

Programación Multi-hebra en Java

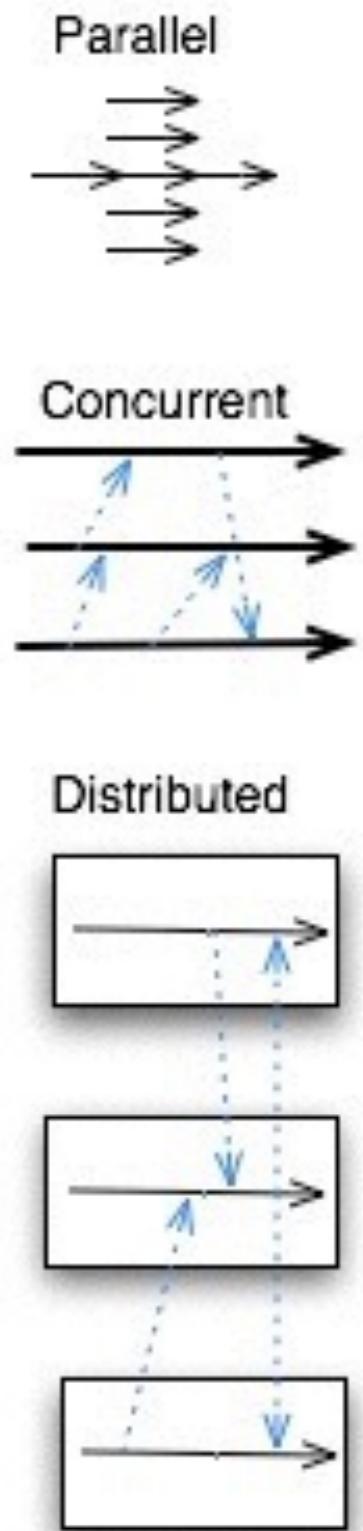
Parte I - Lo Básico

Java Multithreading
Part I - Basics

Samir Genaim

Paralelo, Concurrente, Distribuido ...

- ◆ **Paralelismo.** Se trata de coger un programa y hacer que se ejecute más rápido, dividiendo el código en partes que se pueden ejecutar simultáneamente.
- ◆ **Concurrencia.** Se trata de sistemas que tienen varias partes que están diseñadas para ejecutar simultáneamente, no porque lo hace más rápido, sino porque es una parte necesaria de su funcionalidad.
- ◆ **Distribución.** Se trata de sistemas compuestos de varias partes físicas conectadas por una red de comunicación, cada una ejecutando posiblemente diferentes programas, etc.



Hebras y Procesos

- ◆ Manera de implementar concurrencia/parallelismo
- ◆ Un programa puede tener varios procesos que se ejecutan "simultáneamente", cada uno tiene su propia memoria, archivos , etc. Cada proceso es "como se fuera un programa separado".
- ◆ Cada proceso puede tener varios hebras que comparten sus recursos (memoria , etc.)
- ◆ Normalmente, un programa tiene como mínimo un proceso y cada proceso tiene como mínimo una hebra.
- ◆ En Java, cuando se ejecuta un programa, se crea un proceso con una hebra que ejecuta el método main.
- ◆ Nos centraremos en hilos de Java.

¿Qué Significa “Ejecutando Simultáneamente (en Paralelo)?”?

- ◆ Esto depende de muchos factores. P. ej., si tenemos n hebras ejecutando en paralelo en una plataforma con $m \geq n$ CPUs/Cores, podemos pensar que realmente están ejecutando en paralelo, aunque no tienen que ...
- ◆ Sin embargo, normalmente tenemos hebras (en todos los programas) más que CPUs/Cores.
- ◆ Un gestor de hebras (a nivel del sistema operativo, la JVM, hardware, etc.) alterna la ejecución de las hebras, es decir, se ejecuta un poco de cada hebra para que parezca que están ejecutando en paralelo.

Crear y Empezar una Hebra - I

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

see: examples.threads.ex1.ThreadEx1

Crear y Empezar una Hebra - I

```
public class ThreadEx1 extends Thread {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

Definir una clase que extiende de la clase `java.util.Thread`

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
  
    t1.start();  
    t2.start();  
}
```

Crear y Empezar una Hebra - I

```
public class ThreadEx1 extends Thread {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

Definir una clase que extiende de la clase `java.util.Thread`

El método `run` y todo al que llamamos (directamente o indirectamente) desde `run` se va a ejecutar en una hebra separada.

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
  
    t1.start();  
    t2.start();  
}
```

Crear y Empezar una Hebra - I

```
public class ThreadEx1 extends Thread {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

Definir una clase que extiende de la clase `java.util.Thread`

El método `run` y todo al que llamamos (directamente o indirectamente) desde `run` se va a ejecutar en una hebra separada.

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
  
    t1.start();  
    t2.start();  
}
```

see: `examples.threads.ex1.ThreadEx1`

Crear y Empezar una Hebra - I

```
public class ThreadEx1 extends Thread {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
  
    t1.start();  
    t2.start();  
}
```

Definir una clase que extiende de la clase `java.util.Thread`

El método `run` y todo al que llamamos (directamente o indirectamente) desde `run` se va a ejecutar en una hebra separada.

Crear un objeto que corresponde a la hebra (nos permite controlar la hebra)

1. Una llamada a `start` inicializa la hebra y después empieza la ejecución de `run` en esa hebra.
2. Quien llama a `start` sigue ejecutando en paralelo -- **Asynchronous call**

Crear y Empezar una Hebra - I

```
public class ThreadEx1 extends Thread {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
  
    t1.start();  
    t2.start();  
}
```

Definir una clase que extiende de la clase `java.util.Thread`

El método `run` y todo al que llamamos (directamente o indirectamente) desde `run` se va a ejecutar en una hebra separada.

Crear un objeto que corresponde a la hebra (nos permite controlar la hebra)

1. Una llamada a `start` inicializa la hebra y después empieza la ejecución de `run` en esa hebra.
2. Quien llama a `start` sigue ejecutando en paralelo -- **Asynchronous call**

IMPORTANT: distinguir la hebra del objeto correspondiente. El objeto es sólo para controlar la hebra.

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0

Desc: main está ejecutando en esta hebra

State: **RUNNABLE**



Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0



Desc: main está ejecutando en esta hebra

State: RUNNABLE

Thread: #1



Desc: una hebra que corresponde a t1

State: NEW



Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0



Desc: main está ejecutando en esta hebra

State: RUNNABLE

Thread: #1



Desc: una hebra que corresponde a t1

State: NEW

Thread: #2



Desc: una hebra que corresponde a t2

State: NEW



typeTag: ThreadEx1

id : "TEx1"

typeTag: ThreadEx1

id : "TEx2"

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0



Desc: main está ejecutando en esta hebra

State: **RUNNABLE**

Thread: #1



Desc: una hebra que corresponde a t1

State: **NEW** **RUNNABLE**

Thread: #2



Desc: una hebra que corresponde a t2

State: **NEW**



typeTag: ThreadEx1

id : "TEx1"



typeTag: ThreadEx1

id : "TEx2"

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0



Desc: main está ejecutando en esta hebra

State: **RUNNABLE**

Thread: #1



Desc: una hebra que corresponde a t1

State: **NEW** **RUNNABLE**

Thread: #2



Desc: una hebra que corresponde a t2

State: **NEW** **RUNNABLE**



typeTag: ThreadEx1

id : "TEx1"



typeTag: ThreadEx1

id : "TEx2"

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0

Desc: main está ejecutando en esta hebra

State: ~~RUNNABLE~~ TERMINATED



Thread: #1

Desc: una hebra que corresponde a t1

State: ~~NEW~~ RUNNABLE



Thread: #2

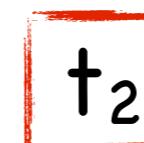
Desc: una hebra que corresponde a t2

State: ~~NEW~~ RUNNABLE



typeTag: ThreadEx1

id : "TEx1"



typeTag: ThreadEx1

id : "TEx2"

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0

Desc: main está ejecutando en esta hebra

State: ~~RUNNABLE~~ TERMINATED



Thread: #1

Desc: una hebra que corresponde a t1

State: ~~NEW~~ ~~RUNNABLE~~ TERMINATED



Thread: #2

Desc: una hebra que corresponde a t2

State: ~~NEW~~ RUNNABLE



typeTag: ThreadEx1

id : "TEx1"



typeTag: ThreadEx1

id : "TEx2"

Crear y Empezar una Hebra - II

```
public class ThreadEx1 extends Thread {  
    private String id;  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new ThreadEx1("TEx1");  
    Thread t2 = new ThreadEx1("TEx2");  
    t1.start();  
    t2.start();  
}
```

Thread: #0

Desc: main está ejecutando en esta hebra

State: ~~RUNNABLE~~ ~~TERMINATED~~



Thread: #1

Desc: una hebra que corresponde a t1

State: ~~NEW~~ ~~RUNNABLE~~ ~~TERMINATED~~



Thread: #2

Desc: una hebra que corresponde a t2

State: ~~NEW~~ ~~RUNNABLE~~ ~~TERMINATED~~



typeTag: ThreadEx1

id : "TEx1"



typeTag: ThreadEx1

id : "TEx2"

Crear y Empezar una Hebra - III

```
public class A extends B implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new Thread( new A("TEx1") );  
    Thread t2 = new Thread( new A("TEx2") );  
  
    t1.start();  
    t2.start();  
}
```

¡Java no tiene herencia múltiple! ¿Cómo podemos usar hebras si la clase ya extiende otra?

Crear y Empezar una Hebra - III

```
public class A extends B implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

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

```
public static void main(String args[]) {  
    Thread t1 = new Thread( new A("TEx1") );  
    Thread t2 = new Thread( new A("TEx2") );  
  
    t1.start();  
    t2.start();  
}
```

¡Java no tiene herencia múltiple! ¿Cómo podemos usar hebras si la clase ya extiende otra?

Crear y Empezar una Hebra - III

```
public class A extends B implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}  
  
public interface Runnable {  
    public abstract void run();  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new Thread( new A("TEx1") );  
    Thread t2 = new Thread( new A("TEx2") );  
  
    t1.start();  
    t2.start();  
}
```

¡Java no tiene herencia múltiple! ¿Cómo podemos usar hebras si la clase ya extiende otra?

```
class Thread implements Runnable {  
    ...  
    private Runnable target;  
    ...  
    public Thread(Runnable target) {  
        this.target = target;  
    }  
    public void run() {  
        if (target != null) {  
            target.run();  
        }  
    }  
    ...  
}
```

Hebras Usando Clases Anónimas

```
public class ThreadEx3 {  
    public static void main(String args[]) {  
        new Thread() {  
            public void run() {  
                int i = 0;  
                while (i < 100) {  
                    System.out.println("T1: " + i);  
                    i++;  
                }  
            }  
        }.start();  
  
        new Thread(new Runnable() {  
            public void run() {  
                int i = 0;  
                while (i < 100) {  
                    System.out.println("T2: " + i);  
                    i++;  
                }  
            }  
        }).start();  
        System.out.println("Main is done!");  
    }  
}
```

see: examples.threads.ex1.ThreadEx3

Hebras Usando Clases Anónimas

```
public class ThreadEx3 {  
    public static void main(String args[]) {  
        new Thread() {  
            public void run() {  
                int i = 0;  
                while (i < 100) {  
                    System.out.println("T1: " + i);  
                    i++;  
                }  
            }  
        }.start();  
  
        new Thread(new Runnable() {  
            public void run() {  
                int i = 0;  
                while (i < 100) {  
                    System.out.println("T2: " + i);  
                    i++;  
                }  
            }  
        }).start();  
  
        System.out.println("Main is done!");  
    }  
}
```

1. Crear una instancia de una clase anónima que extiende **Thread**
2. Llamar a **start** de la clase anónima

Hebras Usando Clases Anónimas

```
public class ThreadEx3 {  
    public static void main(String args[]) {  
        new Thread() {  
            public void run() {  
                int i = 0;  
                while (i < 100) {  
                    System.out.println("T1: " + i);  
                    i++;  
                }  
            }  
        }.start();  
  
        new Thread(new Runnable() {  
            public void run() {  
                int i = 0;  
                while (i < 100) {  
                    System.out.println("T2: " + i);  
                    i++;  
                }  
            }  
        }).start();  
        System.out.println("Main is done!");  
    }  
}
```

1. Crear una instancia de una clase anónima que implementa la interfaz **Runnable** y pásala a una instancia de la clase **Thread**
2. Llamar a **start** de la instancia de **Thread**

¿Cuando un Prog./Hebra Termina?

```
public class TEx1 implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while ( true ) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public class TEx2 implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[]) {  
    Thread t1 = new Thread( new TEx1("A") );  
    Thread t2 = new Thread( new TEx2("B") );  
  
    t1.start();  
    t2.start();  
}
```

- ◆ Una hebra termina cuando el método `run` que corresponde termina ...
- ◆ Un programa termina cuando todas sus hebras terminan ...

¿Cuando un Prog./Hebra Termina?

```
public class TEx1 implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while ( true ) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

No termina

```
public class TEx2 implements Runnable {  
    private String id;  
  
    ThreadEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while ( i<100 ) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

Termina

```
public static void main(String args[]) {  
    Thread t1 = new Thread( new TEx1("A") );  
    Thread t2 = new Thread( new TEx2("B") );  
  
    t1.start();  
    t2.start();  
}  
La hebra de  
main termina  
El programa  
no termina
```

- ◆ Una hebra termina cuando el método `run` que corresponde termina ...
- ◆ Un programa termina cuando todas sus hebras terminan ...

El Estado de Una Hebra

Se puede consultar el estado de una hebra t usando t.getState() -- puede ser uno de los siguientes

| | |
|---------------|--|
| NEW | No se ha empezado todavía |
| RUNNABLE | Ejecutando o esperando al gestor de hebras para que le deje ejecutar. |
| BLOCKED | Bloqueada, esperando para obtener un "monitor lock" |
| WAITING | Esperando un tiempo indefinido , hasta que "ocurra algo" que la desperta |
| TIMED_WAITING | Esperando un tiempo predefinido , o hasta que "ocurra algo" que la desperta |
| TERMINATED | Ya ha terminado ... |

El Estado de Una Hebra

Se puede consultar el estado de una hebra t usando t.getState() -- puede ser uno de los siguientes

| | |
|---------------|--|
| NEW | No se ha empezado todavía |
| RUNNABLE | Ejecutando o esperando al gestor de hebras para que le deje ejecutar. |
| BLOCKED | Bloqueada, esperando para obtener un "monitor lock" |
| WAITING | Esperando un tiempo indefinido , hasta que "ocurra algo" que la desperta |
| TIMED_WAITING | Esperando un tiempo predefinido , o hasta que "ocurra algo" que la desperta |
| TERMINATED | Ya ha terminado ... |

Esperar a que Termine una Hebra

```
public class ThreadEx4 extends Thread {  
    private String id;  
    ThreadEx5(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while (i<100) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
}
```

```
public static void main(String args[])  
    throws InterruptedException {  
  
    Thread t1 = new ThreadEx4("A");  
    Thread t2 = new ThreadEx4("B");  
    t1.start();  
    t2.start();  
    t1.join();  
    t2.join();  
  
    System.out.println("Main is done!");  
}
```

1. La hebra que ejecuta `t.join()` bloquea hasta que la hebra `t` termine.
2. Mientras esperando, su estado sería **WAITING**
3. Se puede interrumpir la hebra bloqueada, en ese caso se lanza una excepción (lo vemos más adelante en detalles)
4. `t.join(n)` espera `n` milisegundos y `t.join(n, m)` espera `n` milisegundos y `m` nanosegundos. Mientras esperando su estado sería **TIMED_WAITING**

Ej.: Calcular una Función en Paralelo

```
public abstract class Function extends Thread {  
  
    protected long result;  
  
    protected abstract void apply() throws InterruptedException;  
  
    public long getRes() {  
        return result;  
    }  
  
    public void run() {  
        try {  
            apply();  
        } catch (InterruptedException e) {  
            System.err.println("...");  
        }  
    }  
}
```

Fact(n), Fib(n), F(n)=fib(n)+fact(n)

```
public class Fact extends Function {  
    private long n;  
  
    Fact(long n) { this.n = n; }  
  
    protected void apply() {  
        long r = 1;  
        for (long i = 1; i <= n; i++) r *= i;  
        this.result = r;  
    }  
}
```

```
public class Fib extends Function {  
    private long n;  
  
    Fib(long n) { this.n = n; }  
  
    protected void apply() {  
        long a = 0, b = 1, i = 0, c;  
        for(int i=1; i < n; i++) {  
            c=a+b; a=b; b=c;  
            i++;  
        }  
        this.result = b;  
    }  
}
```

```
public class F extends Function {  
  
    private long n;  
  
    F(int n) {  
        this.n = n;  
    }  
  
    protected void apply() throws {  
        Function f = new Fact(n);  
        Function g = new Fib(n);  
        f.start();  
        g.start();  
        f.join();  
        g.join();  
        this.result = f.getRes() +  
                     g.getRes();  
    }  
}
```

Ej.: Calcular una Función en Paralelo

```
public class Main {  
  
    public static void main(String[] args) throws InterruptedException {  
  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Introduce an integer: ");  
        long n = sc.nextLong();  
  
        long startTime = System.currentTimeMillis();  
  
        Function t = new F(n);  
        t.start();  
        t.join();  
  
        long taskTimeMs = System.currentTimeMillis() - startTime;  
  
        System.out.println("Res: " + t.getRes());  
        System.out.println("Time: " + taskTimeMs + "ms");  
    }  
}
```

Ej.: Calcular una Función en Paralelo

```
public class Main {  
  
    public static void main(String[] args) throws InterruptedException {  
  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Introduce an integer: ");  
        long n = sc.nextLong();  
  
        long startTime = System.currentTimeMillis();  
  
        Function t = new F(n);  
        t.start();  
        t.join();  
  
        long taskTimeMs = System.currentTimeMillis() - startTime;  
  
        System.out.println("Res: " + t.getRes());  
        System.out.println("Time: " + taskTimeMs + "ms");  
    }  
}
```

Ej.: Calcular una Función en Paralelo

```
public class Main {  
  
    public static void main(String[] args) throws InterruptedException {  
  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Introduce an integer: ");  
        long n = sc.nextLong();  
  
        long startTime = System.currentTimeMillis();  
  
        Function t = new F(n);  
        t.start();  
        t.join();  
  
        long taskTimeMs = System.currentTimeMillis() - startTime;  
  
        System.out.println("Res: " + t.getRes());  
        System.out.println("Time: " + taskTimeMs + "ms");  
    }  
}
```

Ej.: Calcular una Función en Paralelo

```
public class Main {  
  
    public static void main(String[] args) throws InterruptedException {  
  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Introduce an integer: ");  
        long n = sc.nextLong();  
  
        long startTime = System.currentTimeMillis();  
  
        Function t = new F(n);  
        t.start();  
        t.join();  
  
        long taskTimeMs = System.currentTimeMillis() - startTime;  
  
        System.out.println("Res: " + t.getRes());  
        System.out.println("Time: " + taskTimeMs + "ms");  
    }  
}
```

Medir el tiempo de ejecución

Gestión de Hebras

- ◆ `sleep(int m)`, `sleep(int m, int n)`: La hebra "duerme" m milisegundos (y n nanosegundos) -- el gestor de hebras no la programa para ejecutar durante ese tiempo. Su estado sería **TIMED_WAITING**
- ◆ Cada hebra tiene prioridad asignada. El gestor de hebra usa esas prioridades para elegir la siguiente hebra a ejecutar.
 1. `t.getPriority()` devuelve la prioridad de `t`
 2. `t.setPriority(int p)` cambia la prioridad de `t` a `p`
 3. `MIN_PRIORITY` ($=1$), `MAX_PRIORITY` ($=10$), `NORM_PRIORITY` ($=5$)

Gestión de Hebras

- ◆ `Thread.yield()` avisa al gestor de hebras que la hebra actual (el que llama a `yield`) quiere ceder el CPU (es decir, dejar a otras hebras ejecutar) -- el gestor de hebras puede ignorar la petición.
- ◆ Hay más instrucciones que afectan a la gestión de hebras que vamos a ver más adelante ...

Interrumpir a una Hebra

- ◆ Es un mecanismo de comunicación entre hebras ...
- ◆ Una hebra puede interrumpir a otra, se puede ver como una notificación ...
- ◆ `t.interrupt()`: interrumpe la hebra `t` -- el efecto depende del estado de `t` (lo vemos con ejemplos en enseguida). Si el estado de `t` es `RUNNABLE`, el estado de interrupción de `t` cambia a `true`.
- ◆ `Thread.interrupted()`, devuelve `true` si la hebra que invoca `Thread.interrupted()` ha sido interrumpida, y cambia el estado de interrupción a `false`.
- ◆ `t.isInturrrpted()` devuelve `true` si la hebra `t` ha sido interrumpida. En este caso no cambia el estado de interrupción de `t`.

Interrumpir a una Hebra - Ej. I

```
public class IntEx1 implements Runnable {  
  
    private String id;  
  
    IntEx1(String id) {  
        this.id = id;  
    }  
  
    public void run() {  
        int i=0;  
        while ( !Thread.interrupted() ) {  
            System.out.println(id+": "+i);  
            i++;  
        }  
    }  
  
    public static void main(String args[]) throws InterruptedException {  
        Thread t1 = new Thread( new IntEx1("TEx1") );  
        t1.start();  
        Thread.sleep(5000);  
        t1.interrupt();  
    }  
}
```

Interrumpir a una Hebra - Ej. II

```
public class IntEx2 implements Runnable {  
    private String id;  
    IntEx2(String id) {  
        this.id = id;  
    }  
    public void run() {  
        int i=0;  
        while ( !Thread.interrupted() ) {  
            System.out.println(id+": "+i);  
            try {  
                Thread.sleep(3000);  
            } catch (InterruptedException e) {  
                System.out.println("Int while sleeping");  
                Thread.currentThread().interrupt();  
            }  
        }  
    }  
    public static void main(String args[]) throws InterruptedException {  
        Thread t1 = new Thread( new IntEx2("TEx1") );  
        t1.start();  
        Thread.sleep(5000);  
        t1.interrupt();  
    }  
}
```

Si interrumpimos a una hebra mientras su estado es WAITING o TIMED_WAITING se lanza una excepción. En este caso el estado de interrupción no cambia

Interrumpe a sí misma para poder salir de bucle. Se puede conseguir lo mismo con una variable boolean.

Interrumpir a una Hebra - Ej. III

```
public class IntEx3 implements Runnable {  
    public void run() {  
        Thread t2 = new Thread() {  
            public void run() {  
                try { sleep(8000); } catch (InterruptedException e) {}  
                System.out.println("T2 is done!");  
            }  
        };  
        t2.start();  
        try {  
            t2.join();  
        } catch (InterruptedException e) {  
            System.out.println("T1 was interrupted!");  
        }  
        System.out.println("T1 is done!");  
    }  
  
    public static void main(String args[]) throws InterruptedException {  
        Thread t1 = new Thread(new IntEx3());  
        t1.start();  
        Thread.sleep(5000);  
        t1.interrupt();  
        System.out.println("Main is done!");  
    }  
}
```

El Efecto de t.interrupt()

| | |
|---------------|---|
| NEW | No pasa nada |
| RUNNABLE | Cambia el estado de interrupción a true |
| BLOCKED | Cambia el estado de interrupción a true |
| WAITING | Lanza una excepción |
| TIMED_WAITING | Lanza una excepción |
| TERMINATED | No pasa nada |