Linkia FP

Formación Profesional Oficial a Distancia

Linkia FP

DAM – M09 – Clase 02

Programación de servicios y procesos.

UF2 – Procesos y Hilos

Introducción a los Hilos.

M09: Programación de Servicios y Procesos.

99 horas en total.

UF2: Procesos y Hilos (37 horas)
UF3: Sockets y Servicios (37 horas)
UF1: Seguridad y Criptografía (25 horas)

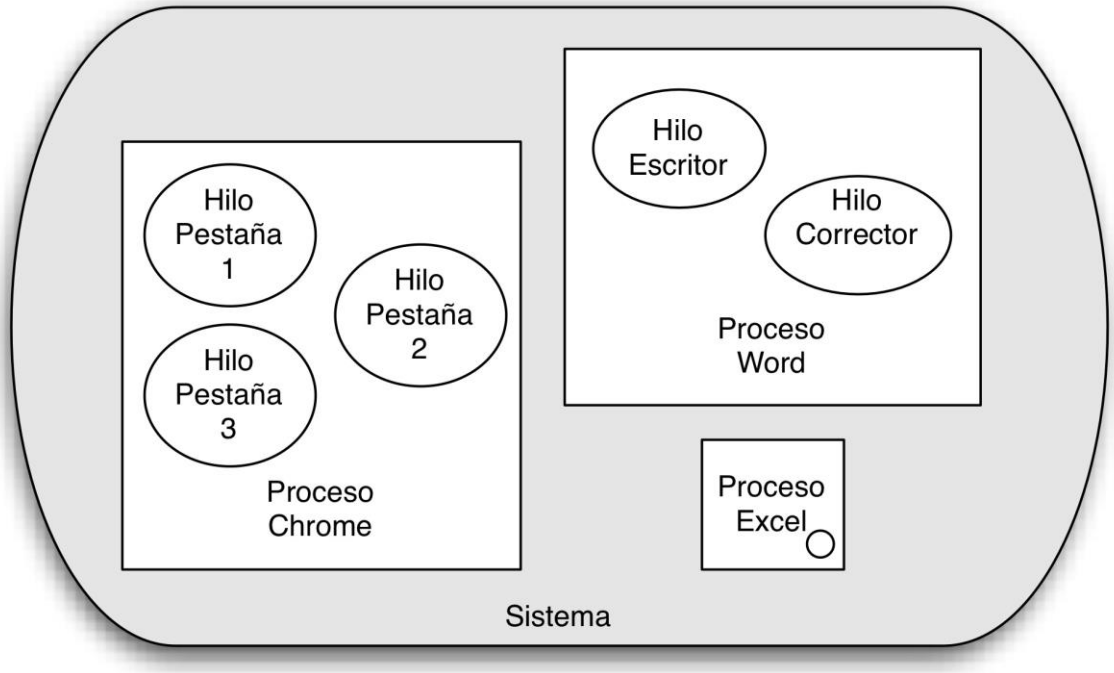
4 Actividades – 3 Tests – 10 Clases Virtuales.

Hilo

- Hilo (thread):
  - Unidad básica de utilización de la CPU, y más concretamente de un *core* del procesador.
  - Secuencia de código que está en ejecución, pero dentro del contexto de un proceso.

- Diferencia con procesos:
  - Los hilos se ejecutan dentro del contexto de un proceso. Dependen de un proceso para ejecutarse.
  - Los procesos son independientes y tienen espacios de memoria diferentes.
  - Dentro de un mismo proceso pueden coexistir varios hilos ejecutándose que compartirán la memoria de dicho proceso.

Hilo

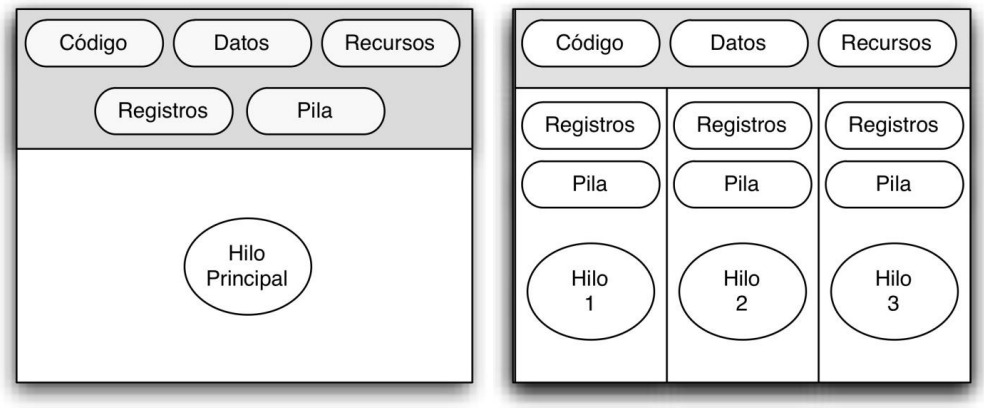

Hilo
Pestaña
1
Hilo
Pestaña
2
Hilo
Pestaña
3
Proceso
Chrome
Hilo
Escritor
Hilo
Corrector
Proceso
Word
Proceso
Excel
Sistema

Multitarea

- Multitarea: ejecución simultanea de varios hilos:
  - **Capacidad de respuesta**. Los hilos permiten a los procesos continuar atendiendo peticiones del usuario aunque alguna de las tareas (hilo) que esté realizando el programa sea muy larga.
  - **Compartición de recursos.** Por defecto, los *threads* comparten la memoria y todos los recursos del proceso al que pertenecen.
  - La creación de nuevos hilos no supone ninguna reserva adicional de memoria por parte del sistema operativo.
  - **Paralelismo real.** La utilización de *threads* permite aprovechar la existencia de más de un núcleo en el sistema en arquitecturas *multicore*.

Recursos compartidos por Hilos

- Los procesos mantienen su propio espacio de direcciones y recursos de ejecución mientras que los hilos dependen del proceso.
  - Comparten con otros hilos la sección de código, datos y otros recursos.
  - Cada hilo tiene su propio contador de programa, conjunto de registros de la CPU y pila para indicar por dónde se está ejecutando.



Proceso con un único thread

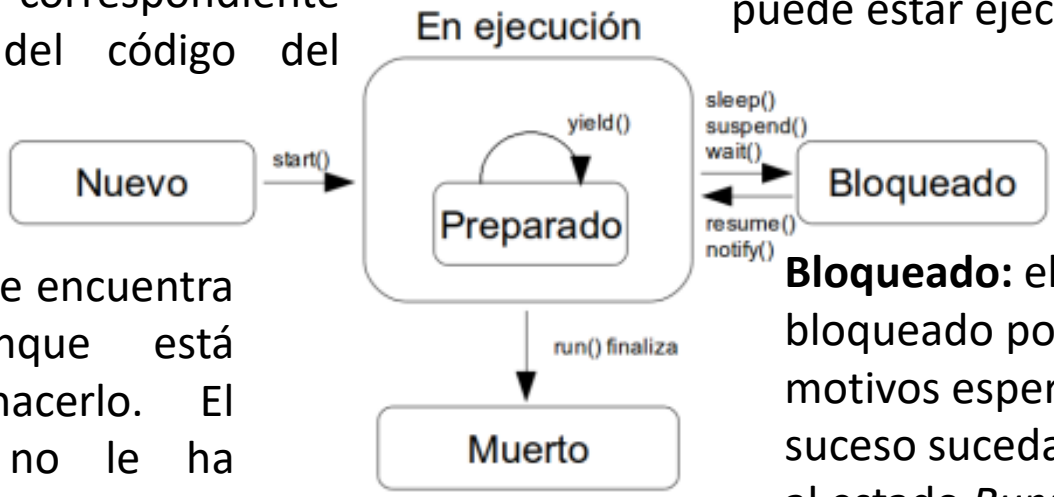
Proceso con varios threads

Estados de un hilo

- Los hilos pueden cambiar de estado a lo largo de su ejecución.
- Se definen los siguientes estados:

Nuevo: el hilo está preparado para su ejecución pero todavía no se ha realizado la llamada correspondiente en la ejecución del código del programa.

***Runnable*:** el hilo está preparado para ejecutarse y puede estar ejecutándose.



Listo: el proceso no se encuentra en ejecución aunque está preparado para hacerlo. El sistema operativo no le ha asignado todavía un procesador para ejecutarse.

Bloqueado: el hilo está bloqueado por diversos motivos esperando que el suceso suceda para volver al estado *Runnable*.

Terminado: el hilo ha finalizado su ejecución.

Gestión de Hilos

- Operaciones básicas:
  - Creación y arranque de hilos. Cualquier programa a ejecutarse es un proceso que tiene un hilo de ejecución principal. Este hilo puede a su vez crear nuevos hilos que ejecutarán código diferente o tareas, es decir el camino de ejecución no tiene por qué ser el mismo.
  - Espera de hilos. Como varios hilos comparten el mismo procesador, si alguno no tiene trabajo que hacer, es bueno suspender su ejecución para que haya un mayor tiempo de procesador disponible.

Creación y Arranque de hilos

Creación de hilos en Java:

- **Implementando la interfaz *Runnable*.** La interfaz *Runnable* proporciona la capacidad de añadir la funcionalidad de un hilo a una clase simplemente implementando dicha interfaz. La interfaz *Runnable* debería ser utilizada si la clase solamente va a utilizar la funcionalidad *run* de los hilos.
- **Extendiendo de la clase *Thread*** mediante la creación de una subclase. La clase *Thread* es responsable de producir hilos funcionales para otras clases e implementa la interfaz *Runnable*

El método *run()* implementa la operación *create* conteniendo el código a ejecutar por el hilo. Dicho método contendrá el hilo de ejecución.

La clase *Thread* define también el método *start()* para implementar la operación *create*. Este método es que comienza la ejecución del hilo de la clase correspondiente.

Creación y Arranque de hilos

Linkia FP

Implementando la interfaz *Runnable*:

```
1  class RunThread implements Runnable {
2
3      Thread t;
4      RunThread () {
5          t = new Thread(this, "Nuevo Thread");
6          System.out.println("Creado hilo: " + t);
7          t.start(); // Arranca el nuevo hilo de ejecución. Ejecuta run
8      }
9
10     public void run() {
11         System.out.println("Hola desde el hilo creado!");
12         System.out.println("Hilo finalizando.");
13     }
14 }
15
16 public class HelloRunnable {
17     public static void main(String args[]) {
18         new RunThread(); // Crea un nuevo hilo de ejecución
19         System.out.println("Hola desde el hilo principal!");
20         System.out.println("Proceso acabando.");
21     }
22 }
23
```

```
Creado hilo: Thread[Nuevo Thread,5,main]
Hola desde el hilo principal!
Proceso acabando.
Hola desde el hilo creado!
Hilo finalizando.
```

Creación y Arranque de hilos

Linkia FP

Extendiendo la clase *Thread*

```
1  class HelloHilo extends Thread {
2
3      public void run() {
4          System.out.println("Hola desde el hilo creado!");
5      }
6  }
7
8  public class HelloThread {
9      public static void main(String args[]) {
10
11     HelloHilo h1= new HelloHilo();// Crea y arranca un nuevo hilo de ejecución
12     h1.start();
13     System.out.println("Hola desde el hilo principal!");
14     System.out.println("Proceso acabando.");
15     }
16 }
```

```
Hola desde el hilo principal!
Proceso acabando.
Hola desde el hilo creado!
```

Creación y Arranque de hilos

Linkia FP

De las dos alternativas, ¿cuál utilizar? Depende de la necesidad:

La utilización de la interfaz *Runnable* es más general, ya que el objeto puede ser una subclase de una clase distinta de *Thread*

La utilización de la interfaz *Runnable* no tiene ninguna otra funcionalidad además de *run*() que la incluida por el programador.

La extensión de la clase *Thread* es más fácil de utilizar, ya que está definida una serie de métodos útiles para la administración de hilos,

La extensión de la clase *Thread* está limitada porque los clases creadas como hilos deben ser descendientes únicamente de dicha clase.

Ejecutar dos Hilos.

```
1
2  public class Repeticion extends Thread {
3
4       private int repeticiones;
5       private String mensaje;
6
7       Repeticion (String msg, int n) {
8        mensaje = msg;
9        repeticiones = n;
10       }
11
12     public void run () {
13
14       for (int i = 1; i <= repeticiones; i++)
15       System.out.println (mensaje + " " + i);
16       }
17
18
19     public static void main (String args[]) {
20
21       Repeticion r1=new Repeticion("Rojo", 5);
22       Repeticion r2=new Repeticion("Azul", 10);
23       r1.start();
24       r2.start ();
25       }
26     }
```

```
Azul 1
Rojo 1
Azul 2
Rojo 2
Azul 3
Rojo 3
Azul 4
Rojo 4
Azul 5
Azul 6
Azul 7
Azul 8
Azul 9
Azul 10
Rojo 5
```

```
Rojo 1
Rojo 2
Azul 1
Azul 2
Azul 3
Azul 4
Azul 5
Azul 6
Azul 7
Azul 8
Azul 9
Azul 10
Rojo 3
Rojo 4
Rojo 5
```

```
Azul 1
Azul 2
Azul 3
Rojo 1
Azul 4
Rojo 2
Azul 5
Azul 6
Azul 7
Azul 8
Azul 9
Azul 10
Rojo 3
Rojo 4
Rojo 5
```

Espera de hilos

Operaciones.

- **Join:** Si un Thread necesita esperar a que otro termine (por ejemplo el Thread padre espera a que termine el hijo) puede usar el método join(). Este método detiene el hilo actual hasta que termine el hilo sobre el que se llama join().
- **Sleep:** duerme un hilo por un período especificado

Ambas operaciones de espera pueden ser interrumpidas, si otro hilo interrumpe al hilo actual mientras está suspendido por dichas llamadas.

- Una interrupción es una indicación a un hilo que debería dejar de hacer lo que esté haciendo para hacer otra cosa.
- Un hilo envía una interrupción mediante la invocación del método *interrupt()* en el objeto del hilo que se quiere interrumpir.

***isAlive*():** comprueba si el hilo no ha finalizado su ejecución antes de trabajar con él.

Ejemplo Join

Linkia FP

```java
public class EjemploJoin extends Thread {

     private String mensaje;

     EjemploJoin (String msg) {
      mensaje = msg;
       }

    public void run () {

     for (int i = 1; i <=5; i++)
     System.out.println (mensaje + " " + i);
     System.out.println ("Fin Hilo " + mensaje);
     }

    public static void main (String args[]) {

     System.out.println ("Inicio programa principal");
     EjemploJoin h1=new EjemploJoin("Alumno Joan");
     EjemploJoin h2=new EjemploJoin("Alumno Ana");
     System.out.println ("Inicio Alumno Joan");
     h1.start();
     System.out.println ("Inicio Alumno Ana");
     h2.start();
     try {
        h1.join();
        h2.join();
    }catch( InterruptedException e ){}
     System.out.println ("Fin programa principal");
     }
}
```

Resultado con join

```
Inicio programa principal
Inicio Alumno Joan
Inicio Alumno Ana
Alumno Ana 1
Alumno Ana 2
Alumno Ana 3
Alumno Ana 4
Alumno Ana 5
Fin Hilo Alumno Ana
Alumno Joan 1
Alumno Joan 2
Alumno Joan 3
Alumno Joan 4
Alumno Joan 5
Fin Hilo Alumno Joan
Fin programa principal
```

```
Inicio programa principal
Inicio Alumno Joan
Inicio Alumno Ana
Fin programa principal
Alumno Joan 1
Alumno Joan 2
Alumno Joan 3
Alumno Joan 4
Alumno Joan 5
Fin Hilo Alumno Joan
Alumno Ana 1
Alumno Ana 2
Alumno Ana 3
Alumno Ana 4
Alumno Ana 5
Fin Hilo Alumno Ana
```

Resultado sin join

Ejemplo Sleep

Linkia FP

```java
public class TestThread {
    public static void main(String args[]) {
    RunnableDemo r1 = new RunnableDemo("Hilo 1");
    r1.start();
    RunnableDemo r2 = new RunnableDemo("Hilo 2");
    r2.start();
    }
}
```

```java
class RunnableDemo implements Runnable {
    private Thread t;
    private String threadName;

    //constructor->inicialización
    RunnableDemo( String name){
        threadName = name;
        System.out.println("Creando " + threadName );
    }
    //código que ejecutan los hilos
    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // para dormir el thread
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrumpido.");
        }
        System.out.println("Thread " + threadName + " finalizado.");
    }

    public void start (){
        System.out.println("Iniciando " + threadName );
        if (t == null)
        {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
}
```

```
Creando Hilo 1
Iniciando Hilo 1
Creando Hilo 2
Iniciando Hilo 2
Running Hilo 1
Thread: Hilo 1, 4
Running Hilo 2
Thread: Hilo 2, 4
Thread: Hilo 1, 3
Thread: Hilo 2, 3
Thread: Hilo 2, 2
Thread: Hilo 1, 2
Thread: Hilo 1, 1
Thread: Hilo 2, 1
Thread Hilo 2 finalizado.
Thread Hilo 1 finalizado.
```

Prioridades

setPriority(int)

- El método *setPriority()* asigna al hilo la prioridad indicada por el valor pasado como parámetro. Hay bastantes constantes predefinidas para la prioridad, definidas en la clase **Thread**, tales como MIN_PRIORITY, NORM_PRIORITY y MAX_PRIORITY, que toman los valores 1, 5 y 10, respectivamente.
- Como guía aproximada de utilización, se puede establecer que la mayor parte de los procesos a nivel de usuario deberían tomar una prioridad en torno a NORM_PRIORITY. Las tareas en segundo plano, como una entrada/salida a red o el nuevo dibujo de la pantalla, deberían tener una prioridad cercana a MIN_PRIORITY.
- Con las tareas a las que se fije la máxima prioridad, en torno a MAX_PRIORITY, hay que ser especialmente cuidadosos, porque si no se hacen llamadas a *sleep()* o *yield()*, se puede provocar que el intérprete Java quede totalmente fuera de control.

getPriority()

- Este método devuelve la prioridad del hilo de ejecución en curso, que es un valor comprendido entre uno y diez.

Ejemplo prioridades

Linkia FP

```
1  public class Serie extends Thread
2  {
3
4   private double sup;
5   private double inf;
6   private double i;
7   private double suma=0;
8
9    Serie(double a,double b){
10       inf=a;
11       sup=b;
12       System.out.println(inf);
13       System.out.println(sup);
14    }
15
16 public void run(){
17      for(i=inf;i<=sup;i++){
18          suma=suma+i;
19          System.out.println("Hilo--> "+this.getName()+ " i = " + i + " Suma= " + suma);
20          }
21      }
22 public static void main(String args[]){
23      Serie h1=new Serie(2,5);
24      h1.setName("Uno");
25      Serie h2=new Serie(5,10);
26      h2.setName("Dos");
27      h1.start();
28      h1.setPriority(MAX_PRIORITY);
29      h2.start();
30      h2.setPriority(MIN_PRIORITY);
31      }
32 }
```

```
2.0
5.0
5.0
10.0
Hilo--> Uno i = 2.0 Suma= 2.0
Hilo--> Uno i = 3.0 Suma= 5.0
Hilo--> Uno i = 4.0 Suma= 9.0
Hilo--> Uno i = 5.0 Suma= 14.0
Hilo--> Dos i = 5.0 Suma= 5.0
Hilo--> Dos i = 6.0 Suma= 11.0
Hilo--> Dos i = 7.0 Suma= 18.0
Hilo--> Dos i = 8.0 Suma= 26.0
Hilo--> Dos i = 9.0 Suma= 35.0
Hilo--> Dos i = 10.0 Suma= 45.0
```

Linkia FP

Pregunta de la 2a clase virtual.

Para que sirve el método run() en los threads.

CLASE

Linkia FP

Formación Profesional Oficial a Distancia