COMMAND

Patrón de Diseño

FINALIDAD

- Convertir una solicitud en un objeto independiente que contiene toda la información sobre la solicitud

DESVENTAJAS DE NO UTILIZARLO

Necesidad de crear múltiples subclases para manejar diferentes acciones → Estructura de código complicada y duplicación de código

Necesidad de invocar las mismas acciones desde varios lugares → Dificulta la mantenibilidad y la coherencia del código.

APLICABILIDAD

- Parametrizar objetos con operaciones.
- Implementar operaciones reversibles.
- Poner operaciones en cola, programar su ejecución, o ejecutarlas de forma remota.

VENTAJAS

- Desacoplar las clases que invocan operaciones de las que realizan esas operaciones.
- Introducir nuevos comandos en la aplicación sin descomponer el código existente.
- Implementar la ejecución diferida de operaciones.
- Ensamblar un grupo de comandos simples para crear uno complejo.

EJEMPLO

```python
# Clase Receiver (Receptor) - Representa la televisión que realizará las acciones.
class Television:
    def turn_on(self):
        print("Televisión encendida")

    def turn_off(self):
        print("Televisión apagada")

# Interfaz Command (Comando) - Define un método "execute" que todas las clases de comando deben implementar.
class Command:
    def execute(self):
        pass

# Clase ConcreteCommand (Comando Concreto) - Implementa un comando específico.
class TurnOnCommand(Command):
    def __init__(self, television):
        self.television = television

    def execute(self):
        self.television.turn_on()

# Clase ConcreteCommand (Comando Concreto) - Implementa otro comando específico.
class TurnOffCommand(Command):
    def __init__(self, television):
        self.television = television

    def execute(self):
        self.television.turn_off()
```

```python
# Clase Invoker (Invocador) - Encargada de ejecutar los comandos.
class RemoteControl:
    def __init__(self):
        self.command = None

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

    def press_button(self):
        self.command.execute()

# Uso del patrón Command
if __name__ == "__main__":
    television = Television()

    turn_on_command = TurnOnCommand(television)
    turn_off_command = TurnOffCommand(television)

    remote = RemoteControl()

    remote.set_command(turn_on_command)
    remote.press_button()

    remote.set_command(turn_off_command)
    remote.press_button()
```