

Лабораторная работа № 4

Выполнила: Рапп Ксения
Александровна
Группа: 6204-010302D

Оглавление

Задание 1.....	3-4
Задание 2	4
Задание 3	4
Задание 4.....	5
Задание 5.....	5-6
Задание 6.....	6-7
Задание 7.....	7-8
Задание 8.....	8-9
Задание 9.....	9-10
Результьат.....	10-17

Целью лабораторной работы было расширение возможностей пакета для работы с функциями одной переменной путем добавления интерфейсов и классов для аналитически заданных функций, а также методов ввода и вывода табулированных функций.

Задание 1: Конструкторы для массивов точек

Добавила конструкторы в оба класса табулированных функций, которые принимают готовый массив точек FunctionPoint[].

Это позволяет создавать функции напрямую из набора точек без промежуточных вычислений.

Особенности реализации:

- Проверка на минимальное количество точек (не менее 2)
- Проверка строгой упорядоченности точек по X с использованием машинного эпсилона 1e-10
- Обеспечение инкапсуляции через создание копий точек

ArrayTabulatedFunction:

```
public ArrayTabulatedFunction(FunctionPoint[] points) throws
IllegalArgumentException {
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек не может
быть меньше двух");
    }
    // Проверяем упорядоченность с учетом машинного эпсилона
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i + 1].getX() - points[i].getX() < 1e-10) {
            throw new IllegalArgumentException("Точки должны быть
строго упорядочены по возрастанию X");
        }
    }
    this.pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount + 2];
    for (int i = 0; i < pointsCount; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

LinkedListTabulatedFunction:

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {
    if (points.length < 2) {
```

```

        throw new IllegalArgumentException("Количество точек не может
быть меньше двух");
    }
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i + 1].getX() - points[i].getX() < 1e-10) {
            throw new IllegalArgumentException("Точки должны быть строго
упорядочены");
        }
    }
    initEmptyList();
    for (FunctionPoint point : points) {
        addNodeToTail().setPoint(new FunctionPoint(point));
    }
}

```

Задание 2: Интерфейс Function

Создала базовый интерфейс Function для всех функций одной переменной и реорганизовала иерархию классов, сделав TabulatedFunction его наследником.

```

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

```

```

public interface TabulatedFunction extends Function {
    // Все методы из Function наследуются автоматически
    int getPointsCount();
    FunctionPoint getPoint(int index);
    // ... остальные методы
}

```

Задание 3: Базовые аналитические функции

Реализовала классы для основных математических функций в пакете functions.basic. Создала иерархию с базовым классом TrigonometricFunction для тригонометрических функций.

Exp - экспоненциальная функция:

- Область определения: $(-\infty, +\infty)$
- Использует Math.exp() для вычислений

Log - логарифмическая функция:

- Основание передается в конструкторе
- Область определения: $(0, +\infty)$

- Проверка корректности основания

Тригонометрические функции:

Создала базовый класс TrigonometricFunction с общей областью определения и наследников для конкретных функций.

Задание 4: Мета-функции

Реализовала в пакете functions.meta классы для комбинирования функций. Все классы реализуют интерфейс Function и поддерживают сериализацию.

Sum - сумма функций:

- Область определения - пересечение областей исходных функций
- Значение - сумма значений исходных функций

Composition - композиция функций:

- $f(g(x))$ - применение одной функции к результату другой

Также реализованы:

- Mult - произведение функций
- Power - возведение функции в степень
- Scale - масштабирование по осям
- Shift - сдвиг по осям

Задание 5: Утилитарный класс Functions

Создала фабричный класс со статическими методами для удобного создания мета-функций. Класс имеет приватный конструктор для запрета создания экземпляров.

```
public class Functions {
    // Приватный конструктор - запрещаем создание объектов
    private Functions() {
        throw new AssertionError("Нельзя создавать объекты класса
Functions");
    }
    // Сдвиг функции вдоль осей
    public static Function shift(Function f, double shiftX, double shiftY) {
        return new Shift(f, shiftX, shiftY);
    }
    // Масштабирование функции вдоль осей
}
```

```

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

// Возведение функции в степень

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

// Сумма двух функций

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

// Произведение двух функций

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

// Композиция двух функций

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

```

Задание 6: Табулирование функций

Реализовала метод для создания табулированного аналога аналитической функции на заданном отрезке.

Особенности:

- Проверка, что отрезок табулирования входит в область определения
- Равномерное распределение точек на отрезке
- Линейная интерполяция между точками

```

public class TabulatedFunctions {
    private TabulatedFunctions() {}

    public static TabulatedFunction tabulate(Function function, double leftX,
    double rightX, int pointsCount) {
        if (leftX < function.getLeftDomainBorder() - 1e-10 ||
```

```

    rightX > function.getRightDomainBorder() + 1e-10) {
        throw new IllegalArgumentException("Границы выходят за
область определения");
    }
    if (pointsCount < 2) throw new IllegalArgumentException("Минимум
2 точки");
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        values[i] = function.getFunctionValue(x);
    }
    return new ArrayTabulatedFunction(leftX, rightX, values); }}
```

Задание 7: Ввод-вывод табулированных функций

Реализовала методы для сохранения и загрузки табулированных функций в различных форматах.

Байтовые потоки:

- Использование DataOutputStream/DataInputStream

```
public static void outputTabulatedFunction(TabulatedFunction function,
OutputStream out) throws IOException {
    DataOutputStream dos = new DataOutputStream(out);
    dos.writeInt(function.getPointsCount());
    for (int i = 0; i < function.getPointsCount(); i++) {
        dos.writeDouble(function.getPointX(i));
        dos.writeDouble(function.getPointY(i));
    }
    dos.flush(); }
```

Символьные потоки (текстовый формат):

- Использование StreamTokenizer для разбора

```
• public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws
IOException {
    BufferedWriter bw = new BufferedWriter(out);
    ...
    // записываем количество точек
    bw.write(String.valueOf(function.getPointsCount()));
    bw.write(" ");
    ...
    // записываем все точки (x1 y1 x2 y2...)
    for (int i = 0; i < function.getPointsCount(); i++) {
        bw.write(String.valueOf(function.getPointX(i))); // записываем x
        bw.write(" ");
        bw.write(String.valueOf(function.getPointY(i))); // записываем y
        if (i < function.getPointsCount() - 1) {
            bw.write(" ");
        }
    }
}
```

```
•     }
•   }
•
•   bw.flush();
• }
•
```

Обработка исключений:

- Потоки не закрываются внутри методов (это ответственность вызывающего кода)
- IOException преобразуется в RuntimeException - вызывающий код не обязан обрабатывать checked исключения
- Это позволяет гибко использовать методы как с автозакрываемыми ресурсами (try-with-resources)

Задание 8: Тестирование функциональности

Создала комплексные тесты в классе Main, проверяющие все аспекты работы системы.

Основные тесты:

1. Базовые функции - проверка аналитических функций
 - Exp: e^x на различных значениях
 - Log: натуральный логарифм $\ln(x)$
 - Sin, Cos, Tan: тригонометрические функции с проверкой области определения
2. Табулирование - сравнение аналитических и табулированных значений
 - Табулирование синуса на отрезке $[0, \pi]$ с 10 точками
 - Сравнение точных и интерполированных значений с шагом 0.1
 - Визуализация разницы между аналитическим и табулированным представлением
3. Мета-функции - проверка правильности комбинирования функций
 - Сумма квадратов: $\sin^2(x) + \cos^2(x) \approx 1$ (проверка тождества)
 - Композиция: $\ln(e^x) = x$ (проверка обратных функций)
 - Масштабирование и сдвиг: проверка преобразований координат
4. Ввод-вывод - проверка сериализации и десериализации
 - Байтовые потоки: экспонента на $[0, 10]$ с 11 точками

- Символьные потоки: логарифм на [1, 10] с 10 точками
- Сравнение исходных и восстановленных данных

5. СерIALIZАЦИЯ - проверка механизмов сохранения объектов
- Serializable: ArrayTabulatedFunction с функцией $\ln(e^x)$
 - Externalizable: LinkedListTabulatedFunction с массивом точек
 - Детальное сравнение до и после сериализации

Пример тестирования из Main:

```
public class Main {
    public static void main(String[] args) {
        // Тест базовых функций
        Sin sin = new Sin();
        Cos cos = new Cos();

        System.out.println("Sin и Cos от 0 до  $\pi$ :");
        for (double x = 0; x <= Math.PI; x += 0.1) {
            System.out.printf("x=% .1f: sin=% .4f, cos=% .4f\n",
                x, sin.getFunctionValue(x), cos.getFunctionValue(x));
        }

        // Тест суммы квадратов  $\sin^2 + \cos^2$ 
        Function sin2 = Functions.power(sin, 2);
        Function cos2 = Functions.power(cos, 2);
        Function sum = Functions.sum(sin2, cos2);

        System.out.println("sin2 + cos2 (должно быть ≈1):");
        for (double x = 0; x <= Math.PI; x += 0.5) {
            double result = sum.getFunctionValue(x);
            System.out.printf("x=% .1f: % .6f %s\n", x, result,
                Math.abs(result - 1) < 1e-5 ? "✓" : "✗");
        }

        // Комплексное тестирование
        testTabulation();
        testInputOutput();
        testSerialization();
    }
}
```

Задание 9: СерIALIZАЦИЯ

Реализовала два подхода к сериализации объектов табулированных функций.

1. Serializable (автоматическая сериализация)

```
public class ArrayTabulatedFunction implements TabulatedFunction,  
Serializable {  
    private static final long serialVersionUID = 1L;  
    // Поля сериализуются автоматически  
}
```

2. Externalizable (ручная сериализация):

```
public class LinkedListTabulatedFunction implements TabulatedFunction,  
Externalizable {  
    @Override  
    public void writeExternal(ObjectOutput out) throws IOException {  
        out.writeInt(pointsCount);  
        FunctionNode current = head.getNext();  
        while (current != head) {  
            out.writeDouble(current.getPoint().getX());  
            out.writeDouble(current.getPoint().getY());  
            current = current.getNext();  
        }  
    }  
}
```

```
@Override  
public void readExternal(ObjectInput in) throws IOException,  
ClassNotFoundException {  
    int count = in.readInt();  
    initEmptyList();  
    for (int i = 0; i < count; i++) {  
        double x = in.readDouble();  
        double y = in.readDouble();  
        addNodeToTail().setPoint(new FunctionPoint(x, y)); } }
```

Сравнение подходов:

- **Serializable** - проще в реализации, но менее гибкий
- **Externalizable** - больше контроля над процессом, возможность оптимизации

Тестирование:

1. Тестирование базовых функций:

Экспонента:

$\exp(0) = 1.0$

$\exp(1) = 2.7182818284590455$

$\exp(2) = 7.38905609893065$

Область определения: от $-\infty$ до ∞

Натуральный логарифм:

$\ln(1) = 0.0$
 $\ln(2) = 0.6931471805599453$
 $\ln(e) = 1.0$
Область определения: от 0.0 до Infinity

Тригонометрические функции:
 $\sin(0) = 0.0$
 $\sin(\pi/2) = 1.0$
 $\cos(0) = 1.0$
 $\cos(\pi) = -1.0$
Область определения: от -Infinity до Infinity

2. Тестирование мета-функций:

Сумма квадратов $\sin^2(x) + \cos^2(x)$:
 $x=0.0: 1.0$ (должно быть 1.0)
 $x=0.5: 1.0$ (должно быть 1.0)
 $x=1.0: 1.0$ (должно быть 1.0)
 $x=1.5: 0.9999999999999998$ (должно быть 1.0)
 $x=2.0: 1.0$ (должно быть 1.0)
 $x=2.5: 0.9999999999999999$ (должно быть 1.0)
 $x=3.0: 0.9999999999999999$ (должно быть 1.0)

Композиция $\ln(e^x)$:
 $x=0.5: \ln(e^{0.5}) = 0.5$ (должно быть 0.5)
 $x=1.0: \ln(e^{1.0}) = 1.0$ (должно быть 1.0)
 $x=1.5: \ln(e^{1.5}) = 1.5$ (должно быть 1.5)
 $x=2.0: \ln(e^{2.0}) = 2.0$ (должно быть 2.0)

3. Тестирование табулирования:

Табулирование синуса на $[0, \pi]$ с 10 точками:
Табулированные точки:
(0.0, 0.0)
(0.3490658503988659, 0.3420201433256687)
(0.6981317007977318, 0.6427876096865393)
(1.0471975511965976, 0.8660254037844386)
(1.3962634015954636, 0.984807753012208)
(1.7453292519943295, 0.9848077530122081)
(2.0943951023931953, 0.8660254037844388)
(2.443460952792061, 0.6427876096865395)
(2.792526803190927, 0.3420201433256689)
(3.141592653589793, 1.2246467991473532E-16)

4. Тестирование работы с функциями и файлами:

1. Sin и Cos на отрезке $[0, \pi]$ с шагом 0.1;
x Sin(x) Cos(x)

0,0	0,000000	1,000000
0,1	0,099833	0,995004
0,2	0,198669	0,980067
0,3	0,295520	0,955336
0,4	0,389418	0,921061
0,5	0,479426	0,877583
0,6	0,564642	0,825336
0,7	0,644218	0,764842
0,8	0,717356	0,696707
0,9	0,783327	0,621610
1,0	0,841471	0,540302
1,1	0,891207	0,453596
1,2	0,932039	0,362358
1,3	0,963558	0,267499
1,4	0,985450	0,169967
1,5	0,997495	0,070737
1,6	0,999574	-0,029200
1,7	0,991665	-0,128844
1,8	0,973848	-0,227202
1,9	0,946300	-0,323290
2,0	0,909297	-0,416147
2,1	0,863209	-0,504846
2,2	0,808496	-0,588501
2,3	0,745705	-0,666276
2,4	0,675463	-0,737394
2,5	0,598472	-0,801144
2,6	0,515501	-0,856889
2,7	0,427380	-0,904072
2,8	0,334988	-0,942222
2,9	0,239249	-0,970958
3,0	0,141120	-0,989992
3,1	0,041581	-0,999135

2. Сравнение точных и табулированных функций:

x	Sin(точн)	Sin(табл)	Ошибка	Cos(точн)	Cos(табл)	Ошибка
0,0	0,000000	0,000000	0,000000	1,000000	1,000000	0,000000
0,1	0,099833	0,097982	0,001852	0,995004	0,982723	0,012281
0,2	0,198669	0,195963	0,002706	0,980067	0,965446	0,014620
0,3	0,295520	0,293945	0,001576	0,955336	0,948170	0,007167
0,4	0,389418	0,385907	0,003512	0,921061	0,914355	0,006706
0,5	0,479426	0,472070	0,007355	0,877583	0,864608	0,012974
0,6	0,564642	0,558234	0,006409	0,825336	0,814862	0,010474
0,7	0,644218	0,643982	0,000235	0,764842	0,764620	0,000222
0,8	0,717356	0,707935	0,009421	0,696707	0,688404	0,008302
0,9	0,783327	0,771888	0,011439	0,621610	0,612188	0,009422

1,0	0,841471	0,835841	0,005630	0,540302	0,535972	0,004330
1,1	0,891207	0,883993	0,007214	0,453596	0,450633	0,002963
1,2	0,932039	0,918022	0,014017	0,362358	0,357141	0,005217
1,3	0,963558	0,952051	0,011508	0,267499	0,263648	0,003851
1,4	0,985450	0,984808	0,000642	0,169967	0,169931	0,000037
1,5	0,997495	0,984808	0,012687	0,070737	0,070437	0,000300
1,6	0,999574	0,984808	0,014766	-0,029200	-0,029056	0,000144
1,7	0,991665	0,984808	0,006857	-0,128844	-0,128549	0,000296
1,8	0,973848	0,966204	0,007644	-0,227202	-0,224761	0,002441
1,9	0,946300	0,932175	0,014125	-0,323290	-0,318254	0,005035
2,0	0,909297	0,898147	0,011151	-0,416147	-0,411747	0,004400
2,1	0,863209	0,862441	0,000768	-0,504846	-0,504272	0,000574
2,2	0,808496	0,798488	0,010008	-0,588501	-0,580488	0,008013
2,3	0,745705	0,734535	0,011170	-0,666276	-0,656704	0,009572
2,4	0,675463	0,670582	0,004881	-0,737394	-0,732920	0,004474
2,5	0,598472	0,594072	0,004401	-0,801144	-0,794171	0,006973
2,6	0,515501	0,507908	0,007593	-0,856889	-0,843917	0,012972
2,7	0,427380	0,421745	0,005635	-0,904072	-0,893664	0,010408
2,8	0,334988	0,334698	0,000290	-0,942222	-0,940984	0,001239
2,9	0,239249	0,236716	0,002533	-0,970958	-0,958261	0,012698
3,0	0,141120	0,138735	0,002385	-0,989992	-0,975537	0,014455
3,1	0,041581	0,040753	0,000828	-0,999135	-0,992814	0,006321

Статистика ошибок:

Максимальная ошибка Sin: 0,01476585

Максимальная ошибка Cos: 0,01462016

Средняя ошибка Sin: 0,00628865

Средняя ошибка Cos: 0,00621498

3. Сумма квадратов табулированных функций:

x	$\text{Sin}^2 + \text{Cos}^2$	Ожидаемое	Ошибка
---	-------------------------------	-----------	--------

x	$\text{Sin}^2 + \text{Cos}^2$	Ожидаемое	Ошибка
0,0	1,000000	1,0	0,000000
0,1	0,975345	1,0	0,024655
0,2	0,970488	1,0	0,029512
0,3	0,985429	1,0	0,014571
0,4	0,984968	1,0	0,015032
0,5	0,970398	1,0	0,029602
0,6	0,975624	1,0	0,024376
0,7	0,999358	1,0	0,000642
0,8	0,975073	1,0	0,024927
0,9	0,970586	1,0	0,029414
1,0	0,985897	1,0	0,014103
1,1	0,984515	1,0	0,015485
1,2	0,970314	1,0	0,029686
1,3	0,975910	1,0	0,024090

1,4	0,998723	1,0	0,001277
1,5	0,974808	1,0	0,025192
1,6	0,970691	1,0	0,029309
1,7	0,986371	1,0	0,013629
1,8	0,984068	1,0	0,015932
1,9	0,970237	1,0	0,029763
2,0	0,976203	1,0	0,023797
2,1	0,998094	1,0	0,001906
2,2	0,974549	1,0	0,025451
2,3	0,970802	1,0	0,029198
2,4	0,986852	1,0	0,013148
2,5	0,983628	1,0	0,016372
2,6	0,970167	1,0	0,029833
2,7	0,976503	1,0	0,023497
2,8	0,997473	1,0	0,002527
2,9	0,974298	1,0	0,025702
3,0	0,970920	1,0	0,029080
3,1	0,987341	1,0	0,012659

Статистика для суммы квадратов:
Максимальная ошибка: 0,02983318
Средняя ошибка: 0,01951143

4. Работа с файлами:

Табулированная экспонента (11 точек):

(0.0, 1.0)

(1.0, 2.7182818284590455)
(2.0, 7.38905609893065)
(3.0, 20.085536923187668)
(4.0, 54.598150033144236)
(5.0, 148.4131591025766)
(6.0, 403.4287934927351)
(7.0, 1096.6331584284585)
(8.0, 2980.9579870417283)
(9.0, 8103.083927575384)
(10.0, 22026.465794806718)

Экспонента записана в файл: exp_function.txt

Экспонента прочитана из файла

Сравнение исходной и прочитанной экспоненты:

Точка 0 (x=0,0): исходная=1,000000, прочитанная=1,000000,
ошибка=0,0000000000

Точка 1 (x=1,0): исходная=2,718282, прочитанная=2,718282,
ошибка=0,0000000000

Точка 2 (x=2,0): исходная=7,389056, прочитанная=7,389056,
ошибка=0,0000000000

Точка 3 ($x=3,0$): исходная=20,085537, прочитанная=20,085537,
ошибка=0,00000000000

Точка 4 ($x=4,0$): исходная=54,598150, прочитанная=54,598150,
ошибка=0,00000000000

Точка 5 ($x=5,0$): исходная=148,413159, прочитанная=148,413159,
ошибка=0,00000000000

Точка 6 ($x=6,0$): исходная=403,428793, прочитанная=403,428793,
ошибка=0,00000000000

Точка 7 ($x=7,0$): исходная=1096,633158, прочитанная=1096,633158,
ошибка=0,00000000000

Точка 8 ($x=8,0$): исходная=2980,957987, прочитанная=2980,957987,
ошибка=0,00000000000

Точка 9 ($x=9,0$): исходная=8103,083928, прочитанная=8103,083928,
ошибка=0,00000000000

Точка 10 ($x=10,0$): исходная=22026,465795, прочитанная=22026,465795,
ошибка=0,00000000000

Максимальная ошибка: 0,00000000000

Средняя ошибка: 0,00000000000

Табулированный логарифм (11 точек):

(0.1, -2.3025850929940455)

(1.09, 0.0861776962410524)

(2.08, 0.7323678937132266)

(3.07, 1.1216775615991057)

(4.06, 1.4011829736136412)

(5.05, 1.6193882432872684)

(6.039999999999999, 1.7984040119467235)

(7.029999999999999, 1.9501867058225735)

(8.02, 2.081938421878423)

(9.01, 2.1983350716202463)

(10.0, 2.302585092994046)

Логарифм записан в файл: log_function.dat

Логарифм также записан в текстовый файл: log_function.txt

Логарифм прочитан из файла

Сравнение исходного и прочитанного логарифма:

Точка 0 ($x=0,1$): исходная=-2,302585, прочитанная=-2,302585,
ошибка=0,00000000000

Точка 1 ($x=1,1$): исходная=0,086178, прочитанная=0,086178,
ошибка=0,00000000000

Точка 2 ($x=2,1$): исходная=0,732368, прочитанная=0,732368,
ошибка=0,00000000000

Точка 3 ($x=3,1$): исходная=1,121678, прочитанная=1,121678,
ошибка=0,00000000000

Точка 4 ($x=4,1$): исходная=1,401183, прочитанная=1,401183,
ошибка=0,00000000000

Точка 5 (x=5,1): исходная=1,619388, прочитанная=1,619388,
ошибка=0,00000000000

Точка 6 (x=6,0): исходная=1,798404, прочитанная=1,798404,
ошибка=0,00000000000

Точка 7 (x=7,0): исходная=1,950187, прочитанная=1,950187,
ошибка=0,00000000000

Точка 8 (x=8,0): исходная=2,081938, прочитанная=2,081938,
ошибка=0,00000000000

Точка 9 (x=9,0): исходная=2,198335, прочитанная=2,198335,
ошибка=0,00000000000

Точка 10 (x=10,0): исходная=2,302585, прочитанная=2,302585,
ошибка=0,00000000000

Максимальная ошибка: 0,00000000000

Средняя ошибка: 0,00000000000

Сравнение форматов хранения:

Размер бинарного файла: 180 байт

Размер текстового файла: 290 байт

5. СерIALIZАЦИЯ (Serializable):

Оригинальная функция $\ln(e^x)$:

(0.0, 0.0)

(1.0, 1.0)

(2.0, 2.0)

(3.0, 3.0)

(4.0, 4.0)

(5.0, 5.0)

Функция сериализована в файл: function_serializable.ser

Функция десериализована из файла

Десериализованная функция:

(0.0, 0.0)

(1.0, 1.0)

(2.0, 2.0)

(3.0, 3.0)

(4.0, 4.0)

(5.0, 5.0)

Сравнение оригинальной и десериализованной функции:

Точка 0 (x=0,0): исходная=0,000000, десериализ.=0,000000, ошибка=0,00000000000

Точка 1 (x=1,0): исходная=1,000000, десериализ.=1,000000, ошибка=0,00000000000

Точка 2 (x=2,0): исходная=2,000000, десериализ.=2,000000, ошибка=0,00000000000

Точка 3 (x=3,0): исходная=3,000000, десериализ.=3,000000, ошибка=0,00000000000

Точка 4 (x=4,0): исходная=4,000000, десериализ.=4,000000, ошибка=0,00000000000

Точка 5 (x=5,0): исходная=5,000000, десериализ.=5,000000, ошибка=0,00000000000

Максимальная ошибка: 0,00000000000

Средняя ошибка: 0,00000000000

Временный файл удален

6. Сериализация (Externalizable):

Оригинальная функция:

(1.0, 0.0)
(2.0, 0.693147)
(3.0, 1.098612)
(4.0, 1.386294)
(5.0, 1.609438)

Функция сериализована в файл: function_externalizable.ser

Функция десериализована из файла

Десериализованная функция:

(1.0, 0.0)
(2.0, 0.693147)
(3.0, 1.098612)
(4.0, 1.386294)
(5.0, 1.609438)

Сравнение оригинальной и десериализованной функции:

Точка 0 ($x=1,0$): исходная=0,000000, десериализ.=0,000000, ошибка=0,0000000000

Точка 1 ($x=2,0$): исходная=0,693147, десериализ.=0,693147, ошибка=0,0000000000

Точка 2 ($x=3,0$): исходная=1,098612, десериализ.=1,098612, ошибка=0,0000000000

Точка 3 ($x=4,0$): исходная=1,386294, десериализ.=1,386294, ошибка=0,0000000000

Точка 4 ($x=5,0$): исходная=1,609438, десериализ.=1,609438, ошибка=0,0000000000

Максимальная ошибка: 0,0000000000

Средняя ошибка: 0,0000000000

Выводы о способах сериализации:

- Serializable: проще в использовании, автоматическая сериализация

- Externalizable: больше контроля, можно оптимизировать процесс

Временный файл удален