**Patrón de diseño STATE**

**Grupo 12**

Definición

Patrón Bridge

Bridge es un patrón de diseño utilizado para desacoplar una abstracción de su

implementación de manera que las dos puedan ser modificadas por separado sin

necesidad de modificar la otra; dicho de otra manera, se desacopla una

abstracción de su implementación para que puedan variar independientemente.

Este patrón puede ser confundido con Adapter ya que incluso Bridge se

implementa utilizando el patrón Adapter, sin embargo, existe una diferencia entre

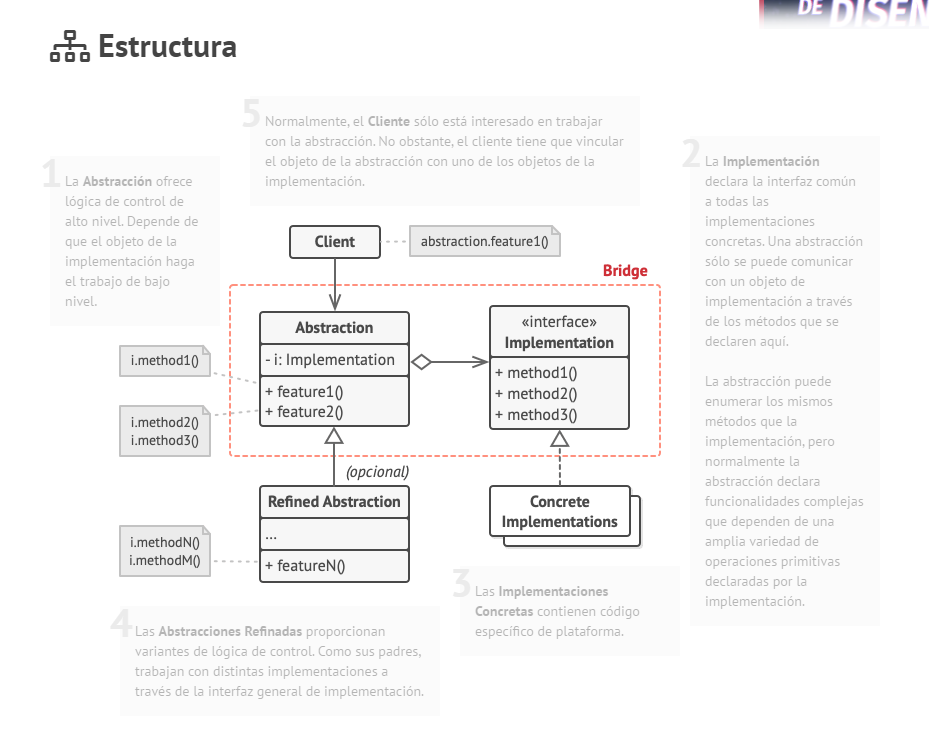
los dos, la cual es clave para distinguirlo, y es que Bridge se centra en desacoplar

la abstracción de la implementación por lo que tiene sentido utilizarse cuando

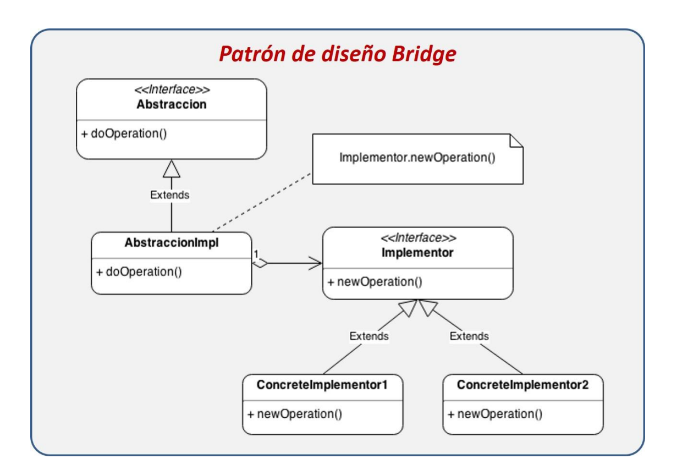
existen interfaces compatibles. En cambio Adapter es utilizada para resolver

escenarios donde las interfaces no son compatibles.

**Estructura del patron:**



**Ejemplo de aplicación del patron:**

****

Abstraction: Interface que define la estructura de la clase adaptadora o

bridge.

AbstractionImpl: Clase utilizada como puente para desacoplar a

Abstraction de Implementor, la clase hereda de Abstraction.

Implementor: Define una estructura de clase común para todas los

ConcreteImplementor. Esta interface no es estrictamente requerida para

implementar el patrón.

ConcreteImplementor: Conjunto de clases que heredan de Implementor

y que son propensas a cambiar, es por esta razón que se opta por

implementar el patrón de diseño Bridge.

**Pros y contras del patrón.**

Al igual que otros patrones, un puente tiene ventajas y desventajas. **Ventajas del patrón de puente:**

1. Mejora la escalabilidad del código: puede agregar funcionalidad sin temor a romper algo en otra parte del programa.
2. Reduce el número de subclases cuando el número de entidades estaría basado en combinaciones de dos conceptos (por ejemplo, formas y colores).
3. Hace posible trabajar por separado en dos jerarquías separadas: Abstracción e Implementación. Dos desarrolladores diferentes pueden realizar cambios sin profundizar en los detalles del código del otro.
4. Reduce el acoplamiento entre clases: el único lugar donde se acoplan las dos clases es el puente (es decir, el Color colorcampo).

**Desventajas del patrón:**

1. Según la situación específica y la estructura general de un proyecto, podría afectar negativamente el rendimiento de un programa (por ejemplo, si necesita inicializar más objetos).
2. Hace que el código sea menos legible debido a la necesidad de cambiar entre las dos clases.

**Bibliografía utilizada:**

Introducción a los Patrones de Diseño - Oscar Blancarte . 2016

Orientación a Objetos. Java y UML – Carlos Fontela – Nueva Librería – 2011 (Capitulo 20)

https://codegym.cc/es/groups/posts/es.296.patron-de-diseno-de-puente