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 №4

Студент Кольчугина Е. А.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Содержание

Задание 1

Добавлены конструкторы, принимающие массив объектов FunctionPoint. Конструкторы проверяют, что точек не менее двух и они упорядочены по возрастанию x. При нарушении условий выбрасывается IllegalArgumentException.

Используется копия массива точек через конструктор копирования new FunctionPoint(points[i]). Это предотвращает изменение внутреннего состояния извне.

```
public LinkedListTabulatedFunction(FunctionPoint[] points){
    if (points.length < 2){
        throw new IllegalArgumentException(s: "Должно быть не менее двух точек");
    }
    for (int i = 1; i < points.length; i++){
        if (points[i] == null || points[i - 1] == null){
            throw new IllegalArgumentException(s: "Точки не могут быть null");
        }
        if (doubleLessOrEquals(points[i].getX(), points[i - 1].getX())){
            throw new IllegalArgumentException(s: "Точки должны возрастать по абсциссе");
        }
    }
    emptyList();

    for (FunctionPoint point : points) {
        addNodeToTail().setPoint(new FunctionPoint(point));
    }
}
```

```
public ArrayTabulatedFunction(FunctionPoint[] points){
    if (points.length < 2) {
        throw new IllegalArgumentException(s: "Количество точек должно быть >= 2");
    }
    for (int i = 1; i < points.length; i++){
        if (points[i] == null || points[i - 1] == null){
            throw new IllegalArgumentException(s: "Точки не могут быть null");
        }
        if (doubleLessOrEquals(points[i].getX(), points[i - 1].getX())){
            throw new IllegalArgumentException(s: "Точки должны возрастать по абсциссе");
        }
    }

    this.pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount + 2];

    for (int i = 0; i < pointsCount; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

Задание 2

Создан общий интерфейс Function с тремя базовыми методами:

- getLeftDomainBorder() - где функция начинается
- getRightDomainBorder() - где функция заканчивается
- getFunctionValue(x) - чему равна функция в точке x

Интерфейс может работать как с аналитическими функциями, так и с табулированными.

TabulatedFunction *является* частным случаем Function (наследует его)

Lab-4-2025 > functions > Function.java > ...

```java
package functions;

public interface Function {

    double getLeftDomainBorder();

    double getRightDomainBorder();

    double getFunctionValue(double x);

}
```

Lab-4-2025 > functions > TabulatedFunction.java > Java > TabulatedFunction > getPointsCount()

```java
package functions;

public interface TabulatedFunction extends Function {

    int getPointsCount();

    FunctionPoint getPoint(int index);

    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;

    double getPointX(int index);

    void setPointX(int index, double x) throws InappropriateFunctionPointException;

    double getPointY(int index);

    void setPointY(int index, double y);

    void deletePoint(int index);

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;


}
```

Задание 3

Созданы базовые математические функции

Экспонента (Exp):

- Функция e^x
- Область определения: все действительные числа (-∞ до +∞)
- Реализация: Math.exp(x)

Lab-4-2025 > functions > basic > Exp.java > Java > Exp

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {
    @Override
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override
    public double getFunctionValue(double x){
        return Math.exp(x);
    }

}
```

Логарифм (Log):

- Функция $\log_a(x)$ - логарифм x по основанию a
- Область определения: x > 0 (только положительные числа)
- Основание задается в конструкторе

Так как Math.log является натуральным логарифмом ln, для подсчета логарифма по заданному основанию использована формула:

$$\log_{base} x = \frac{\ln x}{\ln base}$$

```java
package functions.basic;

import functions.Function;

public class Log implements Function {
    private double base;

    public Log(double base) {
        if(base <= 0 || base == 1) {
            throw new IllegalArgumentException(s: "Основание должно быть больше 0 и не равно 1");
        }
        this.base = base;
    }

    @Override
    public double getLeftDomainBorder() {
        return 0.0;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override
    public double getFunctionValue(double x){
        if (x <= 0){
            return Double.NaN;
        }
        return Math.log(x) / Math.log(base);        // Math.log - это натуральный логарифм
    }

}
```

2. Созданы тригонометрические функции

Вместо того, чтобы создавать три одинаковых класса (Sin, Cos, Tan) с одинаковыми методами getLeftDomainBorder() и getRightDomainBorder(), создан

Класс TrigonometricFunction:

- Определяет общие свойства для всех тригонометрических функций
- Область определения: все действительные числа
- Содержит общую реализацию методов
- Является абстрактным, так как содержит метод без реализации (будет реализован в наследниках)

```java
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {
    @Override
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override
    public abstract double getFunctionValue(double x);

}
```

Конкретные реализации:

- **Sin** (синус): Math.sin(x)

```java
package functions.basic;

public class Sin extends TrigonometricFunction {

    @Override
    public double getFunctionValue(double x){
        return Math.sin(x);
    }
}
```

- **Cos** (косинус): Math.cos(x)

```java
package functions.basic;

public class Cos extends TrigonometricFunction {

    @Override
    public double getFunctionValue(double x){
        return Math.cos(x);
    }
}
```

- **Tan** (тангенс): Math.tan(x)

```java
package functions.basic;

public class Tan extends TrigonometricFunction {

    @Override
    public double getFunctionValue(double x){
        return Math.tan(x);
    }
}
```

Каждый класс наследует от TrigonometricFunction и реализует только вычисление значения

Задание 4

Создан пакет functions.meta — специальная папка в проекте, где содержатся все классы для комбинирования функций. Реализовано 6 классов-комбинаторов:

1. Класс Sum (Сумма)

Складывает значения двух функций:

- Принимает две функции в конструкторе
- Область определения — пересечение областей исходных функций
- Значение в точке x = f1(x) + f2(x)

2025 > functions > meta > J Sum.java > ...

```java
package functions.meta;

import functions.Function;

public class Sum implements Function{
    private Function f_1;
    private Function f_2;

    public Sum(Function f_1, Function f_2){
        this.f_1 = f_1;
        this.f_2 = f_2;
    }

    @Override
    public double getLeftDomainBorder(){
        return Math.max(f_1.getLeftDomainBorder(), f_2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(f_1.getRightDomainBorder(), f_2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f_1.getFunctionValue(x) + f_2.getFunctionValue(x);
    }
}
```

2. Класс Mult (Умножение)

Умножает значения двух функций:

- Область определения — пересечение областей исходных функций
- Значение в точке x = f1(x) * f2(x)

```java
package functions.meta;

import functions.Function;

public class Mult implements Function{
    private Function f_1;
    private Function f_2;

    public Mult(Function f_1, Function f_2){
        this.f_1 = f_1;
        this.f_2 = f_2;
    }

    @Override
    public double getLeftDomainBorder(){
        return Math.max(f_1.getLeftDomainBorder(), f_2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(f_1.getRightDomainBorder(), f_2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f_1.getFunctionValue(x) * f_2.getFunctionValue(x);
    }
}
```

3. Класс Power (Степень)

Возводит функцию в степень:

- Принимает функцию и степень в конструкторе
- Область определения — область исходной функции
- Значение в точке x: f(x)^power

```
package functions.meta;

import functions.Function;

public class Power implements Function{
    private Function f;
    private double power;

    public Power(Function f, double power){
        this.f = f;
        this.power = power;
    }

    @Override
    public double getLeftDomainBorder(){
        return f.getLeftDomainBorder();
    }

    @Override
    public double getRightDomainBorder() {
        return f.getRightDomainBorder();
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return Math.pow(f.getFunctionValue(x), power);
    }
}
```

4. Класс Scale (Масштабирование)

Растягивает или сжимает график функции:

- Принимает функцию и коэффициент растяжения вдоль осей x и y
- Область определения — область исходной функции*коэффициент x
- Значение в точке x = f(x/scaleX) * scaleY
- Работает и с отрицательными коэффициентами (зеркальное отражение).

```java
package functions.meta;

import functions.Function;

public class Scale implements Function{
    private Function f;
    private double scaleX;
    private double scaleY;

    public Scale(Function f, double scaleX, double scaleY){
        this.f = f;
        this.scaleX = scaleX;
        this.scaleY = scaleY;
    }

    @Override
    public double getLeftDomainBorder(){
        if (scaleX >= 0) {
            return f.getLeftDomainBorder() * scaleX;
        } else {
            return f.getRightDomainBorder() * scaleX;
        }
    }

    @Override
    public double getRightDomainBorder() {
        if (scaleX >= 0) {
            return f.getRightDomainBorder() * scaleX;
        } else {
            return f.getLeftDomainBorder() * scaleX;
        }
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f.getFunctionValue(x / scaleX) * scaleY;
    }
}
```

5. Класс Shift (Сдвиг)

Сдвигает график функции:

- Принимает функцию и величины сдвига вдоль осей x и y
- Область определения — область исходной функции + сдвиг x
- Значение в точке x = f(x - shiftX) + shiftY

```java
package functions.meta;

import functions.Function;

public class Shift implements Function{
    private Function f;
    private double shiftX;
    private double shiftY;

    public Shift(Function f, double shiftX, double shiftY){
        this.f = f;
        this.shiftX = shiftX;
        this.shiftY = shiftY;
    }

    @Override
    public double getLeftDomainBorder(){
        return f.getLeftDomainBorder() + shiftX;
    }

    @Override
    public double getRightDomainBorder() {
        return f.getRightDomainBorder() + shiftX;
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f.getFunctionValue(x - shiftX) + shiftY;
    }
}
```

6. Класс Composition (Композиция)

Делает функцию от функции:

- Принимает две функции (важен порядок!)
- Область определения — область внутренней функции (первой по порядку)
- Значение в точке x = f2(f1(x))
- Проверка на то, что результат f1 находится в области определения f2

```java
package functions.meta;

import functions.Function;

public class Composition implements Function{
    private Function f_1;
    private Function f_2;

    public Composition(Function f_1, Function f_2){
        this.f_1 = f_1;
        this.f_2 = f_2;
    }

    @Override
    public double getLeftDomainBorder(){
        return f_1.getLeftDomainBorder();
    }

    @Override
    public double getRightDomainBorder() {
        return f_1.getRightDomainBorder();
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double innerValue = f_1.getFunctionValue(x);
        if (innerValue < f_2.getLeftDomainBorder() || innerValue > f_2.getRightDomainBorder())
            return Double.NaN;
        }
        return f_2.getFunctionValue(innerValue);
    }
}
```

Задание 5

Создан класс Functions, расположенный в пакете functions. В нем содержатся только статические методы, то есть нельзя создать объект этого класса:

- Класс final - от него нельзя наследоваться
- Конструктор private - нельзя вызвать из другого класса
- Если кто-то попробует создать объект, получит ошибку

Внутри реализованы 6 методов shift(), scale(), power(), sum(), mult(), composition()

Каждый метод создает определенный тип комбинированной функции:

```
package functions;

import functions.meta.*;

public final class Functions {
    private Functions() {
        throw new Error(message: "Объекты этого класса нельзя создать");
    }

    public static Function shift(Function f, double shiftX, double shiftY) {
        return new Shift(f, shiftX, shiftY);
    }

    public static Function scale(Function f, double scaleX, double scaleY) {
        return new Scale(f, scaleX, scaleY);
    }

    public static Function power(Function f, double power) {
        return new Power(f, power);
    }

    public static Function sum(Function f_1, Function f_2) {
        return new Sum(f_1, f_2);
    }

    public static Function mult(Function f_1, Function f_2) {
        return new Mult(f_1, f_2);
    }

    public static Function composition(Function f_1, Function f_2) {
        return new Composition(f_1, f_2);
    }
}
```

Задание 6

Создан класс TabulatedFunctions и добавлен ключевой метод tabulate(), который используя функцию, отрезок [от leftX до rightX] и количество точек создает табулированную функцию:

- Класс объявлен как final - от него нельзя наследоваться
- Конструктор приватный - нельзя создать объект
- В конструкторе выбрасывается ошибка в качестве дополнительной защиты

```
package functions;

import java.io.*;
import java.util.StringTokenizer;


public final class TabulatedFunctions {
    private TabulatedFunctions() {
        throw new RuntimeException(message: "Объекты этого класса нельзя создать");
    }

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount){
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException(s: "Границы выходят за область определения");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException(s: "Левая граница области определения должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException(s: "Количество точек должно быть >=2");
        }

        double distance = (rightX - leftX) / (pointsCount - 1);
        double[] values = new double[pointsCount];

        for (int i = 0; i < pointsCount; i++){
            double x = leftX + i * distance;
            values[i] = function.getFunctionValue(x);
        }

        return new ArrayTabulatedFunction(leftX, rightX, values);

    }
```

Задание 7

- Поток (Stream) - это способ передачи данных по частям (побайтово)
- InputStream - чтение данных (из файла, сети, клавиатуры)
- OutputStream запись данных (в файл, сеть, экран)

Создано 4 метода в классе TabulatedFunctions:

Для бинарных файлов (компьютерный формат):

- outputTabulatedFunction() - сохраняет функцию в бинарный файл
- inputTabulatedFunction() - читает функцию из бинарного файла

```
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out){
    DataOutputStream dataOut = new DataOutputStream(out);
    try {
        dataOut.writeInt(function.getPointsCount());
        for (int i = 0; i < function.getPointsCount(); i++) {
            dataOut.writeDouble(function.getPointX(i));
            dataOut.writeDouble(function.getPointY(i));
        }

        dataOut.flush();
    } catch (IOException e) {
        throw new RuntimeException (message: "Ошибка при выводе функции", e);
    }
}

public static TabulatedFunction inputTabulatedFunction(InputStream in){
    DataInputStream dataIn = new DataInputStream(in);
    try {
        int pointsCount = dataIn.readInt();
        FunctionPoint[] points = new FunctionPoint[pointsCount];

        for (int i = 0; i < pointsCount; i++) {
            double x = dataIn.readDouble();
            double y = dataIn.readDouble();
            points[i] = new FunctionPoint(x, y);
        }
        return new ArrayTabulatedFunction(points);

    } catch (IOException e) {
        throw new RuntimeException (message: "Ошибка при вводе функции", e);
    }
}
```

Для текстовых файлов:

- writeTabulatedFunction() - сохраняет функцию в текстовый файл
- readTabulatedFunction() - читает функцию из текстового файла

```
public static void writeTabulatedFunction(TabulatedFunction function, Writer out){
    PrintWriter writer = new PrintWriter(out);
    try {
        writer.print(function.getPointsCount());
        writer.print(c: ' ');

        for (int i = 0; i < function.getPointsCount(); i++) {
            writer.print(function.getPointX(i));
            writer.print(c: ' ');
            writer.print(function.getPointY(i));
            if (i < function.getPointsCount() - 1) {
                writer.print(c: ' ');
            }
        }
        writer.flush();
    } catch (Exception e) {
        throw new RuntimeException(message: "Ошибка при записи функции", e);
    }
}

public static TabulatedFunction readTabulatedFunction(Reader in) {
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    try {
        tokenizer.resetSyntax();
        tokenizer.wordChars(low: '0', hi: '9');
        tokenizer.wordChars(low: '.', hi: '.');
        tokenizer.wordChars(low: '-', hi: '-');
        tokenizer.wordChars(low: 'e', hi: 'e');
        tokenizer.wordChars(low: 'E', hi: 'E');
        tokenizer.whitespaceChars(low: ' ', hi: ' ');
        tokenizer.whitespaceChars(low: '\t', hi: '\t');
        tokenizer.whitespaceChars(low: '\n', hi: '\n');
        tokenizer.whitespaceChars(low: '\r', hi: '\r');


        if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
            throw new RuntimeException(message: "Отсутствует количество точек");
        }
        int pointsCount = Integer.parseInt(tokenizer.sval);
```

```
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD){
                throw new RuntimeException (message: "Отсутствует координата x");
            }
            double x = Double.parseDouble(tokenizer.sval);
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD){
                throw new RuntimeException (message: "Отсутствует координата y");
            }
            double y = Double.parseDouble(tokenizer.sval);
            points[i] = new FunctionPoint(x, y);
        }

        return new ArrayTabulatedFunction(points);

    } catch (IOException e) {
        throw new RuntimeException(message: "Ошибка при чтении", e);
    } catch(NumberFormatException e) {
        throw new RuntimeException(message: "Неверный формат числа", e);
    }
}
```

Использованы:

1. **DataInputStream и DataOutputStream** — классы в Java, которые используются для чтения и записи примитивных типов данных из и в поток соответственно.
2. writeInt() - для записи объектов типа int (4 байта), writeDouble() - 8 байт
3. flush() - если есть данные, которые хранятся где-то внутри и еще не записаны, то они записываются
4. readInt(), readDouble() - чтение объектов типа int и double
5. **Reader и Writer** - классы в Java, которые используются для чтения и записи символьных типов данных
6. PrintWriter — класс в пакете java.io, который используется для вывода данных в текстовом формате
7. write() - для записи в текстовый файл;
8. Класс StreamTokenizer принимает входной поток и разбивает его на «токены», что позволяет считывать токены по одному
9. wordChars() - считывает цифры, точку, минус, e (E) - для степеней
10. whitespaceChars() - показывает, что является разделителем
11. tokenizer.nextToken() - читает следующий "кусочек" текста
12. StreamTokenizer.TT_WORD - является ли словом (числом)
13. sval - получаем прочитанное слово как строку
14. parseInt/parseDouble - преобразование строки в int/double
15. IOException -ошибка, указывающая на проблемы с файлом (не найден, нет прав и т.д.). В качестве решения преобразуем IOException в RuntimeException
16. NumberFormatException - в файле нет числа там, где должно быть число

При использовании try{}поток закроется при выходе из блока (вызывающий код управляет жизненным циклом потока)

Задание 8

Реализован класс Main для проверки работы разработанных классов с методами:

- printFunction() - для вывода значений функции в виде таблицы
- compareFunctions() - для сравнения двух функций и выявления разницы
- testingSinAndCos()тестирует синус и косинус
- testingTabulatedSinAndCos() создает табличные версии и сравнивает с оригиналами
- testingSumOfSquares() проверяет формулу $\sin^2 + \cos^2 = 1$
- testingTabulatedExp() работает с экспонентой и текстовым файлом
- testingTabulatedLog( ) работает с логарифмом и бинарным файлом

```java
import functions.*;
import functions.basic.*;
import functions.meta.*;

import java.io.*;

public class Main {
    Run main | Debug main
    public static void main(String[] args) {
        System.out.println(x: "sin/cos [0, pi]");
        testingSinAndCos();

        System.out.println(x: "\n_tabulated analog of sin/cos [0, pi]_");
        testingTabulatedSinAndCos();

        System.out.println(x: "\n_sum sin^2 + cos^2_");
        testingSumOfSquares();

        System.out.println(x: "\n_tabulated analog of exp [0, 10]_");
        testingTabulatedExp();

        System.out.println(x: "\n_tabulated analog of log [0,10]_");
        testingTabulatedLog();

        System.out.println(x: "\n_testing Serializable_");
        testingSerializable();

        // Тест с Externalizable (если создали такой класс)
        System.out.println(x: "\n_testing Externalizable_");
        testingExternalizable();
    }
```

```java
    private  static void testingSinAndCos() {
        Sin sin = new Sin();
        Cos cos = new Cos();

        System.out.println(x: "\n___ sin ___");
        printFunction(sin, leftX: 0, Math.PI, step: 0.1);
        System.out.println(x: "\n___ cos ___");
        printFunction(cos, leftX: 0, Math.PI, step: 0.1);
    }
```

```java
private  static void testingTabulatedSinAndCos() {
    Sin sin = new Sin();
    Cos cos = new Cos();
    TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
    TabulatedFunction tabulatedCos = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);

    System.out.println(x: "\n___ tabulated sin ___");
    printFunction(tabulatedSin, leftX: 0, Math.PI, step: 0.1);
    System.out.println(x: "\n___ tabulated cos ___");
    printFunction(tabulatedCos, leftX: 0, Math.PI, step: 0.1);
    System.out.println(x: "\n___ comparison sin ____");
    compareFunctions(sin, tabulatedSin, leftX: 0, Math.PI, step: 0.1);
    System.out.println(x: "\n___ comparison cos ____");
    compareFunctions(cos, tabulatedCos, leftX: 0, Math.PI, step: 0.1);
}

private static void testingSumOfSquares() {
    int[] pointCounts = {5, 10, 30, 50};
    for (int pointsCount : pointCounts) {
        System.out.println("Number of points: " + pointsCount);
        Sin sin = new Sin();
        Cos cos = new Cos();
        TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
        TabulatedFunction tabulatedCos = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);
        Function squaredSin = Functions.power(tabulatedSin, power: 2);
        Function squaredCos = Functions.power(tabulatedCos, power: 2);
        Function sum = Functions.sum(squaredSin, squaredCos);
        printFunction(sum, leftX: 0, Math.PI, step: 0.1);
    }

}

private static void testingTabulatedExp(){
    try {
        Exp exp = new Exp();
        TabulatedFunction tabulatedExp = TabulatedFunctions.tabulate(exp, leftX: 0, rightX: 10, pointsCount: 11);
        System.out.println(x: "\nwritten to file exp.txt");
        FileWriter writer = new FileWriter(fileName: "exp.txt");
        TabulatedFunctions.writeTabulatedFunction(tabulatedExp, writer);
        writer.close();

        System.out.println(x: "\nread from file exp.txt");
        FileReader reader = new FileReader(fileName: "exp.txt");
        TabulatedFunction readExp = TabulatedFunctions.readTabulatedFunction(reader);
        reader.close();

        System.out.println(x: "comparison between exp and read exp");
        compareFunctions(exp, readExp, leftX: 0,rightX: 10, step: 1);
    } catch (IOException e) {
        throw new RuntimeException(message: "Ошибка при работе с файлом", e);
    }
```

```java
private static void testingTabulatedLog() {
    try{
        Log log = new Log(Math.E);
        TabulatedFunction tabulatedLog = TabulatedFunctions.tabulate(log, leftX: 1,rightX: 10, pointsCount: 10);

        System.out.println(x: "\n___ written to binary file log.dat___");
        FileOutputStream outStream = new FileOutputStream(name: "log.dat");
        TabulatedFunctions.outputTabulatedFunction(tabulatedLog, outStream);
        outStream.close();

        System.out.println(x: "\nread from file log.dat");
        FileInputStream inStream = new FileInputStream(name: "log.dat");
        TabulatedFunction readLog = TabulatedFunctions.inputTabulatedFunction(inStream);
        inStream.close();

        System.out.println(x: "comparison between log and read log");
        compareFunctions(log, readLog, leftX: 1, rightX: 10, step: 1);

    } catch (IOException e) {
        throw new RuntimeException (message: "Ошибка при работе с файлом", e);
    }
}

private static void printFunction(Function f, double leftX, double rightX, double step) {
    System.out.println(x: " x         f(x)");
    System.out.println(x: "--------------------");
    for (double x = leftX; x <= rightX; x += step) {
        double y = f.getFunctionValue(x);

        String strX = String.format(format: "%6.3f", x);
        String strY;

        if (Double.isNaN(y)) {
            strY = "        NaN";
        } else {
            strY = String.format(format: "%12.6f", y);
        }

        System.out.println(strX + "  " + strY);
    }
}
```

```java
private static void  compareFunctions(Function f_1, Function f_2, double leftX, double  rightX, double step) {
    System.out.println(x: " x         f1(x)        f2(x)        difference");
    System.out.println(x: "-------------------------------------------------");

    for (double x = leftX; x <= rightX; x += step) {
        double y1 = f_1.getFunctionValue(x);
        double y2 = f_2.getFunctionValue(x);
        String strX = String.format(format: "%6.3f", x);
        String strY1, strY2, diffStr;
        if (Double.isNaN(y1)) {

            strY1 = "        NaN";
        } else {
            strY1 = String.format(format: "%12.6f", y1);
        }
```

```
        if (Double.isNaN(y2)) {
            strY2 = "         NaN";
        } else {
            strY2 = String.format(format: "%12.6f", y2);
        }

        if (!Double.isNaN(y1) && !Double.isNaN(y2)) {
            double diff = Math.abs(y1 - y2);
            diffStr = String.format(format: "%12.6f", diff);
        } else if (Double.isNaN(y1) && Double.isNaN(y2)) {
            diffStr = "        0.000";
        } else {
            diffStr = "          ---";
        }

        System.out.println(strX + "   " + strY1 + "   " + strY2 + "   " + diffStr);
    }
}
```

Выводы:

- Табулированные версии близки к оригиналам: чем больше точек, тем точнее аппроксимация
- Там, где точек мало, значения немного отличаются

Текстовый формат легко открыть в любом редакторе, он читаем и универсален, но занимает больше места, чем бинарный, он медленнее и менее точный и эффективный.

Текстовый формат подходит для:

- Сохранения пользовательских настроек
- Экспорта данных в другие программы
- Отладки и тестирования

Бинарный формат подходит для:

- Внутреннего хранения данных
- Быстрой сериализации объектов
- Работы с большими массивами точек

Задание 9

Реализована сериализация объектов всех классов, реализующих интерфейс TabulatedFunction

Сериализация — это процесс преобразования объекта в последовательность байтов, которая может быть сохранена в файле или передана по сети. Десериализация — обратный процесс восстановления объекта из этой последовательности.

private static final long serialVersionUID содержит уникальный идентификатор версии сериализованного класса.

java.io.Serializable:

```
package functions;

import java.io.Serializable;

public class FunctionPoint implements Serializable{

    private static final long serialVersionUID = 1L;
    private double x;
    private double y;
```

```
package functions;

import java.io.Serializable;

public class ArrayTabulatedFunction implements TabulatedFunction, Serializable {
    private static final long serialVersionUID = 2L;

    private FunctionPoint[] points;
    private int pointsCount;
    private static final double EPSILON = 1e-9;

    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount){
        if (leftX >= rightX) {
            throw new IllegalArgumentException(s: "Левая граница области определения должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException(s: "Количество точек должно быть >=2");
        }

        points = new FunctionPoint[pointsCount + 2];
        this.pointsCount = pointsCount;
        double distance = (rightX - leftX)/(pointsCount - 1);

        for (int i = 0; i < pointsCount; i++){
            double x = leftX + i * distance;
            points[i] = new FunctionPoint(x, y: 0.0);
        }
    }
```

```java
package functions;

import java.io.Serializable;

public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable {
    private static final long serialVersionUID = 3L;

    private class FunctionNode implements Serializable {
        private static final long serialVersionUID = 4L;
        private FunctionPoint point;
        private FunctionNode previous;
        private FunctionNode next;

        public FunctionNode(FunctionPoint point){
            this.point = point;
            this.previous = null;
            this.next = null;
        }

        public FunctionNode(FunctionPoint point, FunctionNode previous, FunctionNode next){
            this.point = point;
            this.previous = previous;
            this.next = next;
        }

        public FunctionPoint getPoint() {
            return point;
        }

        public void setPoint(FunctionPoint point) {
            this.point = point;
        }

        public FunctionNode getPrevious() {
            return previous;
        }

        public FunctionNode getNext() {
            return next;
        }

        public void setPrevious(FunctionNode previous) {
            this.previous = previous;
        }

        public void setNext(FunctionNode next) {
            this.next = next;
        }
    }
```

java.io.Serializable: при использовании интерфейса Externalizable необходимо реализовать два обязательных метода — **writeExternal()** и **readExternal()**.

```java
package functions;

import java.io.*;

public class ArrayTabulatedFunctionExternalizable implements TabulatedFunction, Externalizable {

    private FunctionPoint[] points;
    private int pointsCount;
    private static final double EPSILON = 1e-9;

    public ArrayTabulatedFunctionExternalizable() {
        this.points = new FunctionPoint[2];
        this.pointsCount = 0;
    }

    public ArrayTabulatedFunctionExternalizable(double leftX, double rightX, int pointsCount){
        if (leftX >= rightX) {
            throw new IllegalArgumentException(s: "Левая граница области определения должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException(s: "Количество точек должно быть >=2");
        }

        points = new FunctionPoint[pointsCount + 2];
        this.pointsCount = pointsCount;
        double distance = (rightX - leftX)/(pointsCount - 1);

        for (int i = 0; i < pointsCount; i++){
            double x = leftX + i * distance;
            points[i] = new FunctionPoint(x, y: 0.0);
        }
    }

    public ArrayTabulatedFunctionExternalizable(double leftX, double rightX, double[] values){

        if (leftX >= rightX) {
            throw new IllegalArgumentException(s: "Левая граница области определения должна быть меньше правой");
        }
        if (values.length < 2) {
            throw new IllegalArgumentException(s: "Количество точек должно быть >=2");
        }

        this.pointsCount = values.length;
        points = new FunctionPoint[pointsCount + 2];
        double distance = (rightX - leftX)/(pointsCount - 1);

        for (int i = 0; i < pointsCount; i++){
            double x = leftX + i * distance;
            points[i] = new FunctionPoint(x, values[i]);
        }
    }
```

```java
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(pointsCount);
    for (int i = 0; i < pointsCount; i++) {
        out.writeDouble(points[i].getX());
        out.writeDouble(points[i].getY());
    }
}

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    pointsCount = in.readInt();
    points = new FunctionPoint[pointsCount + 2];
    for (int i = 0; i < pointsCount; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        points[i] = new FunctionPoint(x, y);
    }
```

```java
package functions;

import java.io.*;

public class FunctionPointExternalizable implements Externalizable{
    private double x;
    private double y;

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

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


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

1. Serializable (автоматическая сериализация)

Преимущества:

- Простота реализации (только implements Serializable)
- Автоматическая сериализация всей графы объектов
- Не требует дополнительного кода для простых случаев

2. Externalizable (ручная сериализация)

Преимущества:

- Полный контроль над процессом
- Меньший размер файла (только нужные данные)
- Высокая производительность и легкость изменения формата хранения

Консоль:

```
lisako@lisa MINGW64 ~/Documents/oop/Lab-4-2025 (main)
$ javac functions/*.java

$ javac Main.java

$ java Main
sin/cos [0, pi]

___ sin ___
 x         f(x)
-------------------
 0,000      0,000000
 0,100      0,099833
 0,200      0,198669
 0,300      0,295520
 0,400      0,389418
 0,500      0,479426
 0,600      0,564642
 0,700      0,644218
 0,800      0,717356
 0,900      0,783327
 1,000      0,841471
 1,100      0,891207
 1,200      0,932039
 1,300      0,963558
 1,400      0,985450
 1,500      0,997495
 1,600      0,999574
 1,700      0,991665
 1,800      0,973848
 1,900      0,946300
 2,000      0,909297
 2,100      0,863209
 2,200      0,808496
 2,300      0,745705
 2,400      0,675463
 2,500      0,598472
 2,600      0,515501
 2,700      0,427380
 2,800      0,334988
 2,900      0,239249
 3,000      0,141120
 3,100      0,041581

___ cos ___
 x         f(x)
-------------------
 0,000      1,000000
 0,100      0,995004
 0,200      0,980067
 0,300      0,955336
 0,400      0,921061
 0,500      0,877583
 0,600      0,825336
 0,700      0,764842
 0,800      0,696707
 0,900      0,621610
 1,000      0,540302
 1,100      0,453596
 1,200      0,362358
 1,300      0,267499
 1,400      0,169967
 1,500      0,070737
 1,600     -0,029200
 1,700     -0,128844
 1,800     -0,227202
 1,900     -0,323290
 2,000     -0,416147
 2,100     -0,504846
 2,200     -0,588501
 2,300     -0,666276
 2,400     -0,737394
 2,500     -0,801144
 2,600     -0,856889
 2,700     -0,904072
 2,800     -0,942222
 2,900     -0,970958
 3,000     -0,989992
 3,100     -0,999135

_tabulated analog of sin/cos [0, pi]_
```

___ tabulated sin ___

x	f(x)
0,000	0,000000
0,100	0,097982
0,200	0,195963
0,300	0,293945
0,400	0,385907
0,500	0,472070
0,600	0,558234
0,700	0,643982
0,800	0,707935
0,900	0,771888
1,000	0,835841
1,100	0,883993
1,200	0,918022
1,300	0,952051
1,400	0,984808
1,500	0,984808
1,600	0,984808
1,700	0,984808
1,800	0,966204
1,900	0,932175
2,000	0,898147
2,100	0,862441
2,200	0,798488
2,300	0,734535
2,400	0,670582
2,500	0,594072
2,600	0,507908
2,700	0,421745
2,800	0,334698
2,900	0,236716
3,000	0,138735
3,100	0,040753

___ tabulated cos ___

x	f(x)
0,000	1,000000
0,100	0,982723
0,200	0,965446
0,300	0,948170
0,400	0,914355
0,500	0,864608
0,600	0,814862
0,700	0,764620
0,800	0,688404
0,900	0,612188
1,000	0,535972
1,100	0,450633
1,200	0,357141
1,300	0,263648
1,400	0,169931
1,500	0,070437
1,600	-0,029056
1,700	-0,128549
1,800	-0,224761
1,900	-0,318254
2,000	-0,411747
2,100	-0,504272
2,200	-0,580488
2,300	-0,656704
2,400	-0,732920
2,500	-0,794171
2,600	-0,843917
2,700	-0,893664
2,800	-0,940984
2,900	-0,958261
3,000	-0,975537
3,100	-0,992814

___ comparison sin ____

x	f1(x)	f2(x)	difference
0,000	0,000000	0,000000	0,000000
0,100	0,099833	0,097982	0,001852
0,200	0,198669	0,195963	0,002706
0,300	0,295520	0,293945	0,001576
0,400	0,389418	0,385907	0,003512
0,500	0,479426	0,472070	0,007355
0,600	0,564642	0,558234	0,006409
0,700	0,644218	0,643982	0,000235

0,800	0,717356	0,707935	0,009421
0,900	0,783327	0,771888	0,011439
1,000	0,841471	0,835841	0,005630
1,100	0,891207	0,883993	0,007214
1,200	0,932039	0,918022	0,014017
1,300	0,963558	0,952051	0,011508
1,400	0,985450	0,984808	0,000642
1,500	0,997495	0,984808	0,012687
1,600	0,999574	0,984808	0,014766
1,700	0,991665	0,984808	0,006857
1,800	0,973848	0,966204	0,007644
1,900	0,946300	0,932175	0,014125
2,000	0,909297	0,898147	0,011151
2,100	0,863209	0,862441	0,000768
2,200	0,808496	0,798488	0,010008
2,300	0,745705	0,734535	0,011170
2,400	0,675463	0,670582	0,004881
2,500	0,598472	0,594072	0,004401
2,600	0,515501	0,507908	0,007593
2,700	0,427380	0,421745	0,005635
2,800	0,334988	0,334698	0,000290
2,900	0,239249	0,236716	0,002533
3,000	0,141120	0,138735	0,002385
3,100	0,041581	0,040753	0,000828

___ comparison cos ____

x	f1(x)	f2(x)	difference
0,000	1,000000	1,000000	0,000000
0,100	0,995004	0,982723	0,012281
0,200	0,980067	0,965446	0,014620
0,300	0,955336	0,948170	0,007167
0,400	0,921061	0,914355	0,006706
0,500	0,877583	0,864608	0,012974
0,600	0,825336	0,814862	0,010474
0,700	0,764842	0,764620	0,000222
0,800	0,696707	0,688404	0,008302
0,900	0,621610	0,612188	0,009422
1,000	0,540302	0,535972	0,004330
1,100	0,453596	0,450633	0,002963
1,200	0,362358	0,357141	0,005217
1,300	0,267499	0,263648	0,003851
1,400	0,169967	0,169931	0,000037
1,500	0,070737	0,070437	0,000300
1,600	-0,029200	-0,029056	0,000144
1,700	-0,128844	-0,128549	0,000296
1,800	-0,227202	-0,224761	0,002441
1,900	-0,323290	-0,318254	0,005035
2,000	-0,416147	-0,411747	0,004400
2,100	-0,504846	-0,504272	0,000574
2,200	-0,588501	-0,580488	0,008013
2,300	-0,666276	-0,656704	0,009572
2,400	-0,737394	-0,732920	0,004474
2,500	-0,801144	-0,794171	0,006973
2,600	-0,856889	-0,843917	0,012972
2,700	-0,904072	-0,893664	0,010408
2,800	-0,942222	-0,940984	0,001239
2,900	-0,970958	-0,958261	0,012698
3,000	-0,989992	-0,975537	0,014455
3,100	-0,999135	-0,992814	0,006321

sum sin^2 + cos^2
Number of points: 5

x	f(x)
0,000	1,000000
0,100	0,975345
0,200	0,970488
0,300	0,985429
0,400	0,984968
0,500	0,970398
0,600	0,975624
0,700	0,999358
0,800	0,975073
0,900	0,970586
1,000	0,985897
1,100	0,984515
1,200	0,970314
1,300	0,975910
1,400	0,998723
1,500	0,974808
1,600	0,970691
1,700	0,986371
1,800	0,984068

```
 1,900      0,970237
 2,000      0,976203
 2,100      0,998094
 2,200      0,974549
 2,300      0,970802
 2,400      0,986852
 2,500      0,983628
 2,600      0,970167
 2,700      0,976503
 2,800      0,997473
 2,900      0,974298
 3,000      0,970920
 3,100      0,987341
Number of points: 10
 x         f(x)
------------------
 0,000      1,000000
 0,100      0,975345
 0,200      0,970488
 0,300      0,985429
 0,400      0,984968
 0,500      0,970398
 0,600      0,975624
 0,700      0,999358
 0,800      0,975073
 0,900      0,970586
 1,000      0,985897
 1,100      0,984515
 1,200      0,970314
 1,300      0,975910
 1,400      0,998723
 1,500      0,974808
 1,600      0,970691
 1,700      0,986371
 1,800      0,984068
 1,900      0,970237
 2,000      0,976203
 2,100      0,998094
 2,200      0,974549
 2,300      0,970802
 2,400      0,986852
 2,500      0,983628
 2,600      0,970167
 2,700      0,976503
 2,800      0,997473
 2,900      0,974298
 3,000      0,970920
 3,100      0,987341
Number of points: 30
 x         f(x)
------------------
 0,000      1,000000
 0,100      0,975345
 0,200      0,970488
 0,300      0,985429
 0,400      0,984968
 0,500      0,970398
 0,600      0,975624
 0,700      0,999358
 0,800      0,975073
 0,900      0,970586
 1,000      0,985897
 1,100      0,984515
 1,200      0,970314
 1,300      0,975910
 1,400      0,998723
 1,500      0,974808
 1,600      0,970691
 1,700      0,986371
 1,800      0,984068
 1,900      0,970237
 2,000      0,976203
 2,100      0,998094
 2,200      0,974549
 2,300      0,970802
 2,400      0,986852
 2,500      0,983628
 2,600      0,970167
 2,700      0,976503
 2,800      0,997473
 2,900      0,974298
 3,000      0,970920
 3,100      0,987341
Number of points: 50
```

```
 x         f(x)
-------------------
 0,000      1,000000
 0,100      0,975345
 0,200      0,970488
 0,300      0,985429
 0,400      0,984968
 0,500      0,970398
 0,600      0,975624
 0,700      0,999358
 0,800      0,975073
 0,900      0,970586
 1,000      0,985897
 1,100      0,984515
 1,200      0,970314
 1,300      0,975910
 1,400      0,998723
 1,500      0,974808
 1,600      0,970691
 1,700      0,986371
 1,800      0,984068
 1,900      0,970237
 2,000      0,976203
 2,100      0,998094
 2,200      0,974549
 2,300      0,970802
 2,400      0,986852
 2,500      0,983628
 2,600      0,970167
 2,700      0,976503
 2,800      0,997473
 2,900      0,974298
 3,000      0,970920
 3,100      0,987341

_tabulated analog of exp [0, 10]_

written to file exp.txt

read from file exp.txt
comparison between exp and read exp
 x         f1(x)         f2(x)        difference
-------------------------------------------------
 0,000      1,000000      1,000000      0,000000
 1,000      2,718282      2,718282      0,000000
 2,000      7,389056      7,389056      0,000000
 3,000     20,085537     20,085537      0,000000
 4,000     54,598150     54,598150      0,000000
 5,000    148,413159    148,413159      0,000000
 6,000    403,428793    403,428793      0,000000
 7,000   1096,633158   1096,633158      0,000000
 8,000   2980,957987   2980,957987      0,000000
 9,000   8103,083928   8103,083928      0,000000
10,000  22026,465795  22026,465795      0,000000

_tabulated analog of log [0,10]_

___ written to binary file log.dat___

read from file log.dat
comparison between log and read log
 x         f1(x)         f2(x)        difference
-------------------------------------------------
 1,000      0,000000      0,000000      0,000000
 2,000      0,693147      0,693147      0,000000
 3,000      1,098612      1,098612      0,000000
 4,000      1,386294      1,386294      0,000000
 5,000      1,609438      1,609438      0,000000
 6,000      1,791759      1,791759      0,000000
 7,000      1,945910      1,945910      0,000000
 8,000      2,079442      2,079442      0,000000
 9,000      2,197225      2,197225      0,000000
10,000      2,302585      2,302585      0,000000

_testing Serializable_
(ln(e^x) = x):
 x         f(x)
-------------------
 0,000      0,000000
 1,000      1,000000
 2,000      2,000000
 3,000      3,000000
 4,000      4,000000
 5,000      5,000000
```

```
 6,000      6,000000
 7,000      7,000000
 8,000      8,000000
 9,000      9,000000
10,000     10,000000

Serializable to file 'serializable.ser'...
from file 'serializable.ser'...
 x         f(x)
-------------------
 0,000      0,000000
 1,000      1,000000
 2,000      2,000000
 3,000      3,000000
 4,000      4,000000
 5,000      5,000000
 6,000      6,000000
 7,000      7,000000
 8,000      8,000000
 9,000      9,000000
10,000     10,000000

compare:
 x         f1(x)         f2(x)         difference
-------------------------------------------------
 0,000      0,000000      0,000000      0,000000
 1,000      1,000000      1,000000      0,000000
 2,000      2,000000      2,000000      0,000000
 3,000      3,000000      3,000000      0,000000
 4,000      4,000000      4,000000      0,000000
 5,000      5,000000      5,000000      0,000000
 6,000      6,000000      6,000000      0,000000
 7,000      7,000000      7,000000      0,000000
 8,000      8,000000      8,000000      0,000000
 9,000      9,000000      9,000000      0,000000
10,000     10,000000     10,000000      0,000000

_testing Externalizable_
 x         f(x)
-------------------
 0,000      0,000000
 1,000      1,000000
 2,000      2,000000
 3,000      3,000000
 4,000      4,000000
 5,000      5,000000
 6,000      6,000000
 7,000      7,000000
 8,000      8,000000
 9,000      9,000000
10,000     10,000000

Externalizableto file 'externalizable.ser'...
from file 'externalizable.ser'...
Десериализованная функция:
 x         f(x)
-------------------
 0,000      0,000000
 1,000      1,000000
 2,000      2,000000
 3,000      3,000000
 4,000      4,000000
 5,000      5,000000
 6,000      6,000000
 7,000      7,000000
 8,000      8,000000
 9,000      9,000000
10,000     10,000000

compare:
 x         f1(x)         f2(x)         difference
-------------------------------------------------
 0,000      0,000000      0,000000      0,000000
 1,000      1,000000      1,000000      0,000000
 2,000      2,000000      2,000000      0,000000
 3,000      3,000000      3,000000      0,000000
 4,000      4,000000      4,000000      0,000000
 5,000      5,000000      5,000000      0,000000
 6,000      6,000000      6,000000      0,000000
 7,000      7,000000      7,000000      0,000000
 8,000      8,000000      8,000000      0,000000
 9,000      9,000000      9,000000      0,000000
10,000     10,000000     10,000000      0,000000
```