Лабораторная работа 7
по дисциплине «ООП»

Выполнила:
Филатова Анастасия Анатольевна,
студентка группы 6301-030301D,
институт информатики и кибернетики;

Самара 2025

Цель работы :

Цель: Внести изменения в существующий набор типов табулированных функций, позволяющие обрабатывать точки функций по порядку (паттерн «Итератор»), а также выбирать тип объекта табулированной функции при его неявном создании (паттерн «Фабричный метод» и средства рефлексии).

Задание 1

Модификация интерфейса TabulatedFunction:
Интерфейс был расширен для реализации Iterable<FunctionPoint>
Добавлен метод iterator(), возвращающий итератор по точкам функции

```
public interface TabulatedFunction extends Function, java.io.Serializable,
                                           Cloneable, Iterable<FunctionPoint> {
    // ... существующие методы ...

    @Override
    Iterator<FunctionPoint> iterator();
}
```

Реализация итераторов в классах:

В ArrayTabulatedFunction реализован анонимный итератор, работающий с внутренним массивом точек

В LinkedListTabulatedFunction реализован анонимный итератор, работающий со связным списком

Особенности реализации итераторов:

hasNext() - проверяет наличие следующего элемента

next() - возвращает копию следующей точки (сохраняет инкапсуляцию)

remove() - всегда выбрасывает UnsupportedOperationException

Исключения: NoSuchElementException при вызове next() после окончания итерации

Пример реализации в ArrayTabulatedFunction:

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

        @Override
        public boolean hasNext() {
            return currentIndex < pointCount;
        }

        @Override
        public FunctionPoint next() {
            if (!hasNext()) {
                throw new NoSuchElementException("No more elements in the iterator");
            }
            // Возвращаем копию точки, чтобы не нарушить инкапсуляцию
            return (FunctionPoint) points[currentIndex++].clone();
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Remove operation is not supported");
        }
    };
}
```

Задание 2

Создание интерфейса фабрики:

Создан интерфейс TabulatedFunctionFactory с тремя методами создания:

createTabulatedFunction(FunctionPoint[] points)

createTabulatedFunction(double leftX, double rightX, int pointsCount)

createTabulatedFunction(double[] xValues, double[] yValues)

Реализация фабрик в классах:

В ArrayTabulatedFunction добавлен вложенный класс ArrayTabulatedFunctionFactor

В LinkedListTabulatedFunction добавлен вложенный
класс LinkedListTabulatedFunctionFactory

Фабрики создают объекты соответствующих типов

Класс TabulatedFunctions:

Содержит статическое поле для текущей фабрики (по
умолчанию ArrayTabulatedFunctionFactory)

Метод setTabulatedFunctionFactory() для смены фабрики

Методы createTabulatedFunction() для создания объектов через фабрику

Метод tabulate() для табулирования функций с использованием фабрики

```
// Установка фабрики для связного списка
TabulatedFunctions.setTabulatedFunctionFactory(
    new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());

// Создание функции через фабрику
TabulatedFunction function = TabulatedFunctions.tabulate(
    new CosFunction(), 0, Math.PI, 11);
```

Задание 3

Методы создания через рефлексию:

В TabulatedFunctions добавлены три перегруженных метода с параметром Class

Методы проверяют, что переданный класс реализует TabulatedFunction

Используется рефлексия для поиска и вызова конструкторов

Обработка исключений:

Исключения рефлексии (NoSuchMethodException, InstantiationException и др.)

Преобразуются в IllegalArgumentException с сохранением исходной причины

```
public static TabulatedFunction createTabulatedFunction(
        Class<? extends TabulatedFunction> functionClass,
        double leftX, double rightX, int pointsCount) {
    try {
        Constructor<? extends TabulatedFunction> constructor =
            functionClass.getConstructor(double.class, double.class, int.class);
        return constructor.newInstance(leftX, rightX, pointsCount);
    } catch (Exception e) {
        throw new IllegalArgumentException("Failed to create function: " + e.getMessage(), e);
    }
}
```

Вывод:

=== Laboratory Work #7 ===

Iterators and Factory Methods

1. Testing iterators:

a) ArrayTabulatedFunction iteration:

Points: (0.0; 0.0) (1.0; 1.0) (2.0; 4.0) (3.0; 9.0) (4.0; 16.0)

b) LinkedListTabulatedFunction iteration:

Points: (0.0; 0.0) (1.0; 1.0) (2.0; 4.0) (3.0; 9.0) (4.0; 16.0)

c) Testing iterator exceptions:

OK: NoSuchElementException caught: No more elements in the iterator

OK: UnsupportedOperationException caught: Remove operation is not supported

2. Testing factory methods:

a) Default factory (ArrayTabulatedFunction):

Class: ArrayTabulatedFunction

First 3 points:

(0.0; 1.0)

(0.3141592653589793; 0.9510565162951535)

(0.6283185307179586; 0.8090169943749475)

b) Switching to LinkedListTabulatedFunctionFactory:

Class: LinkedListTabulatedFunction

First 3 points:

(0.0; 1.0)

(0.3141592653589793; 0.9510565162951535)

(0.6283185307179586; 0.8090169943749475)

c) Switching back to ArrayTabulatedFunctionFactory:

Class: ArrayTabulatedFunction

First 3 points:

(0.0; 1.0)

(0.3141592653589793; 0.9510565162951535)

(0.6283185307179586; 0.8090169943749475)

3. Testing reflection:

a) Creating ArrayTabulatedFunction via reflection:

Class: ArrayTabulatedFunction

Function: {(0.0; 0.0), (5.0; -0.9589242746631385), (10.0; -0.5440211108893698)}

b) Creating ArrayTabulatedFunction via reflection (arrays):

Class: ArrayTabulatedFunction

Function: {(0.0; 0.0), (5.0; -0.9589242746631385), (10.0; -0.5440211108893698)}

c) Creating LinkedListTabulatedFunction via reflection (points):

Class: LinkedListTabulatedFunction

Function: {(0.0; 0.0), (10.0; 10.0)}

d) Tabulating via reflection:

Class: LinkedListTabulatedFunction

First 5 points:

(0.0; 0.0)

(0.3141592653589793; 0.3090169943749474)

(0.6283185307179586; 0.5877852522924731)

(0.9424777960769379; 0.8090169943749475)

(1.2566370614359172; 0.9510565162951535)

e) Testing error handling:

OK: Correct parameters handled.

OK: Correct numeric parameters handled.

OK: IllegalArgumentException caught for pointCount=1: Failed to create function: null