Name: Vaishali Kale

Email id: kalevaishalir16@gmail.com

Day 22: Junit Assignments

Task 1: Write a set of JUnit tests for a given class with simple mathematical operations (add, subtract, multiply, divide) using the basic @Test annotation.

Solution:::

CODE::::

CalculatorTest code:::

```
package testwithjunit;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.fail;

import org.junit.*;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

public class CalculatorTest {

   @Test
   public void testAdd() {
      Calculator calculator = new Calculator();
      assertEquals(5, calculator.add(2, 3));
      assertEquals(10, calculator.add(5, 5));
      assertEquals(-1, calculator.add(2, -3));
   }

   @Test
   public void testSubtract() {
      Calculator calculator = new Calculator();
      assertEquals(1, calculator.subtract(3, 2));
      assertEquals(0, calculator.subtract(5, 5));
      assertEquals(5, calculator.subtract(2, -3));
```

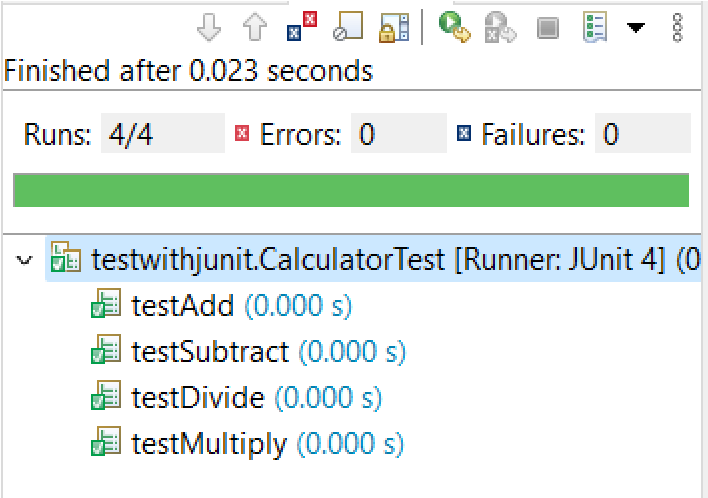
```
    }

    @Test
    public void testMultiply() {
        Calculator calculator = new Calculator();
        assertEquals(6, calculator.multiply(2, 3));
        assertEquals(25, calculator.multiply(5, 5));
        assertEquals(-6, calculator.multiply(2, -3));
    }

    @Test
    public void testDivide() {
        Calculator calculator = new Calculator();
        assertEquals(2, calculator.divide(6, 3));
        assertEquals(1, calculator.divide(5, 5));
        assertEquals(-2, calculator.divide(-6, 3));
        // Test division by zero
        try {
            calculator.divide(6, 0);
            fail("Expected an ArithmeticException to be thrown");
        } catch (ArithmeticException e) {
            // Test passed
        }
    }
}
```

Calculator class:

```
package testwithjunit;
public class Calculator {

    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

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

    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero");
        }
```

```
        return a / b;
    }
}
```

OUTPUT::::



Task 2: Extend the above JUnit tests to use @Before, @After, @BeforeClass, and @AfterClass annotations to manage test setup and teardown.

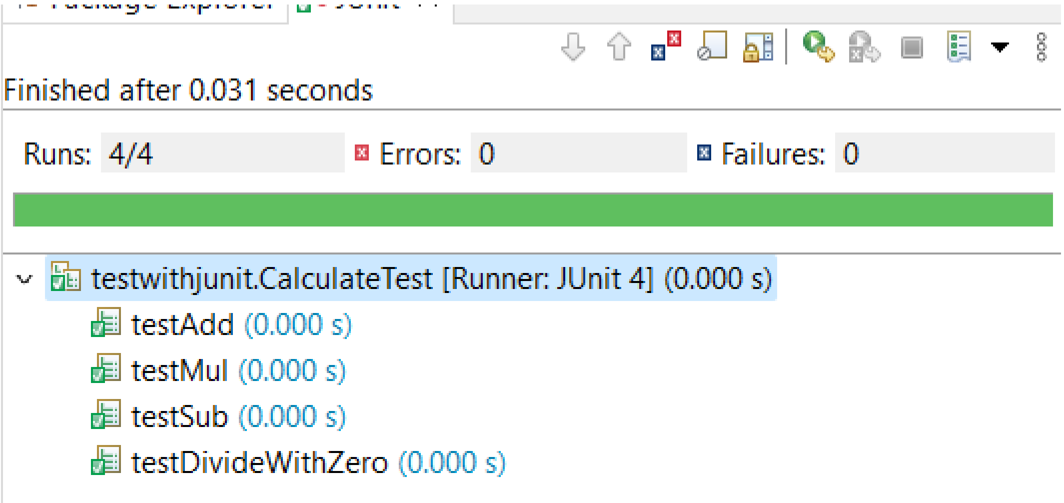
Solution:::

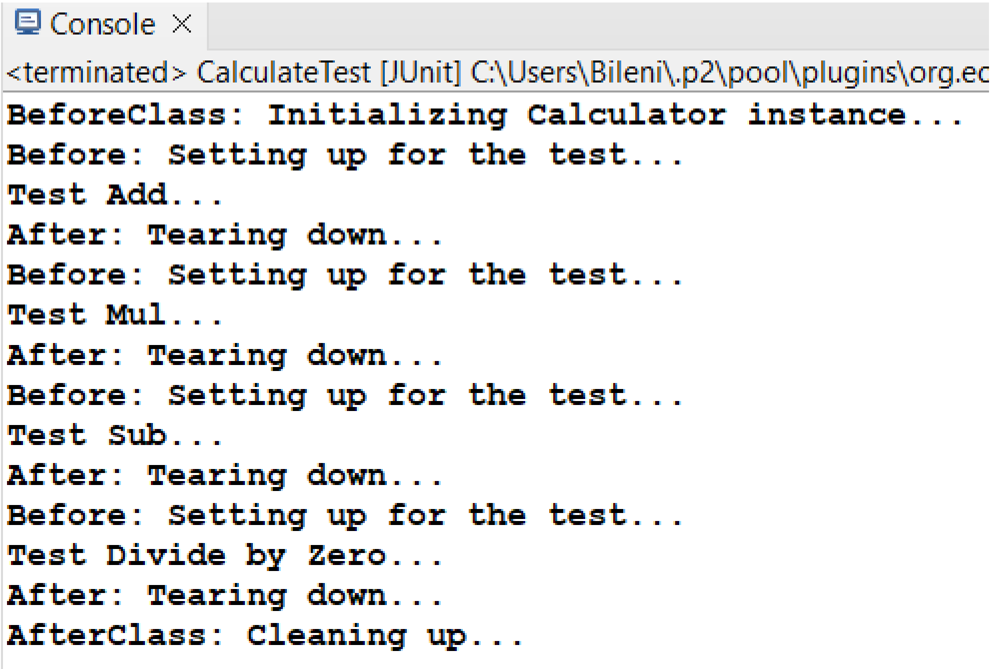
CODE::::

```
package testwithjunit;
import static org.junit.Assert.assertEquals;

import org.junit.*;
import org.junit.rules.ExpectedException;

public class CalculateTest {

    private static Calculator cal;

    @Rule
    public ExpectedException ex = ExpectedException.none();

    @BeforeClass
    public static void setUpClass() {
        System.out.println("BeforeClass: Initializing Calculator instance...");
        cal = new Calculator();
    }
```

```java
    @Before
    public void setUp() {
        System.out.println("Before: Setting up for the test...");
    }

    @Test
    public void testAdd() {
        System.out.println("Test Add...");
        assertEquals(20, cal.add(10, 10));
    }

    @Test
    public void testSub() {
        System.out.println("Test Sub...");
        assertEquals(12, cal.subtract(18, 6));
    }

    @Test
    public void testMul() {
        System.out.println("Test Mul...");
        assertEquals(24, cal.multiply(12, 2));
    }

    @Test
    public void testDivideWithZero() {
        System.out.println("Test Divide by Zero...");
        ex.expect(ArithmeticException.class);
        cal.divide(5, 0);
    }

    @After
    public void tearDown() {
        System.out.println("After: Tearing down...");
    }

    @AfterClass
    public static void tearDownClass() {
        System.out.println("AfterClass: Cleaning up...");
        cal = null;
    }
}
```

OUTPUT::::





Task 3: Create test cases with assertEquals, assertTrue, and assertFalse to validate the correctness of a custom String utility class.

Solution:::

CODE::::

Java Class::::

```
package testwithjunit;

public class StringUtil {

    // Method to check if a string is a palindrome
```

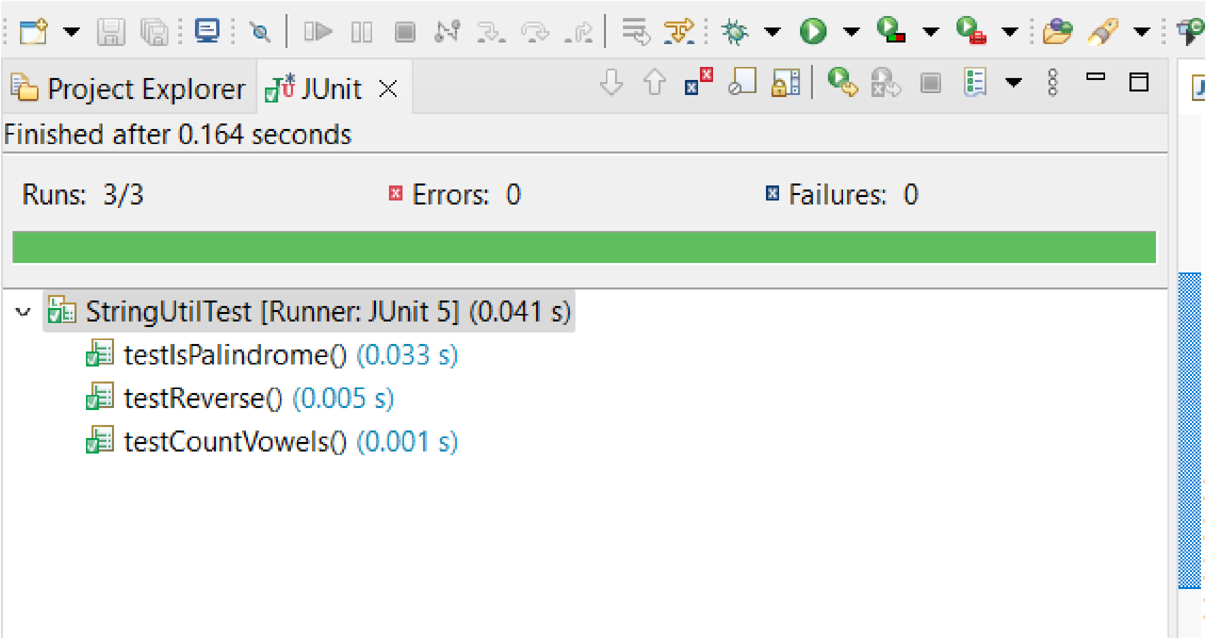
```java
    public static boolean isPalindrome(String str) {
        if (str == null) {
            return false;
        }
        String reversed = new StringBuilder(str).reverse().toString();
        return str.equals(reversed);
    }

    // Method to count the number of vowels in a string
    public static int countVowels(String str) {
        if (str == null) {
            return 0;
        }
        int count = 0;
        for (char c : str.toLowerCase().toCharArray()) {
            if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
                count++;
            }
        }
        return count;
    }

    // Method to reverse a string
    public static String reverse(String str) {
        if (str == null) {
            return null;
        }
        return new StringBuilder(str).reverse().toString();
    }
}
```

Test Class::--

```java
package testwithjunit;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;


public class StringUtilTest {


    @Test

    public void testIsPalindrome() {
```

```java
        assertTrue(StringUtil.isPalindrome("madam"));
        assertTrue(StringUtil.isPalindrome("racecar"));
        assertFalse(StringUtil.isPalindrome(null));
    }

    @Test
    public void testCountVowels() {
        assertEquals(5, StringUtil.countVowels("education"));
        assertEquals(2, StringUtil.countVowels("hello"));
        assertEquals(0, StringUtil.countVowels("bcd"));
        assertEquals(0, StringUtil.countVowels(null));
    }

    @Test
    public void testReverse() {
        assertEquals("olleh", StringUtil.reverse("hello"));
        assertEquals("avaJ", StringUtil.reverse("Java"));
        assertEquals("", StringUtil.reverse(""));
        assertEquals(null, StringUtil.reverse(null));
    }
}
```

OUTPUT::::



Task 4: Research and present a comparison of different garbage collection algorithms (Serial, Parallel, CMS, G1, ZGC) in Java.

Solution:::

1. Serial Garbage Collector

Overview:

- The Serial GC is a simple, single-threaded garbage collector.
- It is suitable for single-threaded environments and small applications where simplicity is preferred over throughput and low pause times.

Characteristics:

- Stop-the-world pauses: Yes, during both minor and major collections.
- Throughput: Low, due to single-threaded execution.
- Latency: High, because the application is paused during GC.
- Use Case: Small applications, single-threaded environments, and applications where pause times are acceptable.

2. Parallel Garbage Collector (Parallel GC)

Overview:

- The Parallel GC, also known as the throughput collector, uses multiple threads to speed up garbage collection.
- Aimed at maximizing application throughput by utilizing all available CPU cores.

Characteristics:

- Stop-the-world pauses: Yes, but multiple threads reduce the overall pause time.
- Throughput: High, as it uses multiple threads.
- Latency: Lower than Serial GC but still noticeable.
- Use Case: Multi-threaded applications and environments where throughput is more critical than pause times.

3. Concurrent Mark-Sweep (CMS) Garbage Collector

Overview:

- The CMS GC aims to minimize pause times by performing most of the GC work concurrently with the application threads.
- It uses multiple threads for the mark-sweep phases.

Characteristics:

- Stop-the-world pauses: Yes, but shorter because most work is done concurrently.
- Throughput: Moderate to high, as it reduces pause times but may not be as efficient as Parallel GC.
- Latency: Low, designed to minimize pause times.
- Use Case: Applications requiring low-latency GC, such as web servers or interactive applications.

4. Garbage First (G1) Garbage Collector

Overview:

- The G1 GC is designed to balance between high throughput and low latency.
- It divides the heap into regions and prioritizes collecting regions with the most garbage first.

Characteristics:

- Stop-the-world pauses: Yes, but aims to keep them predictable and short.
- Throughput: High, with a focus on minimizing full GC events.
- Latency: Low, with the ability to set pause time goals.
- Use Case: Large applications requiring a balance of high throughput and low latency, such as large-scale enterprise applications.

5. Z Garbage Collector (ZGC)

Overview:

- ZGC is a scalable, low-latency garbage collector.
- It aims for very short pause times (typically under 10ms) and can handle heaps ranging from a few hundred megabytes to multiple terabytes.

Characteristics:

- Stop-the-world pauses: Minimal, typically under 10ms.
- Throughput: High, but with a strong emphasis on low pause times.
- Latency: Extremely low, designed for applications needing consistent and predictable response times.
- Use Case: Large applications with stringent latency requirements, such as real-time systems and large in-memory databases.

Applications

Comparison Summary

Feature	Serial GC	Parallel GC	CMS GC	G1 GC	ZGC
Threads	Single-threaded	Multi-threaded	Multi-threaded	Multi-threaded	Multi-threaded
Stop-the-world	High	Moderate	Low	Low	Very Low
Throughput	Low	High	Moderate to High	High	High
Latency	High	Moderate	Low	Low	Very Low
Use Case	Small, single-threaded	Multi-threaded, throughput-sensitive	Low-latency applications	Large, balanced applications	Large, low-latency applications

Conclusion

Choosing the right garbage collector depends on the specific requirements of your application, such as throughput, latency, and application size.

Serial GC is suitable for small, single-threaded applications.

Parallel GC is best for applications where throughput is critical.

CMS GC is ideal for applications needing low pause times.

G1 GC provides a balanced approach, suitable for large applications with mixed requirements.

ZGC is the go-to choice for applications with stringent low-latency needs and large heaps.

Selecting the appropriate garbage collector and tuning it correctly can significantly impact your application's performance and responsiveness.