

Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное

образовательное учреждение высшего образования

«Самарский национальный исследовательский университет имени  
академика С.П. Королева»

Институт информатики и кибернетики Кафедра

технической кибернетики

Отчет по лабораторной работе №4

Дисциплина: «ООП»

Выполнил: Дорофеева Александра Алексеевна

Группа: 6201-120303D

Проверил: преподаватель Борисов Д.С.

Самара, 2025

## Задание 1

```
@  
    public ArrayTabulatedFunction(FunctionPoint[] points) { 3 usages new *  
        if (points.length < 2) {  
            throw new IllegalArgumentException("Количество точек должно быть не меньше двух");  
        }  
  
        for (int i = 1; i < points.length; i++) {  
            if (points[i].getX() <= points[i - 1].getX() + EPSILON) {  
                throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");  
            }  
        }  
  
        this.pointsCount = points.length;  
        this.points = new FunctionPoint[pointsCount + 5];  
        for (int i = 0; i < pointsCount; i++) {  
            this.points[i] = new FunctionPoint(points[i]);  
        }  
    }
```

Реализация конструктора `ArrayTabulatedFunction` с массивом точек с проверкой упорядоченности

```
public LinkedListTabulatedFunction(FunctionPoint[] points) { no usages Dorofeeva Alexsandra  
    if (points.length < 2) {  
        throw new IllegalArgumentException("Количество точек должно быть не меньше двух");  
    }  
  
    for (int i = 1; i < points.length; i++) {  
        if (points[i].getX() <= points[i - 1].getX() + EPSILON) {  
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");  
        }  
    }  
  
    initializeList();  
    for (FunctionPoint point : points) {  
        addNodeToTail().setPoint(new FunctionPoint(point));  
    }  
}
```

Реализация конструктора `LinkedListTabulatedFunction` с массивом точек

## Задание 2

```
1 package functions;  
2  
3 public interface Function { new *  
4     double getLeftDomainBorder(); 12 implementations new *  
5     double getRightDomainBorder(); 12 implementations new *  
6     double getFunctionValue(double x); 14 implementations new *  
7 }
```

```
package functions;  
  
import java.io.Serializable;  
  
public interface TabulatedFunction extends Function, Serializable { 25 usages 3 implementations & Dorofeeva Alexandra  
    int getPointsCount(); 16 usages 3 implementations & Dorofeeva Alexandra  
    FunctionPoint getPoint(int index); no usages 3 implementations & Dorofeeva Alexandra  
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; no usages 3 implementations & Dorofeeva Alexandra  
    double getPointX(int index); 7 usages 3 implementations & Dorofeeva Alexandra  
    void setPointX(int index, double x) throws InappropriateFunctionPointException; no usages 3 implementations & Dorofeeva Alexandra  
    double getPointY(int index); 7 usages 3 implementations & Dorofeeva Alexandra  
    void setPointY(int index, double y); no usages 3 implementations & Dorofeeva Alexandra  
    void deletePoint(int index); no usages 3 implementations & Dorofeeva Alexandra  
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; no usages 3 implementations & Dorofeeva Alexandra  
}
```

Создание интерфейса Function и наследование TabulatedFunction от него

Задание 3

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Exp implements Function { 3 usages new *
6     @Override new *
7     ⚡ public double getLeftDomainBorder() {
8         return Double.NEGATIVE_INFINITY;
9     }
10
11     @Override new *
12     ⚡ public double getRightDomainBorder() {
13         return Double.POSITIVE_INFINITY;
14     }
15
16     @Override new *
17     ⚡ public double getFunctionValue(double x) {
18         return Math.exp(x);
19     }
20 }
```

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Log implements Function { 4 usages new *
6     private double base; 2 usages
7     private static final double EPSILON = 1e-10; 2 usages
8
9     public Log(double base) { 4 usages new *
10         if (base <= 0 || Math.abs(base - 1) < 1e-10) {
11             throw new IllegalArgumentException("Основание логарифма должно быть положительным и не равно 1");
12         }
13         this.base = base;
14     }
15
16     @Override new *
17     public double getLeftDomainBorder() {
18         return 0;
19     }
20
21     @Override new *
22     public double getRightDomainBorder() {
23         return Double.POSITIVE_INFINITY;
24     }
25
26     @Override new *
27     public double getFunctionValue(double x) {
28         if (x < 0) {
29             return Double.NaN;
30         }
31         if (Math.abs(x) < EPSILON) { // x = 0
32             return Double.NEGATIVE_INFINITY;
33         }
34         // Для x = 1, ln(1) = 0
35         if (Math.abs(x - 1.0) < EPSILON) {
36             return 0.0;
37         }
38         return Math.log(x) / Math.log(base);
39     }
40 }
```

## Реализация классов Exp и Log, вычисляющих экспоненту и логарифм

```
1 package functions.basic;
2
3 import functions.Function;
4
5 @public abstract class TrigonometricFunction implements Function { 3 usages 3 inheritors new *
6     @Override new *
7     public double getLeftDomainBorder() {
8         return Double.NEGATIVE_INFINITY;
9     }
10
11     @Override new *
12     public double getRightDomainBorder() {
13         return Double.POSITIVE_INFINITY;
14     }
15 }
```

```
1 package functions.basic;
2
3 public class Cos extends TrigonometricFunction { 4 usages new *
4     @Override new *
5     public double getFunctionValue(double x) {
6         return Math.cos(x);
7     }
8 }
```

```
1 package functions.basic;
2
3 public class Sin extends TrigonometricFunction { 5 usages new *
4     @Override new *
5     public double getFunctionValue(double x) {
6         return Math.sin(x);
7     }
8 }
```

Реализация тригонометрических функций с использованием наследования

Задание 4

```
package functions.meta;

import functions.Function;

public class Sum implements Function { 1 usage  new *
    private Function f1, f2; 4 usages

    public Sum(Function f1, Function f2) { 1 usage  new *
        this.f1 = f1;
        this.f2 = f2;
    }

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

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

    @Override  new *
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Composition implements Function { 1 usage  new *
6     private Function f1, f2; 4 usages
7
8     public Composition(Function f1, Function f2) { 1 usage  new *
9         this.f1 = f1;
10        this.f2 = f2;
11    }
12
13     @Override  new *
14     public double getLeftDomainBorder() {
15         return f1.getLeftDomainBorder();
16     }
17
18     @Override  new *
19     public double getRightDomainBorder() {
20         return f1.getRightDomainBorder();
21     }
22
23     @Override  new *
24     public double getFunctionValue(double x) {
25         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
26             return Double.NaN;
27         }
28         double innerValue = f1.getFunctionValue(x);
29         return f2.getFunctionValue(innerValue);
30     }
31 }
```

## Реализация метафункций Sum и Composition

### Задание 5

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

Реализация утилитного класса Functions с фабричными методами для создания метафункций (сумма, умножение, возведение в степень, масштабирование, сдвиг, композиция)

## Задание 6

```
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {  
    if (pointsCount < 2) {  
        throw new IllegalArgumentException("Количество точек должно быть не меньше двух");  
    }  
    if (leftX >= rightX) {  
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");  
    }  
  
    double functionLeftBorder = function.getLeftDomainBorder();  
    double functionRightBorder = function.getRightDomainBorder();  
    if (leftX < functionLeftBorder || rightX > functionRightBorder) {  
        throw new IllegalArgumentException(  
            "Границы табулирования выходят за область определения функции");  
    }  
  
    double[] values = new double[pointsCount];  
    double step = (rightX - leftX) / (pointsCount - 1);  
    for (int i = 0; i < pointsCount; i++) {  
        double x = leftX + i * step;  
        values[i] = function.getFunctionValue(x);  
    }  
    return new ArrayTabulatedFunction(leftX, rightX, values);  
}
```

Реализация метода `tabulate` для табулирования функции на заданном интервале с проверкой корректности входных параметров и созданием объекта `ArrayTabulatedFunction`

Задание 7

```

public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException { 3 usages new *
    DataOutputStream dataOut = new DataOutputStream(out);
    int pointsCount = function.getPointsCount();
    dataOut.writeInt(pointsCount);

    for (int i = 0; i < pointsCount; i++) {
        dataOut.writeDouble(function.getPointX(i));
        dataOut.writeDouble(function.getPointY(i));
    }
    dataOut.flush();
}

public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException { 2 usages new *
    DataInputStream dataIn = new DataInputStream(in);
    int pointsCount = dataIn.readInt();

    if (pointsCount < 2) {
        throw new IOException("Некорректное количество точек: " + pointsCount);
    }

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

    return new ArrayTabulatedFunction(points);
}

```

Методы `outputTabulatedFunction()` и `inputTabulatedFunction()` реализуют запись и чтение табулированных функций в бинарном формате с использованием `DataOutputStream` и `DataInputStream`.

Формат данных: количество точек (int), затем пары координат X, Y (double).

```

public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException { 3 usages new *
    PrintWriter writer = new PrintWriter(out);
    int pointsCount = function.getPointsCount();
    writer.print(pointsCount);

    for (int i = 0; i < pointsCount; i++) {
        writer.print(" " + function.getPointX(i));
        writer.print(" " + function.getPointY(i));
    }

    writer.flush();
}

```

```
    public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException { 2 usages new *
        StreamTokenizer tokenizer = new StreamTokenizer(in);
        tokenizer.parseNumbers();

        tokenizer.nextToken();
        if (tokenizer.ttype != StreamTokenizer.TT_NUMBER) {
            throw new IOException("Ожидалось количество точек");
        }

        int pointsCount = (int) tokenizer.nval;
        if (pointsCount < 2) {
            throw new IOException("Некорректное количество точек: " + pointsCount);
        }

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            tokenizer.nextToken();
            if (tokenizer.ttype != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Ожидалась координата X точки " + i);
            }
            double x = tokenizer.nval;

            tokenizer.nextToken();
            if (tokenizer.ttype != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Ожидалась координата Y точки " + i);
            }
            double y = tokenizer.nval;

            points[i] = new FunctionPoint(x, y);
        }

        return new ArrayTabulatedFunction(points);
    }
}
```

Методы writeTabulatedFunction() и readTabulatedFunction() реализуют запись и чтение табулированных функций в текстовом формате. Для записи используется PrintWriter, для чтения – StreamTokenizer. Формат данных: количество точек, затем пары координат X, Y, разделенные пробелами

## Задание 8

### Тестирование

```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:G:\intellij IDEA\IntelliJ ID
== ЛАБОРАТОРНАЯ РАБОТА №4 ==
== ТЕСТИРОВАНИЕ ВСЕХ ЗАДАНИЙ ==

== ПРОВЕРКА ln(e^0) = 0 ==
1. Проверка exp(0):
exp(0) = 1.0 (ожидается 1.0)

2. Проверка ln(1):
ln(1) = 0.0 (ожидается 0.0)

3. Проверка ln(e^0):
ln(e^0) = 0.0 (ожидается 0.0)
✓ УСПЕХ: ln(e^0) = 0

=====
== ЗАДАНИЕ 3: БАЗОВЫЕ ФУНКЦИИ ==
1. Экспонента:
Область определения: [-Infinity, Infinity]
exp(0) = 1
exp(1) = 2,718282
exp(2) = 7,389056

2. Натуральный логарифм:
Область определения: [0.0, Infinity]
ln(1) = 0
ln(e) = 1
ln(10) = 2,302585

3. Тригонометрические функции:
sin(0) = 0
sin(pi/2) = 1
cos(0) = 1
cos(pi) = -1
tan(pi/4) = 1
```

==== ЗАДАНИЕ 4-5: МЕТАФУНКЦИИ ===

1. Сумма  $\sin(x) + \cos(x)$ :

при  $x=0$ : 1

при  $x=\pi/4$ : 1,414214

2. Произведение  $\sin(x) * \cos(x)$ :

при  $x=\pi/4$ : 0,5

3. Квадрат синуса  $\sin^2(x)$ :

при  $x=\pi/6$ : 0,25

4. Сдвинутый синус  $\sin(x-\pi/2)$ :

при  $x=\pi/2$ : 0

5. Масштабированный синус  $3*\sin(x/2)$ :

при  $x=\pi$ : 3

6. Композиция  $\sin(\cos(x))$ :

при  $x=0$ : 1

==== ЗАДАНИЕ 6: ТАБУЛИРОВАНИЕ ===

Табулированный синус на  $[0, \pi]$  (10 точек):

Количество точек: 10

Область определения:  $[0.0, 3.141592653589793]$

Точки функции:

Точка 0: x=0,000, y=0,000000

Точка 1: x=0,349, y=0,342020

Точка 2: x=0,698, y=0,642788

Точка 3: x=1,047, y=0,866025

Точка 4: x=1,396, y=0,984808

Точка 5: x=1,745, y=0,984808

Точка 6: x=2,094, y=0,866025

Точка 7: x=2,443, y=0,642788

Точка 8: x=2,793, y=0,342020

Точка 9: x=3,142, y=0,000000

Сравнение с точными значениями:

x=0,000: точное=0,000000, приближ=0,000000, ошибка=0,000000

x=0,524: точное=0,500000, приближ=0,492404, ошибка=0,007596

x=0,785: точное=0,707107, приближ=0,698597, ошибка=0,008510

x=1,047: точное=0,866025, приближ=0,866025, ошибка=0,000000

x=1,571: точное=1,000000, приближ=0,984808, ошибка=0,015192

x=2,094: точное=0,866025, приближ=0,866025, ошибка=0,000000

x=2,356: точное=0,707107, приближ=0,698597, ошибка=0,008510

x=2,618: точное=0,500000, приближ=0,492404, ошибка=0,007596

x=3,142: точное=0,000000, приближ=0,000000, ошибка=0,000000

```
== ЗАДАНИЕ 7: ВВОД/ВЫВОД ==
```

1. Исходная функция (парабола  $y=x^2$ ):

(0,00, 0,00) -> (1,00, 1,00) -> (2,00, 4,00) -> (3,00, 9,00) -> (4,00, 16,00)

2. Бинарный формат:

Записано байт: 84

Прочитано точек: 5

Функция восстановлена корректно: true

3. Текстовый формат:

Текст: 5 0.0 0.0 1.0 1.0 2.0 4.0 3.0 9.0 4.0 16.0

Длина текста: 42 символов

Прочитано точек: 5

Функция восстановлена корректно: true

4. Файловые операции:

Бинарный файл создан: function\_binary.dat

Текстовый файл создан: function\_text.txt

Размер бинарного файла: 84 байт

Размер текстового файла: 42 байт

```
== ЗАДАНИЕ 8: ПОЛНОЕ ТЕСТИРОВАНИЕ ==
```

1. Создание объектов Sin и Cos:

$\sin(\pi/6) = 0,5$

$\cos(\pi/3) = 0,5$

2. Табулированные аналоги (10 точек на  $[0, \pi]$ ):

Табулированный sin имеет 10 точек

Табулированный cos имеет 10 точек

3. Сумма квадратов  $\sin^2(x) + \cos^2(x)$ :

Проверка тождества  $\sin^2(x) + \cos^2(x) = 1$ :

```
x=0,0: 1,0000000000 (ожидается 1.0000000000)
x=0,2: 1,0000000000 (ожидается 1.0000000000)
x=0,4: 1,0000000000 (ожидается 1.0000000000)
x=0,6: 1,0000000000 (ожидается 1.0000000000)
x=0,8: 1,0000000000 (ожидается 1.0000000000)
x=1,0: 1,0000000000 (ожидается 1.0000000000)
x=1,2: 1,0000000000 (ожидается 1.0000000000)
x=1,4: 1,0000000000 (ожидается 1.0000000000)
x=1,6: 1,0000000000 (ожидается 1.0000000000)
x=1,8: 1,0000000000 (ожидается 1.0000000000)
x=2,0: 1,0000000000 (ожидается 1.0000000000)
x=2,2: 1,0000000000 (ожидается 1.0000000000)
x=2,4: 1,0000000000 (ожидается 1.0000000000)
x=2,6: 1,0000000000 (ожидается 1.0000000000)
x=2,8: 1,0000000000 (ожидается 1.0000000000)
x=3,0: 1,0000000000 (ожидается 1.0000000000)
```

4. Экспонента и файловые операции:

Экспонента записана в `exp_tabulated.txt`

Сравнение оригинальной и прочитанной экспоненты:

```
x=0: orig= 1,000000, read= 1,000000, diff=0,0000000000
x=1: orig= 2,718282, read= 2,718282, diff=0,0000000000
x=2: orig= 7,389056, read= 7,389056, diff=0,0000000000
x=3: orig= 20,085537, read= 20,085537, diff=0,0000000000
x=4: orig= 54,598150, read= 54,598150, diff=0,0000000000
x=5: orig=148,413159, read=148,413159, diff=0,0000000000
x=6: orig=403,428793, read=403,428793, diff=0,0000000000
x=7: orig=1096,633158, read=1096,633158, diff=0,0000000000
x=8: orig=2980,957987, read=2980,957987, diff=0,0000000000
x=9: orig=8103,083928, read=8103,083928, diff=0,0000000000
x=10: orig=22026,465795, read=22026,465795, diff=0,0000000000
```

5. Логарифм и бинарные операции:

Логарифм записан в `log_tabulated.dat`

Проверка корректности чтения:

Все значения совпадают!

```
package functions;

import java.io.Serializable;

public class ArrayTabulatedFunction implements TabulatedFunction, Serializable { 4 usages  Dorofeeva Alexandra *
    private int pointsCount; 37 usages
    private FunctionPoint[] points; 36 usages
    private static final double EPSILON = 1e-9; 13 usages
    private static final long serialVersionUID = 1L; no usages
```

Объявление класса ArrayTabulatedFunction, реализующего интерфейс TabulatedFunction с поддержкой сериализации (Serializable)

```
package functions;

import java.io.Serializable;

public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable { no usages
    private static final long serialVersionUID = 2L; no usages
```

```
// Для стандартной сериализации (не Externalizable)
private void writeObject(java.io.ObjectOutputStream out) throws java.io.IOException { no usages  Dorofeeva Alexandra *
    out.defaultWriteObject();
    out.writeInt(size);

    FunctionNode current = head.getNext();
    while (current != head) {
        out.writeDouble(current.getPoint().getX());
        out.writeDouble(current.getPoint().getY());
        current = current.getNext();
    }
}

private void readObject(java.io.ObjectInputStream in)  no usages  Dorofeeva Alexandra *
    throws java.io.IOException, ClassNotFoundException {
    in.defaultReadObject();
    int savedSize = in.readInt();

    initializeList();
    for (int i = 0; i < savedSize; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        addNodeToTail().setPoint(new FunctionPoint(x, y));
    }
}
```

Класс LinkedListTabulatedFunction с поддержкой Serializable и ручной реализацией сериализации через writeObject/readObject

```
package functions;

import java.io.*;

public class LinkedListTabulatedFunctionExternalizable implements TabulatedFunction, Externalizable { 4 usages new *
    private static final long serialVersionUID = 2L;  no usages
```

```
@Override  new *
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(size);

    FunctionNode current = head.getNext();
    while (current != head) {
        out.writeDouble(current.getPoint().getX());
        out.writeDouble(current.getPoint().getY());
        current = current.getNext();
    }
}

@Override  new *
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    int savedSize = in.readInt();

    initializeList();
    for (int i = 0; i < savedSize; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        addNodeToTail().setPoint(new FunctionPoint(x, y));
    }
}
```

Класс LinkedListTabulatedFunctionExternalizable, реализующий интерфейс Externalizable для ручного управления процессом сериализации

```
290
291     private static void testSerialization() throws IOException, ClassNotFoundException { 1 usage  new *
292         System.out.println("\n\n==== ЗАДАНИЕ 9: СЕРИАЛИЗАЦИЯ ===");
293
294         // 1. Проверка тождества ln(e^x) = x
295         System.out.println("\n1. Проверка тождества ln(e^x) = x:");
296         Function exp = new Exp();
297         Function log = new Log(Math.E);
298         Function logOfExp = Functions.composition(log, exp);
299
300         // Проверяем значения от 0 до 10
301         for (int i = 0; i <= 10; i++) {
302             double x = i;
303             double y = logOfExp.getFunctionValue(x);
304             System.out.printf("  x=%1f: ln(e^%1f) = %10f (ожидается %1f)%n",
305                 x, x, y, x);
306             // Проверяем, что значение близко к ожидаемому
307             if (Math.abs(y - x) > 1e-9) {
308                 System.out.printf("  ВНИМАНИЕ: разница = %10f%n", Math.abs(y - x));
309             }
310         }
311
312         // Создаем функцию для сериализации (композиция ln(e^x) = x)
313         TabulatedFunction tabLogExp = TabulatedFunctions.tabulate(logOfExp, [leftX: 0, rightX: 10, pointsCount: 11]);
314
315         System.out.println("\n2. Функция для сериализации: ln(e^x) = x");
316         System.out.println("  Количество точек: " + tabLogExp.getPointsCount());
317         System.out.println("  Все точки оригинала:");
318         for (int i = 0; i < tabLogExp.getPointsCount(); i++) {
319             System.out.printf("  Точка %d: x=%3f, y=%6fn",
320                 i, tabLogExp.getPointX(i), tabLogExp.getPointY(i));
321         }
322
323         // СерIALIZАЦИЯ с Serializable
324         System.out.println("\n3. СерIALIZАЦИЯ с использованием Serializable:");
325         try (ObjectOutputStream oos = new ObjectOutputStream(
326             new FileOutputStream( name: "function_serializable.dat")) {
327             oos.writeObject(tabLogExp);
328         }
```

```
335     File serializableFile = new File( pathname: "function_serializable.dat");
336     System.out.println("  Записано в: function_serializable.dat");
337     System.out.println("  Размер файла: " + serializableFile.length() + " байт");
338
339     // Десериализация
340     TabulatedFunction serialized;
341     try (ObjectInputStream ois = new ObjectInputStream(
342             new FileInputStream( name: "function_serializable.dat")) {
343         serialized = (TabulatedFunction) ois.readObject();
344     }
345
346     System.out.println("\n4. Проверка после десериализации Serializable:");
347     System.out.println("  Количество точек: " + serialized.getPointsCount());
348     System.out.println("  Все точки после десериализации:");
349     for (int i = 0; i < serialized.getPointsCount(); i++) {
350         System.out.printf("  Точка %d: x=%-.3f, y=%-.6f%n",
351                         i, serialized.getPointX(i), serialized.getPointY(i));
352     }
353
354     // Сравнение оригинальной и десериализованной функции
355     System.out.println("\n5. Сравнение оригинальной и десериализованной функции:");
356     boolean serializableMatch = compareFunctions(tabLogExp, serialized);
357     System.out.println("  Функции идентичны: " + serializableMatch);
358
359     if (!serializableMatch) {
360         System.out.println("  Расхождения:");
361         for (int i = 0; i < tabLogExp.getPointsCount(); i++) {
362             double origX = tabLogExp.getPointX(i);
363             double origY = tabLogExp.getPointY(i);
364             double readX = serialized.getPointX(i);
365             double readY = serialized.getPointY(i);
366
367             if (Math.abs(origX - readX) > 1e-9 || Math.abs(origY - readY) > 1e-9) {
368                 System.out.printf("  Точка %d: оригинал (%.6f, %.6f) vs прочитано (%.6f, %.6f)%n",
369                                 i, origX, origY, readX, readY);
370             }
371         }
372     }

```

```
371     }
372 }
373
374 // Создаем Externalizable версию (такой же набор точек для корректного сравнения)
375 System.out.println("\n6. Создание Externalizable версии:");
376 FunctionPoint[] points = new FunctionPoint[11];
377 for (int i = 0; i <= 10; i++) {
378     double x = i;
379     double y = logOfExp.getFunctionValue(x); // тоже ln(e^x) = x
380     points[i] = new FunctionPoint(x, y);
381 }
382 LinkedListTabulatedFunctionExternalizable externalizableFunc =
383     new LinkedListTabulatedFunctionExternalizable(points);
384
385 System.out.println("    Количество точек: " + externalizableFunc.getPointsCount());
386 System.out.println("    Все точки оригинала Externalizable:");
387 for (int i = 0; i < externalizableFunc.getPointsCount(); i++) {
388     System.out.printf("    Точка %d: x=%1f, y=%1f\n",
389         i, externalizableFunc.getPointX(i), externalizableFunc.getPointY(i));
390 }
391
392 // Сериализация с Externalizable
393 try (ObjectOutputStream oos = new ObjectOutputStream(
394     new FileOutputStream( name: "function_externalizable.dat")) ) {
395     oos.writeObject(externalizableFunc);
396 }
397 File externalizableFile = new File( pathname: "function_externalizable.dat");
398 System.out.println("\n7. СерIALIZАЦИЯ Externalizable:");
399 System.out.println("    Записано в: function_externalizable.dat");
400 System.out.println("    Размер файла: " + externalizableFile.length() + " байт");
401
402 // Десериализация
403 LinkedListTabulatedFunctionExternalizable serializedExternal;
404 try (ObjectInputStream ois = new ObjectInputStream(
405     new FileInputStream( name: "function_externalizable.dat")) ) {
406     serializedExternal = (LinkedListTabulatedFunctionExternalizable) ois.readObject();
407 }
```

```

08
09     System.out.println("\n8. Проверка после десериализации Externalizable:");
10    System.out.println("  Количество точек: " + serializedExternal.getPointsCount());
11    System.out.println("  Все точки после десериализации:");
12    for (int i = 0; i < serializedExternal.getPointsCount(); i++) {
13        System.out.printf("  Точка %d: x=%1f, y=%1f\n",
14                           i, serializedExternal.getPointX(i), serializedExternal.getPointY(i));
15    }
16
17    // Проверка совпадения Externalizable
18    System.out.println("\n9. Проверка совпадения Externalizable:");
19    boolean externalizableMatch = compareFunctions(externalizableFunc, serializedExternal);
20    System.out.println("  Функции идентичны: " + externalizableMatch);
21
22    System.out.println("\n10. Сравнение размеров файлов:");
23    System.out.println("  Serializable: " + serializableFile.length() + " байт");
24    System.out.println("  Externalizable: " + externalizableFile.length() + " байт");
25    System.out.println("  Разница: " +
26                       Math.abs(serializableFile.length() - externalizableFile.length()) + " байт");
27
28    System.out.println("\n11. Объяснение переопределения writeObject/readObject:");
29    System.out.println("  В LinkedListTabulatedFunction методы writeObject/readObject");
30    System.out.println("  переопределены для:");
31    System.out.println("    - Контроля над процессом сериализации");
32    System.out.println("    - Игнорирования transient полей (lastAccessedNode, lastAccessedIndex)");
33    System.out.println("    - Восстановления циклической структуры списка");
34    System.out.println("    - Избежания проблем с циклическими ссылками");
35    System.out.println("    - Сохранения только данных (координат точек), не структуры списка");
36    System.out.println("  В Externalizable версии:");
37    System.out.println("    - Полный контроль над форматом данных");
38    System.out.println("    - Более эффективное использование места");
39    System.out.println("    - Возможность обработки разных версий формата");
40}

```

Метод testSerialization, демонстрирующий процесс сериализации и десериализации функций с использованием Serializable и Externalizable, а также сравнение размеров файлов

==== ЗАДАНИЕ 9: СЕРИАЛИЗАЦИЯ ===

1. Проверка тождества  $\ln(e^x) = x$ :

Проверка аналитической функции  $\ln(e^x)$ :

```
x= 0: ln(e^0) = 0,0000000000 (ожидается 0.0)
x= 1: ln(e^1) = 1,0000000000 (ожидается 1.0)
x= 2: ln(e^2) = 2,0000000000 (ожидается 2.0)
x= 3: ln(e^3) = 3,0000000000 (ожидается 3.0)
x= 4: ln(e^4) = 4,0000000000 (ожидается 4.0)
x= 5: ln(e^5) = 5,0000000000 (ожидается 5.0)
x= 6: ln(e^6) = 6,0000000000 (ожидается 6.0)
x= 7: ln(e^7) = 7,0000000000 (ожидается 7.0)
x= 8: ln(e^8) = 8,0000000000 (ожидается 8.0)
x= 9: ln(e^9) = 9,0000000000 (ожидается 9.0)
x=10: ln(e^10) = 10,0000000000 (ожидается 10.0)
```

2. Создание табулированного аналога  $\ln(e^x)$  на  $[0, 10]$  с 11 точками:

Количество точек: 11

Точки функции:

```
Точка 0: x=0,0, y=0,000000
Точка 1: x=1,0, y=1,000000
Точка 2: x=2,0, y=2,000000
Точка 3: x=3,0, y=3,000000
Точка 4: x=4,0, y=4,000000
Точка 5: x=5,0, y=5,000000
Точка 6: x=6,0, y=6,000000
Точка 7: x=7,0, y=7,000000
Точка 8: x=8,0, y=8,000000
Точка 9: x=9,0, y=9,000000
Точка 10: x=10,0, y=10,000000
```

3. Сравнение Serializable vs Externalizable:

4. Сериализация ArrayTabulatedFunction (Serializable):

Записано в: array\_serializable.dat

Размер файла: 445 байт

5. Сериализация `LinkedListTabulatedFunction` (`Externalizable`):

Записано в: `linkedlist_externalizable.dat`

Размер файла: 241 байт

6. Содержимое файлов (первые 64 байта):

`array_serializable.dat` (`Serializable`):

```
AC ED 00 05 73 72 00 20 66 75 6E 63 74 69 6F 6E
73 2E 41 72 72 61 79 54 61 62 75 6C 61 74 65 64
46 75 6E 63 74 69 6F 6E 00 00 00 00 00 00 00 01
02 00 02 49 00 0B 70 6F 69 6E 74 73 43 6F 75 6E
```

Структура: Заголовок `Serializable` + поля класса + данные массива

`linkedlist_externalizable.dat` (`Externalizable`):

```
AC ED 00 05 73 72 00 25 66 75 6E 63 74 69 6F 6E
73 2E 4C 69 6E 6B 65 64 4C 69 73 74 54 61 62 75
6C 61 74 65 64 46 75 6E 63 74 69 6F 6E 00 00 00
00 00 00 00 02 0C 00 00 78 70 77 B4 00 00 00 0B
```

Структура: Размер списка (int) + пары координат (double, double)

7. Десериализация и проверка:

`ArrayTabulatedFunction` восстановлен, точек: 11

`LinkedListTabulatedFunction` восстановлен, точек: 11

8. Сравнение значений исходной и считанной функции (шаг 1):

x	Исходная	Serializable	Externalizable	Разница S	Разница E
0	0,000000	0,000000	0,000000	0,000000	0,000000
1	1,000000	1,000000	1,000000	0,000000	0,000000
2	2,000000	2,000000	2,000000	0,000000	0,000000
3	3,000000	3,000000	3,000000	0,000000	0,000000
4	4,000000	4,000000	4,000000	0,000000	0,000000
5	5,000000	5,000000	5,000000	0,000000	0,000000
6	6,000000	6,000000	6,000000	0,000000	0,000000
7	7,000000	7,000000	7,000000	0,000000	0,000000
8	8,000000	8,000000	8,000000	0,000000	0,000000
9	9,000000	9,000000	9,000000	0,000000	0,000000
10	10,000000	10,000000	10,000000	0,000000	0,000000

9. Проверка тождества  $\ln(e^x) = x$  после десериализации:

```
x= 0: аналит.=0,000000, табул.=0,000000, сериал.=0,000000, внеш.=0,000000
x= 1: аналит.=1,000000, табул.=1,000000, сериал.=1,000000, внеш.=1,000000
x= 2: аналит.=2,000000, табул.=2,000000, сериал.=2,000000, внеш.=2,000000
x= 3: аналит.=3,000000, табул.=3,000000, сериал.=3,000000, внеш.=3,000000
x= 4: аналит.=4,000000, табул.=4,000000, сериал.=4,000000, внеш.=4,000000
x= 5: аналит.=5,000000, табул.=5,000000, сериал.=5,000000, внеш.=5,000000
x= 6: аналит.=6,000000, табул.=6,000000, сериал.=6,000000, внеш.=6,000000
x= 7: аналит.=7,000000, табул.=7,000000, сериал.=7,000000, внеш.=7,000000
x= 8: аналит.=8,000000, табул.=8,000000, сериал.=8,000000, внеш.=8,000000
x= 9: аналит.=9,000000, табул.=9,000000, сериал.=9,000000, внеш.=9,000000
x=10: аналит.=10,000000, табул.=10,000000, сериал.=10,000000, внеш.=10,000000
```

---

## Проверка