Министерство науки и высшего образования Российской Федерации

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

Институт информатики и кибернетики

Отчет по лабораторной работе №7

Дисциплина: «ООП»

Тема «Паттерны проектирования и рефлексия»

Выполнил: Пантелеев Ю.В.

Группа: 6201-120303

Самара, 2025

Задание на лабораторную работу

Задание 1

Я сделал так, чтобы все объекты типа TabulatedFunction можно было использовать в качестве объекта-агрегата в «улучшенном цикле for» (вариант for-each)

TabulatedFunction:

```
package functions;
import java.util.Iterator;

public interface TabulatedFunction extends Function, Cloneable, Iterable<FunctionPoint>{  34 usages  2 i
    public int getPointsCount();  8 usages  2 implementations
    public FunctionPoint getPoint(int index);  3 usages  2 implementations
    public void setPoint(int index, FunctionPoint point) throws InappropriateFunctionPointException;
    public double getPointX(int index);  2 usages  2 implementations
    public void setPointX(int index, double x) throws InappropriateFunctionPointException;  no usages
    public double getPointY(int index);  2 usages  2 implementations
    public void setPointY(int index, double y);  no usages  2 implementations
    public void deletePoint(int index);  no usages  2 implementations
    public void addPoint(FunctionPoint point) throws InappropriateFunctionPointException;  1 usage  2 im
    public Object clone();  2 implementations
    Iterator<FunctionPoint> iterator();  2 implementations
}
```

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

Итератор в ArrayTabulatedFunction:

```
@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");
            }
            return new FunctionPoint(points[currentIndex++]);
        }

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

Итератор в LinkedListTabulatedFunction:

```
public Iterator<FunctionPoint> iterator() {
    return new Iterator<FunctionPoint>() {
        private FunctionNode currentNode = head;  3 usages
        private  int counter = 0;  2 usages
        @Override
        public boolean hasNext() { return counter<pointsCount; }

        @Override
        public FunctionPoint next() {
            if(!hasNext()){
                throw  new java.util.NoSuchElementException("Нет следующего элемента");
            }
            FunctionPoint p = currentNode.point;
            currentNode = currentNode.next;
            counter++;

            return new FunctionPoint(p.getX(),p.getY());
        }

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

Проверка в main:

```
System.out.println("=== Задание 1: Итератор ===");
Function sin = new Sin();
TabulatedFunction arrayFunc = TabulatedFunctions.tabulate(sin, leftX: 0, Math.PI, pointsCount: 5);

System.out.println("Итерация по точкам (улучшенный цикл for):");
for (FunctionPoint p : arrayFunc) {
    System.out.println(p);
}

System.out.println("\n=== Задание 2: Фабрики ===");
Function cos = new Cos();
TabulatedFunction tf;

tf = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("Тип объекта (по умолчанию): " + tf.getClass().getSimpleName());

TabulatedFunctions.setTabulatedFunctionFactory(
        new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory()
);
tf = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("Тип объекта (после смены фабрики): " + tf.getClass().getSimpleName());

StringReader reader = new StringReader( s: "3 0 0 5 25 10 100");
TabulatedFunction readFunc = TabulatedFunctions.readTabulatedFunction(reader);
System.out.println("Тип объекта, прочитанного из Reader: " + readFunc.getClass().getSimpleName());
```

Результат:

```
=== Задание 1: Итератор ===
Итерация по точкам (улучшенный цикл for):
(0,00; 0,00)
(0,79; 0,71)
(1,57; 1,00)
(2,36; 0,71)
(3,14; 0,00)
```

Задание 2

В пакете functions я описал базовый интерфейс фабрик табулированных функций TabulatedFunctionFactory:

```
package functions;

public interface TabulatedFunctionFactory {  5 usages  2 implementations
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointCount);
    Rename usages
    TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values);
    TabulatedFunction createTabulatedFunction(FunctionPoint[] points);  2 usages  2 implementations
}
```

Далее я описал в классах ArrayTabulatedFunction и LinkedListTabulatedFunction классы фабрик:

ArrayTabulatedFunctionFactory:

```
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory{  1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointCount){  1
        return new ArrayTabulatedFunction(leftX,rightX,pointCount);
    }

    @Override  1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }

    @Override  1 usage
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return new ArrayTabulatedFunction(points);
    }
}
```

LinkedListTabulatedFunctionFactory:

```
public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory{  1 usage
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int pointCount){  2
        return new LinkedListTabulatedFunction(leftX,rightX,pointCount);
    }

    @Override  2 usages
    public TabulatedFunction createTabulatedFunction(double leftX, double rightX, double[] values) {
        return new LinkedListTabulatedFunction(leftX, rightX, values);
    }

    @Override  2 usages
    public TabulatedFunction createTabulatedFunction(FunctionPoint[] points) {
        return new LinkedListTabulatedFunction(points);
    }
}
```

В классе TabulatedFunctions объявил приватное статическое поле типа TabulatedFunctionFactory, метод setTabulatedFunctionFactory() и описал три перегруженных метода TabulatedFunction createTabulatedFunction():

```
package functions;

import java.io.*;
import java.lang.reflect.Constructor;

public final class TabulatedFunctions {  9 usages
    public static final double EPSILON = 1e-10;  no usages

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

    private TabulatedFunctions() {  no usages
        throw new AssertionError( detailMessage: "Экземпляры утилитного класса не создаются");
    }

    public static void setTabulatedFunctionFactory(TabulatedFunctionFactory factory) {  1 usage
        TabulatedFunctions.factory = factory;
    }


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

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

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

```java
public static TabulatedFunction tabulate(Function f, double leftX, double rightX, int pointsCount) {
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        values[i] = f.getFunctionValue( x: leftX + i * step);
    }
    return createTabulatedFunction(leftX, rightX, values);
}
```

```java
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {  no usages
    PrintWriter dataOut = new PrintWriter(out);
    dataOut.print(function.getPointsCount() + " ");
    for (int i = 0; i < function.getPointsCount(); i++) {
        dataOut.print(function.getPointX(i) + " " + function.getPointY(i) + (i < function.getPointsCount() - 1 ? " " : ""));
    }
    dataOut.flush();
}
```

```java
public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {  1 usage
    StreamTokenizer tokenizer = new StreamTokenizer(in);
    tokenizer.parseNumbers();
    if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) throw new IOException("Ошибка формата");
    int pointCount = (int) tokenizer.nval;
    FunctionPoint[] points = new FunctionPoint[pointCount];
    for (int i = 0; i < pointCount; i++) {
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) throw new IOException("Ошибка X");
        double x = tokenizer.nval;
        if (tokenizer.nextToken() != StreamTokenizer.TT_NUMBER) throw new IOException("Ошибка Y");
        double y = tokenizer.nval;
        points[i] = new FunctionPoint(x, y);
    }
    return createTabulatedFunction(points);
}
```

Проверка в main:

```java
System.out.println("\n=== Задание 2: Фабрики ===");
Function cos = new Cos();
TabulatedFunction tf;

tf = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("Тип объекта (по умолчанию): " + tf.getClass().getSimpleName());

TabulatedFunctions.setTabulatedFunctionFactory(
        new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory()
);
tf = TabulatedFunctions.tabulate(cos, leftX: 0, Math.PI, pointsCount: 11);
System.out.println("Тип объекта (после смены фабрики): " + tf.getClass().getSimpleName());

StringReader reader = new StringReader( s: "3 0 0 5 25 10 100");
TabulatedFunction readFunc = TabulatedFunctions.readTabulatedFunction(reader);
System.out.println("Тип объекта, прочитанного из Reader: " + readFunc.getClass().getSimpleName());
```

Результат:

```
=== Задание 2: Фабрики ===
Тип объекта (по умолчанию): ArrayTabulatedFunction
Тип объекта (после смены фабрики): LinkedListTabulatedFunction
Тип объекта, прочитанного из Reader: LinkedListTabulatedFunction
```

Задание 3

В классе TabulatedFunctions добавьте ещё три перегруженных версии метода createTabulatedFunction() и перегрузил метод tabulate:

```
public static TabulatedFunction createTabulatedFunction(  1 usage
        Class<? extends TabulatedFunction> functionClass, double leftX, double rightX, int pointsCount) {
    try {
        Constructor<? extends TabulatedFunction> c = functionClass.getConstructor(double.class, double.class, int.class);
        return c.newInstance(leftX, rightX, pointsCount);
    } catch (Exception e) {
        throw new IllegalArgumentException(e);
    }
}

public static TabulatedFunction createTabulatedFunction(  1 usage
        Class<? extends TabulatedFunction> functionClass, double leftX, double rightX, double[] values) {
    try {
        Constructor<? extends TabulatedFunction> c = functionClass.getConstructor(double.class, double.class, double[].class);
        return c.newInstance(leftX, rightX, values);
    } catch (Exception e) {
        throw new IllegalArgumentException(e);
    }
}

public static TabulatedFunction createTabulatedFunction(  1 usage
        Class<? extends TabulatedFunction> functionClass, FunctionPoint[] points) {
    try {
        Constructor<? extends TabulatedFunction> c = functionClass.getConstructor(FunctionPoint[].class);
        return c.newInstance((Object) points);
    } catch (Exception e) {
        throw new IllegalArgumentException(e);
    }
}
```

```
public static TabulatedFunction tabulate(Class<? extends TabulatedFunction> functionClass,  1 usage
                                         Function f, double leftX, double rightX, int pointsCount) {
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);
    for (int i = 0; i < pointsCount; i++) {
        values[i] = f.getFunctionValue( x: leftX + i * step);
    }
    return createTabulatedFunction(functionClass, leftX, rightX, values);
}
```

Проверка в main:

```
System.out.println("\n=== Задание 3: Рефлексивные методы ===");
TabulatedFunction fRef;

fRef = TabulatedFunctions.createTabulatedFunction(
        ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3
);
System.out.println("Рефлексия (границы): " + fRef.getClass().getSimpleName());
System.out.println(fRef);

FunctionPoint[] points = {
        new FunctionPoint( x: 0, y: 0),
        new FunctionPoint( x: 10, y: 10)
};
fRef = TabulatedFunctions.createTabulatedFunction(
        LinkedListTabulatedFunction.class, points
);
System.out.println("Рефлексия (массив точек): " + fRef.getClass().getSimpleName());
System.out.println(fRef);

fRef = TabulatedFunctions.tabulate(
        LinkedListTabulatedFunction.class, new Sin(), leftX: 0, Math.PI, pointsCount: 11
);
System.out.println("Рефлексия (tabulate Sin): " + fRef.getClass().getSimpleName());
System.out.println(fRef);
```

Результат:

```
=== Задание 3: Рефлексивные методы ===
Рефлексия (границы): ArrayTabulatedFunction
{(0,00; 0,00), (5,00; 0,00), (10,00; 0,00)}
Рефлексия (массив точек): LinkedListTabulatedFunction
{(0,00; 0,00), (10,00; 10,00)}
Рефлексия (tabulate Sin): LinkedListTabulatedFunction
{(0,00; 0,00), (0,31; 0,31), (0,63; 0,59), (0,94; 0,81), (1,26; 0,95), (1,57; 1,00), (1,88; 0,95), (2,20; 0,81), (2,51; 0,59), (2,83; 0,31), (3,14; 0,00)}
```