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

Институт «Информатики и кибернетики»

Специальность «Фотоника и оптоинформатика»

Отчет по лабораторной работе № 4

Выполнила: Гамаюнова Д.И.,
группа 6201-120303D
Проверил: преподаватель Борисов Д.С.

Самара 2025

Задание 1.

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction добавила конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint.

```
public ArrayTabulatedFunction(FunctionPoint[] points) {  no usages
    if (points == null) {
        throw new IllegalArgumentException("Массив точек не может быть null");
    }
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2: " + points.length);
    }
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i] == null) {
            throw new IllegalArgumentException("Точка с индексом " + i + " равна null");
        }
        if (points[i + 1] == null) {
            throw new IllegalArgumentException("Точка с индексом " + (i + 1) + " равна null");
        }
        if (points[i].getX() >= points[i + 1].getX()) {
            throw new IllegalArgumentException("Точки не упорядочены по X: " +
                    points[i].getX() + " >= " + points[i + 1].getX());
        }
    }
    this.points = new FunctionPoint[points.length];
    this.size = points.length;
    for (int i = 0; i < points.length; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {  no usages
    if (points == null) {
        throw new IllegalArgumentException("Массив точек не может быть null");
    }
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2: " + points.length);
    }

    for (int i = 0; i < points.length - 1; i++) {
        if (points[i] == null) {
            throw new IllegalArgumentException("Точка с индексом " + i + " равна null");
        }
        if (points[i + 1] == null) {
            throw new IllegalArgumentException("Точка с индексом " + (i + 1) + " равна null");
        }
        if (points[i].getX() >= points[i + 1].getX()) {
            throw new IllegalArgumentException("Точки не упорядочены по X: " +
                    points[i].getX() + " >= " + points[i + 1].getX());
        }
    }
    initializeList();

    for (FunctionPoint point : points) {
        FunctionNode newNode = addNodeToTail();
        newNode.point = new FunctionPoint(point); // Защитная копия
    }
}
```

Задание 2.

В пакете functions создала 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.

```
package functions;

public interface Function {  1 usage  3 implementations
    double getLeftDomainBorder();  4 usages  2 implementations
    double getRightDomainBorder();  4 usages  2 implementations
    double getFunctionValue(double x);  4 usages  2 implementations
}
```

Также исключила соответствующие методы из интерфейса TabulatedFunction и сделала так, чтобы он расширял интерфейс Function. Теперь табулированные функции являются частным случаем функций одной переменной.

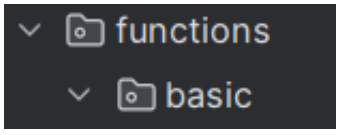
```
package functions;                                                                                    ⚠2

public interface TabulatedFunction extends Function {  10 usages  2 implementations

    int getPointsCount();  10 usages  2 implementations
    FunctionPoint getPoint(int index);  1 usage  2 implementations
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;  2 usages  2 implementations
    double getPointX(int index);  14 usages  2 implementations
    void setPointX(int index, double x) throws InappropriateFunctionPointException;  no usages  2 implementations
    double getPointY(int index);  10 usages  2 implementations
    void setPointY(int index, double y);  2 usages  2 implementations
    void deletePoint(int index);  4 usages  2 implementations
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  4 usages  2 implementations
}
```

Задание 3.

Создайте пакет functions.basic, в котором описала классы ряда функций, заданных аналитически.



В пакете создала публичный класс Exp, объекты которого вычисляют значение экспоненты. Класс реализовывает интерфейс Function.

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {  no usages

    public double getFunctionValue(double x) {  4 usages
        return Math.exp(x);
    }
    public double getLeftDomainBorder() {  4 usages
        return Double.NEGATIVE_INFINITY;
    }
    public double getRightDomainBorder() {  4 usages
        return Double.POSITIVE_INFINITY;
    }
}
```

Аналогично, создала класс Log, объекты которого вычисляют значение логарифма по заданному основанию и основание передается как параметр конструктора.

```java
package functions.basic;                                                                                    4 ^

import functions.Function;

public class Log implements Function {  no usages
    private double base;  3 usages

    public Log(double base) {  no usages
        if (base <= 0 || base == 1) {
            throw new IllegalArgumentException("Основание логарифма должно быть положительным и не равным 1: " + base);
        }
        this.base = base;
    }
    public double getFunctionValue(double x) {  4 usages
        if (x <= 0) {
            return Double.NaN; // Логарифм определен только для x > 0
        }
        return Math.log(x) / Math.log(base);
    }
    public double getLeftDomainBorder() {  4 usages
        return 0; // Фактически (0, +∞), но возвращаем 0 как формальную границу
    }
    public double getRightDomainBorder() {  4 usages
        return Double.POSITIVE_INFINITY;
    }
    public double getBase() {  no usages
        return base;
    }
}
```

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

```
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {  no usages  3 inheritors

    public double getLeftDomainBorder() {  4 usages
        return Double.NEGATIVE_INFINITY;
    }
    public double getRightDomainBorder() {  4 usages
        return Double.POSITIVE_INFINITY;
    }
    public abstract double getFunctionValue(double x);  4 usages  3 implementations
}
```

И создала наследующие от него публичные классы Sin, Cos и Tan.

```
package functions.basic;

public class Sin extends TrigonometricFunction {  no usages

    public double getFunctionValue(double x) {  4 usages
        return Math.sin(x);
    }
}
```

```
package functions.basic;

public class Cos extends TrigonometricFunction {  no usages

    public double getFunctionValue(double x) {  4 usages
        return Math.cos(x);
    }
}
```

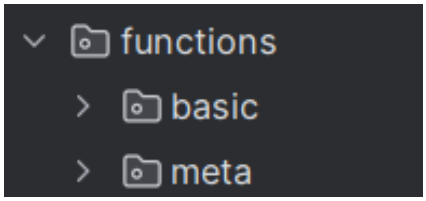
```
package functions.basic;

public class Tan extends TrigonometricFunction {  no usages

    public double getFunctionValue(double x) {  4 usages
        double cosValue = Math.cos(x);
        if (Math.abs(cosValue) < 1e-10) {
            return Double.NaN;
        }
        return Math.tan(x);
    }
}
```

Задание 4.

Создала пакет functions.meta, в котором описаны классы функций, позволяющие комбинировать функции.



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

```java
package functions.meta;

import functions.Function;

public class Sum implements Function {  no usages
    private Function f1;  5 usages
    private Function f2;  5 usages

    public Sum(Function f1, Function f2) {  no usages
        this.f1 = f1;
        this.f2 = f2;
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
    public double getLeftDomainBorder() {  7 usages
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    public double getRightDomainBorder() {  7 usages
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }
    public Function getFunction1() {  no usages
        return f1;
    }
    public Function getFunction2() {  no usages
        return f2;
    }
}
```

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

```java
package functions.meta;

import functions.Function;

public class Mult implements Function {  no usages
    private Function f1;  5 usages
    private Function f2;  5 usages

    public Mult(Function f1, Function f2) {  no usages
        this.f1 = f1;
        this.f2 = f2;
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }
    public Function getFunction1() {  no usages
        return f1;
    }
    public Function getFunction2() {  no usages
        return f2;
    }
}
```

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

```java
package functions.meta;

import functions.Function;

public class Power implements Function {  no usages
    private Function f;  5 usages
    private double power;  3 usages

    public Power(Function f, double power) {  no usages
        this.f = f;
        this.power = power;
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double base = f.getFunctionValue(x);
        return Math.pow(base, power);
    }
    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {
        return f.getRightDomainBorder();
    }
    public Function getFunction() {
        return f;
    }
    public double getPower() {  no usages
        return power;
    }
}
```

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

```java
package functions.meta;                                                    ⚠ 7

import functions.Function;

public class Scale implements Function {  no usages
    private Function f;  5 usages
    private double xScale;  5 usages
    private double yScale;  3 usages

    public Scale(Function f, double xScale, double yScale) {  no usages
        if (xScale == 0) {
            throw new IllegalArgumentException("Коэффициент масштабирования X не может быть 0");
        }
        this.f = f;
        this.xScale = xScale;
        this.yScale = yScale;
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double scaledX = x / xScale;
        return yScale * f.getFunctionValue(scaledX);
    }
    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder() * xScale;
    }
    public double getRightDomainBorder() {
        return f.getRightDomainBorder() * xScale;
    }
    public Function getFunction() {
        return f;
    }
    public double getXScale() {  no usages
        return xScale;
    }
    public double getYScale() {  no usages
        return yScale;
    }
}
```

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

```java
package functions.meta;

import functions.Function;

public class Shift implements Function {  no usages
    private Function f;  5 usages
    private double xShift;  5 usages
    private double yShift;  3 usages

    public Shift(Function f, double xShift, double yShift) {  no usages
        this.f = f;
        this.xShift = xShift;
        this.yShift = yShift;
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f.getFunctionValue( x: x - xShift) + yShift;
    }
    public double getLeftDomainBorder() {
        return f.getLeftDomainBorder() + xShift;
    }
    public double getRightDomainBorder() {
        return f.getRightDomainBorder() + xShift;
    }
    public Function getFunction() {
        return f;
    }
    public double getXShift() {  no usages
        return xShift;
    }
    public double getYShift() {  no usages
        return yShift;
    }
}
```

Также я создала класс Composition, объекты которого описывают композицию двух исходных функций.

```java
package functions.meta;

import functions.Function;

public class Composition implements Function {  no usages
    private Function outer;  3 usages
    private Function inner;  5 usages

    public Composition(Function outer, Function inner) {  no usages
        this.outer = outer;
        this.inner = inner;
    }
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double innerValue = inner.getFunctionValue(x);
        return outer.getFunctionValue(innerValue);
    }
    public double getLeftDomainBorder() {
        return inner.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {
        return inner.getRightDomainBorder();
    }
    public Function getOuterFunction() {  no usages
        return outer;
    }
    public Function getInnerFunction() {  no usages
        return inner;
    }
}
```

Задание 5.

В пакете functions создала класс Functions, содержащий вспомогательные статические методы для работы с функциями. И сделала так что, в программе вне этого класса нельзя создать его объект.

```java
package functions;

import functions.meta.*;

public final class Functions {
    private Functions() {
        throw new AssertionError( detailMessage: "Нельзя создать экземпляр класса Functions");
    }
    public static Function shift(Function f, double shiftX, double shiftY) {
        if (f == null) {
            throw new IllegalArgumentException("Исходная функция не может быть null");
        }
        return new Shift(f, shiftX, shiftY);
    }
    public static Function scale(Function f, double scaleX, double scaleY) {
        if (f == null) {
            throw new IllegalArgumentException("Исходная функция не может быть null");
        }
        if (scaleX == 0) {
            throw new IllegalArgumentException("Коэффициент масштабирования по X не может быть 0");
        }
        return new Scale(f, scaleX, scaleY);
    }
    public static Function power(Function f, double power) {
        if (f == null) {
            throw new IllegalArgumentException("Исходная функция не может быть null");
        }
        return new Power(f, power);
    }
    public static Function sum(Function f1, Function f2) {
        if (f1 == null || f2 == null) {
            throw new IllegalArgumentException("Функции не могут быть null");
        }
        return new Sum(f1, f2);
    }
    public static Function mult(Function f1, Function f2) {
        if (f1 == null || f2 == null) {
            throw new IllegalArgumentException("Функции не могут быть null");
        }
        return new Mult(f1, f2);
    }
    public static Function composition(Function f1, Function f2) {
        if (f1 == null || f2 == null) {
            throw new IllegalArgumentException("Функции не могут быть null");
        }
        return new Composition(f1, f2);
    }
    public static Function difference(Function f1, Function f2) {
        if (f1 == null || f2 == null) {
            throw new IllegalArgumentException("Функции не могут быть null");
        }
        return new Sum(f1, new Scale(f2, xScale: 1, yScale: -1));
    }
    public static Function division(Function f1, Function f2) {
        if (f1 == null || f2 == null) {
            throw new IllegalArgumentException("Функции не могут быть null");
        }
        return new Mult(f1, new Power(f2, power: -1));
    }
}
```

Задание 6.

В пакете functions создала класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями. Метод public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) - получает функцию и возвращает её табулированный аналог. Если указанные границы для табулирования выходят за область определения функции, метод выбрасывает исключение IllegalArgumentException.

```java
package functions;

public final class TabulatedFunctions {

    private TabulatedFunctions() {
        throw new AssertionError( detailMessage: "Нельзя создать экземпляр класса TabulatedFunctions");
    }
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        if (function == null) {
            throw new IllegalArgumentException("Функция не может быть null");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2: " + pointsCount);
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой: " + leftX + " >= " + rightX);
        }
        if (leftX < function.getLeftDomainBorder()) {
            throw new IllegalArgumentException("Левая граница табулирования выходит за область определения функции: "
                    + leftX + " < " + function.getLeftDomainBorder());
        }
        if (rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Правая граница табулирования выходит за область определения функции: "
                    + rightX + " > " + function.getRightDomainBorder());
        }
        TabulatedFunction tabulatedFunc = new ArrayTabulatedFunction(leftX, rightX, pointsCount);

        for (int i = 0; i < pointsCount; i++) {
            double x = tabulatedFunc.getPointX(i);
            double y = function.getFunctionValue(x);
            tabulatedFunc.setPointY(i, y);
        }
        return tabulatedFunc;
    }
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX) {
        int pointsCount = (int) Math.max(2, Math.ceil((rightX - leftX) * 10) + 1);
        return tabulate(function, leftX, rightX, pointsCount);
    }
    public static TabulatedFunction tabulate(Function function, int pointsCount) {
        if (function == null) {
            throw new IllegalArgumentException("Функция не может быть null");
        }

        double leftX = function.getLeftDomainBorder();
        double rightX = function.getRightDomainBorder();

        if (Double.isInfinite(leftX) || Double.isInfinite(rightX)) {
            throw new IllegalArgumentException("Нельзя табулировать функцию с бесконечной областью определения: ["
                    + leftX + ", " + rightX + "]");
        }
        return tabulate(function, leftX, rightX, pointsCount);
    }
}
```

Задание 7.

Добавила в класс TabulatedFunctions методы.

public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) - в указанный поток выводит значения, по которым потом можно будет восстановить табулированную функцию.

```
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
    if (function == null) {
        throw new IllegalArgumentException("Функция не может быть null");
    }
    if (out == null) {
        throw new IllegalArgumentException("Выходной поток не может быть null");
    }
    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();
}
```

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

```
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    if (in == null) {
        throw new IllegalArgumentException("Входной поток не может быть null");
    }
    DataInputStream dataIn = new DataInputStream(in);
    int pointsCount = dataIn.readInt();
    if (pointsCount < 2) {
        throw new IOException("Некорректное количество точек: " + pointsCount);
    }
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        double x = dataIn.readDouble();
        double y = dataIn.readDouble();
        points[i] = new FunctionPoint(x, y);
    }
    return new ArrayTabulatedFunction(points);
}
```

public static void writeTabulatedFunction(TabulatedFunction function, Writer out) -

```
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    if (function == null) {
        throw new IllegalArgumentException("Функция не может быть null");
    }
    if (out == null) {
        throw new IllegalArgumentException("Выходной поток не может быть null");
    }
    PrintWriter writer = new PrintWriter(out);
    writer.print(function.getPointsCount());
    writer.print(' ');
    for (int i = 0; i < function.getPointsCount(); i++) {
        writer.print(function.getPointX(i));
        writer.print(' ');
        writer.print(function.getPointY(i));
        if (i < function.getPointsCount() - 1) {
            writer.print(' ');
        }
    }
    writer.flush();
}
```

public static TabulatedFunction readTabulatedFunction(Reader in) – считывает из указанного потока данные о табулированной функции, создает и настраивать её объект и возвращает его из метода.

```
public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
    if (in == null) {
        throw new IllegalArgumentException("Входной поток не может быть null");
    }
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
        throw new IOException("Ожидалось количество точек");
    }
    int pointsCount = (int) tokenizer.nval;
    if (pointsCount < 2) {
        throw new IOException("Некорректное количество точек: " + pointsCount);
    }
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
            throw new IOException("Ожидалась координата x для точки " + i);
        }
        double x = tokenizer.nval;
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
            throw new IOException("Ожидалась координата y для точки " + i);
        }
        double y = tokenizer.nval;
        points[i] = new FunctionPoint(x, y);
    }
    return new ArrayTabulatedFunction(points);
}
```

Возникающее исключение IOException следует пробросить выше, а не поглотить его, тк вызывающий код не узнает об ошибке и программа продолжит работать с некорректными данными, и ошибка будет обнаружена гораздо позже, из-за чего ее будет сложнее отладить. Я думаю, потоки закрывать не стоит, потому что они могут дальше использоваться вызывающим кодом.

Задание 8.

Создала по одному объекту классов Sin и Cos, вывела в консоль значения этих функций на отрезке от 0 до π с шагом 0,1.

```
private static void testAnalyticFunctions() {  1 usage
    System.out.println("1. Аналитические функции:");
    Function sin = new Sin();
    Function cos = new Cos();

    System.out.println("\n3Значения sin(x) от 0 до π с шагом 0.1:");
    printFunctionValues(sin,  from: 0, Math.PI,  step: 0.1);
    System.out.println("\n3Значения cos(x) от 0 до π с шагом 0.1:");
    printFunctionValues(cos,  from: 0, Math.PI,  step: 0.1);
```

С помощью метода TabulatedFunctions.tabulate() создала табулированные аналоги этих функций на отрезке от 0 до π с 10 точками. Вывела в консоль значения этих функций на отрезке от 0 до π с шагом 0,1 и сравнила со значениями исходных функций.

```
System.out.println("\n\n2. Табулированные аналоги функции :");

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

System.out.println("\nТабулированный sin(x):");
printTabulatedFunction(tabulatedSin);
System.out.println("\nТабулированный cos(x):");
printTabulatedFunction(tabulatedCos);

System.out.println("\nСравнение аналитического и табулированного sin(x):");
compareFunctions(sin, tabulatedSin,  from: 0, Math.PI,  step: 0.1);
```

С помощью методов класса Functions создала объект функции, являющейся суммой квадратов табулированных аналогов синуса и косинуса. Вывела в консоль значения этой функций на отрезке от 0 до π с шагом 0,1. Изменяла количество точек в табулированных аналогах и исследовала, как при этом изменяется результирующая функция.

```
System.out.println("\n\n3. Сумма квадратов:");

Function sinSquared = Functions.power(tabulatedSin, power: 2);
Function cosSquared = Functions.power(tabulatedCos, power: 2);
Function sumOfSquares = Functions.sum(sinSquared, cosSquared);

System.out.println("Значения суммы квадратов с 10 точками табуляции:");
printFunctionValues(sumOfSquares, from: 0, Math.PI, step: 0.1);

System.out.println("\n\nВлияние количества точек:");
for (int points : new int[]{5, 10, 20, 50}) {
    System.out.println("\nКоличество точек: " + points);
    TabulatedFunction sinTab = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, points);
    TabulatedFunction cosTab = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, points);

    Function sin2 = Functions.power(sinTab, power: 2);
    Function cos2 = Functions.power(cosTab, power: 2);
    Function sum = Functions.sum(sin2, cos2);
    double summ = 0;
    int count = 0;
    for (double x=0; x < Math.PI; x += 0.05) {
        summ += sum.getFunctionValue(x);
        count += 1;
    }
    System.out.printf("Средняя погрешность суммы квадратов: %.5f", 1-summ/count );
}
```

С помощью метода TabulatedFunctions.tabulate() создала табулированный аналог экспоненты на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.writeTabulatedFunction() вывела его в файл. Далее с помощью метода TabulatedFunctions.readTabulatedFunction() считала табулированную функцию из этого файла. Вывела и сравнила значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.

```
System.out.println("\n\n4. Экспонента и файлы (символы):");
try {

    Function exp = new Exp();
    TabulatedFunction tabulatedExp = TabulatedFunctions.tabulate(exp, leftX: 0, rightX: 10, pointsCount: 11);

    System.out.println("Исходная табулированная exp(x):");
    printTabulatedFunction(tabulatedExp);

    String textFileName = "exp_function.txt";
    try (FileWriter writer = new FileWriter(textFileName)) {
        TabulatedFunctions.writeTabulatedFunction(tabulatedExp, writer);
    }
    System.out.println("Функция записана в файл: " + textFileName);

    TabulatedFunction readExp;
    try (FileReader reader = new FileReader(textFileName)) {
        readExp = TabulatedFunctions.readTabulatedFunction(reader);
    }

    System.out.println("\nСчитанная из файла exp(x):");
    printTabulatedFunction(readExp);

    System.out.println("\nСравнение исходной и считанной функций:");
    compareTabulatedFunctions(tabulatedExp, readExp, from: 0, to: 10, step: 1);

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

С помощью метода TabulatedFunctions.tabulate() создала табулированный аналог логарифма по натуральному основанию на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.outputTabulatedFunction() вывела его в файл. Далее с помощью метода TabulatedFunctions.inputTabulatedFunction() считала табулированную функцию из этого файла. Вывела и сравнила значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.

```
System.out.println("\n\n5. Логарифм и файлы (байты):");
try {

    Function ln = new Log(Math.E);
    TabulatedFunction tabulatedLn = TabulatedFunctions.tabulate(ln, leftX: 1, rightX: 10, pointsCount: 11);

    System.out.println("Исходная табулированная ln(x) от 1 до 10:");
    printTabulatedFunction(tabulatedLn);

    String binaryFileName = "ln_function.dat";
    try (FileOutputStream fos = new FileOutputStream(binaryFileName)) {
        TabulatedFunctions.outputTabulatedFunction(tabulatedLn, fos);
    }
    System.out.println("Функция записана в файл: " + binaryFileName);

    TabulatedFunction readLn;
    try (FileInputStream fis = new FileInputStream(binaryFileName)) {
        readLn = TabulatedFunctions.inputTabulatedFunction(fis);
    }

    System.out.println("\nСчитанная из файла ln(x):");
    printTabulatedFunction(readLn);

    System.out.println("\nСравнение исходной и считанной функций:");
    compareTabulatedFunctions(tabulatedLn, readLn, from: 1, to: 10, step: 1);
}
catch (IOException e) {
    System.out.println("Ошибка: " + e.getMessage());
}
```

Текстовый формат удобен для отладки, а бинарный формат эффективен для хранения.

Задание 9.

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

public class ArrayTabulatedFunction implements TabulatedFunction, Serializable

```
package functions;

import java.io.*;

public class ArrayTabulatedFunction implements TabulatedFunction, Serializable {
    private static final long serialVersionUID = 1L;  no usages
    private FunctionPoint[] points;  33 usages
    private int size;  26 usages
```

public class FunctionPoint implements Externalizable

```
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeDouble(x);
    out.writeDouble(y);
}

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    x = in.readDouble();
    y = in.readDouble();
}
```

public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable

```
package functions;

import java.io.*;

public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {
    private static class FunctionNode implements Serializable {  38 usages
        private static final long serialVersionUID = 1L;  no usages
```

```
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {  10 usages
    private static class FunctionNode implements Serializable {  38 usages
        private static final long serialVersionUID = 1L;  no usages
        FunctionPoint point;  30 usages
        FunctionNode prev;  17 usages
        FunctionNode next;  21 usages
        public FunctionNode() {  3 usages
            this.point = null;
            this.prev = this;
            this.next = this;
        }
        public FunctionNode(FunctionPoint point) {  no usages
            this.point = point;
            this.prev = this;
            this.next = this;
        }
        public FunctionNode(FunctionPoint point, FunctionNode prev, FunctionNode next) {  no usages
            this.point = point;
            this.prev = prev;
            this.next = next;
        }

    }
    private FunctionNode head;  13 usages
    private int size;  25 usages
    private FunctionNode lastAccessedNode;  3 usages
    private int lastAccessedIndex;  8 usages
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {  5 usages
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница области определения больше или равна правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек меньше двух");
        }

        initializeList();
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            FunctionNode newNode = addNodeToTail();
            newNode.point = new FunctionPoint(x,  y: 0.0);
```

```java
public class Main {
    public static void main(String[] args) throws Exception {
        System.out.println("Тестирование TabuletedFunctions\n");
        testTabulatedFunctions();
        System.out.println("\n\nТестирование аналитических функций и методов TabuletedFunctions\n");
        testAnalyticFunctions();
        System.out.println("\nПроверка исключений через интерфейс\n");
        testExceptionsWithInterface();
        System.out.println("\nТестирование сериализации\n");
        System.out.println(">>> Serializable (ArrayTabulatedFunction)");
        ArrayTabulatedFunction tabSer = testTaskSerializable();

        System.out.println(">>> Externalizable (LinkedListTabulatedFunction)");
        LinkedListTabulatedFunction tabExt = testTaskExternalizable();
        System.out.println("\nСравнение способов хранения");

        int points = Math.min(tabSer.getPointsCount(), tabExt.getPointsCount());

        System.out.println("   x\t\t\tSerializable\t\t\tExternalizable");
        for (int i = 0; i < points; i++) {

            double vSer = tabSer.getPointY(i);
            double vExt =tabExt.getPointY(i);

            System.out.printf("   %.1f\t\t%.6f\t\t\t%.6f %n", tabSer.getPointX(i), vSer, vExt);
        }

        System.out.println("\n   Размер файлов:");
        File fSer = new File( pathname: "ser.bin");
        File fExt = new File( pathname: "ext.bin");
        System.out.println("      Serializable (ser.bin): " + fSer.length() + " байт");
        System.out.println("      Externalizable (ext.bin): " + fExt.length() + " байт");
    }
```

```java
public static ArrayTabulatedFunction testTaskSerializable() throws Exception {  1 usage
    System.out.println("\nСериализация через Serializable:");

    ArrayTabulatedFunction tab = (ArrayTabulatedFunction)
            TabulatedFunctions.tabulate(new Composition(new Log(Math.E), new Exp()),  leftX: 0,  rightX: 10,  pointsCount: 11);

    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream( name: "ser.bin"))) {
        out.writeObject(tab);
    }

    ArrayTabulatedFunction tab2;
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream( name: "ser.bin"))) {
        tab2 = (ArrayTabulatedFunction) in.readObject();
    }

    System.out.println("   Serializable успешно: первая точка: (" + tab2.getPointX( index: 0)+ ", " + tab2.getPointY( index: 0) + ")");

    return tab2; // возвращаем для сравнения
}


public static LinkedListTabulatedFunction testTaskExternalizable() throws Exception {  1 usage
    System.out.println("\nСериализация через Externalizable:");

    LinkedListTabulatedFunction tab = new LinkedListTabulatedFunction (  leftX: 0,  rightX: 10,  pointsCount: 11);

    for (int i = 0; i < tab.getPointsCount(); i++){
        tab.setPointY(i,new Composition(new Log(Math.E), new Exp()).getFunctionValue(tab.getPointX(i)));
    }

    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream( name: "ext.bin"))) {
        tab.writeExternal(out);
    }

    LinkedListTabulatedFunction tab2 = new LinkedListTabulatedFunction();
    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream( name: "ext.bin"))) {
        tab2.readExternal(in);
    }
```

Результаты

```
Тестирование аналитических функций и методов TabuletedFunctions

1. Аналитические функции:

Значения sin(x) от 0 до π с шагом 0.1:
f(0,0) = 0,0000    f(0,1) = 0,0998    f(0,2) = 0,1987    f(0,3) = 0,2955
f(0,4) = 0,3894    f(0,5) = 0,4794    f(0,6) = 0,5646    f(0,7) = 0,6442
f(0,8) = 0,7174    f(0,9) = 0,7833    f(1,0) = 0,8415    f(1,1) = 0,8912
f(1,2) = 0,9320    f(1,3) = 0,9636    f(1,4) = 0,9854    f(1,5) = 0,9975
f(1,6) = 0,9996    f(1,7) = 0,9917    f(1,8) = 0,9738    f(1,9) = 0,9463
f(2,0) = 0,9093    f(2,1) = 0,8632    f(2,2) = 0,8085    f(2,3) = 0,7457
f(2,4) = 0,6755    f(2,5) = 0,5985    f(2,6) = 0,5155    f(2,7) = 0,4274
f(2,8) = 0,3350    f(2,9) = 0,2392    f(3,0) = 0,1411    f(3,1) = 0,0416

Значения cos(x) от 0 до π с шагом 0.1:
f(0,0) = 1,0000    f(0,1) = 0,9950    f(0,2) = 0,9801    f(0,3) = 0,9553
f(0,4) = 0,9211    f(0,5) = 0,8776    f(0,6) = 0,8253    f(0,7) = 0,7648
f(0,8) = 0,6967    f(0,9) = 0,6216    f(1,0) = 0,5403    f(1,1) = 0,4536
f(1,2) = 0,3624    f(1,3) = 0,2675    f(1,4) = 0,1700    f(1,5) = 0,0707
f(1,6) = -0,0292   f(1,7) = -0,1288   f(1,8) = -0,2272   f(1,9) = -0,3233
f(2,0) = -0,4161   f(2,1) = -0,5048   f(2,2) = -0,5885   f(2,3) = -0,6663
f(2,4) = -0,7374   f(2,5) = -0,8011   f(2,6) = -0,8569   f(2,7) = -0,9041
f(2,8) = -0,9422   f(2,9) = -0,9710   f(3,0) = -0,9900   f(3,1) = -0,9991
```

```
2. Табулированные аналоги функции

Табулированный sin(x):
Количество точек: 10
Область определения: [0,00, 3,14]
Точки функции:
   0: x =  0,0000, y =   0,000000
   1: x =  0,3491, y =   0,342020
   2: x =  0,6981, y =   0,642788
   3: x =  1,0472, y =   0,866025
   4: x =  1,3963, y =   0,984808
   5: x =  1,7453, y =   0,984808
   6: x =  2,0944, y =   0,866025
   7: x =  2,4435, y =   0,642788
   8: x =  2,7925, y =   0,342020
   9: x =  3,1416, y =   0,000000

Табулированный cos(x):
Количество точек: 10
Область определения: [0,00, 3,14]
Точки функции:
   0: x =  0,0000, y =   1,000000
   1: x =  0,3491, y =   0,939693
   2: x =  0,6981, y =   0,766044
   3: x =  1,0472, y =   0,500000
   4: x =  1,3963, y =   0,173648
   5: x =  1,7453, y =  -0,173648
   6: x =  2,0944, y =  -0,500000
   7: x =  2,4435, y =  -0,766044
   8: x =  2,7925, y =  -0,939693
   9: x =  3,1416, y =  -1,000000
```

```
Сравнение аналитического и табулированного sin(x):
x       Аналитическая   Табулированная  Погрешность
0,0     0,000000        0,000000        0,000000
0,1     0,099833        0,097982        0,001852
0,2     0,198669        0,195963        0,002706
0,3     0,295520        0,293945        0,001576
0,4     0,389418        0,385907        0,003512
0,5     0,479426        0,472070        0,007355
0,6     0,564642        0,558234        0,006409
0,7     0,644218        0,643982        0,000235
0,8     0,717356        0,707935        0,009421
0,9     0,783327        0,771888        0,011439
1,0     0,841471        0,835841        0,005630
1,1     0,891207        0,883993        0,007214
1,2     0,932039        0,918022        0,014017
1,3     0,963558        0,952051        0,011508
1,4     0,985450        0,984808        0,000642
1,5     0,997495        0,984808        0,012687
1,6     0,999574        0,984808        0,014766
1,7     0,991665        0,984808        0,006857
1,8     0,973848        0,966204        0,007644
1,9     0,946300        0,932175        0,014125
2,0     0,909297        0,898147        0,011151
2,1     0,863209        0,862441        0,000768
2,2     0,808496        0,798488        0,010008
2,3     0,745705        0,734535        0,011170
2,4     0,675463        0,670582        0,004881
2,5     0,598472        0,594072        0,004401
2,6     0,515501        0,507908        0,007593
2,7     0,427380        0,421745        0,005635
2,8     0,334988        0,334698        0,000290
2,9     0,239249        0,236716        0,002533
3,0     0,141120        0,138735        0,002385
3,1     0,041581        0,040753        0,000828
```

```
3. Сумма квадратов:
Значения суммы квадратов с 10 точками табуляции:
f(0,0) = 1,0000    f(0,1) = 0,9753    f(0,2) = 0,9705    f(0,3) = 0,9854
f(0,4) = 0,9850    f(0,5) = 0,9704    f(0,6) = 0,9756    f(0,7) = 0,9994
f(0,8) = 0,9751    f(0,9) = 0,9706    f(1,0) = 0,9859    f(1,1) = 0,9845
f(1,2) = 0,9703    f(1,3) = 0,9759    f(1,4) = 0,9987    f(1,5) = 0,9748
f(1,6) = 0,9707    f(1,7) = 0,9864    f(1,8) = 0,9841    f(1,9) = 0,9702
f(2,0) = 0,9762    f(2,1) = 0,9981    f(2,2) = 0,9745    f(2,3) = 0,9708
f(2,4) = 0,9869    f(2,5) = 0,9836    f(2,6) = 0,9702    f(2,7) = 0,9765
f(2,8) = 0,9975    f(2,9) = 0,9743    f(3,0) = 0,9709    f(3,1) = 0,9873


Влияние количества точек:

Количество точек: 5
Средняя погрешность суммы квадратов: 0,09735
Количество точек: 10
Средняя погрешность суммы квадратов: 0,01982
Количество точек: 20
Средняя погрешность суммы квадратов: 0,00454
Количество точек: 50
Средняя погрешность суммы квадратов: 0,00068

4. Экспонента и файлы (символы):
Исходная табулированная exp(x):
Количество точек: 11
Область определения: [0,00, 10,00]
Точки функции:
   0: x =  0,0000, y =   1,000000
   1: x =  1,0000, y =   2,718282
   2: x =  2,0000, y =   7,389056
   3: x =  3,0000, y =  20,085537
   4: x =  4,0000, y =  54,598150
   5: x =  5,0000, y = 148,413159
   6: x =  6,0000, y = 403,428793
   7: x =  7,0000, y = 1096,633158
   8: x =  8,0000, y = 2980,957987
   9: x =  9,0000, y = 8103,083928
  10: x = 10,0000, y = 22026,465795
Функция записана в файл: exp_function.txt
```

```
Считанная из файла exp(x):
Количество точек: 11
Область определения: [0,00, 10,00]
Точки функции:
   0: x =  0,0000, y =   1,000000
   1: x =  1,0000, y =   2,718282
   2: x =  2,0000, y =   7,389056
   3: x =  3,0000, y =  20,085537
   4: x =  4,0000, y =  54,598150
   5: x =  5,0000, y = 148,413159
   6: x =  6,0000, y = 403,428793
   7: x =  7,0000, y = 1096,633158
   8: x =  8,0000, y = 2980,957987
   9: x =  9,0000, y = 8103,083928
  10: x = 10,0000, y = 22026,465795

Сравнение исходной и считанной функций:
x       Исходная    Считанная   Совпадают
0,0     1,000000    1,000000    да
1,0     2,718282    2,718282    да
2,0     7,389056    7,389056    да
3,0     20,085537   20,085537   да
4,0     54,598150   54,598150   да
5,0     148,413159  148,413159  да
6,0     403,428793  403,428793  да
7,0     1096,633158 1096,633158 да
8,0     2980,957987 2980,957987 да
9,0     8103,083928 8103,083928 да
10,0        22026,465795    22026,465795    да
Все значения совпадают
```

```
5. Логарифм и файлы (байты):
Исходная табулированная ln(x) от 1 до 10:
Количество точек: 11
Область определения: [1,00, 10,00]
Точки функции:
   0: x =  1,0000, y =   0,000000
   1: x =  1,9000, y =   0,641854
   2: x =  2,8000, y =   1,029619
   3: x =  3,7000, y =   1,308333
   4: x =  4,6000, y =   1,526056
   5: x =  5,5000, y =   1,704748
   6: x =  6,4000, y =   1,856298
   7: x =  7,3000, y =   1,987874
   8: x =  8,2000, y =   2,104134
   9: x =  9,1000, y =   2,208274
  10: x = 10,0000, y =   2,302585
Функция записана в файл: ln_function.dat

Считанная из файла ln(x):
Количество точек: 11
Область определения: [1,00, 10,00]
Точки функции:
   0: x =  1,0000, y =   0,000000
   1: x =  1,9000, y =   0,641854
   2: x =  2,8000, y =   1,029619
   3: x =  3,7000, y =   1,308333
   4: x =  4,6000, y =   1,526056
   5: x =  5,5000, y =   1,704748
   6: x =  6,4000, y =   1,856298
   7: x =  7,3000, y =   1,987874
   8: x =  8,2000, y =   2,104134
   9: x =  9,1000, y =   2,208274
  10: x = 10,0000, y =   2,302585
```

```
Сравнение исходной и считанной функций:
x       Исходная    Считанная   Совпадают
1,0     0,000000    0,000000    да
2,0     0,684939    0,684939    да
3,0     1,091556    1,091556    да
4,0     1,380907    1,380907    да
5,0     1,605475    1,605475    да
6,0     1,788942    1,788942    да
7,0     1,944016    1,944016    да
8,0     2,078299    2,078299    да
9,0     2,196703    2,196703    да
10,0        2,302585    2,302585    да
Все значения совпадают

Проверка исключений через интерфейс

Проверка исключений
ArrayTabulatedFunction:
FunctionPointIndexOutOfBoundsException: Индекс точки выходит за границы: 10
InappropriateFunctionPointException: Точка с координатой X=2.0 уже существует
IllegalStateException: Невозможно удалить точку: должно остаться минимум 2 точки

LinkedListTabulatedFunction:
FunctionPointIndexOutOfBoundsException: Индекс за пределами списка: 10
InappropriateFunctionPointException: Точка с координатой X=2.0 уже существует
IllegalStateException: Невозможно удалить точку: должно остаться минимум 2 точки

Проверка исключений в конструкторах:
IllegalArgumentException (левая > правой): Левая граница области определения >= правой
IllegalArgumentException (точек < 2): Количество точек меньше двух
```

```
Тестирование сериализации

>>> Serializable (ArrayTabulatedFunction)

Сериализация через Serializable:
   Serializable успешно: первая точка: (0.0, 0.0)
>>> Externalizable (LinkedListTabulatedFunction)

Сериализация через Externalizable:
   Externalizable успешно: первая точка: 0.0, 0.0)

Сравнение способов хранения
   x            Serializable            Externalizable
   0,0      0,000000            0,000000
   1,0      1,000000            1,000000
   2,0      2,000000            2,000000
   3,0      3,000000            3,000000
   4,0      4,000000            4,000000
   5,0      5,000000            5,000000
   6,0      6,000000            6,000000
   7,0      7,000000            7,000000
   8,0      8,000000            8,000000
   9,0      9,000000            9,000000
   10,0     10,000000           10,000000

   Размер файлов:
      Serializable (ser.bin): 458 байт
      Externalizable (ext.bin): 186 байт

Process finished with exit code 0
```