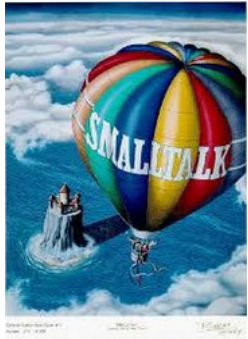




Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue



Programación Concurrente



*Fundamentos de la
Concurrencia*



Temario

- Programas, procesos y concurrencia
- Beneficios de la programación concurrente
- Arquitecturas hardware
- Características de los sistemas concurrentes
- Lenguajes concurrentes

¿Qué es concurrencia?

Según Real Academia Española: <Concurso de varios sucesos en un mismo tiempo>

- La concurrencia es la capacidad de ejecutar un **conjunto de actividades** en **forma simultánea**.
- Permite a distintos objetos actuar al mismo tiempo
- Para la Ciencia de la Computación es relevante para el diseño de hardware, SO, multiprocesadores, computación distribuida, programación y diseño.

En informática, cada una de esas actividades se suele llamar **proceso**.

Donde encontramos concurrencia?

“En todos lados”

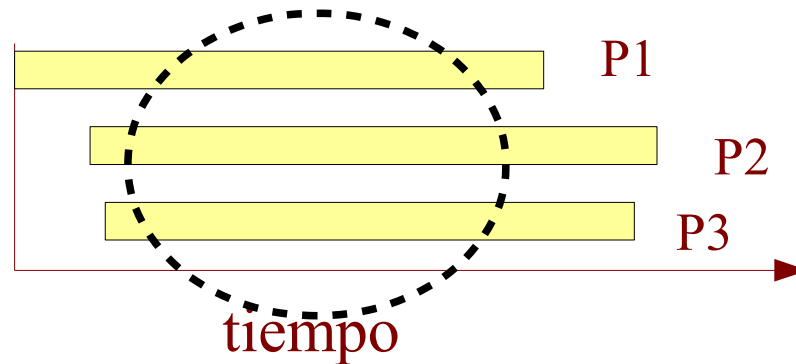
- Cualquier sistema más o menos “inteligente” ...
 - Desde un teléfono hasta un automóvil
 - Navegador Web accediendo información mientras atiende al usuario
 - Varios usuarios conectados al mismo sistema (ej, haciendo una reserva)
 - Juegos

Sistemas de naturaleza concurrente

- **Sistemas de control:** Captura de datos, análisis y actuación (sistemas de tiempo real).
- **Servidores web** que son capaces de atender varias peticiones concurrentemente, servidores de chat, email, etc.
- **Aplicaciones basadas en GUI:** El usuario hace varias peticiones a la aplicación gráfica (Navegador web).
- **Simulación**, o sea programas que modelan sistemas físicos con autonomía.
- Sistemas **Gestores de Bases de Datos**.
- Sistemas **operativos** (controlan la ejecución de los usuarios en varios procesadores, los dispositivos de E/S, etc)

Concurrencia

- Es la **existencia simultánea** de varios procesos en ejecución.
- Dos **procesos** son **concurrentes** cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última



Objetivos de los sistemas concurrentes

- **Ajustar el modelo** de arquitectura de hardware y software al problema del mundo real a resolver.
- **El mundo real ES CONCURRENTE**
- **Incrementar la performance**, mejorando los tiempos de respuesta de los sistemas de procesamiento de datos, a través de un enfoque diferente de la arquitectura física y lógica de las soluciones.
- Se puede mejorar la velocidad de ejecución, mejor utilización de la CPU de cada procesador, y explotación de la concurrencia inherente a la mayoría de los problemas reales.

Qué es un proceso?

- **Programa secuencial:** un solo flujo de control que ejecuta una instrucción y cuando esta finaliza ejecuta la siguiente
PROCESO: programa secuencial
- Un único thread o flujo de control
→ programación secuencial, monoprocesador
- Múltiples threads o flujos de control
 - programa concurrente
 - procesos paralelos
- Los procesos *cooperan y compiten...*

Incumbencias de la PC

- Los procesos pueden “competir” o colaborar entre sí por los recursos del sistema.
- Por tanto, incluyen las tareas de colaboración y sincronización.

- La programación concurrente (PC) se encarga del estudio de las nociones de ejecución concurrente, así como de sus **problemas de comunicación y sincronización**.

Ejemplo Cooperación y competición



- Procesos independientes
 - Raros, poco interesantes

Competencia

Típico en SO y redes debido a recursos compartidos



Problemas:

Deadlock
(interbloqueo)

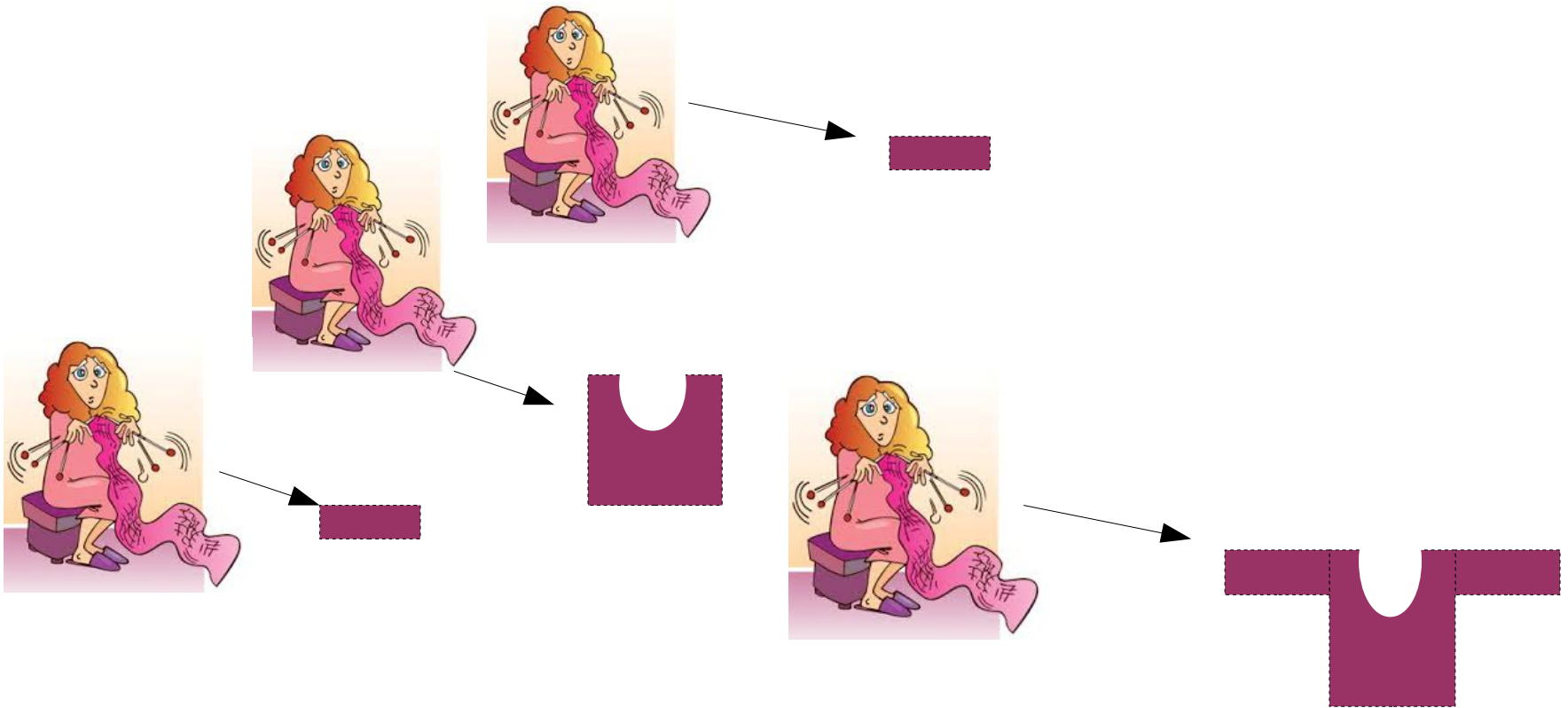


Starvation
(inanición)



Cooperación

- Los procesos se combinan para resolver un problema común
- Sincronización

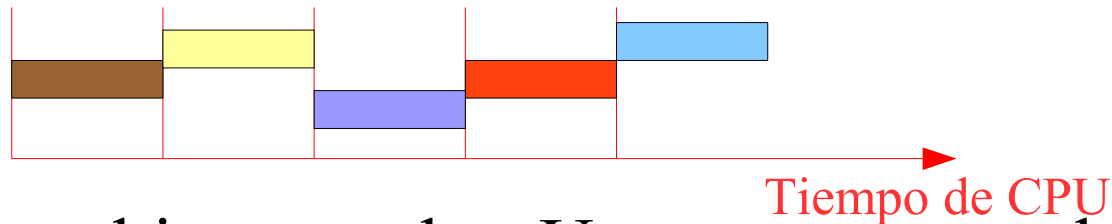


Ventajas y desventajas de la PC

- **Ventaja:** Velocidad de ejecución. Al subdividir un programa en procesos, éstos se pueden “repartir” entre procesadores o gestionar en un único procesador según importancia.
- **Desventaja:** Más complicado de programar.

Hardware

- Sistema monoprocesador: La concurrencia se produce gestionando el tiempo de procesador para cada proceso.



- Sistemas multiprocesador: Un proceso en cada procesador

- Con memoria compartida (procesamiento paralelo)

Fuertemente acoplados

- Memoria local a cada procesador (sist.distribuidos)

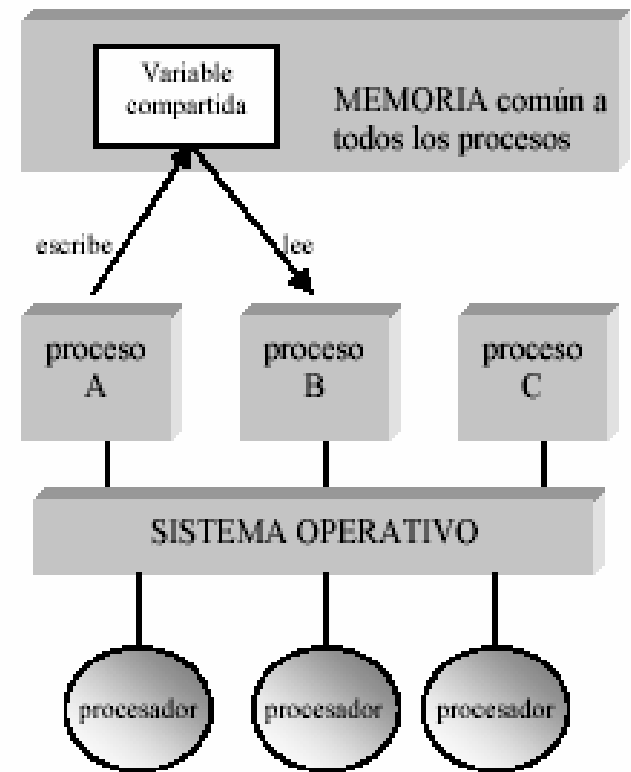
Debilmente acoplados

Sistemas estrechamente acoplados (Multiprocesadores)

La sincronización y comunicación entre procesos se suele hacer mediante variables compartidas

Procesadores comparten memoria y reloj.

- **Ventaja:** aumento de velocidad de procesamiento con bajo coste.
- **Inconveniente:** Escalable sólo hasta decenas o centenares de procesadores

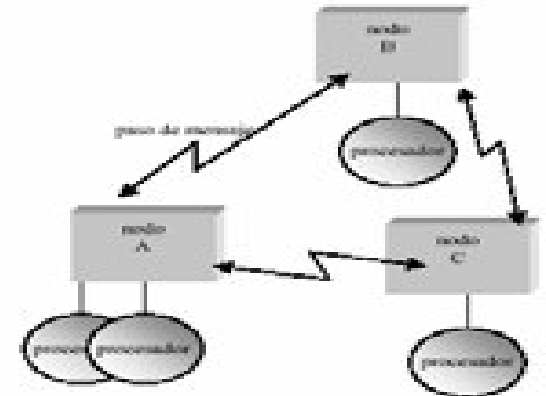


Sistemas débilmente acoplados (Distribuidos)

- Múltiples procesadores conectados mediante una red
- Los procesadores no comparten memoria ni reloj
- Los sistemas conectados pueden ser de cualquier tipo
- Escalable hasta millones de procesadores (ej. Internet)

La forma natural de comunicar y sincronizar procesos es mediante el uso de paso de mensajes.

- **Ventaja:** compartición de recursos dispersos, aumento de velocidad de ejecución, escalabilidad ilimitada, mayor fiabilidad, alta disponibilidad



Lenguajes de programación

Incorporan características que permiten expresar la concurrencia directamente.

Incluyen mecanismos para ...

Ejemplos: Python, **Java**, Smalltalk,

Cómo expresar la concurrencia?

- Las técnicas para producir actividades concurrentes pueden ser:
 - Manuales: Utilizando llamadas al SO o con **bibliotecas de software.**
 - Automáticas: Las detecta el SO en forma automática

Con estas
trabajaremos

Concurrencia en Java

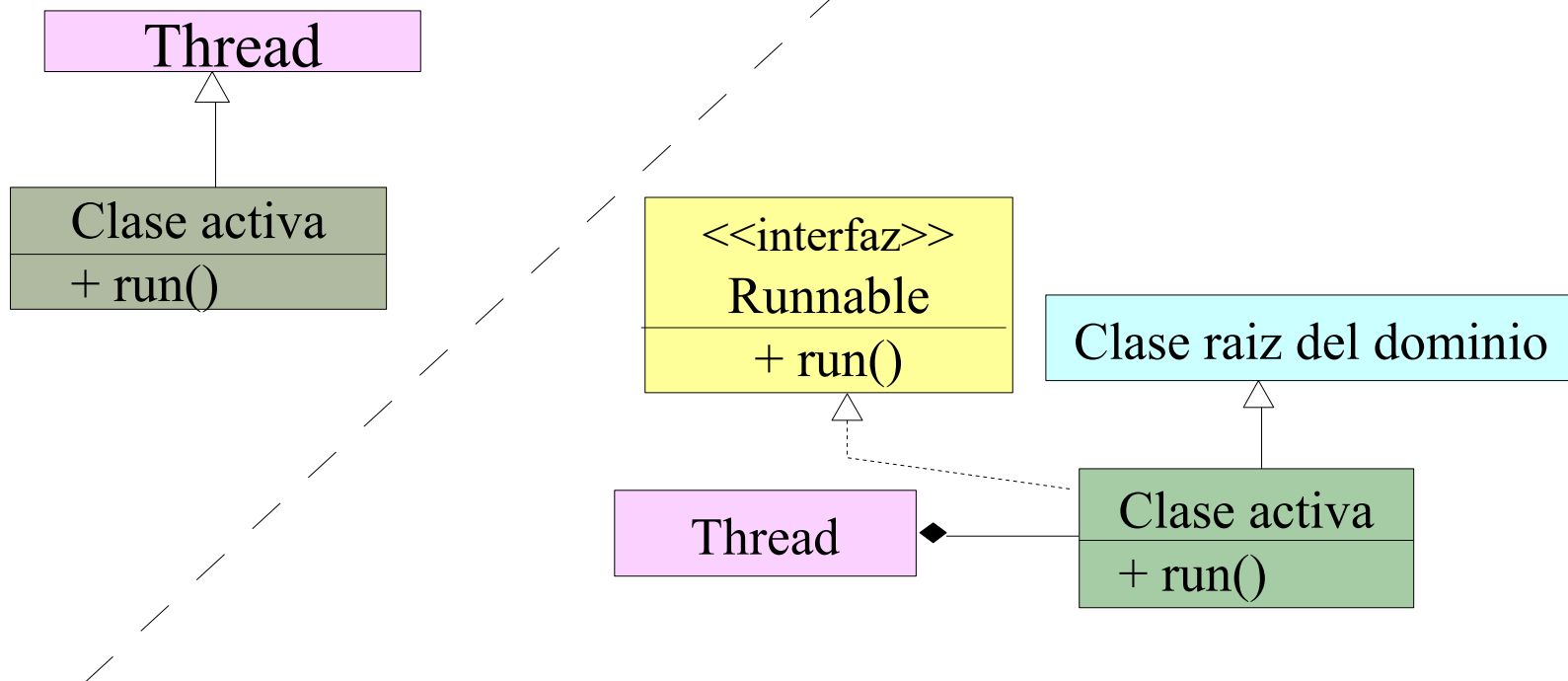
La unidad de concurrencia es el Thread, que comparte el mismo espacio de variables con los restantes threads

Java proporciona manejo de sincronización entre hilos y acceso de forma segura a los objetos compartidos.

Utiliza la clase Thread.

¿Cual conviene más y por qué?

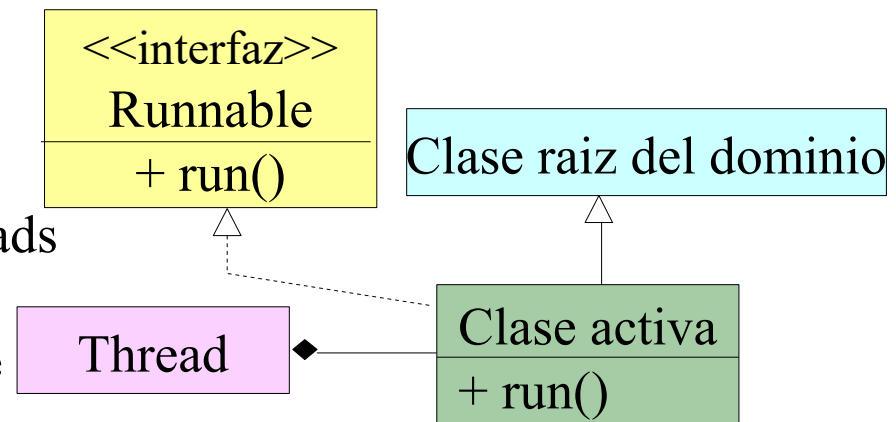
- Ventajas y desventajas:



¿Por qué?

- Esta posibilidad crea un thread a través de la utilización de un objeto que implementa la interfaz Runnable, y con la que se incorpora el método run().
- De esta manera la clase MiClase debe implementar el método run().

- En el programa que declara el nuevo thread, se debe declarar primero el objeto t1 de la clase MiClase, y posteriormente cuando se crea el threads (se instancia el objeto t1 de la clase Thread) se le pasa como parámetro de su constructor.



- Este es el procedimiento mas habitual de crear threads en java, ya que permite pseudo-herencia múltiple (herencia y utilización de interfaz).

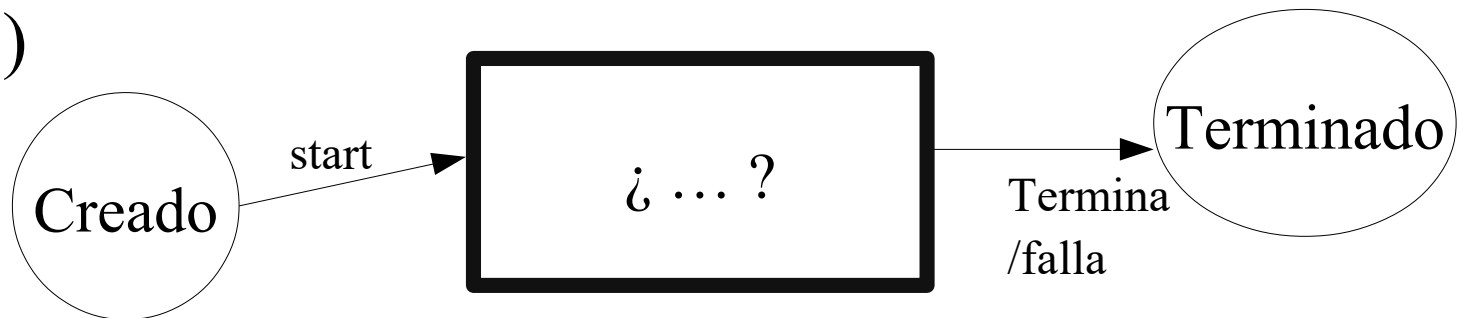
Multitarea en Java: clase Thread

- Un hilo se crea **en Java** instanciando la clase **Thread**.
- El código que ejecuta un thread está definido por el método **run()** que tiene todo objeto que sea instancia de la clase Thread.

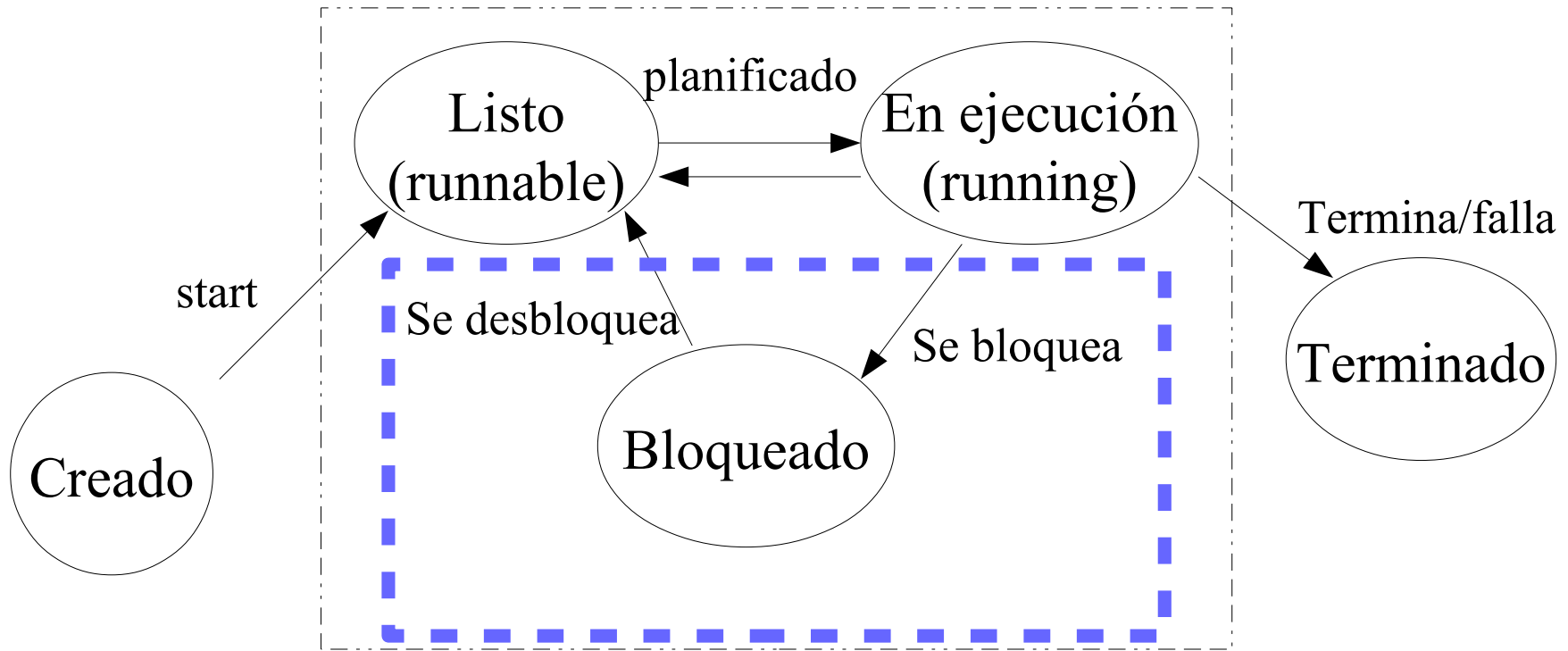
Multitarea en Java: clase Thread

- La ejecución del thread **se inicia** cuando sobre el objeto Thread se ejecuta el método **start()**.
- De forma natural, un **thread termina** cuando en **run()** se alcanza una sentencia **return** o el final del método.

(...)

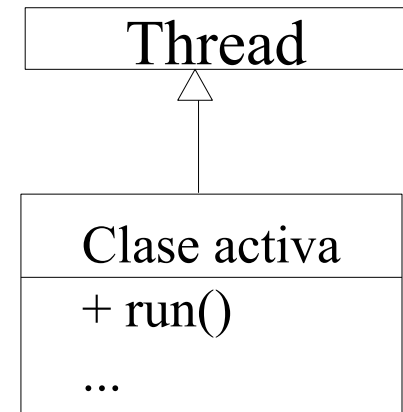


Estados de un hilo



Una forma de crear un hilo

- Crear una *subclase* de Thread
- Implementar el método *run()* con el comportamiento deseado (el método de la clase thread no hace nada)
- *Crear objetos* de esa subclase y activarlos con *start()*.



Crear un hilo por herencia

¿Cuántos hilos hay en ejecución/espera después de t1.start()?

```
public static void main(String[] args){
    PingPong t1 =new PingPong(...);

    // Activación
    t1.start();
    .....

}
```

```
public class PingPong extends Thread{
    // variables propias ...
    // constructor
    public PingPong(...){
        .....
    };

    public void run(){
        // hace algo ...
    }

} //fin clase PingPong
```

¿Cuál termina su ejecución primero?

Crear un hilo por herencia

```
public static void main(String[] args){  
    PingPong t1 = new PingPong(...);  
  
    // Activación  
    t1.start();  
    .....  
  
    t1.join();  
}
```

Espera a que t1
termine su ejecución

```
public class PingPong extends Thread{  
    // variables propias ...  
    // constructor  
    public PingPong(...){  
        .....  
    };  
  
    public void run(){  
        // hace algo ...  
    }  
  
} //fin clase PingPong
```

¿y ahora, ... cuál termina su
ejecución primero?

Características algoritmos concurrentes

- **Rapidez**

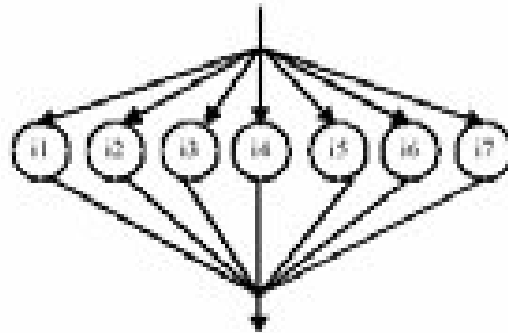
- Problemas arreglo de muchos elementos,
- Operaciones matemáticas $+/^*$
- Contrastar versión concurrente + versión no concurrente
- Utilizar la cantidad de procesadores de la máquina virtual

Constructores de la clase Thread

- **Thread()**
- **Thread**(Runnable threadOb)
- **Thread**(Runnable threadOb, String threadName)
- **Thread**(String threadName)

Un problema propio de la Prog Concurrente

Indeterminismo: Un programa concurrente define un orden **parcial** de ejecución. Ante un conjunto de datos de entrada no se puede saber cual va a ser el flujo de ejecución



Los programas concurrentes pueden producir diferentes resultados en ejecuciones repetidas sobre el mismo conjunto de datos de entrada

Características algoritmos concurrentes

- Rapidez
- Indeterminismo
 - Creamos 2 hilos con nombres diferentes
 - Mostramos una salida con el nombre del hilo procesando

Escenario indeterminístico

- Planificador (scheduler) indeterminístico

```
public class RunThread implements Runnable{  
    public void run() {  
        for (int i=0; i <30; i++){  
            String threadNombre = Thread.currentThread().getName();  
            System.out.println(threadName + "en ejecucion");  
        }  
    }  
}
```

```
public class TestRunThread {  
    ...
```

```
    public static void main (String[] args) {  
        RunThreads runner = new RunThreads();
```

```
        Thread alfa = new Thread (runner);  
        Thread beta = new Thread (runner);
```

```
        alfa.setName ("Hilo Alfa");  
        beta.setName ("Hilo Beta");
```

```
        alfa.start(); beta.start();
```

```
    }  
}
```

instancia Runnable

dos hilos con la
misma implementación
runnable

nombra los hilos

los hilos pasan a estado "runnable/listo"

¿Cual es la salida?

Hilo Alfa en ejecucion
Hilo Alfa en ejecucion
Hilo Alfa en ejecucion
Hilo Alfa en ejecucion
Hilo Beta en ejecucion
Hilo Alfa en ejecucion

..

Posible salida

Problemas de la PC

- Es más difícil analizar y verificar un algoritmo concurrente por el no determinismo.
- Ojo, que existan varias posibilidades de salida NO significa necesariamente que un programa concurrente sea incorrecto

```
proceso P1;  
    var i: integer;  
begin  
    for i:=1 to 5 do x:= x+1;  
end;  
  
proceso P2;  
    var j: integer;  
begin  
    for j:=6 to 15 do x:= x+1;  
end;
```

```
begin  
    x= 0;  
    cobegin  
        P1;  
        P2;  
    coend  
end;
```

¿Cual es la salida?

Entonces...conurrencia...

- **Proceso:** secuencia de acciones que se realizan independientemente de las acciones realizadas por otros procesos.

En general, la prioridad la asigna el SO!!!!

- Los **procesos** que se ejecutan en paralelo, **son objetos** que poseen una prioridad, la cual puede asignarse en un principio e ir modificándose a lo largo de la ejecución.

- La **prioridad** de un proceso describe la importancia que tiene ese proceso por sobre los demás.

Problemas en lenguajes de alto nivel

- Una instrucción de alto nivel se convierte en un conjunto de instrucciones máquina
- Las **instrucciones de máquina** son las que se ejecutan **concurrentemente**
- En **ejecuciones concurrentes** el resultado puede ser **incorrecto**
 - Diferentes posibilidades en cuanto al **orden de ejecución**
 - Pueden ser necesarias **ciertas restricciones** al orden de **ejecución**
- Generalmente existen partes de código con **variables compartidas** y que deben ejecutarse en **exclusión mutua**

Programas correctos

- (PS) Programa secuencial:
 - **Parcialmente correcto:** Dadas unas precondiciones correctas, **si** el programa termina **entonces** se cumplen las postcondiciones
 - **Totalmente correcto:** Dadas unas precondiciones correctas **el programa termina y** se cumplen las postcondiciones
- (PC) Programación concurrente:
 - Pueden **no terminar** nunca y ser **correctos**.
 - Puede tener **múltiples secuencias de ejecución**
 - Cuando es correcto es porque se refiere a **todas** sus posibles secuencias de ejecución.

Cómo expresar la concurrencia?

- Las técnicas para producir actividades concurrentes pueden ser:
 - Manuales: Utilizando llamadas al SO o con **bibliotecas de software.**
 - Automáticas: Las detecta el SO en forma automática

Con estas
trabajaremos

Lenguajes concurrentes

Incorporan características que permiten expresar la concurrencia directamente.

Incluyen mecanismos de sincronización y comunicación entre procesos

Ejemplos: Ada, Python, **Java**, Smalltalk,
.....

Cómo expresar la concurrencia

- Sentencia concurrente:

cobegin

P; Q; R

coend;

- Objetos que representan procesos:

- **task A is** begin P; end;

- **task B is** begin Q; end;

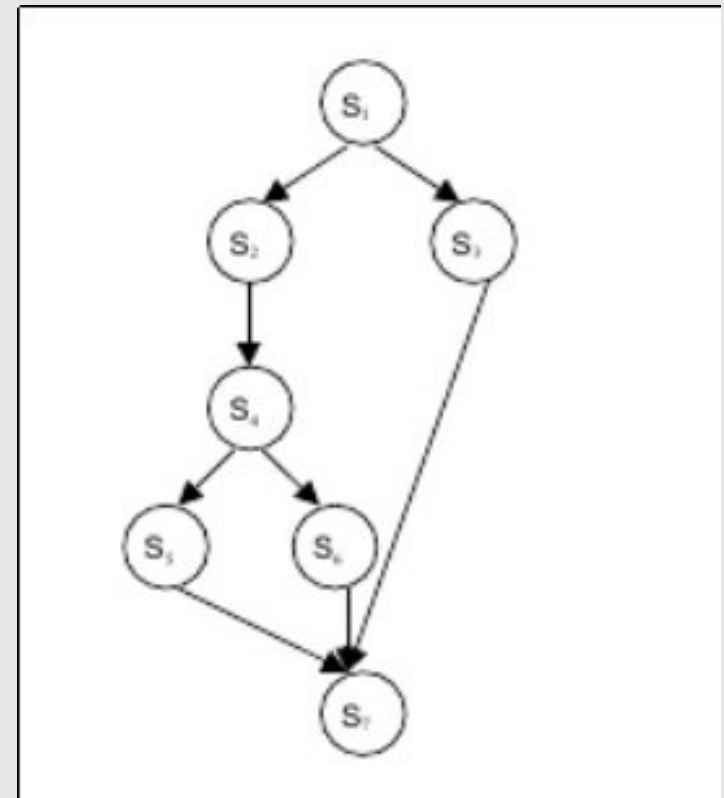
- **task C is** begin R; end;

- Sentencia concurrente múltiple:

forall i:=1 to 1000 do P(i);

- Notación gráfica:

- Grafos de precedencia

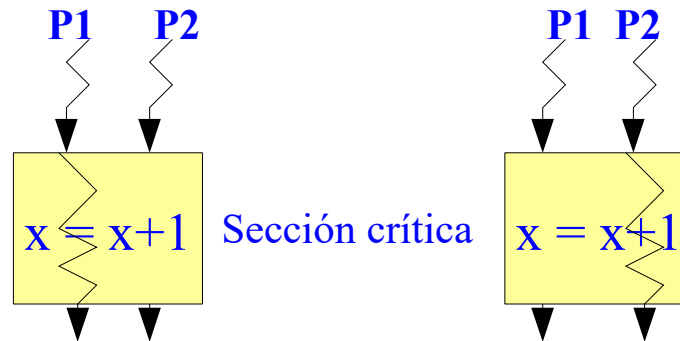


Algoritmos concurrentes

- Rapidez
- Indeterminismo
- Orden de ejecución de los hilos
 - Creamos 4 hilos con nombres diferentes
 - Mostramos el orden en que se van procesando

Problemas de la PC

- Exclusión mutua
 - Sección crítica: porción de código con variables compartidas y que debe ejecutarse en exclusión mutua



- Condición de sincronización
- Verificación

Sincronización - Java

- Java utiliza *Synchronized* se utiliza para sincronizar objetos.
- El bloque synchronized lleva entre paréntesis la referencia a un objeto.
- Cada vez que un thread intenta acceder a un bloque sincronizado le pregunta a ese objeto si no hay algún otro thread ejecutando algún bloque sincronizado con ese objeto
- Otro thread ha realizado el bloqueo (lock), entonces el thread actual es suspendido y puesto en espera hasta que el lock se libere.
- Si el lock está libre, entonces el thread actual bloquea (lock) el objeto y entra a ejecutar el bloque.
- El lock se libera cuando el thread que lo tiene tomado sale del bloque por cualquier razón: termina la ejecución del bloque normalmente, ejecuta un return o lanza una excepción.
- El lock es sobre un objeto en particular.
- Si hay dos bloques synchronized que hacen referencia a distintos objetos, la ejecución de estos bloques **no** será mutuamente excluyente.

Synchronized en Java

- Usar synchronized en un método de instancia es lo mismo que poner un bloque de `synchronized(this){}` que contenga todo el código del método.

Es lo mismo:

```
public synchronized void metodo() {  
    // código del método aca  
}
```

```
public void metodo() {  
    synchronized(this) {  
        // código del método aca  
    }  
}
```

- Si el método es de clase entonces es lo mismo pero el bloque de `synchronized` se aplica a la clase.

Ejemplo, si el método está en la clase `MiClase`

```
public static synchronized void metodo() {  
    // código del método aca  
}
```

```
public static void metodo() {  
    synchronized(MiClase.class) {  
        // código del método aca  
    }  
}
```

Temario

- Problemas de la PC
- Correctitud
 - Propiedades de seguridad (safety)
 - Propiedades de viveza (liveness)
- Procesos
 - Thread (Java)
 - fork (Smalltalk)

PC-Correctos

- Tipos de propiedades de la PC
 - Propiedades de seguridad (**safety**)
 - Son aquellas que aseguran que nada malo va a pasar durante la ejecución del programa
 - Exclusión mutua
 - Condición de sincronización
 - Interbloqueo (pasivo) - **deadlock**
 - Propiedades de viveza (**liveness**)
 - Son aquellas que aseguran que algo bueno pasará eventualmente durante la ejecución del programa
 - Interbloqueo (activo) – **livelock**
 - Inanición - **starvation**

Recordemos...conurrencia

- **Proceso:** secuencia de acciones que se realizan independientemente de las acciones realizadas por otros procesos.
- Los **procesos** que se ejecutan en paralelo, **son objetos** que poseen una prioridad, la cual puede asignarse en un principio e ir modificándose a lo largo de la ejecución.
- La **prioridad** de un proceso, describe la importancia que tiene ese proceso por sobre los demás.

Concurrencia en Smalltalk

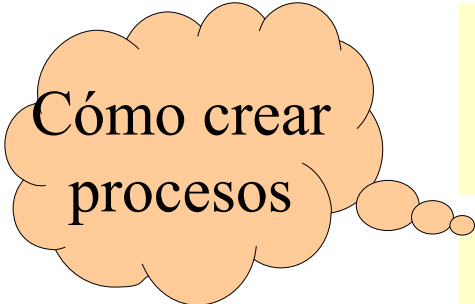
Clase: BlockClosure

(categoría: scheduling)
métodos

fork, Crea y organiza el cuyo código del proceso corriendo en el receptor – corre en forma concurrente.

forkAndWait, Suspende el proceso actual y ejecuta el código en un nuevo proceso, cuando se complete el ***resume*** proceso actual.

forkAt: valorPrioridad, Crea y organiza el proceso en el receptor con una prioridad dada por valorPrioridad. Retorna el nuevo proceso creado.



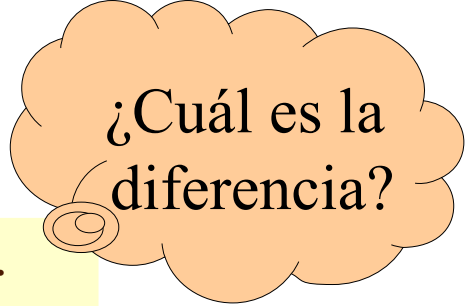
Cómo crear procesos

```
[... "algunas sentencias"  
Transcript show: 'Proceso'] fork.
```

```
| bloqueAcciones proceso |  
bloqueAcciones := [Transcript show: 'Proceso'].  
proceso := bloqueAcciones fork.
```

Procesos en Smalltalk

- Ejecutar



¿Cuál es la diferencia?

```
Transcript show: 'Comienzo ejercicio PC - '.  
(Delay forMilliseconds: 10000) wait.  
Transcript show: 'Termina ejercicio PC'.
```

```
Comienzo ejercicio PC - Termina ejercicio PC
```

```
[Transcript show: 'Comienzo ejercicio PC - '.  
(Delay forMilliseconds: 10000) wait.  
Transcript show: 'Termina ejercicio PC'.] fork.
```