2.1. INTRODUCCION

En el capítulo anterior se estudió la programación concurrente y como se podían realizar programas concurrentes con el lenguaje Java. Se hizo una breve introducción al concepto de hilo y las diferencias entre estos y los procesos.

Recordemos que los hilos comparten el espacio de memoria del usuario, es decir, corren dentro del contexto de otro programa; y los procesos generalmente mantienen su propio espacio de direcciones y entorno de operaciones. Por ello a los hilos se les conoce a menudo como

procesos ligeros.

En este capítulo usaremos los hilos en Java para realizar programas concurrentes.

2.2. QUÉ SON LOS HILOS

Un **hilo** (hebra, thread en inglés) 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. Dentro de cada proceso hay varios hilos ejecutándose. La Figura 2.1 muestra la relación entre hilos y procesos.

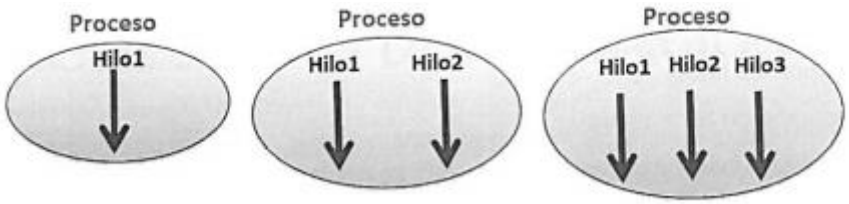

Figura 2.1. Relación entre hilos y procesos.

Podemos usar los hilos para diferentes aplicaciones: para realizar programas que tengan que realizar varias tareas simultáneamente, en los que la ejecución de una parte requiera tiempo y no deba detener el resto del programa. Por ejemplo, un programa que controla sensores en una fábrica, cada sensor puede ser un hilo independiente y recoge un tipo de información; y todos deben controlarse de forma simultánea. Un programa de impresión de documentos debe seguir funcionando, aunque se esté imprimiendo un documento, tarea que se puede llevar por medio de un hilo. Un programa procesador de textos puede tener un hilo comprobando la gramática del texto que estoy escribiendo y otro hilo guardando el texto en disco cada cierto tiempo. En un programa de bases de datos un hilo pinta la interfaz gráfica al usuario. En un servidor web, un hilo puede atender las peticiones entrantes y crear un hilo por cada cliente que tenga que servir.

2.3. CLASES PARA LA CREACION DE HILOS

En Java existen dos formas para crear hilos: extendiendo la clase Thread o implementando la interfaz Runnable. Ambas son parte del paquete java.lang.

2.3.1. La clase THREAD

La forma más simple de añadir funcionalidad de hilo a una clase es extender la clase Thread. O lo que es lo mismo crear una subclase de la clase Thread. Esta subclase debe sobrescribir el método run() con las acciones que el hilo debe desarrollar. La clase Thread define también los

métodos start() y stop() (actualmente en desuso) para iniciar y parar la ejecucion del hilo. La forma general de declarar un hilo extendiendo Thread es la siguiente:

```
class NombreHilo extends Thread {
//propiedades constructores y métodos de la clase
        public void run(){
          //acciones que lleva a cabo el hilo
       }
       }
```

Para crear un objeto hilo con el comportamiento de NombreHilo escribo:

```
NombreHilo h = new NombreHilo();
```

Y para iniciar su ejecución utilizamos el metodo start();

```
h.start();
```

El siguiente ejemplo declara la clase PrimerHilo que extiende la clase Thread, desde el constructor se inicializa una variable numérica que se usara para pintar un número de veces un mensaje; en el método run() se escribe la funcionalidad del hilo:

```
public class PrimerHilo extends Thread {
 private int x;
 PrimerHilo (int x)
 {
 this.x=x;
}

public void run(){
    for(int i=0; i<x; i++)
     System.out.println("En el Hilo... "+i);
   }
}//PrimerHilo
```

A continuación, para crear un objeto hilo escribimos:

```
PrimerHilo p= new PrimerHilo(10);
```

Y para iniciar su ejecución:

```
p.start();
```

Dentro de la clase anterior podemos añadir el método main() para crear el hilo e iniciar su ejecución:

```
public static void main(String[] args){
       PrimerHilo p = new PrimerHilo(10);
      p.start();
}// main
```

En el siguiente ejemplo se crea una clase que extiende Thread. Dentro de la clase se definen el constructor, el método run() con la funcionalidad que realizará el hilo y el método main() donde se crearan 3 hilos. La misión del hilo, descrita en el método run(), será visualizar un mensaje donde se muestre el nombre del hilo que se esta ejecutando y el contenido de un contador. Se utiliza una variable para mostrar el nombre del hilo que se ejecuta, esta variable se pasa al constructor y este se lo pasa al constructor de la clase base Thread mediante la palabra reservada super, para acceder a este nombre se usa el método getName(). Desde el método main() se crean los hilos y para iniciar cada hilo usamos el método start():

```
public class HiloEjemplo1 extends Thread{
//constructor
    public HiloEjemplo1(String nombre){
```

```
    super(nombre);
     System.out.println("CREANDO  HILO:" +  getName());
//método  run
public void run()
  for (int i=0;i<5;i++)
System.out.println("Hilo:"+  getName() +  " C=  "+  i);
}

//
public  static void main(Stringf]  args)
HiloEjemplol  hl =  new  HiloEjemplol("Hilo 1");
HiloEjemplol  h2 = new HiloEjemplol("Hilo 2");
HiloEjemplol  h3 = new  HiloEjemplol("Hilo 3");
hl.start();
h2.start();
h3.start();
System.out.println("3  HILOS  INICIADOS..."); }//  main

}// HiloEjemplo1
```

Es muy típico ver dentro del método run() un bucle infinito de forma que el hilo no termina nunca (más adelante veremos cómo detener el hilo). La ejecución del ejemplo anterior no siempre muestra la misma salida, en este caso se puede observar que los hilos no se ejecutan en el orden en que se crean:

```
CREANDO  HILO:Hilo          1
CREANDO   HILO:Hilo         2
CREANDO   HILO:Hilo         3
3  HILOS  INICIADOS
Hilo:  Hilo1  C       =     0
Hilo:  Hilo1  C       =     1
Hilo:  Hilo1  C       =     2
Hilo:  Hilo1  C       =     3
Hilo:  Hilo1  C       =     4
Hilo:  Hilo3  C       =     0
Hilo:  Hilo2  C       =     0
Hilo:  Hilo3  C       =     1
Hilo:  Hilo3  C       =     2
Hilo:  Hilo3  C       =     3
Hilo:  Hilo3  C       =     4
Hilo:  Hilo2  C       =     1
Hilo:  Hilo2  C       =     2
Hilo:  Hilo2  C       =     3
Hilo:  Hilo2  C       =     4
```

En este ejemplo se ha incluido el método main() dentro de la clase hilo. Podemos definir por un lado la clase hilo y por otro la clase que usa el hilo, tendríamos dos clases, la que extiende Thread, HiloEjemplo 1 _V2 java:

```
public class  HiloEjemplol V2  extends  Thread
   //constructor
   public HiloEjemplol_V2(String nombre){
    super(nombre);
     System.out.println("CREANDO  HILO:" +  getName());
}
//  metodo  run

public void run()    {
```

```
 for (int i=0; i<5; i++)
System.out.println("Hilo:"+ getName()          + " C - "          + i);
}
} //HiloEjemplo1_V2
```

Y la clase que usa el hilo:

```
public class UsaHiloEjemplo1 V2{
 public static void main(String[] args)          {
 HiloEjemplo1 h1 = new HiloEjemplo1("Hilo     1")       ;
 HiloEjemplo1 h2 = new HiloEjemplo1("Hilo     2")       ;
 HiloEjemplo1 h3 = new HiloEjemplo1("Hilo     3");
h1.start();
h2. start();
h3. start ();
System.out.println("3 HILOS INICIADOS...");
}
}//UsaHiloEjemplo1_V2
```

Se compila primero la clase hilo y después la que usa el hilo, se ejecuta la clase que usa el hilo:

```
D:\CAPIT2>javac HiloEjemplo1_V2.java
D:\CAPIT2>javac UsaHiloEjemplo1_V2.java
D:\CAPIT2>java UsaHiloEjemplo1 V2
```

En la siguiente tabla se muestran algunos métodos útiles sobre los hilos, algunos ya se han usado:

MÉTODOS	MISIÓN
int getPriority()	Devuelve la prioridad del hilo.
setPriority(int p)	Cambia la prioridad del hilo al valor entero p.
void interrupt()	Interrumpe la ejecución del hilo
boolean interrupted()	Comprueba si el hilo actual ha sido interrumpido.
Thread currentThread()	Devuelve una referencia al objeto hilo que se está ejecutando actualmente.
boolean isDaemon()	Comprueba si el hilo es un hilo Daemon. Los hilos daemon o demonio son hilos con prioridad baja que normalmente se ejecutan en segundo plano. Un ejemplo de hilo demonio que está ejecutándose continuamente es el recolector de basura (*garbage collector*).
setDaemon(boolean on)	Establece este hilo como hilo Daemon, asignando el valor *true*, o como hilo de usuario, pasando el valor *false*.
void stop()	Detiene el hilo. Este método está en desuso.
Thread currentThread()	Devuelve una referencia al objeto hilo actualmente en ejecución.
int activeCount()	Este método devuelve el número de hilos activos en el grupo de hilos del hilo actual.
Thread.State getState()	Devuelve el estado del hilo: NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED

MÉTODOS	MISIÓN
int getPriority()	Devuelve la prioridad del hilo.
setPriority(int p)	Cambia la prioridad del hilo al valor entero p.
void interrupt()	Interrumpe la ejecución del hilo
boolean interrupted()	Comprueba si el hilo actual ha sido interrumpido.
Thread currentThread()	Devuelve una referencia al objeto hilo que se está ejecutando actualmente.
boolean isDaemon()	Comprueba si el hilo es un hilo Daemon. Los hilos daemon o demonio son hilos con prioridad baja que normalmente se ejecutan en segundo plano. Un ejemplo de hilo demonio que está ejecutándose continuamente es el recolector de basura (*garbage collector*).
setDaemon(boolean on)	Establece este hilo como hilo Daemon, asignando el valor *true*, o como hilo de usuario, pasando el valor *false*.
void stop()	Detiene el hilo. Este método está en desuso.
Thread currentThread()	Devuelve una referencia al objeto hilo actualmente en ejecución.
int activeCount()	Este método devuelve el número de hilos activos en el grupo de hilos del hilo actual.
Thread.State getState()	Devuelve el estado del hilo: NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED

Todo hilo de ejecución en Java debe formar parte de un grupo. Por defecto, si no se especifica ningún grupo en el constructor, los hilos serán miembros del grupo main, que es creado por el sistema cuando arranca la aplicación Java.

La clase ThreadGroup se utiliza para manejar grupos de hilos en las aplicaciones Java. La clase Thread proporciona constructores en los que se puede especificar el grupo del hilo que se está creando en el mismo momento de instanciarlo. El siguiente ejemplo crea un grupo de hilos de nombre Grupo de hilos. A continuación, crea tres hilos usando el siguiente constructor de la clase Thread:

Thread (grupo ThreadGroup, destino Runnable, nombre String)

En el que se especifica el grupo de hilos, el objeto hilo y el nombre del hilo. El código es el siguiente:

```
public class HiloEjemplo2Grupos extends Thread        {
  public void run() {
  System.out.println("Informacion del hilo: "+
        Thread.currentThread().toString());
  for (int i= 0;i<1000; i++) i++;
    System.out.println(Thread.currentThread().getName()+"Finalizando la ejecucion.");

public static void main(Stringf] args) f
    Thread.currentThread().setName("Principal");
    System.out.println(Thread.currentThread().getName());
    System.out.println(Thread.currentThread().toString());

    ThreadGroup grupo = new ThreadGroup("Grupo de hilos");
    HiloEjemplo2Grupos h =new HiloEjemplo2Grupos();

   Thread h1 = new Thread(grupo, h, "Hilo 1");
```

```
Thread h2 = new Thread(grupo, h, "Hilo 2");
Thread h3 = new Thread(grupo, h, "Hilo 3");
hl.start();
h2.start();
h3.start();

System.out.println("3 HILOS CREADOS...");
System.out.println("Hilos activos: "
}
}// HiloEjemplo2Grupos
```

La ejecución muestra la siguiente salida:

```
Principal
Thread[Principa1,5,main]
3 HILOS CREADOS...
Hilos activos: 4
Información del hilo: Thread[Hilo 1,5,Grupo de hilos]
Información del hilo: Thread[Hilo 2,5,Grupo de hilos]
Hilo 1 Finalizando la ejecución.
Hilo 2 Finalizando la ejecución.
Información del hilo: Thread[Hilo [Hilo 3,5,Grupo de hilos]
Hilo 3 Finalizando la ejecución.
```

ACTIVIDAD 2.1

Crea dos clases (hilos) Java que extiendan la clase Thread. Uno de los hilos debe visualizar en pantalla en un bucle infinito la palabra TIC y el otro hilo la palabra TAC. Dentro del bucle utiliza el método sleep() para que nos de tiempo a ver las palabras que se visualizan cuando lo ejecutemos, tendrás que añadir un bloque try-catch (para capturar la excepción InterruptedException). Crea después la función main() que haga uso de los hilos anteriores. ¿Se visualizan los textos TIC y TAC de forma ordenada (es decir TIC TAC TIC TAC ...)?