Tema 4: Hilos 2

1. Estados de un hilo.

Los diferentes estados en los que se puede encontrar un hilo son los siguientes:

- **Nuevo:** Se ha creado un nuevo hilo, pero aún no está disponible para su ejecución.
- **Ejecutable:** El hilo está preparado para ejecutarse. Puede estar Ejecutándose, siempre y cuando se le haya asignado tiempo de procesamiento, o bien que no esté ejecutándose en un instante determinado en beneficio de otro hilo, en cuyo caso estará Preparado.
- **No Ejecutable o Detenido:** El hilo podría estar ejecutándose, pero hay alguna actividad interna al propio hilo que se lo impide, como por ejemplo una espera producida por una operación de Entrada/Salida (E/S). Si un hilo está en estado "No Ejecutable", no tiene oportunidad de que se le asigne tiempo de procesamiento.
- **Muerto o Finalizado:** El hilo ha finalizado. La forma natural de que muera un hilo es finalizando su método run().

El método getState() de la clase thread, permite obtener en cualquier momento el estado en el que se encuentra un hilo. Devuelve por tanto: NEW, RUNNABLE, NO RUNNABLE o TERMINATED.

1. Iniciar un hilo.

Para que el hilo se pueda ejecutar, debe estar en el estado "Ejecutable" y para conseguir ese estado es necesario iniciar o arrancar el hilo mediante el método start() de la clase thread().

El método start() realiza las siguientes tareas:

- Crea los recursos del sistema necesarios para ejecutar el hilo.
- Se encarga de llamar a su método run() y lo ejecuta como un subproceso nuevo e independiente.

Algunas consideraciones importantes que debes tener en cuenta son las siguientes:

- Puedes invocar directamente al método run(), por ejemplo poner hilo1.run() y se ejecutará el código asociado a run() dentro del hilo actual (como cualquier otro método), pero no comenzará un nuevo hilo como subproceso independiente.
- Una vez que se ha llamado al método start() de un hilo, no puedes volver a realizar otra llamada al mismo método. Si lo haces, obtendrás una excepción IllegalThreadStateException.
- El orden en el que inicies los hilos mediante start() no influye en el orden de ejecución delos mismos, lo que pone de manifiesto que el orden de ejecución de los hilos es no-determinístico (no se conoce la secuencia en la que serán ejecutadas la instrucciones del programa).

2. Detener temporalmente un hilo.

Un hilo pasará al estado "No Ejecutable" o "Detenido" por alguna de estas circunstancias:

- **El hilo se ha dormido:** Se ha invocado al método sleep() de la clase thread, indicando el tiempo que el hilo permanecerá deteniendo. Transcurrido ese tiempo, el hilo se vuelve "Ejecutable", en concreto pasa a "Preparado".
- **El hilo está esperando:** El hilo ha detenido su ejecución mediante la llamada al método wait() y no se reanudará hasta que se produzca una llamada al método notify() o notifyAll() por otro hilo.
- **El hilo se ha bloqueado:** El hilo está pendiente de que finalice una operación de E/S en algún dispositivo, o a la espera de algún otro tipo de recurso.

El método suspend() (actualmente en desuso) también permite detener temporalmente un hilo, y en ese caso se reanudaría mediante el método resume() (también en desuso).

3. Finalizar un hilo.

La forma natural de que muera o finalice un hilo es cuando termina de ejecutarse su método run(), pasando al estado 'Muerto'. Una vez que el hilo ha muerto, no lo puedes iniciar otra vez con start(). Si en tu programa deseas realizar otra vez el trabajo desempeñado por el hilo, tendrás que:

- Crear un nuevo hilo con new().
- Iniciar el hilo con start().

Se puede utilizar el método isAlive() de la clase thread para comprobar si un hilo está vivo o no. Un hilo se considera que está vivo (alive) desde la llamada a su método start() hasta su muerte. IsAlive() devuelve verdadero (true) o falso (false), según que el hilo esté vivo o no.

El método stop() de la clase thread (actualmente en desuso) también finaliza un hilo, pero es poco seguro.

2. Gestión y planificación de hilos.

La ejecución de hilos se puede realizar mediante:

- **Paralelismo:** En un sistema con múltiples CPU, cada CPU puede ejecutar un hilo diferente.
- **Pseudoparalelismo:** Si no es posible el paralelismo, una CPU es responsable de ejecutar múltiples hilos.

La ejecución de múltiples hilos en una sola CPU requiere la planificación de una secuencia de ejecución (sheduling). El planificador de hilos de Java (Sheduler) utiliza un algoritmo de secuenciación de hilos denominado fixed priority scheduling que está basado en un sistema de prioridades relativas, de manera que el algoritmo secuencia la ejecución de hilos en base a la prioridad de cada uno de ellos.