Unión Europea

Fondo Social Europeo

El FSE invierte en tu futuro

UD 2 INTRODUCCIÓN A LOS HILOS (PSP)

CFGS Desarrollo de Aplicaciones Multiplataforma

Pepe

GENERALITAT VALENCIANA

Conselleria d'Educació, Investigació, Cultura i Esport

GOBIERNO DE ESPAÑA

MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

Continguts

1 CONCEPTOS PREVIOS

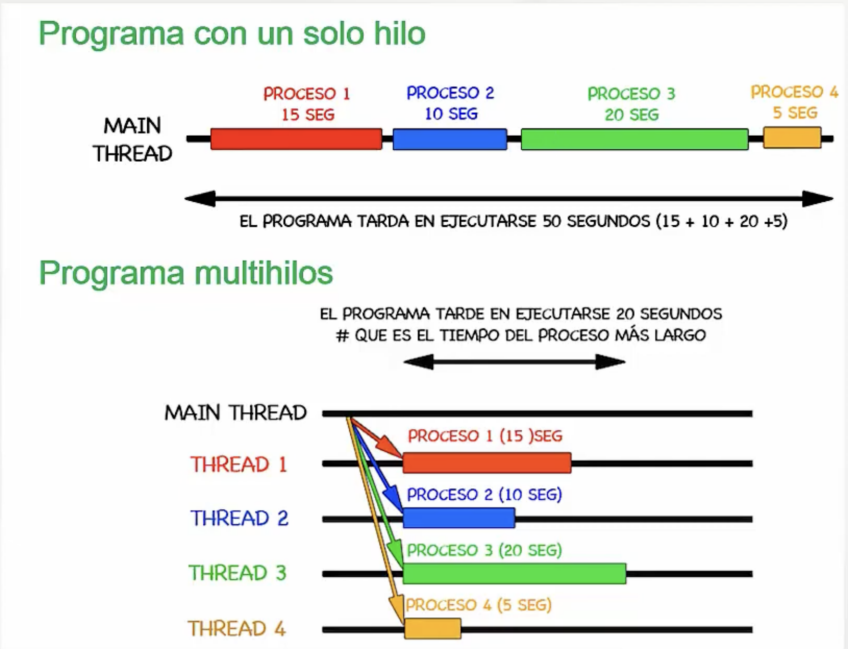
Un hilo (también denominado “proceso ligero”) es una secuencia de código en ejecución **dentro del contexto de un proceso**.

Los hilos no pueden ejecutarse ellos solos, necesitan la supervisión de un proceso padre para ejecutarse.

Se asemejan mucho a los procesos ya que también comparten la CPU, solo hay un hilo activo (en ejecución) en un instante dado y pueden crear sus propios hilos hijos.

La principal diferencia es que los hilos **pueden compartir recursos**, por lo que cuando un hilo modifica un dato, los otros hilos pueden acceder a ese dato modificado.

Los hilos son más frecuentes de lo que parece. De hecho, todos los programas con interfaz gráfica son multihilo porque los eventos y las rutinas de dibujado de las ventanas corren en un hilo distinto al principal. Por ejemplo en Java, AWT o la biblioteca gráfica Swing usan hilos. Si en “Programación” de 1er curso usaste Swing, ahora entenderás algunas cosas “raras” que aparecían en el código.



(a) Tiempo de ejecución

2 Manejo básico de hilos. Crear y lanzar hilos

2.1 Definiendo un hilo (thread)

En Java, un hilo se representa mediante una instancia de la clase *java.lang.thread*. Este objeto Thread se emplea para iniciar, detener o cancelar la ejecución del hilo de ejecución. Los hilos o threads se pueden implementar o definir de dos formas:

1) Extendiendo la clase Thread.
2) Mediante la interfaz Runnable .

Extendiendo la clase Thread.

Esta subclase debe sobreescribir el método **run()** con las acciones que el hilo debe desarrollar. Por otro lado, la clase que quiera lanzar ese método, lo hará a través del método **start()**

```
public class Saludo extends Thread{

    public void run() {
        System.out.println("Saludo desde extends Thread!!!");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Saludo hilo1= new Saludo(); //Se crea un objeto Thread, el hilo
           hilo1

        hilo1.start(); //Invoca a start y pone en marcha el hilo1
    }

}
```

Implementando la interfaz Runnable

Este será realmente el hilo. Una vez creado el hilo, para ponerlo en marcha o iniciarlo invocaremos al método start() del objeto thread (el hilo que hemos creado). El siguiente ejemplo muestra cómo crear un hilo implementado Runnable. El hilo que se crea (objeto thread hilo1) imprime un mensaje de saludo, como en el caso anterior.

```
public class Saludo implements Runnable{

    public void run() {
        System.out.println("Saludo desde implements  Runnable!!!");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
        Saludo miRunnable= new Saludo(); //Se crea un objeto

        Thread hilo1 = new Thread(miRunnable); //Se crea un objeto Thread(
            el hilo hilo1) pasando como argumento al constructor un objeto
            saludo

        hilo1.start(); //Invoca a start y pone en marcha el hilo1
    }

}
```

Entonces, ¿cuando utilizar uno u otro procedimiento?

`Extender la clase thread` es el procedimiento más sencillo, pero no siempre es posible. Si la clase ya hereda de alguna otra clase padre, no será posible heredar también de la clase thread (recuerda que Java no permite la herencia múltiple), por lo que habrá que recurrir al otro procedimiento.

`Implementar Runnable` siempre es posible, es el procedimiento más general y también el más flexible. Por ejemplo, piensa en la programación de applets, cualquiera de ellos tiene que heredar de la clase java.applet.Applet; y en consecuencia ya no puede heredar de thread si se quiere utilizar hilos. En este caso, no queda más remedio que crear los hilos implementando Runnable.

Cuando la Máquina Virtual Java (JVM) arranca la ejecución de un programa, ya hay un hilo ejecutándose, denominado hilo principal del programa, controlado por el método main(), que se ejecuta cuando comienza el programa y es el último hilo que termina su ejecución, ya que cuando este hilo finaliza, el programa termina.

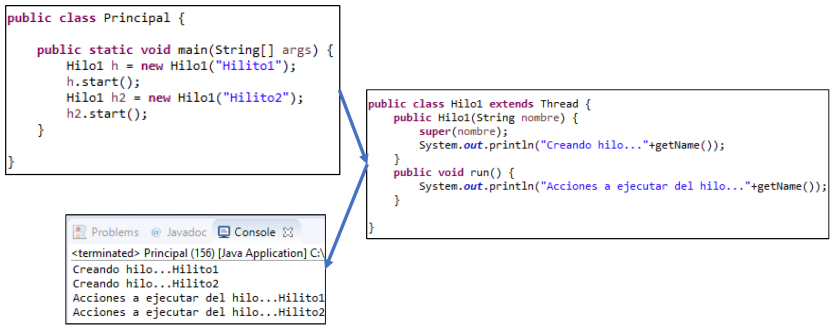
Siempre hay un hilo que ejecuta el método main(), y por defecto, este hilo se llama “main”.

Para saber qué hilo se está ejecutando en un momento dado, el hilo en curso, utilizamos el método currentThread() y que obtenemos su nombre invocando al método getName(), ambos de la clase thread.

```
public class Saludo implements Runnable{

    public void run() {
        System.out.println("Saludo desde implements  Runnable!!!");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Saludo miRunnable= new Saludo(); //Se crea un objeto

        Thread hilo1 = new Thread(miRunnable); //Se crea un objeto Thread(
            el hilo hilo1) pasando como argumento al constructor un objeto
            saludo
```

```
        hilo1.start(); //Invoca a start y pone en marcha el hilo1

        hilo1 = Thread.currentThread();
        //imprime el nombre del hilo en la Salida (función getName())
        System.out.println("El hilo se llama " + hilo1.getName() + "\n");
    }

}
```

P ara establecr el nombre a un hilo hemos de pasárselo desde el constructor de nuestra clase a la clase Thread mediante la palabra reservada “super()”. Más tarde, lo podemos recuperar con el método `getName()`:



(b) Extendido Thread

Veamos el funcionamiento de los hilos. Copia el siguiente hilo, fijate que hay dos filas comentadas, ejecuta de las diferentes manera con las dos lineas comentadas, y descomentando cada una de ellas y observa que pasa.

```
public class MyMainCounter {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
         MyCounterThread t = new MyCounterThread();
         t.start();

         //System.out.println("Hello!!");
         //  System.exit(0);
    }
}
```

```
public class MyCounterThread extends Thread{
     @Override
        public void run()
        {
            for (int i = 1; i <= 10; i++)
```

```
                System.out.println("Counting " + i);
        }
}
```

¿Qué pasa cuando volvemos a ejecutar el programa? Si ejecuta el programa varias veces, descubrirá que a veces cuenta hasta 10, a veces no cuenta nada… y a veces cuenta entre 1 y 10. Esto se debe a que el programa principal finaliza inesperadamente con el método de salida *System.exit(0);*. y luego todos sus hilos también se matan. Si el hilo comenzó a ejecutarse antes de que se eliminara su padre, podrá contar algunos números.

si descomentas *System.out.println("Hello!!");* Tu hilo cuenta hasta 10, y en algún punto intermedio de este conteo aparece el mensaje "¡¡Hola!!". Quizás se muestre antes del número 1, o después del número 7… Depende del momento en que el programa principal llegue al procesador para imprimir su mensaje.

De este ejemplo podemos llegar a algunas conclusiones:

- Cuando lanzamos un hilo desde nuestra aplicación principal, comienza su ejecución paralela e independiente.
- Cuando nuestra aplicación principal finaliza correctamente, nuestro hilo continúa ejecutando su tarea hasta que finaliza.
- Cuando nuestra aplicación principal se ve obligada a finalizar, nuestro hilo también finaliza inesperadamente. Para ser más precisos, si algún hilo de nuestra aplicación llama al método System.exit, todos los hilos finalizarán su ejecución.
- No hay forma de saber el orden exacto en el que la aplicación principal y sus subprocesos producirán sus resultados. Depende del planificador de tareas. De todos modos, aprenderemos brevemente cómo dar más tiempo de CPU a algunos subprocesos a expensas de otros, y también a sincronizar o coordinar subprocesos para producir resultados en un orden determinado.
- Después de iniciar un hilo, no podremos volver a llamar a su método de inicio. Pero podremos llamar a otros métodos para obtener su estado y algunas otras características, como veremos más adelante.

2.2 Estados de un hilo

Una vez que ya sabemos crear un hilo veamos sus estados:

1. NEW: Se crea el objeto, pero no se ejecuta. Queda a la espera de lanzar el método run().
2. RUNNABLE: Al llamar al método start() el hilo pasa a este estado.
3. DEAD: Se entra a este estado cuando finaliza el método run()

4. BLOCKED: Debe ocurrir una de las siguientes circunstancias para que se llegue a este estado:
    - Llamada al método sleep().
    - EL hilo espera a una operación de entrada/salida.
    - Llamada al método wait(). El hilo no volverá a ejecutarse hasta que reciba los mensajes notify() o notifyAll()
    - Llamada al método suspend(). No se volverá ejecutable hasta que reciba el mensaje resume().

El método `getState()` devuelve una constante con el estado del hilo:

1. NEW: Sin iniciar.
2. RUNNABLE: Ejecutandose.
3. BLOCKED: Bloquedado.
4. WAITING: Esperando indefinidamente, a que un evento le active.
5. TIMED_WAITING: Espera limitada por tiempo.
6. TERMINATED: Finalizado.

2.3 Detener temporalmente un hilo

¿Qué significa que un hilo se ha detenido temporalmente? Significa que el hilo ha pasado al estado "No Ejecutable". Y, ¿cómo puede pasar un hilo al estado "No Ejecutable"? Un hilo pasará al estado "No Ejecutable" o "Detenido" por alguna de estas circunstancias:

- **El hilo se ha dormido**. Se ha invocado al método `sleep()` de la clase Thread indicando el tiempo que el hilo permanecerá detenido. Transcurrido ese tiempo, el hilo se vuelve "Ejecutable", en concreto pasa a "Preparado".
- **El hilo está esperando**. El hilo ha detenido su ejecución mediante la llamada al método `wait()` y no se reanudará, pasando a "Ejecutable" (en concreto "Preparado"), hasta que se produzca una llamada al método `notify()` o `notifyAll()` por otro hilo.
- **El hilo se ha bloqueado**. El hilo está pendiente de que finalice una operación de E/S en algún dispositivo, o a la espera de algún otro tipo de recurso; y éste ha sido bloqueado por el sistema operativo. Cuando finalice el bloqueo volverá al estado "Ejecutable", en concreto "Preparado".

2.4 Finalizar un hilo

La forma natural de que finalice un hilo es cuando termina de ejecutarse su método `run()`, pasando al estado 'Muerto'.

Una vez que el hilo ha muerto, **no lo puedes iniciar otra vez con start()**. Si en tu programa deseas realizar otra vez el trabajo desempeñado por el hilo, tendrás que:

1) Crear un nuevo hilo con new().
2) Iniciar el hilo con start().

Puedes utilizar el método `isAlive()` de la clase Thread para comprobar si un hilo está vivo o no. Un hilo se considera que está vivo (alive) desde la llamada a su método start() hasta su muerte. isAlive() devuelve verdadero (true) o falso (false), según que el hilo esté vivo o no.

Cuando el método isAlive() devuelve:

- *False*: sabemos que estamos ante un nuevo hilo recién "creado" o ante un hilo "muerto".
- *True*: sabemos que el hilo se encuentra en estado "ejecutable" o "no ejecutable".

El método `stop()` de la clase Thread (actualmente en desuso) también finaliza un hilo, pero es poco seguro. **No debes utilizarlo**. *Te lo indicamos aquí simplemente porque puede que encuentres programas utilizando este método*.

En el siguiente ejemplo te proporcionamos un programa cuyo hilo principal lanza un hilo secundario que realiza una cuenta atrás desde 10 hasta 1. Desde el hilo principal se verificará la muerte del hilo secundario mediante la función isAlive(). Además mediante el método getState() de la clase Thread vamos obteniendo el estado del hilo secundario. Se usa también el método thread.join() que espera hasta que el hilo muere.

```
public class HiloAuxiliar extends Thread
{
    //código del hilo
      public void run()
      {
        for(int i=10;i>=1;i--)
          System.out.print(i+",");
      }
      public static void main(String[] args) throws Exception
        {
            HiloAuxiliar hilo1 = new HiloAuxiliar();
            //Crea un nuevo hilo. El hilo está en estado Nuevo (new)
            System.out.println("Hilo Auxiliar Nuevo: Estado=" + hilo1.
               getState() );
            //Obtenemos el estado del thread hilo1
            hilo1.start();

            //Inicia el thread hilo1 y pasa al estado Ejecutable
            System.out.println("Hilo Auxiliar Iniciado: Estado="+ hilo1.
               getState());

            System.out.println("Esperamos a que termine el hilo");
```

```
            System.out.println("\n Hilo Auxiliar Muerto: Estado="+ hilo1.
                getState());
        }
    }
```

2.5 Dormir un hilo con sleep

El método sleep() de la clase Thread recibe como argumento el tiempo que deseamos dormir el hilo que lo invoca. Cuando transcurre el tiempo especificado, el hilo vuelve a estar "Ejecutable" ("Preparado") para continuar ejecutándose.

```
Thread.sleep(long milisegundos);
```

Cualquier llamada a sleep() puede provocar una excepción que el compilador de Java nos obliga a controlar ineludiblemente mediante un bloque try-catch.

```
public class Principal {


    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Hilo h1=new Hilo("hilo 1");
        Hilo h2=new Hilo("hilo 2");
        Hilo h3=new Hilo("hilo 3");
        Hilo h4=new Hilo("hilo 4");

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

}
```

```
public class Hilo extends Thread{

    public Hilo(String name){
        super(name);
    }
    public void run() {

                for(int i=0;i<5;i++) {
                System.out.println(getName()+" -> "+i);
                }

                try {
            //espera a que el thread hilo1 muera
```

```
                Thread.sleep(500);
            } catch (InterruptedException e) {
            System.out.println(e);
            }
        }

    }
```

2.6 Resumen de métodos

MÉTODOS	MISIÓN
sleep(mils)	Hace que el hilo se duerma durante mils milisegundos
getName()	Devuelve el nombre del hilo
toString()	Devuelve una representación en formato cadena de este hilo incluyendo el nombre, la prioridad y el grupo de hilos al que pertenece: Thread[Hilo1,2,main]
currentThread()	Devuelve una referencia al objeto hilo actualmente en ejecución (devuelve por tanto un objeto Thread)
activeCount()	Devuelve el número de hilos activos (tipo int)
getPriority	Devuelve la prioridad del hilo (tipo int)
setPriority(int p)	Se establece la prioridad del hilo que será un entero que irá de 1 a 10 y que por defecto es 5.

Mínima: 1 Normal: 5 Máxima: 10 |

3 Grupos de hilos

Java nos permite agrupar algunos hilos para tratarlos como una sola unidad. De esta manera, podemos tener algunos hilos realizando una tarea y controlarlos independientemente del número total de hilos en el grupo.

Para gestionar grupos, tenemos la clase `ThreadGroup`. Podemos crear un grupo básico con un nombre dado, e incluso un grupo dentro de otro grupo, con su propio nombre:

```
ThreadGroup g1 = new ThreadGroup("Grupo principal");
ThreadGroup g2 = new ThreadGroup(g1, "Grupo adicional dentro del grupo
    principal");
```

Para agregar hilos a un grupo, podemos usar algunos de los constructores disponibles en la clase `Thread`. Por ejemplo, si creamos un hilo extendiendo la clase `Thread`, podemos agregarlo a un grupo con este constructor (y otros más, consulta la API para más detalles):

```
public Thread(ThreadGroup group, String name);
```

Si creamos el hilo implementando la interfaz `Runnable`, podemos agregarlo con estos constructores (y otros más, consulta la API para más detalles):

```
public Thread(ThreadGroup group, Runnable target);
public Thread(ThreadGroup group, Runnable target, String name);
```

Una vez que hemos agregado los hilos a un grupo, hay algunos métodos útiles dentro de la clase `ThreadGroup`, tales como:

- `activeCount`: devuelve cuántos hilos en este grupo (y sus subgrupos) están actualmente activos (no han terminado).
- `enumerate(Thread[] array)`: copia en el array especificado cada hilo activo del grupo (y sus subgrupos).
- `interrupt`: interrumpe todos los hilos en el grupo.
- `setMaxPriority` / `getMaxPriority`: establece/obtiene la prioridad máxima de los hilos en el grupo.

3.0.0.1 Ejemplo

El siguiente ejemplo crea algunos hilos a partir de una clase que implementa la interfaz `Runnable`. Se supone que estos hilos generan un número aleatorio entre 1 y 10, duermen el número de segundos especificado por este número aleatorio, y luego imprimen un mensaje en la pantalla. Pero tan pronto como el primer hilo termine su tarea, se interrumpe todo el grupo.

El código para el objeto `Runnable` es:

```
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class MyRandomMessage implements Runnable {
    Random r = new Random(System.currentTimeMillis());

    @Override
    public void run() {
        int time = r.nextInt(10) + 1;
        try {
            TimeUnit.SECONDS.sleep(time);
            System.out.println("El hilo esperó " + time +
                " segundos y terminó.");
        } catch (Exception e) {}
    }
```

```
}
```

Luego, nuestro programa principal sería así:

```
public static void main(String[] args) {
    ThreadGroup g = new ThreadGroup("Mensajes aleatorios");
    MyRandomMessage m = new MyRandomMessage();
    Thread t1 = new Thread(g, m);
    Thread t2 = new Thread(g, m);
    Thread t3 = new Thread(g, m);
    t1.start();
    t2.start();
    t3.start();

    while (g.activeCount() == 3) {
        try {
            Thread.sleep(100);
        } catch (Exception e) {}
    }
    g.interrupt();
}
```

Tan pronto como un hilo termine, el método `activeCount` devolverá un número menor que 3, y el hilo principal terminará su bucle e interrumpirá todos los hilos. Si los otros hilos aún están esperando que expire su tiempo, serán interrumpidos, se lanzará una excepción y no imprimirán su mensaje de finalización.

---

4 Ejercicios

1. Crea un programa `HiloEjemplo1.java` que lanze n hilos con su nombre y que ejecuten un contador en el método run.

```
    CREANDO HILO:Hilo 1
    CREANDO HILO:Hilo 2
    CREANDO HILO:Hilo 3
    Hilo:Hilo 1 = 0
    Hilo:Hilo 1 = 1
    Hilo:Hilo 1 = 2
    Hilo:Hilo 2 = 0
    Hilo:Hilo 2 = 1
    Hilo:Hilo 2 = 2
    Hilo:Hilo 3 = 0
    Hilo:Hilo 3 = 1
    Hilo:Hilo 3 = 2
```

```
    3 HILOS INICIADOS...y finalizados
```

1. Vamos a crear un programa `Simple1.java` que lance 5 hilos. Cada hilo incrementará una variable contador de tipo entero en 1000 unidades. Esta variable estará compartida por todos los hilos. Comprueba el resultado final de la variable y reflexiona sobre el resultado.
2. Vamos a crear un programa que actualize el estado de la cuenta corriente cada vez que un usuario retire dinero, para ello tendremos:
   - La clase `Cuenta` que constará de:
     - Como parámetro int saldo => Saldo de la cuenta
     - int getSaldo(); Devolverá el saldo.
     - void restar(int cantidad) Cuando retiramos dinero
     - void RetirarDinero(int cant, String nom)
       * Si el saldo >= cant Saldrá un mensaje `SE VA A RETIRAR SALDO (ACTUAL ES: ____)`; y se restará de la cantidad.
       * Si no mostrará elmensaje `No puede retirar dinero, NO HAY SALDO( saldo actual)`
   - La clase `Principal` o main donde instanciaremos a la clase Cuenta con una cantidad **x**, además instanciaremos a difrentes hilos de ejecución con los nombres de usuarios y la variable la clase Cuenta.
   - La clase `SacarDinero` tendrá:
     - Como parámetro private Cuenta c. Referencia a la clase cuenta.
     - y en el método *run()* se invocará al método *RetirarDinero* de la clase Cuenta, donde le pasaremos el saldo y el nombre... La ejecución debe de ser algo como:

```
Resultado:
   Ana: SE VA A RETIRAR SALDO (ACTUAL ES: 40)
   Juan: SE VA A RETIRAR SALDO (ACTUAL ES: 40)

   Juan retira =>10 ACTUAL(20)
   Ana retira =>10 ACTUAL(20)
```