DESIGN PATTERNS - CHAPITRE 14

Chain of Responsibility en Python

Transmettre une requête le long d'une chaîne jusqu'au gestionnaire qui peut répondre.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Transmettre une requête le long d'une chaîne jusqu'au gestionnaire qui peut répondre.
- Tester les bornes traitées et la sortie sans responsable.

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

Chain of Responsibility : intention et expérience

Chain of Responsibility : trouver un responsable

Une requête traverse une chaîne de gestionnaires jusqu'à celui qui la traite. Chaque gestionnaire peut répondre ou transmettre au suivant. Le gestionnaire final représente le cas non reconnu ; vérifiez aussi ce cas pour éviter une requête silencieusement perdue.

Python

```python
# example.py
class Handler:
    """
    Abstract handler class.
    """
    def __init__(self, successor=None):
        self._successor = successor

    def handle_request(self, request):
        """
        Handles the request if possible, otherwise defers to the successor.
        """
        if self._successor is not None:
            self._successor.handle_request(request)

class ConcreteHandler1(Handler):
    """
    Concrete handler that handles requests in the range 0 to 10.
    """
    def handle_request(self, request):
        if 0 < request <= 10:
            print(f"Handler 1 handled request {request}")
        else:
            super().handle_request(request)

class ConcreteHandler2(Handler):
    """
    Concrete handler that handles requests in the range 11 to 20.
    """
    def handle_request(self, request):
        if 10 < request <= 20:
            print(f"Handler 2 handled request {request}")
        else:
            super().handle_request(request)

class DefaultHandler(Handler):
    """
```

```
    Default handler that handles any request not handled by previous handlers.
    """
    def handle_request(self, request):
        print(f"End of chain, no handler for request {request}")

# Client code
handler1 = ConcreteHandler1()
handler2 = ConcreteHandler2()
default_handler = DefaultHandler()

handler1._successor = handler2
handler2._successor = default_handler

requests = [2, 5, 14, 22, 18, 3, 35, 27, 20]
for request in requests:
    handler1.handle_request(request)
```

Vérifier et interpréter

Essayez les bornes 0, 10, 11 et 20, puis 21. Identifiez le premier gestionnaire capable de traiter chaque entier. La chaîne permet de réorganiser les responsables, mais son ordre change la priorité ; un maillon absent ou un cycle peut respectivement perdre une requête ou empêcher la terminaison.

Version web : https://www.clearcode.fr/courses/programming/design-patterns/comportement/chain-of-responsibility