

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ ГОСУДАРСТВЕННОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени
академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА №2

Студент Семенова В. Г.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Содержание

Задание 1	3
Задание 2	3
Задание 3	4
Задание 4	5
Задание 5	6
Задание 6	7
Задание 7.	8

Задание 1

В первом задании создаем пакет functions, в котором далее будут создаваться классы программы. То есть в папке с лабораторной создаем папку «functions».

Задание 2

В пакете создаем класс «FunctionPoint». В нем содержатся 2 переменные для координат точки по оси абсцисс и оси ординат.

```
private double x;  
private double y;
```

В классе создаем 3 конструктора. Первый конструктор

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

необходим для создания новой точки с переданными значениями.

Второй конструктор

```
FunctionPoint(FunctionPoint point){  
    this.x=point.x;  
    this.y=point.y;  
}
```

создает объект точки с теми же координатами, что у указанной точки. Это конструктор копирования, который нужен, чтобы исходные данные не изменялись извне.

Третий конструктор

```
FunctionPoint(){  
    this(0.0, 0.0);  
}
```

создает точку с координатами (0,0).

Также в этот класс описаны методы для получения и установки значения поля для каждой переменной.

```
public double get_x(){  
    return x;  
}  
  
public void set_x(double x) {  
    this.x = x;  
}
```

Задание 3

В том же пакете создаем класс «TabulatedFunction»

В нем создаем 2 приватных поля, одно — это массив переменных, второе это счетчик количества переменных.

```
private FunctionPoint[] points;  
private int points_count;
```

Конструктор «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. Интервал делится на равные промежутки по x, на длину, зависящую от количества входных точек. Значение функции в любой точке равно нулю.

```
public TabulatedFunction(double leftX, double rightX, int pointsCount){  
    this.points_count = pointsCount;  
    this.points = new FunctionPoint[pointsCount];  
  
    double step = (rightX - leftX) / (pointsCount - 1);  
  
    for (int i = 0; i < points_count; i++) {  
        double x = leftX + i * step;  
        points[i] = new FunctionPoint(x, 0.0);  
    }  
}
```

Конструктор «TabulatedFunction(double leftX, double rightX, double[] values)» принимает на вход вместо количества точек массив значений функции. Значение координаты X рассчитывает равномерно на отрезке и создает новую переменную на каждом i-ом шаге.

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

Задание 4

Метод «double getLeftDomainBorder()» возвращает левую границу области, т.е. значение x на границе.

Метод «double getRightDomainBorder()» возвращает правую границу области.

Метод «double getFunctionValue(double x)» возвращает значение функции в указанной точке.

Если точка не попадает в интервал, то метод возвращает значение NaN.

Так как переменные x и y вещественные, а вещественные числа в памяти компьютера хранятся в виде последовательности нулей и единиц с ограниченной точностью, то мы не можем напрямую сравнивать 2 вещественных числа. Поэтому чтобы узнать совпадает заданное значение x с каким-нибудь значением из нашего интервала, мы сравниваем их разность по модулю с заданной точностью. Если сравнение пройдено, то метод возвращает значение функции в точке.

Если заданная точка не совпадает ни с какой точкой из интервала, то используем линейную интерполяцию по соседним точкам. Значения y вычисляется с предположением, что между двумя этими точками функция является прямой.

В методе есть «return Double.NaN», который нужен, когда ни одно условие не выполнится. Например, если количество точек будет равно одному.

```
public double getFunctionValue(double x){  
    if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {  
        return Double.NaN;  
    }  
}
```

```

    for (int i = 0; i < points_count; i++) {

        if (Math.abs(x - points[i].get_x()) < 1e-10) {
            return points[i].get_y();
        }

        if (i < points_count - 1) {
            double x1 = points[i].get_x();
            double x2 = points[i + 1].get_x();

            if (x > x1 && x < x2) {
                double y1 = points[i].get_y();
                double y2 = points[i + 1].get_y();
                return y1 + (y2 - y1) * (x - x1) / (x2 - x1);
            }
        }
    }
    return Double.NaN;
}

```

Задание 5

Метод «FunctionPoint getPoint(int index)» создает копию точки с заданным индексом, чтобы исходные данные не изменились.

Метод «setPoint» заменяет указанную точку на переданную. Перед этим идут проверки того, что переданный индекс не выходит за границы и что точка на которую хотим заменить входит в интервал значений x.

```

public void setPoint(int index, FunctionPoint point){
    if (index < 0 || index >= points_count) {
        throw new IndexOutOfBoundsException("Индекс выходит за границы");
    }

    if (point == null) {
        throw new IllegalArgumentException("Точка не может быть null");
    }
    if (index > 0) {
        double leftX = points[index - 1].get_x();
        if (point.get_x() <= leftX) {
            throw new IllegalArgumentException("X должен быть больше чем у
левой точки");
        }
    }
    if (index < points_count - 1) {
        double rightX = points[index + 1].get_x();
        if (point.get_x() >= rightX) {
            throw new IllegalArgumentException("X должен быть меньше чем у
правой точки");
        }
    }
}

```

```

    }
}
points[index] = new FunctionPoint(point);
}

```

Методы «getPointX», «getPointY», «FunctionPoint getPoint» сначала проверяют, что переданный индекс не выходит за границы.

Методы «setPointX(int index, double x)» и «setPointY» сначала проверяют, что переданный индекс попал в интервал. Первый метод также проверяет, что переданное значение x попадает в промежуток. Для второго метода такая проверка не нужна.

Задание 6

Метод «void deletePoint(int index)» проверяет, что указанный индекс попал в интервал, сдвигает элементы влево, начиная с того, который нужно удалить, зануляет последний элемент и уменьшает количество элементов на 1.

Метод «void addPoint(FunctionPoint point)» добавляет новую точку. Для этого сначала проверяется, что значение точки корректно (не null) и что координата x данной точки не совпадает с уже существующей.

Далее в цикле мы сравниваем координату x вставляемой точки с теми, что уже есть в массиве для сохранения упорядоченности.

```

int insertIndex = points_count;

for (int i = 0; i < points_count; i++) {
    if (point.get_x() < points[i].get_x()) {
        insertIndex = i;
        break;
    }
}

```

Если длины массива недостаточно, то создаем новый массив, длина которого больше в два раза, копируем в него старый массив

```

if (points_count >= points.length) {
    FunctionPoint[] newPoints = new FunctionPoint[points.length * 2];
    System.arraycopy(points, 0, newPoints, 0, points_count);
    points = newPoints;
}

```

Параметры в `System.arraycopy(points, 0, newPoints, 0, points_count)`:

`points` – старый массив, который копируем, «0» – индекс с которого начинаем копировать, `newPoints`, «0» – название нового и индекс, с которого заполняем, `points_count` – количество переносимых переменных.

Чтобы вставить переменную в массив на i -ую позицию, сдвигаем вправо все элементы, индекс которых больше i . Вставляем в нужную позицию переменную и увеличиваем счетчик.

```
if (insertIndex < points_count) {
    System.arraycopy(points, insertIndex, points, insertIndex + 1,
points_count - insertIndex);
}
points[insertIndex] = new FunctionPoint(point);
points_count++;
```

Задание 7.

Создаем класс `Main`, содержащий точку входа программы. В консоль выведены точки функции и примеры значений функции. Для вычисления значения функции в точках 2,5 и 6,4 используется интерполяция.

```
:\Users\леноно\OneDrive\Рабочий стол\oop\Lab-2-2025>java -Dfile.encoding=UTF-8 Main
абулированная функция y = x^2:
бласть определения: от 0.0 до 10.0
оличество точек: 11
очки функции:
(0.0; 0.0)
(1.0; 1.0)
(2.0; 4.0)
(3.0; 9.0)
(4.0; 16.0)
(5.0; 25.0)
(6.0; 36.0)
(7.0; 49.0)
(8.0; 64.0)
(9.0; 81.0)
(10.0; 100.0)

начения функции:
(1.0) = 1.0
(2.5) = 6.5
(6.4) = 41.2
(11.0) = NaN
```