

Лабораторная работа № 4

Выполнила: Рапп Ксения

Александровна

Группа: 6204-010302D

Оглавление

Задание 1.....	3-4
Задание 2	5
Задание 3	6
Задание 4.....	6-7
Задание 5.....	7-8
Задание 6.....	8-9
Задание 7.....	9-10
Задание 8.....	11-12
Задание 9.....	12-14
Результат.....	14-19

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) {  
  
    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");  
  
        }  
  
    }  
  
}
```

```

    }

    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 {
```

```
    // остальные методы остаются без изменений
```

```
    int getPointsCount();
```

```
    FunctionPoint getPoint(int index);
```

```
    // ... }
```

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.

```
java
```

```
public class Functions {
```

```
    private Functions() {
```

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

```
    }
```

```
    public static Function shift(Function f, double shiftX, double shiftY) {
```

```
        return new Shift(f, shiftX, shiftY);
```

```
    }
```

```
    public static Function sum(Function f1, Function f2) {
```

```
        return new Sum(f1, f2);
```

```
}
```

```
public static Function composition(Function f1, Function f2) {
```

```
    return new Composition(f1, f2);
```

```
}
```

```
// ... остальные методы для power, mult, scale
```

```
}
```

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

```
java
```

```
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`

java

```

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

java

```
public static void writeTabulatedFunction(TabulatedFunction function,  
Writer out) {
```

```
    PrintWriter writer = new PrintWriter(out);
```

```
    writer.print(function.getPointsCount() + " ");
```

```
    for (int i = 0; i < function.getPointsCount(); i++) {
```

```
        writer.print(function.getPointX(i) + " " + function.getPointY(i) + " ");
```

```
    }
```

```
    writer.flush();
```

```
}
```

Обработка исключений:

- Потоки не закрываются внутри методов (это ответственность вызывающего кода)

Задание 8: Тестирование функциональности

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

Основные тесты:

1. **Базовые функции** - проверка значений sin, cos, exp, log
2. **Табулирование** - сравнение аналитических и табулированных значений
3. **Мета-функции** - проверка правильности комбинирования
4. **Ввод-вывод** - проверка сериализации и десериализации

```
java
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        // Тест синуса и косинуса
```

```
        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) {
```

```
    System.out.printf("x=%.1f: %.6f%n", x, sum.getFunctionValue(x));  
}
```

```
// Тест ввода-вывода
```

```
testInputOutput(); }}
```

Задание 9: Сериализация

Реализовала два подхода к сериализации объектов табулированных функций.

1. Serializable (автоматическая сериализация):

```
java
```

```
public class ArrayTabulatedFunction implements TabulatedFunction,  
Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
// Поля сериализуются автоматически
```

```
}
```

2. Externalizable (ручная сериализация):

```
java
```

```
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 сериализации:

```
java
```

```
// Сериализация
```

```
TabulatedFunction original = TabulatedFunctions.tabulate(function, 0, 10,  
11);
```

```
try (ObjectOutputStream oos = new ObjectOutputStream(new  
FileOutputStream("function.ser"))) {
```

```
    oos.writeObject(original);
```

```
}
```

```
// Десериализация
```

```
try (ObjectInputStream ois = new ObjectInputStream(new  
FileInputStream("function.ser"))) {
```

```
    TabulatedFunction restored = (TabulatedFunction) ois.readObject();
```

```
// Сравнение оригинальной и восстановленной функции }
```

Результат запуска main:

1. Тестирование базовых функций:

$$\text{Exp}(1) = 2,7183$$

Область определения Exp: $[-\text{Infinity}, \text{Infinity}]$

$$\text{Ln}(2) = 0,6931$$

$$\text{Sin}(\pi/2) = 1,0000$$

$$\text{Cos}(\pi) = -1,0000$$

2. Тестирование meta функций:

$\sin^2(x) + \cos^2(x)$ в точках:

$$x=0,0: 1,000000$$

$$x=0,5: 1,000000$$

$$x=1,0: 1,000000$$

$$x=1,5: 1,000000$$

$$x=2,0: 1,000000$$

$$x=2,5: 1,000000$$

$$x=3,0: 1,000000$$

$$\text{Ln}(e^2) = 2,0000$$

3. Тестирование табулирования:

Табулированный синус (первые 5 точек):

$$(0.0, 0.0)$$

$$(0.3490658503988659, 0.3420201433256687)$$

$$(0.6981317007977318, 0.6427876096865393)$$

$$(1.0471975511965976, 0.8660254037844386)$$

(1.3962634015954636, 0.984807753012208)

Сравнение точного и табулированного синуса:

x=0,0: точный=0,000000, табулир=0,000000, разница=0,000000

x=0,1: точный=0,099833, табулир=0,097982, разница=0,001852

x=0,2: точный=0,198669, табулир=0,195963, разница=0,002706

x=0,3: точный=0,295520, табулир=0,293945, разница=0,001576

x=0,4: точный=0,389418, табулир=0,385907, разница=0,003512

x=0,5: точный=0,479426, табулир=0,472070, разница=0,007355

x=0,6: точный=0,564642, табулир=0,558234, разница=0,006409

x=0,7: точный=0,644218, табулир=0,643982, разница=0,000235

x=0,8: точный=0,717356, табулир=0,707935, разница=0,009421

x=0,9: точный=0,783327, табулир=0,771888, разница=0,011439

x=1,0: точный=0,841471, табулир=0,835841, разница=0,005630

x=1,1: точный=0,891207, табулир=0,883993, разница=0,007214

x=1,2: точный=0,932039, табулир=0,918022, разница=0,014017

x=1,3: точный=0,963558, табулир=0,952051, разница=0,011508

x=1,4: точный=0,985450, табулир=0,984808, разница=0,000642

x=1,5: точный=0,997495, табулир=0,984808, разница=0,012687

x=1,6: точный=0,999574, табулир=0,984808, разница=0,014766

x=1,7: точный=0,991665, табулир=0,984808, разница=0,006857

x=1,8: точный=0,973848, табулир=0,966204, разница=0,007644

x=1,9: точный=0,946300, табулир=0,932175, разница=0,014125

x=2,0: точный=0,909297, табулир=0,898147, разница=0,011151

x=2,1: точный=0,863209, табулир=0,862441, разница=0,000768

x=2,2: точный=0,808496, табулир=0,798488, разница=0,010008

x=2,3: точный=0,745705, табулир=0,734535, разница=0,011170

x=2,4: точный=0,675463, табулир=0,670582, разница=0,004881

x=2,5: точный=0,598472, табулир=0,594072, разница=0,004401

x=2,6: точный=0,515501, табулир=0,507908, разница=0,007593

x=2,7: точный=0,427380, табулир=0,421745, разница=0,005635

x=2,8: точный=0,334988, табулир=0,334698, разница=0,000290

x=2,9: точный=0,239249, табулир=0,236716, разница=0,002533

x=3,0: точный=0,141120, табулир=0,138735, разница=0,002385

x=3,1: точный=0,041581, табулир=0,040753, разница=0,000828

4. Тестирование ввода-вывода:

Байтовые потоки:

Сравнение исходной и прочитанной экспоненты:

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

Символьные потоки:

Сериализованные данные: 10 1.0 0.0 2.0 0.6931471805599453 3.0
1.0986122886681096 4.0 1.3862943611198906 5.0 1.6094379124341003 6.0
1.791759469228055 7.0 1.9459101490553132 8.0 2.0794415416798357 9.0
2.1972245773362196 10.0 2.302585092994046

Сравнение логарифмов:

(1.0, 0.0) -> (1.0, 0.0)

(2.0, 0.6931471805599453) -> (2.0, 0.6931471805599453)

(3.0, 1.0986122886681096) -> (3.0, 1.0986122886681096)

(4.0, 1.3862943611198906) -> (4.0, 1.3862943611198906)

(5.0, 1.6094379124341003) -> (5.0, 1.6094379124341003)

(6.0, 1.791759469228055) -> (6.0, 1.791759469228055)

(7.0, 1.9459101490553132) -> (7.0, 1.9459101490553132)

(8.0, 2.0794415416798357) -> (8.0, 2.0794415416798357)

(9.0, 2.1972245773362196) -> (9.0, 2.1972245773362196)

(10.0, 2.302585092994046) -> (10.0, 2.302585092994046)

5. Тестирование сериализации:

Исходная функция ($\ln(e^x)$) - все 11 точек:

x=0,0: y=0,000000 (ожидается: 0,0)

x=1,0: y=1,000000 (ожидается: 1,0)

x=2,0: y=2,000000 (ожидается: 2,0)

x=3,0: y=3,000000 (ожидается: 3,0)

x=4,0: y=4,000000 (ожидается: 4,0)

x=5,0: y=5,000000 (ожидается: 5,0)

x=6,0: y=6,000000 (ожидается: 6,0)

x=7,0: y=7,000000 (ожидается: 7,0)

x=8,0: y=8,000000 (ожидается: 8,0)

x=9,0: y=9,000000 (ожидается: 9,0)

x=10,0: y=10,000000 (ожидается: 10,0)

Функция сериализована в файл: tabulated_function.ser

Десериализованная функция - все 11 точек:

x=0,0: исходная=0,000000, восстановленная=0,000000, разница=0,000000

x=1,0: исходная=1,000000, восстановленная=1,000000, разница=0,000000

x=2,0: исходная=2,000000, восстановленная=2,000000, разница=0,000000

x=3,0: исходная=3,000000, восстановленная=3,000000, разница=0,000000

x=4,0: исходная=4,000000, восстановленная=4,000000, разница=0,000000

x=5,0: исходная=5,000000, восстановленная=5,000000, разница=0,000000

x=6,0: исходная=6,000000, восстановленная=6,000000, разница=0,000000

x=7,0: исходная=7,000000, восстановленная=7,000000, разница=0,000000

x=8,0: исходная=8,000000, восстановленная=8,000000, разница=0,000000

x=9,0: исходная=9,000000, восстановленная=9,000000, разница=0,000000

x=10,0: исходная=10,000000, восстановленная=10,000000, разница=0,000000

Размер файла сериализации: 236 байт