МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Разработка набора классов для работы с табулированными функциями»

по курсу
Объектно-ориентированное программирование

Выполнил: Петрухин Роман,
студент группы 6203-010302D

Оглавление

Задание №1

Пакет «functions» в отличие от предыдущей лабораторной работы я создал в IDE «Intellij IDEA».

Задание №2

Далее следуя ходу лабораторной работы я создал класс «FunctionPoint». В нем я создал два поля «private double x» и «private double y». Далее в соответствии с заданием в классе были созданы конструкторы: FunctionPoint(double x, double y), FunctionPoint(FunctionPoint point) и FunctionPoint(), также для удобства были написаны методы геттеры-сеттеры для полей данного класса. Реализацию конструкторов представлена на рисунке 1.

```
public FunctionPoint(double x, double y) {
    this.x = x;
    this.y = y;
}


FunctionPoint(FunctionPoint point) {  no us
    this.x = point.getX();
    this.y = point.getY();
}

FunctionPoint() {  no usages   zhestok1
    this.x = 0;
    this.y = 0;
}
```

Рисунок 1

Задание №3

После выполнения задания №2 я создал класс «TabulatedFunction». В качестве его полей я создал массив «massiveOfPoints» типа FunctionPoint[] и счетчик элементов массива типа int «amountOfElemnets». Далее были созданы 2 конструктора данного класса. Реализация которых представлена на рисунке 2.

```
public TabulatedFunction(double leftX, double rightX, int pointsCount) {  1 usage  zhestok1
    this.amountOfElements = pointsCount;
    this.massiveOfPoints = new FunctionPoint[Math.max(amountOfElements, 0)]; // Избегаю от

    if (amountOfElements == 1) {
        this.massiveOfPoints[0] = new FunctionPoint(((leftX + rightX) / 2),  y: 0);
    }

    else {
        double distance = (rightX - leftX) / (amountOfElements - 1); // Поиск интервала
        for(int i = 0; i < amountOfElements; i++) {
            double x = leftX + i * distance;
            this.massiveOfPoints[i] = new FunctionPoint(x,  y: 0);
        }
    }
}

public TabulatedFunction(double leftX, double rightX, double[] values) {  no usages  zhestok
    this.amountOfElements = values.length;
    this.massiveOfPoints = new FunctionPoint[Math.max(amountOfElements, 0)];
    if (amountOfElements == 1) {
        this.massiveOfPoints[0] = new FunctionPoint(((leftX + rightX) / 2), values[0]);
    }
    else {
        double distance = (rightX - leftX) / (amountOfElements - 1); // Поиск интервала
        for(int i = 0; i < amountOfElements; i++) {
            double x = leftX + i * distance;
            this.massiveOfPoints[i] = new FunctionPoint(x, values[i]);
        }
    }
}
```

Рисунок 2

Интересным местом в реализации данных конструкторов можно считать лишь по-моему скромному мнению задание размера массива. Изначально вместо использованной функции max() предназначение которой в том, чтобы избежать получения отрицательных размеров массива, а соответственно и ошибок из-за этого я использовал функцию abs(), но потом меня осенило и я понял, что могут возникнуть некоторые логические проблемы с вводом данных (в данном случае все отрицательные = некорректные данные станут корректными, хотя должны быть признаны ошибочными).

Задание №4

Первые две функции описанные в условии задание №4 полностью повторяют написание стандартных геттеров и ничего интересного в их реализации нет. Третий же метод «getFunctionValue» я постарался выполнить так как описано в его условии, его реализация представлена на рисунке 3.

```java
public double getFunctionValue(double x) {  3 usages  zhestok1
    // Поиск интервала, содержащего x
    for (int i = 0; i < amountOfElements - 1; i++) {
        double x1 = massiveOfPoints[i].getX();
        double x2 = massiveOfPoints[i + 1].getX();

        if (x >= x1 && x <= x2) {
            // Линейная интерполяция между точками i и i+1
            double y1 = massiveOfPoints[i].getY();
            double y2 = massiveOfPoints[i + 1].getY();

            // Уравнение прямой: y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
            return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
        }
    }

    // Если x точно совпадает с последней точкой
    if (x == massiveOfPoints[amountOfElements - 1].getX()) {
        return massiveOfPoints[amountOfElements - 1].getY();
    }

    return Double.NaN;
}
```

Рисунок 3

Никогда не думал, что нахождение среднего значения путем проведения прямой в определенном интервале будет называться **интерполяцией**. В этой функции я нахожу нужный мне промежуток, значения функции на краях этого промежутка и по формуле линейной интерполяции нахожу нужное мне значение функции в данной точке.

Задание №5

В задании под номером 5 написание функций с частичкой «get» или «set» практически ничем не отличается от стандартных геттеров-сеттеров. Единственное отличие - это наличие проверки на возможность проведения «сета» указанной точки. Реализация проверки приведена на рисунке 4.

```java
if (index <= 0 || index > amountOfElements || point == null) return;
if (index > 0 && point.getX() <= massiveOfPoints[index - 1].getX()) {
    return;
}
if (index < amountOfElements - 1 && point.getX() >= massiveOfPoints[index + 1].getX()) {
    return;
}  // Проверки на сохранение очередности X координат
```

Рисунок 4

Первый if проверяет положительность index и нахождение его внутри нашего интервала, второй if проверяет что у нашей точки координата больше, чем у точки слева, ну и третий if проверяет, что координата нашей точки меньше, чем координата точки справа.

Задание №6

В 6 задании я сделал функции «deletePoint» и «addPoint», функцию «deletePoint» я реализовал через цикл for, то есть за точку отсчета счетчика я взял i = index, и до конца отрезка я просто сдвигал элементы влево. Таким образом я никак не менял массив, но выполнил задание. Полная реализация представлена на рисунке 5.

```
public void deletePoint(int index) {  1 usage   zhestok1
    if (amountOfElements <= 2 || index < 0 || index >= amountOfElements) {
        return;
    }

    // Сдвигаем все элементы после удаляемого влево
    for (int i = index; i < amountOfElements - 1; i++) {
        massiveOfPoints[i] = massiveOfPoints[i + 1];
    }
    massiveOfPoints[amountOfElements - 1] = null;
    amountOfElements--;
}
```

Рисунок 5

Функцию «addPoint» я реализовал практически также, единственным дополнением послужило увеличение массива, думаю в данном случае это не будет осудительным действом. Реализация данной функции представлена на рисунке 6.

Рисунок 6 на следующей странице

```java
public void addPoint(FunctionPoint point) {  1 usage  zhestok1
    if (point == null) {
        return;
    }

    // Увеличиваем массив если нужно
    if (amountOfElements == massiveOfPoints.length) {
        FunctionPoint[] newPoints = new FunctionPoint[massiveOfPoints.length * 2 + 1];
        for (int i = 0; i < amountOfElements; i++) {
            newPoints[i] = massiveOfPoints[i];
        }
        massiveOfPoints = newPoints;
    }

    // Ищем место для вставки
    int pos = 0;
    while (pos < amountOfElements && massiveOfPoints[pos].getX() < point.getX()) {
        pos++;
    }

    // Проверяем на дубликат
    if (pos < amountOfElements && massiveOfPoints[pos].getX() == point.getX()) {
        return;
    }

    // Сдвигаем элементы вправо
    for (int i = amountOfElements; i > pos; i--) {
        massiveOfPoints[i] = massiveOfPoints[i - 1];
    }

    // Вставляем новую точку
    massiveOfPoints[pos] = new FunctionPoint(point.getX(), point.getY());
    amountOfElements++;
}
}
```

Рисунок 6

Описание же функций я сделал в формате представленном на рисунке 7.

```
/**
 * Добавляет точку по индексу
 * @param point новая точка
 */
```

Рисунок 7

Задание №7

В качестве примера я использовал функцию sin(x). Большую часть функционала данных классов я продемонстрировал на ней. Результаты работы кода продемонстрированы на рисунка 8,9.

```
=== Изменение точки с индексом 1 ===
После изменения:
Точка 0: x=0,00, y=0,00
Точка 1: x=1,00, y=0,50
Точка 2: x=2,09, y=0,87
Точка 3: x=3,14, y=0,00
Точка 4: x=4,19, y=-0,87
Точка 5: x=5,24, y=-0,87
Точка 6: x=6,28, y=-0,00

=== Интерполяция значений ===
sin(1,20) ≈ 0,5669
sin(2,80) ≈ 0,2825
sin(4,10) ≈ -0,7926

=== Границы области определения ===
Левая граница: 0,00
Правая граница: 6,28

=== Изменение координат по отдельности ===
До изменения:
Точка 0: x=0,00, y=0,00
После изменения:
Точка 0: x=0,50, y=0,30

=== Итоговое состояние функции ===
Точка 0: x=0,50, y=0,30
Точка 1: x=1,00, y=0,50
Точка 2: x=2,09, y=0,87
Точка 3: x=3,14, y=0,00
Точка 4: x=4,19, y=-0,87
Точка 5: x=5,24, y=-0,87
Точка 6: x=6,28, y=-0,00
Всего точек: 7
```

Рисунок 8

```
=== Функция sin(x) ===
Исходная функция:
Точка 0: x=0,00, y=0,00
Точка 1: x=1,05, y=0,87
Точка 2: x=2,09, y=0,87
Точка 3: x=3,14, y=0,00
Точка 4: x=4,19, y=-0,87
Точка 5: x=5,24, y=-0,87
Точка 6: x=6,28, y=-0,00

=== Добавление точки (1.5, 0.9) ===
После добавления:
Точка 0: x=0,00, y=0,00
Точка 1: x=1,05, y=0,87
Точка 2: x=1,50, y=0,90
Точка 3: x=2,09, y=0,87
Точка 4: x=3,14, y=0,00
Точка 5: x=4,19, y=-0,87
Точка 6: x=5,24, y=-0,87
Точка 7: x=6,28, y=-0,00

=== Удаление точки с индексом 2 ===
После удаления:
Точка 0: x=0,00, y=0,00
Точка 1: x=1,05, y=0,87
Точка 2: x=2,09, y=0,87
Точка 3: x=3,14, y=0,00
Точка 4: x=4,19, y=-0,87
Точка 5: x=5,24, y=-0,87
Точка 6: x=6,28, y=-0,00
```

Рисунок 9