DESIGN PATTERNS - CHAPITRE 5

Factory Method en Python

Distinguer la sélection d'une fabrique simple de la création redéfinie par une sous-classe.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Distinguer la sélection d'une fabrique simple de la création redéfinie par une sous-classe.
- Relier l'opération générale au Factory Method appelé par héritage.

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

Factory Method : intention et expérience

Fabrique simple : fonction, classe ou registre

Les documents suivants exposent tous `render() -> str`. Les messages simulent un rendu ; aucun fichier PDF ou Word n'est produit. Une fonction choisit leur classe et lève `ValueError` pour un type inconnu.

Python

```python
# example.py
class PDFDocument:
    def __init__(self, content):
        self.content = content

    def render(self):
        return "Rendering PDF: " + self.content

class WordDocument:
    def __init__(self, content):
        self.content = content

    def render(self):
        return "Rendering Word document: " + self.content

def create_document(document_type, content):
    """
    Factory function to create document objects.
    Args:
        document_type (str): The type of document to create ("pdf" or "word").
        content (str): The content of the document.
    Returns:
        PDFDocument or WordDocument: An instance of the appropriate document
class.
    Raises:
        ValueError: If the document type is invalid.
    """
    if document_type == "pdf":
        return PDFDocument(content)
    elif document_type == "word":
        return WordDocument(content)
    else:
        raise ValueError("Invalid document type")

# Usage
pdf_doc = create_document("pdf", "This is a PDF document.")
print(pdf_doc.render())
```

```
word_doc = create_document("word", "This is a Word document.")
print(word_doc.render())
```

Suite du bloc précédent : conservez les définitions de `PDFDocument` et `WordDocument`. Cette classe regroupe la même sélection. Elle n'apporte un avantage sur la fonction que si elle porte une configuration ou un état utile.

Python

```python
# example.py
class DocumentFactory:
    def create_document(self, document_type, content):
        """
        Factory class to create document objects.
        Args:
            document_type (str): The type of document to create ("pdf" or
"word").
            content (str): The content of the document.
        Returns:
            PDFDocument or WordDocument: An instance of the appropriate
document class.
        Raises:
            ValueError: If the document type is invalid.
        """
        if document_type == "pdf":
            return PDFDocument(content)
        elif document_type == "word":
            return WordDocument(content)
        else:
            raise ValueError("Invalid document type")

# Usage
factory = DocumentFactory()
pdf_doc = factory.create_document("pdf", "This is a PDF document.")
print(pdf_doc.render())

word_doc = factory.create_document("word", "This is a Word document.")
print(word_doc.render())
```

Un registre associe une clé à un constructeur dans un dictionnaire. Il remplace les branches de sélection, mais exige toujours d'enregistrer chaque nouveau type. Vérifiez que les deux produits respectent bien le même contrat `render()`.

Python

```python
# example.py
class PDFDocument:
    def __init__(self, content):
        self.content = content

    def render(self):
        return "Rendering PDF: " + self.content

class WordDocument:
    def __init__(self, content):
        self.content = content

    def render(self):
        return "Rendering Word document: " + self.content

class DocumentFactory:
    def __init__(self):
        self._document_types = {
            "pdf": PDFDocument,
            "word": WordDocument
        }

    def create_document(self, document_type, content):
        """
        Factory class to create document objects using a dictionary.
        Args:
            document_type (str): The type of document to create ("pdf" or
"word").
            content (str): The content of the document.
        Returns:
            PDFDocument or WordDocument: An instance of the appropriate
document class.
        Raises:
            ValueError: If the document type is invalid.
        """
        document_class = self._document_types.get(document_type)
        if document_class:
```

```
            return document_class(content)
        else:
            raise ValueError("Invalid document type")

# Usage
factory = DocumentFactory()
pdf_doc = factory.create_document("pdf", "This is a PDF document.")
print(pdf_doc.render())

word_doc = factory.create_document("word", "This is a Word document.")
print(word_doc.render())
```

Factory Method : déléguer la création à une sous-classe

Une fonction de sélection ou un registre n'est pas, à lui seul, le patron GoF **Factory Method**. Dans ce patron, une opération générale appelle une méthode de création redéfinie par les sous-classes. Ici, `export()` reste inchangée et `create_report()` choisit le produit. GoF désigne les quatre auteurs qui ont décrit ce catalogue de patrons orientés objet.

Python

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

class Report:
    def render(self):
        return "Plain report"

class HtmlReport:
    def render(self):
        return "<p>HTML report</p>"

class Exporter(ABC):
    @abstractmethod
    def create_report(self):
        """Factory Method: subclasses choose a product with render()."""
        raise NotImplementedError

    def export(self):
        report = self.create_report()
        return report.render()

class PlainExporter(Exporter):
    def create_report(self):
        return Report()

class HtmlExporter(Exporter):
    def create_report(self):
        return HtmlReport()

print(PlainExporter().export())  # Plain report
print(HtmlExporter().export())   # <p>HTML report</p>
```

Vérifier et interpréter

Ajoutez un exporteur dont create_report retourne un nouveau produit qui respecte render(). L'opération export doit fonctionner sans être modifiée. En contrepartie, chaque variante demande une sous-classe d'exporteur ; si seul un choix de constructeur varie, une fabrique simple ou une fonction reçue en argument peut être suffisante.

Version web : https://www.clearcode.fr/courses/programming/design-patterns/creation/factory-method