

# Técnicas de programación segura

2ºDAM IoT

# Introducción

En esta unidad vamos a ver:

- Validación de entradas con expresiones regulares
- Ficheros de log con la librería Log4j.

# Logs

# Logs

Los ficheros de registro o logs, permiten almacenar información sobre las distintas acciones que realiza tu aplicación.

Estos ficheros permiten realizar un seguimiento detallado de lo que ocurre en el sistema.

Java incorpora la librería **java.util.logging** que permite utilizar la clase `Logger` para llevar un control de todos los eventos que ocurren a lo largo de la ejecución de la aplicación.

Sin embargo, este logger es muy simple y existen librerías externas que nos permiten una mayor configuración.

# Log4j 2

**log4j versión 2** es un framework que ofrece una forma jerárquica de insertar sentencias de log dentro de una aplicación Java.

```
<dependency>
```

```
    <groupId>org.apache.logging.log4j</groupId>
```

```
    <artifactId>log4j-api</artifactId>
```

```
    <version>2.22.1</version>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>org.apache.logging.log4j</groupId>
```

```
    <artifactId>log4j-core</artifactId>
```

```
    <version>2.22.1</version>
```

```
</dependency>
```

## Log4j 2

El fichero de configuración log4j2.xml se guardará la información de qué mensajes de log interesan y dónde y cómo se visualizarán.

Utilizaremos el formato xml aunque Log4j 2 permite guardar esta configuración en otros tipos de formatos.

El fichero lo guardaremos en un directorio llamado resources.

# Log4j 2: Niveles de log

Los niveles normales que puede tener un logger son, de menor a mayor prioridad:

- **TRACE:** Se usa para información más detallada que el nivel debug.
- **DEBUG:** Se utiliza para mensajes de información detallada que son útiles para debugear una aplicación.
- **INFO:** Se utiliza para mensajes de información que resaltan el progreso de la aplicación de una forma general.
- **WARN:** Se utiliza para situaciones que podrían ser potencialmente dañinas.
- **ERROR:** Se usa para eventos de error que podrían permitir que la aplicación continúe ejecutándose.
- **FATAL:** Se usa para errores muy graves, que podrían hacer que la aplicación dejara de funcionar.

Además hay otros dos niveles especiales que son:

- **ALL:** Tiene el nivel más bajo posible y se usa para activar todo el logging.
- **OFF:** Tiene el nivel más alto posible y se usa para evitar cualquier mensaje de log

# Log4J 2 Java

En cada clase se añadirá:

```
private static final Logger log = LogManager.getLogger(NewMain.class);
```

Impresión de los mensajes de log. Le indicaremos la cadena a guardar y llamaremos al método correspondiente con el nivel de log del mensaje.

```
log.trace("mensaje de trace");
```

```
log.debug("mensaje de debug");
```

```
log.info("mensaje de info");
```

```
log.warn("mensaje de warn");
```

```
log.error("mensaje de error");
```

```
log.fatal("mensaje de fatal");
```



# Log4j 2: Fichero de configuración

“**Appenders**” define los distintos tipos de salidas y su formato. Podemos tener más de una. El atributo “**name**” se utiliza para darle un nombre a cada tipo de salida y poder seleccionar la que se utilizará.

**Console** es una etiqueta que nos indica que la salida del log es la consola.

**PatternLayout** con el formato de la salida.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

**Loggers** define qué salida se utiliza de las declaradas en Appenders en el atributo **ref**. En “**level**” se indica el nivel de error que nos interesa guardar en el log.

## Log4j 2

Ejemplo de salida para el fichero de configuración anterior: Se muestran en consola todos los mensajes cuyo nivel de seguridad es superior a info.

Los mensajes se muestran en el formato dado mezclados con otros mensaje propios del programa.

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBra
18:56:02.540 [main] INFO    com.mycompany.ficheroLog.NewMain - mensaje de info
18:56:02.549 [main] WARN    com.mycompany.ficheroLog.NewMain - mensaje de warn
18:56:02.549 [main] ERROR   com.mycompany.ficheroLog.NewMain - mensaje de error
18:56:02.549 [main] FATAL   com.mycompany.ficheroLog.NewMain - mensaje de fatal
Un mensaje en el programa
18:56:02.551 [main] INFO    com.mycompany.ficheroLog.Prueba - un info hola
18:56:02.551 [main] WARN    com.mycompany.ficheroLog.Prueba - un warning
18:56:02.551 [main] ERROR   com.mycompany.ficheroLog.Prueba - un error
```

## Log4j 2

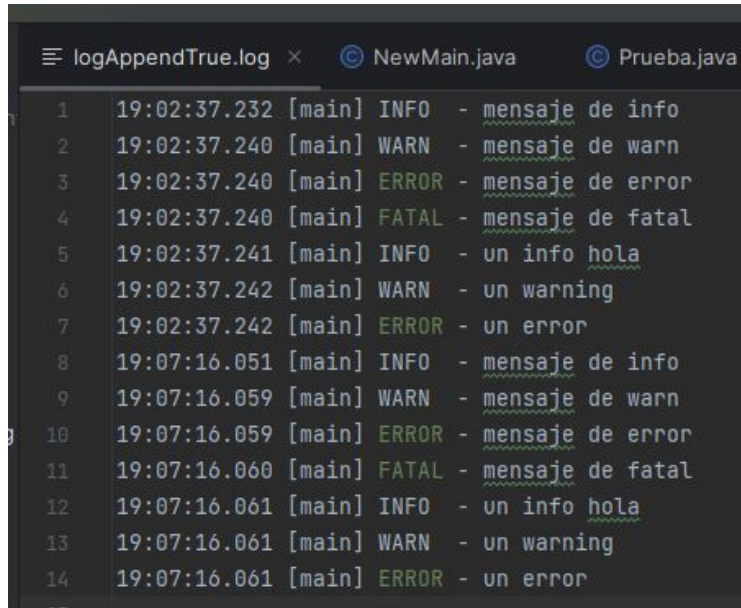
Puede ser interesante guardar los ficheros en un fichero de log en vez de visualizarse por consola. Para ello se crea otro appenders con la etiqueta **File**.

- **name**: Nombre del appender para seleccionarlo desde Loggers.
- **fileName**: Nombre del fichero de log.
- **append**: si se crea un fichero nuevo o se añade la información al ya existen.

```
<File name="FileAppendTrue" fileName="logs/logAppendTrue.log" append="true">
  <PatternLayout>
    <Pattern>%d{HH:mm:ss.SSS} [%t] %-5level - %msg%n</Pattern>
  </PatternLayout>
</File>
```

# Log4j 2

La salida de la configuración anterior después de ejecutar dos veces el programa podría ser algo como esto.

A screenshot of an IDE's console window showing log output. The window has three tabs: 'logAppendTrue.log' (active), 'NewMain.java', and 'Prueba.java'. The log output consists of 14 lines, each with a line number, a timestamp, the thread name '[main]', a log level, and a message. The log levels are INFO, WARN, ERROR, and FATAL, each appearing four times in sequence. The messages are in Spanish and include underlined words like 'mensaje', 'info', 'warn', 'error', and 'fatal'.

```
logAppendTrue.log x NewMain.java Prueba.java
1 19:02:37.232 [main] INFO - mensaje de info
2 19:02:37.240 [main] WARN - mensaje de warn
3 19:02:37.240 [main] ERROR - mensaje de error
4 19:02:37.240 [main] FATAL - mensaje de fatal
5 19:02:37.241 [main] INFO - un info hola
6 19:02:37.242 [main] WARN - un warning
7 19:02:37.242 [main] ERROR - un error
8 19:07:16.051 [main] INFO - mensaje de info
9 19:07:16.059 [main] WARN - mensaje de warn
10 19:07:16.059 [main] ERROR - mensaje de error
11 19:07:16.060 [main] FATAL - mensaje de fatal
12 19:07:16.061 [main] INFO - un info hola
13 19:07:16.061 [main] WARN - un warning
14 19:07:16.061 [main] ERROR - un error
```

## Log4j 2

También es posible guardar el log en formato html. Para ello en vez de **PatternLayout** dentro de una etiqueta File o Console, escribimos :

```
<Console name="ConsoleHTML" target="SYSTEM_OUT">  
  <HTMLayout>  
  </HTMLayout>  
</Console>
```

# Ejemplo

En el aula virtual tienes un proyecto llamado FicheroLog con el fichero log4j2.xml con diferentes configuraciones para que pruebes distintas opciones cambiando el nivel de log a guardar.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
    <File name="FileAppendTrue" fileName="logs/logAppendTrue.log" append="true">
      <PatternLayout>
        <Pattern>%d{HH:mm:ss.SSS} [%t] %-5level - %msg%n</Pattern>
      </PatternLayout>
    </File>
    <File name="FileAppendFalse" fileName="logs/logAppendFalse.log" append="false">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
    </File>
    <Console name="ConsoleHTML" target="SYSTEM_OUT">
      <HTMLLayout>
      </HTMLLayout>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="FileAppendTrue"/>
    </Root>
  </Loggers>
</Configuration>
```