Threads en JAVA

Nociones básicas

Programación 3

TUDAI 2024

Facultad de Cs. Exactas

Threads

Un thread (o hilo de ejecución), en sistemas operativos, es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente).

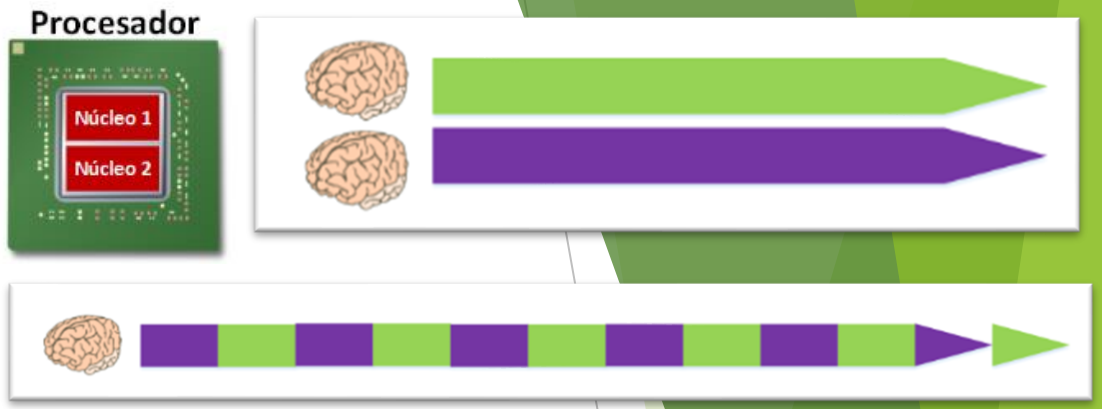
Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, etc.

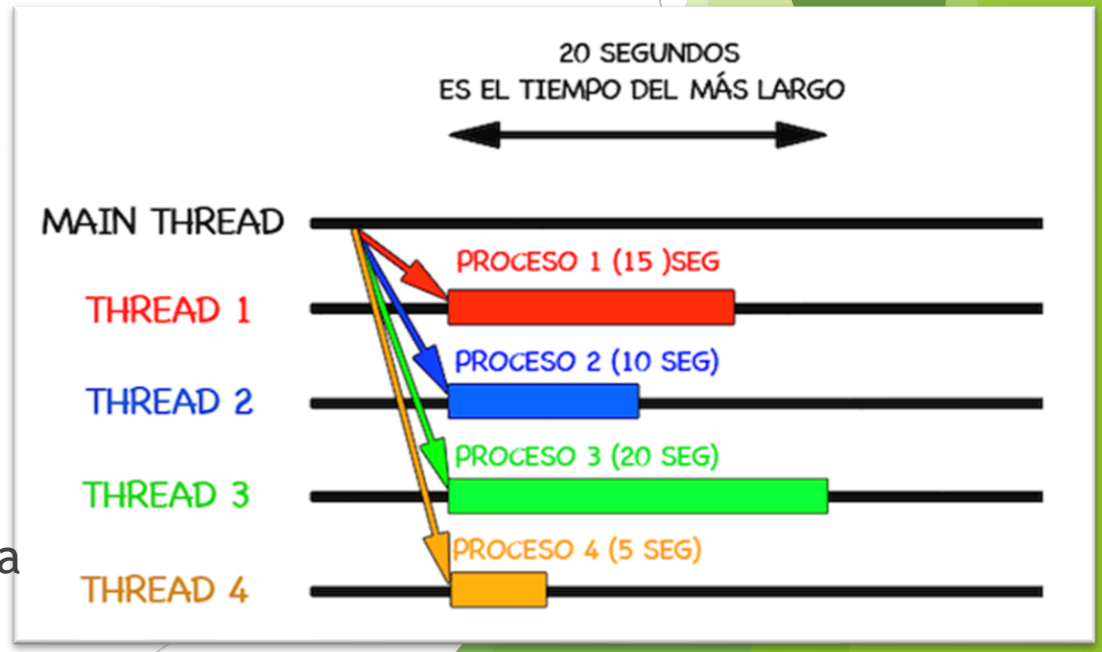
Cada hilo de ejecución tiene su propia pila de ejecución.

El uso de threads permite el diseño de aplicaciones que deben llevar a cabo distintas funciones simultáneamente. Por ej.: un browser de internet, mientras se descarga un archivo, muestra contenido de páginas, o ejecuta un video, etc.

Threads

- Entonces, un thread es básicamente una tarea que puede ser ejecutada concurrentemente (o en paralelo) con otra tarea dentro de un mismo programa.
- Los threads no pueden ejecutarse ellos solos; requieren la supervisión de un proceso padre.
- Un proceso es un programa (o aplicación) en ejecución.
- Dentro de cada proceso hay varios hilos ejecutándose.
  Por ejemplo, Word puede tener un hilo en background chequeando automáticamente la gramática de lo que estoy escribiendo, mientras otro hilo puede estar guardando automáticamente los cambios del documento en el que estoy trabajando.
- Como Word, cada aplicación (proceso) puede correr varios hilos los cuales están realizando diferentes tareas.
  Esto significa que los hilos están siempre asociados con un proceso en particular.
- Cuando se pueden ejecutar varias tareas a la vez se denomina sistema Multitarea.





Threads en JAVA

La JVM soporta y administra threads.

- En Java para utilizar la multitarea tenemos dos opciones:

1. Usar la clase ***Thread*** (es decir que la clase que implementemos debe heredar de la clase ***Thread***). La clase ***Thread*** implementa la Interface ***Runnable***.

   Se debe redefinir el método **run**(). Este método es invocado cuando se inicia el hilo (mediante una llamada al método **start**() de la clase Thread). El hilo inicia su ejecución en el método run() y termina cuando éste termina.

   El método main crea dos objetos de clase ThreadEjemplo y los inicia con la llamada al método start() (el cual inicia el nuevo hilo y llama al método run()).

```
public class ThreadEjemplo extends Thread {
    public ThreadEjemplo(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10 ; i++)
            System.out.println(i + " " + getName());
        System.out.println("Termina thread " + getName());
    }
    public static void main (String [] args) {
        new ThreadEjemplo("Pepe").start();
        new ThreadEjemplo("Juan").start();
        System.out.println("Termina thread main");
    }
}
```

   Realizada la llamada al método **start**(), éste le devuelve control y continua su ejecución, independiente de los otros hilos.

Threads en JAVA

- Segunda Opción:

2. Para los casos en los que no es posible hacer que la clase definida extienda (herede) la clase Thread.

   Esto ocurre cuando en la clase que estamos definiendo (de la que se quiere ejecutar en un hilo independiente) deba extender alguna otra clase.

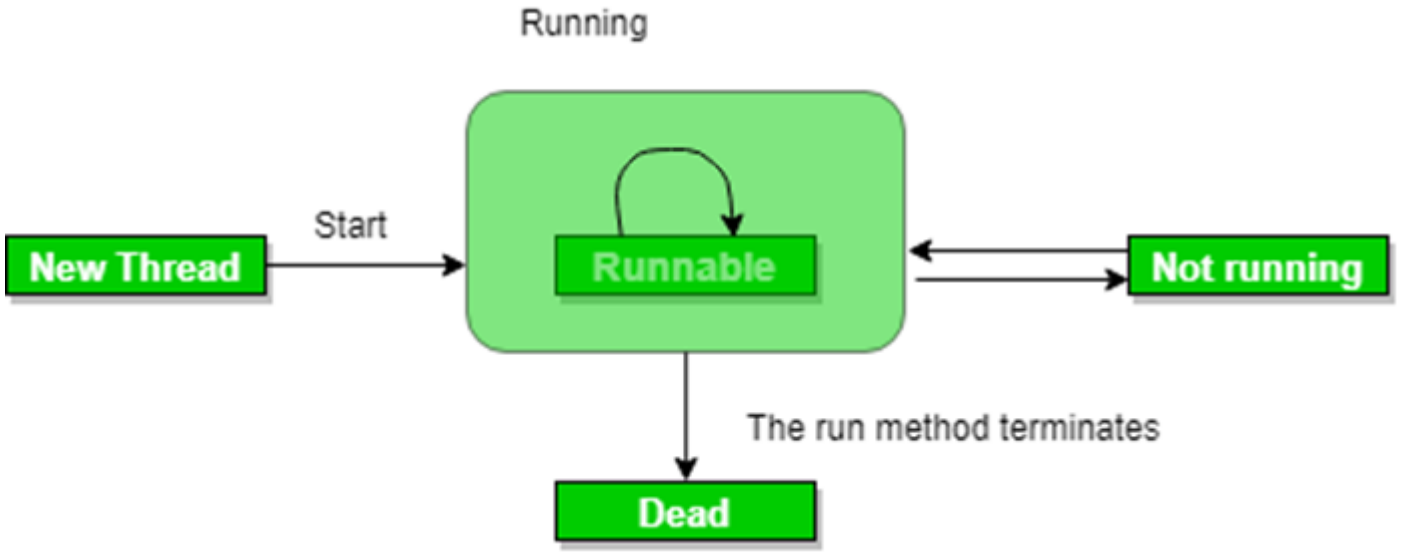
   Dado que no existe herencia múltiple en Java, la citada clase no puede extender a la vez la clase Thread y otra más. Entonces se utiliza la opción de implementar la Interface Runnable.

   El siguiente ejemplo es equivalente al anterior, pero utilizando la interface Runnable:

```
public class ThreadEjemplo implements Runnable {
    public void run() {
        for (int i = 0; i < 5 ; i++)
            System.out.println(i + " " +
                               Thread.currentThread().getName());
        System.out.println("Termina thread " +
                           Thread.currentThread().getName());
    }
    public static void main (String [] args) {
        new Thread (new ThreadEjemplo(), "Pepe").start();
        new Thread (new ThreadEjemplo(), "Juan").start();
        System.out.println("Termina thread main");
    }
}
```

Threads en JAVA

- Estados posibles de un Thread



- Cualquier aplicación en ejecución (proceso) en JAVA tiene al menos un thread llamado ***main thread***, y justamente comienza su ejecución en el método main.

Threads en JAVA

- Ejemplo de cuentas bancarias:
- Supongamos en un banco se tienen 100 cuentas bancarias.
- Cada cuenta inicia con $2000.
- Entonces el monto total de las cuentas es $200.000.
- Se realizan sin parar operaciones de transferencia entre cuentas. De manera que lo que se saca de una cuenta se transfiere a otra. Dado que las operaciones se hacen a cuentas dentro del mismo banco, la suma total $200.000 permanece constante.
- Vamos a similar esto mediante Threads que realicen aleatoriamente las operaciones de extracción.

- PROBLEMA: se requiere un acceso sincronizado al método que modifica las cuentas bancarias.

Threads en JAVA

- Ejemplo Productor/Consumidor:
  - El productor genera un flujo de datos que son recogidos por el consumidor.

  - PROBLEMA: Si no se sincronizan entre ambos puede pasar que el producto este listo para producir algo nuevo y el consumidor no haya procesado lo anterior. O puede pasar que el consumidor intente consumir cuando el productor aun no produjo.

Intro. Sincronización de Threads en JAVA

- Como vimos en muchas ocasiones será necesario sincronizar la ejecución de los threads, se da cuando varios threads acceden a un mismo recurso compartido.

- En los ejemplos vimos la utilización de la instrucción SYNCRONIZED, la cual puede utilizarse en conjunción con WAIT y NOTIFY.

- Existen otras instrucciones y técnicas para sincronizar Threads.

- Java agrega una API de mayor nivel para el manejo de threads de forma más transparente mediante el package java.util.concurrent y la definición de Executors. Aquí se evita al programador tener que crear explícitamente los threads por ejemplo.

- Muy recomendado si se quiere profundizar en el tema ir a la siguiente referencia (JDK 8!):

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html