ALGORITMOS

Clase “Compare”

Método compare

#Compara con un alfabeto de caracteres, numeros, letras de minusculas a mayusculas hasta las vocales tildadas en minuscula y en mayusculas.

Recibe dos parametros (objeto1 y objeto2)

Si objeto1 es igual a objeto2 -> retornar 0

Si la longitud del objeto1 es igual a la longitud del objeto2 ->
    Recorrer el rango de la longitud del objeto1 ->
        Si el indice del objeto1 es menor (<) al indice del objeto2 ->
            Retornar -1

        Caso contrario Si ->
            El indice del objeto1 es mayor (>) al indice el objeto2 ->
                Retornar 1

Si la longitud del objeto1 es menor (<) a la longitud del objeto2 ->
    retornar -1

Si la longitud del objeto1 es mayor (>) a la longitud del objeto2 ->
    Retornar 1

Clase “LinkedList”

Método add

#Agregar elementos a la lista.

Si el estado es verdadero

Si el primero es vacio
    Entonces agrega el primer elemento a la lista.

De lo contrario
    Si comparamos el nombre del primer elemento con el nuevo nombre y estos son mayor a 0
        Entonces la lista se comporta como una lista.

    De lo contrario
        El primero ahora es el previo y el siguiente es el actual
        Cuando el valor del actual sea menor (<) que el valor, devuelve -1
            Entonces el actual va antes que el nuevo valor
            Solo se mueve hasta encontrar el mayor
        Cuando el valor del actual sea igual (=) que el valor, devuelve 0
            La diferencia es que en este guarda incluyendo el actual
            El nuevo valor reemplaza al current
        Cuando el valor del actual sea mayor (>) que el valor, devuelve 1
            Entonces el nuevo valor va antes que el current
            Este guarda después del actual

Caso contrario si el estado es falso
Si el primero es vacio
    Entonces agrega el primer elemento a la lista.

Caso contrario
    El primero es el actual y mientras haya un siguiente del actual
    Se crea el nuevo nodo

Método searchInLL

#Busqueda de un elemento, state =0 para devolver el nodo, state = 1 para regresar un boolean

Recibe los parametros (valor y estado)

Hacemos el primero el actual.

Si el estado es cero entonces
    Si el valor del primero es el igual al recibido del parametro ->
        Retornamos el primero.
    Caso contrario
        Mientras haya un siguiente del actual.
            Hacemos el actual en el siguiente del mismo.
            Si el actual es diferente de vacio (*None)*
                Si el valor actual es igual al del parametro ->
                    Retornamos el actual.

Caso contrario Si el estado es igual a uno (1)
    Si el valor del actual es igual al del parametro ->
        Retornamos Verdadero

    Caso contrario
        Mientras haya un siguiente del actual
            Hacemos el actual en el siguiente del mismo.
            Si el valor del actual es igual al del parametro->
                Retornamos un Verdadero.
    Retornamos falso de no encontrar lo que buscamos.

Método Length
#Obtener el tamaño de la lista.

Hacemos al primero el actual.
Si El actual es distinto a vacio (*None*)
    El tamaño inicial es uno (1)
    Mientras haya un siguiente del actual
        Hacemos el actual en el siguiente del mismo.
        Sumamos 1 al tamaño->
    Retornamos el tamaño
Caso contrario->
    Retornamos 0

Método atPositoion

#Posición

Definimos un tamaño usando el método anterior.
Hacemos al primero el actual.
SI el indece es mayor igual al tamaño
    La busqueda esta fuera del rango
    Retornamos -1
Caso contrario
    SI el indice el igual a 0
        Retornamos el actual
    Caso contrario
        Iniciamos un contador en 0
        Mientras haya un siguiente del actual
        Hacemos el actual en el siguiente del mismo.
        Sumamos uno al contador
        Si el contador es igual al index
        Retornamos el actual

Método removeToNormal

#Elimina elementos nodos dado su valor.

Hacemos al primero el actual.
Si el valor del actual es igual al valor del parametro
    El siguiente del actual es el primero ahora.
Caso contrario
    Mientras haya un siguiente del actual
    Hacemos el actual un previo
    Hacemos el actual en el siguiente del mismo.
    Si el valor del actual es igual al valor de parametro
        El siguiente del previo es el siguiente del actual.

Método RemoveForNameAndType

#Remueve un elemento dado el nombre y el tipo de dato de este.

Hacemos un nodo temporal buscando recursivamente por nombre y tipo
Si el nodo temporal es distinto de falso
    Llamamos el método para eliminar elementos del nodo dado su valor
    (removeToNormal).

Método _printToNormal

#Imprime los elementos de la lista

Hacemos al primero el actual.
Si el actual es diferente a vacio (*None*)
    Mientras haya un siguiente del actual
        SI el estado es igual a 1
            Imprimir el actual
        Caso contrario imprimir el nombre del actual.
    Si el estado es igual a 1
        Imprimir el actual
    Caso contrario
        Imprimir el nombre del actual
Caso contrario
    Retornar nada

Método searchItemForNameAndType

#Esta función sera al momento de verificar si puede agregar o no, un elemento si ya existe uno.

#state==0 si solo quiere que retorne boolean, '1' si quiere que retorne el nodo.

Hacemos al primero el actual
Tomamos el tamaño de la lista enlazada
Si el tamaño de la lista es difente de 0
    Repetir en el rango del tamaño de la lista
        Guardamos la posición de cada item
        Si el nombre del item es igual al nombre enviado del parametro y su tipo de dato es igual al tipo de dato enviado del parametro.
            Si su estado es igual a 0
                Retornamos Verdadero
            Si su estado es 1
                Retornamos el item
Retornamos Falso de no cumplir nada de lo anterior.

Método extractForType

Extraemos un item por el tipo ya sea 1) para Directorios y 2) para archivos

Obtenermos el tamaño con el método length
Creamos dos arreglos para almacenar los directorios y los archivos
Creamos un indice de 0 al rango del tamaño de la lista enlazada.
    Almacenamos cada posición en un item
    Si el tipo de dato del item es igual a uno
        Adjuntamos el item al arreglo de directorios
    Si el tipo de dato del item es igual a dos
        Adjuntamos el item al arreglo de archivos
Al terminar el rango del tamaño de la lista establecemos al primero vacio

Retornamos el arreglo de directorios y archivos.

Método SortLL

#Ordenar la lista enlazada

Obtenemos la lista de directorios y archivos del método anterior (extractForType)
Recorremos de 0 al rango de la longitud de la lista de archivos
    Adjuntamos a la lista de directorios el indice de la lista de archivos

Recorremos de 0 al rango de la longitud de la lista de directorios
    Almacenamos los tabulados de la lista de directorios
    Creamos un padre de la lista de direcotorios
    Creamos un hijo de la lista de directorios

    Agregamos a la lista el valor de la lista de directorios, el nombre de la lista de directorios su tipo de dato el padre y los tabulados.
    Realizamos una busqueda recursiva del valor de la lista de directorios para almacenar sus hijos.

Archivo “TreeN”

Su método constructor recibe

Un root (Inicialmente no hay root)
Una lista enlazada para simplemente guardar los nodos del arbol
Un simple contador, para saber cuantos items hay en el arbol.

Método addElementoToTree

Recive los parametros (valor, nombre,padre,tipo de dato)

#Agregar al arbol con sus respectivos parametros.

Si el root es igual a vacio y su padre es igual a 0
    El root es un nodo.
    Se crea una variable global final
    Se crea una variable global final2
    Incrementa el total de nodos en 1
Caso contrario
    Se llama al método addInner (para agregar internamente

Método addInner

Recibe los parametros (Valor, nombre, padre, tipo de dato)

#Agregado Interno, haciendo uso de la función recursiva ‘search’

En el actual se hace una busquerda en el arbol del padre usando el método searchInTree
#Busca y retorna el padre al que le sera agregado el elemento.
Si no tiene hijos
    Los hijos son una lista enlazada
    El root su tabulado es 0

SI Tiene hijos
    Si el actual no tiene hijos
        Los hijos del actual son una lista enlazada
        Suma un nuevo tabulado
        Se agregan hijos al actual
        El contador de Nodos incrementa en 1

    Caso contrario
        Suma un nuevo tabulado
        Se agrega un nuevo hijo

Caso contrario
    Imprime “No se agrego al arbol”

Método searchInTree

Recibe los parametros (valor)

#Llama a la función interna

Retorna el método search InnerInTree

Método searchInnerInTree

Recibe los parametros (root y valor)

Si el item a buscar es el root
    Retorna el root

Si el item no tiene hijos
#caso que el dir no tenga elemento o el item sea un archivo.
    Salto (Pasa)

Caso contrario
    Guarda el tamano de la LL del hijo.
        Recorre el tamaño
            Devuelve un elemento del ChildList y lo guarda en item.
            En caso de coincidencia.
            Se agrega a la variable global final.
        Caso contrario
            En caso no de haber coincidencia, hace el llamado recursivo con el nuevo elemento.
            Vuelve a llamarse.
Devuelve el nodo encontrado. None, en caso que no.

Método searchInnerInTreeForName

Recibe los parametros (root y nombre)

Si el item a buscar es el root
    Retorna el root

Si el item no tiene hijos
#caso que el dir no tenga elemento o el item sea un archivo.
    Salto (Pasa)

Caso contrario
    Guarda el tamano de la LL del hijo.
        Secorre el tamaño
            Devuelve un elemento del ChildList y lo guarda en item.
            Si En caso de coincidencia.
                Se agrega a la variable global final2.
            Caso contrario
                En caso no de haber coincidencia, hace el llamado recursivo con el nuevo elemento.
                Vuelve a llamarse.
Devuelve el nodo encontrado. None, en caso que no.

Método deleteElementToTree

Recibe como parametro (elemento)

#Elimina un elemento en el arbol, incluyendo si este tiene hijos.

Si se encuentra el elemento buscado
    Guarda el nodo padre
    Nos posicionamos en el hijo y borramos el elemento
    Imprimimos “borrado”
Caso contrario
    Imprimimos “No pudo borrarse

Método convertInner

#Este metodo guarda en una LL TODOS los nodos que existen en el arbol.

Retorna su función recursiva.

Método convertInner

Recibe como parametro un root

#Funcion recursiva para poder GUARDAR TODOS los nodos del arbol en una LL.
#Conversion a LL recursiva.

SI el root es igual al del parametro
    Agregado al LL que sera usada para crear .mem

Si no tiene hijos
    Salta (Pasa)

Caso contrario
    Se guarda el tamaño de la longitud de los hijos
    Se reccore el tamaño
        Se almacena la posición del hijo
        Se agrega el item a la lista a texto plano
        Se hace el llamado recursivo

Método addNewElements

Recibe los parametros (Agregar a nodo, agregar a padre, objeto a arbol)

#[NodoPadre,Lista que se le extraera los elementos]

Si Agregar a nodo no tiene hijos
    Salta (Pasa)

Caso contrario
    Se guarda el tamaño de la longitud de los hijos de agregar a nodo.
    Se recorre el rango del tamaño.
        Se almacena la posición del hijo en un item Actual
        Se almacena el tipo de dato para agregar una extesión
        Si el tipo de dato del item actual es igual a 1
            Se agrega el icono de folder al item agregado
            Se agrega al arbol el objeto

        Caso contrario
            Se agrega el icono al item agregado
            Se agrega al arbol el objeto

        Se hace el llamado a si misma de forma recursiva.

Método convertTreeToPlaneText

Recibe los parametros (Numero de arbol)

Guarda en una LL TODOS los nodos que existen en el arbol.
Se hace una lista enlazada temporal a partir de la lista enlazada básica global.
SI el numero de arbol es igual a uno
    Se almacena la ruta 'Memoria/Tree-A.mem'
Caso contrario Si el numero de arbol es igual a 2
    path = 'Memoria/Tree-B.mem'

Limpiamos el contenido del texto plano
#Esto evita duplicados

Abrimos el texto plano
Recorremos la lista que contiene todos los nodos del arbol
Extrae un item de la Lisla enlazada
Si el nodo es una carpeta
    Escribira el nombre del nodo más /
Caso contrario Si es archivo
    Al final de cada linea salta de linea
#Se limpia la lista, SINO al siguiente guardado ira concatenando.

Método extractItemsToplaneText

Recibe como parametro (Numero de arbol)

Se crean dos listas
Un contador de tabulados y otro de padre

Si el numero de arbol es igual a 1
    Se almacena la ruta "Memoria/Tree-A.mem"
Si el numero de arbol es igual a 2
    Se almacena la ruta "Memoria/Tree-B.mem"

Abrimos el archivo
Leemos linea por linea
    Hacemos un split cada vez que encuentre un “\t”
    adjuntamos en la lista cada split
Cerramos el documento

#Ahora quitamos el salto de linea del final
Recorremos el rango de la longitud de la lista
    Recorremos el rango de la longitud de la lista ahora de forma bisimensional
        Y con strip quitamos cada salto de linea.
Creamos una nueva lista y una sublista
Almacenamos un tipo de dato 1

Recorremos la longitud de la lista
    Tabulados 0
    Recorremos el rango de la longitud de la lista ahora de forma bisimensional
        Almacenamos el indece actual
        Si el indice actual es igual de ‘’
            El tabulado incrementa 1
        Si el indice actual es diferente de ‘’
            Si el indice actual[-1] es igual a “/”
                Tipo de dato es igual a 1
                Eliminamos el “/” del indice actual

Caso contrario SI el indice actual[-1] es distinto de “’/”
Tipo de dato 2
Adjuntamos el indice actual a la sub lista
Adjuntamos el tipo de dato a la sub lista
Adjuntamos los tabulados a la sub lista
La sub lista ahora esta vacio
La sub lista ahora es un arreglo
Retornamos la contrucción de la lista a texto plano
#[value,typed,tabulated]

Método buildListToTextPlane

Recibe como parametro una lista

#Asigna los respectivos padres a cada elemento, basado en su tabulado y recorrido de la lista dada.

Creamos una lista para guardar los padres
Adjuntamos la lista en la posición [0],[0] a los padres guardados
Guardamos el tamaño de la lista
Recorremos en el rango de 1 hasta el tamaño max de la lista.
Hacemos un contador de la posición + 1
SI el contador es menor al tamaño de la lista
Si La lista en la posición [contador][2] es mayor e igual a lista en la posición [i][2] o la lista en la posición Ñ[i][2] es igual a lsita en la posición[i-1][2]
Se adjunta los padres guardados en la posición [i] de la lista
Si la lista en la posición [i][2] es diferente a la lista en la posición [contador][2] y la lista en la posición [i][2] es menor a la lista en la posición[contador][2]
Adjuntamos la lista en la posición[i][0] en los padres guardados
Si la lista en la posición[contador][2] es menor a la lista en la posición[i][2]
Tabulado es igual a la lista en la posición[i][2] - la lista en la posición[contador][2]
Adjuntamos la lista en la posición[i] de los padres guardados en la posición [-1]
Recorremos el rango de 0 a tabulados
Elimina el elemento en el índice dado de la lista.

Contador lo hacemos 0
Caso contrario
Adjuntamos la lista en la posición [i] de los padres guardados en la posición [-1]

Retornamos la lista.

Método clearContentOfPlaneText

Recibe como parametro un número

#Funcion para limpiar el contenido actual del archivo texto plano.

SI el numero es igual a 1

Se almacena la ruta 'Memoria/Tree-A.mem'

Caso contrario si el numero es igual a 2

Se almacena la ruta 'Memoria/Tree-B.mem'

Abrimos el archivo
Leemos linea a linea
Cerramos archivo
Abrimos el archivo
Recorremos linea por lineas
    Si la linea es igual a “” + “\n”
        Escribimos en la linea
Cerramos

Método convertPlaneTextToTree

Recibe como parametro numero de arbol

#Esta funcion transforma una lista de elementos extraidos, agregandolos al arbol con sus respectivas transformaciones a items de tipo QListWidgetItem

Extramos los items del texto plano y los guardamos en una lsita de items
Elimina el elemento en el índice dado de la lista.

Recorremos en el rango de la longitud de la lista de items
    Guardamos el elemento de la lista de items de la posición i
    Almacenamos el nombre del elemento en la posición 0 de los elementos
    Almacenamos la extensión del item
    Almacenamos el tipo de dato del elemento de la posición[1]
    SI el tipo de dato es igual a 1
        Almacenamos el valor de el cual contiene el icono del folder.
        Agregamos el elemento al arbol
    Caso contrario si el tipo de dato es igual a 2
        Almacenamos el valor de el cual conteine el icono del llamado del método
        Agrgamos el elemento al arbol

Método itemExtensions

Recibe como parametro un item

Realiza un split del item al encontrar un “.” y almacenamos la extensión

Si la longitud de la extesión es distinta de 1
    Analiza las posibles extensiones como:
    mp3,pdf,py,js etc... y agrega un icono.
    Y retorna la ruta.

Archivo *inputWindow*

Clase “AppWindowPrincipal”

Su método constructor recibe

El método executionPrincipal
El tamaño 200*100
Establecer banderas de ventanas
Centrado

Método executionPrincipal

Una etiqueta con el texto "SELECT MULTIPLE APPLICATION"
Un botón para continuar y otro para salir
El botón continuar conectado al método openWindow
El botón salir conectado al método exitApplication

Definición del tamaño y diseño de los botones.

Método center

Toma las medidas geometricas de la pantalla

Método openwindow

Abre la interfaz de usuario
Se oculta la actual

Método Application

Cierra el programa.

Archivo Window

Clase App

Su método constructor recibe:

Un titulo:'SIMULADOR DE SISTEMA DE ARCHIVOS'
La geometria de la ventana 800 ancho * 350 alto.
Quitar el borde por defecto de la ventana.
Establecer una hoja estilo cascada

#Esta define el aspecto visual de la aplicación.

Se crean los objetos y las variables.

Los dos arboles que son los objetos tipo arbol para guardado de item de las ListTree

Una lista enlazada para control/guardado de los elementos clickeados.

Dos iconos que se agregan a los elementos de la ListTree

Se realiza el llamado de las funciones de ejecución

#Los procedimientos para todo relacionado al arbol 2, se reutilizara codigo del arbol 1

Agregado por defecto un root en ambas listas.
Los botones, otros botones, el diseño, convertir texto plano a arbol, centrar ventana

Adjuntar items del item root uno y dos.

#[objeto Arbol, Funcion agregar carpeta, funcion agregar archivo, ruta del TextoPlano]

Método buttonsTrees

#Metodo de creacion/gestion de los botones de la ventana.

Despegable para el tipo de dato
Estado de apagado inicial del botón
Los tipos de datos del despegable son ‘Tipo de dato’, ‘Directorio’ y ‘Archivo’
Evento Índice actual cambiado conecta al método (enableBtnTypeData1)
#Mostrara una etiqueta con letras rojas con el texto “Elija un tipo de dato”

En el botón agregar1 se establece un icono
Se define el tamaño del icono 20*20
#Evento-mouse, se muestra un texto al pasar el puntero por el boton.
Evento conecta con método enabledToClickAddButton1
#Conexion de funcion al clickear al boton.

Botón borrar1 se establece un icono
Se define el tamaño del icono 20*20
Evento conecta con método enabledToClickDeleteButton1
#Conexion de funcion al clickear al boton.

Se define una etiqueta label
Por defecto desactivada
#Esta muestra el mensaje “Elija tipo de dato”

##======Se realiza lo mismo pero para el arbol #2======##

Método otherButtons

btnLeft Conectado a enableBtnLeft
#Trasñada el item copiado a TreeList#1

btnRigth Conectado a enableBtnLeft
#Trasñada el item copiado a TreeList#2

Minimize, Maximize, Close Definidos como TooldButton
#Minimizar, maximar, cerrar, la ventana.

ShortcutToAdd #Control + n
Conectado al evento enableEvenetToKeyboardToAdd

ShortcutToDelete #Del
Conectado al evento enableEvenetToKeyboardToDelete

ShortcutToBack #Backspace
Conectado al evento enableEvenetToKeyboardToBack

Método enableEventToKeyboardToAdd

Si el estado Clicktolist es igual a 1
    Llamado al método enabledToClickAddButton1
Si el estado Clicktolist es igual a 2
    Llamado al método enabledToClickAddButton2

Método enableEventToKeyboardToDelete

Si el estado Clicktolist es igual a 1
    Llamado al método enabledToClikDeleteButton1
Si el estado Clicktolist es igual a 2
    Llamado al método enabledToClikDeleteButton2

Método enableEventToKeyboardToBack

Si el estado Clicktolist es igual a 1
    Llamado al método enableDoubleClickToPointPoint1
Si el estado Clicktolist es igual a 2
    Llamado al método enableDoubleClickToPointPoint1

#================ METODOS QUE DEFINEN/ACIONAN/ACTIVA EN DETERMINADOS EVENTOS SOBRE LAS HERRAMIENTAS(BOTONES,CAJAS ETC.)==================
    #----------------------ARBOL #1----------------------
#Metodo que ejecuta al presionar en agregar, tambien gestiona el agregado, si hay items repetidos, este niegua el agregado y manda un mensaje.

#Todos los método se reutiliza cód para el arbol #2

Método enabledToClickAddButton1

Habilitado por defecto
Des habilita el labels1
Busca el nodo padre de la lista actual

Abre la ventana de dialogo para guarda el valor a la lista

Si presiona ok es igual a True y el valor agregado es diferente de “”

    Si el indice actual del tipo de dato del boton es igual a 1
        Labels1 deshabilitado
        Si el hijo del padre actual de la lista es diferente de vacio

            Al buscar el nombre y tipo del hijo del padre de la lista es diferente de True
            #Aqui verifica si hay un nodo con el mismo nombre y tipo.
            #En caso que no existe el mismo, se agrega.
        Caso contrario “Ya existe un elemento con ese nombre”
    Caso contrario agrega el directorio al arbol
Caso contrario Si el hijo del padre actual de la lista es diferente de vacio

    Al buscar el nombre y tipo del hijo del padre de la lista es diferente de True
    #Aqui verifica si hay un nodo con el mismo nombre y tipo.
    #En caso que no existe el mismo, se agrega.
    Caso contrario “Ya existe un elemento con ese nombre”
Caso contrario “Elija el tipo de dato”

Método enabledToClikDeleteButton1

Al seleccionar un item de la lista 1
Si la long de item seleccionado es diferente de 0
    Labels1 deshabilitado
    Se guarda el itempadre de la busqueda en el arbol
    Se recorre en rango de la longitud de la lista seleccionada
        Borra el elemento del arbol
    Si el item padre tiene un nombre diferente de 'home'
        Repintar la lista
    Caso contrario si es igual a 'home'
        Repintar la lista falso
Caso contrario
    Seleccione elementos para borrar

Método enableDoubleClickItemTree1

#Metodo que conecta al clickear 2 veces en un item del TreeListkItemTree1

Labels1 deshabilitado
Botón tipo de dato habilitado
Retorna el item clickeado 2 veces en el TreeList.
Busqueda en produndidad en el arbol 1 para almacenar el padre clikeado

En caso de presionar el ".."
    Se llama a enableDoubleClickToPointPoint1

SI el item clickeado es igual al punto1
    Imprimir "Actualmente esta en el directorio
    El nombre del item clickeado a la posición [-1]

    Caso contrario SI el item clickeado es diferente al punto 1 y el item clickeado es diferente al punto punto

    Adjunta el click padre a los items clickeados

    Si el tipo de dato es igual a 1
        El estad de los elementos del arbol es verdadero.
        Si el item el texto del item clickeado es igual a 'root'
            Saltar (pasar)
        #Busca y devuelve el Nodo en el arbol.
        Repintar el arbol 1
    Caso contrario
        Habilitar labels1 "No es un directorio"

Método enableDoubleClickToPointPoint1

Deshabilitar el labels1

SI la longitud de los items clickeados es mayor a 1
    Buscar en el arbol1
    #Devuelve el NODO padre anterior.
    SI el nodo del padre anterior es una intancia diferente de True
        Si el nombre del itemclickeado en la posición [-2] es igual a ‘home
            Repintar la lista
            El valor en el índice aparece y se elimina.
        Caso contrario
            Repintar la lista1
            El valor en el índice aparece y se elimina.
    Caso contrario
        Si
            Repintar la lista1
            El valor en el índice aparece y se elimina.
        Caso contrario
            Limpiar la lista1
            Agregar items a la lista1
            Estado de elementos del arbol1 : Falso

#ACTIVACION GENERAL DE HERRAMIENTAS.
Método stateToElementsTree1

#Habilita y deshabilita los botones, etiquetas y sohrtcuts.

Método enableBtnTypeData1

Deshabilitar labels1
Si el indice actual del tipo de dato de enableBtnTypeData1 es igual a 0
    Habilitar labels1(True,'¡Elija el tipo de dato!')
Caso contrario
    Deshabilitar labeltextmessage

Método enableLabels1

Recibe como parametro estado y text

#En base a eso va cambiando a lo largo del programa.

Los mismos métodos se repiten para el arbol 2 y listTree2

Método layoutTree

#Gestion y orden de los elementos en la ventana.

#Orden horizontal(botones de control).
#Orden vertical

El contenido es como a almacenado en cajas y luego es ordenado.

#================== General======================#
#Metodo para limpiar el TreeList, recibiendo como parametro, al TreeList"[1 or 2]".

Método cleanListView

Recibe como parametro una lista de objetos

Almacenamos el tamaño de la lista de objetos

Recorremos desde la posición 0 hasta el max del tamaño de la lista de objetos
    Contar el tamaño de la lista de objetos
    Mientras el tamaño sea diferente de 0
        Tomar un item de la posición i de la lista de objetos
        Contar el tamaño de la lista de objetos
    Actualizamos la lista
Repintamos

Método repaintListView

Recibe como parametr la lista de objetos, item, estado = True por default

#[Arbol a pintar,desde que item pintar]

Limpiamos la lista
Si el estado es verdadero
    Si la lista de objetos es igual al la lista del arbol 1
        Agregar item punto 1 a la lista
Agregar item dos puntos a la lista
    Si la lista de objetos es igual al la lista del arbol 2
        Agregar item punto 1 a la lista
        Agregar item dos puntos a la lista

Si hay hijo del item
    Ordenar la lista enlazada (item)
    Tomar la longitud de los item
    Recorrer desde 0 hasta el tamaño maximo
        Almacenar el valor de la posición del hijo del item
        Agregarlo a la lista de objetos.

Método enableBtnRight

Deshabilita labels2
Guarda los items seleccionados.
Retorna el elemento al que se le pegara los items.

Si la longitud de los items seleccionado es diferente de 0
    Buscar en el arbol dosy copiar
    Recorrer el rango de la longitud de la lista seleccionada
        Si el tipo de dato copiado es igual a 1
            Agregar item un icono de folder y el nombre copiado
                Si la busqueda por nombre y tipo del hijo del padre actual es True
                    Remover
                    warningMessage “Elemento ya existente","El elemento '%s' ha sido reemplazado"”
                    Agregar al Treelist2
                Caso contrario
                    Agregar al Treelist2
            Caso contrario
                Agregar al Treelist2
            Buscar en el arbol2 un nuevo padre
            Agregar nuevo elemento al arbol 2

        Caso contrario si el tipo de dato copiado es igual a 2
            #En caso que sea archivo.
            Si hay un hijo en el padre actual
                Si al buscar un nombre y tipo del hijo del padre actual es True
                    Remover
                    warningMessage "Elemento ya existente","El elemento '%s' ha sido reemplazado"
                    Agregar archivo al arbol 2
                Caso contrario
                    Agregar archivo al arbol 2
            Caso contrario
                Agregar archivo al arbol 2
        SI el nombre del padre actual es diferente de ‘home
            Repintar el Treelist
        Caso contrario si es igual a ‘home’
            Repintar el Treelist
        Convertir arbol a texto plano
    Caso contrario
        Habilitar labels1 “No hay elementos copiado”

BtnLeft realiza la inversa del BtnRigth

#Funcion al boton de flecha derecha, lo cual pega al TreeList#2 los elementos seleccionados.

Método centerWindow
    Tomar la geometria de la pantalla
    Mover según los datos

Método minimizeWindow
    Minimizar
    Método closeWindow
        Cerrar

Método maximizeWindow
    Maximizar la pantalla.

Método warningMessage
    #Envia un mensaje de alerta.