Sistema de Notificaciones con Patrones de Comportamiento

Objetivo:
Aplicar al menos tres patrones de comportamiento en la construcción de un sistema de notificaciones adaptable, utilizando buenas prácticas de diseño orientado a objetos en Java.

Contexto del Problema:
Una empresa desea un sistema que permita notificar a los usuarios sobre diferentes eventos (actualización de perfil, alerta de seguridad, promoción, etc.). El sistema debe:

- Elegir el canal de notificación apropiado: Email, SMS o Push → Strategy
- Enviar automáticamente notificaciones cuando ocurren ciertos eventos → Observer
- Formatear los mensajes según el tipo de usuario (Admin, Cliente, Invitado) → Template Method.

—----------

- Validar condiciones antes de enviar una notificación → Chain of Responsibility
- Encapsular el proceso de envío para ejecutarlo, almacenarlo o repetirlo → Command

Requisitos del Sistema (mínimos):

- Selector de canal: Cada notificación debe enviarse por el canal definido en las preferencias del usuario. (Aplicar el patrón Strategy)
- Mecanismo de suscripción a eventos: Diferentes partes del sistema (usuarios, logs, etc.) deben ser notificados cuando ocurre un evento. (Aplicar el patrón Observer)
- Formato personalizado de mensaje según el rol del usuario: Cada tipo de usuario verá el mensaje en un formato diferente. (Aplicar el patrón Template Method)

—----------

- Validación previa al envío: El sistema debe verificar que el mensaje no esté vacío y que el usuario no esté bloqueado antes de enviar la notificación. (Aplicar el patrón Chain of Responsibility)
- Encapsulamiento del envío como orden ejecutable: La notificación debe ser representada como un objeto comando que puede ejecutarse, almacenarse o repetirse. (Aplicar el patrón Command)

Instrucciones técnicas:

1. Implementa una interfaz NotificationStrategy con clases concretas EmailNotification, SMSNotification y PushNotification.
2. Permitir que observadores como User, Logger, o Auditor se suscriban y reaccionen a eventos de negocio.
3. Crea una clase abstracta User con método formatMessage(String message) definido como plantilla, y subclases como AdminUser, ClientUser, etc.
4. Crear subclases como AdminUser, ClientUser, y GuestUser con personalizaciones del formato de mensaje.
5. Definir una clase abstracta NotificationFilter con filtros concretos como: EmptyMessageFilter, BlockedUserFilter.
6. Definir la interfaz NotificationCommand y una clase concreta SendNotificationCommand.
7. Implementar la clase NotificationInvoker, responsable de ejecutar una cola de comandos.

Flujo de Ejecución:

- Se crea un objeto Notification con el usuario, mensaje y canal de preferencia.
- Se aplica la cadena de validación (NotificationFilter) para verificar condiciones previas.
- Si las validaciones se cumplen, se crea un objeto SendNotificationCommand.
- El comando se agrega a un NotificationInvoker que puede ejecutarlo inmediatamente o después.