|  |  |  |
| --- | --- | --- |
| **Estrategia de sincronización de hilos** | **Ejemplo de uso** | **¿Para qué sirve?** |
| **Monitores (métodos syncronized) sin comunicación entre ellos** | Ejemplo de método sincronizado:    Estos métodos sincronizados deben estar en el recurso compartido. | * **Sincronización de hilos** (todos hacen su trabajo a la misma vez). |
| **Monitores + control manual de las condiciones** = uso de métodos de comunicación entre hilos (wait y notify) | * Wait() y notify() de la clase Object * Cuando se usa wait y notify, no es necesario usar yield() | * Sincronización de hilos * **Control de regiones críticas**. * **Comunicación de hilos**: los hilos hacen su trabajo a la vez, pero se establece una serie de reglas o condiciones que se han de cumplir para que cada hilo pueda trabajar “sin pisarse el trabajo”. |
| **Semáforos** | * Aquire(numPermisos) y reléase(numPermisos)   + El numPermisos indica la cantidad de permisos que se apropia y se liberan. | * Sincronización de hilos * **Control de regiones críticas**. * **Comunicación de hilos**: los hilos hacen su trabajo a la vez, pero se establece una serie de reglas o condiciones que se han de cumplir para que cada hilo pueda trabajar “sin pisarse el trabajo”. * **Ejemplo típico**: Lector-Escritor. |
| **Exchanger** |  | * Sincronización de hilos. * Se utiliza para el **intercambio de información entre dos hilos** de forma controlada. Cuando un hilo llame a exchange, se quedará bloqueado esperando a que otro haga una llamada al mismo método. De esta forma, ambos obtendrán el objeto del otro. * **Ejemplo típico**: Productor-consumidor. El hilo productor rellena una lista o búfer de datos, y otro hilo consumidor los consume. Mientras este los consume, el productor ya está produciendo una nueva lista. De esa forma, productor y consumidor trabajan concurrentemente al mismo tiempo.   + **Utilidad de java.util.concurrent**: BlockingQueue. |
| **CountDownLatch** | En el hilo:    En el main: | * Sincronización de hilos. * Permite que **uno o más threads esperen hasta que otros threads finalicen** su trabajo. * Al constructor se le pasa el **nº de hilos que quedan por finalizar su trabajo**. * **Método await()**: el hilo que lo ejecuta (normalmente el principal), queda a la espera de que finalicen el trabajo todos los hilos, o en su defecto, si se especifica por parámetro un timeout, hasta que pase el tiempo especificado. |
| **CyclicBarrier** |  | * **Es igual que** **CountDownLatch**, pero **cada x hilos que completen su trabajo, permite ejecutar una acción** (por ejemplo, sumas parciales en el caso de un sumatorio). |
| **Executor (Pool de hilos)** |  |  |