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»

Тема «Наследование. Ввод и вывод»

Выполнил: Сучков Борис Антонович

Группа: 6201-120303D

Самара 2025

Задание на лабораторную работу

Расширить возможности пакета для работы с функциями одной переменной добавив интерфейсы и классы для аналитически заданных функций, а также методы ввода и вывода табулированных функций.

Не забывайте использовать машинный эпсилон при сравнении вещественных чисел.

Задания

Задание 1

В классах `ArrayTabulatedFunction` и `LinkedListTabulatedFunction` добавьте 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 должны выбрасывать исключение `IllegalArgumentException`. При написании конструкторов обеспечьте корректную инкапсуляцию.

Задание 2

В пакете `functions` создайте 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.
Исключите соответствующие методы из интерфейса `TabulatedFunction` и сделайте так, чтобы он расширял интерфейс `Function`. Теперь табулированные функции буду частным случаем функций одной переменной.

Задание 3

Создайте пакет `functions.basic`, в нём будут описаны классы ряда функций, заданных аналитически.
Создайте в пакете публичный класс `Exp`, объекты которого должны вычислять значение экспоненты. Класс должен реализовывать интерфейс `Function`. Для вычисления экспоненты следует воспользоваться методом `Math.exp()`, а для возвращения значений границ области определения – константами из класса `Double`.

Аналогично, создайте класс `Log`, объекты которого должны вычислять значение логарифма по заданному основанию. Основание должно передаваться как параметр конструктора. Для вычисления логарифма следует воспользоваться методом `Math.log()`.
Прежде, чем перейти к описанию классов для тригонометрических функций (синуса, косинуса и тангенса), обратите внимание на то, что область определения этих функций совпадает, поэтому описывать одинаковые методы в этих классах будет достаточно странным. Проще будет описать базовый класс с реализацией этих методов, а классы конкретных функций наследовать от него.

Создайте класс `TrigonometricFunction`, реализующий интерфейс `Function` и описывающий методы получения границ области определения.
Создайте наследующие от него публичные классы `Sin`, `Cos` и `Tan`, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса. Для получения значений следует воспользоваться методами `Math.sin()`, `Math.cos()` и `Math.tan()`.

Задание 4

Создайте пакет `functions.meta`, в нём будут описаны классы функций, позволяющие комбинировать функции.
Создайте класс `Sum`, объекты которого представляют собой функции, являющиеся суммой двух других функций. Класс должен реализовывать интерфейс `Function`. Конструктор класса должен получать ссылки типа `Function` на объекты суммируемых функций, а область определения функции должна получаться как пересечение областей определения исходных функций.
Аналогично, создайте класс `Mult`, объекты которого представляют собой функции, являющиеся произведением двух других функций.
Создайте класс `Power`, объекты которого представляют собой функции, являющиеся степенью другой функции. Конструктор класса должен получать ссылку на объекты базовой функции и степень, в которую должны возводиться её значения. Область определения функции можно считать совпадающей с областью определения исходной функции (хотя математически это не всегда так).
Создайте класс `Scale`, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат. Конструктор класса должен получать ссылку на объект исходной функции, а также коэффициенты масштабирования вдоль оси абсцисс и оси ординат. Область определения функции должна получаться из области определения исходной функции масштабированием вдоль оси абсцисс, а значение функции – масштабированием значения исходной функции вдоль оси ординат. Коэффициенты масштабирования могут быть отрицательными.
Аналогично, создайте класс `Shift`, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат.
Также создайте класс `Composition`, объекты которого описывают композицию двух исходных функций. Конструктор класса должен получать ссылки на объекты первой и второй функции. Область определения функции можно считать

совпадающей с областью определения исходной функции (хотя математически это не всегда так).

Задание 5

В пакете `functions` создайте класс `Functions`, содержащий вспомогательные статические методы для работы с функциями. Сделайте так, чтобы в программе вне этого класса нельзя было создать его объект. Класс должен содержать 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`.

Задание 6

В пакете `functions` создайте класс `TabulatedFunctions`, содержащий вспомогательные статические методы для работы с табулированными функциями. Сделайте так, чтобы в программе вне этого класса нельзя было создать его объект.
Опишите 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 должен выбрасывать исключение `IllegalArgumentException`.
Поскольку метод возвращает ссылку интерфейсного типа, можно возвращать объект любого из классов, реализующих этот интерфейс. В последующих работах в код будет добавлена возможность выбора класса для создания экземпляра.

Задание 7

В класс `TabulatedFunctions` добавьте следующие методы.
Метод вывода табулированной функции в байтовый поток `public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out)` должен в указанный поток вывести 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)` должен считывать из указанного потока данные о табулированной функции, создавать и настраивать её объект и возвращать его из метода.
Метод записи табулированной функции в символьный поток `public static void writeTabulatedFunction(TabulatedFunction function, Writer out)` должен в указанный поток вывести значения, по которым потом можно будет восстановить табулированную функцию, а именно количество точек в ней и значения координат точек. Проще всего считать, что значения записываются в строку и разделяются пробелами.
Метод чтения табулированной функции из символьного потока `public static TabulatedFunction readTabulatedFunction(Reader in)` должен считывать из указанного потока данные о табулированной функции, создавать и настраивать её объект и возвращать его из метода.
При написании методов в первых трёх случаях необходимо воспользоваться потоками-обёртками, облегчающими ввод и вывод данных в требующейся форме, а в четвёртом случае – классом `StreamTokenizer`.
При написании методов, считывающих табулированную функцию, следует считать, что данные в потоке записаны правильные данные (проверку корректности вводимых данных делать не следует).

Поскольку методы ввода и чтения возвращают ссылку интерфейсного типа, можно возвращать объект любого из классов, реализующих этот интерфейс. В последующих работах в код будет добавлена возможность выбора класса для создания экземпляра.

Подумайте и обоснуйте, как следует в этих методах поступить с возникающим исключением `IOException`.
Подумайте и обоснуйте, следует ли закрывать потоки внутри этих методов.

Задание 8

Проверьте работу написанных классов.

Создайте по одному объекту классов `Sin` и `Cos`, выведите в консоль значения этих функций на отрезке от 0 до π с шагом 0,1.
С помощью метода `TabulatedFunctions.tabulate()` создайте табулированные аналоги этих функций на отрезке от 0 до π с 10 точками. Выведите в консоль значения этих

функций на отрезке от 0 до π с шагом 0,1 и сравните со значениями исходных функций.
С помощью методов класса `Functions` создайте объект функции, являющейся суммой квадратов табулированных аналогов синуса и косинуса. Выведите в консоль значения этой функций на отрезке от 0 до π с шагом 0,1. Попробуйте изменять количество точек в табулированных аналогах и исследуйте, как при этом изменяется результирующая функция.
С помощью метода `TabulatedFunctions.tabulate()` создайте табулированный аналог экспоненты на отрезке от 0 до 10 с 11 точками. С помощью метода `TabulatedFunctions.writeTabulatedFunction()` выведите его в файл. Далее с помощью метода `TabulatedFunctions.readTabulatedFunction()` считайте табулированную функцию из этого файла. Выведите и сравните значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
С помощью метода `TabulatedFunctions.tabulate()` создайте табулированный аналог логарифма по натуральному основанию на отрезке от 0 до 10 с 11 точками. С помощью метода `TabulatedFunctions.outputTabulatedFunction()` выведите его в файл (имя файла должно отличаться от предыдущего случая). Далее с помощью метода `TabulatedFunctions.inputTabulatedFunction()` считайте табулированную функцию из этого файла. Выведите и сравните значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
Изучите содержимое всех получаемых файлов и сделайте выводы о преимуществах и недостатках каждого из форматов хранения.

Задание 9

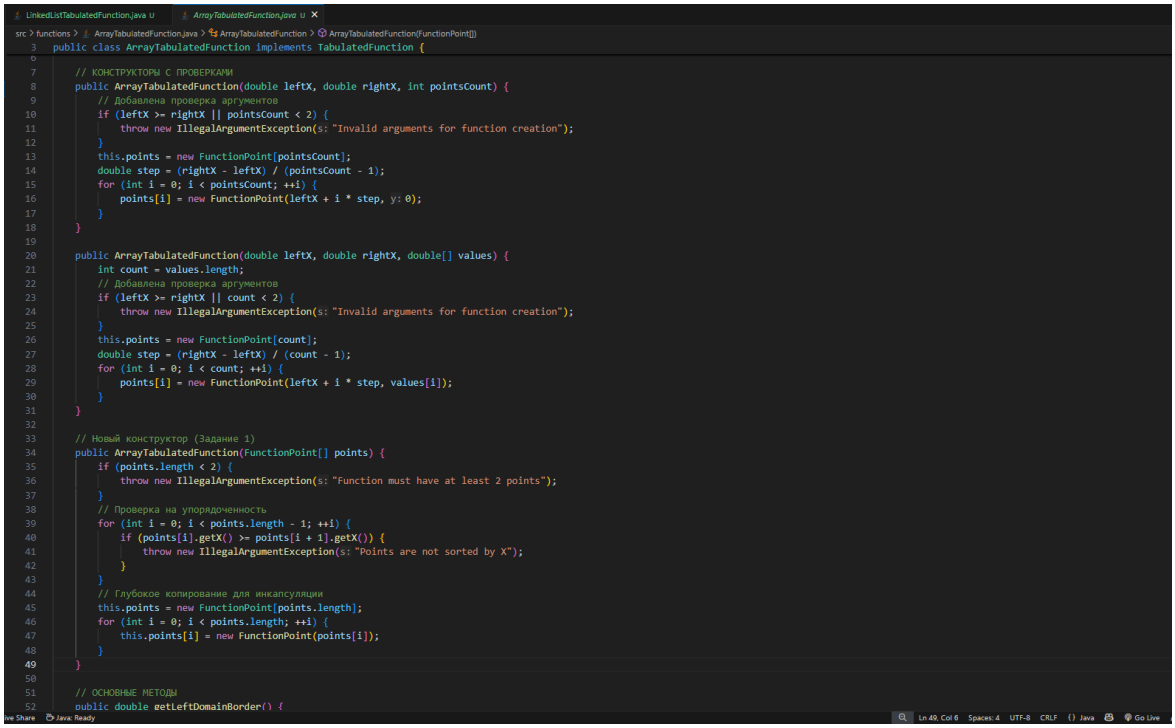
Сделайте так, чтобы объекты всех классов, реализующих интерфейс `TabulatedFunction`, были сериализуемыми.
Для этого рассмотрите два случая:

1. с использованием интерфейса `java.io.Serializable`
2. с использованием интерфейса `java.io.Externalizable`

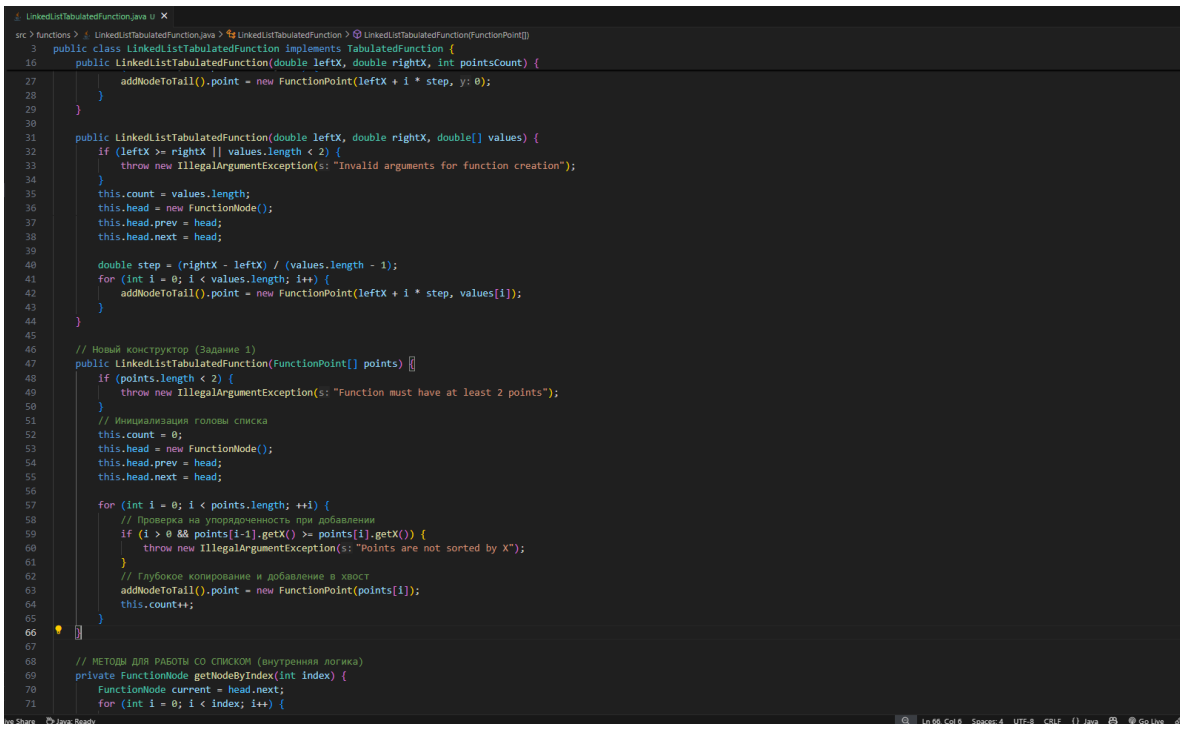
Проверьте работу написанных классов. С помощью метода `TabulatedFunctions.tabulate()` и метода класса `Functions` создайте табулированный аналог логарифма по натуральному основанию, взятого от экспоненты на отрезке от 0 до 10 с 11 точками. Сериализуйте полученный объект в файл (имя файла должно отличаться от предыдущих случаев). Далее десериализуйте табулированную функцию из этого файла. Выведите значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
Изучите содержимое файлов, получаемых при реализации механизма сериализации с использованием интерфейса `java.io.Serializable` и при реализации механизма сериализации с использованием интерфейса `java.io.Externalizable`.
Сделайте выводы о преимуществах и недостатках каждого из способов.

Задание 1

В классах ArrayTabulatedFunction и LinkedListTabulatedFunction я добавил 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 обеспечил корректную инкапсуляцию.



```java
public class ArrayTabulatedFunction implements TabulatedFunction {

    // КОНСТРУКТОРЫ С ПРОВЕРКАМИ
    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
        // Добавлена проверка аргументов
        if (leftX >= rightX || pointsCount < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; ++i) {
            points[i] = new FunctionPoint(leftX + i * step, y: 0);
        }
    }

    public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
        int count = values.length;
        // Добавлена проверка аргументов
        if (leftX >= rightX || count < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.points = new FunctionPoint[count];
        double step = (rightX - leftX) / (count - 1);
        for (int i = 0; i < count; ++i) {
            points[i] = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    // Новый конструктор (Задание 1)
    public ArrayTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException(s: "Function must have at least 2 points");
        }
        // Проверка на упорядоченность
        for (int i = 0; i < points.length - 1; ++i) {
            if (points[i].getX() >= points[i + 1].getX()) {
                throw new IllegalArgumentException(s: "Points are not sorted by X");
            }
        }
        // Глубокое копирование для инкапсуляции
        this.points = new FunctionPoint[points.length];
        for (int i = 0; i < points.length; ++i) {
            this.points[i] = new FunctionPoint(points[i]);
        }
    }

    // ОСНОВНЫЕ МЕТОДЫ
    public double getLeftDomainBorder() {
```



```
public class LinkedListTabulatedFunction implements TabulatedFunction {
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
            addNodeToTail().point = new FunctionPoint(leftX + i * step, y: 0);
        }
    }

    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
        if (leftX >= rightX || values.length < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.count = values.length;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            addNodeToTail().point = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    // Новый конструктор (Задание 1)
    public LinkedListTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException(s: "Function must have at least 2 points");
        }
        // Инициализация головы списка
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        for (int i = 0; i < points.length; ++i) {
            // Проверка на упорядоченность при добавлении
            if (i > 0 && points[i-1].getX() >= points[i].getX()) {
                throw new IllegalArgumentException(s: "Points are not sorted by X");
            }
            // Глубокое копирование и добавление в хвост
            addNodeToTail().point = new FunctionPoint(points[i]);
            this.count++;
        }
    }

    // МЕТОДЫ ДЛЯ РАБОТЫ СО СПИСКОМ (внутренняя логика)
    private FunctionNode getNodeByIndex(int index) {
        FunctionNode current = head.next;
        for (int i = 0; i < index; i++) {
```

Задание 2

В пакете functions я создал 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.
Я исключил соответствующие методы из интерфейса TabulatedFunction и сделал так, чтобы он расширял интерфейс Function. Теперь табулированные функции будут частным случаем функций одной переменной.

Function.java U × TabulatedFunction.java U

src > functions > Function.java > ...

```
1  package functions;
2
3  public interface Function {
4      double getLeftDomainBorder();
5      double getRightDomainBorder();
6      double getFunctionValue(double x);
7  }
8
```

```java
package functions;

/* 1. Расширяем Function (extends Function)
 2. Добавляем Serializable для Задания 9 */
public interface TabulatedFunction extends Function, java.io.Serializable {
    /* Методы getLeftDomainBorder, getRightDomainBorder, getFunctionValue
     теперь наследуются из Function. Удаляем их отсюда. */

    int getPointsCount();
    FunctionPoint getPoint(int index);
    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    double getPointX(int index);
    void setPointX(int index, double x) throws InappropriateFunctionPointException;
    double getPointY(int index);
    void setPointY(int index, double y);
    void deletePoint(int index);
    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
}
```

Задание 3

Я создал пакет functions.basic, в нём описаны классы ряда функций, заданных аналитически.

Я также создал в пакете публичный класс Exp, объекты которого вычисляют значение экспоненты. Класс реализовывает интерфейс Function. Для вычисления экспоненты я воспользовался методом Math.exp(), а для возвращения значений границ области определения – константами из класса Double.

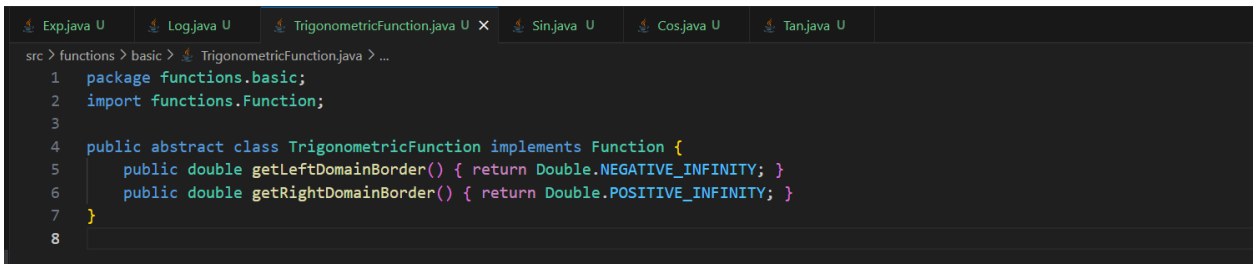
Аналогично, я создал класс Log, объекты которого вычисляют значение логарифма по заданному основанию. Основание передаётся как параметр конструктора. Для вычисления логарифма я воспользовался методом Math.log().

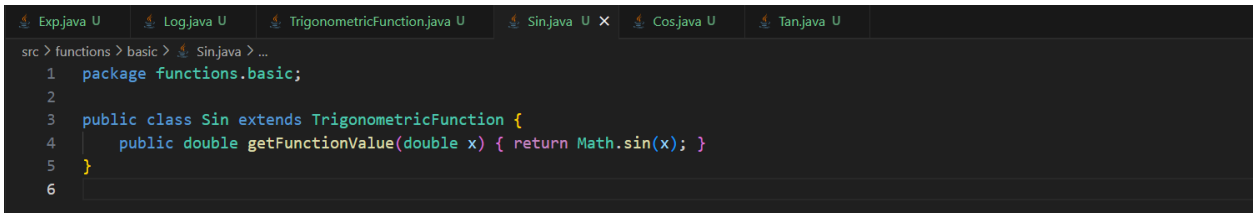
Я создал класс TrigonometricFunction, реализующий интерфейс Function и описывающий методы получения границ области определения.

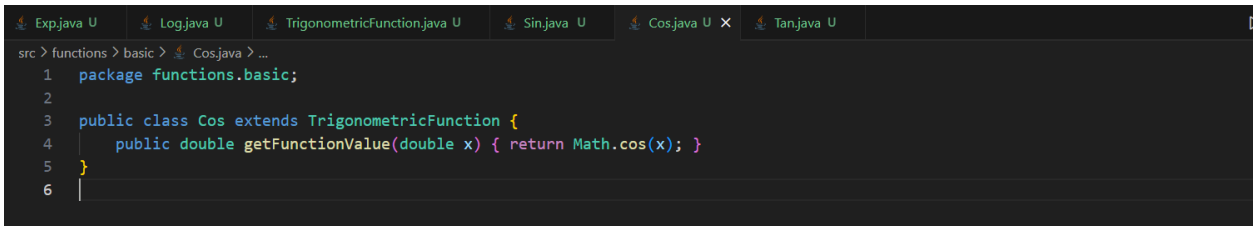
Я создайте наследующие от него публичные классы Sin, Cos и Tan, объекты которых вычисляют, соответственно, значения синуса, косинуса и тангенса. Для получения значений я воспользовался методами Math.sin(), Math.cos() и Math.tan().

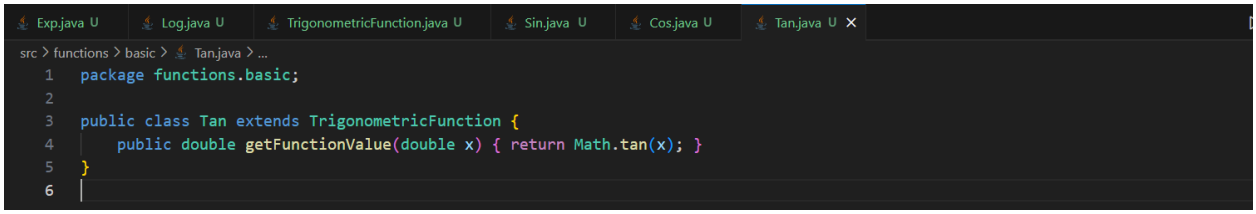
```java
package functions.basic;
import functions.Function;

public class Exp implements Function {
    public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }
    public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
    public double getFunctionValue(double x) { return Math.exp(x); }
}
```

```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(s: "Invalid base");
        this.base = base;
    }
    public double getLeftDomainBorder() { return 0; } // log(x) определён для x > 0
    public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
    public double getFunctionValue(double x) { return Math.log(x) / Math.log(this.base); }
}
```









Задание 4

Я создал пакет functions.meta, в нём описаны классы функций, позволяющие комбинировать функции.

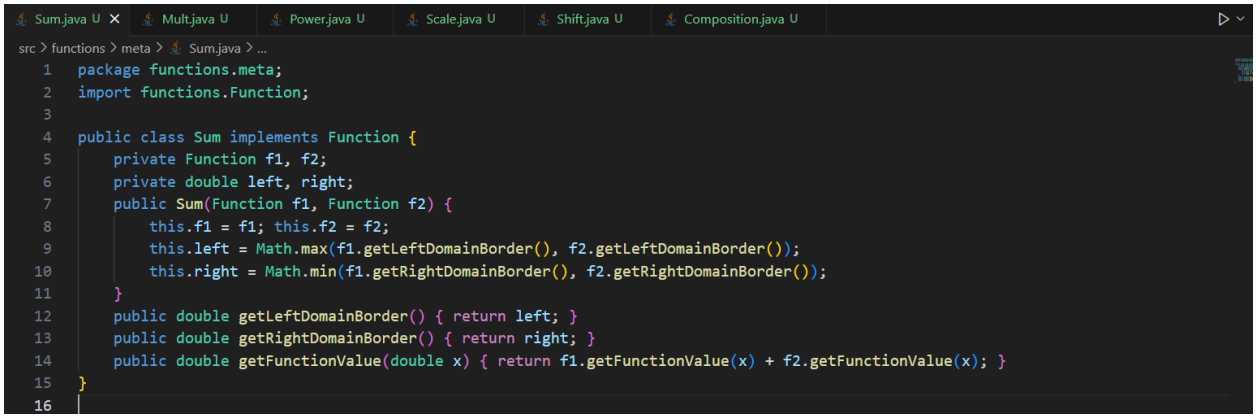
Я также создал класс Sum, объекты которого представляют собой функции, являющиеся суммой двух других функций. Класс реализовывает интерфейс Function. Конструктор класса получает ссылки типа Function на объекты суммируемых функций, а область определения функции получается как пересечение областей определения исходных функций.

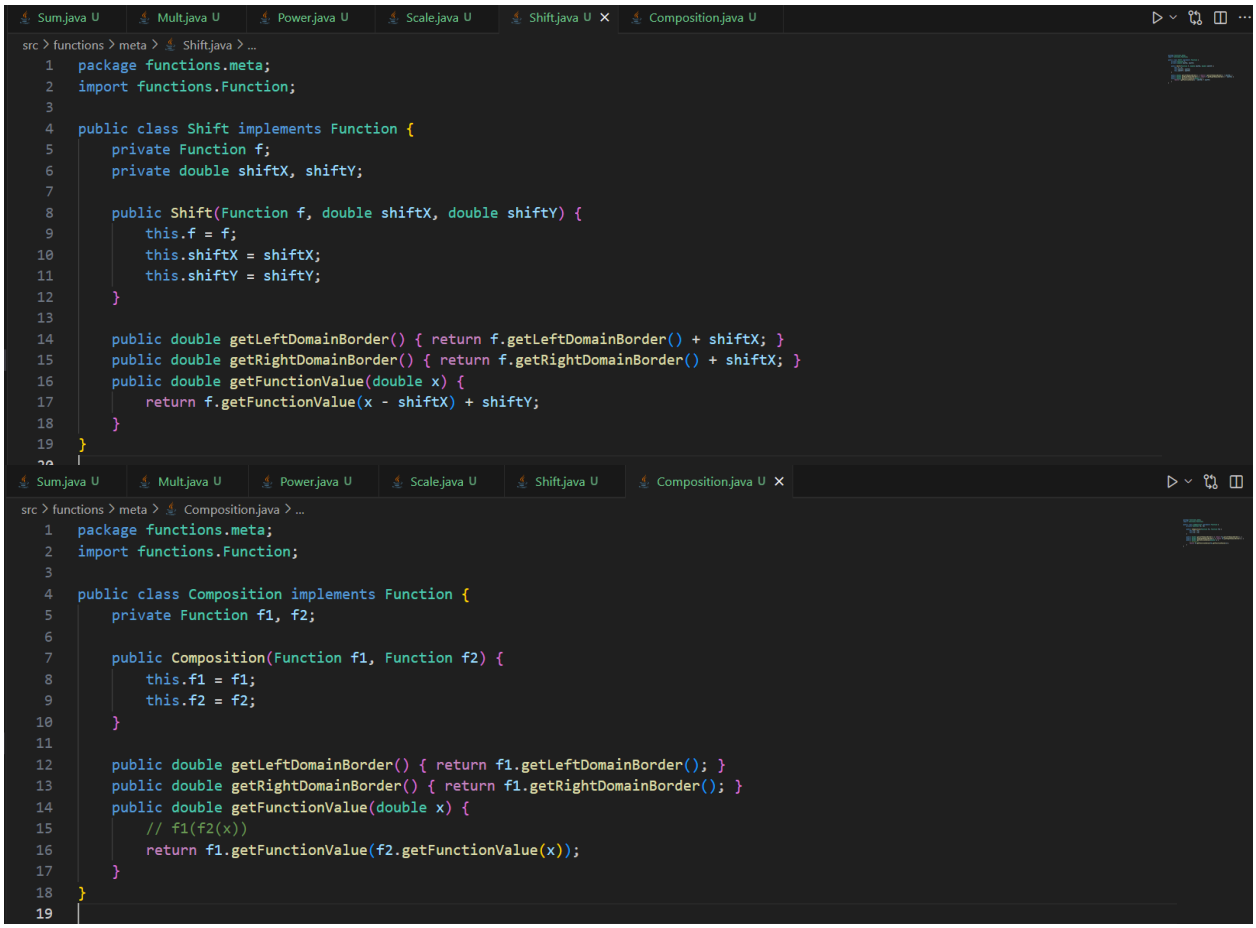
Аналогично, я создал класс Mult, объекты которого представляют собой функции, являющиеся произведением двух других функций.

Я создайте класс Power, объекты которого представляют собой функции, являющиеся степенью другой функции. Конструктор класса получает ссылку на объекты базовой функции и степень, в которую возводится её значения. Область определения функции можно считать совпадающей с областью определения исходной функции (хотя математически это не всегда так).

Я создал класс Scale, объекты которого описывают функции, полученные из исходных функций путём масштабирования вдоль осей координат. Конструктор класса получают 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.

Аналогично, я создал класс Shift, объекты которого описывают функции, полученные из исходных функций путём сдвига вдоль осей координат.
Также создайте 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).





Задание 5

В пакете functions я создал класс Functions, содержащий вспомогательные статические методы для работы с функциями. Я сделал так, что в программе вне этого класса нельзя 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 я воспользовался созданными ранее классами из пакета functions.meta.

Задание 6

В пакете functions я создал класс TabulatedFunctions, содержащий вспомогательные статические методы для работы с табулированными функциями. Я сделал так, что в программе вне этого класса нельзя создать его объект.
Я описал 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. В последующих работах в код будет добавлена возможность выбора класса для создания экземпляра.

Задание 7

В класс TabulatedFunctions я добавил следующие методы.
Метод вывода табулированной функции в байтовый поток public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) в указанный поток выводит значения, по которым потом можно 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, создаёт 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 восстановить табулированную функцию, а именно количество точек в ней и значения координат точек. Проще всего считать, что 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, создаёт и настраивает её объект и возвращает его из метода.
При написании методов в первых трёх случаях я воспользовался потоками-обёртками, облегчающими ввод и вывод данных в требующейся форме, а в четвёртом случае – классом StreamTokenizer.
При написании методов, считывающих табулированную функцию, я считал, что в потоке записаны правильные данные (проверку корректности вводимых данных делать не делал).

Поскольку методы ввода и чтения возвращают ссылку интерфейсного типа, можно возвращать объект любого из классов, реализующих этот интерфейс. В последующих работах в код будет добавлена возможность выбора класса для создания экземпляра.

Я подумал и обосновал, как следует в этих методах поступить с возникающим исключением IOException.
Я подумал и обосновал, следует ли закрывать потоки внутри этих методов.

Что я сделал и как обосновал:

В методах для ввода/вывода (таких как outputTabulatedFunction, inputTabulatedFunction и т.д.) исключение IOException не обрабатывается (через try-catch), а пробрасывается наверх (с помощью throws IOException).

Обоснование: Эти методы являются утилитарными, то есть "инструментами". Они не обладают информацией о том, насколько критична ошибка ввода-вывода (например, "файл не найден" или "диск переполнен").

- Если бы метод "проглотил" исключение, вызвавший его код (например, main) не узнал бы, что операция провалилась.
- Если бы метод сам обработал исключение, он не знал бы, как правильно это сделать (попробовать еще раз? сообщить пользователю? завершить программу?).

Пробрасывая исключение, метод передает ответственность за принятие решения тому, кто его вызвал. Вызывающий код (main) имеет полный контекст и может корректно отреагировать на ошибку.

Обоснование по закрытию потоков

Методы ввода/вывода не закрывают потоки (InputStream, OutputStream, Writer, Reader), которые они получают в качестве параметров.

Обоснование: В Java принято правило: “Кто создал (открыл) поток, тот его и закрывает”.

Эти методы не создают потоки, а получают их “в пользование” от другого кода (в нашем случае — от main, который создает new FileWriter(...)).

- Если бы наш метод-инструмент закрыл поток, то вызывающий код (main) не смог бы, например, записать в этот же файл что-то еще после завершения нашего метода.
- Ответственность за закрытие потока (.close()) лежит на том коде, который его открыл (main), и который точно знает, когда этот поток больше не понадобится.

Задание 8

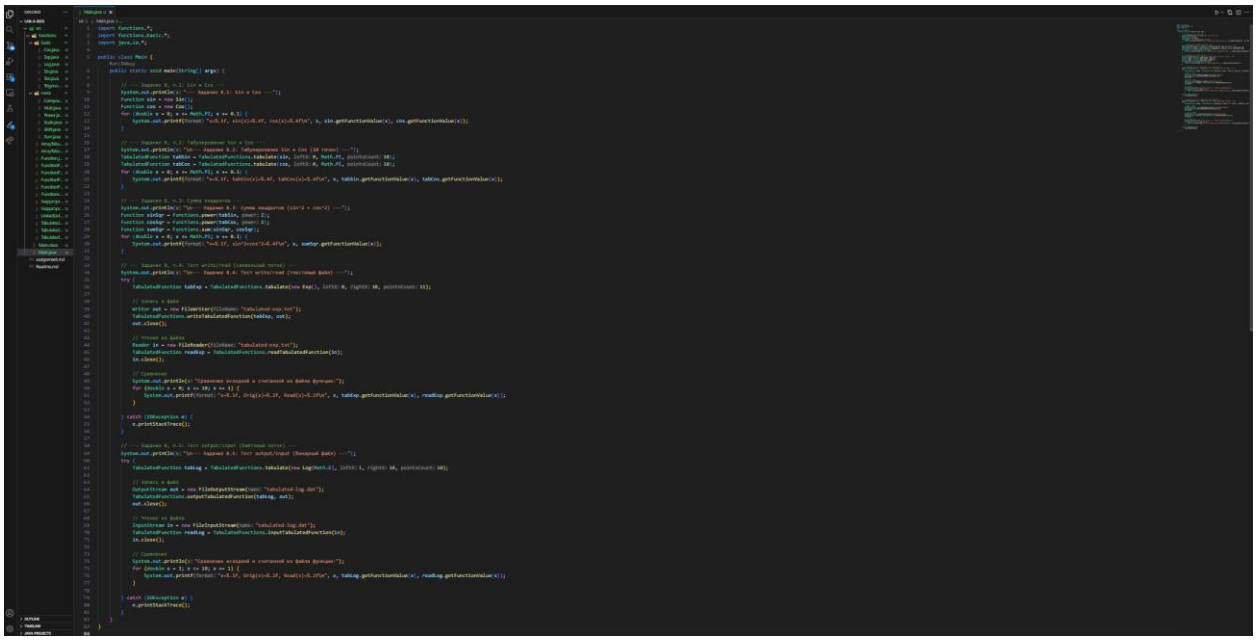
Я проверил работу написанных классов.

Я создал по одному объекту классов Sin и Cos, вывел в консоль значения этих функций на отрезке от 0 до π с шагом 0,1.
С помощью метода TabulatedFunctions.tabulate() я создал табулированные аналоги этих функций на отрезке от 0 до π с 10 точками. Вывел в консоль значения этих функций на отрезке от 0 до π с шагом 0,1 и сравнил со значениями исходных функций.
С помощью методов класса Functions я создал объект функции, являющейся суммой квадратов табулированных аналогов синуса и косинуса. Вывел в консоль значения этой функций на отрезке от 0 до π с шагом 0,1. Я попровал изменять количество точек в табулированных аналогах и исследовал, как при этом изменяется результирующая функция.
С помощью метода TabulatedFunctions.tabulate() я создал табулированный аналог экспоненты на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.writeTabulatedFunction() вывел его в файл. Далее с помощью метода TabulatedFunctions.readTabulatedFunction() считал табулированную функцию из этого файла. Вывел и сравнил значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
С помощью метода TabulatedFunctions.tabulate() я создал табулированный аналог логарифма по натуральному основанию на отрезке от 0 до 10 с 11 точками. С помощью метода TabulatedFunctions.outputTabulatedFunction() вывел его в файл (имя файла отличается от предыдущего случая). Далее с помощью метода TabulatedFunctions.inputTabulatedFunction() я считал табулированную функцию из этого файла. Вывел и сравнил значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
Я изучите содержимое всех получаемых файлов и сделал выводы о преимуществах и недостатках каждого из форматов хранения.

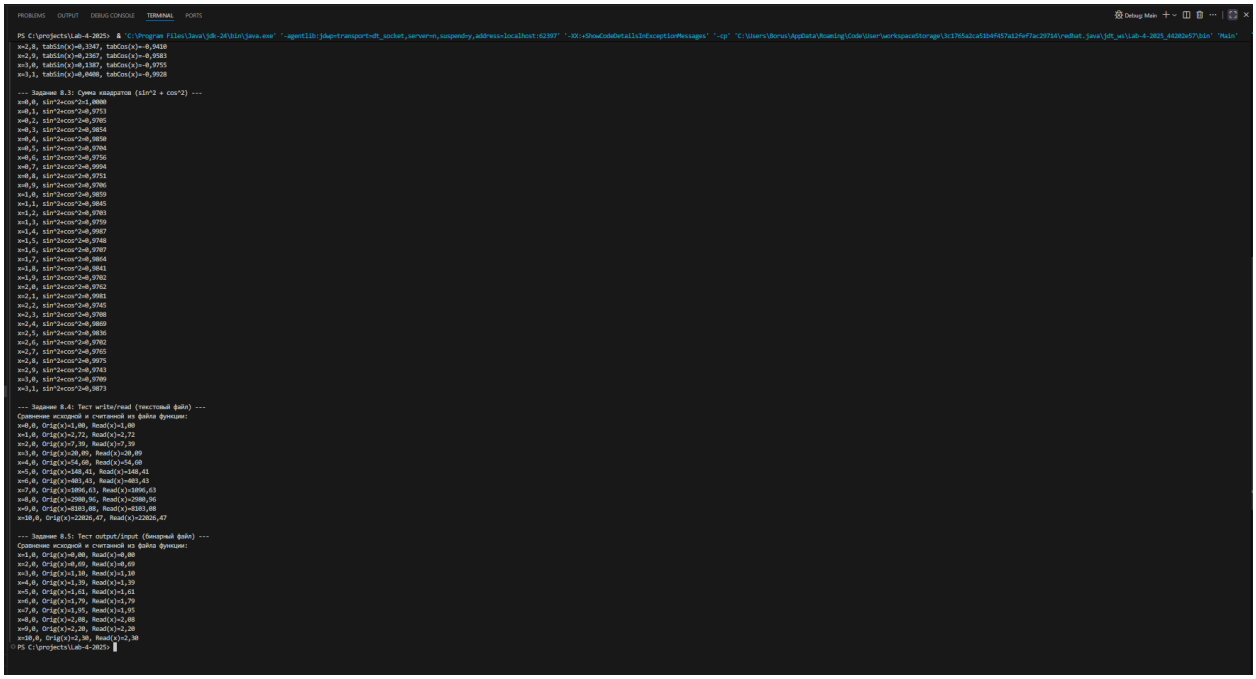
Выводы:

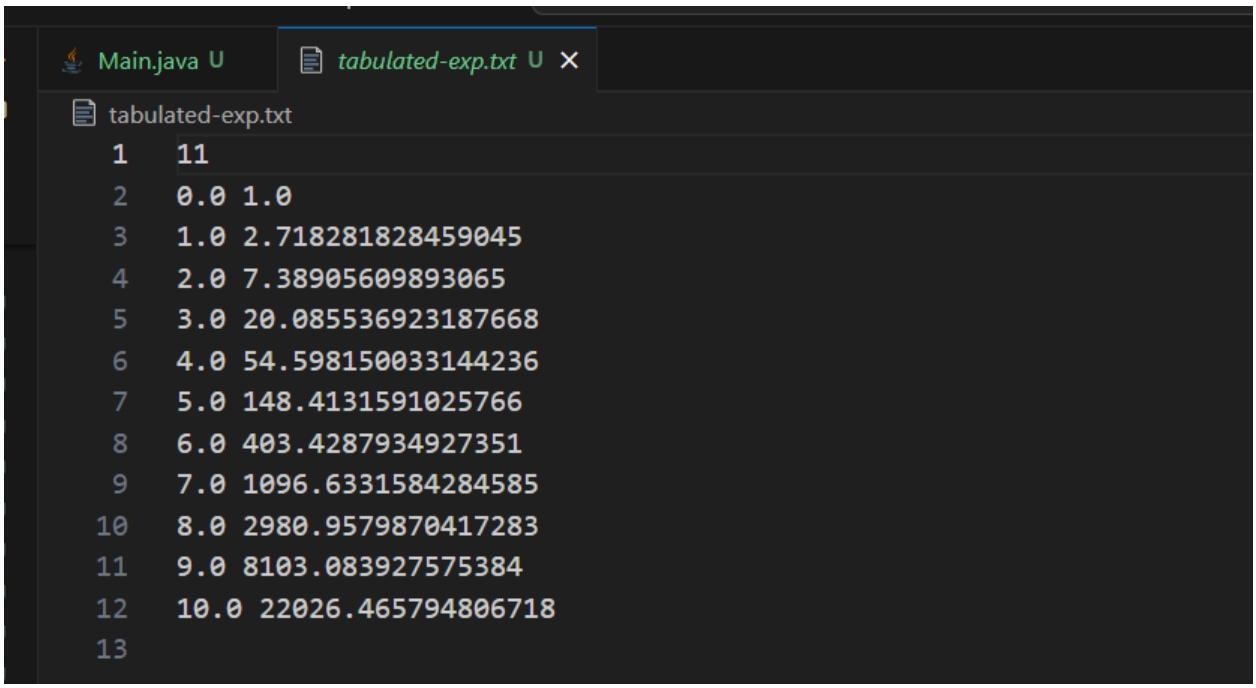
При выполнении работы были созданы два файла для одной и той же функции: tabulated-exp.txt (символьный поток) и tabulated-log.dat (байтовый поток).

1. Символьный поток (write.../read..., файл .txt)
    - Преимущества: Файл является текстовым (tabulated-exp.txt), его можно открыть в любом редакторе (Блокнот, VS Code) и прочитать. Он понятен человеку и его легко можно исправить вручную.
    - Недостатки: Очень неэффективный. Файл занимает много места на диске, так как числа (например, 3.14159) хранятся как последовательность символов, а не как 8 байт. Процесс записи и чтения медленный, так как требует постоянной конвертации чисел в строки и обратно.
2. Байтовый поток (output.../input..., файл .dat)
    - Преимущества: Очень эффективный. Файл (tabulated-log.dat) занимает значительно меньше места, так как числа double записываются напрямую в их 8-байтовом бинарном представлении. Чтение и запись происходят очень быстро, так как отсутствует этап конвертации.
    - Недостатки: Файл абсолютно нечитаем для человека. Его нельзя открыть в Блокноте и понять, что в нём содержится.




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Debug: Main
PS C:\projects\Lab-4-2025> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:62397' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\3c1765a2ca51b4f457a12fef7ac29714\redhat.java\jdt_ws\Lab-4-2025_44202e57\bin' 'Ma
--- Задание 8.1: Sin и Cos ---
x=0,0, sin(x)=0,0000, cos(x)=1,0000
x=0,1, sin(x)=0,0998, cos(x)=0,9950
x=0,2, sin(x)=0,1987, cos(x)=0,9801
x=0,3, sin(x)=0,2955, cos(x)=0,9553
x=0,4, sin(x)=0,3894, cos(x)=0,9211
x=0,5, sin(x)=0,4794, cos(x)=0,8776
x=0,6, sin(x)=0,5646, cos(x)=0,8253
x=0,7, sin(x)=0,6442, cos(x)=0,7648
x=0,8, sin(x)=0,7174, cos(x)=0,6967
x=0,9, sin(x)=0,7833, cos(x)=0,6216
x=1,0, sin(x)=0,8415, cos(x)=0,5403
x=1,1, sin(x)=0,8912, cos(x)=0,4536
x=1,2, sin(x)=0,9320, cos(x)=0,3624
x=1,3, sin(x)=0,9636, cos(x)=0,2675
x=1,4, sin(x)=0,9854, cos(x)=0,1700
x=1,5, sin(x)=0,9975, cos(x)=0,0707
x=1,6, sin(x)=0,9996, cos(x)=-0,0292
x=1,7, sin(x)=0,9917, cos(x)=-0,1288
x=1,8, sin(x)=0,9738, cos(x)=-0,2272
x=1,9, sin(x)=0,9463, cos(x)=-0,3233
x=2,0, sin(x)=0,9093, cos(x)=-0,4161
x=2,1, sin(x)=0,8632, cos(x)=-0,5048
x=2,2, sin(x)=0,8085, cos(x)=-0,5885
x=2,3, sin(x)=0,7457, cos(x)=-0,6663
x=2,4, sin(x)=0,6755, cos(x)=-0,7374
x=2,5, sin(x)=0,5985, cos(x)=-0,8011
x=2,6, sin(x)=0,5155, cos(x)=-0,8569
x=2,7, sin(x)=0,4274, cos(x)=-0,9041
x=2,8, sin(x)=0,3350, cos(x)=-0,9422
x=2,9, sin(x)=0,2392, cos(x)=-0,9710
x=3,0, sin(x)=0,1411, cos(x)=-0,9900
x=3,1, sin(x)=0,0416, cos(x)=-0,9991
--- Задание 8.2: Табулирование Sin и Cos (10 точек) ---
x=0,0, tabSin(x)=0,0000, tabCos(x)=1,0000
x=0,1, tabSin(x)=0,0980, tabCos(x)=0,9827
x=0,2, tabSin(x)=0,1968, tabCos(x)=0,9654
x=0,3, tabSin(x)=0,2939, tabCos(x)=0,9482
x=0,4, tabSin(x)=0,3859, tabCos(x)=0,9144
x=0,5, tabSin(x)=0,4721, tabCos(x)=0,8646
x=0,6, tabSin(x)=0,5582, tabCos(x)=0,8149
x=0,7, tabSin(x)=0,6440, tabCos(x)=0,7646
x=0,8, tabSin(x)=0,7079, tabCos(x)=0,6884
x=0,9, tabSin(x)=0,7710, tabCos(x)=0,6122
x=1,0, tabSin(x)=0,8358, tabCos(x)=0,5368
x=1,1, tabSin(x)=0,8848, tabCos(x)=0,4586
x=1,2, tabSin(x)=0,9188, tabCos(x)=0,3571
x=1,3, tabSin(x)=0,9521, tabCos(x)=0,2636
x=1,4, tabSin(x)=0,9848, tabCos(x)=0,1699
x=1,5, tabSin(x)=0,9848, tabCos(x)=0,0704
x=1,6, tabSin(x)=0,9848, tabCos(x)=-0,0291
x=1,7, tabSin(x)=0,9848, tabCos(x)=-0,1285
x=1,8, tabSin(x)=0,9652, tabCos(x)=-0,2248
x=1,9, tabSin(x)=0,9322, tabCos(x)=-0,3183
x=2,0, tabSin(x)=0,8981, tabCos(x)=-0,4117
x=2,1, tabSin(x)=0,8624, tabCos(x)=-0,5043
x=2,2, tabSin(x)=0,7985, tabCos(x)=-0,5805
x=2,3, tabSin(x)=0,7345, tabCos(x)=-0,6567
x=2,4, tabSin(x)=0,6706, tabCos(x)=-0,7329
x=2,5, tabSin(x)=0,5941, tabCos(x)=-0,7942
x=2,6, tabSin(x)=0,5079, tabCos(x)=-0,8439
x=2,7, tabSin(x)=0,4217, tabCos(x)=-0,8937
x=2,8, tabSin(x)=0,3347, tabCos(x)=-0,9410
x=2,9, tabSin(x)=0,2367, tabCos(x)=-0,9583
x=3,0, tabSin(x)=0,1387, tabCos(x)=-0,9755
x=3,1, tabSin(x)=0,0408, tabCos(x)=-0,9928
--- Задание 8.3: Сумма квадратов (sin^2 + cos^2) ---
x=0,0, sin^2+cos^2=1,0000


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Debug: Main
PS C:\projects\Lab-4-2025> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:62397' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\3c1765a2ca51b4f457a12fef7ac29714\redhat.java\jdt_ws\Lab-4-2025_44202e57\bin' 'Main'
x=2,8, tabSin(x)=0,3347, tabCos(x)=-0,9410
x=2,9, tabSin(x)=0,2367, tabCos(x)=-0,9583
x=3,0, tabSin(x)=0,1387, tabCos(x)=-0,9755
x=3,1, tabSin(x)=0,0408, tabCos(x)=-0,9928
--- Задание 8.3: Сумма квадратов (sin^2 + cos^2) ---
x=0,0, sin^2+cos^2=1,0000
x=0,1, sin^2+cos^2=0,9753
x=0,2, sin^2+cos^2=0,9705
x=0,3, sin^2+cos^2=0,9854
x=0,4, sin^2+cos^2=0,9858
x=0,5, sin^2+cos^2=0,9704
x=0,6, sin^2+cos^2=0,9756
x=0,7, sin^2+cos^2=0,9904
x=0,8, sin^2+cos^2=0,9751
x=0,9, sin^2+cos^2=0,9706
x=1,0, sin^2+cos^2=0,9859
x=1,1, sin^2+cos^2=0,9845
x=1,2, sin^2+cos^2=0,9703
x=1,3, sin^2+cos^2=0,9759
x=1,4, sin^2+cos^2=0,9987
x=1,5, sin^2+cos^2=0,9748
x=1,6, sin^2+cos^2=0,9707
x=1,7, sin^2+cos^2=0,9864
x=1,8, sin^2+cos^2=0,9841
x=1,9, sin^2+cos^2=0,9702
x=2,0, sin^2+cos^2=0,9762
x=2,1, sin^2+cos^2=0,9981
x=2,2, sin^2+cos^2=0,9745
x=2,3, sin^2+cos^2=0,9708
x=2,4, sin^2+cos^2=0,9869
x=2,5, sin^2+cos^2=0,9836
x=2,6, sin^2+cos^2=0,9702
x=2,7, sin^2+cos^2=0,9765
x=2,8, sin^2+cos^2=0,9975
x=2,9, sin^2+cos^2=0,9743
x=3,0, sin^2+cos^2=0,9709
x=3,1, sin^2+cos^2=0,9873
--- Задание 8.4: Тест write/read (текстовый файл) ---
Сравнение исходной и считанной из файла функции:
x=0,0, Orig(x)=1,00, Read(x)=1,00
x=1,0, Orig(x)=2,72, Read(x)=2,72
x=2,0, Orig(x)=7,39, Read(x)=7,39
x=3,0, Orig(x)=20,09, Read(x)=20,09
x=4,0, Orig(x)=54,60, Read(x)=54,60
x=5,0, Orig(x)=148,41, Read(x)=148,41
x=6,0, Orig(x)=403,43, Read(x)=403,43
x=7,0, Orig(x)=1096,63, Read(x)=1096,63
x=8,0, Orig(x)=2980,96, Read(x)=2980,96
x=9,0, Orig(x)=8103,08, Read(x)=8103,08
x=10,0, Orig(x)=22026,47, Read(x)=22026,47
--- Задание 8.5: Тест output/input (бинарный файл) ---
Сравнение исходной и считанной из файла функции:
x=1,0, Orig(x)=0,00, Read(x)=0,00
x=2,0, Orig(x)=0,69, Read(x)=0,69
x=3,0, Orig(x)=1,10, Read(x)=1,10
x=4,0, Orig(x)=1,39, Read(x)=1,39
x=5,0, Orig(x)=1,61, Read(x)=1,61
x=6,0, Orig(x)=1,79, Read(x)=1,79
x=7,0, Orig(x)=1,95, Read(x)=1,95
x=8,0, Orig(x)=2,08, Read(x)=2,08
x=9,0, Orig(x)=2,20, Read(x)=2,20
x=10,0, Orig(x)=2,30, Read(x)=2,30
PS C:\projects\Lab-4-2025>



Задание 9

Я сделайте так, чтобы объекты всех классов, реализующих интерфейс TabulatedFunction, были сериализуемыми.
Для этого я рассмотрел два случая:

1. с использованием интерфейса java.io.Serializable
2. с использованием интерфейса java.io.Externalizable

Я проверьте работу написанных классов. С помощью метода TabulatedFunctions.tabulate() и метода класса Functions я создал табулированный аналог логарифма по натуральному основанию, взятого от экспоненты на отрезке от 0 до 10 с 11 точками. Я сериализовал полученный объект в файл (имя файла отличается от предыдущих случаев). Далее я десериализовал табулированную функцию из этого файла. Я вывел значения исходной и считанной функции на отрезке от 0 до 10 с шагом 1.
Я изучите содержимое файлов, получаемых при реализации механизма сериализации с использованием интерфейса java.io.Serializable и при реализации механизма сериализации с использованием интерфейса java.io.Externalizable. Я сделайте выводы о преимуществах и недостатках каждого из способов.

Выводы:

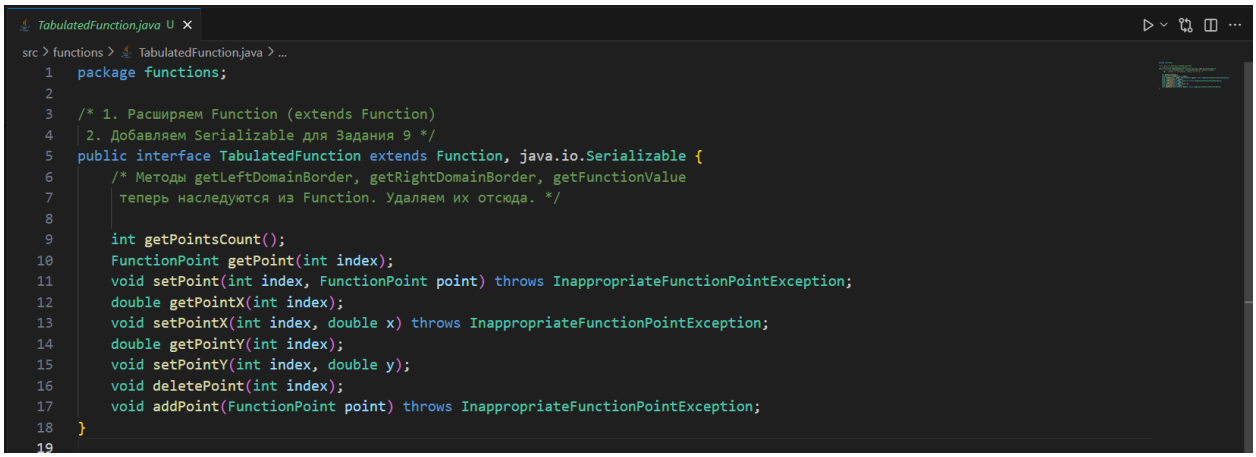
В работе было реализовано два подхода к сериализации: Serializable (для ArrayTabulatedFunction) и Externalizable (для LinkedListTabulatedFunction).

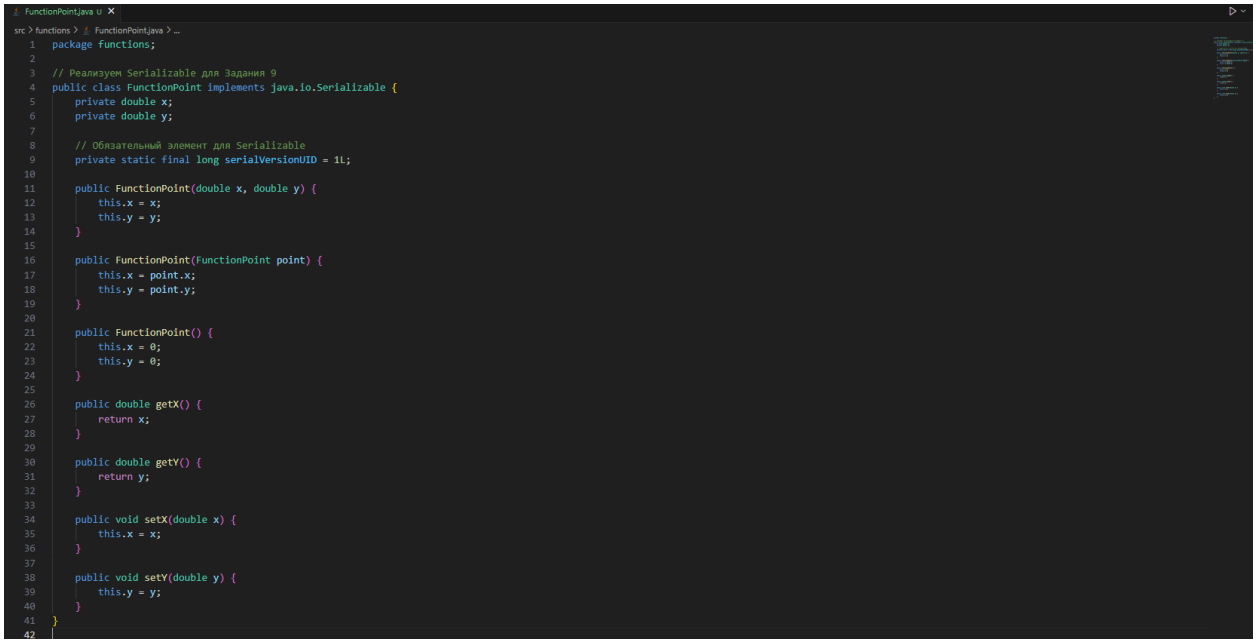
1. Serializable (стандартный механизм Java)
    - Преимущества: Невероятно прост в реализации. Достаточно добавить implements Serializable и serialVersionUID, и Java сделает всё остальное сама.
    - Недостатки: Файл (serialized-function.ser) нечитаем для человека и часто занимает *больше* места, чем даже байтовый поток, так как Java сохраняет в

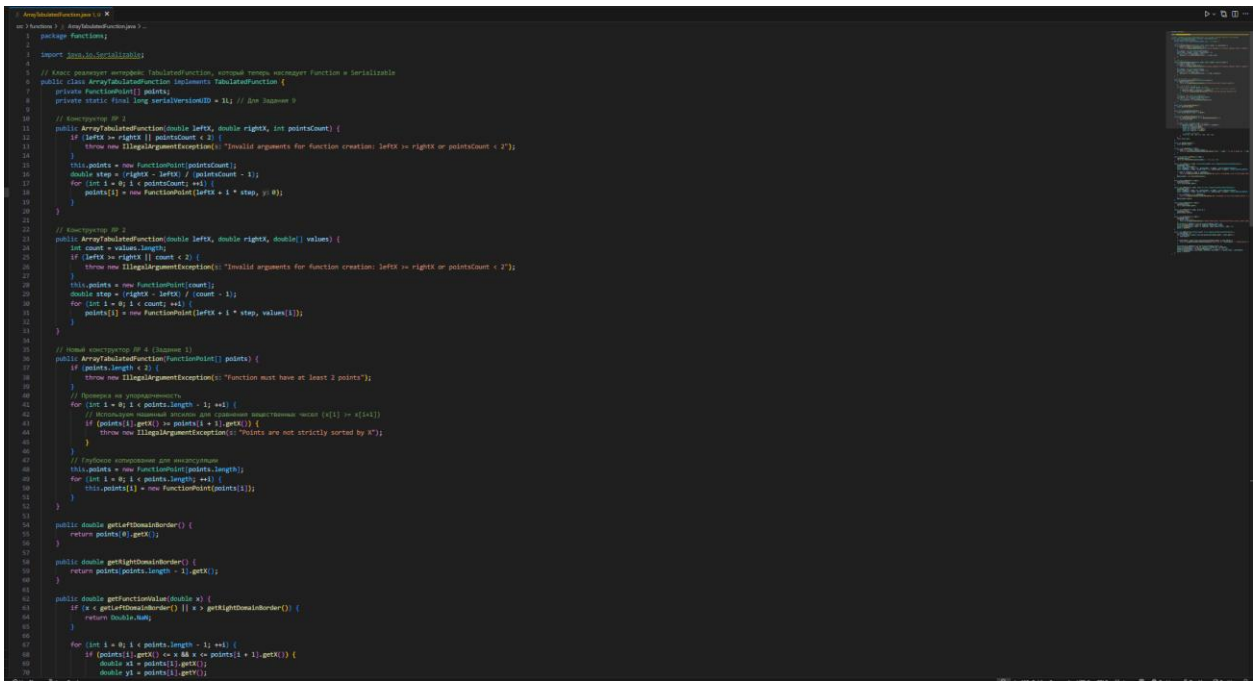
него много служебной информации (полное имя класса, имена полей, иерархию наследования). Механизм "хрупкий" — если изменить класс (например, добавить поле), то прочитать старые сохраненные файлы уже не получится.

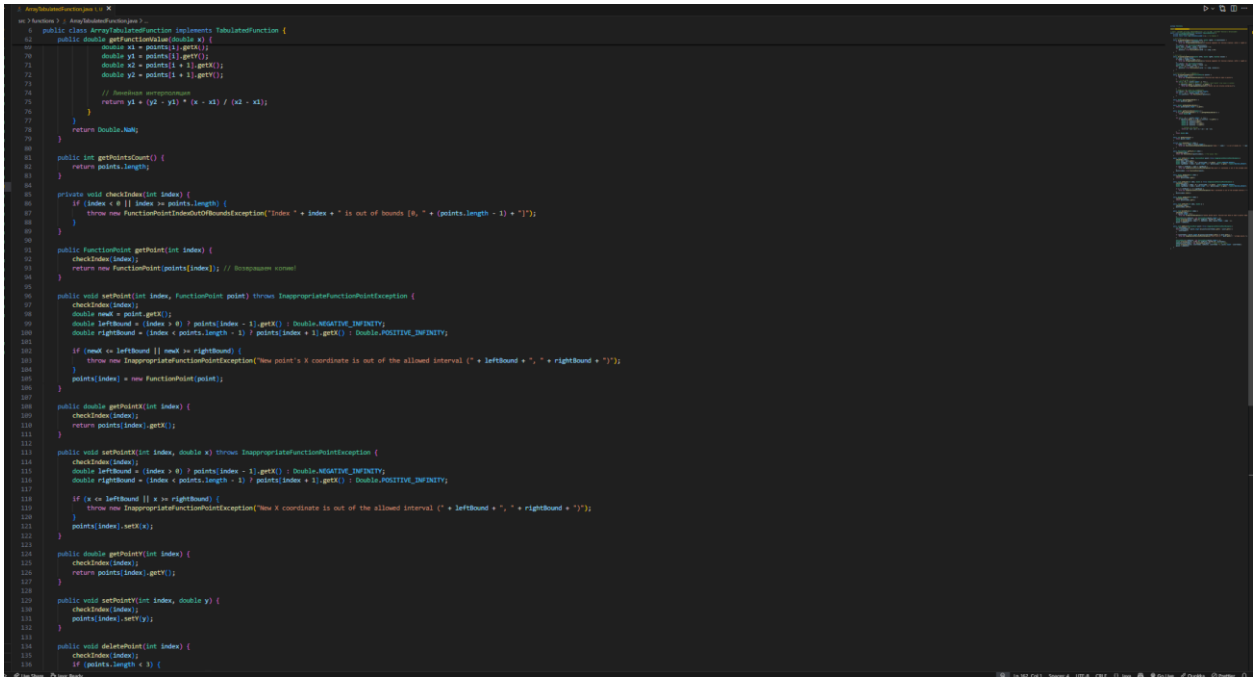
2. Externalizable (ручной контроль)

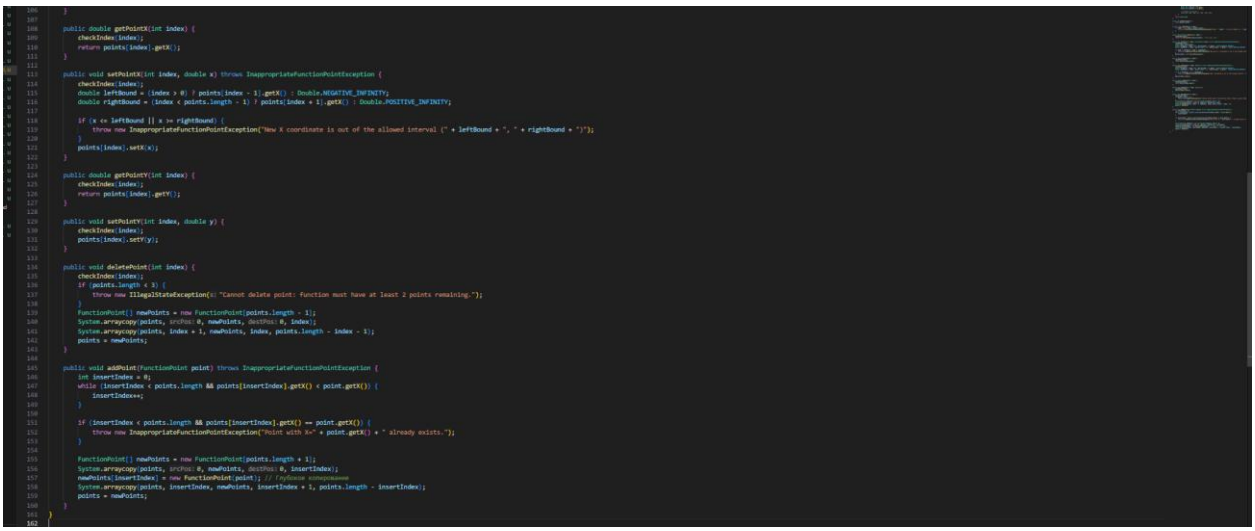
- Преимущества: Обеспечивает полный контроль над процессом сериализации. Файл может быть более компактным, чем при использовании Serializable, поскольку мы сами решаем, какие поля сохранять. Это делает механизм устойчивым к изменениям в классе.
- Недостатки: Значительно сложнее в реализации. Требуется вручную написать методы writeExternal и readExternal, а также обязательно создать публичный конструктор без аргументов. Легко допустить ошибку (например, перепутать порядок чтения полей).



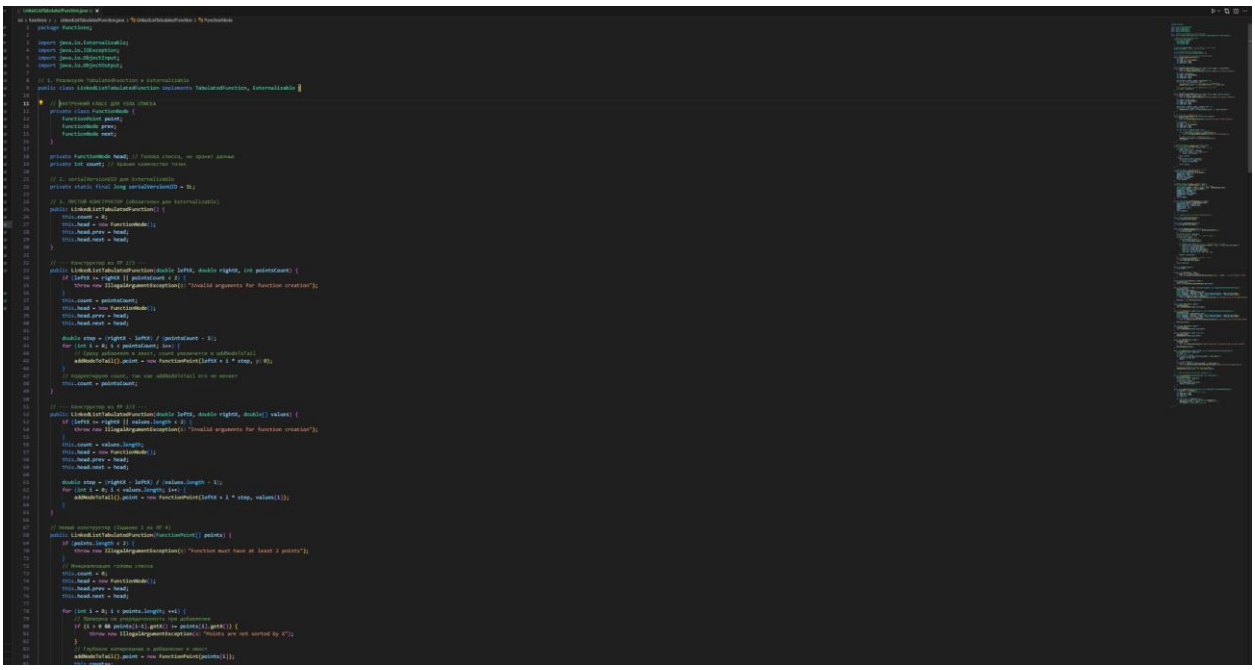




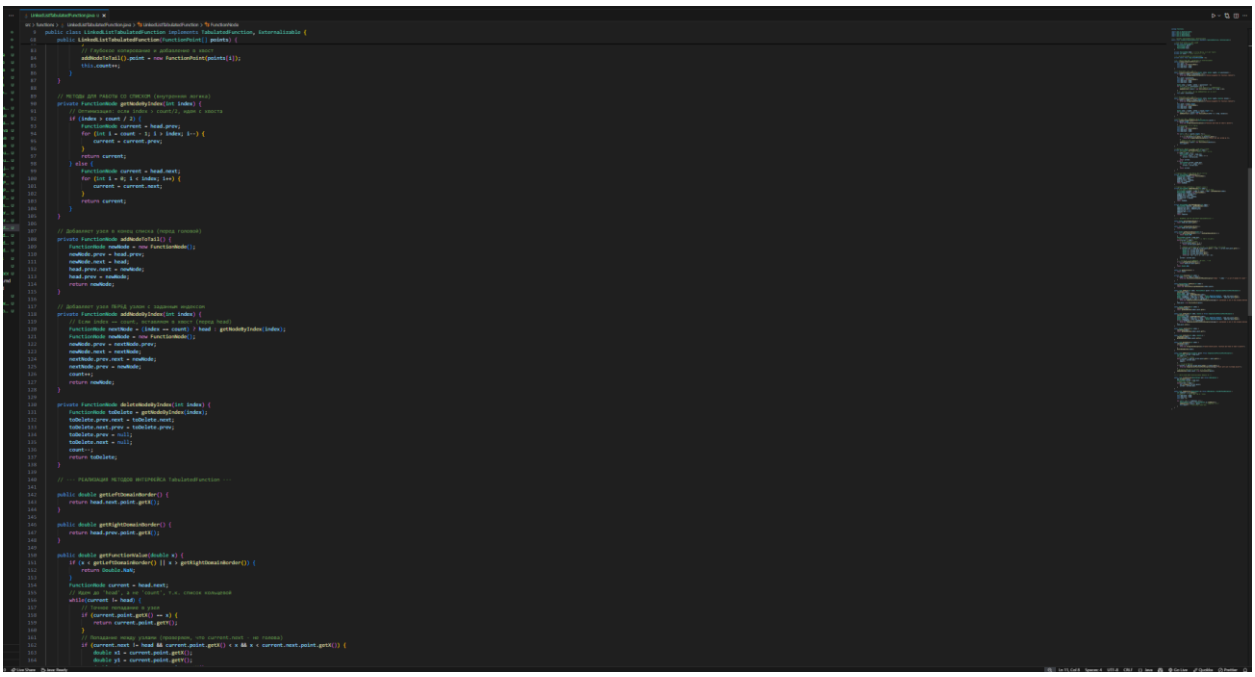


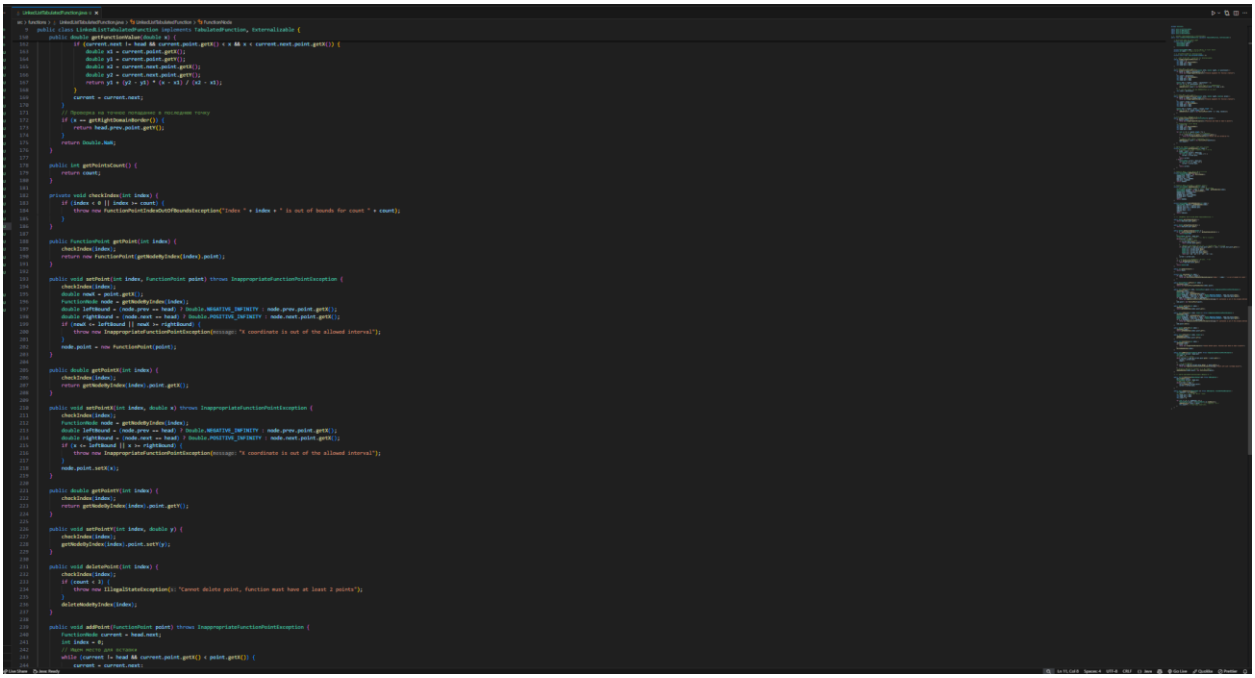


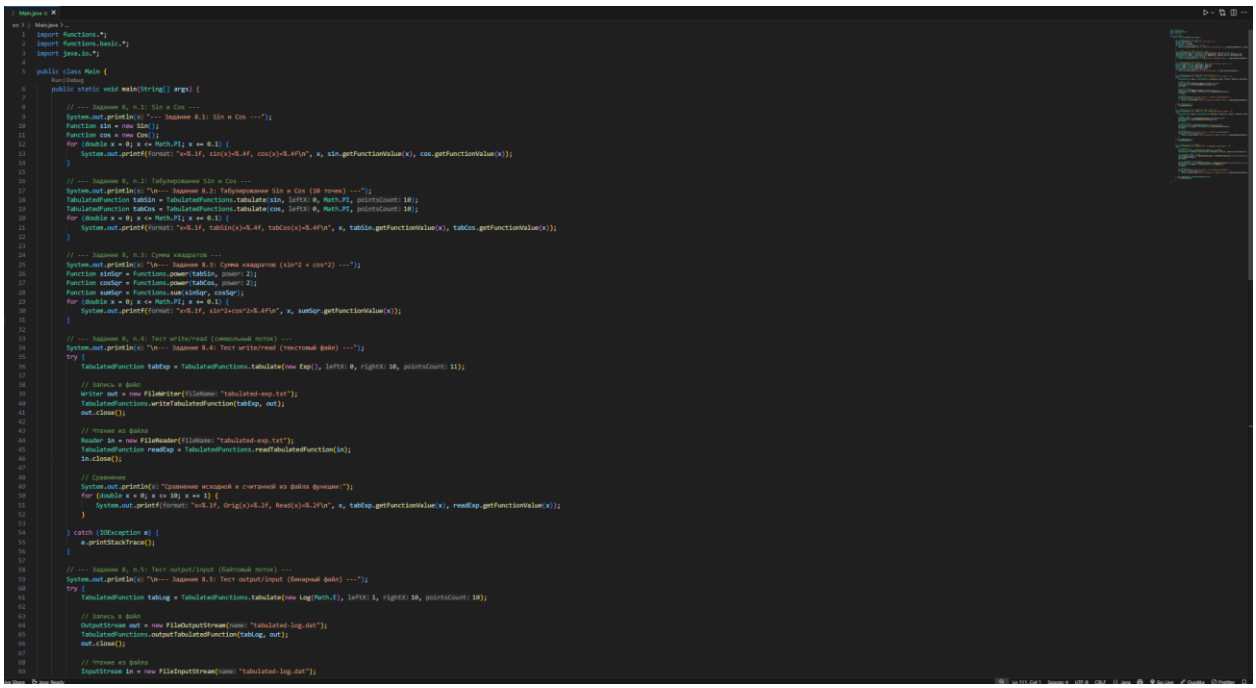
```
    }

    public double getPointX(int index) {
        checkIndex(index);
        return points[index].getX();
    }

    public void setPointX(int index, double x) throws InappropriateFunctionPointException {
        checkIndex(index);
        double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
        double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;

        if (x <= leftBound || x >= rightBound) {
            throw new InappropriateFunctionPointException("New X coordinate is out of the allowed interval (" + leftBound + ", " + rightBound + ")");
        }
        points[index].setX(x);
    }

    public double getPointY(int index) {
        checkIndex(index);
        return points[index].getY();
    }

    public void setPointY(int index, double y) {
        checkIndex(index);
        points[index].setY(y);
    }

    public void deletePoint(int index) {
        checkIndex(index);
        if (points.length < 3) {
            throw new IllegalStateException(s: "Cannot delete point: function must have at least 2 points remaining.");
        }
        FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, index);
        System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
        points = newPoints;
    }

    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
        int insertIndex = 0;
        while (insertIndex < points.length && points[insertIndex].getX() < point.getX()) {
            insertIndex++;
        }

        if (insertIndex < points.length && points[insertIndex].getX() == point.getX()) {
            throw new InappropriateFunctionPointException("Point with X=" + point.getX() + " already exists.");
        }

        FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
        System.arraycopy(points, srcPos: 0, newPoints, destPos: 0, insertIndex);
        newPoints[insertIndex] = new FunctionPoint(point); // Глубокое копирование
        System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
        points = newPoints;
    }
}
```



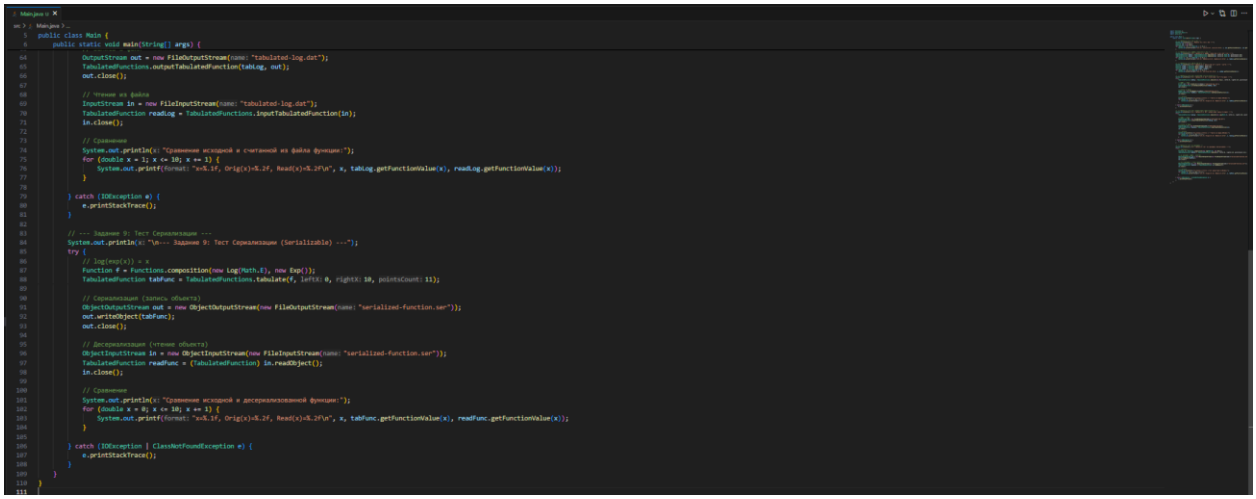
```
package functions;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

// 1. Реализуем TabulatedFunction и Externalizable
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable {

    // Внутренний класс для узла списка
    private class FunctionNode {
        FunctionPoint point;
        FunctionNode prev;
        FunctionNode next;
    }

    private FunctionNode head; // Голова списка, не хранит данных
    private int count; // Кэшированное число точек

    // 2. serialVersionUID для Externalizable
    private static final long serialVersionUID = 1L;

    // 3. ПУСТОЙ КОНСТРУКТОР (обязателен для Externalizable)
    public LinkedListTabulatedFunction() {
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;
    }

    //  Конструктор из ЛР 2/3 
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX || pointsCount < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.count = pointsCount;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            // Узлы добавятся в конец, поэтому используется addNodeToTail
            addNodeToTail().point = new FunctionPoint(leftX + i * step, y: 0);
        }
        // Корректируем count, так как addNodeToTail его не меняет
        this.count = pointsCount;
    }

    //  Конструктор из ЛР 2/3 
    public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
        if (leftX >= rightX || values.length < 2) {
            throw new IllegalArgumentException(s: "Invalid arguments for function creation");
        }
        this.count = values.length;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        double step = (rightX - leftX) / (values.length - 1);
        for (int i = 0; i < values.length; i++) {
            addNodeToTail().point = new FunctionPoint(leftX + i * step, values[i]);
        }
    }

    // Новый конструктор (задание 3 из ЛР 4)
    public LinkedListTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException(s: "Function must have at least 2 points");
        }
        // Инициализация пустого списка
        this.count = 0;
        this.head = new FunctionNode();
        this.head.prev = head;
        this.head.next = head;

        for (int i = 0; i < points.length; ++i) {
            // Проверка на упорядоченность при добавлении
            if (i > 0 && points[i-1].getX() >= points[i].getX()) {
                throw new IllegalArgumentException(s: "Points are not sorted by X");
            }
            // Глубокое копирование и добавление в конец
            addNodeToTail().point = new FunctionPoint(points[i]);
        }
```





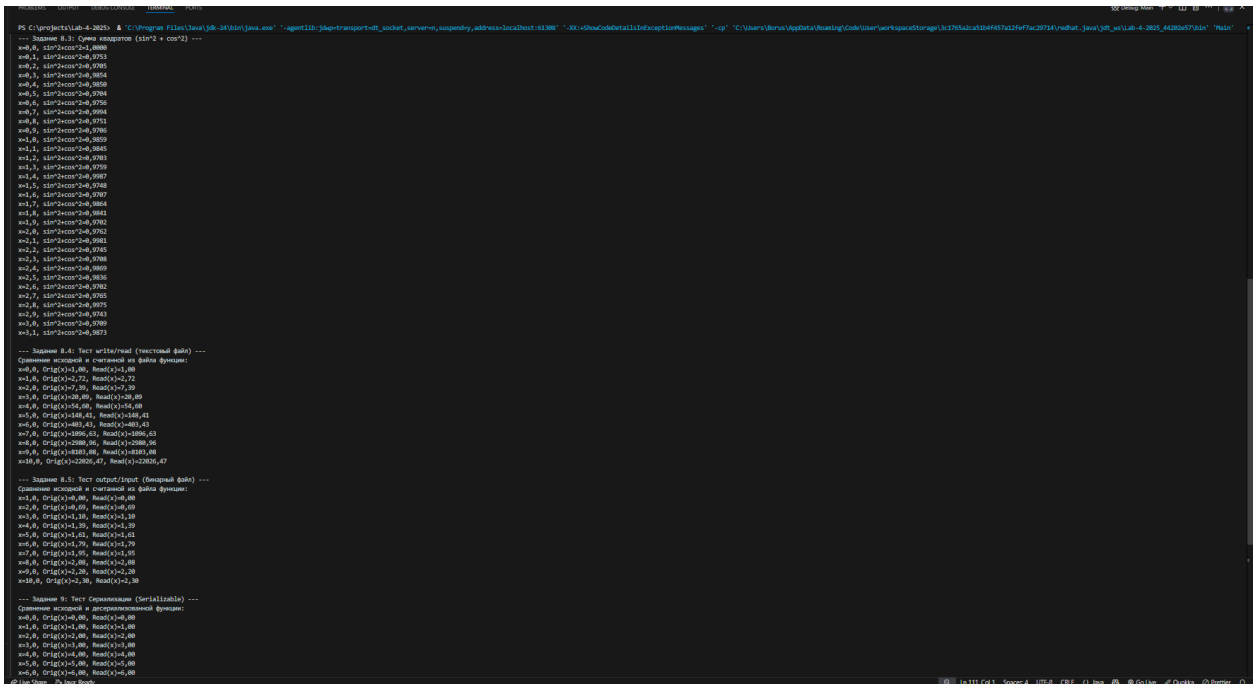


```java
import functions.*;
import functions.basic.*;
import java.io.*;

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

        // --- Задание 8, п.1: Sin и Cos ---
        System.out.println("\n--- Задание 8.1: Sin и Cos ---");
        Function sin = new Sin();
        Function cos = new Cos();
        for (double x = 0; x <= Math.PI; x += 0.1) {
            System.out.printf("x=%.1f, sin(x)=%.4f, cos(x)=%.4f\n", x, sin.getFunctionValue(x), cos.getFunctionValue(x));
        }

        // --- Задание 8, п.2: Табулирование Sin и Cos ---
        System.out.println("\n--- Задание 8.2: Табулирование Sin и Cos (10 точек) ---");
        TabulatedFunction tabSin = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
        TabulatedFunction tabCos = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);
        for (double x = 0; x <= Math.PI; x += 0.1) {
            System.out.printf("x=%.1f, tabSin(x)=%.4f, tabCos(x)=%.4f\n", x, tabSin.getFunctionValue(x), tabCos.getFunctionValue(x));
        }

        // --- Задание 8, п.3: Сумма квадратов ---
        System.out.println("\n--- Задание 8.3: Сумма квадратов (sin^2 + cos^2) ---");
        Function sinSqr = Functions.power(tabSin, power: 2);
        Function cosSqr = Functions.power(tabCos, power: 2);
        Function sumSqr = Functions.sum(sinSqr, cosSqr);
        for (double x = 0; x <= Math.PI; x += 0.1) {
            System.out.printf("x=%.1f, sin^2+cos^2=%.4f\n", x, sumSqr.getFunctionValue(x));
        }

        // --- Задание 8, п.4: Тест write/read (символьный поток) ---
        System.out.println("\n--- Задание 8.4: Тест write/read (текстовый файл) ---");
        try {
            TabulatedFunction tabExp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);

            // Запись в файл
            Writer out = new FileWriter(fileName: "tabulated-exp.txt");
            TabulatedFunctions.writeTabulatedFunction(tabExp, out);
            out.close();

            // Чтение из файла
            Reader in = new FileReader(fileName: "tabulated-exp.txt");
            TabulatedFunction readExp = TabulatedFunctions.readTabulatedFunction(in);
            in.close();

            // Сравнение
            System.out.println("Сравнение исходной и считанной из файла функции:");
            for (double x = 0; x <= 10; x += 1) {
                System.out.printf("x=%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, tabExp.getFunctionValue(x), readExp.getFunctionValue(x));
            }

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

        // --- Задание 8, п.5: Тест output/input (байтовый поток) ---
        System.out.println("\n--- Задание 8.5: Тест output/input (бинарный файл) ---");
        try {
            TabulatedFunction tabLog = TabulatedFunctions.tabulate(new Log(Math.E), leftX: 1, rightX: 10, pointsCount: 10);

            // Запись в файл
            OutputStream out = new FileOutputStream(name: "tabulated-log.dat");
            TabulatedFunctions.outputTabulatedFunction(tabLog, out);
            out.close();

            // Чтение из файла
            InputStream in = new FileInputStream(name: "tabulated-log.dat");
```

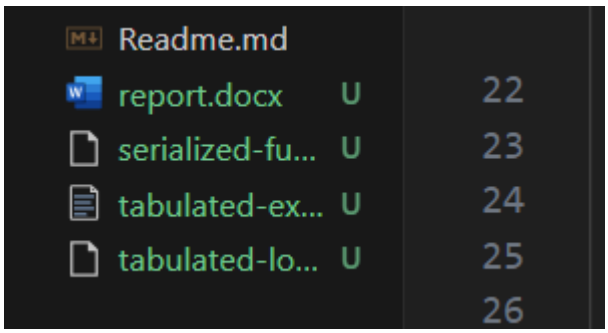


```java
public class Main {
    public static void main(String[] args) {
            OutputStream out = new FileOutputStream(name: "tabulated-log.dat");
            TabulatedFunctions.outputTabulatedFunction(tabLog, out);
            out.close();

            // Чтение из файла
            InputStream in = new FileInputStream(name: "tabulated-log.dat");
            TabulatedFunction readLog = TabulatedFunctions.inputTabulatedFunction(in);
            in.close();

            // Сравнение
            System.out.println("Сравнение исходной и считанной из файла функции:");
            for (double x = 1; x <= 10; x += 1) {
                System.out.printf("x=%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, tabLog.getFunctionValue(x), readLog.getFunctionValue(x));
            }

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

        // --- Задание 9: Тест Сериализации ---
        System.out.println("\n--- Задание 9: Тест Сериализации (Serializable) ---");
        try {
            // log(exp(x)) = x
            Function f = Functions.composition(new Log(Math.E), new Exp());
            TabulatedFunction tabFunc = TabulatedFunctions.tabulate(f, leftX: 0, rightX: 10, pointsCount: 11);

            // Сериализация (запись объекта)
            ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(name: "serialized-function.ser"));
            out.writeObject(tabFunc);
            out.close();

            // Десериализация (чтение объекта)
            ObjectInputStream in = new ObjectInputStream(new FileInputStream(name: "serialized-function.ser"));
            TabulatedFunction readFunc = (TabulatedFunction) in.readObject();
            in.close();

            // Сравнение
            System.out.println("Сравнение исходной и десериализованной функции:");
            for (double x = 0; x <= 10; x += 1) {
                System.out.printf("x=%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, tabFunc.getFunctionValue(x), readFunc.getFunctionValue(x));
            }

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


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\projects\Lab-4-2025> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:61308' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\3c1765a2ca51b4f457a12fef7ac29714\redhat.java\jdt_ws\Lab-4-2025_44202e57\bin' 'Main'
--- Задание 8.1: Sin и Cos ---
x=0,0, sin(x)=0,0000, cos(x)=1,0000
x=0,1, sin(x)=0,0998, cos(x)=0,9950
x=0,2, sin(x)=0,1987, cos(x)=0,9801
x=0,3, sin(x)=0,2955, cos(x)=0,9553
x=0,4, sin(x)=0,3894, cos(x)=0,9211
x=0,5, sin(x)=0,4794, cos(x)=0,8776
x=0,6, sin(x)=0,5646, cos(x)=0,8253
x=0,7, sin(x)=0,6442, cos(x)=0,7648
x=0,8, sin(x)=0,7174, cos(x)=0,6967
x=0,9, sin(x)=0,7833, cos(x)=0,6216
x=1,0, sin(x)=0,8415, cos(x)=0,5403
x=1,1, sin(x)=0,8912, cos(x)=0,4536
x=1,2, sin(x)=0,9320, cos(x)=0,3624
x=1,3, sin(x)=0,9636, cos(x)=0,2675
x=1,4, sin(x)=0,9854, cos(x)=0,1700
x=1,5, sin(x)=0,9975, cos(x)=0,0707
x=1,6, sin(x)=0,9996, cos(x)=-0,0292
x=1,7, sin(x)=0,9917, cos(x)=-0,1288
x=1,8, sin(x)=0,9738, cos(x)=-0,2272
x=1,9, sin(x)=0,9463, cos(x)=-0,3233
x=2,0, sin(x)=0,9093, cos(x)=-0,4161
x=2,1, sin(x)=0,8632, cos(x)=-0,5048
x=2,2, sin(x)=0,8085, cos(x)=-0,5885
x=2,3, sin(x)=0,7457, cos(x)=-0,6663
x=2,4, sin(x)=0,6755, cos(x)=-0,7374
x=2,5, sin(x)=0,5985, cos(x)=-0,8011
x=2,6, sin(x)=0,5155, cos(x)=-0,8569
x=2,7, sin(x)=0,4274, cos(x)=-0,9041
x=2,8, sin(x)=0,3350, cos(x)=-0,9422
x=2,9, sin(x)=0,2392, cos(x)=-0,9710
x=3,0, sin(x)=0,1411, cos(x)=-0,9900
x=3,1, sin(x)=0,0416, cos(x)=-0,9991
--- Задание 8.2: Табулирование Sin и Cos (10 точек) ---
x=0,0, tabSin(x)=0,0000, tabCos(x)=1,0000
x=0,1, tabSin(x)=0,0980, tabCos(x)=0,9827
x=0,2, tabSin(x)=0,1960, tabCos(x)=0,9654
x=0,3, tabSin(x)=0,2939, tabCos(x)=0,9482
x=0,4, tabSin(x)=0,3859, tabCos(x)=0,9344
x=0,5, tabSin(x)=0,4721, tabCos(x)=0,8646
x=0,6, tabSin(x)=0,5582, tabCos(x)=0,8149
x=0,7, tabSin(x)=0,6440, tabCos(x)=0,7646
x=0,8, tabSin(x)=0,7079, tabCos(x)=0,6884
x=0,9, tabSin(x)=0,7719, tabCos(x)=0,6122
x=1,0, tabSin(x)=0,8358, tabCos(x)=0,5360
x=1,1, tabSin(x)=0,8840, tabCos(x)=0,4586
x=1,2, tabSin(x)=0,9180, tabCos(x)=0,3571
x=1,3, tabSin(x)=0,9521, tabCos(x)=0,2636
x=1,4, tabSin(x)=0,9848, tabCos(x)=0,1699
x=1,5, tabSin(x)=0,9848, tabCos(x)=0,0704
x=1,6, tabSin(x)=0,9848, tabCos(x)=-0,0291
x=1,7, tabSin(x)=0,9848, tabCos(x)=-0,1285
x=1,8, tabSin(x)=0,9662, tabCos(x)=-0,2248
x=1,9, tabSin(x)=0,9322, tabCos(x)=-0,3183
x=2,0, tabSin(x)=0,8981, tabCos(x)=-0,4117
x=2,1, tabSin(x)=0,8624, tabCos(x)=-0,5043
x=2,2, tabSin(x)=0,7985, tabCos(x)=-0,5885
x=2,3, tabSin(x)=0,7345, tabCos(x)=-0,6567
x=2,4, tabSin(x)=0,6706, tabCos(x)=-0,7329
x=2,5, tabSin(x)=0,5941, tabCos(x)=-0,7942
x=2,6, tabSin(x)=0,5079, tabCos(x)=-0,8439
x=2,7, tabSin(x)=0,4217, tabCos(x)=-0,8937
x=2,8, tabSin(x)=0,3347, tabCos(x)=-0,9418
x=2,9, tabSin(x)=0,2367, tabCos(x)=-0,9583
x=3,0, tabSin(x)=0,1387, tabCos(x)=-0,9755
x=3,1, tabSin(x)=0,0408, tabCos(x)=-0,9928
--- Задание 8.3: Сумма квадратов (sin^2 + cos^2) ---
x=0,0, sin^2+cos^2=1,0000


PS C:\projects\Lab-4-2025> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:61308' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Borus\AppData\Roaming\Code\User\workspaceStorage\3c1765a2ca51b4f457a12fef7ac29714\redhat.java\jdt_ws\Lab-4-2025_44202e57\bin' 'Main'
--- Задание 8.3: Сумма квадратов (sin^2 + cos^2) ---
x=0,0, sin^2+cos^2=1,0000
x=0,1, sin^2+cos^2=0,9753
x=0,2, sin^2+cos^2=0,9705
x=0,3, sin^2+cos^2=0,9854
x=0,4, sin^2+cos^2=0,9850
x=0,5, sin^2+cos^2=0,9704
x=0,6, sin^2+cos^2=0,9756
x=0,7, sin^2+cos^2=0,9904
x=0,8, sin^2+cos^2=0,9751
x=0,9, sin^2+cos^2=0,9706
x=1,0, sin^2+cos^2=0,9859
x=1,1, sin^2+cos^2=0,9845
x=1,2, sin^2+cos^2=0,9703
x=1,3, sin^2+cos^2=0,9759
x=1,4, sin^2+cos^2=0,9987
x=1,5, sin^2+cos^2=0,9748
x=1,6, sin^2+cos^2=0,9707
x=1,7, sin^2+cos^2=0,9864
x=1,8, sin^2+cos^2=0,9841
x=1,9, sin^2+cos^2=0,9702
x=2,0, sin^2+cos^2=0,9762
x=2,1, sin^2+cos^2=0,9981
x=2,2, sin^2+cos^2=0,9745
x=2,3, sin^2+cos^2=0,9708
x=2,4, sin^2+cos^2=0,9869
x=2,5, sin^2+cos^2=0,9836
x=2,6, sin^2+cos^2=0,9702
x=2,7, sin^2+cos^2=0,9765
x=2,8, sin^2+cos^2=0,9975
x=2,9, sin^2+cos^2=0,9743
x=3,0, sin^2+cos^2=0,9709
x=3,1, sin^2+cos^2=0,9873
--- Задание 8.4: Тест write/read (текстовый файл) ---
Сравнение исходной и считанной из файла функции:
x=0,0, Orig(x)=1,00, Read(x)=1,00
x=1,0, Orig(x)=2,72, Read(x)=2,72
x=2,0, Orig(x)=7,39, Read(x)=7,39
x=3,0, Orig(x)=20,09, Read(x)=20,09
x=4,0, Orig(x)=54,60, Read(x)=54,60
x=5,0, Orig(x)=148,41, Read(x)=148,41
x=6,0, Orig(x)=403,43, Read(x)=403,43
x=7,0, Orig(x)=1096,63, Read(x)=1096,63
x=8,0, Orig(x)=2980,96, Read(x)=2980,96
x=9,0, Orig(x)=8103,08, Read(x)=8103,08
x=10,0, Orig(x)=22026,47, Read(x)=22026,47
--- Задание 8.5: Тест output/input (бинарный файл) ---
Сравнение исходной и считанной из файла функции:
x=1,0, Orig(x)=0,00, Read(x)=0,00
x=2,0, Orig(x)=0,69, Read(x)=0,69
x=3,0, Orig(x)=1,10, Read(x)=1,10
x=4,0, Orig(x)=1,39, Read(x)=1,39
x=5,0, Orig(x)=1,61, Read(x)=1,61
x=6,0, Orig(x)=1,79, Read(x)=1,79
x=7,0, Orig(x)=1,95, Read(x)=1,95
x=8,0, Orig(x)=2,08, Read(x)=2,08
x=9,0, Orig(x)=2,20, Read(x)=2,20
x=10,0, Orig(x)=2,30, Read(x)=2,30
--- Задание 9: Тест Сериализации (Serializable) ---
Сравнение исходной и десериализованной функции:
x=0,0, Orig(x)=0,00, Read(x)=0,00
x=1,0, Orig(x)=1,00, Read(x)=1,00
x=2,0, Orig(x)=2,00, Read(x)=2,00
x=3,0, Orig(x)=3,00, Read(x)=3,00
x=4,0, Orig(x)=4,00, Read(x)=4,00
x=5,0, Orig(x)=5,00, Read(x)=5,00
x=6,0, Orig(x)=6,00, Read(x)=6,00


x=10,0, Orig(x)=2,30, Read(x)=2,30
--- Задание 9: Тест Сериализации (Serializable) ---
Сравнение исходной и десериализованной функции:
x=0,0, Orig(x)=0,00, Read(x)=0,00
x=1,0, Orig(x)=1,00, Read(x)=1,00
x=2,0, Orig(x)=2,00, Read(x)=2,00
x=3,0, Orig(x)=3,00, Read(x)=3,00
x=4,0, Orig(x)=4,00, Read(x)=4,00
x=5,0, Orig(x)=5,00, Read(x)=5,00
x=6,0, Orig(x)=6,00, Read(x)=6,00
x=7,0, Orig(x)=7,00, Read(x)=7,00
x=8,0, Orig(x)=8,00, Read(x)=8,00
x=9,0, Orig(x)=9,00, Read(x)=9,00
x=10,0, Orig(x)=10,00, Read(x)=10,00
PS C:\projects\Lab-4-2025>


Readme.md
report.docx U
serialized-fu... U
tabulated-ex... U
tabulated-lo... U
22
23
24
25
26