20151028

dit
UPM

Programación concurrente — Sincronización condicional

Juan Antonio de la Puente <jpuente@dit.upm.es>

ETSIT UPM

Referencias

- Scott Oaks & Henry Wong
  Java Threads
  O'Reilly Media; 3rd ed (2004)

- Kathy Sierra & Bert Bates
  Head First Java, ch. 15
  O'Reilly Media; 2nd ed (2005)

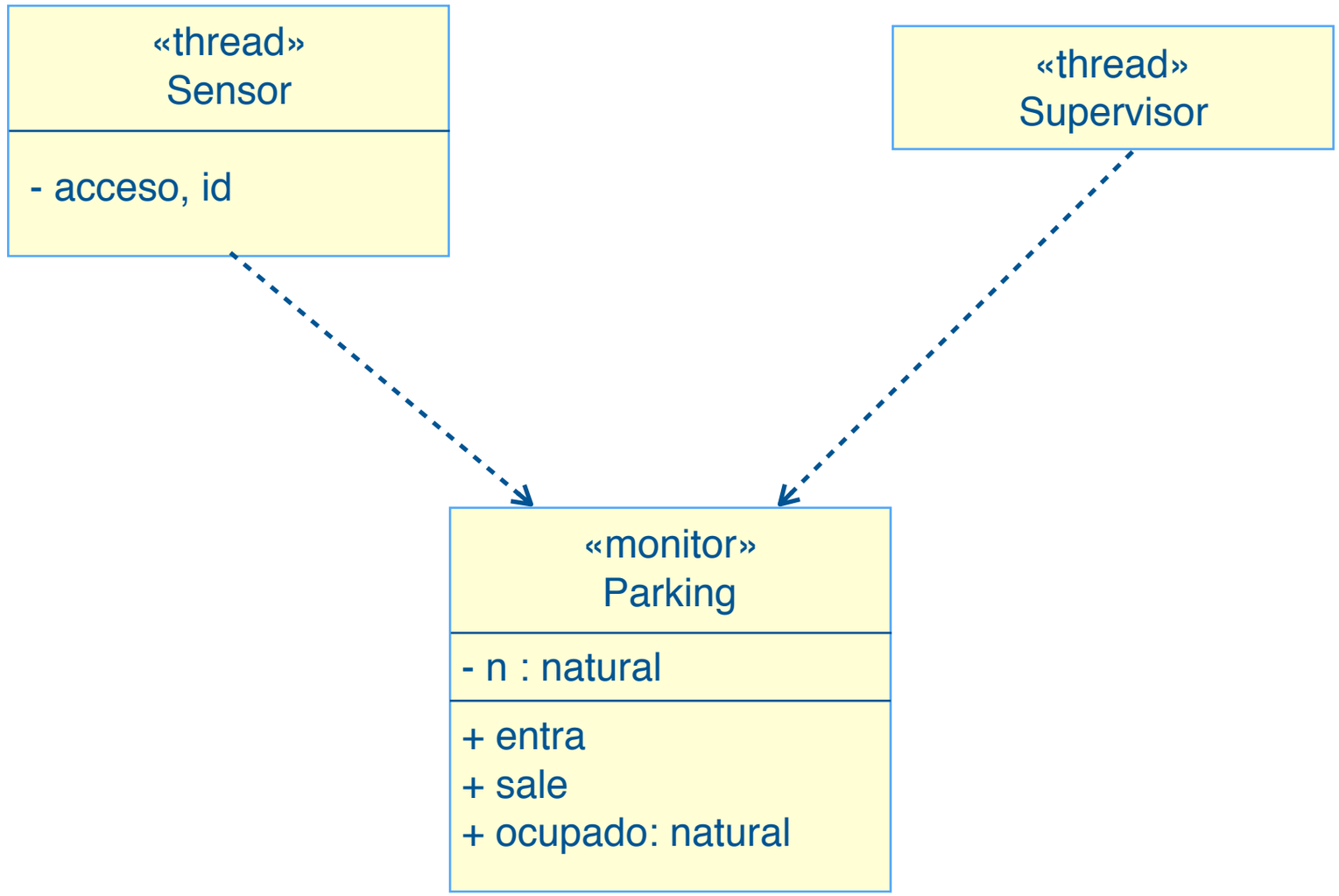
Sincronización condicional

Sincronización condicional

- A veces una hebra tiene que esperar hasta que se cumpla una determinada condición
  - suspende su ejecución hasta que esto ocurra
- Otra hebra hace que se cumpla la condición
  - avisa al que estaba esperando
  - éste puede reanudar su ejecución
- Este tipo de sincronización se llama sincronización condicional
  - sincronización: el avance de una hebra depende de lo que haga otra hebra

Ejemplo: gestión de un estacionamiento

- Las hebras de las clase *Sensor* avisan cuando entra o sale un coche

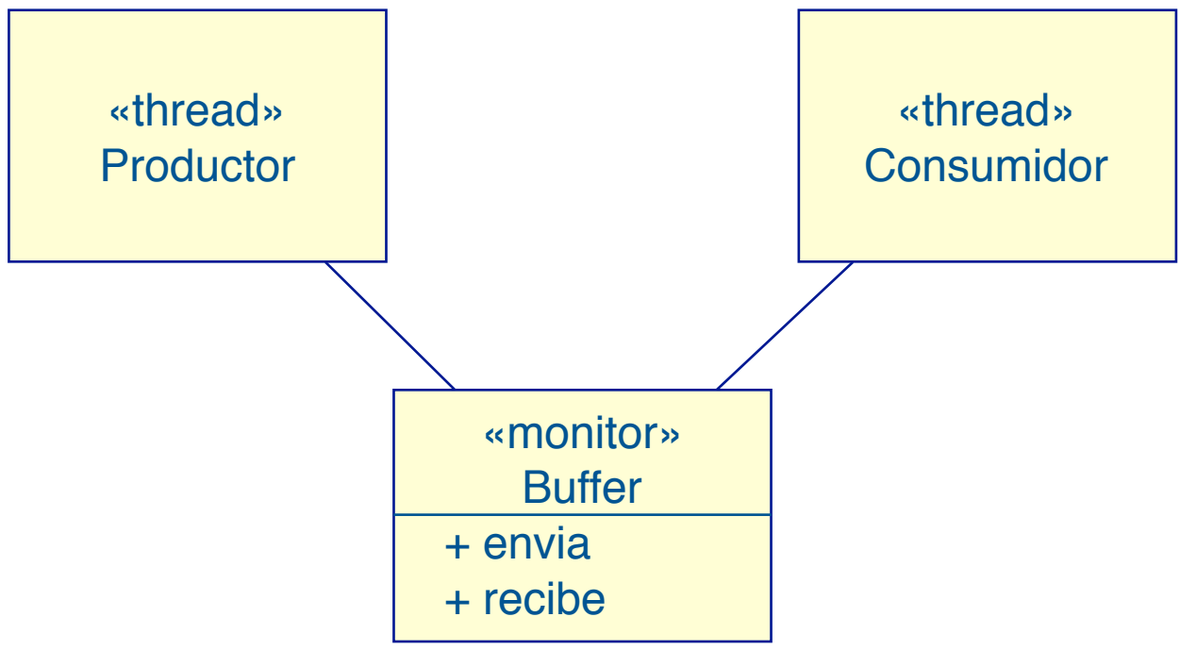


Sincronización

- No deben entrar coches si el estacionamiento está lleno
  - no se puede entrar si n >= capacidad
  - si un sensor llama a *entra* cuando n >= capacidad, debe suspender su ejecución hasta que haya sitio
    - está implícito que en ese caso no se abre la barrera hasta que se pueda continuar

Ejemplo: productor y consumidor

- Una hebra produce elementos de un cierto tipo
- Otra los consume
- Cada una avanza con un ritmo diferente
- Se usa un almacenamiento intermedio (*buffer*)
  - el productor los va poniendo en el buffer según los produce
  - el consumidor los saca del buffer cuando los necesita



Sincronización

- No se pueden extraer datos si el *buffer* está vacío
  - si cuando el consumidor invoca *recibe* no hay datos en el *buffer*, debe suspender su ejecución hasta que los haya
  - es el productor el que los pone
- No se pueden añadir datos si el *buffer* está lleno
  - si cuando el productor invoca *envía* el *buffer* está lleno, debe suspender su ejecución hasta que haya sitio
  - es el consumidor el que hace sitio al extraer datos

Esperar y avisar

Espera condicional

- El método `wait()` suspende la ejecución de la hebra que lo invoca
- Se usa para esperar una condición

```
while (!condición)
        wait();
```

  - **¡siempre en un bucle!**
- Sólo se puede invocar dentro de un método sincronizado
  - o de un bloque sincronizado
- Se libera el cerrojo del objeto atómicamente
  - al mismo tiempo que se hace `wait()`
- Puede lanzar una excepción *InterruptedException*
  - usar manejador o propagar

Aviso de condición

- El método `notify()` reanuda la ejecución de una hebra que esté suspendida por haber hecho `wait()`
- Se usa para avisar de que se cumple una condición

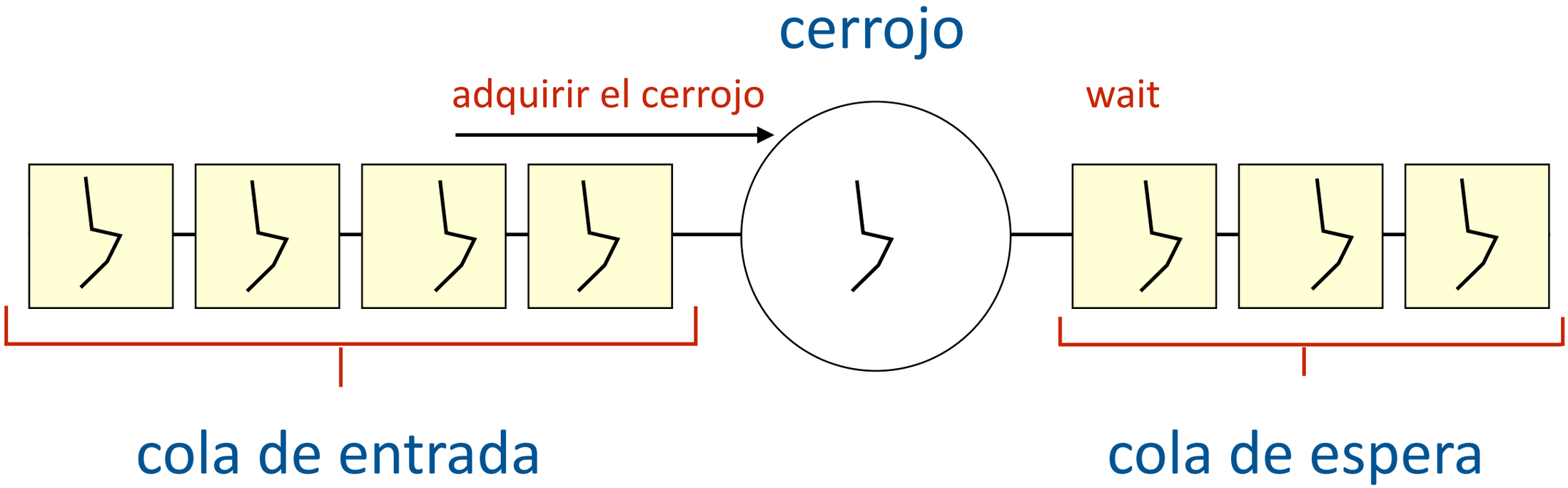
```
// cambiar el estado de alguna condición
notify();
```

- Una hebra avisa a otra
- Sólo se puede invocar dentro de un método sincronizado
  - o de un bloque sincronizado
- No se libera el cerrojo del objeto hasta que termina el método o sentencia sincronizada
  - la hebra que se reanuda intenta adquirir el cerrojo para continuar
  - puede tener que competir con otras hebras

Aviso general

- El método `notifyAll()` reanuda la ejecución de todas las hebras que esté suspendidas por haber hecho `wait()` en el mismo objeto
- El cerrojo del objeto se libera cuando termina el método o sentencia sincronizada
  - todas las hebras que se reanudan compiten para adquirir el cerrojo y poder continuar
- Con `notify()` no se sabe qué hebra se reanuda entre las que estaban suspendidas
- Con `notifyAll()` se reanudan todas y comprueban otra vez la condición

Colas de espera y de entrada



notify vs notifyAll

- Siempre que varias hebras puedan estar esperando condiciones distintas hay que usar `notifyAll`
- Sólo conviene hacer `notify` si
  - todas las hebras esperan la misma condición
  - sólo una hebra puede avanzar cuando se cumple la condición
- En caso de duda, usar `notifyAll`

Ejemplo: estacionamiento

Monitor con condiciones

```
public class Parking {
    private final int capacidad; // número de coches que caben
    private int n = 0;           // número de coches que hay

    // constructor
    public  Parking (int capacidad) {
        this.capacidad = capacidad;
    }
    // entra un coche por una de las puertas
    public synchronized void entra (String puerta) {
        while (n >= capacidad) wait();
        n++;
    }
    // sale un coche por una de las puertas
    public synchronized void sale (String puerta) {
        n–;
        notifyAll();
    }
    // consulta
    public synchronized int ocupado() {
        return n;
    }
}
```

Productor y consumidor

Ejemplo: buffer

```
public class Buffer<E> {

    private E almacen;
    private boolean lleno = false;

    public synchronized void enviar(E dato)
            throws InterruptedException {
        while (lleno) wait();       // espera que haya sitio
        almacen = dato;
        lleno = true;
        notifyAll();                // avisa de que hay un valor
    }

    public synchronized E recibir()
            throws InterruptedException {
        E dato = null;
        while (!lleno) wait(); // espera que haya un valor
        dato = almacen;
        lleno = false;
        notifyAll();        // avisa de que hay sitio
        return dato;
    }

}
```

Ejemplo: productor

```
public class Productor<E> implements Runnable {

    private Buffer<E> b;

    public Productor(Buffer<E> b) {
        this.b = b;
    }

    public void run() {
        while (true) {
            E x =  …;          // producir x
            b.enviar(x);
        }
    }
}

// la sincronización está oculta en el buffer
```

Ejemplo: consumidor

```
public class Consumidor<E> implements Runnable {

    private Buffer<E> b;

    public Consumidor(Buffer<E> b) {
        this.b = b;
    }

    public void run() {
        while (true) {
            E x = b.recibir();
            … // consumir x
        }
    }
}

// la sincronización está oculta en el buffer
```

Resumen

Resumen

- La sincronización condicional se realiza en Java con los métodos wait() y notify() / notifyAll()
  - sólo se pueden usar en métodos/bloques sincronizados
  - wait() suspende la hebra desde donde se invoca
  - notify() reanuda una hebra suspendida
  - notifyAll() reanuda todas las hebras suspendidas

- Hay que hacer wait() siempre en un bucle while
  - para comprobar otra vez la condición de espera cuando despierta

```
while (!condición) wait();
```