

Métodos Avanzados de Programación Científica y Computación

M^a Luisa Díez Platas

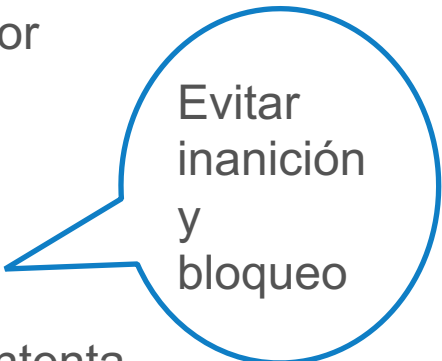
Tema 9. Sincronización (I)

¿Cómo estudiar este tema?

IDEAS CLAVE	LO + RECOMENDADO	+ INFORMACIÓN
¿Cómo estudiar este tema? Sincronización Mecanismos de bloqueo Mecanismos de comunicación Monitores Bibliotecas de Java para concurrencia	No dejes de leer... <code>wait y notify</code> Monitor No dejes de ver... TV Ejemplo hilos y monitores. Ejemplo del barbero	A fondo Interbloqueo Métodos y bloques <code>synchronized</code> Biblioteca <code>java.util.concurrent</code>

Sincronización

Acción que permite el acceso a recursos compartidos por varios hilos, asegurando la consistencia en el acceso
→ estados coherentes



Evitar
inanición
y
bloqueo

Conflictos de lectura/escritura

- un hilo consulta estado de un objeto a la vez que otro intenta modificarlo.
- No en todas las ocasiones (carrera).

Conflictos de escritura/escritura

- Dos hilos tratan de modificar el estado de un objeto.

Sincronización

Acción que permite el acceso a recursos compartidos por varios hilos, asegurando la consistencia en el acceso
→ **estados coherentes**


Problema acceso
concurrente de
varios
hilos → **sincronización**

Soluciones

- Hardware
- Espera activa
- Semáforos y monitores

Mecanismos

- Bloqueo
- Comunicación
- Monitores



```
class Contador {  
    private int contador = 0;  
  
    public void incrementar() {  
        contador++;  
    }  
  
    public void decrementar() {  
        contador--;  
    }  
  
    public int valor() {  
        return contador;  
    }  
}
```

- 1.El hilo1 obtiene el valor del contador. (0)
- 2.El hilo2 obtiene el valor del contador. (0)
- 3.El hilo 1 lo incrementa (1)
- 4.El hilo 2 lo decrementa (-1)

Modos de bloqueo para la sincronización

Todo objeto tiene asociado un cerrojo

Métodos sincronizados

```
class Contador {  
    private int contador = 0;  
  
    public synchronized void incrementar() {  
        contador++;  
    }  
  
    public synchronized void decrementar() {  
        contador--;  
    }  
  
    public synchronized int valor() {  
        return contador;  
    }  
}
```

¡un constructor
no puede ser
sincronizado! y
no debe llamar
a métodos que
modifiquen
estado

Si el método es
estático se
adquiere el
cerrojo de la
clase

No es posible que dos invocaciones de métodos sincronizados sobre el mismo objeto se intercalen → **el hilo adquiere el cerrojo (cerrado)**

Modos de bloqueo para la sincronización

Todo objeto tiene asociado un cerrojo

Bloques sincronizados

```
public synchronized void añadirDato(int dato) {  
  
    ultimo = dato;  
    contadorDatos++;  
  
    listaDatos.add(ultimo);  
}
```

Problemas de falta de vitalidad

```
public void añadirDato(int dato) {  
    synchronized(this) {  
        ultimo = dato;  
        contadorDatos++;  
    }  
    listaDatos.add(ultimo);  
}
```

No es posible que dos invocaciones de métodos sincronizados sobre el mismo objeto se intercalen → **el hilo adquiere el cerrojo (cerrado)**

Modos de bloqueo para la sincronización

Todo objeto tiene asociado un cerrojo

Bloques sincronizados

Los contadores se pueden usar por separado

```
class Contador {  
    private int contador1 = 0;  
    private int contador2 = 0;  
  
    private Object cerrojo1= new Object();  
    private Object cerrojo2= new Object();  
  
    public void incrementar1() {  
        synchronized (cerrojo1){  
            contador1++;  
        }  
    }  
  
    public void incrementar2() {  
        synchronized (cerrojo2){  
            contador2++;  
        }  
    }  
}
```

Interbloqueo

```
Celda.class Celda{  
    private long valor;  
    synchronized long dameValor() {return valor;}  
    synchronized void ponValor(long v) {valor=v;}
```

```
    synchronized void intercambiarValor(Celda otra) {  
        long t=dameValor();  
        long v=otra.dameValor()  
        ponValor(v);  
        otra.ponValor(t)  
    }  
};
```

Adquiere cerrojos en
múltiples objetos

Problema de interbloqueo:

Invocación casi simultanea:

un hilo a `a.intercambiarValor(b)`

otro hilo a `b.intercambiarValor(a)`

Hilo 1	Hilo 2
Invoca <code>a.intercambiarValor(b)</code> , adquiere el cerrojo de a	
Ejecuta <code>dameValor()</code>	Invoca <code>b.intercambiarValor(a)</code> , adquiere el cerrojo de b
Se bloquea en <code>otra.dameValor</code> por que el cerrojo de b lo tiene el otro hilo	Ejecuta <code>dameValor()</code>
	Se bloquea en <code>otra.dameValor</code> por que el cerrojo de a lo tiene el otro hilo

(Lea, D. 2001)

Modos de comunicación. Monitores.

Clase con al menos un método `synchronized`

- `Wait()` libera el cerrojo del objeto y se bloquea el hilo.
- `Notify()` pasa a listo a un hilo que este en espera para adquirir el cerrojo de un monitor.
- `NotifyAll()` pasa a listo a todos los hilos que este en espera para adquirir el cerrojo del monitor.

La comunicación entre hilos se implementa mediante los métodos `notify()`, `notifyAll()`

Modos de comunicación- Monitores

La comunicación entre hilos se implementa mediante los métodos `wait()`, `notify()`, `notifyAll()`

```
public synchronized void insertarElemento (int elemento)
{
    while (cima_ >= maximo_ - 1) {
        try {
            System.out.println ("No puedo añadir
nada");
            wait ();
            System.out.println ("Ya puedo volver
a añadir");
        }
        catch (InterruptedException e) {
            System.err.println ("Se ha producido
un error: " + e.toString ());
        } // catch
    } // while

    cima_++;
    elemento_ [cima_] = elemento;

    notifyAll ();
} // insertarElemento
```

El hilo se pone en cola de **espera** y libera el cerrojo

Todos los hilos de la cola pasan a **ejecutable**

```
private int [] elemento_ = new int [CAPACIDAD_MAXIMA__];
private int cima_;

private int maximo_;
```

Modos de comunicación- Monitores

La comunicación entre hilos se implementa mediante los métodos `wait()`, `notify()`, `notifyAll()`

```
public synchronized int sacarElemento () {
    int elem;

    while (cima_ == -1) {
        try {
            System.out.println ("No puedo sacar
nada");
            wait ();
            System.out.println ("Ya puedo volver
a sacar");
        }
        catch (InterruptedException e) {
            System.err.println ("Se ha producido
un error: " + e.toString ());
        } // catch
    } // while

    elem = elemento_ [cima_];
    cima_--;

    notifyAll ();

    return (elem);
} // sacarElemento
```

El hilo se pone en cola de **espera** y libera el cerrojo

Todos los hilos de la cola pasan a **ejecutable**

Problema de los monitores anidados

Libera el cerrojo del objeto de tipo interno pero no de otros

```
public class Interna {
    protected boolean cond=false;

    public synchronized void esperarCond(){
        while (!cond)
            try{
                wait();
            }catch (InterruptedException e){}
    }

    public synchronized void
conLiberacion(boolean c){
        cond=c;
        notifyAll();
    }
}
```

Agregación exclusiva

Ambas clases
totalmente
sincronizadas

```
public class Externa {
    protected Interna interna=new Interna();

    public synchronized void esperar(){
        interna.esperarCond();
    }

    public synchronized void conLiberacion(boolean c){
        interna.conLiberacion(c);
    }
}
```

Bloqueo(Lockout)

Solución: Si el objeto interno debe ser exclusivo, no sincronizar la clase externa

- ▶ Lea, D. (2001). *Programación concurrente en Java*. Pearson Educación.

UNIVERSIDAD
INTERNACIONAL
DE LA RIOJA

unir

www.unir.net