



# Registro a Newsletter

## Objetivo

Desarrollar una aplicación cliente-servidor en Java que funcione como una página web accesible por HTTPS y permita registrar participantes en una newsletter. El registro se realizará a través de un formulario, y los datos se almacenarán en un archivo local de forma segura y concurrente.

---

## Requisitos Funcionales

### 1. Interfaz web con formulario (HTTPS):

- La aplicación debe ofrecer una página web accesible por **HTTPS**.
- La página contendrá un formulario donde el usuario pueda introducir:
  - Nombre del alumno
  - Correo electrónico

### 1. Validación de datos:

- El campo de correo electrónico debe tener un formato válido.
- Si los datos son inválidos o el correo ya está registrado, se debe mostrar un mensaje de error claro y ofrecer la posibilidad de reintentar mostrando el formulario nuevamente.

### 1. Almacenamiento seguro de registros:

- Si los datos son correctos, se debe guardar la información en un archivo de texto llamado newsletter.txt.
- Cada línea del archivo debe seguir el formato: nombre\_alumno:correo@electronico

### 1. Control de concurrencia:

- Se debe garantizar el acceso seguro al archivo cuando múltiples usuarios se registren simultáneamente (uso de synchronized).

### 1. Cifrado del archivo (opcional):

- El archivo newsletter.txt puede almacenarse de forma cifrada usando un algoritmo simétrico como **AES**.
- En este caso, se debe poder cifrar al guardar y descifrar al leer.

### 1. Ruta adicional para consultar registros (opcional):

- Se puede implementar una segunda ruta accesible por navegador (ej: /listado) que muestre una lista con los alumnos registrados hasta el momento.

## Recomendaciones Técnicas

- Utilizar synchronized o bloques synchronized sobre un objeto común para proteger el acceso al archivo.

- Para la validación del correo electrónico, se puede usar una expresión regular o un campo email, aunque solamente se requiere comprobar que no existe en el archivo duplicado.

## Objetivo Final



El sistema debe ser accesible mediante un navegador web, registrar usuarios de forma segura y concurrente, y reflejar adecuadamente tanto los éxitos como los errores durante el registro.



# Las Abejas y el Oso Goloso



## Descripción del problema

Simularemos un sistema de **productor-consumidor** con  *abejas recolectoras de néctar* y un  *oso goloso* que llega para para comer miel.

Las **abejas** actúan como **productores** que depositan néctar en una **colmena como recurso compartido**, mientras que el **oso** es el **consumidor** que se despierta cuando hay suficiente miel para comer.

---



## Objetivo

Implementar una simulación concurrente en Java utilizando hilos, sincronización y control de acceso a un recurso compartido (la colmena) por hilos productores y consumidores.

---



## Reglas del sistema

- La **colmena** (buffer compartido) tiene una **capacidad máxima de 30 unidades de miel**.
- Hay dos **abejas** (hilos productores) que recolectan néctar y depositan 1 unidad de miel en la colmena. Tras depositar, se marchan en busca de más néctar, por lo que tardarán un tiempo aleatorio entre 2 y 5 segundos en recolectar y depositar.
- Las abejas **no pueden depositar miel** si la colmena está llena.
- El **oso** (hilo consumidor):
  - Se despierta y solo puede consumir de la colmena **solo cuando hay al menos 3 unidades de miel**.
  - Se **come toda la miel** acumulada y luego se marcha, volviendo pasados unos segundos (entre 2 y 15 segundos) para volver a comer.
  - Repite esto hasta que haya comido **5 veces**.
  - Tras la **quinta comilona**, el oso queda satisfecho y se va a **hibernar**.
  - Cuando el oso hiberna, las abejas **se retiran tras rellenar la colmena** y el programa **termina ordenadamente**.
- El acceso a la colmena debe estar correctamente **sincronizado** usando `synchronized`, `wait()` y `notifyAll()`.

---

## Requisitos técnicos

- Utilizar clases que implementen Thread o Runnable para modelar a las abejas y al oso.
  - Usar métodos sincronizados para controlar el acceso al recurso compartido (Colmena).
  - Implementar correctamente las señales de espera y notificación entre hilos (wait(), notifyAll()).
- 

## Posible salida por consola

```
--- Abeja-3 trajo néctar. Miel en el panal: 10
--- Abeja-3 trajo néctar. Miel en el panal: 11
>>> Comilona del oso #3. Miel en la panza: 40
--- Abeja-2 trajo néctar. Miel en el panal: 1
--- Abeja-3 trajo néctar. Miel en el panal: 2
--- Abeja-1 trajo néctar. Miel en el panal: 3
--- Abeja-2 trajo néctar. Miel en el panal: 4
--- Abeja-1 trajo néctar. Miel en el panal: 5
--- Abeja-3 trajo néctar. Miel en el panal: 6
--- Abeja-2 trajo néctar. Miel en el panal: 7
--- Abeja-1 trajo néctar. Miel en el panal: 8
--- Abeja-3 trajo néctar. Miel en el panal: 9
>>> Comilona del oso #4. Miel en la panza: 49
--- Abeja-2 trajo néctar. Miel en el panal: 1
--- Abeja-1 trajo néctar. Miel en el panal: 2
--- Abeja-3 trajo néctar. Miel en el panal: 3
--- Abeja-1 trajo néctar. Miel en el panal: 4
>>> Comilona del oso #5. Miel en la panza: 53
>>> Oso lleno hasta arriba se marcha a hibernar una larga temporada.
Abeja-3 ve que el panal esta cerrado y se marcha.
Abeja-2 ve que el panal esta cerrado y se marcha.
Abeja-1 ve que el panal esta cerrado y se marcha.
Panal de abejas cerrado. Fin del dia.
BUILD SUCCESSFUL (total time: 42 seconds)
```

---

## Objetivos de aprendizaje

- Comprender y aplicar el patrón **productor-consumidor**.
- Usar correctamente la **sincronización de hilos** en Java (wait() / notifyAll()).
- Diseñar una solución concurrente que **finalice de forma controlada**.
- Manejar múltiples hilos con condiciones de parada bien definidas.



# Guardia Civil: Operación Choricillo

En este ejercicio te pondrás en la piel de la **Benemérita** en una jornada de patrulla, donde no paran de detener maleantes con apodos pintorescos, y el juzgado hace lo que puede por impartir justicia.

---




## Objetivo

Simular mediante sockets TCP un sistema distribuido donde:

- Varias patrullas (clientes) hacen detenidos y los entregan en los calabozos para ser juzgados (servidor), imprimiendo por su consola la detención.
  - Cada patrulla detiene a personajes de lo más variopintos (los alias se pueden seleccionar aleatoriamente de una lista de nombres que quieras, a ser posible mote marroneros/barriobajeros).
  - El Juzgado (servidor) recibe estos nombres y los almacena temporalmente en un calabozo.
  - Cada cierto intervalo aleatorio, el juzgado juzga a un detenido
    - Si es culpable: **directo al talego**.
    - Si es inocente: **a la calle, pero fichado**.
  - El servidor imprime por su consola cada juicio. Y al terminar las patrullas, muestra un menú con opciones.
- 

## Requisitos del Proyecto

### Ficheros principales (como mínimo):

-  Juzgado.java
-  Patrulla.java
-  Lanzador.java

### 1. Juzgado.java

- Escucha conexiones de clientes patrulla.
- Almacena los nombres recibidos en una cola.
- Cada pocos segundos (intervalo aleatorio), extrae un detenido de calabozos del juzgado y decide su destino.

- Muestra por su consola algo como:

```
Juzgando a 'El Gasofa'... Resultado: LIBERTAD con cargos.  
Juzgando a 'La Loli'... Resultado: PRISIÓN preventiva.
```

## 2. Patrulla.java

- Envían un *nombre de detenido* elegido aleatoriamente de un array por ejemplo (alias de barrio como "El Chispas", "La Pili", "El Topo").
- Muestran por consola:

```
[PATRULLA 2] Detenido 'El Chispas' detenido y puesto a disposición judicial.
```

## 3. Lanzador.java

- Proveerá un menú con opciones:

```
1. Seguir patrullando  
2. Terminar turno  
3. Ver estadísticas
```

- Si seleccionamos ver estadísticas, mostraremos el **número total de detenidos puestos a disposición judicial, ingresados en prisión y puesto en libertad** con cargos.
- Si seleccionamos seguir patrullando, indicamos el numero de patrullas del turno.
- Si se selecciona 'terminar turno', se cerrará el programa lanzador.