Logging

DigitalHouse>
Coding School

Certified Tech Developer
The Ultimate Degree

¿Qué necesitamos para loguear?

Para poder loguear necesitamos agregar un archivo de configuración. Lo agregaremos en el root del proyecto, es decir, en la carpeta principal. Por ejemplo, si nuestro proyecto tiene el nombre app, en esa carpeta creamos un archivo de configuración con el nombre **log4j.properties** con el siguiente contenido:

¿Qué necesitamos para loguear?

En la primera línea estamos indicando el nivel mínimo de logging y los appenders que vamos a emplear. En este caso usaremos un nivel de logging establecido en DEBUG y creamos dos appenders, stdout y file.

```
log4j.rootLogger=DEBUG, stdout, file
```

La segunda línea sirve para configurar en qué nivel se empezaran a mostrar las advertencias tanto por consola como a almacenarse en el fichero.

```
log4j.logger.infoLogger=DEBUG
```

¿Qué necesitamos para loguear?

Y, con la tercera línea evitamos que los appenders hereden la configuración de sus appenders padres, en caso de que los hubiera (en el nuestro, sería el appender principal así que no tenemos ese problema).

```
log4j.additivity.infoLogger = false
```

Crear la configuración para imprimir mensajes por consola

En la primera línea indicamos qué tipo de logger será, haciendo referencia a la clase que imprimirá los mensajes (¡Recordá la sección **Appenders**!).

```
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
```

En la segunda línea, le decimos que queremos imprimirlo directamente por la consola.

```
log4j.appender.stdout.Target=System.out
```

Crear la configuración para imprimir mensajes por consola

Y las dos últimas líneas son para configurar la plantilla que tendrá cada mensaje. Podés ver todas las posibles opciones de configuración en la página de ayuda de Oracle.

```
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.stdout.layout.ConversionPattern=[%d{yyyy-MM-dd HH:mm:ss}] [ %-5p] [%c{1}:%L] %m%n
```

El mensaje de salida sería algo tal que así:

```
[2018-08-03 11:48:39] [ INFO ] [App:29] Esto es una prueba desde App class
```

Configurar el appender

En estas líneas vamos a hacer exactamente lo mismo que antes pero configurando el appender para que salga a través de un fichero.

En la primera línea configuramos la clase como RollingFileAppender, lo que significa que se crearán distintos ficheros al cumplirse determinadas condiciones que tratamos en las siguientes líneas.

```
log4j.appender.file=org.apache.log4j.RollingFileAppender
```

En la siguiente línea indicamos el nombre (con ruta incluida) que queremos que tenga nuestro fichero de log.

```
log4j.appender.file.File=avisos.log
```

Configurar el appender

Con MaxFileSize establecemos el tamaño máximo que tendrá nuestro fichero, y con MaxBackupIndex indicamos cuántos archivos podemos tener usando el mismo log. A partir de llegar al máximo, comenzarán a sobrescribirse empezando por el más antiguo.

```
log4j.appender.file.MaxFileSize=5MB
```

Y por último, al igual que por consola, indicamos que plantilla tendrán nuestros mensajes.

```
log4j.appender.file.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.file.layout.ConversionPattern=[%d{yyyy-MM-dd
HH:mm:ss}] [ %-5p] [%c{1}:%L] %m%n
```

Concluyendo...

La clase que hemos visto nos da la posibilidad de loguear distintos tipos de logueo, que van de la mano de qué tipo de información queremos mostrar.

Los tipo de logs nos dan información más exacta de qué está pasando, por ejemplo si usamos:

```
logger.info("Este es un mensaje solo con informacion");
```

Vemos que solo nos sirve para agregar un comentario de qué pasó en ese momento, por lo tanto, no nos da mucho valor.

Concluyendo...

Por otro lado, si agregamos algo como:

```
logger.error("El usuario es invalido", e);
```

Claramente el logs nos dice que tenemos un error, y la letra ‘e’ como segundo parámetro nos va a imprimir en el log, el stacktrace del error. Un ejemplo sería:

```
Exception in thread "main" java.lang.NullPointerException at
com.example.myproject.Book.getTitle(Book.java:16) at
com.example.myproject.Author.getBookTitles(Author.java:25) at
com.example.myproject.Bootstrap.main(Bootstrap.java:14)
```

Podemos observar que el error principal fue NullPointerException y vemos donde se generó.

Concluyendo...

Niveles de registro

Como vimos anteriormente, los logs levels son: FATAL, ERROR, WARN, INFO, DEBUG, TRACE y ALL.

Ahora podremos ver la visibilidad, dependiendo de cuál elegimos en:

```
log4j.logger.infoLogger=DEBUG
```

Niveles de registro (X=Visible)

	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
FATAL	X						
ERROR	X	X					
WARN	X	X	X				
INFO	X	X	X	X			
DEBUG	X	X	X	X	X		
TRACE	X	X	X	X	X	X	
ALL	X	X	X	X	X	X	X


DigitalHouse>
Coding School