MobilePay Code Test - Notes (Ioana Jivan)

General remarks:

- I found it challenging to work with the file system and with the multiple threads, as I did not try that before in this context, and even though I struggled with these technical limitations, it was a great exercise to try to come up with a solution.

- **Concurrency**: One issue I found was caused by the fact that the log lines collection is being modified by one thread while being iterated by another. As far as I could see, the code was built around a producer-consumer pattern, where the caller application writes (produces) lines to be logged to a list and moves on as quickly as possible, while the consumer, in another thread, iterates through the list and does the actual logging to the file. The problem is that the list cannot be iterated with a foreach by the consumer while also being added to by the calling application, so it throws an InvalidOperationException. I did not manage to solve this issue, although I think that some solutions could be to iterate with a for loop instead of foreach, to use a different kind of collection, or to manually manage the threads by using locks when writing and reading from the list.

- **Exceptions**: I tried to meet the requirement of keeping the caller application responsive even in the event of errors by doing some exception handling. Even though I log the exception message to the console, I believe it would be better to address different kinds of exceptions in different ways and not only log some text, but that depends on the specifics of the caller application.

- **Unit tests**: The tests do not pass because it was not possible to open my test log files for reading after using the log to write to them. The file was used by another thread for writing. As far as I could see, the StreamWriter in the AsyncLog somehow locks the file for writing and does not release the lock when done. As a result, it was not possible to read from the file to carry out my assert operations. I think a fix could be to dispose of the StreamWriter after each “batch” of lines is logged and use a new one when new lines need to be logged, but I did not know how to implement this. However, by opening the files produced by the test methods, I can see that the output is exactly as expected, so the tests would pass if I knew how to make it possible to read the contents of the files.

Changes to the code:

LogLine.cs

- I moved the responsibility for formatting the log line text to the LogLine class, as opposed to the AsyncLog itself. I also renamed and changed the LineText() method to build the complete string to be logged (timestamp and text), instead of returning just the text.
- I removed the regions because they made the code more confusing and difficult to read, especially as there were not many lines of code in the class.

AsyncLog.cs

- I moved _QuitWithFlush higher up with the other private fields and renamed it to _StopWithFlush for naming consistency. The same with _curDate.
- I changed the constructor to take in the directory path from the consumer application and store it in the _directoryPath private field for later use, instead of having it hardcoded. This provides more flexibility, giving the caller application control over where the files are saved.
- I created 2 private methods (InitializeDirectory and InitializeNewFile) for the directory and file setup and used those inside the MainLoop before the while loop start, instead of the constructor, to make the code cleaner and easier to read. The methods take the directory path from the new private field _directoryPath, which is set in the constructor.
- The MainLoop:
    - I removed the integer f, as I am unsure why it would be necessary to loop 4 times in the list of log lines without executing the code in the body of the foreach loop. I might have misunderstood this.
    - I simplified the code in the if statement for checking for the midnight condition using the InitializeNewFile method mentioned above.
    - The code for formatting the log line text is moved to the LogLine.cs class as indicated before. This enforces the separation of concerns of both classes.
    - I added a try-catch block to allow the caller application to continue running even if there would be an error when writing the log lines. I used a separate catch block for the known InvalidOperationException exception caused by iterating the log lines list while it is being added to by the caller application, and another catch block for all other generic exceptions. I simply log a message and the exception text to the console.
    - I moved the line _handled.Add(logLine); after the writing operation, as it made more sense to have it after the writing of the line to the log is completed.

Program.cs

- Because I changed the constructor of the AsyncLog to take the directory path as a parameter, I had to change the Program.cs file to pass it in. I am not sure if changing the calling application was against the rules of the test, but I believe the change is beneficial because it moves the control of where to write the logs to the calling application, instead of having it hardcoded.

Unit tests

- I used the xUnit framework for the tests. I also used the Arrange-Act-Assert structure for the test methods, so they are easy to read and maintain. It was not possible to read from the test output files to carry out the assert operations, because they were being used for writing by the StreamWriter of the logger, but by opening the files, I can see that the output is as expected. I was not sure how to test the fact that new files are created at midnight, because I did not think it would be a good idea to make it possible to overwrite the logger current date to simulate midnight in the test method.