POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

9. *Chain of Responsability*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

- *Un profesor está en contacto directo con los alumnos pero no puede tomar decisiones ante todas las demandas posibles de éstos (p.e. cambio de plan de estudios, denunciar actitudes fraudulentas del propio profesor, …). Pero tampoco es deseable que el alumno tenga que conocer y discernir qué persona de la cadena de mando es la apropiada para cada queja (p.e. coordinador de la asignatura, jefe de departamento, director de escuela, vicerrector, …)*
- *Una solución mas sencilla para el alumno es que cuando tiene una queja se la comunica al profesor de la asignatura, si no puede/sabe resolverla, éste la eleva al coordinador de la asignatura, si no puede/sabe resolverla, éste la eleva al director del departamento, …*

- ***Evita acoplar al emisor de una petición a su receptor por dar más de un objeto una oportunidad de manejar la petición. Encadenar los objetos que reciben y pasar la solicitud a lo largo de la cadena hasta un objeto que la maneje.***

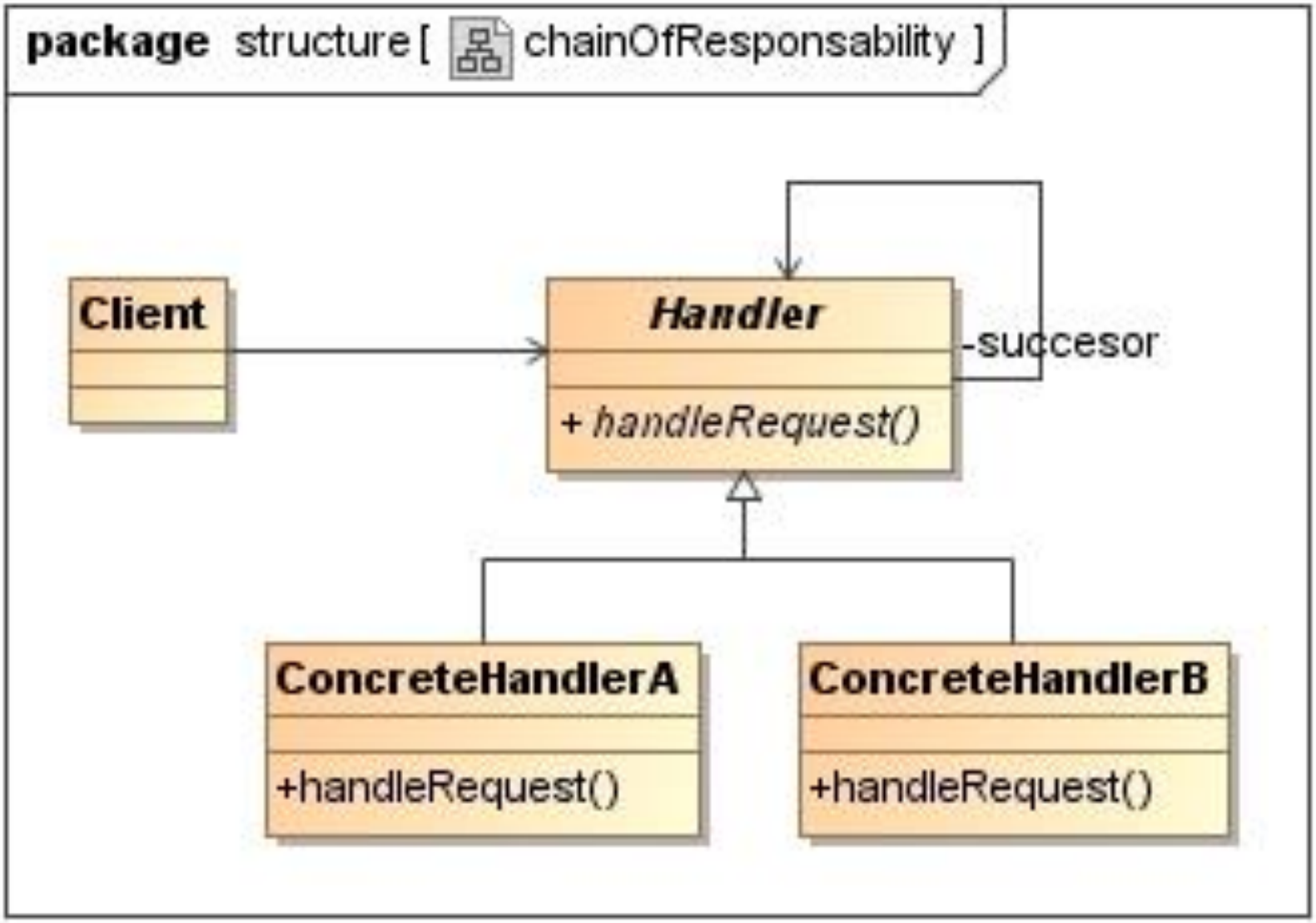
1. Problema

- El primer objeto de la cadena recibe la petición y, o bien la procesa o bien la redirige al siguiente candidato en la cadena, el cual hace lo mismo. El objeto que hizo la petición no tiene conocimiento explícito de quién la tratará – decimos que la petición tiene un receptor implícito -.
  - Para reenviar la petición a lo largo de la cadena, y para garantizar que los receptores permanecen implícitos, cada objeto de la cadena comparte una interfaz común para procesar peticiones y para acceder a su sucesor en la cadena.

1. Problema

- Úsese cuando:
  - Hay más de un objeto que pueden manejar una petición, y el manejador no se conoce a priori, sino que debería determinarse automáticamente.
  - Se quiere enviar una petición a un objeto entre varios sin especificar explícitamente el receptor
  - El conjunto de objetos que pueden tratar una petición debería ser especificado dinámicamente.

2. Solución


package structure [chainOfResponsability]
Client
Handler
+ handleRequest()
-succesor
ConcreteHandlerA
+handleRequest()
ConcreteHandlerB
+handleRequest()

2. Solución

- Hay dos formas de implementar las cadena sucesora:
  - Definir nuevos enlaces (normalmente en el Manejador, pero también podría ser en los objetos ManejadorContexto). Muchas veces se pueden usar referencias a objetos existentes. Por ejemplo, las referencias al padre en una jerarquía de parte-todo pueden definir el sucesor de una parte.
  - Usar los enlaces existentes funciona bien cuando los enlaces permiten la cadena que necesitamos. Nos evita tener que definir explícitamente nuevos enlaces y ahorra tiempo.

2. Solución

- Si no hay referencias preexistentes para definir una cadena, entonces tendremos que introducirlas nosotros mismos. En ese caso, el Manejador no sólo define la interfaz para las peticiones, sino que normalmente también se encarga de mantener el sucesor. Eso permite que el manejador proporcione una implementación predeterminada de ManejarPetición que reenvíe la petición al sucesor (si hay alguno). Si una subclase de ManejadorConcreto no está interesada en dicha petición, no tiene que redefinir la operación de reenvío, puesto que la implementación predeterminada la reenvía incondicionalmente.

2. Solución

- Hay varias opciones para representar las peticiones. En su forma más simple, una petición es una invocación a una operación insertada en el código. Esto resulta conveniente y segura pero entonces sólo se pueden reenviar el conjunto prefijado de peticiones que define la clase Manejador.
  - Una alternativa es más flexible, pero requiere sentencias condicionales para despachar la petición en función de su código. Y, lo que es peor, no hay un modo de pasar los parámetros seguros con respecto al tipo, por lo que éstos deben ser empaquetados y desempaquetados manualmente. Obviamente, esto es menos seguro que invocar una operación directamente.

2. Solución

- Para resolver el problema de los parámetros, podemos usar para las peticiones objetos aparte que incluyen los parámetros de la petición. Una clase Petición puede representar peticiones explícitamente, y se pueden definir nuevos tipos de peticiones mediante herencia. Las subclases pueden definir diferentes parámetros. Los manejadores deben conocer el tipo de petición para acceder a estos parámetros.

3. Consecuencias

- Libera a un objeto de tener que saber qué otro objeto maneja una petición. Un objeto sólo tiene que saber que una petición será manejada "de forma apropiada". Ni el receptor ni el emisor se conocen explícitamente entre ellos, y un objeto de la cadena tampoco tiene que conocer la estructura de ésta
  - Como resultado, la Cadena de Responsabilidad puede simplificar las interconexiones entre objetos. En vez de que los objetos mantengan referencias a todos los posibles receptores, solo tiene una única interfaz a su sucesor.

3. Consecuencias

- Ofrece una flexibilidad añadida para repartir responsabilidades entre objetos. Se pueden añadir o cambiar responsabilidades para tratar una petición modificando la cadena en tiempo de ejecución. Esto se puede combinar con la herencia para especializar los manejadores estáticamente.
- Dado que las peticiones no tienen un receptor explícito, no hay garantía de que sean manejadas – la petición puede alcanzar el final de la cadena sin haber sido procesada -. Una petición también puede quedar sin trata cuando la cadena no está configurada correctamente.

4. Relaciones

- *Observer* para ligar una estructura a la otra
- *State* para que un componente pueda cambiar su comportamiento cuando cambia su estado.
- *Command* para representar peticiones como objetos.
- *Template Method* en las operaciones primitivas para determinar si el objeto debería manejar la petición y para elegir el objeto al cual reenviarla.
- *Composite* donde los padres de los componentes pueden actuar como sucesores.