



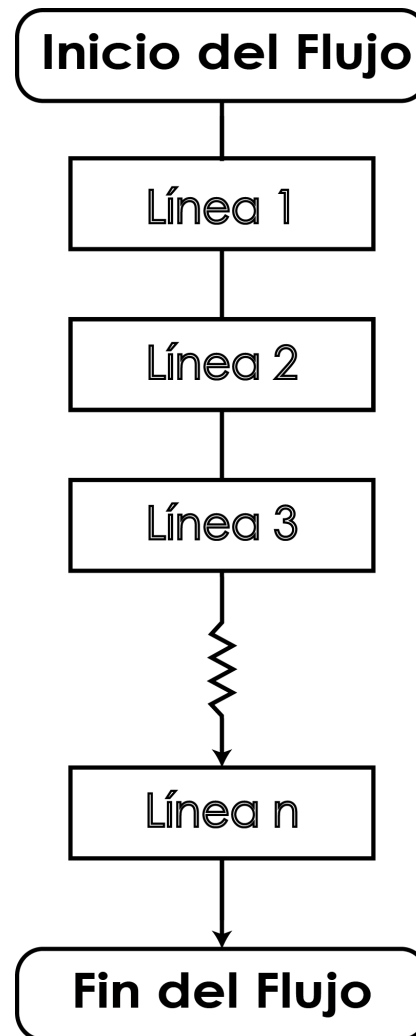
Introducción a la Programación en Java



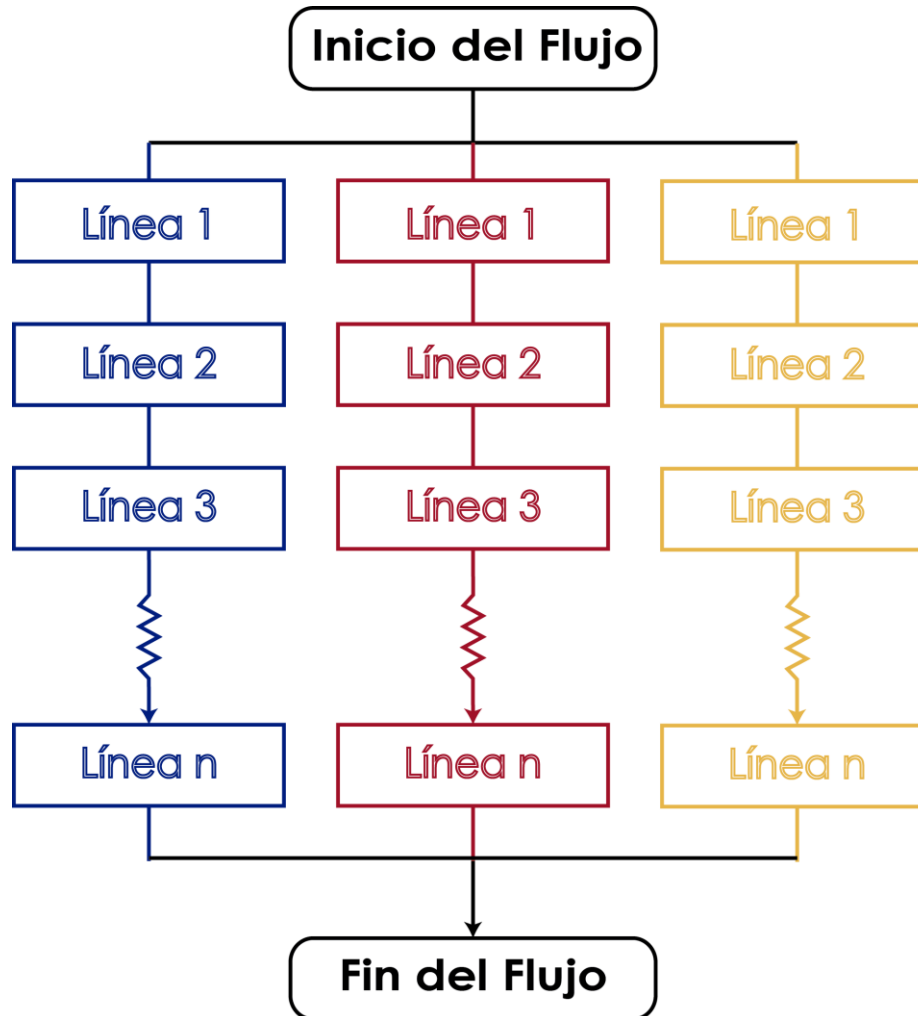
Multihilo en Java

- Java es un lenguaje de programación multihilo.
- El hecho de ser multihilo se refiere a que dos o más tareas se ejecutan "aparentemente" a la vez, dentro de un mismo programa.

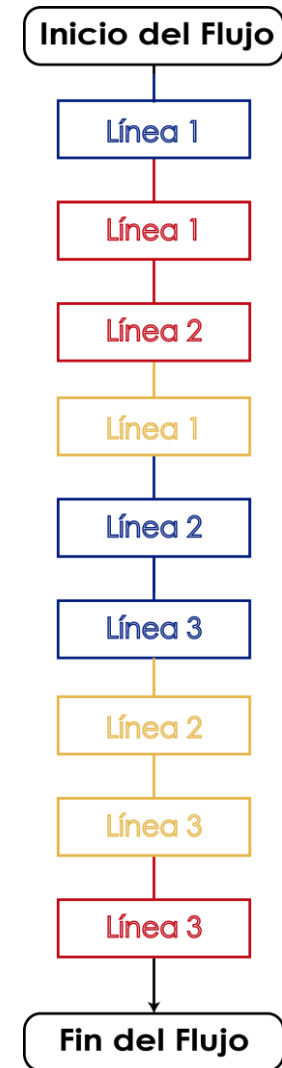
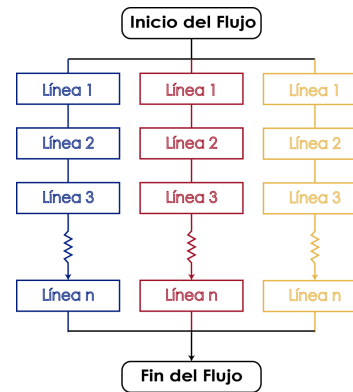
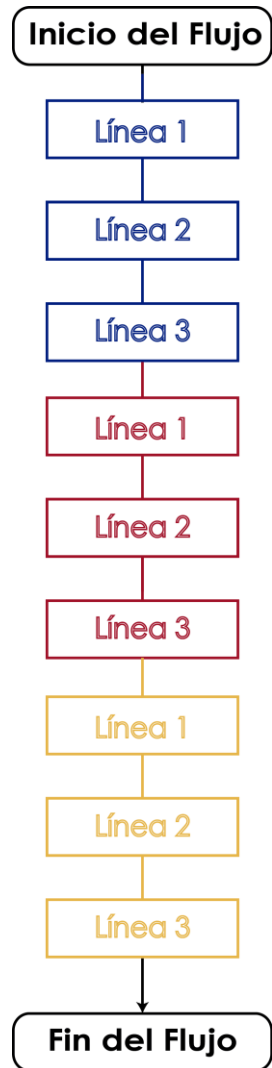
Multihilo en Java (cont.)



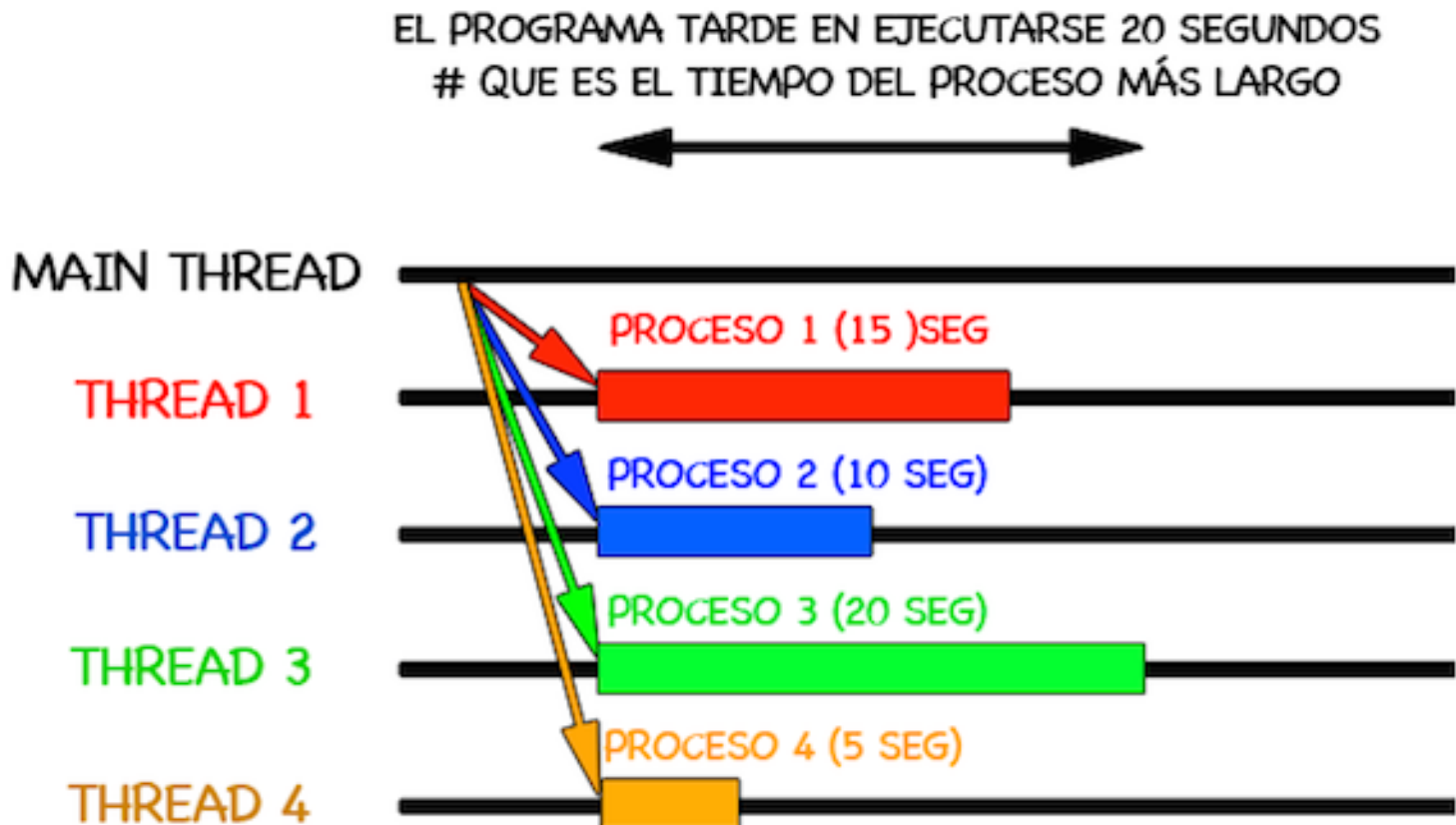
Multihilo en Java (cont.)



Multihilo en Java (cont.)



Multihilo en Java (cont.)



Clase Thread e interfaz Runnable

- En Java para utilizar multihilo se debe usar la clase **Thread**. Dicha clase implementa la Interface **Runnable**. En el siguiente diagrama de clase se muestra la Interface Runnable y la clase Thread con sus principales métodos:



Interfaz (cont.)

- Ejemplo de Interfaz

```
package java.lang;
```

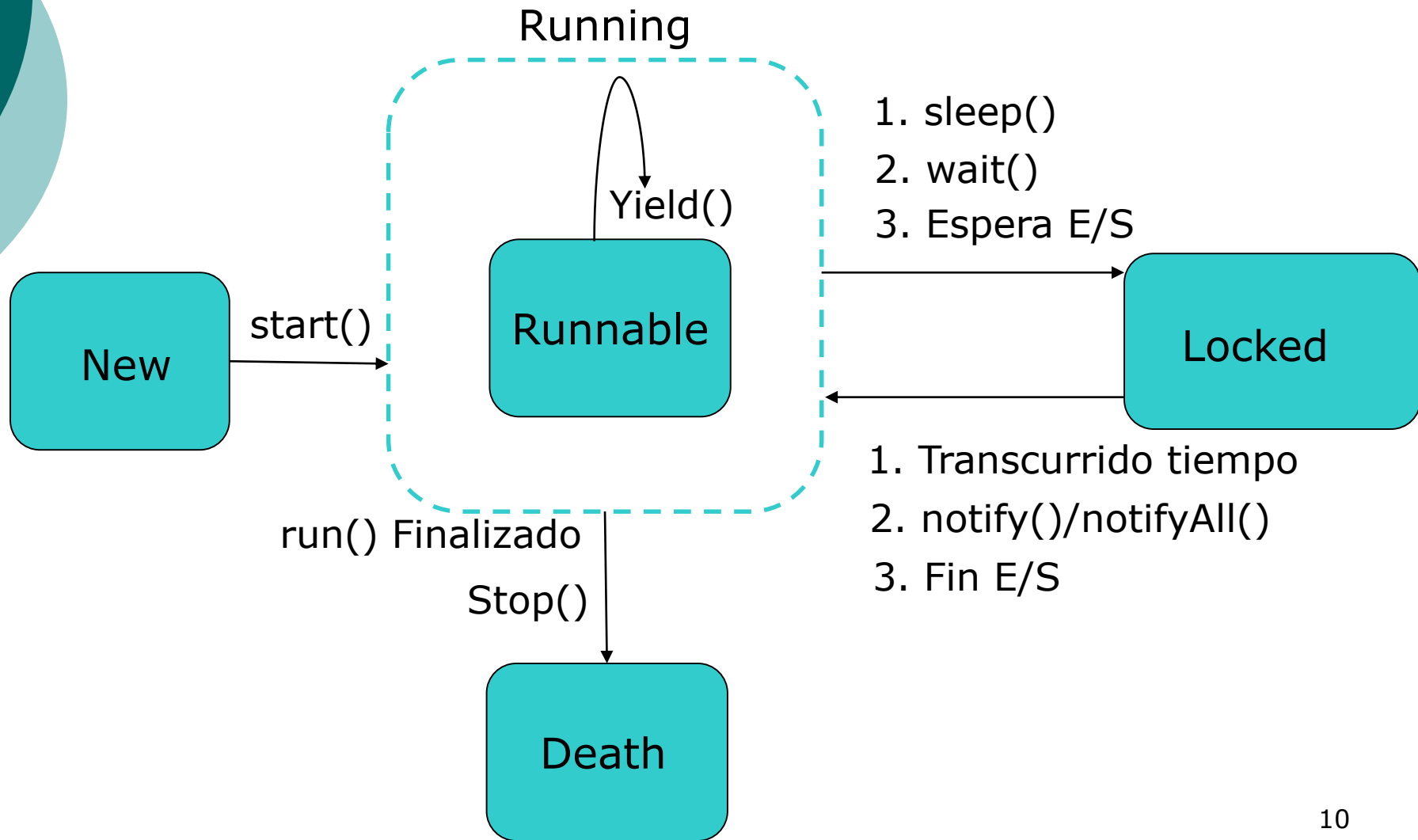
```
public interface Runnable {  
    public abstract void run() ;  
}
```




Principales métodos de la clase Thread

- `run()`: la tarea concurrente es desarrollada en este método
- `start()`: inicia la ejecución de una tarea. Invoca al método `run`
- `sleep()`: pone a dormir un hilo por un tiempo mínimo especificado
- `getName()`: devuelve el nombre de este hilo.
- `yield()`: se detiene temporalmente la ejecución del hilo y permite que el siguiente hilo disponible se ejecute

Estados de un hilo



Clase Thread e interface Runnable

- Hay dos modos de conseguir hilos de ejecución (threads) en Java.
 1. Extender una clase X de la clase *Thread* y redefiniendo el método `run()`.
 2. Implementar la interfaz *Runnable* y definir el método `run()` de dicha interfaz

Nota: La razón de que existan estas dos posibilidades se debe a que en Java no existe la herencia múltiple. Si una clase hereda de otra no puede heredar también de *Thread*, pero si puede implementar la interfaz *Runnable*

Heredando de la clase Thread

Extender una clase X de la clase *Thread* y redefiniendo el método `run()`.

```
public class Hilo extends Thread {  
  
    /**  
     * La clase Thread tiene varios constructores, además del constructor por  
     * defecto (sin argumentos). Al constructor de la subclase Hilo le  
     * pasamos el nombre del hilo y éste se lo pasa al constructor de la  
     * clase base Thread mediante la palabra reservada super.  
     */  
    public Hilo(String nombre) {  
        super(nombre);  
    }  
  
    @Override  
    public void run() {           
        for (int i = 1; i < 10; i++) { //The java.lang.Thread.getName()  
            System.out.println(getName() + ": " + i); //retorna el nombre del hilo.  
        }  
    }  
}
```

Heredando de la clase Thread (cont.)

- Crear dos o más objetos de la clase Hilo en el cuerpo del método **main**.

```
public static void main(String[] args) {  
    // TODO code application logic here  
  
    /**  
     * El nombre de cada hilo se pasa por el constructor.  
     * El hilo está en el estado New, está inicializado,  
     * y listo para ponerlo en marcha llamando al método  
     * start() de la superclase Thread.  
     */  
    Hilo hilo1 = new Hilo("Hilo 1");  
    Hilo hilo2 = new Hilo("Hilo 2");  
}
```

Nota: El hilo se encuentra en el estado *New*, inicializado y listo para ponerlo en marcha llamando al método **start()** de la superclase Thread.

Heredando de la clase Thread (cont.)

- Poner en marcha los hilos

```
public static void main(String[] args) {  
    // TODO code application logic here  
  
    /**  
     * El nombre de cada hilo se pasa por el constructor.  
     * El hilo está en el estado New, está inicializado,  
     * y listo para ponerlo en marcha llamando al método  
     * start de la superbase Thread.  
     */  
    Hilo hilo1 = new Hilo("Hilo 1");  
    Hilo hilo2 = new Hilo("Hilo 2");  
  
    hilo1.start();  
    hilo2.start();  
}
```

Nota: El método **start()** crea los recursos del sistema necesarios para que el hilo se ejecute y a continuación llama al método **run()**, el hilo se dice que está en el estado *Runnable*.

Ver ejemplo: Inicial A, no llama al método sleep().

Heredando de la clase Thread (cont.)

Pausa en la ejecución del hilo

```
public void run() {  
    for(int i=1; i<10; i++){  
        System.out.println(getName()+" : "+i);  
        try{  
            sleep(1000); //El argumento son milisegundos ██████████  
        }catch(InterruptedException ex){} //sleep genera un excepcion  
    }  
}
```

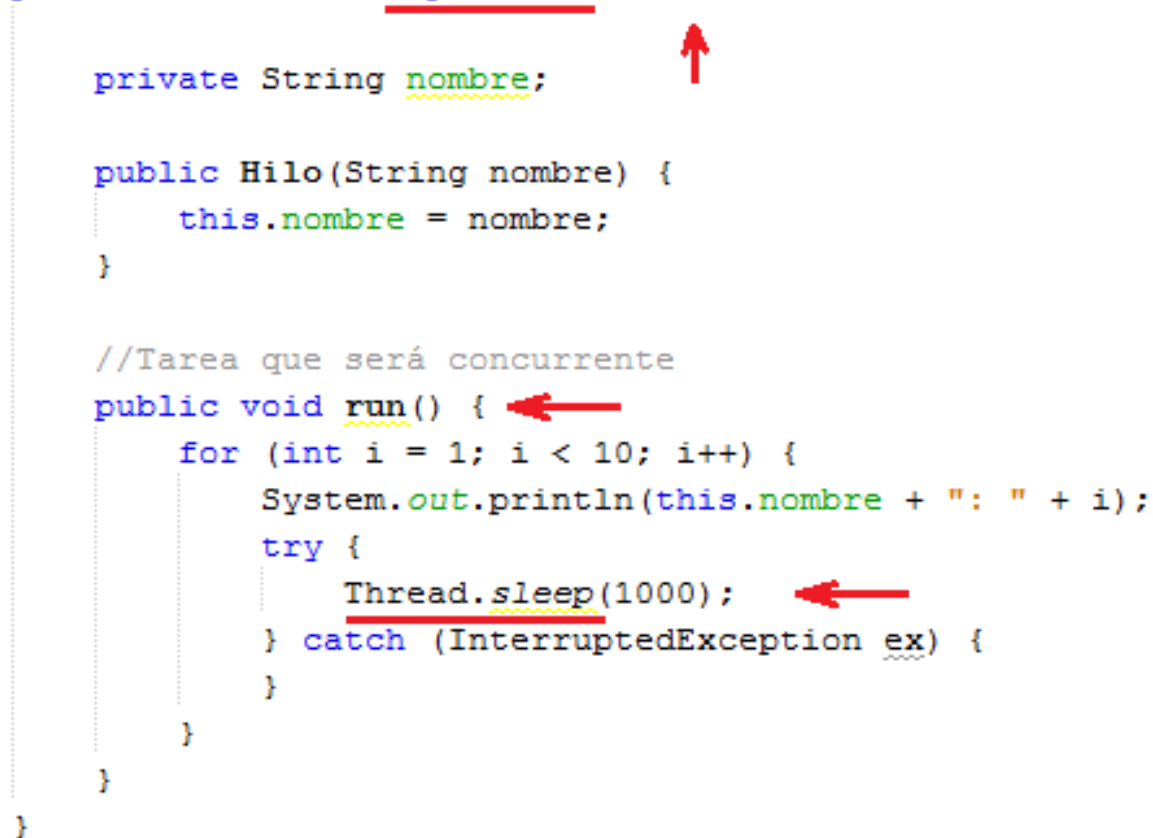
Nota: Cuando se llama al método **sleep()**, el hilo pasa del estado *Runnable* al estado *Locked*, dando la oportunidad a otros hilos de ejecutarse.

Ver ejemplo: Inicial A, usando el método sleep().

Implementar la interfaz *Runnable*

Implementar la interfaz *Runnable* y definir el método `run()`

```
public class Hilo implements Runnable {  
    private String nombre;  
  
    public Hilo(String nombre) {  
        this.nombre = nombre;  
    }  
  
    //Tarea que será concurrente  
    public void run() {  
        for (int i = 1; i < 10; i++) {  
            System.out.println(this.nombre + ": " + i);  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException ex) {  
            }  
        }  
    }  
}
```



Implementar la interfaz *Runnable* (cont.)

- Crear dos o más objetos de la clase `Hilo` en el cuerpo del método **main**.

```
public static void main(String[] args) {  
    // TODO code application logic here  
  
    //Crear dos o más objetos de la clase Hilo  
    Hilo hilo1 = new Hilo("hilo1");  
    Hilo hilo2 = new Hilo("hilo2");  
  
    //Crear un objeto de la clase Thread y pasarle un Runnable  
    Thread thread1 = new Thread(hilo1);  
    Thread thread2 = new Thread(hilo2);  
  
    //Poner en marcha los hilos  
    thread1.start();  
    thread2.start();  
}
```

Ejemplos.

- Ejercicios
 - Proceso de cobro en un supermercado de modo secuencial
 - Proceso de cobro en un supermercado de modo multihilo (**heredando de la clase Thread**)

Proceso de cobro en un supermercado

Simular el proceso de cobro de un supermercado; es decir, los clientes llevan su carro lleno de productos y una cajera les cobra los productos, pasándolos uno a uno por el escáner de la caja registradora. En este caso la cajera debe de procesar la compra cliente a cliente, es decir que primero le cobra al cliente 1, luego al cliente 2 y así sucesivamente.

Para ello se va a definir una **clase "Cajera"** y la **clase "Cliente"** el cual tendrá un "array de enteros" que representarán los productos que ha comprado y el tiempo que la cajera tardará en pasar el producto por el escáner.

Es decir, se tiene un array con [1,3,5] significará que el cliente ha comprado 3 productos y que la cajera tardará en procesar el producto 1, "1 segundo"; el producto 2, "3 segundos" y el producto 3 en "5 segundos", con lo cual tardará en cobrar al cliente toda su compra "9 segundos".

Ver ejemplo Cajera – Cliente.



Práctica

- Práctica
 - Proceso de cobro en un supermercado de modo multihilo (**implementando la interface Runnable**)

Hemos terminado.

