



Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software

Práctica 11

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	1-mayo-2025
Fecha de entrega con extensión y penalización	2-mayo-2025
Fecha de elaboración	29-abril-2025

Objetivo de la Práctica	Profundizar con las operaciones en las listas enlazadas.
Tiempo aproximado de realización	5 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.



Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software

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 la funcionalidad de la clase Pila utilizando listas dinámicas en lugar de arreglos (llamada PilaDinamica). Invoque métodos existentes.

Haga el programa (actividad 2, la cual es el **Desarrollo** del programa, junto con la captura de pantalla del programa funcionando).

Actividad 3:

Implemente la funcionalidad de la clase Cola simple utilizando listas dinámicas en lugar de arreglos (llamada ColaDinamica). Invoque métodos existentes.

Haga el programa (actividad 3, la cual es el **Desarrollo** del programa, junto con la captura de pantalla del programa funcionando).

Actividad 4:

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



Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software

- a) Imprimir en orden inverso los elementos de la clase ListaDin (implementa ListaDatos): `void mostrarOI()`. Use pilas para resolverlo.
- b) Realizar el método que permita encontrar todas las ocurrencias de un elemento en una lista enlazada. El método debe regresar el contenido de los valores encontrados como lista: `ListaDin encontrarLista(Object valor)`. 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 dinámica en una Arreglo, el cual debe regresar como valor de retorno: `Arreglo aListaEstatica()`.
- 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: `Arreglo aListaEstatica(Arreglo elementosADescartar)`.
- e) Hacer un método que guarde los elementos de una lista en una matriz 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: `Arreglo2D aMatriz2D(int filas, int columnas)`.
- f) Hacer un método que agregue al final de la lista los elementos pasados como argumentos en una Arreglo o una ListaDin (`listadatos2`). `boolean agregarLista(ListaDatos listaDatos2)`.
- g) Hacer un método que devuelva una copia de la lista ligada. La primera lista debe ser independiente de la copia: `clonar()`.
- h) Hacer un método que agregue los elementos de una matriz 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`; `RENGLON`): `boolean agregarMatriz2D(Arreglo2D tabla, TipoTabla enumTipoTabla)`.
- i) Hacer el método que vacíe una lista ligada. `void vaciar()`.
- j) Hacer un método que rellene una lista con valores iguales indicados por argumento. `void rellenar(Object valor, int cantidad)`.
- k) Hacer un método que cuente elementos en una lista con valores iguales indicados por el argumento. `int contar(Object valor)`.
- l) Hacer un método que invierta el orden de una lista. `void invertir()`.
- m) Hacer un método que cambie un elemento viejo por uno nuevo dado un número de ocurrencias. `boolean cambiar(Object valorViejo, Object valorNuevo, int numVeces)`.
- n) Hacer un método que cambie un elemento viejo por uno nuevo dada una posición de búsqueda: `boolean cambiar(int indice, Object valor)`.
- o) Hacer un método que devuelva el elemento que se encuentra en una posición dada como argumento: `Object obtener(int indice)`.



Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software

- p) Hacer un método que indique si una lista es igualita a otra lista. Elemento por elemento; en la misma posición deben ser iguales. `boolean esIgual(ListaDatos 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 quitar(int indice)`: elimina un elemento de la lista en una posición específica, regresando el elemento eliminado.
- s) `public ListaDin quitar(Object valor)`: elimina un elemento de la lista dado un valor específico, regresando el elemento eliminado. Puede haber varios elementos iguales en la lista.
- t) Cualquier otro método que venga definido en la interface “ListaDatos” debe programarse y adaptarse para que funcione en una lista dinámica.

Haga el programa (actividad 4, la cual es el **Desarrollo** del programa).

Actividad 5:

Pruebe el funcionamiento del programa de la actividad 2, 3 y 4 con todo y sus capturas de pantalla.

Actividad 6:

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

Actividad 7:

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

Actividad 8:

Finalmente haga las **Conclusiones**.

Actividad 9:

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



Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software

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

Dudas o comentarios: a7donso@gmail.com