

Asignación de Prácticas Número 7

Programación Concurrente y de Tiempo Real

Antonio J. Tomeu¹

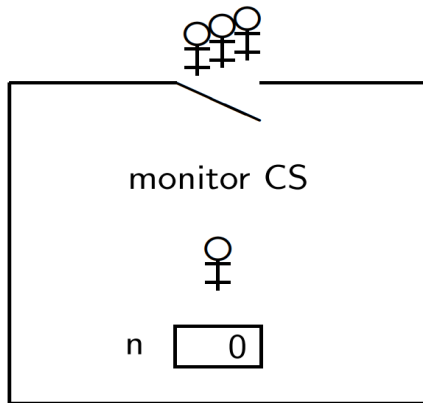
¹Departamento de Ingeniería Informática
Universidad de Cádiz

PCTR, 2021

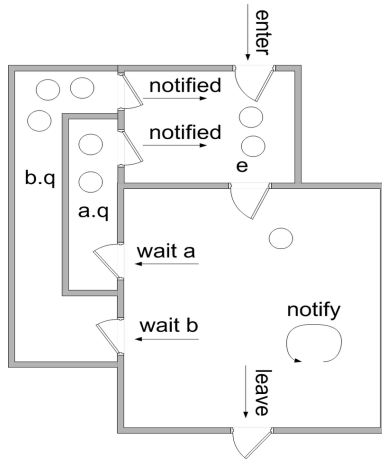
Objetivos de la Práctica

- ▶ Comprender que los monitores escrito en Java con el API estándar se parecen a un monitor teórico...
- ▶ ... pero no son iguales. La aproximación de Java con el API estándar es más «grosera».
- ▶ Aprender a implementar monitores en Java con el API estándar y el protocolo de diseño de monitores.
- ▶ Implementar algunos monitores sencillos con el API estándar de Java.

Modelo Teórico de Monitor: Exclusión Mutua I



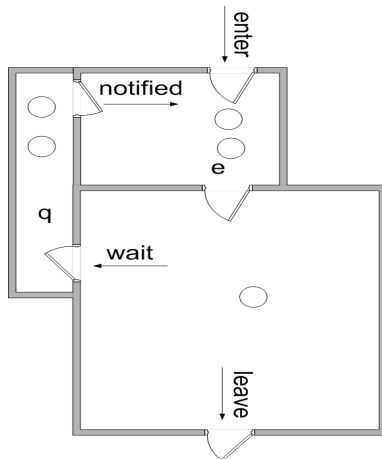
Modelo Teórico de Monitor: Sincronización-Variables de Condición I



- ▶ Cada lenguaje de programación implementa el concepto teórico como sus diseñadores deciden, alejándose más o menos del mismo. En el caso de Java, se tienen dos opciones para implementar un monitor:
- ▶ Mediante el API estándar, que proporciona una aproximación «grosera» al concepto teórico, ya que el sistema de colas para sincronización mediante múltiples variables de condición teórico, está soportado aquí mediante una única cola llamada «wait-set» que no permite dormir (ni despertar) a las hebras por condiciones de sincronización concretas. Ello exige el uso de condiciones de guarda para lograr semánticas homologables al modelo teórico.

- ▶ Mediante el API de alto nivel, que proporciona sistemas de colas complejos para sincronización mediante las variable de condición que se necesiten, lo cuál permite sincronizar a las hebras por condiciones de concretas, y es una implementación más «fina» del concepto teórico, y también más próxima al mismo. Las condiciones de guarda siguen siendo necesarias. Será objeto de análisis en prácticas posteriores.

Monitores en Java con API Estándar: El *Wait-Set* I



Modelo Teórico de Monitor Para un Semáforo: Exclusión Mutua y Sincronización I

- ▶ Escribimos un monitor teórico llamado `Sem` para implementar el concepto de semáforo.
- ▶ Este monitor deberá encapsular un contador (valor del semáforo)...
- ▶ ... y ofrecer un API de operaciones `waitS` y `signalS`, con la misma semántica proporcionada por un semáforo real.
- ▶ Pregunta: Cuántas variables de condición necesitará este monitor?
- ▶ Respuesta: una, para enviar a esperar a los procesos (hebras) que encuentren el valor del semáforo a cero.
- ▶ Pregunta: Cuántas colas tendrá el monitor `Sem` en total?
- ▶ Respuesta: dos; la cola de entrada de tareas al monitor bajo exclusión mutua, y la de la variable de condición.

Modelo Teórico de Monitor Para Simular un Semáforo¹: Pseudocódigo I

```
Monitor Sem is
```

```
begin
```

```
integer s;           //simula el contador del semáforo  
Condition notZero; //simula la cola de espera del semáforo
```

```
procedure waitS is //se ejecuta en e.m. por definición  
begin
```

```
//tareas esperan a que el semáforo aumente
```

```
while (s==0) wait(notZero);
```

```
s:=s-1;
```

```
end;
```

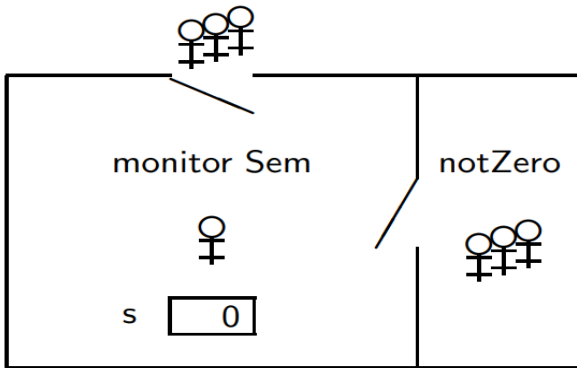
¹Según el algoritmo 7.2: Semaphore simulated with a Monitor, del texto Principles of Concurrent and Distributed Programming (2006) de M. Ben-Ari

Modelo Teórico de Monitor Para Simular un Semáforo: Pseudocódigo I

```
procedure signalS is //se ejecuta en e.m. por definición
begin
  s:=s+1;
  send(notZero); //despertamos a tarea de la cola
end;

begin //código de inicialización del monitor
  s:=k; //semáforo a valor inicial
end;
end.
```

Modelo Teórico de Monitor Para un Semáforo: Gráficamente



Cómo Transformar Un Monitor Teórico A Un Equivalente Funcional En Java? I

- ▶ Fácilmente, aplicando el protocolo específicamente descrito para ello; ahora, repase ese protocolo.
- ▶ Las variables y estructuras de datos del monitor teórico son atributos en Java.
- ▶ Los métodos del monitor teórico se corresponden con sus homólogos en Java... pero etiquetados con `synchronized`.
- ▶ Las múltiples variables de condición del monitor teórico se simulan con un único *wait-set* en Java.
- ▶ El código de inicialización del monitor se sitúa en el constructor de clase en Java.

Modelo Real de Monitor Para un Semáforo con API Estándar de Java I

- ▶ Aplicamos el protocolo de diseño de monitores en Java con el API estándar.
- ▶ Escribimos un monitor en pseudocódigo (ya lo hemos hecho) para el problema.
- ▶ Lo codificamos en Java mediante una clase que:
 - ▶ encapsule a los datos como atributos `private`.
 - ▶ tenga a todos sus métodos (excepto el constructor) etiquetados como `synchronized`.
 - ▶ no hay variables de condición, únicamente el *wait-set*.
 - ▶ cada `wait(C)` del pseudocódigo se implementa en Java con una guarda que controle la condición que nos interesa: `while !(condicion) wait();`
 - ▶ cada `send(C)` del pseudocódigo se implementa en Java con `nofityAll();`
 - ▶ situamos en el constructor el código de inicialización.

Modelo Real de Monitor Para un Semáforo con API Estándar de Java II

- ▶ esto logra una semántica homologable al monitor teórico...
- ▶ ... pero no es gratis.

Modelo Real de Monitor Para un Semáforo con API Estándar de Java I

Escribimos el monitor en Java para simular a un semáforo...

```
1  public class Sem{
2      private int s;
3      public Sem(int s){
4          this.s = s;
5      }
6      public synchronized void waitS(){
7          while(s==0)try{wait();}catch(InterruptedException e){}
8          s=s-1;
9      }
10     public synchronized void signalS(){
11         s=s+1;
12         notifyAll();
13     }
14 }
```

y lo usamos para controlar una condición de concurso:

Modelo Real de Monitor Para un Semáforo con API Estándar de Java II

```
1  public class usaSem
2      extends Thread{
3
4      public static long n = 0;
5      public Sem semaforo;
6
7      public usaSem(Sem semaforo){this.semaforo=semaforo;}
8
9      public void run(){
10         for(long i=0; i<100000000; i++){
11             semaforo.waitS();
12             n++;
13             semaforo.signalS();
14         }
15     }
16
17     public static void main(String[] args) throws Exception{
18         Sem mon_semaforo = new Sem(1);
19         usaSem A          = new usaSem(mon_semaforo);
20         usaSem B          = new usaSem(mon_semaforo);
```


Modelo Real de Monitor Para un Semáforo con API Estándar de Java III

```
21      A.start(); B.start();
22      A.join(); B.join();
23      System.out.println(n);
24  }
25 }
```

Qué hago ahora? I

- ▶ Estudie la solución al monitor del productor-consumidor ofrecida en el Tema 3 del curso teórico, y vea cómo se ha aplicado este mismo protocolo a un caso ligeramente más complejo, donde el monitor teórico trabaja con dos variables de condición, que en Java se «simulan» con un único *wait-set* y el uso adecuado de `guardas+notifyAll()`. A partir de él, realice las siguientes modificaciones y estudie el comportamiento del código resultante:
 - ▶ reduzca el tamaño del buffer a uno y vea qué ocurre.
 - ▶ lance un productor y varios consumidores y vea qué ocurre.
 - ▶ lance varios productores y un consumidor y vea qué ocurre.
- ▶ Resolvemos el ejercicio del *drakkar vikingo*; para ello, debe:
 - ▶ determinar cuántos tipos de tareas tendrá; en este caso, habrá de dos tipos: vikingos y cocinero.

Qué hago ahora? II

- ▶ determinar con más detalle la sincronización descrita en el enunciado: el cocinero espera (en una variable de condición) mientras la marmita está llen, y los vikingos esperan mientras la marmita está vacía (en otra variable de condición).
- ▶ escribir un monitor teórico en pseudocódigo, que modele el problema adecuadamente.
- ▶ escribir un monitor en Java con el API estándar, aplicando el protocolo de diseño de monitores.
- ▶ Resolvemos el problema de los lectores escritores:
 - ▶ es un problema que ya ha sido presentado en clase de problemas; repase esa presentación y, si es necesario, estudie la naturaleza del problema en cualquier texto estándar de concurrencia.
 - ▶ en la carpeta de la práctica se le proporciona el pseudocódigo del monitor teórico que soluciona el problema.
 - ▶ escriba un monitor en Java con el API estándar, aplicando el protocolo de diseño de monitores.

Apéndice: Preguntas a Plantearse **Antes** de Escribir un Monitor Teórico I

- ▶ ¿Cuáles son los recursos compartidos a encapsular en el monitor?
- ▶ ¿Qué tipos de tareas (hebras) están presentes en el problema y utilizarán el monitor?
- ▶ ¿Qué procedimientos (métodos) debe ofrecer el monitor a las hebras que lo utilizarán?
- ▶ ¿Cuál es la sincronización que los métodos del monitor necesitan?
- ▶ ¿Cuántas variables de condición son necesarias para modelar esta sincronización?