Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №11**

**По дисциплине «СПП»**

Выполнил

студент 3 курса группы ПО-3:

Григорьева В.А.

Проверил:

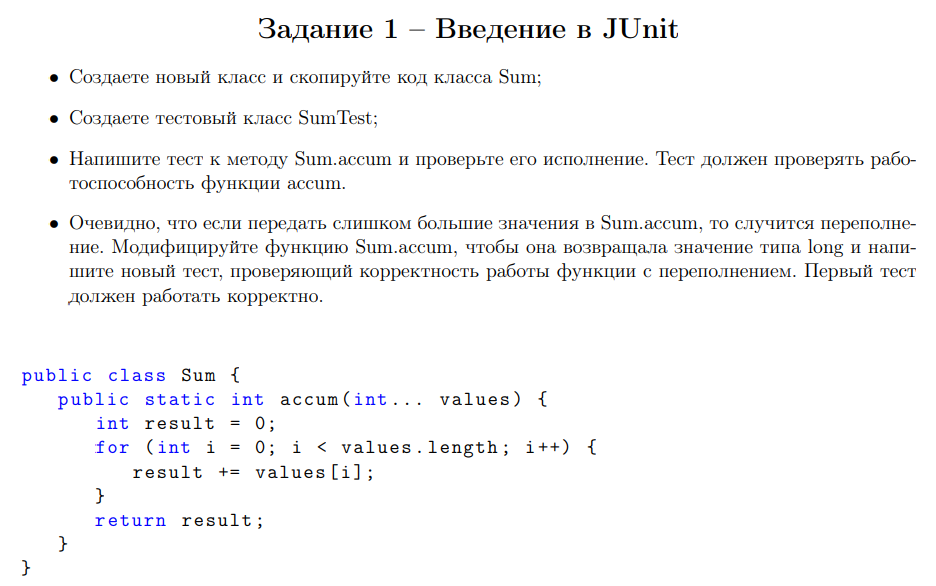
Крощенко А.А.

**Брест, 2021**

**Вариант 8**

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

**Ход работы**:

****

**Код программы:**

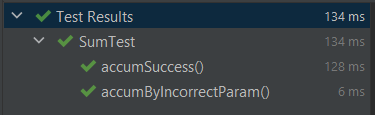
**Sum.java**

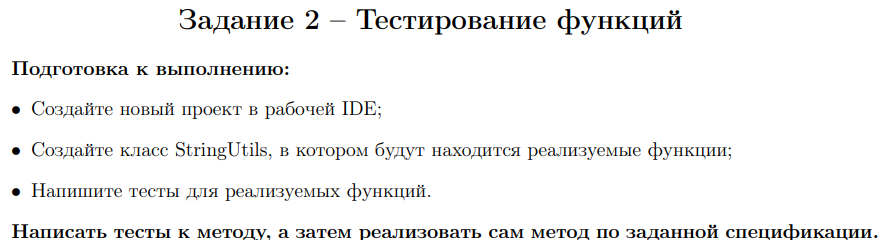
public class Sum {  
 public static Integer accum(Integer... values) { int result = 0;  
 for (int value : values) {  
 result += value;  
 }  
 return result;  
 }  
}

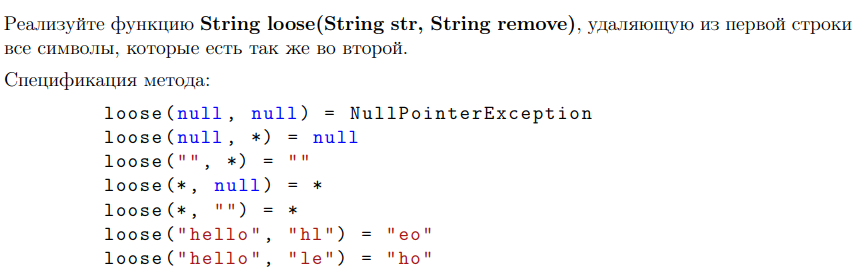
**SumTest.java**

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);  
 }  
}

**Результат выполнения:**





****

**Код программы:**

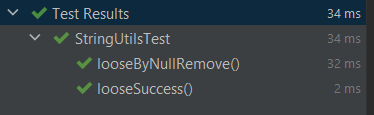
**StringUtils.java**

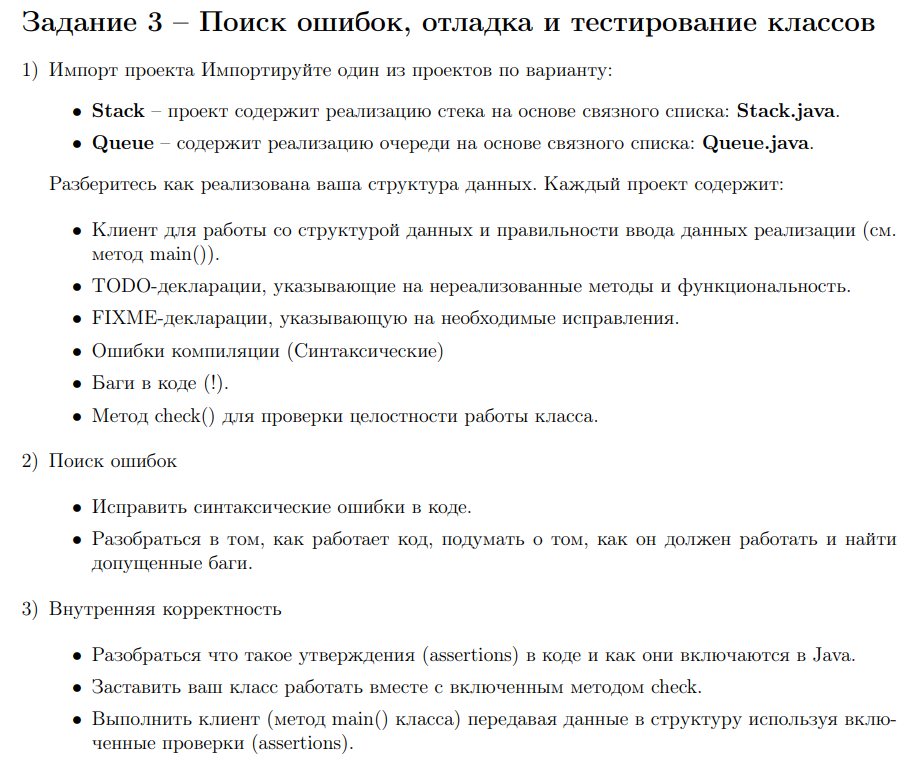
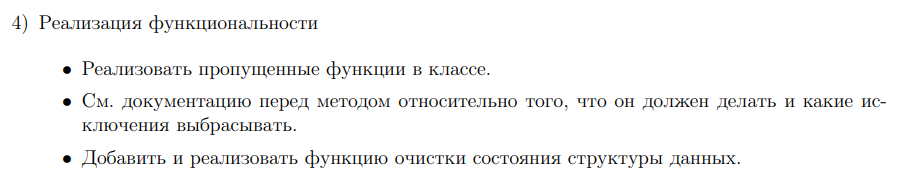
public class StringUtils {  
 public static String loose(String str, String remove) {  
 if (remove == null && str == null)  
 throw new NullPointerException();  
 else if (remove == null)  
 return str;  
 if (str == null)  
 return null;  
 String result = "";  
 for (Character c : str.toCharArray())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

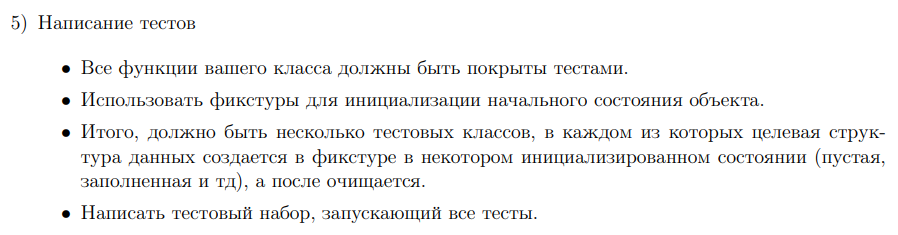
**StringUtilsTest.java**

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, "yum"));  
 *assertEquals*("", StringUtils.*loose*("", "yumm"));  
 *assertEquals*("yummy", StringUtils.*loose*("yummy", null));  
 *assertEquals*("yummy", StringUtils.*loose*("yummy", ""));  
 *assertEquals*("eo", StringUtils.*loose*("lesson", "lsn"));  
 *assertEquals*("on", StringUtils.*loose*("lesson", "les"));  
}

**Результат выполнения:**

****

**** ****



**Код программы:**

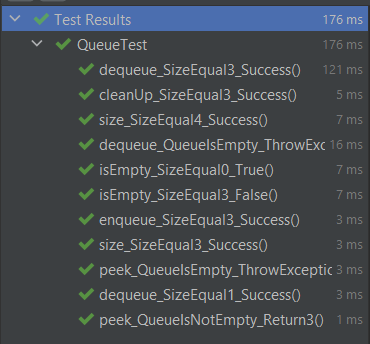
**Queue.java**

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;  
 }  
 }  
}

**QueueTest.java**

public class QueueTest {  
 private Queue<String> queue = new Queue<>();  
  
 */\*\*  
 \* Before.  
 \*/* @BeforeEach  
 public void before() {  
 queue.enqueue("1");  
 queue.enqueue("2");  
 queue.enqueue("3");  
 }  
  
 */\*\*  
 \* After.  
 \*/* @AfterEach  
 public void after() {  
 queue.cleanUp();  
 }  
  
 */\*\*  
 \* Is empty size equal 3 false.  
 \*/* @Test  
 public void isEmpty\_SizeEqual3\_False() {  
 *assertFalse*(queue.isEmpty());  
 }  
  
 */\*\*  
 \* Is empty size equal 0 true.  
 \*/* @Test  
 public void isEmpty\_SizeEqual0\_True() {  
 queue.cleanUp();  
 *assertTrue*(queue.isEmpty());  
 }  
  
 */\*\*  
 \* Size size equal 3 success.  
 \*/* @Test  
 public void size\_SizeEqual3\_Success() {  
 *assertEquals*(3, queue.size());  
 }  
  
 */\*\*  
 \* Size size equal 4 success.  
 \*/* @Test  
 public void size\_SizeEqual4\_Success() {  
 queue.enqueue("4");  
 *assertEquals*(4, queue.size());  
 }  
  
 */\*\*  
 \* Peek queue is empty throw exception.  
 \*/* @Test  
 public void peek\_QueueIsEmpty\_ThrowException() throws NoSuchElementException {  
 Throwable thrown = *assertThrows*(NoSuchElementException.class, () -> {  
 queue.cleanUp();  
 queue.peek();  
 });  
 *assertEquals*(thrown.getClass(), NoSuchElementException.class);  
 }  
  
 */\*\*  
 \* Peek queue is not empty return 3.  
 \*/* @Test()  
 public void peek\_QueueIsNotEmpty\_Return3() {  
 *assertEquals*("3", queue.peek());  
 }  
  
 */\*\*  
 \* Clean up size equal 3 success.  
 \*/* @Test  
 public void cleanUp\_SizeEqual3\_Success() {  
 *assertEquals*(3, queue.size());  
 queue.cleanUp();  
 *assertEquals*(0, queue.size());  
 }  
  
 */\*\*  
 \* Enqueue size equal 3 success.  
 \*/* @Test  
 public void enqueue\_SizeEqual3\_Success() {  
 *assertEquals*(3, queue.size());  
 queue.enqueue("4");  
 *assertEquals*(4, queue.size());  
 }  
  
 */\*\*  
 \* Dequeue size equal 3 success.  
 \*/* @Test  
 public void dequeue\_SizeEqual3\_Success() {  
 *assertEquals*("1", queue.dequeue());  
 }  
  
 */\*\*  
 \* Dequeue queue is empty throw exception.  
 \*/* @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);  
 }  
  
 */\*\*  
 \* Dequeue size equal 1 success.  
 \*/* @Test  
 public void dequeue\_SizeEqual1\_Success() {  
 queue.cleanUp();  
 queue.enqueue("str");  
 *assertEquals*("str", queue.dequeue());  
 *assertEquals*(0, queue.size());  
 }  
}

**Результат выполнения:**

****

**Вывод:** В ходе выполнения лабораторной работы были освоены приемы тестирования кода на примере использования библиотеки JUnit.