Mid Class Review

Goal: Now that we have learned about many trees, we want to see the forest. This exercise will have you create one entity and implement an interface using inheritance. The UI will be given to you as it is not a key component of the concepts presented so far. You should use the IDE extensively; there is no reward for doing things the hard way. This class is a pragmatic approach to the language and not an academic one.

Entity Creation

1) Create a Movie class. Create no more than five (5) key attributes. There should be a subset of attributes that would uniquely define a movie. Remember that movies have been remade in different years and dubbed in other languages. You can add other attributes and associations to other classes. The recommendation is to keep the Movie class to a minimum.
    a. Create the class in the package "`com.<yourlastname>.movierepo.entities`"
    b. Override the `toString()` method.
    c. Override the `equals()` method.
    d. How will you design your constructor?
        i. Default constructor for JSON
        ii. All parameters constructor for immutability?
        iii. A Primary-Key only constructor for minimal persistence?
    e. Hints
        i. For the country's origin/language, Java has a `Local` class
        ii. If you declare a Month attribute, Java has a Month enum class in the Java time package.
    f. Testing
        i. In the `main()` method, create two theoretical "equal" instances of your class and call the `equals()` method with the other object to confirm.
        ii. Sample test driver

```
        Movie m1 = new Movie(1);
        Movie m2 = new Movie(1);

        if (m1.equals(m2)){
            System.out.println("m1.equals(m2)");
        }
```

Implement the Comparable Interface in Movie

2) In your Movie class, implement the `Comparable` interface. Note we made a business assumption that movies with lower numbers would have been created earlier than the ones with higher numbers. (This is an arbitrary assumption just for this class exercise).

Create a MovieDatabase Interface

3) Create the MovieDatabase interface in the package: “`com.<yourlastname>.movierepo.repository`”
   a. Create the below interface
      `List<Movie> getMovies()`

Create a MovieDatabaseImpl Class

4) Create the MovieDatabaseImpl class in the package: “`com.<yourlastname>.movierepo.repository`”
   a. Implement the MovieDatabase interface and implement the below

      `List<Movie> getMovies()`

   b. In the constructor of the MovieDatabaseImpl, load the CSV file and convert a list of CSV values (see sample below) into `List<Movie>`
   c. You will need to read each line using a simple parser and convert each line into a Movie object. Then add that object into your persistent `List<Movie>` collection.
   d. The file name is “movies.csv” and located in the same path as the MovieDatabaseImpl class
   e. Sample Line
      i. `1,Back to the Future,1985,APRIL,US`
   f. In the main of this class, create an instance of your class and make sure the files have been loaded.

Create a Controller Class

5) Create the MovieController class in the package: “`com.<yourlastname>.movierepo.controller`”
    a. The class will take in its constructor a “MovieDatabase ” instance. DO NOT CREATE THE IMPL HERE.
    b. We are simulating dependency injection.
    c. Create and implement two methods
        i. public int getMovieCount()
        ii. List<Movie> getMovies()
    d. This class is very shallow. The class is the intermediate between the UI and the repository. The class should NOT hold any state.

Create a Controller Class

6) Create the ConsoleUI class in the package:
    a. “`com.<yourlastname>.movierepo.ui.console`”
7) Review the skeleton code for the UI. There is a strong need to become an expert in console UI.
8) Implement each of the menu options
    a. Sort by ID
    b. Sort by Name
    c. Sort by Year
    d. Reverse Sort by Year
    e. Search by Name
    f. Search by Year
    g. Search by Locale
9) The display movies method is completed for UI. This is not critical code.
10) For “Sort by ID” take advantage of the Comparable interface found in Movie
11) For Sort By Name
    a. Use the built-in sort in List<T>
    b. e.g. movies.sort(Comparator<T>
12) For Sort by Year use the Comparator<Movie>
    a. Create a separate class or use an inner class
13) For Rever Sort there is a
    a. Collections.reverseOrder

14) Be creative for the searches. There is a simple but brute force way to loop through the entire list. Read the list only once from the controller and pass around the list. DO NOT EXPOSE the list to the search and sort methods. Give those methods the list.