PROGRAMACIÓN MULTIHILO.

2.4. LA CLASE THREAD.

Los métodos para gestionar los hilos son:

METODO	ACCION
void start()	Provoca la llamada al método run() para que dé comienzo la ejecución del hilo.
void run()	El hilo comienza su ejecución tras un *start()*. Independientemente de que haya sido construido a partir de la interfaz *Runnable* o de la clase *Thread*.
String getName()	Devuelve el nombre del hilo
void setName(string nombre)	asigna el nombre al hilo
int getPriority()	Devuelve la prioridad de un hilo
void setPriority(int prioridad)	asigna la prioridad indicada al hilo. Cada hilo tiene una prioridad, que es un valor entero entre 1 y 10, de modo que cuanto mayor sea el valor, mayor es la prioridad.
boolean isAlive()	Devuelve *true* si está en ejecución y *false* en caso contrario. Un hilo está vivo si ha sido lanzado con start() y no ha muerto todavía.
~~void resume()~~	reanuda la ejecución de un hilo suspendido. Método obsoleto
void sleep(long milseg)	hace que el thread actual pase del estado ejecutable a dormido y permanezca en dicho estado durante los milisegundos especificados como parámetro. Una vez que se ha cumplido el tiempo, el thread despierta y pasa automáticamente al estado de ejecutable. Este método puede lanzar una **InterruptedException**, por lo tanto, las llamadas hacia él deben envolverse en un bloque **try … catch**
~~void stop()~~	detiene la ejecución de un hilo. Método obsoleto
~~void suspend()~~	este método suspende un hilo, su estado pasa de ejecutable a suspendido inmediatamente y sólo puede ser reactivado (pasado al estado ejecutable) llamando a su método *resume()*. Método obsoleto
Thread currentThread()	devuelve una referencia al hilo que se está ejecutando actualmente.

void isDaemon()	devuelve verdadero si el hilo es daemon
void join()	Hace que el *thread* que se está ejecutando actualmente pase al estado "esperando" indefinidamente hasta que muera el *thread* sobre el que se realiza el `join()`.
void join(long miliseg)	espera como mucho los milisegundos indicados para que el hilo muera
void setDaemon (boolean on)	marca el hilo como daemon si el parámetro **on** es verdadero o como hilo de usuario si es falso. El método debe ser llamado antes de que el hilo sea lanzado. Los hilos demonio están supeditados a los hilos que los han creado, de tal manera que cuando el creador termina, sus hijos "demonio" también finalizan.
String toString()	devuelve una representación en forma de cadena del hilo, incluyendo su nombre, prioridad y grupo
void yield()	hace que el hilo que se está ejecutando actualmente pase al estado listo, permitiendo a otro hilo ganar el procesador.
void destroy()	destruye el hilo sin realizar ningún tipo de limpieza
void interrupt()	interrumpe la ejecución del hilo
boolean interrupted()	comprueba si el hilo actual ha sido interrumpido
void wait() void wait(long miliseg)	pondría el hilo en el estado "esperando" indefinidamente, hasta que el thread reciba un *notify() o notifyAll().* si le indicamos un tiempo estará esperando durante ese tiempo.

CONSTRUCTORES

public Thread()	crea un nuevo objeto Thread.
public Thread (String nombre)	crea un nuevo objeto Thread asignándole el nombre indicado
public Thread (Runnable target)	crea un nuevo objeto Thread. target es el objeto que contiene el método **run()** que será invocado al lanzar el hilo con **start()**
public Thread (Runnable target, String name)	crea un nuevo objeto Thread, asignándole el nombre indicado. target es el objeto que contiene el método **run()** que será invocado al lanzar el hilo con **start()**

ATRIBUTOS	
int MIN_PRIORITY	la prioridad mínima que un hilo puede tener
int NORM_PRIORITY	la prioridad por defecto que se le asigna a un hilo
int MAX_PRIORITY	la prioridad máxima que un hilo puede tener.

2.5. OBTENER EL HILO PRINCIPAL.

Todo programa Java tiene al menos un hilo, llamado *hilo principal* y que podemos observar con el método *currentThread*, que obtiene el hilo actual.

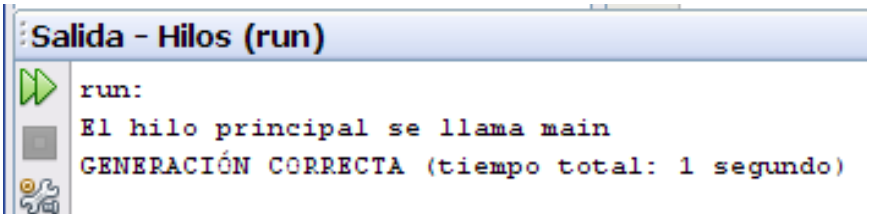
Este hilo es especial por dos razones:

- Desde él se crearán el resto de hilos del programa.
- Debe ser el último hilo que termine su ejecución. (Si un hilo principal finaliza antes que un hijo Java puede bloquearse, hang).

El método run() es el punto de entrada de un nuevo hilo de ejecución concurrente dentro de un programa. El hilo termina cuando finalice el método run().

```
public class HiloPrincipal {
    public static void main (String args[]) {
        Thread hilo = Thread.currentThread();

        System.out.println("El hilo principal se llama "+hilo.getName());
    }
}
```



2.6. MÚLTIPLES PROCESOS.

Podemos crear y ejecutar múltiples hilos en un mismo programa: basta con dar a cada hilo un nuevo objeto.
A continuación tienes un ejemplo en el que se crean 4 hilos y esperamos a que termine cada uno antes de finalizar la aplicación principal, cada uno imprima su nombre una vez por segundo, obtenemos además una ejecución secuencial ordenada.

```
Primero está ejecutándose...
Primero está ejecutándose...
Primero está ejecutándose...
Primero está ejecutándose...
Primero ha finalizado.
Segundo está ejecutándose...
Segundo está ejecutándose...
Segundo está ejecutándose...
Segundo está ejecutándose...
Segundo ha finalizado.
Tercero está ejecutándose...
Tercero está ejecutándose...
Tercero está ejecutándose...
Tercero está ejecutándose...
Tercero ha finalizado.
Cuarto está ejecutándose...
Cuarto está ejecutándose...
Cuarto está ejecutándose...
Cuarto está ejecutándose...
Cuarto ha finalizado.
FIN DE LA APLICACION PRINCIPAL
```

```java
    class Multiples extends Thread{

    //constructor
    public Multiples (String nombre){
        super(nombre);
    }

    //redefinición del método run(), que es el que contiene
    //las indicaciones de lo que hará el hilo
    public void run(){
        try{
            for (int i=0;i<4;i++){
                //Mostrará el nombre del hilo actual
                System.out.println((Thread.currentThread()).getName()+" está ejecutándose.....");
                //Duermo el hilo actual un segundo
                Thread.sleep(1000);

            }
        }catch (InterruptedException ex){}
        System.out.println((Thread.currentThread()).getName()+" ha finalizado.");
    }
}
```

```java
public class MainMultiples{
    public static void main(String args[]){

        Multiples hilo1 = new Multiples("Primero");
        Multiples hilo2 = new Multiples("Segundo");
        Multiples hilo3 = new Multiples("Tercero");
        Multiples hilo4 = new Multiples("Cuarto");

        try{

            hilo1.start();
            hilo1.join();

            hilo2.start();
            hilo2.join();

            hilo3.start();
            hilo3.join();

            hilo4.start();
            hilo4.join();

        }catch (InterruptedException ex){}

        System.out.println("Fin de la aplicación principal");
    }

}
```

3.AGRUPAMIENTO DE HILOS.

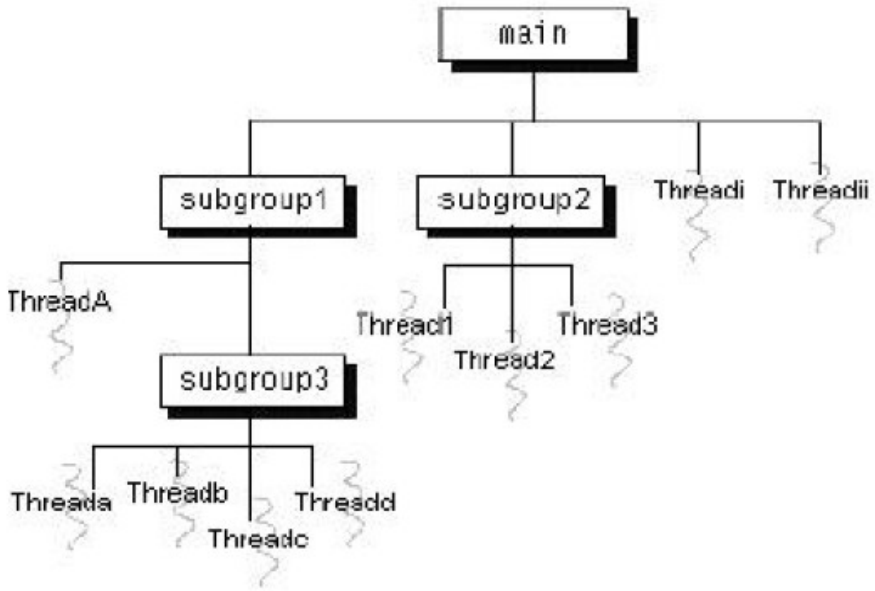
3.1 GRUPOS DE HILOS.

Todo hilo de Java es un miembro de un grupo de hilos. Los grupos de hilos proporcionan un mecanismo de reunión de múltiples hilos dentro de un único objeto y de manipulación de dichos hilos en conjunto, en lugar de una forma individual.

Por ejemplo, se pueden arrancar o suspender todos los hilos que están dentro de un grupo con una única llamada al método. Los grupos de hilos de Java están implementados por la clase ThreadGroup en el paquete java.lang.

Cuando se arranca un programa, el sistema crea un ThreadGroup llamado main. Si en la creación de un nuevo hilo no se especifica a qué grupo pertenece, automáticamente pasa a pertenecer al threadgroup del hilo desde el que ha sido creado (conocido como current threadgroup). Si en dicho programa no se crea ningún ThreadGroup adicional, todos los hilos creados pertenecerán al grupo main (en este grupo se encuentra el método main()).

Una vez que un hilo ha sido asociado a un grupo de hilos, no puede cambiar de grupo.



3.2. CREACIÓN DE UN HILO EN UN GRUPO DE FORMA EXPLÍCITA.

Como hemos mencionado anteriormente, un hilo es un miembro permanente de aquel grupo de hilos al cual se unió en el momento de su creación (no tenemos la posibilidad de cambiarlo posteriormente). De este modo, si quieres poner tu nuevo hilo en un grupo de hilos distinto del grupo por defecto, debes especificarlo explícitamente cuando lo creas.

Para conseguir que un hilo pertenezca a un grupo concreto, hay que indicarlo al crear el nuevo hilo, según uno de los siguientes constructores:

- public Thread(ThreadGroup grupo, Runnable destino)
- public Thread(ThreadGroup *grupo*, String *nombre)*
- public Thread(ThreadGroup grupo, Runnable destino, String *nombre)*

Cada uno de estos constructores crea un nuevo hilo, lo inicializa en base a los parámetros `Runnable` y `String`, y hace al nuevo hilo miembro del grupo especificado.

Por ejemplo, la siguiente muestra de código crea un grupo de hilos (`myThreadGroup`) y entonces crea un hilo (`myThread`) en dicho grupo

```
ThreadGroup miGrupoHilo = new ThreadGroup("Mi grupo hilos ");

Thread miHilo = new Thread(miGrupoHilos, "un hilo para mi grupo" );
```

El `ThreadGroup` pasado al constructor `Thread` no tiene que ser necesariamente un grupo que hayas creado tú, puede tratarse de un grupo creado por el sistema de ejecución de Java, o un grupo creado por la aplicación en la cual se está ejecutando un a*pplet*.

3.3 LA CLASE THREADGROUP

La clase `ThreadGroup` es la implementación del concepto de grupo de hilos en Java. Ofrece, por tanto, la funcionalidad necesaria para la manipulación de grupos de hilos para las aplicaciones Java. Un objeto `ThreadGroup` puede contener cualquier número de hilos. Los hilos de un mismo grupo generalmente se relacionan de algún modo, ya sea por quién los creó, por la función que llevan a cabo, o por el momento en que deberían arrancarse y parar.

El grupo de hilos de más alto nivel en una aplicación Java es el grupo de hilos denominado `main`. La clase `ThreadGroup` tiene métodos que pueden ser clasificados como sigue:

- *Collection Managment Methods* (Métodos de administración del grupo): métodos que manipulan la colección de hilos y subgrupos contenidos en el grupo de hilos.

- *Methods That Operate on the Group* (Métodos que operan sobre el grupo): estos métodos establecen u obtienen atributos del objeto `ThreadGroup`.

- *Methods That Operate on All Threads within a Group* (Métodos que operan sobre todos los hilos dentro del grupo): este es un conjunto de métodos que desarrollan algunas operaciones, como inicio y reinicio, sobre todos los hilos y subgrupos dentro del objeto `ThreadGroup`.

- *Access Restriction Methods* (Métodos de restricción de acceso): `ThreadGroup` y `Thread` permiten al administrador de seguridad restringir el acceso a los hilos en base a la relación de miembro/grupo con el grupo

Métodos de administración del grupo

La clase `ThreadGroup` proporciona un conjunto de métodos que manipulan los hilos y los subgrupos que pertenecen al grupo y permiten a otros objetos solicitar información sobre sus miembros. Por ejemplo, puedes llamar al método `activeCount` de `ThreadGroup` para conocer el número de hilos activos que actualmente hay en el grupo. El método `activeCount` se usa generalmente con el método `enumerate` para obtener un vector (array) que contenga las referencias a todos los hilos activos en un `ThreadGroup`.

Métodos que operan sobre el grupo

La clase ThreadGroup da soporte a varios atributos que son establecidos y recuperados de un grupo de forma global (hacen referencia al concepto de grupo, no a los hilos individualmente).
Se incluyen atributos como la prioridad máxima que cualquiera de los hilos del grupo puede tener, el carácter "*daemon*" o no del grupo, el nombre del grupo, y el padre del grupo.

Los métodos que recuperan y establecen los atributos de `ThreadGroup` operan a nivel de grupo. Consultan o cambian el atributo del objeto de la clase `ThreadGroup`, pero no hacen efecto sobre ninguno de los hilos pertenecientes al grupo. La siguiente es una lista de métodos de `ThreadGroup` que operan a nivel de grupo:

- `getMaxPriority` y `setMaxPriority`
- `getDaemon` y `SetDaemon`
- `getName`
- `getParent` y `parentOf`
- `toString`

Métodos que operan sobre todos los hilos de un grupo

La clase `ThreadGroup` tiene tres métodos que te permiten modificar el estado actual de todos los hilos pertenecientes al grupo:

- `resume`
- `stop`
- `suspend`

Estos métodos suponen el cambio correspondiente de estado para todos y cada uno de los hilos del grupo, así como los de sus subgrupos. No se aplican, por tanto, a un nivel de grupo, sino que se aplican individualmente a todos los miembros.

Métodos de restricción de acceso

La clase `ThreadGroup` no impone ninguna restricción de acceso por sí sola, como permitir a los hilos de un grupo consultar o modificar los hilos de un grupo diferente. En lugar de esto, las clases `Thread` y `ThreadGroup` cooperan con los administradores de seguridad (subclases de la clase `SecurityManager`), la cual impone las restricciones de acceso basándose en la pertenencia de los hilos a los grupos.