Chain of Responsibility

- Cristian Jhair Mejía Navarro
- Jaider Luis De La Rosa Castro

Patrones de diseño de software
Especialización en Ingeniería de Software
Universidad Popular del Cesar

Chain of Responsibility

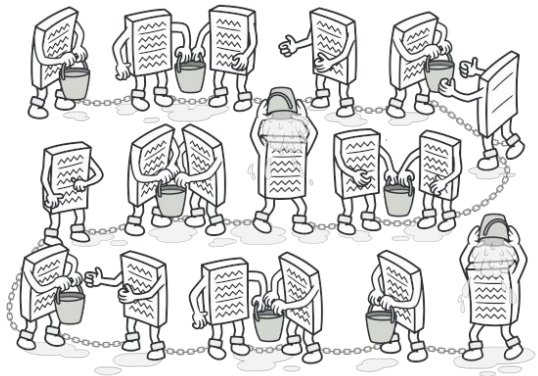
Propósito

¿Qué es?

Chain of Responsibility es un patrón de diseño de comportamiento que te permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

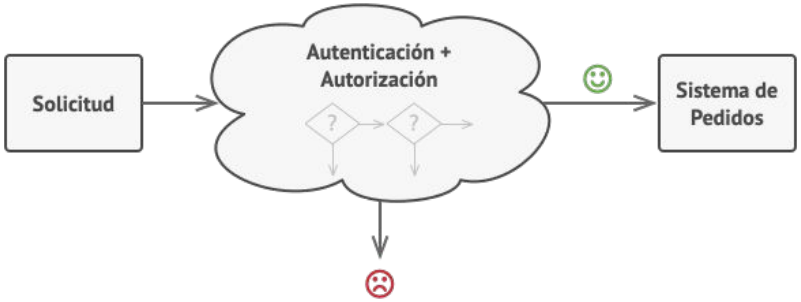
Utilidad

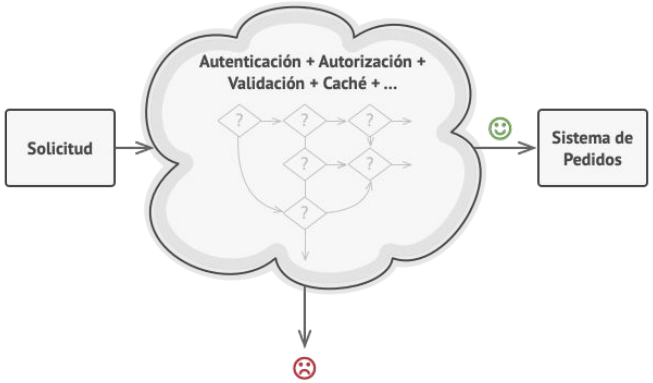
1. Cuando tu programa debe procesar distintos tipos de solicitudes de varias maneras, pero los tipos exactos de solicitudes y sus secuencias no se conozcan de antemano.
2. Utiliza el patrón Chain of Responsibility cuando el grupo de manejadores y su orden deban cambiar durante el tiempo de ejecución.
3. Utiliza el patrón cuando sea fundamental ejecutar varios manejadores en un orden específico.

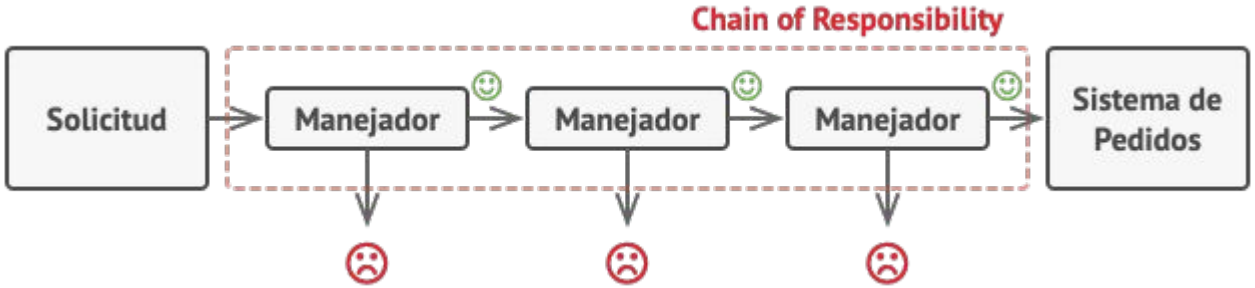


Chain of Responsibility

Cuándo aplicarlo







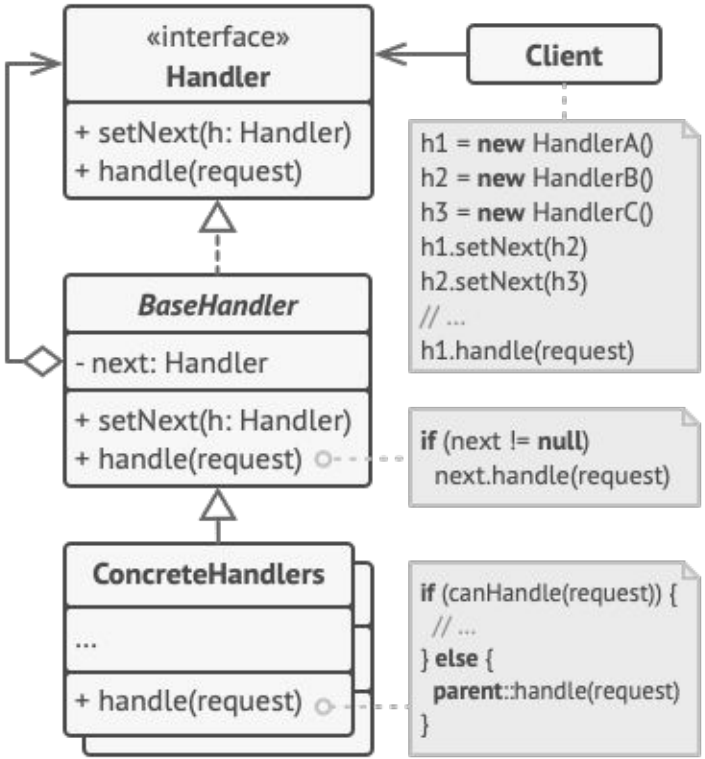
Chain of Responsibility

Estructura

1 La clase **Manejadora** declara la interfaz común a todos los manejadores concretos. Normalmente contiene un único método para manejar solicitudes, pero en ocasiones también puede contar con otro método para establecer el siguiente manejador de la cadena.

2 La clase **Manejadora Base** es opcional y es donde puedes colocar el código boilerplate (segmentos de código que suelen no alterarse) común para todas las clases manejadoras.

Normalmente, esta clase define un campo para almacenar una referencia al siguiente manejador. Los clientes pueden crear una cadena pasando un manejador al constructor o modificador (*setter*) del manejador previo. La clase también puede implementar el comportamiento de gestión por defecto: puede pasar la ejecución al siguiente manejador después de comprobar su existencia.



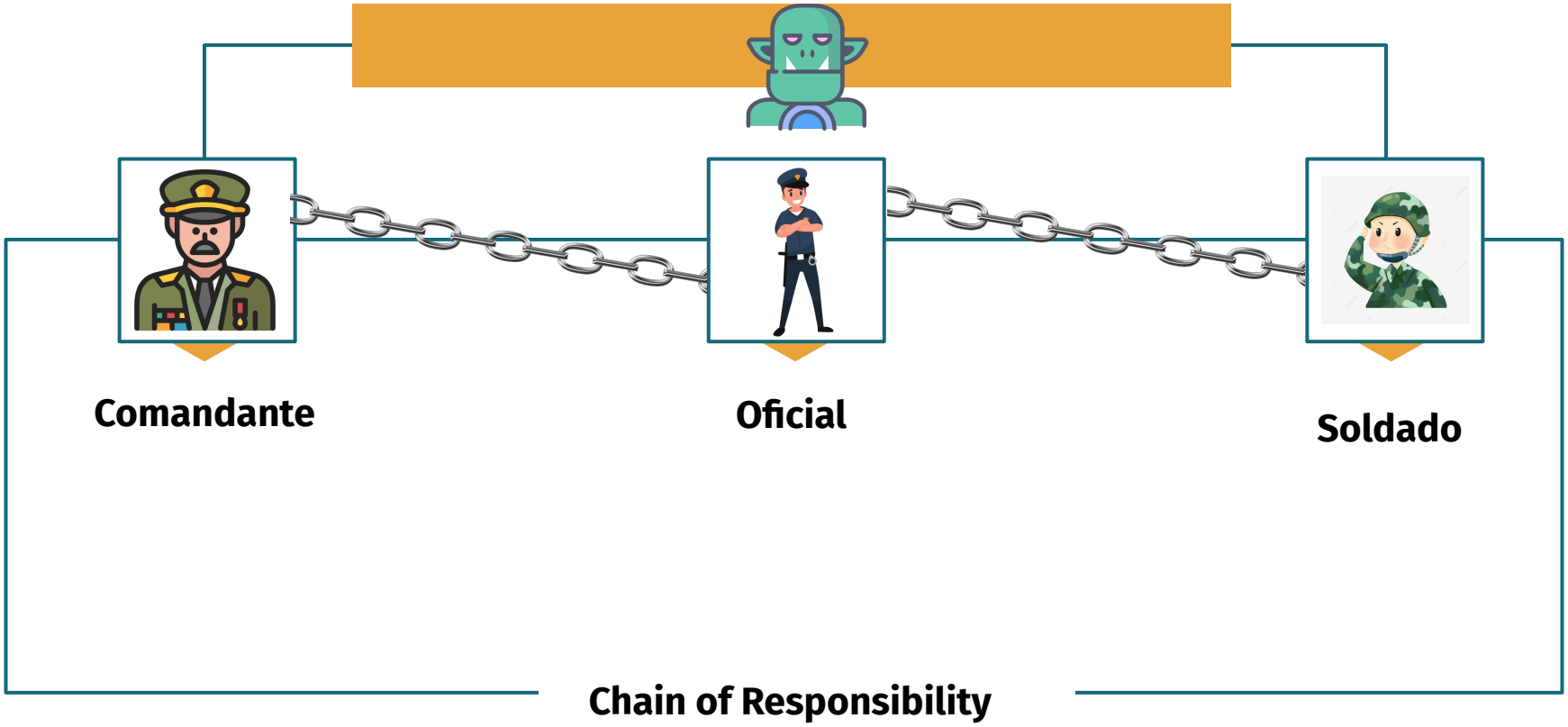
3 Los **Manejadores Concretos** contienen el código para procesar las solicitudes. Al recibir una solicitud, cada manejador debe decidir si procesarla y, además, si la pasa a lo largo de la cadena.

Habitualmente los manejadores son autónomos e inmutables, y aceptan toda la información necesaria únicamente a través del constructor.

4 El **Cliente** puede componer cadenas una sola vez o componerlas dinámicamente, dependiendo de la lógica de la aplicación. Observa que se puede enviar una solicitud a cualquier manejador de la cadena; no tiene por qué ser al primero.

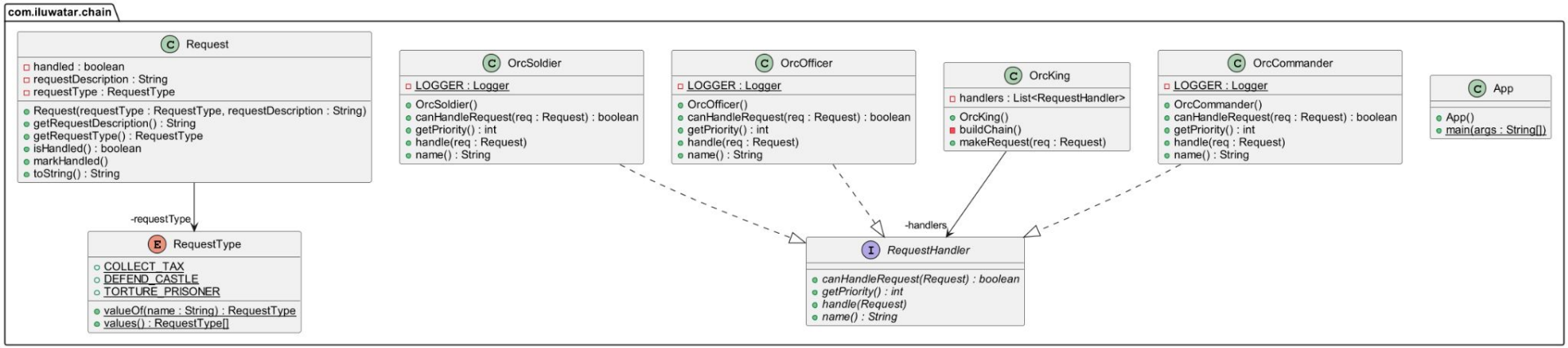
EJEMPLO DE APLICACIÓN

El Rey Orco da órdenes en voz alta a su ejército. El más cercano a reaccionar es el comandante, luego un oficial y después un soldado. El comandante, el oficial y el soldado forman una cadena de responsabilidad.



EJEMPLO DE APLICACIÓN

El Rey Orco da órdenes en voz alta a su ejército. El más cercano a reaccionar es el comandante, luego un oficial y después un soldado. El comandante, el oficial y el soldado forman una cadena de responsabilidad.



Chain of Responsibility

Referencias

[1] Refactoring Guru, "Chain of Responsibility Design Pattern," Refactoring Guru, [Online]. Available: https://refactoring.guru/es/design-patterns/chain-of-responsibility. [Accessed: Jun. 22, 2024].

[2] "Chain of Responsibility Pattern," Java Design Patterns, [Online]. Available: https://java-design-patterns.com/es/patterns/chain-of-responsibility/#explicacion. [Accessed: Jun. 22, 2024].