Assignment 7: White-Box Testing

Goals:

- Get familiar with white-box testing.
- Understand some subtleties of structural coverage.

To complete this individual assignment you must:

- Create a directory called "`Assignment7`" in the root directory of the personal repo we assigned to you. Hereafter, we will call this directory `<dir>`.
- Create a Java class `edu.qc.seclass.BuggyClass` in directory `<dir>/src`. (The actual path will obviously reflect the complete package structure.)

- **Task 1**: Add to the class a method called `buggyMethod1` that contains a division by zero fault such that (1) it is possible to create a test suite that achieves 100% statement coverage and does **not** reveal the fault, and (2) it is possible to create a test suite that achieves less than 50% statement coverage and reveals the fault.
    - The method can have any signature.
    - If you think it is not possible to create such a method, then
        - create an empty method;
        - add a comment in the (empty) body of the method that **concisely but convincingly** explains why creating such method is not possible.
    - Conversely, if you were able to create the method, create two JUnit test classes `edu.qc.seclass.BuggyClassTestSC1a` and `edu.qc.seclass.BuggyClassTestSC1b` for class `BuggyClass` as follows:
    - `BuggyClassTestSC1a` should achieve 100% statement coverage of `buggyMethod1` and **not** reveal the fault therein.
    - `BuggyClassTestSC1b` should achieve less than 50% statement coverage of `buggyMethod1` and reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`. (Also in this case, the actual path will obviously reflect the package structure, and the same holds for the test classes in the subsequent tasks.)

- **Task 2**: Add to the class a method called `buggyMethod2` that contains a division by zero fault such that (1) it is possible to create a test suite that achieves 100% statement coverage and does **not** reveal the fault, and (2) every test suite that achieves more than 50% branch coverage reveals the fault.

    - The method can have any signature.
    - If you think it is not possible to create such a method, then
        - create an empty method;
        - add a comment in the (empty) body of the method that **concisely but convincingly** explains why creating such method is not possible.
    - Conversely, if you were able to create the method, create two JUnit test classes `edu.qc.seclass.BuggyClassTestSC2` and `edu.qc.seclass.BuggyClassTestBC2` for class `BuggyClass` as follows:
    - `BuggyClassTestSC2` should achieve 100% statement coverage of `buggyMethod2` and **not** reveal the fault therein.
    - `BuggyClassTestBC2` should achieve more than 50% branch coverage of `buggyMethod2` and reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 3**: Add to the class a method called `buggyMethod3` that contains a division by zero fault such that (1) it is possible to create a test suite that achieves 100% branch coverage and does **not** reveal the fault, and (2) it is possible to create a test suite that achieves 100% statement coverage, does not achieve 100% branch coverage, and reveals the fault.
    - The method can have any signature.
    - If you think it is not possible to create such a method, then
        - create an empty method;
        - add a comment in the (empty) body of the method that **concisely but convincingly** explains why creating such method is not possible.
    - Conversely, if you were able to create the method, create two JUnit test classes `edu.qc.seclass.BuggyClassTestBC3` and `edu.qc.seclass.BuggyClassTestSC3` for class `BuggyClass` as follows:
    - `BuggyClassTestBC3` should achieve 100% branch coverage of `buggyMethod3` and **not** reveal the fault therein.
    - `BuggyClassTestSC3` should achieve 100% statement coverage of `buggyMethod3`, less than 100% branch coverage of `buggyMethod3`, and reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 4**: Add to the class a method called `buggyMethod4` that contains a division by zero fault such that (1) every test suite that achieves 100% statement coverage reveals the fault, and (2) it is possible to create a test suite that achieves 100% branch coverage and does **not** reveal the fault.
    - The method can have any signature.
    - If you think it is not possible to create such a method, then
        - create an empty method;

        - add a comment in the (empty) body of the method that **concisely but convincingly** explains why creating such method is not possible.
    - Conversely, if you were able to create the method, create two JUnit test classes `edu.qc.seclass.BuggyClassTestSC4` and `edu.qc.seclass.BuggyClassTestBC4` for class `BuggyClass` as follows:
    - `BuggyClassTestSC4` should achieve 100% statement coverage of `buggyMethod4` and reveal the fault therein.
    - `BuggyClassTestBC4` should achieve 100% branch coverage of `buggyMethod4` and **not** reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 5**: Add to class `BuggyClass` a method `buggyMethod5` by completing the code skeleton provided below so that (1) it is possible to create a test suite that achieves 100% statement coverage, and (2) the division by zero fault at line 4 cannot be revealed by any test suite.

```
1. public void buggyMethod5 (int i) {
2.   int x;
3.   [point where you can add code]
4.   x = i/0;
5.   [point where you can add code]
6. }
```

    - In completing the method:
        - you cannot add any extra parameter to the method;
        - you cannot use any exception handling mechanism;
        - you cannot use reflection.
        - you cannot use any code rewriting technique.
        - basically, you can only add zero or more lines of code to replace the placeholders "`[point where you can add code]`".
    - If you think it is not possible to create such a method, then
        - create an empty method;
        - add a comment in the (empty) body of the method that **concisely but convincingly** explains why creating such method is not possible.
    - Conversely, if you were able to create the method, create a JUnit test class `edu.qc.seclass.BuggyClassTestSC5` for class `BuggyClass` as follows:
    - `BuggyClassTestSC5` should achieve 100% statement coverage of `buggyMethod5` and **not** reveal the fault therein.
    - The class should be saved in directory `<dir>/test`.

- As usual, commit and push your code when done and submit the corresponding commit ID on Blackboard.

Notes (important–make sure to read carefully):

1. By "reveal the fault therein", we mean that you should let the tests that cause the division by zero fail with an uncaught exception, so that they are easy to spot.
2. **Do not use compound predicates in your code for the methods of class `BuggyClass`**. That is, only use simple predicates in the form (`<operand1> <operator> <operand2>`), such as "`if (x > 5)`" or "`while (x >= t)`". In other words, **you cannot use logical operators in your predicates (except for not)**.
3. Your Java code should compile and run out of the box with a version of Java >= 1.7.
4. Use JUnit 4 for the tests.
5. This is an **individual assignment**. You are not supposed to collaborate with your team members (or any other person) to solve it. We will enforce this by running a plagiarism detection tool on all assignments. Given the numerous different ways in which the assignment can be solved, similar solutions will be (1) easily spotted and (2) hard to justify.
6. Similarly, make sure not to post solutions on Piazza, whether complete or partial, and also to avoid questions that are too specific and may reveal information about a specific solution. You can obviously ask these types of questions privately to the instructors.