SEMINARIO

4

Ejercicio de diseño

Logger

(cont.)

Diseño del Software

Grado en Ingeniería Informática del Software

Curso 2022-2023

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

(Habíamos quedado aquí...)

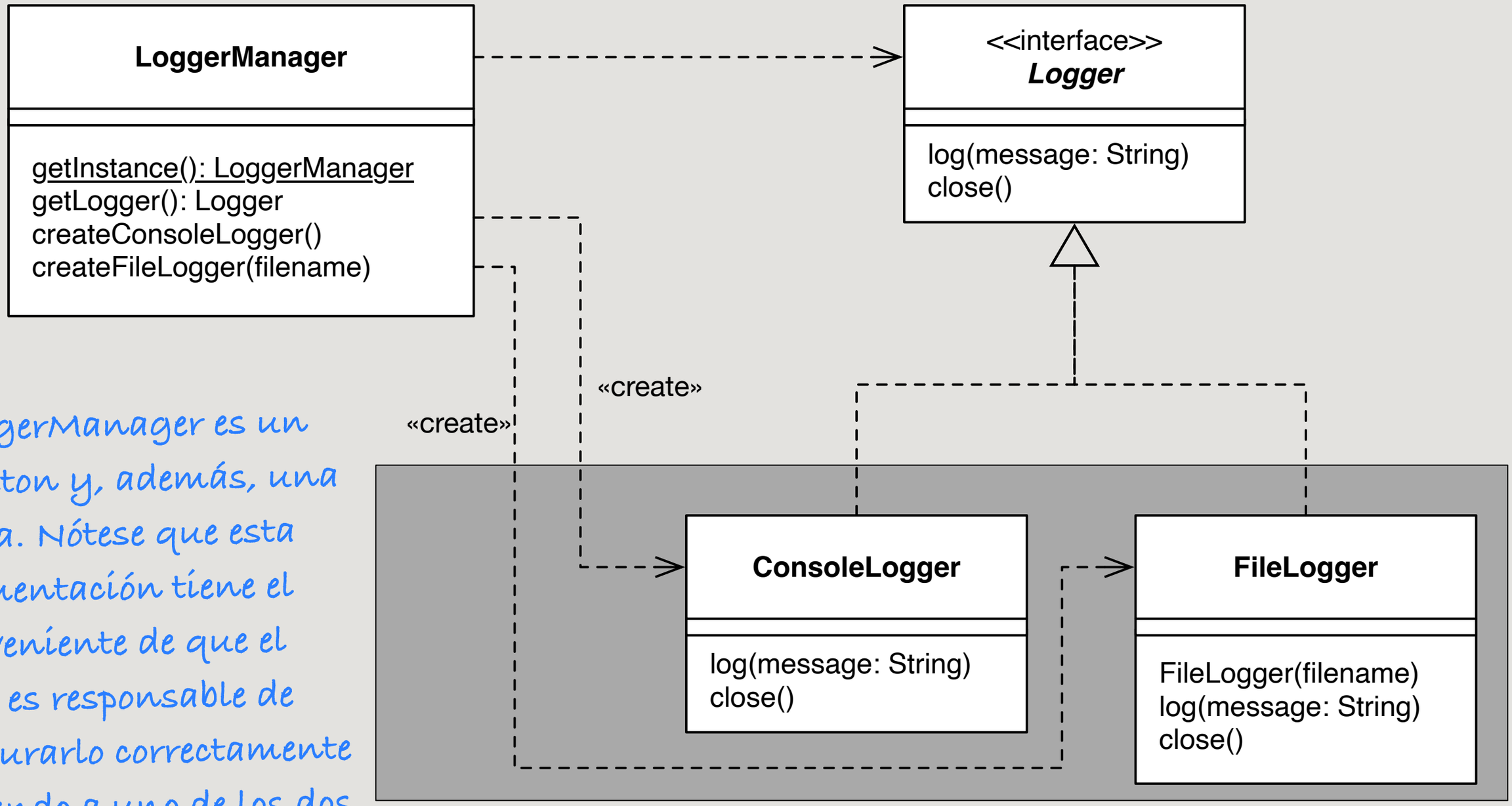
Dos tipos de logger

Introducimos una primera variante: ahora nos surge la necesidad de tener dos tipos de logger (de consola y fichero), manteniendo la misma restricción de que sólo haya un único objeto.

Dos tipos de logger

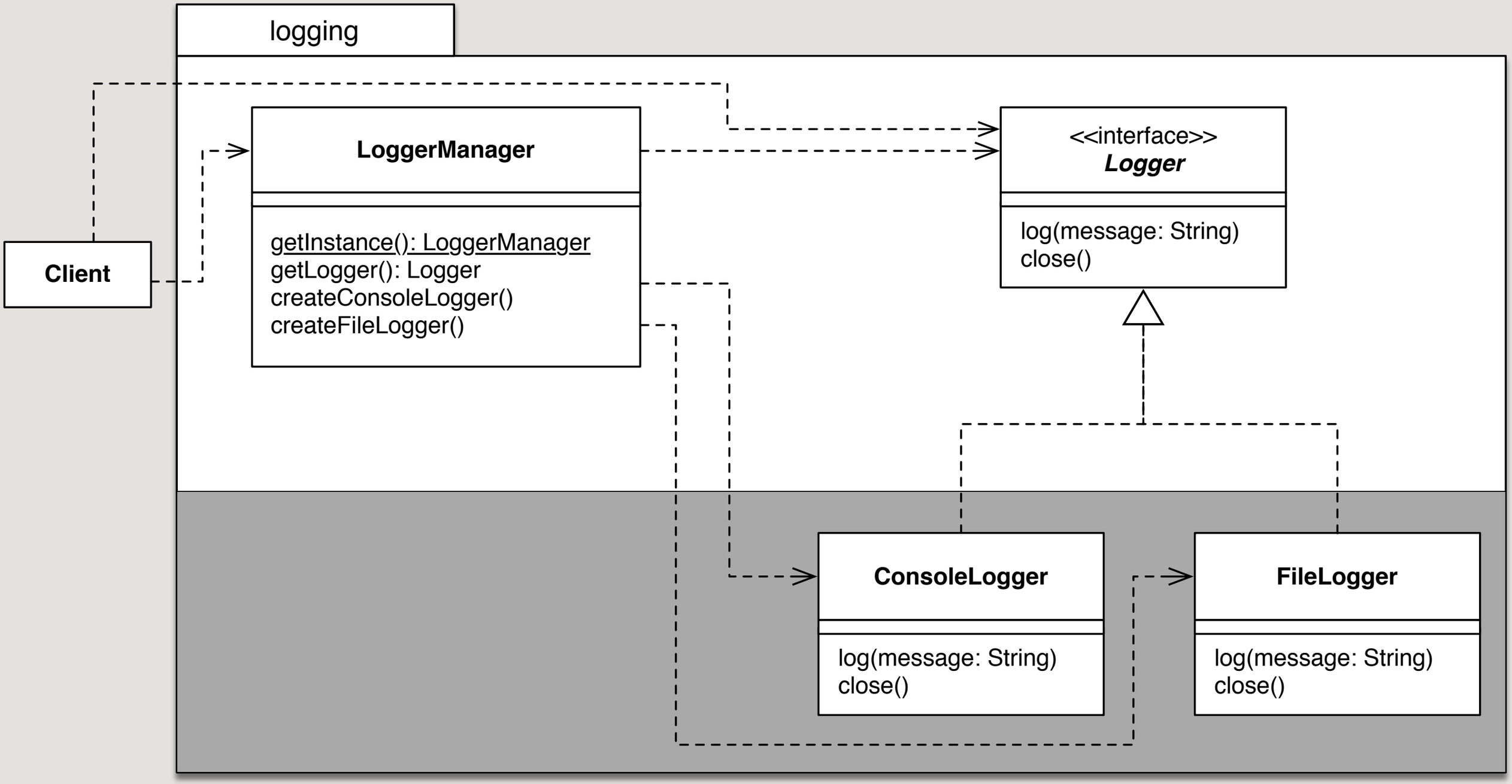
- De consola y fichero
- Una vez decidido el tipo de logger y creado éste, ya no se podrá cambiar
- Se mantiene el requisito de que la instancia del logger creado sea única durante toda la ejecución del programa

Plantead el diseño…



El LoggerManager es un Singleton y, además, una factoría. Nótese que esta implementación tiene el inconveniente de que el cliente es responsable de configurarlo correctamente (llamando a uno de los dos métodos create...) antes de llamar por primera vez a getLogger. Pero también se podría haber leído un fichero de propiedades.

A ConsoleLogger y FileLogger podemos darles visibilidad de paquete para que no sean accesibles desde fuera del paquete «logging»; si no, tendríamos que hacer sus constructores no públicos, para evitar que se pudiesen crear objetos directamente de ellos.


logging
LoggerManager
getInstance(): LoggerManager
getLogger(): Logger
createConsoleLogger()
createFileLogger()
Client
<<interface>>
Logger
log(message: String)
close()
ConsoleLogger
log(message: String)
close()
FileLogger
log(message: String)
close()

Consideraciones

- **En el diseño anterior, ¿es `LoggerManager` algún patrón de diseño [GoF]?**
  - No; llamamos así a cualquier clase que crea y devuelve objetos de otras clases
- **¿Es necesaria?**
  - No; podría encargarse la propia clase `Logger`
  - Aquí lo hacemos así para abstraer la lógica de creación

Configuración de la factoría

- **¿Qué enfoques hay para crear inicialmente el logger del tipo adecuado?**
  - Métodos de creación para cada tipo de logger
    - `public [static] Logger createFileLogger(…)`
    - `public [static] Logger createConsoleLogger(…)`
  - Un método parametrizado
    - `public [static] Logger createLogger(…)`
  - Un fichero de propiedades

Un tercer tipo de logger

Se produce un nuevo cambio en los requisitos: ahora nos piden que haya también un logger en formato HTML.

Logger HTML

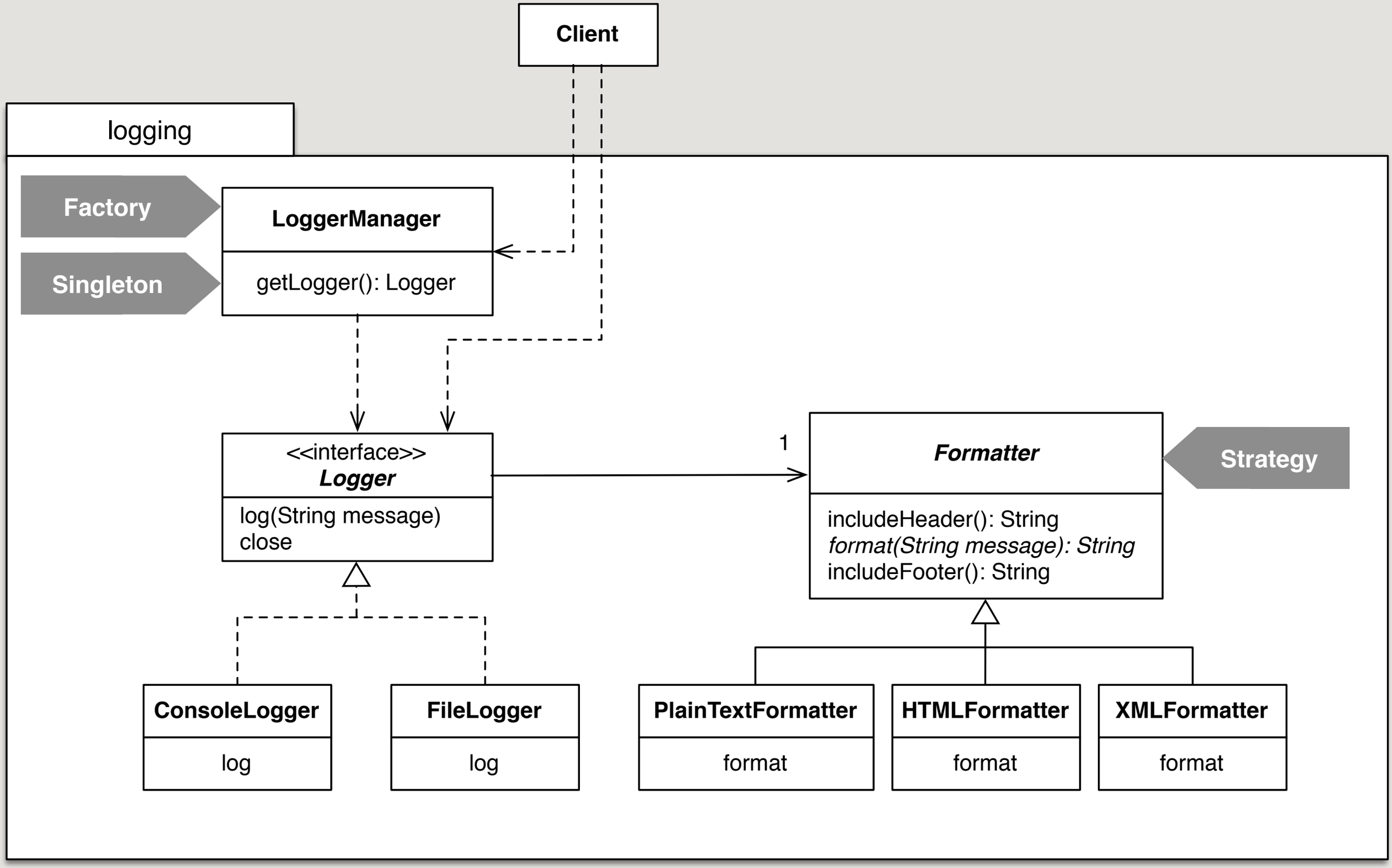
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Logger</title>
</head>
<body>
    <h1>Logger</h1>
    <ul>
        <li>¡Hola, mundo!</li>
        <li>Segundo mensaje</li>
        <li>Y otro más...</li>
    </ul>
</body>
</html>
```

¿Cómo lo haríais?

Pista

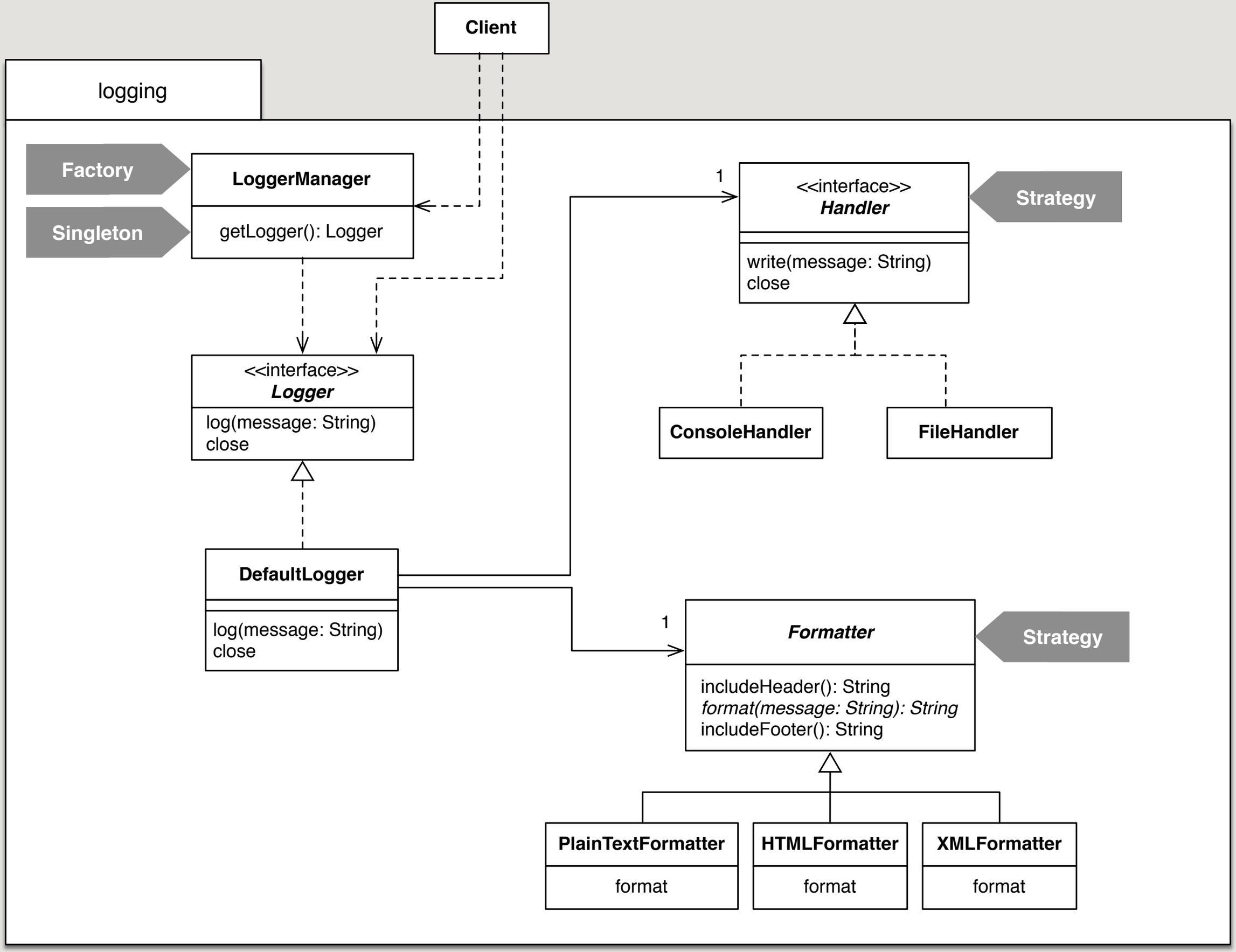
Identificar y aislar el concepto que varía.

- **¿Qué puede variar aquí?**
  - ¿No podrían surgir nuevos tipos de logger que escribiesen en una base de datos relacional, o en un servidor remoto, a través de la red?
  - ¿Y otros formatos como XML?
  - Más aún: ¿no podría tener sentido tener un logger de base datos pero en HTML? ¿O en XML?


Client
logging
Factory
Singleton
LoggerManager
getLogger(): Logger
<<interface>>
Logger
log(String message)
close
1
Formatter
includeHeader(): String
format(String message): String
includeFooter(): String
Strategy
ConsoleLogger
log
FileLogger
log
PlainTextFormatter
format
HTMLFormatter
format
XMLFormatter
format

Tres tipos de logger

- **Es decir, realmente aquí estábamos mezclando dos conceptos independientes:**
  - El dispositivo de salida
    - Consola, fichero, BD, red...
  - El formato
    - Sin formato, HTML, XML...
- **Otra posibilidad sería sacar fuera también, como otra estrategia, el dispositivo de salida**


Client
logging
Factory
Singleton
LoggerManager
getLogger(): Logger
<<interface>>
Logger
log(message: String)
close
DefaultLogger
log(message: String)
close
1
<<interface>>
Handler
write(message: String)
close
Strategy
ConsoleHandler
FileHandler
1
Formatter
includeHeader(): String
format(message: String): String
includeFooter(): String
Strategy
PlainTextFormatter
format
HTMLFormatter
format
XMLFormatter
format

Ahora podéis echar un vistazo al módulo de «logging» de la API de Java y ver cómo está diseñado.