Biblia de IS Dos

(Más IS, menos presupuesto)

PATRONES GOF.

Patrones de diseño				
		Propósito		
		De Creación	Estructurales	De Comportamiento
Ámbito	Clase	• Factory Method	• Adapter (de clases)	• Interpreter • Template Method
	Objeto	• Abstract Factory • Builder • Prototype • Singleton	• Adapter (de objetos) • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

PATRONES DE CREACIÓN:

Los patrones de creación son diferentes según sean de clase o de objeto:

De clase: utilizan la herencia para cambiar la clase de la instancia a crear

De objeto: delegan la creación de las instancias en otros objetos

Factoría Abstracta (de clase):

Su objetivo es crear una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. Una factoría crea objetos devolviendo sus interfaces.
Cuándo usar:

- Un sistema debe ser independiente de cómo se crean, componen y representan sus productos
- Quiere proporcionar una biblioteca de clases de productos, y sólo quiere revelar sus interfaces, no sus implementaciones

Ventajas:

-Aísla las clases concretas de sus clientes

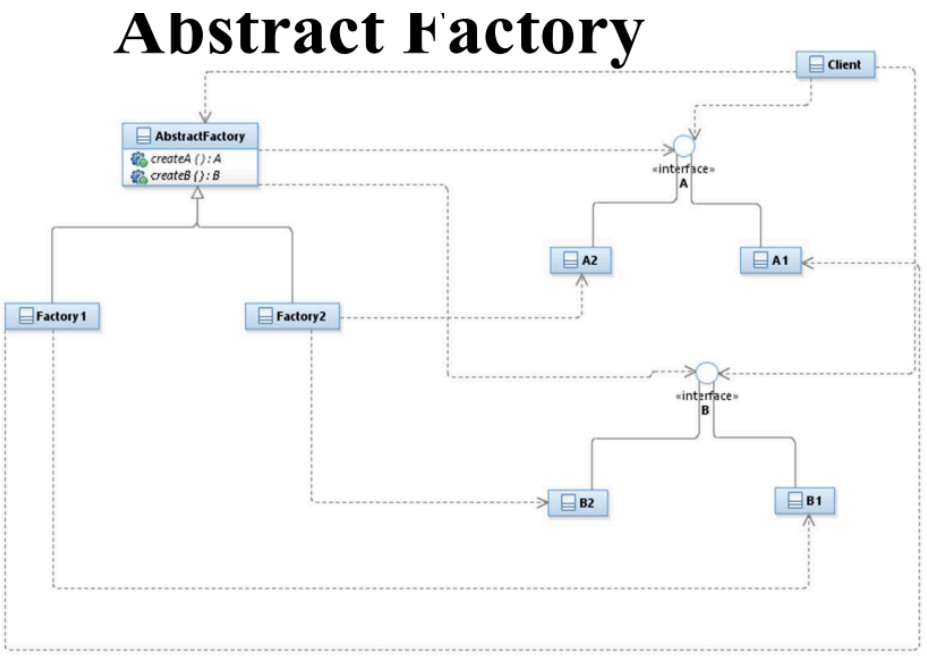
-Facilita el intercambio de familias de productos

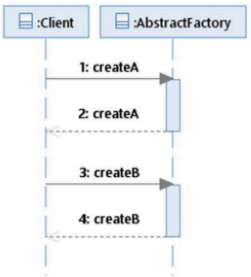
-Promueve la consistencia entre productos

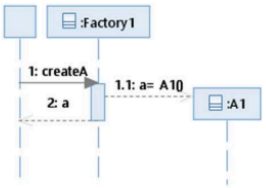
Desventajas:

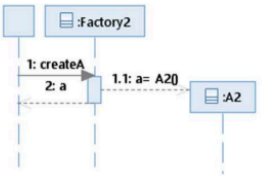
- Es difícil dar cabida a nuevos productos, porque tienes que modificar factoriaAbstracta

UML:









Singleton: (de objeto)

Permite restringir la creación de clases a un único objeto; es decir, garantiza que una clase sólo tenga una instancia (o un número controlado de ellas) y proporciona un punto de acceso global a ellas.

Cuándo usar:

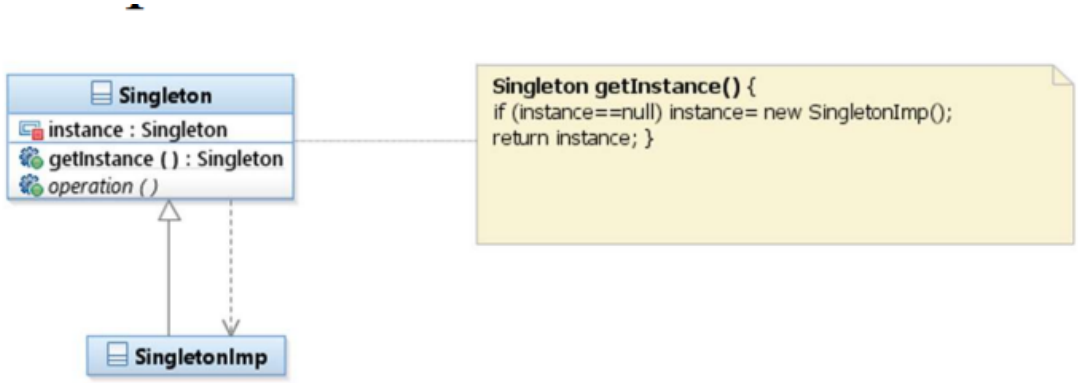
- Una clase solo deba tener una instancia disponible para todos los clientes
- La instancia debería ser extensible por herencia, y los clientes deberían usar una instancia extendida sin modificar su código

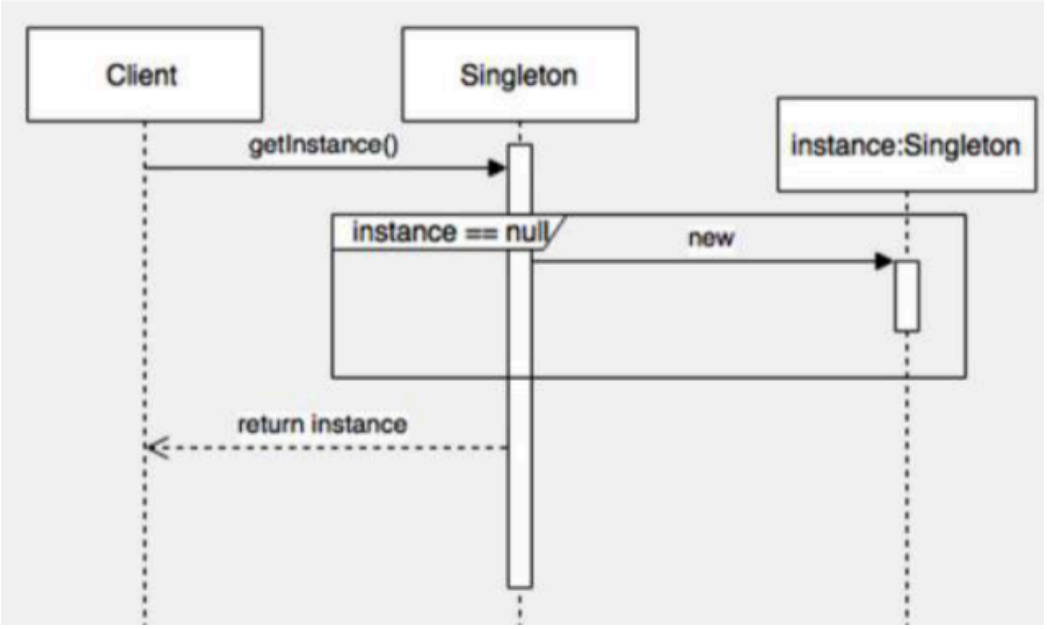
Ventajas:

- Acceso controlado a una instancia (o número controlado)
- Permite número variable de instancias
- Más flexibles que las operaciones estáticas
- Espacio de nombres reducido

Desventajas: es la puta ostia este patrón, ni una mísera desventaja para Antonio.

UML:





Un ejemplo del Singleton es el Controller del proyecto.

PATRONES ESTRUCTURALES:

Los patrones estructurales se preocupan de cómo se combinan las clases y los objetos para formar estructuras más grandes.

- Los de clases hacen uso de la herencia para componer interfaces o implementaciones

- Los de objetos describen formas de componer objetos para obtener nueva funcionalidad

Adapter (de clase o de objeto):

Convierte la interfaz de una clase en otra interfaz que es la que esperan los clientes, y permite que cooperen clases con interfaces incompatibles.

Cuándo usar:

- Se quiera usar una interfaz existente y no concuerda con la que se necesita
- Se quiere utilizar una clase reutilizable que coopere con clases no relacionadas o que no han sido previstas

Ventajas (**adaptador de clase**):
- Permite que el adaptador redefina el comportamiento del adaptable al ser subclase suya
- Introduce un solo objeto y no se necesita ningún puntero adicional para obtener el objeto adaptado

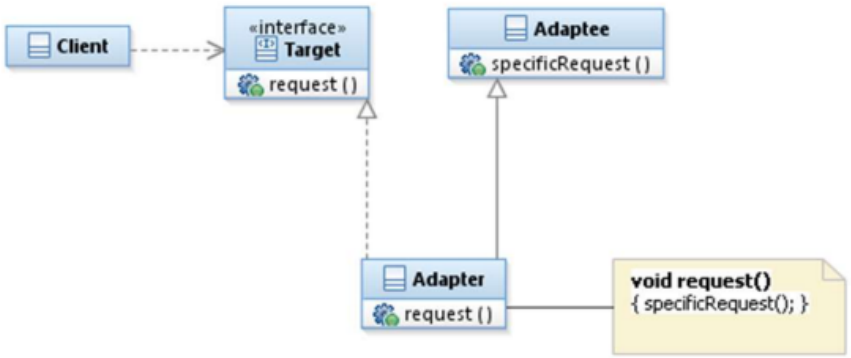
Desventajas (**adaptador de clase**):
- La adaptación es de una clase, pero no de sus subclases

Ventajas (**adaptador de objeto**):
- Un adaptador puede funcionar con muchos adaptables (sus subclases), y añadirles funcionalidad.

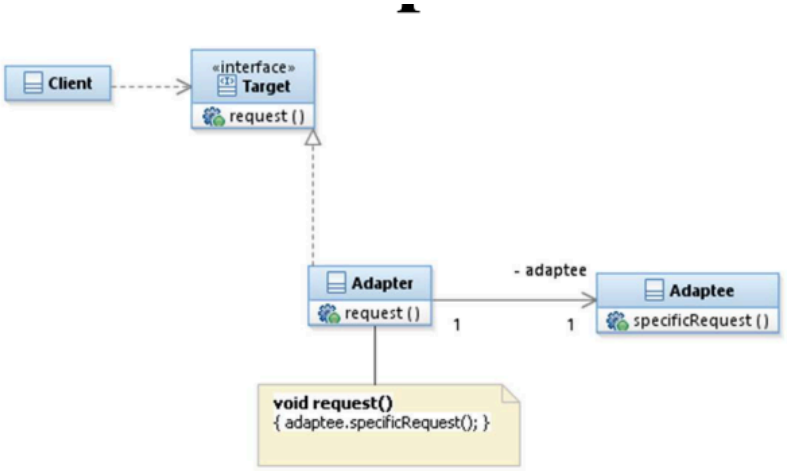
Desventajas (**adaptador de objeto**):
- Tienes que vincular el adaptador a las clases que pudieran aparecer del adaptable

UML:

De clases:



De objetos:



Código de ejemplo:

```
public interface IED {
public int insertar(Comparable objetoP);
public int eliminar(Object idObjeto)};

public class DynamicList {
public int insert(Comparable objectP) {...};
public int delete(Object idObject) {...};
 };

public class ListaDinamica implements IED {
     DynamicList lista;
     ListaDinamica () { lista= new DynamicList(); }
     public int insertar(Object o) {
           return lista.insert(o);
          }
.........
};
```

Composite:

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

Cuándo usar:

- Se quiera representar jerarquías parte-todo
- Se quiera que a los clientes obvien las diferencias entre composiciones de objetos y objetos individuales. Y los tratarán de manera uniforme

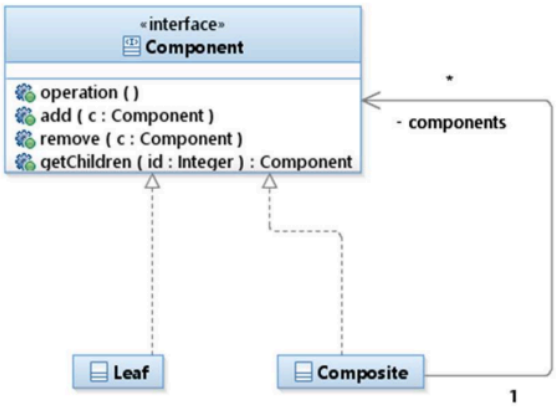
Ventajas:

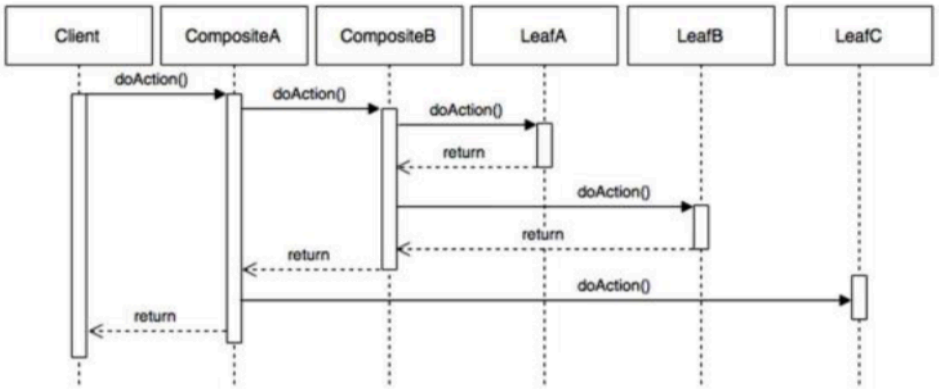
- Define jerarquías de clases formadas por objetos primitivos y compuestos
- Tratamiento uniforme por parte del cliente
- Fácil añadir nuevos tipos de componentes

Desventajas:

- Oculta el tipo del objeto dentro del compuesto

UML:





1.El cliente realizar una acción sobre el CompositeA.
2.CompositeA a su vez realiza una acción sobre CompositaB.
3.CompositeB realiza una acción sobre LeafA y LeafB y el resultado es devuelto a CompositeA.
4.CompositeA propaga la acción sobre LeafC, el cual le regresa un resultado.
5.CompositeA obtiene un resultado final tras la evaluación de toda la estructura y el cliente obtiene un resultado.

Decorator:

Asigna responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la herencia para extender la funcionalidad

Usar cuando:

- Se quiera añadir objetos individuales sin afectar a otros objetos de forma dinámica
- Se desee la posibilidad de eliminar responsabilidades

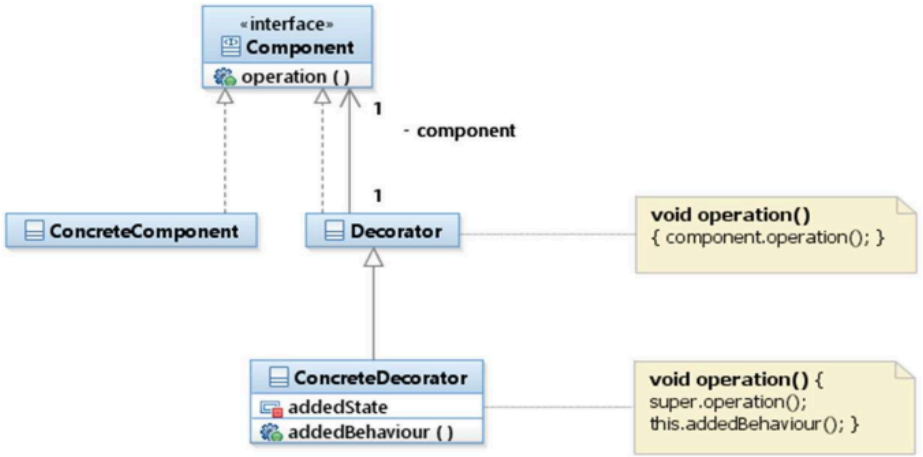
Ventajas:

- Más flexibilidad que la herencia estática
- Evita clases cargadas de funciones en la parte de arriba de la jerarquía, ya que se pueden añadir al decorator

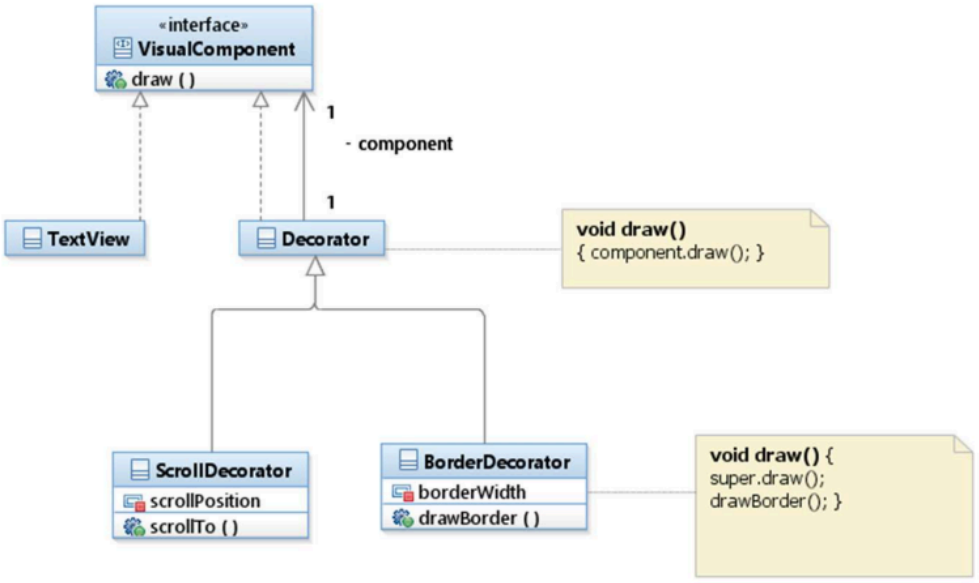
Desventajas:

- Aparecen muchos objetos pequeños
- Decorador y su componente no son idénticos

UML:



Estructura y comportamiento del patrón Decorator



Ejemplo patrón Decorator

Façade:

Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Hace que sea más fácil de usar.

Cuándo usar:

- Queremos poner una interfaz simple a un sistema complejo
- Queremos dividir en capas nuestros subsistemas

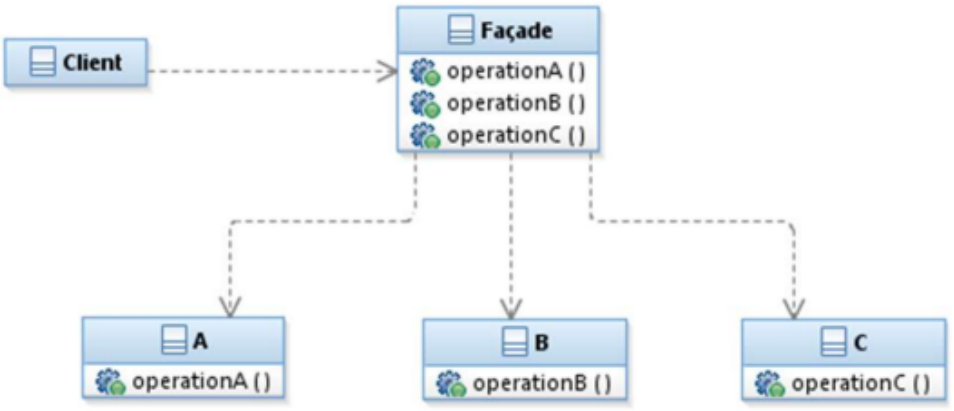
Ventajas:

- El subsistema es más fácil de usar, pues se oculta los componentes del mismo
- Débil acoplamiento entre subsistema y clientes

Desventajas:

- Nuevas operaciones de los componentes deben promocionar hacia la interfaz de la fachada

UML:



Ejemplo de uso:

```
public interface Biblioteca {
     public Integer insertaUsuario(TUsuario usuario);
     public Boolean daDeBajaUsuario(Integer id);
     public TDAOUsuario obtenUsuario(Integer id);
 }

public class BibliotecaImp implements Biblioteca {
     public Integer insertaUsuario(TUsuario usuario) { …… }
     public Boolean daDeBajaUsuario(Integer id) {
     //nótese que no se ha hecho explícito el acceso a estos
     //servicios por parte de la  Biblioteca

     serviciosUsuario.daDeBajaUsuario(id); }
 ……
}
```

PATRONES DE COMPORTAMIENTO:

Los patrones de comportamiento tienen que ver con algoritmos y con la asignación de responsabilidades a objetos

-De clase: usan la herencia para distribuir el comportamiento entre clases

-De objeto: usan la composición para distribuir dicho comportamiento

Command: (de objeto)

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

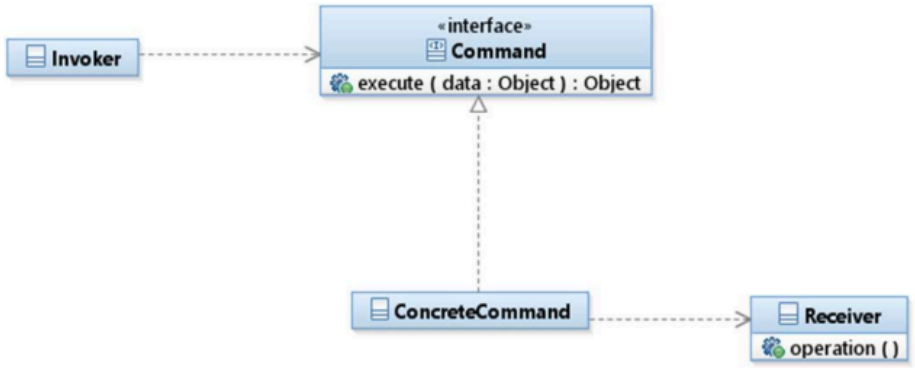
Cuándo usar:

- Parametrizar objetos con una acción a realizar
- Especificar, poner en cola y ejecutar peticiones en diferentes instantes de tiempo
- Permitir registrar los cambios de manera que se puedan volver a aplicar en caso de una caída del sistema

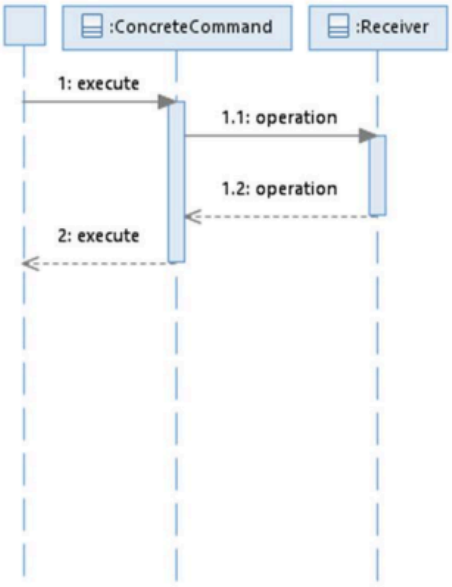
Ventajas:

- Es fácil añadir nuevas órdenes, ya que no hay que cambiar las clases existentes
- Las órdenes son objetos de primera clase
- Desacopla el objeto que invoca la operación de aquél que sabe cómo realizarla

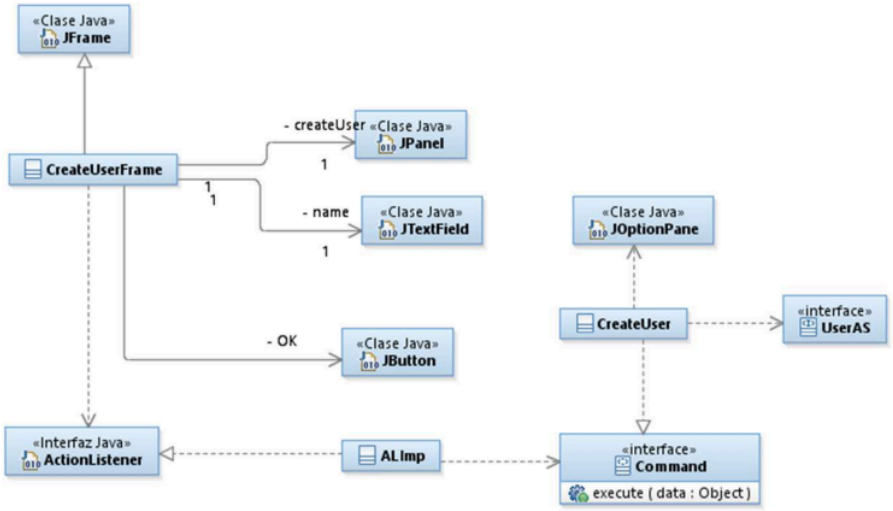
UML:



Estructura del patrón Command



Comportamiento patrón Command



Ejemplo estructura patrón Command

Iterator: (de objeto)

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

Cuándo usar:

- Acceder al contenido de un objeto agregado sin exponer su representación interna
- Permitir recorridos sobre objetos agregados
- Proporcionar una interfaz uniforme para recorrer diferentes estructuras agregadas

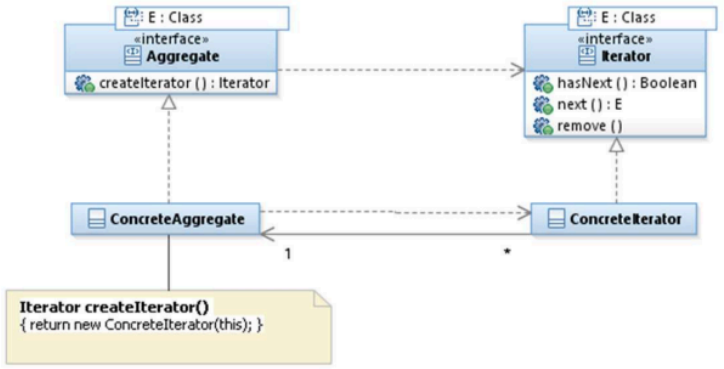
Ventajas:

- Permite variaciones en el recorrido de un agregado
- Simplifica la interfaz del objeto agregado
- Se puede hacer más de un recorrido a la vez sobre el agregado

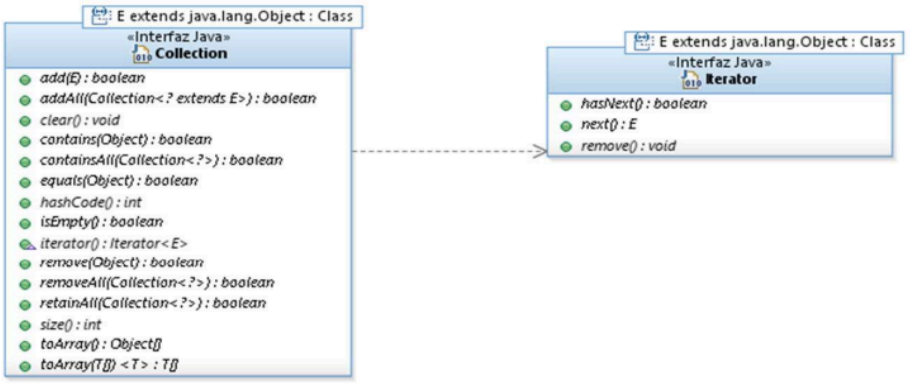
Desventajas:

- Modificaciones en el agregado pueden dejar al iterador inconsistente

UML:



Estructura y comportamiento del patrón Iterator



Ejemplo patrón Iterator

Strategy (de objeto):

Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que los usan

Cuándo usar:

- Muchas clases relacionadas difieren sólo en su comportamiento
- Se necesitas distintas variantes de un algoritmo
- Un algoritmo usa datos que un cliente no debería conocer

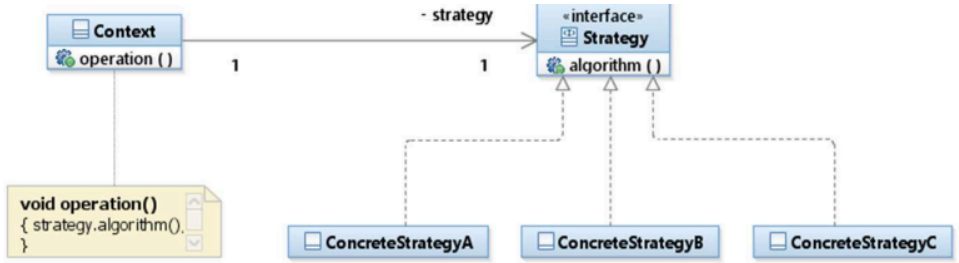
Ventajas:

- Permite familias de algoritmos relacionados
- Alternativa a la herencia
- Las estrategias eliminan las sentencias condicionales

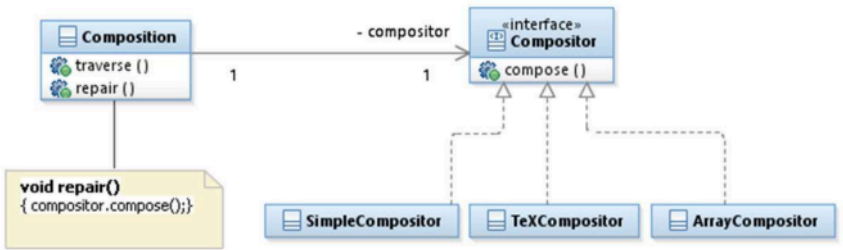
Desventajas:

- Los clientes deben conocer las diferentes estrategias
- Mayor número de objetos

UML:



Estructura y comportamiento del patrón Strategy



Ejemplo patrón Strategy