1. Introducción

Concurrencia es cuando ocurre más de una cosa al mismo tiempo, algo normal en nuestra realidad, la vida diaria está llena de ejemplos en los que hacemos más de una tarea al mismo tiempo… (oímos música, mientras conducimos, mientras hablamos con nuestro/a acompañante…)

Los programas que hemos escrito hasta ahora han realizado una sola tarea a la vez, pero hay muchas aplicaciones en las que un programa tiene que hacer varias cosas al mismo tiempo, o concurrentemente. Por ejemplo, un programa de chat de Internet, que permitiría a varios usuarios participar en un grupo de discusión. El programa tendría que leer los mensajes de varios usuarios al mismo tiempo y transmitirlos a los demás participantes en el grupo. Las tareas de lectura y de difusión tendrían que llevarse a cabo concurrentemente.

En Java la programación concurrente se realiza mediante Threads (Hilos o Hebras, utilizaremos indistintamente cualquiera de estos términos para referirnos a ellos.)

Qué es un Hilo o Thread?

Procesos e Hilos:

http://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

Un Hilo (o Hilo de ejecución o hilo de control) es una sola secuencia de instrucciones dentro de un programa.

Para aplicaciones Java el flujo de control empieza en la primera instrucción del **`main()`** y continúa secuencialmente.

Un sólo thread también tiene un principio y un final, una secuencia, y en un momento dado durante el tiempo de ejecución del thread sólo hay un punto de ejecución. Sin embargo, un thread por sí mismo no es un programa. No puede ejecutarse por sí solo, pero si dentro un programa. De hecho un programa secuencial es un programa con un único hilo de ejecución

No hay nada nuevo en el concepto de un sólo hilo. Pero el juego real alrededor de los hilos no está sobre los hilos secuenciales solitarios, sino sobre la posibilidad de que un solo programa ejecute varios hilos a la vez y que realicen diferentes tareas, interactúen, se comuniquen y/o se sincronicen entre ellos.

Algunos textos utilizan el nombre ***procesos ligeros*** en lugar de hilo. Se denominan procesos ligeros porque se ejecutan dentro del contexto de un programa completo y se aprovechan de los recursos asignados a ese programa y del entorno de éste, aunque como un flujo secuencial de control, un hilo debe conseguir algunos de sus propios recursos dentro de un programa en

ejecución. (Debe tener su propia pila de ejecución y contador de programa, por ejemplo). Este término se utiliza en contraposición a ***procesos pesados*** que serían los procesos concurrentes manejados por el S.O.

Creación de Hilos en Java.

Definir y lanzar Hilos:

http://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

Tenemos dos formas de crear y lanzar un Hilo

1.- Extendiendo la clase Thread

2.- Implementado la interface Runnable

1.- EXTENDIENDO LA CLASE THREAD

- La subclase extiende la clase Thread
  - La Subclase sobreescribe el método `run()` de la clase Thread.
- Ahora podemos crear objetos instancias de la subclase creada
- Al llamar al método `start()` del objeto se inicia la ejecución del Hilo.
  - Java runtime inicia la ejecución del método `run()` del objeto en un Hilo independiente.

PODEMOS INICIAR UN HILO DE LA SUBCLASE CREADA DE DOS FORMAS:

- 1.- El método **`start()`** no está dentro del constructor de la subclase:
  - En este caso el método **`start()`** necesita ser explícitamente invocado después

	Subclase	Crear Objeto- Inciar Hilo

1	`class Hilo extends Thread{`
	`  public void run(){....}`
		`main`
		
		`Hilo h1= new Hilo(); h1.start();`
		`Hilo h2= new Hilo(); h2.start();`
		
		`// más breve, lanza 4 hilos`
		`for (int i=0;i<4;i++)`
		`(new Hilo()).start();`

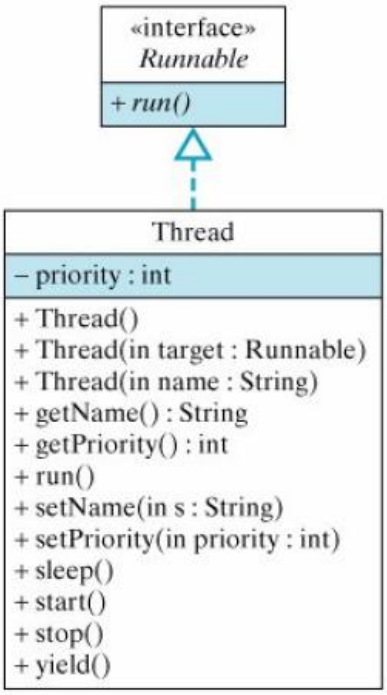
2	`class Hilo extends Thread{`
	`Hilo{`
	`...`
	`start();`
	`}`
	`public void run(){....}`
		`main`
		`Hilo h1= new Hilo();`
		`Hilo h2= new Hilo();`
		
		`//También lanzaríamos dos hilos así`
		`new Hilo();`
		`new Hilo();`

de crear el objeto para ejecutar el hilo.

- 2.- Incluimos el método **`start()`** en el constructor de la subclase:
  - El crear un objeto instancia de la subclase iniciará automáticamente el hilo.

2.- IMPLEMENTANDO LA CLASE RUNNABLE.

El uso de la clase `Thread` requiere utilizar herencia, y puesto que Java no permite herencia múltiple el uso de la interfaz `Runnable` para crear hilos permite que la clase herede de cualquier otra (más versátil). Sin embargo, la ejecución requiere de la construcción de un objeto añadido (aquel que implementa la interfaz `Runnable`) además del objeto de la clase `Thread`.



La clase `Thread` Implementa la interface `Runnable` -que consiste simplemente en el método `run()`–, como hemos dicho otra forma de crear otro hilo es instanciar un objeto `Thread` y pasarle un objeto `Runnable` que constituirá su cuerpo. Esta forma nos permite transformar cualquier clase existente en un Hilo.

La interfaz `Runnable` debe ser implementada por cualquier clase cuyas instancias están destinadas a ser ejecutadas como un hilo La clase debe definir el método `run()`.

Un objeto Runnable es cualquier objeto que implemente la interface `Runnable`. El método run () es como main () para el nuevo hilo.

DOS FORMAS DE LANZAR UN HILO PARA UNA CLASE QUE IMPLEMENTE RUNNABLE:

- 1.- Instanciando el Thread fuera (lo más normal):
  - Implementar el interface **`Runnable`** para la clase implementando el método **`void run()`** , que contendrá las instrucciones que ejecutará el hilo.
  - Crear objetos instancias **`Thread`** pasándole cada objeto **`Runnable`** como parámetro al constructor **`Thread()`.**
  - Iniciamos cada hilo llamando al método start().
- 2.- Instanciando el Thread dentro de la propia clase.

Clase	Crear Objeto- Iniciar Hilo

1
`class Hilo implements Runnable{`
`  public void run(){....}`
	`main`
	`  Thread h1,h2;`
	`  h1= new Thread(new Hilo());h1.star;`
	`  h2= new Thread(new Hilo());h2.star;`
	
	`// más breve, lanza 4 hilos`
	`  for (int i=0;i<4;i++)`
	`  new Thread(new Hilo())).start();`

2
`class Hilo implements Runnable{`
`  Thread t;`
`  Hilo{`
`  ...`
`  t= new Thread(this);`
`  start();`
`  }`
`  public void run(){....}`
 	
`public void start(){`
`  t.start();}`
	`main`
	
	`Hilo h1= new Hilo();`
	`Hilo h2= new Hilo();`

	`main`
	
	`//También lanzaríamos dos hilos así`
	`new Hilo();`
	`new Hilo();`

Entonces…¿Cuál de las dos formas utilizamos para crear hilos?

Podemos decir que la segunda (implementando Runnable) es la más general y la que más flexibilidad ofrece, aunque la primera (creando una subclase de Thread) es más sencilla y fácil de usar para aplicaciones simples. Por lo que la decisión básicamente dependerá de las necesidades del programa y nuestra propia elección.

Library: java.lang.Thread

La clase `java.lang.Thread` contiene diversos constructores sobrecargados y otros métodos para gestionar los hilos, consulta la tabla completa en:

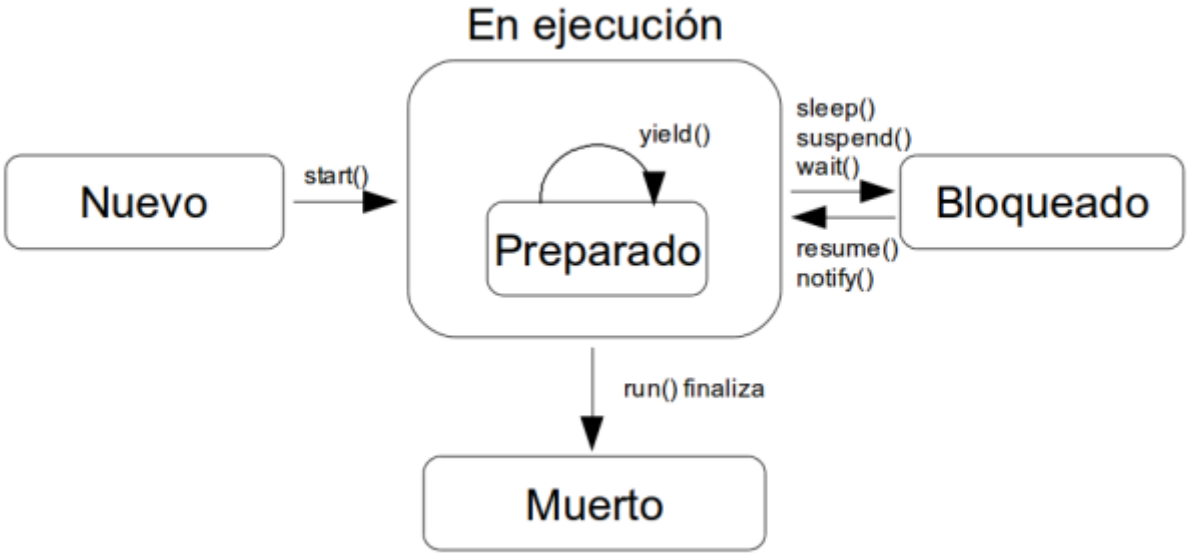
http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

2. Estados de un hilo

http://proton.ucting.udg.mx/tutorial/java/Cap7/thread.html

Un hilo tiene su propio ciclo de vida, durante el cual puede encontrarse en diferentes estados. Java controla el estado de los hilos mediante el llamado planificador de hilos, que se encargará de gestionar qué hilo debe ejecutarse en cada momento y en qué estado deben encontrarse el resto de hilos.

Observad esta imagen.



En ella podéis ver un esquema de los diferentes estados en los que puede encontrarse un hilo (representados en un recuadro) y qué métodos pueden llevar ese estado al hilo (representados por flechas).

¿QUÉ SIGNIFICAN CADA UNO DE ESTOS ESTADOS?

- ***Nuevo.*** Se ha creado un objeto hilo, pero todavía no se le ha asignado ninguna tarea. Para ello, se ha de llamar a su método *start()* y el hilo pasará al estado *preparado*.
- ***Preparado.*** El hilo está preparado para ejecutarse, pero el planificador de hilos es quién debe decidir si puede hacerlo o debe esperar (por ejemplo, a que acabe la ejecución otro hilo).
- ***En ejecución***. Una vez el hilo puede acceder a tiempo de CPU, se ejecutará. Si el hilo finaliza su trabajo completamente, pasará al estado *muerto*. Si el planificador de hilos decide que ha cumplido su periodo de tiempo y el hilo no ha finalizado su trabajo, pasará al estado *preparado* y esperará a que el planificador de hilos vuelva a darle permiso para ejecutarse.
- ***Bloqueado***. El hilo no puede ejecutarse porque espera que ocurra algo. En cuanto ocurra lo que le está dejando bloqueado, pasará al estado *preparado*.
- ***Muerto***. El hilo ha finalizado su tarea y deja de existir.

3. Ejercicios

Debes subir un solo fichero con los proyectos completos comprimidos (El nombre de los proyectos seguirá este formato: (Ejercício1, Ejercício2, Ejercício2-b,...), y el nombre del fichero comprimido será APELLIDO1-APELLIDO2-NOMBRE. **El fichero han de subirlo cada uno de los miembros del grupo**

1.-Captura del hilo principal.

En un programa secuencial también tenemos un hilo (un único hilo) que realiza la secuencia de ejecución, podemos capturar éste -**`Thread h = Thread.currentThread`**(); - y utilizar los métodos de la clase Thread con él:

- Crea un programa con un bucle que muestre una secuencia de 50 números impares comenzando por 1.
- Modifica el código de forma que: Capture el hilo del programa y muestre el nombre del hilo, modifique el nombre del hilo, muestre el nuevo nombre del hilo junto con el número en el bucle durmiendo el hilo durante 2 segundos en cada iteración.

2A.- Varios hilos.

Crea una clase a la que llamaremos **`HiloImprimeID`** que extienda **`Thread`**, con un atributo ID al que asignaremos un valor por parámetro en el constructor.

Tendremos un método **`run()`** que repetirá un bucle un número de veces indicado con otro atributo mostrando el ID del hilo

Crea una clase principal **`Numeros`** que genere 5 hilos con ID 1..5 y los lance, comprueba el funcionamiento con diversos valores para el bucle.

2B.-RUNNABLE.

Modifica el programa anterior de forma que la clase **`HiloImprimeID`** la creas implementando la interface **`Runnable`**, además el método **`run()`** dormirá el hilo un tiempo aleatorio entre 0 y 1 segundo.

Lanza 5 hilos y comprueba el resultado.

3 Espera e interrupción de un Hilo

Thread.join () :

Si un hilo quiere esperar a que otro hilo finalice, debe invocar el método **join()** en el objeto de la clase **Thread** que representa a ese hilo. (el método eleva la excepción **InterruptedExcepcion** si otro hilo le interrumpe mientras espera).Existen versiones de **join()**para indicar el tiempo máximo de espera.

Thread.stop() :

Al diseñar Java se pensó que el método **stop()** era una forma razonable de finalizar hilos El método stop() **existe** en la clase Thread, pero desde Java 1.2 (1998) se **desaconseja** su uso (el método está **obsoleto**, (***deprecated***) puesto que la tarea que realiza el hilo que finaliza se quedaría **a medias** y los objetos que se estuvieran manipulando se quedarían en un estado **inconsistente,** esto produciría errores imprevisibles en la aplicación.

Interrumpir un Hilo:

Para indicar a un hilo que debe finalizar su ejecución, se le envía una **interrupción**. Para interrumpir un hilo, se invoca el método **interrupt()** en el objeto de la clase **Thread** que representa el hilo, pero el hilo **no puede finalizar** su ejecución en un punto arbitrario (por los problemas indicados) si contemplamos la opción de que sea interrumpido necesitará **preguntar periódicamente** si otro hilo le ha interrumpido

Si es así, debería **liberar los recursos** apropiadamente, **cerrar las conexiones** y dejar los objetos en un **estado estable.** Un hilo puede determinar si ha sido interrumpido invocando el método estático **Thread.interrupted()**

Puede ocurrir que el Hilo esté bloqueado por una condición o en un **sleep** cuando otro hilo pretende interrumpirlo (por lo que no puede consultar si lo han interrumpido). Para solucionar este problema, la mayoría de los métodos que se bloquean a la espera de una determinada condición elevan la excepción **InterruptedException.** Esta excepción se eleva cuando se interrumpe el hilo.

Por tanto Para finalizar un hilo el propio hilo tiene que estar preparado para ello cerrando recursos y finalizando su ejecución, tendrá que revisar periódicamente si le han interrumpido (**Thread.interrupted()**).

EJERCICIO:

Debes implementar un programa en Java en el que, el hilo principal Main creara otro hilo con el siguiente comportamiento:

Hilo1 (main):

- Muestra el mensaje **“Estas ya?.**
- Lanza el segundo hilo.
- Espera a que el segundo hilo termine, mientras, cada segundo muestra **“Te queda mucho?**
- Si transcurridos 5 segundos el otro hilo no ha terminado muestra **“Yo me voy !”** e interrumpe al otro hilo y espera a que termine.
- Si el otro hilo termina antes muestra: **“Por fin!!..”**
- **Este hilo siempre terminará el último y mostrará el mensaje “Venga vámonos”**

Hilo2:

- En un bucle aleatorio entre 2 y 8 segundos, cada segundo muestra una de las siguientes frases aleatoriamente: **“Me estoy vistiendo..”, “Me queda un segundo...”, “Esto no me queda bien...”**
- Si el hilo principal le interrumpe antes de terminar dice: **“Que cabrón/a!!!”**
- Si termina antes dice: **“Ya estoy...”**

Nombra los hilos y que cada hilo muestre su nombre cada vez que dice algo.