# Assignment 13

## 1.

For this assignment, we have picked an asynchronous logger. Please keep in mind that in the first version there are bugs which are fixed in the refactored version of the logger. The expected functionality of the component is described below.

Log Component

The component is a small DLL that does asynchronous writing of strings to a file. A call to Write on the interface must be as fast as possible so the calling application can get on with its work without waiting for the log to be written to a file. If we cross midnight, a new file with a new timestamp is made.

It is possible for the component to be stopped in two ways:

- Stop it right away and if there are any outstanding logs, they are not written
- Stop by waiting for it to finish, writing outstanding logs if any

In case of an error, the calling application, is not stopped. It is more important that the application continues to run than lines not being written to the log.

## 2.

### The refactored version contains changes which are summarized below:

It is using a thread-safe data structure for the Loglines as well as a different, simpler infinite logging loop approach. Both of the versions implement the ILog interface and use the FileHandler class for I/O operations. The Logger utilizes the Singleton pattern which ensures only one instance of it can be instantiated at any one time.

The main take from the revision of the code is the decoupling of the folder/file management/writing and the async logger itself. This separation of responsibilities aims to improve maintainability, reusability and also make the code more readable and easier to comprehend. A class called “FileHandler” has been created, which manages the setup of the LogDirectory as well as that of the log files themselves. This includes headers of the files, formatting, naming, as well as writing in the files. Furthermore, error handling has been introduced to prevent the program from crashing, especially on IO/locked files. Also, the FileHandler allows for the LogDirectory to be changed which automatically handles the allocation of log files to the new directory.

Several bugs were fixed such as the modification of the “_lines” collection during the execution of the foreach. A simple ToList() invocation on the collection creates a non-referenceable copy which cannot be modified during the execution of the loop and the exception is avoided.

Another notable bug is failing to create a new log when midnight passes. This was due to the incorrect way of checking the difference between the current time and the startup time of the logger.

Additional changes include reduced nesting of if statements, removing redundant code, using string interpolation, closing Stream Writers to remove file locks.

Test logging can be done to determine how the component operates when midnight has passed.

NUnit has been added as the test framework of choice. Test cases have been created for the major demands. For more details, check the code file.

## 3.

The result is a much more comprehensible codebase with a test suite that allows for verification of newly introduced changes. In terms of future improvements, the LogLine class could be abstracted in order to be more reusable. This opens the possibility of it being customized and used for different versions of the logger.

Furthermore, for convenience purposes, more comprehensive testing can be done on the midnight requirement, e. g. with mocking to simulate that midnight has passed instead of relying on the system clock passing midnight.

Also, the NUnit tests can be executed upon every new commit, through the help of software such as Jenkins or Teamcity in order to streamline the validation of newly added software or modifications to the existing codebase.