Assignment 3

COMP SCI 2ME3 and SFWR ENG 2AA4

March 22, 2021

1 Dates and Deadlines

Assigned: March 8, 2021

Due: March 29, 2021

Last Revised: March 22, 2021

All submissions are made through git, using your own repo located at:

`https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2021/[macid].git`

where `[macid]` should be replaced with your actual macid. The time for all deadlines is 11:59 pm.

2 Introduction

The purpose of this software design exercise is to implement a specification for software modules to support the process of CEAB (Canadian Engineering Board) accreditation. The specification is given in the file `A3P1_Spec.pdf`.

All of your code should be written in Java. At least `CourseT.java` should be documented using doxygen. Your code should follow the given specification exactly. In particular, you should not add public methods or procedures that are not specified and you should not change the number or order of parameters for methods or procedures. If you need private methods or procedures, you can add them by explicitly declaring them as private.

The intention is that your code will be run at the command line (terminal). We are not using an IDE (like Eclipse). This is to simplify our workflow, and to reinforce what is actually happening with the source code files.

Implementation and Testing

Step 1

Implement the following modules in Java: `IndicatorT.java`, `AttributeT.java`, `Measures.java`, `Services.java`, `Norm.java`, `LOsT.java`, `CourseT.java`, and `ProgramT.java`. Blank versions of the required source files have been pushed to your personal repo. All make rules ((`make expt`), (`make test`), (`make doc`) and (`make clean`)) will work "out of the box." They don't do anything, but the infrastructure is in place for you to incrementally develop your code.

Step 2

Verify that the `A3Example.java` file properly compiles with your library. You will need to uncomment the code to test it. You may want to successively remove the comments as you get further into the implementation. This will ensure that you are at least matching the expected syntax. To simplify experimenting with `A3Example.java` a makefile rule (`make expt`) is include to compile and run it.

Step 3

Write doxygen comments in the `CourseT.java` file. You can write doxygen comments in the other files, but you don't have to. Test the supplied `Makefile` rule for `doc`. This rule should compile your documentation into an html and LaTeX version. Along with the supplied `Makefile`, a doxygen configuration file is also given in your initial repo. You should not change these files.

Step 4

In the `src` folder you will have blank stubs for testing. The names of the testing files are, respectively, `TestAttributeT.java`, `TestCourseT.java`, `TestLOsT.java` and `TestProgramT.java`. Use JUnit for testing. You do not have to test the other modules, but you can add them to the testing suite if you like. For this assignment you are not required to submit a lab report, but you should still carefully think about your rationale for test case selection. Please make an effort to test normal cases, boundary cases, and exception cases.

The supplied makefile (named `Makefile`) has a rule named `test`. This rule should run all of your test cases.

Step 5

Push all of your code files to your GitLab project repo. This step should be completed by the deadline.

Notes

1. Please put your name and macid at the top of each of your source files.
2. Your program must work in the ITB labs on mills when compiled with Java, JUnit, LaTeX, doxygen and make.
3. Java specifics:
   - Please use `double` in your implementation of real.
   - Please use simple arrays to implement sequences in the input and output for the public methods, as used in the `A3Example.java` file. (For instance `IndicatorT[]`, `LOsT[]` etc.) Your types should match the code in `A3Example.java`.
   - For inside the modules (internal implementation) you are free to use any classes provided by Java (as long as they are available on mills)
   - All exceptions should be RunTimeExceptions. The names of the exceptions in the spec were chosen to match with standard Java exception. For any exception, please provide a string argument. The string should provide an explanation of the error.
   - It is not required, but you can override the `hashCode` method for LOsT, since this is often considered good practice when you are overriding the equals method.
4. Enumerated types should be implemented using `enum` in Java.
5. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**
6. **Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes. Please monitor all pushes to the course git repo.**