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. Cited material should never be used to complete core assignment specifications. You can and should verify and code you are concerned with your instructor/TA before submission.**
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 screen, 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. Making your code available, even passively, for others to copy, or potentially copy, is also plagiarism.
6. 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/).
7. 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. **The most common penalty is an F on a plagiarized assignment.**
8. For assignments **limited use of generative AI in writing assistance is acceptable. For example, grammar suggestion, or code suggestion tools for programming**. **Programming or text that is largely generative AI produced is not allowed**. Learners are ultimately accountable for the work they submit. **Use of AI tools must be documented in an appendix for the assignment. The documentation should include what tool(s) were used, how they were used, and how the results from the AI were incorporated into the submitted work.** **Failure to cite the use of AI generated content in an assignment will be considered a breach of academic integrity and subject to Academic Misconduct procedures.**

Late Penalty

Assignment Late Policy: Students may choose to submit one or more assignments late, for no more than a total of five (5) days over the entire semester for the combined 4 assignments.

Each 24-hour period late after an assignment deadline counts as one full day regardless of how many hours the assignment was late within that period. For example, deadlines are generally on a Friday, 11:59pm local time. That means an assignment submitted any time Saturday before 11:59pm local time will be considered as 1 day late and count against the total number of late days.

Use these days at your own discretion and without explanation during the course for assignment extensions. For example, you could submit your second assignment 3 days late and your final assignment 2 days late, or just your final assignment 5 days late.

If a student still has days left to use, then their assignment will be graded without penalty. If a student has no more days left, or their submission exceeds the days they have remaining, then they will receive a 0 grade for a late assignment.

Late day usage will be tracked in a D2L grading column.

Goal

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

Technology

Java, Git, Gitlab, Refactoring

Specifics

Java 23, gitlab.cpsc.ucalgary.ca

Description

Find (or create) a functional (runs and compiles) *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 the GUI is separate from data management. We need you to be able to write unit tests for the data part you will be refactoring). Try to limit the complexity of the program to 5-15 classes at most. (Larger codebases are allowed but unnecessary as you will only end up editing a small portion of any larger codebase selected.)*

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. Coupled classes either reference each other, store each other in a list, have a class hierarchy, etc. **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 **three of the refactorings must demonstrate substantial changes to the internal design of the system**. These **three larger** refactorings must be more than just simple changes to code like renaming, adding comments, changes access from private->public, etc.

Each refactoring should be **tracked using separate Git commits** and the **refactorings should all 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 for that language to do unit testing. An example of this is pytest for Python).

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 about 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 branch name the refactorings took place under.

Submit the following using the Assignment 1 Dropbox in D2L:

1. The complete source of the first version of your program. (can download the repo as a .zip and submit that as CPSC501W25A1-src-old.zip)
2. The complete source of the last version of your program (including unit testing code and readme.md). (can download the repo as a .zip and submit that as CPSC501W25A1-src-new.zip)
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. There will be repository with instructions on how to do this shared in class material.

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 **cslinux.ucalgary.ca**. A visitor to your project should be able to **download this artifact from gitlab and run** in the same **cslinux.ucalgary.ca** environment.

The TA will be able to grade this component by seeing the Gitlab pipeline and hosted artifact in your repo and downloading it and running it. Remember to include instructions on how to run the artifact in your readme.md for the TA.

Completing the bonus will be necessary for a student to receive an A+ on this assignment as an A+ requires more than the 50 points available from the regular assignment.

Grading

Version control	Used Git/Gitlab properly. Multiple small commits with informative messages. Used a branch and merged branched into trunk.	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	**Above 50**	**47.5**	**45**	**42.5**	**40**	**37.5**	**35**	**32.5**	**30**	**27.5**	**25**	**Below 25**