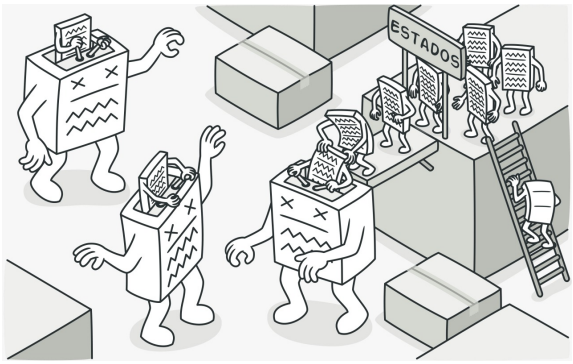
Patrón State

¿Qué es?

Es un patrón de diseño que permite al objeto cambiar su comportamiento cuando su estado interno cambia. Para entenderlo, es como si el objeto cambiara su clase.



Este patrón está relacionado con el concepto de máquina de estados finitos.

En vez de hacer en una misma clase con diferentes situaciones, lo que se hace es, en cada caso se llama a una clase diferentes. Como lo llamaría Tomás “Un Switch de clases”.

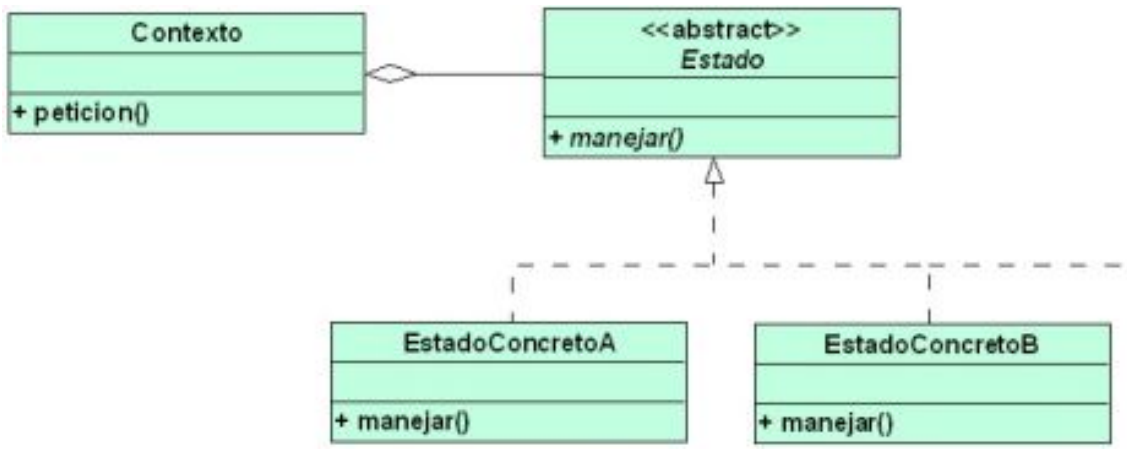
El patrón extrae comportamientos relacionados con el estado, los coloca dentro de clases de estado separadas y fuerza al objeto original a delegar el trabajo de una instancia de esas clases, en lugar de actuar por su cuenta. El patrón sugiere que extraigas todo el código específico del estado y lo metas dentro de un grupo de clases específicas. Como resultado, puedes añadir nuevos estados o cambiar los existentes independientemente entre sí, reduciendo el costo de mantenimiento.

Este patrón resulta útil cuando necesitamos que un objeto se comporte de forma diferente dependiendo del estado interno en el que se encuentre en cada momento. Como por ejemplo los botones e interruptores de un móvil q se comportan de diferentes forma dependiendo del estado actual del smartphone:

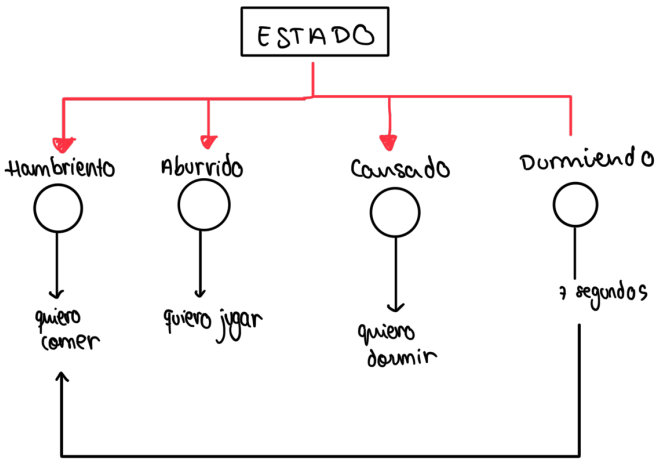
- Cuando el teléfono esta desbloqueado, al pisar el botón se ejecutan varias funciones
- Cuando el tlf esta bloqueado, se desbloquea la pantalla
- Cuando batería de teléfono está baja, pulsar botón q muestre la pantalla de la carga.

Lenguaje UML nuestro código

Según el siguiente diagrama en UML podemos apreciar que el objeto cuyo estado es susceptible de cambiar (Contexto) contendrá una referencia a otro objeto que define los distintos tipos de estado en que se puede encontrar.



Código de ejemplo



Ventajas y desventajas

VENTAJAS
El patrón State está motivado por aquellos objetos en que, según su estado actual, varía su comportamiento ante los diferentes mensajes.

- Principio de responsabilidad única. Organiza el código relacionado con estados particulares en clases separadas.
- Principio de abierto/cerrado. Introduce nuevos estados sin cambiar clases de estado existentes o la clase contexto.
- Simplifica el código del contexto eliminando voluminosos condicionales de máquina de estados.

INCONVENIENTES
- Aplicar el patrón puede resultar excesivo si una máquina de estados sólo tiene unos pocos estados o raramente cambia.

- Resulta complejo administrar los diferentes comportamientos si los estados son muy parecidos.

- Puede resultar difícil el mantenimiento del código.

- A nivel de código, puede ocurrir que se escriban estados similares y que resulte incongruente cuando la complejidad del patrón es menor. Cuanto más extenso, más complicado es su implementación, aunque fuese lo mejor para la solución del problema.

Bibliografía:

https://refactoring.guru/es/design-patterns/state
https://informaticapc.com/patrones-de-diseno/state.php
https://refactoring.guru/es/design-patterns/state/python/example

Juan García A.
Tomás Machín
Paloma Pérez de Madrid