Resumen sobre patrones de diseño*

Patrón

Un patrón es una descripción del problema y la esencia de su solución, que se puede reutilizar en casos distintos.

Categorías de patrones:

- De creación: implica el proceso de instanciar objetos.
- Estructurales: composición de objetos.
- De comportamiento: cómo se comunican los objetos, cooperan y distribuyen las responsabilidades para lograr sus objetivos.

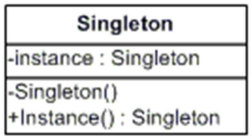
De creación:

Singleton

Asegura que una clase tiene una única instancia y proporciona un punto de acceso global a dicha instancia.

Aplicabilidad.

- Cuando debe haber una sola instancia, y debe ser accesible a los clientes desde un punto de acceso conocido.
- Cuando la única instancia debe ser extensible mediante subclasificación, y los clientes deben ser capaces de usar una instancia extendida sin modificar su código.

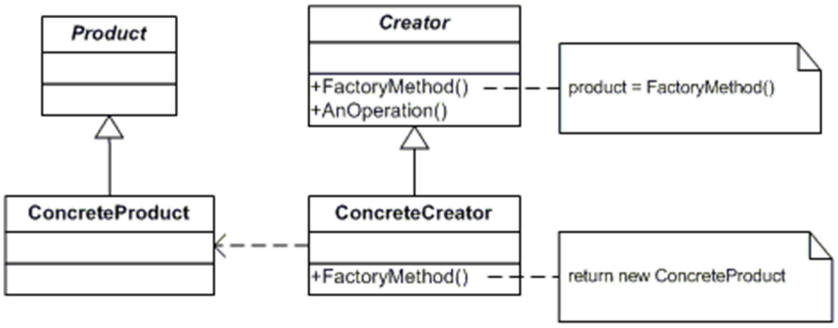


Factory Method

Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

Aplicabilidad

- Una clase no puede prever la clase de los objetos que tiene que crear.
- Una clase quiere que sean sus subclases las que decidan qué objetos crean.
- Las clases delegan la responsabilidad en una de entre varias clases auxiliares, y queremos localizar qué subclase concreta es en la que se delega.

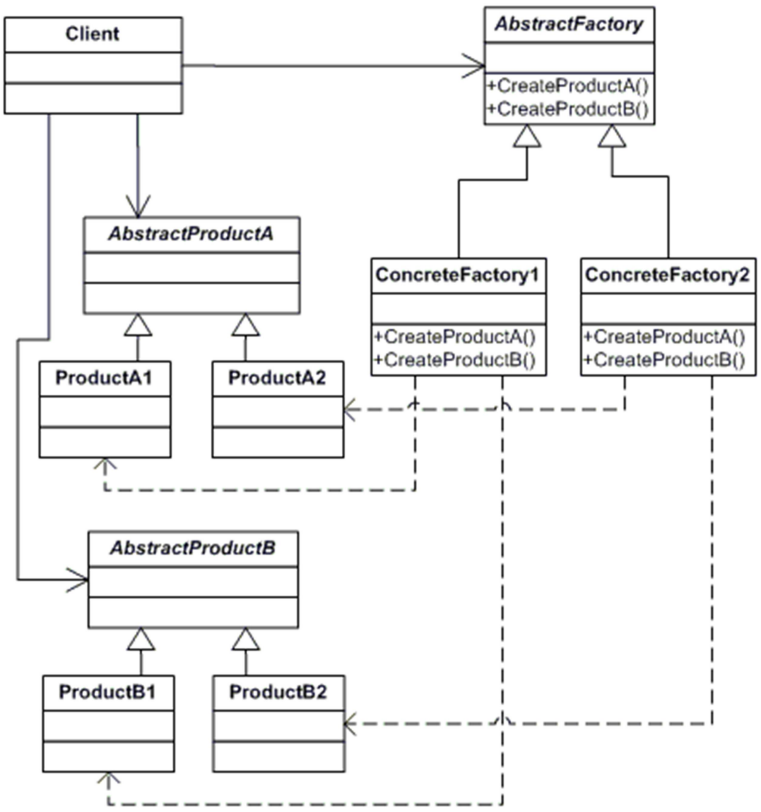


Abstract Factory

Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

Aplicabilidad.

- un sistema que deba ser independiente de cómo se crean, componen y representan sus productos.
- un sistema que deba ser configurado con una familia de productos entre varias.
- una familia de objetos producto relacionados que está diseñada para ser usada conjuntamente, y es necesario hacer cumplir esta restricción.
- se quiere proporcionar una biblioteca de clases productos, y sólo se quiere revelar sus interfaces, no su implementación.



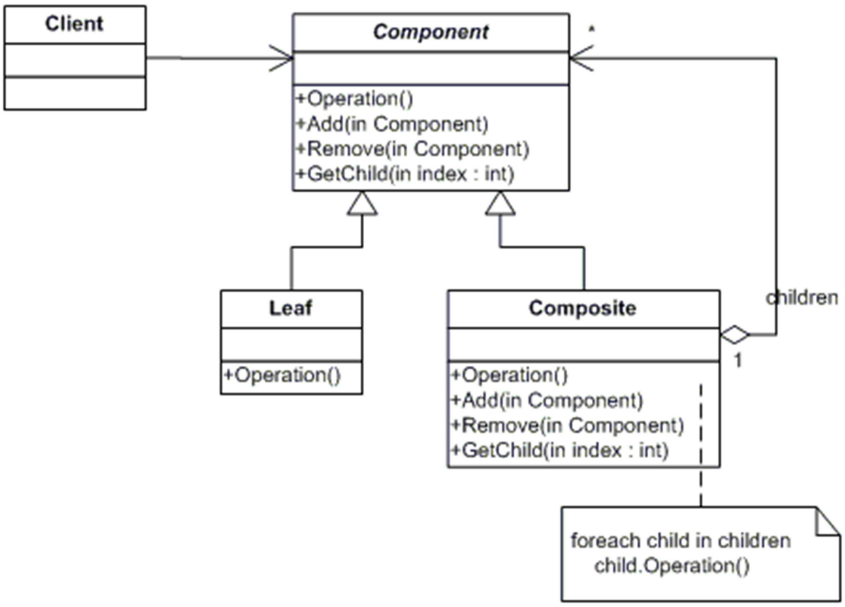
Estructurales

Composite

Componer objetos en estructuras arborescentes para representar jerarquías parte-conjunto.

Aplicabilidad.

- Se quieren representar jerarquías todo/parte de objetos.
- Se quiere que los clientes ignoren la diferencia entre composiciones de objetos y objetos individuales. Los clientes tratarán todos los objetos en la estructura compuesta de manera uniforme.

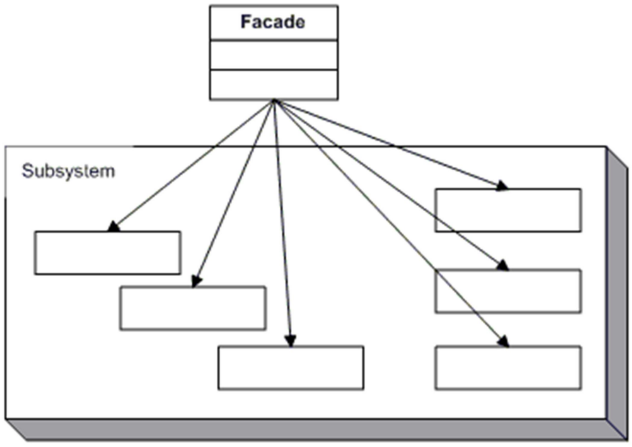


Facade

- Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema.
- Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

Aplicabilidad.

- Queramos proporcionar una interfaz simple a un subsistema complejo. Sólo los clientes que necesitan más personalización necesitarán ir más allá de la fachada.
- Haya muchas dependencias entre los clientes y las clases que implementan una abstracción. La fachada desacopla el subsistema de sus clientes y otros subsistemas (mejora acoplamiento y portabilidad).
- Queramos dividir en capas nuestros subsistemas. La fachada es el punto de entrada a cada subsistema.

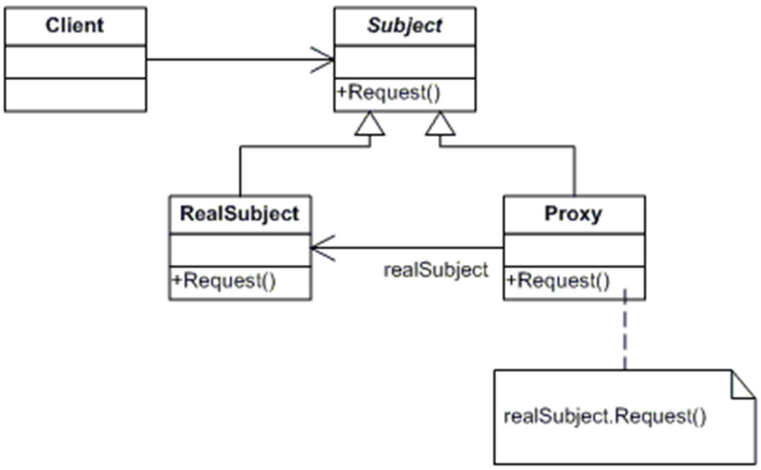


Proxy

Proporcionar un representante o substituto de otro objeto para controlar el acceso a este.

Aplicabilidad.

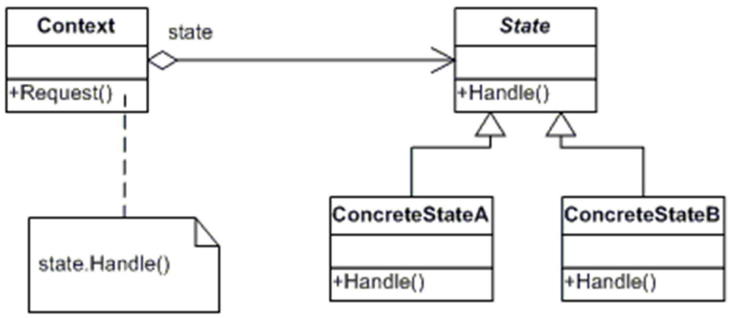
- Cuando hay necesidad de una referencia a un objeto más flexible o sofisticada que un puntero.
- Proxy Remoto: un representante para un objeto que se encuentra en otro espacio de direcciones.
- Proxy Virtual: Crea objetos costosos por encargo (ej.:ImageProxy).
- Proxy de Protección: Controla el acceso al objeto original(permisos de acceso).



De comportamiento

State

Permitir que un objeto modifique su comportamiento cada vez que cambie su estado interno.

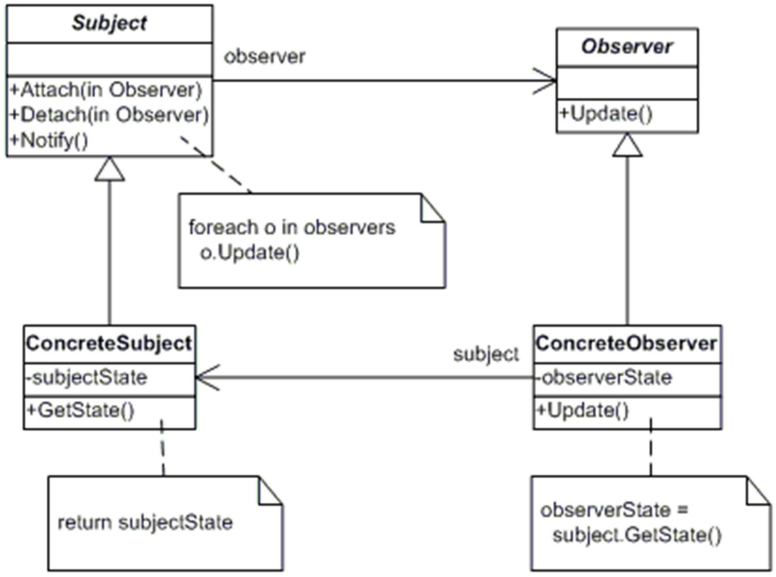


Observer

Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

Aplicabilidad.

- Cuando una abstracción tiene dos aspectos y uno depende del otro.
- Cuando un cambio en un objeto requiere cambiar otros, y no sabemos cuántos hay que cambiar.
- Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quiénes son dichos objetos (esto es, no queremos que los objetos estén fuertemente acoplados).



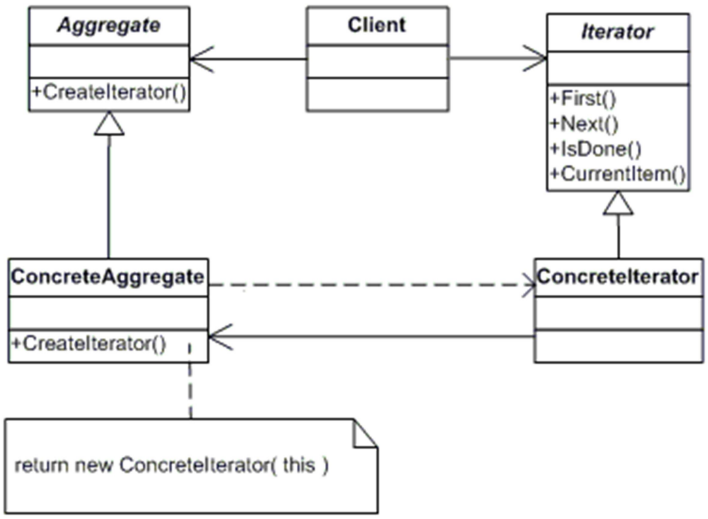
Iterator

Proporciona un medio de acceder a los elementos de un contenedor secuencialmente sin exponer su representación interna.

Aplicabilidad.

- para acceder al contenido de un contenedor sin exponer su representación interna.

- para permitir varios recorridos sobre contenedores.
- para proporcionar una interfaz uniforme para recorres distintos tipos de contenedores (esto es, permitir la iteración polimórfica).



Referencias

* El presente documento fue elaborado como resumen de diversas fuentes y no pretende ser una revisión exhaustiva de todos los patrones existentes, para mayor información puede continuar su búsqueda en:

http://www.dofactory.com/Patterns/Patterns.aspx
http://sourcemaking.com/design_patterns