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 Кафедра технической кибернетики

Лабораторная работа №7

По дисциплине «Обьектно-ориентированное программирование»

Студент: Кораблев Д.С.

Группа: 6203-010302D

Самара, 2025

Задание 1

Сделал так, чтобы все объекты типа TabulatedFunction можно было использовать в качестве объекта-агрегата в «улучшенном цикле for» (вариант for-each), извлекаемые объекты при этом имеют тип FunctionPoint.

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

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

Классы итераторов сделал анонимными. В соответствии с паттерном «Итератор» итераторы работают эффективно и используют знание о внутренней структуре объектов, а не вызывают публичные методы объекта табулированной функции.

Метод удаления всегда выбрасывает исключение UnsupportedOperationException.

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

```java
package functions;

import java.util.Iterator;

public interface TabulatedFunction extends Function, Cloneable, Iterable<FunctionPoint> {
    int getPointsCount();
    FunctionPoint getPoint(int index) throws FunctionPointIndexOutOfBoundsException;
    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;
    TabulatedFunction clone();
}
```

```java
package functions;

import java.io.Serializable;
import java.util.NoSuchElementException;
import java.util.Objects;
import java.util.Iterator;

public class LinkedListTabulatedFunction implements TabulatedFunction,Serializable {
    private class FunctionNode
    {
    private FunctionPoint point;
    private FunctionNode prev;
    private FunctionNode next;

    public FunctionNode()
    {
```

```java
package functions;
import java.io.*;
import java.util.Objects;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable {

    private FunctionPoint points[];
    private int pointsCount;



    public ArrayTabulatedFunction() {
        points = new FunctionPoint[10]; // начальная емкость
        pointsCount = 0;
    }

    public ArrayTabulatedFunction(double leftX, double rightX, int pointsCount) {
        if(leftX >= rightX)
        {
            throw new IllegalArgumentException("the left border is larger than the right one
        }

        if(pointsCount < 2)
        {
            throw new IllegalArgumentException("Number of points is less than 2");
        }
        this.pointsCount = pointsCount;
        points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);
        for (int i = 0; i < pointsCount; i++) {
            points[i] = new FunctionPoint(leftX + i * step, y: 0);
```

```
Тест задания 1
Тест 1: ArrayTabulatedFunction
Точки функции (for-each):
(0.0;0.0)
(2.0;1.0)
(4.0;4.0)
(6.0;9.0)
(8.0;16.0)
(10.0;25.0)

Тест 2: LinkedListTabulatedFunction
Точки функции (for-each):
(0.0;0.0)
(1.0;1.0)
(2.0;8.0)
(3.0;27.0)
(4.0;64.0)

Тест 3: Проверка инкапсуляции
Измененная копия первой точки: (999.0;999.0)
Оригинальная первая точка: (0.0;0.0)
Инкапсуляция сохранена: true

Тест 4: Проверка remove()
UnsupportedOperationException поймано: Remove operation is not supported

Тест 5: Проверка NoSuchElementException
NoSuchElementException поймано: No more points in tabulated function
```

Рисунки 1-4

Задание 2

В пакете functions описал базовый интерфейс фабрик табулированных функций TabulatedFunctionFactory. Интерфейс объявляет три перегруженных метода

TabulatedFunction createTabulatedFunction(), параметры которых соответствуют параметрам конструкторов классов табулированных функций.

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.

В классе TabulatedFunctions описал три перегруженных метода TabulatedFunction createTabulatedFunction(), возвращающих объекты табулированных функций, созданные с помощью текущей фабрики. Параметры методов соответствуют параметры методов фабрики.

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

```
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[] array)
    {
        return new ArrayTabulatedFunction(array);
    }

    }
```

```
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[] array)
    {
        return new LinkedListTabulatedFunction(array);
    }
    }
```

```java
// Приватное статическое поле фабрики с инициализацией Array фабрикой по умолчанию
private static TabulatedFunctionFactory factory =
    new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory();

// Метод для замены фабрики
public static void setTabulatedFunctionFactory(TabulatedFunctionFactory newFactory) {
    factory = newFactory;
}

// Три перегруженных метода createTabulatedFunction, использующие текущую фабрику
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[] array) {
    return factory.createTabulatedFunction(array);
}
```

```java
System.out.println("\nТест задания 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.LinkedListTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
System.out.println(tf.getClass());

TabulatedFunctions.setTabulatedFunctionFactory(new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(f, leftX: 0, Math.PI, pointsCount: 11);
System.out.println(tf.getClass());
```

```
Тест задания 2
class functions.ArrayTabulatedFunction
class functions.LinkedListTabulatedFunction
class functions.ArrayTabulatedFunction
```

Рисунки 5-9

Задание 3

В классе TabulatedFunctions добавил ещё три перегруженных версии метода createTabulatedFunction(). Их параметры повторяют 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 находят в предложенном классе конструктор с соответствующими типами параметров (например, двумя параметрами типа double и одним параметром типа int для метода, создающего объект табулированной функции по левой и правой границе области определения и количеству точек). С помощью найденного конструктора (в него должны быть переданы фактические параметры) создаться объект табулированной функции.

Ссылка на этот объект и должна быть возвращена из метода создания.

В классе TabulatedFunctions перегрузил методы, создающие объекты табулированных функций, добавив 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.

```java
// Метод создания функции через рефлексию с тремя параметрами
public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> functionClass,
        double leftX, double rightX, int pointsCount) {

    try {
        // Получаем конструктор с параметрами (double, double, int)
        Constructor<? extends TabulatedFunction> constructor = functionClass.getConstructor(double.class, double.class, int.class);

        // Создаем объект через конструктор
        return constructor.newInstance(leftX, rightX, pointsCount);

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Класс " + functionClass.getName() + " не имеет конструктора (double, double, int)", e);
    } catch (InstantiationException e) {
        throw new IllegalArgumentException("Невозможно создать экземпляр класса " + functionClass.getName() + " (абстрактный класс?)", e);
    } catch (IllegalAccessException e) {
        throw new IllegalArgumentException("Конструктор класса " + functionClass.getName() + " недоступен", e);
    } catch (InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка в конструкторе класса " + functionClass.getName(), e);
    }
}

// Метод создания функции через рефлексию с массивом значений
public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> functionClass,
        double leftX, double rightX, double[] values) {

    try {
        // Получаем конструктор с параметрами (double, double, double[])
        Constructor<? extends TabulatedFunction> constructor = functionClass.getConstructor(double.class, double.class, double[].class);

        // Создаем объект через конструктор
        return constructor.newInstance(leftX, rightX, values);

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Класс " + functionClass.getName() +  " не имеет конструктора (double, double, double[])", e);
    } catch (InstantiationException e) {
        throw new IllegalArgumentException("Невозможно создать экземпляр класса " + functionClass.getName() + " (абстрактный класс?)", e);
    } catch (IllegalAccessException e) {
        throw new IllegalArgumentException("Конструктор класса " + functionClass.getName() + " недоступен", e);
    } catch (InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка в конструкторе класса " + functionClass.getName(), e);
    }
}
```

```java
// Метод создания функции через рефлексию с массивом точек
public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> functionClass,
        FunctionPoint[] array) {

    try {
        // Получаем конструктор с параметрами (FunctionPoint[])
        Constructor<? extends TabulatedFunction> constructor =
            functionClass.getConstructor(FunctionPoint[].class);

        // Создаем объект через конструктор
        return constructor.newInstance((Object)array); // Приведение типа для varargs

    } catch (NoSuchMethodException e) {
        throw new IllegalArgumentException("Класс " + functionClass.getName() + " не имеет конструктора (FunctionPoint[])", e);
    } catch (InstantiationException e) {
        throw new IllegalArgumentException("Невозможно создать экземпляр класса " +  functionClass.getName() + " (абстрактный класс?)", e);
    } catch (IllegalAccessException e) {
        throw new IllegalArgumentException("Конструктор класса " + functionClass.getName() + " недоступен", e);
    } catch (InvocationTargetException e) {
        throw new IllegalArgumentException("Ошибка в конструкторе класса " + functionClass.getName(), e);
    }
}
```

```java
        System.out.println("\nТест задания 3");

        TabulatedFunction f1;

        System.out.println("=== Тестирование рефлексивного создания ===");

        // 1. ArrayTabulatedFunction через рефлексию (double, double, int)
        f1 = TabulatedFunctions.createTabulatedFunction(functionClass: ArrayTabulatedFunction.class, leftX: 0, rightX: 10, pointsCount: 3);
        System.out.println("1. " + f1.getClass().getSimpleName());
        System.out.println("   " + f1);

        // 2. ArrayTabulatedFunction через рефлексию (double, double, double[])
        f1 = TabulatedFunctions.createTabulatedFunction(functionClass: ArrayTabulatedFunction.class, leftX: 0, rightX: 10, new double[] {0, 5, 10});
        System.out.println("\n2. " + f1.getClass().getSimpleName());
        System.out.println("   " + f1);

        // 3. LinkedListTabulatedFunction через рефлексию (FunctionPoint[])
        f1 = TabulatedFunctions.createTabulatedFunction(
            functionClass: LinkedListTabulatedFunction.class,
            new FunctionPoint[] {
                new FunctionPoint(x: 0, y: 0),
                new FunctionPoint(x: 5, y: 25),
                new FunctionPoint(x: 10, y: 100)
            }
        );
        System.out.println("\n3. " + f1.getClass().getSimpleName());
        System.out.println("   " + f1);

        // 4. tabulate с Sin функцией (использует текущую фабрику)
        f1 = TabulatedFunctions.tabulate(new Sin(), leftX: 0, Math.PI, pointsCount: 11);
        System.out.println("\n4. tabulate с Sin (через фабрику): " + f1.getClass().getSimpleName());
        System.out.println("   " + f1);
    }
```

```
Тест задания 3
=== Тестирование рефлексивного создания ===
1. ArrayTabulatedFunction
   {(0.0;0.0),(5.0;0.0),(10.0;0.0)}

2. ArrayTabulatedFunction
   {(0.0;0.0),(5.0;5.0),(10.0;10.0)}

3. LinkedListTabulatedFunction
   {(0.0;0.0),(5.0;25.0),(10.0;100.0),}

4. tabulate с Sin (через фабрику): ArrayTabulatedFunction
   {(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)}

Тест перегруженных методов с рефлексией
1. Тест tabulate с рефлексией:
Создана функция типа: ArrayTabulatedFunction
Создана функция типа: LinkedListTabulatedFunction

2. Тест inputTabulatedFunction с рефлексией:
Создана функция типа: LinkedListTabulatedFunction

Создана функция типа: LinkedListTabulatedFunction

Создана функция типа: LinkedListTabulatedFunction
Создана функция типа: LinkedListTabulatedFunction
Создана функция типа: LinkedListTabulatedFunction

2. Тест inputTabulatedFunction с рефлексией:
Функция прочитана как: LinkedListTabulatedFunction

3. Тест readTabulatedFunction с рефлексией:
Из текста создана: ArrayTabulatedFunction
```

Рисунки 10-13

Добавил перегруженные методы чтения и tabulate через рефлексию

```java
// Tabulate через рефлексию с указанием класса
    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount, Class<? extends TabulatedFunction> functionClass)
    {
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Заданный интервал выходит за границы области определения функции");
        }

        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

        FunctionPoint[] points = new FunctionPoint[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            double y = function.getFunctionValue(x);
            points[i] = new FunctionPoint(x, y);
        }

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

```java
public static TabulatedFunction inputTabulatedFunction(InputStream in, Class<? extends TabulatedFunction> functionClass) throws IOException {
    DataInputStream dataIn = new DataInputStream(in);
    int pointCount = dataIn.readInt();
    FunctionPoint[] data = new FunctionPoint[pointCount];

    for (int i = 0; i < pointCount; i++) {
        double x = dataIn.readDouble();
        double y = dataIn.readDouble();
        data[i] = new FunctionPoint(x, y);
    }

    // Используем рефлексивный метод для создания функции
    return createTabulatedFunction(functionClass, data);
}

// Добавить этот перегруженный метод readTabulatedFunction с рефлексией
public static TabulatedFunction readTabulatedFunction(Reader in, Class<? extends TabulatedFunction> functionClass) throws IOException {
    StreamTokenizer st = new StreamTokenizer(in);
    st.nextToken();
    int itemsCount = (int) st.nval;
    FunctionPoint[] data = new FunctionPoint[itemsCount];

    for (int i = 0; i < itemsCount; i++) {
        st.nextToken();
        double x = st.nval;
        st.nextToken();
        double y = st.nval;
        data[i] = new FunctionPoint(x, y);
    }

    // Используем рефлексивный метод для создания функции
    return createTabulatedFunction(functionClass, data);
}
```

```java
// Тест 1: tabulate с явным указанием класса
System.out.println("1. Тест tabulate с рефлексией:");
TabulatedFunction tf1 = TabulatedFunctions.tabulate(new Cos(), leftX: 0, Math.PI, pointsCount: 5, functionClass: ArrayTabulatedFunction.class);
System.out.println("Создана функция типа: " + tf1.getClass().getSimpleName());

TabulatedFunction tf2 = TabulatedFunctions.tabulate(new Cos(), leftX: 0, Math.PI, pointsCount: 5, functionClass: LinkedListTabulatedFunction.class);
System.out.println("Создана функция типа: " + tf2.getClass().getSimpleName());

// Тест 2: Чтение из байтового потока с указанием класса
System.out.println("\n2. Тест inputTabulatedFunction с рефлексией:");
try {
    // Создаем тестовую функцию и сериализуем
    TabulatedFunction original = TabulatedFunctions.createTabulatedFunction( functionClass: ArrayTabulatedFunction.class, leftX: 1, rightX: 5, new double[]{1, 4, 9, 16, 25});
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    TabulatedFunctions.outputTabulatedFunction(original, baos);
    byte[] bytes = baos.toByteArray();

    // Читаем как LinkedList функцию
    ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
    TabulatedFunction readAsLinkedList = TabulatedFunctions.inputTabulatedFunction(bais, functionClass: LinkedListTabulatedFunction.class);
    System.out.println("Функция прочитана как: " + readAsLinkedList.getClass().getSimpleName());

} catch (IOException ex)
 {
    ex.printStackTrace();
}

// Тест 3: Чтение из текстового потока с указанием класса
System.out.println("\n3. Тест readTabulatedFunction с рефлексией:");
try {
    String textData = "3\n0.0\n0.0\n2.0\n4.0\n4.0\n16.0\n";
    StringReader reader = new StringReader(textData);

    TabulatedFunction textFunc = TabulatedFunctions.readTabulatedFunction(reader, functionClass: ArrayTabulatedFunction.class);
    System.out.println("Из текста создана: " + textFunc.getClass().getSimpleName());

} catch (IOException ex) {
}
```

Рисунки 14-16