МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИК БЕЛАРУСЬ
УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
КАФЕДРА ИИТ

Лабораторная работа №5
по дисциплине «Современные платформы программирования»

Выполнила:
Андросюк М.М.
Группа: ПО-5
Проверил:
Крощенко А.А.

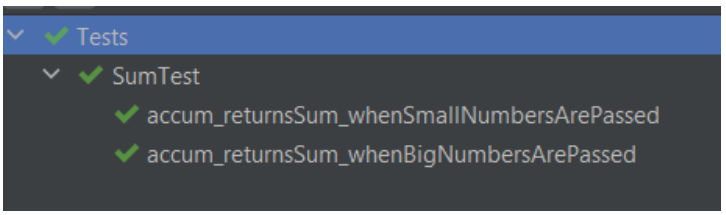
Брест 2022

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

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.

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

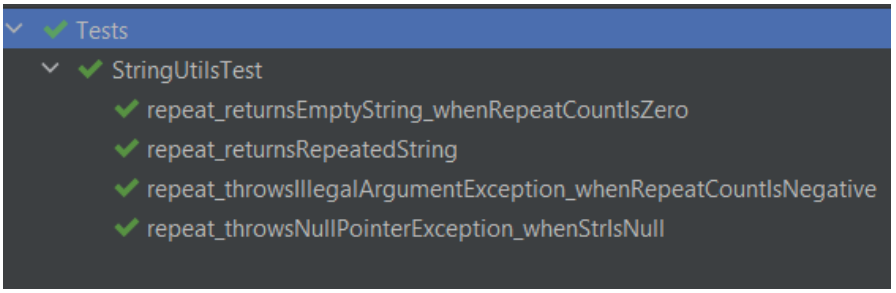
```
public class SumTest {
  @Test
  public void accum_returnsSum_whenSmallNumbersArePassed() {
    Assert.assertEquals(7, Sum.accum(-10, 0, 5, 12));
  }

  @Test
  public void accum_returnsSum_whenBigNumbersArePassed() {
    Assert.assertEquals(3_999_999_995L, Sum.accum(2_000_000_000, 2_000_000_000, 5, -
    10));
  }
}
```



Задание 2: Создайте новый проект в рабочей IDE. Создайте класс StringUtils, в котором будут находится реализуемые функции. Напишите тесты для реализуемых функций. Реализуйте и протестируйте метод String repeat(String pattern, int repeat), который строит строку из указанного паттерна, повторённого заданное количество раз.

```
public final class StringUtils {
  public static String repeat(String str, int
    repeatCount) { if (str == null) {
      throw new NullPointerException();
    }

    if (repeatCount < 0) {
      throw new IllegalArgumentException();
    }
```

```
    StringBuilder result = new StringBuilder(str.length() * repeatCount);

    for (int i = 0; i < repeatCount; i++)
      { result.append(str);
    }

    return result.toString();
  }
}

public class StringUtilsTest {
  @Test
  public void repeat_returnsRepeatedString() {
    Assert.assertEquals("ABCABC", StringUtils.repeat("ABC",
    2));
  }

  @Test
  public void repeat_returnsEmptyString_whenRepeatCountIsZero()
    { Assert.assertEquals("", StringUtils.repeat("ABC", 0));
  }

  @Test(expected = IllegalArgumentException.class)
  public void repeat_throwsIllegalArgumentException_whenRepeatCountIsNegative() {
    StringUtils.repeat("ABC", -3);
  }

  @Test(expected = NullPointerException.class)
  public void repeat_throwsNullPointerException_whenStrIsNull()
    { StringUtils.repeat(null, 10);
  }
}
```



Задание 3

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

  // helper linked list
  class private class
  Node {
    private Item
    item; private
    Node next;
```

```
}

/**
 * Create an empty stack.
 */
public Stack()
  { assert
  check();
}

/**
 * Is the stack empty?
 */
public boolean isEmpty() {
  return N == 0;
}

/**
 * Return the number of items in the stack.
 */
public int size()
  { return N;
}

/**
 * Add the item to the stack.
 */
public void push(Item item)
  { Node oldFirst = first;
  first = new Node();
  first.item = item;
  first.next =
  oldFirst; N++;
  assert check();
}

/**
 * Delete and return the item most recently added to the stack.
 *
 * @throws java.util.NoSuchElementException if stack is empty.
 */
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; // return the saved item
}

/**
```

```
 * Return the item most recently added to the stack without deletion.
 *
 * @throws java.util.NoSuchElementException if stack is empty.
 */
public Item peek()
  { if (isEmpty())
  {
    throw new NoSuchElementException();
  }
  return first.item;
}

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

/**
 * Return string representation.
 */
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();
}

// check internal invariants
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;
    }
  }

  // check internal consistency of instance
  variable N int numberOfNodes = 0;
  for (Node x = first; x != null; x =
    x.next) { numberOfNodes++;
```

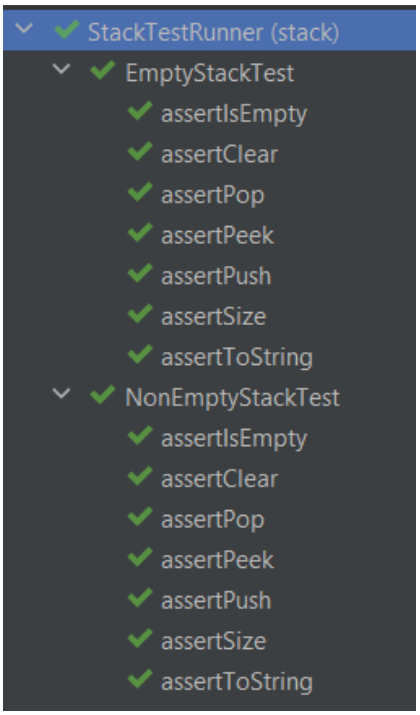
```
    }
    if (numberOfNodes != N) {
      return false;
    }

    return true;
  }
}
```

```
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();
  }
}



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();
  }
}
```

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



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