

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА»

Институт «Информатики и кибернетики»

Специальность «Фотоника и оптоинформатика 6201-120303D»

Отчет по лабораторной работе № 4

Выполнил: студент Султанова А.М.,

группа 6201-120303D

Проверил: преподаватель Борисов Д.С.

Самара 2025

## Задание 1

В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` добавляются конструкторы, которые принимают массив объектов `FunctionPoint`, содержащий все точки функции. Если в переданном массиве содержится менее двух точек, либо если значения абсцисс точек не образуют строго возрастающую последовательность, конструкторы генерируют исключение `IllegalArgumentException`.

```
public ArrayTabulatedFunction(FunctionPoint[] points) { 6 usages new*
    if (points == null) {
        throw new IllegalArgumentException("Массив точек не может быть null");
    }
    if (points.length < 2) {
        throw new IllegalArgumentException("Должно быть не менее двух точек");
    }

    int n = points.length;
    this.points = new FunctionPoint[n];
    this.size = n;

    for (int i = 0; i < n; i++) {
        if (points[i] == null) {
            throw new IllegalArgumentException("Точка не может быть null (index = " + i + ")");
        }

        this.points[i] = new FunctionPoint(points[i]);
        if (i > 0) {
            double prevX = this.points[i - 1].getX();
            double curX = this.points[i].getX();
            if (!(curX > prevX + EPS)) {
                throw new IllegalArgumentException("Массив точек должен быть упорядочен по X (ошибка на индексе " + i + ")");
            }
        }
    }
}
```

```
public LinkedListTabulatedFunction(FunctionPoint[] points) { 22 usages new*
    if (points == null) {
        throw new IllegalArgumentException("Массив точек не может быть null");
    }
    if (points.length < 2) {
        throw new IllegalArgumentException("Должно быть не менее двух точек");
    }

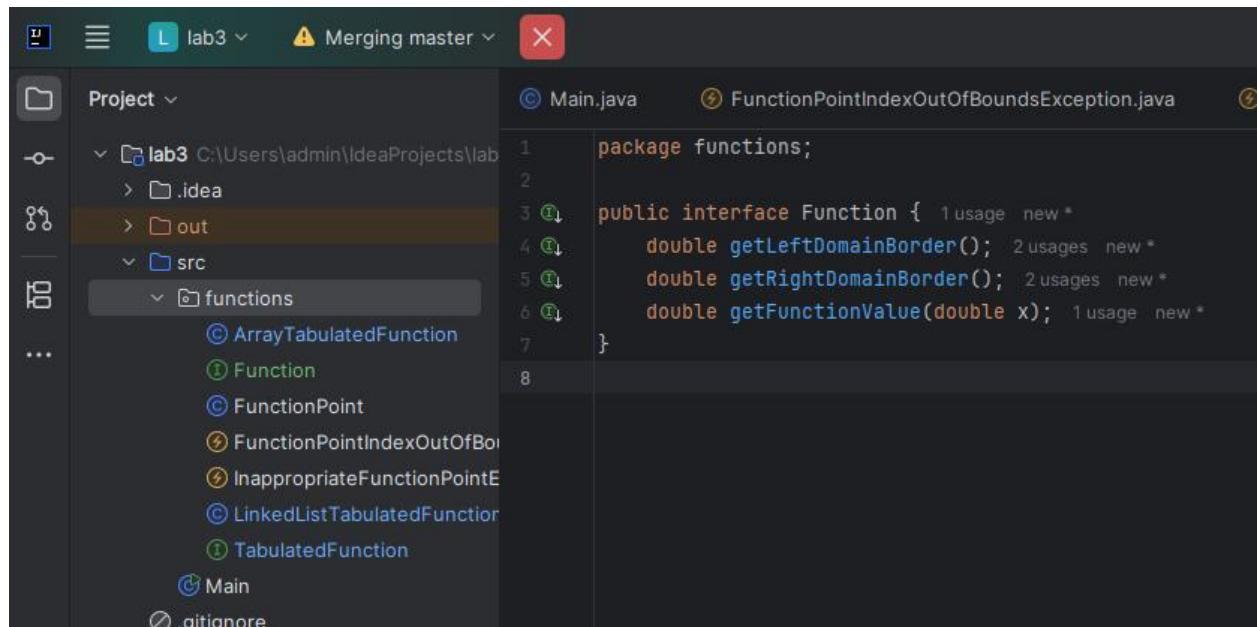
    this.head = new FunctionNode( point: null);
    head.next = head;
    head.prev = head;
    this.size = 0;

    for (int i = 0; i < points.length; i++) {
        if (points[i] == null) {
            throw new IllegalArgumentException("Точка не может быть null (index = " + i + ")");
        }
        FunctionPoint copy = new FunctionPoint(points[i]);
        if (i > 0) {
            double prevX = head.prev.point.getX();
            double curX = copy.getX();
            if (!(curX > prevX + EPS)) {
                throw new IllegalArgumentException("Массив точек должен быть упорядочен по X (ошибка на индексе " + i + ")");
            }
        }

        FunctionNode newNode = addNodeToTail();
        newNode.point = copy;
    }
}
```

## Задание 2

Создаем новый файл Function.java, описывающий функции одной переменной и содержащий следующие методы:



Добавим наследование в TabulatedFunction.java и исключим эти методы из него

```
package functions;

public interface TabulatedFunction extends Function {
    int getPointsCount();
    double getPointX(int index);
    double getPointY(int index);
    void setPointX(int index, double x) throws InappropriateFunctionPointException;
    void setPointY(int index, double y);
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
    void deletePoint(int index);
}
```

## Задание 3

Создаём пакет `functions.basic` , в котором необходимо создать публичный класс `Exp` , который реализует интерфейс `Function` и представляет экспоненциальную функцию.

Требования к реализации:

- Вычисление значений функции должно выполняться с помощью метода `Math.exp()`
- Границы области определения следует задавать с использованием констант класса `Double`

```
package functions.basic;
import functions.Function;
public class Exp implements Function { no usages new *
    public double getLeftDomainBorder() { 2 usages new *
        return -Double.MAX_VALUE;
    }
    public double getRightDomainBorder() { 2 usages new *
        return Double.MAX_VALUE;
    }
    public double getFunctionValue(double x) { 1 usage new *
        return Math.exp(x);
    }
}
```

В пакете `functions.basic` создайте класс `Log` для вычисления логарифмической функции с произвольным основанием.

```
package functions.basic;
import functions.Function;
public class Log implements Function { no usages new *
    private final double base; 2 usages
    public Log(double base) { no usages new *
        if (base <= 0 || base == 1)
            throw new IllegalArgumentException("Основание логарифма должно быть > 0 и ≠ 1");
        this.base = base;
    }
    public double getLeftDomainBorder() { 2 usages new *
        return 0;
    }
    public double getRightDomainBorder() { 2 usages new *
        return Double.MAX_VALUE;
    }
    public double getFunctionValue(double x) { 1 usage new *
        if (x <= 0) return Double.NaN;
        return Math.log(x) / Math.log(base);
    }
}
```

Создаём класс `TrigonometricFunction`, реализующий интерфейс `Function` и описывающий методы получения границ области определения.

```
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {

    public double getLeftDomainBorder() { 2 usages new *
        return -Double.MAX_VALUE;
    }

    public double getRightDomainBorder() { 2 usages new *
        return Double.MAX_VALUE;
    }
}
```

Создаём в пакете functions.basic публичные классы Sin, Cos и Tan, которые реализуют тригонометрические функции.

Требования к реализации:

Класс Sin должен вычислять значение синуса с помощью метода Math.sin()

Класс Cos должен вычислять значение косинуса с помощью метода Math.cos()

Класс Tan должен вычислять значение тангенса с помощью метода Math.tan()

```
package functions.basic;

public class Sin extends TrigonometricFunction { no usages new *

    public double getFunctionValue(double x) { 1 usage new *
        return Math.sin(x);
    }
}
```

```
package functions.basic;

public class Cos extends TrigonometricFunction { no usages new *

    public double getFunctionValue(double x) { 1 usage new *
        return Math.cos(x);
    }
}
```

```
package functions.basic;

public class Tan extends TrigonometricFunction { no usages new *

    public double getFunctionValue(double x) { 1 usage new *
        return Math.tan(x);
    }
}
```

#### Задание 4

Создаём пакет meta, в котором создаем класс Sum, который описывает функцию  $f(x) = f_1(x) + f_2(x)$ .

Область определения — пересечение областей определения двух функций.

The screenshot shows a Java project structure in an IDE. The project is named 'lab3' and contains packages 'functions.meta' and 'functions.basic'. The 'functions.basic' package includes classes for basic mathematical functions like Cos, Exp, Log, Sin, Tan, and TrigonometricFunction. A file named 'Sum.java' is open, which implements the Function interface. It contains methods for getting domain borders and calculating function values for two input functions f1 and f2.

```

1 package functions.meta;
2
3 import functions.Function;
4
5 public class Sum implements Function { no usages new *
6
7     private final Function f1; 4 usages
8     private final Function f2; 4 usages
9
10    public Sum(Function f1, Function f2) { no usages new *
11        this.f1 = f1;
12        this.f2 = f2;
13    }
14
15    public double getLeftDomainBorder() { 4 usages new *
16        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
17    }
18
19    public double getRightDomainBorder() { 4 usages new *
20        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
21    }
22
23    public double getFunctionValue(double x) { 3 usages new *
24        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
25    }
26}

```

Также создаём класс Mult, который описывает функцию  $f(x) = f_1(x) * f_2(x)$ .

Область определения — пересечение областей исходных функций.

The screenshot shows the code for a 'Mult' class. This class also implements the 'Function' interface and takes two input functions f1 and f2. It overrides the methods for getting domain borders and calculating the function value by multiplying the results from f1 and f2.

```

1 package functions.meta;
2
3 import functions.Function;
4
5 public class Mult implements Function { no usages new *
6
7     private Function f1; 4 usages
8     private Function f2; 4 usages
9
10    public Mult(Function f1, Function f2) { no usages new *
11        this.f1 = f1;
12        this.f2 = f2;
13    }
14
15    public double getLeftDomainBorder() { 6 usages new *
16        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
17    }
18
19    public double getRightDomainBorder() { 6 usages new *
20        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
21    }
22
23    public double getFunctionValue(double x) { new *
24        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
25    }
26}

```

Еще класс Power, который описывает функцию  $f(x) = (g(x))^n$ .

Область определения совпадает с областью исходной функции.

```
package functions.meta;

import functions.Function;

public class Power implements Function { no usages new *

    private Function baseFunction; 4 usages
    private double power; 2 usages

    public Power(Function baseFunction, double power) { no usages new *
        this.baseFunction = baseFunction;
        this.power = power;
    }

    public double getLeftDomainBorder() { 7 usages new *
        return baseFunction.getLeftDomainBorder();
    }

    public double getRightDomainBorder() { 7 usages new *
        return baseFunction.getRightDomainBorder();
    }

    public double getFunctionValue(double x) { new *
        return Math.pow(baseFunction.getFunctionValue(x), power);
    }
}
```

Далее класс Scale, он Описывает функцию:  $f(x) = ky * g(x / kx)$ .  
Область определения — область исходной функции, растянутая по x на kx.

```
package functions.meta;

import functions.Function;

public class Scale implements Function { no usages new *

    private Function f; 4 usages
    private double kx; 4 usages
    private double ky; 2 usages

    public Scale(Function f, double kx, double ky) { no usages
        this.f = f;
        this.kx = kx;
        this.ky = ky;
    }

    public double getLeftDomainBorder() { 8 usages new *
        return f.getLeftDomainBorder() * kx;
    }

    public double getRightDomainBorder() { 8 usages new *
        return f.getRightDomainBorder() * kx;
    }

    public double getFunctionValue(double x) { new *
        return ky * f.getFunctionValue( x: x / kx);
    }
}
```

Потом класс Shift, который описывает функцию:  $f(x) = g(x - dx) + dy$ .

Область определения — исходная область, сдвинутая вдоль оси x.

```
package functions.meta;

import functions.Function;

public class Shift implements Function { no usages new *

    private Function f; 4 usages
    private double dx; 4 usages
    private double dy; 2 usages

    public Shift(Function f, double dx, double dy) { no
        this.f = f;
        this.dx = dx;
        this.dy = dy;
    }

    public double getLeftDomainBorder() { new *
        return f.getLeftDomainBorder() + dx;
    }

    public double getRightDomainBorder() { new *
        return f.getRightDomainBorder() + dx;
    }

    public double getFunctionValue(double x) { new *
        return f.getFunctionValue( x: x - dx) + dy;
    }
}
```

Далее класс Composition, это Композиция:  $f(x) = f_1(f_2(x))$ .  
Область определения функции можно считать совпадающей с областью определения исходной функции  $f_2$

```
package functions.meta;

import functions.Function;

public class Composition implements Function { no usages new *

    private Function outer; 2 usages
    private Function inner; 4 usages

    public Composition(Function outer, Function inner) { no usages new *
        this.outer = outer;
        this.inner = inner;
    }

    public double getLeftDomainBorder() { new *
        return inner.getLeftDomainBorder();
    }

    public double getRightDomainBorder() { new *
        return inner.getRightDomainBorder();
    }

    public double getFunctionValue(double x) { new *
        return outer.getFunctionValue(inner.getFunctionValue(x));
    }
}
```

## Задание 5

Создаём класс Functions в пакете functions.

Класс Functions — утилитарный класс с статическими методами, которые создают новые объекты функций на основе существующих.

Приватный конструктор предотвращает создание экземпляров.

Методы делегируют работу соответствующим классам из functions.meta.

public static Function shift(Function f, double shiftX, double shiftY) – возвращает объект функции, полученной из исходной сдвигом вдоль осей;

public static Function scale(Function f, double scaleX, double scaleY) – возвращает объект функции, полученной из исходной масштабированием вдоль осей;

public static Function power(Function f, double power) – возвращает объект функции, являющейся заданной степенью исходной;

public static Function sum(Function f1, Function f2) – возвращает объект функции, являющейся суммой двух исходных;

public static Function mult(Function f1, Function f2) – возвращает объект функции, являющейся произведением двух исходных;

public static Function composition(Function f1, Function f2) – возвращает объект функции, являющейся композицией двух исходных.

```
1 package functions;
2
3 import functions.meta.*;
4
5 public class Functions { no usages new *
6
7     private Functions() {} no usages new *
8
9     @
10    public static Function shift(Function f, double shiftX, double shiftY) { no usages new *
11        return new Shift(f, shiftX, shiftY);
12    }
13
14    @
15    public static Function scale(Function f, double scaleX, double scaleY) { no usages new *
16        return new Scale(f, scaleX, scaleY);
17    }
18
19    @
20    public static Function power(Function f, double power) { no usages new *
21        return new Power(f, power);
22    }
23
24
25    @
26    public static Function sum(Function f1, Function f2) { no usages new *
27        return new Sum(f1, f2);
28    }
29
30    @
31    public static Function mult(Function f1, Function f2) { no usages new *
32        return new Mult(f1, f2);
33    }
34
35    @
36    public static Function composition(Function f1, Function f2) { no usages new *
37        return new Composition(f1, f2);
38    }
39
40 }
```

## Задание 6

Создаём класс TabulatedFunctions в пакете functions.

TabulatedFunctions — утилитарный класс с статическими методами для работы с табулированными функциями. Приватный конструктор предотвращает создание экземпляров. Метод tabulate создаёт линейно интерполированную табулированную функцию по заданной функции и отрезку. Метод проверяет корректность границ и минимальное количество точек. Если указанные границы для табулирования выходят за область определения функции, метод выбрасывает исключение IllegalArgumentException.

```
package functions;

public class TabulatedFunctions { no usages new *
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) { no usages ne
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табулирования выходят за область определения функции");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не меньше двух");
        }

        double step = (rightX - leftX) / (pointsCount - 1);
        double[] yValues = new double[pointsCount];

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

        return new ArrayTabulatedFunction(leftX, rightX, yValues);
    }
}
```

## Задание 7

Создаём метод outputTabulatedFunction(): Сохраняет табулированную функцию в байтовый поток. Сначала записывает количество точек, затем поочередно координаты x и y каждой точки. Использует DataOutputStream. Поток не закрывается внутри метода.

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

Метод `inputTabulatedFunction()`: Читает табулированную функцию из байтового потока. Считывает количество точек, затем координаты x и y. Создаёт объект `TabulatedFunction` на основе считанных данных. Использует `DataInputStream`. Поток не закрывается. Исключение `IOException` прорывается наружу.

```
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    DataInputStream dis = new DataInputStream(in);
    int pointsCount = dis.readInt();
    double[] yValues = new double[pointsCount];
    double leftX = dis.readDouble();
    double rightX = leftX; // будем пересчитывать rightX
    yValues[0] = dis.readDouble();
    for (int i = 1; i < pointsCount; i++) {
        double x = dis.readDouble();
        yValues[i] = dis.readDouble();
        rightX = x;
    }
    return new ArrayTabulatedFunction(leftX, rightX, yValues);
}
```

Метод `writeTabulatedFunction()`: Записывает табулированную функцию в символьный поток. Выводит данные через пробел: сначала количество точек, затем x и y каждой точки. Использует `BufferedWriter`. Удобно для визуального просмотра человеком. Поток не закрывается.

```
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    BufferedWriter bw = new BufferedWriter(out);
    int pointsCount = function.getPointsCount();
    bw.write(String.valueOf(pointsCount));
    bw.write(" ");
    for (int i = 0; i < pointsCount; i++) {
        bw.write(function.getPointX(i) + " " + function.getPointY(i) + " ");
    }
    bw.newLine();
    bw.flush();
}
```

Метод `readTabulatedFunction()`: Считывает табулированную функцию из символьного потока. Использует `StreamTokenizer` для поочерёдного чтения чисел. Создаёт объект `TabulatedFunction` по считанным данным.: Поток не закрывается. Предполагается, что данные корректные.

```
public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException { no usages new *
    StreamTokenizer st = new StreamTokenizer(in);
    st.nextToken();
    int pointsCount = (int) st.nval;
    double[] xValues = new double[pointsCount];
    double[] yValues = new double[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        st.nextToken();
        xValues[i] = st.nval;
        st.nextToken();
        yValues[i] = st.nval;
    }
    return new ArrayTabulatedFunction(xValues[0], xValues[pointsCount - 1], yValues);
}
```

IOException:

Методы (outputTabulatedFunction, inputTabulatedFunction, writeTabulatedFunction, readTabulatedFunction) не перехватывают IOException, а пробрасывают наружу. Это позволяет вызывающему коду решать, как обработать ошибку, и делает методы более гибкими.

Закрытие потоков:

Потоки не закрываются внутри методов, так как они создаются и управляются вызывающим кодом. Закрытие выполняется там, где поток был открыт, чтобы не нарушать управление ресурсами.

Вывод: Методы только обрабатывают данные, исключения пробрасываются наружу, а потоки остаются под контролем вызывающего кода.

## Задание 8

Проверка написанных классов:

Аналитические функции:

Созданы объекты Sin и Cos. Выведены значения функций на отрезке  $[0, \pi]$  с шагом 0,1.

Табулированные функции:

С помощью TabulatedFunctions.tabulate() созданы табулированные аналоги Sin и Cos с 10 точками. Значения на том же отрезке сравнивались с аналитическими.

Сумма квадратов:

С использованием класса Functions создан объект функции, являющейся суммой квадратов табулированных функций. Вывод значений показал приближение к 1, точность зависит от количества точек.

Экспонента:

Создан табулированный аналог Exp на [0,10] с 11 точками. Сохранён в текстовый файл и считан обратно. Значения считанной функции совпали с исходной.

Натуральный логарифм:

Создан табулированный аналог Log на [0,10] с 11 точками. Сохранён в бинарный файл и считан обратно. Значения совпали с исходной функцией.

```

1 import functions.*;
2 import functions.basic.*;
3 import java.io.*;
4
5 public class Main { & SultanovaArina*
6     public static void main(String[] args) { & SultanovaArina*
7         System.out.println("Проверка работы функций \n");
8
9         // Шаг 1: Создаем объекты Sin и Cos
10        Function sin = new Sin();
11        Function cos = new Cos();
12
13        System.out.println("\nЗначения аналитических функций Sin и Cos на [0, π] с шагом 0.1:");
14        for (double x = 0; x <= Math.PI; x += 0.1) {
15            System.out.printf("x=%f sin=%f cos=%f\n", x, sin.getFunctionValue(x), cos.getFunctionValue(x));
16        }
17
18        // Шаг 2: Табулируем функции
19        TabulatedFunction sinTab = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
20        TabulatedFunction cosTab = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);
21
22        System.out.println("\nЗначения табулированных функций Sin и Cos на [0, π] с шагом 0.1:");
23        for (double x = 0; x <= Math.PI; x += 0.1) {
24            double ySin = sinTab.getFunctionValue(x);
25            double yCos = cosTab.getFunctionValue(x);
26            System.out.printf("x=%f sinTab=%f cosTab=%f\n", x, ySin, yCos);
27        }
28
29        // Шаг 3: Сумма квадратов табулированных функций
30        Function sumSquares = Functions.sum(
31            Functions.power(sinTab, power: 2),
32            Functions.power(cosTab, power: 2)
33        );
34
35        System.out.println("\nЗначения суммы квадратов табулированных функций на [0, π] с шагом 0.1:");
36        for (double x = 0; x <= Math.PI; x += 0.1) {
37            System.out.printf("x=%f sumSquares=%f\n", x, sumSquares.getFunctionValue(x));
38        }
39
40        // Шаг 4: Табулируем экспоненту, записываем и читаем из файла
41        try {
42            Function exp = new Exp();
43            TabulatedFunction expTab = TabulatedFunctions.tabulate(exp, leftX: 0, rightX: 10, pointsCount: 11);
44
45            // Запись в файл
46            FileWriter fw = new FileWriter( fileName: "exp_func.txt");
47            TabulatedFunctions.writeTabulatedFunction(expTab, fw);
48            fw.close();
49
50            // Чтение из файла
51            FileReader fr = new FileReader( fileName: "exp_func.txt");
52            TabulatedFunction expTabRead = TabulatedFunctions.readTabulatedFunction(fr);
53            fr.close();
54
55            System.out.println("\nСравнение исходной и считанной табулированной экспоненты:");
56            for (int i = 0; i <= 10; i++) {
57                double x = i;

```

```
56         for (int i = 0; i <= 10; i++) {
57             double x = i;
58             System.out.printf("x=%2f exp=%4f expRead=%4f%n",
59                               x, expTab.getFunctionValue(x), expTabRead.getFunctionValue(x));
60         }
61     } catch (IOException e) {
62         System.out.println("Ошибка работы с файлом экспоненты: " + e.getMessage());
63     }
64
65     // Шаг 5: Табулируем натуральный логарифм, записываем и читаем из файла
66     try {
67         Function log = new Log(Math.E);
68         TabulatedFunction logTab = TabulatedFunctions.tabulate(log, leftX: 0.1, rightX: 10, pointsCount: 11); // левый > 0
69
70         // Запись в файл (байтовый поток)
71         FileOutputStream fos = new FileOutputStream( name: "log_func.bin");
72         TabulatedFunctions.outputTabulatedFunction(logTab, fos);
73         fos.close();
74
75         // Чтение из файла
76         FileInputStream fis = new FileInputStream( name: "log_func.bin");
77         TabulatedFunction logTabRead = TabulatedFunctions.inputTabulatedFunction(fis);
78         fis.close();
79
80         System.out.println("\nСравнение исходного и считанного табулированного логарифма:");
81         for (int i = 0; i < logTab.getPointsCount(); i++) {
82             double x = logTab.getPointX(i);
83             System.out.printf("x=%2f log=%4f logRead=%4f%n",
84                               x, logTab.getFunctionValue(x), logTabRead.getFunctionValue(x));
85         }
86     } catch (IOException e) {
87         System.out.println("Ошибка работы с файлом логарифма: " + e.getMessage());
88     }
89
90 }
91 }
92 }
93 }
```

```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:D:\java\Inte  
Проверка работы функций
```

Значения аналитических функций Sin и Cos на [0, π] с шагом 0.1:

```
x=0,00 sin=0,0000 cos=1,0000
x=0,10 sin=0,0998 cos=0,9950
x=0,20 sin=0,1987 cos=0,9801
x=0,30 sin=0,2955 cos=0,9553
x=0,40 sin=0,3894 cos=0,9211
x=0,50 sin=0,4794 cos=0,8776
x=0,60 sin=0,5646 cos=0,8253
x=0,70 sin=0,6442 cos=0,7648
x=0,80 sin=0,7174 cos=0,6967
x=0,90 sin=0,7833 cos=0,6216
x=1,00 sin=0,8415 cos=0,5403
x=1,10 sin=0,8912 cos=0,4536
x=1,20 sin=0,9320 cos=0,3624
x=1,30 sin=0,9636 cos=0,2675
x=1,40 sin=0,9854 cos=0,1700
x=1,50 sin=0,9975 cos=0,0707
x=1,60 sin=0,9996 cos=-0,0292
x=1,70 sin=0,9917 cos=-0,1288
x=1,80 sin=0,9738 cos=-0,2272
x=1,90 sin=0,9463 cos=-0,3233
x=2,00 sin=0,9093 cos=-0,4161
x=2,10 sin=0,8632 cos=-0,5048
x=2,20 sin=0,8085 cos=-0,5885
x=2,30 sin=0,7457 cos=-0,6663
x=2,40 sin=0,6755 cos=-0,7374
x=2,50 sin=0,5985 cos=-0,8011
x=2,60 sin=0,5155 cos=-0,8569
x=2,70 sin=0,4274 cos=-0,9041
x=2,80 sin=0,3350 cos=-0,9422
x=2,90 sin=0,2392 cos=-0,9710
x=3,00 sin=0,1411 cos=-0,9900
x=3,10 sin=0,0416 cos=-0,9991
```

Значения табулированных функций Sin и Cos на  $[0, \pi]$  с шагом 0.1:

```
x=0,00 sinTab=0,0000 cosTab=1,0000
x=0,10 sinTab=0,0980 cosTab=0,9827
x=0,20 sinTab=0,1960 cosTab=0,9654
x=0,30 sinTab=0,2939 cosTab=0,9482
x=0,40 sinTab=0,3859 cosTab=0,9144
x=0,50 sinTab=0,4721 cosTab=0,8646
x=0,60 sinTab=0,5582 cosTab=0,8149
x=0,70 sinTab=0,6440 cosTab=0,7646
x=0,80 sinTab=0,7079 cosTab=0,6884
x=0,90 sinTab=0,7719 cosTab=0,6122
x=1,00 sinTab=0,8358 cosTab=0,5360
x=1,10 sinTab=0,8840 cosTab=0,4506
x=1,20 sinTab=0,9180 cosTab=0,3571
x=1,30 sinTab=0,9521 cosTab=0,2636
x=1,40 sinTab=0,9848 cosTab=0,1699
x=1,50 sinTab=0,9848 cosTab=0,0704
x=1,60 sinTab=0,9848 cosTab=-0,0291
x=1,70 sinTab=0,9848 cosTab=-0,1285
x=1,80 sinTab=0,9662 cosTab=-0,2248
x=1,90 sinTab=0,9322 cosTab=-0,3183
x=2,00 sinTab=0,8981 cosTab=-0,4117
x=2,10 sinTab=0,8624 cosTab=-0,5043
x=2,20 sinTab=0,7985 cosTab=-0,5805
x=2,30 sinTab=0,7345 cosTab=-0,6567
x=2,40 sinTab=0,6706 cosTab=-0,7329
x=2,50 sinTab=0,5941 cosTab=-0,7942
x=2,60 sinTab=0,5079 cosTab=-0,8439
x=2,70 sinTab=0,4217 cosTab=-0,8937
x=2,80 sinTab=0,3347 cosTab=-0,9410
x=2,90 sinTab=0,2367 cosTab=-0,9583
x=3,00 sinTab=0,1387 cosTab=-0,9755
x=3,10 sinTab=0,0408 cosTab=-0,9928
```

```
Значения суммы квадратов табулированных функций на [0, π] с шагом 0.1:  
x=0,00 sumSquares=1,0000  
x=0,10 sumSquares=0,9753  
x=0,20 sumSquares=0,9705  
x=0,30 sumSquares=0,9854  
x=0,40 sumSquares=0,9850  
x=0,50 sumSquares=0,9704  
x=0,60 sumSquares=0,9756  
x=0,70 sumSquares=0,9994  
x=0,80 sumSquares=0,9751  
x=0,90 sumSquares=0,9706  
x=1,00 sumSquares=0,9859  
x=1,10 sumSquares=0,9845  
x=1,20 sumSquares=0,9703  
x=1,30 sumSquares=0,9759  
x=1,40 sumSquares=0,9987  
x=1,50 sumSquares=0,9748  
x=1,60 sumSquares=0,9707  
x=1,70 sumSquares=0,9864  
x=1,80 sumSquares=0,9841  
x=1,90 sumSquares=0,9702  
x=2,00 sumSquares=0,9762  
x=2,10 sumSquares=0,9981  
x=2,20 sumSquares=0,9745  
x=2,30 sumSquares=0,9708  
x=2,40 sumSquares=0,9869  
x=2,50 sumSquares=0,9836  
x=2,60 sumSquares=0,9702  
x=2,70 sumSquares=0,9765  
x=2,80 sumSquares=0,9975  
x=2,90 sumSquares=0,9743  
x=3,00 sumSquares=0,9709  
x=3,10 sumSquares=0,9873
```

Сравнение исходной и считанной табулированной экспоненты:

```
x=0,00 exp=1,0000 expRead=1,0000  
x=1,00 exp=2,7183 expRead=2,7183  
x=2,00 exp=7,3891 expRead=7,3891  
x=3,00 exp=20,0855 expRead=20,0855  
x=4,00 exp=54,5982 expRead=54,5982  
x=5,00 exp=148,4132 expRead=148,4132  
x=6,00 exp=403,4288 expRead=403,4288  
x=7,00 exp=1096,6332 expRead=1096,6332  
x=8,00 exp=2980,9580 expRead=2980,9580  
x=9,00 exp=8103,0839 expRead=8103,0839  
x=10,00 exp=22026,4658 expRead=22026,4658
```

```
Сравнение исходного и считанного табулированного логарифма:  
x=0,10 log=-2,3026 logRead=-2,3026  
x=1,09 log=0,0862 logRead=0,0862  
x=2,08 log=0,7324 logRead=0,7324  
x=3,07 log=1,1217 logRead=1,1217  
x=4,06 log=1,4012 logRead=1,4012  
x=5,05 log=1,6194 logRead=1,6194  
x=6,04 log=1,7984 logRead=1,7984  
x=7,03 log=1,9502 logRead=1,9502  
x=8,02 log=2,0819 logRead=2,0819  
x=9,01 log=2,1983 logRead=2,1983  
x=10,00 log=2,3026 logRead=2,3026
```

Анализ форматов хранения табулированных функций:

Символьный (текстовый) формат: Плюсы:

Файл читаемый человеком, легко проверять и редактировать.

Можно открывать в любом текстовом редакторе.

Минусы:

Занимает больше места, чем бинарный.

Чтение и запись медленнее, чем в бинарном формате.

Возможны ошибки при парсинге при некорректных данных.

Байтовой (binary) формат:

Плюсы:

Компактное хранение данных, меньше размер файла.

Быстрое чтение и запись, удобно для больших таблиц.

Не требует парсинга текста, меньше вероятность ошибок формата.

Минусы:

Файл нечитаемый человеком без специальных средств.

Любые изменения требуют десериализации и повторной записи.

**Вывод:** Для чтения человеком и небольших функций удобен текстовый формат.

Для эффективного хранения больших таблиц предпочтителен бинарный формат.

## Задание 9

Сделаем так, чтобы объекты всех классов, реализующих интерфейс TabulatedFunction, были сериализуемыми.

с использованием интерфейса java.io.Serializable

с использованием интерфейса java.io.Externalizable

```
1 import functions.*;
2 import functions.basic.*;
3 import functions.meta.*;
4
5 import java.io.*;
6
7 public class Main { & SultanovaArina *
8     public static void main(String[] args) { & SultanovaArina *
9         System.out.println("==> Проверка аналитических функций ==>");
10        checkAnalyticalFunctions();
11
12        System.out.println("\n==> Табулированные функции и сравнение с исходными ==>");
13        checkTabulatedFunctions();
14
15        System.out.println("\n==> Проверка сериализации (Serializable и Externalizable) ==>");
16        checkSerialization();
17    }
18
19 // --- Задание 8: аналитические функции Sin и Cos ---
20 public static void checkAnalyticalFunctions() { 1 usage & SultanovaArina *
21     Function sin = new Sin();
22     Function cos = new Cos();
23
24     System.out.println("\nСинус:");
25     for (double x = 0; x <= Math.PI; x += 0.1) {
26         System.out.printf("sin(%2f) = %.4f\n", x, sin.getFunctionValue(x));
27     }
28
29     System.out.println("\nКосинус:");
30     for (double x = 0; x <= Math.PI; x += 0.1) {
31         System.out.printf("cos(%2f) = %.4f\n", x, cos.getFunctionValue(x));
32     }
33 }
34
35 // --- Задание 8: табулирование и сравнение ---
36 public static void checkTabulatedFunctions() { 1 usage & SultanovaArina *
37     Function sin = new Sin();
38     Function cos = new Cos();
39
40     TabulatedFunction tabSin = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
41     TabulatedFunction tabCos = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);
42
43     // Сумма квадратов табулированных функций
44     Function sumSquares = Functions.sum(
45         Functions.power(tabSin, power: 2),
46         Functions.power(tabCos, power: 2)
47     );
48
49     System.out.println("\nСравнение исходных и табулированных значений на [0, PI]:");
50     for (double x = 0; x <= Math.PI; x += 0.1) {
51         double f = sumSquares.getFunctionValue(x);
52         System.out.printf("x=%2f, sin^2+cos^2=%4f\n", x, f);
53     }
54
55     // Табулирование экспоненты и логарифма
56     TabulatedFunction tabExp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);
```

```

56     TabulatedFunction tabExp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);
57     TabulatedFunction tabLog = TabulatedFunctions.tabulate(new Log(Math.E), leftX: 0, rightX: 10, pointsCount: 11);
58
59     // Сохранение и чтение текстового файла (экспонента)
60     try (FileWriter writer = new FileWriter(fileName: "exp_tab.txt")) {
61         TabulatedFunctions.writeTabulatedFunction(tabExp, writer);
62     } catch (IOException e) {
63         e.printStackTrace();
64     }
65
66     try (FileReader reader = new FileReader(fileName: "exp_tab.txt")) {
67         TabulatedFunction readExp = TabulatedFunctions.readTabulatedFunction(reader);
68         System.out.println("\nСравнение исходной и считанной табулированной экспоненты:");
69         for (double x = 0; x <= 10; x += 1) {
70             System.out.printf("x=%1.1f, original=%1.4f, read=%1.4f\n",
71                             x, tabExp.getFunctionValue(x), readExp.getFunctionValue(x));
72         }
73     } catch (IOException e) {
74         e.printStackTrace();
75     }
76
77     // Сохранение и чтение бинарного файла (логарифм)
78     try (FileOutputStream out = new FileOutputStream(name: "log_tab.bin")) {
79         TabulatedFunctions.outputTabulatedFunction(tabLog, out);
80     } catch (IOException e) {
81         e.printStackTrace();
82     }
83
84     try (FileInputStream in = new FileInputStream(name: "log_tab.bin")) {
85         TabulatedFunction readLog = TabulatedFunctions.inputTabulatedFunction(in);
86         System.out.println("\nСравнение исходного и считанного табулированного логарифма:");
87         for (double x = 0; x <= 10; x += 1) {
88             System.out.printf("x=%1.1f, original=%1.4f, read=%1.4f\n",
89                             x, tabLog.getFunctionValue(x), readLog.getFunctionValue(x));
90         }
91     } catch (IOException e) {
92         e.printStackTrace();
93     }
94 }

```

```

96 // Задание 9: сериализация Serializable и Externalizable
97 public static void checkSerialization() { 1 usage   SultanovaArina*
98     try {
99         // Serializable для ArrayTabulatedFunction
100        TabulatedFunction arrayFunc = TabulatedFunctions.tabulate( new Composition(new Log(Math.E), new Exp()), leftX: 0, rightX: 10, pointsCount: 11 );
101
102        ObjectOutputStream outSer = new ObjectOutputStream(new FileOutputStream(name: "array_ser.dat"));
103        outSer.writeObject(arrayFunc); // Serializable
104        outSer.close();
105
106        ObjectInputStream inSer = new ObjectInputStream(new FileInputStream(name: "array_ser.dat"));
107        TabulatedFunction readArray = (TabulatedFunction) inSer.readObject();
108        inSer.close();
109
110        System.out.println("\nSerializable (Array) проверка:");
111        for (double x = 0; x <= 10; x += 1) {
112            System.out.printf("x=%1.1f, original=%1.4f, read=%1.4f\n",
113                            x, arrayFunc.getFunctionValue(x), readArray.getFunctionValue(x));
114        }
115        // Externalizable для LinkedListTabulatedFunction
116        Function func = new Composition(new Log(Math.E), new Exp());
117        double left = 0;
118        double right = 10;
119        int pointsCount = 11;

```

```

125     // Вычисляем значения функции в точках
126     double[] values = new double[pointsCount];
127     double step = (right - left) / (pointsCount - 1);
128     for (int i = 0; i < pointsCount; i++) {
129         double x = left + i * step;
130         values[i] = func.getFunctionValue(x);
131     }
132
133     // Создаём LinkedListTabulatedFunction с готовыми значениями
134     LinkedListTabulatedFunction listFunc = new LinkedListTabulatedFunction(left, right, values);
135
136     // СерIALIZАЦИЯ Externalizable
137     ObjectOutputStream outExt = new ObjectOutputStream(new FileOutputStream("list_ext.dat"));
138     outExt.writeObject(listFunc); // вызов writeExternal()
139     outExt.close();
140
141     // ДЕСЕРИАЛИЗАЦИЯ
142     ObjectInputStream inExt = new ObjectInputStream(new FileInputStream("list_ext.dat"));
143     LinkedListTabulatedFunction readList = (LinkedListTabulatedFunction) inExt.readObject(); // вызов readExternal()
144     inExt.close();
145
146     System.out.println("\nExternalizable (LinkedList) проверка:");
147     for (double x = 0; x <= 10; x += 1) {
148         System.out.printf("x=%1.1f, original=%4.4f, read=%4.4f\n",
149                           x, listFunc.getFunctionValue(x), readList.getFunctionValue(x));
150     }
151
152 } catch (IOException | ClassNotFoundException e) {
153     e.printStackTrace();
154 }
```

Также для Externalizable в LinkedListTabulatedFunction добавила:

```

7  public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable { 4 usages & SultanovaArina*
8      private FunctionNode head; 36 usages
9      private int size; 25 usages
10     private static final double EPS = Mathulp(1.0); 11 usages
11
12     private static class FunctionNode { 30 usages & SultanovaArina
13         public FunctionPoint point; 25 usages
14         public FunctionNode prev; 21 usages
15         public FunctionNode next; 16 usages
16
17         public FunctionNode(FunctionPoint point) { 8 usages & SultanovaArina
18             this.point = point;
19         }
20     }
21     public LinkedListTabulatedFunction() { no usages & SultanovaArina
22         head = new FunctionNode( point: null );
23         head.next = head;
24         head.prev = head;
25         size = 0;
26     }
27     public void writeExternal(ObjectOutput out) throws IOException { new *
28         out.writeInt(size);
29         for (int i = 0; i < size; i++) {
30             out.writeDouble(getPointX(i));
31             out.writeDouble(getPointY(i));
32         }
33     }
34
35     public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException { new *
36         int n = in.readInt();
37         head = new FunctionNode( point: null );
38         head.next = head;
39         head.prev = head;
40         size = 0;
41         for (int i = 0; i < n; i++) {
42             FunctionNode node = new FunctionNode(new FunctionPoint());
43             node.prev = head.prev;
44             node.next = head;
45             head.prev.next = node;
46             head.prev = node;
47             size++;
48
49             node.point.setX(in.readDouble());
50             node.point.setY(in.readDouble());
51         }
52     }
53 }
```

```
"C:\Program Files\Java\jdk-24\bin\jav
== Проверка аналитических функций ==

Синус:
sin(0,00) = 0,0000
sin(0,10) = 0,0998
sin(0,20) = 0,1987
sin(0,30) = 0,2955
sin(0,40) = 0,3894
sin(0,50) = 0,4794
sin(0,60) = 0,5646
sin(0,70) = 0,6442
sin(0,80) = 0,7174
sin(0,90) = 0,7833
sin(1,00) = 0,8415
sin(1,10) = 0,8912
sin(1,20) = 0,9320
sin(1,30) = 0,9636
sin(1,40) = 0,9854
sin(1,50) = 0,9975
sin(1,60) = 0,9996
sin(1,70) = 0,9917
sin(1,80) = 0,9738
sin(1,90) = 0,9463
sin(2,00) = 0,9093
sin(2,10) = 0,8632
sin(2,20) = 0,8085
sin(2,30) = 0,7457
sin(2,40) = 0,6755
sin(2,50) = 0,5985
sin(2,60) = 0,5155
sin(2,70) = 0,4274
sin(2,80) = 0,3350
sin(2,90) = 0,2392
sin(3,00) = 0,1411
sin(3,10) = 0,0416
```

Косинус:

```
cos(0,00) = 1,0000
cos(0,10) = 0,9950
cos(0,20) = 0,9801
cos(0,30) = 0,9553
cos(0,40) = 0,9211
cos(0,50) = 0,8776
cos(0,60) = 0,8253
cos(0,70) = 0,7648
cos(0,80) = 0,6967
cos(0,90) = 0,6216
cos(1,00) = 0,5403
cos(1,10) = 0,4536
cos(1,20) = 0,3624
cos(1,30) = 0,2675
cos(1,40) = 0,1700
cos(1,50) = 0,0707
cos(1,60) = -0,0292
cos(1,70) = -0,1288
cos(1,80) = -0,2272
cos(1,90) = -0,3233
cos(2,00) = -0,4161
cos(2,10) = -0,5048
cos(2,20) = -0,5885
cos(2,30) = -0,6663
cos(2,40) = -0,7374
cos(2,50) = -0,8011
cos(2,60) = -0,8569
cos(2,70) = -0,9041
cos(2,80) = -0,9422
cos(2,90) = -0,9710
cos(3,00) = -0,9900
cos(3,10) = -0,9991
```

```
== Табулированные функции и сравнение с исходными ==
```

Сравнение исходных и табулированных значений на [0, PI]:

```
x=0,00, sin^2+cos^2=1,0000
x=0,10, sin^2+cos^2=0,9753
x=0,20, sin^2+cos^2=0,9705
x=0,30, sin^2+cos^2=0,9854
x=0,40, sin^2+cos^2=0,9850
x=0,50, sin^2+cos^2=0,9704
x=0,60, sin^2+cos^2=0,9756
x=0,70, sin^2+cos^2=0,9994
x=0,80, sin^2+cos^2=0,9751
x=0,90, sin^2+cos^2=0,9706
x=1,00, sin^2+cos^2=0,9859
x=1,10, sin^2+cos^2=0,9845
x=1,20, sin^2+cos^2=0,9703
x=1,30, sin^2+cos^2=0,9759
x=1,40, sin^2+cos^2=0,9987
x=1,50, sin^2+cos^2=0,9748
x=1,60, sin^2+cos^2=0,9707
x=1,70, sin^2+cos^2=0,9864
x=1,80, sin^2+cos^2=0,9841
x=1,90, sin^2+cos^2=0,9702
x=2,00, sin^2+cos^2=0,9762
x=2,10, sin^2+cos^2=0,9981
x=2,20, sin^2+cos^2=0,9745
x=2,30, sin^2+cos^2=0,9708
x=2,40, sin^2+cos^2=0,9869
x=2,50, sin^2+cos^2=0,9836
x=2,60, sin^2+cos^2=0,9702
x=2,70, sin^2+cos^2=0,9765
x=2,80, sin^2+cos^2=0,9975
x=2,90, sin^2+cos^2=0,9743
x=3,00, sin^2+cos^2=0,9709
x=3,10, sin^2+cos^2=0,9873
```

Сравнение исходной и считанной табулированной экспоненты:

```
x=0,0, original=1,0000, read=1,0000
x=1,0, original=2,7183, read=2,7183
x=2,0, original=7,3891, read=7,3891
x=3,0, original=20,0855, read=20,0855
x=4,0, original=54,5982, read=54,5982
x=5,0, original=148,4132, read=148,4132
x=6,0, original=403,4288, read=403,4288
x=7,0, original=1096,6332, read=1096,6332
x=8,0, original=2980,9580, read=2980,9580
x=9,0, original=8103,0839, read=8103,0839
x=10,0, original=22026,4658, read=22026,4658
```

Сравнение исходного и считанного табулированного логарифма:

```
x=0,0, original=NaN, read=NaN
x=1,0, original=0,0000, read=0,0000
x=2,0, original=0,6931, read=0,6931
x=3,0, original=1,0986, read=1,0986
x=4,0, original=1,3863, read=1,3863
x=5,0, original=1,6094, read=1,6094
x=6,0, original=1,7918, read=1,7918
x=7,0, original=1,9459, read=1,9459
x=8,0, original=2,0794, read=2,0794
x=9,0, original=2,1972, read=2,1972
x=10,0, original=2,3026, read=2,3026
```

```
== Проверка сериализации (Serializable и Externalizable) ==
```

```
Serializable (Array) проверка:
```

```
x=0,0, original=0,0000, read=0,0000  
x=1,0, original=1,0000, read=1,0000  
x=2,0, original=2,0000, read=2,0000  
x=3,0, original=3,0000, read=3,0000  
x=4,0, original=4,0000, read=4,0000  
x=5,0, original=5,0000, read=5,0000  
x=6,0, original=6,0000, read=6,0000  
x=7,0, original=7,0000, read=7,0000  
x=8,0, original=8,0000, read=8,0000  
x=9,0, original=9,0000, read=9,0000  
x=10,0, original=10,0000, read=10,0000
```

```
Externalizable (LinkedList) проверка:
```

```
x=0,0, original=0,0000, read=0,0000  
x=1,0, original=1,0000, read=1,0000  
x=2,0, original=2,0000, read=2,0000  
x=3,0, original=3,0000, read=3,0000  
x=4,0, original=4,0000, read=4,0000  
x=5,0, original=5,0000, read=5,0000  
x=6,0, original=6,0000, read=6,0000  
x=7,0, original=7,0000, read=7,0000  
x=8,0, original=8,0000, read=8,0000  
x=9,0, original=9,0000, read=9,0000  
x=10,0, original=10,0000, read=10,0000
```

Serializable: проще в использовании, не требует ручного управления сериализацией, автоматически сохраняет все поля. Но файлы могут быть большими и менее контролируемыми.

Externalizable: требует ручного кода для записи и чтения полей, но позволяет полностью контролировать формат данных и уменьшить размер файла. Менее удобен, но более гибок и эффективен.