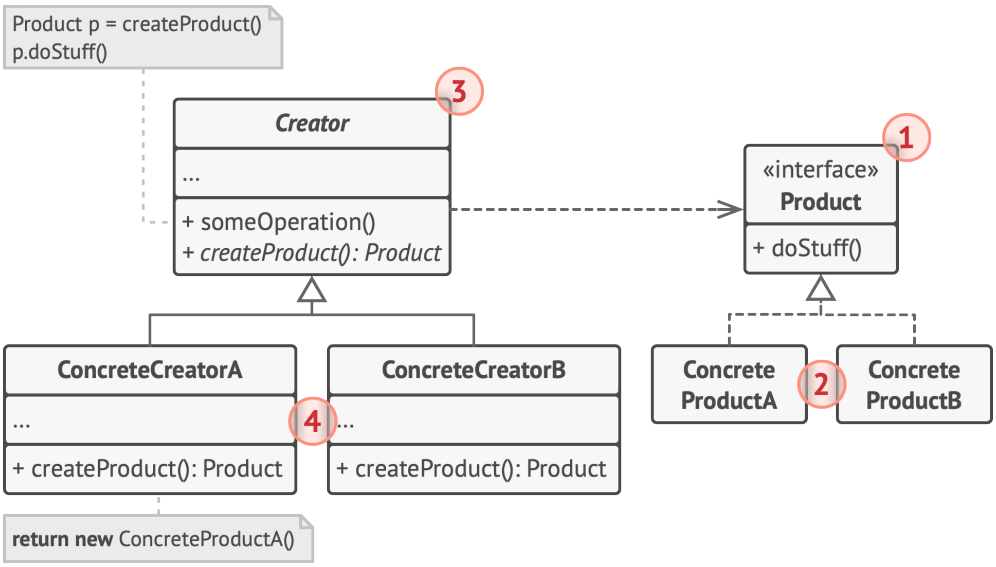
Estructura



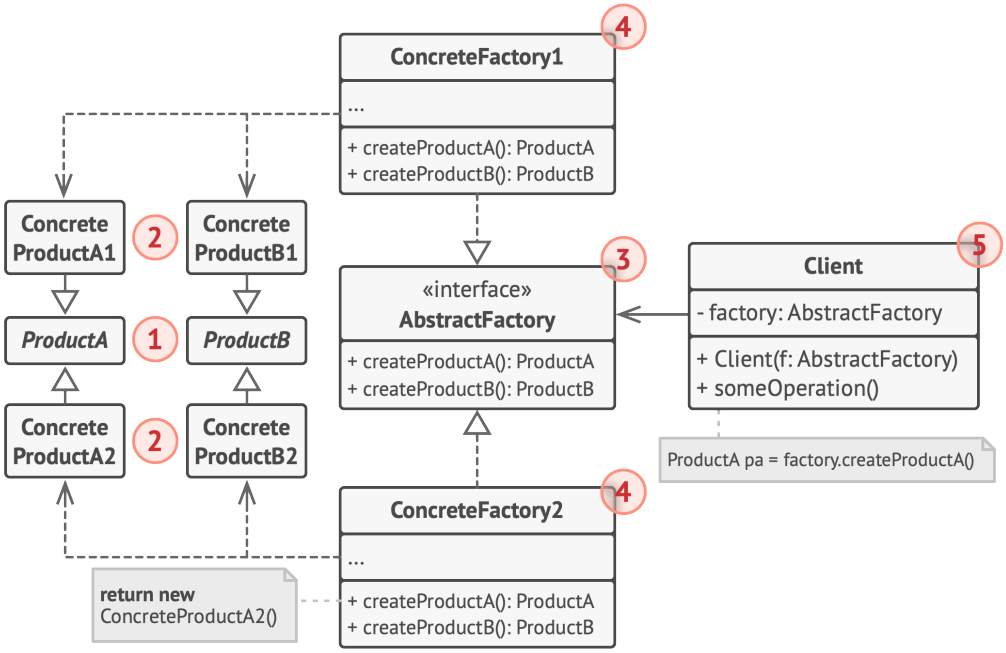
1. El **Producto** declara la interfaz, que es común a todos los objetos que puede producir la clase creadora y sus subclases.

2. Los **Productos Concretos** son distintas implementaciones de la interfaz de producto.

3. La clase **Creadora** declara el método fábrica que devuelve nuevos objetos de producto. Es importante que el tipo de retorno de este método coincida con la interfaz de producto.

   Puedes declarar el patrón Factory Method como abstracto para forzar a todas las subclases a implementar sus propias versiones del método. Como alternativa, el método fábrica base puede devolver algún tipo de producto por defecto.

Estructura



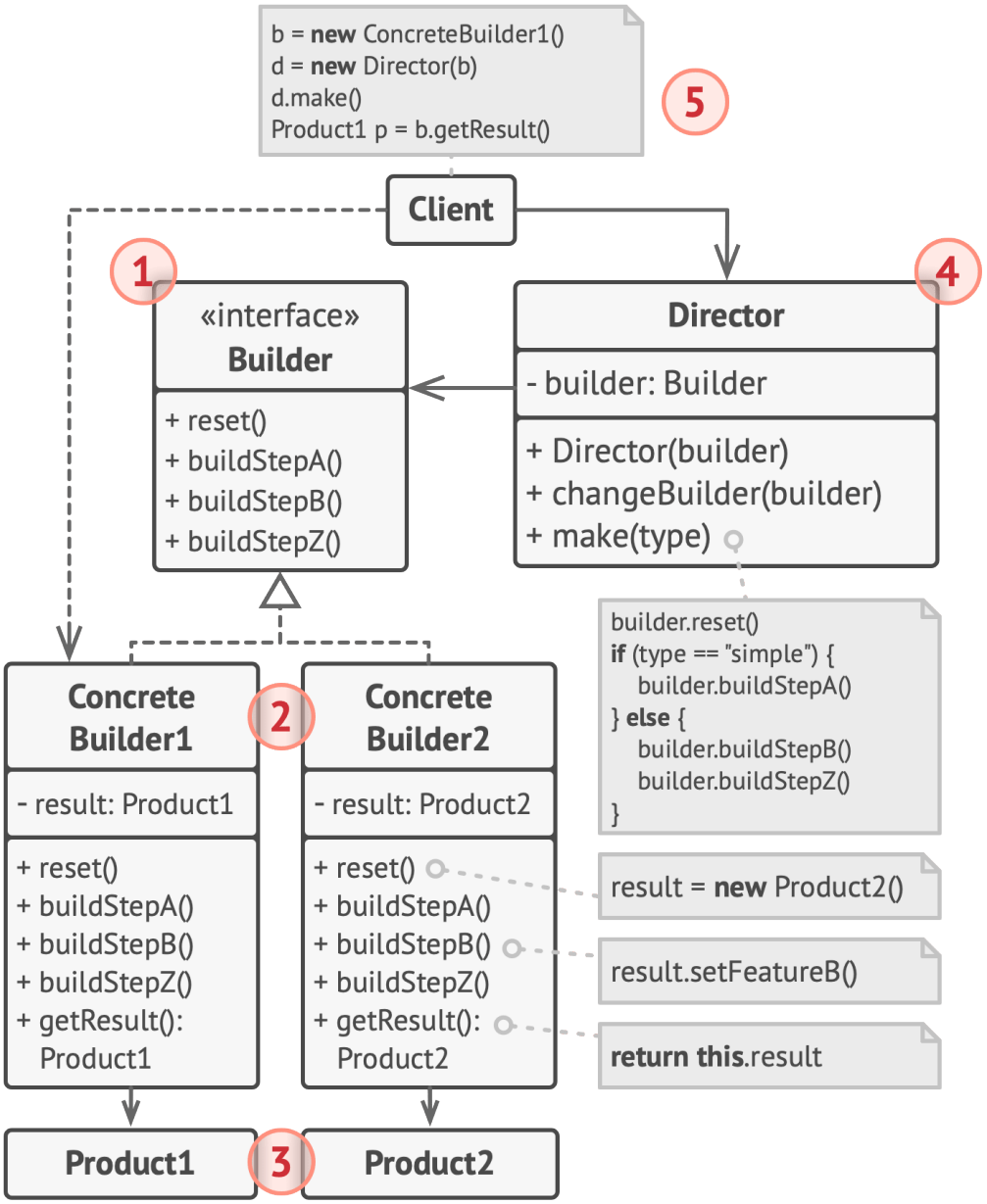
1. Los **Productos Abstractos** declaran interfaces para un grupo de productos diferentes pero relacionados que forman una familia de productos.

2. Los **Productos Concretos** son implementaciones distintas de productos abstractos agrupados por variantes. Cada producto abstracto (silla/sofá) debe implementarse en todas las variantes dadas (victoriano/moderno).

3. La interfaz **Fábrica Abstracta** declara un grupo de métodos para crear cada uno de los productos abstractos.

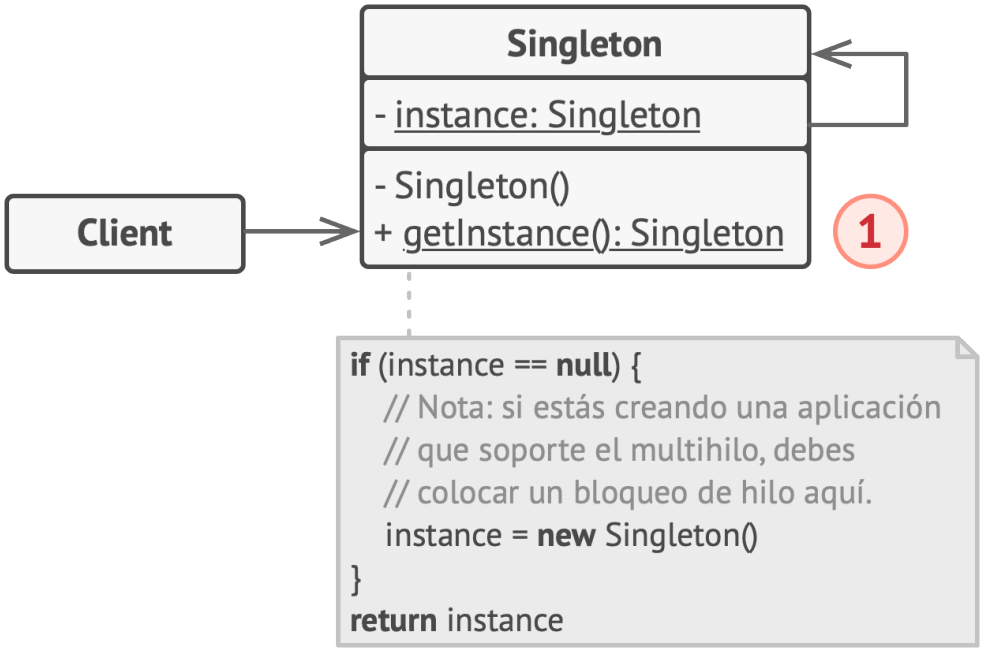
4. Las **Fábricas Concretas** implementan métodos de creación de la fábrica abstracta. Cada fábrica concreta se corresponde con

Estructura



1. La interfaz **Constructora** declara pasos de construcción de producto que todos los tipos de objetos constructores tienen en común.

Estructura

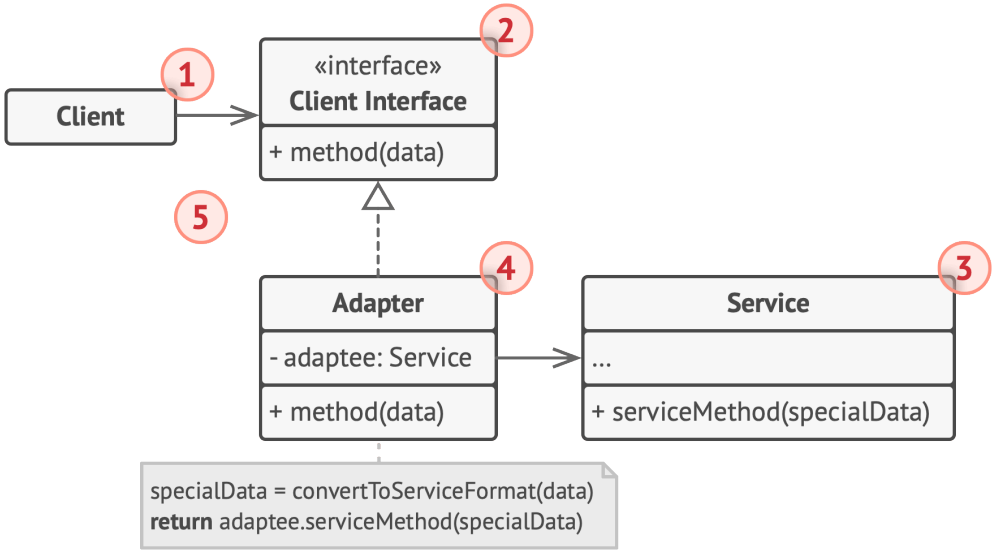


1. La clase **Singleton** declara el método estático `obtenerInstancia` que devuelve la misma instancia de su propia clase.

   El constructor del Singleton debe ocultarse del código cliente. La llamada al método `obtenerInstancia` debe ser la única manera de obtener el objeto de Singleton.

Pseudocódigo

En este ejemplo, la clase de conexión de la base de datos actúa como **Singleton**. Esta clase no tiene un constructor público, por lo que la única manera de obtener su objeto es invocando el método `obtenerInstancia`. Este método almacena en caché



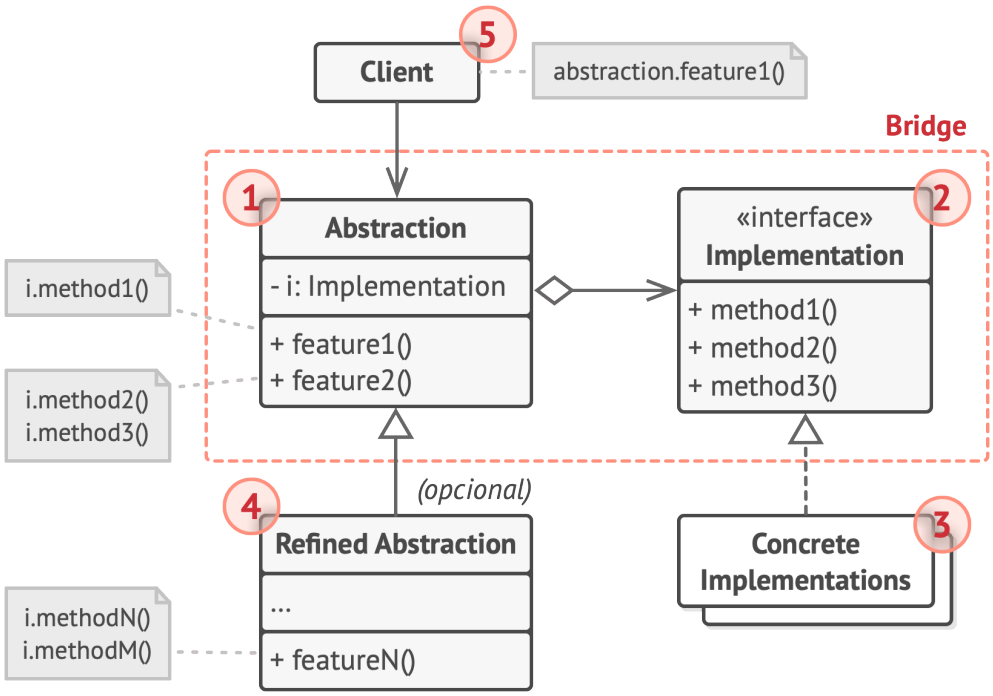
1. La clase **Cliente** contiene la lógica de negocio existente del programa.

2. La **Interfaz con el Cliente** describe un protocolo que otras clases deben seguir para poder colaborar con el código cliente.

3. **Servicio** es alguna clase útil (normalmente de una tercera parte o heredada). El cliente no puede utilizar directamente esta clase porque tiene una interfaz incompatible.

4. La clase **Adaptadora** es capaz de trabajar tanto con la clase cliente como con la clase de servicio: implementa la interfaz con el cliente, mientras envuelve el objeto de la clase de servicio. La clase adaptadora recibe llamadas del cliente a través de la interfaz adaptadora y las traduce en llamadas al objeto envuelto de la clase de servicio, pero en un formato que pueda comprender.

Estructura



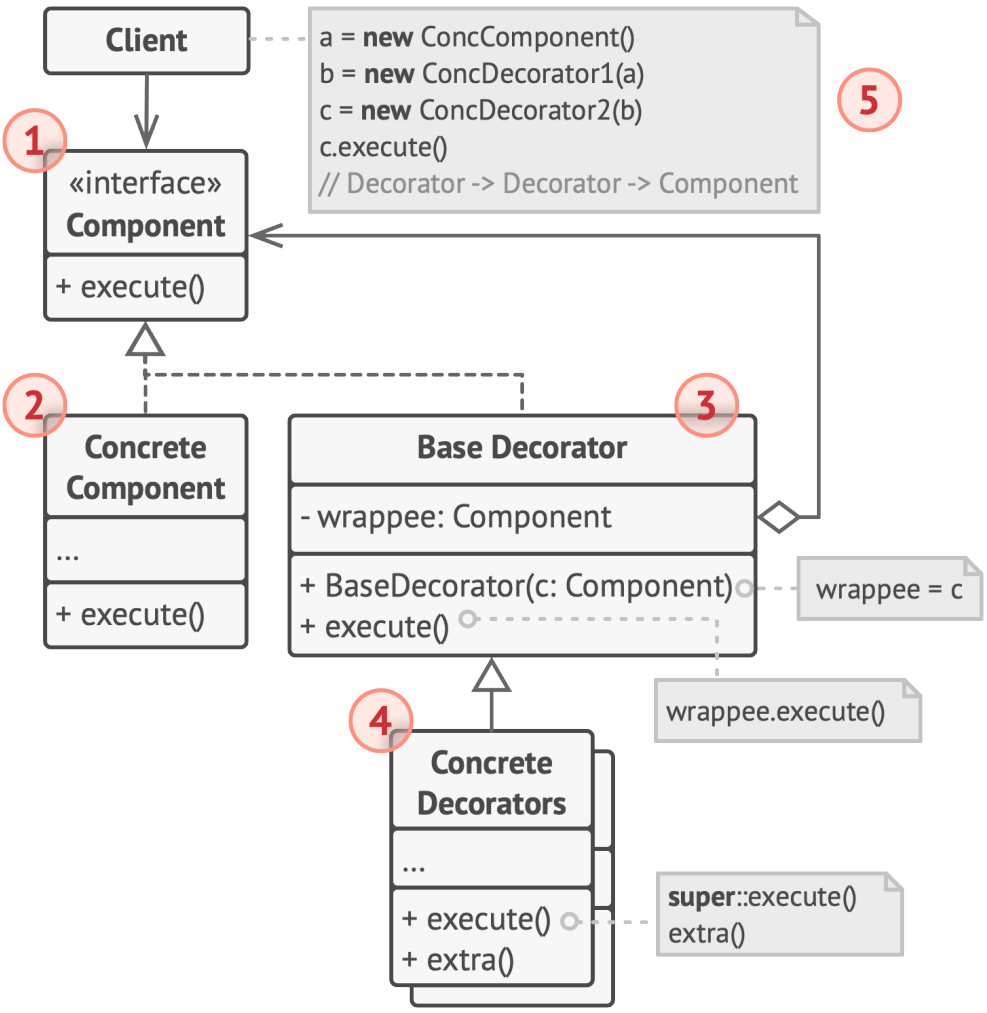
1. La **Abstracción** ofrece lógica de control de alto nivel. Depende de que el objeto de la implementación haga el trabajo de bajo nivel.

2. La **Implementación** declara la interfaz común a todas las implementaciones concretas. Una abstracción sólo se puede comunicar con un objeto de implementación a través de los métodos que se declaren aquí.

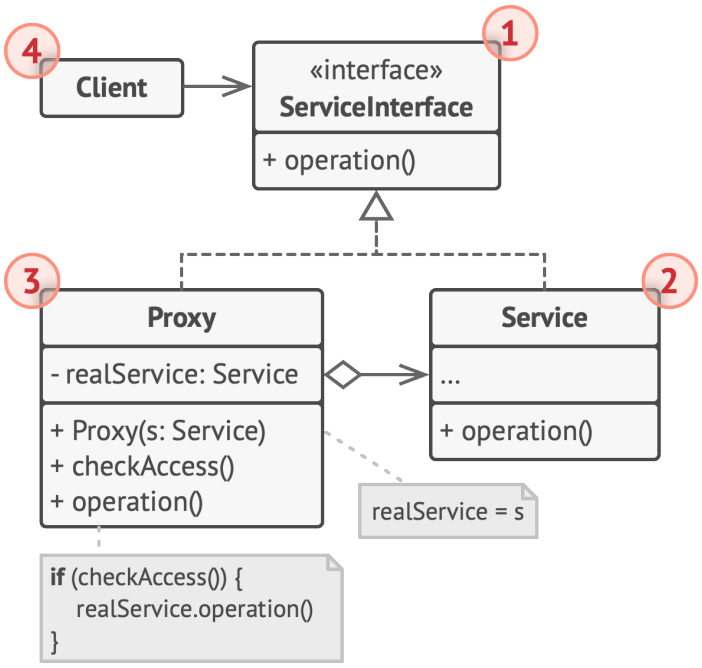
   La abstracción puede enumerar los mismos métodos que la implementación, pero normalmente la abstracción declara funcionalidades complejas que dependen de una amplia variedad de operaciones primitivas declaradas por la implementación.

Vestir ropa es un ejemplo del uso de decoradores. Cuando tienes frío, te cubres con un suéter. Si sigues teniendo frío a pesar del suéter, puedes ponerte una chaqueta encima. Si está lloviendo, puedes ponerte un impermeable. Todas estas prendas "extienden" tu comportamiento básico pero no son parte de ti, y puedes quitarte fácilmente cualquier prenda cuando lo desees.

Estructura



Estructura



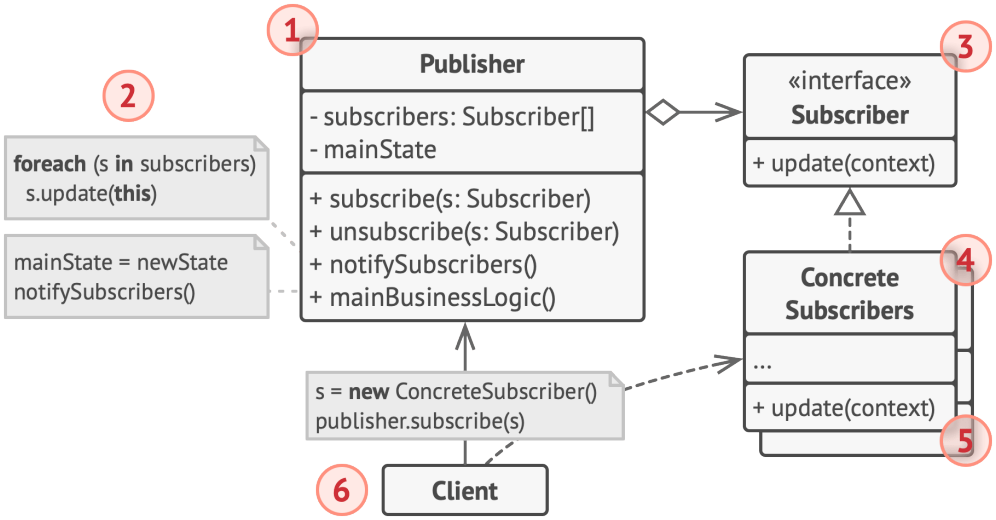
1. La **Interfaz de Servicio** declara la interfaz del Servicio. El proxy debe seguir esta interfaz para poder camuflarse como objeto de servicio.

2. **Servicio** es una clase que proporciona una lógica de negocio útil.

3. La clase **Proxy** tiene un campo de referencia que apunta a un objeto de servicio. Cuando el proxy finaliza su procesamiento (por ejemplo, inicialización diferida, registro, control de acceso, almacenamiento en caché, etc.), pasa la solicitud al objeto de servicio.

El notificador mantiene una lista de suscriptores y sabe qué revistas les interesan. Los suscriptores pueden abandonar la lista en cualquier momento si quieren que el notificador deje de enviarles nuevos números.

Estructura



1. El **Notificador** envía eventos de interés a otros objetos. Esos eventos ocurren cuando el notificador cambia su estado o ejecuta algunos comportamientos. Los notificadores contienen una infraestructura de suscripción que permite a nuevos y antiguos suscriptores abandonar la lista.

2. Cuando sucede un nuevo evento, el notificador recorre la lista de suscripción e invoca el método de notificación declarado en la interfaz suscriptora en cada objeto suscriptor.

3. La interfaz **Suscriptora** declara la interfaz de notificación. En la mayoría de los casos, consiste en un único método