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

Работу выполнила:

студент группы 6201-120303D,

Нагуманова В. Э.

Самара 2025

Содержание

Задание №2.

Для выполнения этого задания я создала класс FunctionPoint. В нем я создала два private поля: private doble x и private double y.
В соответствии с заданием я создала три конструктора:

- 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).

```
public class FunctionPoint {  1 usage
    private double x;  5 usages
    private double y;  5 usages

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

    public FunctionPoint(FunctionPoint point)
    {
        this.x = point.getX();
        this.y= point.getY();
    }

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

Рис. 1

В этом классе я создала методы геттеры-сеттеры getX(), getY(), которые возвращают и меняют значения x и y, для того чтобы иметь возможность доступа к значениям полей private.

```
public double getX(){  1 usage
    return x;
}
public double getY(){  1 usage
    return y;
}

public void setX(double x){  no usages
    this.x = x;
}
public void setY(double y){  no usages
    this.y = y;
}
```

Рис. 2

Задание №3.

Для этого задания в пакете functions создаю публичный класс TabulatedFunction, в котором описана табулированная функция. Приватными значениями будет являться массив точек points типа FunctionPoint[] и счетчик элементов массива int pointsCount.

Следуя заданию, в классе реализованы два конструктора:

-TabulatedFunction(double leftX, double rightX, int pointsCount) – для создания объекта функции по заданным левой и правой границам области определения, а также по количеству точек;

- TabulatedFunction(double leftX, double rightX, double[] values) – для создания функции по левой и правой границам области определения и массиву значений.

```java
package functions;

public class TabulatedFunction {  no usages
    private FunctionPoint[] points; //массив точек, задающих функцию  4 usages
    private int pointsCount;  2 usages

    //создаёт объект табулированной функции по заданным левой и правой границам области определения, а также количеству точек для табулирован
    public TabulatedFunction(double leftX, double rightX, int pointsCount) {  no usages
        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = Math.abs(rightX - leftX) / (pointsCount - 1); //шаг между точками
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint( x: leftX + i * step,  y: 0);
        }
    }

    //создает объект табулированной функции по заданным левой и правой границам области определения, а также значениям функции
    public TabulatedFunction(double leftX, double rightX, double[] values) {  no usages
        int pointsCount = values.length;
        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = Math.abs(rightX - leftX) / (pointsCount - 1); //интервал (шаг между точками)
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint( x: leftX + i * step, values[i]);
        }
    }
}
```

Рис. 3

Задание №4.

В классе TabulatedFunction описаны методы:

- double getLeftDomainBorder() – возвращает значение левой границы области определения табулированной функции;

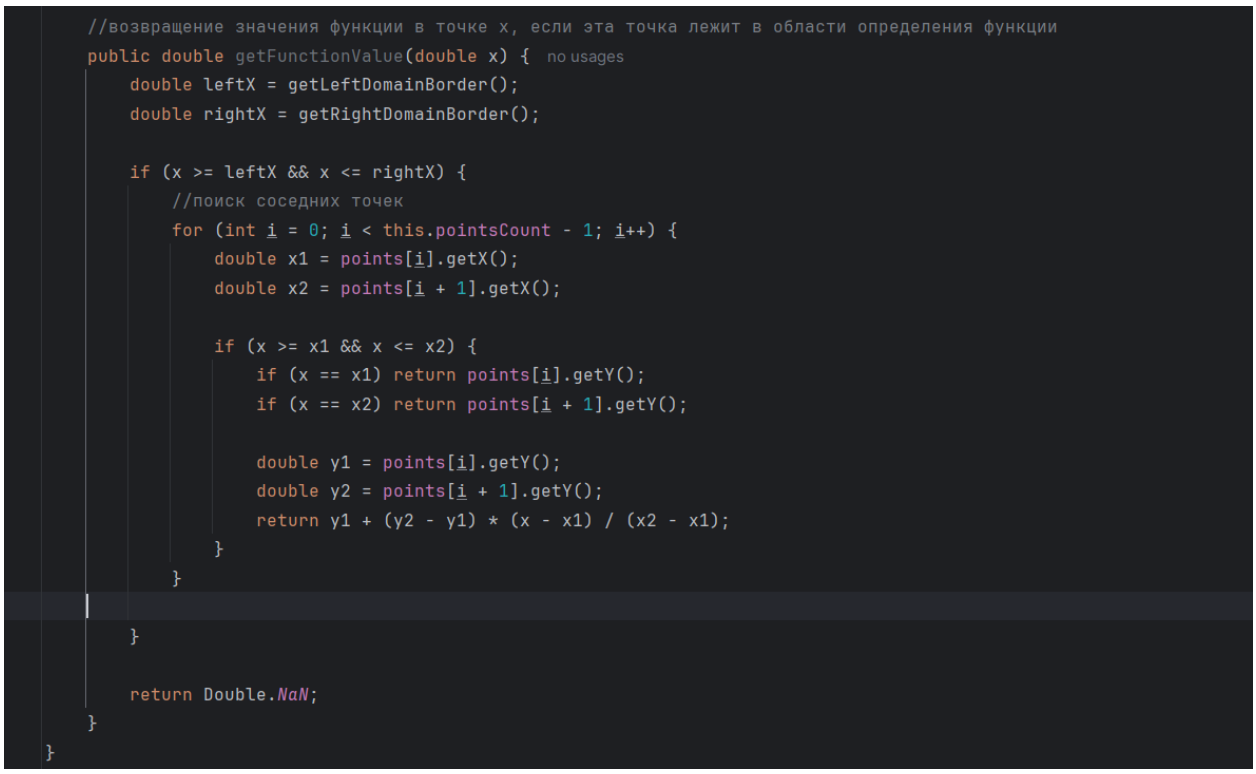
- double getRightDomainBorder() – возвращет значение правой границы области определения табулированной функции;

- double getFunctionValue(double x) – возвращает значение функции в точке x, если эта точка лежит в области определения функции. По заданию при расчете значения используется линейная интерполяция и уравнением прямой, проходящей через две точки. Если точка не принадлежит интервалу, возвращаем значение NaN.

```java
public double getLeftDomainBorder() {  1 usage
    return points[0].getX(); //возвращение левой границы области опрделения
}

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

Рис. 4



```
    //возвращение значения функции в точке x, если эта точка лежит в области определения функции
    public double getFunctionValue(double x) {  no usages
        double leftX = getLeftDomainBorder();
        double rightX = getRightDomainBorder();

        if (x >= leftX && x <= rightX) {
            //поиск соседних точек
            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;
    }
}
```

Рис. 5

Задание №5.

Для этого задания, я описала методы в этом же классе TabulatedFunction, необходимые для работы с точками табулированной функции.

- Метод int getPointsCount() – возвращает количество точек;

- Метод FunctionPoint getPoint(int index) – возвращает копию точки, соответствующей переданному индексу;

- Метод void setPoint(int index, FunctionPoint point) – заменяет указанную точку табулированной функции на переданную;

- Метод double getPointX(int index) – возвращает значение абсциссы точки с указанным номером;

- Метод void setPointX(int index, double x) – изменяет значение абсциссы точки с указанным номером;

- Метод double getPointY(int index) – возвращает значение ординаты точки с указанным номером;

- Метод void setPointY(int index, double y) – изменяет значение ординаты точки с указанным номером;

```
//возвращения количество точек
public int getPointsCount(){  no usages
    return pointsCount;
}

//возвращает копию точки по указанному индексу
public FunctionPoint getPoint(int index) {  no usages
    return new FunctionPoint(points[index]);
}

//заменяет указанную точку табулированной функции на переданную
public void setPoint(int index, FunctionPoint point){  no usages
    if ((index < 0 || index >= pointsCount) || ( index != 0 && point.getX() <= points[index - 1].getX())
            ||  (index != pointsCount-1 &&  point.getX() >= points[index + 1].getX()))
    return;

    points[index]= new FunctionPoint(point);

}
```

Рис. 6

```
//возвращает значение абсциссы точки с указанным номером
public double getPointX(int index){  no usages
    return points[index].getX();
}

//измененяет 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);
}

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

//изменяет значения y
public void setPointY(int index, double y){  no usages
    if (index < 0 || index >= pointsCount)
        return;

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

Рис. 7

Задание №6.

Для выполнения этого задания, в классе TabulatedFunction добавляю методы для изменения количества точек табулированной функции.

- Метод void deletePoint(int index) - удаляет заданную точку табулированной функции. Для реализации я использовала метод arraycope(), часть массива, после точки, которую необходимо удалить, сдвигается на одну позици влево, путем копирования, в результате точка удаляется. Первым параметром является исходный массив, вторым – индекс начальной позиции в исходном массиве, откуда начинается копирование, третьим – массив, в который копируются элементы, четвертый параметр – начальная позиция массива, куда будут вставляться элементы, пятый параметр – количество элементов для копирования;

- Метод void addPoint(FunctionPoint point) – добавляет новую точку табулированной функции. Аналогично удалению, часть массива, путем копирования, сдвигается на одну позицию **вправо**, и в промежуток записывается копия заданной точки. Если массив переполнен, то его размер увеличивается в два раза (для того, чтобы избежать постоянного создания нового массив

```
//удаление точки по указанному индексу
public void deletePoint(int index){  no usages
    if (index < pointsCount) {
        System.arraycopy(points, srcPos: index + 1, points, index, length: pointsCount - index - 1);
        points[--pointsCount] = null;
    }
    return;
}
```

Рис. 8

```java
//добавление новой точки в массив с сохранением порядка по X
public void addPoint(FunctionPoint point) {  no usages
    int index= 0;

    double newX = point.getX();
    //находим позицию для вставки точки
    while (index < pointsCount && newX > points[index].getX())
        index++;
    //проверка для увелечение массива
    if (pointsCount >= points.length) {
        FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, pointsCount);
        points = newPoints;
    }

    if (index < pointsCount)
        System.arraycopy(points, index, points, destPos: index + 1, length: pointsCount - index);

    points[index] = new FunctionPoint(point);
    pointsCount++;

}
```

Рис 9.

Задание №7.

Для последнего задания я прописала класс Main в пакете, который содержит точку входа программы.

В методе main() создаю экземпляр класса TabulatedFunction и задаю значение значение табулированной функции y =2x +2 на интервале [-3.5, 5]

Также для более удобного вывода, в классе TabulatedFunction я реализовала метод printFunc(TabulatedFunction function), для последовательного вывода значений точек переданной функции.

```java
import functions.*;

public class Main {
        public static void main(String[] arg){
            TabulatedFunction func = new TabulatedFunction( leftX: -3.5,  rightX: 5,  pointsCount: 5);
            for (int i=0; i< func.getPointsCount(); i++)
                func.setPointY(i,  y: func.getPointX(i)*2 +2);

            System.out.println("\nФункция: y = 2x + 2 на отрезке [-3.5, 5] из "+func.getPointsCount()+" точек.");
            func.printFunc();

            System.out.println("\nУдаление существующей точки");
            func.deletePoint( index: 4);
            func.printFunc();

            System.out.println("Удаление несуществующей точки");
            func.deletePoint( index: 11);
            func.printFunc();

            System.out.println("\nДобавление точки");
            func.addPoint(new FunctionPoint( x: 1.5,  y: 5));
            func.printFunc();

            System.out.println("\nУстановка некорректной точки для интерполяции");
            func.setPoint( index: 1, new FunctionPoint( x: -0.5,  y: 3));
            func.printFunc();

            System.out.println("Установка некорректной точки для интерполяции");
            func.setPoint( index: 3, new FunctionPoint( x: -2.7,  y: -1));
            func.printFunc();
```

Рис.10

```java
        System.out.println("\nУстановка корректной точки для интерполяции");
        func.setPoint( index: 1, new FunctionPoint( x: -3,  y: 3));
        func.printFunc();

        System.out.println("Установка корректной точки для интерполяции");
        func.setPoint( index: 3, new FunctionPoint( x: 1,  y: 3));
        func.printFunc();

        System.out.println("\ngetFunctionValue");
        for(double i = -5; i < 1; i ++) {
            System.out.printf("(%.2f, %.2f) ", i, func.getFunctionValue(i));
            System.out.println();
        }

    }
```

Рис.11

Пример работы кода:

```
Функция: y = 2x + 2 на отрезке [-3.5, 5] из 5 точек.
(-3,50, -5,00) (-1,38, -0,75) (0,75, 3,50) (2,88, 7,75) (5,00, 12,00)

Удаление существующей точки
(-3,50, -5,00) (-1,38, -0,75) (0,75, 3,50) (2,88, 7,75)
Удаление несуществующей точки
(-3,50, -5,00) (-1,38, -0,75) (0,75, 3,50) (2,88, 7,75)

Добавление точки
(-3,50, -5,00) (-1,38, -0,75) (0,75, 3,50) (1,50, 5,00) (2,88, 7,75)

Установка некорректной точки для интерполяции
(-3,50, -5,00) (-0,50, 3,00) (0,75, 3,50) (1,50, 5,00) (2,88, 7,75)
Установка некорректной точки для интерполяции
(-3,50, -5,00) (-0,50, 3,00) (0,75, 3,50) (1,50, 5,00) (2,88, 7,75)

Установка корректной точки для интерполяции
(-3,50, -5,00) (-3,00, 3,00) (0,75, 3,50) (1,50, 5,00) (2,88, 7,75)
Установка корректной точки для интерполяции
(-3,50, -5,00) (-3,00, 3,00) (0,75, 3,50) (1,00, 3,00) (2,88, 7,75)

getFunctionValue
(-5,00, NaN)
(-4,00, NaN)
(-3,00, 3,00)
(-2,00, 3,13)
(-1,00, 3,27)
(0,00, 3,40)
```

Рис. 12