МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 4

«Расширение возможностей пакета для работы с функциями путем добавления классов для аналитических функций и методов ввода-вывода табличных данных»

по курсу
Объектно-ориентированное программирование

Выполнила: Элибекян Анжелина,
студент группы 6203-010302D

Оглавление

Содержание

Задание №1

В данном задании в классы ArrayTabulatedFunction и LinkedListTabulatedFunction я добавила конструкторы, получающие сразу все точки функции в виде массива объектов типа FunctionPoint

```
// конструктор для получения точек FunctionPoint
public ArrayTabulatedFunction(FunctionPoint [] points) throws IllegalArgumentException {  4 usages
    // проверка, что точек достаточно
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество тчоек должно быть не менее 2");
    }
    // упорядоченность точек по x
    for (int i = 0; i < points.length - 1; i++){
        if(points[i].getX() >= points[i + 1].getX()){
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию x");
        }
    }

    //массив для хранения точек
    this.count = points.length;
    this.points = new FunctionPoint[count + 5];

    //копии точек (инкапсуляция)
    for (int i = 0; i < points.length; i++){
        if (points[i] == null){
            throw new IllegalArgumentException("Точка с индексом " + i + " не может быть null");
        }
        this.points[i] = new FunctionPoint(points[i]);
    }

}
```

```java
// конструктор для получения точек FunctionPoint
public LinkedListTabulatedFunction(FunctionPoint [] points) throws IllegalArgumentException {  2 usage
    // проверка, что точек достаточно
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество тчоек должно быть не менее 2");
    }
    // упорядоченность точек по x
    for (int i = 0; i < points.length - 1; i++){
        if(points[i].getX() >= points[i + 1].getX()){
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию x");
        }
    }

    //инициализация пустого списка
    initializeList();

    //добавление точек с созданием копий
    for (FunctionPoint point : points){
        try {
            addPoint(new FunctionPoint(point));
        } catch (InappropriateFunctionPointException e){

        }
    }
}
```

Задание №2

Далее в пакете functions я создаю интерфейс Function, который содержит методы : возвращение значения левой границы области определения ф-ии, аналогично для правой границы, и возвращение значения ф-ии в заданной точке. Из интерфейса TabulatedFunction соответствующие методы я удаляю.

```java
package functions;

public interface Function {
    //получаем левую границу области определения функции
    public double getLeftDomainBorder();  11 implementations
    //получаем правую границу области определения функции
    public double getRightDomainBorder();  11 implementations
    //вычисляем значение функции в заданной точке
    public double getFunctionValue(double x);  14 implementat
}
```

Задание №3

Создаю пакет functions.basic, в котором будут описаны классы ряда функий, заданных аналитически : Exp; Log; я создаю класс TrigonometricFunction, который в свою очередь также реализует интерфейс и описывает методы получения границ области определения, от данного класса создаю наследующие от него публичные классы Sin, Cos, Tan.

```java
public class Log implements Function {  2 usages
    //основание логарифма
    private double base;  2 usages

    //конструктор логарифма
    public Log(double base){  4 usages
        //проверка на валидность
        if (base <= 0 || base == 1){
            throw new IllegalArgumentException("Основание логарифма должно быть > 0 и не равным 1");
        }
        this.base = base;
    }

    //вычисляем значение логарифма в точке x
    public double getFunctionValue(double x) {
        if (x <= 0) {
            return Double.NaN;
        }
        //используем формулу замены основания
        return Math.log(x) / Math.log(base);
    }

    //возвращаем левую границу области определения
    public double getLeftDomainBorder() {
        //открытый интервал от 0
        return 0;
    }

    //возвращаем правую границу области определения
    public double getRightDomainBorder() {
        //интервал до +бесконечности
        return Double.POSITIVE_INFINITY;
    }
}
```

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {  2 usages
    public double getLeftDomainBorder() {
        //возвращаем левую границу области определения
        return Double.NEGATIVE_INFINITY;
    }
    public double getRightDomainBorder() {
        // возвращаем правую границу области определения
        return Double.POSITIVE_INFINITY;
    }
    public double getFunctionValue(double x) {
        // вычисляем значение экспоненты в точке x
        return Math.exp(x);
    }
}
```

```java
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {  3 usages  3 inh

    //возвращаем левую границу области определения тригонометрических функций
    public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }

    //возвращаем правую границу области определения тригонометрических функций
    public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }

    //абстрактный метод для вычисления значения тригонометрической функции
    public abstract double getFunctionValue(double x);  3 implementations
}
```

```java
package functions.basic;

public class Sin extends TrigonometricFunction {  3 us
    public double getFunctionValue(double x) {

        //возвращаем значение синуса в точке x
        return Math.sin(x);
    }
}
```

```java
package functions.basic;

public class Cos extends TrigonometricFunction {
    public double getFunctionValue(double x) {

        //возвращаем значение косинуса в точке x
        return Math.cos(x);
    }
}
```

```java
package functions.basic;

public class Tan extends TrigonometricFunction{
    public double getFunctionValue(double x) {

        //возвращаем значение тангенса в точке x
        return Math.tan(x);
    }
}
```

Задание №4

Создаю пакет functions.meta, в котором будут описаны классы функций, позволяющие комбинировать ф-ии. Объекты класса Sum представляют собой ф-ии, являющиеся суммой двух других ф-ий, а область определения ф-ии является пересечением областей. Класс Mult создан для произведения двух функций, класс Power для возведения ф-ии в степень, Scale создан для масштабирования функций вдоль осей координат, Shift для сдвига ф-ий вдоль осей координат, класс Composition описывает композицию двух исходных функций(область определения ф-ии можем считать совпадающей с областью определения исходной ф-ии).

Задание №5

В пакете functions создаю класс Functions, который содержит статические методы для работы с функциями. При написании методов использую созданные ранее классы из пакета functions.meta.

```
package functions;

import functions.meta.*;

public class Functions {  7 usages
    //приватный конструктор - запрещает создание экземпляров класса
    private Functions(){  no usages
    }
    //функция сдвига
    public static Function shift(Function f, double shiftX, double shiftY) { return new Shift(f, shiftX, shiftY); }
    //функция масштабирования
    public static Function scale(Function f, double scaleX, double scaleY) { return new Scale(f, scaleX, scaleY); }
    //функция возведения в степень
    public static Function power(Function f, double power) { return new Power(f, power); }
    //функция суммы
    public static Function sum(Function f1, Function f2) { return new Sum(f1, f2); }
    //функция произведения
    public static Function mult(Function f1, Function f2) { return new Mult(f1, f2); }
    //композиция функций
    public static Function composition(Function f1, Function f2) { return new Composition(f1, f2); }
}
```

Задание №6

Работаю в том же самом пакете, создаю класс TabulatedFunctions, содержащий статические методы для работы с табулированными функциями. Я описываю метод - public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount), который получает функцию и возвращает ее табулированный аналог на заданном отрезке. Но если указанные границы выходят за область определения, выбрасываем исключение IllegalArgumentException.

```java
import java.io.*;

public class TabulatedFunctions {  10 usages
    //приватный конструктор - запрещает создание экземпляров класса
    private TabulatedFunctions() {  no usages
    }

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount)
        //проверка на количество точек
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }
        //проверка, находятся ли границы в области определения
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табуляции выходят за область определения");
        }
        //проверка корректности границ
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }
        //создаем табулированную функцию
        ArrayTabulatedFunction tabulatedFunc = new ArrayTabulatedFunction(leftX, rightX, pointsCount);
        //заполняем значения y, вычисляя их из исходной функции
        for (int i = 0; i < pointsCount; i++) {
            double x = tabulatedFunc.getPointX(i);
            double y = function.getFunctionValue(x);
            tabulatedFunc.setPointY(i, y);
        }
        return tabulatedFunc;
    }
```

Задание №7

В данном задании я реализовала все четыре требуемых метода для работы с табулированными функциями в классе TabulatedFunctions. Для методов outputTabulatedFunction и inputTabulatedFunction я использовала потоки-обертки DataOutputStream и DataInputStream соответственно, что позволило удобно записывать и читать примитивные типы данных в бинарном формате. Для методов writeTabulatedFunction и readTabulatedFunction я применила PrintWriter для записи текстовых данных и StreamTokenizer для чтения, что обеспечило корректную работу с символьными потоками. При реализации я сознательно не стала закрывать потоки внутри методов, поскольку управление циклом потоков должно оставаться за вызывающим кодом, который может продолжать использовать эти потоки после выполнения методов. Я не обрабатываю исключения IOException внутри методов, а просто объявляю их через “throws IOException”, чтобы передать вызывающему коду. Это сделано потому, что только вызывающая программа знает, как правильно обработать ошибку ввода-вывода в своей конкретной ситуации. Например, одна программа может показать сообщение об ошибке пользователю, другая - записать в лог, а третья - попробовать альтернативный способ сохранения данных. Если бы я обрабатывала исключения внутри методов, это ограничило бы гибкость использования кода и усложнило бы его повторное использование в разных контекстах.

```java
//выводим табулированную функцию в байтовый поток
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
    //создаем поток-обертку
    DataOutputStream dataOut = new DataOutputStream(out);
    //записываем количество точек функции
    dataOut.writeInt(function.getPointsCount());
    //последовательно записываем координаты всех точек функции
    for (int i = 0; i < function.getPointsCount(); i++) {
        dataOut.writeDouble(function.getPointX(i));
        dataOut.writeDouble(function.getPointY(i));
    }
    //сбрасываем буфер, обеспечивая запись всех данных в поток
    //не закрываем поток, так как он может использоваться вызывающим кодом дальше
    dataOut.flush();
}
```

```java
//выводим табулированную функцию из байтового потока
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
    //создаем поток-обертку
    DataInputStream dataIn = new DataInputStream(in);
    //читаем количество точек функции
    int pointsCount = dataIn.readInt();
    //создаем массивы для хранения координат точек
    double[] xValues = new double[pointsCount];
    double[] yValues = new double[pointsCount];
    //последовательно читаем координаты всех точек из потока
    for (int i = 0; i < pointsCount; i++) {
        xValues[i] = dataIn.readDouble();
        yValues[i] = dataIn.readDouble();
    }
    //создаем массив объектов FunctionPoint из прочитанных координат
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        points[i] = new FunctionPoint(xValues[i], yValues[i]);
    }
    //создаем и возвращаем табулированную функцию
    return new ArrayTabulatedFunction(points);
}
```

```java
//записываем табулированную функцию в символьный поток
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException{
    PrintWriter writer = new PrintWriter(out);

    //записываем количество точек
    writer.print(function.getPointsCount());
    writer.print(" ");
    //записываем координаты всех точек через пробел
    for(int i = 0; i < function.getPointsCount(); i++){
        writer.print(function.getPointX(i));
        writer.print(" ");
        writer.print(function.getPointY(i));
        writer.print(" ");
    }
    //не закрываем поток, а сбрасываем буфер, так как он может использоваться дальше
    writer.flush();
}
```

```java
//читаем табулированную функцию из символьного потока
public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException{
    //StreamTokenizer для чтения чисел
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    //читаем количество точек
    tokenizer.nextToken(); //переходим к следующему токену (числу)
    int pointsCount = (int) tokenizer.nval; //читаем числовое значение
    //создаем массивы для координат
    double[] xValues = new double[pointsCount];
    double[] yValues = new double[pointsCount];
    //читаем координаты всех точек
    for(int i = 0; i < pointsCount; i++){
        //переходим к следующему токену в потоке, который должен быть числом
        //не делаем проверки, так как считаем, что данные в потоке записаны корректно
        tokenizer.nextToken();
        //сохраняем числовое значение токена как x-координату текущей точки
        xValues[i] = tokenizer.nval;
        //аналогично с y-координатой
        tokenizer.nextToken();
        yValues[i] = tokenizer.nval;
    }
    //создаем функцию из массива точек
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for(int i = 0; i < pointsCount; i++){
        points[i] = new FunctionPoint(xValues[i], yValues[i]);
    }
    return new ArrayTabulatedFunction(points);
}
```

Задание №8

Я протестировала работу классов для функций и табулирования, выполнив все требуемые задания. Сначала я создала объекты Sin и Cos и вывела их значения на отрезке от 0 до π с шагом 0.1, затем с помощью TabulatedFunctions.tabulate() создала табулированные аналоги этих функций с 10 точками и сравнила их с оригиналами. Далее я использовала методы класса Functions для создания функции-суммы квадратов табулированных синуса и косинуса, исследовала как количество точек табуляции влияет на точность результата, проверяя тождество $\sin^2 x + \cos^2 x = 1$ при разном количестве точек. Для работы с файлами я создала табулированную экспоненту, записала ее в текстовый файл методом writeTabulatedFunction и считала обратно методом readTabulatedFunction, затем проделала то же самое с логарифмом используя бинарные файлы и методы outputTabulatedFunction/inputTabulatedFunction. Изучив содержимое полученных файлом могу сделать вывод, что текстовые файлы понятны людям, но большие и медленные. Бинарные файлы - быстрые и компактные, но их может прочитать только программа. Для настроек лучше текст, а для данных - бинарный формат.

```
1. ТЕСТИРВОАНИЕ SIN И COS:
==========================
Значения sin(x) и cos(x) на [0, π] с шагом 0.1:
x=0,0: sin=0,000, cos=1,000
x=0,1: sin=0,100, cos=0,995
x=0,2: sin=0,199, cos=0,980
x=0,3: sin=0,296, cos=0,955
x=0,4: sin=0,389, cos=0,921
x=0,5: sin=0,479, cos=0,878
x=0,6: sin=0,565, cos=0,825
x=0,7: sin=0,644, cos=0,765
x=0,8: sin=0,717, cos=0,697
x=0,9: sin=0,783, cos=0,622
x=1,0: sin=0,841, cos=0,540
x=1,1: sin=0,891, cos=0,454
x=1,2: sin=0,932, cos=0,362
x=1,3: sin=0,964, cos=0,267
x=1,4: sin=0,985, cos=0,170
x=1,5: sin=0,997, cos=0,071
x=1,6: sin=1,000, cos=-0,029
x=1,7: sin=0,992, cos=-0,129
x=1,8: sin=0,974, cos=-0,227
x=1,9: sin=0,946, cos=-0,323
x=2,0: sin=0,909, cos=-0,416
x=2,1: sin=0,863, cos=-0,505
x=2,2: sin=0,808, cos=-0,589
x=2,3: sin=0,746, cos=-0,666
x=2,4: sin=0,675, cos=-0,737
x=2,5: sin=0,598, cos=-0,801
x=2,6: sin=0,516, cos=-0,857
x=2,7: sin=0,427, cos=-0,904
x=2,8: sin=0,335, cos=-0,942
```

```
x=2,9: sin=0,239, cos=-0,971
x=3,0: sin=0,141, cos=-0,990
x=3,1: sin=0,042, cos=-0,999

2. ТАБУЛИРОВАНИЕ SIN И COS:
============================
Сравнение оригинальных и табулированных функций:
x=0,0: sin(ориг)=0,000, sin(табл)=0,000, cos(ориг)=1,000, cos(табл)=1,000
x=0,1: sin(ориг)=0,100, sin(табл)=0,098, cos(ориг)=0,995, cos(табл)=0,983
x=0,2: sin(ориг)=0,199, sin(табл)=0,196, cos(ориг)=0,980, cos(табл)=0,965
x=0,3: sin(ориг)=0,296, sin(табл)=0,294, cos(ориг)=0,955, cos(табл)=0,948
x=0,4: sin(ориг)=0,389, sin(табл)=0,386, cos(ориг)=0,921, cos(табл)=0,914
x=0,5: sin(ориг)=0,479, sin(табл)=0,472, cos(ориг)=0,878, cos(табл)=0,865
x=0,6: sin(ориг)=0,565, sin(табл)=0,558, cos(ориг)=0,825, cos(табл)=0,815
x=0,7: sin(ориг)=0,644, sin(табл)=0,644, cos(ориг)=0,765, cos(табл)=0,765
x=0,8: sin(ориг)=0,717, sin(табл)=0,708, cos(ориг)=0,697, cos(табл)=0,688
x=0,9: sin(ориг)=0,783, sin(табл)=0,772, cos(ориг)=0,622, cos(табл)=0,612
x=1,0: sin(ориг)=0,841, sin(табл)=0,836, cos(ориг)=0,540, cos(табл)=0,536
x=1,1: sin(ориг)=0,891, sin(табл)=0,884, cos(ориг)=0,454, cos(табл)=0,451
x=1,2: sin(ориг)=0,932, sin(табл)=0,918, cos(ориг)=0,362, cos(табл)=0,357
x=1,3: sin(ориг)=0,964, sin(табл)=0,952, cos(ориг)=0,267, cos(табл)=0,264
x=1,4: sin(ориг)=0,985, sin(табл)=0,985, cos(ориг)=0,170, cos(табл)=0,170
x=1,5: sin(ориг)=0,997, sin(табл)=0,985, cos(ориг)=0,071, cos(табл)=0,070
x=1,6: sin(ориг)=1,000, sin(табл)=0,985, cos(ориг)=-0,029, cos(табл)=-0,029
x=1,7: sin(ориг)=0,992, sin(табл)=0,985, cos(ориг)=-0,129, cos(табл)=-0,129
x=1,8: sin(ориг)=0,974, sin(табл)=0,966, cos(ориг)=-0,227, cos(табл)=-0,225
x=1,9: sin(ориг)=0,946, sin(табл)=0,932, cos(ориг)=-0,323, cos(табл)=-0,318
x=2,0: sin(ориг)=0,909, sin(табл)=0,898, cos(ориг)=-0,416, cos(табл)=-0,412
x=2,1: sin(ориг)=0,863, sin(табл)=0,862, cos(ориг)=-0,505, cos(табл)=-0,504
x=2,2: sin(ориг)=0,808, sin(табл)=0,798, cos(ориг)=-0,589, cos(табл)=-0,580
x=2,3: sin(ориг)=0,746, sin(табл)=0,735, cos(ориг)=-0,666, cos(табл)=-0,657
x=2,4: sin(ориг)=0,675, sin(табл)=0,671, cos(ориг)=-0,737, cos(табл)=-0,733
x=2,5: sin(ориг)=0,598, sin(табл)=0,594, cos(ориг)=-0,801, cos(табл)=-0,794
```

```
x=2,6: sin(ориг)=0,516, sin(табл)=0,508, cos(ориг)=-0,857, cos(табл)=-0,844
x=2,7: sin(ориг)=0,427, sin(табл)=0,422, cos(ориг)=-0,904, cos(табл)=-0,894
x=2,8: sin(ориг)=0,335, sin(табл)=0,335, cos(ориг)=-0,942, cos(табл)=-0,941
x=2,9: sin(ориг)=0,239, sin(табл)=0,237, cos(ориг)=-0,971, cos(табл)=-0,958
x=3,0: sin(ориг)=0,141, sin(табл)=0,139, cos(ориг)=-0,990, cos(табл)=-0,976
x=3,1: sin(ориг)=0,042, sin(табл)=0,041, cos(ориг)=-0,999, cos(табл)=-0,993

3. СУММА КВАДРАТОВ ТАБУЛИРОВАННЫХ SIN И COS:
==============================================
Значения sin²(x) + cos²(x) на [0, π] с шагом 0.1:
x=0,0: sin²+cos²=1,000
x=0,1: sin²+cos²=0,975
x=0,2: sin²+cos²=0,970
x=0,3: sin²+cos²=0,985
x=0,4: sin²+cos²=0,985
x=0,5: sin²+cos²=0,970
x=0,6: sin²+cos²=0,976
x=0,7: sin²+cos²=0,999
x=0,8: sin²+cos²=0,975
x=0,9: sin²+cos²=0,971
x=1,0: sin²+cos²=0,986
x=1,1: sin²+cos²=0,985
x=1,2: sin²+cos²=0,970
x=1,3: sin²+cos²=0,976
x=1,4: sin²+cos²=0,999
x=1,5: sin²+cos²=0,975
x=1,6: sin²+cos²=0,971
x=1,7: sin²+cos²=0,986
x=1,8: sin²+cos²=0,984
x=1,9: sin²+cos²=0,970
x=2,0: sin²+cos²=0,976
x=2,1: sin²+cos²=0,998
x=2,2: sin²+cos²=0,975
```

```
x=2,2: sin²+cos²=0,975
x=2,3: sin²+cos²=0,971
x=2,4: sin²+cos²=0,987
x=2,5: sin²+cos²=0,984
x=2,6: sin²+cos²=0,970
x=2,7: sin²+cos²=0,977
x=2,8: sin²+cos²=0,997
x=2,9: sin²+cos²=0,974
x=3,0: sin²+cos²=0,971
x=3,1: sin²+cos²=0,987

Исследование влияния количества точек:
Точек: 5, значение в π/2: 1,000000
Точек: 10, значение в π/2: 0,969846
Точек: 20, значение в π/2: 0,993181

4. РАБОТА С ФАЙЛАМИ - ЭКСПОНЕНТА:
================================
Сравнение оригинальной и прочитанной экспоненты:
x=0: оригинал=1,000, прочитано=1,000
x=1: оригинал=2,718, прочитано=2,718
x=2: оригинал=7,389, прочитано=7,389
x=3: оригинал=20,086, прочитано=20,086
x=4: оригинал=54,598, прочитано=54,598
x=5: оригинал=148,413, прочитано=148,413
x=6: оригинал=403,429, прочитано=403,429
x=7: оригинал=1096,633, прочитано=1096,633
x=8: оригинал=2980,958, прочитано=2980,958
x=9: оригинал=8103,084, прочитано=8103,084
x=10: оригинал=22026,466, прочитано=22026,466
```

```
5. РАБОТА С ФАЙЛАМИ - ЛОГАРИФМ:
==============================
Сравнение оригинального и прочитанного логарифма:
x=0,1: оригинал=-2,303, прочитано=-2,303
x=1,1: оригинал=0,093, прочитано=0,093
x=2,1: оригинал=0,740, прочитано=0,740
x=3,1: оригинал=1,130, прочитано=1,130
x=4,1: оригинал=1,410, прочитано=1,410
x=5,1: оригинал=1,628, прочитано=1,628
x=6,1: оригинал=1,808, прочитано=1,808
x=7,1: оригинал=1,960, прочитано=1,960
x=8,1: оригинал=2,091, прочитано=2,091
x=9,1: оригинал=2,208, прочитано=2,208
```

Задание №9

Это задание оказалось наиболее интересным для изучения лично для меня. Я реализовала сериализацию для классов ArrayTabulatedFunction и LinkedListTabulatedFunction с использованием интерфейса Externalizable. В ArrayTabulatedFunction я добавила конструктор без параметров и методы writeExternal/readExternal, которые записывают количество точек и координаты всех точек в бинарном формате. В LinkedListTabulatedFunction я также реализовала Externalizable, добавив методы для записи и чтения узлов списка. Для проверки я создала табулированную функцию композиции логарифма и экспонеты, сериализовала ее в файл "composition_externalizable.dat" и десериализовала обратно. Сравнение показало полное совпадение значений исходной и восстановленной функции. При изучении файлов выяснилось, что Externalizable дает более компактный размер файла по сравнению со стандартной сериализацией, так как позволяет контролировать процесс записи и избежать служебной информации. Основное преимущество Externalizable - полный контроль над процессом сериализации и эффективное использование памяти, а недостаток - необходимость ручной реализации методов. Стандартная сериализация проще в реализации, но может быть менее эффективной.

```java
// класс, реализующий табулированную функцию с использованием массива для хранения точек
public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable {  4 usages
    private FunctionPoint[] points;  // массив для хранения точек функции  37 usages
    private int count;               // текущее количество точек в массиве  29 usages

    //создаем пустой конструктор без параметров (обязательное требование для Externalizable)
    public ArrayTabulatedFunction(){  2 usages
        this.points = new FunctionPoint[0];
        this.count = 0;
    }

    //метод записи объекта в поток
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        //сначала записываем количество точек(для чтения)
        out.writeInt(count);
        //записываем координаты всех точек
        for (int i = 0; i < count; i++) {
            out.writeDouble(points[i].getX());  //x-координата
            out.writeDouble(points[i].getY());  //y-координата
        }
    }

    //метод чтения объекта из потока
    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        //сначала читаем количество точек
        count = in.readInt();
        //создаем массив для хранения точек
        points = new FunctionPoint[count + 5];
        //читаем координаты и создаем точки
        for (int i = 0; i < count; i++) {
            double x = in.readDouble();  //читаем x-координату
            double y = in.readDouble();  //читаем y-координату
            points[i] = new FunctionPoint(x, y);  //создаем новую точку
```

```java
// класс, реализующий табулированную функцию с использованием двусвязного циклического списка
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {  no usage
    // внутренний класс для узла двусвязного списка
    private static class FunctionNode {  22 usages
        private FunctionPoint point;    // данные точки, хранящейся в узле  25 usages
        private FunctionNode prev;      // ссылка на предыдущий узел в списке  15 usages
        private FunctionNode next;      // ссылка на следующий узел в списке  20 usages

        // конструктор с точкой
        // создает узел с заданной точкой и null-ссылками
        public FunctionNode(FunctionPoint point) { this.point = point; }
    }

    private FunctionNode head; // голова циклического списка  20 usages
    private int count;         // количество точек в функции  16 usages

    //конструктор без параметров (обязательное требование Externalizable)
    public LinkedListTabulatedFunction(){  2 usages
        initializeList();
    }

    //метод записи объекта в поток
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        //записываем количество точек
        out.writeInt(count);
        //обходим все узлы списка и записываем координаты точек
        FunctionNode node = head.next;  //начинаем с первого реального узла
        while (node != head) {  //пока не вернемся к голове (циклический список)
            out.writeDouble(node.point.getX());  //x-координата
            out.writeDouble(node.point.getY());  //y-координата
            out.writeDouble(node.point.getX());  //x-координата
            out.writeDouble(node.point.getY());  //y-координата
            node = node.next;  //переходим к следующему узлу
        }
    }

    //метод чтения объекта из потока
    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        //инициализируем пустой список
        initializeList();
        //читаем количество точек
        int pointsCount = in.readInt();
        //читаем координаты и добавляем точки в список
        for (int i = 0; i < pointsCount; i++) {
            double x = in.readDouble();  //читаем x-координату
            double y = in.readDouble();  //читаем y-координату

            //создаем новый узел и добавляем его в конец списка
            FunctionNode newNode = addNodeToTail();
            newNode.point = new FunctionPoint(x, y);
        }
    }
```

```
9. СЕРИАЛИЗАЦИЯ С EXTERNALIZABLE:
==================================
Сравнение исходной и десериализованной функции:
x=0: исходная=0,000, прочитанная=0,000
x=1: исходная=1,000, прочитанная=1,000
x=2: исходная=2,000, прочитанная=2,000
x=3: исходная=3,000, прочитанная=3,000
x=4: исходная=4,000, прочитанная=4,000
x=5: исходная=5,000, прочитанная=5,000
x=6: исходная=6,000, прочитанная=6,000
x=7: исходная=7,000, прочитанная=7,000
x=8: исходная=8,000, прочитанная=8,000
x=9: исходная=9,000, прочитанная=9,000
x=10: исходная=10,000, прочитанная=10,000
```