Universidad de Buenos Aires
Facultad de Ingenieria

Tecnicas de Diseño

Catedra Pantaleo

Trabajo Practico 1.1

Logger

Grupo 20

Alberto Schicht 85267
Sergio Zelechowski 86651

Interfaz de la API

Se provee al cliente con dos formas de crear un objeto Logger.

- Logger()
- Logger(Config config)

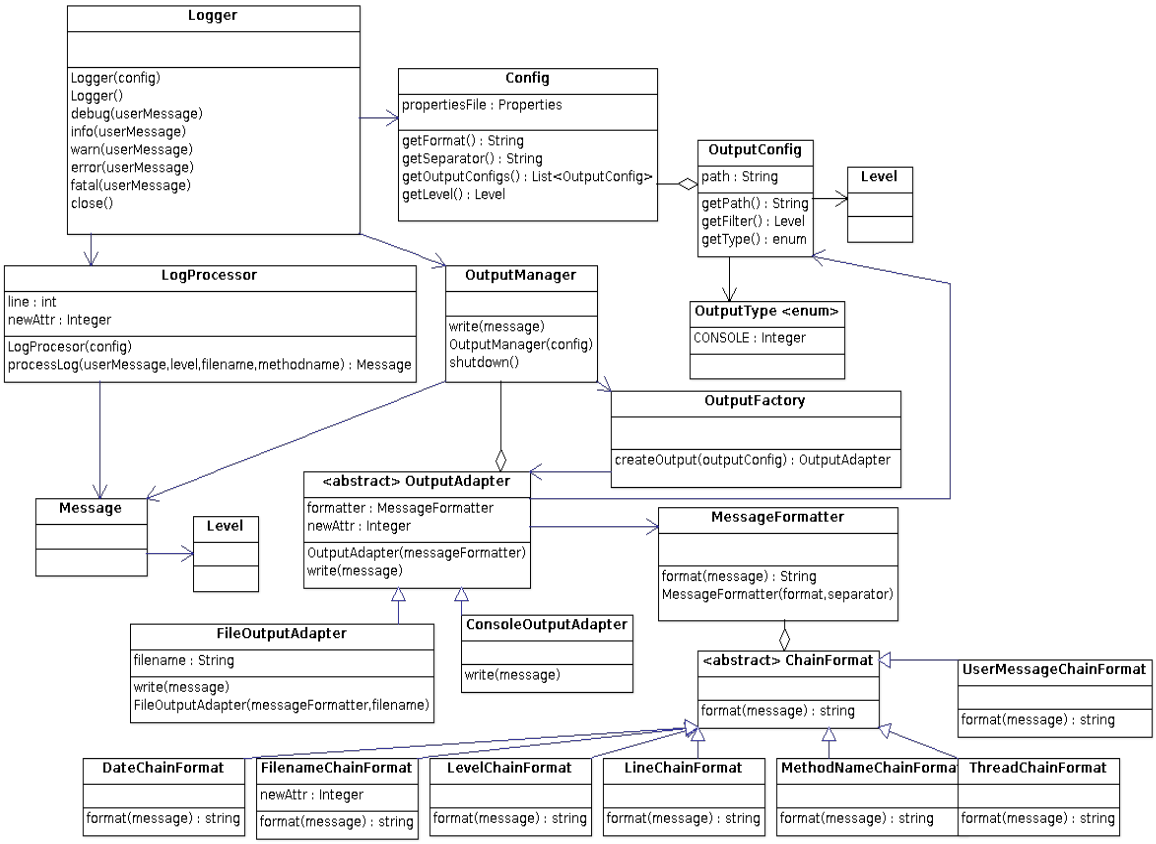
El primer constructor usa la configuración por defecto. El segundo usa un objeto Config que el usuario provee y que crea pasandole la ruta del archivo properties que desea usar. Se decidió encapsular la configuración dentro de un objeto para permitir que sea más extensible en el caso de que se quieran agregar configuraciones aparte del archivo properties.

El Logger sirve como único punto de acceso y facilita así la interfaz para el cliente. Dicha interfaz es muy simple para el cliente, solo debe invocar al método deseado según el nivel de error que desea plasmar (debug > info > warn > error > fatal > off), pasando el mensaje como único parámetro, por ej:

```
Logger logger = new Logger(config);
logger.info("Este es el mensaje que se desea loggear.");
```

El Logger se encargará de llamar al LogProcessor, el cual procesa el mensaje y añade datos como nombre del método, fecha, número de línea, thread Id, etc. y enviarlo al OutputManager encargado de aplicarle el formato al mensaje y gestionar el tipo de salida definido en el Config. Para obtener el nombre del método, clase y archivo desde el cual se origina el mensaje se decidió utilizar Thread.getStackTrace() evitando así que el cliente sea el encargado de suministrar esos datos, simplificando el uso de la app.

Diagrama de clases



Clases

Logger: Es la interfaz para con el cliente. Centraliza el uso de la aplicación.

LogProcessor: Se encarga de procesar cada mensaje, decide cual procesar en función del Level, obtiene otros datos necesarios para armar el objeto Mensaje, el cual viaja al resto de las clases. Se concentra y aísla esta lógica del Logger para hacerlo extensible y testeable.

Config: Se decidió encapsular la configuración dentro de un objeto para permitir que sea más extensible en el caso de que se quieran agregar configuraciones aparte del archivo properties.

OutputConfig: Encapsula la configuración de un medio de loggueo.

OutputType: Determina el tipo de medio en el que se va a escribir. Se prefirió usar un enumerado en una clase aparte puesto heredar y crear clases configuracion de archivo y de salida estándar era hilar demasiado fino sobre una entidad (outputConfig) que es improbable que cambie en su comportamiento (aunque si en su tipo).

Message: Encapsula y modela un mensaje de logueo con toda la información que puede tener (nombre de método y clase, nro de línea, fecha, etc)

Level: Se lo trabaja internamente como un enumerado, centraliza y facilita su uso de esta manera.

OutputManager: Es necesario que una entidad se encargue de distribuir los mensajes a los diferentes medios en los que se va a loguear. Esta clase contiene y manda mensajes a los objetos de tipo OutputAdapter.

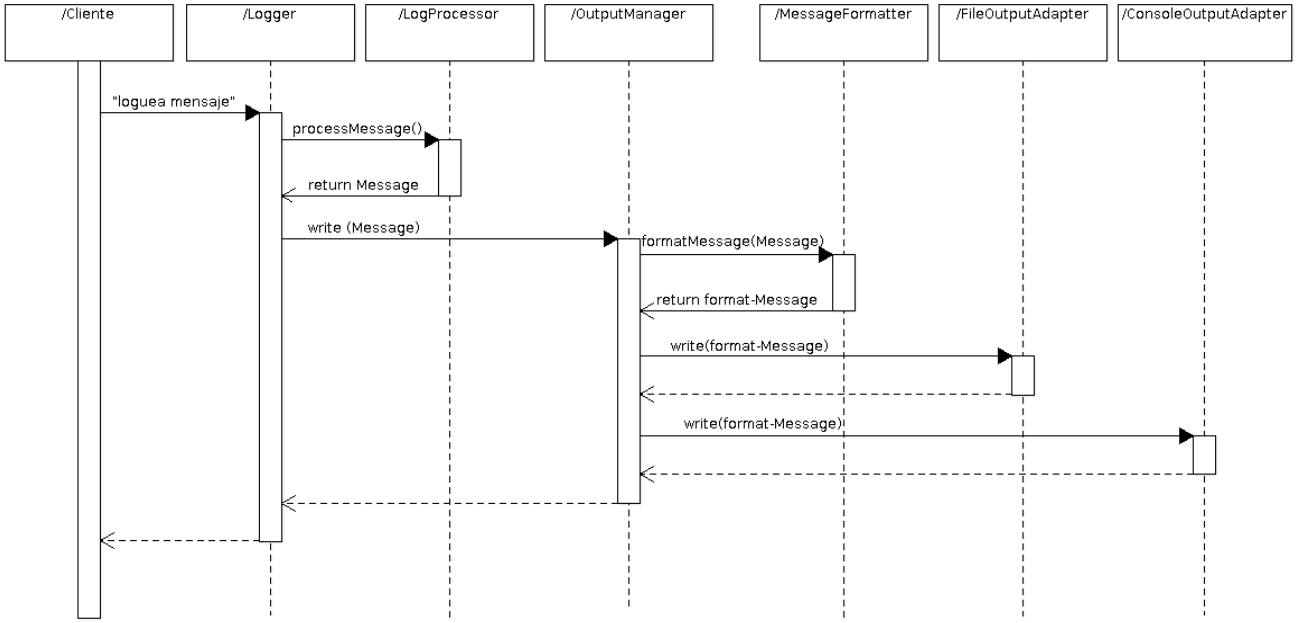
OutputAdapter y sus descendientes: Conocen como escribir sus respectivas salidas y administrar los recursos que implican, cada clase del adapter conoce sus responsabilidades.

MessageFormatter: Da formato a los mensajes, tenerlo en una clase separada lo hace fácilmente extensible.

OutputFactory: Esta clase encapsula la creacion de los outputAdapters.

ChainFormatter y sus descendientes: Este conjunto de clases implementan un patron builder para permitir formatear de manera dinamica los mensajes. Cada clase desendiente conoce como formatear una parte del mensaje a loggear.

Diagrama de secuencia de un Proceso de log



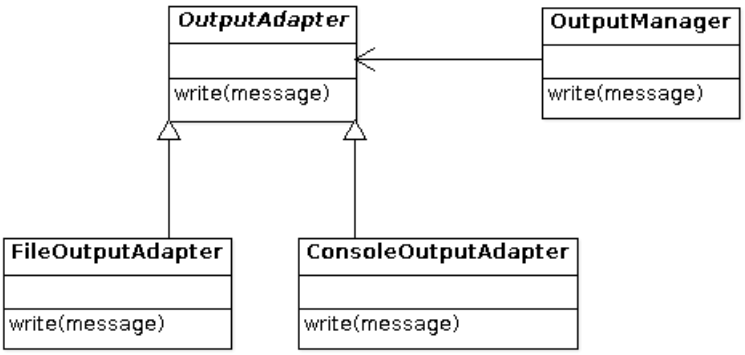
Puede verse como a partir de un mensaje que el cliente envía, se desarrolla la comunicación e interacción de las diferentes clases.

Patrones de diseño usados

Adapter

Fuente: http://en.wikipedia.org/wiki/Adapter_pattern

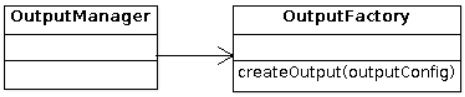
Tanto la salida estándar como un archivo tienen una interfaz determinada para su uso, mediantes el uso del patrón adapter para abstraernos de como se escribe un mensaje en un medio (archivo o salida estándar) y permite que el agregado de un nuevo medio (como por ejemplo el puerto de un servidor) sea fácil y no impacte en el resto del código.



Factory Method

Fuente: http://en.wikipedia.org/wiki/Factory_method_pattern

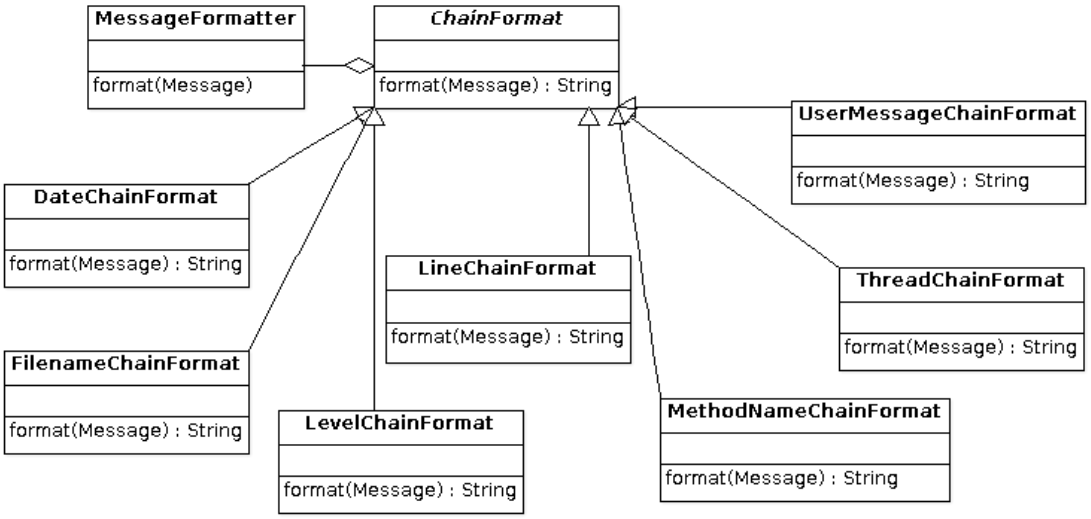
Si bien no se uso un Factory method puro, la creación de los objetos de tipo outputAdapter se delega a una clase con un método encargado de crearlos. De esta manera se aísla el posible cambio en la forma de crear outputs o el agregado de uno nuevo.



Builder

Fuente: http://en.wikipedia.org/wiki/Builder_pattern

El formato de un mensaje de log no se conoce en tiempo de compilación sino que depende del archivo de configuración. Para manejar la complejidad de un formato variable y al mismo tiempo tener una herramienta de logueo extensible ante nuevos cambios se uso el patrón builder.



Uso de herencia

Se usó herencia en dos puntos.

- OutputAdapter
- ChainFormat

En estos dos casos se usaron como parte de los patrones de diseño adapter y builder.