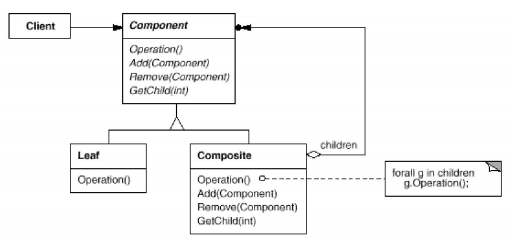
Todos estos patrones definidos a continuación, son patrones estructurales de patrones GoF, antes explicados en otros documentos. Los patrones que vamos a definir son: patrón composición, patrón decorador, patrón comando y patrón estado.

**-- PATRON COMPOSICION--**

Sirve para determinar la estructura compuesta de una parte de nuestra aplicación, en este caso, un ajedrez. Permite construir estructuras de forma jerárquica y construir interfaces gráficas. Por ejemplo, para nuestra app, serían las imágenes de las fichas de los peones, alfiles, etc.

Debe aplicarse cuando se quiera representar una jerarquía de objetos de nuestra aplicación.

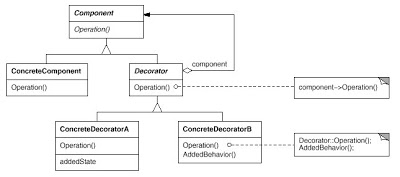
Este patrón tiene ventajas e inconvenientes:

Ventajas: define jerarquías, permite tratamiento uniforme de objetos y facilita añadir nuevos tipos de componentes.

Inconvenientes: oculta el tipo de los objetos dentro del compuesto.

**-- PATRON DECORADOR--**

Su propósito es asignar responsabilidades a un objeto y no a la clase entera de manera flexible y dinámicamente. Se aplica cuando hay que añadir objetos individuales de forma dinámica sin afectar a otros objetos. También se aplica cuando la extensión no es viable mediante herencia debido a superposición de subclases.



Sus ventajas son:

* Más flexible que la herencia
* Evita clases cargadas con funciones

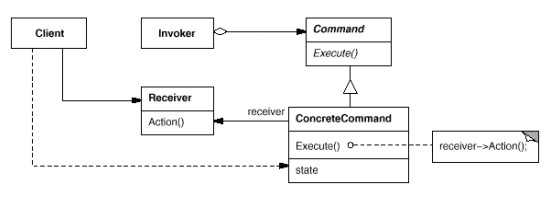
Sus inconvenientes son:

* Un decorador y su componente no son idénticos.
* Aparecen muchos objetos pequeños.

**-- PATRON COMANDO--**

Su propósito es encapsular una acción sobre un objeto de manera que el lugar desde donde se envía la acción y el objeto quedan separados.

Es necesario para enviar peticiones a objetos sin saber nada acerca de la operación solicitada o quien es el que manda esa operación.

Se debe aplicar cuando se quiere especificar y ejecutar peticiones en diferentes situaciones. También cuando se quiere añadir una opción de deshacer acciones.

Su estructura es:

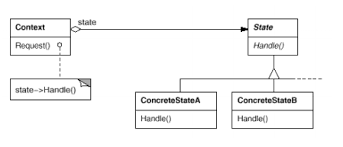
Sus ventajas son:

* Se pueden ensamblar órdenes simples en una orden compuesta.
* Es fácil añadir nuevas órdenes.
* Desacopla el objeto que invoca la acción y el objeto receptor.

**-- PATRON ESTADO--**

Permite que un objeto modifique su comportamiento en función de su estado, Por ejemplo, una conexión de red, el estado puede ser: establecida, escuchando o cerrada. Entonces en función de ese estado se responde al cliente de diferentes maneras.

Debe aplicarse cuando el comportamiento de un objeto depende de su estado y debe cambiar el tiempo de ejecución en función de ese estado.

Su estructura es:

Tiene diferentes ventajas:

* Localiza el comportamiento dependiente del estado y divide dicho comportamiento en diferentes estados.
* Los objetos estado pueden compartirse