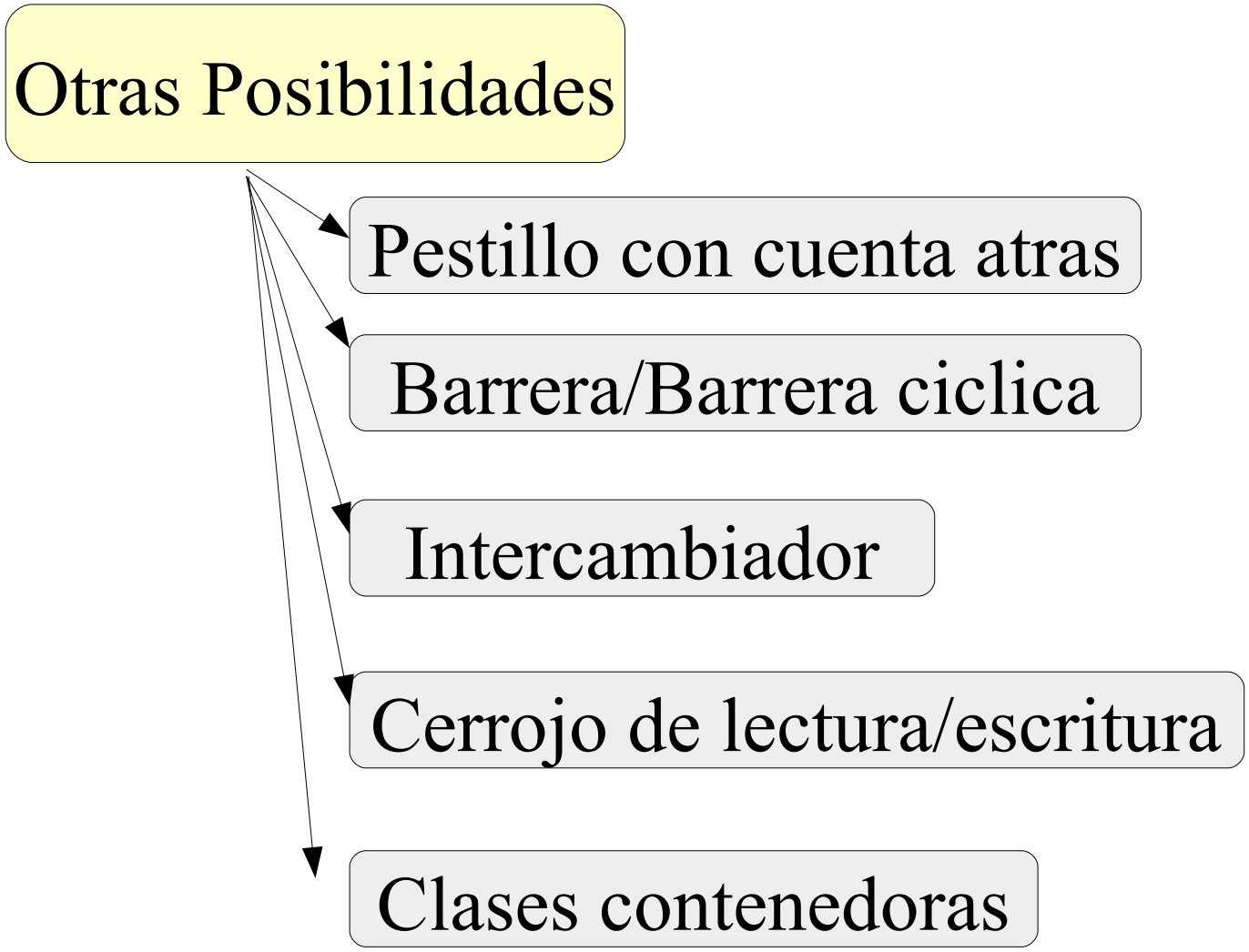
Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue

Programación Concurrente

Otras facilidades para concurrencia

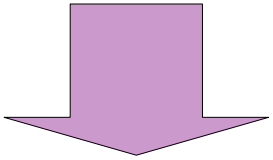
Sincronización



Sincronización de varios hilos

Esperar por un hilo

- Se puede utilizar el método ***join()***



```
// Arranque de hilo
hilo.start();
...
// Espera a que termine
hilo.join();
```

- Esperar por muchos hilos nos implicaría hacer muchas llamadas join(), una por cada hilo activo.
- Para que comiencen varios hilos a la vez o para esperar que todos ellos terminen ...

Pestillo con cuenta atrás

- Mecanismo de sincronización que permite que un conjunto de hilos esperen a un contador que debe llegar a 0
- Es un objeto que se inicializa con un contador N que podemos ir decrementando uno a uno.
- El N indica la cantidad de tareas que pondremos a esperar.
- Cuando el contador llega a cero, liberándose todas las tareas que esperan.
- Son útiles cuando un conjunto fijo de hilos deben esperarse para una tarea en común.
- Operaciones:
  - De espera
  - De decremento

Pestillo con cuenta atrás

- En Java se llaman CountDownLatch
- El objeto CountDownLatch libera los hilos al llegar el contador a 0.
- Operaciones:
  - await()
  - await(timeout, UNIT)
  - countDown()

import java.util.concurrent.CountDownLatch;

Pestillo con cuenta atrás

- Constructor

  CountDownLatch (n)

  n indica la cantidad de veces que debe decrementarse el contador (ejecutarse la operación “countDown”) para que los hilos es espera (por un “await”) puedan continuar

- Ejemplo:

  empezar: countDownLatch(1)

  terminar: countDownLatch(n)

Pestillo con cuenta atrás

Ejemplo:

- empezar: CountDownLatch(1)

  $hilo_1$ --------> empezar.await();

  $hilo_n$ --------> empezar.await();

  main --------> empezar.countDown();

- terminar: countDownLatch(n)

  $hilo_1$ --------> terminar.countDown();

  $hilo_n$ --------> terminar.countDown();

  main --------> terminar.await();

- Ver ejemploCountDownLatch.rar

Barreras

- Mecanismo de sincronización que permite que un conjunto de hilos esperen a llegar a un punto de barrera
- Una barrera de N posiciones retiene los primeros N-1 hilos que llegan. Cuando llega el enésimo, permite que salgan todos. La barrera se rompe y nuevos hilos pasan sin esperar.
- Tambien se puede trabajar con Barrera Ciclica

Barreras

- Mecanismo de sincronización que permite que un conjunto de hilos esperen a llegar a un punto de barrera
  - En Java se llaman CyclicBarrier
  - Soporta un comando run() por punto de corrida
  - El objeto barrera libera los hilos al llegar a la cantidad indicada en el constructor.
- Son útiles cuando un conjunto fijo de hilos deben esperarse para una tarea en común y deben sincronizarse repetidamente.
- La barrera es cíclica porque puede ser reutilizada después que los subprocesos en espera se liberan.
- Se puede considerar que un CountDownLatch es una barrera NO ciclica

CyclicBarrier

- Esta clase se instancia pasándole en el constructor cuántos hilos debe sincronizar.

```
//sincroniza 3 hilos
CyclicBarrier barrera = new CyclicBarrier(3);
```

- Los hilos deben llamar al método await() para esperar por los demas hilos que deben llegar a la barrera

```
public void run () {
  try {
//Se queda bloqueado hasta que los 3 hilos hagan esta llamada
     barrera.await();
  } catch (Exception e) { ...  }
  //código del hilo
}
```

import java.util.concurrent.CyclicBarrier;

Utilización de barreras

```
import java.util.concurrent.CyclicBarrier;
```

```
CyclicBarrier cb = new CyclicBarrier(n)
```

- **await() :** Espera a que todos los hilos definidos hayan entrado a la barrera
- **int await(long timeout, TimeUnit unit) :** Espera a que todos los hilos definidos hayan entrado a la barrera o que pase el tiempo estipulado
- **int getNumberWaiting() :** Retorna el número de hilos que están esperando en la barrera
- **int getParties():** Retorna el número de hilos que requiere esta barrera .
- **boolean isBroken() :** Pregunta si la barrera está quebrada

  es verdadero cuando uno o mas hilos rompen la barrera por interrupción o timeout, o por un reset, o la acción de la barrera falla debido a una excepción.
- **void reset() :** Restablece la barrera al estado inicial

Barrera

Constructores

```
CyclicBarrier(int cantHilosSincronizados)
```

```
CyclicBarrier
(int cantHilosSinc, Runnable accionesBarrera)
```

- Una barrera es un punto de espera a partir del cuál todos los hilos se sincronizan
  - Ningún hilo pasa la barrera hasta que todos los hilos esperados llegan a ella
  - ***Utilidad: sirve para***
    - Unificar resultados parciales
    - Como inicio a la siguiente fase de ejecución simultánea

Ejemplo con barrera

```
//código del Hilo
public class HiloPrueba implements Runnable
      CyclicBarrier barrera;
     HiloPrueba (CyclicBarrier bar){//constructor
           barrera = bar;
       }
     public void run() {
        try { barrera.await(); };
        catch (BrokenBarrierException e) {}
        catch (InterruptedException e) {}
     //codigo a ejecutar cuando se abre barrera…
       System.out.println("sigue ejecutando");
      }       //del run
} // de la clase
```

```
//programa principal
public void main(...) {
  int nHilos = n;
 //numero de hilo que abren barrera
 CyclicBarrier bar = new
                    CyclicBarrier (nHilos);
 new HiloPrueba(bar).start();
```

Intercambiador (exchanger)

- Actúa como un canal síncrono (buffer sin espacio), pero solo soporta un método, tipo *rendezvous*, que combina los efectos de *agregar* y *sacar* del buffer
- La operación, llamada "*exchange*" toma un argumento que representa el objeto ofrecido por un hilo a otro, y retorna el objeto ofrecido por el otro hilo

Dos agentes A y B se sincronizan en 1 punto. A le pasa un dato a B; B le pasa un dato a A.

Intercambiador

- Clase ***Exchanger*** en Java

Ver material disponible en PEDCO, realizado por estudiantes de "Laboratorio de Programación 2015"

import java.util.concurrent.Exchanger;

Clases *Contenedoras*

- Versiones sincronizadas de las principales clases contenedoras
- Disponibles en java.util.concurrent
- Clases:
  - ConcurrentHashMap, CopyOnWriteArrayList
  - ReadWriteLock
- Colas Sincronizadas – Interfaz BlockingQueue
  - *Clases:*
    - *LinkedBlockingQueue, ArrayBlockingQueue, SynchronousQueue, PriorityBlockingQueue,y DelayQueue*

Ver material disponible en PEDCO, realizado por estudiantes de “Laboratorio de Programación 2015”