CSED332 Assignment 2

Due Saturday, September 28

Objectives

- Unit testing and coverage
- Maven (adding dependencies to `pom.xml`)
- Learn Java programming language

Maven and Testing

- Your code need to be compiled using only Maven in a command line for grading. Unlike the previous assignment, your `pom.xml` must be modified to include extra dependencies.
- Maven can generate coverage reports using JaCoCo plugin. We have already added this plugin in `pom.xml` (see the provided `pom.xml` and understand how the plugin is added to the build).
- To run your tests with JaCoCo and produce code coverage results, you can go to where your `pom.xml` is and execute the following commands: mvn test and mvn jacoco:report.
- The command mvn jacoco:report generates human readable reports of coverage information of your unit tests. The reports will be stored in the `target/site/jacoco` directory.
- You MUST ensure that your tests pass on your code. You will get overall 0 points if your submitted code and tests do not work with mvn test.

Problem 1

- The goal of this problem is to create a library management application with unit tests, to evaluate quality of these tests, and to automate build.
- You will create a simple model for this library application, including bools and collections (see the skeleton code for more details).
  - Books are organized in collections.
  - A book can be added or removed from a collection.
  - A collection contains a various number of books, and can also contain other sub-collections.

  For example, a collection Computer Science can contain a series of books about computer science in general and other sub-collections such as Operating System, Software Verification, etc.
- Books, collections, and libraries can be exported to or imported from a file in the JSON format (see `https://www.json.org`).
  - Do not reinvent the wheel. You may want to use a Java library for JSON, such as `org.json` (`https://github.com/stleary/JSON-java`), or any other libraries of your choice.
  - You must modify your `pom.xml` to include extra dependencies for using extra libraries. *You code may not be graded, if you just download such libraries, not using Maven.*
- You will have to write at least one (separate) JUnit test for each method of the classes `Book`, `Collection`, and `Library` in the corresponding test classes.
  - Each test method should test a single behavior with appropriate assertions. *Do not write a single method to check multiple behaviors.*
  - Your submitted tests need to achieve at least 80% **statement coverage** (instruction coverage for JaCoCo). *Do not add arbitrary code to your test method to just increase coverage.*

Problem 2

- The goal of this problem is to implement several functions for Boolean expressions. The syntax of Boolean expressions is given as follows:

$$\begin{array}{rrl} \textsf{Boolean formula} & \varphi ::= & c \mid v \mid \texttt{!}\,\varphi \mid \varphi\ \texttt{\&\&}\ \varphi \mid \varphi\ \texttt{||}\ \varphi \\ \textsf{Boolean constant} & c ::= & \texttt{true} \mid \texttt{false} \\ \textsf{Boolean variable} & v ::= & p_1 \mid p_2 \mid p_3 \mid \cdots \end{array}$$

  A Boolean expression is constructed by constants (`true` and `false`), variables of the form p_i for natural number $i > 0$, and logical operators such as `!` (negation), `&&` (and), and `||` (or).

  - We provide a parser for Boolean expressions using ANTLR4 (`https://www.antlr.org`), but you must modify your `pom.xml` to include extra dependencies for ANTLR4 (antlr4-runtime).

- A Boolean expression e *evaluates* to a truth value (either *true* or *false*), given a *truth assignment* that assign a truth value to Boolean variables in the expression e.

  - For example, the expression $(p_1\ \texttt{||}\ p_2)\ \texttt{\&\&}\ (p_2\ \texttt{||}\ \texttt{!}\ p_3)$ evaluates to *true*, given the truth assignment $\{p_1 \mapsto \textit{true},\ p_2 \mapsto \textit{false},\ p_3 \mapsto \textit{false}\}$.

- We can *simplify* a Boolean expression into an equivalent expression by logical equivalence laws as follows (see `https://en.wikipedia.org/wiki/Logical_equivalence`):

  - Identity and idempotent laws

$$\varphi\ \texttt{\&\&}\ \texttt{true} \equiv \varphi \qquad \varphi\ \texttt{||}\ \texttt{false} \equiv \varphi$$
$$\varphi\ \texttt{\&\&}\ \varphi \equiv \varphi \qquad \varphi\ \texttt{||}\ \varphi \equiv \varphi$$

  - Domination and negation laws

$$\varphi\ \texttt{\&\&}\ \texttt{false} \equiv \texttt{false} \qquad \varphi\ \texttt{||}\ \texttt{true} \equiv \texttt{true}$$
$$\varphi\ \texttt{\&\&}\ \texttt{!}\,\varphi \equiv \texttt{false} \qquad \varphi\ \texttt{||}\ \texttt{!}\,\varphi \equiv \texttt{true}$$

  - De Morgan's laws

$$\texttt{!}\,(\varphi_1\ \texttt{\&\&}\ \varphi_2) \equiv \texttt{!}\,\varphi_1\ \texttt{||}\ \texttt{!}\,\varphi_2 \qquad \texttt{!}\,(\varphi_1\ \texttt{||}\ \varphi_2) \equiv \texttt{!}\,\varphi_1\ \texttt{\&\&}\ \texttt{!}\,\varphi_2$$

  - Absorption laws

$$\varphi_1\ \texttt{||}\ (\varphi_1\ \texttt{\&\&}\ \varphi_2) \equiv \varphi_1 \qquad \varphi_1\ \texttt{\&\&}\ (\varphi_1\ \texttt{||}\ \varphi_2) \equiv \varphi_1$$

  - Double negation law

$$\texttt{!}\,(\texttt{!}\,\varphi) \equiv \varphi$$

  - Distributive laws

$$\varphi_1\ \texttt{||}\ (\varphi_2\ \texttt{\&\&}\ \varphi_3) \equiv (\varphi_1\ \texttt{||}\ \varphi_2)\ \texttt{\&\&}\ (\varphi_1\ \texttt{||}\ \varphi_3)$$
$$\varphi_1\ \texttt{\&\&}\ (\varphi_2\ \texttt{||}\ \varphi_3) \equiv (\varphi_1\ \texttt{\&\&}\ \varphi_2)\ \texttt{||}\ (\varphi_1\ \texttt{\&\&}\ \varphi_3)$$

  To simplify Boolean expressions, these rules are applied (from the left to the right) repeatedly, until no more rule can be applied, modulo the associativity and commutativity of `&&` and `||`.

  - $p_1\ \texttt{\&\&}\ (p_2\ \texttt{\&\&}\ \texttt{!}\ p_1)$ can be simplified into `false`.
  - $(p_1\ \texttt{\&\&}\ \texttt{true})\ \texttt{\&\&}\ (p_2\ \texttt{\&\&}\ \texttt{!}\,(\texttt{!}\ p_1\ \texttt{\&\&}\ \texttt{!}\ p_2))$ can be simplified into $p_1\ \texttt{\&\&}\ p_2$.

- Similarly, you will have to write at least one (separate) JUnit test for each method of the classes Constant, Variable, Negation, Conjunction, and Disjunction, in the test class ExpTest.

  - Each test method should test a single behavior with appropriate assertions. *Do not write a single method to check multiple behaviors.*
  - Your submitted tests need to achieve at least 80% **branch coverage**. *Do not add arbitrary code to your test method to just increase coverage.*

General Instruction

- Download the attached file `homework2.zip`, which contains two directores `problem1` and `problem2`. Each of them can be imported as a separate project into IntelliJ IDEA.
- Initially, mvn test will fail, because no dependencies are declared in `pom.xml`. Modify your `pom.xml` to include required libraries, including JUnit5 (org.junit.jupiter), ANTLR4 (antlr4-runtime), etc.
- The `src/main` directory contains the skeleton code. You should implement all the methods marked with *TODO*. Before writing code, read the description in the source code carefully.
- The `src/test` directory contains test classes. Use JaCoCo to find out how much coverage your tests have. Upload the JaCoCo report in CSV format from `target/site/jacoco/jacoco.csv`.
- As usual, do not modify the existing interfaces, the class names, and the signatures of the public methods. You can add more private methods or private member variables if you want.

Turning in

1. Create a private project with name `homework2` in `https://csed332.postech.ac.kr`, and clone the project on your machine.
2. Commit your changes in your `homework2` project that includes two directories `problem1` and `problem2`, including JaCoCo coverage reports, and push them to the remote repository.
3. The JaCoCo coverage report for each problem, generated by mvn jacoco:report, will be uploaded to the directory `homework2` as follows:
   - `homework2/jacoco1.csv` for Problem 1
   - `homework2/jacoco2.csv` for Problem 2
4. Tag your project with "submitted" and submit your homework. We will use the tagged version of your project for grading.

Reference

- Java Language Specification: `https://docs.oracle.com/javase/specs/`
- Beginning Java 9 Fundamentals 2nd by Kishori Sharan, Apress, 2017 (available online at the POSTECH digital library `http://library.postech.ac.kr`)
- Maven Getting Started Tutorial: `https://maven.apache.org/guides/getting-started/`