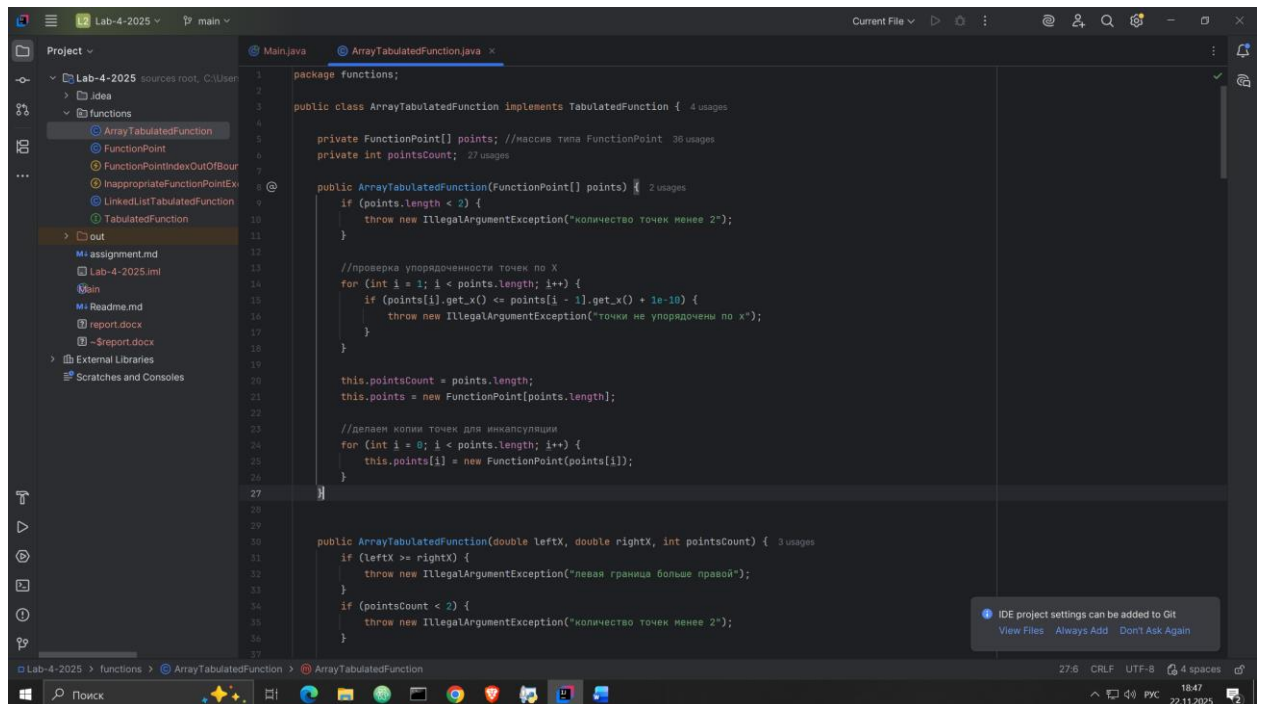


Отчёт по лабораторной работе №4  
Тенигин Валерий 6204-010302D

## Задание 1 – Добавлены конструкторы согласно заданию



```
package functions;

public class ArrayTabulatedFunction implements TabulatedFunction {

    private FunctionPoint[] points; //массив типа FunctionPoint
    private int pointsCount;

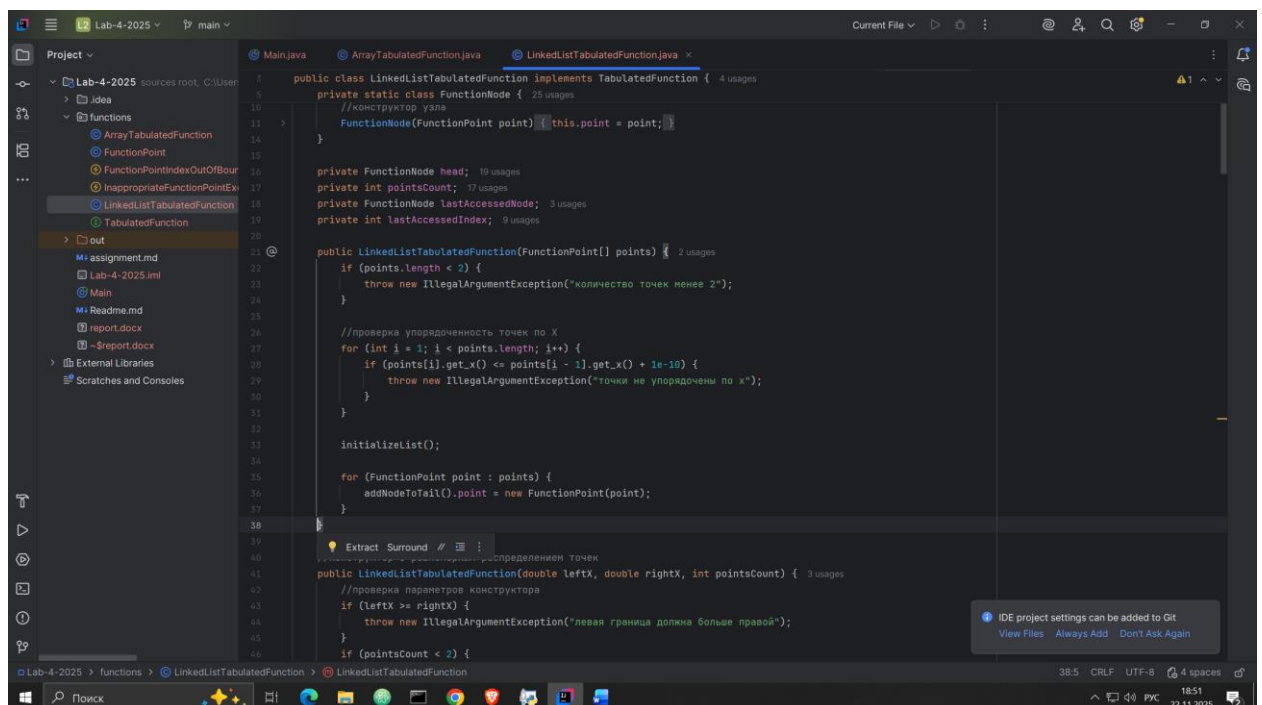
    public ArrayTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException("количество точек менее 2");
        }

        //проверка упорядоченности точек по X
        for (int i = 1; i < points.length; i++) {
            if (points[i].get_x() <= points[i - 1].get_x() + 1e-10) {
                throw new IllegalArgumentException("точки не упорядочены по x");
            }
        }

        this.pointsCount = points.length;
        this.points = new FunctionPoint[points.length];

        //делает копии точек для инкапсуляции
        for (int i = 0; i < points.length; i++) {
            this.points[i] = new FunctionPoint(points[i]);
        }
    }

    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if (leftX >= rightX) {
            throw new IllegalArgumentException("левая граница больше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("количество точек менее 2");
        }
    }
}
```



```
package functions;

public class LinkedListTabulatedFunction implements TabulatedFunction {

    private static class FunctionNode {
        FunctionPoint point;
    }

    private FunctionNode head;
    private int pointsCount;
    private FunctionNode lastAccessedNode;
    private int lastAccessedIndex;

    public LinkedListTabulatedFunction(FunctionPoint[] points) {
        if (points.length < 2) {
            throw new IllegalArgumentException("количество точек менее 2");
        }

        //проверка упорядоченности точек по X
        for (int i = 1; i < points.length; i++) {
            if (points[i].get_x() <= points[i - 1].get_x() + 1e-10) {
                throw new IllegalArgumentException("точки не упорядочены по x");
            }
        }

        initializeList();

        for (FunctionPoint point : points) {
            addNodeToTail().point = new FunctionPoint(point);
        }
    }

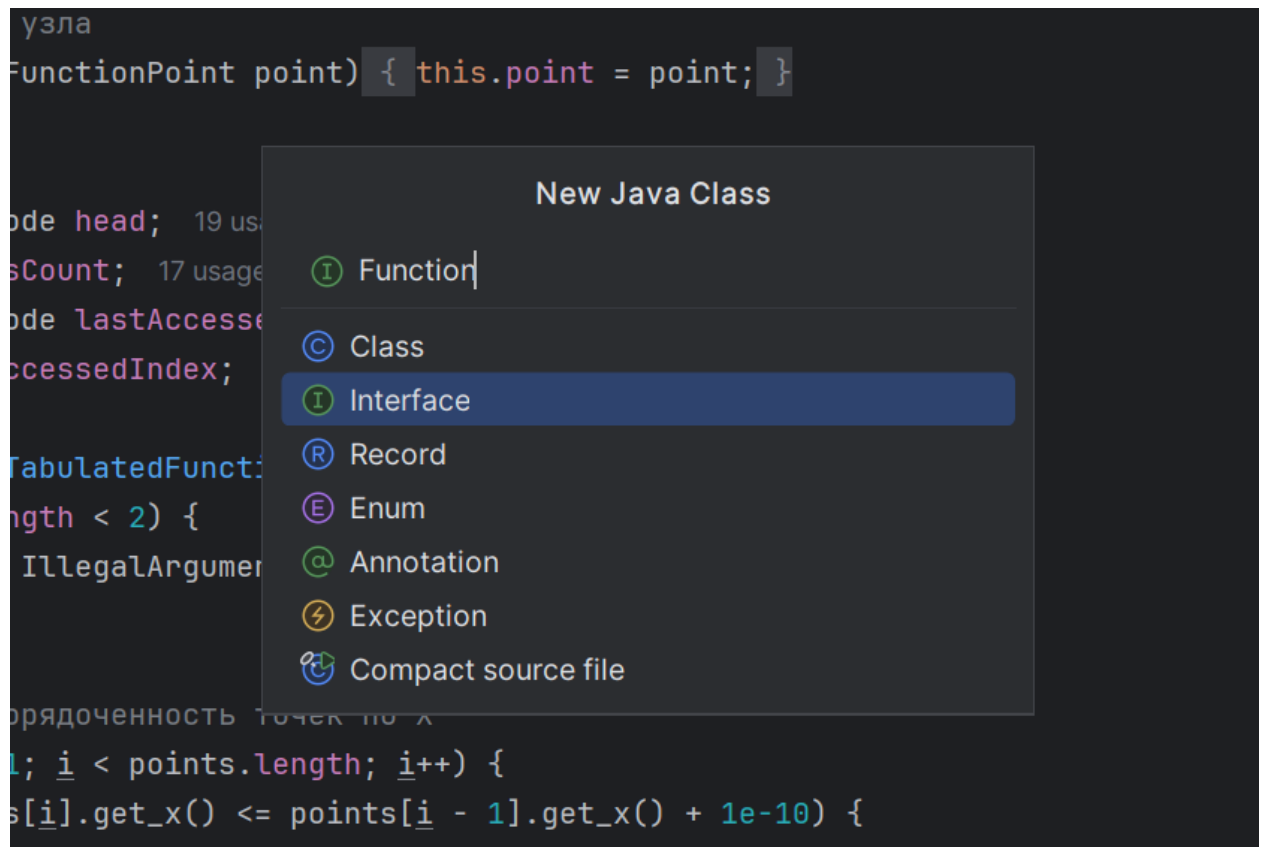
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) {
        //проверка параметров конструктора
        if (leftX >= rightX) {
            throw new IllegalArgumentException("левая граница должна быть больше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("количество точек менее 2");
        }
    }
}
```

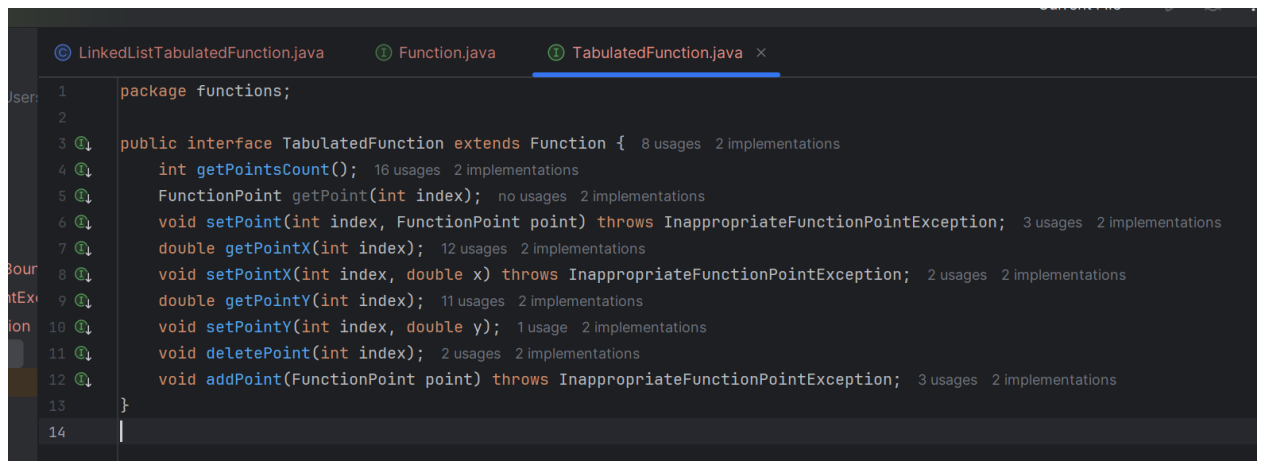
## Задание 2

Создан интерфейс Function с методами: `getLeftDomainBorder()`, `getRightDomainBorder()`, `getFunctionValue(double x)`

Интерфейс TabulatedFunction теперь расширяет Function

Удалены дублирующие методы из TabulatedFunction





```
1 package functions;
2
3 public interface TabulatedFunction extends Function { 8 usages 2 implementations
4     int getPointsCount(); 16 usages 2 implementations
5     FunctionPoint getPoint(int index); no usages 2 implementations
6     void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException; 3 usages 2 implementations
7     double getPointX(int index); 12 usages 2 implementations
8     void setPointX(int index, double x) throws InappropriateFunctionPointException; 2 usages 2 implementations
9     double getPointY(int index); 11 usages 2 implementations
10    void setPointY(int index, double y); 1 usage 2 implementations
11    void deletePoint(int index); 2 usages 2 implementations
12    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException; 3 usages 2 implementations
13 }
14
```

### Задание 3

Создан пакет functions.basic

Реализованы классы – Exp(экспонента), Log (логарифм с заданным основанием), TrigonometricFunction (базовый класс для тригонометрических функций), Sin, Cos, Tan (синус, косинус, тангенс)

## New Package

functions.basic

## New Java Class

Ⓒ Exp

Ⓒ Class

Ⓘ Interface

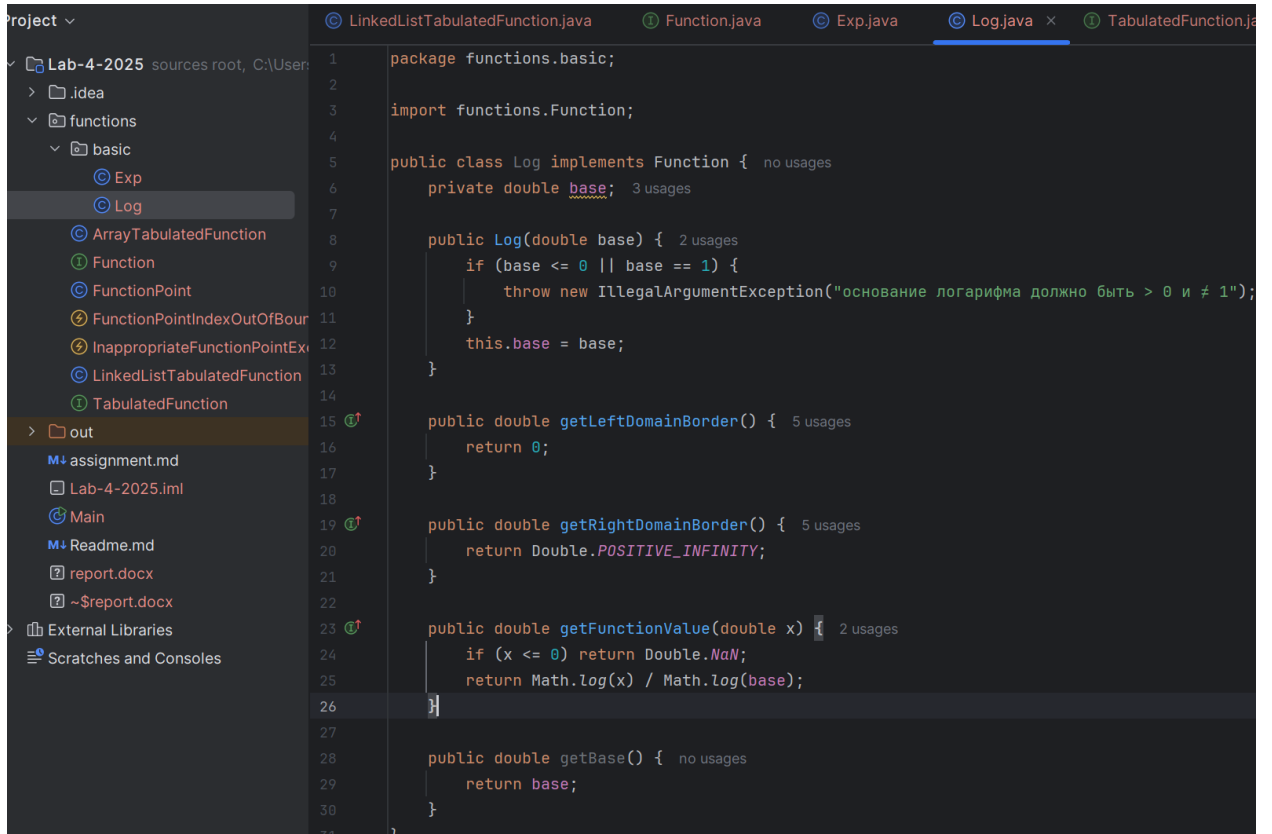
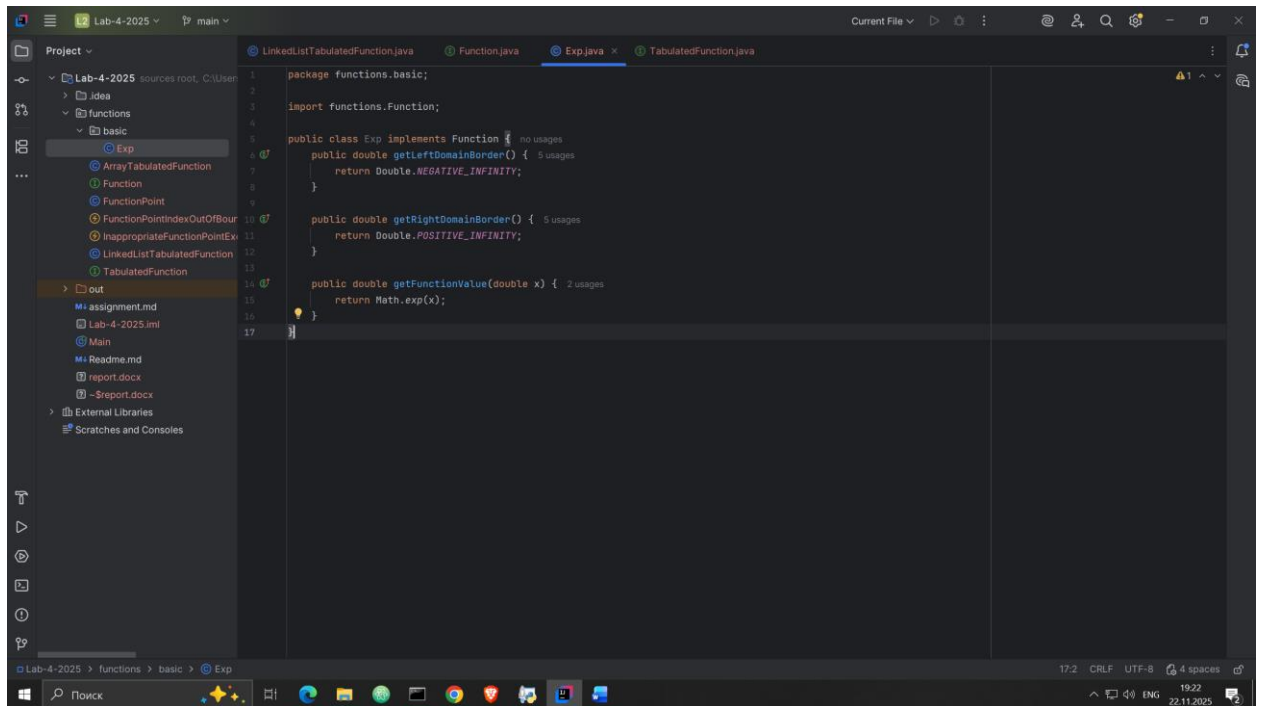
Ⓓ Record

Ⓔ Enum

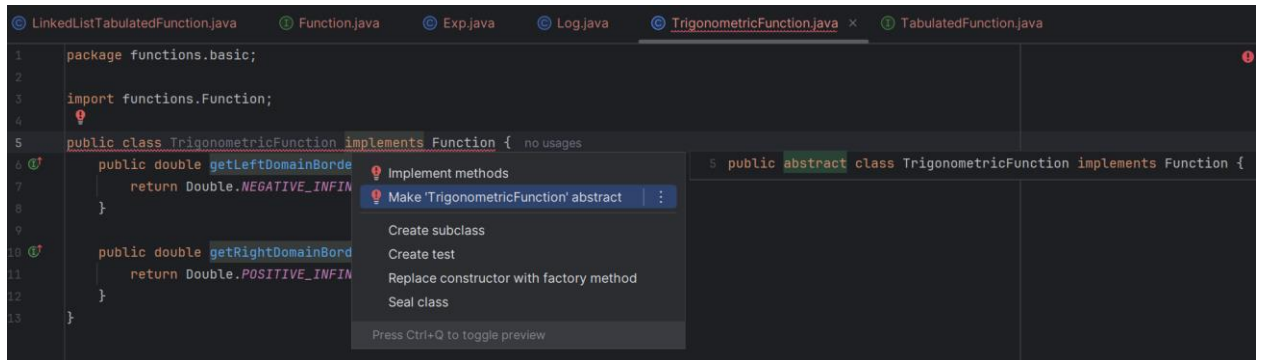
@ Annotation

⚡ Exception

📄 Compact source file



При создании TrigonometricFunction появилась ошибка, idea предложила исправить ошибку используя абстрактный класс



The screenshot shows the IntelliJ IDEA IDE with the `TrigonometricFunction.java` file open. The code defines a class `TrigonometricFunction` that implements the `Function` interface. The class has two methods: `getLeftDomainBorder()` and `getRightDomainBorder()`. A code completion menu is visible, suggesting the option to "Make 'TrigonometricFunction' abstract".

```
package functions.basic;

import functions.Function;

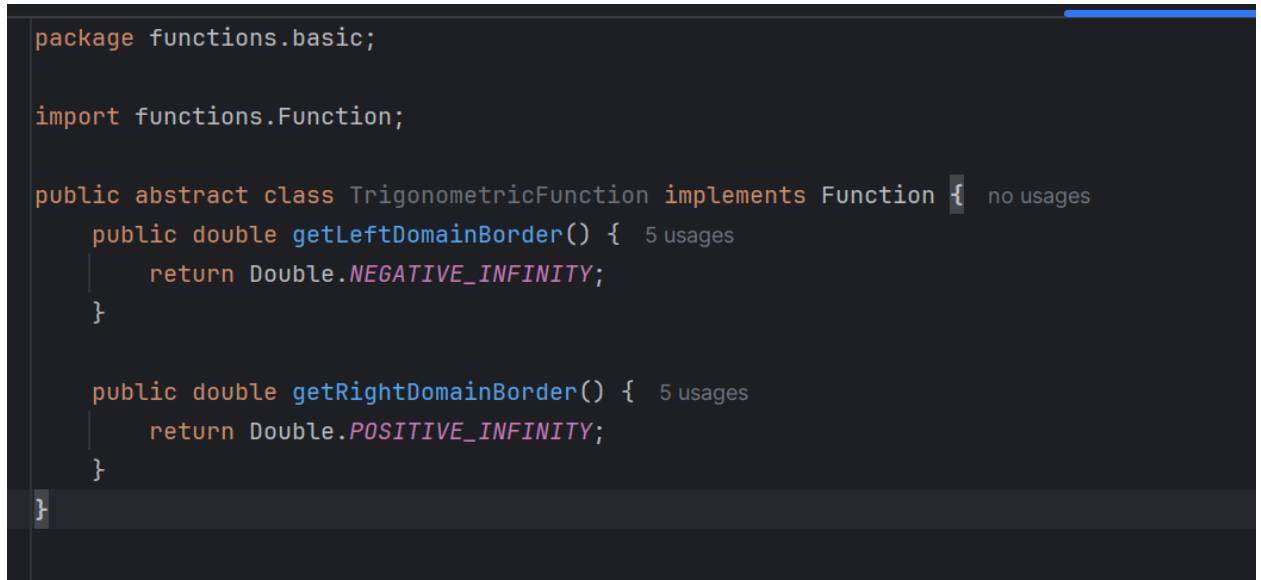
public class TrigonometricFunction implements Function {
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
}
```

Code completion menu options:

- Implement methods
- Make 'TrigonometricFunction' abstract
- Create subclass
- Create test
- Replace constructor with factory method
- Seal class

Press Ctrl+Q to toggle preview



The screenshot shows the final code for the `TrigonometricFunction` class, which is now an abstract class. The code is as follows:

```
package functions.basic;

import functions.Function;

public abstract class TrigonometricFunction implements Function {
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
}
```

```
1 package functions.basic;
2
3 public class Sin extends TrigonometricFunction { no usages
4     public double getFunctionValue(double x) { 2 usages
5         return Math.sin(x);
6     }
7 }
```

© LinkedListTabulatedFunction.java    ⓘ Function.java    © Exp.java    © Log.j

```
1 package functions.basic;
2
3 public class Cos extends TrigonometricFunction { no usages
4     public double getFunctionValue(double x) { 2 usages
5         return Math.cos(x);
6     }
7 }
```

er: © LinkedListTabulatedFunction.java    ⓘ Function.java    © Exp.java

```
1 package functions.basic;
2
3 public class Tan extends TrigonometricFunction { no usages
4     public double getFunctionValue(double x) {
5         return Math.tan(x);
6     }
7 }
```

## Задание 4

Создан пакет functions.meta

Реализованы классы для комбинирования функций - Sum (сумма двух функций), Mult (произведение двух функций), Power (возведение функции в степень), Scale (масштабирование по осям), Shift (сдвиг по осям), Composition (композиция функций)



# New Package

functions.meta

The screenshot shows the IntelliJ IDEA IDE with the 'Sum' class open in the 'functions.meta' package. The class implements the 'Function' interface and contains methods for getting domain borders and function values.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 //складываем две функции
6 public class Sum implements Function { по usages
7     private Function f1, f2; //две функции которые будем складывать 4 usages
8
9     public Sum(Function f1, Function f2) { 2 usages
10         this.f1 = f1;
11         this.f2 = f2;
12     }
13
14     //берем самую левую правую границу из двух функций
15     public double getLeftDomainBorder() {
16         return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
17     }
18
19     //берем самую левую правую границу из двух функций
20     public double getRightDomainBorder() {
21         return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
22     }
23
24     //складываем значения двух функций в точке x
25     public double getFunctionValue(double x) {
26         //если x не входит в область определения - возвращаем NaN
27         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
28             return Double.NaN;
29         }
30         return f1.getFunctionValue(x) + f2.getFunctionValue(x);
31     }
32 }
```

The screenshot shows the IntelliJ IDEA IDE with the 'Power' class open in the 'functions.meta' package. The class implements the 'Function' interface and contains methods for getting domain borders and function values.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 //возводим функцию в степень
6 public class Power implements Function { по usages
7     private Function f; //исходная функция которую возводим в степень 4 usages
8     private double power; //в какую степень возводим 2 usages
9
10     public Power(Function f, double power) { 2 usages
11         this.f = f;
12         this.power = power;
13     }
14
15     //левая граница такая же как у исходной функции
16     public double getLeftDomainBorder() {
17         return f.getLeftDomainBorder();
18     }
19
20     //правая граница такая же как у исходной функции
21     public double getRightDomainBorder() {
22         return f.getRightDomainBorder();
23     }
24
25     //берем значение исходной функции и возводим в степень
26     public double getFunctionValue(double x) {
27         double value = f.getFunctionValue(x);
28         if (Double.isNaN(value)) return Double.NaN;
29         return Math.pow(value, power);
30     }
31 }
```

```
1 package functions.meta;
2
3 import functions.Function;
4
5 //умножаем две функции
6 public class Mult implements Function { no usages
7     private Function f1, f2; //две функции которые будем умножать 4 usages
8
9     public Mult(Function f1, Function f2) { 2 usages
10         this.f1 = f1;
11         this.f2 = f2;
12     }
13
14     //пересечение областей определения слева
15     public double getLeftDomainBorder() {
16         return Math.max(f1.getLeftDomainBorder(), f2.getLeftDomainBorder());
17     }
18
19     //пересечение областей определения справа
20     public double getRightDomainBorder() {
21         return Math.min(f1.getRightDomainBorder(), f2.getRightDomainBorder());
22     }
23
24     //переносим значения двух функций
25     public double getFunctionValue(double x) {
26         if (x < getLeftDomainBorder() || x > getRightDomainBorder()) {
27             return Double.NaN;
28         }
29         return f1.getFunctionValue(x) * f2.getFunctionValue(x);
30     }
31 }
```

```
1 package functions.meta;
2
3 import functions.Function;
4
5 //вкладываем одну функцию в другую f2(f1(x))
6 public class Composition implements Function { no usages
7     private Function f1, f2; //f1 - внутренняя, f2 - внешняя 4 usages
8
9     public Composition(Function f1, Function f2) { 2 usages
10         this.f1 = f1;
11         this.f2 = f2;
12     }
13
14     //левая граница от внутренней функции f1
15     public double getLeftDomainBorder() {
16         return f1.getLeftDomainBorder();
17     }
18
19     //правая граница от внутренней функции f1
20     public double getRightDomainBorder() {
21         return f1.getRightDomainBorder();
22     }
23
24     //сначала вычисляем внутреннюю функцию, потом внешнюю
25     public double getFunctionValue(double x) {
26         double innerValue = f1.getFunctionValue(x); //считаем f1(x)
27         if (Double.isNaN(innerValue)) return Double.NaN;
28         return f2.getFunctionValue(innerValue); //считаем f2(результат)
29     }
30 }
```

The screenshot shows the IntelliJ IDEA IDE with the 'Scale.java' file open. The file is located in the 'functions.meta' package. The code defines a 'Scale' class that implements the 'Function' interface. It includes methods for scaling a function by a factor 'x' and 'y'. The 'getLeftDomainBorder()' and 'getRightDomainBorder()' methods return the scaled domain boundaries. The 'getFunctionValue(double x)' method scales the input 'x' and then calls the original function 'f' to get the value, which is then scaled by 'y'.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 //растягиваем или сжимаем функцию по осям x и y
6 public class Scale implements Function {
7     private Function f; //исходная функция
8     private double scaleX; //во сколько раз растянуть по x
9     private double scaleY; //во сколько раз растянуть по y
10
11     public Scale(Function f, double scaleX, double scaleY) {
12         this.f = f;
13         this.scaleX = scaleX;
14         this.scaleY = scaleY;
15     }
16
17     //левая граница растягивается по x
18     public double getLeftDomainBorder() {
19         return f.getLeftDomainBorder() * scaleX;
20     }
21
22     //правая граница растягивается по x
23     public double getRightDomainBorder() {
24         return f.getRightDomainBorder() * scaleX;
25     }
26
27     //масштабируем координаты и значение функции
28     public double getFunctionValue(double x) {
29         double scaledX = x / scaleX; //обратно масштабируем x
30         if (scaledX < f.getLeftDomainBorder() || scaledX > f.getRightDomainBorder()) {
31             return Double.NaN;
32         }
33         return f.getFunctionValue(scaledX) * scaleY; //масштабируем y
34     }
35 }
```

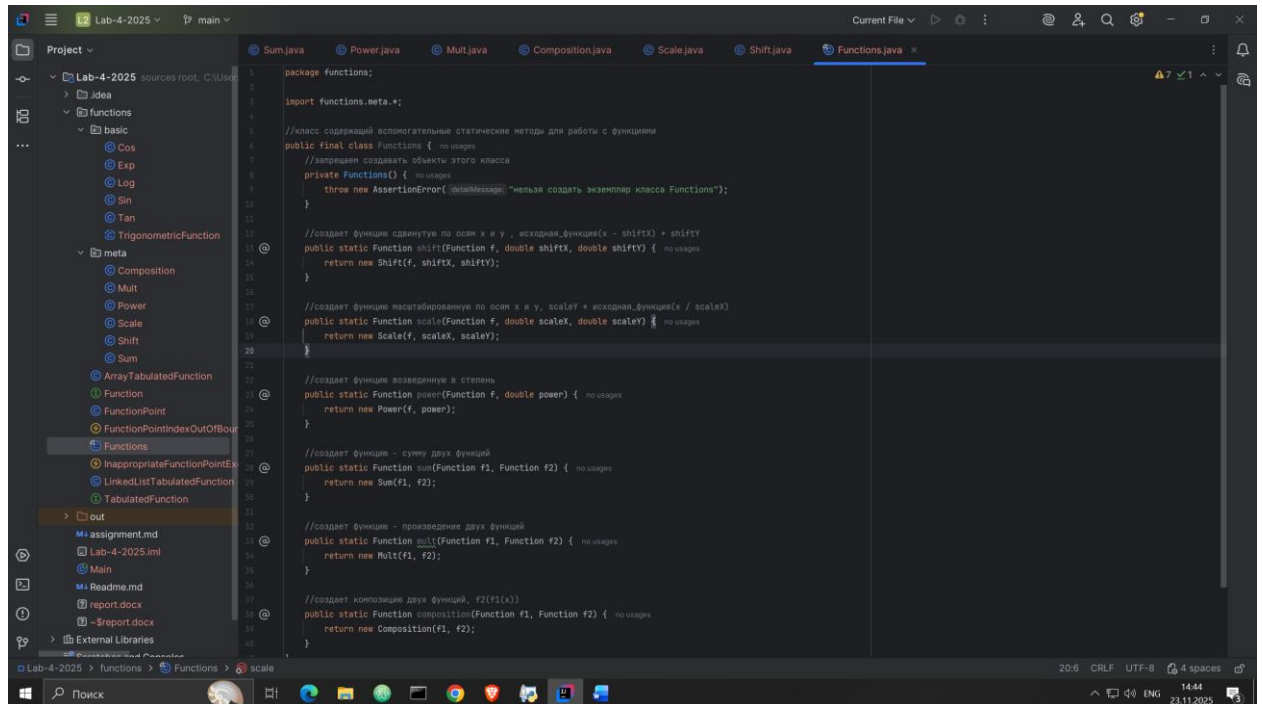
The screenshot shows the IntelliJ IDEA IDE with the 'Shift.java' file open. The file is located in the 'functions.meta' package. The code defines a 'Shift' class that implements the 'Function' interface. It includes methods for shifting a function by a value 'x' and 'y'. The 'getLeftDomainBorder()' and 'getRightDomainBorder()' methods return the shifted domain boundaries. The 'getFunctionValue(double x)' method shifts the input 'x' and then calls the original function 'f' to get the value, which is then shifted by 'y'.

```
1 package functions.meta;
2
3 import functions.Function;
4
5 //сдвигаем функцию влево-право и вверх-вниз
6 public class Shift implements Function {
7     private Function f; //исходная функция
8     private double shiftX; //на сколько сдвинуть по x
9     private double shiftY; //на сколько сдвинуть по y
10
11     public Shift(Function f, double shiftX, double shiftY) {
12         this.f = f;
13         this.shiftX = shiftX;
14         this.shiftY = shiftY;
15     }
16
17     //левая граница сдвигается по x
18     public double getLeftDomainBorder() {
19         return f.getLeftDomainBorder() + shiftX;
20     }
21
22     //правая граница сдвигается по x
23     public double getRightDomainBorder() {
24         return f.getRightDomainBorder() + shiftX;
25     }
26
27     //сдвигаем координаты и значение функции
28     public double getFunctionValue(double x) {
29         double shiftedX = x - shiftX; //обратно сдвигаем x
30         if (shiftedX < f.getLeftDomainBorder() || shiftedX > f.getRightDomainBorder()) {
31             return Double.NaN;
32         }
33         return f.getFunctionValue(shiftedX) + shiftY; //сдвигаем y
34     }
35 }
```

## Задание 5

Создан класс Functions со статическими методами-обёртками

Реализованы методы: shift(), scale(), power(), sum(), mult(), composition()



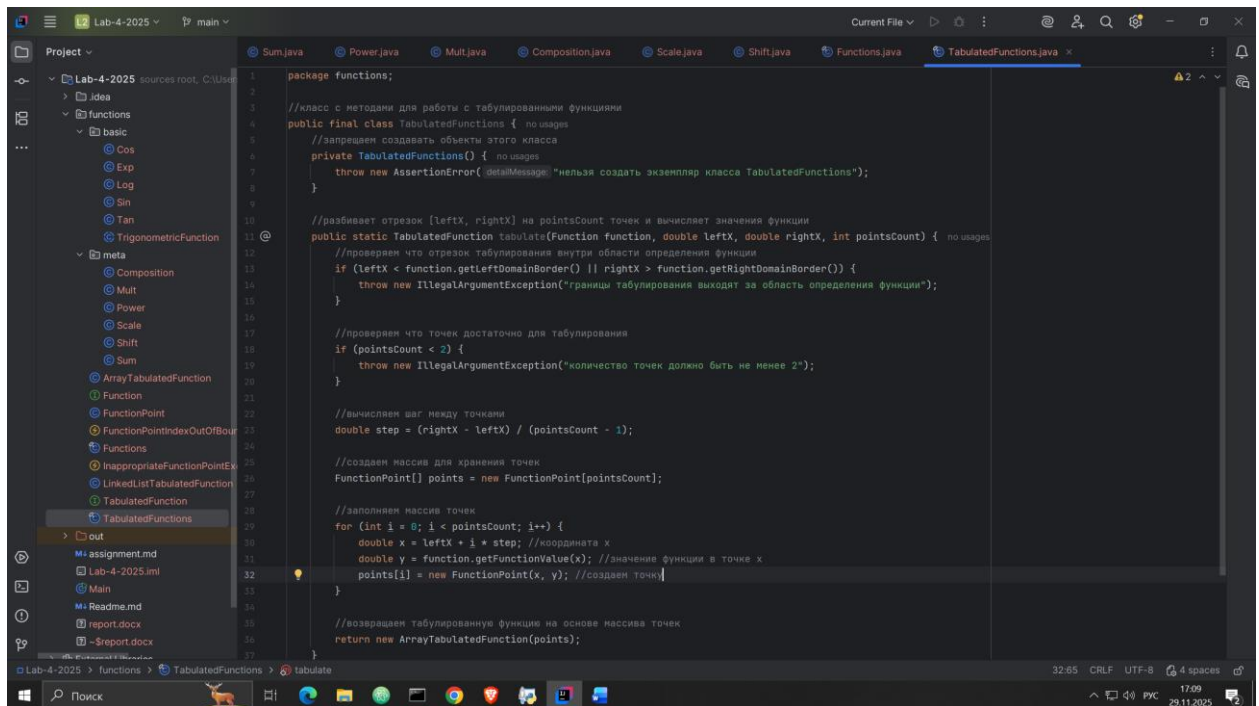
```
1 package functions;
2
3 import functions.meta.*;
4
5 //класс содержащий вспомогательные статические методы для работы с функциями
6 public final class Functions { //по умолчанию
7     //запрещено создавать объекты этого класса
8     private Functions() { //по умолчанию
9         throw new AssertionError("нельзя создать экземпляр класса Functions");
10    }
11
12    //создает функцию сдвинутой по осям x и y, исходная_функция(x - shiftX) + shiftY
13    public static Function shift(Function f, double shiftX, double shiftY) { //по умолчанию
14        return new Shift(f, shiftX, shiftY);
15    }
16
17    //создает функцию масштабированную по осям x и y, scaleY * исходная_функция(x / scaleX)
18    public static Function scale(Function f, double scaleX, double scaleY) { //по умолчанию
19        return new Scale(f, scaleX, scaleY);
20    }
21
22    //создает функцию возведенную в степень
23    public static Function power(Function f, double power) { //по умолчанию
24        return new Power(f, power);
25    }
26
27    //создает функцию - сумму двух функций
28    public static Function sum(Function f1, Function f2) { //по умолчанию
29        return new Sum(f1, f2);
30    }
31
32    //создает функцию - произведение двух функций
33    public static Function mult(Function f1, Function f2) { //по умолчанию
34        return new Mult(f1, f2);
35    }
36
37    //создает композицию двух функций, f2(f1(x))
38    public static Function composition(Function f1, Function f2) { //по умолчанию
39        return new Composition(f1, f2);
40    }
41 }
```

## Задание 6

Создан класс TabulatedFunctions со статическими методами

Реализован метод tabulate() для создания табулированных функций из аналитических

## Добавлены проверки границ области определения и количества точек



## Задание 7

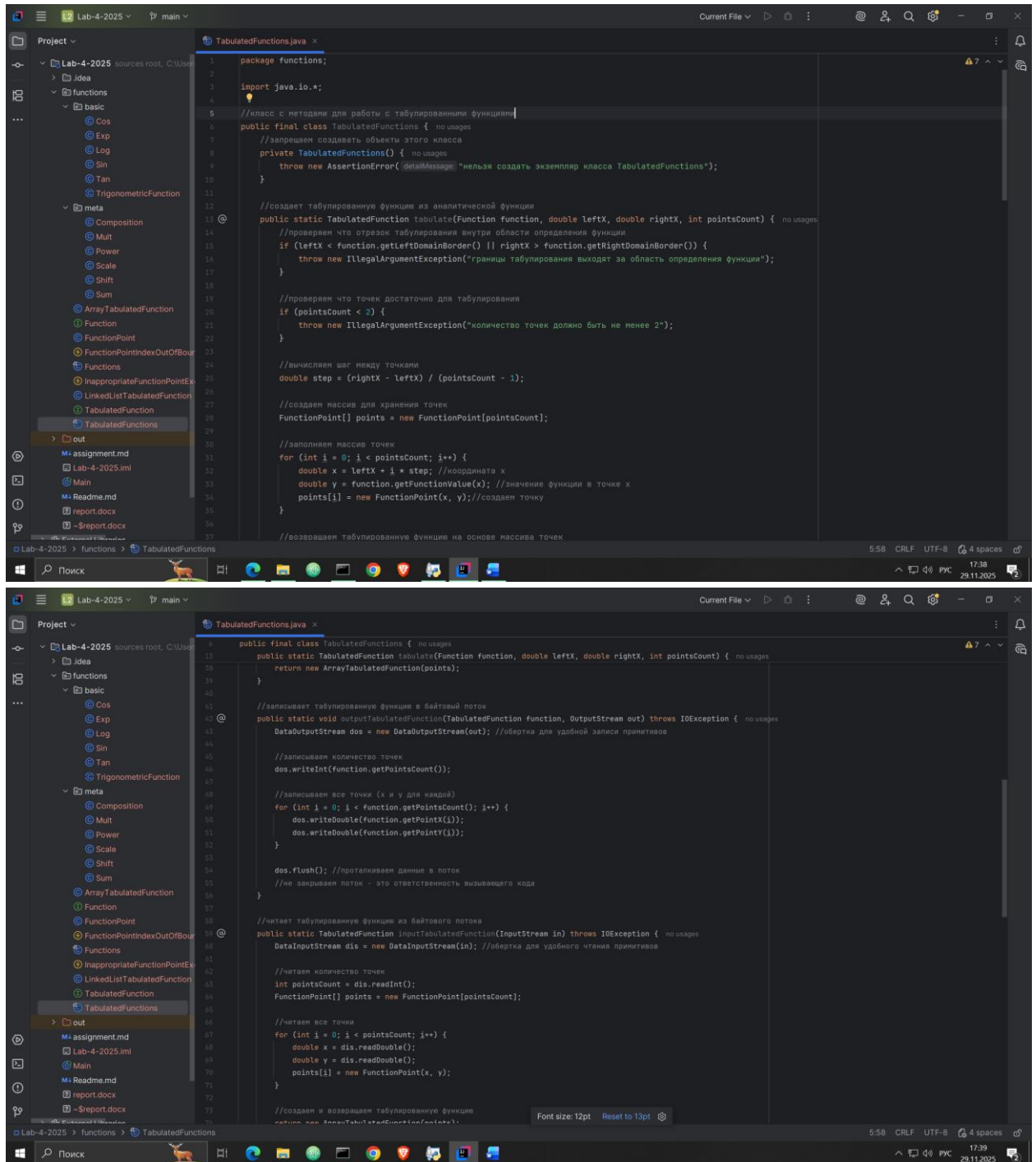
Реализованы методы в классе `TabulatedFunctions`

`outputTabulatedFunction()` - запись в байтовый поток

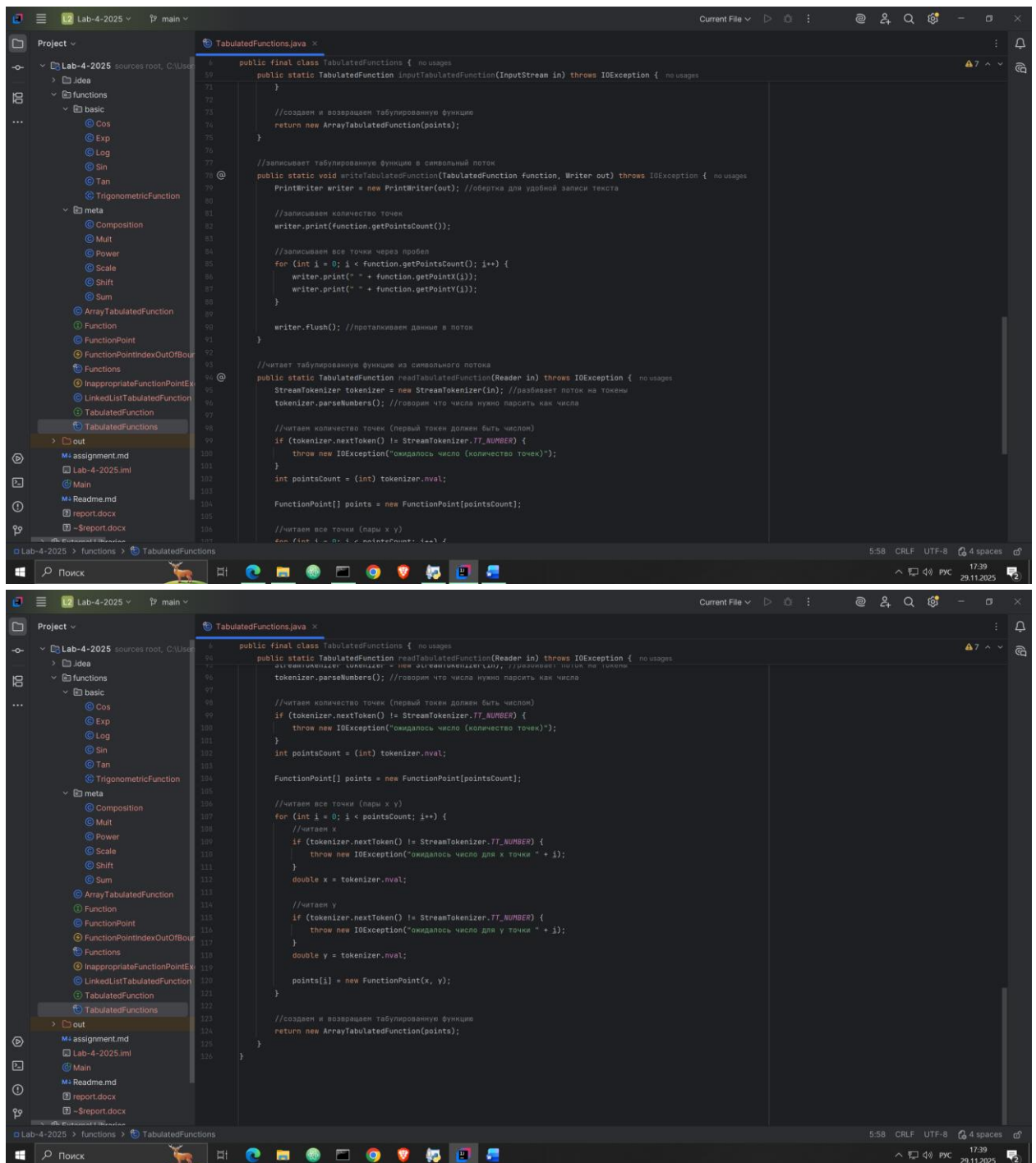
`inputTabulatedFunction()` - чтение из байтового потока

`writeTabulatedFunction()` - запись в символьный поток

## readTabulatedFunction() - чтение из символьного потока







## Задание 8

```
13 public class Main {
14     public static void main(String[] args) throws InappropriateFunctionPointException {
15         System.out.println("=== ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ №4 ===");
16         System.out.println("Создан: 50");
17
18         try {
19             //тест аналитических функций (задание 3)
20             System.out.println("\n=== ТЕСТ 10: АНАЛИТИЧЕСКИЕ ФУНКЦИИ ===");
21
22             Sin sin = new Sin();
23             Cos cos = new Cos();
24             Exp exp = new Exp();
25             Log ln = new Log(Math.E);
26
27             System.out.println("sin(pi/2) = " + sin.getFunctionValue(Math.PI/2));
28             System.out.println("cos(0) = " + cos.getFunctionValue(0));
29             System.out.println("exp(1) = " + exp.getFunctionValue(1));
30             System.out.println("ln(e) = " + ln.getFunctionValue(Math.E));
31
32             System.out.println("область определения sin: [" + sin.getRightDomainBorder() + ", " + sin.getLeftDomainBorder() + "]");
33             System.out.println("область определения ln: [" + ln.getRightDomainBorder() + ", " + ln.getLeftDomainBorder() + "]");
34
35             //тест мета-функций (задание 4)
36             System.out.println("\n=== ТЕСТ 11: МЕТА-ФУНКЦИИ ===");
37
38             Function sumFunc = new Sum(sin, cos);
39             Function multFunc = new Mult(sin, cos);
40             Function powerFunc = new Power(sin, 2);
41             Function shiftFunc = new Shift(sin, 1, 0.5);
42             Function scaleFunc = new Scale(sin, 2, 0.5);
43             Function compFunc = new Composition(sin, cos);
44
45             System.out.println("sin(1) + cos(1) = " + sumFunc.getFunctionValue(1));
46             System.out.println("sin(1) * cos(1) = " + multFunc.getFunctionValue(1));
47             System.out.println("sin^2(1) = " + powerFunc.getFunctionValue(1));
48             System.out.println("sin(1-1) + 0.5 = " + shiftFunc.getFunctionValue(1));
49             System.out.println("sin(1-1) * 0.5 = " + scaleFunc.getFunctionValue(1));
50             System.out.println("3 * sin(1/2) = " + compFunc.getFunctionValue(1));
51
52         } catch (InappropriateFunctionPointException e) {
53             System.out.println(e.getMessage());
54         }
55     }
56 }
```

```
57 //тест класса Functions (задание 5)
58 System.out.println("\n=== ТЕСТ 12: КЛАСС Functions ===");
59
60 Function f1 = Functions.sum(sin, cos);
61 Function f2 = Functions.power(sin, 2);
62 Function f3 = Functions.shift(sin, 1, 0.5);
63 Function f4 = Functions.composition(sin, cos);
64
65 System.out.println("Functions.sum(sin, cos)(1) = " + f1.getFunctionValue(1));
66 System.out.println("Functions.power(sin, 2)(1) = " + f2.getFunctionValue(1));
67 System.out.println("Functions.shift(sin, 1, 0.5)(1) = " + f3.getFunctionValue(1));
68 System.out.println("Functions.composition(sin, cos)(1) = " + f4.getFunctionValue(1));
69
70 //тест табулирования (задание 6)
71 System.out.println("\n=== ТЕСТ 13: ТАБУЛИРОВАНИЕ ===");
72
73 TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sin, 0, Math.PI, 5);
74 System.out.println("табулированный sin на [0, pi] с 5 точками:");
75 for (int i = 0; i < tabulatedSin.getPointsCount(); i++) {
76     System.out.printf("Точка %d: (%.2f, %.4f)\n", i, tabulatedSin.getPointX(i), tabulatedSin.getPointY(i));
77 }
78
79 //тест ввода/вывода (задание 7)
80 System.out.println("\n=== ТЕСТ 14: ВВОД/ВЫВОД ===");
81
82 //Бинарный формат
83 try (FileOutputStream fos = new FileOutputStream("test_binary.dat")) {
84     TabulatedFunctions.outputTabulatedFunction(tabulatedSin, fos);
85 } catch (IOException e) {
86     System.out.println(e.getMessage());
87 }
88 }
```



```
13 public class Main {
14     public static void main(String[] args) throws InappropriateFunctionPointException {
15         System.out.println("\n=== ТЕСТ 14: ВВОД/ВЫВОД ===");
16
17         //Бинарный формат
18         try (FileOutputStream fos = new FileOutputStream("test_binary.dat")) {
19             TabulatedFunctions.outputTabulatedFunction(tabulatedSin, fos);
20             System.out.println("функция записана в бинарный файл");
21         }
22
23         TabulatedFunction readFromBinary;
24         try (FileInputStream fis = new FileInputStream("test_binary.dat")) {
25             readFromBinary = TabulatedFunctions.inputTabulatedFunction(fis);
26             System.out.println("функция прочитана из бинарного файла");
27         }
28
29         //текстовый формат
30         try (FileWriter fw = new FileWriter("test_text.txt")) {
31             TabulatedFunctions.writeTabulatedFunction(tabulatedSin, fw);
32             System.out.println("функция записана в текстовый файл");
33         }
34
35         TabulatedFunction readFromText;
36         try (FileReader fr = new FileReader("test_text.txt")) {
37             readFromText = TabulatedFunctions.readTabulatedFunction(fr);
38             System.out.println("функция прочитана из текстового файла");
39         }
40
41         // Сравнение
42         System.out.println("\nсравнение исходной и считанных функций");
43         for (int i = 0; i < tabulatedSin.getPointsCount(); i++) {
44             double original = tabulatedSin.getPointY(i);
45             double fromBinary = readFromBinary.getPointY(i);
46             double fromText = readFromText.getPointY(i);
47             System.out.printf("точка %d: исходная%.4f, бинарная%.4f, текстовая%.4f\n",
48                             i, original, fromBinary, fromText);
49         }
50     }
51 }
```

```
284         System.out.println("\nсравнение исходной и считанных функций");
285         for (int i = 0; i < tabulatedSin.getPointsCount(); i++) {
286             double original = tabulatedSin.getPointY(i);
287             double fromBinary = readFromBinary.getPointY(i);
288             double fromText = readFromText.getPointY(i);
289             System.out.printf("точка %d: исходная%.4f, бинарная%.4f, текстовая%.4f\n",
290                             i, original, fromBinary, fromText);
291         }
292
293         System.out.println("\n=== ТЕСТ 15: СПОЖНАЯ КОМПОЗИЦИЯ ===");
294
295         // ln(exp(x)) должна быть близка к x
296         Function lnOfExp = Functions.composition(exp, ln);
297         TabulatedFunction tabulatedLnExp = TabulatedFunctions.tabulate(lnOfExp, leftX: 0.1, rightX: 5, pointsCount: 10);
298
299         System.out.println("\nln(exp(x)) на отрезке [0.1, 5]:");
300         for (int i = 0; i < tabulatedLnExp.getPointsCount(); i++) {
301             double x = tabulatedLnExp.getPointX(i);
302             double y = tabulatedLnExp.getPointY(i);
303             double error = Math.abs(x - y);
304             System.out.printf("x=%.1f: ln(exp(x))=%.4f, ошибка=%.6f\n", x, y, error);
305         }
306
307         System.out.println("\n=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ===");
308     }
309 } catch (Exception e) {
310     System.out.println("ошибка в новых тестах: " + e.getMessage());
311 }
312 }
```

```
*** ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ №4 ***  
=====
```

```
*** ТЕСТ 10: АНАЛИТИЧЕСКИЕ ФУНКЦИИ ***  
sin(pi/2) = 1.0  
cos(0) = 1.0  
exp(1) = 2.718281828459045  
ln(e) = 1.0  
область определения sin: [-Infinity, Infinity]  
область определения ln: [0.0, Infinity]
```

```
*** ТЕСТ 11: МЕТА-ФУНКЦИИ ***  
sin(1) + cos(1) = 1.3817732986760363  
sin(1) * cos(1) = 0.4546487134128489  
sin^2(1) = 0.7080734182735712  
sin(1-1) + 0.5 = 0.5  
3 * sin(1/2) = 1.438276615812609  
sin(cos(1)) = 0.6663667453928805
```

```
*** ТЕСТ 12: КЛАСС Functions ***  
Functions.sum(sin, cos)(1) = 1.3817732986760363  
Functions.power(sin, 2)(1) = 0.7080734182735712  
Functions.shift(sin, 0.5, 1)(1) = 1.479425538604203  
Functions.composition(sin, cos)(1) = 0.6663667453928805
```

```
--- ТЕСТ 13: ТАБУЛИРОВАНИЕ ---  
табулированный sin на [0, pi] с 5 точками:  
точка 0: (0.00, 0.0000)  
точка 1: (0.79, 0.7071)  
точка 2: (1.57, 1.0000)  
точка 3: (2.36, 0.7071)  
точка 4: (3.14, 0.0000)
```

```
--- ТЕСТ 14: ВВОД/ВЫВОД ---  
функция записана в бинарный файл  
функция прочитана из бинарного файла  
функция записана в текстовый файл  
функция прочитана из текстового файла
```

```
сравнение исходной и считанных функций  
точка 0: исходная=0.0000, бинарная=0.0000, текстовая=0.0000  
точка 1: исходная=0.7071, бинарная=0.7071, текстовая=0.7071  
точка 2: исходная=1.0000, бинарная=1.0000, текстовая=1.0000  
точка 3: исходная=0.7071, бинарная=0.7071, текстовая=0.7071  
точка 4: исходная=0.0000, бинарная=0.0000, текстовая=1.2246
```

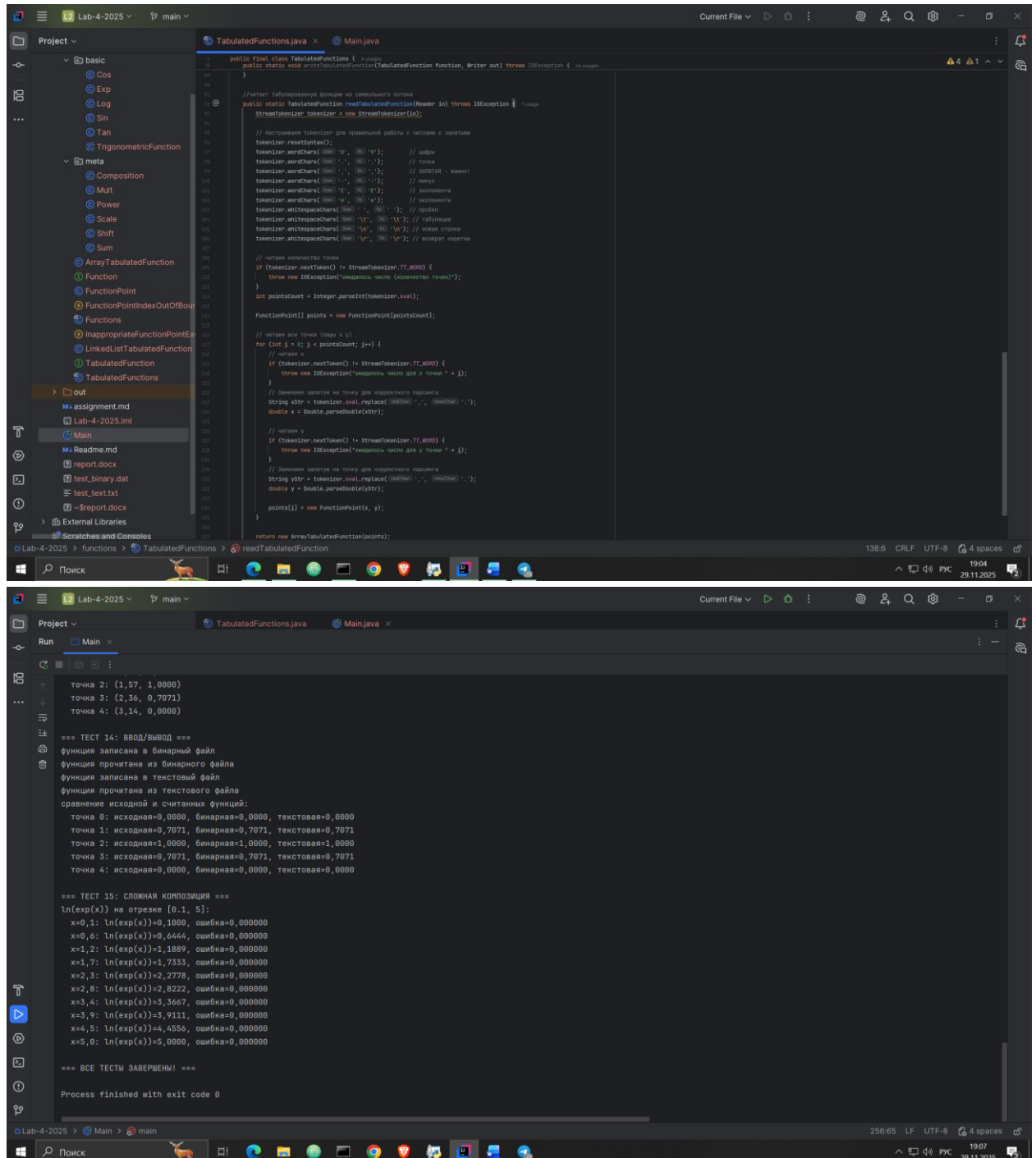
```
*** ТЕСТ 15: СЛОЖНАЯ КОМПОЗИЦИЯ ***  
ln(exp(x)) на отрезке [0.1, 5]:  
x=0.1: ln(exp(x))=0.1000, ошибка=0.000000  
x=0.6: ln(exp(x))=0.6444, ошибка=0.000000  
x=1.2: ln(exp(x))=1.1889, ошибка=0.000000  
x=1.7: ln(exp(x))=1.7333, ошибка=0.000000  
x=2.3: ln(exp(x))=2.2778, ошибка=0.000000  
x=2.8: ln(exp(x))=2.8222, ошибка=0.000000  
x=3.4: ln(exp(x))=3.3667, ошибка=0.000000  
x=3.9: ln(exp(x))=3.9111, ошибка=0.000000  
x=4.5: ln(exp(x))=4.4556, ошибка=0.000000  
x=5.0: ln(exp(x))=5.0000, ошибка=0.000000
```

```
*** ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ***  
Process finished with exit code 0
```

При тестировании выявились проблемы

Проблема : точка 4: текстовая=1,2246 вместо 0,0000,  
StreamTokenizer неправильно парсит числа с запятыми в  
дробной части

# Решение проблемы



## Задание 9

Реализована сериализация с использованием Serializable

Добавлен implements Serializable в классы – FunctionPoint, ArrayTabulatedFunction, LinkedListTabulatedFunction

Протестирована сериализация и десериализация функций

```
TabulatedFunctions.java Main.java FunctionPoint.java x
1 package functions;
2
3 import java.io.Serializable;
4
5 public class FunctionPoint implements Serializable { 50 usages
6     private static final long serialVersionUID = 1L; no usages
7     private double x; 5 usages
8     private double y; 5 usages
9
10    //создаёт объект точки с заданными координатами
```

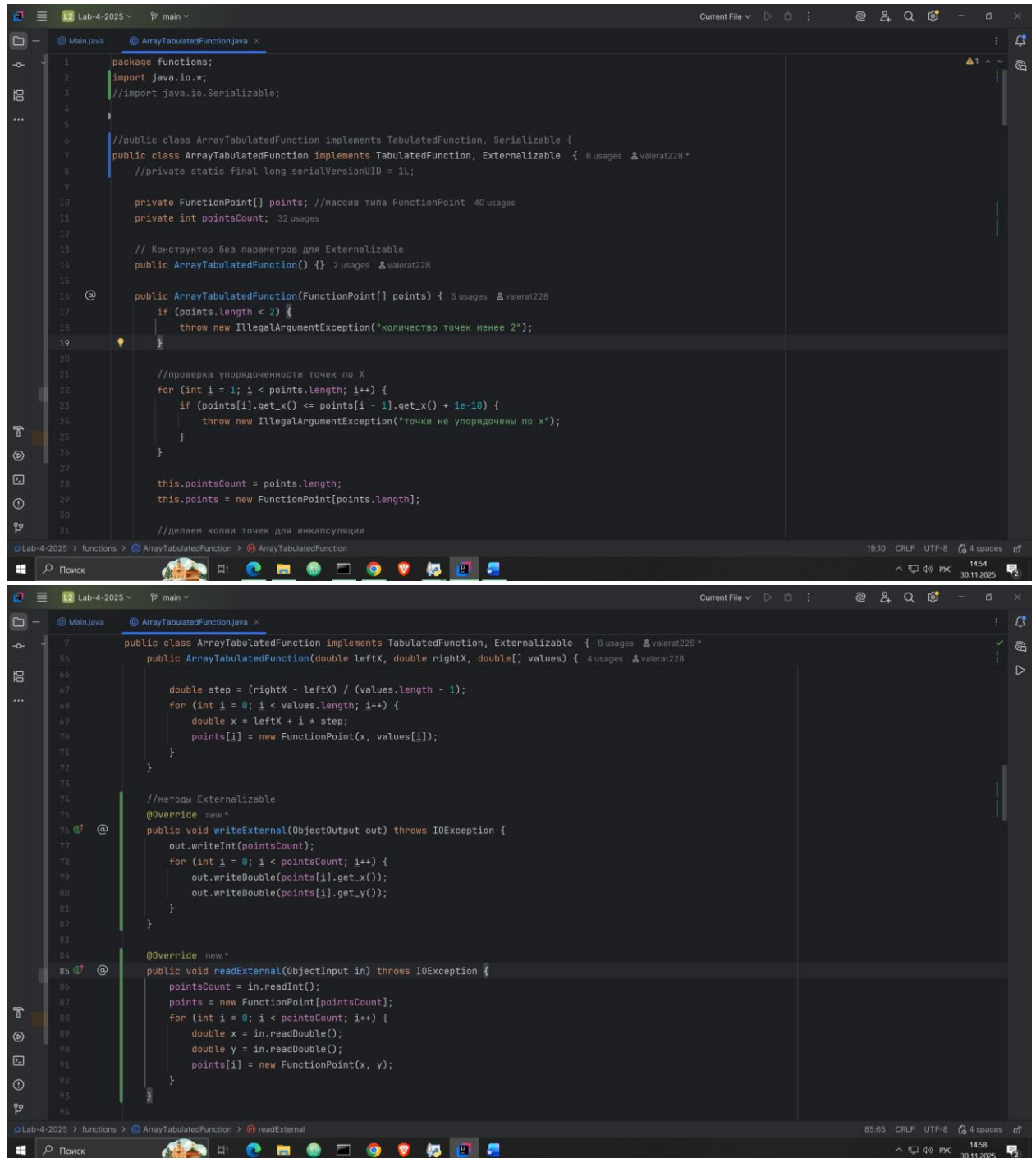
```
TabulatedFunctions.java Main.java FunctionPoint.java ArrayTabulatedFunction.java x
1 package functions;
2
3 import java.io.Serializable;
4
5 public class ArrayTabulatedFunction implements TabulatedFunction, Serializable {
6     private static final long serialVersionUID = 1L; 7 usages
7
8     private FunctionPoint[] points; //массив типа FunctionPoint 36 usages
```

```
TabulatedFunctions.java x Main.java FunctionPoint.java ArrayTabulatedFunction.java LinkedListTabulatedFunction.java x
1 package functions;
2 import java.io.Serializable;
3 public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable { 4 usages
4     private static final long serialVersionUID = 1L; no usages
5
6     private static class FunctionNode implements Serializable { 25 usages
7         private static final long serialVersionUID = 1L; no usages
8         FunctionPoint point; 21 usages
9         FunctionNode prev; 13 usages
10        FunctionNode next; 17 usages
11
12        //конструктор узла
```





## Изменения – ArrayTabulatedFunction сделан Externalizable



The image displays two screenshots of an IDE (IntelliJ IDEA) showing the implementation of the `Externalizable` interface for the `ArrayTabulatedFunction` class. The top screenshot shows the class declaration and the constructor, while the bottom screenshot shows the `writeExternal` and `readExternal` methods.

```
package functions;
import java.io.*;
//import java.io.Serializable;

//public class ArrayTabulatedFunction implements TabulatedFunction, Serializable {
public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable { 8 usages  valerat228
    //private static final long serialVersionUID = 1L;

    private FunctionPoint[] points; //массив типа FunctionPoint 40 usages
    private int pointsCount; 32 usages

    // Конструктор без параметров для Externalizable
    public ArrayTabulatedFunction() {} 2 usages  valerat228

    public ArrayTabulatedFunction(FunctionPoint[] points) { 5 usages  valerat228
        if (points.length < 2) {
            throw new IllegalArgumentException("количество точек менее 2");
        }

        //проверка упорядоченности точек по X
        for (int i = 1; i < points.length; i++) {
            if (points[i].get_x() <= points[i - 1].get_x() + 1e-10) {
                throw new IllegalArgumentException("точки не упорядочены по x");
            }
        }

        this.pointsCount = points.length;
        this.points = new FunctionPoint[points.length];

        //делаем копии точек для инкапсуляции
    }
}
```

```
public ArrayTabulatedFunction(double leftX, double rightX, double[] values) { 4 usages  valerat228
    double step = (rightX - leftX) / (values.length - 1);
    for (int i = 0; i < values.length; i++) {
        double x = leftX + i * step;
        points[i] = new FunctionPoint(x, values[i]);
    }
}

//методы Externalizable
@Override new *
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(pointsCount);
    for (int i = 0; i < pointsCount; i++) {
        out.writeDouble(points[i].get_x());
        out.writeDouble(points[i].get_y());
    }
}

@Override new *
public void readExternal(ObjectInput in) throws IOException {
    pointsCount = in.readInt();
    points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        points[i] = new FunctionPoint(x, y);
    }
}
```

# Новый результат

```
Lab-4-2025  main  Current File  Run  Main  x

Main.java  ArrayTabulatedFunction.java

13 public class Main { @SuppressWarnings
14     public static void main(String[] args) throws InappropriateFunctionPointException { @SuppressWarnings
15
16         System.out.println("\n=== ТЕСТ 16: СЕРИАЛИЗАЦИЯ ===");
17
18         //создан композиция ln(exp(x)) - должна быть близка к x
19         Exp expFunc = new Exp();
20         Log lnFunc = new Log(Math.E);
21         Function composition = Functions.composition(expFunc, lnFunc);
22
23         //табулирует композицию от 0 до 10 с шагом 1
24         TabulatedFunction tabulatedComposition = TabulatedFunctions.tabulate(composition, leftX: 0, rightX: 10, pointsCount: 11);
25
26         System.out.println("Исходная функция ln(exp(x)) на [0, 10]:");
27         for (int i = 0; i < tabulatedComposition.getPointsCount(); i++) {
28             double x = tabulatedComposition.getPointX(i);
29             double y = tabulatedComposition.getPointY(i);
30             System.out.printf("x=%f, ln(exp(x))=%f (должно быть %f)", x, y, x);
31         }
32
33         System.out.println("\n=== Serializable ===");
34         //сериализуем с использованием Serializable (LinkedListTabulatedFunction)
35         TabulatedFunction serializableFunc = new LinkedListTabulatedFunction(leftX: 0, rightX: 10, pointsCount: 11);
36         //заполняем значениями
37         for (int i = 0; i < serializableFunc.getPointsCount(); i++) {
38             double x = serializableFunc.getPointX(i);
39             double y = composition.getFunctionValue(x);
40             serializableFunc.setPointY(i, y);
41         }
42
43         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("serializable.dat"))) {
44             oos.writeObject(serializableFunc);
45             System.out.println("Функция сериализована в serializable.dat (Serializable)");
46         }
47
48         //десериализуем
49         TabulatedFunction deserializedSerializable;
50         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("serializable.dat"))) {
51             deserializedSerializable = (TabulatedFunction) ois.readObject();
52             System.out.println("Функция десериализована из serializable.dat");
53         }
54
55         System.out.println("\n=== Externalizable ===");
56         //сериализуем с использованием Externalizable (ArrayTabulatedFunction)
57         TabulatedFunction externalizableFunc = new ArrayTabulatedFunction(leftX: 0, rightX: 10, pointsCount: 11);
58         //заполняем значениями
59         for (int i = 0; i < externalizableFunc.getPointsCount(); i++) {
60             double x = externalizableFunc.getPointX(i);
61             double y = composition.getFunctionValue(x);
62             externalizableFunc.setPointY(i, y);
63         }
64
65         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("externalizable.dat"))) {
66             oos.writeObject(externalizableFunc);
67             System.out.println("Функция сериализована в externalizable.dat (Externalizable)");
68         }
69
70         //десериализуем
71         TabulatedFunction deserializedExternalizable;
72         try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("externalizable.dat"))) {
73             deserializedExternalizable = (TabulatedFunction) ois.readObject();
74             System.out.println("Функция десериализована из externalizable.dat");
75         }
76
77         //сравнение всех функций
78         System.out.println("\nСравнение всех функций:");
79         boolean allMatch = true;
80         for (int i = 0; i < tabulatedComposition.getPointsCount(); i++) {
81             double original = tabulatedComposition.getPointY(i);
82             double serializable = deserializedSerializable.getPointY(i);
83             double externalizable = deserializedExternalizable.getPointY(i);
84             if (original != serializable || original != externalizable) {
85                 allMatch = false;
86             }
87         }
88         System.out.println("Все значения совпадают: " + allMatch);
89     }
90 }
```

```
Run  Main  x

=== ТЕСТ 16: СЕРИАЛИЗАЦИЯ ===
Исходная функция ln(exp(x)) на [0, 10]:
x=0.0: ln(exp(x))=0.0000 (должно быть 0.0)
x=1.0: ln(exp(x))=1.0000 (должно быть 1.0)
x=2.0: ln(exp(x))=2.0000 (должно быть 2.0)
x=3.0: ln(exp(x))=3.0000 (должно быть 3.0)
x=4.0: ln(exp(x))=4.0000 (должно быть 4.0)
x=5.0: ln(exp(x))=5.0000 (должно быть 5.0)
x=6.0: ln(exp(x))=6.0000 (должно быть 6.0)
x=7.0: ln(exp(x))=7.0000 (должно быть 7.0)
x=8.0: ln(exp(x))=8.0000 (должно быть 8.0)
x=9.0: ln(exp(x))=9.0000 (должно быть 9.0)
x=10.0: ln(exp(x))=10.0000 (должно быть 10.0)

=== Serializable ===
функция сериализована в serializable.dat (Serializable)
функция десериализована из serializable.dat

=== Externalizable ===
функция сериализована в externalizable.dat (Externalizable)
функция десериализована из externalizable.dat

Сравнение всех функций:
точка 0: исходная=0.0000, ser=0.0000(err=0.000000), ext=0.0000(err=0.000000)
точка 1: исходная=1.0000, ser=1.0000(err=0.000000), ext=1.0000(err=0.000000)
точка 2: исходная=2.0000, ser=2.0000(err=0.000000), ext=2.0000(err=0.000000)
точка 3: исходная=3.0000, ser=3.0000(err=0.000000), ext=3.0000(err=0.000000)
точка 4: исходная=4.0000, ser=4.0000(err=0.000000), ext=4.0000(err=0.000000)
точка 5: исходная=5.0000, ser=5.0000(err=0.000000), ext=5.0000(err=0.000000)
точка 6: исходная=6.0000, ser=6.0000(err=0.000000), ext=6.0000(err=0.000000)
точка 7: исходная=7.0000, ser=7.0000(err=0.000000), ext=7.0000(err=0.000000)
точка 8: исходная=8.0000, ser=8.0000(err=0.000000), ext=8.0000(err=0.000000)
точка 9: исходная=9.0000, ser=9.0000(err=0.000000), ext=9.0000(err=0.000000)
точка 10: исходная=10.0000, ser=10.0000(err=0.000000), ext=10.0000(err=0.000000)
Все значения совпадают: true

Сравнение размеров файлов:
Serializable (serializable.dat): 734 байт
Externalizable (externalizable.dat): 236 байт
Бинарный формат (test_binary.dat): 84 байт
Текстовый формат (test_text.txt): 168 байт

=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ===
Process finished with exit code 0
```

