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 Синцов М.Ю
Группа 6201-120303D
Руководитель Борисов Д. С.

САМАРА 2025

Цель работы
Разработка классов для работы с табулированными функциями, включая создание точек функции, хранение табличных значений, выполнение линейной интерполяции и манипуляции с точками функции.

Задание 1: Создание пакета functions
Выполнение: Создан пакет functions для организации структуры проекта. Пакет содержит два основных класса: FunctionPoint и TabulatedFunction.

Задание 2: Класс FunctionPoint
Реализация: Создан класс FunctionPoint, представляющий точку на плоскости с координатами (x, y). Класс включает:

- Три конструктора: с параметрами, копирующий и по умолчанию
- Приватные поля для координат с соблюдением инкапсуляции
- Геттеры и сеттеры для доступа к координатам
- Метод toString() для строкового представления

Задание 3: Класс TabulatedFunction
Реализация: Создан класс TabulatedFunction для представления табулированной функции:

Два конструктора:

- По границам области определения и количеству точек
- По границам области определения и массиву значений
- Автоматическое создание точек через равные интервалы
- Упорядоченное хранение точек по возрастанию x

Задание 4: Методы области определения и вычисления значения
Реализованы методы:

- getLeftDomainBorder() - возвращает левую границу области определения
- getRightDomainBorder() - возвращает правую границу области определения

- getFunctionValue(double x) - вычисляет значение функции в точке x с использованием линейной интерполяции: Возвращает Double.NaN для точек вне области определения; Использует уравнение прямой через две точки для интерполяции

Задание 5: Методы работы с точками
Реализованы методы:

- getPointsCount() - количество точек
- getPoint(int index) - получение копии точки (инкапсуляция)
- setPoint(int index, FunctionPoint point) - замена точки с проверкой порядка
- getPointX(int index), setPointX(int index, double x) - работа с абсциссой
- getPointY(int index), setPointY(int index, double y) - работа с ординатой

Задание 6: Методы изменения количества точек
Реализованы методы:

- deletePoint(int index) - удаление точки с сохранением порядка
- addPoint(FunctionPoint point) - добавление новой точки в правильную позицию
- Использование System.arraycopy() для эффективного копирования массивов
- Динамическое расширение массива при необходимости

Задание 7: Тестирование программы
Создан класс Main для тестирования:

Создание функции $f(x) = x^2$ на интервале [-2, 2]

Вычисление значений в различных точках (включая вне области определения)

Тестирование методов работы с точками

Добавление и удаление точек

Проверка линейной интерполяции

Создание функции синуса через массив значений

Результаты работы

Код класса FunctionPoint:

```
 package functions;

public class FunctionPoint {
    private double x;
    private double y;

    public FunctionPoint(double x, double y){
        this.x = x;
        this.y = y;
    }
    public FunctionPoint(FunctionPoint point){
        x = point.x;
        y = point.y;
    }
    FunctionPoint(){
        x = 0;
        y = 0;
    }
    public double getY(){
        return y;
    }
    public double getX(){
        return x;
    }
    public void setX(double x){
        this.x = x;
    }
    public void setY(double y){
        this.y = y;
    }
    public void showPoint(){
        System.out.println("[" + x + "," + y + "]") ;
    }
}
```

Код класса TabulatedFunctionPoint:

```
package functions;
public class TabulatedFunction {
    private FunctionPoint[] points;
    final double EPSILON = 2.220446049250326E-16;
    private int pointsCount;
    public TabulatedFunction(double leftX, double rightX, int pointsCount){
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i<pointsCount;i++){
            double x = leftX + i*step;
```

```
            points[i] = new FunctionPoint(x, 0);
        }
        this.pointsCount = pointsCount;
    }
    public TabulatedFunction(double leftX, double rightX, double[] values){
        if (values.length < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }
        pointsCount = values.length;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i< pointsCount;i++){
            double x = leftX + i*step;
            points[i] = new FunctionPoint(x, values[i]);
        }

    }
    public double getLeftDomainBorder(){
        return points[0].getX();
    }
    public double getRightDomainBorder(){
        return points[pointsCount-1].getX();
    }
    public int getPointsCount() {
        return pointsCount;
    }

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


        for (int i = 0; i < pointsCount; i++) {
            if (Math.abs(points[i].getX() - x) <= EPSILON) {
                return points[i].getY();
            }
        }


        for (int i = 0; i < pointsCount - 1; i++) {
            if (x >= points[i].getX() && x <= points[i + 1].getX()) {
                double leftX = points[i].getX();
                double rightX = points[i + 1].getX();
                double leftY = points[i].getY();
                double rightY = points[i + 1].getY();


                return leftY + (rightY - leftY) * (x - leftX) / (rightX -
leftX);
            }
        }


        return Double.NaN;
    }
    public void showPoints(){
        for (int i = 1; i<pointsCount;i++){
            System.out.print("Значение точки "+ (i+1) + ": ");
```

```
            points[i].showPoint();
            System.out.println();
        }
    }
    public FunctionPoint getPoint(int index){
        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
        }
        return new FunctionPoint(points[index]);
    }
    public void setPoint(int index, FunctionPoint point){
        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
        }
        if (point.getX()<getLeftDomainBorder() ||
point.getX()>getRightDomainBorder()){
            throw new IndexOutOfBoundsException("Координата x задаваемой
точки лежит вне интервала");
        }
        points[index] = point;
    }
    public double getPointX(int index){
        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
        }
        return points[index].getX();
    }
    public void setPointX(int index, double x){

        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
        }
        if (x<getLeftDomainBorder() || x>getRightDomainBorder()){
            throw new IndexOutOfBoundsException("Координата x задаваемой
точки лежит вне интервала");
        }
        if(x>=points[index+1].getX()||x<=points[index-1].getX()){
            throw new IllegalArgumentException("Новое значение x нарушает
упорядоченность");
        }
        points[index].setX(x);
    }
    public double getPointY(int index){
        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
        }
        return points[index].getY();
    }
    public void setPointY(int index, double y){
        if (index < 0 || index >= pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
        }
        points[index].setY(y);
    }
    public void deletePoint(int index){
        if (index < 0 || index >=pointsCount) {
            throw new IndexOutOfBoundsException("Индекс выходит за границы
массива точек");
```

```
        }
        if (pointsCount <= 2) {
            throw new IllegalStateException("Нельзя удалить точку: функция
должна содержать минимум 2 точки");
        }
        System.arraycopy(points,index +1,points,index,pointsCount-1-index);
        pointsCount--;
        points[pointsCount] = null;


    }
    public void addPoint(FunctionPoint point){
        int insertIndex = 0;
        while (insertIndex < pointsCount && point.getX() >
points[insertIndex].getX()) {
            insertIndex++;
        }
        if (insertIndex < pointsCount && (Math.abs(point.getX() -
points[insertIndex].getX())<=EPSILON)) {
            throw new IllegalArgumentException("Точка с x=" + point.getX() +
" уже существует");
        }
        if (pointsCount == points.length) {
            int newCapacity = points.length + points.length / 2 + 1;
            FunctionPoint[] newPoints = new FunctionPoint[newCapacity];
            System.arraycopy(points, 0, newPoints, 0, pointsCount);
            points = newPoints;
        }

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

Код класса main:

```
import functions.*;
public class Main {
    public static void main(String[] args){
        TabulatedFunction func = new TabulatedFunction(0, 4, 3);
        System.out.println("Исходная функция: ");
        func.showPoints();


        System.out.println("Левая граница: " + func.getLeftDomainBorder());
        System.out.println("Правая граница: " + func.getRightDomainBorder());
        System.out.println("f(2.0) = " + func.getFunctionValue(2.0));
        System.out.println("f(5.0) = " + func.getFunctionValue(5.0)); //
Должен вернуть NaN


        System.out.println("Количество точек: " + func.getPointsCount());
        System.out.println("Точка 1: ");
        func.getPoint(1).showPoint();
        System.out.println("X точки 1: " + func.getPointX(1));
        System.out.println("Y точки 1: " + func.getPointY(1));


        func.setPointY(1, 5.0);
        System.out.println("После изменения Y точки 1: ");
        func.showPoints();
```

```
        func.addPoint(new FunctionPoint(1.5, 2.5));
        System.out.println("После добавления точки (1.5; 2.5): ");
        func.showPoints();

        func.deletePoint(1);
        System.out.println("После удаления точки 1: ");
        func.showPoints();

        System.out.println("f(0.5) = " + func.getFunctionValue(0.5));
        System.out.println("f(1.0) = " + func.getFunctionValue(1.0));
        System.out.println("f(1.7) = " + func.getFunctionValue(1.7));
    }
}
```

Вывод программы:
Исходная функция:
Значение точки 2: [2.0,0.0]

Значение точки 3: [4.0,0.0]

Левая граница: 0.0
Правая граница: 4.0
f(2.0) = 0.0
f(5.0) = NaN
Количество точек: 3
Точка 1:
[2.0,0.0]
X точки 1: 2.0
Y точки 1: 0.0
После изменения Y точки 1:
Значение точки 2: [2.0,5.0]

Значение точки 3: [4.0,0.0]

После добавления точки (1.5; 2.5):
Значение точки 2: [1.5,2.5]

Значение точки 3: [2.0,5.0]

Значение точки 4: [4.0,0.0]

После удаления точки 1:
Значение точки 2: [2.0,5.0]

Значение точки 3: [4.0,0.0]

f(0.5) = 1.25
f(1.0) = 2.5

f(1.7) = 4.25

Выводы
В ходе лабораторной работы были успешно реализованы:

Пакет functions с четкой структурой классов

Класс FunctionPoint с полной инкапсуляцией данных

Класс TabulatedFunction с поддержкой:

Хранения упорядоченных точек

Линейной интерполяции значений

Операций добавления/удаления точек

Проверки корректности операций

Основные принципы ООП:

Инкапсуляция (приватные поля, копирование объектов)

Проверка входных данных

Обработка исключительных ситуаций

Проведено тестирование всех функций:

Корректная работа в граничных точках

Обработка точек вне области определения

Сохранение порядка точек при операциях

Программа демонстрирует правильную работу с табулированными функциями и может быть использована как основа для более сложных математических вычислений.