**Question 1**

**﻿***A screenshot of your finished Registration.name() function from a3.java, showing the additions*

*you made to fulfill the pre- and post-conditions*

A computer screen with text and images

Description automatically generated

﻿

*A screenshot of your code from A3Test.java, showing your tests. Take multiple screenshots if*

*necessary to show all your tests.*

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

﻿A screenshot of your test results from IntelliJ (in the bottom left corner of your screen),

showing all your tests passing. Please expand tests that have been grouped together using the

Expand All button above the tests results window.

**A screenshot of a computer

Description automatically generated**

**Question 2**  
(a) ﻿Write JUnit test cases to give 100%-line coverage. (4M)

A screenshot of a computer program

Description automatically generated

﻿(b) Perform Mutation testing on this function. List all mutations done by the system. Submit the

screenshot of Mutations. (2M)

A screenshot of a computer

Description automatically generated

﻿c) Write additional JUnit test cases to strengthen your test suite to kill all mutants. Specify which test

case kills which mutant. Submit screenshots after killing each mutant. (6M)

Added test cases: **testComplexAdditionWithVoidMethodCall()**, **testComplexAdditionWithTrueReturn()**, **testComplexAdditionWithPrimitiveReturn()**, **testComplexAdditionWithNegatedConditional()**, **testComplexAdditionWithNullReturn()**  
  
A screenshot of a computer

Description automatically generated

Final Test Code:

package a3;  
import org.junit.jupiter.api.Test;  
  
public class CalculatorTest {  
 @Test  
 public void testComplexAdditionWithNegativeResult() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(-1, 10);  
 assert -9 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithPositiveResult() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(3, 7);  
 assert 10 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithZeroResult() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(-2, 2);  
 assert 0 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithBoundaryCondition() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(1, 1);  
 assert -2 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithEmptyReturn() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(5, 5);  
 assert 10 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithFalseReturn() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(0, 0);  
 assert 0 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithIncrement() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(10, 1);  
 assert 11 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithInvertedNegative() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(3, -3);  
 assert 0 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithIncorrectMath() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(2, 3);  
 assert 5 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithNegatedConditional() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(5, 0);  
 assert 5 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithNullReturn() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(4, 2);  
 assert 6 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithPrimitiveReturn() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(7, 3);  
 assert 10 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithTrueReturn() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(0, 0);  
 assert 0 == result;  
 }  
  
 @Test  
 public void testComplexAdditionWithVoidMethodCall() {  
 Calculator calculator = new Calculator();  
 int result = calculator.ComplexAdd(6, 2);  
 assert 8 == result;  
 }  
}

And the main file unchanged:

package a3;  
  
public class Calculator {  
 public int ComplexAdd(int a, int b)  
 {  
 if (a < 2) { return (a+b) \* -1; }  
 else { return a+b; }  
 }  
}