**LABORATORIO 3 ARSW**

Presentado por:

Daniela García Romero

Edwar Lozano Florez

Presentado a:

Diego Andres Triviño Gonzalez

Escuela Colombiana de Ingeniería Julio Garavito

Materia: ARSW

Grupo: 2

15 de febrero de 2023

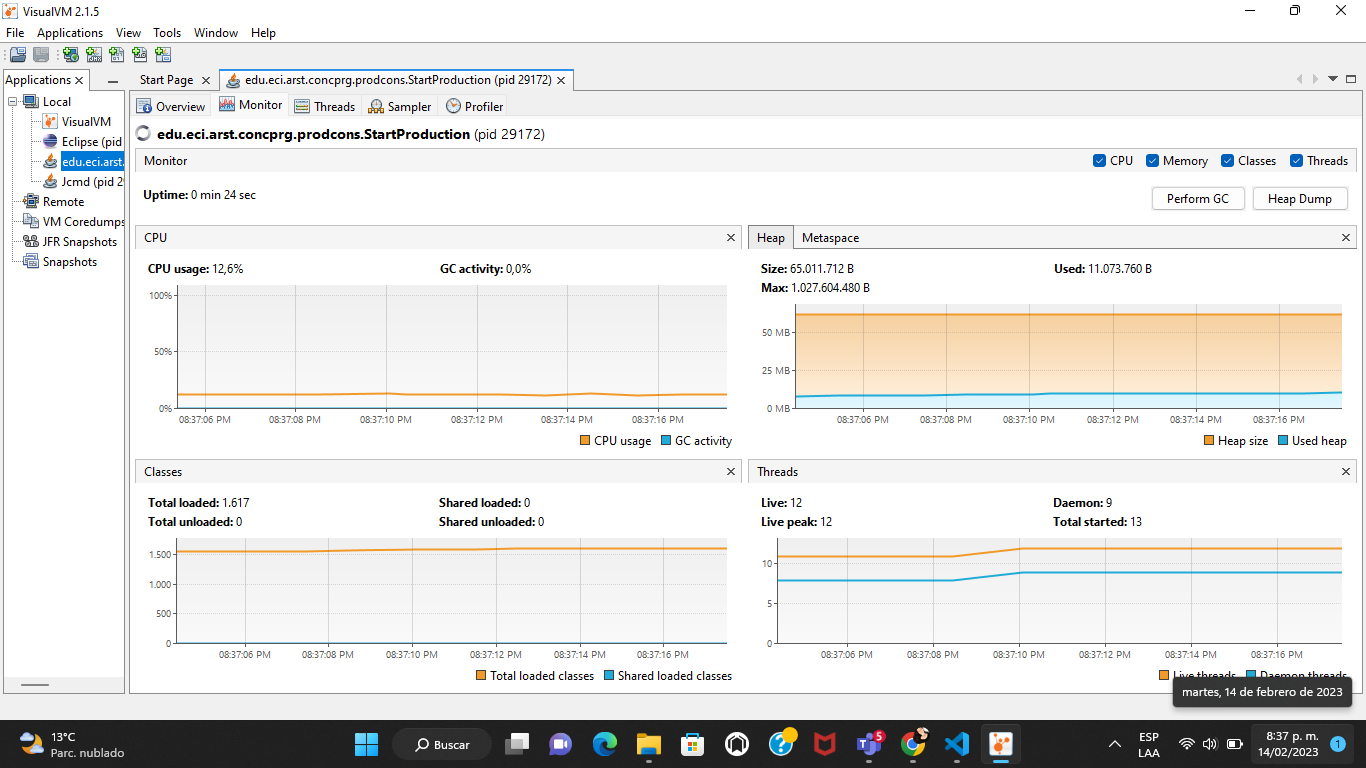
**LABORATORIO 3 ARSW**

**Parte I**

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?

Ejecutamos la clase Start Production:



Se puede observar que hace uso de una gran cantidad de recursos de la CPU, usando el 12,6% de su capacidad. La clase responsable de este consumo es Consumer y Producer.

1. 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.

**Rta:** El consumo de CPU es alto debido a la cantidad de llamadas que realiza la clase Consumidor. Para bajar este consumo limitamos las llamadas haciendo que el programa solo pueda hacer un llamado cada 1 segundo

Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

1. 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.

**Rta:** Teniendo en cuenta el stockLimit de la cola, se aumenta la producción disminuyendo el tiempo en el que se realiza cada petición y asi se aumentara el tiempo en el que consume.

Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

##### Parte II

Sincronización y Dead-Locks.

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.

1. Revise el código e identifique cómo se implementó 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, ¿cuál debería ser este valor?

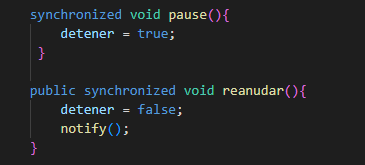
**Rta:** Se crean N inmortales y se agregan a una lista. Para que cada inmortal conozca a quien atacar se usa un random y una variable de verificación del index propio. El valor de la sumatoria de los puntos de vida de todos los jugadores siempre debe ser (N \* 100).

1. Ejecute la aplicación y verifique cómo funcionan la opción ‘pause and check’. ¿Se cumple el invariante?

**Rta:** No se cumple el invariante porque según la función para calcualr el valor del N que definimos en el apartado anterior. El valor de la sumatoria debería ser (7 \* 100) = 700

1. 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’.

Creamos los métodos en la clase Immortal

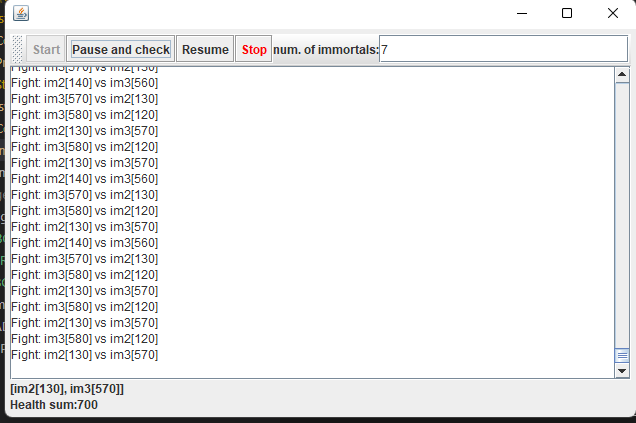


En la clase Thread sincronizamos los hilos haciendo que esperen en caso de que se haya presionado el botón “Pause and check”



1. Verifique nuevamente el funcionamiento (haga clic muchas veces en el botón). ¿Se cumple o no el invariante?

**Rta:** Si se cumple la invariante

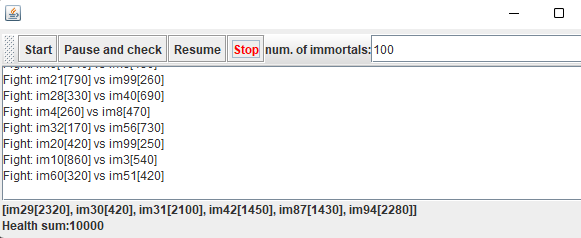


1. Identifique posibles regiones críticas en lo que respecta a la pelea de los inmortales. Implemente una estrategia de bloqueo que evite las condiciones de carrera. Recuerde que, si usted requiere usar dos o más ‘locks’ simultáneamente, puede usar bloques sincronizados anidados:

Captura de pantalla de computadora

Descripción generada automáticamente

1. 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.



Texto

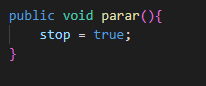
Descripción generada automáticamente

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

1. Para finalizar, implemente la opción STOP.

Creamos el método parar en la clase Immortal



Creamos el botón en la clase ControlFrame

