

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 будут 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 требуемой 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.

```
src > functions > . ArrayTabulatedFunction.java > %s ArrayTabulatedFunction > @ ArrayTabulatedFunction(FunctionPoint[])
3 public class ArrayTabulatedFunction implements TabulatedFunction {
4
5 // КОНСТРУКТОРЫ С ПРОВЕРКАМИ
6
7 public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
8     // добавлена проверка аргументов
9     if (leftX >= rightX || pointsCount < 2) {
10         throw new IllegalArgumentException(s: "Invalid arguments for function creation");
11     }
12     this.points = new FunctionPoint[pointsCount];
13     double step = (rightX - leftX) / (pointsCount - 1);
14     for (int i = 0; i < pointsCount; ++i) {
15         points[i] = new FunctionPoint(leftX + i * step, y: 0);
16     }
17 }
18
19 public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
20     int count = values.length;
21     // добавлена проверка аргументов
22     if (leftX >= rightX || count < 2) {
23         throw new IllegalArgumentException(s: "Invalid arguments for function creation");
24     }
25     this.points = new FunctionPoint[count];
26     double step = (rightX - leftX) / (count - 1);
27     for (int i = 0; i < count; ++i) {
28         points[i] = new FunctionPoint(leftX + i * step, values[i]);
29     }
30 }
31
32 // Новый конструктор (Задание 1)
33 public ArrayTabulatedFunction(FunctionPoint[] points) {
34     if (points.length < 2) {
35         throw new IllegalArgumentException(s: "Function must have at least 2 points");
36     }
37     // Проверка на упорядоченность
38     for (int i = 0; i < points.length - 1; ++i) {
39         if (points[i].getX() >= points[i + 1].getX()) {
40             throw new IllegalArgumentException(s: "Points are not sorted by X");
41         }
42     }
43     // Глубокое копирование для инкапсуляции
44     this.points = new FunctionPoint[points.length];
45     for (int i = 0; i < points.length; ++i) {
46         this.points[i] = new FunctionPoint(points[i]);
47     }
48 }
49
50 // ОСНОВНЫЕ МЕТОДЫ
51 public double getLeftDomainBorder() {
52 }
```

```

src > functions > LinkedListTabulatedFunction.java >
src > functions > LinkedListTabulatedFunction implements TabulatedFunction {
16 public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
27     addNodeToTail().point = new FunctionPoint(leftX + 1 * step, y: 0);
28 }
29
30 public LinkedListTabulatedFunction(double leftX, double rightX, double[] values) {
31     if (leftX >= rightX || values.length < 2) {
32         throw new IllegalArgumentException(s: "Invalid arguments for function creation");
33     }
34     this.count = values.length;
35     this.head = new FunctionNode();
36     this.head.prev = head;
37     this.head.next = head;
38
39     double step = (rightX - leftX) / (values.length - 1);
40     for (int i = 0; i < values.length; i++) {
41         addNodeToTail().point = new FunctionPoint(leftX + i * step, values[i]);
42     }
43 }
44
45 // Новый конструктор (Задание 1)
46 public LinkedListTabulatedFunction(FunctionPoint[] points) {
47     if (points.length < 2) {
48         throw new IllegalArgumentException(s: "Function must have at least 2 points");
49     }
50     // Инициализация головы списка
51     this.count = 0;
52     this.head = new FunctionNode();
53     this.head.prev = head;
54     this.head.next = head;
55
56     for (int i = 0; i < points.length; ++i) {
57         // Проверка на упорядоченность при добавлении
58         if (i > 0 && points[i-1].getX() >= points[i].getX()) {
59             throw new IllegalArgumentException(s: "Points are not sorted by X");
60         }
61         // Глубокое копирование и добавление в хвост
62         addNodeToTail().point = new FunctionPoint(points[i]);
63         this.count++;
64     }
65 }
66
67 // МЕТОДЫ ДЛЯ РАБОТЫ СО СПИСКОМ (внутренняя логика)
68 private FunctionNode getNodeByIndex(int index) {
69     FunctionNode current = head.next;
70     for (int i = 0; i < index; i++) {
71

```

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.

```

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

```

```
Function.java U TabulatedFunction.java U X
src > functions > TabulatedFunction.java > ...
1 package functions;
2
3 /* 1. Расширяем Function (extends Function)
4 2. Добавляем Serializable для Задания 9 */
5 public interface TabulatedFunction extends Function, java.io.Serializable {
6     /* Методы getLeftDomainBorder, getRightDomainBorder, getFunctionValue
7     теперь наследуются из Function. Удаляем их отсюда. */
8
9     int getPointsCount();
10    FunctionPoint getPoint(int index);
11    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
12    double getPointX(int index);
13    void setPointX(int index, double x) throws InappropriateFunctionPointException;
14    double getPointY(int index);
15    void setPointY(int index, double y);
16    void deletePoint(int index);
17    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
18 }
19
```

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()`.

```
Exp.java U X Log.java U TrigonometricFunction.java U Sin.java U Cos.java U Tan.java U
src > functions > basic > Exp.java > ...
1 package functions.basic;
2 import functions.Function;
3
4 public class Exp implements Function {
5     public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }
6     public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
7     public double getFunctionValue(double x) { return Math.exp(x); }
8 }
9
```

```
src > functions > basic > Log.java > ...
1 package functions.basic;
2 import functions.Function;
3
4 public class Log implements Function {
5     private double base;
6     public Log(double base) {
7         if (base <= 0 || base == 1) throw new IllegalArgumentException(s: "Invalid base");
8         this.base = base;
9     }
10    public double getLeftDomainBorder() { return 0; } // log(x) определён для x > 0
11    public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
12    public double getFunctionValue(double x) { return Math.log(x) / Math.log(this.base); }
13 }
14
```

```
src > functions > basic > TrigonometricFunction.java > ...
1 package functions.basic;
2 import functions.Function;
3
4 public abstract class TrigonometricFunction implements Function {
5     public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }
6     public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
7 }
8
```

```
src > functions > basic > Sin.java > ...
1 package functions.basic;
2
3 public class Sin extends TrigonometricFunction {
4     public double getFunctionValue(double x) { return Math.sin(x); }
5 }
6
```

```
src > functions > basic > Cos.java > ...
1 package functions.basic;
2
3 public class Cos extends TrigonometricFunction {
4     public double getFunctionValue(double x) { return Math.cos(x); }
5 }
6
```

```
src > functions > basic > Tan.java > ...
1 package functions.basic;
2
3 public class Tan extends TrigonometricFunction {
4     public double getFunctionValue(double x) { return Math.tan(x); }
5 }
6
```

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).



```
src > functions > meta > Sum.java U ...
1 package functions.meta;
2 import functions.Function;
3
4 public class Sum implements Function {
5     private Function f1, f2;
6     private double left, right;
7     public Sum(Function f1, Function f2) {
8         this.f1 = f1; this.f2 = f2;
9         this.left = Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
10        this.right = Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
11    }
12    public double getLeftDomainBorder() { return left; }
13    public double getRightDomainBorder() { return right; }
14    public double getFunctionValue(double x) { return f1.getFunctionValue(x) + f2.getFunctionValue(x); }
15 }
16
```



```
src > functions > meta > Mult.java U ...
1 package functions.meta;
2 import functions.Function;
3
4 public class Mult implements Function {
5     private Function f1, f2;
6     private double left, right;
7     public Mult(Function f1, Function f2) {
8         this.f1 = f1; this.f2 = f2;
9         this.left = Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
10        this.right = Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
11    }
12    public double getLeftDomainBorder() { return left; }
13    public double getRightDomainBorder() { return right; }
14    public double getFunctionValue(double x) { return f1.getFunctionValue(x) * f2.getFunctionValue(x); }
15 }
16
```

```
src > functions > meta > Power.java > ...
1 package functions.meta;
2 import functions.Function;
3
4 public class Power implements Function {
5     private Function f;
6     private double power;
7
8     public Power(Function f, double power) {
9         this.f = f;
10        this.power = power;
11    }
12
13    public double getLeftDomainBorder() { return f.getLeftDomainBorder(); }
14    public double getRightDomainBorder() { return f.getRightDomainBorder(); }
15    public double getFunctionValue(double x) {
16        return Math.pow(f.getFunctionValue(x), power);
17    }
18 }
19
```

```
src > functions > meta > Scale.java > ...
1 package functions.meta;
2 import functions.Function;
3
4 public class Scale implements Function {
5     private Function f;
6     private double scaleX, scaleY;
7
8     public Scale(Function f, double scaleX, double scaleY) {
9         this.f = f;
10        this.scaleX = scaleX;
11        this.scaleY = scaleY;
12    }
13
14    public double getLeftDomainBorder() { return f.getLeftDomainBorder() * scaleX; }
15    public double getRightDomainBorder() { return f.getRightDomainBorder() * scaleX; }
16    public double getFunctionValue(double x) {
17        return f.getFunctionValue(x / scaleX) * scaleY;
18    }
19 }
20
```

```
src > functions > meta > Shift.java > ...
1 package functions.meta;
2 import functions.Function;
3
4 public class Shift implements Function {
5     private Function f;
6     private double shiftX, shiftY;
7
8     public Shift(Function f, double shiftX, double shiftY) {
9         this.f = f;
10        this.shiftX = shiftX;
11        this.shiftY = shiftY;
12    }
13
14    public double getLeftDomainBorder() { return f.getLeftDomainBorder() + shiftX; }
15    public double getRightDomainBorder() { return f.getRightDomainBorder() + shiftX; }
16    public double getFunctionValue(double x) {
17        return f.getFunctionValue(x - shiftX) + shiftY;
18    }
19 }

src > functions > meta > Composition.java > ...
1 package functions.meta;
2 import functions.Function;
3
4 public class Composition implements Function {
5     private Function f1, f2;
6
7     public Composition(Function f1, Function f2) {
8         this.f1 = f1;
9         this.f2 = f2;
10    }
11
12    public double getLeftDomainBorder() { return f1.getLeftDomainBorder(); }
13    public double getRightDomainBorder() { return f1.getRightDomainBorder(); }
14    public double getFunctionValue(double x) {
15        // f1(f2(x))
16        return f1.getFunctionValue(f2.getFunctionValue(x));
17    }
18 }
19 }
```

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.

```

src > Functions.java U X
1 package functions;
2 import functions.meta.*;
3
4 public class Functions {
5     // Приватный конструктор, чтобы нельзя было создать объект
6     private Functions() {}
7
8     public static Function shift(Function f, double shiftX, double shiftY) {
9         return new Shift(f, shiftX, shiftY);
10    }
11    public static Function scale(Function f, double scaleX, double scaleY) {
12        return new Scale(f, scaleX, scaleY);
13    }
14    public static Function power(Function f, double power) {
15        return new Power(f, power);
16    }
17    public static Function sum(Function f1, Function f2) {
18        return new Sum(f1, f2);
19    }
20    public static Function mult(Function f1, Function f2) {
21        return new Mult(f1, f2);
22    }
23    public static Function composition(Function f1, Function f2) {
24        return new Composition(f1, f2);
25    }
26 }
27

```

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.

```

src > TabulatedFunctions.java 1, U X
1 package functions;
2 import java.io.*;
3
4 public class TabulatedFunctions {
5     // Приватный конструктор
6     private TabulatedFunctions() {}
7
8     // --- Метод для Задания 6 ---
9     public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
10         if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
11             throw new IllegalArgumentException(s: "Tabulation bounds are out of function's domain");
12         }
13         if (pointsCount < 2) {
14             throw new IllegalArgumentException(s: "Points count must be 2 or more");
15         }
16
17         FunctionPoint[] points = new FunctionPoint[pointsCount];
18         double step = (rightX - leftX) / (pointsCount - 1);
19         for (int i = 0; i < pointsCount; i++) {
20             double x = leftX + i * step;
21             points[i] = new FunctionPoint(x, function.getFunctionValue(x));
22         }
23         // Возвращаем объект одного из классов, реализующих интерфейс
24         return new ArrayTabulatedFunction(points);
25         // Можно вернуть и LinkedListTabulatedFunction, по заданию это не важно
26     }
27 }
28

```

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 требуемой 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.

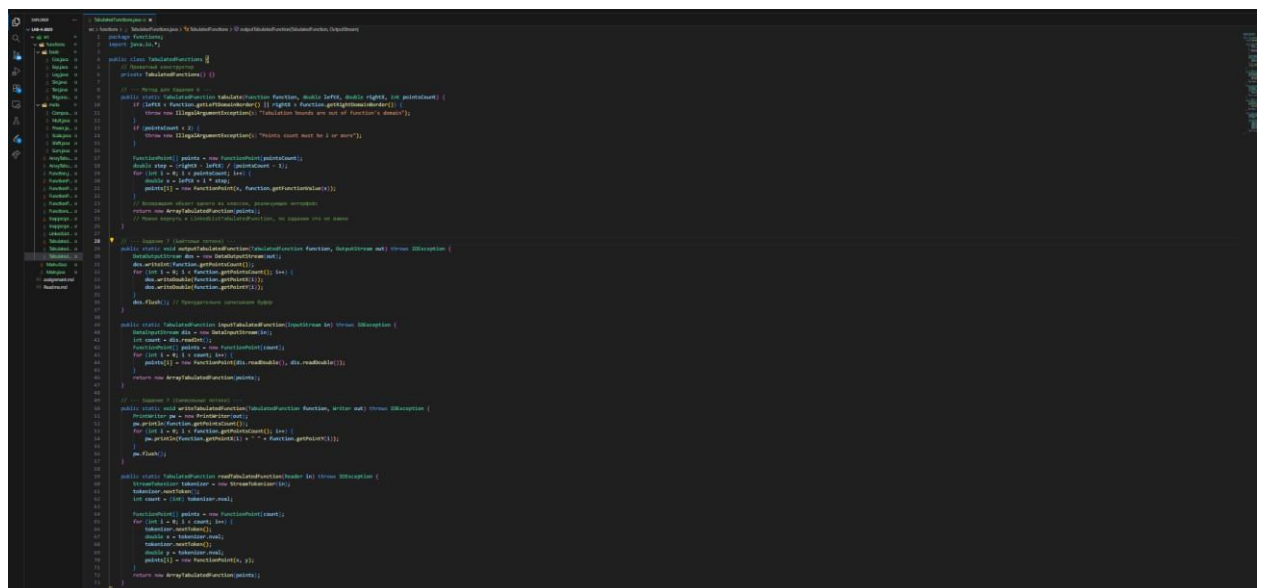
Обоснование по закрытию потоков

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: `tabulatedexp.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.

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).

```

TabulatedFunction.java U X
src > functions > TabulatedFunction.java > ...
1 package functions;
2
3 /* 1. Расширяем Function (extends Function)
4 2. Добавляем Serializable для Задания 9 */
5 public interface TabulatedFunction extends Function, java.io.Serializable {
6     /* Методы getLeftDomainBorder, getRightDomainBorder, getFunctionValue
7     теперь наследуются из Function. Удаляем их отсюда. */
8
9     int getPointsCount();
10    FunctionPoint getPoint(int index);
11    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
12    double getPointX(int index);
13    void setPointX(int index, double x) throws InappropriateFunctionPointException;
14    double getPointY(int index);
15    void setPointY(int index, double y);
16    void deletePoint(int index);
17    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;
18 }
19

```

```

FunctionPoint.java U X
src > functions > FunctionPoint.java > ...
1 package functions;
2
3 // Реализуем Serializable для Задания 9
4 public class FunctionPoint implements java.io.Serializable {
5     private double x;
6     private double y;
7
8     // Обязательный элемент для Serializable
9     private static final long serialVersionUID = 1L;
10
11     public FunctionPoint(double x, double y) {
12         this.x = x;
13         this.y = y;
14     }
15
16     public FunctionPoint(FunctionPoint point) {
17         this.x = point.x;
18         this.y = point.y;
19     }
20
21     public FunctionPoint() {
22         this.x = 0;
23         this.y = 0;
24     }
25
26     public double getX() {
27         return x;
28     }
29
30     public double getY() {
31         return y;
32     }
33
34     public void setX(double x) {
35         this.x = x;
36     }
37
38     public void setY(double y) {
39         this.y = y;
40     }
41 }
42

```

```

ArrayTabulatedFunction.java U X
src > functions > ArrayTabulatedFunction.java > ...
1 package functions;
2
3 import java.io.Serializable;
4
5 // Класс-реализация интерфейса TabulatedFunction, расширяющая интерфейс Function = Serializable
6 public class ArrayTabulatedFunction implements TabulatedFunction {
7     private FunctionPoint[] points;
8     private static final long serialVersionUID = 1L; // для Serializable
9
10    // Конструктор DP 1
11    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
12        if (leftX >= rightX || pointsCount < 2) {
13            throw new IllegalArgumentException("Invalid arguments for function creation: leftX >= rightX or pointsCount < 2");
14        }
15
16        this.points = new FunctionPoint[pointsCount];
17        double step = (rightX - leftX) / (pointsCount - 1);
18        for (int i = 0; i < pointsCount; ++i) {
19            points[i] = new FunctionPoint(leftX + i * step, y: 0);
20        }
21    }
22
23    // Конструктор DP 2
24    public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
25        if (leftX >= rightX || values.length < 2) {
26            throw new IllegalArgumentException("Invalid arguments for function creation: leftX >= rightX or pointsCount < 2");
27        }
28
29        this.points = new FunctionPoint[values.length];
30        double step = (rightX - leftX) / (values.length - 1);
31        for (int i = 0; i < values.length; ++i) {
32            points[i] = new FunctionPoint(leftX + i * step, values[i]);
33        }
34    }
35
36    // Проверка корректности DP 3 (назовем it)
37    public ArrayTabulatedFunction(FunctionPoint[] points) {
38        if (points.length < 2) {
39            throw new IllegalArgumentException("Function must have at least 2 points");
40        }
41
42        // Проверка на корректность
43        for (int i = 0; i < points.length - 1; ++i) {
44            // Проверяем, что значения не строго убывают/возрастают (x[i] <= x[i+1])
45            if (points[i].getX() >= points[i + 1].getX()) {
46                throw new IllegalArgumentException("Points are not strictly sorted by X");
47            }
48        }
49
50        // Проверяем корректность get pointsCount
51        this.points = new FunctionPoint[points.length];
52        for (int i = 0; i < points.length; ++i) {
53            this.points[i] = new FunctionPoint(points[i]);
54        }
55    }
56
57    public double getLeftDomainBorder() {
58        return points[0].getX();
59    }
60
61    public double getRightDomainBorder() {
62        return points[points.length - 1].getX();
63    }
64
65    public double getFunctionValue(double x) {
66        if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
67            return Double.NaN;
68        }
69
70        for (int i = 0; i < points.length - 1; ++i) {
71            if (points[i].getX() <= x && x <= points[i + 1].getX()) {
72                double x1 = points[i].getX();
73                double y1 = points[i].getY();
74

```


[illegible]

```

1  public int[] findShortestPathes(int[][] roads, int[][] dists) {
2      public void checkInterval(int start, int end) {
3          if (start > end) {
4              return;
5          }
6          if (start == end) {
7              return;
8          }
9          if (start < end) {
10             return;
11         }
12     }
13
14     public void checkInterval(int start, int end) {
15         if (start > end) {
16             return;
17         }
18         if (start == end) {
19             return;
20         }
21         if (start < end) {
22             return;
23         }
24     }
25
26     public void checkInterval(int start, int end) {
27         if (start > end) {
28             return;
29         }
30         if (start == end) {
31             return;
32         }
33         if (start < end) {
34             return;
35         }
36     }
37
38     public void checkInterval(int start, int end) {
39         if (start > end) {
40             return;
41         }
42         if (start == end) {
43             return;
44         }
45         if (start < end) {
46             return;
47         }
48     }
49
50     public void checkInterval(int start, int end) {
51         if (start > end) {
52             return;
53         }
54         if (start == end) {
55             return;
56         }
57         if (start < end) {
58             return;
59         }
60     }
61
62     public void checkInterval(int start, int end) {
63         if (start > end) {
64             return;
65         }
66         if (start == end) {
67             return;
68         }
69         if (start < end) {
70             return;
71         }
72     }
73
74     public void checkInterval(int start, int end) {
75         if (start > end) {
76             return;
77         }
78         if (start == end) {
79             return;
80         }
81         if (start < end) {
82             return;
83         }
84     }
85
86     public void checkInterval(int start, int end) {
87         if (start > end) {
88             return;
89         }
90         if (start == end) {
91             return;
92         }
93         if (start < end) {
94             return;
95         }
96     }
97
98     public void checkInterval(int start, int end) {
99         if (start > end) {
100             return;
101         }
102         if (start == end) {
103             return;
104         }
105         if (start < end) {
106             return;
107         }
108     }
109
110     public void checkInterval(int start, int end) {
111         if (start > end) {
112             return;
113         }
114         if (start == end) {
115             return;
116         }
117         if (start < end) {
118             return;
119         }
120     }
121
122     public void checkInterval(int start, int end) {
123         if (start > end) {
124             return;
125         }
126         if (start == end) {
127             return;
128         }
129         if (start < end) {
130             return;
131         }
132     }
133
134     public void checkInterval(int start, int end) {
135         if (start > end) {
136             return;
137         }
138         if (start == end) {
139             return;
140         }
141         if (start < end) {
142             return;
143         }
144     }
145
146     public void checkInterval(int start, int end) {
147         if (start > end) {
148             return;
149         }
150         if (start == end) {
151             return;
152         }
153         if (start < end) {
154             return;
155         }
156     }
157
158     public void checkInterval(int start, int end) {
159         if (start > end) {
160             return;
161         }
162         if (start == end) {
163             return;
164         }
165         if (start < end) {
166             return;
167         }
168     }
169
170     public void checkInterval(int start, int end) {
171         if (start > end) {
172             return;
173         }
174         if (start == end) {
175             return;
176         }
177         if (start < end) {
178             return;
179         }
180     }
181
182     public void checkInterval(int start, int end) {
183         if (start > end) {
184             return;
185         }
186         if (start == end) {
187             return;
188         }
189         if (start < end) {
190             return;
191         }
192     }
193
194     public void checkInterval(int start, int end) {
195         if (start > end) {
196             return;
197         }
198         if (start == end) {
199             return;
200         }
201         if (start < end) {
202             return;
203         }
204     }
205
206     public void checkInterval(int start, int end) {
207         if (start > end) {
208             return;
209         }
210         if (start == end) {
211             return;
212         }
213         if (start < end) {
214             return;
215         }
216     }
217
218     public void checkInterval(int start, int end) {
219         if (start > end) {
220             return;
221         }
222         if (start == end) {
223             return;
224         }
225         if (start < end) {
226             return;
227         }
228     }
229
230     public void checkInterval(int start, int end) {
231         if (start > end) {
232             return;
233         }
234         if (start == end) {
235             return;
236         }
237         if (start < end) {
238             return;
239         }
240     }
241
242     public void checkInterval(int start, int end) {
243         if (start > end) {
244             return;
245         }
246         if (start == end) {
247             return;
248         }
249         if (start < end) {
250             return;
251         }
252     }
253
254     public void checkInterval(int start, int end) {
255         if (start > end) {
256             return;
257         }
258         if (start == end) {
259             return;
260         }
261         if (start < end) {
262             return;
263         }
264     }
265
266     public void checkInterval(int start, int end) {
267         if (start > end) {
268             return;
269         }
270         if (start == end) {
271             return;
272         }
273         if (start < end) {
274             return;
275         }
276     }
277
278     public void checkInterval(int start, int end) {
279         if (start > end) {
280             return;
281         }
282         if (start == end) {
283             return;
284         }
285         if (start < end) {
286             return;
287         }
288     }
289
290     public void checkInterval(int start, int end) {
291         if (start > end) {
292             return;
293         }
294         if (start == end) {
295             return;
296         }
297         if (start < end) {
298             return;
299         }
300     }
301
302     public void checkInterval(int start, int end) {
303         if (start > end) {
304             return;
305         }
306         if (start == end) {
307             return;
308         }
309         if (start < end) {
310             return;
311         }
312     }
313
314     public void checkInterval(int start, int end) {
315         if (start > end) {
316             return;
317         }
318         if (start == end) {
319             return;
320         }
321         if (start < end) {
322             return;
323         }
324     }
325
326     public void checkInterval(int start, int end) {
327         if (start > end) {
328             return;
329         }
330         if (start == end) {
331             return;
332         }
333         if (start < end) {
334             return;
335         }
336     }
337
338     public void checkInterval(int start, int end) {
339         if (start > end) {
340             return;
341         }
342         if (start == end) {
343             return;
344         }
345         if (start < end) {
346             return;
347         }
348     }
349
350     public void checkInterval(int start, int end) {
351         if (start > end) {
352             return;
353         }
354         if (start == end) {
355             return;
356         }
357         if (start < end) {
358             return;
359         }
360     }
361
362     public void checkInterval(int start, int end) {
363         if (start > end) {
364             return;
365         }
366         if (start == end) {
367             return;
368         }
369         if (start < end) {
370             return;
371         }
372     }
373
374     public void checkInterval(int start, int end) {
375         if (start > end) {
376             return;
377         }
378         if (start == end) {
379             return;
380         }
381         if (start < end) {
382             return;
383         }
384     }
385
386     public void checkInterval(int start, int end) {
387         if (start > end) {
388             return;
389         }
390         if (start == end) {
391             return;
392         }
393         if (start < end) {
394             return;
395         }
396     }
397
398     public void checkInterval(int start, int end) {
399         if (start > end) {
400             return;
401         }
402         if (start == end) {
403             return;
404         }
405         if (start < end) {
406             return;
407         }
408     }
409
410     public void checkInterval(int start, int end) {
411         if (start > end) {
412             return;
413         }
414         if (start == end) {
415             return;
416         }
417         if (start < end) {
418             return;
419         }
420     }
421
422     public void checkInterval(int start, int end) {
423         if (start > end) {
424             return;
425         }
426         if (start == end) {
427             return;
428         }
429         if (start < end) {
430             return;
431         }
432     }
433
434     public void checkInterval(int start, int end) {
435         if (start > end) {
436             return;
437         }
438         if (start == end) {
439             return;
440         }
441         if (start < end) {
442             return;
443         }
444     }
445
446     public void checkInterval(int start, int end) {
447         if (start > end) {
448             return;
449         }
450         if (start == end) {
451             return;
452         }
453         if (start < end) {
454             return;
455         }
456     }
457
458     public void checkInterval(int start, int end) {
459         if (start > end) {
460             return;
461         }
462         if (start == end) {
463             return;
464         }
465         if (start < end) {
466             return;
467         }
468     }
469
470     public void checkInterval(int start, int end) {
471         if (start > end) {
472             return;
473         }
474         if (start == end) {
475             return;
476         }
477         if (start < end) {
4
```

```

public void getNthByN(int index, double pi) {
    checkIndex(index);
    getNthByIndex(index, point, nNth*pi);
}

public void calculateNthByN(int index) {
    checkIndex(index);
    if (index < 0) {
        throw new IllegalArgumentException("N'th must be point, function must have at least 2 points");
    }
    calculateByIndex(index);
}

public void addNthAt(FunctionNth point) throws IllegalArgumentException {
    FunctionNth current = head.next;
    int index = 0;
    // loop until we reach n
    while (current != head && current.point.getX() != point.getX()) {
        current = current.next;
        index++;
    }
    // return n if addNth
    if (current != head && current.point.getX() == point.getX()) {
        throw new IllegalArgumentException("N'th already exists in the list. Point with such X already exists");
    }
    // otherwise just make "current" (i.e. the index)
    addNthByIndex(index, point = new FunctionNth(point));
}

// ===== NETWORKS =====> (Chapter 9) ...

public void writeNthAt(FunctionNth nth) throws IOException {
    writeToFile(current);
    FunctionNth current = head.next;
    // if current is null then
    while(current != head) {
        writeNthAt(current.point);
        current = current.next;
    }
}

public void readNthAt(FunctionNth nth) throws IOException, ClassNotFoundException {
    int readNth = Integer.parseInt(nth.getX());
    // if readNth, the value isn't bigger or smaller
    while(readNth <= 0) {
        readNth = readNth + 1;
    }
    while(readNth > 0) {
        for (int i = 0; i < readNth; i++) {
            FunctionNth point = readNthAt(i);
            checkNthAt(i, point = point); // check whether a point
            writeNthAt(i); // save point into file
        }
    }
}

```

```
ManJava 1.8.0_102
src > # ManJava >...
1 import functions.*;
2 import functions.basic.*;
3 import java.io.*;
4
5 public class Main {
6
7     public static void main(String[] args) {
8
9         // --- Задание 8, п.1: Sin и Cos ---
10        System.out.println(x: "Задание 8.1: Sin и Cos ---");
11        Function sin = new Sin();
12        Function cos = new Cos();
13        for (double x = 0; x <= Math.PI; x += 0.1) {
14            System.out.printf(format: "%x%.1f, sin(x)=%.4f, cos(x)=%.4f\n", x, sin.getFunctionValue(x), cos.getFunctionValue(x));
15        }
16
17        // --- Задание 8, п.2: Табулирование Sin и Cos ---
18        System.out.println(x: "\n--- Задание 8.2: Табулирование Sin и Cos (10 точек) ---");
19        TabulatedFunction tabSin = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
20        TabulatedFunction tabCos = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);
21        for (double x = 0; x <= Math.PI; x += 0.1) {
22            System.out.printf(format: "%x%.1f, tabSin(x)=%.4f, tabCos(x)=%.4f\n", x, tabSin.getFunctionValue(x), tabCos.getFunctionValue(x));
23        }
24
25        // --- Задание 8, п.3: Сумма квадратов ---
26        System.out.println(x: "\n--- Задание 8.3: Сумма квадратов (sin^2 + cos^2) ---");
27        Function sinSq = Functions.power(tabSin, power: 2);
28        Function cosSq = Functions.power(tabCos, power: 2);
29        Function sumSq = Functions.sum(sinSq, cosSq);
30        for (double x = 0; x <= Math.PI; x += 0.1) {
31            System.out.printf(format: "%x%.1f, sin^2+cos^2=%.4f\n", x, sumSq.getFunctionValue(x));
32        }
33
34        // --- Задание 8, п.4: Тест write/read (символьный поток) ---
35        System.out.println(x: "\n--- Задание 8.4: Тест write/read (текстовый файл) ---");
36        try {
37            TabulatedFunction tabExp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);
38
39            // Запись в файл
40            Writer out = new FileWriter(fileName: "tabulated-exp.txt");
41            TabulatedFunctions.writeTabulatedFunction(tabExp, out);
42            out.close();
43
44            // Чтение из файла
45            Reader in = new FileReader(fileName: "tabulated-exp.txt");
46            TabulatedFunction readExp = TabulatedFunctions.readTabulatedFunction(in);
47            in.close();
48        }
49    }
50}
```

```
ManJava 1.8.0_102
src > # ManJava >...
5 public class Main {
6
7     public static void main(String[] args) {
8
9         // --- Задание 8, п.4: Тест write/read (символьный поток) ---
10        System.out.println(x: "\n--- Задание 8.4: Тест write/read (текстовый файл) ---");
11        try {
12            TabulatedFunction tabExp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);
13
14            // Запись в файл
15            Writer out = new FileWriter(fileName: "tabulated-exp.txt");
16            TabulatedFunctions.writeTabulatedFunction(tabExp, out);
17            out.close();
18
19            // Чтение из файла
20            Reader in = new FileReader(fileName: "tabulated-exp.txt");
21            TabulatedFunction readExp = TabulatedFunctions.readTabulatedFunction(in);
22            in.close();
23
24            // Сравнение
25            System.out.println(x: "Сравнение исходной и считанной из файла функции:");
26            for (double x = 0; x <= 10; x += 1) {
27                System.out.printf(format: "%x%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, tabExp.getFunctionValue(x), readExp.getFunctionValue(x));
28            }
29        } catch (IOException e) {
30            e.printStackTrace();
31        }
32
33        // --- Задание 8, п.5: Тест output/input (байтовый поток) ---
34        System.out.println(x: "\n--- Задание 8.5: Тест output/input (бинарный файл) ---");
35        try {
36            TabulatedFunction tabLog = TabulatedFunctions.tabulate(new Log(Math.E), leftX: 1, rightX: 10, pointsCount: 10);
37
38            // Запись в файл
39            OutputStream out = new FileOutputStream(name: "tabulated-log.dat");
40            TabulatedFunctions.outputTabulatedFunction(tabLog, out);
41            out.close();
42
43            // Чтение из файла
44            InputStream in = new FileInputStream(name: "tabulated-log.dat");
45            TabulatedFunction readLog = TabulatedFunctions.inputTabulatedFunction(in);
46            in.close();
47
48            // Сравнение
49            System.out.println(x: "Сравнение исходной и считанной из файла функции:");
50            for (double x = 1; x <= 10; x += 1) {
51                System.out.printf(format: "%x%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, tabLog.getFunctionValue(x), readLog.getFunctionValue(x));
52            }
53        } catch (IOException e) {
54            e.printStackTrace();
55        }
56
57        // --- Задание 9: Тест Серяализации (Serializable) ---
58        System.out.println(x: "\n--- Задание 9: Тест Серяализации (Serializable, ArrayTabulatedFunction) ---");
59        try {
60            // log(exp(x)) = x
61            Function f = Functions.composition(new Log(Math.E), new Exp());
62            // Имплементация ArrayTabulatedFunction для теста Serializable
63            ArrayTabulatedFunction arrayTabulatedFunction = new ArrayTabulatedFunction(f, leftX: 0, rightX: 10, pointsCount: 11);
64            // Запись в файл
65            OutputStream out = new FileOutputStream(name: "tabulated-log.dat");
66            TabulatedFunctions.outputTabulatedFunction(arrayTabulatedFunction, out);
67            out.close();
68
69            // Чтение из файла
70            InputStream in = new FileInputStream(name: "tabulated-log.dat");
71            TabulatedFunction readLog = TabulatedFunctions.inputTabulatedFunction(in);
72            in.close();
73
74            // Сравнение
75            System.out.println(x: "Сравнение исходной и считанной из файла функции:");
76            for (double x = 1; x <= 10; x += 1) {
77                System.out.printf(format: "%x%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, arrayTabulatedFunction.getFunctionValue(x), readLog.getFunctionValue(x));
78            }
79        } catch (IOException e) {
80            e.printStackTrace();
81        }
82
83        // --- Задание 9: Тест Серяализации (Serializable) ---
84        System.out.println(x: "\n--- Задание 9: Тест Серяализации (Serializable, ArrayTabulatedFunction) ---");
85        try {
86            // log(exp(x)) = x
87            Function f = Functions.composition(new Log(Math.E), new Exp());
88            // Имплементация ArrayTabulatedFunction для теста Serializable
89            ArrayTabulatedFunction arrayTabulatedFunction = new ArrayTabulatedFunction(f, leftX: 0, rightX: 10, pointsCount: 11);
90            // Запись в файл
91            OutputStream out = new FileOutputStream(name: "tabulated-log.dat");
92            TabulatedFunctions.outputTabulatedFunction(arrayTabulatedFunction, out);
93            out.close();
94
95            // Чтение из файла
96            InputStream in = new FileInputStream(name: "tabulated-log.dat");
97            TabulatedFunction readLog = TabulatedFunctions.inputTabulatedFunction(in);
98            in.close();
99
100           // Сравнение
101           System.out.println(x: "Сравнение исходной и считанной из файла функции:");
102           for (double x = 1; x <= 10; x += 1) {
103               System.out.printf(format: "%x%.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, arrayTabulatedFunction.getFunctionValue(x), readLog.getFunctionValue(x));
104           }
105       } catch (IOException e) {
106           e.printStackTrace();
107       }
108   }
109 }
```

```
src > / Main.java >
5 public class Main {
7     public static void main(String[] args) {
8         // log(exp(x)) = x
88         Function f = Functions.composition(new log(Math.E), new Exp());
89         // Используем ArrayTabulatedFunction для теста Serializable
90         TabulatedFunction tabFunc = TabulatedFunctions.tabulate(f, leftX: 0, rightX: 10, pointsCount: 11);
91
92         // Сериализация (запись объекта)
93         ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(name: "serialized-array.ser"));
94         out.writeObject(tabFunc);
95         out.close();
96
97         // Десериализация (чтение объекта)
98         ObjectInputStream in = new ObjectInputStream(new FileInputStream(name: "serialized-array.ser"));
99         TabulatedFunction readFunc = (TabulatedFunction) in.readObject();
100         in.close();
101
102         // Сравнение
103         System.out.println(x: "Сравнение исходной и десериализованной функции (Array):");
104         for (double x = 0; x <= 10; x += 1) {
105             System.out.printf(format: "x=%1f, Orig(x)=%2f, Read(x)=%2f\n", x, tabFunc.getFunctionValue(x), readFunc.getFunctionValue(x));
106         }
107     } catch (IOException | ClassNotFoundException e) {
108         e.printStackTrace();
109     }
110 }
111
112 // --- ДОБАВЛЕННЫЙ ТЕСТ: Задание 9 (Externalizable) ---
113 System.out.println(x: "\n--- Задание 9: Тест Сериализации (Externalizable, LinkedListTabulatedFunction) ---");
114 try {
115     // Создаем функцию f(x) = x^2
116     Function sqr = new Power(new Function() {
117         public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }
118         public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
119         public double getFunctionValue(double x) { return x; }
120     }, 2);
121
122     // Создаем табулированную функцию на основе LinkedListTabulatedFunction
123     double[] values = {0, 1, 4, 9, 16};
124     TabulatedFunction tabFuncLinked = new LinkedListTabulatedFunction(leftX: 0, rightX: 4, values);
125
126     System.out.println(x: "Исходная функция (LinkedList):");
127     for (int i = 0; i < tabFuncLinked.getPointsCount(); i++) {
128         System.out.printf(format: "Point %d: (%1f, %1f)\n", i, tabFuncLinked.getPointX(i), tabFuncLinked.getPointY(i));
129     }
130
131     // Сериализация (запись объекта)
132     ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(name: "serialized-linkedlist.ser"));
133     out.writeObject(tabFuncLinked);
134     out.close();
135
136     // Десериализация (чтение объекта)
137     ObjectInputStream in = new ObjectInputStream(new FileInputStream(name: "serialized-linkedlist.ser"));
138     TabulatedFunction readFuncLinked = (TabulatedFunction) in.readObject();
139     in.close();
140
141     // Сравнение
142     System.out.println(x: "Сравнение исходной и десериализованной функции (LinkedList):");
143     for (double x = 0; x <= 4; x += 0.5) {
144         System.out.printf(format: "x=%1f, Orig(x)=%2f, Read(x)=%2f\n", x, tabFuncLinked.getFunctionValue(x), readFuncLinked.getFunctionValue(x));
145     }
146 } catch (IOException | ClassNotFoundException e) {
147     e.printStackTrace();
148 }
149
150 }
```

```
125     TabulatedFunction tabFuncLinked = new LinkedListTabulatedFunction(leftX: 0, rightX: 4, values);
126
127     System.out.println(x: "Исходная функция (LinkedList):");
128     for (int i = 0; i < tabFuncLinked.getPointsCount(); i++) {
129         System.out.printf(format: "Point %d: (%1f, %1f)\n", i, tabFuncLinked.getPointX(i), tabFuncLinked.getPointY(i));
130     }
131
132     // Сериализация (запись объекта)
133     ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(name: "serialized-linkedlist.ser"));
134     out.writeObject(tabFuncLinked);
135     out.close();
136
137     // Десериализация (чтение объекта)
138     ObjectInputStream in = new ObjectInputStream(new FileInputStream(name: "serialized-linkedlist.ser"));
139     TabulatedFunction readFuncLinked = (TabulatedFunction) in.readObject();
140     in.close();
141
142     // Сравнение
143     System.out.println(x: "Сравнение исходной и десериализованной функции (LinkedList):");
144     for (double x = 0; x <= 4; x += 0.5) {
145         System.out.printf(format: "x=%1f, Orig(x)=%2f, Read(x)=%2f\n", x, tabFuncLinked.getFunctionValue(x), readFuncLinked.getFunctionValue(x));
146     }
147 } catch (IOException | ClassNotFoundException e) {
148     e.printStackTrace();
149 }
150 }
```

```
PS C:\projects\Lab-4-2025> cd 'C:\projects\Lab-4-2025'; & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=local
host:58520' '-XX:ShowCodeDetails' InExceptionMessages' -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) ---
```

```
PS C:\projects\lab-4-2025> c:\cd 'c:\projects\lab-4-2025' ; & 'C:\Program Files\Java\jdk-24\bin\java.exe' -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=local
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)=0,9144
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)=0,7719, tabCos(x)=0,6122
x=1,0, tabSin(x)=0,8358, tabCos(x)=0,5360
x=1,1, tabSin(x)=0,8840, tabCos(x)=0,4506
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)=-0,5805
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
x=0,1, sin^2+cos^2=0,9753
x=0,2, sin^2+cos^2=0,9705
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\projects\lab-4-2025> c:\cd 'c:\projects\lab-4-2025' ; & 'C:\Program Files\Java\jdk-24\bin\java.exe' -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=local
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=0,9850
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=0,9994
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
```

```
PS C:\projects\Lab-4-2025> c++; cd 'c:\projects\Lab-4-2025'; & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=local
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, ArrayTabulatedFunction) ---
Сравнение исходной и десериализованной функции (Array):
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
```

```
PS C:\projects\Lab-4-2025> c++; cd 'c:\projects\Lab-4-2025'; & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=local
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, ArrayTabulatedFunction) ---
Сравнение исходной и десериализованной функции (Array):
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

--- Задание 9: Тест Сериализации (Externalizable, LinkedListTabulatedFunction) ---
Исходная функция (LinkedList):
Point 0: (0,0, 0,0)
Point 1: (1,0, 1,0)
Point 2: (2,0, 4,0)
Point 3: (3,0, 9,0)
Point 4: (4,0, 16,0)
Сравнение исходной и десериализованной функции (LinkedList):
x=0,0, Orig(x)=0,00, Read(x)=0,00
x=0,5, Orig(x)=0,50, Read(x)=0,50
x=1,0, Orig(x)=1,00, Read(x)=1,00
x=1,5, Orig(x)=2,50, Read(x)=2,50
x=2,0, Orig(x)=4,00, Read(x)=4,00
x=2,5, Orig(x)=6,50, Read(x)=6,50
x=3,0, Orig(x)=9,00, Read(x)=9,00
x=3,5, Orig(x)=12,50, Read(x)=12,50
x=4,0, Orig(x)=16,00, Read(x)=16,00
PS C:\projects\Lab-4-2025>
```

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