Fiche résumé – Pattern État (State)

Le pattern État (State) est un modèle comportemental qui permet à un objet de modifier son comportement lorsque son état interne change, sans recourir à de longues structures conditionnelles (if ou switch).

L'idée est de déléguer les comportements spécifiques à des classes représentant les différents états possibles de l'objet. Ce pattern rend le code plus clair, plus extensible et facilite la gestion de la complexité liée aux transitions d'états.

Exemple d'un besoin

Un téléphone peut se trouver dans différents états : éteint (Off), verrouillé (Locked), ou prêt à être utilisé (Ready). Son comportement varie selon cet état :

- Si le téléphone est éteint, appuyer sur un bouton doit simplement l'allumer.
- Une fois verrouillé, le même bouton n'a plus le même effet : il doit afficher l'écran de déverrouillage.
- Quand il est prêt, ce bouton peut ouvrir le menu ou mettre le téléphone en veille.

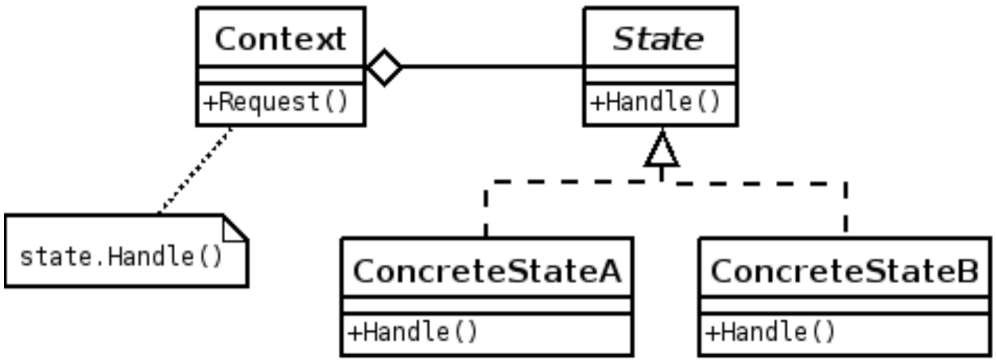
Dans une conception classique, on risquerait d'écrire de nombreux if ou switch pour vérifier l'état actuel du téléphone, rendant le code difficile à maintenir et à faire évoluer.

Solution avec le pattern État

Le pattern État permet d'encapsuler les comportements propres à chaque état dans des classes séparées. Chaque état (Off, Locked, Ready) devient une classe implémentant la même interface (PhoneState, par exemple).

Le téléphone possède une référence vers son état courant, et délègue les actions à cet objet. Ainsi, lorsque l'état change (ex. passage de Locked à Ready), il suffit de remplacer l'objet d'état sans modifier la logique principale. Cela permet de rendre le code plus lisible, extensible et ouvert à l'ajout de nouveaux états.

Voici le diagramme de classe générique du pattern état.



Détails des classes

Classe	**Rôle**
Contexte (Context)	Objet principal dont le comportement change selon l'état (ex. Phone).
État (State)	Interface commune définissant les opérations possibles.
ÉtatConcret (ConcreteState)	Implémente le comportement spécifique à un état particulier (ex. OffState, LockedState, ReadyState).
Contexte (Context)	Objet principal dont le comportement change selon l'état (ex. Phone).
État (State)	Interface commune définissant les opérations possibles.

Principe SOLID

- SRP : chaque classe d'état gère un comportement spécifique.
- OCP : on peut ajouter de nouveaux états sans modifier les existants.
- LSP : tous les états peuvent être utilisés via l'interface State.
- ISP : l'interface State ne contient que les méthodes utiles à tous les états.
- DIP : le contexte dépend de l'abstraction State, pas des implémentations concrètes.

Limites du pattern

Si le nombre d'états est important, cela va augmenter le nombre de classes.

Nécessite une bonne gestion des transitions entre états.

Peut sembler « lourd » pour des systèmes très simples.

Comparaison GOF

Le pattern État est très proche du pattern Stratégie, car tous deux reposent sur la délégation à un objet interne qui implémente une interface commune.

La différence principale réside dans l'intention.

Aspect	**State (État)**	**Strategy (Stratégie)**
Objectif	Faire varier le comportement selon l'état interne d'un objet.	Permettre de choisir dynamiquement un algorithme parmi plusieurs.
Changement	Automatique, déclenché par le contexte (l'objet change d'état).	Contrôlé par le client (on choisit la stratégie à appliquer).
Exemple	Un téléphone qui change de comportement selon qu'il est éteint, verrouillé ou déverrouillé.	Un compresseur qui peut utiliser différents algorithmes de compression (ZIP, RAR, 7z).

Conclusion

Le pattern État permet de rendre le comportement d'un objet flexible et évolutif, en encapsulant la logique de chaque état dans des classes distinctes.

Il améliore la lisibilité, la maintenabilité et respecte pleinement les principes SOLID, tout en évitant la complexité des structures conditionnelles classiques.