Лабораторная работа №2. ООП.

Выполнил лабораторную работу:
Гордеев Максим Дмитриевич
Группа: 6204-010302D

Цель лабораторной работы: Разработать набор классов для работы с функциями одной переменной, заданными в табличной форме.

Ход выполнения лабораторной работы

Задание 2:

Создадим класс **FunctionPoint**, который имеет два поля, обозначающие координаты **X** и **Y:**

```java
package functions;

public class FunctionPoint{

    private double coorX, coorY; // Координаты X, Y

    public FunctionPoint(double x, double y){ // Конструктор с двумя параметрами
        this.coorX = x;
        this.coorY = y;
    }

    public FunctionPoint(FunctionPoint point){ // Коструктор с аргументом точки
        this.coorX = point.getCoorX();
        this.coorY = point.getCoorY();
    }

    public FunctionPoint(){ // Конструктор по умолчанию
        this.coorX = 0;
        this.coorY = 0;
    }
    // Сеттерый
    public void setCoorX(double x){this.coorX = x;}
    public void setCoorY(double y) {this.coorY = y;}

    // Геттеры
    public double getCoorX(){return this.coorX;}
    public double getCoorY() {return this.coorY;}
}
```

Класс имеет три конструктора: с параметрами, по умолчанию и конструктор, параметром которого является сам объект класса **FunctionPoint**. А также в классе есть геттеры и сеттеры для координат.

Задание 3:

Создадим класс табулированной функции **TabulatedFunction**:

```java
public class TabulatedFunction {
    private double leftDomainBorder, rightDomainBorder; // Левая и правая границы

   private FunctionPoint[] points; // Массив точек
   private int size;

    public TabulatedFunction(double leftX, double rightX, int pointsCount){ // Конструктор с количеством точек

        this.leftDomainBorder = leftX; // Левая граница области определения
        this.rightDomainBorder = rightX; // Правая граница области определения
        this.points = new FunctionPoint[pointsCount]; // Массив точек (FunctionPoint)
        this.size = pointsCount;

        double delta = (rightX - leftX) / (pointsCount - 1); // Считаем дельту, размер каждого из отрезков
        for (int i = 0; i < pointsCount-1; ++i){ // Инициализируем значения точек в цикле кроме последнего
            points[i] = new FunctionPoint(leftX + (i * delta), y:0);
        }
        points[pointsCount-1] = new FunctionPoint(rightX, y:0);
    }

    public TabulatedFunction(double leftX, double rightX, double[] values){ // Конструктор с значениями функций
        this.leftDomainBorder = leftX;
        this.rightDomainBorder = rightX;
        this.points = new FunctionPoint[values.length];
        this.size = values.length;

        double delta = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length - 1; ++i) { // В цикле инициализируем значения точек, но уже вместе с y
            points[i] = new FunctionPoint(leftX + (i * delta), values[i]);
        }
        points[values.length- 1] = new FunctionPoint(rightX, values[values.length - 1]);
    }
```

- Поля **leftDomainBorder** и **rightDomainBorder** являются границами области определения функции.
- Массив **FunctionPoint[]** хранит в себе точки табулированной функции.
- Поле **size** – текущий размер массива.

Класс имеет два конструктора, первый конструктор в качестве параметров принимает **leftX** – левая граница области определения, **rightX** – правая граница области определения, **pointsCount –** количество точек. Второй конструктор имеет такие же два параметра, как и первый, но в качестве третьего параметра, принимает список значений функции **values[].** С помощью цикла и расчёта значения **delta**, мы получаем равные отрезки по числовой оси **x.**

Задание 4:

В классе **TabulatedFunction** опишем необходимые методы для работы с функцией:

```java
// Гет методы
public double getLeftDomainBorder(){return leftDomainBorder;}
public double getRightDomainBorder() {return rightDomainBorder;}


// Метод для получения прямой по двум точкам
private double interpolate(double x, double x1, double y1, double x2, double y2){
    return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
}

// Метод для получения значения функции
public double getFunctionValue(double x){
    if (x < this.leftDomainBorder || x > this.rightDomainBorder){ // Если x выходит из диапозона возращаем Double.NaN
        return Double.NaN;
    }


    for (int i = 0; i < size - 1; ++i){ // Перебараем в цикле все отрезки интервала

        if (x >= points[i].getCoorX() && x < points[i+1].getCoorX()){ // Если x входит в определённый отрезок
            return interpolate(x, points[i].getCoorX(), points[i].getCoorY(), points[i+1].getCoorX(), points[i+1].getCoorY());
        }
    }

    if (x == points[size-1].getCoorX()){ // Если x является правой границей интервала
        return points[size-1].getCoorY();
    }

    return Double.NaN;
}
```

Добавили два новых метода **getLeftDomainBorder()** и **getRightDomainBorder()** для получения значений границ области определения. Метод **interpolate()** отвечает за получение значения прямой, построенной по двум точкам. Метод **getFunctionValue(double x)** определяет между какими двумя точками по оси x находится **x** и вызывает метод **interpolate()** для найденных точек.

Задание 5:

Теперь в классе **TabulatedFunction** опишем методы, необходимые для работы с точками табулированной функции:

```java
67      public int getPointsCount(){ // Возвращает количество точек
68          return size;
69      }
70
71      public FunctionPoint getPoint(int index) throws IllegalArgumentException{
72
73          if (index < 0 || index >=size){ // Если не подходит под условие выбрасываем исключение
74              throw new IllegalArgumentException(s:"Выход за пределы массива");
75          }
76
77          FunctionPoint pointCopy = new FunctionPoint(points[index].getCoorX(), points[index].getCoorY()); // Создаём копию, чтобы не нарушать
78          return pointCopy;
79      }
80
81      public void setPoint(int index, FunctionPoint point) throws IllegalArgumentException{
82          if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
83              throw new IllegalArgumentException(s:"Выход за пределы массива");
84          }
85          if (index > 0 && point.getCoorX() < points[index-1].getCoorX()){
86              throw new IllegalArgumentException(s:"x Нарушает порядок,выберите другой индекс");
87          }
88          if (index < size - 1 && point.getCoorX() > points[index + 1].getCoorX()){
89              throw new IllegalArgumentException(s:"x Нарушает порядок,выберите другой индекс");
90          }
91          points[index].setCoorX(point.getCoorX());
92          points[index].setCoorY(point.getCoorY());
93      }
94
95      public double getPointX(int index) throws IllegalArgumentException{
96          if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
97              throw new IllegalArgumentException(s:"Выход за пределы массива");
98          }
99          return points[index].getCoorX();
100     }
101
```

```
101
102     public double getPointY(int index) throws IllegalArgumentException {
103         if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
104             throw new IllegalArgumentException(s:"Выход за пределы массива");
105         }
106         return points[index].getCoorY();
107     }
108
109     public void setPointX(int index, double x) throws IllegalArgumentException{
110         if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
111             throw new IllegalArgumentException(s:"Выход за пределы массива");
112         }
113
114         if (index > 0 && points[index-1].getCoorX() > x ){
115             throw new IllegalArgumentException(s:"x Нарушает порядок,выберите другой индекс");
116         }
117         if (index < size-1 && points[index+1].getCoorX() < x){
118             throw new IllegalArgumentException(s:"x Нарушает порядок,выберите другой индекс");
119         }
120             points[index].setCoorX(x);
121     }
122
123     public void setPointY(int index, double y) throws IllegalArgumentException {
124         if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
125             throw new IllegalArgumentException(s:"Выход за пределы массива");
126         }
127         points[index].setCoorY(y);
128     }
```

- Метод **getPointCount()** возвращает текущее количество точек в массиве **FunctionPoint[]**.
- Метод **getPoint(int index)** в качестве параметра принимает индекс точки и возвращает копию объекта этой точки.
- Метод **setPoint(int index, FunctionPoint point)** в качестве параметров принимает индекс и объект класса **FunctionPoint**. Меняет свойства(координаты) элемента массива **FunctionPoint[]** по индексу **index**, на свойства точки **point.** Проверяет изменение координаты **X** на нарушение порядка, так как координаты точек по **X** должны идти в порядке возрастания.
- Метод **getPointX(int index)** возвращает значение координаты **X** элемента, взятого из массива **FunctionPoint[]** по индексу **index**. Аналогично **getPoinY(int index)**.
- Метод **setPointX(int index, double x)** почти аналогичен методу **setpoint(int index, FunctionPoint point),** он изменяет значение только координаты **X**.
- Метод **setPoinY(int index, double y)** устанавливает значение координаты Y точки, взятой из массива **FunctionPoint[]** по индексу **index**.

Задание 6:

Опишем методы в классе **TabulatedFunction** , изменяющие количество точек табулированной функции.

```java
public void deletePoint(int index) throws IllegalArgumentException{
    if (index < 0 || index >= size) { // Если не подходит под условие выбрасываем исключение
        throw new IllegalArgumentException(s:"Выход за пределы массива");
    }

    System.arraycopy(points, index + 1, points, index, size - index - 1);
    points[size-1] = null;
    --size;


    if (size > 0) { // Обнавляем границы после удаления
        this.leftDomainBorder = points[0].getCoorX();
        this.rightDomainBorder = points[size - 1].getCoorX();
    }
}
```

Метод **deletePoint(int index)** удаляет элемент из массива **FunctionPoint[] points** по индексу **index**. Удаление осуществляется за счёт копирования элементов, находящихся справа от элемента с индексом **index**(не включая элемент с индексом **index**), при этом поле **size**, которое контролирует текущее количество элементов уменьшается на 1. После удаления необходимо обновить границы области определения.

```java
public void addPoint(FunctionPoint point) throws IllegalArgumentException{

    final int k = 2; // Коэфф. расширения длины массива

    double point_x = point.getCoorX();
    int point_idx = size; // Индекс новой точки // По умолчанию в конец

    if (point_x < leftDomainBorder){ // Проверяем расширяет ли точка диапазон
        point_idx = 0;
    }else{
        if (point_x > rightDomainBorder){
            point_idx = size;
        }
    }

    for (int i = 0; i < size; ++i){
        if (point_x == points[i].getCoorX()){ // Если точка по x совпадает с другой точкой
            throw new IllegalArgumentException(s:"Точка должна иметь уникальное значение x");
        }
    }

    for (int i = 0; i < size - 1; ++i){ // Если точка принадлежит интервалу
        if ( points[i].getCoorX() < point_x && point_x < points[i+1].getCoorX()){ // Если входит в один из отрезков интервала
            point_idx = i + 1;
            break;
        }
    }

    if (points.length <= size) { // Проверяем хватает ли длины массива, если нет расширяем массив на 1
        FunctionPoint[] new_points = new FunctionPoint[k*points.length];

        System.arraycopy(points, srcPos:0, new_points, destPos:0, point_idx);
        new_points[point_idx] = new FunctionPoint(point);
        System.arraycopy(points, point_idx, new_points, point_idx+1, size - point_idx);

        points = new_points;
    }else{
        System.arraycopy(points, point_idx, points, point_idx + 1, size - point_idx);
        points[point_idx] = new FunctionPoint(point);

    }
    size++;
    if (point_idx == 0) { // обновляем границы
        this.leftDomainBorder = point_x;
    }
    if (point_idx == size - 1) {
        this.rightDomainBorder = point_x;
    }
```

Метод **addPoint(FunctionPoint point)** добавляет точку в массив **FunctionPoint[] points.** Сначала мы определяем, принадлежит ли точка области определения или расширяет её. Как только нашли индекс добавляемой точки, проверяем есть ли необходимость в расширении массива.

Если есть, создаём новый массив с удвоенной длинной и добавляем точку. Если нет необходимости расширять массив, то просто добавляем точку. После добавления параметр **size** увеличивается на 1. Обязательно обновляем границы области определения.

Задание 7:

Проверяем работу написанных классов:

```java
import functions.*;

public class Main {
    Run | Debug
    public static void main(String[] args) {

        // Создание табулированной функции (например, y = x²)
        System.out.println(x:"1. Создание функции y = x² на отрезке [0, 4] с 5 точками:");

        TabulatedFunction function = new TabulatedFunction(leftX:0, rightX:4, pointsCount:5);

        // Вывод начальных точек
        System.out.println(x:"Начальные точки:");
        for (int i = 0; i < function.getPointsCount(); i++) {
            System.out.printf(format:"(%.2f, %.2f) ", function.getPointX(i), function.getPointY(i));
        }
        System.out.println(x:"\n");

        // Установка реальных значений y = x²
        System.out.println(x:"2. Установка значений y = x²:");

        for (int i = 0; i < function.getPointsCount(); i++) {
            double x = function.getPointX(i);
            function.setPointY(i, x * x);
            System.out.printf(format:"Точка %d: (%.2f, %.2f)\n", i, x, x * x);
        }
        System.out.println();

        // Тестирование вычисления значений в разных точках
        System.out.println(x:"3. Вычисление значений функции:");
        double[] testPoints = { -1, 0, 0.5, 1, 1.5, 2, 3.5, 4, 5 };

        for (double x : testPoints) {
            double y = function.getFunctionValue(x);

            if (Double.isNaN(y)) {
                System.out.printf(format:"x = %.2f -> ВНЕ области определения\n", x);
            } else {
                System.out.printf(format:"x = %.2f -> y = %.2f\n", x, y);
            }
        }
        System.out.println();

        // Добавление новой точки
        System.out.println(x:"4. Добавление точки (2.5, 6.25):");
        try {
            function.addPoint(new FunctionPoint(x:2.5, y:6.25));
            System.out.println(x:"Точка добавлена успешно");
            System.out.println("Теперь точек: " + function.getPointsCount());
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
        System.out.println();

        // 5. Проверка интерполяции после добавления
        System.out.println(x:"5. Проверка интерполяции после добавления:");
        for (double x = 2.2; x <= 2.8; x += 0.2) {
            double y = function.getFunctionValue(x);
            System.out.printf(format:"x = %.2f -> y = %.2f\n", x, y);
        }
        System.out.println();

    }
}
```

```java
        // 6. Удаление точки
        System.out.println(x:"6. Удаление точки с индексом 2:");
        try {
            function.deletePoint(index:2);

            System.out.println(x:"Точка удалена успешно");
            System.out.println("Теперь точек: " + function.getPointsCount());

        } catch (IllegalArgumentException e) {
            System.out.println("Ошибка: " + e.getMessage());
        }
        System.out.println();

        // 7. Проверка границ после изменений
        System.out.println(x:"7. Новые границы области определения:");
        System.out.printf(format:"Левая граница: %.2f\n", function.getLeftDomainBorder());
        System.out.printf(format:"Правая граница: %.2f\n", function.getRightDomainBorder());
        System.out.println();

        // 8. Попытка добавить точку с существующим X
        System.out.println(x:"8. Попытка добавить точку с существующим X:");

        try {
            function.addPoint(new FunctionPoint(x:3.0, y:100)); // Должен быть отказ
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }

        // 9. Финальный вывод всех точек
        System.out.println(x:"\n9. Финальный набор точек:");

        for (int i = 0; i < function.getPointsCount(); i++) {
            System.out.printf(format:"Точка %d: (%.2f, %.2f)\n", i,function.getPointX(i), function.getPointY(i));

        }

    }
```

Результат работы программы после запуска:

```
ExceptionMessages' '-cp' 'C:\Users\USER\AppData\Roaming\Code\User\workspaceStorage\3d3edb13d53563941de0a16c0c22816b\redhat.java\jdt_ws\Lab-2-2025_537fbdb1\bin' 'Main'
1. Создание функции y = x? на отрезке [0, 4] с 5 точками:
Начальные точки:
(0,00, 0,00) (1,00, 0,00) (2,00, 0,00) (3,00, 0,00) (4,00, 0,00)

2. Установка значений y = x?:
Точка 0: (0,00, 0,00)
Точка 1: (1,00, 1,00)
Точка 2: (2,00, 4,00)
Точка 3: (3,00, 9,00)
Точка 4: (4,00, 16,00)

3. Вычисление значений функции:
x = -1,00 -> ВНЕ области определения
x = 0,00 -> y = 0,00
x = 0,50 -> y = 0,50
x = 1,00 -> y = 1,00
x = 1,50 -> y = 2,50
x = 2,00 -> y = 4,00
x = 3,50 -> y = 12,50
x = 4,00 -> y = 16,00
x = 5,00 -> ВНЕ области определения

4. Добавление точки (2.5, 6.25):
Точка добавлена успешно
Теперь точек: 6

5. Проверка интерполяции после добавления:
x = 2,20 -> y = 4,90
x = 2,40 -> y = 5,80
x = 2,60 -> y = 6,80

6. Удаление точки с индексом 2:
Точка удалена успешно
Теперь точек: 5

7. Новые границы области определения:
Левая граница: 0,00
Правая граница: 4,00

8. Попытка добавить точку с существующим X:
Точка должна иметь уникальное значение x

9. Финальный набор точек:
Точка 0: (0,00, 0,00)
Точка 1: (1,00, 1,00)
Точка 2: (2,50, 6,25)
Точка 3: (3,00, 9,00)
Точка 4: (4,00, 16,00)
PS C:\Users\USER\OneDrive\Documents\Java_projects\Lab-2-2025>
```