DESIGN PATTERNS - CHAPITRE 17

Observer en Python

Distribuer un changement à plusieurs abonnés sans intégrer leurs réactions au sujet.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Distribuer un changement à plusieurs abonnés sans intégrer leurs réactions au sujet.
- Tester abonnement, désabonnement et modification de la liste pendant une notification.

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

Observer : intention et expérience

Observer : avertir plusieurs abonnés

Le sujet maintient des observateurs et les avertit d'un changement. Leurs fonctions doivent accepter le même message. Le sujet connaît ce contrat, sans connaître le travail de chaque observateur. Dans ces exemples, les appels sont synchrones : une exception dans un abonné interrompt la notification.

Python

```python
# example.py
class Subject:
    def __init__(self):
        self._observers = []  # list of observers

    def attach(self, observer):
        self._observers.append(observer)

    def detach(self, observer):
        self._observers.remove(observer)

    def notify(self, *args, **kwargs):
        for observer in self._observers:
            observer(*args, **kwargs)

class Observer:
    def update(self, *args, **kwargs):
        pass

# Example usage
def observer_function(message):
    print(f"Observer received: {message}")

subject = Subject()
subject.attach(observer_function)

subject.notify("State changed!") # Output: Observer received: State changed!

subject.detach(observer_function)
subject.notify("This message will not be received.") # Nothing happens
```

Une version avec `connect`, `disconnect` et `send` rend le cycle d'abonnement explicite. Aucun paquet supplémentaire n'est nécessaire. Le parcours d'un instantané de la liste évite qu'un désabonnement pendant une notification ne saute un autre abonné. Après déconnexion, le second message ne produit aucune sortie.

Python

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

    def connect(self, listener):
        if listener not in self._listeners:
            self._listeners.append(listener)

    def disconnect(self, listener):
        self._listeners.remove(listener)

    def send(self, message):
        for listener in tuple(self._listeners):
            listener(message)


def observer_function(message):
    print(f"Observer received: {message}")

signal = Signal()
signal.connect(observer_function)
signal.send("State changed")
signal.disconnect(observer_function)
signal.send("No listener receives this message")
```

Vérifier et interpréter

Abonnez deux fonctions, envoyez un message, puis désabonnez la première et recommencez. Seule la seconde doit recevoir le dernier message. Le sujet ignore la logique des abonnés, mais doit conserver leurs références et définir ce qui se passe lorsqu'un abonné échoue ou se désabonne pendant la notification.

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