Лабораторная работа №2
Болясников Вадим Андреевич 6204-010302D

Задание на лабораторную работу:
Разработать набор классов для работы с функциями одной переменной, заданными в табличной форме.

Задание 1:
В этом задание я создал пакет functions, в котором будут храниться наши классы.

Задание 2:
Я создал класс FunctionPoint внутри пакета functions, данный класс описывает одну точку табулированной функции.
В нем я реализовал следующие контрукции:
• 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).
Также чтобы учесть особенности инкапсуляции, я сделал геттеры и сеттеры.

```java
public class FunctionPoint {
    public double x;
    public double y;

    public FunctionPoint(double x, double y){
        this.x = x;
        this.y = y;
    }
    public FunctionPoint(FunctionPoint point){
        this.x = point.x;
        this.y = point.y;
    }
    public FunctionPoint(){
        this.x = 0.0;
        this.y = 0.0;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

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

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

}
```

Рисунок 1 – Класс FunctionPoint

Задание 3:
В этом же пакете function я реализовал класс TabulatedFunction, который описывал табулированную функцию, для хранения данных о точке использовался массив типа FunctionPoint. Также было сделано упорядочение по значение x.
Также были реализованы 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 (значения функции в точках при этом следует считать равными 0);
• TabulatedFunction(double leftX, double rightX, double[] values) – аналогичен предыдущему конструктору, но вместо количества точек получает значения функции в виде массива.

```
public class TabulatedFunction {
    public FunctionPoint[] points;
    public int pointsCount;

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

        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x, values[i]);
        }
```

Рисунок 2 – Реализация класса TabulatedFunction

Задание 4:

В этом задание я дополнил класс TabulatedFunction, дописал следующие функции:
• Метод double getLeftDomainBorder() возвращает значение левой границы области определения табулированной функции.
• Аналогично, метод double getRightDomainBorder() возвращает значение правой границы области определения табулированной функции.
• Метод double getFunctionValue(double x) возвращает значение функции в точке x, если эта точка лежит в области определения функции. В противном случае метод возвращает значение неопределённости. Для которой была реализована дополнительная функция linearInterpolation.

```java
public double getLeftDomainBorder() {
    return points[0].getX();
}
public double getRightDomainBorder() {
    return points[pointsCount - 1].getX();
}
public double linearInterpolation(FunctionPoint p1, FunctionPoint p2, double x) {
    double x1 = p1.getX();
    double y1 = p1.getY();
    double x2 = p2.getX();
    double y2 = p2.getY();

    return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
}
public double getFunctionValue(double x){
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
        return Double.NaN;
    }
    for(int i = 0; i < pointsCount - 1; i++){
        double curX = points[i].getX();
        double nextX = points[i+1].getX();
        if(x == curX){
            return points[i].getY();
        }
        if(x > curX && x < nextX){
            return linearInterpolation(points[i], points[i+1], x);
        }
        if(x == nextX){
            return points[i+1].getY();
        }
    }
    return points[pointsCount - 1].getY();
}
```

Рисунок 3 – Доп. методы в классе TabulatedFunction

Задание 5-6.

Я ходе этих двух заданий я дописал еще следующие методы:
• Метод int getPointsCount() который возвращает текущее количество точек в табулированной функции.

• Метод FunctionPoint getPoint(int index) который возвращает копию точки по указанному индексу. Создание копии обеспечивает инкапсуляцию, предотвращая модификацию исходной точки извне класса.
• Метод void setPoint(int index, FunctionPoint point) который заменяет точку по указанному индексу на переданную точку. Метод включает проверки для сохранения упорядоченности точек по координате X.

• Метод double getPointX(int index) который возвращает координату X точки с указанным индексом.

• Метод void setPointX(int index, double x) который изменяет координату X точки с указанным индексом. Как и в setPoint(), метод проверяет, что новое значение X не нарушает упорядоченность точек.

• Метод double getPointY(int index) который возвращает координату Y точки с указанным индексом.

• Метод void setPointY(int index, double y) который изменяет координату Y точки с указанным индексом.

• Метод void deletePoint(int index) удаляет точку с указанным индексом. Для эффективного удаления используется System.arraycopy() для сдвига элементов массива, после удаления последний элемент обнуляется для помощи сборщику мусора.

• Метод void addPoint(FunctionPoint point) Добавляет новую точку в табулированную функцию с сохранением упорядоченности по X

```
public double getFunctionValue(double x){
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
        return Double.NaN;
    }
    for(int i = 0; i < pointsCount - 1; i++){
        double curX = points[i].getX();
        double nextX = points[i+1].getX();
        if(x == curX){
            return points[i].getY();
        }
        if(x > curX && x < nextX){
            return linearInterpolation(points[i], points[i+1], x);
        }
        if(x == nextX){
            return points[i+1].getY();
        }
    }
    return points[pointsCount - 1].getY();
}


public int getPointsCount() {
    return pointsCount;
}
public FunctionPoint getPoint(int index){
    return new FunctionPoint(points[index]);
}

public void setPoint(int index, FunctionPoint point){
    if(index > 0 && points[index-1].getX() < point.getX()){
        return;
    }
    if(index < pointsCount - 1 && points[index+1].getX() > point.getX()){
        return;
    }
    points[index] = new FunctionPoint(point);
}

public double getPointX(int index){
    return points[index].getX();
}

public void setPointX(int index, double x){
    if(index > 0 && points[index-1].getX() < x){
        return;
    }
    if(index < pointsCount - 1 && points[index+1].getX() > x){
        return;
    }
    points[index].setX(x);
}

public double getPointY(int index){
    return points[index].getY();
}

public void setPointY(int index, double y){
    points[index].setY(y);
}
```

```
public void deletePoint(int index){
    if (pointsCount - 1 - index >= 0) {
        System.arraycopy(points, index + 1, points, index, pointsCount - 1 - index);
    }
    pointsCount--;
    points[pointsCount] = null;
}

public void addPoint(FunctionPoint point) {
    FunctionPoint newPoint = new FunctionPoint(point);
    double newX = newPoint.getX();

    // Проверяем, не существует ли уже точка с таким X
    for (int i = 0; i < pointsCount; i++) {
        if (Math.abs(points[i].getX() - newX) < 1e-10) {
            // Точка с таким X уже существует - заменяем ее
            points[i] = newPoint;
            return;
        }
    }

    // Создаем новый массив на 1 элемент больше
    FunctionPoint[] newArray = new FunctionPoint[pointsCount + 1];

    // Находим позицию для вставки
    int insertIndex = 0;
    while (insertIndex < pointsCount && points[insertIndex].getX() < newX) {
        insertIndex++;
    }

    // Копируем элементы до позиции вставки с помощью System.arraycopy
    if (insertIndex > 0) {
        System.arraycopy(points, srcPos:0, newArray, destPos:0, insertIndex);
    }

    // Вставляем новую точку
    newArray[insertIndex] = newPoint;

    // Копируем элементы после позиции вставки с помощью System.arraycopy
    if (pointsCount - insertIndex > 0) {
        System.arraycopy(points, insertIndex, newArray, insertIndex + 1, pointsCount - insertIndex);
    }

    // Заменяем старый массив новым
    points = newArray;
    pointsCount++;
}
```

Рисунок 4 – Еще методы в классе TabulatedFunction.

Задание 7.

Был создан main вне пакета function. Там я написал экземпляр класса TabulatedFunction который задает табулированные значения функции f(x) = x^2. А также написал вывод в консоль значения функций на ряде точек. Вывел результат программы после: удаления, добавления и изменения точек.

Вот что вывелось у меня в консоль:

1. СОЗДАНИЕ ФУНКЦИИ f(x) = x^2 на [0, 4]
Функция f(x) = x^2
Количество точек: 5
Область определения: [0.0, 4.0]

2. ВЫЧИСЛЕНИЕ ЗНАЧЕНИЙ ФУНКЦИИ
f(x) = не определено (вне области)
f(x) = 0.0
f(x) = 0.5
f(x) = 1.0
f(x) = 2.5
f(x) = 4.0
f(x) = 6.5
f(x) = 9.0
f(x) = 12.5
f(x) = 16.0
f(x) = не определено (вне области)

3. ИЗМЕНЕНИЕ СУЩЕСТВУЮЩИХ ТОЧЕК
После setPointY(2, 10.0): f(2.0) = 10.0
После setPoint(3, (3.5; 20.0)): f(3.5) = 12.5

4. ДОБАВЛЕНИЕ НОВЫХ ТОЧЕК

После addPoint((-1.0; 1.0)): точек = 6, f(-1.0) = 1.0
После addPoint((1.5; 2.25)): f(1.5) = 2.25
После addPoint((5.0; 25.0)): точек = 8, f(5.0) = 25.0
После addPoint((2.0; 100.0)) - замена: functions.TabulatedFunction@1722011b
f(2.0) = 100.0

5. УДАЛЕНИЕ ТОЧЕК
После deletePoint(0): точек = 7, левая граница = 0.0
После deletePoint(2): точек = 6
После deletePoint(5): точек = 5, правая граница = 4.0