МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №11

По дисциплине: «Современные платформы программирования»

Выполнил:
Студент ФЭИС
3-го курса, группы ПО-5
Белко В. А.
Проверил:
Крощенко А. А.

Брест 2022

Вариант: 2

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

Задание 1 – Введение в JUnit
• 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 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 – Тестирование функций
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.
2) Разработайте метод String repeat(String str, String separator, int repeat), который строит строку из указанного паттерна, повторённого заданное количество раз, вставляя строку-разделитель при каждом повторении.
Спецификация метода:
repeat ("e", "|", 0) = ""
repeat ("e", "|", 3) = "e|e|e"
repeat (" ABC ", ",", 2) = "ABC , ABC "
repeat (" DBE ", "", 2) = " DBEDBE "
repeat (" DBE ", ":", 1) = "DBE"
repeat ("e", -2) = IllegalArgumentException
repeat ("", ":", 3) = "::"
repeat (null , "a", 1) = NullPointerException
repeat ("a", null , 2) = NullPointerException

Задание 3 – Поиск ошибок, отладка и тестирование классов
1) Импорт проекта Импортируйте один из проектов по варианту:
• 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 вклю-
ченные 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.

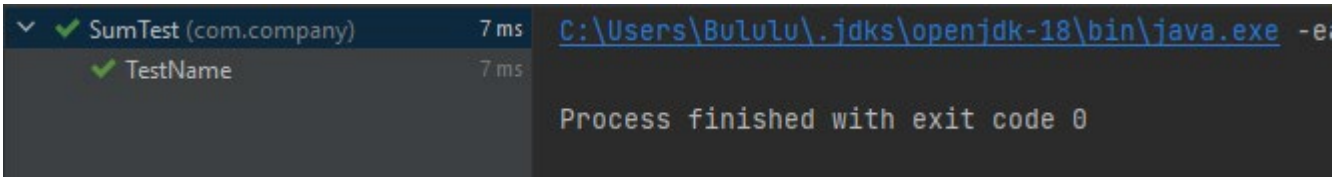
Код программы:
1)

Sum.java:
```
package com.company;

public class Sum {
    public static long accum(int ...values){
        long result = 0;
        for( int i = 0; i < values.length ; i++) {
            result += values [i];
        }
        return result;
    }
}
```

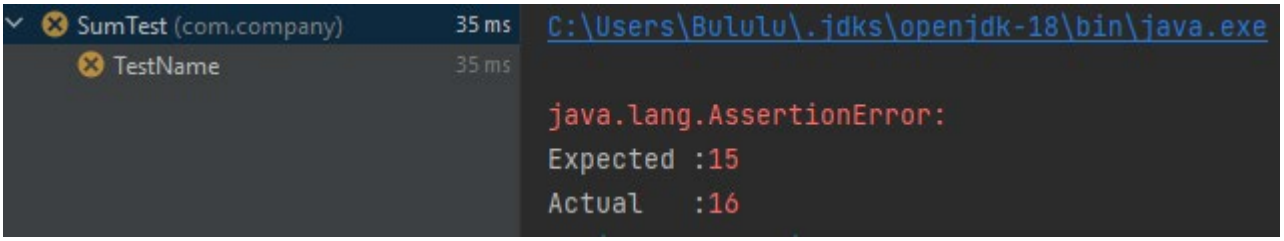
SumTest.java:
```
package com.company;

import org.junit.*; // Импорт всех основных классов и аннотаций JUnit
import static org.junit.Assert.*; // Импорт утверждений
```

```java
public class SumTest {
    @Test
    public void TestName () {
        long test = Sum.accum(2,0,7,7);
    assertEquals(16, test);
    }
}
```

Результат выполнения программы:



И в случае, если числа не совпадают:



2)
Код программы:

StringUtils.java:

```java
package com.company;

public class StringUtils {

    static String repeat(String str, String separator, int repeat){
        if (repeat < 0){
            throw new IllegalArgumentException();
        }

        if(str == null || separator == null){
            throw new NullPointerException();
        }

        if(repeat == 1){
            str = str.trim();
            return str;
        }

        if(separator.equals(",")){
            str = str.trim();
            String temp = "";
            for (int i =1; i <= repeat; i++){
                temp += str + " ";
                if(i<repeat){
                    temp += separator + " ";
                }
            }
            return temp;
        }
```

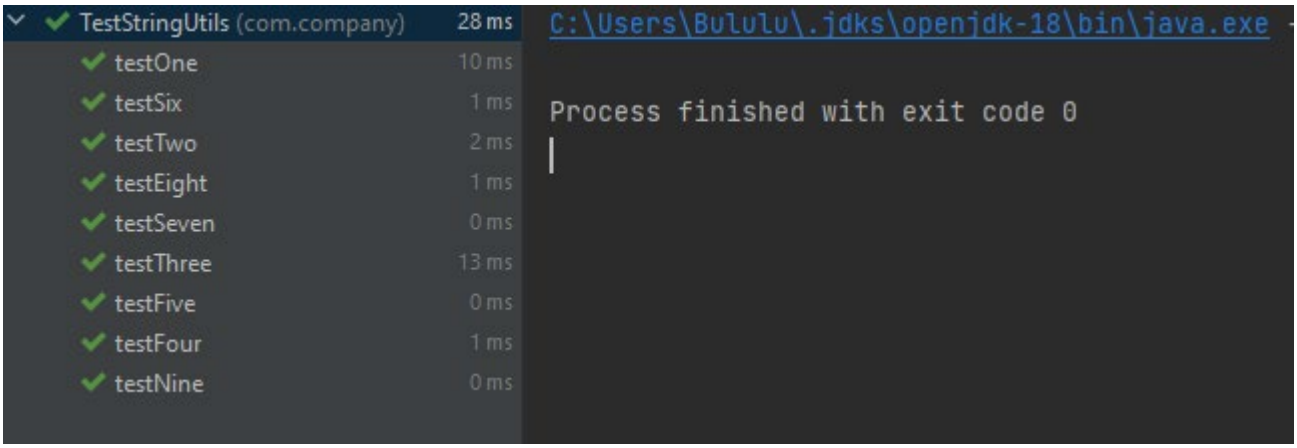
```
        if(separator.equals("")){
            String temp = "";
            str = str.trim();
            for (int i = 1; i <= repeat; i++){
                if(i%2 != 0){
                    temp += " " + str;
                }else {
                    temp += str + " ";
                }
            }
            return temp;
        }

        String res = "";
        for(int i = 1; i <= repeat; i++){
            res += str;
            if(i < repeat){
                res += separator;
            }
        }
        return res;
    }

    static String repeat(String str, int repeat){
        throw new IllegalArgumentException();
    }

    static String repeat(String str){
        throw new IllegalArgumentException();
    }
}
```

TestStringUtils.java:

```
package com.company;

import org.junit.*;
import static org.junit.Assert.*;

public class TestStringUtils {
    @Test
    public void testOne(){
        assertEquals("", StringUtils.repeat("e","|",0));
    }

    @Test
    public void testTwo(){
        assertEquals("e|e|e", StringUtils.repeat("e", "|", 3));
    }

    @Test
    public void testThree(){
        assertEquals("ABC , ABC ", StringUtils.repeat(" ABC ", ",", 2));
    }

    @Test
    public void testFour(){
        assertEquals(" DBEDBE ", StringUtils.repeat(" DBE ", "", 2));
    }

    @Test
    public void testFive(){
        assertEquals("DBE", StringUtils.repeat(" DBE ", ":", 1));
    }

    @Test ( expected = IllegalArgumentException.class )
    public void testSix(){
```

```
        StringUtils.repeat("e", -2);
    }

    @Test
    public void testSeven(){
        assertEquals("::", StringUtils.repeat("", ":", 3));
    }

    @Test (expected = NullPointerException.class )
    public void testEight(){
        StringUtils.repeat(null , "a", 1);
    }

    @Test (expected = NullPointerException.class )
    public void testNine(){
        StringUtils.repeat("a", null , 2);
    }
}
```

Результат выполнения программы:



3)
Код программы:

Stack.java:

```
package stack;

import java.util.NoSuchElementException;

public class Stack<Item> {
    private int N; // size of the stack
    private Node first; // top of stack

    private class Node {
        private Item item;
        private Node next;
    }

    public Stack() {
        first = null;
        N = 0;
        assert check();
    }

    public boolean isEmpty() {
        return (N < 0);
```

```
    }

    public int size() {
        return N;
    }

    public void push(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
        N++;
        assert check();
    }

    public Item pop() {
        // FIXME throw exception if stack is Empty.

        if(size() == 0){
            throw  new NoSuchElementException();
        }
        Item item = first.item; // save item to return
        first = first.next; // delete first node
        N--;
        assert check();
        return item; // return the saved item
    }

    public Item peek() {
        if(size() == 0){
            throw  new NoSuchElementException();
        }else{
            return first.item;
        }
    }

    public String toString() {
        StringBuilder s = new StringBuilder();
        for (Node current = first; current != current.next; current = current.next) {
            Item item = current.item;
            s.append(item);
            if(null == current.next){
                return s.toString();
            }
            s.append(" - ");

        }
        return s.toString();
    }

    private boolean check() {
        if (N == 0) {
            if (first != null) {
                return false;
            }
        } else if (N == 1) {
            if (first == null) {
                return false;
            }
            if (first.next != null) {
                return false;
            }
        } else {
            if (first.next == null) {
                return false;
            }
        }
```

```
        int numberOfNodes = 0;
        for (Node x = first; x != null; x = x.next) {
            numberOfNodes++;
        }
        if (numberOfNodes != N) {
            return false;
        }
        return true;
    }
}
```

StackClient.java:

```
package stack;

import java.util.Scanner;

public class StackClient {
    public static void main(String[] args) {
        Stack<String> s = new Stack<String>();

        Scanner scanner = new Scanner(System.in);

        while (scanner.hasNext()) {
            String item = scanner.next();
            if (!item.equals("-")) {
                s.push(item);
            } else if (!s.isEmpty()) {
                System.out.println(s.pop() + " ");
            }
        }

        System.out.println(s.size());
    }
}
```

StackTest1.java:

```
package stack;

import org.junit.*;
import static org.junit.Assert.*;

public class StackTest1 {

    Stack<String> stack;

    @Before
    public void setUpBeforTest(){
        stack = new Stack<>();
    }

    @Test ( expected = java.util.NoSuchElementException . class )
    public void testPop(){

        stack.pop();
    }

    @Test ( expected = java.util.NoSuchElementException . class )
    public void testPeek(){

        stack.peek();
    }
```

```
}
```

StackTest2.java:

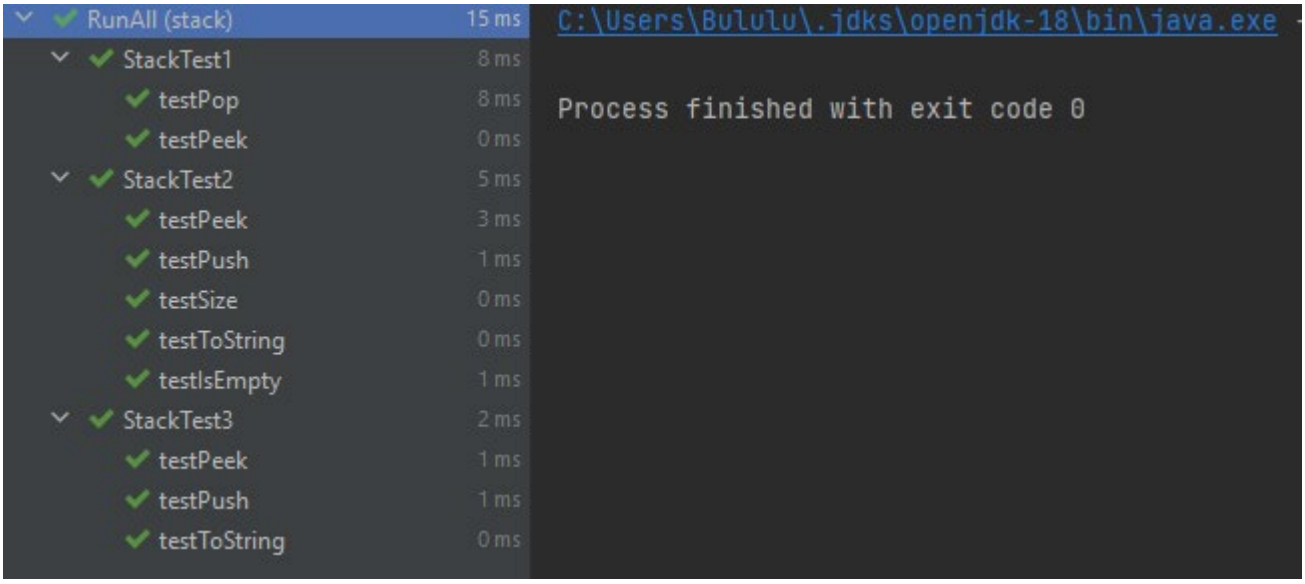
```
package stack;

import org . junit .*;
import static org . junit . Assert .*;

public class StackTest2 {

    Stack<String> stack;

    @Before
    public void setUpBeforTest(){
        stack = new Stack<>();
    }

    @Test
    public void testIsEmpty(){

        assertEquals(false, stack.isEmpty());
    }

    @Test
    public void testSize(){

        assertEquals(0, stack.size());
    }

    @Test
    public void testPush(){

        stack.push("555");
        assertEquals("555", stack.pop());
    }

    @Test
    public void testPeek(){

        stack.push("555");
        assertEquals("555", stack.peek());
    }

    @Test
    public void testToString(){
        stack.push("555");
        stack.push("444");
        stack.push("333");
        assertEquals("333 - 444 - 555", stack.toString());
    }
}
```

StackTest3.java:

```
package stack;

import org . junit .*;
import static org . junit . Assert .*;

public class StackTest3 {

    Stack<String> stack;
```

```
    @Before
    public void setUpBeforTest(){
        stack = new Stack<>();
        stack.push("5");
        stack.push("4");
        stack.push("3");
    }

    @Test
    public void testPush(){

        assertEquals("3", stack.pop());
    }

    @Test
    public void testPeek(){

        assertEquals("3", stack.peek());
    }

    @Test
    public void testToString(){

        assertEquals("3 - 4 - 5", stack.toString());
    }
}
```

Результат выполнения программы:



Вывод: В ходе лабораторной работы научился работать с библиотекой JUnit, освоил приёмы тестирования кода.