Programación de Servicios y Procesos. 2017. Primera Evaluación

Ante la amenaza que Corea del Norte supone, el ejército de los EEUU ha decido crear un sistema de lanzamiento de misiles global.

Así la clase principal Norad (North American Aerospace Defense Command) será la encargada de generar los misiles (Clase Misil).

Como cada misil funciona autónomamente, será un hilo de ejecución. La creación de los hilos se realizará generando un array de hilos dependiendo de una variable, por lo que modificando dicho valor podemos cambiar el número de hilos a crear (misiles).

```
Misiles [] misil = new Misiles[maximo_misiles];
```

El hilo de ejecución principal del Norad **debe esperar a que todos los misiles estén operativos**, es decir que todos los hilos sean creados (sincronizar, p.e. mediante CountDownLatch), entonces imprimirá "todos los misiles armados y listos".

Una vez inicializados los misiles, **éstos quedarán durmiendo a la espera de ser activados** (sincronizar, p.e. mediante CountDownLatch) .

Para el **lanzamiento de los misiles** el Norad leerá el teclado esperando una orden, cuando se escriba la palabra "**ataca**r" los misiles se pondrán en ejecución mediante el mecanismo de sincronización concurrente previamente establecido, por ejemplo de tipo CountDownLatch). Para evitar un lanzamiento en falso se utilizará un sistema de doble verificación. Para ello el Norad creará una tubería de comunicación con cada misil (array de pipes).

```
PipedWriter emisor[] = new PipedWriter[maximo_misiles];
```

Por cada tubería se enviará a cada misil la orden ataca, este imprimirá por pantalla "misil i-ésimo disparado". El Norad ahora quedará en pausa hasta que todos los misiles sean disparados y terminen su misión.

Una vez disparado el misil y tras una pausa aleatoria, el **misil generará un valor aleatorio de "fallo" o "acierto" e incrementará la variable global de aciertos** en función del valor generado. Dado que esta variables es accedida concurrentemente, es crítico que únicamente un hilo la incremente simultáneamente. Para ello debemos utilizar un semáforo para la sección crítica.

Finalizado el ataque **el Norad deberá informar del número de misiles lanzados y de los aciertos.**

Nuestra aplicación constará al menos de la siguientes clases (no son necesarias más clases):
1. **Norad**, que es la aplicación principal que inicia todos los hilos e imprime el resultado.
2. **Misiles**, que es la clase que controla el misil.

Criterios de evaluación:

1. Las clases se ha implementado adecuadamente, realizan la función sugerida y los parámetros de las clases son coherentes con su funcionalidad. (2 puntos)

2. La sincronización de los pasos del ataque se ha realizado adecuadamente, utilizando las clases de concurrencia que java proporciona. (2 puntos)

3. La sincronización de la generación del resultado se ha realizado adecuadamente, considerando los problemas de la sección crítica. (2 puntos)

4. La aplicación presenta una lógica coherente y funciona correctamente. (2 puntos)

Entrega del ejercicio

El ejercicio deberá ser entregado en un archivo comprimido con el nombre y apellidos y todos los archivos fuente utilizados.

Para una mejor compresión, se indica detalladamente el **funcionamiento del Norad**

1. Crear hilos, pipes y demás objetos y variables necesarias
2. Inicializar todos los hilos
3. Esperar que todos los hilos se creen y estén listos antes de continuar (SYNC)
4. Crear todos los flujos de comunicación con los hilos (pipes)
5. Leer teclado a la espera de la orden de **atacar**, cuando se escriba esta orden continuamos
6. Despertar a todos los misiles (SYNC)
7. Enviar por cada pipe de comunicación la palabra **ataca**
8. Espera que todos los misiles hagan su trabajo (SYNC)
9. Imprime los resultados

El funcionamiento del **misil** es el siguiente:

1. Inicializar variables, semáforos u lo que sea necesario
2. Esperar que el misil sea despertado (SYNC)
3. Esperar recibir la orden **ataca** por el pipe de comunicaciones
4. Imprimir que se ha disparado
5. Calcular valor aleatorio de acierto/fallo y actualizar la variable de resultados, teniendo en cuenta la concurrencia

Propuesta de mejora

Como propuesta de mejora, se ha visto que la orden ataca se envía por los pipes secuencialmente, desde el misil 0 al i_ésimo. Por ello se ha planteado la creación de una clase bot, encargada de enviar por el pipe la palabra ataca. Estos hilos se crearán después de los misiles y esperarán a ser despertados todos a la vez mediante un mecanismo de sincronización, y serán los encargados de enviar la palabra ataca por el pipe, cada bot comunicará con su misil.

Seguidamente y a modo de ejemplo se muestran los parámetros de las clases Misiles y Bots

```
Misiles(int posi,CountDownLatch armados,CountDownLatch lanzar,CountDownLatch sblanco,PipedWriter emisor, Semaphore semaforo)

Bots(int posi,PrintWriter flujoS,CountDownLatch lanzarBot){
```