Patrón de Diseño Command: Controlando acciones como objetos

En el desarrollo de software, la gestión de acciones y operaciones puede volverse compleja rápidamente. El patrón Command ofrece una solución elegante para encapsular solicitudes como objetos, lo que permite una mayor flexibilidad y control.

¿Qué es el patrón Command?

Encapsula una Solicitud

Convierte una solicitud o una operación en un objeto independiente. Esto permite parametrizar clientes con diferentes solicitudes.

Desacoplamiento

Separa el objeto que invoca la operación del objeto que realmente sabe cómo realizarla, promoviendo un bajo acoplamiento.

Funcionalidades Avanzadas

Facilita la implementación de funcionalidades como deshacer/rehacer, colas de comandos y el registro de acciones para auditoría o recuperación.

Componentes Clave:

- **Command:** Define la interfaz para ejecutar la operación. Típicamente con un método `execute()`.
- **ConcreteCommand:** Implementa la interfaz Command y enlaza un Receiver con una acción. Contiene una referencia al Receiver y llama a uno de sus métodos.
- **Receiver:** Realiza las operaciones reales cuando el comando lo ejecuta. No conoce los comandos, solo realiza su tarea.
- **Invoker:** Pide al comando que realice su solicitud. No sabe nada sobre la acción concreta que se realizará, solo conoce la interfaz Command.

Ejemplo: Control de una Puerta de Garaje

La Interfaz Command

```
import abc

class Command(abc.ABC):
    @abc.abstractmethod
    def execute(self):
        pass

    @abc.abstractmethod
    def undo(self):
        pass
```

El Receiver: Puerta de Garaje

```
class GarageDoor:
    def open(self):
        print("Puerta del garaje abierta.")

    def close(self):
        print("Puerta del garaje cerrada.")
```

Comandos Concretos

```
class GarageDoorOpenCommand(Command):
    def __init__(self, door: GarageDoor):
        self._door = door

    def execute(self):
        self._door.open()

    def undo(self):
        self._door.close()

class GarageDoorCloseCommand(Command):
    def __init__(self, door: GarageDoor):
        self._door = door

    def execute(self):
        self._door.close()

    def undo(self):
        self._door.open()
```

El Invoker y Uso

```
class RemoteControl:
 def __init__(self):
 self._command = None

 def set_command(self, command: Command):
 self._command = command

 def press_button(self):
 if self._command:
 self._command.execute()

 def press_undo(self):
 if self._command:
 self._command.undo()

# Uso del patrón
garage_door = GarageDoor()
open_command = GarageDoorOpenCommand(garage_door)
close_command = GarageDoorCloseCommand(garage_door)

remote = RemoteControl()

print("--- Abriendo la puerta ---")
remote.set_command(open_command)
remote.press_button() # Puerta del garaje abierta.

print("\n--- Deshaciendo la última acción (cerrando) ---")
remote.press_undo() # Puerta del garaje cerrada.

print("\n--- Cerrando la puerta ---")
remote.set_command(close_command)
remote.press_button() # Puerta del garaje cerrada.

print("\n--- Deshaciendo la última acción (abriendo) ---")
remote.press_undo() # Puerta del garaje abierta.
```