**Code Review Excercise**

**1) If you were to review the following code, what feedback would you give? Please be specific and indicate any errors that would occur as well as other best practices and code refactoring that should be done.**

I have seen a lot of things that should be changed, including errors, not recommendable programming practices, names that disregard the naming conventions, the whole Project being inside a unique file with sql connections, and operations that should be on a whole different class and method (like the write file operations).

* There was an excessive number of booleans in the constructor. I would use a constructor without parameters. In order to choose how to use the object I have set some public variables. Alternatively, an array of booleans can be used for a new constructor (params class[]).
* There shouldn’t be variables with the same name, for example “Bool message” and “String message”.
* The original code doesn't use proper naming conventions, in c# it’s a good practice to have meaningful names, and never prefix or suffix, no matter if the variable is too long.
* When validating, ”message.trim()” is the wrong function to use. I used “string.IsNullOrWhiteSpace(string message)” to check if there is any text in the message that needs to be saved, or if it empty.
* In order to obtain message types like “Error”, “Message”, “Warning”, using an enum with a
* default value makes the code more readable. The default value is there just in case I forget to set the type later. Therefore, I set “None” with value 0 as the default.
* I wouldn’t use so many “if” statements, but a “switch” so as to increase readability.
* In the console’s color,  I have created a method switching the console’s forecolor with the enum of message types as parameter, assigned by default the color it already has (In case there is some kind of error) and also  switched in order to determine the color according to the kind of message.
* I saw concatenation uses repeatedly, so I’ve defined some strings to use in the messages and to have them be standardized.

**2) Rewrite the code based on the feedback you provided in question 1. Please include unit tests on your code.**

I have rewritten the code without using many parameters in the constructors because I think they are not necessary, and a public variable can be set in the class for those configurations. That way, the class can be reused without having to be instantiated multiple times. As I explained on the errors, another way to do it is using an array of parameters sent to the class in another constructor, for example 'params TypeLog[] logType'.

I've refactored all variable names, and haven’t repeated the names, such as DataBase -> RemoteServer. It is a bad practice to write DataBase -> DataBaseRemoteServer, because it is already in the database namespace.

Said the above, I used the name conventions that c# recommends for these cases, in each variable, method, enum, class, namespace names, etc.

I’ve separated database, filemanager, config and a common class (for common use basics) in different classes as well. I could have done different projects in the same solution, but it wasn’t necessary since I didn’t want to change the original class’s idea.

I put the validations in the correct order and improved them, if the program had been designed in another way, the System class would have been extended. Exception for sending custom or extended messages could be included, but I limited myself to adding properties with a boolean for Success or Failure in a general try catch and use the exceptions as the original JobLogger does: 'throw new LogFile.Exception("Error while...") '.

I’ve used enums for these kind of logs, and instead of repeating **if statements** so many times, I used some **switch statements**.

My aim is that after this process of refactoring and reorganization this Project can be maintained over time, and in the case a decision to extend it is made, it can be adapted with minimum changes and understood even by people that haven’t worked on it.

Finally the program is executed with a try-catch logging the exception in case there is a problem.