

# Desarrollo de Software I

## Hilos

**Ing. Eric Gustavo Coronel Castillo**

[gcoronelc@gmail.com](mailto:gcoronelc@gmail.com)

[gcoronelc.blogspot.com](http://gcoronelc.blogspot.com)

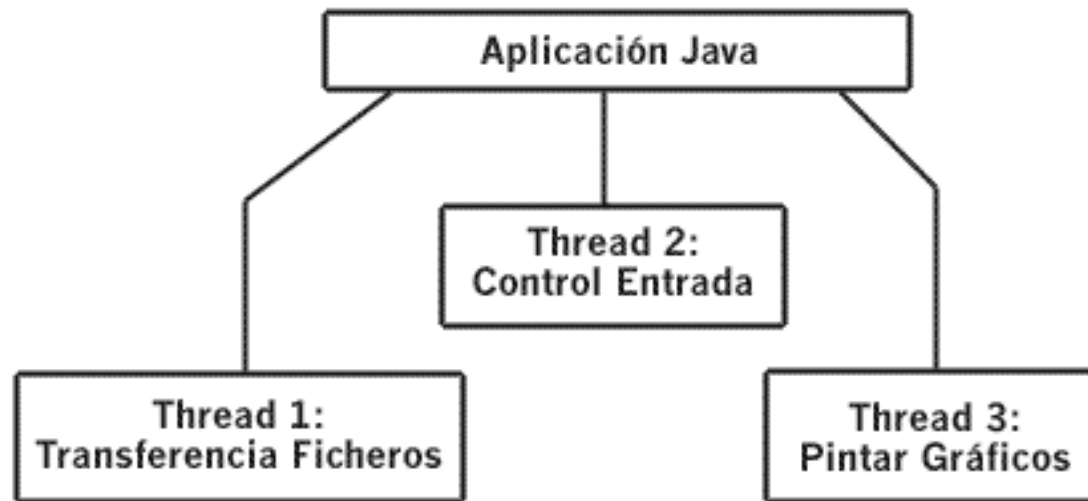
# Índice

---

- ❖ Introducción
- ❖ ¿Qué son los Hilos?
- ❖ Implementación de Hilos
- ❖ Sincronización de Hilos

# Introducción

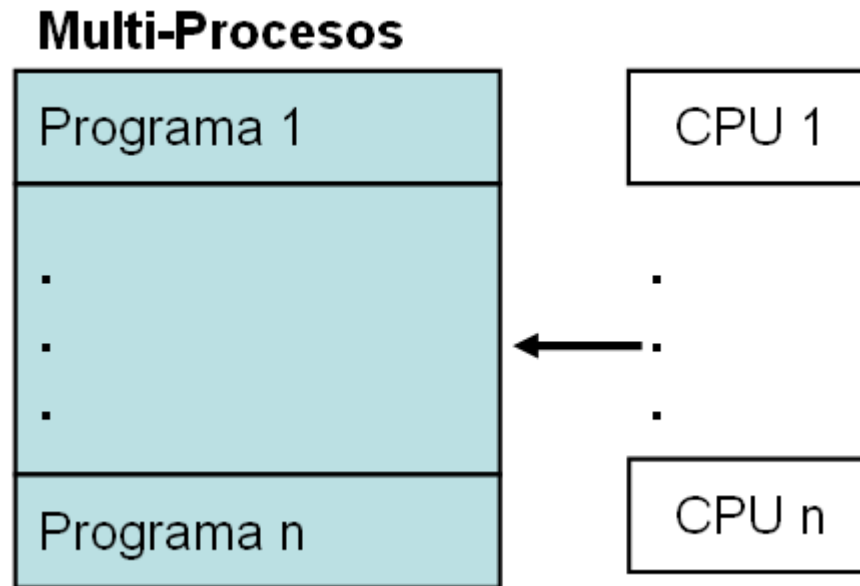
- ❖ Muchas veces necesitamos ejecutar una secuencia de instrucciones, pero mientras se ejecuta, queremos hacer otras tareas. Esto se logra programando hilos (Threads).



# ¿Qué son los Hilos?

- ❖ La Máquina Virtual Java (JVM) es un sistema multihilo. Es decir, es capaz de ejecutar varias secuencias de instrucciones (rutinas) simultáneamente. La JVM gestiona todos los detalles, asignación de tiempos de ejecución, prioridades, etc, de forma similar, como un Sistema Operativo gestiona múltiples procesos (programas).
- ❖ La diferencia básica entre un proceso de Sistema Operativo y un Hilo Java es que los Hilos corren dentro de la JVM, que es un proceso del Sistema Operativo y por tanto comparten todos los recursos, incluida la memoria, las variables y objetos allí definidos. A este tipo de procesos donde se comparte los recursos se les llama a veces "procesos ligeros" (lightweight process).

# ¿Qué son los Hilos?

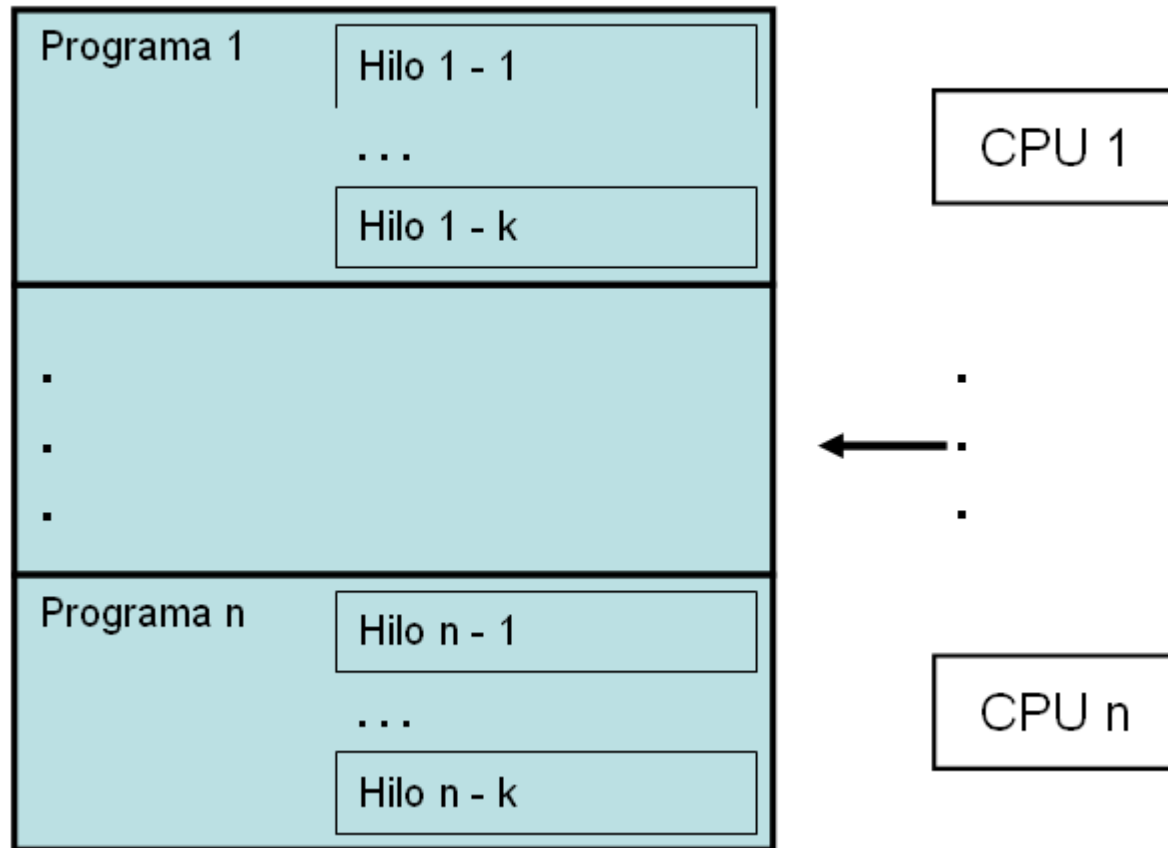


# ¿Qué son los Hilos?

- ❖ Java implementa el concepto de Hilo desde el mismo lenguaje, con algunas clases e interfaces definidas en el paquete **java.lang** y con métodos específicos para la manipulación de Hilos en la clase **Object**.
- ❖ En un sistema multihilo, el hilo es la mínima unidad que se puede ejecutar. Esto significa que un único programa puede ejecutar dos o más tareas en forma simultanea. Por ejemplo, una hoja de cálculo puede estar realizando un cálculo al mismo tiempo que esta imprimiendo, ya que estas dos acciones se realizan por hilos diferentes. Por lo tanto, un sistema multiprocesos trata de gestiones generales, y un sistema multihilos trata de los detalles.

# ¿Qué son los Hilos?

## Multi-Hilos



# Implementación de Hilos

## La Clase Thread

Extendiendo la clase **Thread** podemos implementar un hilo.

El esquema es el siguiente:

```
1. public class nombre_clase extends Thread {  
2.     // Constructor  
3.     public nombre_clase( . . . ){  
4.         // Implementación  
5.     }  
6.     public void run() {  
7.         // Implementación  
8.     }  
9.     . . .  
10. }
```



# Implementación de Hilos

## La Clase Thread

Para ejecutar el hilo se debe invocar al método **start()**.

```
nombre_clase obj;  
obj = new nombre_clase( ... );  
obj.start();
```

# Implementación de Hilos

## Ejemplo 01

```
1. public class TablaMultiplicar extends Thread {
2.     // Campos
3.     private int n;
4.
5.     // Constructor
6.     public TablaMultiplicar(String msg, int n) {
7.         super(msg);
8.         this.n = n;
9.     }
10.    // Código del hilo
11.    public void run() {
12.        String tabla = "\nNombre: " + this.getName() + "\n";
13.        int r;
14.        for (int k = 1; k <= 12; k++) {
15.            r = k * n;
16.            tabla = tabla + "\n" + k + " * " + n + " = " + r;
17.        }
18.        System.out.println(tabla);
19.    }
20.
21.}
```

# Implementación de Hilos

## Ejemplo 01

```
1. public class Programa {  
2.     public static void main(String[] args) {  
3.         // Creación de los Objetos  
4.         TablaMultiplicar t1 = new TablaMultiplicar("Tabla del 5", 5);  
5.         TablaMultiplicar t2 = new TablaMultiplicar("Tabla del 9", 9);  
6.         TablaMultiplicar t3 = new TablaMultiplicar("Tabla del 11", 11);  
7.         // Inicio  
8.         System.out.println("Inicio");  
9.         // Lanzar los Hilos  
10.        t1.start();  
11.        t2.start();  
12.        t3.start();  
13.        // Fin  
14.        System.out.println("Fin");  
15.    }  
16.  
17. }
```

# Implementación de Hilos

## La Interfaz Runnable

- Otra forma de implementar un hilo es crear un clase que implemente la interfaz `Runnable`.
- En cualquier clase que implemente esta interfaz podemos construir un hilo. Para implementar esta interfaz solo debemos implementar un método de nombre `run()`.
- El esquema de la implementación del método es el siguiente:

```
public void run() {  
  
    // implementacion  
  
}
```

# Implementación de Hilos

## La Interfaz Runnable

- Para ejecutar el hilo debemos crear un objeto de tipo `Thread`, dentro de esta clase se definen muchos constructores, pero el que debemos utilizar es:

```
public Thread( Runnable target, String name )
```

- **target** es una instancia de la clase que implementa la interfaz `Runnable`. Esto define donde iniciará el hilo.
- Para iniciar el nuevo hilo debemos ejecutar el método `start()`, de la clase `Thread`.
- El método `start()` invoca el método `run()` implementado en la clase que implementa la interfaz `Runnable`.

# Implementación de Hilos

## La Interfaz Runnable

- El esquema es el siguiente:

```
1. public NombreClase implements Runnable {  
2.     // Constructor  
3.     public NombreClase(){  
4.         // Implementación  
5.     }  
6.     // Código del hilo  
7.     public void run(){  
8.         // Implementación  
9.     }  
10. }
```

# Implementación de Hilos

## Ejemplo 02

```
1.  public class SimpleRunnable implements Runnable{
2.      private int n;
3.      private String nombre;
4.
5.      public SimpleRunnable(String nombre, int n) {
6.          this.nombre = nombre;
7.          this.n = n;
8.      }
9.
10.     public void run() {
11.
12.         Thread t = Thread.currentThread();
13.         t.setName(nombre);
14.
15.         System.out.println("Ejecutando el hilo: " + nombre + "\n");
16.         int suma = 0;
17.         for (int i = 1; i <= n; i++) {
18.             suma += i;
19.         }
20.
21.         String rpt = "La suma de los " + n + " primeros numeros es " + suma;
22.         System.out.println(rpt);
23.     }
24. } // SimpleRunnable
```

# Implementación de Hilos

## Ejemplo 02

```
1.  public class Programa {
2.      public static void main(String[] args) {
3.          System.out.println("Inicio");
4.          SimpleRunnable obj = new SimpleRunnable("Suma de Numeros", 25);
5.          Thread hilo = new Thread(obj); // Creación del hilo
6.          hilo.start(); // Lanzar el hilo
7.          // Espera a que finalice el hilo
8.          try {
9.              hilo.sleep(1000);           // Suspende la ejecución del hilo
10.             hilo.join();                // Espera a que finalice el hilo
11.         } catch (InterruptedException e) {
12.             System.out.println("Hilo " + hilo.getName() + " interrumpido.");
13.         }
14.         System.out.println("Fin");
15.     }
16. } // Programa
```



# Implementación de Hilos

## Hilo Principal

- Cuando un programa Java inicia su ejecución, uno de los hilos se ejecuta de forma inmediata. A este hilo se le suele llamar hilo principal del programa, por que es el único que se ejecuta al comenzar el programa. El hilo principal es importante por dos motivos:
  - Es el hilo desde el que crea el resto de los hilos del programa.
  - A menudo debe ser el último que finaliza su ejecución. Cuando el hilo principal finaliza, el programa termina.
- Aunque el hilo principal se crea automáticamente cuando el programa se inicia, se puede controlar a través del objeto **Thread**. Para ello se debe obtener una referencia a este objeto llamando al método **currentThread()**, que es un miembro **public static** de **Thread**. Su prototipo es:

```
public static Thread currentThread()
```

- Este método retorna una referencia al hilo desde donde se le invoco. Una vez que se tiene una referencia a un hilo principal, se puede controlar del mismo modo que a otro hilo.

# Implementación de Hilos

## Ejemplo 03

```
1. public class Programa {  
2.     public static void main(String[] args) {  
3.         Thread t = Thread.currentThread(); // Obtener referencia al hilo actual  
4.         System.out.println("Hilo Actual: " + t);  
5.         t.setName("Super Hilo"); // Cambiando de nombre al hilo  
6.         System.out.println("Hilo Actual: " + t);  
7.         try {  
8.             for (int n = 5; n > 0; n--) {  
9.                 System.out.println(n + "\n");  
10.                t.sleep(1000);  
11.            }  
12.        } catch (InterruptedException e) {  
13.            System.out.println("Super Hilo - Detenido");  
14.        }  
15.    }  
16.  
17. } // Programa
```

# Sincronización de Hilos

## Introducción

- ❖ Cuando dos ó mas hilos necesitan acceder a un recurso compartido, es necesario asegurar de alguna forma que accedan a dicho recurso de uno en uno. El proceso por el que se consigue esto se llama sincronización.
- ❖ La clave de la sincronización es el concepto de monitor, también llamado semáforo. Un monitor es un objeto que se utiliza como cerrojo exclusivo. Solo uno de los hilos puede tener un monitor en un momento determinado. Cuando un hilo adquiere un cerrojo, se dice que ha entrado en el monitor. Todos los demás hilos que intenten acceder al monitor bloqueado serán suspendidos hasta que el primero salga del monitor; se dice que los otros hilos están esperando al monitor. Un hilo que posee un monitor puede volver a entrar en el mismo monitor si así lo requiere.

# Sincronización de Hilos

## Ejemplo 04

```
1.  public class Message {  
2.      public void show(String msg) {  
3.          Thread t;  
4.          t = Thread.currentThread();  
5.          try {  
6.              System.out.print("::");  
7.              System.out.print(msg);  
8.              t.sleep(1000);  
9.              System.out.println("::");  
10.         } catch (InterruptedException e) {  
11.             System.out.println("Hilo " + t.getName() + " Interrumpido");  
12.         }  
13.     }  
14.  
15. } // Message
```

# Sincronización de Hilos

## Ejemplo 04

```
1.  public class MostrarMensaje implements Runnable {  
2.      private String msg;  
3.      private Message obj;  
4.      public Thread hilo;  
  
5.      public MostrarMensaje(Message obj, String msg, String titulo) {  
6.          this.obj = obj;  
7.          this.msg = msg;  
8.          hilo = new Thread(this);  
9.          hilo.setName(titulo);  
10.         hilo.start();  
  
11.     }  
  
12.     public void run() {  
13.         obj.show(msg);  
14.     }  
15.  
16. } // Mostrar Mensaje
```

# Sincronización de Hilos

## Ejemplo 04

```
1.  public class Programa {
2.      public static void main(String[] args) {
3.          System.out.println("Inicio\n"); // Inicio
4.          Message objMsg = new Message();
5.          MostrarMensaje obj1 = new MostrarMensaje(objMsg, "Java es el Mejor", "Hilo 1");
6.          MostrarMensaje obj2 = new MostrarMensaje(objMsg, "JSP las mas Seguras", "Hilo 2");
7.          MostrarMensaje obj3 = new MostrarMensaje(objMsg, "Java es Garantia", "Hilo 3");
8.          try {
9.              obj1.hilo.join();
10.             obj2.hilo.join();
11.             obj3.hilo.join();
12.         } catch (InterruptedException e) {
13.             System.out.println("Programa Interrumpido");
14.         }
15.         System.out.println("\nFin");
16.     } // main
17.
18. } // Programa
```

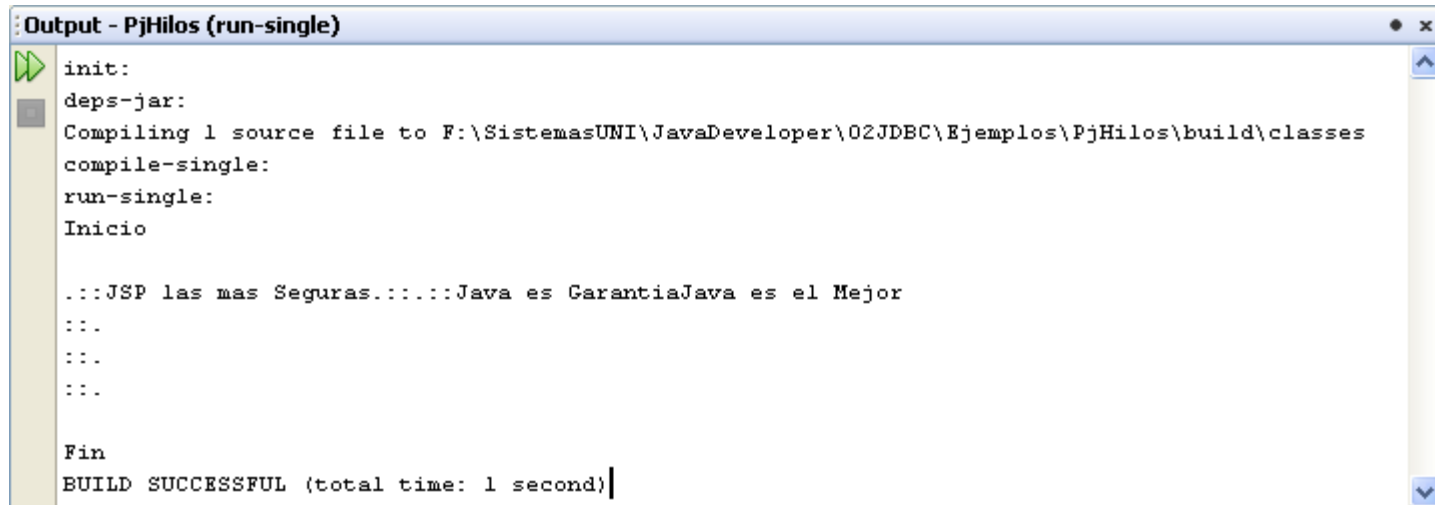
# Sincronización de Hilos

## Ejemplo 04

En el programa de este ejemplo se crean tres hilos, y se espera a que los hilos terminen para que finalice el hilo principal. Se espera el siguiente resultado:

```
...: Java es el Mejor::.  
...: JSP las mas Seguras::.  
...: Java es Garantia::.
```

Pero, debido a que no está sincronizado el acceso al método **show** de la clase **Message** obtenemos el siguiente resultado:



```
Output - PjHilos (run-single)  
init:  
deps-jar:  
Compiling 1 source file to F:\SistemasUNI\JavaDeveloper\O2JDBC\Ejemplos\PjHilos\build\classes  
compile-single:  
run-single:  
Inicio  
  
...:JSP las mas Seguras::...:Java es GarantiaJava es el Mejor  
...:  
...:  
...:  
  
Fin  
BUILD SUCCESSFUL (total time: 1 second)
```

# Sincronización de Hilos

## Ejemplo 04

- ❖ Como se puede ver, al llamar al método `sleep()`, el método `show()` permite que la ejecución se conmute a otro hilo. El resultado es una salida en la que se mezclan tres mensajes. En este programa no hay nada que impida que los tres hilos llamen al mismo método, del mismo objeto y al mismo tiempo.
- ❖ Esto es lo que se conoce como **condición de carrera**, por que los tres hilos compiten para completar el método. Este método utiliza `sleep()` para que los efectos sean notorios y obvios.
- ❖ En la mayoría de las situaciones, la condición de carrera es más sutil y menos predecible, ya que no se puede estar seguro de cuando se produce el intercambio de contexto. Además, esto puede hacer que un programa se ejecute correctamente algunas veces e incorrectamente otras.
- ❖ Para corregir este problema, se debe hacer un acceso en serie a `show()`. Es decir, se debe restringir el acceso a solo un hilo a la vez. Esto se puede conseguir de dos maneras:
  1. Construyendo métodos sincronizados.
  2. Utilizando la instrucción de sincronización



# Sincronización de Hilos

## Palabra Clave synchronized

La palabra clave synchronized asegura que solo un hilo ejecute un método y a la vez este método es bloqueado.

La forma de utilizar esta palabra clave, es en la declaración de los métodos, su sintaxis es la siguiente:

```
[acceso] synchronized tipo nombre_método ( parámetros ) {  
  
    // Implementación  
  
}
```

# Sincronización de Hilos

## Ejemplo 05

```
1. public class Message {  
2.     public synchronized void show(String msg) {  
3.         System.out.print("::");  
4.         System.out.print(msg);  
5.         System.out.println("::.");  
6.     }  
7.  
8. } // Message
```

# Sincronización de Hilos

## Ejemplo 05

```
1.  public class MostrarMensaje implements Runnable {
2.
3.      private String msg;
4.      private Message obj;
5.      public Thread hilo;
6.
7.      public MostrarMensaje(Message obj, String msg, String titulo) {
8.
9.          this.obj = obj;
10.         this.msg = msg;
11.         hilo = new Thread(this);
12.         hilo.setName(titulo);
13.         hilo.start();
14.
15.     }
16.
17.     public void run() {
18.         obj.show(msg);
19.     }
20. } // MostrarMensaje
```

# Sincronización de Hilos

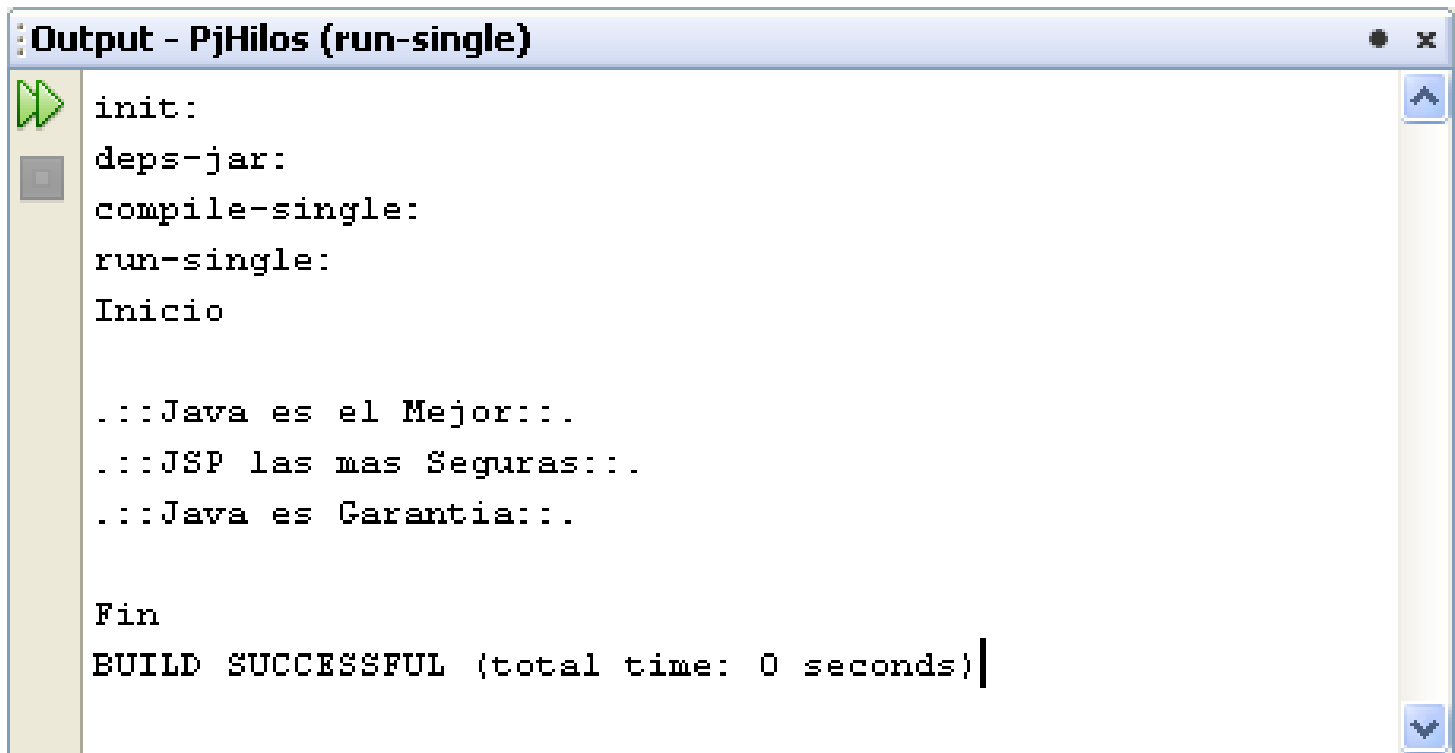
## Ejemplo 05

```
1.  public class Programa {
2.      public static void main(String[] args) {
3.          System.out.println("Inicio\n");
4.          Message objMsg = new Message();
5.          MostrarMensaje obj1 = new MostrarMensaje(objMsg, "Java es el Mejor", "Hilo 1");
6.          MostrarMensaje obj2 = new MostrarMensaje(objMsg, "JSP las mas Seguras", "Hilo 2");
7.          MostrarMensaje obj3 = new MostrarMensaje(objMsg, "Java es Garantia", "Hilo 3");
8.          try {
9.              obj1.hilo.join();
10.             obj2.hilo.join();
11.             obj3.hilo.join();
12.         } catch (InterruptedException e) {
13.             System.out.println("Programa Interrumpido");
14.         }
15.         System.out.println("\nFin");
16.     } // main
17. } // Programa
```

# Sincronización de Hilos

## Ejemplo 05

La ejecución de este programa sincroniza la ejecución de método `show` de la clase `Message`. El resultado es:



```
Output - PjHilos (run-single)
init:
deps-jar:
compile-single:
run-single:
Inicio

.:Java es el Mejor:.
.:JSP las mas Seguras:.
.:Java es Garantia:.

Fin
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Sincronización de Hilos

## Instrucción `synchronized`

Aunque la creación de métodos sincronizados dentro de clases creadas por el propio programador es un método sencillo y eficaz para lograr la sincronización, lamentablemente no funciona con todas las clases. Para comprender por que, analicemos los siguientes dos casos:

1. Se quiere sincronizar el acceso a objetos de una clase que no fue diseñada para acceso multihilo, es decir, esta clase no utiliza métodos sincronizados.
2. Se quiere sincronizar el acceso a objetos de clases de terceros, de las cuales no tenemos el código fuente, por lo tanto, no podemos modificar los métodos.

Afortunadamente tenemos la instrucción `synchronized` para resolver estos problemas. La instrucción `synchronized` se usa para indicar que ciertas partes del código (un método), están sincronizadas, es decir, que solamente un hilo puede acceder a dicho método a la vez.

# Sincronización de Hilos

## Introducción synchronized

Cada método sincronizado posee una especie de llave que puede cerrar o abrir la puerta de acceso. Cuando el hilo intenta acceder al método sincronizado, primero verificará si la llave esta cerrada, en cuyo caso no podrá accederlo. Si el método no tiene cerrada la llave entonces el hilo puede acceder a dicho método sincronizado.

La forma de utilizar a esta instrucción es la siguiente:

```
synchronized( objeto ) {  
  
    // Instrucciones  
  
}
```

En este formato, **objeto** es una referencia al objeto que se quiere sincronizar. Un bloque sincronizado asegura que solo se producirá una llamada al método miembro de objeto después que el hilo actual haya iniciado.

# Sincronización de Hilos

## Ejemplo 06

```
1. public class Message {  
2.     public void show(String msg) {  
3.         System.out.print("::");  
4.         System.out.print(msg);  
5.         System.out.println("::");  
6.     } // show  
7.  
8. } // Message
```



# Sincronización de Hilos

## Ejemplo 06

```
1.  public class MostrarMensaje implements Runnable {
2.
3.      private String msg;
4.      private Message obj;
5.      public Thread hilo;
6.
7.      public MostrarMensaje(Message obj, String msg, String titulo) {
8.          this.obj = obj;
9.          this.msg = msg;
10.         hilo = new Thread(this);
11.         hilo.setName(titulo);
12.         hilo.start();
13.     }
14.
15.     public void run() {
16.         synchronized(obj) {
17.             obj.show(msg);
18.         }
19.     }
20. } // MostrarMensaje
```

# Sincronización de Hilos

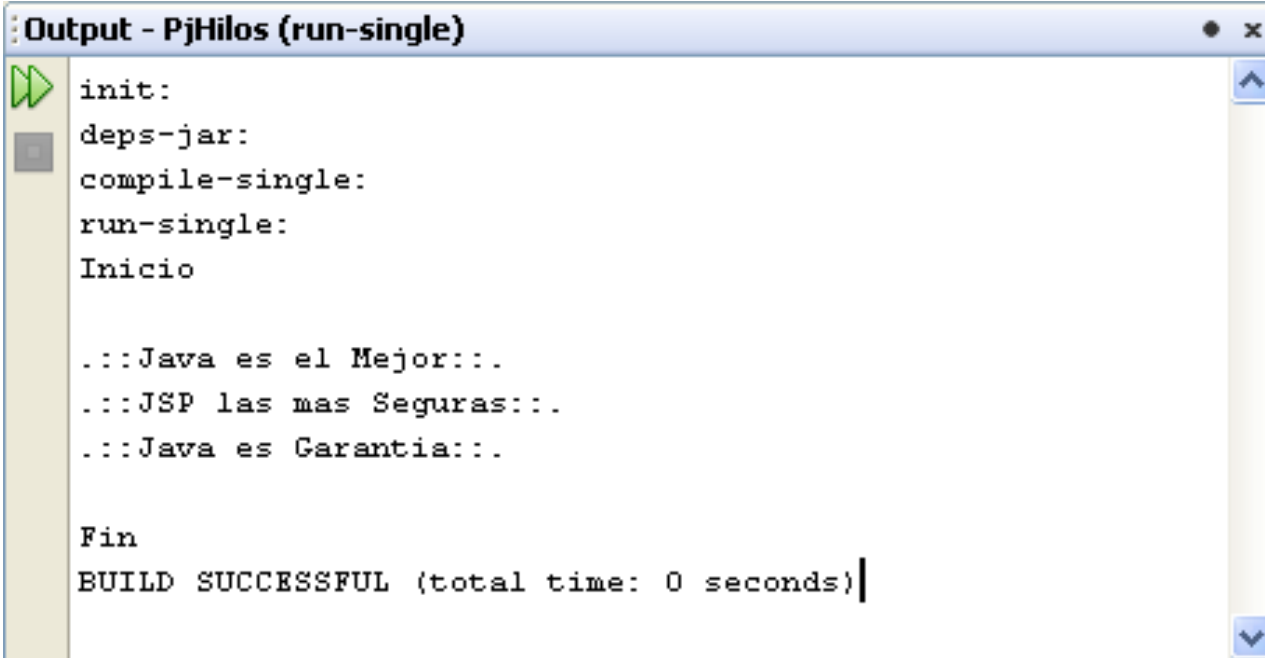
## Ejemplo 06

```
1. public class Programa {  
2.     public static void main(String[] args) {  
3.         System.out.println("Inicio\n");  
4.         Message objMsg = new Message();  
5.         MostrarMensaje obj1 = new MostrarMensaje(objMsg, "Java es el Mejor", "Hilo 1");  
6.         MostrarMensaje obj2 = new MostrarMensaje(objMsg, "JSP las mas Seguras", "Hilo 2");  
7.         MostrarMensaje obj3 = new MostrarMensaje(objMsg, "Java es Garantia", "Hilo 3");  
8.         try {  
9.             obj1.hilo.join();  
10.            obj2.hilo.join();  
11.            obj3.hilo.join();  
12.        } catch (InterruptedException e) {  
13.            System.out.println("Programa Interrumpido");  
14.        }  
15.        System.out.println("\nFin");  
16.    } // main  
17. } // Programa
```

# Sincronización de Hilos

## Ejemplo 06

La ejecución de este programa sincroniza la ejecución de método `show` de la clase `Message`. El resultado es:



```
Output - PjHilos (run-single)
init:
deps-jar:
compile-single:
run-single:
Inicio

...Java es el Mejor...
...JSP las mas Seguras...
...Java es Garantia...

Fin
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Bibliografía

- ❖ Lenguaje de Programación Java 2 Versión 5  
Eric Gustavo Coronel Castillo
- ❖ Desarrollando Soluciones con Java y MySQL Server  
Eric Gustavo Coronel Castillo
- ❖ Java 2 Características Avanzadas - Volumen II  
Cay S. Horstmann y Gary Cornell
- ❖ Piensa en Java  
Bruce Eckel
- ❖ Java Programming Language - Student Guide  
Sun Microsystems
- ❖ Como Programar en Java  
Deitel y Deitel
- ❖ Java 2  
Steven Holzner
- ❖ Aprende Java como si estuviera en primero  
Javier García de Jalón et al