Patrones de Diseño
MediCareDesk

Presentado por:

Yamid Alfonso Gonzalez Torres

Jenny Catherine Herrera Garzon

Edwin Andres Marin Vanegas

Diego Steven Pinzon Yossa

Profesor:

Oscar Eduardo Alvarez Rodriguez

Universidad Nacional de Colombia

Facultad de Ingeniería

Ingeniería de software

2025

I. Introducción

Este documento tiene como objetivo presentar y justificar la selección de dos patrones de diseño orientados a objetos considerados para su implementación en el sistema **MediCareDesk.** Aunque dichos patrones no fueron integrados completamente en esta versión funcional inicial, se documentan sus principios, intención de uso y aportes esperados en el contexto de la arquitectura desarrollada.

Con base en los requerimientos del curso, se descarta el uso del patrón Singleton, y se analizan dos alternativas pertinentes: **Observer** (comportamiento) y **Builder** (creacional).

II. Patrón Observer

A. Definición y propósito

El patrón **Observer** (o "Observador") es un patrón de comportamiento que establece una relación de dependencia de uno a muchos entre objetos. Su objetivo es garantizar que cuando un objeto cambia de estado, todos los objetos que dependen de él sean notificados y actualizados de forma automática.

Este patrón permite implementar mecanismos de suscripción (o publicación-suscripción), donde un objeto central ("sujeto") mantiene una lista de observadores que desean recibir actualizaciones cuando ocurre un evento específico. De esta manera, se evita el acoplamiento directo entre componentes y se facilita la extensibilidad del sistema.

B. Intención de uso en MediCareDesk

El sistema *MediCareDesk* incluye funcionalidades relacionadas con la **programación, verificación y seguimiento de tomas de medicamentos**. Estas acciones desencadenan efectos que involucran a distintos módulos del sistema, como la emisión de alertas, el registro en el historial del paciente y la evaluación del cumplimiento del tratamiento.

En este contexto, se propone el uso del patrón **Observer** para establecer una estructura de notificación entre los módulos del sistema. Por ejemplo:

- El módulo que gestiona la **programación de tomas** actúa como **sujeto**, notificando eventos como "toma activa", "toma omitida" o "toma confirmada".
- Los módulos de **alertas**, **historial clínico** y **estado de tratamiento** actúan como **observadores**, reaccionando automáticamente a los eventos registrados.

Esta propuesta permite que cada módulo opere de forma independiente, respondiendo únicamente cuando recibe notificaciones pertinentes, sin necesidad de mantener referencias directas entre ellos.

C. Justificación

La incorporación del patrón Observer contribuye significativamente a:

- **Reducir el acoplamiento** entre componentes que interactúan frecuentemente.
- **Facilitar el mantenimiento y evolución** del sistema, al permitir añadir nuevos observadores sin modificar el código del sujeto.
- **Organizar el flujo de eventos** dentro del sistema, especialmente en relación con procesos temporizados como las tomas programadas.
- Permitir futuras extensiones, como módulos que envíen notificaciones por correo o sincronicen datos con otras plataformas, sin alterar el núcleo de la lógica de negocio.

Este patrón es coherente con el diseño modular y la arquitectura monolítica adoptada para el sistema, y promueve una gestión clara y escalable de las responsabilidades relacionadas con los eventos internos del sistema.

III. Patrón Builder

A. Definición y propósito

El patrón **Builder** (o "Constructor") es un patrón de creación que permite construir objetos complejos paso a paso, separando su representación final del proceso de construcción. Su utilidad se hace evidente cuando los objetos a crear tienen múltiples atributos, algunos obligatorios y otros opcionales, y cuando la creación directa mediante un único constructor se vuelve difícil de leer, propensa a errores o rígida ante cambios.

El propósito fundamental del patrón Builder es facilitar la construcción de objetos mediante una secuencia fluida y controlada de pasos, lo que mejora la legibilidad del código y asegura la coherencia interna de los objetos resultantes.

B. Intención de uso en MediCareDesk

El sistema *MediCareDesk* gestiona entidades con múltiples atributos y validaciones internas, como **Paciente**, **Medicamento**, y **Toma**. Por ejemplo, un paciente requiere datos como ide_paciente (identificación), nombre, edad, género, contacto de emergencia, y observaciones médicas. Asimismo, los medicamentos incluyen atributos como id, nombre, indicaciones, fecha de caducidad, principio activo y contraindicaciones

La construcción directa de estos objetos mediante constructores tradicionales puede llevar a confusión, especialmente cuando se manejan muchos parámetros y cuando se requieren validaciones antes de que el objeto esté completo.

Se propone implementar el patrón **Builder** para estos casos, mediante clases especializadas como **PacienteBuilder** o **MedicamentoBuilder**, que expongan métodos secuenciales del tipo:

```python
paciente = PacienteBuilder()\
    .setId_paciente("1098347621")\
    .setNombre("Claudia Martínez")\
    .setEdad(75)\
    .setGenero("Femenino")\
    .setContactoEmergencia("3001234567")\
    .setObservaciones("Diabetes tipo II")\
    .build()
```

Este enfoque permite que los objetos se construyan con mayor legibilidad, que se validen antes de su creación y que se mantenga una estructura clara y segura en la manipulación de datos sensibles.

C. Justificación

El patrón Builder aporta al sistema *MediCareDesk* las siguientes ventajas:

- **Claridad en la construcción de objetos complejos**, con múltiples campos configurables.
- **Facilidad para aplicar validaciones** previas a la construcción definitiva del objeto.
- **Reducción de errores de inicialización**, especialmente cuando hay múltiples atributos opcionales.
- **Adaptabilidad ante cambios futuros** en la estructura de los objetos, al centralizar la lógica de creación.

Desde el punto de vista arquitectónico, este patrón mejora la separación de responsabilidades y refuerza la cohesión de las entidades del sistema, particularmente en una aplicación con énfasis en la integridad de datos y en el manejo sensible de información clínica.

IV. Conclusión

La incorporación de patrones de diseño en el desarrollo de *MediCareDesk* responde a la necesidad de estructurar el sistema de forma robusta, clara y escalable. Los patrones **Observer** y **Builder** se ajustan a los escenarios funcionales del proyecto, promoviendo un diseño orientado a

objetos bien fundamentado, alineado con los objetivos del curso y con principios de buena ingeniería de software.

Su aplicación permite anticiparse a futuros requerimientos, facilitar pruebas y mantenimiento, y asegurar la modularidad dentro de una arquitectura monolítica cuidadosamente organizada. La documentación de estos patrones como parte del diseño evidencia un enfoque consciente, fundamentado y profesional en la construcción del sistema.