Patrón de Diseño Command

ANDRES LOPEZ MEZA

DISEÑO Y ARQUITECTURA DE SOFTWARE

Patrón de Diseño

- Es la descripción de la solución a un problema en cierto contexto
- Consiste en dar soluciones

Command

Este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además facilita la parametrización de los métodos.

Proposito

- Encapsula un mensaje como un objeto, con lo que permite gestionar colas o registro de mensaje y deshacer operaciones.
- Soportar restaurar el estado a partir de un momento dado.
- Ofrecer una interfaz común que permita invocar las acciones de forma uniforme y extender el sistema con nuevas acciones de forma más sencilla.

Motivo

- El concepto de "orden" puede ser ambiguo y complejo en los sistemas actuales y al mismo tiempo muy extendido: intérpretes de órdenes del sistema operativo, lenguajes de macros de paquetes ofimáticos, gestores de bases de datos, protocolos de servidores de Internet, etc.

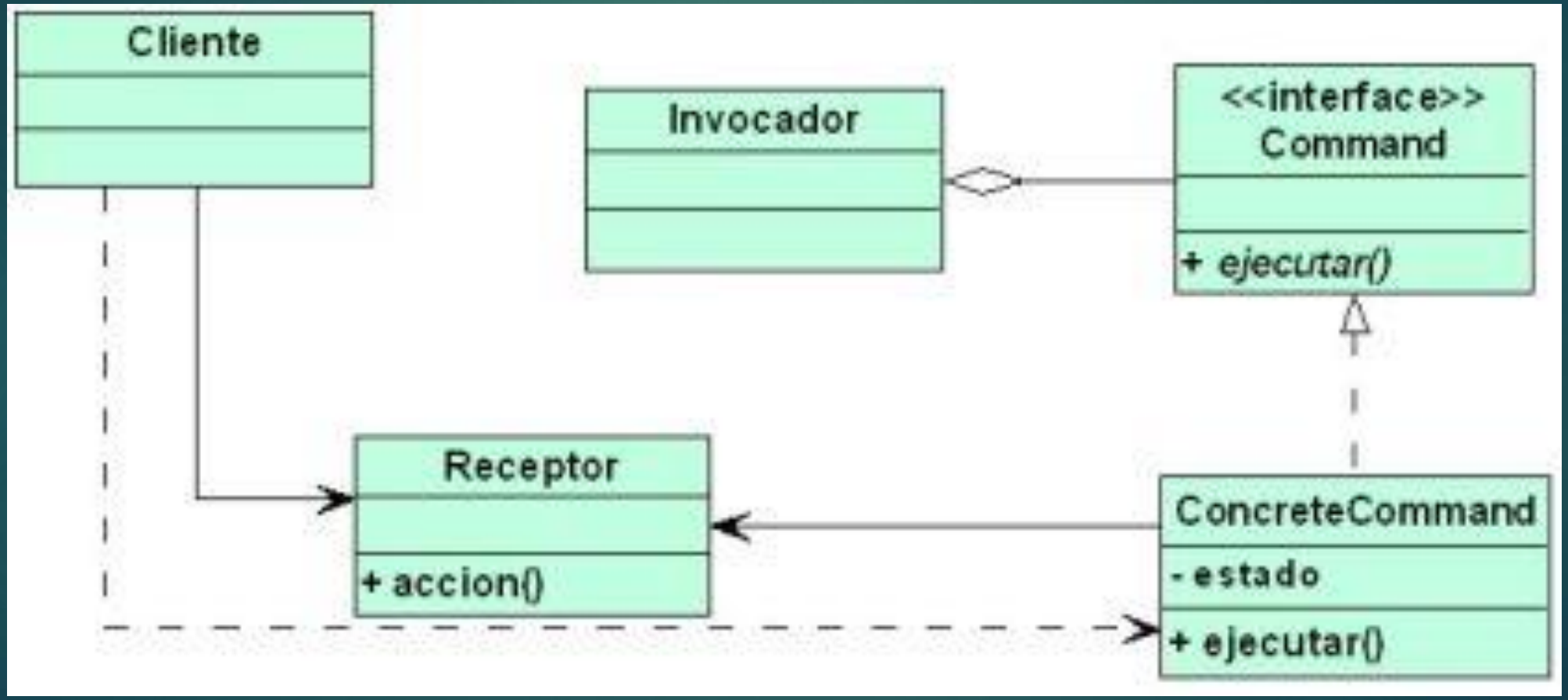
- Este patrón presenta una forma sencilla y versátil de implementar un sistema basado en comandos facilitándose su uso y ampliación.

Aplicaciones

- Facilitar la parametrización de las acciones a realizar.
- Independizar el momento de petición del de ejecución.
- Implementar CallBacks, especificando que órdenes queremos que se ejecuten en ciertas situaciones de otras órdenes. Es decir, un parámetro de una orden puede ser otra orden a ejecutar.
- Soportar el "deshacer".
- Desarrollar sistemas utilizando órdenes de alto nivel que se construyen con operaciones sencillas (primitivas).

Estructura

El **Invocador** no sabe quién es el **Receptor** ni la acción que se realizará, tan sólo invoca un **Comando** que ejecuta la acción adecuada



Ejemplo

- En este ejemplo configuramos el Switch con dos comandos: encender la luz y apagar la luz

- Un beneficio de esta implementación particular del patrón de comando es que el interruptor se puede usar con cualquier dispositivo, no solo con una luz. El Switch en la siguiente implementación de Python enciende y apaga una luz, pero el constructor del Switch puede aceptar cualquier subclase de Command para sus dos parámetros. Por ejemplo, puede configurar el interruptor para iniciar un motor.

En el diagrama de clase UML , la Invoker clase no implementa una solicitud directamente. En su lugar, el Invoke se refiere a la Command interfaz para realizar una solicitud (command.execute()), lo que la hace Invoker independiente de cómo se realiza la solicitud. La Command1clase implementa la Command interfaz realizando una acción en un receptor (receiver1.action1()).
El diagrama de secuencia de UML muestra las interacciones en tiempo de ejecución: el Invoker objeto llama execute()a un Command1 objeto. Command1 llama action1()a un Receiver1objeto, que realiza la solicitud.