INSTITUTO TECNOLÓGICO DE OAXACA

INGENIERÍA EN SISTEMAS COMPUTACIONALES

TÓPICOS AVANZADOS DE PROGRAMACIÓN

SU

LIC. ADELINA CRUZ NIETO

UNIDAD IV

INVESTIGACIÓN II

ANAYA BAUTISTA GUADALUPE BERENICE

25/07/18

Método join
Método synchronized
Método wait
Método notifyall

Método Join()

join

public final void join(long millis)
throws InterruptedException

Waits at most millis milliseconds for this thread to die. A timeout of 0 means to wait forever.

This implementation uses a loop of this.wait calls conditioned on this.isAlive. As a thread terminates the this.notifyAll method is invoked. It is recommended that applications not use wait, notify, or notifyAll on Threadinstances.

Parameters:

millis - the time to wait in milliseconds

Throws:

IllegalArgumentException - if the value of millis is negative

InterruptedException - if any thread has interrupted the current thread. The interrupted status of the current thread is cleared when this exception is thrown.

join

public final void join(long millis,
int nanos)
throws InterruptedException

Waits at most millis milliseconds plus nanos nanoseconds for this thread to die.

This implementation uses a loop of this.wait calls conditioned on this.isAlive. As a thread terminates the this.notifyAll method is invoked. It is recommended that applications not use wait, notify, or notifyAll on Threadinstances.

Parameters:

millis - the time to wait in milliseconds

nanos - 0-999999 additional nanoseconds to wait

Throws:

IllegalArgumentException - if the value of millis is negative, or the value of nanos is not in the range 0-999999

InterruptedException - if any thread has interrupted the current thread. The interrupted status of the current thread is cleared when this exception is thrown.

join

public final void join()
throws InterruptedException

Waits for this thread to die.

An invocation of this method behaves in exactly the same way as the invocation

join(0)

Throws:

InterruptedException - if any thread has interrupted the current thread. The interrupted status of the current thread is cleared when this exception is thrown.

SYNCHRONIZED

La palabra reservada synchronized se usa para indicar que ciertas partes del código, (habitualmente, una función miembro) están sincronizadas, es decir, que solamente un subproceso puede acceder a dicho método a la vez.
Cada método sincronizado posee una especie de llave que puede cerrar o abrir la puerta de acceso. Cuando un subproceso intenta acceder al método sincronizado mirará a ver si la llave está echada, en cuyo caso no podrá accederlo. Si método no tiene puesta la llave entonces el subproceso puede acceder a dicho código sincronizado.
Las siguientes porciones de código son ejemplos de uso del modificador synchronized

```
synchronized public void funcion1(){
    //...
}
public void funcion2(){
      Rectangle rect;
      synchronized(rect){
            rect.width+=2;
      }
      rect.height-=3;
}
```

Un ejemplo que veremos más adelante es la sincronización de una porción de código usando el objeto this, en el interior de una función miembro denominada mover

```
public void mover(){
    synchronized (this) {
     indice++;
     if (indice>= numeros.length) {
          indice=0;
     }
   }
//...
}
```

En la primera porción de código, hemos asegurado un método de modo que un sólo subproceso a la vez puede acceder a la funcion1. En la segunda y tercera porción de código, tenemos un bloque de código asegurado. La anchura width del rectángulo rect no puede ser modificada por varios subprocesos a la vez. La altura height del rectángulo no está dentro del bloque sincronizado y puede ser modificada por varios subprocesos a la vez. El objeto rect se usa en este caso como llave de dicho bloque de código, en el tercer ejemplo este papel lo representa this.
Se debe evitar la sincronización de bloques de código y sustituirlas siempre que sea posible por la sincronización de métodos, lo que está más de acuerdo con el espíritu de la programación orientada a objetos. Se debe tener en cuenta que la sincronización disminuye el rendimiento de una aplicación, por tanto, debe emplearse solamente donde sea estrictamente necesario.

El bloque synchronized lleva entre paréntesis la referencia a un objeto. Cada vez que un thread intenta acceder a un bloque sincronizado le pregunta a ese objeto si no hay algún otro thread que ya tenga el lock para ese objecto. En otras palabras, le pregunta si no hay otro thread ejecutando algun bloque sincronizado con ese objeto (y recalco que es ese objeto porque en eso radica la clave para entender el funcionamiento)
Si el lock está tomado por otro thread, entonces el thread actual es suspendido y puesto en espera hasta que el lock se libere. Si el lock está libre, entonces el thread actual toma el lock del objeto y entra a ejecutar el bloque. Al tomar el lock, cuando venga el proximo thread a intentar ejecutar un bloque sincronizado con ese objeto, será puesto en espera.
¿Cuándo se libera el lock? Se libera cuando el thread que lo tiene tomado sale del bloque por cualquier razón: termina la ejecución del bloque normalmente, ejecuta un return o lanza una excepción.

Es importante notar una vez más que el lock es sobre un objeto en particular. Si hay dos bloques synchronized que hacen referencia a distintos objetos (por más que ambos utilicen el mismo nombre de variable), la ejecución de estos bloques no será mutuamente excluyente.

WAIT

public final void wait(long timeout) throws InterruptedException

Hace que el subproceso actual espere hasta que otro subproceso invoque el método notify () o el método notifyAll () para este objeto, o que haya transcurrido un período de tiempo especificado.
El hilo actual debe poseer el monitor de este objeto.

Este método hace que el hilo actual (llámalo T) se coloque en el conjunto de espera para este objeto y luego renuncie a todas y cada una de las reivindicaciones de sincronización en este objeto. El subproceso T se desactiva para propósitos de programación de subprocesos y permanece inactivo hasta que ocurre una de estas cuatro cosas:

Algún otro subproceso invoca el método de notificación para este objeto y el subproceso T sucede que se elige arbitrariamente como el subproceso que se debe activar.
Algún otro subproceso invoca el método notifyAll para este objeto.
Algún otro hilo interrumpe el hilo T.
La cantidad especificada de tiempo real ha transcurrido, más o menos. Si el tiempo de espera es cero, sin embargo, el tiempo real no se toma en consideración y el hilo simplemente espera hasta que se le notifique.
El subproceso T se elimina del conjunto de espera para este objeto y se vuelve a habilitar para la programación de subprocesos. Luego compite de la manera habitual con otros hilos por el derecho a sincronizarse en el objeto; una vez que ha obtenido el control del objeto, todos sus reclamos de sincronización en el objeto se restauran al status quo ante, es decir, a la situación a partir del momento en que se invocó el método de espera. El hilo T regresa luego de la invocación del método de espera. Por lo tanto, al regresar del método de espera, el estado de sincronización del objeto y del hilo T es exactamente como lo era cuando se invocaba el método de espera.
Un hilo también puede despertarse sin que se le notifique, interrumpa o agote el tiempo, lo que se conoce como activación espuria. Si bien esto raramente ocurrirá en la práctica, las aplicaciones deben evitarlo al probar la condición que debería haber provocado el despertar del hilo y continuar esperando si la condición no se cumple. En otras palabras, las esperas siempre deben ocurrir en bucles, como este:

```
sincronizado (obj) {
   while (<condición no se mantiene>)
      obj.wait (tiempo de espera);
   ... // Realizar una acción apropiada para la condición
}
```

(Para obtener más información sobre este tema, consulte la Sección 3.2.3 en la "Programación simultánea en Java (Segunda Edición)" de Doug Lea (Addison-Wesley, 2000), o el Artículo 50 en "Guía efectiva de lenguaje de programación en Java" de Joshua Bloch (Addison- Wesley, 2001).
Si el hilo actual se ve interrumpido por un hilo antes o mientras espera, se lanza una excepción InterruptedException. Esta excepción no se lanza hasta que el estado de bloqueo de este objeto se haya restaurado como se describió anteriormente.

Tenga en cuenta que el método de espera, ya que coloca el hilo actual en el conjunto de espera para este objeto, desbloquea solo este objeto; cualquier otro objeto en el que se pueda sincronizar el hilo actual permanece bloqueado mientras el hilo espera.

Este método solo debe invocarse mediante un hilo que sea el propietario del monitor de este objeto. Consulte el método de notificación para obtener una descripción de las formas en que un hilo puede convertirse en el propietario de un monitor.

Parámetros:
tiempo de espera: el tiempo máximo para esperar en milisegundos.
Lanza:
IllegalArgumentException: si el valor del tiempo de espera es negativo.
IllegalMonitorStateException: si el hilo actual no es el propietario del monitor del objeto.
InterruptedException: si un hilo ha interrumpido el hilo actual antes o mientras el hilo actual estaba esperando una notificación. El estado interrumpido del hilo actual se borra cuando se lanza esta excepción.
Ver también:
notify (), notifyAll ().

public final void wait(long timeout, int nanos) throws InterruptedException

Hace que el subproceso actual espere hasta que otro subproceso invoque el método notify () o el método notifyAll () para este objeto, o algún otro subproceso interrumpa el subproceso actual, o que haya transcurrido una cierta cantidad de tiempo real.
Este método es similar al método de espera de un argumento, pero permite un control más preciso sobre la cantidad de tiempo para esperar una notificación antes de darse por vencido. La cantidad de tiempo real, medida en nanosegundos, viene dada por:

1000000 * timeout + nanos
En todos los demás aspectos, este método hace lo mismo que el método wait (long) de un argumento. En particular, wait (0, 0) significa lo mismo que wait (0).

El hilo actual debe poseer el monitor de este objeto. El subproceso libera la propiedad de este monitor y espera hasta que se produzca cualquiera de las siguientes dos condiciones:

Otro subproceso notifica que los subprocesos que esperan en el monitor de este objeto se activan a través de una llamada al método notify o al método notifyAll.
El tiempo de espera, especificado por timeout milisegundos más nanos nanosegundos, ha transcurrido.
El subproceso luego espera hasta que pueda volver a obtener la propiedad del monitor y reanuda la ejecución.

Como en la versión de un argumento, son posibles las interrupciones y los despertadores espurios, y este método siempre debe usarse en un bucle:

```
sincronizado (obj) {
    while (<condición no se mantiene>)
        obj.wait (tiempo de espera, nanos);
    ... // Realizar una acción apropiada para la condición
}
```

Este método solo debe invocarse mediante un hilo que sea el propietario del monitor de este objeto. Consulte el método de notificación para obtener una descripción de las formas en que un hilo puede convertirse en el propietario de un monitor.
Parámetros:
tiempo de espera: el tiempo máximo para esperar en milisegundos.
nanos: tiempo adicional, en nanosegundos, rango 0-999999.
Lanza:

IllegalArgumentException - si el valor de timeout es negativo o el valor de nanos no está en el rango 0-999999.
IllegalMonitorStateException: si el hilo actual no es el propietario del monitor de este objeto.
InterruptedException: si un hilo ha interrumpido el hilo actual antes o mientras el hilo actual estaba esperando una notificación. El estado interrumpido del hilo actual se borra cuando se lanza esta excepción.

public final void wait() throws InterruptedException

Hace que el subproceso actual espere hasta que otro subproceso invoque el método notify () o el método notifyAll () para este objeto. En otras palabras, este método se comporta exactamente como si simplemente realizara la espera de llamada (0).
El hilo actual debe poseer el monitor de este objeto. El subproceso libera la propiedad de este monitor y espera hasta que otro subproceso notifique a los subprocesos que esperan en el monitor de este objeto que se activen mediante una llamada al método notify o al método notifyAll. El subproceso luego espera hasta que pueda volver a obtener la propiedad del monitor y reanuda la ejecución.

Como en la versión de un argumento, son posibles las interrupciones y los despertadores espurios, y este método siempre debe usarse en un bucle:

```
sincronizado (obj) {
    while (<condición no se mantiene>)
        obj.wait ();
    ... // Realizar una acción apropiada para la condición
}
```

Este método solo debe invocarse mediante un hilo que sea el propietario del monitor de este objeto. Consulte el método de notificación para obtener una descripción de las formas en que un hilo puede convertirse en el propietario de un monitor.
Lanza:
IllegalMonitorStateException: si el hilo actual no es el propietario del monitor del objeto.
InterruptedException: si un hilo ha interrumpido el hilo actual antes o mientras el hilo actual estaba esperando una notificación. El estado interrumpido del hilo actual se borra cuando se lanza esta excepción.
Ver también:
notify (), notifyAll ()

```java
package com.arquitecturajava;

import java.util.ArrayList;

public class Bolsa {

private ArrayList<Producto> listaProductos = new
ArrayList<Producto>();

public void addProducto(Producto producto) {

if (!estaLlena())
 listaProductos.add(producto);
 }

public ArrayList<Producto> getListaProductos() {
 return listaProductos;

}

public int getSize() {
 return listaProductos.size();
 }
 public boolean estaLlena() {

return listaProductos.size() >= 5;
 }
}
```

```java
1 package com.arquitecturajava;
2
3 public class Producto {
4
5  private String nombre;
6
7  public String getNombre() {
8  return nombre;
9  }
10
11 public void setNombre(String nombre) {
12 this.nombre = nombre;
13 }
14}
```

```java
package com.arquitecturajava;

public class HiloEnvio extends Thread {

private Bolsa bolsa;

public HiloEnvio(Bolsa bolsa) {
 super();
 this.bolsa = bolsa;
 }
```

```
@Override
 public void run() {

if (bolsa.estaLlena() != true) {
try {
 synchronized (bolsa) {
 bolsa.wait();
 }} catch (InterruptedException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }
 System.out.println("Enviando la bolsa con "+
bolsa.getSize()+"elementos");
 }}
public Bolsa getBolsa() {
 return bolsa;
 }
public void setBolsa(Bolsa bolsa) {
 this.bolsa = bolsa;
 }}
package com.arquitecturajava;

public class Principal {

 public static void main(String[] args) {

 Bolsa bolsa= new Bolsa();
 HiloEnvio hilo= new HiloEnvio(bolsa);
 hilo.start();

 for(int i=0;i<=10;i++) {

 Producto p= new Producto();
 try {

 synchronized (bolsa) {

 Thread.sleep(1000);
 if (bolsa.estaLlena()) {
 bolsa.notify();
 }
 }
 } catch (InterruptedException e) {
 e.printStackTrace();
 }

 bolsa.addProducto(p);
 System.out.println(bolsa.getSize());

 }
 }
}
```

NOTIFY y NOTIFYALL

public final void notify ()

Despierta un único hilo que está esperando en el monitor de este objeto. Si hay hilos esperando en este objeto, uno de ellos se elige para ser despertado. La elección es arbitraria y ocurre a discreción de la implementación. Un hilo espera en el monitor de un objeto llamando a uno de los métodos de espera.
El hilo despertado no podrá continuar hasta que el hilo actual renuncie al bloqueo de este objeto. El hilo despertado competirá de la manera habitual con cualquier otro hilo que compita activamente para sincronizarse en este objeto; por ejemplo, el hilo despertado no goza de ningún privilegio o desventaja confiable al ser el siguiente hilo para bloquear este objeto.

Este método solo debe invocarse mediante un hilo que sea el propietario del monitor de este objeto. Un hilo se convierte en el propietario del monitor del objeto de tres maneras:

Ejecutando un método de instancia sincronizada de ese objeto.
Ejecutando el cuerpo de una instrucción sincronizada que se sincroniza en el objeto.
Para objetos de tipo Clase, ejecutando un método estático sincronizado de esa clase.
Solo un hilo a la vez puede poseer el monitor de un objeto.

Lanza:
IllegalMonitorStateException: si el hilo actual no es el propietario del monitor de este objeto.
Ver también:
notifyAll (), wait ()

public final void notifyAll ()
Despierta todos los hilos que están esperando en el monitor de este objeto. Un hilo espera en el monitor de un objeto llamando a uno de los métodos de espera.
Los hilos despertados no podrán continuar hasta que el hilo actual renuncie al bloqueo de este objeto. Los hilos despertados competirán de la manera habitual con cualquier otro hilo que compita activamente para sincronizarse en este objeto; por ejemplo, los hilos despertados no disfrutan de ningún privilegio o desventaja confiable al ser el siguiente hilo para bloquear este objeto.

Este método solo debe invocarse mediante un hilo que sea el propietario del monitor de este objeto. Consulte el método de notificación para obtener una

descripción de las formas en que un hilo puede convertirse en el propietario de un monitor.

Lanza:
IllegalMonitorStateException: si el hilo actual no es el propietario del monitor de este objeto.
Ver también:
notify (), wait ()

```
public class MiListaSincronizada
{
   private LinkedList lista = new LinkedList();

   public synchronized void addDato(Object dato)
   {
      lista.add(dato);
      lista.notify();
   }

   public synchronized Object getDato()
   {
      if (lista.size()==0)
         wait();
      Object dato = lista.get(0);
      lista.remove(0);
      return dato;
   }
}
```

BIBLIOGRAFÍA:

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

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#notify()

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#notifyAll()

http://www.chuidiang.org/java/hilos/wait_y_notify.php

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#wait()

https://www.arquitecturajava.com/java-wait-notify-y-threads/