APUNTES DE HILOS

Programación de Servicios y Procesos

3 DE NOVIEMBRE DE 2024

ERIK AMO TOQUERO

Contenido

Introducción a Hilos

Los procesos son una serie de comandos secuenciales que se ejecutan dentro del procesador, y las aplicaciones son las encargadas de crear esas secuencias de comandos.

Dentro de un proceso, puede haber subprocesos trabajando en paralelo. Esos subprocesos son los **hilos**.

Java tiene un muy buen manejo de los hilos, permitiendo crearlos y terminarlos desde las clases abstractas **Thread** y **Runnable**

Diferencia entre Thread y Runnable

Thread es una clase abstracta de la librería java.lang (La librería principal de Java) encargada de crear hilos. Redefiniendo el método run, se crea todo el código del nuevo subproceso

Runnable es una interfaz que se aplica a clases donde, por x motivos, no es posible aplicar la herencia de Thread (por ejemplo, es una clase que hereda ya de otra clase).

La forma en la que se inician los hilos con Runnable es ligeramente distinta, pero no cambia de forma excesiva.

Creación de un Hilo en Java

Con la clase abstracta Thread, se creará una nueva clase que herede de esta y sobrecargue el método abstracto **run**.
Una vez hecho eso, creará una instancia de la clase creada y se iniciará el hilo con el método **start**.

AVISO: NO EDITAR EL MÉTODO START Y NO COMENZAR CON RUN. No funcionará el hilo.

```
class Hilo extends Thread{
    @Override
    public void run() {
        [...]
    }
}


public class Main {
    public static void main(String[] args) {
        Hilo miHilo = new Hilo();
        miHilo.start();
    }
}
```

Si lo queremos hacer con la interfaz Runnable, se creará una clase que implemente esta y se sobrecargará el método **run**.

Acto seguido, se creará un objeto de tipo Thread que reciba por parámetro una instancia de la clase que hemos creado y se iniciará con **start**.

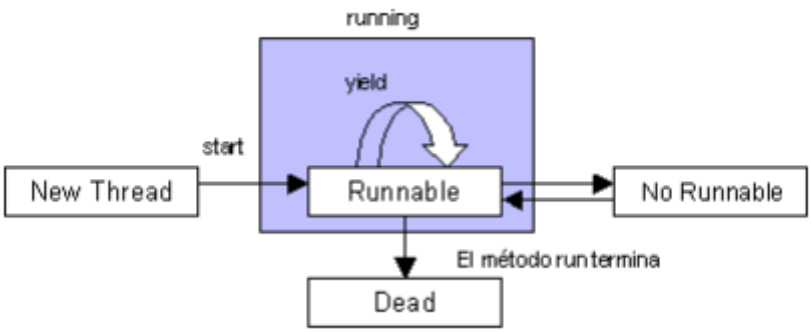
```
class Hilo implements Runnable{
    @Override
    public void run() {
        [...]
    }
}


public class Main {
    public static void main(String[] args) {
        Hilo miHilo = new Hilo();
        Thread thread = new Thread(miHilo);
        thread.start();
    }
}
```

Explicación del Ciclo de Vida de los hilos

Los subprocesos tienen un ciclo de vida predefinido que pasa por distintos estados:

- New Thread -> Hilo recién creado, aún no se ha iniciado ni nada.
- Runnable -> Hilo ejecutándose. Desde que empieza con **start** hasta que termina el método **run** o se interrumpe el hilo
- No Runnable -> Hilo interrumpido temporalmente por los métodos internos **sleep** y **wait** (lanzan una excepción de InterruptedException)
- Dead -> Ha terminado de recorrer el método **run**.



Prioridades

Aunque los hilos aparentemente se ejecuten de forma simultánea y paralela, el sistema ejecuta una instrucción cada vez. El mecanismo de Java para elegir el orden de concurrencia de las instrucciones se llama *fixed priority scheduling*. Esto se refiere a que los hilos siguen una prioridad relativa entre ellas.

La prioridad de los hilos en Java se marca por un número entero. Cuanto más alto sea el número de prioridad, mayor prioridad tiene su ejecución frente a otros hilos. La prioridad de un hilo recién creado es igual a la prioridad del hilo donde se crea. Por ejemplo: Si se crea un hilo dentro de otro que tiene prioridad 3, el nuevo hilo tendrá prioridad 3.

Cuando hay varios hilos en runnable, la VM de Java elige al hilo con mayor prioridad y no pasa al siguiente hasta que:

1. Haya otro hilo de mayor prioridad preparado para ejecutarse
2. El hilo termine o se detenga voluntariamente
3. El hilo se detenga por alguna condición, como son las operaciones de entrada y salida

Si hay varios hilos con la misma prioridad, se ceden el control de forma cíclica

Con la existencia de hilos egoístas (hilos que no se detienen voluntariamente para ceder la ejecución de otros hilos), algunos sistemas operativos han preferido crear la división de tiempos, que **asigna tiempos a cada hilo de misma prioridad y se ceden consecutivamente la ejecución entre ellos**.

Una vez explicado todo esto, decir que, en los hilos de Java, la prioridad se controla con la función **setPriority**

```
class Hilo implements Thread{
    @Override
    public void run() {
        [...]
    }
}


public class Main {
    public static void main(String[] args) {
        Hilo miHilo = new Hilo();
        int prioridad = 1;
        miHilo.setPriority(prioridad);
        miHilo.start();
    }
}
```

Sincronización de Hilos

Los hilos en Java se ejecutan de forma paralela y asíncrona entre ellos. Una vez son iniciados, son independientes entre ellos. Algunas veces interesa que los hilos de un programa tengan alguna relación entre ellos o compartan objetos. Para ello, existe la sincronía, un mecanismo que permite establecer unas reglas para acceder a recursos compartidos.

Esto se suele ver en el caso de que un hilo produzca información que necesite otro hilo.

Para verlo mejor, aquí viene un ejemplo (Sacado de GeeksForGeeks):

```
class Mensajero{
    public void enviar(String mensaje){
        System.out.println("Enviando el mensaje...");
        try{
            Thread.sleep(1000);
        }catch (Exception e){
            System.out.println("Error de Thread");
        }
        System.out.println("Mensaje " + mensaje + " enviado");
    }
}
class EnviarMensaje extends Thread {
    String mensaje;
    Mensajero mensajero;
    public EnviarMensaje(String mensaje, Mensajero mensajero) {
        this.mensaje = mensaje;
        this.mensajero = mensajero;
    }
    @Override
    public void run() {
        synchronized (mensajero){
            mensajero.enviar(mensaje);
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Mensajero mensajero = new Mensajero();
        EnviarMensaje mensaje = new EnviarMensaje("Hola", mensajero);
        EnviarMensaje mensaje2 = new EnviarMensaje("Adios",
mensajero);
        mensaje.start();
        mensaje2.start();
    }
}
```

La explicación es la siguiente:

Los dos hilos EnviarMensaje utilizan el mismo objeto sincronizado, es decir, el objeto Mensajero es un recurso compartido entre hilos.

La sincronía que se ve en **syncrhronized** lo que hace es que el objeto no se sobreescriba en un hilo hasta que haya acabado de ser operado por el hilo que le está utilizando.

Para verlo un poco mejor:

1. El primer hilo se inicia y empieza a operar con el mensajero
2. El segundo hilo se inicia y espera a que el primer hilo termine de operar el synchronized del objeto.
3. Una vez el primer hilo acaba y el objeto queda libre, el segundo utiliza el objeto para operar.