ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА №7

Студент Чячяков А.И.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка

САМАРА 2025

Задание 1

Сделал так, чтобы все объекты типа TabulatedFunction можно было использовать в качестве объекта-агрегата в for-each, извлекаемые объекты при этом должны иметь тип FunctionPoint.

В интерфейсе TabulatedFunction добавил необходимый родительский тип, использовал при этом параметризованный тип (generic type).

В классах, реализующих интерфейс TabulatedFunction, добавил требующийся метод, возвращающий объект итератора.

Классы итераторов сделал анонимными. В соответствии с паттерном «Итератор».

Метод получения следующего элемента должен выбрасывать исключение NoSuchElementException, если следующего элемента нет. Возвращаемый методом объект типа FunctionPoint не должен позволять нарушить инкапсуляцию объекта табулированной функции.

Код 1.1

```
package functions;

public interface TabulatedFunction extends Iterable<FunctionPoint>, Function, Cloneable {
    double getLeftDomainBorder();

    double getRightDomainBorder();

    double getFunctionValue(double x);

    int getPointsCount();

    FunctionPoint getPoint(int index);

    void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;

    double getPointX(int index);

    void setPointX(int index, double x) throws InappropriateFunctionPointException;

    double getPointY(int index);

    void setPointY(int index, double y);

    void deletePoint(int index);

    void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;

    @Override
    String toString();

    @Override
    boolean equals(Object o);

    @Override
    int hashCode();

    Object clone();
}
```

Код 1.2

```java
@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 NoSuchElementException("Больше элементов нет");
            }
            return new FunctionPoint(points[currentIndex++]);
        }

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

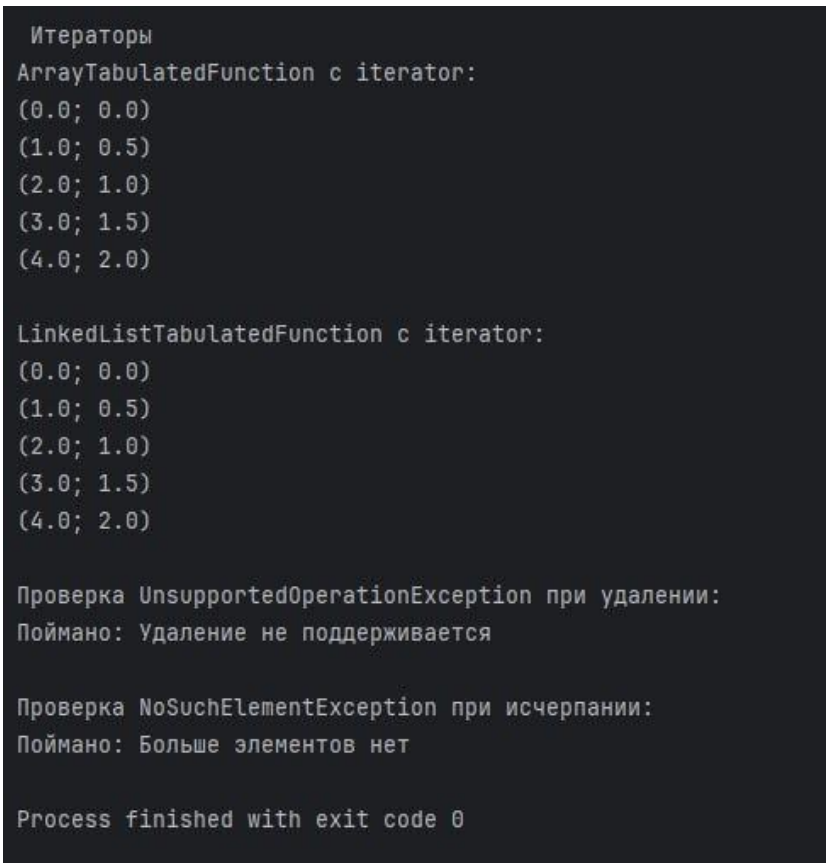
Код 1.3

```java
    public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable {  4 u
        public Object clone() {
                        node.point = new FunctionPoint(current.point);
                        current = current.next;
                    }
                    return copy;
                }
            }

            @Override
            public java.util.Iterator<FunctionPoint> iterator() {
                return new java.util.Iterator<FunctionPoint>() {
                    private FunctionNode current = head.next;  4 usages

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

                    @Override
                    public FunctionPoint next() {
                        if (!hasNext()) {
                            throw new java.util.NoSuchElementException("Больше элементов нет");
                        }
                        FunctionPoint p = new FunctionPoint(current.point);
                        current = current.next;
                        return p;
                    }

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

Код 1.4

```
public class Main {
    private static void complicatedThreads() {  1 usage
            Thread.currentThread().interrupt();
        }

        System.out.println("Оба потока завершили работу (прерваны)");
    }

    private static void testIterators() {  1 usage
        double[] values = {0, 0.5, 1.0, 1.5, 2.0};
        TabulatedFunction arrayFunc = new ArrayTabulatedFunction( leftX: 0,  rightX: 4, values);
        System.out.println("ArrayTabulatedFunction с iterator:");
        for (FunctionPoint p : arrayFunc) {
            System.out.println(p);
        }

        TabulatedFunction listFunc = new LinkedListTabulatedFunction( leftX: 0,  rightX: 4, values);
        System.out.println("\nLinkedListTabulatedFunction с iterator:");
        for (FunctionPoint p : listFunc) {
            System.out.println(p);
        }

        System.out.println("\nПроверка UnsupportedOperationException при удалении:");
        try {
            Iterator<FunctionPoint> it = arrayFunc.iterator();
            if (it.hasNext()) {
                it.next();
                it.remove();
            }
        } catch (UnsupportedOperationException e) {
            System.out.println("Поймано: " + e.getMessage());
        }

        System.out.println("\nПроверка NoSuchElementException при исчерпании:");
        try {
            Iterator<FunctionPoint> it = arrayFunc.iterator();
            while (it.hasNext()) {
                it.next();
            }
            it.next();
        } catch (NoSuchElementException e) {
            System.out.println("Поймано: " + e.getMessage());
        }
    }
```

Вывод:



Задание 2

В пакете functions описал базовый интерфейс фабрик табулированных функций TabulatedFunctionFactory. Интерфейс должен объявлять три перегруженных метода TabulatedFunction createTabulatedFunction(), параметры которых соответствуют параметрам конструкторов классов табулированных функций.

Для удобства опишисал в классах ArrayTabulatedFunction и LinkedListTabulatedFunction классы фабрик ArrayTabulatedFunctionFactory и LinkedListTabulatedFunctionFactory, реализующие интерфейс фабрики и порождающие объекты соответствующих классов табулированных функций. Сделал классы фабрик вложенными и публичными.

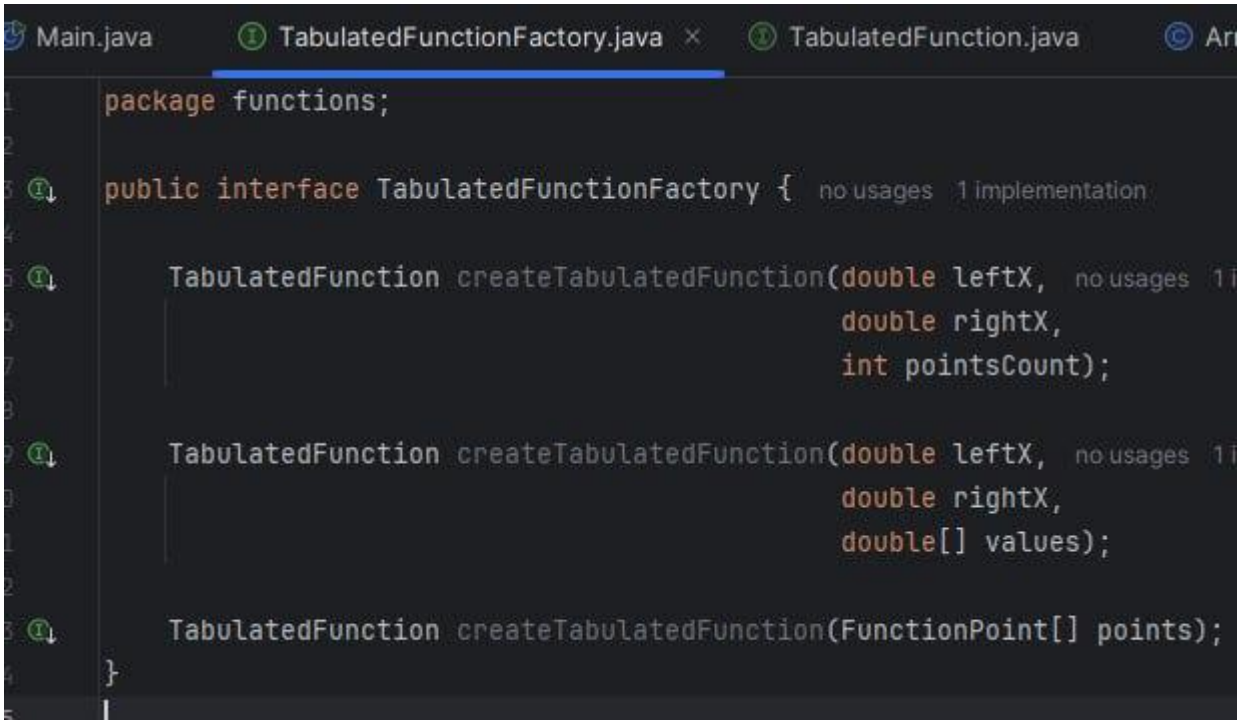
В классе TabulatedFunctions объявил приватное статическое поле типа TabulatedFunctionFactory и проинициализировал его объектом одного из описанных классов фабрик. Также объявил метод setTabulatedFunctionFactory(), позволяющий заменить объект фабрики.

Ещё в классе TabulatedFunctions описал три перегруженных метода TabulatedFunction createTabulatedFunction(), возвращающих объекты

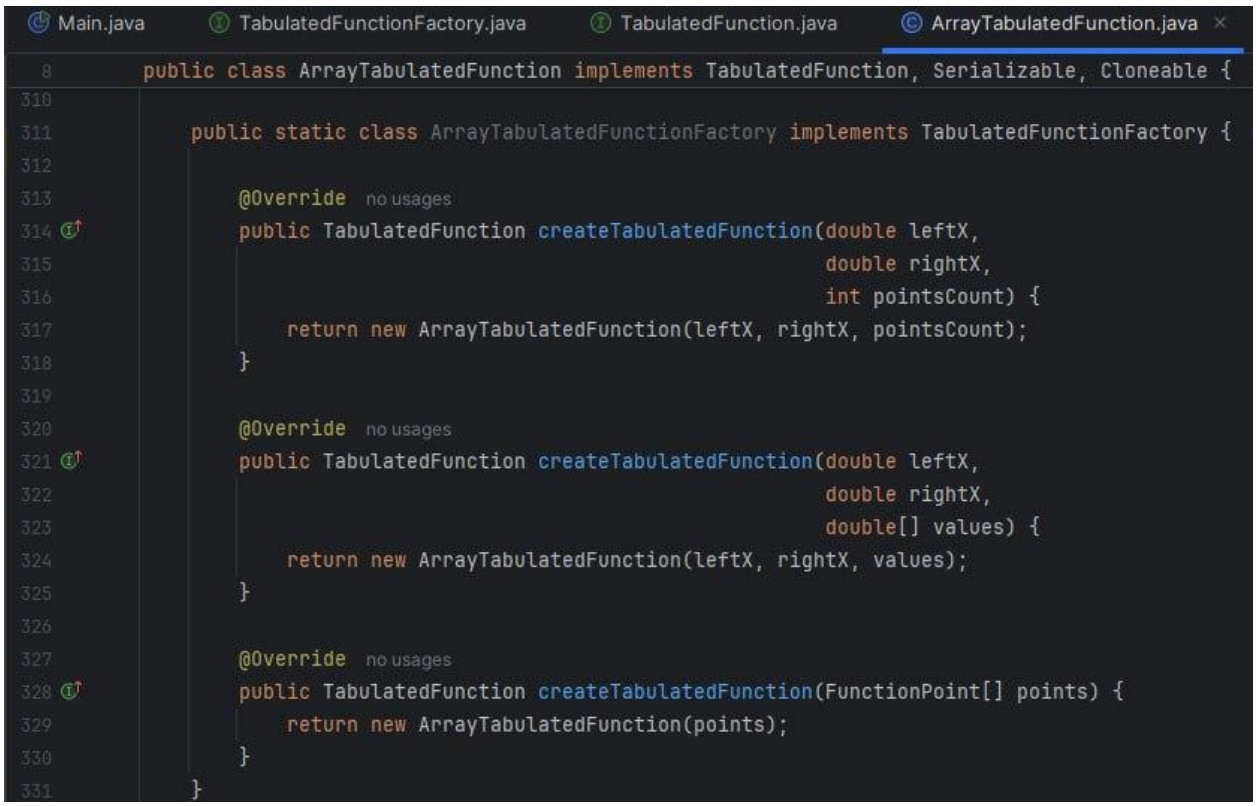
табулированных функций, созданные с помощью текущей фабрики. Параметры методов должны соответствовать параметрам методов фабрики.

В остальных методах класса, где требуется создание объектов табулированных функций, заменил явное создание объектов с помощью конструкторов на вызов соответствующего метода createTabulatedFunction().

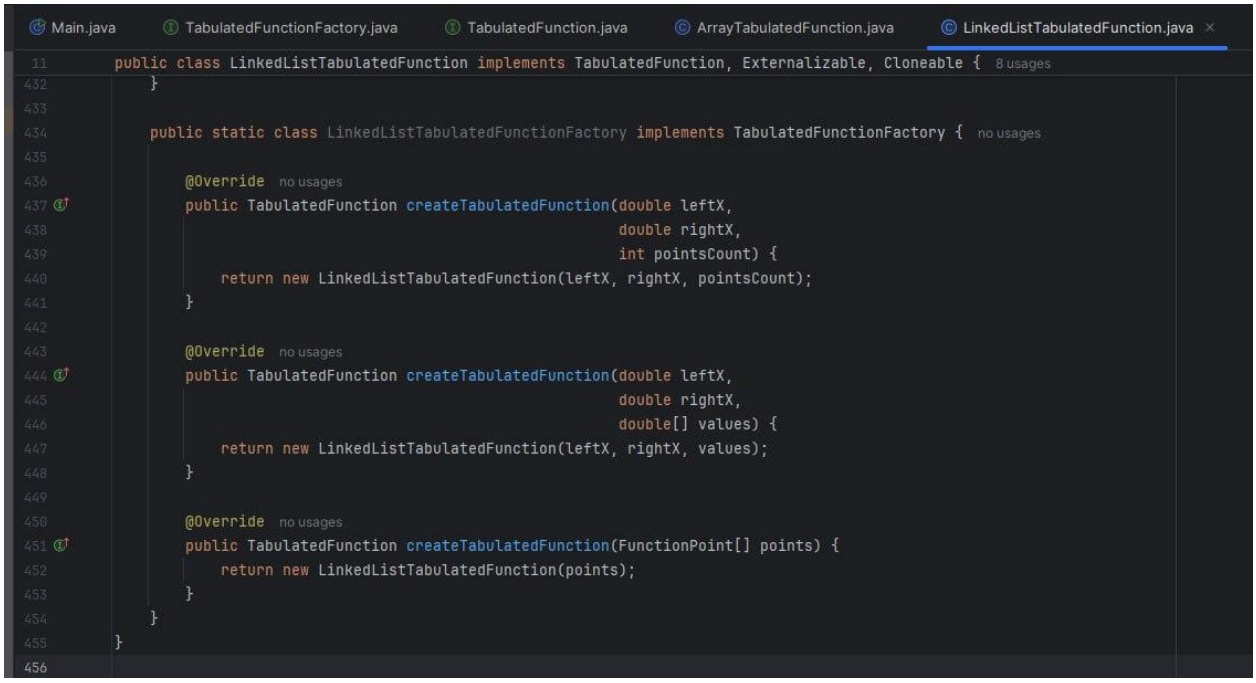
Код 2.1



Код 2.2


Main.java
TabulatedFunctionFactory.java
TabulatedFunction.java
ArrayTabulatedFunction.java
public class ArrayTabulatedFunction implements TabulatedFunction, Serializable, Cloneable {
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
@Override no usages
public TabulatedFunction createTabulatedFunction(double leftX,
double rightX,
int pointsCount) {
return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
}
@Override no usages
public TabulatedFunction createTabulatedFunction(double leftX,
double rightX,
double[] values) {
return new ArrayTabulatedFunction(leftX, rightX, values);
}
@Override no usages
public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
return new ArrayTabulatedFunction(points);
}
}

Код 2.3


Main.java
TabulatedFunctionFactory.java
TabulatedFunction.java
ArrayTabulatedFunction.java
LinkedListTabulatedFunction.java
public class LinkedListTabulatedFunction implements TabulatedFunction, Externalizable, Cloneable { 8 usages
}
public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory { no usages
@Override no usages
public TabulatedFunction createTabulatedFunction(double leftX,
double rightX,
int pointsCount) {
return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
}
@Override no usages
public TabulatedFunction createTabulatedFunction(double leftX,
double rightX,
double[] values) {
return new LinkedListTabulatedFunction(leftX, rightX, values);
}
@Override no usages
public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
return new LinkedListTabulatedFunction(points);
}
}
}

Код 2.4

```
public class TabulatedFunctions {  no usages
    }

    private static TabulatedFunctionFactory factory =  4 usages
            new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

    public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) {  no usages
        if (newFactory == null) {
            throw new IllegalArgumentException("Фабрика не может быть null");
        }
        factory = newFactory;
    }

    public static TabulatedFunction createTabulatedFunction(double leftX,  no usages
                                                             double rightX,
                                                             int pointsCount) {
        return factory.createTabulatedFunction(leftX, rightX, pointsCount);
    }

    public static TabulatedFunction createTabulatedFunction(double leftX,  1 usage
                                                             double rightX,
                                                             double[] values) {
        return factory.createTabulatedFunction(leftX, rightX, values);
    }

    public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {  2 usages
        return factory.createTabulatedFunction(points);
    }

    public static TabulatedFunction tabulate(Function function,  no usages
                                             double leftX,
                                             double rightX,
                                             int pointsCount) {
        if (leftX < function.getLeftDomainBorder()
                || rightX > function.getRightDomainBorder()
                || leftX >= rightX
                || pointsCount < 2) {
            throw new IllegalArgumentException("Некорректные границы табулирования или количество точек");
        }

        double[] values = new double[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        double x = leftX;
```

Код 2.5

```
public class Main {
    }

    private static void testFactories() {  1 usage
        System.out.println("\nПроверка фабрик TabulatedFunctionFactory:");
        Function f = new Cos();
        TabulatedFunction tf;

        tf = TabulatedFunctions.tabulate(f,  leftX: 0, Math.PI,  pointsCount: 11);
        System.out.println("Текущая фабрика, класс: " + tf.getClass().getName());

        TabulatedFunctions.setTabulatedFunctionFactory(
                new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
        tf = TabulatedFunctions.tabulate(f,  leftX: 0, Math.PI,  pointsCount: 11);
        System.out.println("После установки LinkedList фабрики, класс: " + tf.getClass().getName());

        TabulatedFunctions.setTabulatedFunctionFactory(
                new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
        tf = TabulatedFunctions.tabulate(f,  leftX: 0, Math.PI,  pointsCount: 11);
        System.out.println("После возврата Array фабрики, класс: " + tf.getClass().getName());
    }
```

Вывод:

```
Фабрики табулированных функций

Проверка фабрик TabulatedFunctionFactory:
Текущая фабрика, класс: functions.ArrayTabulatedFunction
После установки LinkedList фабрики, класс: functions.LinkedListTabulatedFunction
После возврата Array фабрики, класс: functions.ArrayTabulatedFunction

Process finished with exit code 0
```

Задание 3

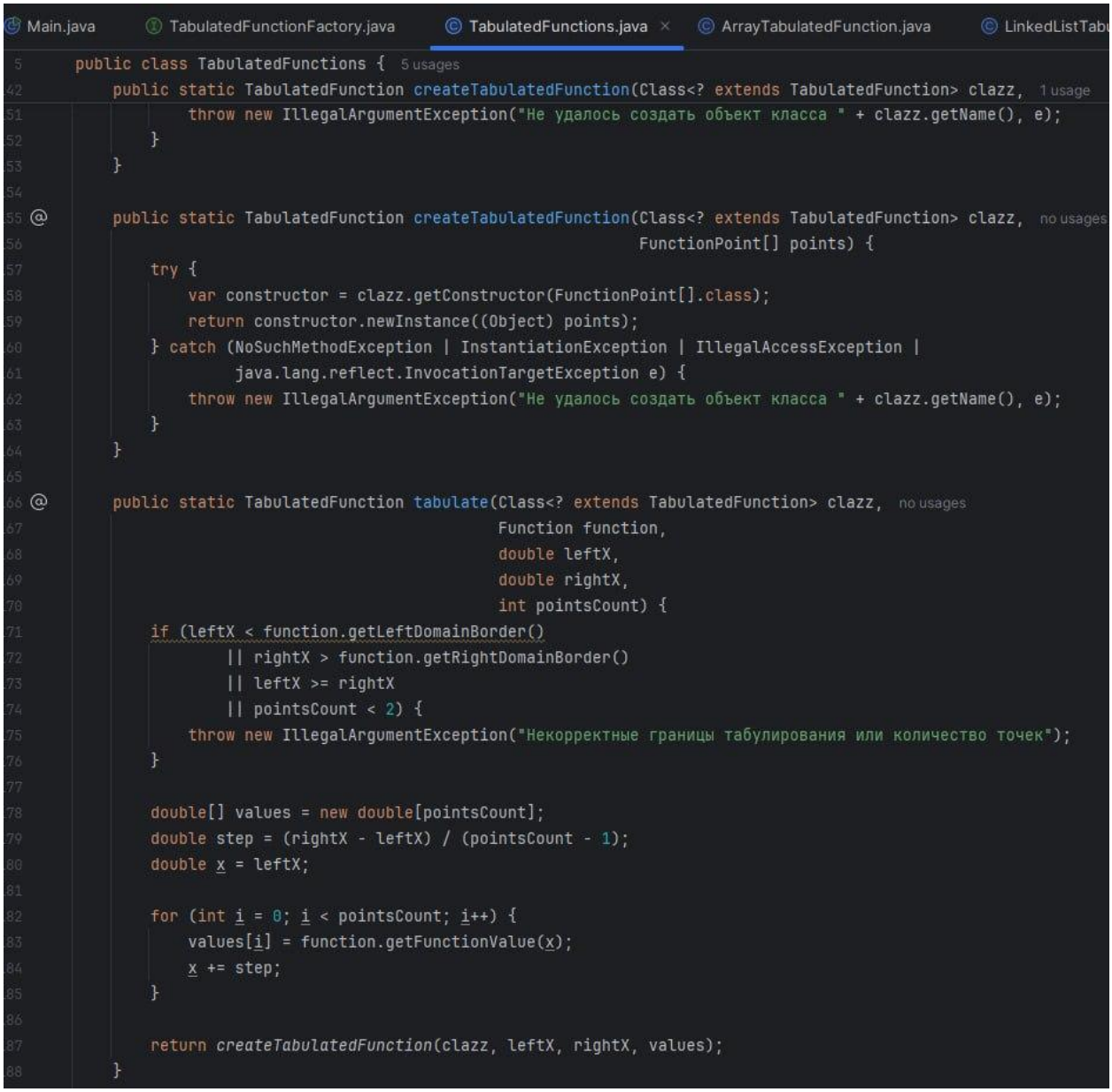
В классе TabulatedFunctions добавил ещё три перегруженных версии метода createTabulatedFunction(). Их параметры должны повторять параметры трёх аналогичных методов, основанных на использовании фабрики, но также эти методы должны получать ссылку типа Class на описание класса, объект которого требуется создать. Сделал так, чтобы в эти методы можно было передать только ссылки на классы, реализующие интерфейс TabulatedFunction.

Новые методы создания объектов должны найти в предложенном классе конструктор с соответствующими типами параметров. С помощью найденного конструктора (в него должны быть переданы фактические параметры) должен быть создан объект табулированной функции. Ссылка на этот объект и должна быть возвращена из метода создания.

В классе TabulatedFunctions перегрузил методы, создающие объекты табулированных функций, добавил версии, принимающие также ссылку

типа Class на описание класса, объект которого требуется создать. Сделал так, чтобы в эти методы можно было передать только ссылки на классы, реализующие интерфейс TabulatedFunction.

Код 3.1



Код 3.2

```java
public static TabulatedFunction readTabulatedFunction(
        Reader in,
        Class<? extends TabulatedFunction> clazz
) throws IOException {

    StreamTokenizer tokenizer = new StreamTokenizer(in);

    tokenizer.nextToken();
    int n = (int) tokenizer.nval;

    FunctionPoint[] points = new FunctionPoint[n];

    for (int i = 0; i < n; i++) {
        tokenizer.nextToken();
        double x = tokenizer.nval;

        tokenizer.nextToken();
        double y = tokenizer.nval;

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

    return createTabulatedFunction(clazz, points);
}

public static TabulatedFunction inputTabulatedFunction(
        InputStream in,
        Class<? extends TabulatedFunction> clazz
) throws IOException {

    DataInputStream dataIn = new DataInputStream(in);

    int n = dataIn.readInt();
    FunctionPoint[] points = new FunctionPoint[n];

    for (int i = 0; i < n; i++) {
        points[i] = new FunctionPoint(
                dataIn.readDouble(),
                dataIn.readDouble()
        );
```

Код 3.3

```java
public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> clazz,
        double leftX,
        double rightX,
        int pointsCount
) {
    try {
        return clazz
                .getConstructor(double.class, double.class, int.class)
                .newInstance(leftX, rightX, pointsCount);
    } catch (Exception e) {
        throw new IllegalArgumentException(e);
    }
}

public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> clazz,
        double leftX,
        double rightX,
        double[] values
) {
    try {
        return clazz
                .getConstructor(double.class, double.class, double[].class)
                .newInstance(leftX, rightX, values);
    } catch (Exception e) {
        throw new IllegalArgumentException(e);
    }
}

public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> clazz,
        FunctionPoint[] points
) {
    try {
        return clazz
                .getConstructor(FunctionPoint[].class)
                .newInstance((Object) points);
    } catch (Exception e) {
        throw new IllegalArgumentException(e);
```

Код 3.4

```java
public static TabulatedFunction tabulate(
        Class<? extends TabulatedFunction> clazz,
        Function function,
        double leftX,
        double rightX,
        int pointsCount
) {
    if (leftX < function.getLeftDomainBorder()
            || rightX > function.getRightDomainBorder()
            || leftX >= rightX
            || pointsCount < 2) {
        throw new IllegalArgumentException("Некорректные границы табулирования или количес
    }

    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    double x = leftX;

    for (int i = 0; i < pointsCount; i++) {
        values[i] = function.getFunctionValue(x);
        x += step;
    }

    return createTabulatedFunction(clazz, leftX, rightX, values);
}
```

Код 3.5

```java
public class Main {
    private static void testReflection() {  1 usage
        System.out.println("\nТестирование рефлексивного создания объектов:");

        TabulatedFunction f;

        f = TabulatedFunctions.createTabulatedFunction(
                ArrayTabulatedFunction.class,  leftX: 0,  rightX: 10,  pointsCount: 3);
        System.out.println("Класс: " + f.getClass().getName());
        System.out.println(f);

        f = TabulatedFunctions.createTabulatedFunction(
                ArrayTabulatedFunction.class,  leftX: 0,  rightX: 10, new double[]{0, 5, 10});
        System.out.println("\nКласс: " + f.getClass().getName());
        System.out.println(f);

        f = TabulatedFunctions.createTabulatedFunction(
                LinkedListTabulatedFunction.class,
                new FunctionPoint[]{
                        new FunctionPoint( x: 0,  y: 0),
                        new FunctionPoint( x: 10,  y: 10)
                }
        );
        System.out.println("\nКласс: " + f.getClass().getName());
        System.out.println(f);

        f = TabulatedFunctions.tabulate(
                LinkedListTabulatedFunction.class, new functions.basic.Sin(),  leftX: 0, Math.PI,  pointsCount: 11);
        System.out.println("\nКласс: " + f.getClass().getName());
        System.out.println(f);
    }
```

Вывод:

```
Рефлексия

Тестирование рефлексивного создания объектов:
Класс: functions.ArrayTabulatedFunction
{(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

Класс: functions.ArrayTabulatedFunction
{(0.0; 0.0), (5.0; 5.0), (10.0; 10.0)}

Класс: functions.LinkedListTabulatedFunction
{(0.0; 0.0), (10.0; 10.0)}

Класс: functions.LinkedListTabulatedFunction
{(0.0; 0.0), (0.3141592653589793; 0.3090169943749474), (0.6283185307179586; 0.5877852522924731), (0.9424777960769379; 0.8090169943749475), (1.2566370614359172; 0.9510565162951535), (1.5707963267948966; 1.0), (1.8849555921538759; 0.9510565162951536), (2.199114857512855; 0.8090169943749475), (2.5132741228718345; 0.5877852522924732), (2.827433388230814; 0.3090169943749475), (3.141592653589793; 1.2246467991473532E-16)}
```