Homework 7

You may discuss any of the assignments with your classmates and tutors (or anyone else) but **all work for all assignments must be entirely your own**. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission.

You should not use any features of Java that we did not cover in class.

Amazon Prime Video

You just got hired for a job as a programmer at Amazon. They are not happy with their current software that keeps track of Amazon Prime Video inventory and your first project is to write a new one. You need to implement the following classes.

Movie

This abstract class represents a single movie offered by Amazon.

The class contains the following private data field, for which there will be a getter and setter:

- rating (double) user rating (a positive floating point number between 0.0 and 4.0)

(Guard against invalid values and throw an IllegalArgumentException with a descriptive message if any of the values are invalid.)

The class contains the following protected, final data fields for which there will be getters:

- year (int) year that the movie was released (a positive integer between 1870 and 2015)
- duration (int) duration of the movie in minutes (a positive integer)
- title (String) title of the movie

The class provides one constructor that takes four parameters (title, year, duration, rating). The constructor should throw an IllegalArgumentException with a descriptive message if any of the values are invalid.

The class should provide toString() method that returns a String object in the following format:

- TITLE: YEAR_RELEASED, rating: RATING, price: PRICE

.. where all the uppercase words are replaced by values of the corresponding data fields. Use the String.format (...) method to format the string so that everything is formatted nicely.

The Movie class should have one more method, getPrice(). This method should be abstract and return a double. (Different types of movies have different price algorithms)

Action

The Action class represents an action movie.

Action extends Movie, adds some additional data, implements the getPrice() and overrides toString().

The class contains the following private instance data fields with a getter and a setter (guard against invalid values and throw an IllegalArgumentException with a descriptive message if any of the values are invalid):

- explosions (int) The number of explosions the movie features. (a positive integer between 1 and 100)

The class provides one constructor that takes five parameters (title, year, duration, rating and explosions). The constructor should validate the arguments and throw an IllegalArgumentException with a descriptive message if any of the values are invalid. Think about how you can use 'super' here.

Action should override the toString() method that returns a String object in the following format:

- TITLE: YEAR, rating: RATING, price: PRICE, explosions: EXPLOSIONS

Use 'super' to invoke Movie's toString.

The class implements getPrice(). The price for an action movie can be calculated as follows: (year + explosions * duration) / 1000

RomCom

The RomCom class represents a romantic comedy.

RomCom extends Movie, adds some additional data, implements the getPrice() and overrides toString().

The class contains the following private data fields with getters and a setters (guard against invalid values and throw an IllegalArgumentException with a descriptive message if any of the values are invalid):

- jerks (int) The number of jerk boyfriends the movie features. (a positive integer between 1 and 10)
- friendzones (int) The number of people in the friendzone the movie features. (a positive integer between 1 and 10)

The class provides one constructor that takes six parameters (title, year, duration, rating, jerks and friendzones). The constructor should validate the arguments and throw an IllegalArgumentException with a descriptive message if any of the values are invalid. Use super.

Action should override the toString() method that returns a String object in the following format

- TITLE: YEAR_RELEASED, rating: RATING, price: PRICE, jerks: JERKS, friendzones: FRIENDZONES

Use 'super' to invoke Movie's toString.

The class implements getPrice(). The price for a romantic comedy movie can be calculated as follows: (jerks + friendzones + year - duration) / 100

Inventory

This class represents the current inventory on a Amazon's servers.

The class should be composed with an ArrayList of Movie objects. Your methods can work with the corresponding methods of the ArrayList class.

The class contains a single private data field:

- list (ArrayList<Movie>) list of all the movies

The class needs to provide the following methods:

- *add* takes a Movie object and has a void return type. If a movie matching the title and year does not exist in the inventory, then the movie is added to the ArrayList. If the movie already exists in the list, then throw a MovieInInventoryException with a descriptive message. MovieInInventoryException is given to you below.
- *remove* takes the title and year as parameters and returns a boolean. If the movie matching the title and year exist in the inventory remove it from the inventory and return true. If the movie matching the title and year does not exist in the inventory, return false.
- *contains* takes the title and year and returns a boolean. If the movie matching the title and year exist in the inventory return true, else return false.
- *size* takes no arguments and returns the number of movies in the inventory. (Hint: look at the ArrayList documentation)
- *toString* returns a multi-line String object containing the list of all the movies in the inventory (you should use the toString() method of the Movie class).

MovieInInventoryException A custom exception that indicates a film has already been added to the inventory. Defined as follows:

```
public class MovieInventoryException extends RuntimeException {
    public MovieInventoryException(String title) {
        super(title + " already in inventory.");
    }
}
```

You can add this to your code in a file called MovieInventoryException.java

TestInventory TestInventory's main method should load a file containing some movies and build objects of the right type for that movie and load them into the inventory. Moveover, the program needs to read in the file and populate the inventory with the movies from the file.

The input file contains one movie per line and the information is separated with dashes "-". The fields on each line are:

For Action movies:

- title - year - duration - rating - genre - explosions

For RomComs:

- title - year - duration - rating - genre - jerks - friendzones

The file provided is called movies_db.txt and is available in the zip that contained these instructions. A Java file for TestInventory is also provided in the zip.

Inside TestInventory *only* modify the file to complete the part in which the input file is read and inventory populated. (Again, do not modify the file other than the file parsing.)

Then make sure that your implementation of your other classes are correct by running TestInventory and reading the console output.

Grading

Does the program compile? If not, you will lose all the points for that problem.

Is the program properly documented? (worth ~20% of each problem)

Proper documentation includes:

- preamble with the name of the author, date of creation and brief description of the program;
- appropriately chosen variable names, i.e., descriptive names;
- comments inside the code describing steps needed to be taken to accomplish the goal of the program;
- appropriate formatting, indentation and use of white space to make the code readable.

Remember that the code is read by humans and it should be easy to read for people who were not involved in its development.

Is the program well developed? (worth ~40% of each problem) Make sure you create variables of appropriate types, use control statements (conditionals and loops) that are appropriate for the task, accomplish your task in a well designed and simple way (not a convoluted algorithm that happens to produce the correct output for some unknown reason). You should also design a friendly and informative user interface.

Is the program correct? (worth ~40% of each problem), Make sure that your program produces valid results that follow the specification of the problem every time it is run. At this point you can assume a "well behaved user" who enters the type of data that you request. If the program is not completely correct, you get credit proportional to how well it is developed and how close you got it to the completely correct code.