Assignment 1: Refactoring (10%)

Fall 2021 – CPSC 501

Due at 23:59, Oct. 9 on D2L

Assignment policy:

- This is an **individual** assignment, so the work you hand in must be your own. Any external sources used must be properly cited (see below).
- Extensions will not be granted to individual students. Requests on behalf of the entire class will only be considered if made more than 24h before the original deadline.
- Some tips to avoid plagiarism in your programming assignments (taken from previous offerings of the course):

  1. Cite all sources of code that you hand in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

     ```
     # the following code is from https://www.quackit.com/python/tutorial/python_hello_world.cfm.
     ```

     Use the complete URL so that the marker can check the source.
  2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
  3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
  4. Collaborative coding is strictly prohibited. Your assignment submission must be strictly your code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. You can not use (even with citation) another student's code.
  5. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. Note that this still applies to the current offering.
  6. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

Instructions:

Find or create a running object-oriented Java program. You must have the proper approval to use this codebase if it isn't yours. The program you use should be poorly structured and able to be improved via refactoring. A recommendation is to pick code you have written in the past, avoiding GUI-based code. If you decide to write new code for this exercise, some project suggestions are

- Accounting or inventory systems for stores or other businesses
- Employee management systems
- Command-line games, e.g. tic tac toe

The program must consist of **5-10 classes** that are coupled together to make the program function. These classes must also incorporate some sort of inheritance structure. For example, there could be an interface that is implemented by two or more subclasses, or a class that is extended by two or more child classes.

Upload this starting project as the initial codebase for a new GitLab project at `gitlab-cpsc.ucalgary.ca`. You will receive instructions on how to use this system during the first week of tutorials. Next, find aspects of the project that would benefit from refactoring and perform at least **five unique refactorings** (i.e. no two refactorings have the same name, so you can't just perform Rename Method five times). At least **two** of the refactorings must result in **substantial changes** to the internal design of the system.

Each refactoring should be tracked using **separate Git commits**, and at least one of the two larger refactorings should be completed using **branch and merge** Git commands. These commits may be done on the local repository, but must then be pushed to the remote repository. Make sure you document each refactoring with a meaningful message in the version control system.

You must also perform **JUnit testing** as you do the refactoring. You will likely want to add or modify tests as you refactor your code into smaller methods. The testing code must also be kept under version control. This unit testing doesn't have to cover the entire project – a recommendation is to choose a couple operations from the initial project, design simple unit tests for them, and then refactor the code relating to these tested operations. This way, the unit tests provide feedback about whether your refactorings are preserving functionality, without you having to write unit tests for the entire project.

Finally, you need to complete a **written report** that describes how you did your refactoring. For each of the five refactorings, your report should answer the following questions:

1. What code in which files was altered? Don't include entire files, just the code snippets that were relevant to the refactoring. Line numbers alone aren't sufficient.
2. What needed to be improved, and why? List any "bad code smells" detected.
3. Which refactoring was applied? What steps did you follow? Use the terminology and mechanics discussed in class or in the Fowler text.
4. What code in which files was the result of the refactoring? See point 1.

5. How was the code tested? Which JUnit test methods were applicable?

6. Why is the code better structured after the refactoring? This could be addressed simultaneously with point 2.

Use SHA numbers to cross-reference your commits as you describe each refactoring. Also make sure you indicate the refactoring that was required to use branch and merge.

In addition, at the beginning of the written report, you need to include **directions for the TA to access your GitLab project**. This is how they will be able to access your code, unit tests, and commit history, so double-check this works correctly before submitting.

The expected length of the report is about 2–3 pages with standard font and margins, but there are no strict requirements. It is essential, however, that your report is clear, easy to read, thorough, and written in complete sentences.

Submission instructions:

Upload your written report as a **PDF file** to the Assignment 1 dropbox on D2L by 23:59 on October 9th. The TAs will use the instructions in your report to access your GitLab project, through which they will grade your submission.

Rubric (100 pts total):

- Version control: Used Git/GitLab properly, Multiple small commits with informative messages (25 pts)
- Branch/merge: One branch and merge operation used for a larger refactoring (5 pts)
- Unit testing: Tests are applicable to refactorings and test robustly for multiple points of failure (15 pts)
- Refactorings: Evidence in version control and report of five clear and systematic refactorings. Two of these refactorings result in larger structural changes (25 pts)
- Report: Description of each of the five refactorings answering provided questions. Report is thorough and written in full sentences. (25 pts)
- Communication: Clear, working instructions on how to access GitLab project. (5 pts)

Note that for this assignment, your work will be graded by **your own** lab TA. If you have questions about the requirements or what is permitted, it is recommended you consult with them directly during tutorial or through Piazza.

---

author: jcleahy@ucalgary.ca