CPSC 501: Advanced Programming Techniques

Assignment 1: Version Control and Refactoring

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

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. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
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.

Late Penalty

Late assignments will not be accepted.

Goal

Experience and develop skills with version control, unit testing, and refactoring.

Technology

Java, Git, Gitlab, Refactoring

Specifics

Java 17, gitlab.cpsc.ucalgary.ca

Description

Find, or create, a running *object-oriented* program codebase. You must have the proper approval to use this codebase if it isn't yours. This program should be poorly structured and able to be improved via refactoring. *A recommendation is to pick code you have written in the past, avoid GUI based code (unless it is well structured and you can use unit tests on some part of it), and try to limit the complexity of the program to 5-10 classes at most.* (Larger codebases are allowed but unnecessary.)

This codebase must have a minimal level of complexity. The project must consist of ***at least 5 classes*** that are coupled together to make the program function. **The codebase might not begin with an inheritance structure, but by the end should contain an inheritance structure consisting of multiple classes**.

Upload this project as the initial codebase for a new **Gitlab** project at **gitlab.cpsc.ucalgary.ca**. Find locations within the project that would benefit from refactoring and perform at least **five unique refactorings**. At least **two of the refactorings must demonstrate substantial changes to the internal design of the system**. Each refactoring should be **tracked using separate Git commits** and **at least one of the larger two refactoring should be completed using branch and merge Git** commands. *These commits will likely be done on the local repository but must be pushed to the gitlab.cpsc.ucalgary.ca remote repository.* Make sure you document each refactoring using a meaningful message in the version control system.

Since the course has used Java examples and the JUnit testing framework, it would be best if the program was written in the Java language, but a program written in another object-oriented language is possible, provided that you can complete all requirements of the assignment (in particular, that you use a framework to do unit testing).

You must also do **unit testing as you do the refactoring**. *It is likely you will 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 complete project. The recommendation is to pick a couple simple operations from the initial codebase, design simple unit tests around them, and then refactor around these tested operations.

Complete a digital formal report that describes how you did your refactoring in the **Gitlab** project in the **readme.md** file for the project. Due to the requirements of this report it is expected that the report will take at least 2.5 pages (if hosted in Word) which is **at least a half page per refactoring**. This report should explain what you did for each refactoring, answering the following questions:

1. What code in which files was altered. (don't include the full source, only the parts relevant to the refactoring).
2. What needed to be improved? That is, what "bad code smell" was detected? Use the terminology found in the Fowler text.
3. What refactoring was applied? What steps did you follow? Use the terminology and mechanics outlined in the Fowler text.
4. What code in which files was the result of the refactoring.
5. How was the code tested?
6. Why is the code better structured after the refactoring?
7. Does the result of the refactoring suggest or enable further refactorings?
8. Use SHA numbers to cross-reference your code as you describe each refactoring. Also, note the one required refactoring that was done utilizing branch/merge.

Submit the following using the Assignment 1 Dropbox in D2L:

1. The complete source of the first version of your program.
2. The complete source of the last version of your program (including unit testing code).
3. Directions/access to the **gitlab.cpsc.ucalgary.ca** project so that your TA can access your project and read your report. TAs will need to be added as at least a 'Reporter' role to your code.

Bonus:

Implement Continuous Integration/Continuous Development through Gitlab for your code. That means you will need to implement a **gitlab-runner** that on each merge/commit will run your unit tests and create a **compiled artifact** for download of your project. The **gitlab-runner** pipeline doesn't have to successfully pass until the last commit of your project. The **gitlab-runner** should be hosted at and produce an artifact file compiled against **linux.cpsc.ucalgary.ca**. A visitor to your project should be able to **download this artifact from gitlab and run** in the same **linux.cpsc.ucalgary.ca** environment.

Grading

Version control	Used Git/Gitlab properly. Multiple small commits with informative messages.	5	
Branch/merge	One branch and merge operation around a larger refactoring.	5	
Unit Testing	Around refactoring and attempt to test robustly for more than one failure possibility.	10	
Refactorings	Evidence in version control and report of clear five refactorings (two larger required).	10	
Report	Description of each of five refactorings and fulfills required points.	15	
readme.md	Proper readme.md report formatting and full sentences.	5	
Total		**50**	
Bonus	Continuous Integration/Deployment	5	

Letter	**A+**	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**C-**	**D+**	**D**	**F**
Points	**47.5**	**45**	**42.5**	**40**	**37.5**	**35**	**32.5**	**30**	**27.5**	**25**	**22.5**	**<22.5**