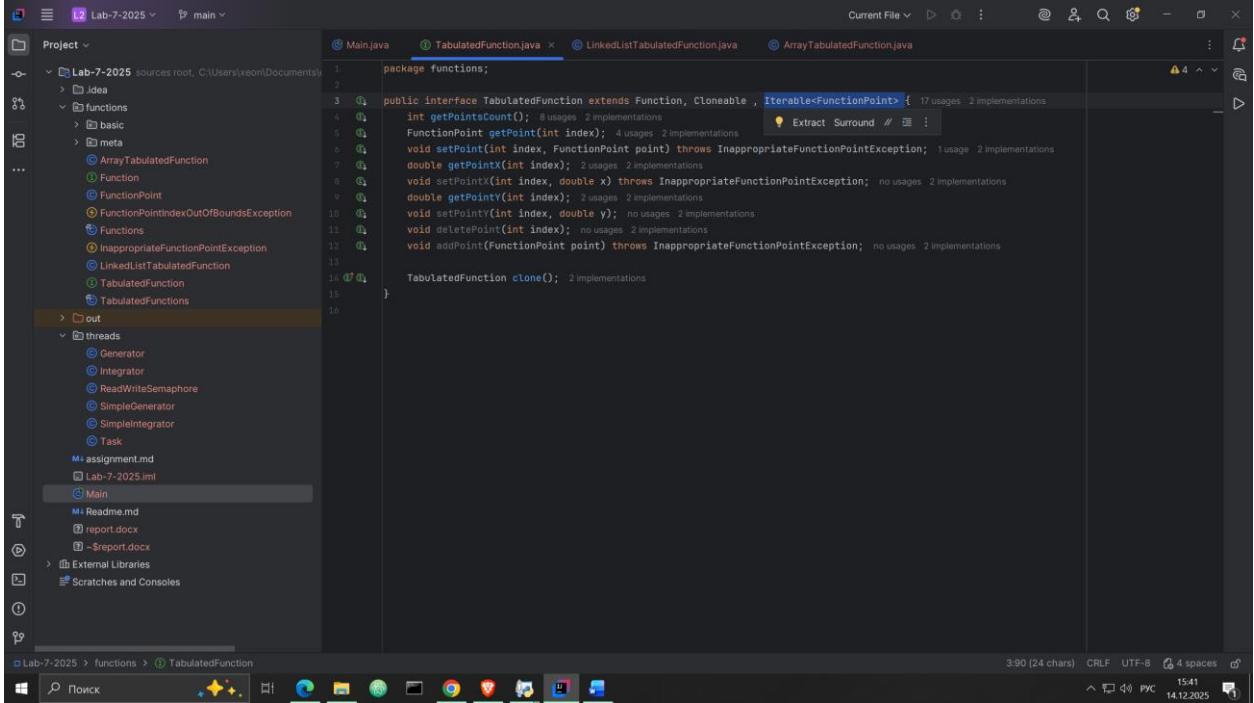


Отчёт по лабораторной работе №7
Тенигин Валерий 6204-010302D

Задание 1

Добавил в интерфейс TabulatedFunction наследование от Iterable<FunctionPoint>



The screenshot shows the IntelliJ IDEA interface. The code editor displays the `TabulatedFunction.java` file, which extends `Function` and `Cloneable`, and implements `Iterable<FunctionPoint>`. The `Iterable` interface is highlighted with a yellow background. The code includes methods for getting and setting points, cloning, and deleting points. The project structure on the left shows packages like `functions`, `basic`, `meta`, and `threads`, along with various Java files and documentation files. The bottom status bar shows file statistics and encoding information.

```
package functions;

public interface TabulatedFunction extends Function, Cloneable, Iterable<FunctionPoint> { ... }
```

Добавил методы:

`iterator()` с анонимным классом

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

`remove()` выбрасывает `UnsupportedOperationException`

```
public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable {
    public TabulatedFunction clone() {
        return cloned;
    }
    catch (CloneNotSupportedException e) {
        throw new AssertionException(message: "копирование не поддерживается", e);
    }
}

@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private int currentIndex = 0; 2 usages

        @Override
        public boolean hasNext() {
            return currentIndex < pointsCount;
        }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new java.util.NoSuchElementException("нет следующего элемента");
            }
            //возвращаем копии точки для сохранения инкапсуляции
            return new FunctionPoint(points[currentIndex++]);
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("удаление не поддерживается");
        }
    };
}
```

Добавил метод iterator() с анонимным классом

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

remove() выбрасывает UnsupportedOperationException

```
public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable, Cloneable {
    public TabulatedFunction clone() {
        catch (CloneNotSupportedException e) {
            throw new AssertionException(message: "копирование не поддерживается", e);
        }
    }

    @Override
    public Iterator<FunctionPoint> iterator() {
        return new Iterator<FunctionPoint>() {
            private FunctionNode currentNode = head.next; 4 usages
            private int currentIndex = 0; 1 usage

            @Override
            public boolean hasNext() {
                return currentNode != head;
            }

            @Override
            public FunctionPoint next() {
                if (!hasNext()) {
                    throw new java.util.NoSuchElementException("нет следующего элемента");
                }
                //возвращаем копию точки для сохранения инкапсуляции
                FunctionPoint point = new FunctionPoint(currentNode.point);
                currentNode = currentNode.next;
                currentIndex++;
                return point;
            }

            @Override
            public void remove() {
                throw new UnsupportedOperationException("удаление не поддерживается");
            }
        };
    }
}
```

Тестирование

```

public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("\n*** ЗАДАНИЕ 3 ***");
        simpleThreads();
    }

    //Задание 4
    {
        System.out.println("\n*** ЗАДАНИЕ 4 ***");
        complicatedThreads();
    }

    System.out.println("\n" + "-".repeat(count: 50));
    System.out.println("== ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ №7 ==");
    System.out.println("-".repeat(count: 50));

    System.out.println("\n*** ЗАДАНИЕ 1 ***");
    {
        System.out.println("\n1. ArrayTabulatedFunction:");
        TabulatedFunction arrayFunc = new ArrayTabulatedFunction(leftX: 0, rightX: 10, pointsCount: 5);
        for (FunctionPoint p : arrayFunc) {
            System.out.println(p);
        }
    }

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

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

```

=====
== 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)

*** ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ***

Process finished with exit code 0

Задание 2

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

```

package functions;

//интерфейс фабрики табулированных функций
public interface TabulatedFunctionFactory {
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount); //создает табулированную функцию по границам и количеству точек
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values); //по границам и массиву значений
    TabulatedFunction createTabulatedFunction(FunctionPoint[] points); //по массиву точек
}

```

Создал два класса которые реализуют интерфейс TabulatedFunctionFactory

```

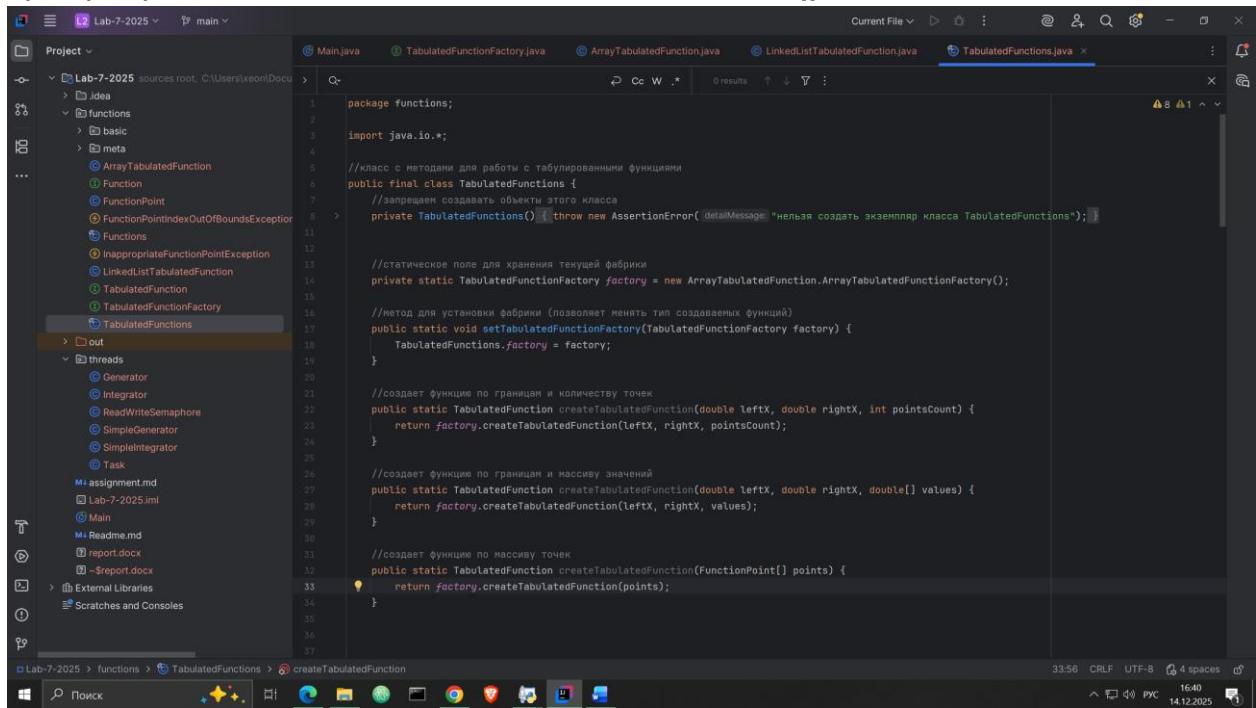
public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable {
    public Iterator<FunctionPoint> iterator() {
        return new Iterator<FunctionPoint>() {
            public FunctionPoint next() {
                return new FunctionPoint(points[currentIndex++]);
            }
            @Override
            public void remove() {
                throw new UnsupportedOperationException("удаление не поддерживается");
            }
        };
    }
    // вложенный класс фабрики для ArrayTabulatedFunction
    public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
        @Override
        public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
            return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
        }
        @Override
        public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
            return new ArrayTabulatedFunction(leftX, rightX, values);
        }
        @Override
        public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
            return new ArrayTabulatedFunction(points);
        }
    }
}

public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable, Cloneable {
    public Iterator<FunctionPoint> iterator() {
        return new Iterator<FunctionPoint>() {
            public FunctionPoint next() {
                ...
            }
            @Override
            public void remove() {
                throw new UnsupportedOperationException("удаление не поддерживается");
            }
        };
    }
    // вложенный класс фабрики для LinkedListTabulatedFunction
    public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory {
        @Override
        public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
            return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
        }
        @Override
        public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
            return new LinkedListTabulatedFunction(leftX, rightX, values);
        }
        @Override
        public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
            return new LinkedListTabulatedFunction(points);
        }
    }
}

```

Добавил
поле factory для хранения текущей фабрики создания функций

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



```
package functions;

import java.io.*;

//класс с методами для работы с табулированными функциями
public final class TabulatedFunctions {
    //запрещен создавать объекты этого класса
    private TabulatedFunctions() { throw new AssertionError("нельзя создать экземпляр класса TabulatedFunctions"); }

    //статическое поле для хранения текущей фабрики
    private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

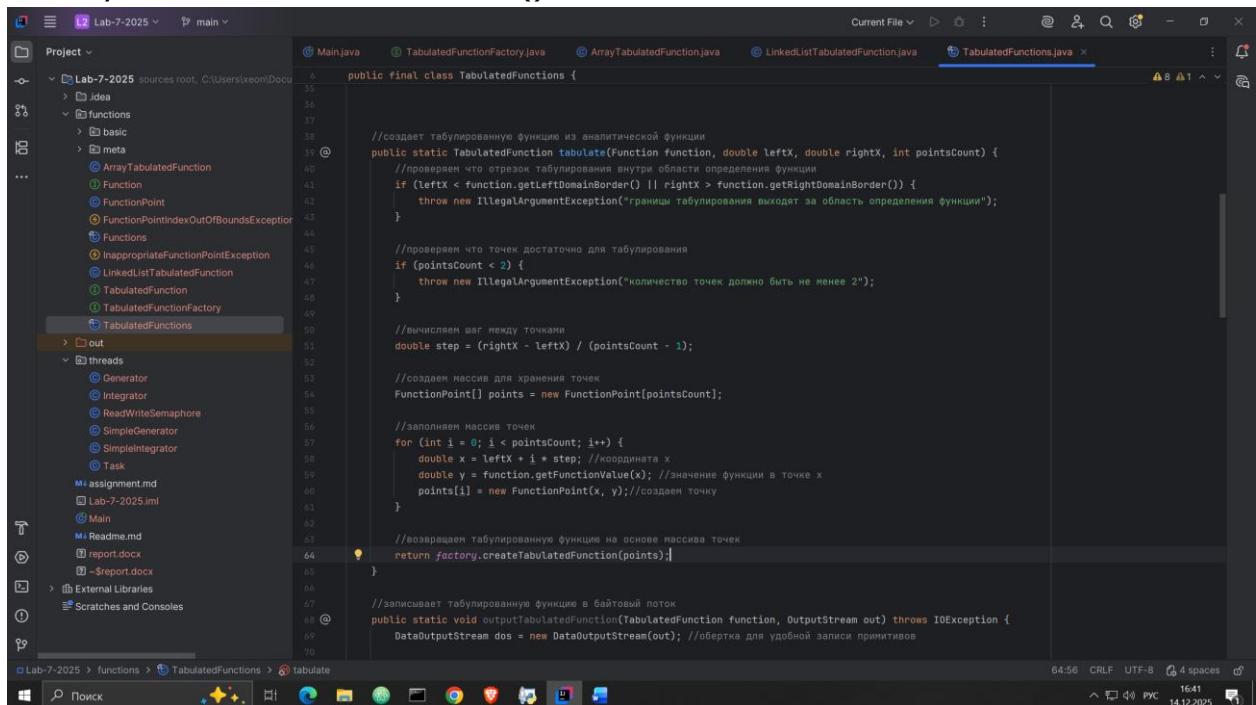
    //метод для установки фабрики (позволяет менять тип создаваемых функций)
    public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {
        TabulatedFunctions.factory = factory;
    }

    //создает функцию по границам и количеству точек
    public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return factory.createTabulatedFunction(leftX, rightX, pointsCount);
    }

    //создает функцию по массиву значений
    public static TabulatedFunction createTabulatedFunction(double[] values) {
        return factory.createTabulatedFunction(values);
    }

    //создает функцию по массиву точек
    public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return factory.createTabulatedFunction(points);
    }
}
```

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



```
public final class TabulatedFunctions {

    //создает табулированную функцию из аналитической функции
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        //проверяем что отрезок табулирования внутри области определения функции
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("границы табулирования выходят за область определения функции");
        }

        //проверяем 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
        return factory.createTabulatedFunction(points);
    }

    //записывает табулированную функцию в байтовый поток
    public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
        DataOutputStream dos = new DataOutputStream(out); //обертка для удобной записи примитивов
    }
}
```

The screenshot shows the Java code for the `TabulatedFunctions` class. The code reads a tabulated function from a stream and creates a `TabulatedFunction` object. It handles reading points and creating a `FunctionPoint` array.

```
public final class TabulatedFunctions {
    public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
        dos.flush(); //проталкиваем данные в поток
        //не закрываем поток - это ответственность вызывающего кода
    }

    //читает табулированную функцию из байтового потока
    public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {
        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
        return factory.createTabulatedFunction(points);
    }

    //записывает табулированную функцию в символьный поток
    public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
        PrintWriter writer = new PrintWriter(out);

        writer.print(function.getPointsCount());

        for (int i = 0; i < function.getPointsCount(); i++) {
            writer.print(" " + function.getPointx(i));
            writer.print(" " + function.getPointy(i));
        }
    }
}
```

The screenshot shows the implementation of the `readTabulatedFunction` method. It uses a StreamTokenizer to read points from a reader, handling whitespace and tokens.

```
public final class TabulatedFunctions {
    public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {
        tokenizer.whitespaceChars('\n', '\n'); // табуляция
        tokenizer.whitespaceChars('\n', '\n'); // новая строка
        tokenizer.whitespaceChars('\n', '\n'); // возврат каретки

        // читаем количество точек
        if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
            throw new IOException("ожидалось число (количество точек)");
        }
        int pointsCount = Integer.parseInt(tokenizer.sval);

        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_WORD) {
                throw new IOException("ожидалось число для x точки " + i);
            }
            // Заменяем запятую на точку для корректного парсинга
            String xStr = tokenizer.sval.replace(',', '.');
            double x = Double.parseDouble(xStr);

            // читаем y
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("ожидалось число для y точки " + i);
            }
            // Заменяем запятую на точку для корректного парсинга
            String yStr = tokenizer.sval.replace(',', '.');
            double y = Double.parseDouble(yStr);

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

        return factory.createTabulatedFunction(points);
    }
}
```

```

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

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

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

    System.out.println("\nВСЕ ТЕСТЫ ЗАВЕРШЕНЫ!");
}

```

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 {
            //ищет 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(points);
        } catch (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {
            throw new IllegalArgumentException("ошибка создания функции: " + e.getMessage(), e);
        }
    }

    //метод табулирования с указанием класса через рефлексию
    public static TabulatedFunction tabulate(Class<? extends TabulatedFunction> functionClass, Function function, double leftY, double rightY, int pointCount) {
}

```

Screenshot of the IntelliJ IDEA IDE showing the code editor for `TabulatedFunctions.java`. The code implements a static method `createTabulatedFunction` that takes a class `Function` and generates a tabulated function object. It handles various function types like `ArrayTabulatedFunction`, `LinkedListTabulatedFunction`, and `TabulatedFunction`.

```

public final class TabulatedFunctions {
    public static TabulatedFunction createTabulatedFunction(Class<? extends TabulatedFunction> functionClass, FunctionPoint[] points) {
        if (functionClass == null || points == null) {
            throw new IllegalArgumentException("ошибка создания функции: " + e.getMessage(), e);
        }
    }

    // метод табулирования с указанием класса через рефлексию
    public static TabulatedFunction tabulate(Class<? extends TabulatedFunction> functionClass, Function function, double leftX, double rightX, int pointsCount) {
        // проверяем что отрезок табулирования внутри области определения функции
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("границы табулирования выходят за область определения функции");
        }

        // проверяем 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
        }

        // используем рефлексивный метод для создания функции указанного класса
        return createTabulatedFunction(functionClass, points);
    }
}

```

Тестираование

Screenshot of the IntelliJ IDEA IDE showing the code editor for `Main.java`. The `main` method demonstrates the usage of the `TabulatedFunctions` class to create different types of tabulated functions and print their points.

```

public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println(tf.getClass());
    }

    System.out.println("\n*** ЗАДАНИЕ 1 ***");
    {
        TabulatedFunction f;

        f = TabulatedFunctions.createTabulatedFunction(
                ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 5);
        System.out.println(f.getClass());
        System.out.println(f);

        f = TabulatedFunctions.createTabulatedFunction(
                LinkedListTabulatedFunction.class,
                new FunctionPoint[] {
                    new FunctionPoint(0, 0),
                    new FunctionPoint(10, 0)
                });
        System.out.println(f.getClass());
        System.out.println(f);

        f = TabulatedFunctions.createTabulatedFunction(
                TabulatedFunction.class,
                new FunctionPoint[] {
                    new FunctionPoint(0, 0),
                    new FunctionPoint(10, 0)
                });
        System.out.println(f.getClass());
        System.out.println(f);
    }

    System.out.println("\n*** ЗАДАНИЕ 2 ***");
    class functions.ArrayTabulatedFunction
    class functions.LinkedListTabulatedFunction
    class functions.TabulatedFunction

    *** ЗАДАНИЕ 3 ***
    class functions.ArrayTabulatedFunction
    {{0,0; 0,0}, {5,0; 0,0}, {10,0; 0,0}}
    class functions.ArrayTabulatedFunction
    {{0,0; 0,0}, {10,0; 10,0}}
    class functions.LinkedListTabulatedFunction
    {{0,0; 0,0}, {10,0; 10,0}}
    class functions.TabulatedFunction
    {{0,0; 0,0}, {0,31; 0,31}, {0,63; 0,59}, {0,94; 0,45}};

    *** ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ***
}

```

Дополнение

The image shows two side-by-side screenshots of a Java code editor, likely PyCharm, displaying the same file: `TabulatedFunctions.java`. The code implements a tabulated function from a byte stream or a reader.

```
public final class TabulatedFunctions { // 16 usages & valera228 *
    }

    //читает табулированную функцию указанного класса из байтового потока
    public static TabulatedFunction inputTabulatedFunction(Class<? extends TabulatedFunction> functionClass, InputStream in) throws IOException {
        DataInputStream dis = new DataInputStream(in);

        //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);
        }

        //используем рефлексивный метод для создания функции указанного класса
        return createTabulatedFunction(functionClass, points);
    }

    //читает табулированную функцию указанного класса из символьного потока
    public static TabulatedFunction readTabulatedFunction(Class<? extends TabulatedFunction> functionClass, Reader in) throws IOException {
        StreamTokenizer tokenizer = new StreamTokenizer(in);

        // Настраиваем tokenizer для правильной работы с числами с запятыми
        tokenizer.resetSyntax();
        tokenizer.wordChars( low:'0', hi:'9' ); // цифры
        tokenizer.wordChars( low:',', hi:','); // точка
        tokenizer.wordChars( low':', hi':'); // запятая
        tokenizer.wordChars( low:'-', hi:'-' ); // минус
        tokenizer.wordChars( low:'E', hi:'E' ); // экспонента
        tokenizer.wordChars( low:'e', hi:'e' ); // экспонента
        tokenizer.whitespaceChars( low:' ', hi:' ' ); // пробел
        tokenizer.whitespaceChars( low:'\t', hi:'\t' ); // табуляция
        tokenizer.whitespaceChars( low:'\r', hi:'\r' ); // возврат каретки
    }

    public static TabulatedFunction readTabulatedFunction(Class<? extends TabulatedFunction> functionClass, Reader in) throws IOException {
        tokenizer.whitespaceChars( low:'\t', hi:'\t' ); // табуляция
        tokenizer.whitespaceChars( low:'\n', hi:'\n' ); // новая строка
        tokenizer.whitespaceChars( low:'\r', hi:'\r' ); // возврат каретки

        //читаем количество точек
        if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
            throw new IOException("ожидалось число (количество точек)");
        }
        int pointsCount = Integer.parseInt(tokenizer.sval);

        FunctionPoint[] points = new FunctionPoint[pointsCount];

        //читаем все точки
        for (int i = 0; i < pointsCount; i++) {
            //читаем x
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("ожидалось число для x точки " + i);
            }
            String xStr = tokenizer.sval.replace( oldChar:',', newChar:'.' );
            double x = Double.parseDouble(xStr);

            //читаем y
            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("ожидалось число для y точки " + i);
            }
            String yStr = tokenizer.sval.replace( oldChar:',', newChar:'.' );
            double y = Double.parseDouble(yStr);

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

        //используем рефлексивный метод для создания функции указанного класса
        return createTabulatedFunction(functionClass, points);
    }
}
```

The code defines a `TabulatedFunctions` class with two static methods: `inputTabulatedFunction` and `readTabulatedFunction`. Both methods read a function from a `InputStream` or a `Reader` respectively. They first read the number of points, then read each point's coordinates (`x` and `y`) from the stream. Finally, they use reflection to create an instance of the specified function class with the read points.

The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "Lab-7-2025" and contains a package "Main.java".
- Code Editor:** The main file "Main.java" contains Java code for testing tabulated functions. It includes imports for `java.util.ArrayList`, `java.util.List`, `java.io.ByteArrayOutputStream`, `java.io.StringWriter`, `java.io.StringReader`, and `java.math.BigDecimal`.
- Code Content:**

```
public class Main {    public static void main(String[] args) throws IOException, InappropriateFunctionPointException {        f = TabulatedFunctions.tabulate(            LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 11);        System.out.println(f.getClass());        System.out.println(f);        // создаем и записываем тестовую функцию        TabulatedFunction testFunc = TabulatedFunctions.createTabulatedFunction(            ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 5);        // тест inputTabulatedFunction        ByteArrayOutputStream byteOut = new ByteArrayOutputStream();        TabulatedFunctions.outputTabulatedFunction(testFunc, byteOut);        ByteArrayInputStream byteIn = new ByteArrayInputStream(byteOut.toByteArray());        TabulatedFunction read1 = TabulatedFunctions.inputTabulatedFunction(            LinkedListTabulatedFunction.class, byteIn);        System.out.println("inputTabulatedFunction: " + read1.getSimpleClassName());        // тест readTabulatedFunction        StringWriter writer = new StringWriter();        TabulatedFunctions.writeTabulatedFunction(testFunc, writer);        StringReader reader = new StringReader(writer.toString());        TabulatedFunction read2 = TabulatedFunctions.readTabulatedFunction(            ArrayTabulatedFunction.class, reader);        System.out.println("readTabulatedFunction: " + read2.getSimpleClassName());        System.out.println("\n==== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ===");    } }
```
- Output Window:** The "Run" tab shows the output of the program:

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

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

*** ЗАДАНИЕ 2 ***
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction

*** ЗАДАНИЕ 3 ***
class functions.ArrayTabulatedFunction
{(0,0; 0,0), (5,0; 0,0), (10,0; 0,0)}
class functions.ArrayTabulatedFunction
{(0,0; 0,0), (10,0; 10,0)}
class functions.LinkedListTabulatedFunction
{(0,0; 0,0), (10,0; 10,0)}
class functions.LinkedListTabulatedFunction
{(0,0; 0,0), (0,31; 0,31), (0,63; 0,59), (0,94; 0,81), (1,26; 0,59)}
inputTabulatedFunction: LinkedListTabulatedFunction
readTabulatedFunction: ArrayTabulatedFunction

*** ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ***
```
- Bottom Bar:** Shows the terminal encoding as "UTF-8", the date/time as "16.12.2025 13:27", and other system information.