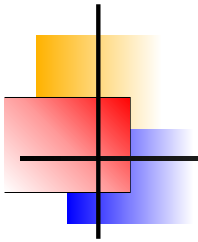


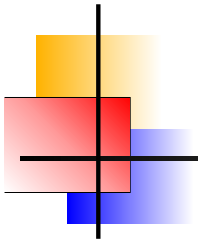
Concurrencia *Creación de Procesos en Java*

Guillermo Román Díez

groman@fi.upm.es

Universidad Politécnica de Madrid

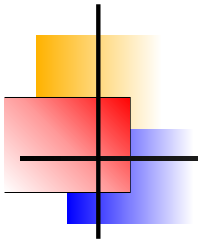
Curso 2016-2017



Concurrencia

Pregunta

¿qué es la concurrencia?



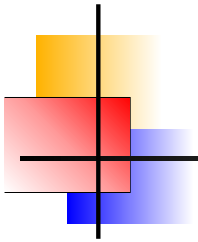
Concurrencia

Pregunta

¿qué es la concurrencia?

Concurrencia (RAE)

"*Acción de concurrir distintas personas, sucesos o cosas en un mismo lugar o tiempo*"



Concurrencia

Pregunta

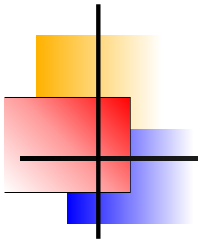
¿qué es la concurrencia?

Concurrencia (RAE)

"*Acción de concurrir distintas personas, sucesos o cosas en un mismo lugar o tiempo*"

Pregunta

¿y la concurrencia en el software?



Concurrencia

Pregunta

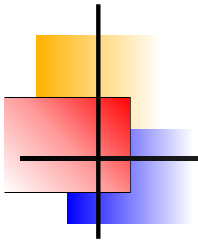
¿qué es la concurrencia?

Concurrencia (RAE)

"*Acción de concurrir distintas personas, sucesos o cosas en un mismo lugar o tiempo*"

Pregunta

¿y la concurrencia en el software? ¿qué es un proceso?



Concurrencia

Pregunta

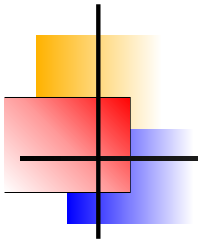
¿qué es la concurrencia?

Concurrencia (RAE)

"*Acción de concurrir distintas personas, sucesos o cosas en un mismo lugar o tiempo*"

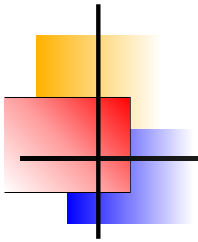
Pregunta

¿y la concurrencia en el software? ¿qué es un proceso? ¿es lo mismo un proceso que un programa?



Programa Secuencial Vs. Programa Concurrente

- **Programa Secuencial**: Define la ejecución **secuencial** de una lista de instrucciones
  - Llamamos **proceso** a la ejecución de un programa secuencial
- **Programa Concurrente**: Define dos o más programas secuenciales que pueden ejecutar concurrentemente en forma de **procesos paralelos**

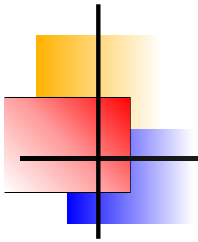


Programa Secuencial Vs. Programa Concurrente

- **Programa Secuencial**: Define la ejecución **secuencial** de una lista de instrucciones
  - Llamamos **proceso** a la ejecución de un programa secuencial
- **Programa Concurrente**: Define dos o más programas secuenciales que pueden ejecutar concurrentemente en forma de **procesos paralelos**

Pregunta

¿es lo mismo concurrencia que paralelismo?



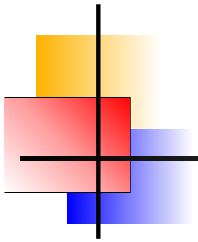
Programa Secuencial Vs. Programa Concurrente

- **Programa Secuencial**: Define la ejecución **secuencial** de una lista de instrucciones
  - Llamamos **proceso** a la ejecución de un programa secuencial
- **Programa Concurrente**: Define dos o más programas secuenciales que pueden ejecutar concurrentemente en forma de **procesos paralelos**

Pregunta

¿es lo mismo concurrencia que paralelismo?

- Si únicamente disponemos de un procesador puede haber concurrencia pero no paralelismo
- Se intercalará la ejecución de los diferentes procesos

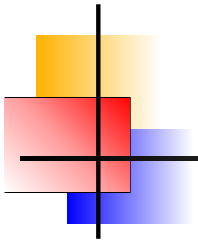


Concurrencia

Concurrencia

Ejecución Simultánea

- **Ejecución Simultánea**: la existencia de varios procesos interactuando a la vez multiplica las posibles ejecuciones del programa

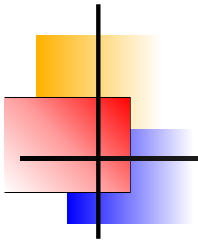


Concurrencia

Concurrencia

Ejecución Simultánea + Indeterminismo

- **Ejecución Simultánea**: la existencia de varios procesos interactuando a la vez multiplica las posibles ejecuciones del programa
- **Indeterminismo**: dependiendo de cada posible ejecución se podrán obtener resultados diferentes

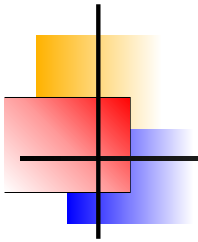


Concurrencia

Concurrencia

Ejecución Simultánea + Indeterminismo + Interacción

- **Ejecución Simultánea**: la existencia de varios procesos interactuando a la vez multiplica las posibles ejecuciones del programa
- **Indeterminismo**: dependiendo de cada posible ejecución se podrán obtener resultados diferentes
- **Interacción**: los procesos pueden interactuar entre sí

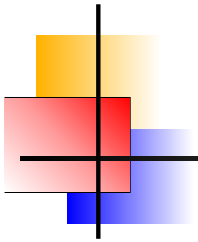


Interacción entre procesos

Interacción

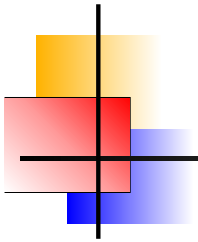
Comunicación + Sincronización

- **Comunicación**: Permite a la ejecución de un proceso influenciar en la ejecución del otro
  - Variables Compartidas
  - Paso de mensajes
- **Sincronización**: Permite establecer cierto **orden** de ejecución de diferentes partes del programa
  - Exclusión mutua
  - Sincronización por condición



Dificultades de la Concurrencia

- La velocidad ejecución de un programa depende de muchos factores
  - No se puede garantizar que programas concurrentes idénticos y ejecutados en el mismo procesador ejecuten exactamente igual
  - Lo único que podemos asumir es que el programa **progresa**
  - No asumimos nada sobre los posibles **ritmos** de ejecución



Dificultades de la Concurrencia

- La velocidad ejecución de un programa depende de muchos factores
  - No se puede garantizar que programas concurrentes idénticos y ejecutados en el mismo procesador ejecuten exactamente igual
  - Lo único que podemos asumir es que el programa **progresa**
  - No asumimos nada sobre los posibles **ritmos** de ejecución
- Incluso los programas más sencillos se componen de instrucciones más pequeñas

Programa Java:

```
y = x + 1;
x = y;
```

Instrucciones Bytecode:

```
0: load x
1: const 1
2: add
3: store y
4: load y
5: store x
```

Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

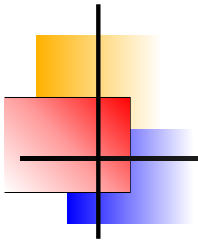
```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3$



Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

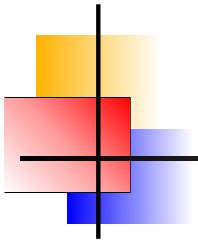
```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \rightarrow x = 3$$



Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \rightarrow x = 3$

$i_0 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \triangleright i_1 \triangleright i_2 \triangleright i_3$

Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \rightarrow x = 3$$

$$i_0 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \triangleright i_1 \triangleright i_2 \triangleright i_3 \rightarrow x = 1$$

Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \rightarrow x = 3$$

$$i_0 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \triangleright i_1 \triangleright i_2 \triangleright i_3 \rightarrow x = 1$$

$$i_0 \triangleright i'_0 \triangleright i_1 \triangleright i'_1 \triangleright i_2 \triangleright i'_2 \triangleright i_3 \triangleright i'_3$$

Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \rightarrow x = 3$$
$$i_0 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \triangleright i_1 \triangleright i_2 \triangleright i_3 \rightarrow x = 1$$
$$i_0 \triangleright i'_0 \triangleright i_1 \triangleright i'_1 \triangleright i_2 \triangleright i'_2 \triangleright i_3 \triangleright i'_3 \rightarrow x = 2$$

Dificultades que presenta la concurrencia

- ¿Cuál será el valor final de `x` si ejecutamos el programa en dos procesadores con un valor inicial `x = 0`?

Proceso 1: (`x = x + 1`)

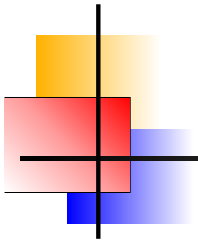
```
i0: load x
i1: const 1
i2: add
i3: store x
```

Proceso 2: (`x = x + 2`)

```
i'0: load x
i'1: const 2
i'2: add
i'3: store x
```

%

$$i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \rightarrow x = 3$$
$$i_0 \triangleright i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \triangleright i_1 \triangleright i_2 \triangleright i_3 \rightarrow x = 1$$
$$i_0 \triangleright i'_0 \triangleright i_1 \triangleright i'_1 \triangleright i_2 \triangleright i'_2 \triangleright i_3 \triangleright i'_3 \rightarrow x = 2$$
$$i'_0 \triangleright i'_1 \triangleright i'_2 \triangleright i'_3 \triangleright i_0 \triangleright i_1 \triangleright i_2 \triangleright i_3 \rightarrow x = 3$$
$$\vdots$$

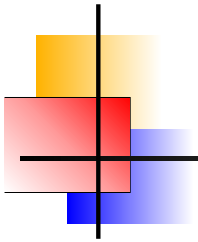


Operaciones atómicas

Operaciones Atómicas

"Son operaciones indivisibles, es decir, deben realizarse de manera completa (o poder deshacerse de manera completa en caso de fallar o ser interrumpidas)"

- Puede estar formada por una o varias instrucciones
- Ningún proceso puede acceder a la información modificada hasta que no se haya completado la operación atómica



Multiprocessing Vs. Multiprogramming

Multiprocessing

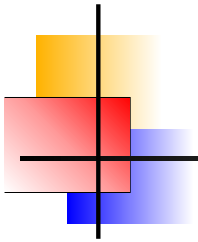
"*varios procesos comparten uno o más procesadores*"

Multiprogramming

"*cada uno de los procesos tiene su propio procesador y se comunican mediante memoria compartida*"

Distributed processing

"*cada uno de los procesos tiene su propio procesador, pero se comunican mediante una red de comunicaciones*"

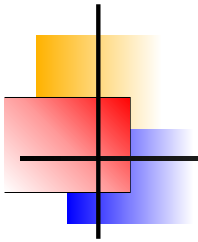


Aspectos claves de la concurrencia

- ¿Cómo indicamos la ejecución concurrente?
- ¿Qué tipo de comunicación entre procesos utilizar?
- ¿Qué mecanismo de sincronización utilizamos?

Objetivos de la asignatura

Los objetivos de la asignatura tratan de dar respuesta a estas preguntas



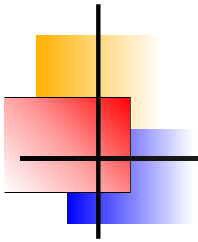
Objetivos de la Asignatura

Diseño y especificación de Programas Concurrentes

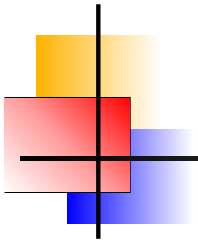
- Definir las interacciones entre los procesos que componen el sistema
- Usar técnicas formales para especificar un programa concurrente
- Detectas las partes del programa que necesitan control de acceso

Estudiar modelos de programación concurrente

- Programas con concurrencia explícita (comunicación y sincronización)
- Programación a través de paso de mensajes

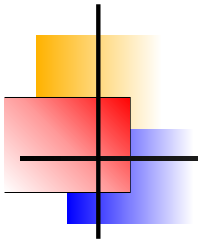


Procesos en Java



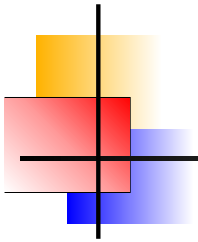
Procesos Vs. Threads

- Proceso
  - Dispone de su propio entorno de ejecución con sus propios recursos y espacio de memoria
  - Se suelen identificar con programas o aplicaciones
  - El cambio entre procesos es lento
  - El SSOO es el encargado de su gestión
  - Se conocen como procesos *pesados* (heavy)
- Thread (hilo o hebra)
  - Los threads creados en el mismo proceso comparten recursos (memoria incluida)
  - El cambio de contexto es rápido
  - Pueden no estar soportados en el sistema operativo
    - Muchas veces lo incluye el lenguaje de programación
  - También conocidos como procesos *ligeros* (lightweight)



¿Cómo implementar Threads en Java?

- Para crear un thread en Java es necesario implementar una clase donde se implemente *el código que ejecutará* el thread
- Esto se puede hacer de dos formas:

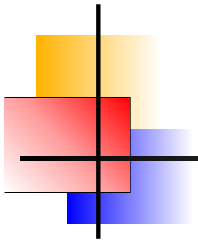


¿Cómo implementar Threads en Java?

- Para crear un thread en Java es necesario implementar una clase donde se implemente *el código que ejecutará* el thread
- Esto se puede hacer de dos formas:

(1) Extender la clase `Thread` y sobreescribir el método `run`

```
public class MiPrimerThread extends Thread {
    public void run() { /*CODIGO AQUI */ }
}
```



¿Cómo implementar Threads en Java?

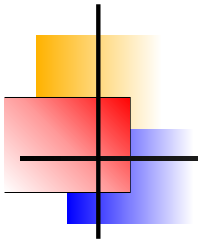
- Para crear un thread en Java es necesario implementar una clase donde se implemente *el código que ejecutará* el thread
- Esto se puede hacer de dos formas:

(1) Extender la clase `Thread` y sobreescribir el método `run`

```
public class MiPrimerThread extends Thread {
    public void run() { /*CODIGO AQUI */ }
}
```

(2) Implementar el interfaz `Runnable`, que requiere implementar el método `run`

```
public class MiPrimerRunnable implements Runnable {
    public void run() { /*CODIGO AQUI */ }
}
```



¿Cómo lanzar Threads en Java?

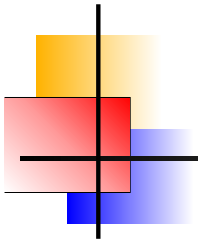
1. Primero se crea el objeto de tipo `Thread`
   - Ojo!! Esto no arranca el nuevo thread, simplemente instancia el objeto
2. Ejecutar el método `start()`
   - `start` lanza el nuevo thread y ejecuta el método `run`

- Extendiendo de `Thread`

```
MiPrimerThread t = new MiPrimerThread();
t.start();
```

- Implementando `Runnable`

```
Runnable runnable = new MiPrimerRunnable();
Thread t = new Thread(runnable);
t.start();
```



¿Cómo lanzar Threads en Java?

1. Primero se crea el objeto de tipo `Thread`
   - Ojo!! Esto no arranca el nuevo thread, simplemente instancia el objeto
2. Ejecutar el método `start()`
   - `start` lanza el nuevo thread y ejecuta el método `run`

- Extendiendo de `Thread`

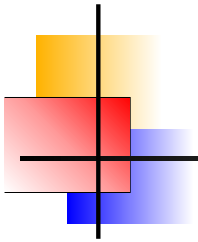
```
MiPrimerThread t = new MiPrimerThread();
t.start();
```

- Implementando `Runnable`

```
Runnable runnable = new MiPrimerRunnable();
Thread t = new Thread(runnable);
t.start();
```

NOTA!!

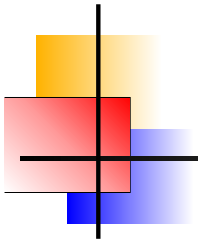
No es lo mismo llamar a `start()` que a `run()!!!!`



Thread.sleep

- El método estático `Thread.sleep(tiempo)` **suspende** la ejecución del thread que está ejecutando un tiempo determinado
  - El tiempo se expresa en ms
  - El método `sleep` puede lanzar `InterruptedException`

```
class ThreadDormido extends Thread {
  public void run() {
    try {
      Thread.sleep(5000);  // A dormir 5 segundos
    } catch (InterruptedException e) {
      System.out.println("Me han interrumpido!");
    }
  }
}
```



Interrupciones

- Las **interrupciones** son una indicación de que un hilo debe detener lo que está haciendo para hacer otra cosa
- Llamando al método `t.interrupt()` se puede *intentar* interrumpir lo que está haciendo thread `t`
- Las interrupciones sólo *interrumpen* el thread cuando se está ejecutando un método que lanza la excepción `InterruptedExecption`
  - El programador del thread interrumpido es quien decide qué hacer para tratar las interrupciones capturando `InterruptedException`
- Si estamos ejecutando, siempre podemos comprobar si nos han **interrumpido** a través de `isInterruted()`

join: Esperar la terminación de un thread

- El método `join()` hace que el thread en ejecución espere a que el thread `t`, termine su ejecución
  - También se puede especificar un tiempo máximo de espera
  - Lanza `InterruptedException`

```
Thread t1 = new Thread();
Thread t2 = new Thread();
t1.start();
t2.start();
try{
    t2.join();
    t1.join();
}
catch (InterruptedException e){
    ...
}
```