DESIGN PATTERNS - CHAPITRE 19

Template Method en Python

Fixer l'ordre d'un algorithme et laisser les sous-classes réaliser certaines étapes.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Fixer l'ordre d'un algorithme et laisser les sous-classes réaliser certaines étapes.
- Distinguer variation par héritage et variation par composition.

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

Template Method : intention et expérience

Template Method : fixer un déroulement

La **Template Method** fixe l'ordre d'un algorithme dans une classe de base. Les sous-classes redéfinissent certaines étapes. Ici, `build_house()` impose fondations, murs, toit, fenêtres, finition. Malgré le mot Builder dans les noms de classes, cette organisation illustre Template Method : l'enchaînement est hérité, aucun objet produit n'est assemblé par un Builder indépendant.

Strategy choisit un collaborateur par composition ; Template Method choisit des étapes par héritage. Préférez la composition si la politique doit changer sur une instance pendant son utilisation.

Python

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

class HouseBuilder(ABC):
    """
    Abstract class defining the template for building a house.
    """

    def build_house(self):
        """
        Template method to define the order of building steps.
        """
        self.prepare_foundation()
        self.build_walls()
        self.install_roof()
        self.install_windows()
        self.finish_house()

    @abstractmethod
    def prepare_foundation(self):
        """
        Abstract method to prepare the foundation.
        Subclasses must implement this.
        """
        pass

    @abstractmethod
    def build_walls(self):
        """
        Abstract method to build the walls.
        Subclasses must implement this.
        """
        pass

    @abstractmethod
    def install_roof(self):
        """
```

```
        Abstract method to install the roof.
        Subclasses must implement this.
        """
        pass

    def install_windows(self):
        """
        Concrete method to install windows.
        """
        print("Installing windows")

    def finish_house(self):
        """
        Concrete method to finish the house.
        """
        print("Finishing house")

class WoodenHouseBuilder(HouseBuilder):
    """
    Concrete class to build a wooden house.
    """

    def prepare_foundation(self):
        print("Preparing wooden foundation")

    def build_walls(self):
        print("Building wooden walls")

    def install_roof(self):
        print("Installing wooden roof")

class StoneHouseBuilder(HouseBuilder):
    """
    Concrete class to build a stone house.
    """

    def prepare_foundation(self):
        print("Preparing stone foundation")
```

```
    def build_walls(self):
        print("Building stone walls")

    def install_roof(self):
        print("Installing stone roof")

# Usage
wooden_house_builder = WoodenHouseBuilder()
wooden_house_builder.build_house()

stone_house_builder = StoneHouseBuilder()
stone_house_builder.build_house()
```

Vérifier et interpréter

Redéfinissez install_windows dans une sous-classe et enregistrez l'ordre des étapes de build_house. La nouvelle opération doit rester entre toit et finition. L'algorithme commun évite de dupliquer cette séquence, mais son héritage lie les variantes à la classe de base ; une variation dynamique justifie souvent une stratégie reçue en argument.

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