Universidad de los Andes
Ingeniería de Sistemas y Computación

ISIS 2203 Infraestructura Computacional

Taller de sincronización de *threads*

1) Retome el ejercicio de calcular el máximo de una matriz, con el objeto repartidor de turnos (`turno`). Sin embargo, en esta ocasión el número de *threads* y el tamaño de la matriz no tienen ninguna relación (puede haber más *threads* que filas o viceversa). `turno` reparte turnos de 0 a nFilas-1, y después retorna -1 siempre; es decir, -1 significa que no hay más turnos. En cuanto a los *threads*, iteran pidiendo turno, buscando el máximo de la fila en cuestión y reportándolo, hasta que les llegue el turno -1.

2) Retome el ejercicio del productor-consumidor. Pero, en esta ocasión, no use objetos auxiliares; haga los `wait` y `notify` directamente sobre el *buffer* mismo.

3) "La pasarela". Vamos a modelar la siguiente situación: se tiene una pasarela larga pero estrecha (no caben dos personas a lo ancho), pero circulan personas en las dos direcciones. En consecuencia, en un momento dado puede haber varias personas en la pasarela, pero todas deben ir en la misma dirección.
   Cada persona se anuncia cuando entra y cuando sale; cuando entra informa en qué dirección va.
   Para esto declare una clase `Pasarela`, con métodos: `entrar( direccion )` y `salir( )`.
   Las personas son modeladas por varios *threads* que invocan a `entrar` (en alguna dirección) y después a `salir`.

Nota: para probar sus programas puede usar los siguientes mecanismos:

- `System.out.println(`*string*`)`: para informar por cuáles puntos del código pasa el *thread* y, eventualmente, cuánto valían las variables en esos puntos. Permite verificar si las secuencias de ejecución son válidas, qué casos se han probado y si el estado (las variables) es coherente con el punto de ejecución.
- `assert` *expresión lógica* `:` *string* : si la expresión lógica es falsa, imprime un mensaje de error con la *string*. Permite verificar si el estado (las variables) es coherente con el punto de ejecución. Para usar `assert` es necesario activar las aserciones (en `Run Configurations` seleccionar la aplicación, en esta la pestaña `Arguments`, y en `VM arguments` escribir `-ea`).
- `sleep( `*#milisegundos*` )`: duerme al *thread* por el número de milisegundos indicado. Puede servir para frenar un proceso con respecto a otro (para que vaya más lento).
- `yield( )`: el *thread* cede el procesador (deja de ejecutar). Puede servir para forzar la entrada de otro *thread* en un cierto punto.