Patrones de diseño

Introducción

Conceptos básicos

- **Patrón de diseño**: Base para la búsqueda de soluciones a **problemas comunes** en el desarrollo de software [...] (wikipedia)
- Para que una solución sea considerada un patrón debe poseer ciertas características:
  - **Efectivo** resolviendo problemas similares en ocasiones anteriores.
  - **Reutilizable:** aplicable a diferentes problemas de diseño en distintas circunstancias.

Introducción

Los patrones pretenden

- Proporcionar catálogos de **elementos reusables** en el diseño de sistemas software.
- **Evitar la reiteración** en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un **vocabulario común** entre diseñadores.
- **Estandarizar** el modo en que se realiza el diseño.
- **Facilitar el aprendizaje** de las nuevas generaciones de diseñadores condensando conocimiento existente.

Introducción

Los patrones NO pretenden

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones.
Es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón.

Abusar o forzar el uso de los patrones puede ser un error.

Introducción

Antipatrón

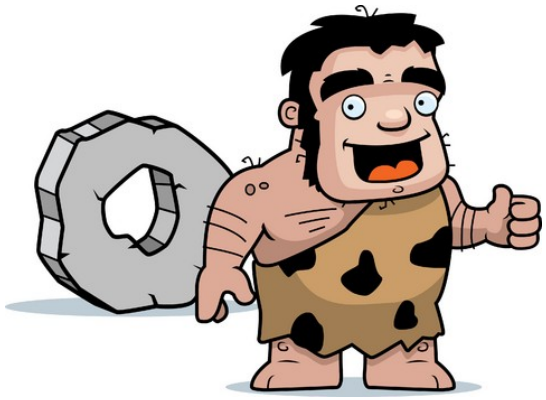
- Un *antipatrón* de diseño es un patrón de diseño que conduce a una mala solución para un problema.
- Evitar los *antipatrones* siempre que sea posible, requiere su reconocimiento e identificación dentro del ciclo de vida del software.
- Patrón de diseño → Buen Camino. 
- Antipatrón de diseño → Mal Camino. 

Más información sobre *antipatrones* de diseño aquí.

Introducción

¿Por qué debo usar patrones?

- Los patrones de diseño **son soluciones bien pensadas** a **problemas conocidos** de programación.
- Muchos programadores han padecido de estos problemas antes y han utilizado estas “soluciones” para ponerles remedio.
- No reinventar la rueda.



Introducción

Un vocabulario común

- **Clase abstracta**: Es una clase que tiene al menos un método abstracto que obliga a que toda la clase sea ***abstract.***
- **Herencia**: Facilita la creación de objetos a partir de otros ya existentes e implica que una subclase obtiene todo el comportamiento (métodos) y eventualmente los atributos (variables) de su superclase (clase padre).
- **Interface**: Contiene cabeceras de métodos y constantes (variables finales), pero no implementación de métodos o miembros de datos no-finales.

Introducción

Un vocabulario común

- **Polimorfismo:** se refiere a la capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente (wikipedia).
- **Sobre carga de métodos**: posibilidad de tener dos o más métodos con el mismo nombre pero funcionalidad diferente.
- **Granularidad**: Densidad de cada objeto.

Introducción

Un poco de historia...

- En 1994 apareció el libro "***Design Patterns: Elements of Reusable Object Oriented Sofware***" escrito por los ahora famosos **Gang of Four** (GoF).
- **GoF:** Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

Tipos de patrones

De creación

- Resuelven problemas relacionados con la creación de instancias de objetos.

De estructura

- Se centran en problemas relacionados con la forma de estructurar las clases.

De comportamiento

- Permiten resolver problemas relacionados con el comportamiento de la aplicación, normalmente en tiempo de ejecución.

Tipos de patrones

De creación

- Abstract factory
- Factory method
- Singleton
- Prototype
- Builder

Tipos de patrones

De estructura

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Tipos de patrones

De comportamiento (1/2)

- Chain of responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer

Tipos de patrones

De comportamiento (2/2)

- State
- Strategy
- Template method
- Visitor

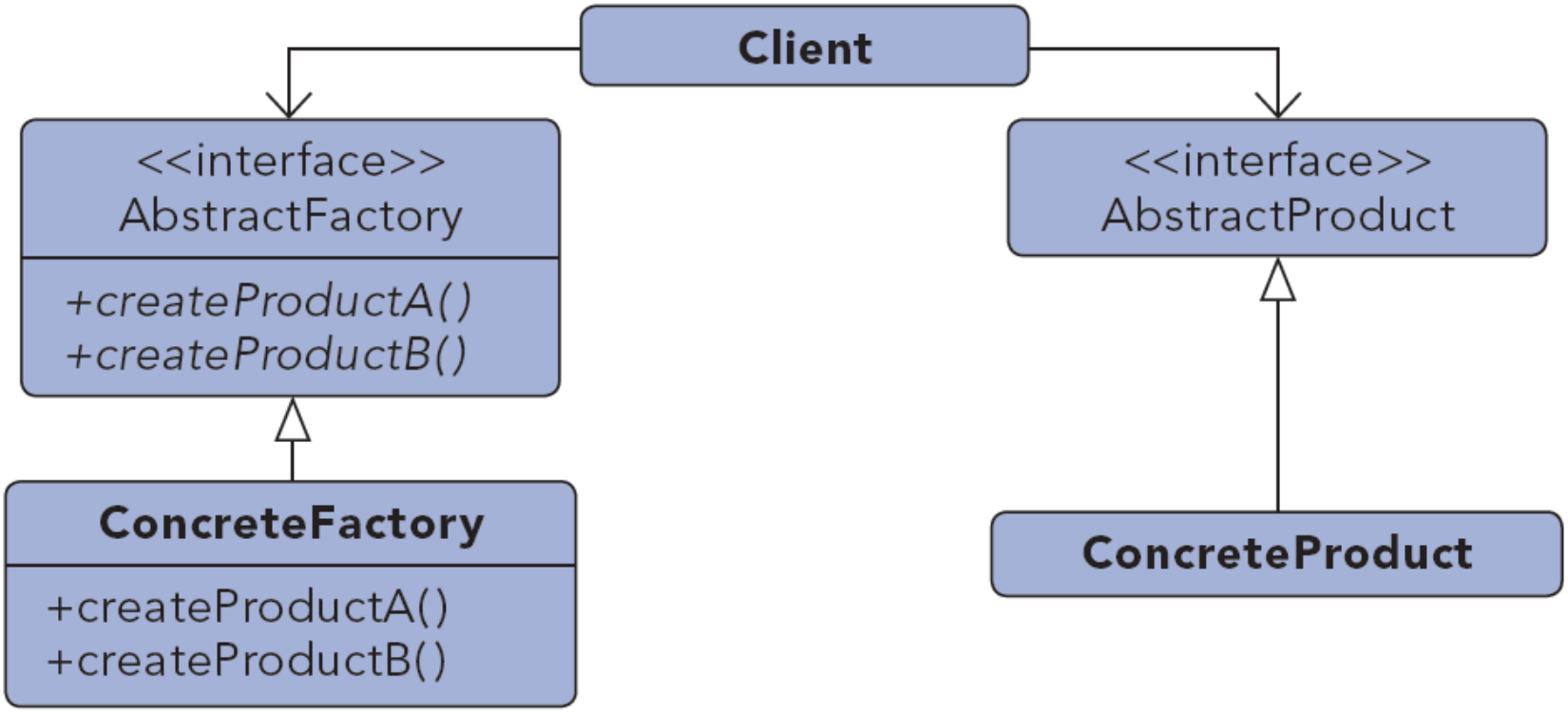
Patrones de creación

Abstract factory

- Proporciona una clase que delega la creación de una o más clases concretas con el fin de entregar objetos específicos.
- Este patrón puede ser utilizado cuando:
  - La creación de objetos debe ser independiente del sistema que los utilice.
  - Los sistemas deben ser capaces de utilizar múltiples familias de objetos.
  - Se usan bibliotecas sin exponer detalles de la implementación.

Patrones de creación

Abstract factory



Ejemplo
Abstract factory

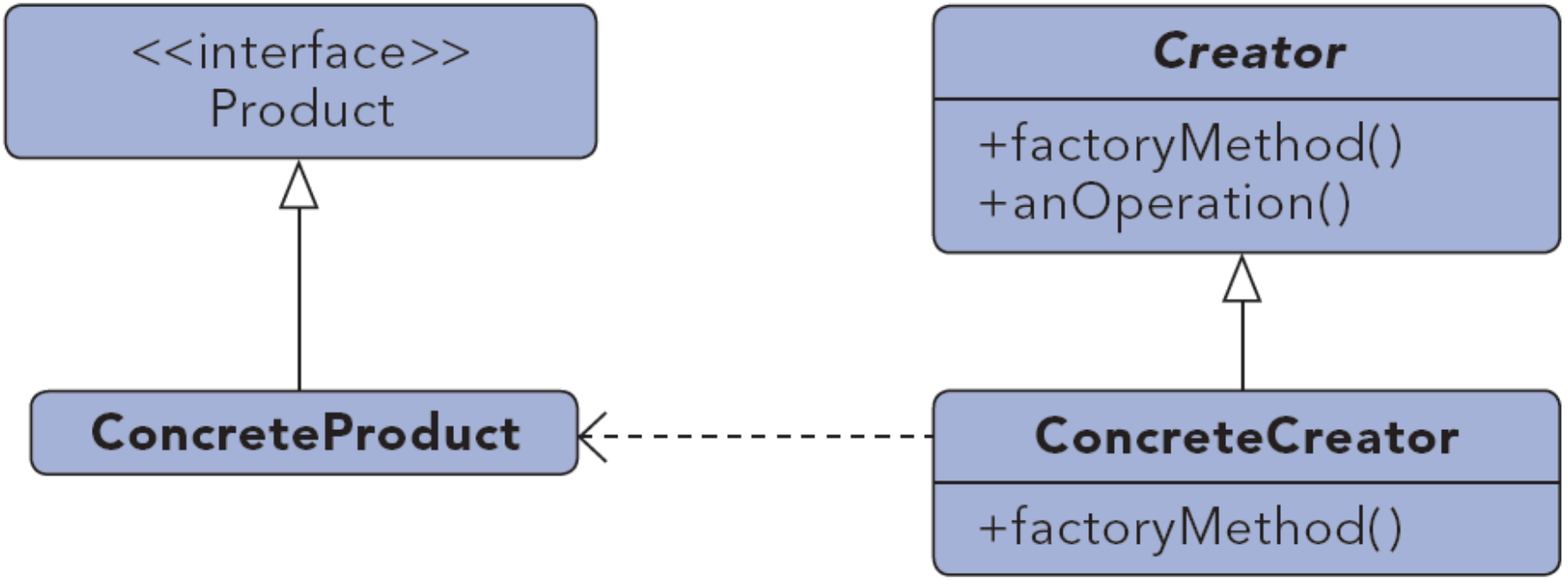
Patrones de creación

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.
- Este patrón puede ser utilizado cuando:
  - Una clase no puede anticipar el tipo de objeto que debe crear.
  - Subclases pueden especificar qué objetos deben ser creados.

Patrones de creación

Factory method



Ejemplo
Factory method

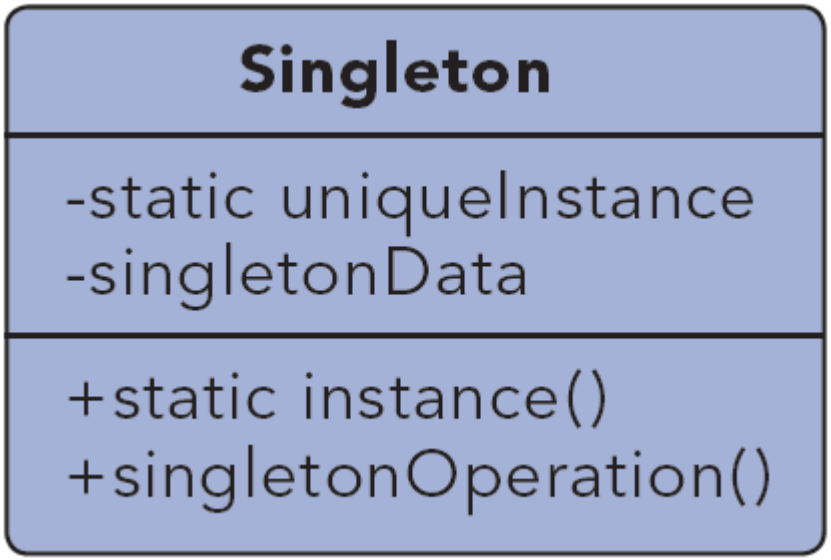
Patrones de creación

Singleton

- Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.
- Este patrón puede ser utilizado cuando:
  - Se requiere **exactamente una instancia** de una clase.
  - Es necesario acceso controlado a un solo objeto.

Patrones de creación

Singleton



Ejemplo Singleton

Patrones de creación

Prototype

- Crear objetos mediante clonación basados en una plantilla de objetos existentes .
- Este patrón puede ser utilizado cuando (1/2):
  - La composición, creación y representación de los objetos debe desacoplarse de un sistema.
  - Las clases que se creen se especifican en tiempo de ejecución.
  - Para un objeto existen un número limitado de combinaciones de estado .

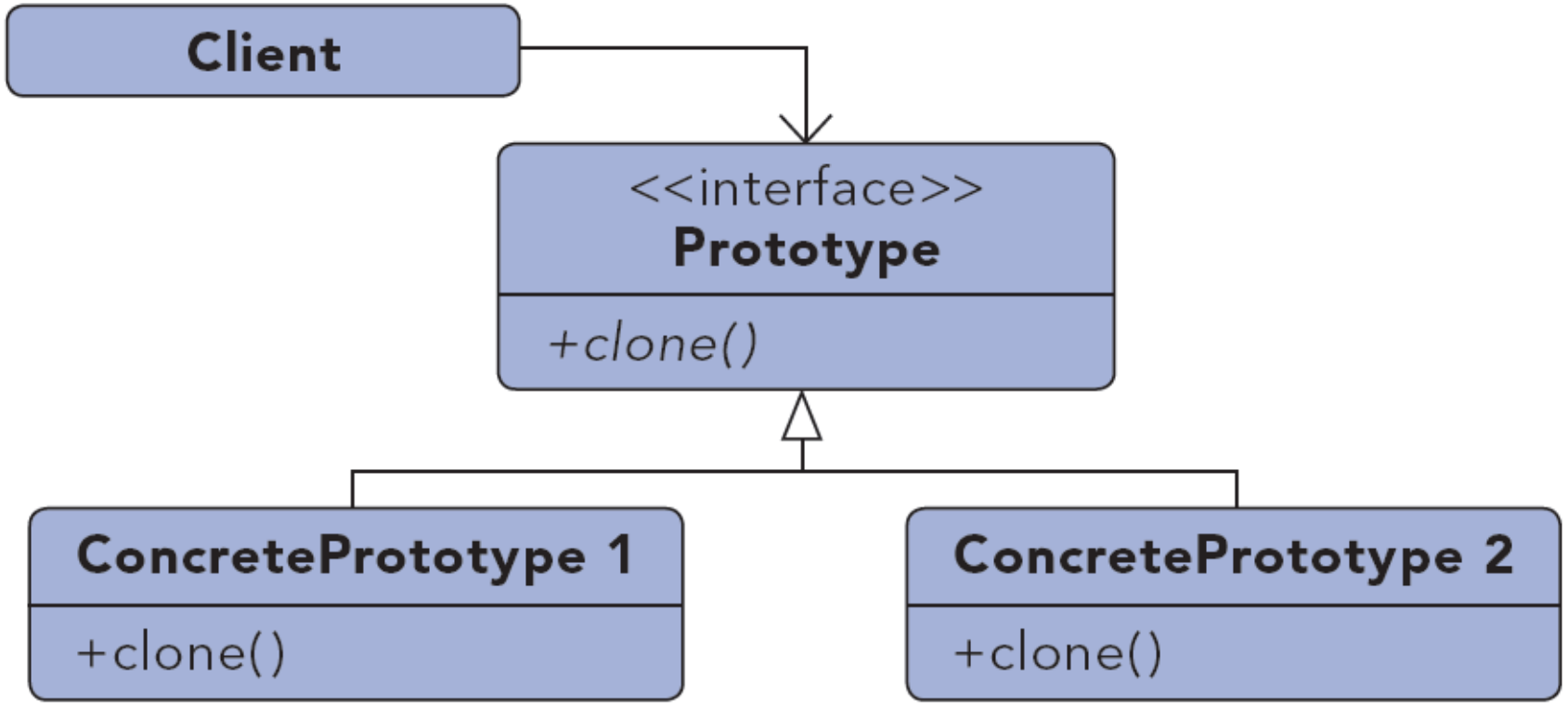
Patrones de creación

Prototype

- Este patrón puede ser utilizado cuando (2/2):
  - Se requiere que objetos o estructuras de objetos sean idénticos o se parezcan mucho a otros objetos o estructuras existentes.
  - La creación inicial de cada objeto es una operación costosa.

Patrones de creación

Prototype



Ejemplo Prototype

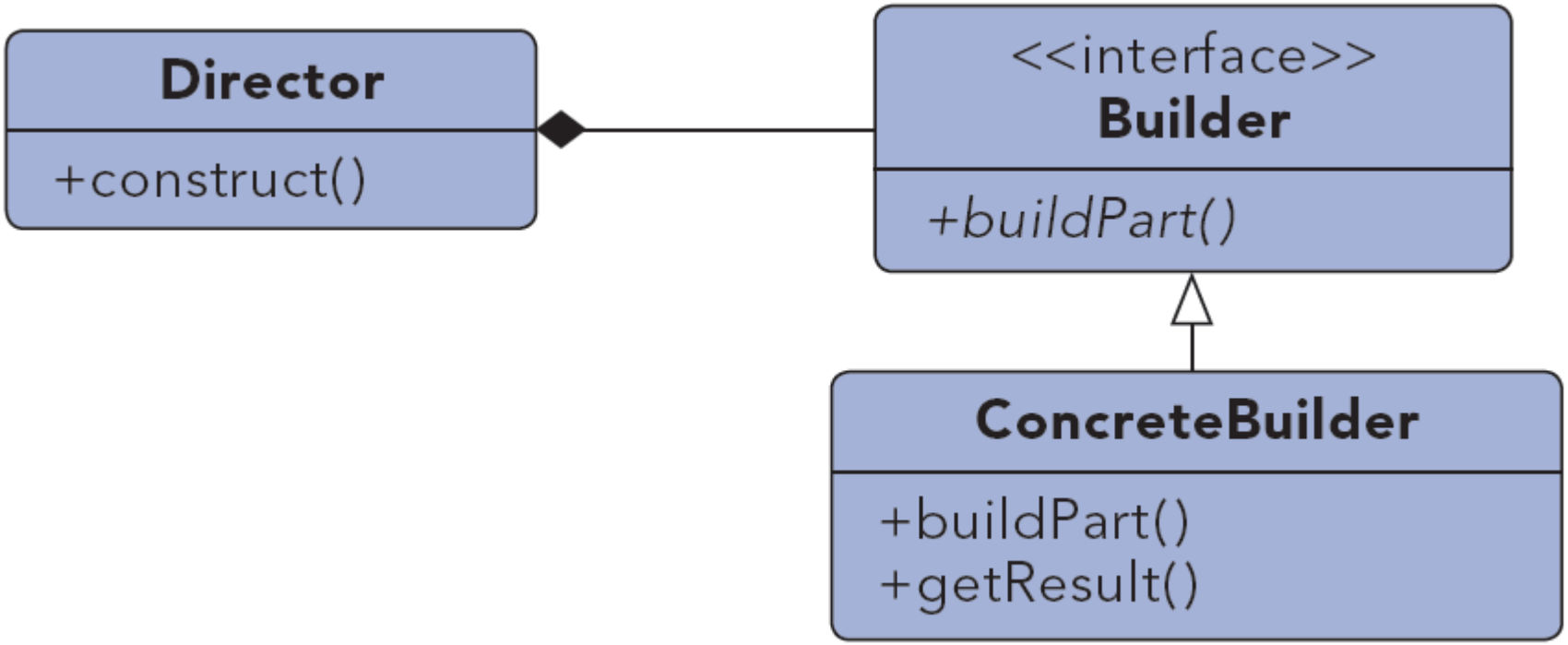
Patrones de creación

Builder

- Centraliza el proceso de creación de un objeto en un único punto, de tal forma que el mismo proceso de construcción pueda crear representaciones diferentes.
- Este patrón puede ser utilizado cuando:
  - Los algoritmos de creación de objetos deben ser desacoplados del sistema.
  - Son obligatorias múltiples representaciones de algoritmos de creación.
  - Se requiere control sobre el proceso de creación en tiempo de ejecución.

Patrones de creación

Builder



Ejemplo Builder

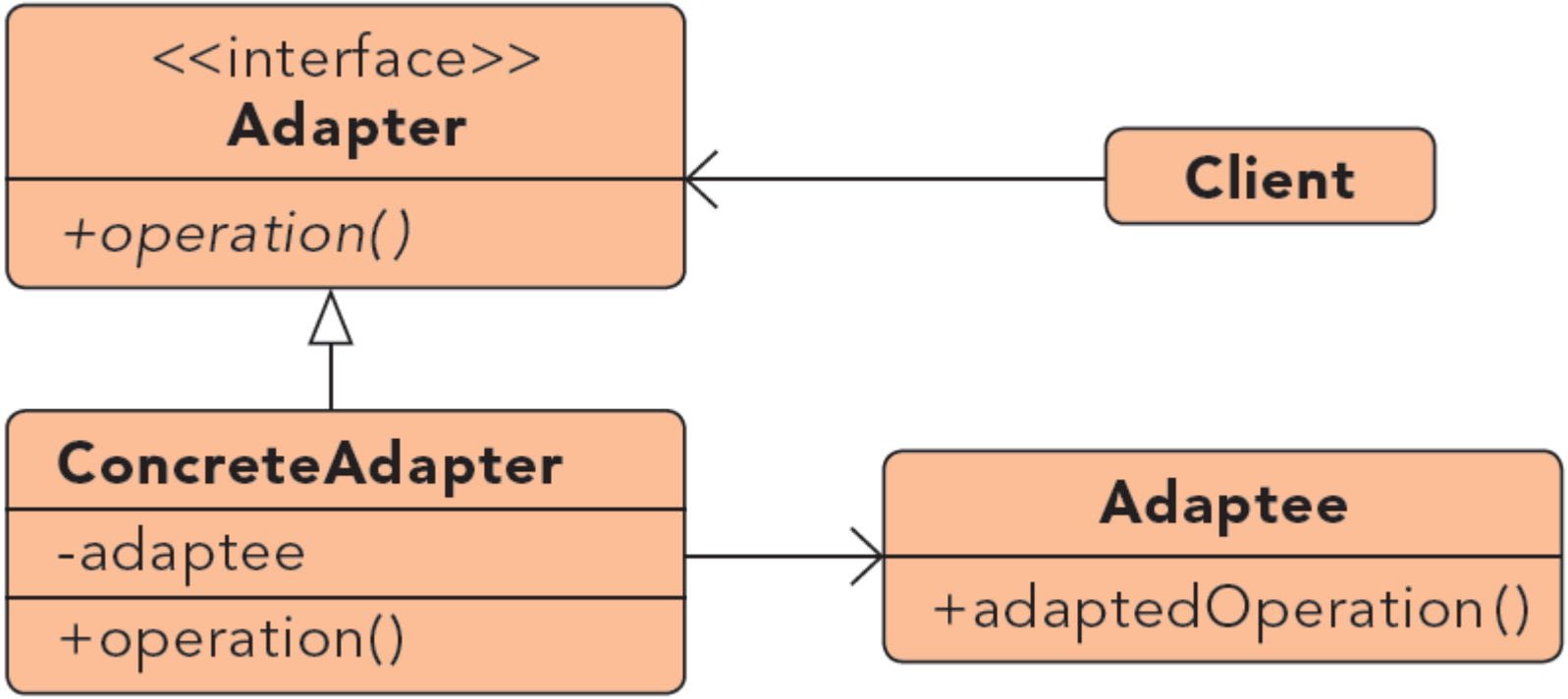
Patrones de estructura

Adapter

- Permite trabajar juntas a **clases con interfaces diferentes** a través de la creación de un objeto común mediante el que puedan comunicarse e interactuar.
- Este patrón puede ser utilizado cuando:
  - Una clase no cumple los requisitos de interfaz.
  - Condiciones complejas “atan” el comportamiento de los objetos a su estado.
  - Las transiciones entre los estados necesitan ser explícitas.

Patrones de estructura

Adapter



Ejemplo Adapter

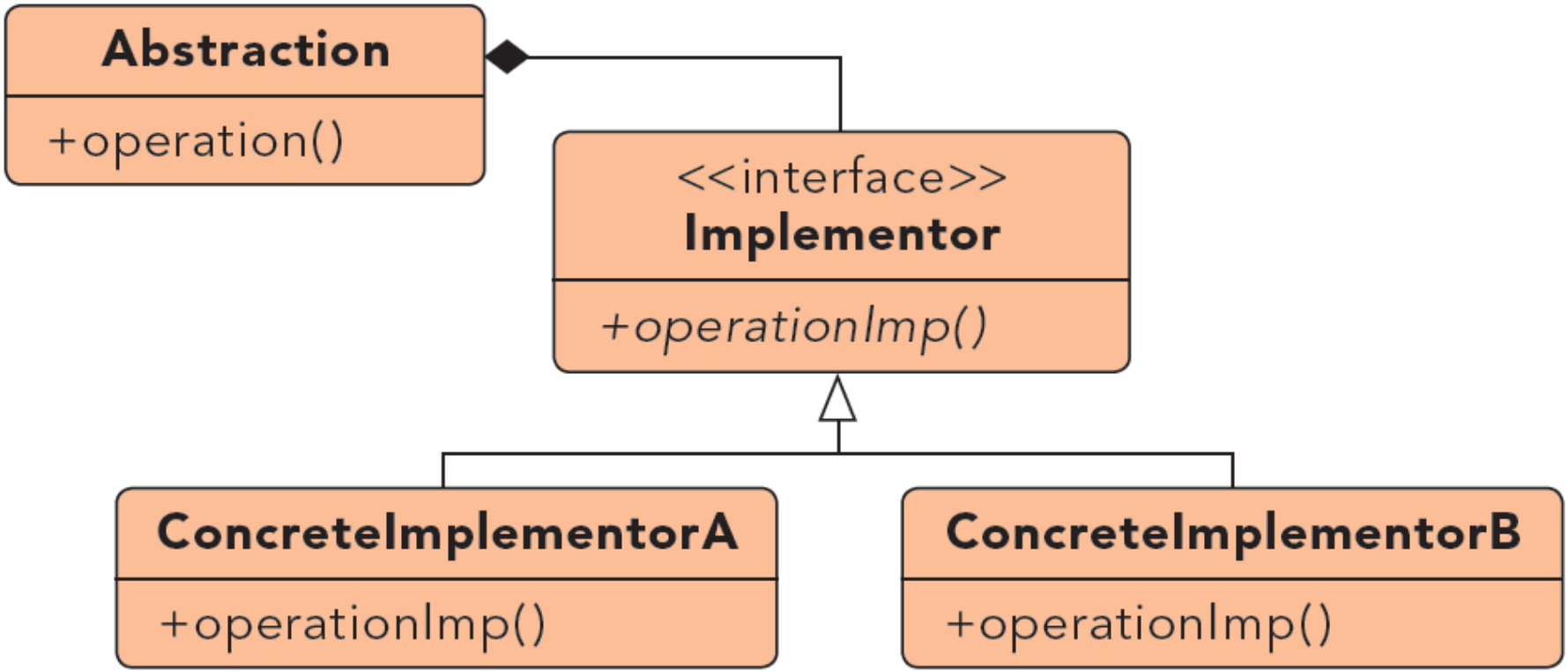
Patrones de estructura

Bridge

- Define una estructura de objeto abstracto con independencia de la implementación con el fin de limitar el acoplamiento.
- Este patrón puede ser utilizado cuando:
  - Las abstracciones e implementaciones no deben ser dependientes en tiempo de compilación.
  - Los cambios en la implementación no deberían tener impacto en los clientes.
  - Los detalles de la implementación se deben ocultar al cliente.

Patrones de estructura

Bridge



Ejemplo Bridge

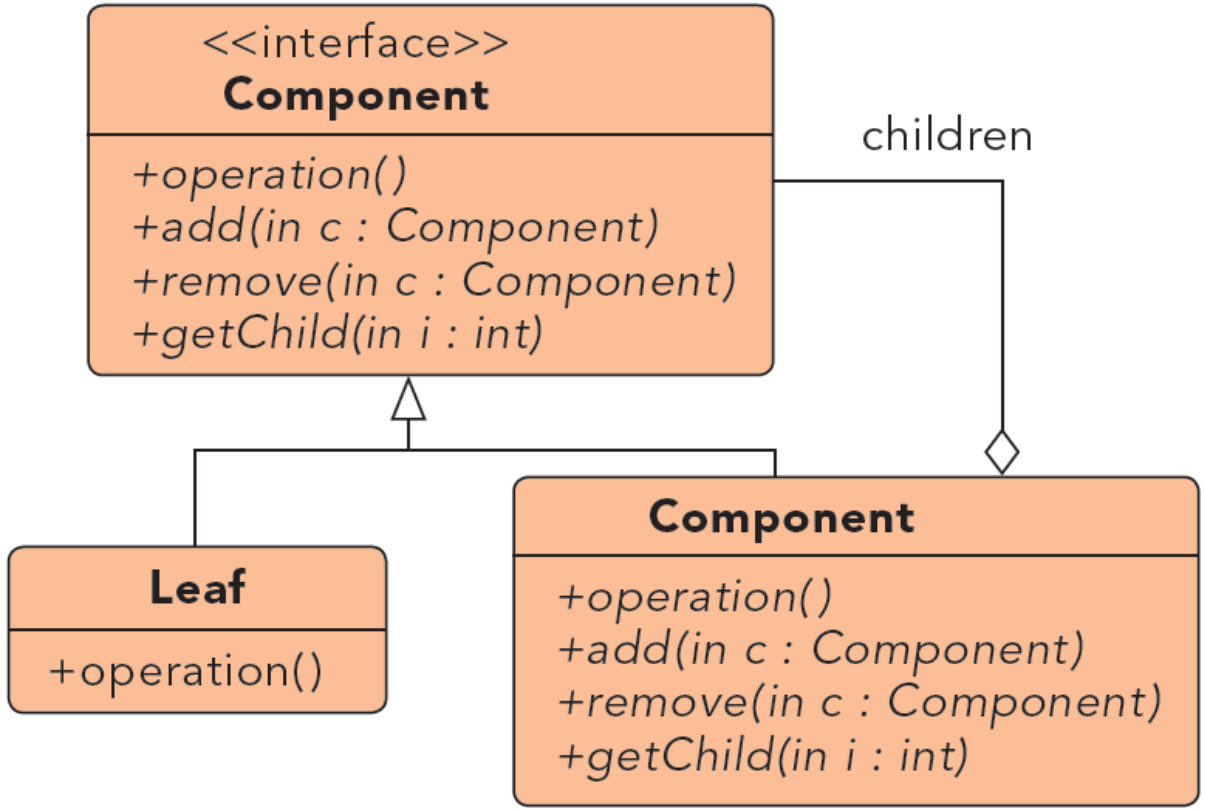
Patrones de estructura

Composite

- Facilita la creación de jerarquías de objetos donde cada objeto se puede tratar de forma independiente o como un conjunto de objetos anidados a través de la misma interfaz.
- Este patrón puede ser utilizado cuando:
  - Se necesitan representaciones jerárquicas de objetos.
  - Los objetos y composiciones de objetos debe ser tratados de manera uniforme.

Patrones de estructura

Composite



Ejemplo Composite

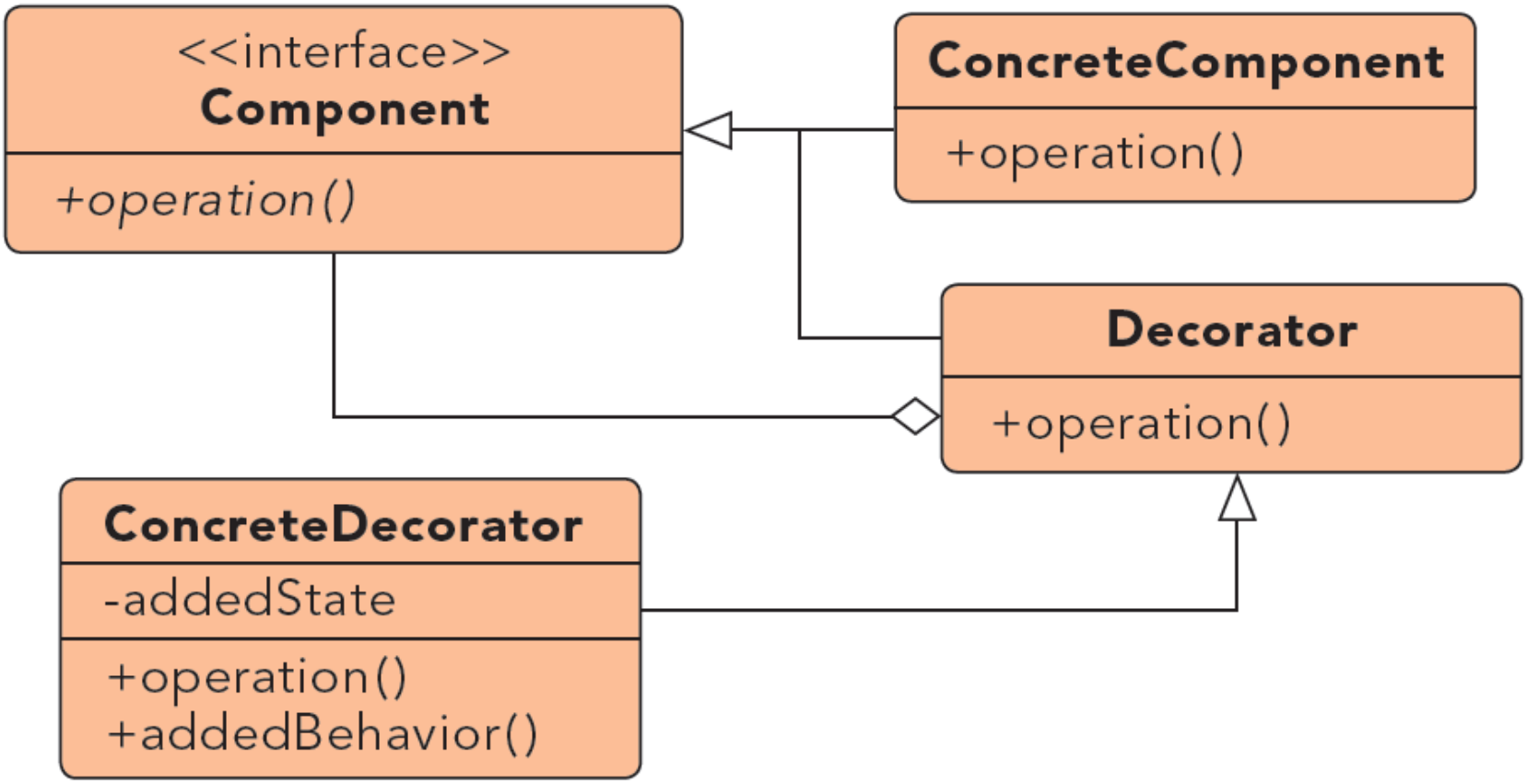
Patrones de estructura

Decorator

- Añade dinámicamente funcionalidad a un objeto. Permite no tener que crear subclases incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.
- Este patrón puede ser utilizado cuando:
  - El comportamiento de objetos debe ser dinámicamente modificable.
  - Las funcionalidades específicas no deben residir en la parte alta de la jerarquía de objetos.

Patrones de estructura

Decorator



Ejemplo Decorator

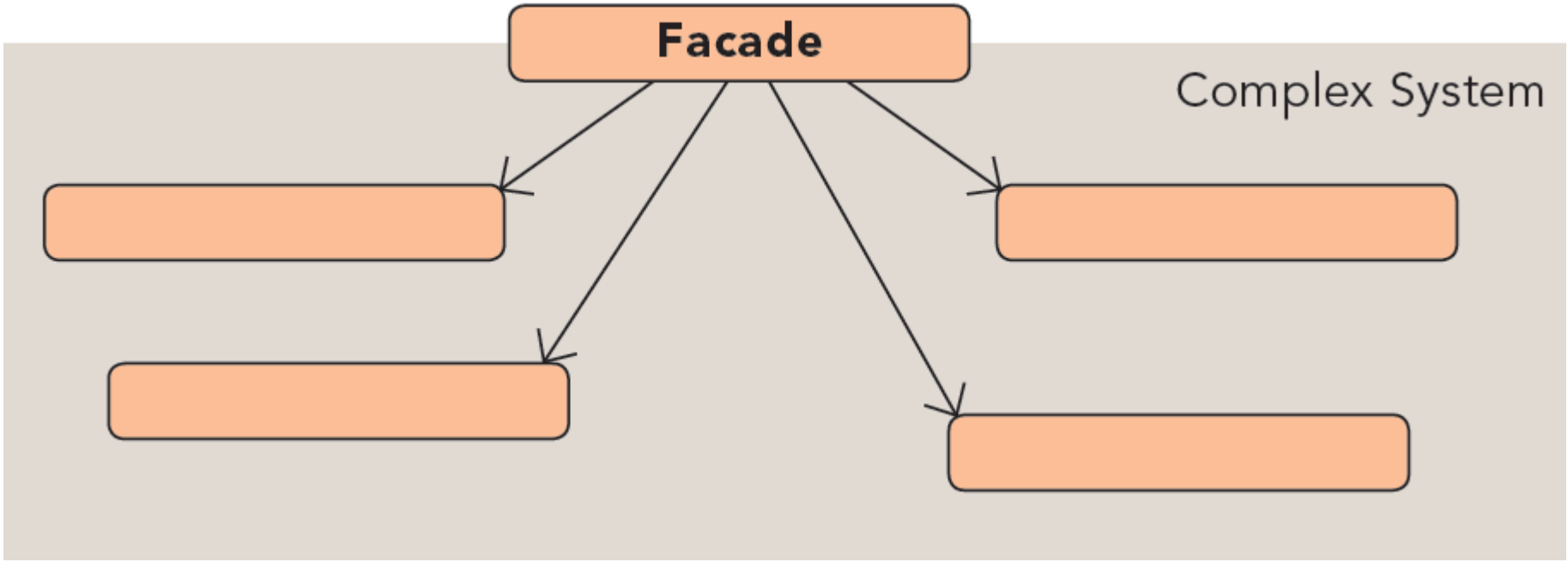
Patrones de estructura

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.
- Este patrón puede ser utilizado cuando:
  - Se necesita una interfaz simple para proporcionar acceso a un sistema complejo.
  - Hay muchas dependencias entre las implementaciones de sistemas y los clientes.

Patrones de estructura

Facade



Ejemplo Facade

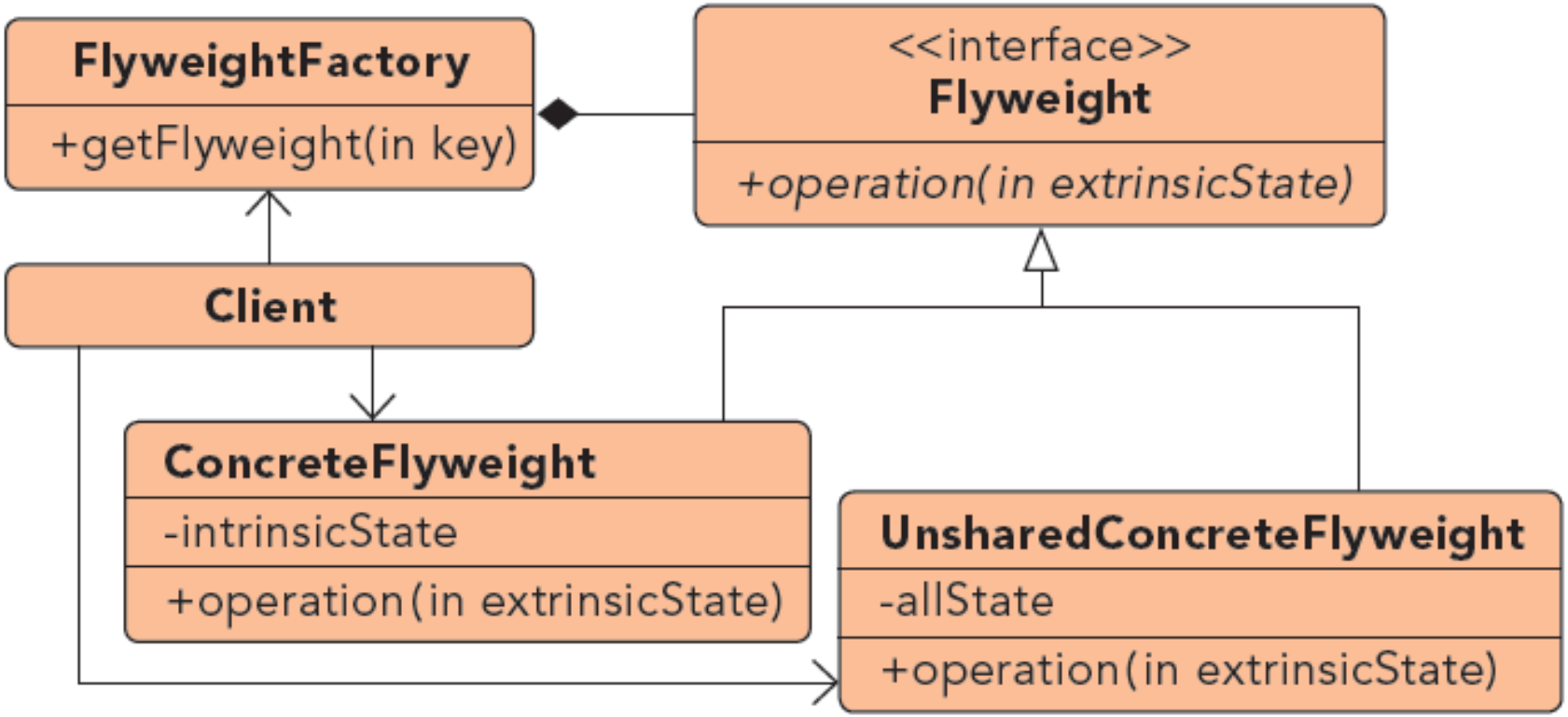
Patrones de estructura

Flyweight

- Facilita la reutilización de muchos objetos de "*grano fino*", haciendo más eficiente la utilización de grandes cantidades de objetos.
- Este patrón puede ser utilizado cuando:
  - Se utilizan muchos objetos parecidos y los costes de almacenamiento son altos.
  - Unos pocos objetos compartidos se pueden sustituir por muchos no compartidos.
  - No tiene importancia la identidad de cada objeto.

Patrones de estructura

Flyweight



Ejemplo Flyweight

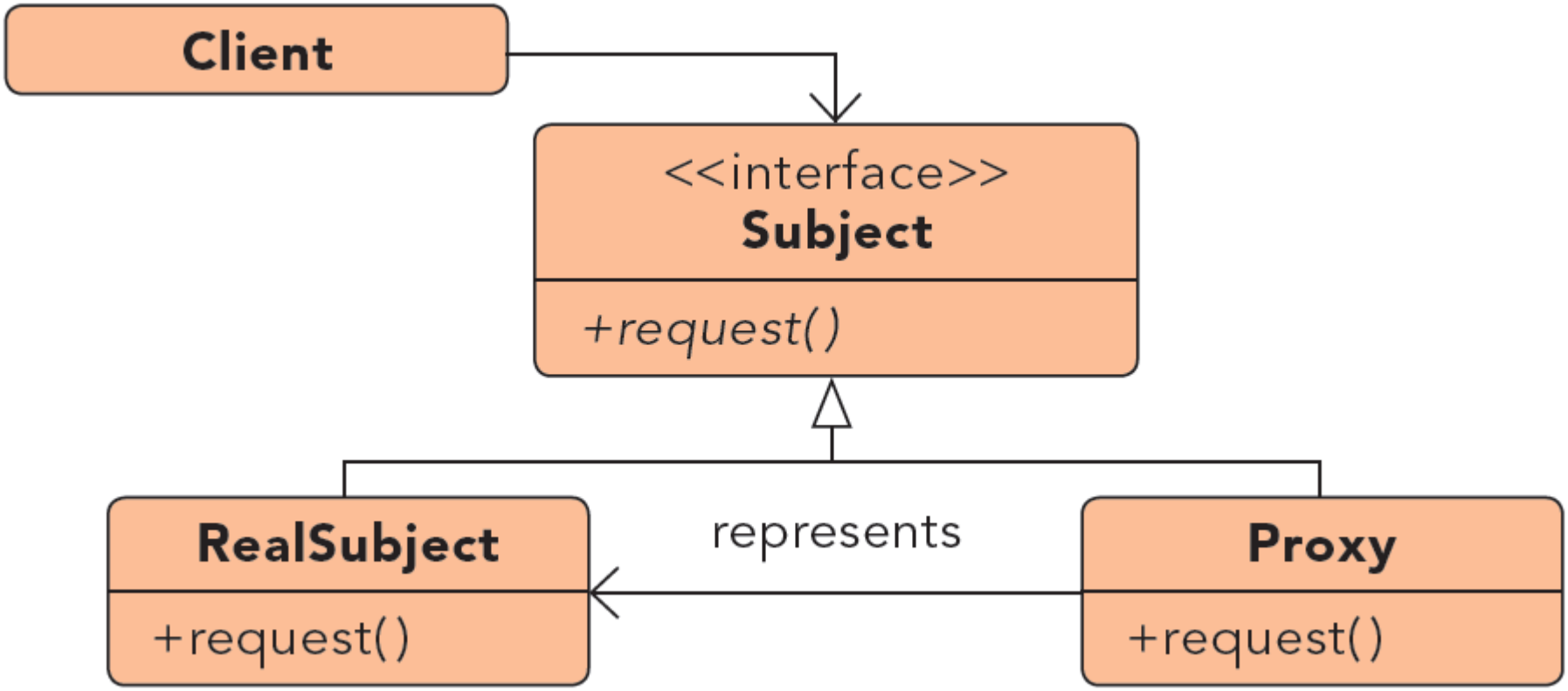
Patrones de estructura

Proxy

- Una clase que permite operar con un objeto de otra clase exponiendo una o más de sus interfaces.
- Este patrón puede ser utilizado cuando:
  - El objeto representado es externo al sistema.
  - Los objetos se deben crear bajo demanda.
  - Se requiere control de acceso para el objeto original.
  - Se requiere añadir funcionalidad cuando se accede a un objeto.

Patrones de estructura

Proxy



Ejemplo Proxy

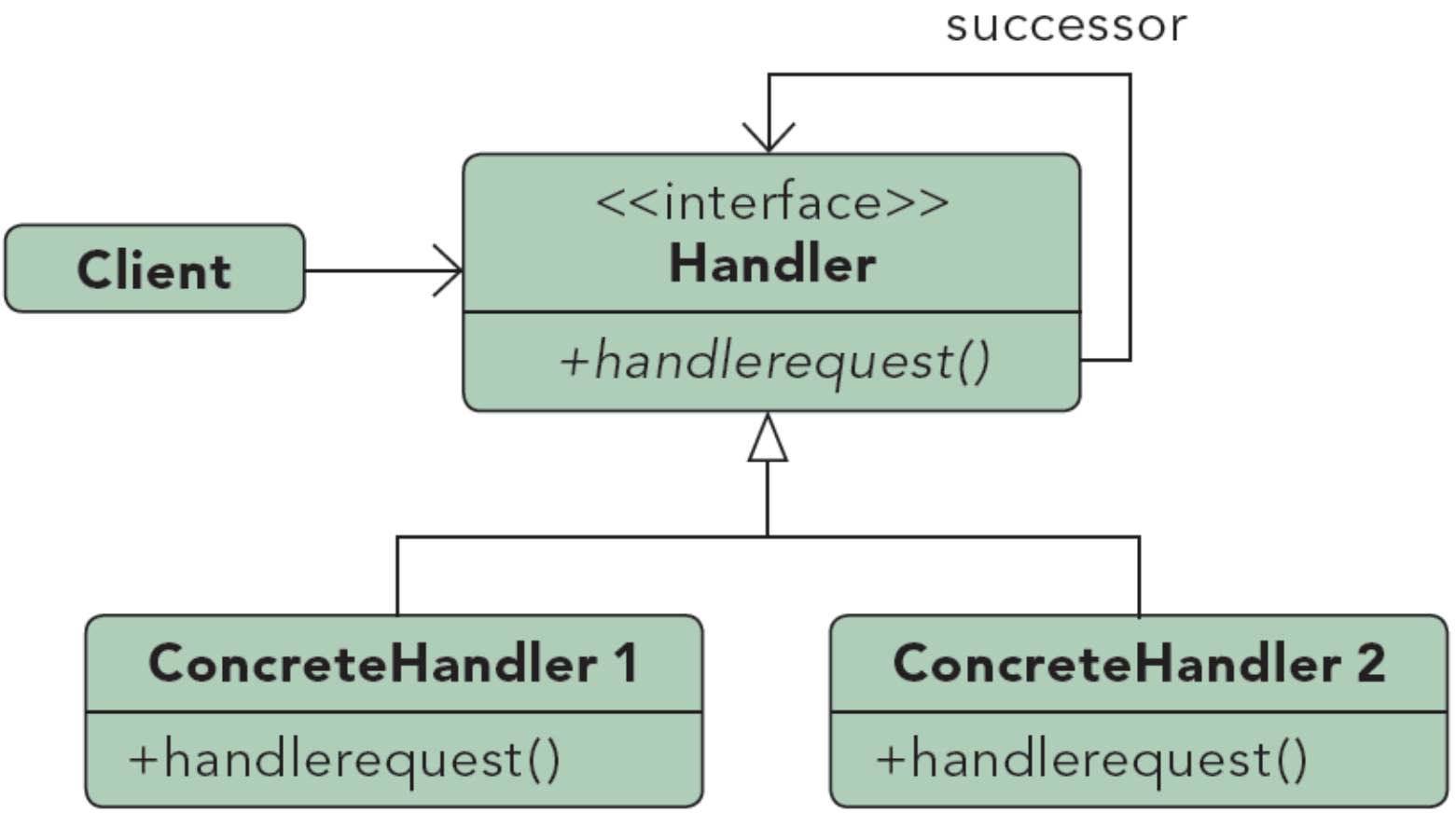
Patrones de comportamiento

Chain of responsibility

- Evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición.
- Este patrón puede ser utilizado cuando:
  - Más de un objeto puede manejar una petición, y el manejador no se conoce a priori.
  - Se quiere enviar una petición a un objeto entre varios sin especificar explícitamente el receptor.
  - El conjunto de objetos que puede tratar una petición debería ser especificado dinámicamente.

Patrones de comportamiento

Chain of responsibility



Ejemplo
Chain of responsibility

Patrones de comportamiento

Command

- Permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además se facilita la parametrización de los métodos.
- Este patrón puede ser utilizado para (1/2):
  - Facilitar la parametrización de las acciones a realizar.
  - Independizar el momento de petición del de ejecución.

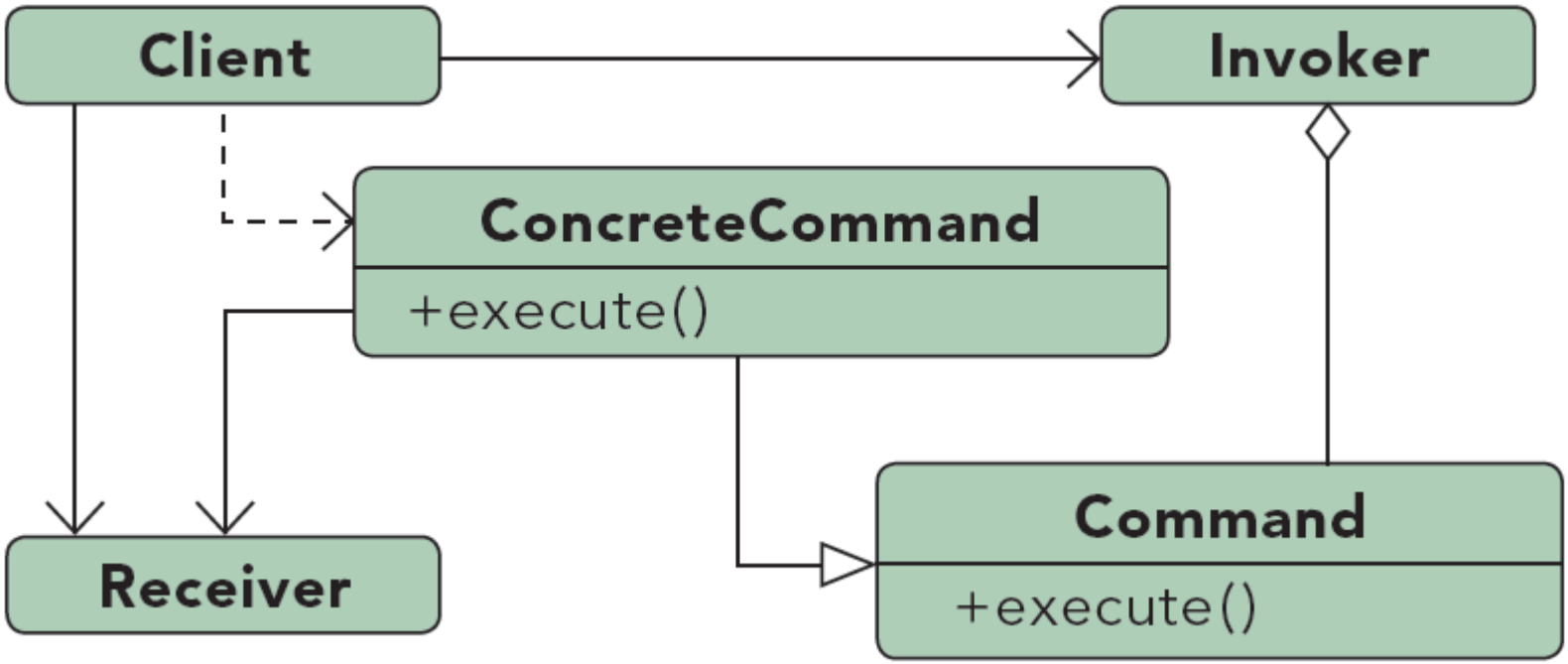
Patrones de comportamiento

Command

- Este patrón puede ser utilizado para (2/2):
  - Implementar CallBacks, especificando que órdenes queremos que se ejecuten en ciertas situaciones.
  - Soportar el "deshacer".
  - Desarrollar sistemas utilizando órdenes de alto nivel que se construyen con operaciones sencillas (primitivas).

Patrones de comportamiento

Command



Ejemplo Command

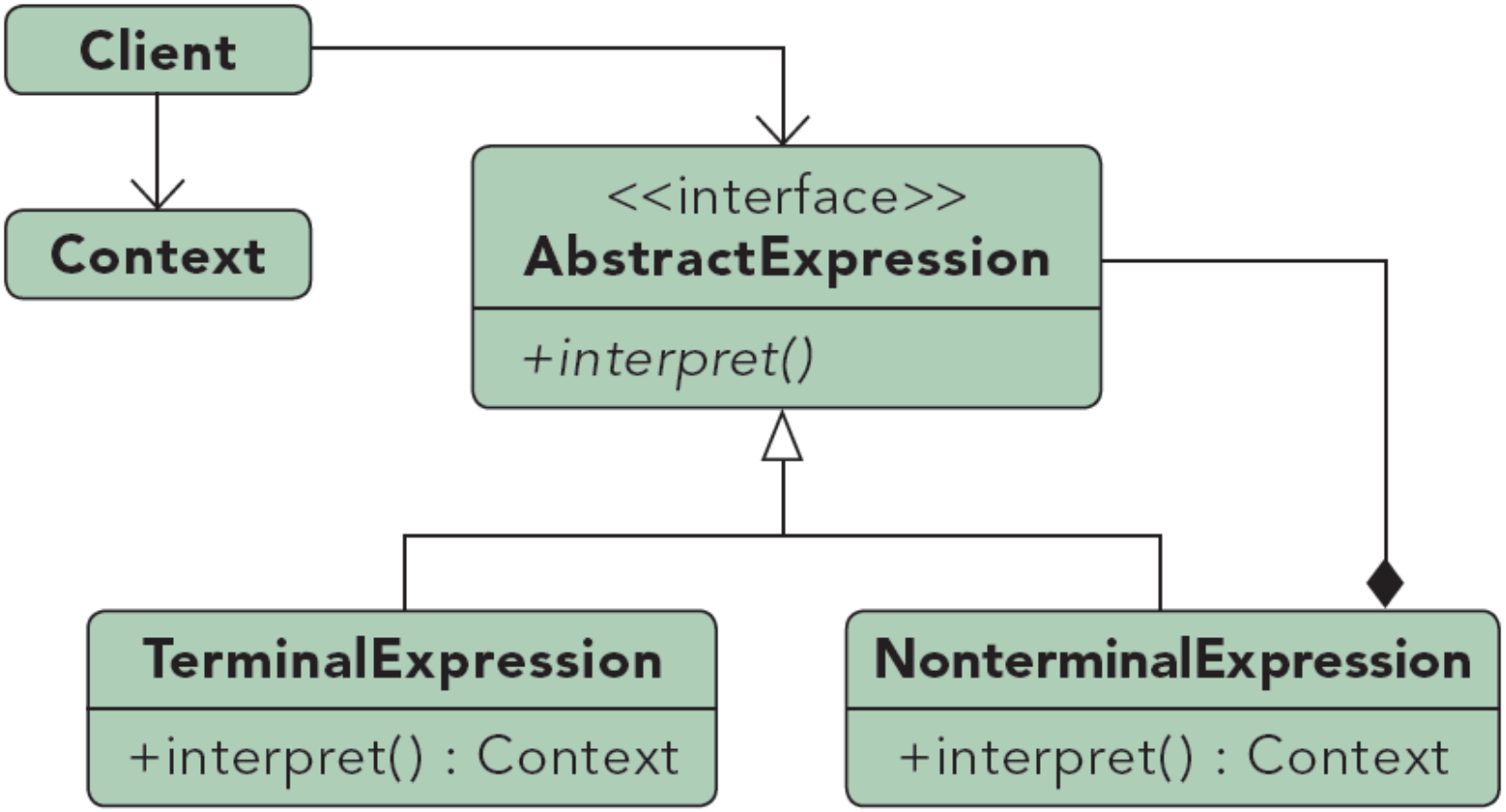
Patrones de comportamiento

Interpreter

- Define una representación para su gramática junto con un intérprete del lenguaje.
- Este patrón puede ser utilizado cuando:
  - Se quiere definir un lenguaje para representar expresiones regulares que representen cadenas a buscar dentro de otras cadenas.
  - Se pretende definir un lenguaje que permita representar las distintas instancias de una familia de problemas.

Patrones de comportamiento

Interpreter



Ejemplo Interpreter

Patrones de comportamiento

Iterator

- Define una interfaz que declara los métodos necesarios para acceder secuencialmente a un grupo de objetos de una colección.

- Algunos de los métodos comunes que se definen en la interfaz Iterador son:
  - Primero()
  - Siguiente()
  - HayMas()
  - ElementoActual()

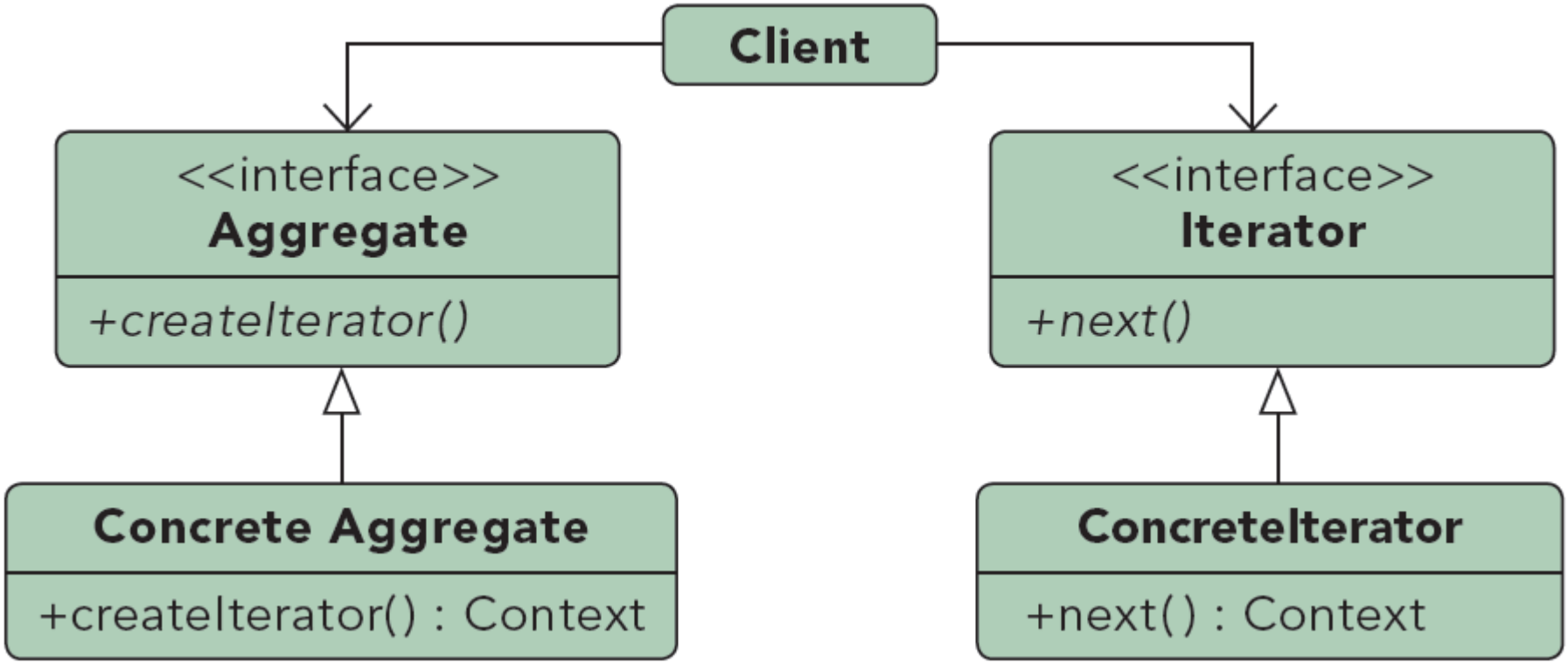
Patrones de comportamiento

Iterator

- Este patrón puede ser utilizado cuando:
  - Se necesita tener acceso a elementos sin tener acceso a toda la representación.
  - Se pretenden hacer recorridos múltiples o concurrentes de los elementos.
  - Existen diferencias sutiles entre los detalles de implementación de varios iteradores.

Patrones de comportamiento

Iterator



Ejemplo Iterator

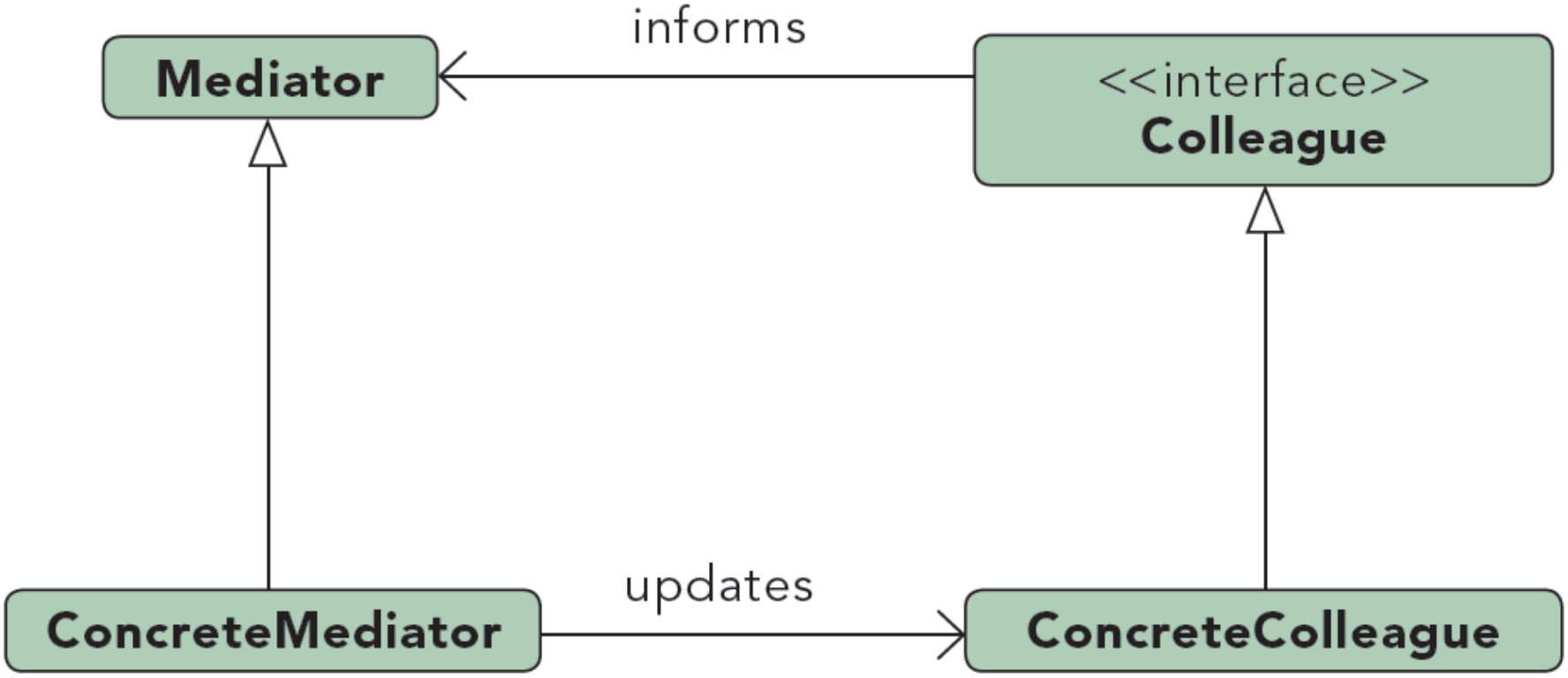
Patrones de comportamiento

Mediator

- Define un objeto que encapsula la manera en que interactúan un conjunto de objetos entre ellos.
- Este patrón puede ser utilizado cuando:
  - La comunicación entre los conjuntos de objetos está bien definido y es complejo.
  - Existen demasiadas relaciones y se necesita un punto común de control o comunicación.

Patrones de comportamiento

Mediator



Ejemplo Mediator

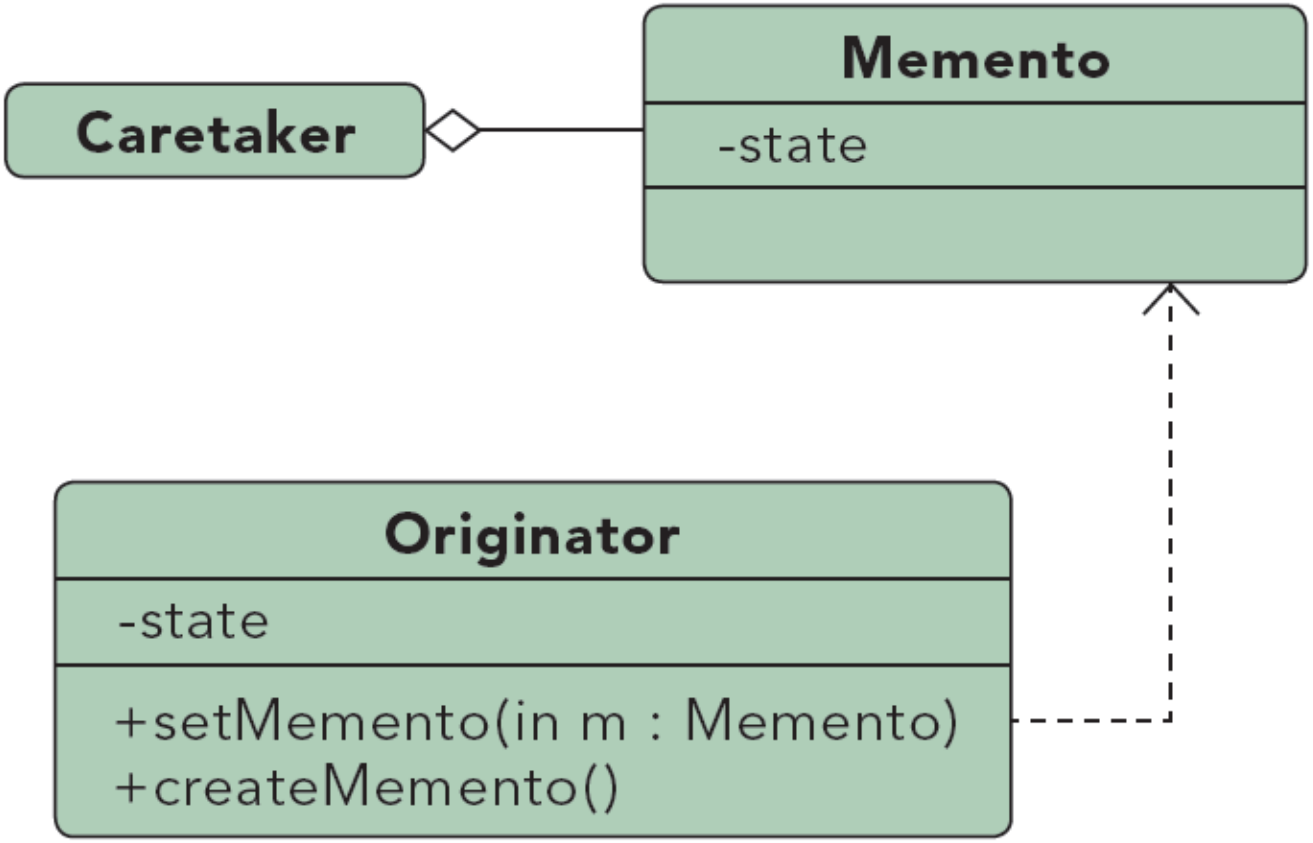
Patrones de comportamiento

Memento

- Su finalidad es almacenar el estado de un objeto (o del sistema completo) en un momento dado de manera que se pueda restaurar en ese punto de manera sencilla.
- Este patrón puede ser utilizado cuando:
  - El estado interno de un objeto debe ser guardado y restaurado en un momento posterior.
  - El estado interno no se puede exponer mediante interfaces sin exponer la implementación.

Patrones de comportamiento

Memento



Ejemplo Memento

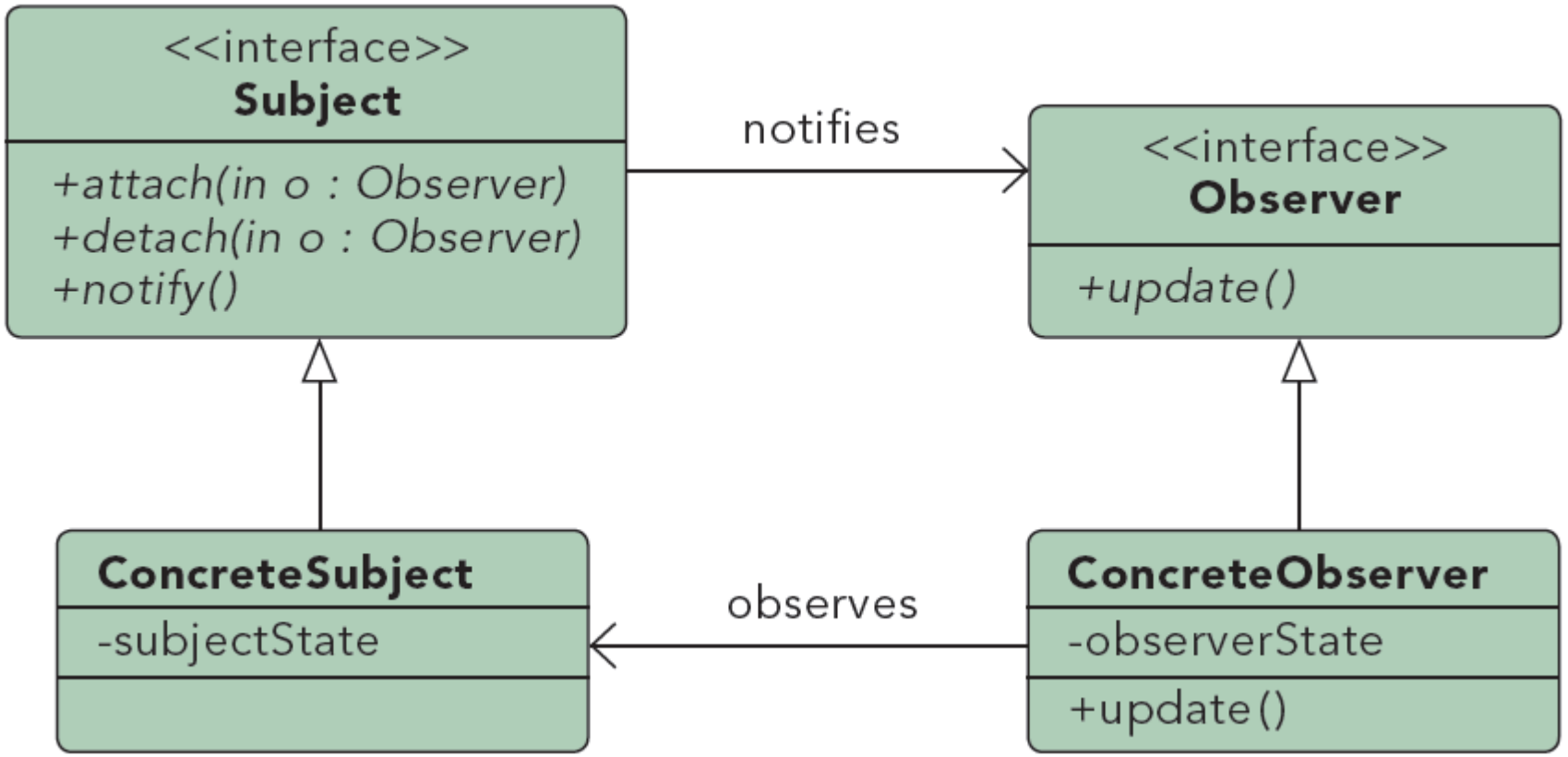
Patrones de comportamiento

Observer

- Define una dependencia del tipo uno-a-muchos entre objetos, de manera que **cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.**
- Este patrón puede ser utilizado cuando:
  - Se necesita consistencia entre clases relacionadas, pero con independecia.
  - Los cambios de estado en uno o más objetos deben dar lugar a comportamiento en otros objetos

Patrones de comportamiento

Observer



Ejemplo Observer

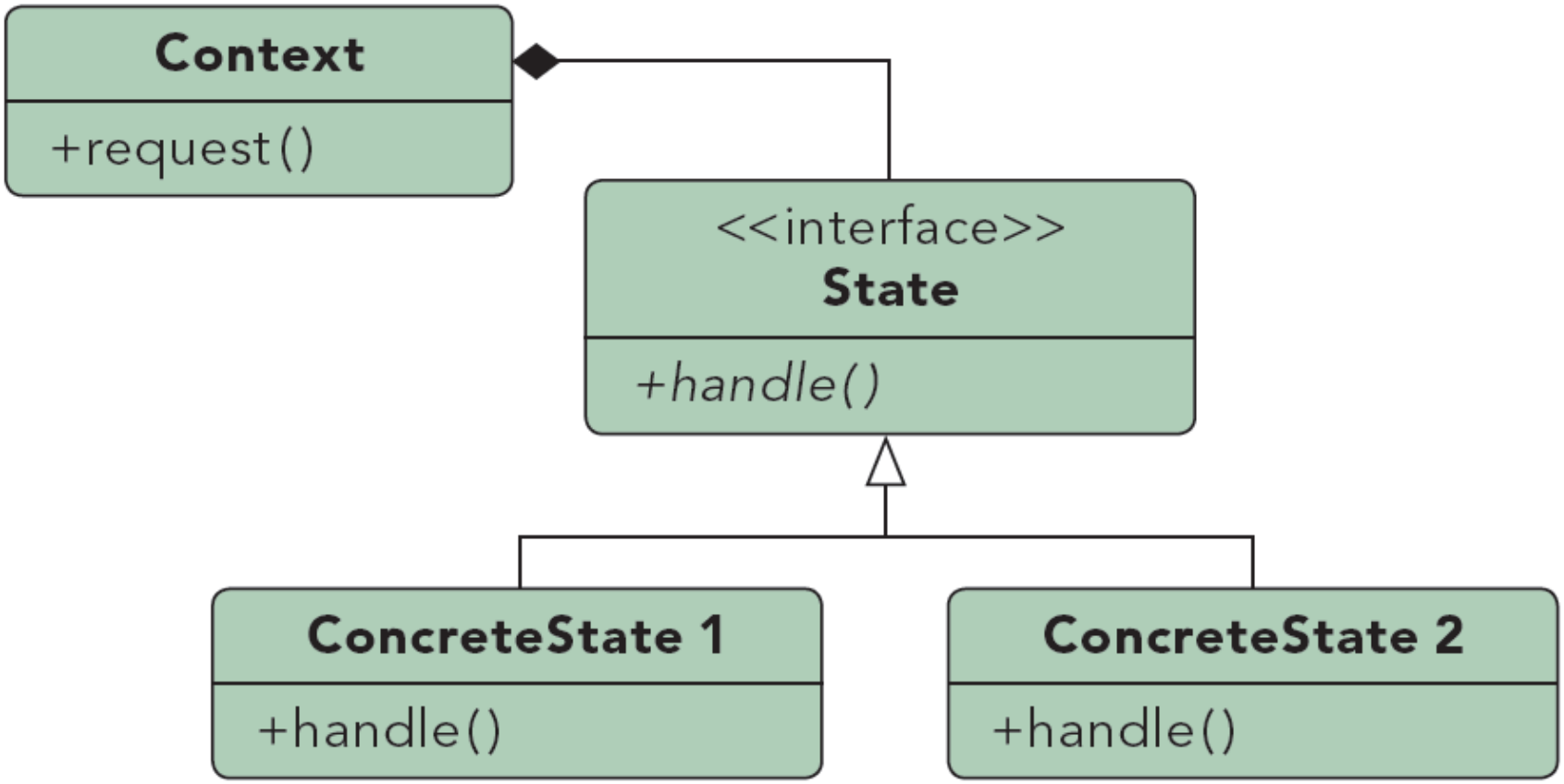
Patrones de comportamiento

State

- Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.
- Ejemplo: una alarma puede tener diferentes estados: *desactivada*, *activada*, *en configuración, etc.* En este caso se puede definir una interfaz *Estado_Alarma*, y luego definir los diferentes estados.
- Este patrón puede ser utilizado cuando se permite a un objeto alterar su comportamiento según el estado interno en que se encuentre.

Patrones de comportamiento

State



Ejemplo State

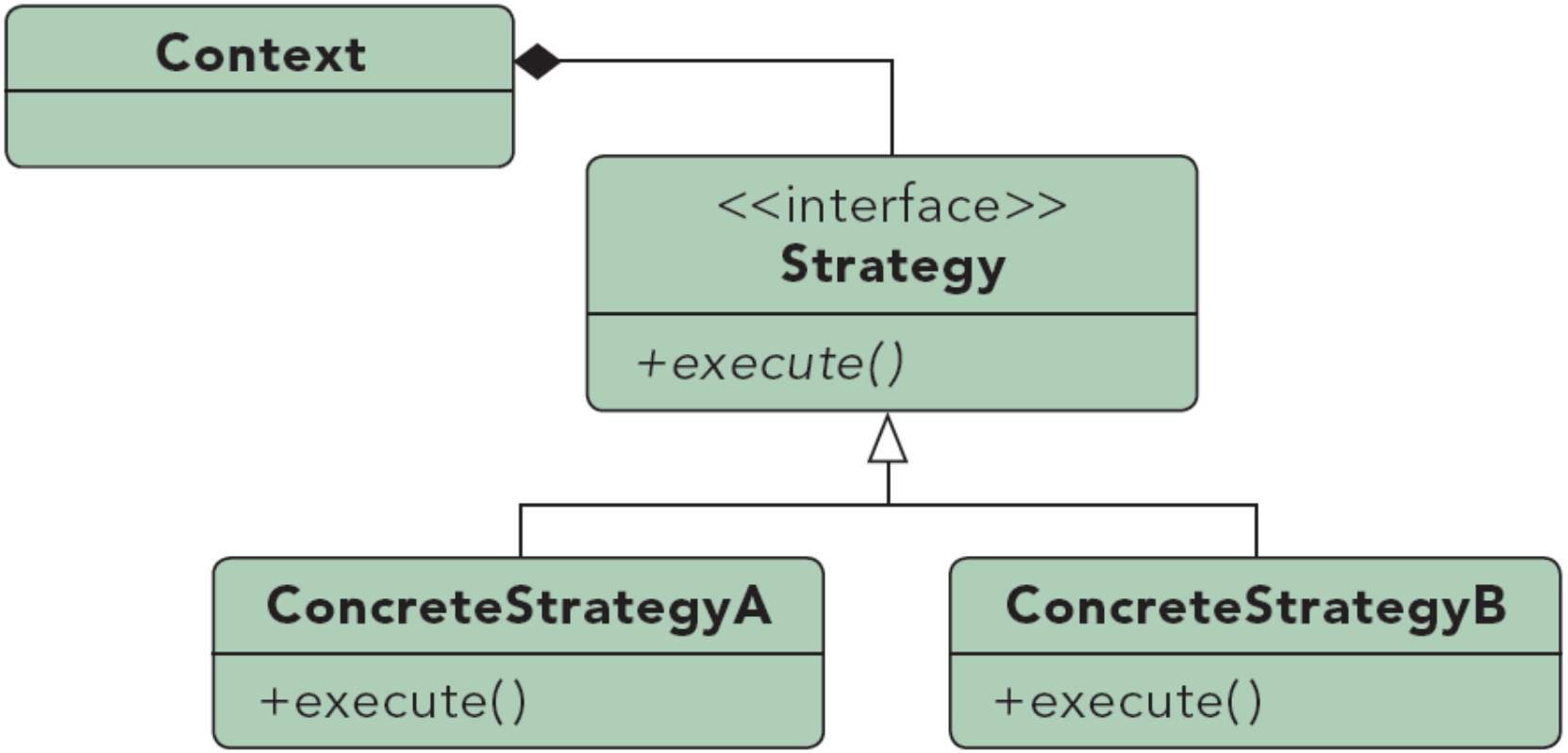
Patrones de comportamiento

Strategy

- Permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.
- Este patrón puede ser utilizado cuando:
  - La única diferencia entre muchas clases relacionadas es su comportamiento.
  - Se requieren múltiples versiones de un algoritmo.
  - El comportamiento de una clase debe ser definido en tiempo de ejecución.

Patrones de comportamiento

Strategy



Ejemplo Strategy

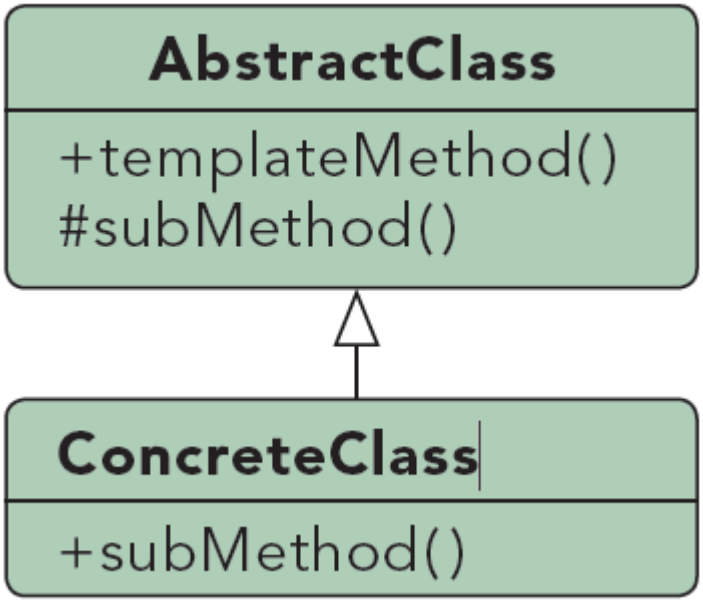
Patrones de comportamiento

Template method

- Define dentro de una operación de una superclase, los pasos de un algoritmo, de forma que todos o parte de estos pasos son redefinidos en las subclases herederas de la citada superclase.
- Este patrón puede ser utilizado cuando:
  - El comportamiento común entre subclases debe estar localizado en una clase común.
  - Las clases padre deben ser capaces de invocar de manera uniforme en el comportamiento de sus subclases.

Patrones de comportamiento

Template method



Ejemplo
Template method

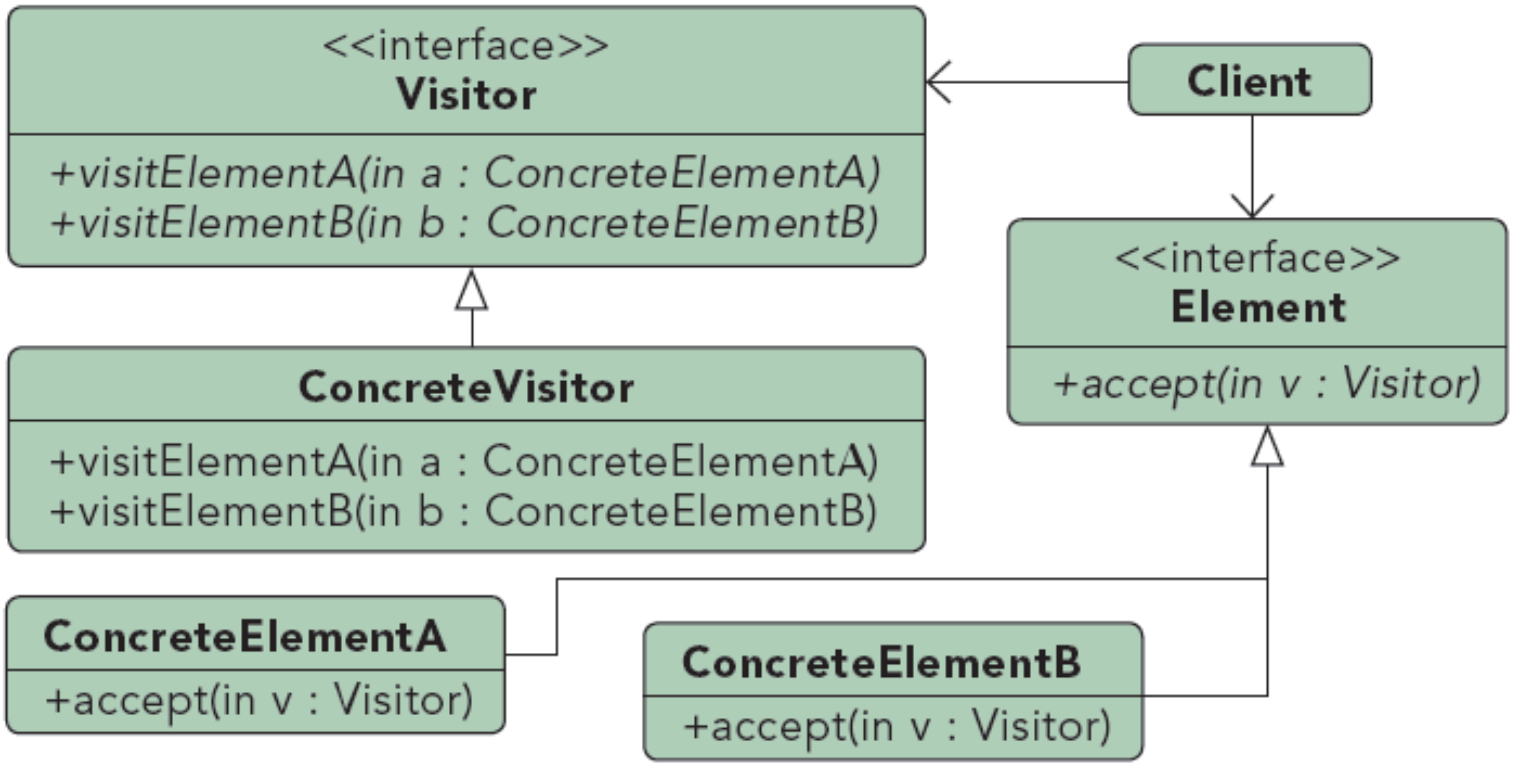
Patrones de comportamiento

Visitor

- Permite aplicar una o más operaciones a un conjunto de objetos en tiempo de ejecución, desacoplando dichas operaciones de la estructura del objeto.
- Este patrón puede ser utilizado:
  - Cuando la estructura del objeto no se puede cambiar, pero sí las operaciones que realiza.
  - Ampliamente en intérpretes, compiladores y procesadores de lenguajes, en general.

Patrones de comportamiento

Visitor

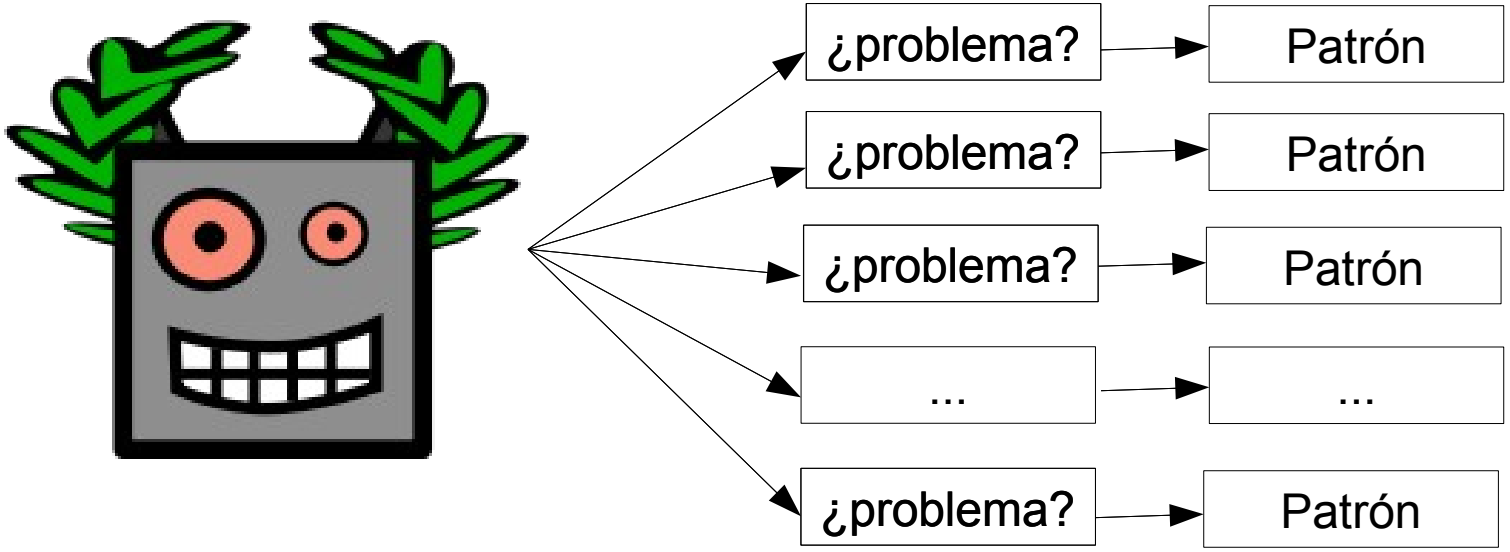


Ejemplo Visitor

Conclusiones

¡Eureka!

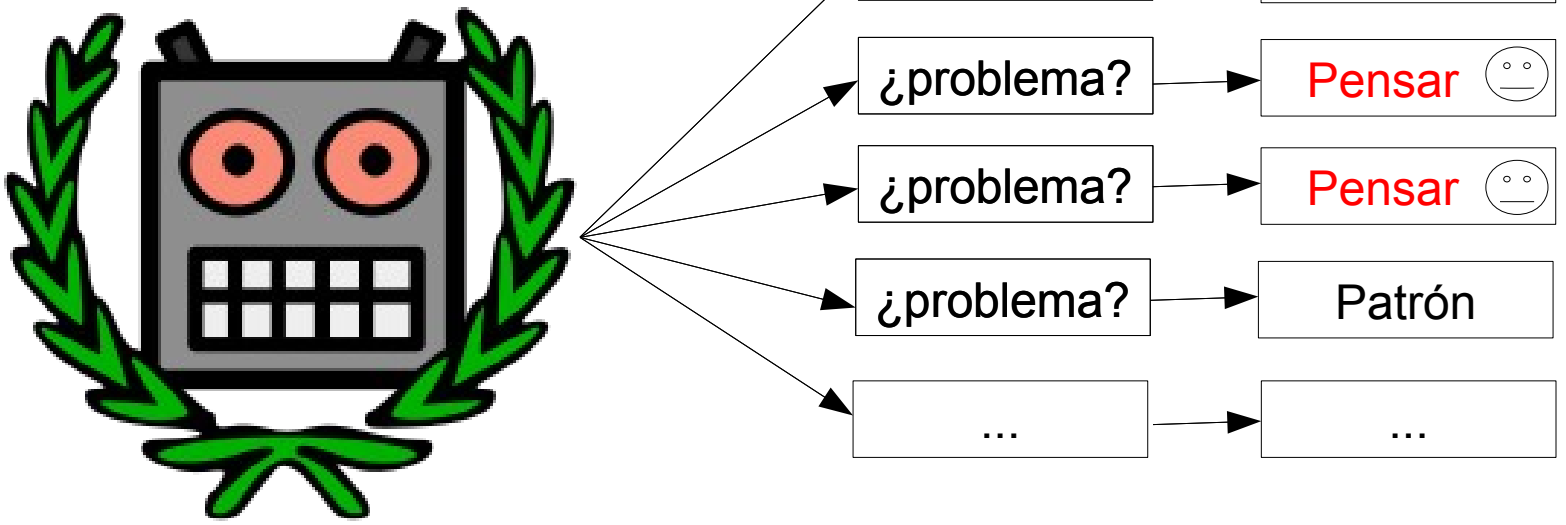
Conozco los patrones de diseño...
ya puedo resolver TODOS mis problemas



Conclusiones

¡Noooooooooooo!

No intentar aplicar los patrones de diseño “con calzador”



Patrones de diseño

Referencias

- Wikipedia: http://en.wikipedia.org/wiki/Design_pattern_(computer_science)
- Patrones con ejemplos en Java: http://sourcemaking.com/design_patterns
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns CD. Addison-Wesley 1998.
- Jaroslav Tulach. Practical API Design: Confessions of a Java Framework Architect. APRESS 2008.

Patrones de diseño

Referencias

- Timothy G. Mattson, Beverly A. Sanders, Berna L. Massingill. Patterns for Parallel Programming. Addison-Wesley 2008.
- Dzone (diagramas de clases, definiciones, etc.): http://java.dzone.com/articles/design-patterns-factory
- Los ejemplos se han adaptados de la página: http://www.fluffycat.com/Java-Design-Patterns/

Patrones de diseño

FIN

Iker Canarias
iker.canarias (gmail)