


Java y JVM: programación concurrente

Adolfo López Díaz



JVM

→ Máquina virtual

◆ Ambiente de programación virtualizado

→ Permite la ejecución de programas Java ejecutables multiplataforma

Programación concurrente

→ Dos unidades básicas de ejecución:

- ◆ Procesos
- ◆ Hilos

→ Modelo principal de ejecución en Java: Hilos

Hilos

→ Definición

- ◆ Sección de código que se ejecuta independientemente.

→ Utilidad

- ◆ Permite aprovechar mejor los recursos en los sistemas multiprocesador.

Creación de hilos en Java

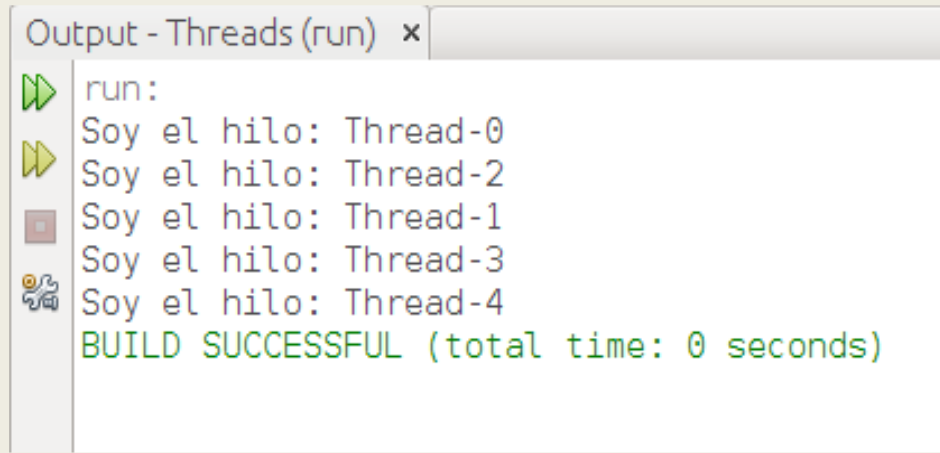
- Utilizando una clase que extienda de la clase Thread y sobrescribiendo su método run().
- Construyendo una clase que implemente la interfaz Runnable y luego creando un objeto de la clase Thread que recibe como parámetro el objeto Runnable.

Creación de hilos en Java

```
public class Prueba implements Runnable {  
  
    public void run(){  
        System.out.print("Soy el hilo "+ Thread.currentThread().getName());  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) throws InterruptedException{  
        Prueba prueba = new Prueba();  
        for(int i = 0; i<5; i++){  
            Thread thread = new Thread(prueba);  
            thread.start();  
        }  
    }  
}
```

Creación de hilos en Java

A screenshot of an IDE's output window titled "Output - Threads (run) x". The window contains the following text: "run:", "Soy el hilo: Thread-0", "Soy el hilo: Thread-2", "Soy el hilo: Thread-1", "Soy el hilo: Thread-3", "Soy el hilo: Thread-4", and "BUILD SUCCESSFUL (total time: 0 seconds)". On the left side of the window, there are several icons: a green play button, a yellow play button, a red stop button, and a magnifying glass icon.

```
Output - Threads (run) x
run:
Soy el hilo: Thread-0
Soy el hilo: Thread-2
Soy el hilo: Thread-1
Soy el hilo: Thread-3
Soy el hilo: Thread-4
BUILD SUCCESSFUL (total time: 0 seconds)
```

→ Necesidad de sincronización

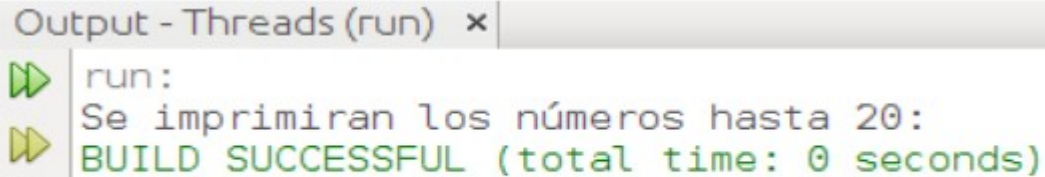
Join: Ejemplo

```
public class Prueba implements Runnable {
    @Override
    public void run(){
        System.out.println("Se imprimiran los números hasta 20:");
        for(int i=1; i<=20; i++){
            System.out.print(i+" ");
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) throws InterruptedException{
        Prueba prueba = new Prueba();
        Thread thread = new Thread(prueba);
        thread.start();

        System.exit(0);
    }
}
```


Join: Ejemplo



Output - Threads (run) x

```
run:  
Se imprimiran los números hasta 20:  
BUILD SUCCESSFUL (total time: 0 seconds)
```

The screenshot shows a terminal window titled "Output - Threads (run) x". It contains the following text: "run:", "Se imprimiran los números hasta 20:", and "BUILD SUCCESSFUL (total time: 0 seconds)". The text is displayed in a monospaced font with green highlights for the second and third lines. On the left side of the terminal, there are several icons: a green play button, a yellow play button, a red square, and a person icon.

- El hilo no logra iniciar su ejecución porque el hijo padre (main) termina su ejecución.
- Usar el método `join()`.

Join

```
public class Main {
    public static void main(String[] args) throws InterruptedException{
        Prueba prueba = new Prueba();
        Thread thread = new Thread(prueba);
        thread.start();
        /* el hilo principal (main) espera a que el hilo thread termine su ejecución */
        try{
            thread.join();
        }catch (InterruptedException e){}
        System.exit(0);
    }
}
```

Join

Output - Threads (run) x



run:



Se imprimiran los números hasta 20:

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



BUILD SUCCESSFUL (total time: 0 seconds)



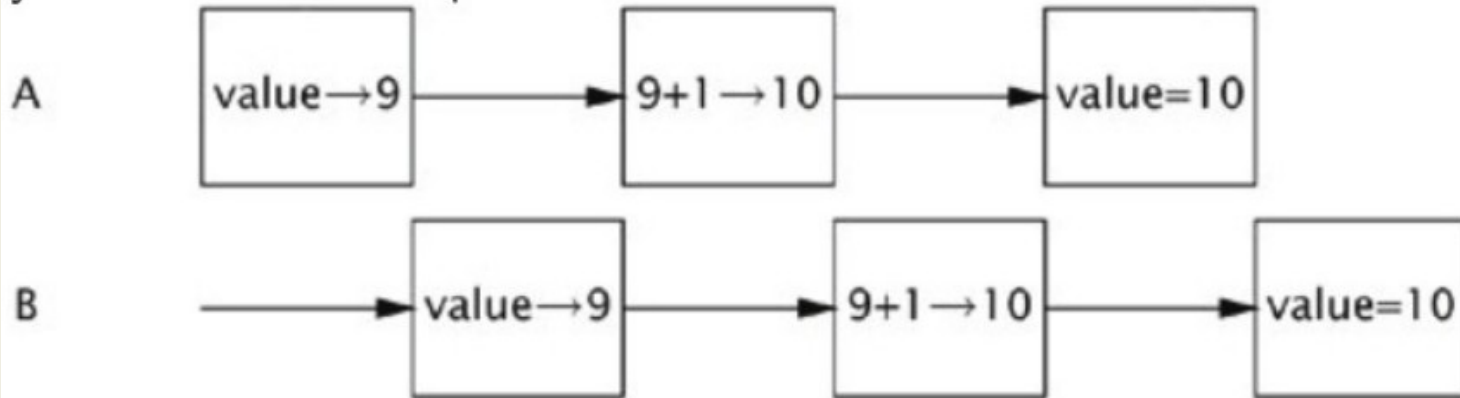
Atributos de los hilos

→ Prioridad

- ◆ Thread.MAX_PRIORITY
- ◆ Thread.MIN_PRIORITY
- ◆ Thread.NORMAL_PRIORITY.

→ Métodos setPriority(int newPriority) y getPriority()

Race condition



Cerros

- Atomicidad en la ejecución de instrucciones
- Bloques sincronizados
 - ◆ métodos
 - ◆ bloques de instrucciones

Cerros

→ Bloque de instrucciones sincronizado

```
public void addName(String name) {  
    synchronized(this) {  
        lastName = name;  
        nameCount++;  
    }  
    nameList.add(name);  
}
```

→ Método sincronizado

```
public synchronized void incrContador() {  
    contador = contador + 1;  
}
```

Cerros

- En java cada objeto tiene asociado un cerrojo
- El bloqueo se adquiere al entrar y se libera al salir
- Son utilizados por JVM, el programador no tiene acceso

Cerros: Ejemplo

```
public class Prueba implements Runnable {
    @Override
    public void run(){
        synchronized(this){
            System.out.println("\nSe imprimiran los números hasta 10:");
            for(int i=1; i<=10; i++){
                System.out.print(i+" ");
            }
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) throws InterruptedException{
        Prueba prueba = new Prueba();
        for(int i = 0; i<10; i++){
            Thread thread = new Thread(prueba);
            thread.start();
        }
    }
}
```

Cerrojos: Ejemplo

Se imprimiran los números hasta 10:

1 2 3 4 5 6 7 8 9

Se imprimiran los números hasta 10:

Se imprimiran los números hasta 10:

1 2 3 4 5

Se imprimiran los números hasta 10:

1 2 3 4 5 6 7 8

Se imprimiran los números hasta 10:

9

Se imprimiran los números hasta 10:

6 1 10 7 2 3 4 1 10 1 2 3

Se imprimiran los números hasta 10:

Se imprimiran los números hasta 10:

2 5 8 3 6 1 1 4 2 2 3 4 5 6 7 8 9 10 7 4 9 5 6 7 8

Se imprimiran los números hasta 10:

3 5 4 1 9 8 10 9 10 10 2

Se imprimiran los números hasta 10:

5 6 6 1 3 2 7 7 8 3 4 4 9 8 10 5 6 7 8 9 10 5 9 6 10 7 8 9 10 BUILD SUCCESSFUL (total time: 0 seconds)

Cerrojos: Ejemplo

```
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10 BUILD SUCCESSFUL (total time: 0 seconds)
```

Deadlocks

→ Java no proporciona control sobre situaciones peligrosas de la programación concurrente, por ejemplo en los interbloqueos o deadlocks.

Deadlocks: Ejemplo

```
public class Demo6 implements Runnable {  
private Object obj1;  
private Object obj2;  
public Demo6(Object o1, Object o2) {  
obj1 = o1;  
obj2 = o2;  
}  
}
```

```
public void run() {  
for (int i = 0; i < 1000; i++) {  
synchronized (obj1) {  
synchronized (obj2)  
System.out.println(i)  
}  
}  
}
```

```
public static void main(String[] args) {  
Object o1 = new Object();  
Object o2 = new Object();  
new Thread(new Demo6(o1, o2)).start();  
new Thread(new Demo6(o2, o1)).start();  
}  
}
```

Semáforos

- Clase Semaphore del paquete `java.util.concurrent`
- Semáforos contadores
- Permiten a un recurso ser compartido por varios procesos.
- Métodos `acquire()` y `release()`

Semáforos: Ejemplo

```
import java.util.concurrent.Semaphore;

public class Prueba implements Runnable {
    Semaphore semaphore = new Semaphore(1);
    @Override
    public void run(){
        semaphore.acquire();
        System.out.println("\nSe imprimiran los números hasta 10:");
        for(int i=1; i<=10; i++){
            System.out.print(i+" ");
        }
        semaphore.release();
    }
}
```

Semáforos: Ejemplo

```
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
Se imprimiran los números hasta 10:  
1 2 3 4 5 6 7 8 9 10  
BUILD SUCCESSFUL (total time: 0 seconds)
```


Referencias

- Fernández, J. (2012). *Java 7 Concurrency Cookbook*. Birmingham, Reino Unido: Packt Publishing.
- Göetz, B., Peierls, T., Bloch, J. (2006). *Java Concurrency In Practice*. EE.UU: Addison-Wesley.
- Gómez, P. (2000). *Concurrencia en Java*. Recuperado el 4 de noviembre de 2013, de rt00149b.eresmas.net/Otras/ConcurrenciaJAVA/ConcurrenciaEnJava.PDF.
- Horvilleur, G. (s.f.). *Multithreading y Java*. Recuperado el 4 de noviembre de 2013, de <http://www.comunidadjava.org/files/CuartaReunion/JavaYMultithreading.pdf>.
- Jenkov, J. (s.f.). *java.util.concurrent.Semaphore*. Recuperado el 4 de noviembre de 2013, de <http://tutorials.jenkov.com/java-util-concurrent/semaphore.html>.
- Morin, P. (s.f.). *The Java Virtual Machine (JVM)*. Recuperado el 4 de noviembre de 2013, del sitio Web de Carleton University: <http://cg.scs.carleton.ca/~morin/teaching/3002/notes/jvm.pdf>.

Referencias

- Oracle. (s.f.). *The Java Tutorials: Lesson Concurrency*. Recuperado el 4 de noviembre de 2013, de <http://docs.oracle.com/javase/tutorial/essential/concurrency/>
- Silberchatz, A., Baer, P y Gagne, G. (2013). *Operating Systems Concepts* (9a ed). EE.UU: John Wiley & Sons.
- UNAM, Posgrado en Ciencia e Ingeniería de la Computación. (s.f). *Concurrencia en Java*. Recuperado el 4 de noviembre de 2013, del sitio Web de la Universidad Nacional Autónoma de México: <http://www.matematicas.unam.mx/jloa/concurrencia.pdf>.