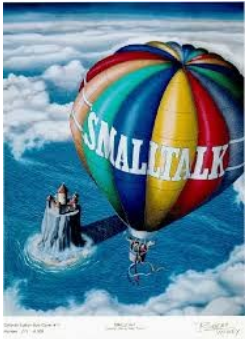




Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue



Programación Concurrente



*Fundamentos de la
Concurrencia*



Bibliografía

- En Biblioteca:
 - Concurrent Programming in Java: Design Principles and Patterns, *Doug Lea*. Addison Wesley - 1997
 - Orientación a Objetos con Java y UML, *Carlos Fontella*, Nueva Libreria S.R.L.
 - Java Concurrency in Practice, *Brian Goetz et all.*, Pearson Education , Inc. 2006
 - Thinking in JAVA . *Bruce Eckel*. President, MindView Inc – 2008
 - A Brain-Friendly Guide – Head First Java -*Katty Sierra & Bert Bates*
 - High Performance JAVA Platform Computing. Multithread and Networked Programming. The Sun Microsystems Press Java Series. *Thomas W.Christopher* – *George K. Thiruvathukal*,
- Otros:
 - Sun Educational Service, El lenguaje de Programación Java, Sun Microsystems. Inc – *Guía del Estudiante*

Situaciones reales

- Al levantarse
- Al ponerse a estudiar...
- Llevar adelante varias materias.....

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: <Acaecimiento o concurso de varios sucesos en un mismo tiempo>

- La concurrencia es como un **conjunto de actividades** que se desarrollan de **forma simultánea**.
- En informática, cada una de esas actividades se suele llamar **proceso**.

Programa vs. Proceso

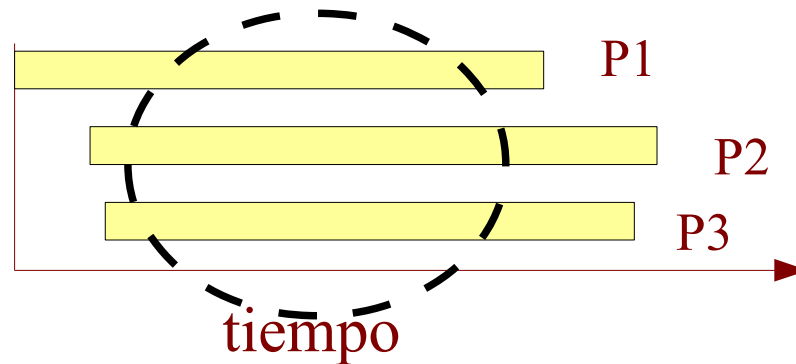
- **Programa:** Conjunto de sentencias/instrucciones que se ejecutan secuencialmente. Concepto **estático**.
- **Proceso:** Básicamente, se puede definir como un programa en ejecución. Líneas de código en ejecución de manera **dinámica**.

Se asemeja al concepto de **Clase** de la POO

Se asemeja al concepto de **Objeto** de la POO

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



Ejemplo Cooperación y competición

- Torta 2 bols
-
- Torta 1 bol

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**.

Ventajas y desventajas de la PC

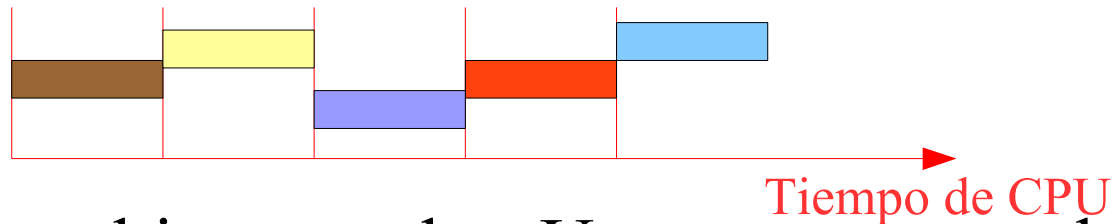
- **Ventaja:** Velocidad de ejecución. Al subdividir un programa en procesos, éstos se pueden “repartir” entre procesadores o gestionar en un único procesador según importancia.
- **Desventaja:** Más complicado de programar.

Problemas concurrentes

- **Problemas** 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)

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)

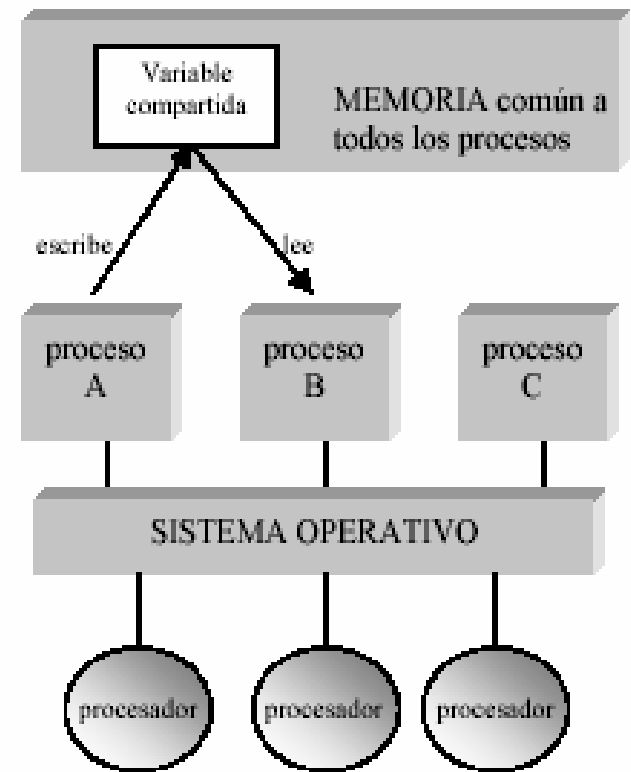
Debilmente acoplados

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

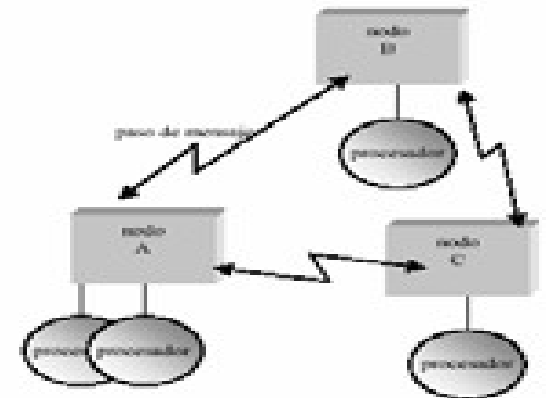


Sistemas débilmente acoplados (Distribuidos)

- 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)

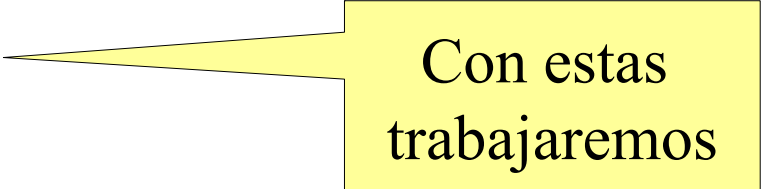
La forma natural de comunicar y sincronizar procesos es mediante el uso de paso de mensajes.

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



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;

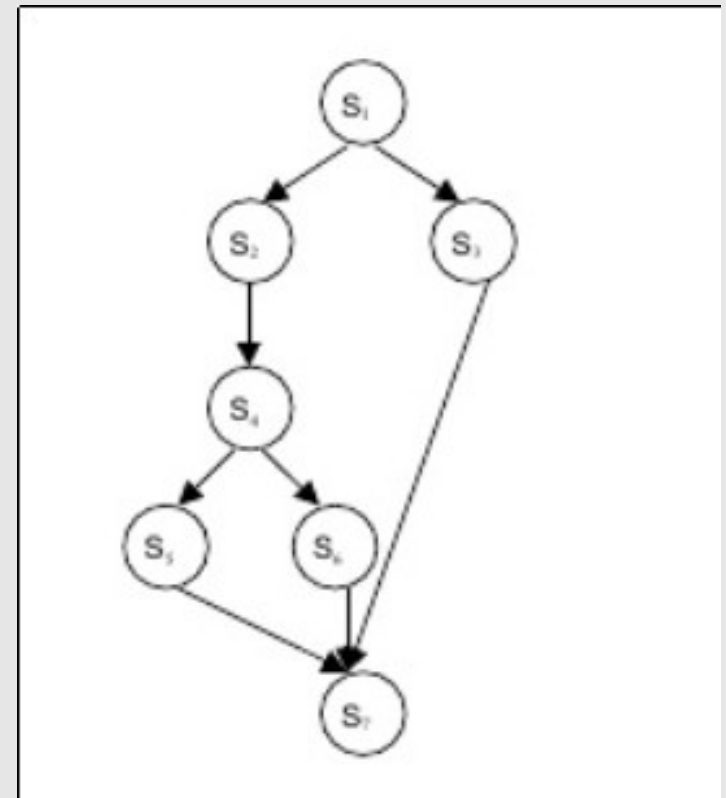
- **task C is** begin R; end;

- Sentencia concurrente múltiple:

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

- Notación gráfica:

- Grafos de precedencia



Volvamos a los problemas

- Calefón y agua ...
- Cuarto oscuro...
- Impresión en una organización...

Concurrencia en Java

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

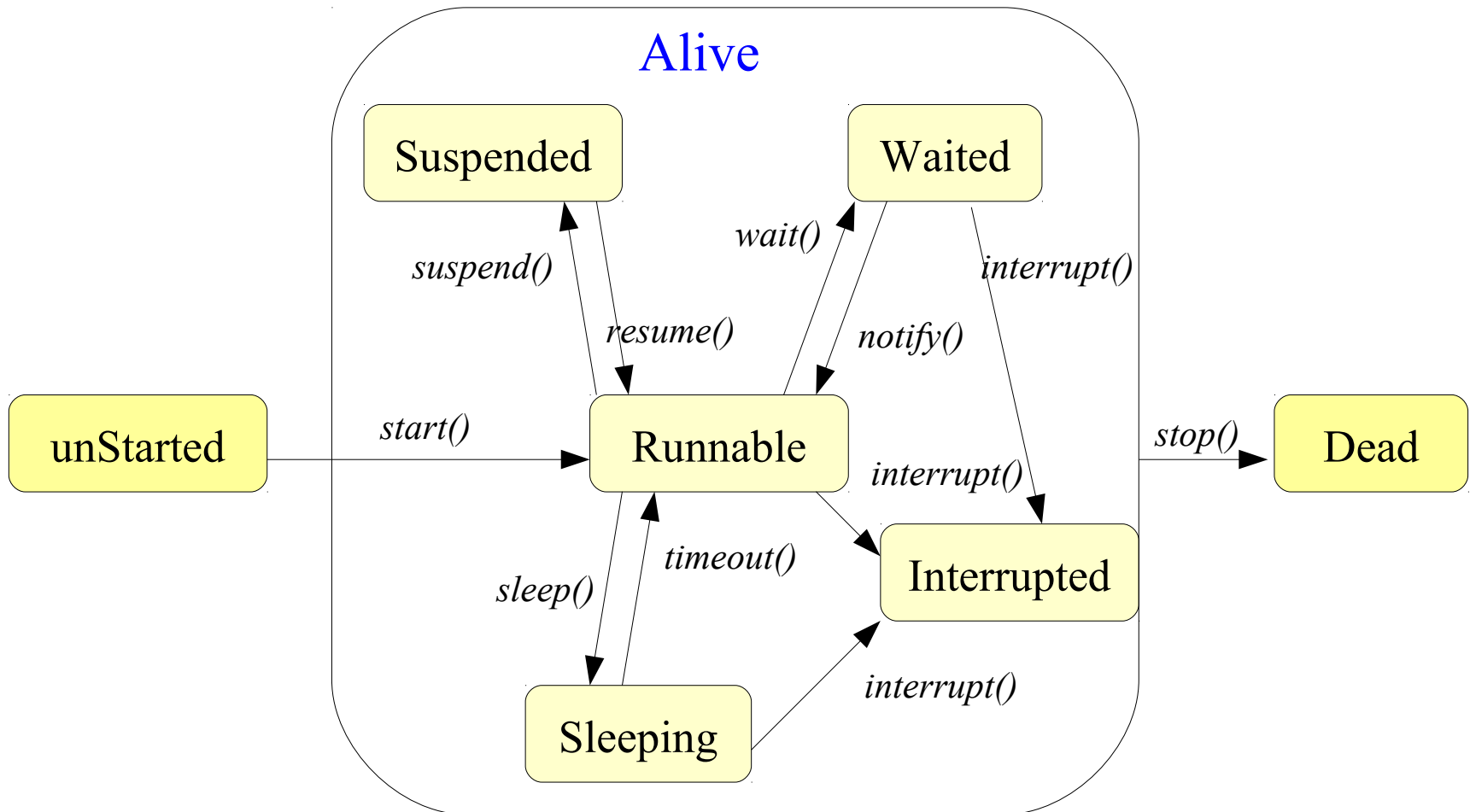
Java proporciona manejo de sincronización entre hilos y acceso de forma segura a los objetos compartidos.

Utiliza la clase Thread.

Multitarea en Java: clase Threads

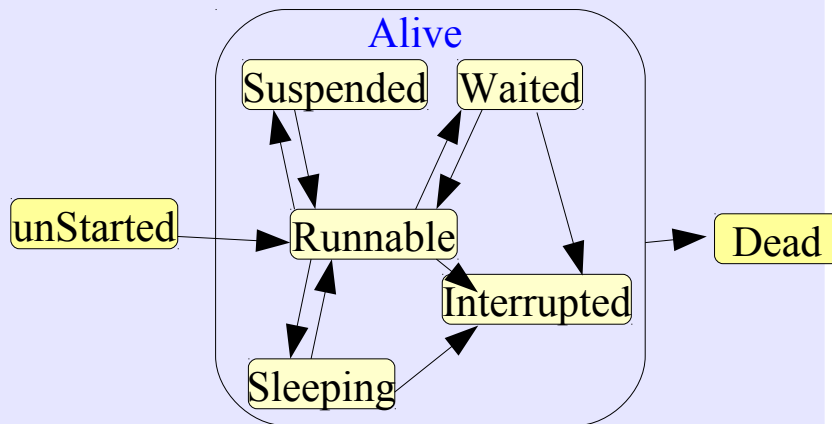
- Un thread se crea **en Java** instanciando un objeto de la clase **Thread**.
- El código que ejecuta un thread está definido por el método **run()** que tiene todo objeto que sea instancia de la clases Thread.
- La ejecución del thread **se inicia** cuando sobre el objeto Thread se ejecuta el método **start()**.
- De forma natural, un **thread termina** cuando en **run()** se alcanza una sentencia **return** o el final del método.
(Existen otras formas de terminación forzada)

Estados del Thread



Estados del Thread

- “UnStarted”: El thread ha sido instanciado pero no comenzó
- “Alive”: Se ejecuta el método `run()`.



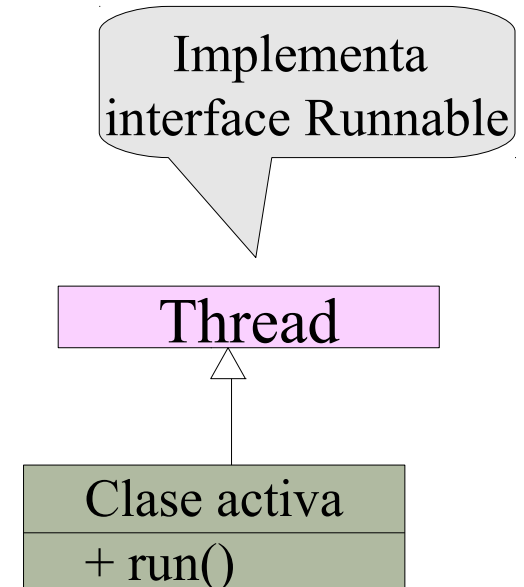
- “Dead”: El thread ha finalizado, por:
 - - El método `run()` ejecuta la sentencia `return` o finaliza.
 - - Desde un thread activo se invoca `stop()`.
 - - No se recupera la excepción *InterruptedException* estando en *Waiting* o *Sleeping*.

Subestados de Alive

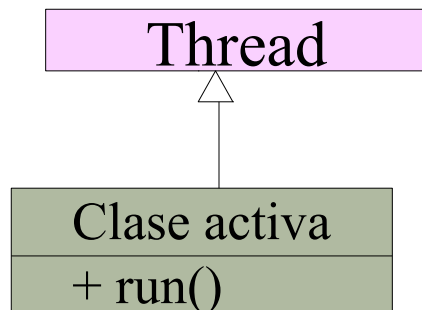
- “Runnable”: ejecución planificada por el procesador.
- “Interrupted”: se ha invocado el método `Interrupt` sobre el thread.
- “Suspended”: El thread está suspendido indefinidamente. El thread no está siendo planificado.
- “Sleeping”: El thread está suspendido temporalmente (durante el número de ms establecido por el argumento del método `sleep()` que lo ha dormido).
- “Waiting”: El thread está suspendido indefinidamente (sobre la variable de condición del objeto thread por haber invocado el mismo el método `wait()`).

Una forma de crear un hilo

- Crear una *subclase* de Thread
- Definir la implementación del método *run()* para implementar la interfaz *Runnable*.
- *Crear objetos* de esa subclase y activarlos con *start()*, luego finalizarlos con *stop()*.



Crear un thread por herencia



```
public static void main(String[] args){
    // 2 threads
    PingPong t1 = new PingPong(...);

    // Activación
    t1.start();

    .....
    // Finalización
    t1.stop();
}
```

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

} //fin clase PingPong
```

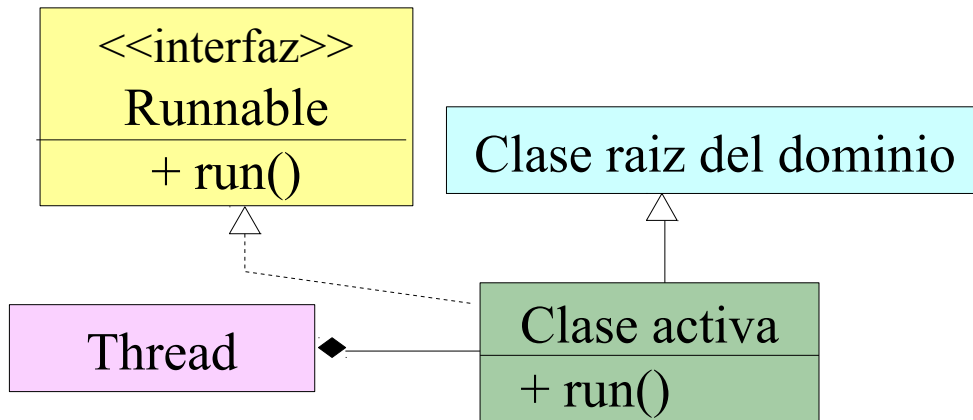

Crear Thread por Interfaz Runnable

- Crear una clase que implemente la interfaz Runnable

```
public interface Runnable){  
    //la provee Java  
    public abstract void run()  
    ....
```

```
public class MiClase implements Runnable){  
    // ....  
    public void run() {  
        .....}  
    ...
```

- Crear objetos de esa clase
- Crear objetos de Thread con objetos de la clase nueva instanciados.

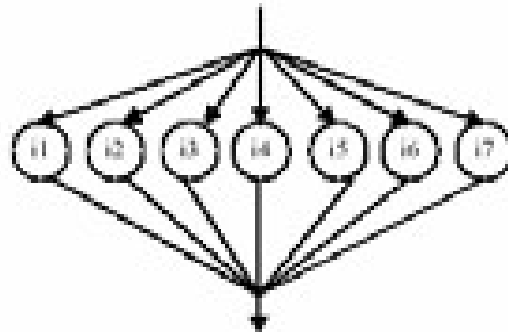


```
.....  
MiClase ot = new MiClase();  
Thread t1 = new Thread(ot);  
t1.start();  
.....
```

Un problema propio de la PC

- 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

Sentencia concurrente:
cobegin
....
coend;



```
begin  
  cobegin  
    i1;i2;i3;i4;i5;i6;i7  
  coend  
end;
```

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

Escenario indeterminístico

- Veamos cómo un manejador (scheduler) puede ser indeterminístico

```
public class MiEjemplo implements Runnable){  
    public void run() {  
        ir()  
    }  
    public void ir() {  
        hacer()  
    }  
    public void hacer() {  
        System.out.println(" thread Mio ");  
    }  
}
```

```
public interface Runnable){  
    //la provee Java  
    public abstract void run()  
    .....  
}
```

```
public class TestMiThread{  
    public static void (String[] args) {  
        MiClase ot = new MiClase();  
        Thread t1 = new Thread(ot);  
        t1.start();  
  
        System.out.println(" vuelta en el main")  
    }  
}
```

- ¿Cual es la salida?

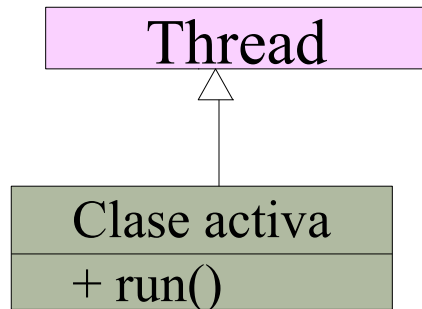
Poner un thread a dormir

- Una de las formas de turnar los threads es utilizar la directiva de dormir (sleep) `Thread.sleep(cantMilisegundos)`
- El método sleep lanza una excepción del tipo InterruptedException, por lo tanto debe hacerse dentro de un try/catch

```
try {Thread.sleep(2000); }  
catch (InterruptedException ex){  
    ex.printStackTrace();  
};
```

```
public class MiEjemplo implements Runnable){  
    public void run() {  
        ir()  
    }  
    public void ir() {  
        //el código va aca  
        hacer()  
    }  
    public void hacer() {  
        System.out.println(" thread Mio ");  
    }  
}
```

Thread: activación, dormida y finalización



```
public static void main(String[] args){
    // 2 threads
    PingPong t1 = new PingPong("PING",33);
    PingPong t2 = new PingPong("PONG",10);
    // Activación
    t1.start(); t2.start();
    // Espera 2 segundos
    try { sleep(5000);
        } catch (InterruptedException e) {};
    // Finaliza la ejecución de los threads
    t1.stop(); t2.stop();
}
```

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

} //fin clase PingPong
```

Crear un thread por Interfaz

- Crear una clase que implemente la interface Runnable

Crear objetos de esa clase

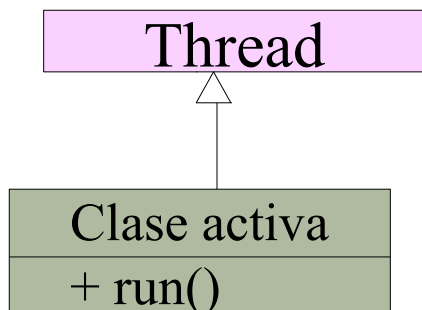
- Crear objetos de Thread con objetos de la clase nueva instanciados.

```
public class PruebaRunnable
public static void main(String[] args){
    // 2 objetos definen los métodos run
    PingPong o1 =new PingPong("PING",33);
    PingPong o2= new PingPong("PONG",10);
    // Se crean los threads
    Thread t1 = new Thread (o1);
    Thread t2 = new Thread (o2);
    // se activan los threads
    t1.start;
    t2.start;
}
}
```

```
public class PingPong extends Thread{
    private String pal; // Lo que va a escribir.
    private int delay; // Tiempo entre escritura

    public PingPong(String cartel,int cantMs){
        pal = cartel;
        delay = cantMs;
    };
    public void run(){ //sobrescribe run()de Thread
        while(true){
            System.out.print(pal + " ");
            try{sleep(delay);}
            catch(InterruptedException e){ return; }
        }
    } //fin método run()
} //fin clase PingPong
```

Crear un thread por herencia



```
public static void main(String[] args){
    // 2 threads
    PingPong t1 = new PingPong("PING",33);
    PingPong t2 = new PingPong("PONG",10);
    // Activación
    t1.start(); t2.start();
    // Espera 2 segundos
    try { sleep(5000);
        } catch (InterruptedException e) {};
    // Finaliza la ejecución de los threads
    t1.stop(); t2.stop();
}
```

```
public class PingPong extends Thread{
    private String pal; // Lo que va a escribir.
    private int delay; // Tiempo entre escritura

    public PingPong(String cartel,int cantMs){
        pal = cartel;
        delay = cantMs;
    };
    public void run(){ //sobrescribe run() de Thread
        while(true){
            System.out.print(pal+ " ");
            try{sleep(delay);}
            catch(InterruptedException e){
                return; }
        }
    } //fin método run()
} //fin clase PingPong
```

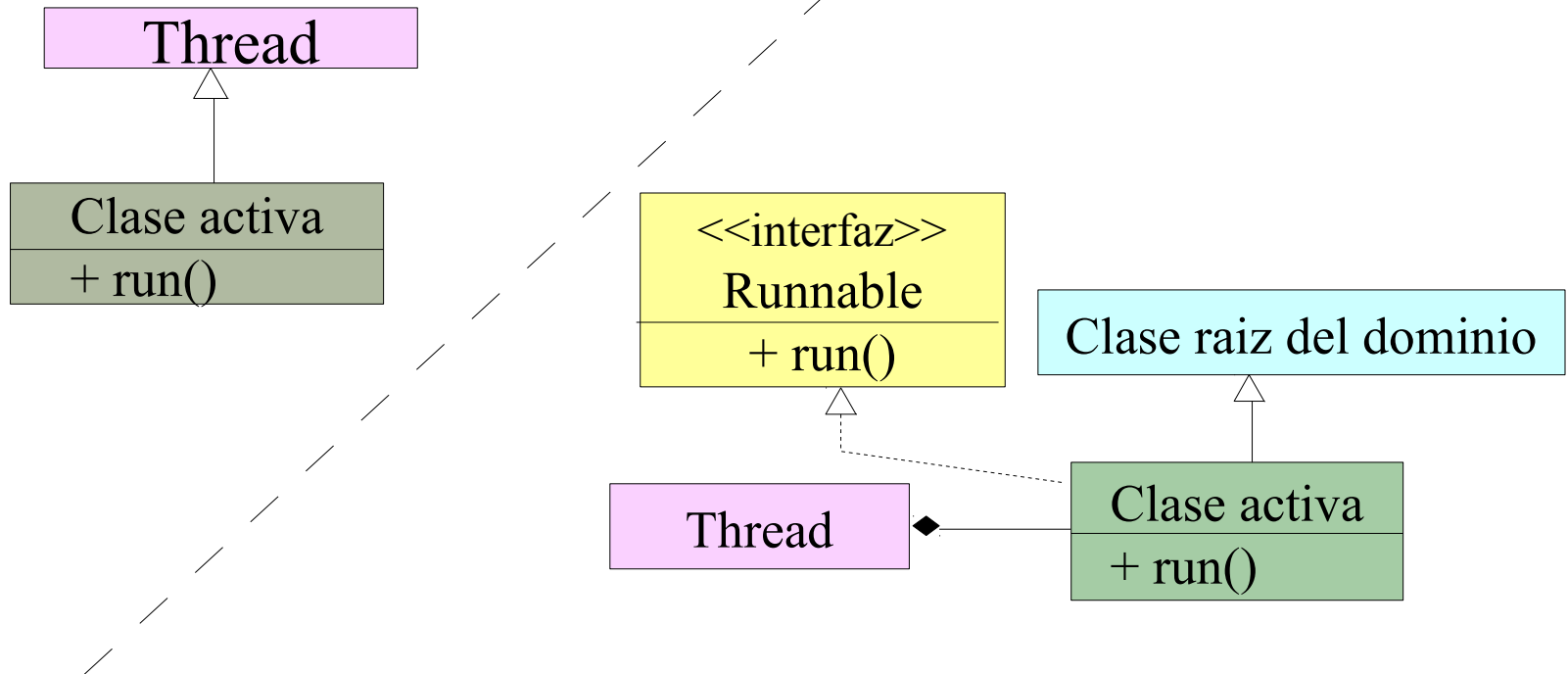
Constructores de la clase Thread

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

- **Thread**(ThreadGroup groupOb, Runnable threadOb)
- **Thread**(ThreadGroup groupOb, Runnable threadOb, String threadName);
- **Thread**(ThreadGroup groupOb, String threadName)

¿Cual conviene más y por qué?

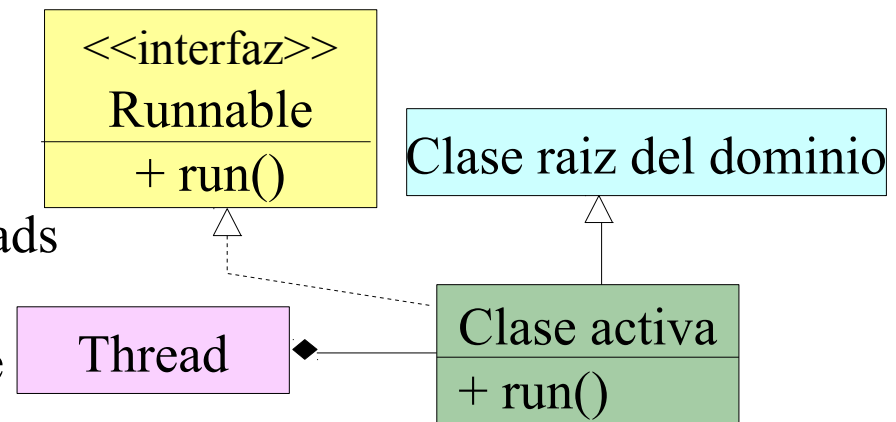
- Ventajas y desventajas:



¿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).

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.

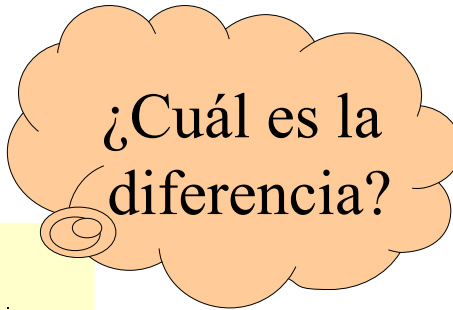
Cómo crear
procesos

```
[... "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.
```