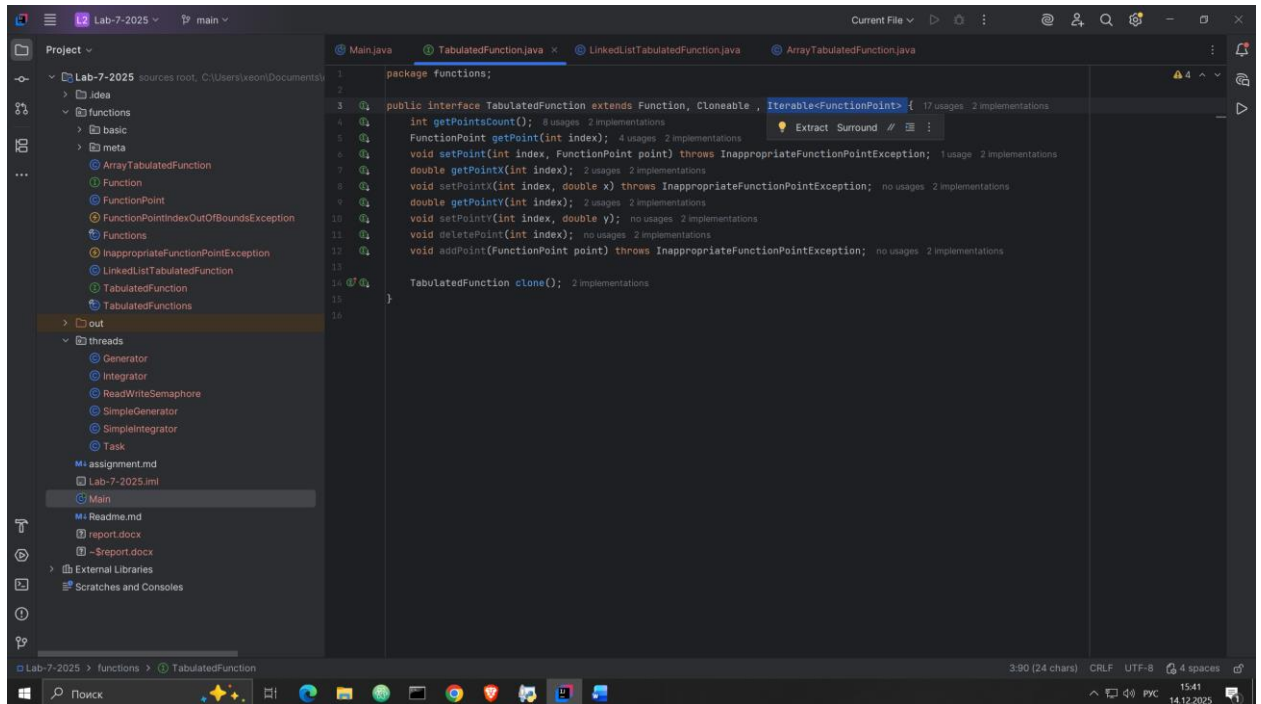


Отчёт по лабораторной работе №7
Тенигин Валерий 6204-010302D

Задание 1

Добавил в интерфейс TabulatedFunction наследование от `Iterable<FunctionPoint>`

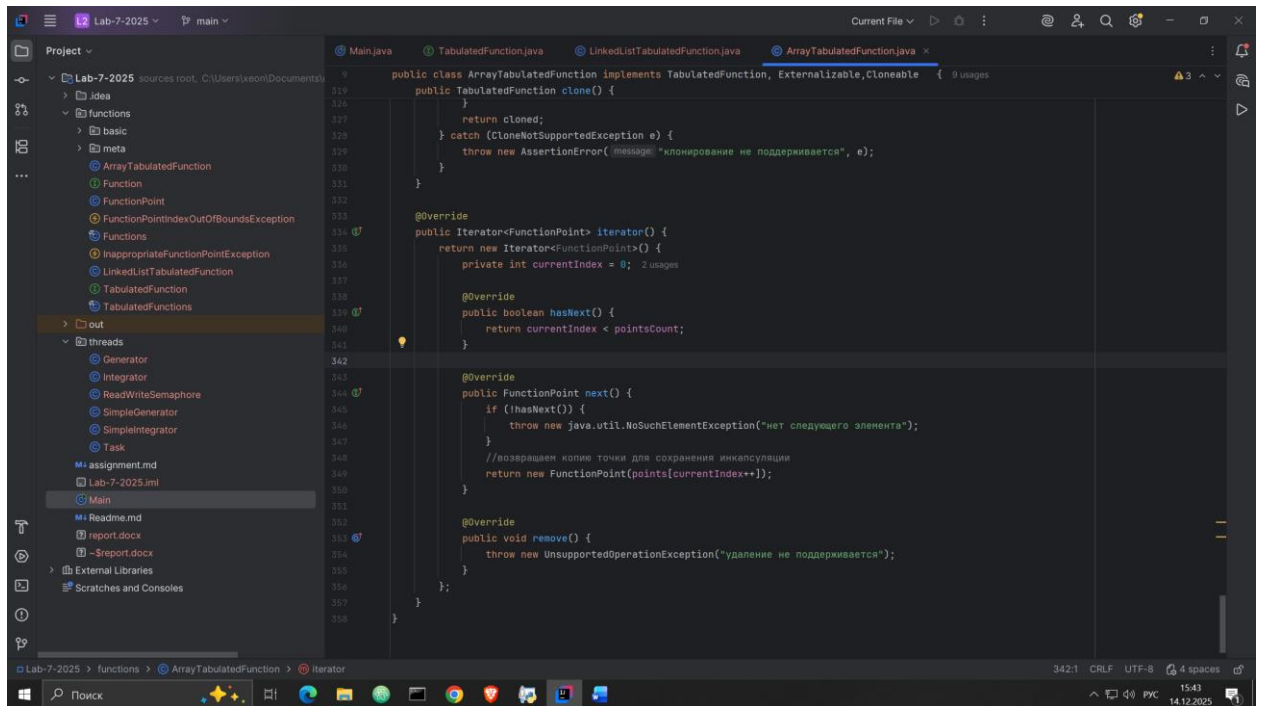


Добавил методы:

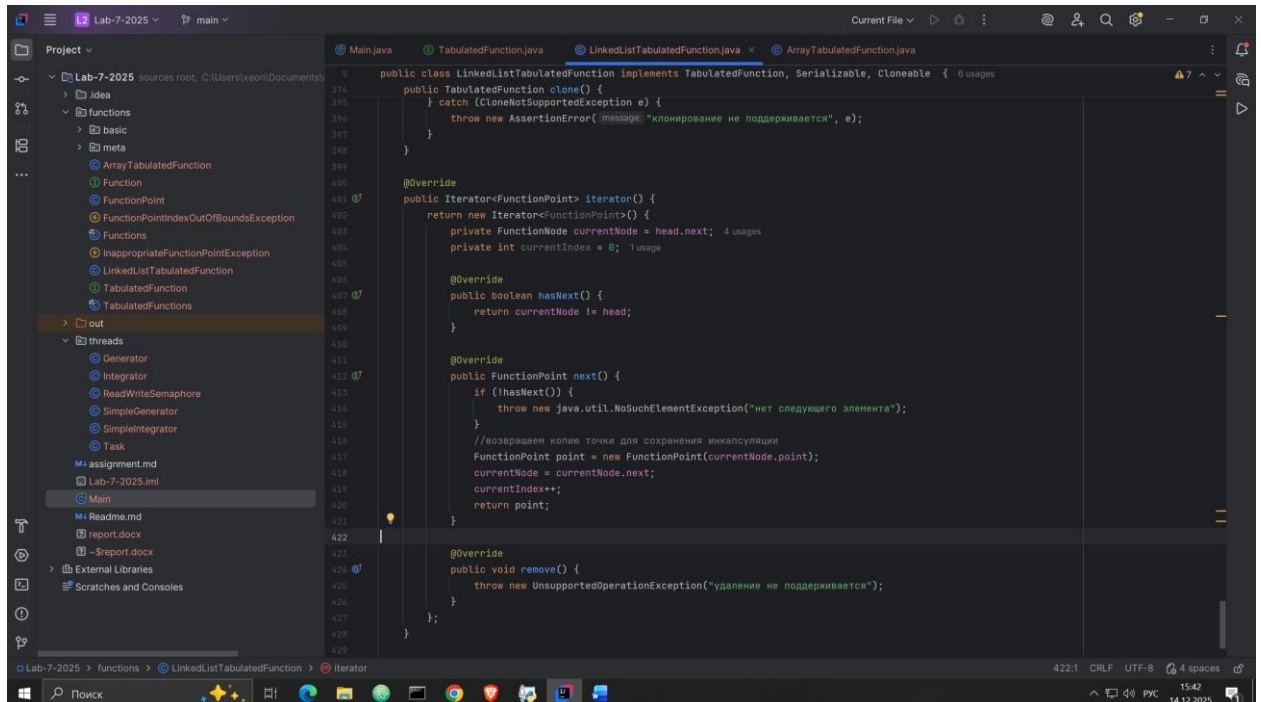
`iterator()` с анонимным классом

`next()` возвращает копии точек для сохранения инкапсуляции и выбрасывает `NoSuchElementException` при отсутствии элементов

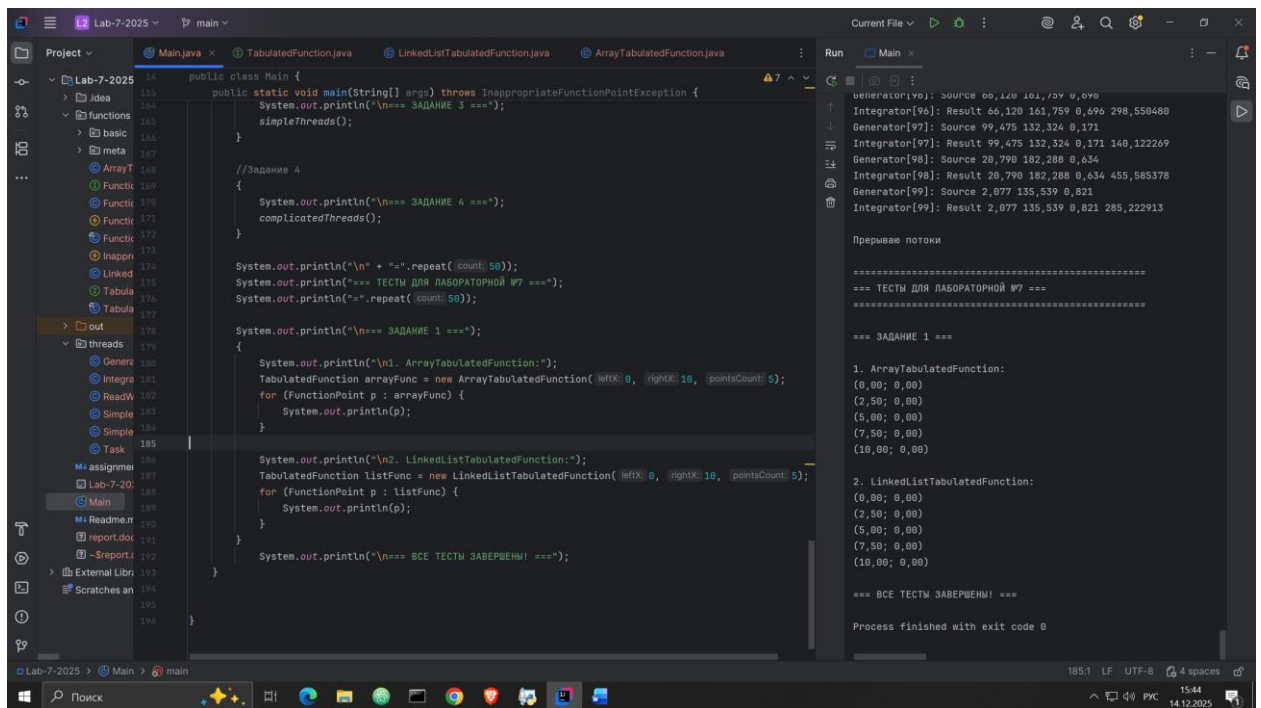
`remove()` выбрасывает `UnsupportedOperationException`



Добавил метод `iterator()` с анонимным классом
`next()` возвращает копии точек для сохранения инкапсуляции и выбрасывает `NoSuchElementException` при отсутствии элементов
`remove()` выбрасывает `UnsupportedOperationException`

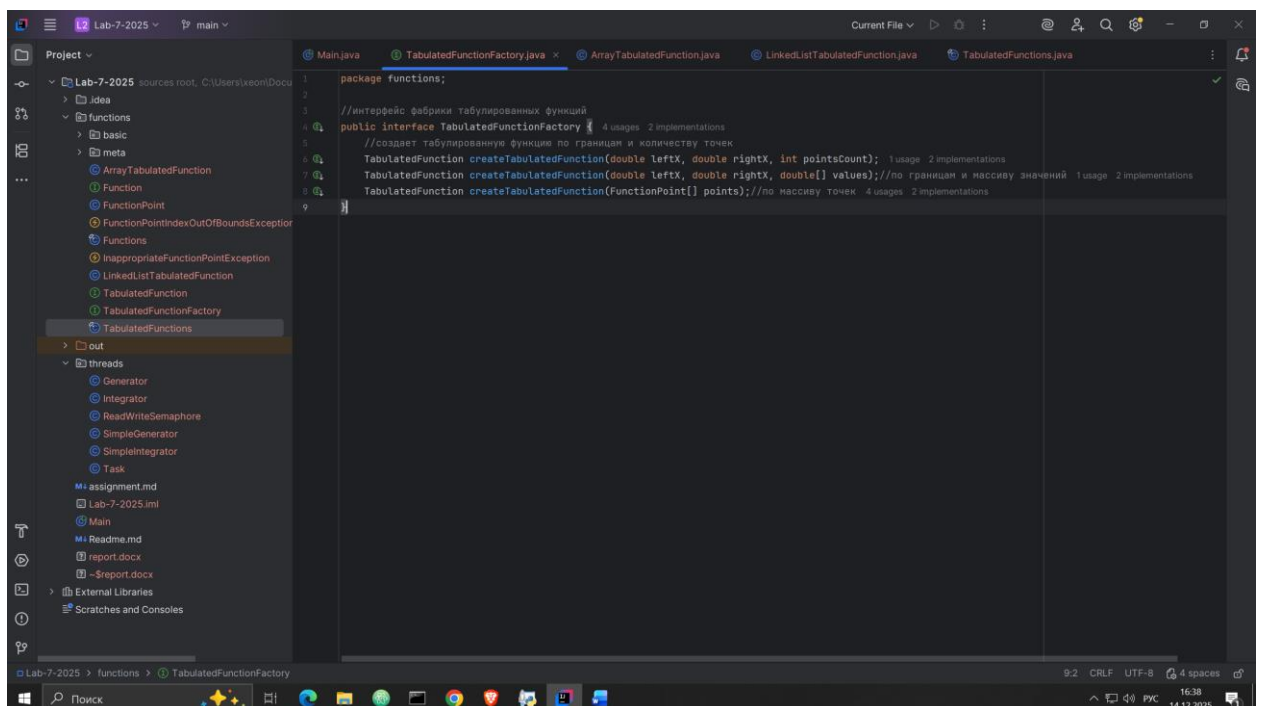


Тестирование

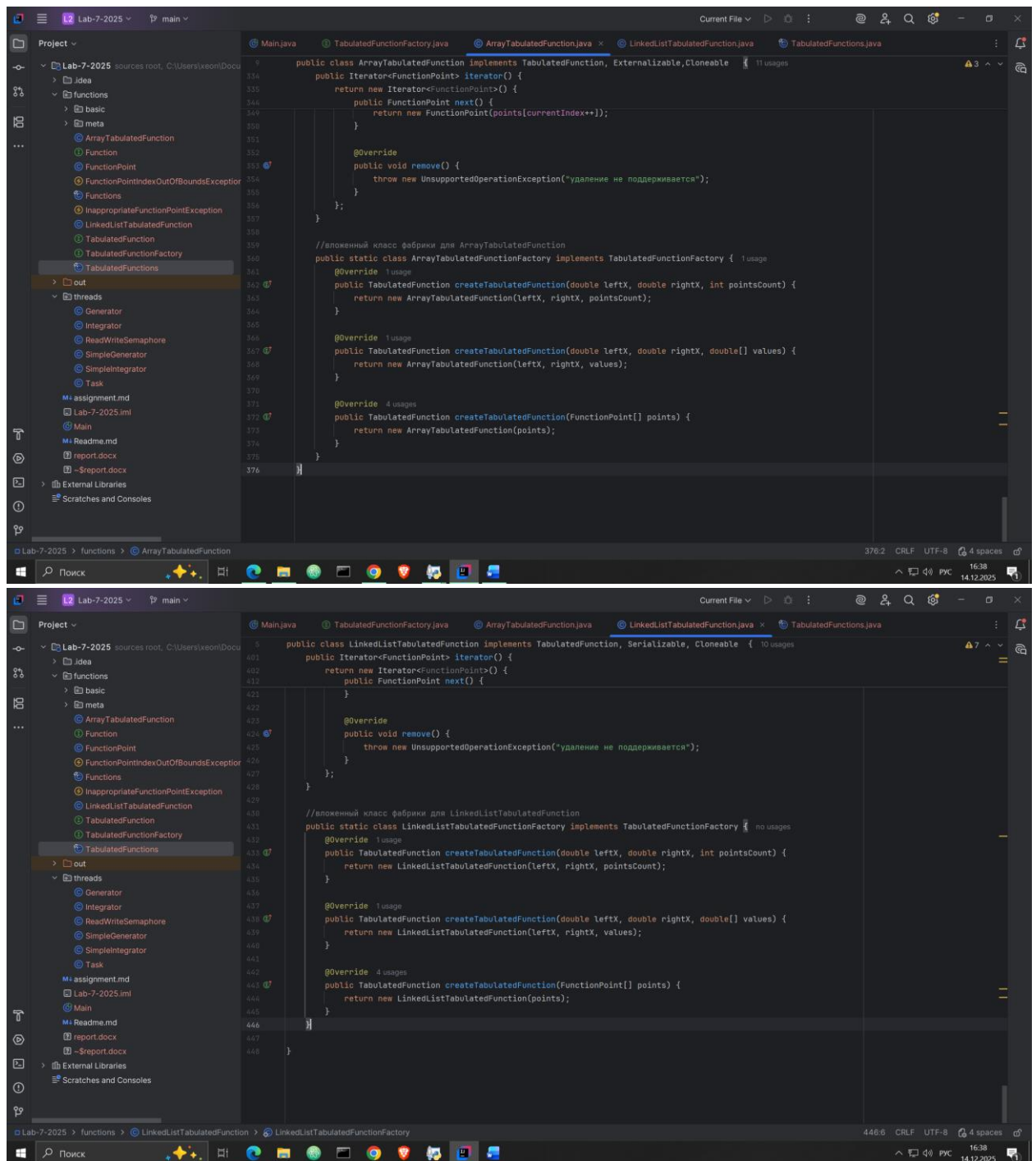


Задание 2

Создал интерфейс TabulatedFunctionFactory в пакете functions, который содержит три перегруженных метода createTabulatedFunction() где параметры методов соответствуют конструкторам табулированных функций

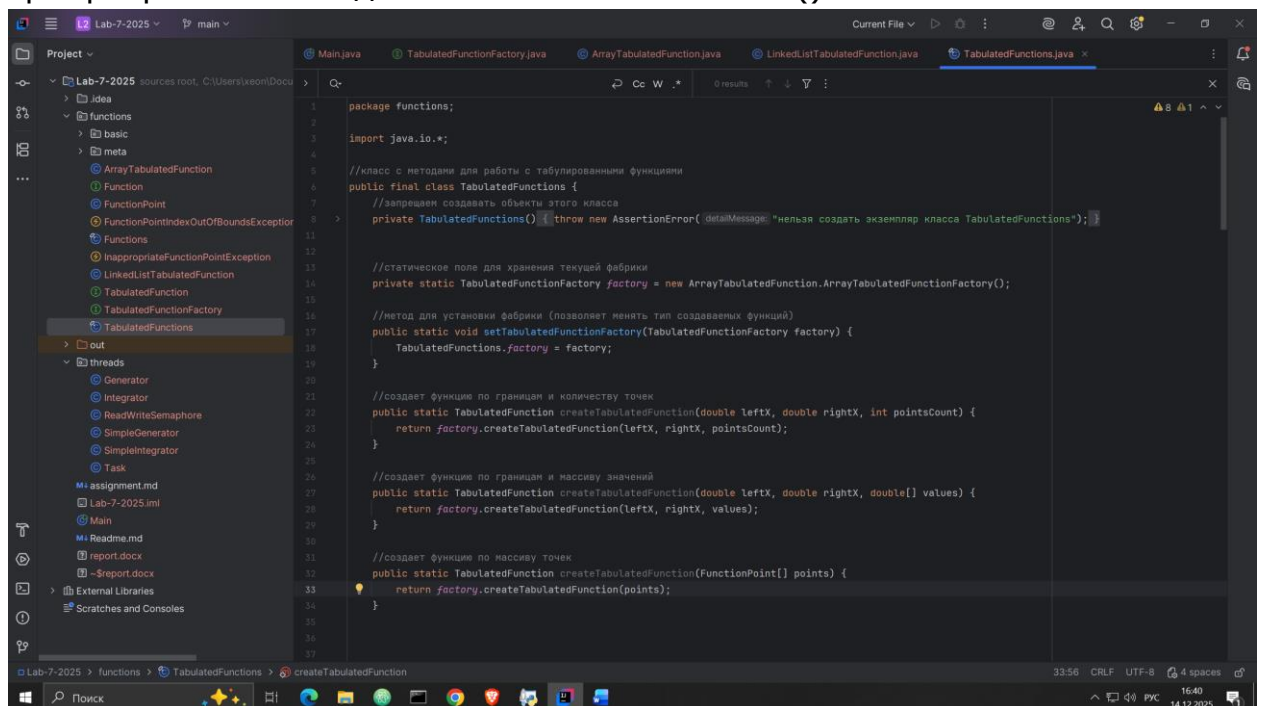


Создал два класса которые реализуют интерфейс TabulatedFunctionFactory



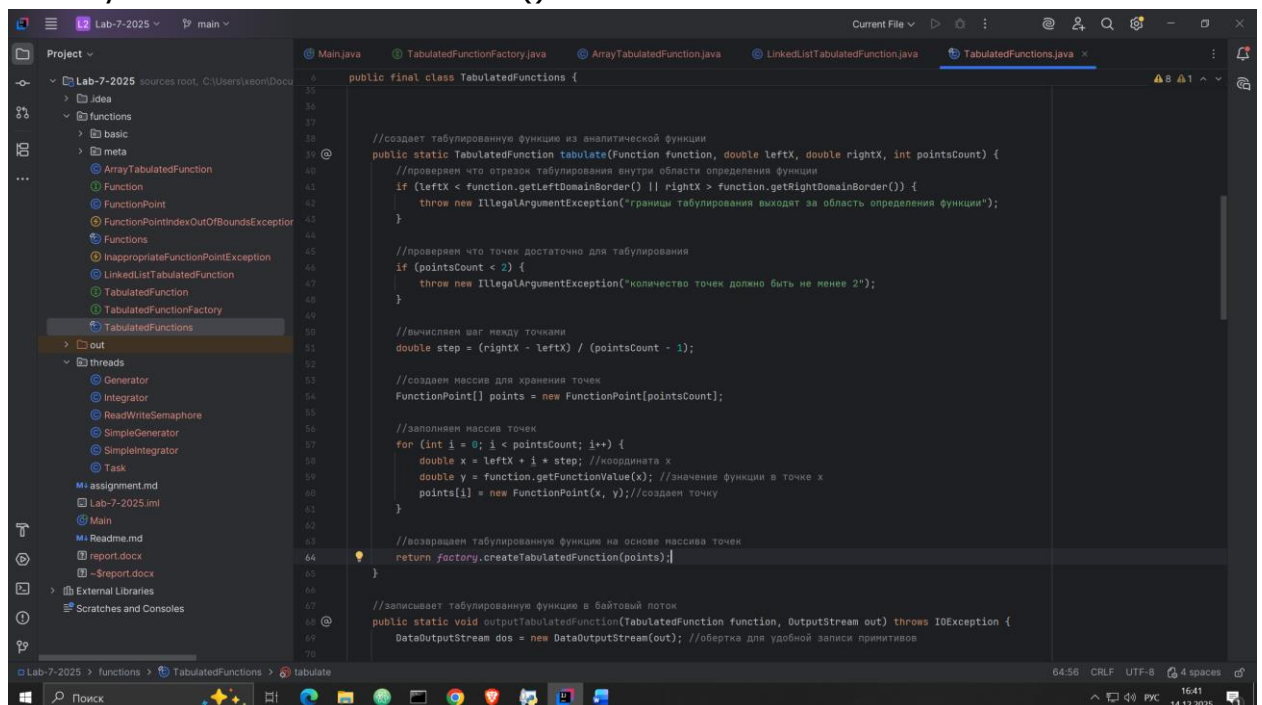
Добавил
поле factory для хранения текущей фабрики создания функций

метод `setTabulatedFunctionFactory()` для смены фабрики
три фабричных метода `createTabulatedFunction()`



```
1 package functions;
2
3 import java.io.*;
4
5 //класс с методами для работы с табулированными функциями
6 public final class TabulatedFunctions {
7     //запрещаем создавать объекты этого класса
8     private TabulatedFunctions() { throw new AssertionError("нельзя создать экземпляр класса TabulatedFunctions"); }
9
10
11
12
13
14 //статическое поле для хранения текущей фабрики
15 private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();
16
17 //метод для установки фабрики (позволяет менять тип создаваемых функций)
18 public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {
19     TabulatedFunctions.factory = factory;
20 }
21
22 //создает функцию по границам и количеству точек
23 public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
24     return factory.createTabulatedFunction(leftX, rightX, pointsCount);
25 }
26
27 //создает функцию по границам и массиву значений
28 public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
29     return factory.createTabulatedFunction(leftX, rightX, values);
30 }
31
32 //создает функцию по массиву точек
33 public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
34     return factory.createTabulatedFunction(points);
35 }
36
37 }
```

Заменил все `new ArrayTabulatedFunction()` в методах `tabulate()`,
`inputTabulatedFunction()` и `readTabulatedFunction()` на вызовы
`factory.createTabulatedFunction()`



```
36 public final class TabulatedFunctions {
37
38
39
40 //создает табулированную функцию из аналитической функции
41 public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
42     //проверим что отрезок табулирования внутри области определения функции
43     if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
44         throw new IllegalArgumentException("границы табулирования выходят за область определения функции");
45     }
46
47     //проверим что точек достаточно для табулирования
48     if (pointsCount < 2) {
49         throw new IllegalArgumentException("количество точек должно быть не менее 2");
50     }
51
52     //вычисляем шаг между точками
53     double step = (rightX - leftX) / (pointsCount - 1);
54
55     //создаем массив для хранения точек
56     FunctionPoint[] points = new FunctionPoint[pointsCount];
57
58     //заполняем массив точек
59     for (int i = 0; i < pointsCount; i++) {
60         double x = leftX + i * step; //координата x
61         double y = function.getFunctionValue(x); //значение функции в точке x
62         points[i] = new FunctionPoint(x, y); //создаем точку
63     }
64
65     //возвращаем табулированную функцию на основе массива точек
66     return factory.createTabulatedFunction(points);
67 }
68
69 //записывает табулированную функцию в байтовый поток
70 public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
71     DataOutputStream dos = new DataOutputStream(out); //обертка для удобной записи примитивов
72 }
```

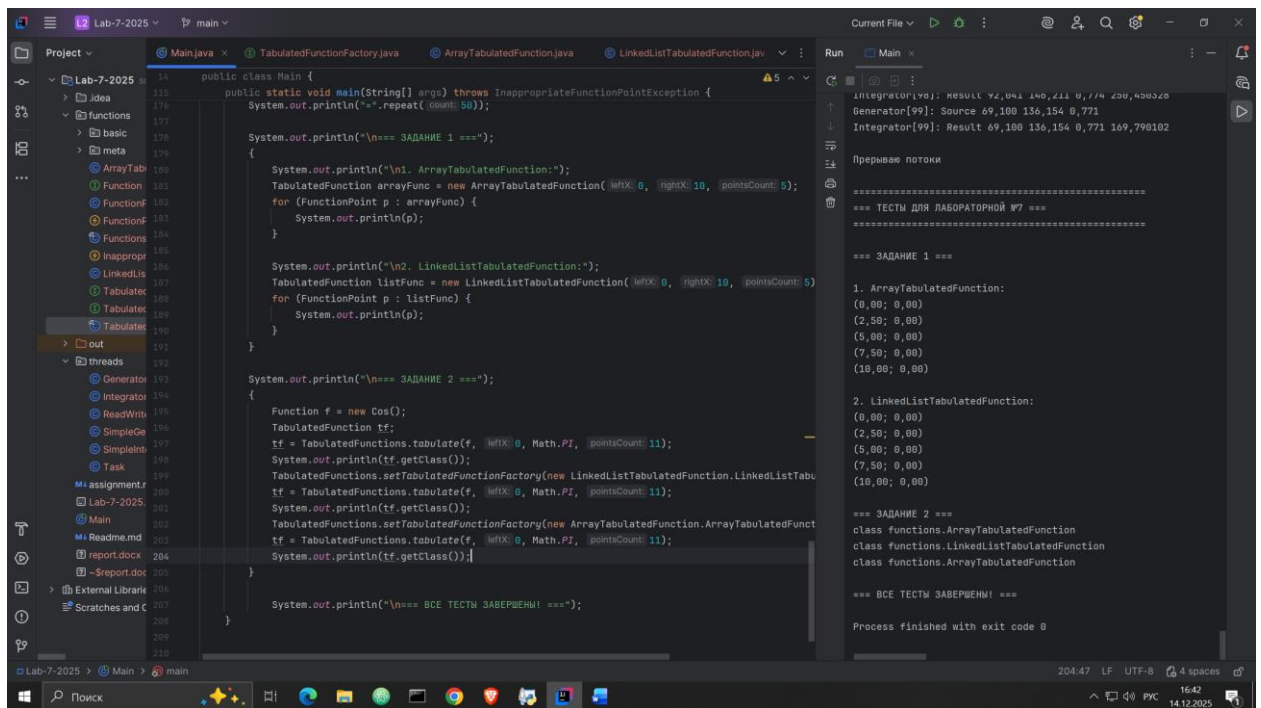


```
9 public final class TabulatedFunctions {
10     public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
11     }
12
13     //проталкиваем данные в поток
14     //не закрываем поток - это ответственность вызывающего кода
15
16     //читает табулированную функцию из байтового потока
17     public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
18         DataInputStream dis = new DataInputStream(in); //обертка для удобного чтения примитивов
19
20         //читаем количество точек
21         int pointsCount = dis.readInt();
22         FunctionPoint[] points = new FunctionPoint[pointsCount];
23
24         //читаем все точки
25         for (int i = 0; i < pointsCount; i++) {
26             double x = dis.readDouble();
27             double y = dis.readDouble();
28             points[i] = new FunctionPoint(x, y);
29         }
30
31         //создаем и возвращаем табулированную функцию
32         return factory.createTabulatedFunction(points);
33     }
34
35     //записывает табулированную функцию в символичный поток
36     public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
37         PrintWriter writer = new PrintWriter(out);
38
39         writer.print(function.getPointsCount());
40
41         for (int i = 0; i < function.getPointsCount(); i++) {
42             writer.print(" " + function.getPointX(i));
43             writer.print(" " + function.getPointY(i));
44         }
45     }
46 }
```

100:56 CRLF UTF-8 4 spaces

```
9 public final class TabulatedFunctions {
10     public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
11         tokenizer.whitespaceChars(new char[] {'\t', '\n'}); //табуляция
12         tokenizer.whitespaceChars(new char[] {'\n', '\r'}); //новая строка
13         tokenizer.whitespaceChars(new char[] {'\r', '\n'}); //возврат каретки
14
15         //читаем количество точек
16         if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
17             throw new IOException("ожидалось число (количество точек)");
18         }
19         int pointsCount = Integer.parseInt(tokenizer.sval);
20         FunctionPoint[] points = new FunctionPoint[pointsCount];
21
22         //читаем все точки (пары x y)
23         for (int i = 0; i < pointsCount; i++) {
24             //читаем x
25             if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
26                 throw new IOException("ожидалось число для x точки " + i);
27             }
28             //Заменяем запятую на точку для корректного парсинга
29             String xStr = tokenizer.sval.replace(',', '.');
30             double x = Double.parseDouble(xStr);
31
32             //читаем y
33             if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
34                 throw new IOException("ожидалось число для y точки " + i);
35             }
36             //Заменяем запятую на точку для корректного парсинга
37             String yStr = tokenizer.sval.replace(',', '.');
38             double y = Double.parseDouble(yStr);
39
40             points[i] = new FunctionPoint(x, y);
41         }
42         return factory.createTabulatedFunction(points);
43     }
44 }
```

163:56 CRLF UTF-8 4 spaces



```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("== ЗАДАНИЕ 1 ==");
        {
            System.out.println("\n1. ArrayTabulatedFunction:");
            TabulatedFunction arrayFunc = new ArrayTabulatedFunction(0, 10, 5);
            for (FunctionPoint p : arrayFunc) {
                System.out.println(p);
            }

            System.out.println("\n2. LinkedListTabulatedFunction:");
            TabulatedFunction listFunc = new LinkedListTabulatedFunction(0, 10, 5);
            for (FunctionPoint p : listFunc) {
                System.out.println(p);
            }
        }

        System.out.println("== ЗАДАНИЕ 2 ==");
        {
            Function f = new Cos();
            TabulatedFunction tf;
            tf = TabulatedFunctions.tabulate(f, 0, Math.PI, 11);
            System.out.println(tf.getClass());
            TabulatedFunctions.setTabulatedFunctionFactory(new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
            tf = TabulatedFunctions.tabulate(f, 0, Math.PI, 11);
            System.out.println(tf.getClass());
            TabulatedFunctions.setTabulatedFunctionFactory(new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
            tf = TabulatedFunctions.tabulate(f, 0, Math.PI, 11);
            System.out.println(tf.getClass());
        }

        System.out.println("== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ==");
    }
}
```

```
Integrator[70]: Result 72,041 140,211 0,174 420,420240
Generator[99]: Source 69,100 136,154 0,771
Integrator[99]: Result 69,100 136,154 0,771 169,790102

Прерываем потоки

=====
=== ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ №7 ===
=====

=== ЗАДАНИЕ 1 ===

1. ArrayTabulatedFunction:
(0,00; 0,00)
(2,50; 0,00)
(5,00; 0,00)
(7,50; 0,00)
(10,00; 0,00)

2. LinkedListTabulatedFunction:
(0,00; 0,00)
(2,50; 0,00)
(5,00; 0,00)
(7,50; 0,00)
(10,00; 0,00)

=== ЗАДАНИЕ 2 ===
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction

=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ===

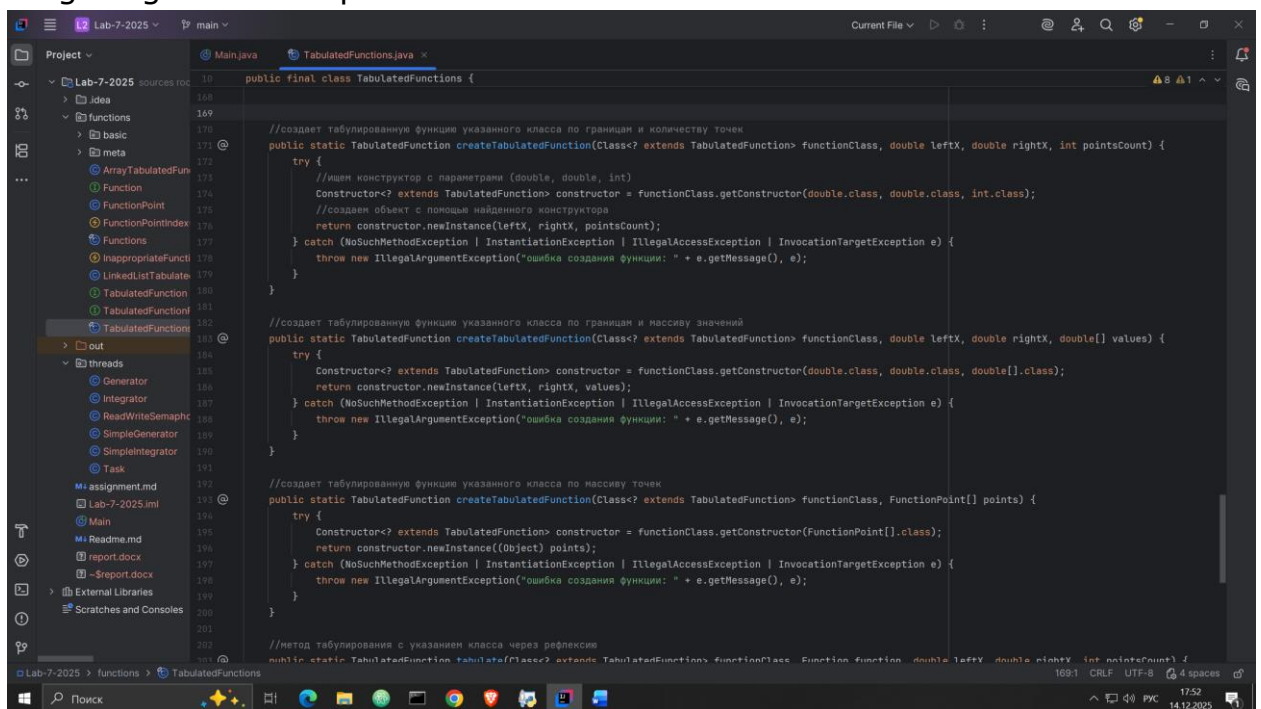
Process finished with exit code 0
```

Задание 3

Добавил три перегруженных метода createTabulatedFunction() для рефлексивного создания

Добавил метод tabulate() с параметром Class для рефлексивного табулирования

Добавил обработку исключений рефлексии с преобразованием в IllegalArgumentException

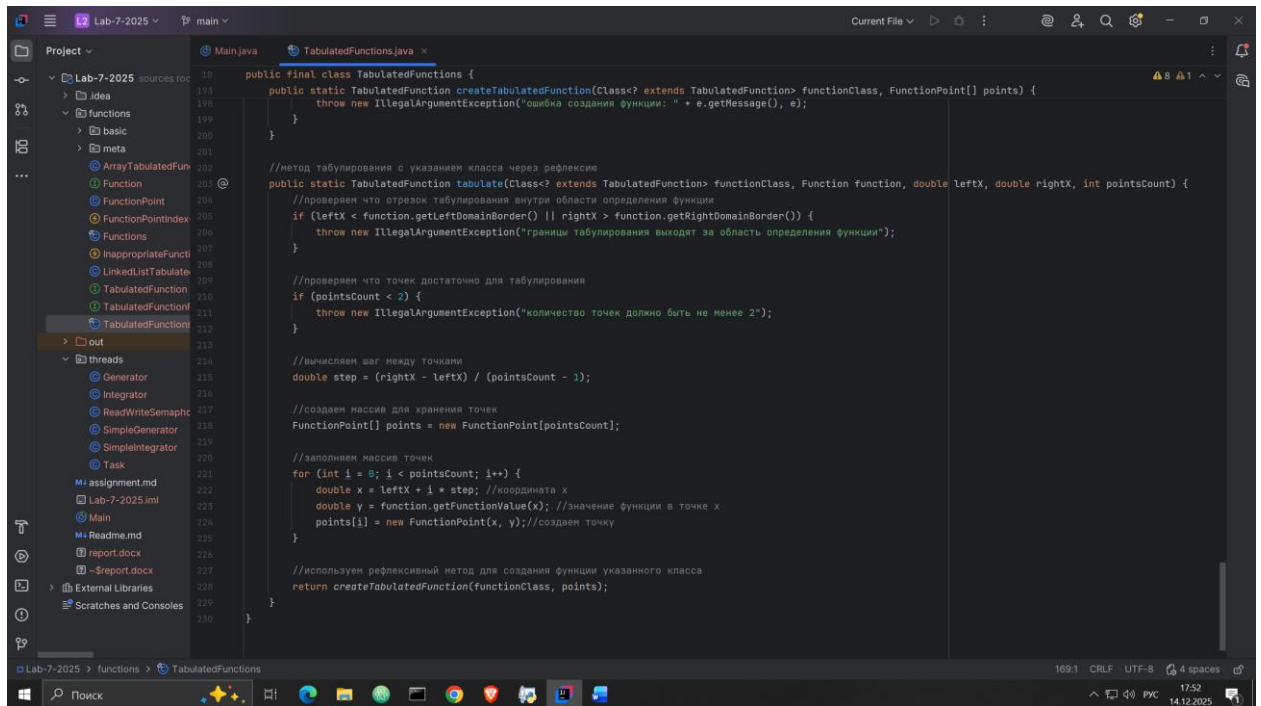


```
public final class TabulatedFunctions {
    //создает табулированную функцию указанного класса по границам и количеству точек
    public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> functionClass, double leftX, double rightX, int pointsCount) {
        try {
            //ищем конструктор с параметрами (double, double, int)
            Constructor<? extends TabulatedFunction> constructor = functionClass.getConstructor(double.class, double.class, int.class);
            //создаем объект с помощью найденного конструктора
            return constructor.newInstance(leftX, rightX, pointsCount);
        } catch (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {
            throw new IllegalArgumentException("ошибка создания функции: " + e.getMessage(), e);
        }
    }

    //создает табулированную функцию указанного класса по границам и массиву значений
    public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> functionClass, double leftX, double rightX, double[] values) {
        try {
            Constructor<? extends TabulatedFunction> constructor = functionClass.getConstructor(double.class, double.class, double[].class);
            return constructor.newInstance(leftX, rightX, values);
        } catch (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {
            throw new IllegalArgumentException("ошибка создания функции: " + e.getMessage(), e);
        }
    }

    //создает табулированную функцию указанного класса по массиву точек
    public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> functionClass, FunctionPoint[] points) {
        try {
            Constructor<? extends TabulatedFunction> constructor = functionClass.getConstructor(FunctionPoint[].class);
            return constructor.newInstance((Object) points);
        } catch (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {
            throw new IllegalArgumentException("ошибка создания функции: " + e.getMessage(), e);
        }
    }

    //метод табулирования с указанием класса через рефлексии
    public static TabulatedFunction tabulate(Class<? extends TabulatedFunction> functionClass, Function function, double leftX, double rightX, int pointsCount) {
        //...
    }
}
```

Тестирование

