IIC2113 - Diseño Detallado de Software

CLASE 4 - Patrones de diseño y arquitectura

Pontificia Universidad Católica de Chile
2020-2

Índice

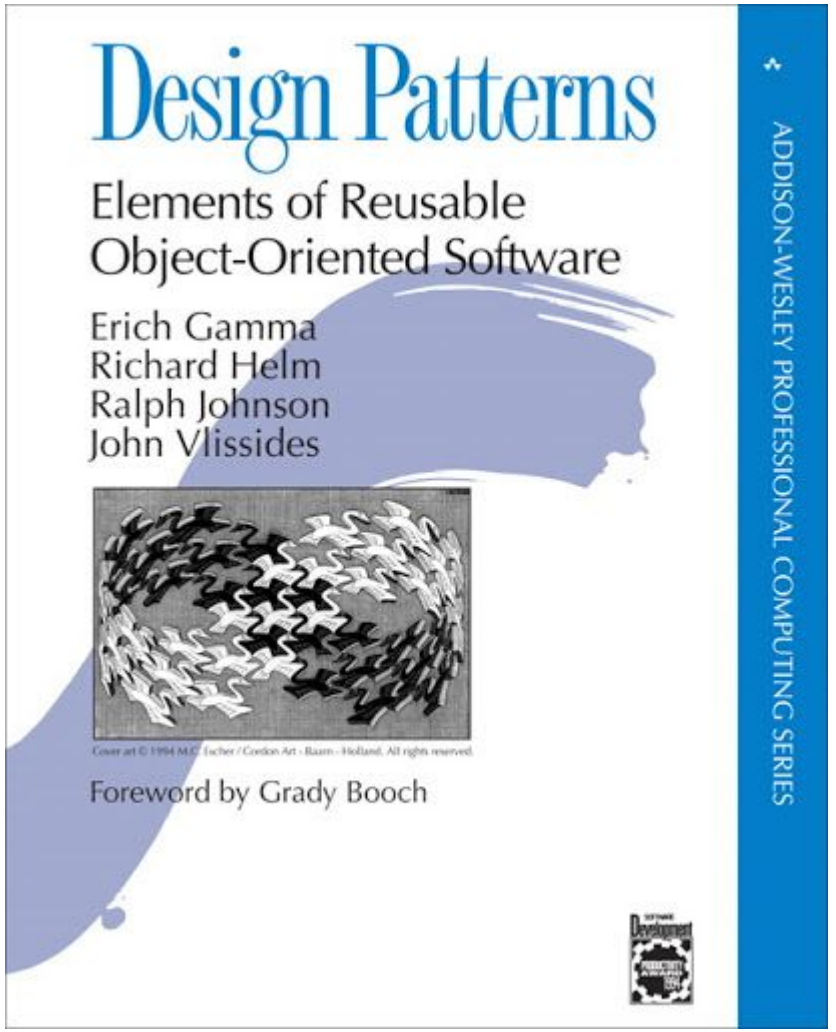
01. Patrones de Diseño

Qué es, tipos y revisión de patrones

¿Qué es un patrón de diseño?

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, In such a way that you can use this solution a million times over, without ever doing it the same way twice".

- Christopher Alexander, A Pattern Language (1977)



¿Qué es un patrón de diseño?

Para efectos del curso un patrón de diseño son descripciones que comunican objetos y clases, las cuales se personalizan con el fin de resolver un problema generalizado de diseño en un contexto particular.

→ Veremos algunos de estos patrones aplicados a OOP (pero existen más y no solo para OOP).

Un patrón de diseño se caracteriza por 3 componentes:

Tipos de patrones de diseño

Según proposito:

- **Creacionales**: Se centran en la creación, composición y representación de los objetos.
- **Estructurales**: Se centran en cómo los objetos se organizan e integran en un sistema.
- **De comportamiento**: Se centran en las interacciones y responsabilidades entre objetos.

Según scope:

- **Clase**: Relaciones entre clases y subclases, a través de herencia generalmente, por lo que sus relaciones son fijas durante la compilación.
- **Objeto**: Relaciones entre objetos, por lo que son dinámicos en el tiempo de ejecución.

Tipos de patrones de diseño

		Proposito		
		Creacional	**Estructural**	**De comportamiento**
Scope	**Clase**	- Factory Method	- Adapter	- Interpreter - Template Method
	Objeto	- Abstract Factory - Builder - Prototype - Singleton	- Adapter - Bridge - Composite - Decorator - Facade - Flyweight - Proxy	- Chain of Responsibility - Command - Iterator - Mediator - Memento - Observer - State - Strategy - Visitor

https://www.dofactory.com

Los ejemplos prácticos los veremos desde dofactory.

Tipos de patrones de diseño

		Proposito		
		Creacional	**Estructural**	**De comportamiento**
Scope	**Clase**	- Factory Method	- Adapter	- Interpreter - Template Method
	Objeto	- Abstract Factory - Builder - Prototype - Singleton	- Adapter - Bridge - Composite - Decorator - Facade - Flyweight - Proxy	- Chain of Responsibility - Command - Iterator - Mediator - Memento - Observer - State - Strategy - Visitor

Patrones creacionales: Factory Method

Define una interfaz para la creación de un objeto, pero delega a las subclases que decidan qué clase instanciar.

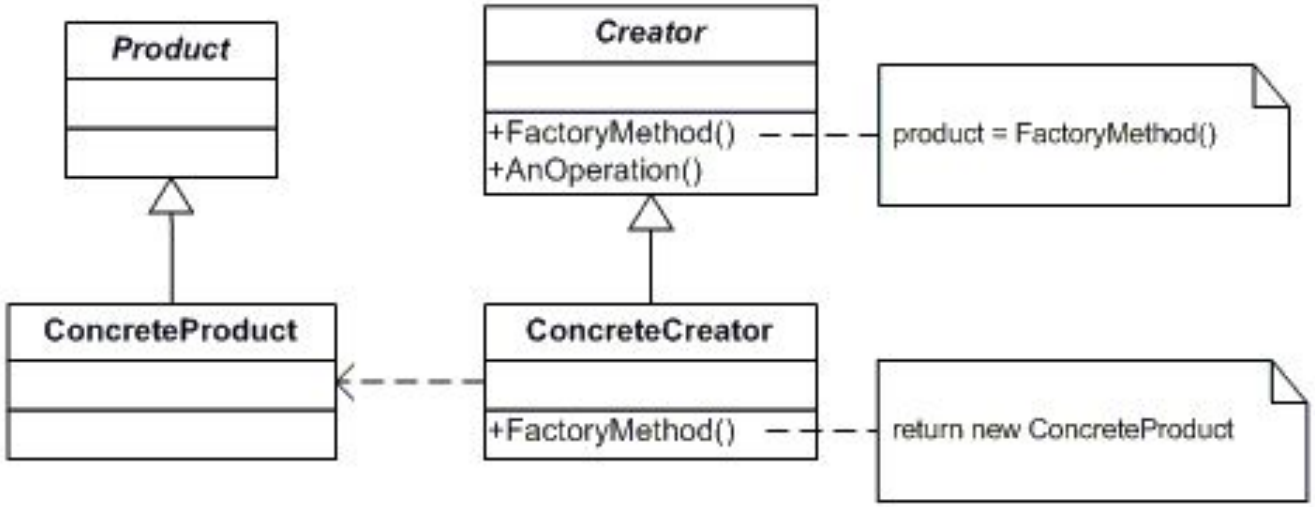
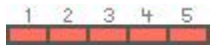


Diagrama y ejemplo en https://www.dofactory.com/net/factory-method-design-pattern

Patrones creacionales: Factory Method



¿Cuándo se utiliza?

- Una clase no puede anticipar las clases de los objetos que debe crear.
- Una clase necesita permitir a sus subclases especificar la creación de objetos.
- Una clase delega la responsabilidad de crear objetos a sus subclases, para así encapsular lógica.

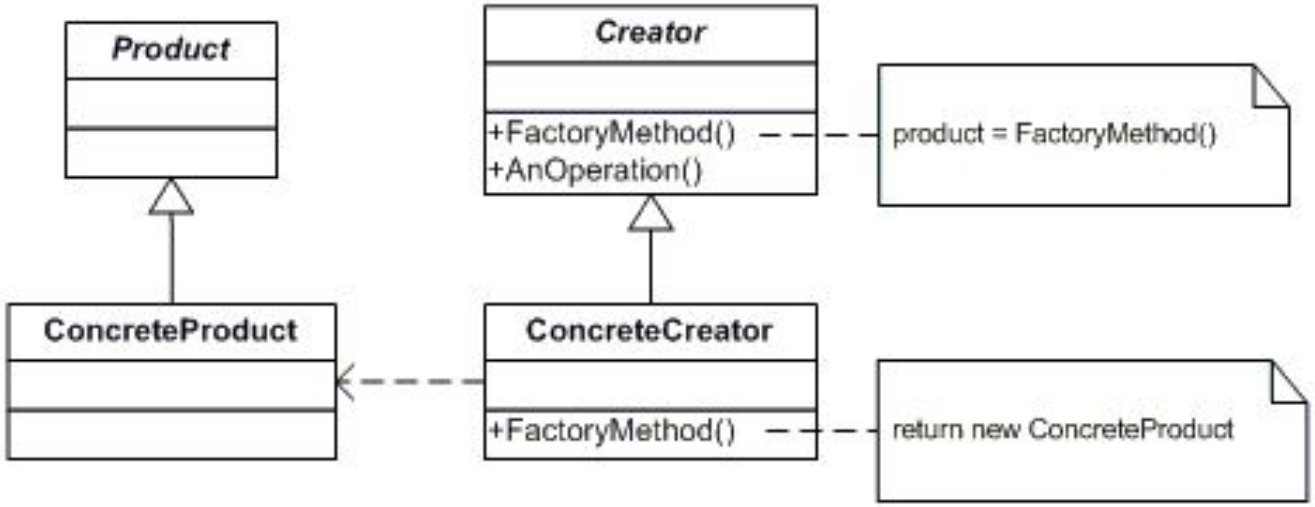
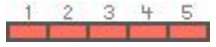


Diagrama y ejemplo en https://www.dofactory.com/net/factory-method-design-pattern

Tipos de patrones de diseño

		Proposito		
		Creacional	**Estructural**	**De comportamiento**
Scope	**Clase**	- Factory Method	- Adapter	- Interpreter - Template Method
	Objeto	- Abstract Factory - Builder - Prototype - Singleton	- Adapter - Bridge - Composite - Decorator - Facade - Flyweight - Proxy	- Chain of Responsibility - Command - Iterator - Mediator - Memento - Observer - State - Strategy - Visitor

Patrones creacionales: Abstract Factory



Provee una interfaz para la creación de familias de objetos relacionadas o dependientes sin especificar sus clases concretas.

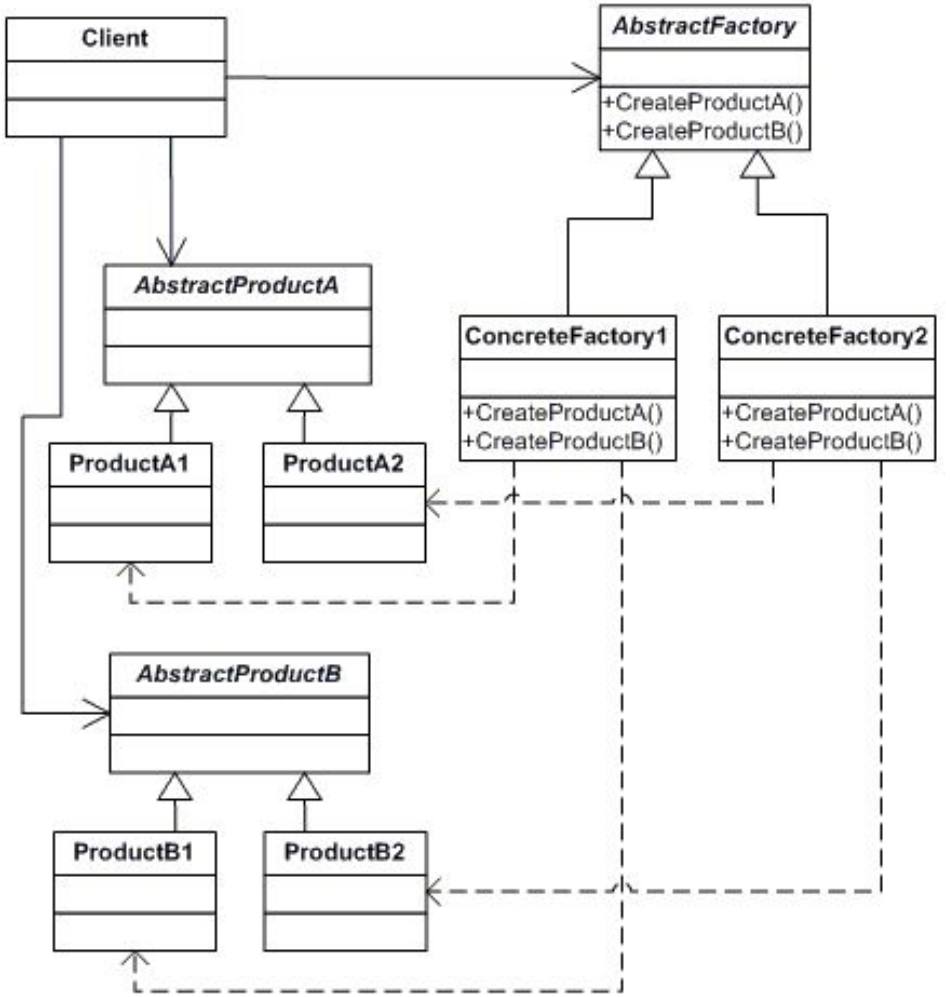


Diagrama y ejemplo en https://www.dofactory.com/net/abstract-factory-design-pattern

Patrones creacionales: Abstract Factory

¿Cuándo se utiliza?

- Un sistema debe ser independiente de los productos que crea, compone y representa.
- Un sistema debe ser configurado con una entre múltiples familias de productos.
- Una familia de productos relacionados está diseñada para ser utilizada en conjunto, y se debe forzar esta regla.
- Se desea proveer una librer´ıa de productos, pero ocultar su implementación.

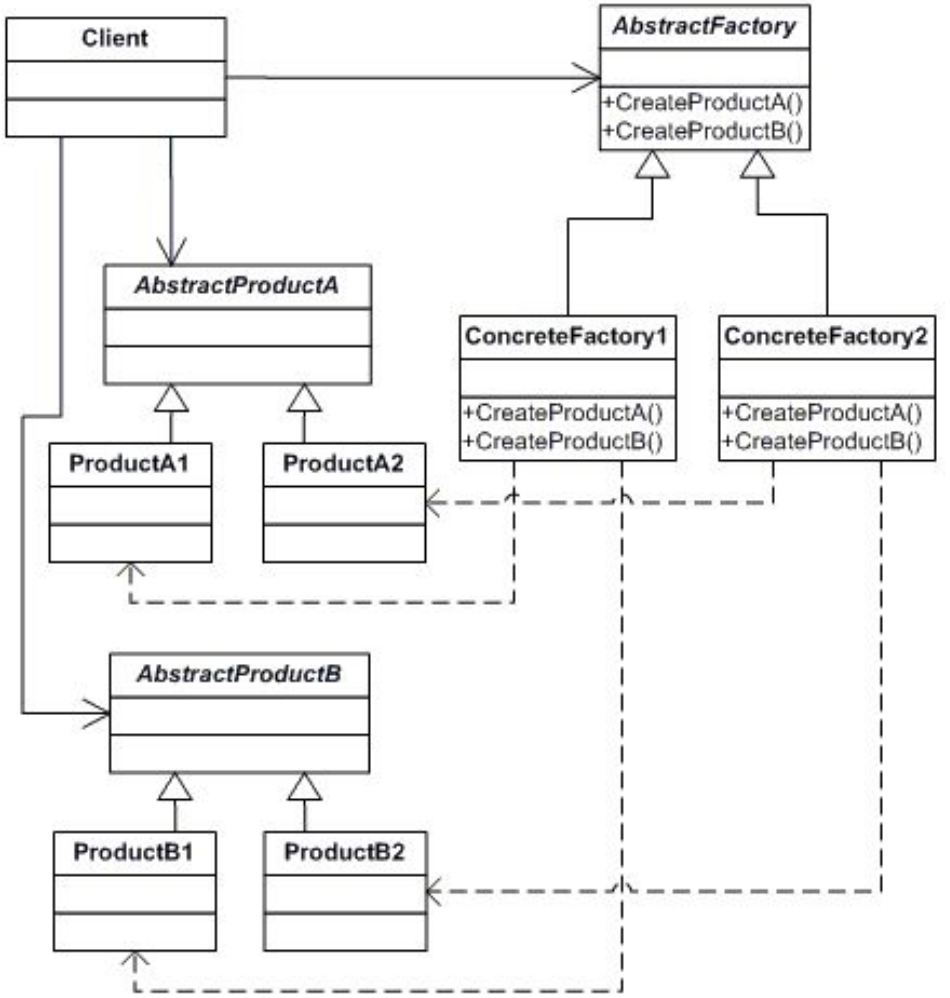


Diagrama y ejemplo en https://www.dofactory.com/net/abstract-factory-design-pattern

Patrones creacionales: Builder

Separa la construcción de un objeto complejo de su representación, para que así el mismo proceso de construcción pueda crear diferentes representaciones.

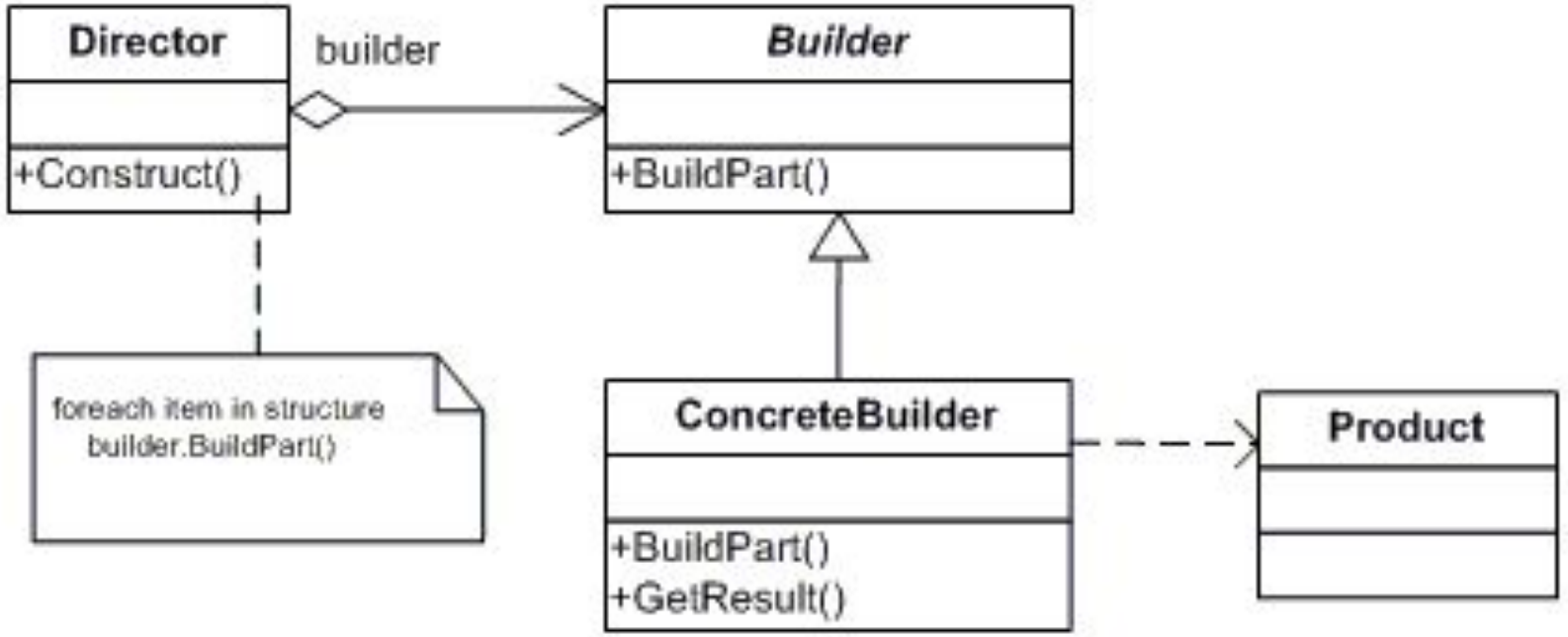


Diagrama y ejemplo en https://www.dofactory.com/net/builder-design-pattern

Patrones creacionales: Builder

¿Cuándo se utiliza?

- El algoritmo para crear objetos complejos debería ser independiente de las partes que componen el objeto, y de cómo se construyen.
- El proceso de construcción debe permitir diferentes representaciones del objeto que se construye.

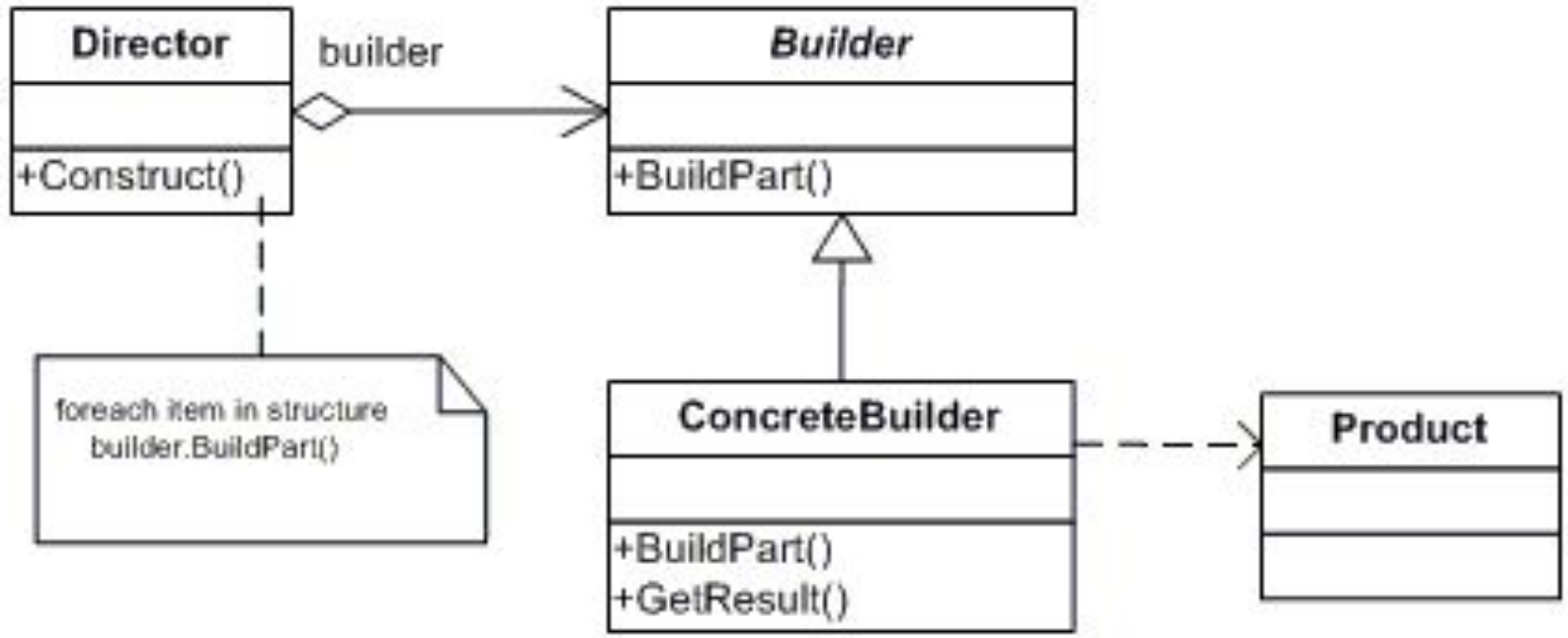


Diagrama y ejemplo en https://www.dofactory.com/net/builder-design-pattern

Patrones creacionales: Prototype

Especifica los tipos de objetos a crear utilizando prototipos, y crea objetos nuevos a través de copias de estos prototipos.

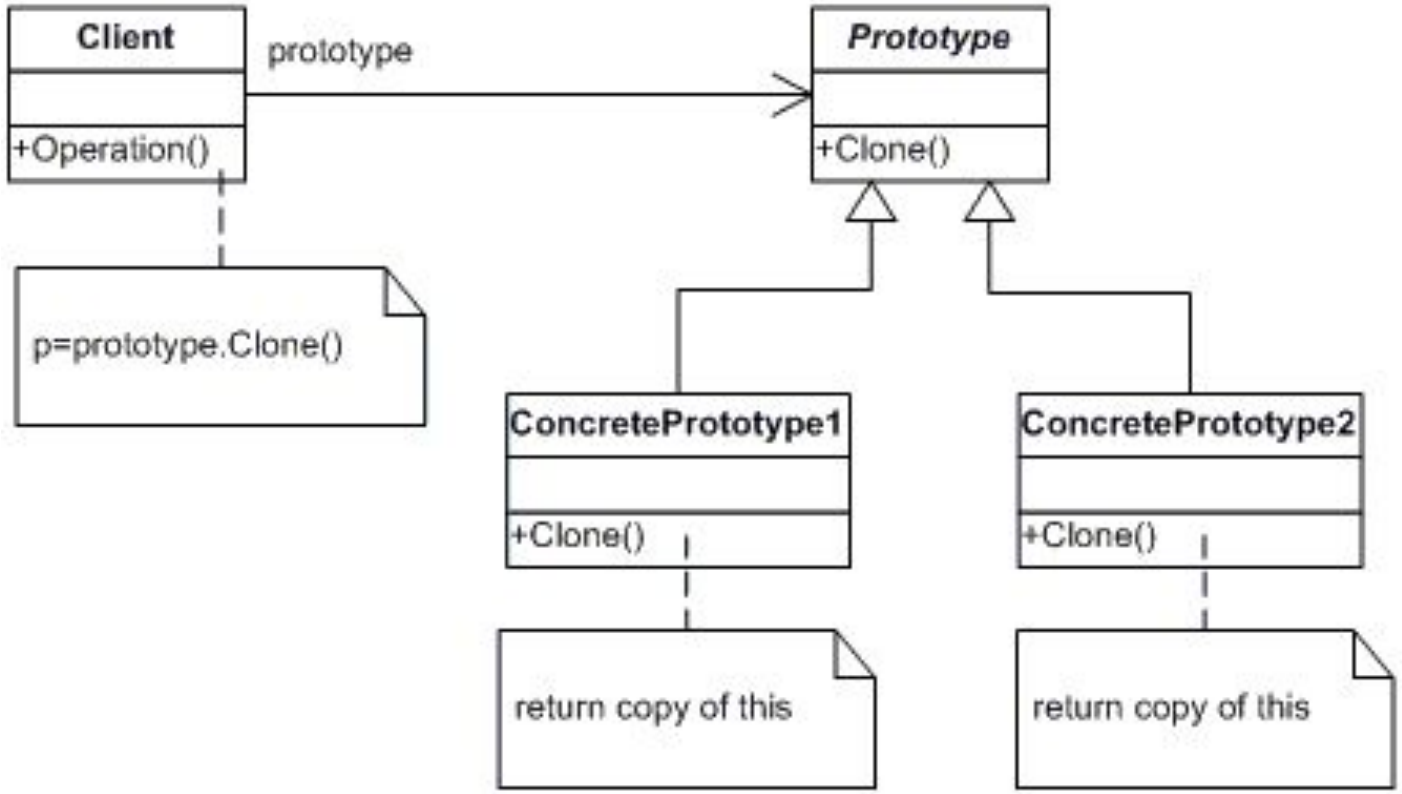


Diagrama y ejemplo en https://www.dofactory.com/net/prototype-design-pattern

Patrones creacionales: Prototype

¿Cuándo se utiliza?

- Las clases a crear se especifican en tiempo de ejecución
- Los objetos a crear son similares, y se pueden desprender de un objeto existente
- Instancias de una clase tienen una cantidad limitada de estados. Puede ser más conveniente instanciar una sola vez estos estados, para luego entregar copias.

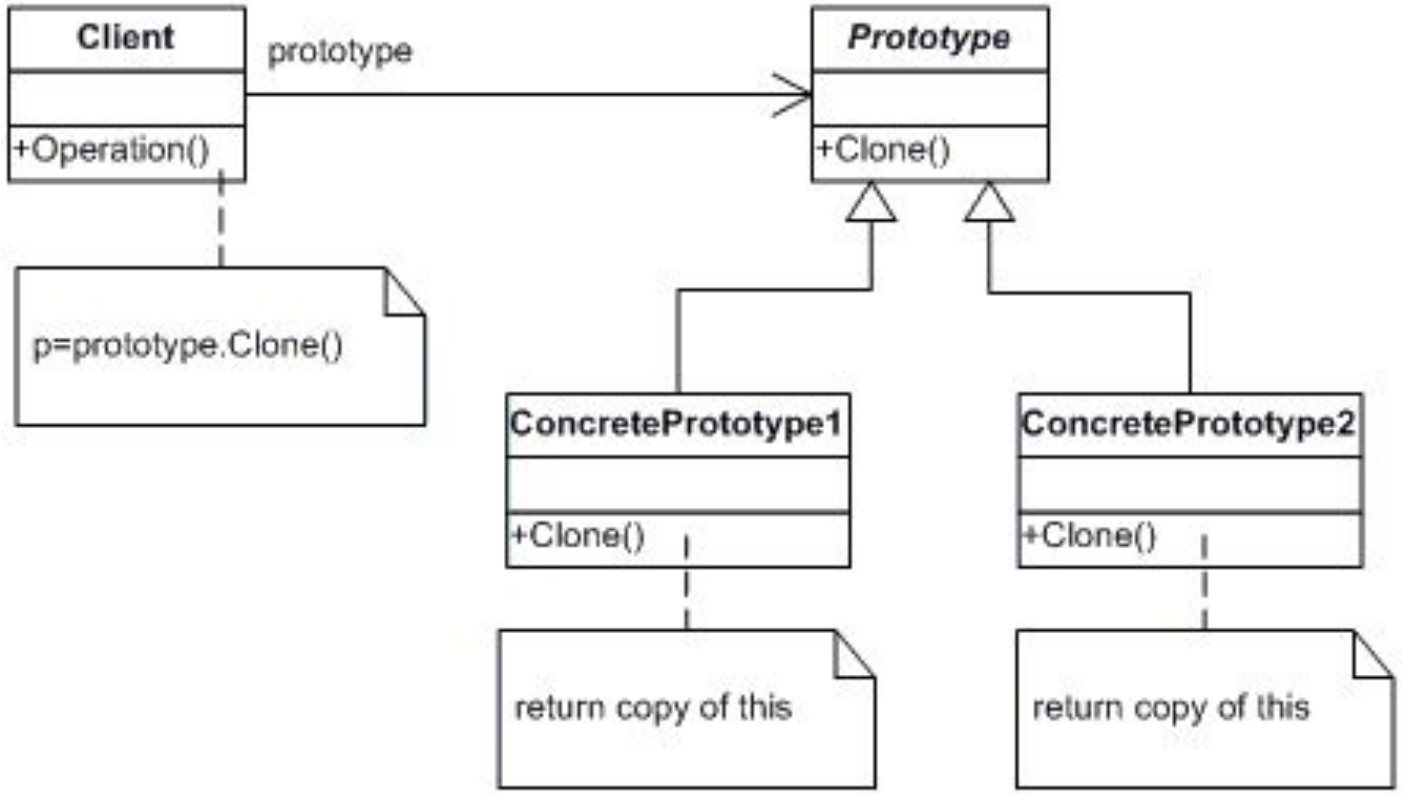


Diagrama y ejemplo en https://www.dofactory.com/net/prototype-design-pattern

Patrones creacionales: Singleton

Garantiza que una clase tenga solamente una instancia y provee un acceso global a la instancia.

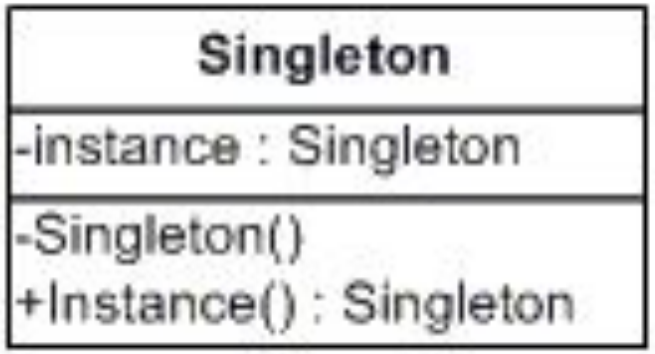


Diagrama y ejemplo en https://www.dofactory.com/net/singleton-design-pattern

Patrones creacionales: Singleton

¿Cuándo se utiliza?

- Debe haber exactamente una sola instancia de una clase, y puede ser accedida por distintos clientes

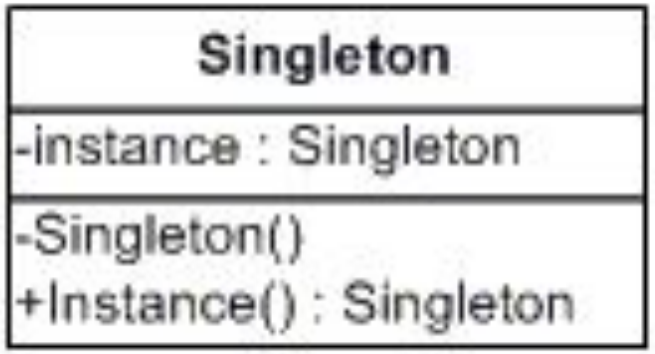


Diagrama y ejemplo en https://www.dofactory.com/net/singleton-design-pattern

Tipos de patrones de diseño

		Proposito		
		Creacional	**Estructural**	**De comportamiento**
Scope	**Clase**	- Factory Method	- Adapter	- Interpreter - Template Method
	Objeto	- Abstract Factory - Builder - Prototype - Singleton	- Adapter - Bridge - Composite - Decorator - Facade - Flyweight - Proxy	- Chain of Responsibility - Command - Iterator - Mediator - Memento - Observer - State - Strategy - Visitor

Patrones estructurales: Adapter

Convierte la interfaz de una clase en una interfaz que sea esperada por el cliente. Adapter permite a clases distintas interactuar, que sin su interfaz común serían incompatibles.

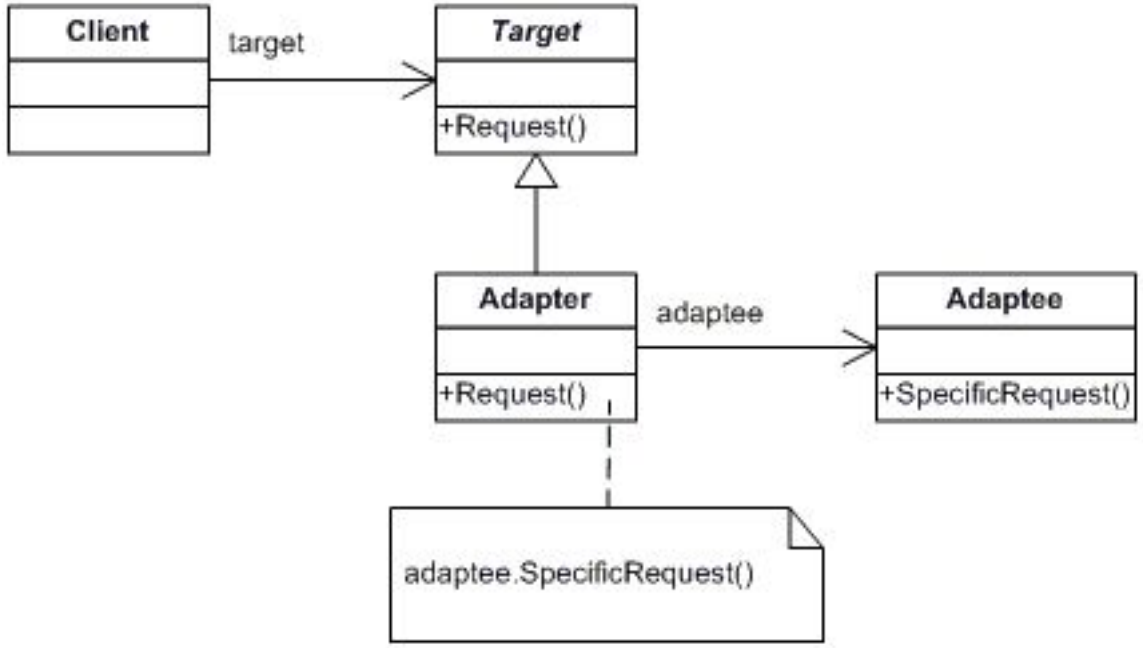


Diagrama y ejemplo en https://www.dofactory.com/net/adapter-design-pattern

Patrones estructurales: Adapter

¿Cuándo se utiliza?

- Se necesita utilizar una clase existente, pero su interfaz es incompatible con lo que se necesita.
- Se quiere crear una clase que coopere con otras clases que utilizan interfaces incompatibles.

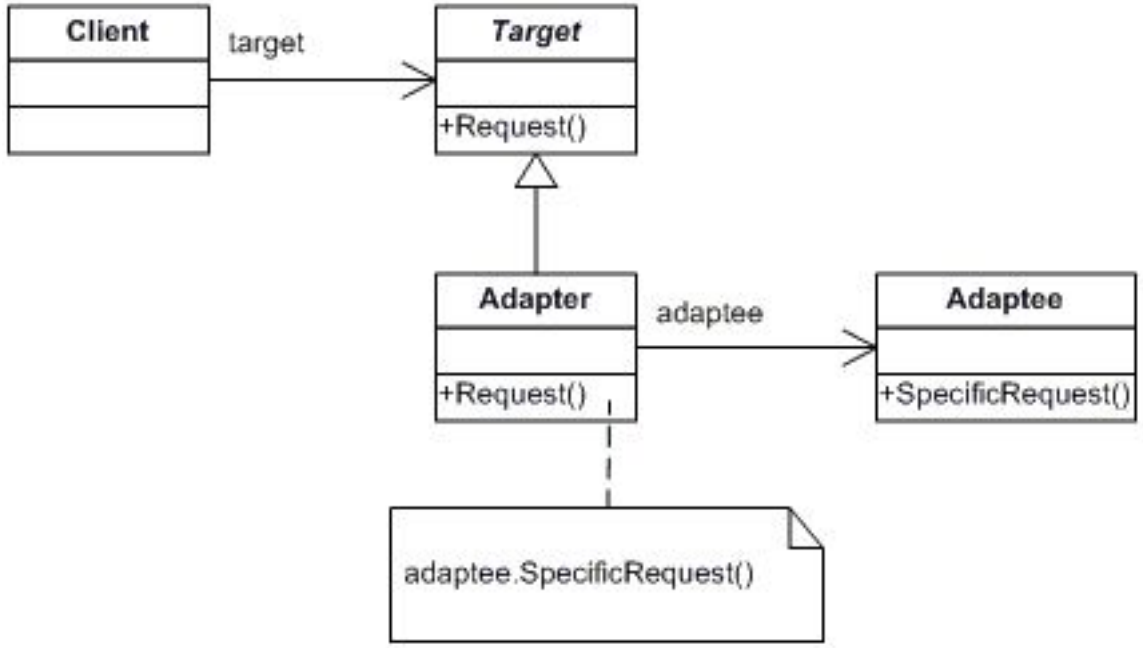


Diagrama y ejemplo en https://www.dofactory.com/net/adapter-design-pattern

Patrones estructurales: Bridge

Desacopla una abstracción de su implementación, para que ambas puedan variar independientemente

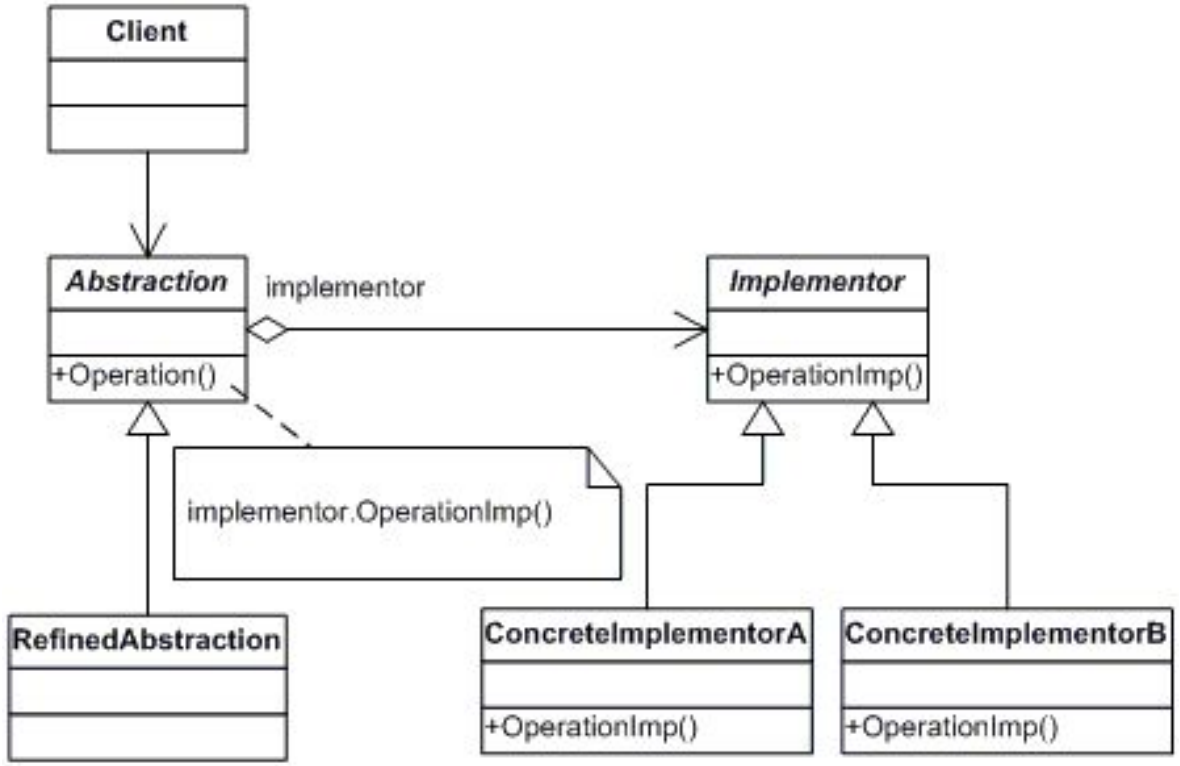


Diagrama y ejemplo en https://www.dofactory.com/net/bridge-design-pattern

Patrones estructurales: Bridge

¿Cuándo se utiliza?

- Se quiere evitar una unión permanente entre una abstracción y su implementación.
- Una abstracción y su implementación deberían ser extensibles.
- Cambios en la implementación de una abstracción no deberían tener impacto en los clientes.

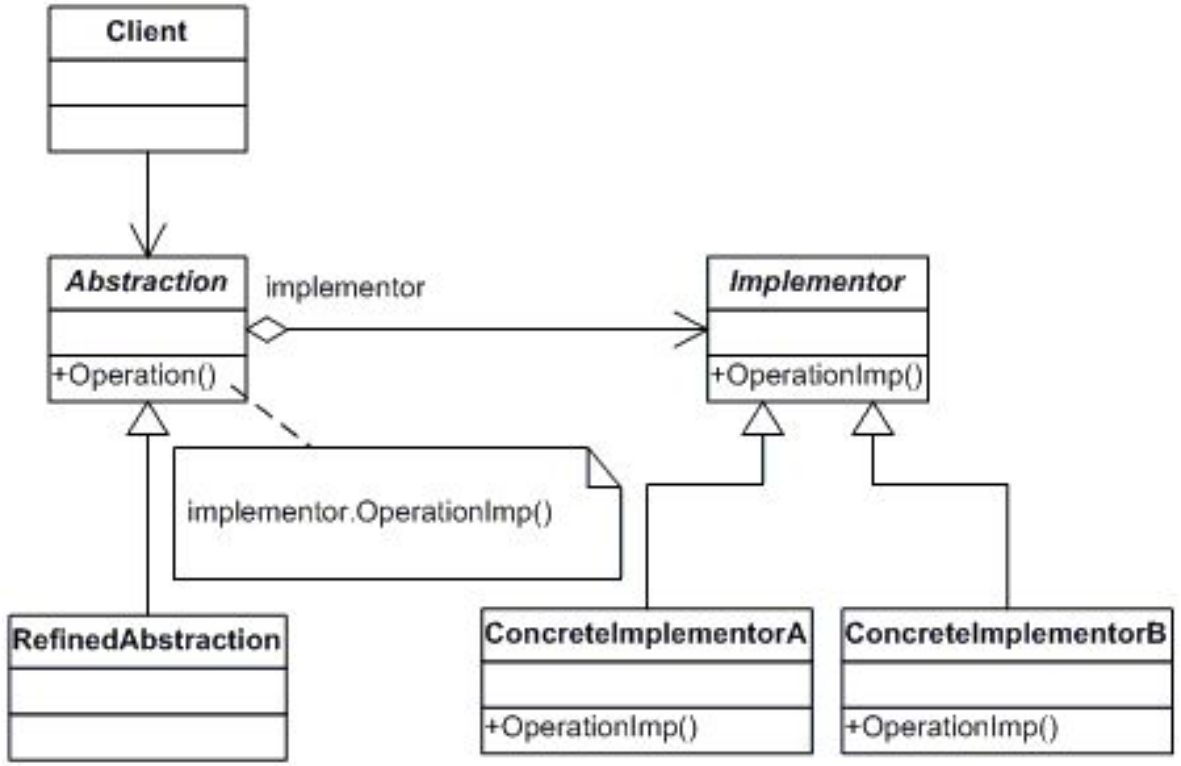
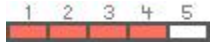


Diagrama y ejemplo en https://www.dofactory.com/net/bridge-design-pattern

Patrones estructurales: Composite



Compone objetos en estructuras de árboles para representar jerarquías completas. Composite permite a los clientes tratar objetos y composiciones de objetos de igual manera.

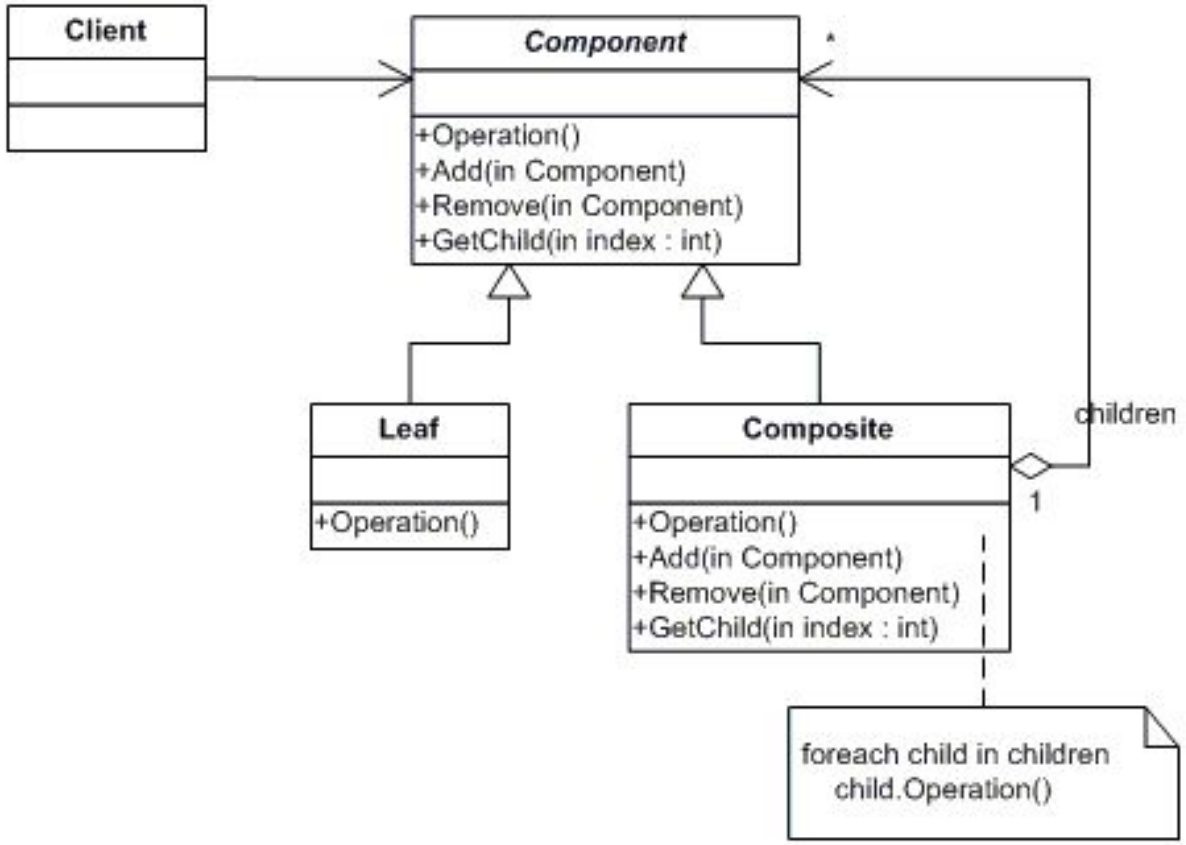


Diagrama y ejemplo en https://www.dofactory.com/net/composite-design-pattern

Patrones estructurales: Composite

¿Cuándo se utiliza?

- Se quiere representar jerarquías completas de objetos.
- Se quiere que los clientes puedan ignorar las diferencias entre objetos compuestos y objetos individuales. Los clientes tratarán a todos los objetos de la jerarquía de manera uniforme.

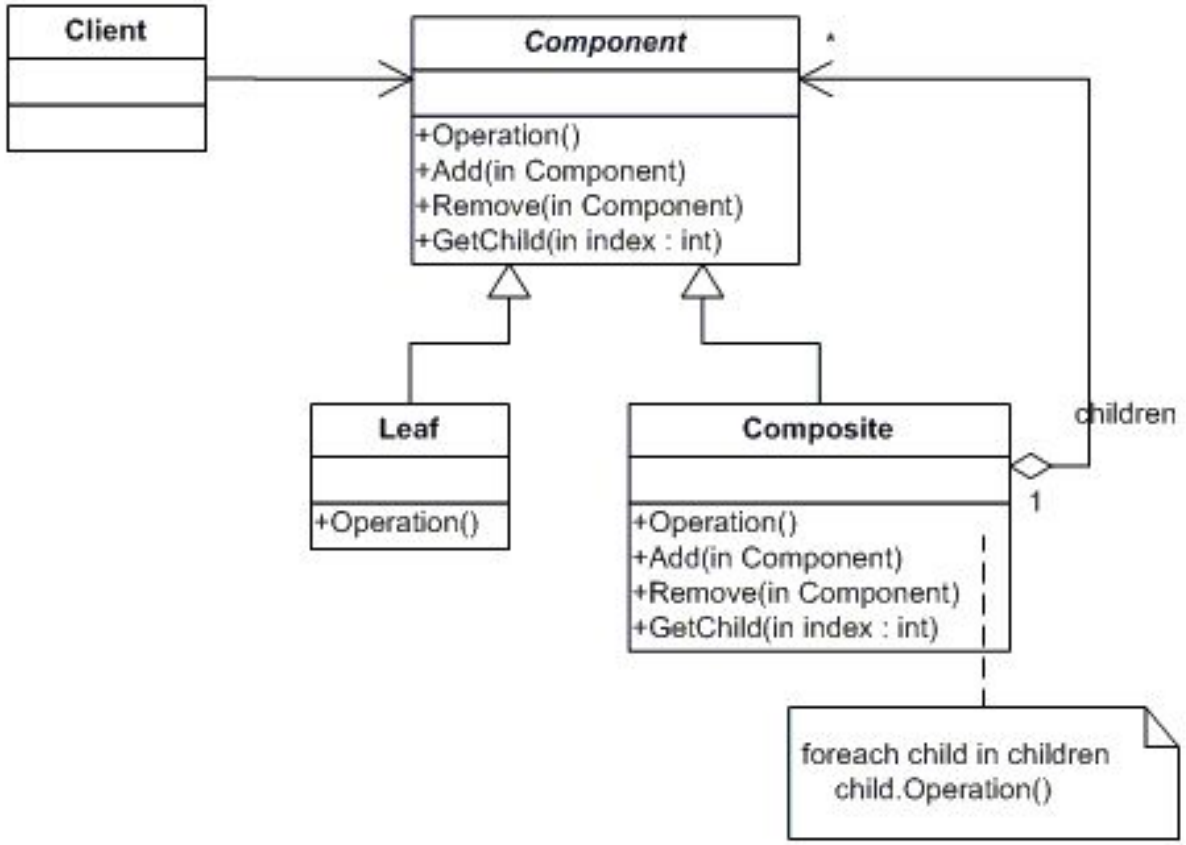


Diagrama y ejemplo en https://www.dofactory.com/net/composite-design-pattern

Patrones estructurales: Decorator

Agrega responsabilidad adicional a un objeto de manera dinámica. Decoradores permiten una alternativa flexible para extender funcionalidad de subclases.

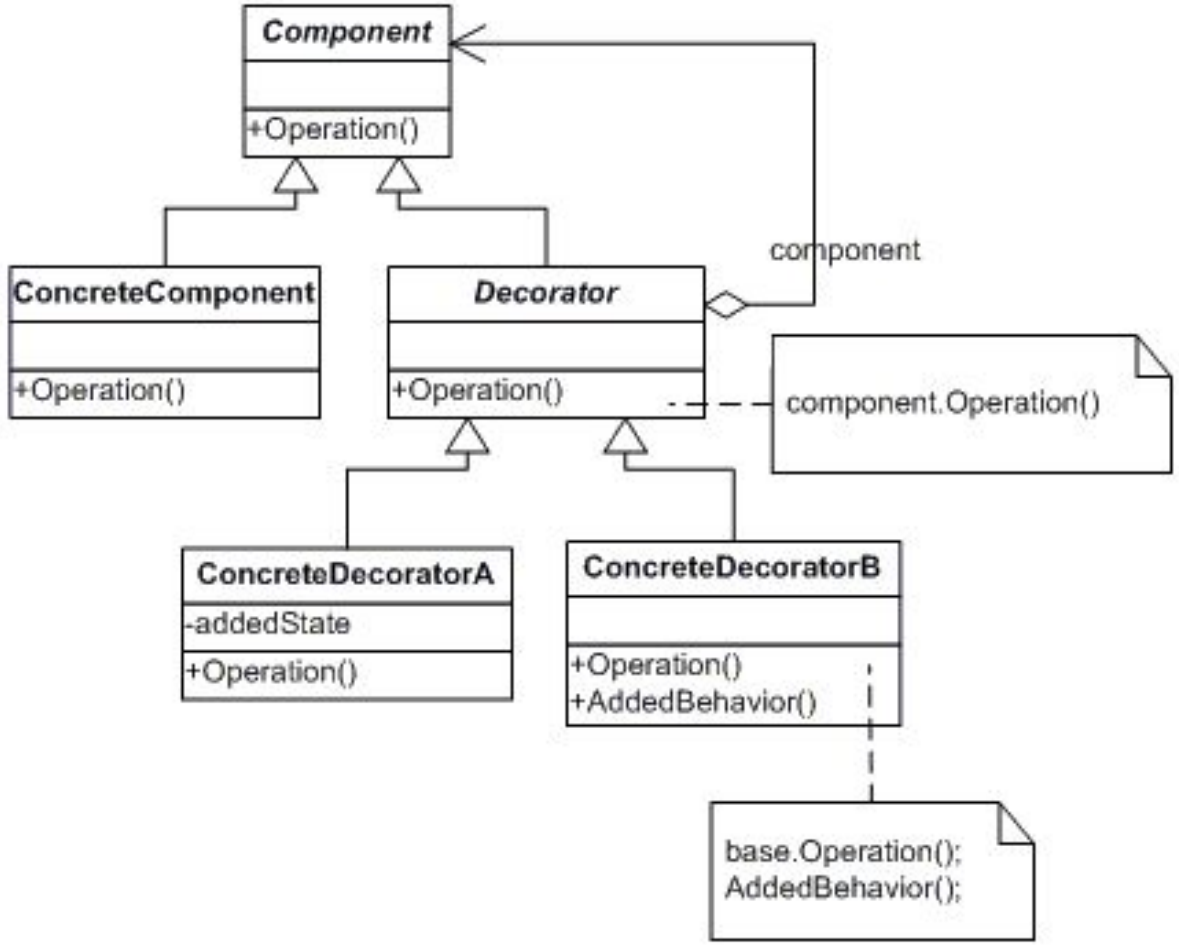


Diagrama y ejemplo en https://www.dofactory.com/net/decorator-design-pattern

Patrones estructurales: Decorator

¿Cuándo se utiliza?

- Para agregar responsabilidad a objetos individuales de manera dinámica y transparente (sin afectar otros objetos).
- Para responsabilidades que pueden ser retiradas.
- Cuando extender con sub-clases es impracticable. Puede ser por un gran número de extensiones independientes, o que no se pueden desprender sub-clases a partir de una clase.

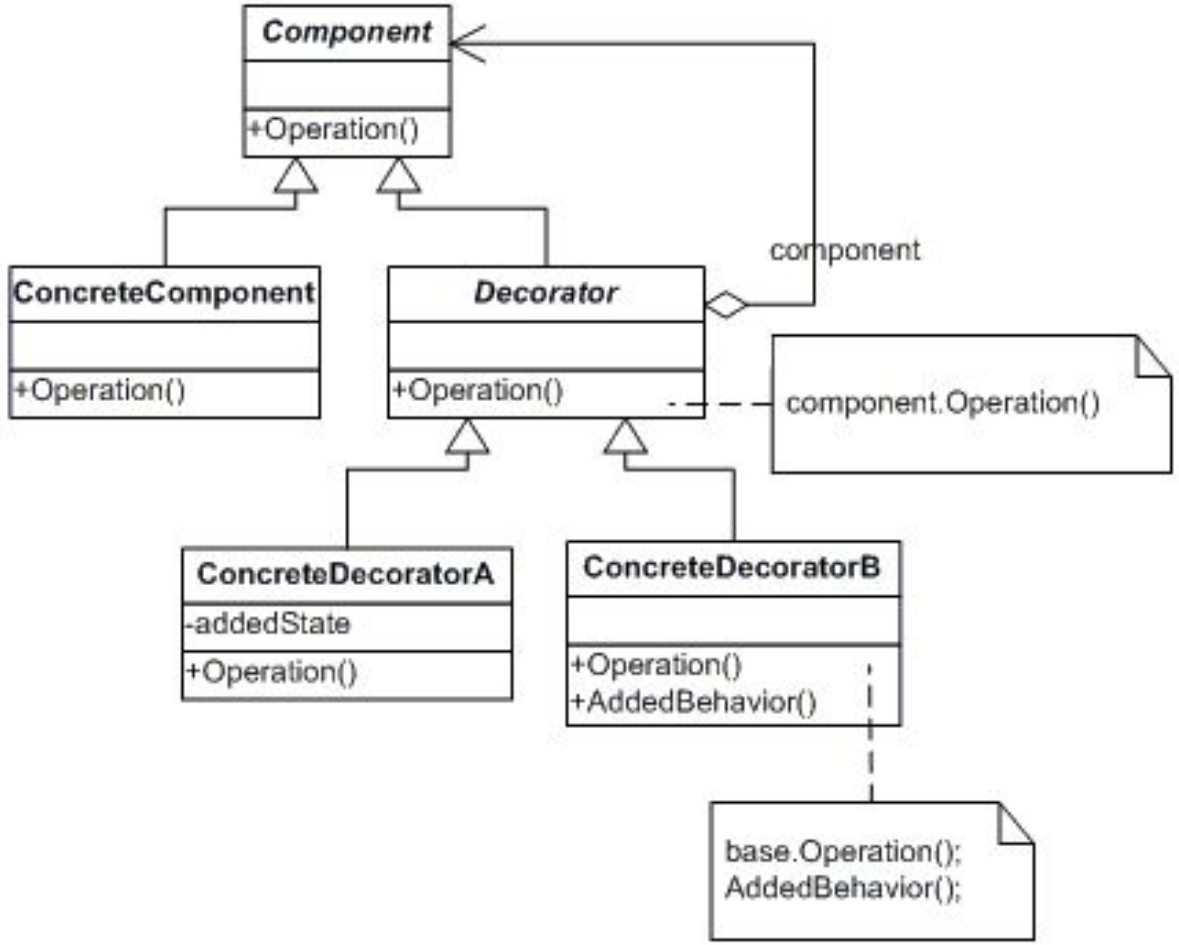


Diagrama y ejemplo en https://www.dofactory.com/net/decorator-design-pattern

Patrones estructurales: Facade

Provee una interfaz unificada a un conjunto de interfaces en un sub-sistema. Facade define una interfaz de alto nivel que hace a un sistema más fácil de utilizar.

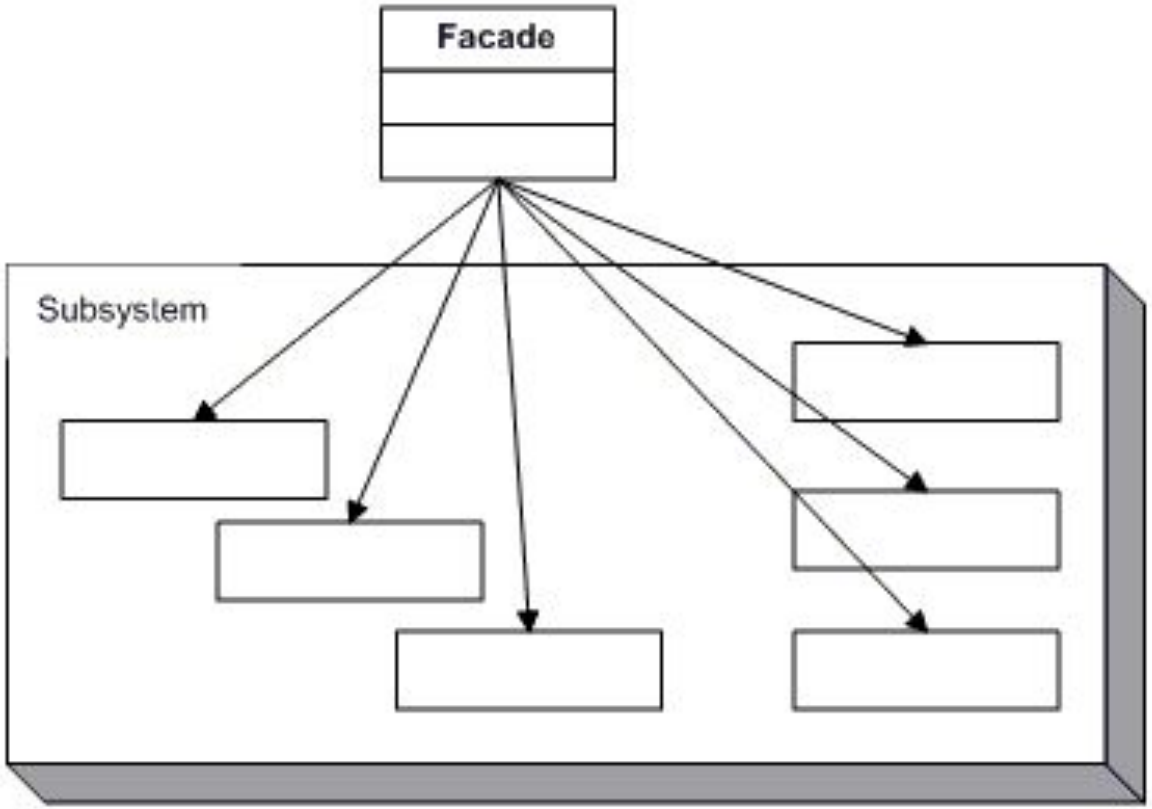


Diagrama y ejemplo en https://www.dofactory.com/net/facade-design-pattern

Patrones estructurales: Facade

¿Cuándo se utiliza?

- Se quiere proveer una interfaz simple para un sistema complejo (altamente configurable).
- Desacoplar los clientes de los sub-sistemas.
- Ordenar los sub-sistemas por capas, con una facade como punto de entrada para cada capa.

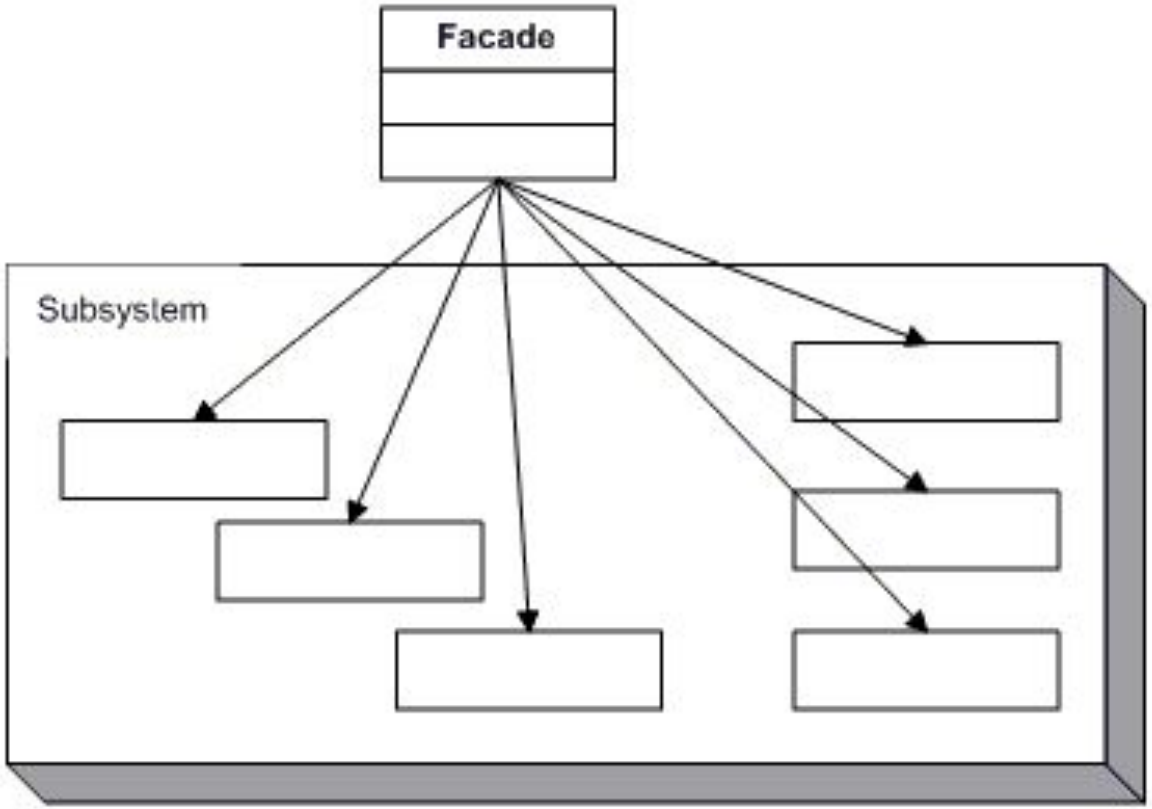


Diagrama y ejemplo en https://www.dofactory.com/net/facade-design-pattern

Patrones estructurales: Flyweight

Utiliza valores por referencia para soportar grandes números de objetos.

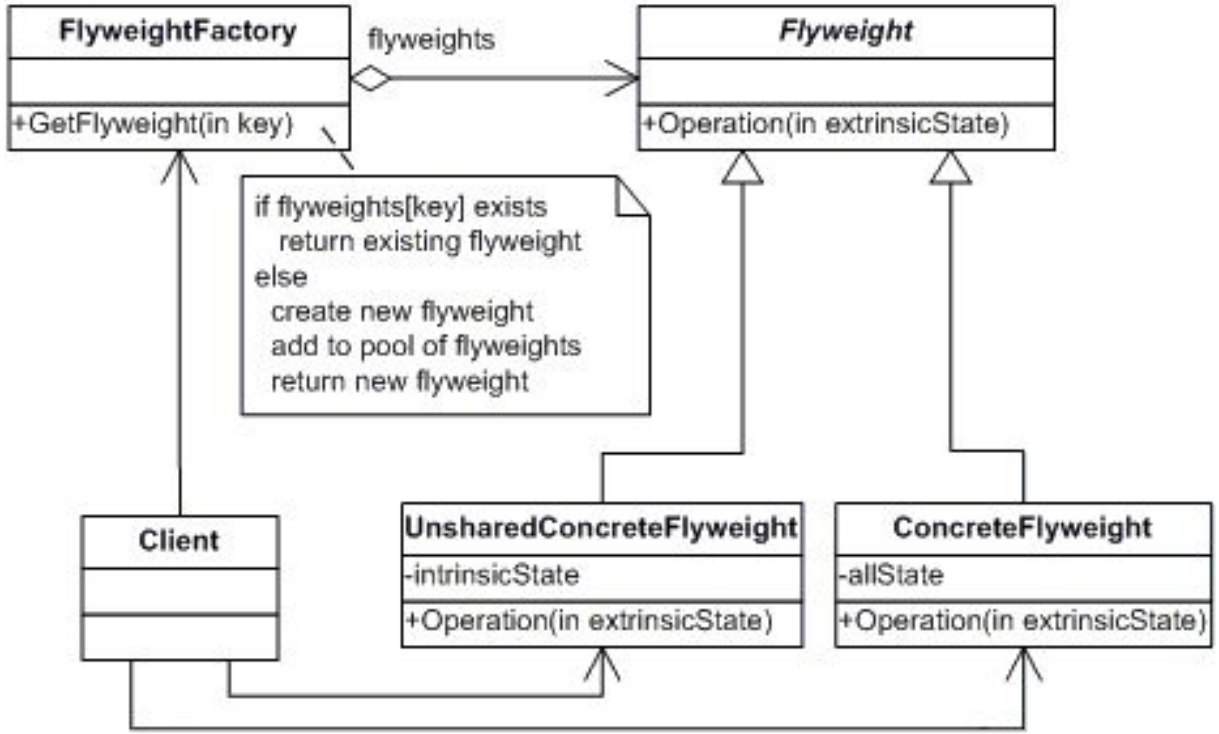


Diagrama y ejemplo en https://www.dofactory.com/net/flyweight-design-pattern

Patrones estructurales: Flyweight

¿Cuándo se utiliza?

- Una aplicación utiliza un gran número de objetos.
- El costo de almacenamiento es caro dada la enorme cantidad de objetos.
- La aplicación no depende de la identidad exacta del objeto. La diferenciación de objetos externa.

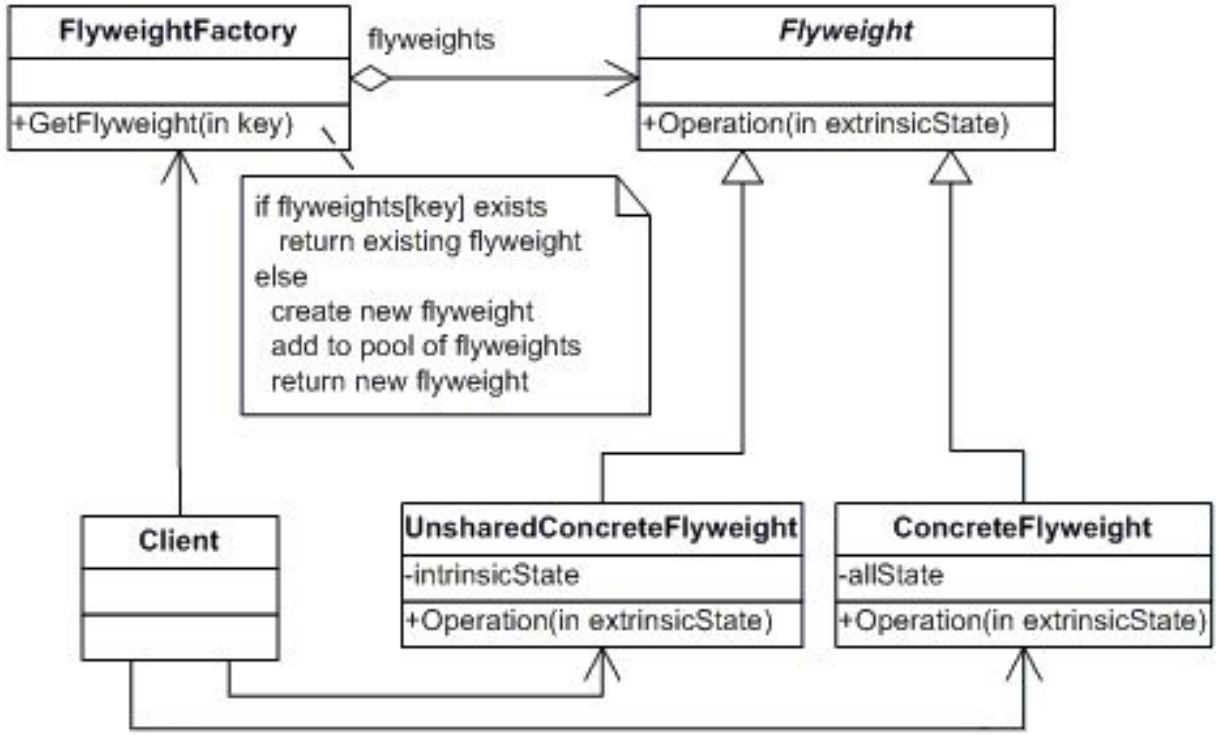


Diagrama y ejemplo en https://www.dofactory.com/net/flyweight-design-pattern

Patrones estructurales: Proxy

Provee un sustituto para que ese objeto sustituto controle el acceso al original.

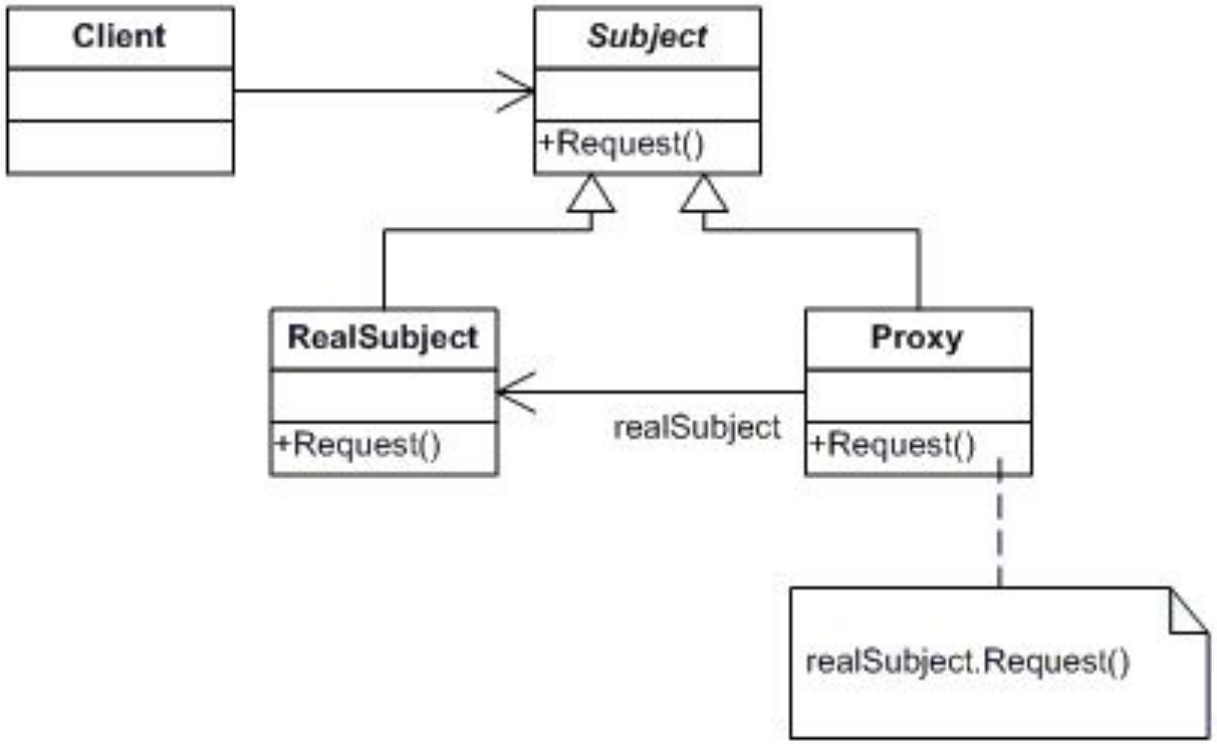
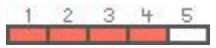


Diagrama y ejemplo en https://www.dofactory.com/net/proxy-design-pattern

Patrones estructurales: Proxy



¿Cuándo se utiliza?

- Se requiere una referencia más sofisticada que un puntero.

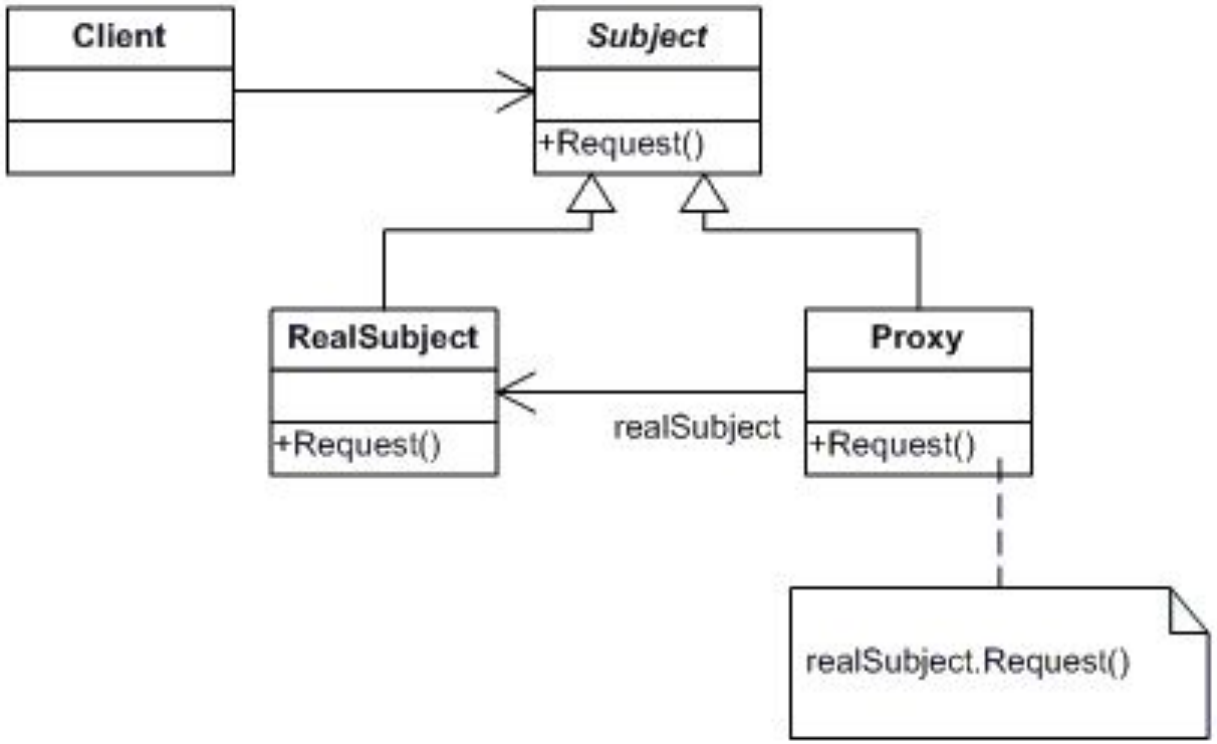


Diagrama y ejemplo en https://www.dofactory.com/net/proxy-design-pattern

Tipos de patrones de diseño

		Proposito		
		Creacional	**Estructural**	**De comportamiento**
Scope	**Clase**	- Factory Method	- Adapter	- Interpreter - Template Method
	Objeto	- Abstract Factory - Builder - Prototype - Singleton	- Adapter - Bridge - Composite - Decorator - Facade - Flyweight - Proxy	- Chain of Responsibility - Command - Iterator - Mediator - Memento - Observer - State - Strategy - Visitor

Patrones de comportamiento: Interpreter

Dado un lenguaje, define una representación de su gramática junto con un intérprete que utiliza la representación para interpretar frases en el idioma.

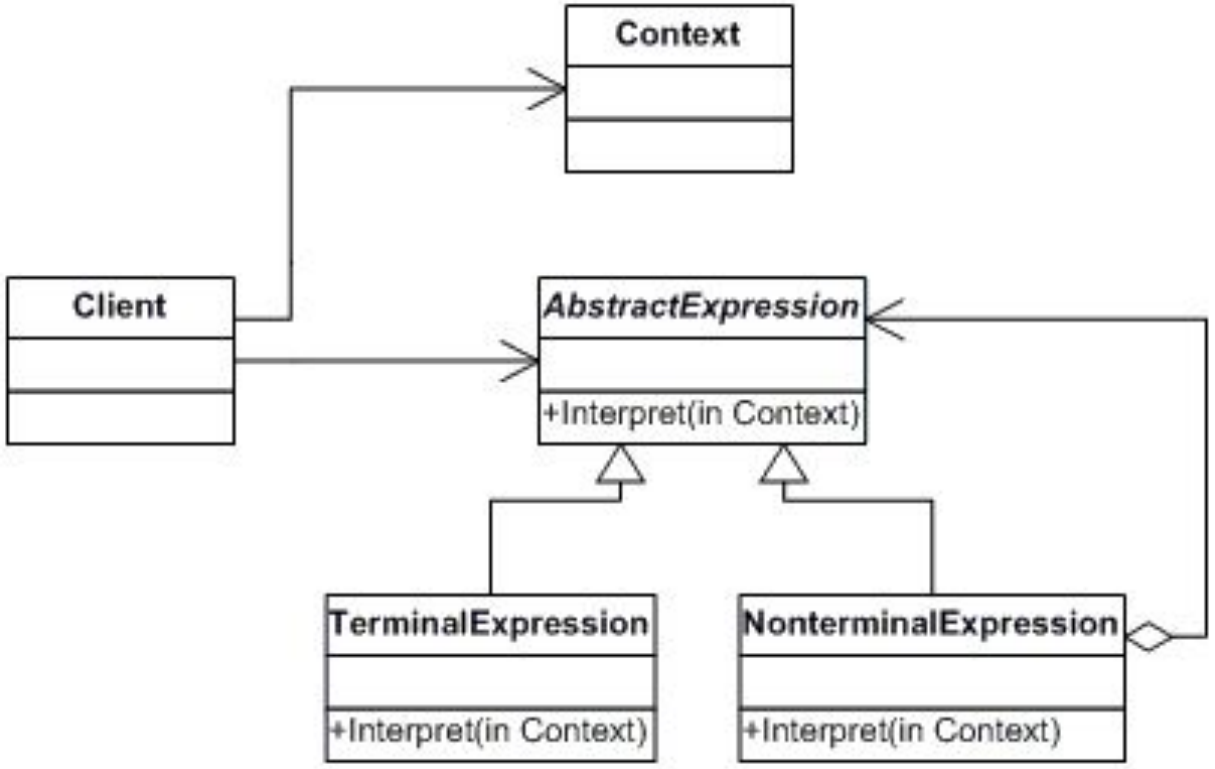


Diagrama y ejemplo en https://www.dofactory.com/net/interpreter-design-pattern

Patrones de comportamiento: Interpreter

¿Cuándo se utiliza?

- La gramática del lenguaje es simple.
- La eficiencia no es un requisito principal.

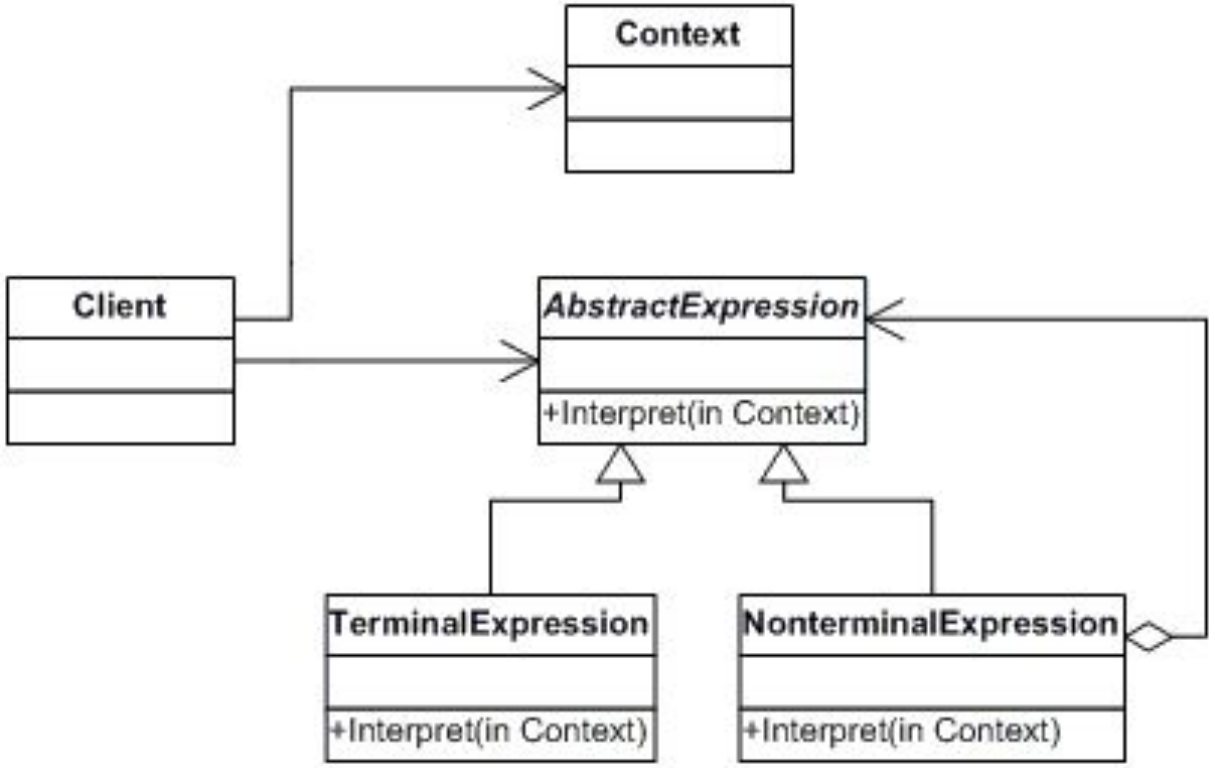


Diagrama y ejemplo en https://www.dofactory.com/net/interpreter-design-pattern

Patrones de comportamiento: Template Method

Define la estructura de un algoritmo, delegando algunos pasos a subclases. Template Method permite que las subclases redefinen algunos pasos del algoritmo, sin cambiar la estructura de este.

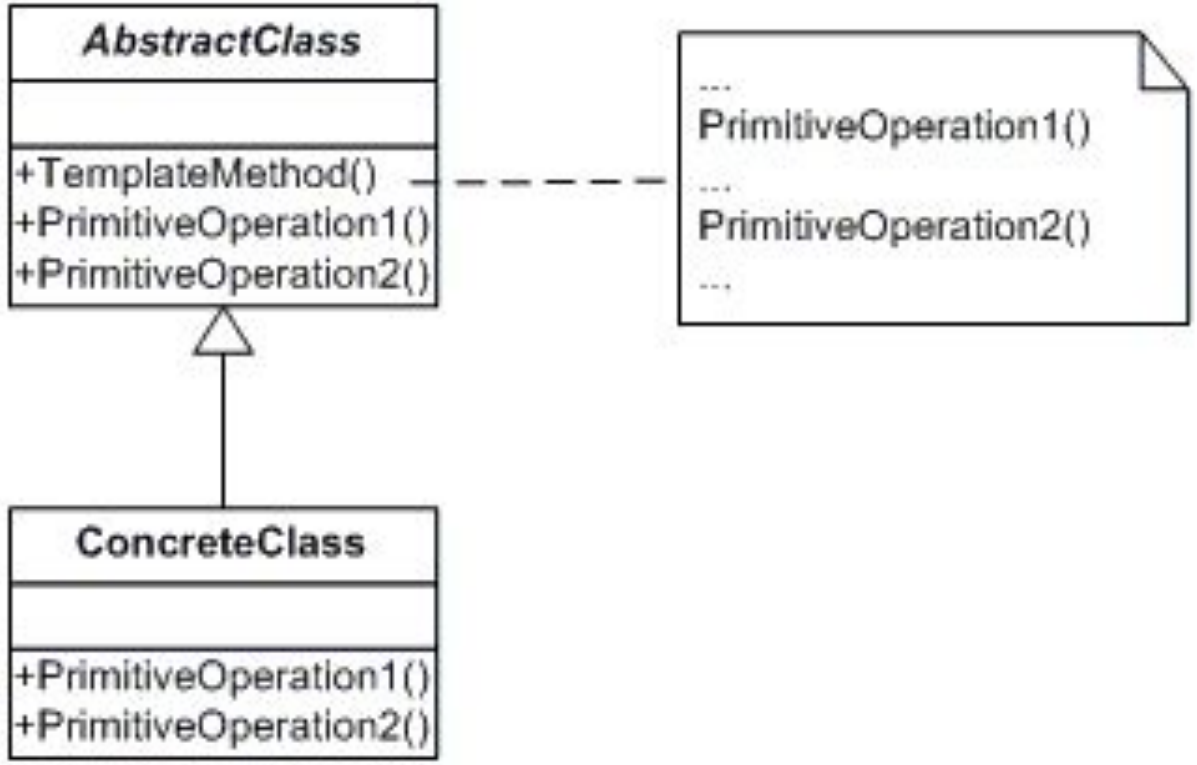
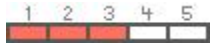


Diagrama y ejemplo en https://www.dofactory.com/net/template-method-design-pattern

Patrones de comportamiento: Template Method



¿Cuándo se utiliza?

- Para implementar una sola vez la parte invariante de un algoritmo, y permitir que algunos pasos cambien
- Para agrupar compartimiento común, y así evitar código duplicado

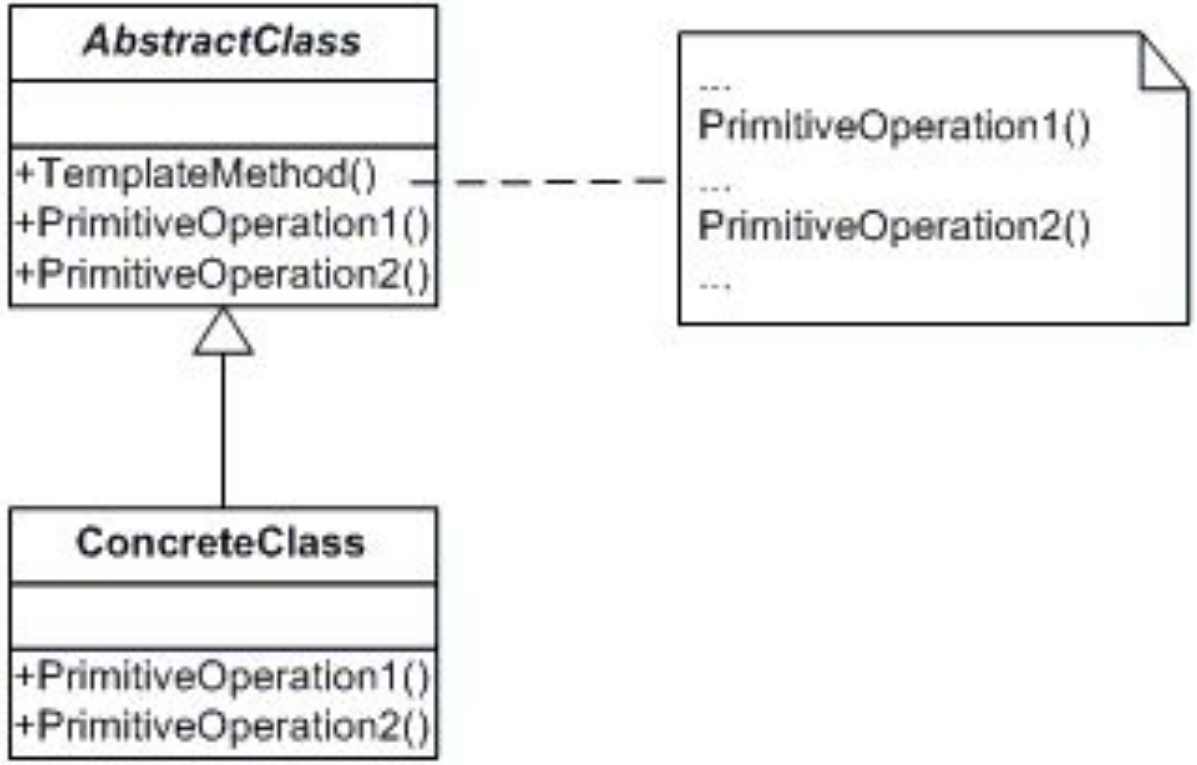


Diagrama y ejemplo en https://www.dofactory.com/net/template-method-design-pattern

Tipos de patrones de diseño

		Proposito		
		Creacional	**Estructural**	**De comportamiento**
Scope	**Clase**	- Factory Method	- Adapter	- Interpreter - Template Method
	Objeto	- Abstract Factory - Builder - Prototype - Singleton	- Adapter - Bridge - Composite - Decorator - Facade - Flyweight - Proxy	- Chain of Responsibility - Command - Iterator - Mediator - Memento - Observer - State - Strategy - Visitor

Patrones de comportamiento: Chain of Responsibility

Evita el acoplamiento entre el objeto que envía una solicitud y el objeto que la recibe, al dar la oportunidad de que más de un objeto procese la solicitud. Encadena los objetos solicitados y delega la solicitud a lo largo de la cadena hasta que un objeto se encargue de ella.

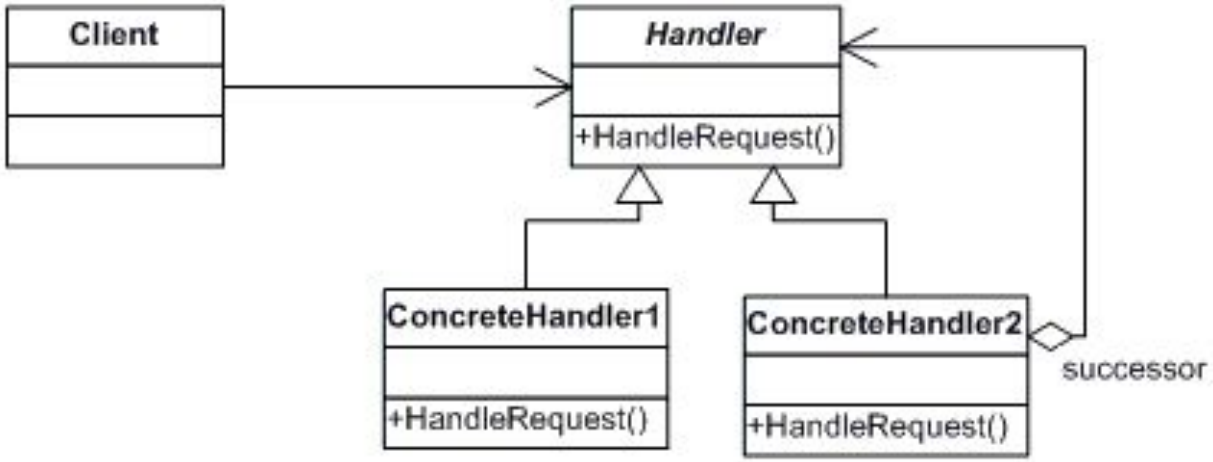


Diagrama y ejemplo en https://www.dofactory.com/net/chain-of-responsibility-design-pattern

Patrones de comportamiento: Chain of Responsibility

¿Cuándo se utiliza?

- Más de un objeto puede encargarse de una solicitud, y el que se hará cargo es desconocido.
- Se quiere generar una solicitud a uno de varios objetos sin especificar el receptor.
- El conjunto de objetos que puede hacerse cargo de la solicitud se define dinámicamente.

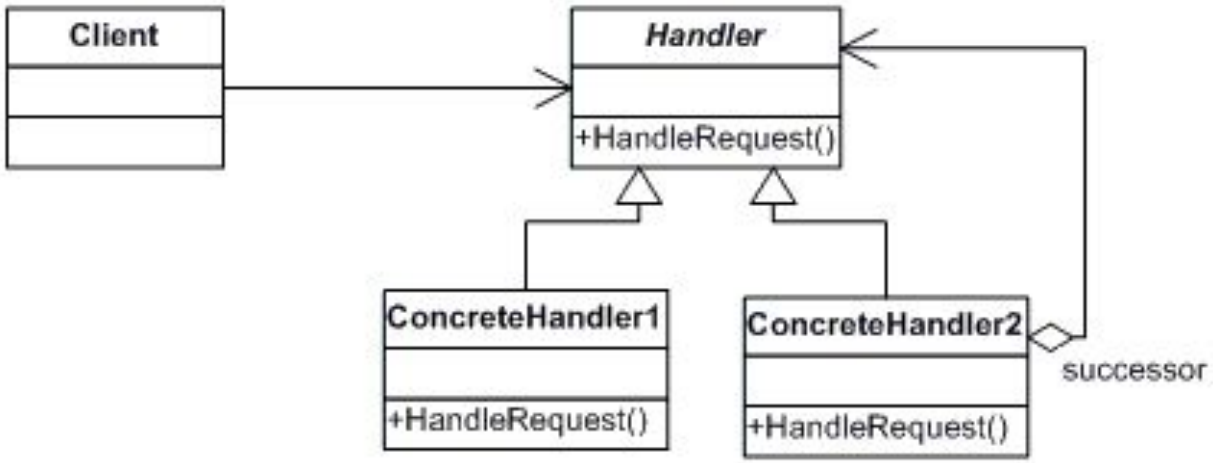


Diagrama y ejemplo en https://www.dofactory.com/net/chain-of-responsibility-design-pattern

Patrones de comportamiento: Command

Encapsula una solicitud como un objeto, para así permitir la parametrización de clientes con solicitudes distintas, encolar o registrar solicitudes y soportar operaciones que se pueden deshacer.

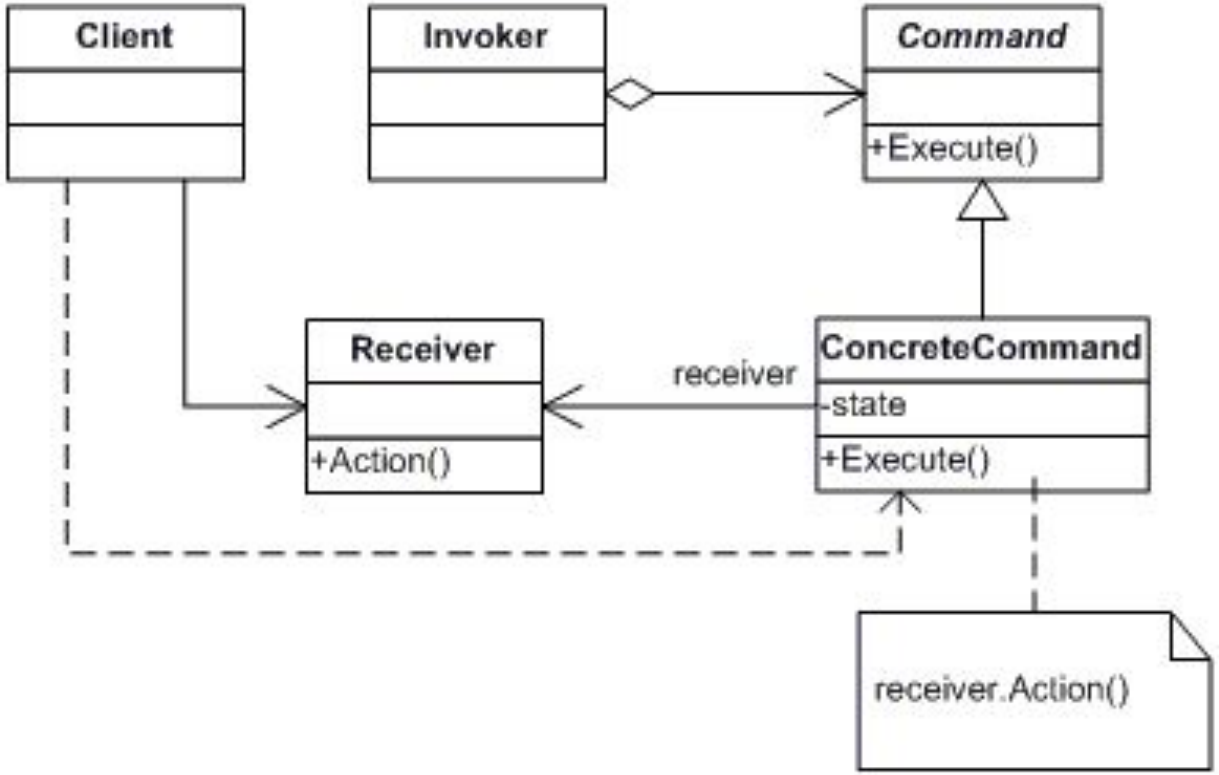


Diagrama y ejemplo en https://www.dofactory.com/net/command-design-pattern

Patrones de comportamiento: Command

¿Cuándo se utiliza?

- Se necesita parametrizar objetos con una acción a realizar. Command es un reemplazo para callbacks en OOP.
- Especificar, encolar y ejecutar solicitudes en momentos diferentes. Soportar que la solicitud se deshaga.
- Estructurar un sistema alrededor de operaciones complejas, formadas a partir de operaciones primitivas.

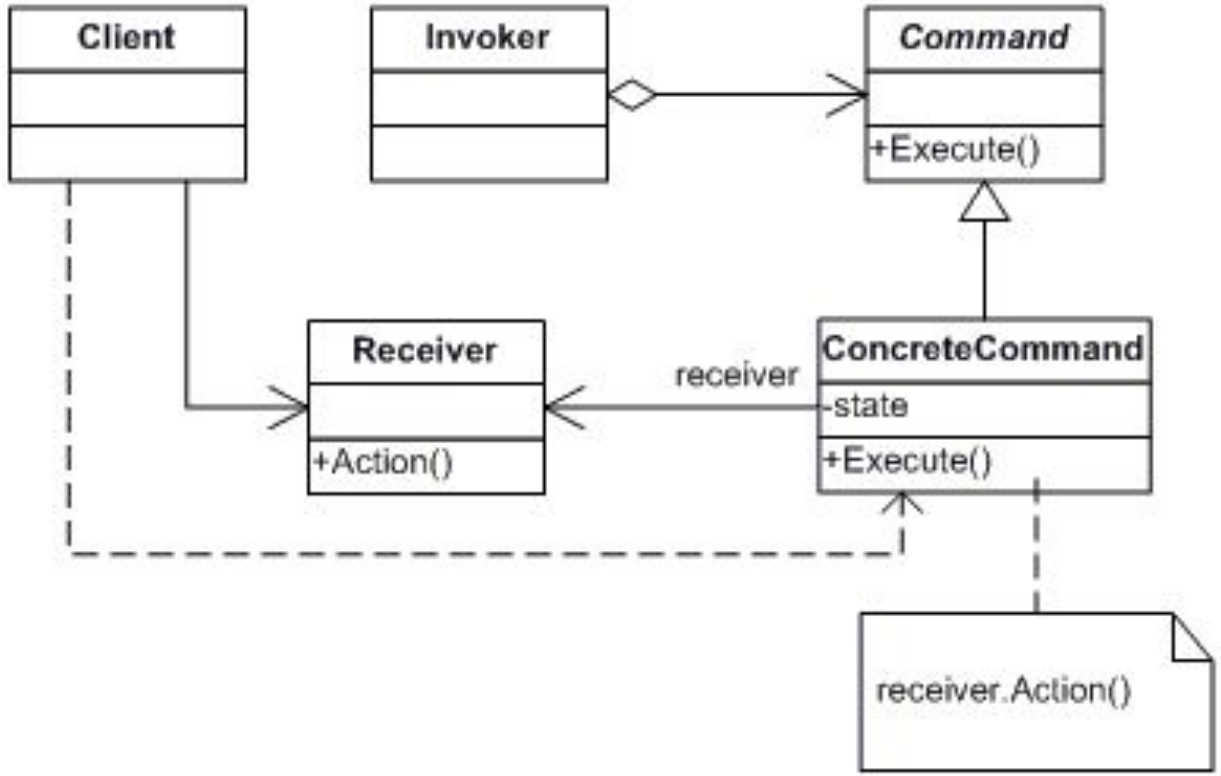


Diagrama y ejemplo en https://www.dofactory.com/net/command-design-pattern

Patrones de comportamiento: Iterator

Provee una forma de acceder a los elementos de una agregación de objetos secuenciales sin exponer su representación subyacente.

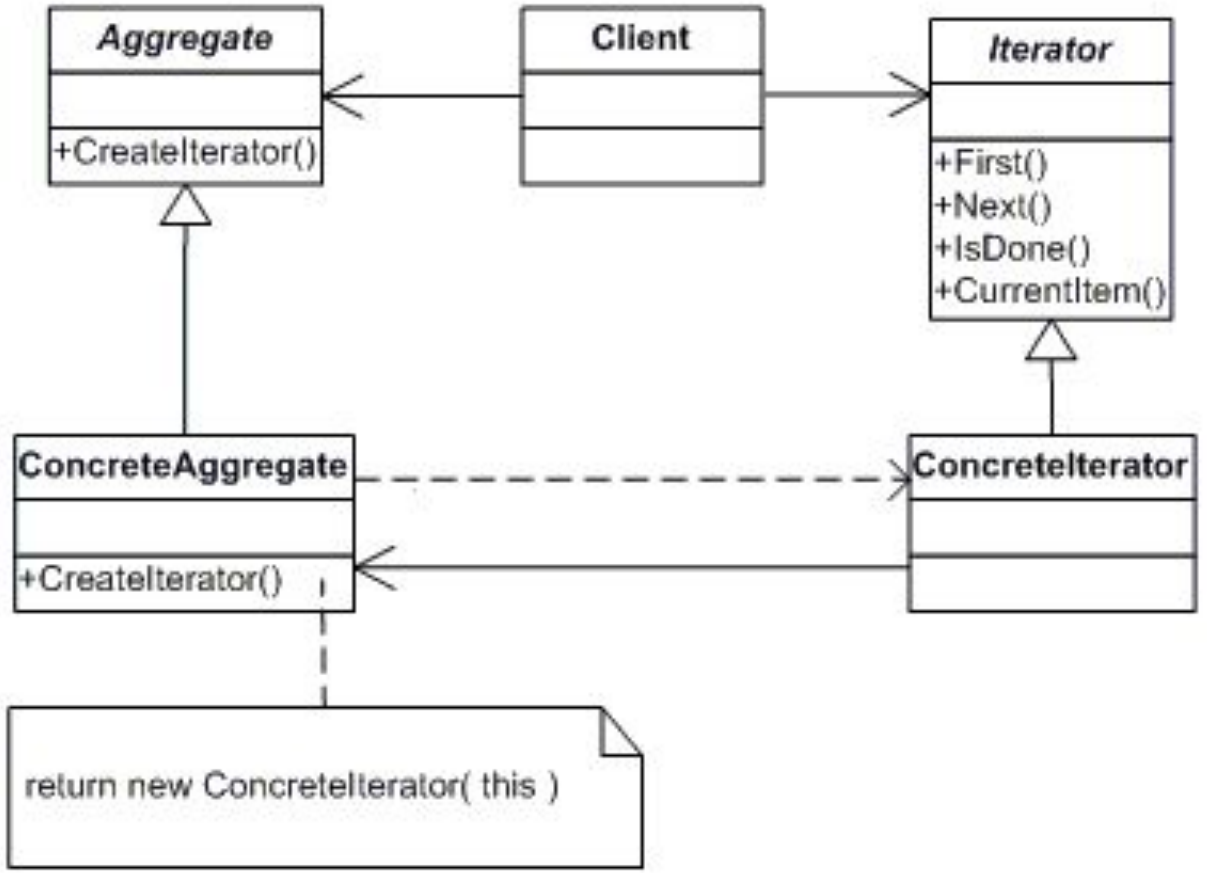


Diagrama y ejemplo en https://www.dofactory.com/net/iterator-design-pattern

Patrones de comportamiento: Iterator

¿Cuándo se utiliza?

- Para acceder al contenido de un objeto agregado sin exponer su representación interna.
- Para soportar múltiples recorridos de objetos agregados.
- Para proveer una interfaz uniforme para recorrer diferentes estructuras agregadas.

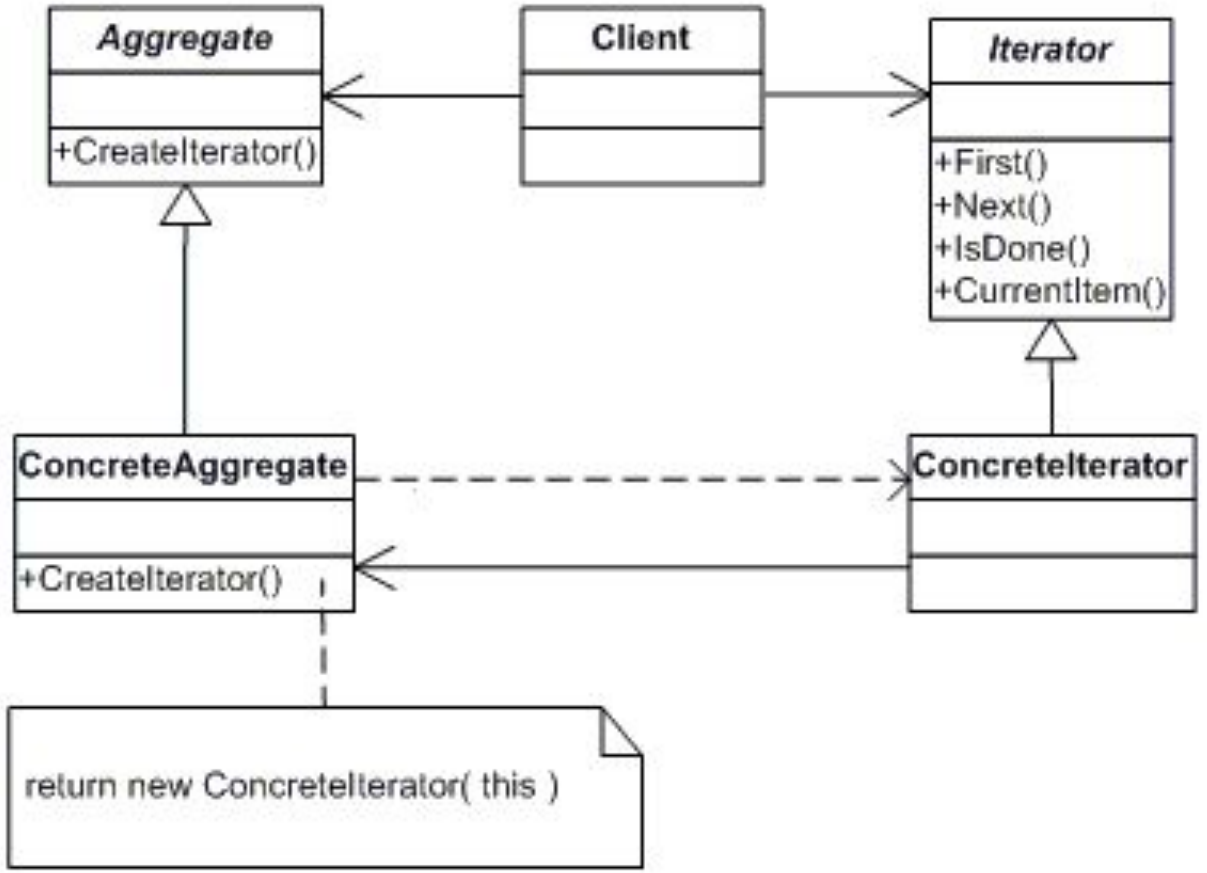
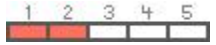


Diagrama y ejemplo en https://www.dofactory.com/net/iterator-design-pattern

Patrones de comportamiento: Mediator



Define un objeto que encapsula como un conjunto de objetos interactúan. Este patrón promueve el bajo acoplamiento al evitar que los objetos se referencian entre ellos explícitamente, y permite variar sus interacciones independientemente.

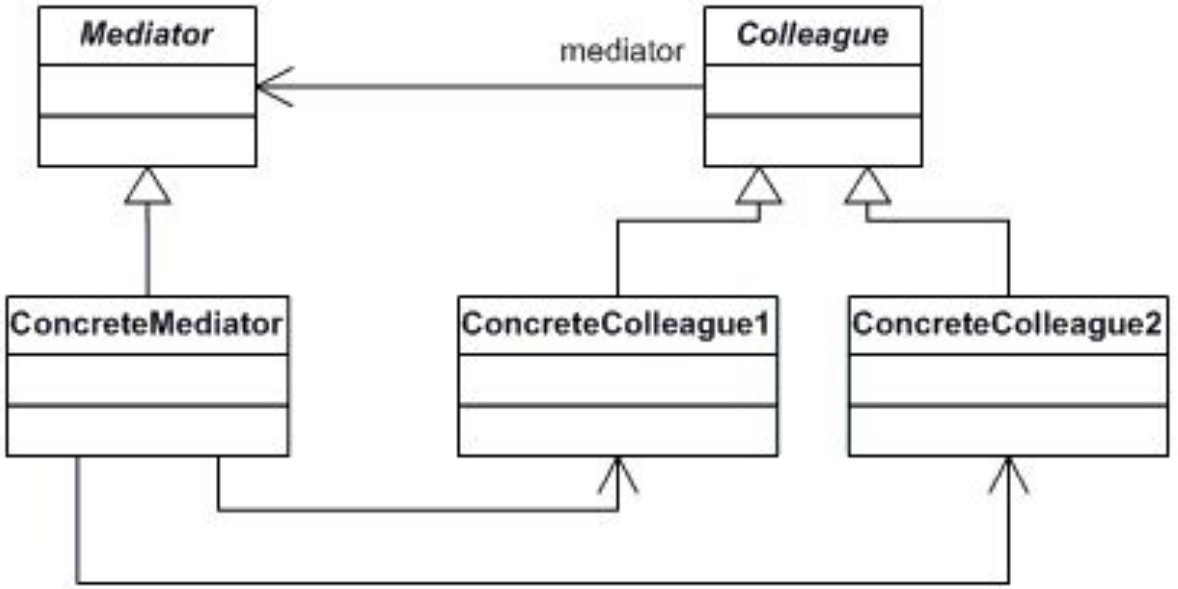
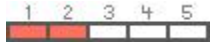


Diagrama y ejemplo en https://www.dofactory.com/net/mediator-design-pattern

Patrones de comportamiento: Mediator



¿Cuándo se utiliza?

- Un conjunto de objetos se comunican de una forma compleja, pero bien definida.
- Un comportamiento distribuido en múltiples clases debería ser configurable sin generar muchas subclases.

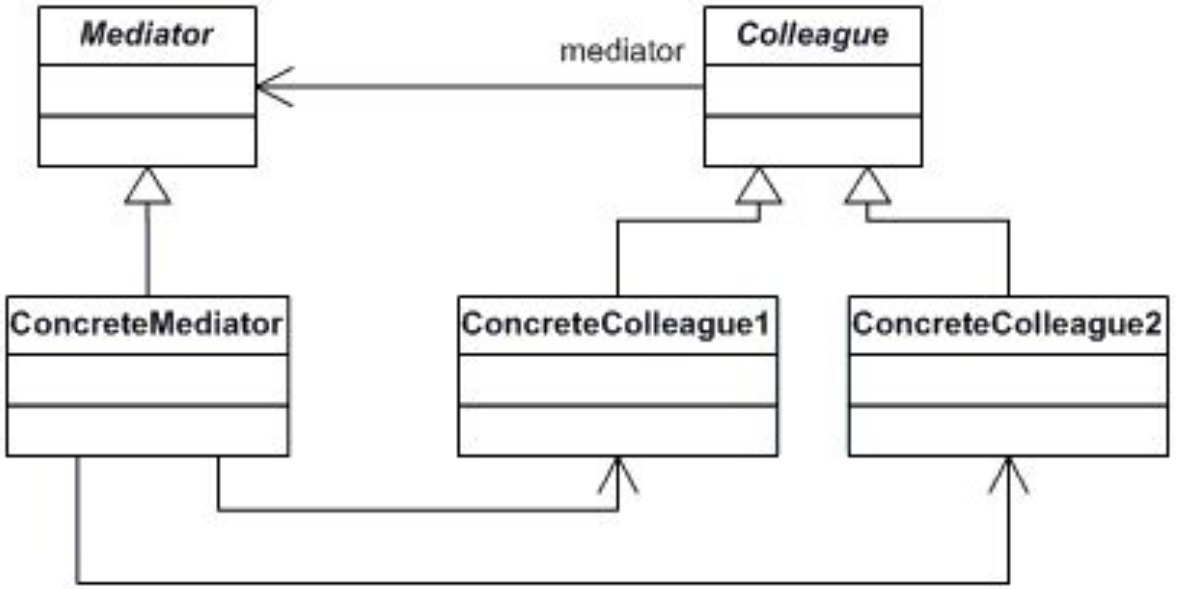


Diagrama y ejemplo en https://www.dofactory.com/net/mediator-design-pattern

Patrones de comportamiento: Memento

Sin incumplir el principio de ocultamiento, captura y expone el estado interno de un objeto para que este pueda volver a ese estado en otro momento.

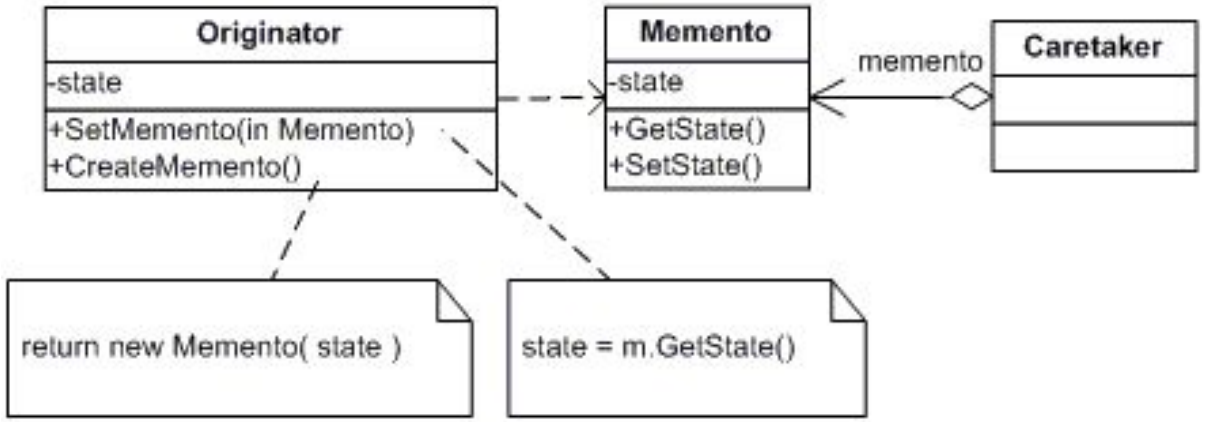


Diagrama y ejemplo en https://www.dofactory.com/net/memento-design-pattern

Patrones de comportamiento: Memento

¿Cuándo se utiliza?

- Una “fotografía” de una parte del estado de un objeto debe ser grabada para poder restaurarla.
- Una interfaz directa para obtener el estado de un objeto expondría detalles de implementación, violando el principio de ocultamiento.

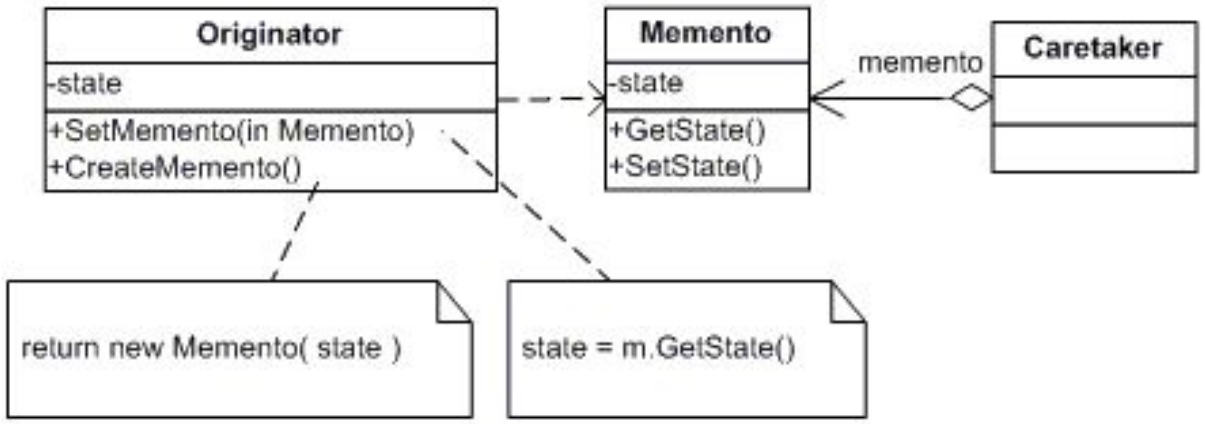


Diagrama y ejemplo en https://www.dofactory.com/net/memento-design-pattern

Patrones de comportamiento: Observer

Define una relación de dependencia entre uno y N objetos, para que cuando ese objeto cambie su estado, todos los N sean notificados y se actualicen automáticamente.

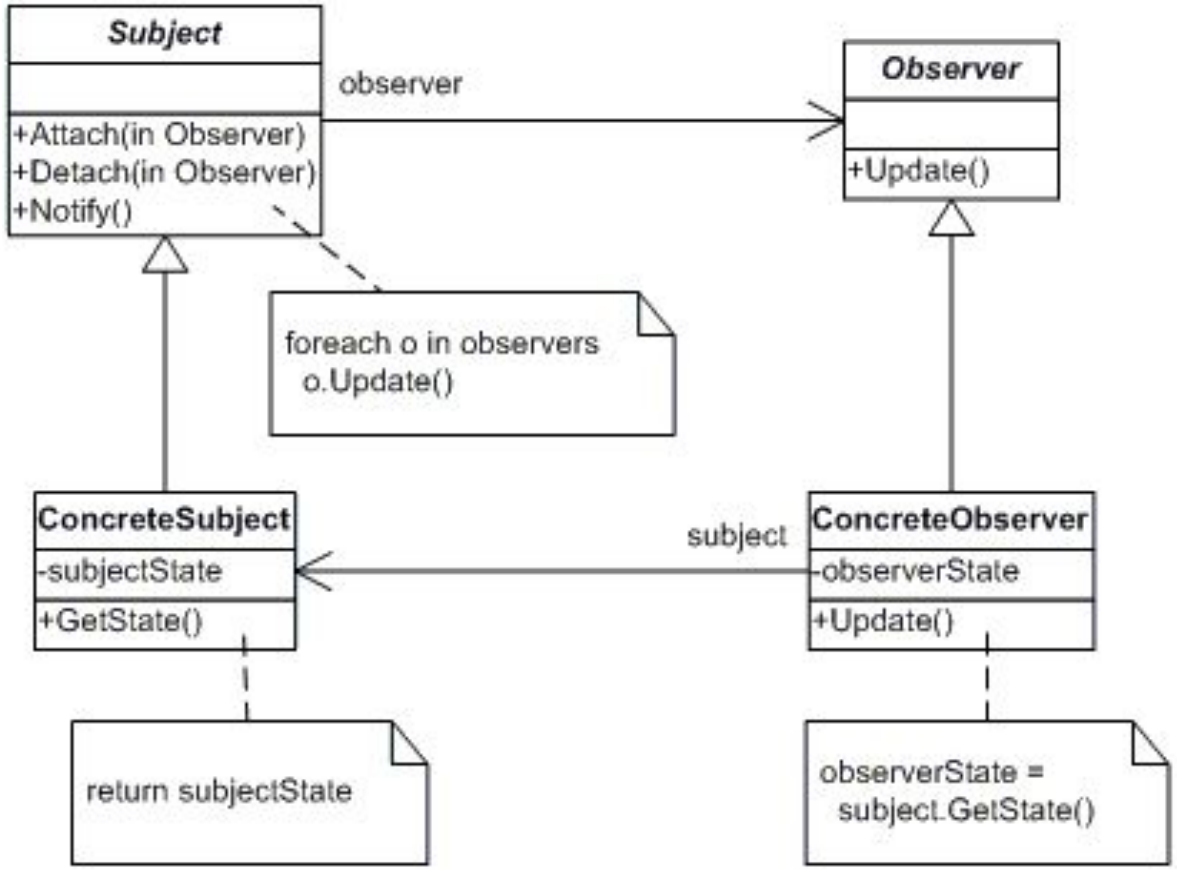


Diagrama y ejemplo en https://www.dofactory.com/net/observer-design-pattern

Patrones de comportamiento: Observer

¿Cuándo se utiliza?

- Cuando una abstracción tiene dos aspectos, uno dependiente del otro. Encapsular estos aspectos en objetos separados permite variarlos y reutilizarlos independientemente.
- Cuando un cambio en un objeto genera cambios en otros, y no se sabe cu´antos objetos deben cambiar.
- Cuando un objeto debería ser capaz de notificar a otros objetos sin realizar supuestos de quiénes son estos objetos. En otras palabras, se quiere reducir el acoplamiento entre objetos dependientes.

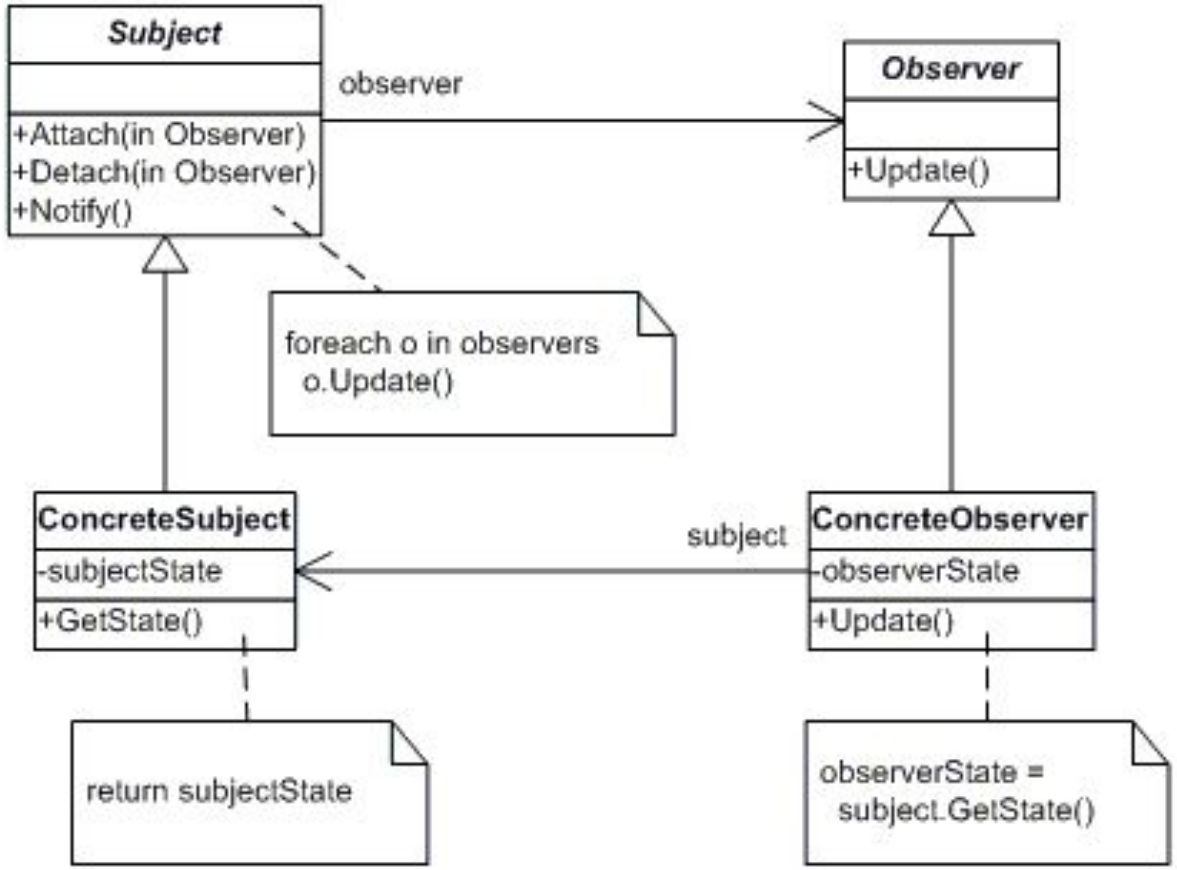


Diagrama y ejemplo en https://www.dofactory.com/net/observer-design-pattern

Patrones de comportamiento: State

Permite a un objeto cambiar su comportamiento cuando su estado interno cambia. Parecería que el objeto cambió de clase.

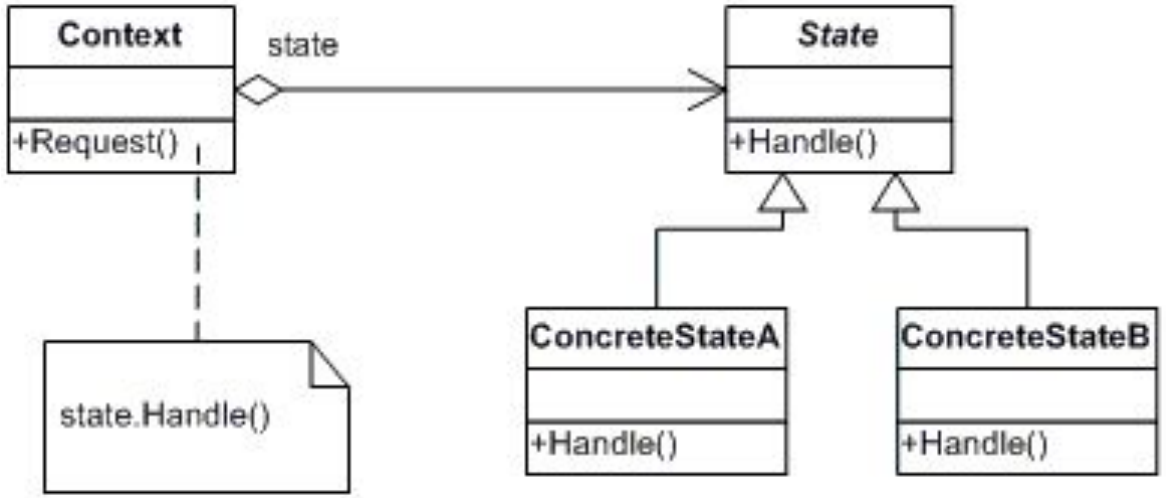


Diagrama y ejemplo en https://www.dofactory.com/net/state-design-pattern

Patrones de comportamiento: State

¿Cuándo se utiliza?

- El comportamiento de un objeto depende de su estado, y debe cambiar su comportamiento en tiempo de ejecuci´on dependiendo de su estado.
- Las operaciones tienen gran cantidad de condicionales que dependen del estado del objeto. Estas condiciones son representadas por constantes, y/o se reutilizan en distintas operaciones. State permite encapsular cada una de las alternativas del condicional en una clase separada. Esto permite tratar el estado del objeto como un objeto en si, que puede variar independientemente.

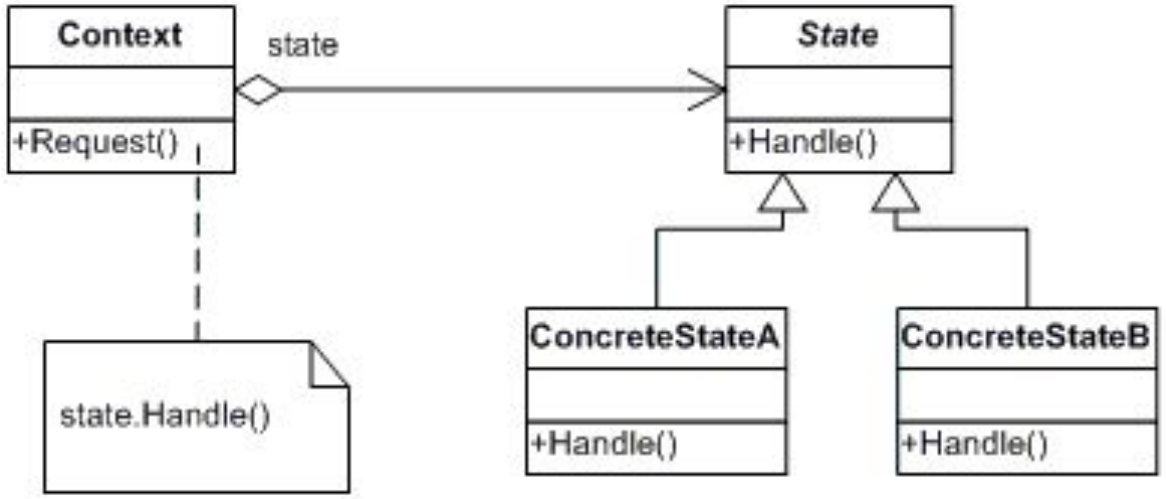


Diagrama y ejemplo en https://www.dofactory.com/net/state-design-pattern

Patrones de comportamiento: Strategy

Define una familia de algoritmos, encapsula cada uno, y los hace intercambiables. Strategy permite al algoritmo cambiar independientemente de los clientes que lo utilizan.

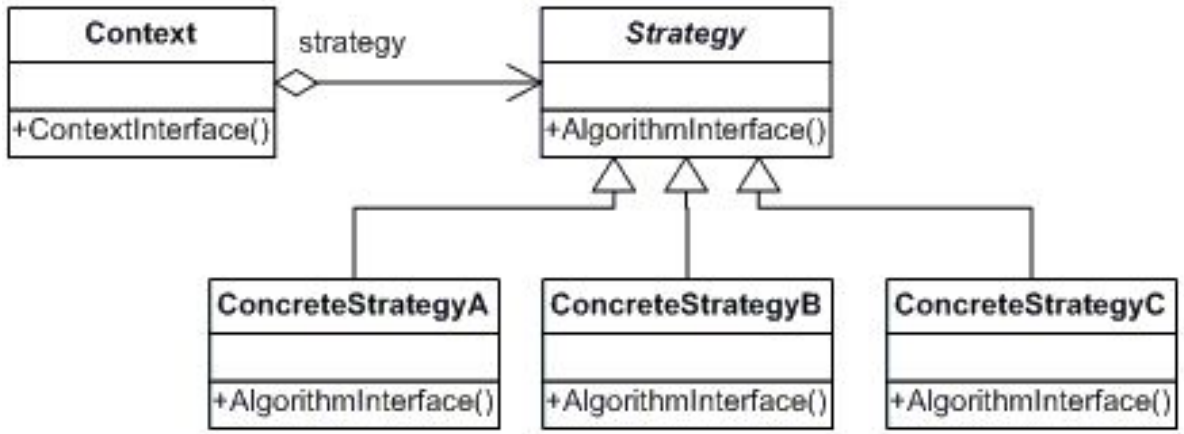


Diagrama y ejemplo en https://www.dofactory.com/net/strategy-design-pattern

Patrones de comportamiento: Strategy

- **¿Cuándo se utiliza?**
- Varias clases relacionadas se diferencian solamente por su comportamiento. Strategy permite cambiar la configuración de una clase a través de la definición de su comportamiento.
- Se necesita diferenciar las variantes de un algoritmo.
- Para ocultar estructuras complejas utilizadas solamente por un algoritmo.
- Una clase define muchos comportamientos, que aparecen como condicionales en sus operaciones. En vez de reutilizar condicionales, se pueden encapsular alternativas bajo una clase Strategy.

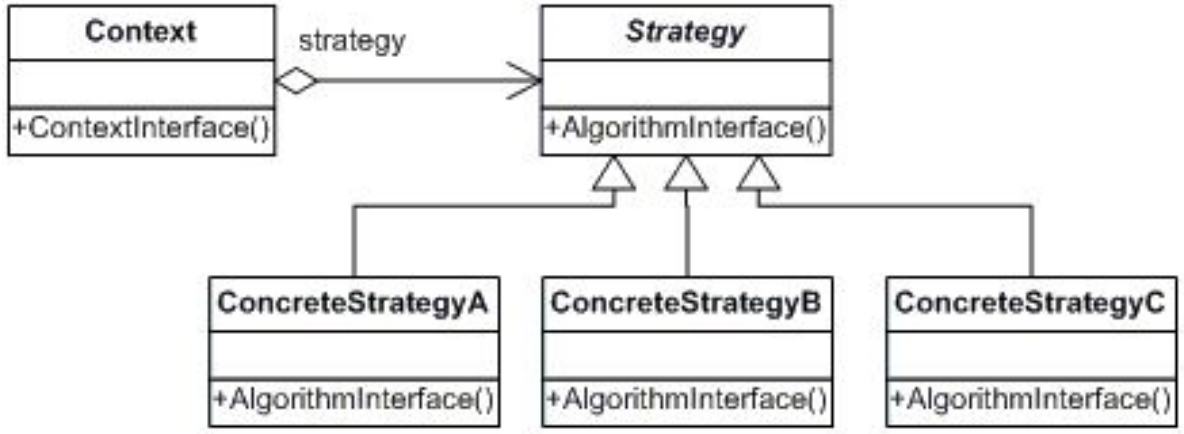


Diagrama y ejemplo en https://www.dofactory.com/net/strategy-design-pattern

Patrones de comportamiento: Visitor

Representa una operación que sea ejecutada en los elementos que componen la estructura de un objeto. Visitor permite definir una nueva operación sin cambiar las clases de los elementos en los cuales se aplica.

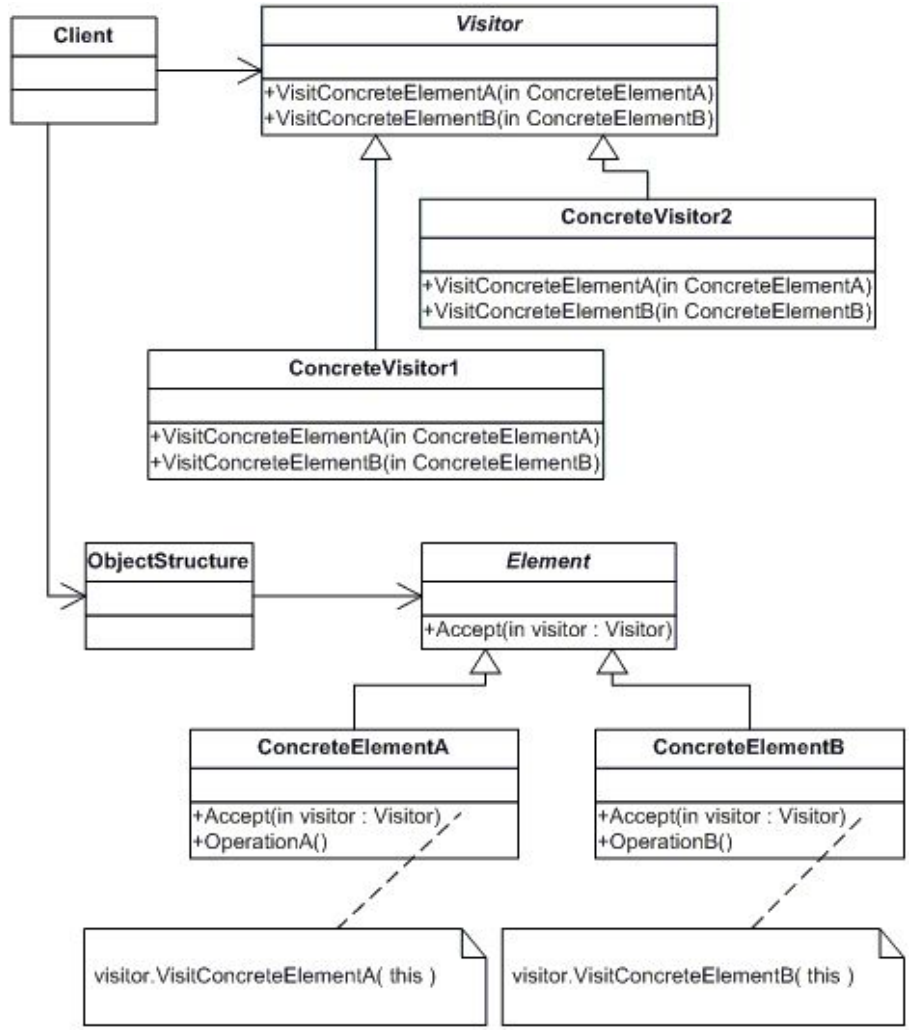


Diagrama y ejemplo en https://www.dofactory.com/net/visitor-design-pattern

Patrones de comportamiento: Visitor

¿Cuándo se utiliza?

- La estructura de un objeto contiene diversas clases de objetos con distintas interfaces, y se desea ejecutar operaciones en esos objetos independientemente de las clases en concreto.
- Muchas operaciones distintas y sin relaci´on se aplican en los objetos de una estructura, y se quiere mantener estas clases simples, sin la declaración de estas operaciones

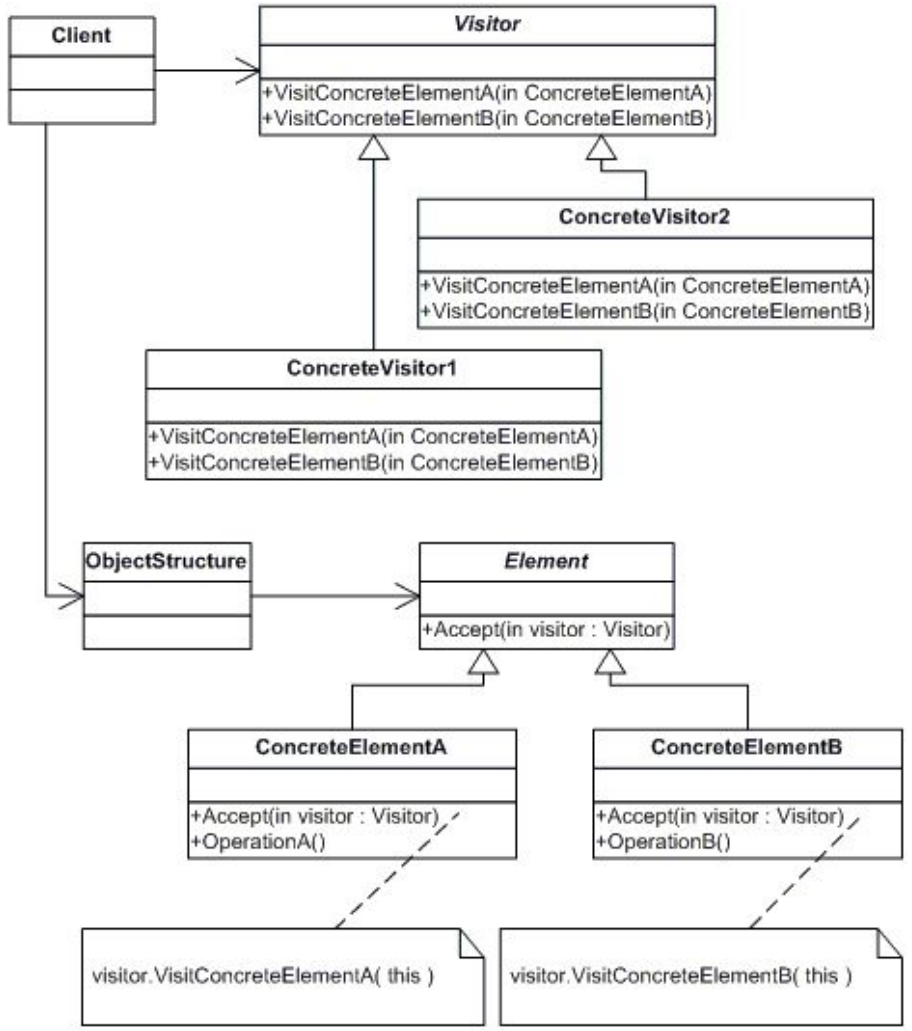


Diagrama y ejemplo en https://www.dofactory.com/net/visitor-design-pattern

03. Próxima Clase

Próxima clase

- Patrones de Arquitectura y CLEAN

Clase 4 - Patrones

Bibliografía

- Pressman, R. S. (2009). Software Engineering: A Practitioner's Approach
- Gamma, E., Helm. R., Johnson R., Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software

IIC2113 - Diseño Detallado de Software

Fernanda Sepúlveda - mfsepulveda@uc.cl

Pontificia Universidad Católica de Chile
2020-2