Utilización de Logger en los ejercicios de la asignatura

Este proyecto ilustra el funcionamiento del Logger que vamos a utilizar durante todo el curso. Un logger sirve para redireccionar a uno o varios dispositivos información sobre el funcionamiento del programa. Típicamente esa información consistirá en trazas de error, y esos dispositivos podrán ser la consola, ficheros, e incluso podría ser la propia base de datos.

Visualizar estos mensajes de error por consola está más orientado a monitorizar el programa en tiempo de desarrollo, mientras que su almacenamiento en un dispositivo persistente (p.e., ficheros o una tabla de la base de datos) es sobretodo útil en explotación, para conocer la traza de funcionamiento del sistema antes de que ocurriese un error o un comportamiento inesperado.

Los mensajes de error se estructuran por niveles. Por ejemplo, los que son del nivel ERROR son más importantes que los del nivel WARN, estos a su vez más importantes que los del nivel INFO y estos más importantes que los de nivel DEBUG etc ... De tal forma que cuando en nuestro código hagamos una llamada al logger, le proporcionaremos de alguna manera, en esa misma llamada, el nivel del error. Por ejemplo, si el logger es el objeto ***l***:

```
l.debug("Esta es una notificación de nivel debug");
l.info("Esta es una notificación de nivel info");
l.warn("Esta es una notificación de nivel warn");
l.error("Esta es una notificación de nivel error");
```

De esta forma, más adelante, típicamente a través de un fichero de configuración que use el proyecto, le podemos decir, por ejemplo, que sólo salgan por pantalla los de la categoría mayor o igual que WARN (ERROR y WARN), pero que por ejemplo en el fichero de log se guarden todos los mensajes de todas las categoría.

El uso del Logger nos debería obligar a abandonar la usual técnica de "*siembra de printfs*" cuando queremos depurar un programa. En lugar de llenarlo todo de *printfs* que luego borramos o comentamos, podremos mantener aquellos más útiles asociándolos a un nivel bajo (típicamente **debug**), que sólo visualizaríamos durante la depuración. Para ello, cambiaríamos el fichero de configuración de manera que lo permitiese. Cuando acabase la depuración, podríamos volver a restaurar el fichero de configuración pidiendo que sólo se registrasen, por ejemplo, las notificaciones de un nivel igual o superior a INFO. De esta forma ocultaríamos temporalmente esas notificaciones DEBUG.

SLF4J

SLF4J no es un logger en si mismo, sino una "***fachada***" que permite invocar distintas implementaciones de logger. De esta manera, si programamos con SLF4J y el logger del fabricante Pepito, y el año que viene queremos cambiar al logger del fabricante Menganito, no tendríamos que cambiar ni una línea de código. Simplemente cambiaríamos los jars de una fabricante por los de otro[1].

El jar puente que hace la traducción de las llamadas entre SLF4J y el fabricante de nuestro logger puede que esté entre los JARs de SLF4J (por ejemplo, típicamente el correspondiente al logger que integra el JDK), o podría ser provisto por el propio fabricante del logger (por ejemplo Apache suministra el jar puente de su logger Log4J versión 2).

Aunque, SLF4J soporta algún nivel más, los niveles que son compatibles con el logger que vamos ausar (Log4J), son los 4 del ejemplo anterior:

- **DEBUG** es el más bajo y lo usaremos sólo para depuración (p.e., imprimir valores de variables).
- **INFO** lo usaremos típicamente para mensajes notificando un comportamiento normal (p.e., conexión establecida OK, fichero procesado correctamente, etc ...)
- **WARN** para avisos, normalmente corresponden con circunstancias extrañas pero que no necesariamente sean un error. Quizás es conveniente tener una traza de esas circunstancias por si más tarde hay un error analizar si tiene algo que ver con el WARN (p.e., la consulta de clientes ha devuelto 0 filas, o no se ha actualizado ninguna fila, etc ...)
- **ERROR** indica que se ha producido alguna excepción.

Log4J

Log4J es un logger implementado por Apache. Su fichero de configuración deberá de estar en alguna parte del class path, nosotros lo pondremos siempre colgando de la carpeta SRC para que al compilar el proyecto Eclipse lo copie actualizado a la carpeta BIN.
Hay varios formatos de fichero de configuración posibles (properties, YAML, JSON y XML).
Nosotros este curso utilizaremos un fichero *.properties* como el del proyecto
(ver **src\log4j.properties**).
En nuestro caso ese fichero está preparado para mostrar por pantalla todas las notificaciones, , mientras que en el fichero de log guardamos de INFO en adelante. **Esto no es lo más adecuado. Lo normal es que (al menos en explotación) la información en el fichero sea más detallada que en consola. Por ejemplo, que en consola sólo salgan los mensaje de nivel ERROR, mientras que en el fichero de**

1 En nuestro caso, con hacer ese cambio de jars en la user_library, el cambio se propagaría a todos los proyectos que tengan esa user_library en su build path,

log guardamos todo (i.e., notificaciones DEBUG o superiores). Pero puedes cambiarlo, simplemente editando estas dos líneas del fichero **log4j.properties**:

1. Cambia `log4j.appender.theConsoleAppender.Threshold=DEBUG` por `log4j.appender.theConsoleAppender.Threshold=ERROR`
2. Cambia `log4j.appender.theFileAppender.Threshold=INFO` por `log4j.appender.theFileAppender.Threshold=DEBUG`

Si eres de esos alumnos que nunca miran el fichero log, y que sólo se fijan en los errores que salen por consola, no hace falta que hagas estos cambios.
También puedes desactivar una de las dos salidas poniendo `Threshold=OFF` en la línea correspondiente.
En nuestros proyectos el fichero de logger se llamará **log4j.log** y estará en la capeta **res**, como puedes comprobar en la línea `log4j.appender.theFileAppender.File=res/log4.log`. Si no tienes el fichero aún creado, Log4J te lo crearía por primera vez.
Como no nos corresponde en esta asignatura entrar en el detalle del fichero de configuración, con estas breves indicaciones debería ser suficiente para el trabajo con los proyectos de la asignatura.

¿Cómo se instancia el logger desde SLF4J?

Pidiendoselo a la clase LoggerFactory de SLF4J (ver imports en Prueba.java). Por ejemplo:

```
l =    LoggerFactory.getLogger(Prueba.class);
```

instancia un *logger* llamado l que estará ativo en la clase propia ***Prueba.class***. Por tanto, cuando instanciemos cualquier objeto que queramos sea rastreado por el logger, deberíamos de incluir una sentencia de este tipo con el nombre de la clase a la que pertenece ese objeto.
El logger suele ser una variable estática de la clase, por lo que habitualmente la inicializaremos en su declaración, evitando así que no esté inicializada para su uso en métodos estáticos de dicha clase.
Por ejemplo, así:

```
private static Logger l = LoggerFactory.getLogger(Prueba.class);
```