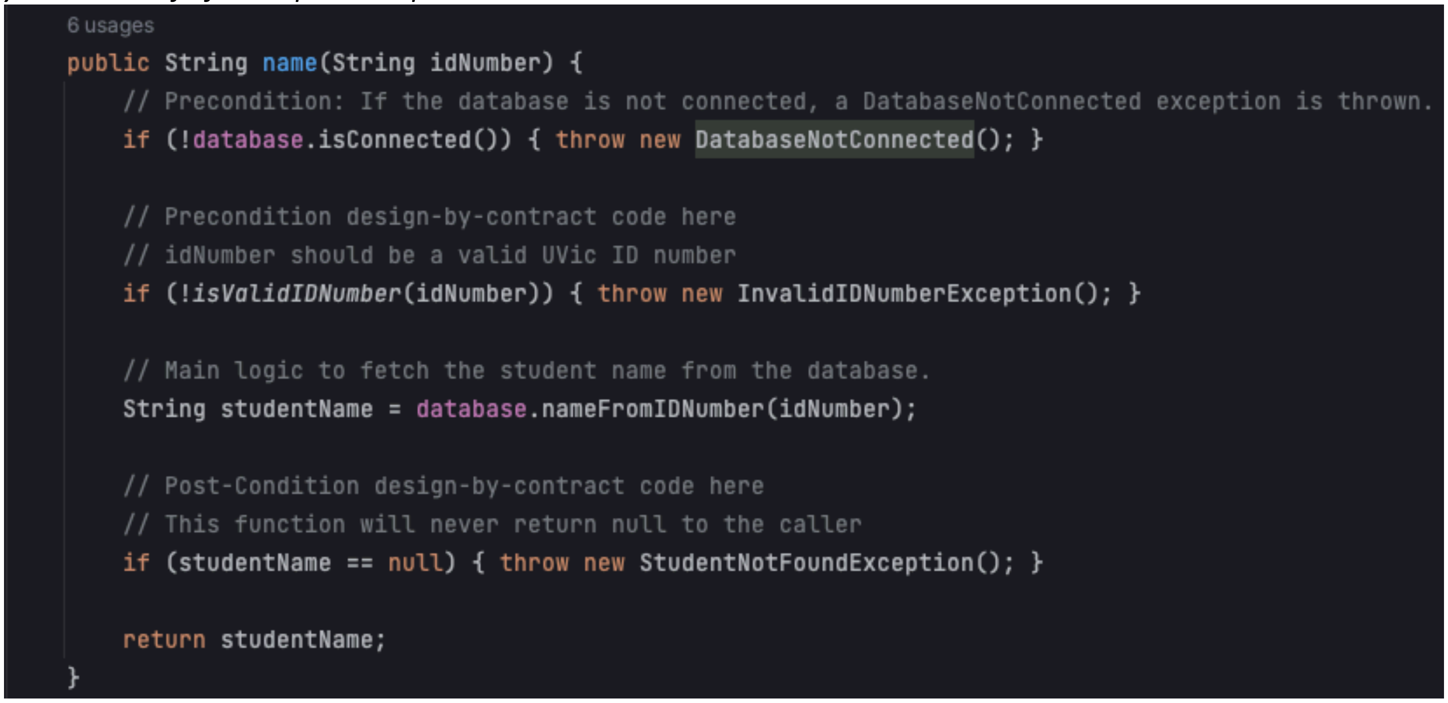
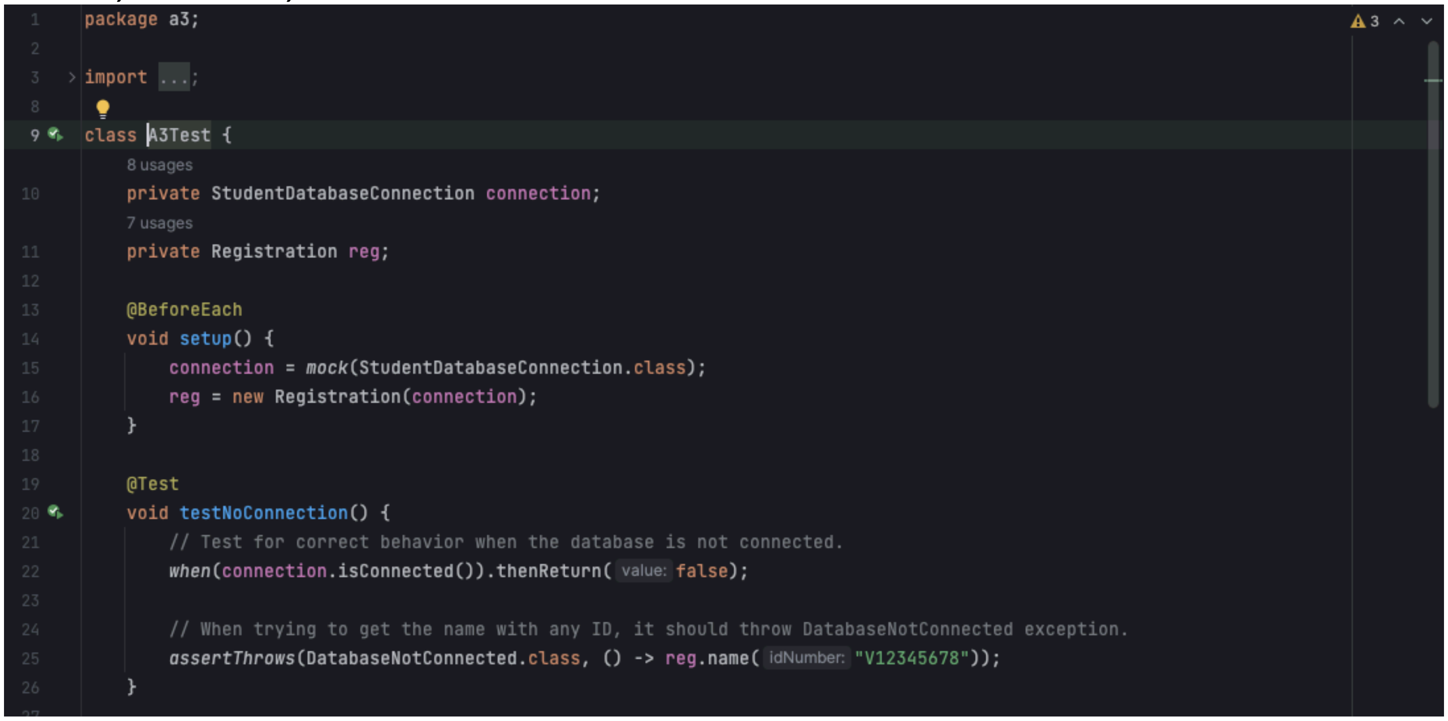
Question 1

A screenshot of your finished Registration.name() function from a3.java, showing the additions you made to fulfill the pre- and post-conditions

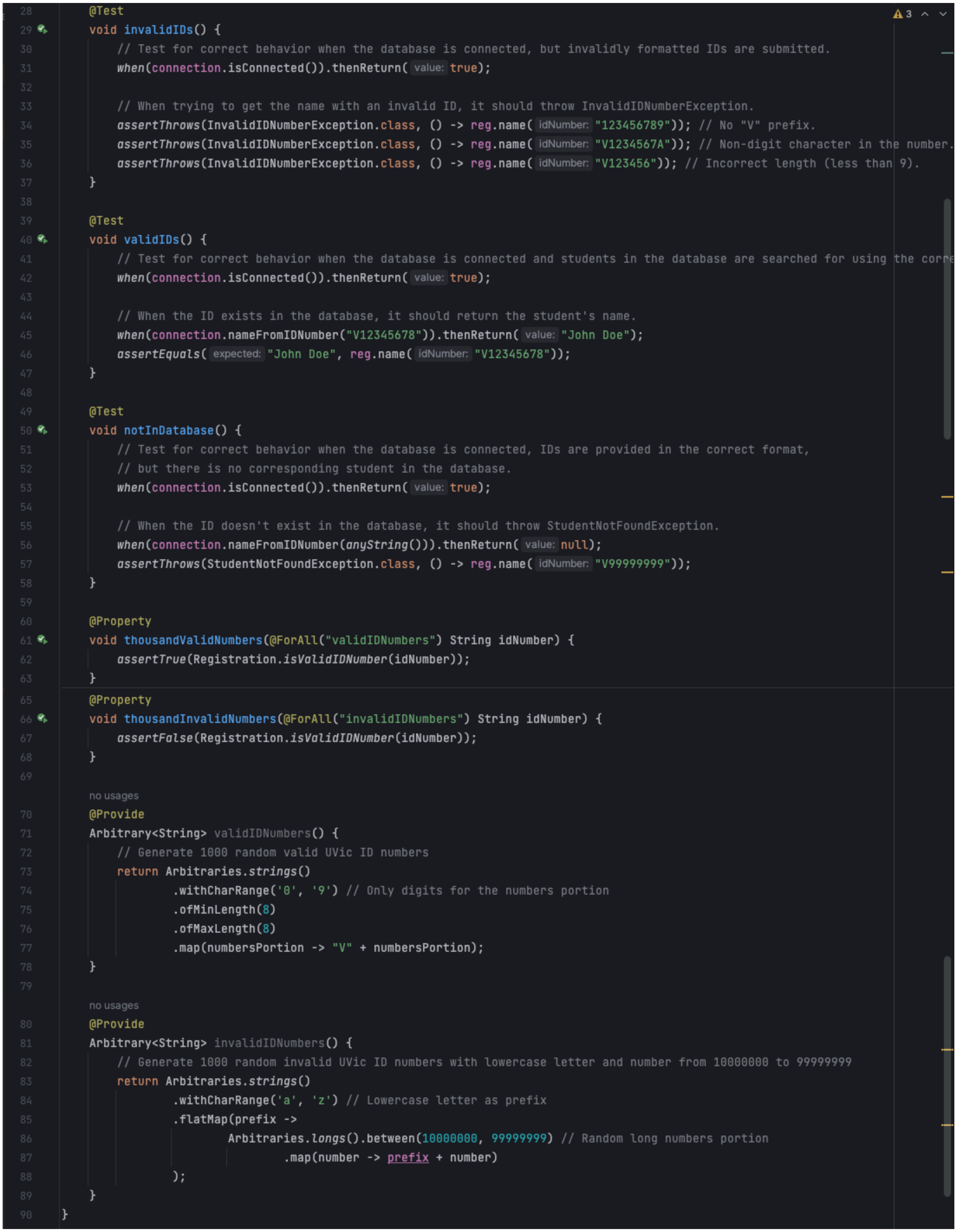


```java
6 usages
public String name(String idNumber) {
    // Precondition: If the database is not connected, a DatabaseNotConnected exception is thrown.
    if (!database.isConnected()) { throw new DatabaseNotConnected(); }

    // Precondition design-by-contract code here
    // idNumber should be a valid UVic ID number
    if (!isValidIDNumber(idNumber)) { throw new InvalidIDNumberException(); }

    // Main logic to fetch the student name from the database.
    String studentName = database.nameFromIDNumber(idNumber);

    // Post-Condition design-by-contract code here
    // This function will never return null to the caller
    if (studentName == null) { throw new StudentNotFoundException(); }

    return studentName;
}
```

A screenshot of your code from A3Test.java, showing your tests. Take multiple screenshots if necessary to show all your tests.

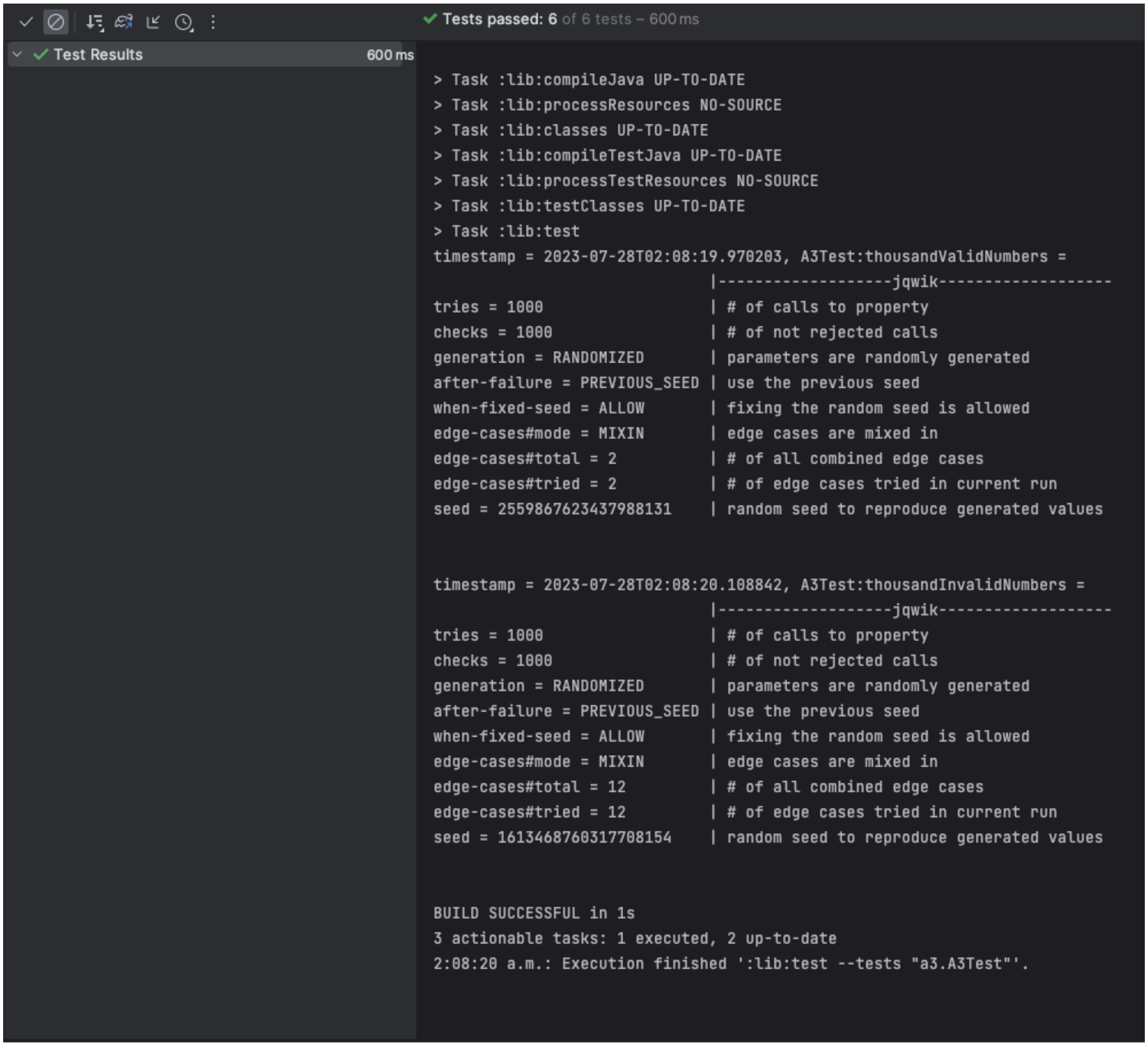


```java
package a3;

import ...;

class A3Test {
    8 usages
    private StudentDatabaseConnection connection;
    7 usages
    private Registration reg;

    @BeforeEach
    void setup() {
        connection = mock(StudentDatabaseConnection.class);
        reg = new Registration(connection);
    }

    @Test
    void testNoConnection() {
        // Test for correct behavior when the database is not connected.
        when(connection.isConnected()).thenReturn( value: false);

        // When trying to get the name with any ID, it should throw DatabaseNotConnected exception.
        assertThrows(DatabaseNotConnected.class, () -> reg.name( idNumber: "V12345678"));
    }
```



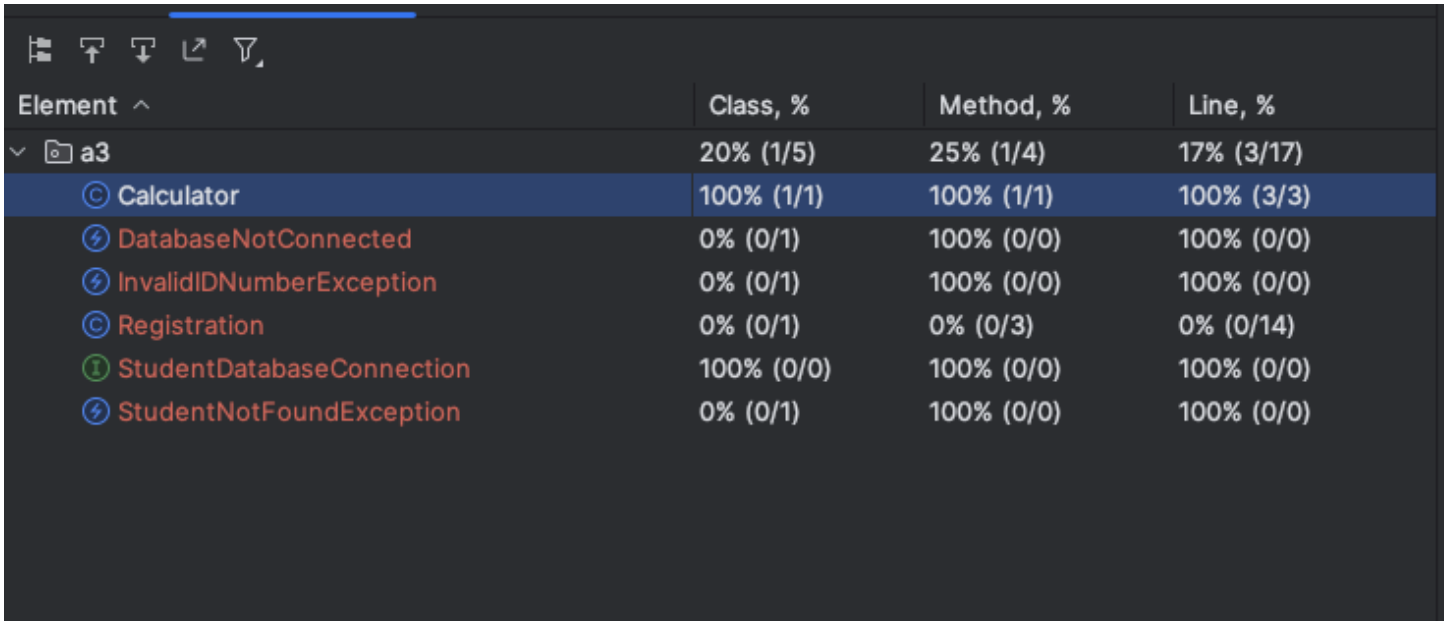
```java
    @Test
    void invalidIDs() {
        // Test for correct behavior when the database is connected, but invalidly formatted IDs are submitted.
        when(connection.isConnected()).thenReturn( value: true);

        // When trying to get the name with an invalid ID, it should throw InvalidIDNumberException.
        assertThrows(InvalidIDNumberException.class, () -> reg.name( idNumber: "123456789")); // No "V" prefix.
        assertThrows(InvalidIDNumberException.class, () -> reg.name( idNumber: "V1234567A")); // Non-digit character in the number.
        assertThrows(InvalidIDNumberException.class, () -> reg.name( idNumber: "V123456")); // Incorrect length (less than 9).
    }

    @Test
    void validIDs() {
        // Test for correct behavior when the database is connected and students in the database are searched for using the corre
        when(connection.isConnected()).thenReturn( value: true);

        // When the ID exists in the database, it should return the student's name.
        when(connection.nameFromIDNumber("V12345678")).thenReturn( value: "John Doe");
        assertEquals( expected: "John Doe", reg.name( idNumber: "V12345678"));
    }

    @Test
    void notInDatabase() {
        // Test for correct behavior when the database is connected, IDs are provided in the correct format,
        // but there is no corresponding student in the database.
        when(connection.isConnected()).thenReturn( value: true);

        // When the ID doesn't exist in the database, it should throw StudentNotFoundException.
        when(connection.nameFromIDNumber(anyString())).thenReturn( value: null);
        assertThrows(StudentNotFoundException.class, () -> reg.name( idNumber: "V99999999"));
    }

    @Property
    void thousandValidNumbers(@ForAll("validIDNumbers") String idNumber) {
        assertTrue(Registration.isValidIDNumber(idNumber));
    }
    @Property
    void thousandInvalidNumbers(@ForAll("invalidIDNumbers") String idNumber) {
        assertFalse(Registration.isValidIDNumber(idNumber));
    }

    no usages
    @Provide
    Arbitrary<String> validIDNumbers() {
        // Generate 1000 random valid UVic ID numbers
        return Arbitraries.strings()
                .withCharRange('0', '9') // Only digits for the numbers portion
                .ofMinLength(8)
                .ofMaxLength(8)
                .map(numbersPortion -> "V" + numbersPortion);
    }

    no usages
    @Provide
    Arbitrary<String> invalidIDNumbers() {
        // Generate 1000 random invalid UVic ID numbers with lowercase letter and number from 10000000 to 99999999
        return Arbitraries.strings()
                .withCharRange('a', 'z') // Lowercase letter as prefix
                .flatMap(prefix ->
                        Arbitraries.longs().between(10000000, 99999999) // Random long numbers portion
                                .map(number -> prefix + number)
                );
    }
}
```

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.



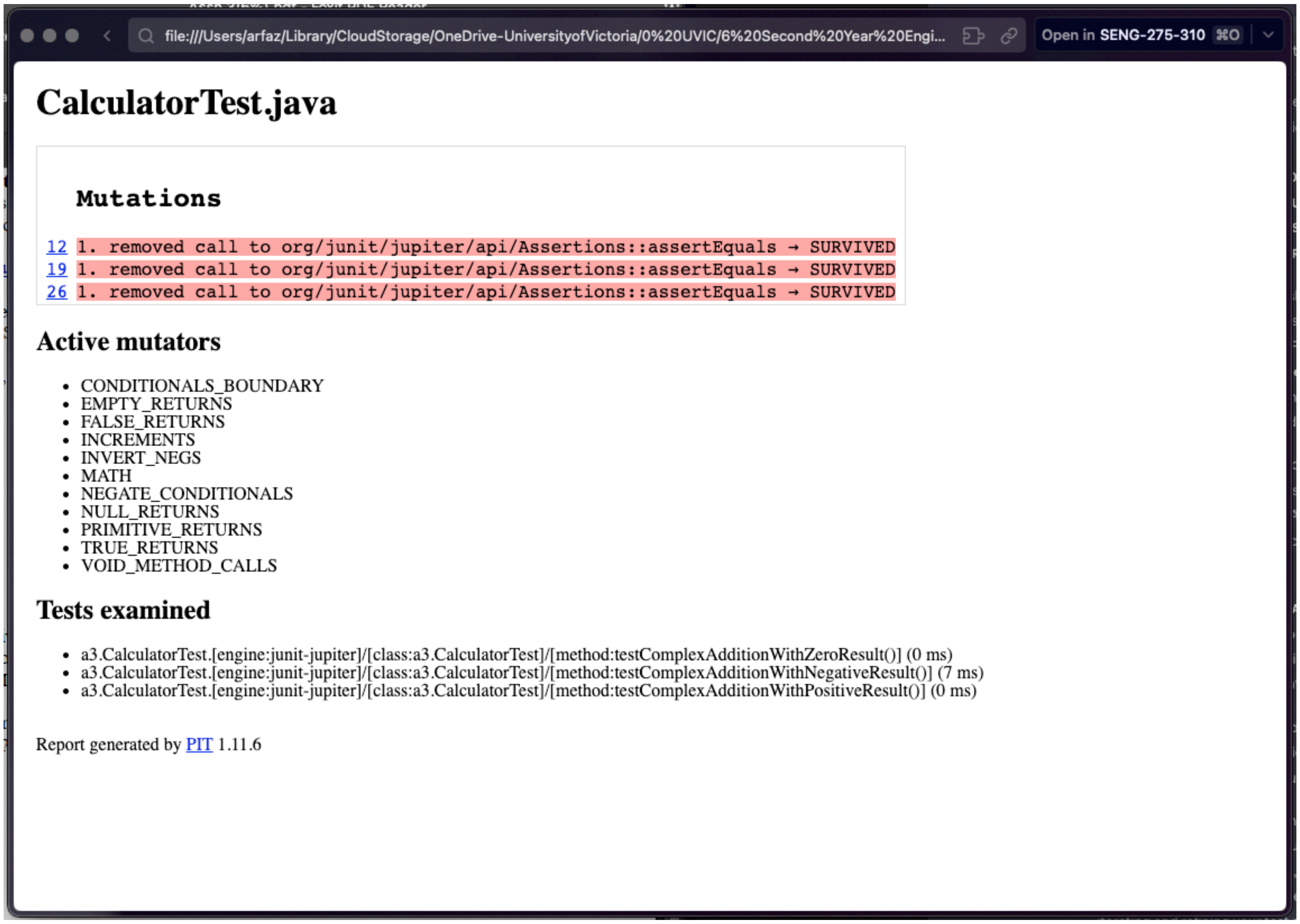
```
> Task :lib:compileJava UP-TO-DATE
> Task :lib:processResources NO-SOURCE
> Task :lib:classes UP-TO-DATE
> Task :lib:compileTestJava UP-TO-DATE
> Task :lib:processTestResources NO-SOURCE
> Task :lib:testClasses UP-TO-DATE
> Task :lib:test
timestamp = 2023-07-28T02:08:19.970203, A3Test:thousandValidNumbers =
                              |-------------------jqwik-------------------
tries = 1000                  | # of calls to property
checks = 1000                 | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = PREVIOUS_SEED | use the previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 2          | # of all combined edge cases
edge-cases#tried = 2          | # of edge cases tried in current run
seed = 2559867623437988131    | random seed to reproduce generated values


timestamp = 2023-07-28T02:08:20.108842, A3Test:thousandInvalidNumbers =
                              |-------------------jqwik-------------------
tries = 1000                  | # of calls to property
checks = 1000                 | # of not rejected calls
generation = RANDOMIZED       | parameters are randomly generated
after-failure = PREVIOUS_SEED | use the previous seed
when-fixed-seed = ALLOW       | fixing the random seed is allowed
edge-cases#mode = MIXIN       | edge cases are mixed in
edge-cases#total = 12         | # of all combined edge cases
edge-cases#tried = 12         | # of edge cases tried in current run
seed = 1613468760317708154    | random seed to reproduce generated values


BUILD SUCCESSFUL in 1s
3 actionable tasks: 1 executed, 2 up-to-date
2:08:20 a.m.: Execution finished ':lib:test --tests "a3.A3Test"'.
```

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



Element	Class, %	Method, %	Line, %
a3	20% (1/5)	25% (1/4)	17% (3/17)
Calculator	100% (1/1)	100% (1/1)	100% (3/3)
DatabaseNotConnected	0% (0/1)	100% (0/0)	100% (0/0)
InvalidIDNumberException	0% (0/1)	100% (0/0)	100% (0/0)
Registration	0% (0/1)	0% (0/3)	0% (0/14)
StudentDatabaseConnection	100% (0/0)	100% (0/0)	100% (0/0)
StudentNotFoundException	0% (0/1)	100% (0/0)	100% (0/0)

(b) Perform Mutation testing on this function. List all mutations done by the system. Submit the screenshot of Mutations. (2M)



file:///Users/arfaz/Library/CloudStorage/OneDrive-UniversityofVictoria/0%20UVIC/6%20Second%20Year%20Engi...

Open in SENG-275-310

CalculatorTest.java

Mutations

```
12 1. removed call to org/junit/jupiter/api/Assertions::assertEquals → SURVIVED
19 1. removed call to org/junit/jupiter/api/Assertions::assertEquals → SURVIVED
26 1. removed call to org/junit/jupiter/api/Assertions::assertEquals → SURVIVED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

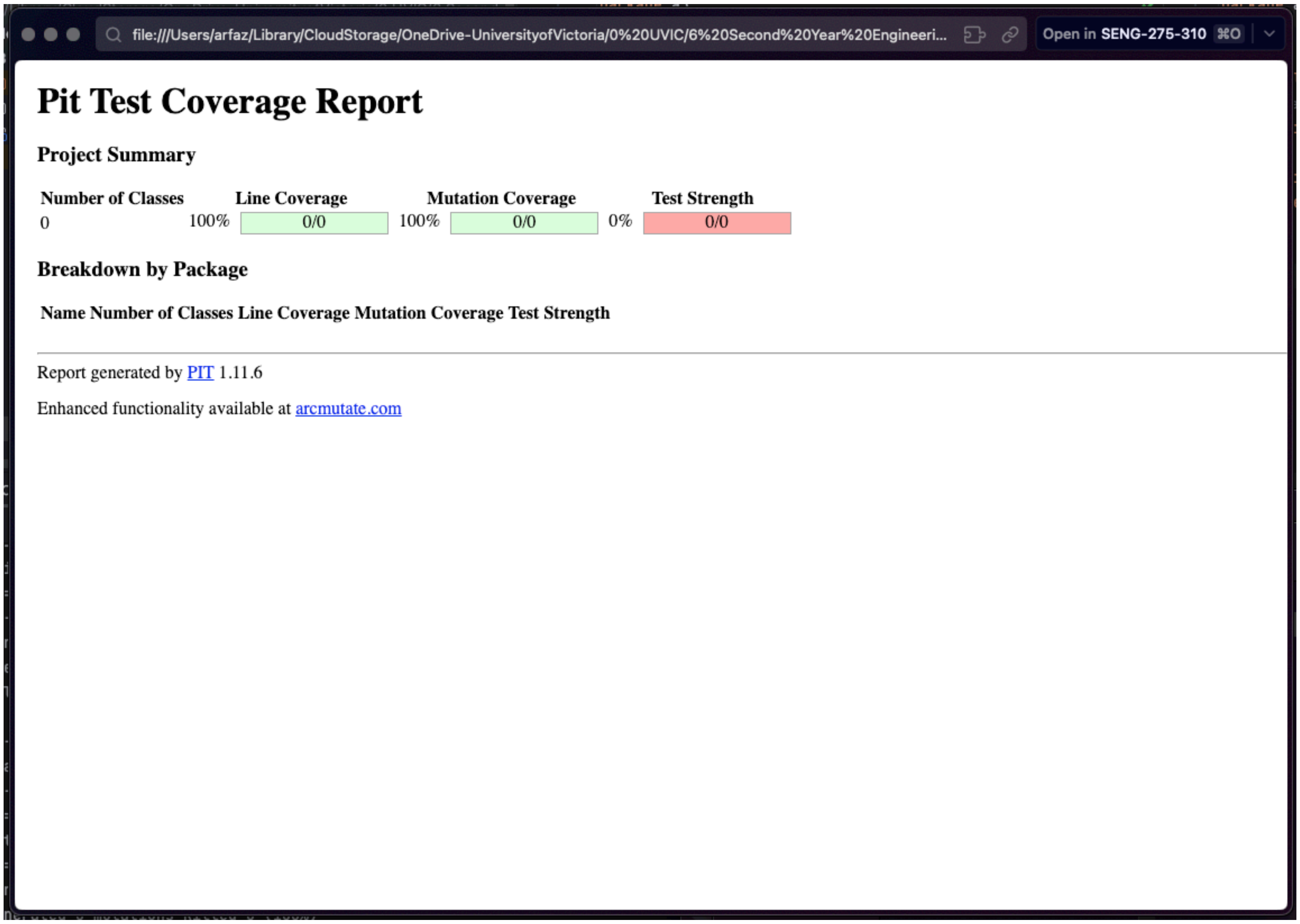
Tests examined

- a3.CalculatorTest.[engine:junit-jupiter]/[class:a3.CalculatorTest]/[method:testComplexAdditionWithZeroResult()] (0 ms)
- a3.CalculatorTest.[engine:junit-jupiter]/[class:a3.CalculatorTest]/[method:testComplexAdditionWithNegativeResult()] (7 ms)
- a3.CalculatorTest.[engine:junit-jupiter]/[class:a3.CalculatorTest]/[method:testComplexAdditionWithPositiveResult()] (0 ms)

Report generated by PIT 1.11.6

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()**


file:///Users/arfaz/Library/CloudStorage/OneDrive-UniversityofVictoria/0%20UVIC/6%20Second%20Year%20Engineeri...

Open in SENG-275-310 ⌘O

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
0	100%	0/0	100%	0/0	0%	0/0

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength

Report generated by PIT 1.11.6

Enhanced functionality available at arcmutate.com

Final Test Code:

```java
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:

```java

package a3;

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