown below is this shorter file that has six lines, the header line first followed by six movies.

The lines are so long that they are wrapping in this document, each movie line is displayed here in two to three lines. For example the first movie has the IMDB number 0006414,

the name of the movie is "Behind the Screen," and the movie was made in 1916 in the USA. The genres for this movie are Short, Comedy, and Romance. The director is Charles Chaplin,

and the length of the film is 30 minutes. The last item is a link to a .jpg image of a poster for the movie if one exists, or "N/A" if there is not one.

ratedmovies_short.csv:

id,title,year,country,genre,director,minutes,poster
0006414,"Behind the Screen",1916,"USA","Short, Comedy, Romance","Charles
  Chaplin",30,"http://ia.media-imdb.com/images/M
  /MV5BMTkyNDYyNTczNF5BMl5BanBnXkFtZTgwMDU2MzAwMzE@._V1_SX300.jpg"
0068646,"The Godfather",1972,"USA","Crime, Drama","Francis Ford Coppola",175
  ,"http://ia.media-imdb.com/images/M
  /MV5BMjEyMjcyNDI4MF5BMl5BanBnXkFtZTcwMDA5Mzg3OA@@._V1_SX300.jpg"
0113277,"Heat",1995,"USA","Action, Crime, Drama","Michael Mann",170,"http://ia
  .media-
imdb.com/images/M/MV5BMTM1NDc4ODkxNV5BMl5BanBnXkFtZTcwNTI4ODE3MQ
@@
  ._V1_SX300.jpg"
1798709,"Her",2013,"USA","Drama, Romance, Sci-Fi","Spike Jonze",126,"http://ia
  .media-
imdb.com/images/M/MV5BMjA1Nzk0OTM2OF5BMl5BanBnXkFtZTgwNjU2NjEwMDE@
  ._V1_SX300.jpg"

0790636,"Dallas Buyers Club",2013,"USA","Biography, Drama","Jean-Marc VallÃ©e"
,117,"N/A"

Next there is a CSV file with rating/rater data, the smallest of which is called
ratings_short.csv. Again, the other similar files are quite large so we have created this small file
with just a few entries in it that you can use for testing. Each file of this type has a
header first followed by one line for each rating.
Shown below is this shorter file that has eleven lines. The first line is the header. The
second line shows the rater_id is 1, the IMDB movie ID is 0068646, the movie was rated a 10,
and the time was 1381620027.
ratings_short.csv:
rater_id,movie_id,rating,time
1,0068646,10,1381620027
1,0113277,10,1379466669
2,1798709,10,1389948338
2,0790636,7,1389963947
2,0068646,9,1382460093
3,1798709,9,1388641438
4,0068646,8,1362440416
4,1798709,6,1398043318
5,0068646,9,1364834910
5,1798709,8,1404338202

Assignment

In this assignment you will create a new class named FirstRatings to process the movie
and ratings data and to answer questions about them.

You may find it helpful to use CSVParser and CSVRecord. Note that FirstRatings will need the following three import statements:

import edu.duke.*;
import java.util.*;
import org.apache.commons.csv.*;

Specifically for this assignment you will write the following methods in a new class named FirstRatings:

Write a method named loadMovies that has one parameter, a String named filename. This method should process every record from the CSV file whose name is filename, a file of movie information, and return an ArrayList of type Movie with all of the movie data from the file.

Write a void method named testLoadMovies that should do several things.

- Call the method loadMovies on the file ratedmovies_short.csv and store the result in an ArrayList local variable . Print the number of movies, and print each movie.
You should see there are five movies in this file, which are all shown above. After this works you should comment out the printing of the movies.
If you run your program on the file ratedmoviesfull.csv, you should see there are 3143 movies in the file.

- Add code to determine how many movies include the Comedy genre. In the file ratedmovies_short.csv, there is only one.

- Add code to determine how many movies are greater than 150 minutes in length. In the file ratedmovies_short.csv, there are two.

- Add code to determine the maximum number of movies by any director, and who the directors are that directed that many movies. Remember that some movies may have more than one director.
 In the file ratedmovies_short.csv the maximum number of movies by any director is one, and there are five directors that directed one such movie.

In the FirstRatings class, write a method named loadRaters that has one parameter named filename. This method should process every record from the CSV file whose name is filename,
a file of raters and their ratings, and return an ArrayList of type Rater with all the rater data from the file.

Write a void method named testLoadRaters that should do several things.

- Call the method loadRaters on the file ratings_short.csv and store the result in a local ArrayList variable. Print the total number of raters. Then for each rater,
print the rater's ID and the number of ratings they did on one line, followed by each rating (both the movie ID and the rating given) on a separate line.
If you run your program on the file ratings_short.csv you will see there are five raters. After it looks like it works,
you may want to comment out the printing of each rater and their ratings. If you run your program on the file ratings.csv, you should get 1048 raters.

- Add code to find the number of ratings for a particular rater you specify in your code. For example, if you run this code on the rater whose rater_id is 2
for the file ratings_short.csv, you will see they have three ratings.

- Add code to find the maximum number of ratings by any rater. Determine how many raters have this maximum number of ratings and who those raters are.
If you run this code on the file ratings_short.csv, you will see rater 2 has three ratings, the maximum number of ratings of all the raters,

and that there is only one rater with three ratings.

- Add code to find the number of ratings a particular movie has. If you run this code on the file ratings_short.csv for the movie "1798709", you will see it was rated by four raters.

- Add code to determine how many different movies have been rated by all these raters. If you run this code on the file ratings_short.csv, you will see there were four movies rated.

Step Two: Simple Recommendations

Video: Average Ratings
https://www.coursera.org/lecture/java-programming-recommender/average-ratings-cUknF

Assignment 2: Second Ratings && Movie runner ave
Provided starter files can be found at:

http://www.dukelearntoprogram.com/course5/index.php

In this exercise, you will add to the programming project you did for the first assignment.
You will continue to use the provided classes Movie.java, Rating.java, and Rater.java
that were provided for the first assignment and you will also use the FirstRatings
class you wrote to read in and store information about movies and ratings of movies by
different movie raters. You will build on this assignment by calculating average ratings
of movies.
You will work with two new classes, SecondRatings which we will give you parts of, and
a new class MovieRunnerAverage, which you will create.

Assignment:

In this assignment you will modify a new class named SecondRatings, which has been
started for you, to do many of the calculations focusing on computing averages on
movie ratings.
You will also create a second new class named MovieRunnerAverage, which you will
use to test the methods you created in SecondRatings by creating a SecondRatings
object in
MovieRunnerAverage and calling its methods.

Specifically for this assignment you will write the following classes and methods:

- Note that the SecondRatings class has been started for you. This class includes two private variables, one named myMovies of type ArrayList of type Movie, and a second one named myRaters of type ArrayList of type Rater. A default constructor has also been created for you. Until you create the second constructor (see below), the class will not compile.

- Write an additional SecondRatings constructor that has two String parameters named moviefile and ratingsfile.
The constructor should create a FirstRatings object and then call the loadMovies and loadRaters methods in FirstRatings to read in all the movie and ratings data and store them in the two private ArrayList variables of the SecondRatings class, myMovies and myRaters.

- In the SecondRatings class, write a public method named getMovieSize, which returns the number of movies that were read in and stored in the ArrayList of type Movie.

- In the SecondRatings class, write a public method named getRaterSize, which returns the number of raters that were read in and stored in the ArrayList of type Rater.

- Create a new class named MovieRunnerAverage. In this class, create a void method named printAverageRatings that has no parameters. This method should:

1.- Create a SecondRatings object and use the CSV filenames of movie information and ratings information from the first assignment when calling the constructor.
2.- Print the number of movies and number of raters from the two files by calling the appropriate methods in the SecondRatings class.
Test your program to make sure it is reading in all the data from the two files.
For example, if you run your program on the files ratings_short.csv and ratedmovies_short.csv, you should see 5 raters and 5 movies.
We will add more code to this method in a bit.

- In the SecondRatings class, write a private helper method named getAverageByID
that has two parameters: a String named id representing a movie ID and an integer
named minimalRaters.

This method returns a double representing the average movie rating for this ID if there
are at least minimalRaters ratings. If there are not minimalRaters ratings, then it returns
0.0.

- In the SecondRatings class, write a public method named getAverageRatings, which
has one int parameter named minimalRaters.

This method should find the average rating for every movie that has been rated by at
least minimalRaters raters.

Store each such rating in a Rating object in which the movie ID and the average rating
are used in creating the Rating object.

The method getAverageRatings should return an ArrayList of all the Rating objects for
movies that have at least the minimal number of raters supplying a rating.

For example, if minimalRaters has the value 10, then this method returns rating
information (the movie ID and its average rating) for each movie that has at least 10
ratings.

You should consider calling the private getAverageByID method.

- In the SecondRatings class, write a method named getTitle that has one String
parameter named id, representing the ID of a movie.

This method returns the title of the movie with that ID. If the movie ID does not exist,
then this method should return a String indicating the ID was not found.

- In the MovieRunnerAverage class in the printAverageRatings method, add code to
print a list of movies and their average ratings,

for all those movies that have at least a specified number of ratings, sorted by averages.

Specifically, this method should print the list of movies, one movie per line

(print its rating first, followed by its title) in sorted order by ratings, lowest rating to highest rating.

For example, if printAverageRatings is called on the files ratings_short.csv and ratedmovies_short.csv with the argument 3, then the output will display two movies:

8.25 Her

9.0 The Godfather

- In the SecondRatings class, write a method getID that has one String parameter named title representing the title of a movie.

This method returns the movie ID of this movie. If the title is not found, return an appropriate message such as "NO SUCH TITLE."

Note that the movie title must be spelled exactly as it appears in the movie data files.

- In the MovieRunnerAverage class, write the void method getAverageRatingOneMovie, which has no parameters.

This method should first create a SecondRatings object, reading in data from the movie and ratings data files.

Then this method should print out the average ratings for a specific movie title, such as the movie "The Godfather".

If the moviefile is set to the file named ratedmovies_short.csv, and the ratingsfile is set to the file ratings_short.csv, then the average for the movie "The Godfather" would be 9.0.

Step Three: Interfaces, Filters, Database

Video: Filtering Recommendations

https://www.coursera.org/lecture/java-programming-recommender/filtering-recomendations-4PgHm

Assignment 3: Efficiency && Filter

Provided starter files can be found at:

http://www.dukelearntoprogram.com/course5/index.php

In this exercise you will continue to build on the program you wrote for the previous assignment. You will also use new classes we provide for this assignment.

In this exercise you will make your program more efficient and also use filters to be able to ask questions about movies with several traits.

You should begin by creating a new Java project and copying your Java code and the data directory from the last assignment, since you will be making several changes.

Assignment 1: Efficiency

In the first part of this assignment you will focus on making the program you have already written more efficient.

You will start with your files from the previous assignment and make a Rater interface and then make a more efficient Rater class.

Specifically for this assignment, you will do the following.

- Change the name of the class Rater.java to PlainRater.java. Be sure to compile it to make sure your newly named class works—that you've made the changes necessary for the class to

function with the name PlainRater.

- Create a new public interface named Rater. Add methods to this new interface by copying all the method signatures from the PlainRater class.

Copy just the methods—do not include the constructors or the private instance variables. The first line of the interface should be:

public interface Rater {
- Now add code to PlainRater so that it implements the Rater interface.

&#65533;&#65533;public class PlainRater implements Rater

- After making that change, try compiling your FirstRatings.java program. In order to get FirstRatings.java to compile, you will need to make only one change.
Where you have the code

rater = new Rater();

You'll need to change this so that you assign new PlainRater() to the rater object. After that change, compile FirstRatings. Try running your MovieRunnerAverage class; it should run as before.

- Create a new class named EfficientRater, and copy the PlainRater class into this class. You will make several changes to this class, including:

Change the ArrayList of type Rating private variable to a HashMap<String,Rating>. The key in the HashMap is a movie ID, and its value is a rating associated with this movie.

You will need to change addRating to instead add a new Rating to the HashMap with the value associated with the movie ID String item as the key in the HashMap.

The method hasRating should now be much shorter; it no longer needs a loop.

What other changes need to be made?

- Now change FirstRatings to use EfficientRater instead of PlainRater. You should now be able to compile FirstRatings and SecondRatings. Try running your MovieRunnerAverage class.

It should run as before, but much faster.

Additional Starter Files for Assignment 2

For this part of the assignment you will be given several new files.

The class MovieDatabase—This class is an efficient way to get information about movies. It stores movie information in a HashMap for fast lookup of movie information given a movie ID.

The class also allows filtering movies based on queries. All methods and fields in the class are static.

This means you'll be able to access methods in MovieDatabase without using new to create objects, but by calling methods like MovieDatabase.getMovie("0120915").

This class has the following parts:

- A HashMap named ourMovies that maps a movie ID String to a Movie object with all the information about that movie.

- A public initialize method with one String parameter named moviefile. You can call this method with the name of the file used to initialize the movie database.

- A private initialize method with no parameters that will load the movie file ratedmoviesfull.csv if no file has been loaded.

This method is called as a safety check with any of the other public methods to make sure there is movie data in the database.

- A private loadMovies method to build the HashMap.

- A containsID method with one String parameter named id. This method returns true if the id is a movie in the database, and false otherwise.

- Several getter methods including getYear, getTitle, getMovie, getPoster, getMinutes, getCountry, getGenres, and getDirector.

Each of these takes a movie ID as a parameter and returns information about that movie.

- A size method that returns the number of movies in the database.

- A filterBy method that has one Filter parameter named f. This method returns an ArrayList of type String of movie IDs that match the filtering criteria.

The interface Filter has only one signature for the method satisfies. Any filters that implement this interface must also have this method.

The method satisfies has one String parameter named id representing a movie ID. This method returns true if the movie satisfies the criteria in the method and returns false otherwise.

The class TrueFilter can be used to select every movie from MovieDatabase. It's satisfies method always returns true.

The class YearsAfterFilter is a filter for a specified year; it selects only those movies that were created on that year or created later than that year.

If the year is 2000, then all movies created in the year 2000 and the years after (2001, 2002, 2003, etc) would be selected if used with MovieDatabase.filterBy.

The class AllFilters combines several filters. This class has the following:

- A private variable named filters that is an ArrayList of type Filter.

- An addFilter method that has one parameter named f of type Filter. This method allows one to add a Filter to the ArrayList filters.

- A satisfies method that has one parameter named id representing a movie ID. This method returns true if the movie satisfies the criteria of all the filters in the filters ArrayList. Otherwise this method returns false.

This part of the assignment will focus on using the new class MovieDatabase, which uses a HashMap to store movie information so that looking up that information is more efficient.

This part also filters movies based on several criteria to narrow down search results. We have given you the Filter interface and sample filters TrueFilter and YearAfterFilter, in addition to the AllFilters class for using multiple Filters. You will create some new Filters as described below. For example, you may want to get only those movies with the genre of comedy. You'll also answer questions using multiple Filters, such as finding all movies that are dramas that came out in 2000 or later.

Specifically for this assignment, you will do the following.

- Create a new class named ThirdRatings. Copy your code from SecondRatings into this class. Movies will now be stored in the MovieDatabase instead of in the instance variable myMovies, so you will want to remove the private variable myMovies. The constructor will no longer have a moviefile parameter—movies will be stored in the MovieDatabase class.
- ThirdRatings has only one private variable named myRaters to store an ArrayList of Raters.
- The default constructor should look like this:

public ThirdRatings() {
    this("ratings.csv");
}

- A second constructor should have only one String parameter named ratingsfile. This constructor should call the method loadRaters from the FirstRatings class to fill the

myRaters ArrayList.

- You can remove all the methods that are movie specific, such as getMovieSize, getID, and getTitle.

- You will need to modify getAverageRatings. Note that myMovies no longer exists. Instead, you'll need to get all the movies from the MovieDatabase class and store them in an ArrayList
 of movie IDs. Thus, you will need to modify getAverageRatings to call MovieDatabase with a filter, and in this case you can use the TrueFilter to get every movie.

ArrayList<String> movies = MovieDatabase.filterBy(new TrueFilter());

Then for each movie ID in the ArrayList movies, you'll need to calculate its averageRating and return an ArrayList of Ratings for each movie that was rated by minimalRaters.

Make sure this class compiles before moving on.

- Create a new class named MovieRunnerWithFilters that you will use to find the average rating of movies using different filters.

Copy the printAverageRatings method from the MovieRunnerAverage class into this class. You will make several changes to this method:

Instead of creating a SecondRatings object, you will create a ThirdRatings object. Note that this only has one parameter, the name of a file with ratings data.

Print the number of raters after creating a ThirdsRating object.

You'll call the MovieDatabase initialize method with the moviefile to set up the movie database.

Print the number of movies in the database.

You will call getAverageRatings with a minimal number of raters to return an ArrayList of type Rating.

Print out how many movies with ratings are returned, then sort them, and print out the rating and title of each movie.

For example, if you run the printAverageRatings method on the files ratings_short.csv and ratedmovies_short.csv with a minimal rater of 1, you should see

read data for 5 raters
read data for 5 movies
found 4 movies
7.0 Dallas Buyers Club
8.25 Her
9.0 The Godfather
10.0 Heat

- You will use the YearsAfterFilter to calculate the number of movies in the database that have at least a minimal number of ratings and came out in a particular year or later.

In the ThirdRatings class, write a public helper method named getAverageRatingsByFilter that has two parameters, an int named minimalRaters for the minimum number of ratings a movie
must have and a Filter named filterCriteria. This method should create and return an ArrayList of type Rating of all the movies that have at least minimalRaters ratings and satisfies
the filter criteria. This method will need to create the ArrayList of type String of movie IDs from the MovieDatabase using the filterBy method before calculating those averages.

In the MovieRunnerWithFilters class, create a void method named printAverageRatingsByYear that should be similar to printAverageRatings, but should also create a YearAfterFilter and call getAverageRatingsByFilter to get an ArrayList of type Rating of all the movies that have a specified number of minimal ratings

and came out in a specified year or later. Print the number of movies found, and for each movie found, print its rating, its year, and its title.

For example, if you run the printAverageRatingsByYear method on the files ratings_short.csv and ratedmovies_short.csv with a minimal rater of 1 and the year 2000, you should see

read data for 5 raters
read data for 5 movies
found 2 movies
7.0 2013 Dallas Buyers Club
8.25 2013 Her

- Add a GenreFilter
Create a new class named GenreFilter that implements Filter. The constructor should have one parameter named genre representing one genre, and the satisfies method should return true if a movie has this genre. Note that movies may have several genres.

- In the MovieRunnerWithFilters class, create a void method named printAverageRatingsByGenre that should create a GenreFilter and call getAverageRatingsByFilter to get an ArrayList of type Rating of all the movies that have a specified number of minimal ratings and include a specified genre. Print the number of movies found, and for each movie, print its rating and its title on one line, and its genres on the next line. For example, if you run the printAverageRatingsByGenre method on the files ratings_short.csv

and ratedmovies_short.csv with a minimal rater of 1 and the genre "Crime", you should see

read data for 5 raters
read data for 5 movies
found 2 movies
9.0 The Godfather
   Crime, Drama
10.0 Heat
   Action, Crime, Drama

- Add a MinutesFilter
Create a new class named MinutesFilter that implements Filter. Its satisfies method should return true if a movie's running time is at least min minutes and no more than max minutes.

In the MovieRunnerWithFilters class, create a void method named printAverageRatingsByMinutes that should create a MinutesFilter and call getAverageRatingsByFilter to get an ArrayList
of type Rating of all the movies that have a specified number of minimal ratings and their running time is at least a minimum number of minutes and no more than a maximum number of
minutes. Print the number of movies found, and for each movie print its rating, its running time, and its title on one line. For example, if you run the printAverageRatingsByMinutes
method on the files ratings_short.csv and ratedmovies_short.csv with a minimal rater of 1, minimum minutes of 110, and maximum minutes of 170, then you should see

read data for 5 raters

read data for 5 movies

found 3 movies

7.0 Time: 117 Dallas Buyers Club

8.25 Time: 126 Her

10.0 Time: 170 Heat

- Add a DirectorsFilter

Create a new class named DirectorsFilter that implements Filter. The constructor should have one parameter named directors representing a list of directors separated by commas

(Example: "Charles Chaplin,Michael Mann,Spike Jonze", and its satisfies method should return true if a movie has at least one of these directors as one of its directors. Note that each movie may have several directors.

In the MovieRunnerWithFilters class, create a void method named printAverageRatingsByDirectors that should create a DirectorsFilter and call getAverageRatingsByFilter to get an

ArrayList of type Rating of all the movies that have a specified number of minimal ratings and include at least one of the directors specified. Print the number of movies found,

and for each movie print its rating and its title on one line, and all its directors on the next line.

For example, if you run the printAverageRatingsByDirectors method on the files ratings_short.csv and ratedmovies_short.csv with a minimal rater of 1 and the directors set to

"Charles Chaplin,Michael Mann,Spike Jonze", you should see:

read data for 5 raters

read data for 5 movies

found 2 movies

8.25 Her

   Spike Jonze

10.0 Heat

   Michael Mann

Note that the movie "Behind the Screen" with director "Charles Chaplin" does not appear because no one rated it.

- Now use the AllFilters class to combine asking questions about average ratings by genre and films on or after a particular year. You don't need to create a new class. In the MovieRunnerWithFilters class, create a void method named printAverageRatingsByYearAfterAndGenre that should create an AllFilters object that includes criteria based on movies

that came out in a specified year or later and have a specified genre as one of its genres. This method should call getAverageRatingsByFilter to get an ArrayList of type Rating of all

the movies that have a specified number of minimal ratings and the two criteria based on year and genre.

Print the number of movies found, and for each movie, print its rating, its year, and its title on one line, and all its genres on the next line.

For example, if you run the printAverageRatingsByYearAfterAndGenre method on the files ratings_short.csv and ratedmovies_short.csv with a minimal rater of 1, the year set to 1980,

and the genre set to "Romance", then you should see:

read data for 5 raters

read data for 5 movies

1 movie matched

8.25 2013 Her

   Drama, Romance, Sci-Fi

- Use the AllFilters class to combine asking questions about average ratings by length of film in minutes and directors.

In the MovieRunnerWithFilters class, create a void method named printAverageRatingsByDirectorsAndMinutes that should create an AllFilters object that includes criteria based on running

time (at least a specified minimum number of minutes and at most a specified maximum number of minutes), and directors (at least one of the directors in a list of specified directors—directors are separated by commas). This method should call getAverageRatingsByFilter to get an ArrayList of type Rating of all the movies that have a specified number of

minimal ratings and the two criteria based on minutes and directors. Print the number of movies found, and for each movie, print its rating, its time length, and its title on one line,

and all its directors on the next line. For example, if you run the printAverageRatingsByDirectorsAndMinutes method on the files ratings_short.csv and ratedmovies_short.csv with a

minimal rater of 1, minimum minutes set to 30, maximum minutes set to 170, and the directors set to "Spike Jonze,Michael Mann,Charles Chaplin,Francis Ford Coppola", then you should see:

read data for 5 raters
read data for 5 movies
2 movies matched
8.25 Time: 126 Her
    Spike Jonze
10.0 Time: 170 Heat
    Michael Mann

Step Four: Weighted Averages

Video: Calculating Weighted Averages
https://www.coursera.org/lecture/java-programming-recommender/calculating-weighted-averages-VaYjU

Assignment 4: Rater Database

******There are many classes and not many explanations as I have been working this project at four different stages as a requirement for this last month of the course.

For this part, we will be chiefly working the classes "RaterDatabase" and "ForthRatings".

Provided starter files can be found at:

http://www.dukelearntoprogram.com/course5/index.php

In this exercise you will continue to build on the program you wrote for the previous assignment.

You will continue to make your program more efficient with a RaterDatabase class that is designed and implemented similarly to the MovieDatabase class you used recently— you will

use both of these database classes in this assignment. You will also calculate a different kind of average movie rating,

one based on weighting ratings made by raters who are more like you, or like any given rater—valuing their ratings more than raters who don't have your tastes in movies.

To calculate these weighted averages you will need to calculate similarity scores for each rater to find out which raters you are more similar to than others,

so you can weight ratings accordingly.

Additional File for Assignment

For this assignment you will be given one class RaterDatabase, which is an efficient way to get information about raters. This class contains:

- A HashMap named ourRaters that maps a rater ID String to a Rater object that includes all the movie ratings made by this rater.
- A public static initialize method with one String parameter named filename. You can call this method with the name of the file used to initialize the rater database.
- A private initialize method with no parameters that initializes the HashMap ourRaters if it does not exist.
- A public static void addRatings method that has one String parameter named filename. You could alternatively call this method to add rater ratings to the database from a file.
- A public static void addRaterRating method that has three parameters, a String named raterID representing a rater ID, a String named movieID that represents a movie ID, and a double named rating that is the rating the rater raterID has given to the movie movieID. This function can be used to add one rater and their movie rating to the database.
Notice that the method addRatings calls this method.
- A method getRater has one String parameter named id. This method returns a Rater that has this ID.
- A method getRaters that has no parameters. This method returns an ArrayList of Raters from the database.
- A method size that has no parameters. This method returns the number of raters in the database.

Specifically for this assignment you will do the following:

- Create a new project and copy over your classes from the last exercise.

- Create a new class named FourthRatings. Copy over the following methods from the class ThirdRatings and get FourthRatings to compile. Do not copy over any of the other methods.

You should not copy, nor should you have any instance variables in FourthRatings— you'll use RaterDatabase and MovieDatabase static methods in place of instance variables—so where
you have code with myRaters, you need to replace the code with calls to methods in the RaterDatabase class.

The methods to copy into FourthRatings from ThirdRatings are below (you'll need to modify these after copying): getAverageByID, getAverageRatings, and getAverageRatingsByFilter.

- Create a new class named MovieRunnerSimilarRatings. Copy the two methods printAverageRatings and printAverageRatingsByYearAfterAndGenre from MovieRunnerWithFilters to this new class
 and modify them to work with a FourthRatings object instead of a ThirdRatings object. You can copy more of the methods into your new Runner class,
 but these two should be enough to test that FourthRatings has been set up correctly. When you run these two you should get the same output you get when those methods run with the
 ThirdRatings object.

- In the FourthRatings class, write the following methods—two are private helper methods, and one is the method that will return movie recommendations based on similarities:

Write the private helper method named dotProduct, which has two parameters, a Rater named me and a Rater named r.
This method should first translate a rating from the scale 0 to 10 to the scale -5 to 5 and return the dot product of the ratings of movies that they both rated.

This method will be called by getSimilarities.

Write the private method named getSimilarities, which has one String parameter named id—this method computes a similarity rating for each rater in the RaterDatabase (except the rater with the ID given by the parameter) to see how similar they are to the Rater whose ID is the parameter to getSimilarities.
This method returns an ArrayList of type Rating sorted by ratings from highest to lowest rating with the highest rating first and only including those raters who have a positive similarity rating since those with negative values are not similar in any way.
Note that in each Rating object the item field is a rater's ID,
and the value field is the dot product comparison between that rater and the rater whose ID is the parameter to getSimilarities.
Be sure not to use the dotProduct method with parameter id and itself!

Write the public method named getSimilarRatings, which has three parameters: a String named id representing a rater ID, an integer named numSimilarRaters,
and an integer named minimalRaters. This method should return an ArrayList of type Rating, of movies and their weighted average ratings using only the top numSimilarRaters
with positive ratings and including only those movies that have at least minimalRaters ratings from those most similar raters (not just minimalRaters ratings overall).
For example, if minimalRaters is 3 and a movie has 4 ratings but only 2 of those ratings were made by raters in the top numSimilarRaters, that movie should not be included.
These Rating objects should be returned in sorted order by weighted average rating from largest to smallest ratings. This method is very much like the getAverageRatings method
you have written previously. In particular this method should:

For every rater, get their similarity rating to the given parameter rater id. Include only those raters with positive similarity ratings—those that are more similar to rater id. Which method could you call?

For each movie, calculate a weighted average movie rating based on: Use only the top (largest) numSimilarRaters raters.

For each of these raters, multiply their similarity rating by the rating they gave that movie. This will emphasize those raters who are closer to the rater id,

since they have greater weights. The weighted average movie rating for a particular movie is the sum of these weighted average ratings

(for each rater multiply their similarity rating by their rating for the movie), divided by the total number of such ratings.

This method returns an ArrayList of Ratings for movies and their calculated weighted ratings, in sorted order.

- Write the public method getSimilarRatingsByFilter, which is similar to the getSimilarRatings method but has one additional Filter parameter named filterCriteria and uses that filter

to access and rate only those movies that match the filter criteria.

- Add the following methods to the MovieRunnerSimilarRatings class:

Write a void method printSimilarRatings that has no parameters. This method creates a new FourthRatings object, reads data into the MovieDatabase and RaterDatabase, and then calls getSimilarRatings for a particular rater ID, a number for the top number of similar raters, and a number of minimal rateSimilarRatings,

and then lists recommended movies and their similarity ratings. For example, using the files ratedmoviesfull.csv and ratings.csv and the rater ID 65, the number of minimal raters 5,

and the number of top similar raters set to 20, the movie returned with the top rated average is "The Fault in Our Stars".

Write a void method printSimilarRatingsByGenre that has no parameters. This method is similar to printSimilarRatings but also uses a genre filter and then lists recommended movies
and their similarity ratings, and for each movie prints the movie and its similarity rating on one line and its genres on a separate line below it.
For example, using the files ratedmoviesfull.csv and ratings.csv, the genre "Action", the rater ID 65, the number of minimal raters set to 5,
and the number of top similar raters set to 20, the movie returned with the top rated average is "Rush".

Write a void method printSimilarRatingsByDirector that has no parameters. This method is similar to printSimilarRatings but also uses a director filter and then lists recommended
movies and their similarity ratings, and for each movie prints the movie and its similarity rating on one line and its directors on a separate line below it.
For example, using the files ratedmoviesfull.csv and ratings.csv, the directors "Clint Eastwood,Sydney Pollack,David Cronenberg,Oliver Stone", the rater ID 1034,
the number of minimal raters set to 3, and the number of top similar raters set to 10, the movie returned with the top rated average is "Unforgiven".

Write a void method printSimilarRatingsByGenreAndMinutes that has no parameters. This method is similar to printSimilarRatings but also uses a genre filter and a minutes filter and
then lists recommended movies and their similarity ratings, and for each movie prints the movie, its minutes,
and its similarity rating on one line and its genres on a separate line below it. For example, using the files ratedmoviesfull.csv and ratings.csv, the genre "Adventure", minutes between 100 and 200 inclusive, the rater ID 65, the number of minimal raters set to 5, and the number of top similar raters set to 10,
the movie returned with the top rated average is "Interstellar".

- Write a void method printSimilarRatingsByYearAfterAndMinutes that has no parameters. This method is similar to printSimilarRatings but also uses a year-after filter and a minutes

filter and then lists recommended movies and their similarity ratings, and for each movie prints the movie, its year, its minutes, and its similarity rating on one line.

For example, using the files ratedmoviesfull.csv and ratings.csv, the year 2000, minutes between 80 and 100 inclusive, the rater ID 65, the number of minimal raters set to 5, and the number of top similar raters set to 10, the movie returned with the top rated average is "The Grand Budapest Hotel".

Assignment 5: Farewell

Video: Farewell from the Instructor Team
https://www.coursera.org/lecture/java-programming-recommender/farewell-from-the-instructor-team-VJXb6