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 Java Project:**
   - Create a new Java project named **SingletonPatternExample**.

     Created a Folder named SingletonPatternExample.
2. **Define a Singleton Class:**
   - Create a class named Logger that has a private static instance of itself.

     Code:-

```
using System;

namespace Singleton
{
   public class Logger
   {
      // Step 2: Create a private static instance of Logger
      private static Logger instance;

      // Step 2: Make constructor private to prevent instantiation
      private Logger()
      {
         Console.WriteLine("Logger instance created");
      }

      // Step 3: Provide a public static method to get the instance
      public static Logger GetInstance()
      {
         if (instance == null)
         {
                instance = new Logger(); // Lazy initialization
            }
            return instance;
        }

        // Sample method to demonstrate logging
        public void Log(string message)
        {
            Console.WriteLine("Log: " + message);
        }
    }
}
```

- Ensure the constructor of Logger is private.

  The Logger is Private
- Provide a public static method to get the instance of the Logger class.
- Provided 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.

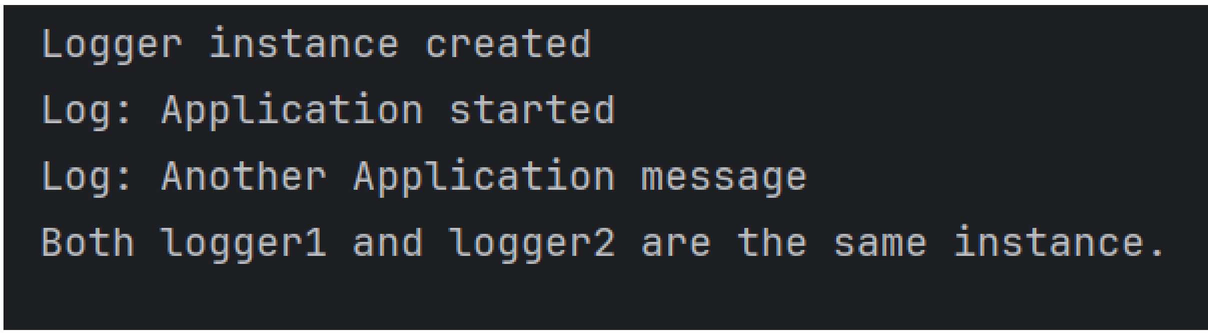
```
using System;

namespace Singleton
{
    public class Program
    {
        public static void Main(string[] args)
        {
            // Get two Logger instances
            Logger logger1 = Logger.GetInstance();
            Logger logger2 = Logger.GetInstance();

            // Test logging
            logger1.Log("Application started");
            logger2.Log("Another log message");

            // Verify that both logger references point to the same object
            if (logger1 == logger2)
            {
```

```csharp
                Console.WriteLine("Both logger instances are the same (singleton confirmed).");
            }
            else
            {
                Console.WriteLine("Logger instances are different (singleton failed).");
            }
        }
    }

    public sealed class Logger
    {
        private static Logger instance = null;
        private static readonly object padlock = new object();

        private Logger()
        {
        }

        public static Logger GetInstance()
        {
            lock (padlock)
            {
                if (instance == null)
                {
                    instance = new Logger();
                }
                return instance;
            }
        }

        public void Log(string message)
        {
            Console.WriteLine(message);
        }
    }
}
```

59. **Test the Singleton Implementation:**

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

      A test class was created and the output of the code was:



```
Logger instance created
Log: Application started
Log: Another Application message
Both logger1 and logger2 are the same instance.
```