

Programación II

Patron de diseño Strategy

Ricardo Enrique Reyes Lozano

2018130213

Kenneth Vittetoe

9/12/2019

Strategy

El patrón Estrategia / Strategy es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina como se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea.

**Los distintos componentes de este patrón son**

* Interfaz Strategy: será aquella interfaz que define el nombre del método o métodos que conformarán la estrategia.
* Clases Strategy concretas: todas aquellas clases que implementen la interfaz Strategy dando forma al algoritmo.
* Contexto: elemento donde se desarrollará la estrategia.

**Tipo**

Comportamiento, a nivel de objetos.

**Propósito**

Permite a los algoritmos variar con independencia de los clientes que los usan. El propósito de un patrón de diseño es capturar el conocimiento de un diseño de software y hacerlo reusable.

**¿Cuándo usarlo?**

* Cuando muchas clases relacionadas sólo se diferencian en su comportamiento.
* Cuando se necesitan diferentes variantes de un algoritmo.
* Cuando un algoritmo usa datos que los clientes no tienen por qué conocer.
* Cuando una clase define muchos comportamientos, los cuales se manifiestan como definiciones condicionales múltiples de sus operaciones.

**Implementación**

Las posibles estrategias se ejecutan dentro de un objeto de contexto que se encarga de recuperar la estrategia apropiada para el cliente. Cada una de las estrategias implementa una interfaz que define la firma del método de la estrategia.

**Ventajas**

* Las familias jerárquicas de estrategias definen algoritmos y comportamientos que enfatizan la reutilización. La herencia puede ayudar a sacar factor común a la funcionalidad de los algoritmos.
* El encapsulamiento de algoritmos en clases separadas ofrece una ventajosa alternativa a la especialización por herencia del contexto para obtener un comportamiento diferente, que promueve la independencia, la facilidad de entender el diseño y la posibilidad de futuras extensiones.
* Se eliminan las costosas definiciones de comportamiento multicondicionales.
* Se posibilita ofrecer diferentes implementaciones del mismo comportamiento, en función de restricciones como el espacio en memoria o el tiempo de respuesta.

**Inconvenientes**

* Los clientes deben tener un cierto conocimiento de cada estrategia, para así poder elegir en cada situación cual es la más apropiada.
* Dado que todas las estrategias comportan una interfaz común, si las diferencias entre ellas son grandes, es probable que mucha de la información que se les pasa no sea de utilidad más que a las más complejas.
* Puede producirse una gran explosión en el número de objetos del sistema.