**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.CoverageClass 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 coverageMethod1 that contains a [division by zero fault](https://docs.oracle.com/javase/7/docs/api/java/lang/ArithmeticException.html) 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.CoverageClassTestSC1 and edu.gatech.seclass.CoverageClassTestBC1 for class CoverageClass as follows:
    - CoverageClassTestSC1 should achieve 100% statement coverage of coverageMethod1, less than 100% branch coverage, and **not** reveal the fault therein.
    - CoverageClassTestBC1 should achieve 100% branch coverage of coverageMethod1 and reveal the fault therein.
    - Both classes should be saved in directory <dir>/test.
* **Task 2**: Add to the class a method called coverageMethod2 that contains a [division by zero fault](https://docs.oracle.com/javase/7/docs/api/java/lang/ArithmeticException.html) such that (1) it is possible to create a test suite that achieves less than 100% statement coverage and reveals the fault, and (2) it is possible to create a test suite that achieves 100% path coverage and 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.CoverageClassTestSC2 and edu.gatech.seclass.CoverageClassTestPC2 for class CoverageClass as follows:
    - CoverageClassTestSC2 should achieve **less than** 100% statement coverage of coverageMethod2 and reveal the fault therein.
    - CoverageClassTestPC2 should achieve 100% path coverage of coverageMethod2 and **not** reveal the fault therein.
    - Both classes should be saved in directory <dir>/test.
* **Task 3**: Add to the class a method called coverageMethod3 that contains a division by zero fault such that (1) it is possible to create a test suite that achieves **less than** 100% statement coverage and reveals the fault, and (2) **every** test suite that achieves 100% statement coverage does **not** reveal the fault (“every test suite” must include at least one test suite - do not use dead code).
  + 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.CoverageClassTestSC3A and edu.gatech.seclass.CoverageClassTestSC3B for class CoverageClass as follows:
    - CoverageClassTestSC3A should achieve **less** than 100% statement coverage of coverageMethod3 and reveal the fault therein.
    - CoverageClassTestSC3B should achieve 100% statement coverage of coverageMethod3 and **not** reveal the fault therein.
    - Both classes should be saved in directory <dir>/test.
* **Task 4**: Add to the class a method called coverageMethod4 that contains a [division by zero fault](https://docs.oracle.com/javase/7/docs/api/java/lang/ArithmeticException.html) 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.CoverageClassTestBC4 and edu.gatech.seclass.CoverageClassTestSC4 for class CoverageClass as follows:
    - CoverageClassTestBC4 should achieve 100% branch coverage of coverageMethod4 and **not** reveal the fault therein.
    - CoverageClassTestSC4 should achieve 100% statement coverage of coverageMethod4 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 CoverageClass the method coverageMethod5 provided here, including the final, commented part (i.e., the tables):

**public boolean** coverageMethod5 (**boolean** a, **boolean** b) {

**int** x = 1;

**int** y = 1;

**if**(a)

y -= x ;

**else**

x += y;

**if**(b)

y = x/y;

**else**

y = x\*y;

**return** (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 coverageMethod5.
* 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):**

1. By “reveal the fault therein”, we mean that **the tests which show the integer division by zero fault should FAIL with an uncaught** [**ArithmeticException**](https://docs.oracle.com/javase/7/docs/api/java/lang/ArithmeticException.html), so that they are easy to spot. Do not catch the exception in your tests or methods.
2. **Do not use compound predicates in your code for the methods of class CoverageClass**. That is, only use simple predicates in the form (<operand1> <operator> <operand2>), such as “if (x > 5)”. In other words, **you cannot use logical operators (such as &&, ||) in your predicates, or nested if statements**. This does not require complex code.
3. **Do not use dead or unreachable code** in your code for the methods of class CoverageClass. It must be possible to make a test suite with the coverage required for each class.
4. Your code should compile and run out of the box with a Java version 1.8.
5. Use JUnit 4 for your JUnit tests.
6. **Read the requirements carefully.** For example, “**Every** test suite” with a given property refers to **all possible test suites** for your method that satisfy that property, not only the examples you give.
7. You must choose whether each task is possible or not possible. You cannot guess both for one task.
8. 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.
9. 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.