

EXPLICACIÓN PROBLEMA LECTORES – ESCRITORES



Se trata de una serie de lectores (hilos lectores) y escritores (hilos escritores) que acceden a una Base de Datos (BD).

Varios lectores pueden estar a la vez utilizando la BD, pues ninguno de ellos modifica nada, sólo lee.

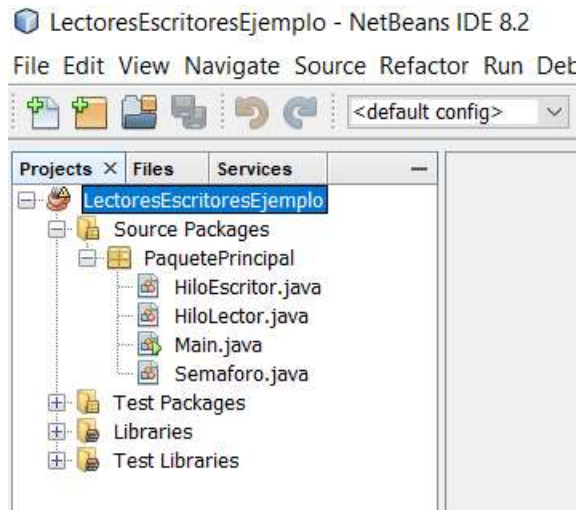
Cuando un escritor accede a la BD, ésta debe estar libre, es decir, no puede haber ningún lector o escritor utilizándola, ya que el escritor modifica los datos.

Casos que se pueden presentar

- Llega un lector:
 - Si hay uno o varios lectores en la BD, el lector podrá acceder.
 - Si hay un escritor, entonces el lector deberá esperar a que el escritor acabe con su trabajo para entrar.
- Llega un escritor:
 - Si hay uno o varios lectores en la BD, el escritor deberá esperar a que todos los lectores que están en la BD terminen.
 - Si hay un escritor, el que quiere entrar deberá esperar a que el que hay en la BD termine su trabajo.

Clases que creamos

- **Clase Semaforo.** Indicará el estado en el que se encuentra la BD. Según el estado del semáforo, los lectores o escritores podrán o no acceder a la BD para realizar su trabajo.
- **Clase HiloLector.** Hilo que implementa un lector que intenta acceder a la BD para leer.
- **Clase HiloEscritor.** Hilo que implementa un escritor que intenta acceder a la BD para escribir.
- **Clase Main.** Crea e inicia los hilos permitiendo aplicar el semáforo al acceso de los lectores y escritores a la BD.



Clase Semaforo

Estados:

Según el estado del semáforo, puede ocurrir:

- Estado **LIBRE (0)**, indica que no hay nadie ni escribiendo ni leyendo en la BD. Por tanto, cualquier lector o escritor podrá acceder a ella. Un escritor deberá esperar a que la BD pase por este estado antes de entrar a escribir.
- Estado **CON_LECTORES (1)**, indica que hay lectores en la BD, por lo que ningún escritor podrá acceder a ella, pero sí otros lectores.
- Estado **CON_ESCRITOR (2)**, indica que hay un escritor en la BD, por lo que nadie (ni lectores, ni escritores) podrán acceder a ella.

```
public class Semaforo {  
  
    public final static int LIBRE = 0;  
    //indica que no hay lectores leyendo, ni ningún escritor escribiendo.  
    //En este estado pueden entrar lectores a leer, o un escritor a escribir  
    public final static int CON_LECTORES = 1;  
    //constante que indica que hay lectores leyendo. Puede entrar un nuevo  
    //lector a leer, pero no puede entrar ningún escritor a escribir  
    public final static int CON_ESCRITOR = 2;  
    //constante que indica que hay escritores escribiendo. En este estado, no  
    //puede entrar ningún lector a leer, ni ningún escritor a escribir  
    private int estado = LIBRE;  
    //estado del semáforo (inicialmente: libre)  
    private int tLectores = 0;  
    //número de lectores (inicialmente: ninguno)
```

Método accesoLeer():

El método **accesoLeer()** controla la entrada de lectores a la BD.

Observa que, cuando finalice la espera invocada con wait(), el lector entra a leer en la Bd.

```
/******  
 * método que da acceso a la lectura de datos  
 */  
public synchronized void accesoLeer() {  
    //método sincronizado. Sólo un hilo lo usa a la vez  
    String nombre = Thread.currentThread().getName();  
    //guarda el nombre del hilo que se hace con el método  
    if (estado == LIBRE) {  
        //BD sin lectores ni escritores. Puede entrar a leer  
        System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " entra a leer.");  
        //mensaje para comprobar el funcionamiento  
        estado = CON_LECTORES;  
        //cambia estado, ya hay lector  
    } else if (estado != CON_LECTORES) {  
        //si no está libre, ni con lectores  
        while (estado == CON_ESCRITOR) {  
            try {  
                System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " trata de leer.ESPERA");  
                //mensaje para comprobar el funcionamiento  
                wait();  
                //pone en espera al hilo que intenta leer datos  
            } catch (InterruptedException e) {  
                System.out.println(e);  
            }  
        }  
        System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " entra a leer.");  
        //mensaje para comprobar el funcionamiento  
        estado = CON_LECTORES;  
        //cambia estado, ya hay lector  
    } else { //en este punto el estado es CON_LECTORES  
        System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " entra a leer.");  
        //mensaje para comprobar funcionamiento  
    }  
    tLectores++;  
    //otro lector más  
    System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " Leyendo.....");  
    //mensaje para comprobar funcionamiento  
}
```

Método accesoEscribir():

El método **accesoEscribir()** controla la entrada de escritores en la BD.

Observa que, cuando finalice la espera invocada con wait(), el escritor entra a escribir en la BD.

```
/******  
 * método que da acceso para escribir datos si el estado de la BD lo permite  
 */  
public synchronized void accesoEscribir() {  
    String nombre = Thread.currentThread().getName();  
    //guarda el nombre del hilo que se hace con el método  
    if (estado == LIBRE) {  
        //sin lectores ni escritores  
        System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " entra a escribir.");  
        //mensaje para comprobar el funcionamiento  
        estado = CON_ESCRITOR;  
        //cambia estado  
    } else { //si no está libre  
        while (estado != LIBRE) {  
            //mientras BD está ocupada con lectores, o con un escritor  
            try {  
                System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " trata de escribir.ESPERA");  
                //mensaje para comprobar funcionamiento  
                wait();  
                //pone en espera al hilo que intenta escribir datos  
            } catch (InterruptedException e) {  
                System.out.println(e);  
            }  
        }  
        // el estado ahora es LIBRE  
        System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " entra a escribir.");  
        //mensaje para comprobar el funcionamiento  
        estado = CON_ESCRITOR;  
        //cambia estado  
    }  
    System.out.println("BD:" + estado + " " + tLectores + "L " + nombre + " Escribiendo..");  
    //mensaje para comprobar el funcionamiento  
}
```

Métodos escrituraFinalizada() y lecturaFinalizada():

Observa que, tanto en el método **lecturaFinalizada()** como en el **escrituraFinalizada()** la notificación **notify()** a los hilos que esperan sólo se hace cuando la BD está LIBRE.

```
/******  
 * método que invoca un HiloEscritor al terminar de escribir, para  
 * actualizar el estado del semáforo y en su caso notificarlo a los hilos  
 * en espera.  
 * Por supuesto, no se permite que dos hilos ejecuten estas instrucciones  
 * a la vez  
 */  
public synchronized void escrituraFinalizada() {  
    estado = LIBRE;  
    //cambia estado  
    System.out.println(Thread.currentThread().getName() + ": Ya ha escrito");  
    //mensaje para comprobar el funcionamiento  
    notify();  
    //notifica a los hilos en espera que ya ha finalizado  
}  
  
/******  
 * método que que invoca un HiloLector cuando termina de escribir, para  
 * actualizar el estado del semáforo y en su caso notificarlo a los hilos  
 * en espera  
 *  
 * por supuesto, no se permite que dos hilos ejecuten estas instrucciones  
 * a la vez  
 */  
public synchronized void lecturaFinalizada() {  
    System.out.println(Thread.currentThread().getName() + ": Ya ha leído");  
    //mensaje para comprobar el funcionamiento  
    tLectores--;  
    //un lector menos leyendo  
    if (tLectores == 0) {  
        //no hay lectores en la BD  
        estado = LIBRE;  
        //cambia el estado  
        notify();  
        //notifica a los hilos en espera que ya ha finalizado  
    }  
}
```

Clase HiloEscritor

Constructor y método run():

Observa que al constructor del hilo escritor se le pasa un semáforo.

El hilo invoca los métodos **accesoEscribir()** y **escrituraFinalizada()**.

```
public HiloEscritor(String nombre, Semaforo s) {
    this.setName(nombre);
    this.semaforo = s;
}
@Override
public void run() {
    //método con el comportamiento del hilo
    System.out.println(getName() + ": Intentando escribir");
    //mensaje para la Salida y comprobar funcionamiento
    semaforo.accesoEscribir();
    //el hilo ha escrito
    try {
        sleep((int) (Math.random() * 50));
        //duerme el hilo un tiempo aleatorio antes de comunicar el fin de
        //la lectura, para dar ocasión de que los demás hilos hagan
        //intentos fallidos de lectura/escritura y comprobar funcionamiento
    } catch (InterruptedException e) {
        System.out.println(e);
    }
    semaforo.escrituraFinalizada();
    //comunica al semáforo la finalización de la escritura
}
```

Clase HiloLector

Constructor y método run():

Observa que al constructor del hilo se le pasa un objeto Semáforo.
El hilo invoca los métodos **accesoLeer()** y **lecturaFinalizada()**.

```
public HiloLector(String nombre, Semaforo s) {
    this.setName(nombre);
    this.semaforo = s;
}
/*****
 * el método run() del hilo que lee los datos */
@Override
public void run() {
    System.out.println(getName() + ": Intentando leer");
    //mensaje de salida para comprobar el funcionamiento
    semaforo.accesoLeer();
    //el hilo ha leído
    try {
        sleep((int) (Math.random() * 50));
    } catch (InterruptedException e) {
        System.out.println(e);
    }
    //duerme al hilo antes de que éste comunique que ha finalizado, para
    //poder ver accesos fallidos, con fines de comprobar funcionamiento
    semaforo.lecturaFinalizada();
    //comunica al semáforo la finalización de la lectura
}
```

Clase Main

Método main():

Se crean e inician 5 hilos lectores y 2 hilos escritores.

```
public static void main(String args[]) {
    Semaforo smfro = new Semaforo();
    //semáforo de control

    //pone 5 lectores a leer y 2 escritores a escribir, controlados por
    //el mismo semáforo
    for (int i = 1; i <= 5; i++) {
        new HiloLector("Lector" + i, smfro).start();
    }

    for (int i = 1; i <= 2; i++) {
        new HiloEscritor("Escritor" + i, smfro).start();
    }
}
```

Resultado de la ejecución

run:

Lector1: Intentando leer

Lector5: Intentando leer

BD:0 0L Lector1 entra a leer.

BD:1 1L Lector1 Leyendo.....

Lector4: Intentando leer

Lector3: Intentando leer

Lector2: Intentando leer

BD:1 1L Lector5 entra a leer.

BD:1 2L Lector5 Leyendo.....

BD:1 2L Lector2 entra a leer.

BD:1 3L Lector2 Leyendo.....

BD:1 3L Lector3 entra a leer.

BD:1 4L Lector3 Leyendo.....

BD:1 4L Lector4 entra a leer.

BD:1 5L Lector4 Leyendo.....

Lector4: Ya ha leído

Lector5: Ya ha leído

Lector1: Ya ha leído

Lector2: Ya ha leído

Lector3: Ya ha leído

Escritor1: Intentando escribir

Escritor2: Intentando escribir

BD:0 0L Escritor1 entra a escribir.

BD:2 0L Escritor1 Escribiendo..

Escritor1: Ya ha escrito

BD:0 0L Escritor2 entra a escribir.

BD:2 0L Escritor2 Escribiendo..

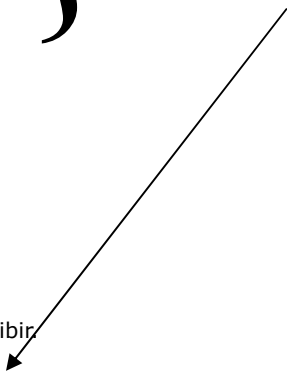
Escritor2: Ya ha escrito

BUILD SUCCESSFUL (total time: 0 seconds)

Observa que puede haber varios lectores leyendo a la vez



Los escritores sólo pueden acceder a escribir cuando no hay nadie leyendo



Observaciones

Controlar otras situaciones:

¿Qué pasa cuando hay varios lectores activos y un escritor desea escribir?

- Si permitimos que entren todos los lectores y compartan datos entre ellos, no permitiendo que entre el escritor, entonces éste puede estar esperando indefinidamente.
- Si damos mayor prioridad a los escritores entonces tendremos el mismo problema, ya que puede que lleguen muchos escritores y por tanto dejen en espera a los lectores indefinidamente.

¿Quién nos interesa que tenga preferencia?

¿Qué pasa si tenemos muchos escritores esperando?

¿Y si tenemos muchos lectores esperando?

Observa que es un tema complicado y que nos da una idea de los problemas que nos podemos encontrar cuando desarrollamos aplicaciones multihilo.