Bridge

Nombre:
Bridge

Clasificación del patrón:
Estructural

Intención:
Desacoplar una abstracción de su implementación para que ambas puedan variar independientemente.

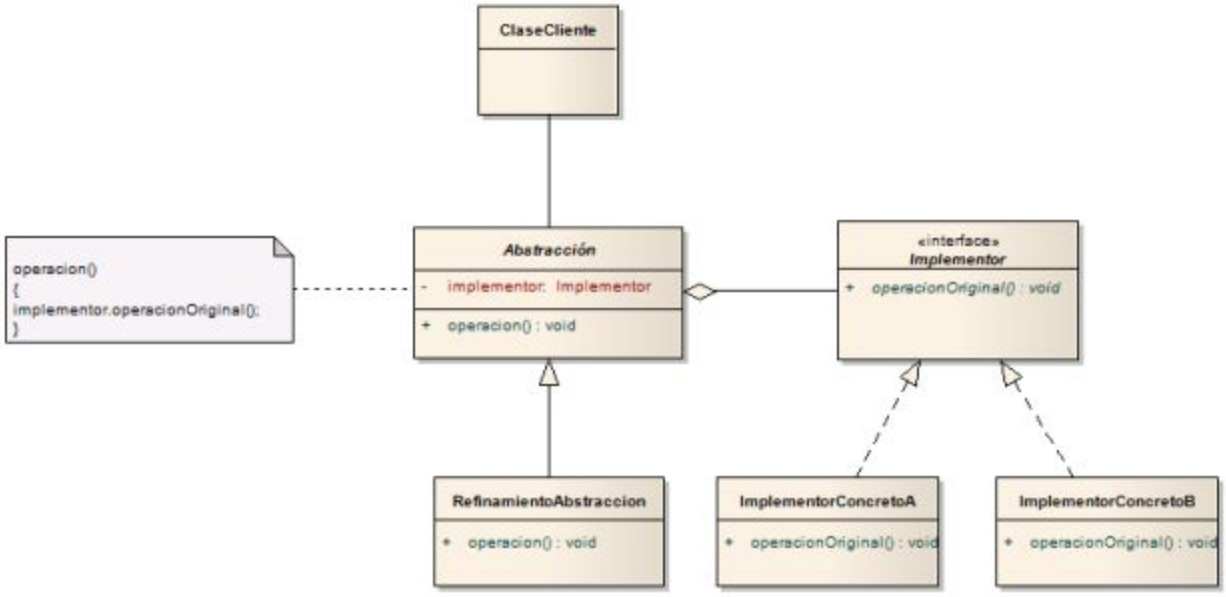
Otros nombres:
Handle/Body, Puente, Manejar/Cuerpo.

Motivación:
Cuando una abstracción puede tener varias implementaciones, comúnmente se hace uso de la herencia para acomodarla a las diferentes necesidades. Lo anterior se hace muy poco flexible, difícil de mantener y modificar, por lo tanto es necesario que un patrón solucione el problema.

Aplicabilidad:
Se debe usar cuando se quiere:

- Evitar un permanente acoplamiento entre una abstracción y su implementación, en especial si la implementación tiene que ser cambiada a un runtime.
- Extender la abstracción y su implementación de manera independiente.
- Evitar impactos negativos o recompilación total en el cliente cuando la implementación de una abstracción es cambiada.
- Evita la proliferación de clases en el código.

Estructura:



Participantes:

- Abstracción: Define una interfaz abstracta. Mantiene la referencia al Implementador.
- Abstracción Refinada: Extiende la interfaz definida por la abstracción.
- Implementador: Define la interface para la implementación de clases. Típicamente, la interface Implementador define operaciones primitivas mientras que Abstracción define operaciones de alto nivel con base en las primitivas.
- Implementador Concreto: Implementa la interface Implementador y define su concreta implementación.

Colaboraciones:

Abstracción emite los pedidos de los clientes a su objeto Implementador.

Ventajas:

- La abstracción y su implementación son desacopladas. Por ello, la abstracción puede ser configurada como un runtime con diferentes implementaciones.
- Permite un tratamiento en capas que permite un enfoque más estructurado al sistema. Por ejemplo al cambiar una implementación no requiere la recompilación de la abstracción y su cliente.
- Es fácil extender su una abstracción y su implementación de manera independiente.
- Es posible ocultar los detalles de la implementación de una abstracción del cliente.

Desventajas:

- Crea una indirección que puede afectar el rendimiento del sistema.

Implementación:

- Solo el Implementador: Cuando exista solo una implementación no se hace necesario crear una clase Implementador abstracta. Es un caso especial del patrón y es muy útil cuando un cambio en la implementación de una clase no debe afectar a los clientes existentes.
- Creando el objeto implementador adecuado: Si una Abstracción conoce todas las clases ImplementadorConcreto puede decidir cual instanciar dependiendo de los parámetros del constructor. También es posible cambiar una implementación inicial dependiendo el uso, o es posible delegar la decisión a otro objeto.
- Compartiendo implementadores: Handle/Body en C++ se puede usar para compartir implementaciones de muchos objetos. Body almacena una cuenta de referencia que la clase Handle incrementa y decrementa.
- Usando herencia múltiple: Se puede usar para asociar una interfaz con su implementación.

Código de ejemplo:

- Implementor, motor:

```
1  // Implementor
2  public interface IMotor
3  {
4      void InyectarCombustible(double cantidad);
5      void ConsumirCombustible();
6  }
```

- Diesel:

```
// ImplementorConcretoA
public class Diesel : IMotor
{
    #region IMotor Members

    public void InyectarCombustible(double cantidad)
    {
        Console.WriteLine("Inyectando " + cantidad + " ml. de Gasoil");
    }

    public void ConsumirCombustible()
    {
        RealizarExplosion();
    }

    #endregion

    private void RealizarExplosion()
    {
        Console.WriteLine("Realizada la explosión del Gasoil");
    }
}
```

- Gasolina:

```
// ImplementorConcretoB
public class Gasolina : IMotor
{
    #region IMotor Members

    public void InyectarCombustible(double cantidad)
    {
        Console.WriteLine("Inyectando " + cantidad + " ml. de Gasolina");
    }

    public void ConsumirCombustible()
    {
        RealizarCombustion();
    }

    #endregion

    private void RealizarCombustion()
    {
        Console.WriteLine("Realizada la combustión de la Gasolina");
    }
}
```

- Vehículo:

```
// Abstracción
public abstract class Vehiculo
{
    private IMotor motor;

    public Vehiculo(IMotor motor)
    {
        this.motor = motor;
    }

    // Encapsulamos la funcionalidad de la interfaz IMotor
    public void Acelerar(double combustible)
    {
        motor.InyectarCombustible(combustible);
        motor.ConsumirCombustible();
    }

    public void Frenar()
    {
        Console.WriteLine("El vehículo está frenando.");
    }

    // Método abstracto
    public abstract void MostrarCaracteristicas();
}
```

- Berlina:

```
// RefinamientoAbstraccionA
public class Berlina : Vehiculo
{
    // Atributo propio
    private int capacidadMaletero;

    // La implementacion de los vehículos se desarrolla de forma independiente
    public Berlina(IMotor motor, int capacidadMaletero) : base(motor)
    {
        this.capacidadMaletero = capacidadMaletero;
    }

    // Implementación del método abstracto
    public override void MostrarCaracteristicas()
    {
        Console.WriteLine("Vehiculo de tipo Berlina con un maletero con una capacidad de " +
            capacidadMaletero + " litros.");
    }
}
```

- Mono volúmen:

```
public class Monovolumen : Vehiculo
{
    // Atributo propio
    private bool puertaCorrediza;

    // La implementacion de los vehículos se desarrolla de forma independiente
    public Monovolumen(IMotor motor, bool puertaCorrediza)
        : base(motor)
    {
        this.puertaCorrediza = puertaCorrediza;
    }

    // Implementación del método abstracto
    public override void MostrarCaracteristicas()
    {
        Console.WriteLine("Vehiculo de tipo Berlina " + (puertaCorrediza ? "con" : "sin") +
            " puerta corrediza.");
    }
}
```

Usos conocidos:

- DOM y KDE3.
- ET++, framework de aplicaciones.

Patrones relacionados:

- Adapter
- Abstract Factory

Bibliografía:

No específico. (No específico). GoF Design Patterns (Versión 2.1.0) [Aplicación móvil]. Descargado de: https://drive.google.com/file/d/0BywiVyFlIabXcVhGZlJBcnhWTkU/view.

García, D. PATRONES ESTRUCTURALES (IV): PATRÓN BRIDGE. (Marzo 2014). PATRONES ESTRUCTURALES (IV): PATRÓN BRIDGE. Consultado en: https://danielggarcia.wordpress.com/2014/03/17/patrones-estructurales-iv-patron-bridge/comment-page-1/.

Junta de Andalucía. (s.f). Puente. Marco de Desarrollo de la Junta de Andalucía. http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/201.