Laboratorio 3 ARSW 2023-2

Johan Sebastian Garcia Martinez

Ejercicio - Programación concurrente, condiciones de carrera y sincronización de hilos

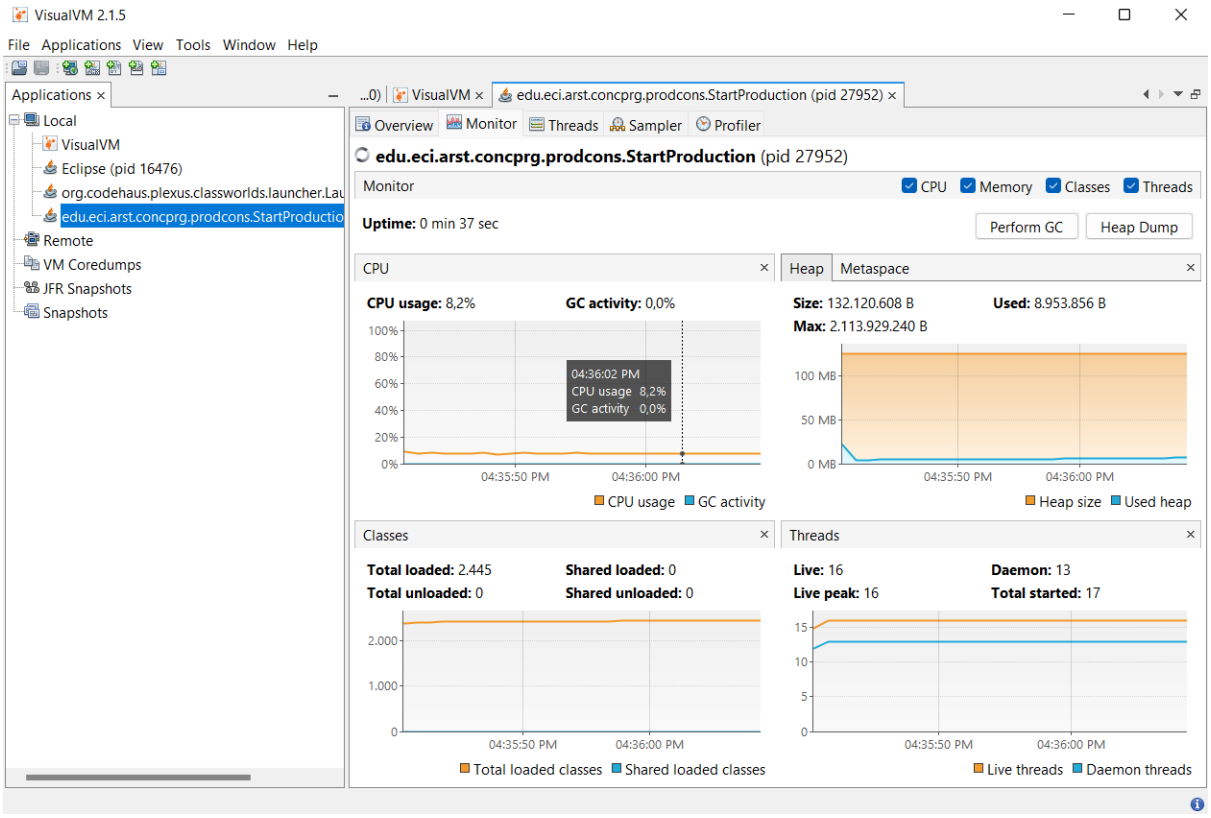
Parte 1.
Control de hilos con wait/notify. Productor/Consumidor

Revise el funcionamiento del programa y ejecútalo. Mientras esto ocurre, ejecute VisualVM y revise el consumo de CPU del proceso correspondiente. A qué se debe este consumo?, ¿cuál es la clase responsable?

Para el funcionamiento del programa podremos determinar que existe una clase o hilo principal llamada “StartProduction”, la cual orquesta el funcionamiento de las otras dos clases o hilos llamados, “productor” y “consumidor”, y lo que hace es iniciar al consumidor para que ejecute sus procesos y luego da una pausa de 5 segundos por ejecución y luego inicia al consumidor para que realice sus actividades

Una vez dentro de las actividades de cada clase podemos identificar que la principal función del productor es generar un numero random, imprimirlo en pantalla y añadirlo a la cola donde se encuentran todos los números generados, y para la clase consumidor es extraer los números de la misma cola y mostrarlos en pantalla.

LAS RESPUESTAS A LAS PREGUNTAS SE ENCUENTRAN EN EL ARCHIVO RESPUESTAS.TXT



2.

Las modificaciones realizadas a las clases para disminuir el consumo de CPU del proceso.

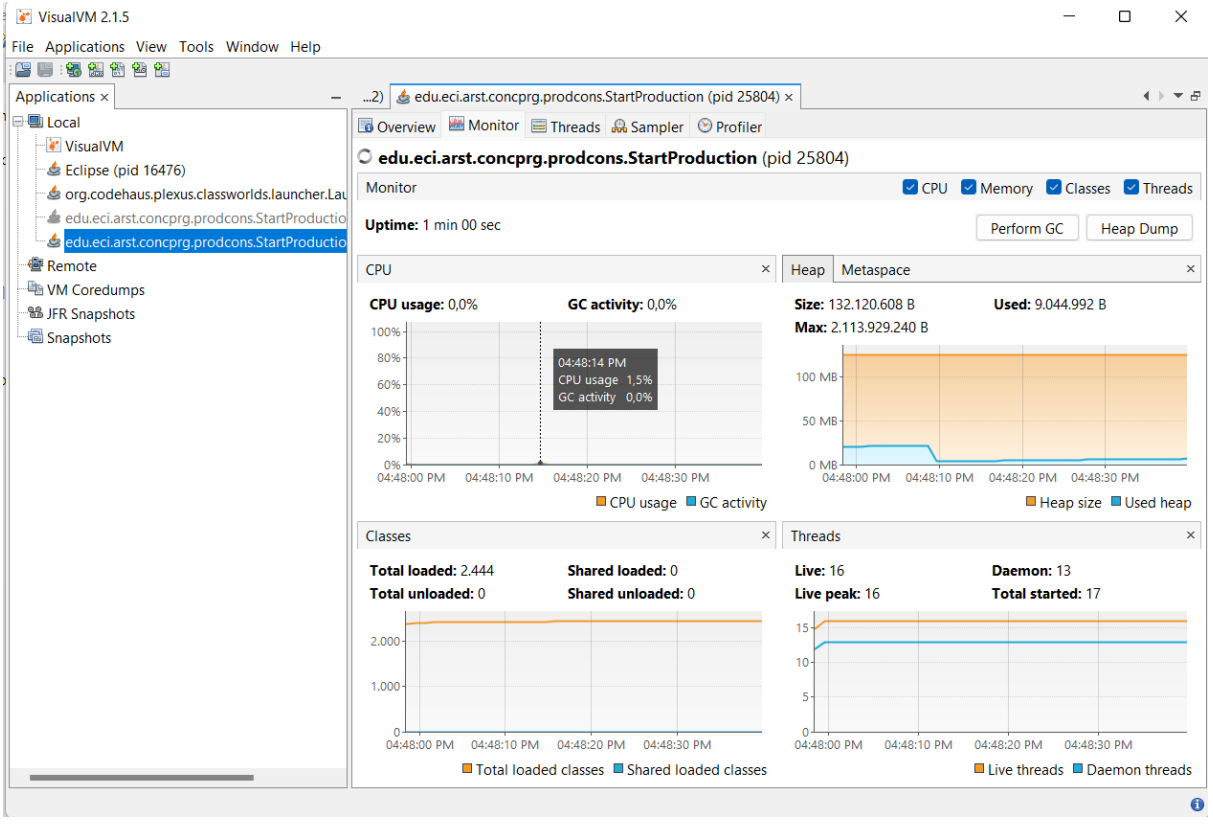
Clase productor

```
@Override
public void run() {
    while (true) {

        dataSeed = dataSeed + rand.nextInt(100);
        System.out.println("Producer added " + dataSeed);
        queue.add(dataSeed);

        synchronized (queue) {
            queue.notifyAll();
        }
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            Logger.getLogger(Producer.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
```

Clase consumidor

```
@Override
public void run() {
    while (true) {
        while(queue.size()==0) {
            synchronized(queue) {
                try {
                    queue.wait();
                }catch(InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
        }
        if (queue.size() > 0) {
            int elem=queue.poll();
            System.out.println("Consumer consumes "+elem);
        }

    }
```

Análisis en VisualVM del proceso después de los ajustes



3.

Para lograr que se respete el límite de producción que se determinó para el stock es necesario, ya que el productor produce más rápido de lo que el consumidor consume, agregar una condición para el productor que cuando llegue al límite este se detenga para darle que se cumpla

Implementación en la clase productor

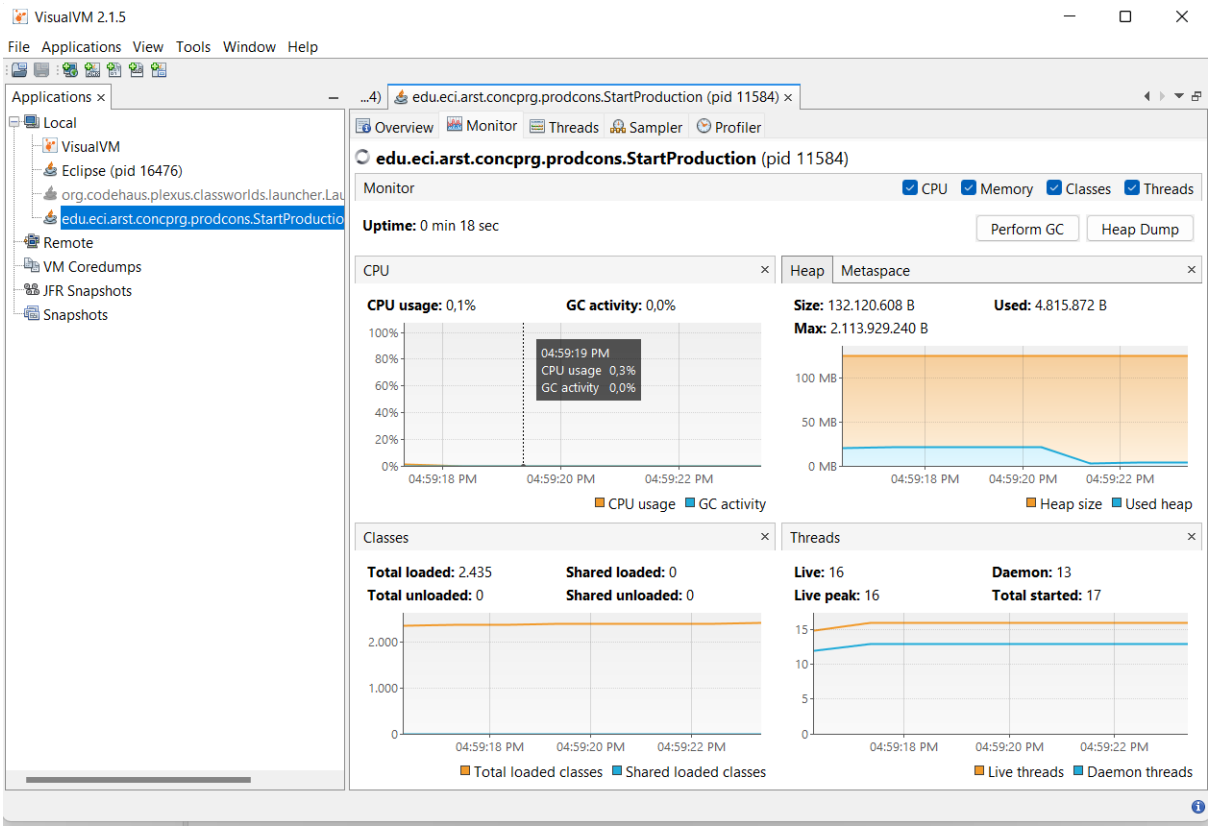
```
@Override
public void run() {
    while (true) {

        while(stockLimit < queue.size()) {
            try {
                queue.wait();
            }catch(InterruptedException e){
                throw new RuntimeException(e);
            }
        }

        dataSeed = dataSeed + rand.nextInt(100);
        System.out.println("Producer added " + dataSeed);
        queue.add(dataSeed);

        synchronized (queue) {
            queue.notifyAll();
        }
        try {
            Thread.sleep(1000);
```

Si el stock se cumplió, cada vez que el consumidor extraiga un elemento de la cola, se le debe notificar al productor que puede seguir produciendo

Implementación en la clase Consumidor

```
        if (queue.size() > 0) {
            int elem=queue.poll();
            System.out.println("Consumer consumes "+elem);
        }

        synchronized(queue) {
            queue.notifyAll();
        }

        try {
            Thread.sleep(1000);
        }catch(InterruptedException ex) {
            Logger.getLogger(Producer.class.getName()).log(Level.SEVERE,null,ex);

        }
    }
```

Rendimiento del CPU



Parte 3. Sincronización y Deadlocks

1. Revise el programa de highlander-simulator

- Toda la configuración de los jugadores inmortales se encuentra en el método “setUpInmortals” de la clase de ControlFrame, donde se toma el valor de las variables las cuales se definen de la siguiente manera:
  - DEFAULT_IMMORTAL_HEALTH: Salud del jugador inmortal
  - DEFAULT_DAMAGE_VALUE: Daño del ataque de cada jugador.
  los cuales se encuentran como variables globales y luego encontramos la cantidad de jugadores dentro de la variable “numOfImmortals” directamente en el constructor de la clase

- Se tienen N Jugadores inmortales los cuales se encuentran definidos en la variable “numOfImmortals”

```
numOfImmortals = new JTextField();
numOfImmortals.setText("7");
toolBar.add(numOfImmortals);
numOfImmortals.setColumns(10);
```

- Cada jugador conoce a los N-1 jugadores restantes, esto se debe a que en el constructor de cada jugador inmortal, se envía como parámetro el LinkedList con la información de todos los jugadores inmortales que le preceden

```
public Immortal(String name, List<Immortal> immortalsPopulation, int health, int defaultDamageValue, ImmortalUpdateReportCallback ucb) {
    super(name);
    this.updateCallback=ucb;
    this.name = name;
    this.immortalsPopulation = immortalsPopulation;
    this.health = health;
    this.defaultDamageValue=defaultDamageValue;
}
```

- cada jugador ataca permanentemente a otro inmortal seleccionado de manera random, con un número que se genera por método random dentro del método run del hilo inmortal y el ataque está definido dentro del método “fight” de la misma clase hilo, al cual le llega como parámetro el jugador al cual va a realizar su ataque, el valor de “defaultDamageValue” se le resta a la vida del jugador atacado y se le suma a la vida al jugador que ataca

```
public void fight(Immortal i2) {

    if (i2.getHealth() > 0) {
        i2.changeHealth(i2.getHealth() - defaultDamageValue);
        this.health += defaultDamageValue;
        updateCallback.processReport("Fight: " + this + " vs " + i2+"\n");
    } else {
        updateCallback.processReport(this + " says:" + i2 + " is already dead!\n");
    }

}
```

2. Dada la implementación del juego se puede determinar una condición de carrera que se puede definir como una colisión de eventos donde varios hilos entran a modificar una misma variable, por lo que no es thread-safe, esto debería controlarse con sincronización para poder cumplir el invariante definido como:

   Invariante: La sumatoria de la salud de todos los jugadores siempre debe ser la misma ya que el ataque resta tantos puntos de vida a la víctima cómo suman al atacante.

   El valor de **N** para el invariante sería el producto de la cantidad de jugadores multiplicado por la salud definida inicialmente para cada uno, en un momento donde no se encuentren haciendo operaciones de incremento/reducción en la variable de salud.

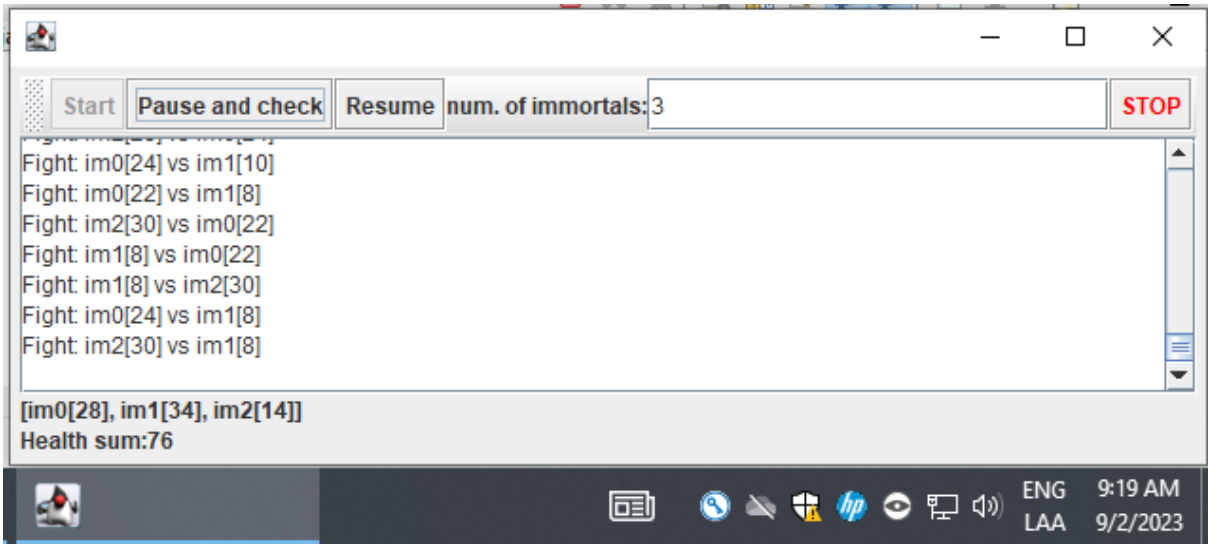
   En el código, el valor **N** para la sumatoria de las vidas de todos los jugadores está dado de la siguiente manera:

   numOfInmortals * DEFAULT_IMMORTAL_HEALTH = N

   como la variable “numOfImmortals” es un string, vamos a obtener su valor con el siguiente código:

   Integer.parseInt(numOfImmortals.getText());

3. Ejecute la aplicación y verifique cómo funciona la opción de “pause and check”:



El valor determinado para las variables de este ejemplo fueron:

- Default_immortal_health = 10
- Default_damage_value = 2
- numOfImmortals = 3

Por lo tanto el valor del invariante **N** que se puede ver reflejado en el label como **Health sum:** debería ser 3*10 , pero como podemos observar en este caso nos muestra 76, esto nos muestra que el invariante no se está cumpliendo.

4. 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’.

```
public void fight(Immortal i2) {
    synchronized(this){
        synchronized(i2) {
            if (i2.getHealth() > 0) {
                i2.changeHealth(i2.getHealth() - defaultDamageValue);
                this.health += defaultDamageValue;
                updateCallback.processReport("Fight: " + this + " vs " + i2+"\n");
            } else {
                updateCallback.processReport(this + " says:" + i2 + " is already dead!\n
            }
        }
    }

}
```