

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 Богданчиков М.А.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка \_\_\_\_\_

**Цель лабораторной работы:** Расширить возможности пакета для работы с функциями одной переменной добавив интерфейсы и классы для аналитически заданных функций, а также методы ввода и вывода табулированных функций.

## Ход выполнения лабораторной работы

### Задание 1: В

классах ArrayTabulatedFunction и LinkedListTabulatedFunction добавлены конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint. Если точек задано меньше двух, или если точки в массиве не упорядочены по значению абсциссы, конструкторы выбрасывают исключение IllegalArgumentException.

```
public ArrayTabulatedFunction(FunctionPoint[] points) {  
    if (points.length < 2) {  
        throw new IllegalArgumentException("At least 2 points required");  
    }  
    for (int i = 1; i < points.length; i++) {  
        if (points[i].getX() <= points[i-1].getX() + EPSILON) {  
            throw new IllegalArgumentException("Points must be strictly ordered by increasing X");  
        }  
    }  
    this.pointsCount = points.length;  
    this.points = new FunctionPoint[pointsCount + 10];  
  
    for (int i = 0; i < pointsCount; i++) {  
        this.points[i] = new FunctionPoint(points[i]);  
    }  
}
```

### Код 1.1 (конструктор для ArrayTabulatedFunction)

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {  
    this();  
    if (points.length < 2) {  
        throw new IllegalArgumentException("At least 2 points required");  
    }  
    for (int i = 1; i < points.length; i++) {  
        if (points[i].getX() <= points[i-1].getX() + EPSILON) {  
            throw new IllegalArgumentException("Points must be strictly ordered by increasing X");  
        }  
    }  
    for (FunctionPoint point : points) {  
        try {  
            addPoint(new FunctionPoint(point));  
        } catch (InappropriateFunctionPointException e) {  
            throw new RuntimeException("Unexpected error in constructor", e);  
        }  
    }  
}
```

### Код 1.2 (конструктор для LinkedListTabulatedFunction)

**Задание 2:** В пакете functions создан интерфейс Function, описывающий функции одной переменной и содержащий следующие методы:

- public double getLeftDomainBorder() – возвращает значение левой границы области определения функции;
- public double getRightDomainBorder() – возвращает значение правой границы области определения функции;
- public double getFunctionValue(double x) – возвращает значение функции в заданной точке.

Исключены соответствующие методы из интерфейса TabulatedFunction и теперь он расширяет интерфейс Function.

```
package functions;

public interface Function {
    double getLeftDomainBorder();
    double getRightDomainBorder();
    double getFunctionValue(double x);
}
```

**Код 2.1**

```
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;
    String toString();
}
```

**Код 2.2**

### Задание 3:

Создан пакет functions.basic, в нём описаны классы ряда функций, заданных аналитически.

Создан в пакете публичный класс Exp, объекты которого должны вычислять значение экспоненты. Класс реализует интерфейс Function.

Создан класс Log, объекты которого должны вычислять значение логарифма по заданному основанию. Класс реализует интерфейс Function.

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

Созданы наследующие от него публичные классы Sin, Cos и Tan, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса.

```
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);
    }
}
```

Код 3.1

```
package functions.basic;
import functions.Function;
public class Log implements Function {
    private double base;
    public Log(double base) {
        if (base <= 0 || Math.abs(base - 1) < 1e-10) {
            throw new IllegalArgumentException("Base must be positive and not equal to 1");
        }
        this.base = base;
    }
    @Override
    public double getLeftDomainBorder() {
        return 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);
    }
    public double getBase() {
        return base;
    }
}
```

## Код 3.2

```
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;
    }
}
```

## Код 3.3

```
package functions.basic;

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

**Код 3.4**

```
package functions.basic;

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

**Код 3.5**

```
package functions.basic;

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

**Код 3.6**

**Задание 4:** Создан пакет functions.meta, в нём описаны классы функций, позволяющие комбинировать функции.

Создан класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций. Класс реализует интерфейс Function.

Создан класс Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций.

Создан класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции.

Создан класс Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат.

Создан класс Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат.

Создан класс Composition, объекты которого описывают композицию двух исходных функций.

```
package functions.meta;
import functions.Function;
public class Sum implements Function {
    private Function f1;
    private Function f2;

    public Sum(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }

    @Override
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double y1 = f1.getFunctionValue(x);
        double y2 = f2.getFunctionValue(x);

        if (Double.isNaN(y1) || Double.isNaN(y2)) {
            return Double.NaN;
        }
        return y1 + y2;
    }
}
```

## Код 4.1

```
package functions.meta;
import functions.Function;
public class Mult implements Function {
    private Function f1;
    private Function f2;
    public Mult(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }
    @Override
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    @Override
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }
    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double y1 = f1.getFunctionValue(x);
        double y2 = f2.getFunctionValue(x);

        if (Double.isNaN(y1) || Double.isNaN(y2)) {
            return Double.NaN;
        }
        return y1 * y2;
    }
}
```

## Код 4.2

```
package functions.meta;

import functions.Function;

public class Composition implements Function {
    private Function f1;
    private Function f2;

    public Composition(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }

    @Override
    public double getLeftDomainBorder() {
        return f1.getLeftDomainBorder();
    }

    @Override
    public double getRightDomainBorder() {
        return f1.getRightDomainBorder();
    }

    @Override
    public double getFunctionValue(double x) {
        double y1 = f1.getFunctionValue(x);
        if (Double.isNaN(y1)) {
            return Double.NaN;
        }
        return f2.getFunctionValue(y1);
    }
}
```

Код 4.3

```
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) {
        double shiftedX = x - shiftX;
        double y = f.getFunctionValue(shiftedX);
        if (Double.isNaN(y)) {
            return Double.NaN;
        }
        return y + shiftY;
    }
}
```

#### Код 4.4

```
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() {
        return f.getLeftDomainBorder() * scaleX;
    }

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

    @Override
    public double getFunctionValue(double x) {
        double scaledX = x / scaleX;
        double y = f.getFunctionValue(scaledX);
        if (Double.isNaN(y)) {
            return Double.NaN;
        }
        return y * scaleY;
    }
}
```

Код 4.5

```
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) {
        double y = f.getFunctionValue(x);
        if (Double.isNaN(y)) {
            return Double.NaN;
        }
        return Math.pow(y, power);
    }
}
```

**Задание 5:** В пакете functions создан класс Functions, содержащий вспомогательные статические методы для работы с функциями. В программе вне этого класса нельзя создать его объект. Класс содержит следующие методы:

- 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.

```
package functions;

import functions.meta.*;

public final class Functions {
    private Functions() {
    }

    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 f1, Function f2) {
        return new Sum(f1, f2);
    }

    public static Function mult(Function f1, Function f2) {
        return new Mult(f1, f2);
    }

    public static Function composition(Function f1, Function f2) {
        return new Composition(f1, f2);
    }
}
```

## Код 5

**Задание 6:** В пакете functions создан класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями. В программе вне этого класса нельзя было создать его объект.

Описан в классе метод public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount), получающий функцию и возвращающий её табулированный аналог на заданном отрезке с заданным количеством точек.

```
public final class TabulatedFunctions {
    private TabulatedFunctions() {
    }
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Points count must be at least 2");
        }
        if (leftX < function.getLeftDomainBorder() - 1e-10 || rightX > function.getRightDomainBorder() + 1e-10) {
            throw new IllegalArgumentException("Tabulation interval is outside function domain");
        }
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            double y = function.getFunctionValue(x);
            points[i] = new FunctionPoint(x, y);
        }
        return new ArrayTabulatedFunction(points);
    }
}
```

## Код 6

**Задание 7:** В класс TabulatedFunctions добавлены следующие методы.

Метод вывода табулированной функции в байтовый поток public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) в указанный поток выводит значения, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек.

Метод ввода табулированной функции из байтового потока public static TabulatedFunction inputTabulatedFunction(InputStream in) считывает из указанного потока данные о табулированной функции, создаёт и настраивает её объект и возвращать его из метода.

Метод записи табулированной функции в символьный поток public static void writeTabulatedFunction(TabulatedFunction function, Writer out) в указанный поток выводит значения, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек.

Метод чтения табулированной функции из символьного потока public static TabulatedFunction readTabulatedFunction(Reader in) считывает из указанного потока данные о табулированной функции, создаёт и настраивает её объект и возвращает его из метода.

```
26     public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) {
27         try (DataOutputStream dos = new DataOutputStream(out)) {
28             dos.writeInt(function.getPointsCount());
29             for (int i = 0; i < function.getPointsCount(); i++) {
30                 FunctionPoint point = function.getPoint(i);
31                 dos.writeDouble(point.getX());
32                 dos.writeDouble(point.getY());
33             }
34         } catch (IOException e) {
35             throw new RuntimeException("Error writing tabulated function to stream", e);
36         }
37     }
38     public static TabulatedFunction inputTabulatedFunction(InputStream in) {
39         try (DataInputStream dis = new DataInputStream(in)) {
40             int pointsCount = dis.readInt();
41             if (pointsCount < 2) {
42                 throw new IllegalArgumentException("Invalid data: points count must be at least 2");
43             }
44             FunctionPoint[] points = new FunctionPoint[pointsCount];
45             for (int i = 0; i < pointsCount; i++) {
46                 double x = dis.readDouble();
47                 double y = dis.readDouble();
48                 points[i] = new FunctionPoint(x, y);
49             }
50             return new ArrayTabulatedFunction(points);
51         } catch (IOException e) {
52             throw new RuntimeException("Error reading tabulated function from stream", e);
53         }
54     }
```

## Код 7.1

```
35         throw new RuntimeException("Error writing tabulated function to stream", e);
36     }
37 }
38 public static TabulatedFunction inputTabulatedFunction(InputStream in) {
39     try (DataInputStream dis = new DataInputStream(in)) {
40         int pointsCount = dis.readInt();
41         if (pointsCount < 2) {
42             throw new IllegalArgumentException("Invalid data: points count must be at least 2");
43         }
44         FunctionPoint[] points = new FunctionPoint[pointsCount];
45         for (int i = 0; i < pointsCount; i++) {
46             double x = dis.readDouble();
47             double y = dis.readDouble();
48             points[i] = new FunctionPoint(x, y);
49         }
50         return new ArrayTabulatedFunction(points);
51     } catch (IOException e) {
52         throw new RuntimeException("Error reading tabulated function from stream", e);
53     }
54 }
55 public static void writeTabulatedFunction(TabulatedFunction function, Writer out) {
56     try (PrintWriter writer = new PrintWriter(out)) {
57         writer.print(function.getPointsCount());
58         for (int i = 0; i < function.getPointsCount(); i++) {
59             FunctionPoint point = function.getPoint(i);
60             writer.print(" " + point.getX() + " " + point.getY());
61         }
62         writer.flush();
63     } catch (Exception e) {
64         throw new RuntimeException("Error writing tabulated function to writer", e);
65     }
66 }
67 public static TabulatedFunction readTabulatedFunction(Reader in) {
```

## Код 7.2

```

67     public static TabulatedFunction readTabulatedFunction(Reader in) {
68         try {
69             StreamTokenizer tokenizer = new StreamTokenizer(in);
70             tokenizer.parseNumbers();
71             if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
72                 throw new IllegalArgumentException("Expected number of points");
73             }
74             int pointsCount = (int) tokenizer.nval;
75             if (pointsCount < 2) {
76                 throw new IllegalArgumentException("Points count must be at least 2");
77             }
78             FunctionPoint[] points = new FunctionPoint[pointsCount];
79             for (int i = 0; i < pointsCount; i++) {
80                 if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
81                     throw new IllegalArgumentException("Expected X coordinate");
82                 }
83                 double x = tokenizer.nval;
84                 if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
85                     throw new IllegalArgumentException("Expected Y coordinate");
86                 }
87                 double y = tokenizer.nval;
88                 points[i] = new FunctionPoint(x, y);
89             }
90             return new ArrayTabulatedFunction(points);
91         } catch (IOException e) {
92             throw new RuntimeException("Error reading tabulated function from reader", e);
93         }
94     }
95 }
```

### Код 7.3

В реализованных методах исключения IOException перехватываются и обрабатываются в непроверяемое RuntimeException с сохранением оригинальной причины через цепочку исключений. Такой подход выбран для упрощения API — пользовательский код освобождается от необходимости явной обработки проверяемых исключений ввода-вывода, что повышает удобство использования методов. При этом информация об ошибке не теряется и может быть получена при необходимости через getCause().

Методы не закрывают переданные им потоки (InputStream, OutputStream, Reader, Writer), оставляя ответственность за их жизненный цикл вызывающему коду. Это соответствует принципу «кто создал, тот и закрывает» и позволяет гибко переиспользовать потоки для последующих операций. Внутри методов создаются только обёрточные потоки (DataInputStream, PrintWriter и т.п.), которые автоматически закрываются с использованием конструкции try-with-resources, так как их время жизни ограничено рамками метода.

**Задание 8:** В файле Main.java была написана проверка всех созданных классов. Все реализованные классы работают корректно. Аналитические функции (Sin, Cos, Exp, Log) вычисляют значения точно, табулированные функции (ArrayTabulatedFunction, LinkedListTabulatedFunction) правильно интерполируют значения между узлами. Операции над табулированными функциями (возведение в квадрат, сложение) усиливают погрешность

интерполяции.

**Текстовый формат** можно прочитать человеку, но занимает больше места и операции с ним проходят медленнее. **Бинарный формат** компактнее и быстрее, но нечитаем для человека. Оба формата обеспечивают точное восстановление данных.

```
1 import functions.*;
2 import functions.basic.*;
3 import functions.meta.*;
4 import java.io.*;
5
6 public class Main {
7     public static void main(String[] args) {
8         try {
9
10             System.out.println("==> Часть 1: Создание объектов Sin и Cos ==>");
11             Sin sin = new Sin();
12             Cos cos = new Cos();
13
14             System.out.println("Значения Sin и Cos на отрезке [0, π] с шагом 0.1:");
15             System.out.println(" x\t\tSin(x)\t\tCos(x)");
16             System.out.println("-----");
17             for (double x = 0; x <= Math.PI + 1e-10; x += 0.1) {
18                 System.out.printf("%6.2f\t%10.6f\t%10.6f\n",
19                     x, sin.getFunctionValue(x), cos.getFunctionValue(x));
20             }
21
22             System.out.println("\n\n==> Часть 2: Табулирование Sin и Cos (10 точек) ==>");
23             TabulatedFunction tabSin = TabulatedFunctions.tabulate(sin, 0, Math.PI, 10);
24             TabulatedFunction tabCos = TabulatedFunctions.tabulate(cos, 0, Math.PI, 10);
25
26             System.out.println("Сравнение аналитических и табулированных функций:");
27             System.out.println(" x\t\tSin(аналит)\t\tSin(табул)\t\tРазница\t\tCos(аналит)\t\tCos(табул)\t\tРазница");
28             System.out.println("-----");
29             for (double x = 0; x <= Math.PI + 1e-10; x += 0.1) {
30                 double sinAnalytic = sin.getFunctionValue(x);
31                 double sinTab = tabSin.getFunctionValue(x);
32                 double cosAnalytic = cos.getFunctionValue(x);
33                 double cosTab = tabCos.getFunctionValue(x);
34                 System.out.printf("%6.2f\t%10.6f\t%10.6f\t%10.6f\t%10.6f\t%10.6f\t%10.6f\n",
35                     x, sinAnalytic, sinTab, sinTab - sinAnalytic, cosAnalytic, cosTab, cosTab - cosAnalytic);
```

## Код 8.1

```

35             x, sinAnalytic, sinTab, sinAnalytic - sinTab,
36             cosAnalytic, cosTab, cosAnalytic - cosTab);
37         }
38
39         System.out.println("\n\n==> Часть 3: Сумма квадратов  $\sin^2 + \cos^2 ==$ ");
40
41         for (int points : new int[]{10, 20, 50, 100}) {
42             System.out.println("\n--> Количество точек табуляции: " + points + " --");
43             TabulatedFunction tabSinN = TabulatedFunctions.tabulate(sin, 0, Math.PI, points);
44             TabulatedFunction tabCosN = TabulatedFunctions.tabulate(cos, 0, Math.PI, points);
45
46             Function sinSquared = Functions.power(tabSinN, 2);
47             Function cosSquared = Functions.power(tabCosN, 2);
48             Function sumSquares = Functions.sum(sinSquared, cosSquared);
49
50             System.out.println(" x\t\tsin2+cos2\t0тклонение от 1");
51             System.out.println("-----");
52             double maxError = 0;
53             for (double x = 0; x <= Math.PI + 1e-10; x += 0.1) {
54                 double value = sumSquares.getFunctionValue(x);
55                 double error = Math.abs(value - 1.0);
56                 if (error > maxError) maxError = error;
57                 System.out.printf("%6.2f\t%12.8f\t%12.8f\n", x, value, error);
58             }
59             System.out.printf("Максимальная погрешность: %.8f\n", maxError);
60         }
61         System.out.println("\n\n==> Часть 4: Текстовый формат (экспонента) ==");
62         Exp exp = new Exp();
63         TabulatedFunction tabExp = TabulatedFunctions.tabulate(exp, 0, 10, 11);
64
65         System.out.println("Запись экспоненты в файл exp.txt...");
66         try (FileWriter writer = new FileWriter("exp.txt")) {
67             TabulatedFunctions.writeTabulatedFunction(tabExp, writer);
68         }

```

## Код 8.2

```
69
70     System.out.println("Чтение экспоненты из файла exp.txt...");
71     TabulatedFunction readExp;
72     try (FileReader reader = new FileReader("exp.txt")) {
73         readExp = TabulatedFunctions.readTabulatedFunction(reader);
74     }
75
76     System.out.println("\nСравнение исходной и считанной экспоненты:");
77     System.out.println("  x\t\tОригинал\tПрочитано\tРазница");
78     System.out.println("-----");
79     for (int i = 0; i <= 10; i++) {
80         double x = i;
81         double orig = tabExp.getFunctionValue(x);
82         double read = readExp.getFunctionValue(x);
83         System.out.printf("%6.1f\t%12.6f\t%12.6f\t%12.6f%n", x, orig, read, orig - read);
84     }
85
86     System.out.println("\nСодержимое файла exp.txt:");
87     try (BufferedReader br = new BufferedReader(new FileReader("exp.txt"))) {
88         String line;
89         while ((line = br.readLine()) != null) {
90             System.out.println(line);
91         }
92     }
93
94     System.out.println("\n\n==> Часть 5: Бинарный формат (логарифм) ===");
95     Log ln = new Log(Math.E);
96     TabulatedFunction tabLn = TabulatedFunctions.tabulate(ln, 1, 10, 10);
97
98     System.out.println("Запись логарифма в файл ln.bin...");
99     try (FileOutputStream fos = new FileOutputStream("ln.bin")) {
100        TabulatedFunctions.outputTabulatedFunction(tabLn, fos);
101    }
102
```

### Код 8.3

```
103     System.out.println("Чтение логарифма из файла ln.bin...");
104     TabulatedFunction readLn;
105     try (FileInputStream fis = new FileInputStream("ln.bin")) {
106         readLn = TabulatedFunctions.inputTabulatedFunction(fis);
107     }
108
109     System.out.println("\nСравнение исходного и считанного логарифма:");
110     System.out.println(" x\t\オリジнаl\tПрочитано\tРазница");
111     System.out.println("-----");
112     for (int i = 1; i <= 10; i++) {
113         double x = i;
114         double orig = tabLn.getFunctionValue(x);
115         double read = readLn.getFunctionValue(x);
116         System.out.printf("%6.1f\t%12.6f\t%12.6f\t%12.6f%n", x, orig, read, orig - read);
117     }
118
119     System.out.println("\n\n== Анализ форматов хранения ==");
120     File textFile = new File("exp.txt");
121     File binFile = new File("ln.bin");
122
123     System.out.println("Текстовый файл (exp.txt):");
124     System.out.println(" Размер: " + textFile.length() + " байт");
125
126     System.out.println("Бинарный файл (ln.bin):");
127     System.out.println(" Размер: " + binFile.length() + " байт");
128
129     int expectedBinarySize = 4 + (10 * 2 * 8);
130     System.out.println(" Ожидаемый размер: " + expectedBinarySize + " байт (4 + 10*2*8)");
131 } catch (Exception e) {
132     System.err.println("Ошибка во время выполнения теста:");
133     e.printStackTrace();
134 }
135 }
136 }
```

## Код 8.4

```

PS C:\Javalabs\lab4> [Console]::OutputEncoding = [System.Text.Encoding]::UTF8
PS C:\Javalabs\lab4> javac functions\*.java functions\basic\*.java functions\meta\*.java Main.java
PS C:\Javalabs\lab4> java Main
== Часть 1: Создание объектов Sin и Cos ==
Значения Sin и Cos на отрезке [0, π] с шагом 0.1:
  x          Sin(x)        Cos(x)
-----
  0,00    0,000000    1,000000
  0,10    0,099833    0,995004
  0,20    0,198669    0,980067
  0,30    0,295520    0,955336
  0,40    0,389418    0,921061
  0,50    0,479426    0,877583
  0,60    0,564642    0,825336
  0,70    0,644218    0,764842
  0,80    0,717356    0,696707
  0,90    0,783327    0,621610
  1,00    0,841471    0,540302
  1,10    0,891207    0,453596
  1,20    0,932039    0,362358
  1,30    0,963558    0,267499
  1,40    0,985450    0,169967
  1,50    0,997495    0,070737
  1,60    0,999574    -0,029200
  1,70    0,991665    -0,128844
  1,80    0,973848    -0,227202
  1,90    0,946300    -0,323290
  2,00    0,909297    -0,416147
  2,10    0,863209    -0,504846
  2,20    0,808496    -0,588501
  2,30    0,745705    -0,666276
  2,40    0,675463    -0,737394
  2,50    0,598472    -0,801144
  2,60    0,515501    -0,856889
  2,70    0,427380    -0,904072
  2,80    0,334988    -0,942222
  2,90    0,239249    -0,970958
  3,00    0,141120    -0,989992
  3,10    0,041581    -0,999135

```

## Реализация 8.1

== Часть 2: Табулирование Sin и Cos (10 точек) ==						
Сравнение аналитических и табулированных функций:						
x	Sin(аналит)	Sin(табул)	Разница	Cos(аналит)	Cos(табул)	Разница
0,00	0,000000	0,000000	0,000000	1,000000	1,000000	0,000000
0,10	0,099833	0,097982	0,001852	0,995004	0,982723	0,012281
0,20	0,198669	0,195963	0,002706	0,980067	0,965446	0,014620
0,30	0,295520	0,293945	0,001576	0,955336	0,948170	0,007167
0,40	0,389418	0,391926	-0,002508	0,921061	0,930893	-0,009832
0,50	0,479426	0,489908	-0,010482	0,877583	0,913616	-0,036033
0,60	0,564642	0,587889	-0,023247	0,825336	0,896339	-0,071004
0,70	0,644218	0,685871	-0,041653	0,764842	0,879062	-0,114220
0,80	0,717356	0,783852	-0,066496	0,696707	0,861786	-0,165079
0,90	0,783327	0,881834	-0,098507	0,621610	0,844509	-0,222899
1,00	0,841471	0,979816	-0,138345	0,540302	0,827232	-0,286930
1,10	0,891207	1,077797	-0,186590	0,453596	0,809955	-0,356359
1,20	0,932039	1,175779	-0,243740	0,362358	0,792679	-0,430321
1,30	0,963558	1,273760	-0,310202	0,267499	0,775402	-0,507903
1,40	0,985450	1,371742	-0,386292	0,169967	0,758125	-0,588158
1,50	0,997495	1,469723	-0,472228	0,070737	0,740848	-0,670111
1,60	0,999574	1,567705	-0,568131	-0,029200	0,723571	-0,752771
1,70	0,991665	1,666586	-0,674022	-0,128844	0,706295	-0,835139
1,80	0,973848	1,763668	-0,789820	-0,227202	0,689018	-0,916220
1,90	0,946300	1,861650	-0,915349	-0,323290	0,671741	-0,995031
2,00	0,909297	1,959631	-1,050334	-0,416147	0,654464	-1,070611
2,10	0,863209	2,057613	-1,194403	-0,504846	0,637187	-1,142033
2,20	0,808496	2,155594	-1,347098	-0,588501	0,619911	-1,208412
2,30	0,745705	2,253576	-1,507871	-0,666276	0,602634	-1,268910
2,40	0,675463	2,351557	-1,676094	-0,737394	0,585357	-1,322751
2,50	0,598472	2,449539	-1,851067	-0,801144	0,568080	-1,369224
2,60	0,515501	2,547520	-2,032019	-0,856889	0,550803	-1,407692
2,70	0,427380	2,645502	-2,218122	-0,904072	0,533527	-1,437599
2,80	0,334988	2,743484	-2,408495	-0,942222	0,516250	-1,458472
2,90	0,239249	2,841465	-2,602216	-0,970958	0,498973	-1,469931
3,00	0,141120	2,939447	-2,798327	-0,989992	0,481696	-1,471689
3,10	0,041581	3,037428	-2,995847	-0,999135	0,464419	-1,463555

## Реализация 8.2

==== Часть 3: Сумма квадратов  $\sin^2 + \cos^2$  ===

--- Количество точек табуляции: 10 ---

x	$\sin^2 + \cos^2$	Отклонение от 1
0,00	1,00000000	0,00000000
0,10	0,97534529	0,02465471
0,20	0,97048832	0,02951168
0,30	0,98542910	0,01457090
0,40	0,98496848	0,01503152
0,50	0,97039758	0,02960242
0,60	0,97562443	0,02437557
0,70	0,99935789	0,00064211
0,80	0,97507306	0,02492694
0,90	0,97058598	0,02941402
1,00	0,98589664	0,01410336
1,10	0,98451477	0,01548523
1,20	0,97031375	0,02968625
1,30	0,97591048	0,02408952
1,40	0,99872269	0,00127731
1,50	0,97480774	0,02519226
1,60	0,97069054	0,02930946
1,70	0,98637108	0,01362892
1,80	0,98406796	0,01593204
1,90	0,97023683	0,02976317
2,00	0,97620344	0,02379656
2,10	0,99809440	0,00190560
2,20	0,97454934	0,02545066
2,30	0,97080201	0,02919799
2,40	0,98685244	0,01314756
2,50	0,98362807	0,01637193
2,60	0,97016682	0,02983318
2,70	0,97650331	0,02349669
2,80	0,99747303	0,00252697
2,90	0,97429784	0,02570216
3,00	0,97092040	0,02907960
3,10	0,98734070	0,01265930

Максимальная погрешность: 0,02983318

Реализация 8.3

--- Количество точек табуляции: 20 ---		
x	$\sin^2 + \cos^2$	Отклонение от 1
0,00	1,00000000	0,00000000
0,10	0,99348018	0,00651982
0,20	0,99548137	0,00451863
0,30	0,99587637	0,00412363
0,40	0,99335893	0,00664107
0,50	0,99936251	0,00063749
0,60	0,99363270	0,00636730
0,70	0,99511764	0,00488236
0,80	0,99630265	0,00369735
0,90	0,99326897	0,00673103
1,00	0,99875630	0,00124370
1,10	0,99381649	0,00618351
1,20	0,99478519	0,00521481
1,30	0,99676021	0,00323979
1,40	0,99321028	0,00678972
1,50	0,99818136	0,00181864
1,60	0,99403156	0,00596844
1,70	0,99448402	0,00551598
1,80	0,99724905	0,00275095
1,90	0,99318287	0,00681713
2,00	0,99763770	0,00236230
2,10	0,99427791	0,00572209
2,20	0,99421412	0,00578588
2,30	0,99776916	0,00223084
2,40	0,99318673	0,00681327
2,50	0,99712532	0,00287468
2,60	0,99455554	0,00544446
2,70	0,99397550	0,00602450
2,80	0,99832055	0,00167945
2,90	0,99322187	0,00677813
3,00	0,99664422	0,00335578
3,10	0,99486445	0,00513555
Максимальная погрешность: 0,00681713		

Реализация 8.4

--- Количество точек табуляции: 50 ---

x	$\sin^2 + \cos^2$	Отклонение от 1
0,00	1,00000000	0,00000000
0,10	0,99898735	0,00101265
0,20	0,99956783	0,00043217
0,30	0,99910459	0,00089541
0,40	0,99925289	0,00074711
0,50	0,99933906	0,00066094
0,60	0,99905519	0,00094481
0,70	0,99969077	0,00030923
0,80	0,99897473	0,00102527
0,90	0,99985181	0,00014819
1,00	0,99901151	0,00098849
1,10	0,99945642	0,00054358
1,20	0,99916552	0,00083448
1,30	0,99917826	0,00082174
1,40	0,99943677	0,00056323
1,50	0,99901733	0,00098267
1,60	0,99982526	0,00017474
1,70	0,99897365	0,00102635
1,80	0,99971516	0,00028484
1,90	0,99904720	0,00095280
2,00	0,99935654	0,00064346
2,10	0,99923799	0,00076201
2,20	0,99911516	0,00088484
2,30	0,99954602	0,00045398
2,40	0,99899101	0,00100899
2,50	0,99997128	0,00002872
2,60	0,99898410	0,00101590
2,70	0,99959005	0,00040995
2,80	0,99909443	0,00090557
2,90	0,99926820	0,00073180
3,00	0,99932199	0,00067801
3,10	0,99906359	0,00093641

Максимальная погрешность: 0,00102635

Реализация 8.5

--- Количество точек табуляции: 100 ---		
x	$\sin^2 + \cos^2$	Отклонение от 1
0,00	1,00000000	0,00000000
0,10	0,99987073	0,00012927
0,20	0,99978753	0,00021247
0,30	0,99975042	0,00024958
0,40	0,99975939	0,00024061
0,50	0,99981444	0,00018556
0,60	0,99991556	0,00008444
0,70	0,99994421	0,00005579
0,80	0,99983287	0,00016713
0,90	0,99976761	0,00023239
1,00	0,99974843	0,00025157
1,10	0,99977534	0,00022466
1,20	0,99984832	0,00015168
1,30	0,99996738	0,00003262
1,40	0,99989540	0,00010460
1,50	0,99980199	0,00019801
1,60	0,99975467	0,00024533
1,70	0,99975343	0,00024657
1,80	0,99979826	0,00020174
1,90	0,99988918	0,00011082
2,00	0,99997511	0,00002489
2,10	0,99985357	0,00014643
2,20	0,99977810	0,00022190
2,30	0,99974871	0,00025129
2,40	0,99976540	0,00023460
2,50	0,99982817	0,00017183
2,60	0,99993703	0,00006297
2,70	0,99992233	0,00007767
2,80	0,99981871	0,00018129
2,90	0,99976118	0,00023882
3,00	0,99974973	0,00025027
3,10	0,99978436	0,00021564
Максимальная погрешность: 0,00025157		

## Реализация 8.6

```

==== Часть 4: Текстовый формат (экспонента) ====
Запись экспоненты в файл exp.txt...
Чтение экспоненты из файла exp.txt...

Сравнение исходной и считанной экспоненты:
  x      Оригинал    Прочитано    Разница
-----
```

0,0	1,000000	1,000000	0,000000
1,0	2,718282	2,718282	0,000000
2,0	7,389056	7,389056	0,000000
3,0	20,085537	20,085537	0,000000
4,0	54,598150	54,598150	-0,000000
5,0	148,413159	148,413159	0,000000
6,0	403,428793	403,428793	0,000000
7,0	1096,633158	1096,633158	0,000000
8,0	2980,957987	2980,957987	0,000000
9,0	8103,083928	8103,083928	0,000000
10,0	22826,465795	22826,465795	-0,000000

Содержимое файла exp.txt:  
11 0.0 1.0 1.0 2.718281828459045 2.0 7.38905609893065 3.0 20.085536923187668 4.0 54.598150033144236 5.0 148.4131591025766 6.0 403.4287934927351 7.0  
1096.6331584284585 8.0 2980.9579870417283 9.0 8103.083928 10.0 22826.465794806718

```

==== Часть 5: Бинарный формат (логарифм) ====
Запись логарифма в файл ln.bin...
Чтение логарифма из файла ln.bin...

Сравнение исходного и считанного логарифма:
  x      Оригинал    Прочитано    Разница
-----
```

1,0	0,000000	0,000000	0,000000
2,0	0,693147	0,693147	0,000000
3,0	1,098612	1,098612	0,000000
4,0	1,386294	1,386294	0,000000
5,0	1,609438	1,609438	0,000000
6,0	1,791759	1,791759	0,000000
7,0	1,945910	1,945910	0,000000
8,0	2,079442	2,079442	0,000000
9,0	2,197225	2,197225	0,000000
10,0	2,302585	2,302585	0,000000

## Реализация 8.

### ==== Анализ форматов хранения ====

**Текстовый файл (exp.txt):**

**Размер: 235 байт**

**Бинарный файл (ln.bin):**

**Размер: 164 байт**

## Реализация 8.

**Задание 9:** Объекты всех классов, реализующих интерфейс TabulatedFunction, были сделаны сериализуемыми. Были модифицированы все классы, представляющие табулированные функции (FunctionPoint, ArrayTabulatedFunction, LinkedListTabulatedFunction) для поддержки сериализации через интерфейс Serializable. Каждому классу присвоен уникальный serialVersionUID, что гарантирует обратную совместимость при изменениях. Сериализованные объекты корректно сохраняются в файлы и восстанавливаются без потери данных и функциональности.

**Serializable** обеспечивает простоту реализации (достаточно implements интерфейса) и автоматическую сериализацию всех полей, но создаёт крупные файлы из-за метаданных классов. **Externalizable** даёт полный контроль над процессом, позволяя оптимизировать размер файла и повысить производительность, однако требует ручной реализации методов и более подвержен ошибкам. Для табулированных функций предпочтительнее

Externalizable, так как можно сериализовать только координаты точек, исключив внутренние структуры данных.

```
1 package functions;
2
3 import java.io.*;
4
5 public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
6     private static final long serialVersionUID = 3L;
7     protected class FunctionNode implements Serializable {
8         private static final long serialVersionUID = 4L;
```

## Код 9.1

```
349     @Override
350     public void writeExternal(ObjectOutput out) throws IOException {
351         out.writeInt(pointsCount);
352
353         FunctionNode current = head.next;
354         while (current != head) {
355             out.writeDouble(current.point.getX());
356             out.writeDouble(current.point.getY());
357             current = current.next;
358         }
359     }
360
361     @Override
362     public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
363         head = new FunctionNode();
364         head.prev = head;
365         head.next = head;
366         pointsCount = 0;
367         lastAccessedNode = head;
368         lastAccessedIndex = -1;
369
370         int count = in.readInt();
371         for (int i = 0; i < count; i++) {
372             double x = in.readDouble();
373             double y = in.readDouble();
374             try {
375                 addPoint(new FunctionPoint(x, y));
376             } catch (InappropriateFunctionPointException e) {
377                 throw new IOException("Ошибка при чтении функции", e);
378             }
379         }
380     }
381 }
```

## Код 9.2 (методы в LinkedListTabulatedFunction)

```
package functions;
import java.io.Serializable;

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

### Код 9.3

```
2 import java.io.Serializable;
3 public class FunctionPoint implements Serializable {
4     private static final long serialVersionUID = 1L;
```

### Код 9.4

```
120     System.out.println("\n\n==> Часть 6: СерIALIZАЦИЯ (Задание 9) ==>");
121     Log lnForComp = new Log(Math.E);
122     Exp expForComp = new Exp();
123     Composition lnExp = new Composition(lnForComp, expForComp);
124
125     System.out.println("Табулирование функции ln(exp(x)) = x на [0, 10] с 11 точками...");
126     TabulatedFunction tabLnExp = TabulatedFunctions.tabulate(lnExp, 0, 10, 11);
127
128     System.out.println("\n1. СерIALIZАЦИЯ через Serializable");
129     File serFile1 = new File("serializable_func.ser");
130
131     try (ObjectOutputStream oos = new ObjectOutputStream(
132             new FileOutputStream(serFile1))) {
133         oos.writeObject(tabLnExp);
134         System.out.println("Создан файл: " + serFile1.getName());
135         System.out.println("Размер файла: " + serFile1.length() + " байт");
136     }
137
138     TabulatedFunction serialized1;
139     try (ObjectInputStream ois = new ObjectInputStream(
140             new FileInputStream(serFile1))) {
141         serialized1 = (TabulatedFunction) ois.readObject();
142     }
143
144     System.out.println("\nПроверка корректности десериализации:");
145     System.out.println(" x\t\tОригинал\tДесериал.\tСовпадает?");
146     System.out.println("-----");
147     boolean allMatch1 = true;
148     for (int i = 0; i <= 10; i++) {
149         double x = i;
150         double orig = tabLnExp.getFunctionValue(x);
151         double deser = serialized1.getFunctionValue(x);
152         boolean match = Math.abs(orig - deser) < 1e-10;
```

### Код 9.5

```

153     if (!match) allMatch1 = false;
154     System.out.printf("%6.1f\t%12.6f\t%12.6f\t%10s\n",
155         x, orig, deser, match ? "✓" : "✗");
156 }
157 System.out.println("Все значения совпадают: " + (allMatch1 ? "ДА" : "НЕТ"));
158
159 System.out.println("\nАнализ содержимого Serializable файла:");
160 System.out.println("Первые 100 байт файла (hex):");
161 try (FileInputStream fis = new FileInputStream(serFile1)) {
162     byte[] buffer = new byte[100];
163     int bytesRead = fis.read(buffer);
164     for (int i = 0; i < bytesRead; i++) {
165         if (i % 16 == 0) System.out.printf("\n%04X: ", i);
166         System.out.printf("%02X ", buffer[i] & 0xFF);
167     }
168     System.out.println();
169 }
170
171 System.out.println("\n2. СерIALIZАЦИЯ через Externalizable (LinkedListTabulatedFunction)");
172 File serFile2 = new File("externalizable_linkedList.ser");
173
174 FunctionPoint[] pointsArray = new FunctionPoint[11];
175 for (int i = 0; i <= 10; i++) {
176     double x = i;
177     double y = x; // ln(exp(x)) = x
178     pointsArray[i] = new FunctionPoint(x, y);
179 }
180
181 LinkedListTabulatedFunction linkedListFunc = new LinkedListTabulatedFunction(pointsArray);
182
183 try (ObjectOutputStream oos = new ObjectOutputStream(
184     new FileOutputStream(serFile2))) {
185     oos.writeObject(linkedListFunc);
186     System.out.println("Создан файл: " + serFile2.getName());

```

## Код 9.6

```

187         System.out.println("Размер файла: " + serFile2.length() + " байт");
188     }
189
190     LinkedListTabulatedFunction serializedLinkedList;
191     try (ObjectInputStream ois = new ObjectInputStream(
192             new FileInputStream(serFile2))) {
193         serializedLinkedList = (LinkedListTabulatedFunction) ois.readObject();
194     }
195
196     System.out.println("\nПроверка LinkedListTabulatedFunction (Externalizable):");
197     System.out.println(" x\t\tОригинал\tДесериал.\tСовпадает?");
198     System.out.println("-----");
199     boolean allMatchLinkedList = true;
200     for (int i = 0; i <= 10; i++) {
201         double x = i;
202         double orig = linkedListFunc.getFunctionValue(x);
203         double deser = serializedLinkedList.getFunctionValue(x);
204         boolean match = Math.abs(orig - deser) < 1e-10;
205         if (!match) allMatchLinkedList = false;
206         System.out.printf("%6.1f\t%12.6f\t%12.6f\t%10s%n",
207                           x, orig, deser, match ? "✓" : "✗");
208     }
209     System.out.println("Все значения совпадают: " + (allMatchLinkedList ? "ДА" : "НЕТ"));
210
211     System.out.println("\nАнализ содержимого Externalizable файла:");
212     System.out.println("Первые 100 байт файла (hex):");
213     try (FileInputStream fis = new FileInputStream(serFile2)) {
214         byte[] buffer = new byte[100];
215         int bytesRead = fis.read(buffer);
216         for (int i = 0; i < bytesRead; i++) {
217             if (i % 16 == 0) System.out.printf("\n%04X: ", i);
218             System.out.printf("%02X ", buffer[i] & 0xFF);
219         }
220         System.out.println();
221     }

```

## Код 9.7

```

221     }
222
223     System.out.println("\n== Сравнение Serializable и Externalizable ==");
224     System.out.println("\nРазмеры файлов:");
225     System.out.println(" Serializable: " + serFile1.length() + " байт");
226     System.out.println(" Externalizable: " + serFile2.length() + " байт");
227     System.out.println(" Разница: " + (serFile1.length() - serFile2.length()) + " байт");
228
229 } catch (Exception e) {
230     System.err.println("Ошибка во время выполнения теста:");
231     e.printStackTrace();
232 }
233 }
234 }

```

## Код 9.8

==== Часть 6: Сериализация (Задание 9) ===

Табулирование функции  $\ln(\exp(x)) = x$  на  $[0, 10]$  с 11 точками...

### 1. Сериализация через Serializable

Создан файл: `serializable_func.ser`

Размер файла: 450 байт

Проверка корректности десериализации:

x	Оригинал	Десериал.	Совпадает?
0, 0	0,000000	0,000000	✓
1, 0	1,000000	1,000000	✓
2, 0	2,000000	2,000000	✓
3, 0	3,000000	3,000000	✓
4, 0	4,000000	4,000000	✓
5, 0	5,000000	5,000000	✓
6, 0	6,000000	6,000000	✓
7, 0	7,000000	7,000000	✓
8, 0	8,000000	8,000000	✓
9, 0	9,000000	9,000000	✓
10, 0	10,000000	10,000000	✓

Все значения совпадают: ДА

Анализ содержимого Serializable файла:

Первые 100 байт файла (hex):

```
0000: AC ED 00 05 73 72 00 20 66 75 6E 63 74 69 6F 6E
0010: 73 2E 41 72 72 61 79 54 61 62 75 6C 61 74 65 64
0020: 46 75 6E 63 74 69 6F 6E 00 00 00 00 00 00 00 02
0030: 02 00 02 49 00 0B 70 6F 69 6E 74 73 43 6F 75 6E
0040: 74 5B 00 06 70 6F 69 6E 74 73 74 00 1A 5B 4C 66
0050: 75 6E 63 74 69 6F 6E 73 2F 46 75 6E 63 74 69 6F
0060: 6E 50 6F 69
```

### Реализация 9.1

## 2. Сериализация через Externalizable (LinkedListTabulatedFunction)

Создан файл: externalizable\_linkedList.ser

Размер файла: 241 байт

Проверка LinkedListTabulatedFunction (Externalizable):

x	Оригинал	Десериал.	Совпадает?
0, 0	0,000000	0,000000	✓
1, 0	1,000000	1,000000	✓
2, 0	2,000000	2,000000	✓
3, 0	3,000000	3,000000	✓
4, 0	4,000000	4,000000	✓
5, 0	5,000000	5,000000	✓
6, 0	6,000000	6,000000	✓
7, 0	7,000000	7,000000	✓
8, 0	8,000000	8,000000	✓
9, 0	9,000000	9,000000	✓
10, 0	10,000000	10,000000	✓

Все значения совпадают: ДА

Анализ содержимого Externalizable файла:

Первые 100 байт файла (hex):

```
0000: AC ED 00 05 73 72 00 25 66 75 6E 63 74 69 6F 6E
0010: 73 2E 4C 69 6E 6B 65 64 4C 69 73 74 54 61 62 75
0020: 6C 61 74 65 64 46 75 6E 63 74 69 6F 6E 00 00 00
0030: 00 00 00 00 03 0C 00 00 78 70 77 B4 00 00 00 0B
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050: 3F F0 00 00 00 00 00 00 3F F0 00 00 00 00 00 00
0060: 40 00 00 00
```

==== Сравнение Serializable и Externalizable ===

Размеры файлов:

Serializable: 450 байт

Externalizable: 241 байт

Разница: 209 байт

## Реализация 9.2