**EJERCICIOS CONCEPTUALES**

1. **Beneficios de los patrones de diseño**

**Estandarización**: Permite que diferentes desarrolladores trabajen con un mismo enfoque estructurado.  
**Ejemplo***:* En un equipo de desarrollo grande, el uso del patrón *Singleton* para la configuración de la base de datos evita múltiples instancias y problemas de concurrencia.

**Reutilización**: Evita la necesidad de reinventar soluciones para problemas comunes.  
**Ejemplo***:* Un sistema de autenticación puede utilizar el patrón *Factory Method* para crear distintos tipos de autenticación (OAuth, JWT, LDAP).

**Mantenibilidad**: Reduce el acoplamiento y mejora la organización del código.  
**Ejemplo***:* En un sistema de notificaciones, usar *Observer* permite que los módulos de email, SMS y push notifications sean independientes.

1. **Identificación de patrones en código existente.**

Revisando el código en githug https://github.com/joserafaelaga/Notificaciones-Android.git

El patrón Identificado es: **Builder**.

El patrón **Builder** se usa cuando se necesita construir un objeto complejo paso a paso, facilitando la personalización sin necesidad de múltiples constructores.

* **Notification.Builder mBuilder = new Notification.Builder(M**

Se instancia Notification.Builder, que permite configurar progresivamente el objeto. Despué se puede agregar características

* mBuilder.setSmallIcon(android.R.drawable.stat\_sys\_warning);
* mBuilder.setContentTitle("Nuestra primera notificación");
* mBuilder.setContentText("Texto para la notificación");
* mBuilder.setLargeIcon(bmp);
* mBuilder.setAutoCancel(true);
* mBuilder.setSound(alarmSound);
* mBuilder.setVibrate(new long[] { 1000, 1000, 1000, 1000, 1000 });
* mBuilder.setProgress(100,50,true);
* mBuilder.setContentIntent(contIntent);

1. **Clasificación de patrones y cuándo usarlos**

**Creacionales**: Cuando se necesita flexibilidad en la creación de objetos sin acoplamiento directo. Un Ejemplo puede ser***Factory Method*** *en sistemas de pago para diferentes pasarelas*.

**Estructurales**: Para organizar mejor la comunicación entre clases y simplificar interacciones. Otro ejemplo puede ser***Facade*** *en un sistema de reservas como EasyBook*).

**De comportamiento**: Cuando se debe gestionar la comunicación y flujo de datos entre objetos. Un ejemplo puede ser***Observer*** *en sistemas de eventos en tiempo real*).