

Conceptos de programación concurrente

Programación de redes, sistemas y servicios

- Conceptos fundamentales
 - Procesos vs hilos
 - Tipos de concurrencia
 - Características de los procesos concurrentes
- Exclusión mutua y sincronización
- Introducción a la concurrencia en Java

Conceptos fundamentales

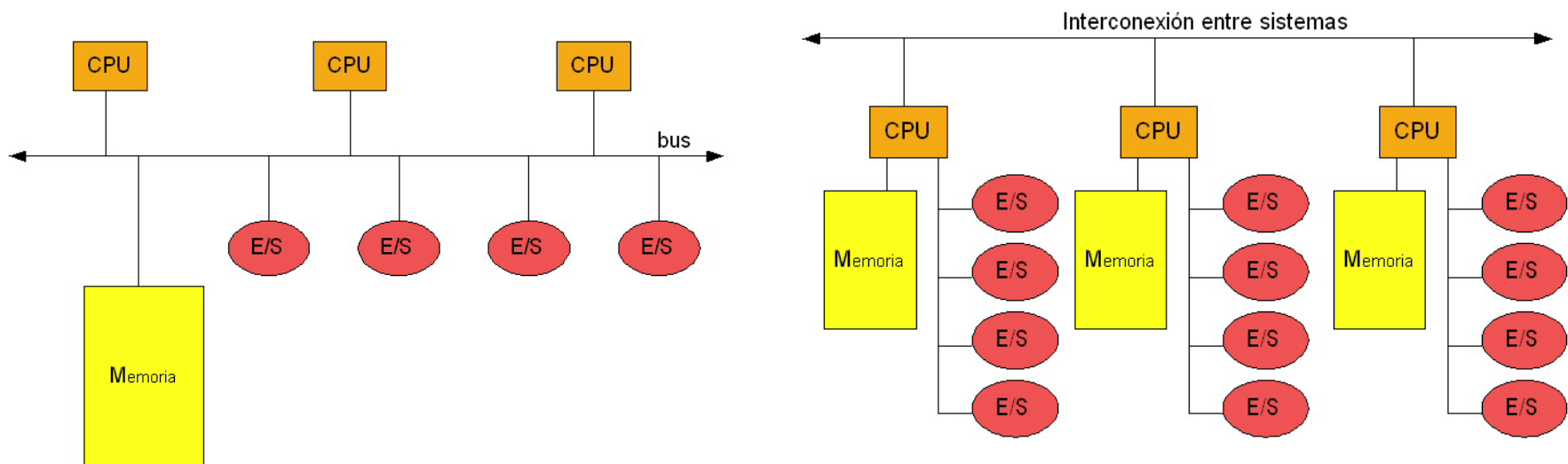
Introducción

- Un *programa secuencial* es el conjunto de sentencias escritas de forma secuencial y cuya ejecución sigue dicha secuencia
- Un *programa concurrente* es el conjunto de *programas secuenciales* (tareas) ejecutados “a la vez”
- La *programación concurrente* es el conjunto de técnicas de programación que implementan paralelismo potencial y solucionan problemas de sincronización y comunicación
- Introduce conceptos importantes como: *proceso*, *hilo* (*thread*), *sincronización*

Conceptos fundamentales

Procesos vs hilos

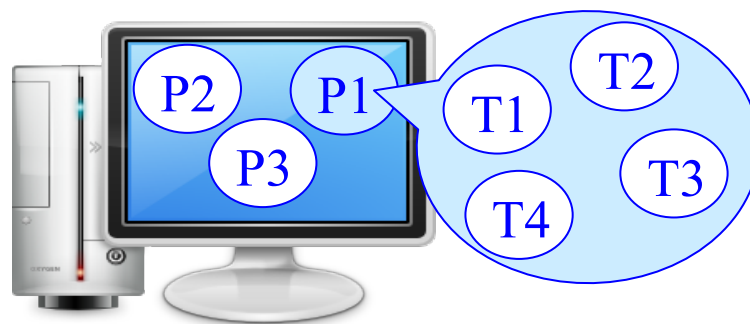
- En la práctica, se habla de:
 - **Multiprogramación**: si se intercala las instrucciones de varias tareas en un único procesador (conurrencia lógica)
 - **Multiproceso**: si se hace en un multiprocesador con memoria compartida o multiprocesador con memoria local (conurrencia real)



Conceptos fundamentales

Procesos vs hilos

- Un **proceso** es un programa en ejecución que se identifica mediante su nombre y un identificador
 - Los procesos son independientes, con espacio de direcciones propio
 - Se comunican sólo mediante mecanismos habilitados por el SO
- Un **hilo** (*thread*) es una secuencia de código que se ejecuta en el dominio de protección de un proceso
 - Los hilos de un mismo proceso comparten el espacio de direcciones del proceso
- La ejecución de un proceso conlleva siempre la ejecución de uno o varios hilos



Conceptos fundamentales

Ejecución de tareas concurrentes

- Las tareas concurrentes (procesos o hilos) pueden ejecutarse independientemente entre sí o no
 - Si no son independientes necesitan interaccionar entre ellos
- El motivo de esta dependencia puede ser:
 - **Cooperación** en un determinado trabajo
 - Uso de información o **recursos compartidos (competición)**
- Las tareas concurrentes son ejecutadas bajo control del sistema operativo
 - De forma entrelazada, según el número de procesadores y la política de planificación del SO
 - A diferencia de los programas secuenciales el flujo del programa no siempre es el mismo (**Indeterminismo**)

- Conceptos fundamentales
 - Procesos vs hilos
 - Tipos de concurrencia
 - Ejecución de tareas concurrentes
- Introducción a la concurrencia en Java
- Exclusión mutua y sincronización

Introducción a la concurrencia en Java

Hilos en Java

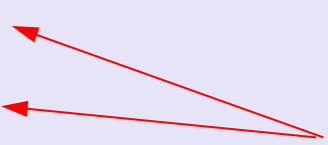
- Java es un lenguaje *multi-thread*
 - Un proceso puede tener más de un *thread*. Al arrancar un programa existe un hilo principal, que puede crear otros hilos
 - Los *hilos* de un proceso comparten el mismo espacio de memoria
 - Un *hilo* es un objeto derivado de una clase con capacidad de ejecución concurrente
- Las clases e interfaces relacionadas con los hilos son:
 - La clase ***Thread*** permite que otra clase que derive de ella se comporte como un hilo o *thread*
 - La interfaz ***Runnable*** permite que la clase que la implemente se comporte como un hilo
 - La clase ***Object*** aporta los métodos: *wait*, *notify* and *notifyAll* (para la sincronización entre hilos)

Introducción a la concurrencia en Java

Hilos en Java

```
class MiProcesoMultiHilo extends Thread {  
    // variables de instancia y estáticas  
    // métodos de la clase  
    public void run() {  
        // código de lo que se quiere realizar  
    }  
}
```

```
...  
MiProcesoMultiHilo hilo1=new MiProcesoMultiHilo();  
MiProcesoMultiHilo hilo2=new MiProcesoMultiHilo();  
hilo1.start();  
hilo2.start();  
...
```



Método de la clase Thread

Introducción a la concurrencia en Java

Hilos en Java

```
class MiProcesoMultiHilo extends OtraClase implements Runnable {  
    // variables de instancia y estáticas  
    // métodos de la clase  
    public void run() {  
        // código de lo que se quiere realizar  
    }  
}
```

```
...  
MiProcesoMultiHilo hilo1=new MiProcesoMultiHilo();  
MiProcesoMultiHilo hilo2=new MiProcesoMultiHilo();  
new Thread(hilo1).start();  
new Thread(hilo2).start();  
...
```

Introducción a la concurrencia en Java

Hilos en Java

- Cada *thread* tiene una prioridad (valor entero entre 1 y 10)
- El *thread* inicial tiene una prioridad por defecto (5)
- El sistema operativo determina el *thread* a ejecutarse en función de su prioridad (mayor valor, mayor prioridad)
- Cuando se crea un *thread*, éste hereda la prioridad de su padre
- El programador puede variar la prioridad de un *thread*

Introducción a la concurrencia en Java

Hilos en Java

- Un *thread* fuera de la región crítica puede:

- Pararse temporalmente usando el método:

```
public void sleep(long milisegundos)
```

- Esperar a que otro *thread* termine su ejecución usando el método:

```
public void join()
```

Introducción a la concurrencia en Java

Hilos en Java



Ejercicio: Realizar una clase Java llamada *Productor* que implemente un *thread*, que genere 10 números enteros aleatorios entre 0 y 100 y presente en pantalla su identificador y el número aleatorio generado. A través del constructor debe indicarse el valor del identificador (tipo *int*).

Ejercicio: Realizar la clase pública que crea dos objetos del tipo anterior, con identificador 1 y 2 y los ejecuta concurrentemente.

Introducción a la concurrencia en Java

Hilos en Java

```
1  package principal;
2  ☐ import java.util.Random;
3
4  public class Productor extends Thread {
5
6      private int id;
7      private Random aleatorio;
8
9      ☐ public Productor(int id) {
10         this.id = id;
11         aleatorio = new Random(System.currentTimeMillis() * id);
12     }
13
14     @Override
15     ☒ ☐ public void run() {
16         int i;
17         for (i = 0; i < 10; i++) {
18             int numero = aleatorio.nextInt(101);
19             System.out.println("Soy el productor " + id +
20                               " y el número es: " + numero);
21         }
22     }
23 }
```

Introducción a la concurrencia en Java

Hilos en Java

Ejercicio: Realizar un programa Java que cree un *thread* llamado *Productor*, que genere 10 números enteros aleatorios entre 0 y 100 y presente en pantalla su identificador y el número aleatorio generado. A través del constructor debe introducirse el identificador del productor (*byte*).



Ejercicio: Realizar la clase pública que crea dos objetos del tipo anterior, con identificador 1 y 2 y los ejecuta concurrentemente.

Introducción a la concurrencia en Java

Hilos en Java

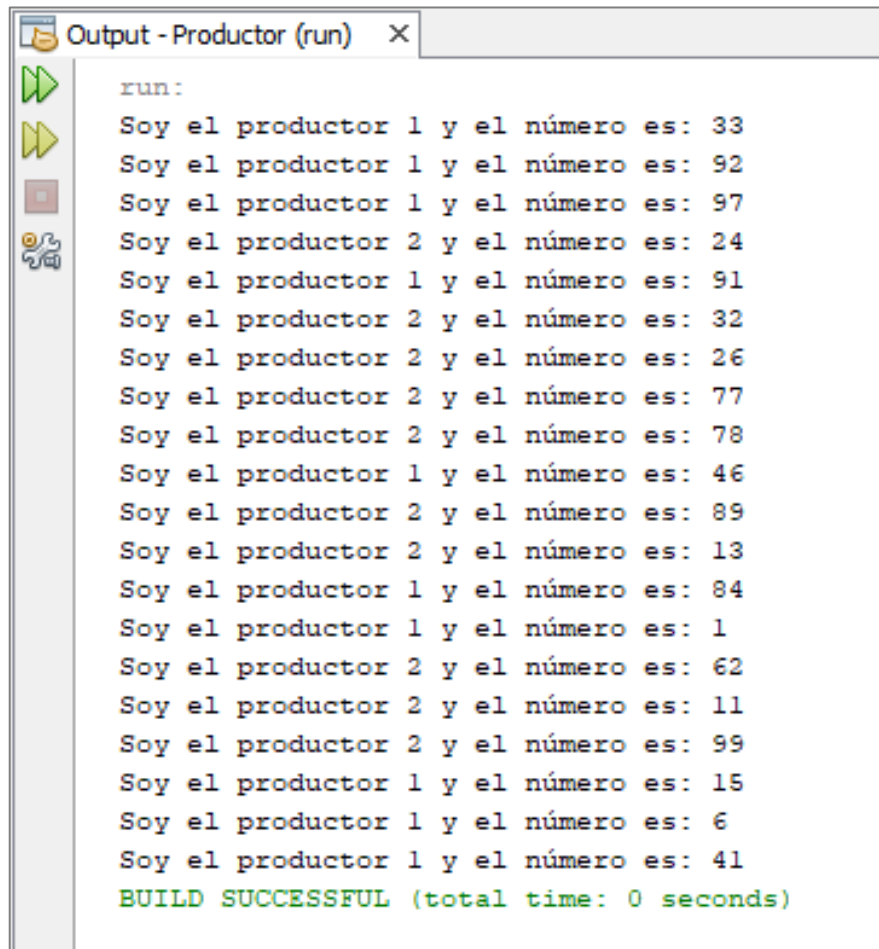
```
1  package principal;
2
3  public class Main {
4
5      public static void main(String[] arg) {
6          Productor p1 = new Productor(1);
7          Productor p2 = new Productor(2);
8          p1.start();
9          p2.start();
10
11     }
12 }
```

¿Qué pasaría si se llamara a `.run()` en vez de `.start()`?

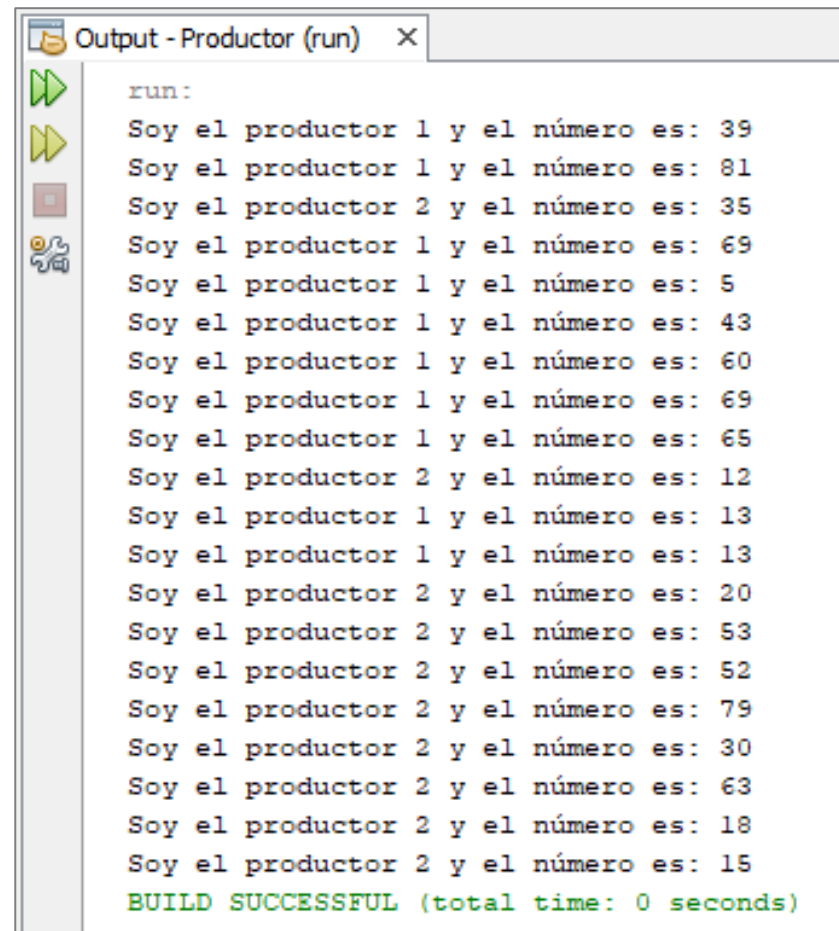
Introducción a la concurrencia en Java

Hilos en Java

Ejecución de hilos sin relación entre ellos



```
run:
Soy el productor 1 y el número es: 33
Soy el productor 1 y el número es: 92
Soy el productor 1 y el número es: 97
Soy el productor 2 y el número es: 24
Soy el productor 1 y el número es: 91
Soy el productor 2 y el número es: 32
Soy el productor 2 y el número es: 26
Soy el productor 2 y el número es: 77
Soy el productor 2 y el número es: 78
Soy el productor 1 y el número es: 46
Soy el productor 2 y el número es: 89
Soy el productor 2 y el número es: 13
Soy el productor 1 y el número es: 84
Soy el productor 1 y el número es: 1
Soy el productor 2 y el número es: 62
Soy el productor 2 y el número es: 11
Soy el productor 2 y el número es: 99
Soy el productor 1 y el número es: 15
Soy el productor 1 y el número es: 6
Soy el productor 1 y el número es: 41
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
run:
Soy el productor 1 y el número es: 39
Soy el productor 1 y el número es: 81
Soy el productor 2 y el número es: 35
Soy el productor 1 y el número es: 69
Soy el productor 1 y el número es: 5
Soy el productor 1 y el número es: 43
Soy el productor 1 y el número es: 60
Soy el productor 1 y el número es: 69
Soy el productor 1 y el número es: 65
Soy el productor 2 y el número es: 12
Soy el productor 1 y el número es: 13
Soy el productor 1 y el número es: 13
Soy el productor 2 y el número es: 20
Soy el productor 2 y el número es: 53
Soy el productor 2 y el número es: 52
Soy el productor 2 y el número es: 79
Soy el productor 2 y el número es: 30
Soy el productor 2 y el número es: 63
Soy el productor 2 y el número es: 18
Soy el productor 2 y el número es: 15
BUILD SUCCESSFUL (total time: 0 seconds)
```

Introducción a la concurrencia en Java

Hilos en Java

```
1  package principal;
2  import java.util.Random;
3
4  public class Productor extends Thread {
5
6      private int id;
7      private Random aleatorio;
8
9      public Productor(int id) {
10         this.id = id;
11         aleatorio = new Random(System.currentTimeMillis() * id);
12     }
13
14     @Override
15     public void run() {
16         int i;
17         for (i = 0; i < 10; i++) {
18             int numero = aleatorio.nextInt(101);
19             System.out.println("Soy el productor " + id +
20                               " y el número es: " + numero);
21         }
22     }
23 }
```

Introducción a la concurrencia en Java

Hilos en Java

Ejercicio: Realizar un programa Java con una clase *thread* llamada *Productor*, que genere 10 números enteros aleatorios entre 0 y 100 y presente en pantalla su identificador y el número aleatorio generado. A través del constructor debe introducirse el identificador del productor (*byte*).

Ejercicio: Realizar la clase pública que crea dos objetos del tipo anterior, con identificador 1 y 2 y los ejecuta concurrentemente.

Ejercicio: Modificar la clase *Productor* para que después de presentar el número aleatorio, se “*duerma*” durante los milisegundos indicados en el número generado.

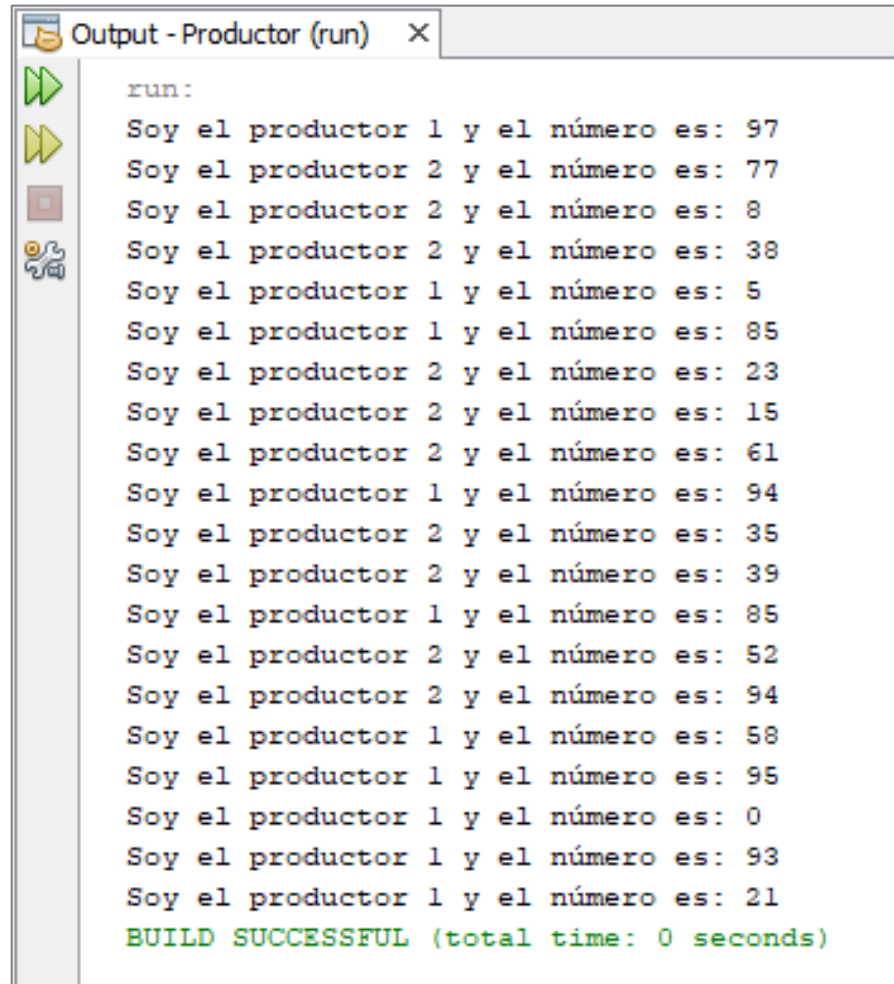
Introducción a la concurrencia en Java

Hilos en Java

```
1  package principal;
2  import java.util.Random;
3
4  public class Productor extends Thread {
5
6      private int id;
7      private Random aleatorio;
8
9      public Productor(int id) {
10         this.id = id;
11         aleatorio = new Random(System.currentTimeMillis() * id);
12     }
13
14     @Override
15     public void run() {
16         int i;
17         for (i = 0; i < 10; i++) {
18             int numero = aleatorio.nextInt(101);
19             System.out.println("Soy el productor " + id +
20                               " y el número es: " + numero);
21             try {
22                 sleep(numero);
23             } catch (InterruptedException ex) {}
24         }
25     }
26 }
```

Introducción a la concurrencia en Java

Hilos en Java



```
run:
Soy el productor 1 y el número es: 97
Soy el productor 2 y el número es: 77
Soy el productor 2 y el número es: 8
Soy el productor 2 y el número es: 38
Soy el productor 1 y el número es: 5
Soy el productor 1 y el número es: 85
Soy el productor 2 y el número es: 23
Soy el productor 2 y el número es: 15
Soy el productor 2 y el número es: 61
Soy el productor 1 y el número es: 94
Soy el productor 2 y el número es: 35
Soy el productor 2 y el número es: 39
Soy el productor 1 y el número es: 85
Soy el productor 2 y el número es: 52
Soy el productor 2 y el número es: 94
Soy el productor 1 y el número es: 58
Soy el productor 1 y el número es: 95
Soy el productor 1 y el número es: 0
Soy el productor 1 y el número es: 93
Soy el productor 1 y el número es: 21
BUILD SUCCESSFUL (total time: 0 seconds)
```

Introducción a la concurrencia en Java

Hilos en Java

Ejercicio: Realizar un programa Java con una clase *thread* llamada *Productor*, que genere 10 números enteros aleatorios entre 0 y 100 y presente en pantalla su identificador y el número aleatorio generado. A través del constructor debe introducirse el identificador del productor (*byte*).

Ejercicio: Realizar la clase pública que crea dos objetos del tipo anterior, con identificador 1 y 2 y los ejecuta concurrentemente.

Ejercicio: Modificar la clase *Productor* para que después de presentar el número aleatorio, se duerma durante los milisegundos indicados en el número leído.

Ejercicio: Implementar los cambios necesarios para usar **Runnable** en vez de **Thread**

Introducción a la concurrencia en Java

Hilos en Java

```
1 package principal;
2 import java.util.Random;
3
4 public class Productor implements Runnable {
5
6     private int id;
7     private Random aleatorio;
8
9     public Productor(int id) {
10         this.id = id;
11         aleatorio = new Random(System.currentTimeMillis() * id);
12     }
13
14     @Override
15     public void run() {
16         int i;
17         for (i = 0; i < 10; i++) {
18             int numero = aleatorio.nextInt(101);
19             System.out.println("Soy el productor " + id +
20                 " y el número es: " + numero);
21             try {
22                 Thread.sleep(numero);
23             } catch (InterruptedException ex) {}
24         }
25     }
26 }
```

Introducción a la concurrencia en Java

Hilos en Java

```
1  package principal;
2
3
4  public class Main {
5
6      public static void main(String[] arg) {
7          Productor p1 = new Productor(1);
8          Productor p2 = new Productor(2);
9          new Thread(p1).start();
10         new Thread(p2).start();
11     }
12 }
13
```


- Conceptos fundamentales
 - Procesos vs hilos
 - Tipos de concurrencia
 - Ejecución de tareas concurrentes
- Introducción a la concurrencia en Java
- Exclusión mutua y sincronización

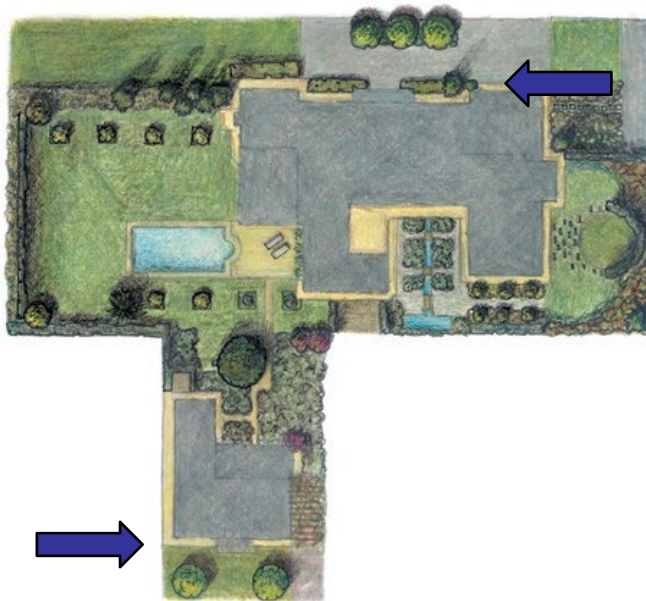
Exclusión mutua y sincronización

Introducción

- A menudo, las tareas concurrentes interaccionan entre ellas para finalizar correctamente un trabajo
 - A esta interacción se le denomina **sincronización**
 - Controla el orden en el que se ejecutan las tareas
- Tipos de sincronización: **cooperación** y **competición**
- La tarea P1 **coopera** con la tarea P2 cuando P1 debe esperar que P2 finalice, para continuar con su ejecución
- La tarea P1 **compite** con la tarea P2 cuando debe acceder a un dato al que está accediendo P2. P1 debe esperar a que P2 termine el acceso
 - Aplicación de la **exclusión mutua**

Exclusión mutua y sincronización

Introducción



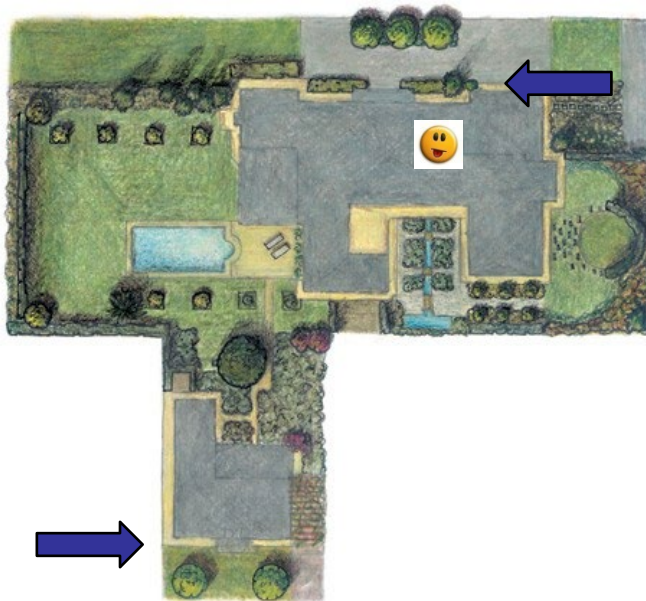
Proceso 2

VISITANTES 0

Proceso 1

Exclusión mutua y sincronización

Introducción



Proceso 2

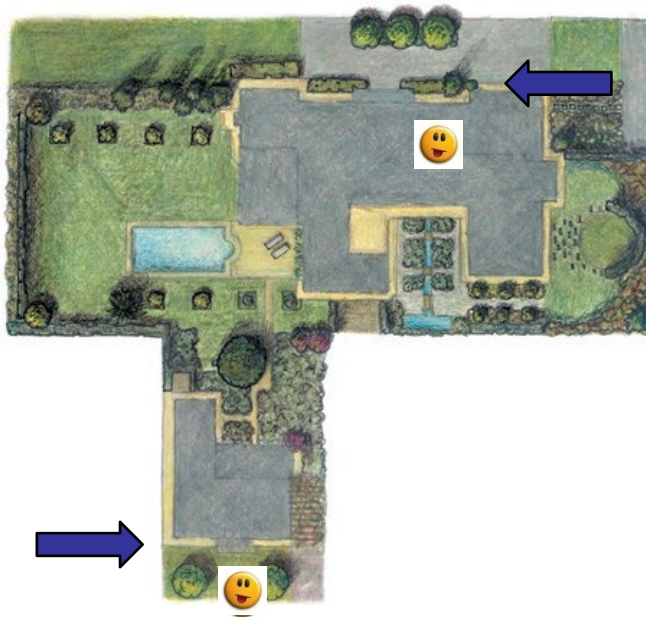
VISITANTES

1

Proceso 1

Exclusión mutua y sincronización

Introducción



Proceso 2

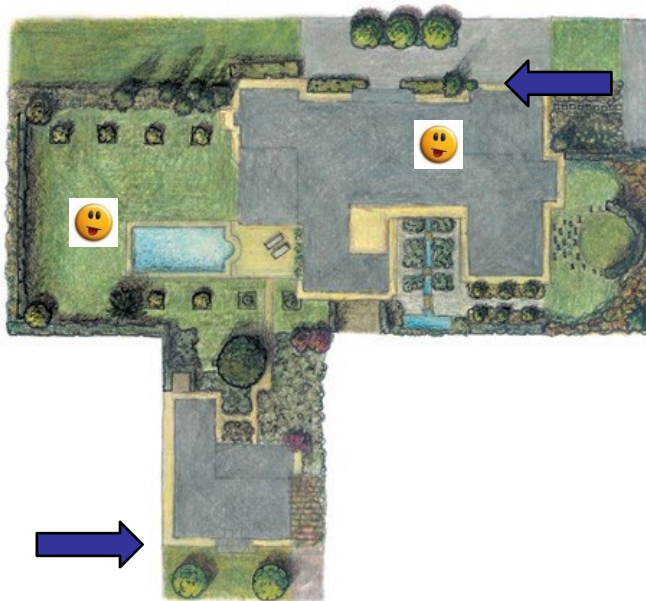
VISITANTES

1

Proceso 1

Exclusión mutua y sincronización

Introducción



Proceso 2

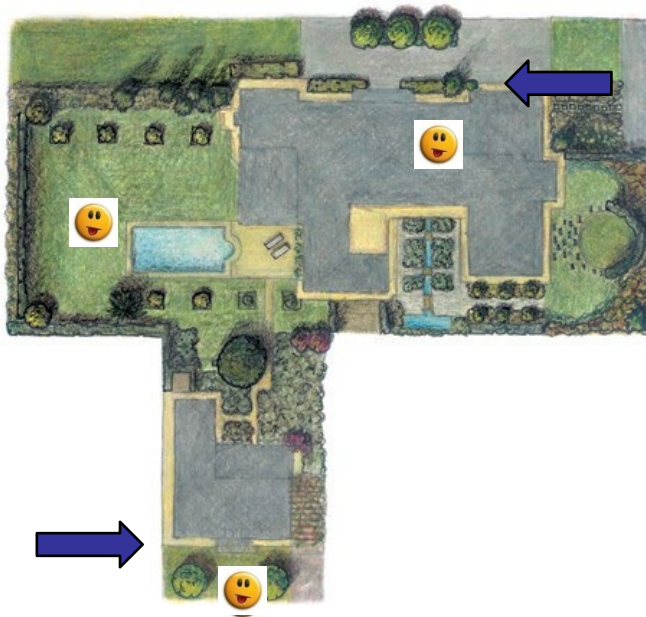
VISITANTES

2

Proceso 1

Exclusión mutua y sincronización

Introducción



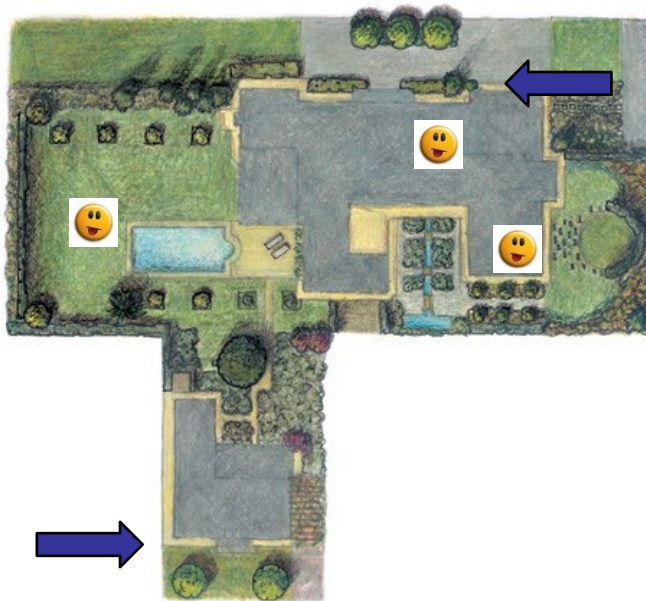
Proceso 2

VISITANTES 2

Proceso 1

Exclusión mutua y sincronización

Introducción



Proceso 2

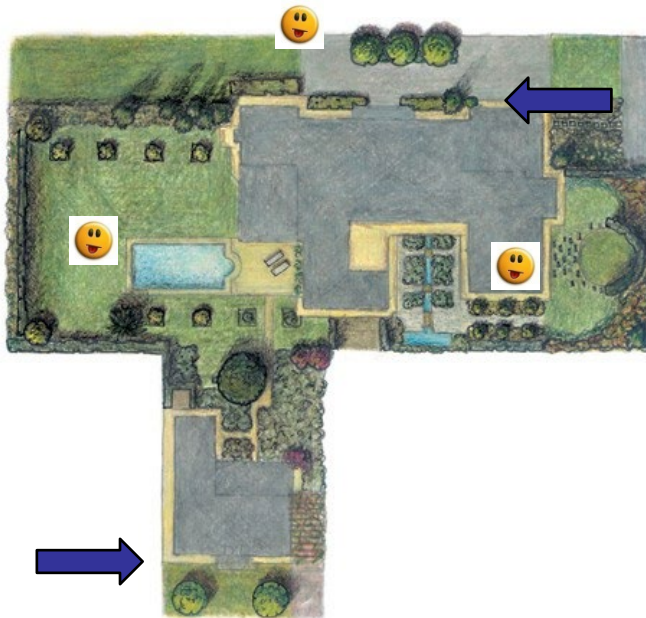
VISITANTES

3

Proceso 1

Exclusión mutua y sincronización

Introducción



Proceso 2

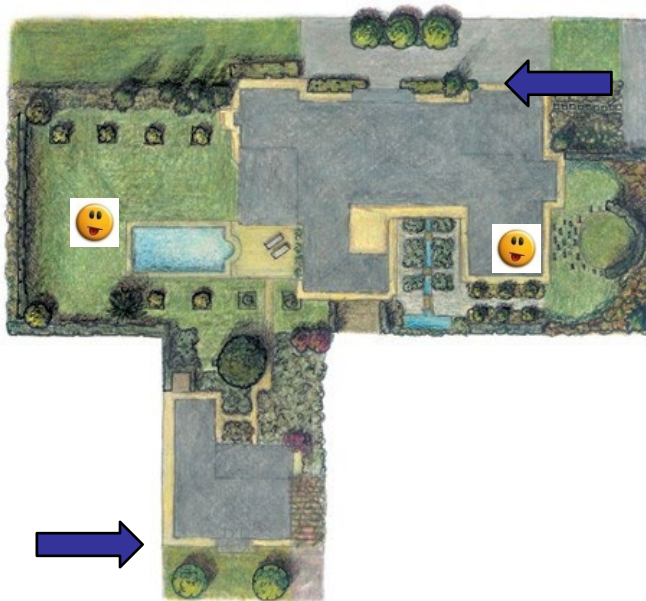
VISITANTES

2

Proceso 1

Exclusión mutua y sincronización

Introducción



Proceso 2

VISITANTES 2

Proceso 1

- Puede traer problemas porque las acciones de un proceso puede interferir con otro de forma incorrecta

Exclusión mutua y sincronización

Introducción

- La **exclusión mutua** permite el bloqueo temporal, por una tarea, de un recurso compartido
 - Otras tareas que necesiten el recurso compartido deben esperar
- Las instrucciones que se ejecutan en exclusión mutua forman una **región crítica**
- Cuando la tarea termina de ejecutar la **región crítica**, el SO permite que una tarea en espera entre en ella
- En su implementación evitar
 - **Bloqueo mutuo o deadlock**: Una tarea espera por un suceso que nunca va a ocurrir
 - **Retraso indefinido**: Todas las peticiones deben ser satisfechas (no se puede retrasar una petición indefinidamente)

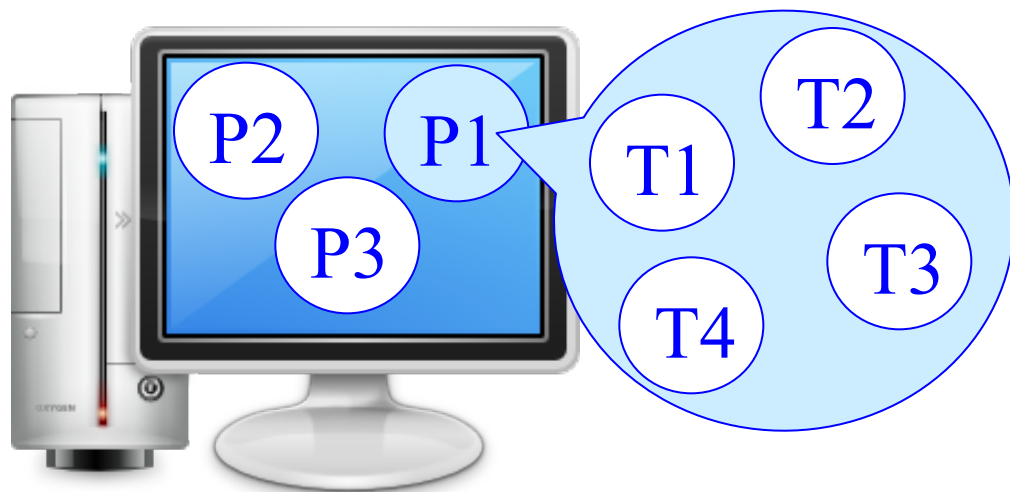
Exclusión mutua y sincronización

Introducción

- Para garantizar la **exclusión mutua** se pueden usar soluciones:
 - Basadas en *variables compartidas* (operaciones de acceso a memoria son atómicas):
 - **Semáforos**
 - **Regiones críticas y objetos condicionales**
 - **Monitores**
 - Basadas en *paso de mensajes* (operaciones de envío y recepción son atómicas):
 - **Paso de mensajes** (*send/receive*)
 - **Llamadas a procedimientos remotos** (RPC)
 - **Invocaciones remotas** (RMI)

Exclusión mutua y sincronización

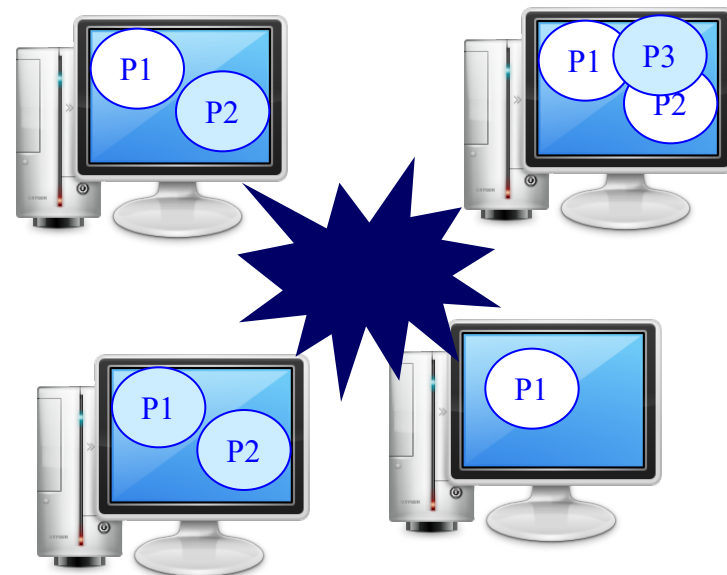
Introducción



Un computador con más
de un proceso (algunos
con más de un hilo)

Si hay concurrencia dependiente entre procesos o entre hilos **de un mismo computador** se aplican técnicas basadas en:

➤ Variables compartidas: **semáforos**, **monitores**, ...



Más de un computador con más de
un proceso cada uno (algunos con
más de un hilo)

Si hay concurrencia dependiente entre procesos **de distintos computadores**, se aplican técnicas basadas en:

➤ Paso de mensajes a través de **sockets**, **RPC** o **RMI**