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 Ефремова А. С.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Задание №1

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction создала конструкторы, получающие все точки в виде массива объектов типа FunctionPoint. В них добавления проверки минимального количества точек и упорядочиванию по координатам X.

```
public LinkedListTabulatedFunction(FunctionPoint[] points){
    if (points.length < 2) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() < points[i - 1].getX()){
            throw new IllegalStateException(s: "Массив не упорядочен по координатам X");
        }
    }
    pointsCount = points.length;
    head = new FunctionNode(point: null, prev: null, next: null);
    head.setNext(head);
    head.setPrev(head);
    for (int i = 0; i < points.length; i++) {
        addNodeToTail(new FunctionPoint(points[i]));
    }
}
```

```
public ArrayTabulatedFunction(FunctionPoint[] points) {
    if (points.length < 2) {
        throw new IllegalStateException(s: "В массиве не может быть только одна точка");
    }
    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() < points[i - 1].getX()){
            throw new IllegalStateException(s: "Массив не упорядочен по координатам X");
        }
    }
    pointsCount = points.length;
    this.points = new FunctionPoint[points.length];
    for (int i = 0; i < points.length; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

Задание №2

В пакете functions создала интерфейс Function, убрал в TabulatedFunction соответствующие методы и сделал так, что он расширяет интерфейс Function.

```
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 ;
    double getPointX(int index);
    void setPointX(int index,double x) throws InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index,double y);
    void deletePoint(int index) throws InappropriateFunctionPointException;
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Задание №3

Создала пакет functions.basic и описал в нем функции позволяющие 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.NEGATIVE_INFINITY;
    }

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

    @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) {
            throw new IllegalArgumentException(s: "Основание логарифма должно быть > 0");
        }
        this.base = base;
    }

    @Override
    public double getLeftDomainBorder() {
        return 0;
    }

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

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

Так как границы у синуса, косинуса и тангенса совпадают, то создала класс

TrigonometricFunction в котором описаны методы получения границ. И уже от этого класса наследуются публичные классы Sin, Cos, Tan.

```
package functions.basic;

import functions.Function;

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

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

Задание №4

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

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

```
package functions.meta;

import functions.Function;

public class Sum implements Function {
    private Function Func1;
    private Function Func2;

    public Sum(Function Func1, Function Func2) throws IllegalArgumentException {
        if (Func1 == null || Func2 == null){
            throw new IllegalArgumentException(s: "Функции не могут быть null");
        }
        this.Func1 = Func1;
        this.Func2 = Func2;
    }

    @Override
    public double getLeftDomainBorder() {
        return Math.max(Func1.getLeftDomainBorder(), Func2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(Func1.getRightDomainBorder(), Func2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        return Func1.getFunctionValue(x) + Func2.getFunctionValue(x);
    }
}
```

Создала 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 Func1;
    private Function Func2;

    public Mult(Function Func1, Function Func2) throws IllegalArgumentException {
        if (Func1 == null || Func2 == null){
            throw new IllegalArgumentException(s: "Функции не могут быть null");
        }
        this.Func1 = Func1;
        this.Func2 = Func2;
    }

    @Override
    public double getLeftDomainBorder() {
        return Math.max(Func1.getLeftDomainBorder(), Func2.getLeftDomainBorder());
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(Func1.getRightDomainBorder(), Func2.getRightDomainBorder());
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        return Func1.getFunctionValue(x) * Func2.getFunctionValue(x);
    }
}
```

Создала класс Power, позволяющий вернуть функцию, которая является степенью другой функции.

```java
package functions.meta;

import functions.Function;

public class Power implements Function {
    private Function Func;
    private double cons;

    public Power(Function Func, double cons) {
        if (Func == null){
            throw new IllegalArgumentException(s: "Функция не могут быть null");
        }
        this.Func = Func;
        this.cons = cons;
    }

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

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

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        return Math.pow(Func.getFunctionValue(x), cons);
    }
}
```

Создала класс Scale, позволяющий масштабировать функцию вдоль осей координат.

```java
package functions.meta;

import functions.Function;

public class Scale implements Function {
    private Function Func;
    private double scaleX, scaleY;

    public Scale(Function Func, double scaleX, double scaleY) {
        if (Func == null){
            throw new IllegalArgumentException(s: "Функция не может быть null");
        }
        this.Func = Func;
        this.scaleX = scaleX;
        this.scaleY = scaleY;
    }

    @Override
    public double getLeftDomainBorder() {
        if (scaleX > 0){
            return Func.getLeftDomainBorder() * scaleX;
        } else if (scaleX < 0){
            return Func.getRightDomainBorder() * scaleX;
        } else {
            return Double.NaN;
        }

    }

    @Override
    public double getRightDomainBorder() {
        if (scaleX > 0){
            return Func.getRightDomainBorder() * scaleX;
        } else if (scaleX < 0){
            return Func.getLeftDomainBorder() * scaleX;
        } else {
            return Double.NaN;
        }
    }

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

Создала класс Shift позволяющий сдвигать функцию вдоль осей координат.

```java
package functions.meta;

import functions.Function;

public class Shift implements Function {
    Function Func;
    double shiftX, shiftY;

    public Shift(Function Func, double shiftX, double shiftY){
        if (Func == null){
            throw new IllegalArgumentException(s: "Функция не может быть null");
        }
        this.Func = Func;
        this.shiftX = shiftX;
        this.shiftY = shiftY;
    }

    @Override
    public double getLeftDomainBorder(){
        return Func.getLeftDomainBorder() - shiftX;
    }

    @Override
    public double getRightDomainBorder(){
        return Func.getRightDomainBorder() - shiftY;
    }

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

}
```

Создала класс Composition, позволяющий возвращать композицию двух функций.

```java
package functions.meta;

import functions.Function;

public class Composition implements Function {
    private Function Func1;
    private Function Func2;

    public Composition(Function Func2, Function Func1) {
        if (Func1 == null || Func2 == null){
            throw new IllegalArgumentException(s: "Функции не могут быть null");
        }
        this.Func1 = Func1;
        this.Func2 = Func2;
    }

    @Override
    public double getLeftDomainBorder() {
        return Func1.getLeftDomainBorder();
    }

    @Override
    public double getRightDomainBorder() {
        return Func2.getRightDomainBorder();
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()){
            return Double.NaN;
        }
        return Func1.getFunctionValue(Func2.getFunctionValue(x));
    }
}
```

Задание 5

В пакете functions создала класс Functions, содержащий вспомогательные статические методы для работы с функциями. Сделала 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 final class 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 создала класс TabulatedFunctions.

Создала в классе метод public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount), получающий функцию и возвращающий её табулированный аналог на заданном отрезке с заданным количеством точек.

```
package functions;

import java.io.*;

public final class TabulatedFunctions {

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        if(leftX >= rightX){
            throw new IllegalArgumentException(s: "Левая граница больше или равна правой");
        }
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException(s: "Заданные границы выходят за область определения");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException(s: "Требуется не менее 2 точек");
        }
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            points[i] = new FunctionPoint(x, function.getFunctionValue(x));
        }

        return new ArrayTabulatedFunction(points);
    }
```

Задание 7

В класс TabulatedFunctions добавила следующие методы.

Метод вывода табулированной функции в байтовый поток public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out), позволяющий 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), позволяющий считать из указаного потока данные о табулированной функции.

```
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
    DataOutputStream dataOut = new DataOutputStream(out);
    dataOut.writeInt(function.getPointsCount());
    for (int i = 0; i < function.getPointsCount(); ++i) {
        FunctionPoint point = function.getPoint(i);
        dataOut.writeDouble(point.getX());
        dataOut.writeDouble(point.getY());
    }
    dataOut.flush();
}

public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    DataInputStream dis = new DataInputStream(in);
    int pointCount = dis.readInt();
    FunctionPoint[] points = new FunctionPoint[pointCount];

    for (int i = 0; i < pointCount; i++) {
        points[i] = new FunctionPoint(dis.readDouble(), dis.readDouble());
    }
    return new ArrayTabulatedFunction(points);
}
```

Метод записи табулированной функции в символьный поток public static void writeTabulatedFunction(TabulatedFunction function, Writer out) позволяющий 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) позволяющий считать из указаного потока данные о табулированной функции.

```
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    BufferedWriter Writer = new BufferedWriter(out);
    int pointsCount = function.getPointsCount();
    Writer.write(" " + pointsCount);

    for (int i = 0; i < pointsCount; i++) {
        FunctionPoint point = function.getPoint(i);
        Writer.write("\n " + point.getX());
        Writer.write(" " + point.getY());
    }
    Writer.flush();
}

public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
    StreamTokenizer st = new StreamTokenizer(in);
    st.nextToken();
    int pointsCount = (int) st.nval;
    FunctionPoint[] points = new FunctionPoint[pointsCount];

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

    return new ArrayTabulatedFunction(points);
}
```

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, так как если бы метод закрыл поток, клиент получил бы исключение при попытке дописать данные.

Задание №8

При pointsCount = 10.

```
Значения sin и cos

sin(0,00) = 0,000000     cos(0,00) = 1,000000
sin(0,10) = 0,099833     cos(0,10) = 0,995004
sin(0,20) = 0,198669     cos(0,20) = 0,980067
sin(0,30) = 0,295520     cos(0,30) = 0,955336
sin(0,40) = 0,389418     cos(0,40) = 0,921061
sin(0,50) = 0,479426     cos(0,50) = 0,877583
sin(0,60) = 0,564642     cos(0,60) = 0,825336
sin(0,70) = 0,644218     cos(0,70) = 0,764842
sin(0,80) = 0,717356     cos(0,80) = 0,696707
sin(0,90) = 0,783327     cos(0,90) = 0,621610
sin(1,00) = 0,841471     cos(1,00) = 0,540302
sin(1,10) = 0,891207     cos(1,10) = 0,453596
sin(1,20) = 0,932039     cos(1,20) = 0,362358
sin(1,30) = 0,963558     cos(1,30) = 0,267499
sin(1,40) = 0,985450     cos(1,40) = 0,169967
sin(1,50) = 0,997495     cos(1,50) = 0,070737
sin(1,60) = 0,999574     cos(1,60) = -0,029200
sin(1,70) = 0,991665     cos(1,70) = -0,128844
sin(1,80) = 0,973848     cos(1,80) = -0,227202
sin(1,90) = 0,946300     cos(1,90) = -0,323290
sin(2,00) = 0,909297     cos(2,00) = -0,416147
sin(2,10) = 0,863209     cos(2,10) = -0,504846
sin(2,20) = 0,808496     cos(2,20) = -0,588501
sin(2,30) = 0,745705     cos(2,30) = -0,666276
sin(2,40) = 0,675463     cos(2,40) = -0,737394
sin(2,50) = 0,598472     cos(2,50) = -0,801144
sin(2,60) = 0,515501     cos(2,60) = -0,856889
sin(2,70) = 0,427380     cos(2,70) = -0,904072
sin(2,80) = 0,334988     cos(2,80) = -0,942222
sin(2,90) = 0,239249     cos(2,90) = -0,970958
sin(3,00) = 0,141120     cos(3,00) = -0,989992
sin(3,10) = 0,041581     cos(3,10) = -0,999135

Значения табулированных sin и cos

sin(0,00) = 0,000000     cos(0,00) = 1,000000
sin(0,10) = 0,097982     cos(0,10) = 0,982723
sin(0,20) = 0,195963     cos(0,20) = 0,965446
sin(0,30) = 0,293945     cos(0,30) = 0,948170
sin(0,40) = 0,385907     cos(0,40) = 0,914355
sin(0,50) = 0,472070     cos(0,50) = 0,864608
sin(0,60) = 0,558234     cos(0,60) = 0,814862
sin(0,70) = 0,643982     cos(0,70) = 0,764620
sin(0,80) = 0,707935     cos(0,80) = 0,688404
sin(0,90) = 0,771888     cos(0,90) = 0,612188
sin(1,00) = 0,835841     cos(1,00) = 0,535972
sin(1,10) = 0,883993     cos(1,10) = 0,450633
sin(1,20) = 0,918022     cos(1,20) = 0,357141
sin(1,30) = 0,952051     cos(1,30) = 0,263648
sin(1,40) = 0,984808     cos(1,40) = 0,169931
sin(1,50) = 0,984808     cos(1,50) = 0,070437
sin(1,60) = 0,984808     cos(1,60) = -0,029056
sin(1,70) = 0,984808     cos(1,70) = -0,128549
sin(1,80) = 0,966204     cos(1,80) = -0,224761
sin(1,90) = 0,932175     cos(1,90) = -0,318254
sin(2,00) = 0,898147     cos(2,00) = -0,411747
sin(2,10) = 0,862441     cos(2,10) = -0,504272
sin(2,20) = 0,798488     cos(2,20) = -0,580488
sin(2,30) = 0,734535     cos(2,30) = -0,656704
sin(2,40) = 0,670582     cos(2,40) = -0,732920
sin(2,50) = 0,594072     cos(2,50) = -0,794171
sin(2,60) = 0,507908     cos(2,60) = -0,843917
sin(2,70) = 0,421745     cos(2,70) = -0,893664
sin(2,80) = 0,334698     cos(2,80) = -0,940984
sin(2,90) = 0,236716     cos(2,90) = -0,958261
sin(3,00) = 0,138735     cos(3,00) = -0,975537
sin(3,10) = 0,040753     cos(3,10) = -0,992814
```

```
Рассчет модуля разности между табулированными и исходными значениями sin и cos

Разность sin = 0,000000          Разность cos = 0,000000
Разность sin = 0,001852          Разность cos = 0,012281
Разность sin = 0,002706          Разность cos = 0,014620
Разность sin = 0,001576          Разность cos = 0,007167
Разность sin = 0,003512          Разность cos = 0,006706
Разность sin = 0,007355          Разность cos = 0,012974
Разность sin = 0,006409          Разность cos = 0,010474
Разность sin = 0,000235          Разность cos = 0,000222
Разность sin = 0,009421          Разность cos = 0,008302
Разность sin = 0,011439          Разность cos = 0,009422
Разность sin = 0,005630          Разность cos = 0,004330
Разность sin = 0,007214          Разность cos = 0,002963
Разность sin = 0,014017          Разность cos = 0,005217
Разность sin = 0,011508          Разность cos = 0,003851
Разность sin = 0,000642          Разность cos = 0,000037
Разность sin = 0,012687          Разность cos = 0,000300
Разность sin = 0,014766          Разность cos = 0,000144
Разность sin = 0,006857          Разность cos = 0,000296
Разность sin = 0,007644          Разность cos = 0,002441
Разность sin = 0,014125          Разность cos = 0,005035
Разность sin = 0,011151          Разность cos = 0,004400
Разность sin = 0,000768          Разность cos = 0,000574
Разность sin = 0,010008          Разность cos = 0,008013
Разность sin = 0,011170          Разность cos = 0,009572
Разность sin = 0,004881          Разность cos = 0,004474
Разность sin = 0,004401          Разность cos = 0,006973
Разность sin = 0,007593          Разность cos = 0,012972
Разность sin = 0,005635          Разность cos = 0,010408
Разность sin = 0,000290          Разность cos = 0,001239
Разность sin = 0,002533          Разность cos = 0,012698
Разность sin = 0,002385          Разность cos = 0,014455
Разность sin = 0,000828          Разность cos = 0,006321

Сумма квадратов синуса и косинуса

sin(0,00)^2 + cos(0,00)^2 = 1,000000
sin(0,10)^2 + cos(0,10)^2 = 0,975345
sin(0,20)^2 + cos(0,20)^2 = 0,970488
sin(0,30)^2 + cos(0,30)^2 = 0,985429
sin(0,40)^2 + cos(0,40)^2 = 0,984968
sin(0,50)^2 + cos(0,50)^2 = 0,970398
sin(0,60)^2 + cos(0,60)^2 = 0,975624
sin(0,70)^2 + cos(0,70)^2 = 0,999358
sin(0,80)^2 + cos(0,80)^2 = 0,975073
sin(0,90)^2 + cos(0,90)^2 = 0,970586
sin(1,00)^2 + cos(1,00)^2 = 0,985897
sin(1,10)^2 + cos(1,10)^2 = 0,984515
sin(1,20)^2 + cos(1,20)^2 = 0,970314
sin(1,30)^2 + cos(1,30)^2 = 0,975910
sin(1,40)^2 + cos(1,40)^2 = 0,998723
sin(1,50)^2 + cos(1,50)^2 = 0,974808
sin(1,60)^2 + cos(1,60)^2 = 0,970691
sin(1,70)^2 + cos(1,70)^2 = 0,986371
sin(1,80)^2 + cos(1,80)^2 = 0,984068
sin(1,90)^2 + cos(1,90)^2 = 0,970237
sin(2,00)^2 + cos(2,00)^2 = 0,976203
sin(2,10)^2 + cos(2,10)^2 = 0,998094
sin(2,20)^2 + cos(2,20)^2 = 0,974549
sin(2,30)^2 + cos(2,30)^2 = 0,970802
sin(2,40)^2 + cos(2,40)^2 = 0,986852
sin(2,50)^2 + cos(2,50)^2 = 0,983628
sin(2,60)^2 + cos(2,60)^2 = 0,970167
sin(2,70)^2 + cos(2,70)^2 = 0,976503
sin(2,80)^2 + cos(2,80)^2 = 0,997473
sin(2,90)^2 + cos(2,90)^2 = 0,974298
sin(3,00)^2 + cos(3,00)^2 = 0,970920
sin(3,10)^2 + cos(3,10)^2 = 0,987341
```

```
Значение экспонент

Exp = 1,000000   TabExp = 1,000000
Exp = 2,718282   TabExp = 2,718282
Exp = 7,389056   TabExp = 7,389056
Exp = 20,085537          TabExp = 20,085537
Exp = 54,598150          TabExp = 54,598150
Exp = 148,413159         TabExp = 148,413159
Exp = 403,428793         TabExp = 403,428793
Exp = 1096,633158        TabExp = 1096,633158
Exp = 2980,957987        TabExp = 2980,957987
Exp = 8103,083928        TabExp = 8103,083928
Exp = 22026,465795       TabExp = 22026,465795

Значение натурального логарифма

Ln(1) = 0,000000         TabLn(1) = 0,000000
Ln(2) = 0,684939         TabLn(2) = 0,684939
Ln(3) = 1,091556         TabLn(3) = 1,091556
Ln(4) = 1,380907         TabLn(4) = 1,380907
Ln(5) = 1,605475         TabLn(5) = 1,605475
Ln(6) = 1,788942         TabLn(6) = 1,788942
Ln(7) = 1,944016         TabLn(7) = 1,944016
Ln(8) = 2,078299         TabLn(8) = 2,078299
Ln(9) = 2,196703         TabLn(9) = 2,196703
Ln(10) = 2,302585        TabLn(10) = 2,302585
```

При увеличении pointsCount до 30 совпадение значений табулированной функции с изначальное увеличивается, поэтому сумма квадратов ближе к 1, чем при pointsCount = 10.

```
Сумма квадратов синуса и косинуса

sin(0,00)^2 + cos(0,00)^2 = 1,000000
sin(0,10)^2 + cos(0,10)^2 = 0,999168
sin(0,20)^2 + cos(0,20)^2 = 0,998474
sin(0,30)^2 + cos(0,30)^2 = 0,997919
sin(0,40)^2 + cos(0,40)^2 = 0,997503
sin(0,50)^2 + cos(0,50)^2 = 0,997225
sin(0,60)^2 + cos(0,60)^2 = 0,997086
sin(0,70)^2 + cos(0,70)^2 = 0,997086
sin(0,80)^2 + cos(0,80)^2 = 0,997225
sin(0,90)^2 + cos(0,90)^2 = 0,997502
sin(1,00)^2 + cos(1,00)^2 = 0,997917
sin(1,10)^2 + cos(1,10)^2 = 0,998472
sin(1,20)^2 + cos(1,20)^2 = 0,999165
sin(1,30)^2 + cos(1,30)^2 = 0,999997
sin(1,40)^2 + cos(1,40)^2 = 0,999171
sin(1,50)^2 + cos(1,50)^2 = 0,998476
sin(1,60)^2 + cos(1,60)^2 = 0,997921
sin(1,70)^2 + cos(1,70)^2 = 0,997504
sin(1,80)^2 + cos(1,80)^2 = 0,997226
sin(1,90)^2 + cos(1,90)^2 = 0,997087
sin(2,00)^2 + cos(2,00)^2 = 0,997086
sin(2,10)^2 + cos(2,10)^2 = 0,997224
sin(2,20)^2 + cos(2,20)^2 = 0,997500
sin(2,30)^2 + cos(2,30)^2 = 0,997916
sin(2,40)^2 + cos(2,40)^2 = 0,998470
sin(2,50)^2 + cos(2,50)^2 = 0,999162
sin(2,60)^2 + cos(2,60)^2 = 0,999993
sin(2,70)^2 + cos(2,70)^2 = 0,999173
sin(2,80)^2 + cos(2,80)^2 = 0,998479
sin(2,90)^2 + cos(2,90)^2 = 0,997923
sin(3,00)^2 + cos(3,00)^2 = 0,997506
sin(3,10)^2 + cos(3,10)^2 = 0,997227
```

Значения при использовании Serializable

```
Логарифм от экспоненты():
Exp(1) = 1,000000        TabExp(1) = 1,000000
Exp(2) = 2,000000        TabExp(2) = 2,000000
Exp(3) = 3,000000        TabExp(3) = 3,000000
Exp(4) = 4,000000        TabExp(4) = 4,000000
Exp(5) = 5,000000        TabExp(5) = 5,000000
Exp(6) = 6,000000        TabExp(6) = 6,000000
Exp(7) = 7,000000        TabExp(7) = 7,000000
Exp(8) = 8,000000        TabExp(8) = 8,000000
Exp(9) = 9,000000        TabExp(9) = 9,000000
Exp(10) = 10,000000      TabExp(10) = 10,000000
```

Значения при использовании Externalizable

```
Логарифм от экспоненты():
Exp(1) = 1,000000        TabExp(1) = 1,000000
Exp(2) = 2,000000        TabExp(2) = 2,000000
Exp(3) = 3,000000        TabExp(3) = 3,000000
Exp(4) = 4,000000        TabExp(4) = 4,000000
Exp(5) = 5,000000        TabExp(5) = 5,000000
Exp(6) = 6,000000        TabExp(6) = 6,000000
Exp(7) = 7,000000        TabExp(7) = 7,000000
Exp(8) = 8,000000        TabExp(8) = 8,000000
Exp(9) = 9,000000        TabExp(9) = 9,000000
Exp(10) = 10,000000      TabExp(10) = 10,000000
PS C:\Users\LuckyFrut\Documents\University\Lab-4-2025>
```

Для Externalizable создала два метода для записи и вывода.

```java
@Override
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(pointsCount);
    for (int i = 0; i < pointsCount; i++) {
        out.writeDouble(points[i].getX());
        out.writeDouble(points[i].getY());
    }
}

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    pointsCount = in.readInt();
    points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; ++i) {
        points[i] = new FunctionPoint(in.readDouble(), in.readDouble());
    }
}
```