DESIGN PATTERNS - CHAPITRE 20

Mediator en Python

Centraliser les interactions entre collègues dans un coordinateur explicite.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Centraliser les interactions entre collègues dans un coordinateur explicite.
- Vérifier les destinataires d'un message sans relier chaque participant aux autres.

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

Mediator : intention et expérience

Coordonner les interactions

Si chaque composant appelle directement tous les autres, les dépendances se multiplient. Un **Mediator** centralise leurs interactions : chaque participant parle au médiateur. Ici, un utilisateur connaît le salon mais pas la liste des autres utilisateurs.

Le contrat `send(sender, message)` transmet un message aux autres inscrits, sans le retourner à l'expéditeur. Les boîtes de réception sont de simples listes en mémoire ; aucun message réseau n'est envoyé. Créez trois utilisateurs, envoyez un message et comptez les réceptions.

Le médiateur simplifie les relations entre participants, mais peut concentrer trop de règles. Séparez les responsabilités si le salon devient une classe qui gère tout. Observer distribue un événement aux abonnés ; Mediator exprime surtout une coordination entre collègues. Les deux intentions peuvent être combinées.

Python

```python
# example.py
class ChatRoom:
    def __init__(self):
        self._users = []

    def register(self, user):
        if user not in self._users:
            self._users.append(user)

    def send(self, sender, message):
        if sender not in self._users:
            raise ValueError("Unknown sender")
        for user in tuple(self._users):
            if user is not sender:
                user.receive(sender.name, message)

class User:
    def __init__(self, name, room):
        self.name = name
        self.room = room
        self.inbox = []
        room.register(self)

    def send(self, message):
        self.room.send(self, message)

    def receive(self, sender, message):
        self.inbox.append((sender, message))

room = ChatRoom()
alice = User("Alice", room)
bob = User("Bob", room)
alice.send("Hello")
print(bob.inbox)    # [('Alice', 'Hello')]
print(alice.inbox)  # []
```

Vérifier et interpréter

Ajoutez un troisième utilisateur puis envoyez un message : chacun des deux destinataires le reçoit une fois et l'expéditeur ne le reçoit pas. Comparez avec une solution où chaque utilisateur garde la liste de tous les autres. Le médiateur réduit ces liens directs, mais centralise les règles de distribution.

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