PATRONES DE DISEÑO DE SOFTWARE

INTRUCTORÍAS DE “INGENIERÍA DE SOFTWARE”

CATEDRÁTICO: | ING. MILAGRO ALICIA DE REYES| 30-07-2018

INSTRUCTOR: | BACHILLER RAÚL MAURICIO PORTILLO MUÑOZ | 30-07-2018

INDICE

Contenido

INTRODUCCIÓN

Si se busca la forma de solucionar un problema de "Desarrollo" se puede extraer, explicar y reutilizar en múltiples ámbitos, para que así entonces podamos encontrarnos ante un patrón de diseño de software. Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción 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.

El concepto de patrón de diseño lleva existiendo desde finales de los 70, pero su verdadera popularización surgió en los 90 con el lanzamiento del libro de Design Pattern (Patrón de Diseño).

OBJETIVOS

OBJETIVO GENERAL**:** Conocer sobre los diferentes Patrones de Diseño de Software y sus utilidades para solucionar problemas de desarrollo.

OBJETIVOS ESPECÍFICOS**:**

- Comprender que es un Patrón de Diseño de Software.
- Identificar en qué casos es correcto aplicar un Patrón de Desarrollo, en los proyectos de Desarrollo.
- Aplicar Patrones de Desarrollo en Sistemas CRM.
- Emplear Patrón de Desarrollo MVC.

DESARROLLO

¿Qué es un patrón de diseño de software?

"Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software." En otras palabras, 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).

Objetivos de los patrones:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas 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 condensando conocimiento ya existente.

Categorías de patrones según la escala o nivel de abstracción:

- Patrones de arquitectura: Aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.
- Patrones de diseño: Aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.
- Dialectos: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

PATRONES DE DISEÑO

Este es un pequeño listado de los patrones de diseño de software más conocidos. Los patrones se dividen en distintos grupos según el tipo de problema que resuelven, a saber:

Patrones creacionales.

Son los que facilitan la tarea de creación de nuevos objetos, de tal forma que el proceso de creación pueda ser desacoplado de la implementación del resto del sistema.
Los patrones creacionales están basados en dos conceptos:

- Encapsular el conocimiento acerca de los tipos concretos que nuestro sistema utiliza. Estos patrones normalmente trabajarán con interfaces, por lo que la implementación concreta que utilicemos queda aislada.
- Ocultar cómo estas implementaciones concretas necesitan ser creadas y cómo se combinan entre sí.

Los patrones creacionales más conocidos son:

- **Abstract Factory:** Nos provee una interfaz que delega la creación de un conjunto de objetos relacionados sin necesidad de especificar en ningún momento cuáles son las implementaciones concretas.
- **Factory Method:** Expone un método de creación, delegando en las subclases la implementación de este método.
- **Builder:** Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción nos puede servir para crear representaciones diferentes.
- **Singleton:** limita a uno el número de instancias posibles de una clase en nuestro programa, y proporciona un acceso global al mismo.
- **Prototype:** Permite la creación de objetos basados en "plantillas". Un nuevo objeto se crea a partir de la clonación de otro objeto.

Patrones estructurales.

Son patrones que nos facilitan la modelización de nuestro software especificando la forma en la que unas clases se relacionan con otras.
Estos son los patrones estructurales que definió la Gang of Four:

- **Adapter**: Permite a dos clases con diferentes interfaces trabajar entre ellas, a través de un objeto intermedio con el que se comunican e interactúan.
- **Bridge**: Desacopla una abstracción de su implementación, para que las dos puedan evolucionar de forma independiente.
- **Composite**: Facilita la creación de estructuras de objetos en árbol, donde todos los elementos emplean una misma interfaz. Cada uno de ellos puede a su vez contener un listado de esos objetos, o ser el último de esa rama.
- **Decorator**: Permite añadir funcionalidad extra a un objeto (de forma dinámica o estática) sin modificar el comportamiento del resto de objetos del mismo tipo.
- **Facade**: Una facade (o fachada) es un objeto que crea una interfaz simplificada para tratar con otra parte del código más compleja, de tal forma que simplifica y aísla su uso. Un ejemplo podría ser crear una fachada para tratar con una clase de una librería externa.
- **Flyweight**: Una gran cantidad de objetos comparte un mismo objeto con propiedades comunes con el fin de ahorrar memoria.
- **Proxy**: Es una clase que funciona como interfaz hacia cualquier otra cosa: una conexión a Internet, un archivo en disco o cualquier otro recurso que sea costoso o imposible de duplicar.

Patrones de comportamiento.

En este último grupo se encuentran la mayoría de los patrones, y se usan para gestionar algoritmos, relaciones y responsabilidades entre objetos.
Los patrones de comportamiento son:

- **Command**: Son objetos que encapsulan una acción y los parámetros que necesitan para ejecutarse.

- **Chain of responsibility**: Se evita acoplar al emisor y receptor de una petición dando la posibilidad a varios receptores de consumirlo. Cada receptor tiene la opción de consumir esa petición o pasárselo al siguiente dentro de la cadena.
- **Interpreter**: Define una representación para una gramática, así como el mecanismo para evaluarla. El árbol de sintaxis del lenguaje se suele modelar mediante el patrón Composite.
- **Iterator**: Se utiliza para poder movernos por los elementos de un conjunto de forma secuencial sin necesidad de exponer su implementación específica.
- **Mediator**: Objeto que encapsula cómo otros conjuntos de objetos interactúan y se comunican entre sí.
- **Memento**: Este patrón otorga la capacidad de restaurar un objeto a un estado anterior
- **Observer**: Los objetos son capaces de suscribirse a una serie de eventos que otro objetivo va a emitir, y serán avisados cuando esto ocurra.
- **State**: Permite modificar la forma en que un objeto se comporta en tiempo de ejecución, basándose en su estado interno.
- **Strategy**: Permite la selección del algoritmo que ejecuta cierta acción en tiempo de ejecución.
- **Template Method**: Especifica el esqueleto de un algoritmo, permitiendo a las subclases definir cómo implementan el comportamiento real.
- **Visitor**: Permite separar el algoritmo de la estructura de datos que se utilizará para ejecutarlo. De esta forma se pueden añadir nuevas operaciones a estas estructuras sin necesidad de modificarlas.

PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos es un paradigma de programación que utiliza la abstracción para crear modelos basados en el mundo real.
Utiliza diversas técnicas de paradigmas previamente establecidas, incluyendo la modularidad**,** polimorfismo **y** encapsulamiento. Para poder aplicar correctamente los Patrones de Diseño de Software, se deben recordar las bases de POO, ya que es esencial para la creación y estructuración de scripts que apliquen dichos modelos, para lo cual es necesario mencionar unas palabras reservadas que nos servirán para la efectiva aplicación de estos patrones.

Terminología:

- **Clase:** Define las características del Objeto.
- **Objeto:** Una instancia de una Clase.
- **Propiedad:** Una característica del Objeto, como el color.
- **Método:** Una capacidad del Objeto, como caminar.
- **Constructor:** Es un método llamado en el momento de la creación de instancias.
- **Herencia:** Una Clase puede heredar características de otra Clase.
- **Encapsulamiento:** Una Clase sólo define las características del Objeto, un Método sólo define cómo se ejecuta el Método.
- **Abstracción:** La conjunción de herencia compleja, métodos y propiedades que un objeto debe ser capaz de simular en un modelo de la realidad.
- **Polimorfismo:** Diferentes Clases podrían definir el mismo método o propiedad.

Paquetes, clases y objetos.

A estas alturas el término “Clase” y “paquete” ya debería de sonar familiar. Todos los programas anteriores se han programado de una forma sencilla, todo el código dentro de una clase, pero este no es la forma de programar con un paradigma orientado a objetos. Para introducirnos en el diseño de clases en el lenguaje Java es importante conocer las definiciones del paradigma orientado a objetos.

Paquete.

Los paquetes no son más que una carpeta que ayudan a organizar las clases en grupos para facilitar el acceso a ellas cuando las necesitamos en algún programa, reducen los conflictos entre los nombres de las clases en caso de que se llamen igual (Pero de paquetes diferentes) y permiten proteger el acceso a las clases.

Para referirnos a una clase de un paquete, tenemos que hacer anteponiendo el nombre de la misma en el nombre de su paquete, exceptuando que el paquete haya sido importado explícitamente.

Clase.

Una clase es un tipo definido por el usuario (Véase como un archivo con extensión .java) en el cual se especifican los atributos y métodos de los objetos que se crearan a partir de la misma. Los atributos definen el estado de un determinado objeto y los métodos son operaciones que definen su comportamiento. Los atributos y los métodos forman parte general de una clase.
Como hemos visto anteriormente la forma de crear una clase es mediante dos partes: El nombre de la clase precedida por la palabra reservada class, y el cuerpo de la clase encerrado entre llaves.

```
class nombreClase{
    //cuerpo de la clase
}
```

Se puede definir un modificador de acceso al crear la clase, public. Por ejemplo:

```
public class nombreClase{
    //cuerpo de la clase
}
```

Una clase declarada pública indica que podemos utilizar la clase desde cualquier lado, es decir, una clase externa, dentro o fuera del mismo paquete, también se puede utilizar desde un paquete externo (siempre y cuando se haya importado el paquete).

Una clase sin modificador de acceso indica que la clase puede ser accedida por cualquier clase interna del mismo. No puede ser utilizada por cualquier otro paquete.

Nota: Una clase no se puede crear con los modificadores private ni protected.

Atributos.

Los atributos constituyen la estructura interna de los objetos de una clase. Para declarar un atributo se hace de la misma manera que al declarar una variable cualquiera, exceptuando por que se tiene que especificar el modificador de acceso. Por ejemplo:

```
public class Ejemplo{
    public String cadena;
    private int numero;
}
```

Nota: En una clase, cada atributo debe tener un nombre único, sin embargo, se puede utilizar un mismo nombre de atributo en otra clase dentro del mismo paquete, ya que cada clase es distinta, y por lo tanto cada clase tiene su propia estructura.

Es posible asignar algún valor al declarar un atributo, este será el valor inicial de un atributo, aunque no se debería hacer ya que es trabajo de un constructor. Ejemplo:

```
public class Ejemplo{
    public String cadena=”Hola”;
    private int numero=1;
}
```

También podemos declarar como atributos de una clase, referencias a objetos de otra clase existente. Por ejemplo, se define un objeto de la clase creada anteriormente.

```
public class EjemploDos{
    public Ejemplo ejem;

}
```

Métodos de una clase.

Los métodos definen las operaciones que se pueden realizar con los atributos de una clase. En vista de la POO el conjunto de todos estos métodos corresponde con el conjunto de mensajes a los que los objetos de una clase pueden responder, es decir, para la comunicación entre un objeto se utilizan métodos y estos a su vez se comunican mediante mensajes.

Para definir un método de una clase se hace de la misma manera que nuestro método main (Main es el método principal de una clase y es el primer método que se ejecuta al compilar un programa). Ejemplo:

```
public class Ejemplo{
    public String cadena=”Hola”;
    private int numero=1;

    public void main(String args[]){
        //Algun código
    }

     public void saludo(){
        System.out.println(“Este es el saludo de un médoto”);
     }
}
```

Modificadores de acceso en atributos y métodos.

Las clases básicamente incluye la idea de ocultar datos, consiste en que no se pueda acceder directamente a los atributos o métodos de un objeto, sino que hay que hacerlo mediante los métodos de una clase. Al hacer esto, el usuario de la clase sólo tendrá acceso a uno a más métodos que le permitirán acceder a los miembros privados.

Para controlar el acceso a los miembros de una clase se utilizan los modificadores private (privado), protected (protegido) y public (público) y predeterminado (omitiéndolas). Si observamos la clase anterior Ejemplo identificamos que se han declarado dos atributos, los cuales uno es público y otro privado, y un método que también es público.

Acceso predeterminado.

Es cuando se omite el tipo de acceso al declarar un atributo o método de la clase. Por ejemplo:

```
public class Ejemplo{
    String cadena=”Hola”;
    int numero=1;

    void saludo(){
        System.out.println(“Este es el saludo de un médoto”);
    }
}
```

El acceso predeterminado indica que el objeto de una clase puede ser accedido desde cualquier clase perteneciente al mismo paquete. Ninguna clase fuera de éste paquete puede tener acceso a estos miembros.

Acceso Público.

Un miembro declarado con el acceso público indica que puede ser accedido por un objeto de esa clase en cualquier parte de la aplicación, puede ser una clase del mismo paquete, una clase de un paquete externo.

```
public class Ejemplo{
    public String cadena=”Hola”;
    public int numero=1;

    public void saludo(){
        System.out.println(“Este es el saludo de un médoto”);
    }
}
```

Acceso Privado.

Un miembro de la clase declarado privado indica que el miembro solo puede ser accedido por un objeto de es mima clase, esto es, no puede ser accedido ni por otra clase del mismo paquete, ni por una clase externa al paquete.

```
public class Ejemplo{
    private String cadena=”Hola”;
    private int numero=1;

    private void saludo(){
        System.out.println(“Este es el saludo de un médoto”);
    }
}
```

Acceso Protegido.

Un miembro de la clase declarado como protegido se puede acceder a sus miembros otras clases que estén dentro del mismo paquete, sin embargo no pueden ser accedidos por alguna clase de un paquete externo.

```
public class Ejemplo{
    protected String cadena=”Hola”;
    protected int numero=1;

    protected void saludo(){
        System.out.println(“Este es el saludo de un médoto”);
    }

}
```

Definición de objeto.

Para entender mejor éste término pongámonos a pensar, en el mundo real los objetos se crean a partir de otros objetos, un objeto puede ser cualquier cosa, un automóvil, un árbol, una televisión, etc; y cada objeto tiene sus propios estados (Atributos) y comportamientos (Métodos). Por ejemplo, un automóvil tiene sus atributos como puede ser número de puertas, color, marca, modelo, etc. Y su comportamiento es el de arrancar, acelerar, frenarse, etc, estos conjuntos de acciones determinan a un objeto.

¿Cómo se crea un objeto?

Para eso debemos tener en cuenta lo que es una clase de objetos, pongamos un ejemplo. Piense en un molde para hacer galletas, el molde sería nuestra clase y las galletas nuestros objetos. De la misma manera se crean los objetos en la POO, a partir de un molde (clase) se crean los objetos, el molde debe de agrupar las propiedades comunes de todos los objetos, pero no todos los objetos tienen que ser iguales, es decir, no todas

las galletas deben tener la mima figura, o el mismo sabor. Esto es que cada objeto que construyamos de una clase tendrá sus propios datos.

Un objeto se crea en el momento en que se define una variable de dicha clase, por ejemplo:

```
Ejemplo objeto=new Ejemplo();
```

CONCLUSIÓN

Como lograste ver, nos encontramos ante una lista muy variada de patrones de diseño de software que nos dan la oportunidad de crear nuestros scripts de manera mucho más sencilla con estructuras probadas y que funcionan. La mayor complejidad radica en saber cuándo utilizarlas, algo que nos dará la práctica.