Untitled

#Practica 5. Singleton #ejemplo de patron de diseño singleton -sistema de registro de logs.

class Logger: #atributo de la clase para guardar la instancia unica \_instancia = None # Metodo **new** controla la creacion del objeto antes de init. se asegura que solo exista una unica instancia de logger def **new**(cls, \*args, \*\*kwargs): if cls.\_instancia is None: cls.\_instancia = super().\_\_new\_\_(cls) cls.\_instancia.archivo = open(“log.txt”, “a”) return cls.\_instancia #devuelve la minima instancia

def registro(self, mensaje):  
 self.archivo.write(mensaje + "\n")  
 self.archivo.flush() #asegura que el mensaje se escriba inmediatamente en el archivo

registro1 = Logger() #estamos creando la unica instancia registro2 = Logger() #intento de crear una segunda instancia

registro1.registro(“Primer mensaje de log”) registro2.registro(“Segundo mensaje de log”)

print(registro1 is registro2) #deberia ser True, ambas variables apuntan a la misma instancia

# 1. ¿Qué pasaría si eliminamos la verificación if cls.\_instancia is None en el método **new**?

def **new**(cls, \*args, \*\*kwargs): # ❌ VERIFICACIÓN ELIMINADA cls.\_instancia = super().\_\_new\_\_(cls) cls.\_instancia.archivo = open(“log.txt”, “a”) return cls.\_instancia #devuelve la minima instancia

#Se crearían múltiples instancias: Cada vez que se llame Logger(), se crearía una nueva instancia

#Sobrescritura del archivo: El atributo \_instancia se sobrescribiría constantemente

##Pérdida del patrón Singleton: La clase dejaría de ser un Singleton y se comportaría como una clase normal

#Problemas con el archivo: Posiblemente se intentaría abrir múltiples veces el mismo archivo, lo que podría causar errores de acceso

#Fuga de memoria: Las instancias anteriores se perderían en memoria sin poder ser accedidas

# 2.En la línea print(registro1 is registro2). ¿Qué significa que devuelva True en el contexto del Singleton?

#Confirmación del patrón: Verifica que efectivamente estamos trabajando con una única instancia

# Consistencia global: Todas las referencias a Logger acceden al mismo objeto

# Estado compartido: Cualquier cambio realizado a través de registro1 se reflejará inmediatamente en registro2

# Control centralizado: Existe un único punto de control para el sistema de logs

# 3. ¿Crees que siempre es buena idea usar Singleton para todo lo que es global? Da un ejemplo donde no sería recomendable.

#No siempre es recomendable usar Singleton para todo lo que es global. Algunos ejemplos donde no sería recomendable incluyen:

# ❌ MAL USO - Singleton para configuración

class ConfiguracionApp: \_instancia = None

def \_\_new\_\_(cls):  
 if cls.\_instancia is None:  
 cls.\_instancia = super().\_\_new\_\_(cls)  
 cls.\_instancia.cargar\_configuracion()  
 return cls.\_instancia  
  
def cargar\_configuracion(self):  
 # Carga configuración desde archivo  
 pass