

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 требующейся форме, а в четвёртом случае – классом `StreamTokenizer`.

При написании методов, считающих 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 функции 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:

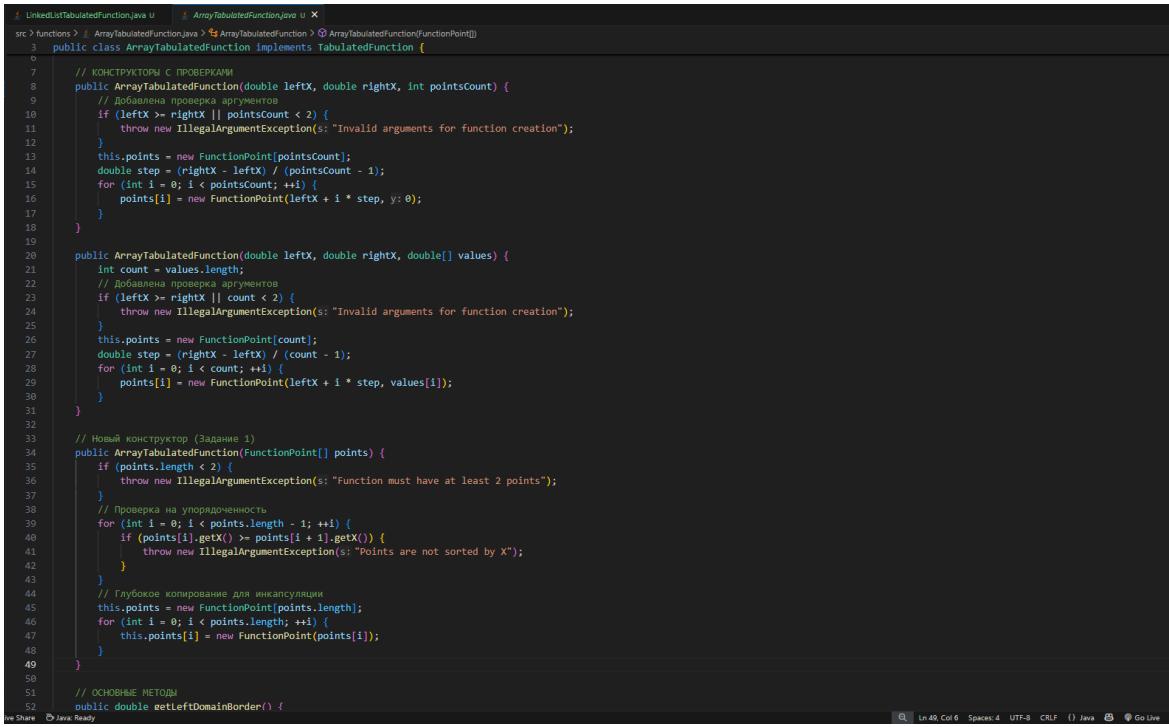
1. с использованием интерфейса `java.io.Serializable`
2. с использованием интерфейса `java.io.Externalizable`

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.



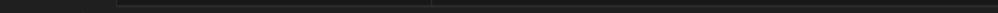
```
src > Functions > ↗ ArrayTabulatedFunction.java > ↗ ArrayTabulatedFunction > ↗ ArrayTabulatedFunction<FunctionPoint[]>
3  public class ArrayTabulatedFunction implements TabulatedFunction {
4
5      // КОНСТРУКТОРЫ С ПРОВЕРКАМИ
6      public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
7          // Добавлена проверка аргументов
8          if (leftX >= rightX || pointsCount < 2) {
9              throw new IllegalArgumentException("Invalid arguments for function creation");
10         }
11         this.points = new FunctionPoint[pointsCount];
12         double step = (rightX - leftX) / (pointsCount - 1);
13         for (int i = 0; i < pointsCount; ++i) {
14             points[i] = new FunctionPoint(leftX + i * step, y: 0);
15         }
16     }
17
18
19
20     public ArrayTabulatedFunction(double leftX, double rightX, double[] values) {
21         int count = values.length;
22         // Добавлена проверка аргументов
23         if (leftX >= rightX || count < 2) {
24             throw new IllegalArgumentException("Invalid arguments for function creation");
25         }
26         this.points = new FunctionPoint[count];
27         double step = (rightX - leftX) / (count - 1);
28         for (int i = 0; i < count; ++i) {
29             points[i] = new FunctionPoint(leftX + i * step, values[i]);
30         }
31     }
32
33     // Новый конструктор (Задание 1)
34     public ArrayTabulatedFunction(FunctionPoint[] points) {
35         if (points.length < 2) {
36             throw new IllegalArgumentException("Function must have at least 2 points");
37         }
38         // Проверка на упорядоченность
39         for (int i = 0; i < points.length - 1; ++i) {
40             if (points[i].getX() >= points[i + 1].getX()) {
41                 throw new IllegalArgumentException("Points are not sorted by X");
42             }
43         }
44         // Глубокое копирование для инкапсуляции
45         this.points = new FunctionPoint[points.length];
46         for (int i = 0; i < points.length; ++i) {
47             this.points[i] = new FunctionPoint(points[i]);
48         }
49     }
50
51     // ОСНОВНЫЕ МЕТОДЫ
52     public double getLeftDomainBorder() {
53
54
55
56
57
58
59
59 }
```

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
```

```

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().

```

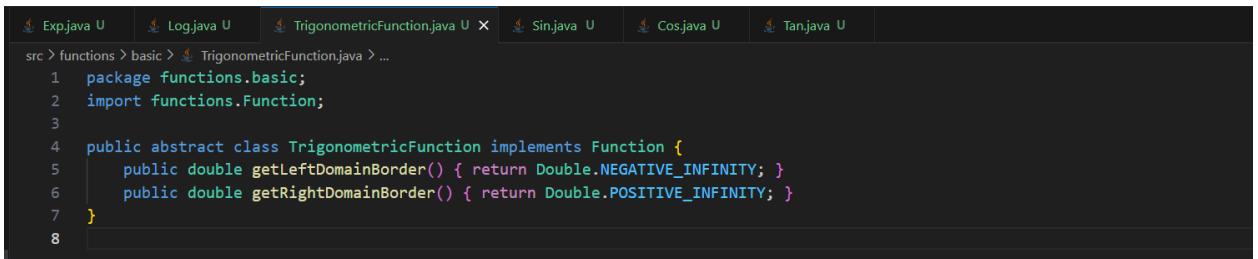
src > functions > basic > Exp.java > ...
1 package functions.basic;
2 import functions.Function;
3
4 public class Exp implements Function {
5     public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }
6     public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
7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

```

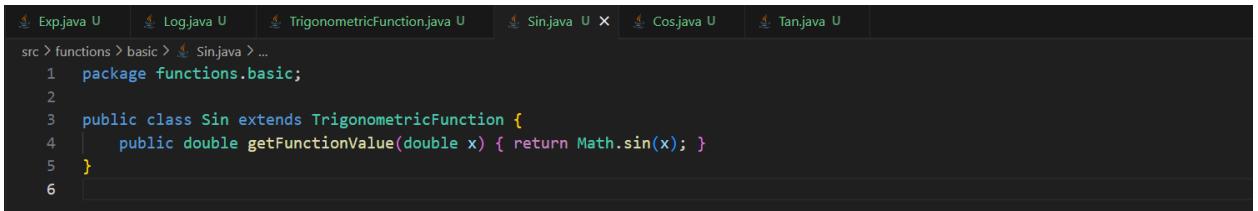
```

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

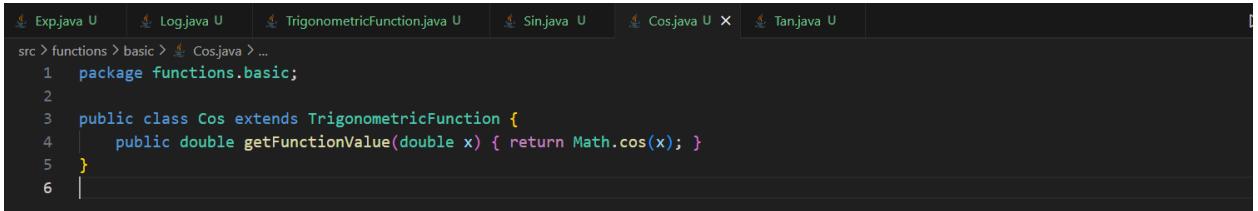
```



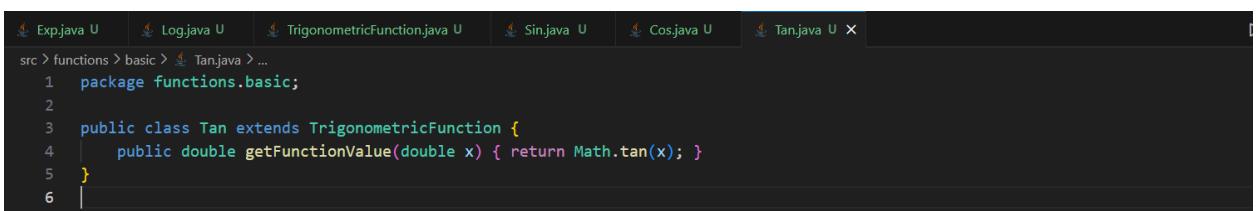
```
src > functions > basic > TrigonometricFunction.java > ...
1 package functions.basic;
2 import functions.Function;
3
4 public abstract class TrigonometricFunction implements Function {
5     public double getLeftDomainBorder() { return Double.NEGATIVE_INFINITY; }
6     public double getRightDomainBorder()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

```

Sum.java U X Mult.java U Power.java U Scale.java U Shift.java U Composition.java U
src > functions > meta > Sum.java > ...
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

```

```

Sum.java U Mult.java U X Power.java U Scale.java U Shift.java U Composition.java U
src > functions > meta > Mult.java > ...
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

```

```

Sum.java U Mult.java U Power.java U X Scale.java U Shift.java U Composition.java U
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

```

```

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 }

```



```

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 }

```



```

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

```

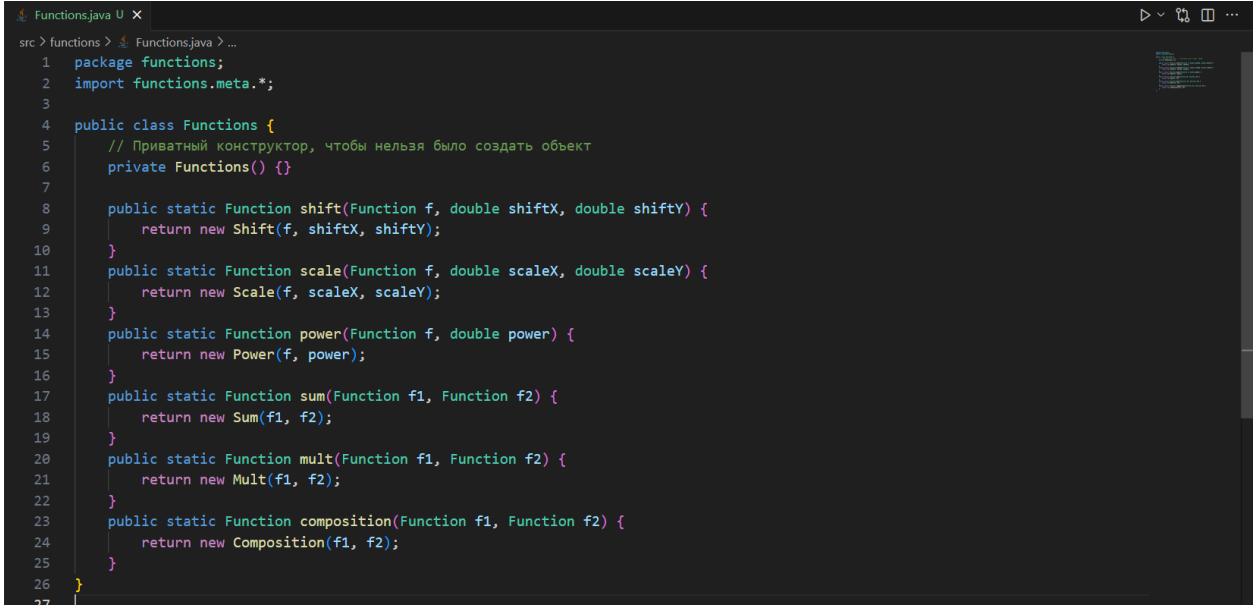
Задание 5

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 > Functions.java > ...
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.

```
TabulatedFunctions.java 1, U
src > functions > TabulatedFunctions.java > ...
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, но 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 требующейся форме, а в четвёртом случае – классом StreamTokenizer.

При написании методов, считающих 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 прорывается 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?).

Прорасывая 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 попробовал 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
PS C:\Program Files\Java\jdk-24\bin\java.exe' -agentlib:jdwp=transport=dt_socket,server=y,address=localhost:62397 "-Xt:showCodeDetailsInExceptionMessages" "-cp" "C:\Users\Korus\AppData\Roaming\Code\User\workspaceStorage\3c17652ca3b4f457a12fe7ac20714\redhat\java\lib\*.jar" "lab-4-2825_44302e5715in"
... Задание 8.2: Таблицы для Sin и Cos (10 точек) ...
x0,0, sin(x)=0.000, cos(x)=1.000
x0,1, sin(x)=0.000, cos(x)=0.9998
x0,2, sin(x)=0.198, cos(x)=0.9881
x0,3, sin(x)=0.255, cos(x)=0.9421
x0,4, sin(x)=0.314, cos(x)=0.8721
x0,5, sin(x)=0.378, cos(x)=0.8776
x0,6, sin(x)=0.5665, cos(x)=0.8251
x0,7, sin(x)=0.7268, cos(x)=0.7648
x0,8, sin(x)=0.774, cos(x)=0.6967
x0,9, sin(x)=0.7831, cos(x)=0.6216
x1,0, sin(x)=0.7831, cos(x)=0.5453
x1,1, sin(x)=0.8912, cos(x)=0.4536
x1,2, sin(x)=0.932, cos(x)=0.3624
x1,3, sin(x)=0.951, cos(x)=0.2739
x1,4, sin(x)=0.9585, cos(x)=0.1798
x1,5, sin(x)=0.9975, cos(x)=0.0799
x1,6, sin(x)=0.9999, cos(x)=0.0002
x1,7, sin(x)=0.9917, cos(x)=0.1288
x1,8, sin(x)=0.9789, cos(x)=0.2772
x1,9, sin(x)=0.951, cos(x)=0.4123
x2,0, sin(x)=0.8993, cos(x)=0.4161
x2,1, sin(x)=0.8612, cos(x)=0.5648
x2,2, sin(x)=0.804, cos(x)=0.6459
x2,3, sin(x)=0.7437, cos(x)=0.6663
x2,4, sin(x)=0.6795, cos(x)=0.7374
x2,5, sin(x)=0.6075, cos(x)=0.7811
x2,6, sin(x)=0.5333, cos(x)=0.8569
x2,7, sin(x)=0.4274, cos(x)=0.9041
x2,8, sin(x)=0.318, cos(x)=0.9223
x2,9, sin(x)=0.2392, cos(x)=0.9718
x3,0, sin(x)=0.1411, cos(x)=0.9989
x3,1, sin(x)=0.0416, cos(x)=0.9991
... Задание 8.2: Таблицы для Sin и Cos (10 точек) ...
x0,0, tabSin(x)=0.000, tabCos(x)=1.000
x0,1, tabSin(x)=0.000, tabCos(x)=0.9997
x0,2, tabSin(x)=0.198, tabCos(x)=0.9854
x0,3, tabSin(x)=0.255, tabCos(x)=0.9423
x0,4, tabSin(x)=0.385, tabCos(x)=0.8244
x0,5, tabSin(x)=0.4723, tabCos(x)=0.8766
x0,6, tabSin(x)=0.6448, tabCos(x)=0.7659
x0,7, tabSin(x)=0.6448, tabCos(x)=0.7646
x0,8, tabSin(x)=0.7879, tabCos(x)=0.6984
x0,9, tabSin(x)=0.8993, tabCos(x)=0.4122
x1,0, tabSin(x)=0.8993, tabCos(x)=0.4161
x1,1, tabSin(x)=0.8612, tabCos(x)=0.5648
x1,2, tabSin(x)=0.804, tabCos(x)=0.6459
x1,3, tabSin(x)=0.7437, tabCos(x)=0.6663
x1,4, tabSin(x)=0.6795, tabCos(x)=0.7374
x1,5, tabSin(x)=0.6075, tabCos(x)=0.7811
x1,6, tabSin(x)=0.5333, tabCos(x)=0.8569
x1,7, tabSin(x)=0.4274, tabCos(x)=0.9041
x1,8, tabSin(x)=0.318, tabCos(x)=0.9223
x1,9, tabSin(x)=0.2392, tabCos(x)=0.9718
x3,0, tabSin(x)=0.1411, tabCos(x)=0.9989
x3,1, tabSin(x)=0.0416, tabCos(x)=0.9991
... Задание 8.3: Сумма квадратов ( $\sin^2 + \cos^2$ ) ...
x0,0, sin2cos2=1.0000

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Program Files\Java\jdk-24\bin\java.exe' -agentlib:jdwp=transport=dt_socket,server=y,address=localhost:62397 "-Xt:showCodeDetailsInExceptionMessages" "-cp" "C:\Users\Korus\AppData\Roaming\Code\User\workspaceStorage\3c17652ca3b4f457a12fe7ac20714\redhat\java\lib\*.jar" "lab-4-2825_44302e5715in"
... Задание 8.3: Сумма квадратов ( $\sin^2 + \cos^2$ ) ...
x0,0, sin2cos2=1.0000
x0,1, sin2cos2=0.9953
x0,2, sin2cos2=0.9957
x0,3, sin2cos2=0.9958
x0,4, sin2cos2=0.9958
x0,5, sin2cos2=0.9957
x0,6, sin2cos2=0.9954
x0,7, sin2cos2=0.9954
x0,8, sin2cos2=0.9951
x0,9, sin2cos2=0.9951
x1,0, sin2cos2=0.9948
x1,1, sin2cos2=0.9945
x1,2, sin2cos2=0.9943
x1,3, sin2cos2=0.9941
x1,4, sin2cos2=0.9939
x1,5, sin2cos2=0.9937
x1,6, sin2cos2=0.9935
x1,7, sin2cos2=0.9934
x1,8, sin2cos2=0.9932
x1,9, sin2cos2=0.9930
x2,0, sin2cos2=0.9928
x2,1, sin2cos2=0.9926
x2,2, sin2cos2=0.9924
x2,3, sin2cos2=0.9922
x2,4, sin2cos2=0.9920
x2,5, sin2cos2=0.9918
x2,6, sin2cos2=0.9916
x2,7, sin2cos2=0.9914
x2,8, sin2cos2=0.9912
x2,9, sin2cos2=0.9910
x3,0, sin2cos2=0.9908
x3,1, sin2cos2=0.9906
... Задание 8.4: Чтение/запись (текстовый файл) ...
Создан текстовый и считанный из файла фрагмент:
x0,0, 0
x0,1, 72
x0,2, 72
x0,3, 72
x0,4, 72
x0,5, 72
x0,6, 72
x0,7, 72
x0,8, 72
x0,9, 72
x1,0, 0
x1,1, 0
x1,2, 0
x1,3, 0
x1,4, 0
x1,5, 0
x1,6, 0
x1,7, 0
x1,8, 0
x1,9, 0
x2,0, 0
x2,1, 0
x2,2, 0
x2,3, 0
x2,4, 0
x2,5, 0
x2,6, 0
x2,7, 0
x2,8, 0
x2,9, 0
x3,0, 0
x3,1, 0
... Задание 8.4: Чтение/запись (текстовый файл) ...
Создан текстовый и считанный из файла фрагмент:
x0,0, 0
x0,1, 0
x0,2, 0
x0,3, 0
x0,4, 0
x0,5, 0
x0,6, 0
x0,7, 0
x0,8, 0
x0,9, 0
x1,0, 0
x1,1, 0
x1,2, 0
x1,3, 0
x1,4, 0
x1,5, 0
x1,6, 0
x1,7, 0
x1,8, 0
x1,9, 0
x2,0, 0
x2,1, 0
x2,2, 0
x2,3, 0
x2,4, 0
x2,5, 0
x2,6, 0
x2,7, 0
x2,8, 0
x2,9, 0
x3,0, 0
x3,1, 0
... Задание 8.5: Тест сортировки (текущий файл) ...
Создан текстовый и считанный из файла фрагмент:
x0,0, 0
x0,1, 0
x0,2, 0
x0,3, 0
x0,4, 0
x0,5, 0
x0,6, 0
x0,7, 0
x0,8, 0
x0,9, 0
x1,0, 0
x1,1, 0
x1,2, 0
x1,3, 0
x1,4, 0
x1,5, 0
x1,6, 0
x1,7, 0
x1,8, 0
x1,9, 0
x2,0, 0
x2,1, 0
x2,2, 0
x2,3, 0
x2,4, 0
x2,5, 0
x2,6, 0
x2,7, 0
x2,8, 0
x2,9, 0
x3,0, 0
x3,1, 0
... Задание 8.5: Тест сортировки (текущий файл) ...
Создан текстовый и считанный из файла фрагмент:
x0,0, 0
x0,1, 0
x0,2, 0
x0,3, 0
x0,4, 0
x0,5, 0
x0,6, 0
x0,7, 0
x0,8, 0
x0,9, 0
x1,0, 0
x1,1, 0
x1,2, 0
x1,3, 0
x1,4, 0
x1,5, 0
x1,6, 0
x1,7, 0
x1,8, 0
x1,9, 0
x2,0, 0
x2,1, 0
x2,2, 0
x2,3, 0
x2,4, 0
x2,5, 0
x2,6, 0
x2,7, 0
x2,8, 0
x2,9, 0
x3,0, 0
x3,1, 0
PS C:\Program Files\Java\jdk-24\bin

```

```
tabulated-exp.txt
1 11
2 0.0 1.0
3 1.0 2.718281828459045
4 2.0 7.38905609893065
5 3.0 20.085536923187668
6 4.0 54.598150033144236
7 5.0 148.4131591025766
8 6.0 403.4287934927351
9 7.0 1096.6331584284585
10 8.0 2980.9579870417283
11 9.0 8103.083927575384
12 10.0 22026.465794806718
13
```

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 *большие* 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).

The screenshot shows two Java code files in an IDE:

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 }
```

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 }
```

```
/* ArrayTableFunction.java */
1 package com;
2
3 import java.util.SortedMap;
4
5 // Xxxx passenger interface, Serializable interface, without using message function + Serializable
6 public class ArrayTableFunction implements TabulatedFunction {
7     private FunctionPoint[] points;
8
9     private static final long serialVersionUID = 1L; // See javadoc
10
11     // ArrayTableFunction (0)
12     public ArrayTableFunction(double leftX, double rightX, int pointsCount) {
13         if (leftX >= rightX || pointsCount < 2) {
14             throw new IllegalArgumentException("Invalid arguments for function creation: leftX >= rightX or pointsCount < 2");
15         }
16
17         this.points = new FunctionPoint[pointsCount];
18         double step = rightX - leftX / (pointsCount - 1);
19         for (int i = 0; i < pointsCount; ++i) {
20             points[i] = new FunctionPoint(leftX + i * step, y(0));
21         }
22     }
23
24     // ArrayTableFunction (2)
25     public ArrayTableFunction(double leftX, double rightX, double[] values) {
26         if (values.length < 2) {
27             throw new IllegalArgumentException("Function must have at least 2 points");
28         }
29
30         // Parameters are unparametrized
31         for (int i = 0; i < values.length - 1; ++i) {
32             if (values[i] > values[i + 1]) {
33                 throw new IllegalArgumentException("Values do not define a continuous increasing vector (x[i] > x[i+1])");
34             }
35             if (values[i].getX() > values[i + 1].getX()) {
36                 throw new IllegalArgumentException("Points are not strictly sorted by X");
37             }
38         }
39
40         // Replace computation and synchronization
41         this.points = new FunctionPoint[values.length];
42         for (int i = 0; i < values.length; ++i) {
43             this.points[i] = new FunctionPoint(values[i]);
44         }
45     }
46
47     public double getLeftmostX() {
48         return points[0].getX();
49     }
50
51     public double getRightmostX() {
52         return points[points.length - 1].getX();
53     }
54
55     public double getLeftmostY() {
56         if (x < getLeftmostX() || x > getRightmostX()) {
57             return Double.NaN;
58         }
59
60         for (int i = 0; i < points.length - 1; ++i) {
61             if (points[i].getX() < x & x < points[i + 1].getX()) {
62                 double xi = points[i].getY();
63                 double yi = points[i + 1].getY();
64
65                 // Linear interpolation
66                 return yi + (y2 - y1) * (x - xi) / (x2 - xi);
67             }
68         }
69         return Double.NaN;
70     }
71
72     public int getPointsCount() {
73         return points.length;
74     }
75
76     private void checkIndex(int index) {
77         if (index < 0 || index > points.length) {
78             throw new FunctionPointIndexOutOfBoundsException("Index " + index + " is out of bounds [0, " + (points.length - 1) + "]");
79         }
80     }
81
82     public FunctionPoint getPoint(int index) {
83         checkIndex(index);
84         return new FunctionPoint(points[index]); // Noncopyable return
85     }
86
87     public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException {
88         checkIndex(index);
89         double x = point.getX();
90         double y1 = points[index].getY();
91         double y2 = points[index + 1].getY();
92         double xi = point.getX();
93         double yi = point.getY();
94
95         // Linear interpolation
96         return yi + (y2 - y1) * (x - xi) / (x2 - xi);
97     }
98
99     public double getPointX(int index) {
100         checkIndex(index);
101         return points[index].getX();
102     }
103
104     public void setPointX(int index, double x) throws InappropriateFunctionPointException {
105         checkIndex(index);
106         double leftBound = (index > 0) ? points[index - 1].getX() : Double.NEGATIVE_INFINITY;
107         double rightBound = (index < points.length - 1) ? points[index + 1].getX() : Double.POSITIVE_INFINITY;
108
109         if (newX < leftBound || newX > rightBound) {
110             throw new InappropriateFunctionPointException("New point's X coordinate is out of the allowed interval (" + leftBound + ", " + rightBound + ")");
111         }
112         points[index] = new FunctionPoint(point);
113     }
114
115     public double getPointY(int index) {
116         checkIndex(index);
117         return points[index].getY();
118     }
119
120     public void setPointY(int index, double y) {
121         checkIndex(index);
122         points[index].setY(y);
123     }
124
125     public void deletePoint(int index) {
126         checkIndex(index);
127         if (points.length < 3) {
128             throw new InappropriateFunctionPointException("Function must have at least 2 points");
129         }
130         points[index] = null;
131     }
132
133 }
```



```
/* ArrayTableFunctionTest.java */
1 package com;
2
3 import org.junit.Test;
4
5 import static org.junit.Assert.*;
6
7 // Xxxx passenger interface, Serializable interface, without using message function + Serializable
8 public class ArrayTableFunctionTest {
9     @Test
10     public void testGetLeftmostX() {
11         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
12         assertEquals(0.0, f.getLeftmostX(), 0.001);
13     }
14
15     @Test
16     public void testGetRightmostX() {
17         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
18         assertEquals(1.0, f.getRightmostX(), 0.001);
19     }
20
21     @Test
22     public void testGetLeftmostY() {
23         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
24         assertEquals(0.0, f.getLeftmostY(), 0.001);
25     }
26
27     @Test
28     public void testGetPoint() {
29         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
30         FunctionPoint p = f.getPoint(2);
31         assertEquals(0.5, p.getX(), 0.001);
32         assertEquals(0.5, p.getY(), 0.001);
33     }
34
35     @Test
36     public void testSetPoint() {
37         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
38         f.setPoint(2, new FunctionPoint(0.5, 1.0));
39         FunctionPoint p = f.getPoint(2);
40         assertEquals(0.5, p.getX(), 0.001);
41         assertEquals(1.0, p.getY(), 0.001);
42     }
43
44     @Test
45     public void testSetPointX() {
46         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
47         f.setPointX(2, 0.5);
48         FunctionPoint p = f.getPoint(2);
49         assertEquals(0.5, p.getX(), 0.001);
50         assertEquals(0.5, p.getY(), 0.001);
51     }
52
53     @Test
54     public void testSetPointY() {
55         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
56         f.setPointY(2, 1.0);
57         FunctionPoint p = f.getPoint(2);
58         assertEquals(0.5, p.getX(), 0.001);
59         assertEquals(1.0, p.getY(), 0.001);
60     }
61
62     @Test
63     public void testDeletePoint() {
64         ArrayTableFunction f = new ArrayTableFunction(0, 1, 5);
65         f.deletePoint(2);
66         FunctionPoint p = f.getPoint(2);
67         assertNull(p);
68     }
69 }
```

```

138     }
139 
140     public double getPointX(int index) {
141         checkIndex(index);
142         if (index < 0 || index > points.length - 1)
143             throw new IndexOutOfBoundsException("Index " + index + " is out of bounds");
144         return points[index].getX();
145     }
146 
147     public void setPointX(int index, double x) throws InappropriateFunctionPointException {
148         checkIndex(index);
149         if (index < 0 || index > points.length - 1)
150             throw new IndexOutOfBoundsException("Index " + index + " is out of bounds");
151         if (x < Double.NEGATIVE_INFINITY || x > Double.POSITIVE_INFINITY)
152             throw new InappropriateFunctionPointException("New x coordinate is out of the allowed interval (" + leftBound + ", " + rightBound + ")");
153         points[index].setX(x);
154     }
155 
156     public double getPointY(int index) {
157         checkIndex(index);
158         return points[index].getY();
159     }
160 
161     public void setPointY(int index, double y) {
162         checkIndex(index);
163         points[index].setY(y);
164     }
165 
166     public void deletePoint(int index) {
167         checkIndex(index);
168         if (points.length < 3) {
169             throw new IllegalStateException("Cannot delete point: function must have at least 2 points remaining.");
170         }
171         FunctionPoint[] newPoints = new FunctionPoint[points.length - 1];
172         System.arraycopy(points, 0, newPoints, 0, index);
173         System.arraycopy(points, index + 1, newPoints, index, points.length - index - 1);
174         points = newPoints;
175     }
176 
177     public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException {
178         if (points.length == 0)
179             insertIndex(0);
180         while (true) {
181             if (points.length == points.length + 1)
182                 insertIndex();
183             if (insertIndex < points.length && points[insertIndex].getX() == point.getX())
184                 throw new InappropriateFunctionPointException("Point with x=" + point.getX() + " already exists.");
185             if (insertIndex < points.length && points[insertIndex].getX() > point.getX())
186                 break;
187         }
188         FunctionPoint[] newPoints = new FunctionPoint[points.length + 1];
189         System.arraycopy(points, 0, newPoints, 0, insertIndex);
190         newPoints[insertIndex] = new FunctionPoint(point); // replace
191         System.arraycopy(points, insertIndex, newPoints, insertIndex + 1, points.length - insertIndex);
192         points = newPoints;
193     }
194 }

```

```

1  package com.jacob.bates.function;
2 
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.function.Function;
6  import java.util.function.ToDoubleFunction;
7  import java.util.function.UnaryOperator;
8  import java.util.stream.Collectors;
9  import java.util.stream.IntStream;
10 
11  /**
12   * Function class that implements the Function interface.
13   */
14  public class Function implements Function<Double, Double>, ToDoubleFunction, Serializable {
15 
16     private static final long serialVersionUID = 1L;
17 
18     // 1. Parameter count or max arity
19     protected FunctionMode mode;
20     protected FunctionMode point;
21     protected FunctionMode[] args;
22     protected FunctionMode next;
23 
24     // 2. Function body or externalizable
25     private FunctionBody head;
26 
27     // 3. SerialVersionUID or externalizable
28     private static final long serialVersionUID = 1L;
29 
30     // 4. Function arguments (functions are externalizable)
31     protected List<Function> arguments;
32 
33     private int count;
34     private FunctionMode[] args;
35     private FunctionMode head;
36     private FunctionMode next;
37 
38     // ... implementing the API ...
39 
40     public Function(double left, double right, int pointsCount) {
41         if (left > right || pointsCount < 2)
42             throw new IllegalArgumentException("Invalid arguments for function creation");
43         this.count = pointsCount;
44         this.args = args;
45         this.head = head;
46         this.next = next;
47 
48         double step = (right - left) / (values.length - 1);
49         for (int i = 0; i < pointsCount; i++) {
50             addFunctionPoint(left + i * step, values[i]);
51         }
52         argumentsCount = arguments.size();
53         arguments = arguments.stream().map(Function::values).collect(Collectors.toList());
54         headCount = headCount();
55         head = head();
56         nextCount = nextCount();
57         next = next();
58     }
59 
60     // ... implementing the API ...
61 
62     public LineableFunction<Double, Double> values() {
63         if (args.length != 0)
64             throw new IllegalArgumentException("Invalid arguments for function creation");
65         this.values = values;
66         this.head = head();
67         this.next = next();
68         this.args = args;
69 
70         double step = (right - left) / (values.length - 1);
71         for (int i = 0; i < values.length; i++) {
72             addFunctionPoint(left + i * step, values[i]);
73         }
74     }
75 
76     // Head constructor (values is not null)
77     public Function(double left, double right, FunctionMode[] points) {
78         if (left > right || points.length < 2)
79             throw new IllegalArgumentException("Function must have at least 2 points");
80         this.values = values;
81         this.head = head();
82         this.next = next();
83         this.args = args;
84 
85         for (int i = 0; i < points.length; i++) {
86             if (i < 2)
87                 addFunctionPoint(left + i * step, points[i].getX());
88             else if (i < 4)
89                 addFunctionPoint(left + i * step, points[i].getY());
90             else
91                 addFunctionPoint(left + i * step, points[i].getZ());
92         }
93     }
94 
95     // Head constructor (values is null)
96     public Function(double left, double right, FunctionMode[] points, FunctionMode[] values) {
97         if (left > right || points.length < 2)
98             throw new IllegalArgumentException("Function must have at least 2 points");
99         this.values = values;
100        this.head = head();
101        this.next = next();
102        this.args = args;
103 
104        for (int i = 0; i < points.length; i++) {
105            if (i < 2)
106                addFunctionPoint(left + i * step, points[i].getX());
107            else if (i < 4)
108                addFunctionPoint(left + i * step, points[i].getY());
109            else
110                addFunctionPoint(left + i * step, points[i].getZ());
111        }
112    }
113 
114    // Head constructor (values is null)
115    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args) {
116        if (left > right || points.length < 2)
117            throw new IllegalArgumentException("Function must have at least 2 points");
118        this.values = values;
119        this.head = head();
120        this.next = next();
121        this.args = args;
122 
123        for (int i = 0; i < points.length; i++) {
124            if (i < 2)
125                addFunctionPoint(left + i * step, points[i].getX());
126            else if (i < 4)
127                addFunctionPoint(left + i * step, points[i].getY());
128            else
129                addFunctionPoint(left + i * step, points[i].getZ());
130        }
131    }
132 
133    // Head constructor (values is null)
134    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head) {
135        if (left > right || points.length < 2)
136            throw new IllegalArgumentException("Function must have at least 2 points");
137        this.values = values;
138        this.head = head();
139        this.next = next();
140        this.args = args;
141 
142        for (int i = 0; i < points.length; i++) {
143            if (i < 2)
144                addFunctionPoint(left + i * step, points[i].getX());
145            else if (i < 4)
146                addFunctionPoint(left + i * step, points[i].getY());
147            else
148                addFunctionPoint(left + i * step, points[i].getZ());
149        }
150    }
151 
152    // Head constructor (values is null)
153    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next) {
154        if (left > right || points.length < 2)
155            throw new IllegalArgumentException("Function must have at least 2 points");
156        this.values = values;
157        this.head = head();
158        this.next = next();
159        this.args = args;
160 
161        for (int i = 0; i < points.length; i++) {
162            if (i < 2)
163                addFunctionPoint(left + i * step, points[i].getX());
164            else if (i < 4)
165                addFunctionPoint(left + i * step, points[i].getY());
166            else
167                addFunctionPoint(left + i * step, points[i].getZ());
168        }
169    }
170 
171    // Head constructor (values is null)
172    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode) {
173        if (left > right || points.length < 2)
174            throw new IllegalArgumentException("Function must have at least 2 points");
175        this.values = values;
176        this.head = head();
177        this.next = next();
178        this.args = args;
179        this.mode = mode;
180 
181        for (int i = 0; i < points.length; i++) {
182            if (i < 2)
183                addFunctionPoint(left + i * step, points[i].getX());
184            else if (i < 4)
185                addFunctionPoint(left + i * step, points[i].getY());
186            else
187                addFunctionPoint(left + i * step, points[i].getZ());
188        }
189    }
190 
191    // Head constructor (values is null)
192    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2) {
193        if (left > right || points.length < 2)
194            throw new IllegalArgumentException("Function must have at least 2 points");
195        this.values = values;
196        this.head = head();
197        this.next = next();
198        this.args = args;
199        this.mode = mode;
200        this.mode2 = mode2;
201 
202        for (int i = 0; i < points.length; i++) {
203            if (i < 2)
204                addFunctionPoint(left + i * step, points[i].getX());
205            else if (i < 4)
206                addFunctionPoint(left + i * step, points[i].getY());
207            else
208                addFunctionPoint(left + i * step, points[i].getZ());
209        }
210    }
211 
212    // Head constructor (values is null)
213    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3) {
214        if (left > right || points.length < 2)
215            throw new IllegalArgumentException("Function must have at least 2 points");
216        this.values = values;
217        this.head = head();
218        this.next = next();
219        this.args = args;
220        this.mode = mode;
221        this.mode2 = mode2;
222        this.mode3 = mode3;
223 
224        for (int i = 0; i < points.length; i++) {
225            if (i < 2)
226                addFunctionPoint(left + i * step, points[i].getX());
227            else if (i < 4)
228                addFunctionPoint(left + i * step, points[i].getY());
229            else
230                addFunctionPoint(left + i * step, points[i].getZ());
231        }
232    }
233 
234    // Head constructor (values is null)
235    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4) {
236        if (left > right || points.length < 2)
237            throw new IllegalArgumentException("Function must have at least 2 points");
238        this.values = values;
239        this.head = head();
240        this.next = next();
241        this.args = args;
242        this.mode = mode;
243        this.mode2 = mode2;
244        this.mode3 = mode3;
245        this.mode4 = mode4;
246 
247        for (int i = 0; i < points.length; i++) {
248            if (i < 2)
249                addFunctionPoint(left + i * step, points[i].getX());
250            else if (i < 4)
251                addFunctionPoint(left + i * step, points[i].getY());
252            else
253                addFunctionPoint(left + i * step, points[i].getZ());
254        }
255    }
256 
257    // Head constructor (values is null)
258    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5) {
259        if (left > right || points.length < 2)
260            throw new IllegalArgumentException("Function must have at least 2 points");
261        this.values = values;
262        this.head = head();
263        this.next = next();
264        this.args = args;
265        this.mode = mode;
266        this.mode2 = mode2;
267        this.mode3 = mode3;
268        this.mode4 = mode4;
269        this.mode5 = mode5;
270 
271        for (int i = 0; i < points.length; i++) {
272            if (i < 2)
273                addFunctionPoint(left + i * step, points[i].getX());
274            else if (i < 4)
275                addFunctionPoint(left + i * step, points[i].getY());
276            else
277                addFunctionPoint(left + i * step, points[i].getZ());
278        }
279    }
280 
281    // Head constructor (values is null)
282    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6) {
283        if (left > right || points.length < 2)
284            throw new IllegalArgumentException("Function must have at least 2 points");
285        this.values = values;
286        this.head = head();
287        this.next = next();
288        this.args = args;
289        this.mode = mode;
290        this.mode2 = mode2;
291        this.mode3 = mode3;
292        this.mode4 = mode4;
293        this.mode5 = mode5;
294        this.mode6 = mode6;
295 
296        for (int i = 0; i < points.length; i++) {
297            if (i < 2)
298                addFunctionPoint(left + i * step, points[i].getX());
299            else if (i < 4)
300                addFunctionPoint(left + i * step, points[i].getY());
301            else
302                addFunctionPoint(left + i * step, points[i].getZ());
303        }
304    }
305 
306    // Head constructor (values is null)
307    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7) {
308        if (left > right || points.length < 2)
309            throw new IllegalArgumentException("Function must have at least 2 points");
310        this.values = values;
311        this.head = head();
312        this.next = next();
313        this.args = args;
314        this.mode = mode;
315        this.mode2 = mode2;
316        this.mode3 = mode3;
317        this.mode4 = mode4;
318        this.mode5 = mode5;
319        this.mode6 = mode6;
320        this.mode7 = mode7;
321 
322        for (int i = 0; i < points.length; i++) {
323            if (i < 2)
324                addFunctionPoint(left + i * step, points[i].getX());
325            else if (i < 4)
326                addFunctionPoint(left + i * step, points[i].getY());
327            else
328                addFunctionPoint(left + i * step, points[i].getZ());
329        }
330    }
331 
332    // Head constructor (values is null)
333    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8) {
334        if (left > right || points.length < 2)
335            throw new IllegalArgumentException("Function must have at least 2 points");
336        this.values = values;
337        this.head = head();
338        this.next = next();
339        this.args = args;
340        this.mode = mode;
341        this.mode2 = mode2;
342        this.mode3 = mode3;
343        this.mode4 = mode4;
344        this.mode5 = mode5;
345        this.mode6 = mode6;
346        this.mode7 = mode7;
347        this.mode8 = mode8;
348 
349        for (int i = 0; i < points.length; i++) {
350            if (i < 2)
351                addFunctionPoint(left + i * step, points[i].getX());
352            else if (i < 4)
353                addFunctionPoint(left + i * step, points[i].getY());
354            else
355                addFunctionPoint(left + i * step, points[i].getZ());
356        }
357    }
358 
359    // Head constructor (values is null)
360    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9) {
361        if (left > right || points.length < 2)
362            throw new IllegalArgumentException("Function must have at least 2 points");
363        this.values = values;
364        this.head = head();
365        this.next = next();
366        this.args = args;
367        this.mode = mode;
368        this.mode2 = mode2;
369        this.mode3 = mode3;
370        this.mode4 = mode4;
371        this.mode5 = mode5;
372        this.mode6 = mode6;
373        this.mode7 = mode7;
374        this.mode8 = mode8;
375        this.mode9 = mode9;
376 
377        for (int i = 0; i < points.length; i++) {
378            if (i < 2)
379                addFunctionPoint(left + i * step, points[i].getX());
380            else if (i < 4)
381                addFunctionPoint(left + i * step, points[i].getY());
382            else
383                addFunctionPoint(left + i * step, points[i].getZ());
384        }
385    }
386 
387    // Head constructor (values is null)
388    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10) {
389        if (left > right || points.length < 2)
390            throw new IllegalArgumentException("Function must have at least 2 points");
391        this.values = values;
392        this.head = head();
393        this.next = next();
394        this.args = args;
395        this.mode = mode;
396        this.mode2 = mode2;
397        this.mode3 = mode3;
398        this.mode4 = mode4;
399        this.mode5 = mode5;
400        this.mode6 = mode6;
401        this.mode7 = mode7;
402        this.mode8 = mode8;
403        this.mode9 = mode9;
404        this.mode10 = mode10;
405 
406        for (int i = 0; i < points.length; i++) {
407            if (i < 2)
408                addFunctionPoint(left + i * step, points[i].getX());
409            else if (i < 4)
410                addFunctionPoint(left + i * step, points[i].getY());
411            else
412                addFunctionPoint(left + i * step, points[i].getZ());
413        }
414    }
415 
416    // Head constructor (values is null)
417    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11) {
418        if (left > right || points.length < 2)
419            throw new IllegalArgumentException("Function must have at least 2 points");
420        this.values = values;
421        this.head = head();
422        this.next = next();
423        this.args = args;
424        this.mode = mode;
425        this.mode2 = mode2;
426        this.mode3 = mode3;
427        this.mode4 = mode4;
428        this.mode5 = mode5;
429        this.mode6 = mode6;
430        this.mode7 = mode7;
431        this.mode8 = mode8;
432        this.mode9 = mode9;
433        this.mode10 = mode10;
434        this.mode11 = mode11;
435 
436        for (int i = 0; i < points.length; i++) {
437            if (i < 2)
438                addFunctionPoint(left + i * step, points[i].getX());
439            else if (i < 4)
440                addFunctionPoint(left + i * step, points[i].getY());
441            else
442                addFunctionPoint(left + i * step, points[i].getZ());
443        }
444    }
445 
446    // Head constructor (values is null)
447    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12) {
448        if (left > right || points.length < 2)
449            throw new IllegalArgumentException("Function must have at least 2 points");
450        this.values = values;
451        this.head = head();
452        this.next = next();
453        this.args = args;
454        this.mode = mode;
455        this.mode2 = mode2;
456        this.mode3 = mode3;
457        this.mode4 = mode4;
458        this.mode5 = mode5;
459        this.mode6 = mode6;
460        this.mode7 = mode7;
461        this.mode8 = mode8;
462        this.mode9 = mode9;
463        this.mode10 = mode10;
464        this.mode11 = mode11;
465        this.mode12 = mode12;
466 
467        for (int i = 0; i < points.length; i++) {
468            if (i < 2)
469                addFunctionPoint(left + i * step, points[i].getX());
470            else if (i < 4)
471                addFunctionPoint(left + i * step, points[i].getY());
472            else
473                addFunctionPoint(left + i * step, points[i].getZ());
474        }
475    }
476 
477    // Head constructor (values is null)
478    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13) {
479        if (left > right || points.length < 2)
480            throw new IllegalArgumentException("Function must have at least 2 points");
481        this.values = values;
482        this.head = head();
483        this.next = next();
484        this.args = args;
485        this.mode = mode;
486        this.mode2 = mode2;
487        this.mode3 = mode3;
488        this.mode4 = mode4;
489        this.mode5 = mode5;
490        this.mode6 = mode6;
491        this.mode7 = mode7;
492        this.mode8 = mode8;
493        this.mode9 = mode9;
494        this.mode10 = mode10;
495        this.mode11 = mode11;
496        this.mode12 = mode12;
497        this.mode13 = mode13;
498 
499        for (int i = 0; i < points.length; i++) {
500            if (i < 2)
501                addFunctionPoint(left + i * step, points[i].getX());
502            else if (i < 4)
503                addFunctionPoint(left + i * step, points[i].getY());
504            else
505                addFunctionPoint(left + i * step, points[i].getZ());
506        }
507    }
508 
509    // Head constructor (values is null)
510    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14) {
511        if (left > right || points.length < 2)
512            throw new IllegalArgumentException("Function must have at least 2 points");
513        this.values = values;
514        this.head = head();
515        this.next = next();
516        this.args = args;
517        this.mode = mode;
518        this.mode2 = mode2;
519        this.mode3 = mode3;
520        this.mode4 = mode4;
521        this.mode5 = mode5;
522        this.mode6 = mode6;
523        this.mode7 = mode7;
524        this.mode8 = mode8;
525        this.mode9 = mode9;
526        this.mode10 = mode10;
527        this.mode11 = mode11;
528        this.mode12 = mode12;
529        this.mode13 = mode13;
530        this.mode14 = mode14;
531 
532        for (int i = 0; i < points.length; i++) {
533            if (i < 2)
534                addFunctionPoint(left + i * step, points[i].getX());
535            else if (i < 4)
536                addFunctionPoint(left + i * step, points[i].getY());
537            else
538                addFunctionPoint(left + i * step, points[i].getZ());
539        }
540    }
541 
542    // Head constructor (values is null)
543    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14, FunctionMode[] mode15) {
544        if (left > right || points.length < 2)
545            throw new IllegalArgumentException("Function must have at least 2 points");
546        this.values = values;
547        this.head = head();
548        this.next = next();
549        this.args = args;
550        this.mode = mode;
551        this.mode2 = mode2;
552        this.mode3 = mode3;
553        this.mode4 = mode4;
554        this.mode5 = mode5;
555        this.mode6 = mode6;
556        this.mode7 = mode7;
557        this.mode8 = mode8;
558        this.mode9 = mode9;
559        this.mode10 = mode10;
560        this.mode11 = mode11;
561        this.mode12 = mode12;
562        this.mode13 = mode13;
563        this.mode14 = mode14;
564        this.mode15 = mode15;
565 
566        for (int i = 0; i < points.length; i++) {
567            if (i < 2)
568                addFunctionPoint(left + i * step, points[i].getX());
569            else if (i < 4)
570                addFunctionPoint(left + i * step, points[i].getY());
571            else
572                addFunctionPoint(left + i * step, points[i].getZ());
573        }
574    }
575 
576    // Head constructor (values is null)
577    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14, FunctionMode[] mode15, FunctionMode[] mode16) {
578        if (left > right || points.length < 2)
579            throw new IllegalArgumentException("Function must have at least 2 points");
580        this.values = values;
581        this.head = head();
582        this.next = next();
583        this.args = args;
584        this.mode = mode;
585        this.mode2 = mode2;
586        this.mode3 = mode3;
587        this.mode4 = mode4;
588        this.mode5 = mode5;
589        this.mode6 = mode6;
590        this.mode7 = mode7;
591        this.mode8 = mode8;
592        this.mode9 = mode9;
593        this.mode10 = mode10;
594        this.mode11 = mode11;
595        this.mode12 = mode12;
596        this.mode13 = mode13;
597        this.mode14 = mode14;
598        this.mode15 = mode15;
599        this.mode16 = mode16;
600 
601        for (int i = 0; i < points.length; i++) {
602            if (i < 2)
603                addFunctionPoint(left + i * step, points[i].getX());
604            else if (i < 4)
605                addFunctionPoint(left + i * step, points[i].getY());
606            else
607                addFunctionPoint(left + i * step, points[i].getZ());
608        }
609    }
610 
611    // Head constructor (values is null)
612    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14, FunctionMode[] mode15, FunctionMode[] mode16, FunctionMode[] mode17) {
613        if (left > right || points.length < 2)
614            throw new IllegalArgumentException("Function must have at least 2 points");
615        this.values = values;
616        this.head = head();
617        this.next = next();
618        this.args = args;
619        this.mode = mode;
620        this.mode2 = mode2;
621        this.mode3 = mode3;
622        this.mode4 = mode4;
623        this.mode5 = mode5;
624        this.mode6 = mode6;
625        this.mode7 = mode7;
626        this.mode8 = mode8;
627        this.mode9 = mode9;
628        this.mode10 = mode10;
629        this.mode11 = mode11;
630        this.mode12 = mode12;
631        this.mode13 = mode13;
632        this.mode14 = mode14;
633        this.mode15 = mode15;
634        this.mode16 = mode16;
635        this.mode17 = mode17;
636 
637        for (int i = 0; i < points.length; i++) {
638            if (i < 2)
639                addFunctionPoint(left + i * step, points[i].getX());
640            else if (i < 4)
641                addFunctionPoint(left + i * step, points[i].getY());
642            else
643                addFunctionPoint(left + i * step, points[i].getZ());
644        }
645    }
646 
647    // Head constructor (values is null)
648    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14, FunctionMode[] mode15, FunctionMode[] mode16, FunctionMode[] mode17, FunctionMode[] mode18) {
649        if (left > right || points.length < 2)
650            throw new IllegalArgumentException("Function must have at least 2 points");
651        this.values = values;
652        this.head = head();
653        this.next = next();
654        this.args = args;
655        this.mode = mode;
656        this.mode2 = mode2;
657        this.mode3 = mode3;
658        this.mode4 = mode4;
659        this.mode5 = mode5;
660        this.mode6 = mode6;
661        this.mode7 = mode7;
662        this.mode8 = mode8;
663        this.mode9 = mode9;
664        this.mode10 = mode10;
665        this.mode11 = mode11;
666        this.mode12 = mode12;
667        this.mode13 = mode13;
668        this.mode14 = mode14;
669        this.mode15 = mode15;
670        this.mode16 = mode16;
671        this.mode17 = mode17;
672        this.mode18 = mode18;
673 
674        for (int i = 0; i < points.length; i++) {
675            if (i < 2)
676                addFunctionPoint(left + i * step, points[i].getX());
677            else if (i < 4)
678                addFunctionPoint(left + i * step, points[i].getY());
679            else
680                addFunctionPoint(left + i * step, points[i].getZ());
681        }
682    }
683 
684    // Head constructor (values is null)
685    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14, FunctionMode[] mode15, FunctionMode[] mode16, FunctionMode[] mode17, FunctionMode[] mode18, FunctionMode[] mode19) {
686        if (left > right || points.length < 2)
687            throw new IllegalArgumentException("Function must have at least 2 points");
688        this.values = values;
689        this.head = head();
690        this.next = next();
691        this.args = args;
692        this.mode = mode;
693        this.mode2 = mode2;
694        this.mode3 = mode3;
695        this.mode4 = mode4;
696        this.mode5 = mode5;
697        this.mode6 = mode6;
698        this.mode7 = mode7;
699        this.mode8 = mode8;
700        this.mode9 = mode9;
701        this.mode10 = mode10;
702        this.mode11 = mode11;
703        this.mode12 = mode12;
704        this.mode13 = mode13;
705        this.mode14 = mode14;
706        this.mode15 = mode15;
707        this.mode16 = mode16;
708        this.mode17 = mode17;
709        this.mode18 = mode18;
710        this.mode19 = mode19;
711 
712        for (int i = 0; i < points.length; i++) {
713            if (i < 2)
714                addFunctionPoint(left + i * step, points[i].getX());
715            else if (i < 4)
716                addFunctionPoint(left + i * step, points[i].getY());
717            else
718                addFunctionPoint(left + i * step, points[i].getZ());
719        }
720    }
721 
722    // Head constructor (values is null)
723    public Function(double left, double right, FunctionMode[] points, FunctionMode[] values, FunctionMode[] args, FunctionMode[] head, FunctionMode[] next, FunctionMode[] mode, FunctionMode[] mode2, FunctionMode[] mode3, FunctionMode[] mode4, FunctionMode[] mode5, FunctionMode[] mode6, FunctionMode[] mode7, FunctionMode[] mode8, FunctionMode[] mode9, FunctionMode[] mode10, FunctionMode[] mode11, FunctionMode[] mode12, FunctionMode[] mode13, FunctionMode[] mode14, FunctionMode[] mode15, FunctionMode[] mode16, FunctionMode[] mode17, FunctionMode[] mode18, FunctionMode[] mode19, FunctionMode[] mode20) {
724        if (left > right || points.length < 2)
725            throw new IllegalArgumentException("Function must have at least 2 points");
726        this.values = values;
727        this.head = head();
728        this.next = next();
729        this.args = args;
730        this.mode = mode;
731        this.mode2 = mode2;
732        this.mode3 = mode3;
733        this.mode4 = mode4;
734        this.mode5 = mode5;
735        this.mode6 = mode6;
736        this.mode7 = mode7;
737        this.mode8 = mode8;
738        this.mode9 = mode9;
739        this.mode10 = mode10;
740        this.mode11 = mode11;
741        this.mode
```

```
on function > immediately(function) > immediately(function) > function

9  public class FunctionIntervalIntersection implements IntervalIntersection, Externalizable {
10
11     public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
12
13         if (currentPoint == null) {
14             currentPoint = new Node();
15             currentPoint.setX((x + y) / 2);
16             currentPoint.setY((y - x) / 2);
17         }
18
19         double y1 = currentPoint.getY();
20
21         double y2 = currentPoint.getY();
22         double y3 = currentPoint.getY();
23
24         double y4 = (y - x) * (x - z) / (y - z);
25
26         return y4;
27     }
28
29     current = current.next;
30
31     if (x == getRightBoundaryIndex()) {
32         return head.prev.point.get();
33     }
34
35     return Double.NaN;
36
37 }
38
39 public int getLeftIndex() {
40
41     return count;
42 }
43
44 private void checkIndex(int index) {
45
46     if (index < 0 || index > count) {
47         throw new FunctionIntervalIntersectionIndexOutOfBoundsException("index " + index + " is out of bounds for count " + count);
48     }
49 }
50
51 public FunctionPoint getPoint(int index) {
52
53     checkIndex(index);
54
55     return new FunctionPoint(getLeftBoundaryIndex(), node.prev.point.get());
56 }
57
58 public void addPoint(int index, FunctionPoint point) throws ImproperFunctionPointException {
59
60     checkIndex(index);
61
62     FunctionPoint node = getRightBoundaryIndex();
63
64     double leftbound = node.next == head ? Double.NEGATIVE_INFINITY : node.prev.point.get();
65     double rightbound = node.next == head ? Double.POSITIVE_INFINITY : node.next.point.get();
66
67     if (rightbound <= leftbound || node == rightbound) {
68         throw new ImproperFunctionPointException(message); // "coordinate is not of the allowed interval";
69     }
70
71     node.point = new FunctionPoint(point);
72
73 }
74
75 public double getPoint(int index) {
76
77     checkIndex(index);
78
79     return new FunctionPoint(index, node.prev.point.get());
80 }
81
82 public void setPoint(int index, double x) throws ImproperFunctionPointException {
83
84     checkIndex(index);
85
86     FunctionPoint node = getRightBoundaryIndex();
87
88     double leftbound = node.next == head ? Double.NEGATIVE_INFINITY : node.prev.point.get();
89     double rightbound = node.next == head ? Double.POSITIVE_INFINITY : node.next.point.get();
90
91     if (rightbound <= leftbound || node == rightbound) {
92         throw new ImproperFunctionPointException(message); // "coordinate is not of the allowed interval";
93     }
94
95     node.point.setX(x);
96
97 }
98
99 public void getPoint(int index) {
100
101     checkIndex(index);
102
103     return getRightBoundaryIndex(index).point.get();
104 }
105
106 public void setPoint(int index, double y) {
107
108     checkIndex(index);
109
110     getRightBoundaryIndex(index).point.setY(y);
111 }
112
113 public void determine(int index) {
114
115     checkIndex(index);
116
117     if (current == null) {
118         throw new IllegalStateException("cannot define point, function must have at least 2 points");
119     }
120
121     determineBoundaryIndex();
122 }
123
124 public void addPoint(FunctionPoint point) throws ImproperFunctionPointException {
125
126     FunctionNode current = head.next;
127
128     if (current == null) {
129         throw new IllegalStateException("cannot add point, function must have at least 2 points");
130     }
131
132     determineBoundaryIndex();
133
134     current = current.next;
135
136     if (current == null) {
137         throw new IllegalStateException("cannot add point, function must have at least 2 points");
138     }
139
140     current = current.next;
141
142     if (current == null) {
143         throw new IllegalStateException("cannot add point, function must have at least 2 points");
144     }
145
146     current = current.next;
147
148     if (current == null) {
149         throw new IllegalStateException("cannot add point, function must have at least 2 points");
150     }
151
152     current = current.next;
153
154     if (current == null) {
155         throw new IllegalStateException("cannot add point, function must have at least 2 points");
156     }
157
158     current = current.next;
159
160     if (current == null) {
161         throw new IllegalStateException("cannot add point, function must have at least 2 points");
162     }
163
164     current = current.next;
165
166     if (current == null) {
167         throw new IllegalStateException("cannot add point, function must have at least 2 points");
168     }
169
170     current = current.next;
171
172     if (current == null) {
173         throw new IllegalStateException("cannot add point, function must have at least 2 points");
174     }
175
176     current = current.next;
177
178     if (current == null) {
179         throw new IllegalStateException("cannot add point, function must have at least 2 points");
180     }
181
182     current = current.next;
183
184     if (current == null) {
185         throw new IllegalStateException("cannot add point, function must have at least 2 points");
186     }
187
188     current = current.next;
189
190     if (current == null) {
191         throw new IllegalStateException("cannot add point, function must have at least 2 points");
192     }
193
194     current = current.next;
195
196     if (current == null) {
197         throw new IllegalStateException("cannot add point, function must have at least 2 points");
198     }
199
200     current = current.next;
201
202     if (current == null) {
203         throw new IllegalStateException("cannot add point, function must have at least 2 points");
204     }
205
206     current = current.next;
207
208     if (current == null) {
209         throw new IllegalStateException("cannot add point, function must have at least 2 points");
210     }
211
212     current = current.next;
213
214     if (current == null) {
215         throw new IllegalStateException("cannot add point, function must have at least 2 points");
216     }
217
218     current = current.next;
219
220     if (current == null) {
221         throw new IllegalStateException("cannot add point, function must have at least 2 points");
222     }
223
224     current = current.next;
225
226     if (current == null) {
227         throw new IllegalStateException("cannot add point, function must have at least 2 points");
228     }
229
230     current = current.next;
231
232     if (current == null) {
233         throw new IllegalStateException("cannot add point, function must have at least 2 points");
234     }
235
236     current = current.next;
237
238     if (current == null) {
239         throw new IllegalStateException("cannot add point, function must have at least 2 points");
240     }
241
242     current = current.next;
243
244     if (current == null) {
245         throw new IllegalStateException("cannot add point, function must have at least 2 points");
246     }
247
248     current = current.next;
249
250     if (current == null) {
251         throw new IllegalStateException("cannot add point, function must have at least 2 points");
252     }
253
254     current = current.next;
255
256     if (current == null) {
257         throw new IllegalStateException("cannot add point, function must have at least 2 points");
258     }
259
260     current = current.next;
261
262     if (current == null) {
263         throw new IllegalStateException("cannot add point, function must have at least 2 points");
264     }
265
266     current = current.next;
267
268     if (current == null) {
269         throw new IllegalStateException("cannot add point, function must have at least 2 points");
270     }
271
272     current = current.next;
273
274     if (current == null) {
275         throw new IllegalStateException("cannot add point, function must have at least 2 points");
276     }
277
278     current = current.next;
279
280     if (current == null) {
281         throw new IllegalStateException("cannot add point, function must have at least 2 points");
282     }
283
284     current = current.next;
285
286     if (current == null) {
287         throw new IllegalStateException("cannot add point, function must have at least 2 points");
288     }
289
290     current = current.next;
291
292     if (current == null) {
293         throw new IllegalStateException("cannot add point, function must have at least 2 points");
294     }
295
296     current = current.next;
297
298     if (current == null) {
299         throw new IllegalStateException("cannot add point, function must have at least 2 points");
300     }
301
302     current = current.next;
303
304     if (current == null) {
305         throw new IllegalStateException("cannot add point, function must have at least 2 points");
306     }
307
308     current = current.next;
309
310     if (current == null) {
311         throw new IllegalStateException("cannot add point, function must have at least 2 points");
312     }
313
314     current = current.next;
315
316     if (current == null) {
317         throw new IllegalStateException("cannot add point, function must have at least 2 points");
318     }
319
320     current = current.next;
321
322     if (current == null) {
323         throw new IllegalStateException("cannot add point, function must have at least 2 points");
324     }
325
326     current = current.next;
327
328     if (current == null) {
329         throw new IllegalStateException("cannot add point, function must have at least 2 points");
330     }
331
332     current = current.next;
333
334     if (current == null) {
335         throw new IllegalStateException("cannot add point, function must have at least 2 points");
336     }
337
338     current = current.next;
339
340     if (current == null) {
341         throw new IllegalStateException("cannot add point, function must have at least 2 points");
342     }
343
344     current = current.next;
345
346     if (current == null) {
347         throw new IllegalStateException("cannot add point, function must have at least 2 points");
348     }
349
350     current = current.next;
351
352     if (current == null) {
353         throw new IllegalStateException("cannot add point, function must have at least 2 points");
354     }
355
356     current = current.next;
357
358     if (current == null) {
359         throw new IllegalStateException("cannot add point, function must have at least 2 points");
360     }
361
362     current = current.next;
363
364     if (current == null) {
365         throw new IllegalStateException("cannot add point, function must have at least 2 points");
366     }
367
368     current = current.next;
369
370     if (current == null) {
371         throw new IllegalStateException("cannot add point, function must have at least 2 points");
372     }
373
374     current = current.next;
375
376     if (current == null) {
377         throw new IllegalStateException("cannot add point, function must have at least 2 points");
378     }
379
380     current = current.next;
381
382     if (current == null) {
383         throw new IllegalStateException("cannot add point, function must have at least 2 points");
384     }
385
386     current = current.next;
387
388     if (current == null) {
389         throw new IllegalStateException("cannot add point, function must have at least 2 points");
390     }
391
392     current = current.next;
393
394     if (current == null) {
395         throw new IllegalStateException("cannot add point, function must have at least 2 points");
396     }
397
398     current = current.next;
399
399 }
```

```
public void setIndex(int index, double y) {
    checkIndex(index);
    getCoordinates(index).point.sety(y);
}

public void deletePoint(int index) {
    checkIndex(index);
    if (count < 2) {
        throw new IllegalArgumentException("Can't delete point, Function must have at least 2 points");
    }
    deleteIndex(index);
}

public void addFunction(OneDimensionalPoint point) throws inappropriateFunctionException {
    FunctionPoint current = head.next;
    int index = 0;
    do {
        if (current == null || current.point.getx() == point.getx()) {
            indices.add(index);
        }
        current = current.next;
        index++;
    } while (current != null && current.point.getx() == point.getx());
    if (current != null) {
        throw new inappropriateFunctionException("Point with such x already exists");
    }
    increaseCount();
    addFunction(current, y);
    addFunction(index, point);
}

// ... writing methods for iterable (chapter 8) ...

public void writeToFile(Writer writer) throws IOException {
    out.write(count);
    out.write("\n");
    for (int i = 0; i < count; i++) {
        // Increases counter every
        // writeLine
        out.write(indices.get(i));
        out.write("\t");
        out.write(getFunction(i).point);
        current = current.next;
    }
}

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    count = in.readInt();
    if (count < 2) {
        throw new IllegalArgumentException("Can't read function, Function must have at least 2 points");
    }
    head.next = null;
    this.head.next = head;
    this.count = 0;
    for (int i = 0; i < count; i++) {
        OneDimensionalPoint point = (OneDimensionalPoint) in.readObject();
        addFunction(i, point); // Space addition a point
        this.count++;
    }
}
```

```
1 package X;
2
3 import java.util.*;
4 import Functions.*;
5 import Functions.math.*;
6 import java.io.*;
7
8 public class Main {
9
10     public static void main(String[] args) {
11
12         // ---- Вызовем x = 3.14 Sin в Cos ----
13         System.out.println("x = 3.14 Sin в Cos ...");
14         Function sin = new Sin();
15         Function cos = new Cos();
16         for (double x = 0; x <= Math.PI; x += 0.1) {
17             System.out.print(String.format("x=%-3.1f, sin(x)=%.4f, cos(x)=%.4f\n", x, sin.getValue(x), cos.getValue(x)));
18         }
19
20         // ---- Вызовем x = 3.14 TanhSin в Cos (30 градусов) ----
21         System.out.println("x = 3.14 TanhSin в Cos (30 градусов) ...");
22         TabulatedFunction tabCos = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 10);
23         TabulatedFunction tabSin = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 10);
24         for (double x = 0; x <= Math.PI; x += 0.1) {
25             System.out.print(String.format("x=%-3.1f, tabSin(x)=%.4f, tabCos(x)=%.4f\n", x, tabSin.getValue(x), tabCos.getValue(x)));
26         }
27
28         // ---- Вызовем x = 3.14 Cosec косинус ----
29         System.out.println("x = 3.14 Cosec косинус ...");
30         Function cosec = Functions.powerTabCos(power: 2);
31         Function cos = Functions.sin();
32         Function suseg = Functions.sum(sineg, cosec);
33         for (double x = 0; x <= Math.PI; x += 0.1) {
34             System.out.print(String.format("x=%-3.1f, tabCosec(x)=%.4f\n", x, suseg.getValue(x)));
35         }
36
37         // ---- Вызовем x = 3.14 Text write/read (создадим поток) ----
38         System.out.println("x = 3.14 Text write/read (создадим поток) ...");
39         System.out.println("x = 3.14 Text write/read (текстовый файл) ...");
40
41         TabulatedFunction tabExp = TabulatedFunctions.tabulate(new Exp(), leftX: 0, rightX: 10, pointsCount: 11);
42
43         // Поместим в файл
44         Writer out = new FileWriter("fileWriter.txt");
45         TabulatedFunctions.writeTabulatedFunction(tabExp, out);
46         out.close();
47
48         // Читаем из файла
49         BufferedReader reader = new FileReader("fileReader.txt");
50         TabulatedFunction readExp = TabulatedFunctions.readTabulatedFunction(reader);
51         reader.close();
52
53         // Сравнение
54         System.out.println("x = 3.14 Text output/input (сравниваем с текущей из файла функцией)");
55         for (double x = 0; x <= 10; x += 1) {
56             System.out.print(String.format("x=%-3.1f, Orig(x)=%.2f, Read(x)=%.2f\n", x, tabExp.getValue(x), readExp.getValue(x)));
57         }
58
59     } catch (IOException e) {
60         e.printStackTrace();
61     }
62
63     // ---- Вызовем x = 3.14 Text output/input (создадим поток) ----
64     System.out.println("x = 3.14 Text output/input (создадим поток) ...");
65     try {
66         TabulatedFunction tabLog = TabulatedFunctions.tabulate(new Log(Math.E), leftX: 1, rightX: 10, pointsCount: 10);
67
68         // Добавим x = 0
69         OutputStream out = new FileOutputStream("fileOutputStream.txt");
70         TabulatedFunctions.writeTabulatedFunction(tabLog, out);
71         out.close();
72
73         // Читаем из файла
74         InputStream in = new FileInputStream("fileInputStream.txt");
75         TabulatedFunction readLog = TabulatedFunctions.readTabulatedFunction(in);
76
77     } catch (IOException e) {
78         e.printStackTrace();
79     }
80 }
```

```

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:61080" "-XX:+ShowCodeDetailsInExceptionMessages" "-cp" "C:\Users\Boris\AppData\Roaming\Code\User\workspaceStorage\bc1765a2ca10ef457a12fe7ac29734\redhat_jdk\lib\Lab-4-2025_4430ba57\bin" "Main"
... Задание 8.1. Син и Кос (10 точек) ...
x<0, sin(x)=0.0000, cos(x)=1.0000
x<1, sin(x)=0.8415, cos(x)=0.5403
x<2, sin(x)=0.9093, cos(x)=0.4161
x<3, sin(x)=0.9848, cos(x)=0.1736
x<4, sin(x)=0.9995, cos(x)=0.2211
x<5, sin(x)=0.9799, cos(x)=0.7795
x<6, sin(x)=0.9272, cos(x)=0.3827
x<7, sin(x)=0.8462, cos(x)=0.7648
x<8, sin(x)=0.7374, cos(x)=0.6958
x<9, sin(x)=0.6026, cos(x)=0.8033
x<10, sin(x)=0.4463, cos(x)=0.9393
x<11, sin(x)=0.2812, cos(x)=0.9583
x<12, sin(x)=0.0985, cos(x)=0.9843
x<13, sin(x)=0.9636, cos(x)=0.3075
x<14, sin(x)=0.9854, cos(x)=0.1708
x<15, sin(x)=0.9993, cos(x)=0.2007
x<16, sin(x)=0.9999, cos(x)=0.0292
x<17, sin(x)=0.9971, cos(x)=0.1289
x<18, sin(x)=0.9822, cos(x)=0.2222
x<19, sin(x)=0.9463, cos(x)=0.3233
x<20, sin(x)=0.8814, cos(x)=0.4161
x<21, sin(x)=0.7853, cos(x)=0.5048
x<22, sin(x)=0.6685, cos(x)=0.5885
x<23, sin(x)=0.5374, cos(x)=0.6953
x<24, sin(x)=0.4074, cos(x)=0.7734
x<25, sin(x)=0.2885, cos(x)=0.8011
x<26, sin(x)=0.1584, cos(x)=0.8409
x<27, sin(x)=0.0278, cos(x)=0.8641
x<28, sin(x)=0.1359, cos(x)=0.9422
x<29, sin(x)=0.3141, cos(x)=0.7739
x<30, sin(x)=0.4990, cos(x)=0.9991
x<31, sin(x)=0.6454, cos(x)=0.9991
... Задание 8.2. Табуляция sine и cosine (10 точек) ...
x<0, tabCos(x)=0.0000, tabCos(x)=1.0000
x<1, tabCos(x)=0.5403, tabCos(x)=0.9991
x<2, tabCos(x)=0.9093, tabCos(x)=0.9999
x<3, tabCos(x)=0.9848, tabCos(x)=0.9999
x<4, tabCos(x)=0.9995, tabCos(x)=0.9999
x<5, tabCos(x)=0.9999, tabCos(x)=0.9999
x<6, tabCos(x)=0.9999, tabCos(x)=0.9999
x<7, tabCos(x)=0.9999, tabCos(x)=0.9999
x<8, tabCos(x)=0.9999, tabCos(x)=0.9999
x<9, tabCos(x)=0.9999, tabCos(x)=0.9999
x<10, tabCos(x)=0.9999, tabCos(x)=0.9999
x<11, tabCos(x)=0.9999, tabCos(x)=0.9999
x<12, tabCos(x)=0.9999, tabCos(x)=0.9999
x<13, tabCos(x)=0.9999, tabCos(x)=0.9999
x<14, tabCos(x)=0.9999, tabCos(x)=0.9999
x<15, tabCos(x)=0.9999, tabCos(x)=0.9999
x<16, tabCos(x)=0.9999, tabCos(x)=0.9999
x<17, tabCos(x)=0.9999, tabCos(x)=0.9999
x<18, tabCos(x)=0.9999, tabCos(x)=0.9999
x<19, tabCos(x)=0.9999, tabCos(x)=0.9999
x<20, tabCos(x)=0.9999, tabCos(x)=0.9999
x<21, tabCos(x)=0.9999, tabCos(x)=0.9999
x<22, tabCos(x)=0.9999, tabCos(x)=0.9999
x<23, tabCos(x)=0.9999, tabCos(x)=0.9999
x<24, tabCos(x)=0.9999, tabCos(x)=0.9999
x<25, tabCos(x)=0.9999, tabCos(x)=0.9999
x<26, tabCos(x)=0.9999, tabCos(x)=0.9999
x<27, tabCos(x)=0.9999, tabCos(x)=0.9999
x<28, tabCos(x)=0.9999, tabCos(x)=0.9999
x<29, tabCos(x)=0.9999, tabCos(x)=0.9999
x<30, tabCos(x)=0.9999, tabCos(x)=0.9999
x<31, tabCos(x)=0.9999, tabCos(x)=0.9999
... Задание 8.3. Сумма квадратов (sin^2 + cos^2) ...
x<0, sin^2+cos^2=1.0000

```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PLATES
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:61080" "-XX:+ShowCodeDetailsInExceptionMessages" "-cp" "C:\Users\Boris\AppData\Roaming\Code\User\workspaceStorage\bc1765a2ca10ef457a12fe7ac29734\redhat_jdk\lib\Lab-4-2025_4430ba57\bin" "Main"
... Задание 8.3. Сумма квадратов (sin^2 + cos^2) ...
x<0, 1.0000<2e-20, 0.9999
x<1, 1.0000<2e-20, 0.9999
x<2, 1.0000<2e-20, 0.9999
x<3, 1.0000<2e-20, 0.9999
x<4, 1.0000<2e-20, 0.9999
x<5, 1.0000<2e-20, 0.9999
x<6, 1.0000<2e-20, 0.9999
x<7, 1.0000<2e-20, 0.9999
x<8, 1.0000<2e-20, 0.9999
x<9, 1.0000<2e-20, 0.9999
x<10, 1.0000<2e-20, 0.9999
x<11, 1.0000<2e-20, 0.9999
x<12, 1.0000<2e-20, 0.9999
x<13, 1.0000<2e-20, 0.9999
x<14, 1.0000<2e-20, 0.9999
x<15, 1.0000<2e-20, 0.9999
x<16, 1.0000<2e-20, 0.9999
x<17, 1.0000<2e-20, 0.9999
x<18, 1.0000<2e-20, 0.9999
x<19, 1.0000<2e-20, 0.9999
x<20, 1.0000<2e-20, 0.9999
x<21, 1.0000<2e-20, 0.9999
x<22, 1.0000<2e-20, 0.9999
x<23, 1.0000<2e-20, 0.9999
x<24, 1.0000<2e-20, 0.9999
x<25, 1.0000<2e-20, 0.9999
x<26, 1.0000<2e-20, 0.9999
x<27, 1.0000<2e-20, 0.9999
x<28, 1.0000<2e-20, 0.9999
x<29, 1.0000<2e-20, 0.9999
x<30, 1.0000<2e-20, 0.9999
x<31, 1.0000<2e-20, 0.9999
... Задание 8.4. Тест write/read (текстовый файл) ...
Сравнение исходной и считанной из файла функции:
x<0, Orig(x)=0.00, Read(x)=0.00
x<0.5, Orig(x)=0.50, Read(x)=0.50
x<1, Orig(x)=1.00, Read(x)=1.00
x<1.5, Orig(x)=1.50, Read(x)=1.50
x<2, Orig(x)=2.00, Read(x)=2.00
x<2.5, Orig(x)=2.50, Read(x)=2.50
x<3, Orig(x)=3.00, Read(x)=3.00
x<3.5, Orig(x)=3.50, Read(x)=3.50
x<4, Orig(x)=4.00, Read(x)=4.00
x<4.5, Orig(x)=4.50, Read(x)=4.50
x<5, Orig(x)=5.00, Read(x)=5.00
x<5.5, Orig(x)=5.50, Read(x)=5.50
x<6, Orig(x)=6.00, Read(x)=6.00
x<6.5, Orig(x)=6.50, Read(x)=6.50
x<7, Orig(x)=7.00, Read(x)=7.00
x<7.5, Orig(x)=7.50, Read(x)=7.50
x<8, Orig(x)=8.00, Read(x)=8.00
x<8.5, Orig(x)=8.50, Read(x)=8.50
x<9, Orig(x)=9.00, Read(x)=9.00
x<9.5, Orig(x)=9.50, Read(x)=9.50
x<10, Orig(x)=10.00, Read(x)=10.00
x<10.5, Orig(x)=10.50, Read(x)=10.50
... Задание 8.5. Тест сериализации (бинарный файл) ...
Сравнение исходной с считанной из файла функции:
x<0, Orig(x)=0.00, Read(x)=0.00
x<1, Orig(x)=1.00, Read(x)=1.00
x<2, Orig(x)=2.00, Read(x)=2.00
x<3, Orig(x)=3.00, Read(x)=3.00
x<4, Orig(x)=4.00, Read(x)=4.00
x<5, Orig(x)=5.00, Read(x)=5.00
x<6, Orig(x)=6.00, Read(x)=6.00
x<7, Orig(x)=7.00, Read(x)=7.00
x<8, Orig(x)=8.00, Read(x)=8.00
x<9, Orig(x)=9.00, Read(x)=9.00
x<10, Orig(x)=10.00, Read(x)=10.00
... PS C:\projects\Lab-4-2025

```



```

x=10.0, Orig(x)=2.38, Read(x)=2.38
... Задание 9: Тест сериализации (Serializable) ...
Сравнение исходной и считанной из файла функции:
x<0, Orig(x)=0.00, Read(x)=0.00
x<1, Orig(x)=1.00, Read(x)=1.00
x<2, Orig(x)=2.00, Read(x)=2.00
x<3, Orig(x)=3.00, Read(x)=3.00
x<4, Orig(x)=4.00, Read(x)=4.00
x<5, Orig(x)=5.00, Read(x)=5.00
x<6, Orig(x)=6.00, Read(x)=6.00
x<7, Orig(x)=7.00, Read(x)=7.00
x<8, Orig(x)=8.00, Read(x)=8.00
x<9, Orig(x)=9.00, Read(x)=9.00
x<10, Orig(x)=10.00, Read(x)=10.00
... 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