# Exercise 1: Implementing the Singleton Pattern (.NET)

## What I Learned

In this exercise, I implemented the Singleton design pattern using C# in a .NET environment. The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. It is very useful when we want a single shared resource such as a logging utility throughout the application's lifecycle.

## Concepts Covered

- Singleton Design Pattern in .NET

- Private constructor and static instance

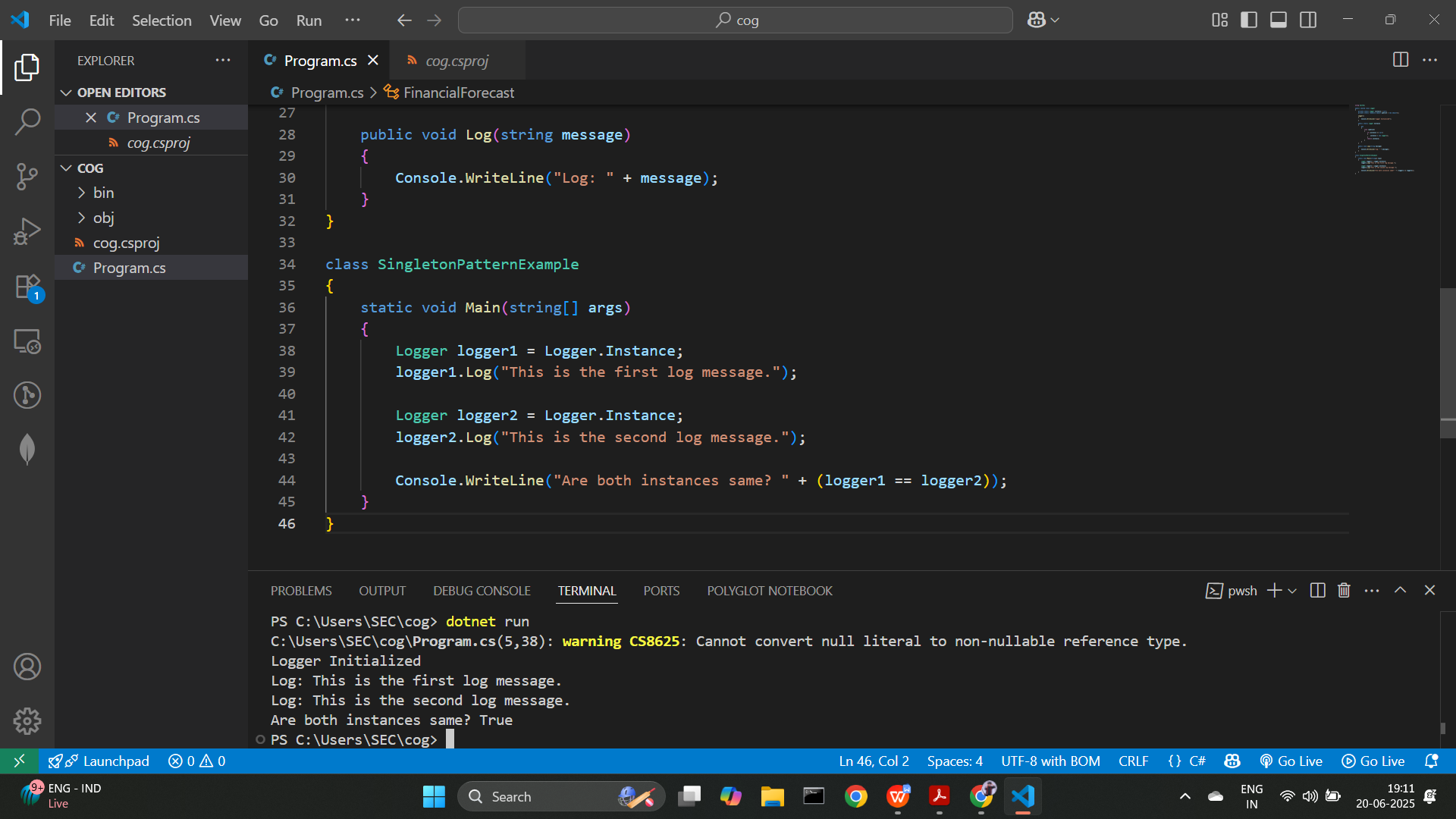
- Thread-safe singleton implementation

- Consistent usage of a single logger instance

## C# Code

using System;  
  
public sealed class Logger  
{  
 private static Logger instance = null;  
 private static readonly object padlock = new object();  
  
 Logger()  
 {  
 Console.WriteLine("Logger Initialized");  
 }  
  
 public static Logger Instance  
 {  
 get  
 {  
 lock (padlock)  
 {  
 if (instance == null)  
 {  
 instance = new Logger();  
 }  
 return instance;  
 }  
 }  
 }  
  
 public void Log(string message)  
 {  
 Console.WriteLine("Log: " + message);  
 }  
}  
  
class SingletonPatternExample  
{  
 static void Main(string[] args)  
 {  
 Logger logger1 = Logger.Instance;  
 logger1.Log("This is the first log message.");  
  
 Logger logger2 = Logger.Instance;  
 logger2.Log("This is the second log message.");  
  
 Console.WriteLine("Are both instances same? " + (logger1 == logger2));  
 }  
}

## ouput



## Conclusion

By implementing the Singleton pattern in .NET, I made sure that the Logger class has only one instance. Using lock ensures thread-safety and prevents multiple threads from creating separate instances. The program output confirms that both logger1 and logger2 point to the same object.