

# Metodología de Diseño - Problema 2

## Parte 2: Conociendo patrones de diseño GOF

### Objetivo

Aplicar patrones para resolver problemas de diseño

## 1. Patrones de Diseño GOF

### ¿Qué son?

El patrón de diseño Estrategia es uno de los patrones de diseño conocidos como GoF (de Gang of Four, por sus cuatro creadores: Gamma, Helm, Johnson y Vlissides). Son patrones de **diseño orientado a objetos**, que encapsulan soluciones a problemas comunes de diseño.

Existen tres tipos de patrones de diseño GoF, que resuelven tres tipos de problema distintos:

- **Creacionales**, que resuelven problemas de creación de objetos, como por ejemplo, ¿cómo crear un objeto si no sé su clase? ¿cómo crear un objeto idéntico a otro? ¿cómo crear un objeto de la misma clase de otro, pero sin saber su clase?
- **Estructurales**, que resuelven problemas para organizar objetos en estructuras dinámicas, recursivas o en contextos de ambigüedad, como por ejemplo ¿Cómo represento un árbol?, ¿cómo le puedo agregar atributos a una clase en tiempo de ejecución? ¿cómo represento una estructura recursiva de todo y partes?
- **De Comportamiento**, que resuelven problemas de comportamiento dinámico o en condiciones de ambigüedad, como por ejemplo ¿Cómo hago que un objeto cambie su comportamiento para un mismo método en tiempo de ejecución? (Como el patrón estrategia)

### Revisemos algunos

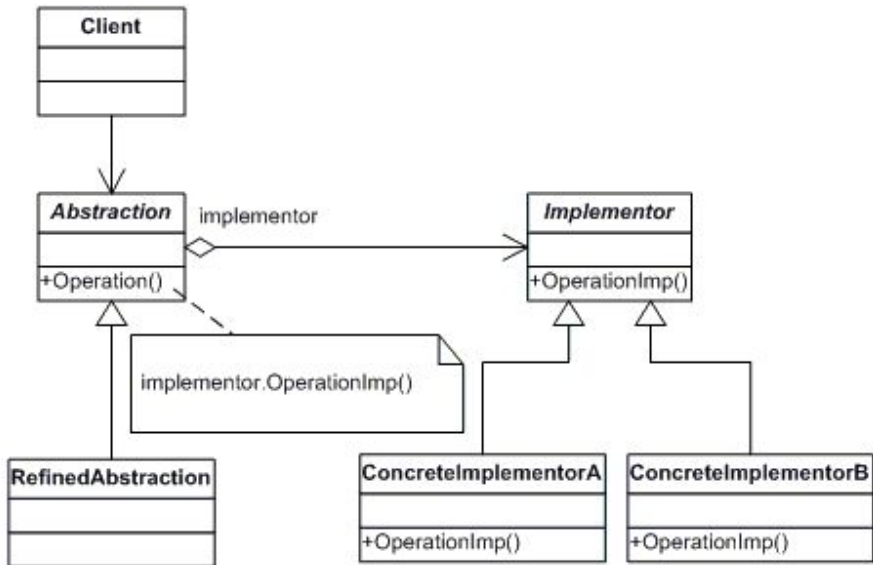
En la URL siguiente, encontrará ejemplos de los patrones de diseño GoF en C#. Además están clasificados con un ranking de popularidad.

<http://dofactory.com/net/design-patterns>

Revise 6 de los más populares, dos por cada tipo de patrón.

## 2. Actividad

1.- Invente un requerimiento para nuestro juego Extreme Fighter, que para ser resuelto requiera utilizar alguno de los patrones de diseño estudiados. Siga el ejemplo de a continuación.

<b>Requerimiento</b>	Como cada tipo de personaje tiene un poder especial distinto, el diseño debería permitir que independiente del nombre del método del súper poder, se llamara a un método de superpoder genérico para todos los luchadores.
<b>Problema</b>	Cada luchador tiene un nombre de método distinto para su super poder.
<b>Solución</b>	Crear una abstracción para el método, separando su definición de su implementación
<b>Patrón a aplicar</b>	Bridge <a href="http://dofactory.com/net/bridge-design-pattern">http://dofactory.com/net/bridge-design-pattern</a>
<b>Diagrama UML</b>	 <pre> classDiagram     class Client     class Abstraction {         +Operation()     }     class RefinedAbstraction     class Implementor {         +OperationImp()     }     class ConcreteImplementorA {         +OperationImp()     }     class ConcreteImplementorB {         +OperationImp()     }      Client --&gt; Abstraction     Abstraction &lt; -- RefinedAbstraction     Abstraction o--&gt; Implementor : implementor     Implementor &lt; -- ConcreteImplementorA     Implementor &lt; -- ConcreteImplementorB     RefinedAbstraction ..&gt; Implementor : implementor.OperationImp()             </pre> <p>The diagram illustrates the Bridge Design Pattern. It features a <b>Client</b> that depends on an <b>Abstraction</b> interface. The <b>Abstraction</b> interface defines a <code>+Operation()</code> method. A <b>RefinedAbstraction</b> class inherits from <b>Abstraction</b>. An <b>Implementor</b> interface defines a <code>+OperationImp()</code> method. Two concrete classes, <b>ConcreteImplementorA</b> and <b>ConcreteImplementorB</b>, inherit from <b>Implementor</b>. The <b>Abstraction</b> interface has an aggregation relationship with the <b>Implementor</b> interface, labeled "implementor". The <b>RefinedAbstraction</b> class has a dependency on the <b>Implementor</b> interface, with a note indicating it calls <code>implementor.OperationImp()</code>.</p>

Presente sus hallazgos al curso, en el formato solicitado, indicando las principales dificultades encontradas.

**RNL-EGL/20191S**