

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

ОТЧЕТ  
по лабораторной работе №11  
Дисциплина «СПП»

Выполнил:  
Студент гр. ПО-3

Будяков В.В.

Проверил:  
Крощенко А. А.

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

**Задание:**

- Создаете тестовый класс SumTest;
- Напишите тест к методу Sum.accum
- Создайте класс StringUtils, в котором будут находиться реализуемые функции; • Напишите тесты для реализуемых функций.
- Реализуйте функцию String loose(String str, String remove), удаляющую из первой строки все символы, которые есть так же во второй.

Спецификация метода:

```
loose (null , null ) = NullPointerException  
loose (null , *) = null  
loose ("", *) = ""  
loose (* , null ) = *  
loose (* , "") = *  
loose (" hello ", "hl") = "eo"  
loose (" hello ", "le") = "ho"
```

- Импорт проекта Импортируйте один из проектов по варианту:
  - Stack – проект содержит реализацию стека на основе связанного списка: Stack.java.
  - Queue – содержит реализацию очереди на основе связанного 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.

## **Ход работы**

### **1) Текст программы:**

#### **1. Sum**

```
public class Sum {  
    public static Integer accum(Integer... values) { int result = 0;  
        for (int value : values) {  
            result += value;  
        }  
        return result;  
    }  
}
```

#### **2. SumTest**

```
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*;  
  
public class SumTest {  
    @Test  
    public void accumSuccess() {  
        Integer accum = Sum.accum(1, 2, 3, 5);  
        assertNotNull(accum);  
        assertEquals(Integer.valueOf(11), accum); }  
  
    @Test  
    public void accumByIncorrectParam() throws NullPointerException {  
        Throwable thrown = assertThrows(NullPointerException.class, () -> {  
            Integer accum = Sum.accum(null, 2, 3, 5);  
        });  
        assertEquals(thrown.getClass(), NullPointerException.class);  
    }  
}
```

#### **3. StringUtils**

```
public class StringUtils {  
    public static String loose(String str, String remove) {  
        if (remove == null && str == null)  
            throw new NullPointerException();  
        else if (remove == null)  
            return str;  
        if (str == null)  
            return null;  
        String result = "";  
        for (Character c : str.toCharArray()) {
```

```

        if (!remove.contains(c.toString()))
            result = result.concat(c.toString());
    }
    return result;
}
}

```

#### 4. StringUtilsTest

```

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class StringUtilsTest {
    @Test
    public void looseByNullRemove() throws NullPointerException {
        Throwable thrown = assertThrows(NullPointerException.class, () -> {
            StringUtils.loose(null, null);
        });
        assertEquals(thrown.getClass(), NullPointerException.class);
    }
    @Test
    public void looseSuccess() {
        assertNull(StringUtils.loose(null, "any"));
        assertEquals("", StringUtils.loose("", "anystr"));
        assertEquals("anyyyy", StringUtils.loose("anyyyy", null));
        assertEquals("anyyyy", StringUtils.loose("anyyyy", ""));
        assertEquals("eo", StringUtils.loose("hello", "hl"));
        assertEquals("ho", StringUtils.loose("hello", "le"));
    }
}

```

#### 5. Queue

```

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;
    }
    /**
     * Create an empty queue. */
    public Queue() { first = null;
        last = null;
        N = 0;
        assert check();
    }
    /**
     * Is the queue empty? */
}

```

```

    * @return the boolean */
public boolean isEmpty() {
    return first == null;
}
/**
    * Return the number of items in the queue. *
    * @return the int
    */
public int size() {
    return N;
}

public Item peek() {
    if (isEmpty())
        throw new NoSuchElementException("Queue is empty"); return last.item;
}
/**
    * Clean up. */
public void cleanUp() { first = null;
    last = null;
    N = 0; }
/**
    * Add the item to the queue. *
    * @param item the item
    */
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;
}
/**
    * Return string representation. */
public String toString() {
    StringBuilder s = new StringBuilder();
    for (Node x = first; x == null; x = x.next) {

```

```

        s.append(x.item).append(" "); }
    return s.toString();
}
// check internal invariants
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;
        }
    }
// check internal consistency of instance variable last
    Node lastNode = first;
    while (lastNode.next != null) {
        lastNode = lastNode.next;
    }
    return last == lastNode;
}
}
}

```

## 6. QueueTest

```

import org.junit.jupiter.api.*;

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

import java.util.NoSuchElementException;

```

```

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

    @BeforeEach
    public void before() {
        queue.enqueue("1");
        queue.enqueue("2");
        queue.enqueue("3");
    }

    @AfterEach
    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());
}
}

```



Результаты:

Test Results	25 ms
SumTest	25 ms
accumSuccess()	22 ms
accumByIncorrectParam()	3 ms

ests passed: 2

Run | TODO | Problems | Terminal | Build

Test Results	32 ms
StringUtilsTest	32 ms
looseByNullRemove()	28 ms
looseSuccess()	4 ms

ests passed: 2

Run | TODO | Problems | Terminal | Build

Test Results	31 ms
QueueTest	31 ms
dequeue_SizeEqual3_Success()	21 ms
cleanUp_SizeEqual3_Success()	1 ms
size_SizeEqual4_Success()	1 ms
dequeue_QueueIsEmpty_ThrowException()	2 ms
isEmpty_SizeEqual0_True()	1 ms
isEmpty_SizeEqual3_False()	1 ms
enqueue_SizeEqual3_Success()	
size_SizeEqual3_Success()	1 ms
peek_QueueIsEmpty_ThrowException()	1 ms
dequeue_SizeEqual1_Success()	1 ms
peek_QueueIsNotEmpty_Return3()	1 ms

Run | TODO | Problems | Terminal | Build

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