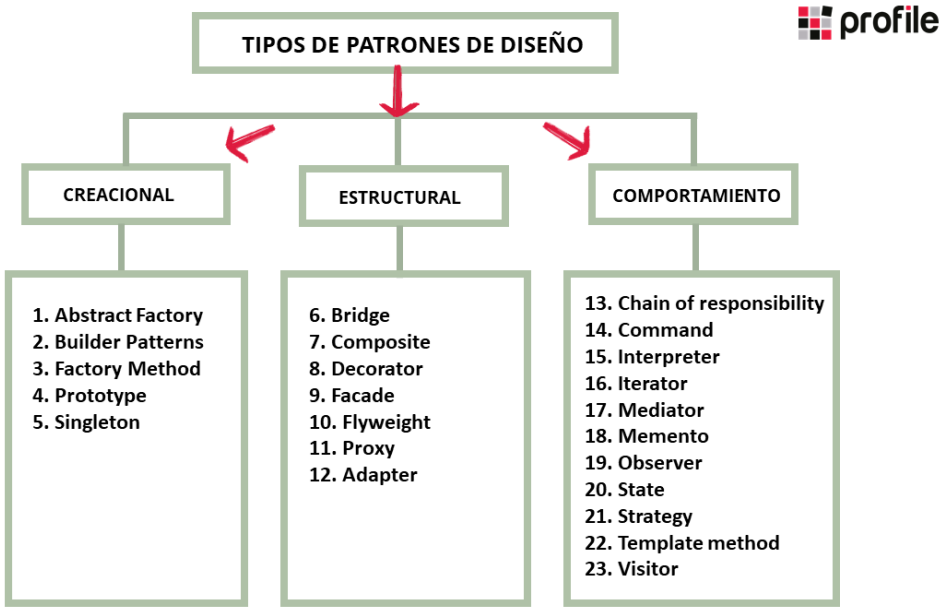
Patrones de diseño:



Los patrones de diseño son soluciones reutilizables a problemas comunes que ocurren en el diseño de software. Estos patrones representan las mejores prácticas que los desarrolladores pueden seguir para resolver problemas específicos en el desarrollo de software. Acá hay algunos ejemplos de los patrones de diseños mas comunes

Patrones Creacionales: Se enfocan en la forma en que se crean los objetos.

Singleton: Asegura que una clase tenga solo una instancia y proporciona un punto de acceso global a ella.

Factory Method: Define una interfaz para crear un objeto, pero permite a las subclases alterar el tipo de objeto que se creará.

Abstract Factory: Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.

Builder: Separa la construcción de un objeto complejo de su representación, permitiendo que el mismo proceso de construcción cree diferentes representaciones.

Prototype: Permite crear nuevos objetos copiando un objeto existente, conocido como prototipo.

Patrones Estructurales: Se ocupan de la composición de clases y objetos.

Adapter: Permite que clases con interfaces incompatibles trabajen juntas.

Decorator: Permite agregar responsabilidades adicionales a un objeto de manera dinámica.

Facade: Proporciona una interfaz simplificada a un conjunto de interfaces en un subsistema.

Composite: Permite a los clientes tratar objetos individuales y composiciones de objetos de manera uniforme.

Proxy: Proporciona un sustituto o marcador de posición para otro objeto para controlar el acceso a él.

Patrones de Comportamiento: Se centran en la comunicación entre objetos.

Observer: Define una dependencia uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente.

Strategy: Permite definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables.

Command: Encapsula una solicitud como un objeto, lo que permite parametrizar a los clientes con diferentes solicitudes, encolar o registrar solicitudes y soportar operaciones que se pueden deshacer.

State: Permite que un objeto altere su comportamiento cuando su estado interno cambia.

Template Method: Define el esqueleto de un algoritmo en una operación, diferiendo algunos pasos a las subclases.

Estos patrones ayudan a los desarrolladores a escribir código más flexible, reutilizable y mantenible.

Ejemplo de patrones de diseño, en este caso usaremos el Singleton, que pertenece a los creacionales:

```
// Clase Singleton
public class Singleton {
    // Variable estática privada para almacenar la única instancia
    private static Singleton instanciaUnica;

    // Constructor privado para evitar la creación de instancias desde fuera
    private Singleton() {
        // Inicialización de la instancia
    }

    // Método público estático para obtener la instancia única
    public static Singleton getInstancia() {
        if (instanciaUnica == null) {
            instanciaUnica = new Singleton();
        }
        return instanciaUnica;
    }

    // Otros métodos de la clase
    public void mostrarMensaje() {
        System.out.println("¡Hola desde Singleton!");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Obtener la única instancia de Singleton
        Singleton singleton = Singleton.getInstancia();

        // Usar la instancia
        singleton.mostrarMensaje();
    }
}
```