МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

(Самарский университет)

ОТЧЕТ ПО

ЛАБОРАТОРНОЙ РАБОТЕ №7

«Работа с паттернами и использование рефлексии Java»

По курсу

Объектно-ориентированное программирование

Работу выполнила:

студент группы 6201-120303D,

Нагуманова В. Э.

Самара 2025

Содержание

Задание №1.

Лабораторную работу начинаю с добавления итераторов во все объекты TabulatedFunction. Для их реализации добавляю наследование. Затем добавляю итератор в ArrayTabulatedFunction и в LinkedListTabulatedFunction.

```
public interface TabulatedFunction extends Iterable<FunctionPoint>, Function, Cloneable {  13 usages  2 implementations
```

Рис. 1

```
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private int index = 0;  //текущий индекс для итерации  3 usages

        @Override
        public boolean hasNext() { //проверка на существование точки
            return index < pointsCount;
        }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Следующей точки не существует");
            }
            FunctionPoint point = points[index];
            FunctionPoint copyPoint = new FunctionPoint(point.getX(), point.getY()); //создаем копию точки
            index++;
            return copyPoint;//возвращает копию
        }

        @Override
        public void remove() { //удаление точки не предусмотрено
            throw new UnsupportedOperationException("Удаление точки невозможно");
        }
```

Рис. 2

```java
@Override
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private FunctionNode curr = head.getNext();

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("Элемента не существует");//если точек нет
            }
            FunctionPoint point = new FunctionPoint(curr.getPoint());
            curr = curr.getNext();
            return point;
        }

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

Рис. 3

Оба итератора возвращают копии точек чтобы не нарушить инкапсуляцию, метод remove() не поддерживается.

Затем в main я проверила код через вывод всех точек функции в цикле for-each.

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("\n1.Проверка работы итераторов классов табулированных функций. \n");
        TabulatedFunction arrFunc = new ArrayTabulatedFunction( leftX: 0,  rightX: 10,  pointsCount: 5);//создание табулированной функции

        System.out.println("Массив:");
        for (FunctionPoint p : arrFunc) { //цикл for-each по точкам функции
            System.out.println(p); //вывод текущей точки
        }

        TabulatedFunction linkFunc = new LinkedListTabulatedFunction( leftX: 0,  rightX: 10,  pointsCount: 5);//создание табулированной функции
        System.out.println("\nСписок:");
        for (FunctionPoint p : linkFunc) {// цикл for-each по точкам функции
            System.out.println(p); //вывод текущей точки
        }
    }
```

Рис. 4

```
1.Проверка работы итераторов классов табулированных функций.

Массив:
(0,0; 0,0)
(2,5; 0,0)
(5,0; 0,0)
(7,5; 0,0)
(10,0; 0,0)

Список:
(0,0; 0,0)
(2,5; 0,0)
(5,0; 0,0)
(7,5; 0,0)
(10,0; 0,0)
```

Рис. 5

Задание №2.

Для выполнения этого задания требуется реализовать паттерн «Фабричный метод». В пакете functions описала базовый интерфейс фабрик табулированных функций TabulatedFunctionFactory

```java
package functions;

public interface TabulatedFunctionFactory {  no usages
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] values);  no usages
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount);  no usages
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values);  no usages
}
```

Рис. 6

в классах ArrayTabulatedFunction и LinkedListTabulatedFunction описала классы фабрик ArrayTabulatedFunctionFactory и LinkedListTabulatedFactory, реализующие интерфейс фабрики и порождающие объекты соответствующих классов табулированных функций.

```java
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {  no usages
    @Override  no usages
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {
        return new ArrayTabulatedFunction(leftX, rightX, pointsCount);//массивная функция с границами и количеством точек
    }

    @Override  no usages
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return new ArrayTabulatedFunction(leftX, rightX, values);//массивная функция с границами и значениями
    }

    @Override  no usages
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return new ArrayTabulatedFunction(points);//массивная табулированная функция из точек
    }
}
```

Рис. 7

```java
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); //создание связной функции с границами и количеством точек
    }

    @Override  no usages
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return new LinkedListTabulatedFunction(leftX, rightX, values); //создание связной функции с границами и значением
    }

    @Override  no usages
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return new LinkedListTabulatedFunction(points); //создание табулированной связной функции из точек
    }
}
```

Рис. 8

```java
private static TabulatedFunctionFactory factory = new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory(); //фабрика по умолчанию  4 usages

//метод для замены фабрики
public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) {  no usages
    factory = newFactory; //устанавливаем новую фабрику
}

//три перегруженных метода
public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointsCount) {  no usages
    return factory.createTabulatedFunction(leftX, rightX, pointsCount); //создает функцию через фабрику с границами и количеством точек
}

public static TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {  no usages
    return factory.createTabulatedFunction(leftX, rightX, values); //создает функцию через фабрику с границами и значениями
}

public static TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {  no usages
    return factory.createTabulatedFunction(points); //создает функцию через фабрику из точек
}
```

Рис. 9

В классе TabulatedFunctions объявила и описала все неибходимые методы, реализован метод setTabulatedFunctionFactory(), позволяющий заменить текущую фабрику на другую. В существующих методах класса теперь возвращаемые табулированные функции создаются не напрямую, а с помощью фабрики.

```java
public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {  no usages
    if (function == null)  throw new IllegalArgumentException("Функция не задана");

    if (pointsCount < 2) throw new IllegalArgumentException("Количсетво точек должно быть не менее 2"); //проверка на минимальное количсетво точек

    if (leftX >= rightX) throw new IllegalArgumentException("Левая граница должнна быть меньше правой");

    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder())
    {
        throw new IllegalArgumentException("Границы табулирования выходят за область определения"); //проверка на вхождение границ в область определения
    }

    TabulatedFunction tabulatedFunc = createTabulatedFunction(leftX, rightX, pointsCount);

    //записываем значения в массив
    for (int i = 0; i < pointsCount; i++){
        double x = tabulatedFunc.getPointX(i);
        double y = function.getFunctionValue(x);
        tabulatedFunc.setPointY(i, y);
    }
    return tabulatedFunc;
}
```

Рис. 10

```java
//чтение данных из файла
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {  no usages
    DataInputStream inputData = new DataInputStream(in);
    int pointsCount = inputData.readInt();
    FunctionPoint[] points = new FunctionPoint[pointsCount];

    for (int i = 0; i < pointsCount; i++) {
        double x = inputData.readDouble();
        double y = inputData.readDouble();
        points[i] = new FunctionPoint(x, y);
    }

    return createTabulatedFunction(points);

}
```

Рис. 11

```java
    public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {  no usages
        StreamTokenizer tokenizer = new StreamTokenizer(in); //токенизатор для чтения
        tokenizer.nextToken();
        int pointsCount = (int) tokenizer.nval; //кол-во точек\

        //читаем количество точек и создаем массив
        FunctionPoint[] points = new FunctionPoint[pointsCount];

        for (int i = 0; i < pointsCount; i++) {
            tokenizer.nextToken();
            double x = tokenizer.nval; //x
            tokenizer.nextToken();
            double y = tokenizer.nval; // y
            points[i] = new FunctionPoint(x, y); //новая точка
        }

        return createTabulatedFunction(points);
    }
```

Рис. 12

Вывод работы

```
2.Проверка работы фабрик.

class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction
```

Рис. 13

Задание №3.

Для этого задания требуется реализовать рефлексивное создание объектов. Три перегруженных метода createTabulatedFunction() принимают параметр Class <?>, добавила проверки.

```
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, int pointsCount) {  1 usage
    try {
        //проверяем, что класс реализует tabulatedFunction
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
        }
        Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, int.class); //ищем нужный конструктор

        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount); //создаем объект через рефлексию
    } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException | InstantiationException e) {
        throw new IllegalArgumentException("Ошибка при создании объекта", e);
    }
}

public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, double leftX, double rightX, double[] values) {  no usages
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) { //проверяем, что класс реализует tabulatedFunction
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
        }
        Constructor<?> constructor = functionClass.getConstructor(double.class, double.class, double[].class); //щем нужный конструктор

        return (TabulatedFunction) constructor.newInstance(leftX, rightX, values); //создаем объект через рефлексию
    } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException | InstantiationException e) {
        throw new IllegalArgumentException("Ошибка при создании объекта", e);
    }
}
```

Рис. 14

```
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass, FunctionPoint[] points) {  2 usages
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) { //проверяем, что класс реализует TabulatedFunction
            throw new IllegalArgumentException("Класс должен реализовывать интерфейс TabulatedFunction");
        }
        Constructor<?> constructor = functionClass.getConstructor(FunctionPoint[].class); //ищем нужный конструктор

        return (TabulatedFunction) constructor.newInstance((Object) points); //создаем объект через рефлексию
    } catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException | InstantiationException e) {
        throw new IllegalArgumentException("Ошибка при создании объекта", e);
    }
}
```

Рис. 15

```java
//метод tabulate с рефлексией
public static TabulatedFunction tabulate(Class<?> functionClass, Function function, double leftX, double rightX, int pointsCount) {  no usages
    if (function == null)
        throw new IllegalArgumentException("Функция null"); //проверяем что функция не null
    if (pointsCount < 2) {
        throw new IllegalArgumentException("Недостаточное количество точек"); //проверка на минимальное кол-во точек
    }
    if (leftX >= rightX) {
        throw new IllegalArgumentException("Левая граница меньше правой");
    }
    if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Границы табулирования выходят за область определения"); //проверка на нахлждение границ в области определения
    }

    TabulatedFunction tabulatedFunc = createTabulatedFunction(functionClass, leftX, rightX, pointsCount);
    // записываем значения в массив
    for (int i = 0; i < pointsCount; i++){
        double x = tabulatedFunc.getPointX(i);
        double y = function.getFunctionValue(x);
        tabulatedFunc.setPointY(i, y);
    }

    return tabulatedFunc;
}
```

Рис.16

```java
public static TabulatedFunction inputTabulatedFunction(Class<?> functionClass, InputStream in) throws IOException {  no usages
    DataInputStream inputData = new DataInputStream(in); //поток для чтения данных
    int pointsCount = inputData.readInt(); //читаем кол-во точек
    FunctionPoint[] points = new FunctionPoint[pointsCount]; //массив для точек

    for (int i = 0; i < pointsCount; i++) {
        double x = inputData.readDouble(); //координата x
        double y = inputData.readDouble(); //координата y
        points[i] = new FunctionPoint(x, y); //новая точка
    }

    return createTabulatedFunction(functionClass, points); //создание табулированной функции через рефлексию

}

public static TabulatedFunction readTabulatedFunction(Class<?> functionClass, Reader in) throws IOException {  no usages
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    tokenizer.nextToken();
    int pointsCount = (int) tokenizer.nval;
    FunctionPoint[] points = new FunctionPoint[pointsCount];

    for (int i = 0; i < pointsCount; i++) {
        tokenizer.nextToken();
        double x = tokenizer.nval; //координата x
        tokenizer.nextToken();
        double y = tokenizer.nval; //координата y
        points[i] = new FunctionPoint(x, y);//создаем новую точку
    }

    return createTabulatedFunction(functionClass,points);
}
```

Рис. 17

В main проверила работу методов, предложенным кодом:

```
3. Проверка рефлексии.
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)(0,3; 0,3)(0,6; 0,6)(0,9; 0,8)(1,3; 1,0)(1,6; 1,0)(1,9; 1,0)(2,2; 0,8)(2,5; 0,6)(2,8; 0,3)(3,1; 0,0)}
```

Рис. 18