

ЛАБОРАТОРНАЯ РАБОТА №4

Курс: Объектно-ориентированное
программирование.

Студент: Кузнецов Эрик Витальевич
Группа: 6204-010302D
Преподаватель: Борисов Дмитрий Сергеевич

Задание 1

Ход выполнения работы:

- 1) В класс ArrayTabulatedFunction добавляем конструктор, принимающий массив точек FunctionPoint[]
- 2) Проверяем, что количество точек не менее 2, иначе выбрасываем IllegalArgumentException
- 3) Проверяем упорядоченность точек по значению X, иначе выбрасываем IllegalArgumentException
- 4) Создаем копии точек для обеспечения инкапсуляции
- 5) В класс LinkedListTabulatedFunction добавляем аналогичный конструктор.

```
        }

    // ЗАДАНИЕ 1
    public ArrayTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

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

        this.pointsCount = points.length;
        this.points = new FunctionPoint[pointsCount + 10];
        for (int i = 0; i < pointsCount; i++) {
            this.points[i] = new FunctionPoint(points[i]);
        }
    }
```

```
public LinkedListTabulatedFunction(FunctionPoint[] points) { //новый констурктор (задание 1)
    this();
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

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

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

Рисунок 1,2-изображение с конечным результатом для задания 1.

Задание 2.

Ход выполнения работы:

1) Создаем интерфейс Function в пакете functions

2) Добавляем методы в интерфейс:

getLeftDomainBorder() - возвращает левую границу области определения

getRightDomainBorder() - возвращает правую границу области определения

getFunctionValue(double x) - возвращает значение функции в точке

3) Убираем эти методы из интерфейса TabulatedFunction

4) Делаем так, чтобы TabulatedFunction расширял интерфейс Function

5) Реализуем эти методы в классах ArrayTabulatedFunction и LinkedListTabulatedFunction

The screenshot shows a code editor with two tabs open:

- Function.java**:

```
package functions;

public interface Function {
    double getLeftDomainBorder();
    double getRightDomainBorder();
    double getFunctionValue(double x);
}
```
- TabulatedFunction.java**:

```
package functions;

public interface TabulatedFunction extends Function{
    // Методы работы с точками
    int getPointsCount(); // Метод, возвращающий количество точек
    FunctionPoint getPoint(int index); // Метод, возвращающий ссылку на объект по индексу
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; // Метод, заменяющий точку по индексу на заданную
    double getPointX(int index); // Возвращает X точки по указанному индексом
    void setPointX(int index, double x) throws InappropriateFunctionPointException; // Метод, изменяющий абсциссу точки по индексу
    double getPointY(int index);
    void setPointY(int index, double y);

    // Методы изменения количества точек (удаление или добавление)
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
    void printTabulatedFunction(); // Вывод в консоль
}
```

Рисунок 3,4-Снимок Экрана с конечным результатом для задания 2

Задание 3.

Ход выполнения работы:

- 1) Создаем пакет functions.basic для аналитических функций
- 2) Создаем класс Exp, реализующий интерфейс Function:
getLeftDomainBorder() возвращает Double.NEGATIVE_INFINITY
getRightDomainBorder() возвращает Double.POSITIVE_INFINITY
getFunctionValue() использует Math.exp()
- 3) Создаем класс Log, реализующий интерфейс Function:
Конструктор принимает основание логарифма
Проверяем основание > 0 и $\neq 1$
getFunctionValue() использует Math.log()
- 4) Создаем абстрактный класс TrigonometricFunction с реализацией методов границ области определения
- 5) Создаем классы Sin, Cos, Tan, наследующие от TrigonometricFunction:
getFunctionValue() использует Math.sin(), Math.cos(), Math.tan()

The screenshot shows a Java code editor with four tabs open, each containing a class definition:

- Exp.java**:

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Exp implements Function {
6     public double getLeftDomainBorder() {
7         return Double.NEGATIVE_INFINITY;
8     }
9
10    public double getRightDomainBorder() {
11        return Double.POSITIVE_INFINITY;
12    }
13
14    public double getFunctionValue(double x) {
15        return Math.exp(x);
16    }
17}
```
- Sin.java**:

```
1 package functions.basic;
2
3 public class Sin extends TrigonometricFunction {
4     public double getFunctionValue(double x) {
5         return Math.sin(x);
6     }
7 }
8
```
- Cos.java**:

```
1 package functions.basic;
2
3 public class Cos extends TrigonometricFunction {
4     public double getFunctionValue(double x) {
5         return Math.cos(x);
6     }
7 }
8
```
- Tan.java**:

```
1 package functions.basic;
2
3 public class Tan extends TrigonometricFunction {
4     public double getFunctionValue(double x) {
5         return Math.tan(x);
6     }
7 }
8
```

The screenshot shows a Java code editor with two files open:

- Log.java** (Left Editor):

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Log implements Function {
6     private double base;
7
8     public Log(double base) {
9         if (base <= 0 || base == 1) {
10             throw new IllegalArgumentException("Основание логарифма должно быть >1 && !=1");
11         }
12         this.base = base;
13     }
14
15     public double getLeftDomainBorder() {
16         return 0;
17     }
18
19     public double getRightDomainBorder() {
20         return Double.POSITIVE_INFINITY;
21     }
22
23     public double getFunctionValue(double x) {
24         if (x <= 0) {
25             return Double.NaN;
26         }
27         return Math.log(x) / Math.log(base);
28     }
29 }
30
```
- TrigonometricFunction.java** (Right Editor):

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public abstract class TrigonometricFunction implements Function {
6     public double getLeftDomainBorder() {
7         return Double.NEGATIVE_INFINITY;
8     }
9
10    public double getRightDomainBorder() {
11        return Double.POSITIVE_INFINITY;
12    }
13 }
14
```

Рисунок 5,6,7-Снимок Экрана с конечным результатом для задания 3

Задание 4

Ход выполнения работы:

1) Создаем пакет functions.meta для комбинированных функций

2) Создаем класс Sum для суммы двух функций:

Конструктор принимает две функции

Область определения - пересечение областей исходных функций

3) Создаем класс Power для возведения функции в степень:

Конструктор принимает функцию и степень

Область определения совпадает с исходной функцией

4) Создаем класс Scale для масштабирования:

Конструктор принимает функцию и коэффициенты масштабирования

Масштабирует область определения и значения функции

5) Создаем класс Shift для сдвига:

Конструктор принимает функцию и величины сдвига

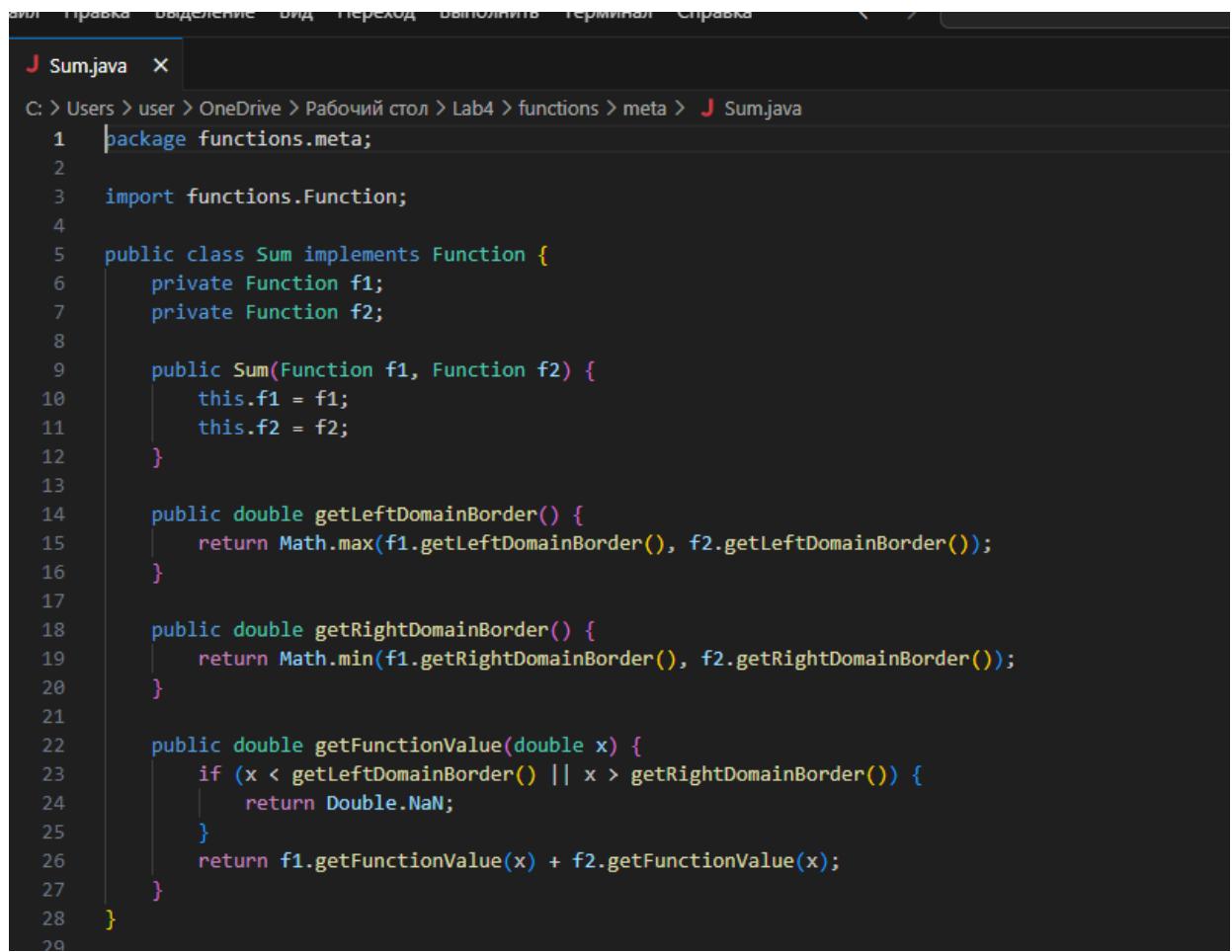
6) Создаем класс Composition для композиции функций:

Конструктор принимает две функции

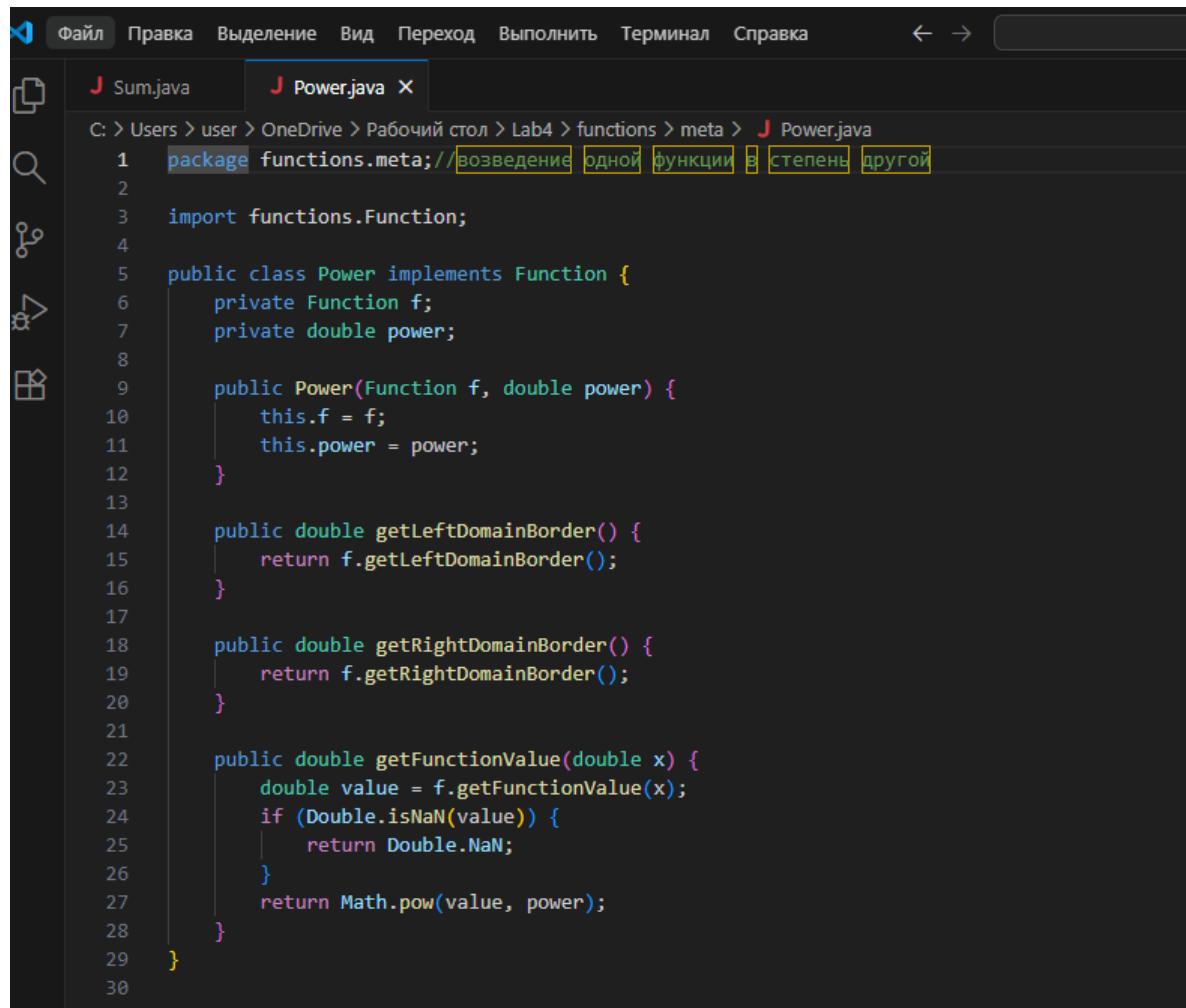
Область определения совпадает с первой функцией

7) Создаем класс Mult для произведения двух функций:

Аналогично классу Sum



```
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     public double getLeftDomainBorder() {
15         return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
16     }
17
18     public double getRightDomainBorder() {
19         return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
20     }
21
22     public double getFunctionValue(double x) {
23         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
24             return Double.NaN;
25         }
26         return f1.getFunctionValue(x) + f2.getFunctionValue(x);
27     }
28 }
29
```



```
1 package functions.meta; //возведение одной функции в степень другой
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     public double getLeftDomainBorder() {
15         return f.getLeftDomainBorder();
16     }
17
18     public double getRightDomainBorder() {
19         return f.getRightDomainBorder();
20     }
21
22     public double getFunctionValue(double x) {
23         double value = f.getFunctionValue(x);
24         if (Double.isNaN(value)) {
25             return Double.NaN;
26         }
27         return Math.pow(value, power);
28     }
29 }
30
```

```
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    public double getLeftDomainBorder() {
17        return f.getLeftDomainBorder() * scaleX;
18    }
19
20    public double getRightDomainBorder() {
21        return f.getRightDomainBorder() * scaleX;
22    }
23
24    public double getFunctionValue(double x) {
25        double scaledX = x / scaleX;
26        if (scaledX < f.getLeftDomainBorder() || scaledX > f.getRightDomainBorder()) {
27            return Double.NaN;
28        }
29        return f.getFunctionValue(scaledX) * scaleY;
30    }
31}
32
```

```
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    public double getLeftDomainBorder() {
17        return f.getLeftDomainBorder() + shiftX;
18    }
19
20    public double getRightDomainBorder() {
21        return f.getRightDomainBorder() + shiftX;
22    }
23
24    public double getFunctionValue(double x) {
25        double shiftedX = x - shiftX;
26        if (shiftedX < f.getLeftDomainBorder() || shiftedX > f.getRightDomainBorder()) {
27            return Double.NaN;
28        }
29        return f.getFunctionValue(shiftedX) + shiftY;
30    }
31}
32
```

Файл Правка Выделение Вид Переход Выполнить Терминал Справка ← →

J Shift.java J Composition.java X

C: > Users > user > OneDrive > Рабочий стол > Lab4 > functions > meta > J Composition.java

```
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    public double getLeftDomainBorder() {
15        return f1.getLeftDomainBorder();
16    }
17
18    public double getRightDomainBorder() {
19        return f1.getRightDomainBorder();
20    }
21
22    public double getFunctionValue(double x) {
23        double innerValue = f1.getFunctionValue(x);
24        if (Double.isNaN(innerValue)) {
25            return Double.NaN;
26        }
27        return f2.getFunctionValue(innerValue);
28    }
29
30 }
```

Файл Правка Выделение Вид Переход Выполнить Терминал Справка ← →

J Mult.java X

C: > Users > user > OneDrive > Рабочий стол > Lab4 > functions > meta > J Mult.java

```
1 package functions.meta; //Произведение двух функций
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    public double getLeftDomainBorder() {
15        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
16    }
17
18    public double getRightDomainBorder() {
19        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
20    }
21
22    public double getFunctionValue(double x) {
23        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
24            return Double.NaN;
25        }
26        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
27    }
28
29 }
```

Рисунок 8-13-Снимок Экрана с конечным результатом для задания 4.

Задание 5

Ход выполнения работы:

1) Создаем класс Functions в пакете functions

Делаем конструктор приватным (утилитный класс)

Добавляем статические методы:

shift() - возвращает функцию сдвига

scale() - возвращает масштабированную функцию

power() - возвращает функцию в степени

sum() - возвращает сумму функций

mult() - возвращает произведение функций

composition() - возвращает композицию функций

2) В методах используем соответствующие классы из пакета functions.meta.

The screenshot shows a Java code editor with the file `Functions.java` open. The code defines a static class `Functions` containing methods for shifting, scaling, raising to power, summing, multiplying, and composing functions. A private constructor prevents instantiation.

```
1 package functions;
2
3 import functions.meta.*;
4
5 public final class Functions {
6     // Приватный конструктор чтобы запретить создание экземпляров
7     private Functions() {
8         throw new AssertionError("Нельзя создать экземпляр класса Functions");
9     }
10
11    public static Function shift(Function f, double shiftX, double shiftY) {
12        return new Shift(f, shiftX, shiftY);
13    }
14
15    public static Function scale(Function f, double scaleX, double scaleY) {
16        return new Scale(f, scaleX, scaleY);
17    }
18
19    public static Function power(Function f, double power) {
20        return new Power(f, power);
21    }
22
23    public static Function sum(Function f1, Function f2) {
24        return new Sum(f1, f2);
25    }
26
27    public static Function mult(Function f1, Function f2) {
28        return new Mult(f1, f2);
29    }
30
31    public static Function composition(Function f1, Function f2) {
32        return new Composition(f1, f2);
33    }
34}
35
```

Рисунок 14-Снимок Экрана с конечным результатом для задания 5.

Задание 6

Ход выполнения работы:

1) Создаем класс TabulatedFunctions в пакете functions

Делаем конструктор приватным (утилитный класс)

2) Реализуем метод tabulate():

Проверяем, что границы табулирования входят в область определения функции. Проверяем, что количество точек не менее 2

4) Создаем массив точек с равномерным шагом. Вычисляем значения функции в этих точках. Возвращаем табулированную функцию

```
TabulatedFunctions.java
C: > Users > user > OneDrive > Рабочий стол > Lab4 > functions > TabulatedFunctions.java
1 package functions;
2
3 import java.io.*;
4 import java.util.StringTokenizer;
5
6 public final class TabulatedFunctions {
7     // Приватный конструктор чтобы запретить создание экземпляров
8     private TabulatedFunctions() {
9         throw new AssertionError("Нельзя создать экземпляр класса TabulatedFunctions");
10    }
11
12    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
13        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
14            throw new IllegalArgumentException("Границы табулирования выходят за область определения функции");
15        }
16        if (pointsCount < 2) {
17            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
18        }
19
20        // Создаем массив точек
21        FunctionPoint[] points = new FunctionPoint[pointsCount];
22        double step = (rightX - leftX) / (pointsCount - 1);
23
24        for (int i = 0; i < pointsCount; i++) {
25            double x = leftX + i * step;
26            double y = function.getFunctionValue(x);
27            points[i] = new FunctionPoint(x, y);
28        }
29
30        // Возвращаем табулированную функцию (пока используем ArrayTabulatedFunction)
31        return new ArrayTabulatedFunction(points);
32    }
33    public static double getFunctionValue(TabulatedFunction function, double x) {
34        return function.getFunctionValue(x);
35    }
36
37    public static double getLeftDomainBorder(TabulatedFunction function) {
38        return function.getLeftDomainBorder();
39    }
40
41    public static double getRightDomainBorder(TabulatedFunction function) {
42        return function.getRightDomainBorder();
43    }
44}
```

Рисунок 15-Снимок Экрана с конечным результатом для задания 6.

Задание 7

Ход выполнения работы:

1) Добавляем в TabulatedFunctions метод

outputTabulatedFunction():

Использует DataOutputStream для записи в байтовый

поток

Записывает количество точек и координаты (x,y) каждой

точки

2)Добавляем метод inputTabulatedFunction():

Использует DataInputStream для чтения из байтового

потока

Читает количество точек и создает массив точек

3)Добавляем метод writeTabulatedFunction():

Использует PrintWriter для записи в символьный поток

Записывает данные в текстовом формате с пробелами-
разделителями

4)Добавляем метод readTabulatedFunction():

Использует StreamTokenizer для чтения из символьного
потока

Парсит текстовые данные и создает табулированную
функцию

Обрабатываем IOException, обрачивая в
RuntimeException

```

// Метод вывода в байтовый поток
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) {
    try (DataOutputStream dos = new DataOutputStream(out)) {
        dos.writeInt(function.getPointsCount());
        for (int i = 0; i < function.getPointsCount(); i++) {
            FunctionPoint point = function.getPoint(i);
            dos.writeDouble(point.getX());
            dos.writeDouble(point.getY());
        }
    } catch (IOException e) {
        // Пробрасываем RuntimeException, так как IOException - проверяемое исключение
        throw new RuntimeException("Ошибка при выводе функции в поток", e);
    }
}

// Метод ввода из байтового потока
public static TabulatedFunction inputTabulatedFunction(InputStream in) {
    try (DataInputStream dis = new DataInputStream(in)) {
        int pointsCount = dis.readInt();
        FunctionPoint[] points = new FunctionPoint[pointsCount];

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

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

// Метод записи в символьный поток
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) {
    try (PrintWriter writer = new PrintWriter(out)) {
        writer.print(function.getPointsCount() + " ");
        for (int i = 0; i < function.getPointsCount(); i++) {
            FunctionPoint point = function.getPoint(i);
            writer.print(point.getX() + " " + point.getY() + " ");
        }
    }
    // PrintWriter не бросает IOException в методах print/println
}

```

```

// Метод чтения из символьного потока
public static TabulatedFunction readTabulatedFunction(Reader in) {
    try {
        StreamTokenizer tokenizer = new StreamTokenizer(in);
        tokenizer.parseNumbers();

        // Читаем количество точек
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
            throw new RuntimeException("ожидалось количество точек");
        }
        int pointsCount = (int) tokenizer.nval;

        FunctionPoint[] points = new FunctionPoint[pointsCount];

        for (int i = 0; i < pointsCount; i++) {
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new RuntimeException("ожидалась координата x");
            }
            double x = tokenizer.nval;

            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new RuntimeException("ожидалась координата y");
            }
            double y = tokenizer.nval;

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

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

```

Рисунок 16,17-Снимок экрана с конечным результатом для задания 7.

Задание 8

Ход выполнения работы:

- 1) Создаем объекты Sin и Cos, выводим их значения на отрезке $[0, \pi]$
- 2) Создаем табулированные аналоги с помощью `TabulatedFunctions.tabulate()`
- 3) Сравниваем значения исходных и табулированных функций
- 4) Создаем функцию суммы квадратов табулированных синуса и косинуса
- 5) Исследуем влияние количества точек на точность

6) Тестируем экспоненту:

Создаем табулированную экспоненту

Записываем в текстовый файл с помощью
`writeTabulatedFunction()`

Читаем из файла с помощью `readTabulatedFunction()`

Сравниваем исходную и считанную функции

7) Тестируем логарифм:

Создаем табулированный логарифм

Записываем в бинарный файл с помощью
`outputTabulatedFunction()`

Читаем из файла с помощью `inputTabulatedFunction()`

Сравниваем исходную и считанную функции

8) Анализируем файлы разных форматов, сравниваем их преимущества и недостатки

```
== ТЕСТИРОВАНИЕ ВСЕХ ФУНКЦИЙ ==

1. ТЕСТИРОВАНИЕ SIN И COS
Sin на отрезке [0, 3.14]:
sin(0,0)=0,000000      sin(0,1)=0,099833      sin(0,2)=0,198669      sin(0,3)=0,295520      sin(0,4)=0,389418
sin(0,5)=0,479426      sin(0,6)=0,564642      sin(0,7)=0,644218      sin(0,8)=0,717356      sin(0,9)=0,783327
sin(1,0)=0,841471      sin(1,1)=0,891207      sin(1,2)=0,932039      sin(1,3)=0,963558      sin(1,4)=0,985450
sin(1,5)=0,997495      sin(1,6)=0,999574      sin(1,7)=0,991665      sin(1,8)=0,973848      sin(1,9)=0,946300
sin(2,0)=0,90297      sin(2,1)=0,863289      sin(2,2)=0,808496      sin(2,3)=0,745785      sin(2,4)=0,675463
sin(2,5)=0,598472      sin(2,6)=0,515501      sin(2,7)=0,427380      sin(2,8)=0,334988      sin(2,9)=0,239249
sin(3,0)=0,141120      sin(3,1)=0,041581

Cos на отрезке [0, 3.14]:
cos(0,0)=1,000000      cos(0,1)=0,995094      cos(0,2)=0,980067      cos(0,3)=0,955336      cos(0,4)=0,921061
cos(0,5)=0,877583      cos(0,6)=0,825336      cos(0,7)=0,764842      cos(0,8)=0,696787      cos(0,9)=0,621610
cos(1,0)=0,540382      cos(1,1)=0,453596      cos(1,2)=0,362358      cos(1,3)=0,267499      cos(1,4)=0,169967
cos(1,5)=0,070737      cos(1,6)=0,029200      cos(1,7)=0,128844      cos(1,8)=0,227202      cos(1,9)=0,323290
cos(2,0)=0,416147      cos(2,1)=0,504846      cos(2,2)=0,588501      cos(2,3)=0,666276      cos(2,4)=0,737394
cos(2,5)=0,801144      cos(2,6)=0,856889      cos(2,7)=0,904072      cos(2,8)=0,942222      cos(2,9)=0,970958

2. ТАБУЛИРОВАННЫЕ АНАЛОГИ
Сравнение исходного и табулированного Sin:
x=0,0: Orig=0,000000 Tabulated=0,000000 difference=0,000000
x=0,3: Orig=0,295520 Tabulated=0,293945 difference=0,001576
x=0,6: Orig=0,564642 Tabulated=0,558234 difference=0,006409
x=0,9: Orig=0,783327 Tabulated=0,771888 difference=0,011439
x=1,2: Orig=0,932039 Tabulated=0,918022 difference=0,014017
x=1,5: Orig=0,997495 Tabulated=0,984888 difference=0,012687
x=1,8: Orig=0,973848 Tabulated=0,966284 difference=0,007644
x=2,1: Orig=0,946300 Tabulated=0,924411 difference=0,000768
x=2,4: Orig=0,675463 Tabulated=0,670582 difference=0,004881
x=2,7: Orig=0,427380 Tabulated=0,421745 difference=0,005635
x=3,0: Orig=0,141120 Tabulated=0,138735 difference=0,002385

Сравнение исходного и табулированного Cos:
x=0,0: Orig=1,000000 Tabulated=0,000000 difference=1,000000
x=0,3: Orig=0,955336 Tabulated=0,293945 difference=0,661392
x=0,6: Orig=0,825336 Tabulated=0,558234 difference=0,267102
x=0,9: Orig=0,621610 Tabulated=0,771888 difference=0,150278
x=1,2: Orig=0,362358 Tabulated=0,918022 difference=0,555664
x=1,5: Orig=0,070737 Tabulated=0,984888 difference=0,914071
x=1,8: Orig=0,227202 Tabulated=0,966204 difference=-1,193406
x=2,1: Orig=0,504846 Tabulated=0,862441 difference=-1,367287
x=2,4: Orig=0,737394 Tabulated=0,670582 difference=-1,407976
x=2,7: Orig=0,942222 Tabulated=0,421745 difference=-1,325817
x=3,0: Orig=0,989992 Tabulated=0,138735 difference=-1,128727

x=0,1: Orig=0,099833 Tabulated=0,097982 difference=0,001852
x=0,4: Orig=0,389418 Tabulated=0,385907 difference=0,003512
x=0,7: Orig=0,644218 Tabulated=0,643982 difference=0,000235
x=1,0: Orig=0,841471 Tabulated=0,835841 difference=0,005630
x=1,3: Orig=0,963558 Tabulated=0,952051 difference=0,011508
x=1,6: Orig=0,999574 Tabulated=0,984888 difference=0,014766
x=1,9: Orig=0,946300 Tabulated=0,932175 difference=0,014125
x=2,2: Orig=0,880496 Tabulated=0,798488 difference=0,010008
x=2,5: Orig=0,598472 Tabulated=0,594072 difference=0,004401
x=2,8: Orig=0,334988 Tabulated=0,334698 difference=0,000290
x=3,1: Orig=0,041581 Tabulated=0,040753 difference=0,000828

x=0,2: Orig=0,198669 Tabulated=0,195963 difference=0,002706
x=0,5: Orig=0,479426 Tabulated=0,472070 difference=0,007355
x=0,8: Orig=0,717356 Tabulated=0,707935 difference=0,009421
x=1,1: Orig=0,891207 Tabulated=0,883993 difference=0,007214
x=1,4: Orig=0,984888 Tabulated=0,984888 difference=0,000642
x=1,7: Orig=0,991665 Tabulated=0,984888 difference=0,006857
x=2,0: Orig=0,909297 Tabulated=0,898147 difference=0,011151
x=2,3: Orig=0,745785 Tabulated=0,734535 difference=0,011170
x=2,6: Orig=0,515501 Tabulated=0,507908 difference=0,007593
x=2,9: Orig=0,239249 Tabulated=0,236716 difference=0,002533

x=0,0: Orig=0,980067 Tabulated=0,195963 difference=0,784103
x=0,3: Orig=0,877583 Tabulated=0,472070 difference=0,405512
x=0,6: Orig=0,696707 Tabulated=0,707935 difference=0,011229
x=0,9: Orig=0,453596 Tabulated=0,883993 difference=0,438397
x=1,2: Orig=0,169967 Tabulated=0,984888 difference=0,814841
x=1,5: Orig=0,029200 Tabulated=0,984888 difference=-1,014007
x=1,8: Orig=0,323290 Tabulated=0,932175 difference=-1,255465
x=2,1: Orig=0,588501 Tabulated=0,798488 difference=-1,386989
x=2,4: Orig=0,801144 Tabulated=0,594072 difference=-1,395215
x=2,7: Orig=0,942222 Tabulated=0,334698 difference=-1,276920
x=3,1: Orig=0,999135 Tabulated=0,040753 difference=-1,039888

3. СУММА КВАДРАТОВ SIN И COS
(sin(x))**2 + (cos(x))**2 на отрезке [0, 3.14]:
sum(0,0)=1,000000      sum(0,1)=0,975345      sum(0,2)=0,970488      sum(0,3)=0,985429      sum(0,4)=0,984968
sum(0,5)=0,970398      sum(0,6)=0,975624      sum(0,7)=0,999358      sum(0,8)=0,975073      sum(0,9)=0,970586
sum(1,0)=0,985897      sum(1,1)=0,984515      sum(1,2)=0,970314      sum(1,3)=0,975910      sum(1,4)=0,998723
sum(1,5)=0,974808      sum(1,6)=0,970691      sum(1,7)=0,986371      sum(1,8)=0,984068      sum(1,9)=0,970237
sum(2,0)=0,976293      sum(2,1)=0,998894      sum(2,2)=0,974549      sum(2,3)=0,976882      sum(2,4)=0,986852
sum(2,5)=0,983628      sum(2,6)=0,970167      sum(2,7)=0,976503      sum(2,8)=0,997473      sum(2,9)=0,974298

4. ВЛИЯНИЕ КОЛИЧСТВА ТОЧЕК НА ТОЧНОСТЬ
Точек: 5, максимальная ошибка: 0,146396
Точек: 10, максимальная ошибка: 0,029833
Точек: 20, максимальная ошибка: 0,006817
Точек: 50, максимальная ошибка: 0,001026

5. ТЕСТИРОВАНИЕ С ЭКСПОНЕНТОЙ (текстовый файл)
Экспонента записана в exp_function.txt
Экспонента прочитана из exp_function.txt
Сравнение исходной и считанной экспоненты:
x=0,0: Orig=0,000000 C=1,000000 Совпадают
x=1,0: Orig=2,718282 C=2,718282 Совпадают
x=2,0: Orig=7,389056 C=7,389056 Совпадают
x=3,0: Orig=20,085537 C=20,085537 Совпадают
x=4,0: Orig=54,598150 C=54,598150 Совпадают
x=5,0: Orig=148,413150 C=148,413150 Совпадают
x=6,0: Orig=403,428793 C=403,428793 Совпадают
x=7,0: Orig=1096,633158 C=1096,633158 Совпадают
x=8,0: Orig=2980,957987 C=2980,957987 Совпадают
x=9,0: Orig=8103,083928 C=8103,083928 Совпадают
x=10,0: Orig=22026,465795 C=22026,465795 Совпадают

6. ТЕСТИРОВАНИЕ С ЛОГАРИФМОМ (бинарный файл)
Логарифм записан в log_function.bin
Логарифм прочитан из log_function.bin
Сравнение исходной и считанной логарифма:
x=0,1: Orig=-2,302585 C=-2,302585 Совпадают
x=1,1: Orig=0,092705 C=0,092705 Совпадают
x=2,1: Orig=0,740233 C=0,740233 Совпадают
x=3,1: Orig=1,130147 C=1,130147 Совпадают
x=4,1: Orig=1,409999 C=1,409999 Совпадают
x=5,1: Orig=1,628429 C=1,628429 Совпадают
x=6,1: Orig=1,807603 C=1,807603 Совпадают
x=7,1: Orig=1,959502 C=1,959502 Совпадают
x=8,1: Orig=2,091344 C=2,091344 Совпадают
x=9,1: Orig=2,207812 C=2,207812 Совпадают
```

7. АНАЛИЗ ФАЙЛОВ РАЗНЫХ ФОРМАТОВ

Текстовый файл экспоненты:

Размер: 236 байт

Формат: текстовый

Содержимое:

11 0.0 1.0 2.718281828459045 2.0 7.38905609893065 3.0 20.085536923187668 4.0 54.598150033144236 5.0 148.4131591025766 6.0 403.4287934927351 7.0 1896.6331584284585 8.0 2980.95798704

Преимущества: человекочитаемый, легко отлаживать, совместим между системами

Недостатки: больший размер, медленнее парсинг, нет типизации данных

Бинарный файл экспоненты:

Размер: 188 байт

Формат: бинарный

Первые 58 байт (hex):

00 00 00 00 00 00 00 00 00 00 3F F0 00 00 00 00 00 00 00 00 40 05 BF 0A 8B 14 57 69 40 00 00 00 00 00 00 00 40 1D 8E 64 B8 D4

Преимущества: компактный размер, быстрая запись/чтение, эффективное использование памяти

Недостатки: не человекочитаемый, зависит от архитектуры, сложнее отлаживать

Текстовый файл логарифма:

Размер: 292 байт

Формат: текстовый

Содержимое:

11 0.1 -2.3025850929940455 1.09 0.08617769624105241 2.08 0.7323678937132266 3.07 1.1216775615991057 4.06 1.4011829736136412 5.05 1.6193882432872684 6.0399999999999999 1.7984040119467235

0716202463 10.0 2.302585092994046

Преимущества: человекочитаемый, легко отлаживать, совместим между системами

Недостатки: больший размер, медленнее парсинг, нет типизации данных

Бинарный файл логарифма:

Размер: 188 байт

Формат: бинарный

Первые 58 байт (hex):

00 00 00 00 3F B9 99 99 99 99 9A C0 02 6B B1 BB B5 55 15 3F F1 70 A3 D7 0A 3D 71 3F B6 0F BD D2 FF FC 37 40 00 A3 D7 0A 3D 70 A4 3F E7 6F 8E CB 04

Преимущества: компактный размер, быстрая запись/чтение, эффективное использование памяти

Недостатки: не человекочитаемый, зависит от архитектуры, сложнее отлаживать

8. СРАВНЕНИЕ ТЕКСТОВОГО И БИНАРНОГО ФОРМАТОВ

СРАВНЕНИЕ ФОРМАТОВ ХРАНЕНИЯ:

Формат	Размер	Человекочитаем.	Скорость
Текстовый	236 байт	Да	Медленно
Бинарный	188 байт	Нет	Быстро
Serializable	236 байт	Нет	Средне

ВЫВОДЫ:

Текстовый формат - лучший для отладки и совместимости

Бинарный формат - лучший для производительности и размера

Serializable - удобен для сложных объектов, но зависит от версии Java

Рисунок 18-20-Снимок экрана с конечным результатом для задания 8.

Задание 9

Ход выполнения работы:

1) Делаем классы `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` сериализуемыми

2) Добавляем `implements Serializable` и `serialVersionUID`

В `ArrayTabulatedFunction` реализуем интерфейс `Externalizable`:

`writeExternal()` записывает количество точек и координаты

`readExternal()` читает данные и восстанавливает объект

3) Тестируем сериализацию:

4) Создаем композицию логарифма и экспоненты

5) Табулируем композицию

6) Сериализуем в файл с помощью `ObjectOutputStream`

7) Десериализуем из файла с помощью `ObjectInputStream`

8) Сравниваем исходную и десериализованную функции

9) Анализируем файл сериализации, сравниваем с другими форматами

```
9. ТЕСТИРОВАНИЕ СЕРИАЛИЗАЦИИ
Функция сериализована в serializable_function.ser
Функция десериализована из serializable_function.ser
Сравнение исходной и десериализованной функции:
x=0,1: Orig=0,100000 D=0,100000 Совпадает      x=1,1: Orig=1,100000 D=1,100000 Совпадает      x=2,1: Orig=2,100000 D=2,100000 Совпадает
x=5,1: Orig=5,100000 D=5,100000 Совпадает      x=6,1: Orig=6,100000 D=6,100000 Совпадает      x=7,1: Orig=7,100000 D=7,100000 Совпадает

Файл сериализации (Serializable):
Размер: 236 байт
Формат: сериализованный объект
Размер файла: 236 байт
Преимущества: сохраняет структуру объектов, удобен для сложных объектов, автоматическая сериализация
Недостатки: не человекочитаемый, зависит от версии Java, больший размер чем бинарный
PS C:\Users\user\OneDrive\Рабочий стол\Lab4> █
```

Рисунок 21-Снимок экрана с конечным результатом для задания 9.