



Patrones de Diseño:
Ejercicios.
Patrones Estructurales

Ejercicio 1

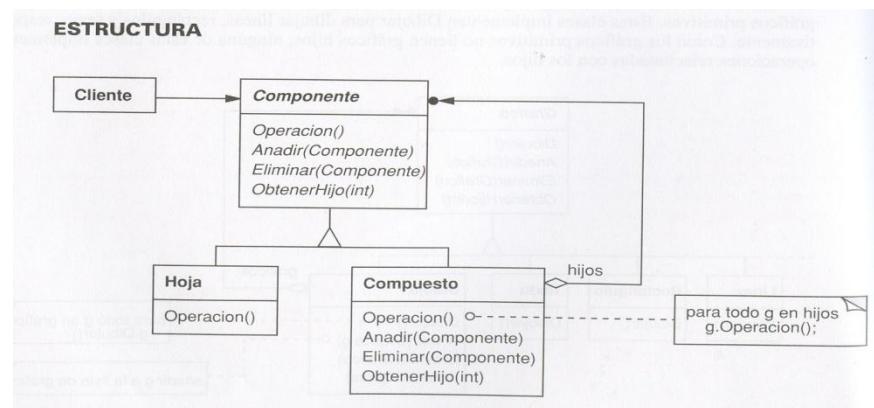
- Se desea crear una aplicación para el manejo de TPV (Terminal de Punto de Venta) en diferentes supermercados. Se quiere diseñar la aplicación de tal forma que contemple la aplicación de descuentos al total de una venta. Habrá diferentes políticas de descuentos como:
 - "los miércoles un 5% al total de una venta"
 - "un 10% en cada compra superior a 150€"
 - "descuentos para determinados tipos de clientes", etc.

La aplicación debe permitir la acumulación de distintos descuentos para una misma venta. ¿Cómo podrías diseñar la aplicación utilizando patrones de diseño de forma que la incorporación de nuevos descuentos fuera lo más sencilla posible? ¿Cómo lo implementarías?



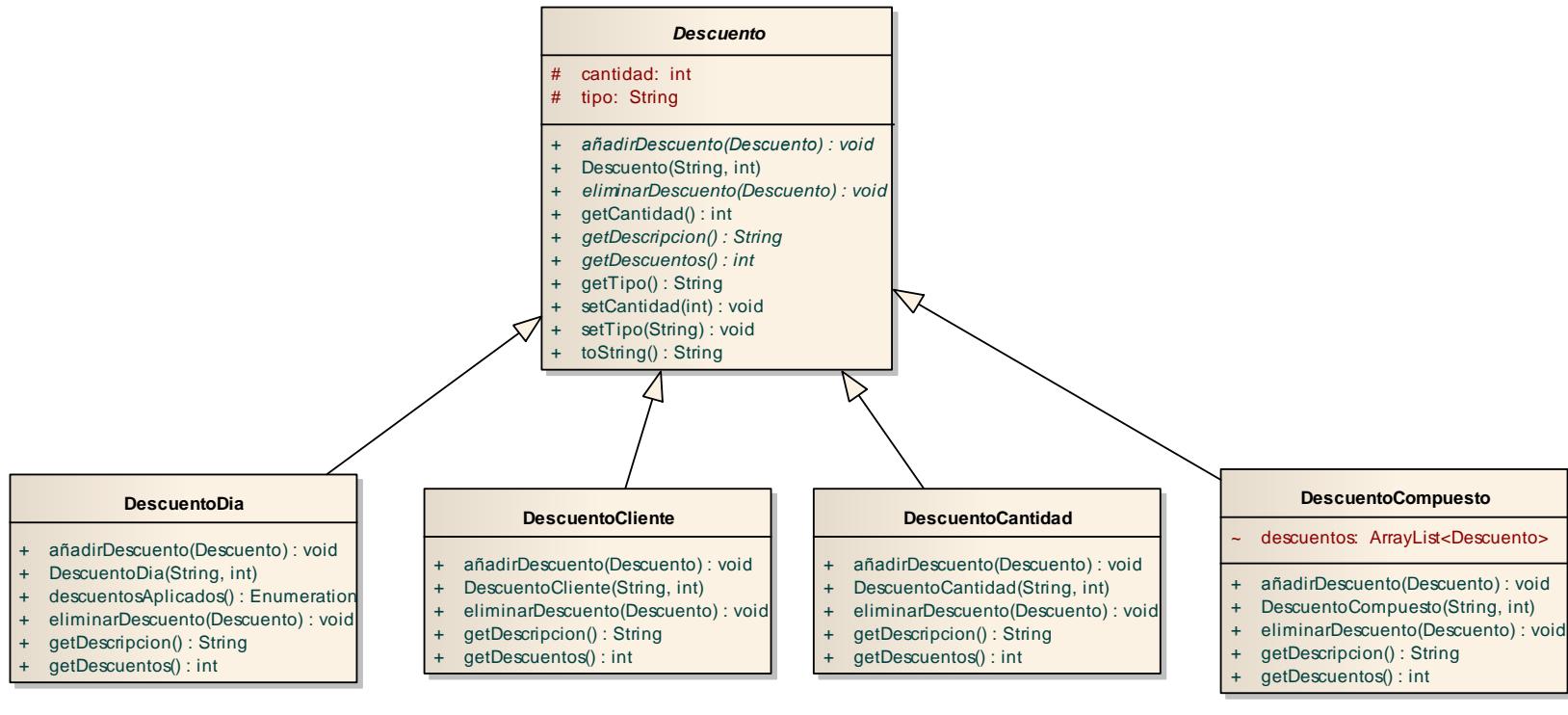
Ejercicio 1. Solución

- El patrón que mejor se adapta a este ejercicio es el Composite.
 - *Construir objetos de complejidad mayor mediante otros más sencillos de forma recursiva, formando una jerarquía en estructura de árbol. Permite que todos los elementos tanto individuales como compuestos se traten de una manera uniforme por los clientes.*



Ejercicio 1. Solución

class Class Model



Ejercicio 2

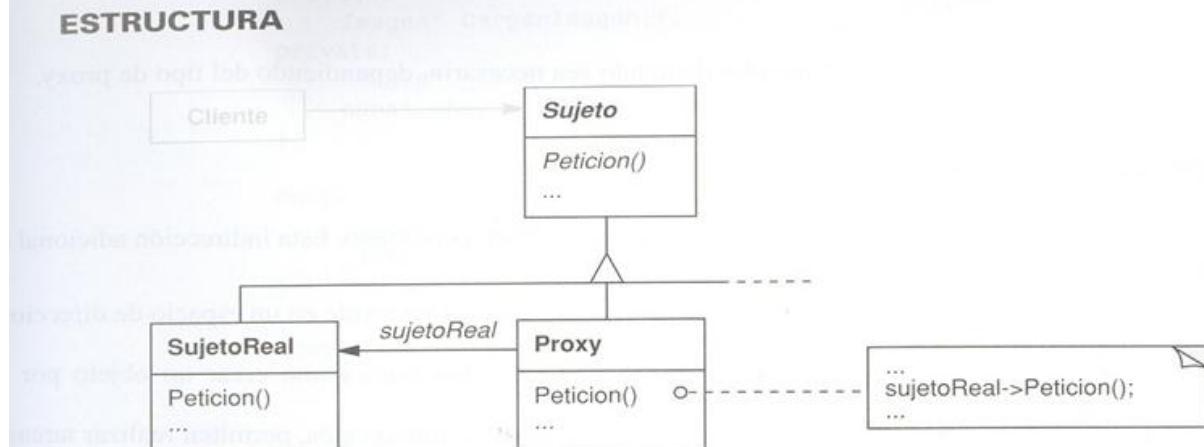
- Se desea realizar una aplicación de consulta de datos de los empleados de una empresa. De cada empleado se almacenan: identificador, contraseña, nombre, departamento y sueldo. Un empleado puede consultar tanto sus propios datos como los de sus compañeros. Como se puede suponer algunos de estos datos son confidenciales y solo los puede consultar el propio empleado.

Por lo tanto, si un empleado consulta sus datos se le mostrarán: identificador, nombre, departamento y sueldo; pero si es otro empleado solo se mostrarán: identificador, nombre y departamento. ¿Cómo podría proporcionarse una solución a este supuesto utilizando patrones de diseño? ¿Cómo lo implementarías?

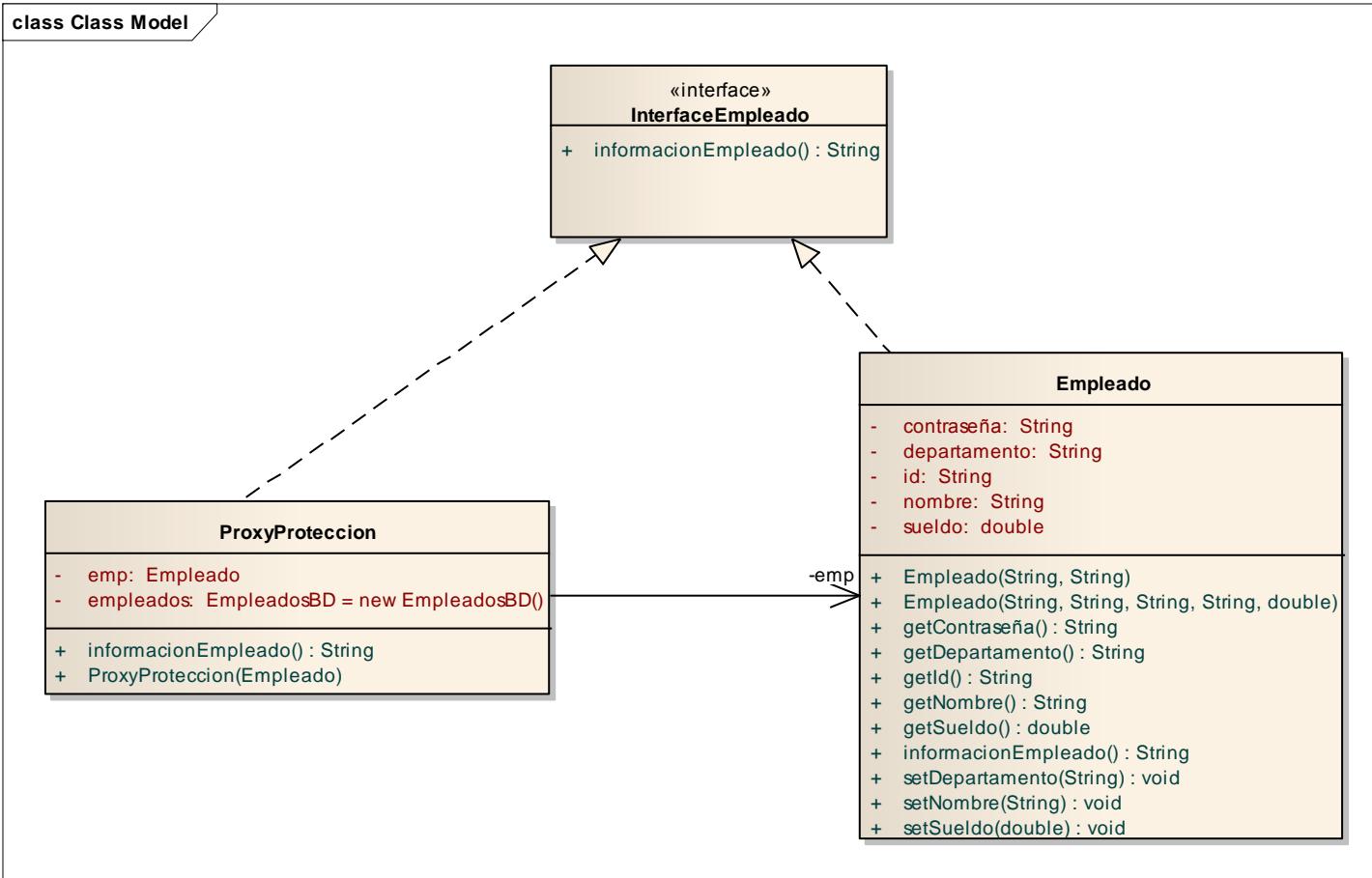


Ejercicio 2. Solución

- El patrón que mejor se adapta a este ejercicio es el Proxy.
 - *Proporciona un representante o sustituto de otro objeto para controlar el acceso a éste.*
 - Proxy de Protección: *Para controlar los derechos de acceso a un objeto.*



Ejercicio 2. Solución



Ejercicio 3

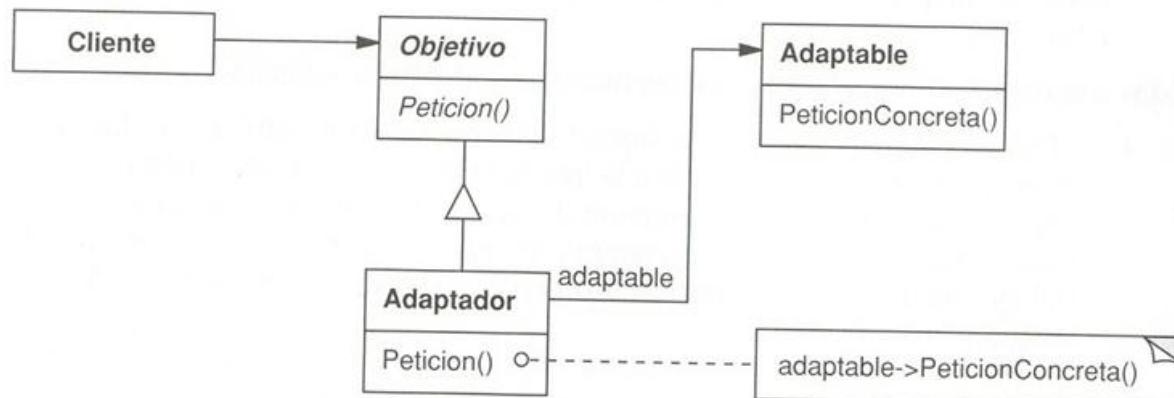
- Se desea realizar una aplicación para consultar los datos de los clientes de una tienda. La tienda tiene desarrollada una aplicación de facturación que utiliza actualmente donde se almacenan en cada factura además de los datos propios de las facturas los datos de sus clientes. Por lo tanto la nueva aplicación debe reutilizar los datos de la aplicación de facturación para obtener los datos de los clientes. Cada factura contiene los siguientes datos:
 - De la factura: Número, concepto, importe de la factura y fecha.
 - Del cliente: CIF, nombre y dirección.

¿Cómo podrías reutilizar la aplicación de facturación para proporcionar los datos de los clientes? ¿Qué patrón utilizarías? ¿Cómo lo implementarías?

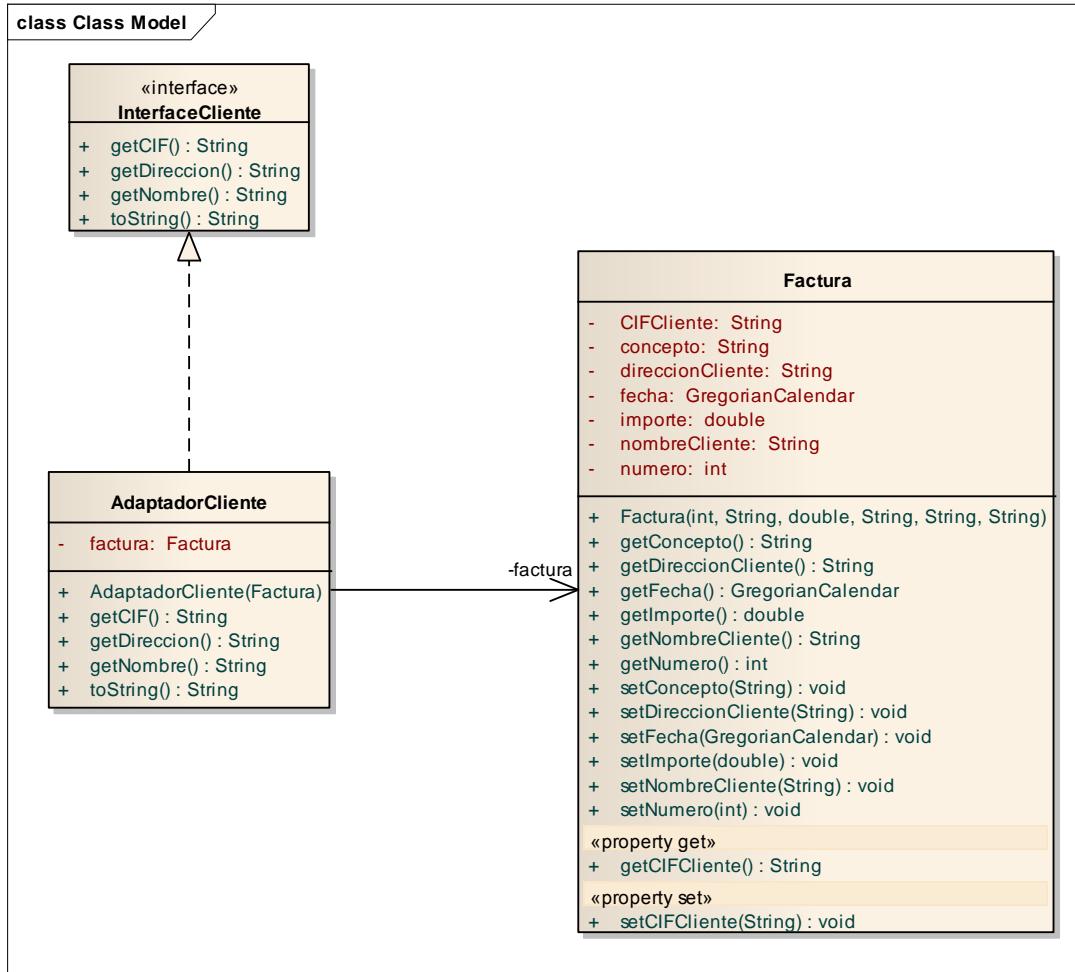


Ejercicio 3. Solución

- El patrón que mejor se adapta a este ejercicio es el Adapter.
 - Sirve de *intermediario entre dos clases, convirtiendo las interfaces de una clase para que pueda ser utilizada por otra. Permite que cooperen clases que tienen interfaces incompatibles.*



Ejercicio 3. Solución



Ejercicio 4

- Sea una clase TextView que representa un componente gráfico de tipo ventana de texto, que es una subclase de una clase Component raíz de la jerarquía de clases que representan componentes gráficos. Se desea definir ventanas de texto con diferentes tipos de bordes (3D, Plain) y barras de desplazamiento (horizontal y vertical). La clase TextView tiene un método dibujar :

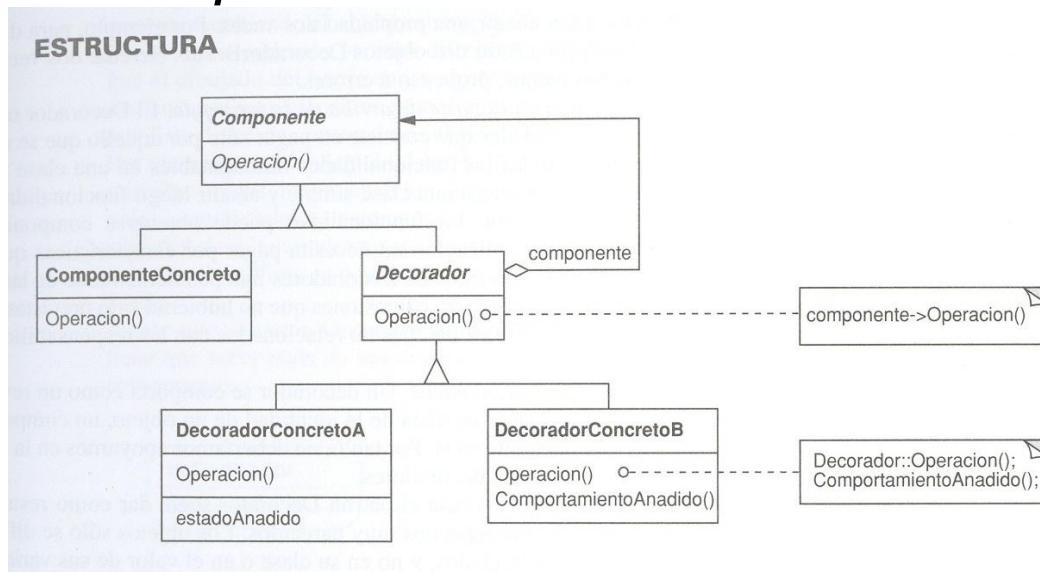
```
class TextView {  
    public void dibujar() { // código para dibujar el objeto TextView }  
}
```

Realizar el diagrama de clases resultado de aplicar el patrón Decorator a la jerarquía de clases del problema.



Ejercicio 4. Solución

- El patrón Decorator.
 - *Añadir nuevas responsabilidades dinámicamente a un objeto sin modificar su apariencia externa o su función, es una alternativa a crear demasiadas subclases por herencia.*



Ejercicio 4. Solución

