Отчет по лабораторной работе №7

Паттерны проектирования: Итератор, Фабричный метод и рефлексия

Выполнила: Андреяшкина Мария
Группа: 6204-010302D

Оглавление

Задание 1 2
Задание 2 6
Задание 3 10
Выводы 15

Задание 1

Итератор

Цель

Сделать объекты TabulatedFunction итерируемыми для использования в for-each цикле.

Реализация

1.1 Изменения в TabulatedFunction.java

```
public interface TabulatedFunction extends Function,
Iterable<FunctionPoint>, Cloneable {
    // ... существующие методы остаются без изменений
}
```

1.2 Итератор в ArrayTabulatedFunction.java

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

        @Override
        public void remove() {
```

```
            throw new UnsupportedOperationException("Удаление не
поддерживается");
        }
    };
}
```

1.3 Итератор в LinkedListTabulatedFunction.java

```
@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("Нет следующего
элемента");
            }
            FunctionPoint point = new
FunctionPoint(currentNode.point);
            currentNode = currentNode.next;
            return point;
        }

        @Override
        public void remove() {
```

```
            throw new UnsupportedOperationException("Удаление не
поддерживается");
        }
    };
}
```

Тестирование в Main.java

```
System.out.println("1. Тестирование итераторов:");

TabulatedFunction arrayFunc = new ArrayTabulatedFunction(0, 10, 4);
System.out.println("Массивная функция (4 точки от 0 до 10):");
for (FunctionPoint p : arrayFunc) {
    System.out.println("  Точка: " + p);
}

TabulatedFunction listFunc = new LinkedListTabulatedFunction(0, 10,
4);
System.out.println("\nСписковая функция (4 точки от 0 до 10):");
for (FunctionPoint p : listFunc) {
    System.out.println("  Точка: " + p);
}
```

Вывод программы

```
1. Тестирование итераторов:

Массивная функция (4 точки от 0 до 10):
  Точка: (0.0; 0.0)
  Точка: (3.3333333333333335; 0.0)
  Точка: (6.666666666666667; 0.0)
  Точка: (10.0; 0.0)

Списковая функция (4 точки от 0 до 10):
```

```
  Точка: (0.0; 0.0)
  Точка: (3.3333333333333335; 0.0)
  Точка: (6.666666666666667; 0.0)
  Точка: (10.0; 0.0)
```

Задание 2

Фабричный

Цель

Реализовать динамический выбор типа создаваемых табулированных функций.

Реализация

2.1 Создание TabulatedFunctionFactory.java

```
package functions;


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

2.2 Фабрика в ArrayTabulatedFunction.java

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

2.3 Фабрика в LinkedListTabulatedFunction.java

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

2.4 Изменения в TabulatedFunctions.java

```
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);
}


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


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

Тестирование в Main.java

```
System.out.println("\n2. Тестирование фабричного метода:");
```

```
Function sin = new Sin();
TabulatedFunction tf;


System.out.println("\nИзначальная фабрика
(ArrayTabulatedFunction):");
tf = TabulatedFunctions.tabulate(sin, 0, Math.PI, 4);
System.out.println("  Создана функция типа: " +
tf.getClass().getSimpleName());


System.out.println("\nМеняем фабрику на
LinkedListTabulatedFunction:");
TabulatedFunctions.setTabulatedFunctionFactory(
    new
LinkedListTabulatedFunction.LinkedListTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(sin, 0, Math.PI, 4);
System.out.println("  Теперь функция типа: " +
tf.getClass().getSimpleName());


System.out.println("\nВозвращаем фабрику обратно:");
TabulatedFunctions.setTabulatedFunctionFactory(
    new ArrayTabulatedFunction.ArrayTabulatedFunctionFactory());
tf = TabulatedFunctions.tabulate(sin, 0, Math.PI, 4);
System.out.println("  Функция типа: " +
tf.getClass().getSimpleName());
```

Вывод программы

```
2. Тестирование фабричного метода:


Изначальная фабрика (ArrayTabulatedFunction):
  Создана функция типа: ArrayTabulatedFunction


Меняем фабрику на LinkedListTabulatedFunction:
  Теперь функция типа: LinkedListTabulatedFunction
```

```
Возвращаем фабрику обратно:
  Функция типа: ArrayTabulatedFunction
```

Задание 3

Рефлексия

Цель

Реализовать создание объектов через рефлексию по имени класса.

Реализация

3.1 Методы с рефлексией в TabulatedFunctions.java

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

        Constructor<?> constructor = functionClass.getConstructor(
            double.class, double.class, int.class);
        return (TabulatedFunction) constructor.newInstance(leftX,
rightX, pointsCount);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании
функции", e);
    }
}
```

```
public static TabulatedFunction createTabulatedFunction(Class<?>
functionClass,
        double leftX, double rightX, double[] values) {
    try {
        if
(!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен
реализовывать TabulatedFunction");
        }

        Constructor<?> constructor = functionClass.getConstructor(
            double.class, double.class, double[].class);
        return (TabulatedFunction) constructor.newInstance(leftX,
rightX, values);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании
функции", e);
    }
}

public static TabulatedFunction createTabulatedFunction(Class<?>
functionClass,
        FunctionPoint[] points) {
    try {
        if
(!TabulatedFunction.class.isAssignableFrom(functionClass)) {
            throw new IllegalArgumentException("Класс должен
реализовывать TabulatedFunction");
        }

        Constructor<?> constructor =
functionClass.getConstructor(FunctionPoint[].class);
```

```
        return (TabulatedFunction)
constructor.newInstance((Object)points);
    } catch (Exception e) {
        throw new IllegalArgumentException("Ошибка при создании
функции", e);
    }
}

public static TabulatedFunction tabulate(Class<?> functionClass,
        Function function, double leftX, double rightX, int
pointsCount) {
    double[] values = new double[pointsCount];
    double step = (rightX - leftX) / (pointsCount - 1);

    for (int i = 0; i < pointsCount; i++) {
        double x = leftX + i * step;
        if (i == pointsCount - 1) {
            x = rightX;
        }
        values[i] = function.getFunctionValue(x);
    }

    return createTabulatedFunction(functionClass, leftX, rightX,
values);
}
```

Тестирование в Main.java

```
System.out.println("\n3. Тестирование рефлексии:");

System.out.println("\na) Создание через рефлексию
(ArrayTabulatedFunction):");
TabulatedFunction f1 = TabulatedFunctions.createTabulatedFunction(
```

```
    ArrayTabulatedFunction.class, 0, 5, 3);
System.out.println("  Создана: " + f1.getClass().getSimpleName());
System.out.println("  Содержимое: " + f1);

System.out.println("\nb) Создание через рефлексию
(LinkedListTabulatedFunction):");
TabulatedFunction f2 = TabulatedFunctions.createTabulatedFunction(
    LinkedListTabulatedFunction.class, 0, 5, new double[] {0, 2.5,
5});
System.out.println("  Создана: " + f2.getClass().getSimpleName());
System.out.println("  Содержимое: " + f2);

System.out.println("\nc) Табулирование через рефлексию:");
TabulatedFunction f3 = TabulatedFunctions.tabulate(
    LinkedListTabulatedFunction.class, new Cos(), 0, Math.PI, 3);
System.out.println("  Тип: " + f3.getClass().getSimpleName());
System.out.println("  Значения Cos от 0 до PI:");
for (FunctionPoint p : f3) {
    System.out.println("    x = " + String.format("%.2f", p.getX())
+
                       ", y = " + String.format("%.4f", p.getY()));
}
```

Вывод программы

```
3. Тестирование рефлексии:

a) Создание через рефлексию (ArrayTabulatedFunction):
  Создана: ArrayTabulatedFunction
  Содержимое: {(0.0; 0.0), (2.5; 0.0), (5.0; 0.0)}

b) Создание через рефлексию (LinkedListTabulatedFunction):
  Создана: LinkedListTabulatedFunction
```

```
  Содержимое: {(0.0; 0.0), (2.5; 2.5), (5.0; 5.0)}

c) Табулирование через рефлексию:
  Тип: LinkedListTabulatedFunction
  Значения Cos от 0 до PI:
    x = 0.00, y = 1.0000
    x = 1.57, y = 0.0000
    x = 3.14, y = -1.0000
```

Итоговый вывод программы

```
=== Лабораторная работа №7 ===

1. Тестирование итераторов:

Массивная функция (4 точки от 0 до 10):
  Точка: (0.0; 0.0)
  Точка: (3.3333333333333335; 0.0)
  Точка: (6.666666666666667; 0.0)
  Точка: (10.0; 0.0)

Списковая функция (4 точки от 0 до 10):
  Точка: (0.0; 0.0)
  Точка: (3.3333333333333335; 0.0)
  Точка: (6.666666666666667; 0.0)
  Точка: (10.0; 0.0)



2. Тестирование фабричного метода:

Изначальная фабрика (ArrayTabulatedFunction):
```

```
  Создана функция типа: ArrayTabulatedFunction

Меняем фабрику на LinkedListTabulatedFunction:
  Теперь функция типа: LinkedListTabulatedFunction

Возвращаем фабрику обратно:
  Функция типа: ArrayTabulatedFunction



3. Тестирование рефлексии:

a) Создание через рефлексию (ArrayTabulatedFunction):
  Создана: ArrayTabulatedFunction
  Содержимое: {(0.0; 0.0), (2.5; 0.0), (5.0; 0.0)}

b) Создание через рефлексию (LinkedListTabulatedFunction):
  Создана: LinkedListTabulatedFunction
  Содержимое: {(0.0; 0.0), (2.5; 2.5), (5.0; 5.0)}

c) Табулирование через рефлексию:
  Тип: LinkedListTabulatedFunction
  Значения Cos от 0 до PI:
    x = 0,00, y = 1,0000
    x = 1,57, y = 0,0000
    x = 3,14, y = -1,0000
```

Выводы

1. **Задание 1 выполнено:** Итераторы работают для обеих реализаций TabulatedFunction
2. **Задание 2 выполнено:** Фабричный метод позволяет динамически менять тип создаваемых функций

3. **Задание 3 выполнено:** Рефлексия корректно создает объекты по имени класса

Все паттерны реализованы согласно требованиям задания и работают корректно.