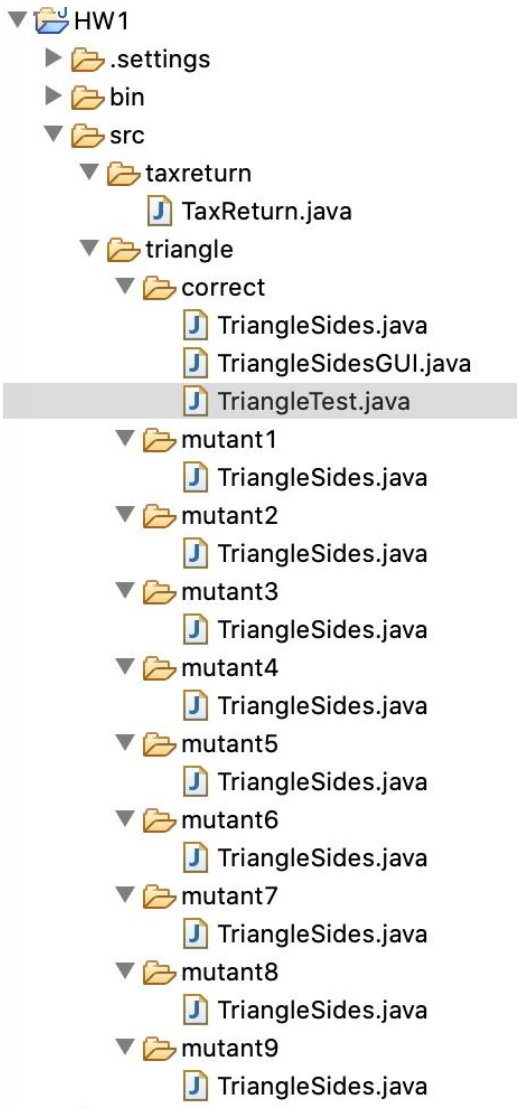
Assignment #3

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.



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;

1.1 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)

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

```java
package triangle.correct;
import org.junit.Test;

import static org.junit.Assert.*;

public class TriangleMutationTest {
    @Test
    public void testMutant1() {
        assertEquals("Correct should report NOTRIANGLE for sides 3, 1, 1",
                TriangleSides.TriangleType.NOTRIANGLE,
                new TriangleSides(a:3, b:1, c:1).reportTriangleType());
    }

    // Test for Mutant 2
    @Test
    public void testMutant2() {
        assertEquals("Correct should report ISOSCELES for sides 3, 2, 2",
                TriangleSides.TriangleType.ISOSCELES,
                new TriangleSides(a:3, b:2, c:2).reportTriangleType());
    }

    // Test for Mutant 3
    @Test
    public void testMutant3() {
        assertEquals("Correct should report ISOSCELES for sides 3, 2, 2",
                TriangleSides.TriangleType.ISOSCELES,
                new TriangleSides(a:3, b:2, c:2).reportTriangleType());
    }
```

```
29
30      // Test for Mutant 4
31      @Test
32      public void testMutant4() {
33          assertEquals("Correct should report SCALENE for sides 3, 4, 5",
34                  TriangleSides.TriangleType.SCALENE,
35                  new TriangleSides(a:3, b:4, c:5).reportTriangleType());
36      }
37
38      // Test for Mutant 5
39      @Test
40      public void testMutant5() {
41          assertEquals("Correct should report NOTRIANGLE for sides 5, 1, 1",
42                  TriangleSides.TriangleType.NOTRIANGLE,
43                  new TriangleSides(a:5, b:1, c:1).reportTriangleType());
44      }
45
46      // Test for Mutant 6
47      @Test
48      public void testMutant6() {
49          assertEquals("Correct should report SCALENE for sides 4, 3, 5",
50                  TriangleSides.TriangleType.SCALENE,
51                  new TriangleSides(a:4, b:3, c:5).reportTriangleType());
52      }
53
54      // Test for Mutant 7
55      @Test
56      public void testMutant7() {
57          assertEquals("Correct should report NOTRIANGLE for sides 2, 3, 1",
58                  TriangleSides.TriangleType.NOTRIANGLE,
59                  new TriangleSides(a:2, b:3, c:1).reportTriangleType());
60      }
61
62      // Test for Mutant 8
63      @Test
64      public void testMutant8() {
65          assertEquals("Correct should report NOTRIANGLE for sides 0, 2, 3",
66                  TriangleSides.TriangleType.NOTRIANGLE,
67                  new TriangleSides(a:0, b:2, c:3).reportTriangleType());
68      }
69
70      // Test for Mutant 9
71      @Test
72      public void testMutant9() {
73          assertEquals("Correct should report NOTRIANGLE for sides -1, 2, 2",
74                  TriangleSides.TriangleType.NOTRIANGLE,
75                  new TriangleSides(-1, b:2, c:2).reportTriangleType());
76      }
77  }
```

1.3 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

```
1  package triangle.mutant1;
2  import org.junit.Test;
3
4  import static org.junit.Assert.*;
5
6  public class TriangleMutationTest {
7      // Test for Mutant 2
8      @Test
9      public void testMutant1() {
10         assertEquals("Mutant 1 should report NOTRIANGLE for sides 3, 1, 1",    Expected [NOTRIANGLE] but was [ISOSCELES]
11                 TriangleSides.TriangleType.NOTRIANGLE,
12                 new TriangleSides(a:3, b:1, c:1).reportTriangleType());
13     }
14 }
```

mutant2

```
1  package triangle.mutant2;
2  import org.junit.Test;
3
4  import static org.junit.Assert.*;
5
6  public class TriangleMutationTest {
7      // Test for Mutant 2
8      @Test
9      public void testMutant2() {
10         assertEquals("Mutant 2 should report ISOSCELES for sides 3, 2, 2",    Expected [ISOSCELES] but was [EQUILATERAL]
11                 TriangleSides.TriangleType.ISOSCELES,
12                 new TriangleSides(a:3, b:2, c:2).reportTriangleType());
13     }
14 }
```

mutant3

```
1  package triangle.mutant3;
2  import org.junit.Test;
3
4  import static org.junit.Assert.*;
5
6  public class TriangleMutationTest {
7      // Test for Mutant 3
8      @Test
9      public void testMutant3() {
10         assertEquals("Mutant 3 should report ISOSCELES for sides 3, 2, 2",    Expected [ISOSCELES] but was [SCALENE]
11                 TriangleSides.TriangleType.ISOSCELES,
12                 new TriangleSides(a:3, b:2, c:2).reportTriangleType());
13     }
14 }
```

mutant4

```
1  package triangle.mutant4;
2  import org.junit.Test;
3
4  import static org.junit.Assert.*;
5
6  public class TriangleMutationTest {
7      // Test for Mutant 4
8      @Test
9      public void testMutant4() {
10         assertEquals("Mutant 4 should report SCALENE for sides 3, 4, 5",    Expected [SCALENE] but was [ISOSCELES]
11                 TriangleSides.TriangleType.SCALENE,
12                 new TriangleSides(a:3, b:4, c:5).reportTriangleType());
13     }
14 }
```

mutant5

```java
package triangle.mutant5;
import org.junit.Test;

import static org.junit.Assert.*;

public class TriangleMutationTest {
    // Test for Mutant 5
    @Test
    public void testMutant5() {
        assertEquals("Mutant 5 should report NOTRIANGLE for sides 5, 1, 1",    Expected [NOTRIANGLE] but was [ISOSCELES]
                TriangleSides.TriangleType.NOTRIANGLE,
                new TriangleSides(a:5, b:1, c:1).reportTriangleType());
    }
}
```

mutant6

```java
package triangle.mutant6;
import org.junit.Test;

import static org.junit.Assert.*;

public class TriangleMutationTest {
    // Test for Mutant 6
    @Test
    public void testMutant6() {
        assertEquals("Mutant 6 should report SCALENE for sides 4, 3, 5",    Expected [SCALENE] but was [ISOSCELES]
                TriangleSides.TriangleType.SCALENE,
                new TriangleSides(a:4, b:3, c:5).reportTriangleType());
    }
}
```

mutant7

```java
package triangle.mutant7;
import org.junit.Test;

import static org.junit.Assert.*;

public class TriangleMutationTest {
    // Test for Mutant 7
    @Test
    public void testMutant7() {
        assertEquals("Mutant 7 should report NOTRIANGLE for sides 2, 3, 1",    Expected [NOTRIANGLE] but was [SCALENE]
                TriangleSides.TriangleType.NOTRIANGLE,
                new TriangleSides(a:2, b:3, c:1).reportTriangleType());
    }
}
```

mutant8

```java
package triangle.mutant8;
import org.junit.Test;

import static org.junit.Assert.*;

public class TriangleMutationTest {
    // Test for Mutant 8
    @Test
    public void testMutant8() {
        assertEquals("Mutant 8 should report Undefined Behavior for sides 0, 2, 3",
                TriangleSides.TriangleType.NOTRIANGLE,
                new TriangleSides(a:0, b:2, c:3).reportTriangleType());
    }
}
```

mutant9

```java
package triangle.mutant9;
import org.junit.Test;

import static org.junit.Assert.*;

public class TriangleMutationTest {
    // Test for Mutant 9
    @Test
    public void testMutant9() {
        assertEquals("Correct should report NOTRIANGLE for sides -1, 2, 2",
                TriangleSides.TriangleType.NOTRIANGLE,
                new TriangleSides(-1, b:2, c:2).reportTriangleType());
    }
}
```

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

2. 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

2.1 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)

2.2 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

2.3 Write a Junit class TaxReturnTest.java in the same folder as TaxReturn.java.

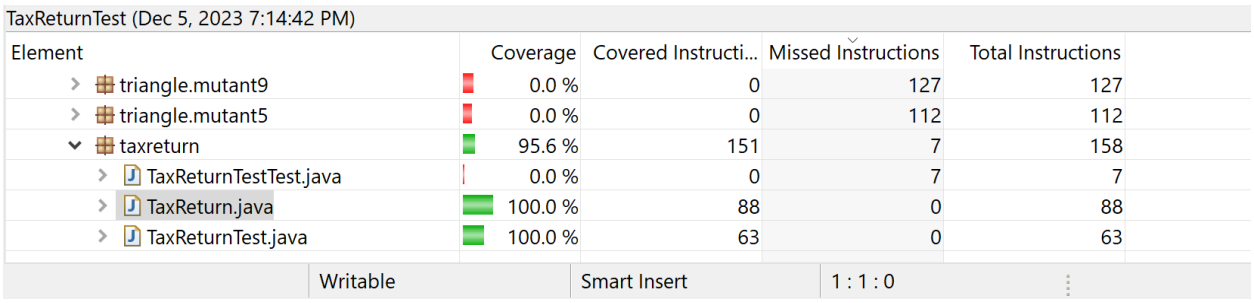
```
1  package taxreturn;
2
3  import org.junit.jupiter.api.Test;
4
5  public class TaxReturnTest {
6
7      @Test
8      public void testSingleLowIncome() {
9          TaxReturn taxReturn = new TaxReturn(20000, TaxReturn.SINGLE);
10         double tax = taxReturn.getTax();
11         // No assertions needed, just invoking the method for coverage
12     }
13
14     @Test
15     public void testSingleMidIncome() {
16         TaxReturn taxReturn = new TaxReturn(30000, TaxReturn.SINGLE);
17         double tax = taxReturn.getTax();
18         // No assertions needed
19     }
20
21     @Test
22     public void testSingleHighIncome() {
23         TaxReturn taxReturn = new TaxReturn(60000, TaxReturn.SINGLE);
24         double tax = taxReturn.getTax();
25         // No assertions needed
26     }
27
28     @Test
29     public void testMarriedLowIncome() {
30         TaxReturn taxReturn = new TaxReturn(30000, TaxReturn.MARRIED);
31         double tax = taxReturn.getTax();
32         // No assertions needed
33     }
34
```

```java
    @Test
    public void testMarriedMidIncome() {
        TaxReturn taxReturn = new TaxReturn(40000, TaxReturn.MARRIED);
        double tax = taxReturn.getTax();
        // No assertions needed
    }

    @Test
    public void testMarriedHighIncome() {
        TaxReturn taxReturn = new TaxReturn(90000, TaxReturn.MARRIED);
        double tax = taxReturn.getTax();
        // No assertions needed
    }
}
```

```java

    private static final double MARRIED_BASE2 = 5370;
    private static final double MARRIED_BASE3 = 19566;

    private double income;
    private int status;

    public TaxReturn(double anIncome, int aStatus) {
        income = anIncome;
        status = aStatus;
    }

    public double getTax() {
        double tax = 0;

        if (status == SINGLE) {
            if (income <= SINGLE_CUTOFF1)
                tax = RATE1 * income;
            else if (income <= SINGLE_CUTOFF2)
                tax = SINGLE_BASE2 + RATE2 * (income - SINGLE_CUTOFF1);
            else
                tax = SINGLE_BASE3 + RATE3 * (income - SINGLE_CUTOFF2);
        } else {
            if (income <= MARRIED_CUTOFF1)
                tax = RATE1 * income;
            else if (income <= MARRIED_CUTOFF2)
                tax = MARRIED_BASE2 + RATE2 * (income - MARRIED_CUTOFF1);
            else
                tax = MARRIED_BASE3 + RATE3 * (income - MARRIED_CUTOFF2);
        }
        return tax;
    }

}
```

Problems Javadoc Declaration Coverage

TaxReturnTest (Dec 5, 2023 7:14:42 PM)



3. Modified Condition/Decision Coverage (10 points)

Create a set of four tests that meets the MC/DC criterion of (C_1 and C_2) or C_3 where C_1, C_2, and C_3 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.