**Escuela Colombiana de Ingeniería**

**Arquitecturas de Software – ARSW  
Solución de preguntas del Laboratorio**

**Autores:**Danilo Alejandro Villarraga Guerrero  
Jairo Gonzales

**Ejercicio – programación concurrente, condiciones de carrera y sincronización de hilos. EJERCICIO INDIVIDUAL O EN PAREJAS.**

**Parte I – Antes de terminar la clase.**

Control de hilos con wait/notify. Productor/consumidor.

1. Revise el funcionamiento del programa y ejecútelo. Mientras esto ocurren, ejecute jVisualVM y revise el consumo de CPU del proceso correspondiente. ¿A qué se debe este consumo?, cual es la clase responsable?
2. Haga los ajustes necesarios para que la solución use más eficientemente la CPU, teniendo en cuenta que -por ahora- la producción es lenta y el consumo es rápido. Verifique con JVisualVM que el consumo de CPU se reduzca.
3. Haga que ahora el productor produzca muy rápido, y el consumidor consuma lento. Teniendo en cuenta que el productor conoce un límite de Stock (cuantos elementos debería tener, a lo sumo en la cola), haga que dicho límite se respete. Revise el API de la colección usada como cola para ver cómo garantizar que dicho límite no se supere. Verifique que, al poner un límite pequeño para el 'stock', no haya consumo alto de CPU ni errores.

**Parte II. – Avance para el martes, antes de clase.**

Sincronización y Dead-Locks.

[](https://camo.githubusercontent.com/d7f7294e30cc5391d42016981647d208593dd57c/687474703a2f2f66696c65732e6578706c6f736d2e6e65742f636f6d6963732f4d6174742f42756d6d65642d666f72657665722e706e67)

1. Revise el programa “highlander-simulator”, dispuesto en el paquete edu.eci.arsw.highlandersim. Este es un juego en el que:
   * Se tienen N jugadores inmortales.
   * Cada jugador conoce a los N-1 jugador restantes.
   * Cada jugador, permanentemente, ataca a algún otro inmortal. El que primero ataca le resta M puntos de vida a su contrincante, y aumenta en esta misma cantidad sus propios puntos de vida.
   * El juego podría nunca tener un único ganador. Lo más probable es que al final sólo queden dos, peleando indefinidamente quitando y sumando puntos de vida.
2. Revise el código e identifique cómo se implemento la funcionalidad antes indicada. Dada la intención del juego, un invariante debería ser que la sumatoria de los puntos de vida de todos los jugadores siempre sea el mismo(claro está, en un instante de tiempo en el que no esté en proceso una operación de incremento/reducción de tiempo). Para este caso, para N jugadores, cual debería ser este valor?.  
   Este valor corresponde al valor que se le da de vida al jugador, en este caso es definida por defecto con un valor de 100, multiplicado por el numero de jugadores. Ejemplo: 2 jugadores y 100 vida = 200.
3. Ejecute la aplicación y verifique cómo funcionan las opción ‘pause and check’. Se cumple el invariante? No se cumple, ya que el valor que se muestra al oprimir el botón pause and check arroja valores diferentes cada vez, además cada vez mas altos, lo que indica el incumplimiento del invariante.
4. Una primera hipótesis para que se presente la condición de carrera para dicha función (pause and check), es que el programa consulta la lista cuyos valores va a imprimir, a la vez que otros hilos modifican sus valores. Para corregir esto, haga lo que sea necesario para que efectivamente, antes de imprimir los resultados actuales, se pausen todos los demás hilos. Adicionalmente, implemente la opción ‘resume’.
5. Verifique nuevamente el funcionamiento (haga clic muchas veces en el botón). Se cumple o no el invariante?.  
   Al utilizar variables atomicas y ademas sincronizar los hilos, se logra mantener la integridad de los valores durante toda la ejecución del programa, y asi logramos que el invariante se cumpla.
6. Identifique posibles regiones críticas en lo que respecta a la pelea de los inmortales. Implemente una estrategia de bloqueo que evite las condiciones de carrera. Recuerde que, si usted requiere usar dos o más ‘locks’ simultáneamente, puede usar bloques sincronizados anidados:
7. synchronized(locka){
8. synchronized(lockb){
9. …
10. }

}

1. Tras implementar su estrategia, ponga a correr su programa, y ponga atención a si éste se llega a detener. Si es así, use los programas jps y jstack para identificar por qué el programa se detuvo.
2. Plantee una estrategia para corregir el problema antes identificado (puede revisar de nuevo las páginas 25 y 26 de la lectura).
3. Una vez corregido el problema, rectifique que el programa siga funcionando de manera consistente cuando se ejecutan 100, 1000 o 10000 inmortales. Si en estos casos grandes se empieza a incumplir de nuevo el invariante, debe analizar lo realizado en el paso 4.
4. Para finalizar, implemente la opción STOP.