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.TestCoverageClass` in directory `<dir>/src`. (The actual path will obviously reflect the package structure.)
- Perform the tasks described below.

- **Task 1**: Add to the class a method called `testCoverageMethod1` that contains a division by zero fault such that (1) it is possible to create a test suite with less than 100% statement coverage that does find the fault, and (2) it is possible to create a test suite that achieves 100% statement coverage does **not** reveal 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.TestCoverageClassTestSC1a` and `edu.gatech.seclass.TestCoverageClassTestSC1b` for class `TestCoverageClass` as follows:
    - `TestCoverageClassTestSC1a` should achieve less than 100% statement coverage of `testCoverageMethod4` and reveal the fault therein.
    - `TestCoverageClassTestSC1b` should achieve 100% statement coverage of `testCoverageMethod1` and not reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 2**: Add to the class a method called `testCoverageMethod2` that contains a division by zero fault such that (1) **every** test suite that achieves 100% statement coverage **but** less than 100% branch coverage does **not** reveal the fault, and (2) it is possible to create a test suite that achieves

100% branch 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.TestCoverageClassTestSC2` and `edu.gatech.seclass.TestCoverageClassTestBC2` for class `TestCoverageClass` as follows:
    - `TestCoverageClassTestSC2` should achieve 100% statement coverage of `testCoverageMethod2`, less than 100% branch coverage, and **not** reveal the fault therein.
    - `TestCoverageClassTestBC2` should achieve 100% branch coverage of `testCoverageMethod2` and reveal the fault therein.
    - Both classes should be saved in directory `<dir>/test`.

- **Task 3**: Add to the class a method called `testCoverageMethod3` that contains a division by zero fault such that (1) **every** test suite that achieves 100% statement coverage does **not** reveal the fault (and it must be possible to create at least one test suite with 100% statement coverage), 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.TestCoverageClassTestSC3a` and `edu.gatech.seclass.TestCoverageClassTestSC3b` for class `TestCoverageClass` as follows:
        - `CoverageClassTestSC3A` should achieve 100% statement coverage of `coverageMethod3` and **not** reveal the fault therein.
        - `CoverageClassTestSC3B` should achieve **less than** 100% statement coverage of `coverageMethod3` and reveal the fault therein.
        - Both classes should be saved in directory `<dir>/test`.

- **Task 4**: Add to the class a method called `testCoverageMethod4` 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.TestCoverageClassTestBC4` and `edu.gatech.seclass.TestCoverageClassTestSC4` for class `TestCoverageClass` as follows:
        - `TestCoverageClassTestBC4` should achieve 100% branch coverage of `testCoverageMethod4` and **not** reveal the fault therein.
        - `TestCoverageClassTestSC4` should achieve 100% statement coverage of `testCoverageMethod4` and reveal the fault therein.
        - Both classes should be saved in directory `<dir>/test`. (The full actual path will obviously also reflect the package structure, and the same holds for the test classes in the subsequent tasks.)

- **Task 5**: Add to class `TestCoverageClass` the method `testCoverageMethod5` provided here, including the final, commented part (i.e., the tables):

```java
public boolean testCoverageMethod5 (boolean a, boolean b) {
   int x = 2;
   int y = 4;
   if(a)
       x += 2;
   else
       y = y/x;
   if(b)
       y -= 4;
   else
       y -= 2;
   return ((x/y)>= 0);
}


// ================
//
// Fill in column “output” with T, F, or E:
//
```

```
// | a | b |output|
// ================
// | T | T |      |
// | T | F |      |
// | F | T |      |
// | F | F |      |
// ================
//
// Fill in the blanks in the following sentences with
// "NEVER", "SOMETIMES" or "ALWAYS":
//
// Test suites with 100% statement coverage _____ reveal the
fault in this method.
// Test suites with 100% branch coverage ______ reveal the
fault in this method.
// Test suites with 100% path coverage ______ reveal the fault
in this method.
// ================
```

- Fill in the table in the comments, as follows:
  - For every possible input, fill in the output column indicating whether the output is T (true), F (false), or E (division by 0 exception)
  - In the sentences following the table, fill in the three blanks with either "NEVER", "SOMETIMES", or "ALWAYS" to indicate whether a test suite with 100% coverage for the specified criterion NEVER reveals the fault, SOMETIMES reveals the fault, or ALWAYS reveals the fault in the provided `testCoverageMethod5`.

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

Notes (important–make sure to read carefully):

- By "reveal the fault therein", we mean that the tests which show the integer division by zero fault should FAIL with an uncaught ArithmeticException, so that they are easy to spot.
- Do not use compound predicates in your code for the methods of class **`TestCoverageClass`**. 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, or nested if statements**.
- **Do not use dead code or unreachable code** in your code for the methods of class `TestCoverageClass`. It must be possible to make at least one test suite with the coverage required for each class.
- Your code should compile and run out of the box with Java 11 or 12.
- Read the requirements carefully. For example, "**Every** test suite..." refers to **all possible test suites** for your method, not only the example test suite you write.
- Use JUnit 4 for your JUnit tests.

- 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.
- 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.