Лабораторная работа по ООП №4

Студент: Тарасова Полина

Группа: 6204-010302D

Задание 1

Я добавила конструкторы в классы ArrayTabulatedFunction и LinkedListTabulatedFunction, которые принимают сразу массив объектов FunctionPoint. Было важно, чтобы конструкторы проверяли, что в массива не меньше двух точек и правильный порядок по абсциссе. Это предотвратило бы некорректное создание функции. Я сделала копирование точек внутри конструктора, чтобы соблюсти инкапсуляцию. В итоге конструкторы корректно создают объекты, а при нарушении условий выбрасывается IllegalArgumentException.

```
public ArrayTabulatedFunction(FunctionPoint[] points) {
    if (points == null || points.length < 2) {
        throw new IllegalArgumentException("Количество точек < 2");
    }

    Arrays.sort(points, Comparator.comparingDouble(FunctionPoint::getX));
    for (int i = 0; i < points.length - 1; i++) {
        if (Math.abs(points[i + 1].getX() - points[i].getX()) <= EPS) {
            throw new IllegalArgumentException("Такая точка уже есть");
        }
    }
    this.points = new FunctionPoint[points.length];
    for (int i = 0; i < points.length; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
    this.pointsCount = points.length;
}
```

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {
    if (points == null || points.length < 2) {
        throw new IllegalArgumentException("Количество точек < 2");
    }
    Arrays.sort(points, Comparator.comparingDouble(FunctionPoint::getX));
    for (int i = 0; i < points.length - 1; i++) {
        if (Math.abs(points[i + 1].getX() - points[i].getX()) <= EPS) {
            throw new IllegalArgumentException("Такая точка уже есть");
        }
    }
    this.points = new FunctionPoint[points.length];
    for (int i = 0; i < points.length; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
    this.pointsCount = points.length;
}
```

Задание 2

Я создала интерфейс Function с методами getLeftDomainBorder(), getRightDomainBorder() и getFunctionValue(double x). Табулированные функции теперь частный случай, поэтому TabulatedFunction расширяет Function. Благодаря этому можно работать с разными типами функций через один интерфейс.

```
package functions;

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

Задание 3

В пакете functions.basic я создала классы Exp, Log, Sin, Cos и Tan. Для тригонометрических функций сделала базовый класс TrigonometricFunction, чтобы не дублировать методы получения границ области определения. Это оказалось удобно: наследование уменьшает количество кода и облегчает поддержку. В классе Log пришлось подумать о том, как задавать основание через конструктор. В итоге все функции можно легко использовать как объекты типа Function.

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);
    }
}
```

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);
    }
}
```

TrigonometricFunction:

```
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {
    public abstract double getFunctionValue(double x);

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

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

Sin:

```
package functions.basic;

import functions.Function;

public class Sin extends TrigonometricFunction{
    public double getFunctionValue(double x) {
        return Math.sin(x);
    }
}
```

Cos:

```
package functions.basic;
```

```
import functions.Function;

public class Cos extends TrigonometricFunction{
    public double getFunctionValue(double x) {
        return Math.cos(x);
    }
}
```

Tan:

```
package functions.basic;

import functions.Function;

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

Задание 4

Я создала классы для комбинирования функций в пакете functions.meta: Sum, Mult, Power, Scale, Shift, Composition. Нужно было продумать, как правильно вычислять границы области определения для суммы и произведения функций. При масштабировании и сдвиге важно было учитывать отрицательные коэффициенты. Сейчас функции корректно комбинируются и возвращают правильные значения.

Sum:

```
package functions.meta;

import functions.Function;

public class Sum implements Function{
    private Function f1;
    private Function f2;

    public Sum(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }

    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());

    }

    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(),
f2.getRightDomainBorder());
```

```
    }

    public double getFunctionValue(double x) {
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

Mult:

```
package functions.meta;

import functions.Function;

public class Mult implements Function{
    private Function f1;
    private Function f2;

    public Mult(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }

    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());

    }

    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(),
f2.getRightDomainBorder());

    }

    public double getFunctionValue(double x) {
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
}
```

Power:

```
package functions.meta;

import functions.Function;

public class Power implements Function{
    private Function f;
    private double n;

    public Power(Function f, double n) {
        this.f = f;
        this.n = n;
    }

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

    public double getRightDomainBorder() {
        return f.getRightDomainBorder();
    }
```

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

Scale:

```
import functions.Function;

public class Scale implements Function{
    private Function f;
    private double scaleY;
    private double scaleX;

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

    public double getLeftDomainBorder() {
        double right = f.getRightDomainBorder() * scaleX;
        double left = f.getLeftDomainBorder() * scaleX;
        return Math.min(left, right);
    }

    public double getRightDomainBorder() {
        double right = f.getRightDomainBorder() * scaleX;
        double left = f.getLeftDomainBorder() * scaleX;
        return Math.max(left, right);
    }

    public double getFunctionValue(double x) {
        return scaleY * f.getFunctionValue(x / scaleX);
    }
}
```

Shift:

```
package functions.meta;

import functions.Function;

public class Shift implements Function {
    private Function f;
    private double shiftY;
    private double shiftX;

    public Shift(Function f, double shiftY, double shiftX) {
        this.f = f;
        this.shiftY = shiftY;
        this.shiftX = shiftX;
    }

    public double getLeftDomainBorder() {
        double right = f.getRightDomainBorder() + shiftX;
        double left = f.getLeftDomainBorder() + shiftX;
        return Math.min(left, right);
    }

    public double getRightDomainBorder() {
        double right = f.getRightDomainBorder() + shiftX;
```

```
        double left = f.getLeftDomainBorder() + shiftX;
        return Math.max(left, right);
    }

    public double getFunctionValue(double x) {
        return f.getFunctionValue(x - shiftX) + shiftY;
    }
}
```

Composition:

```
package functions.meta;

import functions.Function;

public class Composition implements Function{
    private Function f1;
    private Function f2;

    public Composition(Function f1, Function f2) {
        this.f1 = f1;
        this.f2 = f2;
    }

    public double getLeftDomainBorder() {
        return f1.getLeftDomainBorder();
    }

    public double getRightDomainBorder() {
        return f1.getRightDomainBorder();
    }

    public double getFunctionValue(double x) {
        return f1.getFunctionValue(f2.getFunctionValue(x));
    }
}
```

Задание 5

Я создала класс Functions с методами shift(), scale(), power(), sum(), mult(), composition(). Это упрощает создание новых функций без прямого использования конструкторов классов из functions.meta. Сделала конструктор приватным, чтобы нельзя было создавать объект класса. Теперь статические методы корректно возвращают новые объекты функций.

```
package functions;

import functions.meta.Shift;
import functions.meta.Scale;
import functions.meta.Power;
import functions.meta.Sum;
import functions.meta.Mult;
import functions.meta.Composition;
```

```
public class Functions {

    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

Я создала класс TabulatedFunctions с методом tabulate(Function function, double leftX, double rightX, int pointsCount). Важно было проверять границы табулирования, чтобы не выйти за область определения функции. Также удобно возвращать объект типа TabulatedFunction, чтобы можно было использовать его везде, где требуется интерфейс Function. В итоге метод создаёт табулированные функции корректно и с равномерным шагом.

```
package functions;

import java.io.*;

public class TabulatedFunctions {

    private TabulatedFunctions() {}

    public static TabulatedFunction tabulate(Function function, double leftX,
double rightX, int pointsCount) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница >= правой");
        }
        double step = (rightX - leftX) / (pointsCount - 1);
        FunctionPoint[] points;
```

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

Задание 7

Я добавила методы записи и чтения табулированных функций в TabulatedFunctions:

- байтовые потоки (outputTabulatedFunction, inputTabulatedFunction),
- символьные потоки (writeTabulatedFunction, readTabulatedFunction).

Я решила не закрывать потоки внутри методов, потому что поток может использоваться дальше. С IOException сделала выброс исключения наружу, чтобы вызывающий код решал, что делать. Функции корректно сохраняются и считываются из файлов и потоков, значения совпадают с исходными.

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

    public static TabulatedFunction inputTabulatedFunction(InputStream in)
throws IOException {
        DataInputStream dataIn = new DataInputStream(in);
        int pointsCount = dataIn.readInt();
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек < 2");
        }
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            double x = dataIn.readDouble();
            double y = dataIn.readDouble();
            points[i] = new FunctionPoint(x, y);
        }
        return new ArrayTabulatedFunction(points);
    }
```

```
    public static void writeTabulatedFunction(TabulatedFunction function,
Writer out) throws IOException {
        out.write(String.valueOf(function.getPointsCount()));
        out.write(" ");
        for (int i = 0; i < function.getPointsCount(); i++) {
            out.write(String.valueOf(function.getPointX(i)));
            out.write(" ");
            out.write(String.valueOf(function.getPointY(i)));
            out.write(" ");
        }
        out.write("\n");
        out.flush();
    }

    public static TabulatedFunction readTabulatedFunction(Reader in) throws
IOException{
        StreamTokenizer st = new StreamTokenizer(in);
        st.nextToken();
        int pointsCount = (int) st.nval;
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек < 2");
        }
        for (int i = 0; i < pointsCount; i++) {
            st.nextToken();
            double x = st.nval;
            st.nextToken();
            double y = st.nval;
            points[i] = new FunctionPoint(x, y);
        }
        return new ArrayTabulatedFunction(points);
    }
}
```

Задание 8

Я проверила работу функций:

- вывела Sin и Cos на отрезке от 0 до π с шагом 0,1;
- создала табулированные аналоги на 10 точках и сравнила с исходными;
- создала функцию суммы квадратов табулированных синуса и косинуса;
- табулировала Exp и Log, сохранила в файлы и считала обратно.

Увеличение числа точек улучшает точность приближения. Формат текстового файла удобен для просмотра человеком, а бинарный — экономит место и быстрее читается программой. Результаты совпадают с ожиданиями, файлы читаются корректно,

приближения табулированных функций соответствуют аналитическим значениям.

```
Function sin = new Sin();
Function cos = new Cos();
for (double i = 0; i <= Math.PI + EPS; i += 0.1) {
    System.out.println("x = " + i + ", sin(x) = " + sin.getFunctionValue(i) +
", cos(x) = " + cos.getFunctionValue(i));
}


TabulatedFunction tabSin = TabulatedFunctions.tabulate(sin, 0, Math.PI, 10);
TabulatedFunction tabCos = TabulatedFunctions.tabulate(cos, 0, Math.PI, 10);
double right = tabSin.getRightDomainBorder();
double left = tabSin.getLeftDomainBorder();
double step = (right - left) / 10;
for (double i = left; i <= right + EPS; i += step) {
    System.out.println("x = " + i + ", sin(x) = " + sin.getFunctionValue(i) +
", tabsin(x) = " + tabSin.getFunctionValue(i));
    System.out.println("x = " + i + ", cos(x) = " + cos.getFunctionValue(i) +
", tabcos(x) = " + tabCos.getFunctionValue(i));
}

Function tabSin2 = Functions.power(tabSin, 2);
Function tabCos2 = Functions.power(tabCos, 2);
Function sumSquares = Functions.sum(tabSin2, tabCos2);
for (double i = left; i <= right + EPS; i += step) {
    System.out.println("x = " + i + ", sin^2 + cos^2 = " +
sumSquares.getFunctionValue(i));
}

Function exp = new Exp();
TabulatedFunction tabExp = TabulatedFunctions.tabulate(exp, 0, 10, 11);
try (Writer out = new FileWriter("exp.txt")) {
    TabulatedFunctions.writeTabulatedFunction(tabExp, out);
} catch (IOException e) {
    e.printStackTrace();
}
TabulatedFunction readExp = null;
try (Reader in = new FileReader("exp.txt")) {
    readExp = TabulatedFunctions.readTabulatedFunction(in);
} catch (IOException e) {
    e.printStackTrace();
}

for (double i = 0; i <= 10; i++) {
    System.out.println("x = " + i + ", exp(x) = " + exp.getFunctionValue(i) +
", tabExp(x) = " + tabExp.getFunctionValue(i) + ", readExp(x) = " +
readExp.getFunctionValue(i));
}

Function log = new Log(Math.E);
TabulatedFunction tabLog = TabulatedFunctions.tabulate(log, 0, 10, 11);
try (OutputStream out = new FileOutputStream("log.txt")) {
    TabulatedFunctions.outputTabulatedFunction(tabLog, out);
} catch (IOException e) {
    e.printStackTrace();
}
TabulatedFunction inputLog = null;
try (InputStream in = new FileInputStream("log.txt")) {
    inputLog = TabulatedFunctions.inputTabulatedFunction(in);
} catch (IOException e) {
```

```
    e.printStackTrace();
}
for (double i = 0.1; i <= 10; i++) {
    System.out.println("x = " + i + ", log(x) = " + log.getFunctionValue(i) +
", tabLog(x) = " + tabLog.getFunctionValue(i) + ", readLog(x) = " +
inputLog.getFunctionValue(i));
}
```

Задание 9

Я сделала так, чтобы TabulatedFunction была сериализуемой через Serializable и Externalizable. Serializable проще в реализации, но Externalizable даёт больше контроля над записью и чтением данных. Я проверила оба способа, сериализовав табулированный логарифм экспоненты. Объекты корректно сохраняются и считываются, значения совпадают с исходными. При использовании Externalizable файл получается чуть компактнее, и можно явно управлять порядком данных.

```
Function logExp = new Function() {
    @Override
    public double getLeftDomainBorder() {
        return exp.getLeftDomainBorder();
    }

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

    @Override
    public double getFunctionValue(double x) {
        return Math.log(exp.getFunctionValue(x));
    }
};
TabulatedFunction tabLogExp = TabulatedFunctions.tabulate(logExp, 0.1, 10,
11);
try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("logExp.ser"))) {
    out.writeObject(tabLogExp);
} catch (IOException e) {
    e.printStackTrace();
}
TabulatedFunction readTabLogExp = null;
try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream("logExp.ser"))) {
    readTabLogExp = (TabulatedFunction) in.readObject();
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
for (double i = 0.1; i <= 10; i++) {
    System.out.println("x = " + i + ", log(exp(x)) = " +
logExp.getFunctionValue(i) + ", tabLogExp(x) = " +
tabLogExp.getFunctionValue(i) + ", readTabLogExp(x) = " +
```

```
readTabLogExp.getFunctionValue(i));
}
```