DESIGN PATTERNS - CHAPITRE 15

Command en Python

Représenter une action et son destinataire par un objet déclenché par un autre composant.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Représenter une action et son destinataire par un objet déclenché par un autre composant.
- Distinguer exécution d'une commande et restauration d'un état pour l'annuler.

Prérequis. Savoir utiliser classes, héritage, composition, listes, dictionnaires et exceptions Python. Les mécanismes spécifiques sont expliqués avant leur emploi.

Command : intention et expérience

Command : représenter une action par un objet

Une **Command** rassemble l'action et son destinataire. La télécommande déclenche `execute()` sans connaître les détails de la lampe. Cette version n'implémente pas d'historique ni d'annulation. Une extension avec `undo()` devrait conserver l'état antérieur pour le restaurer, pas seulement effectuer l'action opposée.

Python

```python
# example.py
class Command:
    """
    Abstract command class.
    """
    def execute(self):
        """
        Executes the command.
        """
        raise NotImplementedError

class Light:
    """
    Receiver class that performs the actual actions.
    """
    def turn_on(self):
        print("Light turned on")

    def turn_off(self):
        print("Light turned off")

class TurnOnCommand(Command):
    """
    Concrete command to turn on the light.
    """
    def __init__(self, light):
        self._light = light

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

class TurnOffCommand(Command):
    """
    Concrete command to turn off the light.
    """
    def __init__(self, light):
        self._light = light
```

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

class RemoteControl:
    """
    Invoker class that holds and executes commands.
    """
    def __init__(self):
        self._commands = {}

    def set_command(self, command_name, command):
        """
        Assigns a command to a specific button.
        """
        self._commands[command_name] = command

    def press_button(self, command_name):
        """
        Executes the command associated with the button.
        """
        if command_name in self._commands:
            self._commands[command_name].execute()
        else:
            print(f"No command associated with {command_name}")

# Client code
light = Light()
turn_on_command = TurnOnCommand(light)
turn_off_command = TurnOffCommand(light)

remote = RemoteControl()
remote.set_command("on", turn_on_command)
remote.set_command("off", turn_off_command)

remote.press_button("on")
remote.press_button("off")
```

Vérifier et interpréter

Affectez une autre commande au bouton on puis déclenchez-le. Le code de la télécommande ne doit pas changer. L'objet commande facilite report d'exécution ou historique, mais ces possibilités exigent du code supplémentaire : cet exemple ne stocke ni instant d'exécution ni état antérieur.

Version web : https://www.clearcode.fr/courses/programming/design-patterns/comportement/command