PROGRAMACIÓN DE SERVICIOS Y PROCESOS 2º DAM

Andrés Chillón Sesma

PROGRAMACIÓN DE PROCESOS Y SERVICIOS

UNIDAD 2: Programación multihilo

CONTENIDOS

Hilos:

- Estados de un hilo. Cambios de estado.
- Recursos compartidos por los hilos.
- Hilos de usuario vs. hilos de sistema. Modelos de hilos.
- Planificación de hilos.
- Hilo principal de un programa.

Elementos relacionados con la programación de hilos. Librerías y clases.

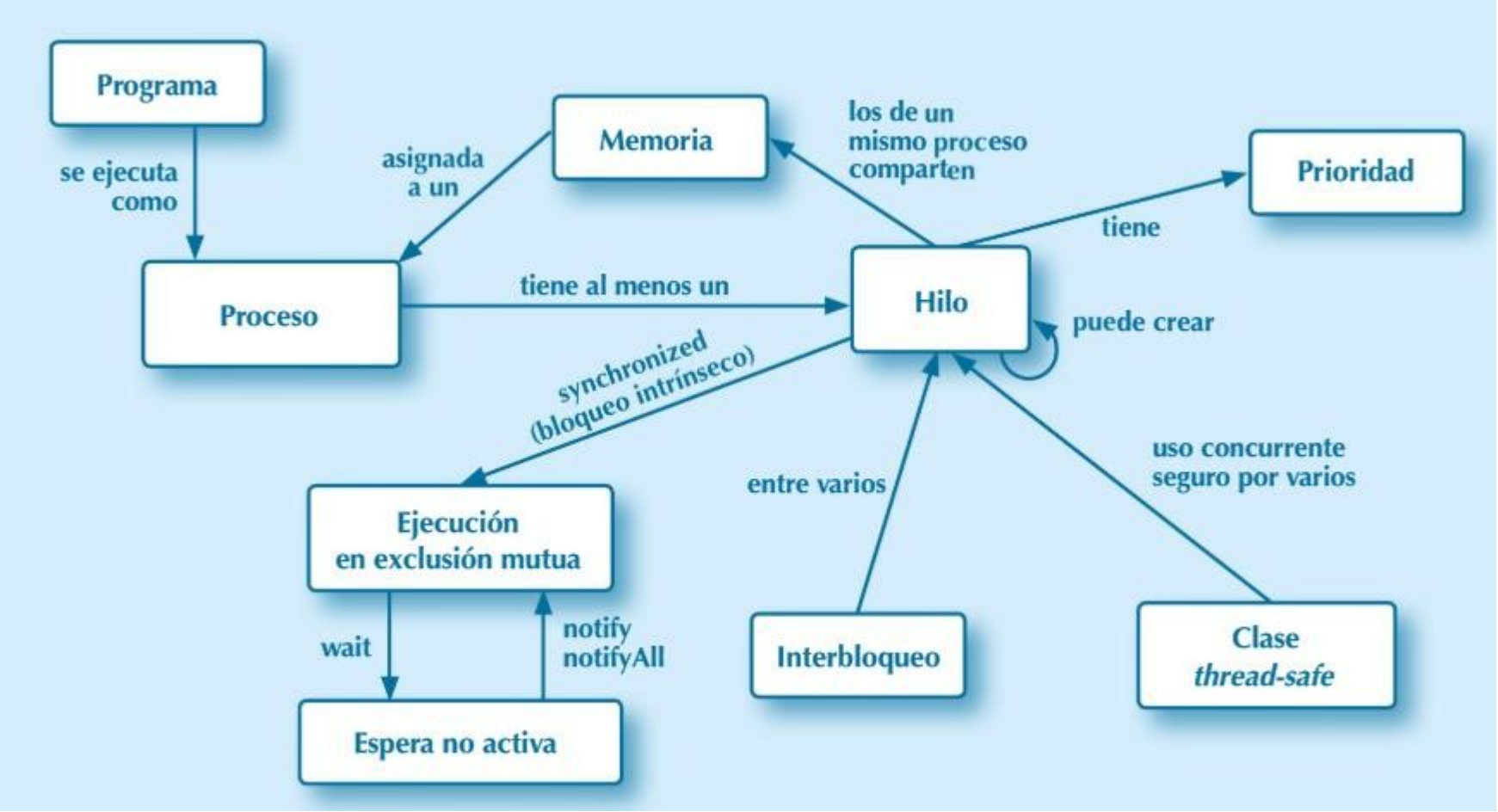
Gestión de hilos:

- Creación, ejecución y finalización de hilos.
- Sincronización de hilos. Exclusión mutua. Condiciones de sincronización.
- Compartición de información (comunicación) entre hilos. Recursos compartidos.
- Mecanismos de comunicación y sincronización de hilos (semáforos, monitores, paso de mensajes).
- Prioridades.
- Hilos demonio
- Problemas. Inanición, interbloqueos.
- Grupos (pool) de hilos.

- Temporizadores y tareas periódicas.

Programación de aplicaciones multihilo.

Programación multihilo

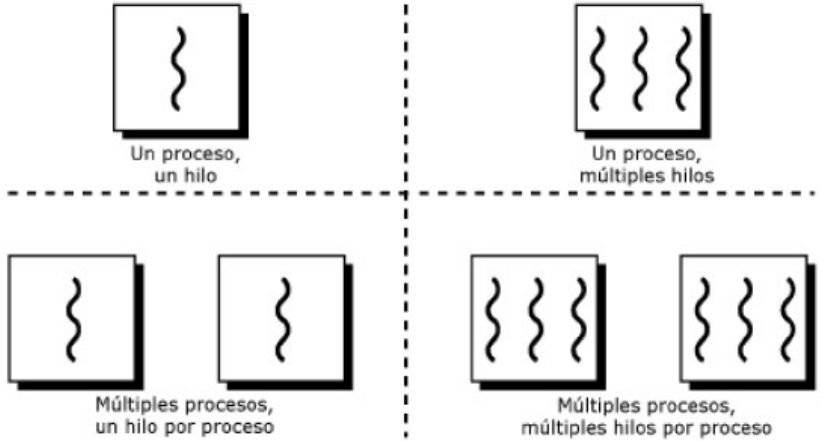


▪ Hilos

Un hilo(proceso ligero) es una **unidad básica de utilización de la CPU**, y consiste en un contador de programa, un juego de registros y un espacio de pila

Los hilos dentro de una misma aplicación comparten:
- La sección de código.
- La sección de datos.
- Los recursos del SO(archivos abiertos y señales).

Un proceso tradicional o pesado es igual a una tarea con un sol hilo.



Hilos

Los hilos permiten la ejecución concurrente de varias secuencias de instrucciones asociadas a diferentes funciones dentro de un mismo proceso, compartiendo un mismo espacio de direcciones y las mismas estructuras de datos del núcleo.

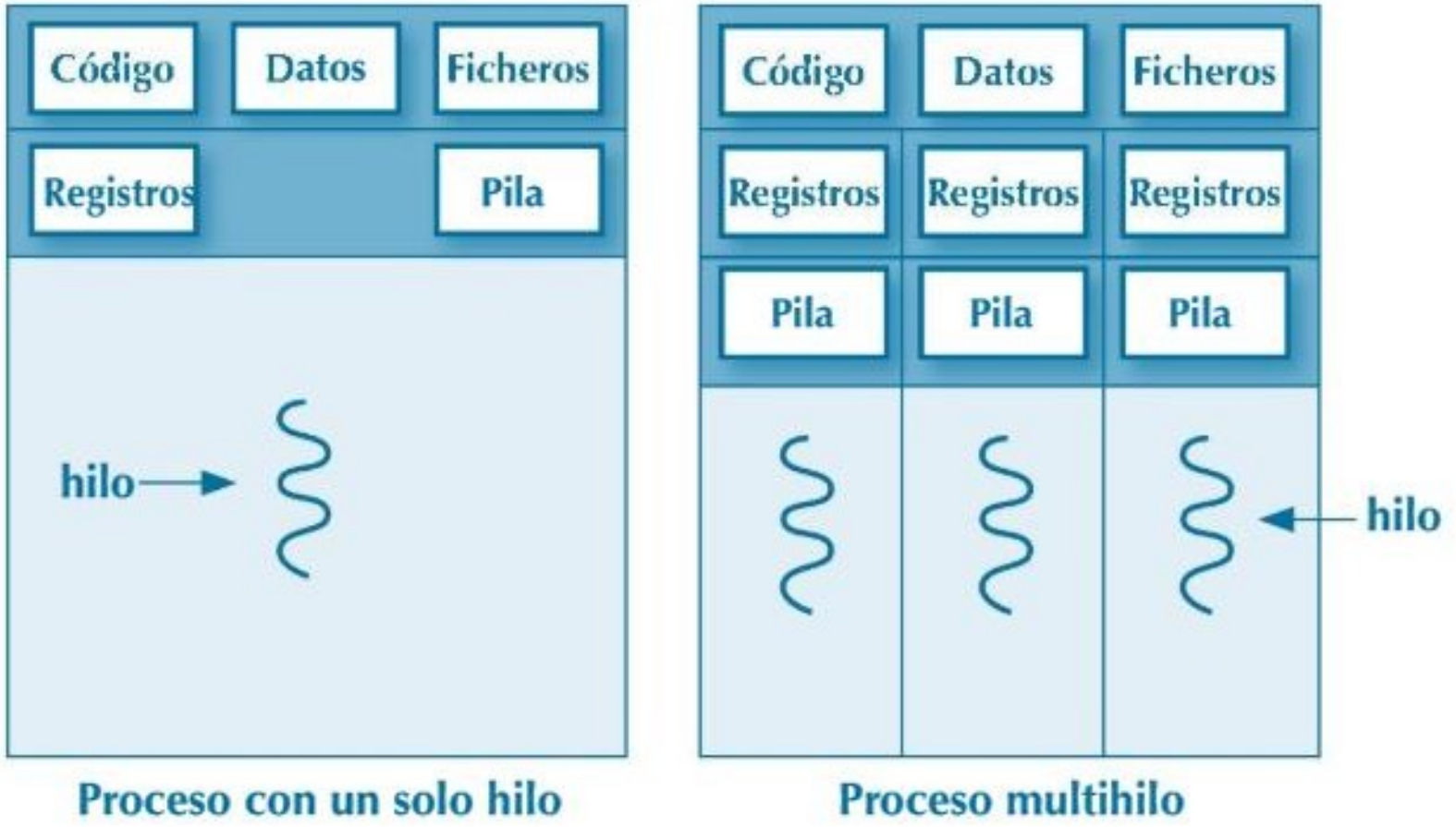
Recursos compartidos entre los hilos:

- Código(instrucciones).
- Variables globales.
- Ficheros y dispositivos abiertos.

Recursos no compartidos entre los hilos:

- **Contador del programa** (cada hilo puede ejecutar una sección distinta de código).
- **Registros de CPU**.
- Pila **para las variables locales** de los procedimientos a las que se invoca después de crear un hilo.
- **Estado**: distintos hilos pueden estar en ejecución, listos o bloqueados esperando un evento.

- **Hilos y procesos**



- **Hilos – clase Thread**

Método	Funcionalidad
`void run()`	Se ejecuta cuando se lanza el hilo. Es el punto de entrada del hilo, como el método `main()` es el punto de entrada del proceso.
`void start()`	Lanza el hilo. La JVM crea el hilo y ejecuta su método `run()`.
`static void sleep(long ms)` `static void sleep(long ms, long ns)`	Detiene la ejecución del hilo actualmente en ejecución durante un tiempo, que se puede indicar en microsegundos o en una combinación de microsegundos y nanosegundos.

`void join()`
`void join(long ms)`
`void join(long ms, long ns)`
	Espera a que termine el hilo. Se puede indicar un tiempo máximo de espera, bien en milisegundos, bien en una combinación de milisegundos y nanosegundos.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Thread.html

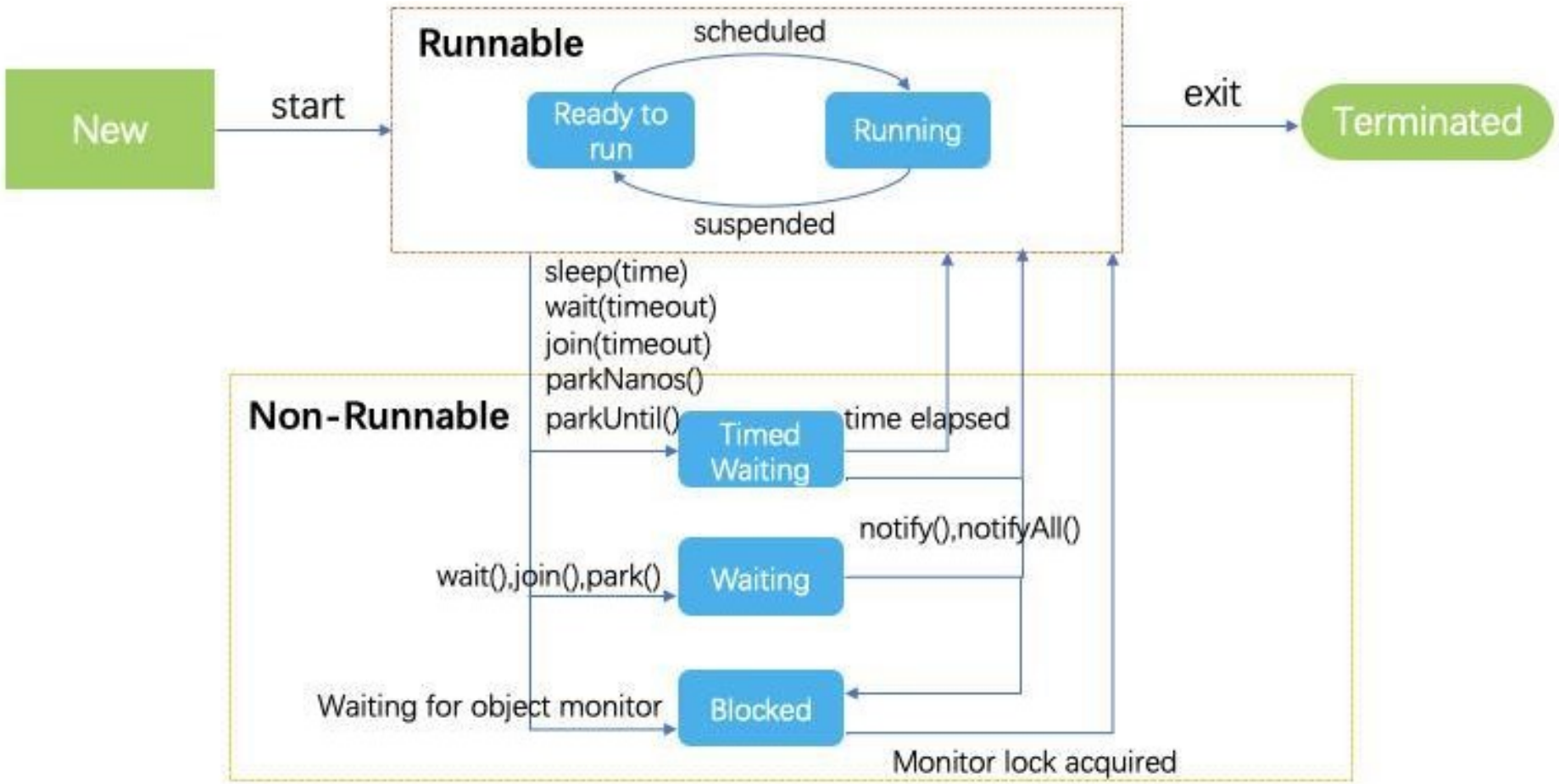
Hilos – clase Thread

`public void interrupt()`
`public boolean isInterrupted()`
`public static boolean interrupted()`
	El primer método interrumpe la ejecución de un hilo. El segundo método verifica si se ha interrumpido un hilo. El tercer método (estático) verifica si se ha interrumpido la ejecución del hilo actual, y borra el estado de interrupción, de manera que una llamada posterior devolvería `false`, a menos que se vuelva a interrumpir.

`boolean isAlive()`	Comprueba si el hilo está vivo. Un hilo está vivo cuando se ha iniciado y no ha terminado su ejecución.
`int getPriority()` `void setPriority(int nuevaPrior)`	Se puede asignar una prioridad a un hilo, y se puede obtener la prioridad de un hilo.
`static Thread currentThread()`	Devuelve un objeto de clase `Thread` correspondiente al hilo en ejecución actualmente.
`long getId()`	Devuelve el identificador del hilo.
`String getName()` `void setName(String nombre)`	Se puede asignar un nombre a un hilo, y se puede recuperar el nombre del hilo.
`Thread.State getState()`	Devuelve el estado del hilo.
`boolean isDaemon()` `void setDaemon(boolean on)`	Un hilo puede ser de tipo *daemon*. La distinción es importante, porque la JVM termina su ejecución cuando no queda ningún hilo activo o cuando solo quedan hilos de tipo *daemon*.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Thread.html

- **Hilos – Estados**



https://programmerclick.com/article/48881046531/

▪ Sección crítica . Condición de carrera .Condiciones para la exclusión mutua

Sección crítica: Segmento de código que manipula un recurso y debe ser ejecutado de forma atómica.

Condición de carrera: Situación en que el correcto resultado de un programa depende del orden en que se intercale la ejecución de las instrucciones en sus diferentes hilos.

Condiciones para la exclusión mutua

- Solamente se permite un proceso puede estar simultáneamente en la sección crítica de un recurso.
- No debe ser posible que un proceso que solicite acceso a una sección crítica sea postergado indefinidamente.
- Cuando ningún proceso este en una sección crítica, cualquier proceso que solicite su entrada lo hará sin demora.
- No se puede hacer suposiciones sobre la velocidad relativa de los procesos ni el número de procesadores.

- Un proceso permanece en su sección crítica durante un tiempo finito.

▪ Problemas Sincronización

• **Interbloqueo o punto muerto (deadlock)**

Los hilos no se ejecutan por quedarse a la espera de la ejecución de otros hilos

• **Innanición (starvation)**

Un hilo no puede acceder a un recurso compartido porque otros hilos no le permiten el acceso

• **Interbloqueo activo (livelock)**

Los hilos no se ejecutan porque se estorban mutuamente, precisamente por intentar ambos evitar un interbloqueo

▪ Sincronización

• **join**() Se espera la terminación del hilo que invoca a este método antes de continuar

• **Thread.sleep**(int) El hilo que ejecuta esta llamada permanece *dormido* durante el tiempo especificado como parámetro (en ms)

• **isAlive**() Comprueba si el hilo permanece activo todavía (no ha terminado su ejecución)

• **yield**() Sugiere al *scheduler* que sea otro hilo el que se ejecute en lugar del hilo actual (no se asegura). No queda suspendido, sino que sigue estando preparado para la ejecución. El planificador se activa y carga otro hilo para ejecutar, que podría ser el mismo.

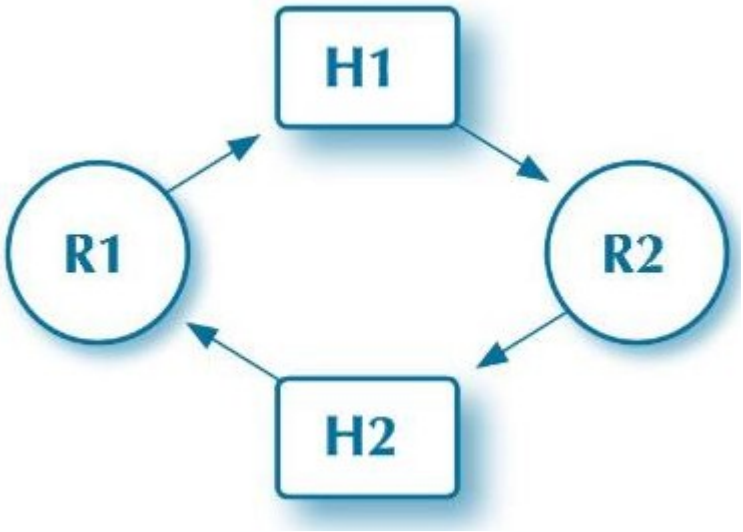
▪ Métodos sincronización

- Proporcionan exclusión mutua para acceder a un recurso compartido.
- Uso de synchronized al declarar un método public **synchronized** void metodo()
- No es posible ejecutar simultáneamente dos métodos sincronizados del mismo objeto.

Sincronizando todo, se pierde la ventaja de la concurrencia… pero se puede sincronizar únicamente la parte del código del método que interesa:

```
public void addName(String name)
{ //... synchronized ( this )
{ // Código importante, peligro de concurrencia } //... }
```

- **Interbloqueo**




```
public boolean transferencia(Cuenta c1, Cuenta c2, int cantidad) {
  boolean result = false;
  synchronized(c1) {
    synchronized(c2) {
      if (c1.getSaldo() >= cantidad) {
        c1.sacar(cantidad);
        c2.ingresar(cantidad);
        result = true;
      }
    }
  }
  return result;
}
```

Material de estudio

- https://docs.oracle.com/javase/tutorial/essential/concurrency/
- Programación concurrente
- https://www.tutorialspoint.com/java/java_multithreading.htm
- https://www.tutorialspoint.com/java_concurrency/index.htm
- https://www.dit.upm.es/~pepe/libros/concurrency/index.html#!1001