Interfaz Runnable

Java ejecutable es una interfaz que se utiliza para ejecutar código en un hilo concurrente. Es una interfaz que es implementada por cualquier clase si queremos que las instancias de esa clase sean ejecutadas por un hilo.

La interfaz ejecutable tiene un método **run ()** indefinido con void como tipo de retorno y no acepta argumentos. El resumen del método del método run () se proporciona a continuación:

Implementación de Runnable

Es la forma más sencilla de crear un hilo implementando Runnable. Uno puede crear un hilo en cualquier objeto implementando Runnable. Para implementar un Runnable, uno solo tiene que implementar el método de ejecución.

En este método, tenemos el código que queremos ejecutar en un hilo concurrente. En este método, podemos usar variables, instanciar clases y realizar una acción como lo hace el hilo principal. El hilo permanece hasta el regreso de este método. El método de ejecución establece un punto de entrada a un nuevo hilo.

Existen varias diferencias entre la clase Thread y la interfaz Runnable según su rendimiento, uso de memoria y composición.

- Al extender el hilo, hay una sobrecarga de métodos adicionales, es decir, consumen memoria excesiva o indirecta, tiempo de cálculo u otros recursos.
- Dado que en Java, solo podemos extender una clase, y por lo tanto, si extendimos la clase Thread, no podremos extender ninguna otra clase. Es por eso que debemos implementar la interfaz Runnable para crear un hilo.
- Runnable hace que el código sea más flexible ya que, si estamos extendiendo un hilo, entonces nuestro código solo estará en un hilo mientras que, en el caso de ejecutable, uno puede pasarlo en varios servicios ejecutores, o pasarlo al entorno de un solo hilo.
- El mantenimiento del código es fácil si implementamos la interfaz Runnable.

Comenzando un hilo

El método start () de la clase Thread se usa para iniciar un hilo recién creado. Realiza las siguientes tareas:

- Comienza un nuevo hilo (con una nueva pila de llamadas).
- El hilo se mueve del estado Nuevo al estado Ejecutable.
- Cuando el hilo tiene la oportunidad de ejecutarse, se ejecutará su método run () objetivo.

¿Podemos comenzar un hilo dos veces?

No. Después de comenzar un hilo, nunca más se puede volver a iniciar. Si lo hace, se *genera* una *IllegalThreadStateException* . En tal caso, el hilo se ejecutará una vez, pero por segunda vez, arrojará una excepción.

¿Qué sucede si llamamos al método run () directamente en lugar del todo start ()?

- Cada hilo comienza en una pila de llamadas separada.
- Invocando el método run () desde el hilo principal, el método run () pasa a la pila de llamadas actual en lugar de al comienzo de una nueva pila de llamadas.

Constantes definidas en la clase Thread:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

La prioridad predeterminada de un hilo es 5 (NORM_PRIORITY). El valor de MIN_PRIORITY es 1 y el valor de MAX_PRIORITY es 10.

Hilo de demonio en Java

El hilo del demonio en java es un hilo del proveedor de servicios que proporciona servicios al hilo del usuario. Su vida depende de la misericordia de los hilos de usuario, es decir, cuando todos los hilos de usuario mueren, JVM termina este hilo automáticamente.

Hay muchos subprocesos de Java Daemon que se ejecutan automáticamente, por ejemplo, gc, finalizer, etc.

Puede ver todos los detalles escribiendo jconsole en el símbolo del sistema. La herramienta jconsole proporciona información sobre las clases cargadas, el uso de memoria, la ejecución de subprocesos, etc.

Puntos a recordar para Daemon Thread en Java

- Proporciona servicios a hilos de usuario para tareas de soporte en segundo plano. No tiene ningún papel en la vida que servir hilos de usuario.
- Su vida depende de hilos de usuario.
- Es un hilo de baja prioridad.

¿Por qué JVM termina el subproceso de daemon si no hay subproceso de usuario?

El único propósito del hilo del demonio es que proporciona servicios al hilo del usuario para tareas de soporte en segundo plano. Si no hay un hilo de usuario, ¿por qué JVM debería seguir ejecutando este hilo? Es por eso que JVM termina el subproceso de daemon si no hay subproceso de usuario.

Métodos para Java Daemon thread por clase Thread

1)	public void setDaemon (estado booleano)	se usa para marcar el hilo actual como hilo de demonio o hilo de usuario.
2)	public boolean isDaemon ()	se usa para verificar que current sea daemon.

Grupo de subprocesos

El grupo de subprocesos de **Java** representa un grupo de subprocesos de trabajo que esperan el trabajo y se reutilizan muchas veces.

En el caso del grupo de subprocesos, se crea un grupo de subprocesos de tamaño fijo. El proveedor de servicios extrae un subproceso del grupo de subprocesos y le asigna un trabajo. Después de completar el trabajo, el hilo está contenido en el grupo de hilos nuevamente.

Ventaja de Java Thread Pool

Mejor rendimiento Ahorra tiempo porque no hay necesidad de crear un nuevo hilo.

ThreadGroup en Java

Java proporciona una forma conveniente de agrupar múltiples hilos en un solo objeto. De esta manera, podemos suspender, reanudar o interrumpir un grupo de subprocesos mediante una sola llamada al método.

> Nota: Ahora los métodos suspend (), resume () y stop () están en desuso.

El grupo de subprocesos de Java se implementa mediante la clase *java.lang.ThreadGroup* .

Un ThreadGroup representa un conjunto de hilos. Un grupo de hilos también puede incluir el otro grupo de hilos. El grupo de subprocesos crea un árbol en el que cada grupo de subprocesos, excepto el grupo de subprocesos inicial, tiene un elemento primario.

Un subproceso puede acceder a información sobre su propio grupo de subprocesos, pero no puede acceder a la información sobre el grupo de subprocesos principal del grupo de subprocesos o cualquier otro grupo de subprocesos.