**PATRON SINGLETON**



Singleton es un patrón de diseño creacional crea una clase con una única instancia la cual proporciona un punto de acceso global a esa instancia, además es necesario que la clase tenga un campo estático privado para almacenar esta instancia.

***Método de uso***

Agrega un campo estático y privado en la clase para almacenar la única instancia del Singleton.

Define un método estático y público que permita obtener la instancia del Singleton.

Implementa una inicialización diferida dentro del método estático. La primera vez que se llame, deberá crear un nuevo objeto y almacenarlo en el campo estático. En las llamadas posteriores, devolverá siempre la misma instancia.

Declara el constructor de la clase como privado. Esto garantizará que solo el método estático pueda instanciar el objeto, evitando que otros objetos accedan directamente al constructor.

Revisa el código cliente y reemplaza cualquier creación directa del Singleton mediante el constructor por llamadas al método de creación estático.

***Ventajas del patrón Singleton***

1. Control de acceso a la instancia única: Garantiza que solo exista una instancia de la clase en todo el programa, evitando la creación de múltiples objetos innecesarios.
2. Ahorro de memoria y recursos: Al evitar múltiples instancias, se reduce el consumo de memoria y se optimiza el uso de recursos, especialmente útil para manejar conexiones a bases de datos o acceso a archivos.
3. Punto de acceso global: Permite que múltiples partes del código accedan a la misma instancia sin necesidad de pasar referencias entre objetos.
4. Facilita la implementación de ciertas funcionalidades: Es útil para gestionar registros de logs, configuración de aplicaciones, gestión de conexiones, cachés, etc.

***Desventajas del patrón Singleton***

1. **Dificulta las pruebas unitarias**: Al ser una instancia global, puede generar dependencias ocultas, lo que hace más difícil la prueba de componentes individuales sin afectar otras partes del sistema.
2. **Acoplamiento excesivo**: Los módulos que dependen del Singleton quedan acoplados a su implementación, lo que reduce la flexibilidad y la capacidad de modificar o extender la funcionalidad sin afectar otras partes del código.
3. **Riesgo de crear una instancia oculta**: En entornos multihilo, si el Singleton no está bien implementado con sincronización adecuada, puede haber problemas de concurrencia y se podrían crear múltiples instancias accidentalmente.
4. **Dificultad en la gestión de ciclo de vida**: La instancia única puede permanecer en memoria durante toda la ejecución del programa, lo que podría causar fugas de memoria si no se maneja correctamente.
5. **Rompe el principio de responsabilidad única (SRP)**: Un Singleton no solo administra su propia instancia, sino que también suele contener lógica de negocio, lo que viola el principio de responsabilidad única y puede hacer que el código sea más difícil de mantener.

***¿Cuándo usar Singleton?***

Cuando se necesita una única instancia global (ejemplo: gestores de configuración, registros, conexiones a base de datos).

Cuando el objeto es costoso de crear y debe compartirse entre múltiples partes del sistema.

***¿Cuándo evitar Singleton?***

Cuando se necesita escalabilidad o flexibilidad en la aplicación.

Cuando puede generar problemas de concurrencia en entornos multihilo.

Cuando afecta la capacidad de hacer pruebas unitarias de manera aislada.

***Explicación De Código***

patrón se aplica en la clase conexionS.java. La finalidad de este Singleton es controlar la conexión a la base de datos asegurando que solo haya una conexión activa en toda la ejecución del programa.

1. **Uso de una variable estática** → Se define una variable estática que almacenará la única instancia de la clase.
2. **Constructor privado** → Se evita que otras clases creen nuevas instancias directamente.
3. **Método getinstancia ()** → Es el encargado de crear la instancia solo si no existe; si ya existe, simplemente la devuelve.
4. **Métodos conectar () y desconectar ()** → Simulan una conexión a una base de datos.

Cuando el programa principal (app.java) intenta obtener una conexión llamando a conexionS.getinstancia(), siempre devuelve la misma instancia, lo que garantiza que **no haya múltiples conexiones simultáneas**, optimizando el uso de recursos.