Command

Propósito

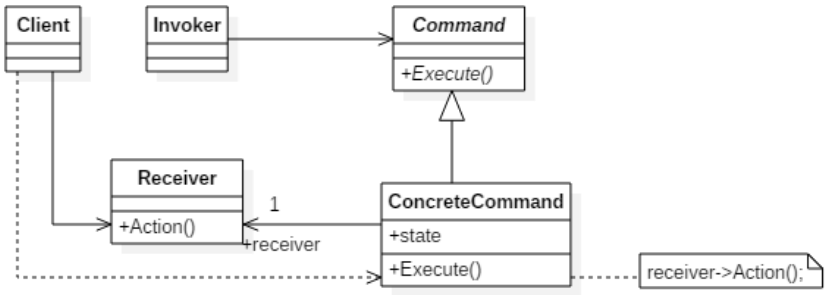
Encapsula una petición como un objeto, permitiendo parametrizar a los clientes con diferentes peticiones y soportar operaciones deshacer.

Aplicabilidad

El patrón Command se utiliza cuando:

- Se desea parametrizar objetos con una acción a ejecutar. Se puede expresar tal parametrización en un lenguaje procedural con una función callback, esto es, una función que se registra en alguna parte y que se ejecuta más tarde. Los commands son la versión orientada a objetos de las funciones callback
- Se desea especificar, alistar (meter en una cola) y ejecutar peticiones en diferentes momentos. Un objeto Command puede tener un tiempo de vida diferente de petición original. Si se puede representar el receptor de una petición en un espacio de direcciones diferente, se podría transferir el comando a un proceso diferente y completar la petición allí.
- Soporta deshacer. La operación Execute() del Command puede guardar el estado para revertir sus efectos en el propio comando. La interface del Command debe tener una operación Unexecute() que revierte los efectos de la operación. Los comandos ejecutados se almacenan en una lista (un histórico). Se puede conseguir niveles ilimitados de deshacer y repetir (undo y redo) recorriendo la lista en ambas direcciones y ejecutando Execute o Unexecute.
- Soporta registro de cambios de modo que se pueden volver a aplicar en el caso de que el sistema falle. Se puede ampliar la interface del Command con operaciones de carga y almacenamiento que permitan almacenar un registro persistente de los cambios. La recuperación de un fallo implica cargar los comandos almacenados y volver a ejecutarlos.
- Estructura un sistema con operaciones de alto nivel construidas sobre operaciones primitivas. Tal estructura es común en sistemas de información que soportan transacciones. Una transacción encapsula un conjunto de cambios sobre unos datos. El patrón Command ofrece una forma de modelar transacciones. Los Commands tienen una interface común, lo cual permite invocar todas las transacciones de la misma manera. El patrón además facilita la extensión del sistema con nuevas transacciones.

Estructura



Participantes

- Command: declara una interface para ejecutar una operación
- ConcreteCommand:
    - Define un vínculo entre un objeto Receiver y una acción
    - Implementa la operación Execute invocando la operación correspondiente sobre el objeto Receiver
- Client: crea un objeto ConcreteCommand y define su receptor
- Invoker: pide al comando que ejecute la petición
- Receiver: sabe cómo realizar las operaciones asociadas. Cualquier clase puede servir como Receiver

Colaboraciones

- El cliente crea un objeto ConcreteCommand y especifica su receptor
- Un objeto Invoker almacena el objeto ConcreteComand
- El Invoker emite una petición llamando Execute sobre el comando. Cuando los comandos soportan deshacer, el objeto ConcreteCommand almacena el estado previo a la ejecución de la operación Execute
- El objeto ConcreteComand invoca operaciones sobre su receptor para llevar a cabo la petición

Consecuencias

El patrón Command tiene las siguientes consecuencias:

1. El Command desacopla el objeto que invoca la operación del objeto que sabe cómo llevarla a cabo
2. Los Commands se dice que son first-class objects. Es decir que se pueden manipular y extender como cualquier otro objeto.
3. Puedes ensamblar comandos mediante un comando composite. Un ejemplo es la clase MacroCommand que se describe más adelante (en el libro Design Patterns). En general, los comandos compuestos son una instancia del patrón Composite.
4. Es fácil agregar nuevos comandos ya que no hay que modificar las clases existentes