JUNIT

Exercise 1: Setting Up JUnit

Scenario: You need to set up JUnit in your Java project to start writing unit tests.

Steps:

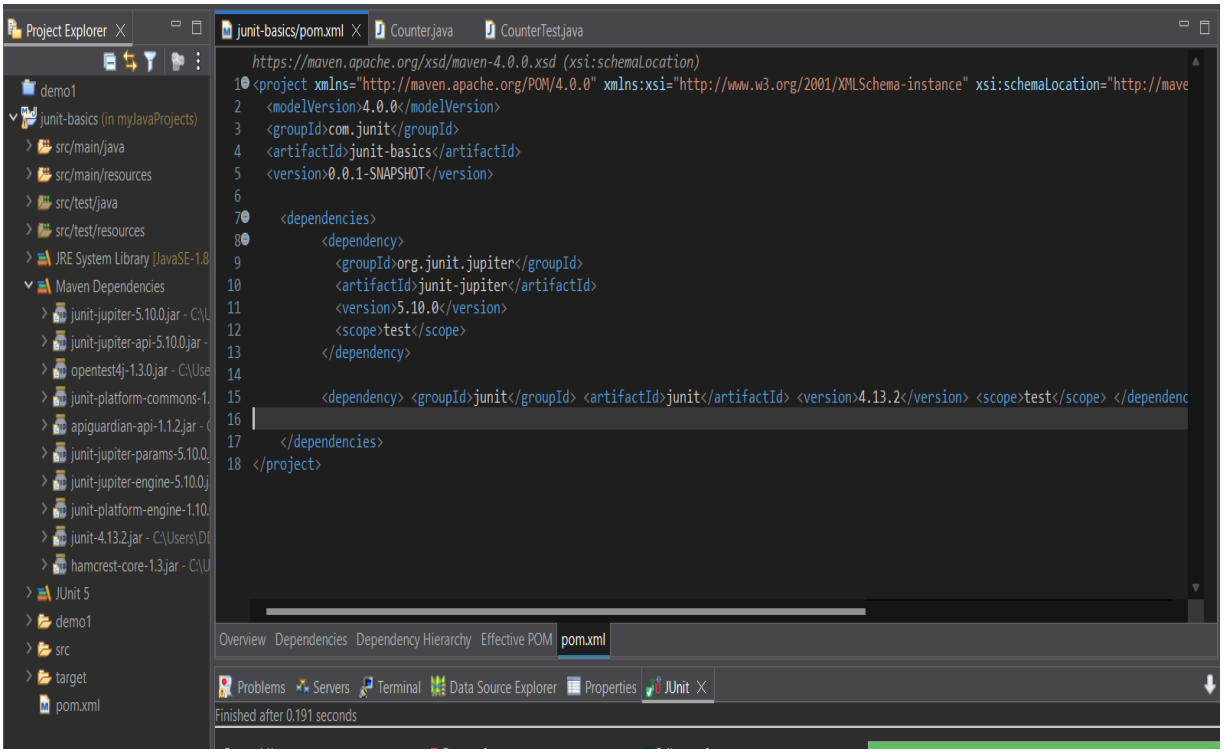
1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml: <dependency> <groupId>junit</groupId> <artifactId>junit</artifactId> <version>4.13.2</version> <scope>test</scope> </dependency>

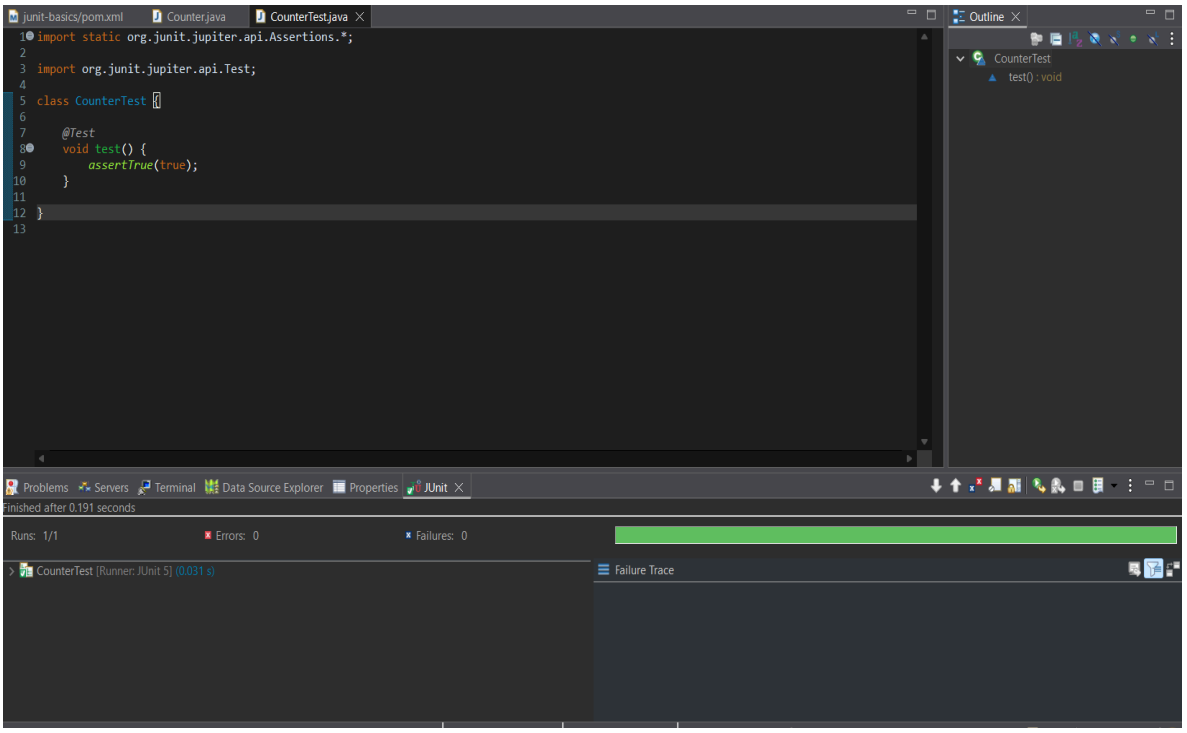
3. Create a new test class in your project.

Output Screenshots

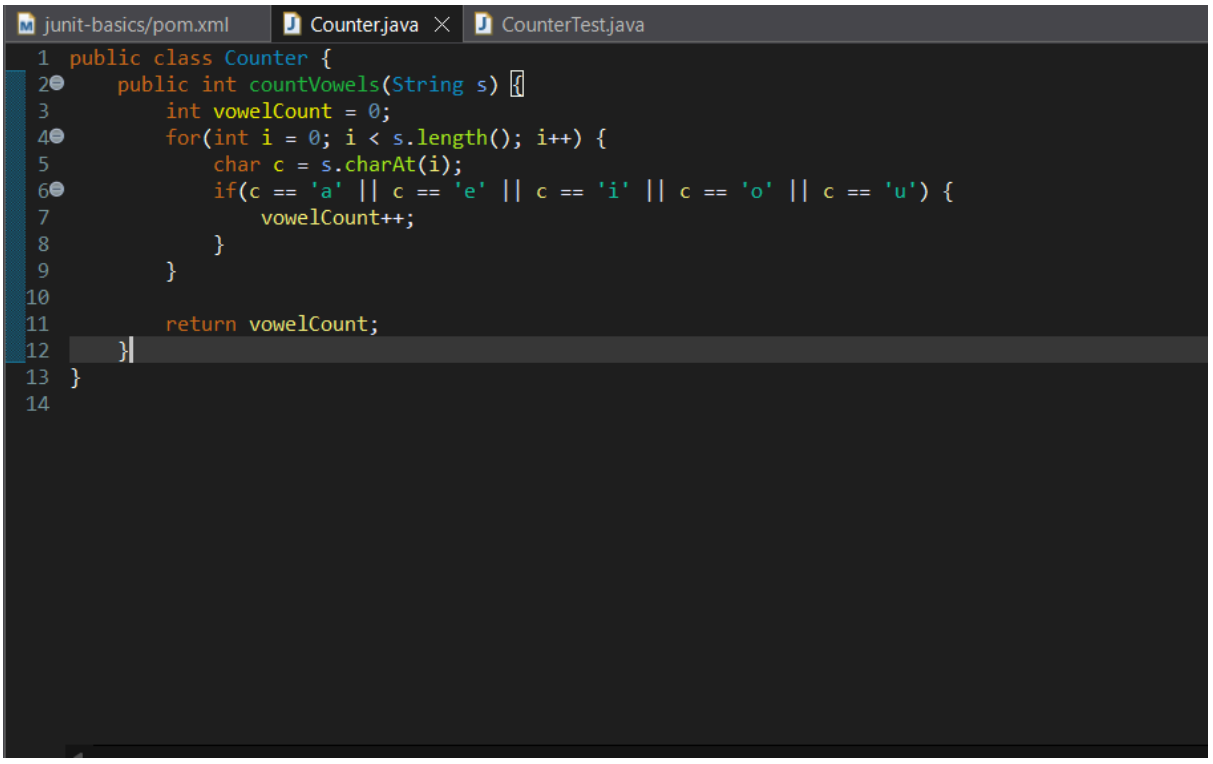
pom.xml



Test class



Created a counter class and a method for counting vowels in a string



Exercise 2: Writing Basic JUnit Tests

Scenario: You need to write basic JUnit tests for a simple Java class.

Steps:

1. Create a new Java class with some methods to test.

2. Write JUnit tests for these methods.

Solution

- Created a Counter class that has methods like countVowels(string) and countConsonants(string)
- Ran the tests and obtained a failure
- Corrected the countConsonants code to check whether the character is part of the English alphabet.
- Re-ran the tests and all tests have passed

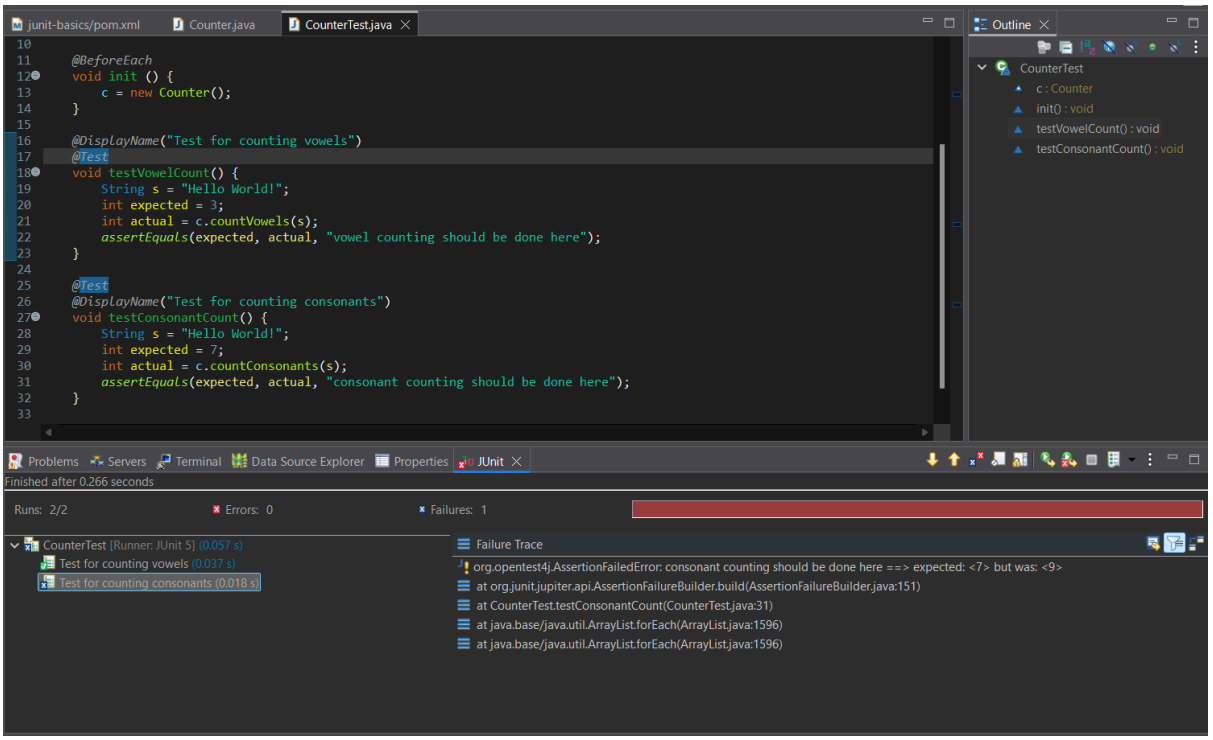
Counter.java

```
public class Counter {
	public int countVowels(String s) {
		int vowelCount = 0;
		for(int i = 0; i < s.length(); i++) {
			char c = s.charAt(i);
			if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
{
				vowelCount++;
			}
		}

		return vowelCount;
	}

	public int countConsonants(String s) {
		int consonantsCount = 0;
		for(int i = 0; i < s.length(); i++) {
			char c = s.charAt(i);
			if(Character.isAlphabetic(c) && c != 'a' && c != 'e' && c !=
'i' && c != 'o' && c != 'u') {
				consonantsCount++;
			}
		}

		return consonantsCount;
	}
}
```

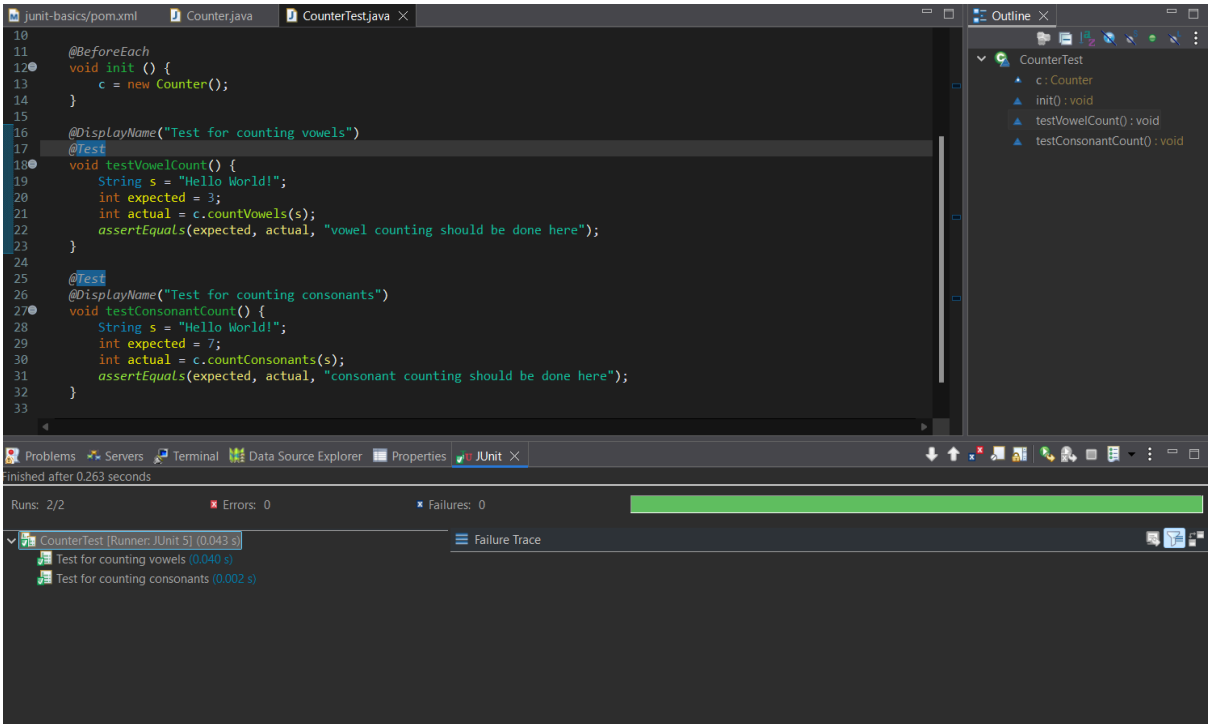
CounterTest.java

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
class CounterTest {

	Counter c;

	@BeforeEach
	void init () {
		c = new Counter();
	}

	@DisplayName("Test for counting vowels")
	@Test
	void testVowelCount() {
		String s = "Hello World!";
		int expected = 3;
		int actual = c.countVowels(s);
		assertEquals(expected, actual, "vowel counting should be done
here");
	}

	@Test
	@DisplayName("Test for counting consonants")
	void testConsonantCount() {
		String s = "Hello World!";
		int expected = 7;
		int actual = c.countConsonants(s);
		assertEquals(expected, actual, "consonant counting should be done
here");
	}
}
```

Incorrect Result



Correct Result



Exercise 3: Assertions in JUnit

Scenario: You need to use different assertions in JUnit to validate your test results.

Steps:

1. Write tests using various JUnit assertions.

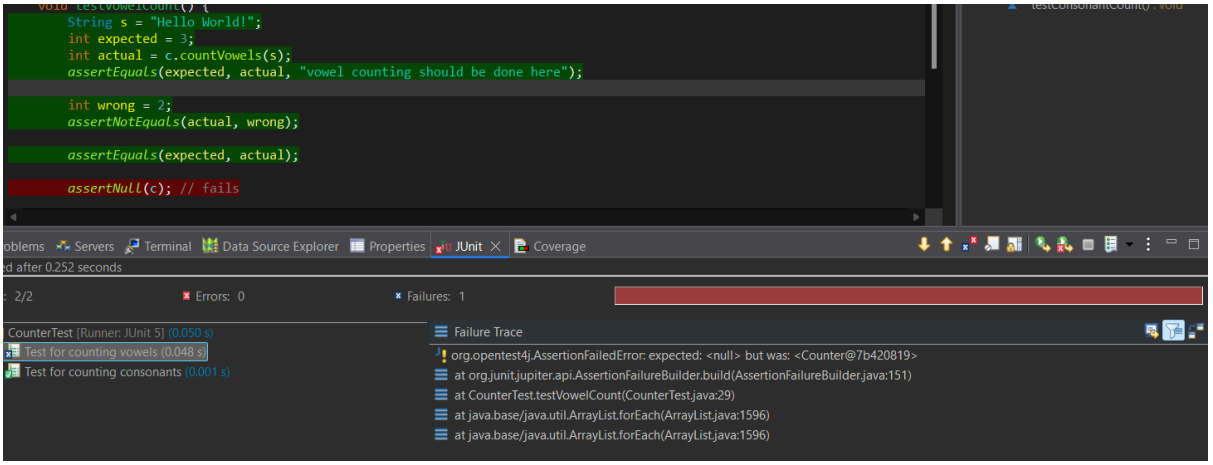
Solution

`testVowelCount()` method

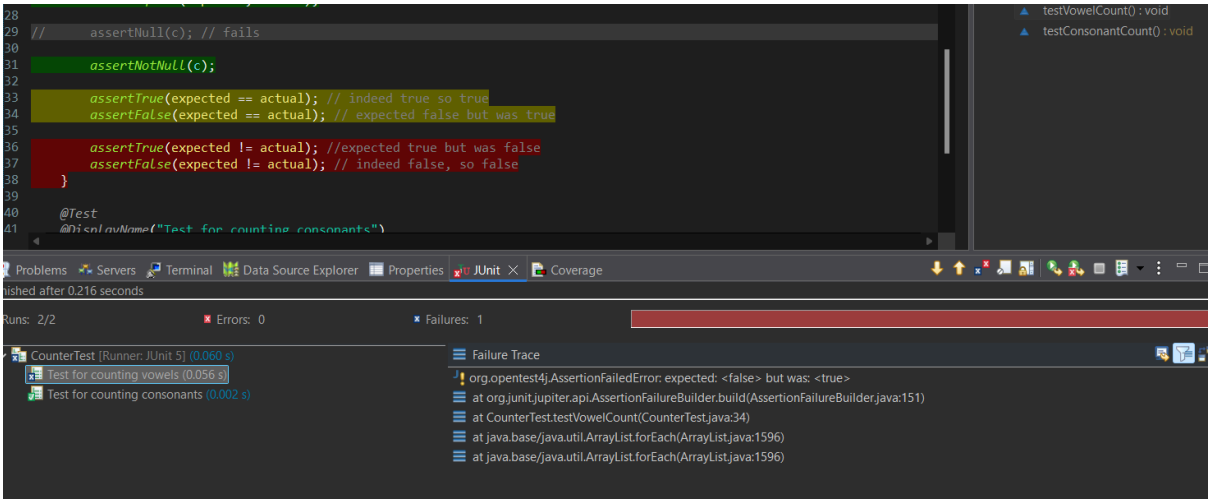
```
@DisplayName("Test for counting vowels")
	@Test
	void testVowelCount() {
		String s = "Hello World!";
		int expected = 3;
		int actual = c.countVowels(s);
		assertEquals(expected, actual, "vowel counting should be done
here");

		int wrong = 2;
		assertNotEquals(actual, wrong);

		assertEquals(expected, actual);

		assertNull(c); // fails

		assertNotNull(c);

		assertTrue(expected == actual); // indeed true so true
		assertFalse(expected == actual); // expected false but was true

		assertTrue(expected != actual); //expected true but was false
		assertFalse(expected != actual); // indeed false, so false
	}
```

Output Screenshots

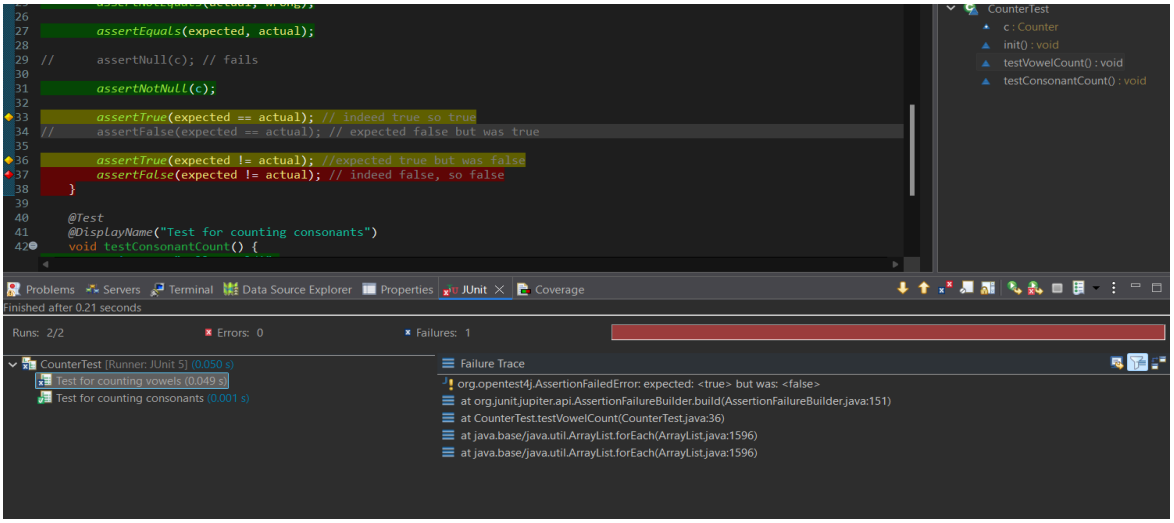
assertNull(c);



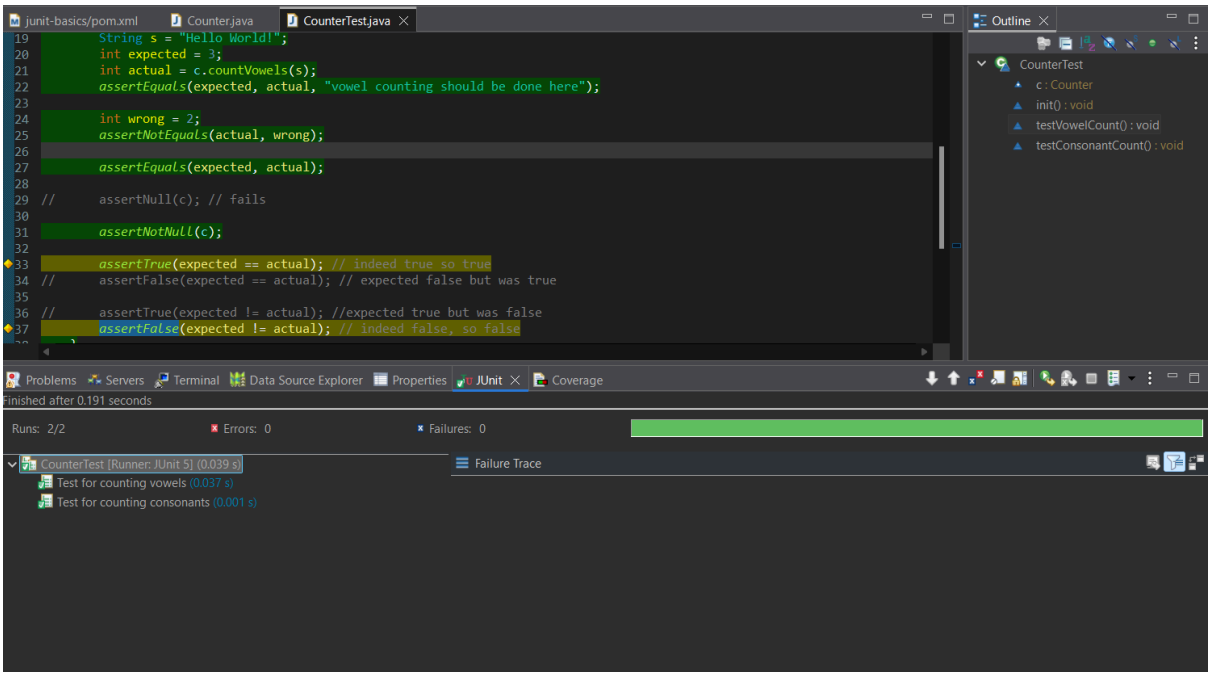
assertFalse(expected == actual);



assertTrue(expected != actual);



Success


junit-basics/pom.xml
Counter.java
CounterTest.java
String s = "Hello World!";
int expected = 3;
int actual = c.countVowels(s);
assertEquals(expected, actual, "vowel counting should be done here");
int wrong = 2;
assertNotEquals(actual, wrong);
assertEquals(expected, actual);
// assertNull(c); // fails
assertNotNull(c);
assertTrue(expected == actual); // indeed true so true
// assertFalse(expected == actual); // expected false but was true
// assertTrue(expected != actual); //expected true but was false
assertFalse(expected != actual); // indeed false, so false
Outline
CounterTest
c : Counter
init() : void
testVowelCount() : void
testConsonantCount() : void
Problems
Servers
Terminal
Data Source Explorer
Properties
JUnit
Coverage
Finished after 0.191 seconds
Runs: 2/2
Errors: 0
Failures: 0
CounterTest [Runner: JUnit 5] (0.039 s)
Test for counting vowels (0.037 s)
Test for counting consonants (0.001 s)
Failure Trace

Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods.

Solution

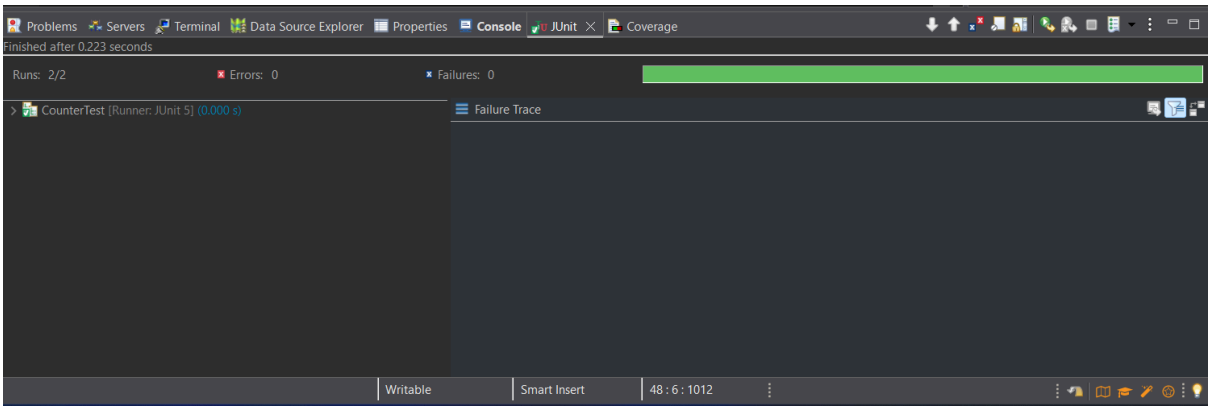
CounterTest.java

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
class CounterTest {

	Counter c;

	@BeforeEach
	void init () {
		c = new Counter();
	}

	@AfterEach
	void cleanUp() {
		System.out.println("well done! cleaning up..!");
	}


	@DisplayName("Test for counting vowels")
	@Test
	void testVowelCount() {
		// Arrange
		String s = "Hello World!";
		int expected = 3;
		int wrong = 2;

		// Act
		int actual = c.countVowels(s);

		// Assert
		assertEquals(expected, actual, "vowel counting should be done
```

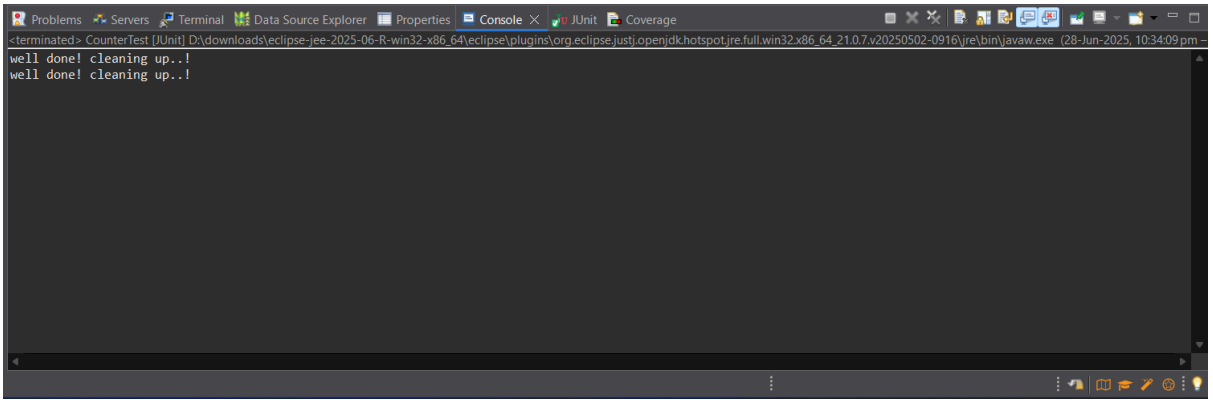
```
here");
                assertNotEquals(actual, wrong);
        }

        @Test
        @DisplayName("Test for counting consonants")
        void testConsonantCount() {
                String s = "Hello World!";
                int expected = 7;

                int actual = c.countConsonants(s);

                assertEquals(expected, actual, "consonant counting should be done
here");
        }
}
```

Output Screenshots

Successful Tests



Clean up console statements



MOCKITO

Exercise 1: Mocking and Stubbing

Scenario: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

Solution

Exercise 2: Verifying Interactions

Scenario: You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

Solution

SLF4J

Exercise 1: Logging Error Messages and Warning Levels

Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Solution