Homework 3: JSPs and MVC

For this homework, you will practice writing Java EE web apps that have a model-view-controller (MVC) architecture.

You will do this homework as a team; however, each member of your team will be responsible for the completion of a particular task.

Step 1. Make movieclub pages do something

You will continue working on the **movieclub** pages that you began in Homework 2. In particular, each team member must complete one of the tasks below. Each task describes a dynamic web page based on the forms you created in Homework 2.

You must use an MVC architecture in implementing your task (i.e., using a combination of JSPs, servlets, and POJOs).

A number of the tasks use one or both of the following provided classes (in package **edu.memphis.movieclub.model**):

(1) **UserManager**: This object stores an array of registered users. To get a handle to handle to the object, call its **getInstance** method, like this:

```
UserManager umgr = UserManager.getInstance();
```

It has an **addUser** method for adding **User** objects, and a **getUsers** method for getting an **ArrayList** of registered users.

(2) **MovieLibrary**: This object is basically the same as **UserManager**, except it stores **Movie**s, not **User**s.

Additionally, you and your teammates will need to create a number of other model classes (details in the task descriptions below). I have provided empty class skeletons for these in the **models** package.

NOTE: The details of the classes you are to create are spread throughout the task descriptions. For example, different aspects of the **User** class are spread across tasks 1, 3, 5, and 6. You will need to coordinate with your teammates to prevent issues when implementing these.

Step 2. Submit (by tagging) your team's submission

The following instructions are essentially the same as last time; only the tag name has changed.

Attention! Before performing this step, you must make sure that all team members have committed their edits to the **trunk** in the repository.

Only one team member (the leader) performs the following.

First, you must fill out the **README.txt** file in your project's **trunk**. The file should list which team member performed each task (one team member per task).

To submit work in this course, you must tag it. Then, I will checkout the revision that you tagged and grade it. By tagging, you tell me that you are done, and this is the version you want me to grade.

The tag you must use for this homework is **hw3** (case sensitive, no spaces).

To tag the current revision of your trunk as **hw3**, do as follows:

1. Go to the **SVN Repository Exploring** perspective in Eclipse.
2. In the **SVN Repositories** view, find the **trunk** folder that you want to tag.
3. Right-click on the **trunk** folder, and click **Show History**. This should open the **History** view with a table listing the past commits to the **trunk**.
4. In the History table, right-click the newest revision (i.e., the one with the greatest revision number), and click **Tag from…** This should open a **Create Tag** dialog.
5. Enter **hw3** into the **Tag** field and optionally enter a log comment, then click **OK**. This should create the tag!

To verify that tagging was successful, open the following URL in a web browser (replacing *YOUR_TEAM* with the appropriate name):

https://utopia.cs.memphis.edu/course/comp4081-2013fall/teams/*YOUR_TEAM*/movieclub/tags/

You should see an **hw3** folder, and within that folder should be the **src** folder along with the **README.txt**, **pom.xml**, and **.project** files.

The Tasks

Task 1: Register User

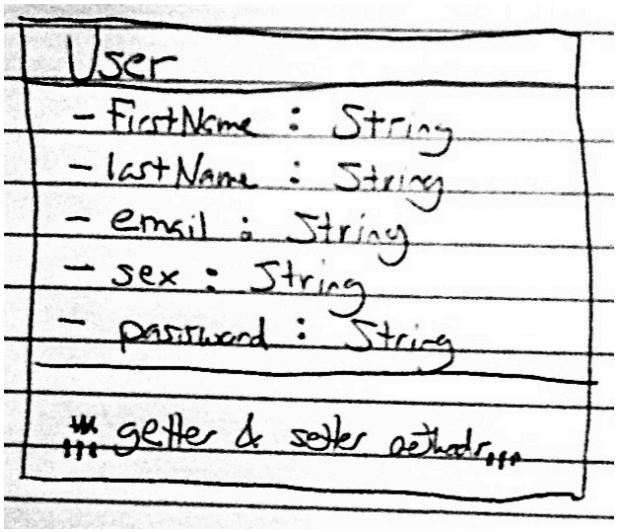
New users can be registered with the system.

Create a servlet **RegisterUserServlet** that:

- accepts a GET request and responds with the Register User form from HW2.
- accepts a POST request sent from the form, creates a **User** object based on the parameters, adds a **User** object to the **UserManager**, and responds with the form (empty).

Create a servlet **ListUsersServlet** that accepts a GET request and responds with a listing of each registered user that includes all the information entered in the Register User form.

You will need a **User** class with the following methods and attributes:



Task 2: Add Movie

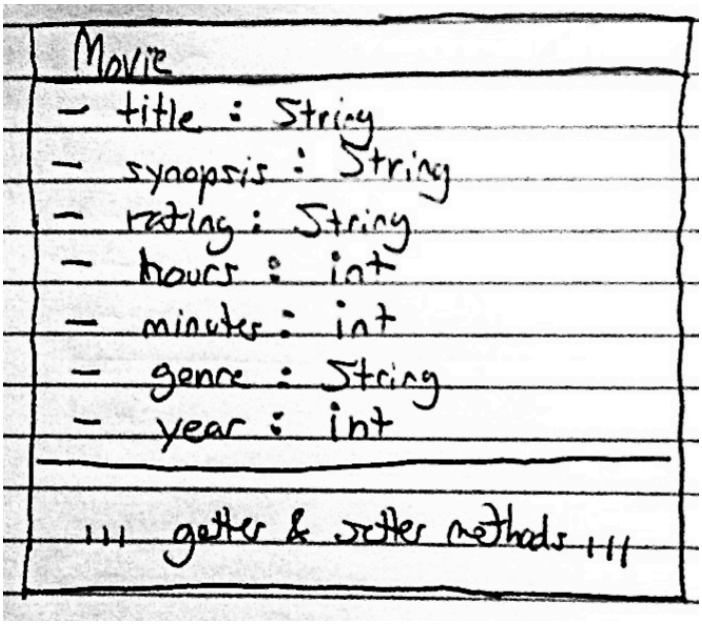
Movies can be added to the library of movies.

Create a servlet **AddMovieServlet** that:

- accepts a GET request and responds with the Add Movie form from HW2.
- accepts a POST request sent from the form, creates a **Movie** based on the parameters, adds a **Movie** object to the **MovieLibrary**, and responds with the form (empty).

Create a servlet **ListMoviesServlet** that accepts a GET request and responds with a listing of each movie in the library that includes for each movie all the info entered into the above form.

You will need a **Movie** class with the following methods and attributes:



Task 3: Top-10 List

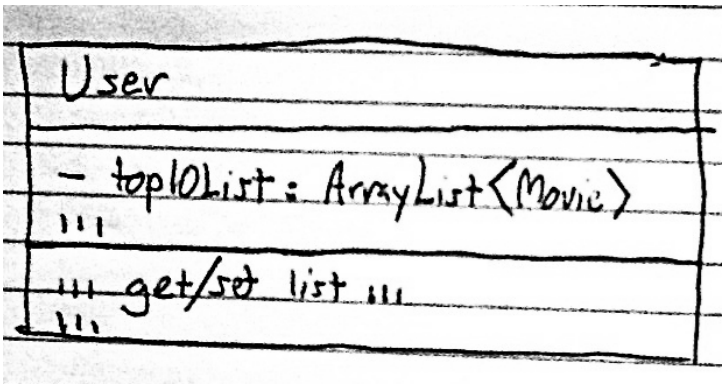
Each user has a top-10 list of their favorite movies.

Create a servlet **EnterTop10List** that:

- accepts a GET request and responds with the Top-10 List form from HW2. However, your form must be modified as follows. (1) It must populate each dropdown with the names of all movies stored in the **MovieLibrary**. (2) You must modify the original form by adding a drop-down that lists all the registered users, so the end-user can choose which user the list belongs to.
- accepts a POST request from the form, and uses the parameters to appropriately set the top-10 list of the selected user. The servlet must respond with the form (same as with GET).

Create a servlet **ShowAllTop10Lists** that accepts a GET request, and lists each registered user's top-10 list.

You will need a **User** class with the following attributes and methods (plus some of the ones from Task #1):



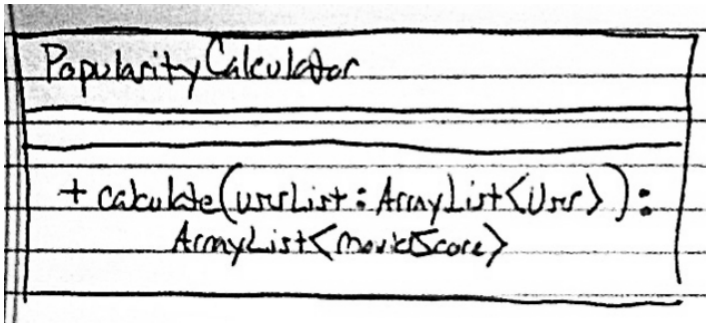
Task 4: Most Popular Movies

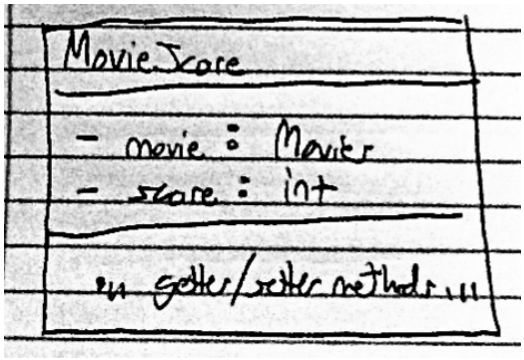
Based on users' top-10 lists, the most popular movies can be identified.

Create a servlet **ShowMostPopularServlet** that:

- accepts a GET request and responds with the Most Popular form from HW2. However, your form must be modified as follows. It must not show the list of most popular movies, just the form elements (e.g., the genre chooser).
- accepts a POST request from the form, and uses the genre parameter to go through each user's top-10 list and count up the number of times each movie of that genre appears. The servlet must respond with a form that is the same as with GET, except that it must list the movies sorted from most occurrences to least. (Don't worry about ties.)

You will need classes **PopularityCalculator** and **MovieScore** with the following attributes and methods:





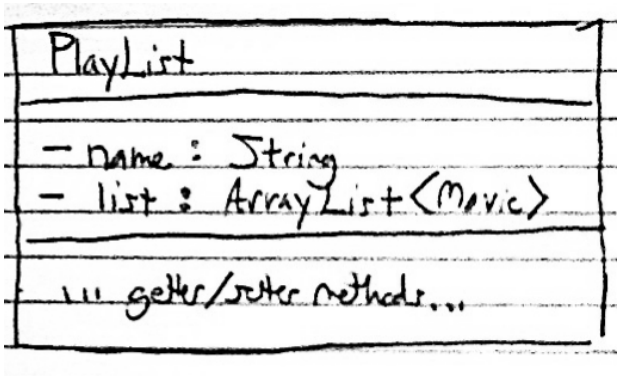
Task 5: Movie Playlist

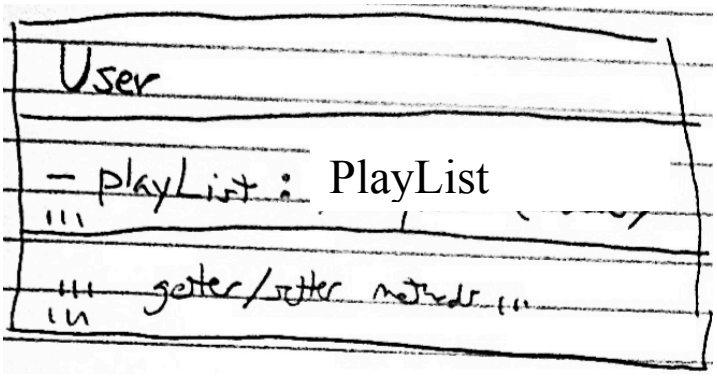
Users may create a playlist of movies.

Create a servlet **CreatePlayListServlet** that:

- accepts a GET request and responds with the Movie Playlist form from HW2. However, your form must be modified as follows. It must not show the list of movies, just the form elements. It must add a drop-down element for choosing the user (and which lists all registered users).
- accepts a POST request from the form, and uses the user and other parameters to add the movie to the user's playlist. If the submitted playlist title is non-empty, then update the title; otherwise, leave the title unchanged. The servlet must respond with the form, same as with GET, except that it must fill in the user's name and list title, and list the movies in that playlist (including the one that was just added).

You will need classes **PlayList** and **User** with the following attributes and methods (plus some of the ones from Task #1):





Task 6: Movie Review

Users can write movie reviews.

Create a servlet **PostReviewServlet** that:

- accepts a GET request and responds with the Movie Review form from HW2. However, your form must be modified as follows. It must add a drop-down element for choosing the user (and which lists all registered users).
- accepts a POST request from the form, and uses the user and other parameters to add the the review to the User's list of reviews. With a listing of all the user's reviews (including the one that was just added).

You will need classes **User** and **Review** with the following attributes and methods (plus some of the ones from Task #1):