# Investigación patrón de diseño

Patron de diseño: “Strategy”.  
Tipo: “Comportamiento”.  
  
El patron de diseño “strategy” es un patron del tipo de comportamiento, que su funcion permite al usuario interrumpir un flujo del sistema, con el objetivo de ser el cliente quien decida la nueva ruta del flujo, con la idea de poder seleccionar un algoritmo distinto, diferentes entidades o simplemente seleccionar el escenario deseado.  
  
Su uso puede verse en muchos escenarios en diferentes sistemas, como ser dentro de calculadoras avanzadas o resolutores de problemas, donde actualmente se tiene implementado un pequeño espacio donde el usuario puede seleccionar el tipo de metodologia o estrategia que se tomara para poder resolver una consigana matematica.  
Otra opcion puede ser en el aparatdo de pago con distintos metodos de pagos, en donde el usuario puede tener la libertad de elegir el metodo de pago que mejor se ajuste a su transaccion, definiendo de esa forma un flujo completamente diferente dentro del sistema dependiendo de la opcion que elija.  
  
Diagrama de caso: “Metodos de pago”  
  
  
  
  
  
  
  
  
  
  
  
Pseudocodigo:  
entrada\_usuario = “Tarjeta” // u otro metodo de pago

**RuteStrategy**

**Presentacion**

**Cliente**

**Tarjeta**

**Efectivo**

**QR**

**IMetodoPago**

Rute\_strategy = new Rute\_strategy()

Return Rute\_strategy.MetodoPago(entrada\_usuario)

/////////////////////////////////////////////////////////////////

// dentro de rute\_strategy

IMetodoPago MetodoPago(entrada){

Flujo\_definido = ListadeMetodos[entrada]

Return Flujo\_definido.disp ? “cobrando por ….” : “metodo no disponible”

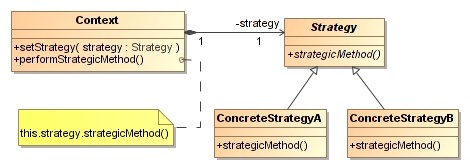
}

El **patrón Strategy** es similar a otros patrones de diseño que permiten cambiar el comportamiento de un objeto de manera flexible. Aquí hay algunos patrones que se parecen o tienen relación con Strategy:

1. **State**:
   * Se parece a Strategy porque también encapsula comportamientos en clases separadas.
   * La diferencia clave es que **State** representa estados de un objeto y permite transiciones entre ellos, mientras que **Strategy** encapsula algoritmos intercambiables sin una noción de estado.
2. **Command**:
   * Ambos permiten encapsular lógica en clases separadas.
   * **Command** encapsula una acción para ser ejecutada más tarde, mientras que **Strategy** encapsula un algoritmo para ser ejecutado inmediatamente.
3. **Template Method**:
   * Ambos permiten modificar el comportamiento sin alterar la estructura general.
   * **Template Method** define el esqueleto de un algoritmo en una clase base y permite que las subclases implementen detalles específicos, mientras que **Strategy** permite intercambiar algoritmos completos en tiempo de ejecución.
4. **Decorator**:
   * Ambos permiten modificar el comportamiento de un objeto sin alterar su estructura base.
   * **Decorator** añade funcionalidad envolviendo objetos dinámicamente, mientras que **Strategy** permite seleccionar diferentes algoritmos en tiempo de ejecución.

Otras formas de aplicar el patron strategy según fuentes:  
Diagrama

El contenido generado por IA puede ser incorrecto.

  
  
  
FUENTES:   
Wikipedia. (s.f.). *Strategy (patrón de diseño)*. Wikipedia, la enciclopedia libre. Recuperado el 30 de marzo de 2025, de <https://es.wikipedia.org/wiki/Strategy_(patr%C3%B3n_de_dise%C3%B1o)>

Refactoring Guru. (s.f.). *Patrón de diseño Strategy*. Refactoring Guru. Recuperado el 30 de marzo de 2025, de <https://refactoring.guru/es/design-patterns/strategy>