**Definición**

El patrón de diseño Singleton es un patrón de creación que asegura que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia. Este patrón es útil en situaciones donde se necesita exactamente un objeto para coordinar acciones en todo el sistema.

**Implementación Básica**

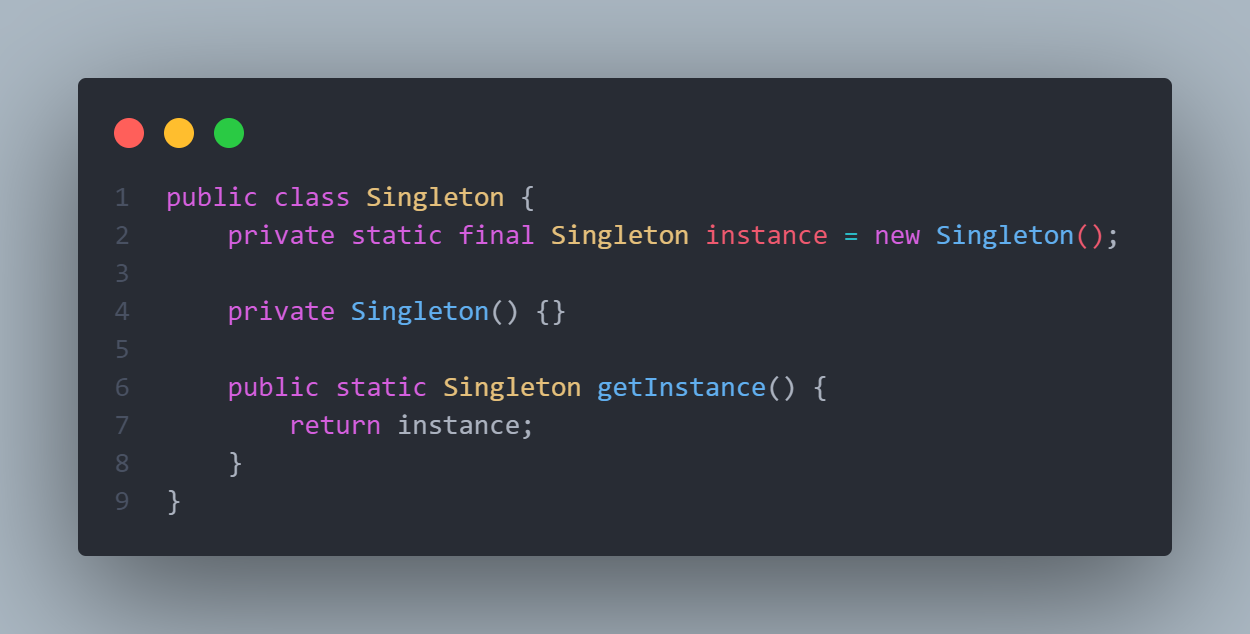
Para implementar el patrón Singleton, se deben seguir estos pasos:

1. **Constructor privado:** Evita que otros objetos instancien la clase directamente.
2. **Instancia estática privada:** Una referencia estática a la única instancia de la clase.
3. **Método estático público:** Un método que proporciona acceso a la instancia única, creando la instancia si aún no existe.

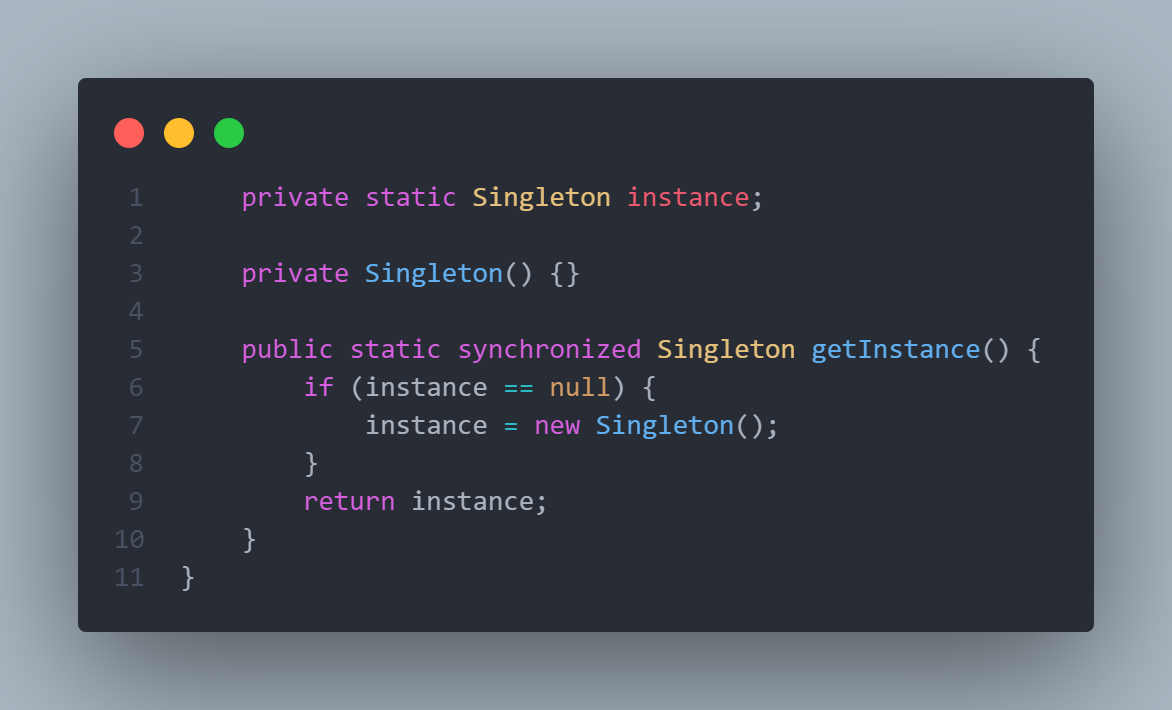


**Variantes del Patrón Singleton**

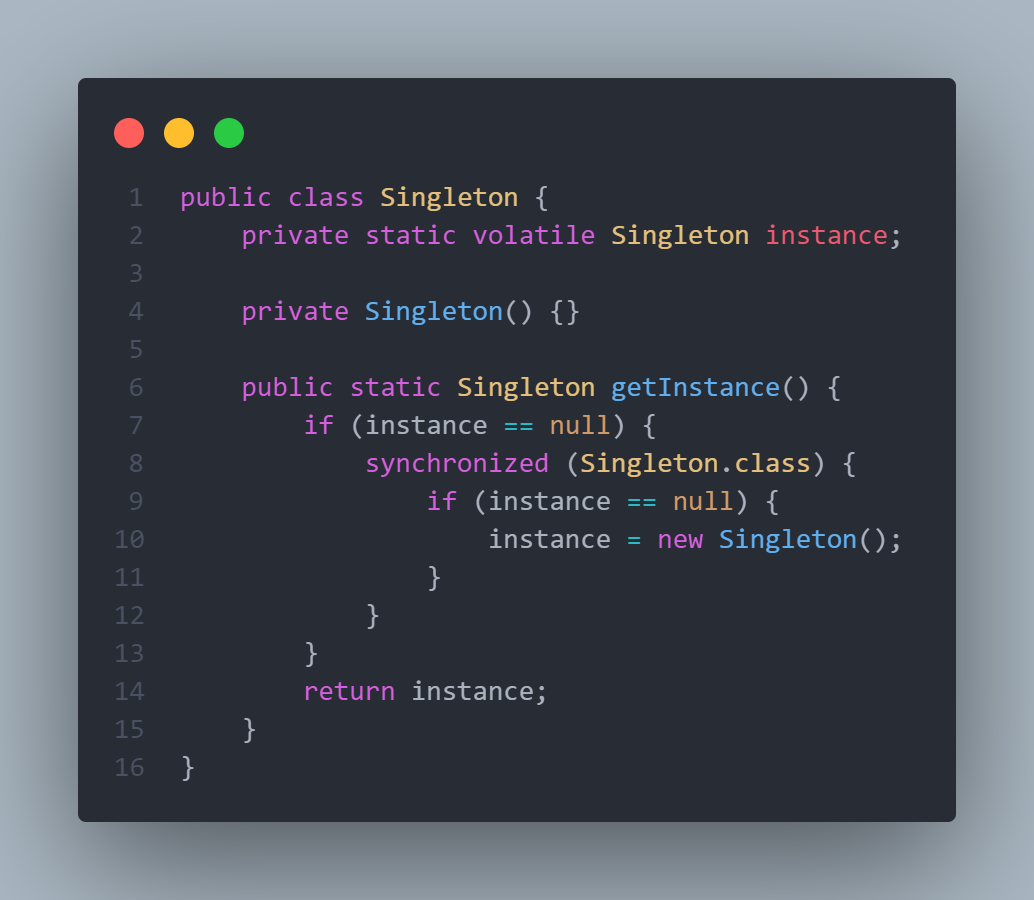
1. **Singleton con Inicialización Temprana (Eager Initialization):** La instancia se crea en el momento de la carga de la clase.



1. **Singleton con Inicialización Perezosa (Lazy Initialization):** La instancia se crea solo cuando se necesita.



1. **Singleton con Bloqueo Doble (Double-Checked Locking):** Optimiza el Singleton para uso en entornos multihilo.



**Situaciones de Uso**

El patrón Singleton es útil en diversas situaciones, incluyendo:

1. **Control de Acceso a Recursos Compartidos:** Cuando se necesita coordinar el acceso a un recurso compartido, como una base de datos o un archivo.
2. **Controladores y Gestores:** Para controlar la creación y gestión de recursos, como controladores de impresoras o gestores de conexión.
3. **Configuraciones Globales:** Cuando se necesita una configuración global accesible en toda la aplicación.
4. **Gestión de Log:** Para asegurarse de que todas las clases de una aplicación utilicen el mismo objeto de log.

**Ventajas**

1. **Control sobre la instancia:** Garantiza que sólo exista una instancia de la clase.
2. **Acceso global:** Proporciona un punto de acceso global a la instancia única.
3. **Evita la creación de múltiples instancias:** Ideal para situaciones donde se requiere un único punto de control.

**Desventajas**

1. **Problemas de Pruebas Unitarias:** Puede dificultar la creación de pruebas unitarias debido a la dependencia global.
2. **Problemas de Concorrencia:** En entornos multihilo, la implementación debe manejar la sincronización correctamente.
3. **Responsabilidad Única:** Puede violar el principio de responsabilidad única si se abusa del patrón para gestionar demasiadas responsabilidades.