INSTITUTO TECNOLÓGICO DE OAXACA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

TÓPICOS AVANZADOS DE PROGRAMACIÓN

SU

LIC. ADELINA CRUZ NIETO

UNIDAD IV

INVESTIGACIÓN I
HILOS EN JAVA

ANAYA BAUTISTA GUADALUPE BERENICE

20/07/18

INVESTIGACIÓN

1. ¿QUE ES UN HILO?

Un hilo (Thread) es un proceso en ejecución dentro de un programa java main return Thread t t.start() run() finalización.
Los threads son estructuras secuenciales que combinan las líneas de 2 o más procesos en un solo proceso.
La finalización depende del hilo (Thread.suspend, stop están depreciados).
Los hilos implementan prioridad y mecanismos de sincronización.
Cualquier clase se puede hacer hilo.

Un Hilo es un trozo de código de nuestro programa que puede ser ejecutado al mismo tiempo que otro.
En esencia la multitarea nos permite ejecutar varios procesos a la vez; es decir, de forma concurrente y por tanto eso nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes. Evidentemente no podemos ejecutar infinitos procesos de forma concurrente ya que el hardware tiene sus limitaciones, pero raro es a día de hoy los ordenadores que no tengan más de un núcleo por tanto en un procesador con dos núcleos se podrían ejecutar dos procesos a la vez y así nuestro programa utilizaría al máximo los recursos hardware.

Desde el punto de vista del usuario existen dos áreas bien diferenciadas, que el desarrollador ha de tener en cuenta:

Primer plano: Aquí se ejecuta únicamente un hilo llamado “hilo principal”. Aquí hemos programado siempre, sin conocimiento de que estábamos trabajando ya con hilos. Es el hilo que trabaja con las vistas, es decir, con la interfaz gráfica que ve el usuario: botones, ventanas emergentes, campos editables, etc.
También, puede ser usado para hacer cálculos u otros procesamientos complejos, aunque estos deberían de evitarse hacerse en este hilo a toda costa –salvo si es imposible que se hagan en otro hilo. Cabe señalar, que el primer plano influirá en la felicidad del usuario con nuestra aplicación. Aquí es donde el usuario interacciona de manera directa, además todo lo que pase aquí lo ve y lo siente.
El desarrollador ha de tener especial cuidado al trabajar con el hilo principal, pues será juzgado por el usuario –si la aplicación va lenta es porque el primer plano va lento y esto al usuario no le gusta nada. También es importante saber, que una mala gestión del primer plano por parte del desarrollador, será castigada por el sistema operativo (por ejemplo: en Android si el hilo principal de una aplicación es bloqueado más de 5 segundos, la aplicación se cerrará mostrando una ventana de forzar cierre; y seguro que recuerdas comportamientos parecidos en otros sistemas operativos cuando te dice que “la aplicación no responde, ¿deseas finalizar su ejecución?”).

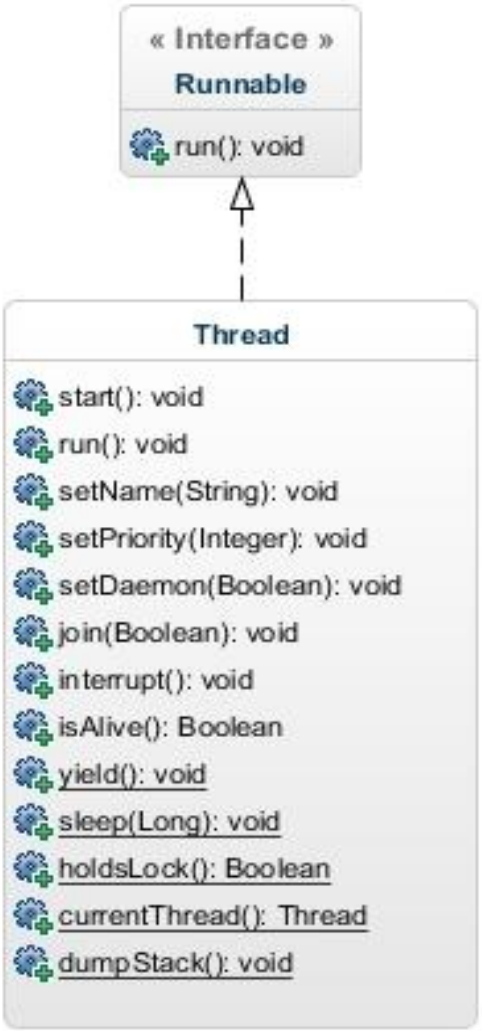
Segundo plano (o en inglés background): Se ejecuta todo el resto de hilos. El segundo plano tiene la característica de darse en el mismo momento que el primer plano. Aquí los hilos deberían de llevar las ejecuciones pesadas de la aplicación.
El segundo plano el usuario no lo ve, es más, ni le interesa, para el usuario no existe. Por lo que comprenderás, que el desarrollador puede moverse libremente por este segundo plano –dentro de unos límites. Aquí el desarrollador se puede resarcir y hacer que, por ejemplo, un método tarde horas en ejecutarse, ya que el usuario ni lo sentirá –aunque si está esperando sí que lo notará, con lo que un poco de cuidado también.

2. ¿QUÉ MÉTODOS IMPLEMENTAN LOS HILOS?

Usualmente para poder utilizarlos tenemos que crear clases que extienden a la clase Thread, y reescribir el metodo principal “run()”, el cual es el que se va a ejecutar principalmente al iniciar un hilo, thread o nuevo proceso en java.

En Java para utilizar la multitarea debemos de usar la clase Thread (es decir que la clase que implementemos debe heredar de la clase Thread) y la clase Thread implementa la Interface Runnable.

Clase Thread
- Implementa Runnable
- start() → run()
- stop(), suspend()
- setPriority()
- sleep()
- Hereda de Object
- wait(), notify()



Los hilos se comunican generalmente a través de campos y los objetos que tienen esos campos
- Es una forma de comunicación eficiente
- Pero puede plantear errores de interferencias entre hilos
- La sincronización es la herramienta para evitar este tipo de problemas, definiendo órdenes estrictos de ejecución.

wait() suspende el hilo hasta que se recibe una notificación del objeto sobre el que espera

- Solución para espera ocupada

● El proceso que espera debe tener el bloqueo intrínseco del objeto que invoca al wait
● Si no, da un error (IllegalMonitorStateException)
● Una vez invocado, el proceso suspende su ejecución y libera el bloqueo
● wait(int time) espera sólo durante un tiempo
● notify()/notifyAll() informan a uno/todos los procesos esperando por el objeto que lo invoca de que pueden continuar.

this.wait() → espera por esta instancia de la clase
● this.getClass().wait() o getClass().wait() → espera por la clase de esta instancia
● objeto.wait() → espera por el objeto que sea.
● objetoEstático.wait() → espera por la clase del objeto estático que sea
● En cualquier caso, dentro de synchronized() sobre el objeto/clase a la que espera
● Para liberar, notify/notifyAll sobre el objeto/clase sobre la que se espere

3. ¿EN QUÉ MOMENTO ES POSIBLE UTILIZAR LOS HILOS?

En esencia la multitarea nos permite ejecutar varios procesos a la vez; es decir, de forma concurrente y por tanto eso nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes. Evidentemente no podemos ejecutar infinitos procesos de forma concurrente ya que el hardware tiene sus limitaciones, pero raro es a día de hoy los ordenadores que no tengan más de un núcleo por tanto en un procesador con dos núcleos se podrían ejecutar dos procesos a la vez y así nuestro programa utilizaría al máximo los recursos hardware.

4. ¿CUALES SON LAS VENTAJAS DE USAR HILOS?

Evita que dos invocaciones de métodos sincronizados del mismo objeto se mezclen. Cuando un hilo ejecuta un método sincronizado de un objeto, todos los hilos que invoquen métodos sincronizados del objeto se bloquearán hasta que el primer hilo termine con el objeto.
● Al terminar un método sincronizado, se garantiza que todos los hilos verán los cambios realizados sobre el objeto.

Cuando un hilo invoca un método sincronizado, adquiere el bloqueo intrínseco del objeto correspondiente.

● Si invoca un método estático sincronizado, adquiere el bloqueo intrínseco de la clase, independiente de los de sus objetos.

5. ¿CUALES SON LAS DESVENTAJAS DE USAR HILOS?

Espera ocupada: un proceso espera por un recurso, pero la espera consume CPU
● while(!recurso) ; wait()
● Interbloqueo (deadlock): varios procesos compiten por los mismos recursos pero ninguno los consigue
● Inanición: un proceso nunca obtiene los recursos que solicita, aunque no esté interbloqueado
● Autobloqueo: un proceso espera por recursos que ya posee → sincronización reentrante

Autobloqueo: un proceso tiene el bloqueo intrínseco de un objeto
● Mientras lo tiene, vuelve a pedirlo
● La sincronización reentrante evita que un proceso se bloquee a sí mismo en una situación de autobloqueo
● Implementada en el núcleo de Java.

6. ¿ES POSIBLE UTILIZAR MÁS DE UN HILO EN EL IDE NETBEANS?

Cuando se crea una aplicación que puede ser utilizada por múltiples instancias (personas, máquinas y demás) simultáneamente (típico caso de un cliente - servidor) es muy común la creación de hilos para cada una de las instancias que ingresa o interactúa con nuestro sistema permitiendo así evitar problemas con lo que se conoce como código bloqueante. El código bloqueante, es un segmento de código (un método o función) que tarda una cantidad considerable de tiempo en ejecutarse (en tiempos de computación más de 5 segundos es muy significativo) y que de no ser por tener diferentes hilos asignados para cada usuario ejecutándose independientemente uno del otro, bloquearía la aplicación para los demás mientras alguno realiza el proceso bloqueante.

La mayoría de las veces el software utiliza toda la memoria ram y la cpu bloqueando otras tareas o incluso tildando por completo el ordenador. Esto se produce porque el proceso se ejecuta en primer o segundo plano consumiendo todos los recursos y bloqueando los demás procesos. Una manera de solucionar este problema es utilizando la multitarea, aprovechando que los ordenadores tienen más de un procesador. Si programamos nuestra aplicación con métodos de multitarea el ordenador no quedara bloqueado aunque si más lento pero el usuario podrá seguir ejecutando otra aplicación, ya que los procesadores pueden responder a distintas hilos ejecutando en paralelo.

Un Hilo o Thread es una parte del código que tiene la característica de ser ejecutado en forma paralela es decir al mismo tiempo que otra parte del código. También tiene la limitación del hardware ya que la multitarea depende de la cantidad de núcleos que tenga el procesador, es decir si tiene 2 núcleos podrá ejecutar dos tareas al mismo tiempo .
Existen dos planos de ejecución para un software el primer plano y el segundo plano

El Primer plano o Foreground, es lo que el usuario ve en su pantalla, que trabaja con botones, ventanas, campos de texto, imágenes, etc. Los cálculos mas comunes se realizan en el primer plano o hilo principal de la aplicación. Aquí debemos tener en cuenta que si tenemos un diseño muy pesado con demasiada gráfica, el usuario puede sentir que la aplicación es lenta y no se debe a la multitarea sino a los recursos que consume para mostrar cada pantalla.

Actualmente muchos software advierten si una aplicación consume demasiados recursos o pone en peligro la estabilidad del sistema y otros directamente cierran la aplicación que causa problemas como por Android,

donde en caso que un programa o software bloquea el dispositivo por màs de 5 segundos, lanza un mensaje de advertencia o en el peor de los casos el programa se cerrará informando que la aplicación se ha detenido. En Firefox si una web se ponen muy lenta también suelen aparecer mensajes de advertencia script no responde o Forzar cierre.

El Segundo plano o Background, aquí es donde se procesan los hilos o la multitarea. Mientas se muestra alguna pantalla, en el segundo plano deberíamos ejecutar todos los procesos pesados, aquí el usuario no ve lo que ocurre, es donde se ejecutan los servicios de software, los enlaces de red, los conexiones a bases de datos, los cálculos complejos, etc. Aquí los procesos pueden tardar el tiempo que necesiten mientras el usuario realiza otras tareas.

7. ¿QUE EXCEPCIONES ESTÁN DISPONIBLES EN JAVA AL HABLAR DE HILOS?

En LocateRegistry.getRegistry(), obtengo el error:

- java.rmi.ConnectException: Connection refused to host
- Esto suele ocurrir si el registro RMI no se ha iniciado.
- Se puede iniciar desde un terminal con – rmiregistry [port]
- O desde java con – Runtime.getRuntime().exec("rmiregistry");
- O, mejor, desde java con – LocateRegistry.createRegistry(int port);
- Ojo con iniciar un registro cuando ya hay otro corriendo
- En registro.rebind() tengo el error:
- java.rmi.UnmarshalException: error unmarshalling arguments; nested exception is: java.lang.ClassNotFoundException: rmi.Corredor
- Esto ocurre porque el servidor no encuentra los codebase para las clases registradas en RMI
- Hay que definir la ruta a los codebases para RMI de una de estas maneras: – -Djava.rmi.server.codebase=”urlClases” – System.setProperty("java.rmi.server.codebase", "file:/Users/rodri/Documents/workspace/assoo/bin/"); – Ojo: debe ser una url, apuntar a las clases compiladas (/bin/), y terminar en / – Ojo: necesitamos el codebase para hacer el stub, así que la llamada debe estar ANTES de ese punto
- En el servidor o el cliente, tengo el error:
- java.security.AccessControlException: access denied (java.net.SocketPermission 127.0.0.1:1099 connect,resolve)
- Hemos activado el gestor de seguridad, pero tiene una seguridad muy alta/indadecuada
- Crear un fichero file.policy con la política de la seguridad y enlazarlo con -Djava.security.policy como se vio en los ejemplos más arriba
- La política de seguridad deberá implementarse en cliente y servidor
- Recordad que la política de seguridad en principio sólo es necesaria si usamos como parámetro/valor de retorno en los métodos remotos algún objeto que no esté en la API.

8. BIBLIOGRAFÍA

https://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/

https://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

https://www.cs.buap.mx/~mtovar/doc/PCP/EjemplosHilos.pdf

https://www.programarya.com/Cursos-Avanzados/Java/Threads-Hilos

http://codigoprogramacion.com/cursos/java/133-uso-de-hilos-o-threads-en-java.html#.W1EkE9JKjIV

http://vis.usal.es/rodrigo/documentos/aso/JavaHilos.pdf

https://www.solvetic.com/tutoriales/article/1474-programaci%C3%B3n-multitarea-o-multihilos-con-java/