Patrones de diseño

Humberto Cervantes Maceda

Noviembre 2005

Patrones de diseño

- **Inspirados de una idea postulada por un arquitecto llamado Christopher Alexander**
  - “Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe la esencia de la solución a ese problema, de tal forma que se puede recurrir a esta solución un millon de veces, sin jamas hacer las cosas de la misma manera”
- **Patrones de diseño en Orientación a Objetos**
  - Descripciones de objetos que se comunican y clases que están configuradas para resolver un problema general de diseño en un contexto particular.
  - Design patters: Elements of reusable OO Software
    - GoF: Gamma, Helm, Johnson, Vlissides (1995)

Patrones Orientados a Objetos

- **Cada patrón esta descrito por**
  - Nombre
    - Es un 'handle' que permite describir el problema, sus soluciones y sus consecuencias en una o dos palabras
  - Problema
    - Describe el problema y su contexto. Describe cuándo aplicar un patrón
  - Solución
    - Describe los elementos que hacen el diseño, sus relaciones, responsabilidades y colaboraciones. Sin embargo no describe un diseño o implementación particulares
  - Consecuencias
    - Resultados y efectos secundarios de usar el patrón

Separación interfase - implementación

- **Más que una patrón de diseño, es una práctica fundamental que fomenta el bajo acoplamiento**
  - Un cliente que depende de una interfase no depende de una implementación particular.
  - Varias clases pueden implementar la misma interfase, y pueden ser sustituidas.

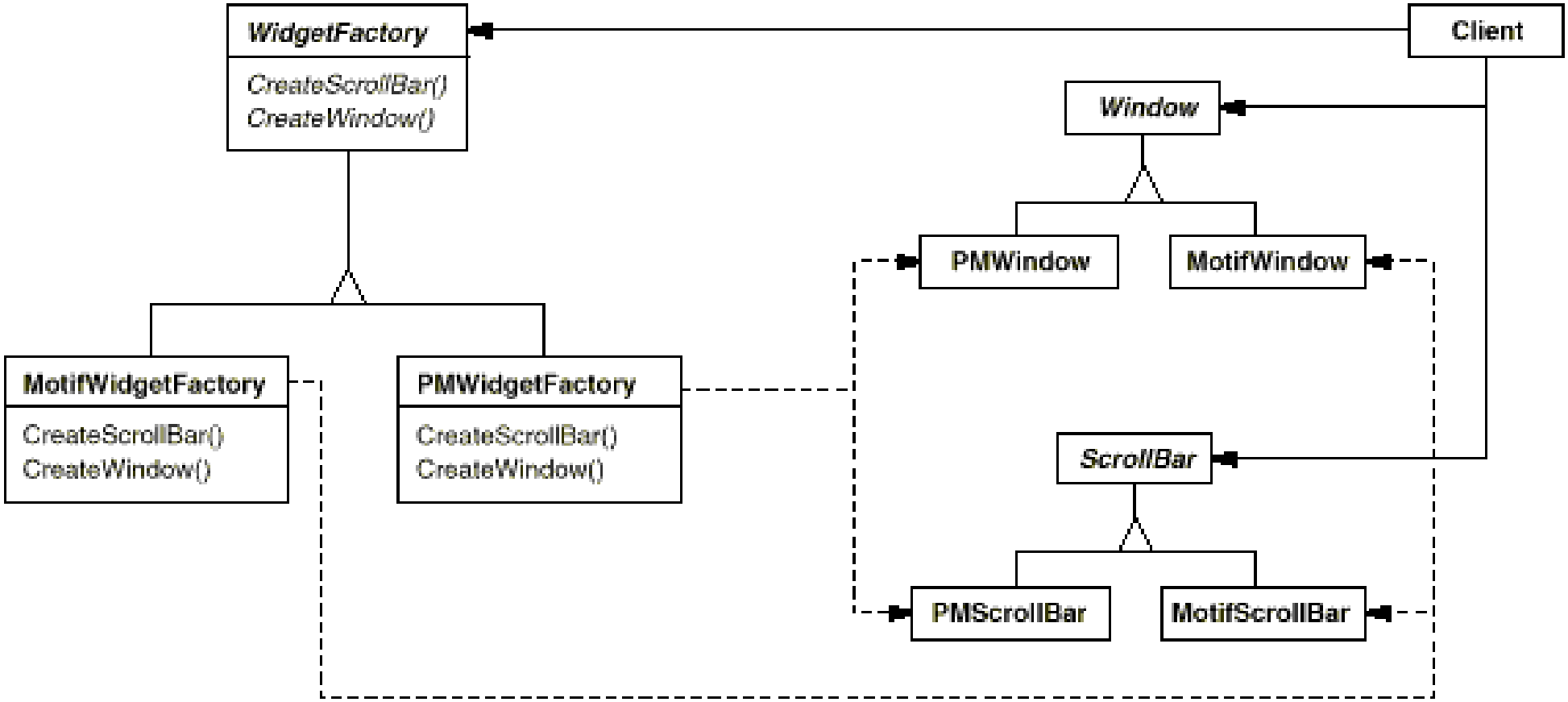
- **Java**
  - Java provee un tipo de dato llamado interface que equivale a una clase abstracta pura
  - Una clase Java puede heredar solamente de otra clase, pero puede implementar un numero indefinido de interfases

Patrón Model-View-Controller

- **MVC: Modelo - Vista - Controlador**
  - Utilizado para construir interfaces de usuario
  - Modelo
    - La aplicación
  - Vista
    - Presentación en pantalla, refleja estado del modelo
    - Pueden haber múltiples vistas
  - Controlador
    - Interacción con usuario
    - Cada vista tiene un controlador
  - Los elementos del patrón MVC pueden, a su vez, ser construidos otros patrones

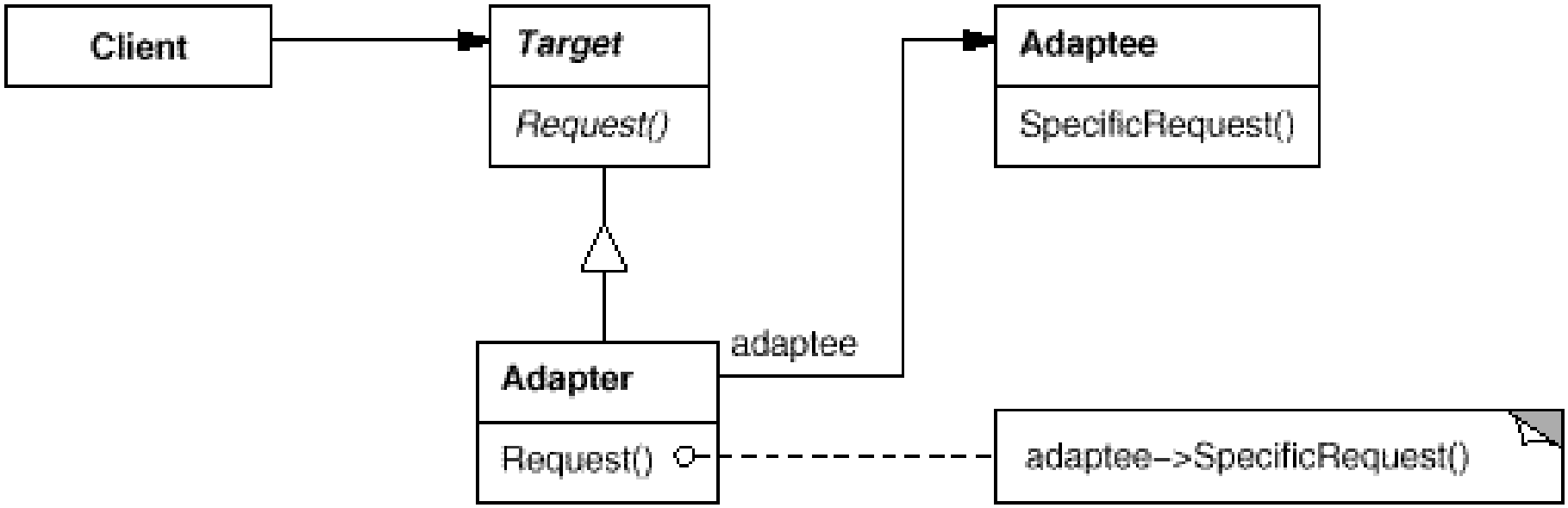
Patrón Abstract Factory

- **Intención**
  - Proveer una interfase para crear familias de objetos relacionados o dependientes sin especificar clases concretas



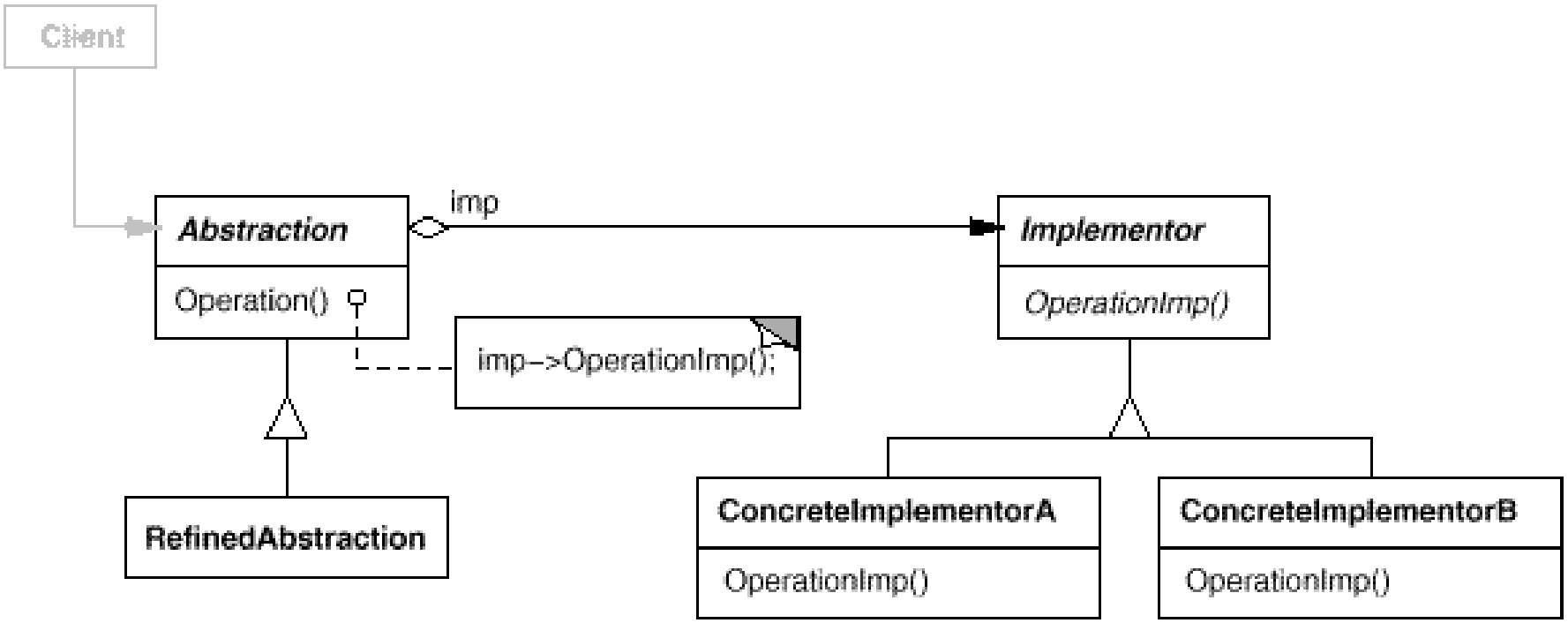
Patrón Adapter

- **Intención**
  - Convierte la interfase de una clase en la interfase de otra clase que un cliente requiere. El adaptador permite que clases que normalmente no podrían trabajar juntas debido a interfaces incompatibles puedan hacerlo.



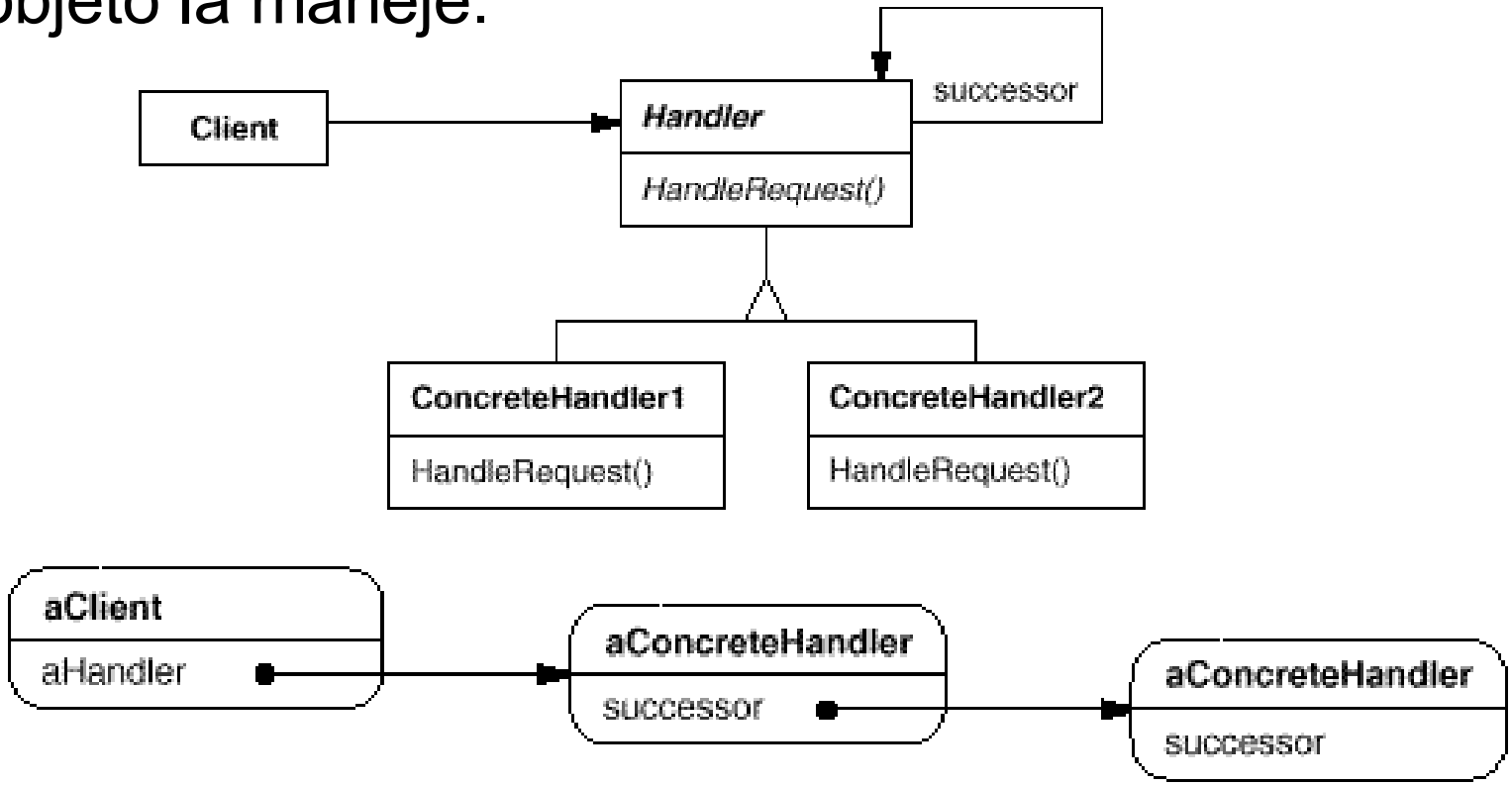
Patrón Bridge

- **Intención**
  - Desacoplar una abstracción de su implementación, para que ambas puedan evolucionar de forma independiente



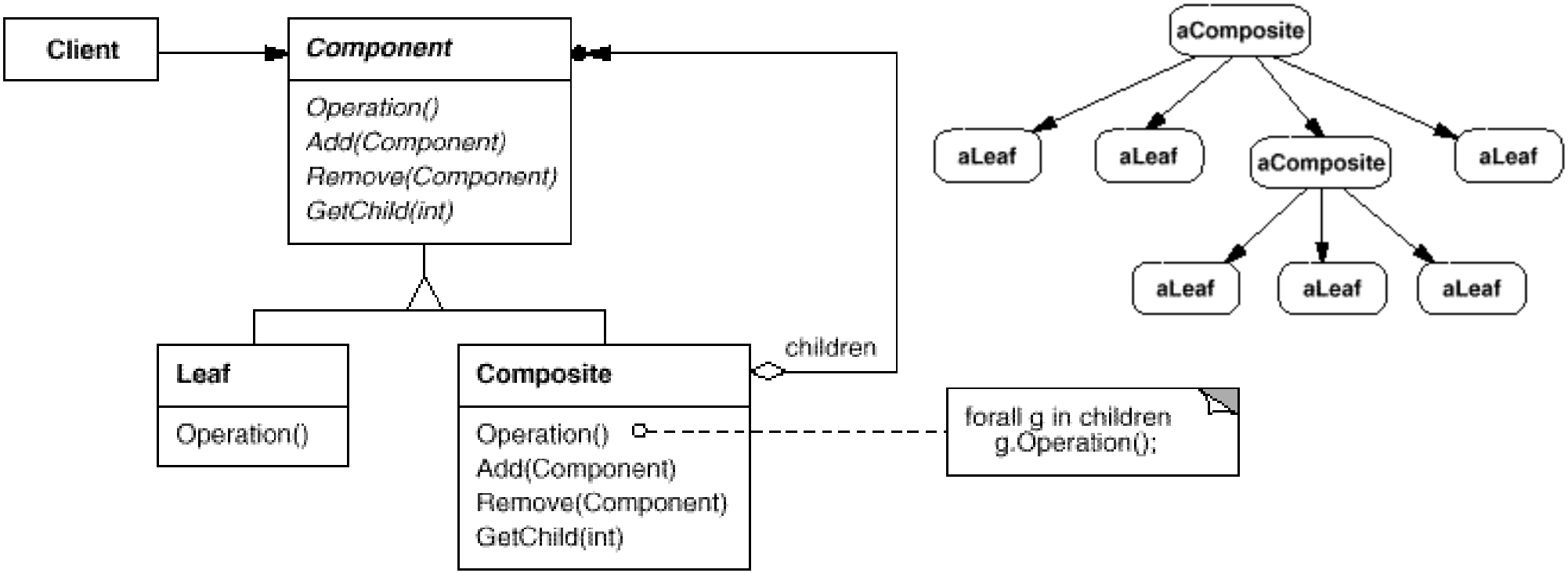
Patrón Chain of responsability

- **Intención**
  - Evitar acoplar el remitente de una petición al receptor de esta, permitiendo a más de un objeto la posibilidad de manejar la petición. Encadenar los receptores y pasar la petición a lo largo de la cadena hasta que un objeto la maneje.



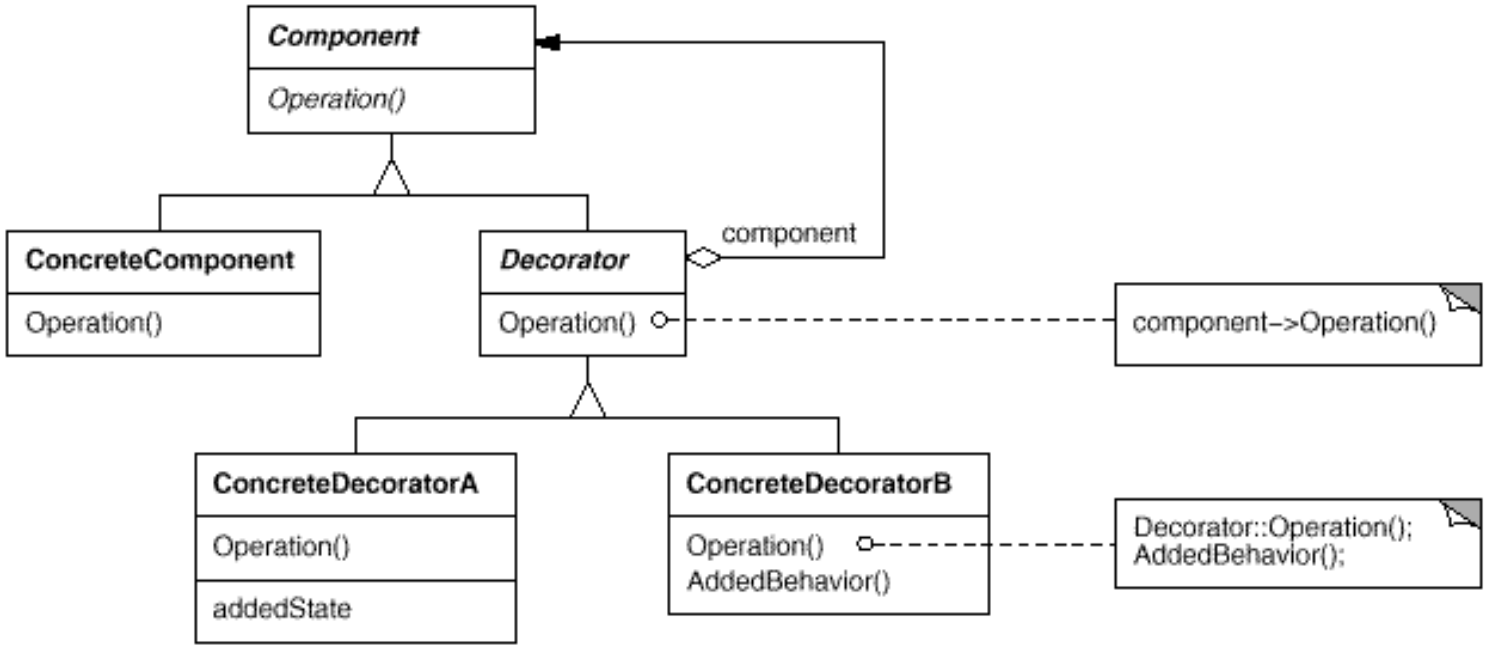
Patrón Composite

- **Intención**
  - Componer objetos en estructuras arborescentes para representar jerarquías parte-todo. El composite permite que los clientes traten a objetos individuales y composiciones de objetos de manera uniforme.



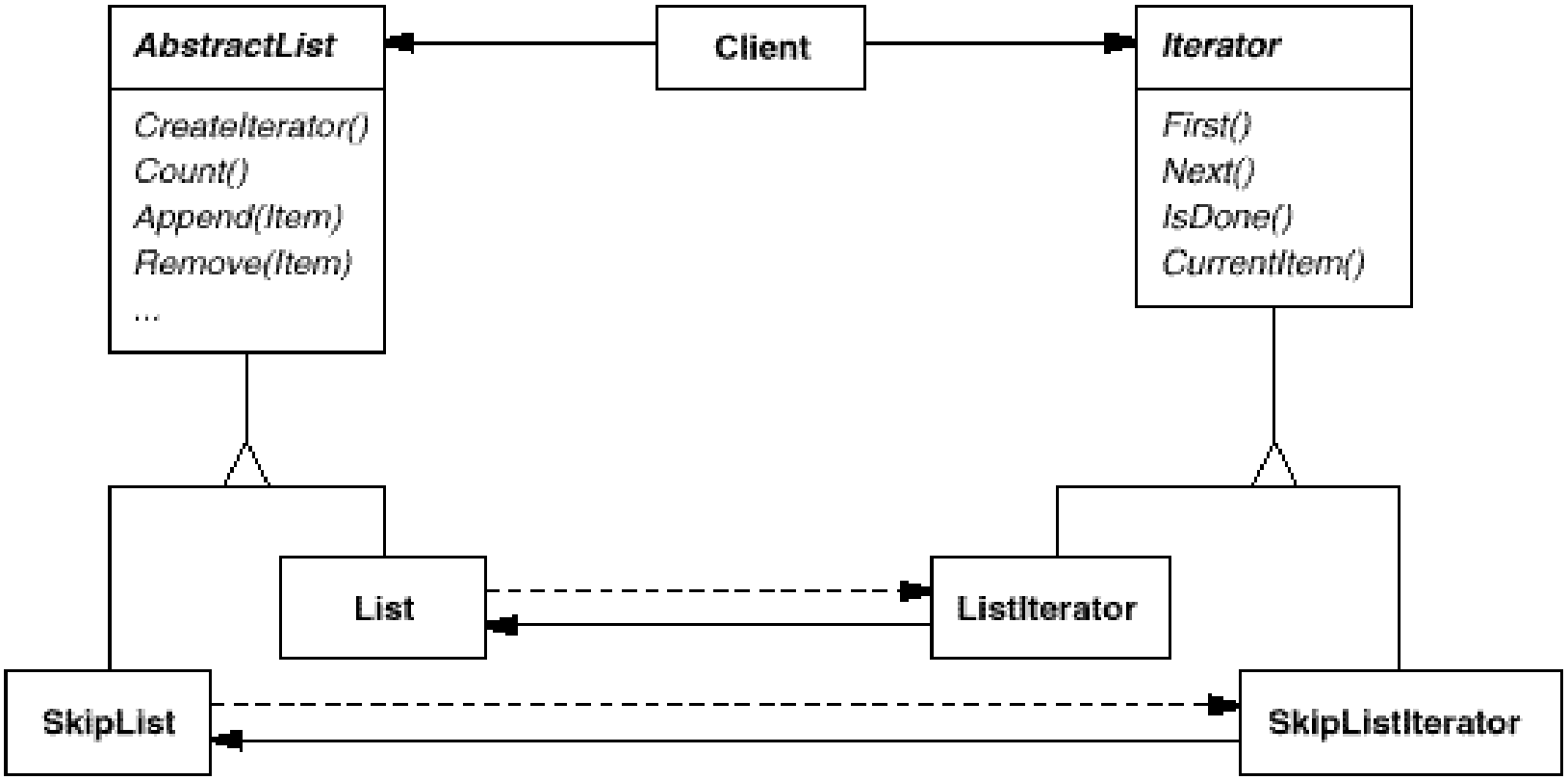
Patrón Decorator

- **Intención**
  - Agregar responsabilidades adicionales a un objeto de forma dinámica. Los decoradores proveen una alternativa flexible a las sub-clases para extender funcionalidad
    - Comportamiento adicional puede ser usado si clientes están al tanto de que se trata de un decorador



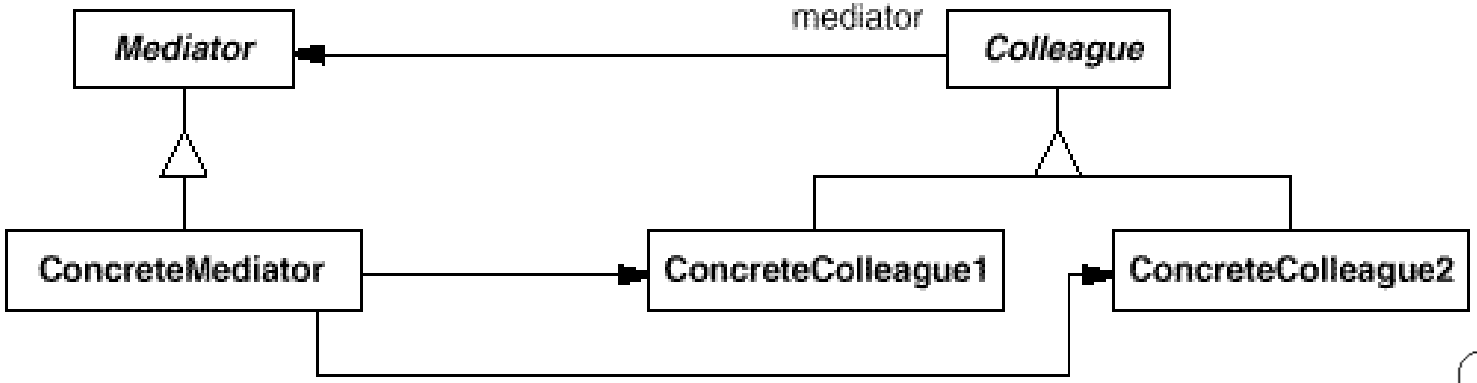
Patron Iterator

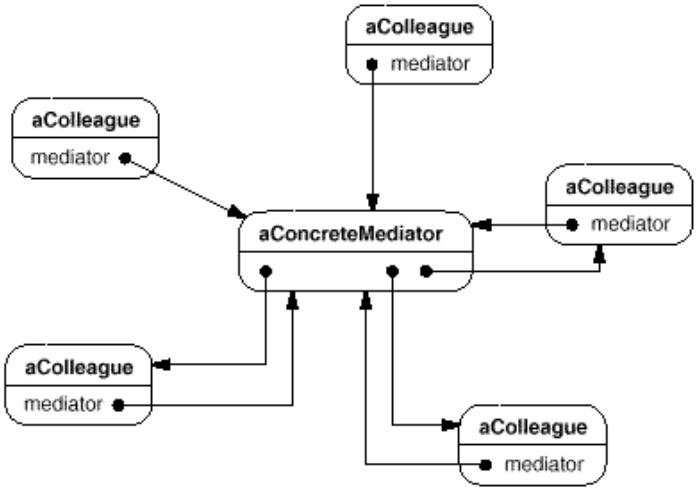
- **Intención**
  - Provee una manera de acceder a los elementos de un objeto agregado de manera secuencial sin exponer su representación interna.



Patrón Mediator

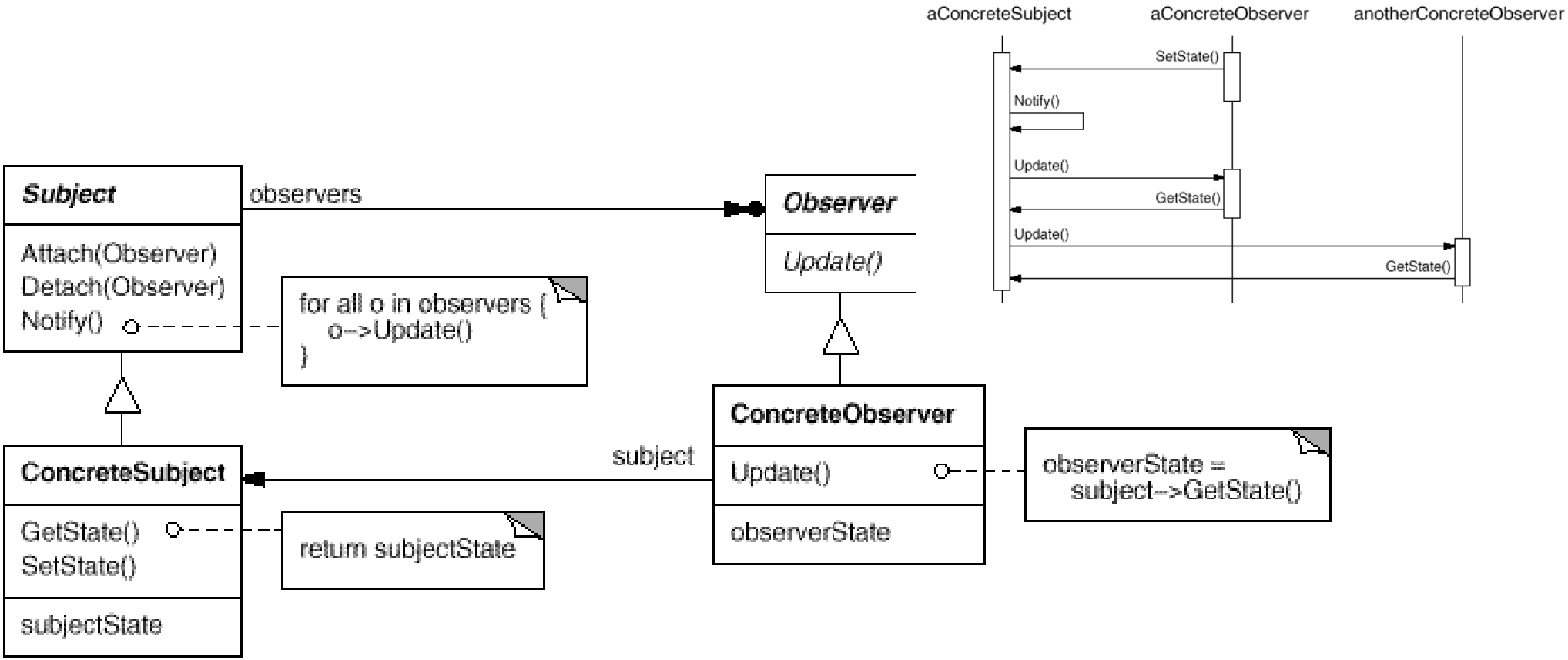
- **Intención**
  - Define un objeto que encapsula la manera como un conjunto de objetos interactúan. Mediador promueve el bajo acoplamiento al impedir que los objetos se referencien de manera explicita de, y permite variar su interacción de manera independiente.





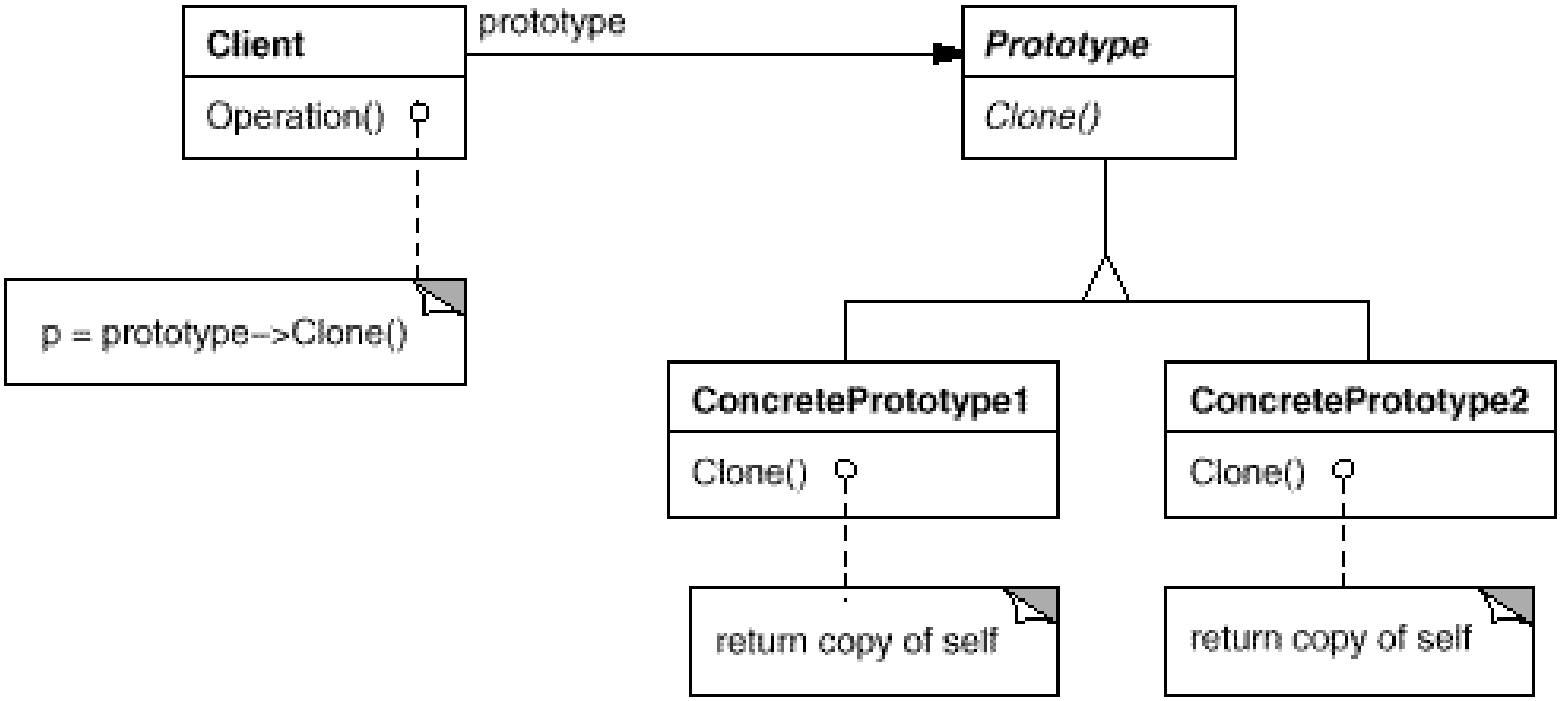
Patrón Observer

- **Intención**
  - Define una dependencia de 1..n entre objetos para que cuando un objeto cambie su estado, todos los objetos dependientes sean notificados y actualizados de manera automática.



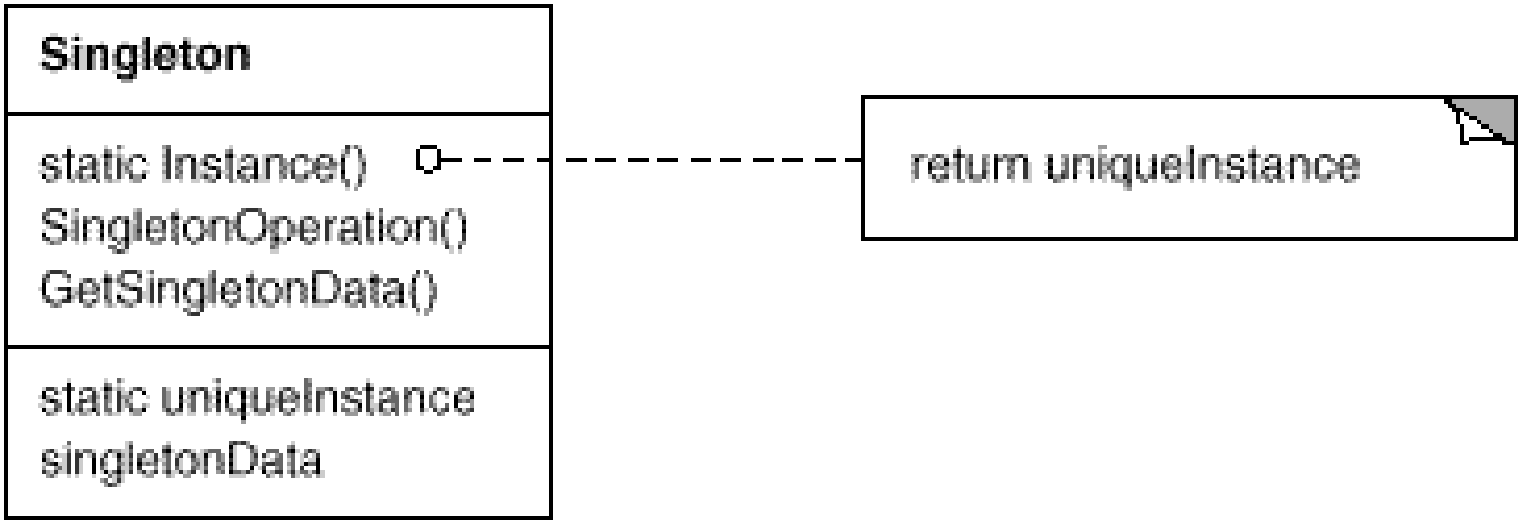
Patrón Prototype

- **Intención**
  - Especificar los tipos de objetos a crear mediante una instancia prototipo, y crear nuevos objetos copiando esta instancia.
    - En Java la clase Object provee el método clone()



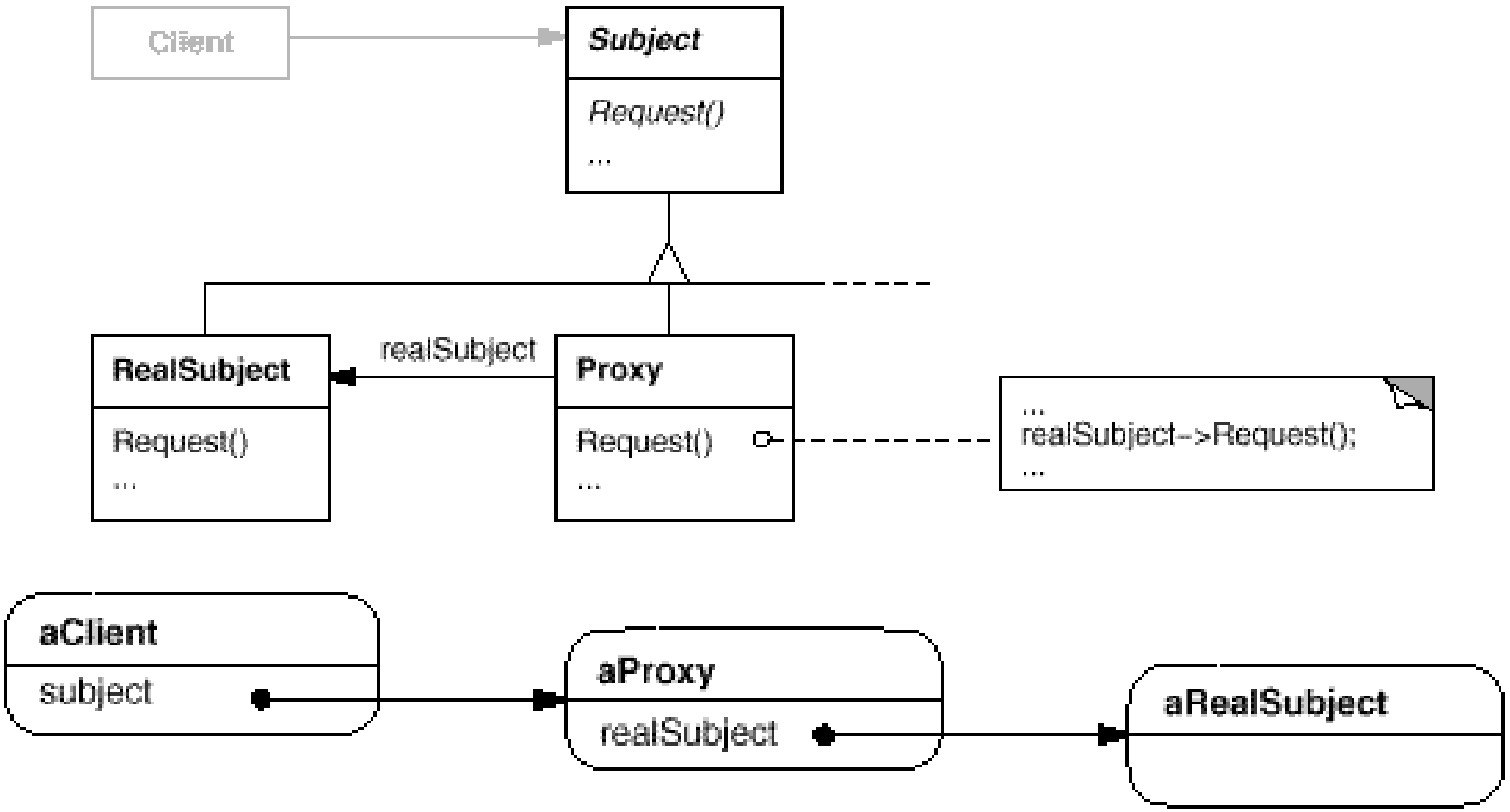
Patrón Singleton

- **Intención**
  - Asegurar que una clase tenga una sola instancia, y proveer un punto de acceso global a esta.
    - Se puede restringir el acceso al constructor declarándolo como privado.
    - Ojo con multi-threading



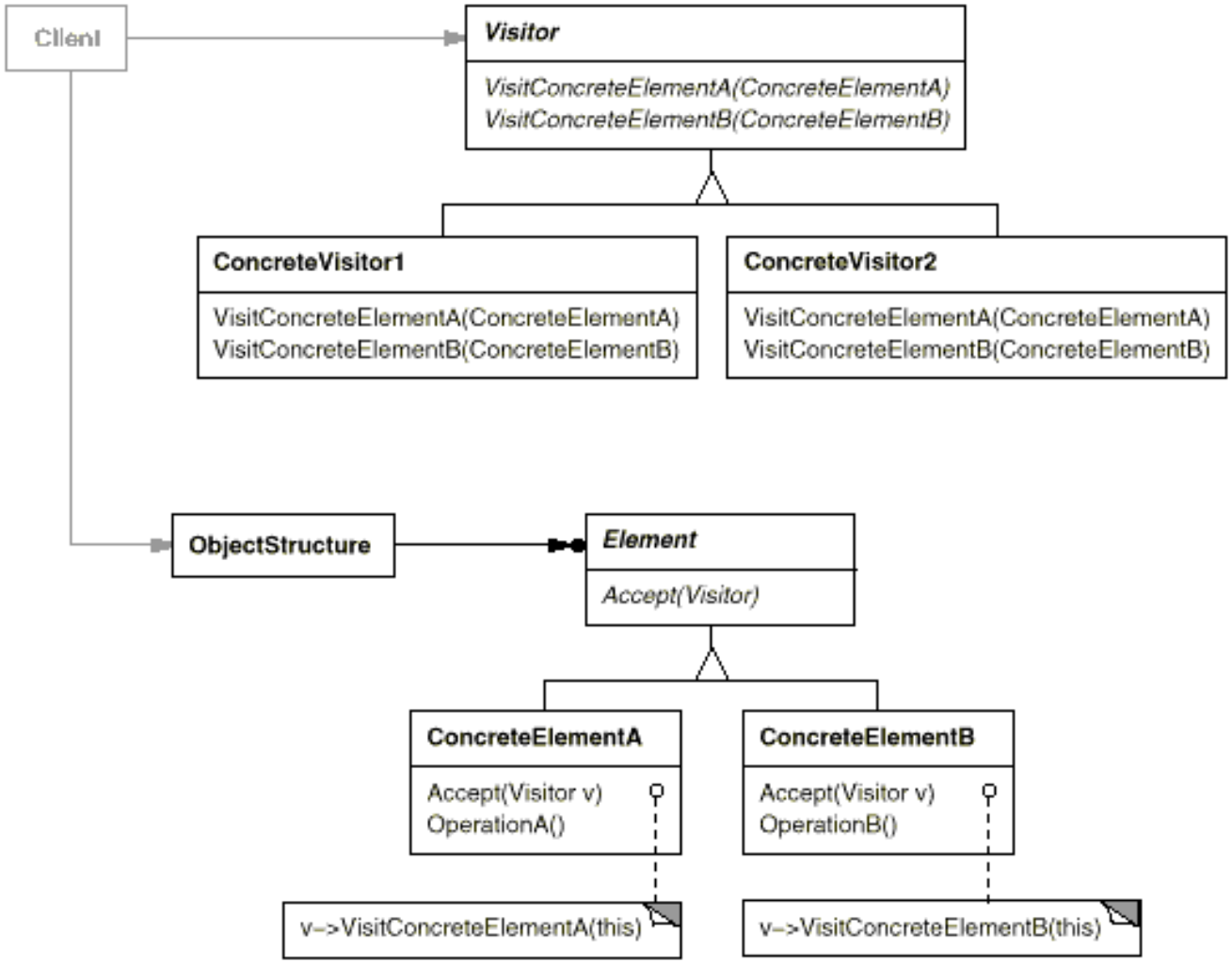
Patrón Proxy

- **Intención**
  - Proveer un remplazante o un soclo (paceholder) hacia un otro objeto para controlar el acceso a este último.



Patrón Visitor

- **Intención**
  - Representa una operación a ser efectuada en los elementos de una estructura de objetos. El visitor permite definir una nueva operación sin cambiar las clases de los elementos sobre los cuales opera.



Patrón Inversión de control

- **Intención**
  - Evitar que un objeto sea responsable de controlar la creación de otros objetos y para asociarse a ellos. Los objetos con los que colabora un objeto le son pasados por un tercero. Esto promueve el desacoplamiento, sobre todo si las dependencias del objeto cliente están descritas en términos de interfases.

  - Inyección de dependencias
    - Constructor
    - Setters