Министерство науки и высшего образования Российской Федерации

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

Институт информатики и кибернетики Кафедра

технической кибернетики

Отчет по лабораторной работе №4

Дисциплина: «ООП»

Выполнил: Дорофеева Александра Алексеевна

Группа: 6201-120303D

Проверил: преподаватель Борисов Д.С.

Самара, 2025

Задание 1

```
@    public ArrayTabulatedFunction(FunctionPoint[] points) {  3 usages  new *
         if (points.length < 2) {
             throw new IllegalArgumentException("Количество точек должно быть не меньше двух");
         }

         for (int i = 1; i < points.length; i++) {
             if (points[i].getX() <= points[i - 1].getX() + EPSILON) {
                 throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");
             }
         }

         this.pointsCount = points.length;
         this.points = new FunctionPoint[pointsCount + 5];
         for (int i = 0; i < pointsCount; i++) {
             this.points[i] = new FunctionPoint(points[i]);
         }
     }
```

Реализация конструктора ArrayTabulatedFunction с массивом точек с проверкой упорядоченности

```
public LinkedListTabulatedFunction(FunctionPoint[] points) {  no usages  Dorofeeva Alexsandra
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не меньше двух");
    }

    for (int i = 1; i < points.length; i++) {
        if (points[i].getX() <= points[i - 1].getX() + EPSILON) {
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");
        }
    }

    initializeList();
    for (FunctionPoint point : points) {
        addNodeToTail().setPoint(new FunctionPoint(point));
    }
}
```

Реализация конструктора LinkedListTabulatedFunction с массивом точек

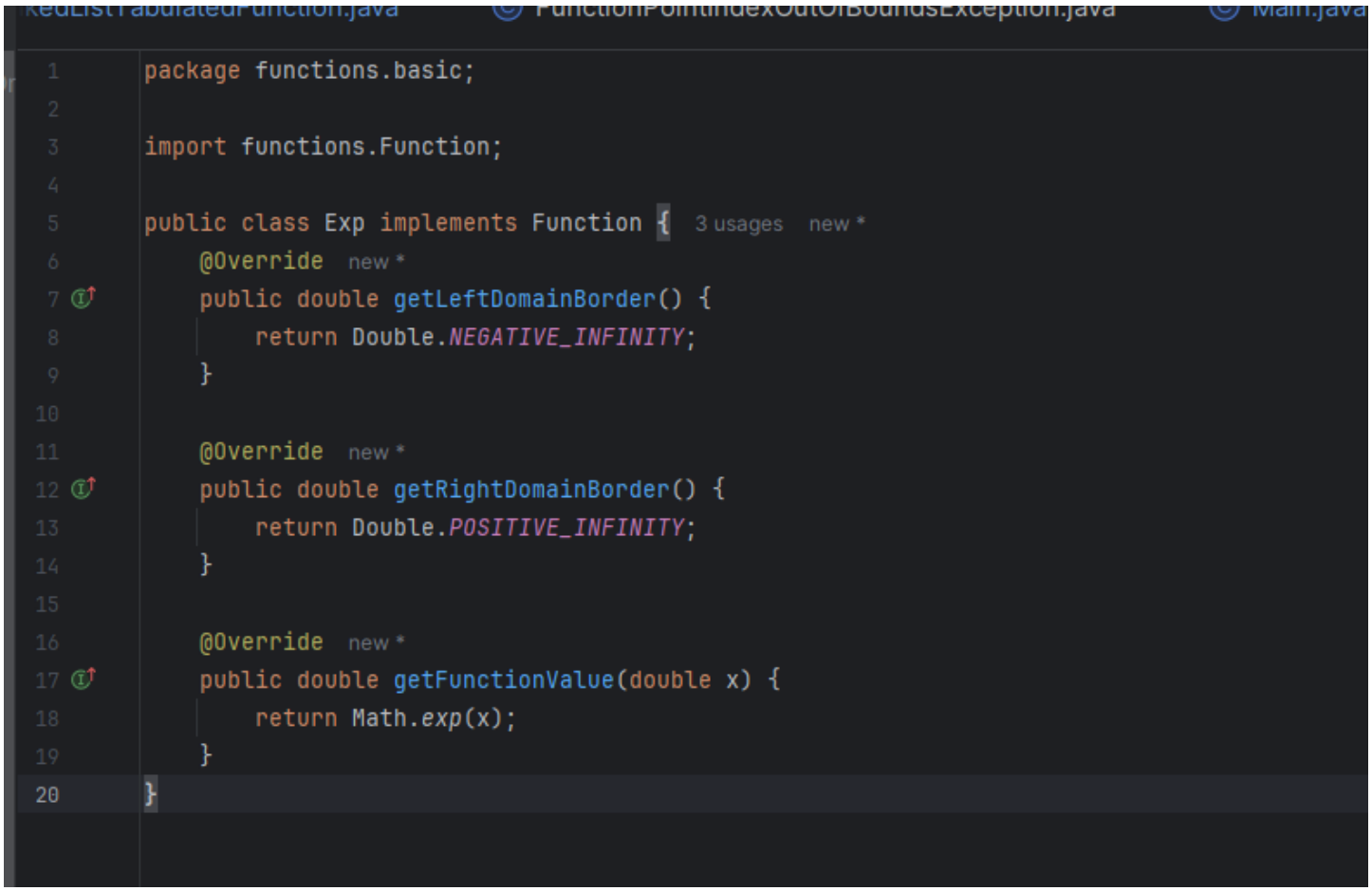
Задание 2

```java
package functions;

public interface Function {  new *
    double getLeftDomainBorder();  12 implementations  new *
    double getRightDomainBorder();  12 implementations  new *
    double getFunctionValue(double x);  14 implementations  new *
}
```

```java
package functions;

import java.io.Serializable;

public interface TabulatedFunction extends Function, Serializable {  25 usages  3 implementations  Dorofeeva Alexsandra
    int getPointsCount();  16 usages  3 implementations  Dorofeeva Alexsandra
    FunctionPoint getPoint(int index);  no usages  3 implementations  Dorofeeva Alexsandra
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;  no usages  3 implementations  Dorofeeva Alexsandra
    double getPointX(int index);  7 usages  3 implementations  Dorofeeva Alexsandra
    void setPointX(int index, double x) throws InappropriateFunctionPointException;  no usages  3 implementations  Dorofeeva Alexsandra
    double getPointY(int index);  7 usages  3 implementations  Dorofeeva Alexsandra
    void setPointY(int index, double y);  no usages  3 implementations  Dorofeeva Alexsandra
    void deletePoint(int index);  no usages  3 implementations  Dorofeeva Alexsandra
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  no usages  3 implementations  Dorofeeva Alexsandra
}
```

Создание интерфейса Function и наследование TabulatedFunction от него

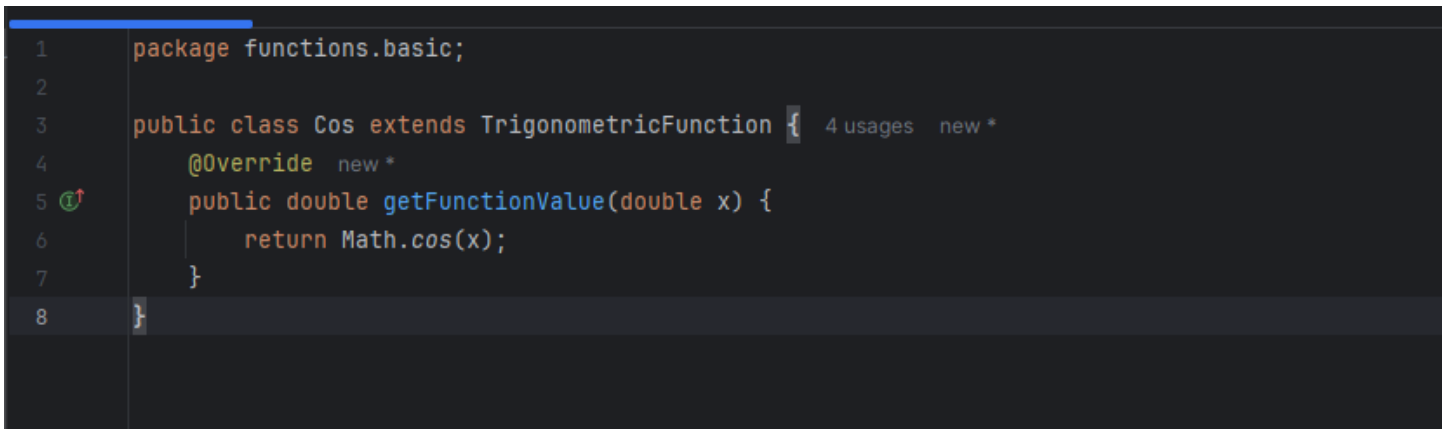
Задание 3


LinkedListTabulatedFunction.java
FunctionPointIndexOutOfBoundsException.java
Main.java
package functions.basic;
import functions.Function;
public class Exp implements Function { 3 usages new *
@Override new *
public double getLeftDomainBorder() {
return Double.NEGATIVE_INFINITY;
}
@Override new *
public double getRightDomainBorder() {
return Double.POSITIVE_INFINITY;
}
@Override new *
public double getFunctionValue(double x) {
return Math.exp(x);
}
}

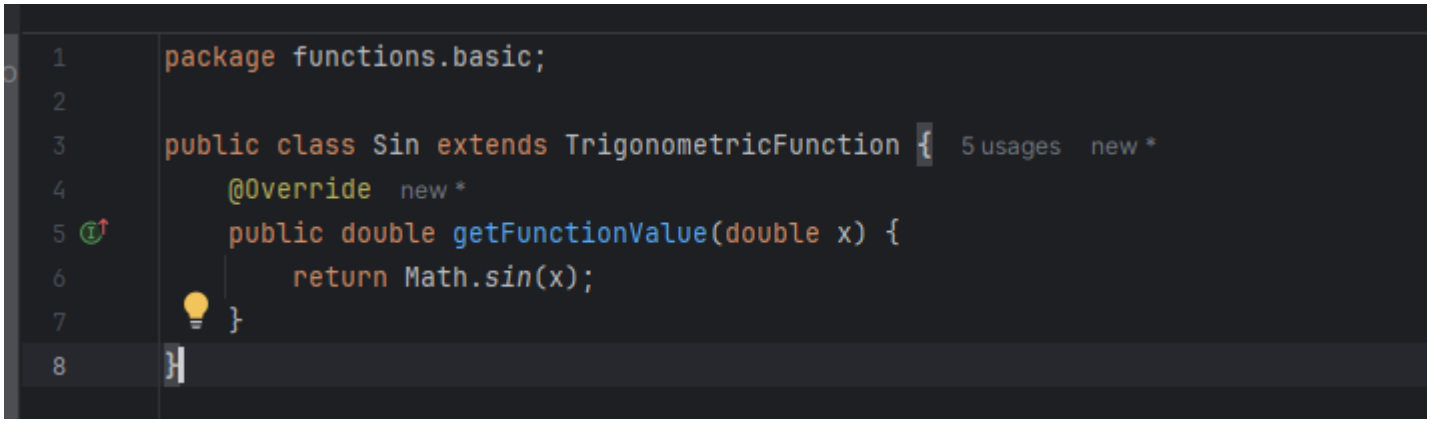
```java
package functions.basic;

import functions.Function;

public class Log implements Function {  3 usages  new *
    private double base;  2 usages

    public Log(double base) {  3 usages  new *
        if (base <= 0 || Math.abs(base - 1) < 1e-10) {
            throw new IllegalArgumentException("Основание логарифма должно быть положительным и не равно 1");
        }
        this.base = base;
    }

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

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

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

Реализация классов Exp и Log, вычисляющих экспоненту и логарифм

```java
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {  3 usages  3 inheritors  new *
    @Override  new *
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

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



```java
package functions.basic;

public class Cos extends TrigonometricFunction {  4 usages  new *
    @Override  new *
    public double getFunctionValue(double x) {
        return Math.cos(x);
    }
}
```

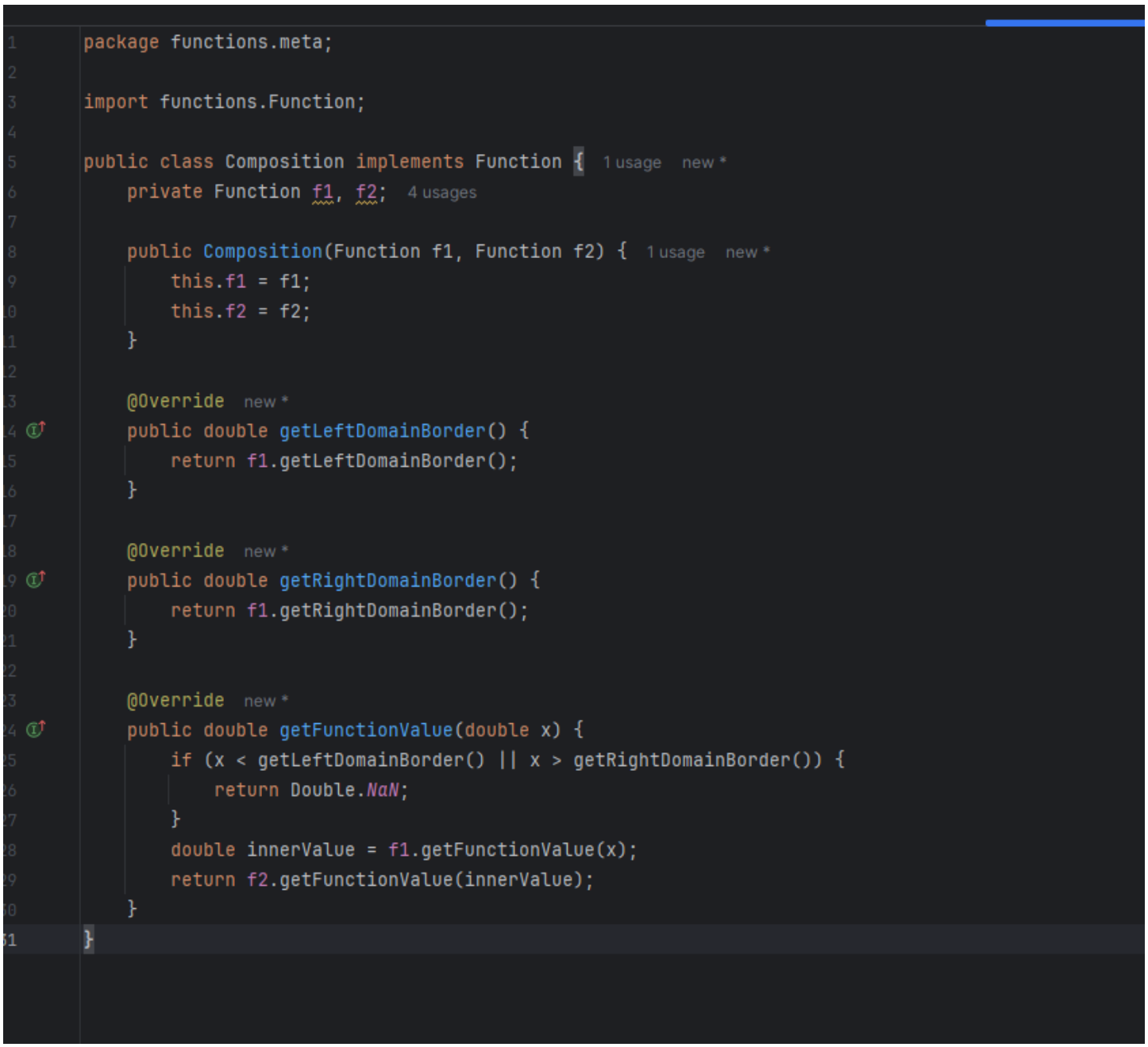


```java
package functions.basic;

public class Sin extends TrigonometricFunction {  5 usages  new *
    @Override  new *
    public double getFunctionValue(double x) {
        return Math.sin(x);
    }
}
```

Реализация тригонометрических функций с использованием наследования

Задание 4

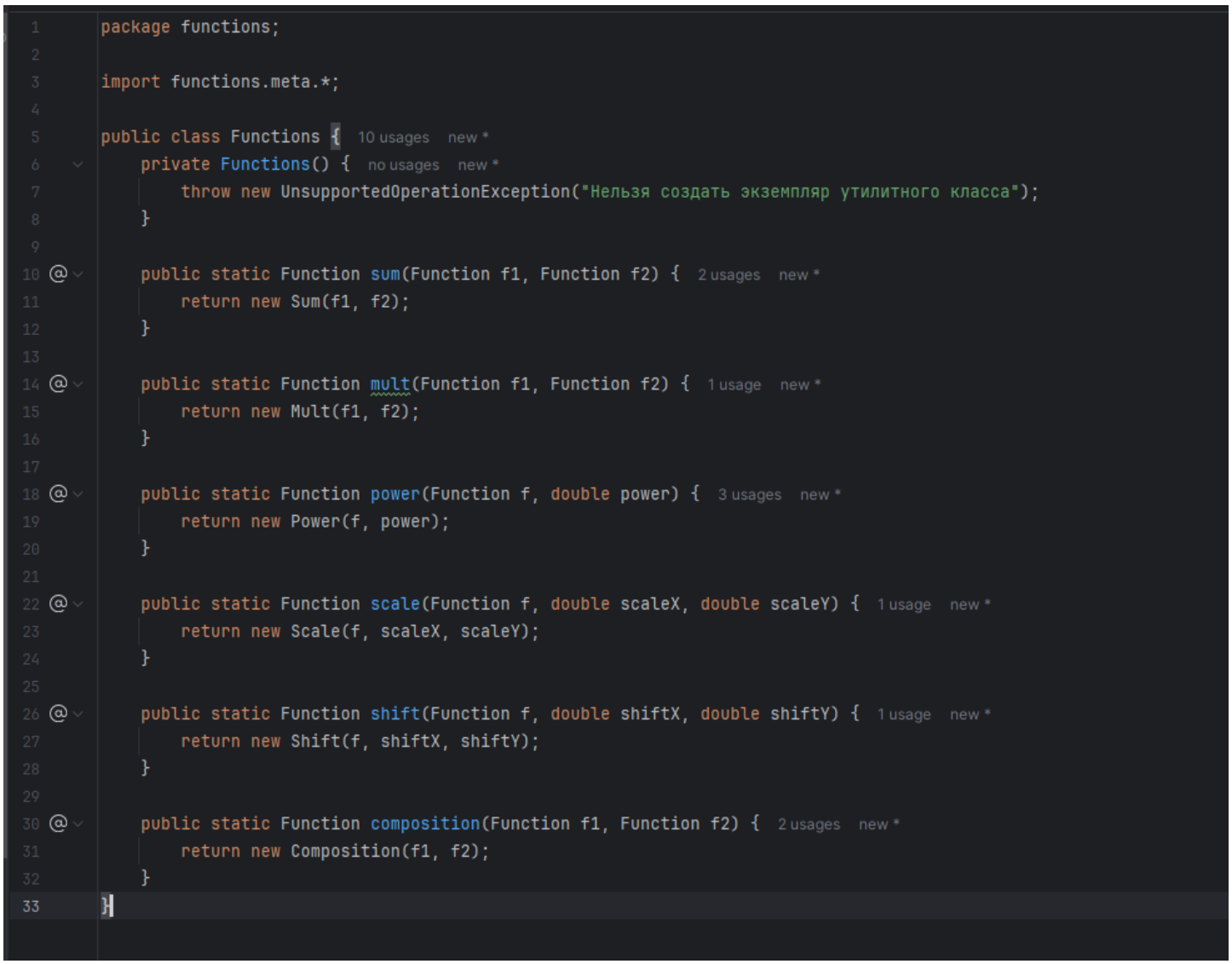
```java
package functions.meta;

import functions.Function;

public class Sum implements Function {  1 usage  new *
    private Function f1, f2;  4 usages

    public Sum(Function f1, Function f2) {  1 usage  new *
        this.f1 = f1;
        this.f2 = f2;
    }

    @Override  new *
    public double getLeftDomainBorder() {
        return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
    }

    @Override  new *
    public double getRightDomainBorder() {
        return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
    }

    @Override  new *
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        return f1.getFunctionValue(x) + f2.getFunctionValue(x);
    }
}
```



```java
package functions.meta;

import functions.Function;

public class Composition implements Function {  1 usage  new *
    private Function f1, f2;  4 usages

    public Composition(Function f1, Function f2) {  1 usage  new *
        this.f1 = f1;
        this.f2 = f2;
    }

    @Override  new *
    public double getLeftDomainBorder() {
        return f1.getLeftDomainBorder();
    }

    @Override  new *
    public double getRightDomainBorder() {
        return f1.getRightDomainBorder();
    }

    @Override  new *
    public double getFunctionValue(double x) {
        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
            return Double.NaN;
        }
        double innerValue = f1.getFunctionValue(x);
        return f2.getFunctionValue(innerValue);
    }
}
```

Реализация метафункций Sum и Composition

Задание 5



```java
package functions;

import functions.meta.*;

public class Functions {  10 usages  new *
    private Functions() {  no usages  new *
        throw new UnsupportedOperationException("Нельзя создать экземпляр утилитного класса");
    }

    public static Function sum(Function f1, Function f2) {  2 usages  new *
        return new Sum(f1, f2);
    }

    public static Function mult(Function f1, Function f2) {  1 usage  new *
        return new Mult(f1, f2);
    }

    public static Function power(Function f, double power) {  3 usages  new *
        return new Power(f, power);
    }

    public static Function scale(Function f, double scaleX, double scaleY) {  1 usage  new *
        return new Scale(f, scaleX, scaleY);
    }

    public static Function shift(Function f, double shiftX, double shiftY) {  1 usage  new *
        return new Shift(f, shiftX, shiftY);
    }

    public static Function composition(Function f1, Function f2) {  2 usages  new *
        return new Composition(f1, f2);
    }
}
```

Реализация утилитного класса Functions с фабричными методами для создания метафункций (сумма, умножение, возведение в степень, масштабирование, сдвиг, композиция)

Задание 6

```java
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не меньше двух");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }

    double functionLeftBorder = function.getLeftDomainBorder();
    double functionRightBorder = function.getRightDomainBorder();
    if (leftX < functionLeftBorder || rightX > functionRightBorder) {
        throw new IllegalArgumentException(
                "Границы табулирования выходят за область определения функции");
    }

    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        values[i] = function.getFunctionValue(x);
    }
    return new ArrayTabulatedFunction(leftX, rightX, values);
}
```

Реализация метода tabulate для табулирования функции на заданном интервале с проверкой корректности входных параметров и созданием объекта ArrayTabulatedFunction

Задание 7

```java
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {  3 usages  new *
    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 {  2 usages  new *
    DataInputStream dataIn = new DataInputStream(in);
    int pointsCount = dataIn.readInt();

    if (pointsCount < 2) {
        throw new IOException("Некорректное количество точек: " + pointsCount);
    }

    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);
}
```

Реализация методов сериализации и десериализации табулированной функции в бинарном формате (outputTabulatedFunction / inputTabulatedFunction)

Задание 8

Тестирование

```
"C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:G:\intellij IDEA\IntelliJ IDEA Commur
=== ЛАБОРАТОРНАЯ РАБОТА №4 ===
=== ТЕСТИРОВАНИЕ ВСЕХ ЗАДАНИЙ ===

=== ЗАДАНИЕ 3: БАЗОВЫЕ ФУНКЦИИ ===

1. Экспонента:
   Область определения: [-Infinity, Infinity]
   exp(0) = 1
   exp(1) = 2,718282
   exp(2) = 7,389056

2. Натуральный логарифм:
   Область определения: [0.0, Infinity]
   ln(1) = 0
   ln(e) = 1
   ln(10) = 2,302585

3. Тригонометрические функции:
   sin(0) = 0
   sin(π/2) = 1
   cos(0) = 1
   cos(π) = -1
   tan(π/4) = 1


=== ЗАДАНИЕ 4-5: МЕТАФУНКЦИИ ===

1. Сумма sin(x) + cos(x):
   при x=0: 1
   при x=π/4: 1,414214

2. Произведение sin(x) * cos(x):
   при x=π/4: 0,5

3. Квадрат синуса sin²(x):
   при x=π/6: 0,25
```

```
4. Сдвинутый синус sin(x-π/2):
   при x=π/2: 0

5. Масштабированный синус 3*sin(x/2):
   при x=π: 3

6. Композиция sin(cos(x)):
   при x=0: 1


=== ЗАДАНИЕ 6: ТАБУЛИРОВАНИЕ ===

Табулированный синус на [0, π] (10 точек):
Количество точек: 10
Область определения: [0.0, 3.141592653589793]

Точки функции:
   Точка 0: x=0,000, y=0,000000
   Точка 1: x=0,349, y=0,342020
   Точка 2: x=0,698, y=0,642788
   Точка 3: x=1,047, y=0,866025
   Точка 4: x=1,396, y=0,984808
   Точка 5: x=1,745, y=0,984808
   Точка 6: x=2,094, y=0,866025
   Точка 7: x=2,443, y=0,642788
   Точка 8: x=2,793, y=0,342020
   Точка 9: x=3,142, y=0,000000
```

```
Сравнение с точными значениями:
   x=0,000: точное=0,000000, приближ=0,000000, ошибка=0,000000
   x=0,524: точное=0,500000, приближ=0,492404, ошибка=0,007596
   x=0,785: точное=0,707107, приближ=0,698597, ошибка=0,008510
   x=1,047: точное=0,866025, приближ=0,866025, ошибка=0,000000
   x=1,571: точное=1,000000, приближ=0,984808, ошибка=0,015192
   x=2,094: точное=0,866025, приближ=0,866025, ошибка=0,000000
   x=2,356: точное=0,707107, приближ=0,698597, ошибка=0,008510
   x=2,618: точное=0,500000, приближ=0,492404, ошибка=0,007596
   x=3,142: точное=0,000000, приближ=0,000000, ошибка=0,000000


=== ЗАДАНИЕ 7: ВВОД/ВЫВОД ===

1. Исходная функция (парабола y=x²):
   (0,00, 0,00) ->    (1,00, 1,00) ->    (2,00, 4,00) ->    (3,00, 9,00) ->    (4,00, 16,00)

2. Бинарный формат:
   Записано байт: 84
   Прочитано точек: 5
   Функция восстановлена корректно: true

3. Текстовый формат:
   Текст: 5 0.0 0.0 1.0 1.0 2.0 4.0 3.0 9.0 4.0 16.0
   Длина текста: 42 символов
   Прочитано точек: 5
   Функция восстановлена корректно: true

4. Файловые операции:
   Бинарный файл создан: function_binary.dat
   Текстовый файл создан: function_text.txt
   Размер бинарного файла: 84 байт
   Размер текстового файла: 42 байт
```

```
=== ЗАДАНИЕ 8: ПОЛНОЕ ТЕСТИРОВАНИЕ ===

1. Создание объектов Sin и Cos:
   sin(π/6) = 0,5
   cos(π/3) = 0,5

2. Табулированные аналоги (10 точек на [0, π]):
   Табулированный sin имеет 10 точек
   Табулированный cos имеет 10 точек

3. Сумма квадратов sin²(x) + cos²(x):
   Проверка тождества sin²(x) + cos²(x) = 1:
     x=0,0: 1,0000000000 (ожидается 1.0000000000)
     x=0,2: 1,0000000000 (ожидается 1.0000000000)
     x=0,4: 1,0000000000 (ожидается 1.0000000000)
     x=0,6: 1,0000000000 (ожидается 1.0000000000)
     x=0,8: 1,0000000000 (ожидается 1.0000000000)
     x=1,0: 1,0000000000 (ожидается 1.0000000000)
     x=1,2: 1,0000000000 (ожидается 1.0000000000)
     x=1,4: 1,0000000000 (ожидается 1.0000000000)
     x=1,6: 1,0000000000 (ожидается 1.0000000000)
     x=1,8: 1,0000000000 (ожидается 1.0000000000)
     x=2,0: 1,0000000000 (ожидается 1.0000000000)
     x=2,2: 1,0000000000 (ожидается 1.0000000000)
     x=2,4: 1,0000000000 (ожидается 1.0000000000)
     x=2,6: 1,0000000000 (ожидается 1.0000000000)
     x=2,8: 1,0000000000 (ожидается 1.0000000000)
     x=3,0: 1,0000000000 (ожидается 1.0000000000)
```

```
4. Экспонента и файловые операции:
   Экспонента записана в exp_tabulated.txt
   Сравнение оригинальной и прочитанной экспоненты:
     x=0: orig=  1,000000, read=  1,000000, diff=0,0000000000
     x=1: orig=  2,718282, read=  2,718282, diff=0,0000000000
     x=2: orig=  7,389056, read=  7,389056, diff=0,0000000000
     x=3: orig= 20,085537, read= 20,085537, diff=0,0000000000
     x=4: orig= 54,598150, read= 54,598150, diff=0,0000000000
     x=5: orig=148,413159, read=148,413159, diff=0,0000000000
     x=6: orig=403,428793, read=403,428793, diff=0,0000000000
     x=7: orig=1096,633158, read=1096,633158, diff=0,0000000000
     x=8: orig=2980,957987, read=2980,957987, diff=0,0000000000
     x=9: orig=8103,083928, read=8103,083928, diff=0,0000000000
     x=10: orig=22026,465795, read=22026,465795, diff=0,0000000000

5. Логарифм и бинарные операции:
   Логарифм записан в log_tabulated.dat
   Проверка корректности чтения:
   Все значения совпадают!
```

Задание 9

```
package functions;

import java.io.Serializable;

public class ArrayTabulatedFunction implements TabulatedFunction, Serializable {  4 usages  Dorofeeva Alexsandra *
    private int pointsCount;  37 usages
    private FunctionPoint[] points;  36 usages
    private static final double EPSILON = 1e-9;  13 usages
    private static final long serialVersionUID = 1L;  no usages
```

Объявление класса ArrayTabulatedFunction, реализующего интерфейс TabulatedFunction с поддержкой сериализации (Serializable)

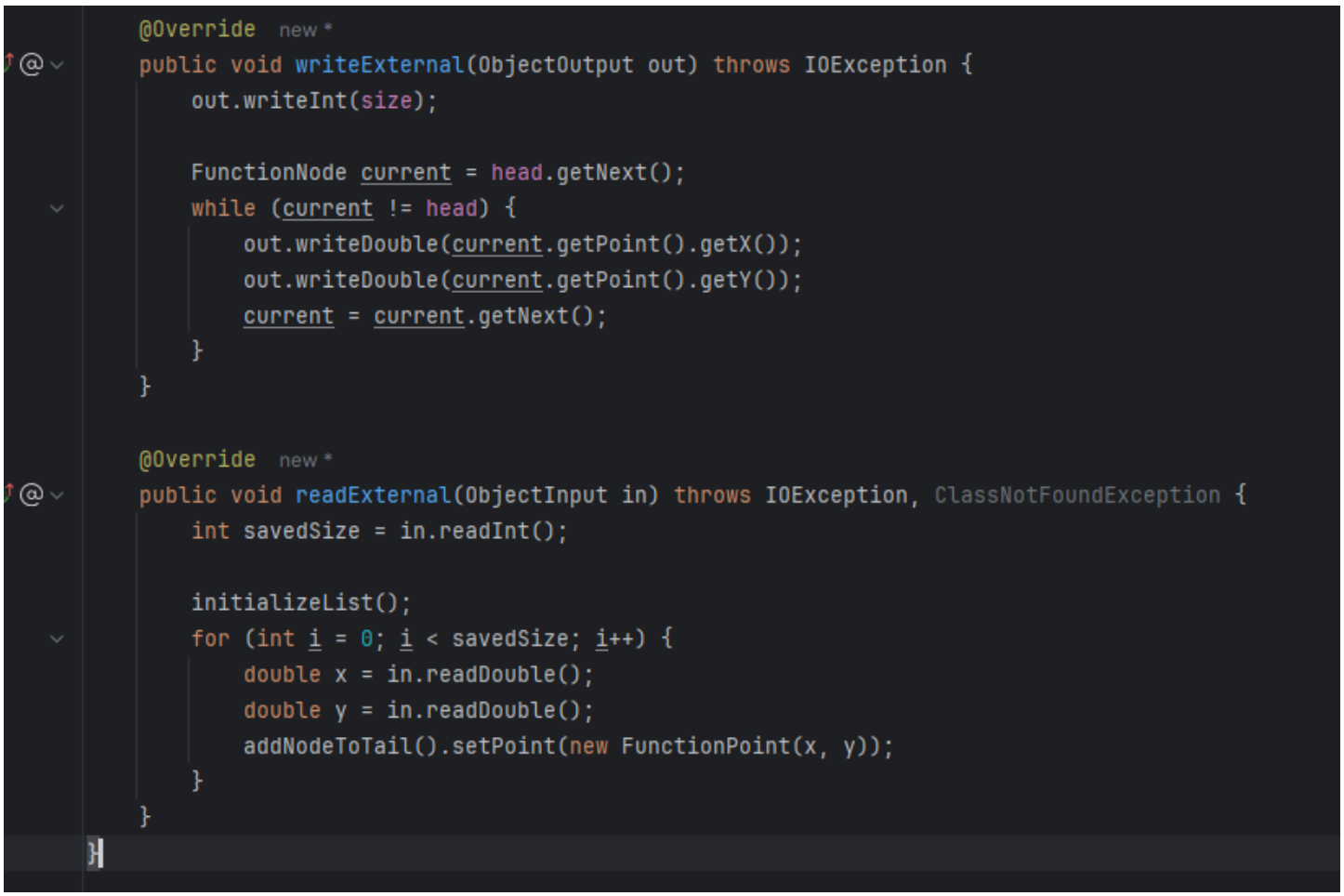
```
package functions;

import java.io.Serializable;

public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable {  no
    private static final long serialVersionUID = 2L;  no usages
```

```
    // Для стандартной сериализации (не Externalizable)
    private void writeObject(java.io.ObjectOutputStream out) throws java.io.IOException {  no usages  Dorofeeva Alexsandra *
        out.defaultWriteObject();
        out.writeInt(size);

        FunctionNode current = head.getNext();
        while (current != head) {
            out.writeDouble(current.getPoint().getX());
            out.writeDouble(current.getPoint().getY());
            current = current.getNext();
        }
    }

    private void readObject(java.io.ObjectInputStream in)  no usages  Dorofeeva Alexsandra *
            throws java.io.IOException, ClassNotFoundException {
        in.defaultReadObject();
        int savedSize = in.readInt();

        initializeList();
        for (int i = 0; i < savedSize; i++) {
            double x = in.readDouble();
            double y = in.readDouble();
            addNodeToTail().setPoint(new FunctionPoint(x, y));
        }
    }
}
```

382:2

Класс LinkedListTabulatedFunction с поддержкой Serializable и ручной реализацией сериализации через writeObject/readObject

```
package functions;

import java.io.*;

public class LinkedListTabulatedFunctionExternalizable implements TabulatedFunction, Externalizable {  4 usages  new *
    private static final long serialVersionUID = 2L;  no usages
```



```
    @Override  new *
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeInt(size);

        FunctionNode current = head.getNext();
        while (current != head) {
            out.writeDouble(current.getPoint().getX());
            out.writeDouble(current.getPoint().getY());
            current = current.getNext();
        }
    }

    @Override  new *
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        int savedSize = in.readInt();

        initializeList();
        for (int i = 0; i < savedSize; i++) {
            double x = in.readDouble();
            double y = in.readDouble();
            addNodeToTail().setPoint(new FunctionPoint(x, y));
        }
    }
}
```

Класс LinkedListTabulatedFunctionExternalizable, реализующий интерфейс Externalizable для ручного управления процессом сериализации

```java
private static void testSerialization() throws IOException, ClassNotFoundException {  1 usage  new *
    System.out.println("\n\n=== ЗАДАНИЕ 9: СЕРИАЛИЗАЦИЯ ===");

    // 1. Проверка тождества ln(e^x) = x
    System.out.println("\n1. Проверка тождества ln(e^x) = x:");
    Function exp = new Exp();
    Function log = new Log(Math.E);
    Function logOfExp = Functions.composition(log, exp);

    // Проверяем значения от 0 до 10
    for (int i = 0; i <= 10; i++) {
        double x = i;
        double y = logOfExp.getFunctionValue(x);
        System.out.printf("   x=%.1f: ln(e^%.1f) = %.10f (ожидается %.1f)%n",
                x, x, y, x);
        // Проверяем, что значение близко к ожидаемому
        if (Math.abs(y - x) > 1e-9) {
            System.out.printf("   ВНИМАНИЕ: разница = %.10f%n", Math.abs(y - x));
        }
    }

    // Создаем функцию для сериализации (композиция ln(e^x) = x)
    TabulatedFunction tabLogExp = TabulatedFunctions.tabulate(logOfExp,  leftX: 0,  rightX: 10,  pointsCount: 11);

    System.out.println("\n2. Функция для сериализации: ln(e^x) = x");
    System.out.println("   Количество точек: " + tabLogExp.getPointsCount());
    System.out.println("   Все точки оригинала:");
    for (int i = 0; i < tabLogExp.getPointsCount(); i++) {
        System.out.printf("   Точка %d: x=%.3f, y=%.6f%n",
                i, tabLogExp.getPointX(i), tabLogExp.getPointY(i));
    }

    // Сериализация с Serializable
    System.out.println("\n3. Сериализация с использованием Serializable:");
    try (ObjectOutputStream oos = new ObjectOutputStream(
            new FileOutputStream( name: "function_serializable.dat"))) {
        oos.writeObject(tabLogExp);
    }
```

```
        File serializableFile = new File( pathname: "function_serializable.dat");
        System.out.println("   Записано в: function_serializable.dat");
        System.out.println("   Размер файла: " + serializableFile.length() + " байт");

        // Десериализация
        TabulatedFunction deserialized;
        try (ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream( name: "function_serializable.dat"))) {
            deserialized = (TabulatedFunction) ois.readObject();
        }

        System.out.println("\n4. Проверка после десериализации Serializable:");
        System.out.println("   Количество точек: " + deserialized.getPointsCount());
        System.out.println("   Все точки после десериализации:");
        for (int i = 0; i < deserialized.getPointsCount(); i++) {
            System.out.printf("   Точка %d: x=%.3f, y=%.6f%n",
                    i, deserialized.getPointX(i), deserialized.getPointY(i));
        }

        // Сравнение оригинальной и десериализованной функции
        System.out.println("\n5. Сравнение оригинальной и десериализованной функции:");
        boolean serializableMatch = compareFunctions(tabLogExp, deserialized);
        System.out.println("   Функции идентичны: " + serializableMatch);

        if (!serializableMatch) {
            System.out.println("   Расхождения:");
            for (int i = 0; i < tabLogExp.getPointsCount(); i++) {
                double origX = tabLogExp.getPointX(i);
                double origY = tabLogExp.getPointY(i);
                double readX = deserialized.getPointX(i);
                double readY = deserialized.getPointY(i);

                if (Math.abs(origX - readX) > 1e-9 || Math.abs(origY - readY) > 1e-9) {
                    System.out.printf("   Точка %d: оригинал (%.6f, %.6f) vs прочитано (%.6f, %.6f)%n",
                            i, origX, origY, readX, readY);
                }
            }
```

```
        }
    }

    // Создаем Externalizable версию (такой же набор точек для корректного сравнения)
    System.out.println("\n6. Создание Externalizable версии:");
    FunctionPoint[] points = new FunctionPoint[11];
    for (int i = 0; i <= 10; i++) {
        double x = i;
        double y = logOfExp.getFunctionValue(x); // тоже ln(e^x) = x
        points[i] = new FunctionPoint(x, y);
    }
    LinkedListTabulatedFunctionExternalizable externalizableFunc =
            new LinkedListTabulatedFunctionExternalizable(points);

    System.out.println("   Количество точек: " + externalizableFunc.getPointsCount());
    System.out.println("   Все точки оригинала Externalizable:");
    for (int i = 0; i < externalizableFunc.getPointsCount(); i++) {
        System.out.printf("   Точка %d: x=%.1f, y=%.1f%n",
                i, externalizableFunc.getPointX(i), externalizableFunc.getPointY(i));
    }

    // Сериализация с Externalizable
    try (ObjectOutputStream oos = new ObjectOutputStream(
            new FileOutputStream( name: "function_externalizable.dat"))) {
        oos.writeObject(externalizableFunc);
    }
    File externalizableFile = new File( pathname: "function_externalizable.dat");
    System.out.println("\n7. Сериализация Externalizable:");
    System.out.println("   Записано в: function_externalizable.dat");
    System.out.println("   Размер файла: " + externalizableFile.length() + " байт");

    // Десериализация
    LinkedListTabulatedFunctionExternalizable deserializedExternal;
    try (ObjectInputStream ois = new ObjectInputStream(
            new FileInputStream( name: "function_externalizable.dat"))) {
        deserializedExternal = (LinkedListTabulatedFunctionExternalizable) ois.readObject();
    }
```

```
        System.out.println("\n8. Проверка после десериализации Externalizable:");
        System.out.println("   Количество точек: " + deserializedExternal.getPointsCount());
        System.out.println("   Все точки после десериализации:");
        for (int i = 0; i < deserializedExternal.getPointsCount(); i++) {
            System.out.printf("   Точка %d: x=%.1f, y=%.1f%n",
                    i, deserializedExternal.getPointX(i), deserializedExternal.getPointY(i));
        }

        // Проверка совпадения Externalizable
        System.out.println("\n9. Проверка совпадения Externalizable:");
        boolean externalizableMatch = compareFunctions(externalizableFunc, deserializedExternal);
        System.out.println("   Функции идентичны: " + externalizableMatch);

        System.out.println("\n10. Сравнение размеров файлов:");
        System.out.println("   Serializable: " + serializableFile.length() + " байт");
        System.out.println("   Externalizable: " + externalizableFile.length() + " байт");
        System.out.println("   Разница: " +
                Math.abs(serializableFile.length() - externalizableFile.length()) + " байт");

        System.out.println("\n11. Объяснение переопределения writeObject/readObject:");
        System.out.println("    В LinkedListTabulatedFunction методы writeObject/readObject");
        System.out.println("    переопределены для:");
        System.out.println("    - Контроля над процессом сериализации");
        System.out.println("    - Игнорирования transient полей (lastAccessedNode, lastAccessedIndex)");
        System.out.println("    - Восстановления циклической структуры списка");
        System.out.println("    - Избежания проблем с циклическими ссылками");
        System.out.println("    - Сохранения только данных (координат точек), не структуры списка");
        System.out.println("\n    В Externalizable версии:");
        System.out.println("    - Полный контроль над форматом данных");
        System.out.println("    - Более эффективное использование места");
        System.out.println("    - Возможность обработки разных версий формата");
    }
```

Метод testSerialization, демонстрирующий процесс сериализации и десериализации функций с использованием Serializable и Externalizable, а также сравнение размеров файлов

```
=== ЗАДАНИЕ 9: СЕРИАЛИЗАЦИЯ ===

1. Проверка тождества ln(e^x) = x:
   x=0,0: ln(e^0,0) = NaN (ожидается 0,0)
   x=1,0: ln(e^1,0) = 1,0000000000 (ожидается 1,0)
   x=2,0: ln(e^2,0) = 2,0000000000 (ожидается 2,0)
   x=3,0: ln(e^3,0) = 3,0000000000 (ожидается 3,0)
   x=4,0: ln(e^4,0) = 4,0000000000 (ожидается 4,0)
   x=5,0: ln(e^5,0) = 5,0000000000 (ожидается 5,0)
   x=6,0: ln(e^6,0) = 6,0000000000 (ожидается 6,0)
   x=7,0: ln(e^7,0) = 7,0000000000 (ожидается 7,0)
   x=8,0: ln(e^8,0) = 8,0000000000 (ожидается 8,0)
   x=9,0: ln(e^9,0) = 9,0000000000 (ожидается 9,0)
   x=10,0: ln(e^10,0) = 10,0000000000 (ожидается 10,0)

2. Функция для сериализации: ln(e^x) = x
   Количество точек: 11
   Все точки оригинала:
   Точка 0: x=0,000, y=NaN
   Точка 1: x=1,000, y=1,000000
   Точка 2: x=2,000, y=2,000000
   Точка 3: x=3,000, y=3,000000
   Точка 4: x=4,000, y=4,000000
   Точка 5: x=5,000, y=5,000000
   Точка 6: x=6,000, y=6,000000
   Точка 7: x=7,000, y=7,000000
   Точка 8: x=8,000, y=8,000000
   Точка 9: x=9,000, y=9,000000
   Точка 10: x=10,000, y=10,000000

3. Сериализация с использованием Serializable:
   Записано в: function_serializable.dat
   Размер файла: 445 байт
```

```
4. Проверка после десериализации Serializable:
   Количество точек: 11
   Все точки после десериализации:
   Точка 0: x=0,000, y=NaN
   Точка 1: x=1,000, y=1,000000
   Точка 2: x=2,000, y=2,000000
   Точка 3: x=3,000, y=3,000000
   Точка 4: x=4,000, y=4,000000
   Точка 5: x=5,000, y=5,000000
   Точка 6: x=6,000, y=6,000000
   Точка 7: x=7,000, y=7,000000
   Точка 8: x=8,000, y=8,000000
   Точка 9: x=9,000, y=9,000000
   Точка 10: x=10,000, y=10,000000

5. Сравнение оригинальной и десериализованной функции:
   Функции идентичны: true

6. Создание Externalizable версии:
   Количество точек: 11
   Все точки оригинала Externalizable:
   Точка 0: x=0,0, y=NaN
   Точка 1: x=1,0, y=1,0
   Точка 2: x=2,0, y=2,0
   Точка 3: x=3,0, y=3,0
   Точка 4: x=4,0, y=4,0
   Точка 5: x=5,0, y=5,0
   Точка 6: x=6,0, y=6,0
   Точка 7: x=7,0, y=7,0
   Точка 8: x=8,0, y=8,0
   Точка 9: x=9,0, y=9,0
   Точка 10: x=10,0, y=10,0
```

```
7. Сериализация Externalizable:
   Записано в: function_externalizable.dat
   Размер файла: 255 байт

8. Проверка после десериализации Externalizable:
   Количество точек: 11
   Все точки после десериализации:
   Точка 0: x=0,0, y=NaN
   Точка 1: x=1,0, y=1,0
   Точка 2: x=2,0, y=2,0
   Точка 3: x=3,0, y=3,0
   Точка 4: x=4,0, y=4,0
   Точка 5: x=5,0, y=5,0
   Точка 6: x=6,0, y=6,0
   Точка 7: x=7,0, y=7,0
   Точка 8: x=8,0, y=8,0
   Точка 9: x=9,0, y=9,0
   Точка 10: x=10,0, y=10,0

9. Проверка совпадения Externalizable:
   Функции идентичны: true

10. Сравнение размеров файлов:
   Serializable: 445 байт
   Externalizable: 255 байт
   Разница: 190 байт

11. Объяснение переопределения writeObject/readObject:
    В LinkedListTabulatedFunction методы writeObject/readObject
    переопределены для:
    - Контроля над процессом сериализации
    - Игнорирования transient полей (lastAccessedNode, lastAccessedIndex)
    - Восстановления циклической структуры списка
    - Избежания проблем с циклическими ссылками
    - Сохранения только данных (координат точек), не структуры списка
```

```
    В Externalizable версии:
    - Полный контроль над форматом данных
    - Более эффективное использование места
    - Возможность обработки разных версий формата
```

Проверка