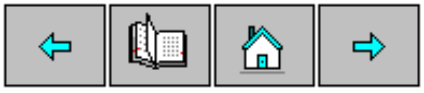
La clase derivada de *Thread*

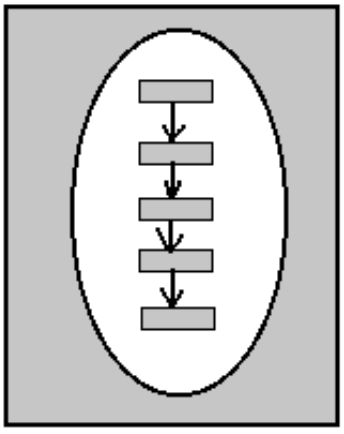


Subprocesos (threads)

Todos los programadores están familiarizados con los programas secuenciales. Todos ellos tienen un principio, una secuencia de ejecución y un final. Por ejemplo, una aplicación comienza en la función *main*, ejecuta las sentencias del cuerpo de dicha función en orden consecutivo y el programa acaba cuando se llega al final de dicha función.

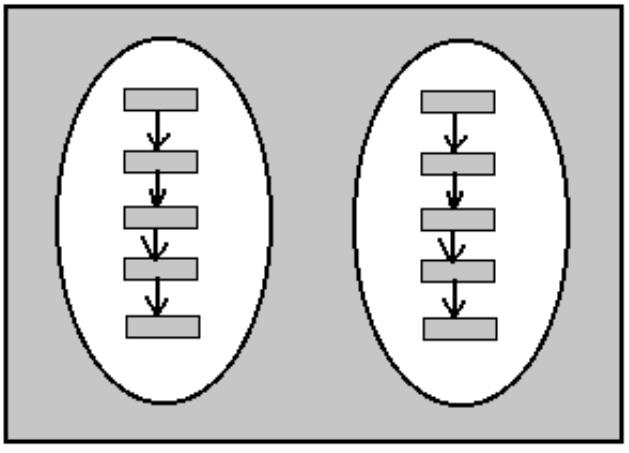
Un subproceso es similar a un programa secuencial, tiene un principio, una secuencia y un final. La diferencia fundamental estriba en que un subproceso no es un programa que pueda correr aislado, sino que se ejecuta dentro de un programa.

En la figura se muestra un subproceso que corre dentro de un programa.



un programa
un subproceso

En esta otra figura se muestran dos subprocesos corriendo dentro de un mismo programa



un programa
dos subprocesos

Todos estamos acostumbrados a manejar programas que contienen varios subprocesos en ejecución. Por ejemplo, mientras escribimos en el procesador de textos Word un subproceso se encarga de verificar la ortografía, y de señalar la palabra que no está correctamente escrita, e incluso de corregirnos según vamos escribiendo.

La ilusión de la ejecución paralela de los subprocesos en un sistema que tiene una única CPU proviene del hecho de que cada subproceso tiene la oportunidad de ejecutar una porción de código cada vez a intervalos regulares. Esta aproximación se denomina *timeslicing*. Como la CPU ejecuta millones de instrucciones por segundo, la percepción para el usuario es el de una ejecución en paralelo.

El método *run*

El método *run* es el corazón del subproceso, es donde tiene lugar la acción del subproceso. Hay dos modos de proporcionar el el método *run* a un subproceso:

- Derivando una clase de *Thread* y redefiniendo el método *run*
- Implementando el interface *Runnable* y definiendo la función *run* de dicho interface.

La razón de que existan estas dos posibilidades es que en Java no existe la herencia múltiple. Si una clase deriva de otra no podemos hacer que derive también de *Thread*. Por ejemplo, un applet deriva de la clase base *Applet* por tanto, ha de implementar el interface *Runnable* para que pueda definir el método *run*. En el estudio de la animación veremos esta aproximación.

Derivando de la clase *Thread*

 **thread0**: Hilo.java, ThreadApp.java

Creamos una clase derivada de *Thread* que redefine el método *run*.

```
public class Hilo extends Thread {
  public Hilo(String nombre) {
    super(nombre);
  }
     public void run(){
        //definir run...
    }
}
```

La clase *Thread* tiene varios constructores, además del constructor por defecto (sin argumentos). Al contructor de la clase derivada *Hilo* le pasamos el nombre del subproceso y éste se lo pasa al constructor de la clase base *Thread* mediante la palabra reservada **super**.

Creando el subproceso

Creamos dos objetos (o dos subprocesos) de la clase *Hilo*, en el cuerpo de la función *main* de la aplicación.

```
        Hilo hilo1=new Hilo("Subproceso 1");
        Hilo hilo2=new Hilo("Supproceso 2");
```

El nombre de cada subproceso se pasa en el único parámetro de su constructor.

El subproceso está en el estado New Thread, el subproceso está inicializado, y listo para ponerlo en marcha llamando al método *start* de la clase base *Thread*.

Poniendo en marcha el subproceso

El método *start* crea los recuros del sistema necesarios para que el subproceso se ejecute y a continuación, llama al método *run*, el subproceso se dice que está en el estado Runnable.

```
        hilo1.start();
        hilo2.start();
```

Corriendo el subproceso

La redefinición de la función miembro *run* en la clase derivada *Hilo*, es muy simple. Se ejecuta un bucle **for**, y dentro del bucle se imprime, el nombre del subproceso, que se obtiene mediante la función miembro *getName* de la clase *Thread* y el valor que va tomando la variable contador *i* del bucle.

```
     public void run(){
        for(int i=1; i<10; i++){
            System.out.println(getName()+": "+i);
    }
```

Ejecutando la aplicación

```
public class ThreadApp {
  public static void main(String[] args) {
        Hilo hilo1=new Hilo("Subproceso 1");
        Hilo hilo2=new Hilo("Supproceso 2");
        hilo1.start();
        hilo2.start();
  }
}
```

Al ejecutar la aplicación, se crea primero el objeto *hilo1*, luego el objeto *hilo2*.

El objeto *hilo1*, llama a *start*, y a continuación se llama a su función miembro *run*, que imprime el nombre del subproceso Subproceso1, y a continuación el valor de la variable contador *i*. La salida debería ser la siguiente.

```
Subproceso1: 1
Subproceso1: 2
Subproceso1: 3
Subproceso1: 4
Subproceso1: 5
Subproceso1: 6
```

```
Subproceso1: 7
Subproceso1: 8
Subproceso1: 9
```

El objeto *hilo2*, llama a *start* y a continuación a su función miembro *run*, que imprime el nombre del subproceso Subproceso2, y a continuación el valor de la variable contador *i*. La salida a continuación de la del cuadro anterior, debería ser la siguiente.

```
Subproceso2: 1
Subproceso2: 2
Subproceso2: 3
Subproceso2: 4
Subproceso2: 5
Subproceso2: 6
Subproceso2: 7
Subproceso2: 8
Subproceso2: 9
```

La salida que vemos en la consola no es la misma que se ha descrito, debido a que el sistema da la oprtunidad al segundo subproceso de ejecutarse tal como se ve en la figura de la izquierda un poco más abajo.

Pausa en la ejecución del subproceso

Podemos mejorar el método *run*, introduciendo una pausa durante la ejecución del subproceso

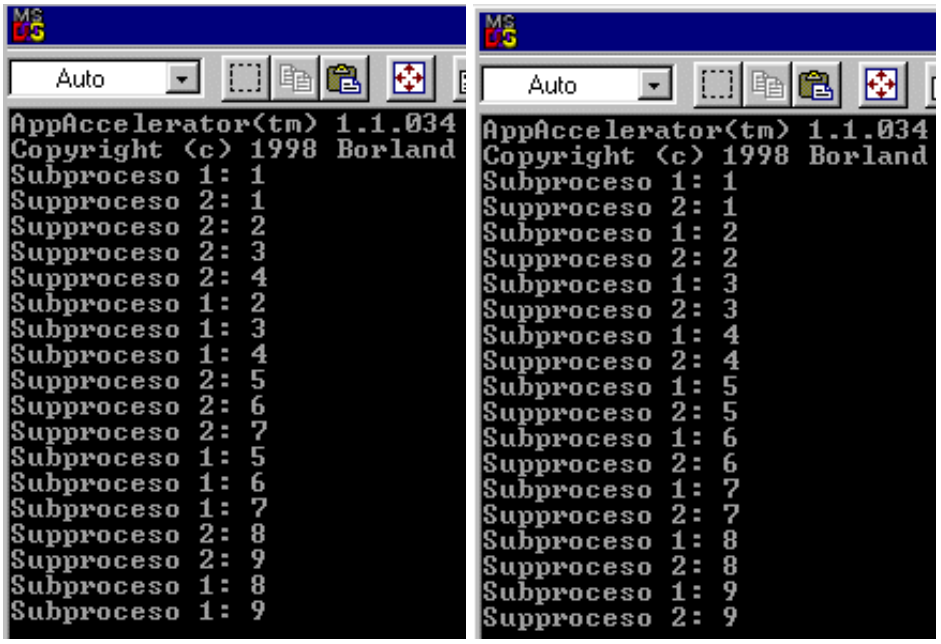
```
public void run(){
   for(int i=1; i<10; i++){
       System.out.println(getName()+": "+i);
       try{
           sleep(100);
       }catch(InterruptedException ex){}
   }
```

Cuando se llama a la función *sleep*, el subproceso pasa del estado Runnable al estado Not Runnable, dando la oprtunidad a otros subprocesos de ejecutarse. La función *sleep* se hereda de *Thread* y su argumento es el tiempo de pausa en milisegundos. El bloque **try..catch** que comprende al método es necesario pero no tiene por qué hacer nada específico.

Una vez que se ha llamado a la sentencia *sleep* el subproceso vuelve al estado Runnable. En general, un subproceso está en el estado Not Runnable cuando:

- Se llama al método *sleep*
- Se llama al método *supend*
- Se llama al método *wait*, para esperar hasta que se satisfaga alguna condición
- Cuando el subproceso está bloquedo en una operación Entrada/Salida.

Cuando volvemos a correr la aplicación la salida cambia (figura de la derecha). Los subprocesos 1 y 2 se ejecutan una vez cada uno, primero el Subproceso1 y luego el Subproceso2. Para obtener este resultado, quitar los delimitadores de comentarios **/*...*/** que anulan el bloque **try...catch** en la definición del método *run* de la clase *Hilo*.



Muerte del subproceso

Un proceso pasa al estado Death (muerto) cuando se completa su método *run*. Por ejemplo, cuando se completa el bucle **for**, se sale del bucle y se llega al final de la función *run*, el subproceso muere de muerte natural.

Un subproceso se muere cuando ya no es necesario, es decir cuando

- El método *run* finaliza su ejecución
- Se llama al método *stop* de la clase *Thread*

Un subproceso en el estado Death no puede ser revivido y ejecutado de nuevo.

```
public class Hilo extends Thread {
  public Hilo(String nombre) {
    super(nombre);
  }
     public void run(){
        for(int i=1; i<10; i++){
            System.out.println(getName()+": "+i);
            try{
                sleep(100);
            }catch(InterruptedException ex){}
        }
    }
}

public class ThreadApp {
  public static void main(String[] args) {
        Hilo hilo1=new Hilo("Subproceso 1");
        Hilo hilo2=new Hilo("Supproceso 2");
        hilo1.start();
        hilo2.start();
  }
}
```

Prioridades

thread1: Hilo.java, ThreadApplet1.java

Como hemos visto, en los ordenadores que tienen una única CPU, los subprocesos corren uno cada vez proporcionando la ilusión de procesos que se ejecutan al mismo tiempo. Se puede modificar la prioridad de los subprocesos después de su creación mediante *setPriority*. A esta función se le pasa un entero comprendido entre dos valores mínimo y máximo. Cuando mayor sea el entero, mayor será la prioridad con la que se ejecuta el correspondiente subproceso.

La clase *Thread* define tres constantes que representan los niveles de prioridad relativos para los subprocesos. Tomando 1 el valor de la mínima prioridad y 10 el valor de la máxima prioridad.

- MIN_PRIORITY
- MAX_PRIORITY
- NORM_PRORITY

Un ejemplo de subproceso de baja prioridad es el que libera la memoria no usada que se ejecuta en la Máquina Virtual Java. Aún cuando la liberación de memoria es una tarea muy importante, su baja prioridad evita que que el procesador esté ocupado demasiado tiempo en ella, dejando a los procesos críticos el uso prioritario de la CPU. Esto no significa que en un momento dado se pueda agotar toda la memoria y se bloquee el sistema, ya que cuando la memoria se agota, los subprocesos críticos entran en estado de espera, dejando a la CPU que ejecute el subproceso que libera la memoria no usada.

Para obtener el nivel del prioridad de un subproceso se usa la función *getPriority*

```
int prioridad=hilo1.getPriority();
```

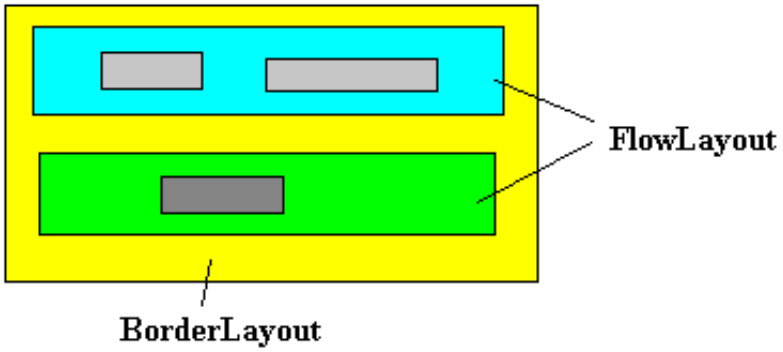
Teniendo en cuanta que la prioridad es una propiedad relativa, se puede cambiar la prioridad de un subproceso respecto de otro aumentando o disminuyendo su valor de prioridad, por ejemplo,

```
hilo2.setPriority(hilo1.getPriority()+1);
```

En el ejemplo anterior, podemos observar la prioridad, siempre que el tiempo de pausa, el argumento de la función *sleep*, sea pequeño (menor que 5). No obstante, vamos a crear un applet en el que podamos observar visualmente el efecto de la prioridad de dos subprocesos.

Diseño

Crear el applet, en modo de diseño (pestaña **Design**) situar dos paneles, uno en la parte superior y otro en la parte inferior. Sobre el panel superior poner dos controles de edición (*TextField*), sobre el panel inferior poner un botón (*Button*).



Cambiar las propiedades de los controles en sus respectivas hojas de propiedades

Establecer *FlowLayout* como gestor de diseño de los paneles, de modo que queden centrados en el panel y suficientemente separados horizontalmente.

Establecer *BorderLayout* como gestor de diseño del applet, de modo que el panel superior quede al norte (NORTH) y el inferior al centro (CENTER) o al sur (SOUTH).

Crear una clase denominada *Hilo*, semejante a la del apartado anterior, sustituyendo la sentencia que imprime un texto en la consola por la sentencia que escribe en un control de edición. Naturalmente, dentro de la clase *Hilo* se tiene que tener acceso al control de edición pasándoselo en su constructor. Para observar las prioridades, se emplea un bucle **for** dentro de la función miembro *run* de 4000 o 5000 pasos, y se pone un tiempo de pausa en la función *sleep* pequeño de 1 ó 2.

Respuesta a las acciones del usuario

En la función respuesta a la acción de pulsar sobre el botón:

- Crear dos subprocesos mediante **new**
- Establecer la prioridad de cada subproceso llamando a *setPriority*
- Ponerlos en marcha llamando a la función *start*.

Comentarios

Puede entenderse fácilmente la clase *Hilo*, la única diferencia es que tiene un miembro dato que es el control de edición en el cual se va a mostrar el valor de la variable contador *i* del bucle **for**.

```
import java.awt.*;

public class Hilo extends Thread {
    private TextField contador;
    public Hilo(TextField contador, String nombre) {
        super(nombre);
        this.contador=contador;
    }
    public void run(){
        for(int i=1; i<5000; i++){
            contador.setText(String.valueOf(i));
            try{
                sleep(1);
            }catch(InterruptedException ex){}
        }
    }
}
```

La definición de la función respuesta a la acción de pulsar el botón titulado Empieza es la siguiente.

```
void btnEmpieza_actionPerformed(ActionEvent e) {
    Hilo hilo1=new Hilo(tTexto1, "Subproceso 1");
    Hilo hilo2=new Hilo(tTexto2, "Subproceso 2");
    hilo1.setPriority(Thread.MAX_PRIORITY);
    hilo2.setPriority(Thread.MIN_PRIORITY);
    hilo1.start();
    hilo2.start();
}
```

El Subproceso 1 tiene la máxima prioridad (está a la izquierda en el applet), y el Subproceso 2 tiene mínima prioridad (está a la derecha en el applet). Como vemos, el subproceso de más lata prioridad no anula al subproceso de más baja prioridad. El sistema escoge primero al de más alta prioridad, pero da alguna oportunidad al de más baja prioridad de ejecutarse.

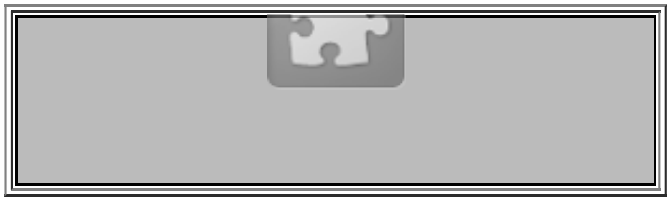
Se suguiere al lector cambiar las prioridades relativas de los subprocesos, y en cada uno de los casos observar el efecto del cambio en el tiempo de pausa, (argumento de la función *sleep*).

Control de la ejecución de los subprocesos

thread3: Hilo.java, ThreadApplet3.java

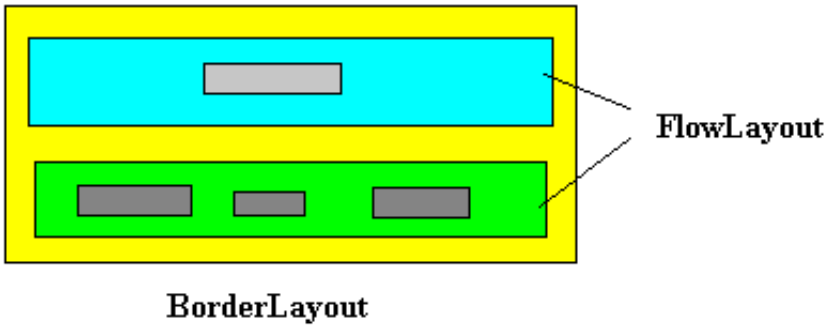
Vamos a crear un applet en el que corra un subproceso:

- El subproceso se pone en marcha pulsando en el botón titulado Empieza
- Se pone en estado de espera pulsando en el botón Pausa, se reanuda su ejecución pulsando en el mismo botón que ahora se titula Continua.
- Se para o mata el subproceso pulsando en el botón titulado Parar.



Diseño

Crear el applet, en modo de diseño (pestaña **Design**) situar dos paneles, uno en la parte superior y otro en la parte inferior. Sobre el panel superior poner un control de edición (*TextField*), sobre el panel inferior poner tres botones (*Button*).



Cambiar las propiedades de los controles en sus respectivas hojas de propiedades

Establecer *FlowLayout* como gestor de diseño de los paneles, de modo que queden centrados en el

panel y suficientemente separados horizontalmente.

Establecer *BorderLayout* como gestor de diseño del applet, de modo que el panel superior quede al norte (NORTH) y el inferior al centro (CENTER) o al sur (SOUTH).

Crear un clase *Hilo* semejante a la estudiada en el apartado anterior, sustituyendo el bucle **for** por un bucle sin fin, **while(true)**. La clase *Hilo* tendra como miembro dato la variable contador *i*, cuyo valor inicial cero se establece en el constructor

```
import java.awt.*;

public class Hilo extends Thread {
    private TextField contador;
    private int i=0;
    public Hilo(TextField contador, String nombre) {
        super(nombre);
        this.contador=contador;
        i=0;
    }
    public void run(){
        while(true){
            i++;
            contador.setText(String.valueOf(i));
            try{
                sleep(200);
            }catch(InterruptedException ex){}
        }
    }
}
```

Funciones respuesta

En modo diseño (pestaña **Design**), haciendo doble-clic sobre cada uno de los botones se genera el nombre de la función respuesta.

En la función respuesta a la acción de pulsar el botón titulado Empieza, se crea el subproceso y se pone en marcha, tal como se ha visto en el apartado anterior.

```
    void btnEmpezar_actionPerformed(ActionEvent e) {
        btnPausa.setEnabled(true);
        btnParar.setEnabled(true);
        btnPausa.setLabel("  Pausa  ");
        bPausa=true;
        if(hilo1==null){
            hilo1=new Hilo(tTexto1, "Subproceso 1");
            hilo1.start();
        }
    }
```

En la función respuesta a la acción de pulsar en el botón Pausa, se llama a dos funciones de la clase *Thread*: *suspend* y *resume*. La primera pone el subproceso en estado Not Runnable hasta que una llamada a *resume* lo pone de nuevo en estado Runnable. Según sea el valor **true** o **false** de un miembro dato *bPausa* de tipo **boolean** se cambia el título del botón y se llama a *suspend* o a *resume*, tal como se ve en la definición de la función respuesta.

```
    void btnPausa_actionPerformed(ActionEvent e) {
        if(bPausa==true){
            hilo1.suspend();
            btnPausa.setLabel("Continua");
            bPausa=false;
```

```
        }else{
            btnPausa.setLabel("  Pausa  ");
            hilo1.resume();
            bPausa=true;
        }
    }
```

En la definición de la función miembro *run*, de la clase *Hilo*, se ejecuta un bucle sin fin. Tiene que haber algún modo de terminar la ejecución del subproceso. Ya hemos visto que un subproceso alcanza el estado Death (muerte), cuando se llega al final de la función *run*. El otro modo, es llamar desde el subproceso a la función *stop* miembro de la clase *Thread*.

La definición de la función respuesta a la pulsación sobre el botón Parar es la siguiente

```
    void btnParar_actionPerformed(ActionEvent e) {
        hilo1.stop();
        hilo1=null;
    }
```

Una vez que el subproceso *hilo1* está en estado Death no se puede volver a llamar desde *hilo1* a *start* para ponerlo en marcha, hay que volver a crear un nuevo subproceso. Esta es la razón por la que se asigna **null** a *hilo1* una vez parado el subproceso mediante la función *stop*. En la función respuesta a la acción de pulsar en el botón titulado Empieza escribimos

```
        if(hilo1==null){
            hilo1=new Hilo(tTexto1, "Subproceso 1");
            hilo1.start();
        }
```