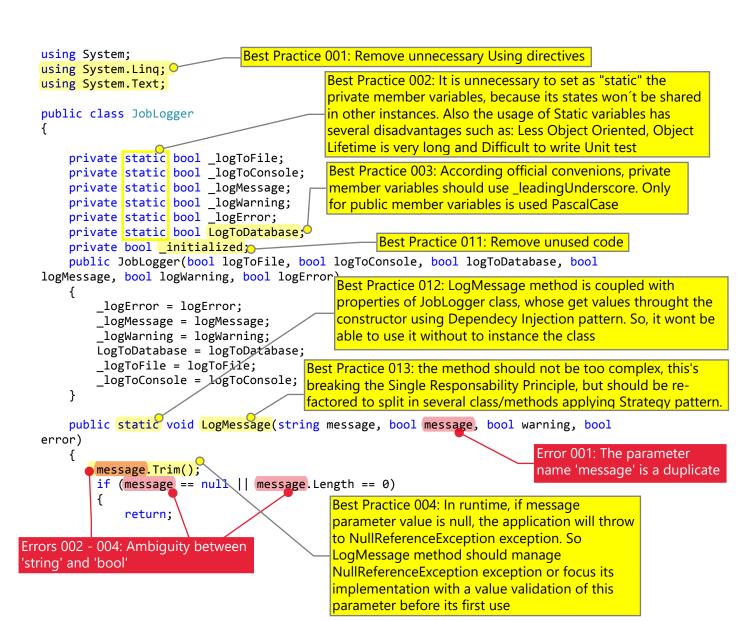# Code Review

Please review the following code snippet. Assume that all referenced assemblies have been properly included.

The code is used to log different messages throughout an application. We want the ability to be able to log to a text file, the console and/or the database. Messages can be marked as message, warning or error. We also want the ability to selectively be able to choose what gets logged, such as to be able to log only errors or only errors and warnings.

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

```csharp
using System;
using System.Linq;
using System.Text;

public class JobLogger
{
    private static bool _logToFile;
    private static bool _logToConsole;
    private static bool _logMessage;
    private static bool _logWarning;
    private static bool _logError;
    private static bool LogToDatabase;
    private bool _initialized;
    public JobLogger(bool logToFile, bool logToConsole, bool logToDatabase, bool logMessage, bool logWarning, bool logError)
    {
        _logError = logError;
        _logMessage = logMessage;
        _logWarning = logWarning;
        LogToDatabase = logToDatabase;
        _logToFile = logToFile;
        _logToConsole = logToConsole;
    }

    public static void LogMessage(string message, bool message, bool warning, bool error)
    {
        message.Trim();
        if (message == null || message.Length == 0)
        {
            return;
```

Best Practice 001: Remove unnecessary Using directives

Best Practice 002: It is unnecessary to set as "static" the private member variables, because its states won´t be shared in other instances. Also the usage of Static variables has several disadvantages such as: Less Object Oriented, Object Lifetime is very long and Difficult to write Unit test

Best Practice 003: According official convenions, private member variables should use _leadingUnderscore. Only for public member variables is used PascalCase

Best Practice 011: Remove unused code

Best Practice 012: LogMessage method is coupled with properties of JobLogger class, whose get values throught the constructor using Dependecy Injection pattern. So, it wont be able to use it without to instance the class

Best Practice 013: the method should not be too complex, this's breaking the Single Responsability Principle, but should be re-factored to split in several class/methods applying Strategy pattern.

Error 001: The parameter name 'message' is a duplicate

Errors 002 - 004: Ambiguity between 'string' and 'bool'

Best Practice 004: In runtime, if message parameter value is null, the application will throw to NullReferenceException exception. So LogMessage method should manage NullReferenceException exception or focus its implementation with a value validation of this parameter before its first use

```csharp
        }
        if (!_logToConsole && !_logToFile && !LogToDatabase)
        {
            throw new Exception("Invalid configuration");
        }
        if ((!_logError && !_logMessage && !_logWarning) || (!message && !warning
    && !error))
        {
            throw new Exception("Error or Warning or Message must be specified");
        }

        System.Data.SqlClient.SqlConnection connection = new
    System.Data.SqlClient.SqlConnection(System.Configuration.ConfigurationManager.AppS
    ettings["ConnectionString"]);
        connection.Open();
        int t;
        if (message && _logMessage)
        {
            t = 1;
        }
        if (error && _logError)
        {
            t = 2;
        }
        if (warning && _logWarning)
        {
            t = 3;
        }
        System.Data.SqlClient.SqlCommand command = new
    System.Data.SqlClient.SqlCommand("Insert into Log Values('" + message + "', " +
    t.ToString() + ")");
        command.ExecuteNonQuery();

        string l;
        if
    (!System.IO.File.Exists(System.Configuration.ConfigurationManager.AppSettings["Log
    FileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt"))
        {
            l =
    System.IO.File.ReadAllText(System.Configuration.ConfigurationManager.AppSettings["
    LogFileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt");
        }
        if (error && _logError)
        {
            l = l + DateTime.Now.ToShortDateString() + message;
        }
        if (warning && _logWarning)
        {
            l = l + DateTime.Now.ToShortDateString() + message;
        }
        if (message && _logMessage)
        {
            l = l + DateTime.Now.ToShortDateString() + message;
        }
    System.IO.File.WriteAllText(System.Configuration.ConfigurationManager.AppSettings[
    "LogFileDirectory"] + "LogFile" + DateTime.Now.ToShortDateString() + ".txt", l);
```

Best Practice 010: Avoid too many conditions into if clause else there will technical debt after continuos inspection with SonarQube job

Errors 005 - 007: Ambiguity between 'string' and 'bool'

Best Practice 005: SqlConnection connection variable is never closed. This section should be wrapped with "using" statement, in order to acquire the SqlConnection as a resource, and to provide a simpler way to specify when the unmanaged resource is needed by your program, and when it is no longer needed.

Best Practice 008: In order to avoid Sql Injection Attack, SqlCommand sentence mustn´t contains SQL scripting else should be a call to some Store Procedure or any DDL

Best Practice 009: SqlCommand constructor should be based on the SqlConnection, and set up its CommandType and CommandTimeout properties

Error 008: Use of unassigned local variable 't'

Best Practice 014: If there isn´t the LogFile txt. it should be created before to try read it

Errors 009 - 011: Use of unassigned local variable 'l'

Errors 013 - 016: Ambiguity between 'string' and 'bool'

Best Practice 006: Apply Sintatic Sugar using lambda operator "+="

Best Practice 007: Apply Switch Case statement

Best Practice 015: Duplicated code should be re-factored as a string variable

Errors 012: Use of unassigned local variable 'l'

```csharp
        if (error && _logError)
        {
            Console.ForegroundColor = ConsoleColor.Red;
        }
        if (warning && _logWarning)
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
        }
        if (message && _logMessage)
        {
            Console.ForegroundColor = ConsoleColor.White;
        }
        Console.WriteLine(DateTime.Now.ToShortDateString() + message);
    }
}
```

Best Practice 007: Apply Switch Case statement

Errors 017 - 018: Ambiguity between 'string' and 'bool'