Лабораторная работа №7

Паттерны проектирования: Итератор и Фабричный метод

Выполнил: Пожидаев Богдан

Группа: 6204-010302D

1. Задание 1 – Реализация паттерна «Итератор»

1.1. Ход выполнения

Были внесены следующие изменения в код:

1. **Интерфейс TabulatedFunction** расширен для поддержки итерации:

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

2. **Класс ArrayTabulatedFunction** получил реализацию итератора:

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

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

      @Override
      public FunctionPoint next() {
         if (!hasNext()) throw new NoSuchElementException();
         return new FunctionPoint(points[currentIndex++]);
      }
   };
}
```

3. **Класс LinkedListTabulatedFunction** также реализовал итератор:

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

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

        @Override
        public FunctionPoint next() {
            if (!hasNext()) throw new NoSuchElementException();
            FunctionPoint point = new FunctionPoint(currentNode.point);
            currentNode = currentNode.next;
            return point;
        }
    };
}
```

2.2. Результат

Теперь объекты табулированных функций можно использовать в цикле for-each:

```
TabulatedFunction func = new ArrayTabulatedFunction(0, 10, new double[]{0,1,4,9,16});
for (FunctionPoint point : func) {
    System.out.println(point);
}
```

Вывод программы:

```
text
(0.0; 0.0)
(2.5; 1.0)
(5.0; 4.0)
(7.5; 9.0)
(10.0; 16.0)
```

3. Задание 2 – Реализация паттерна «Фабричный метод»

3.1. Ход выполнения

1. **Создан интерфейс фабрики:**

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

2. **Реализованы фабрики для каждого класса:**

```
// В ArrayTabulatedFunction
public static class ArrayTabulatedFunctionFactory implements TabulatedFunctionFactory {
   @Override
   public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int
pointsCount) {
      return new ArrayTabulatedFunction(leftX, rightX, pointsCount);
   }
}
```

```
// В LinkedListTabulatedFunction
public static class LinkedListTabulatedFunctionFactory implements TabulatedFunctionFactory {
   @Override
   public TabulatedFunction createTabulatedFunction(double leftX, double rightX, int
pointsCount) {
      return new LinkedListTabulatedFunction(leftX, rightX, pointsCount);
   }
}
```

3. **Обновлен класс TabulatedFunctions:**

```
public class TabulatedFunctions {
   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);
   }
}
```

3.2. Результат

Тестирование фабрик:

```
Function cos = new Cos();
TabulatedFunction tf;

// Фабрика по умолчанию (Array)
tf = TabulatedFunctions.tabulate(cos, 0, Math.PI, 11);
System.out.println("Default: " + tf.getClass().getSimpleName());

// LinkedList фабрика
TabulatedFunctions.setTabulatedFunctionFactory(
   new LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(cos, 0, Math.PI, 11);
```

```
System.out.println("LinkedList: " + tf.getClass().getSimpleName());


// Array фабрика
TabulatedFunctions.setTabulatedFunctionFactory(
    new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(cos, 0, Math.PI, 11);
System.out.println("Array: " + tf.getClass().getSimpleName());
```

Вывод программы:

```
Default: ArrayTabulatedFunction
LinkedList: LinkedListTabulatedFunction
Array: ArrayTabulatedFunction
```

4. Задание 3 – Использование рефлексии

4.1. Ход выполнения

Добавлены методы с рефлексией в класс TabulatedFunctions:

```
public static TabulatedFunction createTabulatedFunction(Class<?> functionClass,
                                    double leftX, double rightX, int pointsCount) {
    try {
        if (!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Class must implement TabulatedFunction");
        }

        Constructor<?> constructor = functionClass.getConstructor(
            double.class, double.class, int.class);
        return (TabulatedFunction) constructor.newInstance(leftX, rightX, pointsCount);

    } catch (Exception e) {
        throw new IllegalArgumentException("Error creating function", e);
    }
```

```
}
```

4.2. Результат

Тестирование рефлексии:

```
TabulatedFunction f;

f = TabulatedFunctions.createTabulatedFunction(
   ArrayTabulatedFunction.class, 0, 10, 3);
System.out.println("Array via reflection: " + f.getClass().getSimpleName());

f = TabulatedFunctions.createTabulatedFunction(
   LinkedListTabulatedFunction.class, 0, 10, new double[]{0, 5, 10});
System.out.println("LinkedList via reflection: " + f.getClass().getSimpleName());

f = TabulatedFunctions.tabulate(
   LinkedListTabulatedFunction.class, new Sin(), 0, Math.PI, 11);
System.out.println("Tabulate with reflection: " + f.getClass().getSimpleName());
```

Вывод программы:

```
Array via reflection: ArrayTabulatedFunction
LinkedList via reflection: LinkedListTabulatedFunction
Tabulate with reflection: LinkedListTabulatedFunction
```

- **Фабричный метод** позволил гибко управлять созданием объектов
- **Рефлексия** дала возможность динамически выбирать типы создаваемых объектов

Полученные навыки позволяют создавать более гибкие и расширяемые программные системы, соответствующие принципам SOLID.

=== Тестирование итератора ===

ArrayTabulatedFunction points:

(0.0; 0.0)

(2.5; 1.0)

(5.0; 4.0)

(7.5; 9.0)

(10.0; 16.0)

LinkedListTabulatedFunction points:

(0.0; 0.0)

(2.5; 1.0)

(5.0; 4.0)

(7.5; 9.0)

(10.0; 16.0)

=== Тестирование фабричного метода ===

Default factory: ArrayTabulatedFunction

LinkedList factory: LinkedListTabulatedFunction

Array factory: ArrayTabulatedFunction

=== Тестирование рефлексии ===

Array via reflection: ArrayTabulatedFunction

{(0.0; 0.0), (5.0; 0.0), (10.0; 0.0)}

LinkedList via reflection: LinkedListTabulatedFunction

{(0.0; 0.0), (5.0; 5.0), (10.0; 10.0)}

Tabulate with reflection: LinkedListTabulatedFunction

{(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)}