Лабораторная работа 4
по дисциплине «ООП»

Выполнила:
Филатова Анастасия Анатольевна,
студентка группы 6301-030301D,
институт информатики и кибернетики;

Самара 2025

Цель работы:

Расширить возможности пакета для работы с функциями одной переменной добавив интерфейсы и классы для аналитически заданных функций, а также методы ввода и вывода табулированных функций.

Задание 1

Добавлены конструкторы в ArrayTabulatedFunction и LinkedListTabulatedFunction, принимающие массивы FunctionPoint[]. Конструкторы выполняют проверки:

Количество точек ≥ 2

Точки упорядочены по возрастанию

```
package functions;

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

При нарушении условий выбрасывается IllegalArgumentException

Задание 2

Создан интерфейс Function с методами:

getLeftDomainBorder()

getRightDomainBorder()

getFunctionValue(double x)

Интерфейс TabulatedFunction теперь расширяет Function.

Задание 3

Создан пакет functions.basic с классами:

Exp (экспонента)

```
public class Exp implements Function {
    @Override
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }
}
```

Log (логарифм с произвольным основанием)

```
public class Log implements Function {
    private double base;

    public Log(double base) {
        if (base <= 0 || FunctionPoint.equals(base, 1)) {
            throw new IllegalArgumentException("Base must be positive and not equal to 1");
        }
        this.base = base;
    }

    @Override
    public double getLeftDomainBorder() {
        return 0;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }

    @Override
    public double getFunctionValue(double x) {
        if (x <= 0) {
            return Double.NaN;
        }
        return Math.log(x) / Math.log(base);
    }
}
```

Sin, Cos, Tan (тригонометрические)

```
package functions.basic;

public class Tan extends TrigonometricFunction {
    @Override
    public double getFunctionValue(double x) {
        return Math.tan(x);
    }
}
```

TrigonometricFunction (базовый абстрактный класс)

```
public abstract class TrigonometricFunction implements Function {
    @Override
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    @Override
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
}
```

Задание 4

Создан пакет functions.meta с классами для комбинирования:

Sum, Mult (арифметические операции)

Power (возведение в степень)

Scale, Shift (преобразования координат)

Composition (композиция)

Задание 5

Создан класс Functions со статическими фабричными методами.

Класс Functions реализован как утилитный (utility) класс с соблюдением следующих принципов:

1)Приватный конструктор - предотвращает создание экземпляров класса

2)Ключевое слово final - запрещает наследование

3)Только статические методы - все функциональность доступна без инстанцирования

Задание 6

Создан класс TabulatedFunctions с методом tabulate().

```
public static TabulatedFunction tabulate(Function function,
                                         double leftX,
                                         double rightX,
                                         int pointsCount) {
        if (leftX < function.getLeftDomainBorder() ||
            rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Tabulation interval is outside function
domain");
        }

        if (pointsCount < 2) {
            throw new IllegalArgumentException("At least 2 points required");
        }

        double[] xValues = new double[pointsCount];
        double[] yValues = new double[pointsCount];

        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            xValues[i] = leftX + i * step;
            yValues[i] = function.getFunctionValue(xValues[i]);
        }

        return new ArrayTabulatedFunction(xValues, yValues);
    }
```

Задание 7

Реализованы 4 метода:

outputTabulatedFunction() / inputTabulatedFunction() (бинарный формат)

```
// Задание 7: байтовые потоки
    public static void outputTabulatedFunction(TabulatedFunction function,
                                               OutputStream out) throws IOException {
        try (DataOutputStream dataOut = new DataOutputStream(out)) {
            int pointCount = function.getPointCount();
            dataOut.writeInt(pointCount);

            for (int i = 0; i < pointCount; i++) {
                dataOut.writeDouble(function.getPointX(i));
                dataOut.writeDouble(function.getPointY(i));
            }
        }
    }

    public static TabulatedFunction inputTabulatedFunction(InputStream in)
            throws IOException {
        try (DataInputStream dataIn = new DataInputStream(in)) {
            int pointCount = dataIn.readInt();

            double[] xValues = new double[pointCount];
            double[] yValues = new double[pointCount];

            for (int i = 0; i < pointCount; i++) {
                xValues[i] = dataIn.readDouble();
                yValues[i] = dataIn.readDouble();
            }

            return new ArrayTabulatedFunction(xValues, yValues);
        }
    }
```

writeTabulatedFunction() / readTabulatedFunction() (текстовый формат)

```
// Задание 7: символьные потоки
    public static void writeTabulatedFunction(TabulatedFunction function,
                                              Writer out) throws IOException {
        PrintWriter writer = new PrintWriter(out);
        int pointCount = function.getPointCount();
        writer.print(pointCount);

        for (int i = 0; i < pointCount; i++) {
            writer.print(" " + function.getPointX(i));
            writer.print(" " + function.getPointY(i));
        }
        writer.flush();
        // Не закрываем writer, чтобы не закрывать переданный out
    }

    public static TabulatedFunction readTabulatedFunction(Reader in)
            throws IOException {
        StreamTokenizer tokenizer = new StreamTokenizer(in);
        tokenizer.parseNumbers();

        // Читаем количество точек
```

```
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
            throw new IOException("Expected number of points");
        }
        int pointCount = (int) tokenizer.nval;

        if (pointCount < 2) {
            throw new IOException("At least 2 points required");
        }

        double[] xValues = new double[pointCount];
        double[] yValues = new double[pointCount];

        for (int i = 0; i < pointCount; i++) {
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Expected X value");
            }
            xValues[i] = tokenizer.nval;

            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("Expected Y value");
            }
            yValues[i] = tokenizer.nval;
        }

        return new ArrayTabulatedFunction(xValues, yValues);
    }
}
```

Задание 9

Реализована поддержка Serializable и Externalizable.

```
// Сериализация
try (ObjectOutputStream oos = new ObjectOutputStream(...)) {
    oos.writeObject(tabulatedFunction);
}
// Десериализация
try (ObjectInputStream ois = new ObjectInputStream(...)) {
    TabulatedFunction restored = (TabulatedFunction) ois.readObject();
}
```

Задание 8

Создан тестовый класс Main с проверками всех функций.

Вывод :

PS C:\Users\79170\Desktop\Мои проекты\lab1\Lab-4-2025> java -cp out Main

=== Lab Work #4 ===

1. Testing basic functions:

Sin and Cos from 0 to PI with step 0.1:

sin(0,0) = 0, cos(0,0) = 1

sin(0,1) = 0,0998, cos(0,1) = 0,995

sin(0,2) = 0,1987, cos(0,2) = 0,9801

sin(0,3) = 0,2955, cos(0,3) = 0,9553

sin(0,4) = 0,3894, cos(0,4) = 0,9211

sin(0,5) = 0,4794, cos(0,5) = 0,8776

sin(0,6) = 0,5646, cos(0,6) = 0,8253

sin(0,7) = 0,6442, cos(0,7) = 0,7648

sin(0,8) = 0,7174, cos(0,8) = 0,6967

sin(0,9) = 0,7833, cos(0,9) = 0,6216

sin(1,0) = 0,8415, cos(1,0) = 0,5403

sin(1,1) = 0,8912, cos(1,1) = 0,4536

sin(1,2) = 0,932, cos(1,2) = 0,3624

sin(1,3) = 0,9636, cos(1,3) = 0,2675

sin(1,4) = 0,9854, cos(1,4) = 0,17

sin(1,5) = 0,9975, cos(1,5) = 0,0707

sin(1,6) = 0,9996, cos(1,6) = -0,0292

sin(1,7) = 0,9917, cos(1,7) = -0,1288

sin(1,8) = 0,9738, cos(1,8) = -0,2272

sin(1,9) = 0,9463, cos(1,9) = -0,3233

sin(2,0) = 0,9093, cos(2,0) = -0,4161

sin(2,1) = 0,8632, cos(2,1) = -0,5048

sin(2,2) = 0,8085, cos(2,2) = -0,5885

sin(2,3) = 0,7457, cos(2,3) = -0,6663

sin(2,4) = 0,6755, cos(2,4) = -0,7374

sin(2,5) = 0,5985, cos(2,5) = -0,8011

sin(2,6) = 0,5155, cos(2,6) = -0,8569

sin(2,7) = 0,4274, cos(2,7) = -0,9041

sin(2,8) = 0,335, cos(2,8) = -0,9422

sin(2,9) = 0,2392, cos(2,9) = -0,971

sin(3,0) = 0,1411, cos(3,0) = -0,99

sin(3,1) = 0,0416, cos(3,1) = -0,9991

2. Testing tabulation:

Tabulated Sin and Cos (10 points):

Point 0: sin(0)=0, cos(0)=1

Point 1: sin(0,3491)=0,342, cos(0,3491)=0,9397

Point 2: sin(0,6981)=0,6428, cos(0,6981)=0,766

Point 3: sin(1,0472)=0,866, cos(1,0472)=0,5

Point 4: sin(1,3963)=0,9848, cos(1,3963)=0,1736

Point 5: sin(1,7453)=0,9848, cos(1,7453)=-0,1736

Point 6: sin(2,0944)=0,866, cos(2,0944)=-0,5

Point 7: sin(2,4435)=0,6428, cos(2,4435)=-0,766

Point 8: sin(2,7925)=0,342, cos(2,7925)=-0,9397

Point 9: sin(3,1416)=0, cos(3,1416)=-1

Comparison of exact and tabulated values:

x=0,0: sin=0,0000(tab=0,0000), cos=1,0000(tab=1,0000)

x=0,5: sin=0,4794(tab=0,4721), cos=0,8776(tab=0,8646)

x=1,0: sin=0,8415(tab=0,8358), cos=0,5403(tab=0,5360)

x=1,5: sin=0,9975(tab=0,9848), cos=0,0707(tab=0,0704)

x=2,0: sin=0,9093(tab=0,8981), cos=-0,4161(tab=-0,4117)

x=2,5: sin=0,5985(tab=0,5941), cos=-0,8011(tab=-0,7942)

x=3,0: sin=0,1411(tab=0,1387), cos=-0,9900(tab=-0,9755)

3. Testing function combinations:

sinB?(x) + cosB?(x) (should be в?? 1):

x=0,0: 1,000000

x=0,2: 0,970488

x=0,4: 0,984968

x=0,6: 0,975624

x=0,8: 0,975073

x=1,0: 0,985897

x=1,2: 0,970314

x=1,4: 0,998723

x=1,6: 0,970691

x=1,8: 0,984068

x=2,0: 0,976203

x=2,2: 0,974549

x=2,4: 0,986852

x=2,6: 0,970167

x=2,8: 0,997473

x=3,0: 0,970920

Effect of point count on accuracy:

5 points: average error = 0,095927

10 points: average error = 0,019511

20 points: average error = 0,004496

50 points: average error = 0,000682

4. Testing text input/output:

Function written to exp_function.txt

Comparison of original and read function:

x | original | read | difference

---

0 | 1,0000 | 1,0000 | 0,000000

1 | 2,7183 | 2,7183 | 0,000000

2 | 7,3891 | 7,3891 | 0,000000

3 | 20,0855 | 20,0855 | 0,000000

4 | 54,5982 | 54,5982 | 0,000000

5 | 148,4132 | 148,4132 | 0,000000

6 | 403,4288 | 403,4288 | 0,000000

7 | 1096,6332 | 1096,6332 | 0,000000

8 | 2980,9580 | 2980,9580 | 0,000000

9 | 8103,0839 | 8103,0839 | 0,000000

10 | 22026,4658 | 22026,4658 | 0,000000

Content of exp_function.txt:

11 0.0 1.0 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 1096.6331584284585 8.0 2980.9579870417283 9.0 8103.083927575384 10.0 22026.465794806718

5. Testing binary input/output:

Function written to log_function.bin

Comparison of original and read function:

x | original | read | difference

---

1 | 0,0000 | 0,0000 | 0,000000

2 | 0,6931 | 0,6931 | 0,000000

3 | 1,0986 | 1,0986 | 0,000000

4 | 1,3863 | 1,3863 | 0,000000

5 | 1,6094 | 1,6094 | 0,000000

6 | 1,7918 | 1,7918 | 0,000000

7 | 1,9459 | 1,9459 | 0,000000

8 | 2,0794 | 2,0794 | 0,000000

9 | 2,1972 | 2,1972 | 0,000000

10 | 2,3026 | 2,3026 | 0,000000

Binary file size: 164 bytes

6. Testing serialization:

a) Serializable:

Object serialized to function_serializable.ser

Deserialized function:

x=0: 0,0000 (expected: 0,0000)

x=1: 1,0000 (expected: 1,0000)

x=2: 2,0000 (expected: 2,0000)

x=3: 3,0000 (expected: 3,0000)

x=4: 4,0000 (expected: 4,0000)

x=5: 5,0000 (expected: 5,0000)

x=6: 6,0000 (expected: 6,0000)

x=7: 7,0000 (expected: 7,0000)

x=8: 8,0000 (expected: 8,0000)

x=9: 9,0000 (expected: 9,0000)

x=10: 10,0000 (expected: 10,0000)

File size: 236 bytes

b) Externalizable (ArrayTabulatedFunction):

Object serialized to function_externalizable.ser

Deserialized function:

Point 0: (0, 0)

Point 1: (1, 1)

Point 2: (2, 2)

Point 3: (3, 3)

Point 4: (4, 4)

Point 5: (5, 5)

Point 6: (6, 6)

Point 7: (7, 7)

Point 8: (8, 8)

Point 9: (9, 9)

Point 10: (10, 10)

File size: 236 bytes

File size comparison:

Serializable: 236 bytes

Externalizable: 236 bytes

Difference: 0 bytes