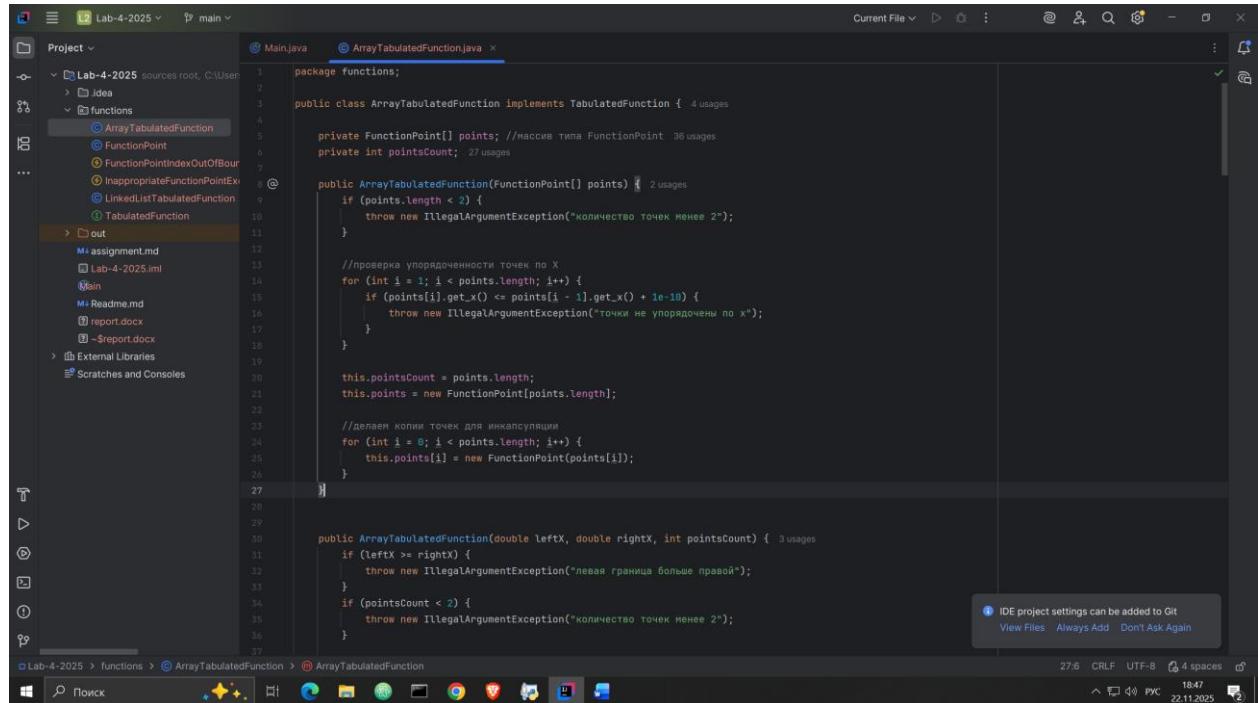


Отчёт по лабораторной работе №4
Тенигин Валерий 6204-010302D

Задание 1 – Добавлены конструкторы согласно заданию



```
package functions;

public class ArrayTabulatedFunction implements TabulatedFunction { 4 usages

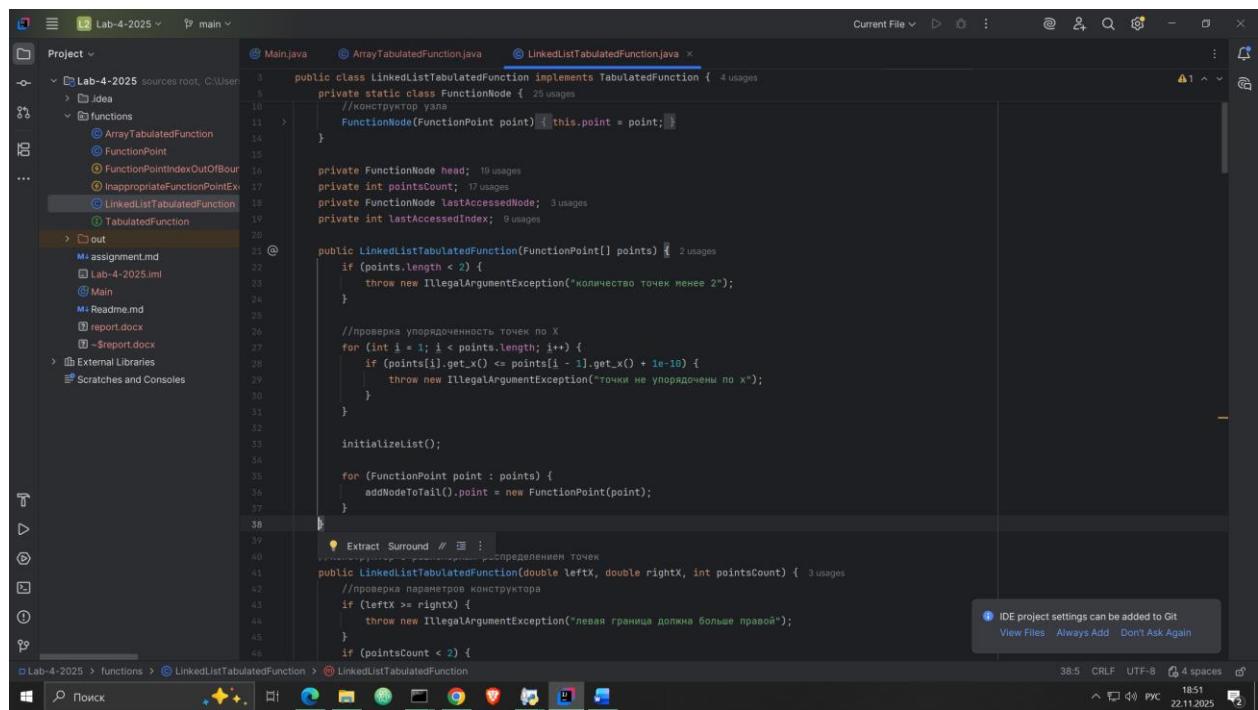
    private FunctionPoint[] points; //массив типа FunctionPoint 36 usages
    private int pointsCount; 27 usages

    public ArrayTabulatedFunction(FunctionPoint[] points) { 2 usages
        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];
    }

    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) { 3 usages
        if (leftX > rightX) {
            throw new IllegalArgumentException("левая граница должна быть меньше правой");
        }
        if (pointsCount < 2) {
            throw new IllegalArgumentException("количество точек менее 2");
        }
    }
}
```



```
public class LinkedListTabulatedFunction implements TabulatedFunction { 4 usages

    private static class FunctionNode { 25 usages
        //конструктор узла
        FunctionNode(FunctionPoint point) { this.point = point; }
    }

    private FunctionNode head; 19 usages
    private int pointsCount; 17 usages
    private FunctionNode lastAccessedNode; 3 usages
    private int lastAccessedIndex; 9 usages

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

        //проверка упорядоченность 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);
    }

    Extract Surround // [ ] : определение точек
    public LinkedListTabulatedFunction(double leftX, double rightX, int pointsCount) { 3 usages
        //проверка параметров конструктора
        if (leftX > rightX) {
            throw new IllegalArgumentException("левая граница должна быть меньше правой");
        }
        if (pointsCount < 2) {
    }
```

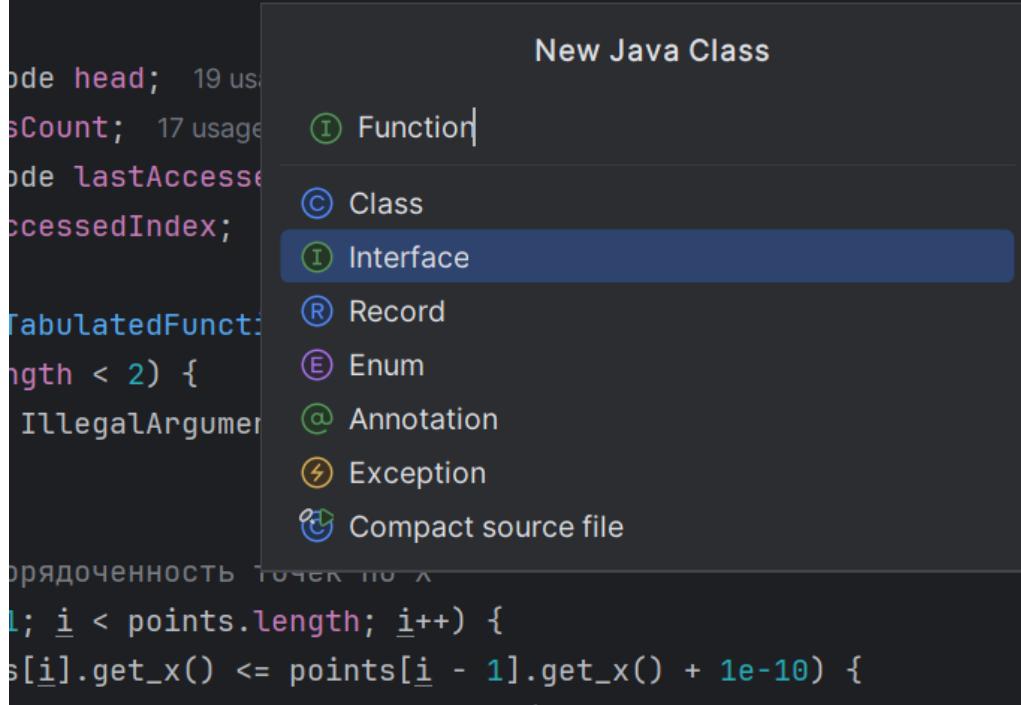
Задание 2

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

Интерфейс TabulatedFunction теперь расширяет Function

Удалены дублирующие методы из TabulatedFunction

```
узла
FunctionPoint point) { this.point = point; }
```



```
in <

  © LinkedListTabulatedFunction.java ×  ⓘ Function.java ×

1 package functions;
2
3 public interface Function { no usages
4     double getLeftDomainBorder();
5     double getRightDomainBorder();
6     double getFunctionValue(double x);
7 }
8
```

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

The image shows two side-by-side Java code editors, likely from the IntelliJ IDEA IDE, displaying the `Exp.java` and `Log.java` files respectively.

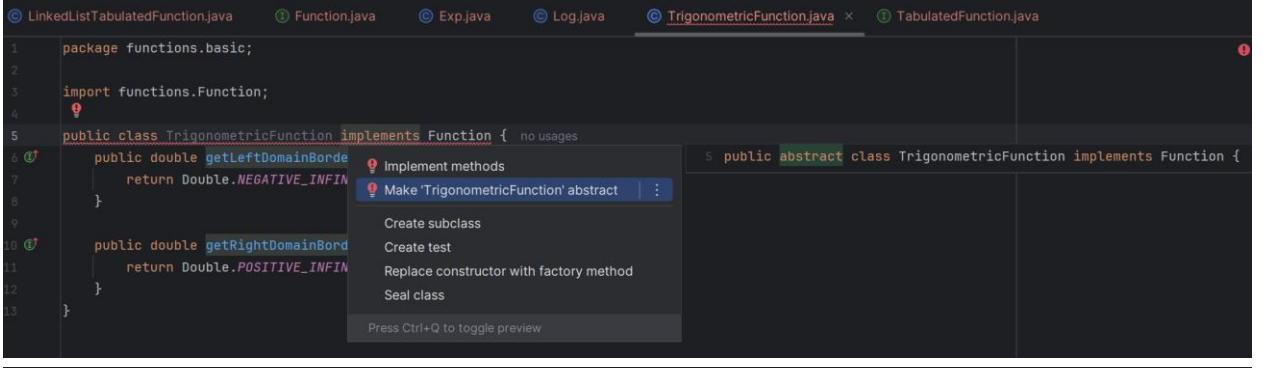
Top Editor (Exp.java):

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Exp implements Function { no usages
6     public double getLeftDomainBorder() { 5 usages
7         return Double.NEGATIVE_INFINITY;
8     }
9
10    public double getRightDomainBorder() { 5 usages
11        return Double.POSITIVE_INFINITY;
12    }
13
14    public double getFunctionValue(double x) { 2 usages
15        return Math.exp(x);
16    }
17}
```

Bottom Editor (Log.java):

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class Log implements Function { no usages
6     private double base; 3 usages
7
8     public Log(double base) { 2 usages
9         if (base <= 0 || base == 1) {
10             throw new IllegalArgumentException("основание логарифма должно быть > 0 и ≠ 1");
11         }
12         this.base = base;
13     }
14
15     public double getLeftDomainBorder() { 5 usages
16         return 0;
17     }
18
19     public double getRightDomainBorder() { 5 usages
20         return Double.POSITIVE_INFINITY;
21     }
22
23     public double getFunctionValue(double x) { 2 usages
24         if (x <= 0) return Double.NaN;
25         return Math.log(x) / Math.log(base);
26     }
27
28     public double getBase() { no usages
29         return base;
30     }
31}
```

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



The screenshot shows an IDE interface with several tabs at the top: LinkedListTabulatedFunction.java, Function.java, Exp.java, Log.java, TrigonometricFunction.java (which is the active tab), and TabulatedFunction.java. In the code editor, there is a class definition:

```
1 package functions.basic;
2
3 import functions.Function;
4
5 public class TrigonometricFunction implements Function { no usages
6     public double getLeftDomainBorder() {
7         return Double.NEGATIVE_INFINITY;
8     }
9
10    public double getRightDomainBorder() {
11        return Double.POSITIVE_INFINITY;
12    }
13}
```

A context menu is open over the class definition, listing the following options:

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

Below the code editor, a preview pane shows the modified code after selecting the "Make 'TrigonometricFunction' abstract" option:

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

public abstract class TrigonometricFunction implements Function { no usages
    public double getLeftDomainBorder() { 5 usages
        return Double.NEGATIVE_INFINITY;
    }

    public double getRightDomainBorder() { 5 usages
        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.java
```

```
ser: 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     }

© LinkedListTabulatedFunction.java   ⚡ Function.java   © Exp.java
```

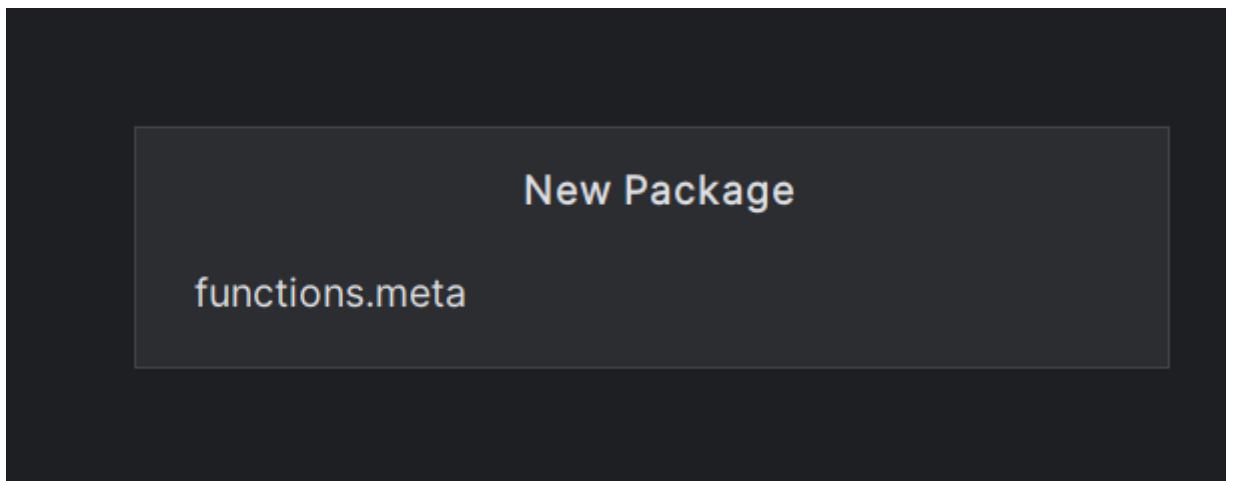
```
er: 1 package functions.basic;
2
3     public class Tan extends TrigonometricFunction { no usages
4     ⚡↑         public double getFunctionValue(double x) {
5             ⚡ return Math.tan(x);
6         }
7     }

© LinkedListTabulatedFunction.java   ⚡ Function.java   © Exp.java
```

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)



Project Lab-4-2025 main

```
Sum.java Power.java Mult.java Composition.java Scale.java Shift.java
```

```
1 package functions.meta;
2
3 import functions.Function;
4
5 // складываем две функции
6 public class Sum implements Function {  no 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 правую левую 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 }
```

Lab-4-2025 > functions > meta > Sum > getRightDomainBorder

Project Lab-4-2025 main

```
Sum.java Power.java Mult.java Composition.java Scale.java Shift.java
```

```
1 package functions.meta;
2
3 import functions.Function;
4
5 // возодим функцию в степень
6 public class Power implements Function {  no 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 }
```

Lab-4-2025 > functions > meta > Power > getLeftDomainBorder

```
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 //заключаем 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 }
```

Lab-4-2025 main

```

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 { 6 usages
7     private Function f; //исходная функция 6 usages
8     private double scaleX; //во сколько раз растянуть по x 4 usages
9     private double scaleY; //во сколько раз растянуть по y 2 usages
10
11     public Scale(Function f, double scaleX, double scaleY) { 2 usages
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 }

```

Lab-4-2025 > functions > meta > Scale > getRightDomainBorder

22:40 CRLF UTF-8 4 spaces

Лабораторная работа №4

Лабораторная работа №4

Lab-4-2025 main

```

1 package functions.meta;
2
3 import functions.Function;
4
5 //сдвигаем функцию влево-вправо и вверх-вниз
6 public class Shift implements Function { 6 usages
7     private Function f; //исходная функция 6 usages
8     private double shiftX; //на сколько сдвинуть по x 4 usages
9     private double shiftY; //на сколько сдвинуть по y 2 usages
10
11     public Shift(Function f, double shiftX, double shiftY) { 2 usages
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 }

```

Lab-4-2025 > functions > meta > Shift > getFunctionValue

31:31 CRLF UTF-8 4 spaces

Задание 5

Создан класс Functions со статическими методами-обёртками

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

```
package functions;

import functions.meta.*;

//класс содержащий вспомогательные статические методы для работы с функциями
public final class Functions { }

//запрещаем создавать объекты этого класса
private Functions() { }

throw new AssertionException(detailMessage: "нельзя создать экземпляр класса Functions");

}

//создает функцию сдвигнутую по оси x и y , исходная функция(x - shiftX) + shiftY
public static Function shift(Function f, double shiftX, double shiftY) { }

return new Shift(f, shiftX, shiftY);

}

//создает функцию масштабированную по осям x и y , scaleY * исходная функция(x / scaleX)
public static Function scale(Function f, double scaleX, double scaleY) { }

return new Scale(f, scaleX, scaleY);

}

//создает функцию возведенную в степень
public static Function power(Function f, double power) { }

return new Power(f, power);

}

//создает функцию - сумму двух функций
public static Function sum(Function f1, Function f2) { }

return new Sum(f1, f2);

}

//создает функцию - произведение двух функций
public static Function mult(Function f1, Function f2) { }

return new Mult(f1, f2);

}

//создает композицию двух функций, f2(f1(x))
public static Function composition(Function f1, Function f2) { }

return new Composition(f1, f2);

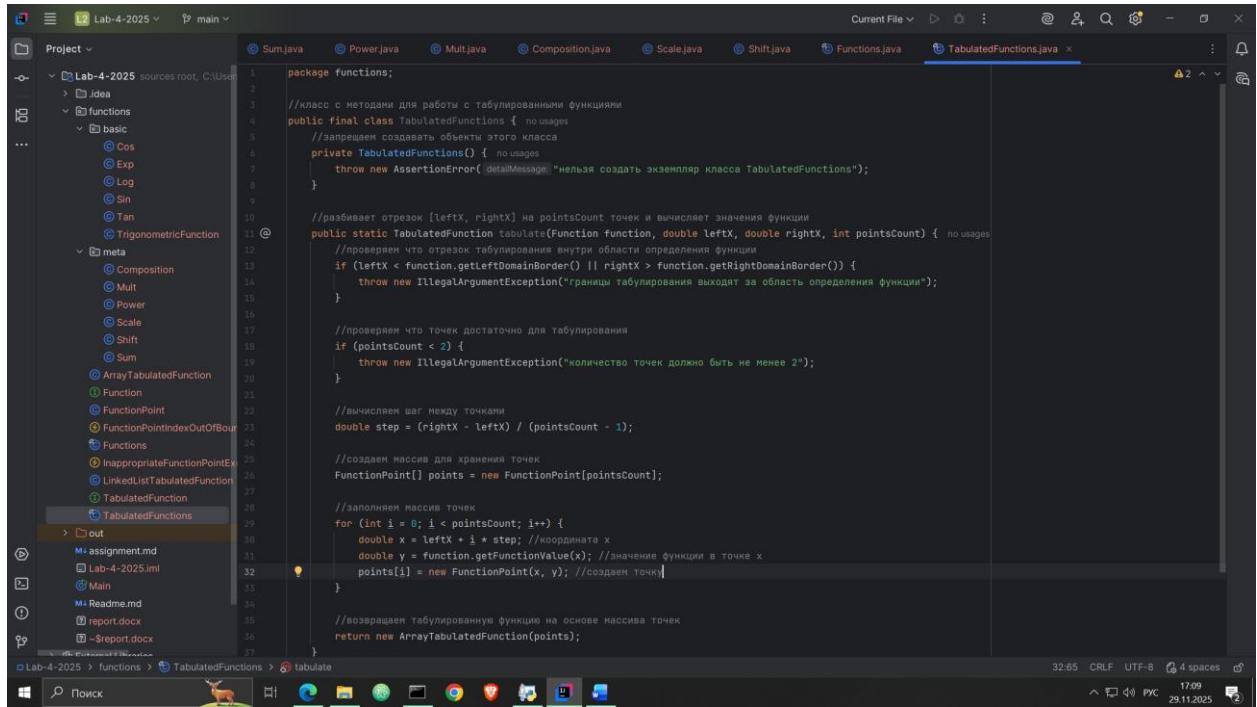
}
```

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



```
1 package functions;
2
3 //класс с методами для работы с табулированными функциями
4 public final class TabulatedFunctions { no usages
5     //запрещаем создавать объекты этого класса
6     private TabulatedFunctions() { no usages
7         throw new AssertionError("нельзя создать экземпляр класса TabulatedFunctions");
8     }
9
10    //разбивает отрезок [leftX, rightX] на pointsCount точек и вычисляет значения функции
11    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) { no usages
12        //проверяем что отрезок табулирования внутри области определения функции
13        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
14            throw new IllegalArgumentException("границы табулирования выходят за область определения функции");
15        }
16
17        //проверяем что точек достаточно для табулирования
18        if (pointsCount < 2) {
19            throw new IllegalArgumentException("количество точек должно быть не менее 2");
20        }
21
22        //вычисляем шаг между точками
23        double step = (rightX - leftX) / (pointsCount - 1);
24
25        //создаем массив для хранения точек
26        FunctionPoint[] points = new FunctionPoint[pointsCount];
27
28        //заполняем массив точек
29        for (int i = 0; i < pointsCount; i++) {
30            double x = leftX + i * step; //координата x
31            double y = function.getFunctionValue(x); //значение функции в точке x
32            points[i] = new FunctionPoint(x, y); //создаем точку
33        }
34
35        //возвращаем табулированную функцию на основе массива точек
36        return new ArrayTabulatedFunction(points);
37    }
38}
```

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

The image shows two side-by-side screenshots of a Java IDE (IntelliJ IDEA) displaying the same code file, `TabulatedFunctions.java`, in two different states of completion.

Left Screenshot: Shows the initial state of the code. The class `TabulatedFunctions` is defined with a constructor that throws an `AssertionError` if instantiated. It contains a static method `tabulate` which takes a function and returns a `FunctionPoint[]`. The code includes validation for domain boundaries and point count.

```
package functions;
import java.io.*;
public final class TabulatedFunctions { no usages
    // запрещаем создавать объекты этого класса
    private TabulatedFunctions() { no usages
        throw new AssertionError(detailMessage: "нельзя создать экземпляр класса TabulatedFunctions");
    }
    // создает табулированную функцию из аналитической функции
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) { no usages
        // проверим что отрезок табулирования внутри области определения функции
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("границы табулирования выходят за область определения функции");
        }
        // проверим что точек достаточно для табулирования
        if (pointsCount < 2) {
            throw new IllegalArgumentException("количество точек должно быть не менее 2");
        }
        // вычисляем шаг между точками
        double step = (rightX - leftX) / (pointsCount - 1);
        // создаем массив для хранения точек
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        // заполняем массив точек
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step; // координата x
            double y = function.getFunctionValue(x); // значение функции в точке x
            points[i] = new FunctionPoint(x, y); // создаем точку
        }
        // возвращаем табулированную функцию на основе массива точек
    }
}
```

Right Screenshot: Shows the completed state of the code. The `tabulate` method now returns a `ArrayTabulatedFunction`. The `outputTabulatedFunction` and `inputTabulatedFunction` methods have been implemented to write and read tabulated functions from a byte stream.

```
public final class TabulatedFunctions { no usages
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) { no usages
        return new ArrayTabulatedFunction(points);
    }
    // записывает табулированную функцию в байтовый поток
    public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException { no usages
        DataOutputStream dos = new DataOutputStream(out); // обертка для удобной записи примитивов
        // записываем количество точек
        dos.writeInt(function.getPointsCount());
        // записываем все точки (x и y для каждой)
        for (int i = 0; i < function.getPointsCount(); i++) {
            dos.writeDouble(function.getPointX(i));
            dos.writeDouble(function.getPointY(i));
        }
        dos.flush(); // прогоняем данные в поток
        // не закрываем поток - это ответственность вызывающего кода
    }
    // читает табулированную функцию из байтового потока
    public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException { no usages
        DataInputStream dis = new DataInputStream(in); // обертка для удобного чтения примитивов
        // читаем количество точек
        int pointsCount = dis.readInt();
        FunctionPoint[] points = new FunctionPoint[pointsCount];
        // читаем все точки
        for (int i = 0; i < pointsCount; i++) {
            double x = dis.readDouble();
            double y = dis.readDouble();
            points[i] = new FunctionPoint(x, y);
        }
        // создаем и возвращаем табулированную функцию
    }
}
```

Lab-4-2025

```

public final class TabulatedFunctions {
    public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
        // создаем и возвращаем табулированную функцию
        return new ArrayTabulatedFunction(points);
    }

    // записывает табулированную функцию в системный поток
    public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
        PrintWriter writer = new PrintWriter(out); // обертка для удобной записи текста

        // записываем количество точек
        writer.print(function.getPointsCount());

        // записываем все точки через пробел
        for (int i = 0; i < function.getPointsCount(); i++) {
            writer.print(" " + function.getPointX(i));
            writer.print(" " + function.getPointY(i));
        }

        writer.flush(); // проталкиваем данные в поток
    }

    // читает табулированную функцию из символьного потока
    public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
        StreamTokenizer tokenizer = new StreamTokenizer(in); // разбивает поток на токены
        tokenizer.parseNumbers(); // говорит что числа нужно парсить как числа

        // читаем количество точек (первый токен должен быть числом)
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
            throw new IOException("ожидалось число (количество точек)");
        }
        int pointsCount = (int) tokenizer.nval;

        FunctionPoint[] points = new FunctionPoint[pointsCount];

        // читаем все точки (пары x y)
        for (int i = 0; i < pointsCount; i++) {
            // читаем x
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("ожидалось число для x в точке " + i);
            }
            double x = tokenizer.nval;

            // читаем y
            if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) {
                throw new IOException("ожидалось число для y в точке " + i);
            }
            double y = tokenizer.nval;

            points[i] = new FunctionPoint(x, y);
        }

        // создаем и возвращаем табулированную функцию
        return new ArrayTabulatedFunction(points);
    }
}

```

Задание 8

The screenshot shows a Java IDE interface with two tabs: `TabulatedFunctions.java` and `Main.java`. The `TabulatedFunctions.java` tab is active. The code is a `Main` class with a static void `main` method. It imports `InappropriateFunctionPointException` from a package. The code includes several `System.out.println` statements to test various functions like `Sin`, `Cos`, `Exp`, `Log`, `Tan`, and `TrigonometricFunction`. A tooltip is visible over the `getFunctionValue` method call, indicating it is specified by the `Function` interface. The code is annotated with comments for tests 10 and 11.

```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("==> ТЕСТ ДЛЯ ЛАБОРАТОРНОЙ №4 ==>");
        System.out.println("==>".repeat(count 50));

        try {
            //тест аналитических функций (задание 3)
            System.out.println("==> ТЕСТ 10: АНАЛИТИЧЕСКИЕ ФУНКЦИИ ==>");

            Sin sin = new Sin();
            Cos cos = new Cos();
            Exp exp = new Exp();
            Log ln = new Log(Math.E);

            System.out.println("sin(n/2) = " + sin.getFunctionValue(Math.PI/2));
            System.out.println("cos(0) = " + cos.getFunctionValue(0));
            System.out.println("exp(1) = " + exp.getFunctionValue(1));
            System.out.println("ln(e) = " + ln.getFunctionValue(Math.E));

            System.out.println("область определения sin: [" + sin.getRightDomainBorder() + "]");
            System.out.println("область определения ln: [" + ln.getRightDomainBorder() + "]");

            //тест мета-функций (задание 4)
            System.out.println("==> ТЕСТ 11: МЕТА-ФУНКЦИИ ==>");

            Function sumFunc = new Sum(sin, cos);
            Function multFunc = new Mult(sin, cos);
            Function powerFunc = new Power(sin, power: 2);
            Function shiftFunc = new Shift(sin, shiftX: 1, shiftY: 0.5);
            Function scaleFunc = new Scale(sin, scaleX: 2, scaleY: 3);
            Function compFunc = new Composition(sin, cos);

            System.out.println("sin(1) + cos(1) = " + sumFunc.getFunctionValue(1));
            System.out.println("sin(1) * cos(1) = " + multFunc.getFunctionValue(1));
            System.out.println("sin^2(1) = " + powerFunc.getFunctionValue(1));
            System.out.println("sin(1-1) + 0.5 = " + shiftFunc.getFunctionValue(1));
            System.out.println("3 * sin(1/2) = " + scaleFunc.getFunctionValue(1));
        }
    }
}
```

The screenshot shows a Java IDE interface with two tabs: `TabulatedFunctions.java` and `Main.java`. The `TabulatedFunctions.java` tab is active. The code is a `Main` class with a static void `main` method. It imports `InappropriateFunctionPointException` from a package. The code includes several `System.out.println` statements to test methods of the `Functions` class, such as `sum`, `power`, `shift`, and `composition`. It also demonstrates tabulation of the `sin` function. A tooltip is visible over the `tabulate` method call, indicating it is specified by the `TabulatedFunctions` interface. The code is annotated with comments for tests 12, 13, and 14.

```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("sin(1) * cos(1) = " + sumFunc.getFunctionValue(1));
        System.out.println("sin(1) * cos(1) = " + multFunc.getFunctionValue(1));
        System.out.println("sin^2(1) = " + powerFunc.getFunctionValue(1));
        System.out.println("sin(1-1) + 0.5 = " + shiftFunc.getFunctionValue(1));
        System.out.println("3 * sin(1/2) = " + scaleFunc.getFunctionValue(1));
        System.out.println("sin(cos(1)) = " + compFunc.getFunctionValue(1));

        //тест класса Functions (задание 5)
        System.out.println("==> ТЕСТ 12: КЛАСС Functions ==>");

        Function f1 = Functions.sum(sin, cos);
        Function f2 = Functions.power(sin, power: 2);
        Function f3 = Functions.shift(sin, shiftX: 0.5, shiftY: 1);
        Function f4 = Functions.composition(sin, cos);

        System.out.println("Functions.sum(sin, cos)(1) = " + f1.getFunctionValue(1));
        System.out.println("Functions.power(sin, 2)(1) = " + f2.getFunctionValue(1));
        System.out.println("Functions.shift(sin, 0.5, 1)(1) = " + f3.getFunctionValue(1));
        System.out.println("Functions.composition(sin, cos)(1) = " + f4.getFunctionValue(1));

        //тест табулирования (задание 6)
        System.out.println("==> ТЕСТ 13: ТАБУЛИРОВАНИЕ ==>");

        TabulatedFunction tabulatedSin = TabulatedFunctions.tabulate(sin, left: 0, right: Math.PI, pointsCount: 5);
        System.out.println("табулированный sin на [0, π] с 5 точками");
        for (int i = 0; i < tabulatedSin.getPointsCount(); i++) {
            System.out.printf(" точка %d: (%.2f, %.4f)%n", i, tabulatedSin.getPointX(i), tabulatedSin.getPointY(i));
        }

        //тест ввода/вывода (задание 7)
        System.out.println("==> ТЕСТ 14: ВВОД/ВЫВОД ==>");

        //бинарный формат
        try (FileOutputStream fos = new FileOutputStream(name: "test_binary.dat")) {
            TabulatedFunctions.outputTabulatedFunction(tabulatedSin, fos);
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("\n*** TECT 14: БВД/БвД ***");

        // бинарный формат
        try (FileOutputStream fos = new FileOutputStream( fileName: "test_binary.dat" )) {
            TabulatedFunctions.outputTabulatedFunction(tabulatedSin, fos);
            System.out.println("функция записана в бинарный файл");
        }

        TabulatedFunction readFromBinary;
        try (FileInputStream fis = new FileInputStream( fileName: "test_binary.dat" )) {
            readFromBinary = TabulatedFunctions.inputTabulatedFunction(fis);
            System.out.println("функция прочитана из бинарного файла");
        }

        // текстовый формат
        try (FileWriter fw = new FileWriter( fileName: "test_text.txt" )) {
            TabulatedFunctions.writeTabulatedFunction(tabulatedSin, fw);
            System.out.println("функция записана в текстовый файл");
        }

        TabulatedFunction readFromText;
        try (FileReader fr = new FileReader( fileName: "test_text.txt" )) {
            readFromText = TabulatedFunctions.readTabulatedFunction(fr);
            System.out.println("функция прочитана из текстового файла");
        }

        // Сравнение
        System.out.println("сравнение исходной и считанных функций");
        for (int i = 0; i < tabulatedSin.getPointsCount(); i++) {
            double original = tabulatedSin.getPoint(i);
            double fromBinary = readFromBinary.getPoint(i);
            double fromText = readFromText.getPoint(i);
            System.out.printf("точка %d: исходная=%4f, бинарная=%4f, текстовая=%4f\n",
                i, original, fromBinary, fromText);
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("\n*** TECT 15: СЛОЖНАЯ КОМПОЗИЦИЯ ***");

        // ln(exp(x)) должна быть близка к x
        Function lnOfExp = Functions.composition(exp, ln);
        TabulatedFunction tabulatedLnExp = TabulatedFunctions.tabulate(lnOfExp, leftX: 0.1, rightX: 5, pointsCount: 10);

        System.out.println("\nln(exp(x)) на отрезке [0.1, 5]:");
        for (int i = 0; i < tabulatedLnExp.getPointsCount(); i++) {
            double x = tabulatedLnExp.getPoint(i);
            double y = tabulatedLnExp.getPoint(i);
            double error = Math.abs(x - y);
            System.out.printf("x=%1f: ln(exp(x))=%4f, ошибка=%6f\n", x, y, error);
        }

        System.out.println("\n*** ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ***");

        } catch (Exception e) {
            System.out.println("ошибка в новых тестах: " + e.getMessage());
        }
    }
}
```

```

*** ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ №4 ***
=====
*** ТЕСТ 10: АНАЛИТИЧЕСКИЕ ФУНКЦИИ ***
sin(π/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) = 1.3817732906768363
sin(1) * cos(1) = 0.4546487134128409
sin2(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) = 1.3817732906768363
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, π] 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))=0,6446, 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

```
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException { ... }

//читает табулированную функцию из символьного потока
public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException { ... }

// создает токенайзер для правильной работы с числами с запятой
tokenizer.setDecimalSeparator('.');
tokenizer.wordChars('0', '9'); // цифры
tokenizer.wordChars(' ', '\u043f'); // точка
tokenizer.wordChars(' ', '\u043d'); // ЗНАЧАЩИЕ - знаки!
tokenizer.wordChars('0', '9'); // минус
tokenizer.wordChars('0', '9'); // знаки
tokenizer.wordChars(',', '\u043f'); // запятая
tokenizer.wordChars('-', '\u043f'); // минус
tokenizer.wordChars('/', '\u043f'); // деление
tokenizer.wordChars('*', '\u043f'); // умножение
tokenizer.wordChars('(', '\u043f'); // скобка
tokenizer.wordChars(')', '\u043f'); // конец скобки
tokenizer.wordChars('\n', '\u043f'); // новая строка
tokenizer.wordChars('\r', '\u043f'); // возврат каретки

// игнорирует пробелы
if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
    throw new IOException("запись чисто (количество точек)");
}
int pointsCount = Integer.parseInt(tokenizer.sval);

FunctionPoint[] points = new FunctionPoint[pointsCount];

// считает все точки (x,y)
for (int i = 0; i < pointsCount; i++) {
    // считаем x
    if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
        throw new IOException("запись чисто - число для x точки " + i);
    }
    // заменим запятую на точку для корректного парсинга
    String xstr = tokenizer.sval.replace(',', '.');
    double x = Double.parseDouble(xstr);

    // считаем y
    if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
        throw new IOException("запись чисто - число для y точки " + i);
    }
    // заменим запятую на точку для корректного парсинга
    String ystr = tokenizer.sval.replace(',', '.');
    double y = Double.parseDouble(ystr);

    points[i] = new FunctionPoint(x, y);
}

return new ArrayTabulatedFunction(points);
}

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=0,0000

*** 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))=1,1899, 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
```

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 ×

```
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 ×

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

⌚ TabulatedFunctions.java × ⚖ Main.java ⚖ FunctionPoint.java ⚖ ArrayTabulatedFunction.java ⚖ LinkedListTabulatedFunction.java ×

```
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
```

```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("n==== ТЕСТ 16: СЕРИАЛИЗАЦИЯ ====");

        //создаем композицию ln(exp(x)) - должна быть близка к x
        Exp expFunc = new Exp();
        Log lnFunc = new Log(Math.E);
        Function composition = Functions.composition(expFunc, lnFunc);

        //табулируем композицию
        TabulatedFunction tabulatedComposition = TabulatedFunctions.tabulate(composition, leftX: 0.1, rightX: 10, pointsCount: 1000);

        System.out.println("Исходная функция ln(exp(x)):");
        for (int i = 0; i < tabulatedComposition.getPointsCount(); i++) {
            System.out.print(" точка №" + (i+1) + ": (" + tabulatedComposition.getPointX(i) + ", " + tabulatedComposition.getPointY(i) + ")");
        }

        //серIALIZУЕМ с использованием Serializable
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new File("serializable.dat")))) {
            oos.writeObject(tabulatedComposition);
            System.out.println("Функция сериализована в serializable.dat");
        }

        //десериализуем
        TabulatedFunction deserializedComposition;
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File("serializable.dat")))) {
            deserializedComposition = (TabulatedFunction) ois.readObject();
            System.out.println("Функция десериализована из serializable.dat");
        }

        //сравниваем исходную и десериализованную функции
        System.out.println("Сравнение исходной и десериализованной функций:");
        boolean allMatch = true;
        for (int i = 0; i < tabulatedComposition.getPointsCount(); i++) {
            if (tabulatedComposition.getPointX(i) != deserializedComposition.getPointX(i) || tabulatedComposition.getPointY(i) != deserializedComposition.getPointY(i)) {
                allMatch = false;
            }
        }

        if (allMatch) {
            System.out.println("СерIALIZАЦИЯ работает правильно - все значения совпадают!");
        } else {
            System.out.println("Ошибка сериализации - значения не совпадают!");
        }

        //анализ файлов
        File serializableFile = new File("serializable.dat");
        File binaryFile = new File("test_binary.dat");
        File textField = new File("test_text.txt");

        System.out.println("\nСравнение размеров файлов:");
        System.out.printf(" СерIALIZАЦИЯ (serializable.dat): %d байт\n", serializableFile.length());
        if (binaryFile.exists()) {
            System.out.printf(" Бинарный формат (test_binary.dat): %d байт\n", binaryFile.length());
        }
        if (textField.exists()) {
            System.out.printf(" Текстовый формат (test_text.txt): %d байт\n", textField.length());
        }

        System.out.println("\n==== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ====");

    } catch (Exception e) {
        System.out.println("ошибка в новых тестах: " + e.getMessage());
        e.printStackTrace();
    }
}
```

== ТЕСТ 16: СЕРИАЛИЗАЦИЯ ==
Исходная функция ln(exp(x)):
точка 0: (0,1, 0,000)
точка 1: (1,1, 1,0900)
точка 2: (2,1, 2,0800)
точка 3: (3,1, 3,0700)
точка 4: (4,1, 4,0600)
точка 5: (5,1, 5,0500)
точка 6: (6,0, 6,0400)
точка 7: (7,0, 7,0300)
точка 8: (8,0, 8,0200)
точка 9: (9,0, 9,0100)
точка 10: (10,0, 10,0000)
функция сериализована в serializable.dat
функция десериализована из serializable.dat
Сравнение исходной и десериализованной функций:
точка 0: исходная=0,0000, десериализованная=0,
точка 1: исходная=1,0900, десериализованная=1,
точка 2: исходная=2,0800, десериализованная=2,
точка 3: исходная=3,0700, десериализованная=3,
точка 4: исходная=4,0600, десериализованная=4,
точка 5: исходная=5,0500, десериализованная=5,
точка 6: исходная=6,0400, десериализованная=6,
точка 7: исходная=7,0300, десериализованная=7,
точка 8: исходная=8,0200, десериализованная=8,
точка 9: исходная=9,0100, десериализованная=9,
точка 10: исходная=10,0000, десериализованная=10
Сериализация работает правильно - все значения с одинаковыми точностями

Сравнение размеров файлов:
СерIALIZАЦИЯ (serializable.dat): 440 байт
Бинарный формат (test_binary.dat): 84 байт
Текстовый формат (test_text.txt): 148 байт

```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("n==== ТЕСТ 17: ОШИБКА СЕРИАЛИЗАЦИИ ====");

        //создаем композицию ln(exp(x)) - должна быть близка к x
        Exp expFunc = new Exp();
        Log lnFunc = new Log(Math.E);
        Function composition = Functions.composition(expFunc, lnFunc);

        //табулируем композицию
        TabulatedFunction tabulatedComposition = TabulatedFunctions.tabulate(composition, leftX: 0.1, rightX: 10, pointsCount: 1000);

        System.out.println("Исходная функция ln(exp(x)):");
        for (int i = 0; i < tabulatedComposition.getPointsCount(); i++) {
            System.out.print(" точка №" + (i+1) + ": (" + tabulatedComposition.getPointX(i) + ", " + tabulatedComposition.getPointY(i) + ")");
        }

        //серIALIZУЕМ с использованием Serializable
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new File("serializable.dat")))) {
            oos.writeObject(tabulatedComposition);
            System.out.println("Функция сериализована в serializable.dat");
        }

        //десериализуем
        TabulatedFunction deserializedComposition;
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File("serializable.dat")))) {
            deserializedComposition = (TabulatedFunction) ois.readObject();
            System.out.println("Функция десериализована из serializable.dat");
        }

        //сравниваем исходную и десериализованную функции
        System.out.println("Сравнение исходной и десериализованной функций:");
        boolean allMatch = true;
        for (int i = 0; i < tabulatedComposition.getPointsCount(); i++) {
            if (tabulatedComposition.getPointX(i) != deserializedComposition.getPointX(i) || tabulatedComposition.getPointY(i) != deserializedComposition.getPointY(i)) {
                allMatch = false;
            }
        }

        if (allMatch) {
            System.out.println("СЕРИАЛИЗАЦИЯ работает правильно - все значения совпадают!");
        } else {
            System.out.println("Ошибка сериализации - значения не совпадают!");
        }

        //анализ файлов
        File serializableFile = new File("serializable.dat");
        File binaryFile = new File("test_binary.dat");
        File textField = new File("test_text.txt");

        System.out.println("\nСравнение размеров файлов:");
        System.out.printf(" СерIALIZАЦИЯ (serializable.dat): %d байт\n", serializableFile.length());
        if (binaryFile.exists()) {
            System.out.printf(" Бинарный формат (test_binary.dat): %d байт\n", binaryFile.length());
        }
        if (textField.exists()) {
            System.out.printf(" Текстовый формат (test_text.txt): %d байт\n", textField.length());
        }

        System.out.println("\n==== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ====");

    } catch (Exception e) {
        System.out.println("ошибка в новых тестах: " + e.getMessage());
        e.printStackTrace();
    }
}
```

== ТЕСТ 17: ОШИБКА СЕРИАЛИЗАЦИИ ==
Исходная функция ln(exp(x)):
точка 0: (1,1, 1,0900)
точка 1: (2,1, 2,0800)
точка 2: (3,1, 3,0700)
точка 3: (4,1, 4,0600)
точка 4: (5,1, 5,0500)
точка 5: (6,0, 6,0400)
точка 6: (7,0, 7,0300)
точка 7: (8,0, 8,0200)
точка 8: (9,0, 9,0100)
точка 10: (10,0, 10,0000)
функция сериализована в serializable.dat
функция десериализована из serializable.dat
Сравнение исходной и десериализованной функций:
точка 0: исходная=0,1000, десериализованная=0,
точка 1: исходная=1,0900, десериализованная=1,
точка 2: исходная=2,0800, десериализованная=2,
точка 3: исходная=3,0700, десериализованная=3,
точка 4: исходная=4,0600, десериализованная=4,
точка 5: исходная=5,0500, десериализованная=5,
точка 6: исходная=6,0400, десериализованная=6,
точка 7: исходная=7,0300, десериализованная=7,
точка 8: исходная=8,0200, десериализованная=8,
точка 9: исходная=9,0100, десериализованная=9,
точка 10: исходная=10,0000, десериализованная=10
Сериализация работает правильно - все значения с одинаковыми точностями

Сравнение размеров файлов:
СерIALIZАЦИЯ (serializable.dat): 440 байт
Бинарный формат (test_binary.dat): 84 байт
Текстовый формат (test_text.txt): 148 байт

== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ==

Process finished with exit code 0