Informe proyecto Examen Libre Estructuras de datos

Alumno: Juan Cruz Feuilles

LU:100598

Introducción:

Con el objetivo de desarrollar la aplicación solicitada en el proyecto se implemento lo listado a continuación:

- TDA Lista Simplemente Enlazada.

- TDA Cola con prioridad (Heap).

-Comparadores de String (Alfabético y Alfabético “inverso”).

Detalles Implementación:

Para la TDA Lista, en el inciso del proyecto se especifica bien que lo que se desea, es la implementación de la Lista Simplemente Enlazada con un solo centinela al primer elemento. Que significa esto, que para todo elemento de la lista (posición) solo se tendrá referencia a su elemento asociado en el caso de esta aplicación (String) y a la posición siguiente ya sea una posición valida o nula en caso del último elemento o “cola” y también se tendrá acceso a la cabeza de la lista.

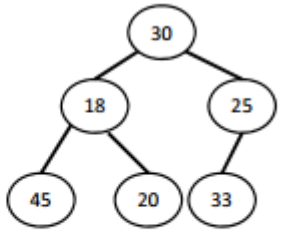
Esto hace que cada vez que se requiera una posición que no sea la cabeza de la lista, esta se deberá recorrer hasta encontrar la posición deseada, este sería el caso de los métodos last() y addLast() por ejemplo. Dicha TDA fue testeada con los testers que brinda la cátedra.

Obs: El único método que quizás no este especificado en el TDA Lista que se realizo es clonarLista(L), que retorna un clon de la lista L, para mantener intacta la lista original y no se modificarla en los métodos invocados sobre la misma realizando un clonar previo.

Para la TDA Cola con Prioridad, la implementación requerida fue la que utiliza Heap, si bien la idea de esta implementación es la utilización de un árbol binario completo, la representación elegida fue usando un arreglo que represente el árbol binario, la representación es tal que dado arreglo ocupado desde la posición 1 a N siendo N un numero especifico de elementos dado por el usuario de la cola, para cada elemento en una posición “i” del arreglo en el intervalo cerrado [1,N] su padre será el elemento ubicado en la posición “i/2” su hermano izquierdo “2i” y su hermano derecho “2i+1”

Ejemplo de árbol representado con arreglo.

Hijo_izquierdo(i) = 2i
Hijo_derecho(i) = 2i+1
Padre(i) = i div 2



	0	1	2	3	4	5	6	7	8	9	10	11	12	13
árbol		30	18	25	45	20	33							

Como detalle de implementación se debe aclarar que el elemento mínimo se ubicara en la posición 1 de arreglo y la posición 0 no se usa

Para el método min() es simplemente retornar el elemento en la posición 1, mientras que el

removeMin() es retornar el elemento en la posición uno y se eliminarla reemplazándolo por el último elemento luego se elimina la última posición para que no se repita, luego se recorre de la raíz(1) hacia abajo haciendo buble(burbujeo) ordenando los hijos.

Insertar(elemento) para insertar un elemento, se inserta al final del arreglo, y parado en la última posición se hace un buble(burbujeo) hacia arriba, recorre comparando el elemento actual(i) y si padre(i/2) si el padre es menor los intercambio y actualizo el valor de (i=i/2), si no se puede intercambiar más quedo ordenado.

HeapSort(L) con la implementación de la cola completa, se crea una cola donde se insertan todos los elementos que se desean ordenar y después se salvan utilizando el método removeMin() utilizando el orden deseado.

Para especificar las implementaciones del Heap se pueden hacer un comparador que compare los elementos según el criterio deseado por el programador, en este caso un orden alfabético. La implementación se implemento utilizando el numero asociado a los ordinales de los elementos a comparar, los cuales se comparaban uno a uno, los caracteres válidos a comparar es el abecedario tanto de mayúsculas y minúsculas incluyendo la letra “ñ” y las vocales acentuadas en castellano.

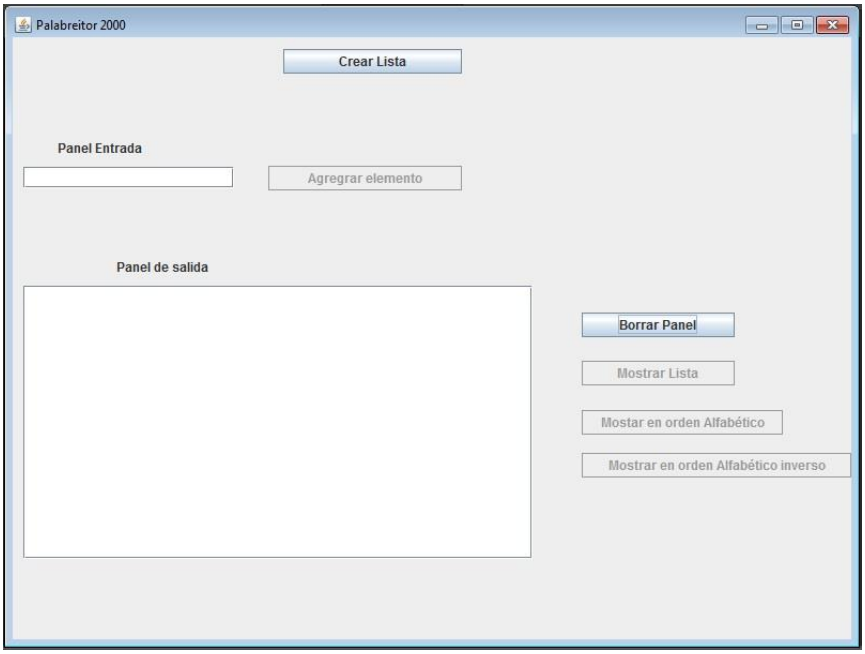
Para la comparación de lo ordinales utilice la representación de números reales doble (double) esto me permitió agregar los caracteres “Ñ” y “ñ” en un valor medio a sus letras vecinas en el abecedario, luego se realizo un corrimiento de manera que

todas las letras se reconozcan como mayúsculas (ordinal de minúscula-32) la vocales acentuadas se agregaron caso a caso a su carácter mayúscula correspondiente. El comparador permite distinguir la precedencia entre una palabra y su prefijo utilizando las longitudes de las cadenas.

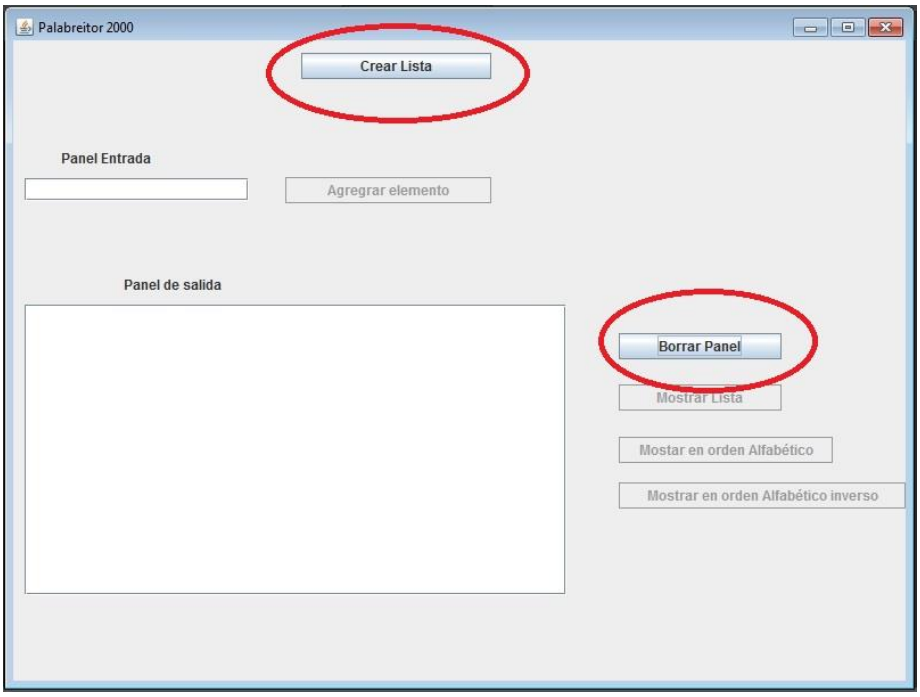
Programa de prueba:

Como se solicito se realizo un programa de prueba que posee una interfaz de usuario que se explicara a continuación.

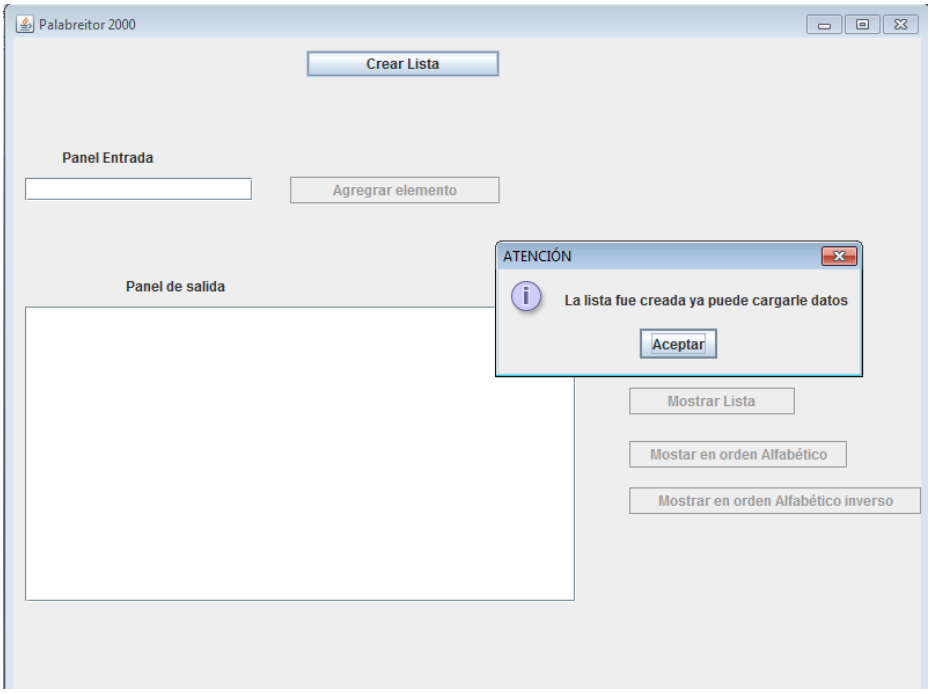
Al iniciar la aplicación se verá la siguiente pantalla.



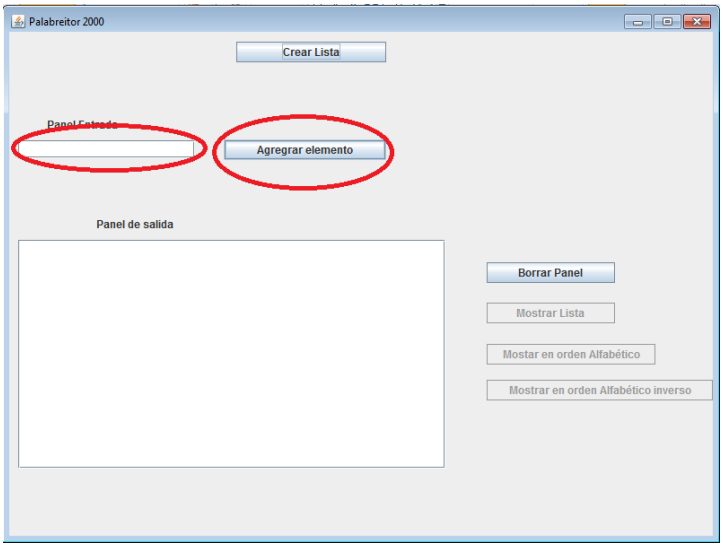
Como se podrá ver solo dos funciones están activas, “Crear Lista que” crea una lista de Strings vacía, y “Borrar Panel” que si el cuadro de texto bajo la etiqueta “Panel salida” tiene elementos permite borrarlos



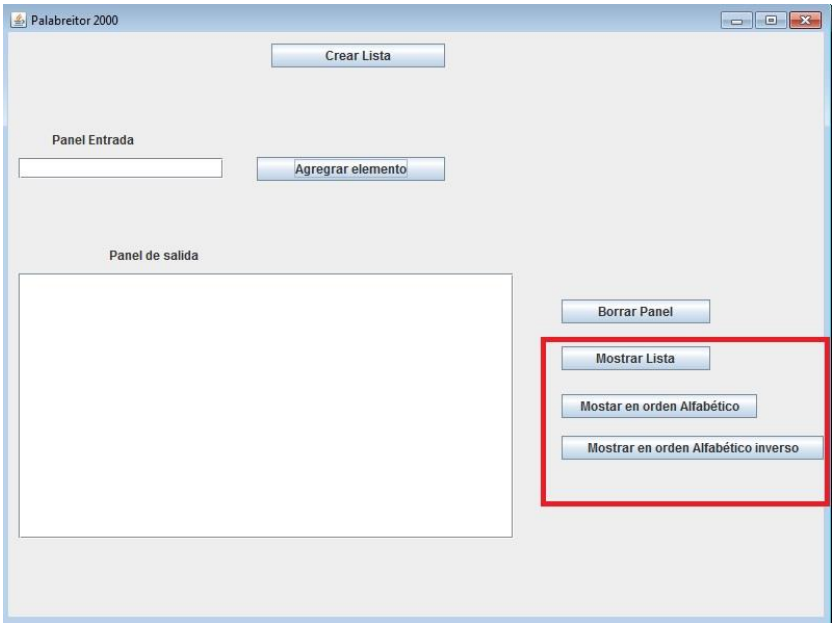
Si se pulsa “Crear Lista” aparecerá el siguiente cartel notificando que la lista se creó y nos habilitara la función ”Agregar elemento”



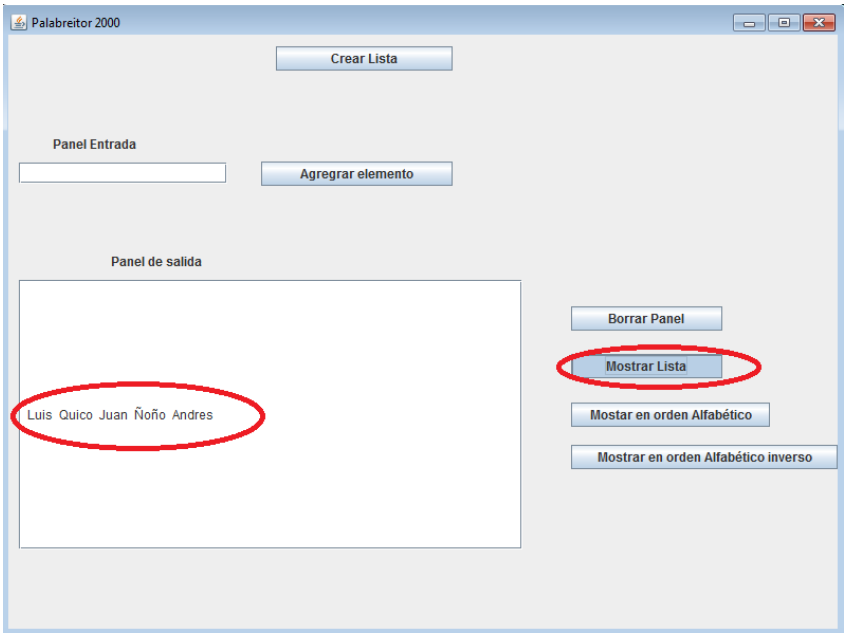
En cuadro de texto ingrese la cadena String que se desea ingresar y “Agregar elemento” lo agrega a la lista creada, agregue tantos como desee. Cuando pulsar “Agregar elemento” se habilitaran las restantes funciones.



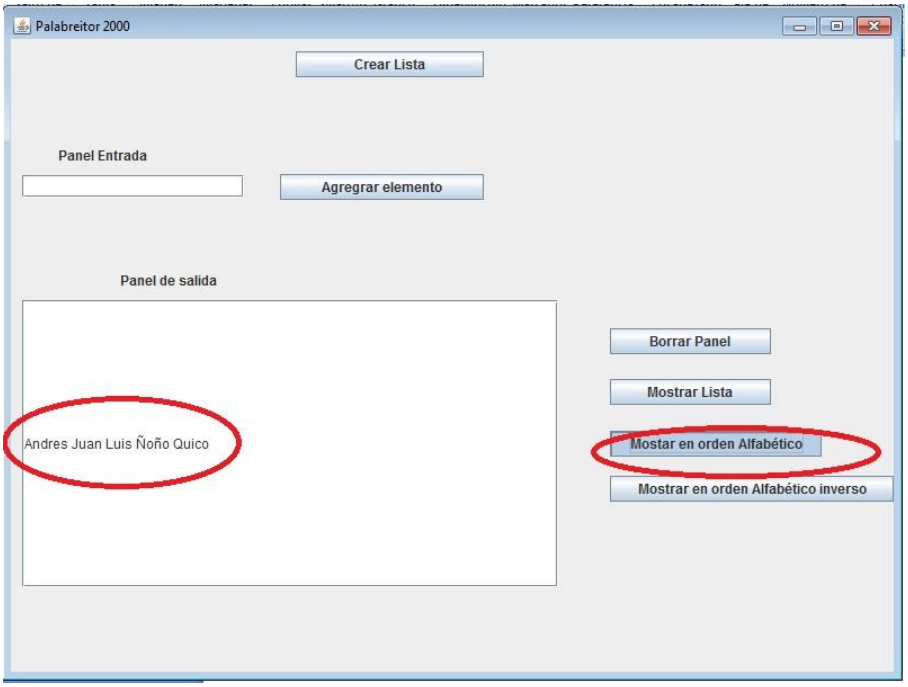
Las restantes funciones son muy intuitivas le recomiendo que ingrese al menos dos elementos para que tenga sentido pedir un orden.



“Mostrar lista” muestra los elementos en el orden que fueron agregados.



“Mostrar en orden Alfabético” muestra los elementos en orden alfabético.



Y como era de esperar “Mostrar en orden Alfabético inverso” muestra lo contrario.