Strategy

Patrón de Diseño

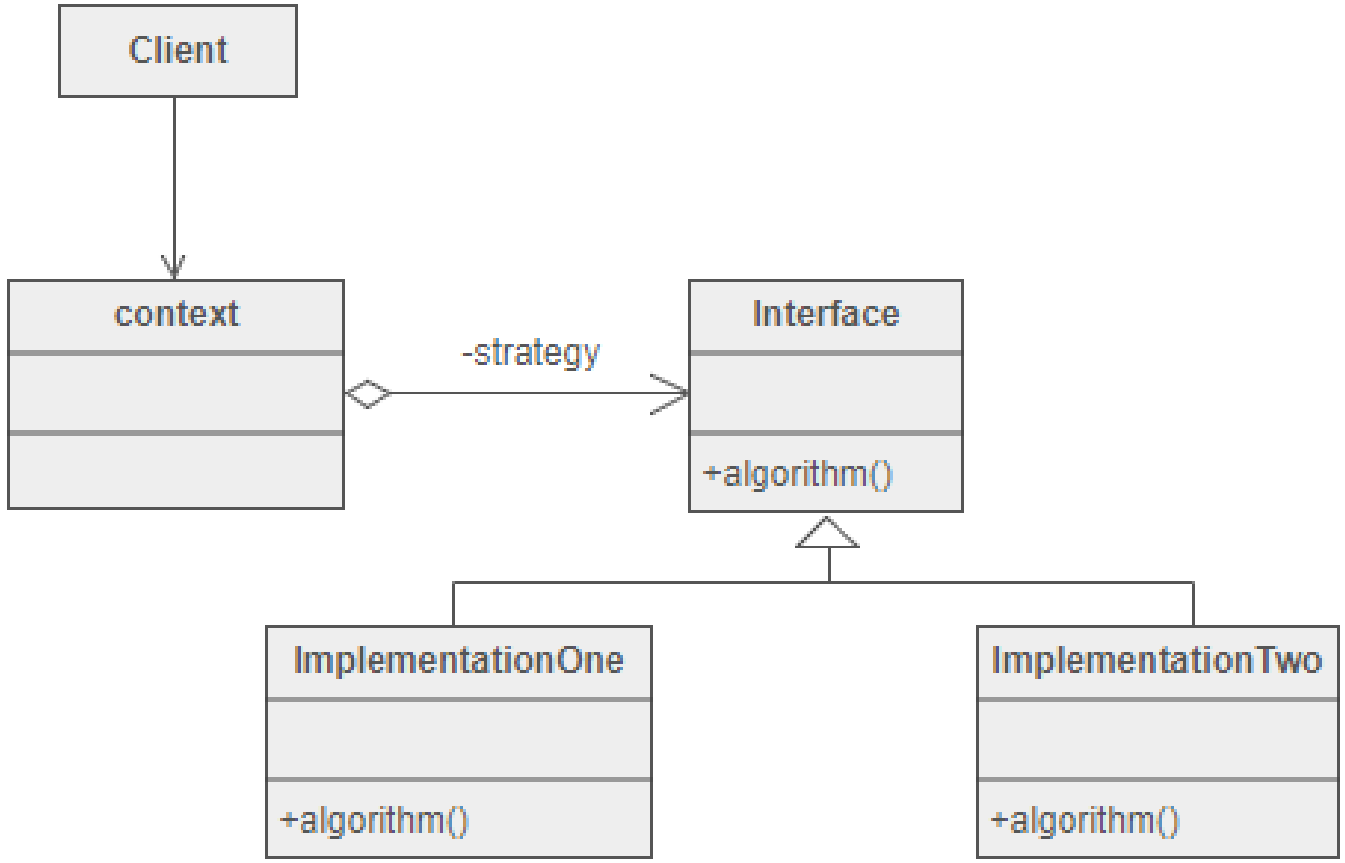
Descripción

- **Clasificación:** De comportamiento (behavioral).
- **Intención:**

1. Definir una familia de algoritmos, encapsularlos y hacerlos intercambiables.
2. Capturar la abstracción en un interfaz y ocultar la implementación, entre otros detalles, en clases derivadas.

- **Otros nombres:** Policy pattern.
- **Motivación:** Siguiendo el “open-closed principle”, buscan que el cliente pueda usar una interfaz y olvidarse de la molestia de modificar las clases derivadas directamente.
- **Aplicación:** Situaciones con diferentes maneras de llegar a un mismo objetivo.

Estructura


Client
context
-strategy
Interface
+algorithm()
ImplementationOne
+algorithm()
ImplementationTwo
+algorithm()

Descripción

- **Participantes:** Cliente, clase contexto, interfaz, algoritmos.
- **Colaboraciones:** El cliente interactúa con la clase contexto que está ligada a la interfaz. Esta hereda los algoritmos a utilizar por el cliente listos para utilizarse.
- **Consecuencias:**

- Pros:

- Previene el uso de muchos condicionales (if, else, switch).
- Puedes cambiar los algoritmos sin tener que modificar el contexto.
- Escalable.

-Contras:

- Los clientes deben saber de antemano los algoritmos implementados para hacer un uso adecuado.

¿Qué debo tomar en cuenta antes de implementarlo?

Siendo que uno puede agregar los algoritmos que desee, se debe cuidar la cantidad de objetos en la aplicación y procurar en medida de lo posible usar solamente los que son absolutamente necesarios.

¿Dónde se aplica este patrón de diseño?

Algunas aplicaciones conocidas son las siguientes:

- Ordenamiento (sorting): Cambio de algoritmo según se requiera.
- Encriptación: Diferentes algoritmos dependiendo del tamaño de archivo.
- Juegos: Por ejemplo, en un RPG donde n clases tienen un método ataque() pero cada una de estas lo ejecuta de manera distinta.
- Validaciones: Checar objetos conforme a ciertas reglas y tener la posibilidad de añadir más si se requieren.

Patrones relacionados

- Strategy -> Factory
- State -> Abstract Factory