**DOCUMENTACION PATRONES GoF**

**Integrantes:**

**67000962 - Erick Santiago Sierra Ramírez**

**Rodrigo Aranda Fernández**

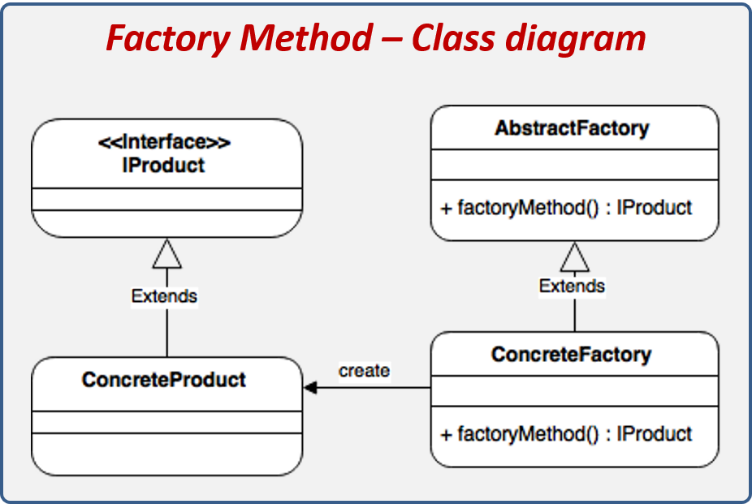
**CONSTRUCCION DE SOFTWARE**

**INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**

**UNIVERSIDAD CATOLICA DE COLOMBIA FACULTAD DE INGENIERÍA BOGOTÁ D.C**

**2024**

**Descripcion y análisis - Patrones GoF**

**1. Factory Method**

**a) Categoría:** Creacional

**b) Problema que resuelve:** Crea objetos sin especificar la clase exacta a instanciar

**c) Implicaciones:**

* Facilita la extensión sin modificar el código existente
* Reduce el acoplamiento entre la clase y los objetos que crea
* Promueve la responsabilidad única.

**d) Desventajas:**

* Aumenta el número de clases
* Puede ser complejo con muchas subclases
* No es adecuado para pocas variantes

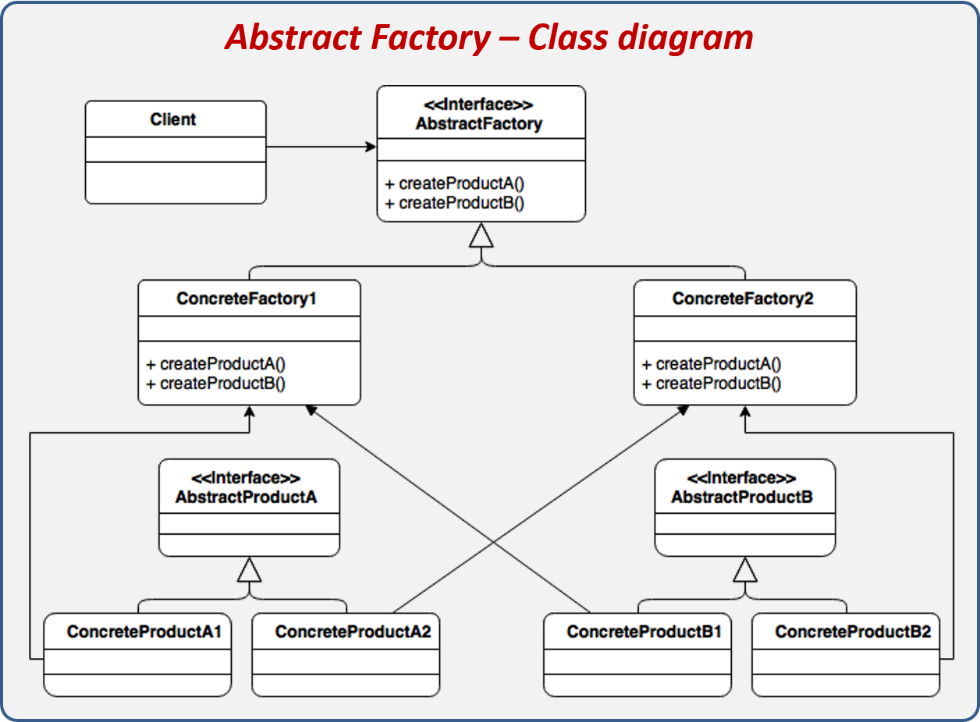
**e) Patrones Relacionados:**

* Abstract Factory
* Prototype

**f) Aspectos donde es utilizado:**

* Creación de botones
* interfaces gráficas

**2. Abstract Factory**



**a) Categoría:** Creacional

**b) Problema que resuelve:** Permite crear familias de objetos relacionados sin especificar sus clases concretas

**c) Implicaciones:**

* Facilita la creación de familias de objetos
* Promueve la consistencia entre productos
* Aísla al cliente de las implementaciones concretas

**d) Desventajas:**

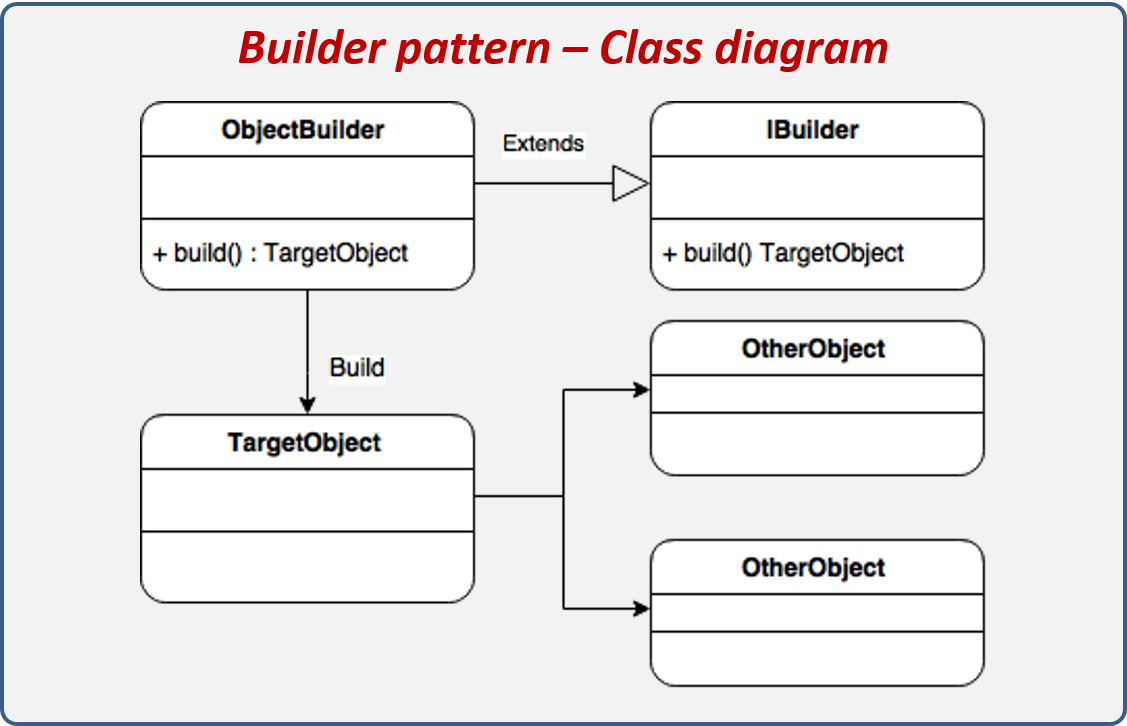
* Aumenta la complejidad del sistema.
* El cambio de familias completas puede ser difícil
* Más clases y código por gestionar

**e) Patrones Relacionales:**

* Factory Method
* Singleton

**f) Aspectos donde es utilizado:**

* Interfaces gráficas multiplataforma

**3. Builder**

**a) Categoría:** Creacional

**b) Problema que resuelve:** Facilita la creación de objetos complejos paso a paso

**c) Implicaciones:**

* Ideal para objetos complejos con muchos parámetros
* Separación de la construcción y representación
* Mejora la legibilidad

**d) Desventajas:**

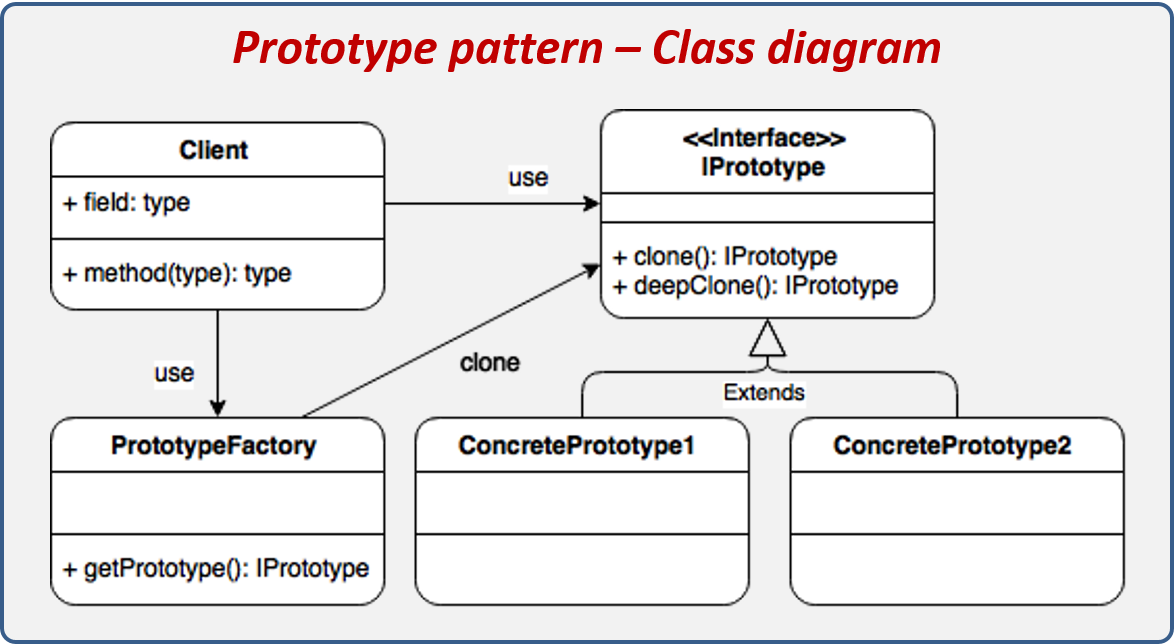
* Puede añadir complejidad innecesaria para objetos simples
* Más código para gestionar
* Los objetos no se crean de inmediato, lo que puede ser confuso

**e) Patrones Relacionales:**

* Composite

**f) Aspectos donde es utilizado:**

* Creación de configuraciones
* Estructuras complejas

**4. Prototype**

**a) Categoría:** Creacional

**b) Problema que resuelve:** Crea objetos clonando otros ya existentes

**c) Implicaciones:**

* Clona objetos sin depender de sus clases concretas
* Reduce la necesidad de crear nuevas instancias desde cero
* Optimiza el rendimiento en sistemas grandes

**d) Desventajas:**

* Difícil de implementar en objetos complejos con referencias circulares
* Puede ser difícil manejar la clonación profunda
* Puede duplicar estados no deseados

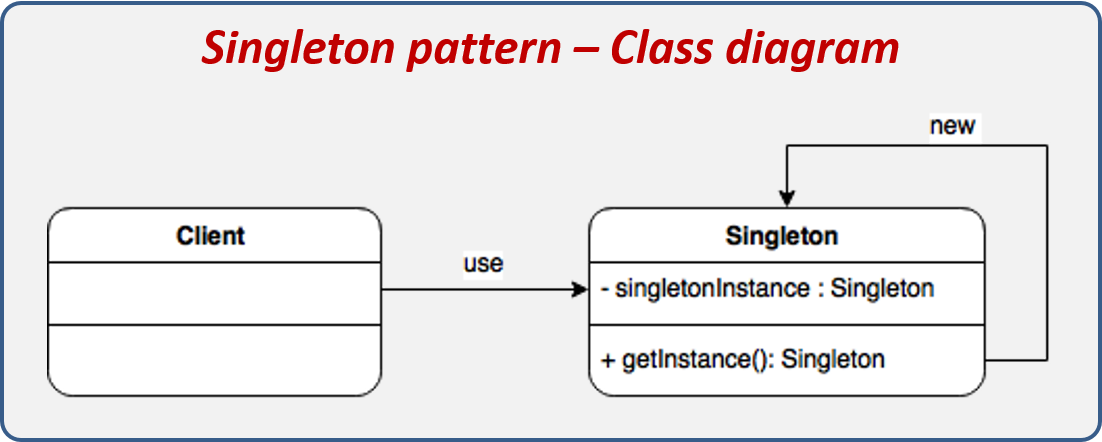
**e) Patrones Relacionales:**

* Singleton
* Factory Method

**f) Aspectos donde es utilizado:**

* Creación de objetos en juegos
* procesamiento de imágenes

**5. Singleton**



**a) Categoría:** Creacional

**b) Problema que resuelve:** Garantiza que solo exista una instancia de una clase y proporciona un punto de acceso global

**c) Implicaciones:**

* Garantiza una única instancia
* Controla el acceso global a los recursos
* Ahorra memoria en comparación con múltiples instancias

**d) Desventajas:**

* Puede crear dependencias ocultas entre clases
* Dificulta el testeo en paralelo
* Viola el principio de responsabilidad única

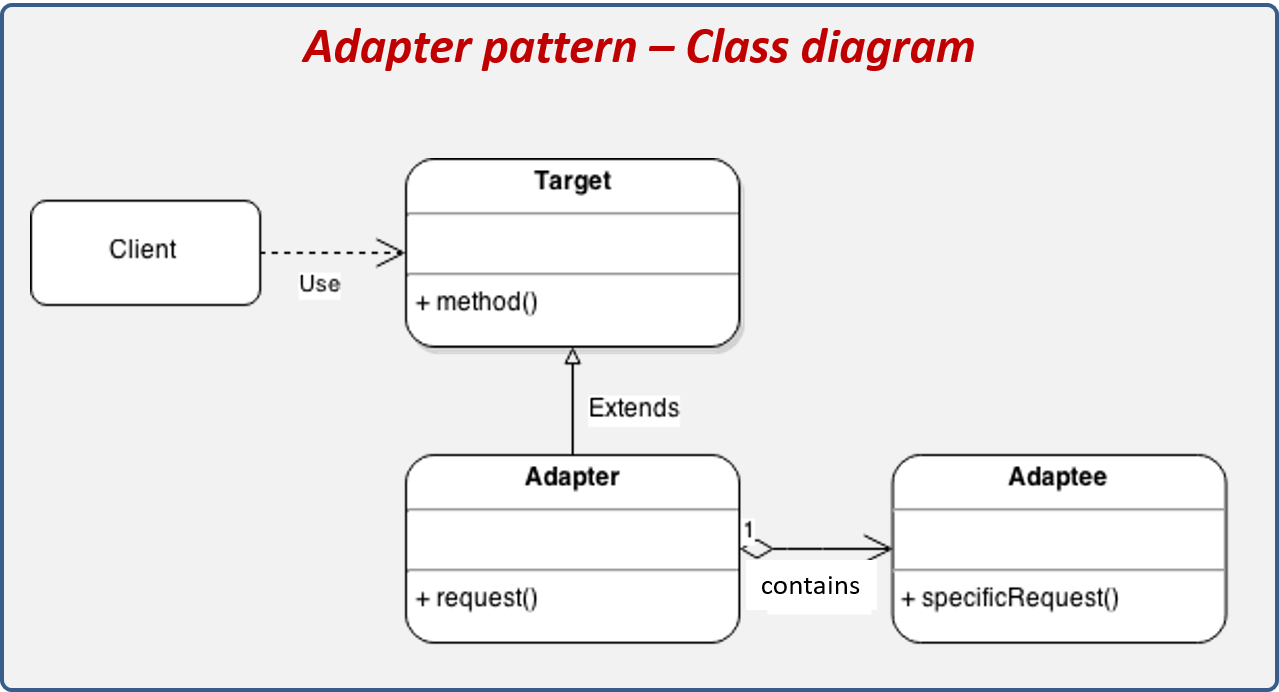
**e) Patrones Relacionales:**

* Abstract Factory
* Prototype

**f) Aspectos donde es utilizado:**

* Manejo de configuraciones globales
* recursos únicos

**6. Adapter**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Permite que interfaces incompatibles trabajen juntas al adaptar una interfaz a otra

**c) Implicaciones:**

* Permite trabajar con interfaces incompatibles.
* Promueve la reutilización de clases existentes.
* Facilita la integración de sistemas heredados

**d) Desventajas:**

* Añade una capa extra de complejidad
* Puede afectar el rendimiento
* La estructura del código puede volverse difícil de seguir si se abusa

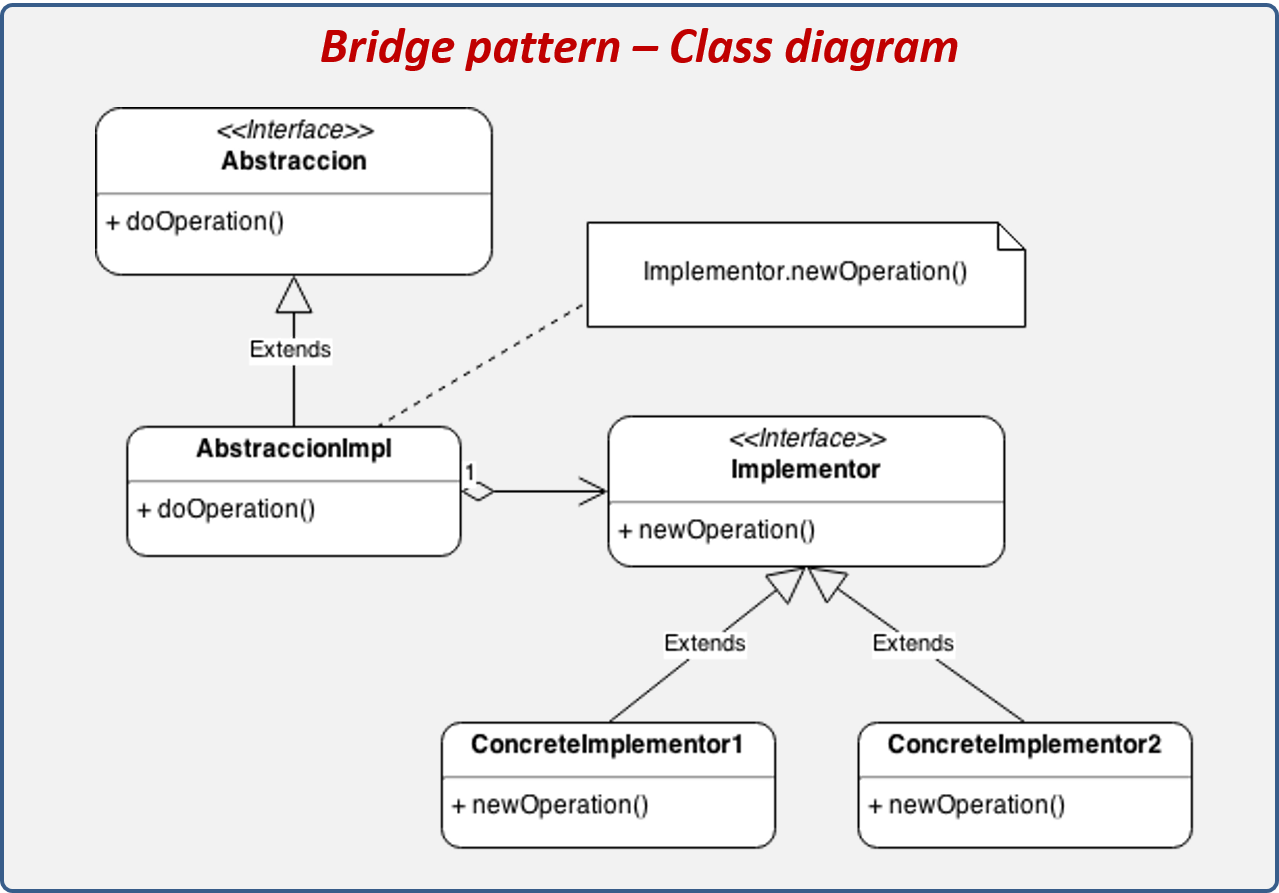
**e) Patrones Relacionales:**

* Bridge
* Proxy

**f) Aspectos donde es utilizado:**

* Drivers
* integración de APIs

**7. Bridge**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Desacopla una abstracción de su implementación, permitiendo variar ambos independientemente

**c) Implicaciones:**

* Desacopla la implementación de la abstracción
* Mejora la flexibilidad y escalabilidad
* Facilita cambios en las implementaciones sin afectar a los clientes

**d) Desventajas:**

* Aumenta el número de clases
* Mayor complejidad inicial en comparación con otras soluciones
* Difícil de implementar en sistemas simples

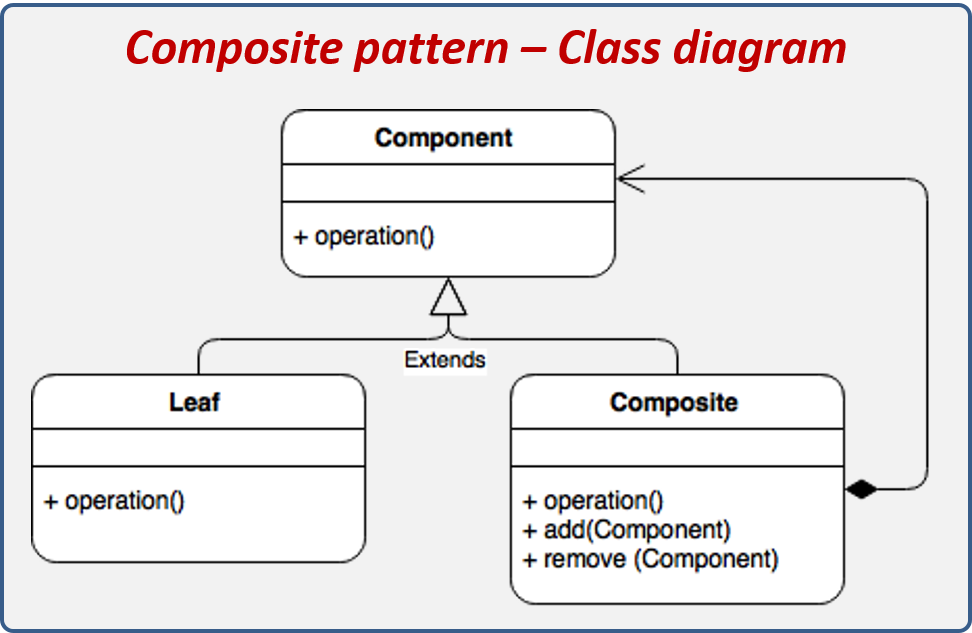
**e) Patrones Relacionales:**

* Adapter
* Strategy

**f) Aspectos donde es utilizado:**

* Interfaces gráficas flexibles
* Manejadores de dispositivos

**8. Composite**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Permite tratar objetos individuales y compuestos de forma uniforme

**c) Implicaciones:**

* Facilita la manipulación de jerarquías de objetos
* Simplifica el código del cliente
* Maneja estructuras complejas con facilidad

**d) Desventajas:**

* Puede ser demasiado general y flexible
* Difícil de restringir las operaciones en componentes específicos
* La gestión de los hijos puede volverse compleja

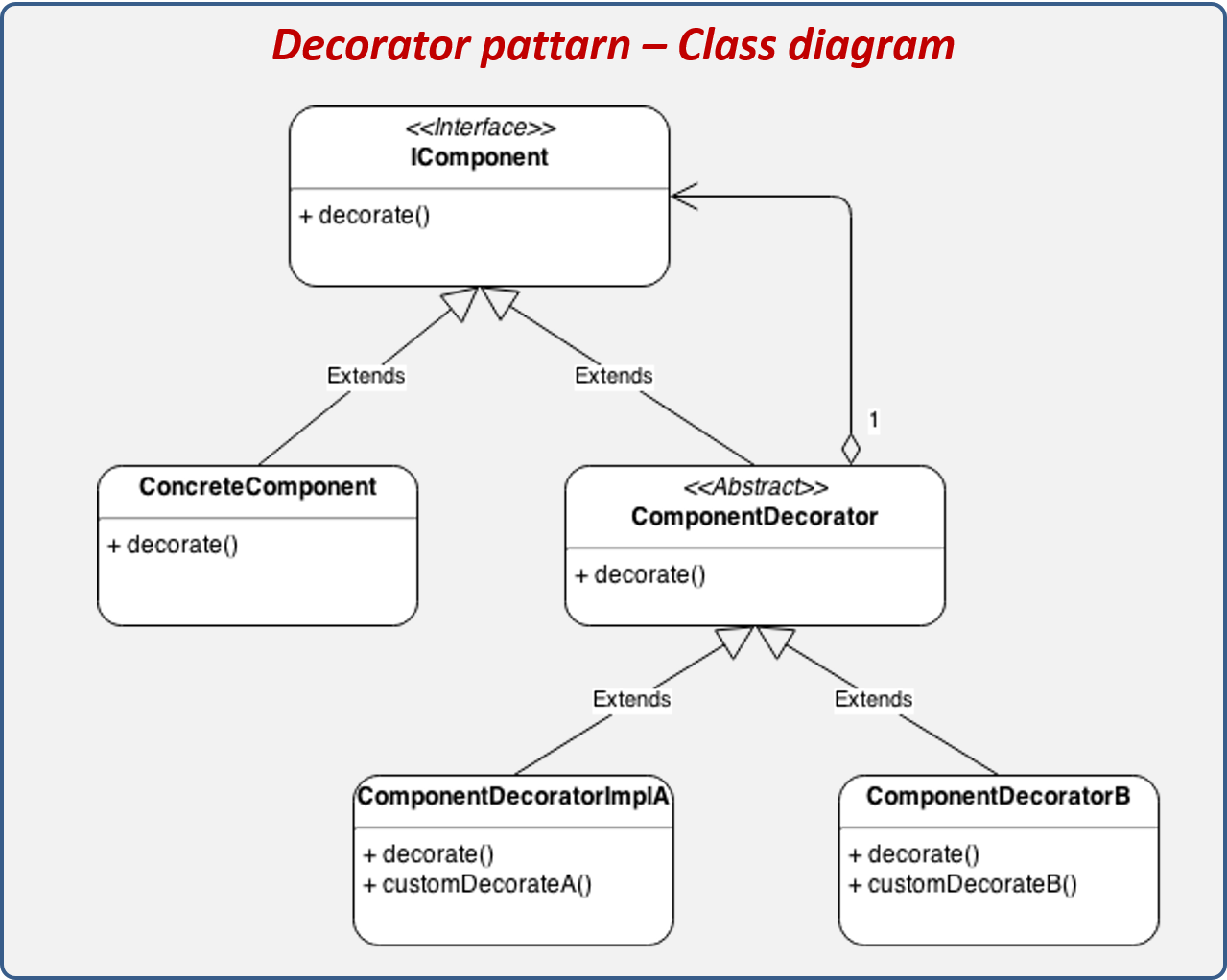
**e) Patrones Relacionales:**

* Decorator
* Flyweight

**f) Aspectos donde es utilizado:** a

* Jerarquías de elementos gráficos
* Estructuras de árbol como menús o documentos

**9. Decorator**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Añade responsabilidades adicionales a un objeto dinámicamente, sin modificar su estructura

**c) Implicaciones:**

* Añade responsabilidades a objetos de manera dinámica
* Fomenta el principio de responsabilidad única
* Evita grandes jerarquías de herencia

**d) Desventajas:**

* Puede generar muchas clases pequeñas y difíciles de gestionar.
* La complejidad aumenta con decoradores anidados.
* Puede ser difícil depurar

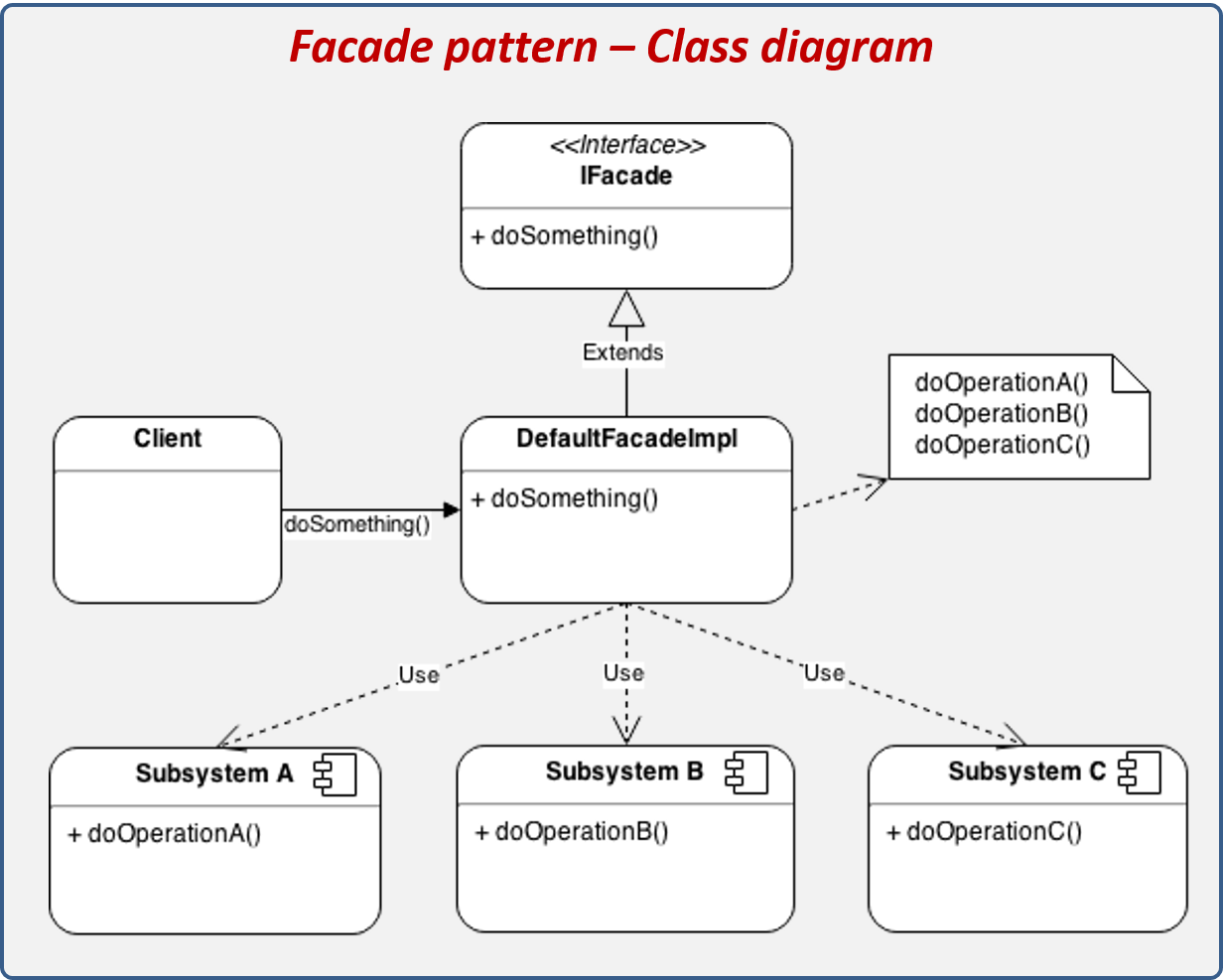
**e) Patrones Relacionales:**

* Composite
* Adapter

**f) Aspectos donde es utilizado:**

* Interfaces gráficas
* Streaming de datos

**10. Facade**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Proporciona una interfaz simplificada a un conjunto de interfaces en un subsistema complejo

**c) Implicaciones:**

* Simplifica la interfaz de un sistema complejo
* Promueve el bajo acoplamiento
* Hace el sistema más fácil de usar

**d) Desventajas:**

* Puede ocultar funcionalidades importantes
* Se puede volver un "Dios" objeto si crece mucho
* Dificulta la extensión si no se diseña bien

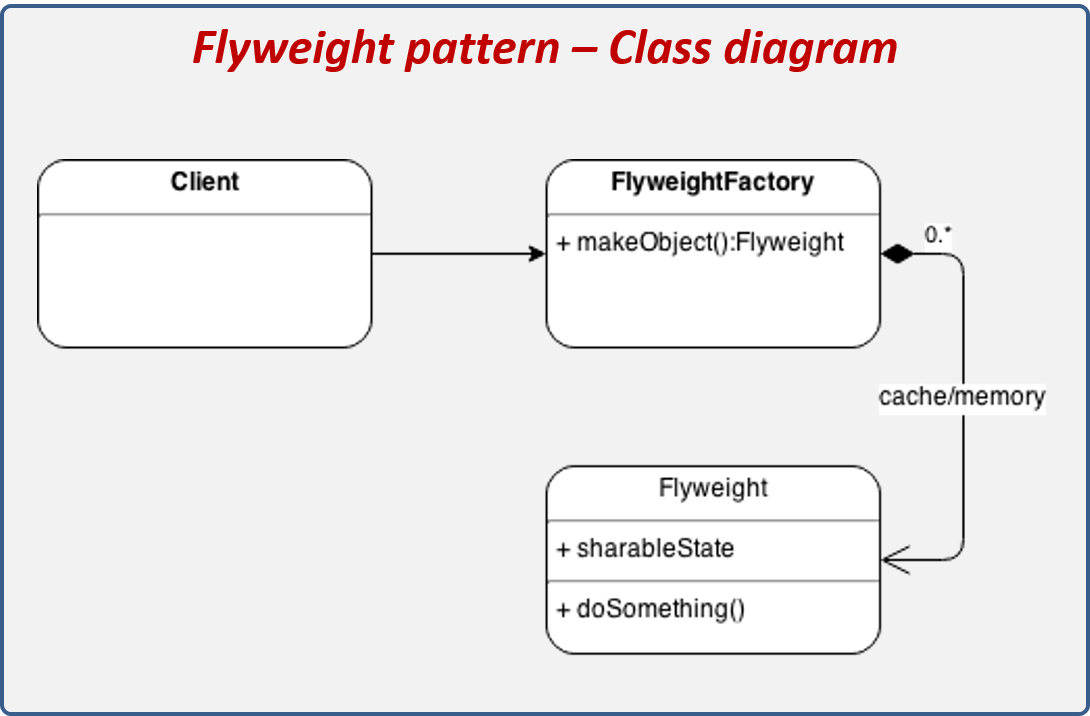
**e) Patrones Relacionales:**

* Mediator
* Singleton

**f) Aspectos donde es utilizado:**

* Librerías complejas
* Capas de interacción en arquitecturas de software complejas

**11. Flyweight**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Minimiza el uso de memoria compartiendo datos entre múltiples objetos similares

**c) Implicaciones:**

* Ahorra memoria al compartir objetos similares
* Aumenta la eficiencia en sistemas con muchos objetos similares
* Escalable en sistemas grandes

**d) Desventajas:**

* Complejidad añadida al gestionar estados compartidos
* Difícil de implementar en sistemas pequeños
* Puede complicar la gestión de ciclos de vida de objetos

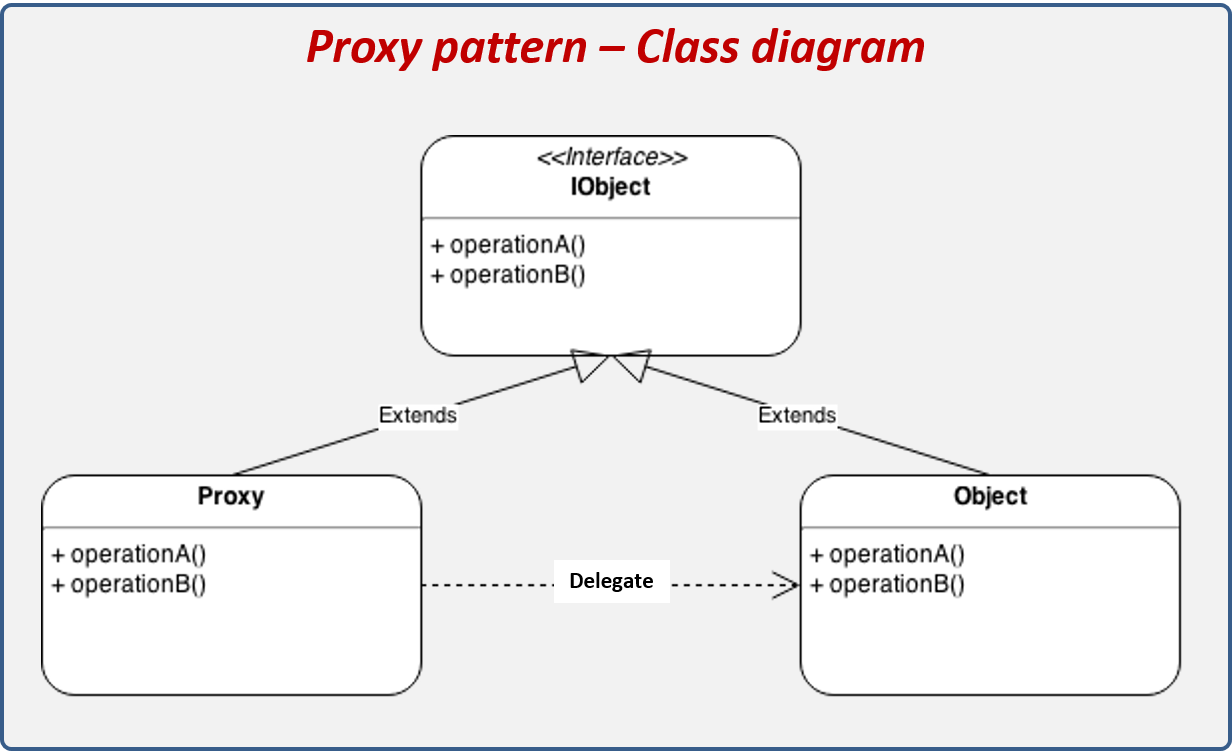
**e) Patrones Relacionales:**

* Composite

**f) Aspectos donde es utilizado:**

* Sistemas de gráficos
* Sistemas de caché
* Manejo de caracteres en editores

**12. Proxy**



**a) Categoría:** Estructural

**b) Problema que resuelve:** Proporciona un sustituto o placeholder para otro objeto para controlar su acceso

**c) Implicaciones:**

* Controla el acceso a objetos
* Proporciona funcionalidades adicionales sin modificar la clase original
* Optimiza recursos al cargar objetos solo cuando se necesitan

**d) Desventajas:**

* Aumenta la complejidad del código
* Puede introducir retardos debido a la intermediación
* Dependencia en el proxy puede complicar pruebas

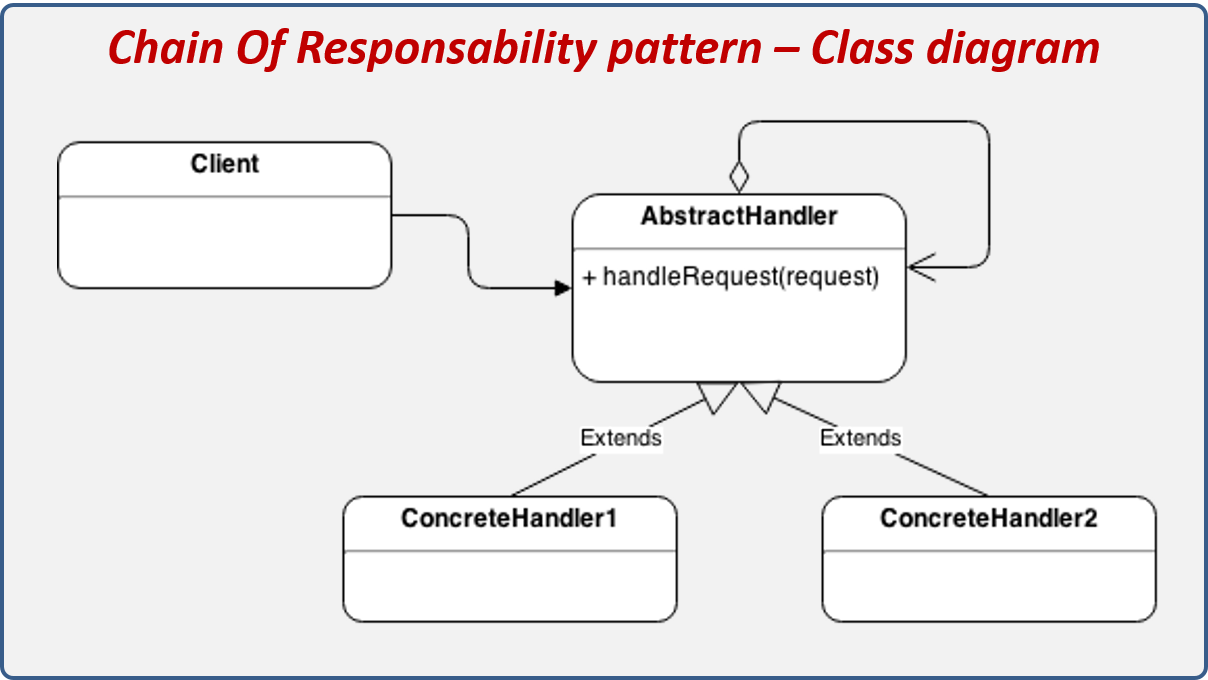
**e) Patrones Relacionales:**

* Decorator
* Adapter
* Singleton

**f) Aspectos donde es utilizado:**

* Control de acceso a objetos remotos
* Servicios diferidos o de alto coste

**13. Chain of Responsibility**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Evita acoplar el remitente de una petición con su receptor, dándole a más de un objeto la oportunidad de manejar la petición

**c) Implicaciones:**

* Flexibilidad para asignar responsabilidades
* Desacopla emisores y receptores de peticiones
* Facilita la extensión y modificación de responsabilidades

**d) Desventajas:**

* No hay garantía de que la solicitud sea procesada
* Puede ser difícil seguir el flujo de las solicitudes
* Las cadenas largas pueden afectar el rendimiento

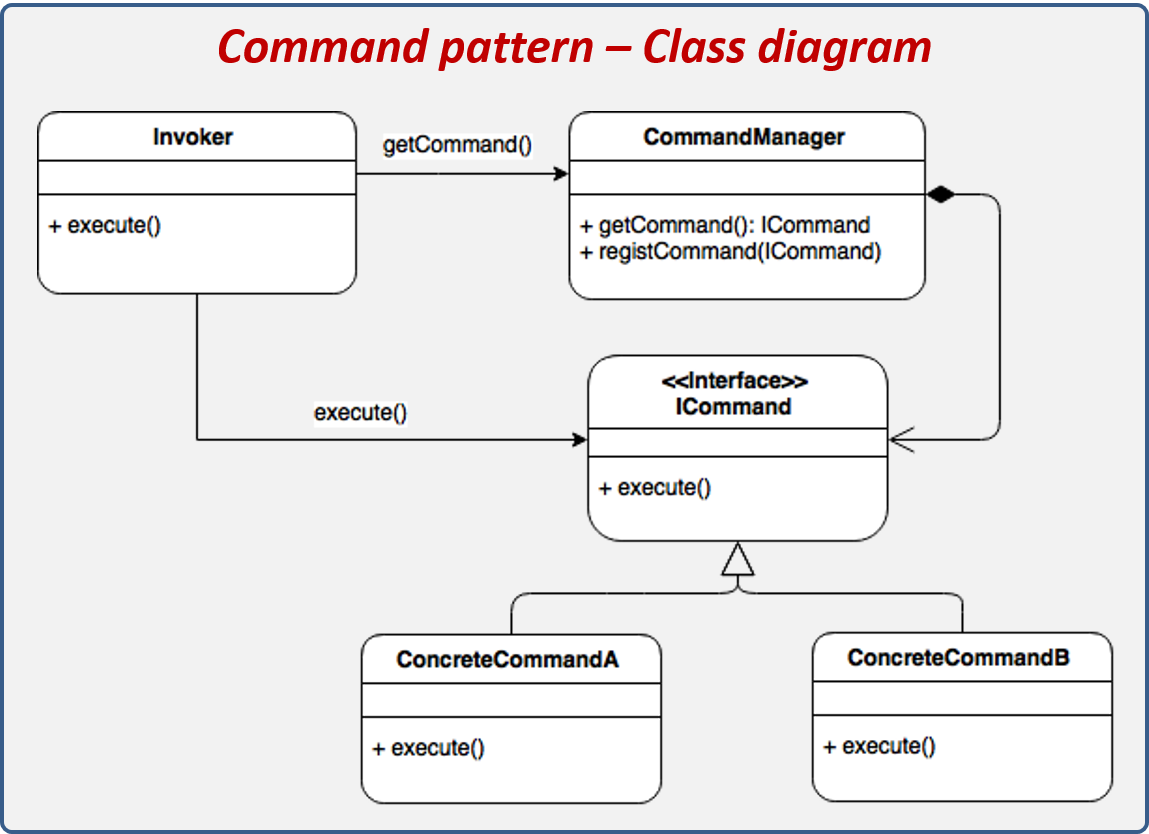
**e) Patrones Relacionales:**

* Mediator
* Command

**f) Aspectos donde es utilizado:**

* Manejo de eventos en sistemas gráficos
* Procesamiento de solicitudes

**14. Command**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Encapsula una solicitud como un objeto, permitiendo parametrizar los clientes con colas de solicitudes, registros de operaciones y más

**c) Implicaciones:**

* Permite operaciones reversibles (deshacer/rehacer).
* Desacopla emisores de receptores.
* Flexibiliza las operaciones encoladas o diferidas

**d) Desventajas:**

* Aumenta el número de clases
* Puede complicar la gestión de múltiples comandos
* Sobrecarga innecesaria para operaciones simples

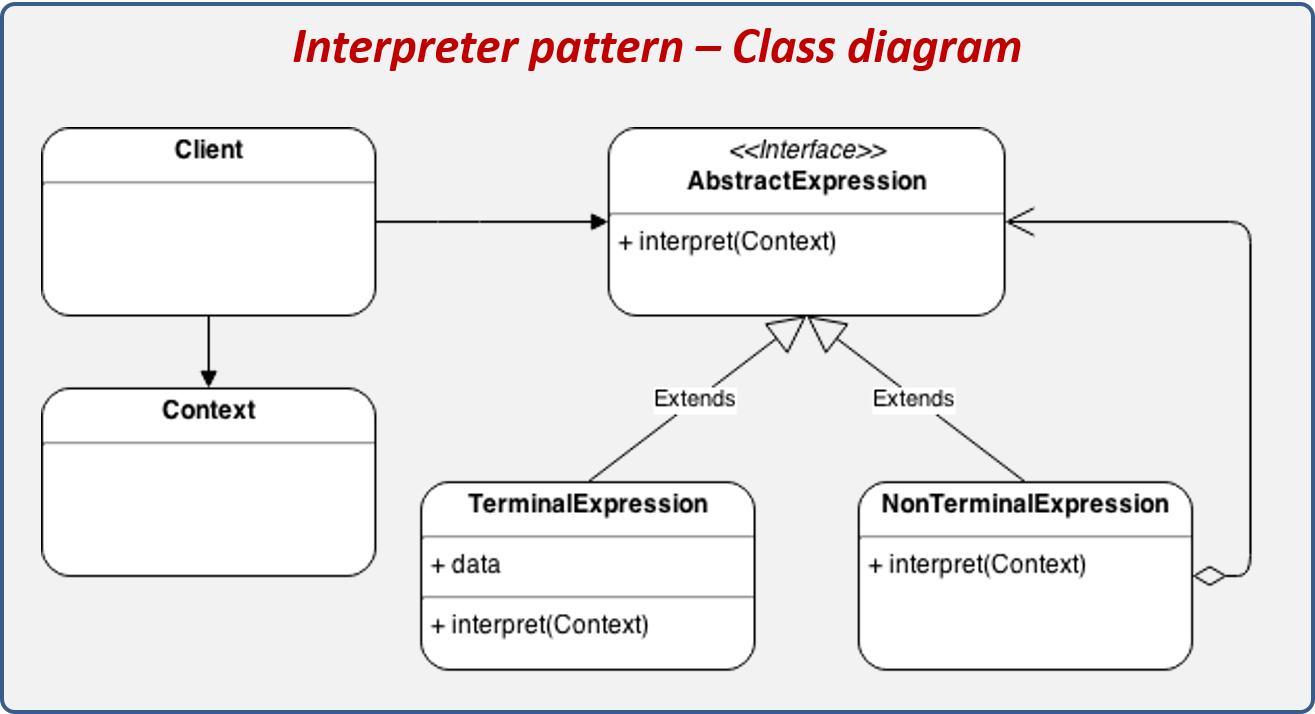
**e) Patrones Relacionales:**

* Mediator
* Chain of Responsibility

**f) Aspectos donde es utilizado:**

* Sistemas de colas
* Deshacer/rehacer operaciones en editores

**15. Interpreter**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Dado un lenguaje, define una representación para su gramática junto con un intérprete que utiliza dicha representación para interpretar sentencias

**c) Implicaciones:**

* Facilita la implementación de lenguajes y gramáticas específicos
* Extensible para nuevas reglas de interpretación
* Fomenta el uso de gramáticas formales

**d) Desventajas:**

* Ineficiente para gramáticas complejas
* Difícil de mantener con muchas reglas
* Puede ser complicado escalar con lenguajes grandes

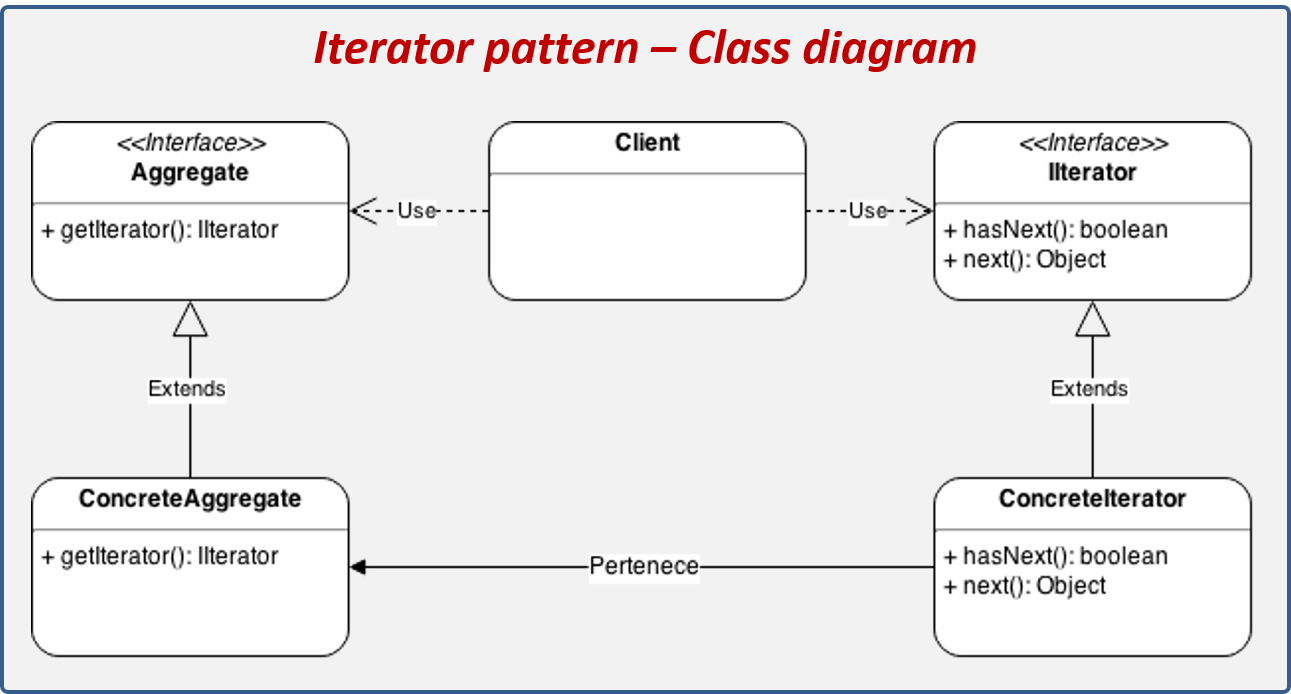
**e) Patrones Relacionales:**

* Flyweight
* Composite

**f) Aspectos donde es utilizado:**

* Compiladores
* Evaluadores de expresiones matemáticas y lógicas

**16. Iterator**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Proporciona una manera de acceder a los elementos de un objeto agregado secuencialmente sin exponer su representación subyacente

**c) Implicaciones:**

* Facilita la navegación por colecciones sin exponer su estructura interna
* Proporciona una interfaz uniforme
* Promueve el desacoplamiento

**d) Desventajas:**

* Puede añadir complejidad a colecciones simples
* Aumenta el número de clases
* Puede ser ineficiente con colecciones muy grandes si no se optimiza

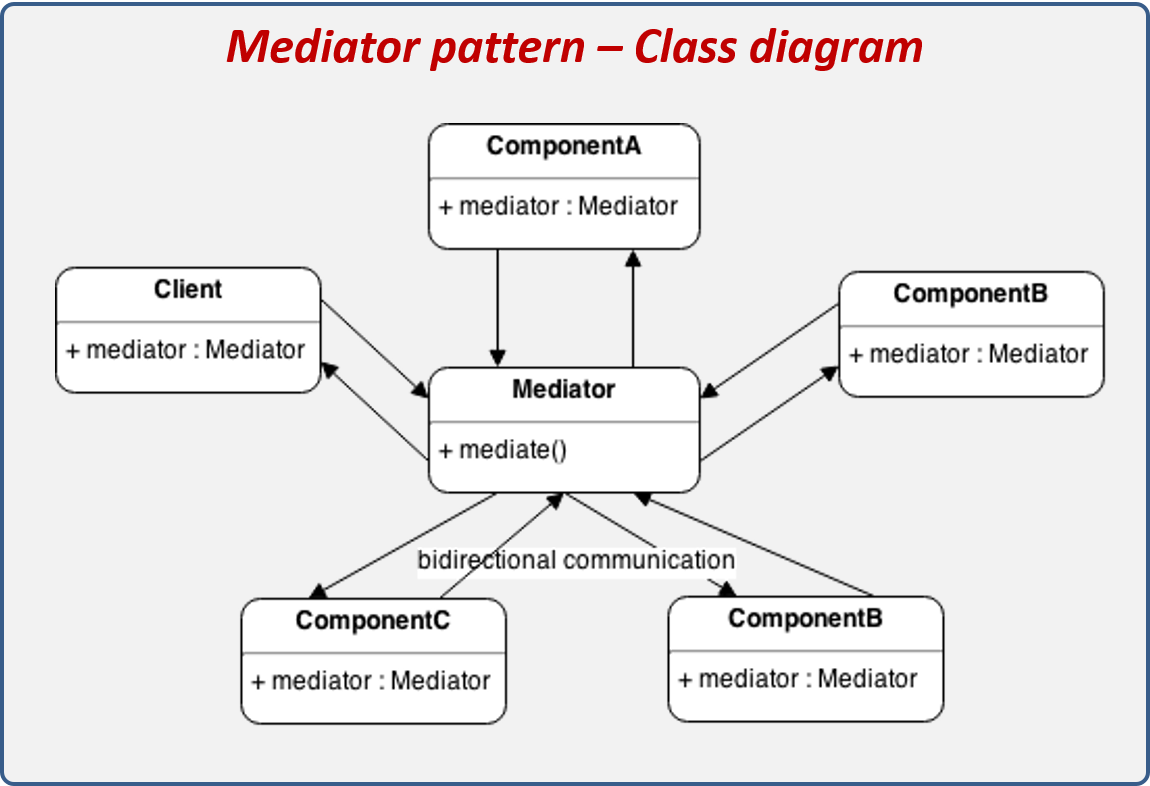
**e) Patrones Relacionales:**

* Composite
* Factory Method

**f) Aspectos donde es utilizado:**

* Listas
* Colecciones de objetos

**17. Mediator**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Define un objeto que encapsula cómo interactúan un conjunto de objetos, promoviendo el bajo acoplamiento entre ellos

**c) Implicaciones:**

* Reduce las dependencias entre los componentes
* Facilita la comunicación entre objetos
* Simplifica la gestión de interacciones complejas

**d) Desventajas:**

* Puede centralizar demasiado la lógica en el mediador
* Aumenta la complejidad si no se gestiona bien
* Difícil de mantener si crece mucho

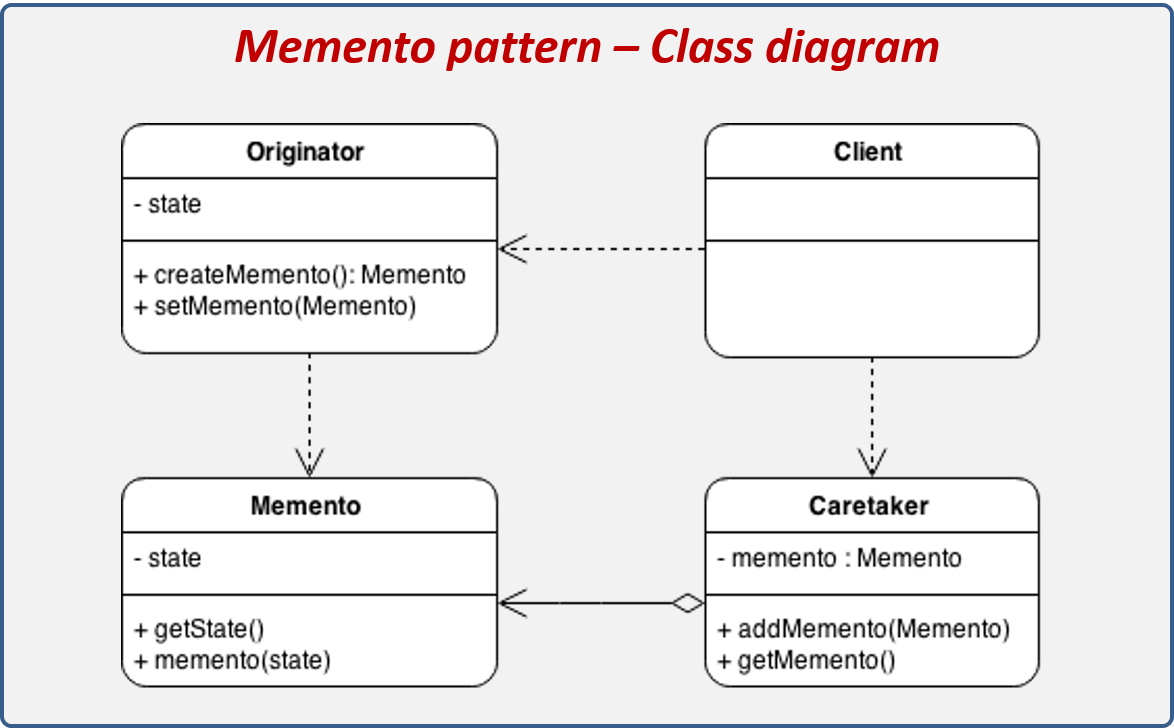
**e) Patrones Relacionales:**

* Observer
* Command

**f) Aspectos donde es utilizado:**

* Manejo de diálogos entre ventanas en interfaces gráficas
* Comunicaciones en sistemas distribuidos

**18. Memento**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Captura y externaliza el estado interno de un objeto sin violar la encapsulación, permitiendo restaurar dicho estado más adelante

**c) Implicaciones:**

* Permite guardar y restaurar el estado de un objeto
* Aísla el código del cliente de los detalles de implementación
* Facilita la implementación de deshacer/rehacer

**d) Desventajas:**

* Aumenta el uso de memoria
* Puede complicar la gestión de estados
* No es adecuado para objetos con estados complejos o dependientes

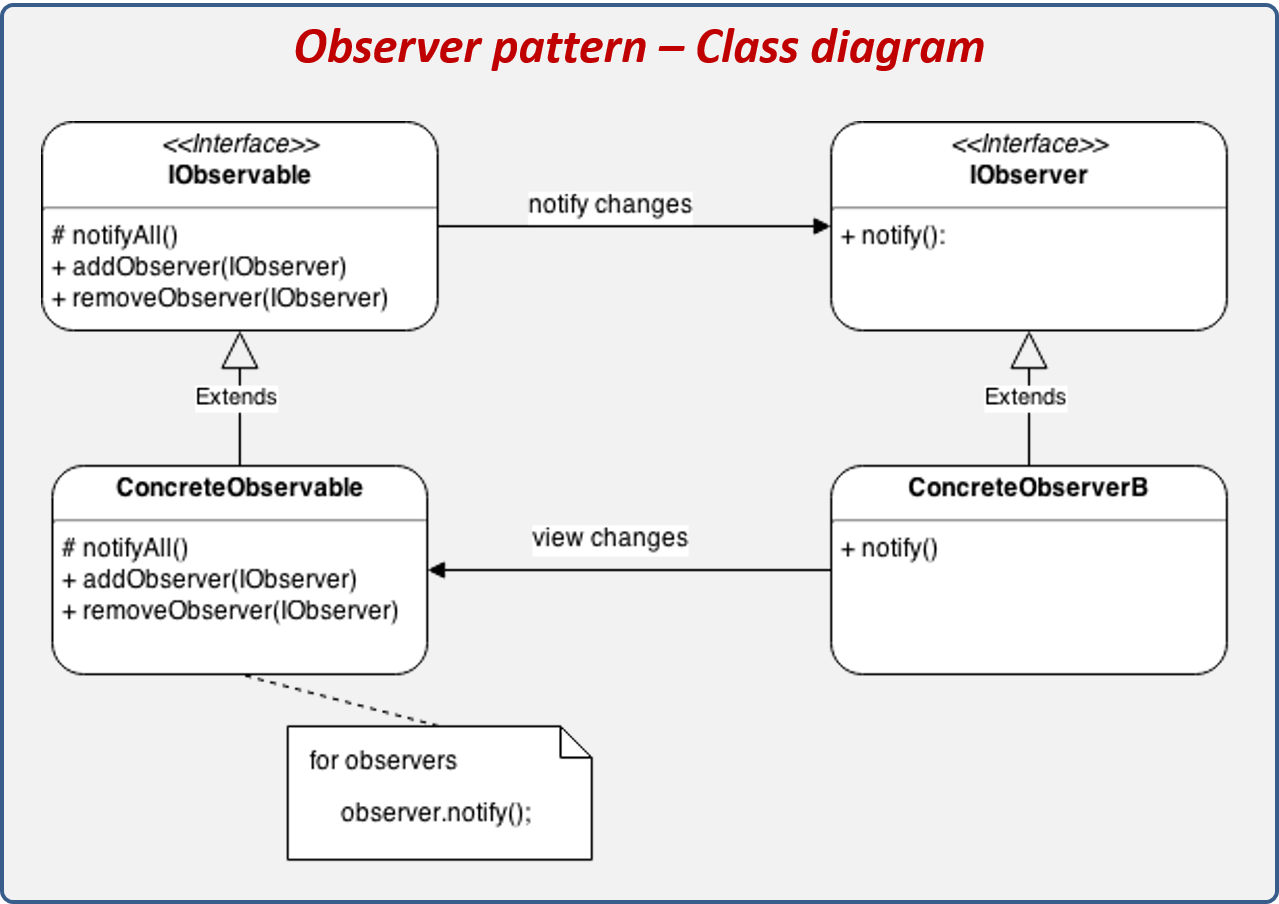
**e) Patrones Relacionales:**

* Command
* Observer

**f) Aspectos donde es utilizado:**

* Deshacer/rehacer en editores
* Sistemas de caché

**19. Observer**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Define una dependencia uno a muchos entre objetos de manera que cuando uno cambia de estado, todos sus dependientes son notificados y actualizados automáticamente

**c) Implicaciones:**

* Desacopla el sujeto de los observadores
* Permite actualizaciones automáticas
* Promueve flexibilidad para agregar o eliminar observadores

**d) Desventajas:**

* Puede volverse ineficiente si hay muchos observadores
* La gestión de errores puede ser compleja
* Las dependencias circulares son difíciles de gestionar

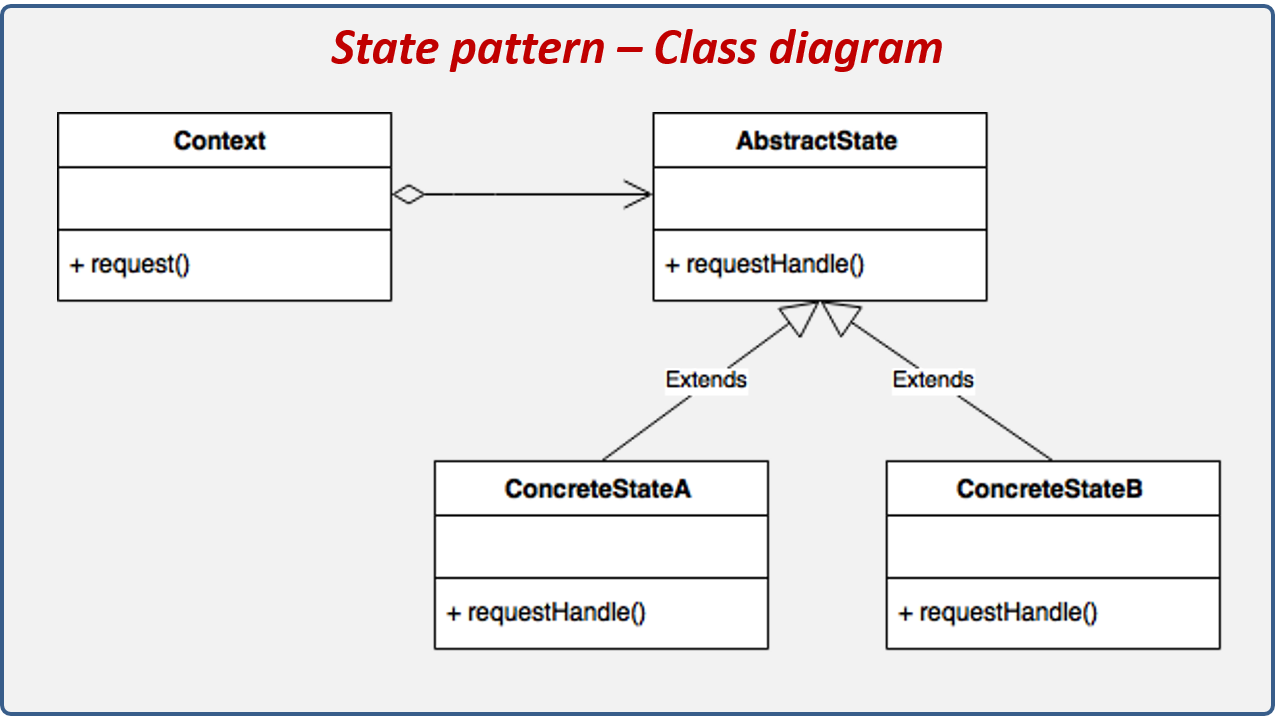
**e) Patrones Relacionales:**

* Mediator
* Singleton

**f) Aspectos donde es utilizado:**

* Suscripción a eventos
* Sistemas de notificaciones

**20. State**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Permite a un objeto alterar su comportamiento cuando su estado interno cambia. El objeto parecerá cambiar su clase

**c) Implicaciones:**

* Facilita la gestión de estados complejos
* Promueve el bajo acoplamiento entre estados
* Permite la modificación del comportamiento dinámicamente

**d) Desventajas:**

* Aumenta el número de clases
* Puede ser difícil gestionar la transición entre estados
* Aumenta la complejidad del código

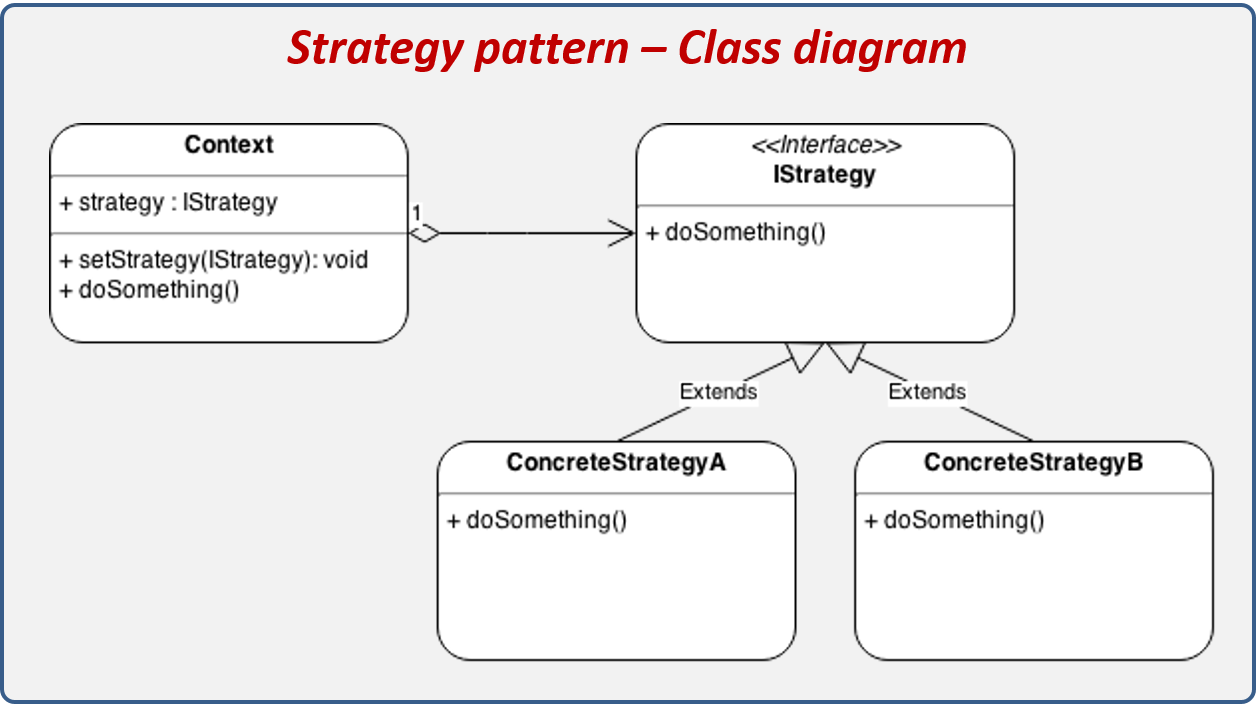
**e) Patrones Relacionales:**

* Strategy
* Memento

**f) Aspectos donde es utilizado:**

* Sistemas de automatización
* Máquinas de estados finitos

**21. Strategy**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Define una familia de algoritmos, encapsula cada uno, y los hace intercambiables. Permite que el algoritmo varíe independientemente de los clientes que lo usan

**c) Implicaciones:**

* Permite cambiar algoritmos en tiempo de ejecución
* Aísla el código del cliente de los algoritmos
* Facilita la reutilización de algoritmos

**d) Desventajas:**

* Aumenta el número de clases
* El cliente debe conocer las estrategias para usarlas
* Puede ser complejo si hay muchas estrategias diferentes

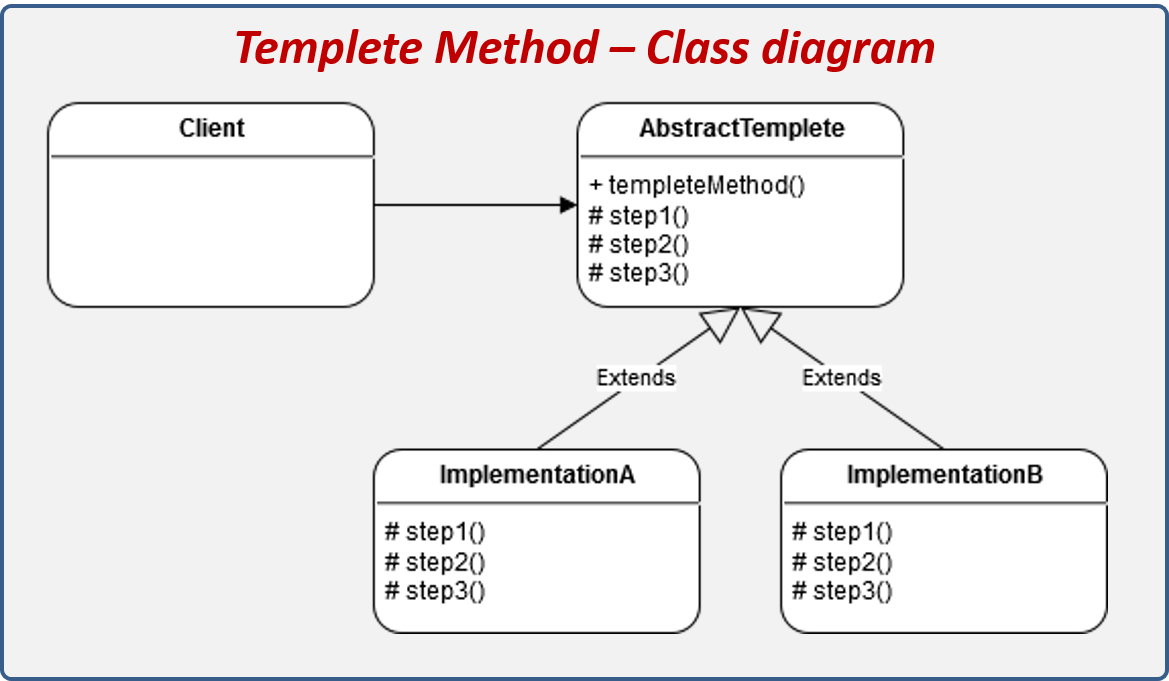
**e) Patrones Relacionales:**

* State
* Template Method

**f) Aspectos donde es utilizado:**

* Algoritmos de ordenación
* Motores de recomendaciones

**22. Template Method**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Define el esqueleto de un algoritmo en una operación, dejando que las subclases reemplacen ciertos pasos sin cambiar la estructura del algoritmo

**c) Implicaciones:**

* Define la estructura de un algoritmo en una clase base
* Permite la reutilización de código
* Facilita la modificación de pasos específicos del algoritmo

**d) Desventajas:**

* Puede limitar la flexibilidad si los pasos son fijos
* Aumenta la complejidad con muchos pasos
* Las subclases deben ser cuidadosas con el método base

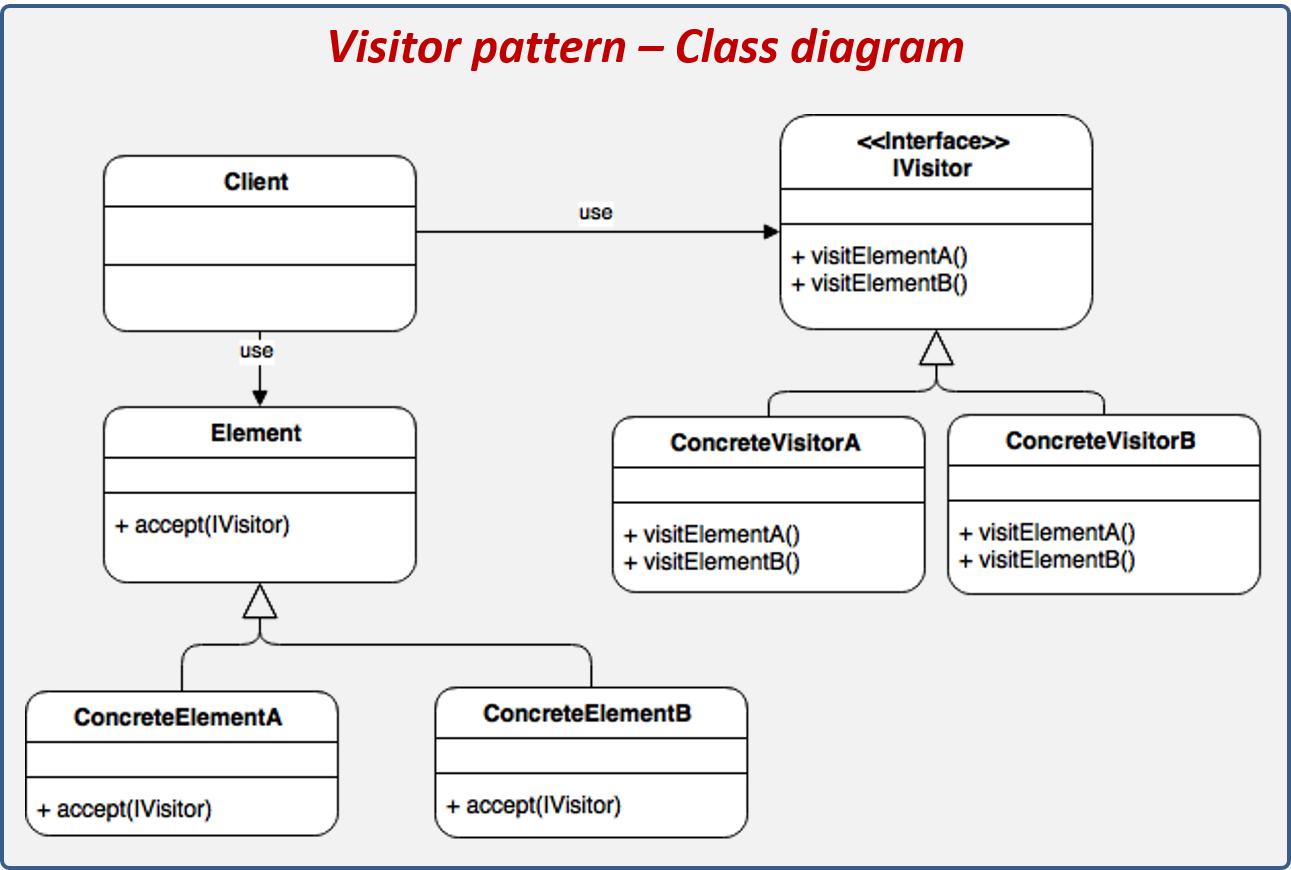
**e) Patrones Relacionales:**

* Strategy
* Factory Method

**f) Aspectos donde es utilizado:**

* Definición de algoritmos base en frameworks
* Pasos de procesos fijos

**23. Visitor**



**a) Categoría:** Comportamiento

**b) Problema que resuelve:** Representa una operación que se realiza sobre los elementos de una estructura de objetos, permitiendo definir nuevas operaciones sin cambiar las clases de los elementos

**c) Implicaciones:**

* Permite agregar nuevas operaciones sin modificar las clases de los elementos
* Facilita la separación de operaciones y objetos
* Fomenta la extensibilidad de operaciones

**d) Desventajas:**

* Aumenta la complejidad con muchos visitantes
* Puede ser difícil de mantener con muchas estructuras
* Cambios en la estructura de elementos pueden requerir cambios en todos los visitantes

**e) Patrones Relacionales:**

* Composite
* Interpreter

**f) Aspectos donde es utilizado:**

* Compiladores
* Sistemas que operan sobre estructuras de datos complejas