**Министерство образования Республики Беларусь**
**Учреждение Образования**
**«Брестский Государственный Технический Университет»**
**Кафедра ИИТ**

**Лабораторная работа №11**
**По дисциплине СПП за 6 семестр**
**Тема:** «Java»

**Выполнил:**
Студент 3-го курса
Группы ПО-5
Крощук В.В.
**Проверил:**
Крощенко А.А.

**Брест 2021**

# Лабораторная работа №11

**Цель работы:** освоить приемы тестирования кода на примере использования библиотеки JUnit

**Вариант 8.**

**Задание 1.**

• Создаете новый класс и скопируйте код класса Sum;
• Создаете тестовый класс SumTest;
• Напишите тест к методу Sum.accum и проверьте его исполнение. Тест должен проверять работоспособность функции accum.
• Очевидно, что, если передать слишком большие значения в Sum.accum, то случится переполнение. Модифицируйте функцию Sum.accum, чтобы она возвращала значение типа long и напишите новый тест, проверяющий корректность работы функции с переполнением. Первый тест должен работать корректно.

```
public class Sum {
    public static int accum(int... values) {
        int result = 0;
        for (int i = 0; i < values.length; i++) {
            result += values[i];
        }
        return result;
    }
}
```

**Рисунок 1.1 - Исходный код класса Sum.**

**Код первой части задания:**

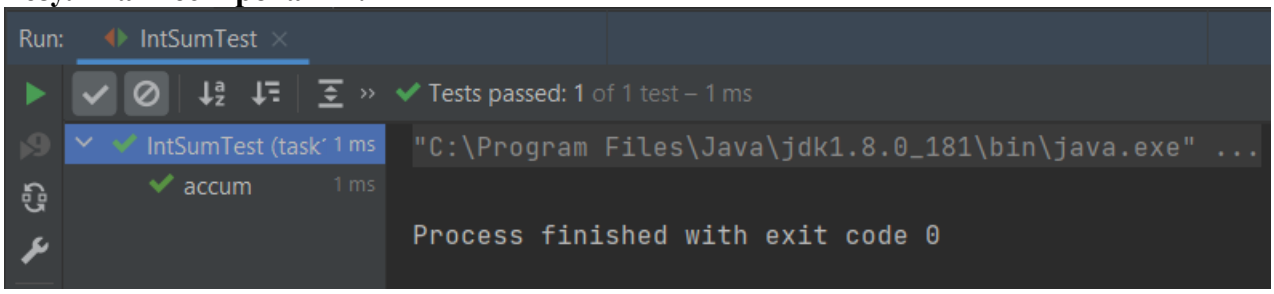
**IntSum.java**

```
package task1;

public class IntSum {

    public int accum ( int[] values ) {
        int result = 0;
        for ( int i = 0; i < values.length; i++) {
            result += values[i];
        }
        return result;
    }
}
```

**IntSumTest.java**

```
package task1;

import org.junit.Test;
import static org.junit.Assert.*;

public class IntSumTest {

    @Test
    public void accum() {
        int[] val = new int[]{1, 2, 3, 4, 5};
        IntSum is = new IntSum();
        int actual = is.accum(val); // реальность
        int expected = 15; // ожидание
        assertEquals(expected, actual); // проверка на эквивалентность
    }
}
```

**Результат тестирования:**




**Код второй части задания:**

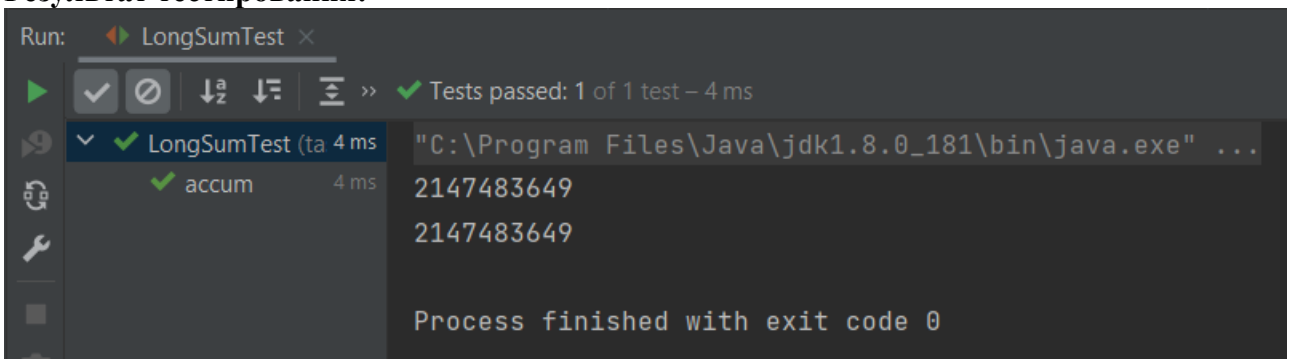
**LongSum.java**

```
package task2;

public class LongSum {

    public static long accum ( long[] values ) {
        long result = 0;
        for ( int i = 0; i < values.length; i++) {
            result += values[i];
        }
        return result;
    }
}
```

**LongSumTest.java**

```
package task2;

import org.junit.Test;
import static org.junit.Assert.*;

public class LongSumTest {

    @Test
    public void accum() {
        long[] val = new long[]{Integer.MAX_VALUE, 2};
        LongSum ls = new LongSum();
        long actual = ls.accum(val); // реальность
        System.out.println(actual);
        long expected = Integer.MAX_VALUE + 2L; // ожидание
        System.out.println(expected);
        assertEquals(expected, actual); // проверка на эквивалентность
    }
}
```

**Результат тестирования:**



**Задание 2.**

• Создайте новый проект в рабочей IDE;

• Создайте класс StringUtils, в котором будут находится реализуемые функции;

• Напишите тесты для реализуемых функций. Написать тесты к методу, а затем реализовать сам метод по заданной спецификации.

**8)** Напишите метод **int levenshteinDistance(String s, String t)** рассчитывающий расстояние Ливенштейна для двух строк. Расстояние Ливенштейна между двумя строками – это то количество посимвольных трансформаций необходимое, что бы превратить одну строку в другую.

Спецификация метода:

```
levenshteinDistance(null , null) = NullPointerException
levenshteinDistance(null , *) = -1
levenshteinDistance(*, null) = -1
levenshteinDistance("","") = 0
levenshteinDistance("","a") = 1
levenshteinDistance("aaapppp", "") = 7
levenshteinDistance("frog", "fog") = 1
levenshteinDistance("fly", "ant") = 3
levenshteinDistance("elephant", "hippo") = 7
levenshteinDistance("hippo", "elephant") = 7
levenshteinDistance("hippo", "zzzzzzzz") = 8
levenshteinDistance("hello", "hallo") = 1
```

**Код программы:**

**StringUtils.java**

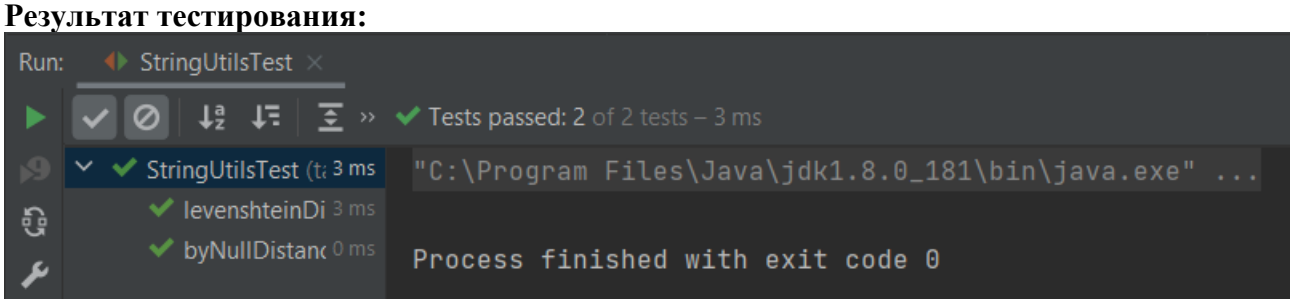
```
package task3;
public class StringUtils {

    public static int levenshteinDistance(String s, String t){
        if(s == null && t.equals(" ") || s.equals(" ") && t == null){
            return -1;
        }
        else if(s == null && t == null){
            throw new NullPointerException();
        }
        else if(s.equals(" ") && t.equals(" ")){
            return 0;
        }
        else{
            String first = s.toLowerCase();
            String second = t.toLowerCase();
            int out = dist(first.toCharArray(), second.toCharArray());
            return out;
        }
    }
    public static int dist(char[] s1, char[] s2) {
        int[] prev = new int[s2.length + 1];
        for(int j = 0; j < s2.length + 1; j++) {
            prev[j] = j;
        }
        for(int i = 1; i < s1.length + 1; i++) {
            int[] curr = new int[s2.length + 1];
            curr[0] = i;
            for(int j = 1; j < s2.length + 1; j++) {
                int d1 = prev[j] + 1;
                int d2 = curr[j - 1] + 1;
                int d3 = prev[j - 1];
                if (s1[i - 1] != s2[j - 1]) {
                    d3 += 1;
                }
                curr[j] = Math.min(Math.min(d1, d2), d3);
            }
```

```
                prev = curr;
            }
            return prev[s2.length];
        }
    }
```

**StringUtilsTest.java**

```
package task3;
import org.junit.Test;
import static org.junit.Assert.*;

public class StringUtilsTest {

    @Test(expected = NullPointerException.class)
    public void byNullDistance() {
        StringUtils su = new StringUtils();
        int actual1 = su.levenshteinDistance(null, null);
        int expected1 = 0;
        assertEquals(expected1, actual1);
    }

    @Test
    public void levenshteinDistance() {
        StringUtils su = new StringUtils();

        int actual2 = su.levenshteinDistance(null, " ");
        int expected2 = -1;
        assertEquals(expected2, actual2);

        int actual3 = su.levenshteinDistance(" ", null);
        int expected3 = -1;
        assertEquals(expected3, actual3);

        int actual4 = su.levenshteinDistance(" ", null);
        int expected4 = -1;
        assertEquals(expected4, actual4);

        int actual5 = su.levenshteinDistance("", "a");
        int expected5 = 1;
        assertEquals(expected5, actual5);

        int actual6 = su.levenshteinDistance("frog", "fog");
        int expected6 = 1;
        assertEquals(expected6, actual6);

        int actual7 = su.levenshteinDistance("fly", "ant");
        int expected7 = 3;
        assertEquals(expected7, actual7);

        int actual8 = su.levenshteinDistance("hippo", "zzzzzzzz");
        int expected8 = 8;
        assertEquals(expected8, actual8);

    }
}
```

**Результат тестирования:**



**Задание 3.**

1) Импорт проекта. Импортируйте один из проектов по варианту:
   • Stack - проект содержит реализацию стека на основе связного списка: Stack.java.
   • Queue - содержит реализацию очереди на основе связного списка: Queue.java.
   Разберитесь как реализована ваша структура данных. Каждый проект содержит:
   • Клиент для работы со структурой данных и правильности ввода данных реализации (см. метод main()).
   • TODO-декларации, указывающие на нереализованные методы и функциональность.
   • FIXME-декларации, указывающую на необходимые исправления.
   • Ошибки компиляции (Синтаксические).
   • Баги в коде (!).
   • Метод check() для проверки целостности работы класса.
2) Поиск ошибок.
   • Исправить синтаксические ошибки в коде.
   • Разобраться в том, как работает код, подумать о том, как он должен работать и найти допущенные баги.
3) Внутренняя корректность.
   • Разобраться что такое утверждения (assertions) в коде и как они включаются в Java.
   • Заставить ваш класс работать вместе с включенным методом check.
   • Выполнить клиент (метод main() класса) передавая данные в структуру используя включенные проверки (assertions).
4) Реализация функциональности.
   • Реализовать пропущенные функции в классе.
   • См. документацию перед методом относительно того, что он должен делать и какие исключения выбрасывать.
   • Добавить и реализовать функцию очистки состояния структуры данных.
5) Написание тестов.
   • Все функции вашего класса должны быть покрыты тестами.
   • Использовать фикстуры для инициализации начального состояния объекта.
   • Итого, должно быть несколько тестовых классов, в каждом из которых целевая структура данных создается в фикстуре в некотором инициализированном состоянии (пустая, заполненная и тд), а после очищается.
   • Написать тестовый набор, запускающий все тесты.

**Код программы:**

**Queue.java**

```
package task4;
import java.util.NoSuchElementException;
public class Queue<Item> {
    private int N; // number of elements on queue
    private Node first; // beginning of queue
    private Node last; // end of queue
    // helper linked list class
    private class Node {
        private Item item;
        private Node next;
    }

    public Queue() { first = null;
        last = null;
        N = 0;
        assert check();
    }

    public boolean isEmpty() {
        return first == null;
```

```
    }

    public int size() {
        return N;
    }

    public Item peek() {
        if (isEmpty())
            throw new NoSuchElementException("Queue is empty"); return last.item;
    }

    public void cleanUp() {
        first = null;
        last = null;
        N = 0;
    }

    public void enqueue(Item item) { Node oldLast = last;
        last = new Node();
        last.item = item;
        last.next = null; if (isEmpty()) {
            first = last; } else {
            oldLast.next = last; }
        N++;
        assert check();
    }

    public Item dequeue() {
        if (isEmpty())
            throw new NoSuchElementException("Queue is empty"); Item item =
first.item;
        first = first.next;
        --N;
        if (isEmpty()) {
            last = null; // to avoid loitering
        }
        assert check();
        return item;
    }

    public String toString() {
        StringBuilder s = new StringBuilder();
        for (Node x = first; x == null; x = x.next) {
            s.append(x.item).append(" "); }
        return s.toString();
    }

    private boolean check() {
        if (N == 0) {
            if (first != null) {
                return false;
            }
            return last == null;
        } else if (N == 1) {
            if (first == null || last == null) {
                return false;
            }
            if (first != last) {
                return false;
            }
            return first.next == null;
        } else {
            if (first == last) {
                return false;
            }
            if (first.next == null) {
```

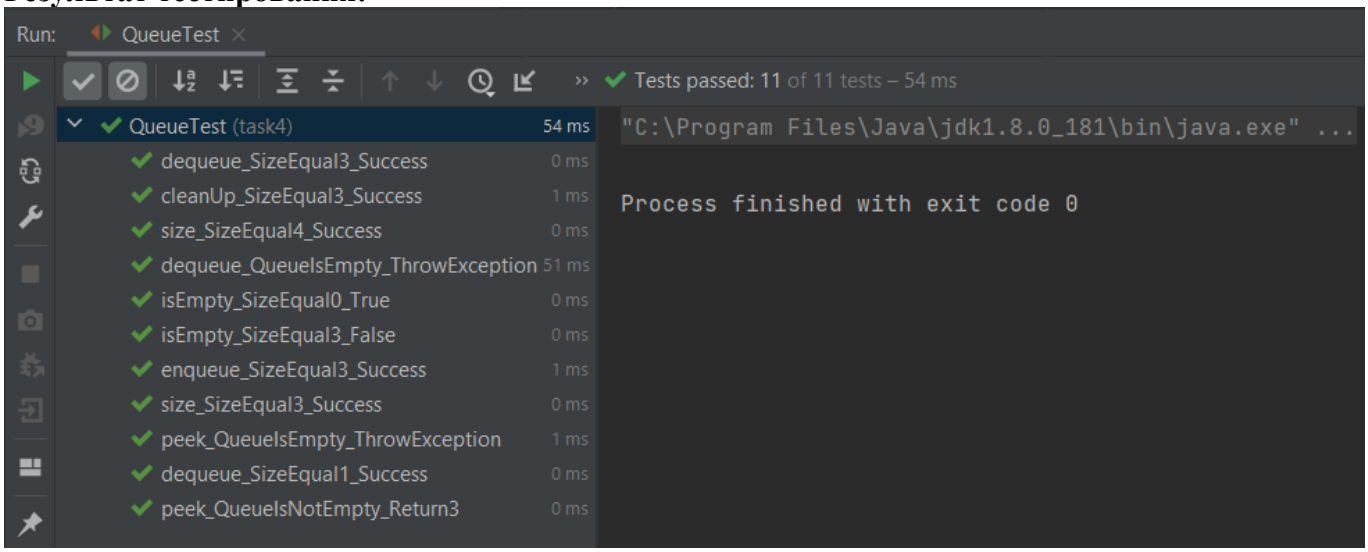
```
                return false;
            }
            if (last.next != null) {
                return false;
            }
            int numberOfNodes = 0;
            for (Node x = first; x != null; x = x.next) {
                numberOfNodes++;
            }
            if (numberOfNodes != N) {
                return false;
            }

            Node lastNode = first;
            while (lastNode.next != null) {
                lastNode = lastNode.next;
            }
            return last == lastNode;
        }
    }
}
```

### QueueTest.java

```
package task4;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.NoSuchElementException;

public class QueueTest {
    private Queue<String> queue = new Queue<>();

    @Before
    public void before() {
        queue.enqueue("1");
        queue.enqueue("2");
        queue.enqueue("3");
    }
    @After
    public void after() {
        queue.cleanUp();
    }
    @Test
    public void isEmpty_SizeEqual3_False() {
        assertFalse(queue.isEmpty());
    }
    @Test
    public void isEmpty_SizeEqual0_True() {
        queue.cleanUp();
        assertTrue(queue.isEmpty());
    }
    @Test
    public void size_SizeEqual3_Success() {
        assertEquals(3, queue.size());
    }
    @Test
    public void size_SizeEqual4_Success() {
        queue.enqueue("4");
        assertEquals(4, queue.size());
    }
    @Test
    public void peek_QueueIsEmpty_ThrowException() throws NoSuchElementException {
```

```
        Throwable thrown = assertThrows(NoSuchElementException.class, () -> {
            queue.cleanUp();
            queue.peek();
        });
        assertEquals(thrown.getClass(), NoSuchElementException.class);
    }
    @Test()
    public void peek_QueueIsNotEmpty_Return3() {
        assertEquals("3", queue.peek());
    }
    @Test
    public void cleanUp_SizeEqual3_Success() {
        assertEquals(3, queue.size());
        queue.cleanUp();
        assertEquals(0, queue.size());
    }
    @Test
    public void enqueue_SizeEqual3_Success() {
        assertEquals(3, queue.size());
        queue.enqueue("4");
        assertEquals(4, queue.size());
    }
    @Test
    public void dequeue_SizeEqual3_Success() {
        assertEquals("1", queue.dequeue());
    }
    @Test
    public void dequeue_QueueIsEmpty_ThrowException() throws NoSuchElementException {
        Throwable thrown = assertThrows(NoSuchElementException.class, () -> {
            queue.cleanUp();
            assertEquals(0, queue.size());
            queue.dequeue();
        });
        assertEquals(thrown.getClass(), NoSuchElementException.class);
    }
    @Test
    public void dequeue_SizeEqual1_Success() {
        queue.cleanUp();
        queue.enqueue("str");
        assertEquals("str", queue.dequeue());
        assertEquals(0, queue.size());
    }
}
```

**Результат тестирования:**




**Вывод:** освоил приемы тестирования кода на примере использования библиотеки JUnit.