LOGS EN JAVA

Los niveles de log son una forma de clasificar los mensajes que genera una aplicación según su importancia o severidad. En una aplicación Java, al utilizar frameworks de logging (como SLF4J junto con Logback, Log4j2, entre otros), se define un umbral o nivel mínimo a registrar, y solo los mensajes que cumplen o exceden ese nivel se escriben en el log. A continuación, se describe el funcionamiento de los principales niveles:

* **DEBUG:**  
  Se utiliza para mostrar detalles que ayudan a depurar la aplicación. Incluye datos sobre el flujo de ejecución, valores de variables o el inicio y final de métodos críticos. Este nivel es útil en ambientes de desarrollo o pruebas.
* **INFO:**  
  Indica eventos importantes que ocurren durante el funcionamiento normal de la aplicación, como el inicio o la finalización de servicios, o la realización exitosa de operaciones. Es adecuado tanto en desarrollo como en producción para tener una visión general del comportamiento de la aplicación.
* **WARN:**  
  Registra situaciones inesperadas o potencialmente problemáticas que, aunque no impiden el funcionamiento, podrían derivar en errores si no se revisan. Se usa para indicar anomalías o condiciones atípicas.
* **ERROR:**  
  Se utiliza para registrar errores serios que afectan el flujo normal de la aplicación. Este nivel indica que ha ocurrido un fallo y, normalmente, incluye información de la excepción y su traza para facilitar la resolución del problema.

Al configurar el logger en una aplicación Java se establece un nivel de umbral; por ejemplo, si se configura en INFO, se registrarán los mensajes de INFO, WARN y ERROR, mientras que los de DEBUG y TRACE quedarán suprimidos. Esto ayuda a controlar la cantidad de información registrada, optimizando el rendimiento y facilitando la revisión de logs según el entorno en el que se despliegue la aplicación.

Dentro del código, definir y ubicar correctamente cada nivel de log es clave para obtener información útil sin sobrecargar la salida. Aquí algunas recomendaciones específicas:

1. **DEBUG:**
   1. **Cuándo:** Úsalo para detalles finos del flujo de ejecución o para valores de variables en secciones complejas o de difícil depuración.
   2. **Dónde:** En el inicio y fin de métodos críticos, en bloques condicionales que puedan afectar la lógica y justo antes de operaciones cuya falla sea difícil de reproducir.
   3. **Ejemplo:**  
      logger.debug("Iniciando procesamiento en el método X con parámetros {}", parametros);
2. **INFO:**
   1. **Cuándo:** Indica eventos importantes en el flujo normal de la aplicación, como la inicialización, finalización de procesos o confirmación de operaciones clave.
   2. **Dónde:** Al iniciar y completar tareas significativas, o cuando se realizan operaciones que el usuario final podría esperar que se registren (como loguear una sesión).
   3. **Ejemplo:**  
      logger.info("Servicio Y iniciado correctamente");
3. **WARN:**
   1. **Cuándo:** Registra situaciones anómalas que no detienen la ejecución pero que pueden indicar posibles problemas futuros, como entradas inesperadas o condiciones de degradación.
   2. **Dónde:** En bloques catch donde se manejen excepciones que se puedan recuperar o en situaciones donde se detecta un comportamiento no estándar.
   3. **Ejemplo:**  
      logger.warn("La configuración del parámetro Z no es la recomendada, se usará el valor por defecto");
4. **ERROR:**
   1. **Cuándo:** Para errores que afectan el funcionamiento normal de la aplicación y que generalmente requieren atención inmediata.
   2. **Dónde:** Dentro de bloques catch para capturar excepciones que impiden la correcta ejecución, registrando tanto el mensaje como el stack trace para facilitar el diagnóstico.
   3. **Ejemplo:**  
      logger.error("Error al procesar la solicitud", exception);

Adicionalmente, es recomendable que:

* Los mensajes sean claros y contengan suficiente contexto (como identificadores de transacción o usuario) para facilitar la trazabilidad.
* Se evite la sobreabundancia de logs en niveles superiores (INFO, WARN y ERROR) en código crítico para no saturar la salida en producción.
* La ubicación de los logs se mantenga coherente en toda la aplicación, estableciendo un patrón o guía para que otros desarrolladores sigan las mismas convenciones.

Siguiendo estas pautas, lograrás un registro de eventos más útil y manejable durante el ciclo de vida de la aplicación.