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

Выполнила: Гонтарь Анастасия Вячеславовна,
студент группы 6203-010302D

Содержание

Задание №1

Для выполнения этого задания я вручную создала папку functions

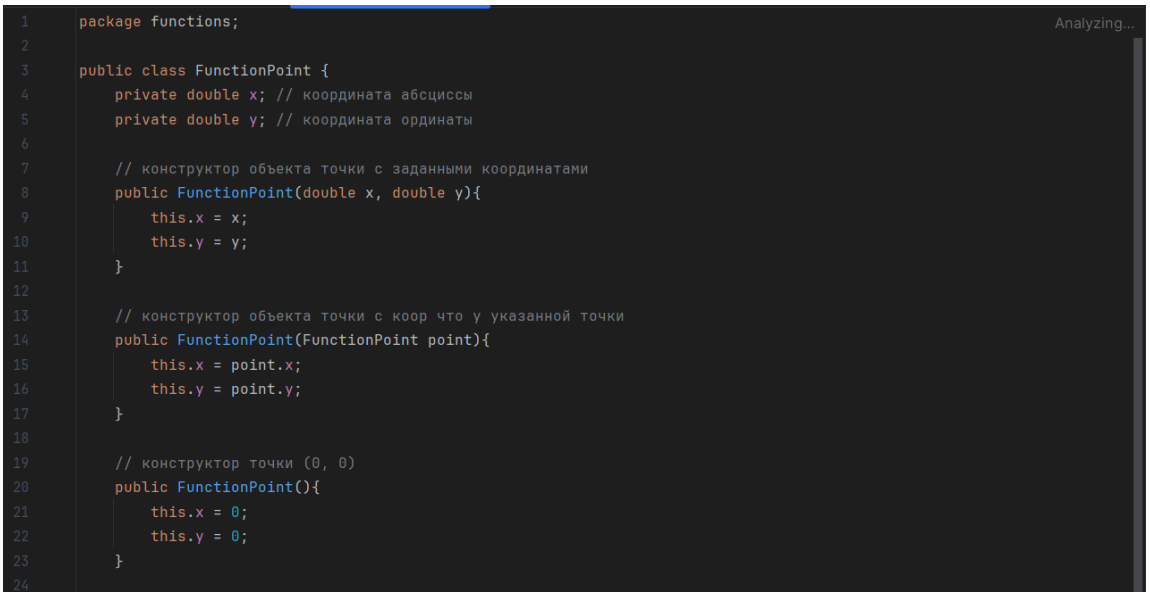
Задание №2

Для выполнения этого задания, я в пакете *functions* создала класс *FunctionPoint*, объект которого описывает одну точку табулированной функции. В классе описаны приватные поля: координата точки по оси абсцисс (x) и координата точки по оси ординат (y), а также следующие конструкторы:

• *FunctionPoint(double x, double y)* – создаёт объект точки с заданными координатами;

• *FunctionPoint(FunctionPoint point)* – создаёт объект точки с теми же координатами, что у указанной точки;

• *FunctionPoint()* – создаёт точку с координатами (0; 0).




```java
package functions;

public class FunctionPoint {
    private double x; // координата абсциссы
    private double y; // координата ординаты

    // конструктор объекта точки с заданными координатами
    public FunctionPoint(double x, double y){
        this.x = x;
        this.y = y;
    }

    // конструктор объекта точки с коор что у указанной точки
    public FunctionPoint(FunctionPoint point){
        this.x = point.x;
        this.y = point.y;
    }

    // конструктор точки (0, 0)
    public FunctionPoint(){
        this.x = 0;
        this.y = 0;
    }

}
```

Также учитывая особенность инкапсуляции, я описала “геттеры” и “сеттеры”

```
    // геттеры
    public double getX() { return x; }

    public double getY() { return y; }

    // сеттеры
    public void setX(double x) { this.x = x; }

    public void setY(double y) { this.y = y; }
}
```

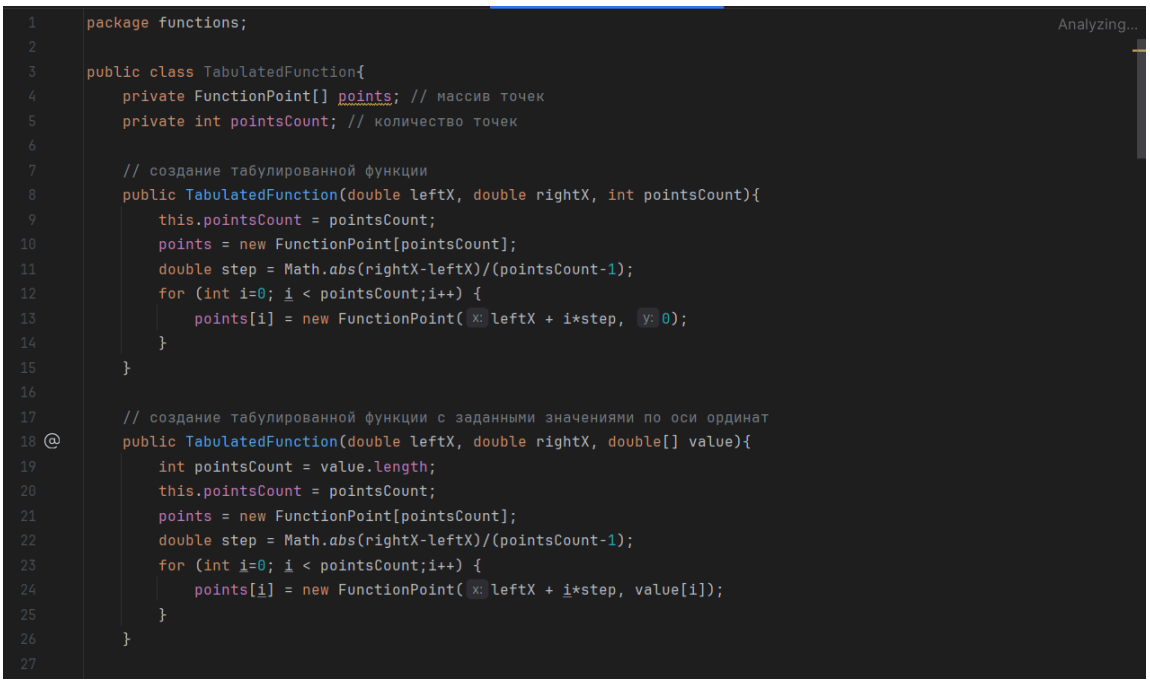
Задание №3

Для выполнения этого задания я в пакете functions создала класс *TabulatedFunction*, объект которого описывает табулированную функцию. Для хранения данных о точках используется массив типа *FunctionPoint*, для хранения информации о количестве точек используется переменная *pointsCount*.

В классе описаны следующие конструкторы:

• *TabulatedFunction(double leftX, double rightX, int pointsCount)* – создаёт объект табулированной функции по заданным левой и правой границе области определения, а также количеству точек для табулирования

• *TabulatedFunction(double leftX, double rightX, double[] values)* – аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива.



```java
package functions;

public class TabulatedFunction{
    private FunctionPoint[] points; // массив точек
    private int pointsCount; // количество точек

    // создание табулированной функции
    public TabulatedFunction(double leftX, double rightX, int pointsCount){
        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = Math.abs(rightX-leftX)/(pointsCount-1);
        for (int i=0; i < pointsCount;i++) {
            points[i] = new FunctionPoint( x: leftX + i*step,  y: 0);
        }
    }

    // создание табулированной функции с заданными значениями по оси ординат
    public TabulatedFunction(double leftX, double rightX, double[] value){
        int pointsCount = value.length;
        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = Math.abs(rightX-leftX)/(pointsCount-1);
        for (int i=0; i < pointsCount;i++) {
            points[i] = new FunctionPoint( x: leftX + i*step, value[i]);
        }
    }

```

Задание №4

Для выполнения этого задания в классе *TabulatedFunction* я описала методы, необходимые для работы с функцией:

• Метод *double getLeftDomainBorder()* , возвращающий значение левой границы области определения табулированной функции.

• Метод *double getRightDomainBorder()*, возвращающий значение правой границы области определения табулированной функции.

• Метод *double getFunctionValue(double x)* , возвращающий значение функции в точке x, если эта точка лежит в области определения функции. В противном случае метод должен возвращать значение неопределённости (Nan). Значение вычисляется при помощи уравнения прямой, проходящей через 2 точки

```java
// возвращение левой границы области определения
public double getLeftDomainBorder() { return points[0].getX(); }

// возвращение правой границы области определения
public double getRightDomainBorder() { return points[this.pointsCount - 1].getX(); }

// возвращение y, если точка лежит в области определения функции
public double getFunctionValue(double x){
    double leftX = getLeftDomainBorder();
    double rightX = getRightDomainBorder();

    if (x>= leftX && x <= rightX){
        // ищем соседние точки
        for (int i = 0; i < this.pointsCount - 1; i++) {
            double x1 = points[i].getX();
            double x2 = points[i + 1].getX();

            if (x >= x1 && x <= x2) {
                if (x == x1) return points[i].getY();
                if (x == x2) return points[i + 1].getY();

                double y1 = points[i].getY();
                double y2 = points[i + 1].getY();
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
        }

    }
    return Double.NaN;
}
```

Задание №5

Для выполнения этого задания в классе *TabulatedFunction* я описала методы, необходимые для работы с точками табулированной функции:

• Метод *int getPointsCount()*, возвращающий количество точек.

• Метод *FunctionPoint getPoint(int index)* , возвращающий копию точки, соответствующей переданному индексу.

• Метод *void setPoint(int index, FunctionPoint point)* , заменяющий указанную точку табулированной функции на переданную.

• Метод *double getPointX(int index)* , возвращающий значение абсциссы точки с указанным номером.

• Метод *void setPointX(int index, double x)* , изменяющий значение абсциссы точки с указанным номером.

• Метод *double getPointY(int index)* , возвращающий значение ординаты точки с указанным номером.

• Метод *void setPointY(int index, double y)* , изменяющий значение ординаты точки с указанным номером.

```
64        // возвращения количества точек
65        public int getPointsCount(){  1 usage
66            return pointsCount;
67        }
68
69        // возвращение точки
70        public FunctionPoint getPoint(int index) {  no usages
71            return new FunctionPoint(points[index]);
72        }
73
74        // изменение указанной точки на заданную
75        public void setPoint (int index, FunctionPoint point){  4 usages
76            if ((index < 0 || index >= pointsCount) || ( index != 0 && point.getX() <= points[index - 1].getX()) ||
77                    (index != pointsCount-1 &&  point.getX() >= points[index + 1].getX()))
78
79                return;
80
81            points[index]= new FunctionPoint(point);
82
83        }
84
85        // возвращение координаты x
86        public double getPointX(int index){  1 usage
87            return points[index].getX();
88        }
89
```

```
// изменение значения x
public void setPointX(int index, double x){  no usages
    if ((index < 0 || index >= pointsCount) || ( index != 0 && x <= points[index - 1].getX()) ||
            (index != pointsCount-1 &&  x >= points[index + 1].getX()))
        return;

    points[index].setX(x);
}

// возвращения координаты y
public double getPointY (int index){  no usages
    return points[index].getY();
}

// изменения значения y
public void setPointY(int index, double y){  1 usage
    if (index < 0 || index >= pointsCount)
        return;

    points[index].setY(y);
}
```

Задание №6

Для выполнения этого задания в классе *TabulatedFunction* я описала методы, изменяющие количество точек табулированной функции:

• Метод *void deletePoint(int index)* удаляет заданную точку, если корректен заданный индекс. При помощи метода arraycopy часть массива после

точки, которую нужно удалить, переносится копированием на одну позицию влево, тем самым точка удаляется

• Метод *void addPoint(FunctionPoint point)* добавляет новую точку табулированной функции. Алгоритм схож с удалением, только часть массива копируется на позицию вправо, а в получившийся промежуток записывается копия заданной точки. В случае если массив переполнен, то размер массива увеличивается в 2 раза, чтобы избежать многократного создания нового массива.

```
112         // удаление точки
113         public void deletePoint(int index){  2 usages
114             if (index>= 0 && index < pointsCount) {
115                 System.arraycopy(points,  srcPos: index + 1, points, index,  length: pointsCount - index - 1);
116                 points[--pointsCount] = null;
117             }
118             return;
119         }
120
121         // добавление точки
122 @       public void addPoint(FunctionPoint point) {  1 usage
123             int index= 0;
124
125             double newX = point.getX();
126             // находим позицию для вставки точки
127             while (index < pointsCount && newX > points[index].getX())
128                 index++;
129
130             // проверяем нужно ли увеличивать размер массива
131             if (pointsCount >= points.length) {
132                 FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
133                 System.arraycopy(points,  srcPos: 0, newPoints,  destPos: 0, pointsCount);
134                 points = newPoints;
135             }
136
137             if (index < pointsCount)
138                 System.arraycopy(points, index, points,  destPos: index + 1,  length: pointsCount - index);
139
140             points[index] = new FunctionPoint(point);
141             pointsCount++;
142
```

Задание №7

Для удобного вывода я описала метод *outFunction* в классе *TabulatedFunction*

```
    // вывод массива точек
    public void outFunction(){  8 usages
        for (int i = 0; i<pointsCount; i++)
            System.out.printf("(%.2f, %.2f) ", points[i].getX(), points[i].getY());

        System.out.println();
    }
```

В пакете по умолчанию я создала класс *Main*, содержащий точку входа программы.В методе main() создала экземпляр класса *TabulatedFunction* и задала для него табулированные значения функции *y=1.2*x* на интервале [1, 5.6].

Я проверила работу написанных классов:

1) Попробовала удалить точку с корректным индексом и некорректным, выходящим за пределы массива. Программа повела себя ожидаемо, удалив только первую точку
2) Добавила точку и проверила на какую позицию она встала. Результат получился ожидаемым
3) Попробовала заменить существующую точку на заданную. Результат ожидаемый: точка не принадлежит промежутку интерполяции и замена не была проведена. Также я проверила, что будет если попытаться заменить точку, индекс которой выходит за пределы массива, получила ожидаемый результат и не замененную точку. Следующие 2 точки подошли - произошла замена.
4) Проверила корректность работы метода, который возвращает значение точки, если она лежит в области определения заданной функции. Я задала интервал, часть которого принадлежит этой область, а другая часть напротив. Ожидаемый результат: для части, не принадлежащей области определения, значения точек не были вычислены и были записаны как *Nan*.

```java
import functions.*;

public class Main {
    public static void main(String[] arg){
        TabulatedFunction func = new TabulatedFunction( leftX: 1,  rightX: 5.6,  pointsCount: 8);
        for (int i =0; i < func.getPointsCount(); i++)
            func.setPointY(i,  y: func.getPointX(i)*1.2);

        System.out.println("Функция: y=1.2x");
        func.outFunction();

        System.out.println("Проверка удаления существующей точки");
        func.deletePoint( index: 3);
        func.outFunction();

        System.out.println("Проверка удаления несуществующей точки");
        func.deletePoint( index: 8);
        func.outFunction();

        System.out.println("Проверка добавления точки");
        func.addPoint(new FunctionPoint( x: 3.0,  y: 4.0));
        func.outFunction();

        System.out.println("Проверка установки некорректной для интерполяции точки");
        func.setPoint( index: 2, new FunctionPoint( x: 1.2,  y: 5));
        func.outFunction();

        System.out.println("Проверка установки некорректной для интерполяции точки");
        func.setPoint( index: -3, new FunctionPoint( x: -10,  y: 5));
        func.outFunction();

        System.out.println("Проверка установки корректной для интерполяции точки");
        func.setPoint( index: 0, new FunctionPoint( x: 1.3,  y: 5));
        func.outFunction();

        System.out.println("Проверка установки корректной для интерполяции точки");
        func.setPoint( index: 2, new FunctionPoint( x: 2.2,  y: 5));
        func.outFunction();

        System.out.println("Проверка getFunctionValue");
        for (double i = -1; i < 3; i += 0.8) {
            System.out.printf("(%.2f, %.2f) ", i, func.getFunctionValue(i));
            System.out.println();
        }

    }
}
```

```
PS C:\Users\nastya\JavaProjects\Lab-2-2025> javac Main.java
PS C:\Users\nastya\JavaProjects\Lab-2-2025> java Main
Функция: y=1.2x
(1,00, 1,20) (1,66, 1,99) (2,31, 2,78) (2,97, 3,57) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка удаления существующей точки
(1,00, 1,20) (1,66, 1,99) (2,31, 2,78) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка удаления несуществующей точки
(1,00, 1,20) (1,66, 1,99) (2,31, 2,78) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка добавления точки
(1,00, 1,20) (1,66, 1,99) (2,31, 2,78) (3,00, 4,00) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка установки некорректной для интерполяции точки
(1,00, 1,20) (1,66, 1,99) (2,31, 2,78) (3,00, 4,00) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка установки некорректной для интерполяции точки
(1,00, 1,20) (1,66, 1,99) (2,31, 2,78) (3,00, 4,00) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка установки корректной для интерполяции точки
(1,30, 5,00) (1,66, 1,99) (2,31, 2,78) (3,00, 4,00) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка установки корректной для интерполяции точки
(1,30, 5,00) (1,66, 1,99) (2,20, 5,00) (3,00, 4,00) (3,63, 4,35) (4,29, 5,14) (4,94, 5,93) (5,60, 6,72)
Проверка getFunctionValue
(-1,00, NaN)
(-0,20, NaN)
(0,60, NaN)
(1,40, 4,16)
(2,20, 5,00)
PS C:\Users\nastya\JavaProjects\Lab-2-2025>
```