МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра компьютерной математики и программирования

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

acc.		Д.А.Кочин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

«Оценка качества тестовой базы»

по курсу: УПРАВЛЕНИЕ КАЧЕСТВОМ ПО

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4832	31.10.20	К.А. Корюнщенков
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1. Цель работы:

Целью данной лабораторной работы является получение навыков по функциональному тестированию кода методом белого ящика (всех ветвей), а также инъекция багов и оценка выполнения при этом тестов.

2. Задание на лабораторную работу:

1) Оценить по отдельности и вместе покрытие тестами, разработанными в лабораторной работе номер 2 и 3.

2) Описать недостающие тесты

3) Выполнить инъекцию багов, оценить качество разработанных тестов.

3. Описание программы:

В ходе лабораторной работы проводилось тестирование программы, для написания которой использовалась среда разработки JetBrains IDEA и язык программирования Java. Покрытие класса Parse Unit-тестами проверялось с помощью библиотеки JUnit.

Main.java

```
class Pair<T,U> {
    public final T t;
    public final U u;

    public Pair(T t, U u) {
        this.t= t;
        this.u= u;
    }
}

enum ErrorEnum {
    Good,
    ErrorSymbol,
    ErrorFirstOrLastItemIsNotNumber
}

public class Main {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Введите уравнение: ");
        String data = in.nextLine();
        System.out.println(Parse.action(data));
    }
}

class Parse{
    public static boolean isNumeric(char str) {
        try {
            Integer.parseInt(String.valueOf(str));
            return true;
        } catch(NumberFormatException e){
            return false;
        }
    }

    public static ErrorEnum checkData(String data){
        for (int i=0;i<data.length();i++){
```

```
            if ((i == 0 || i == data.length() - 1) &&
!isNumeric(data.charAt(i))) {
                return ErrorEnum.ErrorFirstOrLastItemIsNotNumber;
            }
            if (isNumeric(data.charAt(i)) |
String.valueOf(data.charAt(i)).equals("+")
                    | String.valueOf(data.charAt(i)).equals("*") |
String.valueOf(data.charAt(i)).equals("/") |
String.valueOf(data.charAt(i)).equals("-")){
                continue;
            } else {
                return ErrorEnum.ErrorSymbol;
            }
        }
        return ErrorEnum.Good;
    }

    public static Pair<LinkedList<Integer>, LinkedList<String>>
parseData(String data){
        LinkedList<Integer> number = new LinkedList<Integer>();
        LinkedList<String> action = new LinkedList<String>();
        String timeNumber = "";
        for (int i=0;i<data.length();i++){
            if (!isNumeric(data.charAt(i))){
                action.add(String.valueOf(data.charAt(i)));
                if (timeNumber != "") {
                    number.add(Integer.valueOf(timeNumber));
                    timeNumber = "";
                }
            } else{
                timeNumber += String.valueOf(data.charAt(i));
            }
        }
        if (timeNumber != "") {
            number.add(Integer.valueOf(timeNumber));
            timeNumber = "";
        }
        return new Pair(number,action);
    }

    public static Pair<LinkedList<Integer>, LinkedList<String>>
hardAction(LinkedList<Integer> number, LinkedList<String> action){
        Integer i = 0;
        Boolean check = true;
        while (check){
            if (action.get(i).equals("*")) {
                number.set(i,number.get(i)*number.get(i+1));
                number.remove(i+1);
                action.remove(i+1-1);
                i = 0;
            }else if (action.get(i).equals("/")) {
                number.set(i,number.get(i)/number.get(i+1));
                number.remove(i+1);
                action.remove(i+1-1);
                i = 0;
            }
            i += 1;
            if (action.size() <= i){
                break;
            }
        }
        return new Pair(number,action);
    }
```

```
    public static String simpleAction(LinkedList<Integer> number,
LinkedList<String> action){
        Integer i = 0;

        while (!action.isEmpty()){
            if (action.get(i).equals("+")) {
                number.set(i,number.get(i)+number.get(i+1));
                number.remove(i+1);
                action.remove(i+1-1);
                i = 0;
            }else if (action.get(i).equals("-")) {
                number.set(i,number.get(i)-number.get(i+1));
                number.remove(i+1);
                action.remove(i+1-1);
                i = 0;
            }
        }
        return String.valueOf(number.get(0));
    }

    public static String action(String data){
        switch (checkData(data)){
            case Good: {
                Pair<LinkedList<Integer>, LinkedList<String>> parse =
parseData(data);
                parse = hardAction(parse.t, parse.u);
                return simpleAction(parse.t, parse.u);
            }
            case ErrorSymbol:
                return "Error - symbol";
            case ErrorFirstOrLastItemIsNotNumber:
                return "Error - first or last item is not number";
            default:
                return null;
        }
    }
}
```

MainTest.java

```
class MainTest {

    @org.junit.jupiter.api.Test
    void action_1(){
        String line = Parse.action("5+5*1-2");
        Assert.assertEquals("8",line);
    }

    @org.junit.jupiter.api.Test
    void action_2(){
        String line = Parse.action("5+1*2");
        Assert.assertEquals("7",line);
    }

    @org.junit.jupiter.api.Test
    void action_3(){
        String line = Parse.action("5/4");
        Assert.assertEquals("1",line);
    }

    @org.junit.jupiter.api.Test
    void action_4(){
```

```
        String line = Parse.action("+5");
        Assert.assertEquals("Error - first or last item is not number",line);
    }

    @org.junit.jupiter.api.Test
    void action_5(){
        String line = Parse.action("5^2");
        Assert.assertEquals("Error - symbol",line);
    }

    @org.junit.jupiter.api.Test
    void action_6(){
        String line = Parse.action("-5+2");
        Assert.assertEquals("Error - first or last item is not number",line);
    }


    @org.junit.jupiter.api.Test
    void action_black_1(){
        String line = Parse.action("5+5+1-2");
        Assert.assertEquals("9",line);
    }

    @org.junit.jupiter.api.Test
    void action_black_2(){
        String line = Parse.action("5+5+1-2");
        Assert.assertEquals("9",line);
    }

    @org.junit.jupiter.api.Test
    void action_black_3(){
        String line = Parse.action("5+1*2");
        Assert.assertEquals("7",line);
    }

    @org.junit.jupiter.api.Test
    void action_black_4(){
        String line = Parse.action("5+4");
        Assert.assertEquals("9",line);
    }
}
```

4. Спецификация на тесты:

Функция action (string) – тестирование методом белого ящика

Имя теста	Описание сценария	Входные данные	Выходные данные
action _1	Входная строка корректная и нам вернется результат выражения	Строка: 5+5*1-2	Строка: 8
action_2	Входная строка корректная и нам вернется результат выражения	Строка: 5+1*2	Строка: 7
action_3	Входная строка корректная и нам вернется результат выражения	Строка: 5/4	Строка: 1

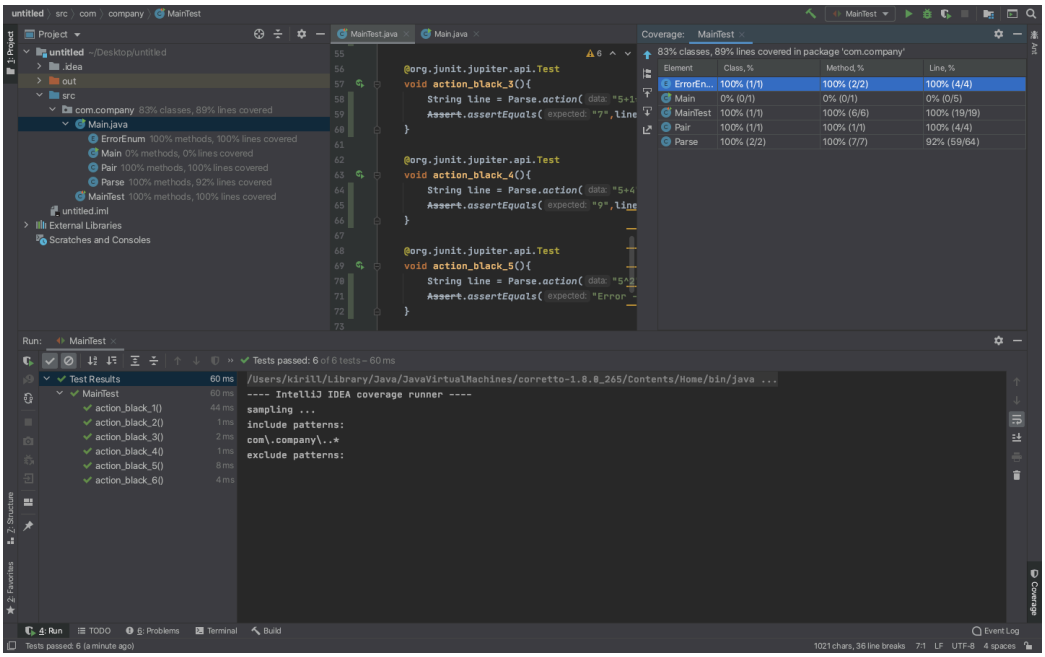
action_4	Входная строка начинается не с числа и из-за этого нам возвращается сообщение Error – first or last item is not number	Строка: +5	Строка: Error – first or last item is not number
action_5	Входная строка содержит неизвестный символ (возведение в степень) и из-за этого нам возвращается сообщение Error - symbol	Строка: 5^2	Строка: Error - symbol
action_6	Входная строка начинается не с числа и из-за этого нам возвращается сообщение Error – first or last item is not number	Строка: -5+2	Строка: Error – first or last item is not number

Функция action (string) – тестирование методом черного ящика

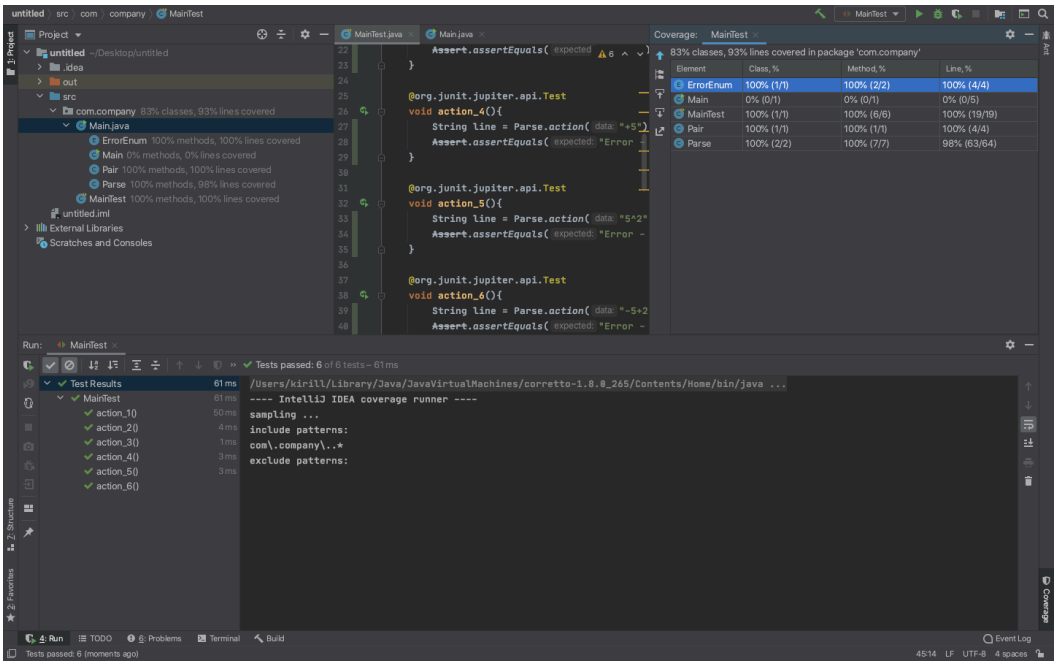
Имя теста	Описание сценария	Входные данные	Выходные данные
action _1	Преобразование входных данных в строку, которое содержит чисто или ошибку	Строка: 5+5+1-2	Строка: 9
action_2	Преобразование входных данных в строку, которое содержит чисто или ошибку	Строка: 5+1*2	Строка: 7
action_3	Преобразование входных данных в строку, которое содержит чисто или ошибку	Строка: 5+4	Строка: 9
action_4	Преобразование входных данных в строку, которое содержит чисто или ошибку	Строка: +	Строка: не верные входные данные
action_5	Преобразование входных данных в строку, которое содержит чисто или ошибку	Строка: 5^2	Строка: Error - symbol
action_6	Преобразование входных данных в строку, которое содержит чисто или ошибку	Строка: -5+2	Строка: Error – first or last item is not number

5. Анализ покрытия функции Unit-тестами:

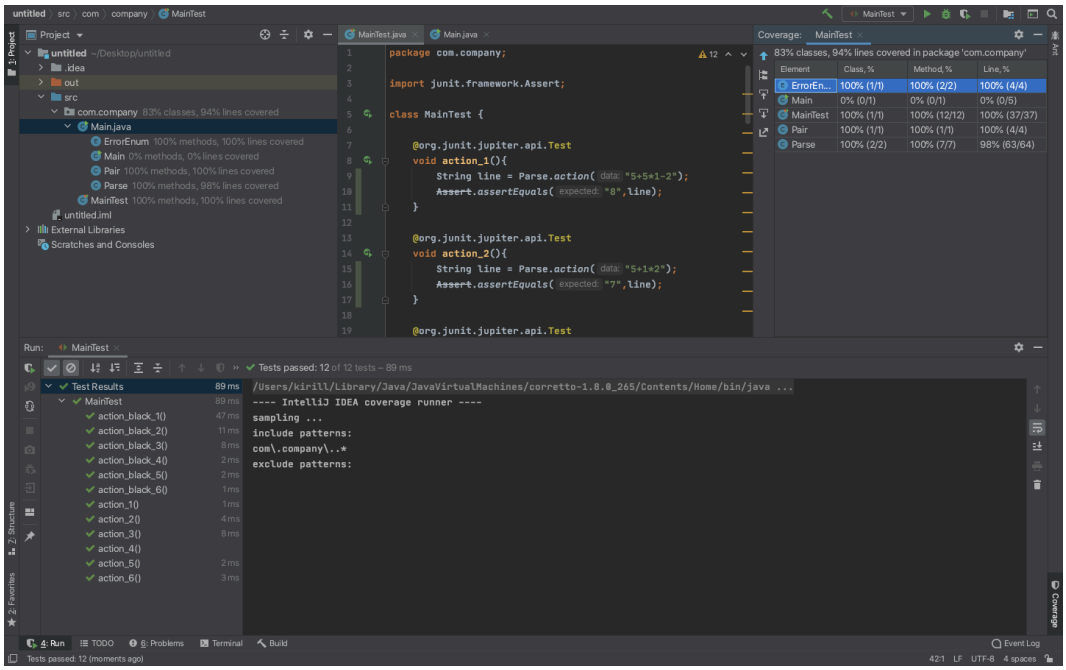
1 Оценка покрытия функции action тестами, разработанными в лабораторной работе №2 (методом черного ящика).



2) Оценка покрытия функции action тестами, разработанными в лабораторной работе №3 (методом белого ящика).



3) Оценка покрытия функции action тестами, разработанными в лабораторных работах №2 и №3.



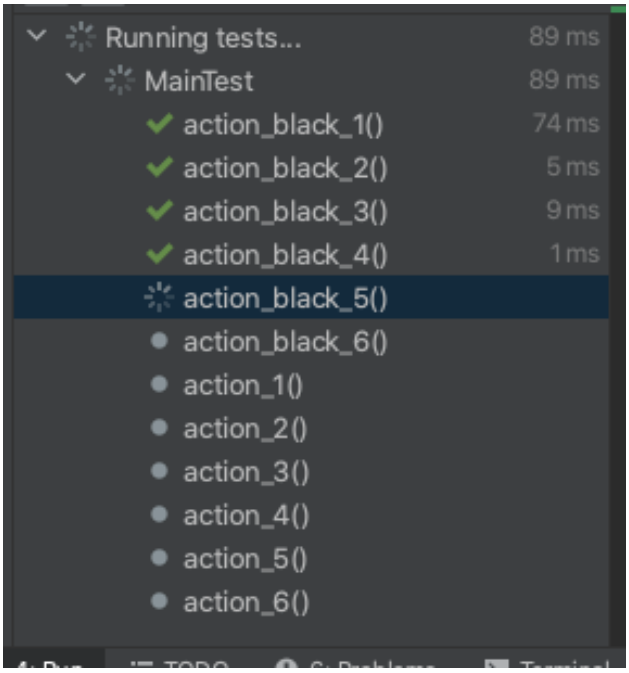
6. Инъекция багов:

1) Изменим условие перехода к следующей итерации в цикле при проверке данных в строке.

```
if (isNumeric(data.charAt(i)) | String.valueOf(data.charAt(i)).equals("+")
        | String.valueOf(data.charAt(i)).equals("*") | String.valueOf(data.charAt(i)).equals("/") |
        String.valueOf(data.charAt(i)).equals("-")){
    continue;
```

Изменим на:

```
if (isNumeric(data.charAt(i)) | String.valueOf(data.charAt(i)).equals("+")
        | String.valueOf(data.charAt(i)).equals("*") | String.valueOf(data.charAt(i)).equals("/") |
        String.valueOf(data.charAt(i)).equals("-")){
    break;
```



Тесты action_black_5 не будут выполнены, поскольку данные будут зациклены т.к условие выхода из перехода к следующей итерации было нарушено и программа вышла из цикла.

7. Выводы:

В ходе данной лабораторной работы были получены навыки по функциональному тестированию кода методом белого ящика (всех ветвей), а также по инъекции багов и оценке выполнения при этом тестов.