INSTITUTO TECNOLÓGICO SUPERIOR DE JEREZ

JEREZ DE GARCÍA SALINAS A 15 DE MARZO DEL 2019

NOMBRE:

GUADALUPE VÁZQUEZ DE LA TORRE

NUMERO DE CONTROL:

S17070158

CORREO:

guvadlt@Outlook.com

CARRERA:

INGENIERÍA EN SISTEMAS COMPUTACIONALES

NOMBRE DE LA MATERIA:

TOPICOS AVANZADOS DE PROGRAMACIÓN

CUARTO SEMESTRE

“ACTIVIDAD 4- TRADUCCIÓN”

DOCENTE:

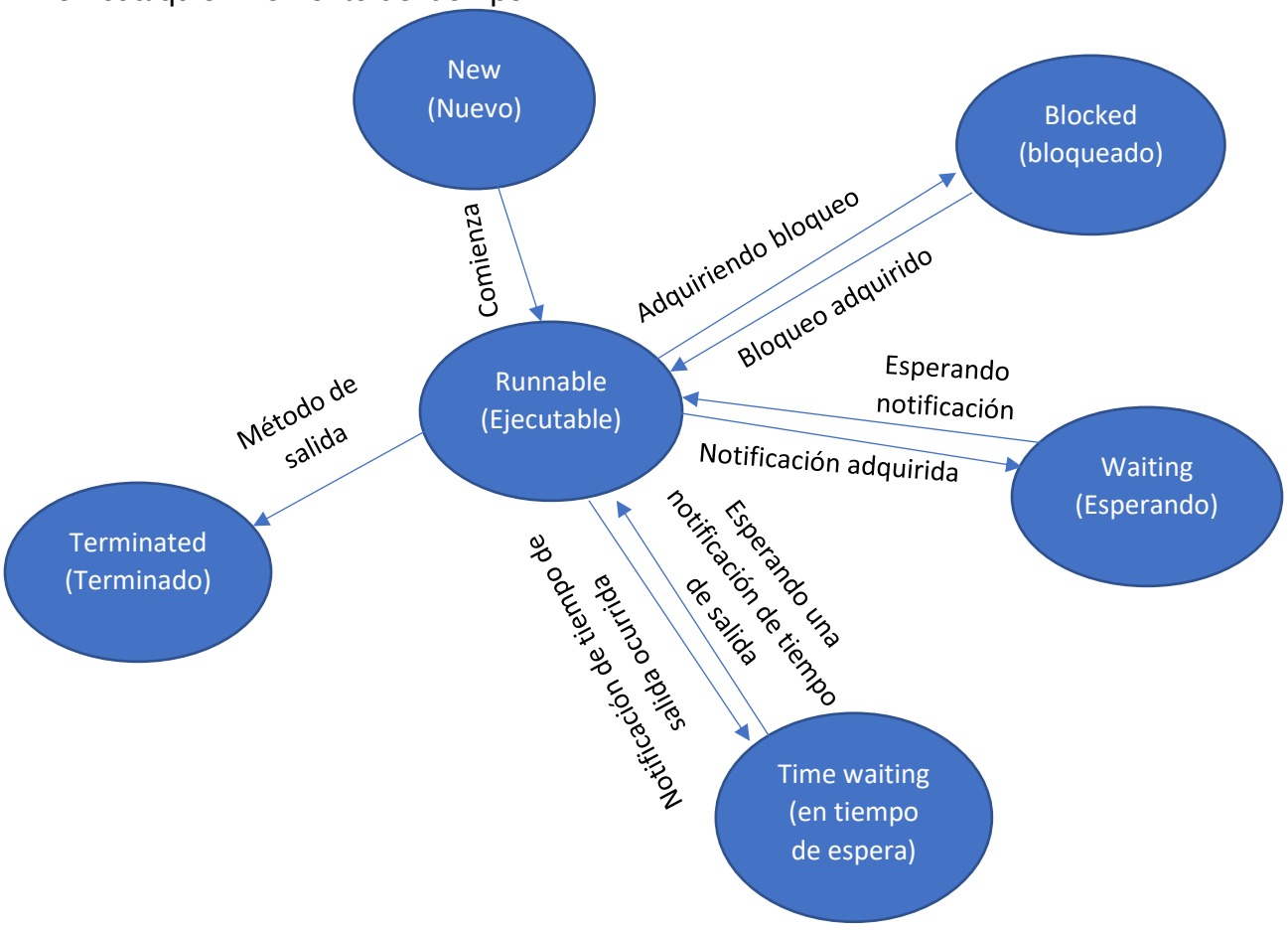
SALVADOR ACEVEDO SANDOVAL

Ciclo de vida y estados de un hilo en Java

Un hilo en Java en cualquier momento existe en alguno de los siguientes estados. Un hilo se encuentra solo en uno de los estados mostrados en cualquier instante:

1. New (Nuevo)
2. Runnable (Ejecutable)
3. Blocked (Bloqueado)
4. Waiting (Esperando)
5. Timed Waiting (Tiempo de espera)
6. Terminated (Terminado)

El diagrama que se muestra a continuación representa varios estados de un hilo en cualquier momento del tiempo.



Ciclo de vida de un hilo

1. **Nuevo hilo**: cuando se crea un nuevo hilo, se encuentra en el nuevo estado. El hilo aún no ha comenzado a ejecutarse cuando el hilo está en este estado. Cuando un hilo se encuentra en el nuevo estado, el código aún no se ha ejecutado y no ha comenzado a ejecutarse.
2. **Estado ejecutable**: un hilo que está listo para ejecutarse se mueve al estado ejecutable. En este estado, un hilo podría estar ejecutándose o podría estar listo para ejecutarse en cualquier momento. Es responsabilidad del programador de hilos dar tiempo para que se ejecute el hilo. Un programa de hilos múltiples asigna una cantidad de tiempo fija a cada hilo individual. Todos y cada uno de los hilos se ejecutan durante un breve periodo de tiempo y luego se detienen y entregan la CPU a otro hilo, para que otros hilos tengan la oportunidad de ejecutarse. Cuando esto sucede, todos los hilos que están listos para ejecutarse, esperando la CPU y el hilo que se está ejecutando actualmente, se encuentran en estado ejecutable.
3. **Estado bloqueado / en espera**: cuando un hilo está temporalmente inactivo, se encuentra en uno de los siguientes estados:
    - Bloqueado
    - Esperando

    Por ejemplo, cuando un hilo está esperando que se complete la I/O, se encuentra en el estado bloqueado. Es responsabilidad del programador de hilos reactivar y programar un hilo bloqueado / en espera. Un hilo en este estado no puede continuar su ejecución hasta que se mueva al estado ejecutable. Cualquier hilo en estos estados no consume ningún ciclo de CPU.

    Un hilo está en estado bloqueado cuando intenta acceder a una sección protegida de código que actualmente está bloqueada por otro hilo. Cuando la sección protegida está desbloqueada, la programación selecciona uno de los hilos que está bloqueado para esa sección y lo mueve al estado ejecutable. Considerando que, un hilo está en estado de

espera cuando espera a otro hilo en una condición. Cuando se cumple esta condición, se notifica al programador y el hilo en espera se mueve al estado ejecutable.

Si un hilo que se está ejecutando actualmente se mueve al estado bloqueado / en espera, el programador de hilos programará otro hilo en el estado ejecutable. Es responsabilidad del programador de hilos determinar qué hilo ejecutar.

4. **Tiempo de espera**: un hilo se encuentra en estado de tiempo de espera cuando llama a un método con un parámetro de tiempo de espera. Un hilo se encuentra en este estado hasta que se completa el tiempo de espera o hasta que se recibe una notificación. Por ejemplo, cuando un hilo llama a suspensión o espera condicional, se mueve al estado de espera temporizada.
5. **Estado terminado**: un hilo termina debido a cualquiera de los siguientes motivos:
    - Porque existe normalmente. Esto sucede cuando el código del hilo ha sido ejecutado por completo por el programa.
    - Porque se produjo un evento erróneo inusual, como un error de segmentación o una excepción no controlada.
      Un hilo que se encuentra en estado terminado ya no consume ningún ciclo de CPU.

Implementando Estados de Hilos en Java

En Java, para obtener el estado actual del hilo, use el método Thread.getState() para obtener el estado actual del hilo. Java proporciona la clase java.lang.Thread.State que define las constantes ENUM para el estado de un hilo, como resume lo siguiente:

1. **Tipo constante: Nuevo (NEW)**
   Declaración: public static final Thread.State NEW
   Descripción: estado del hilo para un hilo que aún no se ha iniciado.

2. **Tipo de constante: Ejecutable (RUNNABLE)**

   Declaración: public static final Thread.State RUNNABLE

   Descripción: Estado del hilo para un hilo ejecutable. Un hilo en el estado ejecutable se está ejecutando en la máquina virtual Java pero puede estar esperando otros recursos del sistema operativo, como el procesador.

3. **Tipo de constante: Bloqueado (BLOCKED)**

   Declaración: public static final Thread.State BLOCKED

   Descripción: Estado del hilo para un hilo bloqueado esperando un bloqueo del monitor. Un hilo en el estado bloqueado está esperando a que un bloqueo de monitor ingrese a un bloque / método sincronizado o vuelva a ingresar a un bloque / método sincronizado después de llamar a Object.wait().

4. **Tipo constante: En espera (WAITING)**

   Declaración: public static final Thread.State WAITING

   Descripción: Estado del hilo para un hilo en espera. Estado del hilo para un hilo en espera. Un hilo está en estado de espera debido a que se llama a uno de los siguientes métodos:

   - Object.wait sin tiempo de espera
   - Thread.join sin tiempo de espera
   - LockSupport.park

   Un hilo en el estado de espera está esperando a que otro hilo realice una acción en particular.

5. **Tipo de constante: Tiempo de espera (TIMED_WAITING)**

   Declaración: public static final Thread.State TIMED_WAITING

   Descripción: Estado del hilo para un hilo en espera con un tiempo de espera especificado. Un hilo está en el estado de espera programada debido a que se llama a uno de los siguientes métodos con un tiempo de espera positivo especificado:

   - Thread.sleep
   - Object.wait with timeout
   - Thread.join with timeout

- LockSupport.parkNanos
- LockSupport.parkUntil

6. **Tipo de constante: Terminado (TERMINATED)**

   Declaración: public static final Thread.State TERMINATED

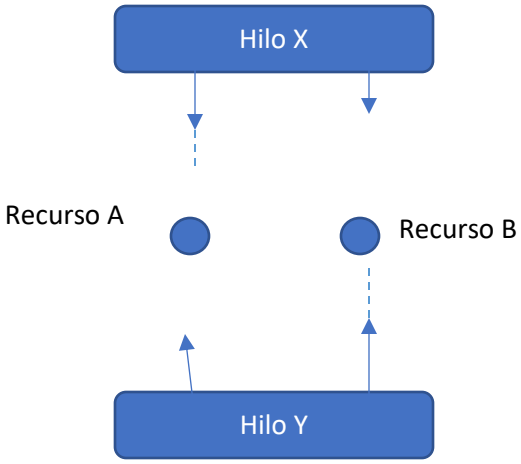
   Descripción: Estado del hilo para un hilo terminado. El hilo ha finalizado su ejecución.

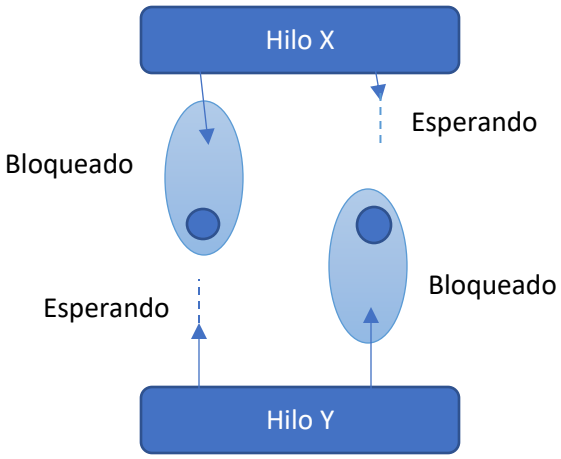
   Explicación: cuando se crea un nuevo hilo, el hilo está en el estado NUEVO. Cuando se llama al método start() en un hilo, el programador de hilos lo mueve al estado Ejecutable. Siempre que se llama al método join() en una instancia de hilo, el hilo actual que ejecuta esa declaración esperará a que este hilo se mueva al estado Terminado. Por lo tanto, antes de que se imprima la declaración final en la consola, el programa llama a join() en el hilo2 haciendo que el hilo1 espere mientras el hilo2 completa su ejecución y se mueve al estado Terminado.thread1 pasa al estado En espera porque está esperando que thread2 complete su ejecución como lo ha llamado join en thread2.

Interbloqueo en multiprocesos (MultiHilo) de Java

La palabra clave sincronized se usa para hacer que la clase o el método sean seguros para hilos, lo que significa que solo un hilo puede tener bloqueo del método sincronizado y usarlo, otros hilos tienen que esperar hasta que se libere el bloqueo y cualquiera de ellos obtenga ese bloqueo.

Es importante utilizarlo si nuestro programa se está ejecutando en un entorno de hilos múltiples donde dos o más hilos se ejecutan simultáneamente. Pero a veces también causa un problema que se llama punto muerto. A continuación, se muestra un ejemplo simple de la condición de punto muerto.





El programa ejemplo se ejecuta durante un tiempo indefinido, porque los subprocesos se encuentran en una condición de punto muerto y no permiten que se ejecute el código. Ahora veamos paso a paso lo que está sucediendo allí.

1. El hilo t1 se inicia y llama al método test1 tomando el bloqueo de objeto de s1.
2. El hilo t2 se inicia y llama al método test2 tomando el bloqueo de objeto de s2.
3. t1 imprime test1-begin y t2 imprime test-2 comienza y ambos esperan 1 segundo, por lo que ambos hilos pueden iniciarse si alguno de ellos no lo está.
4. t1 imprime test1-begin y t2 imprime test-2 comienza y ambos esperan 1 segundo, por lo que ambos hilos pueden iniciarse si alguno de ellos no lo está.
5. t1 intenta tomar el bloqueo de objeto de s2 y llamar al método test2, pero como t2 ya lo ha adquirido, espera hasta que se libere. No liberará el bloqueo de s1 hasta que obtenga el bloqueo de s2.
6. Lo mismo sucede con t2. Intenta tomar el bloqueo de objeto de s1 y llamar al método test1, pero ya está adquirido por t1, por lo que tiene que esperar hasta que t1 libere el bloqueo. t2 tampoco liberará el bloqueo de s2 hasta que obtenga el bloqueo de s1.
7. Ahora, ambos hilos están en estado de espera, esperando que los demás liberen bloqueos. Ahora hay una carrera en torno a la condición de que quien libere primero el bloqueo.
8. Como ninguno de ellos está listo para liberar el bloqueo, esta es la condición de bloqueo muerto.
9. Cuando ejecutes este programa, parecerá que la ejecución está en pausa

Detectar la condición de bloqueo muerto

También podemos detectar el interbloqueo ejecutando este programa en cmd. Tenemos que recoger Thread Dump. El comando para recopilar depende del tipo de sistema operativo. Si estamos usando Windows y Java 8, el comando es jcmd $ PID Thread.print

Podemos obtener PID ejecutando el comando jps.

Se menciona claramente que se encontró un punto muerto. Es posible que aparezca el mismo mensaje cuando lo intente en su máquina.

Evitar la condición de bloqueo mortal

Podemos evitar la condición de bloqueo muerto conociendo sus posibilidades. Es un proceso muy complejo y no es fácil de atrapar. Pero aún si lo intentamos, podemos evitar esto. Hay algunos métodos por los cuales podemos evitar esta condición. No podemos eliminar por completo su posibilidad, pero podemos reducir.

- **Evite los bloques anidados**: esta es la razón principal del interbloqueo. El interbloqueo ocurre principalmente cuando damos bloqueos a varios subprocesos. Evita dar bloqueo a múltiples hilos si ya hemos dado a uno.
- **Evite bloqueos innecesarios**: deberíamos tener bloqueo solo a los miembros que se requieren. Tener un bloqueo innecesario puede provocar un bloqueo muerto.
- **Usando unión de hilo**: la condición de bloqueo muerto aparece cuando un hilo está esperando a que termine otro. Si se produce esta condición, podemos usar Thread.join con el tiempo máximo que crea que durará la ejecución.

Puntos importantes:

- Si los hilos están esperando el uno al otro para finalizar, la condición se conoce como punto muerto.
- La condición de punto muerto es una condición compleja que se produce solo en el caso de varios hilos.
- La condición de punto muerto puede romper su código en tiempo de ejecución y puede destruir la lógica empresarial.
- Debemos evitar esta condición tanto como podamos.

Bibliografía:

- Geeks for geeks (s.f), Lifecycle and States of a Thread in Java, recuperado de: https://www.geeksforgeeks.org/lifecycle-and-states-of-a-thread-in-java/
- Geeks for geeks (s.f), Deadlock in Java Multithreading, recuperado de: https://www.geeksforgeeks.org/deadlock-in-java-multithreading/