UANL

FCFM

Doo Tarea 6

Adrián Campos Treviño

1547530

¿Qué son los patrones de diseño?

Los parones de diseño son una técnica para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacciono o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características.

Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Para qué sirven los patrones de diseño

Los patrones de diseño pretenden:

• Proporcionar catálogos de elementos reusables en el diseño de sistema software

• Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente

• Formalizar un vocabulario común entre diseñadores

• Estandarizar el modo en que se realiza el diseño

• Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensado conocimiento ya existente

Para que NO sirven los patrones de diseño

Los patrones de diseño NO pretenden:

• Imponer ciertas alternativas de diseño frente a otras

• Eliminar la creatividad inherente al proceso de diseño

¿Cómo se documenta un patrón de diseño?

• Nombre e intención del patrón

    • Referencia al patrón

    • Incrementa el vocabulario de diseño

• Problema y contexto

    • Cuando aplicar el patrón

- Solución
    - Estructura: elementos que conforman el diseño, sus relaciones, responsabilidades y colaboraciones
        - Es una descripción abstracta de como una disposición de elementos (clase y objetos) solucionan el problema
    - Se ilustra con un ejemplo de código
- Consecuencias (positivas y negativas)
    - Necesidades (tiempo, memoria), aspectos de implementación y lenguaje de programación, flexibilidad, extensibilidad, portabilidad
- Patrones relacionados

Categorías de patrones de diseño

- Patrones de creación
    - Tratan de la inicialización y configuración de clases y objetos
- Patrones estructurales
    - Tratan de desacoplar interfaz e implementación de clases y objetos
        - ¿Cómo se componen clases y objetos?
- Patrones de comportamiento
    - Tratan de las interacciones dinámicas entre sociedades de clases y objetos
        - ¿Cómo interaccionan y se distribuyen responsabilidades los objetos?

Ejemplos de patrones creacionales

Abstract Factory

El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces graficas de distintos tipos (ventana, menú, botón, etc.)

Factory Method

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();
                }
        }
```

Prototype

Se basa en la clonación de ejemplares copiándolos de un prototipo

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;
                }
        }
}
```