ADOO

Alumno: *López Manríquez Ángel*
Tema: *Patrones de diseño*
Fecha: *19 de junio de 2018*
Grupo: *2CM10*

M. en C. Cordero López Martha Rosa

Instituto Politecnico Nacional
Escuela Superior de Cómputo

Índice

Patrones de diseño

Lopez Manriquez Angel
Loretto Estrada Galilea América
Pérez García Atziri
2CM10

19 de junio de 2018

1. Que es un patrón de diseño

Los patrones de diseño son un tema importante para los programadores, ya que mediante estos ofrecen soluciones a problemas comunes y cotidianos a la hora de diseñar alguna aplicación.

Cuando se busca una definición acerca de que es un patrón de diseño, la mas adecuada es la siguiente: "Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software."

En otras palabras, estas herramientas nos brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

La manera de utilizar cada patron depende de factores como el hecho de comprender correctamente en que momento pueden usarse, y teniendolos presentes en el momento del diseño, esto es ser capaz de relacionar el problema con el patron de diseño que mejor lo resuelva.

Existen varios patrones de diseño popularmente conocidos, los cuales se clasifican como se muestra a continuación:

- Patrones Creacionales: Inicialización y configuración de objetos.
- Patrones Estructurales: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
- Patrones de Comportamiento: Más que describir objetos o clases, describen la comunicación entre ellos.

Veamos un poco en qué consisten los distintos tipos de patrones, cuáles son sus fines y qué beneficios nos aportan.

2. Patrones Creacionales

2.1. Fábrica Abstracta (Abstract Factory)

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.).

2.2. Método de Fabricación (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();
            }
      }
```

3. Prototipado (Prototype)

Se basa en la clonación de ejemplares copiándolos de un prototipo.

4. Singleton

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

```
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();
            }
      }
```

5. MVC (Model View Controler)

Este patrón plantea la separación del problema en tres capas: la capa model, que representa la realidad; la capa controler , 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 interaccionar con el usuario. Principio de la página

6. Patrones Estructurales

- Adaptador (Adapter): Convierte una interfaz en otra.
- Puente (Bridge): Desacopla una abstracción de su implementación permitiendo modificarlas independientemente.
- Objeto Compuesto (Composite): Utilizado para construir objetos complejos a partir de otros más simples, utilizando para ello la composición recursiva y una estructura de árbol.
- Envoltorio (Decorator): Permite añadir dinámicamente funcionalidad a una clase existente, evitando heredar sucesivas clases para incorporar la nueva funcionalidad.
- Fachada (Facade): Permite simplificar la interfaz para un subsistema.
- Peso Ligero (Flyweight): Elimina la redundancia o la reduce cuando tenemos gran cantidad de objetos con información idéntica.
- Apoderado (Proxy): Un objeto se aproxima a otro.

7. Patrones de Comportamiento

- Cadena de responsabilidad (Chain of responsibility): La base es permitir que más de un objeto tenga la posibilidad de atender una petición.
- Orden (Command): Encapsula una petición como un objeto dando la posibilidad de "deshacer" la petición.
- Intérprete (Interpreter): Intérprete de lenguaje para una gramática simple y sencilla.
- Iterador (Iterator): Define una interfaz que declara los métodos necesarios para acceder secuencialmente a una colección de objetos sin exponer su estructura interna.
- Mediador (Mediator): Coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
- Recuerdo (Memento): Almacena el estado de un objeto y lo restaura posteriormente.
- Observador (Observer): Notificaciones de cambios de estado de un objeto.

```
1  public class ConcreteCreator extends Creator
2  {
3    protected Product FactoryMethod()
4        {
5              return new ConcreteProduct();
6        }
7  }
8  public interface Product{}
9  public class ConcreteProduct implements Product{}
10       public class Client
11       {
12             public static void main(String args[])
13             {
14                   Creator UnCreator;
15                   UnCreator = new ConcreteCreator();
16                   UnCreator.AnOperations();
17             }
18       }
```

- Estado (Server): Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.
- Estrategia (Strategy): Utilizado para manejar la selección de un algoritmo.
- Método plantilla (Template Method): Algoritmo con varios pasos suministrados por una clase derivada.
- Visitante (Visitor): Operaciones aplicadas a elementos de una estructura de objetos heterogénea.

8. Conclusión

Como verán, si es que este artículo logró ilustrar sobre las distintas categorías y tipos de patrones de diseño, no debemos "reinventar la rueda" en varias de nuestras aplicaciones. Hay mucho trabajo ya realizado, testeado y aceptado que en un entorno similar a mi problema ya aporta una solución satisfactoria.