Práctica 16

Datos generales:

Nombre de la Práctica	Listas simplemente ligadas
Nombre de la carrera	Ingeniería de Software
Nombre de la materia	Estructuras de Datos
Número y nombre de Unidad(es) temática(s)	III. Estructuras lineales.
Docente que imparte la materia	Aldonso Becerra Sánchez
Fecha de entrega para los alumnos	29-septiembre-2021
Fecha de entrega con extensión y penalización	30-septiembre-2021
Fecha de elaboración	29-septiembre-2021

Objetivo de la Práctica	Profundizar con las operaciones en las listas enlazadas.
Tiempo aproximado de realización	3 horas
Introducción	La memoria dinámica es un elemento importante en el manejo de información abundante donde no se sabe de antemano cuantos datos son los requeridos, por tanto solventa las limitaciones de la memoria estática. Las listas enlazadas permiten la manipulación de la memoria dinámica a través de la liga de nodos sucesivos.

Referencias que debe consultar el alumno (si se requieren):

Referencia 1:

1. Cairo, Osvaldo; Guardati, Silvia. Estructura de Datos, Tercera Edición. McGraw-Hill, México, Tercera Edición, 2006.

Referencia 2:

2. Mark Allen Weiss. Estructura de datos en Java. Ed. Addison Wesley.

Referencia 3:

3. Joyanes Aguilar, Luis. Fundamentos de Programación. Algoritmos y Estructuras de Datos. Tercera Edición, 2003. McGraw – Hill.

Actividades que debe realizar el alumno:

Actividad inicial:

Generar el reporte en formato IDC.

Actividad 1:

Primero genere la **Introducción**.

Actividad 2:

Implemente el código de listas encadenadas (la lista no debe tener atributos que lleven el control numérico de cuántos elementos existen actualmente en la lista):

a) Imprimir en orden inverso los elementos de la clase ListaEncadenada: `void imprimirOrdenInverso()`. Este método viene de la interface "ListaDatos".
b) Realizar el método que permita encontrar un elemento en una lista encadenada. El método debe regresar el contenido del info encontrado: `Object buscar(Object elemento)`. Este método viene de la interface "ListaDatos". Lo interesante de este método es que se debe analizar cómo se puede reutilizar hacia métodos existentes que usan su funcionalidad.
c) Hacer un método que guarde todos los elementos de una lista en un ArregloDatos, el cual debe regresar como valor de retorno: `ArregloDatos aArreglo()`.
d) Hacer un método que guarde todos los elementos de una lista a un arreglo siempre y cuando no sean iguales a los que contiene el arreglo pasado como argumento: `ArregloDatos aArreglo(Arreglo elementosADescartar)`.
e) Hacer un método que guarde los elementos de una lista en una tabla 2d. A este método se le pasarán el número de renglones y columnas de la matriz resultante. En caso que no se ajusten de renglones o columnas con la cantidad de elementos de la lista, se deben rellenar con null: `Tabla2D aTabla2D(int filas, int columnas)`.

f) Hacer un método que agregue al final de la lista los elementos pasados como argumentos en un ArregloDatos o una ListaEncadenada (listadatos2). Este viene de la interface "ListaDatos". `boolean agregarLista(Object listaDatos2)`.
g) Hacer un método que devuelva una copia de la lista ligada. La primera lista debe ser independiente de la copia: `Object clonar()`. Esta viene de la interface "ListaDatos".
h) Hacer un método que agregue los elementos de una tabla 2d pasada como argumento al final de una lista. Los elementos irán agregando renglón por renglón o columna por columna, según se defina a través de una enumerado (COLUMNA; FILA): `boolean agregarTabla2D(Tabla2D tabla, TipoTabla enumTipoTabla)`.
i) Hacer el método que vacíe una lista ligada. Viene desde la interface "ListaDatos". `void vaciar()`.
j) Hacer un método que rellene una lista con valores iguales indicados por argumento. Este método viene de la interface "ListaDatos": `void rellenar(Object elemento, int cantidad)`.
k) Hacer un método que cuente elementos en una lista con valores iguales indicados por el argumento. Este método viene de la interface "ListaDatos". `int contar(Object elemento)`.
l) Hacer un método que invierta el orden de una lista. Este método viene de la interface "ListaDatos". `void invertir()`.
m) Hacer un método que cambie un elemento viejo por uno nuevo dado un número de ocurrencias. Este método viene de la interface "ListaDatos". `boolean cambiar(Object elementoViejo, Object elementoNuevo, int numVeces)`.
n) Hacer un método que cambie un elemento viejo por uno nuevo dada una posición de búsqueda: `boolean cambiar(int indice, Object elemento)`.
o) Hacer un método que devuelva el elemento que se encuentra en una posición dada como argumento: `Object obtener(int indice)`.
p) Hacer un método que indique si una lista es igualita a otra lista. Elemento por elemento; en la misma posición deben ser iguales. Este método viene de la interface "ListaDatos": `boolean esIgual(Object listaDatos2)`.
q) `public Object redimensionar(int maximo)`: redimensiona el tamaño de la lista al nuevo tamaño indicado por máximo. Si el tamaño es menor, los elementos sobrantes deben ser eliminados. Si el tamaño es mayor, los datos anteriores deben conservarse, y los nuevos espacios deben llenarse con null.
r) `public Object eliminar(int indice)`**:** elimina un elemento de la lista en una posición especifica, regresando el elemento eliminado.

s) Cualquier otro método que venga definido en la interface "ListaDatos" debe programarse y adaptarse para que funcione en una lista ligada.

Haga el programa (actividad 2, la cual es el **Desarrollo** del programa).

Actividad 3:

Pruebe el funcionamiento del programa de la actividad 2 con todo y sus capturas de pantalla.

Actividad 4:

Realice la sección de **Código agregado** (diagrama de clases UML).

Actividad 5:

Realice la sección de **Pre-evaluación** (use los lineamientos establecidos).

Actividad 6:

Finalmente haga las **Conclusiones**.

Actividad 7:

Enviar en http://ingsoftware.reduaz.mx/moodle

Archivo anexo que se requiere para esta tarea (opcional):

Dudas o comentarios: a7donso@gmail.com