**Objetivos**

**Unidad 4: Estructuras y Algoritmos Recursivos**

OE4.1 Emplear el concepto de recursividad como una alternativa a la estructura de control iterativa.

OE4.2 Aplicar la computación recursiva en la solución de problemas de naturaleza inherentemente autocontenida.

OE4.3 Utilizar árboles binarios de búsqueda para representar grupos de objetos que mantienen entre ellos una relación de orden.

OE4.4 Escribir algoritmos recursivos para manipular estructuras de información recursivas y explicar las ventajas que, en este caso, estos algoritmos tienen sobre los algoritmos iterativos.

**Enunciado**

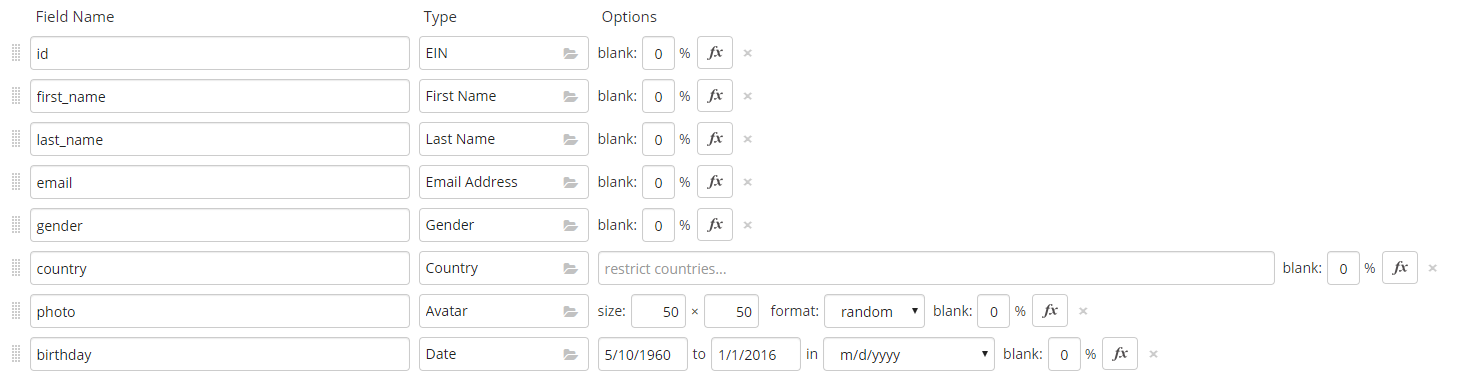
La [IV Copa Panamericana de Voleibol Masculino Sub-21](https://es.wikipedia.org/wiki/Copa_Panamericana_de_Voleibol_Masculino_Sub-21_de_2019) se celebrará del 5 al 11 de mayo de 2019 en Lima, (Perú) y la federación tiene la necesidad de registrar algunos datos de los espectadores que desean asistir al evento y por motivo de algunas labores de logística durante el evento, el personal de registro y control estará permanentemente realizando búsquedas de espectadores y participantes en el software, utilizando como criterio el id de registro del evento.

Usted ha sido elegido para desarrollar un programa que permita leer la información de los posibles asistentes al evento desde un archivo de texto delimitado por comas y cargar la información de manera que se puedan hacer búsquedas a través del identificador del registro del evento.

El programa debe permitir leer el archivo de texto con la información de todos los posibles participantes. Esta información se cargará en una estructura de datos de tipo árbol binario de búsqueda (ABB) cuya relación de orden estará dada por el id de registro del evento. Una vez cargada esta información en el árbol de espectadores, se seleccionará aleatoriamente el 50% de esos espectadores para agregarlos a una lista doblemente enlazada, que será la estructura de datos que almacenará los inscritos al evento o participantes al evento.

Luego de la lectura del archivo y la creación de las dos estructuras de datos: una para los posibles participantes y otra para los inscritos al evento, se debe permitir hacer búsquedas por id, para cada una de las estructuras creadas. Su programa debe indicar, en cualquier caso, si el espectador se encuentra o no entre los buscados así como el tiempo que tardó la búsqueda. Si el espectador fue encontrado, deben visualizarse sus datos en un panel a la izquierda.

Los datos de entrada al programa pueden ser generados utilizando la herramienta web [**https://www.mockaroo.com**](https://www.mockaroo.com/)que permite generar datos aleatorios de forma personalizada. La configuración y orden de los datos generados debe ser la que se muestra a continuación:



Se debe generar el máximo de filas posible para hacer buenas pruebas y el formato en que se debe exportar debe ser CSV el cual delimita los archivos de texto con coma por defecto.

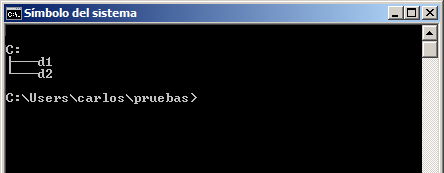


Ya que el máximo es 1000, realice pruebas del tiempo de la consulta para archivos de entrada mas grande como 100000, por ejemplo (generando 100 veces 1000 y uniéndolos). Compare los tiempos de las búsquedas cuando se realizan sobre todos los programadores y los participantes.

La interfaz del programa debe cumplir con los lineamientos:

* Permite cargar los datos de un archivo plano (ejemplo: archivo.csv) y confirmar la correcta carga, la ruta que se entra por teclado puede ser relativa o absoluta, debe especificarle al usuario final cual debe ingresar.
* Buscar espectador por id, debe cuantificar cuánto tiempo tarda la búsqueda y mostrar los datos del espectador (la foto en esta versión no será visualizada).
* Buscar participantes por id, debe cuantificar cuánto tiempo tarda la búsqueda y mostrar los datos del participante (la foto en esta versión no será visualizada).

El programa también contará con una opción que podrá visualizar los datos de las dos estructuras de un país determinado, donde la estructura del árbol binario de búsqueda se verá como un árbol y la lista se visualizará en forma consecutiva. La estructura árbol puede ser vista de la siguiente manera:



Donde c: es la raíz del árbol y d1 y d2 son hijos de c:,para cada opción se visualizará sólo los datos del país seleccionado (puede ser el componente JOptionPane de Java), eso quiere decir, que para la estructura de espectadores el árbol pintado solo debe tener espectadores de la ciudad seleccionada e igual para la estructura de los participantes.

**Entregables.** **1.** Requerimientos. **2.** Diagrama de clases de modelo e interfaz (no generado automáticamente). **3.** Implementación completa de todos los requerimientos en Java. **4.** Diseño de casos de pruebas unitarias de todas las funcionalidades relacionadas con los árboles. **5.** Implementación de todas las pruebas unitarias diseñadas y corriendo perfectamente. **6**. Todos los métodos para administrar las estructuras deben ser implementados usando recursividad. **7**. La estructura de los espectadores deben ser un árboles binarios de búsqueda y los participantes una lista doblemente enlazada, tal como se explicaron en el tema visto en clase y no podrán usar otro tipo de contenedor como array, ArrayList, etc.

**Fecha de Entrega:** martes 15 de octubre de 2019 a las 11:55 p.m. a través de Moodle.

**Integrantes**: el laboratorio debe realizarse individualmente.

**Documentación pruebas**

**Objetivo:** Demostrar que el método busca correctamente los participantes poniendo como prueba dos casos, que buscara un participante que no existe y uno que si existe.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
|
| CopaPanamericana | buscarParticipantePosible  (String) | El escenario inicializa cuatro personajes con los siguientes atributos:  -Personaje uno, nombre: participante, apellido: Uno, Id: 1  -participante dos, nombre: participante, apellido: Dos, Id: 2  -participante tres, nombre: participante, apellido: Tres, id: 3  -participante cuatro, nombre: participante, Apellido: Cuatro, id: 4 |  | Verdadero, el método busca el posible participante correctamente. |
|
|
|
|
|
|
|
|
|
|

**Objetivo:** Probar que el método agrega correctamente un posible participante.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
|
| CopaPanamericana | agregarParticipante  Posible() | El escenario inicializa cuatro participantes con los siguientes atributos:  -participante uno, nombre: participante, apellido: Uno, Id: 1  -participante dos, nombre: participante, apellido: Dos, Id: 2  -Personaje tres, nombre: Personaje, apellido: Tres, id: 3  -Personaje cuatro, nombre: Personaje, Apellido: Cuatro, id: 4 |  | Verdadero, el método agrega correctamente al participante |
|
|
|
|
|
|
|
|
|
|

**Objetivo:** Demostrar que el método busca correctamente los participantes poniendo como prueba dos casos, que buscara un participante que no existe y uno que si existe.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
|
| CopaPanamericana | buscarParticipanteInscrito  (String) | El escenario inicializa cuatro personajes con los siguientes atributos:  -Personaje uno, nombre: participante, apellido: Uno, Id: 1  -participante dos, nombre: participante, apellido: Dos, Id: 2  -participante tres, nombre: participante, apellido: Tres, id: 3  -participante cuatro, nombre: participante, Apellido: Cuatro, id: 4 |  | Verdadero, el método busca el posible participante correctamente. |
|
|
|
|
|
|
|
|
|
|

**Objetivo:** Probar que el método agrega correctamente los participantes inscritos aleatoriamente.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clase** | **Método** | **Escenario** | **Valores de entrada** | **Resultado** |
|
| CopaPanamericana | agregarParticipante  InscritoAleatoriamente() | El escenario inicializa cuatro participantes con los siguientes atributos:  -participante uno, nombre: participante, apellido: Uno, Id: 1  -participante dos, nombre: participante, apellido: Dos, Id: 2  -Personaje tres, nombre: Personaje, apellido: Tres, id: 3  -Personaje cuatro, nombre: Personaje, Apellido: Cuatro, id: 4 |  | Verdadero, el método agrega  Correctamente los participantes inscritos de manera aleatoria. |

**Requerimientos funcionales**

|  |  |
| --- | --- |
| **Nombre:** | R. #1. Cargar de un archivo plano la información de los posibles participantes |
| **Resumen:** | El programa debe leer un archivo plano donde se encuentra la información de todos los posibles participantes a la Copa Panamericana |
| Entradas: | Ruta del archivo |
|  |  |
| Resultados: | Datos cargados con éxito |
|  | No fue posible cargar los datos |

|  |  |
| --- | --- |
| **Nombre:** | R. #2. Seleccionar entre los posibles participantes el 50% de ellos aleatoriamente |
| **Resumen:** | El programa debe seleccionar entre todos los posibles participantes el 50% de ellos para que sean los participantes inscritos. |
| Entradas: |  |
|  |  |
| Resultados: | Participantes seleccionados con éxito |
|  | Ocurrió un error al seleccionar los participantes inscritos |

|  |  |
| --- | --- |
| **Nombre:** | R. #3. Buscar posible participante por ID |
| **Resumen:** | El programa debe permitir al usuario buscar un posible participante por el ID |
| Entradas: | ID del posible participante que se desea buscar |
|  |  |
| Resultados: | Posible participante encontrado con éxito |
|  | No fue posible encontrar el participante con el ID ingresado |

|  |  |
| --- | --- |
| **Nombre:** | R. #4. Buscar un participante inscrito por ID |
| **Resumen:** | El programa debe permitir al usuario buscar un participante inscrito por el ID |
| Entradas: | ID del participante inscrito que se desea buscar |
|  |  |
| Resultados: | Participante inscrito encontrado con éxito |
|  | No fue posible encontrar el participante con el ID ingresado |

**Trazabilidad**

