# Patrón de Diseño: Singleton

## Definición

El patrón Singleton se emplea para garantizar que una clase posea una única instancia en todo el sistema y proporcionar un acceso centralizado a ella. Este enfoque es útil cuando se necesita un único punto de control o coordinación global, como en el caso de un manejador de configuración o una conexión a base de datos compartida.

## Objetivo

Asegurar que una clase tenga una sola instancia activa y permitir su acceso mediante un método global.

## Aplicaciones comunes

- Gestión de configuraciones en sistemas complejos.

- Control de acceso a recursos compartidos, como archivos o conexiones de red.

- Implementación de registros de log (logging).

## Ventajas

- Control centralizado: Permite gestionar desde un único lugar los recursos compartidos.

- Instancia única garantizada: Evita la duplicación innecesaria de objetos costosos.

- Acceso sencillo: La instancia se obtiene mediante un método estático, lo que facilita su uso.

## Desventajas

- Dificulta pruebas unitarias: Al estar acoplado globalmente, puede ser complicado de simular o reemplazar en pruebas.

- Posible abuso: Puede ser utilizado incorrectamente como una solución rápida, generando código mal estructurado.

- Problemas en entornos multihilo: Si no se maneja correctamente, pueden surgir errores de concurrencia.

## Ejemplo en Java

public class GestorConfiguracion {  
 private static GestorConfiguracion instanciaUnica;  
  
 private GestorConfiguracion() {  
 // Constructor privado  
 }  
  
 public static GestorConfiguracion obtenerInstancia() {  
 if (instanciaUnica == null) {  
 instanciaUnica = new GestorConfiguracion();  
 }  
 return instanciaUnica;  
 }  
}