Лабораторная работа №4

Валиневич Владислав Александрович
группа: 6204-010302D

Цель 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.

Задание 1:

Добавим в классы ArrayTabulatedFunction и LinkedListTabulatedFunction конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint.

```
    // Конструктор 3:
    public LinkedListTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException("Кол-во точек меньше 2");
        }

        // Проверяем упорядоченность точек по X
        for (int i = 1; i < points.length; i++) {
            if (points[i].getX() <= points[i - 1].getX()) {
                throw new IllegalArgumentException("Точки не упорядочены по значению X");
            }
        }

        initializeList();


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

```
    }


    // Конструктор 3:
    public ArrayTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException("Кол-во точек меньше 2");
        }

        // Проверяем упорядоченность точек по X
        for (int i = 1; i < points.length; i++) {
            if (points[i].getX() <= points[i - 1].getX()) {
                throw new IllegalArgumentException("Точки не упорядочены по значению X");
            }
        }


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

Также не забываем выбрасывать исключения.

Задание 2:

В пакете functions создадим 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.

Интерфейс TabulatedFunction теперь расширяет Function

Табулированные функции стали частным случаем функций одной переменной.

ArrayTabulatedFunction.java | LinkedListTabulatedFunction.java | Function.java | TabulatedFunction.java

```java
package functions;

public interface TabulatedFunction extends Function {

    // Методы getLeftDomainBorder(), getRightDomainBorder() и getFunctionValue(double x)
    // теперь наследуются от интерфейса Function

    // Работа с точками
    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 FunctionPointIndexOutOfBoundsException, InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;

    String toString();
}
```

ArrayTabulatedFunction.java | LinkedListTabulatedFunction.java | Function.java | TabulatedFunction.java

```java
package functions;

public interface Function {

    // Возвращает значение левой границы области определения функции

    double getLeftDomainBorder();

    // Возвращает значение правой границы области определения функции

    double getRightDomainBorder();

    //Возвращает значение функции в заданной точке

    double getFunctionValue(double x);
}
```

Задание 3:

Создайте пакет functions.basic и в нем реализуем классы аналитических функций:

Exp.java – экспонента.

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {

    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

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

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

Log.java - логарифм с основанием.

```java
package functions.basic;

import functions.Function;

public class Log implements Function {
    private double base;

    public Log(double base) {
        if (base <= 0 || base == 1) {
            throw new IllegalArgumentException("Основание логарифмы отрицательное или равно одному");
        }
        this.base = base;
    }


    public double getLeftDomainBorder() {
        return 0;
    }


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


    public double getFunctionValue(double x) {
        if (x <= 0) {
            return Double.NaN; // Логарифм не определен для неположительных x
        }
        return Math.log(x) / Math.log(base); // Формула замены основания, log a  b = log c  b / log c  a
    }

    public double getBase() {
        return base;
    }
}
```

Sin.java, Cos.java, Tan.java - тригонометрические функции.

```java
package functions.basic;

public class Sin extends TrigonometricFunction {


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

```java
package functions.basic;

public class Cos extends TrigonometricFunction {

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

```java
package functions.basic;

public class Tan extends TrigonometricFunction {

    public double getFunctionValue(double x) {
        // Если косинус равен 0, то тангенс не определен
        double cosValue = Math.cos(x);
        if (Math.abs(cosValue) < 1e-10) {
            return Double.NaN;
        }
        return Math.tan(x);
    }
}
```

TrigonometricFunction.java - базовый класс для тригонометрических функций

```java
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {

    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

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

Задание 4:

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

Sum.java - сумма 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;
    }

    public double getLeftDomainBorder() {
        // Пересечение областей определения
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }


    public double getRightDomainBorder() {
        // Пересечение областей определения
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }


    public double getFunctionValue(double x) {
        // Проверяем, что x принадлежит пересечению областей определения
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

Scale.java – масштабирование

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


    public double getLeftDomainBorder() {
        if (scaleX > 0) {
            return f.getLeftDomainBorder() * scaleX;
        } else if (scaleX < 0) {
            return f.getRightDomainBorder() * scaleX;
        } else {
            return Double.NaN; // Масштаб 0 не имеет смысла
        }
    }


    public double getRightDomainBorder() {
        if (scaleX > 0) {
            return f.getRightDomainBorder() * scaleX;
        } else if (scaleX < 0) {
            return f.getLeftDomainBorder() * scaleX;
        } else {
            return Double.NaN; // Масштаб 0 не имеет смысла
        }
    }


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

Mult.java произведение двух функций

```java
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;
    }


    public double getLeftDomainBorder() {
        // Пересечение областей определения
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }


    public double getRightDomainBorder() {
        // Пересечение областей определения
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }


    public double getFunctionValue(double x) {
        // Проверяем, что x принадлежит пересечению областей определения
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
}
```

Power.java – возведение в степень

```java
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;
    }


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


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


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


}
```

Shift сдвиг вдоль осей координат

```java
package functions.meta;

import functions.Function;

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

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


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


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


    public double getFunctionValue(double x) {

        double shiftedX = x - shiftX;


        if (shiftedX < f.getLeftDomainBorder() || shiftedX > f.getRightDomainBorder()) {
            return Double.NaN;
        }


        return f.getFunctionValue(shiftedX) + shiftY;
    }
}
```

Composition.java представляет композицию двух функций - применение одной функции к результату другой

```java
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;
    }


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


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


    public double getFunctionValue(double x) {
        // Вычисляем значение первой функции
        double intermediate = f1.getFunctionValue(x);
        if (Double.isNaN(intermediate)) {
            return Double.NaN;
        }

        // Проверяем, что intermediate принадлежит области определения второй функции
        if (intermediate < f2.getLeftDomainBorder() || intermediate > f2.getRightDomainBorder()) {
            return Double.NaN;
        }


        return f2.getFunctionValue(intermediate);
    }
}
```

Задание 5:

В пакете functions создадим класс Functions, содержащий вспомогательные статические методы для работы с функциями.

```java
package functions;

import functions.meta.*;

public class Functions {

    // Приватный конструктор, чтобы нельзя было создать объект класса
    private Functions() {
    }

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

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

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

    //Сумма функций
    public static Function sum(Function f1, Function f2) {
        return new Sum(f1, f2);
    }

    //Произведение функции
    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);
    }
}
```

Также сделаем приватный конструктор чтобы нельзя было создать объект класса.

Задание 6:

В пакете functions создадим класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями.

```java
package functions;

public class TabulatedFunctions {

    // Приватный конструктор, чтобы нельзя было создать объект класса
    private TabulatedFunctions() {
    }

    //Табулирует функцию на заданном отрезке с заданным количеством точек
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        // Проверяем, что границы табулирования находятся в области определения функции
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табулирования выходят за область определения функции");
        }

        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }

        // Создаем массив значений Y
        double[] values = new double[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        // Заполняем массив значений, вычисляя функцию в каждой точке
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            values[i] = function.getFunctionValue(x);
        }

        // Возвращаем табулированную функцию
        return new ArrayTabulatedFunction(leftX, rightX, values);

    }
}
```

Задание 7:

Реализованы методы для работы с потоками:

```
package functions;

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

public class TabulatedFunctions {

    // Приватный конструктор, чтобы нельзя было создать объект класса
    private TabulatedFunctions() {
    }

    //Табулирует функцию на заданном отрезке с заданным количеством точек
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        // Проверяем, что границы табулирования находятся в области определения функции
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табулирования выходят за область определения функции");
        }

        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }

        // Создаем массив значений Y
        double[] values = new double[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        // Заполняем массив значений, вычисляя функцию в каждой точке
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            values[i] = function.getFunctionValue(x);
        }

        // Возвращаем табулированную функцию
        return new ArrayTabulatedFunction(leftX, rightX, values);

    }

    //Выводит табулированную функцию в байтовый поток
    public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) {
        try (DataOutputStream dataout = new DataOutputStream(out)) {

            dataout.writeInt(function.getPointsCount());

            for (int i = 0; i < function.getPointsCount(); i++) {
                dataout.writeDouble(function.getPointX(i));
                dataout.writeDouble(function.getPointY(i));
            }
            dataout.flush();
        } catch (IOException e) {

            throw new RuntimeException("Ошибка вывода табулированной функции", e);
        }
    }


    //Вводит табулированную функцию из байтового потока
    public static TabulatedFunction inputTabulatedFunction(InputStream in) {
        DataInputStream dataIn = new DataInputStream(in);
        try {
```

```
            int pointsCount = dataIn.readInt();


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



            return new ArrayTabulatedFunction(points);

        } catch (IOException e) {
            throw new RuntimeException("Ошибка ввода табулированной функции", e);
        }

    }
    //Записывает табулированную функцию в символьный поток
    public static void writeTabulatedFunction(TabulatedFunction function, Writer out) {
        PrintWriter writer = new PrintWriter(new BufferedWriter(out));
        try {
            // Записываем количество точек
            writer.print(function.getPointsCount());
            writer.print(' ');

            // Записываем координаты всех точек через пробел
            for (int i = 0; i < function.getPointsCount(); i++) {
                FunctionPoint point = function.getPoint(i);
                writer.print(point.getX());
                writer.print(' ');
                writer.print(point.getY());
                if (i < function.getPointsCount() - 1) {
                    writer.print(' ');
                }
            }


            writer.flush();

        } catch (Exception e) {
            throw new RuntimeException("Ошибка записи табулированной функции", e);
        }

    }

    public static TabulatedFunction readTabulatedFunction(Reader in) {
        try {
            StreamTokenizer tokenizer = new StreamTokenizer(in);
            tokenizer.nextToken();
            int pointsCount = (int) tokenizer.nval;

            FunctionPoint[] points = new FunctionPoint[pointsCount];

            for (int i = 0; i < pointsCount; i++) {
                tokenizer.nextToken();
                double x = tokenizer.nval;

                tokenizer.nextToken();
                double y = tokenizer.nval;

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

            return new ArrayTabulatedFunction(points);
        } catch (IOException e) {
            throw new RuntimeException("Ошибка при чтении из потока", e);
        }
```

Возникающие исключение IOException преобразуем в RuntimeException, IOException - это проверяемое исключение, которое должно быть либо обработано, либо объявить в сигнатуре метода, а RuntimeException - это непроверяемое исключение.

Задание 8

Задание 8:

Проверим работу написанных классов

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

import java.io.*;

public class main {
    public static void main(String[] args) {
        try {
            //Создание объектов Sin и Cos, вывод значений от 0 до π с шагом 0.1
            System.out.println("Исходные функции Sin и Cos");
            Sin sinFunc = new Sin();
            Cos cosFunc = new Cos();

            System.out.println("x\tSin(x)\tCos(x)");
            for (double x = 0; x <= Math.PI; x += 0.1) {
                System.out.printf("%.1f\t%.6f\t%.6f%n", x, sinFunc.getFunctionValue(x), cosFunc.getFunctionValue(x));
            }
            System.out.println();

            //Создание табулированных аналогов с 10 точками
            System.out.println("Табулированные функции (10 точек)");
            TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sinFunc, 0, Math.PI, 10);
            TabulatedFunction tabulatedCos = TabulatedFunctions.tabulate(cosFunc, 0, Math.PI, 10);

            System.out.println("x\tTabSin(x)\tTabCos(x)");
            for (double x = 0; x <= Math.PI; x += 0.1) {
                System.out.printf("%.1f\t%.6f\t%.6f%n", x, tabulatedSin.getFunctionValue(x), tabulatedCos.getFunctionValue(x));
            }
            System.out.println();

            //Сумма квадратов табулированных функций
            System.out.println("Сумма квадратов табулированных функций");
            Function sinSquared = Functions.power(tabulatedSin, 2);
            Function cosSquared = Functions.power(tabulatedCos, 2);
            Function sumOfSquares = Functions.sum(sinSquared, cosSquared);

            System.out.println("x\tSin2+Cos2");
            for (double x = 0; x <= Math.PI; x += 0.5) {
                System.out.println(x + "\t" + sumOfSquares.getFunctionValue(x));
            }
            System.out.println();

            //Работа с разным количеством точек
            System.out.println("Исследование с разным количеством точек");
            int[] pointCounts = {5, 10, 15, 20};
            for (int count : pointCounts) {
                TabulatedFunction sinTab = TabulatedFunctions.tabulate(sinFunc, 0, Math.PI, count);
                TabulatedFunction cosTab = TabulatedFunctions.tabulate(cosFunc, 0, Math.PI, count);
                Function sum = Functions.sum(Functions.power(sinTab, 2), Functions.power(cosTab, 2));
```

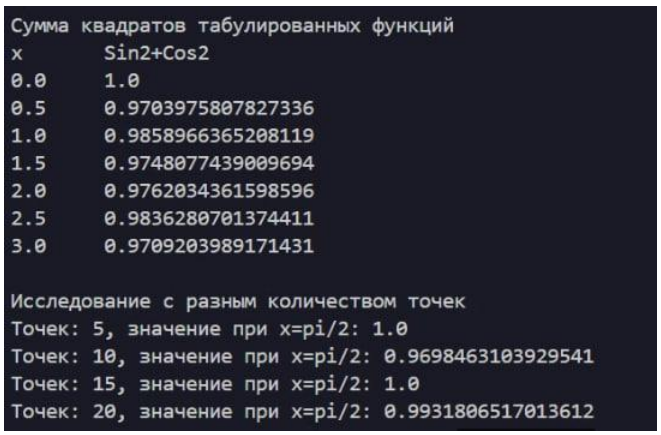
```
    TabulatedFunction cosTab = TabulatedFunctions.tabulate(cosFunc, 0, Math.PI, count);
    Function sum = Functions.sum(Functions.power(sinTab, 2), Functions.power(cosTab, 2));

    System.out.println("Точек: " + count + ", значение при x=pi/2: " + sum.getFunctionValue(Math.PI/2));
}
System.out.println();

//Работа с экспонентой - текстовый формат
System.out.println("Экспонента - текстовый формат");
Exp expFunc = new Exp();
TabulatedFunction tabulatedExp = TabulatedFunctions.tabulate(expFunc, 0, 10, 11);

// Запись в файл
try (FileWriter writer = new FileWriter("exp_function.txt")) {
    TabulatedFunctions.writeTabulatedFunction(tabulatedExp, writer);
}

// Чтение из файла
TabulatedFunction readExp;
try (FileReader reader = new FileReader("exp_function.txt")) {
    readExp = TabulatedFunctions.readTabulatedFunction(reader);
}

System.out.println("x\tИсходная\tСчитанная");
for (double x = 0; x <= 10; x += 1) {
    System.out.println(x + "\t" + tabulatedExp.getFunctionValue(x) + "\t" + readExp.getFunctionValue(x));
}
System.out.println();

//Работа с логарифмом - бинарный формат
System.out.println("Логарифм - бинарный формат");
Log logFunc = new Log(Math.E); // Натуральный логарифм
TabulatedFunction tabulatedLog = TabulatedFunctions.tabulate(logFunc, 0.1, 10, 11); // начинаем с 0.1, т.к. ln(0) не определен

// Запись в файл
try (FileOutputStream out = new FileOutputStream("log_function.bin")) {
    TabulatedFunctions.outputTabulatedFunction(tabulatedLog, out);
}

// Чтение из файла
TabulatedFunction readLog;
try (FileInputStream in = new FileInputStream("log_function.bin")) {
    readLog = TabulatedFunctions.inputTabulatedFunction(in);
}

System.out.println("x\tИсходная\tСчитанная");
for (double x = 0.1; x <= 10; x += 1) {
    System.out.println(x + "\t" + tabulatedLog.getFunctionValue(x) + "\t" + readLog.getFunctionValue(x));
}
System.out.println();

//Анализ файлов
System.out.println("Анализ форматов хранения");
```

```
//Анализ файлов
System.out.println("Анализ форматов хранения");
File textFile = new File("exp_function.txt");
File binaryFile = new File("log_function.bin");

System.out.println("Размер текстового файла: " + textFile.length() + " байт");
System.out.println("Размер бинарного файла: " + binaryFile.length() + " байт");
```

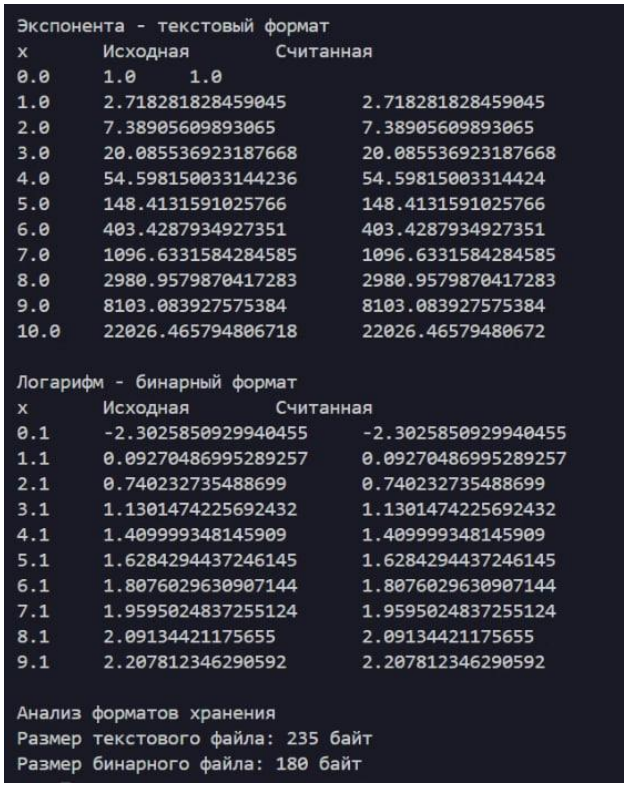
Вывод консоли:

```
Исходные функции Sin и Cos
x       Sin(x)  Cos(x)
0,0     0,000000        1,000000
0,1     0,099833        0,995004
0,2     0,198669        0,980067
0,3     0,295520        0,955336
0,4     0,389418        0,921061
0,5     0,479426        0,877583
0,6     0,564642        0,825336
0,7     0,644218        0,764842
0,8     0,717356        0,696707
0,9     0,783327        0,621610
1,0     0,841471        0,540302
```

```
Табулированные функции (10 точек)
x       TabSin(x)       TabCos(x)
0,0     0,000000        1,000000
0,1     0,097982        0,982723
0,2     0,195963        0,965446
0,3     0,293945        0,948170
0,4     0,385907        0,914355
0,5     0,472070        0,864608
0,6     0,558234        0,814862
0,7     0,643982        0,764620
0,8     0,707935        0,688404
0,9     0,771888        0,612188
1,0     0,835841        0,535972
```



```
Сумма квадратов табулированных функций
x       Sin2+Cos2
0.0     1.0
0.5     0.9703975807827336
1.0     0.9858966365208119
1.5     0.9748077439009694
2.0     0.9762034361598596
2.5     0.9836280701374411
3.0     0.9709203989171431

Исследование с разным количеством точек
Точек: 5, значение при x=pi/2: 1.0
Точек: 10, значение при x=pi/2: 0.9698463103929541
Точек: 15, значение при x=pi/2: 1.0
Точек: 20, значение при x=pi/2: 0.9931806517013612
```



```
Экспонента - текстовый формат
x       Исходная        Считанная
0.0     1.0     1.0
1.0     2.718281828459045       2.718281828459045
2.0     7.38905609893065        7.38905609893065
3.0     20.085536923187668      20.085536923187668
4.0     54.598150033144236      54.59815003314424
5.0     148.4131591025766       148.4131591025766
6.0     403.4287934927351       403.4287934927351
7.0     1096.6331584284585      1096.6331584284585
8.0     2980.9579870417283      2980.9579870417283
9.0     8103.083927575384       8103.083927575384
10.0    22026.465794806718      22026.46579480672

Логарифм - бинарный формат
x       Исходная        Считанная
0.1     -2.3025850929940455     -2.3025850929940455
1.1     0.09270486995289257     0.09270486995289257
2.1     0.740232735488699       0.740232735488699
3.1     1.1301474225692432      1.1301474225692432
4.1     1.409999348145909       1.409999348145909
5.1     1.6284294437246145      1.6284294437246145
6.1     1.8076029630907144      1.8076029630907144
7.1     1.9595024837255124      1.9595024837255124
8.1     2.09134421175655        2.09134421175655
9.1     2.207812346290592       2.207812346290592

Анализ форматов хранения
Размер текстового файла: 235 байт
Размер бинарного файла: 180 байт
```

При сохранении в текстовый формат, мы сможем в будущем открыть файл и спокойно прочитать все сохраненные числа, а также отредактировать его. Главными недостатками такого сохранения являются большой размер (в отличии от бинарного формата, 235 байт на 180 байт разница), медленная обработка из за парсинга строк, нет типизации.
Бинарный формат более компактный, высокая скорость работы засчет прямого чтения данных, безопаснее и сохраняет точность данных (Без потерь при сериализации/десериализации и сохраняет тип double)
Задание 9:

```
        // Тестирование сериализации
        System.out.println("=== Тестирование сериализации ===");

        // Создаем композицию функций: ln(exp(x)) = x
        Exp expFunc2 = new Exp();
        Log logFunc2 = new Log(Math.E);
        Function composition = Functions.composition(logFunc2, expFunc2); // ln(exp(x))

        // Создаем табулированную версию
        TabulatedFunction tabulatedComp = TabulatedFunctions.tabulate(composition, 0, 10, 11);

        System.out.println("Исходная функция ln(exp(x)) = x:");
        System.out.println("x\tИсходная");
        for (double x = 0; x <= 10; x += 1) {
            System.out.println(x + "\t" + tabulatedComp.getFunctionValue(x));
        }
        System.out.println();

        // Создаем массив точек из табулированной функции
        FunctionPoint[] points = new FunctionPoint[tabulatedComp.getPointsCount()];
        for (int i = 0; i < tabulatedComp.getPointsCount(); i++) {
            points[i] = tabulatedComp.getPoint(i);
        }

        // Тестируем сериализацию ArrayTabulatedFunction (Externalizable)
        System.out.println(" ArrayTabulatedFunction (Externalizable)");

        // Создаем ArrayTabulatedFunction через массив точек
        TabulatedFunction arrayFunc = new ArrayTabulatedFunction(points);

        // Сериализуем в файл
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("array_function.dat"))) {
            out.writeObject(arrayFunc);
            System.out.println("ArrayTabulatedFunction сериализован в array_function.dat");
        }

        // Десериализуем из файла
        TabulatedFunction deserializedArrayFunc;
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("array_function.dat"))) {
            deserializedArrayFunc = (TabulatedFunction) in.readObject();
            System.out.println("ArrayTabulatedFunction десериализован из array_function.dat");
        }

        System.out.println("x\tИсходная\tДесериализованная");
        for (double x = 0; x <= 10; x += 1) {
            System.out.println(x + "\t" + arrayFunc.getFunctionValue(x) + "\t" + deserializedArrayFunc.getFunctionValue(x));
        }
        System.out.println();

        // Тестируем сериализацию LinkedListTabulatedFunction (Serializable)
        System.out.println(" LinkedListTabulatedFunction (Serializable)");
```

```
        // Тестируем сериализацию LinkedListTabulatedFunction (Serializable)
        System.out.println(" LinkedListTabulatedFunction (Serializable)");

        // Создаем LinkedListTabulatedFunction через массив точек
        TabulatedFunction linkedListFunc = new LinkedListTabulatedFunction(points);

        // Сериализуем в файл
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("linkedlist_function.dat"))) {
            out.writeObject(linkedListFunc);
            System.out.println("LinkedListTabulatedFunction сериализован в linkedlist_function.dat");
        }

        // Десериализуем из файла
        TabulatedFunction deserializedLinkedListFunc;
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("linkedlist_function.dat"))) {
            deserializedLinkedListFunc = (TabulatedFunction) in.readObject();
            System.out.println("LinkedListTabulatedFunction десериализован из linkedlist_function.dat");
        }

        System.out.println("x\tИсходная\tДесериализованная");
        for (double x = 0; x <= 10; x += 1) {
            System.out.println(x + "\t" + linkedListFunc.getFunctionValue(x) + "\t" + deserializedLinkedListFunc.getFunctionValue(x));
        }
        System.out.println();

        // Анализ файлов сериализации
        System.out.println(" Анализ файлов сериализации ");
        File externalizableFile = new File("array_function.dat");
        File serializableFile = new File("linkedlist_function.dat");

        System.out.println("Размер файла Externalizable (Array): " + externalizableFile.length() + " байт");
        System.out.println("Размер файла Serializable (LinkedList): " + serializableFile.length() + " байт");
        System.out.println();


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

Допишем main, сериализуем массив ArrayTabulatedFunction с

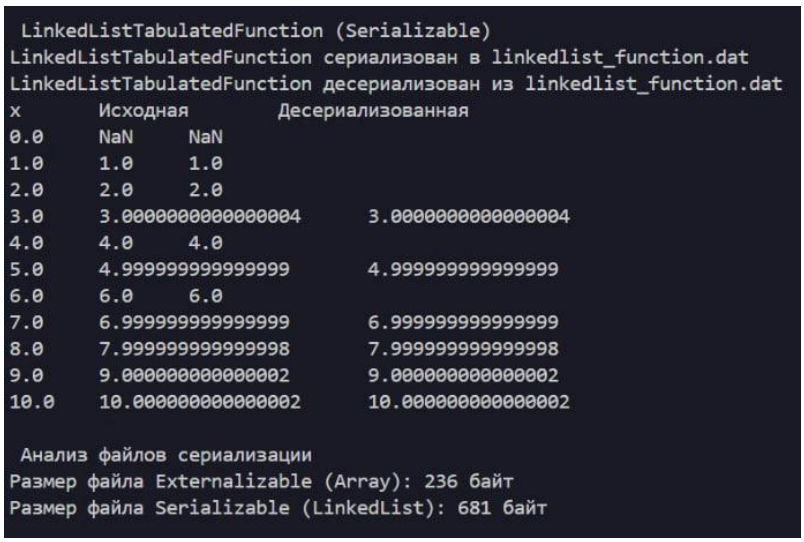
использованием интерфейса java.io.Externalizable, а список LinkedListTabulatedFunction с использованием интерфейса java.io.Serializable. Для этого:

```java
package functions;

import java.io.*;

public class ArrayTabulatedFunction  implements TabulatedFunction, Externalizable {
    private FunctionPoint[] points; //массив точек
    private static final double EPSILON = 1e-10;

    // Пустой конструктор необходим для Externalizable
    public ArrayTabulatedFunction() {
        this.points = new FunctionPoint[0];
    }
```

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

public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable {
    private static class FunctionNode implements Serializable {

        private FunctionPoint point;
        private FunctionNode prev;
        private FunctionNode next;

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

    private FunctionNode head;
```

Вывод консоли:

```
=== Тестирование сериализации ===
Исходная функция ln(exp(x)) = x:
x       Исходная
0.0     NaN
1.0     1.0
2.0     2.0
3.0     3.0000000000000004
4.0     4.0
5.0     4.999999999999999
6.0     6.0
7.0     6.999999999999999
8.0     7.999999999999998
9.0     9.000000000000002
10.0    10.000000000000002

 ArrayTabulatedFunction (Externalizable)
ArrayTabulatedFunction сериализован в array_function.dat
ArrayTabulatedFunction десериализован из array_function.dat
x       Исходная        Десериализованная
0.0     NaN     NaN
1.0     1.0     1.0
2.0     2.0     2.0
3.0     3.0000000000000004      3.0000000000000004
4.0     4.0     4.0
5.0     4.999999999999999       4.999999999999999
6.0     6.0     6.0
7.0     6.999999999999999       6.999999999999999
8.0     7.999999999999998       7.999999999999998
9.0     9.000000000000002       9.000000000000002
10.0    10.000000000000002      10.000000000000002
```



Путем протяженной работы, я понял, что:

Serializable- проще реализовать, автоматически обрабатывает циклические ссылки в двусвязном списке, графы объектов, наследование, сохраняет все поля, но имеет большой размер файла, низкую производительность и избыток данных, так как сохраняется все

Externalizable – минимальный размер файла, сохраняет только массив точек, сложная реализация.

Вывод: Я расширил возможности пакета для работы с функциями одной переменной добавив интерфейсы и классы для аналитически заданных функций, а также методы ввода и вывода табулированных функций.