14. Паттерны проектирования, поведенческие паттерны

Теоретическое введение

Паттерны проектирования — это повторяющиеся решения типичных задач проектирования программного обеспечения. Они помогают сделать код более гибким, поддерживаемым и повторно используемым. Порождающие паттерны проектирования играют важную роль в управлении процессом создания объектов, предоставляя гибкие способы инкапсуляции создания объектов и обеспечения их корректного использования.

Поведенческие паттерны (Creational Patterns) — Поведенческие паттерны проектирования описывают способы взаимодействия объектов и классов друг с другом, фокусируясь на алгоритмах и распределении обязанностей. Эти паттерны помогают обеспечить эффективное и гибкое взаимодействие между компонентами системы, упрощая коммуникацию и управление поведением объектов.

Основные задачи поведенческих паттернов:

1. Управление взаимодействием объектов: Обеспечение гибкости и надёжности при взаимодействии между объектами. Поведенческие паттерны определяют, как объекты сотрудничают и передают данные друг другу.
2. Инкапсуляция алгоритмов: позволяют отделить алгоритмы от классов, использующих их, что упрощает их замену или изменение без влияния на остальную систему.
3. Разделение обязанностей: чётко определяют роли объектов и классов в процессе выполнения операции, улучшая читаемость и поддержку кода.

Основные поведенческие паттерны

1. Chain of Responsibility (Цепочка обязанностей)

Цель: позволяет передавать запрос последовательно по цепочке обработчиков. Каждый обработчик может обработать запрос или передать его следующему.

Применение: Обработка событий или запросов в веб-приложениях или логгер, например с уровнями INFO, BEGUG, ERROR.

Пример реализации приведен на Листинге 14.1.

Листинг 14.1 – Пример реализации паттерна Chain on Responbillity

```
abstract class Logger {
    protected Logger nextLogger;

    public void setNextLogger(Logger nextLogger) {
        this.nextLogger = nextLogger;
    }

    public void logMessage(String message) {
        if (nextLogger != null) {
            nextLogger.logMessage(message);
        }
    }
}

class ConsoleLogger extends Logger {
    public void logMessage(String message) {
        System.out.println("Console Logger: " + message);
    }
}
```

2. Command (Команда)

Цель: инкапсулирует запрос как объект, позволяя параметризовать клиентов различными запросами.

Применение: Реализация отмены действий (Undo/Redo), выполнение очереди команд.

Пример реализации приведен на Листинге 14.2.

Листинг 14.2 – Пример реализации паттерна Command

```
interface Command { void execute(); }


class Light {
    public void on() { System.out.println("Light is ON"); }
}


class LightOnCommand implements Command {
    private Light light;
    public LightOnCommand(Light light) { this.light = light; }
    public void execute() { light.on(); }
}
```

3. Iterator (Итератор)

Цель: обеспечивает последовательный доступ к элементам коллекции без раскрытия её внутренней структуры.

Применение: Перебор элементов коллекции (списки, массивы).

Пример реализации приведен на Листинге 14.3.

Листинг 14.3 – Пример реализации паттерна Iterator

```
class NameRepository {
    private String[] names = {"Alice", "Bob"};

    public Iterator<String> getIterator() {
        return new ArrayIterator();
    }

    private class ArrayIterator implements Iterator<String> {
        int index = 0;
        public boolean hasNext() { return index < names.length; }
        public String next() { return names[index++]; }
    }
}
```

4. Mediator (Посредник)

Цель: Обеспечивает централизованное взаимодействие между объектами, уменьшает их связанность.

Применение: Управление элементами интерфейса или модулями системы.

Пример реализации приведен на Листинге 14.4.

Листинг 14.4 – Пример реализации паттерна Mediator

```
class ChatRoom {
    public static void showMessage(String user, String message) {
        System.out.println(user + ": " + message);
    }
}
```

5. Memento (Снимок)

Цель: позволяет сохранять и восстанавливать состояние объекта, не раскрывая его внутреннего устройства.

Применение: Реализация функции сохранения и восстановления, например, текстовый редактор.

Пример реализации приведен на Листинге 14.5.

Листинг 14.5 – Пример реализации паттерна Momento

```
class TextEditor {
    private String text;
    public void setText(String text) { this.text = text; }
    public String save() { return text; }
}
```

6. Observer (Наблюдатель)

Цель: описывает зависимость «один ко многим», где изменение одного объекта обновляет все зависимые.

Применение: Реализация подписки на события, например, уведомления.

Пример реализации приведен на Листинге 14.6.

Листинг 14.6 – Пример реализации паттерна Observer

```
interface Observer { void update(); }
class User implements Observer {
    public void update() { System.out.println("User notified"); }
}
```

7. State (Состояние)

Цель: позволяет объекту изменять своё поведение при изменении состояния.

Применение: Управление состояниями объектов, например, автомат состояний заказов.

Пример реализации приведен на Листинге 14.7.

Листинг 14.7 – Пример реализации паттерна State

```
interface State { void handle(); }
class NewOrderState implements State {
    public void handle() { System.out.println("Processing new order"); }
}
```

8. Strategy (Стратегия)

Цель: определяет семейство алгоритмов и делает их взаимозаменяемыми.

Применение: Выбор алгоритма на основе условий, например, разные способы сортировки.

Пример реализации приведен на Листинге 14.8.

Листинг 14.8 – Пример реализации паттерна Strategy

```
interface DiscountStrategy { double applyDiscount(double price); }
class NoDiscount implements DiscountStrategy {
    public double applyDiscount(double price) { return price; }
}
```

9. Template Method (Шаблонный метод)

Цель: определяет основу алгоритма, позволяя подклассам переопределять шаги.

Применение: Создание алгоритмов с изменяемыми шагами, например, обработка данных.

Пример реализации приведен на Листинге 14.9.

Листинг 14.9 – Пример реализации паттерна Template Method

```
abstract class Beverage {
    final void prepare() {
        boilWater();
        brew();
    }
    abstract void brew();
    void boilWater() { System.out.println("Boiling water"); }
}
```

10. Visitor (Посетитель)

Цель: добавляет новые операции для объектов структуры без изменения их классов.

Применение: Выполнение операций над элементами сложной структуры, например, обработка элементов структуры документа.

Пример реализации приведен на Листинге 14.10.

Листинг 14.10 – Пример реализации паттерна Visitor

```
interface Element { void accept(Visitor visitor); }


class Paragraph implements Element {
    public void accept(Visitor visitor) { visitor.visit(this); }
}
```

Практическое задание

В данной практической работе представлено 10 вариантов заданий. Выбор варианта осуществляется в соответствии с порядковым номером студента в списке. Если порядковый номер студента равен 1, выполняется задание варианта №1; если порядковый номер равен 10 — выполняется вариант №10. В случае, если порядковый номер превышает количество вариантов (например, 11 или более), задание выбирается циклически, начиная с варианта №1.

Вариант №1

Chain of Responsibillity – Задание

Реализовать систему обработки запросов разного уровня важности (например, логирование или обработка заявок). Каждый запрос должен обрабатываться соответствующим обработчиком.

Требования:

Создать обработчики для уровней сообщений (INFO, WARNING, ERROR, CRITICAL, DEBUG).

Реализовать возможность передачи запроса по цепочке обработчиков.

Продемонстрировать обработку запроса разного уровня.

Вариант №2

Command – Задание

Разработать систему управления бытовыми приборами (например, телевизор или кондиционер). Команды включения и выключения должны быть инкапсулированы в объекты.

Требования:

Создать интерфейс Command с методом execute().

Реализовать команды TurnOnCommand и TurnOffCommand.

Реализовать класс RemoteControl, который вызывает команды.

Вариант №3

Iterator – Задание

Создать систему для перебора элементов коллекции (например, список студентов). Итератор должен предоставлять доступ к элементам без раскрытия структуры.

Требования:

Создать интерфейс Iterator с методами hasNext() и next().

Реализовать класс коллекции StudentList.

Реализовать итератор для перебора студентов.

Вариант №4

Mediator – Задание

Реализовать систему чата, где несколько пользователей могут обмениваться сообщениями через посредника.

Требования:

Создать интерфейс Mediator для координации сообщений.

Реализовать классы User, которые будут отправлять и получать сообщения через посредника.

Продемонстрировать обмен сообщениями между пользователями.

Вариант №5

Memento – Задание

Создать текстовый редактор с возможностью сохранения и восстановления состояния текста.

Требования:

Реализовать класс TextEditor для редактирования текста.

Реализовать создание снимка состояния (Memento).

Добавить возможность восстановления состояния редактора.

Вариант №6

Observer – Задание

Реализовать систему уведомлений для пользователей. Когда происходит событие (например, изменение состояния объекта), все подписчики должны получать уведомление.

Требования:

Создать интерфейс Observer и класс Subject для управления подписчиками.

Реализовать класс User, который будет получать уведомления.

Проверить отправку уведомлений подписчикам при изменении состояния.

Вариант №7

State – Задание

Создать систему управления состоянием заказа в интернет-магазине (например, New, Processing, Shipped).

Требования:

Реализовать интерфейс State с методом handle().

Создать классы для каждого состояния заказа.

Продемонстрировать переходы между состояниями.

Вариант №8

Strategy – Задание

Реализовать систему расчёта стоимости доставки. Пользователь может выбрать разные стратегии доставки (например, стандартная, экспресс, курьерская).

Требования:

Создать интерфейс DeliveryStrategy с методом calculateCost().

Реализовать несколько стратегий доставки.

Продемонстрировать выбор стратегии и расчёт стоимости.

Вариант №9

Template Method – Задание

Создать систему для приготовления напитков (например, чай и кофе). Шаблонный метод должен определять последовательность шагов, а подклассы — реализовать конкретные шаги.

Требования:

Создать абстрактный класс Beverage с шаблонным методом prepare().

Реализовать классы Tea и Coffee.

Продемонстрировать приготовление напитков с использованием шаблонного метода.

Вариант №10

Visitor – Задание

Реализовать систему для обработки элементов структуры (например, элементы документа). Каждая операция обработки должна быть вынесена в отдельный класс.

Требования:

Создать интерфейс Visitor с методами для обработки различных элементов.

Реализовать классы элементов (Paragraph, Image).

Продемонстрировать выполнение операций над элементами через посетителя.