1ºqUNIVERSIDAD CATÓLICA DE SANTA MARÍA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LABORATORIO 03:

LIST & LINKED LIST

I
OBJETIVOS

- Utilizar los conceptos de genericidad para implementar el TAD lista genérica.
- Definir la interface genérica de una lista
- Definir y crear la estructura de un nodo capaz de almacenar objetos de tipos genéricos.
- Definir y crear la lista enlazada genérica cuyas operaciones sean capaces de manipular tipos de datos genéricos.
- Utilizar herencia para crear listas enlazadas ordenadas genéricas.

II
TEMAS A TRATAR

- Definiciones generales
- Operaciones
- Implementación de listas enlazadas definidas por el usuario
- Listas enlazadas ordenadas.

III
MARCO TEORICO

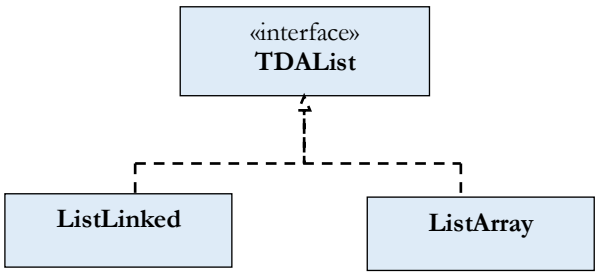
Definiciones generales

TDA Lista

Un TDA es un Tipo Abstracto de Datos, el mismo que incluye los datos y las operaciones que pueden ser aplicados a estos datos. Un ejemplo de un TDA es el tipo de datos Integer, cuyos datos representa el conjunto de números enteros cuyas operaciones son la suma, la resta, multiplicación y división entre otras.

El TDA Lista representa una secuencia de elementos cuya organización obedece a una estructura lineal de los elementos, en los que se puede realizar las operaciones de inserción, eliminación, búsqueda, etc. de un elemento.

Este TDA se puede implementar de diferentes formas tal como se observa en el siguiente diagrama:



El tipo ListArray implementar el TDA Lista utilizando como estructura de datos (contenedor de datos) un arreglo en el cual, como ya se conoce, los elementos se encuentran almacenados en posiciones sucesivas de memoria y por lo tanto, el acceso a los elementos se realiza a través del índice del elemento correspondientemente.

Otra forma es utilizando como estructura de datos una lista enlazada.

Lista Enlazada

Una lista enlazada es una colección de componentes llamados **nodos**, donde cada uno de ellos tiene dos partes: uno almacena la información relevante (**dato**), y la otra guarda la dirección del siguiente nodo de la lista. A este último se le llama **next** (**enlace/siguiente**).

Todos excepto el último nodo, tienen una dirección en el campo **next**.

La dirección del primer nodo de la lista se guarda en una ubicación separada (una variable de referencia), llamada **cabeza** (**head** o **first**).

La diferencia principal de una lista enlazada con un arreglo es que en un arreglo los elementos deben de estar almacenados en posiciones continuas de la memoria, sin embargo, en una lista no.

Operaciones de una Lista

Las operaciones básicas de una lista enlazada son:

boolean isEmptyList(): determina si la lista está vacía

int length(): determina la cantidad (longitud) de elementos que hay en la lista

void destroyList(): elimina los elementos de la lista dejándola vacía

int search(x): verifica si el elemento x está o no en la lista, si está retorna su posición.

void insertFirst(x): inserta el nuevo elemento x al inicio de la lista

void insertLast(x): inserta el nuevo elemento x al final de la lista

void removeNode(x): elimina el elemento x de la lista.

IV
ACTIVIDADES

Para las actividades tenga en cuenta el diagrama de la sección anterior.

01. En un nuevo proyecto, cree y declare la interface genérica **TDAList** en la que declare las operaciones de una lista (recuerde que una interface contiene elementos públicos y abstractos).

02. TDA Lista implementada a través de un arreglo de objetos genéricos

a) Cree la clase **ListArray** que implemente la interface **TDAList** y los métodos de ésta, en donde se utilice como estructura de datos un arreglo (no utilice ninguna clase de la Collections List - ArrayList).

b) En una clase TestList pruebe las operaciones de la lista ListArray para diferentes tipos de datos. Uno de ellos debe ser el tipo **Person**, definida a partir de la una clase con los atributos nombre, apellidos y edad de una persona.

c) Examine el comportamiento de las operaciones de esta lista basada en el uso de arreglos, y conteste las siguientes preguntas:

i. ¿ Cuál es el orden de complejidad de la inserción de un elemento en una lista basada en arreglo de N elementos en el mejor de los casos? ¿En qué circunstancia se puede dar este orden?

ii. ¿ Cuál es el orden de complejidad de eliminar el primer elemento en una lista basada en arreglos en donde los elementos no tengan ningún orden, en comparación de una que esta ordenada?

03. Implementación del TDA Lista a través del uso de una lista enlazada definida por el usuario. Para esto considere lo siguiente:

a) Agregue al proyecto la clase genérica **Node** en la que se tiene la declaración de la estructura de un nodo de la lista enlazada que se muestra a continuación.

```java
public class Node<T>{
    private T data;
    private Node<T> next;
    public Node(T data){ this (data, null); }
    public Node(T data, Node<T> next){
        this.data = data;
        this.next = next;
     }
    public T getData() { return this.data; }
    public Node<T> getData() { return this.next; }
    public void setData(T data) { this.data = data; }
    public void setNext(Node<T> next) { this.next = next; }

    //Function to returns the information stored in the node
    //Postcondition: Returns info.
    public String toString(){
        //Here write your code
    }
}
```

b) Agregue la clase genérica **ListLinked** que debe de implementar la interface **TDAList**, por tanto, esta clase deberá implementar los métodos abstractos de esta interface.

```java
public class ListLinked <T> implements TDAList <T> {
    protected Node<T> first;

    public ListLinked(){
        this.first = null;
    }

    //Function to determine whether the list is empty.
    //Postcondition: Returns true if the list is empty; otherwise, returns false.
    public boolean isEmpty(){
        //Here write your code
    }

    //Function to return the number of nodes in the list.
    //Postcondition: The value of count is returned.
    public int length(){
        //Here write your code
    }

    //Function to delete all the nodes from the list.
    //Postcondition: first = null, count = 0
    protected void destroyList(){
        //Here write your code
    }

    //Function to determine whether searchItem is in the list.
    //Postcondition: Returns the position if searchItem is found in the list;
    //               otherwise, it returns -1.
    public int search(T searchItem){
        //Here write your code
    }
```

```
    //Function to insert newItem in the list.
    //Postcondition: first points to the new list and newItem is
    //               inserted at the beginning of the list.
    public void insertFirst(T newItem){
        //Here write your code
    }

    //Function to return newItem at the end of the list.
    //Postcondition: first points to the new list, newItem is inserted at
    //               the end of the list.
    public void insertLast(T newItem){
        //Here write your code
    }

    //Function to delete deleteItem from the list.
    //Postcondition: If found, the node containing deleteItem is deleted from the
    //               list, first points to the first node of the updated list.
    public void removeNode(T deleteItem) {
        //Here write your code
    }

    //Function to returns the information stored in the list
    //Postcondition: Returns info.
    public String toString() {
        //Here write your code
    }
}
```

c) En la clase **TestList** cree varias listas enlazadas de diferentes tipos en las que deberá de verificar las operaciones de lista enlazada definida. Uno de ellos debe ser **Person**.

V
EJERCICIOS

01. Implementación de una lista enlazada ordenada.

Muchas veces es necesario mantener un cierto orden entre los elementos de una lista enlazada. Si bien, la operación de búsqueda será mucho más eficiente en una lista enlazada ordenada a una que no esté ordenada, la operación de inserción puede tornarse un poco más lenta en vista que el elemento nuevo debe ser insertado en el lugar que le corresponda.

Por ejemplo, el código del método de búsqueda de un elemento en una lista ordenada ascendentemente puede ser el siguiente:

```
boolean search(T item) {
    Node<T> aux = this.first;

    while( aux != null && aux.getData().compareTo(item) < 0 )
          aux = aux.getNext ();

    if(aux != null)
         return aux.getData().equals(item);
    return false;
}
```

Entonces:

a) Al proyecto anterior, agréguele una clase **OrderListLinked** que derive de **ListLinked**.

b) Redefina los métodos de búsqueda, inserción y eliminación respectivamente, de forma que el resultado de la inserción y de la eliminación mantenga el orden de la lista, y que la búsqueda considere el orden.
Nota: fíjese que en este tipo de lista, algunos métodos de la lista simple no tienen sentido puesto que no mantienen los datos ordenados.

¿Cuáles métodos de la clase ListLinked no serían pertinentes usar en la una lista del tipo OrderListLinked?

c) Modifique el código de la clase **TestList** de tal manera que pueda usted probar la implementación de una lista ordenada con diferentes tipos de datos concretos.

d) Cree una lista enlazada ordenada de personas y pruebe las operaciones implementadas.

e) Analice como se realizan cada una de las operaciones tanto en la lista simple como en la lista ordenada, y suponga que se tienen N elementos almacenados en cada una de las listas. A partir del análisis responda las siguientes preguntas:

¿ Cuál es el orden de complejidad de la inserción de un elemento en una lista enlazada simple de N elementos en el mejor de los casos? ¿En qué circunstancia se puede dar este orden?

¿ Cuál es el orden de complejidad de la inserción de un elemento en una lista enlazada ordenada de N elementos en el mejor de los casos? ¿En qué circunstancia se puede dar este orden?

¿ Cuál es el orden de complejidad de la inserción de un elemento en una lista enlazada ordenada de N elementos en el peor de los casos? ¿En qué circunstancia se puede dar este orden?

02. Palindrome Linked List.

Sea una lista enlazada de caracteres y se desea saber si la secuencia de caracteres constituye una secuencia palíndroma o no.

Por ejemplo:

```
Input      : A —> B —> C —> B —> A —> null
Output     : The linked list Es palindrome

Input      : A —> B —> C —> C —> B —> null
Output     : The linked list No es palindrome
```

Las siguientes son algunas de las soluciones que permitan verificar si una lista contiene o no una secuencia palíndroma.

2.1 Alternativa de Solución 1

La idea es atravesar recursivamente hasta el final de la lista enlazada y construya una cadena a partir de los caracteres de los nodos en orden de visita. Luego, a medida que se desarrolla la recurrencia, construya otra cadena a partir de los nodos de la lista enlazada, pero esta vez, el orden encontrado de los nodos procesados es el opuesto, es decir, desde el último nodo hacia el nodo principal. Si ambas strings construidas son iguales, podemos decir que la lista enlazada es un palíndromo.

a) El algoritmo se puede implementar de la siguiente manera en Java. Para esto cree un nuevo proyecto Java, e implemente el siguiente código en la clase **TestSol1** y en la Clase **Node**:

```
class TestSol1 {
   // Construir 's1' y 's2' a partir de la lista enlazada dada con secuencias consecutivas
   // enumera los elementos en la dirección hacia adelante y hacia atrás
   public static void construct(Node head, StringBuilder s1, StringBuilder s2) {
      if (head == null) {
         return;
      }
      s1.append(head.data);
      construct(head.next, s1, s2);
      s2.append(head.data);
```

```
    }

    // Metodo para verificar si una lista enlazada de caracteres dada es un palíndromo
    public static boolean isPalindrome(Node head) {

        // construye la string 's1' y 's2' con elementos consecutivos de la lista enlazada
        // comenzando desde el principio y el final
        StringBuilder s1 = new StringBuilder(), s2 = new StringBuilder();
        Construct(head, s1, s2);

        // comprueba si la lista enlazada es un palíndromo
        return s1.toString().equals(s2.toString());
    }

    public static void main(String[] args) {
        Node head = new Node('A');
        head.next = new Node('B');
        head.next.next = new Node('C');
        head.next.next.next = new Node('B');
        head.next.next.next.next = new Node('A');
        if (isPalindrome(head)) {
            System.out.println("Linked List Es palindrome.");
        }
        else {
            System.out.println("Linked List is No es palindrome.");
        }
    }
}

class Node {
    char data;
    Node next;
    Node(char ch) {
        this.data = ch;
        this.next = null;
    }
}
```

b) Realice diferentes pruebas para diferentes secuencias de caracteres.

2.2 Alternativa de Solución 2

También podemos determinar si una lista enlazada es un palíndromo o no sin construir una string de caracteres. Esto se puede hacer recursivamente comparando los datos del primer nodo con el último nodo, los datos del segundo nodo con el penúltimo nodo, etc.

Podemos hacer esto con el uso de dos punteros principales como parámetros para la función recursivamente. La idea es avanzar recursivamente el segundo puntero hasta llegar al final de la lista enlazada. Cuando se desarrolle la recurrencia, compare el carácter apuntado por el primer puntero con el del segundo puntero. Si en algún momento los caracteres no coinciden, la lista enlazada no puede ser un palíndromo. Para mantener el puntero izquierdo sincronizado con el puntero derecho, avance el puntero izquierdo al siguiente nodo después de cada llamada recursivamente.

A continuación, se muestra la implementación de la idea en Java:

c) En el mismo proyecto implemente el siguiente código en la clase **TestSol2.** La clase **Node** a la que se hace referencia es la del ejercicio anterior:

```
class TestSol2 {
  // Envoltorio sobre la clase `Node`
  static class NodeWrapper {
      public Node node;
      NodeWrapper(Node node) {
          this.node = node;
      }
  }
```

```
    // Método recursivo para verificar si una lista enlazada de caracteres
    // dada es un palíndromo
    public static boolean isPalindrome(NodeWrapper left, Node right) {
        if (right == null) {
            return true;
        }

        // Devuelve falso en el primer desajuste
        if (!isPalindrome(left, right.next)) {
            return false;
        }

        // Copiar el hijo izquierdo
        Node prev_left = left.node;

        // Avanzar el hijo izquierdo al siguiente nodo.
        // Este cambio se reflejaría en las llamadas recursivos principales.
        left.node = left.node.next;

        // Para que la lista enlazada sea un palíndromo, el carácter de la izquierda
        // el nodo debe coincidir con el caracter del nodo derecho
        return (prev_left.data == right.data);
    }

    public static void main(String[] args) {
        Node head = new Node('A');
        head.next = new Node('B');
        head.next.next = new Node('C');
        head.next.next.next = new Node('B');
        head.next.next.next.next = new Node('A');

        // Ajustar nodo, para que su referencia pueda cambiarse dentro de `isPalindrome()`
        NodeWrapper left = new NodeWrapper(head);

        if (isPalindrome(left, head)) {
            System.out.println("Linked List Es palindrome.");
        }
        else {
            System.out.println("Linked List No es palindrome.");
        }
    }
}
```

d) Realice las pruebas con diferentes secuencias a fin de verificar si son o no palíndromas.

03. Análisis de las soluciones

a) Realice la prueba de escritorio (manualmente) del algoritmo implementado en java de la Alternativa de solución 1 y la Alternativa de solución 2, al problema del palíndromo para el caso en que la lista almacena la secuencia: “”

b) A partir del análisis y prueba de escritorio, responda las siguientes preguntas suponiendo que se tienen N elementos almacenados en la lista enlazada.

   i. ¿Cuál es el orden de complejidad del método isPalindrome de la Alternativa de solución 1 implementado en Java?

   ii. ¿Cuál es el orden de complejidad del método isPalindrome de la Alternativa de solución 2 implementado en Java?

04. Aplicando Genericidad a la lista palindrome

Modifique la clase **Node**, así como, los métodos de las clases **TestSol1 y TestSol2** de modo que en un nodo sea posible almacenar algún objeto de cualquier tipo de dato y que los métodos de verificación implementen sus algoritmos para manipular cualquier tipo de dato también. Pruebe lo realizado generando listas de: Golosinas, Personas, String y de Integer.

VI
PARA INVESTIGAR

1. ¿Qué son las clases Wrapper o envoltorios y para que se utilizan?
2. ¿Qué es la clase StringBuilder y para que se utiliza?
3. Investigue a cerca de las clases estándar de java y de C++(STL) que implementen listas enlazadas, y pruebe su funcionamiento.

VII
BIBLIOGRAFÍA Y REFERENCIAS

- Weiss M., "Data Structures & Problem Solving Using Java", Addison-Wesley, 2010.

CALIFICACIÓN DEL LABORATORIO

CRITERIO A SER EVALUADO		Nivel				
		A	B	C	D	NP
R1	Uso de estandares y buenas práticas de programación	1.0	0.8	0.5	0.3	0.0
R2	Informe en el formato correspondiente, cuidando los aspectos de redacción y orden, y que contiene en detalle las actividades y los ejercicios realizados.	3.0	2.8	1.5	0.8	0
R3	Correcta resolución de las actividades realizada en la sesión de laboratorio (4 pto. cada actividad)	8.0	3.0(*)	2.0(*)	1.0(*)	0.0(*)
R4	Correcta resolución de los ejercicios (2 pto. cada ejercicio)	8.0	1.5(*)	1.0(*)	0.5(*)	0.0(*)
	Total	**20**				

(*) : de cada actvidad o ejercicio