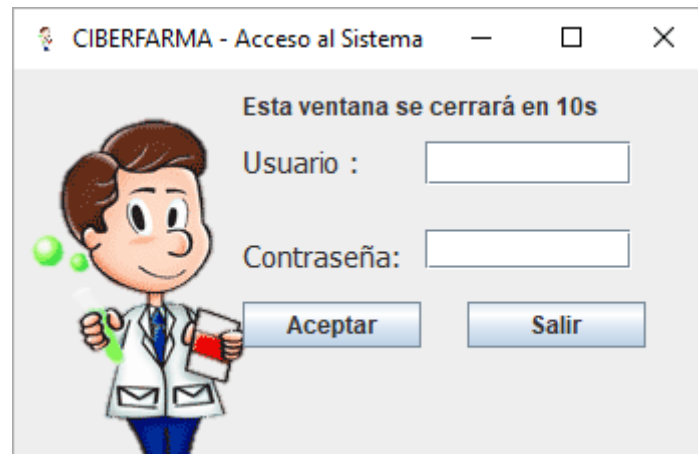


Lenguaje de programación I



Caso

- ✓ Ciberfarma necesita darle seguridad a su sistema de acceso, de manera que después de 10s se cierre la ventana.
- ✓ Si es que pulsa el botón aceptar abrirá la ventana Principal



Contenido

- Threads
 - Conceptos básicos
 - Ciclo de vida
 - Proceso de creación

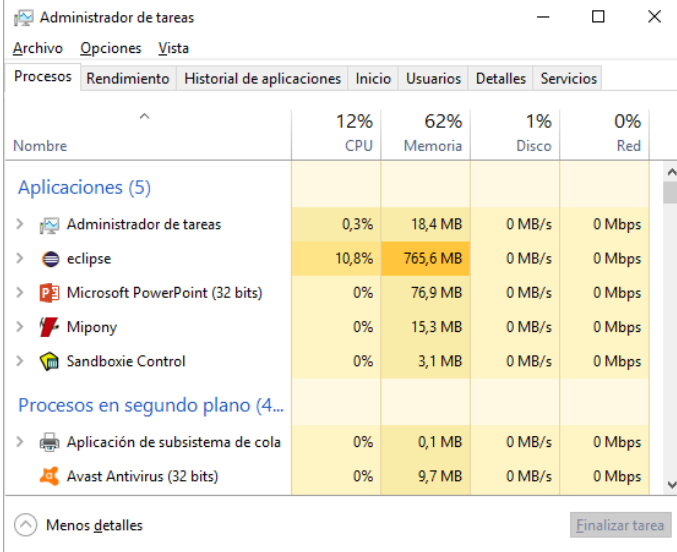
✓ Ejercicios

Logros de la Unidad

- Crear aplicaciones que utilicen eficientemente hilos de ejecución mediante la clase Thread y la interface Runnable utilizando Eclipse como herramienta de desarrollo.

Introducción a los hilos

- Al usar computadoras, la multitarea, nos permite realizar varios procesos a la vez, sin embargo, debemos considerar, que estos comparten los mismos recursos del computador.



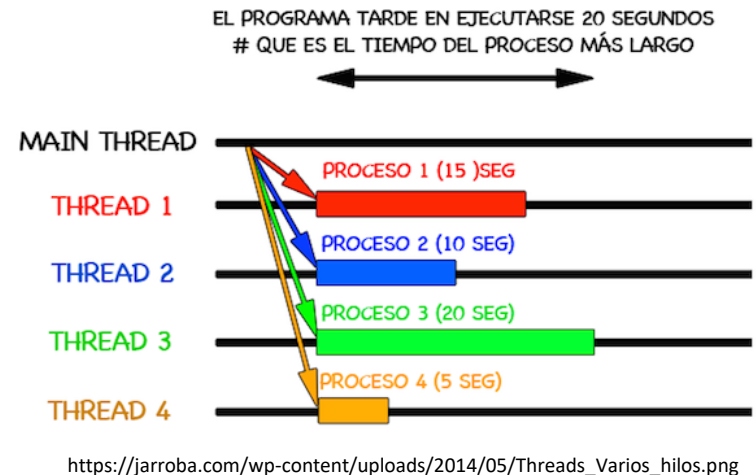
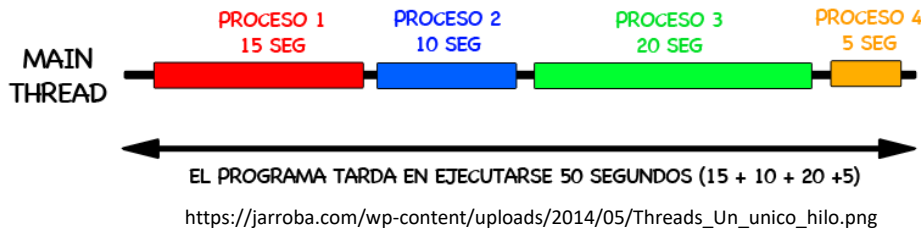
The screenshot shows the Windows Task Manager window with the 'Rendimiento' (Performance) tab selected. It displays a table of system resource usage. The 'Aplicaciones' (Applications) section is expanded, showing the following data:

Nombre	12% CPU	62% Memoria	1% Disco	0% Red
Aplicaciones (5)				
Administrador de tareas	0,3%	18,4 MB	0 MB/s	0 Mbps
eclipse	10,8%	765,6 MB	0 MB/s	0 Mbps
Microsoft PowerPoint (32 bits)	0%	76,9 MB	0 MB/s	0 Mbps
Mipony	0%	15,3 MB	0 MB/s	0 Mbps
Sandboxie Control	0%	3,1 MB	0 MB/s	0 Mbps
Procesos en segundo plano (4...)				
Aplicación de subsistema de cola	0%	0,1 MB	0 MB/s	0 Mbps
Avast Antivirus (32 bits)	0%	9,7 MB	0 MB/s	0 Mbps

- Dentro de un programa, un Thread o hilo es un **flujo secuencial simple** dentro de un proceso.

Introducción a los hilos

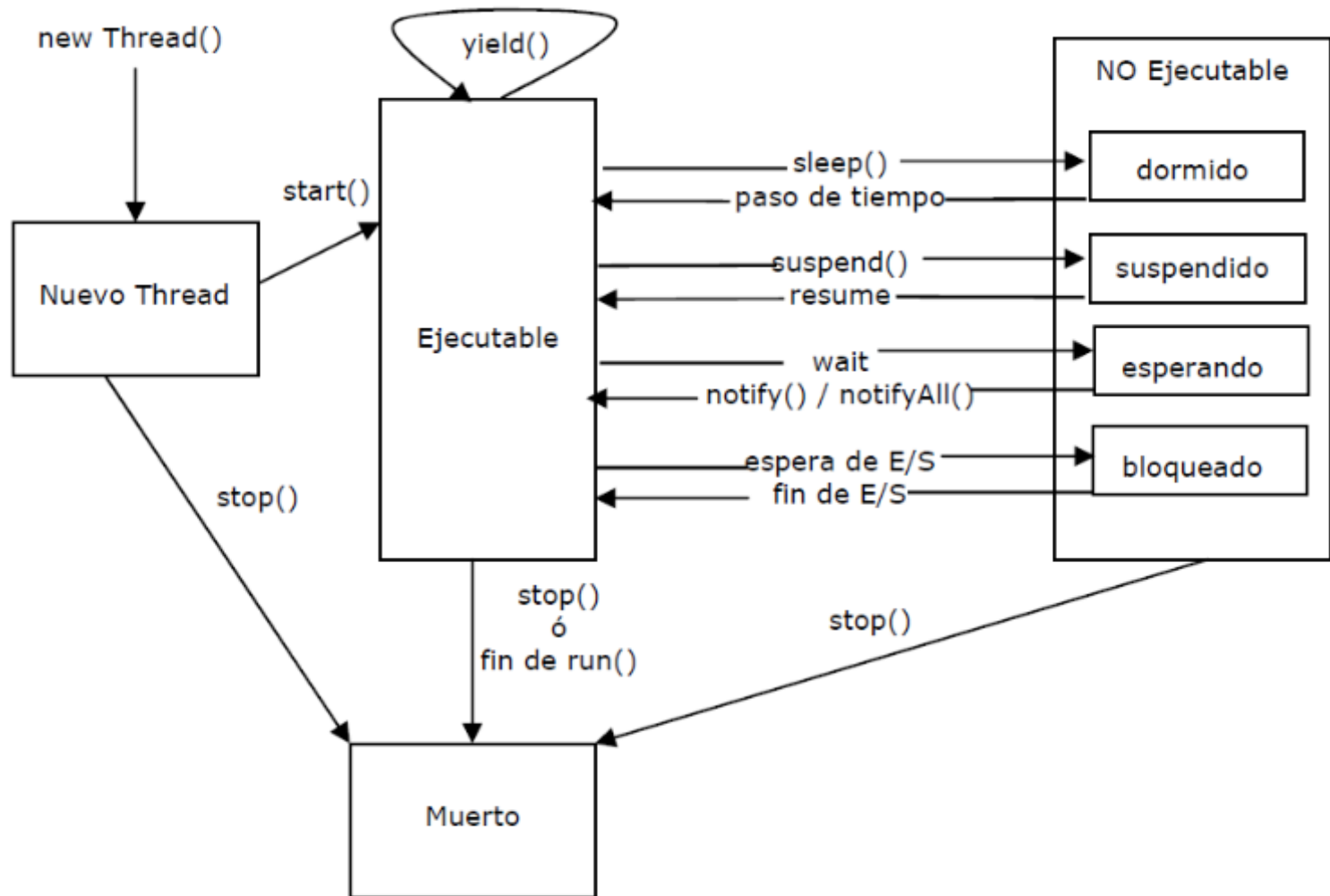
- ✓ La creación de hilos es una característica que permite a las aplicaciones **realizar varios procesos** "aparentemente" a la vez, facilitando la realización de procesos en segundo plano.



- ✓ En el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

Introducción a los hilos

✓ Un hilo tiene un **ciclo de vida** que va desde su creación hasta su finalización



Introducción a los hilos

- ✓ Para gestionar procesos multitarea, mediante hilos, podemos:
- ✓ Heredando la clase **Thread** o implementando la interfaz **Runnable**.

Paso 1. Crear una clase que gestiona el hilo, sobrescribiendo el método **run()**

```
public class MiHilo extends Thread {  
    public void run() {  
        // código del hilo  
    }  
}
```

```
public class MiHilo implements Runnable {  
    public void run() {  
        // código del hilo  
    }  
}
```

Paso 2. Instanciar o Implementar la clase de gestión en la clase aplicación

```
MiHilo hilo = new MiHilo();
```

```
MiHilo tarea = new MiHilo();  
Thread hilo = new Thread(tarea);
```

Paso 3. Iniciar el hilo

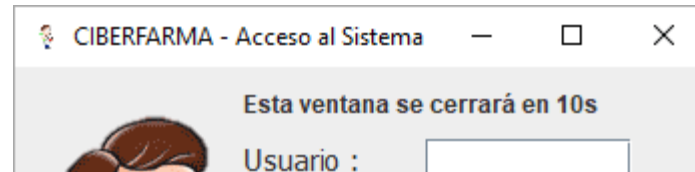
```
hilo.start();
```

```
hilo.start();
```

Aplicación



✓ Del caso:



✓ Para el conteo usaremos:

```
void iniciaTiempo() {  
    for (int i = 10; i >= 0; i--) {  
        lblTiempo.setText(i + "s");  
    }  
}
```

✓ Agregamos una **pausa de 1s**, usando el método sleep:

```
try {  
    Thread.sleep(1000);  
} catch (InterruptedException e) {  
    System.out.println(e);  
}
```

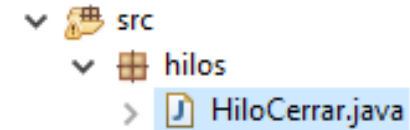
✓ **Importante.** Observa el resultado al ejecutar el código.



Aplicación



- ✓ Creando el paquete y la clase de gestión del hilo:



- ✓ Paso 1. Heredando la clase Thread, sobrescribiendo el método y colocando el código del hilo:

```
public class HiloCerrar extends Thread {  
  
    public void run () {  
        for (int i = 10; i >= 0; i--) {  
            lblTiempo.setText(i + "s");  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                System.out.println(e);  
            }  
        }  
    }  
}
```

La variable **lblTiempo** está definida en otra clase, por lo que habrá que modificarla a **public static**.

```
public static JLabel lblTiempo;
```

Se llamará como:

```
Logueo.lblTiempo.setText(i + "s");
```

- ✓ Paso 2. Instanciamos la clase:

```
void iniciaTiempo() {  
    // Paso 1. Crear la clase y escribir el método run  
    // Paso 2. Instanciar la clase Hilo  
    HiloCerrar hilo = new HiloCerrar();  
    // Paso 3. iniciar el hilo  
    hilo.start();  
}
```

- ✓ Paso 3. Iniciar el hilo:

Aplicación



✓ Para lograr que al terminar el tiempo se cierre sólo la ventana **Logueo**:

✓ Paso 1. Heredando la clase Thread, sobrescribiendo el método y colocando el código del hilo:

```
public class HiloCerrar extends Thread {  
    private JFrame ventana;  
  
    public HiloCerrar(JFrame ventana) {  
        this.ventana = ventana;  
    }  
  
    public void run () {  
        for (int i = 10; i >= 0; i--) {  
            Logueo.lblTiempo.setText(i + "s");  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                System.out.println(e);  
            }  
        }  
        ventana.dispose();  
    }  
}
```

✓ Paso 2. Instanciamos la clase:

```
void iniciaTiempo() {  
    // Paso 1. Crear la clase y escribir el método run  
    // Paso 2. Instanciar la clase Hilo  
    HiloCerrar hilo = new HiloCerrar(this);  
    // Paso 3. iniciar el hilo  
    hilo.start();  
}
```

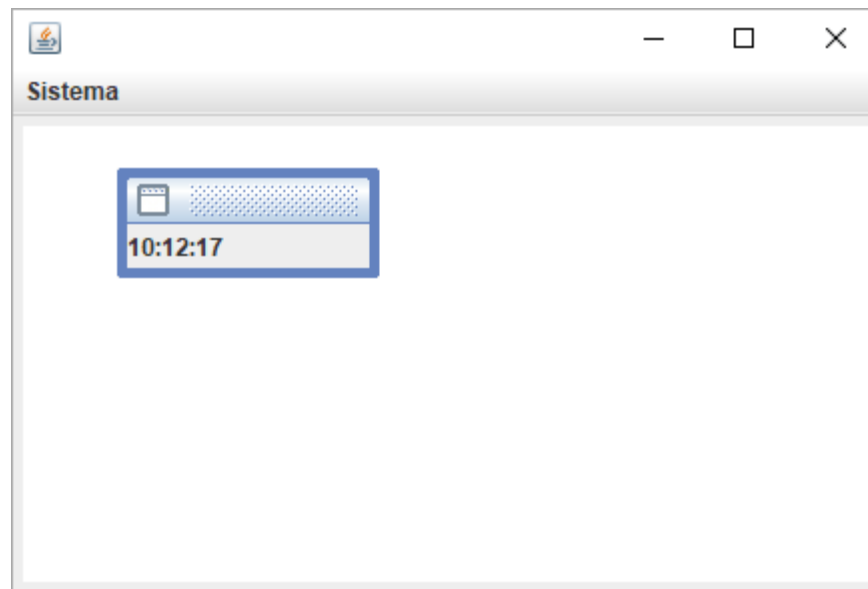
✓ Paso 3. Iniciar el hilo:



Aplicación 2



- ✓ Ciberfarma, necesita colocar un reloj, en su ventana principal, para lo cual se debe actualizar cada segundo y mostrarlo en la etiqueta respectiva

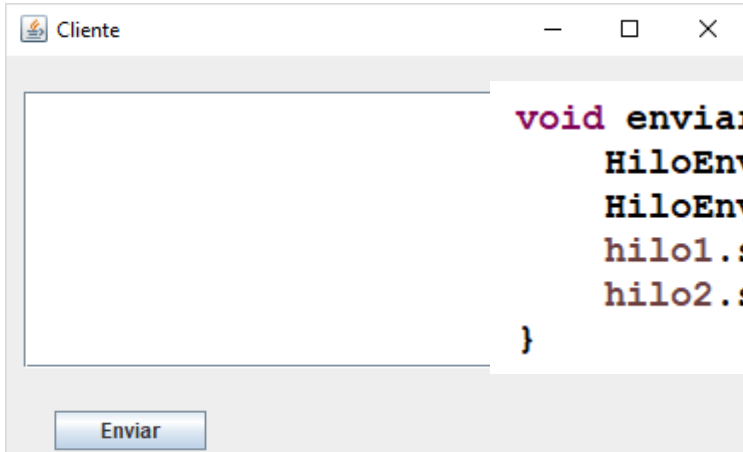


Sincronización



Caso

- ✓ Abre el archivo **Editor**, completa la siguiente actividad y observa el resultado



```
public class HiloEnvio extends Thread {  
    private String nombre;  
  
    public HiloEnvio(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public void run() {  
        for (int i = 1; i <= 10; i++) {  
            Editor.txtEditor.append(nombre + " " + i + "\n");  
        }  
    }  
}
```

¿Cuál terminará primero? ¿Por qué?

Métodos Sincronizados

- **Todos** los objetos de Java **tienen asociado** su propio monitor implícito.
- Para entrar en el monitor de un objeto sólo hay que llamar a un método como **synchronized**.

```
class Ejemplo {  
    synchronized void algo(String msg) { ...
```

- Cuando un hilo esté ejecutando un método sincronizado, todos los demás hilos que intenten ejecutar cualquier método sincronizado del mismo objeto tendrán que esperar.



Sentencia Synchronized

- Este mecanismo requiere colaboración entre los hilos. El que hace el código debe acordarse de poner `synchronized` siempre que vaya a usar fichero. Si no lo hace, el mecanismo no sirve de nada.
- A veces la solución de sincronizar todo un método no es posible o lo más adecuado.

```
synchronized (objeto) {  
    // sentencias que se sincronizan  
} ...
```

- Al poner **`synchronized(objeto)`** marcamos el **objeto** como ocupado desde que se abren las llaves hasta que se cierran. Cuando un segundo hilo intenta **sincronizar** el **objeto**, se bloquea, en espera de que el primero termine.



Aplicación



✓ Modifica el código para sincronizar los mensajes



```
public class HiloEnvio extends Thread {
    private String nombre;

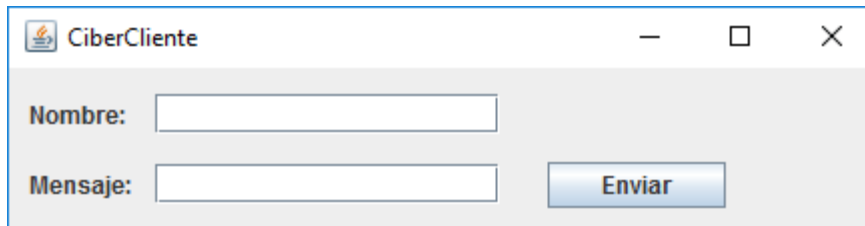
    public HiloEnvio(String nombre) {
        this.nombre = nombre;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            Editor.txtEditor.append(nombre + " " + i + "\n");
        }
    }
}
```

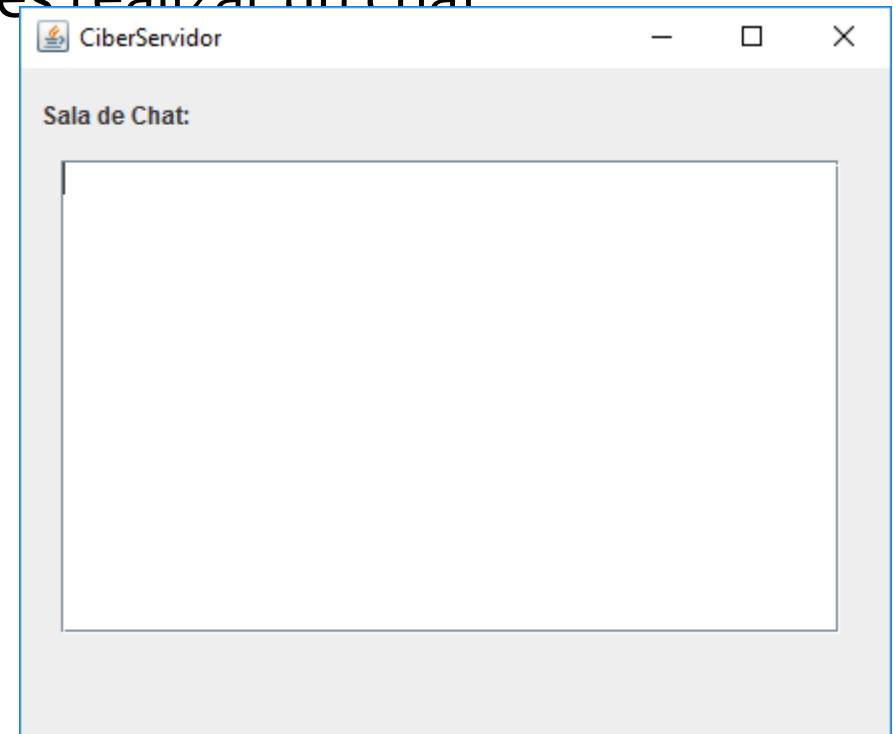

Propuesto 2



- ✓ Otro ejemplo del uso de hilos, es realizar un chat
- ✓ Diseñe lo siguiente:



A screenshot of a Windows application window titled "CiberCliente". It features a light gray background and standard window controls (minimize, maximize, close) in the top right corner. The interface includes two text input fields: the first is labeled "Nombre:" and the second is labeled "Mensaje:". To the right of the "Mensaje:" field is a blue button with the text "Enviar".



A screenshot of a Windows application window titled "CiberServidor". It has a light gray background and standard window controls. Below the title bar, the text "Sala de Chat:" is displayed. The main area of the window is a large, empty white rectangle, intended for displaying chat messages.

Propuesto 2

✓ Para el cliente. Ej:

```
Cliente.java
79 void enviarMensaje() {
80     String HOST = "10.143.136.23"; // "localhost";
81     int PUERTO = 9090;
82     try {
83         // Creamos nuestro socket
84         Socket socket = new Socket(HOST, PUERTO);
85
86         DataOutputStream flujo = new DataOutputStream(socket.getOutputStream());
87
88         // Enviamos un mensaje
89         flujo.writeUTF(txtNombre.getText() + " dice " + txtMensaje.getText());
90
91         // Cerramos la conexión
92         socket.close();
93
94     } catch (Exception e) {
95         System.out.println("Error en cliente >> " + e.getMessage());
96     }
97 }
```

Propuesto 2

- ✓ Para el servidor. Llamamos a un hilo que constantemente revise mensajes enviados. Ej:

```
public void run() {  
    try {  
        ServerSocket servidor = new ServerSocket(9090);  
  
        while (true) {  
            Socket cli = servidor.accept();  
  
            DataInputStream entrada = new DataInputStream(cli.getInputStream());  
  
            String mensaje = entrada.readUTF();  
  
            Servidor.txtSalida.append(mensaje + "\n");  
  
            cli.close();  
        }  
    } catch (IOException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

Agregar:
servidor.close();

Sincronización de hilos

- ✓ Cada hilo contiene todos los datos y métodos necesarios para ejecutarse en su propio espacio.
- ✓ Sin embargo, existen muchas situaciones donde ejecutar threads concurrentes que compartan datos y deben considerar el estado y actividad de otros threads.
- ✓ Este conjunto de situaciones de programación son conocidos como escenarios 'productor/consumidor'; donde el productor genera un canal de datos que es consumido por el consumidor.
- ✓ Ej:
- ✓ En este ejemplo el Productor y el Consumidor comparten datos a través de un objeto **CubbyHole** común.



- ✓ El Productor genera un entero entre 0 y 9 (inclusive), lo almacena en un objeto "CubbyHole", e imprime el número
- ✓ El Consumidor, consume todos los enteros de CubbyHole tan rápidamente como estén disponibles.

```
class Producer extends Thread {  
    private CubbyHole cubbyhole;  
    private int number;  
  
    public Producer(CubbyHole c, int number) {  
        cubbyhole = c;  
        this.number = number;  
    }  
  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            cubbyhole.put(i);  
            System.out.println("Productor #" + this.number + " pone: " + i);  
            try {  
                sleep((int) (Math.random() * 100));  
            } catch (InterruptedException e) {  
            }  
        }  
    }  
}
```

```
class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
            System.out.println("Consumidor #" + this.number + " obtiene: " + value);
        }
    }
}

class ProducerConsumerTest {
    public static void main(String[] args) {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);

        p1.start();
        c1.start();
    }
}
```

Referencia

- ✓ <https://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/>
- ✓ <http://labojava.blogspot.pe/2012/10/sincronizacion.html>
- ✓ [http://www.binarykode.com/bdescargas/Manuales%20y%20Documentos/JAVA/Interfaces%20de%20Usuario/Tutorial%20JAVA%20avanzado%20\(I\)/threads/synchronization.html](http://www.binarykode.com/bdescargas/Manuales%20y%20Documentos/JAVA/Interfaces%20de%20Usuario/Tutorial%20JAVA%20avanzado%20(I)/threads/synchronization.html)

GRACIAS



SEDE MIRAFLORES

Calle Díez Canseco Cdra 2 / Pasaje Tello
Miraflores – Lima
Teléfono: 633-5555

SEDE INDEPENDENCIA

Av. Carlos Izaguirre 233
Independencia – Lima
Teléfono: 633-5555

SEDE BREÑA

Av. Brasil 714 – 792
(CC La Rambla – Piso 3)
Breña – Lima
Teléfono: 633-5555

SEDE TRUJILLO

Calle Borgoño 361
Trujillo
Teléfono: (044) 60-2000

SEDE SAN JUAN DE LURIGANCHO

Av. Próceres de la Independencia 3023-3043
San Juan de Lurigancho – Lima
Teléfono: 633-5555

SEDE SAN MIGUEL

Av. Federico Gallese 847
San Miguel – Lima
Teléfono: 632-4900

SEDE BELLAVISTA

Av. Mariscal Oscar R. Benvides 3866 – 4070
(CC Mall Aventura Plaza)
Bellavista – Callao
Teléfono: 633-5555

SEDE AREQUIPA

Av. Porongoché 500
(CC Mall Aventura Plaza)
Paucarpata - Arequipa
Teléfono: (054) 60-3535