***WEEK-1 mandatory HandsOn Exercise(DATA PATTERNS AND PRINCIPLE)***

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:** You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

**1.Create a New c# Project:**

* + Create a new c# project named **SingletonPatternExample**.

**2.Define a Singleton Class:**

* + Create a class named Logger that has a private static instance of itself.
  + Ensure the constructor of Logger is private.
  + Provide a public static method to get the instance of the Logger class.

**3.Implement the Singleton Pattern:**

* + Write code to ensure that the Logger class follows the Singleton design pattern.

**4.Test the Singleton Implementation:**

* + Create a test class to verify that only one instance of Logger is created and used across the application.

***CODE:***

//Logger.cs

public sealed class Logger

{

    private static readonly Lazy<Logger> \_instance =

        new Lazy<Logger>(() => new Logger());

    private Logger()

    {

        Console.WriteLine("Logger instance created");

    }

    public static Logger Instance => \_instance.Value;

    public void Log(string message)

    {

        Console.WriteLine($"[LOG] {DateTime.Now}: {message}");

    }

}

//program.cs

class Program

{

    static void Main(string[] args)

    {

        Logger logger1 = Logger.Instance;

        Logger logger2 = Logger.Instance;

        Console.WriteLine($"Same instance? {ReferenceEquals(logger1, logger2)}");

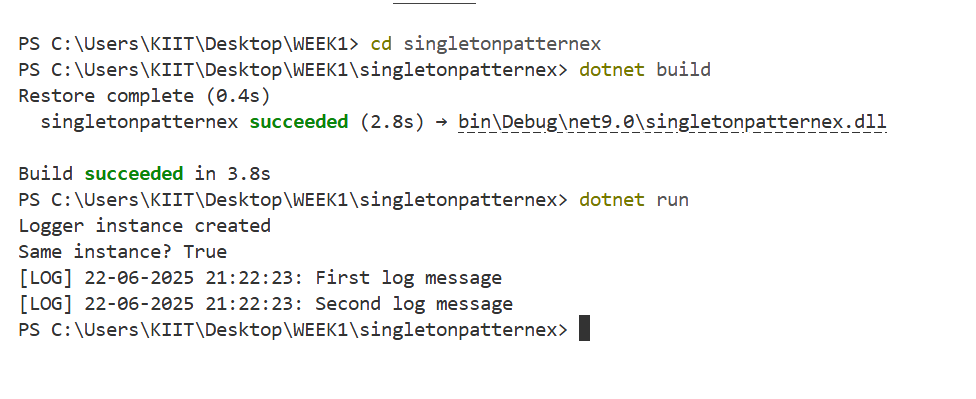
        logger1.Log("First log message");

        logger2.Log("Second log message");

}

}

***Output:***

******