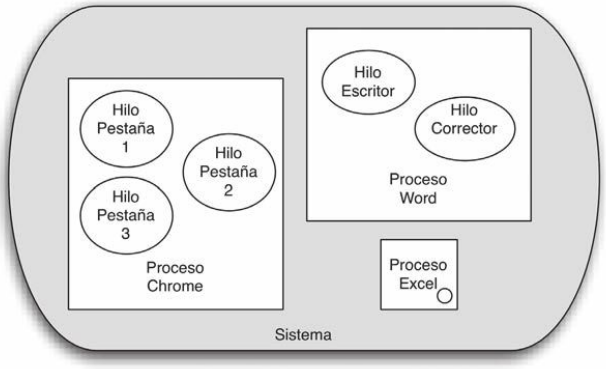
Programación de Procesos y Servicios

UT2: Programación de Hilos
2º DAM

Conceptos Básicos

Hilo o Thread: Unidad básica de utilización de la CPU, y más concretamente de un core del procesador. Así un thread se puede definir como la secuencia de código que está en ejecución, pero dentro del contexto de un proceso.

- Los hilos se ejecutan dentro del contexto de un proceso, por lo que dependen de un proceso para ejecutarse.
- Los procesos son independientes y tienen espacios de memoria diferentes, dentro de un mismo proceso pueden coexistir varios hilos ejecutándose que compartirán la memoria de dicho proceso.



Conceptos Básicos

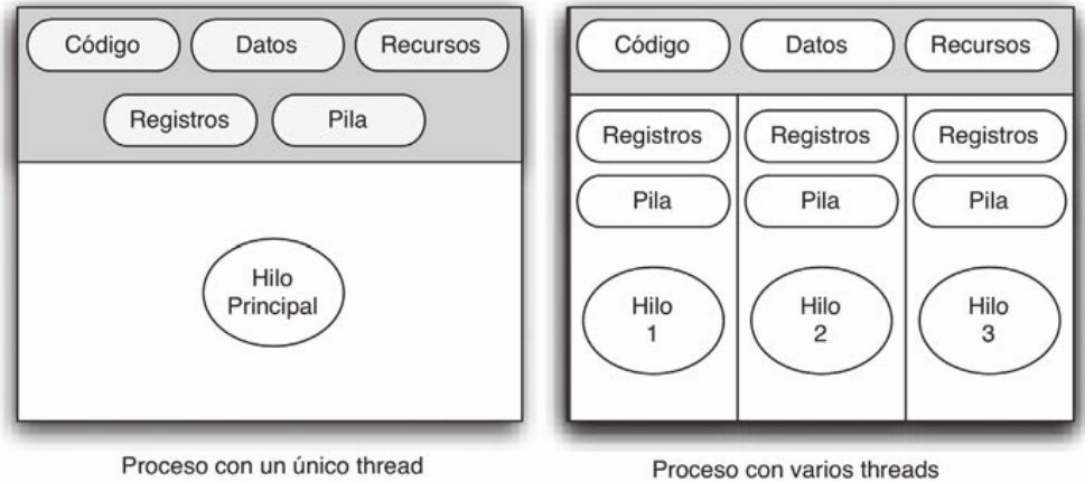
Frente a la multiprogramación de procesos, la multitarea presenta bastantes ventajas:

- Capacidad de respuesta: Los hilos permiten atender varias peticiones simultáneamente, mejorando la interacción del usuario, especialmente en servidores.
- Compartición de recursos: Los hilos comparten la memoria del proceso, facilitando la comunicación entre ellos sin necesidad de mecanismos adicionales, aunque requiere sincronización para evitar conflictos.
- Eficiencia de recursos: Crear hilos es más barato en términos de memoria y recursos que crear nuevos procesos.
- Paralelismo real: Los hilos pueden ejecutarse en múltiples núcleos, aprovechando el paralelismo en arquitecturas multicore, algo que no es posible con procesos independientes.

Recursos Compartidos

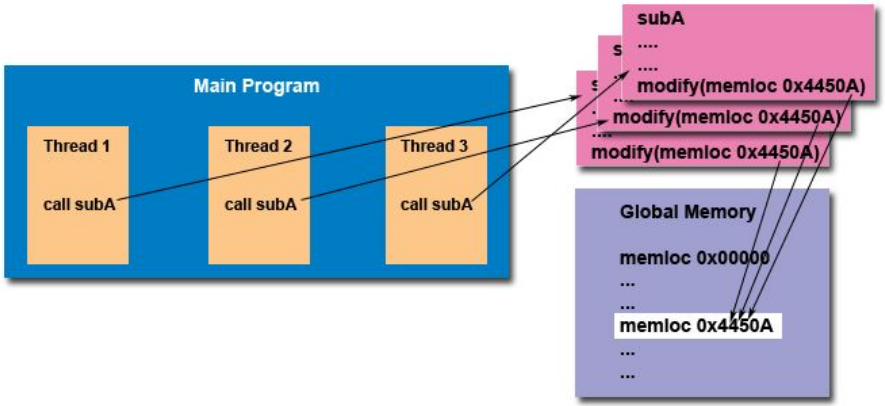
Los procesos mantienen su propio espacio de direcciones y recursos de ejecución mientras que los hilos dependen del proceso, por lo que comparten con otros hilos la sección de código, datos y otros recursos. Cada hilo tiene:

- Contador de programa
- Registros de la CPU
- Pila de ejecución



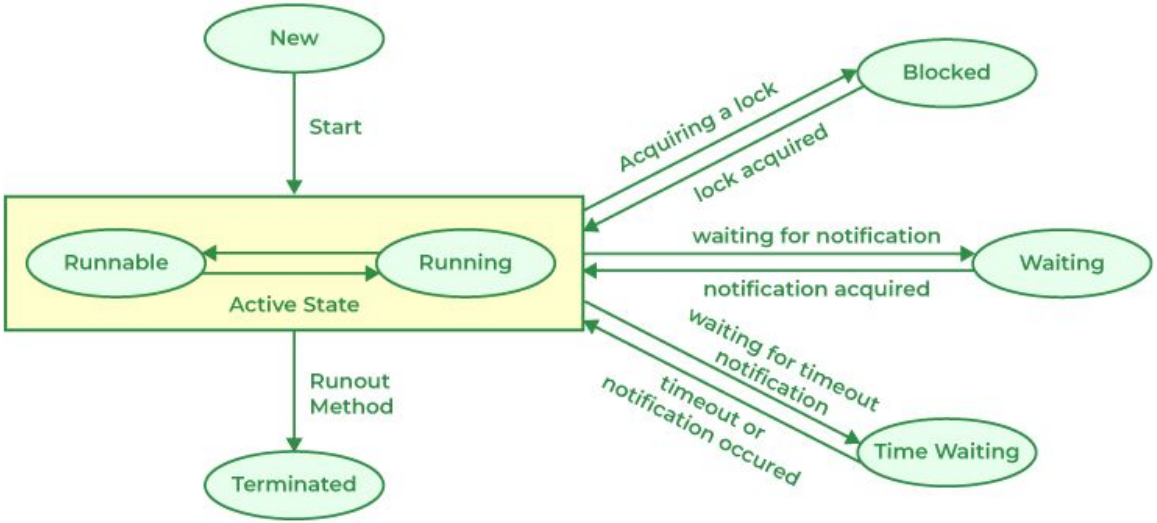
Conceptos Básicos

Los hilos comparten el espacio de memoria y recursos del proceso, lo que hace su creación más eficiente que la de procesos, que requieren estructuras adicionales y reserva de recursos. Un programa tiene un hilo principal que puede crear otros hilos para ejecutar tareas diferentes de forma independiente



Estados de un Hilo

Al igual que los procesos, los hilos pueden cambiar de estado a lo largo de su ejecución. Su comportamiento dependerá del estado en el que se encuentren:



Clase Thread

Thread es la clase base de Java para definir hilos de ejecución concurrentes dentro de un mismo programa

En Java, como lenguaje orientado a objetos, el concepto de concurrencia está asociado a los objetos

- Son los objetos los que actúan concurrentemente con otros

Las clases de objetos que puedan actuar concurrentemente deben extender la clase ***Thread***

- Ejemplo:

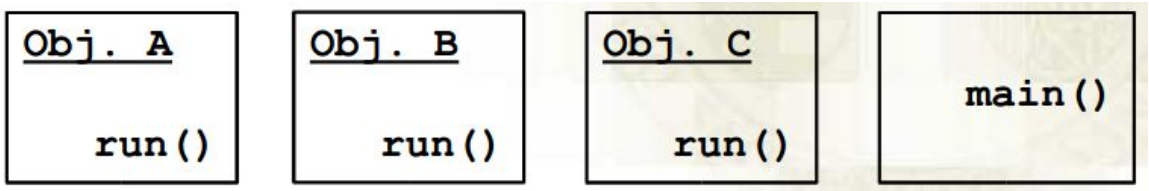
```
class miClaseConcurrente extends Thread {...}
```

Clase Thread: run

Las clases derivadas de *Thread* deben incluir el método *run*, que especifica la tarea a realizar:

```
public void run()
```

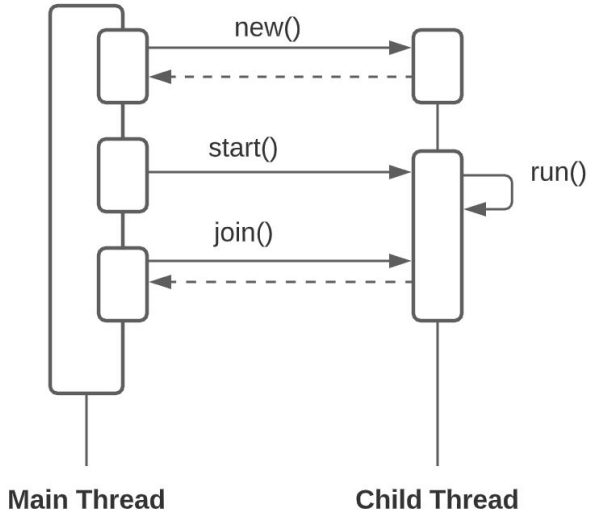
La ejecución del método *run* de un *Thread* puede realizarse concurrentemente con otros métodos *run* de otros *Thread* y con el método *main*.



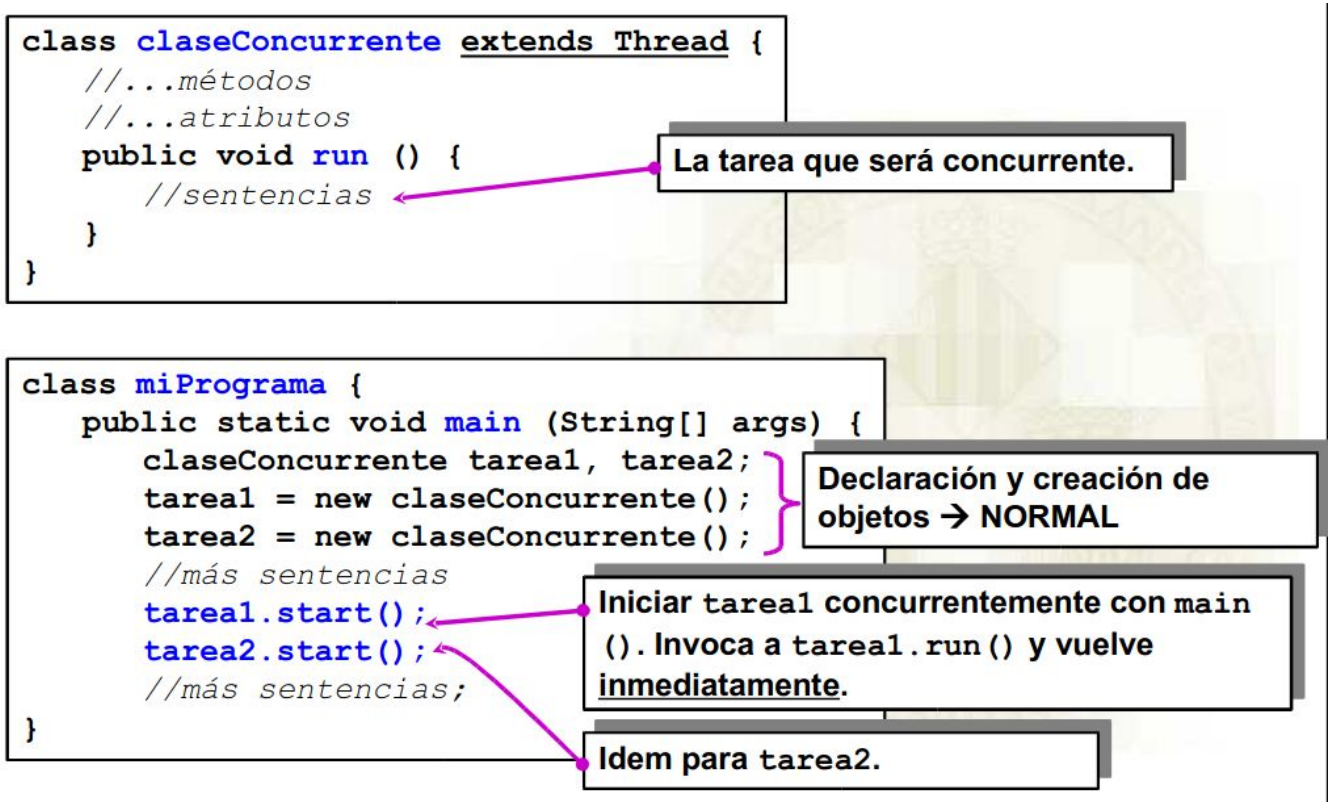
Clase Thread: start

El inicio de la ejecución de una tarea se realiza mediante el método *start()* heredado de Thread.

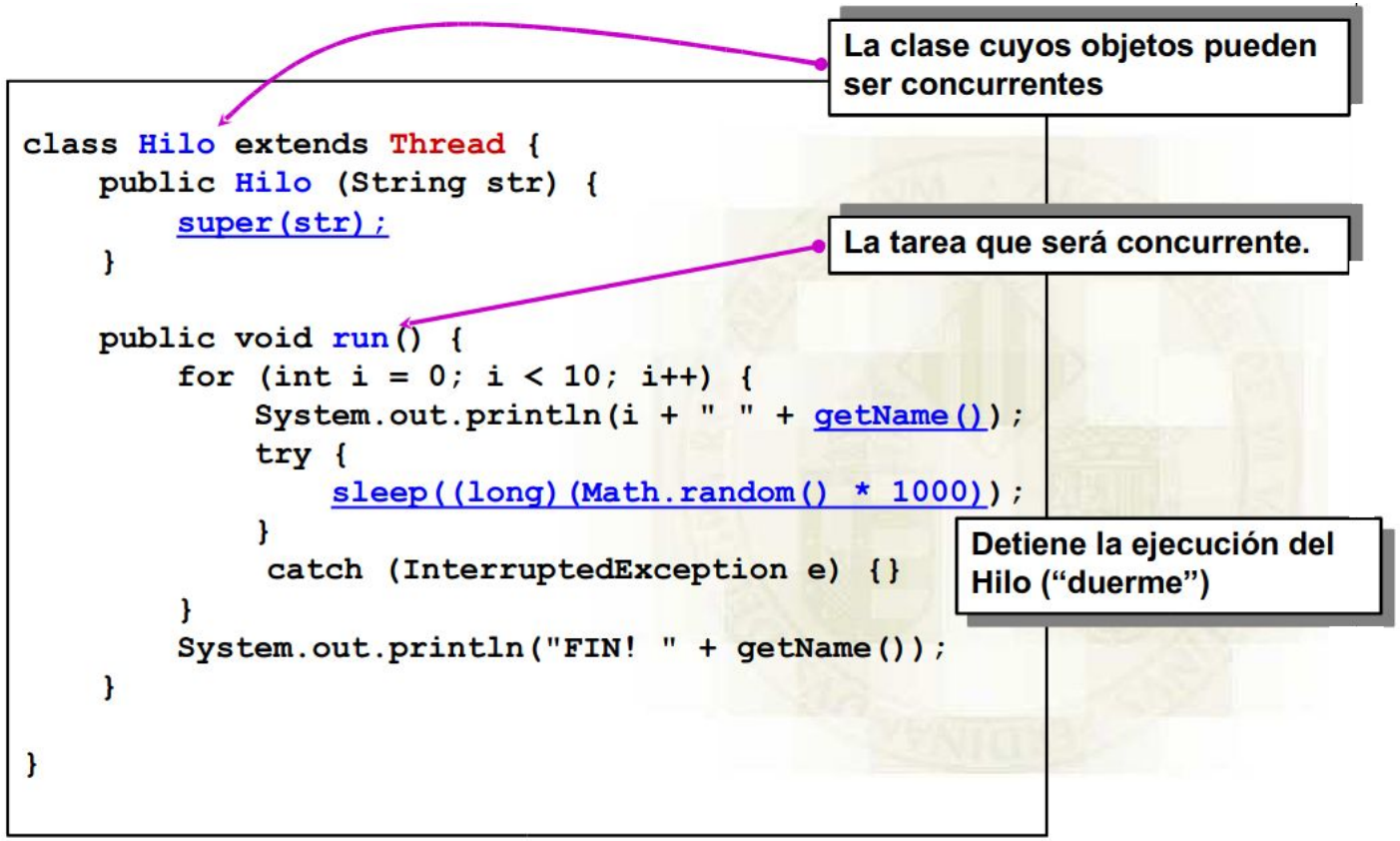
start() es un método especial que invoca *run()* y devuelve inmediatamente el control a la tarea que lo ha llamado



Clase Thread: Lanzar tarea concurrente



Clase Thread: Ejemplo




```
class Hilo extends Thread {
    public Hilo (String str) {
        super(str);
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            }
             catch (InterruptedException e) {}
        }
        System.out.println("FIN! " + getName());
    }

}
```

Clase Thread: Ejemplo

```
public class Demo {
    public static void main (String[] args) {
        Hilo uno, dos;

        uno = new Hilo("Jamaica");
        dos = new Hilo("Fiji");

        uno.start();
        dos.start();

        System.out.println("main no hace nada");
    }
}
```

Crear los objetos con un nombre

Arrancar los 2 hilos

Clase Thread: Ejemplo

```
public class Demo {
    public static void main (String[] args) {
          Hilo uno, dos;

          uno = new Hilo("Jamaica");
          dos = new Hilo("Fiji");

          uno.start();
          dos.start();

          System.out.println("main no hace nada");
    }
}
```

```
main no hace nada
0 Jamaica
0 Fiji
1 Jamaica
1 Fiji
2 Jamaica
2 Fiji
3 Jamaica
3 Fiji
4 Fiji
5 Fiji
4 Jamaica
6 Fiji
7 Fiji
5 Jamaica
8 Fiji
6 Jamaica
9 Fiji
7 Jamaica
8 Jamaica
FIN! Fiji
9 Jamaica
FIN! Jamaica
```

Interfaz Runnable

Problema: ¿Qué ocurre si se desea hacer concurrentes los objetos de una clase que hereda de otra que no es Thread?

- Ejemplo: `class Circulo extends Figura {...}`
- No puede heredar también de Thread → Java no permite la herencia múltiple

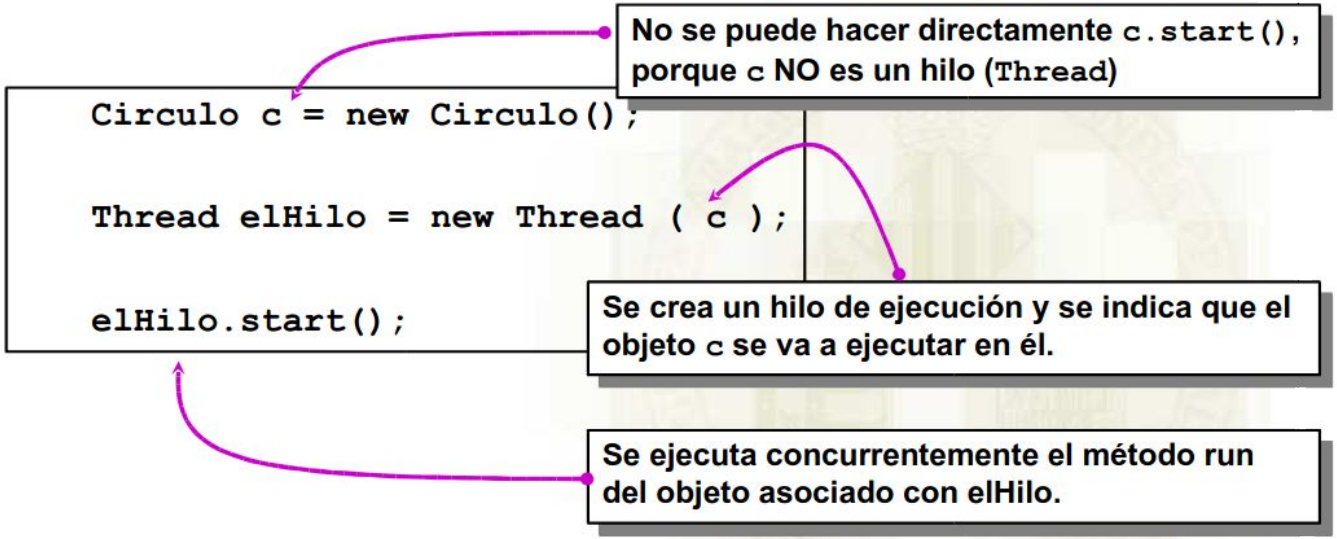
Solución: Java permite implementar múltiples interfaces → `interface Runnable`

Interfaz Runnable

Para crear un hilo utilizando la interfaz *Runnable* se debe crear una nueva clase que implemente la interfaz,teniendo que implementar únicamente el método *run()* con la tarea a realizar.

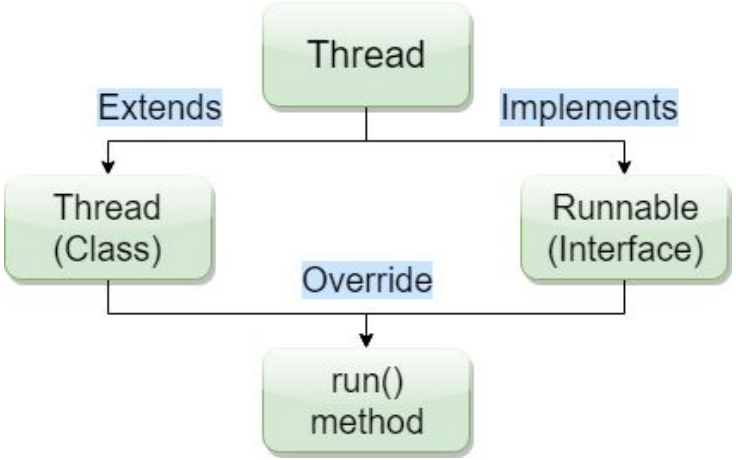
```
class Circulo extends Figura implements Runnable {...}
```

Interfaz Runnable



¿Interfaz Runnable o Subclass de Thread?

No hay nada que indique que una forma es mejor que otra. Ambos métodos son similares y el resultado es el mismo. El método preferido debería ser implementar Runnable, y pasarle la instancia al constructor de Thread.

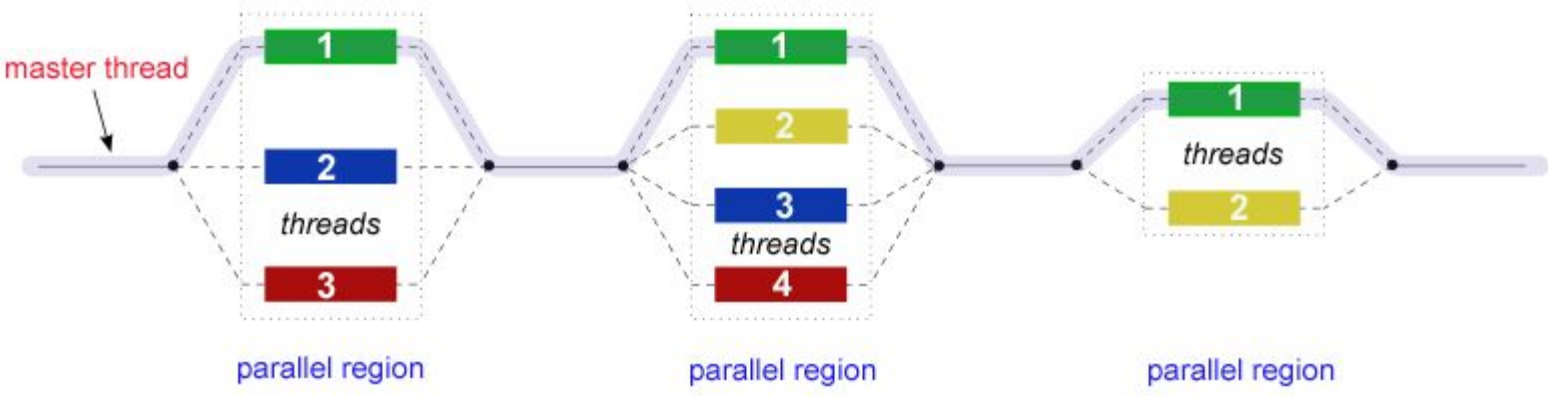


Reunificación de tareas (join)

Un programa concurrente, en general, tiene bloques secuenciales:

- Existe concurrencia en partes concretas del programa, no necesariamente en todo él.

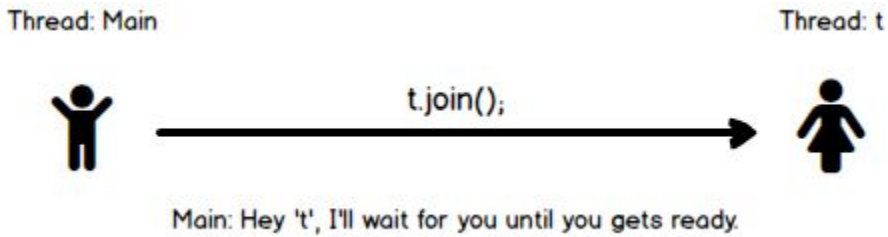
Después de bloques concurrentes puede ser preciso reunificar el flujo de control del programa para continuar la secuencia.



Reunificación de tareas (join)

Método join()

Permite la reunificación de hilos. Se llama desde otro *thread* y hace que el *thread* que lo invoca se bloquee hasta que el *thread* termine. Es parecido a *p.waitFor()* para los procesos.



Reunificación de tareas (join)

Ejercicio

Crea un programa en Java que utilice hilos para realizar operaciones sobre una lista de números enteros.

- Un hilo debe calcular la suma de todos los números de la lista,
- Otro hilo calculará el producto de esos mismos números.
- El hilo principal debe restar el resultado de la multiplicación al de la suma.

Finalmente, imprime en pantalla los resultados de la suma, la multiplicación y la resta.

Interrupciones

Una **interrupción** es una señal que se envía a un hilo para indicarle que debe dejar de hacer lo que está haciendo y terminar lo antes posible.

- No detiene el hilo de manera forzada, sino que permite que el hilo sepa que ha sido interrumpido para que tome las acciones apropiadas, como finalizar su ejecución o liberar recursos.

Interrupciones

Cuando un método es interrumpido:

- Se establece un **flag de interrupción** dentro del hilo, que indica que fue interrumpido.
- Si el hilo estaba bloqueado en una operación que puede lanzar una excepción, como *sleep()*, *wait()* o *join()*, se lanzará una excepción InterruptedException.

Un hilo envía una interrupción mediante la invocación del método *interrupt()* en el objeto del hilo que se quiere interrumpir.

```
thread.interrupt();
```

Interrupciones

Una vez comprobado si el hilo ha sido interrumpido se puede o bien finalizar su ejecución, o lanzar *InterruptedException* para manejarla en una sentencia *catch* centralizada en aplicaciones complejas.

```
public void run() {
      for (int i = 0; i < NDatos; i++) {
            try {
                  System.out.println("Esperando a recibir dato!");
                  Thread.sleep(500);
            } catch (InterruptedException e) {
                  System.out.println("Hilo interrumpido.");
                  return;
            }
            // Gestiona dato i
      }
      System.out.println("Hilo finalizando correctamente.");
}
```

Interrupciones

Queda a criterio del programador decidir exactamente cómo un hilo responde a una interrupción, pero en muchos de los casos lo que se hace es finalizar su ejecución.

Resumen Clase Thread

Método	Tipo de retorno	Descripción
start()	*void*	Implementa la operación *create.* Comienza la ejecución del hilo de la clase correspondiente. Llama al método *run()*
run()	*void*	Si el hilo se construyó implementando la interfaz *Runnable*, entonces se ejecuta el método *run()* de ese objeto. En caso contrario, no hace nada
currentThread()	*static Thread*	Devuelve una referencia al objeto hilo que se está ejecutando actualmente
join()	*void*	Implementa la operación *join* para hilos
sleep(long milis)	*static void*	El hilo que se está ejecutando se suspende durante el número de milisegundos especificado
interrupt()	*void*	Interrumpe el hilo del objeto
interrupted()	*static boolean*	Comprueba si el hilo ha sido interrumpido
isAlive()	*boolean*	Devuelve *true* en caso de que el hilo esté vivo, es decir, no haya terminado el método *run()* en su ejecución

Prioridades

El criterio para la ejecución de múltiples hilos sobre una CPU se basa en prioridades.

- Java utiliza por defecto un planificador apropiativo. Si un hilo con mayor prioridad pasa al estado "Runnable", el sistema interrumpirá la ejecución de hilos con menor prioridad para darle paso.
- Cuando los hilos tienen la misma prioridad, el planificador del sistema asigna los núcleos de manera equitativa, a menudo utilizando tiempo compartido si el sistema operativo lo permite.
- Los hilos heredan la prioridad del hilo que los crea

Prioridades

setPriority()

La prioridad de los hilos se puede establecer utilizando el método *setPriority()* de la clase *Thread*. Las prioridades de un hilo varían en un rango de enteros comprendido entre *MIN_PRIORITY* y *MAX_PRIORITY.*