

Programación Concurrente en Java: Threads

Luis Fernando Llana Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

24 de abril de 2008

Luis Fernando Llana Díaz Departamento de Sistemas Informáticos y ComputaciónUniversidad Complutense de Madrid
Programación Concurrente en Java: Threads

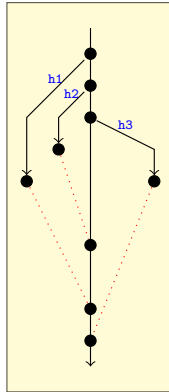
Ejemplos de *programación concurrente*

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

Luis Fernando Llana Díaz Departamento de Sistemas Informáticos y ComputaciónUniversidad Complutense de Madrid
Programación Concurrente en Java: Threads

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();
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

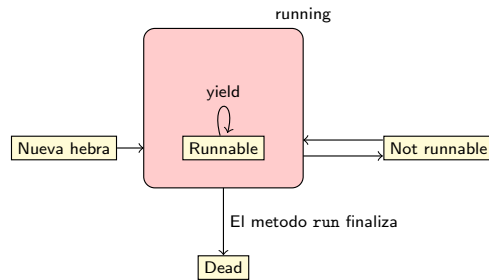
Threads

Implementado el interfaz `java.lang.Runnable`.

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Ciclo de vida de una hebra



- Se crea la hebra.
- La hebra está en ejecución. Se ejecuta el método **start**.
- La hebra se suspende: ha ejecutado **sleep**, **join**, **wait**.
- La hebra finaliza. El método **run** finaliza.

Parar una hebra

Usar el método **interrupt**

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

1
2
3
4
5
6
7

Métodos **Deprecated**: **stop**, **suspend**, **resume**.

Una hebra para cunado está **Not Runnable**, ha ejecutado **sleep**, **join**, **wait**.

Pueden lanzar **InterruptedException**, la hebra debería parar.

Esperamos a que una hebra acabe

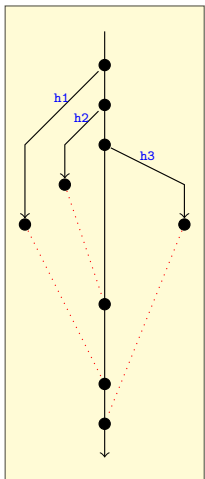
Método: `join`

```
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();  
}
```

1
2
3
4
5
6
7
8
9
10
11

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).

Cerros de objetos

- Cada objeto en Java tiene un cerrojo

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

1
2
3

- 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) {  
        /* */  
    }  
}
```

1
2
3
4

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

1
2
3

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() {
    boolean para = false;
    while (!para) {
        try {
            ciclo();
        } catch (InterruptedException e) {
            para = true;
        }
    }
}

private void ciclo() throws InterruptedException {
    monitor.entrar();
    InterruptedException salir=null;
    try {
        regCritica();
    } catch (InterruptedException e) {
        salir=e;
    } finally {
        monitor.salir();
        if (salir!=null) {
            throw salir;
        }
        int t = random.nextInt(tiempoDentro);
        sleep(t);
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

Monitor regiones críticas

```
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--;
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

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() {
    try {
        monitor.permisoLeer();
        lee();
    } finally {
        monitor.finLeer();
    }
}
```

 1
2
3
4
5
6
7
8

```
public void run() {
    try {
        monitor.permisoEscribir();
        escribe();
    } finally {
        monitor.finEscribir();
    }
}
```

 1
2
3
4
5
6
7
8

Lectores/Escritores II

```
package simulacion.lectoresEscritores;
public class Monitor {
    private int escrEsperando; //número escritores esperando
    private int numLect; // número lectores leyendo
    private int numEscr; // número escritores escribiendo
    // escrEsperando >= numEscr, 0 <= numEscr <= 1 numLect >= 0, numLect > 0 ---> numEscr = 0
    public Monitor() {
        escrEsperando = 0; numLect = 0; numEscr = 0;
    }
    public synchronized void permisoLeer() throws InterruptedException {
        while (numEscr > 0 || escrEsperando > 0) wait();
        numLect++;
    }
    public synchronized void finLeer() {
        numLect--;
        notifyAll();
    }
    public synchronized void permisoEscribir() throws InterruptedException {
        escrEsperando++;
        while (numLect > 0) wait();
        numEscr++;
    }
    public synchronized void finEscribir() {
        escrEsperando--;
        numEscr--;
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Lectores/Escritores III

```
}
}
```

26
27

Lectores/Escritores IV

```
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, Monitor m){
        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");
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Lectores/Escritores V

```
}
}
```

26
27

Lectores/Escritores VI

```
package simulacion.lectoresEscritores;
class Simulador {
    private static int numUsuario = -1;
    private static long horaInicio;
    public synchronized static int sigUsuario() {
        numUsuario++;
        return numUsuario;
    }
    public Simulador(double tLlegadaLect, double tLlegadaEscr,
        int tLeyendo, int tEscribiendo, int tMax)
        throws InterruptedException {
        Monitor monitor = new Monitor();
        GeneradorLectores generaLectores =
            new GeneradorLectores(tLlegadaLect, tLeyendo, tMax, 1111, monitor);
        GeneradorEscritores generaEscritores =
            new GeneradorEscritores(tLlegadaEscr, tEscribiendo, tMax, 3333, monitor);
        generaLectores.start();
        generaEscritores.start();
        horaInicio = System.currentTimeMillis();
        Thread.sleep(tMax*1000);
        generaLectores.interrupt();
        generaEscritores.interrupt();
        generaLectores.join();
        generaEscritores.join();
    }
}
```

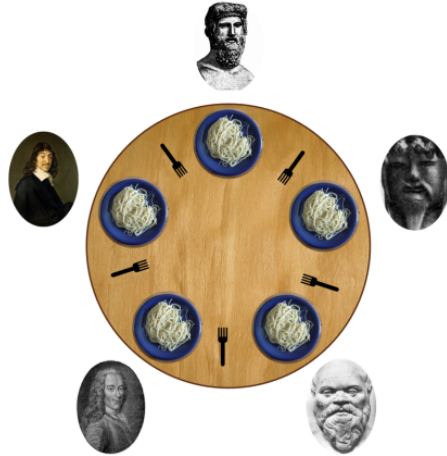
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Lectores/Escritores VII

```
public static void main (String[] args) throws Exception {
    Simulador s = new Simulador(2, 8, 1, 2, 30);
}
}
```

1
2
3
4

Filósofos I



Filósofos II

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

1
2
3
4
5
6

$$comiendo[i] \begin{cases} true & \text{filósofo } i \text{ está comiendo} \\ false & \text{filósofo } i \text{ NO está comiendo} \end{cases}$$
$$INV \equiv comiendo[i] \rightarrow \neg comiendo[i \oplus 1] \wedge \neg comiendo[i \ominus 1]$$

Filósofos III

```
public class Mesa {
    private int numFilosofos;
    private boolean [] comiendo;
    public Mesa(int n) {
        numFilosofos = n;
        comiendo = new boolean[numFilosofos];
        for (int i = 0; i < numFilosofos; i++) { comiendo[i] = false; }
    }
    /* permisoComer(int i) y cedePermiso(int i) */
}
```

1
2
3
4
5
6
7
8
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; }
```

1
2
3
4
5
6
7
8
9
10
11
12
13

Filósofos V

```
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;
            }
        }
    }
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

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);
}
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

Filósofos VII

```
}
```

26

Filósofos VIII

```
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();  
    }  
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23