Лабораторная работа № 4

Выполнила: Оленина Арина Игоревна
группа 6204-010302D

Оглавление

Задание 1

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction я добавила конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint. Если точек задано меньше двух, или если точки в массиве не упорядочены по значению абсциссы, конструкторы выбрасывают исключение IllegalArgumentException. При написании конструкторов я обеспечила инкапсуляцию (в конструкторах ArrayTabulatedFunction и LinkedListTabulatedFunction создала копию каждой точки)
Результат:

```
public ArrayTabulatedFunction(FunctionPoint[] points) {
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек
меньше двух");
    }
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i].getX() >= points[i + 1].getX()) {
            throw new IllegalArgumentException("Точки не
упорядочены по значению абсциссы");
        }
    }

    this.pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount + 10]; // Запас
места

    for (int i = 0; i < pointsCount; i++) {
        this.points[i] = new FunctionPoint(points[i]); //
Создаем копию точки
    }

public LinkedListTabulatedFunction(FunctionPoint[] points) {
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек
меньше двух");
    }
    for (int i = 0; i < points.length - 1; i++) { // Проверка
упорядоченности точек по X
        if (points[i].getX() >= points[i + 1].getX()) {
            throw new IllegalArgumentException("Точки не
упорядочены по значению абсциссы");
        }
    }
    // Инициализация списка
    head = new FunctionNode(null);
    head.next = head;
    head.prev = head;
    this.pointsCount = 0;
    for (FunctionPoint point : points) { // Добавление точек в
список
```

```
        addNodeToTail().point = new FunctionPoint(point); //
Создаем копию
    }
}
```

Задание 2

В пакете functions создала интерфейс Function, описывающий функции одной переменной и содержащий следующие методы:
• public double getLeftDomainBorder() – возвращает значение левой границы области определения функции;
• public double getRightDomainBorder() – возвращает значение правой границы области определения функции;
• public double getFunctionValue(double x) – возвращает значение функции в заданной точке.
Исключила соответствующие методы из интерфейса TabulatedFunction и сделала так, чтобы он расширял интерфейс Function (public interface TabulatedFunction extends Function{}). Теперь табулированные функции будут частным случаем функций одной переменной.

```
package functions;

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

Задание 3

Создала пакет functions.basic, в котором описала классы ряда функций, заданных аналитически.
Создала в пакете публичный класс Exp, объекты которого вычисляют значение экспоненты. Класс должен реализует интерфейс Function. Для вычисления экспоненты использовала метод Math.exp(), а для возвращения значений границ области определения – константы из класса Double.
Аналогично, создала класс Log, объекты которого вычисляют значение логарифма по заданному основанию. Основание передается как параметр конструктора. Для вычисления логарифма использовала метод Math.log().
Создала абстрактный класс TrigonometricFunction, реализующий интерфейс Function и описывающий методы получения границ области определения. Далее создала наследующие от него публичные классы Sin, Cos и Tan, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса. Для получения значений использовала методы Math.sin(), Math.cos() и Math.tan().
Результат:

```
package functions.basic;
import functions.Function;
```

```java
public class Exp implements Function{
    @Override
    public double getLeftDomainBorder() { // Левая граница
        return Double.NEGATIVE_INFINITY;
    }
    @Override
    public double getRightDomainBorder() { // Правая граница
        return Double.POSITIVE_INFINITY;
    }
    @Override
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }
}
```

```java
package functions.basic;
import functions.Function;

public class Log implements Function {
    private double base;

    public Log(double base) {
        if (base <= 0 || base == 1) {
            throw new IllegalArgumentException("Основание
логарифма должно быть положительным и не равным 1");
        }
        this.base = base;
    }

    @Override
    public double getLeftDomainBorder() { // Левая граница
        return 0.0; // Логарифм определен для x > 0
    }

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

    @Override
    public double getFunctionValue(double x) {
        if (x <= 0) {
            return Double.NaN; // Логарифм не определен для
неположительных x
        }
        return Math.log(x) / Math.log(base); // Формула смены
основания (то есть ln(x)/ln(base) = log base (x)
    }

    public double getBase() {
        return base;
    }
}
```

```java
package functions.basic;
import functions.Function;

public abstract class TrigonometricFunction implements Function{
    @Override
    public double getLeftDomainBorder() { // Левая граница
        return Double.NEGATIVE_INFINITY;
    }
    @Override
    public double getRightDomainBorder() { // Правая граница
        return Double.POSITIVE_INFINITY;
    }
    @Override
    public abstract double getFunctionValue(double x);
}
package functions.basic;

public class Cos extends TrigonometricFunction {
    @Override
    public double getFunctionValue(double x) {
        return Math.cos(x);
    }
}
package functions.basic;

public class Sin extends TrigonometricFunction {
    @Override
    public double getFunctionValue(double x) {
        return Math.sin(x);
    }
}
package functions.basic;

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

Задание 4

Затем я создала пакет functions.meta, в нём описала классы функций, позволяющие комбинировать функции.
Создала класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций. Класс реализует интерфейс Function. Конструктор класса получает ссылки типа Function на объекты суммируемых функций, а область определения функции получается как пересечение областей определения исходных функций.

```java
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;
    }
    @Override
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(),
f2.getLeftDomainBorder()); // Для определения левой границы
нужно искать максимум из границ
    }

    @Override
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(),
f2.getRightDomainBorder()); // Для определения правой границы
нужно искать минимум из границ
    }
    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x >
getRightDomainBorder()) { // Проверка на диапазон
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```

Аналогично, создала класс Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций.

```java
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;
    }
    @Override
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(),
f2.getLeftDomainBorder()); // Для определения левой границы
нужно искать максимум из границ
    }
```

```
    @Override
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(),
f2.getRightDomainBorder()); // Для определения правой границы
нужно искать минимум из границ
    }
    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x >
getRightDomainBorder()) { // Проверка на диапазон
            return Double.NaN;
        }
        return f1.getFunctionValue(x) * f2.getFunctionValue(x);
    }
}
```

Создала класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции. Конструктор класса получает ссылку на объекты базовой функции и степень, в которую должны возводиться её значения.

```
package functions.meta;
import functions.Function;

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

    public Power(Function f, double pow) {
        this.f = f;
        this.pow = pow;
    }
    @Override
    public double getLeftDomainBorder() { // Левая граница
        return f.getLeftDomainBorder();
    }

    @Override
    public double getRightDomainBorder() { // Правая граница
        return f.getRightDomainBorder();
    }
    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x >
getRightDomainBorder()) { // Проверка на диапазон
            return Double.NaN;
        }
        return Math.pow(f.getFunctionValue(x), pow);
    }
}
```

Создала класс Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат. Конструктор класса получает ссылку на объект исходной функции, а также коэффициенты масштабирования вдоль оси абсцисс и оси ординат. Область определения функции получается из области определения исходной функции масштабированием вдоль оси абсцисс, а значение функции – масштабированием значения исходной функции вдоль оси ординат. Коэффициенты масштабирования могут быть отрицательными.

```
package functions.meta;

import functions.Function;

public class Scale implements Function {
    private Function f;
    private double kX;
    private double kY;

    public Scale(Function f, double kX, double kY) {
        this.f = f;
        this.kX = kX;
        this.kY = kY;
    }

    @Override
    public double getLeftDomainBorder() { // Левая граница
        if (kX > 0) {
            return f.getLeftDomainBorder() / kX;
        } else if (kX < 0) {
            return f.getRightDomainBorder() / kX;
        } else {
            return Double.NaN; // Масштаб 0 недопустим
        }
    }

    @Override
    public double getRightDomainBorder() { // Правая граница
        if (kX > 0) {
            return f.getRightDomainBorder() / kX;
        } else if (kX < 0) {
            return f.getLeftDomainBorder() / kX;
        } else {
            return Double.NaN;
        }
    }

    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x >
getRightDomainBorder()) {
            return Double.NaN;
        }
        return kY * f.getFunctionValue(x * kX);
```

```
    }
}
```

Аналогично, создала класс Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат.

```
package functions.meta;

import functions.Function;

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

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

    @Override
    public double getLeftDomainBorder() { // Левая граница
        return f.getLeftDomainBorder() - shiftX;
    }

    @Override
    public double getRightDomainBorder() { // Правая граница
        return f.getRightDomainBorder() - shiftX;
    }

    @Override
    public double getFunctionValue(double x) {
        if (x + shiftX < getLeftDomainBorder() || x + shiftX >
getRightDomainBorder()) {
            return Double.NaN;
        }
        return f.getFunctionValue(x + shiftX) + x + shiftY;
    }
}
```

Также создала класс Composition, объекты которого описывают композицию двух исходных функций. Конструктор класса получает ссылки на объекты первой и второй функции. Область определения функции можно считать совпадающей с областью определения исходной функции (хотя математически это не всегда так).

```
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;
    }
    @Override
    public double getLeftDomainBorder() {
        return f1.getLeftDomainBorder(); // Левая граница
    }

    @Override
    public double getRightDomainBorder() {
        return f1.getRightDomainBorder(); // Правая граница
    }
    @Override
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x >
getRightDomainBorder()) { // Проверка на диапазон
            return Double.NaN;
        }
        return f2.getFunctionValue(f1.getFunctionValue(x));
    }
}
```

Задание 5

В пакете functions создала класс Functions, содержащий вспомогательные статические методы для работы с функциями. Сделала так, чтобы в программе вне этого класса нельзя было создать его объект. Класс содержит следующие методы:

- public static Function shift(Function f, double shiftX, double shiftY) – возвращает объект функции, полученной из исходной сдвигом вдоль осей;
- public static Function scale(Function f, double scaleX, double scaleY) – возвращает объект функции, полученной из исходной масштабированием вдоль осей;
- public static Function power(Function f, double power) – возвращает объект функции, являющейся заданной степенью исходной;
- public static Function sum(Function f1, Function f2) – возвращает объект функции, являющейся суммой двух исходных;
- public static Function mult(Function f1, Function f2) – возвращает объект функции, являющейся произведением двух исходных;
- public static Function composition(Function f1, Function f2) – возвращает объект функции, являющейся композицией двух исходных.

При написании методов следует воспользоваться созданными ранее классами из пакета functions.meta.
Результат:

```
package functions;
import functions.meta.*;

public class Functions {
    private Functions() {}; // Приватный конструктор для
предотвращения создания объекта

    public static Function shift(Function f, double shiftX,
double shiftY){ //возвращает объект функции, полученной из
исходной сдвигом вдоль осей
        return new Shift(f, shiftX, shiftY);
    };
    public static Function scale(Function f, double kX, double
kY){ //возвращает объект функции, полученной из исходной
масштабированием вдоль осей
        return new Scale(f, kX, kY);
    };
    public static Function power(Function f, double pow) {
//возвращает объект функции, являющейся заданной степенью
исходной
        return new Power(f, pow);
    };
    public static Function sum(Function f1, Function f2) {
//возвращает объект функции, являющейся суммой двух исходных
        return new Sum(f1, f2);
    };
    public static Function mult(Function f1, Function f2) {
//возвращает объект функции, являющейся произведением двух
исходных
        return new Mult(f1, f2);
    };
    public static Function composition(Function f1, Function f2)
{ //возвращает объект функции, являющейся композицией двух
исходных
        return new Composition(f1, f2);
    };
}
```

Задание 6

В пакете functions создала класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями. Сделала так, чтобы в программе вне этого класса нельзя было создать его объект.
Описала в классе метод public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount), получающий функцию и

возвращающий её табулированный аналог на заданном отрезке с заданным количеством точек.
Если указанные границы для табулирования выходят за область определения функции, метод выбрасывает исключение IllegalArgumentException.
Поскольку метод возвращает ссылку интерфейсного типа, можно возвращать объект любого из классов, реализующих этот интерфейс (я возвращаю объект класса ArrayTabulatedFunction) .

```
package functions;

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

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        // Проверки
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы выходят за область определения функции");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        FunctionPoint[] points = new FunctionPoint[pointsCount]; // Создание массива точек
        double step = (rightX - leftX) / (pointsCount - 1);

        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

В класс TabulatedFunctions добавила следующие методы.
Метод вывода табулированной функции в байтовый поток public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) выводит в указанный поток значения, по которым потом можно будет восстановить

табулированную функцию, а именно количество точек в ней и значения координат точек.
Метод ввода табулированной функции из байтового потока public static TabulatedFunction inputTabulatedFunction(InputStream in) считывает из указанного потока данные о табулированной функции, создает и настраивает её объект и возвращает его из метода.
Метод записи табулированной функции в символьный поток public static void writeTabulatedFunction(TabulatedFunction function, Writer out) в указанный поток выводит значения, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек (значения записываются в строку и разделяются пробелами).
Метод чтения табулированной функции из символьного потока public static TabulatedFunction readTabulatedFunction(Reader in) считывает из указанного потока данные о табулированной функции, создает и настраивает её объект и возвращает его из метода.
При написании методов в первых трёх случаях я воспользовалась потоками-обёртками, облегчающими ввод и вывод данных в требующейся форме, а в четвёртом случае – классом StreamTokenizer.
Поскольку методы ввода и чтения возвращают ссылку интерфейсного типа, можно возвращать объект любого из классов, реализующих этот интерфейс(я вернула объект класса ArrayTabulatedFunction).

Я объявила методы с throws IOException, так как обработка ошибок ввода-вывода – ответственность вызванного кода. Я также не стала закрывать потоки внутри этих методов, так как поток может еще использоваться дальше.

```
public static void outputTabulatedFunction(TabulatedFunction
function, OutputStream out) throws IOException { //Метод вывода
табулированной функции в байтовый поток
    DataOutputStream dataOut = new DataOutputStream(out);

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

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

    dataOut.flush(); // Не закрываем поток, чтобы пользователь
мог продолжать с ним работать
}

public static TabulatedFunction
inputTabulatedFunction(InputStream in) throws IOException {
//Метод ввода табулированной функции из байтового потока
    DataInputStream dataIn = new DataInputStream(in);
```

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

    // Читаем координаты точек
    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 { //Метод записи
табулированной функции в символьный поток
    PrintWriter writer = new PrintWriter(out);
    // Записываем количество точек
    writer.print(function.getPointsCount());
    writer.print(" ");
    // Записываем координаты точек через пробел
    for (int i = 0; i < function.getPointsCount(); i++) {
        writer.print(function.getPointX(i));
        writer.print(" ");
        writer.print(function.getPointY(i));
        if (i < function.getPointsCount() - 1) {
            writer.print(" ");
        }
    }
    writer.flush(); // Не закрываем поток
}

public static TabulatedFunction readTabulatedFunction(Reader in)
throws IOException { //Метод чтения табулированной функции из
символьного потока
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    // Читаем количество точек
    tokenizer.nextToken();
    int pointsCount = (int) tokenizer.nval;
    // Читаем координаты точек
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        tokenizer.nextToken();
        double x = tokenizer.nval;

        tokenizer.nextToken();
        double y = tokenizer.nval;

        points[i] = new FunctionPoint(x, y);
    }

    return new ArrayTabulatedFunction(points);
}
```

Задание 8

Затем я проверила работу написанных классов

Создала по одному объекту классов Sin и Cos, вывела в консоль значения этих функций на отрезке от 0 до π с шагом 0,1.
С помощью метода TabulatedFunctions.tabulate() создала табулированные аналоги этих функций на отрезке от 0 до π с 10 точками. Вывела в консоль значения этих функций на отрезке от 0 до π с шагом 0,1 и сравнила со значениями исходных функций.
С помощью методов класса Functions создала объект функции, являющейся суммой квадратов табулированных аналогов синуса и косинуса. Вывела в консоль значения этой функции на отрезке от 0 до π с шагом 0,1. Изменила количество точек в табулированных аналогах (5 и 20 точек) и исследовала, как при этом изменяется результирующая функция. Чем больше точек, тем меньше значение погрешности и тем ближе значение функции к единице. При 5 точках погрешность достаточно велика, а при 20 – минимальна.
С помощью метода TabulatedFunctions.tabulate() создала табулированный аналог экспоненты на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.writeTabulatedFunction() вывела его в файл exp_function.txt. Далее с помощью
метода TabulatedFunctions.readTabulatedFunction() считала табулированную функцию из этого файла. Вывела и сравнила значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
С помощью метода TabulatedFunctions.tabulate() создала табулированный аналог логарифма по натуральному основанию на отрезке от 0 до 10 с 11 точками. С помощью
метода TabulatedFunctions.outputTabulatedFunction() вывела его в файл log_function.bin. Далее с помощью
метода TabulatedFunctions.inputTabulatedFunction() считала табулированную функцию из этого файла. Вывела и сравнила значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
Изучила содержимое всех получаемых файлов.
Преимущества и недостатки форматов:

Текстовый формат

+ можно открыть в любом текстовом редакторе и прочитать содержимое

- больший размер файла (235 байт)

- медленнее чтение и запись

Бинарный формат

+ меньший размер файла, достаточно компактно (180 байт)

+ быстрее чтение и запись

+ нет потерь при преобразовании

- нечитаемый для человека

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

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

        // Объекты классов Sin и Cos
        Function sinFunction = new Sin();
        Function cosFunction = new Cos();

        // Значения этих функций на отрезке от 0 до π с шагом
0,1
        System.out.println("Sin от 0 до π с шагом 0.1:");
        printFunctionValues(sinFunction, 0, Math.PI, 0.1);

        System.out.println("\nCos от 0 до π с шагом 0.1:");
        printFunctionValues(cosFunction, 0, Math.PI, 0.1);

        // Табулированные аналоги этих функций на отрезке от 0
до π с 10 точками
        TabulatedFunction tabulatedSin =
TabulatedFunctions.tabulate(sinFunction, 0, Math.PI, 10);
        TabulatedFunction tabulatedCos =
TabulatedFunctions.tabulate(cosFunction, 0, Math.PI, 10);

        // Значения этих функций на отрезке от 0 до π с шагом
0,1
        System.out.println("\nТабулированный Sin (10 точек) от 0
до π с шагом 0.1:");
        printFunctionValues(tabulatedSin, 0, Math.PI, 0.1);

        System.out.println("\nТабулированный Cos (10 точек) от 0
до π с шагом 0.1:");
        printFunctionValues(tabulatedCos, 0, Math.PI, 0.1);

        // Объект функции, являющейся суммой квадратов
табулированных аналогов синуса и косинуса
        Function sinSquared = Functions.power(tabulatedSin, 2);
        Function cosSquared = Functions.power(tabulatedCos, 2);
        Function sumOfSquares = Functions.sum(sinSquared,
cosSquared);

        // Значения этой функций на отрезке от 0 до π с шагом
```

```
0,1
        System.out.println("\nСумма квадратов табулированных Sin
и Cos (10 точек) от 0 до π с шагом 0.1:");
        printFunctionValues(sumOfSquares, 0, Math.PI, 0.1);

        // Тест с 5 точками
        System.out.println(("Тест с 5 точками"));
        TabulatedFunction sin5 = TabulatedFunctions.tabulate(new
Sin(), 0, Math.PI, 5);
        TabulatedFunction cos5 = TabulatedFunctions.tabulate(new
Cos(), 0, Math.PI, 5);
        Function sum5 = Functions.sum(Functions.power(sin5, 2),
Functions.power(cos5, 2));
        printFunctionValues(sum5, 0, Math.PI, 0.1);

        // Тест с 20 точками
        System.out.println(("Тест с 20 точками"));
        TabulatedFunction sin20 =
TabulatedFunctions.tabulate(new Sin(), 0, Math.PI, 20);
        TabulatedFunction cos20 =
TabulatedFunctions.tabulate(new Cos(), 0, Math.PI, 20);
        Function sum20 = Functions.sum(Functions.power(sin20,
2), Functions.power(cos20, 2));
        printFunctionValues(sum20, 0, Math.PI, 0.1);

        // Работа с файлами (экспонента)
        TabulatedFunction expTabulated =
TabulatedFunctions.tabulate(new Exp(), 0, 10, 11);
        // Запись экспоненты
        try (FileWriter writer = new
FileWriter("exp_function.txt")) {

TabulatedFunctions.writeTabulatedFunction(expTabulated, writer);
        }

        // Чтение
        TabulatedFunction readExp;
        try (FileReader reader = new
FileReader("exp_function.txt")) {
            readExp =
TabulatedFunctions.readTabulatedFunction(reader);
        }

        System.out.println("Сравнение табулированной и считанной
из файла экспоненты:");
        compareFunctions(expTabulated, readExp, 0, 10, 1);

        // Работа с файлами (логарифм)
        TabulatedFunction logTabulated =
TabulatedFunctions.tabulate(new Log(Math.E), 1, 10, 11);

        // Запись
        try (FileOutputStream out = new
```

```
FileOutputStream("log_function.bin")) {

TabulatedFunctions.outputTabulatedFunction(logTabulated, out);
        }

        // Чтение
        TabulatedFunction readLog;
        try (FileInputStream in = new
FileInputStream("log_function.bin")) {
            readLog =
TabulatedFunctions.inputTabulatedFunction(in);
        }

        System.out.println("Сравнение табулированного и
считанного из файла логарифма:");
        compareFunctions(logTabulated, readLog, 1, 10, 1);
    }

    private static void printFunctionValues(Function function,
double from, double to, double step) {
        for (double x = from; x <= to + 1e-10; x += step) {
            double value = function.getFunctionValue(x);
            if (!Double.isNaN(value)) {
                System.out.printf("x = " + x + " y = " + value +
"\n");
            }
        }
    }

    private static void compareFunctions(Function f1, Function
f2, double from, double to, double step) {
        for (double x = from; x <= to + 1e-10; x += step) {
            double v1 = f1.getFunctionValue(x);
            double v2 = f2.getFunctionValue(x);
            if (!Double.isNaN(v1) && !Double.isNaN(v2)) {
                System.out.printf("x = " + x + " первая = " + v1
+ " вторая = " + v2 + "\n");
            }
        }
    }

    private static void printFileContent(String filename) {
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Ошибка чтения файла: " +
e.getMessage());
        }
```

```
    }
}
```

Вывод следующий:

```
Sin от 0 до п с шагом 0.1:
x = 0.0 y = 0.0
x = 0.1 y = 0.09983341664682815
x = 0.2 y = 0.19866933079506122
x = 0.30000000000000004 y = 0.2955202066613396
x = 0.4 y = 0.3894183423086505
x = 0.5 y = 0.479425538604203
x = 0.6 y = 0.5646424733950354
x = 0.7 y = 0.644217687237691
x = 0.7999999999999999 y = 0.7173560908995227
x = 0.8999999999999999 y = 0.7833269096274833
x = 0.9999999999999999 y = 0.8414709848078964
x = 1.0999999999999999 y = 0.8912073600614353
x = 1.2 y = 0.9320390859672263
x = 1.3 y = 0.963558185417193
x = 1.4000000000000001 y = 0.9854497299884603
x = 1.5000000000000002 y = 0.9974949866040544
x = 1.6000000000000003 y = 0.9995736030415051
x = 1.7000000000000004 y = 0.9916648104524686
x = 1.8000000000000005 y = 0.973847630878195
x = 1.9000000000000006 y = 0.9463000876874142
x = 2.0000000000000004 y = 0.9092974268256815
x = 2.1000000000000005 y = 0.8632093666488735
x = 2.2000000000000006 y = 0.8084964038195899
x = 2.3000000000000007 y = 0.7457052121767197
x = 2.400000000000001 y = 0.6754631805511503
x = 2.500000000000001 y = 0.5984721441039558
x = 2.600000000000001 y = 0.5155013718214634
x = 2.700000000000001 y = 0.42737988023382895
x = 2.800000000000001 y = 0.33498815015590383
x = 2.9000000000000012 y = 0.23924932921398112
x = 3.0000000000000013 y = 0.1411200080598659
x = 3.1000000000000014 y = 0.04158066243328916

Cos от 0 до п с шагом 0.1:
x = 0.0 y = 1.0
x = 0.1 y = 0.9950041652780258
x = 0.2 y = 0.9800665778412416
x = 0.30000000000000004 y = 0.955336489125606
x = 0.4 y = 0.9210609940028851
x = 0.5 y = 0.8775825618903728
x = 0.6 y = 0.8253356149096783
x = 0.7 y = 0.7648421872844885
x = 0.7999999999999999 y = 0.6967067093471655
x = 0.8999999999999999 y = 0.6216099682706645
x = 0.9999999999999999 y = 0.5403023058681398
x = 1.0999999999999999 y = 0.4535961214255775
x = 1.2 y = 0.3623577544766736
x = 1.3 y = 0.26749882862458735
x = 1.4000000000000001 y = 0.16996714290024081
```

```
x = 1.5000000000000002 y = 0.07073720166770268
x = 1.6000000000000003 y = -0.029199522301289037
x = 1.7000000000000004 y = -0.12884449429552508
x = 1.8000000000000005 y = -0.22720209469308753
x = 1.9000000000000006 y = -0.32328956686350396
x = 2.0000000000000004 y = -0.4161468365471428
x = 2.1000000000000005 y = -0.5048461045998579
x = 2.2000000000000006 y = -0.5885011172553463
x = 2.3000000000000007 y = -0.6662760212798248
x = 2.400000000000001 y = -0.737393715541246
x = 2.500000000000001 y = -0.8011436155469343
x = 2.600000000000001 y = -0.8568887533689478
x = 2.700000000000001 y = -0.9040721420170617
x = 2.800000000000001 y = -0.9422223406686585
x = 2.9000000000000012 y = -0.9709581651495908
x = 3.0000000000000013 y = -0.9899924966004456
x = 3.1000000000000014 y = -0.9991351502732795

Табулированный Sin (10 точек) от 0 до π с шагом 0.1:
x = 0.0 y = 0.0
x = 0.1 y = 0.09798155360510165
x = 0.2 y = 0.1959631072102033
x = 0.30000000000000004 y = 0.29394466081530496
x = 0.4 y = 0.38590680571121505
x = 0.5 y = 0.47207033789781927
x = 0.6 y = 0.5582338700844235
x = 0.7 y = 0.6439824415274444
x = 0.7999999999999999 y = 0.707935358675545
x = 0.8999999999999999 y = 0.7718882758236456
x = 0.9999999999999999 y = 0.8358411929717461
x = 1.0999999999999999 y = 0.8839933571281424
x = 1.2 y = 0.9180219935851436
x = 1.3 y = 0.9520506300421447
x = 1.4000000000000001 y = 0.984807753012208
x = 1.5000000000000002 y = 0.984807753012208
x = 1.6000000000000003 y = 0.984807753012208
x = 1.7000000000000004 y = 0.984807753012208
x = 1.8000000000000005 y = 0.966204042925035
x = 1.9000000000000006 y = 0.932175406468034
x = 2.0000000000000004 y = 0.8981467700110329
x = 2.1000000000000005 y = 0.8624409082617227
x = 2.2000000000000006 y = 0.7984879911136221
x = 2.3000000000000007 y = 0.7345350739655215
x = 2.400000000000001 y = 0.670582156817421
x = 2.500000000000001 y = 0.594071569547527
x = 2.600000000000001 y = 0.5079080373609227
x = 2.700000000000001 y = 0.42174450517431844
x = 2.800000000000001 y = 0.3346977889881713
x = 2.9000000000000012 y = 0.23671623538306957
x = 3.0000000000000013 y = 0.1387346817779678
x = 3.1000000000000014 y = 0.04075312817286608

Табулированный Cos (10 точек) от 0 до π с шагом 0.1:
```

x = 0.0 y = 1.0
x = 0.1 y = 0.9827232084876878
x = 0.2 y = 0.9654464169753757
x = 0.30000000000000004 y = 0.9481696254630635
x = 0.4 y = 0.914354644443779
x = 0.5 y = 0.864608105941514
x = 0.6 y = 0.8148615674392491
x = 0.7 y = 0.7646204979800333
x = 0.7999999999999999 y = 0.6884043792119047
x = 0.8999999999999999 y = 0.6121882604437761
x = 0.9999999999999999 y = 0.5359721416756473
x = 1.0999999999999999 y = 0.45063345391435955
x = 1.2 y = 0.35714054363391856
x = 1.3 y = 0.26364763335347763
x = 1.4000000000000001 y = 0.16993052093895483
x = 1.5000000000000002 y = 0.07043744393442489
x = 1.6000000000000003 y = -0.029055633070105086
x = 1.7000000000000004 y = -0.12854871007463503
x = 1.8000000000000005 y = -0.22476145104951817
x = 1.9000000000000006 y = -0.3182543613299591
x = 2.0000000000000004 y = -0.41174727161039987
x = 2.1000000000000005 y = -0.5042718354168345
x = 2.2000000000000006 y = -0.5804879541849632
x = 2.3000000000000007 y = -0.656704072953092
x = 2.400000000000001 y = -0.7329201917212208
x = 2.500000000000001 y = -0.7941706620070894
x = 2.600000000000001 y = -0.8439172005093544
x = 2.700000000000001 y = -0.8936637390116193
x = 2.800000000000001 y = -0.9409837494179168
x = 2.9000000000000012 y = -0.958260540930229
x = 3.0000000000000013 y = -0.9755373324425413
x = 3.1000000000000014 y = -0.9928141239548535

Сумма квадратов табулированных Sin и Cos (10 точек) от 0 до п с шагом 0.1:
x = 0.0 y = 1.0
x = 0.1 y = 0.9753452893472049
x = 0.2 y = 0.9704883234380686
x = 0.30000000000000004 y = 0.9854291022725908
x = 0.4 y = 0.9849684785101429
x = 0.5 y = 0.9703975807827336
x = 0.6 y = 0.975624427798983
x = 0.7 y = 0.9993578909268825
x = 0.7999999999999999 y = 0.9750730613812004
x = 0.8999999999999999 y = 0.9705859765791769
x = 0.9999999999999999 y = 0.9858966365208117
x = 1.0999999999999999 y = 0.9845147652334687
x = 1.2 y = 0.9703137486131723
x = 1.3 y = 0.9759104767365345
x = 1.4000000000000001 y = 0.9987226923395387
x = 1.5000000000000002 y = 0.9748077439009694
x = 1.6000000000000003 y = 0.9706905402060587
x = 1.7000000000000004 y = 0.9863710812548067

x = 1.8000000000000005 y = 0.9840679624425679
x = 1.9000000000000006 y = 0.9702368269293845
x = 2.0000000000000004 y = 0.9762034361598597
x = 2.1000000000000005 y = 0.9980944042379682
x = 2.2000000000000006 y = 0.9745493369065118
x = 2.3000000000000007 y = 0.9708020143187142
x = 2.400000000000001 y = 0.9868524364745752
x = 2.500000000000001 y = 0.9836280701374409
x = 2.600000000000001 y = 0.9701668157313703
x = 2.700000000000001 y = 0.9765033060689583
x = 2.800000000000001 y = 0.9974730266221714
x = 2.9000000000000012 y = 0.974297840397828
x = 3.0000000000000013 y = 0.9709203989171432
x = 3.1000000000000014 y = 0.9873407021801173
Тест с 5 точками
x = 0.0 y = 1.0
x = 0.1 y = 0.9349117663199064
x = 0.2 y = 0.8888163567108488
x = 0.30000000000000004 y = 0.8617137711728265
x = 0.4 y = 0.8536040097058402
x = 0.5 y = 0.8644870723098893
x = 0.6 y = 0.8943629589849742
x = 0.7 y = 0.9432316697310947
x = 0.7999999999999999 y = 0.9893117483522796
x = 0.8999999999999999 y = 0.926996815809249
x = 0.9999999999999999 y = 0.8836747073372538
x = 1.0999999999999999 y = 0.8593454229362943
x = 1.2 y = 0.8540089626063706
x = 1.3 y = 0.8676653263474826
x = 1.4000000000000001 y = 0.9003145141596303
x = 1.5000000000000002 y = 0.9519565260428137
x = 1.6000000000000003 y = 0.9790284496050897
x = 1.7000000000000004 y = 0.9194868181991217
x = 1.8000000000000005 y = 0.8789380108641893
x = 1.9000000000000006 y = 0.8573820276002928
x = 2.0000000000000004 y = 0.854818868407432
x = 2.1000000000000005 y = 0.8712485332856068
x = 2.2000000000000006 y = 0.9066710222348173
x = 2.3000000000000007 y = 0.9610863352550636
x = 2.400000000000001 y = 0.9691501037584304
x = 2.500000000000001 y = 0.9123817734895252
x = 2.600000000000001 y = 0.8746062672916557
x = 2.700000000000001 y = 0.855823585164822
x = 2.800000000000001 y = 0.8560337271090238
x = 2.9000000000000012 y = 0.8752366931242617
x = 3.0000000000000013 y = 0.913432483210535
x = 3.1000000000000014 y = 0.9706210973678441
Тест с 20 точками
x = 0.0 y = 1.0
x = 0.1 y = 0.9934801762782068
x = 0.2 y = 0.9954813685939703
x = 0.30000000000000004 y = 0.9958763728929726
x = 0.4 y = 0.9933589338027065

```
x = 0.5 y = 0.9993625107499969
x = 0.6 y = 0.9936326956262082
x = 0.7 y = 0.995117641167469
x = 0.7999999999999999 y = 0.9963026540644755
x = 0.8999999999999999 y = 0.9932689681997069
x = 0.9999999999999999 y = 0.9987562983724949
x = 1.0999999999999999 y = 0.9938164918467103
x = 1.2 y = 0.9947851906134687
x = 1.3 y = 0.9967602121084793
x = 1.4000000000000001 y = 0.9932102794692081
x = 1.5000000000000002 y = 0.9981813628674935
x = 1.6000000000000003 y = 0.9940315649397132
x = 1.7000000000000004 y = 0.994484016931969
x = 1.8000000000000005 y = 0.997249047024984
x = 1.9000000000000006 y = 0.9931828676112102
x = 2.0000000000000004 y = 0.9976377042349929
x = 2.1000000000000005 y = 0.994277914905217
x = 2.2000000000000006 y = 0.9942141201229702
x = 2.3000000000000007 y = 0.9977691588139896
x = 2.400000000000001 y = 0.993186732625713
x = 2.500000000000001 y = 0.9971253224749932
x = 2.600000000000001 y = 0.9945555417432219
x = 2.700000000000001 y = 0.9939755001864723
x = 2.800000000000001 y = 0.9983205474754961
x = 2.9000000000000012 y = 0.9932218745127168
x = 3.0000000000000013 y = 0.9966442175874943
x = 3.1000000000000014 y = 0.9948644454537273
Сравнение табулированной и считанной из файла экспоненты:
x = 0.0 первая = 1.0 вторая = 1.0
x = 1.0 первая = 2.718281828459045 вторая = 2.718281828459045
x = 2.0 первая = 7.38905609893065 вторая = 7.38905609893065
x = 3.0 первая = 20.085536923187668 вторая = 20.085536923187668
x = 4.0 первая = 54.598150033144236 вторая = 54.59815003314424
x = 5.0 первая = 148.4131591025766 вторая = 148.4131591025766
x = 6.0 первая = 403.4287934927351 вторая = 403.4287934927351
x = 7.0 первая = 1096.6331584284585 вторая = 1096.6331584284585
x = 8.0 первая = 2980.9579870417283 вторая = 2980.9579870417283
x = 9.0 первая = 8103.083927575384 вторая = 8103.083927575384
x = 10.0 первая = 22026.465794806718 вторая = 22026.46579480672
Сравнение табулированного и считанного из файла логарифма:
x = 1.0 первая = 0.0 вторая = 0.0
x = 2.0 первая = 0.6849389451733685 вторая = 0.6849389451733685
x = 3.0 первая = 1.0915557288409405 вторая = 1.0915557288409405
x = 4.0 первая = 1.3809073142651356 вторая = 1.3809073142651356
x = 5.0 первая = 1.605474876269883 вторая = 1.605474876269883
x = 6.0 первая = 1.7889424800868703 вторая = 1.7889424800868703
x = 7.0 первая = 1.9440155622247723 вторая = 1.9440155622247723
x = 8.0 первая = 2.078298641800016 вторая = 2.078298641800016
x = 9.0 первая = 2.196703273605849 вторая = 2.196703273605849
x = 10.0 первая = 2.302585092994046 вторая = 2.302585092994046
```

Задание 9

Сделала так, чтобы объекты всех классов, реализующих интерфейс TabulatedFunction, были сериализуемыми.
Для этого рассмотрела два случая:

1. с использованием интерфейса java.io.Serializable

```
import java.io.Serializable;

public class FunctionPoint implements Serializable {
  private static final long serialVersionUID = 1L;

import java.io.Serializable;

public class ArrayTabulatedFunction implements TabulatedFunction,
Serializable {
    private static final long serialVersionUID = 1L;
```

2. с использованием интерфейса java.io.Externalizable

```
package functions;
import java.io.*;

public class ArrayTabulatedFunction implements TabulatedFunction,
Externalizable {
    private FunctionPoint[] points; //Массив для хранения точек
    private int pointsCount; //Счетчик точек
    public ArrayTabulatedFunction() {} // Конструктор по умолчанию для
Externalizable
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeInt(pointsCount);
        for (int i = 0; i < pointsCount; i++) {
            out.writeDouble(points[i].getX());
            out.writeDouble(points[i].getY());
        }
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException {
        pointsCount = in.readInt();
        points = new FunctionPoint[pointsCount + 10];
        for (int i = 0; i < pointsCount; i++) {
            double x = in.readDouble();
            double y = in.readDouble();
            points[i] = new FunctionPoint(x, y);
        }
    }
```

Проверила работу написанных классов. С помощью метода TabulatedFunctions.tabulate() и метода класса Functions создала табулированный аналог логарифма по натуральному основанию, взятого от экспоненты на отрезке от 0 до 10 с 11 точками. Сериализовала полученный объект в файл (имя файла должно отличаться от предыдущих случаев). Далее десериализовала табулированную функцию из этого файла. Вывела значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
Изучила содержимое файлов, получаемых при реализации механизма сериализации с использованием интерфейса java.io.Serializable и при

реализации механизма сериализации с использованием интерфейса java.io.Externalizable.

```
// Сериализация с использованием Serializable
System.out.println("\nАвтоматическая сериализация
(serializable)");
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("serializable_function.ser"))) {
    oos.writeObject(tabulatedComposition);
}

TabulatedFunction readSerializable;
try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("serializable_function.ser"))) {
    readSerializable = (TabulatedFunction) ois.readObject();
}

System.out.println("Сравнение Serializable (ln(exp(x)) = x):");
compareFunctions(tabulatedComposition, readSerializable, 0, 10,
1);
```

Вывод:

Автоматическая сериализация (serializable)
Сравнение Serializable (ln(exp(x)) = x):
x = 2.0 первая = 2.0 вторая = 2.0
x = 3.0 первая = 3.0000000000000004 вторая = 3.0000000000000004
x = 4.0 первая = 4.0 вторая = 4.0
x = 5.0 первая = 4.999999999999999 вторая = 4.999999999999999
x = 6.0 первая = 6.0 вторая = 6.0
x = 7.0 первая = 6.999999999999999 вторая = 6.999999999999999
x = 8.0 первая = 7.999999999999998 вторая = 7.999999999999998
x = 9.0 первая = 9.000000000000002 вторая = 9.000000000000002
x = 10.0 первая = 10.000000000000002 вторая = 10.000000000000002

```
// Сериализация с использованием externalizable
System.out.println("\nРучная сериализация (externalizable)");
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("externalizable_function.ser"))) {
    oos.writeObject(tabulatedComposition);
}

TabulatedFunction readExternalizable;
try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("externalizable_function.ser"))) {
    readExternalizable = (TabulatedFunction) ois.readObject();
}

System.out.println("Сравнение Externalizable (ln(exp(x)) =
x):");
compareFunctions(tabulatedComposition, readExternalizable, 0,
10, 1);
```

Ручная сериализация (externalizable)

Сравнение Externalizable (ln(exp(x)) = x):

x = 2.0 первая = 2.0 вторая = 2.0

x = 3.0 первая = 3.0000000000000004 вторая = 3.0000000000000004

x = 4.0 первая = 4.0 вторая = 4.0

x = 5.0 первая = 4.999999999999999 вторая = 4.999999999999999

x = 6.0 первая = 6.0 вторая = 6.0

x = 7.0 первая = 6.999999999999999 вторая = 6.999999999999999

x = 8.0 первая = 7.999999999999998 вторая = 7.999999999999998

x = 9.0 первая = 9.000000000000002 вторая = 9.000000000000002

x = 10.0 первая = 10.000000000000002 вторая = 10.000000000000002

Я использовала Externalizable для класса ArrayTabulatedFunction, а Serializable – для FunctionPoint. Автоматическая сериализация используется для меньших классов, где производительность не критична, также это проще, так как Java сама все сериализует (получился файл 450 байт). Этот способ медленнее, размер файла больше. Ручную сериализацию я использовала, чтобы отредактировать что и как сохранять (контролировать процесс), оптимизировать процесс и получить меньший размер файла быстрее, но в этом способе выше вероятность получения ошибки и сложнее реализация (получился файл 236 байт). Мне кажется, способ с ручной сериализацией больше подходит для класса ArrayTabulatedFunction, чем автоматический.