ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА №4

Студент Трофимов А. В.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Задание №1

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction были добавлены конструкторы, принимающие массив объектов типа FunctionPoint. Они проверяли корректность данных: должно быть минимум 2 точки, и точки должны быть упорядочены по возрастанию X.

```
60      // НОВЫЙ конструктор с массивом точек (Задание 1)
61      public ArrayTabulatedFunction(FunctionPoint[] pointsArray) {
62          if (pointsArray.length < 2) {
63              throw new IllegalArgumentException(s: "At least 2 points required");
64          }
65
66          // Проверка упорядоченности
67          for (int i = 0; i < pointsArray.length - 1; i++) {
68              if (pointsArray[i].getX() >= pointsArray[i + 1].getX()) {
69                  throw new IllegalArgumentException(s: "Points must be ordered by X");
70              }
71          }
72
73          this.points = new FunctionPoint[pointsArray.length + 10];
74          this.pointsCount = pointsArray.length;
75
76          for (int i = 0; i < pointsArray.length; i++) {
77              this.points[i] = new FunctionPoint(pointsArray[i]);
78          }
79      }
```

```
35      // НОВЫЙ конструктор с массивом точек (Задание 1)
36      public LinkedListTabulatedFunction(FunctionPoint[] pointsArray) {
37          if (pointsArray.length < 2) {
38              throw new IllegalArgumentException(s: "At least 2 points required");
39          }
40
41          // Проверка упорядоченности
42          for (int i = 0; i < pointsArray.length - 1; i++) {
43              if (pointsArray[i].getX() >= pointsArray[i + 1].getX()) {
44                  throw new IllegalArgumentException(s: "Points must be ordered by X");
45              }
46          }
47
48          init();
49          for (FunctionPoint point : pointsArray) {
50              addNodeToTail(new FunctionPoint(point));
51          }
52      }
```

Задание №2

В пакете functions был создан интерфейс Function, а так же в TabulatedFunction были убраны соответствующие методы. Это позволило расширить интерфейс Function.

```java
package functions;

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

```java
package functions;

public interface TabulatedFunction extends Function {
    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point)
        throws InappropriateFunctionPointException, FunctionPointIndexOutOfBoundsException;
    double getPointX(int index) throws FunctionPointIndexOutOfBoundsException;
    void setPointX(int index, double x)
        throws InappropriateFunctionPointException, FunctionPointIndexOutOfBoundsException;
    double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;
    void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;

    void deletePoint(int index)
        throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Задание №3

Был создал пакет functions.basic. В нем были функции, которые позволили вычислять значение экспоненты, логарифма, косинуса, синуса и тангенса.

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {
    @Override
    public double getLeftDomainBorder() {
        return -Double.MAX_VALUE;
    }

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

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

```java
package functions.basic;

import functions.Function;

public class Log implements Function {
    private double base;

    public Log(double base) {
        if (base <= 0 || Math.abs(base - 1) < 1e-9) {
            throw new IllegalArgumentException(s: "Base must be positive and not equal to 1");
        }
        this.base = base;
    }

    @Override
    public double getLeftDomainBorder() {
        return 0; // Логарифм определен для x > 0
    }

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

    @Override
    public double getFunctionValue(double x) {
        if (x < 0) return Double.NaN;
        return Math.log(x) / Math.log(base);
    }

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

```java
package functions.basic;

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

```java
package functions.basic;

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

```java
package functions.basic;

public class Tan extends TrigonometricFunction {
    @Override
    public double getFunctionValue(double x) {
        // Тангенс не определен при x = π/2 + πk
        double cos = Math.cos(x);
        if (Math.abs(cos) < 1e-9) {
            return Double.NaN;
        }
        return Math.tan(x);
    }
}
```

Так как синус, косинус и тангенс имеют одинаковую область определения, создан общий базовый класс TrigonometricFunction с уже реализованными границами. От него наследуются: sin, cos, tan.

```java
package functions.basic;

import functions.Function;

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

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

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

Задание №4

Был создан пакет functions.meta, в котором были созданы классы, которые позволяют комбинировать функции.

Был создан класс Sum, который возвращает сумму двух других функций. Границы задаются с помощью минимального и максимального значения из двух функций.

```java
package functions.meta;

import functions.Function;

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

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

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

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

    @Override
    public double getFunctionValue(double x) {
        // Проверяем, что x в области определения обоих функций
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

Был создан класс Mult, который аналогичен Sum, только возвращает произведение двух других 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;
    }

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

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

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

Был создан класс Power, который позволяет вернуть функцию, которая является степенью другой функции.

```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;
    }

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

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

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

Был создан класс Scale, который позволяет масштабировать функцию вдоль осей координат.

```java
package functions.meta;

import functions.Function;

public class Scale implements Function {
    private Function f;
    private double scaleX;
    private double scaleY;

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

    @Override
    public double getLeftDomainBorder() {
        if (scaleX > 0) {
            return f.getLeftDomainBorder() * scaleX;
        } else {
            return f.getRightDomainBorder() * scaleX;
        }
    }

    @Override
    public double getRightDomainBorder() {
        if (scaleX > 0) {
            return f.getRightDomainBorder() * scaleX;
        } else {
            return f.getLeftDomainBorder() * scaleX;
        }
    }

    @Override
    public double getFunctionValue(double x) {
        return f.getFunctionValue(x / scaleX) * scaleY;
    }
}
```

Был создан класс Shift, который позволяет сдвигать функцию 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;
    }

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

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

    @Override
    public double getFunctionValue(double x) {
        return f.getFunctionValue(x - shiftX) + shiftY;
    }
}
```

Был создан класс Composition, который позволяет возвращать композицию двух функций.

```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;
    }

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

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

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

Задание 5

В пакете functions создал класс Functions, содержащий вспомогательные статические методы для работы с функциями. Сделал так, чтобы в программе вне этого класса нельзя было создать его объект.

- public static Function shift(Function f, double shiftX, double shiftY) – возвращает объект функции, полученной из исходной сдвигом вдоль осей;
- public static Function scale(Function f, double scaleX, double scaleY) – возвращает объект функции, полученной из исходной масштабированием вдоль осей;
- public static Function power(Function f, double power) – возвращает объект функции, являющейся заданной степенью исходной;
- public static Function sum(Function f1, Function f2) – возвращает объект функции, являющейся суммой двух исходных;
- public static Function mult(Function f1, Function f2) – возвращает объект функции, являющейся произведением двух исходных;
- public static Function composition(Function f1, Function f2) – возвращает объект функции, являющейся композицией двух исходных.

```java
package functions;

import functions.meta.*;

public class Functions {
    // Приватный конструктор, чтобы нельзя было создать экземпляр
    private Functions() {}

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

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

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

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

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

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

Задание 6

В пакете functions был создан класс TabulatedFunctions.

В классе был создан метод public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount), который получает функцию и возвращает её табулированный аналог на заданном отрезке с заданным количеством точек.

```java
package functions;

import java.io.*;

public class TabulatedFunctions {
    // Приватный конструктор
    private TabulatedFunctions() {}

    // Задание 6: метод табулирования
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException(s: "Tabulation interval is outside function domain");
        }

        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);
        }

        // Используем ArrayTabulatedFunction по умолчанию
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }
```

Задание 7

В класс TabulatedFunctions были добавлены следующие методы.

Метод вывода табулированной функции в байтовый поток public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out), который позволяет вывести в байтовый поток значения, по которым после можно будет восстановить табулированную функцию.

Метод ввода табулированной функции из байтового потока public static TabulatedFunction inputTabulatedFunction(InputStream in), который позволяет считать из указанного потока данные о табулированной функции.

```
29      public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
30          DataOutputStream dos = new DataOutputStream(out);
31          dos.writeInt(function.getPointsCount());
32
33          for (int i = 0; i < function.getPointsCount(); i++) {
34              FunctionPoint point = function.getPoint(i);
35              dos.writeDouble(point.getX());
36              dos.writeDouble(point.getY());
37          }
38
39          dos.flush();
40      }
41
42      public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
43          DataInputStream dis = new DataInputStream(in);
44          int pointsCount = dis.readInt();
45
46          FunctionPoint[] points = new FunctionPoint[pointsCount];
47          for (int i = 0; i < pointsCount; i++) {
48              double x = dis.readDouble();
49              double y = dis.readDouble();
50              points[i] = new FunctionPoint(x, y);
51          }
52
53          // Используем ArrayTabulatedFunction по умолчанию
54          return new ArrayTabulatedFunction(points);
55      }
```

Метод записи табулированной функции в символьный поток public static void writeTabulatedFunction(TabulatedFunction function, Writer out), который позволяет вывести в символьный поток значения, по которым после можно будет восстановить табулированную функцию.

Метод чтения табулированной функции из символьного потока public static TabulatedFunction readTabulatedFunction(Reader in), который позволяет считать из указаного потока данные о табулированной функции.

```
57      public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
58          PrintWriter writer = new PrintWriter(out);
59          writer.print(function.getPointsCount());
60
61          for (int i = 0; i < function.getPointsCount(); i++) {
62              FunctionPoint point = function.getPoint(i);
63              writer.print(" " + point.getX() + " " + point.getY());
64          }
65
66          writer.flush();
67      }
68
69      public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
70          StreamTokenizer tokenizer = new StreamTokenizer(in);
71          tokenizer.parseNumbers();
72
73          // Читаем количество точек
74          tokenizer.nextToken();
75          int pointsCount = (int)tokenizer.nval;
76
77          FunctionPoint[] points = new FunctionPoint[pointsCount];
78
79          for (int i = 0; i < pointsCount; i++) {
80              tokenizer.nextToken();
81              double x = tokenizer.nval;
82
83              tokenizer.nextToken();
84              double y = tokenizer.nval;
85
86              points[i] = new FunctionPoint(x, y);
87          }
88
89          // Используем ArrayTabulatedFunction по умолчанию
90          return new ArrayTabulatedFunction(points);
91      }
92  }
```

Для обработки исключения IOException, которое может возникать, если метод не нашел файл, если отсутствуют права доступа из-за ошибки чтения и других некоторых причин, в каждом методе объявлено, что он выбрасывает исключение IOException.

Поток закрывать не следует, т. к. если бы метод закрыл поток, клиент получил бы исключение при попытке дописать данные.

Задание №8

Проверка базовых функций

```
=== Задание 8: Тестирование аналитических функций ===

1. Тестирование Sin и Cos:
Sin(0) = 0.0
Cos(0) = 1.0
Sin(p/2) = 1.0
Cos(p/2) = 6.1232339957367660E-17
```

Проверка табулирования

```
2. Табулирование Sin и Cos:
Табулированный Sin в точке p/2: 0.9848077530122080
Табулированный Cos в точке p/2: 5.5511151231257830E-17
```

Проверка тригонометрического тождества

При увеличении pointsCount совпадение значений табулированной функции с изначальной увеличивается, поэтому сумма квадратов ближе к 1, чем при pointsCount = 10.

```
3. Сумма квадратов Sin и Cos (должна быть ~1):
f(0,0) = 1,000000
f(0,1) = 0,975345
f(0,2) = 0,970488
f(0,3) = 0,985429
f(0,4) = 0,984968
f(0,5) = 0,970398
f(0,6) = 0,975624
f(0,7) = 0,999358
f(0,8) = 0,975073
f(0,9) = 0,970586
f(1,0) = 0,985897
f(1,1) = 0,984515
f(1,2) = 0,970314
f(1,3) = 0,975910
f(1,4) = 0,998723
f(1,5) = 0,974808
f(1,6) = 0,970691
f(1,7) = 0,986371
f(1,8) = 0,984068
f(1,9) = 0,970237
f(2,0) = 0,976203
f(2,1) = 0,998094
f(2,2) = 0,974549
f(2,3) = 0,970802
f(2,4) = 0,986852
f(2,5) = 0,983628
f(2,6) = 0,970167
f(2,7) = 0,976503
f(2,8) = 0,997473
f(2,9) = 0,974298
f(3,0) = 0,970920
f(3,1) = 0,987341
```

Проверка комбинированных функций

```
=== Тестирование комбинированных функций ===

4. Тестирование Exp и Log:
exp(0) = 1.0
ln(1) = 0.0
ln(exp(2)) = 2.0
exp(x-1) в точке 1 = 1.0
```

Тестирование ввода/вывода

```
=== Тестирование ввода/вывода ===

5. Тестирование записи/чтения файлов:
Сравнение исходной и прочитанной из текстового файла:
Точка 0: исходная=1,000000, прочитанная=1,000000
Точка 1: исходная=2,718282, прочитанная=2,718282
Точка 2: исходная=7,389056, прочитанная=7,389056
Точка 3: исходная=20,085537, прочитанная=20,085537
Точка 4: исходная=54,598150, прочитанная=54,598150
Точка 5: исходная=148,413159, прочитанная=148,413159
Точка 6: исходная=403,428793, прочитанная=403,428793
Точка 7: исходная=1096,633158, прочитанная=1096,633158
Точка 8: исходная=2980,957987, прочитанная=2980,957987
Точка 9: исходная=8103,083928, прочитанная=8103,083928
Точка 10: исходная=22026,465795, прочитанная=22026,465795

Сравнение исходной и прочитанной из бинарного файла:
Точка 0: исходная=1,000000, прочитанная=1,000000
Точка 1: исходная=2,718282, прочитанная=2,718282
Точка 2: исходная=7,389056, прочитанная=7,389056
Точка 3: исходная=20,085537, прочитанная=20,085537
Точка 4: исходная=54,598150, прочитанная=54,598150
Точка 5: исходная=148,413159, прочитанная=148,413159
Точка 6: исходная=403,428793, прочитанная=403,428793
Точка 7: исходная=1096,633158, прочитанная=1096,633158
Точка 8: исходная=2980,957987, прочитанная=2980,957987
Точка 9: исходная=8103,083928, прочитанная=8103,083928
Точка 10: исходная=22026,465795, прочитанная=22026,465795
```

Проверка сериализации

```
=== Задание 9: Тестирование сериализации ===

6. Тестирование сериализации:
Сравнение после сериализации:
Точка 0: исходная=0,000000, после сериализации=0,000000
Точка 1: исходная=1,000000, после сериализации=1,000000
Точка 2: исходная=2,000000, после сериализации=2,000000
Точка 3: исходная=3,000000, после сериализации=3,000000
Точка 4: исходная=4,000000, после сериализации=4,000000
Точка 5: исходная=5,000000, после сериализации=5,000000
Точка 6: исходная=6,000000, после сериализации=6,000000
Точка 7: исходная=7,000000, после сериализации=7,000000
Точка 8: исходная=8,000000, после сериализации=8,000000
Точка 9: исходная=9,000000, после сериализации=9,000000
Точка 10: исходная=10,000000, после сериализации=10,000000
```

Serializable:
Плюс — не нужно писать код, все работает автоматически
Минус — нельзя контролировать формат сериализации
Externalizable:
Плюс — полный контроль над данными и оптимизация
Минус — нужно вручную писать методы сериализации

Для Externalizable были созданы два метода для записи и вывода.

```
12      @Override
13      public void writeExternal(ObjectOutput out) throws IOException {
14          out.writeInt(pointsCount);
15          for (int i = 0; i < pointsCount; i++) {
16              out.writeDouble(points[i].getX());
17              out.writeDouble(points[i].getY());
18          }
19      }
20
21      @Override
22      public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
23          pointsCount = in.readInt();
24          points = new FunctionPoint[pointsCount + 10];
25          for (int i = 0; i < pointsCount; i++) {
26              double x = in.readDouble();
27              double y = in.readDouble();
28              points[i] = new FunctionPoint(x, y);
29          }
30      }
```