UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN ®

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

DISEÑO ORIENTADO A OBJETOS

Assignment 6 “Patrones de diseño”

ALUMNO: SALVADOR RAFAEL GONZÁLEZ GARCÍA

MATRICULA: 1690539

GRUPO: 006

Patrones de diseño

¿Qué son?

Un patrón de diseño es una forma reutilizable de resolver un problema común, es decir, son soluciones para problemas típicos y recurrentes que nos podemos encontrar a la hora de desarrollar una aplicación.

Esto es porque, aunque nuestra aplicación sea única, tendrá partes comunes con otras aplicaciones: acceso a datos, creación de objetos, operaciones entre sistemas etc. Y, en lugar de reinventar algo que ya existe, podemos solucionar problemas utilizando algún patrón, ya que son soluciones probadas y documentadas por multitud de programadores.

Los patrones de diseño son muy útiles por los siguientes motivos:

1. Te ahorran tiempo; porque al conocer los patrones de diseño, contarás con un conjunto de herramientas que ya han sido probadas ayudándote a la solución de la mayor parte de tu problema, haciendo que la parte de desarrollo sea en más fácil ya que ganas tiempo en no buscar una solución a tu problema.

2. Te ayudan a estar seguro de la validez de tu código; esto es porque estos patrones ya pudieron ser probados por varios programadores a lo largo de los últimos años, documentado los resultados que se obtuvieron con la utilización de dichos patrones.

3. Establecen un lenguaje común; los patrones de diseño ayudan a los desarrolladores, a avanzar mucho más rápido, con un código más fácil de entender para todos y mucho más robusto.

¿Cómo se documentan?

Los patrones de diseño se documentan utilizando plantillas formales con unos campos estandarizados. Existen varias plantillas ampliamente aceptadas, aunque todas ellas deben al menos documentar las siguientes partes de un patrón:

- Nombre: describe el problema de diseño.

- El problema: describe cuándo aplicar el patrón.

- La solución: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.

La plantilla más utilizada es la plantilla GOF que consta de los siguientes campos:

- 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: ¿Cuales 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

Método de Fabricación

Parte del principio de que las subclases determinan la clase a implementar.

```
public class ConcreteCreator extends Creator{
        protected Product FactoryMethod(){
                return new ConcreteProduct();
        }
}

public interface Product{}
public class ConcreteProduct implements Product{}
        public class Client{
                public static void main(String args[]){
                        Creator UnCreator;
                        UnCreator = new ConcreteCreator();
                        UnCreator.AnOperations();
                }
        }
```

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;
                }
      }
    }
```

Patrones de comportamiento

```
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
```