U1-AE1 - Actividad evaluable de multiproceso

1 Objetivos

- Comprender el concepto de multiproceso.
- Saber crear múltiples procesos a partir de una aplicación Java.
- Aprender a gestionar la comunicación entre procesos.
- Desarrollar aplicaciones multiproceso para resolución de problemas complejos.

2 Ejercicio Inicial

Como consecuencia del cierre de la administración en Estados Unidos, la NASA ha dejado de recibir fondos y uno de sus departamentos, el Departamento de NEOs (Near-Earth Objects), ha enviado a todo el personal a su casa y han apagado sus centros de datos.

Este departamento era el encargado de rastrear asteroides cercanos y detectar posibles colisiones con la Tierra. Desesperados por seguir con su labor, han lanzado una petición mundial para que desarrolladores de software de todo el mundo les ayuden a seguir con su labor. Nosotros vamos a aportar nuestro granito de arena y ayudaremos implementando su algoritmo de detección en nuestro modesto PC.

A continuación, se expone lo que piden:

El objetivo del programa NEOs Help es que desarrolles una aplicación Java (NEO analyzer) que calcule la probabilidad de que un objeto tipo NEO colisione con la Tierra en los próximos 10 años. La aplicación deberá realizar lo siguiente:

1. Leer la información de un NEO de un fichero de datos CSV (comma separated values) en el que cada línea corresponde a un NEO, con los siguientes parámetros:
    a. Nombre del objeto
    b. Posición (relativa a la tierra)
    c. Velocidad NEO en kilómetros - por segundo relativa al Sol)

    Se proporciona un ejemplo más abajo
2. Calcular la probabilidad de que el NEO colisione con la Tierra.
3. Mostrar como salida las probabilidades de colisión de cada NEO (con 3 decimales) y si la probabilidad es mayor de un 10% lanzar una alerta mundial (vale con un System.err.println). Si no, lanzar un mensaje que transmita tranquilidad.

4. Mostrar por pantalla el tiempo de ejecución total de la aplicación y el tiempo medio deejecución por cada NEO que se ha analizado.

Ejemplo de CSV:

```
Apophis,12,200
Bennu,8,150
Didymos,10,180
Miguelón,7,120
Armageddon,1.1,10.1
Apocalypse,100.5,25.7
Calamity,127.9,33.9
Cataclysm,429.7,51.4
Killer no.1,5.5,99.7
Debacle,1000.5,150.6
Donald Trump,7.6,957.1
Ragnarok,8.9,1231.8
Doom,325.1,1975.4
Death by Rock,432.5,842.5
Ctrl+Alt+Supr,321.8,987.8
Windows Blue Screen,31.8,999.9
```

Para calcular el resultado de la probabilidad de colisión se debe realizar una simulación mediante el siguiente código:

```
double posicionTierra = 1;
double velocidadTierra = 100;
double correccion = 0;

for (int i = 0; i < (10 * 365 * 24 * 60 * 60); i++) {
    posicionNEO = posicionNEO + velocidadNEO * i;
    posicionTierra = posicionTierra + velocidadTierra * i;
    correccion += Math.sin(posicionNEO) * Math.cos(posicionTierra);
}
double resultado = 100 * Math.random() *
        Math.pow(((posicionNEO - posicionTierra) /
                  (posicionNEO + posicionTierra)), 2)
        + correccion % 1;
```

Como existen gran cantidad de NEOs para estudiar y los cálculos son complejos y costosos a nivel computacional, se va a aprovechar la capacidad multiproceso de sus equipos.

En principio, se envía a cada voluntario un archivo csv (*comma separated values*) en los que cada línea contiene con la siguiente información:

- Nombre del asteroide
- Velocidad en km/s
- Posición respecto al sol

Cada voluntario procesará, usando procesado paralelo, un número de NEOs igual al número de cores (o procesadores lógicos) que tiene en su sistema, y ese debería ser el

número de líneas del csv. Si no es así, se procesará igualmente pero sacará una advertencia en pantalla.

El número de cores del sistema se obtiene con:

```
Runtime.getRuntime().availableProcessors();
```

3 Ejercicio de ampliación 1

Ahora se propone una nueva versión (NEO Analyzer 2) para máquinas más rápidas, en el que se pueden analizar tantos NEOs como líneas tenga el archivo, aunque exceda el número de procesadores del sistema.

Inicialmente se analizarán tantos NEOs como cores haya disponibles. Después, en el momento en que un core quede libre, se lanzará el análisis de un nuevo NEO. Así se continuará mientras haya líneas que leer en el archivo, procurando evitar esperas y tiempos muertos y que la CPU esté ocupada al máximo posible.

4 Ejercicio de ampliación 2

Como el funcionamiento a base de “pantallazos” tiene sus límites, se propone que en el Neo Analyzer 3 las alertas se guarden en disco, es decir, el resultado del procesado debe ser un único .txt con la lista de todos los NEOs analizados y su probabilidad de colisión.

5 Restricciones

El paralelismo se debe realizar mediante procesos (clase Runtime o ProcessBuilder y clase Process) sin hilos ni threads. En otras palabras, un proceso Java lanza mediante la máquina virtual Java (JVM) otros procesos Java que se ejecutan en paralelo.

6 Entregables

Se debe entregar el ejercicio inicial, y también los otros dos si da tiempo y se quiere subir nota. En otras palabras:

- Neo Analyzer – obligatorio: cuenta para nota
- Neo Analyzer 2 – opcional para subir nota
- Neo Analyzer 3 – opcional para subir nota

Para cada uno de estos proyectos se enviará un archivo comprimido (zip, rar...) con:

- Archivos .java, uno por clase (no incluir directorios ni proyectos enteros)
- Un archivo de texto (puede ser simplemente un .txt) explicando:
    - En qué sistemas se ha probado (Windows y/o Linux)
    - El tiempo medio de ejecución.
    - Los principales obstáculos encontrados al desarrollar el código y hacer las pruebas, y las correcciones que has tenido que hacer sobre el plan inicial (también se puede indicar en comentarios en el propio código)
    - Limitaciones del programa, si las hubiere

NOTA: Sé que algunos vais a ir directos al Neo Analyzer 3 pero me interesa que hagáis cada una de las versiones para ver el proceso de desarrollo que habéis seguido.

Si durante el desarrollo tenéis dudas graves que os bloqueen podéis enviadme un mensaje de Teams o de correo electrónico.

Se valorará:

- Que el programa funcione y encuentren los NEOs.
- Que cumpla con lo que se especifica en los enunciados
- El interfaz, es decir, que los mensajes al usuario sean limpios y fáciles de entender.
- Organización y simplicidad del código
- Legibilidad