4 Homework 4: Multi-Class Design

Due: Sunday 10/22 11:59pm

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. Specifically, you are required to update the design documentation of any class, method, or field that you add or change.
- 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 `hw4`, which should be based upon the course repository's `hw4` branch as well as your `hw3` branch. See Making the `hw4` branch.
- On the `hw4` 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.

4.1 Add Film method

Add a **`public`** method to **`Film`** called **`isAppropriateFor`** that takes a **`Rating`** and indicates whether the film's rating is appropriate for an audience that can watch up to the given rating.

4.2 Limit recommendations by rating

The `BaseUserTest` class has a new failing test case called `testRatings`. (You should have fixed the previous two failing tests in homework 3.) **The test case reflects the correct intended behavior.**

The test case fails because the user is created with a maximum acceptable rating of G, but the recommendations methods produce films with higher (that is, inappropriate) ratings. In general, the recommendation methods in `User` should not include films with ratings that are inappropriate for the user.

Fix the code in the `User` class so that the test case passes. Add further tests in `UserTest` class.

4.3 Making the `hw4` branch

Two sets of instructions for creating your `hw4` branch are provided: one using `git` commands, the other using IntelliJ.

Both sets of instructions assume that your local working repo has two remotes: `origin` and `course-repo`. The `origin` remote should refer to your gitlab repository. The `course-repo` remote should refer to the course repository; that is, the instructor's gitlab repository. Run `git remote` to list the remotes, and run `git remote show origin` and `git remote show course-repo` to double-check that the remotes refer to the correct repositories.

4.3.1 Make the branch with `git` commands

To use `git` commands to create your `hw4` branch, run the following commands in order:

- `git fetch course-repo` — Gets any new content from the `course-repo` remote.
- `git branch hw4 course-repo/hw4`
- `git switch hw4`
- Pause. The `course-repo/hw4` branch is based on the `course-repo/hw3` branch, but it contains one new commit, with the commit message "WebFilmz: add Rating". Use `git show` to see a rudimentary text view of the changes in that commit; alternatively, use a GUI tool like `gitk` or the `Show Git Log` menu item in IntelliJ's `Git` menu.
- `git merge hw3` — Merges your work from `hw3`. *You must resolve any* ***merge conflicts*** *by preserving the spirit of both sets of changes. You must create a merge commit.*
- `git push --set-upstream origin hw4`

Go to your gitlab page and verify that it now contains a `hw4` branch.

4.3.2 Make the branch with IntelliJ

As an alternative, you can create and set up your `hw4` branch using IntelliJ. These instructions also assume you already have `origin` and `course-repo` remotes.

- Open the `WebFilmz` project in IntelliJ.
- Click the `Git > Fetch` menu item to get any new contents from your remotes.
- Click the `Git > Branches` menu item, then in the branches window that pops up navigate into `Remote`, then `course-repo`, then select `hw4`, then select the `New branch from 'course-repo/hw4'...` menu item. When the dialog box pops up, name the new branch `hw4` and check the `Checkout branch` box, then click the `Create` button.
- Pause. Click the `Git > Show Git Log` menu item, then click the "WebFilmz: add Rating" line in the commit list in the middle. Then double-click the `Catalog.java` file in the tree to the right to open a "Repository Diff" tab. Single-click on other files in the right tree to see their changes.
- Click the `Git > Merge...` menu item. Enter `hw3` as the branch to merge. Do not add any merge options. Click the `Merge` button.

  There will probably be at least one conflict. Click the `Merge...` button to open a conflict resolution window. On the left and right are the two lines of changes to be merged; in the middle is an editable copy of their common ancestor. View the changes; non-conflicting changes are in blue or green, and conflicts are in red.

  Click the "Apply non-conflicting changes" button to take care of the simple cases.

  Then manually resolve the conflicts. Each conflict region on the left or right has an arrow icon and an X icon. Clicking the arrow copies the conflicting code to the middle and marks that conflict resolved; clicking the X marks the conflict as resolved without changing the middle section. *Don't expect to resolve the conflict just by clicking. You must edit the middle code to be coherent.* Click the `Apply` button when you are done.
- Click the `Git > Push...` menu item and push the `hw4` branch to the `origin` remote.

Go to your gitlab page and verify that it now contains a `hw4` branch.