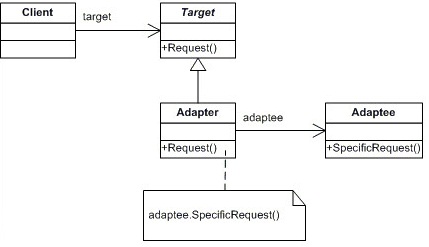
**-- PATRON ADAPTADOR--**

Nos sirve cuando tenemos una clase con una interfaz definida que queremos poder utilizar a través de la interfaz definida por otra clase. Nos permite que dos clases cooperen de manera distinta a la que, en principio, iban a funcionar.

Debe aplicarse cuando queremos usar una clase existente a través de la interfaz de una clase distinta. También cuando queramos crear una clase que sea reutilizable y además, coopere con otras clases con las que no estaba cooperando.

Puede haber adaptadores de clases y de objetos. Los de objetos son los que usan varias de las subclases que ya existen para hacer la adaptación.

**Adaptadores de clases**

Ventajas:

* Redefinen parte del comportamiento del objeto que estamos adaptando.
* Introducimos un único objeto para hacer la adaptación.

Inconvenientes:

* Solo podemos adaptar una clase.

**Adaptadores de objetos**

Ventajas:

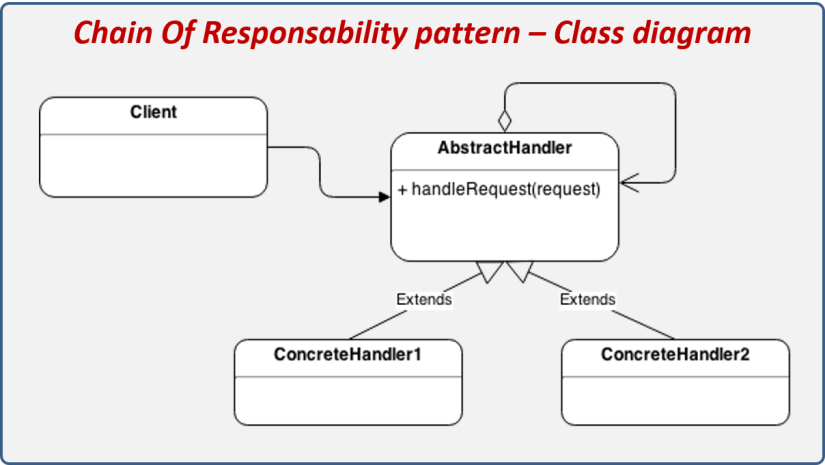
* Permite que un adaptador funciones con muchas subclases.

Inconvenientes:

* El objeto adaptador necesita vincularse a todas las clases que pudieran aparecer dentro del adaptable.

**-- PATRON CHAINS OF RESPONSABILITY--**

Su propósito es desacoplar al emisor y al receptor de una petición. Por ejemplo, en una ayuda contextual de una interfaz gráfica, no necesitamos saber cuál es el contexto concreto donde vamos a dar la ayuda. Quien necesita la ayuda, la solicita y los objetos de la interfaz van retransmitiendo esa petición jerárquicamente hasta que alguno es capaz de darla y se nos muestra la ayuda contextual.

La aplicamos cuando haya más de un objeto que pueda manejar una petición y no sabemos cuál de ellos va a manejar esa petición.

Ventajas:

* Reduce acoplamiento entre clases.
* Añade flexibilidad para asignar responsabilidades a objetos.

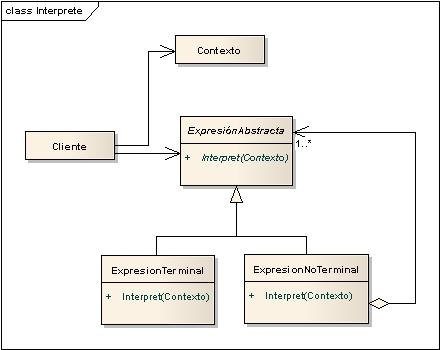
Inconvenientes:

* No se garantiza que la petición se reciba.

**-- PATRON INTÉRPRETE--**

Se utilizar para interpretar un lenguaje sencillo para poder dar flexibilidad a las estructuras que estamos reconociendo.

Lo aplicaremos cuando haya un lenguaje que podamos interpretar. Para ello, el lenguaje debe ser simple y que además la eficiencia no sea una preocupación critica.



Ventajas:

* Es sencillo ampliar y cambiar gramática.
* Es fácil implementar la gramática.
* No resulta muy difícil añadir modos de interpretación de expresiones.

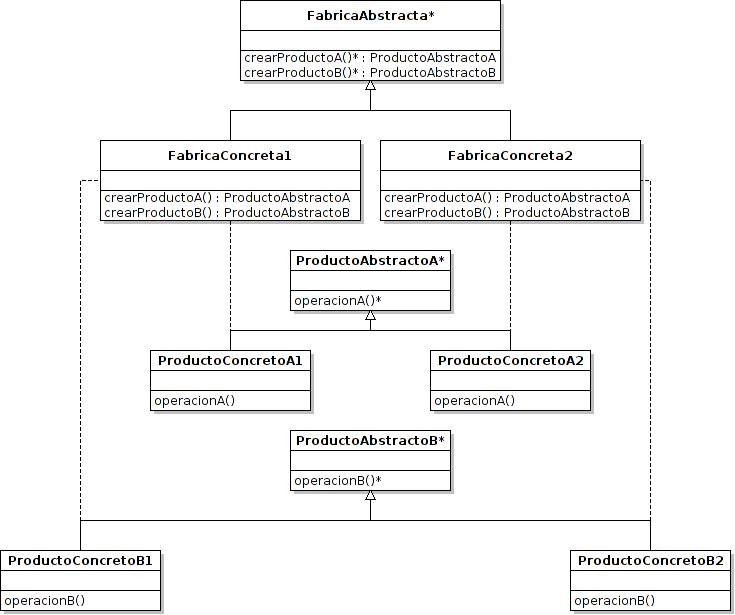
Inconvenientes:

* Las gramáticas complejas son difíciles de mantener.

**-- PATRON FÁBRICA ABSTRACTA--**

Proporciona la manera de crear objetos compuestos por familias de otros objetos que tienen una interdependencia entre sí.

La aplicaremos en estos casos:

* Cuando un sistema tenga que ser independiente de parte de los objetos que tiene ese sistema.
* Cuando podamos configurar un sistema con una familia de productos a elegir entre varias familias.
* Cuando tengamos una serie de objetos que tienen que trabajar juntos pero se pueden elegir distintas opciones para crear cada uno de esos elementos.

Ventajas:

* Aísla clases concretas de clientes.
* Facilita intercambio de familias de productos.
* Promueve la consistencia entre productos.

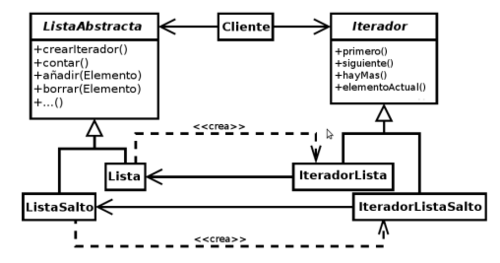
Inconvenientes:

* Complicado añadir nuevos tipos de objetos que tenemos que crear.

**-- PATRON ITERADOR--**

Nos informa de una nueva forma de acceder secuencialmente a los elementos de un objeto agregado sin necesidad de representación externa. Lo usamos cuando queremos recorrer una colección de objetos de la forma más adecuada mediante iteraciones.

Lo aplicaremos en los siguientes casos:

* Cuando queramos acceder al contenido de un objeto sin exponer su representación.
* Cuando queramos recorrer ese objeto simultáneamente y de distintas maneras.
* Cuando queramos darle a diferentes estructuras de datos la misma interfaz.

Ventajas:

* Permite hacer diferentes recorridos.
* Simplifica interfaces.
* Se puede hacer más de un recorrido a la vez.

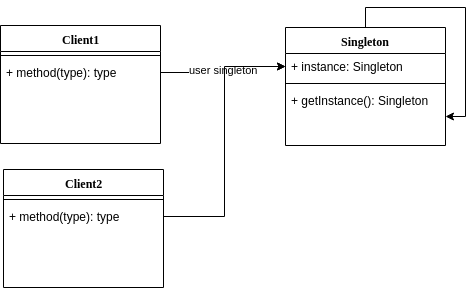
**-- PATRON SINGLETON--**

Garantiza que una clase tiene una única instancia y queremos proporcionar un método que nos sirva de acceso único a esa instancia.

La manera de garantizar que solo existe una instancia y que solo hay una forma de acceder a ella es que sea la propia clase la responsable de crear la instancia y de dar acceso a la misma.

Lo aplicamos:

* Necesitemos exactamente una instancia de la clase.
* La instancia sea extensible mediante herencia y que los clientes que usan el singleton sean capaces de usar esa instancia sin modificar su código.

Ventajas:

* Acceso controlado a la instancia.
* Permite tener número variable de instancias.
* Más flexible que las operaciones de clases estáticas.
* Permite refinar operaciones.