Programming Exercise: Step Three

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.

  ```
  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 then 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.

Assignment 2: Filters

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
```