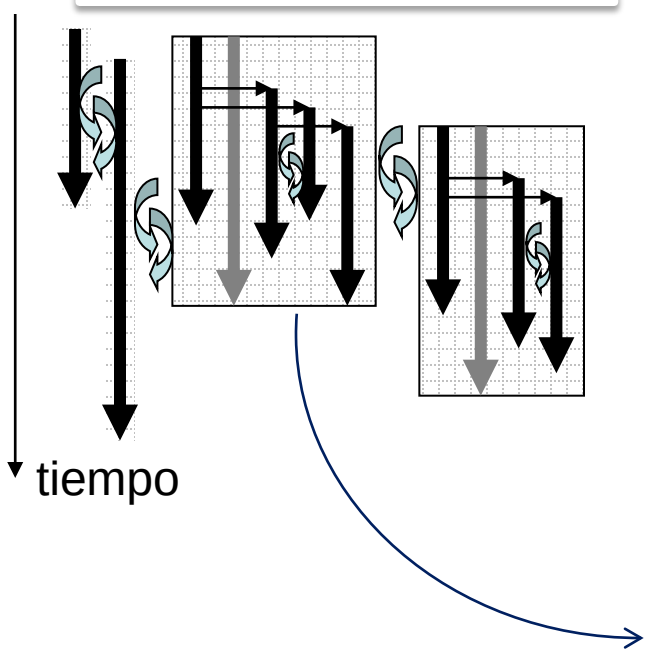
MASTER EN MODELIZACIÓN MATEMÁTICA, ESTADÍSTICA Y COMPUTACIÓN
2011-2012

Curso: Bases de datos y programación orientada a objetos
Parte POO

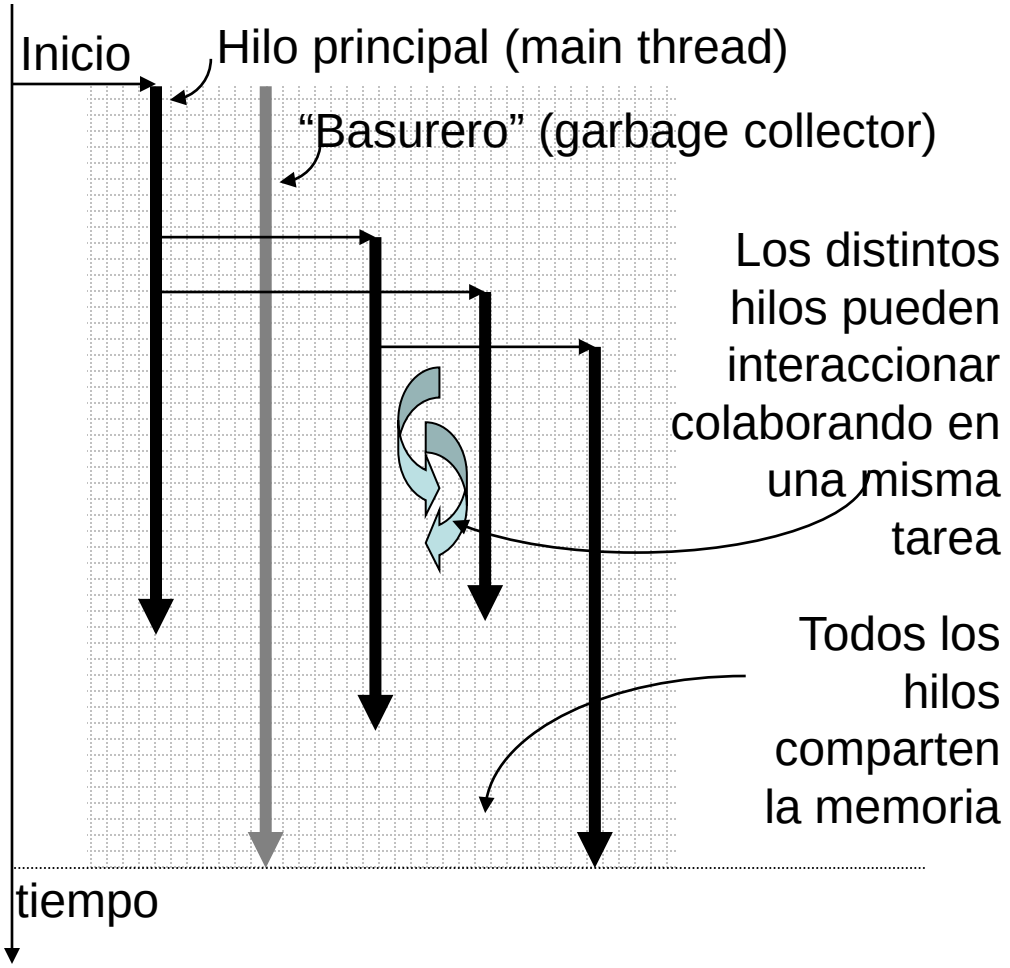
Hilos (Threads)

german.bordel@ehu.es
http://gtts.ehu.es/German

Procesos en un S.O.



Threads (Hilos) en un proceso

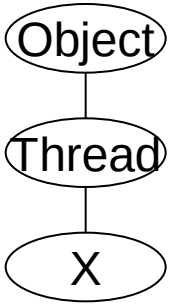


Tema:

Demonios

{setDaemon(boolean); isDaemon}

german.bordel@ehu.es
http://gtts.ehu.es/German

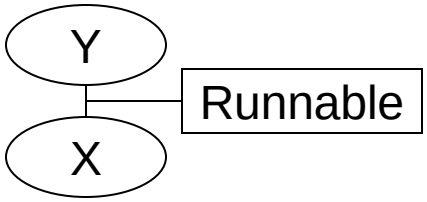


Objeto de subclase de Thread

```
class X  extends Thread {
.....

public void run()
 { // código origen del hilo
  }
}
```

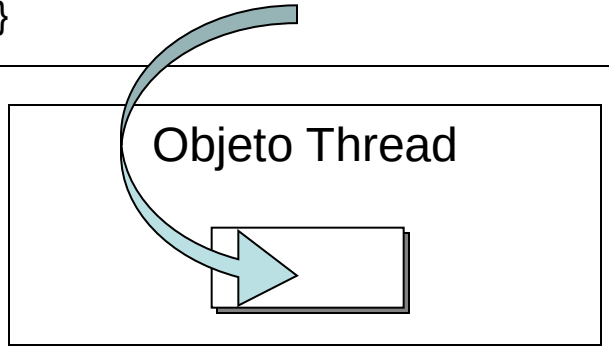
```
X a = new X(); a.start();
```

*El start() puede situarse en el constructor



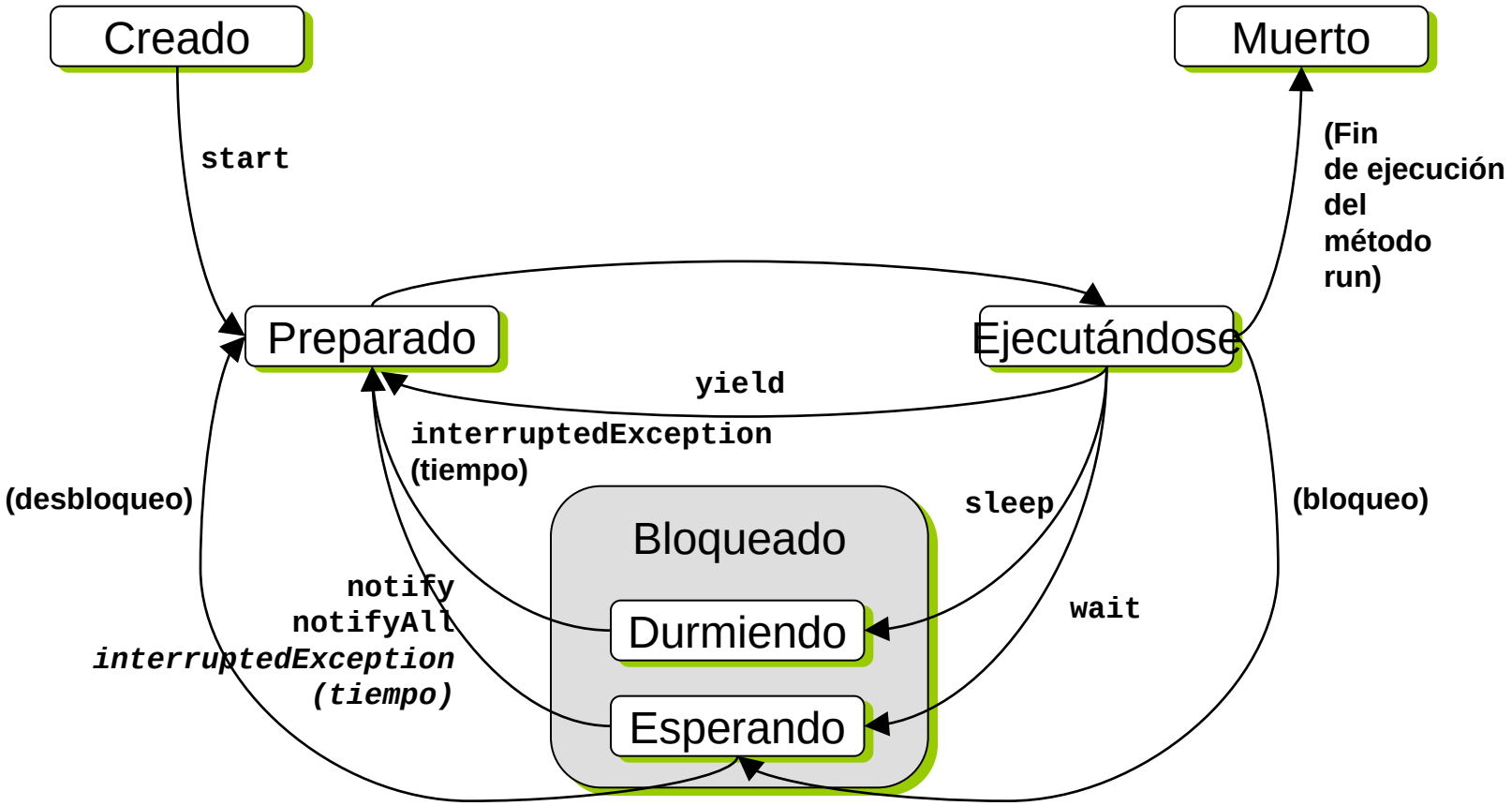
Objeto de clase Runnable

```
class X  extends Y implements Runnable {
.....

public void run()
 { // código origen del hilo
  }
}
```



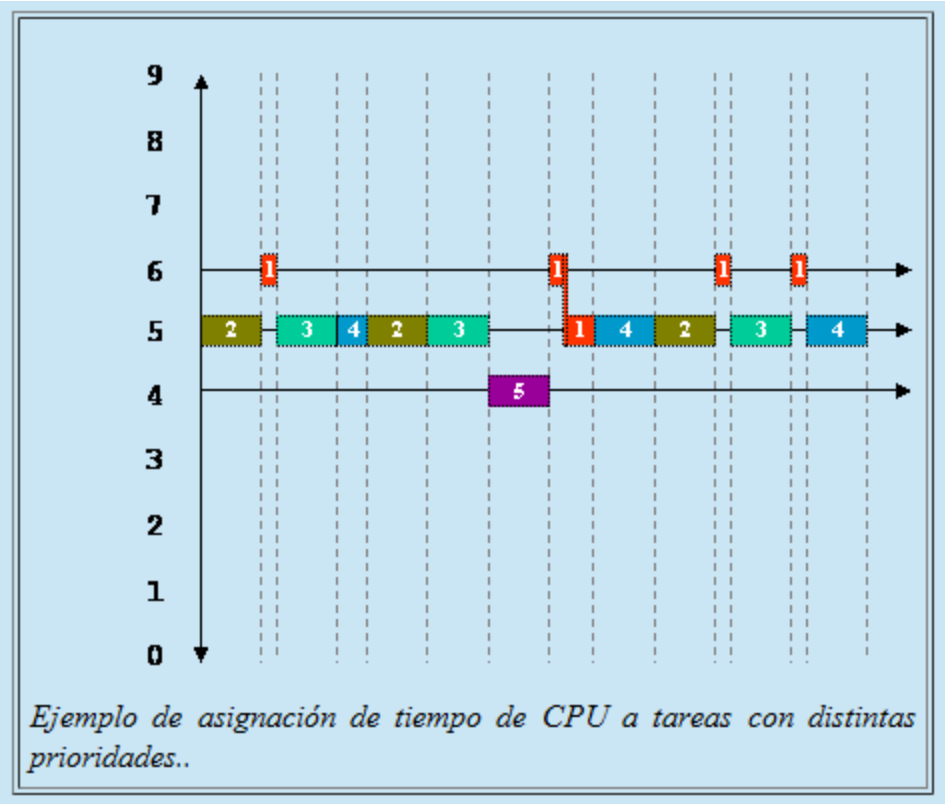

```
X a = new X(); Thread t=new Thread(a); t.start();
```

Hilos – Ciclo de vida



Tema:

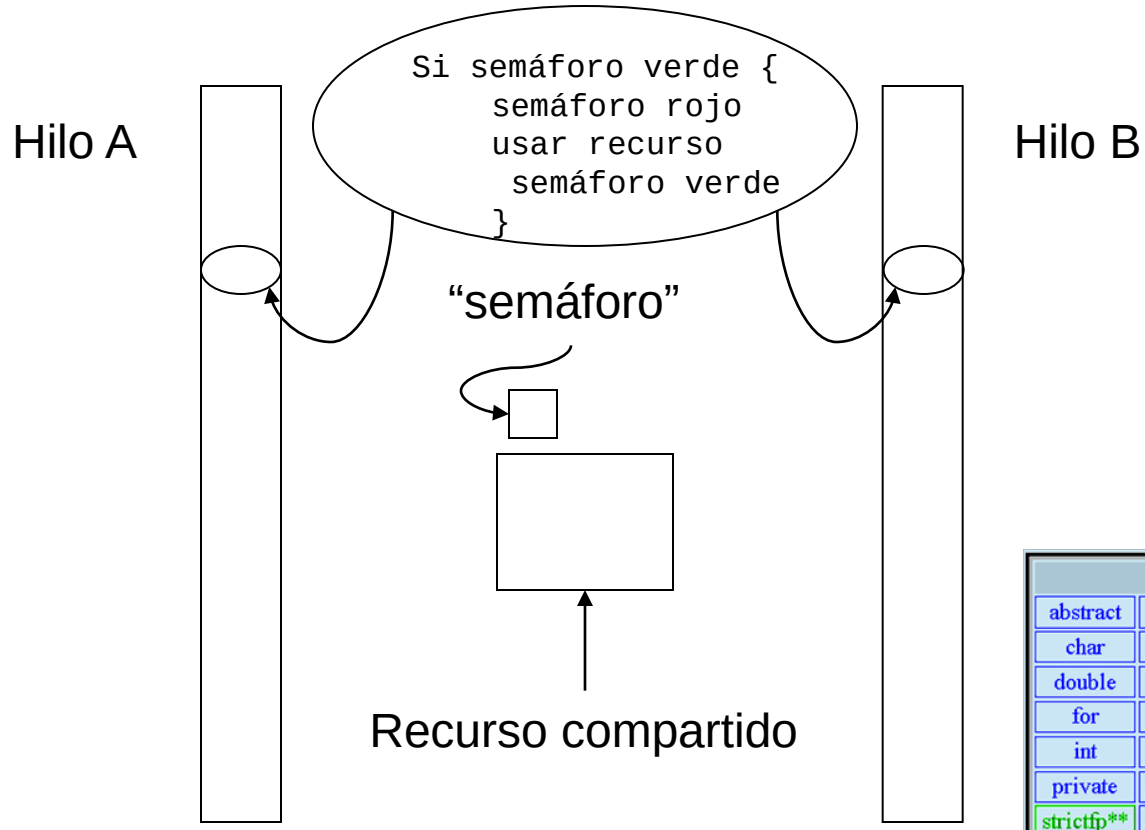
prioridades y “scheduling” {setPriority(int); getPriority()}



Ejemplo de asignación de tiempo de CPU a tareas con distintas prioridades..

Mecanismos proporcionados por Java para el entorno multi-hilo

- Exclusión mutua (secciones criticas)
- Bloqueo de recursos



Palabras reservadas en Java					
abstract	boolean	break	byte	case	catch
char	class	const*	continue	default	do
double	else	extends	final	finally	float
for	goto*	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp**	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while

Temas:

secciones críticas {synchronized}
sincronización

german.bordel@ehu.es
http://gtts.ehu.es/German

```
public class Cubiculo {
            private int contenido;
            private boolean disponible = false;

            public synchronized int get() {
                ...
            }

            public synchronized void put(int valor) {
                ...
            }
        }
```

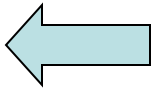
Sección crítica

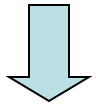
wait / notify

```
public synchronized int get() {
  while (!disponible) {
   // esperar a que el productor genere un valor
    try { wait();} catch (InterruptedException e) {}
  }
  disponible = false;
  // notificar al productor que el valor ha sido recogido
  notifyAll();
  return contenido;
  }

public synchronized void put(int valor) {
  while (disponible) {
    // esperar a que el consumidor recoja un valor
    try { wait(); } catch (InterruptedException e) {}
  }
  contenido = valor;
  disponible = true;
  // notificar al consumidor que el valor ha sido generado
  notifyAll();
}
```

Productor / consumidor

```
public class Productor extends Thread {
            private Cubiculo cubiculo;
            private int numero;

            public Productor(Cubiculo c, int numero) {
                cubiculo = c; this.numero = numero;
            }

            public void run() {
                for (int i = 0; i < 10; i++) {
                    cubiculo.put(i);
                    System.out.println("("+numero+ ") >> " + i);
                    try {
                        sleep((int)(Math.random() * 100));
                    } catch (InterruptedException e) { }
                }
            }
        }
```





```
public class Consumidor extends Thread {
    private Cubiculo cubiculo;
    private int numero;

    public Consumidor(Cubiculo c, int numero) {
        cubiculo = c; this.numero = numero;
        setDaemon(true);
    }

    public void run() {
        int valor = 0;
        while (true) {
            valor = cubiculo.get();
            System.out.println("  (" + numero+ ") << " + valor);
            yield();
        }
    }
}
```

Comprobando el funcionamiento

```
public class MainProdCons extends Object {

  public static void main (String args[]) {
    Cubiculo cubiculo=new Cubiculo();
    Productor p1=new Productor(cubiculo,1);
    Productor p2=new Productor(cubiculo,2);
    Productor p3=new Productor(cubiculo,3);
    Consumidor c1=new Consumidor(cubiculo,1);
    Consumidor c2=new Consumidor(cubiculo,2);
    Consumidor c3=new Consumidor(cubiculo,3);

    p1.start();
    p2.start();
    p3.start();
    c1.start();
    c2.start();
    c3.start();

    }

}
```

```
(1) >> 0        (3) >> 5
  (1) << 0        (1) << 5
(2) >> 0        (2) >> 5
  (2) << 0        (2) << 5
(3) >> 0        (3) >> 6
  (3) << 0        (3) << 6
(2) >> 1        (1) >> 5
  (1) << 1        (1) << 5
(3) >> 1        (2) >> 6
  (2) << 1        (2) << 6
(1) >> 1        (3) >> 7
  (3) << 1        (3) << 7
(1) >> 2        (2) >> 7
  (1) << 2        (1) << 7
(3) >> 2        (1) >> 6
  (2) << 2        (2) << 6
(2) >> 2        (3) >> 8
  (3) << 2        (3) << 8
(3) >> 3        (2) >> 8
  (1) << 3      (1) >> 7
(1) >> 3          (1) << 8
  (2) << 3        (2) << 7
(3) >> 4        (3) >> 9
  (3) << 4        (3) << 9
(2) >> 3        (2) >> 9
  (1) << 3        (1) << 9
(1) >> 4        (1) >> 8
  (2) << 4        (2) << 8
(2) >> 4        (1) >> 9
  (3) << 4        (3) << 9
```

Ojo!. Algo va mal

Arreglado... (no todo)

```
public void run() {
  for (int i = 0; i < 10; i++) {
    synchronized(cubiculo){
      cubiculo.put(i);
      System.out.println("("+numero+ ") >> " + i);
    }
    try {
      sleep((int)(Math.random() * 100));
    } catch (InterruptedException e) { }
  }
}
```

Productor / consumidor

```
public void run() {
  int valor = 0;
  while (true) {
    synchronized(cubiculo) {
      valor = cubiculo.get();
      System.out.println("  (" + numero+ ") << " + valor);
    }
    yield();
  }
}
```

```
(1) >> 0        (3) >> 4
  (1) << 0        (2) << 4
(1) >> 1        (2) >> 5
  (2) << 1        (3) << 5
(2) >> 0        (1) >> 6
  (3) << 0        (1) << 6
(3) >> 0        (3) >> 5
  (1) << 0        (2) << 5
(1) >> 2        (2) >> 6
  (2) << 2        (3) << 6
(2) >> 1        (1) >> 7
  (3) << 1        (1) << 7
(3) >> 1        (3) >> 6
  (1) << 1        (2) << 6
(1) >> 3        (2) >> 7
  (2) << 3        (3) << 7
(2) >> 2        (1) >> 8
  (3) << 2        (1) << 8
(3) >> 2        (3) >> 7
  (1) << 2        (2) << 7
(2) >> 3        (2) >> 8
  (2) << 3        (1) << 8
(1) >> 4        (1) >> 9
  (3) << 4        (3) << 9
(3) >> 3        (3) >> 8
  (1) << 3        (2) << 8
(2) >> 4        (2) >> 9
  (2) << 4        (1) << 9
(1) >> 5        (3) >> 9
  (1) << 5
```

Ojo!. Algo va mal

german.bordel@ehu.es
http://gtts.ehu.es/German

Inanición (starvation)
Ocurre cuando uno o más hilos de un programa ven siempre bloqueado su acceso a un recurso y por tanto no pueden progresar

Interbloqueo (deadlock)
Es una forma “terminal” de inanición. Ocurre cuando dos o más hilos esperan a una condición que no puede satisfacerse. El interbloqueo más habitual consiste en que dos (o más) hilos esperan a que otro haga algo de un modo circular.

Atributo “`volatil`”