Patrón Strategy

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

1. Motivación
2. Diagrama UML

1 | Motivación

“

Vamos a separar los diferentes comportamientos de una clase y así cambiar la estrategia para cuando sea preciso.

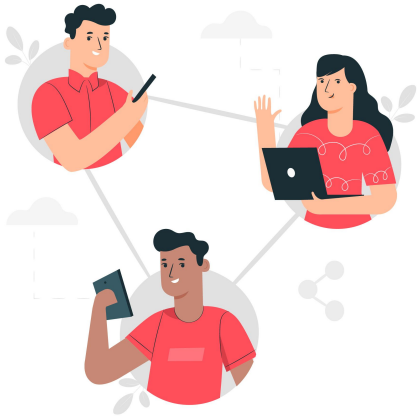
”

Propósito

Un determinado objeto va a tener un comportamiento que puede ser simple y siempre el mismo, pero a veces este comportamiento se vuelve más complejo y de acuerdo con las necesidades, cambiante.

El patrón Strategy hace que los algoritmos varíen independientemente del cliente que los esté usando.

Propone una solución simple basada en un objeto que cambia y cuyo comportamiento es el que se adapta a la circunstancia.



Solución

Se dispone de una interface, **Estrategia** (Strategy), que sirve para establecer las firmas de los métodos que vamos a hacer que cambien y las diferentes clases que implementan esta interface, **Estrategias Concretas** (ConcreteStrategy), que son las que tienen las diferentes algoritmos para realizar la tarea.

Podremos ver algún caso donde queremos que la **Estrategia** también tenga algún atributo que sea común a cada **Estrategia concreta**. En este caso la **Estrategia** será una clase abstracta y las **Estrategias concretas** serán heredadas.

Ventajas y desventajas

✓	El uso del patrón proporciona una alternativa a la herencias de clases, ya que puede realizarse un cambio dinámico de estrategia.
✓	Si un algoritmo utiliza información que no deberían conocer los clientes, la utilización del patrón Strategy evita la exposición de dichas estructuras.
✓	Este patrón de diseño nos sirve para intercambiar un sin número de estrategias posibles.
✗	Aumenta el número de objetos creados, por lo que se produce una aumento en el intercambio de objetos entre estrategia y contexto

2 | Diagrama UML

Diagrama Solución Patrón Strategy

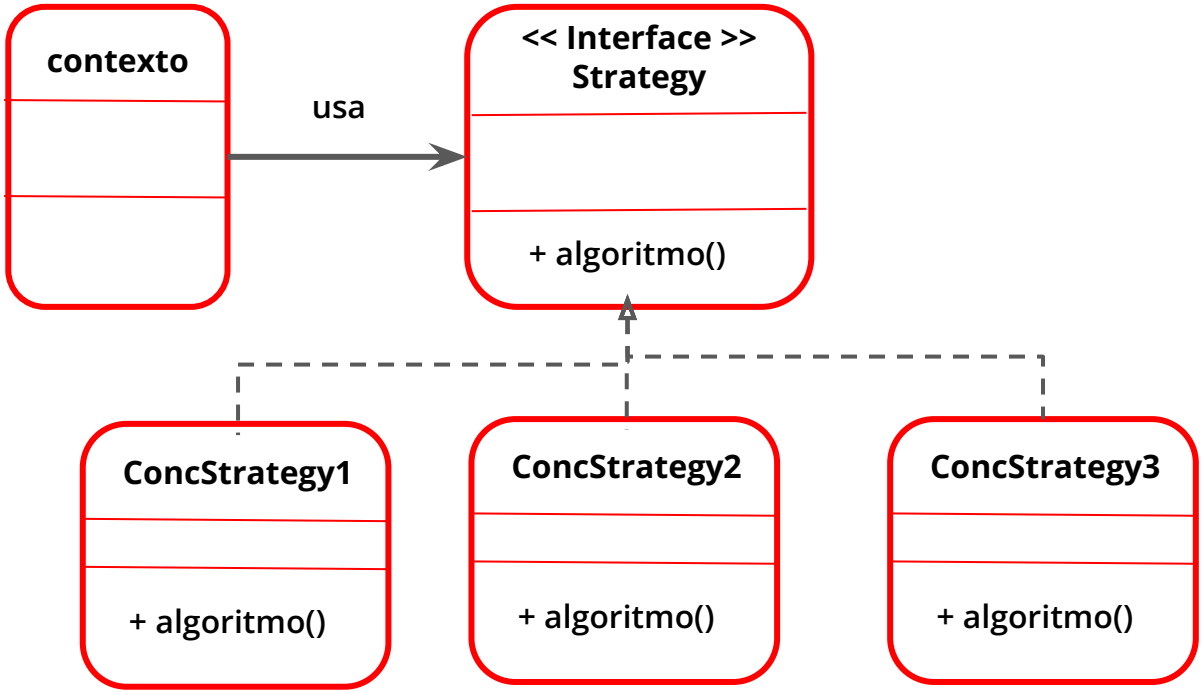


Diagrama Solución Patrón Strategy

- **Contexto:** es el elemento que usa los algoritmos, delega en la jerarquía de estrategias. Configura una estrategia concreta mediante una referencia a la estrategia necesaria y puede, por lo tanto, cambiar la estrategia de acuerdo a sus necesidades.
- **Interface Strategy:** declara una interface común para todos los algoritmos soportados. Esta interface será usada por el contexto para invocar a la estrategia concreta. También puede usarse una clase de tipo Abstract en caso de precisar variables o métodos comunes a las diferentes estrategias.
- **ConcStrategy:** implementa el algoritmo utilizando la interfaz definida por la estrategia. Sirve para tener en esta clase una lógica que se puede reutilizar y aislada del contexto.

¿Cómo funciona?

Cuando el **contexto** quiere utilizar algún algoritmo concreto primero setea la variable (que tendrá un objeto que implemente la interface Strategy) con un valor adecuado. Luego invoca el método que desea ejecutar. El que recibe esta invocación será un objeto **EstrategiaConcreta** que lo ejecuta con el algoritmo propio.


DigitalHouse>
Coding School