НАЦИОНАЛЬНО-ИССЛЕДОВАТЕЛЬСКАЯ КОРПОРАЦИЯ ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1 по предмету:

“Тестирование программного обеспечения”

Выполнили студенты:

Стуков Егор Александрович

Нестеров Иван Алексеевич

Группа № P33302

Преподаватель:

Харитонова Анастасия Евгеньевна

г. Санкт-Петербург

2023

Задание:

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели

Вариант: 9052

1. Функция sin(x)
2. Программный модуль для работы с хеш-таблицей с разрешением коллизий методом цепочек (Hash String, http://www.cs.usfca.edu/~galles/visualization/BucketSort.html)
3. Описание предметной области:
   Через несколько тысяч лет, когда их галактика уже лежала в руинах, они поняли, в конце концов, что все это было ужасной ошибкой, и тогда оба враждующих боевых флота объединили свои остатки с тем, чтобы совершить совместное нападение на нашу Галактику, положительно определенную как источник обидной фразы.

Выполнение:

https://github.com/nesterrovv/software-testing

1. Тестирование разложения функции в степенной ряд:

Реализация алгоритма:

```
package com.nesterrovv.sinePowerSeries;

import com.nesterrovv.exceptions.IncorrectTermsAmountExceptions;

public class SinePowerSeriesCalculator {

    private static double calculateFactorial(double functionArgument) {
        double temporaryResult = 1;
        for (int i = 1; i <= functionArgument; i++) {
            temporaryResult *= i;
        }
        return temporaryResult;
    }
    public static double calculateSinePowerSeries(double functionArgument) {
        /*
        taking into account the periodicity of the sine, we take the remainder of
the division of the angle by 360
        and translate this angle into radians
         */
        double x = Math.PI / 180 * (functionArgument % 360); //
        // preparations
        double member;
        double sum = 0;
        double tinyValue = 1e-15;
        double sign = 1;
        double power = x;
        double factorial = 1;
        double multiplier = 1;
        // get sum of the series
        do {
            // calculating current member of the series
            member = sign * power / factorial;
            // Appending to sum
            sum += member;
            // preparing next step
            sign *= -1;
            multiplier++;
            factorial *= multiplier;
            multiplier++;
            factorial *= multiplier;
            power *= x * x;
        } while (Math.abs(member) > tinyValue);
        return sum;
    }

}
```

Тестирование алгоритма:

```
package com.nesterrovv.sinePowerSeries;

import com.nesterrovv.sinePowerSeries.SinePowerSeriesCalculator;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class SinePowerSeriesCalculatorTest  {

    double standardPositiveArgument;
    double standardNegativeArgument;
    double zeroArgument;
    double bigPositiveArgument;
    double bigNegativeArgument;
    int standardPositiveTermsAmount;
    int bigPositiveTermsAmount;
    int zeroTermsAmount;

    int oneArgument;
    int minusOneArgument;
    int negativeTermsAmount;

    final double epsilon = 1e-11;

    @Before
    public void initializeData() {
        this.standardPositiveArgument = 42;
        this.standardNegativeArgument = -42;
        this.zeroArgument = 0;
        this.bigPositiveArgument = 1234567;
        this.bigNegativeArgument = -1234567;
        this.standardPositiveTermsAmount = 100;
        this.zeroTermsAmount = 0;
        this.negativeTermsAmount = -42;
    }

    @Test
    public void checkViaStandardPositiveArgument() {
        double expectedResult = 0.66913060635;
        Assert.assertEquals(expectedResult,
                SinePowerSeriesCalculator
                        .calculateSinePowerSeries(standardPositiveArgument),
epsilon);
    }

    @Test
    public void checkViaStandardNegativeArgument() {
        double expectedResult  = -0.66913060635;
        Assert.assertEquals(expectedResult,

SinePowerSeriesCalculator.calculateSinePowerSeries(standardNegativeArgument),
epsilon);
    }

    @Test
    public void checkViaZeroArgument() {
```

```
        double expectedResult  = 0;
        Assert.assertEquals(expectedResult,
                SinePowerSeriesCalculator.calculateSinePowerSeries(zeroArgument),
epsilon);
    }

    @Test
    public void checkViaBigPositiveArgument() {
        double expectedResult  = 0.79863551004;
        Assert.assertEquals(expectedResult,

SinePowerSeriesCalculator.calculateSinePowerSeries(bigPositiveArgument), epsilon);
    }

    @Test
    public void checkViaBigNegativeArgument() {
        double expectedResult  = -0.79863551004;
        Assert.assertEquals(expectedResult,

SinePowerSeriesCalculator.calculateSinePowerSeries(bigNegativeArgument), epsilon);
    }

}
```

2. Тестирование реализации алгоритма BucketSort:

Реализация алгоритма:

```
package com.nesterrovv.bucketSort;

import java.util.*;

public class BucketSort {
    public static void sort(double[] array) {

        if (array == null)
            return;
        double[] positiveNumbers = Arrays.stream(array).filter(e -> e >=
0).toArray();
        double[] negativeNumbers = Arrays.stream(array).filter(e -> e < 0).map(e ->
-e).toArray();

        sortPositive(negativeNumbers);
        sortPositive(positiveNumbers);

        for (int i = 0; i < negativeNumbers.length; i++) {
            array[i] = -negativeNumbers[negativeNumbers.length - 1 - i];
        }

        if (positiveNumbers.length > 0)
            System.arraycopy(positiveNumbers, 0, array, negativeNumbers.length,
positiveNumbers.length);
    }

    private static void sortPositive(double[] array) {

        //noinspection unchecked
        List<Double>[] buckets = new LinkedList[array.length];

        for (int i = 0; i < array.length; i++) {
            buckets[i] = new LinkedList<>();
        }

        var maxOpt = Arrays.stream(array).max();

        if (maxOpt.isEmpty()) {
            return;
        }

        double maxArrayValue = maxOpt.getAsDouble();

        for (double curElement : array) {
            int index = (int) (curElement * array.length / (maxArrayValue + 1));
            var curList = buckets[index];

            boolean appended = false;
            for (int j = 0; j < curList.size(); j++) {
                if (curElement <= curList.get(j)) {
                    curList.add(j, curElement);
                    appended = true;
                    break;
```

```
                }
            }
            if (!appended)
                curList.add(curElement);
        }

        int elementIndex = 0;
        for (List<Double> curList : buckets) {
            for (double e : curList) {
                array[elementIndex] = e;
                elementIndex++;
            }
        }
    }
}
```

Модульное тестирование алгоритма:

```
package com.nesterrovv.bucketSort;

import com.nesterrovv.bucketSort.BucketSort;
import org.junit.Assert;
import org.junit.Test;

public class BucketSortTest {

    private final double delta = 1e-10;
    @Test
    public void sortZeroToOne() {
        double[] testArray = {0.84, 0.777, 0.99, 0.124, 0.02, 0.004, 0.123};
        double[] expectedArray = {0.004, 0.02, 0.123, 0.124, 0.777, 0.84, 0.99};

        BucketSort.sort(testArray);
        Assert.assertArrayEquals(expectedArray, testArray, delta);

    }
    @Test
    public void sortEmptyArray() {
        double[] testArray = {};
        double[] expectedArray = {};

        BucketSort.sort(testArray);
        Assert.assertArrayEquals(expectedArray, testArray, delta);
    }
    @Test
    public void sortNullArray() {
        double[] testArray = null;
        double[] expectedArray = null;

        BucketSort.sort(testArray);
        Assert.assertEquals(expectedArray, testArray);
    }
    @Test
    public void sortEqualValues() {
        double[] testArray = {0, 0, 0, 0};
        double[] expectedArray = {0, 0, 0, 0};

        BucketSort.sort(testArray);
```

```
        Assert.assertArrayEquals(expectedArray, testArray, delta);
    }
    @Test
    public void sortZeroToPositive() {
        double[] testArray = {734, 245, 2345, 32, 1_000_000_000, 503, 1, 0.055};
        double[] expectedArray = {0.055, 1, 32, 245, 503, 734, 2345,
1_000_000_000};

        BucketSort.sort(testArray);
        Assert.assertArrayEquals(expectedArray, testArray, delta);
    }
    @Test
    public void sortNegativeToZero() {
        double[] testArray = {-1, -5, -7, -13, -234, 0, 0};
        double[] expectedArray = {-234, -13, -7, -5, -1, 0, 0};

        BucketSort.sort(testArray);
        Assert.assertArrayEquals(expectedArray, testArray, delta);
    }
    @Test
    public void sortSmallValues() {
        double[] testArray = {1e-1, 1e-5, 1e-7, 1e-3, 1e-9};
        double[] expectedArray = {1e-9, 1e-7, 1e-5, 1e-3, 1e-1};

        BucketSort.sort(testArray);
        Assert.assertArrayEquals(expectedArray, testArray, delta);
    }
    @Test
    public void sortNegativeToPositive() {
        double[] testArray = {1000, -203, 0, 4, -0.05, 1, 0, -87};
        double[] expectedArray = {-203, -87, -0.05, 0, 0, 1, 4, 1000};

        BucketSort.sort(testArray);
        Assert.assertArrayEquals(expectedArray, testArray, delta);
    }
}
```

3. Тестовое покрытие для доменной модели

Разработанная доменная модель располагается в пакете `com.nesterovv.pojosModel`

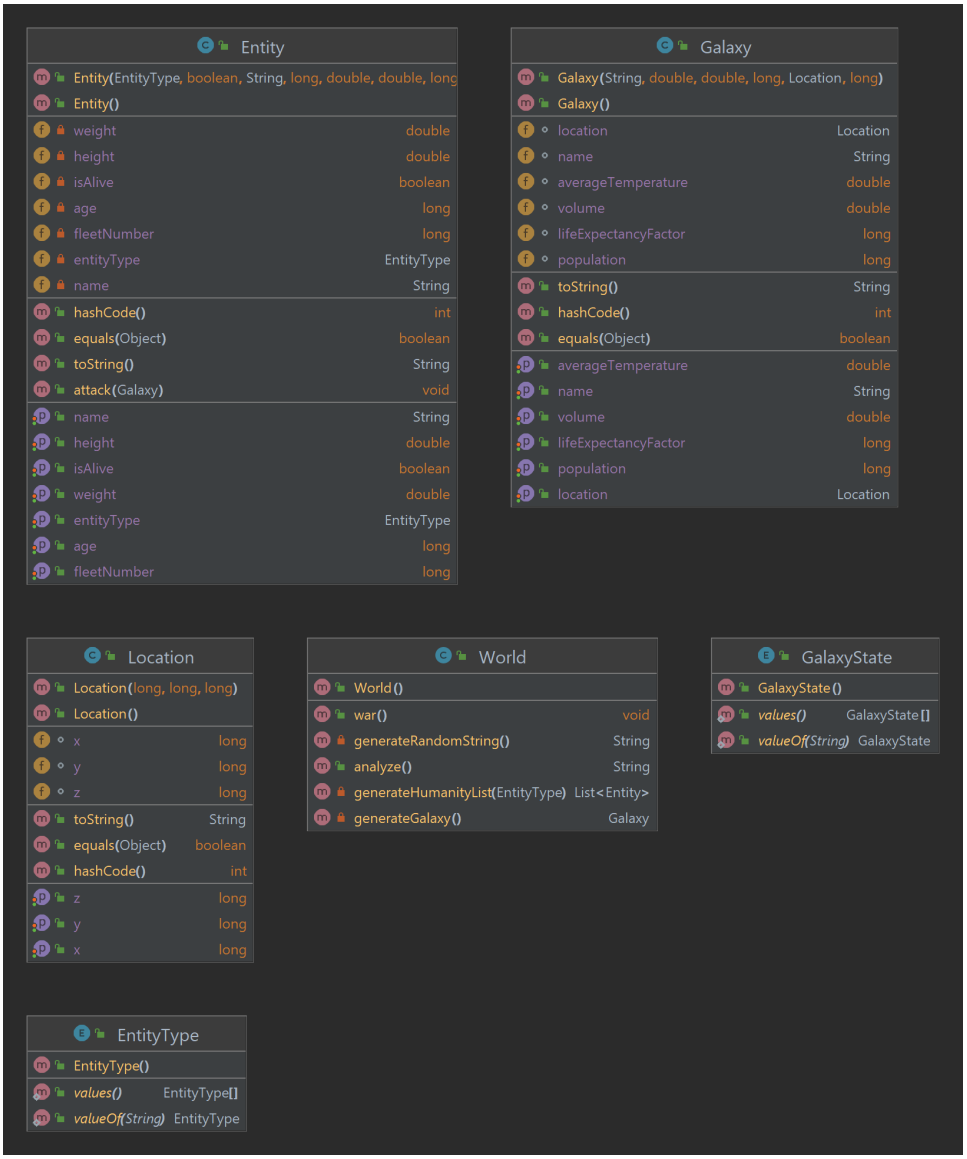

Рисунок 1. UML-диаграмма разработанной доменной модели.

Тестирование доменной модели:

```
package com.nesterrovv.pojosModel;

import org.junit.Assert;
import org.junit.Test;

public class PojosModelTest {

    @Test
    public void checkAttack() {
        Galaxy sample = new Galaxy();
```

```
        sample.setLifeExpectancyFactor(200);
        new Entity().attack(sample);
        long expected = 100;
        Assert.assertEquals(expected, sample.getLifeExpectancyFactor());
    }

    @Test
    public void checkFirstCaseOfAnalyze() {
        Galaxy first = new Galaxy();
        first.setLifeExpectancyFactor(100);
        Galaxy second = new Galaxy();
        second.setLifeExpectancyFactor(200);
        String test = new World().analyze();
        String expected = "Retribution awaits the galaxy 2";
        Assert.assertEquals(expected, test);
    }

    @Test
    public void checkSecondCaseOfAnalyze() {
        Galaxy first = new Galaxy();
        first.setLifeExpectancyFactor(200);
        Galaxy second = new Galaxy();
        second.setLifeExpectancyFactor(100);
        String test = new World().analyze();
        String expected = "Retribution awaits the galaxy 1";
        Assert.assertEquals(expected, test);
    }

    @Test
    public void checkThirdCaseOfAnalyze() {
        Galaxy first = new Galaxy();
        first.setLifeExpectancyFactor(200);
        Galaxy second = new Galaxy();
        second.setLifeExpectancyFactor(200);
        String test = new World().analyze();
        String expected = "Retribution awaits the galaxy 2";
        Assert.assertEquals(expected, test);
    }

}
```

Выводы:

В ходе выполнения данной лабораторной работы было проведено модульное тестирование для трех программ: функции разложения синуса в ряд Тейлора, алгоритма BucketSort и кода доменной модели, составленной по описанию предметной области.
В ходе составления тестов были подобраны тестовые данные, содержащие не только классы эквивалентности, но и граничные случаи.
Была разработана доменная модель, с учетом предоставляемого ею функционала пользователю было разработано тестовое покрытие.