3 Homework 3: Multi-Class Design

Due: Sunday 10/15 11:59pm

Update (10/13 2pm): Gradescope Autograder section added.

Update (10/13 3:30pm): Added clarification in Complete by-genre recommendations.

In this homework, you will design, implement, and test extensions to the existing WebFilmz code base.

The homework will be graded based on the principles of class design, the rules for Class Design Documentation, the correctness of your implementation, the quality and comprehensiveness of your testing, and the reasonableness of your inferences about issues that the homework does not explicitly specify.

Additional rules:

- Each class must be in the `cs410.webfilmz` package.
- Classes and methods must be named exactly as written.
- You cannot use this document as an excuse to avoid writing important information in the Class Design Documentation.
- You must not change the signature of any existing `public` method, unless the homework explicitly says to.
- You must not add any non-`private` methods, fields, or constructors to `Film`, `User`, `Catalog`, or `ILikeFilm` unless the homework explicitly says to. You may override the `toString` method on any of those classes. You may add `private` helper methods to any class.
- You must not modify the `BaseFilmTest`, `BaseUserTest`, and `BaseCatalogTest` classes. You should add your test cases to `UserTest` and `CatalogTest` instead.
- Your work must be on a branch named `hw3`, which should be based upon the course repository's `hw3` branch. See Making the `hw3` branch.
- On the `hw3` branch, there must be a directory named `WebFilmz`, and the contents of that directory must follow the Maven Directory Layout. It must be possible to build and test your project using Maven (`mvn compile` and `mvn test`, respectively).
- If your work is not in the correct branch, or not in the correct directory, or cannot be built with Maven, then it will not be graded and you will not receive credit for the assignment.

3.1 Fix personalized recommendations

The `BaseUserTest` has two failing test cases: `getRecommendationsNewReleases` and `getRecommendationsByDirector`. **The tests cases reflect the correct intended behavior.**

Both test cases fail because the methods in **`User`** do not remove films that the user has already watched from the set of recommendations. In general, the recommendation methods in **`User`** should not include films the user has already watched.

Fix the code in the **`User`** class so that the test cases pass.

Note: the **`getAllRecommendations`** method now takes an integer argument called **`initialGenericRecsCount`** that represents the number of recommendations to be generated from the catalog *before* any watched films are removed. The rationale is that if the user has already watched all recent films, the "New Releases" recommendations should be empty, rather than containing movies from the 1990s.

3.2 Complete by-genre recommendations

Design, implement, and test the **`public getRecommendationsByGenre`** method in **`Catalog`**. Remove the direct dependence on the **`User`** class in the same way as the **`getRecommendationsByDirector`** method.

Clarification (10/13 3:30pm): The commented-out signature in `Catalog.java` is ***wrong****; it adds the extra word "Personal" into the method name. The method name should be* ***`getRecommendationsByGenre`*** *as stated above.*

Design, implement, and test the **`public getRecommendationsByGenre`** method in the **`User`** class.

Update the **`getAllRecommendations`** method to also include entries for `"Favorite Genres"`, `"Most Watched"`, and `"Most Liked"` in its result.

3.3 Cache liked directors and genres in `User`

Change **`User`** to store the set of the user's liked directors and genres. Update **`isLikedDirector`** and **`isLikedGenre`** to use this data instead of the existing loop over all liked films. Update the class's design documentation as appropriate.

3.4 Making the `hw3` branch

Here is how you should create your `hw3` branch.

Go to your local working repo. Run `git remote show`. It will probably print one line, `origin`. Git uses the name `origin` by default for the remote repository you cloned your working copy from. Run `git remote show origin`. It will print a bunch of information, but at the top should be two URLs pointing to your gitlab repo.

Run the following commands in order:

- either `git remote add course-repo https://gitlab.com/rmculpepper/cs410-f23` or `git remote add course-repo git@gitlab.com:rmculpepper/cs410-f23` —depending on

whether you are using an SSH key to access gitlab. This adds the course repo as a separate "remote" (that is, a reference to a remote repository) named `course-repo`.

- `git fetch course-repo` — Gets any new content from the `course-repo` remote.
- `git branch hw3 course-repo/hw3`
- `git switch hw3`
- `git push --set-upstream origin hw3`

Go to your gitlab page and verify that it now contains a `hw3` branch.

3.5 Gradescope Autograder

There is now a Homework 3 assignment available on Gradescope with a limited autograder. It works in about the same way as for Homework 2.

The autograder expects a single file named `hw3.json`, which should contain a JSON object with a single key named `"gitlab_user"`, whose value must be your Gitlab user name. It must be the same Gitlab user name you used for Homework 1. For example, my `hw3.json` file would have the following contents:

```
{ "gitlab_user": "rmculpepper" }
```

You can click "Rerun Autograder" to make the autograder pull the most recent version of your code from Gitlab.