Multithreading es una característica de Java que permite la ejecución concurrente de dos o más partes de un programa para una utilización máxima de la CPU. Cada parte de dicho programa se llama hilo. Entonces, los hilos son procesos livianos dentro de un proceso.

Lectura Recomendada

Introducción a Hilos

1. Creando un hilo

Usted crea un hilo instanciando un objeto de tipo Thread. La clase Thread encapsula un objeto que se puede ejecutar. Como se mencionó, Java define dos formas en las que puede crear un objeto ejecutable:

Los hilos se pueden crear utilizando dos mecanismos:

1. Extender la clase Thread
2. Implementar la interfaz Runnable

La mayoría de los ejemplos utilizarán el enfoque que implementa Runnable. Sin embargo, veremos ambas formas.

Recuerde: ambos enfoques aún usan la clase Thread para crear instancias, acceder y controlar el hilo. La única diferencia es cómo se crea una clase habilitada para hilos.

La interfaz **Runnable** abstrae una unidad de código ejecutable. Puede construir un hilo en cualquier objeto que implemente la interfaz Runnable. Runnable define solo un método llamado **run()**, que se declara así:

```
public void run()
```

Dentro de **run()**, se definirá el código que constituye el nuevo hilo. Es importante entender que *run()* puede llamar a otros métodos, usar otras clases y declarar variables como el hilo principal. La única diferencia es que *run()* establece el punto de entrada para otro hilo de ejecución concurrente dentro de su programa. Este hilo terminará cuando retorne *run()*.

Después de crear una clase que implemente **Runnable**, instanciará un objeto del tipo Thread en un objeto de esa clase. El hilo define varios constructores. El que usaremos primero se muestra aquí:

```
Thread(Runnable threadOb)
```

En este constructor, *threadOb* es una instancia de una clase que implementa la interfaz *Runnable*. Esto define dónde comenzará la ejecución del hilo.

Una vez creado, el nuevo hilo no comenzará a ejecutarse hasta que llame a su método **start()**, que se declara dentro de Thread. En esencia, *start()* ejecuta una llamada a *run()*. El método *start()* se muestra aquí:

```
void start()
```

2. Ejemplo de hilo mediante la implementación de la interfaz Runnable

Aquí hay un ejemplo que crea un nuevo hilo y lo ejecuta:

```
//Crea un hilo implementando Runnable.

//Los objetos de MiHilo se pueden ejecutar en sus propios hilos
// porque MiHilo implementa Runnable.
 class MiHilo implements Runnable {
    String nombreHilo;

    MiHilo(String nombre){
        nombreHilo=nombre;
    }
    //Punto de entrada del hilo
    //Los hilos comienzan a ejecutarse aquí
    public void run(){
        System.out.println("Comenzando "+nombreHilo);
        try {
            for (int contar=0; contar<10; contar++){
                Thread.sleep(400);
                System.out.println("En "+nombreHilo+", el recuento "+contar);
            }
        }catch (InterruptedException exc){
            System.out.println(nombreHilo + "Interrumpido.");
        }
        System.out.println("Terminando "+nombreHilo);
    }
}

class UsoHilos{
    public static void main(String[] args) {
        System.out.println("Hilo principal iniciando.");

        //Primero, construye un objeto MiHilo.
        MiHilo mh=new MiHilo("#1");

        //Luego, construye un hilo de ese objeto.
        Thread nuevoh=new Thread(mh);

        //Finalmente, comienza la ejecución del hilo.
        nuevoh.start();

        for (int i=0; i<50;i++){
            System.out.print(" .");
        }try{
            Thread.sleep(100);
        }catch (InterruptedException exc){
            System.out.println("Hilo principal interrumpido.");
        }
        System.out.println("Hilo principal finalizado.");
    }
}
```

Miremos de cerca este programa. Primero, *MiHilo* implementa *Runnable*. Esto significa que un objeto de tipo *MiHilo* es adecuado para usar como un hilo y se puede pasar al constructor de Thread.

2.1. Explicación de hilo en Java (I)

Dentro de *run()*, se establece un bucle que cuenta de 0 a 9. Observe la llamada a *sleep()*. El método *sleep()* hace que el hilo del que se llama suspenda la ejecución durante el período especificado de milisegundos. Su forma general se muestra aquí:

```
static void sleep(long milisegundos) throws InterruptedException
```

La cantidad de milisegundos para suspender se especifica en *milisegundos*. Este método puede lanzar una **InterruptedException**. Por lo tanto, las llamadas a ella deben estar envueltas en un bloque **try**.

El método **sleep()** también tiene una segunda forma, que le permite especificar el período en términos de milisegundos y nanosegundos si necesita ese nivel de precisión. En run(), sleep() pausa el hilo durante 400 milisegundos cada vez a través del bucle. Esto permite que el hilo se ejecute con la lentitud suficiente para que pueda verlo ejecutar.

Dentro de main(), se crea un nuevo objeto Thread mediante la siguiente secuencia de instrucciones:

```
//Primero, construye un objeto MiHilo.
 MiHilo mh=new MiHilo("#1");
//Luego, construye un hilo de ese objeto.
 Thread nuevoh=new Thread(mh);
//Finalmente, comienza la ejecución del hilo.
 nuevoh.start();
```

Como sugieren los comentarios, primero se crea un objeto de *MiHilo*. Este objeto luego se usa para construir un objeto *Thread*. Esto es posible porque *MiHilo* implementa Runnable. Finalmente, la ejecución del nuevo hilo se inicia llamando a **start()**. Esto hace que comience el método run() del hilo hijo.

Después de llamar a *start()*, la ejecución vuelve a *main()*, y entra *main()* para el ciclo. Observe que este ciclo itera 50 veces, pausando 100 milisegundos cada vez a través del ciclo. Ambos hilos continúan ejecutándose, compartiendo la CPU en sistemas de una sola CPU, hasta que terminan sus bucles. El resultado producido por este programa es el siguiente. Debido a las diferencias entre los entornos informáticos, el resultado preciso que ve puede diferir ligeramente del que se muestra aquí:

```
Hilo principal iniciando.
.Comenzando #1
.....En #1, el recuento 0
....En #1, el recuento 1
....En #1, el recuento 2
....En #1, el recuento 3
....En #1, el recuento 4
....En #1, el recuento 5
....En #1, el recuento 6
....En #1, el recuento 7
....En #1, el recuento 8
....En #1, el recuento 9
Terminando #1
........Hilo principal finalizado.
```

2.2. Explicación de hilo en Java (II)

Hay otro punto de interés para notar en este primer ejemplo de hilos. Para ilustrar el hecho de que el hilo *main* y *mh* se ejecutan simultáneamente, es necesario evitar que *main()* termine hasta que termine *mh*.

Aquí, esto se hace a través de las diferencias de tiempo entre los dos hilos. Porque las llamadas a **sleep()** dentro del bucle **for** de **main()** causan un retraso total de 5 segundos (50 iteraciones por 100 milisegundos), pero el retardo total dentro del bucle **run()** es de solo 4 segundos (10 iteraciones por 400 milisegundos), **run()** finalizará aproximadamente 1 segundo antes de *main()*. Como resultado, tanto el hilo *main* como *mh* se ejecutarán simultáneamente hasta que termine *mh*. Luego, aproximadamente 1 segundo más tarde, *main()* finaliza.

Aunque este uso de las diferencias de tiempo para asegurar que *main()* termina al final es suficiente para este simple ejemplo, no es algo que normalmente se usaría en la práctica. Java proporciona formas mucho mejores de esperar a que termine un hilo. Sin embargo, es suficiente para los próximos programas.

Otro punto: en un programa multihilo, a menudo querrás que **el hilo principal sea el último hilo que termine ejecutando**. Como regla general, un programa continúa ejecutándose hasta que todos sus hilos hayan finalizado. Por lo tanto, no es obligatorio finalizar el hilo principal al final. Sin embargo, a menudo es una buena práctica seguirla, especialmente cuando recién está aprendiendo sobre los hilos.

3. Ejemplo de hilo al extender la clase Thread

La implementación de Runnable es una forma de crear una clase que pueda instanciar objetos hilos. Extender de Thread es la otra. En este ejemplo, verá cómo extender Thread creando un programa funcionalmente similar al programa UsoHilos que se mostró anteriormente.

Cuando una clase extiende de **Thread**, debe anular el método **run()**, que es el punto de entrada para el nuevo hilo. También debe llamar a **start()** para comenzar la ejecución del nuevo hilo. Es posible anular otros métodos Thread, pero no es necesario.

- Crea un archivo llamado ExtendThread.java. Comience este archivo con las siguientes líneas:

```
class MiHilo extends Thread{
   //Construye un nuevo hilo.
    MiHilo(String nombre){
    //super se usa para llamar a la versión del constructor de Thread
        super(nombre);
    }
    //Punto de entrada del hilo
    public void run(){
        System.out.println(getName()+" iniciando.");
    //Como ExtendThread extiende de Thread, puede llamar directamente
    //a todos los métodos de Thread, incluido el método getName().
    try {
        for (int cont=0;cont<10;cont++){
            Thread.sleep(400);
            System.out.println("En "+getName()+ ", el recuento es "+cont);
        }
    }catch (InterruptedException exc){
        System.out.println(getName()+ "interrumpido.");
```

```
    }
        System.out.println(getName()+ "finalizando.");
    }
}

class ExtendThread{
     public static void main(String[] args) {
         System.out.println("Iniciando hilo principal.");

         MiHilo mh=new MiHilo("#1");

         mh.start();

         for (int i=0;i<50;i++){
             System.out.print(".");
         try {
             Thread.sleep(100);
         }catch (InterruptedException exc) {
             System.out.println("Hilo principal interrumpido");
         }
         }
         System.out.println("Hilo principal finalizado");
     }
 }
```

4. Crear múltiples hilos

Los ejemplos anteriores han creado solo un hilo hijo. Sin embargo, su programa puede engendrar tantos hilos como necesite. Por ejemplo, el siguiente programa crea tres hilos hijos:

```
class MiHilo implements Runnable{
    Thread hilo;

    //Construye un nuevo hilo.
     MiHilo(String nombre){
         hilo= new Thread(this,nombre);
     }

     //Un método de fábrica que crea e inicia un hilo.

     public static MiHilo crearYComenzar (String nombre){
         MiHilo miHilo=new MiHilo(nombre);
         miHilo.hilo.start(); //Inicia el hilo
         return miHilo;
     }

     //Punto de entrada de hilo.
     public void run(){
         System.out.println(hilo.getName()+" iniciando.");
         try {
             for (int count=0; count<10;count++){
                 Thread.sleep(400);
                 System.out.println("En "+hilo.getName()+ ", el recuento es
"+count);
             }
         }catch (InterruptedException exc){
             System.out.println(hilo.getName()+ " interrumpudo.");
         }
         System.out.println(hilo.getName()+" terminado.");
     }
 }
```

```
class MasHilos {
     public static void main(String[] args) {
         System.out.println("Hilo principal iniciando.");

         MiHilo miHilo1 = MiHilo.crearYComenzar("#1");
         MiHilo miHilo2 = MiHilo.crearYComenzar("#2");
         MiHilo miHilo3 = MiHilo.crearYComenzar("#3");

         for (int i = 0; i < 50; i++) {
             System.out.print(".");
             try {
                 Thread.sleep(100);
             } catch (InterruptedException exc) {
                 System.out.println("Hilo principal interrumpido.");
             }
         }
         System.out.println("Hilo principal finalizado");
     }
 }
```

Salida:

```
Hilo principal iniciando.
.#2 iniciando.
#1 iniciando.
#3 iniciando.
....En #1, el recuento es 0
En #2, el recuento es 0
...
En #3, el recuento es 9
#3 terminado.
.........Hilo principal finalizado
```

Como puede ver, una vez iniciados, los tres hilos hijos comparten la CPU. Tenga en cuenta que en esta ejecución **los hilos se inician en el orden en que se crean**. Sin embargo, esto puede no ser siempre el caso.

Java es libre de programar la ejecución de los hilos a su manera. Por supuesto, debido a las diferencias en el tiempo o el entorno, el resultado preciso del programa puede variar, por lo que no se sorprenda si ve resultados ligeramente diferentes cuando prueba el programa.