

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 _____

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 }
80
81 // НОВЫЙ конструктор с массивом точек (Задание 1)
82 public LinkedListTabulatedFunction(FunctionPoint[] pointsArray) {
83     if (pointsArray.length < 2) {
84         throw new IllegalArgumentException(s: "At least 2 points required");
85     }
86
87     // Проверка упорядоченности
88     for (int i = 0; i < pointsArray.length - 1; i++) {
89         if (pointsArray[i].getX() >= pointsArray[i + 1].getX()) {
90             throw new IllegalArgumentException(s: "Points must be ordered by X");
91         }
92     }
93
94     init();
95     for (FunctionPoint point : pointsArray) {
96         addNodeToTail(new FunctionPoint(point));
97     }
98 }
```

Задание №2

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

```
1 package functions;
2
3 public interface Function {
4     double getLeftDomainBorder();
5     double getRightDomainBorder();
6     Q double getFunctionValue(double x);
7 }
```

```
1 package functions;
2
3 public interface TabulatedFunction extends Function {
4     int getPointsCount();
5     FunctionPoint getPoint(int index);
6     void setPoint(int index, FunctionPoint point)
7         |    throws InappropriateFunctionPointException, FunctionPointIndexOutOfBoundsException;
8     double getPointX(int index) throws FunctionPointIndexOutOfBoundsException;
9     void setPointX(int index, double x)
10        |   throws InappropriateFunctionPointException, FunctionPointIndexOutOfBoundsException;
11     double getPointY(int index) throws FunctionPointIndexOutOfBoundsException;
12     void setPointY(int index, double y) throws FunctionPointIndexOutOfBoundsException;
13
14     void deletePoint(int index)
15         |   throws FunctionPointIndexOutOfBoundsException, IllegalStateException;
16     Q void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
17 }
```

Задание №3

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

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Exp implements Function {
6     @Override
7     public double getLeftDomainBorder() {
8         return -Double.MAX_VALUE;
9     }
10
11     @Override
12     public double getRightDomainBorder() {
13         return Double.MAX_VALUE;
14     }
15
16     @Override
17     public double getFunctionValue(double x) {
18         return Math.exp(x);
19     }
20 }
21
22 package functions.basic;
23
24 import functions.Function;
25
26 public class Log implements Function {
27     private double base;
28
29     public Log(double base) {
30         if (base <= 0 || Math.abs(base - 1) < 1e-9) {
31             throw new IllegalArgumentException("Base must be positive and not equal to 1");
32         }
33         this.base = base;
34     }
35
36     @Override
37     public double getLeftDomainBorder() {
38         return 0; // Логарифм определен для x > 0
39     }
40
41     @Override
42     public double getRightDomainBorder() {
43         return Double.MAX_VALUE;
44     }
45
46     @Override
47     public double getFunctionValue(double x) {
48         if (x <= 0) return Double.NaN;
49         return Math.log(x) / Math.log(base);
50     }
51
52     public double getBase() {
53         return base;
54     }
55 }
56
57 package functions.basic;
58
59 public class Cos extends TrigonometricFunction {
60     @Override
61     public double getFunctionValue(double x) {
62         return Math.cos(x);
63     }
64 }
```

```
1 package functions.basic;
2
3 public class Sin extends TrigonometricFunction {
4     @Override
5     public double getFunctionValue(double x) {
6         return Math.sin(x);
7     }
8 }
9 package functions.basic;
10
11 public class Tan extends TrigonometricFunction {
12     @Override
13     public double getFunctionValue(double x) {
14         // Тангенс не определен при  $x = \pi/2 + \pi k$ 
15         double cos = Math.cos(x);
16         if (Math.abs(cos) < 1e-9) {
17             return Double.NaN;
18         }
19         return Math.tan(x);
20     }
21 }
```

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`.

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public abstract class TrigonometricFunction implements Function {
6     @Override
7     public double getLeftDomainBorder() {
8         return -Double.MAX_VALUE;
9     }
10
11     @Override
12     public double getRightDomainBorder() {
13         return Double.MAX_VALUE;
14     }
15
16     @Override
17     public abstract double getFunctionValue(double x);
18 }
```

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.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Sum implements Function {
6     private Function f1;
7     private Function f2;
8
9     public Sum(Function f1, Function f2) {
10         this.f1 = f1;
11         this.f2 = f2;
12     }
13
14     @Override
15     public double getLeftDomainBorder() {
16         return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
17     }
18
19     @Override
20     public double getRightDomainBorder() {
21         return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
22     }
23
24     @Override
25     public double getFunctionValue(double x) {
26         // Проверяем, что x в области определения обоих функций
27         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
28             return Double.NaN;
29         }
30         return f1.getFunctionValue(x) + f2.getFunctionValue(x);
31     }
32 }
```

Был создан класс Mult, который аналогичен Sum, только возвращает произведение двух других функций.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Mult implements Function {
6     private Function f1;
7     private Function f2;
8
9     public Mult(Function f1, Function f2) {
10         this.f1 = f1;
11         this.f2 = f2;
12     }
13
14     @Override
15     public double getLeftDomainBorder() {
16         return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
17     }
18
19     @Override
20     public double getRightDomainBorder() {
21         return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
22     }
23
24     @Override
25     public double getFunctionValue(double x) {
26         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
27             return Double.NaN;
28         }
29         return f1.getFunctionValue(x) * f2.getFunctionValue(x);
30     }
31 }
```

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

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Power implements Function {
6     private Function f;
7     private double power;
8
9     public Power(Function f, double power) {
10         this.f = f;
11         this.power = power;
12     }
13
14     @Override
15     public double getLeftDomainBorder() {
16         return f.getLeftDomainBorder();
17     }
18
19     @Override
20     public double getRightDomainBorder() {
21         return f.getRightDomainBorder();
22     }
23
24     @Override
25     public double getFunctionValue(double x) {
26         double value = f.getFunctionValue(x);
27         if (Double.isNaN(value)) {
28             return Double.NaN;
29         }
30         return Math.pow(value, power);
31     }
32 }
```

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

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Scale implements Function {
6     private Function f;
7     private double scaleX;
8     private double scaleY;
9
10    public Scale(Function f, double scaleX, double scaleY) {
11        this.f = f;
12        this.scaleX = scaleX;
13        this.scaleY = scaleY;
14    }
15
16    @Override
17    public double getLeftDomainBorder() {
18        if (scaleX > 0) {
19            return f.getLeftDomainBorder() * scaleX;
20        } else {
21            return f.getRightDomainBorder() * scaleX;
22        }
23    }
24
25    @Override
26    public double getRightDomainBorder() {
27        if (scaleX > 0) {
28            return f.getRightDomainBorder() * scaleX;
29        } else {
30            return f.getLeftDomainBorder() * scaleX;
31        }
32    }
33
34    @Override
35    public double getFunctionValue(double x) {
36        return f.getFunctionValue(x / scaleX) * scaleY;
37    }
38 }
```

Был создан класс Shift, который позволяет сдвигать функцию вдоль осей координат.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Shift implements Function {
6     private Function f;
7     private double shiftX;
8     private double shiftY;
9
10    public Shift(Function f, double shiftX, double shiftY) {
11        this.f = f;
12        this.shiftX = shiftX;
13        this.shiftY = shiftY;
14    }
15
16    @Override
17    public double getLeftDomainBorder() {
18        return f.getLeftDomainBorder() + shiftX;
19    }
20
21    @Override
22    public double getRightDomainBorder() {
23        return f.getRightDomainBorder() + shiftX;
24    }
25
26    @Override
27    public double getFunctionValue(double x) {
28        return f.getFunctionValue(x - shiftX) + shiftY;
29    }
30}
```

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

```
1 package functions.meta;
2
3 import functions.Function;
4
5 public class Composition implements Function {
6     private Function f1;
7     private Function f2;
8
9     public Composition(Function f1, Function f2) {
10        this.f1 = f1;
11        this.f2 = f2;
12    }
13
14    @Override
15    public double getLeftDomainBorder() {
16        return f1.getLeftDomainBorder();
17    }
18
19    @Override
20    public double getRightDomainBorder() {
21        return f1.getRightDomainBorder();
22    }
23
24    @Override
25    public double getFunctionValue(double x) {
26        double innerValue = f1.getFunctionValue(x);
27        if (Double.isNaN(innerValue)) {
28            return Double.NaN;
29        }
30        return f2.getFunctionValue(innerValue);
31    }
32}
```

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.

```
1 package functions;
2
3 import functions.meta.*;
4
5 public class Functions {
6     // Приватный конструктор, чтобы нельзя было создать экземпляра
7     private Functions() {}
8
9     public static Function shift(Function f, double shiftX, double shiftY) {
10         return new Shift(f, shiftX, shiftY);
11     }
12
13     public static Function scale(Function f, double scaleX, double scaleY) {
14         return new Scale(f, scaleX, scaleY);
15     }
16
17     public static Function power(Function f, double power) {
18         return new Power(f, power);
19     }
20
21     public static Function sum(Function f1, Function f2) {
22         return new Sum(f1, f2);
23     }
24
25     public static Function mult(Function f1, Function f2) {
26         return new Mult(f1, f2);
27     }
28
29     public static Function composition(Function f1, Function f2) {
30         return new Composition(f1, f2);
31     }
32 }
```

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.

```
1 package functions;
2
3 import java.io.*;
4
5 public class TabulatedFunctions {
6     // Приватный конструктор
7     private TabulatedFunctions() {}
8
9     // Задание 6: метод табулирования
10    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
11        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
12            throw new IllegalArgumentException("Tabulation interval is outside function domain");
13        }
14
15        double[] values = new double[pointsCount];
16        double step = (rightX - leftX) / (pointsCount - 1);
17
18        for (int i = 0; i < pointsCount; i++) {
19            double x = leftX + i * step;
20            values[i] = function.getFunctionValue(x);
21        }
22
23        // Используем ArrayTabulatedFunction по умолчанию
24        return new ArrayTabulatedFunction(leftX, rightX, values);
25    }
}
```

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 указанного 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 возникнуть, 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) = 6.123233995736766E-17
```

Проверка табулирования

2. Табулирование Sin и Cos:

```
Табулированный Sin в точке p/2: 0.984807753012208  
Табулированный Cos в точке p/2: 5.551115123125783E-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 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     }
```