DESIGN PATTERNS - CHAPITRE 22

# State en Python

Changer les opérations permises d'un objet selon son état et ses transitions.

Par Ali ZAINOUL · Cours gratuit

### Ce que vous allez apprendre

- Changer les opérations permises d'un objet selon son état et ses transitions.
- Tester chaque opération dans les états brouillon et publié.

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

## State : intention et expérience

### Définir les états et les transitions

Un même appel doit parfois répondre différemment selon une phase de vie. Le patron **State** déplace ces comportements vers des objets états et donne au contexte une référence vers son état courant.

Notre document commence en brouillon : il peut être modifié puis publié si le texte n'est pas vide. Une fois publié, il refuse l'édition et une deuxième publication. La transition est déclenchée par l'état brouillon. Testez chaque couple état/opération, dont le texte vide.

State ressemble à Strategy par sa composition, mais exprime un cycle de vie et ses transitions plutôt qu'une politique choisie librement par le client. Pour deux branches stables, une condition peut suffire ; les classes états deviennent utiles quand plusieurs opérations varient ensemble. L'attribut reste public dans cette démonstration et pourrait être modifié en dehors du protocole.

Python

```python
# example.py
class Draft:
    def publish(self, document):
        if not document.text.strip():
            raise ValueError("An empty document cannot be published")
        document.state = Published()

    def edit(self, document, text):
        document.text = text

class Published:
    def publish(self, document):
        raise RuntimeError("Already published")

    def edit(self, document, text):
        raise RuntimeError("Published documents cannot be edited")

class Document:
    def __init__(self, text):
        self.text = text
        self.state = Draft()

    def publish(self):
        self.state.publish(self)

    def edit(self, text):
        self.state.edit(self, text)

document = Document("First draft")
document.edit("Ready")
document.publish()
print(type(document.state).__name__)  # Published
try:
    document.edit("Too late")
except RuntimeError as error:
    print(error)
```

## Vérifier et interpréter

Essayez de publier un texte vide, d'éditer un brouillon puis d'éditer un document publié. La publication invalide doit conserver l'état Draft ; une publication valide mène à Published. Les objets états regroupent les règles d'une phase, au prix d'un nombre de classes et de transitions à contrôler.

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