Министерство образования Республики Беларусь

Учреждение образования «Брестский государственный технический университет»

Кафедра ИИТ

Отчёт по лабораторной работе №11

Выполнил:

студент 3 курса

группы ПО-5

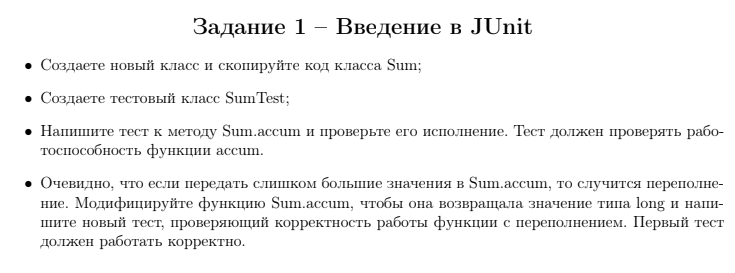
Лозейко П. А.

Проверил:

Крощенко А. А.

Брест 2022

**Вариант 9**



Класс **Sum:**

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 ;

}

}

**SumTest:**

import org.testng.annotations.Test;

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

public class SumTest {

@Test

public void accumSuccess() {

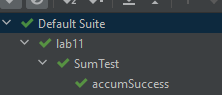
int accum = Sum.accum(1, 2, 3, 4, 5);

assertEquals(15, accum);

}

}

Результат:



Модифицированный метод accum:

public static long modaccum(long ... values){

long result = 0;

for ( int i = 0; i < values.length;i++){

result += values [i];

}

return result ;

}

Тест для метода:

@Test

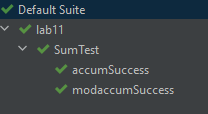
public void modaccumSuccess(){

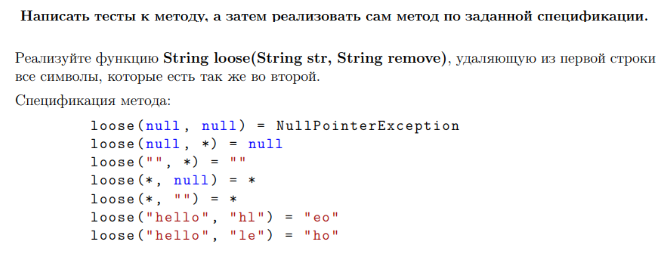
long accum = Sum.modaccum(1\_000\_000\_000,1\_000\_000\_000 );

assertEquals(2\_000\_000\_000, accum);

}

Результат:





Класс **task2:**

public class task2 {

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;

}

}

**task2Test:**

import org.junit.platform.commons.util.StringUtils;

import org.testng.annotations.Test;

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

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

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

public class task2Test {

@Test

public void looseByNullRemove() throws NullPointerException {

Throwable thrown = assertThrows(NullPointerException.class, () -> {

task2.loose(null, null);

});

assertEquals(thrown.getClass(), NullPointerException.class);

}

@Test

public void looseSuccess() {

assertEquals("", task2.loose("", "hello"));

assertEquals("hello", task2.loose("hello", null));

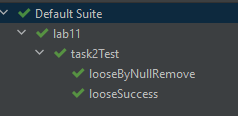
assertEquals("hello", task2.loose("hello", ""));

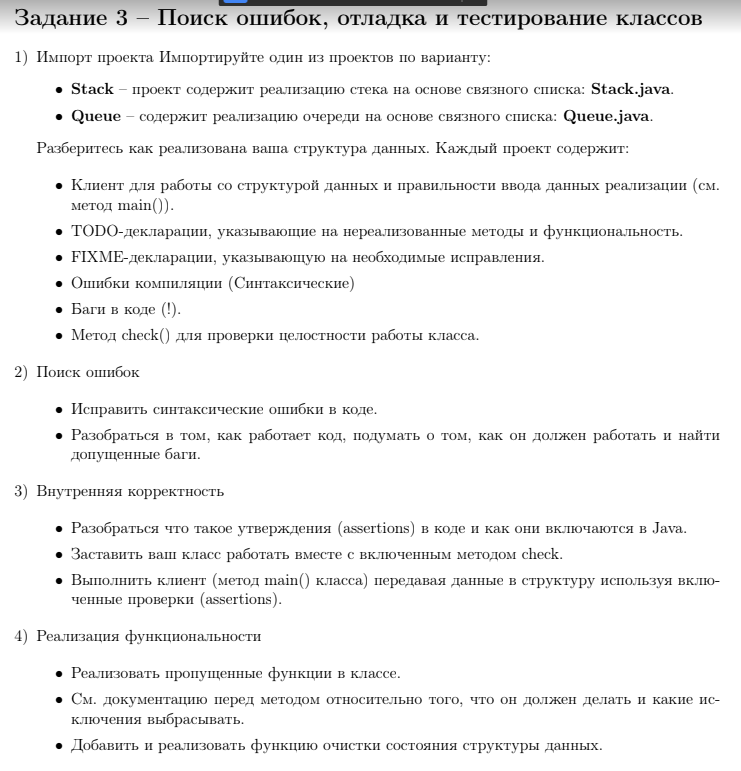
assertEquals("eo", task2.loose("hello", "hl"));

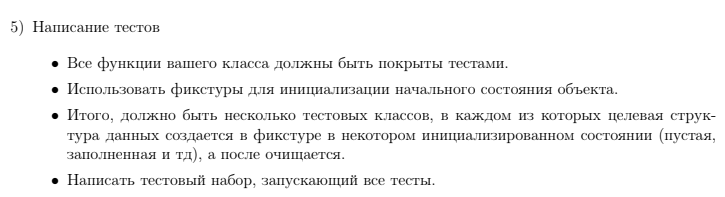
assertEquals("o", task2.loose("hello", "hel"));

}

}







**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;

}

}

}

**QueueTest:**

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.testng.annotations.Test;

import java.util.NoSuchElementException;

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

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());

}

}

