ЛАБОРАТОРНАЯ РАБОТА №4

Расширение пакета для работы с функциями одной переменной

по курсу
Объектно-ориентированное программирование

Группа 6204-010302D

Студент: С.0.Куропаткин.

Преподаватель: Борисов Дмитрий Сергеевич.

Задание 1
Конструкторы с массивом точек

В классы ArrayTabulatedFunction и LinkedListTabulatedFunction добавлены конструкторы, принимающие массив объектов FunctionPoint:

```
public LinkedListTabulatedFunction(FunctionPoint[] array) {
    if (array.length < 2)
        throw new IllegalArgumentException();

    for (int i = 0; i < array.length - 1; ++i)
        if (array[i].getX() >= array[i + 1].getX())
            throw new IllegalArgumentException();


    for (int i = 0; size < array.length; ++i) {
        head = addNodeToTail();
        head.point = new FunctionPoint(array[i]);
    }
    head = head.next;
}
```

```
public ArrayTabulatedFunction(FunctionPoint[] array) {

    if (array.length < 2) {
        throw new IllegalArgumentException();
    }

    size = array.length;

    for (int i = 1; i < array.length; ++i) {
        if (array[i-1].getX() >= array[i].getX()) {
            throw new IllegalArgumentException();
        }
    }

    points = new FunctionPoint[array.length + 10];
    System.arraycopy(array, 0, points, 0, array.length);
}
```

Задание 2
Интерфейс Function и наследование

Создан интерфейс Function, описывающий общие свойства всех функций:

```
package functions;

public interface Function {
    // Метод, возвращающий левую границу области определения
    double getLeftDomainBorder() throws InappropriateFunctionPointException;
    // Метод, возвращающий правую границу области определения
    double getRightDomainBorder() throws InappropriateFunctionPointException;
    // Метод, возвращающий значение функции в точке x с использованием
линейной интерполяции
    double getFunctionValue(double x) throws
InappropriateFunctionPointException;
}
```

Задание 3
Аналитически заданные функции
Создан пакет functions.basic с классами аналитических функций:

Класс Exp:
```
package functions.basic;

import functions.Function;

public class Exp implements Function {

    public double getLeftDomainBorder()  {
        return Double.NEGATIVE_INFINITY;
    }

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

    public double getFunctionValue(double x)  {
        return (Math.exp(x));
    }
}
```
Класс Log:
```
package functions.basic;

import functions.Function;

public class Log implements Function {
    private double base;
    public Log(double base) {
        this.base = base;
    }

    public double getLeftDomainBorder()  {
        return 0;
    }

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

    public double getFunctionValue(double x) {
        return (Math.log(x)/ Math.log((base)));
    }
}
```
Базовый класс TrigonometricFunction:
```
package functions.basic;

import functions.Function;

public class TrigonometricFunction implements Function {

    public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }

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

    public double getFunctionValue(double x) { return 0; }
}
```

Классы Sin, Cos, Tan:

```java
package functions.basic;

public class Sin extends TrigonometricFunction{

    public double getLeftDomainBorder()  {
        return super.getLeftDomainBorder();
    }

    public double getRightDomainBorder() {
        return super.getRightDomainBorder();
    }

    public double getFunctionValue(double x) {
        return (Math.sin(x));
    }
}
package functions.basic;

public class Cos extends TrigonometricFunction{

    public double getLeftDomainBorder()  {
        return super.getLeftDomainBorder();
    }

    public double getRightDomainBorder() {
        return super.getRightDomainBorder();
    }

    public double getFunctionValue(double x) {
        return (Math.cos(x));
    }
}
package functions.basic;

public class Tan extends TrigonometricFunction{

    public double getLeftDomainBorder() { return
super.getLeftDomainBorder(); }

    public double getRightDomainBorder() { return
super.getRightDomainBorder(); }

    public double getFunctionValue(double x) { return (Math.tan(x)); }
}
```

Задание 4
Функции-метаморфозы

Создан пакет functions.meta с классами для комбинирования функций:

Класс Sum:

```
package functions.meta;

import functions.Function;
import functions.InappropriateFunctionPointException;

public class Sum implements Function {

    private Function a, b;
    public Sum(Function a,Function b) { this.a = a; this.b = b; }

    public double getLeftDomainBorder() throws
InappropriateFunctionPointException {
        return (Math.max(a.getLeftDomainBorder(), b.getLeftDomainBorder()));
    }

    public double getRightDomainBorder() throws
InappropriateFunctionPointException {
        return (Math.min(a.getRightDomainBorder(), b.getRightDomainBorder()));
    }

    public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
        return (a.getFunctionValue(x) + b.getFunctionValue(x));
    }
}
```

Класс Scale:

```
package functions.meta;

import functions.Function;
import functions.InappropriateFunctionPointException;

public class Scale implements Function {
    private Function a;
    private double x, y;

    public Scale(Function a, double x, double y) { this.a = a; this.x = x;
this.y = y; }

    public double getLeftDomainBorder() throws
InappropriateFunctionPointException {
        return (a.getLeftDomainBorder() * x);
    }

    public double getRightDomainBorder() throws
InappropriateFunctionPointException {
        return (a.getRightDomainBorder() * x);
    }

    public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
        return (a.getFunctionValue(x / this.x) * this.y);
    }
}
```

Класс Mult:

```
package functions.meta;

import functions.Function;
import functions.InappropriateFunctionPointException;

public class Mult implements Function {

    private Function a, b;
    public Mult(Function a,Function b) { this.a = a; this.b = b; }

    public double getLeftDomainBorder() throws
InappropriateFunctionPointException {
        return (Math.max(a.getLeftDomainBorder(), b.getLeftDomainBorder()));
    }

    public double getRightDomainBorder() throws
InappropriateFunctionPointException {
        return (Math.min(a.getRightDomainBorder(), b.getRightDomainBorder()));
    }

    public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
        return (a.getFunctionValue(x) * b.getFunctionValue(x));
    }
}
```

Класс Power:

```
package functions.meta;

import functions.Function;
import functions.InappropriateFunctionPointException;

public class Power implements Function {
    private Function a;
    double n;
    public Power(Function a,double n) { this.a = a; this.n = n; }

    public double getLeftDomainBorder() throws
InappropriateFunctionPointException {
        return a.getLeftDomainBorder();
    }

    public double getRightDomainBorder() throws
InappropriateFunctionPointException {
        return a.getRightDomainBorder();
    }

    public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
        return (Math.pow(a.getFunctionValue(x), n));
    }
}
```

Класс Shift:

```
package functions.meta;

import functions.Function;
import functions.InappropriateFunctionPointException;

public class Shift implements Function {
    private Function a;
    private double x, y;

    public Shift(Function a, double x, double y) { this.a = a; this.x = x;
this.y = y; }

    public double getLeftDomainBorder() throws
InappropriateFunctionPointException {
        return (a.getLeftDomainBorder() + x);
    }

    public double getRightDomainBorder() throws
InappropriateFunctionPointException {
        return (a.getRightDomainBorder() + x);
    }

    public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
        return (a.getFunctionValue(x - this.x) + this.y);
    }
}
```

Класс Composition:

```
package functions.meta;

import functions.Function;
import functions.InappropriateFunctionPointException;

public class Composition implements Function {
    Function a, b;

    public Composition(Function a,Function b) { this.a = a; this.b = b; }

    public double getLeftDomainBorder() throws
InappropriateFunctionPointException {
        return a.getLeftDomainBorder();
    }

    public double getRightDomainBorder() throws
InappropriateFunctionPointException {
        return a.getRightDomainBorder();
    }

    public double getFunctionValue(double x) throws
InappropriateFunctionPointException {
        return (a.getFunctionValue(b.getFunctionValue(x)));
    }
}
```

Задание 5

Вспомогательный класс Functions

Создан класс Functions со статическими методами:

```java
package functions;

import functions.meta.*;

public class Functions {

    // Приватный конструктор длдя предотвращения создания объекта
    private Functions() {};

    public static Function shift(Function f, double shiftX, double shiftY){
        return new Shift(f, shiftX, shiftY);
    }
    public static Function scale(Function f, double scaleX, double scaleY) {
        return new Scale(f, scaleX, scaleY);
    }
    public static Function power(Function f,double power){
        return new Power(f, power);
    }
    public static Function sum(Function f1, Function f2){
        return new Sum(f1, f2);
    }
    public static Function mult(Function f1, Function f2){
        return new Mult(f1, f2);
    }
    public static Function composition(Function f1, Function f2){
        return new Composition(f1, f2);
    }
}
```

Задание 6
Табулирование функций

Создан класс TabulatedFunctions с методом табулирования:

```java
package functions;

import java.io.*;

public class TabulatedFunctions {

    // Приватный конструктор для предотвращения создания объекта
    private TabulatedFunctions() {};

    public static TabulatedFunction tabulate(Function function, double leftX,
double rightX, int pointsCount) throws InappropriateFunctionPointException {

        if (leftX < function.getLeftDomainBorder() || rightX >
function.getRightDomainBorder()) {
            throw new IllegalArgumentException();
        }

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        double interval = (Math.abs(rightX - leftX)) / (pointsCount - 1);

        for (int i = 0; i < pointsCount; ++i){
            points[i] = new FunctionPoint((leftX + i * interval),
function.getFunctionValue(leftX + i * interval));
        }
        return new ArrayTabulatedFunction(points);
    }
```

Задание 7

Ввод-вывод табулированных функций

Добавлены методы в класс TabulatedFunctions:

```java
public static void outputTabulatedFunction(TabulatedFunction function,
OutputStream out) throws IOException {

        DataOutputStream dataOut = new DataOutputStream(out);

        // Записываем количество точек
        dataOut.writeInt(function.getPointsCount());

        // Записываем значения координат точек
        for (int i = 0; i < function.getPointsCount(); ++i) {
            dataOut.writeDouble(function.getPointX(i));
            dataOut.writeDouble(function.getPointY(i));
        }

        // Закрываем поток
        dataOut.close();
    }

    public static TabulatedFunction inputTabulatedFunction(InputStream in)
throws IOException {

        DataInputStream dataIn = new DataInputStream(in);

        // Считываем количество точек
        int pointsCount = dataIn.readInt();

        // Создаем массив точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];

        // Считываем значения координат точек
        for (int i = 0; i < pointsCount; ++i) {
            points[i] = new FunctionPoint(dataIn.readDouble(),
dataIn.readDouble());
        }

        // Закрываем поток
        dataIn.close();

        // Создаем и возвращаем объект табулированной функции
        return new ArrayTabulatedFunction(points);
    }

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

        BufferedWriter writer = new BufferedWriter(out);

        // Записываем количество точек
        writer.write(Integer.toString(function.getPointsCount()));

        // Записываем значения координат точек
        for (int i = 0; i < function.getPointsCount(); ++i) {
            writer.write(' ');
            writer.write(Double.toString(function.getPointX(i)));
            writer.write(' ');
            writer.write(Double.toString(function.getPointY(i)));
        }
        // Закрываем поток
        writer.close();
    }

    public static TabulatedFunction readTabulatedFunction(Reader in) throws
IOException {

        StreamTokenizer tokenizer = new StreamTokenizer(in);

        // Переводим токен
        tokenizer.nextToken();

        // Считали кол-во точек
        int pointsCount = (int)tokenizer.nval;

        double x, y;
        FunctionPoint points[] = new FunctionPoint[pointsCount];

        // Считываем значения координат точек
        for(int i = 0; i < pointsCount; ++i){
            tokenizer.nextToken();
            x = tokenizer.nval;
            tokenizer.nextToken();
            y = tokenizer.nval;
            points[i] = new FunctionPoint(x, y);
        }
        // Создаем и возвращаем объект табулированной функции
        return new ArrayTabulatedFunction(points);
    }
}
```

Задание 8

Тестирование

Создан класс Main для тестирования:

```
import functions.*;
import functions.basic.*;
import functions.meta.*;
import java.io.*;

public class Main {
    public static void main(String[] args) throws
InappropriateFunctionPointException {

        // Создание объектов Sin и Cos
        Function sinFunction = new Sin();
        Function cosFunction = new Cos();

        // Вывод значений Sin и Cos на отрезке от 0 до п с шагом 0.1
        System.out.println("Вывод значений Sin и Cos на отрезке от 0 до 2п с
шагом 0.1:");
        printFunctionValues(sinFunction, 0, Math.PI, 0.1);
        printFunctionValues(cosFunction, 0, Math.PI, 0.1);

        // Табулирование Sin и Cos на отрезке от 0 до п с 10 точками
        TabulatedFunction sinTabulated =
TabulatedFunctions.tabulate(sinFunction, 0,  Math.PI, 10);
        TabulatedFunction cosTabulated =
TabulatedFunctions.tabulate(cosFunction, 0,  Math.PI, 10);
```

```
        // Вывод значений табулированных Sin и Cos на отрезке от 0 до π с
шагом 0.1
        System.out.println("Вывод значений табулированных Sin и Cos на
отрезке от 0 до 2π с шагом 0.1:");
        printTabulatedFunctionValues(sinTabulated, 0,  Math.PI, 0.1);
        printTabulatedFunctionValues(cosTabulated, 0,  Math.PI, 0.1);

        // Создание функции, являющейся суммой квадратов табулированных Sin и
Cos
        Function sumOfSquares = Functions.sum(Functions.power(sinTabulated,
2), Functions.power(cosTabulated, 2));

        // Вывод значений суммы квадратов на отрезке от 0 до π с шагом 0.1
        System.out.println("Вывод значений суммы квадратов на отрезке от 0 до
2π с шагом 0.1:");
        printFunctionValues(sumOfSquares, 0,  Math.PI, 0.1);



        // Создание объекта Exp
        Function expFunction = new Exp();

        // Табулирование экспоненты на отрезке от 0 до 10 с 11 точками
        TabulatedFunction expTabulated =
TabulatedFunctions.tabulate(expFunction, 0, 10, 11);

        // Запись табулированной экспоненты в файл

        try {
            FileWriter fileWriter = new FileWriter("exp_tabulated.txt");
            TabulatedFunctions.writeTabulatedFunction(expTabulated,
fileWriter);
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Чтение табулированной функции из файла
        try {
            FileReader fileReader = new FileReader("exp_tabulated.txt");
            TabulatedFunction readExpTabulated =
TabulatedFunctions.readTabulatedFunction(fileReader);
            fileReader.close();

            // Вывод и сравнение значений исходной и считанной функций на
отрезке от 0 до 10 с шагом 1
            System.out.println("Вывод и сравнение значений исходной и
считанной функций Exp на отрезке от 0 до 10 с шагом 1:");
            System.out.println("Исходная функция:");
            printTabulatedFunctionValues(expTabulated, 0, 10, 1);
            System.out.println("Считанная функция:");
            printTabulatedFunctionValues(readExpTabulated, 0, 10, 1);

        } catch (IOException e) {
            e.printStackTrace();
        }

        // Создание объекта Log
        Function logFunction = new Log(Math.E);

        // Табулирование логарифма на отрезке от 0 до 10 с 11 точками
        TabulatedFunction logTabulated =
TabulatedFunctions.tabulate(logFunction, 0.1, 10, 11);
```

```
        // Запись табулированного логарифма в файл
        try {
            FileOutputStream fileOut = new
FileOutputStream("log_tabulated.txt");
            TabulatedFunctions.outputTabulatedFunction(logTabulated, fileOut);
            fileOut.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Чтение табулированной функции из файла
        try {
            FileInputStream fileIn = new FileInputStream("log_tabulated.txt");
            TabulatedFunction readLogTabulated =
TabulatedFunctions.inputTabulatedFunction(fileIn);
            fileIn.close();

            // Вывод и сравнение значений исходной и считанной функций на
отрезке от 0 до 10 с шагом 1
            System.out.println("Вывод и сравнение значений исходной и
считанной функций Log на отрезке от 0 до 10 с шагом 1:");
            System.out.println("Исходная функция:");
            printTabulatedFunctionValues(logTabulated, 0.1, 10, 1);
            System.out.println("Считанная функция:");
            printTabulatedFunctionValues(readLogTabulated, 0.1, 10, 1);

        } catch (IOException e) {
            e.printStackTrace();
        }
        // Табулирование логарифма на отрезке от 0 до 10 с 11 точками
        TabulatedFunction logTabulatedSer =
TabulatedFunctions.tabulate(logFunction, 0.1, 10, 11);

        // Сериализация
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("log_tabulated_serializable.txt"))) {
            oos.writeObject(logTabulatedSer);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Десериализация
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("log_tabulated_serializable.txt"))) {
            TabulatedFunction readLogTabulated = (TabulatedFunction)
ois.readObject();

            // Вывод и сравнение значений исходной и считанной функций на
отрезке от 0 до 10 с шагом 1
            System.out.println("Вывод и сравнение значений исходной и
считанной функций Log на отрезке от 0 до 10 с шагом 1:");
            System.out.println("Исходная функция:");
            printTabulatedFunctionValues(logTabulatedSer, 0.1, 10, 1);
            System.out.println("Считанная функция:");
            printTabulatedFunctionValues(readLogTabulated, 0.1, 10, 1);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }


        // Табулирование логарифма на отрезке от 0 до 10 с 11 точками
        TabulatedFunction logTabulatedEx =
TabulatedFunctions.tabulate(logFunction, 0.1, 10, 11);

// Сериализация
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("log_tabulated_externalizable.txt"))) {
            oos.writeObject(logTabulatedEx);
        } catch (IOException e) {
            e.printStackTrace();
        }

// Десериализация
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("log_tabulated_externalizable.txt"))) {
            TabulatedFunction readLogTabulated = (TabulatedFunction)
ois.readObject();

            // Вывод и сравнение значений исходной и считанной функций на
отрезке от 0 до 10 с шагом 1
            System.out.println("Вывод и сравнение значений исходной и
считанной функций Log на отрезке от 0 до 10 с шагом 1:");
            System.out.println("Исходная функция:");
            printTabulatedFunctionValues(logTabulatedEx, 0.1, 10, 1);
            System.out.println("Считанная функция:");
            printTabulatedFunctionValues(readLogTabulated, 0.1, 10, 1);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    // Вспомогательный метод для вывода значений функции на отрезке с
заданным шагом
    private static void printFunctionValues(Function function, double from,
double to, double step) throws InappropriateFunctionPointException {
        for (double x = from; x <= to; x += step) {
            System.out.println("Function value at x = " + x + ": " +
function.getFunctionValue(x));
        }
    }

    // Вспомогательный метод для вывода значений табулированной функции на
отрезке с заданным шагом
    private static void printTabulatedFunctionValues(TabulatedFunction
tabulatedFunction, double from, double to, double step) throws
InappropriateFunctionPointException {
        for (double x = from; x <= to; x += step) {
            System.out.println("Tabulated function value at x = " + x + ": "
+ tabulatedFunction.getFunctionValue(x));
        }
    }

}
```

Задание 9

Сериализация

Реализована сериализация двумя способами:

```
// Сериализация
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("log_tabulated_serializable.txt"))) {
            oos.writeObject(logTabulatedSer);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Десериализация
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("log_tabulated_serializable.txt"))) {
            TabulatedFunction readLogTabulated = (TabulatedFunction)
ois.readObject();

            // Вывод и сравнение значений исходной и считанной функций на
отрезке от 0 до 10 с шагом 1
            System.out.println("Вывод и сравнение значений исходной и
считанной функций Log на отрезке от 0 до 10 с шагом 1:");
            System.out.println("Исходная функция:");
            printTabulatedFunctionValues(logTabulatedSer, 0.1, 10, 1);
            System.out.println("Считанная функция:");
            printTabulatedFunctionValues(readLogTabulated, 0.1, 10, 1);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }


        // Табулирование логарифма на отрезке от 0 до 10 с 11 точками
        TabulatedFunction logTabulatedEx =
TabulatedFunctions.tabulate(logFunction, 0.1, 10, 11);

// Сериализация
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("log_tabulated_externalizable.txt"))) {
            oos.writeObject(logTabulatedEx);
        } catch (IOException e) {
            e.printStackTrace();
        }

// Десериализация
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("log_tabulated_externalizable.txt"))) {
            TabulatedFunction readLogTabulated = (TabulatedFunction)
ois.readObject();

            // Вывод и сравнение значений исходной и считанной функций на
отрезке от 0 до 10 с шагом 1
            System.out.println("Вывод и сравнение значений исходной и
считанной функций Log на отрезке от 0 до 10 с шагом 1:");
            System.out.println("Исходная функция:");
            printTabulatedFunctionValues(logTabulatedEx, 0.1, 10, 1);
            System.out.println("Считанная функция:");
            printTabulatedFunctionValues(readLogTabulated, 0.1, 10, 1);

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
```