



ESCUELA SUPERIOR DE COMPUTO



PROGRAMACIÓN ORIENTADA A OBJETOS

TAREA

HILOS

Comprender, utilizar y aplicar correctamente el uso de los
Hilos y la programación multihilo en el lenguaje de
programación Java

Rubio Haro Rodrigo R.

CDMX. JUNIO, 2020.

INSTITUTO POLITÉCNICO NACIONAL

Tarea: Hilos

1. Introducción

A diferencia de otros lenguajes Orientados a Objetos, Java permite crear hilos para trabajar de forma concurrente y hasta la versión 1.7 esto es **real**, sin embargo, en la versión 1.8 de Java, se incluyó una librería que puede hacer **programación paralela**. Aquí las diferencias.

Programación concurrente: Es la rama de la informática que trabaja con las técnicas de programación que se usan para expresar el paralelismo entre tareas y para resolver los problemas de comunicación y sincronización entre procesos.

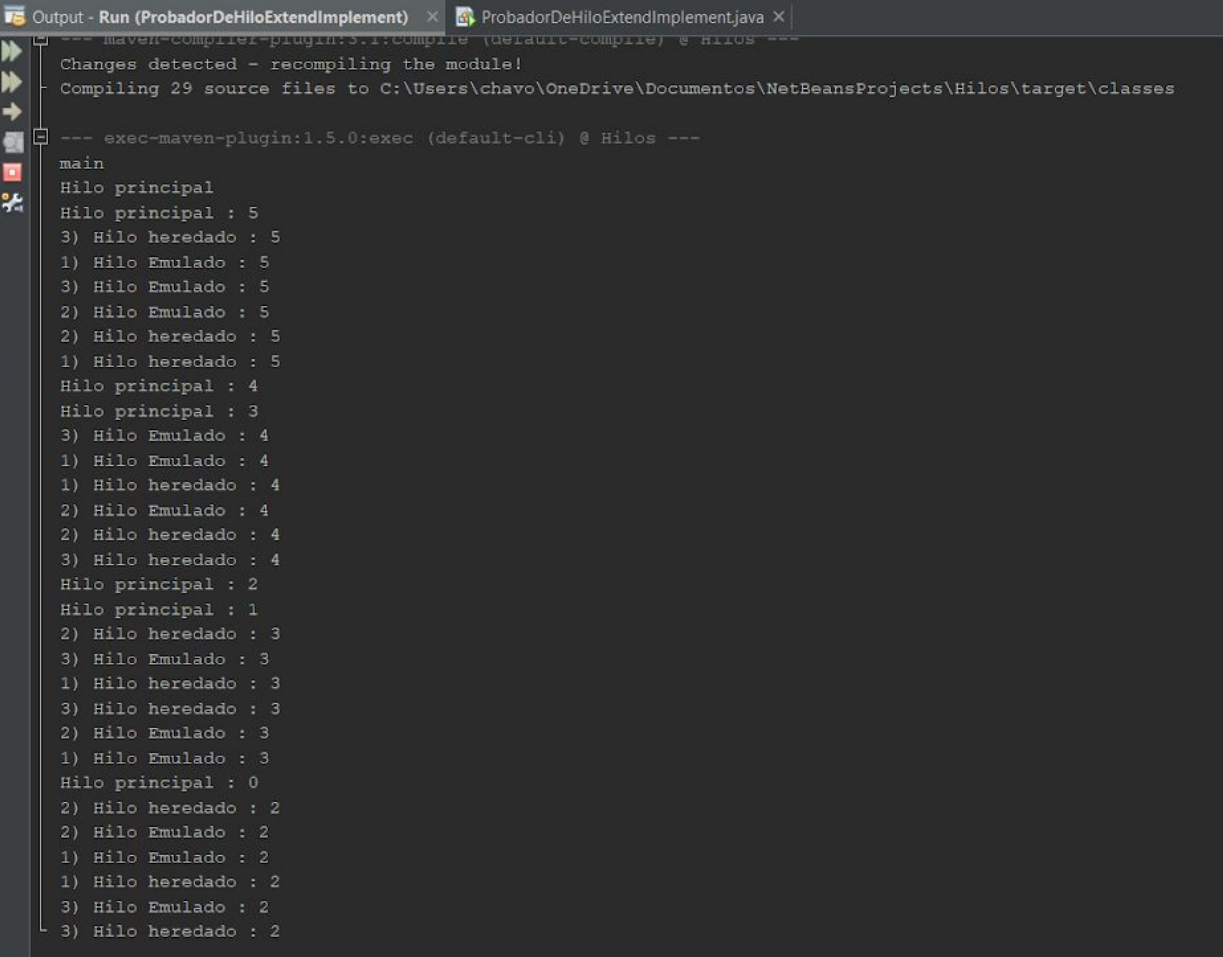
Programación paralela: Es el uso simultáneo de los múltiples recursos de hardware y software para resolver un problema computacional.

Solo se verán los hilos bajo el paradigma de la programación concurrente, en un entorno multitarea basada en hilos, el hilo es la unidad de código más pequeña con que se puede trabajar. Es decir, un programa simple puede realizar dos o más tareas a la vez. Por ejemplo, en un editor de texto se puede estar dando formato al texto a la vez que imprimiendo, ya que estas dos acciones las realizan dos hilos distintos. La multitarea basada en procesos trata con temas generales, y la multitarea basada en hilos gestiona los detalles.

2. Desarrollo

2.1 Creación de hilos extendiendo e implementando.

Se capturó, ejecutó y modificó¹, ligeramente, el Código 3.4.1 presentando, finalmente la siguiente salida.



```

--- maven-compiler-plugin:3.1:compile (default-compile) @ Hilos ---
Changes detected - recompiling the module!
Compiling 29 source files to C:\Users\chavo\OneDrive\Documentos\NetBeansProjects\Hilos\target\classes

--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---
main
Hilo principal
Hilo principal : 5
3) Hilo heredado : 5
1) Hilo Emulado : 5
3) Hilo Emulado : 5
2) Hilo Emulado : 5
2) Hilo heredado : 5
1) Hilo heredado : 5
Hilo principal : 4
Hilo principal : 3
3) Hilo Emulado : 4
1) Hilo Emulado : 4
1) Hilo heredado : 4
2) Hilo Emulado : 4
2) Hilo heredado : 4
3) Hilo heredado : 4
Hilo principal : 2
Hilo principal : 1
2) Hilo heredado : 3
3) Hilo Emulado : 3
1) Hilo heredado : 3
3) Hilo heredado : 3
2) Hilo Emulado : 3
1) Hilo Emulado : 3
Hilo principal : 0
2) Hilo heredado : 2
2) Hilo Emulado : 2
1) Hilo Emulado : 2
1) Hilo heredado : 2
3) Hilo Emulado : 2
3) Hilo heredado : 2

```

¹ Todas las modificaciones habrán sido enviadas en el archivo adjunto

2.2 Creación de múltiples hilos

Se capturó, ejecutó y modificó, ligeramente, el Código 3.4.2 presentando, finalmente la siguiente salida.

```
Nuevo hilo : Thread[Uno,5,main]
Nuevo hilo : Thread[Dos,5,main]
Nuevo hilo : Thread[Tres,5,main]
Uno : 5
Dos : 5
Tres : 5
Dos : 4
Tres : 4
Uno : 4
Dos : 3
Tres : 3
Uno : 3
Dos : 2
Uno : 2
Tres : 2
Dos : 1
Tres : 1
Uno : 1
Sale del hilo : Dos
Sale del hilo : Uno
Sale del hilo : Tres
Sale del hilo principal.
```

2.3 Esperando la ejecución entre hilos utilizando isAlive() y join()

Se capturó, ejecutó y modificó, ligeramente, el Código 3.4.3 presentando, finalmente la siguiente salida.

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---
Nuevo hilo : Thread[Uno,5,main]
Nuevo hilo : Thread[Dos,5,main]
Nuevo hilo : Thread[Tres,5,main]
El hilo Uno esta vivo : true
El hilo Dos esta vivo : true
El hilo Tres esta vivo : true
Espera la finalizacion de los otros hilos.,,
Dos : 5 Tres : 5 Uno : 5 Tres : 4 Dos : 4 Uno : 4 Tres : 3 Uno : 3 Dos : 3 Dos : 2 Uno : 2 Tres : 2 Tres : 1 Uno : 1 Dos : 1

Sale del hilo : Tres
Sale del hilo : Dos
Sale del hilo : Uno
El hilo Uno esta vivo : false
El hilo Dos esta vivo : false
El hilo Tres esta vivo : false
Sale del hilo principal.
-----
BUILD SUCCESS
-----
```

2.4 Cambiando la prioridad de los hilos

Se capturó, ejecutó y modificó, ligeramente, el Código 3.4.4 presentando, finalmente la siguiente salida.

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---
Programa iniciado
```

Lo anterior era un hecho para los viejos sistemas operativos de Windows en su versión XP, Vista y Windows 7, sin embargo ya no es posible cambiar de prioridad a un hilo, el planificador de java aunque reciba hilos de distinta prioridad, el sistema operativo los tratará con la misma prioridad, es decir con NORM_PRIORITY = 5.

El programa efectivamente no pasó de ese mensaje debido a lo recién mencionado, haciendo uso de Windows 10.

2.5 Sincronización de hilos

Se capturó, ejecutó y modificó, ligeramente, el Código 3.4.5, conforme al código 2.3.6 presentando, mostrando finalmente la siguiente salida.

```
[Hola]
[Mundo]
[Sincronizado]
-----
BUILD SUCCESS
-----
Total time: 6.337 s
```

2.6 Sincronización de hilos

Se repitió el proceso para el Código 3.4.7, conforme al código 2.3.6 presentando, mostrando finalmente la siguiente salida, que fue igual que en el ejemplo anterior, ya que cada hilo antes de empezar espera a que el anterior termine. Incluso en un mejor tiempo.

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---
[Hola]
[Mundo]
[Sincronizado]
-----
BUILD SUCCESS
-----
Total time: 3.974 s
```

2.7 Productor/Consumidor No sincronizado

El programa se compiló sin ninguna modificación y presentó la siguiente salida.

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---  
Pulse Control C para finalizar.  
Produce : 0  
Produce : 1  
Produce : 2  
Produce : 3  
Produce : 4  
Produce : 5  
Produce : 6  
Produce : 7  
Produce : 8  
Produce : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9  
Consume : 9
```

2.7 Productor/Consumidor Sincronizado

Se capturó, y modificó el programa con las correcciones correspondientes, y presentó la siguiente salida.

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---  
Pulse Control C para finalizar.  
Produce : 0  
Consume : 0  
Produce : 1  
Consume : 1  
Produce : 2  
Consume : 2  
Produce : 3  
Consume : 3  
Produce : 4  
Consume : 4  
Produce : 5  
Consume : 5  
Produce : 6  
Consume : 6  
Produce : 7  
Consume : 7  
Produce : 8  
Consume : 8  
Produce : 9  
Consume : 9
```

```
-----  
BUILD SUCCESS  
-----
```

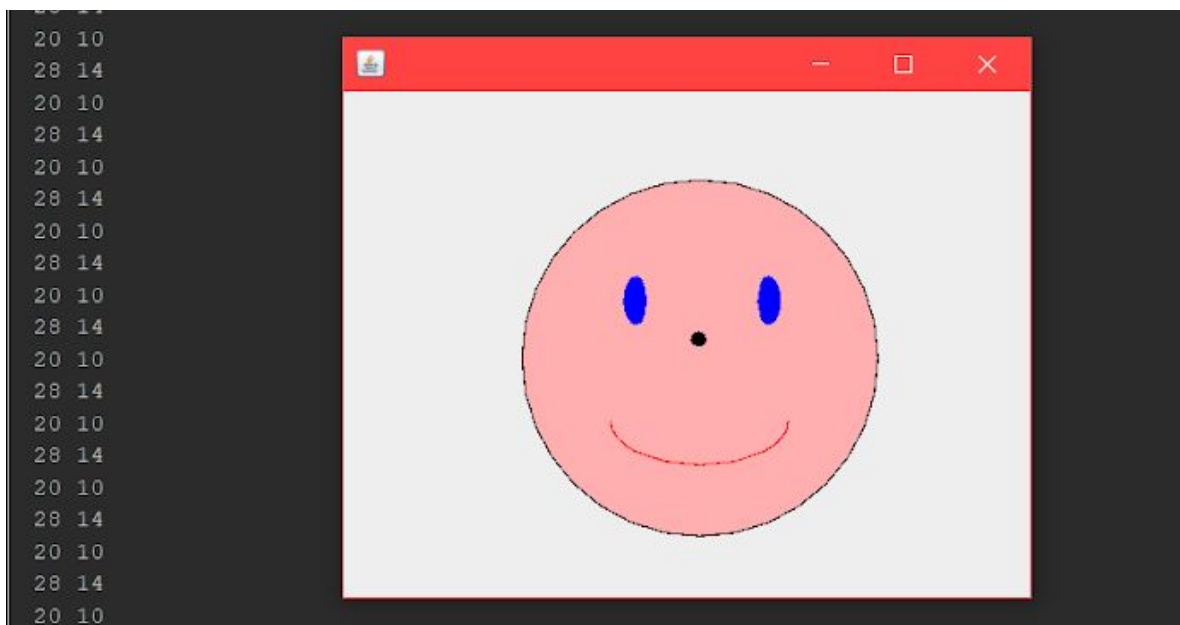
2.8 Bloqueos

El bloqueo se produce **cuando** dos hilos tienen una dependencia circular en un par de objetos sincronizados.

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Hilos ---
Hilo hijo entra en B.obstruir()
Hilo Principal entra en A.construir()
Hilo hijo llama al metodo A.ultimo()
Hilo Principal llama al metodo B.ultimo()
```

El programa al bloquearse tuvo que ser finalizado pulsando **ctrl C**

2.9 Programación Multihilo



Tremenda carita. Interesante aplicación de los hilos.

2.10 Marquesita

Diseño

El primer cálculo de importante es la distribución de los cuatro sectores para los nombres, la siguiente imagen muestra un modelo sencillo de distribución que generado con la clase **Random**² de Java. Generando una coordenada aleatoria dentro de un determinado rango.

Para una pantalla de 1800 píxeles, serán cuatro sectores iguales con un tamaño de 450 píxeles, a cada uno de estos sectores se le cercenara el espacio correspondiente al ancho de la palabra (JLabel width), en este caso proponemos 350 píxeles. Cada sector tendrá un límite inferior y uno superior. El límite inferior para el primer Rango, llamémoslo A_1 , será cero puesto que nos interesa que pueda, según el azar, aparecer desde el píxel inicial de nuestro espacio. Ahora, para el límite superior se le restará el tamaño del ancho de la palabra al tamaño del sector.

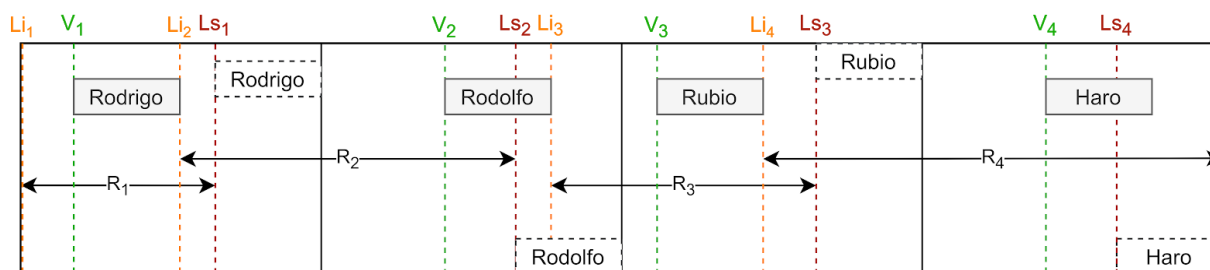
Por lo tanto para el primer sector tendríamos, con estos parámetros, un límite inferior $Li_1 = 0$ y un límite superior $Li_1 = 450 - 350 = 100$.

Ahora, para el segundo sector tendremos un rango R_2 definido con los límites:

$Li_2 = V_1 + width$, donde V_1 es el valor escogido al azar del sector 1 y $width$ es el tamaño de la palabra³, y un límite superior $Li_2 = (450*2) - 350 = 550$. Repitiendo el proceso para el rango R_3 del sector 3, obviándose, indicamos únicamente el límite superior $Li_3 = 1000$.

Finalmente, para el sector 4, el rango quedará definido como,

$Li_4 = V_3 + width$, donde V_3 es el valor escogido al azar del sector 3 y $width$ es el tamaño de la palabra, y un límite superior $Li_4 = (450*4) - 350 = 1450$.



Modelo de Distribución de Posición Aleatoria por Sector

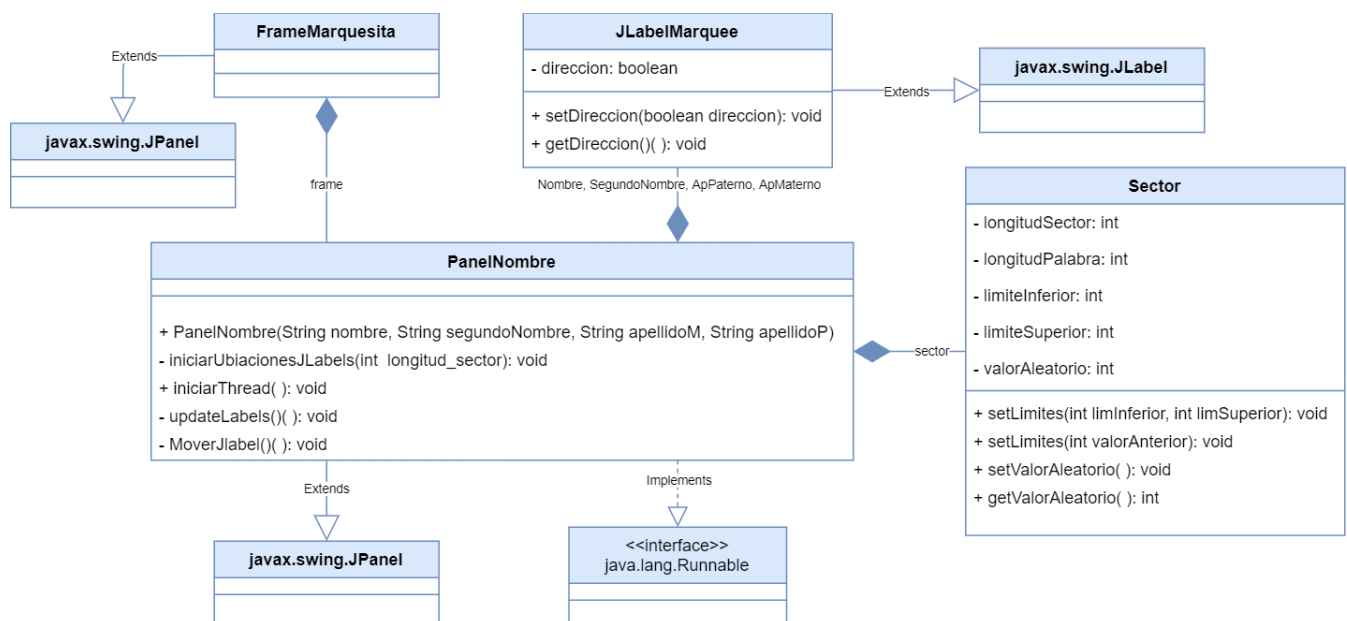
² La clase Random está disponible en la siguiente dirección:

<https://docs.oracle.com/javase/8/docs/api/java/util/Random.html>

³ Por cuestiones prácticas todas las palabras recibieron para esta actividad el mismo tamaño, $width = 350$ píxeles lo que ocasiona que los últimos apellidos, al ser más pequeños choquen antes debido al espacio en blanco.

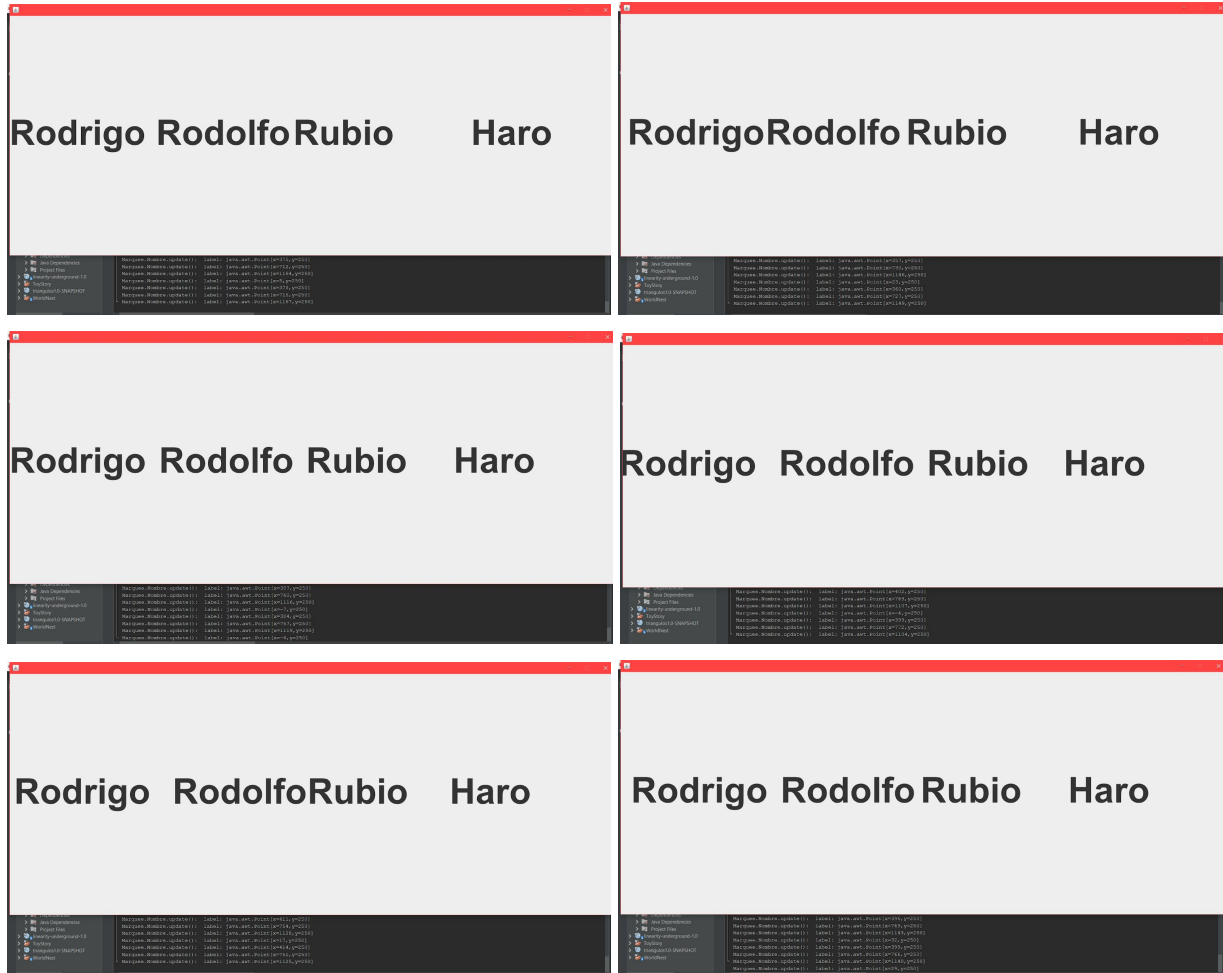
Diagrama de Clases

Se proponen cuatro clases indispensables para el desarrollo de esta práctica, tomando de partida la estructura del ejemplo de la carita generada con hilos, y por supuesto lo anterior mencionado, el modelo de distribución de posición Aleatoria manejando sectores. PanelNombre contendrá una instancia de sector, que se reutilizará para simular tener los 4 sectores, puesto que no es de interés conservar los límites anteriores una vez desplegada la JLabel de un sector. PanelNombre será además un JPanel y manejará el hilo para Pintar las diferentes posiciones de los JLabel, se tendrá uno de estos últimos por cada nombre y apellido.



Ejecución

Unas breves capturas del resultado, se presentaron varios retos interesantes, la consola imprime la ubicación de los JLabel en cada momento.



3. Conclusión

Los hilos, en lo particular se me hacen algo complicados sobre todo multihilos y el tema de sincronización. Sin embargo entiendo la importancia de estos. Vi, con sumo interés que algunos problemas como el de la multiplicación de matrices de forma óptima hacen uso de algoritmos con multihilos.