

Programación Multihilo

En muchas ocasiones las sentencias que forman una **tarea** (programa) deben ejecutarse de manera secuencial. Al fin y al cabo, es necesario seguir una serie de pasos **en orden** para completar esa tarea. Pero puede ocurrir que esos pasos no tengan por qué ser secuenciales. ¿Y si hubiese partes que pudiesen ser ejecutadas **simultáneamente**? Si mi programa tiene que escribir un fichero, mientras tanto puede estar haciendo otras cosas mientras termina.

En otras ocasiones, que un programa haga varias cosas a la vez no es una opción, sino **una necesidad**. Sucede mucho en el entorno web sin que nos demos cuenta. Si cuando nos conectamos por ejemplo a una página web el servidor tuviese que esperar a terminar de atender a un cliente antes de atender a otro, el tiempo de espera sería tan elevado que cosas como los foros, redes sociales o YouTube no existirían.

En ambos casos la solución es la **Programación Multihilo**. Es decir, elaborar programas que sean capaces de ejecutar **varias partes de su código a la vez**. Nótese que el concepto es diferente al de la Programación Multiproceso.

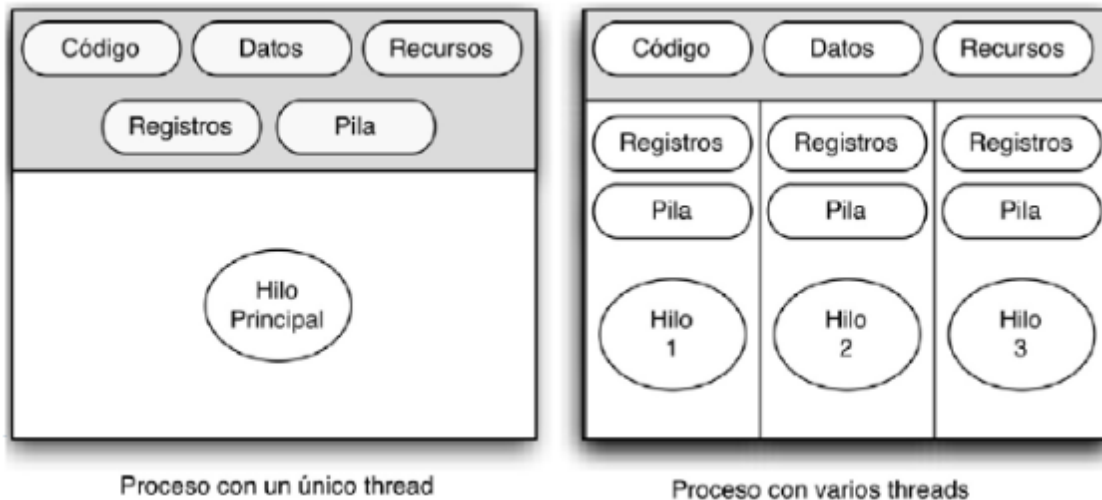
Codificar un programa usando **Hilos** no tiene mucha complicación en Java. El verdadero problema es conocer todas las condiciones, restricciones y problemas que tiene trabajar con **Hilos**. Sin entrar en detalles, nos estamos refiriendo a la **Sincronización** de los Hilos.

Introducción a los Hilos

Como ya sabemos, desde el punto de vista del Sistema Operativo todo son **Procesos**. Un Proceso cambia de estado, compite por el procesador, pide recursos, etc. Para él, un Proceso no deja de ser una secuencia de instrucciones una detrás de la otra. Nosotros no podemos intervenir (directamente) en la gestión de Procesos, dado que el Sistema Operativo se encarga de hacerlo todo él sólo.

Pero sí es posible aprovechar la **Programación Multihilo** para elaborar programas que tengan algo que decir respecto a la concurrencia. Y esto es porque todos los Programas tienen **al menos un Hilo** de ejecución. Un **Hilo**, también llamado **Thread**, es una unidad pequeña de computación dentro del contexto de un Proceso.

En un Programa secuencial normal – todos los que no usan Multihilo – en realidad sí que existe un **Hilo** de ejecución. Es decir, una secuencia de instrucciones que se ejecutan una detrás de la otra (if o while incluidos). Pero si somos capaces de programar varios Hilos en el programa – Multihilo – algunas sentencias del programa **se ejecutarán a la vez**.



Características de los Hilos

Algunas características de los Hilos:

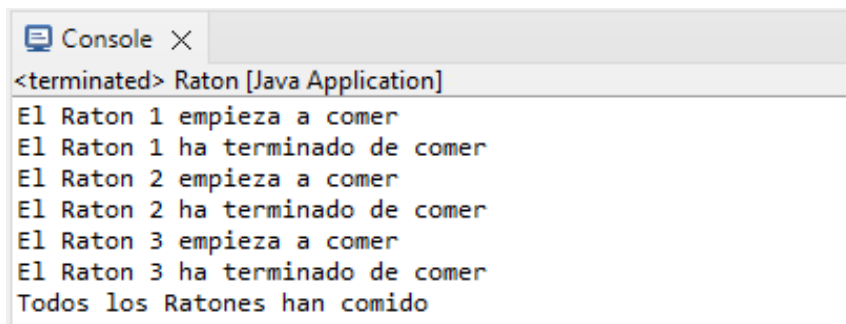
- **Dependen de un Proceso.** Nunca pueden existir fuera del contexto de un Proceso.
- **Ligeros.** Dado que se ejecutan dentro del contexto de un Proceso, uno o más **Hilos** no requiere asignación de nuevos recursos de memoria.
- **Colisiones.** Dentro de un mismo Proceso, varios Hilos pueden intentar acceder a las mismas variables, lo que puede provocar problemas de **Concurrencia** si varios Hilos tratan de leer o escribir esas variables.
- **Paralelismo.** Es posible aprovechar los núcleos del Procesador consiguiendo paralelismo real (un Hilo en cada núcleo).
- **Concurrencia.** Permiten atender de manera concurrente y simultánea varias peticiones diferentes.

Programación secuencial o de único Hilo

No hay mucho que explicar sobre este tipo de Programas, puesto que son los que habéis estado programando en la asignatura de Programación de 1º. Vamos a poner como ejemplo el Programa llamado Ratón, dado que nos permitirá comprender mejor la **Programación Multihilo** en el siguiente apartado.

Copia y ejecuta el Programa **Ratón** que viene con los apuntes. Verás que se limita a crear tres Ratones y a darles de comer. Date cuenta de que cuando empieza a comer el Ratón 1, hasta que no termine el siguiente Ratón no puede empezar a comer.

La salida será algo como esto:

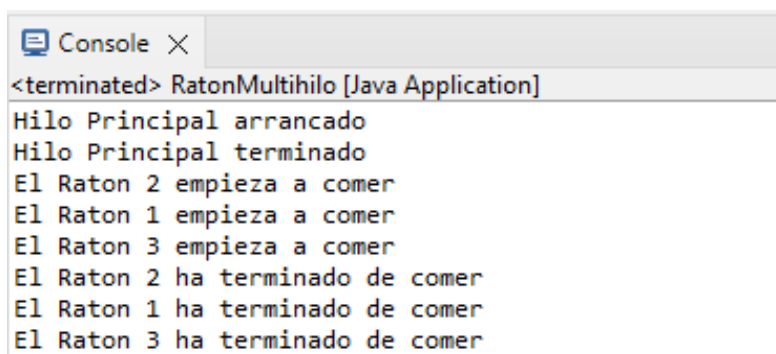


```
Console X
<terminated> Raton [Java Application]
El Raton 1 empieza a comer
El Raton 1 ha terminado de comer
El Raton 2 empieza a comer
El Raton 2 ha terminado de comer
El Raton 3 empieza a comer
El Raton 3 ha terminado de comer
Todos los Ratones han comido
```

Programación concurrente o Multihilo

Pero vamos a ver... tenemos tres Ratones y ningún recurso compartido entre ellos, por tanto, podemos hacer que coman los tres a la vez. ¿No? Pues vamos a ello. Copia y ejecuta el Programa **RatónMultihilo** que viene con los apuntes. Fíjate en los cambios que hay entre este programa y el programa Ratón.

La salida será algo como esto:



```
Console X
<terminated> RatonMultihilo [Java Application]
Hilo Principal arrancado
Hilo Principal terminado
El Raton 2 empieza a comer
El Raton 1 empieza a comer
El Raton 3 empieza a comer
El Raton 2 ha terminado de comer
El Raton 1 ha terminado de comer
El Raton 3 ha terminado de comer
```

¿Cómo es esto posible? Sin entrar en detalles, lo que ha ocurrido es que el **Hilo Principal** del programa ha creado **tres Hilos** y los ha arrancado uno detrás del otro. Dado que los **Hilos** por defecto se ejecutan a la vez, cada uno de ellos ha seguido su código hasta que ha terminado. Es por eso que aparecen por consola textos “desordenados”.

Hay **dos formas** de crear un Thread en Java:

- La primera consiste crear una clase nueva que **extienda** de la **clase Thread**.
- La segunda consiste en crear una clase que **implemente** la interfaz **Runnable**.

En ambos casos, será necesario que sobrescribamos/implementemos el método **run ()**, que contiene las instrucciones de lo que deseamos que haga el Hilo. Para arrancar el Hilo, únicamente tendremos que llamar al método **start ()**.

Un ejemplo sencillo de **Hilo** que extiende de **Thread**:

```
public class MiThread1 extends Thread {
    public void run() {
        System.out.println("Soy el hilo creado. Voy a contar.");
        for (int x=1; x<=10; x++)
            System.out.println(x);
    }
}

public class Principal {
    public static void main(String[] args) {
        MiThread1 t1 = new MiThread1();
        t1.start();
        System.out.println("Hola desde el principal");
    }
}
```

Un ejemplo sencillo de Hilo que implementa **Runnable**:

```
public class MiThread2 implements Runnable {
    public void run() {
        System.out.println("Soy el hilo creado. Voy a contar.");
        for (int x=1; x<=10; x++)
            System.out.println(x + " ");
    }
}

public class Principal {
    public static void main(String[] args) {
        MiThread2 mit2 = new MiThread2();
        Thread t2 = new Thread(mit2);
        t2.start();
        System.out.println("Hola desde el principal");
    }
}
```

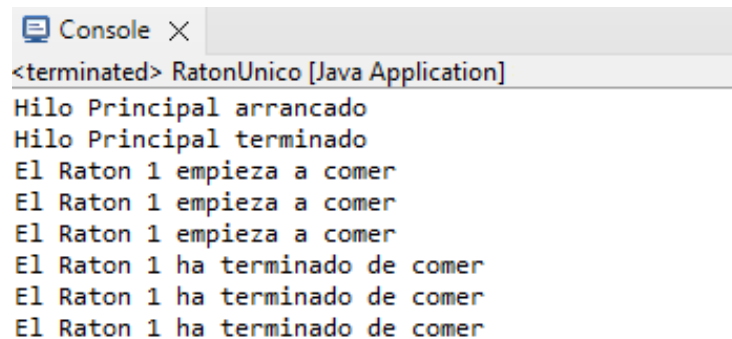
Ejercicio 1

Crea el programa **RatónRunnable** a partir de **RatónMultihilo**, pero implementando la interfaz **Runnable**.

¿Thread o Runnable?

La respuesta corta es “cualquiera de las dos”. No obstante, hay diferencias. Si usamos el **interfaz Runnable** tenemos una ventaja clara sobre la **herencia de Thread**. Es posible lanzar muchos hilos sobre un único objeto, a diferencia de la herencia de Thread que generará un objeto por cada Hilo. Puedes comprobarlo en el programa **RatónUnico**.

La salida será algo como esto:



```
Console X
<terminated> RatonUnico [Java Application]
Hilo Principal arrancado
Hilo Principal terminado
El Raton 1 empieza a comer
El Raton 1 empieza a comer
El Raton 1 empieza a comer
El Raton 1 ha terminado de comer
El Raton 1 ha terminado de comer
El Raton 1 ha terminado de comer
```

Pero trabajar así es un problema. Es importante insistir en que, por muchos Hilos que se hagan, en el ejemplo **RatónUnico** estamos trabajando sobre **un mismo Objeto** en memoria. Esto significa que todas las variables son compartidas, lo que podría producir **errores de concurrencia**. Intenta lanzar el programa **RatónUnicoBucle** y comprobarás que los mensajes son bastante inconsistentes.

La última línea escrita por consola es algo así:

```
El Raton 1 ha terminado de comer
Alimento consumido: 971
```

¿Cómo es posible? Debería de haber sido 1000, dado que estamos lanzando 1000 Hilos. No obstante, como están actualizando **todos a la vez** la variable alimentoConsumido, al final se pisan los unos a los otros y se generan errores de concurrencia.