Patrones de diseño: Comportamiento

Observer y Template by: Sierra Team

Group Members

SIERRA

Camila Saenz

Wendy Araya

Felipe Masís

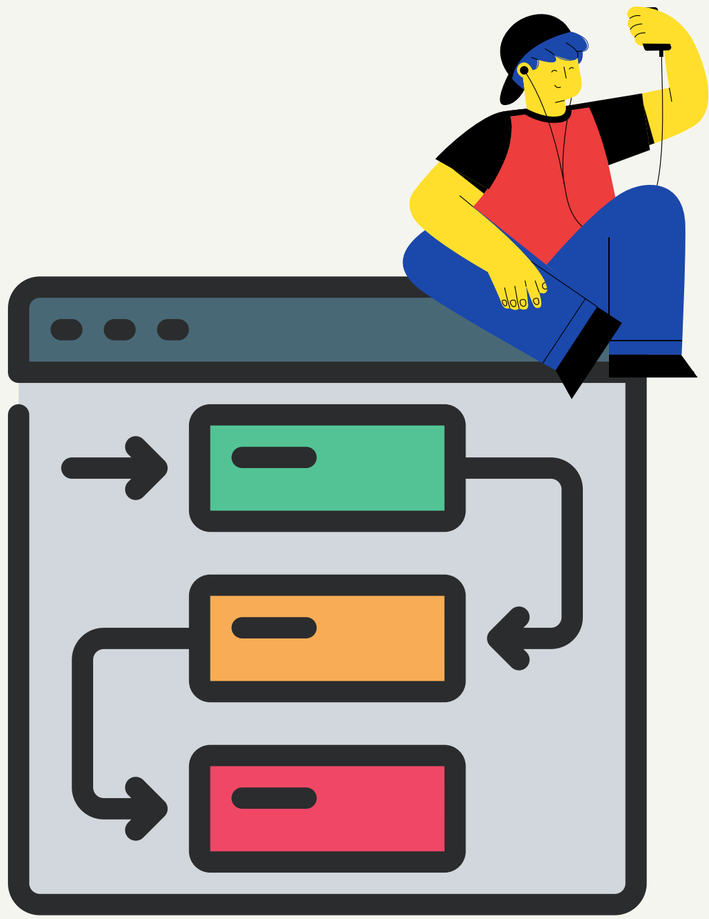
Observer

Part 01

Background

Observer ó Publish-subscribe dependents

El patrón observer define una dependencia de uno a muchos, entre los objetos así que cuando un objeto cambia su estado, todos los dependientes son notificados y actualizados automáticamente.



¿Qué lo compone?

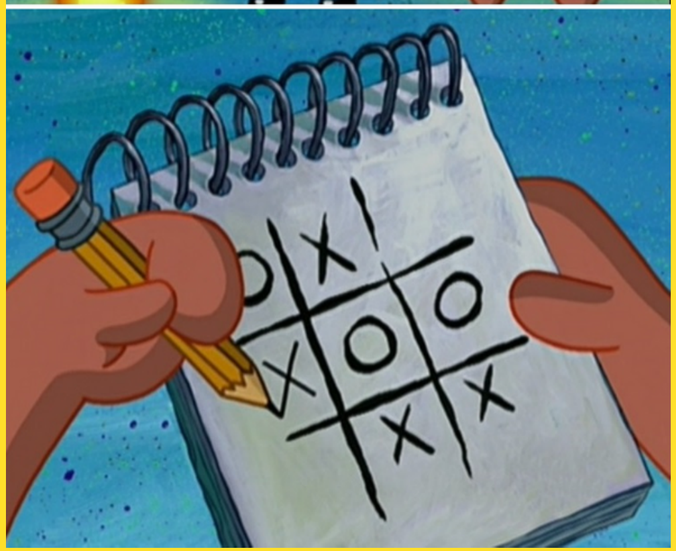
Sujeto

Un sujeto puede tener cualquier número de observadores dependientes de él. Cada vez que el sujeto cambia su estado, le notifica a todos sus observadores.

Observador

En respuesta, cada observador consultará al sujeto para sincronizar su estado con el estado de este.

¿Cuándo se usa?



Cuando una abstracción tiene dos aspectos y uno depende del otro. Al encapsular estos aspectos en objetos separados permite modificarlos y re utilizarlos de forma dependiente.

Cuando un cambio en un objeto requiere cambiar otros y no sabemos cuántos objetos necesitan cambiarse.

Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quienes son dichos objetos. En otras palabras, cuando no queremos que estos objetos estén fuertemente acoplados.

Componentes de Observer

Sujeto

Conoce a sus observadores. Un sujeto puede ser observado por cualquier número de objetos Observador.
Proporciona una interfaz para asignar y quitar objetos observador.

Observador

Define una interfaz para actualizar los objetos que deben ser notificados ante cambios en un sujeto.

Sujeto Concreto

Almacena el estado de interés para los objetos Observador Concreto
Envía una notificación a sus observadores cuando cambia su estado.

Observador Concreto

Mantiene una referencia a un objeto Sujeto Concreto
Guarda un estado que debería ser consistente con el del sujeto.
Implementa la interfaz de actualización del observador para mantener su estado consistente con el del sujeto.

Observer:

Ventajas y puntos a destacar

Modificación de los sujetos y observadores de forma independiente

Así que reutilizar objetos sin reutilizar sus observadores y viceversa. Con esto se pueden añadir observadores sin modificar el sujeto u otros observadores.

Acoplamiento abstracto entre sujeto y observador

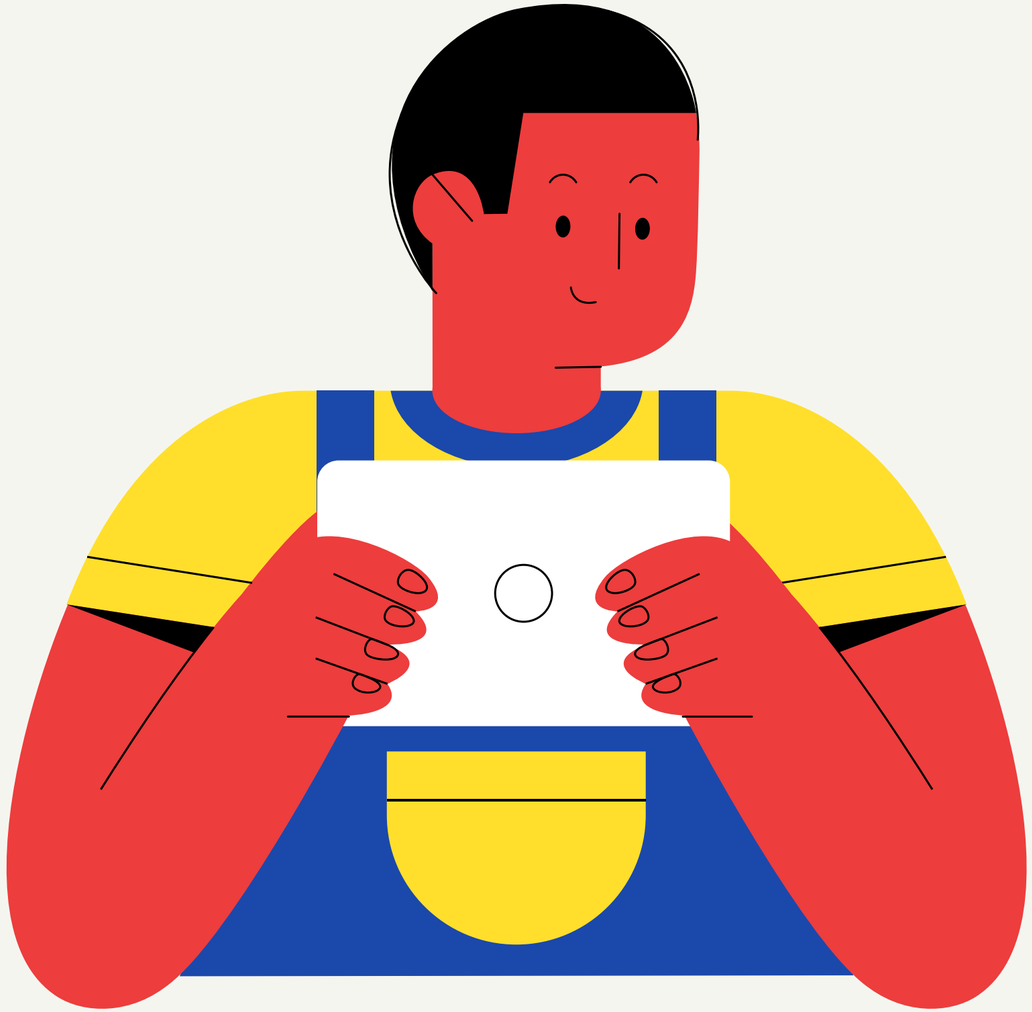
El sujeto solo sabe que tiene una lista de observadores y cada uno de esos se ajusta a interfaz simple de la clase abstracta observador. El sujeto no conoce la clase concreta de ningún observador así que el acoplamiento entre ambas clases es mínimo.

Capacidad de comunicación mediante difusión

A diferencia de una petición ordinaria, la notificación enviada por un sujeto no necesita especificar su receptor. La notificación se envía automáticamente a todos los objetos interesados que se hayan suscrito. Al sujeto no le importa cuántos hay, su única responsabilidad es notificar a los observadores. De esta forma tenemos la libertad de quitar o añadir observadores en cualquier momento.

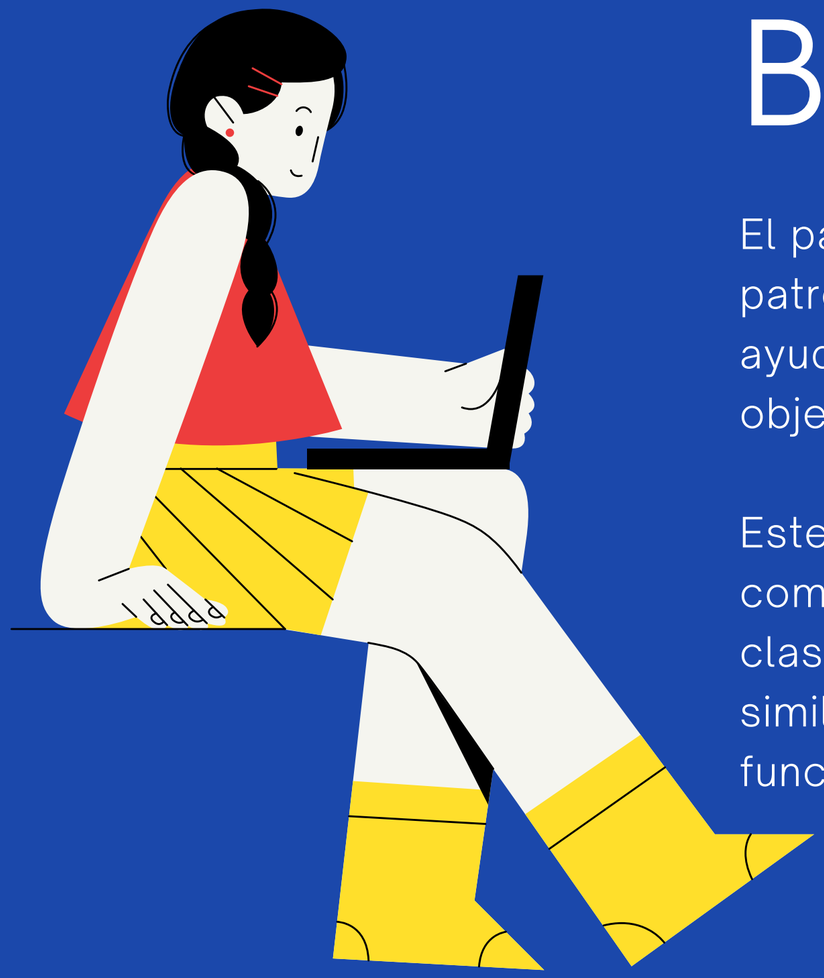
Ejemplo Práctico

Part 02



Template

Part 03



Background

El patrón de diseño Template Method forma parte de la familia de patrones denominados de comportamiento. Este tipo de patrones ayudan a resolver problemas de interacción entre clases y objetos.

Este patrón nace de la necesidad de extender determinados comportamientos dentro del código por parte de diferentes clases. Es decir, diferentes clases tienen un comportamiento similar pero que difiere en determinados aspectos puntuales en función de la clase.

¿Cómo se implementa?

Solución

Se propone abstraer todo el comportamiento que comparten las entidades en una clase (abstracta) de la que, posteriormente, extenderán dichas entidades. Esta superclase definirá un método que contendrá el esqueleto de ese algoritmo común (método plantilla o template method) y delegará determinada responsabilidad en las clases hijas, mediante uno o varios métodos abstractos que deberán implementar sin cambiar su estructura.

Funcionalidades

1

Cuando se quiera implementar las partes de un algoritmo que no cambian y dejar que las subclases implementan aquellas otras que puedan variar.

2

Cuando se requiera reducir código, por lo que movemos cierto código a una clase base común evitar código duplicado.

3

Cuando se quiera controlar el modo en que las subclases extienden la clase base. Haciendo que solo sea a través de unos métodos de plantilla datos.

¿Cuándo se usa?

Cuando dentro del código se detecte pasos o métodos que tendrán que repetirse una y otra vez.

Cuando se presenten muchos Switch en una sección de código.

Cuando se desconozca la magnitud que pueda tener una clase padre para las clases hijas.

Beneficios de aplicar este patrón

1.

Se puede reutilizar una clase abstracta en las subclases

2.

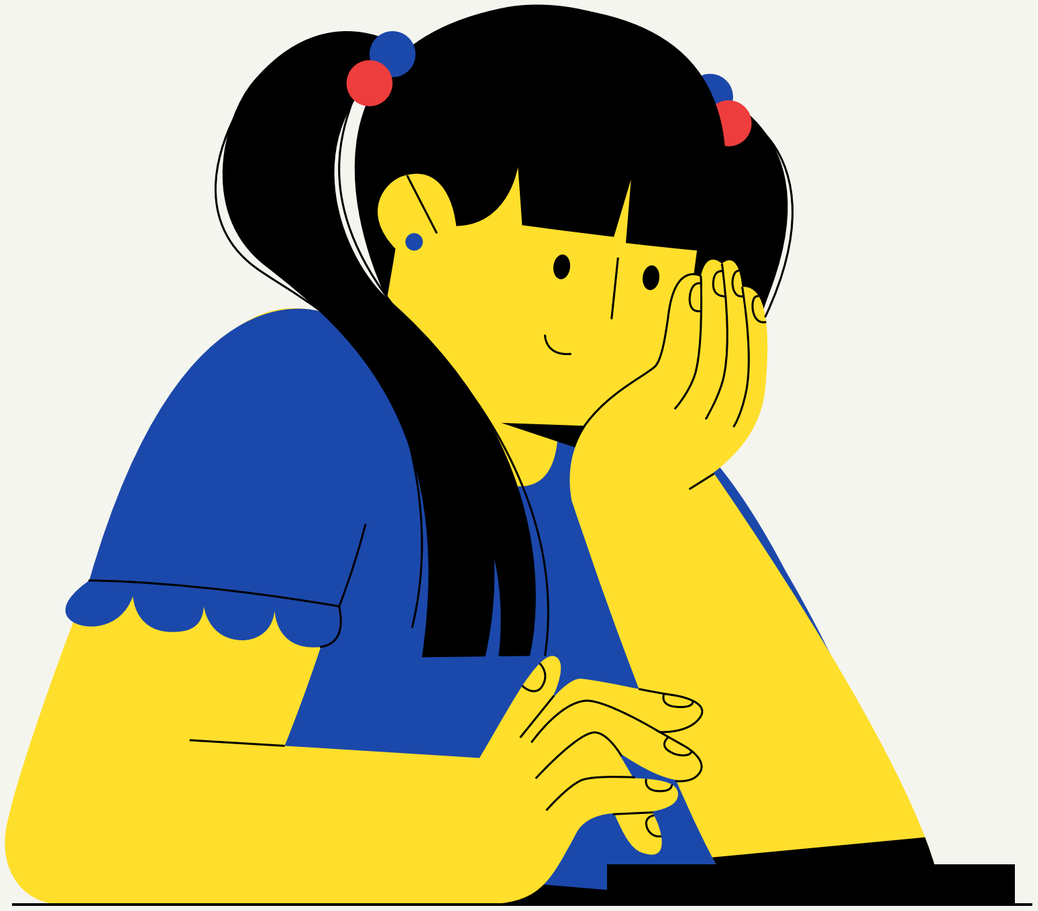
No se modifica el comportamiento de la clase padre.

3.

Nada afecta a la clase padre porque la herencia solo extiende la funcionalidad.

Importante:

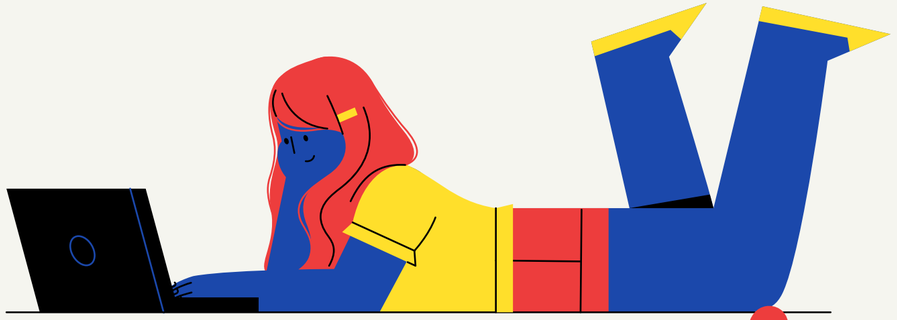
Se recomienda utilizar el patrón Template Method cuando se detecte a tiempo que el diseño tendrá pasos que puedan repetirse una y otra vez. Es importante recordar que el objetivo es crear una clase abstracta con métodos que puedan ser sobre-escritos más no modificar el comportamiento de la clase padre.



Ejemplo Práctico

Part 04

¿Preguntas?



Thank you!