DESIGN PATTERNS - CHAPITRE 18

Strategy en Python

Confier une opération à une politique interchangeable reçue par le contexte.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Confier une opération à une politique interchangeable reçue par le contexte.
- Comparer une stratégie sous forme de fonction et sous forme d'objet.

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

Strategy : intention et expérience

Strategy : choisir une politique interchangeable

Une **Strategy** représente une manière d'effectuer une opération ; le contexte la reçoit comme dépendance. Les objets de paiement exposent ici `pay(amount)`. Tout est simulé par des affichages : aucun paiement n'est exécuté. Les données sont fictives ; une application réelle ne doit pas conserver un cryptogramme ni journaliser des coordonnées sensibles.

Les montants flottants simplifient cette simulation. Pour des calculs monétaires réels, définissez une unité entière ou une règle d'arrondi avec `Decimal`.

Python

```python
# example.py
from abc import ABC, abstractmethod

class PaymentStrategy(ABC):
    """
    Abstract base class for payment strategies.
    Defines the common interface that all strategies must implement.
    """
    @abstractmethod
    def pay(self, amount: float):
        """
        Abstract method to perform payment.
        Subclasses must implement this method.
        """
        pass

class CreditCardPayment(PaymentStrategy):
    """
    Concrete strategy for credit card payments.
    """
    def __init__(self, card_number: str, cvv: str):
        self.card_number = card_number
        self.cvv = cvv

    def pay(self, amount: float):
        print(f"Paying {amount} using Credit Card: {self.card_number}")

class PayPalPayment(PaymentStrategy):
    """
    Concrete strategy for PayPal payments.
    """
    def __init__(self, email: str):
        self.email = email

    def pay(self, amount: float):
        print(f"Paying {amount} using PayPal: {self.email}")
```

```python
class ShoppingCart:
    """
    Context class that uses the strategy.
    It doesn't know the concrete implementation of the strategy.
    """
    def __init__(self, payment_strategy: PaymentStrategy):
        self.payment_strategy = payment_strategy

    def checkout(self, amount: float):
        self.payment_strategy.pay(amount)

# Example usage
credit_card = CreditCardPayment(card_number="1234-5678-9012-3456", cvv="123")
paypal = PayPalPayment(email="user@example.com")

cart1 = ShoppingCart(payment_strategy=credit_card)
cart1.checkout(100.0)  # Paying 100.0 using Credit Card: 1234-5678-9012-3456

cart2 = ShoppingCart(payment_strategy=paypal)
cart2.checkout(50.0)   # Paying 50.0 using PayPal: user@example.com
```

Une fonction peut représenter la stratégie lorsque celle-ci n'a pas d'état propre. Ici, `checkout` lui transmet le montant et ses arguments nommés. L'appelant doit fournir les arguments adaptés à la stratégie ; une signature commune plus étroite réduit ce risque.

Python

```python
# example.py
def credit_card_payment(amount: float, card_number: str, cvv: str):
    """
    Payment strategy using a credit card.
    """
    print(f"Paying {amount} using Credit Card: {card_number}")

def paypal_payment(amount: float, email: str):
    """
    Payment strategy using PayPal.
    """
    print(f"Paying {amount} using PayPal: {email}")

def checkout(amount: float, payment_strategy, **kwargs):
    """
    Checkout function that accepts a payment strategy and its arguments.
    """
    payment_strategy(amount, **kwargs)

# Example usage
checkout(amount=100.0, payment_strategy=credit_card_payment, card_number="1234-5678-9012-3456", cvv="123")
# Paying 100.0 using Credit Card: 1234-5678-9012-3456

checkout(amount=50.0, payment_strategy=paypal_payment, email="user@example.com")
# Paying 50.0 using PayPal: user@example.com
```

Vérifier et interpréter

Remplacez la stratégie d'un panier par une autre implémentation de pay. Le panier continue à transmettre le même montant. Le contrat doit permettre ce remplacement ; l'appelant conserve cependant la responsabilité de choisir la politique. Créer une classe par politique devient inutile quand une fonction sans état suffit.

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