Ejecución de hebras

En realidad, todas las aplicaciones escritas en Java son aplicaciones multihebra (recuerde el recolector de basura).

Hebras vs. Procesos

- Los cambios de contexto son más costosos en el caso de los procesos al tener que cambiar el espacio de direcciones en el que se trabaja.
- La comunicación entre procesos que se ejecutan en distintos espacios de memoria es más costosa que cuando se realiza a través de la memoria que comparten las distintas hebras de un proceso.
- Sin embargo, un fallo en una hebra puede ocasionar la caída completa de la aplicación mientras que, si se crean procesos independientes, los fallos se pueden contener en el interior de un proceso, con lo que la tolerancia a fallos de la aplicación será mayor.

Creación y ejecución de hebras en Java

La forma más sencilla de crear una hebra en Java es diseñar una subclase de `java.lang.Thread`.

En la subclase redefiniremos el método `run()`, que viene a ser algo así como el `main()` de una hebra.

Para comenzar la ejecución paralela de la nueva hebra, usaremos su método `start()`.

```
public class HebraContador extends Thread
{
  private int contador = 10;

  private static int hebras = 0;

  // Constructor

  public HebraContador()
  {
    super("Hebra " + ++hebras);
  }

  // Salida estándar

  public String toString()
  {
    return getName() + ": " + contador;
  }

  // Ejecución de la hebra

  public void run()
  {
    while (contador>0) {
          System.out.println(this);
          contador--;
    }
  }

  // Programa principal

  public static void main(String[] args)
  {
    int    i;
    Thread hebra;

    for (i = 0; i < 5; i++)
        hebra = new HebraContador();
        hebra.start();
    }
  }
}
```

El orden en que se realizan las operaciones de las distintas hebras puede cambiar de una ejecución a otra
(en función de cómo se asigne la CPU
a la distintas hebras de nuestro programa).

NOTA: En el ejemplo anterior, lo más probable es que las cinco hebras se ejecuten secuencialmente salvo que incrementemos el valor inicial del contador.

Si queremos ceder explícitamente la CPU para que se le asigne a otra hebra distinta a la actual, podemos utilizar `yield()`:

```
public void run()
{
  while (contador>0) {
        System.out.println(this);
        contador--;
        yield();
  }
}
```

El método `yield()` no se suele utilizar en la práctica, aunque sí suele ser útil detener la ejecución de una hebra durante un período de tiempo determinado (expresado en milisegundos) con una llamada al método `sleep()`:

```
public void run()
{
  while (contador>0) {
        System.out.println(this);
        contador--;
        try {
            sleep(1000);
        } catch (InterruptedException e) {
        }
  }
}
```

Como Java sólo soporta herencia simple y puede que el objeto que representa nuestra hebra sea mejor implementarlo como subclase de otra clase de nuestro sistema, Java nos permite crear una hebra a partir de cualquier objeto que implemente la interfaz `Runnable`, que define únicamente un método `run()`:

```
public class Contador implements Runnable
{
  private String id;
  private int    contador = 10;

  private static int hebras = 0;

  public Contador()
  {
    hebras++;
    id = "Hebra " + hebras;
  }

  public void run()
  {
    while (contador>0) {
          System.out.println(this);
          contador--;

          try {
              Thread.sleep(1000);
          } catch (InterruptedException e) {
          }
    }
  }

  public static void main(String[] args)
  {
    int    i;
    Thread hebra;

    for (i = 0; i < 5; i++) {
        hebra = new Thread ( new Contador() );
        hebra.start();
    }
  }
}
```

La prioridad de las hebras

La prioridad de una hebra le indica al planificador de CPU la importancia de una hebra, de tal modo que le asignará más tiempo de CPU a las hebras de mayor prioridad.

La prioridad de una hebra se cambia con una llamada al método `setPriority()`, usualmente en el constructor de la hebra:

```
public class HebraContador extends Thread
…
  public HebraContador (int prioridad)
  {
    super("Hebra " + ++hebras);
    setPriority (prioridad);
  }
```

Por defecto, la prioridad de las hebras es
`Thread.NORM_PRIORITY`

En el siguiente fragmento de código, la última hebra que lanzamos se ejecutará “antes” que las demás:

```
  int    i;
  Thread hebra;

  for (i = 0; i < 5; i++) {
      hebra = new HebraContador (Thread.MIN_PRIORITY);
      hebra.start();
  }

  hebra = new HebraContador (Thread.MAX_PRIORITY);
  hebra.start();
```

Cuando en nuestra aplicación tenemos varias hebras ejecutándose, les daremos mayor prioridad aquellas hebras cuyo tiempo de respuesta deba ser menor.

Finalización de la ejecución de una hebra

Se puede utilizar el método `join()` de la clase `Thread` para esperar la finalización de la ejecución de una hebra:

```
private static final int N = 5;

public static void main(String[] args)
{
  int    i;
  Thread hebra[] = new Thread[N];

  for (i=0; i<N; i++) {
      hebra[i] = new HebraContador(Thread.NORM_PRIORITY);
      hebra[i].start();
  }

  for (i=0; i<5; i++) {

      try {
          hebra[i].join();
      } catch (InterruptedException e) {
      }

      System.out.println (hebra[i]+" ha terminado");
    }

    System.out.println
          (“El programa principal ha terminado");
  }
```