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.

Errores y malas prácticas:

* No se deberían usar atributos, en vez de ello se deberían usar propiedades para acceder a los atributos de la clase.
* En el código presentado se utilizan atributos con una determinada nomenclatura (prefijo “\_”), sin embargo no todos los atributos presentan este formato: ***LogToDatabase*** debería cambiarse por ***\_logToDatabase***, en caso se desee permanecer con el uso y definición de estos atributos.
* El método ***LogMessage*** debería recibir como parámetros solamente el texto del mensaje y el tipo del mensaje (message, warning, o error), y no ser como en el ejemplo, ya que esto permite pensar que un mensaje podría ser de más de un tipo a la vez, lo cual no es correcto según lo solicitado para el ejercicio.
* El método ***LogMessage*** posee dos parámetros con el mismo nombre (***message***), lo cual presenta un error de compilación. Para corregir esto, se debe cambiar uno de los nombres de estos parámetros.
* La validación si el texto del mensaje es nulo o vacio debe realizarse con el método provisto por C# para dicha funcionalidad: ***string.IsNullOrEmpty***
* Probablemente, en caso el mensaje sea nulo o vacio, se debería lanzar una excepción indicándolo.
* La validación de que el mensaje sea de un tipo, no debería validar también que el objeto ***JobLogger*** permita el loggeo de al menos un tipo de mensaje, ya que esto último es parte de la configuración, por lo tanto podría ir junto con la validación que lanza la excepción “Invalid configuration”.
* Las variables auxiliares “t” y “l” deben de inicializarse. Además, en caso se mantenga su uso luego de realizar las correcciones, deben cambiar sus nombres por otros que representen mejor la funcionalidad o rol que van a realizar o cumplir.
* En la sección del loggeo hacia una base de datos, no se está validando realmente si se debe o no grabar en dicha base de datos; es decir, la aplicación intentara grabar en base de datos a pesar de que la configuración indique que no se debe grabar en base de datos.
* El loggeo o registro del mensaje en la base de datos sería mejor que grabe el mensaje y el tipo de mensaje, y no un numero (1, 2 o 3) para indicarlo.
* En general, la grabación hacia una base de datos debiera realizarse en una capa especial de persistencia de datos hacia una base de datos, junto con toda la lógica para abrir, preparar, y cerrar la conexión hacia dicha base de datos.
* En la sección del loggeo hacia un archivo de texto, no se está validando realmente si se debe o no grabar en dicho archivo de texto; es decir, la aplicación intentara grabar en el archivo de texto a pesar de que la configuración indique que no se debe grabar en un archivo de texto.
* La lógica de concatenación del nuevo mensaje en el archivo realmente está haciendo lo mismo para todos los casos; es decir, que al final siempre grabara lo mismo en el archivo de texto, sin importar la configuración.
* En general, aunque no tan importante como la persistencia hacia una base de datos, la grabación hacia archivos de texto debiera realizarse en una capa especial para dicho motivo, junto con la lógica para abrir, preparar, y cerrar el archivo de texto.
* En la sección del loggeo hacia la línea de comando, no se está validando realmente si se debe o no mostrarse el mensaje por línea de comandos; es decir, la aplicación intentara mostrar el mensaje por línea de comandos a pesar de que la configuración indique que no se debe loggear hacia la línea de comando.
* También sería recomendable que el loggeo hacia la línea de comando lo realice otra clase que sea implementada para este motivo.

Recomendaciones, buenas prácticas y refactoring:

* En general, y para efectos de este ejercicio, no vamos a construir una arquitectura que soporte las diferentes capas que podría tener una aplicación que contenga esta funcionalidad. Sabemos que, dependiendo de las características de cada aplicación, se requieren capas para distribuir la funcionalidad que soporte la presentación de información, la lógica de negocio, la persistencia de datos, el uso de servicios, la comunicación u orquestación entre las distintas capas, etcétera. Vamos a trabajar solamente con un proyecto, y distribuiremos la funcionalidad y las posibles capas en carpetas con la intención de concentrarnos en las características principales de este ejercicio.
* Vamos a utilizar propiedades en vez de atributos para los parámetros de configuración que permiten determinar si se loggeara o no los mensajes, advertencias y/o errores, así como determinar si se loggeara o no hacia base de datos, archivo de texto, y/o línea de comandos.
* Vamos a separar la funcionalidad que permita el loggeo hacia base de datos, archivo de texto, y línea de comandos, por lo cual necesitaremos propiedades que nos permitan obtener una instancia de cada uno de estos proveedores o gestores.
* En el constructor de la clase ***JobLogger*** vamos a inicializar estos proveedores.
* Vamos a crear una clase “***Message***” para encapsular el texto y el tipo del mensaje, y así poder controlar mejor y con mayor claridad las operaciones a realizar y validar sobre dicho mensaje.
* Vamos a crear un método “***ValidateMessageAndConfiguration***” que permita verificar que tanto el mensaje como los parámetros de configuración sean los correctos.
* Si el texto del mensaje es nulo o vacio, lanzaremos una excepción con el mensaje “*Message must contain some text*”.
* Si la configuración no permite el loggeo en ninguno de los medios o no permite el loggeo de ningún tipo de mensaje, lanzaremos una excepción con el mensaje “*Invalid configuration*”.
* Si el tipo del mensaje no es de ninguno de los tres tipos definidos, lanzaremos una excepción con el mensaje “*Type of message must be specified*”.
* Vamos a crear tres métodos, uno para controlar el loggeo en cada uno de los tres medios según la configuración. Dentro de cada uno de estos métodos, si se determina que debe realizarse el loggeo, se utilizará el proveedor o gestor respectivo para realizar dicho loggeo.
* Invocar a cada uno de estos métodos para cada medio de loggeo, se controlará validando si es que se debe loggear en cada medio según la configuración.
* En donde sea posible, se tratará de cambiar el uso de múltiples condiciones “*if*”, por el uso de sentencias “*switch*”. Particularmente esto lo utilizaremos para determinar qué tipo de mensaje se debe de loggear según el tipo de mensaje contenido en el objeto “Message” enviado como parámetro a cada método.
* En cada uno de los tres métodos de control de loggeo, así como en el método principal “Log”, vamos a utilizar variables auxiliares que nos permitan determinar si es que el loggeo se dio satisfactoriamente o no. Por lo tanto, cada uno de estos métodos devolverá un valor booleano que indicará con valor “*true*” si se realizó exitosamente el loggeo. Esto también nos servirá para controlar y realizar nuestras pruebas unitarias.
* El método “***LogMessageToFile***” controlará el loggeo hacia archivos de texto. En caso se determine que se debe realizar el loggeo, se invocará al proveedor “***FileServiceProvider***” que proveerá un método “***SaveLog***” que permita loggear en un archivo de texto.
* Esta clase “***FileManager***” y su interfaz “***IFileManager***” deberían encontrarse en una capa o proyecto especifico que se encargue de realizar esta tarea. Para efectos de este ejercicio, ambos serán colocados en la carpeta “***Managers***”.
* El método “***LogMessageToConsole***” controlará el loggeo hacia línea de comandos. En caso se determine que se debe realizar el loggeo, se invocará al proveedor “***ConsoleServiceProvider***” que proveerá un método “***ShowLog***” que permita loggear en la línea de comando.
* Esta clase “***ConsoleManager***” y su interfaz “***IConsoleManager***” deberían encontrarse en una capa o proyecto especifico que se encargue de realizar esta tarea. Para efectos de este ejercicio, ambos serán colocados en la carpeta “***Managers***”.
* El método “***LogMessageToDatabase***” controlará el loggeo hacia una base de datos. En caso se determine que se debe realizar el loggeo, se invocará al proveedor “***DatabaseServiceProvider***” que proveerá un método “***SaveLog***” que permita loggear en la base de datos.
* Esta clase “***DatabaseManager***” y su interfaz “***IDatabaseManager***” deberían encontrarse en una capa o proyecto especifico que se encargue de realizar esta tarea. Para efectos de este ejercicio, ambos serán colocados en la carpeta “***Managers***”.
* En el código presentado como solución, el código que registra en base de datos se encuentra comentado, indicando que se debe completar mediante una forma de manejo de persistencia de datos. Además, esto permite que nuestras pruebas unitarias que contemplen dicho escenario no den como resultado “*Failed*”.

Observaciones adicionales:

* Muchas buenas prácticas y patrones de diseño adicionales podrían aplicarse para mejorar y optimizar la funcionalidad y código presentados en este ejercicio. Mencionaré algunas buenas prácticas y patrones que no se han desarrollado en el código presentado como solución, pero que podrían aplicarse según las características y requerimientos de la aplicación que vaya a utilizar la clase JobLogger.
* Al desacoplar la funcionalidad de la aplicación en distintas capas y proyectos, sería recomendable utilizar alguna implementación de gestión de inyección de dependencia, como *Unity* o *Ninject*. Con esto solucionaríamos problemas como los de referencia circular, etcétera.
* Sería recomendable también, junto con el uso de inyección de dependencia, el utilizar “*Lazy Instantiation*”, de forma que la instanciación de los proveedores o gestores se realice justo al momento en que estos van a ser utilizados.
* Sería recomendable tener un mejor control del manejo de excepciones. Para esto podríamos utilizar *Enterprise Library* y así poder loggear las excepciones, así como poder determinar diferentes tipos de excepciones como de lógica de negocio, etcétera.