UD13.- Hilos de ejecución

Módulo: Programación
1º DAM

CONTENIDOS

- Introducción
- Definiciones
- Thead
- Runnable
- Gestión de hilos
- Sincronización de hilos

Introducción

- La programación multihilo permite realizar diferentes tareas en una aplicación de forma concurrente.

- La mayoría de las aplicaciones que usamos en nuestros ordenadores (editores de texto, navegadores, editores gráficos …) son multihilo.

- En un editor de texto, por ejemplo, un hilo puede estar controlando la ortografía del texto, otro puede estar atento a las pulsaciones sobre los iconos de la interfaz gráfica y el otro guardando el documento.

Definición

- La programación tradicional está diseñada para trabajar de forma secuencial, de modo que cuando termina un proceso se pone en marcha otro.

- Los hilos son una forma de ejecutar paralelamente diferentes partes de un mismo programa. En Java los hilos son conocidos como Threads.

- Un hilo o thread es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

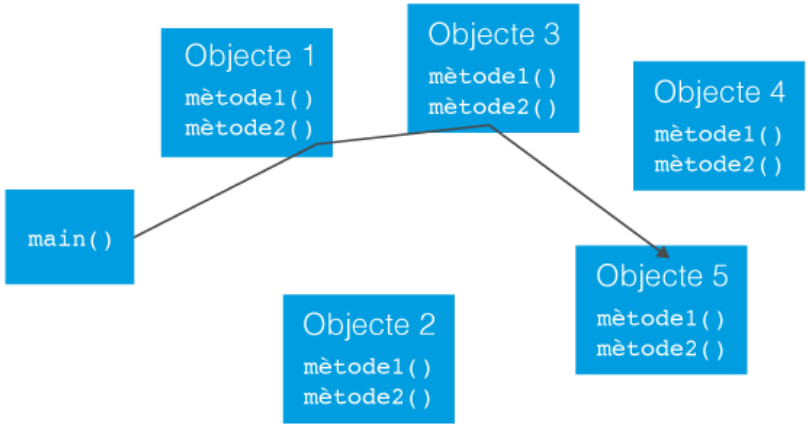
Relación proceso hilo

- Los **procesos** no comparten memoria entre ellos, son independientes, llevan información sobre su estado e interactúan con otros procesos a través de mecanismos de comunicación dados por el sistema operativo.
- Es por eso que se llaman **entidades pesadas**.

- Los **hilos**, en cambio, comparten recursos. El cambio de estado y de ejecución se realiza mucho más rápidamente y pueden comunicarse entre ellos utilizando los datos de la memoria que comparten.
- Es por eso que se llaman **entidades ligeras**.

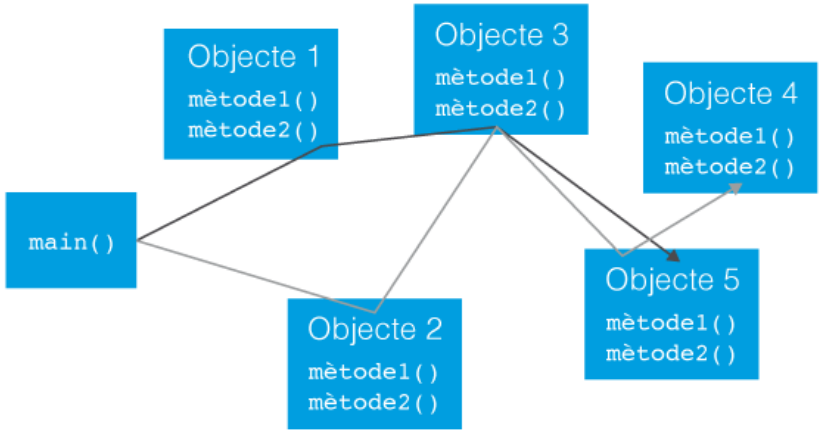
Ejecución de un proceso

- Cuando iniciamos un programa en Java hay un hilo en ejecución, el hilo principal, que es creado por el método main().
- Si no se crean más hilos que el principal, sólo existe uno y será el encargado de hacer las llamadas a los métodos y crear objetos que indique el programa.



Ejecución de un proceso con hilos (I)

- Desde el hilo principal se pueden crear otros hilos que irán ejecutando otros métodos de los objetos instanciados o métodos nuevos que sea necesario crear, de forma simultánea.



Ejecución de un proceso con hilos (II)

- Un hilo creado por main() puede crear otros hilos, que a su vez pueden crear nuevos hilos.

- Los hilos creados dependen directamente de sus creadores, de esta manera se crean jerarquías de padres e hijos de hilos de ejecución.

Los hilos en Java

- Java define la funcionalidad principal de la gestión de los hilos en la clase **Thread**.

- La herencia nos permitirá, de forma fácil, incorporar la funcionalidad de la clase Thread en nuestras implementaciones.

- Una forma alternativa a incorporar la funcionalidad de Thread en nuestras implementaciones es usar la interfaz **Runnable**.

Clase Thread. Constructores (I)

- **Thread**(): Reserva memoria para crear un objeto Thread.
- **Thread**(Runnable target): reserva memoria para la creación de un nuevo objeto de la clase Thread el que gestionará la ejecución del objeto Runnable pasado por parámetro.
- **Thread**(Runnable target, String name): reserva memoria para la creación de un nuevo objeto de la clase Thread el que gestionará la ejecución del objeto Runnable pasado por parámetro. El objeto creado se podrá identificar con el nombre pasado en el segundo parámetro.

Clase Thread. Constructores (II)

- **Thread**(ThreadGroup group, Runnable target): reserva memoria para un nuevo objeto Thread que gestionará la ejecución del objeto Runnable (target) y que pertenece al grupo de hilos (group).
- **Thread**(ThreadGroup group, Runnable target, String name): reserva memoria para un nuevo objeto Thread identificado con el nombre name, que pertenece al grupo de hilos (group) y que gestionará la ejecución del objeto Runnable target.
- **Thread**(ThreadGroup group, String name): crea un nuevo objeto Thread identificado por name y que pertenece al grupo de hilos concreto (group).

Clase Thread. Métodos (I)

- static Thread **currentThread**(): nos devuelve el hilo que se está ejecutando.
- string **getName**(): devuelve el nombre del hilo.
- void **setName**(String nombre): asigna un nombre al hilo.
- int **getPriority**(): nos dice la prioridad de ejecución del hilo.
- void **setPriority**(int Prioridad): asigna la prioridad de ejecución del hilo.
- ThreadGroup **getThreadGroup**(): devuelve el grupo de hilos al que pertenece el hilo.

Clase Thread. Métodos (II)

- boolean **isAlive**(): mira si el hilo está activo. Es decir, si ha habido una llamada inicial a su método start() y aún no ha finalizado la ejecución.
- boolean **isDaemon**(): nos indica si el hilo es un demonio o no.
- void **setDaemon**(boolean on): indica si el hilo será un demonio o un hilo de usuario. Los demonios están ligados a su creador, si su creador termina su ejecución, los hijos demonios también finalizan.

Clase Thread. Métodos (III)

- void **join**(): Detiene la ejecución del hilo que hace la llamada y espera la finalización del hilo invocado.
- void **join**(long Millis): espera la finalización del hilo invocado, pero una vez pasado el tiempo del parámetro (expresado en milisegundos), se continuará la ejecución.
- void **run**(): Representa el inicio de ejecución del hilo. Esta ejecución comienza inmediatamente después de haber llamado el método start().
- void **start**(): pone un hilo en estado de preparado para ejecutarse.

Clase Thread. Métodos (IV)

- static void **sleep**(long Millis): hace que un hilo que se está ejecutando se duerma durante un tiempo, que se establece en milisegundos. El hilo no pierde ninguna propiedad de bloqueo.
- static void **yield**(): hace que el hilo que se está ejecutando pase a estado de preparado. Sale del procesador.
- String **toString**(): devuelve una cadena que representa el hilo, nombre prioridad y su grupo.

La interfaz Runnable

- Tener que heredar de la clase Thread para crear clases con esta funcionalidad conlleva un problema, ya que Java no soporta la herencia múltiple y por tanto nos resultará imposible hacer que la nueva clase extienda a otras clases que no sean Thread.
- Para evitar esta limitación podemos utilizar la Interfaz **Runnable**.
- La interfaz Runnable fuerza la existencia del método run() pero no limita la herencia.
- Los objetos Runnable no tienen la capacidad de ejecutarse como un hilo independiente, hay que utilizar un objeto Thread para gestionar su ejecución.

La clase Object

- En la clase Object de Java se definen una serie de métodos para apoyar a los hilos de ejecución.
- **wait**(): este método hace que un hilo de ejecución pase a un estado de espera hasta que se le notifique que puede continuar la ejecución.
- **notify**(): es el método encargado de notificar a un hilo que se encuentra en espera de que puede continuar la ejecución.
- **notifyAll**(): funciona de forma similar a notify(), pero notifica que pueden continuar la ejecución todos los hilos en espera.

Creación de un hilo

- Como ya hemos comentado, existen 2 formas:
  - Heredando la clase Thread.
  - Implementando la interfaz Runnable.

- Si queremos crear una clase en la que sus instancias sean hilos, tendremos que heredar la clase Thread.

- Si queremos crear hilos en los que nuestra clase herede de alguna otra clase, deberemos utilizar la interfaz Runnable.

Creación de un hilo. Thread (I)

- La clase Thread tiene un método llamado **run**() que tendremos que sobrescribir para codificar las instrucciones a procesar durante la ejecución del hilo.
- Podríamos decir que el método **run**() es para un hilo, lo que es el método **main**() para un programa.
- Si queremos ejecutar un hilo invocaremos a su método **start**().
- Su invocación iniciará la ejecución de un nuevo hilo justo en la primera instrucción del método **run**(). Es decir la llamada de **start**() implica la invocación posterior del método **run**().

Creación de un hilo. Thread (II)

```
public class TestingThread extends Thread {
    private string strImprimir;
    public TestingThread (String str) {
        strImprimir = Str;
    }
    @Override
    public void run () {
        for(int x =0; X <5; X ++) {
            System.out.println (strImprimir+ "" + X);
        }
    }
}
```

Creación de un hilo. Thread (III)

```
public static void main (String [] args) {
    Thread primer = new TestingThread("Hilo 1");
    Thread segundo = new TestingThread("Hilo 2");
    // Hemos creado dos hilos primero y segundo, pero no se han
ejecutado.
    // Para poder ejecutar deben llamar a los métodos start()
    primer.start();
    segon.start();

    System.out.println ("Final Hilo Principal");
}
```

Creación de un hilo. Thread (IV)

- Un ejemplo de salida podría ser:

```
Final Hilo Principal
Hilo 2 0
Hilo 2 1
Hilo 2 2
Hilo 1 0
Hilo 1 1
Hilo 1 2
Hilo 1 3
Hilo 2 3
Hilo 1 4
Hilo 2 4
```

- Pero el orden de ejecución puede ser diferente cada vez que ejecutamos. Depende del planificador del SO

Creación de un hilo. Runnable (I)

- Como hemos mencionado, Java no soporta la herencia múltiple, por tanto si queremos crear hilos en los que nuestra clase herede de alguna otra clase, deberemos utilizar la interfaz Runnable.
- La clase Thread contiene un constructor que nos permite pasarle como parámetro cualquier objeto que implemente la interfaz Runnable.

Creación de un hilo. Runnable (II)

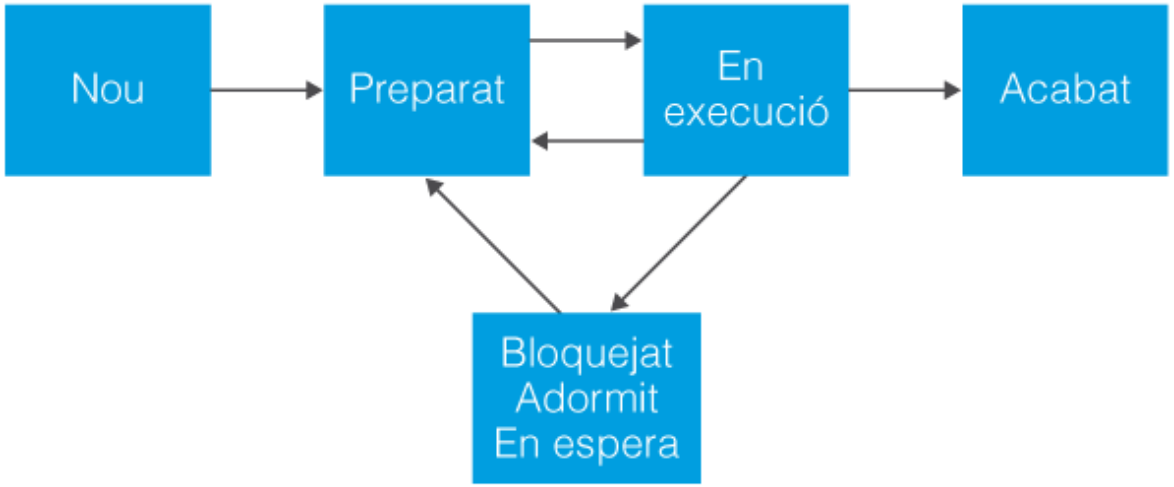
```
public class TestingRunnable implements Runnable {
    private string strImprimir;
    public TestingRunnable (String str) {
        this.strImprimir = Str;
    }
    @Override
    public void run () {
        for(int x =0; X <5; X ++) {
            System.out.println (strImprimir+ "" + X);
        }
    }
}
```

Creación de un hilo. Runnable (III)

```
public static void main (String [] args) {

    TestingRunnable objRunnable1 = new TestingRunnable("Hilo 1");

    TestingRunnable objRunnable2 = new TestingRunnable("Hilo 2");

    // Creamos dos Hilos y le pasamos por parámetros los objetos de la
clase TestingRunnable

    Thread primer = new Thread(objRunnable1);

    Thread segundo = new Thread(objRunnable2);

    // Hemos creado dos hilos primero y segundo, pero no se han
ejecutado.

    // Para poderlo ejecutar se debe llamar al método start()

    primer.start ();

    segon.start ();

    System.out.println ("Final Hilo Principal");

}
```

Estados de un hilo (I)

- El ciclo de vida de un hilo contempla diferentes estados por los que puede ir pasando durante su ejecución.



Estados de un hilo (II)

- **Nuevo** es el estado en que se encuentra un hilo cuando ha sido creado con el método new().
- El método start() pone el hilo en estado de **Preparado**. Aunque no está en ejecución, si no que está preparado para utilizar el procesador.
- El planificador del SO determinará cuando hará uso del procesador, entrando en el estado de **Ejecución**.
- Del estado de ejecución puede pasar a un estado de **Bloqueado**, **Dormido** o en **Espera** por diferentes motivos.

Estados de un hilo (III)

- Diremos que un hilo está vivo (**Alive**), mientras transite entre los estados **Preparado**, en **Ejecución**, **Bloqueado**, **Dormido** o en **Espera**.
- Cuando un hilo finalice su ejecución llegando al final del método run, pasará al estado de **Acabado** y diremos que está muerto (**Dead**), porque desde este estado no podrá volverse a ejecutar más.
- La invocación del método start() de un hilo acabado, provocará una excepción en tiempo de ejecución.

Consideraciones

- La finalización de un hilo no implica que el objeto Thread desaparezca y libere la memoria.
- Las instancias de Thread siguen las mismas reglas que cualquier otro objeto, mientras se encuentran asignadas a alguna variable, se mantendrán en memoria.
- Podemos forzar la liberación de memoria asignando un valor null a la variable que mantenía el objeto Thread, una vez éste haya finalizado.

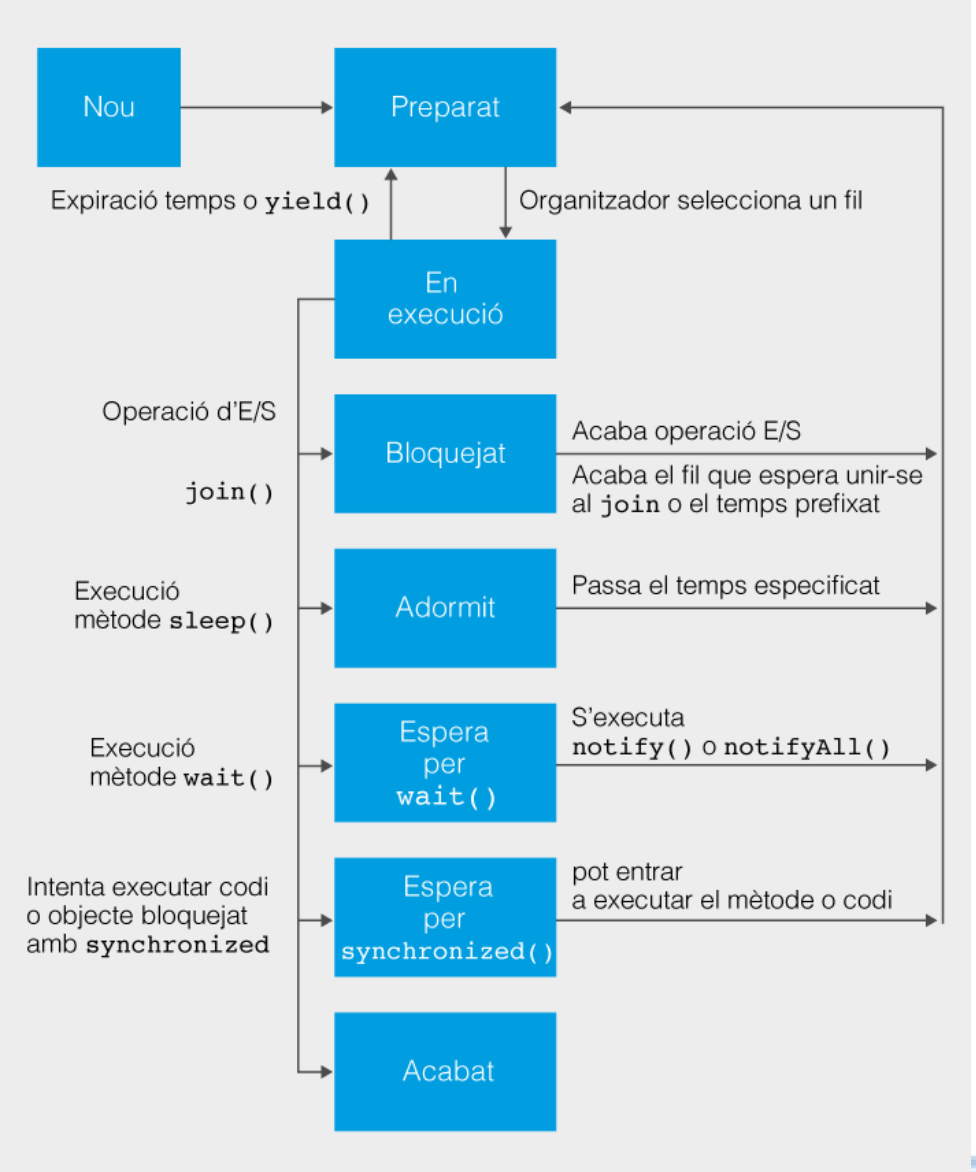
Cambios de estado (I)

- Mientras un hilo está en ejecución, podrá salir por varios motivos:
  - Ha expirado el tiempo de uso del procesador por parte del hilo o se ejecuta el método **yield**().
  - Inicia una operación de entrada / salida y pasa a un estado de bloqueado hasta que la operación termine.
  - Llama al método **join**() de otro hilo para esperar la finalización del mismo, el hilo invocador pasa al estado de bloqueado hasta que el hilo invocado pase al estado terminado.

Cambios de estado (II)

- Intenta ejecutar un código que está sincronizado (**synchronized**) y no puede porque hay otro hilo ejecutándolo, pasa a un estado de bloqueado hasta que el bloque sincronizado quede liberado.
- Se llama el método **wait**() Pasándolo al estado de espera. Cuando un hilo entra en estado de espera, sólo podrá salir si se invoca alguno de sus métodos notify() o notifyAll().
- Se invoca el método **sleep**(). El hilo pasa a estado dormido hasta que haya transcurrido el tiempo indicado como parámetro. El estado dormido, es muy útil para mantener un hilo parado durante un tiempo determinado minimizando al máximo los recursos utilizados.

Cambios de estado (III)



- La clase Thread contiene el método **isAlive**() que nos indica si un hilo ha sido inicializado con el método start().
- Si isAlive() devuelve false, sabemos que el hilo está en el estado de nuevo o finalizado, no siendo posible diferenciar estos dos estados.
- En cambio si devuelve true nos dice que el hilo ha sido inicializado con start() y se encuentra en cualquiera de los otros estados.

Otros conceptos

- La programación concurrente es un tema complejo que requiere tiempo y práctica, ya que tendemos a pensar de forma secuencial, pero hay que ir con mucho cuidado porque no podemos saber cuál será el orden de ejecución de los diversos hilos.
- En el tintero se han quedados algunos conceptos importantes como:
  - La gestión de prioridades entre hilos
  - La Interfaz Callable