МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №11

Специальность ПО-5(о)

Выполнил

Д. С. Бриштен,

студент группы ПО-5

Проверил

А. А. Крощенко

ст. преп. каф. ИИТ

Брест 2022

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

Вариант 3

**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.

**Sum.java**

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

**SumTest.java**

public class SumTest {  
 @Test  
 public void accum\_returnSum\_whenSmallNumbersArePassed() {  
 Assert.*assertEquals*(7, Sum.*accum*(-10, 0, 5, 12));  
 }  
  
 @Test  
 public void accum\_returnSum\_whenBigNumbersArePassed() {  
 Assert.*assertEquals*(5\_000\_000\_000L, Sum.*accum*(10, 3\_000\_000\_000L, -10, 2\_000\_000\_000L));  
 }  
}

**Задание 2.** Напишите метод String keep(String str, String pattern) который оставляет в первой строке  
все символы, которые присутствуют во второй.  
Спецификация метода:  
keep (null , null ) = NullPointerException  
keep (null , \*) = null  
keep ("", \*) = ""  
keep (\*, null ) = ""  
keep (\*, "") = ""  
keep (" hello ", "hl") = " hll "  
keep (" hello ", "le") = " ell "

**StringUtils.java**

public final class StringUtils {  
 public static String keep(String str, String pattern) {  
 if (str == null && pattern == null) {  
 throw new NullPointerException();  
 }  
 if (str == null) {  
 return null;  
 }  
 if (pattern == null) {  
 return "";  
 }  
 if (str.equals("") || pattern.equals("")) {  
 return "";  
 }  
  
 StringBuilder result = new StringBuilder();  
 for (char ch : str.toCharArray()) {  
 if (pattern.indexOf(ch) == -1) {  
 continue;  
 }  
 result.append(ch);  
 }  
 return result.toString();  
 }  
}

**StringUtilsTest.java**

public class StringUtilsTest {  
 @Test(expected = NullPointerException.class)  
 public void keep\_throwsNullPointerException\_whenStrAndPatternIsNull() {  
 StringUtils.*keep*(null, null);  
 }  
 @Test  
 public void keep\_return\_whenStrIsNull() {  
 Assert.*assertNull*(StringUtils.*keep*(null, "str"));  
 }  
  
 @Test  
 public void repeat\_returnsEmptyString\_whenStrIsEmptyString() {  
 Assert.*assertEquals*("", StringUtils.*keep*("", "str"));  
 }  
  
 @Test  
 public void repeat\_returnsEmptyString\_whenPatternIsNull() {  
 Assert.*assertEquals*("", StringUtils.*keep*("str", null));  
 }  
  
 @Test  
 public void repeat\_returnsEmptyString\_whenPatternIsEmptyString() {  
 Assert.*assertEquals*("", StringUtils.*keep*("str", ""));  
 }  
  
 @Test  
 public void keep\_returnsKeepString() {  
 Assert.*assertEquals*("hll", StringUtils.*keep*("hello", "hl"));  
 Assert.*assertEquals*("ell", StringUtils.*keep*("hello", "le"));  
 }  
}

**Задание 3.** Поиск ошибок, отладка и тестирование классов – **Stack**.

**Stack.java**

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() {  
 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() {  
 if (isEmpty()) {  
 throw new NoSuchElementException();  
 }  
 Item item = first.item; // save item to return  
 first = first.next; // delete first node  
 N--;  
 assert check();  
 return item;   
 }  
public Item peek() {  
 if (isEmpty()) {  
 throw new NoSuchElementException();  
 }  
 return first.item;  
 }  
  
 public void clear() {  
 first = null;  
 N = 0;  
 assert check();  
 }  
public String toString() {  
 StringBuilder s = new StringBuilder();  
 for (Node current = first; current != null; current = current.next) {  
 Item item = current.item;  
 s.append(item).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**

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());  
 }  
}

**StackTestRunner.java**

@RunWith(Suite.class)  
@SuiteClasses({EmptyStackTest.class, NonEmptyStackTest.class})  
public class StackTestRunner {  
}

**EmptyStackTest.java**

public class EmptyStackTest {  
 private final Stack<String> stack = new Stack<>();  
  
 @Test  
 public void assertIsEmpty() {  
 Assert.*assertTrue*(stack.isEmpty());  
 }  
  
 @Test  
 public void assertSize() {  
 Assert.*assertEquals*(0, stack.size());  
 }  
 @Test  
 public void assertPush() {  
 String str = "String";  
 stack.push(str);  
  
 Assert.*assertFalse*(stack.isEmpty());  
 Assert.*assertEquals*(1, stack.size());  
 Assert.*assertEquals*(str + " ", stack.toString());  
 }  
 @Test(expected = java.util.NoSuchElementException.class)  
 public void assertPop() {  
 stack.pop();  
 }  
 @Test(expected = java.util.NoSuchElementException.class)  
 public void assertPeek() {  
 stack.peek();  
 }  
 @Test  
 public void assertClear() {  
 stack.clear();  
  
 Assert.*assertTrue*(stack.isEmpty());  
 Assert.*assertEquals*(0, stack.size());  
 Assert.*assertEquals*("", stack.toString());  
 }  
  
 @Test  
 public void assertToString() {  
 Assert.*assertEquals*("", stack.toString());  
 }  
  
 @After  
 public void clearStack(){  
 stack.clear();  
 }  
}

**NonEmptyStackTest.java**

public class NonEmptyStackTest {  
 private final Stack<String> stack = new Stack<>();  
  
 @Before  
 public void setUp() {  
 stack.push("string1");  
 stack.push("string2");  
 }  
  
 @Test  
 public void assertIsEmpty() {  
 Assert.*assertFalse*(stack.isEmpty());  
 }  
  
 @Test  
 public void assertSize() {  
 Assert.*assertEquals*(2, stack.size());  
 }  
  
 @Test  
 public void assertPeek() {  
 Assert.*assertEquals*(2, stack.size());  
 Assert.*assertEquals*("string2", stack.peek());  
 Assert.*assertEquals*(2, stack.size());  
 }  
  
 @Test  
 public void assertPush() {  
 String str = "string3";  
 stack.push(str);  
  
 Assert.*assertFalse*(stack.isEmpty());  
 Assert.*assertEquals*(3, stack.size());  
 Assert.*assertEquals*(str + " string2 string1 ", stack.toString());  
 }  
  
 @Test  
 public void assertPop() {  
 Assert.*assertEquals*(2, stack.size());  
 Assert.*assertEquals*("string2", stack.pop());  
 Assert.*assertEquals*(1, stack.size());  
 }  
  
 @Test  
 public void assertClear() {  
 stack.clear();  
  
 Assert.*assertTrue*(stack.isEmpty());  
 Assert.*assertEquals*(0, stack.size());  
 Assert.*assertEquals*("", stack.toString());  
 }  
  
 @Test  
 public void assertToString() {  
 Assert.*assertEquals*("string2 string1 ", stack.toString());  
 }  
  
 @After  
 public void clearStack() {  
 stack.clear();  
 }  
}

Вывод: освоены приемы тестирования кода на примере использования библиотеки JUnit.