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 №1

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

САМАРА 2025

Задание 1.
Добавил конструкторы в классах ArrayTabulatedFunction и LinkedListTabulatedFunction получающие сразу все точки функции в виде массива объектов типа FunctionPoint
ArrayTabulatedFunction

```
public ArrayTabulatedFunction(FunctionPoint[] points){
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не
менее 2");
    }
    for(int i = 1; i<points.length;i++){
        if(points[i-1].getX()>=points[i].getX()){
            throw new IllegalArgumentException("Точки в массиве не
упорядочены" +
                    " по значению абсциссы");
        }
    }
    pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount];
    for (int i = 0; i< pointsCount;i++){
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

LinkedListTabulatedFunction

```
public LinkedListTabulatedFunction(FunctionPoint[] points){
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не
менее 2");
    }
    for(int i = 1; i<points.length;i++){
        if(points[i-1].getX()>=points[i].getX()){
            throw new IllegalArgumentException("Точки в массиве не
упорядочены" +
                    " по значению абсциссы");
        }
    }
    initializeList();
    for (int i = 0; i<points.length;i++){
        addNodeToTail().point = new FunctionPoint(points[i]) ;
    }
}
```

Задание 2
Создал интерфейс 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.
Исключил соответствующие методы из интерфейса TabulatedFunction и сделал так, чтобы он расширял интерфейс Function

Function

```
package functions;

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

TabulatedFunction

```
package functions;
import java.io.*;
public interface TabulatedFunction extends Function,Serializable {
    public int getPointsCount();
    public void showPoints();
    public FunctionPoint getPoint(int index);
    public void setPoint(int index, FunctionPoint point) throws
InappropriateFunctionPointException;
    public double getPointX(int index);
    public void setPointX(int index, double x) throws
InappropriateFunctionPointException;
    public double getPointY(int index);
    public void setPointY(int index, double y);
    public void deletePoint(int index);
    public void addPoint(FunctionPoint point) throws
InappropriateFunctionPointException;
}
```

B

Задание 3

1. Создан пакет functions.basic
2. Созданы классы:
    - Exp - экспонента
    - Log - логарифм с произвольным основанием
    - TrigonometricFunction - абстрактный класс для тригонометрических функций
    - Sin, Cos, Tan - конкретные тригонометрические функции

Exp

```
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);
    }
}
```

Log

```
package functions.basic;


import functions.Function;

public class Log implements Function{
    private double base;
    final double EPSILON = 2.220446049250326E-16;
    public Log(double base){
        if(base<=0 || Math.abs(base - 1.0) < EPSILON){
            throw new IllegalArgumentException("Основание логарифма должно
быть" +
                    " положительным и не равным 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){
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        return Math.log(x)/Math.log(base);
    }
}
```

TrigonometricFunction

```
package functions.basic;


import functions.Function;

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

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

    @Override
    public abstract double  getFunctionValue(double x);
}
```

Sin

```
package functions.basic;

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

Cos

```
package functions.basic;

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

Tan

```
package functions.basic;

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

Задание 9

1. Создан пакет functions.meta
2. Реализованы классы для комбинирования функций:
    - Sum - сумма двух функций
    - Mult - произведение двух функций
    - Power - функция в степени
    - Scale - масштабирование функции
    - Shift - сдвиг функции
    - Composition - композиция функций

Sum

```
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 getRightDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }

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

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

Mult

```
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.min(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }

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

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        return f1.getFunctionValue(x)*f2.getFunctionValue(x);
    }
}
```

Power

```
package functions.meta;

import functions.Function;

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

    public Power(Function f, double power){
        this.f = f;
        this.power = power;
    }
    @Override
    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder();
    }

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

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        return Math.pow(f.getFunctionValue(x), power);
    }
}
```

Scale

```
package functions.meta;

import functions.Function;

public class Scale implements Function {
```

```
private Function baseFunction;
private double xScale;
private double yScale;
final double EPSILON = 2.220446049250326E-16;

public Scale(Function baseFunction, double xScale, double yScale){
    if (Math.abs(xScale) < EPSILON) {
        throw new IllegalArgumentException("Коэффициент масштабирования по X"
+
                " не может быть нулевым");
    }
    this.baseFunction = baseFunction;
    this.xScale = xScale;
    this.yScale = yScale;
}
    @Override
    public double getLeftDomainBorder() {
        if(xScale > 0){
        return baseFunction.getLeftDomainBorder()/xScale;
        }
        else{
        return baseFunction.getRightDomainBorder()/xScale;
        }
    }

    @Override
    public double getRightDomainBorder() {
        if(xScale > 0){
            return baseFunction.getRightDomainBorder()/xScale;
        }
        else{
            return baseFunction.getLeftDomainBorder()/xScale;
        }
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        double scaledX = x * xScale;
        double baseValue = baseFunction.getFunctionValue(scaledX);
        return baseValue * yScale;
    }
}
```

Shift

```
package functions.meta;

import functions.Function;

public class Shift implements Function {
    private Function baseFunction;
    private double xShift;
    private double yShift;
    public Shift(Function baseFunction, double xShift, double yShift){
        this.baseFunction =baseFunction;
        this.xShift = xShift;
        this.yShift = yShift;
    }
    @Override
    public double getLeftDomainBorder() {
        return baseFunction.getLeftDomainBorder()+xShift;
```

```
    }

    @Override
    public double getRightDomainBorder() {
        return baseFunction.getRightDomainBorder()+xShift;
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        double shiftedX = x+xShift;
        return baseFunction.getFunctionValue(shiftedX)+yShift;
    }
}
```

Composition

```
package functions.meta;

import functions.Function;

public class Composition implements Function {
    private Function outerFunction;
    private Function innerFunction;
    public Composition(Function outerFunction, Function innerFunction){
        this.outerFunction = outerFunction;
        this.innerFunction = innerFunction;
    }
    @Override
    public double getLeftDomainBorder() {
        return innerFunction.getLeftDomainBorder();
    }

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

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение x не входит в
область определения");
        }
        double innerValue = innerFunction.getFunctionValue(x);
        if (innerValue < outerFunction.getLeftDomainBorder() || innerValue >
outerFunction.getRightDomainBorder()) {
            throw new IllegalArgumentException("Значение  внутренней функции
не входит в область определения внешней");
        }
        return outerFunction.getFunctionValue(innerValue);
    }
}
```

Задание 5

Создан класс Functions со статическими методами для создания комбинаций функций

Functions

```
package functions;
import functions.meta.*;

public final class Functions {
    private Functions(){}
    public static Function shift(Function f, double shiftX, double shiftY){
        Shift shift = new Shift(f,shiftX,shiftY);
        return shift;
    }
    public static Function scale(Function f, double scaleX, double scaleY){
        Scale scale = new Scale(f,scaleX,scaleY);
        return scale;
    }
    public static Function power(Function f, double power){
        Power powerF = new Power(f,power);
        return powerF;
    }
    public static Function sum(Function f1, Function f2){
        Sum sum = new Sum(f1,f2);
        return sum;
    }
    public static Function mult(Function f1, Function f2){
        Mult mult = new Mult(f1,f2);
        return mult;
    }
    public static Function composition(Function f1, Function f2){
        Composition composition = new Composition(f1,f2);
        return composition;
    }
}
```

Задание 6

Создан класс TabulatedFunctions с методами для табулирования функций

TabulatedFunctions

```
package functions;

import java.io.*;

public class TabulatedFunctions {

    private TabulatedFunctions(){}
    public static TabulatedFunction tabulate(Function function, double leftX,
double rightX, int pointsCount){
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }

if(leftX<function.getLeftDomainBorder()||rightX>function.getRightDomainBorder
()){
            throw new IllegalArgumentException("Указанные границы для
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);
    }
    public static void outputTabulatedFunction(TabulatedFunction function,
OutputStream out)throws IOException {
        DataOutputStream dataOut = new DataOutputStream(out);
        try {
            dataOut.writeInt(function.getPointsCount());

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

            dataOut.flush();
        } finally {
        }
    }
    public static TabulatedFunction inputTabulatedFunction(InputStream
in)throws IOException {
        DataInputStream dataIn = new DataInputStream(in);
        try {
            int pointsCount = dataIn.readInt();

            if (pointsCount < 2) {
                throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
            }

            double[] xValues = new double[pointsCount];
            double[] yValues = new double[pointsCount];
            for (int i = 0; i < pointsCount; i++) {
                xValues[i] = dataIn.readDouble();
                yValues[i] = dataIn.readDouble();
            }

            for (int i = 1; i < pointsCount; i++) {
                if (xValues[i] <= xValues[i - 1]) {
                    throw new IOException("Некорректные данные: значения X
должны быть упорядочены по возрастанию");
                }
            }
            double leftX = xValues[0];
            double rightX = xValues[xValues.length - 1];
            return new ArrayTabulatedFunction(leftX,rightX,yValues);

        } catch (EOFException e) {
        throw new IOException("Неожиданный конец потока: данные неполные",
e);
        } catch (IllegalArgumentException e) {
        throw new IOException("Некорректные данные: " + e.getMessage(), e);
        } finally {

        }
    }
    public static void writeTabulatedFunction(TabulatedFunction function,
Writer out)throws IOException{
        BufferedWriter writer = new BufferedWriter(out);
        try {
```

```
            writer.write(String.valueOf(function.getPointsCount()));
            writer.write(" ");

            for (int i = 0; i < function.getPointsCount(); i++) {
                writer.write(String.valueOf(function.getPointX(i)));
                writer.write(" ");
                writer.write(String.valueOf(function.getPointY(i)));

                if (i < function.getPointsCount() - 1) {
                    writer.write(" ");
                }
            }

            writer.newLine();
            writer.flush();
        } finally {
        }
    }
    public static TabulatedFunction readTabulatedFunction(Reader in)throws
IOException{
        StreamTokenizer tokenizer = new StreamTokenizer(in);
        try {
            // Настраиваем токенизатор
            tokenizer.resetSyntax();
            tokenizer.wordChars('0', '9');
            tokenizer.wordChars('.', '.');
            tokenizer.wordChars('-', '-');
            tokenizer.wordChars('e', 'e');
            tokenizer.wordChars('E', 'E');
            tokenizer.whitespaceChars(' ', ' ');
            tokenizer.whitespaceChars('\t', '\t');
            tokenizer.whitespaceChars('\n', '\n');
            tokenizer.whitespaceChars('\r', '\r');


            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("Ожидалось количество точек (целое
число)");
            }

            int pointsCount;
            try {
                pointsCount = Integer.parseInt(tokenizer.sval);
            } catch (NumberFormatException e) {
                throw new IOException("Некорректное количество точек: " +
tokenizer.sval, e);
            }

            if (pointsCount < 2) {
                throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
            }


            double[] xValues = new double[pointsCount];
            double[] yValues = new double[pointsCount];

            for (int i = 0; i < pointsCount; i++) {

                if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                    throw new IOException("Ожидалось значение x для точки " +
i);
                }
```

```
                try {
                    xValues[i] = Double.parseDouble(tokenizer.sval);
                } catch (NumberFormatException e) {
                    throw new IOException("Некорректное значение x для точки
" + i + ": " + tokenizer.sval, e);
                }


                if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                    throw new IOException("Ожидалось значение y для точки " +
i);
                }

                try {
                    yValues[i] = Double.parseDouble(tokenizer.sval);
                } catch (NumberFormatException e) {
                    throw new IOException("Некорректное значение y для точки
" + i + ": " + tokenizer.sval, e);
                }
            }


            for (int i = 1; i < pointsCount; i++) {
                if (xValues[i] <= xValues[i-1]) {
                    throw new IOException("Некорректные данные: значения X
должны быть упорядочены по возрастанию");
                }
            }


            double leftX = xValues[0];
            double rightX = xValues[xValues.length - 1];
            return new ArrayTabulatedFunction(leftX,rightX,yValues);

        } finally {
        }
    }
}
```

Задание 6 и 7.
В пакете functions создан класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями:

- 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.
- Метод 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 public static TabulatedFunction inputTabulatedFunction(InputStream in) считывает из указанного потока данные о табулированной функции, создает и настраивает её объект и возвращает его из метода.
- Метод 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.
- Метоод public static TabulatedFunction readTabulatedFunction(Reader in) считывает из указанного потока данные о табулированной функции, создавает и настраивекь её объект и возвращает его из метода.

Код класса

```
package functions;

import java.io.*;
```

```
public class TabulatedFunctions {

    private TabulatedFunctions(){}
    public static TabulatedFunction tabulate(Function function, double leftX,
double rightX, int pointsCount){
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть
не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть
меньше правой");
        }

if(leftX<function.getLeftDomainBorder()||rightX>function.getRightDomainBorder
()){
            throw new IllegalArgumentException("Указанные границы для
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);
    }
    public static void outputTabulatedFunction(TabulatedFunction function,
OutputStream out)throws IOException {
        DataOutputStream dataOut = new DataOutputStream(out);
        try {
            dataOut.writeInt(function.getPointsCount());

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

            dataOut.flush();
        } finally {
        }
    }
    public static TabulatedFunction inputTabulatedFunction(InputStream
in)throws IOException {
        DataInputStream dataIn = new DataInputStream(in);
        try {
            int pointsCount = dataIn.readInt();

            if (pointsCount < 2) {
                throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
            }

            double[] xValues = new double[pointsCount];
            double[] yValues = new double[pointsCount];
            for (int i = 0; i < pointsCount; i++) {
                xValues[i] = dataIn.readDouble();
                yValues[i] = dataIn.readDouble();
            }

            for (int i = 1; i < pointsCount; i++) {
```

```
                if (xValues[i] <= xValues[i - 1]) {
                    throw new IOException("Некорректные данные: значения X
должны быть упорядочены по возрастанию");
                }
            }
            double leftX = xValues[0];
            double rightX = xValues[xValues.length - 1];
            return new ArrayTabulatedFunction(leftX,rightX,yValues);

        } catch (EOFException e) {
        throw new IOException("Неожиданный конец потока: данные неполные",
e);
        } catch (IllegalArgumentException e) {
        throw new IOException("Некорректные данные: " + e.getMessage(), e);
        } finally {

        }
    }
    public static void writeTabulatedFunction(TabulatedFunction function,
Writer out)throws IOException{
        BufferedWriter writer = new BufferedWriter(out);
        try {
            writer.write(String.valueOf(function.getPointsCount()));
            writer.write(" ");

            for (int i = 0; i < function.getPointsCount(); i++) {
                writer.write(String.valueOf(function.getPointX(i)));
                writer.write(" ");
                writer.write(String.valueOf(function.getPointY(i)));

                if (i < function.getPointsCount() - 1) {
                    writer.write(" ");
                }
            }

            writer.newLine();
            writer.flush();
        } finally {
        }
    }
    public static TabulatedFunction readTabulatedFunction(Reader in)throws
IOException{
        StreamTokenizer tokenizer = new StreamTokenizer(in);
        try {
            // Настраиваем токенизатор
            tokenizer.resetSyntax();
            tokenizer.wordChars('0', '9');
            tokenizer.wordChars('.', '.');
            tokenizer.wordChars('-', '-');
            tokenizer.wordChars('e', 'e');
            tokenizer.wordChars('E', 'E');
            tokenizer.whitespaceChars(' ', ' ');
            tokenizer.whitespaceChars('\t', '\t');
            tokenizer.whitespaceChars('\n', '\n');
            tokenizer.whitespaceChars('\r', '\r');


            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("Ожидалось количество точек (целое
число)");
            }

            int pointsCount;
            try {
```

```
                pointsCount = Integer.parseInt(tokenizer.sval);
            } catch (NumberFormatException e) {
                throw new IOException("Некорректное количество точек: " +
tokenizer.sval, e);
            }

            if (pointsCount < 2) {
                throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
            }


            double[] xValues = new double[pointsCount];
            double[] yValues = new double[pointsCount];

            for (int i = 0; i < pointsCount; i++) {

                if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                    throw new IOException("Ожидалось значение x для точки " +
i);
                }

                try {
                    xValues[i] = Double.parseDouble(tokenizer.sval);
                } catch (NumberFormatException e) {
                    throw new IOException("Некорректное значение x для точки
" + i + ": " + tokenizer.sval, e);
                }


                if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                    throw new IOException("Ожидалось значение y для точки " +
i);
                }

                try {
                    yValues[i] = Double.parseDouble(tokenizer.sval);
                } catch (NumberFormatException e) {
                    throw new IOException("Некорректное значение y для точки
" + i + ": " + tokenizer.sval, e);
                }
            }


            for (int i = 1; i < pointsCount; i++) {
                if (xValues[i] <= xValues[i-1]) {
                    throw new IOException("Некорректные данные: значения X
должны быть упорядочены по возрастанию");
                }
            }


            double leftX = xValues[0];
            double rightX = xValues[xValues.length - 1];
            return new ArrayTabulatedFunction(leftX,rightX,yValues);

        } finally {
        }
    }
}
```

Задание 8 и 9.

В TabulatedFunction добавлено наследование от интерфейса Serializable

В классы ArrayTabulatedFunction, LinkedListTabulatedFunction и FunctionPoint добавлена реализация интерфейса Serializable.

```
public class LinkedListTabulatedFunction implements TabulatedFunction {
    private static class FunctionNode implements Serializable {
        private static final long serialVersionUID = 1L;
        FunctionPoint point;
        transient FunctionNode prev;
        transient FunctionNode next;

        FunctionNode(FunctionPoint point) {
            this.point = point;
        }
    }
    private static final long serialVersionUID = 1L;
    private FunctionNode head;
    private int pointsCount;
    private transient FunctionNode lastAccessedNode;
    private transient int lastAccessedIndex;
    private static final double EPSILON = 2.220446049250326E-16;
```

Также во внутренний класс FunctionNode добавлена реализация интерфейса Serializable

Создан новый класс ArrayTabulatedFunctionExternalizable на основе ArrayTabulatedFunction реализующий интерфейс Externalizable

Новые методы интерфейса Externalizable.

```
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(pointsCount);
    for (int i = 0; i < pointsCount; i++) {
        out.writeDouble(points[i].getX());
        out.writeDouble(points[i].getY());
    }
}
public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
    pointsCount = in.readInt();
    if (pointsCount < 2) {
        throw new InvalidObjectException("pointsCount должно быть >= 2");
    }

    points = new FunctionPoint[pointsCount + 5];
    for (int i = 0; i < pointsCount; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        points[i] = new FunctionPoint(x, y);
    }
}
```

Код класса Main

```
import functions.*;

import functions.basic.*;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;

import static java.nio.file.Files.newBufferedWriter;

public class Main {
```

```
    public static void main(String[] args) throws Exception {
        try {
            // Часть 1: Создание объектов Sin и Cos
            System.out.println("1. СОЗДАНИЕ ОБЪЕКТОВ Sin И Cos\n");

            Function sin = new Sin();
            Function cos = new Cos();

            System.out.println("   Значения sin(x) и cos(x) на отрезке [0, π]
с шагом 0.1:");
            System.out.println("   x\t\t\tsin(x)\t\t\tcos(x)");
            System.out.println("   ----------------------------------------
------------------------");
            for (double x = 0; x <= Math.PI; x += 0.1) {
                double sinValue = sin.getFunctionValue(x);
                double cosValue = cos.getFunctionValue(x);
                System.out.printf("   %.6f\t\t%.10f\t\t%.10f%n", x, sinValue,
cosValue);
            }

            // Часть 2: Табулированные аналоги Sin и Cos
            System.out.println("\n\n2. ТАБУЛИРОВАННЫЕ АНАЛОГИ Sin И Cos (10
точек)\n");

            TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sin,
0, Math.PI, 10);
            TabulatedFunction tabulatedCos = TabulatedFunctions.tabulate(cos,
0, Math.PI, 10);

            System.out.println("   Сравнение точных и табулированных
значений:");
            System.out.println("
x\t\t\tsin(x)\t\t\ttabSin(x)\t\tcos(x)\t\t\ttabCos(x)");
            System.out.println("   ----------------------------------------
----------------------------------------------------");

            // Для вычисления погрешностей
            double maxSinError = 0;
            double maxCosError = 0;
            double sumSinError = 0;
            double sumCosError = 0;
            int count = 0;

            for (double x = 0; x <= Math.PI; x += 0.1) {
                double exactSin = sin.getFunctionValue(x);
                double tabSin = tabulatedSin.getFunctionValue(x);
                double exactCos = cos.getFunctionValue(x);
                double tabCos = tabulatedCos.getFunctionValue(x);

                double sinError = Math.abs(exactSin - tabSin);
                double cosError = Math.abs(exactCos - tabCos);

                maxSinError = Math.max(maxSinError, sinError);
                maxCosError = Math.max(maxCosError, cosError);
                sumSinError += sinError;
                sumCosError += cosError;
                count++;

                System.out.printf("
%.6f\t\t%.10f\t%.10f\t\t%.10f\t%.10f%n",
                        x, exactSin, tabSin, exactCos, tabCos);
            }

            double avgSinError = sumSinError / count;
```

```
            double avgCosError = sumCosError / count;

            System.out.println("\n   Статистика погрешностей (10 точек):");
            System.out.printf("   sin(x): средняя погрешность = %.10f,
максимальная = %.10f%n",
                    avgSinError, maxSinError);
            System.out.printf("   cos(x): средняя погрешность = %.10f,
максимальная = %.10f%n",
                    avgCosError, maxCosError);

            // Часть 3: Сумма квадратов табулированных функций
            System.out.println("\n\n3. СУММА КВАДРАТОВ ТАБУЛИРОВАННЫХ ФУНКЦИЙ
sin²(x) + cos²(x)\n");

            System.out.println("   Теория: sin²(x) + cos²(x) ≡ 1 для всех
x");
            System.out.println("   Проверка точности для разного количества
точек табуляции:");
            System.out.println("\n   Кол-во точек\tМакс. отклонение от
1\t\tСред. отклонение от 1");
            System.out.println("   ----------------------------------------
----------------------");

            int[] pointCounts = {5, 10, 20, 50, 100};
            for (int points : pointCounts) {
                // Создаем табулированные функции
                TabulatedFunction ts = TabulatedFunctions.tabulate(sin, 0,
Math.PI, points);
                TabulatedFunction tc = TabulatedFunctions.tabulate(cos, 0,
Math.PI, points);

                // Создаем функцию суммы квадратов
                Function sumOfSquares = Functions.sum(
                        Functions.power(ts, 2),
                        Functions.power(tc, 2)
                );

                double maxDeviation = 0;
                double sumDeviation = 0;
                int deviationCount = 0;

                for (double x = 0; x <= Math.PI; x += 0.1) {
                    double value = sumOfSquares.getFunctionValue(x);
                    if (!Double.isNaN(value)) {
                        double deviation = Math.abs(value - 1.0);
                        maxDeviation = Math.max(maxDeviation, deviation);
                        sumDeviation += deviation;
                        deviationCount++;
                    }
                }

                double avgDeviation = sumDeviation / deviationCount;
                System.out.printf("   %d\t\t%.10f\t\t\t%.10f%n",
                        points, maxDeviation, avgDeviation);
            }

            // Часть 4: Экспонента и работа с текстовыми файлами
            System.out.println("\n\n4. ЭКСПОНЕНТА: ТЕКСТОВЫЙ ФОРМАТ
ХРАНЕНИЯ\n");

            // Создаем табулированную экспоненту
            Function exp = new Exp();
            TabulatedFunction tabulatedExp = TabulatedFunctions.tabulate(exp,
0, 10, 11);
```

```
            System.out.println("   Создана табулированная экспонента exp(x)
на [0, 10] с 11 точками");

            // Сохраняем в текстовый файл
            Path textFile = Paths.get("textFile.txt");
            try (BufferedWriter writer = Files.newBufferedWriter(textFile,
StandardCharsets.UTF_8)) {
                TabulatedFunctions.writeTabulatedFunction(tabulatedExp,
writer);
                System.out.println("   Функция записана в текстовый файл: " +
textFile.toAbsolutePath());
            }

            // Читаем из текстового файла
            TabulatedFunction restoredExp;
            try (Reader reader = Files.newBufferedReader(textFile)) {
                restoredExp =
TabulatedFunctions.readTabulatedFunction(reader);
                System.out.println("   Функция прочитана из текстового
файла");
            }

            // Сравниваем значения
            System.out.println("\n   Сравнение исходной и восстановленной
экспоненты:");
            System.out.println("   x\t\tИсходная
exp(x)\t\tВосстановленная\t\tРазность");
            System.out.println("   ------------------------------------------
-----------------------");

            double maxExpDiff = 0;
            for (double x = 0; x <= 10; x += 1.0) {
                double original = tabulatedExp.getFunctionValue(x);
                double restored = restoredExp.getFunctionValue(x);
                double diff = Math.abs(original - restored);
                maxExpDiff = Math.max(maxExpDiff, diff);

                System.out.printf("   %.0f\t\t%.10f\t%.10f\t\t%.15f%n",
                        x, original, restored, diff);
            }
            System.out.printf("\n   Максимальная разность: %.15f%n",
maxExpDiff);

            // Анализ текстового файла
            System.out.println("\n   Анализ текстового файла:");
            String textContent = new String(Files.readAllBytes(textFile));
            System.out.println("   Содержимое файла: \"" + textContent.trim()
+ "\"");
            System.out.println("   Размер файла: " + Files.size(textFile) + "
байт");

            // Часть 5: Логарифм и работа с бинарными файлами
            System.out.println("\n\n5. ЛОГАРИФМ: БИНАРНЫЙ ФОРМАТ
ХРАНЕНИЯ\n");

            // Создаем табулированный логарифм (начинаем с 0.1, так как ln(0)
= -∞)
            Function ln = new Log(Math.E);
            TabulatedFunction tabulatedLn = TabulatedFunctions.tabulate(ln,
0.1, 10, 11);

            System.out.println("   Создан табулированный натуральный логарифм
ln(x) на [0.1, 10] с 11 точками");
```

```
            // Сохраняем в бинарный файл
            Path binaryFile = Paths.get("ln_function.bin");
            try (OutputStream out = Files.newOutputStream(binaryFile)) {
                TabulatedFunctions.outputTabulatedFunction(tabulatedLn, out);
                System.out.println("   Функция записана в бинарный файл: " +
binaryFile.toAbsolutePath());
            }

            // Читаем из бинарного файла
            TabulatedFunction restoredLn;
            try (InputStream in = Files.newInputStream(binaryFile)) {
                restoredLn = TabulatedFunctions.inputTabulatedFunction(in);
                System.out.println("   Функция прочитана из бинарного
файла");
            }

            // Сравниваем значения
            System.out.println("\n   Сравнение исходного и восстановленного
логарифма:");
            System.out.println("   x\t\tИсходный
ln(x)\t\tВосстановленный\t\tРазность");
            System.out.println("   ----------------------------------------
-----------------------");

            double maxLnDiff = 0;
            for (double x = 0.1; x <= 10; x += 1.0) {
                double original = tabulatedLn.getFunctionValue(x);
                double restored = restoredLn.getFunctionValue(x);
                double diff = Math.abs(original - restored);
                maxLnDiff = Math.max(maxLnDiff, diff);

                System.out.printf("   %.1f\t\t%.10f\t\t%.10f\t\t%.15f%n",
                        x, original, restored, diff);
            }
            System.out.printf("\n   Максимальная разность: %.15f%n",
maxLnDiff);

            // Анализ бинарного файла
            System.out.println("\n   Анализ бинарного файла:");
            System.out.println("   Размер файла: " + Files.size(binaryFile) +
" байт");

            // Выводим начало бинарного файла в hex
            System.out.println("   Первые 64 байта файла (hex):");
            try (InputStream in = Files.newInputStream(binaryFile)) {
                byte[] buffer = new byte[64];
                int bytesRead = in.read(buffer);

                for (int i = 0; i < bytesRead; i++) {
                    if (i % 16 == 0) {
                        System.out.printf("%n   %04X: ", i);
                    }
                    System.out.printf("%02X ", buffer[i]);
                    if (i % 8 == 7) {
                        System.out.print(" ");
                    }
                }
                System.out.println();

                // ASCII представление
                System.out.print("   ASCII: ");
                for (int i = 0; i < bytesRead; i++) {
                    char c = (char) buffer[i];
```

```
                    System.out.print(Character.isISOControl(c) || c > 127 ?
'.' : c);
                }
                System.out.println();
            }

            System.out.println("=== ТЕСТИРОВАНИЕ СЕРИАЛИЗАЦИИ ===\n");

            try {
                // Часть 1: Создание сложной функции
                System.out.println("1. СОЗДАНИЕ ТАБУЛИРОВАННОЙ ФУНКЦИИ\n");

                // Создаем функцию ln(exp(x)) = x (должно быть тождество)
                Function exp1 = new Exp();
                Function ln1 = new Log(Math.E);

                // Создаем композицию: ln(exp(x)) = x
                Function composition = Functions.composition(ln1, exp1);

                // Табулируем на отрезке [0, 10] с 11 точками
                TabulatedFunction tabulatedFunc =
TabulatedFunctions.tabulate(composition, 0, 10, 11);

                System.out.println("   Создана функция: ln(exp(x)) = x");
                System.out.println("   Количество точек: " +
tabulatedFunc.getPointsCount());
                System.out.println("   Область определения: [" +
                        tabulatedFunc.getLeftDomainBorder() + ", " +
                        tabulatedFunc.getRightDomainBorder() + "]");

                System.out.println("\n   Значения исходной функции:");
                System.out.println("   x\t\tf(x)");
                System.out.println("   -------------------");
                for (double x = 0; x <= 10; x += 1.0) {
                    System.out.printf("   %.1f\t\t%.6f%n", x,
tabulatedFunc.getFunctionValue(x));
                }

                // Часть 2: Сериализация с Serializable
                System.out.println("\n\n2. СЕРИАЛИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ
Serializable\n");

                Path serializableFile =
Paths.get("function_serializable.dat");

                // Сериализация
                try (ObjectOutputStream out = new ObjectOutputStream(
                        Files.newOutputStream(serializableFile))) {
                    out.writeObject(tabulatedFunc);
                    System.out.println("   Функция сериализована в файл: " +
                            serializableFile.toAbsolutePath());
                }

                // Десериализация
                TabulatedFunction deserializedFunc;
                try (ObjectInputStream in = new ObjectInputStream(
                        Files.newInputStream(serializableFile))) {
                    deserializedFunc = (TabulatedFunction) in.readObject();
                    System.out.println("   Функция десериализована из
файла");
                }

                // Сравнение
                System.out.println("\n   Сравнение исходной и
```

```
десериализованной функции:");
                System.out.println("
x\t\tИсходная\tДесериализ.\tРазность");
                System.out.println("   ----------------------------------------
---------------");

                double maxDiffSerializable = 0;
                for (double x = 0; x <= 10; x += 1.0) {
                    double original = tabulatedFunc.getFunctionValue(x);
                    double restored = deserializedFunc.getFunctionValue(x);
                    double diff = Math.abs(original - restored);
                    if (diff > maxDiffSerializable) maxDiffSerializable =
diff;

                    System.out.printf("   %.1f\t\t%.6f\t%.6f\t%.10f%n",
                            x, original, restored, diff);
                }
                System.out.printf("\n   Максимальная разность: %.10f%n",
maxDiffSerializable);

                // Информация о файле
                System.out.println("\n   Информация о файле Serializable:");
                System.out.println("   Размер файла: " +
Files.size(serializableFile) + " байт");

                // Часть 3: Создание и сериализация с Externalizable
                System.out.println("\n\n3. СЕРИАЛИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ
Externalizable\n");

                // Создаем функцию с Externalizable
                TabulatedFunction externalizableFunc =
                        new ArrayTabulatedFunctionExternalizable(0, 10, 11);

                // Заполняем значениями
                for (int i = 0; i < externalizableFunc.getPointsCount(); i++)
{
                    double x = externalizableFunc.getPointX(i);
                    externalizableFunc.setPointY(i,
composition.getFunctionValue(x));
                }

                Path externalizableFile =
Paths.get("function_externalizable.dat");

                // Сериализация
                try (ObjectOutputStream out = new ObjectOutputStream(
                        Files.newOutputStream(externalizableFile))) {
                    out.writeObject(externalizableFunc);
                    System.out.println("   Функция (Externalizable)
сериализована в файл: " +
                            externalizableFile.toAbsolutePath());
                }

                // Десериализация
                TabulatedFunction deserializedExternalizable;
                try (ObjectInputStream in = new ObjectInputStream(
                        Files.newInputStream(externalizableFile))) {
                    deserializedExternalizable = (TabulatedFunction)
in.readObject();
                    System.out.println("   Функция десериализована из
файла");
                }

                // Сравнение
```

```
                System.out.println("\n   Сравнение исходной и
десериализованной функции:");
                System.out.println("
x\t\tИсходная\tДесериализ.\tРазность");
                System.out.println("   ----------------------------------------
---------------");

                double maxDiffExternalizable = 0;
                for (double x = 0; x <= 10; x += 1.0) {
                    double original = externalizableFunc.getFunctionValue(x);
                    double restored =
deserializedExternalizable.getFunctionValue(x);
                    double diff = Math.abs(original - restored);
                    if (diff > maxDiffExternalizable) maxDiffExternalizable =
diff;

                    System.out.printf("   %.1f\t\t%.6f\t%.6f\t%.10f%n",
                            x, original, restored, diff);
                }
                System.out.printf("\n   Максимальная разность: %.10f%n",
maxDiffExternalizable);

                // Информация о файле
                System.out.println("\n   Информация о файле
Externalizable:");
                System.out.println("   Размер файла: " +
Files.size(externalizableFile) + " байт");


                // Часть 4: Очистка
                System.out.println("\n\n7. ОЧИСТКА ФАЙЛОВ\n");

                Files.deleteIfExists(serializableFile);
                Files.deleteIfExists(externalizableFile);
                Files.deleteIfExists(linkedListFile);
                System.out.println("   Временные файлы удалены");

                System.out.println("\n=== ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ===");

            } catch (Exception e) {
                System.out.println("Ошибка: " + e.getMessage());
                e.printStackTrace();
            }
        } finally {
        }
    }

        // Дополнительный тест для проверки версионности
    private static void testVersioning () {
            System.out.println("\n\nДополнительно: проверка версионности");

            try {
                // Создаем функцию
                TabulatedFunction func = TabulatedFunctions.tabulate(new
Sin(), 0, Math.PI, 5);

                // Сохраняем
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                try (ObjectOutputStream out = new ObjectOutputStream(baos)) {
                    out.writeObject(func);
                }

                byte[] data = baos.toByteArray();
                System.out.println("Размер сериализованных данных: " +
```

```
data.length + " байт");

                // Восстанавливаем
                ByteArrayInputStream bais = new ByteArrayInputStream(data);
                try (ObjectInputStream in = new ObjectInputStream(bais)) {
                    TabulatedFunction restored = (TabulatedFunction)
in.readObject();
                    System.out.println("Функция успешно восстановлена");
                    System.out.println("Класс: " +
restored.getClass().getName());
                }

            } catch (Exception e) {
                System.out.println("Ошибка версионности: " + e.getMessage());
            }
        }

}
```

Результат выполнения программы
1. СОЗДАНИЕ ОБЪЕКТОВ Sin И Cos

Значения sin(x) и cos(x) на отрезке [0, π] с шагом 0.1:

x	sin(x)	cos(x)
0,000000	0,0000000000	1,0000000000
0,100000	0,0998334166	0,9950041653
0,200000	0,1986693308	0,9800665778
0,300000	0,2955202067	0,9553364891
0,400000	0,3894183423	0,9210609940
0,500000	0,4794255386	0,8775825619
0,600000	0,5646424734	0,8253356149
0,700000	0,6442176872	0,7648421873
0,800000	0,7173560909	0,6967067093
0,900000	0,7833269096	0,6216099683
1,000000	0,8414709848	0,5403023059
1,100000	0,8912073601	0,4535961214
1,200000	0,9320390860	0,3623577545
1,300000	0,9635581854	0,2674988286
1,400000	0,9854497300	0,1699671429
1,500000	0,9974949866	0,0707372017
1,600000	0,9995736030	-0,0291995223
1,700000	0,9916648105	-0,1288444943
1,800000	0,9738476309	-0,2272020947
1,900000	0,9463000877	-0,3232895669
2,000000	0,9092974268	-0,4161468365
2,100000	0,8632093666	-0,5048461046
2,200000	0,8084964038	-0,5885011173
2,300000	0,7457052122	-0,6662760213
2,400000	0,6754631806	-0,7373937155
2,500000	0,5984721441	-0,8011436155
2,600000	0,5155013718	-0,8568887534
2,700000	0,4273798802	-0,9040721420
2,800000	0,3349881502	-0,9422223407
2,900000	0,2392493292	-0,9709581651
3,000000	0,1411200081	-0,9899924966
3,100000	0,0415806624	-0,9991351503

2. ТАБУЛИРОВАННЫЕ АНАЛОГИ Sin И Cos (10 точек)

Сравнение точных и табулированных значений:

x	sin(x)	tabSin(x)	cos(x)	tabCos(x)
0,000000	0,0000000000	0,0000000000	1,0000000000	1,0000000000
0,100000	0,0998334166	0,0979815536	0,9950041653	0,9827232085
0,200000	0,1986693308	0,1959631072	0,9800665778	0,9654464170
0,300000	0,2955202067	0,2939446608	0,9553364891	0,9481696255
0,400000	0,3894183423	0,3859068057	0,9210609940	0,9143546444
0,500000	0,4794255386	0,4720703379	0,8775825619	0,8646081059
0,600000	0,5646424734	0,5582338701	0,8253356149	0,8148615674
0,700000	0,6442176872	0,6439824415	0,7648421873	0,7646204980
0,800000	0,7173560909	0,7079353587	0,6967067093	0,6884043792
0,900000	0,7833269096	0,7718882758	0,6216099683	0,6121882604
1,000000	0,8414709848	0,8358411930	0,5403023059	0,5359721417
1,100000	0,8912073601	0,8839933571	0,4535961214	0,4506334539
1,200000	0,9320390860	0,9180219936	0,3623577545	0,3571405436
1,300000	0,9635581854	0,9520506300	0,2674988286	0,2636476334
1,400000	0,9854497300	0,9848077530	0,1699671429	0,1699305209
1,500000	0,9974949866	0,9848077530	0,0707372017	0,0704374439
1,600000	0,9995736030	0,9848077530	-0,0291995223	-0,0290556331
1,700000	0,9916648105	0,9848077530	-0,1288444943	-0,1285487101
1,800000	0,9738476309	0,9662040429	-0,2272020947	-0,2247614510
1,900000	0,9463000877	0,9321754065	-0,3232895669	-0,3182543613
2,000000	0,9092974268	0,8981467700	-0,4161468365	-0,4117472716
2,100000	0,8632093666	0,8624409083	-0,5048461046	-0,5042718354
2,200000	0,8084964038	0,7984879911	-0,5885011173	-0,5804879542
2,300000	0,7457052122	0,7345350740	-0,6662760213	-0,6567040730
2,400000	0,6754631806	0,6705821568	-0,7373937155	-0,7329201917
2,500000	0,5984721441	0,5940715695	-0,8011436155	-0,7941706620
2,600000	0,5155013718	0,5079080374	-0,8568887534	-0,8439172005
2,700000	0,4273798802	0,4217445052	-0,9040721420	-0,8936637390
2,800000	0,3349881502	0,3346977890	-0,9422223407	-0,9409837494
2,900000	0,2392493292	0,2367162354	-0,9709581651	-0,9582605409
3,000000	0,1411200081	0,1387346818	-0,9899924966	-0,9755373324
3,100000	0,0415806624	0,0407531282	-0,9991351503	-0,9928141240

Статистика погрешностей (10 точек):
sin(x): средняя погрешность = 0,0062886470, максимальная = 0,0147658500
cos(x): средняя погрешность = 0,0062149839, максимальная = 0,0146201609

3. СУММА КВАДРАТОВ ТАБУЛИРОВАННЫХ ФУНКЦИЙ $\sin^2(x) + \cos^2(x)$

Теория: $\sin^2(x) + \cos^2(x) \equiv 1$ для всех x
Проверка точности для разного количества точек табуляции:

Кол-во точек	Макс. отклонение от 1	Сред. отклонение от 1
5	0,1463959903	0,0959266430
10	0,0298331843	0,0195114252
20	0,0068171324	0,0044958833
50	0,0010263525	0,0006815110
100	0,0002515669	0,0001688266

4. ЭКСПОНЕНТА: ТЕКСТОВЫЙ ФОРМАТ ХРАНЕНИЯ

Создана табулированная экспонента exp(x) на [0, 10] с 11 точками
Функция записана в текстовый файл: C:\Users\MIXPC\IdeaProjects\laba 4\textFile.txt
Функция прочитана из текстового файла

Сравнение исходной и восстановленной экспоненты:

x	Исходная exp(x)	Восстановленная	Разность
0	1,0000000000	1,0000000000	0,000000000000000
1	2,7182818285	2,7182818285	0,000000000000000
2	7,3890560989	7,3890560989	0,000000000000000
3	20,0855369232	20,0855369232	0,000000000000000
4	54,5981500331	54,5981500331	0,000000000000000
5	148,4131591026	148,4131591026	0,000000000000000
6	403,4287934927	403,4287934927	0,000000000000000
7	1096,6331584285	1096,6331584285	0,000000000000000
8	2980,9579870417	2980,9579870417	0,000000000000000
9	8103,0839275754	8103,0839275754	0,000000000000000
10	22026,4657948067	22026,4657948067	0,000000000000000

Максимальная разность: 0,000000000000000

Анализ текстового файла:
Содержимое файла: "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 8103.083927575384 10.0 22026.465794806718"
Размер файла: 237 байт

5. ЛОГАРИФМ: БИНАРНЫЙ ФОРМАТ ХРАНЕНИЯ

Создан табулированный натуральный логарифм ln(x) на [0.1, 10] с 11 точками
Функция записана в бинарный файл: C:\Users\MIXPC\IdeaProjects\laba 4\ln_function.bin
Функция прочитана из бинарного файла

Сравнение исходного и восстановленного логарифма:

x	Исходный ln(x)	Восстановленный	Разность
0,1	-2,3025850930	-2,3025850930	0,000000000000000
1,1	0,0927048700	0,0927048700	0,000000000000000
2,1	0,7402327355	0,7402327355	0,000000000000000
3,1	1,1301474226	1,1301474226	0,000000000000000
4,1	1,4099993481	1,4099993481	0,000000000000000
5,1	1,6284294437	1,6284294437	0,000000000000000
6,1	1,8076029631	1,8076029631	0,000000000000000
7,1	1,9595024837	1,9595024837	0,000000000000000
8,1	2,0913442118	2,0913442118	0,000000000000000
9,1	2,2078123463	2,2078123463	0,000000000000000

Максимальная разность: 0,000000000000000

Анализ бинарного файла:
Размер файла: 180 байт
Первые 64 байта файла (hex):

```
  0000: 00 00 00 0B 3F B9 99 99  99 99 99 9A C0 02 6B B1
  0010: BB B5 55 15 3F F1 70 A3  D7 0A 3D 71 3F B6 0F BD
  0020: D2 FF FC 37 40 00 A3 D7  0A 3D 70 A4 3F E7 6F 8E
  0030: CB 04 70 55 40 08 8F 5C  28 F5 C2 8F 3F F1 F2 64
  ASCII: ....?.........k...U.?.p...=q?......7@....=p.?.o...pU@..\(...?..d
=== ТЕСТИРОВАНИЕ СЕРИАЛИЗАЦИИ ===

1. СОЗДАНИЕ ТАБУЛИРОВАННОЙ ФУНКЦИИ

  Создана функция: ln(exp(x)) = x
  Количество точек: 11
  Область определения: [0.0, 10.0]

  Значения исходной функции:
  x               f(x)
  ------------------
  0,0             0,000000
  1,0             1,000000
  2,0             2,000000
  3,0             3,000000
  4,0             4,000000
  5,0             5,000000
  6,0             6,000000
  7,0             7,000000
  8,0             8,000000
  9,0             9,000000
  10,0            10,000000

2. СЕРИАЛИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ Serializable

  Функция сериализована в файл: C:\Users\MIXPC\IdeaProjects\laba 4\function_serializable.dat
  Функция десериализована из файла

  Сравнение исходной и десериализованной функции:
  x               Исходная        Десериализ.     Разность
  ---------------------------------------------------
  0,0             0,000000        0,000000        0,0000000000
  1,0             1,000000        1,000000        0,0000000000
  2,0             2,000000        2,000000        0,0000000000
  3,0             3,000000        3,000000        0,0000000000
  4,0             4,000000        4,000000        0,0000000000
  5,0             5,000000        5,000000        0,0000000000
  6,0             6,000000        6,000000        0,0000000000
  7,0             7,000000        7,000000        0,0000000000
  8,0             8,000000        8,000000        0,0000000000
  9,0             9,000000        9,000000        0,0000000000
  10,0            10,000000       10,000000       0,0000000000

  Максимальная разность: 0,0000000000

  Информация о файле Serializable:
  Размер файла: 458 байт


3. СЕРИАЛИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ Externalizable
```

Функция (Externalizable) сериализована в файл: C:\Users\MIXPC\IdeaProjects\laba 4\function_externalizable.dat
Функция десериализована из файла

Сравнение исходной и десериализованной функции:

x	Исходная	Десериализ.	Разность
0,0	0,000000	0,000000	0,0000000000
1,0	1,000000	1,000000	0,0000000000
2,0	2,000000	2,000000	0,0000000000
3,0	3,000000	3,000000	0,0000000000
4,0	4,000000	4,000000	0,0000000000
5,0	5,000000	5,000000	0,0000000000
6,0	6,000000	6,000000	0,0000000000
7,0	7,000000	7,000000	0,0000000000
8,0	8,000000	8,000000	0,0000000000
9,0	9,000000	9,000000	0,0000000000
10,0	10,000000	10,000000	0,0000000000

Максимальная разность: 0,0000000000

Информация о файле Externalizable:
Размер файла: 250 байт

4. ОЧИСТКА ФАЙЛОВ

Временные файлы удалены

=== ТЕСТИРОВАНИЕ ЗАВЕРШЕНО ===