Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «Современные платформы программирования»

Выполнила:
Студентка 3 курса
Группы ПО-6
Юсковец М.А.
Проверил:
Монтик Н.С.

Брест, 2023

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

Ход работы:

Задание:

Вариант 25

Задание 1 – Введение в JUnit

- 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;
   }
}
```

Задание 2 – Тестирование функций

Подготовка к выполнению:

- 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.

7) Напишите метод **String substringBetween(String str, String open, String close)** выделяющий подстроку относительно открывающей и закрывающей строки.

Спецификация метода:

```
substringBetween(null, null, null) = NullPointerException
substringBetween(null, *, *) = null
substringBetween(*, null, *) = null
substringBetween(*, *, null) = null
substringBetween("", "", "") = ""
substringBetween("", "", "]") = null
substringBetween("", "[", "]") = null
substringBetween("yabcz", "", "") = ""
substringBetween("yabcz", "y", "z") = "abc"
substringBetween("yabczyabcz", "y", "z") = "abc"
substringBetween("wx[b]yz", "[", "]") = "b"
```

Задание 3 – Поиск ошибок, отладка и тестирование классов

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.

Задание 1:

Текст программы:

Sum.java

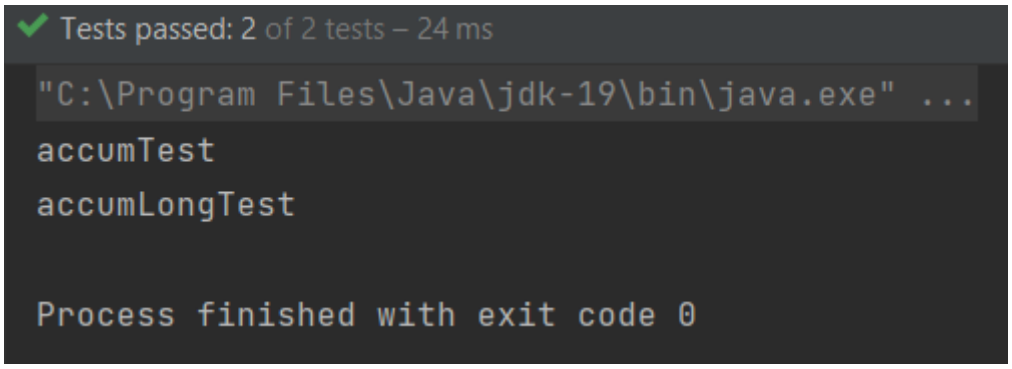
```
package com.example.spp_lab4;

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 ;
    }

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

SumTest.java

```
package com.example.spp_lab4;

import org.junit.jupiter.api.Test;

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

class SumTest {

    @Test
    public void accumTest() {
        int[] values = {5, 6, 3};
        int expecteds = 14;
        int result = Sum.accum(values);
        System.out.println("accumTest");
        assertEquals(expecteds, result);
    }


    @Test
```

```
    public void accumLongTest() {
        int[] values = {599999999, 699999999, 399999999};
        long expecteds = 1699999997L;
        long result = Sum.accum(values);
        System.out.println("accumLongTest");
        assertEquals(expecteds, result);
    }
}
```

Результат программы:



Задание 2:

Текст программы:

StringUtils.java

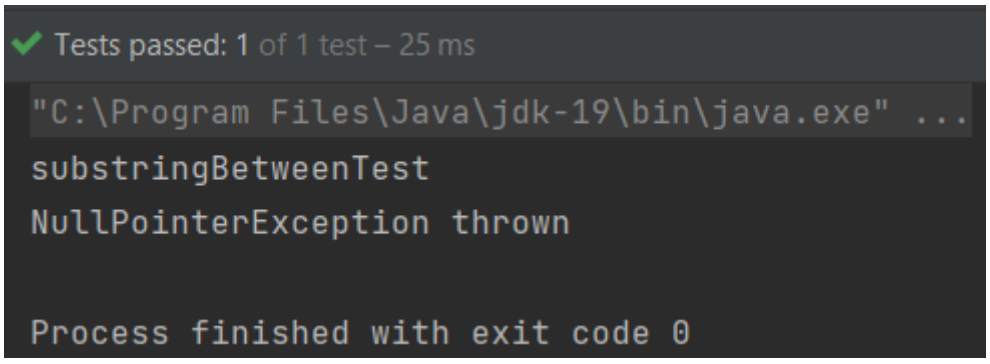
```
package com.example.spp_lab4;

import java.util.Objects;

public class StringUtils {
    public static String substringBetween(String str, String open, String
close) {
        String result = null;

        if (str == null && open == null && close == null)
            throw new NullPointerException();
        else if (str == null || open == null || close == null)
            return null;
        else if (!Objects.isNull(str) && open.equals("") && close.equals(""))
            return "";

        int openIndex = str.indexOf(open);
        int closeIndex = str.indexOf(close);

        if (openIndex < 0 || closeIndex < 0)
            return null;
        else if (closeIndex < openIndex)
            return null;
        else if (openIndex == closeIndex)
            return "";

        result = str.substring(openIndex + 1, closeIndex);

        return result;
```

```
    }
}
```

StringUtilsTest.java

```
package com.example.spp_lab4;

import org.junit.jupiter.api.Test;

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

class StringUtilsTest {

    @Test
    public void substringBetweenTest() throws NullPointerException {
        System.out.println("substringBetweenTest");

        assertNull(StringUtils.substringBetween(null , "*" , "*"));
        assertNull(StringUtils.substringBetween("*", null, "*"));
        assertNull(StringUtils.substringBetween("*", "*", null ));
        assertEquals(StringUtils.substringBetween("", "", ""), "");
        assertNull(StringUtils.substringBetween("", "", "]"));
        assertNull(StringUtils.substringBetween("", "[", "]"));
        assertEquals(StringUtils.substringBetween(" yabcz ", "", ""), "");
        assertEquals(StringUtils.substringBetween(" yabcz ", "y", "z"),
"abc");
        assertEquals(StringUtils.substringBetween(" yabczyabcz ", "y", "z"),
"abc");
        assertEquals(StringUtils.substringBetween("wx[b]yz", "[", "]"), "b");

        try {
            StringUtils.substringBetween (null, null, null);
        } catch (NullPointerException ex) {
            System.out.println("NullPointerException thrown");
        }
    }
}
```

Результат программы:



Задание 3:

Текст программы:

Stack.java

```java
package com.example.spp_lab4;

import java.util.NoSuchElementException;

/**
 * The <tt>Stack</tt> class represents a last-in-first-out (LIFO) stack of
 * generic items. It supports the usual <em>push</em> and <em>pop</em>
 * operations, along with methods for peeking at the top item, testing if the
 * stack is empty, and iterating through the items in LIFO order.
 * <p>
 */
//TODO FIXME Find Bugs & Write Tests
public class Stack<Item> {
    private int N; // size of the stack
    private Node first; // top of stack
    private Node last;

    // helper linked list class
    private class Node {
        private Item item;
        private Node next;
    }

    /**
     * Create an empty stack.
     */
    public Stack() {
        clear();
    }

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

    /**
     * Is the stack empty?
     */
    public boolean isEmpty() {
        // TODO Implement method
        return N == 0;
    }

    /**
     * Return the number of items in the stack.
     */
    public int size() {
        return N;
    }

    /**
     * Add the item to the stack.
     */
    public void push(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        last = first;
        N++;
        assert check();
    }
```

```java
    /**
     * Delete and return the item most recently added to the stack.
     *
     * @throws java.util.NoSuchElementException if stack is empty.
     */
    public Item pop() {
        // FIXME throw exception if stack is Empty.
        if (isEmpty())
            throw new NoSuchElementException();

        Item item = first.item; // save item to return
        first = first.next; // delete first node
        N--;
        assert check();
        return item; // return the saved item
    }

    /**
     * Return the item most recently added to the stack without deletion.
     *
     * @throws java.util.NoSuchElementException if stack is empty.
     */
    public Item peek() {
        // TODO implement function
        // FIXME throw exception if stack is Empty.
        if (isEmpty())
            throw new NoSuchElementException();

        return first.item;
    }

    public int search(Item searchItem) {
        if (isEmpty())
            return -1;

        int counter = 0;
        Node item = first;

        while (item != null) {
            if (item.item.equals(searchItem)) {
                return counter;
            }

            counter++;
            item = item.next;
        }

        return -1;
    }

    /**
     * Return string representation.
     */
    public String toString() {
        StringBuilder s = new StringBuilder();
        for (Node current = first; current != null; current = current.next) {
            Item item = current.item;
            s.append(item).append(" ");
        }
        return s.toString();
    }

    // check internal invariants
```

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

        // check internal consistency of instance variable N
        int numberOfNodes = 0;

        for (Node x = first; x != null; x = x.next) {
            numberOfNodes++;
        }

        if (numberOfNodes != N) {
            return false;
        }

        return true;
    }
}
```

StackClient.java

```
package com.example.spp_lab4;

import java.util.Scanner;


/**
 * A test client.
 */
public class StackClient {
    public static void main(String[] args) {
        Stack<String> s = new Stack<String>();

        Scanner scanner = new Scanner(System.in);

        while (scanner.hasNext()) {
            String item = scanner.next();
            if (item.equals("stop"))
                break;
            else if (!item.equals("-")) {
                s.push(item);
            } else if (!s.isEmpty()) {
                System.out.println("Deleted element: " + s.pop());
```

```
            }
        }

        System.out.println("Size: " + s.size());
    }
}
```

EmptyStackTest.java

```
package com.example.spp_lab4;

import java.util.NoSuchElementException;
import org.junit.*;
import org.junit.Test;

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

public class EmptyStackTest {
    Stack<String> s = new Stack<String>();

    @Before
    public void initBefore() {
        s.clear();
    }

    @After
    public void initAfter() {

    }

    /**
     * Test of clear method, of class Stack.
     */
    @Test
    public void clearTest() {
        System.out.println("clearTest");

        s.clear();

        assertTrue(s.isEmpty());
        assertEquals(s.size(), 0);
    }

    /**
     * Test of isEmpty method, of class Stack.
     */
    @Test
    public void isEmptyTest() {
        System.out.println("isEmptyTest");

        assertTrue(s.isEmpty());
    }

    /**
     * Test of size method, of class Stack.
     */
    @Test
    public void sizeTest() {
        System.out.println("sizeTest");

        assertEquals(s.size(), 0);
```

```
    }

    /**
     * Test of push method, of class Stack.
     */
    @Test
    public void pushTest() {
        System.out.println("pushTest");

        s.push("1");

        assertEquals(s.size(), 1);
        assertEquals(s.search("1"), 0);
        assertEquals(s.peek(), "1");
    }

    /**
     * Test of pop method, of class Stack.
     */
    @Test
    public void popTest() throws NoSuchElementException {
        System.out.println("popTest");

        try {
            s.pop();
        } catch (NoSuchElementException ex) {
            System.out.println("NoSuchElementException");
        }
    }

    /**
     * Test of peek method, of class Stack.
     */
    @Test
    public void peekTest() throws NoSuchElementException {
        System.out.println("peekTest");

        try {
            s.peek();
        } catch (NoSuchElementException ex) {
            System.out.println("NoSuchElementException");
        }
    }

    /**
     * Test of search method, of class Stack.
     */
    @Test
    public void searchTest() {
        System.out.println("searchTest");

        assertEquals(s.search("1"), -1);
    }

    /**
     * Test of toString method, of class Stack.
     */
    @Test
    public void toStringTest() {
        System.out.println("toStringTest");

        assertEquals(s.toString(), "");
    }
}
```

NotEmptyStackTest.java

```java
package com.example.spp_lab4;

import org.junit.*;
import org.junit.Before;
import org.junit.Test;

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

public class NotEmptyStackTest {
    Stack<String> s = new Stack<String>();

    @Before
    public void initBefore() {
        s.clear();
        s.push("7");
        s.push("1");
        s.push("4");
    }

    @After
    public void initAfter() {

    }

    /**
     * Test of clear method, of class Stack.
     */
    @Test
    public void clearTest() {
        System.out.println("clearTest");

        s.clear();

        assertTrue(s.isEmpty());
        assertEquals(s.size(), 0);
    }

    /**
     * Test of isEmpty method, of class Stack.
     */
    @Test
    public void isEmptyTest() {
        System.out.println("isEmptyTest");

        assertFalse(s.isEmpty());
    }

    /**
     * Test of size method, of class Stack.
     */
    @Test
    public void sizeTest() {
        System.out.println("sizeTest");

        assertEquals(s.size(), 3);
    }

    /**
     * Test of push method, of class Stack.
```

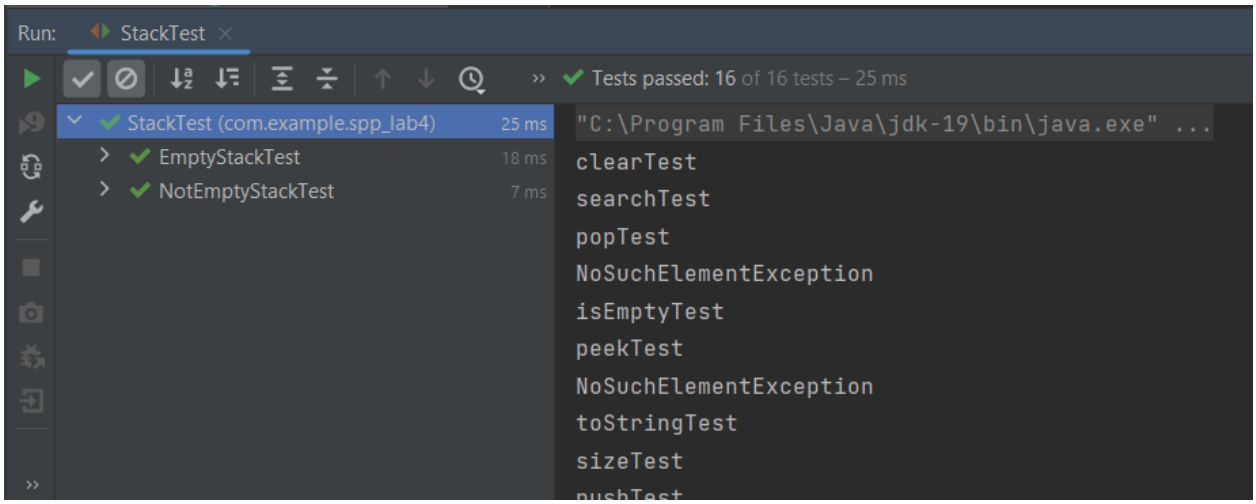
```
     */
    @Test
    public void pushTest() {
        System.out.println("push");

        s.push("0");

        assertEquals(s.size(), 4);
        assertEquals(s.search("1"), 2);
        assertEquals(s.peek(), "0");
    }

    /**
     * Test of pop method, of class Stack.
     */
    @Test
    public void popTest() {
        System.out.println("popTest");

        assertEquals(s.pop(), "4");
        assertEquals(s.size(), 2);
    }

    /**
     * Test of peek method, of class Stack.
     */
    @Test
    public void peekTest() {
        System.out.println("peekTest");

        assertEquals(s.peek(), "4");
    }

    /**
     * Test of search method, of class Stack.
     */
    @Test
    public void searchTest() {
        System.out.println("searchTest");

        assertEquals(s.search("7"), 2);
        assertEquals(s.search("1"), 1);
        assertEquals(s.search("4"), 0);
    }

    /**
     * Test of toString method, of class Stack.
     */
    @Test
    public void toStringTest() {
        System.out.println("toStringTest");

        assertEquals(s.toString(), "4 1 7 ");
    }
}
```

StackTest.java

```
package com.example.spp_lab4;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
```

```java
@RunWith(Suite.class)
@Suite.SuiteClasses({
        EmptyStackTest.class,
        NotEmptyStackTest.class
})
public class StackTest {

}
```

Результат программы:



Вывод: освоили приемы тестирования кода на примере использования библиотеки JUnit.