Look at the Logger class below. With the current implementation, we can create multiple loggers writing to the same log file in parallel.Use the singleton pattern to ensure only a single logger can be instantiated for a given file.

|  |
| --- |
| public class Logger {      private String fileName;      public Logger(String fileName) {          this.fileName = fileName;      }      public void write(String message) {          Console.WriteLine(“Writing a message to the log.");      }  } |

Step 1: Create a new project call it singletonlogger app

Step 2: Implement this singleton class

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace SingletonLoggerApp  {  public class Logger  {  private string fileName;  private static Dictionary<string, Logger> instances = new Dictionary<string, Logger>();  private static readonly object lockObject = new object();  private Logger(string fileName)  {  this.fileName = fileName;  }  public void Write(string message)  {  Console.WriteLine("Writing a message to the log.");  }  public static Logger GetInstance(string fileName)  {  lock (lockObject)  {  if (!instances.ContainsKey(fileName))  instances[fileName] = new Logger(fileName);  return instances[fileName];  }  }  }  } |

Step 3:Test the application from program.cs

|  |
| --- |
| namespace SingletonLoggerApp  {  internal class Program  {  static void Main(string[] args)  {  var logger1 = Logger.GetInstance("file1");  var logger2 = Logger.GetInstance("file1");  Console.WriteLine(logger1 == logger2);  var logger3 = Logger.GetInstance("file2");  Console.WriteLine (logger1 == logger3);  }  }  } |