Unit Testing using JUNIT

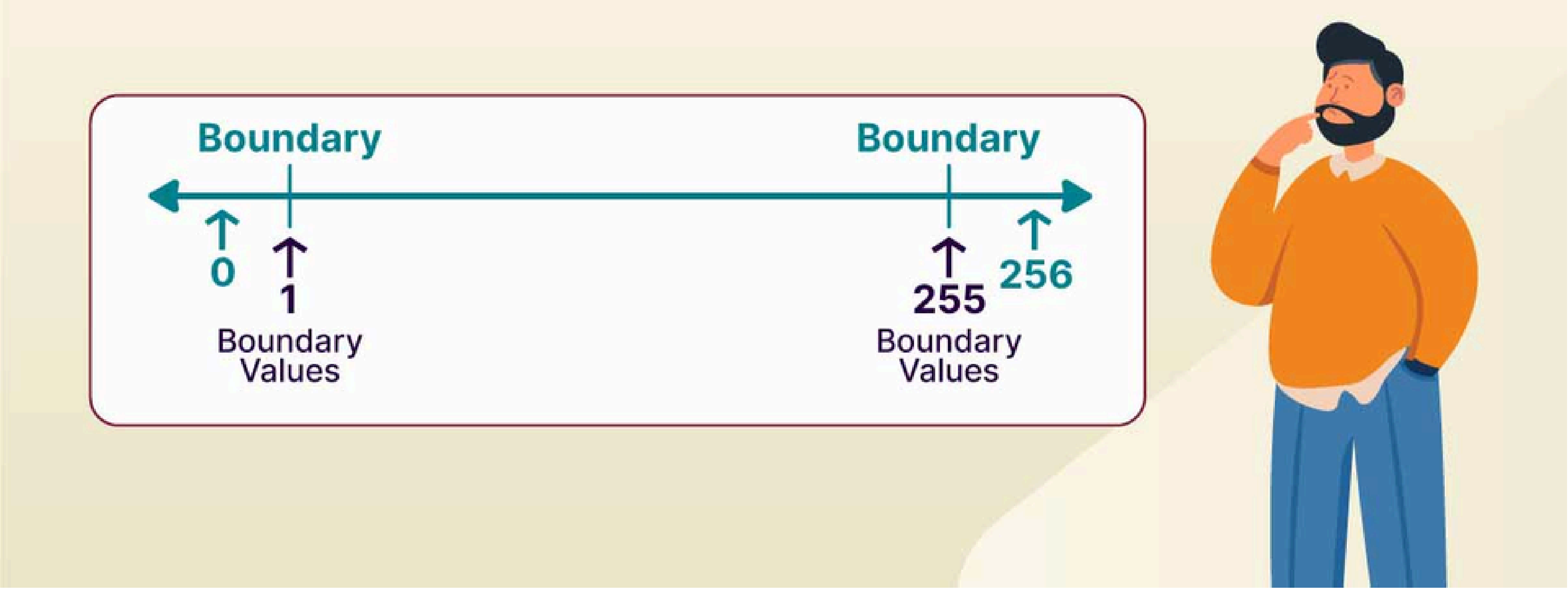
Presented by:
Sireesha Akurathi
Masters in Information Technology

What is Boundary Value Analysis (BVA)?

- Boundary Value Analysis is a black-box testing technique where test cases are designed based on the boundaries of input domains. It focuses on values at, just below, and just above the valid input limits.

Why Use BVA?

- Errors often occur at edge values
- Fewer test cases with higher fault detection rate
- Efficient for functions handling range-based inputs



What is Equivalence Class Testing (ECT)?

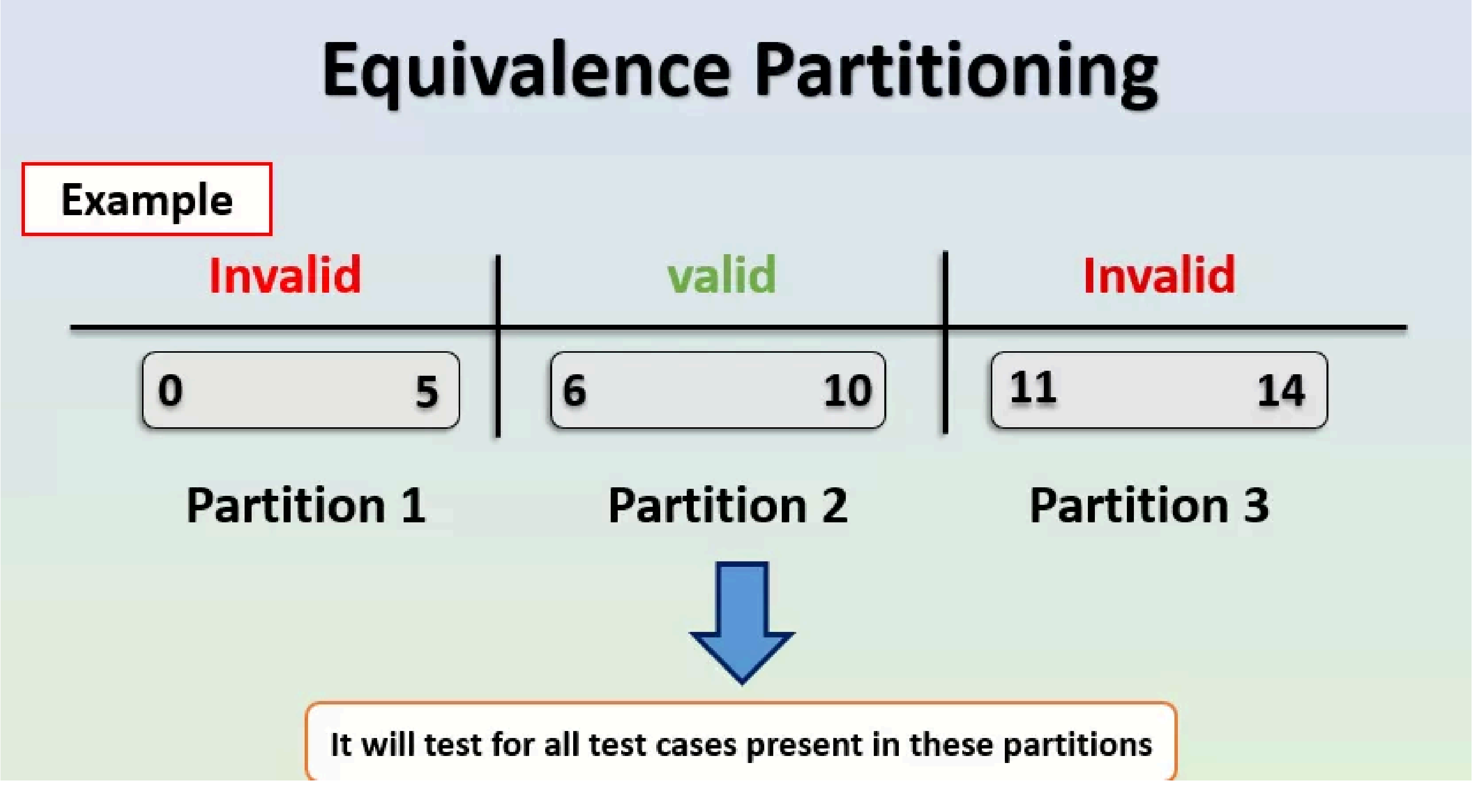
- Equivalence Class Testing is a black-box testing technique that divides input data into equivalence classes, where the system is expected to behave similarly for all values in the same class.
- You only need to test one value from each class, as others are assumed to behave the same.

Types of Classes:
Valid Class: Inputs that are acceptable and should produce expected results.
Invalid Class: Inputs that are outside the defined limits and should be rejected or trigger errors.

Example (Month Input):
Valid range: 1–12
Valid Class: {1, 6, 12} → Expect normal processing
Invalid Class: {0, 13, -5} → Expect error handling

Why Use ECT?

- Reduces the total number of test cases
- Ensures broad functional coverage
- Useful for validating input fields, form data, and user inputs



Comparision of BVA and ECT

Key Differences Between BVA & ECT

Feature	Boundary Value Analysis (BVA)	Equivalence Class Testing (ECT)
Focus	Inputs at the edges of a range	Representative inputs from partitions
Test Case Volume	Fewer, focused on min/max values	Fewer, focused on class representatives
Best For	Numeric ranges, limits	Input field validation, form data
Error Detection	Detects issues near boundary conditions	Detects logical errors in data groups
Example	Days: 1, 2, 30, 31	Days: Valid class {1–31}, Invalid {-1, 32}

Real-World Application of BVA & ECT

Smart Energy Systems

- BVA: Test sensor readings at threshold (e.g., voltage: 0V, 240V, 250V)
- ECT: Classify readings into safe, warning, and critical zones

Date Calculators / Scheduling Systems

- BVA: Input edge dates like Feb 28, 29, Mar 1
- ECT: Valid year range (1700–2024), Invalid (1699, 2025)

Web Forms & User Inputs

- BVA: Age field (Min: 18, Max: 100 → test 17, 18, 100, 101)
- ECT: Valid email formats vs invalid ones (missing @, domain, etc.)

Banking/Finance Applications

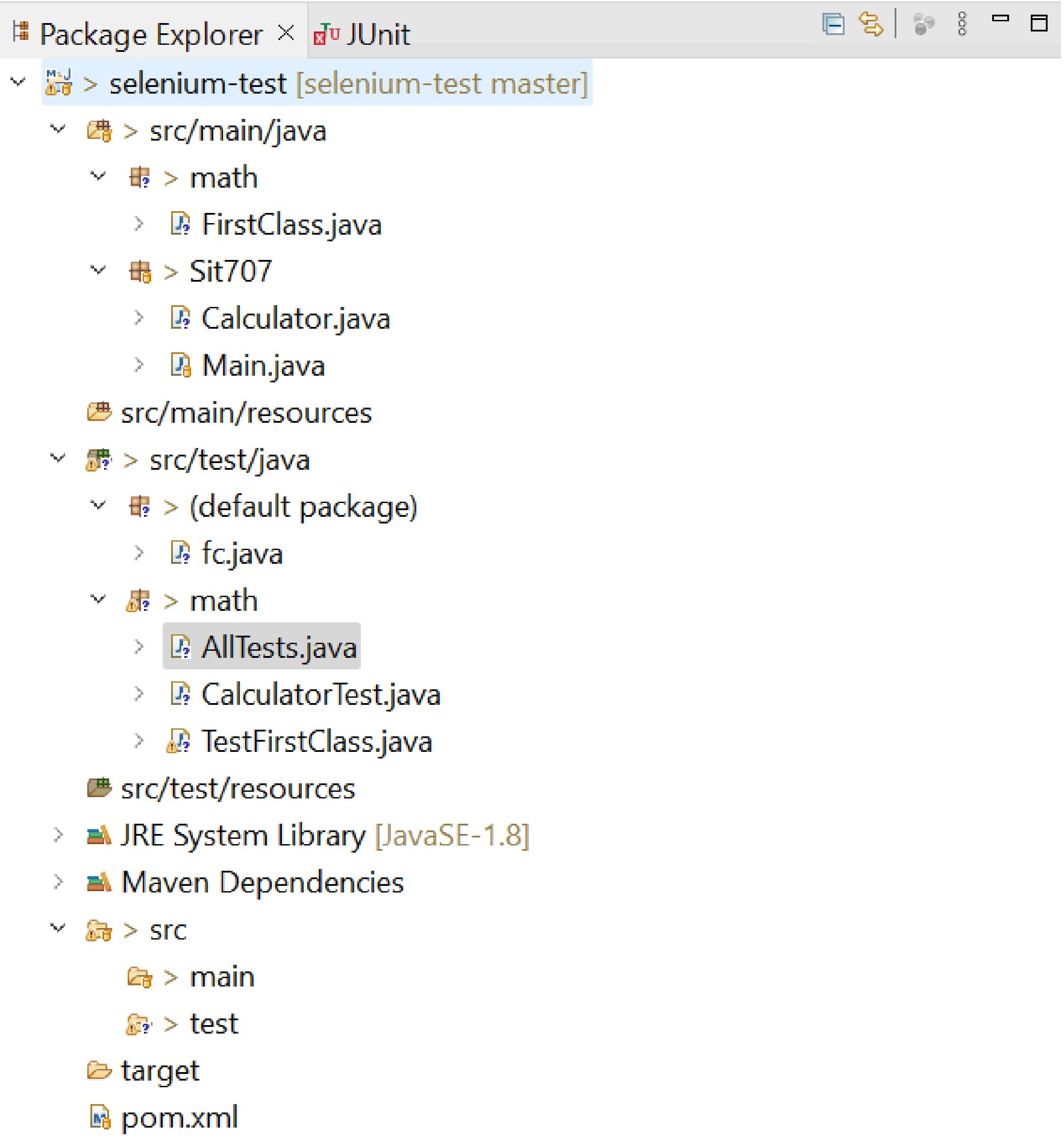
- BVA: Test transaction limits ($0, $1, $9999, $10000)
- ECT: Negative amounts (invalid), zero (edge), typical (valid)

JUnit Basics & Setup(Active Learning)

- Created a Maven Java project in Eclipse.
- Implemented a simple class: Calculator.java with multiply(int a, int b) method.
- Created CalculatorTest.java in the src/test/java folder.
- Used JUnit v4 annotations to write and run tests.

Key Takeaways

- @Test is used to mark a test method.
- Assert.assertEquals() checks expected vs actual values.
- Tests can be grouped using Test Suites.
- Proper test naming improves readability (e.g., testMultiplyPositiveNumbers()).



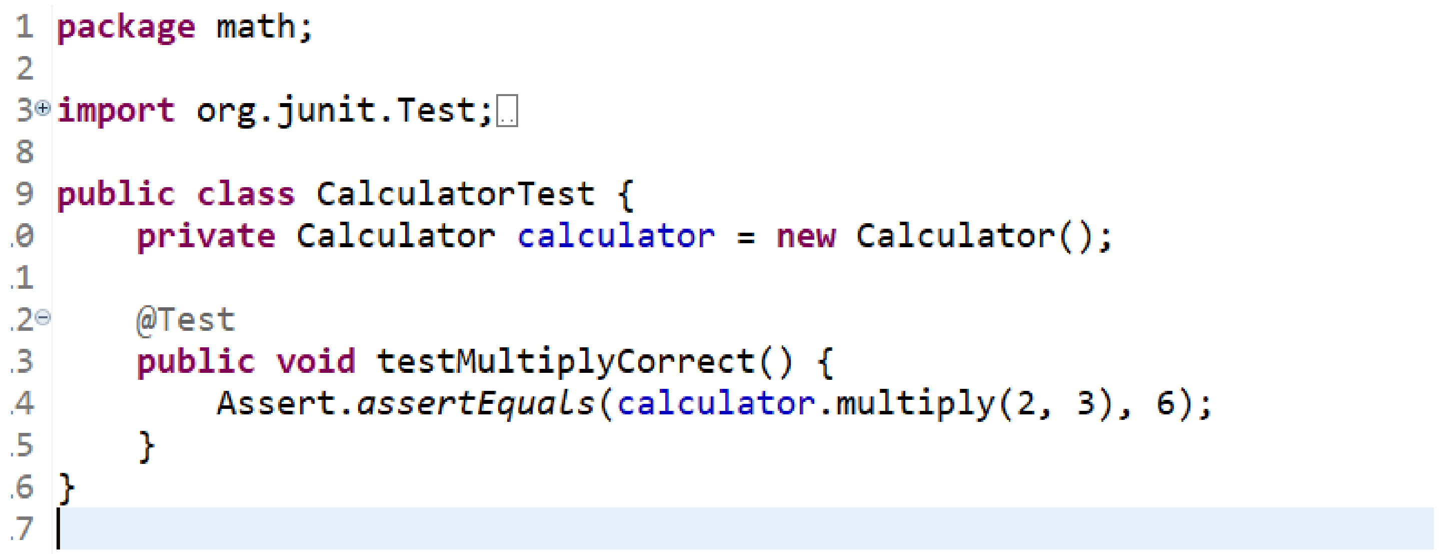
Package Explorer | JUnit

- > selenium-test [selenium-test master]
  - > src/main/java
    - > math
      - FirstClass.java
    - > Sit707
      - Calculator.java
      - Main.java
  - src/main/resources
  - > src/test/java
    - > (default package)
      - fc.java
    - > math
      - AllTests.java
      - CalculatorTest.java
      - TestFirstClass.java
  - src/test/resources
  - JRE System Library [JavaSE-1.8]
  - Maven Dependencies
  - > src
    - > main
    - > test
  - target
  - pom.xml

JUnit Basics & Setup(Active Learning)

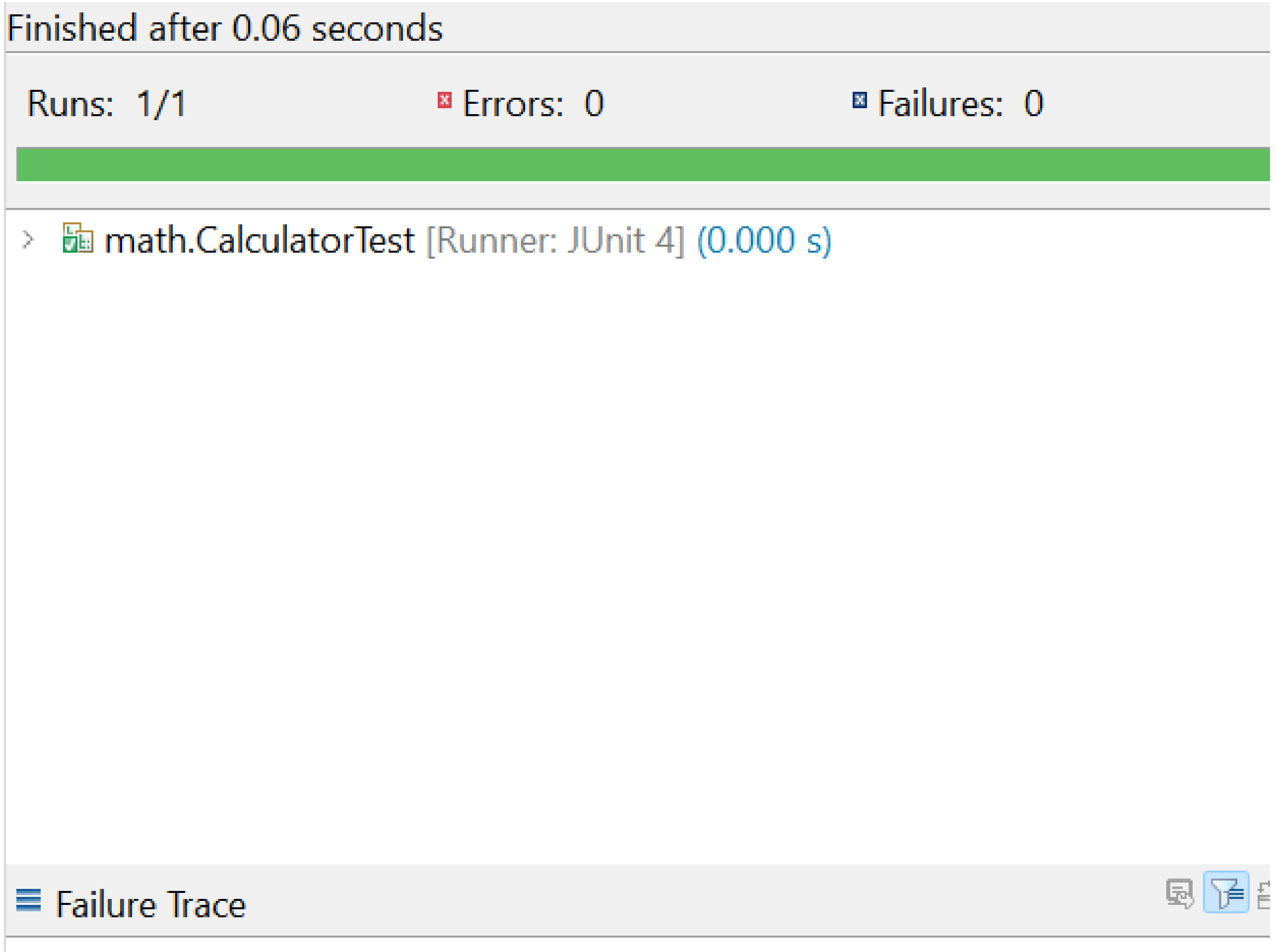
Tools Used

- Eclipse IDE
- JUnit v4
- Maven Project Structure



```
package math;

import org.junit.Test;

public class CalculatorTest {
    private Calculator calculator = new Calculator();

    @Test
    public void testMultiplyCorrect() {
        Assert.assertEquals(calculator.multiply(2, 3), 6);
    }
}

```

fig: CalculatorTest.java



Finished after 0.06 seconds

Runs: 1/1 Errors: 0 Failures: 0

math.CalculatorTest [Runner: JUnit 4] (0.000 s)

fig: JUNIT tab

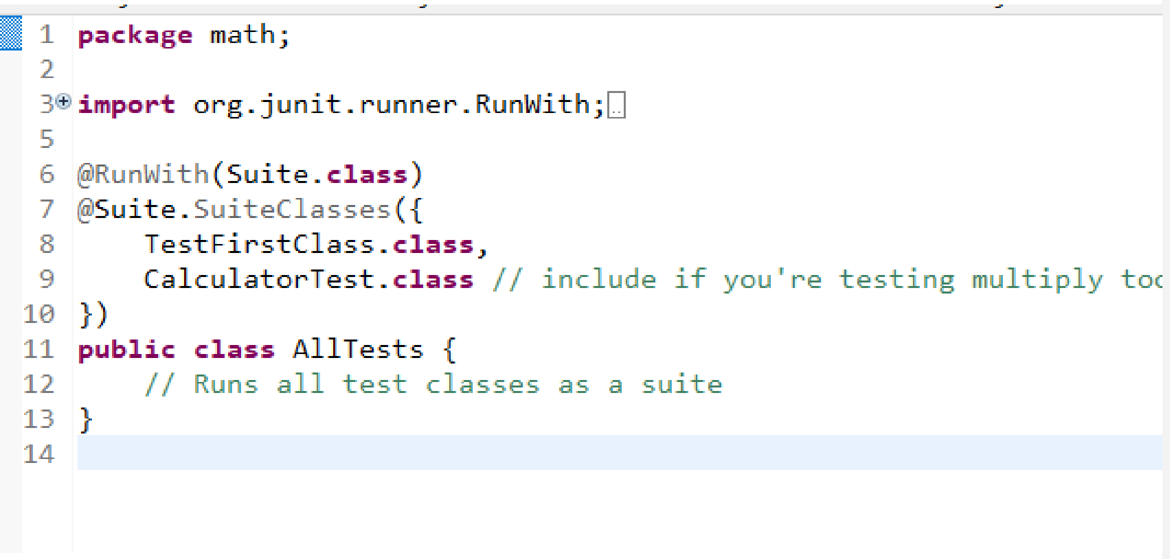
Task 1 – JUnit Tests for add() and concat()

What I Implemented:

- Created class: FirstClass.java
- add(int a, int b)
- concat(String a, String b)
- Created test class: TestFirstClass.java
- Wrote test methods with descriptive names for different input scenarios.

Test Suite:

Created AllTests.java to run all tests in one go.



```java
package math;

import org.junit.runner.RunWith;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestFirstClass.class,
    CalculatorTest.class // include if you're testing multiply to
})
public class AllTests {
    // Runs all test classes as a suite
}

```

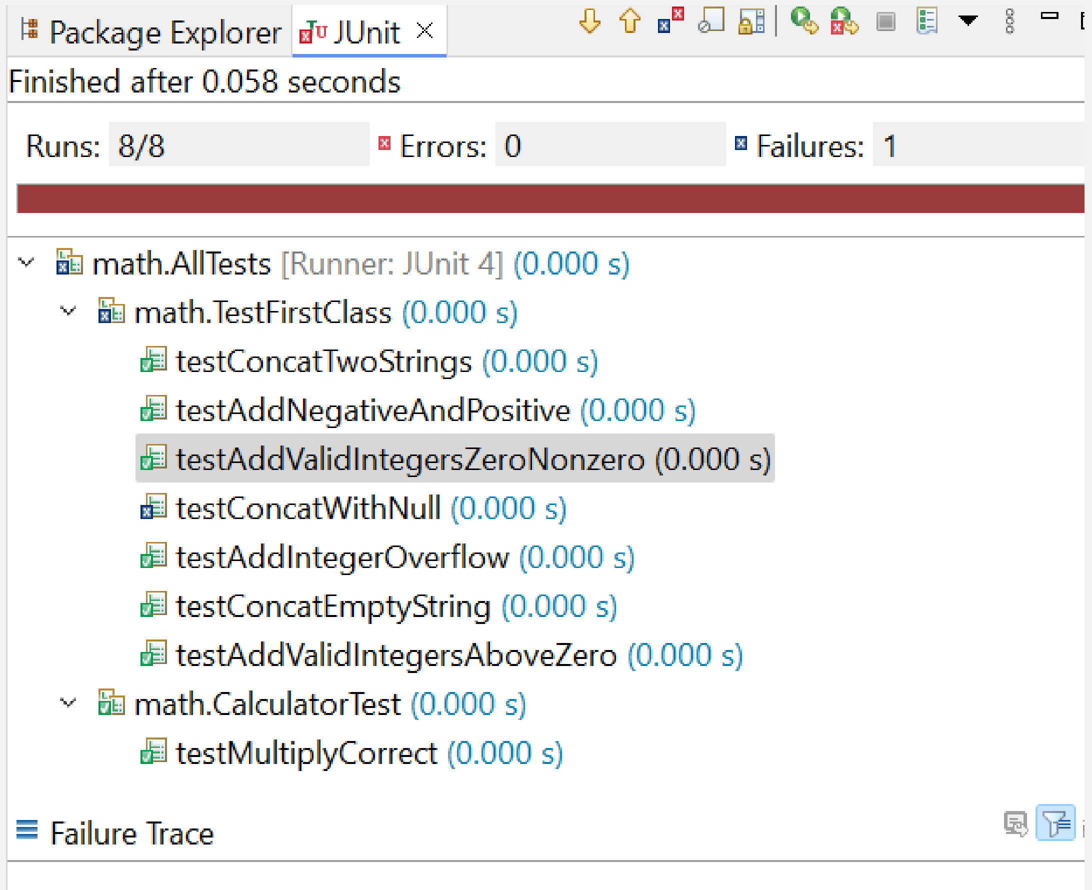
Test Case Examples:

add() Tests:

testAddValidIntegersAboveZero() → add(2, 2)
testAddNegativeAndPositive() → add(-2, 2)
testAddIntegerOverflow() → add(Integer.MAX_VALUE, 1)

concat() Tests:

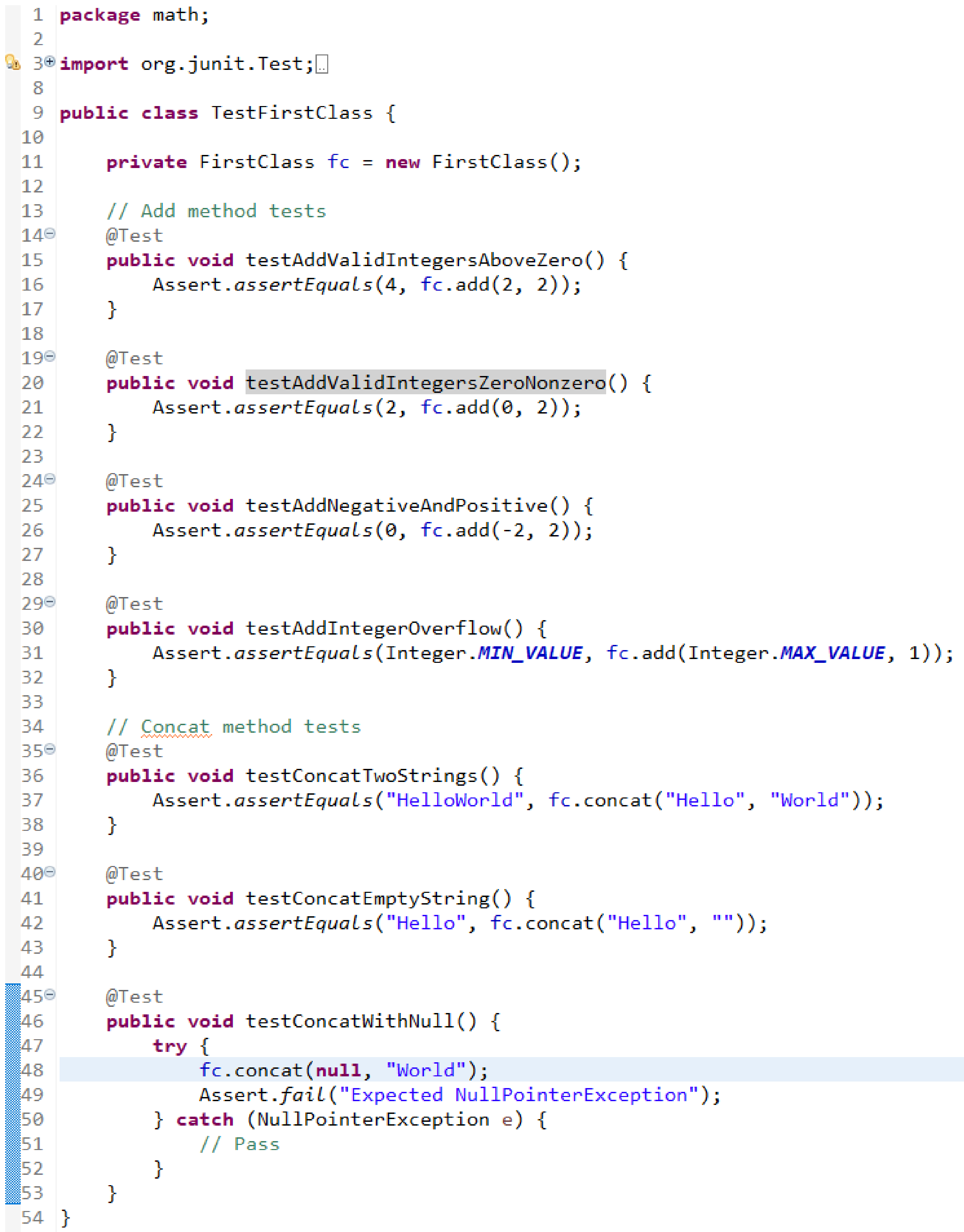
testConcatTwoStrings() → "Hello", "World"
testConcatEmptyString() → "Hello", ""
testConcatWithNull() → null, "World" → Expected to fail with NullPointerException



Finished after 0.058 seconds

Runs: 8/8 Errors: 0 Failures: 1

```
math.AllTests [Runner: JUnit 4] (0.000 s)
    math.TestFirstClass (0.000 s)
        testConcatTwoStrings (0.000 s)
        testAddNegativeAndPositive (0.000 s)
        testAddValidIntegersZeroNonzero (0.000 s)
        testConcatWithNull (0.000 s)
        testAddIntegerOverflow (0.000 s)
        testConcatEmptyString (0.000 s)
        testAddValidIntegersAboveZero (0.000 s)
    math.CalculatorTest (0.000 s)
        testMultiplyCorrect (0.000 s)
```



```java
package math;

import org.junit.Test;

public class TestFirstClass {

    private FirstClass fc = new FirstClass();

    // Add method tests
    @Test
    public void testAddValidIntegersAboveZero() {
        Assert.assertEquals(4, fc.add(2, 2));
    }

    @Test
    public void testAddValidIntegersZeroNonzero() {
        Assert.assertEquals(2, fc.add(0, 2));
    }

    @Test
    public void testAddNegativeAndPositive() {
        Assert.assertEquals(0, fc.add(-2, 2));
    }

    @Test
    public void testAddIntegerOverflow() {
        Assert.assertEquals(Integer.MIN_VALUE, fc.add(Integer.MAX_VALUE, 1));
    }

    // Concat method tests
    @Test
    public void testConcatTwoStrings() {
        Assert.assertEquals("HelloWorld", fc.concat("Hello", "World"));
    }

    @Test
    public void testConcatEmptyString() {
        Assert.assertEquals("Hello", fc.concat("Hello", ""));
    }

    @Test
    public void testConcatWithNull() {
        try {
            fc.concat(null, "World");
            Assert.fail("Expected NullPointerException");
        } catch (NullPointerException e) {
            // Pass
        }
    }
}
```

Task 1 Summary

add() Method:

- Passed for normal inputs: add(2, 2), add(0, 5)
- Broke on boundary overflow:
- add(Integer.MAX_VALUE, 1) → Result wraps to Integer.MIN_VALUE

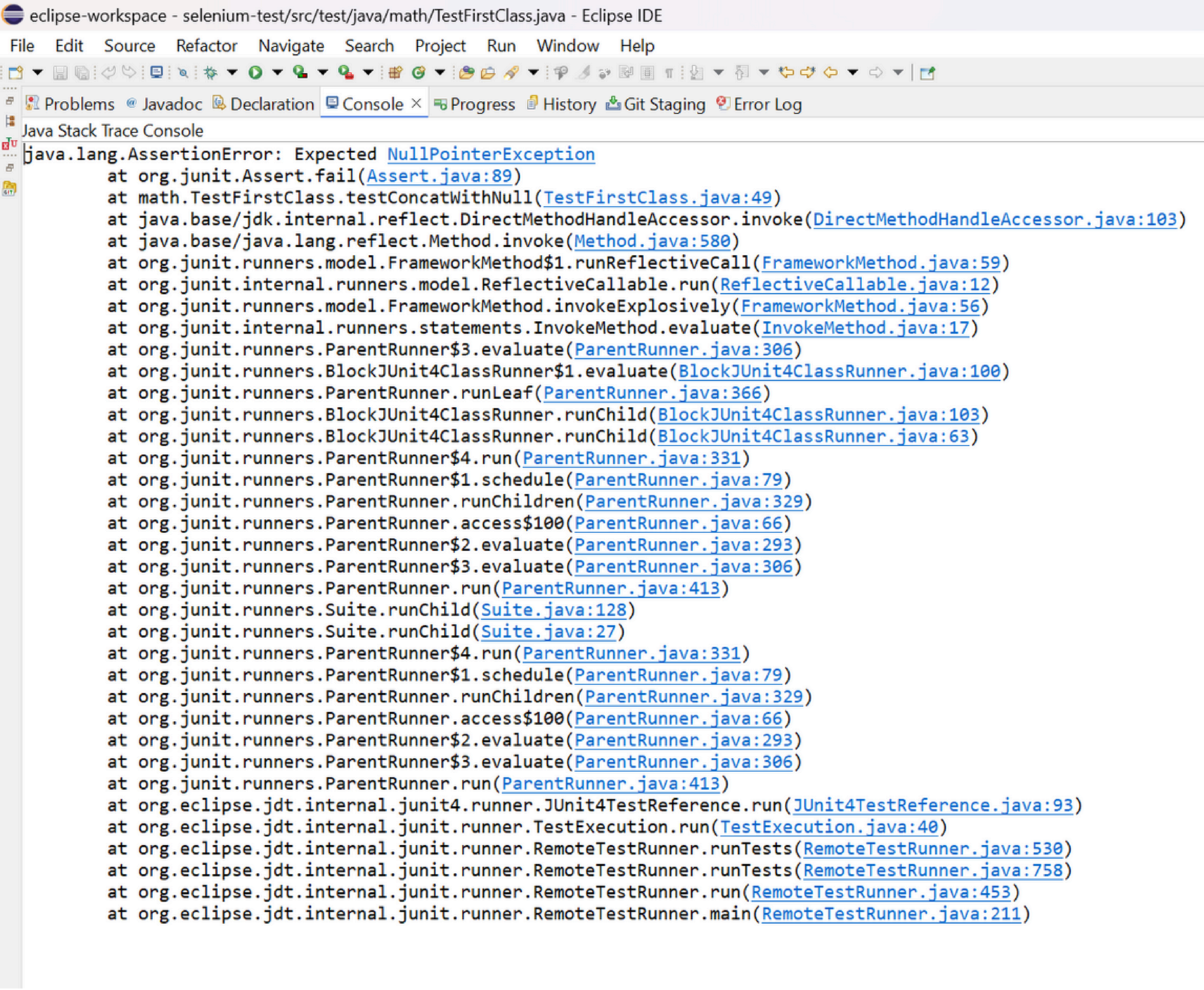
Cause: Java doesn't throw error on overflow

concat() Method:

- Passed for valid and empty strings
- Broke with null input:
- concat(null, "World") → Throws NullPointerException

Cause: No null handling in source code

Function	Type of Test	Breaking Point	Real-World Risk
add()	Boundary Testing	Integer Overflow	Financial errors, data loss
concat()	Robustness Testing	Null input	App crashes, form failures


eclipse-workspace - selenium-test/src/test/java/math/TestFirstClass.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems
Javadoc
Declaration
Console ×
Progress
History
Git Staging
Error Log
Java Stack Trace Console

```
java.lang.AssertionError: Expected NullPointerException
	at org.junit.Assert.fail(Assert.java:89)
	at math.TestFirstClass.testConcatWithNull(TestFirstClass.java:49)
	at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:103)
	at java.base/java.lang.reflect.Method.invoke(Method.java:580)
	at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
	at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
	at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:56)
	at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
	at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
	at org.junit.runners.BlockJUnit4ClassRunner$1.evaluate(BlockJUnit4ClassRunner.java:100)
	at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:366)
	at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:103)
	at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:63)
	at org.junit.runners.ParentRunner$4.run(ParentRunner.java:331)
	at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:79)
	at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:329)
	at org.junit.runners.ParentRunner.access$100(ParentRunner.java:66)
	at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:293)
	at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
	at org.junit.runners.ParentRunner.run(ParentRunner.java:413)
	at org.junit.runners.Suite.runChild(Suite.java:128)
	at org.junit.runners.Suite.runChild(Suite.java:27)
	at org.junit.runners.ParentRunner$4.run(ParentRunner.java:331)
	at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:79)
	at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:329)
	at org.junit.runners.ParentRunner.access$100(ParentRunner.java:66)
	at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:293)
	at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
	at org.junit.runners.ParentRunner.run(ParentRunner.java:413)
	at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:93)
	at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:40)
	at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:530)
	at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:758)
	at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:453)
	at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:211)
```

Task 2 - ChatGPT-Generated Tests

Prompt Given to ChatGPT:

Write JUnit test cases for the following code:

```
public class Calculator {
    public int multiply(int a, int b) {
        return a * b;
    }
}
```

ChatGPT Output Included:

```
@Test
public void testMultiplyPositiveNumbers() {
    Assert.assertEquals(6, calculator.multiply(2, 3));
}

@Test
public void testMultiplyWithZero() {
    Assert.assertEquals(0, calculator.multiply(0, 5));
}

@Test
public void testMultiplyWithNegative() {
    Assert.assertEquals(-6, calculator.multiply(-2, 3));
}
```

Observations:

- Used @Test annotation and meaningful method names
- Covered typical inputs: positive, zero, negative
- Did not include:
- Exception testing
- Test suites
- Overflow or boundary test cases

Comparison with Task 1 Tests:

Aspect	ChatGPT-Generated	Manually Written
Positive/Basic Inputs	Covered	Covered
Edge Cases / Overflow	Not covered	Covered
Exception Handling	Missing	Included (null case)
Test Suite Usage	Not included	Implemented (AllTests)

Reflection

Key Learnings This Week:

- Understood and applied Boundary Value Analysis (BVA) and Equivalence Class Testing (ECT).
- Learned how to write effective unit tests using JUnit, including:
- Valid/invalid input testing
- Exception handling
- Use of test suites
- Gained hands-on experience identifying breaking points in code using BVA & ECT.
- Used ChatGPT to assist in generating test cases and compared them with manually crafted tests.

Practical Application:

These testing techniques are essential for:

- Energy systems (e.g., sensor calibration limits)
- Finance systems (e.g., preventing overflow)
- Web apps (e.g., input validation, preventing crashes)

thank you