Programación Concurrente en Java: Threads

Luis Fernando Llana Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

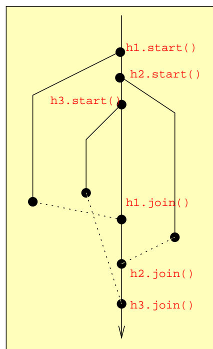
7 de mayo de 2007

Ejemplos de *programación concurrente*

- En un sistema operativo, diversos programas compiten por los recursos del sistema: memoria, dispositivos.
- Bases de datos.
- Aplicaciones Web.

Hebras, hilos

En un programa concurrente puede haber *varios hilos de computación*.



Sincronización de Objetos

- Puede haber varias hebras ejecutando *simultáneamente* métodos de objetos
- Es necesario *sincronizar* los accesos al objeto.

Threads

Extendiendo la clase java.lang.Thread.

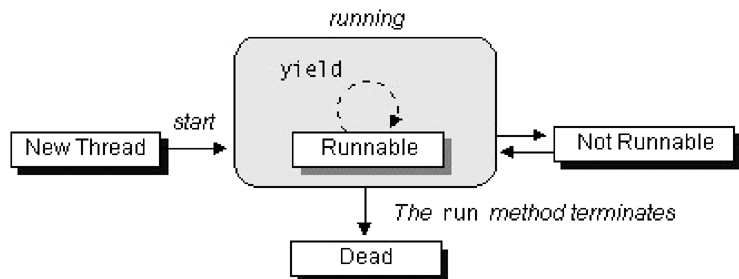
```
public class PrThread extends Thread{
    public PrThread(String s) {
        super(s);
    }
    public final void run() {
        boolean sigue=true;
        for (int i=0; i<100 && sigue; i++) {
            try {
                System.out.println(getName()+":"+i);
                sleep(20);
            } catch (InterruptedException e) {
                System.out.println(getName()+" interrumpida");
                sigue=false;
            }
        }
    }
    public static final void main(final String[] args){
        Thread p = new PrThread("mia");
        p.start();
    }
}
```

Threads

Implementado el interfaz java.lang.Runnable.

```
public class PrRunnable implements Runnable {                          1
    public final void run() {                                         2
        Thread hebra = Thread.currentThread();                        3
        boolean sigue=true;                                           4
        for (int i=0; i<100 && sigue; i++) {                          5
            try {                                                     6
                System.out.println(hebra.getName()+":"+i);            7
                hebra.sleep(20);                                      8
            } catch (InterruptedException e) {                        9
                System.out.println(hebra.getName()+" interrumpida");  10
                sigue=false;                                          11
            }                                                         12
        }                                                             13
    }                                                                 14
    public static final void main(final String[] args) {              15
        Thread p = new Thread(new PrRunnable(),"mia");                16
        p.start();                                                    17
    }                                                                 18
}                                                                     19
```

Ciclo de vida de una hebra



Tras crear una hebra y se ejecuta el método start, la hebra puede estár:

1. En ejecución.
2. Suspendida: ha ejecutado sleep, join, wait.

Parar una hebra

Usar el método `interrupt`

```
public static void ex1() throws InterruptedException{      1
    Thread h = new PrThread("1");                           2
    h.start();                                              3
    Thread.sleep(100);                                      4
    h.interrupt();                                          5
    System.out.println(h.isInterrupted());                  6
}                                                           7
```

Métodos *Deprecated*: `stop`, `suspend`, `resume`.
Una hebra para cunado está *Not Runnnable*, ha ejecutado `sleep`, `join`, `wait`.
Pueden lanzar `InterruptedException`, la hebra debería parar.

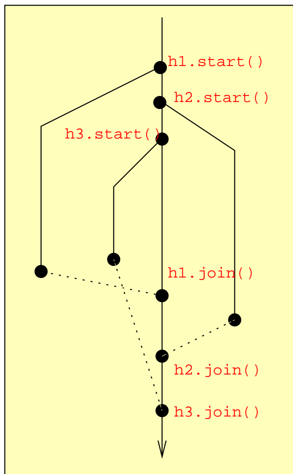
Esperamos a que una hebra acabe

Método: join

```
public static void ex2() throws InterruptedException {
    Thread h1 = new PrThread("1");
    Thread h2 = new PrThread("2");
    Thread h3 = new PrThread("3");
    h1.start();
    h2.start();
    h3.start();
    h1.join();
    h2.join();
    h3.join();
}
```

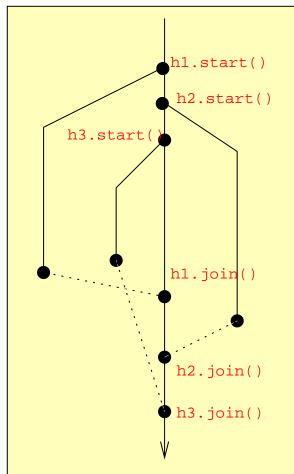
Esperamos a que una hebra acabe

Método: `join`



Esperamos a que una hebra acabe

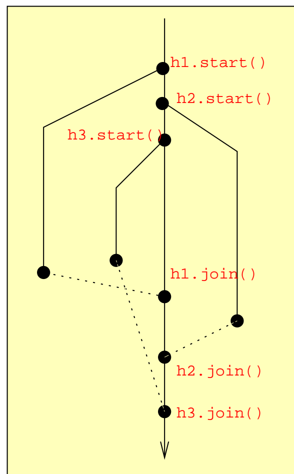
Método: `join`



1. `h1.start()`: `h1` se ejecuta.

Esperamos a que una hebra acabe

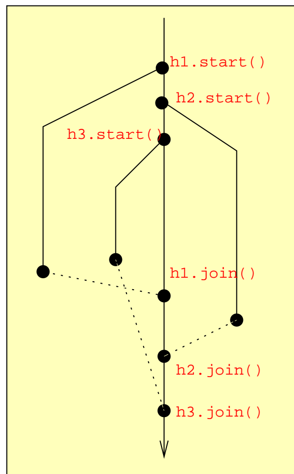
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.

Esperamos a que una hebra acabe

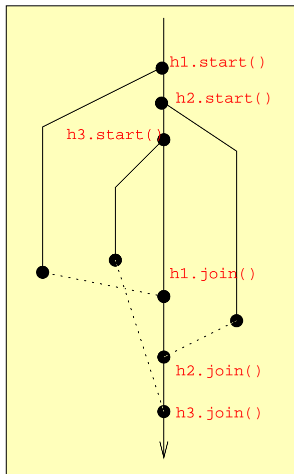
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.

Esperamos a que una hebra acabe

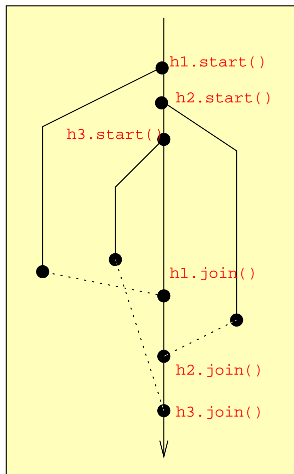
Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.
4. `h1.join()`: esperamos a que `h1` pare.

Esperamos a que una hebra acabe

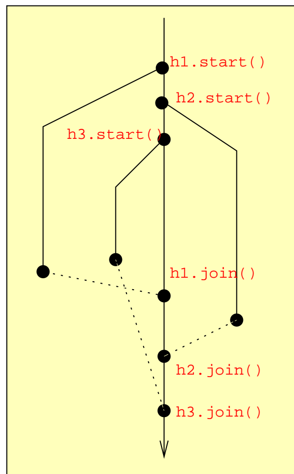
Método: join



1. h1.start(): h1 se ejecuta.
2. h2.start(): h2 se ejecuta.
3. h3.start(): h3 se ejecuta.
4. h1.join(): esperamos a que h1 pare.
5. h2.join(): esperamos a que h2 pare.

Esperamos a que una hebra acabe

Método: `join`



1. `h1.start()`: `h1` se ejecuta.
2. `h2.start()`: `h2` se ejecuta.
3. `h3.start()`: `h3` se ejecuta.
4. `h1.join()`: esperamos a que `h1` pare.
5. `h2.join()`: esperamos a que `h2` pare.
6. `h3.join()`: esperamos a que `h3` pare (no hace falta).

Cerrojos de objetos

- Cada objeto en Java tiene un cerrojo

```
synchronized (obj) {
   /*   */
    }
```

  - Sólo pueda haber una hebra *propietaria* del cerrojo.
  - Sólo una hebra propietaria del cerrojo puede ejecutar un código `synchronized`.

- El cerrojo puede abarcar a todo un método

```
type method (...) {
  synchronized(this) {
     /*    */
   }
```

```
synchronized type method (...) {
     /*   */
}
```

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

- La hebra queda *suspendida* hasta que alguien la *despierte*.
- Se libera para que otra hebra pueda adquirirlo.
- Cuando es liberarla debe adquirir de nuevo el cerrojo para seguir la ejecución.

`wait(tiempo)` Igual, pero se queda dormida un `tiempo` máximo.

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

- La hebra queda *suspendida* hasta que alguien la *despierte*.
- Se libera para que otra hebra pueda adquirirlo.
- Cuando es liberarla debe adquirir de nuevo el cerrojo para seguir la ejecución.

`wait(tiempo)` Igual, pero se queda dormida un `tiempo` máximo.

`notify()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `notify()` del objeto.

Durmiéndose/depertándose en un objeto

`wait()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `wait()` del objeto.

- La hebra queda *suspendida* hasta que alguien la *despierte*.
- Se libera para que otra hebra pueda adquirirlo.
- Cuando es liberarla debe adquirir de nuevo el cerrojo para seguir la ejecución.

`wait(tiempo)` Igual, pero se queda dormida un `tiempo` máximo.

`notify()` Una hebra que tiene el cerrojo de un objeto puede invocar el método `notify()` del objeto.

- Despierta *una* hebra suspendida en el objeto.
- *No* libera el cerrojo del objeto.

`notifyAll` Igual, pero despierta a todas.

Regiones críticas

El acceso a las regiones críticas debe ser *exclusivo*

```
    public void run() {                                              1
        boolean para = false;                                        2
        while (!para) {                                              3
            try {                                                    4
                ciclo();                                             5
            } catch (InterruptedException e) {                       6
                para = true;                                         7
            }                                                        8
        }                                                            9
    }                                                                10
    private void ciclo() throws InterruptedException {               11
        monitor.entrar();                                            12
        InterruptedException salir=null;                            13
        try {                                                        14
            regCritica();                                            15
        } catch (InterruptedException e) {                           16
            salir=e;                                                 17
        } finally {                                                  18
            monitor.salir();                                         19
            if (salir!=null) {                                       20
                throw salir;                                         21
            }                                                        22
            int t = random.nextInt(tiempoDentro);                    23
            sleep(t);                                                24
        }                                                            25
    }                                                                26
```

Monitor regiones críticas

```java
public class Monitor {
    private int caben; //caben>=0
    public Monitor() {
        caben=1;
    }
    public synchronized void salir() {
        caben++;
        notify();
    }

    public synchronized void entrar() throws InterruptedException{
        while (caben==0) wait();
        caben--;
    }
}
```

Colecciones sincronizadas

En la clase `java.util.Collections` encontramos los siguientes métodos estáticos:

```
public static <T> Collection<T> synchronizedCollection(Collection<T> c)
```
1

```
public static <T> Set<T> synchronizedSet(Set<T> s)
```
1

```
public static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)
```
1

```
public static <T> List<T> synchronizedList(List<T> list)
```
1

```
public static <K,V> Map<K,V> synchronizedMap(Map<K,V> m)
```
1

```
public static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)
```
1

Lectores/Escritores I

```
public void run()  {                    1
    try {                               2
        monitor.permisoLeer();          3
        lee();                          4
    } finally {                         5
        monitor.finLeer();              6
    }                                   7
}                                       8
```

```
public void run()  {                    1
    try {                               2
        monitor.permisoEscribir();      3
        escribe();                      4
    } finally {                         5
        monitor.finEscribir();          6
    }                                   7
}                                       8
```

Lectores/Escritores II

```
package simulacion.lectoresEscritores;                                          1
public class Monitor {                                                          2
    private int escrEsperando;  //núm escritores esperando                      3
    private int numLect; // núm lectores leyendo                                4
    private int numEscr; // núm escritores escribiendo                          5
// escrEsperando>=numEscr, 0<=numEscr<=1 numLect>=0, nunLect>0 ---> numEscr=0   6
    public Monitor() {                                                          7
        escrEsperando=0; numLect=0; numEscr=0;                                  8
    }                                                                           9
    public synchronized void permisoLeer() throws InterruptedException {        10
        while (numEscr>0 || escrEsperando>0) wait();                            11
        numLect++;                                                              12
    }                                                                           13
    public synchronized void finLeer() {                                        14
        numLect--;                                                              15
        notifyAll();                                                            16
    }                                                                           17
    public synchronized void permisoEscribir() throws InterruptedException {    18
        escrEsperando++;                                                        19
        while (numLect>0) wait();                                               20
        numEscr++;                                                              21
    }                                                                           22
    public synchronized void finEscribir() {                                    23
        escrEsperando--;                                                        24
        numEscr--;                                                              25
    }                                                                           26
}                                                                               27
```

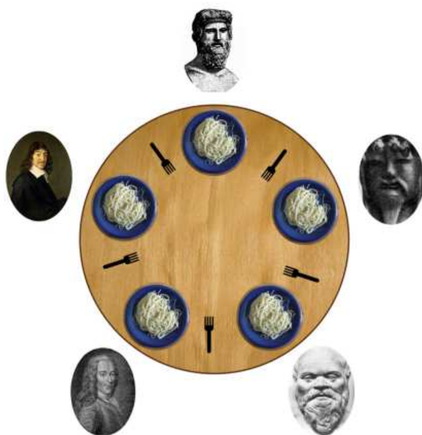
Lectores/Escritores III

```java
package simulacion.lectoresEscritores;
import soporte.Aleatorio;
public class GeneradorLectores extends Thread {
    private double tMedioLlegada;
    private int tLeyendo;
    private int tMaxSimulacion;
    private Aleatorio aleatorio;
    private Monitor monitor;
    public GeneradorLectores (double tllegada, int tleyendo,  int tmax, int semilla, Mon
        tMedioLlegada = tllegada; tLeyendo = tleyendo;  tMaxSimulacion = tmax;
        aleatorio = new Aleatorio(semilla);
        monitor = m;
    }
    public void run() {
        System.out.println("Empezando la generación de lectores");
        try {
            while (true) {
                int sigCoche = aleatorio.poisson(tMedioLlegada);
                this.sleep(sigCoche*1000);
                Lector lector = new Lector(Simulador.sigUsuario(), tLeyendo, monitor);
                System.out.println("Comenzando "+lector);
                lector.start();
            }
        } catch ( InterruptedException e ) {}
        System.out.println("Simulación lectores finalizada");
    }
}
```

Lectores/Escritores IV

```
package simulacion.lectoresEscritores;                                    1
class Simulador {                                                         2
    private static int  numUsuario = -1;                                  3
    private static long horaInicio;                                       4
    public synchronized static int sigUsuario() {                         5
        numUsuario++;                                                     6
        return numUsuario;                                                7
    }                                                                     8
    public Simulador(double tLlegadaLect, double tLlegadaEscr,            9
                     int tLeyendo, int tEscribiendo, int tMax)            10
        throws InterruptedException {                                     11
        Monitor monitor = new Monitor();                                  12
        GeneradorLectores generaLectores =                                13
            new GeneradorLectores(tLlegadaLect, tLeyendo, tMax, 1111, monitor);   14
        GeneradorEscritores generaEscritores =                            15
            new GeneradorEscritores(tLlegadaEscr, tEscribiendo, tMax, 3333, monitor);   16
        generaLectores.start();                                           17
        generaEscritores.start();                                         18
        horaInicio = System.currentTimeMillis();                          19
        Thread.sleep(tMax*1000);                                          20
        generaLectores.interrupt();                                       21
        generaEscritores.interrupt();                                     22
        generaLectores.join();                                            23
        generaEscritores.join();                                          24
    }                                                                     25
    public static void main (String[] args) throws Exception {            26
        Simulador s = new Simulador(2, 8, 1, 2, 30);                      27
    }                                                                     28
```

Filósofos I



Filósofos II

```
while (true) {
  piensa();
  mesa.pideTenedores();
  come();
  mesa.cedeTenedores();
}
```

$$comiendo[i] \begin{cases} true & \text{filósofo } i \text{ está comiendo} \\ false & \text{filósofo } i \text{ NO está comiendo} \end{cases}$$

$$INV \equiv comiendo[i] \rightarrow \neg comiendo[i \oplus 1] \wedge \neg comiendo[i \ominus 1]$$

Filósofos III

```
public class Mesa {                                                           1
    private int numFilosofos;                                                 2
    private boolean [] comiendo;                                              3
    public Mesa(int n) {                                                      4
        numFilosofos = n;;                                                    5
        comiendo = new boolean[numFilosofos];                                 6
        for (int i = 0; i < numFilosofos; i++) { comiendo[i] = false; }       7
    }                                                                         8
    /*  permisoComer(int i) y cedePermiso(int i) */                           9
}                                                                             10
```

Filósofos IV

```
public synchronized void permisoComer(int i) throws InterruptedException{
  while (comiendo[ant(i)] || comiendo[sig(i)]) { wait(); }
  comiendo[i]=true;
}

public synchronized void cedePermiso(int i) {
  comiendo[i]=false;
  notifyAll();
}

public int sig(int i) { return (i+1) % numFilosofos; }

public int ant(int i) { return  ( i-1+numFilosofos ) % numFilosofos; }
```

Filósofos V

```java
public class Filosofo extends Thread {
    private Random random;
    private int tiempoComiendo, tiempoPensando;
    private Mesa mesa;
    private int id;
    public Filosofo (Random r,
                     int tc, int tp,
                     Mesa m,
                     int i) {
        random = r;
        tiempoPensando = tp;
        tiempoComiendo = tc;
        mesa = m;
        id = i;
    }
    public void run() {
        boolean para = false;
        while (!para) {
            try {
                ciclo();
            } catch (InterruptedException e) {
                para = true;
            }
        }
    }
```

Filósofos VI

```
    private void ciclo() throws InterruptedException {
        piensa(id,tiempoPensando);
        mesa.permisoComer(id);
        try {
            come(id,tiempoComiendo);
        } catch (InterruptedException e) {
            mesa.cedePermiso(id);
            throw e;
        }
        mesa.cedePermiso(id);
    }
    private void espera(int tiempo) throws InterruptedException {
        int t = random.nextInt(tiempo);
        sleep(t);
    }
    private  void piensa(int id, int tiempo) throws InterruptedException {
        System.out.println("El filósofo "+id+" empieza a pensar");
        espera(tiempo);
        System.out.println("El filósofo "+id+" acaba de pensar");
    }
    private void come(int id, int tiempo)  throws InterruptedException {
        System.out.println("El filósofo "+id+" empieza a comer"+":"+mesa);
        espera(tiempo);
        System.out.println("El filósofo "+id+" acaba de comer"+":"+mesa);
    }
}
```

Filósofos VII

```
public static void main (String[] args) throws InterruptedException {
    int  numFilosofos = 5;
    int tiempoPensando = 1000;
    int tiempoComiendo  = 2000;
    int tiempoParada = 10000;
    Random r = new Random();
    Mesa mesa = new Mesa(numFilosofos);
    Filosofo [] filosofo = new Filosofo[numFilosofos];
    for (int i = 0; i < numFilosofos ; i++) {
        filosofo[i] = new Filosofo(r,tiempoComiendo, tiempoPensando,
                                   mesa, i);
        filosofo[i].start();
    }

    for (int i = 0; i < numFilosofos ; i++) {
        Thread.sleep(tiempoParada);
        System.out.println("Parando filósofo "+i+"............................");
        filosofo[i].interrupt();
    }
    for (int i = 0; i < numFilosofos ; i++) {
        filosofo[i].join();
    }
}
```