23. Threads I

23.1. Qué es un Thread

La Máquina Virtual Java (JVM) es un sistema multi-thread. Es decir, es capaz de ejecutar varias secuencias de ejecución (programas) simultáneamente. La JVM gestiona todos los detalles, asignación de tiempos de ejecución, prioridades, etc, de forma similar a como gestiona un Sistema Operativo múltiples procesos. La diferencia básica entre un proceso de Sistema Operativo y un Thread Java es que los Threads corren dentro de la JVM, que es un proceso del Sistema Operativo y por tanto comparten todos los recursos, incluida la memoria y las variables y objetos allí definidos. A este tipo de procesos donde se comparte los recursos se les llama a veces 'procesos ligeros' (lightweight process).

Java da soporte al concepto de Thread desde el mismo lenguaje, con algunas clases e interfaces definidas en el package java.lang y con métodos específicos para la manipulación de Threads en la clase Object.

Desde el punto de vista de las aplicaciones los threads son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea. Por ejemplo un Thread puede encargarse de la comunicación con el usuario, mientras otros actuan en segundo plano, realizando la transmisión de un fichero, accediendo a recursos del sistema (cargar sonidos, leer ficheros ...), etc. De hecho todos los programas con interface gráfico (AWT o Swing) son multithread porque los eventos y las rutinas de dibujado de las ventanas corren en un thread distinto al principal.

23.2. La Clase Thread

La forma más directa para hacer un programa multi-thread es extender la clase Thread, y redefinir el método run(). Este método es invocado cuando se inicia el thread (mediante una llamada al método start() de la clase thread). El thread se inicia con la llamada al método run y termina cuando termina éste. El ejemplo ilustra estas ideas:

```
public class ThreadEjemplo extends Thread {
    public ThreadEjemplo(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10 ; i++)
            System.out.println(i + " " +
getName());
        System.out.println("Termina thread " +
getName());
```

```
        }
        public static void main (String [] args) {
            new ThreadEjemplo("Pepe").start();
            new ThreadEjemplo("Juan").start();
            System.out.println("Termina thread main");
        }
    }
```

Compila y ejecuta el programa. La salida, será algo asi:

```
Termina thread main
0 Pepe
1 Pepe
2 Pepe
3 Pepe
0 Juan
4 Pepe
1 Juan
5 Pepe
2 Juan
6 Pepe
3 Juan
7 Pepe
4 Juan
8 Pepe
5 Juan
9 Pepe
6 Juan
Termina thread Pepe
7 Juan
8 Juan
9 Juan
Termina thread Juan
```

Ejecuta varias veces el programa. Verás que no siempre se ejecuta igual.

Notas sobre el programa:

- La clase Thread está en el package java.lang. Por tanto no es necesario el import.
- El constructor **public** `Thread(String str)` recibie un parámetro que es la identificación del Thread.
- El método run contiene el bloque de ejecución del Thread. Dentro de él, el método getName() devuelve el nombre del Thread (el que se ha pasado como argumento al constructor).
- El método main crea dos objetos de clase ThreadEjemplo y los inicia con la llamada al método start(). (el cual inicia el nuevo thread y llama al método run()).
- Observa en la salida el primer mensaje, de finalización del thread main. La ejecución de los threads es asíncrona. Realiza la llamada al método start(), éste le devuelve control y continua su ejecución, independiente de los otros threads.
- En la salida los mensajes de un thread y otro se van mezclando. La máquina virtual asigna tiempos a cada thread.

23.3. La Interface Runnable

La interface Runnable proporciona un método alternativo a la utilización de la clase Thread, para los

casos en los que no es posible hacer que nuestra clase extienda la clase Thread. Esto ocurre cuando nuestra clase, que deseamos correr en un thread independiente deba extender alguna otra clase. Dado que no existe herencia múltiple, nuestra clase no puede extender a la vez la clase Thread y otra más. En este caso nuestra clase debe implantar la interface Runnable, variando ligeramente la forma en que se crean e inician los nuevos threads.

El siguiente ejemplo es equivalente al del apartado anterior, pero utilizando la interface Runnable:

```
public class ThreadEjemplo implements Runnable {
    public void run() {
        for (int i = 0; i < 5 ; i++)
            System.out.println(i + " " +
Thread.currentThread().getName());
        System.out.println("Termina thread " +
Thread.currentThread().getName());
    }
    public static void main (String [] args) {
        new Thread ( new ThreadEjemplo() , "Pepe").start();
        new Thread ( new ThreadEjemplo() , "Juan").start();
        System.out.println("Termina thread main");
    }
}
```

Observese en este caso:

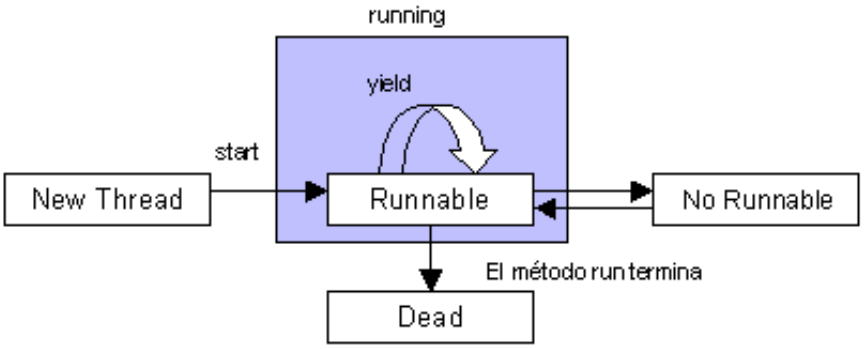
- Se implanta la interface Runnable en lugar de extender la clase Thread.
- El constructor que había antes no es necesario.
- En el main observa la forma en que se crea el thread. Esa expresión es equivalente a:

  ```
  ThreadEjemplo ejemplo = new ThreadEjemplo();
  Thread thread = new Thread ( ejemplo , "Pepe") ;
  thread.start();
  ```

  - Primero se crea la instancia de nuestra clase.
  - Después se crea una instancia de la clase Thread, pasando como parámetros la referencia de nuestro objeto y el nombre del nuevo thread.
  - Por último se llama al método start de la clase thread. Este método iniciará el nuevo thread y llamará al método run() de nuestra clase.
- Por útlimo, obsérvese la llamada al método getName() desde run(). getName es un método de la clase Thread, por lo que nuestra clase debe obtener una referencia al thread propio. Es lo que hace el método estático currentThread() de la clase Thread.

23.4. El ciclo de vida de un Thread

El gráfico resume el ciclo de vida de un thread:



Cuando se instancia la clase Thread (o una subclase) se crea un nuevo Thread que está en en su estado inicial ('New Thread' en el gráfico). En este estado es simplemente un objeto más. No existe todavía el thread en ejecución. El único método que puede invocarse sobre él es el método start.

Cuando se invoca el método start sobre el thread el sistem crea los recursos necesarios, lo planifica (le asigna prioridad) y llama al método run. En este momento el thread está corriendo.

Si el método run invoca internamente el método sleep o wait o el thread tiene que esperar por una operación de entrada/salida, entonces el thread pasa al estado 'no runnable' (no ejecutable) hasta que la condición de espera finalice. Durante este tiempo el sistema puede ceder control a otros threads activos.

Por último cuando el método run finaliza el thread termina y pasa a la situación 'Dead' (Muerto).

Ultima actualización - 18-Febrero-2001

Antonio Bel Puchol - abelp@arrakis.es