1.- Estados de un hilo.

El ciclo de vida de un hilo comprende los diferentes estados en los que puede estar un hilo desde que se crea o nace hasta que finaliza o muere.

De manera general, los diferentes estados en los que se puede encontrar un hilo son los siguientes:

Nuevo (new): se ha creado un nuevo hilo, pero aún no está disponible para su ejecución.

Ejecutable (runnable): 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 (no runnable): 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 (terminated): 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.

1.1.- Iniciar un hilo.

Cuando se crea un nuevo hilo o thread mediante el método new(), no implica que el hilo ya se pueda ejecutar.

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().

En los ejemplos anteriores, recuerda que teníamos el código hilo1.start(); que precisamente se encargaba de iniciar el hilo representado por el objeto thread hilo1.

En realidad 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.

Es por esto último que cuando se invoca a start() se suele decir que el hilo está "corriendo" ("running"), pero recuerda que esto no significa que el hilo esté ejecutándose en todo momento, ya que un hilo "Ejecutable" puede estar "Preparado" o "Ejecutándose" según tenga o no asignado tiempo de procesamiento.

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).

1.2.- Detener temporalmente un hilo.

¿Qué significa que un hilo se ha detenido temporalmente? Significa que el hilo ha pasado al estado "No Ejecutable".

Y ¿cómo puede pasar un hilo al estado "No Ejecutable"? 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, pasará a "Ejecutable" (en concreto "Preparado") hasta que se produzca una llamada al método notify() o notifyAll() por otro hilo. Estudiaremos detalladamente estos métodos de la clase Object cuando veamos la sincronización y comunicación de hilos.

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; ha sido bloqueado por el sistema operativo. Cuando finaliza el bloqueo, vuelve al estado "Ejecutable", en concreto "Preparado".

El método suspend() (actualmente en desuso o deprecated) también permite detener temporalmente un hilo, y en ese caso se reanudaría mediante el método resume() (también en desuso). No debes utilizar estos métodos, de la clase thread ya que no son seguros y provocan muchos problemas. Te lo indicamos simplemente porque puede que encuentres programas que aún utilizan estos métodos.

1.3.- 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().

Y ¿hay alguna forma de comprobar si un hilo no ha muerto?
No exactamente, pero puedes 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.

Cuando el método isAlive() devuelve:

False: sabemos que estamos ante un nuevo hilo recién "creado" o ante un hilo "muerto".

True: sabemos que el hilo se encuentra en estado "ejecutable" o "no ejecutable".

El método stop() de la clase thread (actualmente en desuso) también finaliza un hilo, pero es poco seguro. No debes utilizarlo. Te lo indicamos aquí simplemente porque puede que encuentres programas utilizando este método.

1.4.- Ejemplo. Dormir un hilo con sleep.

¿Por qué puede interesar dormir un hilo? Pueden ser diferentes las razones que nos lleven a dormir un hilo durante unos instantes. En este apartado veremos un ejemplo en el que si no durmiéramos unos instantes al hilo que realiza un cálculo, no le daría tiempo al hilo que dibuja el resultado a presentarlo en pantalla.¿Cómo funciona el método sleep()?

El método sleep() de la clase thread recibe como argumento el tiempo que deseamos dormir el hilo que lo invoca. Cuando transcurre el tiempo especificado, el hilo vuelve a estar "Ejecutable" ("Preparado") para continuar ejecutándose.

Hay dos formas de llamar a este método.

La primera le pasa como argumento un entero (positivo) que representa milisegundos:

sleep(long milisegundos)

La segunda le agrega un segundo argumento entero (esta vez, entre 1 y 999999), que representa un tiempo extra en nanosegundos que se sumará al primer argumento:

sleep(long milisegundos, int nanosegundos)

Cualquier llamada a sleep() puede provocar una excepción, que el compilador de Java nos obliga a controlar ineludiblemente mediante un bloque try-catch.

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.

El funcionamiento del algoritmo es el siguiente:

El hilo elegido para ejecutarse, siempre es el hilo "Ejecutable" de prioridad más alta.

Si hay más de un hilo con la misma prioridad, el orden de ejecución se maneja mediante un algoritmo por turnos (round-rubin) basado en una cola circular FIFO (Primero en entrar, primero en salir).

Cuando el hilo que está "ejecutándose" pasa al estado de "No Ejecutable" o "Muerto", se selecciona otro hilo para su ejecución.

La ejecución de un hilo se interrumpe, si otro hilo con prioridad más alta se vuelve "Ejecutable". El hecho de que un hilo con una prioridad más alta interrumpa a otro se denomina "planificación apropiativa" ('preemptive sheudling').

Pero la responsabilidad de ejecución de los hilos es del Sistemas Operativos sobre el que corre la JVM, y Sistemas Operativos distintos manejan los hilos de manera diferente:

En un Sistema Operativo que implementa time-slicing (subdivisión de tiempo), el hilo que entra en ejecución, se mantiene en ella sólo un micro-intervalo de tiempo fijo o cuanto (quantum) de procesamiento, de manera que el hilo que está "ejecutándose" no solo es interrumpido si otro hilo con prioridad más alta se vuelve "Ejecutable", sino también cuando su "cuanto" de ejecución se acaba. Es el patrón seguido por Linux, y por todos los Windows a partir de Windows 95 y NT.

En un Sistema Operativo que no implementa time-slicing el hilo que entra en ejecución, es ejecutado hasta su muerte; salvo que regrese a "No ejecutable", u otro hilo de prioridad más alta alcance el estado de "Ejecutable" (en cuyo caso, el primero regresa a "preparado" para que se ejecute el segundo). Es el patrón seguido en el Sistema Operativo Solaris.