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 Синцов М.Ю

Группа 6201-120303D

Руководитель Борисов Д. С.

САМАРА 2025

Задание 1: Реализация паттерна "Итератор"

Ход выполнения

1. Модифицировал интерфейс TabulatedFunction, добавив наследование от Iterable<FunctionPoint>
2. Реализовал анонимные классы итераторов в классах ArrayTabulatedFunction и LinkedListTabulatedFunction
3. Обеспечил защиту инкапсуляции путем возвращения копий объектов FunctionPoint
4. Реализовал корректную обработку граничных случаев:
    - NoSuchElementException при отсутствии следующего элемента
    - UnsupportedOperationException при вызове remove()

```
public interface TabulatedFunction extends
Function,Serializable,Cloneable,Iterable<FunctionPoint>
```

В ArrayTabulatedFunction

```
public Iterator<FunctionPoint> iterator(){
    return new Iterator<FunctionPoint>() {
        private int currentIndex = 0;
        @Override
        public boolean hasNext() {
            return currentIndex < getPointsCount();
        }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("No more points in tabulated
function");
            }
            FunctionPoint point = getPoint(currentIndex);
            currentIndex++;
            return new FunctionPoint(point.getX(), point.getY());
        }
        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not
```

```
supported");
        }

    };
}
```

B LinkedListTabulatedFunction

```
public Iterator<FunctionPoint> iterator(){
    return new Iterator<FunctionPoint>() {
        private FunctionNode currentNode = head.next;
        @Override
        public boolean hasNext() {
            return currentNode != null && currentNode.point != null;
        }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("No more points in tabulated
function");
            }
            FunctionPoint point = currentNode.point;
            currentNode = currentNode.next;
            return new FunctionPoint(point.getX(), point.getY());
        }
        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not
supported");
        }
    };
}
```

Main:

```
System.out.println("Задание 1");
System.out.println("   ----------------------------------------------------");
```

```
TabulatedFunction f = new ArrayTabulatedFunction(0, 10, new double[] {0, 1, 4,
9, 16, 25});

// Использование улучшенного цикла for
System.out.println("Points in tabulated function:");
for (FunctionPoint p : f) {
    System.out.println(p);
}

// Тестирование с LinkedListTabulatedFunction
TabulatedFunction linkedListFunc = new LinkedListTabulatedFunction(
        new FunctionPoint[] {
                new FunctionPoint(0, 0),
                new FunctionPoint(1, 1),
                new FunctionPoint(2, 4),
                new FunctionPoint(3, 9)
        }
);

System.out.println("\nPoints in linked list tabulated function:");
for (FunctionPoint p : linkedListFunc) {
    System.out.println(p);
}
```

Результат выполнения:

Задание 1

Points in tabulated function:

[0.0,0.0]

[2.0,1.0]

[4.0,4.0]

[6.0,9.0]

[8.0,16.0]

[10.0,25.0]

Points in linked list tabulated function:

[0.0,0.0]

[1.0,1.0]

[2.0,4.0]

[3.0,9.0]

Задание 2: Реализация паттерна "Фабричный метод"

Ход выполнения

1. Создал интерфейс TabulatedFunctionFactory с тремя фабричными методами
2. Реализовал фабрики как вложенные статические классы в ArrayTabulatedFunction и LinkedListTabulatedFunction
3. Добавил статическое поле фабрики в классе TabulatedFunctions
4. Заменил прямое создание объектов на вызовы фабричных методов

Интерфейс TabulatedFunctionFactory :

```
package functions;

public interface TabulatedFunctionFactory {
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
int pointsCount);
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
double[] values);
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points);
}
```

Фабрики:

```
public static class ArrayTabulatedFunctionFactory implements
TabulatedFunctionFactory{
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
int pointsCount){
        return new ArrayTabulatedFunction(leftX,rightX,pointsCount);
    }
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
double[] values){
        return new ArrayTabulatedFunction(leftX,rightX,values);
    }
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points){
        return new ArrayTabulatedFunction(points);
```

```
    }
}
```

```
public static class LinkedListTabulatedFunctionFactory implements
TabulatedFunctionFactory{
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
int pointsCount){
        return new LinkedListTabulatedFunction(leftX,rightX,pointsCount);
    }
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX,
double[] values){
        return new LinkedListTabulatedFunction(leftX,rightX,values);
    }
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points){
        return new LinkedListTabulatedFunction(points);
    }
}
```

```
private static TabulatedFunctionFactory factory = new
ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();
public static void setTabulatedFunctionFactory(TabulatedFunctionFactory
factory){
    TabulatedFunctions.factory = factory;
}
public static TabulatedFunction createTabulatedFunction(double leftX, double
rightX, int pointsCount){
    return factory.createTabulatedFunction(leftX,rightX,pointsCount);
}
public static TabulatedFunction createTabulatedFunction(double leftX, double
rightX, double[] values){
    return factory.createTabulatedFunction(leftX,rightX,values);
}
public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points){
    return factory.createTabulatedFunction(points);
}
```

Измененные методы:

```
public static TabulatedFunction tabulate(Function function, double leftX, double
rightX, int pointsCount){
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не
менее 2");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше
правой");
    }

if(leftX<function.getLeftDomainBorder()||rightX>function.getRightDomainBorder
()){
        throw new IllegalArgumentException("Указанные границы для
табулирования" +
                " выходят за область определения функции");
    }
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);

    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        values[i] = function.getFunctionValue(x);
    }
    return factory.createTabulatedFunction(leftX,rightX,values);
}
```

```
public static TabulatedFunction readTabulatedFunction(Reader in)throws
IOException{
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    try {
        // Настраиваем токенизатор
        tokenizer.resetSyntax();
        tokenizer.wordChars('0', '9');
        tokenizer.wordChars('.', '.');
        tokenizer.wordChars('-', '-');
        tokenizer.wordChars('e', 'e');
        tokenizer.wordChars('E', 'E');
        tokenizer.whitespaceChars(' ', ' ');
        tokenizer.whitespaceChars('\t', '\t');
        tokenizer.whitespaceChars('\n', '\n');
        tokenizer.whitespaceChars('\r', '\r');
```

```
        if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
            throw new IOException("Ожидалось количество точек (целое число)");
        }

        int pointsCount;
        try {
            pointsCount = Integer.parseInt(tokenizer.sval);
        } catch (NumberFormatException e) {
            throw new IOException("Некорректное количество точек: " +
tokenizer.sval, e);
        }

        if (pointsCount < 2) {
            throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
        }


        double[] xValues = new double[pointsCount];
        double[] yValues = new double[pointsCount];

        for (int i = 0; i < pointsCount; i++) {

            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("Ожидалось значение x для точки " + i);
            }

            try {
                xValues[i] = Double.parseDouble(tokenizer.sval);
            } catch (NumberFormatException e) {
                throw new IOException("Некорректное значение x для точки " + i + ":
" + tokenizer.sval, e);
            }


            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("Ожидалось значение y для точки " + i);
            }

            try {
                yValues[i] = Double.parseDouble(tokenizer.sval);
            } catch (NumberFormatException e) {
                throw new IOException("Некорректное значение y для точки " + i + ":
" + tokenizer.sval, e);
```

```
            }
        }


        for (int i = 1; i < pointsCount; i++) {
            if (xValues[i] <= xValues[i-1]) {
                throw new IOException("Некорректные данные: значения X должны
быть упорядочены по возрастанию");
            }
        }


        double leftX = xValues[0];
        double rightX = xValues[xValues.length - 1];
        return factory.createTabulatedFunction(leftX,rightX,yValues);

    } finally {
    }
}
```

Main:

```
System.out.println("Задание 2");
System.out.println("   ----------------------------------------------------");
Function f1 = new Cos();
TabulatedFunction tf;
tf = TabulatedFunctions.tabulate(f1, 0, Math.PI, 11);
System.out.println(tf.getClass());
TabulatedFunctions.setTabulatedFunctionFactory(new
        LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f1, 0, Math.PI, 11);
System.out.println(tf.getClass());
TabulatedFunctions.setTabulatedFunctionFactory(new
        ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f1, 0, Math.PI, 11);
System.out.println(tf.getClass());
```

Результат выполнения:

Задание 2

   ----

class functions.ArrayTabulatedFunction

class functions.LinkedListTabulatedFunction

class functions.ArrayTabulatedFunction

Задание 3: Использование рефлексии

Ход выполнения

1. Добавил перегруженные методы createTabulatedFunction, принимающие параметр типа Class<? extends TabulatedFunction>
2. Реализовал поиск конструкторов с помощью рефлексии
3. Обеспечил обработку исключений с преобразованием в IllegalArgumentException
4. Добавил метод tabulate, использующий рефлексию для создания объектов

Перегруженные методы createTabulatedFunction:

```
public static TabulatedFunction createTabulatedFunction(double leftX, double
rightX, int pointsCount, Class<?extends TabulatedFunction> functionClass){
    try {
        Constructor<? extends TabulatedFunction> constructor =
functionClass.getConstructor(double.class, double.class, int.class);
        return constructor.newInstance(leftX, rightX, pointsCount);
    } catch (NoSuchMethodException | InstantiationException |
             IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException("Cannot create tabulated function", e);
    }
}
public static TabulatedFunction createTabulatedFunction(double leftX, double
rightX, double[] values, Class<?extends TabulatedFunction> functionClass){
    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("Cannot create tabulated function", e);
    }
}
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("Cannot create tabulated function", e);
    }
}
```

Остальные перегруженные методы:

```
public static TabulatedFunction tabulate(Function function, double leftX, double
rightX, int pointsCount, Class<?extends TabulatedFunction> functionClass){
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не
менее 2");
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница должна быть меньше
правой");
    }

if(leftX<function.getLeftDomainBorder()||rightX>function.getRightDomainBorder
()){
        throw new IllegalArgumentException("Указанные границы для
табулирования" +
                " выходят за область определения функции");
    }
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);

    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        values[i] = function.getFunctionValue(x);
    }
    try {
        Constructor<? extends TabulatedFunction> constructor =
functionClass.getConstructor(double.class, double.class, int.class);
        return constructor.newInstance(leftX, rightX, pointsCount);
    } catch (NoSuchMethodException | InstantiationException |
             IllegalAccessException | InvocationTargetException e) {
        throw new IllegalArgumentException("Cannot create tabulated function", e);
    }
}
```

```
public static TabulatedFunction readTabulatedFunction(Reader in, Class<?extends
TabulatedFunction> functionClass)throws IOException{
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    try {
        // Настраиваем токенизатор
        tokenizer.resetSyntax();
        tokenizer.wordChars('0', '9');
        tokenizer.wordChars('.', '.');
        tokenizer.wordChars('-', '-');
        tokenizer.wordChars('e', 'e');
        tokenizer.wordChars('E', 'E');
        tokenizer.whitespaceChars(' ', ' ');
        tokenizer.whitespaceChars('\t', '\t');
        tokenizer.whitespaceChars('\n', '\n');
        tokenizer.whitespaceChars('\r', '\r');


        if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
            throw new IOException("Ожидалось количество точек (целое число)");
        }

        int pointsCount;
        try {
            pointsCount = Integer.parseInt(tokenizer.sval);
        } catch (NumberFormatException e) {
            throw new IOException("Некорректное количество точек: " +
tokenizer.sval, e);
        }

        if (pointsCount < 2) {
            throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
        }


        double[] xValues = new double[pointsCount];
        double[] yValues = new double[pointsCount];

        for (int i = 0; i < pointsCount; i++) {

            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("Ожидалось значение x для точки " + i);
            }
```

```
            try {
                xValues[i] = Double.parseDouble(tokenizer.sval);
            } catch (NumberFormatException e) {
                throw new IOException("Некорректное значение x для точки " + i + ":
" + tokenizer.sval, e);
            }


            if (tokenizer.nextToken() != StreamTokenizer.TT_WORD) {
                throw new IOException("Ожидалось значение y для точки " + i);
            }

            try {
                yValues[i] = Double.parseDouble(tokenizer.sval);
            } catch (NumberFormatException e) {
                throw new IOException("Некорректное значение y для точки " + i + ":
" + tokenizer.sval, e);
            }
        }


        for (int i = 1; i < pointsCount; i++) {
            if (xValues[i] <= xValues[i-1]) {
                throw new IOException("Некорректные данные: значения X должны
быть упорядочены по возрастанию");
            }
        }


        double leftX = xValues[0];
        double rightX = xValues[xValues.length - 1];
        try {
            Constructor<? extends TabulatedFunction> constructor =
functionClass.getConstructor(double.class, double.class, int.class);
            return constructor.newInstance(leftX, rightX, pointsCount);
        } catch (NoSuchMethodException | InstantiationException |
                 IllegalAccessException | InvocationTargetException e) {
            throw new IllegalArgumentException("Cannot create tabulated function",
e);
        }

    } finally {
    }
}
```

```
public static TabulatedFunction inputTabulatedFunction(InputStream in,
Class<?extends TabulatedFunction> functionClass)throws IOException {
    DataInputStream dataIn = new DataInputStream(in);
    try {
        int pointsCount = dataIn.readInt();

        if (pointsCount < 2) {
            throw new IOException("Некорректные данные: количество точек
должно быть не менее 2");
        }

        double[] xValues = new double[pointsCount];
        double[] yValues = new double[pointsCount];
        for (int i = 0; i < pointsCount; i++) {
            xValues[i] = dataIn.readDouble();
            yValues[i] = dataIn.readDouble();
        }

        for (int i = 1; i < pointsCount; i++) {
            if (xValues[i] <= xValues[i - 1]) {
                throw new IOException("Некорректные данные: значения X должны
быть упорядочены по возрастанию");
            }
        }
        double leftX = xValues[0];
        double rightX = xValues[xValues.length - 1];
        try {
            Constructor<? extends TabulatedFunction> constructor =
functionClass.getConstructor(double.class, double.class, int.class);
            return constructor.newInstance(leftX, rightX, pointsCount);
        } catch (NoSuchMethodException | InstantiationException |
                 IllegalAccessException | InvocationTargetException e) {
            throw new IllegalArgumentException("Cannot create tabulated function",
e);
        }

    } catch (EOFException e) {
        throw new IOException("Неожиданный конец потока: данные неполные",
e);
    } catch (IllegalArgumentException e) {
        throw new IOException("Некорректные данные: " + e.getMessage(), e);
    } finally {

    }
}
```

Main:

```
System.out.println("Задание 3");
System.out.println("   ----------------------------------------------------");
TabulatedFunction f2;

f2 = TabulatedFunctions.createTabulatedFunction( 0, 10, 3,
ArrayTabulatedFunction.class);
System.out.println(f2.getClass());
System.out.println(f2);

f2 = TabulatedFunctions.createTabulatedFunction( 0, 10, new double[] {0,
10},ArrayTabulatedFunction.class);
System.out.println(f2.getClass());
System.out.println(f2);

f2 =
TabulatedFunctions.createTabulatedFunction(LinkedListTabulatedFunction.class,
        new FunctionPoint[] {
                new FunctionPoint(0, 0),
                new FunctionPoint(10, 10)
        }
);
System.out.println(f2.getClass());
System.out.println(f2);

f2 = TabulatedFunctions.tabulate(new Sin(), 0, Math.PI, 11,
LinkedListTabulatedFunction.class);
System.out.println(f2.getClass());
System.out.println(f2);
```

Результат выполнения:

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], [0.3141592653589793,0.0], [0.6283185307179586,0.0], [0.9424777960769379,0.0], [1.2566370614359172,0.0], [1.5707963267948966,0.0], [1.8849555921538759,0.0], [2.199114857512855,0.0], [2.5132741228718345,0.0], [2.827433388230814,0.0], [3.141592653589793,0.0]}