Patrón
State

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

1. Motivación
2. Diagrama UML

1 | Motivación

Vas a aprender una manera de hacer que los objetos cambien según su estado.

Propósito

Cuando se requiere que un objeto tenga diferentes comportamientos según el estado en que se encuentra, resulta complicado poder manejar el cambio de comportamientos y los estados de dicho objeto.

El patrón *State* propone una buena solución a esta complicación, creando básicamente un objeto por cada estado posible del objeto que lo invoca.

Solución

Se implementa una clase para cada estado diferente del objeto y cada clase implementará los métodos cuyo comportamiento varía según ese estado.

Así, siempre se tendrá una referencia a un estado concreto y se comunicará con este para resolver sus responsabilidades.

Ventajas y desventajas

Ventajas:
Se localizan fácilmente las responsabilidades de los estados específicos ya que se encuentran en las clases que corresponden a cada estado. Esto brinda una mayor claridad en el desarrollo y el mantenimiento posterior.

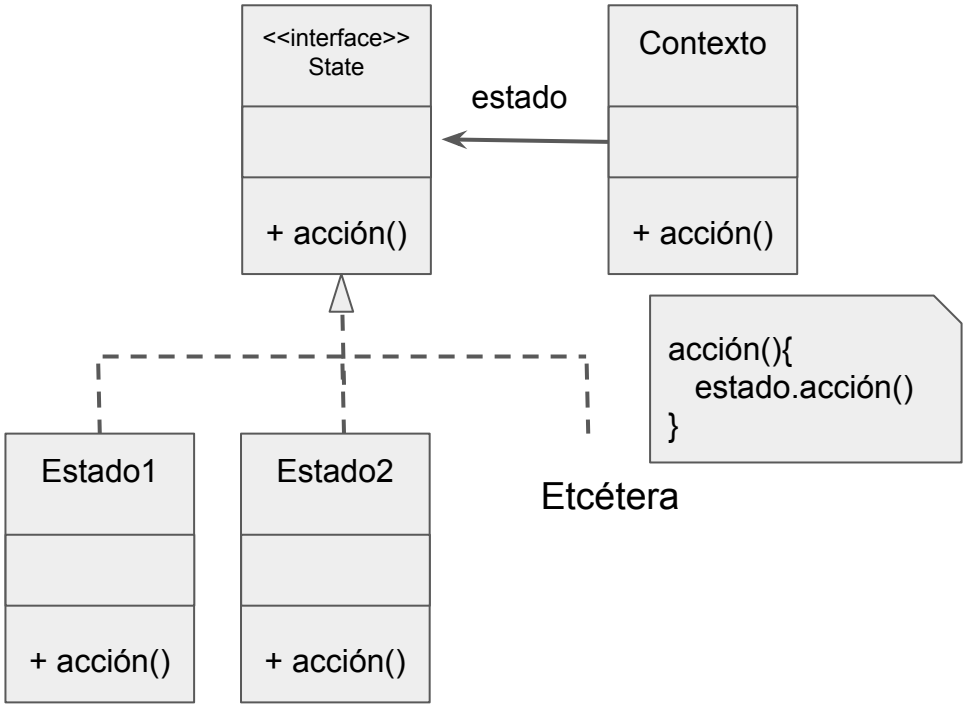
- Hace los cambios de estado explícitos al estar representado cada estado en una clase.
- Facilita la ampliación de estados.
- Permite a un objeto cambiar de clase en tiempo de ejecución dado que al cambiar sus responsabilidades por las de otro objeto de otra clase, la herencia y responsabilidades del primero han cambiado por las del segundo.

Desventaja:

- Se incrementa el número de subclases.

2 | Diagrama UML

Diagrama solución patrón *State*



- **Clase contexto:** define la interfaz con el cliente. La instancia de contexto es la que define su estado actual.
- **Interface *State* (estado):** interface para el encapsulamiento de la responsabilidades asociadas con un estado particular de contexto. Define las responsabilidades de cada estado.
- **Clase estado:** cada una implementa el comportamiento o responsabilidad de contexto.

¿Cómo funciona?

La clase **contexto** envía mensajes al objeto dentro de su código que contiene una instancia de **estado** para brindarle a estos la responsabilidad que debe cumplir el objeto **contexto**. Así, el objeto **contexto** va cambiando las responsabilidades según el estado en que se encuentre, puesto que también cambia instancia de **estado** al hacer un cambio de estado.

Dicho en pocas palabras:
Contexto le dice a la instancia de **estado** que haga la **acción…**.. Pero cuando cambia la instancia de la clase **estado** (**Estado1**, **Estado2**, etc) la **acción** se realiza de forma diferente según este.

Conclusiones

El patrón no indica exactamente dónde definir las transiciones de un estado a otro. Existen dos formas de solucionar esto:

1. Definiendo estas transiciones dentro de la clase contexto.
2. Definiendo estas transiciones en las subclases de *State*.

Es más conveniente utilizar la primera solución cuando el criterio a aplicar es fijo, es decir, no se modificará. En cambio, la segunda resulta conveniente cuando este criterio es dinámico.Este se presenta en la dependencia de código entre las subclases.
También hay que evaluar en la implementación cuándo crear instancias de estado concreto distintas o utilizar la misma instancia compartida. Esto dependerá si el cambio de estado es menos o más frecuente respectivamente.


DigitalHouse>
Coding School