Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Lic. En Seguridad de Tecnologías de la Información

Patrones de diseño

Alumno: Roberto Iván Hernández Chavarría

Matricula: 1452359

Materia: Diseño Orientado a Objetos

Profesor: Lic. Miguel Ángel Salazar Santillán

Patrones de diseño

Los Patrones de diseño son herramientas para solucionar problemas de Diseño enfocados al desarrollo de software, estos patrones deben ser reusables permitiendo así que sean adaptados a diferentes problemáticas.

Cuando hablamos de problemáticas nos referimos a condiciones o problemas reiterativos en el desarrollo de software donde la solución se encuentra identificada mediante la aplicación de una serie de pasos, adaptando el sistema a una estructura definida por un patrón, garantizando que esta solución pueda ser aplicada cuantas veces sea necesario en circunstancias similares.

Existen 3 tipos generales de patrones de diseño, teniendo cada uno de ellos varios tipos:

- **Creacionales**: Giran en torno a la creación de Objetos

✓ Object Pool (no pertenece a los patrones especificados por GoF): se obtienen objetos nuevos a través de la clonación. Utilizado cuando el costo de crear una clase es mayor que el de clonarla. Especialmente con objetos muy complejos. Se especifica un tipo de objeto a crear y se utiliza una interfaz del prototipo para crear un nuevo objeto por clonación. El proceso de clonación se inicia instanciando un tipo de objeto de la clase que queremos clonar.

✓ Abstract Factory (fábrica abstracta): permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo, la creación de interfaces gráficas de distintos tipos (ventana, menú, botón, etc.).

✓ Builder (constructor virtual): abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.

✓ Factory Method (método de fabricación): centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística, es decir, la diversidad de casos particulares que se pueden prever, para elegir el subtipo que crear. Parte del principio de que las subclases determinan la clase a implementar.

✓ Prototype (prototipo): crea nuevos objetos clonándolos de una instancia ya existente.

✓ Singleton (instancia única): garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Restringe la instanciación de una clase o valor de un tipo a un solo objeto. A continuación se muestra un ejemplo de este patrón:

✓ Model View Controller (MVC) Es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Este patrón plantea la separación del problema en tres capas: la capa **model**, que representa la realidad; la capa **controller ,** que conoce los métodos y atributos del modelo, recibe y realiza lo que el usuario quiere hacer; y la capa **vista**, que muestra un aspecto del modelo y es utilizada por la capa anterior para interactuar con el usuario.

- **Estructurales**: Se enfocan en la estructura de clases y objetos que las componen

✓ Adapter o Wrapper (Adaptador o Envoltorio): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.

✓ Bridge (Puente): Desacopla una abstracción de su implementación.

- ✓ Composite (Objeto compuesto): Permite tratar objetos compuestos como si de uno simple se tratase.
- ✓ Decorator (Decorador): Añade funcionalidad a una clase dinámicamente.
- ✓ Facade (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- ✓ Flyweight (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
- ✓ Proxy: Proporciona un intermediario de un objeto para controlar su acceso.
- ✓ Module: Agrupa varios elementos relacionados, como clases, singletons, y métodos, utilizados globalmente, en una entidad única.

- **De Comportamiento**: Definen el modo en que las clases y objetos son relacionados, el comportamiento he interacción entre ellos.

- ✓ Chain of Responsibility (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- ✓ Command (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- ✓ Interpreter (Intérprete): Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- ✓ Iterator (Iterador): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- ✓ Mediator (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- ✓ Memento (Recuerdo): Permite volver a estados anteriores del sistema.
- ✓ Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- ✓ State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- ✓ Strategy (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- ✓ Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- ✓ Visitor (Visitante): Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

Existen varias maneras de documentar los patrones de diseño, pero todas se basan en plantillas, las cuales contienen varios elementos, como nombre, descripción del problema y solución, algunas contienen muchas más detalles que otras, dependiendo del tipo de patrón de diseño a documentar. Una de las más utilizadas es la llamada “GOF”, la cual contiene los siguientes elementos:

- Nombre del patrón: Ayuda a recordar la esencia del patrón.
- Clasificación: Los patrones originalmente definidos por GOF se clasifican en tres categorías "Creacional", "Estructural", "Comportamiento".
- Propósito / Problema: ¿Qué problema aborda el patrón?
- También conocido como . : Otros nombres comunes con los que es conocido el patrón.
- Motivación: Escenario que ilustra el problema.
- Aplicabilidad: Situaciones en las que el patrón puede ser utilizado.

- Estructura: Diagramas UML que representan las clases y objetos en el patrón.
- Participantes: Clases y/o objetos que participan en el patrón y responsabilidades de cada uno de ellos
- Colaboraciones: ¿Cómo trabajan en equipo los participantes para llevar a cabo sus responsabilidades?
- Consecuencias: ¿Cuáles son los beneficios y resultados de la aplicación del patrón?
- Implementación: Detalles a considerar en las implementaciones. Cuestiones específicas de cada lenguaje OO.
- Código de ejemplo
- Usos conocidos: Ejemplos del mundo real.
- Patrones relacionados: Comparación y discusión de patrones relacionados. Escenarios donde puede utilizarse en conjunción con otros patrones.

Ejemplos de patrones de diseño:

- Creacionales

Singleton

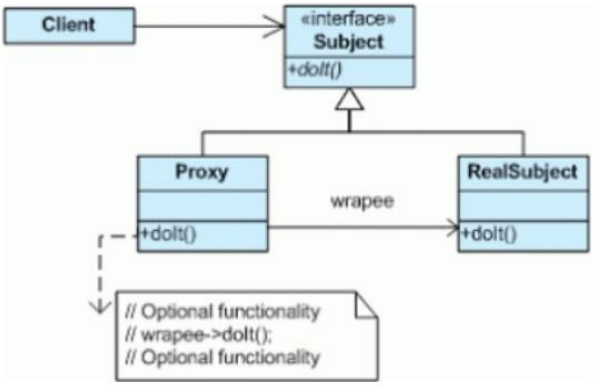
Restringe la instanciación de una clase o valor de un tipo a un solo objeto.

```
public sealed class Singleton
{
    private static volatile Singleton instance;
    private static object syncRoot = new Object();
    private Singleton()
    {
        System.Windows.Forms.MessageBox.Show("Nuevo Singleton");
    }
    public static Singleton GetInstance
    {
        get
        {
            if (instance == null)
            {
                lock(syncRoot)
                {
                    if (instance == null)
                        instance = new Singleton();
                }
            }
            return instance;
        }
    }
}
```

- Estructurales

Proxy

Proporciona un representante de otro objeto para controlar el acceso al objeto que está siendo representado.



- Comportamiento

Observador (*Observer*): Notificaciones de cambios de estado de un objeto.

```
Public Class Articulo
   Delegate Sub DelegadoCambiaPrecio(ByVal unPrecio As Object)
   Public Event CambiaPrecio As DelegadoCambiaPrecio
   Dim _cambiaPrecio As Object
   Public WriteOnly Property Precio()
      Set(ByVal value As Object)
         _cambiaPrecio = value
         RaiseEvent CambiaPrecio(_cambiaPrecio)
      End Set
   End Property
End Class
Public Class ArticuloObservador
   Public Sub Notify(ByVal unObjecto As Object)
      Console.WriteLine("El nuevo precio es:" & unObjecto)
   End Sub


End Class
```