En un objeto de la clase CMensaje se registra **un mensaje y sólo uno**, que deja un hilo de la clase Productor, dependiendo de un entero que el hilo recibe.

Este mensaje será "consumido" por un hilo de la clase Consumidor.

Nos da igual qué hilo Productor deja el mensaje y qué hilo Consumidor lo lee. Pero ningún mensaje se puede "perder".

**Podemos tener lanzados n hilos Productor y x hilos Consumidor, pero nos tenemos que asegurar, que no se pierda ningún mensaje, es decir, mensaje producido tiene que ser consumido. Además consumido una sola vez.**

Recordad, los hilos tanto productor como consumidor están ejecutándose al mismo tiempo,

"simultánea/concurrentemente si más hilos que UCP".

Los hilos Consumidor están todo el tiempo intentando acceder al objeto mensaje, objeto de la clase CMensaje, ejecutando su método obtener(). Este método está definido como una sección crítica en la clase que define al objeto, de modo que, para ejecutarlo, el hilo tiene que adquirir el monitor del objeto, con esto impedimos que dos hilos consumidores lean al mismo tiempo el mismo mensaje y que cualquier hilo productor genere un nuevo mensaje al mismo tiempo que el consumidor lo lee.

Los hilos Productor están intentando todo el tiempo, cada msegs, (con eso tenemos aquí un ejemplo de modificar ritmo de ejecución bloqueando el hilo, durmiéndolo cada msegs), registrar un mensaje accediendo al objeto de tipo CMensaje e intentando ejecutar su sección crítica almacenar definida en la clase a la que pertenece el objeto.

En el hilo primario creamos el objeto y lanzamos un hilo Productor y otro hilo Consumidor,

En este primer ejemplo sólo lanzamos un hilo de cada clase, para que verifiquemos que aun así el código no está bien diseñado.

El primer hilo de tipo Productor, recibe objeto mensaje

El segundo hilo lanzado, hilo de tipo Consumidor, recibe **MISMO** objeto, ambos hilos acceden al mismo recurso/objeto, son hilos cooperantes, pero ADEMÁS **realizan tareas distintas** ¡!!.

Los hilos son objetos de clases distintas, por tanto, realizan distintas tareas sobre el mismo objeto.

En la versión 0 (errónea), nos equivocamos y no contemplamos todas las tareas que debe implementar el programador en su diseño. Para demostrar que:

ESTA VEZ ES NECESARIO, ADEMÁS DE IMPLEMENTAR LA SINCRONIZACIÓN, IMPLEMENTAR UNA TAREA DE SECUENCIACIÓN...

Tenemos que, además de sincronizar los hilos (estableciendo secciones críticas) diseñar código para secuenciar las tareas de cada clase

--->obtener esta secuencia PRODUCTOR-CONSUMIDOR-PRODUCTOR-CONSUMIDOR

LA TAREA DE SECUENCIACIÓN SE TIENE QUE HACER DENTRO DE LA /LAS SECCIONES CRÍTICAS Y CON EL USO DE LOS MÉTODOS wait() y notify(),notifyAll().