Tema 4 -Hilos 3

1. Sincronización y comunicación de hilos.

¿Cómo conseguimos que los hilos se ejecuten de manera coordinada? Utilizando sincronización y comunicación de hilos:

- **Sincronización:** Es la capacidad de informar de la situación de un hilo a otro. El objetivo es establecer la secuencialidad correcta del programa.
- **Comunicación:** Es la capacidad de transmitir información desde un hilo a otro. El objetivo es el intercambio de información entre hilos para operar de forma coordinada.

En Java la sincronización y comunicación de hilos se consigue mediante:

- **Monitores:** Se crean al marcar bloques de código con la palabra synchronized.
- **Semáforos:** Podemos implementar nuestros propios semáforos, o bien utilizar la clase Semaphore incluida en el paquete java.util.concurrent.
- **Notificaciones:** Permiten comunicar hilos mediante los métodos wait(), notify() y notifyAll() de la clase java.lang.Object.

1. La clase Semaphore.

La clase Semaphore del paquete java.util.concurrent, permite definir un semáforo para controlar el acceso a un recurso compartido.

Para crear y usar un objeto Semaphore haremos lo siguiente:

- Indicar al constructor Semaphore (int permisos) el total de permisos que se pueden dar para acceder al mismo tiempo al recurso compartido. Este valor coincide con el número de hilos que pueden acceder a la vez al recurso.
- Indicar al semáforo mediante el método acquire(), que queremos acceder al recurso, o bien mediante acquire(int permisosAdquirir) cuántos permisos se quieren consumir al mismo tiempo.
- Indicar al semáforo mediante el método release(), que libere el permiso, o bien mediante release(int permisosLiberar), cuantos permisos se quieren liberar al mismo tiempo.

Hay otro constructor Semaphore (int permisos, boolean justo) que mediante el parámetro justo permite garantizar que el primer hilo en invocar adquire () será el primero en adquirir un permiso cuando sea liberado garantizando un orden secuencial de acceso.

¿Desde dónde se deben invocar estos métodos? Esto dependerá del uso de Semaphore.

- Si se usa para proteger secciones críticas, la llamada a los métodos acquire() y release() se hará desde el recurso compartido o sección crítica, y el número de permisos pasado al constructor será 1.
- Si se usa para comunicar hilos, en este caso un hilo invocará al método acquire() y otro hilo invocará al método release() para así trabajar de manera coordinada. El número de permisos pasado al constructor coincidirá con el número máximo de hilos bloqueados en la cola o lista de espera para adquirir un permiso.

2. *Información compartida entre hilos.*

Las secciones críticas son aquellas secciones de código que no pueden ejecutarse concurrentemente, pues en ellas se encuentran los recursos o información que comparten diferentes hilos, y que por tanto pueden ser problemáticas.

La forma de proteger las secciones críticas es mediante sincronización. La sincronización se consigue mediante:

- **Exclusión mutua:** Asegurar que un hilo tiene acceso a la sección crítica de forma exclusiva y por un tiempo finito.
- **Por condición:** Asegurar que un hilo no progrese hasta que se cumpla una determinada condición.

En Java, la sincronización para el acceso a recursos compartidos se basa en el concepto de monitor.

3. *Monitores. Métodos synchronized.*

En Java, un monitor es una porción de código protegida por un mutex o lock. Para crear un monitor en Java, hay que marcar un bloque de código con la palabra synchronized.

Añadir synchronized a un método significará que:

- Hemos creado un monitor asociado al objeto.
- Solo un hilo puede ejecutar el método synchronized de ese objeto a la vez.
- Los hilos que necesitan acceder a ese método synchronized permanecerán bloqueados y en espera.
- Cuando el hilo finaliza la ejecución del método synchronized, los hilos en espera de poder ejecutarlo se desbloquearán. El planificador Java seleccionará a uno de ellos.

4. *Monitores. Segmentos de código synchronized.*

Hay casos en los que no se puede, o no interesa sincronizar un método. La forma de resolver esta situación es poner las llamadas a los métodos que se quieren sincronizar dentro de segmentos sincronizados de la siguiente forma: synchronized (objeto){ // sentencias segmento; }. En este caso el funcionamiento es el siguiente:

- El objeto que se pasa al segmento, es el objeto donde está el método que se quiere sincronizar.
- Dentro del segmento se hará la llamada al método que se quiere sincronizar.
- El hilo que entra en el segmento declarado synchronized se hará con el monitor del objeto, si está libre, o se bloqueará en espera de que quede libre. El monitor se libera al salir el hilo del segmento de código synchronized.
- Solo un hilo puede ejecutar el segmento synchronized a la vez.

5. *Comunicación entre hilos con métodos de java.lang.Object.*

La comunicación entre hilos la podemos ver como un mecanismo de auto-sincronización, que consiste en logar que un hilo actúe solo cuando otro ha concluido cierta actividad (y viceversa).

Java soporta comunicación entre hilos mediante los siguientes métodos de la clase java.lang.Object:

- **wait():** Detiene el hilo (pasa a "no ejecutable"), el cual no se reanudará hasta que otro hilo notifique que ha ocurrido lo esperado.
  - **wait(long tiempo):** Como el caso anterior, solo que ahora el hilo también puede reanudarse(pasar a "ejecutable) si ha concluido el tiempo pasado como parámetro.
- **notify():** Notifica a uno de los hilos puestos en espera para el mismo objeto, que ya puede continuar.
  - **notifyAll():** Notifica a todos los hilos puestos en espera para el mismo objeto que ya pueden continuar.

6. *El problema del interbloqueo (deadlock).*

El interbloqueo o bloqueo mutuo (deadlock) consiste en que uno a más hilos, se bloquean o esperan indefinidamente.

¿Cómo se llega a una situación de interbloqueo?

- Porque cada hilo espera a que le llegue un aviso de otro hilo que nunca le llega.
- Porque todos los hilos, de forma circular, esperan para acceder a un recurso.

Otro problema, menos frecuente, es la inanición (starvation), que consiste en que un hilo es desestimado para su ejecución. Se produce cuando un hilo no puede tener acceso regular a los recursos compartidos y no puede avanzar, quedando bloqueado. Esto puede ocurrir porque el hilo nunca es seleccionado para su procesamiento o bien porque otros hilos que compiten por el mismo recurso se lo impiden.