ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 2

«Разработка набора классов для работы с табулированными функциями»

по курсу
Объектно-ориентированное программирование

Выполнил Родомакин Михаил
студент группы 6203-010302D

Оглавление

Задание №1

Используя IDE “IntelliJ IDEA” создаем пакет functions.

Задание №2

Далее по следуюя указаниям в задании я создал класс FunctionPoint, в котором обозначил два private double поля x и y. Также были добавлены конструкторы FunctionPoint(double x, double y), FunctionPoint(FunctionPoint point) и FunctionPoint(). Для последующих заданий были написаны геттеры и сеттеры полей.

```
public class FunctionPoint {  20 usages
    private double x,y;  6 usages
    public FunctionPoint(double x, double y){  4 usages
        this.x = x;
        this.y = y;
    }
    public FunctionPoint(FunctionPoint point) {  3 usages
        this.x = point.x;
        this.y = point.y;
    }

    public FunctionPoint(){  1 usage
        this.x = 0;
        this.y = 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 Конструкторы, геттеры и сеттеры класса FunctionPoint

Задание №3

Далее был создан класс TabulatedFunction с полем-массивом типа FunctionPoint[] названным Points, в котором мы будем хранить наши точки, а также с полем типа int length, который будет счетчиком элементов. Тажке были написаны два конструктора для этого класса.

Позднее я дописал проверку и исправил ошибки с количеством элементов в функции, которое не может быть меньше двух, т.к. функция по определению должна иметь хотя бы две точки, и добавил перемену местами левой и правой границы по x если они введены не в том порядке.

```
//Конструкторы
public TabulatedFunction(double leftX, double rightX, int pointsCount){  1 usage
    this.Points =  new FunctionPoint[pointsCount];
    this.length = Math.max(pointsCount,2); //длина должжна быть больше 2

    if (leftX > rightX) { //Проверка на соответствие левой и правой границы
        double temp = leftX;
        leftX = rightX;
        rightX = temp;
    }

    double delta = (rightX - leftX) / (length-1); //Создание точек через равные по x промежутки
    for (int i = 0; i < pointsCount; i++){
        double x = leftX + i*delta;
        this.Points[i] = new FunctionPoint(x, y: 0);
    }
}
```

```
//Конструктор по значениям
public TabulatedFunction(double leftX, double rightX, double[] values){  no usages
    if(values.length < 2) { //длина должжна быть больше 2
        double[] valuesExtended = {values[0], values[0]};//Добавляю копию точки в массив
        values = valuesExtended;
    }
    this.Points =  new FunctionPoint[values.length];
    this.length = values.length;

    if (leftX > rightX) { //Проверка на соответствие левой и правой границы
        double temp = leftX;
        leftX = rightX;
        rightX = temp;
    }

    double delta = (rightX - leftX) / (length-1);
    for (int i = 0; i < length; i++){ //Создание точек через равные по x промежутки
        double x = leftX + i*delta;
        this.Points[i] = new FunctionPoint(x, values[i]);
    }
}
```

Рисунок 2 Конструкторы класса TabulatedFunction

Задание №4

Функции getLeftDomainBorder() и getRightDomainBorder() являются простыми геттерами x значений. Они выводят x первого и последнего элементов соответственно, т.к. мы храним точки по возрастанию значения абсциссы.

Функция getFunctionValue была написана в соответствии с условием задания. Она должна вывести значение функции по данной абсциссе, даже если нужного x нет среди записанных точек. Для решения таких ситуаций, по заданию, следует использовать линейную интерполяцию. Все это было реализовано в этой функции.

```
public double getFunctionValue(double x){  3 usages
    FunctionPoint p1 = new FunctionPoint();
    FunctionPoint p2 = p1;

    if (x<this.getLeftDomainBorder() || x>this.getRightDomainBorder())
        return Double.NaN; //x в пределах области функции
    for(int i = 0; i<length-1; i++){
        if (Points[i+1].getX()>=x) { //Ищщем нужный отрезок между точками
            p1 = Points[i];
            p2 = Points[i+1];
            //Считаем значение по формуле
            return p1.getY() + (x - p1.getX())* (p2.getY() - p1.getY()) / (p2.getX() - p1.getX());
        }
    }
    return Double.NaN;
}
```

Рисунок 3 Ф-ия getFunctionValue с интерполяцией

Задание №5

Далее было добавлено еще несколько геттеров и сеттеров для нашего класса. В частности getPointsCount() возвращает length – количество элементов в массиве, getPoint(int index) возвращает копию точки по индексу (аналогично getPointX и getPointY). Сеттеры оказались сложнее(для точки и x) т.к. требуется проверять не только попадание в область определения функции, но и смотреть чтобы значения менялись в нужном интервале и точки оставались отсортированными по абсциссе.

```
//Сеттеры
public void setPoint(int index, FunctionPoint point) {  1 usage
    if (index < 0 || index > length-1 || point == null) return; //Проверка индекса
    //Проверка попадания в интервалы
    if (point.getX() <= Points[index - 1].getX() || ( index < length -1 && point.getX() >= Points[index + 1].getX())) return;
    Points[index] = new FunctionPoint(point);
}

public void setPointX(int index, double x){  1 usage
    if (index < 0 || index > length-1) return; //Проверка индексов
    //Проверка попадания в интервалы
    if ( (index > 0 && x <= Points[index - 1].getX()) || ( index < length -1 && x >= Points[index + 1].getX())) return;
    Points[index].setX(x);
}

public void setPointY(int index, double y){  2 usages
    if (index < 0 || index > length-1) return; //Проверка индексов
    Points[index].setY(y);
}
```

Рисунок 4 Сеттеры точки в классе TabulatedFunction

Задание №6

В этом задании требуется написать функции удаления и добавления точки в функцию.
Удаление было сделано через сдвиг элементов после идущих после удаляемого на один влево, после чего переменная длины уменьшается на один. Таким образом пересоздавать массив не требуется.

```
public void deletePoint(int index){  1 usage
    if (length <= 2 || index < 0 || index > length -1) return; //Проверка индекса и длины

    for (int i = index; i < length-1; i++) //Сдвигаем элементы влево
        Points[i] = Points[i+1];
    Points[length-1] = null; //Заменяем последний на ноль
    length--; //Уменьшаем переменную длины
}
```

Рисунок 5 Ф-ия deletePoint

Функция добавления точки сделана практически также, но элементы сдвигаются вправа от места добавления. Для того чтобы это было возможно, нужно либо чтобы массив до этого уже был уменьшен на 1 функцией удаления элемента, либо чтобы его длина была увеличина. Для этого мы проверяем разницу между фактической длиной массива и переменной длины, после чего либо увеличиваем переменную, пересоздаем массив увеличивая размер.

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

    int index = 0; //Ищщем место для новой точки по x
    while (index < length && point.getX() > Points[index].getX()){
        index++;
    }
    if(Points[index].getX() == point.getX()){ //Если совпадает с другой точкой
        return;
    }

    if (length == Points.length){ //Если массив первоначального размера, увеличиваем его
        FunctionPoint[] extendedPoints = new FunctionPoint[Points.length + 1];
        for (int i = 0; i< length; i++)
            extendedPoints[i] = Points[i];

        Points = extendedPoints;
    }

    for (int i=length; i> index; i--){ //Сдвиг элементов вправо
        Points[i] = Points[i-1];
    }
    Points[index] = new FunctionPoint(point); //Добавление элемента и увеличение переменной длины
    length++;
}
```

Рисунок 6 Ф-ия addPoint

Задание №7

В качестве примера я использовал функцию y=x^2 и функционал класса был продемонстрирован на ней, используя 5 точек в интервалле от 0 до 10;

```
-_-_-_-Ф-ия: y = x^2-_-_-_-
Исходная ф-ия:
Точка 0: x=0.0 y=0.0
Точка 1: x=2.5 y=6.25
Точка 2: x=5.0 y=25.0
Точка 3: x=7.5 y=56.25
Точка 4: x=10.0 y=100.0

_-_-_-Добавление точки (3, 15)-_-_-_
После добавления:
Точка 0: x=0.0 y=0.0
Точка 1: x=2.5 y=6.25
Точка 2: x=3.0 y=15.0
Точка 3: x=5.0 y=25.0
Точка 4: x=7.5 y=56.25
Точка 5: x=10.0 y=100.0

-_-_-_-Удаление точки с индексом 4-_-_-_-
После удаления:
Точка 0: x=0.0 y=0.0
Точка 1: x=2.5 y=6.25
Точка 2: x=3.0 y=15.0
Точка 3: x=5.0 y=25.0
Точка 4: x=10.0 y=100.0

-_-_-_-Изменение точки с индексом 1-_-_-_-
После изменения:
Точка 0: x=0.0 y=0.0
Точка 1: x=1.5 y=8.0
Точка 2: x=3.0 y=15.0
Точка 3: x=5.0 y=25.0
Точка 4: x=10.0 y=100.0

-_-_-_-Интерполяция значений-_-_-_-
3.5^2 = 17.5
6.5^2 = 47.5
8^2 = 70.0
```

Рисунок 8 Тесты

```
-_-_-_-Границы области определения-_-_-_-
Левая граница: 0.0
Правая граница: 10.0

-_-_-_-Изменение координат по отдельности-_-_-_-
До изменения: Точка 0: 0.0; 0.0
После изменения: Точка 0: 1.0; 3.0

-_-_-_-Итоговое состояние функции-_-_-_-
Точка 0: x=1.0 y=3.0
Точка 1: x=1.5 y=8.0
Точка 2: x=3.0 y=15.0
Точка 3: x=5.0 y=25.0
Точка 4: x=10.0 y=100.0
Всего точек: 5

Process finished with exit code 0
```

Рисунок 9 Тесты