

# Concurrencia: hilos

## Hilos

- Los hilos son objetos, subclases de `Thread` (o implementación de `Runnable`)
- En la clase de un hilo se debe definir el método `run`
- Para ejecutar un hilo:
  - Crear un objeto de la clase del hilo
  - Invocar el método `start`
  - `start` hace que se ejecute el método `run`
  - `run` típicamente consiste en un bucle
- Las clases `Thread` y `Object` proporcionan métodos para gestionar y sincronizar los hilos

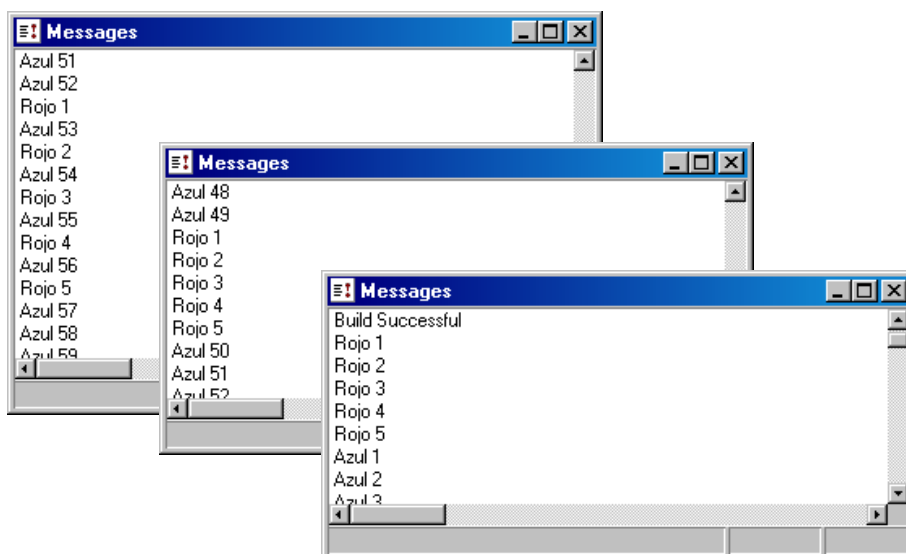
## Creación de un hilo: la clase Thread

```
class Repeticion extends Thread {
    private int repeticiones;
    private String mensaje;
    Repeticion (String msg, int n) {
        mensaje = msg; repeticiones = n;
    }
    public void run () {
        for (int i = 1; i <= repeticiones; i++)
            System.out.println (mensaje + " " + i);
    }
}
```

```
public static void main (String args[]) {
    Repeticion r1 = new Repeticion ("Rojo", 5);
    Repeticion r2 = new Repeticion ("Azul", 80);
    r1.start ();
    r2.start ();
}
```

3

## Ejecución de hilos concurrentes



4

## Creación de un hilo: la interfaz Runnable

```
class Repeticion2 implements Runnable {
    private int repeticiones;
    private String mensaje;
    Repeticion2 (String msg, int n) {
        mensaje = msg; repeticiones = n;
    }
    public void run () {
        for (int i = 1; i <= repeticiones; i++)
            System.out.println (mensaje + " " + i);
    }
}
```

```
public static void main (String args[]) {
    Repeticion r1 = new Repeticion ("Rojo", 5);
    Thread r2 = new Thread (new Repeticion2 ("Azul", 80));
    r1.start ();
    r2.start ();
}
```

5

## Exclusión mutua: métodos `synchronized`

- Evitar interferencias entre hilos que manipulan un mismo objeto
- Métodos `synchronized` de un mismo objeto no pueden ejecutarse a la vez en hilos concurrentes
- El primer hilo que invoca uno de los métodos bloquea el objeto
- Los demás hilos que llamen al método quedan bloqueados mientras tanto
- El primer hilo desbloquea el objeto cuando:
  - Termina
  - Se detiene con `sleep` o `wait`
- En ese momento los demás hilos compiten por continuar
  - Sólo uno lo consigue
  - Los demás vuelven a quedar en espera
- `synchronized` también se puede aplicar a bloques de código

6

## Exclusión mutua: ejemplo

```
class Plaza {
    Pasajero pasajero = null;
    boolean reservada () { return pasajero == null; }
    void reservar (Pasajero p) { pasajero = p; }
}

class Vuelo {
    Plaza plazas[];
    Vuelo (...) { ... }
    synchronized void reservar (Pasajero p) {
        for (int i = 0; i < plazas.length; i++)
            if (!plazas[i].reservada ())
                plazas[i].reservar (p);
    }
}
```

7

```
class TerminalVenta extends Thread {
    public void run () {
        Vuelo v;
        Pasajero p;
        ...
        v.reservar (p);
        ...
    }
}
```

```
public static void main (String args[]) {
    TerminalVenta t1 = new TerminalVenta (...);
    TerminalVenta t2 = new TerminalVenta (...);
    t1.start ();
    t2.start ();
}
```

8

```

class TerminalVenta extends Thread {
    public void run () {
        Vuelo v;
        Pasajero p;
        ...
        // Si reservar no es synchronized
        synchronized (v) {
            v.reservar (p);
        }
        ...
    }
}

```

9

## Coordinación de hilos: ejemplo

```

class Peaton extends Thread {
    Semaforo semaforo;
    Peaton (Semaforo s) { semaforo = s; }
    public void run () {
        while (true) semaforo.setColor ("Verde");
    }
}

```

```

class Controlador extends Thread {
    Semaforo semaforo;
    Controlador (Semaforo s) { semaforo = s; }
    public void run () {
        while (true) semaforo.setColor ("Rojo");
    }
}

```

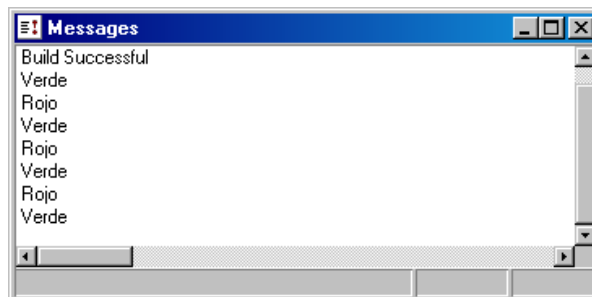
10

```
class Semaforo {  
    private String color = "Rojo";  
    String getColor () { return color; }  
    synchronized void setColor (String str) {  
        try {  
            while (str.equals (color)) wait ();  
            Thread.sleep (2000); // Demora de 2 seg.  
            color = str; System.out.println (color);  
            notifyAll ();  
        } catch (InterruptedException ex) {}  
    }  
}
```

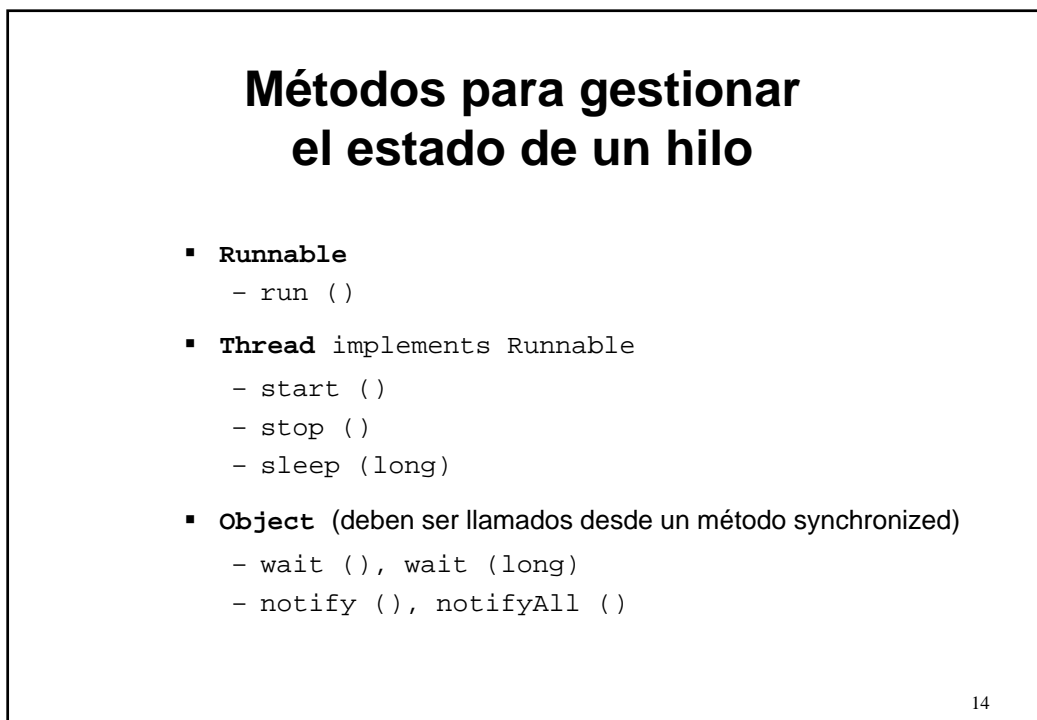
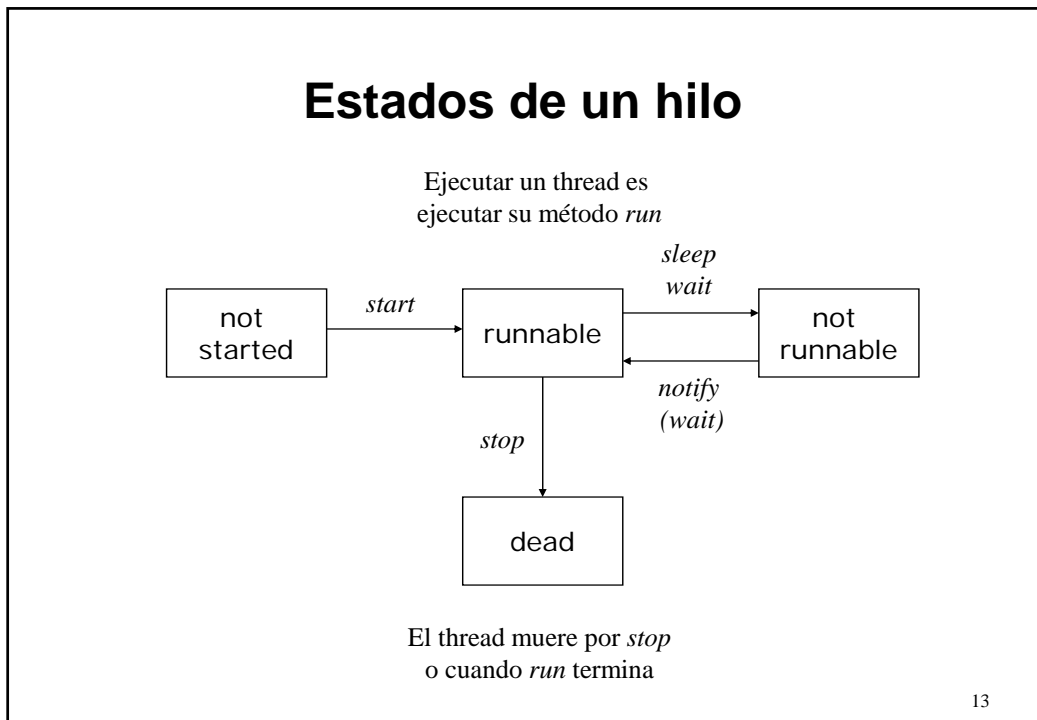
¿Y si se hiciese  
**wait(2000)**?

11

```
public static void main (String args[]) {  
    Semaforo s = new Semaforo ();  
    Controlador c = new Controlador (s);  
    Peaton p = new Peaton (s);  
    p.start ();  
    c.start ();  
}
```



12



## Dormir un hilo: `wait` vs. `sleep`

<code>sleep</code>	<code>wait</code>
<ul style="list-style-type: none"><li>Se invoca sobre un <code>Thread</code> o sobre la clase <code>Thread</code></li></ul>	<ul style="list-style-type: none"><li>Se invoca sobre un <code>Object</code></li><li>Sólo puede ser llamado desde un método <code>synchronized</code></li></ul>
<ul style="list-style-type: none"><li>No se puede despertar hasta que termina</li></ul>	<ul style="list-style-type: none"><li>Puede ser despertado con <code>notify</code> desde otro método <code>synchronized</code> del mismo objeto</li></ul>
<ul style="list-style-type: none"><li>Termina después de un tiempo</li></ul>	<ul style="list-style-type: none"><li><code>wait()</code> termina por <code>notify()</code></li><li><code>wait(n)</code> termina cuando transcurren <code>n</code> mseg. o por <code>notify()</code></li></ul>

15

## Terminar un hilo con `stop`

*Deprecated!*

```
class Hilo extends Thread {  
    public void run () {  
        while (true) {  
            ...  
        }  
    }  
}
```

```
public static void main (String args[]) {  
    Hilo h = new Hilo ();  
    h.start ();  
    ...  
    h.stop ();  
}
```

16

## Terminar un hilo saliendo de run

```
class Hilo extends Thread {  
    boolean seguir;  
    public void run () {  
        seguir = true;  
        while (seguir) {  
            ...  
        }  
    }  
}
```

```
public static void main (String args[]) {  
    Hilo h = new Hilo ();  
    h.start ();  
    ...  
    h.seguir = false;  
}
```

17

## Prioridad de hilos

- Asignación de una prioridad: `setPriority (int)`
  - `Thread.MIN_PRIORITY` 1
  - `Thread.NORM_PRIORITY` 5
  - `Thread.MAX_PRIORITY` 10
- Un hilo deja de ejecutarse sólo cuando:
  - El hilo deja de ser `runnable`
  - Se arranca otro hilo de mayor prioridad
  - En algunos sistemas operativos, cuando el planificador activa otro hilo de igual prioridad
- Dependencia del sistema operativo

18

## Otros métodos de Thread

- `Thread.currentThread()`
- `set/getName(String), toString()`
- `setDaemon(boolean)`
- `yield(), interrupt(), suspend(), resume(), destroy()`
- `isAlive()`
- Otras clases y utilidades para manejo de alto nivel (Java 1.5):  
`Lock, Executor, thread pools, etc.`

19

## Resumen

- Los hilos se pueden crear extendiendo `Thread` o implementando `Runnable` y definiendo `run`
- Los métodos `synchronized` son mutuamente excluyentes entre sí para un mismo objeto
- Con `sleep, wait, notify`, y utilizando prioridades, se coordinan los hilos
- La interacción entre hilos es delicada, no hay una regla general: se resuelve considerando todas las combinaciones posibles
- Es responsabilidad del programador evitar `deadlocks, starvation, o livelocks`

20