Contenido

[CADENA DE RESPONSABILIDADES 2](#_Toc124448555)

[Diagrama Genérico 2](#_Toc124448556)

[DECORADOR 2](#_Toc124448557)

[Diagrama Genérico 2](#_Toc124448558)

[ESTRATEGIA 3](#_Toc124448559)

[Diagrama Genérico 3](#_Toc124448560)

[FÁBRICA ABSTRACTA 4](#_Toc124448561)

[Diagrama Genérico 4](#_Toc124448562)

[OBSERVER 5](#_Toc124448563)

[Diagrama Genérico 5](#_Toc124448564)

[PLANTILLA 6](#_Toc124448565)

[Diagrama Genérico 6](#_Toc124448566)

[SINGLETON 6](#_Toc124448567)

[Diagrama Genérico 6](#_Toc124448568)

[MEMENTO 7](#_Toc124448569)

[Diagrama Genérico 7](#_Toc124448570)

# CADENA DE RESPONSABILIDADES

Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

## Diagrama Genérico

* **Cliente:**  Inicia la petición que llega a la cadena en busca del responsable.
* **Manejador:** Define una Interfaz para manejar peticiones
* **Manejador Concreto:** Define las responsabilidades de cada componente. Si puede manejar una petición, la procesa, en caso contrario busca al siguiente.

# DECORADOR

El objetivo del patrón Decorator es agregar dinámicamente funcionalidades suplementarias a un objeto. Esta agregación de funcionalidades no modifica la interfaz del objeto.

## Diagrama Genérico

* **ComponenteAbstracto:** Es la interfaz común al componente y a los decoradores.
* **ComponenteConcreto:** Es el objeto inicial al que se deben agregar las nuevas funcionalidades.
* **Decorador:** Es una clase abstracta que guarda una referencia hacia el componente
* **DecoradorConcretoA y DecoradorConcretoB:** Son subclases concretas de Decorador que tienen como objetivo implementar las funcionalidades agregadas al componente.

# ESTRATEGIA

El patrón de diseño strategy permite establecer en tiempo de ejecución el rol de comportamiento de una clase.

## Diagrama Genérico

* + **Context:** La clase Contextomantiene una referencia a una de las estrategias concretas y se comunica con este objeto únicamente a través de la interfaz estrategia.
  + **Estrategy:** La interfaz **Estrategia** es común a todas las estrategias concretas. Declara un método que la clase contexto utiliza para ejecutar una estrategia.
  + **Concrete Strategy:** Las Estrategias Concretasimplementan distintas variaciones de un algoritmo que la clase contexto utiliza.

# FÁBRICA ABSTRACTA

Es un patrón de diseño creacional que nos permite producir familias de objetos relacionados sin especificar sus clases concretas. Conjunto de Factory métodos, También conocido como kit.

## Diagrama Genérico

* **Fabrica abstracta:** es una interfaz que define las firmas de los métodos que crean los distintos productos.  
  **Fabrica concreta 1, Fabrica concreta 2:** son las clases concretas que implementan los métodos que crean los productos para cada familia de productos. Conociendo la familia y el producto, son capaces de crear una instancia del producto para esta familia.
* **Producto Abstracto A, Producto Abstracto B:** son las clases abstractas de los productos independientemente de su familia. Las familias se introducen en las subclases concretas.  
  **Producto A1, Producto A2, Producto B1, Producto B2:** son los productos finales creados por las fabrica concretas (Fabrica concreta 1, Fabrica concreta 2).  
  **Cliente:** es la clase que utiliza la interfaz de Fabrica Abstracta.

# OBSERVER

Es un patrón de comportamiento, se debe usar el patrón observador cuando un elemento quiere estar pendiente de otro, sin tener que estar comprobando de forma continua si ha cambiado o no.

## Diagrama Genérico

* **Sujeto (subject)**: El sujeto proporciona una interfaz para **agregar (attach)** y **eliminar (detach)** observadores. El Sujeto conoce a todos sus observadores.
* **Observador (observer)**: Define el método que usa el sujeto para notificar cambios en su estado (update/notify).
* **Sujeto concreto (concrete subject)**: Mantiene el estado de interés para los observadores concretos y los notifica cuando cambia su estado. No tienen por qué ser elementos de la misma jerarquía.
* **Observador concreto (concrete observer)**: Mantiene una referencia al sujeto concreto e implementa la interfaz de actualización, es decir, guardan la referencia del objeto que observan, así en caso de ser notificados de algún cambio, pueden preguntar sobre este cambio.

# PLANTILLA

Es un patrón de diseño que define el esqueleto de un algoritmo en una operación, delegando algunos pasos a las subclases. El método de plantilla permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar la estructura del algoritmo.

## Diagrama Genérico

* **Clase Abstracta:** Clase que incluye el método Plantilla, así como la llamada a los métodos abstractos que invoca este método
* **Clase Concreta:** Subclase que implementa los métodos abstractos utilizados por el método plantilla de la clase abstracta. Puede haber varias clases concretas

# SINGLETON

El patrón Singleton tiene como objetivo asegurar que una clase sólo posee una instancia y proporcionar un método de clase único que devuelva esta instancia.

## Diagrama Genérico

# MEMENTO

Memento es un patrón de diseño de comportamiento, que guarda parte o todo el estado interno de un objeto, para que este estado pueda ser restaurado posteriormente. Esta operación debe ocurrir sin romper el principio del encapsulamiento.

## Diagrama Genérico

* **Originator:** Es el objeto para el que se va a guardar el estado. Crea el Memento y lo utiliza para recuperar su estado
* **Memento:** Almacena el estado del Originator en un momento determinado
* **Caretaker:** Registra los cambios del Originator. Nos permite viajar entre los distintos estados del Originator.
* **Cliente:** Es el que afectar al Originator y registra al nuevo estado con el Caretaker. Es decir, realiza el cambio sobre el objeto y registra el estado.

# BRIDGE

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.

## Diagrama Genérico

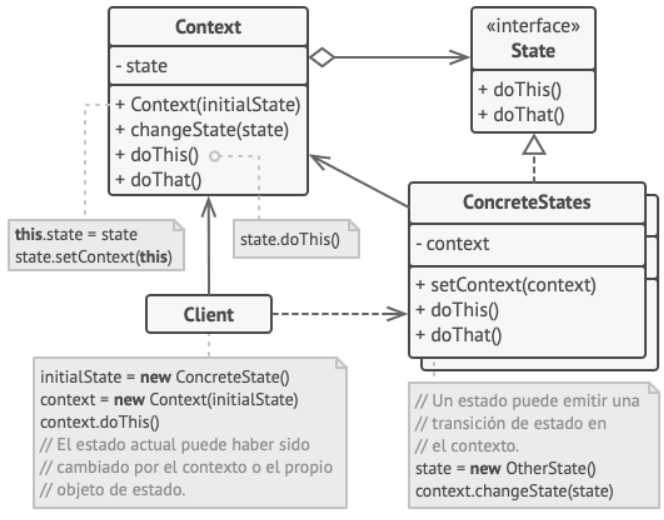
1. **La Abstracción** ofrece lógica de control de alto nivel. Depende de que el objeto de la implementación haga el trabajo de bajo nivel.
2. **La Implementación**declara la interfaz común a todas las implementaciones concretas. Una abstracción sólo se puede comunicar con un objeto de implementación 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.
3. **Las Implementaciones Concretas** contienen código específico de plataforma.
4. **Las Abstracciones Refinadas** proporcionan variantes de lógica de control. Como sus padres, trabajan con distintas implementaciones a través de la interfaz general de implementación.
5. **Normalmente, el Cliente** sólo está interesado en trabajar con la abstracción. No obstante, el cliente tiene que vincular el objeto de la abstracción con uno de los objetos de la implementación.

# ESTADO

La idea del Patrón Estado es crear objetos que representan los diferentes estados y un objeto que será el contexto que variará su comportamiento en función del estado del objeto.

El Patrón Estado, nos va a ayudar a simplificar y a ver de una manera más fácil e intuitiva de los diferentes estados en los que nos vamos a encontrar.

## Diagrama Genérico



* La clase **Contexto** almacena una referencia a uno de los objetos de estado concreto y le delega todo el trabajo específico del estado.
* La interfaz **Estado** declara los métodos específicos del estado.
* Los **Estados Concretos** proporcionan sus propias implementaciones para los métodos específicos del estado.