Patrones de diseño

5 de octubre de 2019

Índice general

Capítulo 1

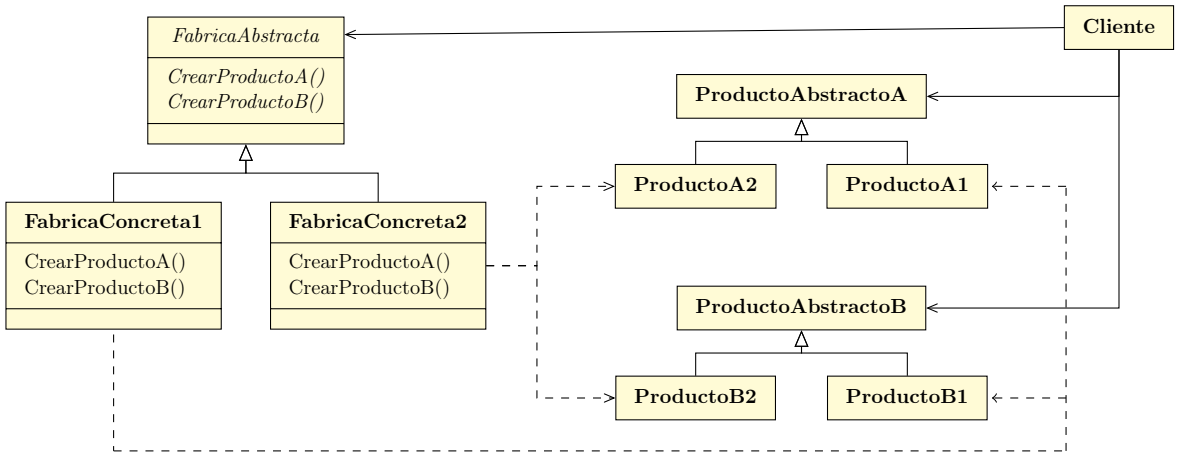
Patrones de creación

1.1. Abstract Factory (COMPLETAR)

1.1.1. Propósito (COMPLETAR)

1.1.2. Aplicabilidad (COMPLETAR)

1.1.3. Estructura



1.1.4. Participantes (COMPLETAR)

1.1.5. Colaboraciones (COMPLETAR)

1.1.6. Consecuencias (COMPLETAR)

1.1.7. Patrones relacionados (COMPLETAR)

1.1.8. Documentación

Pattern	**Nombre**
based on	Abstract Factory
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.
where	*FabricaAbstracta* **is** *crearProductoA()* **is** *crearProductoB()* **is** FabricaConcreta1 **is** FabricaConcreta2 **is** *ProductoAbstractoA* **is** ProductoA1 **is** ProductoA2 **is** *ProductoAbstractoB* **is** ProductoB1 **is** ProductoB2 **is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño

Capítulo 2

Patrones estructurales

2.1. Birdge

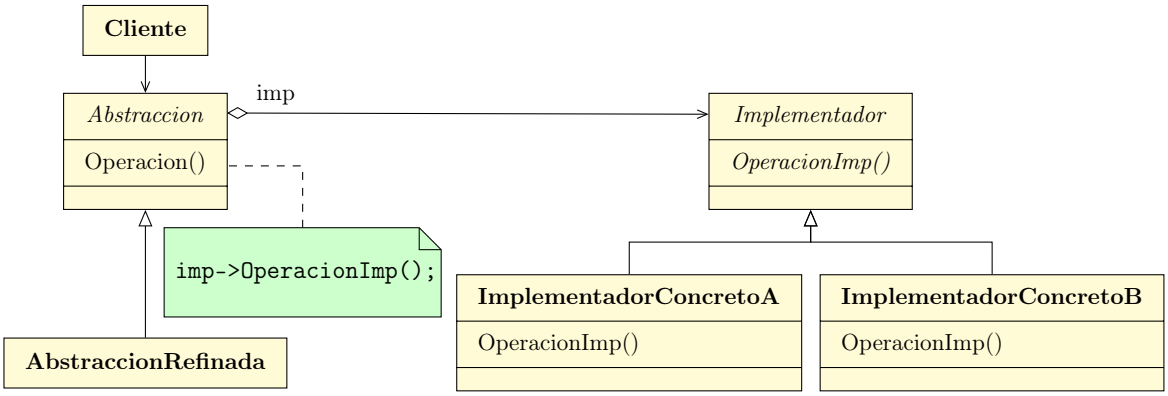
2.1.1. Propósito

Desacopla una abstracción de una implementación de manera tal que ambas puedan variar de forma independiente.

2.1.2. Aplicabilidad

- Evitar un enlace permanente entre una abstracción y su implementación. Por ejemplo, cuando debe seleccionarse o cambiarse la implementación en tiempo de ejecución.
- Evitar que los cambios en la implementación de una abstracción impacten en el código cliente; es decir, su código no tendría que ser recompilado.
- Extender de forma independiente las diferentes abstracciones y sus implementaciones.
- Compartir una implementación entre varios objetos.

2.1.3. Estructura



2.1.4. Participantes

Abstracción

- Define la interfaz de abstracción.
- Mantiene una referencia a un objeto de tipo Implementador.

Abstracción Refinada

- Extiende la interfaz definida por Abstracción.

Implementador

- Define la interfaz de las clases de implementación. Esta interfaz no tiene por que corresponderse exactamente con la de Abstracción; de hecho, ambas interfaces pueden ser muy distintas. Normalmente la interfaz Implementador solo proporciona operaciones primitivas, y Abstracción define operaciones de mas alto nivel basadas en dichas primitivas.

Implementador Concreto

- Implementa la interfaz Implementador y define su implementación concreta.

2.1.5. Colaboraciones

Abstracción redirige las peticiones del cliente a su objeto Implementador.

2.1.6. Consecuencias

- *Desacopla la interfaz y la implementación.* No une permanentemente una implementación a una interfaz, sino que la implementación puede configurarse en tiempo de ejecución. Incluso es posible que un objeto cambie su implementación en tiempo de ejecución.

  Además, este desacoplamiento potencia una división en capas que puede dar lugar a sistemas mejor estructurados. La parte de alto nivel de un sistema sólo tiene que conocer a Abstracción y a Implementador.

- *Mejora la extensibilidad.* Podemos extraer las jerarquías de Abstracción y de Implementador de forma independiente.

- *Oculta detalles de implementación a los clientes.* Podemos aislar a los clientes de los detalles de implementación, como el compartimiento de objetos implementadores y el correspondiente mecanismo de conteo de referencias (si es que hay alguno).

2.1.7. Patrones relacionados

- El patrón Abstract Factory puede crear y configurar el Bridge.

- ~~El patrón Adapter esta orientado a conseguir que trabajen juntas clases que no están relacionadas. Normalmente se aplica a sistemas que ya han sido diseñados. El patrón Bridge, por otro lado, se usa al comenzar un diseño para permitir que abstracciones e implementaciones varíen independientemente unas de otras.~~

2.1.8. Documentación

Pattern based on	**Nombre en del diseño** Bridge		
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.		
where	*Abstraccion*	**is**	Elemento del diseño
	operacion()	**is**	Elemento del diseño
	imp	**is**	Elemento del diseño
	AbstraccionRefinada	**is**	Elemento del diseño
	Implementador	**is**	Elemento del diseño
	operacionImp()	**is**	Elemento del diseño
	ImplementadorConcretoA	**is**	Elemento del diseño
	ImplementadorConcretoB	**is**	Elemento del diseño
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño		

2.2. Composite

2.2.1. Propósito

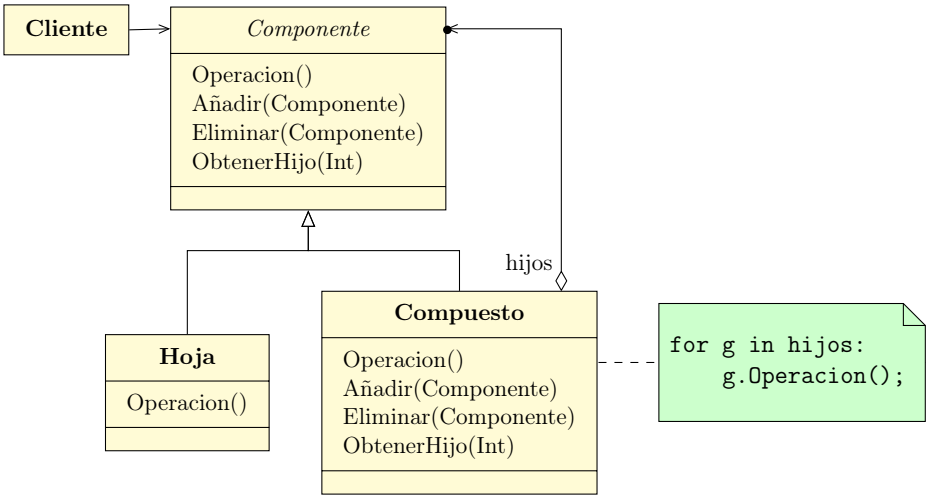
Compone objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

2.2.2. Aplicabilidad

- Representar jerarquías de objetos parte-todo.
- Abstraer las diferencias entre composiciones de objetos y objetos individuales. Los clientes tratarán a todos los objetos de la estructura

compuesta de manera uniforme.

2.2.3. Estructura



2.2.4. Participantes

Componente

- Declara la interfaz de los objetos de la composición.
- Implementa el comportamiento predeterminado de la interfaz que es común a todas las clases.
- Declara una interfaz para acceder a sus componentes hijos y gestionarlos.
- Opcionalmente, define una interfaz para acceder al padre de un componente en la estructura recursiva y, si es necesario, la implementa.

Hoja

- Representa objetos hoja en la composición. Una hoja no tiene hijos.
- Define el comportamiento de los objetos primitivos de la composición.

Compuesto

- Define el comportamiento de los componentes que tienen hijos.
- Almacena componentes hijos.
- Implementa las operaciones de la interfaz Componente relacionadas con los hijos.

Cliente

- Manipula objetos en la composición a través de la interfaz Componente.

2.2.5. Colaboraciones

Los Clientes usan la interfaz de la clase Componente para interactuar con los objetos de la estructura compuesta. Si el recipiente es una Hoja, la petición se trata correctamente. Si es un Compuesto, normalmente redirige las peticiones a sus componentes hijos, posiblemente realizando operaciones adicionales antes o después.

2.2.6. Consecuencias

- Define jerarquías de clases formadas por objetos primitivos y compuestos. Allí donde el código espere un objeto primitivo, también podrá recibir un objeto compuesto.
- Simplifica el cliente. Los cliente pueden tratar uniformemente a las estructuras compuestas y a los objetos individuales.
- Facilita añadir nuevos tipos de componentes. Si se definen nuevas subclases Compuesto u Hoja, éstas funcionarán automáticamente con las estructuras y el código cliente existentes. No hay que cambiar los clientes para las nuevas clases Componente.
- Puede hacer que un diseño sea demasiado general. La desventaja de facilitar añadir nuevos componentes es que hace más difícil restringir los componentes de un compuesto.

2.2.7. Patrones relacionados

- ~~Muchas veces se usa el enlace al componente padre para implementar el patrón Chain of Responsibility.~~
- El patrón Decorator (Wrapper) suele usarse junto con el Composite. Cuando se usan juntos decoradores y compuestos, normalmente ambos tendrán una clase padre común. Por lo tanto, los decoradores tendrán que admitir la interfaz Componente con operaciones como Añadir, Eliminar y ObtenerHijo.
- ~~El patron Flyweight permite compartir componentes, si bien en ese caso éstos ya no pueden referirse a sus padres.~~
- Se puede usar el patrón Iterator para recorrer las estructuras definidas por el patrón Composite.
- El patrón Visitor localiza operaciones y comportamiento que de otro modo estaría distribuido en varias clases Compuesto y Hoja.

2.2.8. Documentación

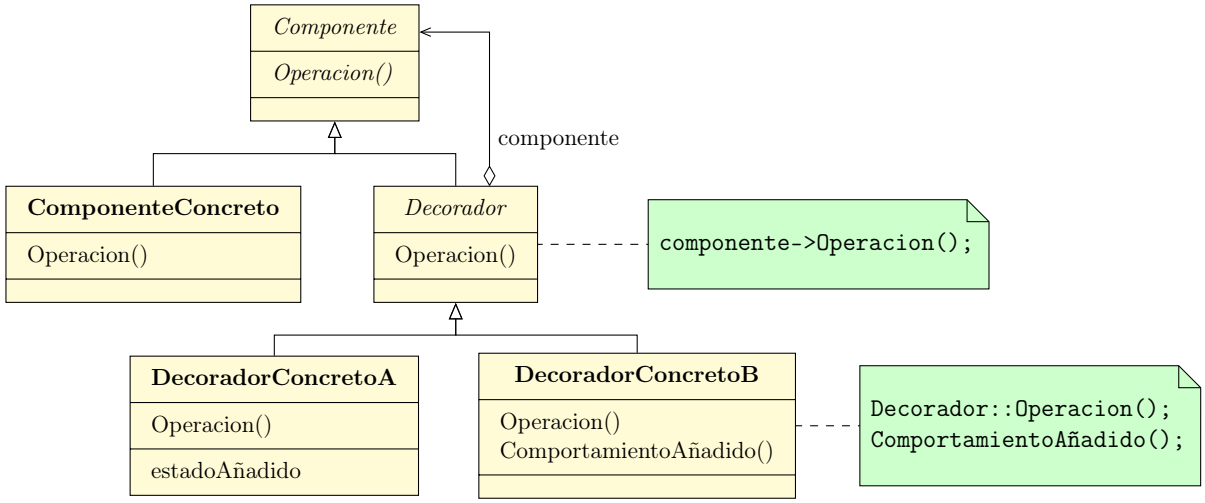
Pattern	**Nombre**	
based on	Composite	
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.	
where	*Componente*	**is**
	operacion()	**is**
	añadir(Componente)	**is**
	eliminar(Componente)	**is**
	obtenerHijo(int)	**is**
	Compuesto	**is**
	Hoja	**is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño	

2.3. Wrapper/Decorator (COMPLETAR)

2.3.1. Propósito (COMPLETAR)

2.3.2. Aplicabilidad (COMPLETAR)

2.3.3. Estructura



2.3.4. Participantes (COMPLETAR)

2.3.5. Colaboraciones (COMPLETAR)

2.3.6. Consecuencias (COMPLETAR)

2.3.7. Patrones relacionados (COMPLETAR)

2.3.8. Documentación

Pattern	**Nombre**	
based on	Wrapper	
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.	
where	*Objetivo*	**is**
	peticion()	**is**
	Adaptable	**is**
	peticionConcreta()	**is**
	Adaptador	**is**
	implementacion	**is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño	

Capítulo 3

Patrones de comportamiento

3.1. Command

3.1.1. Propósito

Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de las peticiones, y poder deshacer las operaciones.

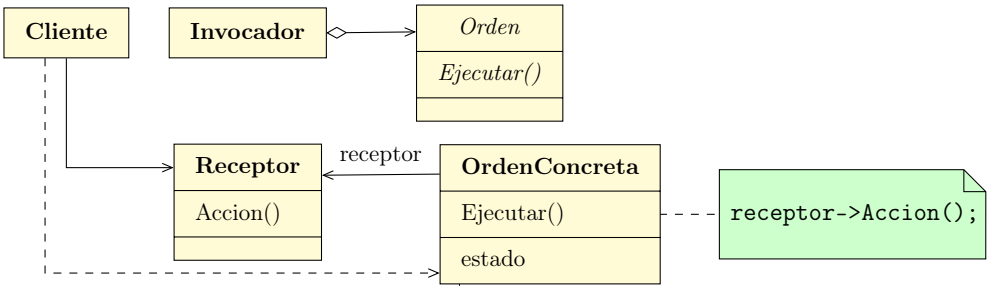
3.1.2. Aplicabilidad

- Parametrizar objetos con una acción a realizar. Los objetos Orden son un sustituto orientado a objetos para las funciones callback.
- Especificar, poner en cola y ejecutar peticiones en diferentes instantes de tiempo. Un objeto Orden puede tener un tiempo de vida independiente de la petición original.
- Permitir deshacer. La operación Ejecutar de Orden puede guardar en la propia orden el estado que anule sus efectos. Debe añadirse a la interfaz Orden una operación Deshacer que anule los efectos de una llamada anterior a Ejecutar. Las órdenes ejecutadas se guardan en una lista que hace las veces de historial.
- Permitir registrar los cambios de manera que se puedan volver a aplicar en caso de una caída del sistema. Aumentando la interfaz Orden con operaciones para cargar y guardar se puede mantener un registro

persistente de los cambios. Recuperarse de una caída implica volver a cargar desde el disco las órdenes guardadas y volver a ejecutarlas con la operación Ejecutar.

- Estructurar un sistema alrededor de operaciones de alto nivel construidas sobre operaciones básicas. Dicha estructura es común en los sistemas de información que permiten transacciones.

3.1.3. Estructura



3.1.4. Participantes

Orden

- Declara una interfaz para ejecutar una operación.

OrdenConcreta

- Define un enlace entre un objeto Receptor y una acción.
- Implementa Ejecutar invocando la correspondiente operación u operaciones del Receptor.

Cliente

- Crea un objeto OrdenConcreta y establece su receptor.

Invocador

- Le pide a la orden que ejecute la petición.

Receptor

- Sabe cómo llevar a cabo las operaciones asociadas a una petición. Cualquier clase puede hacer actuar como Receptor.

3.1.5. Colaboraciones

- El cliente crea un objeto OrdenConcreta y especifica su receptor.
- Un objeto Invocador almacena el objeto OrdenConcreta.
- El invocador envía una petición llamando a Ejecutar sobre la orden. Cuando las órdenes se pueden deshacer, OrdenConcreta guarda el estado para deshacer la orden antes de llamar a Ejecutar.
- El objeto OrdenConcreta invoca operaciones de su receptor para llevar a cabo la petición.

3.1.6. Consecuencias

- Orden desacopla el objeto que invoca la operación de aquél que sabe cómo realizarla.
- Las órdenes son objetos de primera clase. Pueden ser manipulados y extendidos como cualquier otro objeto.
- Se pueden ensamblar órdenes en una orden compuesta. En general, las órdenes compuestas son una instancia del patrón Composite.
- Es mas fácil añadir nuevas órdenes, ya que no hay que cambiar las clases existentes.

3.1.7. Patrones relacionados

- Se puede usar el patrón Composite para implementar ordenes compuestas.
- ~~Un memento puede mantener el estado que necesitan las órdenes para anular (deshacer) sus efectos.~~
- ~~Una orden que debe ser copiada antes de ser guardada en el historial funciona como un Prototype.~~

3.1.8. Documentación

Pattern	**Nombre**	
based on	Command	
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.	
where	Invocador	**is**
	Orden	**is**
	ejecutar()	**is**
	Receptor	**is**
	Accion	**is**
	OrdenConcreta	**is**
	estado	**is**
	receptor	**is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño	

3.2. Iterator

3.2.1. Propósito

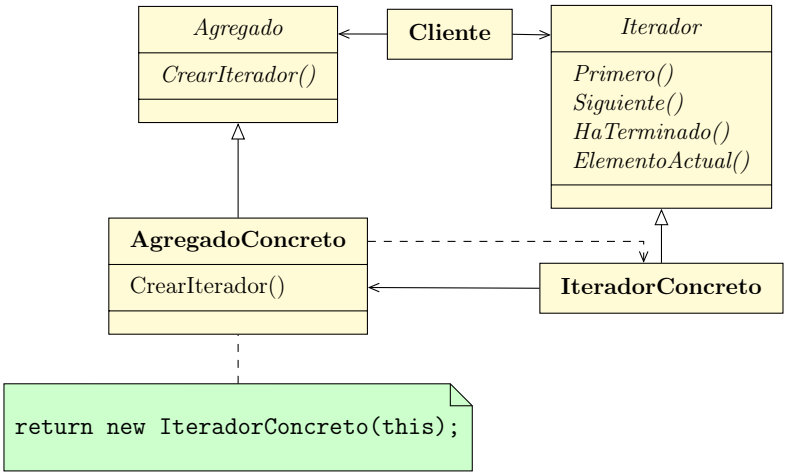
Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

3.2.2. Aplicabilidad

- Acceder al contenido de un objeto agregado sin exponer su representación interna.
- Permitir varios recorridos sobre objetos agregados.

- Proporcionar una interfaz uniforme para recorrer diferentes estructuras agregadas.

3.2.3. Estructura



3.2.4. Participantes

Iterador

- Define una interfaz para recorrer los elementos y acceder a ellos.

IteradorConcreto

- Implementa la interfaz Iterador.
- Mantiene la posición actual en el recorrido del agregado.

Agregado

- Define una interfaz para crear un objeto Iterador.

AgregadoConcreto

- Implementa la interfaz de creación de Iterador para devolver una instancia del IteradorConcreto apropiado.

3.2.5. Colaboraciones

- Un IteradorConcreto sabe cuál es el objeto actual del agregado y puede calcular el objeto siguiente en el recorrido.

3.2.6. Consecuencias

- Permite variaciones en el recorrido de un agregado. Los agregados pueden recorrerse de muchas formas. Los iteradores facilitan cambiar el algoritmo de recorrido: basta con sustituir la instancia de iterador por otra diferente. También se pueden definir subclases de Iterador para permitir nuevos recorridos.
- Los iteradores simplifican la interfaz de los agregados. La interfaz de recorrido de Iterador elimina la necesidad de una interfaz parecida en Agregado, simplificando así la interfaz del agregado.
- Se puede hacer más de un recorrido a la vez sobre un agregado. Un iterador mantiene su propio estado del recorrido, por lo tanto, es posible estar realizando más de un recorrido al mismo tiempo.

3.2.7. Patrones relacionados

- Composite: los iteradores suelen aplicarse a estructuras recursivas como los compuestos.
- ~~Factory Method: los iteradores polimórficos se basan en métodos de fabricación para crear instancias de las subclases apropiadas de Iterator.~~
- ~~El patrón Memento suele usarse conjuntamente con el patrón Iterator. Un iterador puede usar un memento para representar el estado de una interacción. El iterador almacena el memento internamente.~~

3.2.8. Documentación

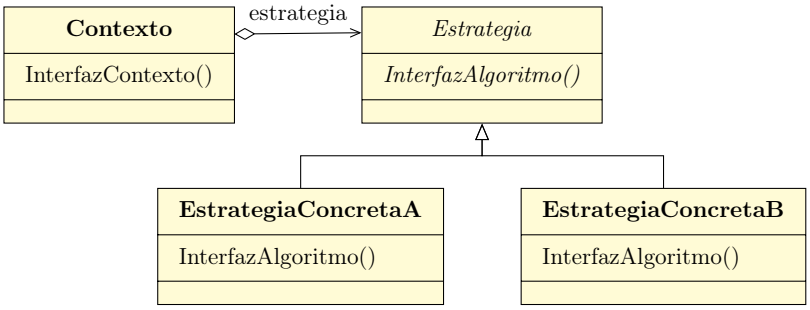
Pattern	**Nombre**	
based on	Iterator	
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.	
where	*Agregado*	**is**
	crearIterador()	**is**
	AgregadoConcreto	**is**
	Iterador	**is**
	primero()	**is**
	siguente()	**is**
	haTerminado()	**is**
	elementoActual()	**is**
	IteradorConcreto	**is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño	

3.3. Strategy (COMPLETAR)

3.3.1. Propósito (COMPLETAR)

3.3.2. Aplicabilidad (COMPLETAR)

3.3.3. Estructura



3.3.4. Participantes (COMPLETAR)

3.3.5. Colaboraciones (COMPLETAR)

3.3.6. Consecuencias (COMPLETAR)

3.3.7. Patrones relacionados (COMPLETAR)

3.3.8. Documentación

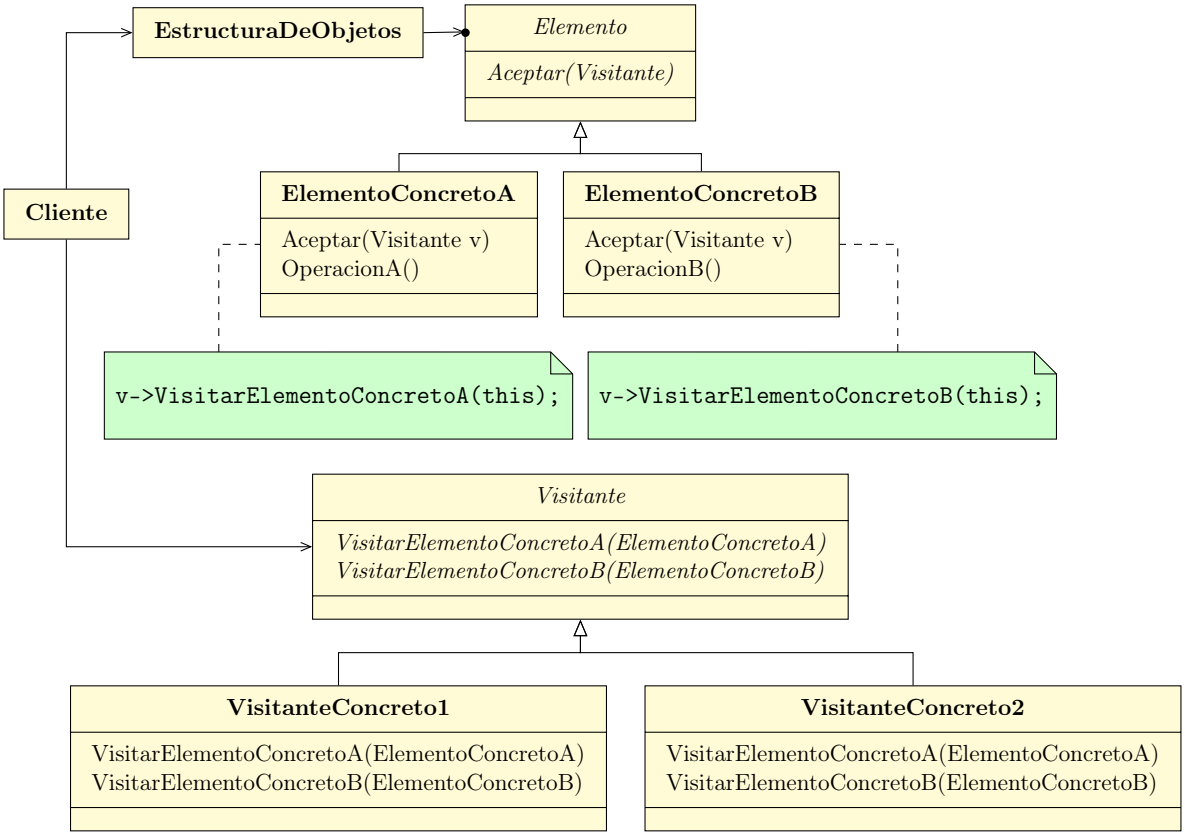
Pattern	**Nombre**
based on	Strategy
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.
where	Contexto **is** interfazContexto() **is** estrategia **is** *Estrategia* **is** *interfazAlgoritmo()* **is** EstrategiaConcretaA **is** EstrategiaConcretaB **is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño

3.4. Visitor (COMPLETAR)

3.4.1. Propósito (COMPLETAR)

3.4.2. Aplicabilidad (COMPLETAR)

3.4.3. Estructura



3.4.4. Participantes (COMPLETAR)

3.4.5. Colaboraciones (COMPLETAR)

3.4.6. Consecuencias (COMPLETAR)

3.4.7. Patrones relacionados (COMPLETAR)

3.4.8. Documentación

Pattern	**Nombre**	
based on	Visitor	
because	Fundamentación de la elección del patrón en términos de: ▪ Los cambios que este admite y los cambios probables anticipados en el diseño concreto. ▪ Las necesidades funcionales de alguna parte del sistema. ▪ Las restricciones de diseño que se deseen imponer.	
where	*Visitante*	**is**
	visitanteConcretoElementoA(ElementoConcretoA)	**is**
	visitanteConcretoElementoB(ElementoConcretoB)	**is**
	VisitanteConcreto1	**is**
	VisitanteConcreto2	**is**
	EstructuraDeObjetos	**is**
	Elemento	**is**
	aceptar(Visitante)	**is**
	ElementoConcretoA	**is**
	operacionA()	**is**
	ElementoConcretoB	**is**
	operacionB()	**is**
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño	