

Hilos

en Java

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

26 de noviembre de 2025



Temas

- 1 Concurrencia
 - Introducción
- 2 Hilos
 - Thread y Runnable
 - Interrupciones
- 3 Sincronización
 - Comunicación entre hilos
 - Vitalidad
- 4 Aplicación

Concurrencia

- 1 Concurrencia
- 2 Hilos
- 3 Sincronización
- 4 Aplicación

Temas

- 1 Concurrencia
 - Introducción

El rol del sistema operativo

- Las computadoras modernas realizan varias tareas del tal forma que el usuario las percibe como simultáneas.
- Esta simultaneidad puede ser real o simulada.
- El sistema operativo es responsable de:
 - La creación y borrado de procesos.
 - Asignar tiempo de ejecución a los procesos.
 - Proveer mecanismos para:
 - 1 Sincronizar,
 - 2 comunicar y
 - 3 resolver bloqueos mutuos:
 - Abrazos mortales entre procesos (*deadlock*)
 - Bloqueos en vida o bloqueos activos (*livelock*)

Java

- La Máquina Virtual de Java abstrae las peculiaridades de cada computadora y su sistema operativo, por lo que asume responsabilidades semejantes.
- La máquina virtual es ejecutada como un proceso por el sistema operativo.
- Cada máquina virtual en ejecución puede crear *hilos* (*threads*), que son procesos ligeros que emulan la realización de actividades de forma simultánea, dentro de un mismo programa.

Procesos e hilos

Procesos

- Programa en ejecución o tarea a cargo del SO.
- Recursos propios:
 - Memoria
 - Tiempo de UCP
 - Archivos
 - Dispositivos de E/S
- Sistema = Colección de procesos
- Hay dos tipos de procesos:
 - 1 Del sistema operativo
 - 2 Del usuario
- En Java pueden ser iniciados con un `ProcessBuilder`.

Hilos

- Un proceso está compuesto de hilos.
- Comparten recursos:
 - Memoria
 - Archivos
- Una aplicación tiene:
 - Un *hilo principal*.
 - Hilos del sistema:
 - Administran memoria (y recolección de basura).
 - Manejan señales.
 - Hilos iniciados por el programa.
- Se utilizan con la clase `Thread`

Ambiente de ejecución de Java

- La máquina virtual de Java es un proceso.
- Cada hilo tiene su propia pila de ejecución, pero todas comparten el montículo y tienen acceso al área de métodos.

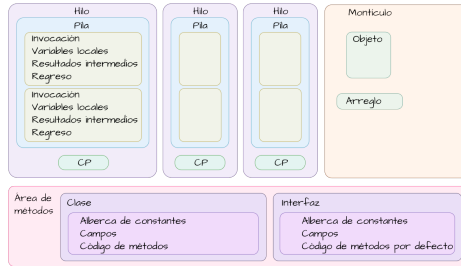


Figura: Adaptado de: <http://www.programcreek.com/2013/04/jvm-run-time-data-areas/>. La versión completa se encuentra en: Chapter2. The Structure of the Java Virtual Machine, en <http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html#jvms-2.5>

Hilos

- 1 Concurrencia
- 2 Hilos
- 3 Sincronización
- 4 Aplicación

Temas

2 Hilos

- Thread y Runnable
- Interrupciones

Thread

- La clase `Thread` se encarga del manejo y ejecución de hilos.
- La ejecución del hilo inicia con su método `start()`.
- Métodos interrumpibles (lanzan `InterruptedException`):
 - La ejecución de un hilo se puede pausar por un tiempo aproximado con `sleep(long)` o `sleep(long, int)`.
 - Un hilo puede esperar a que otro termine con el método del objetivo `join()`.
- Es posible crear nuevos hilos:
 - 1 Extendiendo esta clase y sobrescribiendo el método `void run()`, pero ya no será posible heredar de otra clase.
 - 2 Implementando la interfaz `Runnable` y pasando el objeto como argumento al constructor de `Thread`.

Ejemplo

Código: HolaThread.java

```
1 public class HolaThread extends Thread {
2     @Override
3     public void run() {
4         System.out.format("¡Hola desde el hilo %s!\n",
5                             Thread.currentThread().getName());
6     }
7     public static void main(String args[]) {
8         System.out.format("Iniciando en el hilo %s\n",
9                             Thread.currentThread().getName());
10        new HolaThread().start();
11    }
12 }
```

Este ejemplo produce la salida:

```
$ java HolaThread
Iniciando en el hilo main
¡Hola desde el hilo Thread-1!
```

Runnable

- Para usos más generales, se implementa la interfaz `Runnable`, con su único método `void run()`.
- El hilo nuevo sea crea utilizando el `Runnable` como argumento:

```
1 public class HolaRunnable implements Runnable {  
2     public void run() {  
3         System.out.format(";Hola desde el hilo %s!\n",  
4             Thread.currentThread().getName());  
5     }  
6     public static void main(String args[]) {  
7         System.out.format("Iniciando en el hilo %s\n",  
8             Thread.currentThread().getName());  
9         new Thread(new HolaRunnable()).start();  
10    }  
11 }
```

- La salida es idéntica al caso anterior.

Temas

2 Hilos

- Thread y Runnable
- Interrupciones

Interrupciones

- Las *interrupciones* son un método de comunicación entre hilos.
- Frecuentemente se les utiliza para avisar a un hilo que debe terminar su ejecución.
- Para interrumpir a un hilo se utiliza la *bandera de estado de interrupción*:
 - El hilo que desea interrumpir a otro hilo le avisa invocando el método `interrupt()` sobre el objeto que representa a su objetivo.
 - Un hilo que puede ser interrumpido debe preguntar periódicamente por el estado de `Thread.interrupted()`.
 - El hilo interrumpido puede optar por lanzar una `InterruptedException`.

```
1  import java.io.IOException;
2
3  public class Television extends Thread {
4      public static final int Q1 = 113;
5      private static final int[] canales = {2,4,5,7,9,11,13,28,34,40};
6      private int canal = 2;
7      private boolean encendida = true;
8
9      public void cambiaCanal() {
10         canal = canales[(int)(Math.random() * canales.length)];
11         System.out.format("Viendo_ahora_el_canal_%d%n", canal);
12     }
13
14     public void apaga() { encendida = false; }
15
16     @Override
17     public void run() {
18         cambiaCanal();
19         while(encendida) {
20             if (Thread.interrupted()) { cambiaCanal(); }
21             try {
22                 Thread.sleep(200);
23             } catch (InterruptedException ie) {
24                 System.out.println("No_dejan_dormir...");
25                 cambiaCanal();
26             }
27         }
28     }
29 }
```



```
1 public static void main(String[] args){
2     int input;
3     Television tele = new Television();
4     tele.start();
5     try {
6         while((input = System.in.read()) != 'q') {
7             tele.interrupt();
8         }
9         tele.apaga();
10    } catch(IOException ioe) { }
11 }
```

- Cuando se ejecuta resulta en:

Viendo ahora el canal 40

No dejan dormir...

Viendo ahora el canal 9

cambia

No dejan dormir...

Viendo ahora el canal 9

Viendo ahora el canal 28

q

Sincronización

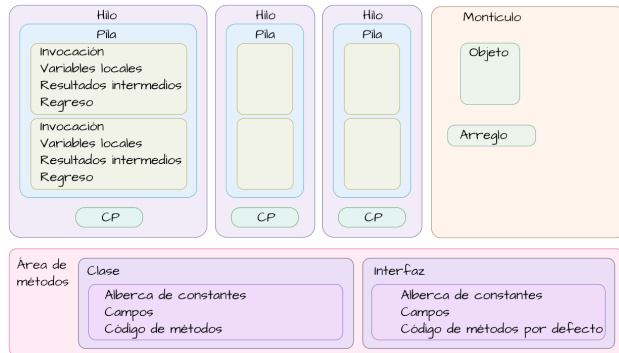
- 1 Concurrencia
- 2 Hilos
- 3 Sincronización**
- 4 Aplicación

Temas

- 3 Sincronización
 - Comunicación entre hilos
 - Vitalidad

Comunicación entre hilos

- Los hilos se comunican a través de:
 - los atributos de objetos en el montículo (fields),
 - los objetos a los cuales refieren dichos atributos,
 - campos en el área de métodos.



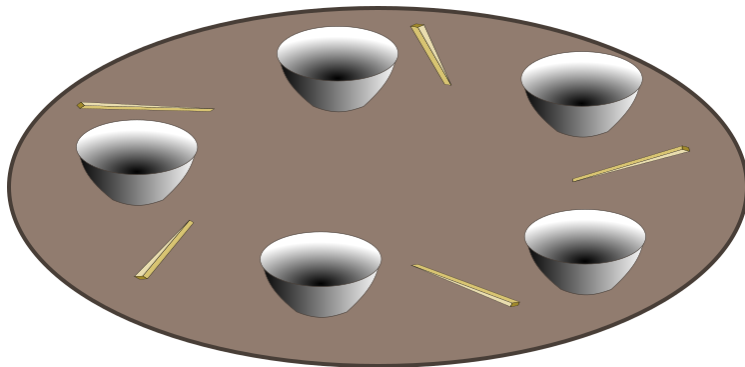
Errores

Imaginemos a un grupo de niños compitiendo por utilizar los mismos juguetes. Dicho escenario permitirá prever los problemas que pueden surgir al ejecutar hilos:

- Errores por acceso concurrente a la memoria:
 - Interferencia entre hilos
 - Errores en la consistencia de la memoria.
- Errores por coordinación de acceso entre los hilos:
 - Contención entre hilos (ejecución lenta o suspensión)
 - Abrazo mortal (*deadlock*).
 - Muerte de hambre (*starvation*).
 - Bloqueo en vida (*livelock*).



Problema de los filósofos



Candados

- Cuando algún recurso sólo debe ser utilizado por un hilo a la vez se le asocia un *candado*.
- El hilo que *adquiera* el candado es el único que puede acceder al recurso.
- Al finalizar su tarea debe *liberar* el candado.
- Todo objeto en Java tiene asociado un candado intrínseco.
- En Java, si un hilo posee un candado requerido para acceder otra región de código, puede seguir usando el candado que ya tiene.

Enunciados sincronizados

```
1 public
2     void addName (String name) {
3         synchronized(this) {
4             lastName = name;
5             nameCount++;
6         }
7         // llamada a otro método
8         nameList.add(name);
9     }
```

```
1 public class MsLunch {
2     private long c1 = 0;
3     private long c2 = 0;
4     private Object lock1 = new Object();
5     private Object lock2 = new Object();
6
7     public void inc1() {
8         synchronized(lock1) {
9             c1++;
10        }
11    }
12
13    public void inc2() {
14        synchronized(lock2) {
15            c2++;
16        }
17    }
18 }
```


Métodos sincronizados

- Garantizan que todas las instrucciones dentro del método serán ejecutadas, antes de que otros hilos puedan invocar otros métodos sincronizados en el mismo objeto.

Código: SynchronizedCounter.java

```
1 public class SynchronizedCounter {
2     private int c = 0;
3     // Compila a: leer, incrementar, guardar
4     public synchronized void increment() {
5         c++;
6     }
7     public synchronized void decrement() {
8         c--;
9     }
10    public synchronized int value() {
11        return c;
12    }
13 }
```

Acciones atómicas

- Se realizan en su totalidad, no pueden ser entrelazadas con la ejecución de otra acción.
- Leer y escribir de variables referencia.
- Leer y escribir en variables de tipo primitivo, excepto `long` y `double`.
- Leer y escribir para *todas* las variables declaradas como `volatile` (incluyendo `long` y `double`).
- Cuando una variable es declarada como `volatile`, cualquier acción de escritura deberá completarse antes de otras lecturas, lo cual reduce el riesgo de *inconsistencias en la memoria*.

Temas

3 Sincronización

- Comunicación entre hilos
- Vitalidad

Vitalidad (*liveness*)

Definición (Vitalidad)

Lo relativo al ciclo de vida de un hilo y su capacidad para ser ejecutado.

Problemas

- Los problemas característicos que afectan a la vitalidad de un hilo son:
 - Abrazo mortal** (*deadlock*). Dos hilos quedan bloqueados permanentemente esperándose el uno al otro.
 - Muerte de hambre** (*starvation*). Otros hilos frecuentemente usan recursos con candados, usándolos durante mucho tiempo, por lo que este hilo puede avanzar poco con sus labores.
 - Bloqueo en vida** (*livelock*). Varios hilos responden las acciones de otros interrumpiendo sus tareas, por lo que ninguno progresa. Ejemplo: la danza de dos personas en el corredor cediéndose el paso.

Bloques vigilados

- Cuando es necesario esperar a que se cumpla una condición para que el hilo realice su trabajo, se puede utilizar `wait()`.

Ejemplo: Productor consumidor

Tomado del tutorial de Java sobre concurrencia (Oracle s.f.).

Código: ProducerConsumerExample.java

```
1 public class ProducerConsumerExample {
2
3     public static void main(String[] args) {
4         // Memoria compartida
5         Drop drop = new Drop();
6         // Escribe mensajes
7         new Thread(new Producer(drop)).start();
8         // Lee/borra mensajes
9         new Thread(new Consumer(drop)).start();
10    }
11
12 }
```

Código: Drop.java

```
1 public class Drop {                                // Depósito de mensajes
2     private String message;
3     private boolean empty = true;
4     public synchronized String take() {
5         while (empty) {                             // No hay mensaje qué entregar
6             try{ wait(); } catch (InterruptedException e) {} // Espera notificación
7         }
8         empty = true;
9         notifyAll();                                // Avisar que queda vacío
10        return message;                             // Entrega el mensaje
11    }
12    public synchronized void put(String message) {
13        while (!empty) {                             // El depósito está lleno
14            try{ wait(); } catch (InterruptedException e) {} // Espera notificación
15        }
16        empty = false;
17        this.message = message;                      // Guarda un mensaje
18        notifyAll();                                 // Avisar que tiene un mensaje
19    }
20 }
```


Código: Producer.java

```
1 import java.util.Random;
2 public class Producer implements Runnable {
3     private Drop drop;                                // Referencia al depósito
4     public Producer(Drop drop) { this.drop = drop; }
5     public void run() {
6         String importantInfo[] = {
7             "Mares_eat_oats",           "Does_eat_oats",
8             "Little_lambs_eat_ivy",     "A_kid_will_eat_ivy_too"
9         };
10        Random random = new Random();
11        for (int i = 0; i < importantInfo.length; i++) {
12            drop.put(importantInfo[i]);    // Transmite mensajes uno por uno
13            try {
14                Thread.sleep(random.nextInt(5000));
15            } catch (InterruptedException e) {}
16        }
17        drop.put("DONE");
18    }
19 }
```

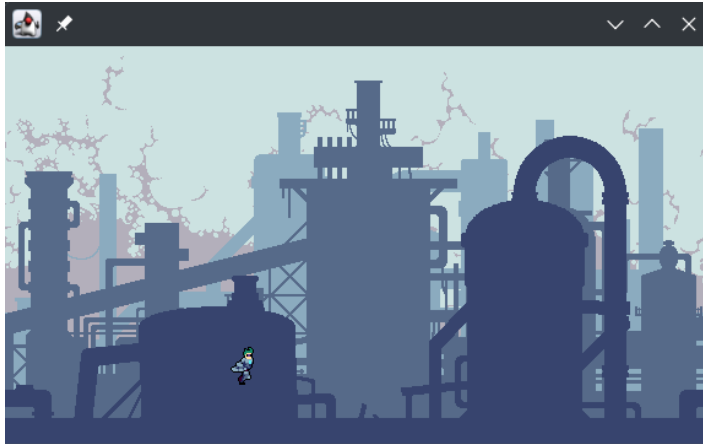
Código: Consumer.java

```
1  import java.util.Random;
2  public class Consumer implements Runnable {
3      private Drop drop;                                // Referencia al depósito
4      public Consumer(Drop drop) {
5          this.drop = drop;
6      }
7
8      public void run() {
9          Random random = new Random();
10         for (String message = drop.take();
11             !message.equals("DONE");
12             message = drop.take()) {
13             System.out.format("MESSAGE_RECEIVED: %s%n",
14                               message);
15             try {
16                 Thread.sleep(random.nextInt(5000));
17             } catch (InterruptedException e) {}
18         }
19     }
20 }
```

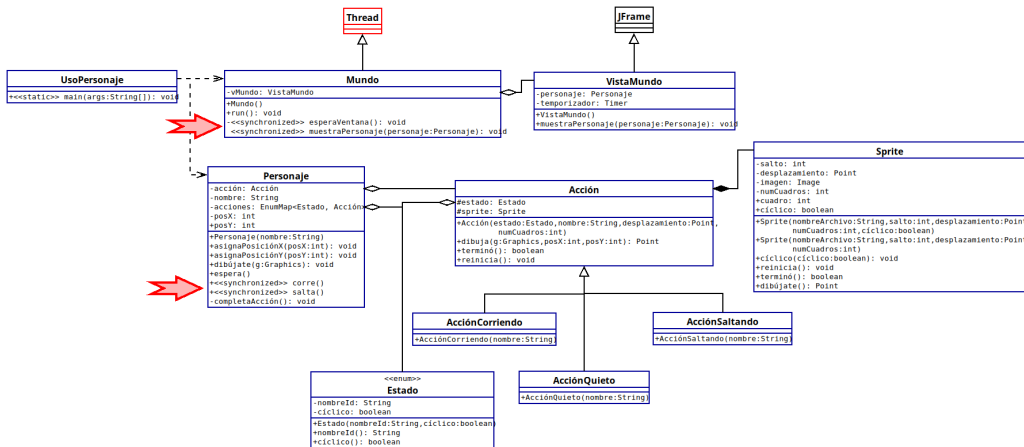
Aplicación

- 1 Concurrencia
- 2 Hilos
- 3 Sincronización
- 4 Aplicación**



Mundo



UML



Referencias

-  Abraham Silberschatz Peter Baer Galvin, Greg Gagne (s.f.). *Operating System Concepts*. 9th. Wiley. 920 págs.
-  Oracle (s.f.). *The Java Tutorials. Lesson: Concurrency*. URL: <https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual

