Java Design Patterns

Chain of responsibility

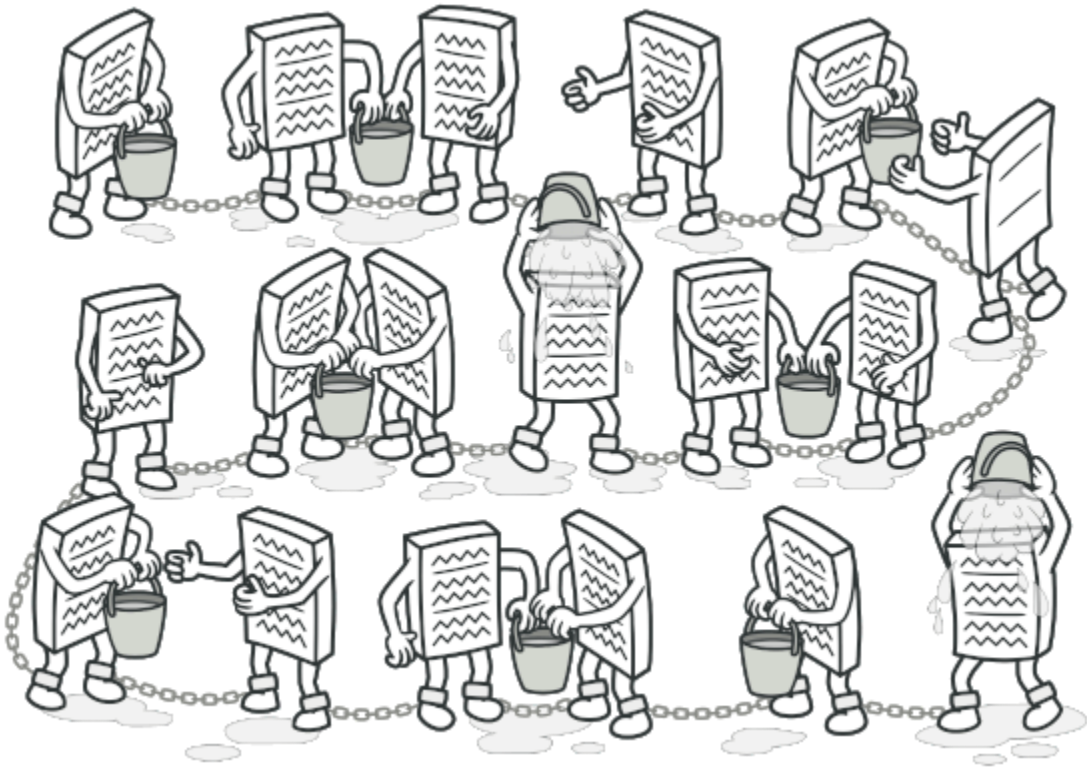
Тема

Chain of responsibility

Суть паттерна

Цепочка обязанностей

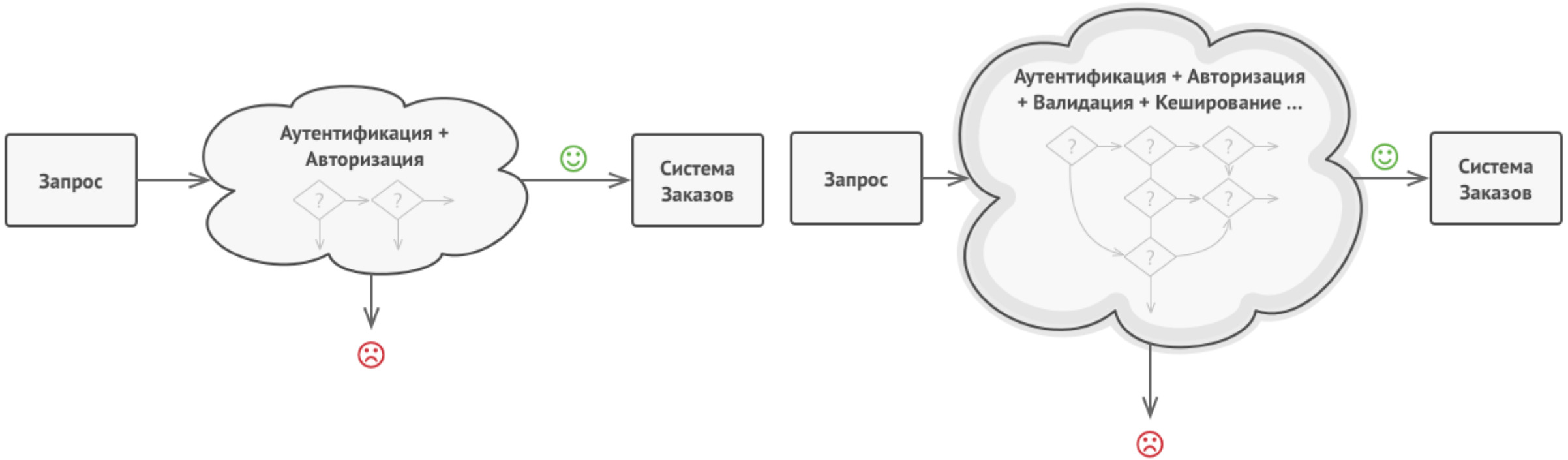
Цепочка обязанностей — это поведенческий паттерн проектирования, который позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи.



Проблема

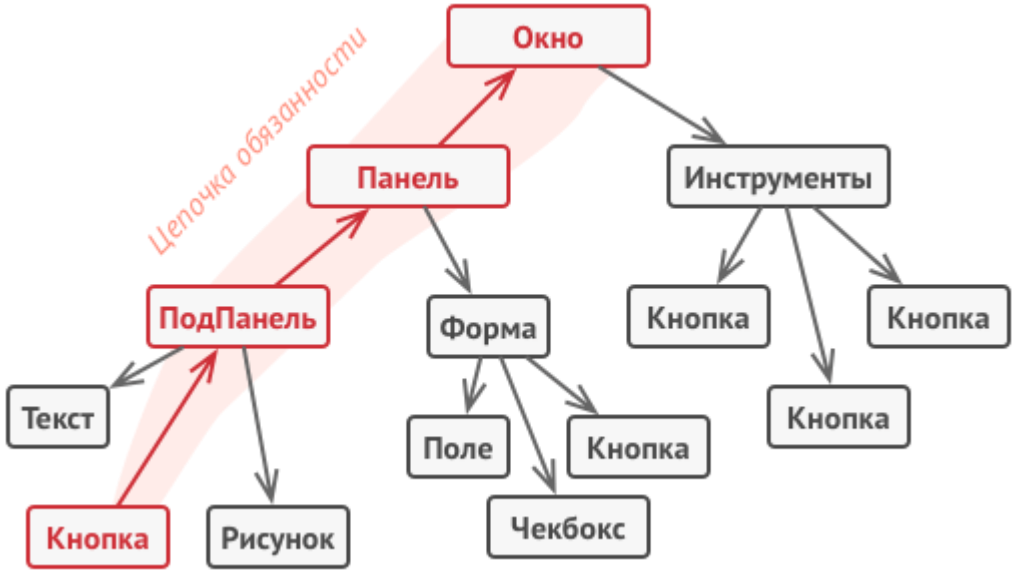
Постановка задачи

Представьте, что вы делаете систему приёма онлайн заказов. Вы хотите ограничить к ней доступ так, чтобы только авторизованные пользователи могли создавать заказы. Кроме того, определённые пользователи, владеющие правами администратора, должны иметь полный доступ к заказам.



Решение задачи

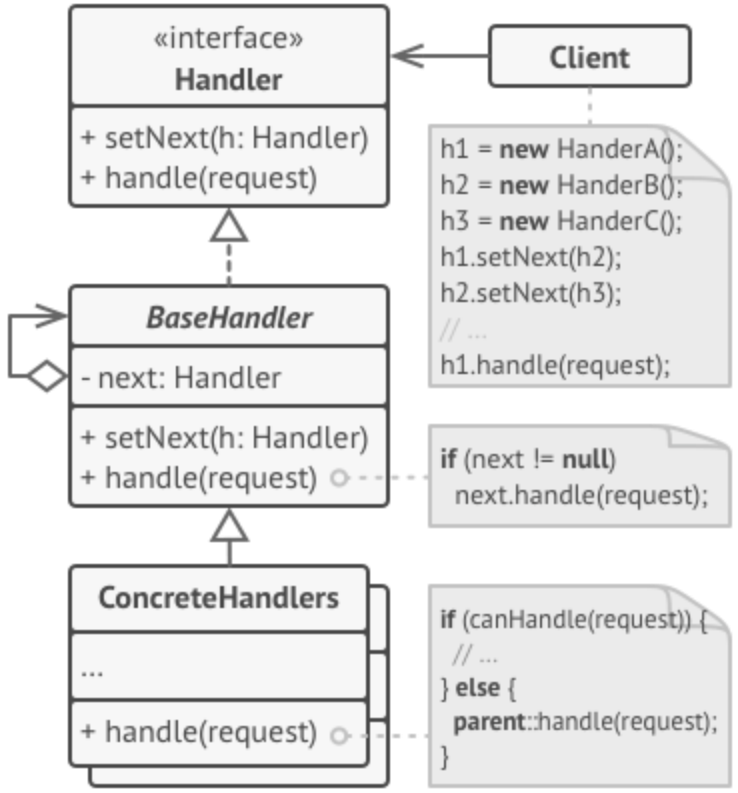
Как и многие другие поведенческие паттерны, Цепочка обязанностей базируется на том, чтобы превратить отдельные поведения в объекты. В нашем случае, каждая проверка переедет в отдельный класс с единственным методом выполнения. Данные запроса, над которым происходит проверка, будут передаваться в метод как аргументы.



Структура

Структура паттерна

1. Обработчик определяет общий для всех конкретных обработчиков интерфейс. Обычно, достаточно описать единственный метод обработки запросов, но иногда здесь может быть определён и метод выставления следующего обработчика.
2. Базовый обработчик — опциональный класс, который позволяет избавиться от дублирования одного и того же кода во всех конкретных обработчиках.
3. Конкретные обработчики содержат код обработки запросов. При получении запроса каждый обработчик решает, может ли он обработать запрос или нет, а также стоит ли передать его следующему объекту.
4. Клиент составляет цепочки обработчиков один раз или динамически, в зависимости от логики программы. Клиент может отправить запрос любому из объектов цепочки, причём это не всегда первый объект в цепочке.



Применимость

Применение паттерна

1. Когда программа содержит несколько объектов, способных обработать тот или иной запрос, однако заранее неизвестно какой запрос придёт и какой обработчик понадобится.
2. Когда важно, чтобы обработчики выполнялись один за другим в строгом порядке.
3. Когда набор объектов, способных обработать запрос, должен задаваться динамически.

Шаги реализации

Алгоритм реализации паттерна

1. Создайте интерфейс обработчика и опишите в нём основной метод обработки.
2. Имеет смысл создать абстрактный базовый класс обработчиков, чтобы не дублировать реализацию метода получения следующего обработчика во всех конкретных обработчиках.
3. Один за другим создайте классы конкретных обработчиков и реализуйте в них методы обработки запросов. При получении запроса каждый обработчик должен решить:
   - Может он обработать запрос или нет?
   - Следует передать запрос следующему обработчику или нет?
4. Клиент может собирать цепочку обработчиков самостоятельно, опираясь на свою бизнес-логику, либо получать уже готовые цепочки извне. В последнем случае, цепочки собирают фабричные объекты исходя из конфигурации приложения или текущего окружения.
5. Клиент может посылать запросы любому обработчику в цепи, а не только первому. Запрос будет передаваться по цепочке пока какой-то обработчик не откажется передавать его дальше, либо когда будет достигнут конец цепи.
6. Клиент должен знать о динамической природе цепочки и быть готов к таким случаям:
   - Цепочка может состоять из единственного объекта.
   - Запросы могут не достигать конца цепи.
   - Запросы могут достигать конца, оставаясь необработанными.

Плюсы и недостатки

Плюсы:

- Уменьшает зависимость между клиентом и обработчиками.
- Реализует *принцип единственной обязанности*.
- Реализует *принцип открытости/закрытости*.

Минусы:

- Запрос может остаться никем не обработанным.

Отношение с другими паттернами

- Цепочка обязанностей, Команда, Посредник и Наблюдатель показывают различные способы работы отправителей запросов с их получателями.
- Цепочку обязанностей часто используют вместе с Компоновщиком. В этом случае, запрос передаётся от дочерних компонентов к их родителям.
- Обработчики в Цепочке обязанностей могут быть выполнены в виде Команд. В этом случае множество разных операций может быть выполнено над одним и тем же контекстом, коим является запрос.
- Цепочка обязанностей и Декоратор имеют очень похожие структуры. Оба паттерна базируются на принципе рекурсивного выполнения операции через серию связанных объектов. Но есть и несколько важных отличий.

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association