# INSTITUTO POLITÉCNICO NACIONAL

## UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

**Material para el curso de Programación Orientada a Objetos**

# Contenido

4.4 Manejo de hilos

4.4.1 Concepto
4.4.2 Creación
4.4.3 Sincronización

# Threads

Es una línea de ejecución separada en una aplicación, generalmente independiente. Multi threading es a la aplicación lo que Multi tasking es al Sistema Operativo.

Java es uno de los primeros lenguajes que ya nacieron pensando en hilos de ejecución concurrente. En lenguajes como C++, dichos procesos paralelos debían ser administrados por el Sistema Operativo, lo cual es muy pesado hablando de recursos, en cambio una aplicación multi thread en Java es controlada por la maquina virtual, es decir que para el SO se trata de un solo proceso. Por ejemplo, el *garbage collection* se ejecuta como un thread separado de baja prioridad.

Desde el momento que ejecutamos la aplicación se crea un primer hilo o thread, llamado precisamente *main*.

El constructor de la clase *Thread* recibe como parámetro un objeto destino que debe implementar la interface *Runnable*, y por tanto debe tener el método run (), que es el único definido en la interface. Este objeto destino es el que será controlado por el nuevo thread.

Este método run () puede ser llamado explícitamente, pero en ese caso correría en el mismo hilo. En cambio, cuando se ejecuta el método start () de cada Thread adicional, se bifurca la ejecución iniciando en paralelo el nuevo hilo y llamando implícitamente el método run ()

Veamos los métodos más usuales de esta clase, todos son *public void*:

| | METODO | UTILIZACION | EXCEPCION LANZADA |
|---|---|---|---|
| static | sleep(long n) | Pausa la ejecución del que está corriendo durante **n** milisegundos | InterruptedException |
| | resume() | Continua la ejecución después de ser suspendido | |
| | run() | Contiene el corazón del thread | |
| synchronized | start() | Inicia la operación del thread, llamando a run() | IllegalThreadStateException |
| | stop() | Lo elimina permanentemente | |
| | suspend() | Lo detiene temporalmente | |
| | join(Thread t) | Suspende la ejecución hasta que acabe **t** | |
| static | yield() | Cede el paso para que otro thread entre a ejecución | |
| | wait(long n) | Pausa la ejecución del que está corriendo durante **n** milisegundos, o hasta que otro lance un notify() | |

| | | | |
|---|---|---|---|
| | notify() | Despierta arbitrariamente algún thread “dormido” | |

Por ejemplo, si tenemos un Applet donde queremos que se despliegue continuamente la hora:

```
public void start () {
        while (true) {
                showStatus ( (new Date () ).toString () );
                try {
                        Thread.sleep (1000);        // pausa 1 segundo
                }
                catch (Exception e) {
                }
        }
}       // start del Applet
```

Esto congelaría toda la aplicación, pues el ciclo no termina nunca. Necesitamos un segundo thread de la siguiente manera:

```
// imports necesarios
public class Animacion extends Applet implements Runnable {
        Thread t = new Thread(this);       // segundo hilo
        // declarar e instanciar los demas atributos

        public void run() {             // llamado por t.start()
                while (true) {
                        showStatus ( (new Date () ).toString () );
                        try {
                                Thread.sleep (1000);        // pausa 1 segundo
                        }
                        catch (Exception e) {
                        }
                }
        }       // run

        public void init() {
                // add() visualmente los componentes
                t.start();       // llama a run()
        }
        public void start()    {
                t.resume();
        }       // start del applet
        public void stop()    {
                t.suspend();
        }       // stop del applet
        public void destroy() {
                t.stop();
        }
        // demas funcionalidad del hilo primario del applet
}       // clase Animacion
```

En el ejemplo anterior, la aplicación crea un objeto thread y se vuelve su propio destino; esto es simple pues no necesitamos nuevas clases, pero tenemos el inconveniente de que solo puede crearse un thread adicional.

Si necesitáramos más de un thread adicional, hay que crear una nueva clase que herede de *Thread*, o bien que implemente la interface *Runnable* y que contenga un objeto de la clase *Thread.*

Es importante mencionar que no siempre podemos tener certeza total de los resultados, pues después de que arranca cada hilo, el tiempo y orden en que entran y salen de ejecución no es determinístico.

## Sincronización de Threads

Aunque operan independientemente uno de otro, puede existir código que podría causar problemas si se ejecutara simultáneamente en más de un thread; dicho código no se considera seguro o “thread-safe”.

Para protegernos contra estos problemas, debemos declarar el método que contiene dicho código con la palabra *synchronized*; con esto, el método solo puede ser llamado por un thread a la vez, hasta terminar, y si otros threads lo solicitan, deberán esperar su turno.

También es muy importante que un método *synchronized* sea lo más breve posible.

## Referencias

“Mastering Java 2.0”
Sun Microsystems