Assignment_9

Problem Statement 1: JUnit 5

a. Create a unit test using JUnit 5 to check if the length of the string matches the given length.

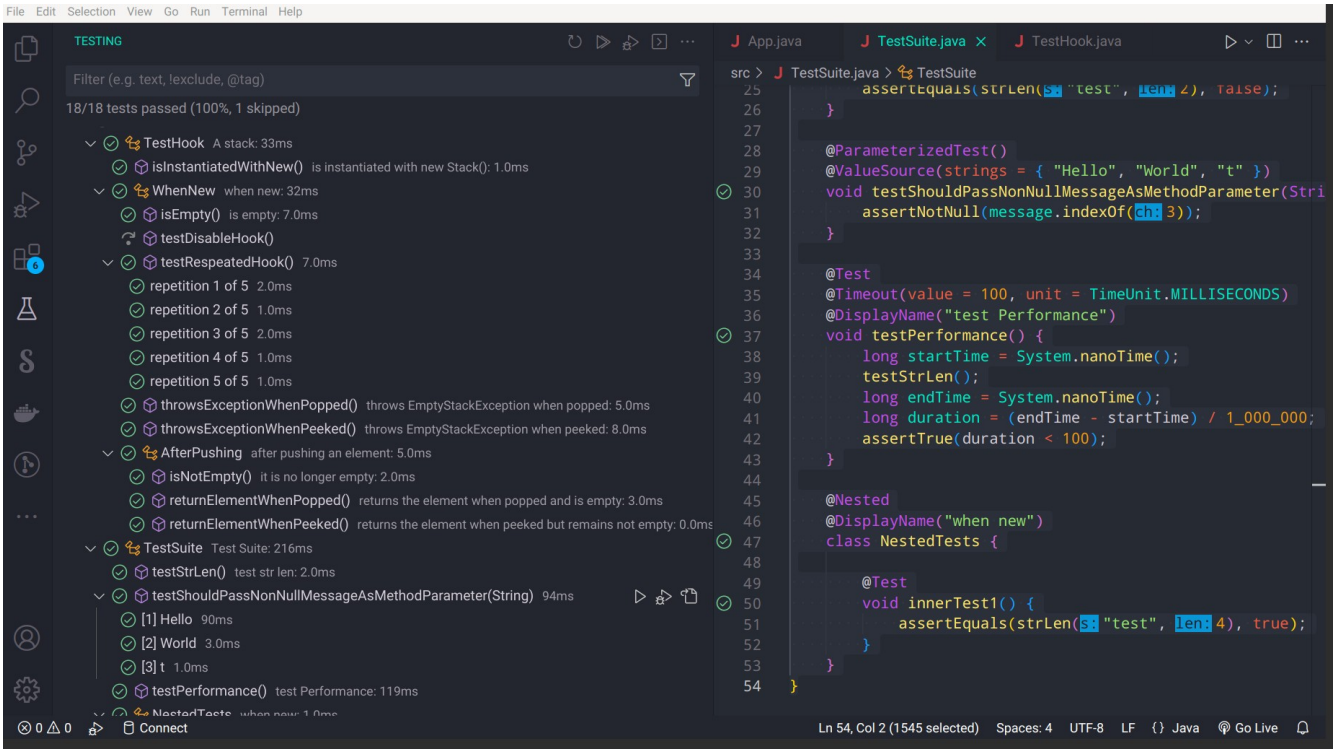
b. Create a unit test using Junit 5 to test null pointer exception. Create test using @ParameterizedTest annotation.

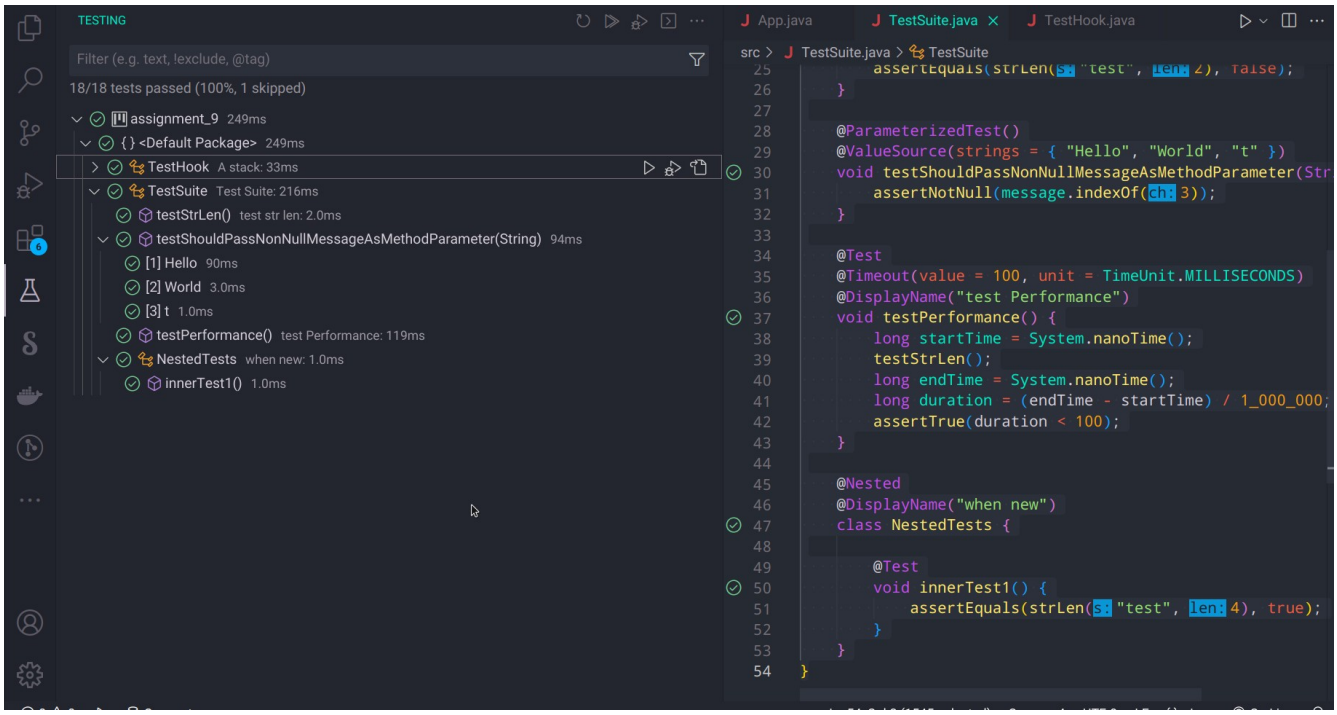
c. Create a unit test using JUnit 5 to check the performance of code.

d. Create some tests and group the tests using @Nested annotation.

e. Create unit tests using the JUnit 5 annotations @DisplayName, @BeforeEach, @AfterEach and @Disable. Create a test and use @RepeatedTest annotation.

Screenshot





a) Code

PART_1 Test Hook.java

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.EmptyStackException;
import java.util.Stack;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.Test;

@DisplayName("A stack")
class TestHook {

    Stack<Object> stack;

    @Test
    @DisplayName("is instantiated with new Stack()")
    void isInstantiatedWithNew() {
        new Stack<>();
    }

    @Nested
    @DisplayName("when new")
```

```
class WhenNew {

    @BeforeEach
    void createNewStack() {
        stack = new Stack<>();
    }

    @AfterEach
    void tearDown() {
        // code to be executed after each test
    }

    @Test
    @DisplayName("is empty")
    void isEmpty() {
        assertTrue(stack.isEmpty());
    }

    @Test
    @Disabled("This test is currently not implemented")
    void testDisableHook() {
        // test code
    }

    @RepeatedTest(5)
    void testRespeatedHook() {
        // test code
    }

    @Test
    @DisplayName("throws EmptyStackException when popped")
    void throwsExceptionWhenPopped() {
        assertThrows(EmptyStackException.class, stack::pop);
    }

    @Test
    @DisplayName("throws EmptyStackException when peeked")
    void throwsExceptionWhenPeeked() {
        assertThrows(EmptyStackException.class, stack::peek);
    }

    @Nested
    @DisplayName("after pushing an element")
    class AfterPushing {

        String anElement = "an element";

        @BeforeEach
        void pushAnElement() {
```

```
            stack.push(anElement);
        }

        @Test
        @DisplayName("it is no longer empty")
        void isNotEmpty() {
            assertFalse(stack.isEmpty());
        }

        @Test
        @DisplayName("returns the element when popped and is empty")
        void returnElementWhenPopped() {
            assertEquals(anElement, stack.pop());
            assertTrue(stack.isEmpty());
        }

        @Test
        @DisplayName("returns the element when peeked but remains not empty")
        void returnElementWhenPeeked() {
            assertEquals(anElement, stack.peek());
            assertFalse(stack.isEmpty());
        }
    }
  }
}
```

PART 2 TestSuit.java

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.concurrent.TimeUnit;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

@DisplayName("Test Suite")
class TestSuite {
  // strLen String,Number -> Bool
  public static boolean strLen(String s, int len) {
    return s.length() == len;
  }

  @Test
  @DisplayName("test str len")
  void testStrLen() {
```

```
        assertEquals(strLen("test", 4), true);
        assertEquals(strLen("test", 2), false);
    }

    @ParameterizedTest()
    @ValueSource(strings = { "Hello", "World", "t" })
    void testShouldPassNonNullMessageAsMethodParameter(String message) {
        assertNotNull(message.indexOf(3));
    }

    @Test
    @Timeout(value = 100, unit = TimeUnit.MILLISECONDS)
    @DisplayName("test Performance")
    void testPerformance() {
        long startTime = System.nanoTime();
        testStrLen();
        long endTime = System.nanoTime();
        long duration = (endTime - startTime) / 1_000_000;
        assertTrue(duration < 100);
    }

    @Nested
    @DisplayName("when new")
    class NestedTests {

        @Test
        void innerTest1() {
            assertEquals(strLen("test", 4), true);
        }
    }
}
```