2.6. GESTION DE PRIORIDADES

En el lenguaje de programación Java, cada hilo tiene una prioridad. Por defecto, un hilo hereda la prioridad del hilo padre que le crea, esta se puede aumentar o disminuir mediante el método setPriority(). El método getPriority() devuelve la prioridad del hilo.

La prioridad no es más que un valor entero entre 1 y 10, siendo el valor 1 la mínima prioridad, MIN PRIORITY; y el valor 10 la máxima, MAX_PRIORITY. NORM_PRIORITY se define como 5. El planificador elige el hilo que debe ejecutarse en función de la prioridad asignada; se ejecutara primero el hilo de mayor prioridad. Si dos o más hilos están listos para ejecutarse y tienen la misma prioridad, la máquina virtual va cediendo control de forma cíclica.

El planificador es la parte de la máquina virtual de Java que decide que hilo ejecutar en cada momento. Da más ventaja a hilos con mayor prioridad; hilos de igual prioridad en algún momento se ejecutarán.

El hilo de mayor prioridad sigue funcionando hasta que:

- Cede el control llamando al método yield() para que otros hilos de igual prioridad se ejecuten.
- Deja de ser ejecutable (ya sea por muerte o por entrar en el estado de bloqueo).
- Un hilo de mayor prioridad se convierte en ejecutable (porque se encontraba dormido o su operación de E/S ha finalizado o alguien lo desbloquea llamando a los métodos notifyAll() notify()).

El uso del método yield() devuelve automáticamente el control al planificador, el hilo pasa a un estado de listo para ejecutar. Sin este método el mecanismo de multihilos sigue funcionando aunque algo más lentamente.

En el siguiente ejemplo se crea una clase que extiende Thread, se define una variable contador que será incrementada en el método run(), se define un método para obtener el valor de la variable y otro método para finalizar el hilo, en el constructor se le da nombre al hilo:

```
class HiloPrioridad1 extends Thread {
private int c = 0;
private boolean stopHilo = false;
public HiloPrioridadI(String nombre) {
super(nombre);
}
public int getContador() {
return c;
}
public void pararHilo() {
stopHilo = true;
}
public void run() {
´ while (!stopHilo) {
try {
Thread.sleep(2);
} catch (Exception e) { }
c++;
}
System.out.println("Fin hilo "1-this.getName());
```

```
}//
}//HiloPrioridad1
```

A continuación se crea la clase EjemploHiloPrioridad1 con el método main() en el que se definen 3 objetos de la clase HiloPrioridad1, a cada uno se le asigna una prioridad. El contador del hilo al que se le ha asignado mayor prioridad contará más deprisa que el de menos prioridad. Al finalizar cada hilo se muestran los valores del contador invocando a cada método getComador() del hilo:

```
public class EjemploHiloPrioridad1      {
public  static  void main(String  args[]) {
HiloPrioridadl  h1   =  new  HiloPrioridadl("Hilo1");
HiloPrioridadl h2    = new HiloPrioridadl("Hilo2");
HiloPrioridadl  h3   =  new  HiloPrioridadl("Hilo3");

H1.setPriority(Thread.NORM_PRIORITY);
h2.setPriority(Thread.MAX_PRIORITY);
h3.setPriority(Thread.MIN PRIORITY);

h1.start();
h2.start();
h3.start();

try {
Thread.sleep(10000);
}catch (Exception e)            {}

h1.pararHilo()      ;
h2.pararHilo()      ;
h3.pararHilo()      ;
System.out.println("h2 (Prioridad Maxima):  " + h2.getContador());
System.out.println("h1 (Prioridad Normal):  " + h1.getContador());
System.out.println("h3 (Prioridad Minima):  "+ h3.getContador());
}
}// EjemploHiloPrioridad1
```

Varias ejecuciones del programa muestran las siguientes salidas, se puede observar que el máximo valor del contador lo obtiene el objeto hilo con prioridad máxima, y el mínimo el de prioridad mínima:

```
h2      (Prioridad Maxima):     4856
Fin     hilo    Hilo3
Fin     hilo    Hilo1
Fin     hilo    Hilo2
h1      (Prioridad Normal):4855
h3      (Prioridad Minima):     4854

h2      (Prioridad Maxima):     4767
h1      (Prioridad Normal):4684
h3      (Prioridad Minima):     4668
Fin     hilo            Hilo2
Fin     hilo            Hilo1
Fin     hilo    Hilo3

h2      (Prioridad Maxima):     4565
```

```
h1      (Prioridad Normal):4545
h3      (Prioridad Minima):     4523
Fin     hilo            Hilo2
Fin     hilo            Hilo1
Fin     hilo            Hilo3
```

Pero no siempre ocurre esto. Podemos encontrar la siguiente salida en la que se observa que los valores de los contadores no dependen de la prioridad asignada al hilo:

```
h2      (Prioridad Maxima):     4822
Fin  hilo   Hilo3
Fin  hilo   Hilo2
Fin  hilo   Hilo1
H1      (Prioridad Normal):     4823
h3      (Prioridad Minima):     4823

h2      (Prioridad Maxima):     4518
Fin     hilo    Hilo2
h1      (Prioridad Normal):     4518
h3      (Prioridad Minima):     4517
Fin  hilo   Hilo1
Fin  hilo   Hilo3
```

En el siguiente ejemplo se asignan diferentes prioridades a cada uno de los kilos de la clase EjemploHiloPrioridad2 que se crean. En la ejecución se puede observar que no siempre el hilo con más prioridad es el que antes se ejecuta:

```
public class EjemploHiloPrioridad2  extends  Thread {
EjemploHiloPrioridad2(String nom){
        this.setName(nom);
}

public void run(){
        System.out.println("Ejecutando  E"      +  getName()    +  "]");
        for      (int  i    =1;   i    <=5;  i++)
                 System.out.println("\t("+getName()+":   "        +  i+")");
}
public  static void main(Stringrj  args)    {
        EjemploHiloPrioridad2  h1 =  new EjemploHiloPrioridad2("Uno");
        EjemploHiloPrioridad2  h2 =  new  EjemploHiloPrioridad2("Dos");
        EjemploHiloPrioridad2  h3 =  new EjemploHiloPrioridad2("Tres");
        EjemploHiloPrioridad2  h4 =  new EjemploHiloPrioridad2("Cuatro");
        EjemploHiloPrioridad2  h5 =  new EjemploHiloPrioridad2("Cinco");

        //asignamos  prioridad
        h1.setPriority(Thread.MIN_PRIORITY);
        h2. setPriority(3);
        h3.setPriority(Thread.NORM_PRIORITY); +
        h4.setPriority(7);
        h5.setPriority(Thread.MAX_PRIORITY);

        //se ejecutan  los  hilos
```

```
        h1.start();
        h2.start();
        h3.start();
        h4.start();
        h5.start();
        }
}
```

Un ejemplo de ejecución muestra la siguiente salida:

```
Ejecutando      [Cuatro]
        (Cuatro:        1)
        (Cuatro:        2)
        (Cuatro:        3)
        (Cuatro:        4)
        (Cuatro:        5)
Ejecutando      [Tres]
        (Tres:  1)
        (Tres:  2)
        (Tres:  3)
        (Tres:  4)
        (Tres:  5)
Ejecutando      [Cinco]
        (Cinco: 1)
        (Cinco: 2)
        (Cinco: 3)
        (Cinco: 4)
        (Cinco: 5)
Ejecutando      [Dos]
        (Dos:   1)
        (Dos:   2)
        (Dos:   3)
        (Dos:   4)
        (Dos:   5)
Ejecutando      [Uno]
        (Uno:   1)
        (Uno:   2)
        (Uno:   3)
        (Uno:   4)
        (Uno:   5)
```

A la hora de programar hilos con prioridades hemos de tener en cuenta que el comportamiento no está garantizado y dependerá de la plataforma en la que se ejecuten los programas y de las aplicaciones que se ejecuten al mismo tiempo. En la práctica casi nunca hay que establecer a mano las prioridades.

Cuando un hilo entra en ejecución y no cede voluntariamente el control para que puedan ejecutarse otros hilos, se dice que es un "hilo egoísta". Algunos sistemas operativos, como Windows, combaten estas situaciones con una estrategia de planificación por división de tiempos, que opera con hilos de igual prioridad que compiten por la CPU. En estas condiciones

el sistema operativo divide el tiempo de proceso de la CPU en espacios de tiempo y asigna el tiempo de proceso a los hilos dependiendo de su prioridad. Así se impide que uno de ellos se apropie del sistema durante un intervalo de tiempo prolongado.

ACTIVIDAD 2.6

Prueba los ejemplos anteriores variando la prioridad y el orden de ejecución de cada hilo. Comprueba los resultados para el primer ejemplo y para el segundo.