PARTE 3: Ejercicio de programación JAVA

Duración 70 minutos

Este ejercicio comprende **2** actividades (no se especifica un orden de las mismas):

a) Desarrollo de la funcionalidad especificada más abajo
b) Desarrollo de los casos de prueba ("test case", al menos uno) para verificar la corrección de la funcionalidad implementada.

ESCENARIO

"Se le llama seis grados de separación a la hipótesis que intenta probar que cualquiera en la Tierra puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no tiene más de cinco intermediarios (conectando a ambas personas con sólo seis enlaces), algo que se ve representado en la popular frase «el mundo es un pañuelo». La teoría fue inicialmente propuesta en 1930 por el escritor húngaro Frigyes Karinthy en un cuento llamado Chains."

…

"Recogida también en el libro Six Degrees: The Science of a Connected Age del sociólogo Duncan Watts, y que asegura que es posible acceder a cualquier persona del planeta en tan sólo seis «saltos»."

El famoso juego "***6 Grados de Kevin Bacon"*** es una aplicación muy popular de este principio, y consiste en vincular a un actor con Kevin Bacon a través de películas en las que hayan trabajado juntos. El número mínimo de vínculos es el *número de Bacon de* un actor. Por ejemplo, Tom Hanks tiene un número de Bacon de 1, pues trabajó con Kevin Bacon en *Apolo 13*. Sally Field tiene un número de Bacon 2 porque trabajó en *Forest Gump* con Tom Hanks, que a su vez trabajó en *Apolo 13* con Kevin Bacon. Casi todos los actores más conocidos tienen un número de Bacon de 1 o 2. Claro está, este algoritmo puede aplicarse a cualquier actor.

Supongamos que hemos decidido representar estas relaciones mediante un grafo no dirigido, y contamos con una lista de actores y otra de relaciones entre los mismos (la relación entre dos actores directa sería una película).

Deseamos realizar una aplicación que permita, para un actor cualquiera "***ZZ***" representado en nuestra estructura, ***Listar todos los actores cuyo "número de ZZ" sea menor o igual a un valor indicado.***

Esta funcionalidad **DEBE SER IMPLEMENTADA CON EL MENOR ORDEN DEL TIEMPO DE EJECUCIÓN POSIBLE.**

Funcionalidad a desarrollar:

Descargar de la webasignatura el archivo "**Parcial2-2018.zip**" que contiene el Proyecto NETBEANS a ser completado.

Se desea:

1. Generar una estructura apropiada para representar el problema y cargarla a partir de los archivos de entrada indicados. Ver detalles de los archivos de entrada más abajo.
2. Desarrollar una funcionalidad que permita, dado un cierto actor "ZZ" (se indicará en el pizarrón), devolver los datos de todos los actores cuyo "número de ZZ" sea menor o igual a un cierto valor dado
3. Emitir un archivo de salida "salida.txt" con los resultados de las ejecuciones de la operación indicada, con un actor (contacto) por cada línea (se solicitarán 2 ejecuciones), seguido de su "número de ZZ".
4. Implementar las verificaciones básicas que aseguren que las funcionalidades anteriores se pueden ejecutar correctamente para el grafo representado.

De TGrafoNoDirigido

- Collection <TVertice> listarContactos (String nombreActor, int maxSaltos)

De TVertice

- void listarContactos (Collection<TVertice> visitados, int maxSaltos);

Test cases.

Implementa los **Casos de Prueba** necesarios para verificar el correcto funcionamiento de los métodos desarrollados.

ENTREGA

Subir a la webasignatura, en la tarea "**PARCIAL2-PARTE3**" el proyecto completo realizado, más el archivo de salida "**salida.txt**".

ARCHIVOS DE ENTRADA

- "**actores.txt**": lista de actores en la base de datos de películas
- "**en_pelicula.txt**": cada línea representa dos actores que han trabajado en una misma película. Se omite esta última por simplicidad, sustituyendo siempre por un carácter arbitrario.

NOTA IMPORTANTE: LOS ARCHIVOS **PUEDEN** CONTENER ERRORES. SE **DEBEN** IMPLEMENTAR LAS PRECAUCIONES BASICAS PARA EVITAR QUE EL SISTEMA COLAPSE ANTE ERRORES O INCONGRUENCIA DE DATOS.

RUBRICA DE CALIFICACIÓN: se utilizarán los siguientes criterios en la evaluación del trabajo remitido:

1. **EJECUCIÓN: 30%**
   - Correcta lectura de los archivos de datos y creación de las estructuras definidas.
   - Consideraciones de seguridad con respecto a los datos – chequeos de consistencia y corrección realizados
   - Ejecución en tiempo razonable
   - Emisión de los resultados correctos
2. **DESARROLLO 35%**
   - Selección e implementación del método con un algoritmo que tenga el **MENOR ORDEN DEL TIEMPO DE EJECUCION POSIBLE**
   - Estructura de datos utilizada (pertinencia, eficiencia)
   - Cumplimiento de las interfaces publicadas
   - Corrección del método solicitado
   - Implementación de chequeos necesarios para cumplimiento de la funcionalidad requerida
   - Programa principal, generación correcta del archivo de salida.
3. **CALIDAD DEL CÓDIGO (10 %)**
   - Nombres de variables y métodos
   - Aplicación correcta y rigurosa del paradigma de programación orientada a objetos
   - Invocación racional y eficiente de métodos y funciones
   - Encapsulación, modularidad
   - Utilización apropiada de clases de colecciones y genéricos necesarios
4. **PRUEBAS DE UNIDAD 25%.**
   - Calidad de los tests desarrollados, todas las condiciones normales y de borde, se testean todos los métodos, uso del enfoque inductivo en los tests.