**Observaciones Prueba Log**

La clase JobLogger incluye seis atributos estáticos que establece el constructor de la clase, esto nos dará problemas porque los atributos estáticos serán compartidos por todas las instancias en la aplicación. Deberíamos transformar los atributos estáticos en atributos de Instancias eliminando la palabra clave estática. Además, el atributo LogToDatabase debe renombrarse como \_logToDatabase y el atributo \_initialized debe eliminarse de la clase porque no se usa ni se hace referencia a ella en la clase.

Los seis atributos booleanos se deben reemplazar por dos tipos de Enum: uno para almacenar el tipo de registro (consola, archivo y base de datos) y otro para la gravedad del mensaje (mensaje, advertencia y error).

**Método LogMessage:**

El primer problema que tiene este método es que los primeros dos parámetros tienen el mismo nombre, pero tienen diferentes tipos de datos, cadena y bool.

El mensaje de código. Trim () no valida si el mensaje de cadena es nulo, en caso de que el mensaje sea nulo arrojará una NullReferenceException.

Deberíamos gestionar excepciones personalizadas creando, por ejemplo, una clase LoggerException que sea hija de la clase de excepción para detectar o identificar que la excepción proviene de la clase JobLogger.

Deberíamos evitar usar nombres de variables como "L" o "T" porque esos nombres no expresan el significado o el propósito de las variables.

**Registro de DataBase:**

Deberíamos incluir "using" para evitar tener nombres largos como System.Data.SqlClient.SqlConnection y usar la palabra clave var para reducir la longitud del código. Por ejemplo, podríamos usar System.Data.SqlClient ;.

Para la cadena de conexión debemos usar ConfigurationManager.ConnectionStrings en lugar de ConfigurationManager.AppSettings y la declaración de conexión debe incluirse dentro de una keywork usando para cerrar la conexión cuando terminemos de usar la conexión. El comando no está asociado con la conexión, que emitirá una excepción. Deberíamos usar parámetros en la cadena de consulta para evitar Inyecciones de SQL y mantener nuestro código en mantenimiento.

**Registro de archivos:**

La condición Primero ejecutará el método ReadAllText cuando el archivo especificado no exista, esto arrojará una Excepción IO, además, la lógica es almacenar el contenido del Archivo en una variable de cadena llamada l, esto podría causar un uso de memoria alto o fuera de excepciones de memoria. Deberíamos usar el método File.AppendText para evitar este tipo de riesgos. Para el nombre del archivo de registro, debemos usar un formato de fecha como AAAAMMDD para evitar incluir barras invertidas en el nombre del archivo de registro.

Deberíamos usar una variable de ruta para almacenar la ruta del archivo y reutilizar esa variable dentro del FileLogger.

**Registro de consola:**

Deberíamos restablecer el color predeterminado de la consola después de llamar al comando ConsoleWriteLine para evitar mantener el último color utilizado en la consola.

**Código de Refactorización**

Para el código Refactoring se usó el Patrón de Estrategia porque tenemos Loggers con diferentes comportamientos para la acción LogMensage, para hacerlo tenemos que crear una Interfaz y clases específicas para implementar el método de Interfaz LogMensage. Si necesitamos agregar otro método de registro, podemos crear fácilmente otra clase que implemente la interfaz de ILogger.

En este caso, la clase de registrador es el contexto del patrón de estrategia donde AsignarTipoLog seleccionará el tipo de clase de registro que se utilizará.