

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

## INFORME DE LABORATORIO

### (formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	SISTEMAS DISTRIBUIDOS				
TITULO DE LA PRÁCTICA:	Los Hilos (Threads)				
NÚMERO DE PRÁCTICA:	01	AÑO LECTIVO:	2024	NRO. SEMESTRE:	2024A
FECHA DE PRESENTACIÓN	03/05/2024	HORA DE PRESENTACIÓN			
INTEGRANTE (s) Calcina Puma Esteven Antonio				NOTA (0-20)	
DOCENTE(s): Mg. Maribel Molina Barriga					

RESULTADOS Y PRUEBAS
<p><b>I. Marco Conceptual</b></p> <p>Un hilo (Thread) es un proceso en ejecución dentro de un programa Puede haber varios hilos en ejecución simultánea Los hilos implementan prioridad y mecanismos de sincronización La finalización depende del hilo Cualquier clase se puede hacer hilo extends Thread implements Runnable</p> <p>Un proceso es una entidad que posee 2 características importantes: Recursos: Básicamente: un espacio de direcciones (programas, datos y pila y un PCB), archivos, memoria, etc. El SOP realiza la función de protección para evitar interferencias no deseadas entre procesos en relación con los recursos. Planificación/Ejecución: El proceso sigue una ruta de ejecución. Tiene un PC, un Estado de ejecución (Listo, bloqueado, ejecutándose, etc.) y una prioridad. Estas dos características son independientes y pueden ser tratadas como tales por los sistemas de operación. En algunos sistemas de operación se le denomina a la unidad activa hilo (thread) y a la unidad propietaria de recursos se le suele denominar proceso o tarea.</p> <p>EN UN ENTORNO MULTITHILO SE LE ASOCIA A LOS PROCESOS:</p> <ul style="list-style-type: none"> <li>• Un espacio de direcciones virtuales que soporta la imagen del proceso.</li> <li>• Acceso protegido a procesadores, otros procesos, archivos y recursos de E/S</li> <li>• En un entorno multithilo se asocian a cada hilo:</li> <li>• Un estado de ejecución.</li> <li>• Un PC, un contexto (conjunto de registros) que se almacena cuando no está en ejecución.</li> <li>• Una pila.</li> <li>• Un espacio de almacenamiento para variables locales.</li> <li>• Acceso a la memoria y recursos del proceso.</li> </ul> <p>Para que la comunicación entre procesos sea posible es necesario usar las llamadas al sistema. Los threads comparten el mismo espacio de direcciones. Por lo tanto, el cambio de contexto entre un</p>

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 2</p>

thread y otro que pertenecen al mismo proceso pudiera hacerse de forma totalmente independiente del sistema de operación. Los cambios de contexto de procesos son más costosos. Implican salvar el contexto del proceso, cambio de modo(trap al sistema de operación, etc), otro cambio de modo, restaurar contexto del nuevo proceso. IMPLEMENTACIONES DE HILOS: Threads a nivel de usuario. Threads a nivel de kernel Enfoque combinado, procesos Livianos

Usando hilos, se puede permitir el uso de llamadas al sistema bloqueantes sin necesidad de “bloquear” todo el proceso. Esta propiedad vuelve a los hilos particularmente atractivos para su uso dentro de sistemas distribuidos. Concentrémonos en la arquitectura Cliente Servidor.

#### CLIENTES MULTIHILOS

Sirven para esconder la latencia de comunicación a través de la red.

##### Ejemplo 1: Navegadores WEB

En muchos casos una página WEB consiste de un texto plano con múltiples figuras. Con frecuencia el navegador, establece la conexión con el servidor, recupera y comienza a desplegar la página HTML (incluso se permite al usuario el desplazamiento dentro de la página) mientras el navegador continúa recuperando otros archivos que conforman la página.

Desarrollar navegadores multihilos simplifica este hecho de forma considerable. Tan pronto como llega la página principal se pueden activar hilos que se encarguen de recuperar las demás partes. Cada hilo establece su propia conexión con el servidor. Mientras tanto el usuario advierte el retardo en las imágenes, pero puede ir explorando el documento.

Si el servidor está saturado o es lento no se observarán mejoras notables en el rendimiento.

##### Ejemplo 2: Servidores WEB Replicados

En muchos casos los servidores se replican en distintas máquinas y cada servidor proporciona el mismo conjunto de documentos WEB. Están localizados en el mismo sitio y se conocen por el mismo nombre.

Cuando entra una petición para una página WEB es re-entregada a uno de los servidores (usando round-robin u otra técnica de balanceo de carga).

Cuando se usan clientes multihilos cada conexión puede ir a una réplica diferente del mismo servidor. En este caso los distintos archivos se transmiten en paralelo asegurando que la página WEB completa se despliega en un tiempo más corto

El principal uso de la tecnología multihilos está del lado del servidor. Básicamente buscan mejorar el desempeño (aún en servidores monoprocesador) y la forma cómo se estructura el servidor.

Ejemplo: Por lo general un servidor de archivos espera una petición de entrada para una operación de archivo, posteriormente ejecuta la petición (operación bloqueante al disco) y luego envía la respuesta de regreso.

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 3</p>

Modelo Servidor/Trabajador: Las peticiones son enviadas por los clientes hasta el servidor. Después de examinar la petición el hilo servidor elige un hilo trabajador sin utilizar y le encarga la petición. El hilo trabajador realiza la lectura, lo cual puede provocar que se suspenda hasta que los datos sean recuperados.

Si el hilo se suspende, el procesador, selecciona otro para su ejecución. ¿Cómo se programa un servidor de archivos en la ausencia de hilos? El servidor recibe a peticiones, las examina y las trata de resolver antes de recibir la siguiente petición. Mientras espera por el disco el servidor está ocioso y no procesa otra petición. Se procesan menos peticiones por segundo (throughput).

Con los hilos se gana un rendimiento considerable. Cada hilo se programa secuencialmente en forma tradicional.

Supongamos que los hilos no están disponibles, pero los diseñadores del sistema consideran inaceptable el rendimiento debido al uso de un solo hilo y operaciones bloqueantes

## II. EJERCICIOS RESUELTOS:

*Caso Estudio: simular el proceso de cobro de un supermercado*

*Unos clientes van con un 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 vamos a definir una clase "Cajera" y una clase "Cliente" el cual tendrá un "array de enteros" que representaran los productos que ha comprado y el tiempo que la cajera tardará en pasar el producto por el escáner; es decir, que si tenemos 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'.*

*Código en Anexo 1:*

*Explicado este ejemplo vamos a ver cómo hemos definido estas clases:*

- Clase "Cajera.java"
- Clase "Cliente.java"
- Clase "Main.java":

*Como vemos se procesa primero la compra del cliente 1 y después la compra del cliente 2*

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 4</p>

*tardando en procesar ambas compras un tiempo de 26 segundos.*

*¿Y si en vez de procesar primero un cliente y después otro, procesásemos los dos a la vez?,*

*¿Cuánto tardaría el programa en ejecutarse?. Pues bien si en vez de haber solo una Cajera (es decir un solo hilo), hubiese dos Cajeras (es decir dos hilos o threads) podríamos procesar los dos clientes a la vez y tardar menos tiempo en ejecutarse el programa.*

*Para ello debemos de modificar la clase "Cajera.java" y hacer que esta clase herede de la clase Thread para heredar y sobre-escribir algunos de sus métodos. Primero vamos a ver como codificamos esta nueva clase "CajeraThread.java" y después explicamos sus características.*

*Lo primero que vemos y que ya hemos comentado es que la clase "CajeraThread" debe de heredar de la clase Thread: "extendsThread".*

*Otra cosa importante que vemos es que hemos sobre-escrito el método "run()" (de ahí la etiqueta @Override) . Este método es imprescindible sobre-escribirlo (ya que es un método que está en la clase Runnable y la clase Thread Implementa esa Interface) porque en él se va a codificar la funcionalidad que se ha de ejecutar en un hilo; es decir, que lo que se programe en el método "run()" se va a ejecutar de forma secuencial en un hilo.*

*En esta clase "CajeraThread" se pueden sobre-escribir más métodos para que hagan acciones sobre el hilo o thread como, por ejemplo, parar el thread, ponerlo en reposos, etc*

- Clase CajeraThread.java

*A continuación vamos a ver como programamos el método Main para que procese a los clientes de forma paralela y ver como se tarda menos en procesar todo. El método Main esta en la clase "MainThread.java" que tiene el siguiente contenido:*

- Clase "MainThread.java"

*Ahora vamos a ver cuál sería el resultado de esta ejecución y vamos a comprobar como efectivamente el programa se ejecuta de forma paralela y tarda solo 15 segundos en terminar su ejecución.*

*Otra forma de hacer lo mismo, pero sin heredar de la clase "Thread" es implementar la Interface "Runnable". En este caso no dispondremos ni podremos sobre-escribir los métodos de la clase*

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 5</p>

*Thread ya que no la vamos a utilizar y solo vamos a tener que sobre-escribir el método "run()".*

*En este caso solo será necesario implementar el método "run()" para que los procesos implementados en ese método se ejecuten en un hilo diferente. Vamos a ver un ejemplo de cómo utilizando objetos de las clases "Cliente.java" y "Cajera.java" podemos implementar la multitarea en la misma clase donde se llama al método Main de la aplicación. A continuación, vemos la codificación en la clase:*

- Clase "MainRunnable.java"

*El concepto de multitarea o multiprocesamiento es bastante sencillo de entender ya que solo consiste en hacer varias cosas a la vez sin que se vea alterado el resultado final. Como ya se ha dicho en la entrada no todo se puede paralelizar y en muchas ocasiones suele ser complicado encontrar la manera de paralelizar procesos dentro de una aplicación sin que esta afecte al resultado de la misma, por tanto, aunque el concepto sea fácil de entender el aplicarlo a un caso práctico puede ser complicado para que el resultado de la aplicación no se vea afectado.*

Primero probamos el código sin aplicar hilos y vemos que el tiempo total es de 26 segs.

*a. Ejecutando código de Cajera-Cliente*

```
stv@stv--pc: ~/SD/labs/lab01
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/SD/labs/lab01
We start again !!! stv
stv@stv--pc:[$cd SD/labs/lab01/
stv@stv--pc:[$ls
cajera.java Cliente.java Main.java
stv@stv--pc:[$ls
cajera.java Cliente.java Main.java
stv@stv--pc:[$javac cajera.java Cliente.java Main.java
stv@stv--pc:[$ls
cajera.class cajera.java Cliente.class Cliente.java Main.class Main.java
stv@stv--pc:[$java Main
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg
Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
stv@stv--pc:[$
```

```

stv@stv--pc: ~/SD/labs/lab01
Archivo Acciones Editar Vista Ayuda

stv@stv--pc: ~/SD/labs/lab01

19 public cajera() {
18 }
17 public cajera(String nombre) {
16     this.nombre = nombre;
15 }
14 public String getNombre() {
13     return nombre;
12 }
11 public void setNombre(String nombre) {
10     this.nombre = nombre;
9 }
8 public void procesarCompra(Cliente cliente, long timeStam
7 p) {
6     //timeStamp es un tiempo de referenciam, para saber cua
5 nto esta demorando
4     System.out.println("La cajera " + this.nombre +
3 " COMIENZA A PROCESAR LA COMPRA DEL CLIENTE " + cli
2 ente.getNombre() +
1 " EN EL TIEMPO: " + (System.currentTimeMillis() - t
imeStamp) / 1000 +
0 "seg");
23 for (int i = 0; i < cliente.getCarroCompra().length; i+
1 ) {
2     this.esperarXsegundos(cliente.getCarroCompra()[i]);
3     System.out.println("Procesado el producto " + (i + 1)
+
4 " ->Tiempo: " + (System.currentTimeMillis() - tim
5 eStamp) / 1000 +
6 "seg");
7     System.out.println("La cajera " + this.nombre + " HA TE
8 RMINADO DE PROCESAR " +
9 cliente.getNombre() + " EN EL TIEMPO: " +
0 (System.currentTimeMillis() - timeStamp) / 1000 + "
1 seg");
2 }
3 }

11 public class Main {
10     public static void main(String[] args) {
9         Cliente cliente1 = new Cliente("Cliente 1", new int[] {
8 2, 2, 1, 5, 2, 3 });
7         Cliente cliente2 = new Cliente("Cliente 2", new int[] {
6 1, 3, 5, 1, 1 });
5         cajera cajera1 = new cajera("Cajera 1");
4         cajera cajera2 = new cajera("Cajera 2");
3         // Tiempo inicial de referencia
2         long initialTime = System.currentTimeMillis();
1         cajera1.procesarCompra(cliente1, initialTime);
0         cajera2.procesarCompra(cliente2, initialTime);
12 }
13 }

[3] Main.java [java] 12 : 1 Todo
14 public class Cliente {
13     private String nombre;
12     private int[] carroCompra;
11     public Cliente() {
10 }
9     public Cliente(String nombre, int[] carroCompra) {
8         this.nombre = nombre;
7         this.carroCompra = carroCompra;
6 }
5     public String getNombre() {
4         return nombre;
3 }
2     public void setNombre(String nombre) {
1         this.nombre = nombre;
0 }

15 }
1     public int[] getCarroCompra() {
2         return carroCompra;
3 }

[1] cajera.java [+] [java] 23 : 61 27% [2] Cliente.java [java] 15 : 3 Comienzo

```

Ahora probamos usando hilos, heredando de la clase Threads y sobrescribiendo la clase run()

```
stv@stv--pc: ~/SD/labs/lab01
Archivo Acciones Editar Vista Ayuda

stv@stv--pc: ~/SD/labs/lab01
19 this.initialTime = initialTime;
18 }
17 public Cliente getCliente() {
16 return cliente;
15 }
14 public void setCliente(Cliente cliente) {
13 this.cliente = cliente;
12 }
11 @Override
10 public void run() {
9 System.out.println("La cajera " + this.nombre + " COMIE
NZA A PROCESAR LA COMPRA DEL CLIENTE "
8 + this.cliente.getNombre() + " EN EL TIEMPO: "
7 + (System.currentTimeMillis() - this.initialTime) /
6 1000
5 + "seg");
4 for (int i = 0; i < this.cliente.getCarroCompra().length; i++) {
3 // Se procesa el pedido en X segundos
2 this.esperarXsegundos(cliente.getCarroCompra()[i]);
1 System.out.println("Procesado el producto " + (i + 1)
+ " del cliente " + this.cliente.getNombre() + "
>Tiempo: "
43 + (System.currentTimeMillis() - this.initialTime)
/ 1000
1 + "seg");
2 }
3 System.out.println("La cajera " + this.nombre + " HA TE
MINADO DE PROCESAR "
4 + this.cliente.getNombre() + " EN EL TIEMPO: "
5 + (System.currentTimeMillis() - this.initialTime) /
6 1000
7 + "seg");
8 private void esperarXsegundos(int segundos) {
9 try {
10 Thread.sleep(segundos * 1000);
11 } catch (InterruptedException ex) {
14 public class MainThread {
13
12 public static void main(String[] args) {
11 Cliente cliente1 = new Cliente("Cliente 1", new int[] {
2, 2, 1, 5, 2, 3 });
10 Cliente cliente2 = new Cliente("Cliente 2", new int[] {
1, 3, 5, 1, 1 });
9 //Tiempo inicial de referencia
8 long initialTime = System.currentTimeMillis();
7 CajeraThread cajera1 = new CajeraThread("Cajera 1", cli
ente1, initialTime);
6 CajeraThread cajera2 = new CajeraThread("Cajera 2", cli
ente2, initialTime);
5
4 cajera1.start();
3 cajera2.start();
2 }
1
15 }
[4] CajeraThread.java [+] [java] 43 : 39 85% [5] MainThread.java [java] 15 : 1 Todo
18:33 1 may
```



```

stv@stv--pc: ~/SD/labs/lab01
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/SD/labs/lab01
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg
Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
stv@stv--pc:~/SD/labs/lab01]$javac CajeraThread.java Main
Main.java      MainThread.java
stv@stv--pc:~/SD/labs/lab01]$javac CajeraThread.java Main
Main.java      MainThread.java
stv@stv--pc:~/SD/labs/lab01]$javac CajeraThread.java MainThread.java
stv@stv--pc:~/SD/labs/lab01]$java MainThread
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
stv@stv--pc:~/SD/labs/lab01]$
  
```

Y por ultimo probamos implementando la interface Runnable

```
stv@stv--pc: ~/SD/labs/lab01
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/SD/labs/lab01
19 public class MainRunnable implements Runnable{
18
17     private Cliente cliente;
16     private cajera cajera;
15     private long initialTime;
14     public MainRunnable (Cliente cliente, cajera cajera, long initialTime){
13         this.cajera = cajera;
12         this.cliente = cliente;
11         this.initialTime = initialTime;
10     }
9
8     public static void main(String[] args) {
7         Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5, 2, 3 });
6         Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1, 1 });
5         cajera cajera1 = new cajera("Cajera 1");
4         cajera cajera2 = new cajera("Cajera 2");
3         // Tiempo inicial de referencia
2         long initialTime = System.currentTimeMillis();
1         Runnable proceso1 = new MainRunnable(cliente1, cajera1, initialTime);
20         Runnable proceso2 = new MainRunnable(cliente2, cajera2, initialTime);
1         new Thread(proceso1).start();
2         new Thread(proceso2).start();
3     }
4     @Override
5     public void run() {
6         this.cajera.procesarCompra(this.cliente, this.initialTime);
7     }
8
9 }
```

[6] MainRunnable.java [java] 20 : 47 Todo

:q

stv Brave-b...windows stv@stv--pc qtermi...indows 18:53 1 may

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 11</p>

```

stv@stv--pc: ~/SD/labs/lab01
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/SD/labs/lab01
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
stv@stv--pc:~/SD/labs/lab01$javac MainRunnable.java
stv@stv--pc:~/SD/labs/lab01$java MainRunnable
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 ->Tiempo: 1seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 3 ->Tiempo: 9seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
stv@stv--pc:~/SD/labs/lab01$

```

#### b. Evaluar resultados obtenidos.

Se puede ver que realizando las operaciones con hilos el programa se ejecuta en menor tiempo, esto es debido a la concurrencia de los hilos

#### c. Escriba un reporte sobre las tareas realizadas y resultados.

- Primero se probó el programa sin usar hilos, como resultado el programa se demoró 26 segundos y se hizo de forma secuencial.
- Segundo se probó usando hilos heredando de la Clase Threads y se demoró 15 segundos
- Tercero se probó usando hilos, pero esta vez se implementó Runnable y se demoró 15 segundos

#### d. Escriba sus conclusiones

- Los hilos pueden mejorar el rendimiento de la ejecución, por medio de la concurrencia
- La implementación de los hilos en Java es relativamente fácil y sencilla, a diferencia de c++

## CASO PRODUCTOR CONSUMIDOR

a. Implementar, ejecutar y probar el código de los anexos 2, de manera adecuada.

```

stv@stv--pc: ~/SD/labs/lab01/productorConsumidor
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/S...ductorConsumidor

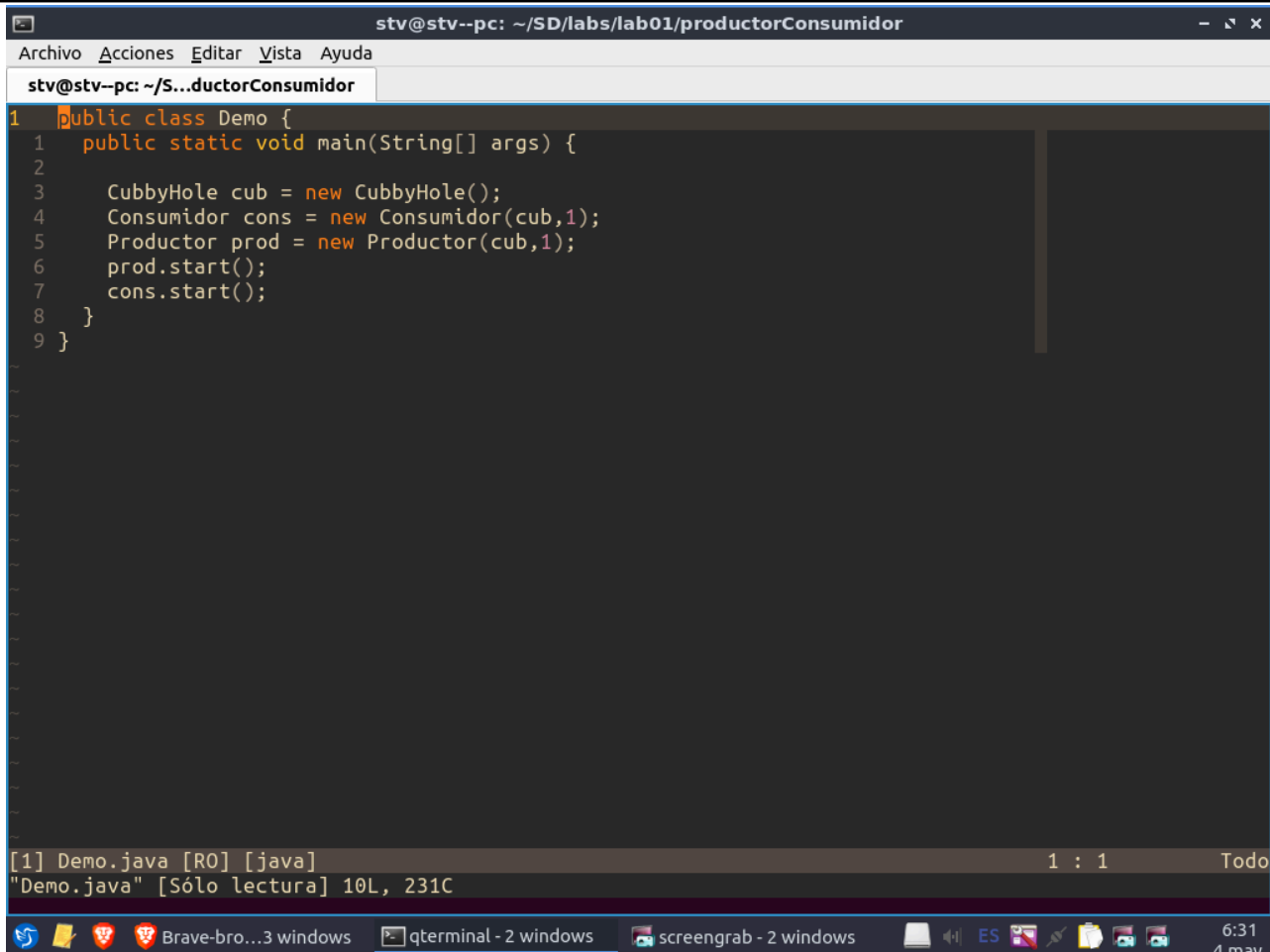
18 public class CubbyHole{
17
16     private int contents;
15     private boolean available = false;
14
13     public synchronized int get() {
12         while (available == false) {
11             try {
10                 wait();
9             } catch (InterruptedException e) { }
8         }
7         available = false;
6         notifyAll();
5         return contents;
4     }
3     public synchronized void put(int value) {
2         while (available == true) {
1             try {
19                 wait();
1             } catch (InterruptedException e) { }
2         }
3         contents = value;
4         available = true;
5         notifyAll();
6     }
7 }

15 public class Consumidor extends Thread {
14
13     private CubbyHole cubbyhole;
12     private int numero;
11
10     public Consumidor(CubbyHole c, int numero) {
9         cubbyhole = c;
8         this.numero = numero;
7     }
6     public void run() {
5         int value = 0;
4         for (int i = 0; i < 10; i++) {
3             value = cubbyhole.get();
2             System.out.println("Consumidor #" + this.numero + " obtiene:" + valu
e);
1         }
16     }
1 }

[1] Consumidor.java [java] 16 : 3 Todo
3     public Productor(CubbyHole c, int numero) {
2         cubbyhole = c;
1         this.numero = numero;
9     }
1     public void run() {
2         for (int i = 0; i < 10; i++) {
3             cubbyhole.put(i);
4             System.out.println("Productor #" + this.numero + " pone:" + i);
5             try {
6                 sleep((int)(Math.random() * 100));
7             }
8             catch (InterruptedException e) { }
9         }
10     }
11 }

[3] CubbyHole.java [java] 19 : 15 Todo [5] Productor.java [java] 9 : 3 Final
"Productor.java" 20 lines --70%--

```





```
stv@stv--pc: ~/SD/labs/lab01/productorConsumidor
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/S...ductorConsumidor
1 public class Demo {
2     public static void main(String[] args) {
3         CubbyHole cub = new CubbyHole();
4         Consumidor cons = new Consumidor(cub,1);
5         Productor prod = new Productor(cub,1);
6         prod.start();
7         cons.start();
8     }
9 }
```

[1] Demo.java [R0] [java] 1 : 1 Todo  
"Demo.java" [Sólo lectura] 10L, 231C

Brave-bro...3 windows qterminal - 2 windows screengrab - 2 windows 6:31 4 may

```
stv@stv--pc:[~/SD/labs/lab01/productorConsumidor]$java Demo
Productor #1pone:0
Consumidor #1 obtiene:0
Productor #1pone:1
Consumidor #1 obtiene:1
Productor #1pone:2
Consumidor #1 obtiene:2
Productor #1pone:3
Consumidor #1 obtiene:3
Productor #1pone:4
Consumidor #1 obtiene:4
Productor #1pone:5
Consumidor #1 obtiene:5
Productor #1pone:6
Consumidor #1 obtiene:6
Consumidor #1 obtiene:7
Productor #1pone:7
Productor #1pone:8
Consumidor #1 obtiene:8
Productor #1pone:9
Consumidor #1 obtiene:9
stv@stv--pc:[~/SD/labs/lab01/productorConsumidor]$
```

*b. Evaluar resultados obtenidos.*

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 15</p>

```

stv@stv--pc:[~/SD/labs/lab01/productorConsumidor]$java Demo
Productor #1pone:0
Consumidor #1 obtiene:0
Consumidor #1 obtiene:1
Productor #1pone:1
Productor #1pone:2
Consumidor #1 obtiene:2
Consumidor #1 obtiene:3
Productor #1pone:3
Productor #1pone:4
Consumidor #1 obtiene:4
Consumidor #1 obtiene:5
Productor #1pone:5
Consumidor #1 obtiene:6
Productor #1pone:6
Consumidor #1 obtiene:7
Productor #1pone:7
Productor #1pone:8
Consumidor #1 obtiene:8
Productor #1pone:9
Consumidor #1 obtiene:9
stv@stv--pc:[~/SD/labs/lab01/productorConsumidor]$

```

Se puede observar que se crean 10 enteros por parte del productor y son consumidos por el consumidor. Aparentemente parece que el Consumidor obtiene un producto antes que los produzca por la impresión. Sin embargo no sucede así ya que CubbyHole hace la sincronización de que no se consuma hasta que el productor produzca algo. Si bien aparece esa impresión, es porque los for de Consumidor y Productor estan en hilos, y la sincronización está en cómo se producen y consumen, mas no en el System.out.println (el que se ve en pantalla).

*c. Escriba un reporte sobre las tareas realizadas y resultados.*

- Se probó el programa una vez

```
stv@stv--pc:[~/SD/labs/lab01/productorConsumidor]$java Demo
Productor #1pone:0
Consumidor #1 obtiene:0
Productor #1pone:1
Consumidor #1 obtiene:1
Productor #1pone:2
Consumidor #1 obtiene:2
Productor #1pone:3
Consumidor #1 obtiene:3
Productor #1pone:4
Consumidor #1 obtiene:4
Productor #1pone:5
Consumidor #1 obtiene:5
Productor #1pone:6
Consumidor #1 obtiene:6
Consumidor #1 obtiene:7
Productor #1pone:7
Productor #1pone:8
Consumidor #1 obtiene:8
Productor #1pone:9
Consumidor #1 obtiene:9
stv@stv--pc:[~/SD/labs/lab01/productorConsumidor]$
```

Se observa que “aparentemente” el producto 7 se consume antes que se produzca

- Se probó una segunda vez



```

stv@stv--pc: ~/SD/labs/lab01/productorConsumidor
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/S...ductorConsumidor
Exception in thread "Thread-1" Exception in thread "Thread-0" java.lang.IllegalMonitorStateException
    at java.base/java.lang.Object.notifyAll(Native Method)
    at CubbyHole.get(CubbyHole.java:13)
    at Consumidor.run(Consumidor.java:13)
java.lang.IllegalMonitorStateException
    at java.base/java.lang.Object.notifyAll(Native Method)
    at CubbyHole.put(CubbyHole.java:24)
    at Productor.run(Productor.java:12)
stv@stv--pc:~/SD/labs/lab01/productorConsumidor]$javac Demo.java Productor.java Consumidor.java Cubby
Hole.java
stv@stv--pc:~/SD/labs/lab01/productorConsumidor]$java Demo
Productor #1pone:0
Consumidor #1 obtiene:0
Consumidor #1 obtiene:1
Productor #1pone:1
Productor #1pone:2
Consumidor #1 obtiene:2
Consumidor #1 obtiene:3
Productor #1pone:3
Productor #1pone:4
Consumidor #1 obtiene:4
Consumidor #1 obtiene:5
Productor #1pone:5
Consumidor #1 obtiene:6
Productor #1pone:6
Consumidor #1 obtiene:7
Productor #1pone:7
Productor #1pone:8
Consumidor #1 obtiene:8
Productor #1pone:9
Consumidor #1 obtiene:9
stv@stv--pc:~/SD/labs/lab01/productorConsumidor]$

```

Se ve que hay varios productos que aparentemente se consumen antes que se produzcan

- Se revisa el código

```

Archivo  Acciones  Editar  Vista  Ayuda
stv@stv-pc: ~/S...ductorConsumidor

5 public class CubbyHole{
6   private int contents;
7   private boolean available = false;
8   public synchronized int get() {
9     while (available == false) {
10      try {
11        wait();
12      } catch (InterruptedException e) { }
13    }
14    available = false;
15    notifyAll();
16    return contents;
17  }
18  public synchronized void put(int value) {
19    while (available == true) {
20      try {
21        wait();
22      } catch (InterruptedException e) { }
23    }
24    contents = value;
25    available = true;
26    notifyAll();
27  }
28 }

9   cubbyhole = c;
10  this.numero = numero;
11 }
12 public void run() {
13   int value = 0;
14   for (int i = 0; i < 10; i++) {
15     value = cubbyhole.get();
16     System.out.println("Consumidor #" + this.numero + " obtiene:" + value);
17   }
18 }

[1] Consumidor.java [java] 16 : 3 Final
12 public class Productor extends Thread {
13   private CubbyHole cubbyhole;
14   private int numero;
15   public Productor(CubbyHole c, int numero) {
16     cubbyhole = c;
17     this.numero = numero;
18   }
19   public void run() {
20     for (int i = 0; i < 10; i++) {
21       cubbyhole.put(i);
22       System.out.println("Productor #" + this.numero + " pone:" + i);
23       try {
24         sleep((int)(Math.random() * 100));
25       }
26       catch (InterruptedException e) { }
27     }
28 }

[3] CubbyHole.java [java] 6 : 10 Todo [5] Productor.java [java] 13 : 31 Comienzo

```

Se concluye que la sincronización está en lo que producen y consumen, mas no en la impresión de la línea 13 en el Productor

Esto se demuestra al modificar el código

```

stv@stv--pc: ~/SD/labs/lab01/productorConsumidor
Archivo Acciones Editar Vista Ayuda
stv@stv--pc: ~/S...ductorConsumidor

22 public class CubbyHole{
21
20 private int contents;
19 private boolean available = false;
18
17 public synchronized int get() {
16 while (available == false) {
15 try {
14 wait();
13 } catch (InterruptedException e) { }
12 }
11 System.out.println("Consumidor #" + " obtiene: "
10 "+contents");
9 available = false;
8 notifyAll();
7 return contents;
6 }
5 public synchronized void put(int value) {
4 while (available == true) {
3 try {
2 wait();
1 } catch (InterruptedException e) { }
23 contents = value;
1 System.out.println("Productor #" + " pone: "
+ contents);
2 available = true;
3 notifyAll();
4 }
5 }
6 }

7 cubbyhole = c;
6 this.numero = numero;
5 }
4 public void run() {
3 int value = 0;
2 for (int i = 0; i < 10; i++) {
1 value = cubbyhole.get();
14 }
1 }
2 }

[1] Consumidor.java [java] 14 : 5 Final
9 public class Productor extends Thread {
8
7 private CubbyHole cubbyhole;
6 private int numero;
5
4 public Productor(CubbyHole c, int numero) {
3 cubbyhole = c;
2 this.numero = numero;
1 }
10 public void run() {
1 for (int i = 0; i < 10; i++) {
2 cubbyhole.put(i);
3 try {
4 sleep((int)(Math.random() * 100));
5 }
6 catch (InterruptedException e) { }
7 }
8 }
9 }

[3] CubbyHole.java [java] 23 : 21 Todo [5] Productor.java [java] 10 : 21 Todo
"CubbyHole.java" 29L, 643C escritos

```

Ponemos los System.out.println sincronizados y el resultado es el siguiente

	<p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
<b>Aprobación:</b> 2022/03/01	<b>Código:</b> GUIA-PRLE-001	<b>Página:</b> 20

```

note.java
stv@stv--pc: [~/SD/labs/lab01/productorConsumidor]$ java Demo
Productor #pone:0
Consumidor # obtiene:0
Productor #pone:1
Consumidor # obtiene:1
Productor #pone:2
Consumidor # obtiene:2
Productor #pone:3
Consumidor # obtiene:3
Productor #pone:4
Consumidor # obtiene:4
Productor #pone:5
Consumidor # obtiene:5
Productor #pone:6
Consumidor # obtiene:6
Productor #pone:7
Consumidor # obtiene:7
Productor #pone:8
Consumidor # obtiene:8
Productor #pone:9
Consumidor # obtiene:9
stv@stv--pc: [~/SD/labs/lab01/productorConsumidor]$

```

Lo que confirma que el monitor sincroniza correctamente

#### d. Escriba sus conclusiones

- Java puede sincronizar hilos por medio del uso de monitores. El cual es una clase con métodos con la palabra synchronized, el cual hace que solo un hilo ejecute dicho método.
- Recordar que solo todo lo que está dentro de dicho método está sincronizado.

### III. CUESTIONARIO:

#### a. ¿Por qué es importante el estudio de hilos y multihilos en un sistema distribuido?

- Usualmente los Sistemas distribuidos manejan concurrencia, entonces ahí es donde entran los hilos y los multihilos.
- También facilitan la comunicación y la sincronización de los hilos

#### b. Describe ¿Cómo están compuestos los hilos? y ¿Cuál es la diferencia entre hilos y procesos?

Los hilos son componentes de un proceso que comparten el mismo espacio de memoria y recursos del proceso principal. Cada hilo tiene su propia pila de ejecución, contador de programa y registros, pero comparte el código, datos y archivos del proceso principal.

Diferencia entre hilos y procesos:

Los procesos son entidades independientes que tienen su propio espacio de memoria y recursos asignados por el sistema operativo.

Los hilos comparten el mismo espacio de memoria y recursos del proceso principal.

La creación y destrucción de procesos es más costosa que la de hilos.

La comunicación entre procesos es más compleja y requiere mecanismos como tuberías o memoria compartida, mientras que los hilos pueden comunicarse a través de variables compartidas.

Los hilos se utilizan para aprovechar el paralelismo dentro de un proceso, mientras que los procesos se utilizan para separar tareas independientes o para aislar recursos.

*c. Realice un cuadro comparativo de ventajas y desventajas del uso de hilos*

<i>Ventajas</i>	<i>Desventajas</i>
Escalabilidad y rendimiento: Los hilos permiten el procesamiento paralelo, lo que mejora el rendimiento del sistema.	Complejidad de programación: La programación con hilos puede ser más compleja debido a problemas como condiciones de carrera, deadlocks y la necesidad de sincronización adecuada.
Eficiencia en el uso de recursos: Los hilos comparten memoria y recursos con el proceso principal, lo que reduce la sobrecarga.	Problemas de seguridad: Si un hilo tiene un error, puede afectar a otros hilos dentro del mismo proceso.
Comunicación y sincronización: Los hilos facilitan la comunicación y sincronización entre tareas dentro de un proceso.	Dificultad de depuración: Depurar problemas en programas con múltiples hilos puede ser más difícil debido a la naturaleza concurrente.
Responsividad: Los hilos permiten que una aplicación responda a eventos simultáneos sin bloquear la interfaz de usuario.	Sobrecarga de scheduling: Si hay demasiados hilos, el sistema operativo puede tener una sobrecarga de scheduling, lo que puede disminuir el rendimiento.

### METODOLOGÍA DE TRABAJO

*Colocar la metodología de trabajo que ha utilizado el estudiante o el grupo para resolver la práctica, es decir el procedimiento/secuencia de pasos en forma general.*

### REFERENCIAS Y BIBLIOGRAFÍA

	<p style="text-align: center;"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<p style="text-align: center;"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2022/03/01</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p><b>Página:</b> 22</p>

- [1] Tanenbaum, A.S. (2008). *Sistemas distribuidos: principios y paradigmas*. México. Pearson Educación.
- [2] Ceballos, F. J. (2006). *Java 2, Curso de programación*. México: Alfaomega, RaMa.
- [3] Deitel, H. M., & Deitel, P. J. (2004). *Cómo programar en Java*. México: Pearson Educación.
- [4] García Tomás, J., Ferrando, S., & Piattini, M. (2001). *Redes para procesos distribuidos*. México: AlfaomegaRa-Ma.
- [5] Orfali, R., & Harkey, D. (1998). *Client/Server Programming with Java and CORBA*. USA: Wiley
- [6] *programación multihilo* <https://oscarmaestre.github.io/servicios/textos/tema2.html>
- [7] *Threads en Java*  
[https://lc.fie.umich.mx/~rochoa/Materias/PROGRAMACION/PROGRAMACION\\_2/HILOS.pdf](https://lc.fie.umich.mx/~rochoa/Materias/PROGRAMACION/PROGRAMACION_2/HILOS.pdf)
- [8] <https://biblus.us.es/bibing/proyectos/abreproy/11320/fichero/Capitulos%252F13.pdf>