Actividad 1. Patrones de diseño de Software

ESTEFANIA AGUDELO BUSTAMANTE

DEARROLLO DE SOFTWARE

DILSA ENITH TRIANA MARTINEZ

PATRONES DE DISEÑO

03/03/2025

Introducción

Los patrones de diseño de software son soluciones probadas y reutilizables para problemas comunes en el desarrollo de aplicaciones. Estos patrones proporcionan una estructura clara para organizar el código y mejorar su mantenimiento. En este informe, se presentan tres patrones fundamentales: MVC (Modelo-Vista-Controlador), Observador y Singleton, junto con sus características y ejemplos en Java.

Desde mi perspectiva, estos patrones son esenciales para escribir código más eficiente y modular. Al comprender su propósito, podemos diseñar aplicaciones más escalables y fáciles de mantener, evitando la redundancia y promoviendo buenas prácticas de desarrollo.

Mapa Conceptual de los Patrones de Diseño

Patrón MVC (Modelo-Vista-Controlador)

Definición: Separa la lógica de negocio (modelo), la interfaz gráfica (vista) y la gestión de eventos (controlador). Esto facilita la organización del código y permite cambios en la interfaz sin afectar la lógica.

Considero que este patrón es fundamental en el desarrollo de aplicaciones con interfaces gráficas. Su estructura modular mejora la claridad y el mantenimiento del código.

Características:

- **Modelo:** Gestiona los datos y la lógica de negocio.
- **Vista:** Muestra la información al usuario.
- **Controlador:** Maneja las interacciones y actualiza la vista.

Ejemplo en Java:

```
class Modelo { String dato = "Hola, MVC"; }

class Vista { void mostrar(String d) { System.out.println(d); } }

class Controlador {
   Modelo m = new Modelo(); Vista v = new Vista();
   void actualizar() { v.mostrar(m.dato); }
}
```

Patrón Observador

Definición: Permite que un objeto (sujeto) notifique automáticamente a múltiples objetos (observadores) cuando su estado cambia. Se usa en eventos y notificaciones.

En mi opinión, este patrón es útil para sistemas en tiempo real y aplicaciones que requieren actualización dinámica, como notificaciones y cambios de estado en interfaces gráficas.

Características:

- **Sujeto:** Mantiene una lista de observadores.
- **Observadores:** Se actualizan cuando el sujeto cambia.
- **Uso:** Implementado en eventos, sistemas de notificación y programación reactiva.

Ejemplo en Java:

```
interface Observador { void actualizar(); }

class Sujeto {

   List<Observador> obs = new ArrayList<>();

   void agregar(Observador o) { obs.add(o); }

   void notificar() { obs.forEach(Observador::actualizar); }

}
```

Patrón Singleton

Definición: Garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a ella.

Veo este patrón como una forma eficiente de optimizar recursos en aplicaciones que requieren una única instancia de un objeto, como conexiones a bases de datos o registros de logs.

Características:

- **Única instancia:** Solo se crea un objeto de la clase.
- **Método de acceso estático:** Permite acceder a la instancia desde cualquier parte del programa.
- **Uso:** Común en gestión de recursos compartidos y configuración global.

Ejemplo en Java:

```
class Singleton {

   private static Singleton instancia = new Singleton();

   private Singleton() {}

   public static Singleton getInstancia() { return instancia; }
```

}

Conclusión

Comprender estos patrones ayuda a desarrollar software más estructurado y escalable. **MVC** permite modularidad en aplicaciones gráficas, **Observador** facilita la comunicación entre objetos y **Singleton** optimiza el uso de recursos.

Desde mi punto de vista, aplicar estos patrones correctamente mejora la eficiencia del código y evita problemas como la alta dependencia entre componentes o la duplicación de código. Su uso es esencial en proyectos de desarrollo profesional.

5. Referencias

- Gamma, E. et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*.
- Freeman, E. & Robson, E. (2004). *Head First Design Patterns*.