Patrones Estructurales

Bridge

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"
[Christopher Alexander].

Por: Brayan Estiven Aguirre Aristizábal – 20231020156
Santiago Zamudio Diaz – 20231020128
Daniel Josué Rozo Rodríguez – 20222020006

1. Introducción

En el desarrollo de software, a menudo nos enfrentamos a la necesidad de separar la abstracción de su implementación para que ambas puedan variar de manera independiente. Esta situación es común en aplicaciones donde la escalabilidad, flexibilidad y mantenimiento son cruciales, tales como interfaces gráficas de usuario, sistemas de archivos, y motores de renderizado.

El patrón Bridge, uno de los patrones estructurales de diseño, nos proporciona una solución elegante para desacoplar una abstracción de su implementación. Este patrón permite que ambas puedan evolucionar de manera independiente sin afectar el funcionamiento del sistema. A través del uso de interfaces y la composición, el patrón Bridge facilita la creación de estructuras de software que son más manejables, extensibles y mantenibles. Veamos en detalle cómo funciona este patrón y en qué situaciones puede ser especialmente útil.

2. Bridge

Nombre del patrón:
Bridge

Clasificación del patrón:
Estructural

Intención:
Desacoplar una abstracción de su implementación, de modo que ambos puedan variar independientemente

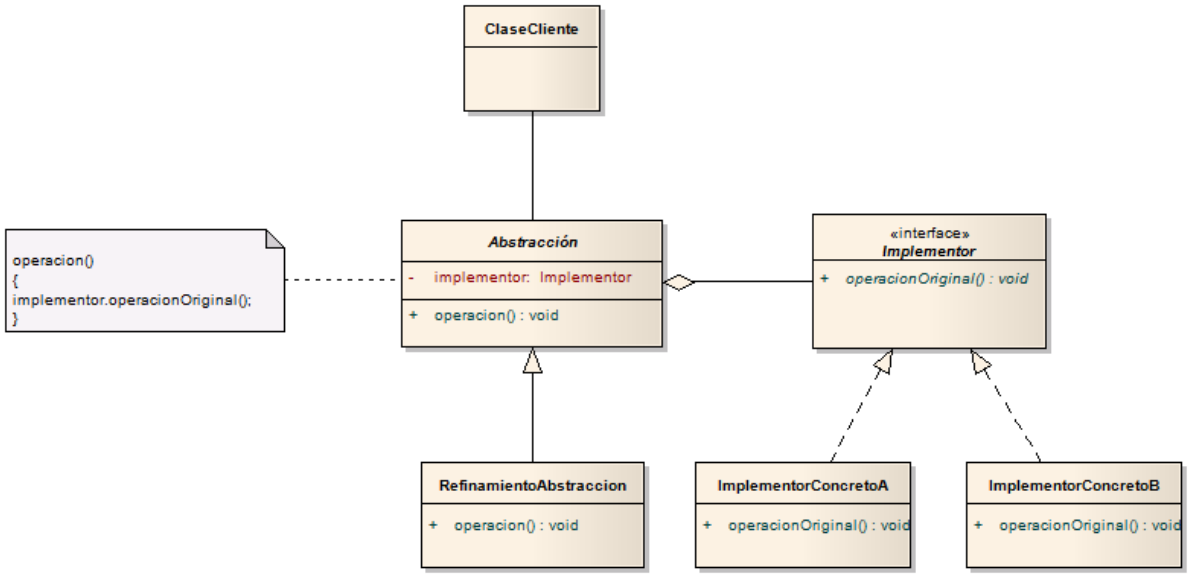
También conocido por:
Handle/Body, Puente

Motivación:
El patrón Bridge aborda la necesidad de permitir que las abstracciones y sus implementaciones varíen independientemente, eliminando el acoplamiento rígido entre ellas. Esto se logra mediante una interfaz de puente que facilita la interacción sin dependencia directa, permitiendo cambios y extensiones en ambas partes sin afectar la otra.

Aplicabilidad:
El patrón Bridge se usa cuando:

- se necesita separar la lógica de negocio de la lógica de presentación, como por ejemplo, separar la lógica de una aplicación de la lógica de la interfaz de usuario.
- se necesita desacoplar una abstracción de su implementación
- se necesita mejorar la flexibilidad de un sistema, permitiendo que la abstracción y la implementación se cambien o se reemplacen sin afectar al otro.

Estructura:



Participantes:

- **Clase Cliente:** Representa el código que utiliza la abstracción, es decir, que llama a los métodos de la clase Abstracción.

- **Abstracción:** Define la interfaz de alto nivel para la funcionalidad. Tiene un atributo implementor que es una referencia a la clase Implementor.

- **Implementor:** Define la interfaz de bajo nivel para la implementación. Define la funcionalidad real que se utiliza para la abstracción.
- **RefinamientoAbstraccion:** Es una clase que hereda de Abstracción y proporciona una implementación más específica de la abstracción.

- **ImplementorConcretoA** e **ImplementorConcretoB:** Son clases concretas que implementan la interfaz Implementor y proporcionan implementaciones distintas de la funcionalidad.

Colaboraciones:

En el patrón Bridge, la ClaseCliente interactúa con la Abstracción, que proporciona una interfaz de alto nivel sin conocer la implementación concreta. La Abstracción delega la funcionalidad real a un objeto Implementor, una interfaz implementada por las clases ImplementorConcretoA y ImplementorConcretoB. La RefinamientoAbstraccion hereda de la Abstracción y proporciona una especialización de esta. Así, la Abstracción puede utilizar cualquier implementación concreta (ImplementorConcretoA o ImplementorConcretoB) para realizar su funcionalidad, lo que permite separar la abstracción de la implementación y lograr un código más flexible y mantenible.

Consecuencias:

- **Aumento de la abstracción:** El patrón Bridge puede llevar a un aumento de la abstracción en el diseño, lo que puede hacer que el código sea más difícil de entender y mantener.
- **Mayor uso de recursos:** La creación de varias clases y la relación entre ellas puede requerir más recursos del sistema, lo que puede afectar el rendimiento.

Implementacion:

La implementación del patrón Bridge implica definir una interfaz Abstracción que declara los métodos que se van a utilizar por la clase cliente, y una interfaz Implementor que define los métodos que se van a utilizar por la abstracción para delegar la implementación. Luego, se crea una clase Abstracción concreta que implementa la interfaz Abstracción y tiene un atributo de tipo Implementor, y se crean clases Implementor concretas que implementan la interfaz Implementor.

Los clientes del patrón Bridge pueden utilizar la clase Abstracción concreta para acceder a la implementación concreta, sin necesidad de conocer los detalles de la implementación. De esta forma, se puede cambiar la implementación concreta sin afectar a la clase cliente, y se puede agregar nuevas implementaciones concretas sin modificar la clase Abstracción concreta.

Código Ejemplo:

```
public abstract class AbstraccionFiguras {
   public abstract void dibujarFigura();
   public abstract void setImplementor(ImplementorTamaños
implementor);
```

```
public class CuadradoConcreto extends
AbstraccionFiguras{
   private ImplementorTamaños implementor;
   public CuadradoConcreto(ImplementorTamaños
implementor){
      this.implementor = implementor;
   }
   public void setImplementor(ImplementorTamaños
implementor) {
      this.implementor = implementor;
   }
   @Override
   public void dibujarFigura() {
      implementor.dibujarPorTamaño("cuadrado");
```

```
public class TrianguloConcreto extends
AbstraccionFiguras{
   private ImplementorTamaños implementor;
   public TrianguloConcreto(ImplementorTamaños
implementor){
      this.implementor = implementor;
   }
   public void setImplementor(ImplementorTamaños
implementor) {
      this.implementor = implementor;
   }
   @Override
   public void dibujarFigura() {
      implementor.dibujarPorTamaño("Triangulo");
```

```
public abstract class ImplementorTamaños {
   public abstract void dibujarPorTamaño(String
Figura);
```

```
public class ImplementorGrande extends
ImplementorTamaños{
   @Override
   public void dibujarPorTamaño(String figura) {
      System.out.println("Imprimiendo un "+ figura + "
grande");
```

```
public class ImplementorPequeño extends
ImplementorTamaños{

  @Override

  public void dibujarPorTamaño(String Figura) {

    System.out.println("Imprimiendo un " + Figura + "
pequeño");
```

```
public class Main {

  public static void main(String[] args) {

    ImplementorTamaños implementorGrande = new ImplementorGrande();

    AbstraccionFiguras cuadrado = new
CuadradoConcreto(implementorGrande);

    AbstraccionFiguras triangulo = new
TrianguloConcreto(implementorGrande);

    cuadrado.dibujarFigura();

    triangulo.dibujarFigura();

    ImplementorTamaños implementorPequeño = new ImplementorPequeño();

    cuadrado.setImplementor(implementorPequeño);

    cuadrado.dibujarFigura();

    triangulo.setImplementor(implementorPequeño);

    triangulo.dibujarFigura();
```

Usos Conocidos:

En sistemas de bases de datos y En sistemas de archivos es frecuentemente utilizado.

Patrones relacionados:

Adapter y Composite.

3. Referencias Bibliográficas

[1] Bridge. (s.f.-a). Reactive Programming - arquitectura y desarrollo de software. https://reactiveprogramming.io/blog/es/patrones-de-diseno/bridge

[2] Bridge. (s.f.-b). Refactoring and Design Patterns. https://refactoring.guru/es/design-patterns/bridge

[3] Patrones estructurales (IV): Patrón bridge. (s.f.). Let's code something up! https://danielggarcia.wordpress.com/2014/03/17/patrones-estructurales-iv-patron-bridge/