Assignment 1 – Test-Driven Development

This assignment is very loosely defined so that, if you do TDD properly, you can experience the benefit of test-driven development. You will come up with an idea, and an initial design for how you plan to implement that idea. Using TDD you will implement that initial design. Because you are writing tests firsts you will experience firsthand how writing tests first helps to "steer" or drive the design of your code. What you initially thought was a good way to design your classes and methods may change as you go to use those classes. You'll find better approaches or more convenient implementations that better suit how you intend to use the class, all because you are writing tests first. In the end you will have a set of code that, if you do TDD properly, will be 100% covered by tests and that you can honestly say has no bugs.

Requirements

You must write a command-line Java program. It doesn't matter what it does, but it should do something, not just be a collection of random classes and methods. Here are some examples:

- Write a small text-based game with a computer opponent.
- Write a "choose your own adventure" style interactive story with a complex decision tree
- Write a rudimentary chat bot that tries to hold a conversation with you

Those are just examples; you can think up other things to do as well. Whatever you build it must satisfy these requirements:

1. It must use only the base Java SDK (whatever the latest one is)
2. When the marker clones your individual git repository to their local machine, your program **must compile and run with the following command line commands**:
    a. cd A1
    b. javac *.java
    c. java Main
3. It must interact with the user via command-line
4. It must pretend to use a database, so that you can demonstrate mocking a database. Define an interface for your database class, and then mock that interface. It is ok for your Main class to instantiate your mock and to use the mock in place of the real database when we run your code for marking.
5. You must write 800 lines of code, spread across at least 3 different classes and 9 different methods.
6. All methods you create must be covered by tests. Tests should be easy to locate because you name your test classes and methods after the classes and methods they are testing. E.g. class User would have a class UserTest, if there is a method named login() in User, then UserTest would have loginTest(). We are looking for a 1:1 pairing of tests to methods. This is very simple to achieve if you do TDD properly and write your tests first.

7. When it first runs it should output instructions to the marker to inform the marker how to operate the program.

Marking Rubric

Your work will be graded by the following rubric:

Exceptional (A+)
Your program meets the minimum lines of code requirement. Your code is 100% covered by unit tests. Your database mock object is perfect and provides mock data for all program scenarios. Your tests are exhaustive and cover every possible range of inputs and scenarios in your program.

Very Good (A)
Your program meets the minimum lines of code requirement. Your code is 100% covered by unit tests. Your database mock object is almost perfect and provides mock data for all program scenarios, you are missing no more than 1 scenario or set of data for tests. Your tests are exhaustive and cover almost every possible range of inputs and scenarios in your program. There is at most 1 test that should be written to consider your testing exhaustive.

Good (B to A-)
Your program meets the minimum lines of code requirement. Your code is 90% covered by unit tests. Your database mock object is decent and provides mock data for most program scenarios. Your tests are decent and cover most of the possible range of inputs and scenarios in your program.

Minimal (B-)
Your code is 80% covered by unit tests. Your database mock object has some issues but is present and being used by tests. Your tests are reasonable.

Unacceptable (F)
Your code is < 80% covered by unit tests, **and/or**
your database mock object is missing or fails to properly fill the role of a DB mock, **and/or**
your code fails to meet the minimum lines of code requirement, **and/or**
your code does not compile or fails to run from the marker's command line.

IF your code does not compile, or fails to run, when the marker compiles and runs it on their machine they will contact you and inform you of this situation. You will then have 2 days to correct the problem. Once corrected your marker will attempt to compile and run your code again, if it does not compile or does not run your grade will be **F.** If it does compile and run, you may still earn grades however you will be capped at a B+. Therefore, it is in your interest to verify that your code compiles and runs. You can SSH into hector.cs.dal.ca with your CSID if you wish to test your code somewhere other than your own machine.

How to Submit

- All files you submit for this assignment must be in the private gitlab repository set up for you for this course. Go to https://git.cs.dal.ca and you will find this repository already created for you.
- The first thing you must do is clone this repository to your local machine. This is where you will do all of your assignment work for this semester. You may need to create the main branch, follow the instructions in gitlab to configure your repo. Add a readme.md file to check you have things working if necessary.
- Next, in your repository create a folder named **A1**
- **Put all of your java files directly in this A1 folder, no subfolders.**
- When you are done, commit and push your work to the **MAIN** branch. This is the code we will grade.