Tema 2

Logger

Tema 2
Registro de traza de ejecución con la clase Logger

1. Introducción
2. Objeto Logger
3. Niveles de registro
4. Generar mensajes o trazas
5. Salida de mensajes en archivos

1.- Introduccion

- La clase **Logger** permite crear mensajes para el seguimiento o registro de la ejecución de una aplicación
  - Uso de depuración de la aplicación
  - Registro de estados
  - Visualización de los valores de variables
- La clase Logger ofrece la ventaja de poder generar la salida en ficheros de diferentes formatos (XML,HTML,XHTML etc…)

2.- Clase Logger

- Un objeto **Logger** implementa el patron Singleton
  - No tiene constructor
  - Obtenemos un objeto Logger mediante la llamada a un método estático **getLogger()**

    static Logger getLogger(String name)
    - **name :** es el nombre de la clase que está utilizando el logger precedida con el nombre del paquete

```
Logger.getLogger("paquete.NombreClase")
```

    - Generalmente los objetos logger se almacenan como objetos constantes dentro de la clase

```
final Logger LOG = Logger.getLogger("paquete.NombreClase");
```

2.- Clase Logger

```
//Usando la constante creada anteriormente
LOG.log(________);
//O tomando cada vez el objeto con getLogger()
Logger.getLogger("paquete.NombreClase").log(________);
//O recogiendo el nombre completo de la clase con getName()
Logger.getLogger(NombreClase.class.getName()).log(________);
```

3.- niveles de registro de mensajes

- Cada mensaje generado con la clase Logger debe tener asignado un nivel de importancia
- Niveles: De mayor importancia a menos:
  - SEVERE: Nivel de mensaje indicando un error serio.
  - WARNING: Indica un error potencial.
  - INFO: Para mensajes informativos.
  - CONFIG: Usado con mensajes relacionados con la configuración.
  - FINE: Proporciona información de traza de la ejecución.
  - FINER: Información de traza más detallada.
  - FINEST: Información de traza muy detallada.
- Para establecer estos niveles usamos la clase Level de Java,
  - Tiene una serie de constantes estáticas relacionadas con estos niveles: Level.SEVERE, Level.WARNING, etc.

4.- Generar mensajes o trazas

- La clase Logger ofrece el método **log()** para poder generar los mensajes de registro. El formato más sencillo es:
  - void log(Level level, String msg)
  - 1ª parámetro:
    - el nivel de importancia del mensaje usando las constantes de la clase Level
  - 2º parámetro
    - el mensaje en sí.

```
LOG.log(Level.SEVERE, "Se ha producido un error");
```

4.- Generar mensajes o trazas

- La propia clase Logger dispone de métodos para cada uno de los niveles.
  - void severe(String msg)
  - void warning(String msg)
  - void info(String msg)
  - void config(String msg)
  - void fine(String msg)
  - void finer(String msg)
  - void finest(String msg)

```
LOG.severe("Se ha producido un error");
```

5.- Salida de mensajes en archivos

- Los mensajes de Logger aparecen por defecto en la pantalla.
- El objetivo final de un sistema log es almacenar los mensajes en ficheros
- Para ello usamos el método **addHandler** de la clase logger, que permite añadir nuevos manipuladores de los mensajes
- Tipos de handler:
  - ConsoleHandler:
    - Muestra los mensajes a través de System.err que puede ser la salida estándar.
  - FileHandler:
    - Permite guardar los mensajes en archivos.
  - SocketHandler:
    - Puede enviar los mensajes a través de la red.

5.- Salida de mensajes en archivos

- FileHandler:
  - Genera por defecto un fichero de tipo XML

```
FileHandler fileXml = new FileHandler("Logging.xml");
LOG.addHandler(fileXml);
```

  - Si queremos un fichero de texto plano necesitamos un formateador

```
FileHandler fileTxt = new FileHandler("Logging.txt");
SimpleFormatter formatterTxt = new SimpleFormatter();
fileTxt.setFormatter(formatterTxt);
LOG.addHandler(fileTxt);
```