INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	Tecnología de objeto				
TÍTULO DE LA PRÁCTICA:	*Ejercicios-Singleton*				
NÚMERO DE PRÁCTICA:	*09*	**AÑO LECTIVO:**	*2025-B*	**NRO. SEMESTRE:**	*VI*
FECHA DE PRESENTACIÓN	*08/12/2025*	**HORA DE PRESENTACIÓN**	*23:59:00*		
INTEGRANTE (s): -Gutierrez Ccama Juan Diego				**NOTA:**	
DOCENTE(s): *Ing. CARLO JOSE LUIS CORRALES DELGADO*					

SOLUCIÓN Y RESULTADOS

I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS

REPOSITORIO: https://github.com/UJKMjuandi/teo/tree/main/Laboratorio/Laboratorio09

II. SOLUCIÓN DEL CUESTIONARIO

Práctica 07 – Patrón de Diseño Singleton

Introducción

En esta práctica se desarrollan distintos ejercicios orientados a la comprensión y aplicación del patrón de diseño Singleton. Dicho patrón permite asegurar que una clase tenga una única instancia y proporciona un punto de acceso global a ella, lo cual resulta útil para el manejo de recursos compartidos y estados globales del sistema.

Los ejercicios progresan desde una implementación básica hasta escenarios más complejos, como el uso en videojuegos y entornos multihilo.

Ejercicio 01 – Implementación directa del Singleton

Objetivo
Comprender la estructura básica del patrón Singleton y verificar que solo exista una instancia de una clase durante la ejecución del programa.

Descripción
Se implementa una clase llamada `Configuracion`, la cual almacena configuraciones generales del sistema, como el idioma y la zona horaria. El patrón Singleton garantiza que, aunque se creen múltiples objetos, todos hagan referencia a la misma instancia.

Implementación
La clase utiliza un atributo estático para almacenar la instancia única y redefine el método de creación del objeto para evitar duplicaciones. Además, se incluye el método `mostrar_configuracion()` para visualizar los valores actuales.

Verificación
En el archivo principal se crean múltiples referencias a la clase `Configuracion` y se comprueba que todas apuntan a la misma instancia mediante la comparación de referencias.

Conclusión
Este ejercicio demuestra el uso básico del patrón Singleton para manejar configuraciones globales y evitar la duplicación de recursos.

Ejercicio 02 – Singleton con recursos compartidos (Logger)

Objetivo
Aplicar el patrón Singleton para manejar un recurso compartido, asegurando que todos los módulos del sistema utilicen un único archivo de registro.

Descripción
Se implementa una clase `Logger` que escribe mensajes de registro en un archivo de texto llamado `bitacora.log`. El patrón Singleton garantiza que todos los mensajes provengan de una única instancia y se escriban en el mismo archivo.

Cada mensaje incluye la fecha y hora en la que fue registrado.

Implementación
La clase `Logger` controla su única instancia mediante un atributo estático y proporciona el método `log(mensaje)` para escribir en el archivo. El programa principal simula el uso del logger desde distintos módulos.

Verificación
Al ejecutar el programa, se observa que todos los mensajes se almacenan en un solo archivo, confirmando que se utiliza una única instancia del logger.

Conclusión
Este ejercicio muestra cómo el patrón Singleton es ideal para la gestión de recursos compartidos como archivos de registro.

Ejercicio 03 – Conexión simulada a base de datos

Objetivo
Simular una conexión a base de datos utilizando el patrón Singleton para asegurar que solo exista una conexión activa.

Descripción
Se implementa la clase `ConexionBD`, la cual representa una conexión simulada a una base de datos. El Singleton garantiza que todas las partes del sistema compartan la misma conexión.

Implementación
La clase incluye los métodos `conectar()`, `desconectar()` y `estado()`, permitiendo controlar y consultar el estado de la conexión. Si se intenta crear una nueva conexión, se devuelve la existente.

Verificación
El programa principal crea múltiples referencias a la conexión y comprueba que todas comparten el mismo estado.

Conclusión
Este ejercicio demuestra la utilidad del Singleton para manejar recursos críticos como conexiones a bases de datos.

Ejercicio 04 – Singleton en un juego

Objetivo
Aplicar el patrón Singleton para manejar el estado global de un juego y permitir que distintos componentes accedan a la misma información.

Descripción
Se implementa la clase `ControlJuego`, que mantiene el estado global del juego, incluyendo el nivel actual, el puntaje y las vidas disponibles.

Componentes como el jugador, los enemigos y la interfaz acceden y modifican este estado compartido.

Implementación
La clase utiliza el patrón Singleton para garantizar una única instancia del estado del juego. Se incluyen métodos para modificar el nivel, el puntaje y las vidas.

Verificación

El archivo principal simula distintos módulos que acceden al mismo `ControlJuego`, comprobando que los cambios se reflejan globalmente.

Conclusión
Este ejercicio evidencia el uso del Singleton para gestionar estados globales en videojuegos y sistemas interactivos.

Ejercicio 05 – Singleton seguro en entornos multihilo

Objetivo
Garantizar que el patrón Singleton funcione correctamente en entornos concurrentes con múltiples hilos de ejecución.

Descripción
Se modifica el ejercicio del Logger para hacerlo seguro en entornos multihilo, evitando que se creen múltiples instancias al mismo tiempo.

Implementación
Se utiliza un mecanismo de bloqueo (`Lock`) junto con la técnica de doble verificación (double-checked locking) para asegurar que solo se cree una instancia del Singleton.

Verificación
El programa principal crea varios hilos que escriben simultáneamente en el archivo de log, confirmando que todos utilizan la misma instancia.

Conclusión
Este ejercicio demuestra cómo proteger un Singleton en escenarios concurrentes y prevenir condiciones de carrera.

CUESTIONARIO:

1. ¿Qué desventajas tiene el patrón Singleton en pruebas unitarias?

El patrón Singleton dificulta las pruebas unitarias debido a que mantiene un estado global compartido. Esto puede generar dependencias entre pruebas y afectar su independencia, haciendo necesario reiniciar o simular la instancia.

2. ¿Cuándo no es recomendable usar Singleton?

No es recomendable usar Singleton cuando se requiere alta flexibilidad, múltiples instancias, escalabilidad, paralelismo o cuando se desea facilitar las pruebas unitarias y el mantenimiento del sistema.

3. ¿Cómo se diferencia de una clase estática?

Una clase estática no puede instanciarse y solo contiene métodos y atributos estáticos. En cambio, el Singleton es una clase normal que tiene una única instancia, puede implementar herencia, interfaces y mantener control explícito sobre su ciclo de vida.

III. CONCLUSIONES

A través de esta práctica se ha demostrado cómo el patrón Singleton puede aplicarse en distintos contextos, desde configuraciones simples hasta sistemas concurrentes. Su uso adecuado permite controlar recursos críticos y estados globales, aunque debe aplicarse con cuidado para evitar problemas de acoplamiento y testeo.

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA