Características patrones de diseño.

Estudiantes:

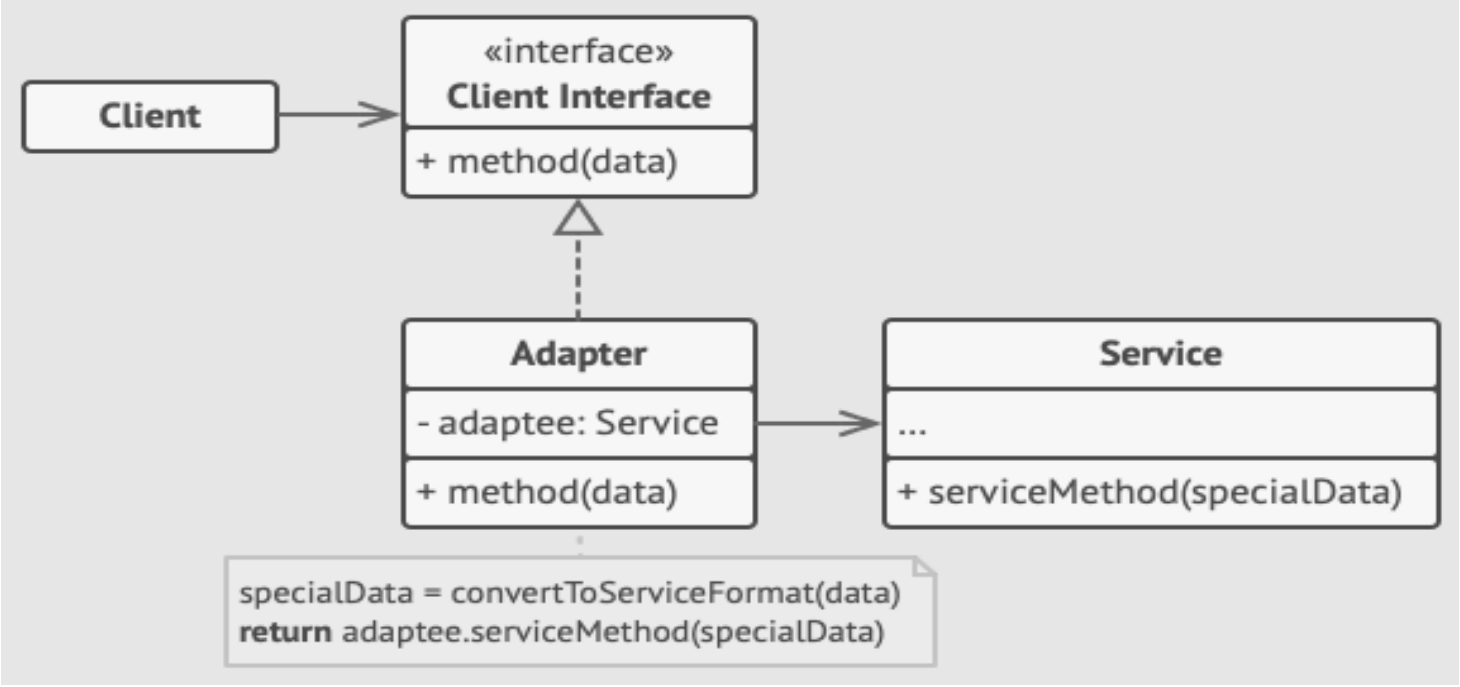
Anna María Sánchez Rojas.

Johand Esteban Castro Rodriguez.

Daniel Felipe Eraso Acero.

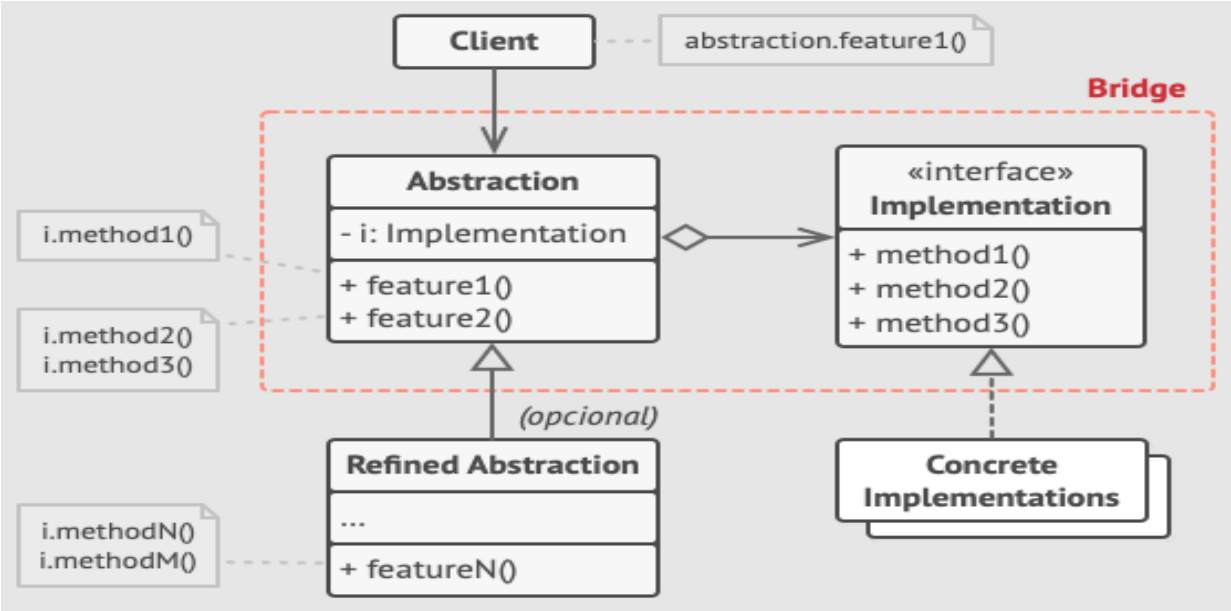
Adapter.

- Es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.
- Utiliza la clase adapter cuando quiera usar una clase existente, pero cuya interfaz no sea compatible con el resto del código.
- El patrón Adapter te permite crear una clase intermedia que sirva como traductora entre tu código y una clase heredada, una clase de un tercero o cualquier otra clase con una interfaz extraña.



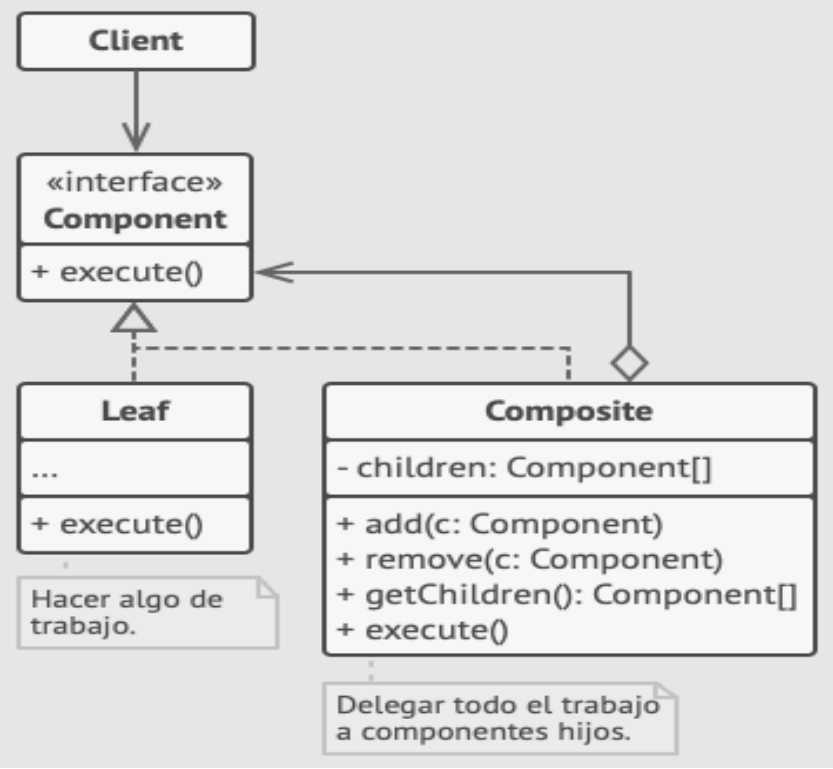
Bridge.

- Es un patrón de diseño estructural que te permite dividir una clase grande, o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra.
- Utiliza el patrón Bridge cuando quieras dividir y organizar una clase monolítica que tenga muchas variantes de una sola funcionalidad (por ejemplo, si la clase puede trabajar con diversos servidores de bases de datos).
- El patrón Bridge te permite dividir la clase monolítica en varias jerarquías de clase. Después, puedes cambiar las clases de cada jerarquía independientemente de las clases de las otras. Esta solución simplifica el mantenimiento del código y minimiza el riesgo de descomponer el código existente.
- Utiliza el patrón cuando necesites extender una clase en varias dimensiones ortogonales (independientes).
- Utiliza el patrón Bridge cuando necesites poder cambiar implementaciones durante el tiempo de ejecución.



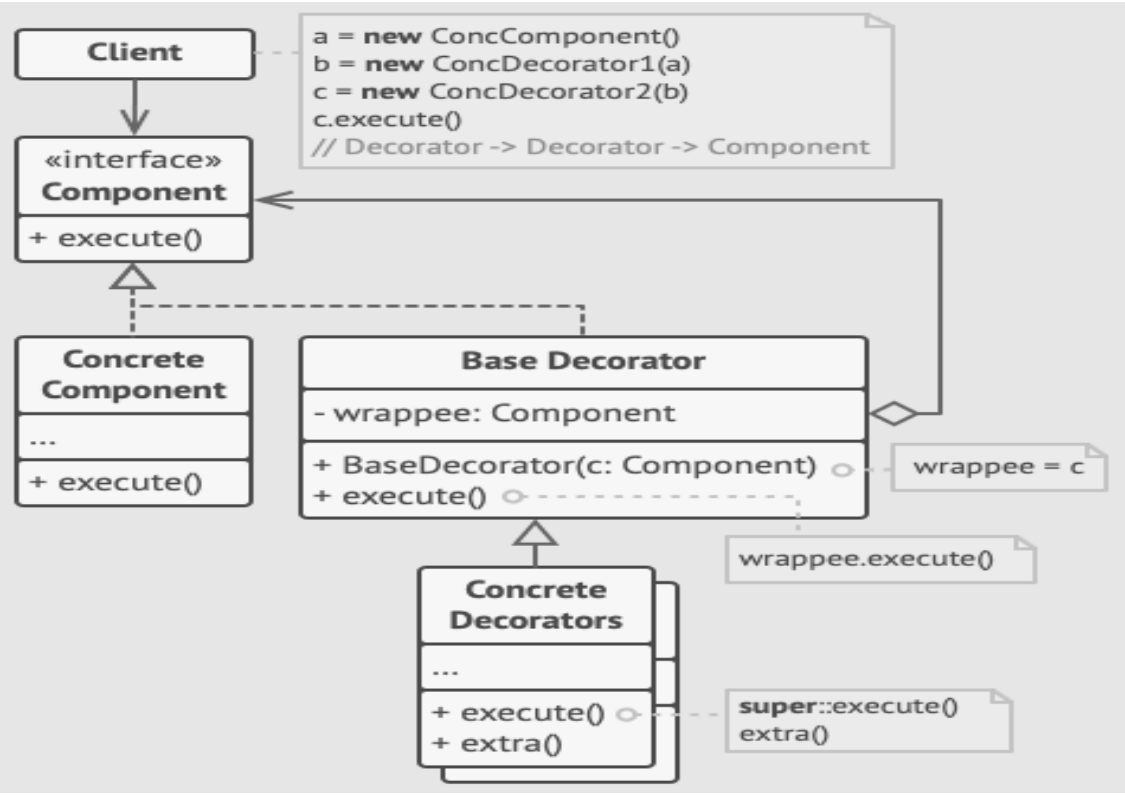
Composite.

- Es un patrón de diseño estructural que te permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.
- Utiliza el patrón Composite cuando tengas que implementar una estructura de objetos con forma de árbol.
- El patrón Composite te proporciona dos tipos de elementos básicos que comparten una interfaz común: hojas simples y contenedores complejos. Un contenedor puede estar compuesto por hojas y por otros contenedores. Esto te permite construir una estructura de objetos recursivos anidados parecida a un árbol.
- Utiliza el patrón cuando quieras que el código cliente trate elementos simples y complejos de la misma forma.
- Todos los elementos definidos por el patrón Composite comparten una interfaz común. Utilizando esta interfaz, el cliente no tiene que preocuparse por la clase concreta de los objetos con los que funciona.



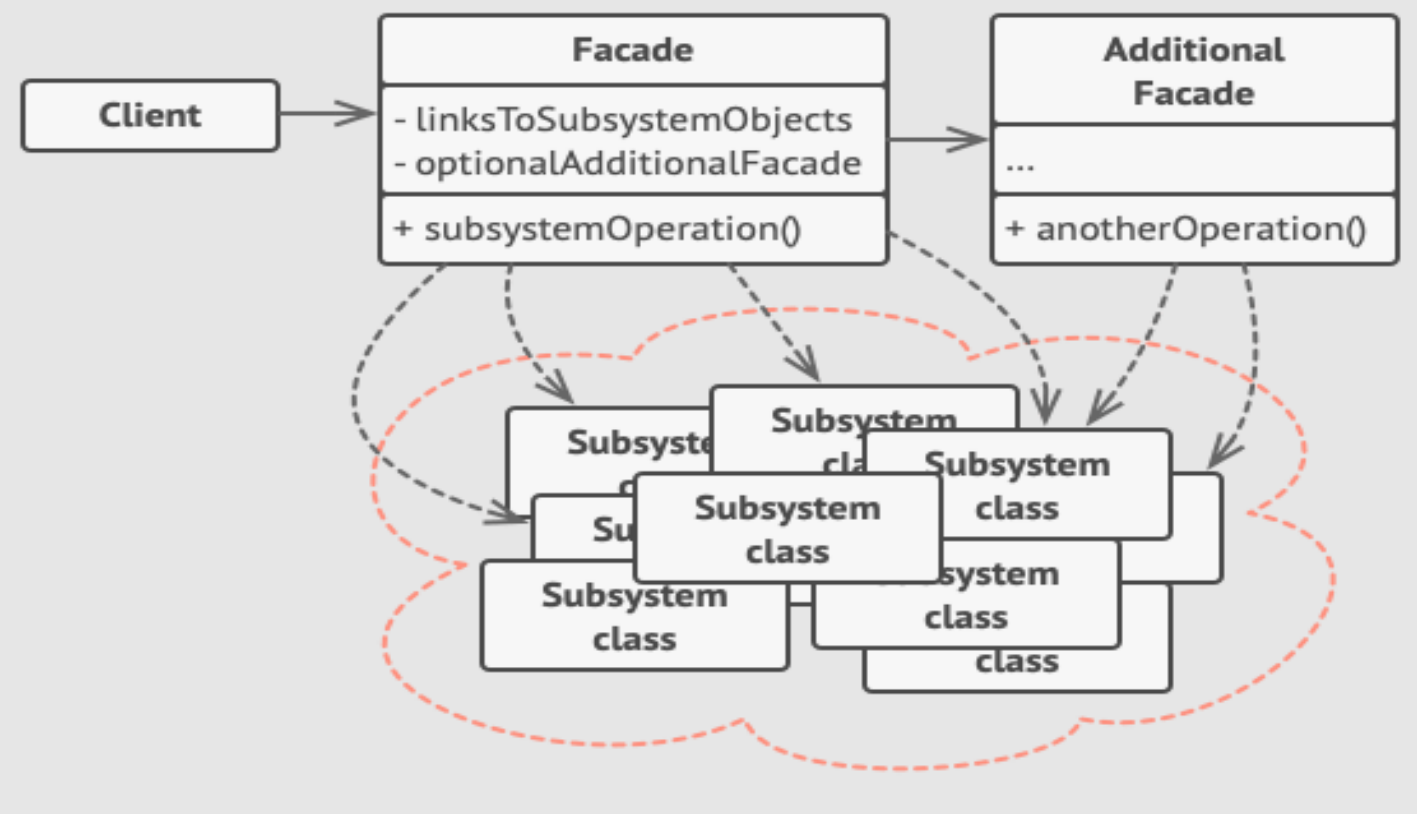
Decorator.

- Es un patrón de diseño estructural que te permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.
- Utiliza el patrón Decorator cuando necesites asignar funcionalidades adicionales a objetos durante el tiempo de ejecución sin descomponer el código que utiliza esos objetos.
- El patrón Decorator te permite estructurar tu lógica de negocio en capas, crear un decorador para cada capa y componer objetos con varias combinaciones de esta lógica, durante el tiempo de ejecución. El código cliente puede tratar a todos estos objetos de la misma forma, ya que todos siguen una interfaz común.
- Utiliza el patrón cuando resulte extraño o no sea posible extender el comportamiento de un objeto utilizando la herencia.
- Muchos lenguajes de programación cuentan con la palabra clave final que puede utilizarse para evitar que una clase siga extendiéndose. Para una clase final, la única forma de reutilizar el comportamiento existente será envolver la clase con tu propio wrapper, utilizando el patrón Decorator.



Facade.

- Es un patrón de diseño estructural que proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.
- Utiliza el patrón Facade cuando necesites una interfaz limitada pero directa a un subsistema complejo.
- Utiliza el patrón Facade cuando quieras estructurar un subsistema en capas.



Proxy.

- Es un patrón de diseño estructural que te permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.
- En lugar de crear el objeto cuando se lanza la aplicación, puedes retrasar la inicialización del objeto a un momento en que sea realmente necesario.
- El proxy puede registrar cada solicitud antes de pasarla al servicio.