Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №11**

По дисциплине «СПП»

за 6-й семестр

Выполнил:

студент 3 курса

группы ПО-3 (1)

Лущ М. Г.

Проверил:

Крощенко А.А.

Брест, 2021

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

**Вариант:** 14

**Задание 1:**

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.

**Задание 2:**

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.

Вариант 5) Реализуйте и протестируйте метод int indexOfDifference(String str1, String str2) который сравнивает две строки и возвращает индекс той позиции, в которой они различаются. Например, indexOfDifference("i am a machine" , "i am a robot") должно вернуть 7.

Спецификация метода:

indexOfDifference (null , null ) = NullPointerException

indexOfDifference ("", "") = -1

indexOfDifference ("", "abc ") = 0

indexOfDifference ("abc ", "") = 0

indexOfDifference ("abc ", " abc ") = -1

indexOfDifference ("ab", " abxyz ") = 2

indexOfDifference (" abcde ", " abxyz ") = 2

indexOfDifference (" abcde ", "xyz") = 0

**Задание 3:**

1) Импорт проекта Импортируйте один из проектов по варианту (2):

• 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.

**Код программы:**

**1) queue**

**Queue**

package queue;  
  
import java.util.NoSuchElementException;  
  
public class Queue<Item> {  
 private int N;  
 private Node first;  
 private Node last;  
  
 private class Node  
 {  
 private Item item;  
 private Node next;  
 }  
  
 public Queue()  
 {  
 clear();  
 }  
  
 public void clear()  
 {  
 first = null;  
 last = null;  
 N = 0;  
 assert check();  
 }  
  
 public boolean isEmpty()  
 {  
 return first == null;  
 }  
  
 public int size()  
 {  
 return N;  
 }  
  
 public Item peek()  
 {  
 if (isEmpty())  
 {  
 throw new NoSuchElementException(); // added  
 }  
 return first.item;  
 }  
  
 public void enqueue(Item item)  
 {  
 Node oldLast = last;  
  
 last = new Node();  
 last.item = item;  
 last.next = null;  
  
 ++N;  
  
 if (isEmpty())  
 {  
 first = last;  
 }  
 else  
 {  
 oldLast.next = last;  
 }  
  
 assert check();  
 }  
  
 public Item dequeue()  
 {  
 if (isEmpty())  
 {  
 throw new NoSuchElementException();  
 }  
  
 Item item = first.item;  
 first = first.next;  
  
 --N;  
  
 if (isEmpty())  
 {  
 last = null;  
 }  
  
 assert check();  
 return item;  
 }  
  
 public String toString()  
 {  
 StringBuilder s = new StringBuilder();  
  
 for (Node x = first; x != null; x = x.next)  
 {  
 s.append(x.item).append(" ");  
 }  
  
 return s.toString();  
 }  
  
 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 == null || last == null)  
 {  
 return false;  
 }  
 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;  
 }  
  
 Node lastNode = first;  
  
 while (lastNode.next != null)  
 {  
 lastNode = lastNode.next;  
 }  
  
 return last == lastNode;  
 }  
 }  
}

**Queue tests:**

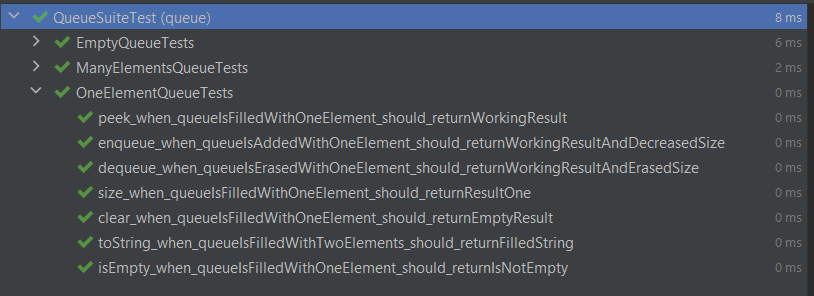
package queue;  
  
import org.junit.After;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class EmptyQueueTests  
{  
 Queue<String> queue = new Queue<>();  
  
 @After  
 public void queueClear() throws Exception  
 {  
 queue.clear();  
 }  
  
 @Test  
 public void clear\_when\_queueIsCleaned\_should\_returnEmptyResult()  
 {  
 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());  
 }  
}

package queue;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class ManyElementsQueueTests  
{  
 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());  
 }  
}

package queue;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
import static org.junit.Assert.\*;  
  
public class OneElementQueueTests  
{  
 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());  
 }  
}

package queue;  
  
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
  
@RunWith(Suite.class)  
@Suite.SuiteClasses ({EmptyQueueTests.class, ManyElementsQueueTests.class, OneElementQueueTests.class})  
public class QueueSuiteTest  
{  
  
}

**Результат выполнения тестов:**

****

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