Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

По дисциплине: «СПП»

Тема: “Основы тестирования. JUnit”

Выполнил:

Студент 3 курса

Группы ПО-7

Комиссаров А.Е.

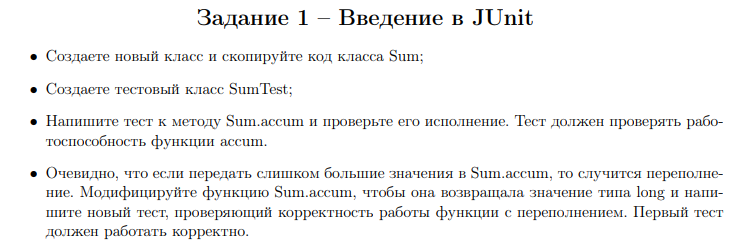
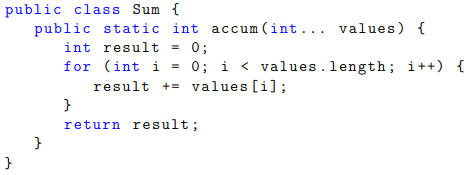
Проверил:

Монтик Н.С.

2023

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

**Ход выполнения:**

****

**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

}

}

**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);

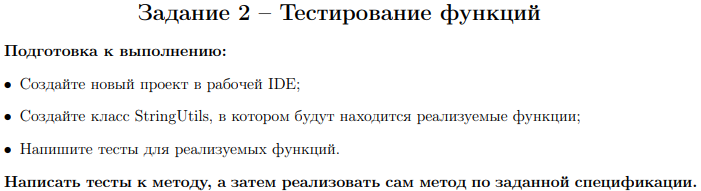
long expected = Integer.MAX\_VALUE + 2L; // ожидание

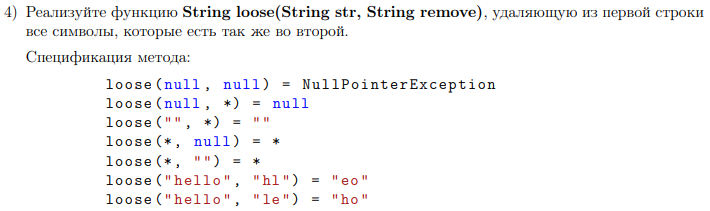
System.out.println(expected);

assertEquals(expected, actual); // проверка на эквивалентность

}

}

****

****

**StringUtils.java**

package task3;

public class StringUtils {

public static String loose(String str, String remove){

if(str == null && remove != null){

return null;

}

else if(str == null && remove == null){

throw new NullPointerException();

}

else if(str.equals(" ")){

return str;

}

else{

String out;

String st = str.toLowerCase();

String rem = remove.toLowerCase();

char[] t = new char[st.length()];

for(int k = 0; k < st.length(); k++){

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

if(st.charAt(k) == rem.charAt(i)){

st = st.replace(st.charAt(k), ' ');

}

}

}

out = st.replaceAll(" ", "");

return out;

}

}

}

**StringUtilsTest.java**

package task3;

import org.junit.Test;

import static org.junit.Assert.\*;

public class StringUtilsTest {

@Test(expected = NullPointerException.class)

public void looseByNullRemove() {

StringUtils su = new StringUtils();

String actual1 = su.loose(null, null);

String expected1 = "";

assertEquals(expected1, actual1);

}

@Test

public void loose() {

StringUtils su = new StringUtils();

String actual2 = su.loose(null, "help");

String expected2 = null;

assertEquals(expected2, actual2);

assertNull(actual2);

String actual3 = su.loose("help", "");

String expected3 = "help";

assertEquals(expected3, actual3);

String actual4 = su.loose(" ", "help");

String expected4 = " ";

assertEquals(expected4, actual4);

String actual5 = su.loose("help", " ");

String expected5 = "help";

assertEquals(expected5, actual5);

String actual6 = su.loose("Hello My World", "hmwl");

String expected6 = "eoyord";

assertEquals(expected6, actual6);

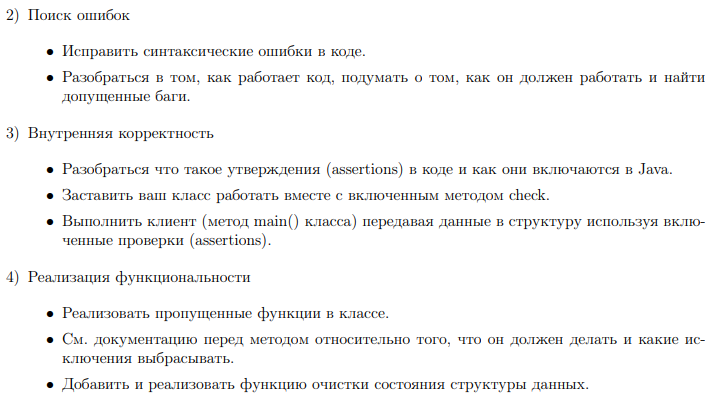
}

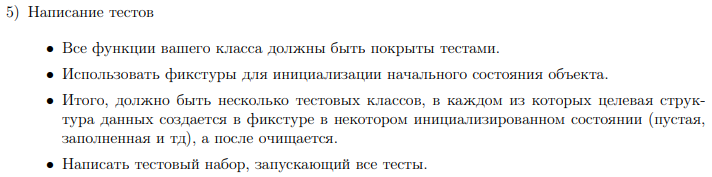
}

****

****

****

****

****

**Queue.java**

package queue;  
  
import java.util.NoSuchElementException;  
  
/\*\*  
 \* The <tt>Queue</tt> class represents a first-in-first-out (FIFO) queue of  
 \* generic items. It supports the usual <em>enqueue</em> and <em>dequeue</em>  
 \* operations, along with methods for peeking at the top item, testing if the  
 \* queue is empty, and iterating through the items in FIFO order.  
 \*/  
  
public class Queue<Item> {  
 // the number of elements  
 private int N;  
  
 // the head  
 private Node first;  
  
 // the tail //fixed  
 private Node last;  
  
 // simple Node  
 private class Node {  
 private Item item;  
 private Node next;  
 }  
  
 /\*\*  
 \* Create an empty queue.  
 \*/  
 public Queue() {  
 clear(); //fixed  
 }  
  
 /\*\*  
 \* Clear queue.  
 \*/  
 public void clear() {  
 first = null;  
 last = null;  
 N = 0;  
 assert check();  
 }  
  
 /\*\*  
 \* Is the queue empty?  
 \*/  
 public boolean isEmpty() {  
 return first == null; // != // ==  
 }  
  
 /\*\*  
 \* Return the number of items in the queue.  
 \*/  
 public int size() {  
 return N;  
 }  
  
 /\*\*  
 \* Return the item least recently added to the queue.  
 \*  
 \* @throws java.util.NoSuchElementException if queue is empty.  
 \*/  
 public Item peek() {  
 if (isEmpty()) {  
 throw new NoSuchElementException(); // added  
 }  
 return first.item;  
 }  
  
 /\*\*  
 \* Add the item to the queue.  
 \*/  
 public void enqueue(Item item) {  
 Node oldLast = last; // oldlast // oldLast  
  
 last = new Node();  
 last.item = item;  
 last.next = null;  
  
 ++N; // added  
  
 if (isEmpty()) {  
 first = last;  
 }  
 else {  
 oldLast.next = last;  
 }  
  
 assert check();  
 }  
  
 /\*\*  
 \* Remove and return the item on the queue least recently added.  
 \*  
 \* @throws java.util.NoSuchElementException if queue is empty.  
 \*/  
 // name fixed  
 public Item dequeue() {  
 if (isEmpty()) {  
 throw new NoSuchElementException(); // added  
 }  
  
 Item item = first.item;  
 first = first.next;  
  
 --N;  
  
 if (isEmpty()) {  
 last = null; // fixed  
 }  
  
 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(" "); // added  
 }  
  
 return s.toString();  
 }  
  
 // internal invariants checking  
 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 {  
 // become more wide  
 if (first == null || last == null) {  
 return false;  
 }  
 if (first == last) {  
 return false;  
 }  
 if (first.next == null) {  
 return false;  
 }  
 if (last.next != null) {  
 return false;  
 }  
  
 // internal consistency of instance variable N checking  
 int numberOfNodes = 0;  
  
 for (Node x = first; x != null; x = x.next) {  
 numberOfNodes++;  
 }  
  
 if (numberOfNodes != N) {  
 return false;  
 }  
  
 // internal consistency of instance variable last checking  
 Node lastNode = first;  
  
 while (lastNode.next != null) {  
 lastNode = lastNode.next;  
 }  
  
 return last == lastNode;  
 }  
 }  
}

**QueueClient.java**

package queue;  
  
import java.util.Scanner;  
  
public class QueueClient {  
  
 /\*\*  
 \* A test client.  
 \*/  
 public static void main(String[] args) {  
 Queue<String> q = new Queue<String>();  
  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.println("Start...");  
 System.out.println("Enter your values. If you entered '-', the element will be showed");  
  
 while (scanner.hasNext()) {  
 String item = scanner.next();  
  
 if (!item.equals("-")) {  
 q.enqueue(item);  
 System.out.println("Elements has: "+q.size());  
 }  
 else if (!q.isEmpty()) {  
 System.out.println(q.dequeue() + " ");  
 System.out.println("Elements has: "+q.size());  
 }  
 }  
 }  
}

**2) test->queue**

**QueueEmptyTest.java**

package queue;  
  
import org.junit.After;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class QueueEmptyTest {  
 Queue<String> queue = new Queue<>();  
  
 @After  
 public void queueClear() throws Exception {  
 queue.clear();  
 }  
  
 @Test  
 public void clear\_when\_queueIsCleaned\_should\_returnEmptyResult() {  
 queue.clear();  
 assertTrue(queue.isEmpty());  
 assertEquals(0, queue.size());  
 assertEquals("", queue.toString());  
 }  
  
 @Test  
 public void isEmptyResult\_when\_queueIsCleaned\_should\_returnIsEmpty() {  
 assertTrue(queue.isEmpty());  
 }  
  
 @Test  
 public void size\_when\_queueIsCleaned\_should\_returnNullSize() {  
 assertEquals(0, queue.size());  
 }  
  
 @Test(expected = java.util.NoSuchElementException.class)  
 public void peek\_when\_queueIsCleaned\_should\_returnException() {  
 queue.peek();  
 }  
  
 @Test  
 public void enqueue\_when\_queueIsAddedWithOneElement\_should\_returnWorkingResult() {  
 queue.enqueue("Test1");  
 assertFalse(queue.isEmpty());  
 assertEquals(1, queue.size());  
 assertEquals("Test1 ", queue.toString());  
 }  
  
 @Test(expected = java.util.NoSuchElementException.class)  
 public void dequeue\_when\_queueIsCleaned\_should\_returnException() {  
 queue.dequeue();  
 }  
  
 @Test  
 public void toString\_when\_queueIsCleaned\_should\_returnEmptyString() {  
 assertEquals("", queue.toString());  
 }  
}

**QueueManyElementsTest.java**

package queue;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class QueueManyElementsTest {  
 Queue<String> queue = new Queue<>();  
  
 @Before  
 public void init() throws Exception {  
 queue.enqueue("Test1");  
 queue.enqueue("Test2");  
 }  
  
 @After  
 public void queueClear() throws Exception {  
 queue.clear();  
 }  
  
 @Test  
 public void clear\_when\_queueIsFilledWithTwoElements\_should\_returnEmptyResult() {  
 queue.clear();  
 assertTrue(queue.isEmpty());  
 assertEquals(0, queue.size());  
 assertEquals("", queue.toString());  
 }  
  
 @Test  
 public void isEmpty\_when\_queueIsFilledWithTwoElements\_should\_returnIsNotEmpty() {  
 assertFalse(queue.isEmpty());  
 }  
  
 @Test  
 public void size\_when\_queueIsFilledWithTwoElements\_should\_returnResultTwo() {  
 assertEquals(2, queue.size());  
 }  
  
 @Test  
 public void peek\_when\_queueIsFilledWithTwoElements\_should\_returnWorkingResult() {  
 assertEquals(2, queue.size());  
 assertEquals("Test1", queue.peek());  
 assertEquals(2, queue.size());  
 }  
  
 @Test  
 public void enqueue\_when\_queueIsAddedWithOneElement\_should\_returnWorkingResultAndDecreasedSize() {  
 queue.enqueue("Test3");  
 assertFalse(queue.isEmpty());  
 assertEquals(3, queue.size());  
 assertEquals("Test1 Test2 Test3 ", queue.toString());  
 }  
  
 @Test  
 public void dequeue\_when\_queueIsErasedWithOneElement\_should\_returnWorkingResultAndErasedSize() {  
 assertEquals(2, queue.size());  
 assertEquals("Test1", queue.dequeue());  
 assertEquals(1, queue.size());  
 }  
  
 @Test  
 public void toString\_when\_queueIsFilledWithTwoElements\_should\_returnFilledString() {  
 assertEquals("Test1 Test2 ", queue.toString());  
 }  
}

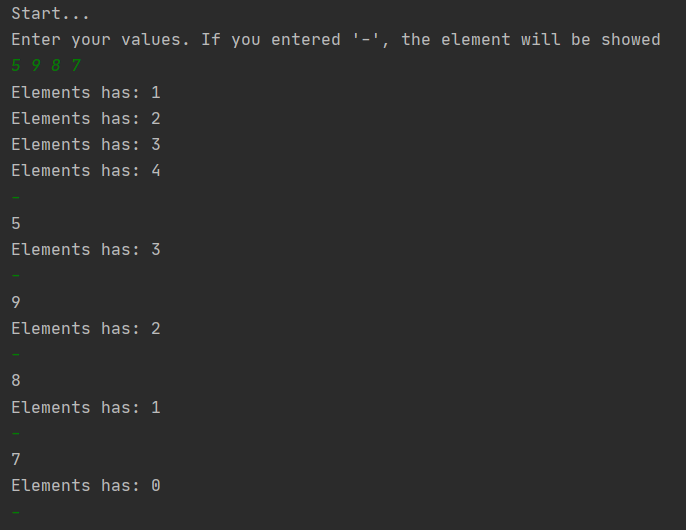
**QueueOneElementTest.java**

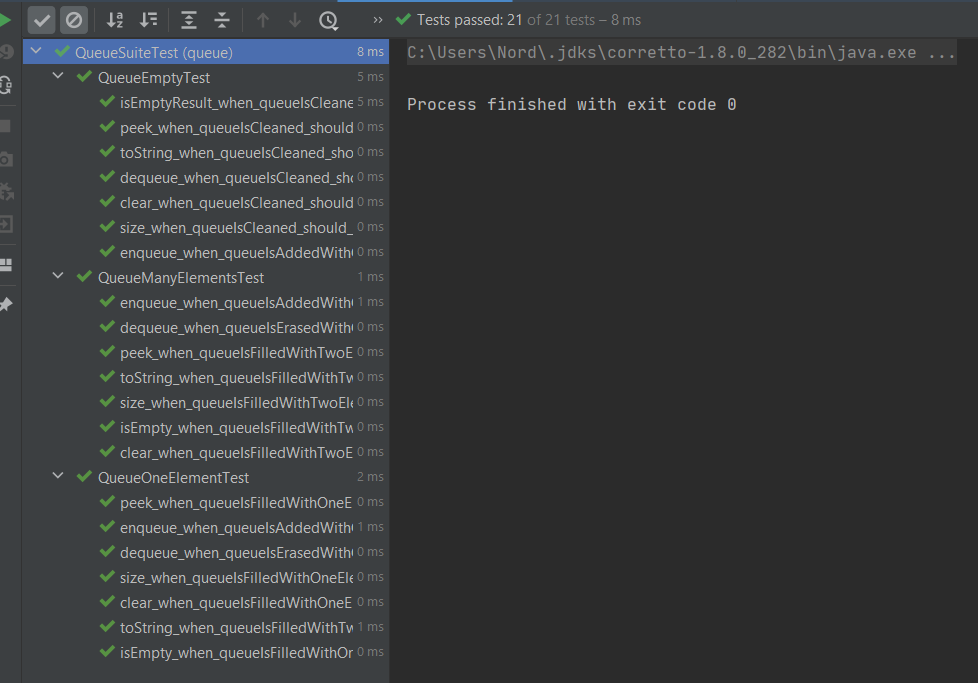
package queue;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class QueueOneElementTest {  
 Queue<String> queue = new Queue<>();  
  
 @Before  
 public void init() throws Exception {  
 queue.enqueue("Test1");  
 }  
  
 @After  
 public void queueClear() throws Exception {  
 queue.clear();  
 }  
  
 @Test  
 public void clear\_when\_queueIsFilledWithOneElement\_should\_returnEmptyResult() {  
 queue.clear();  
 assertTrue(queue.isEmpty());  
 assertEquals(0, queue.size());  
 assertEquals("", queue.toString());  
 }  
  
 @Test  
 public void isEmpty\_when\_queueIsFilledWithOneElement\_should\_returnIsNotEmpty() {  
 assertFalse(queue.isEmpty());  
 }  
  
 @Test  
 public void size\_when\_queueIsFilledWithOneElement\_should\_returnResultOne() {  
 assertEquals(1, queue.size());  
 }  
  
 @Test  
 public void peek\_when\_queueIsFilledWithOneElement\_should\_returnWorkingResult() {  
 assertEquals(1, queue.size());  
 assertEquals("Test1", queue.peek());  
 assertEquals(1, queue.size());  
 }  
  
 @Test  
 public void enqueue\_when\_queueIsAddedWithOneElement\_should\_returnWorkingResultAndDecreasedSize() {  
 queue.enqueue("Test2");  
 assertFalse(queue.isEmpty());  
 assertEquals(2, queue.size());  
 assertEquals("Test1 Test2 ", queue.toString());  
 }  
  
 @Test  
 public void dequeue\_when\_queueIsErasedWithOneElement\_should\_returnWorkingResultAndErasedSize() {  
 assertEquals(1, queue.size());  
 assertEquals("Test1", queue.dequeue());  
 assertEquals(0, queue.size());  
 assertTrue(queue.isEmpty());  
 }  
  
 @Test  
 public void toString\_when\_queueIsFilledWithTwoElements\_should\_returnFilledString() {  
 assertEquals("Test1 ", queue.toString());  
 }  
}

**QueueSuiteTest.java**

package queue;  
  
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
  
@RunWith(Suite.class)  
@Suite.SuiteClasses ({QueueEmptyTest.class, QueueManyElementsTest.class, QueueOneElementTest.class})  
public class QueueSuiteTest {  
  
}

**Результат работы программы:**

****

****

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