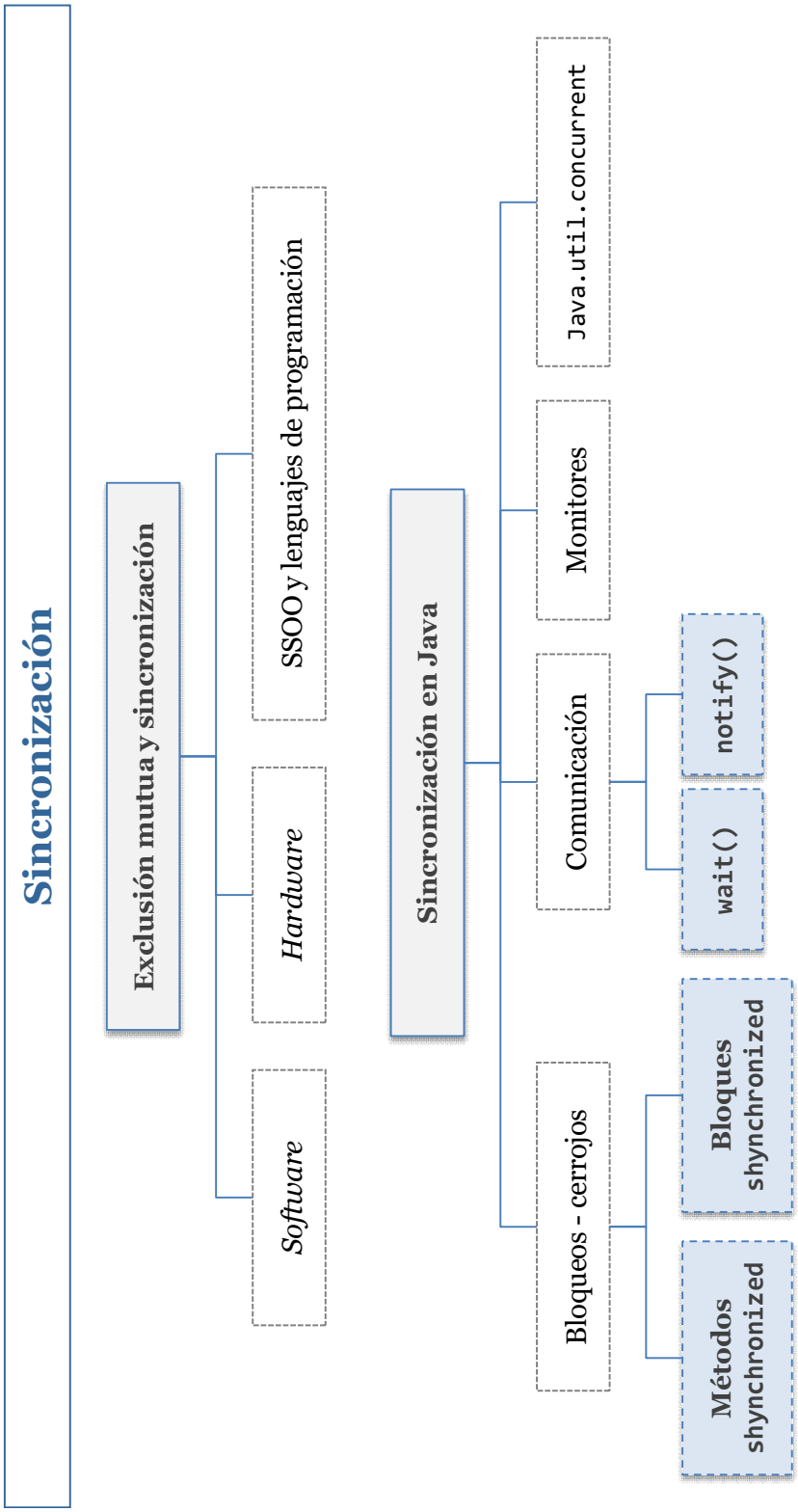
Sincronización

TEMA 9

Esquema



Ideas clave

9.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee el **capítulo 2 (páginas 19-37)** del siguiente libro, disponible en la Biblioteca Virtual de UNIR:

Espinosa, A. R., Argente, E. & Muñoz, F. D. (2013). *Concurrencia y sistemas distribuidos*. Valencia: Editorial de la Universidad Politécnica de Valencia.

Además, deberás leer el capítulo 6 **(páginas 209-214)** del siguiente libro, disponible en el aula virtual en virtud del artículo 32.4 de la Ley de Propiedad Intelectual:

Sznajdleder, P. A. (2013). *Java a Fondo: estudio del lenguaje y desarrollo de aplicaciones* (2ª ed.). Buenos Aires: Alfaomega Grupo Editor.

Para comprobar si has comprendido los conceptos puedes realizar el test de autoevaluación del tema.

En este tema vamos a ver en profundidad cómo se sincronizan los hilos en Java mediante diferentes mecanismos de sincronización.

9.2. Sincronización

El **problema de sincronización de hilos** surge cuando varios hilos intentan acceder a los mismos datos y/o a los mismos recursos, para lo que se debe establecer un orden. Para resolver el problema de la exclusión mutua y la sincronización se pueden adoptar distintos tipos de soluciones:

Soluciones por *hardware*	Soluciones por *software*	Otras soluciones
Inhabilitación de interrupciones, instrucciones especiales de máquina	Modelos basados en «espera activa» (repetir un ciclo que no hace nada un número de veces)	Soportados por el sistema operativo o por los lenguajes de programación: semáforos, monitores, etc.

En este tema vamos a estudiar los mecanismos de sincronización en Java:

- Mecanismos de bloqueo.
- Mecanismos de comunicación.
- Monitores.

9.3. Mecanismos de bloqueo

En Java todo objeto derivado de la clase `Object` tiene asociado un **`lock` o cerrojo**.

Un cerrojo tiene **dos estados**: abierto y cerrado. Uno o más hilos pueden cooperar por medio de `locks` o cerrojos, de forma que un hilo que pida un cerrojo que se encuentre cerrado por otro hilo, se detendrá hasta que el otro hilo libere el cerrojo.

Métodos `synchronized`

Un cerrojo puede abarcar a todo un método. Para ello se declara al método como `synchronized`.

Ejemplo:

```
syncrhonized tipo MiMétodo()
```

Solo un hilo es el propietario del cerrojo. Solo el hilo propietario del cerrojo puede ejecutar un código `synchronized`.

Si un hilo tiene el cerrojo de un método `synchronized`, ningún otro hilo puede ejecutar otro método `synchronized` del mismo objeto y se queda suspendido hasta que el hilo que tiene el cerrojo lo libere.

Bloques `synchronized`

Los bloques `synchronized` se utilizan para implementar las secciones críticas.

```
synchronized {
        Conjunto de sentencias
}
```


Java permite declarar una sección crítica sobre un objeto:

```
      synchronized (objeto) {
            Conjunto de sentencias
}
```

9.4. Mecanismos de comunicación

La **comunicación** es soportada en la clase `Object` mediante los métodos `wait()` y `notify()`.

La ejecución de `wait()` supone que el hilo queda a la espera de que suceda algo para poder continuar; el hilo volverá a la situación de «ejecutable» cuando se reciba un `notify()`.

Métodos:

- **`void wait()`**
- **`void wait(long timeout)`**: indica el tiempo máximo que el hilo debe estar parado.
- **`void wait(long timeout, int nanos)`**: indica el tiempo máximo que el hilo debe estar parado.
- **`void notify() //:`** lanza una señal indicando al sistema que puede activar uno de los hilos que se encuentran bloqueados esperando para acceder a ese objeto.
- **`void notifyAll()//:`** lanza una señal a todos los hilos que están esperando la liberación del objeto.

Las llamadas a estos métodos han de estar en un **método sincronizado**, ya que de otra forma se obtendrá una excepción en tiempo de ejecución.

9.5. Monitores

Monitor

Es un objeto cuya clase tiene al menos un método `synchronized`.

En un instante dado, solo un hilo podrá ejecutar un método `synchronized` de un monitor, el resto de hilos que llamen a un método `synchronized` del monitor quedarán suspendidos, hasta que el hilo que posee el cerrojo lo libere. La sincronización se realiza por cada instancia del monitor.

Los hilos que esperan para entrar en el objeto y ejecutar uno de sus métodos `synchronized` forman una cola.

Los métodos `wait()`, `notify()` y `notifyAll()` permiten controlar los accesos al monitor:

- **La llamada a `wait()`** dentro de un método del monitor, provoca que se libere el cerrojo sobre el monitor y que se bloquee al hilo que llamó al método.

- **La llamada a `notify()`** dentro de un método del monitor, provoca que un hilo en espera vuelva a estar listo y pueda intentar obtener el cerrojo del monitor.

- **La llamada a `notifyAll()`** dentro de un método del monitor, provoca que todos los hilos que esperaban por el cerrojo puedan intentar obtenerlo.

Ejemplo:

```
public class Main {

    public static void main(String args[]) {

        Monitor monitor = new Monitor();
        MiClase hola = new MiClase("hola", 0, monitor);
        hola.start();
        MiClase mundo = new MiClase("mundo", 1, monitor);
        mundo.start();
    }
}
```



```
class MiClase extends Thread {

    private Monitor monitor;
    private String texto;
    private int orden;

    public MiClase (String texto, int orden, Monitor monitor) {
        this.texto = texto;
        this.monitor = monitor;
        this.orden = orden;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                monitor.escribir(texto + " ", orden);
                sleep(100);
            } catch (InterruptedException ex);
        }
    }
}

class Monitor {

    private int actual = 0;

    public synchronized void escribir(String texto, int orden) {

        while (orden != actual) {
            try {
                wait();
            } catch (InterruptedException ex);
        }

        System.out.print(texto);
        actual = (actual + 1) % 2;
        notifyAll();
    }
}
```


9.6. Bibliotecas de Java para concurrencia

Existen varias bibliotecas que proporcionan clases e interfaces que resuelven muchos problemas comunes de concurrencia:

- `java.util.concurrent`
- `java.util.concurrent.atomic`
- `java.util.concurrent.lock`

En estas bibliotecas encontramos:

- Cerrojos de exclusión mutua (`Lock`)
- Variables atómicas
- Colecciones concurrentes
- Semáforos
- Barreras

Lo + recomendado

No dejes de leer…

`Wait` y `notify`

En este artículo veremos cómo utilizar las sentencias `wait` y `notify` para la sincronización de hilos.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://www.arquitecturajava.com/java-wait-notify-y-threads/

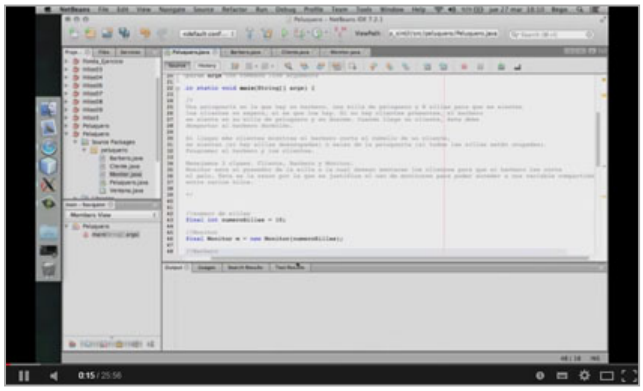
Monitor

En este artículo veremos cómo interactúa un programa con dos procesos (`threads`) independientes que se van intercalando según el procesador los va ejecutando. También se introduce el concepto de monitor para controlar procesos comunes.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://www.webtutoriales.com/articulos/programacion-concurrente-procesos-en-java

No dejes de ver...

Ejemplo hilos y monitores. Ejemplo del barbero



Vídeo en el que se explica la implementación del problema del barbero en Java mediante monitores.

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=r0F6xv304Rk

+ Información

A fondo

Interbloqueo

En esta parte del tutorial elaborado por Jesús Luna Quiroga sobre concurrencia en Java, encontramos interesantes ejemplos de interbloqueo.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://jrlq.blogspot.com.es/2013/05/concurrencia-en-java-parte-4.html

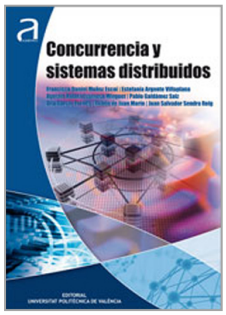
Métodos y bloques `synchronized`

El siguiente artículo se explica la diferencia entre métodos y bloques `synchronized`.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: https://edgomezf.wordpress.com/2013/10/09/diferencia-entre-sincronizacion-de-bloque-y-metodo-en-java/

Biblioteca `java.util.concurrent`

Espinosa, A. R., Argente, E. & Muñoz, F. D. (2013). *Concurrencia y sistemas distribuidos*. Valencia: Editorial de la Universidad Politécnica de Valencia.



En el capítulo 5 de este libro se explica la biblioteca `java.util.concurrent`.

El libro está disponible en la Biblioteca Virtual de UNIR.

Test

1. Los cerrojos en Java:

A. No existen.

B. Se implementan mediante los métodos `sleep` y `resume`.

C. Se implementan mediante los métodos `wait` y `notify`.

D. Se implementan mediante métodos y bloques `synchronized`.

2. Si un hilo posee el cerrojo de un método:

A. Ningún otro hilo puede ejecutarse.

B. Ningún otro hilo puede ejecutar otro método del objeto.

C. Ningún otro hilo puede ejecutar otro método `synchronized` del objeto.

D. Impide la creación de nuevos hilos.

3. Un hilo que ha llamado a `wait()`:

A. Es despertado por un hilo que llama a `run()`.

B. Es despertado por un hilo que llama a `notify()`.

C. Es despertado por un hilo que llama a `yiedl()`.

D. Un hilo demonio.

4. Un hilo que ejecuta un método `synchronized`:

A. Si no puede continuar, llamará al método `wait()`.

B. Si no puede continuar, bloqueará al resto de hilos.

C. Si no puede continuar, llamará al método `stop()`.

D. Si no puede continuar, llamará al método `isAlive()`.

5. Las llamadas a métodos `wait()` y `notify()`:

A. Deben estar en un manejador de excepciones.

B. Deben estar dentro de métodos `synchronized`.

C. Deben estar en el programa principal.

D. Deben estar en el método `run()` del objeto.

6. El método `notify()`:

A. Bloquea un hilo.

B. Desbloquea todos los hilos.

C. Desbloquea un hilo.

D. Pone un hilo a la espera de un suceso.

7. El método `notifyAll()`:

A. Bloquea todos los hilos.

B. Desbloquea todos los hilos.

C. Desbloquea un hilo.

D. Pone a todos los hilos a la espera de un suceso.

8. Un monitor es:

A. Una instancia de una clase con atributos `synchronized`.

B. Una instancia de una clase con al menos un atributo `synchronized`.

C. Una instancia de una clase con todos sus métodos `synchronized`.

D. Una instancia de una clase con al menos un método `synchronized`.

9. Un monitor:

A. Es un objeto que implementa del acceso en exclusión mutua a sus métodos.

B. Es una clase que implementa la interfaz `Runnable`.

C. Es una clase que hereda de la clase `Thread`.

D. Es una instancia de la clase `Thread`.

10. El método `notify()`:

A. Permite indicar que hilo se va a desbloquear.

B. No permite indicar que hilo se va a desbloquear.

C. Debe ser llamado por un hilo no propietario del cerrojo del monitor.

D. Debe ser llamado el hilo principal.

