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 call this directory `<dir>`.
- Create a Java class `edu.gatech.seclass.GlitchyClass` in directory `<dir>/src`. (The actual path will obviously reflect the package structure.)

- **Task 1**: Add to the class a method called `glitchyMethod1` 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 less than 100% statement coverage and reveals the fault.
    - The method can have any signature.
    - If you think it is not possible to create a method meeting both requirements, 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, then create two JUnit test classes `edu.gatech.seclass.GlitchyClassTestSC1` and `edu.gatech.seclass.GlitchyClassTestBC1` for class `GlitchyClass` as follows:
        - `GlitchyClassTestBC1` should achieve 100% branch coverage of `glitchyMethod1` and **not** reveal the fault therein.
        - `GlitchyClassTestSC1` should achieve less than 100% statement coverage of `glitchyMethod1` 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 `glitchyMethod2` 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)

every test suite that achieves 100% statement coverage reveals the fault.
  - The method can have any signature.
  - If you think it is not possible to create a method meeting both requirements, 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, then create two JUnit test classes `edu.gatech.seclass.GlitchyClassTestBC2` and `edu.gatech.seclass.GlitchyClassTestSC2` for class `GlitchyClass` as follows:
    - `GlitchyClassTestBC2` should achieve 100% branch coverage of `glitchyMethod2` and **not** reveal the fault therein.
    - `GlitchyClassTestSC2` should achieve 100% statement coverage of `glitchyMethod2` and reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 3**: Add to the class a method called `glitchyMethod3` that contains a division by zero fault such that (1) it is possible to create a test suite that achieves less than 100% branch coverage and reveals the fault, and (2) it is **not possible** to create a test suite that achieves less than 100% statement coverage **and** reveals the fault.
  - The method can have any signature.
  - If you think it is not possible to create a method meeting both requirements, 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, then create two JUnit test classes `edu.gatech.seclass.GlitchyClassTestBC3` and `edu.gatech.seclass.GlitchyClassTestSC3` for class `GlitchyClass` as follows:
    - `GlitchyClassTestBC3` should achieve less than 100% branch coverage of `glitchyMethod3` and reveal the fault therein.
    - `GlitchyClassTestSC3` should achieve 100% statement coverage of `glitchyMethod3`, and reveal the fault therein - it should **not** be possible to create a test suite with **less than** 100% statement coverage which reveals the fault.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 4**: Add to the class a method called `glitchyMethod4` 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 100% branch coverage reveals the fault.

- The method can have any signature.
- If you think it is not possible to create a method meeting both requirements, 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, then create two JUnit test classes `edu.gatech.seclass.GlitchyClassTestSC4` and `edu.gatech.seclass.GlitchyClassTestBC4` for class `GlitchyClass` as follows:
    - `GlitchyClassTestSC4` should achieve 100% statement coverage of `glitchyMethod4` and **not** reveal the fault therein.
    - `GlitchyClassTestBC4` should achieve 100% branch coverage of `glitchyMethod4` and reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 5**: Add to class `GlitchyClass` the method `glitchyMethod5` provided here, including the final, commented part (i.e., the table):

```
public boolean glitchyMethod5 (boolean a, boolean b) {
  int x = 1;
  int y = 1;
  if(a)
    x -=1;
  else
    y +=1;
  if(b)
    x -=1;
  else
    y +=1;
  return (y/x > 0);
}
// | a | b |output|
// ================
// | T | T |      |
// | T | F |      |
// | F | T |      |
// | F | F |      |
// ================
// Coverage required: __________
```

    - Fill in the table in the comments, as follows.

    - For every possible input, write whether the output is T (true), F (false), or E (division by 0 exception)
    - In the "Coverage required" entry, write the minimal level of coverage that a test suite must achieve to guarantee that the fault in the code is revealed (i.e., a division by zero occurs), among statement, branch, and path coverage. In other words: If statement coverage guarantees that the fault is revealed, then you should write
      `Coverage required: statement coverage`
      If statement coverage does not guarantee that the fault is revealed, but branch coverage does, you should write
      `Coverage required: branch coverage`
      If neither statement nor branch coverage guarantees that the fault is revealed, but path coverage does, you should write
      `Coverage required: path coverage`
      Finally, if none of these coverage criteria considered guarantees that the fault is revealed, you should write
      `Coverage required: N/A`
    - If the answer is not "N/A", create a JUnit test class `edu.gatech.seclass.GlitchyClassTestXC5` for class `GlitchyClass` as follows:
    - `GlitchyClassTestXC5` should achieve 100% of the coverage named in your comment for `glitchyMethod5` and reveal the fault therein.
    - The class should be saved in directory `<dir>/test`.

- As usual, commit and push your code to your individual, assigned repository when done and submit the corresponding commit ID on T-Square.

Notes (important–make sure to read carefully):

1. By "reveal the fault therein", we mean that **the tests which show the division by zero fault should 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 `GlitchyClass`**. 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 (such as &&, ||) in your predicates**.
3. Your code should compile and run out of the box with a Java version >= 1.6
4. Use JUnit 4 for your JUnit 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 on Piazza any solution, 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 this type of questions privately to the instructors.