

OBSERVER

Observer es un patrón de diseño de comportamiento que te permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.

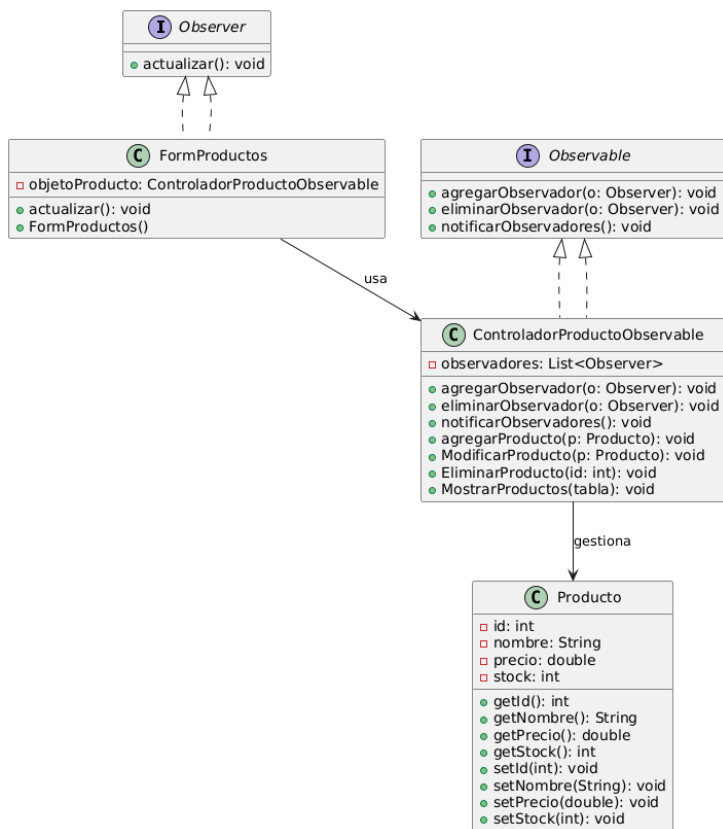
Cómo implementarlo

1. Repasa la lógica del negocio e intentar dividirla en dos partes: la funcionalidad central, independiente del resto de código, actuará como notificador; el resto se convertirá en un grupo de clases suscriptoras.
2. Declarar la interfaz suscriptora. Como mínimo, deberá declarar un único método actualizar.
3. Declarar la interfaz notificadora y describe un par de métodos para añadir y eliminar de la lista un objeto suscriptor. Recuerda que los notificadores deben trabajar con suscriptores únicamente a través de la interfaz suscriptora.
4. Decidir dónde colocar la lista de suscripción y la implementación de métodos de suscripción. Normalmente, este código tiene el mismo aspecto para todos los tipos de notificadores, por lo que el lugar obvio para colocarlo es en una clase abstracta derivada directamente de la interfaz notificadora. Los notificadores concretos extienden esa clase, heredando el comportamiento de suscripción.
5. Crear clases notificadoras concretas. Cada vez que suceda algo importante dentro de una notificadora, deberá notificar a todos sus suscriptores.
6. Implementar los métodos de notificación de actualizaciones en clases suscriptoras concretas. La mayoría de las suscriptoras necesitarán cierta información de contexto sobre el evento, que puede pasarse como argumento del método de notificación.
7. El cliente debe crear todos los suscriptores necesarios y registrarlos con los notificadores adecuados.

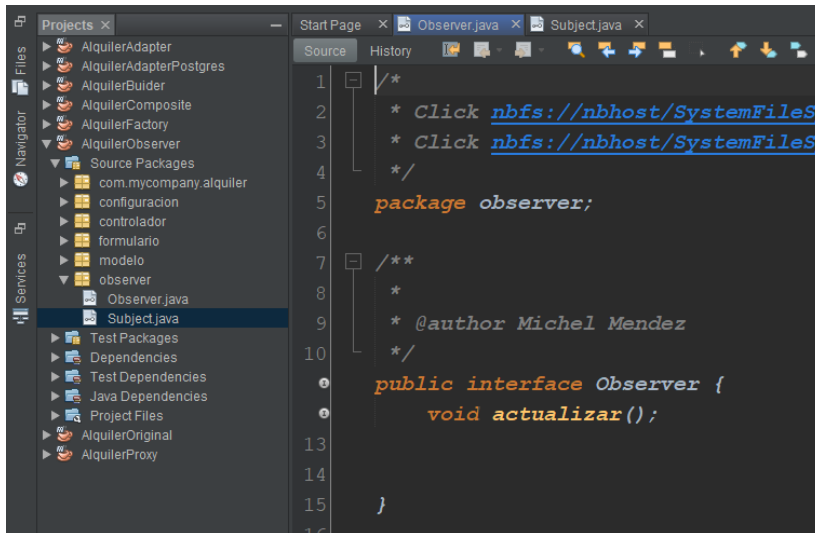
Pros y contras

- *Principio de abierto/cerrado.* Puedes introducir nuevas clases suscriptoras sin tener que cambiar el código de la notificadora (y viceversa si hay una interfaz notificadora).
- Puedes establecer relaciones entre objetos durante el tiempo de ejecución.
- Los suscriptores son notificados en un orden aleatorio.

IMPLEMENTACION DEL PATRON EN EL PROYECTO DE ALQUILER UML



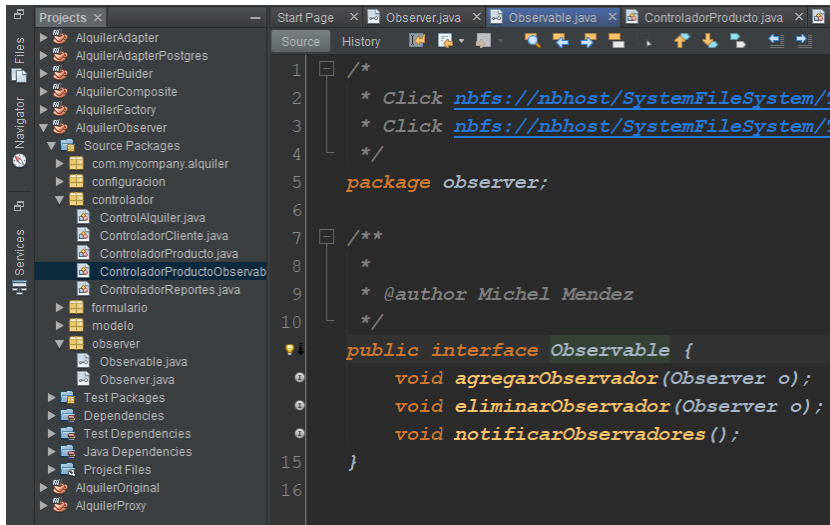
IMPLEMENTACION DEL PATRON EN EL CODIGO



INTERFAZ OBSERVER

Esta interfaz define el contrato para los observadores.

Cualquier clase que quiera "observar cambios" deberá implementar este método `actualizar()`

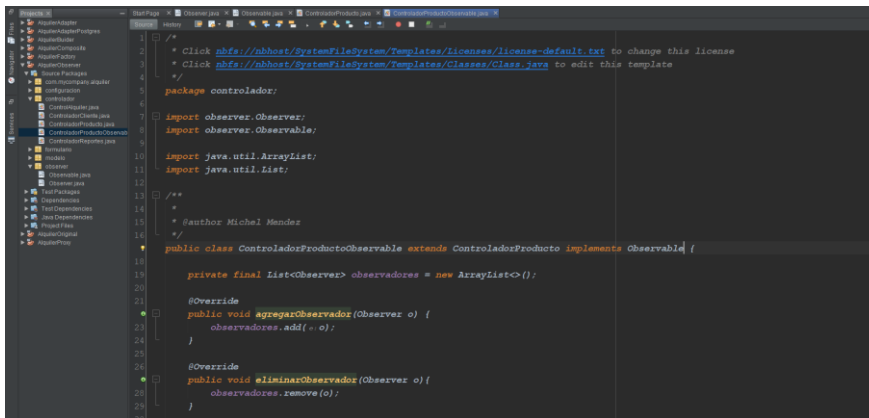


Esta interfaz define el comportamiento del observador.

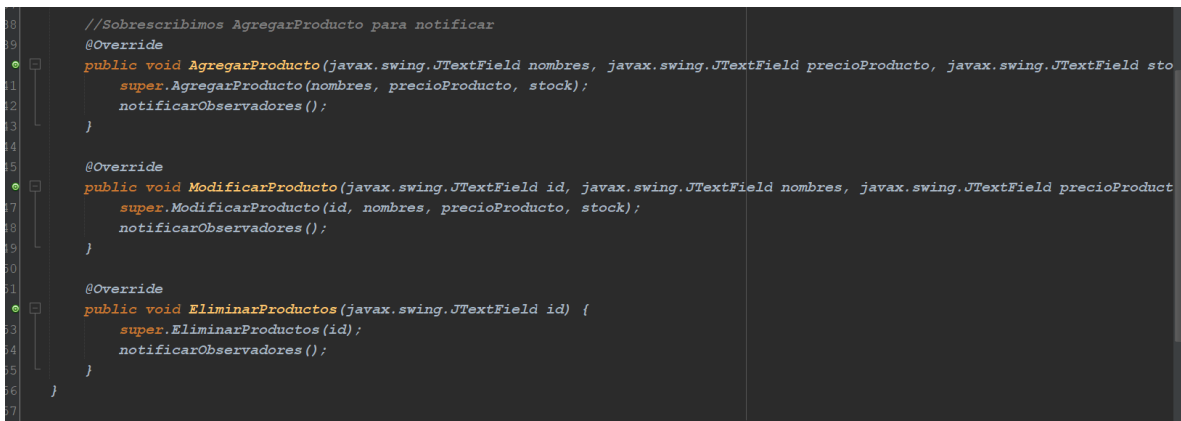
Cualquier clase que quiera permitir que otros la "observen" debe implementar esta interfaz.

Clase ControladorProductoObservable

Implementa la interfaz Observable. Es el sujeto observado, es decir, el que notificara a los formularios cuando se agregue/modifique/elimine un producto.



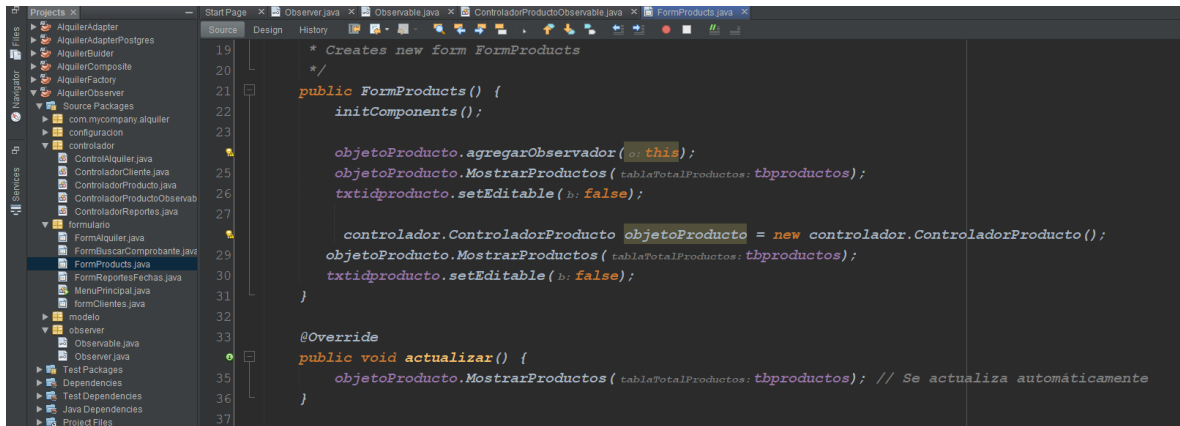
notificarObservadores() ← aquí se notifica a todos los observadores



Clase FormProductos

Esta clase es un observador, porque implementa Observer

- En el constructor, FormProductos se registra como observador del ControladorProductoObservable.
- Cuando otro formulario o proceso modifica los productos, se llama notificarObservadores() y se actualiza automáticamente.



```
19  * Creates new form FormProductos
20  */
21  public FormProductos() {
22      initComponents();
23
24      objetoProducto.agregarObservador(this);
25      objetoProducto.MostrarProductos( tablaTotalProductos.tbproductos);
26      txtidproducto.setEditable( b: false);
27
28      controlador.ControladorProducto objetoProducto = new controlador.ControladorProducto();
29      objetoProducto.MostrarProductos( tablaTotalProductos.tbproductos);
30      txtidproducto.setEditable( b: false);
31  }
32
33  @Override
34  public void actualizar() {
35      objetoProducto.MostrarProductos( tablaTotalProductos.tbproductos); // Se actualiza automáticamente
36  }
37
```