Questions

0. All documentation and specifications about what the code is ***intended*** to do is provided within the code. You can assume that all comments and documentation are correct. What is incorrect is (parts of) the implementation.

1. **[10 marks]** Using Git, clone and pull the repository for your exam. Add your name and Banner ID to `Answers.txt`. git add, commit, and push.
   NOTE: To get full marks you **MUST** perform a commit **AND** push after each question. This way, if something goes wrong, each part can be reviewed separately. Feel free to commit more than once for any question. More commits is better than not enough!
   ADDITIONAL NOTE: Sometimes the Git server is slow when many students access it at the same time, particularly at the beginning and end of the exam period. To avoid any issues, you **must** perform a commit after every question.

2. **[25 marks] Debugging and Testing**.
   The unit tests all pass. However, the *component tests* in `ComponentTests.java` fail.
   (a) Debug and **FIX** the issues causing these tests to fail. There are at least five (5) bugs in the *source* code. **Assume that there are no bugs in the tests themselves.** For each bug found, include the following in `Answers.txt`:
      (i) A brief description of what caused the bug,
      (ii) The location (file, method and/or line number) of the bug, and
      (iii) How you fixed the bug.
   (b) If any bug you discover exists within a method that has unit tests, you must create a new unit test that *exposes* the bug. That is, create a new unit test that fails *before* you fix the bug. *After* you fix the bug, the unit test should pass.
   (c) All unit and component tests should now pass. Commit and push all your updates.

3. **[20 marks] Defensive programming.**
   In this question you will identify areas within the code for assertions and exceptions. Carefully review the code specifications within the code's comments and documentation.
   (a) Identify one (1) location in the code where an assertion is appropriate. In `Anwers.txt`, describe the location of the assert and what property is to be asserted. Implement the assert.
   (b) Identify three (3) different methods where throwing an **exception** is appropriate. For each exception added, include the following in `Answers.txt`:
      (i) The class and method that should throw the exception,
      (ii) What exception to throw, and
      (iii) Under what conditions should it be thrown.
   (c) Implement the exceptions of part (b) in the code. If any test fails now that you have implemented the exception, fix the test.
   (d) For each exception added in part (c), add a new unit test case that causes the exception to be thrown. The new tests should *pass* when the exception is thrown.
   (e) All unit and component tests should now pass. Commit and push all your updates.

4. **[15 marks] Data-level, Statement-level, and Routine-level Refactoring.**
   The code provided, now debugged and augmented with defensive programming, is still less than ideal in terms of code quality. Time to refactor.
   (a) Identify three (3) **different** opportunities to apply procedural refactoring. Multiple refactorings may apply to the same code region. For each refactoring identified, include in `Answers.txt`:
      (i) The location (file, method/line number) of the original code,
      (ii) The code smell that inspired the refactor,
      (iii) A brief description of the proposed fix.
   (b) Implement the refactorings identified in part (a). After implementing each refactoring, ensure that all tests pass before implementing the next refactoring. You should *not* need to modify the tests themselves to make them pass.
   (c) All unit and component tests should now pass. Commit and push all your updates.

5. **[30 marks] Class-level Refactoring.**
   To continue improving code quality, we now want to implement class-level refactoring. In particular, we are looking to apply refactorings which will improve the code quality, and the code's **extensibility** and **adaptability** to new requirements.
   (a) Identify three (3) **class-level** refactoring opportunities to be applied to the code. For each refactoring identified, include in `Answers.txt`:
      (i) The class or classes to be impacted by the refactoring,
      (ii) The location and brief description of the issue,
      (iii) Which SOLID principle (if any) is violated,
      (iv) How to fix the issue.
   (b) Implement the refactorings identified in part (a). After implementing each refactoring, ensure that all tests pass before implementing the next refactoring. Since interfaces are likely to be modified during class-level refactoring, you may need to modify the tests themselves to make them pass.
   (c) All unit and component tests should now pass. Commit and push all your updates.

End of Exam

- Ensure your name and Banner ID are included in `Answers.txt`.
- Ensure all files are added, committed, and pushed to the Git server. If your work is not pushed, it does not count. You can double check your submission is complete by going to https://git.cs.dal.ca/courses/2023-fall/csci-2134/final/????.git, where ???? is your CSID.
- All work must be committed by 11:30. The timestamp of the commit will determine this. Pushing to the Git server will be closed at 11:35. So you have some buffer time to push out your final commit. (Times are adjusted accordingly if you have accommodations.)

Good luck! Enjoy the winter break!