élogos

MULTITAREA
LA CLASE THREAD
Y LA INTERFAZ
RUNNABLE

élogos

ÍNDICE

MULTITAREA. LA CLASE THREAD Y LA INTERFAZ RUNNABLE

1. Estructura básica de un programa 3
2. Herencia de la clase Thread 3
3. Interface Runnable. Prioridad de Hilos 5
4. Ciclo de vida de un hilo 7

1. Estructura básica de un programa

Concepto de Tarea y Multitarea

1. Programación Multihilo en Java. La clase Thread

Scheduler es un subsistema de Sistema Operativo que se encarga de ir asignando ciclos de CPU a los distintos programas o tareas que van a ser ejecutados. A estos programas se les conoce también como **hilos.**

El sistema que utiliza Sheduler se denomina Roun Rubin de tal forma que cuando se llega al tiempo máximo ejecuta una interrupción del sistema y saca el programa fuera del procesador para que se ejecute otro programa consiguiendo un buen rendimiento del sistema.

La clase Thread

Todo hilo en ejecución es gestionado por un objeto de la clase Thread, clase que se encuentra en el paquete java.lang.

Entre los métodos más importantes que proporciona esta clase para controlar los hilos en ejecución tenemos:

hilo main , hilo que comienza y termina en la función main().
void<Thread>.setName(String Name) → Cambia el nombre del hilo.
static Thread<Thread>. CurrentThread() → Obtiene el hilo actual.
void<Thread> sleep (Long miliseconds) throws InterruptedException → Duerme el hilo.

2. Herencia de la clase Thread

Si queremos crear aplicaciones con la posibilidad de tener varios hilos en ejecución concurrente (multitarea), tendremos que implementar el código de dichos hilos en clases personalizadas que hereden Thread.

La clase Thread proporciona el método run (), que deberá ser sobrescrito por las subclases para definir en él el código asociado a los hilos:

```
Class MiTarea extends Thread{
    @Override
    public void run(){
        //código del hilo
    }
}
```

Cada hilo se creará como un objeto de la clase MiTarea y será puesto en ejecución llamando al método **start()** heredado de Thread. La llamada a start() provocará que el scheduler tome el control del hilo.

Ejemplo con más de un hilo

Cada hilo o tarea está representada por un objeto de la clase Coche, que al derivar de Thread se considerará un hilo.

El método run que representa la CPU, será un bucle for de 1 a 10 que simulará que los coches dan 10 vueltas a un circuito y en cada vuelta extraemos el nombre de la tarea y lo imprimimos, así sabremos en cada vuelta qué coche va primero.

```
class Coche extends Thread {
static int posicion = 0;
Coche (String nombre) {
super(nombre);  // Llama al constructor de Thread
}
public void run() {
for (int i=0; i < 10; i++) {
System.out.println ("Kilometro "+i+" coche "+getName());
try {
sleep ((int) (Math.random() * 1000));
}
catch (InterruptedException ex) {}
System.out.println ("Acabe: "+
getName()+" en la posición "+(++posicion));
}
} // fin de clase
class HiloCoche {
public static void main (String [] args) {
// Se crean los objetos.
Coche c1  = new Coche("Porche");
Coche c2  = new Coche("Jaguar");
Coche c3  = new Coche ("Seat");
// Se lanzan los hilos.
c1.start();
c2.start();
c3.start();
} // Fin del main
} //  Fin de la clase.
```

3. Interface Runnable. Prioridad de Hilos

Interface Runnable

Se utiliza siempre que se desea que un objeto se comporte como un Thread pero tenemos el inconveniente de que su clase tiene, por fuerza, que derivar de otra clase diferente -y como ya sabemos que en Java no existe la herencia múltiple- no podremos derivar a la vez de Thread.

Para ello implementamos (implements) la interfaz Runnable:

Definición

```
public interface Runnable {
public void run()
}
```

Esta interfaz contiene el método run() vacío, y cualquier clase que implemente dicha interfaz habrá de redefinir dicho método.

```
public class NuevaClase extends ClasePadre
implements Runnable{
        public void run (){
        //código multitarea
        }
}
```

Ejecución de hilos

Las clases que implementan la interfaz Runnable, al no heredar Thread, no disponen del método start() para poner en ejecución concurrente los hilos.

En este caso, para crear los hilos, tendremos que hacer uso de alguno de los siguientes constructores de Thread:

- Thread(Runnable target). Crea un hilo a partir del objeto que implementa la interfaz Runnable
- Thread(Runnable target, String name). Igual que el anterior, pero asignando un nombre al hilo

Para poner dos hilos en ejecución a partir de la clase Runnable NuevaClase:

```
NuevaClase target = new NuevaClase();
new Thread(target, “hilo1”).start();
New Thread(target, “hilo2”).start();
```

Como podemos apreciar, el mismo objeto Runnable puede ser utilizado para la creación de todos los hilos, y es que cuando se implementa la multitarea a través de la interfaz Runnable, **los hilos son objetos Thread naturales**, no los objetos de la clase de implementación.

Ejemplo multitarea con Runnable

Veamos un sencillo ejemplo de aplicación multitarea basada en el uso de la interfaz Runnable.
Se trata de crear dos hilos que se ejecuten de forma concurrente, de modo que cada hilo muestre su nombre por pantalla 100 veces.

```
public class Impresion implements Runnable{
    public void run(){
        for(int i=1;i<=100;i++){
            //obtiene el objeto Thread en ejecución
            //y muestra su nombre

System.out.println(Thread.currentThread().getName());
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {}
        }
    }
}
public class Inicio {
    public static void main(String[] args) {
        Impresion target=new Impresion();
        new Thread(target, "hilo1").start();
        new Thread(target, "hilo2").start();
    }
}
```

Resultado

```
hilo1
hilo2
hilo2
hilo1
hilo1
hilo2
hilo2
hilo1
hilo1
hilo2
hilo1
Hilo2
:
```

Prioridad de los hilos

Los hilos tienen prioridad. La prioridad se mide de 0 a 10, pero dependiendo del sistema operativo, la prioridad se mide de diferente forma. Esto quiere decir que JVM traduce la prioridad de JAVA a la prioridad del sistema operativo.

Por defecto, todos los hilos tienen como valor 5 a la prioridad.

Método para cambiar la prioridad:

Void <Thread>.SetPriority(int Priority)

Thread tiene constantes que indican el valor de la prioridad:

- Thread.MIN_PRIORITY
- Thread.MAX_PRIORITY
- Thread.NORMAL_PRIORITY

4. Ciclo de vida de un hilo

Estados de un hilo

Running (Ejecutando)
Es un estado en el que sólo puede estar un hilo en toda la máquina excepto si la máquina tiene más de un procesador (1 microprocesador = 1 hilo).

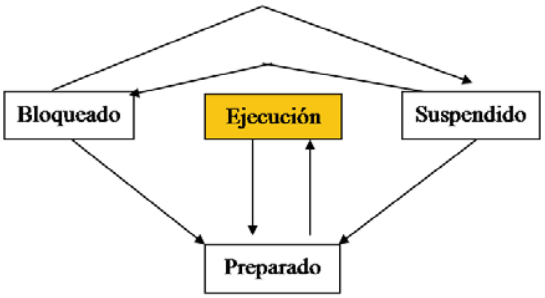
Ready (Preparado)
Es el estado en el que están todos los hilos a la espera de ejecución ya que el microprocesador está ejecutando otro hilo.

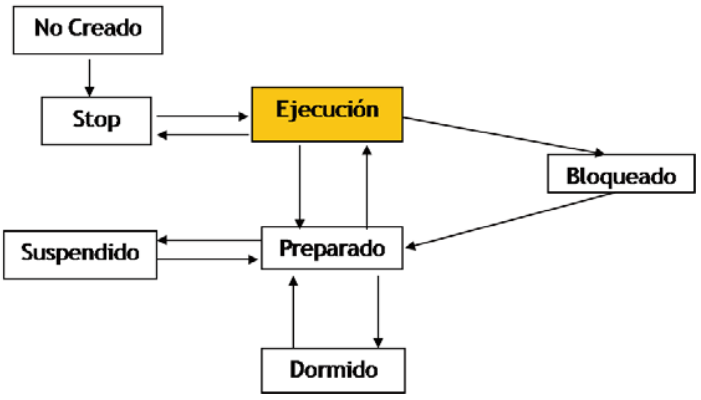
Blocked (Bloqueado)
Es el estado en el que está un hilo cuando está realizando una operación de entrada/salida.

Suspended (Suspendido)

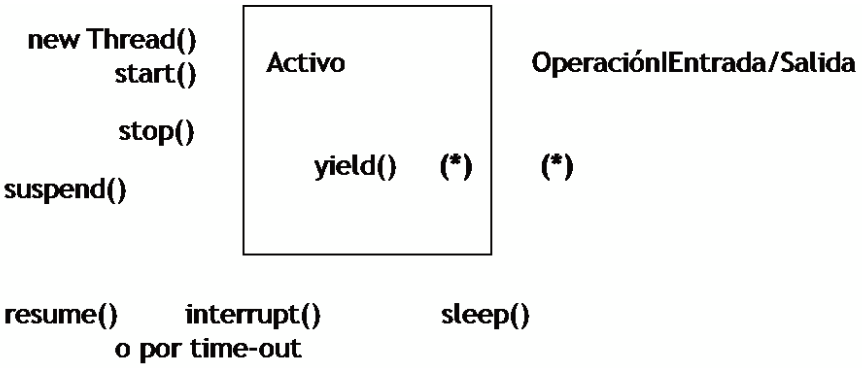
Es el estado en el que está un hilo cuando otro hilo lo ha dormido o él mismo se ha dormido.

Diagrama de estados en el sistema operativo:





Estados programáticos



- (*) Estos estados son controlados por el sistema operativo. No se puede tratar desde Java. Sólo se puede ceder el paso en la ejecución con el método yield().
- (1) Tras haber realizado un stop() y posteriormente se realiza un start() se comienza a ejecutar el hilo desde el principio.
- (2) Tras hacer suspend() si se realiza un resume() se comienza la ejecución del hilo desde donde se dejó.