Assignment #3

Turn in your work as **a single Word or PDF file**.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Problem 1** | | | **Problem 2** | | | **Problem 3** | **Total**  **points** |
| **1.1** | **1.2** | **1.3** | **2.1** | **2.2** | **2.3** |
| 36 pts | 9 pts | 5 pts | 15 pts | 15 pts | 10 pts | 10 | 100 pts |

The given source code is structured as follows. Do not make any change to the code. If you choose to use a different programming language, it is your responsibility to translate all code correctly.

Graphical user interface

Description automatically generated

1. **Fault Detection Conditions**

This problem is based on the given triangle package. It consists of 10 folders: correct, mutant1, mutant 2, …, and mutant 9. TriangleSides.java in the “correct” folder is the original correct version. The reportTriangleType method in TriangleSides.java reports the type (EQUILATERAL, SCALENE, ISOSCELES and NOTRIANGLE) of a triangle, given the lengths of three integer sides a, b, and c. TriangleSides.java in each mutantX (X=1, …, 9) folder is a variant (mutant) of the original version, where the reportTriangleType method is modified. The modifications are summarized below.

|  |  |  |
| --- | --- | --- |
| *mutantX* | *Original* | *Mutated version* |
| 1 | if (a<b+c && b<a+c && c<b+a){ | if (b<a+c && c<b+a){ |
| 2 | if (a==b && b==c) | if (b==c) |
| 3 | if (a==b || b==c || c==a) | if (a==b || c==a) |
| 4 | return TriangleType. SCALENE; | return TriangleType.*ISOSCELES*; |
| 5 | if (a<b+c && b<a+c && c<b+a){ }  else return TriangleType.NOTRIANGLE; | if-else removed |
| 6 | if (a==b || b==c || c==a) | if (a==b || b<=c || c==a) |
| 7 | if (a<b+c && b<a+c && c<b+a){ | if (a<b+c && b<=a+c && c<b+a){ |
| 8 | if (a<=0 || b<=0 || c<=0) | if (a<=0 && b<=0 && c<=0) |
| 9 | if (a<=0 || b<=0 || c<=0)  return TriangleType.NOTRIANGLE; | // if (a<=0 || b<=0 || c<=0)  // return TriangleType.NOTRIANGLE; |

* 1. Compare each mutant with the correct version of TriangleSides.java and formulate the fault detection condition, including all reachability, necessity and propagation constraints. If the fault detection condition of a mutant is satisfiable, then create a test case that satisfies the condition. Note that one test case may satisfy the fault detection conditions of multiple variants. Try your best to find such cases although this is not required. This will help reduce the number of test cases.

Provide your fault detection conditions and test cases in the following table **(36 points).**

|  |  |  |
| --- | --- | --- |
| *mutantX* | *Fault detection condition* | *Test case (input & oracle value) if it is a non-equivalent mutant* |
| 1 | A test case where **a** is greater than or equal to **b + c**, but **b < a + c** and **c < b + a** are true. | **a = 3, b = 1, c = 1** (Original: NOTRIANGLE, Mutant: SCALENE) |
| 2 | A test case where **b** equals **c**, but **a** is not equal to **b**. | **a = 3, b = 2, c = 2** (Original: ISOSCELES, Mutant: EQUILATERAL) |
| 3 | A test case where **b** equals **c**, but **a** does not equal **b** or **c**. | **a = 3, b = 2, c = 2** (Original: ISOSCELES, Mutant: SCALENE) |
| 4 | A test case that is classified as SCALENE. | **a = 3, b = 4, c = 5** (Original: SCALENE, Mutant: ISOSCELES) |
| 5 | A test case where a triangle is not formed (**a >= b + c** or similar). | **a = 5, b = 1, c = 1** (Original: NOTRIANGLE, Mutant: ISOSCELES) |
| 6 | A test case where **b <= c**, but it's not an ISOSCELES. | **a = 4, b = 3, c = 5** (Original: SCALENE, Mutant: ISOSCELES) |
| 7 | A test case where **b** equals **a + c**, but a triangle is still formed. | **a = 2, b = 3, c = 1** (Original: NOTRIANGLE, Mutant: SCALENE) |
| 8 | A test case where one side is non-positive, but not all. | **a = 0, b = 2, c = 3** (Original: NOTRIANGLE, Mutant: NOTRIANGLE) |
| 9 | Any test case where one or more sides are non-positive. | **a = -1, b = 2, c = 2** (Original: NOTRIANGLE, Mutant: NOTRIANGLE) |

* 1. Write a Junit class TriangleMutationTest.java in the correct folder to implement all of your test cases in Problem 1.1.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* 1. Copy your TriangleMutationTest.java in Problem 1.2 to each mutantX (X=1, 2, …, 9) folder. Make sure the package statement in the copied TriangleMutationTest.java is correct so that you can test TriangleSides.java in the same mutantX folder. Summarize the test execution results (pass/fail) in the following table and provide the screenshot of the test execution result for each mutantX (X=1, 2, …, 9).

mutant1

A computer screen with colorful text

Description automatically generated

mutant2

A computer screen shot of a black screen

Description automatically generated

mutant3

A screen shot of a computer

Description automatically generated

mutant4

A computer screen with text on it

Description automatically generated

mutant5

A computer screen with colorful text

Description automatically generated

mutant6

A computer screen with text on it

Description automatically generated

mutant7

A computer screen with text on it

Description automatically generated

mutant8

A screen shot of a computer code

Description automatically generated

mutant9

A screen shot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| *mutantX* | *Pass or Fail?* |
| 1 | Fail |
| 2 | Fail |
| 3 | Fail |
| 4 | Fail |
| 5 | Fail |
| 6 | Fail |
| 7 | Fail |
| 8 | Pass |
| 9 | Pass |

1. **Code Coverage**

Given TaxReturn.java, design a minimum set of test cases that fully achieves the decision coverage of the getTax method. The decisions are as follows:

|  |
| --- |
| status == SINGLE |
| income <= SINGLE\_CUTOFF1 |
| income<= SINGLE\_CUTOFF2 |
| income <= MARRIED\_CUTOFF1 |
| income<= MARRIED\_CUTOFF2 |

* 1. List the test cases (inputs only; oracle values are not needed) in Table 2.1.

Table 2.1 Test cases (test inputs)

|  |  |  |
| --- | --- | --- |
| **Test case no** | **Test input** | |
| **status** | **income** |
| 1 | SINGLE | 20000 (less than SINGLE\_CUTOFF1) |
| 2 | SINGLE | 30000 (between SINGLE\_CUTOFF1 and SINGLE\_CUTOFF2) |
| 3 | SINGLE | 60000 (more than SINGLE\_CUTOFF2) |
| 4 | MARRIED | 30000 (less than MARRIED\_CUTOFF1) |
| 5 | MARRIED | 40000 (between MARRIED\_CUTOFF1 and MARRIED\_CUTOFF2) |
| 6 | MARRIED | 90000 (more than MARRIED\_CUTOFF2) |

* 1. Complete Table 2.2 to demonstrate that the test cases in Table 2.1 have achieved the decision coverage of every decision. Each entry in Table 2.2 is the list of test case numbers in Table 2.1 that make the corresponding decision true or false.

Table 2.2 Decision coverage

|  |  |  |
| --- | --- | --- |
| **Decision** | **Test Case ID (s)** | |
| **True** | **False** |
| status == SINGLE | 1,2,3 | 4,5,6 |
| income <= SINGLE\_CUTOFF1 | 1 | 2,3 |
| income<= SINGLE\_CUTOFF2 | 1,2 | 3 |
| income <= MARRIED\_CUTOFF1 | 4 | 5,6 |
| income<= MARRIED\_CUTOFF2 | 4,5 | 6 |

* 1. Write a Junit class TaxReturnTest.java in the same folder as TaxReturn.java. A screenshot of a computer program

     Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **Modified Condition/Decision Coverage**  **(10 points)**

Create a set of four tests that meets the MC/DC criterion of (C1 and C2) or C3 where C1, C2, and C3 are Boolean variables.

1. **Test Case 1**: **C1 = true, C2 = true, C3 = false**
   * Outcome: **(true and true) or false** = **true**
   * Justification: Shows the impact of C1 and C2 being true.
2. **Test Case 2**: **C1 = false, C2 = true, C3 = false**
   * Outcome: **(false and true) or false** = **false**
   * Justification: Varies C1 while keeping C2 and C3 constant, showing the independent effect of C1.
3. **Test Case 3**: **C1 = true, C2 = false, C3 = false**
   * Outcome: **(true and false) or false** = **false**
   * Justification: Varies C2 while keeping C1 and C3 constant, showing the independent effect of C2.
4. **Test Case 4**: **C1 = false, C2 = false, C3 = true**
   * Outcome: **(false and false) or true** = **true**
   * Justification: Demonstrates the effect of C3, independent of C1 and C2.