Лабораторная работа №4

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

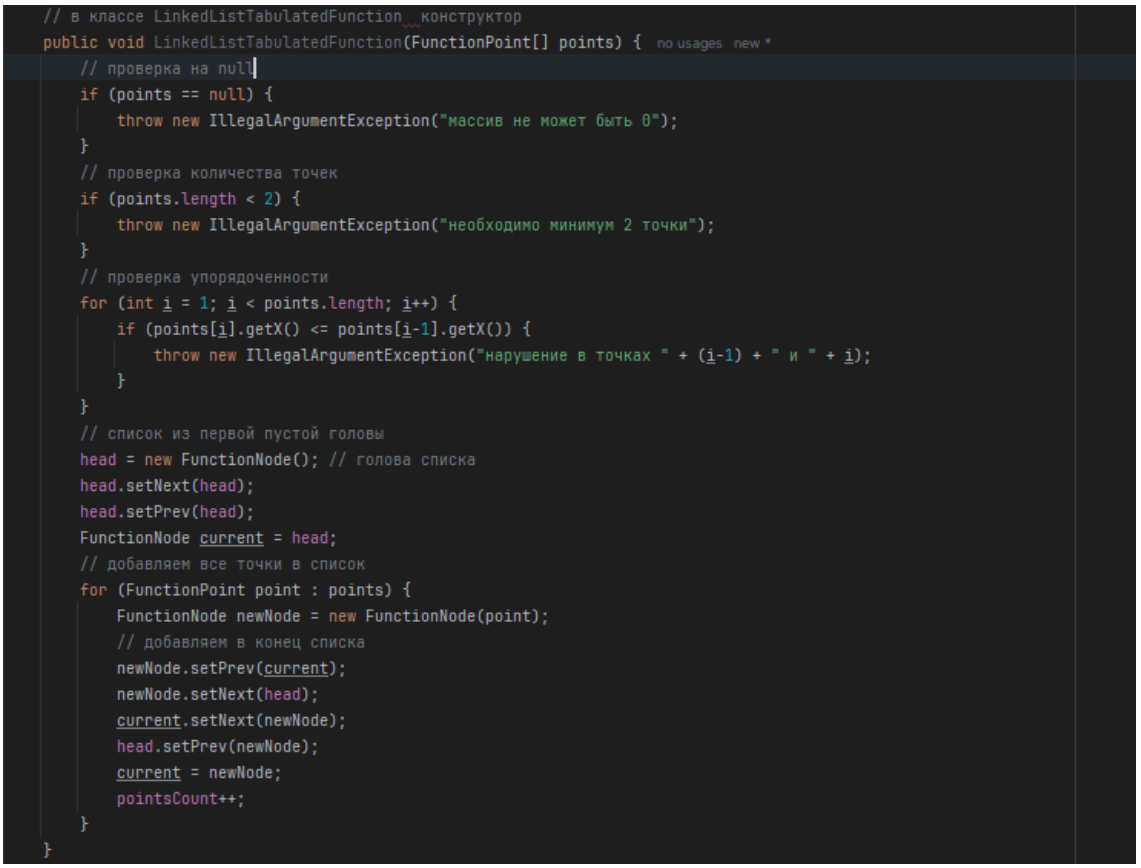
Ход выполнения

Задание 1

В рамках первого задания требуется в классах ArrayTabulatedFunction и LinkedListTabulatedFunction добавить конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint. Если точек задано меньше двух, или если точки в массиве не упорядочены по значению абсциссы, конструкторы должны выбрасывать исключение IllegalArgumentException. Для этого в классе ArrayTabulatedFunction создадим конструктор ArrayTabulatedFunction, который принимает массив FunctionPoint и выполняет необходимые проверки: сначала проверяет, что количество точек не меньше двух, затем проверяет упорядоченность точек по возрастанию значения X. Если какое-либо из условий нарушено, конструктор выбрасывает исключение IllegalArgumentException с соответствующим сообщением об ошибке. После успешного прохождения проверок конструктор создает внутренний массив для хранения точек с небольшим запасом и копирует в него все переданные точки, устанавливая счетчик точек равным размеру входного массива. Аналогичный конструктор добавляется в класс LinkedListTabulatedFunction.

```java
// в классе ArrayTabulatedFunction добавляем массив чисел
public ArrayTabulatedFunction(FunctionPoint[] points) {  2 usages  new *
    // проверка на null
    if (points == null) {
        throw new IllegalArgumentException("массив не может быть 0");
    }

    // проверка количества точек
    if (points.length < 2) {
        throw new IllegalArgumentException("необходимо минимум 2 точки");
    }

    // создание копий
    pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount + 10]; // с запасом

    // копируем первую точку
    this.points[0] = new FunctionPoint(points[0]);

    // копируем остальные и проверяем упорядоченность
    for (int i = 1; i < pointsCount; i++) {
        // проверка на то, что x[i] > x[i-1] чтобы шли по попорядку
        if (points[i].getX() <= points[i-1].getX()) {
            throw new IllegalArgumentException(" нарушение в точках " + (i-1) + " и " + i );
        }
        // инкапсуляция
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

Рис. 1



```
// в классе LinkedListTabulatedFunction конструктор
public void LinkedListTabulatedFunction(FunctionPoint[] points) {  no usages  new *
    // проверка на null
    if (points == null) {
        throw new IllegalArgumentException("массив не может быть 0");
    }
    // проверка количества точек
    if (points.length < 2) {
        throw new IllegalArgumentException("необходимо минимум 2 точки");
    }
    // проверка упорядоченности
    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() <= points[i-1].getX()) {
            throw new IllegalArgumentException("нарушение в точках " + (i-1) + " и " + i);
        }
    }
    // список из первой пустой головы
    head = new FunctionNode(); // голова списка
    head.setNext(head);
    head.setPrev(head);
    FunctionNode current = head;
    // добавляем все точки в список
    for (FunctionPoint point : points) {
        FunctionNode newNode = new FunctionNode(point);
        // добавляем в конец списка
        newNode.setPrev(current);
        newNode.setNext(head);
        current.setNext(newNode);
        head.setPrev(newNode);
        current = newNode;
        pointsCount++;
    }
}
```

Рис. 2

Задание 2

Во втором задании **в** пакете functions создаем интерфейс Function, описывающий функции одной переменной и содержащий следующие методы:

• public double getLeftDomainBorder() – возвращает значение левой границы области определения функции;

• public double getRightDomainBorder() – возвращает значение правой границы области определения функции;

• public double getFunctionValue(double x) – возвращает значение функции в заданной точке.

Также эти методы нужно исключить соответстенно из интерфейса TabulatedFunction и сделайте так, чтобы он расширял интерфейс Function.(Рис.3-4)

```java
package functions;

public interface Function {  no usages  3 implementations  new *
    public double getLeftDomainBorder();  no usages  2 implementations  new *
    public double getRightDomainBorder();  no usages  2 implementations  new *
    public double getFunctionValue(double x);  no usages  2 implementations  new *
}
```

Рис.3

```java
package functions;

public interface TabulatedFunction extends Function{  4 usages  2 implementations  new *
    // методы работы с точками
    int getPointsCount();  8 usages  2 implementations  new *
    FunctionPoint getPoint(int index);  no usages  2 implementations  new *
    double getPointX(int index);  6 usages  2 implementations  new *
    double getPointY(int index);  7 usages  2 implementations  new *
    void setPointY(int index, double y);  2 usages  2 implementations  new *
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;  no usages  2 implementations  new *
    void setPointX(int index, double x) throws InappropriateFunctionPointException;  1 usage  2 implementations  new *
    void deletePoint(int index);  4 usages  2 implementations  new *
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  2 usages  2 implementations  new *
}
```

Рис.4

Задание 3

В третьем задании создайте пакет functions.basic, в нём должны быть описаны классы ряда функций, заданных аналитически.

1)Создать в пакете публичный класс Exp, объекты которого должны вычислять значение экспоненты.

2)Создайть класс Log, объекты которого должны вычислять значение логарифма по заданному основанию.

3)Создайть класс TrigonometricFunction, реализующий интерфейс Function и описывающий методы получения границ области определения.

4)Создайте наследующие от него публичные классы Sin, Cos и Tan, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса.

(Рис 4-9)

```
package functions.basic;
//должен импортировать из Function
import functions.Function;

public class Exp implements Function {  no usages  new *
    public double getLeftDomainBorder() {  1 usage  new *
        return Double.NEGATIVE_INFINITY;  // экспонента (не понял сначала что такое посмотрел в интернете)
    }
    public double getRightDomainBorder() {  1 usage  new *
        return Double.POSITIVE_INFINITY;
    }

    public double getFunctionValue(double x) {  7 usages  new *
        return Math.exp(x);  // e^x
    }
}
```

```java
package functions.basic;

import functions.Function;

public class Log implements Function {  no usages  new *
    private double base;  // основание логарифма  3 usages

    public Log(double base) {  2 usages  new *
        // проверка логарифма(сам не додумался)
        if (base <= 0 || base == 1) {
            throw new IllegalArgumentException("основание логарифма >0 и не равно 1 должно быть");
        }
        this.base = base;
    }
    public double getBase() {  no usages  new *
        return base;
    }
    public double getLeftDomainBorder() {  1 usage  new *
        return 0;  // логарифм определен при x > 0
    }
    public double getRightDomainBorder() {  1 usage  new *
        return Double.POSITIVE_INFINITY;
    }
    public double getFunctionValue(double x) {  7 usages  new *
        if (x <= 0) {
            throw new IllegalArgumentException("Логарифм определен только при x > 0");
        }
        return Math.log(x) / Math.log(base);  // log_a(x) = ln(x)/ln(a)
    }
```

```java
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {  no usages  3 inheritors  new *
    public double getLeftDomainBorder() {  1 usage  new *
        return Double.NEGATIVE_INFINITY;  // тригонометрические функции определяются
    }
    public double getRightDomainBorder() {  1 usage  new *
        return Double.POSITIVE_INFINITY;
    }
    // как я понял нужно делать абстрактным чтобы не было ошибки
    public abstract double getFunctionValue(double x);  7 usages  3 implementations  new *
}
```

```java
package functions.basic;
//синус
public class Sin extends TrigonometricFunction {  no usages  new *
    public double getFunctionValue(double x) { return Math.sin(x); }
}
```

```java
package functions.basic;
// косинус
public class Cos extends TrigonometricFunction {  no usages  new *
    public double getFunctionValue(double x) {  new *
        return Math.cos(x);
    }
}
```

```java
package functions.basic;
// тангенс
public class Tan extends TrigonometricFunction {  no usages  new *
    public double getFunctionValue(double x) {  new *
        return Math.tan(x);
    }
}
```

Рис.4-9

Задание 4

В 4 задании создаем пакет functions.meta, в нём будут описаны классы функций, позволяющие комбинировать функции.

1)Создаем класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций.

2)Создаем класс Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций.

3)Создаем класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции.

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

5)Создаем класс Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат.

6)Создаем класс Composition, объекты которого описывают композицию двух исходных функций. (Рис 10-15)

va Function.java TabulatedFunction.java Exp.java Log.java TrigonometricF

```java
package functions.meta;

import functions.Function;

public class Mult implements Function {  no usages  new *
    // аналогичные наши переменные
    private Function f1, f2;  4 usages
    public Mult(Function f1, Function f2) {  2 usages  new *
        this.f1 = f1;
        this.f2 = f2;
    }
    public double getLeftDomainBorder() {  new *
        // пересечение областей определения
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    public double getRightDomainBorder() {  new *
        // пересечение областей определения
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    public double getFunctionValue(double x) {  new *
        // проверка, что x в области определения обеих функций
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("x вне области определения");
        }
        // возвращаем уже само произведение
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
}
```

```java
package functions.meta;

import functions.Function;

public class Sum implements Function {  no usages  new *
    // наши переменные
    private Function f1, f2;  4 usages

    public Sum(Function f1, Function f2) {  2 usages  new *
        this.f1 = f1;
        this.f2 = f2;
    }
    public double getLeftDomainBorder() {  4 usages  new *
        // пересечение областей определения
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }
    public double getRightDomainBorder() {  4 usages  new *
        // пересечение областей определения
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }
    public double getFunctionValue(double x) {  new *
        // проверка, что x в области определения обеих функций
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("x вне области определения"); // исключение в случае не того
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x); // возвращаем саму сумму уже
    }
}
```

```java
package functions.meta;

import functions.Function;

public class Scale implements Function {  no usages  new *
    // коэффициенты
    private Function f;  6 usages
    private double scaleX, scaleY;  4 usages

    public Scale(Function f, double scaleX, double scaleY) {  2 usages  new *
        this.f = f;
        this.scaleX = scaleX;
        this.scaleY = scaleY;
    }

    public double getLeftDomainBorder() {  new *
        // масштабирование области определения
        return f.getLeftDomainBorder() * scaleX;
    }

    public double getRightDomainBorder() { return f.getRightDomainBorder() * scaleX; }

    public double getFunctionValue(double x) {  new *
        // обратное масштабирование x, затем масштабирование y
        double originalX = x / scaleX;
        if (originalX < f.getLeftDomainBorder() || originalX > f.getRightDomainBorder()) {
            throw new IllegalArgumentException("x вне области определения"); // исключение
        }
        return f.getFunctionValue(originalX) * scaleY;
    }
}
```

```java
package functions.meta;

import functions.Function;

public class Power implements Function {  no usages  new *
    // значение и число
    private Function f;  4 usages
    private double power;  2 usages

    public Power(Function f, double power) {  2 usages  new *
        this.f = f;
        this.power = power;
    }
    public double getLeftDomainBorder() {  new *
        // пересечение областей определения
        return f.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {  new *
        // пересечение областей определения
        return f.getRightDomainBorder();
    }
    public double getFunctionValue(double x) {  new *
        // проверка, что x в области определения обеих функций
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("x вне области определения"); // исключение
        }
        return Math.pow(f.getFunctionValue(x), power); //возвращаем число в степени
    }
}
```

```java
package functions.meta;

import functions.Function;

public class Shift implements Function {  no usages  new *
    private Function f;  6 usages
    private double shiftX, shiftY;  4 usages
    public Shift(Function f, double shiftX, double shiftY) {  2 usages  new *

        this.f = f;
        this.shiftX = shiftX;
        this.shiftY = shiftY;
    }
    public double getLeftDomainBorder() { return f.getLeftDomainBorder() + shiftX; }
    public double getRightDomainBorder() { return f.getRightDomainBorder() + shiftX; }
    public double getFunctionValue(double x) {  new *
        double originalX = x - shiftX;
        if (originalX < f.getLeftDomainBorder() || originalX > f.getRightDomainBorder()) {
            throw new IllegalArgumentException("x вне области определения");// исключения
        }
        return f.getFunctionValue(originalX) + shiftY;
    }
}
```

```java
package functions.meta;

import functions.Function;

public class Composition implements Function {  no usages  new *
    private Function outer, inner;  4 usages
    public Composition(Function outer, Function inner) {  2 usages  new *
        this.outer = outer;
        this.inner = inner;
    }

    public double getLeftDomainBorder() {  new *
        return inner.getLeftDomainBorder();
    }
    public double getRightDomainBorder() {  new *
        return inner.getRightDomainBorder();
    }
    public double getFunctionValue(double x) {  new *
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            throw new IllegalArgumentException("x вне области определения");
        }
        double innerValue = inner.getFunctionValue(x);
        // проверка, что innerValue в области определения outer
        if (innerValue < outer.getLeftDomainBorder() || innerValue > outer.getRightDomainBorder()) {
            throw new IllegalArgumentException("значение внутренней функции вне области определения внешней"); // исключение
        }
        return outer.getFunctionValue(innerValue);
    }
}
```

Рис.(10-15)

Задание 5

В 5 задании в пакете functions создайть класс Functions, содержащий вспомогательные статические методы для работы с функциями и сделайть так, чтобы в программе вне этого класса нельзя было создать его объект. Класс должен содержать следующие методы (Рис.16)

```
// основные методы 5 ЗАДАНИЯ, используются методы из пакета functions.meta
public static Function shift(Function f, double shiftX, double shiftY) {  no usages  new *
    checkFunction(f);  // ← Теперь одна строка!
    return new Shift(f, shiftX, shiftY);
}
public static Function scale(Function f, double scaleX, double scaleY) {  no usages  new *
    checkFunction(f);
    checkScaleCoefficients(scaleX, scaleY);
    return new Scale(f, scaleX, scaleY);
}
public static Function power(Function f, double power) {  no usages  new *
    checkFunction(f);
    return new Power(f, power);
}
public static Function sum(Function f1, Function f2) {  no usages  new *
    checkFunctions(f1, f2);
    return new Sum(f1, f2);
}
public static Function mult(Function f1, Function f2) {  no usages  new *
    checkFunctions(f1, f2);
    return new Mult(f1, f2);
}
public static Function composition(Function f1, Function f2) {  no usages  new *
    checkFunctions(f1, f2);
    return new Composition(f1, f2);
}
```

Рис.16

Задание 6

В 6 пункте в пакете functions создаем класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями, нужно сделайте так, чтобы в программе вне этого класса нельзя было создать его объект. Для этого первым делом прописываем исключение `private TabulatedFunctions() {`

```
    throw new AssertionError("Нельзя создавать
объекты класса TabulatedFunctions");
```

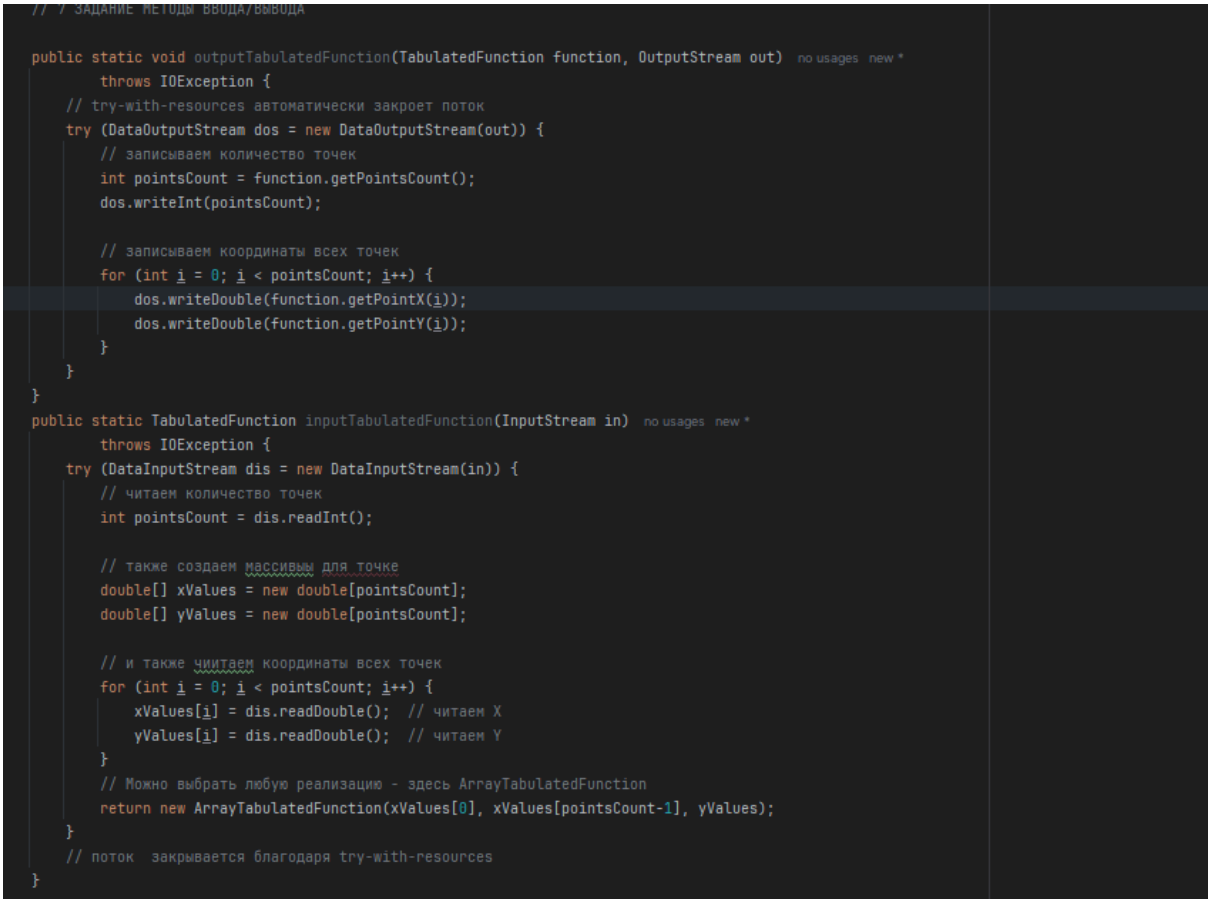
после чего уже пишется основная часть метода TabulatedFunctions.(Рис.17)

```
public static TabulatedFunction tabulate(Function function, double leftX,  double rightX, int pointsCount) {  no usages  new *
    // проверки параметров
    if (function == null) {
        throw new IllegalArgumentException("функция не может быть null");
    }
    if (pointsCount < 2) {
        throw new IllegalArgumentException("pointsCount должен быть >= 2");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("leftX должен быть < rightX");
    }
    // проверка, что отрезок в области определения функции
    if (leftX < function.getLeftDomainBorder() ||  rightX > function.getRightDomainBorder()) {
        throw new IllegalArgumentException(
                "Отрезок табулирования [" + leftX + ", " + rightX + "] " +  "выходит за область определения функции [" + function.getLeftDomainBorder() + ", " +
        );
    }

    // создаем массивы для точек
    double[] xValues = new double[pointsCount];
    double[] yValues = new double[pointsCount];

    // ищем шаг
    double step = (rightX - leftX) / (pointsCount - 1);

    // заполняем массивы
    for (int i = 0; i < pointsCount; i++) {
        xValues[i] = leftX + i * step;
        yValues[i] = function.getFunctionValue(xValues[i]);
    }
    // возвращаем табулированную функцию
    return new ArrayTabulatedFunction(leftX, rightX, yValues);
}
```

Рис.17

Задание 7

В 7 пункте в класс TabulatedFunctions реализовываем методы для работы с потоками ввода-вывода в классе TabulatedFunctions. Добавляем четыре статических метода: outputTabulatedFunction и inputTabulatedFunction для работы с байтовыми потоками, а также writeTabulatedFunction и readTabulatedFunction для работы с символьными потоками.(Рис.18)



```
// 7 ЗАДАНИЕ МЕТОДЫ ВВОДА/ВЫВОДА

public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out)  no usages  new *
        throws IOException {
    // try-with-resources автоматически закроет поток
    try (DataOutputStream dos = new DataOutputStream(out)) {
        // записываем количество точек
        int pointsCount = function.getPointsCount();
        dos.writeInt(pointsCount);

        // записываем координаты всех точек
        for (int i = 0; i < pointsCount; i++) {
            dos.writeDouble(function.getPointX(i));
            dos.writeDouble(function.getPointY(i));
        }
    }
}
public static TabulatedFunction inputTabulatedFunction(InputStream in)  no usages  new *
        throws IOException {
    try (DataInputStream dis = new DataInputStream(in)) {
        // читаем количество точек
        int pointsCount = dis.readInt();

        // также создаем массивы для точке
        double[] xValues = new double[pointsCount];
        double[] yValues = new double[pointsCount];

        // и также чиитаем координаты всех точек
        for (int i = 0; i < pointsCount; i++) {
            xValues[i] = dis.readDouble();  // читаем X
            yValues[i] = dis.readDouble();  // читаем Y
        }
        // Можно выбрать любую реализацию - здесь ArrayTabulatedFunction
        return new ArrayTabulatedFunction(xValues[0], xValues[pointsCount-1], yValues);
    }
    // поток  закрывается благодаря try-with-resources
}
```

Рис. 18

Задание 8

В 8 задании мы проверяем работу всех написанных классов.
Проверены все реализации: созданы Sin, Cos, выведены значения на [0, π].
Методом TabulatedFunctions.tabulate() получены табулированные версии, сравнение показало корректность интерполяции. Создана функция $\sin^2+\cos^2$ через classes.meta, значения соответствуют теории (≈1).
Протестированы файловые операции: экспонента записана в exp.txt (текст) и прочитана, логарифм — в log.bin (бинарный) и восстановлен.(Рис.19-22)

```
ЗАДАНИЕ 8: тесты работ классов
1. Sin и Cos от 0 до π с шагом 0.1:
x         sin(x)       cos(x)
0.0      0.0      1.0
0.1      0.09983341664682815      0.9950041652780258
0.2      0.19866933079506122      0.9800665778412416
0.30000000000000004      0.2955202066613396      0.955336489125606
0.4      0.3894183423086505      0.9210609940028851
0.5      0.479425538604203      0.8775825618903728
0.6      0.5646424733950354      0.8253356149096783
0.7      0.644217687237691      0.7648421872844885
0.7999999999999999      0.7173560908995227      0.6967067093471655
0.8999999999999999      0.7833269096274833      0.6216099682706645
0.9999999999999999      0.8414709848078964      0.5403023058681398
1.0999999999999999      0.8912073600614353      0.4535961214255775
1.2      0.9320390859672263      0.3623577544766736
1.3      0.963558185417193      0.26749882862458735
1.4000000000000001      0.9854497299884603      0.16996714290024081
1.5000000000000002      0.9974949866040544      0.07073720166770268
1.6000000000000003      0.9995736030415051      -0.029199522301289037
1.7000000000000004      0.9916648104524686      -0.12884449429552508
1.8000000000000005      0.973847630878195      -0.22720209469308753
1.9000000000000006      0.9463000876874142      -0.32328956686350396
2.0000000000000004      0.9092974268256815      -0.4161468365471428
2.1000000000000005      0.8632093666488735      -0.5048461045998579
2.2000000000000006      0.8084964038195899      -0.5885011172553463
2.3000000000000007      0.7457052121767197      -0.6662760212798248
2.400000000000001      0.6754631805511503      -0.737393715541246
2.500000000000001      0.5984721441039558      -0.8011436155469343
2.600000000000001      0.5155013718214634      -0.8568887533689478
2.700000000000001      0.42737988023382895      -0.9040721420170617
2.800000000000001      0.33498815015590383      -0.9422223406686585
2.9000000000000012      0.23924932921398112      -0.9709581651495908
3.0000000000000013      0.1411200080598659      -0.9899924966004456
3.1000000000000014      0.04158066243328916      -0.9991351502732795
```

```
3.1000000000000014      0.04158066243328916      -0.9991351502732795
 Табулированные аналоги (10 точек):
x         Sin (точный) Sin (табулированный) Cos (точный) Cos (табулированный)
0.0 0.0 0.0 1.0 1.0
0.1 0.09983341664682815 0.09798155360510165 0.9950041652780258 0.9827232084876878
0.2 0.19866933079506122 0.1959631072102033 0.9800665778412416 0.9654464169753757
0.30000000000000004 0.2955202066613396 0.29394466081530496 0.955336489125606 0.9481696254630635
0.4 0.3894183423086505 0.38590680571121505 0.9210609940028851 0.914354644443779
0.5 0.479425538604203 0.47207033789781927 0.8775825618903728 0.864608105941514
0.6 0.5646424733950354 0.5582338700844235 0.8253356149096783 0.8148615674392491
0.7 0.644217687237691 0.6439824415274444 0.7648421872844885 0.7646204979800333
0.7999999999999999 0.7173560908995227 0.707935358675545 0.6967067093471655 0.6884043792119047
0.8999999999999999 0.7833269096274833 0.7718882758236456 0.6216099682706645 0.6121882604437761
0.9999999999999999 0.8414709848078964 0.8358411929717461 0.5403023058681398 0.5359721416756473
1.0999999999999999 0.8912073600614353 0.8839933571281424 0.4535961214255775 0.4506334539143595
1.2 0.9320390859672263 0.9180219935851436 0.3623577544766736 0.35714054363391856
1.3 0.963558185417193 0.9520506300421447 0.26749882862458735 0.26364763335347763
1.4000000000000001 0.9854497299884603 0.984807753012208 0.16996714290024081 0.16993052093895483
1.5000000000000002 0.9974949866040544 0.984807753012208 0.07073720166770268 0.07043744393442489
1.6000000000000003 0.9995736030415051 0.984807753012208 -0.029199522301289037 -0.029055633070105086
1.7000000000000004 0.9916648104524686 0.984807753012208 -0.12884449429552508 -0.12854871007463503
1.8000000000000005 0.973847630878195 0.966204042925035 -0.22720209469308753 -0.22476145104951817
1.9000000000000006 0.9463000876874142 0.932175406468034 -0.32328956686350396 -0.3182543613299591
2.0000000000000004 0.9092974268256815 0.8981467700110329 -0.4161468365471428 -0.41174727161039987
2.1000000000000005 0.8632093666488735 0.8624409082617227 -0.5048461045998579 -0.5042718354168345
2.2000000000000006 0.8084964038195899 0.7984879911136221 -0.5885011172553463 -0.5804879541849632
2.3000000000000007 0.7457052121767197 0.7345350739655215 -0.6662760212798248 -0.656704072953092
2.400000000000001 0.6754631805511503 0.670582156817421 -0.737393715541246 -0.7329201917212208
2.500000000000001 0.5984721441039558 0.594071569547527 -0.8011436155469343 -0.7941706620070894
2.600000000000001 0.5155013718214634 0.5079080373609227 -0.8568887533689478 -0.8439172005093544
2.700000000000001 0.42737988023382895 0.42174450517431844 -0.9040721420170617 -0.8936637390116193
2.800000000000001 0.33498815015590383 0.3346977889881713 -0.9422223406686585 -0.9409837494179168
2.9000000000000012 0.23924932921398112 0.23671623538306957 -0.9709581651495908 -0.958260540930229
3.0000000000000013 0.1411200080598659 0.1387346817779678 -0.9899924966004456 -0.9755373324425413
3.1000000000000014 0.04158066243328916 0.04075312817286608 -0.9991351502732795 -0.9928141239548535
умма квадратов sin² + cos²:
```

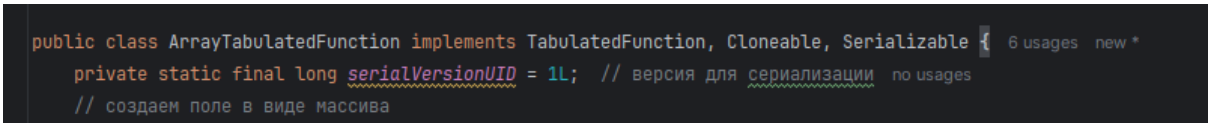
```
умма квадратов sin² + cos²:
Теория: sin²(x) + cos²(x) = 1 для любого x
x          sin²+cos² (10 точек)  sin²+cos² (20 точек)
0.0 1.0 1.0
0.1 0.9753452893472049 0.9934801762782068
0.2 0.9704883234380686 0.9954813685939703
0.30000000000000004 0.9854291022725908 0.9958763728929726
0.4 0.9849684785101429 0.9933589338027065
0.5 0.9703975807827336 0.9993625107499969
0.6 0.975624427798983 0.9936326956262082
0.7 0.9993578909268825 0.995117641167469
0.7999999999999999 0.9750730613812004 0.9963026540644755
0.8999999999999999 0.9705859765791769 0.9932689681997069
0.9999999999999999 0.9858966365208117 0.9987562983724949
1.0999999999999999 0.9845147652334687 0.9938164918467103
1.2 0.9703137486131723 0.9947851906134687
1.3 0.9759104767365345 0.9967602121084793
1.4000000000000001 0.9987226923395387 0.9932102794692081
1.5000000000000002 0.9748077439009694 0.9981813628674935
1.6000000000000003 0.9706905402060587 0.9940315649397132
1.7000000000000004 0.9863710812548067 0.994484016931969
1.8000000000000005 0.9840679624425679 0.997249047024984
1.9000000000000006 0.9702368269293845 0.9931828676112102
2.0000000000000004 0.9762034361598597 0.9976377042349929
2.1000000000000005 0.9980944042379682 0.994277914905217
2.2000000000000006 0.9745493369065118 0.9942141201229702
2.3000000000000007 0.9708020143187142 0.9977691588139896
2.400000000000001 0.9868524364745752 0.993186732625713
2.500000000000001 0.9836280701374409 0.9971253224749932
2.600000000000001 0.9701668157313703 0.9945555417432219
2.700000000000001 0.9765033060689583 0.9939755001864723
2.800000000000001 0.9974730266221714 0.9983205474754961
2.9000000000000012 0.974297840397828 0.9932218745127168
3.0000000000000013 0.9709203989171432 0.9966442175874943
3.1000000000000014 0.9873407021801173 0.9948644454537273
Экспонента: запись/чтение текстового файла (exp.txt)
Сравнение исходной и прочитанной функции:
```

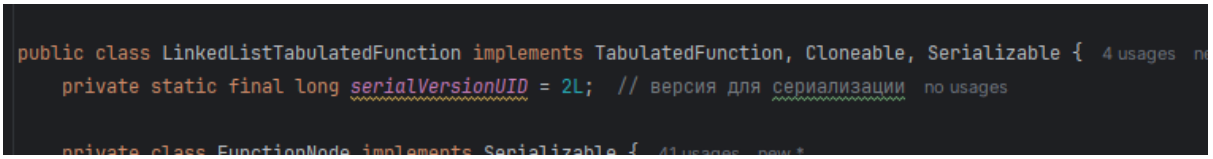
```
3.1000000000000014 0.9873407021801173 0.9948644454537273
Экспонента: запись/чтение текстового файла (exp.txt)
Сравнение исходной и прочитанной функции:
x         Исходная e^x  Прочитанная  Разница
0      1,000000      1,000000      0,0000000000
1      2,718282      2,718282      0,0000000000
2      7,389056      7,389056      0,0000000000
3      20,085537      20,085537      0,0000000000
4      54,598150      54,598150      0,0000000000
5      148,413159      148,413159      0,0000000000
6      403,428793      403,428793      0,0000000000
7      1096,633158      1096,633158      0,0000000000
8      2980,957987      2980,957987      0,0000000000
9      8103,083928      8103,083928      0,0000000000
10      22026,465795      22026,465795      0,0000000000
(логарифм, запись/чтение бинарного файла (log.bin)
Сравнение исходной и прочитанной функции:
x         Исходная ln(x)  Прочитанная  Разница
1.0 -0.1309825573257748 -0.1309825573257748 0.0
2.0 0.6801505040185054 0.6801505040185054 0.0
3.0 1.0941506153849527 1.0941506153849527 0.0
4.0 1.3842432516733665 1.3842432516733665 0.0
5.0 1.6083677751219336 1.6083677751219336 0.0
6.0 1.7911710515968466 1.7911710515968466 0.0
7.0 1.9455872302505781 1.9455872302505781 0.0
8.0 2.079276771049012 2.079276771049012 0.0
9.0 2.1971593478854805 2.1971593478854805 0.0
10.0 2.302585092994046 2.302585092994046 0.0

Process finished with exit code 0
```

Рис. 19-22

Задание 9

В 9 задании реализована поддержка сериализации для классов ArrayTabulatedFunction и LinkedListTabulatedFunction через интерфейс Serializable. Добавлены поля serialVersionUID. Протестирована на функции ln(e^x): объект сериализован в файл и успешно восстановлен с полным сохранением данных. Serializable обеспечивает простую реализацию с автоматическим сохранением всех полей. Также добавил поддержку Externalizable в LinkedListTabulatedFunction. Создал конструктор без параметров, без него Externalizable не работает. Реализовал два метода: writeExternal, который сохраняет все точки функции в поток, и readExternal, который читает их обратно и восстанавливает связный список. В main добавил тесты для ArrayTabulatedFunction и LinkedListTabulatedFunction: создал функции, сериализовал их в байтовый массив, десериализовал обратно, что значения в ключевых точках совпадают до и после сериализации..(Рис.23-25)





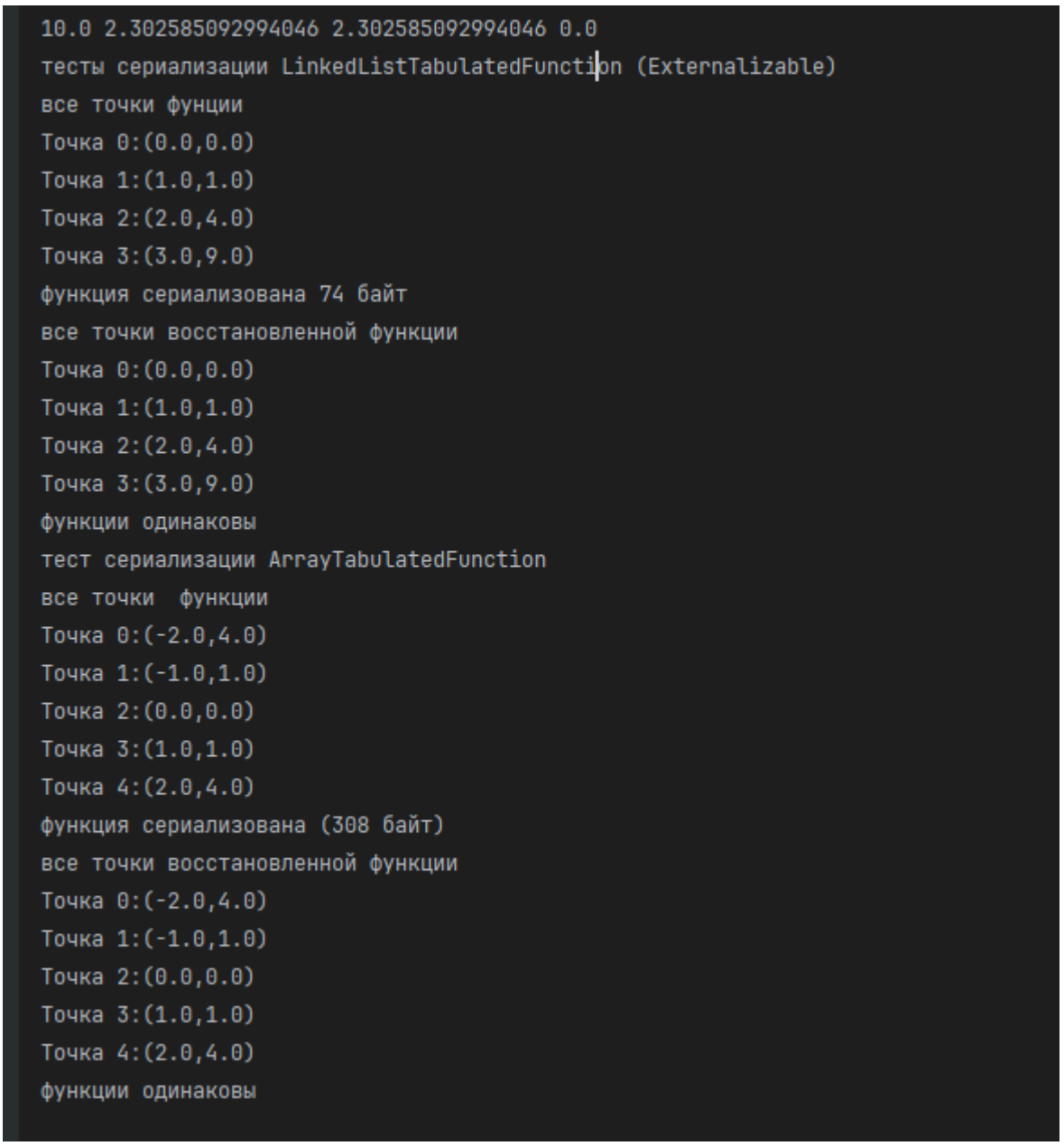

10.0 2.302585092994046 2.302585092994046 0.0
тесты сериализации LinkedListTabulatedFunction (Externalizable)
все точки фунции
Точка 0:(0.0,0.0)
Точка 1:(1.0,1.0)
Точка 2:(2.0,4.0)
Точка 3:(3.0,9.0)
функция сериализована 74 байт
все точки восстановленной функции
Точка 0:(0.0,0.0)
Точка 1:(1.0,1.0)
Точка 2:(2.0,4.0)
Точка 3:(3.0,9.0)
функции одинаковы
тест сериализации ArrayTabulatedFunction
все точки функции
Точка 0:(-2.0,4.0)
Точка 1:(-1.0,1.0)
Точка 2:(0.0,0.0)
Точка 3:(1.0,1.0)
Точка 4:(2.0,4.0)
функция сериализована (308 байт)
все точки восстановленной функции
Точка 0:(-2.0,4.0)
Точка 1:(-1.0,1.0)
Точка 2:(0.0,0.0)
Точка 3:(1.0,1.0)
Точка 4:(2.0,4.0)
функции одинаковы

Рис.(23-25)