Patrones GOF estructurales

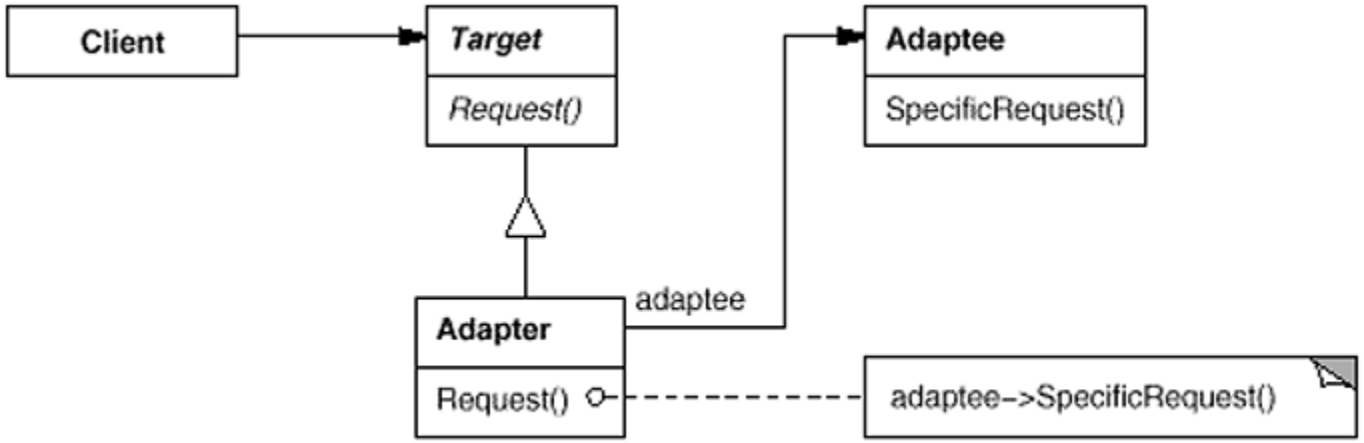
Diseño de Sistemas Software

Patrones GOF estructurales

- Los patrones estructurales tratan con cómo se componen las clases y los objetos para formar estructuras más complejas.
  - Los **patrones estructurales de clases** usan la herencia para componer interfaces o implementaciones.
  - Los **patrones estructurales de objetos** usan la composición para definir nuevos objetos que añaden nuevas funcionalidades.

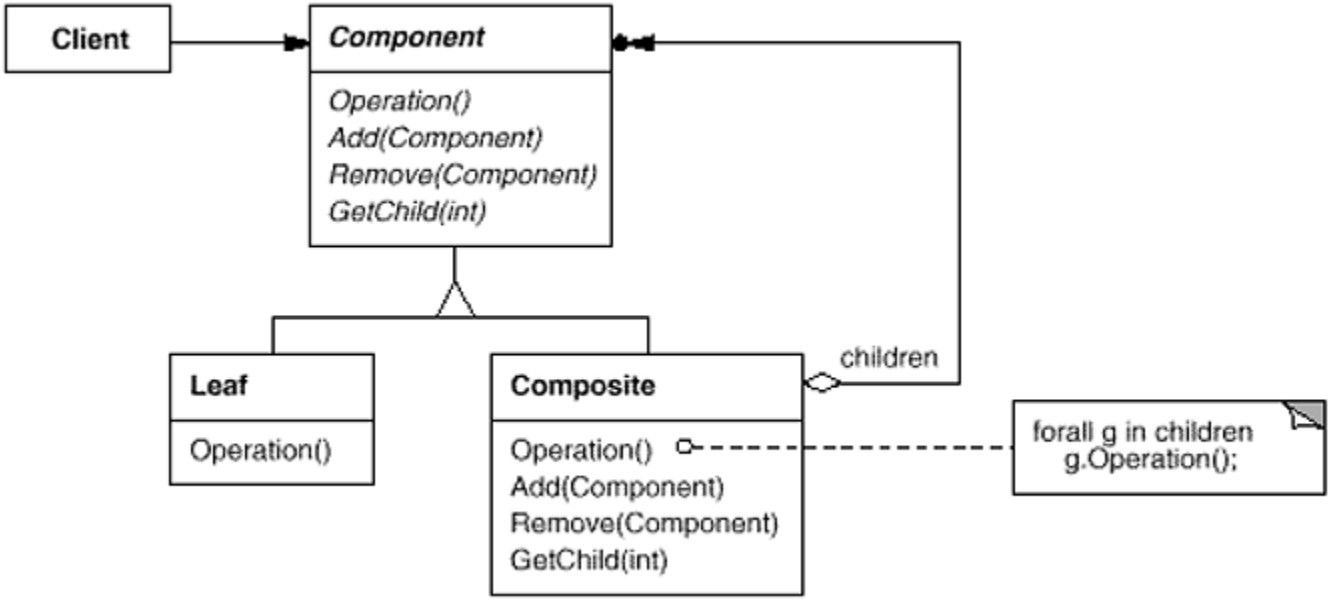
Adapter

- **Adapter**: oficia de intermediario entre dos clases cuyas interfaces son incompatibles de manera tal que puedan ser utilizadas en conjunto.



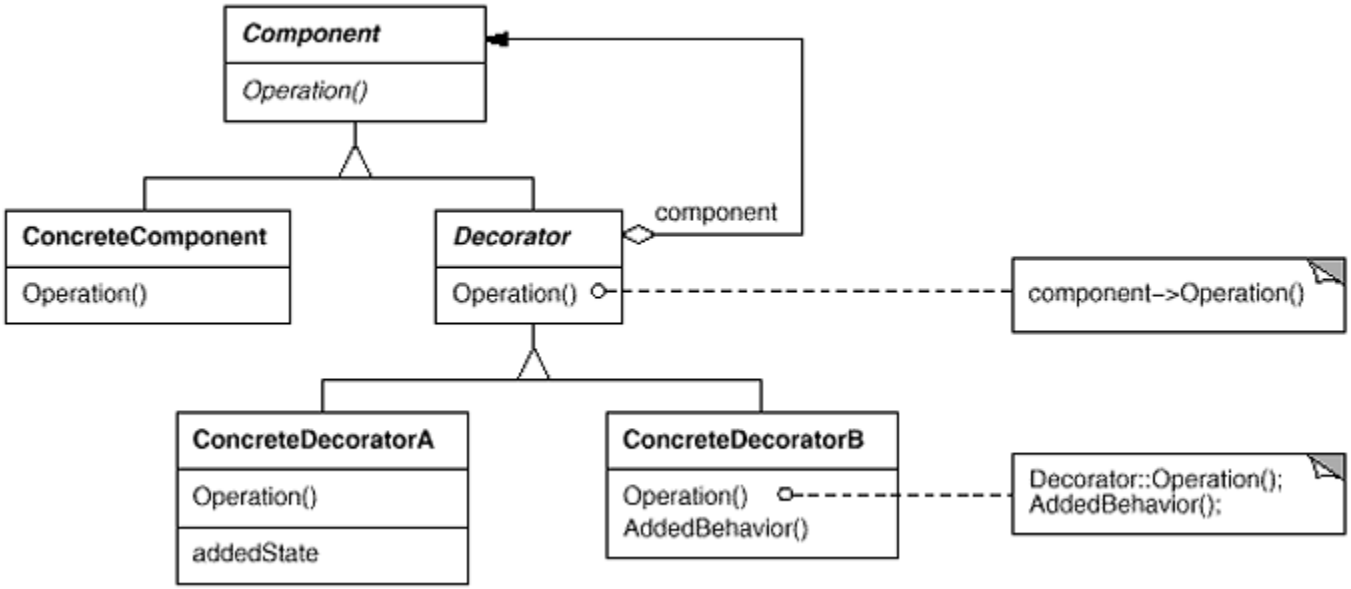
Composite

- **Composite**: compone objetos en estructuras de árboles para representar jerarquías parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los complejos.



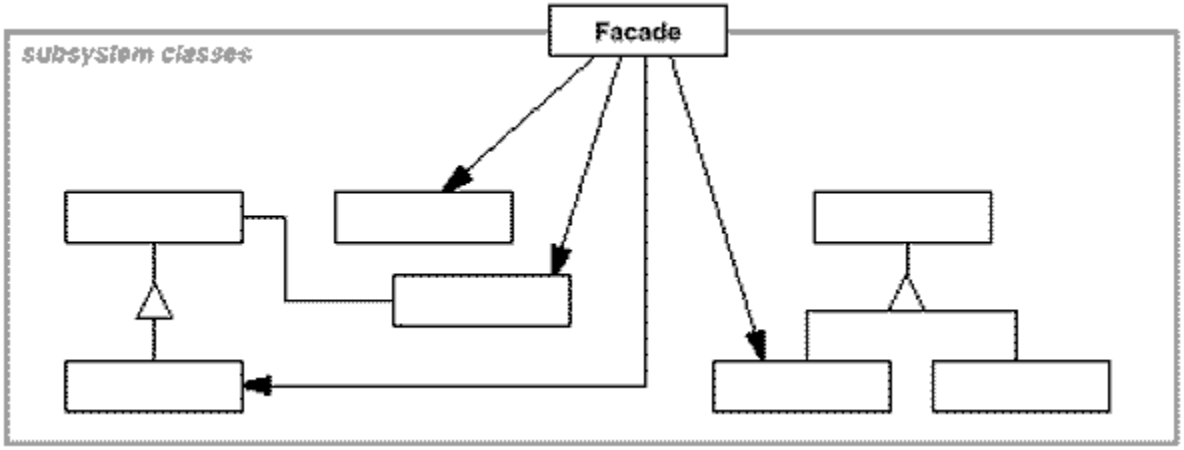
Decorator

- **Decorator**: agrega o limita responsabilidades adicionales a un objeto de forma dinámica, proporcionando una alternativa flexible a la herencia para extender funcionalidad .



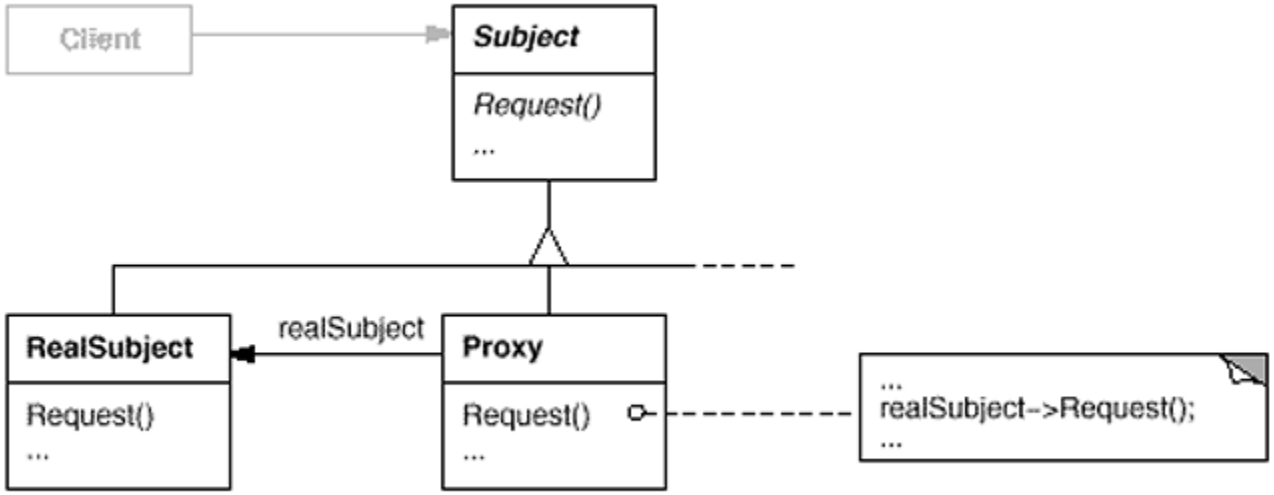
Façade

- **Façade**: proporciona una interfaz simplificada para un conjunto de interfaces de subsistemas. Define una interfaz de alto nivel que hace que un subsistema sea más fácil de usar.



Proxy

- **Proxy**: provee un sustituto o representante de un objeto para controlar el acceso a éste. Este patrón posee las siguientes variantes:
  - Proxy remoto: se encarga de representar un objeto remoto como si estuviese localmente.
  - Proxy virtual: se encarga de crear objetos de gran tamaño bajo demanda.
  - Proxy de protección: se encarga de controlar el acceso al objeto representado.



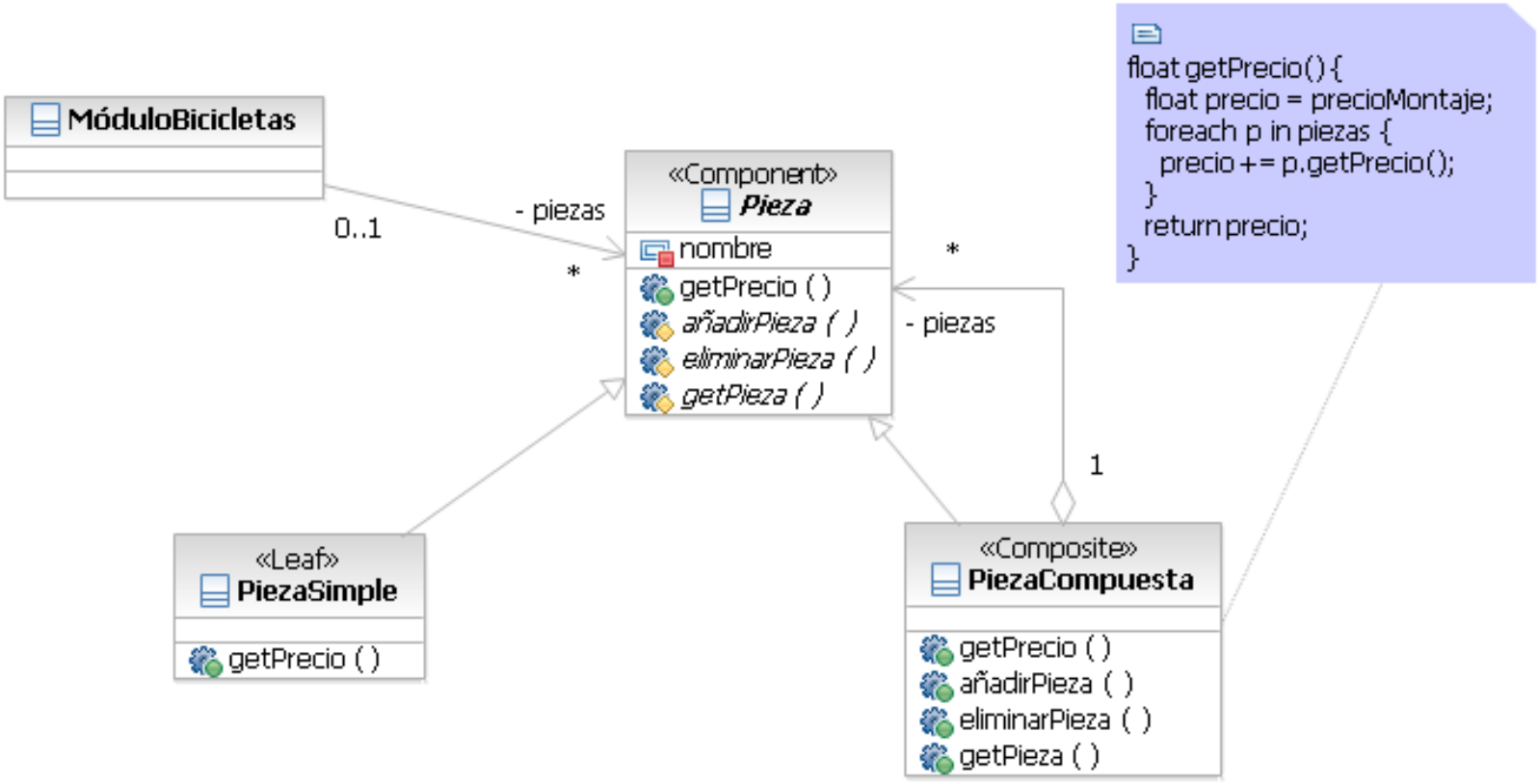
Ejercicios

Ejercicio 1

- **Problema**: Decathlon nos ha pedido que le creemos un módulo para el manejo de la configuración de bicicletas a medida. Para ello, la tienda nos ha proporcionado la siguiente lista de partes, que podría irse especializando con nuevas partes, simples o agregadas: rueda, horquilla, eje, radio, tuercas del radio de las ruedas, cámara, tubo, llanta, armazón, manillar, ..., cada una con su precio. En un principio supondremos que Decathlon trabaja con un solo modelo de cada parte. Decathlon quiere que su software pueda llamar a nuestro módulo sin tener que preocuparse de si está tratando con una bicicleta completa o con partes aisladas. De momento nos ha pedido un prototipo donde la única funcionalidad necesaria es 'getPrecio()'. Ese precio se compone de un precio de mano de obra (solo aplicable si el cliente pide una pieza que tiene que ser montada) y un precio base por cada una de las piezas. Plantea un diseño que resuelva este problema.

Ejercicio 1

- **Solución**: patrón Composite



Ejercicio 2

- **Problema**:Imagine que el profesor Jackson llama al profesor Ernst a media noche porque alguien ha descubierto que hay un problema relacionado con el sistema que están desarrollando para la famosa cadena de tiendas Decathlon: éste debe ser capaz de soportar bicicletas que se puedan volver a pintar (para cambiar su color). Los profesores dividen el trabajo: el profesor Jackson escribirá la clase ColorPalette con un método que, dado un nombre como “rojo ”, “azul” o “ceniza”, devuelva un array con tres valores RGB, y el profesor Devadas escribirá un código que utilice esta clase.
  Los profesores hacen esto, pasan los tests al trabajo realizado, se van de fin de semana, y dejan los archivos .class para que los becarios del proyecto los integren. Estos se dan cuenta de que el profesor Devadas ha escrito un código que depende de:

```
interface ColorPalette{
        // devuelve valores RGB
        int[] getColor(String name);
}
```

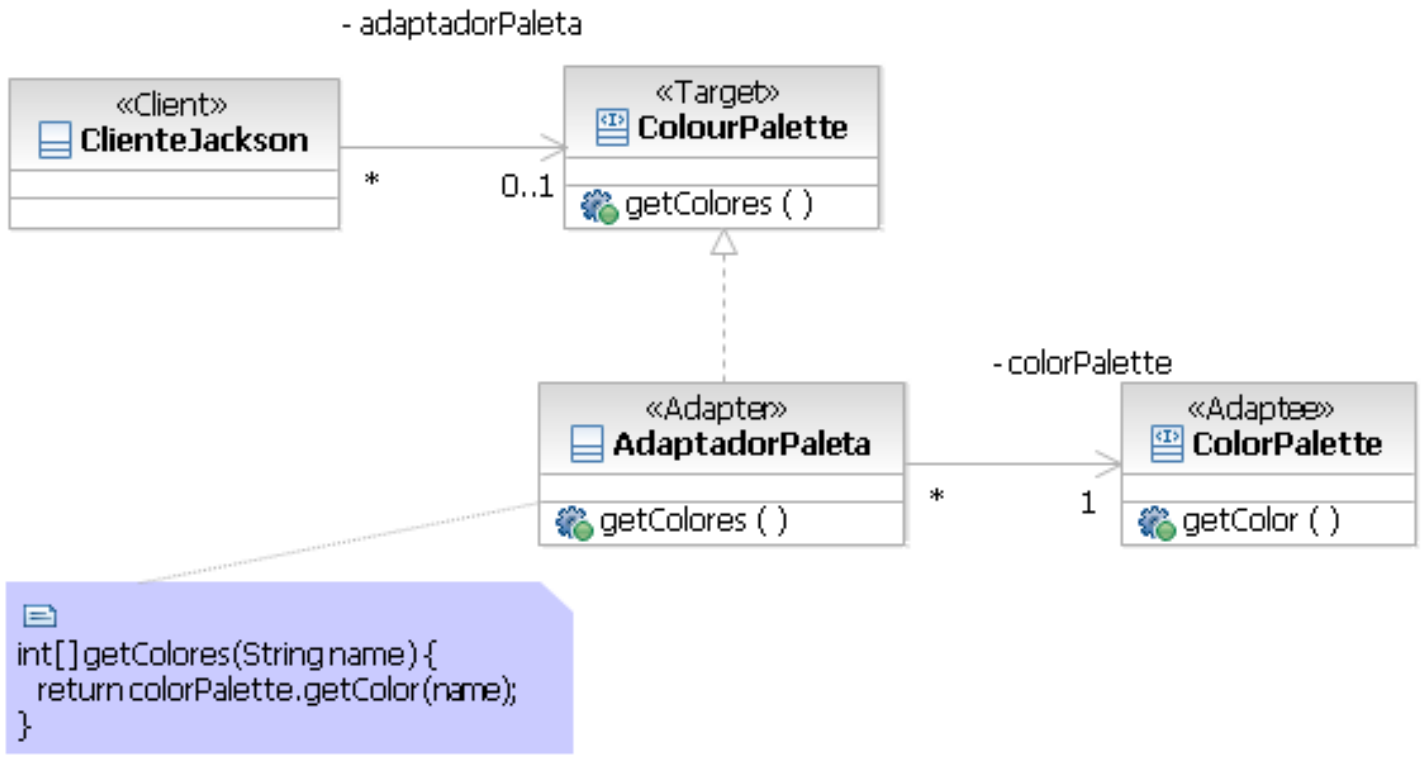
  Sin embargo, el profesor Jackson implementa una clase que se adhiere a:

```
interface ColourPalette{
        // devuelve valores RGB
        int[] getColores(String name);
}
```

  ¿Qué es lo que los becarios tienen que hacer? Ellos no tienen acceso a la fuente, y no disponen de tiempo para volver a implementar y volver a pasar las pruebas.

Ejercicio 2

- **Solución**: patrón Adapter

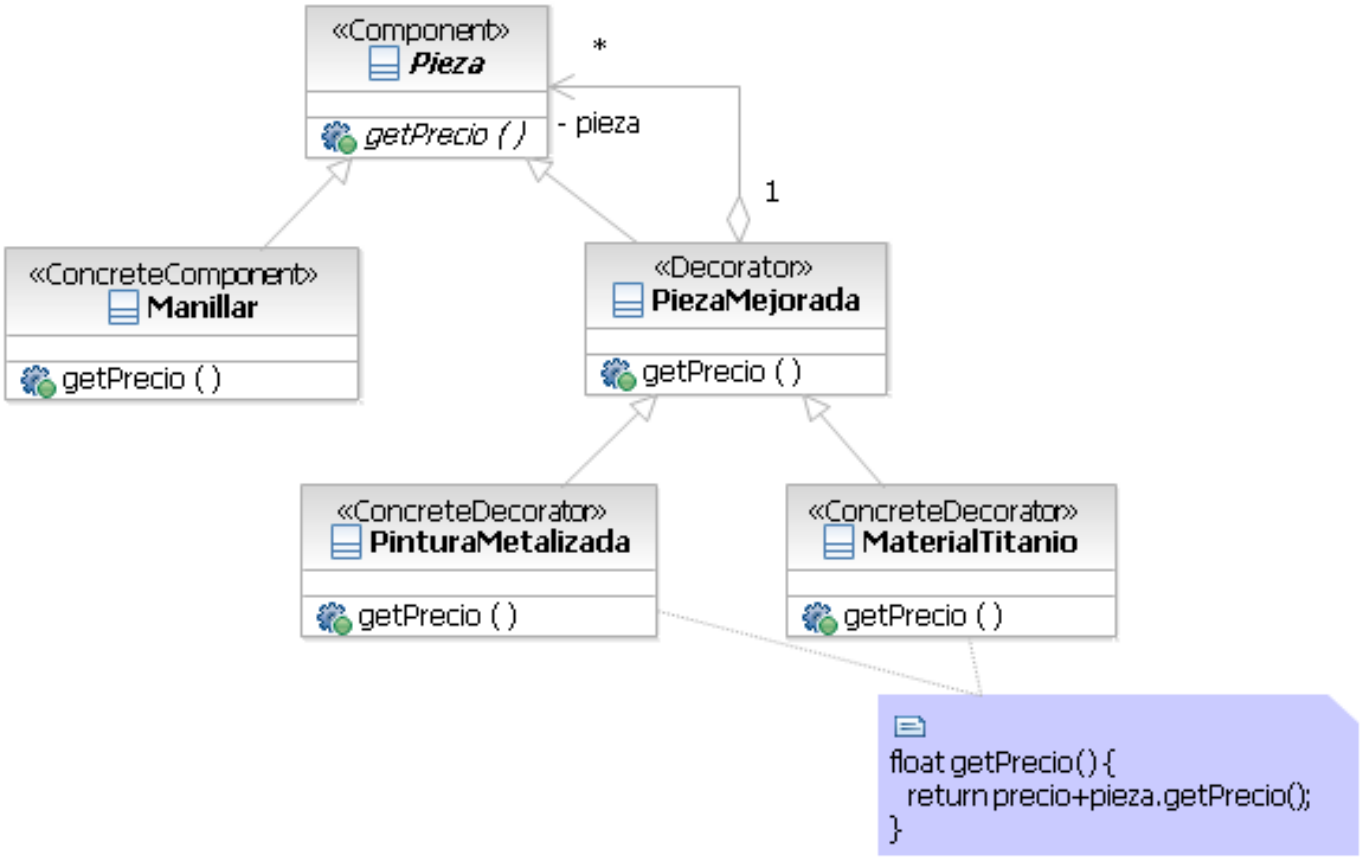


Ejercicio 3

- **Problema**: Muy contentos del trabajo realizado con el manejo de las partes de la bicicleta, Decathlon nos ha pedido un nuevo prototipo. Esta vez quiere que se extienda la funcionalidad de la aplicación para poder trabajar con distintas variaciones de las propiedades de las piezas. E.g. un manillar tiene un precio base, que puede verse incrementado si queremos algún ‘extra’ (e.g. pintura metalizada, materiales de gama superior, etc.). Partiendo del diseño anterior, enriquécelo para que soporte esta nueva variación.

Ejercicio 3

- **Solución**: patrón Decorator

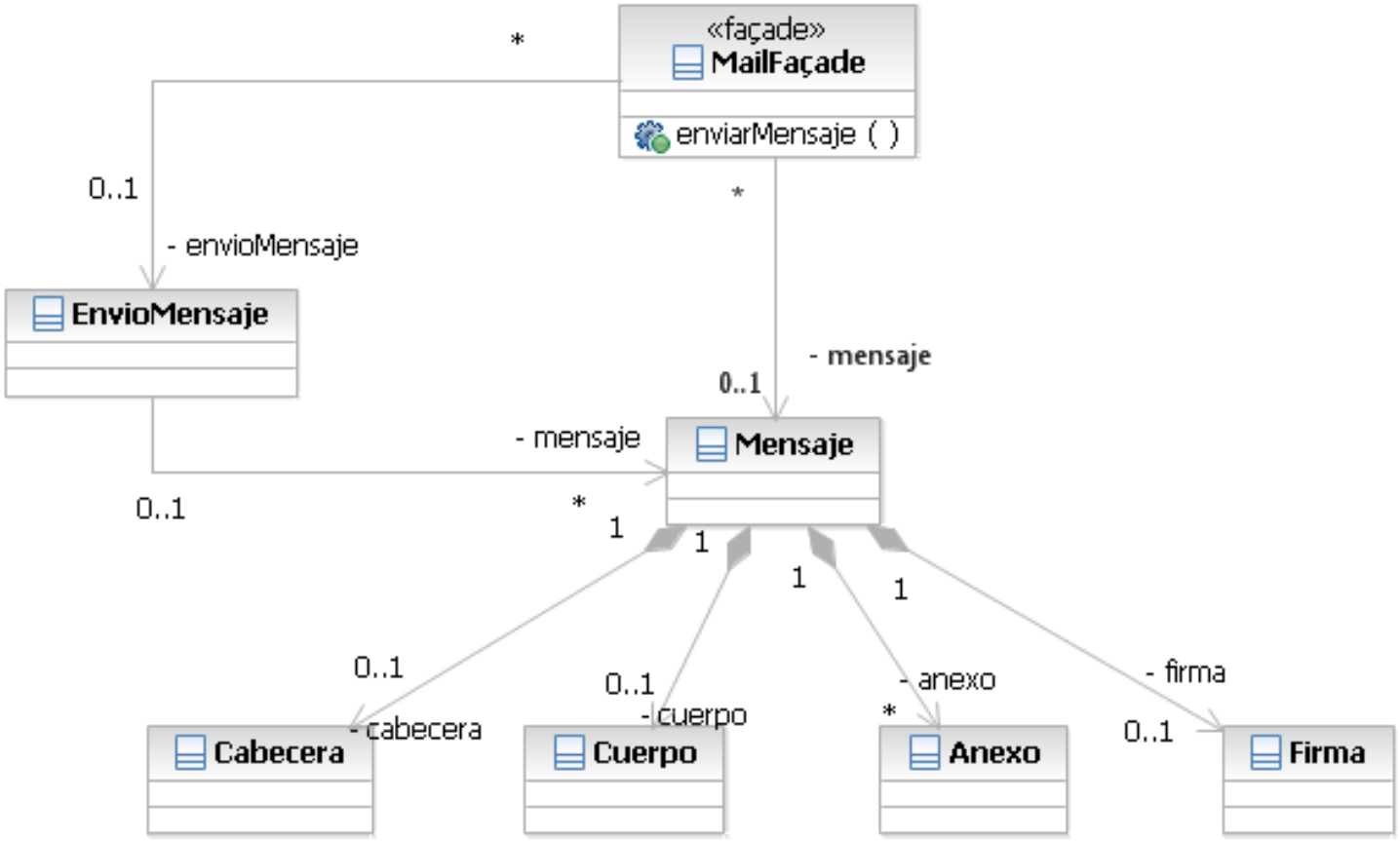


Ejercicio 4

- **Problema**: Sea un conjunto de clases que permiten la creación y envío de mensajes de correo electrónico y que entre otras incluye clases que representan el cuerpo del mensaje, los anexos, la cabecera, el mensaje, la firma digital y una clase encargada de enviar el mensaje. El código cliente debe interactuar con instancias de todas estas clases para el manejo de los mensajes, por lo que debe conocer en qué orden se crean esas instancias; cómo colaboran esas instancias para obtener la funcionalidad deseada y cuáles son las relaciones entre las clases. Idea una solución basada en algún patrón tal que se reduzcan las dependencias del código cliente con esas clases y se reduzca la complejidad de dicho código cliente para crear y enviar mensajes. Dibuja el diagrama de clases que refleje la solución e indica qué patrón has utilizado.

Ejercicio 4

- **Solución**: patrón Façade

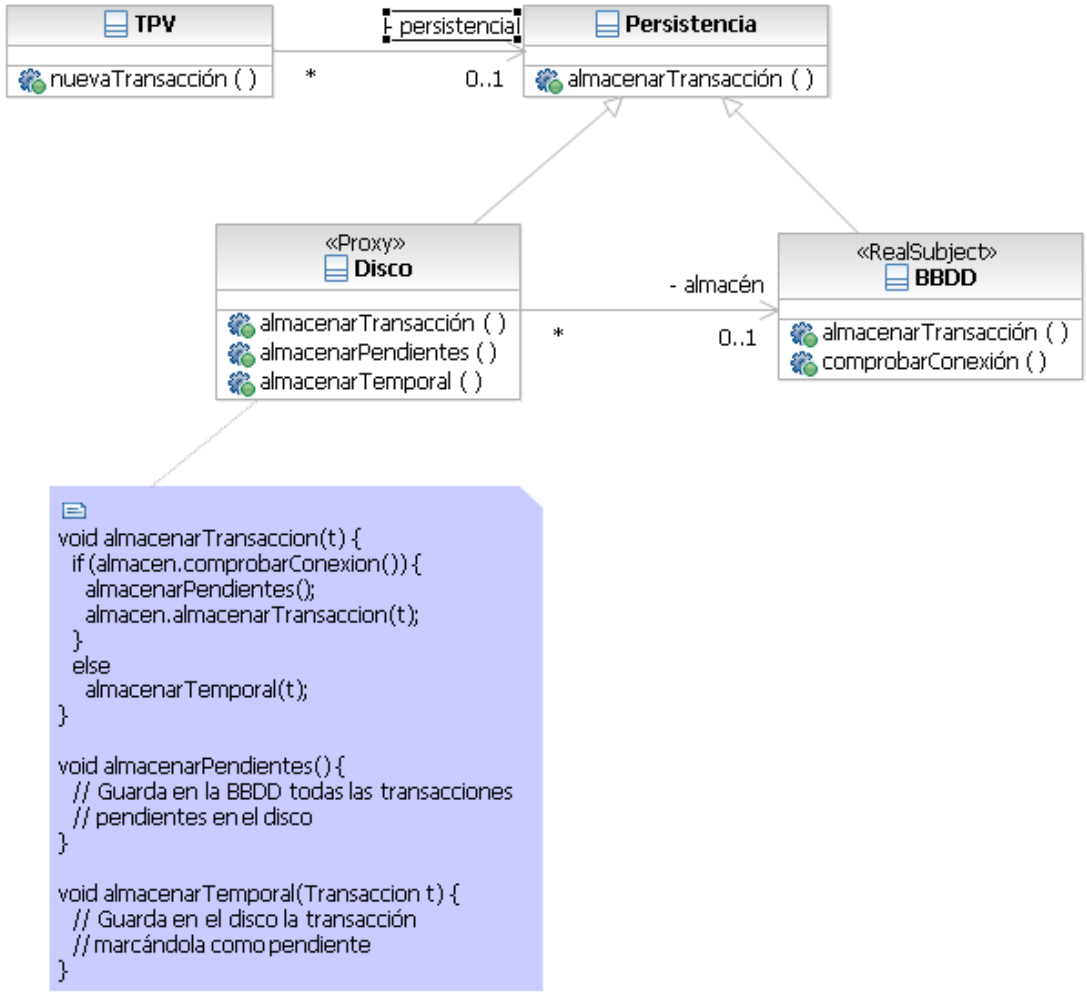


Ejercicio 5

- **Problema**: Se desea almacenar en una base de datos todas las transacciones realizadas por un terminal punto de venta (TPV). El terminal puede trabajar en modo conectado o no conectado, dependiendo de si la conexión a internet está en funcionamiento en el momento de almacenar los datos. Como es muy importante que no se pierda ninguna transacción, en el caso de que se esté trabajando en modo sin conexión, los datos deben almacenarse en el disco hasta que la conexión se restablezca.

Ejercicio 5

- **Solución**: patrón Proxy



Bibliography

- *Design Patterns: Elements of Reusable Object-Oriented Software*. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley Professional, 1994.