ICS 45J: Programming in Java – Winter 2017
Lab 3
Deadline: 2/17/2017

For this lab, I want you to focus on dealing with arrays and Exception handling. You will write a simple movie rental program that manages the movie inventory and rentals of those movies. This program will provide a simple command-line interface that allows users to interact with the application and perform actions such as creating a new movie in inventory, removing a movie from the inventory, renting a movie, and returning movie to the inventory. Once the program is up and running, you will also write a test suite in JUnit covering various types of cases discussed in class. (Normal, error and boundary).

There are many ways a movie rental program especially with a user interface instead of a command prompt but the main objective is to work with arrays and manage these data types manually (rather than using predefined objects that already provide most of the functionality for you. We'll talk more about ArrayLists later in the quarter so you are not to use them for this lab).

The first thing the user sees is a brief welcome message and a small menu describing commands that the user can make. For example:

```
Welcome to Class Roster Manager!
Select an action based on the following menu:
----------
am: Add Movie
dm: Discontinue Movie
rm: Rent Movie
rr: Return Rental
  p: Print Movie Inventory
  q: Quit Program
----------
Enter Command:
```

Since I am expecting us to use standard arrays and we don't have as much flexibility with size you will limit the maximum number of unique movies in inventory to 20 and the number of available to be rented movie cannot exceed 10.

The menu choices are case insensitive and are defined as follows:

- am: Add Movie
  - Adds a movie to be managed by the program.
  - The program will prompt the user for the movie code (i.e. "HPDH2"), and the movie name (i.e. "Harry Potter and the Deathly Hollows Part 2").
    - If the program already has the maximum number of movies (i.e. 20), then a short error is reported to the user and the menu is displayed again.
  - There are no restrictions on what the valid strings for a movie code or name could be with the exception for being empty
    - If either the movie code or movie name are empty a short error is reported to the user and the menu is displayed again.
  - There is no specific order to maintain for the list of movies this program will manage. o The movie codes must be unique.
    - If a duplicate is being added, then a brief error is reported and the menu is displayed again.
- dm: Discontinue Movie
  - Removes a movie currently managed by the program.
  - The program will prompt the user for the movie code. The program will then delete the movie with the movie code from memory.
    - If the move codes do not match any in the inventory (note – movie codes are case insensitive), then a brief error is reported and the menu is displayed again.
    - If the inventory list is empty, then a brief error is reported and the menu is displayed again.
    - If there are copies of the movie still out for rental, then a brief error is reported with the names of is currently renting them and the menu is displayed again.
- am: Rent Movie
  - Adds a renter to a specific movie in the system.
  - The program will prompt the user for the movie code that the renter will be renting, the renter

ID, and the first / last name of the renter.

- Similar to the Discontinue movie case, if the movie code does not match an existing movie code, then a brief error is reported and the menu is displayed again.
- There are no restrictions on what the first and last name strings can be except for empty ones. (I.e. the renter has to have at least 1 character for each first and last name)
  - If either the first or last name are empty a short error is reported to the user and the menu is displayed again.
- The renter ID must be a positive integer greater than 0 (your program can assume the renter will enter a integer, but not cannot assume it will be a positive integer).
  - If the renter's ID is negative a short error is reported to the user and the menu is displayed again.
- Renter IDs must be unique.
- If a duplicate renter ID is found, then a brief error is reported.
- Renters will be ordered lexicographically by last name and you will need to manually sort each renter when adding them as a renter of the movie.
  - In the event of a tie (i.e. two renters have the same last name), their renter ID will be used (the smaller ID appears before the larger ID).
- If the movies are all rented (i.e. it contains10 renters), then a brief error is reported and the menu is displayed again.

- rr: Return Rental
  - Returns a rented movie from a specific movie in the system.
  - The program will prompt the renter for the movie code that the renter will be returning and the renter's ID.
    - If the movie code does not match an existing movie code, then a brief error is reported and the menu is displayed again.
    - If the renter ID does not exist in the movie's list of renters, a brief error is reported and the menu is displayed again.
    - If the movie has zero renters, then a brief error is reported and the menu is displayed again.
- p: Print Inventory
  - Prints the inventory of movies, including all of the movies and renters renting each movie.
    - This includes the movie code, the movie name, the number of copies for each movie that are already rented and the list of renters including their renter ID, last name and first name.
- q: Quit Program
  - Quits the program when this option is entered.

Exceptions

Since there are several scenarios where a brief error is reported, I want you to practice Exception handling to "fine-tune" the error messages that are displayed to the user. The Exception classes you will need to create are minimal without a body. The Exceptions you will need to create and use in your program are:

- DuplicateMovieException: Thrown when a movie already exists when trying to add one.
- MovieLimitException: Thrown when a movie is trying to be added and the number of movies is at the max.
- EmptyMovieInfoException: Thrown when the movie code and/or movie name is empty when trying to add one
- MovieNotFoundException: Thrown when a movie that does not exist.
- EmptyMovieListException: Thrown when trying to return a movie when the inventory list is empty.
- EmptyRenterListException: Thrown when trying to return a movie that does not have any copies currently being rented..
- RenterLimitException: Thrown when a renter is renting to a movie without any available copies.
- RenterNotFoundException: Thrown when trying to remove a renter for a movie that doesn't exist.
- DuplicateRenterException: Thrown when trying to add a renter to a movie and a duplicate renter ID exists.
- InvalidRenterIDException: Thrown when the user enters an invalid renter ID.
- EmptyRenterNameException: Thrown when the user enters an empty first and/or last name for the renter.
- RentedMovieException: Thrown when the user attempts to discontinue a movie that has copies currently rented out.

Organization

I will provide some requirements that you will need to follow:

- Lab3.java.
  - Contains the main method. All it does is construct a MovieManager and calls .run().
- MovieManager.java
  - A class representing the highest layer of managing the class roster containing:
    - An array of Movies
    - Total number of current Movies.
  - Public Methods
    - run()
      - Runs the main loop of the program. This displays the menu, accepts user input, and handles each of the user commands.
      - Handles the custom Exceptions and displays the messages to the user.
    - addMovie(Movie m) throws MovieLimitException, EmptyMovieInfoException, DuplicateMovieException
      - Adds a movie to the array of movies
    - discontinueMovie(String movieCode) throws MovieNotFoundException,
      - EmptyMovieListException, RentedMovieException
      - Removes a movie from the array of Movies.
    - rentMovie(String movieCode, Renter s) throws RenterLimitException, DuplicateRenterException, MovieNotFoundException, EmptyRenterNameException
      - Adds a renter to an already existing movieCode.
    - returnRental (int renterId, String movieCode) throws RenterNotFoundException, EmptyRenterListException, MovieNotFoundException
      - Removes a renter from an already existing movieCode.
    - printInventory()
      - Prints the information for all movies and their renters.
- Movies.java
  - Class that represents a movie containing:
    - A String representing the movie code.
    - A String representing the movie name.
    - An int representing the count of rented copies of the movie.
    - An array of Renters renting the movie.
  - Public Methods
    - rentMovie(Renter s) throws RenterLimitException, DuplicateRenterException
      - Adds a renter to the movie. The renters array order will need to be maintained (see requirements above).
    - returnRental(int renterId) throws EmptyRenterListException, RenterNotFoundException
      - Removes a renter from the movie based on their renters ID.
    - Any setters / getters you need to use in your program.
- Renter.java
  - Class representing a renter containing:
    - An int representing the renter ID.
    - A String representing the renter's first name.
    - A String representing the renter's last name.
  - Public Methods
    - Any setters / getters you need to use in your program.
- MovieManagerUI.java
  - Class representing the I/O between the program and user.
  - There really isn't a "state" to keep when using this object, so all of its methods can be static.
  - Public Methods
    - static void printMenu()
      - Prints the menu options to the console.
    - static String getCommand()
      - Gets the command from the user. Loops until the user enters a valid command and returns the valid command.
    - Any method to help obtain information from the user including:
      - Getting the movie information when adding a new movie.
      - Getting the movie code when adding a renter to a movie or trying to remove a movie.
      - Getting the renter information when adding a renter to a movie.
      - Getting the renter ID when trying to remove a renter from a movie.

Testing

Unlike the previous labs, this one will have a specific testing component. You will write a test suite in JUnit (as discussed in lecture) going through various cases and confirming your program handles them as expected. I want you to test the MovieManager's public methods that add / delete movies and renters. Since these methods accept constructed objects, you can manually create them in your tests, call the appropriate methods, and write assert statements (or throw Exceptions) to confirm it's behaving the way you expect. You should test normal and boundary cases such as "does my movie list contain renters in the proper order when inserting them?" or "is the correct Exception thrown when I try to add a renter to a movie at max capacity?" There are many things to keep in mind, but take a stab at it and use your imagination!

Program Submission Expectations:

- Comment block for each java file provide which includes the following:
  - Name(s), IDs for the developer(s)
  - A brief description of what that file contains (A minimum of 2 sentences)
- Commenting of methods
  - A brief description of what the method does
  - For JUnits, the test case type and method being tested
- Uploading format:
  - To make it easier for downloading when we are grading, it is expected that all your files are uploaded in a zip file.
- Files to include:
  - All ".java" files to successfully run your program including your Junit test files
- Testing Report:
  - A document that includes:
    - Screens shots of the console output
    - Screenshots of your Junit results
    - Be sure to demonstrate all the exception conditions working
- Paired Programming Evaluation Form
  - Uploaded to EEE ONLY IF you did paired programming in this lab.