DESIGN PATTERNS - CHAPITRE 31

Justifier un choix de conception

Justifier une solution à partir d'un contrat, d'une variation et de ses compromis.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Décrire la portée et le cycle de vie d'un état partagé.
- Proposer une solution plus simple lorsqu'elle répond au même besoin.

Prérequis. Avoir étudié les patrons de création, de structure et de comportement.

Justifier un choix de conception

Avant d'ajouter un patron, formulez le changement à rendre possible et le comportement à préserver. Cette version courte de Singleton mémorise un objet avec `__new__`, mais un éventuel `__init__` serait encore appelé à chaque construction. Elle ne constitue pas une infrastructure de gestion de ressources.

Python

```python
# example.py
class Singleton:
    _instance = None  # Private attribute to hold the instance

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance

# Example usage:
# first_instance = Singleton()
# second_instance = Singleton()
# print(first_instance is second_instance)  # Output: True
```

Cette fabrique simple de boutons termine le parcours. Décrivez sa politique pour un système inconnu, puis comparez-la à celle qui lèverait `ValueError`. Un choix de conception n'est complet qu'avec ses cas limites.

Python

```python
# example.py
class Button:
    def __init__(self):
        self.label = "Generic Button"

    def click(self):
        print("Generic Button Clicked")

class WindowsButton(Button):
    def __init__(self):
        self.label = "Windows Button"

    def click(self):
        print("Windows Button Clicked")

class LinuxButton(Button):
    def __init__(self):
        self.label = "Linux Button"

    def click(self):
        print("Linux Button Clicked")

class ButtonFactory:
    def create_button(self, os_type):
        if os_type == "Windows":
            return WindowsButton()
        elif os_type == "Linux":
            return LinuxButton()
        else:
            return Button()  # Default button

# Example Usage:
# factory = ButtonFactory()
# button1 = factory.create_button("Windows")
# print(button1.label)  # Output: Windows Button
# button1.click()      # Output: Windows Button Clicked
```

Vérifier son raisonnement

- Quel contrat le client utilise-t-il ?
- Quelle variation justifie une fonction, une composition ou un héritage ?
- Quel état est partagé, par qui, et pendant combien de temps ?
- Quelles erreurs et quelles limites faut-il tester ?
- Une solution plus simple répond-elle au même besoin ?

Références Python

Consultez le **modèle de données** pour la construction des objets et les protocoles, le **glossaire des itérateurs** et la documentation de **functools.wraps** pour les décorateurs.

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