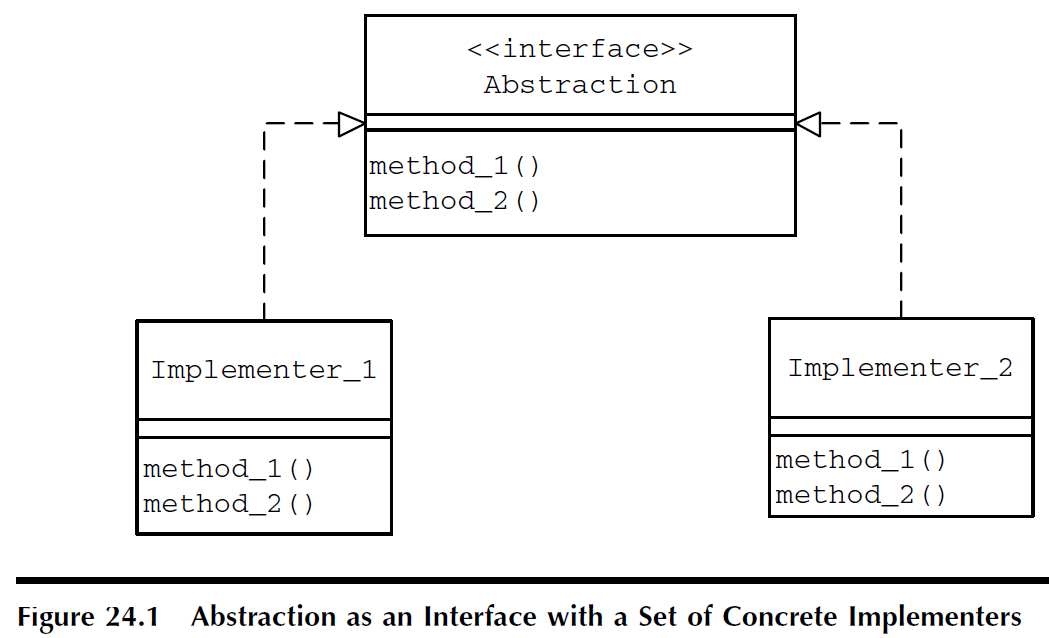
# Patrón Bridge

El patrón Bridge promueve la separación de la interfaz de una abstracción de su implementación.

* **Abstracción**: se refiere al proceso de identificar el conjunto de atributos y el comportamiento de un objeto que es específico para un uso particular.

Esta visión específica de un objeto puede diseñarse como un objeto separado omitiendo atributos y comportamiento relevantes. El objeto resultante en sí puede ser referido como una abstracción. Tenga en cuenta que un objeto dado puede tener más de una abstracción asociada, cada una con un uso distinto.

* Una abstracción dada puede tener una o más implementaciones para sus métodos (comportamiento). En términos de implementación, una abstracción puede diseñarse como una interfaz con uno o más implementadores concretos (Figura 24.1).



En la jerarquía de clases que se muestra en la Figura 24.1, la interfaz Abstracción declara un conjunto de métodos que representan el resultado de abstraer características comunes de diferentes objetos.

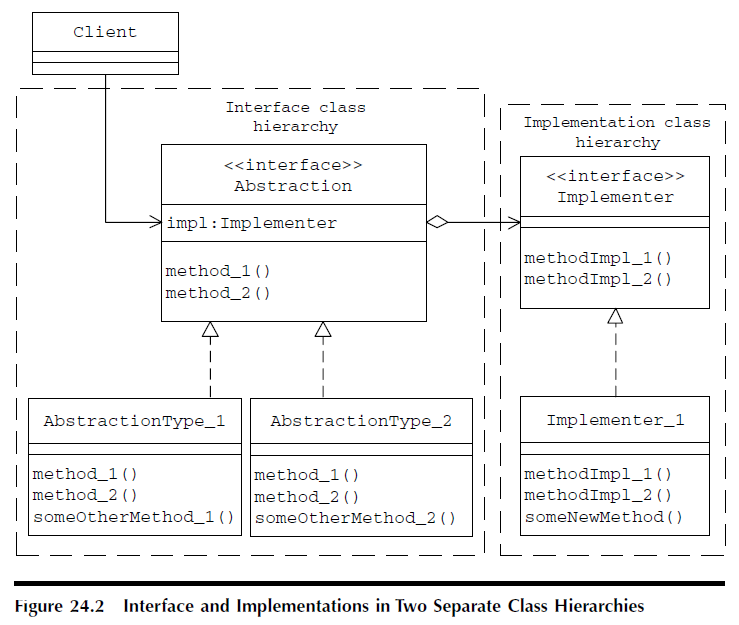
* Tanto el Implementer\_1 como el Implementer\_2 representan el conjunto de implementadores de abstracción.

Este enfoque sufre de las siguientes dos limitaciones:

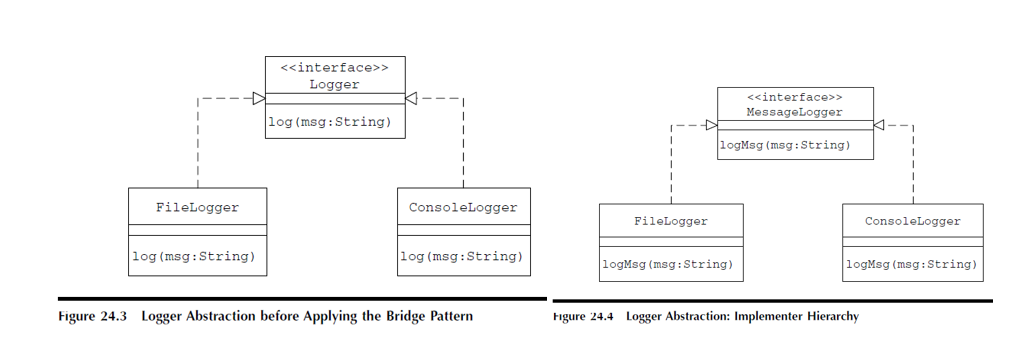
* + Cuando existe la necesidad de subclasificar la jerarquía por alguna otra razón, podría conducir a un número exponencial de subclases y pronto tendremos una explosión en la jerarquía de clases.
  + Tanto la interfaz de abstracción como su implementación están estrechamente vinculadas y, por lo tanto, no pueden modificarse independientemente sin afectarse entre sí.

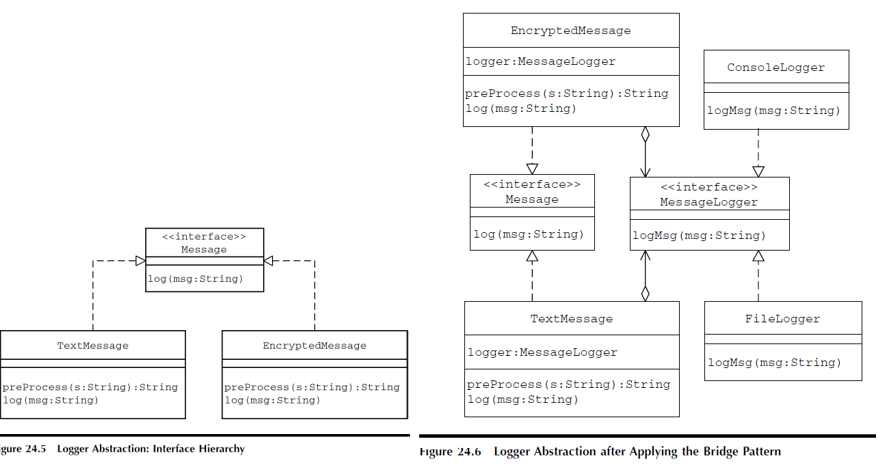
## Ventajas del patrón Bridge

* Lograr un diseño de abstracción más eficiente y manejable.
* El diseño de una abstracción utilizando el patrón Bridge separa sus interfaces de las implementaciones.
* Tanto las interfaces como las implementaciones de una abstracción se pueden colocar en jerarquías de clase separadas (Figura 24.2): la Abstracción mantiene una referencia de objeto del tipo Implementador.



* Una aplicación cliente puede elegir el tipo de abstracción deseado de la jerarquía de la clase **Abstracción**. El objeto de abstracción se puede configurar **con una instancia de un implementador** apropiado de la jerarquía de clases del Implementador.
* Esta capacidad de combinar dinámicamente abstracciones e implementaciones puede ser muy útil en términos **de ampliar la funcionalidad sin subclasificar**. El objeto **Abstracción** puede ofrecer cierta cantidad de procesamiento antes de reenviar la llamada al objeto **Implementador**.
* Este tipo de arreglo de clase desacopla completamente la interfaz y la implementación de una abstracción y permite que las clases en la interfaz y la jerarquía de implementación varíen sin afectarse entre sí.





### Aspectos destacados del diseño de las clases de interfaz de abstracción

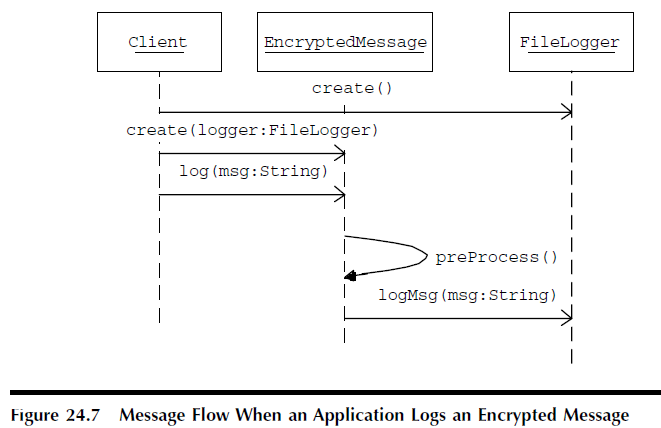
* **Logger abstraction interface classes — TextMessage and Encrypted-Message —:** no provea implementación para el servicio de registro de mensajes real.
* Las clases como FileLogger y Console-Logger en la jerarquía de **clases del implementador** de abstracción proporcionan la implementación real del registro de mensajes.
* Los objetos **cliente no usan** directamente la interfaz expuesta por las clases de **implementador de abstracción**.
* Cada clase de interfaz de abstracción mantiene una referencia de objeto del tipo MessageLogger (**implementador de abstracción**). Cada vez que un objeto cliente crea un objeto de interfaz de abstracción, configura el objeto de interfaz con un objeto MessageLogger.
* Siempre que un objeto de cliente invoca el método de registro en un objeto de interfaz de abstracción, el objeto de interfaz realiza el preprocesamiento requerido y utiliza el servicio de registro de mensajes del objeto MessageLogger que contiene.
* La funcionalidad de **preprocesamiento está destinada a ser utilizada internamente solo por los objetos de la interfaz de abstracción (private)** y no debe estar disponible para los objetos del cliente. Para garantizar esto, el método preProcess en las clases de interfaz de abstracción TextMessage y EncryptedMessage está diseñado como un método privado. Como parte de su implementación del método preProcess, EncryptedMessage cifra un mensaje entrante desplazando todos los caracteres a la derecha en una posición.

Como resultado de mantener la interfaz y la implementación de la abstracción del registro en dos jerarquías de clase separadas, las interfaces y las implementaciones de la abstracción del registrador están completamente desacopladas. Siempre que un cliente (Listado 24.5) necesita registrar un mensaje:

1. Crea una instancia de una clase de implementador de MessageLogger adecuada, como FileLogger o ConsoleLogger.
2. Crea una instancia de una clase de implementador “Message” adecuada, como TextMessage o EncryptedMessage.
3. Configura el objeto implementador “Message” con el objeto implementador MessageLogger creado en el Paso 1. Este objeto se mantiene dentro del objeto implementador de mensajes.
4. Llama al método log (String) en el objeto implementador de mensajes creado en el paso 2.
5. El objeto implementador de mensajes lleva a cabo el procesamiento requerido para transformar el mensaje entrante al formato deseado (encriptar el mensaje de entrada en el caso de EncryptedMessage) y reenvía el mensaje transformado al objeto implementador MessageLogger que contiene invocando su logMessage (String ) método.

* Esta relación entre las clases en la interfaz y la jerarquía de clases del implementador puede verse como un puente en este caso.

El flujo de mensajes cuando un objeto de la aplicación usa el tolog de abstracción del registrador, un mensaje cifrado se puede representar como en la Figura 24.7.



La separación de la interfaz de abstracción del registrador de su implementación permite que se modifiquen independientemente sin tener que modificar el otro. Después de implementar el diseño , si un objeto de cliente necesita registrar mensajes en un nuevo formato, digamos HTML, este requisito se puede abordar fácilmente diseñando una nueva clase de interfaz de abstracción de registrador HTMLMessage como implementador de la interfaz de mensajes existente. La clase HTMLMessage se puede diseñar para proporcionar el procesamiento requerido para transformar un mensaje entrante en texto HTML y utilizar una clase implementadora de abstracción para registrar realmente el mensaje transformado. Esta adición de una nueva clase de interfaz no afecta a los implementadores de abstracción existentes. Además, agregar una nueva clase para cada nuevo tipo de formato de mensaje mantiene el crecimiento lineal de la clase. De manera similar, una nueva implementación de abstracción de registrador, como los mensajes tolog DBLogger a una base de datos puede agregarse sin tener que modificar o subclasificar la jerarquía de clases.

## PATRÓN DE PUENTE VS PATRÓN DE ADAPTADOR

Similitudes:

* Tanto el patrón del adaptador como el patrón del puente son similares, ya que ambos trabajan para ocultar los detalles de la implementación subyacente del cliente.

Diferencias:

* El patrón **Adaptador** tiene como objetivo hacer que las clases funcionen juntas y que de otro modo no podrían debido a interfaces incompatibles. Un adaptador está destinado a cambiar la interfaz de un objeto existente. Un Adaptador requiere una clase de adaptador (existente), lo que indica que el patrón Adaptador es más adecuado para las necesidades después del diseño inicial del sistema.
* **El patrón Puente es más un patrón de tiempo de diseño.** Se utiliza cuando el diseñador tiene control sobre las clases en el sistema. Se aplica antes de que se haya implementado un sistema para permitir que tanto las interfaces de abstracción como sus implementaciones se varíen independientemente sin que se afecten entre sí.
* En el contexto del patrón Bridge, el problema de las interfaces incompatibles no existe. Los objetos de cliente siempre usan la interfaz expuesta por las clases de interfaz de abstracción. Por lo tanto, tanto el patrón Puente como el patrón Adaptador se utilizan para resolver diferentes problemas de diseño.