UNIVERSIDAD CATÓLICA DE SANTA MARÍA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LABORATORIO 05:

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
DURACIÓN DE LA PRACTICA

- Dos sesiones (04 horas)

IV
RECURSOS

- Equipos de cómputo.
- Lenguaje de programación Java y pseudocodigo.
- IDE de desarrollo. Eclipse, VCode.
- Herramientas de seguimiento de versiones: Git & GitHub.

V

MARCO TEÒRICO

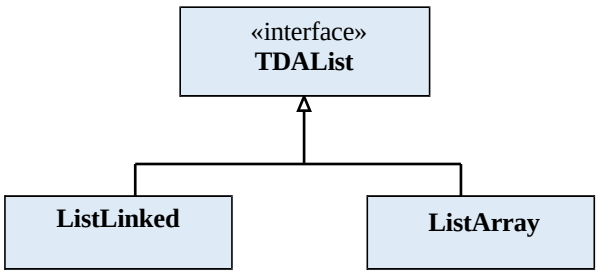
Definiciones generales

TDA Lista

Un TDA es un Tipo de Datos Abstracto, 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**).

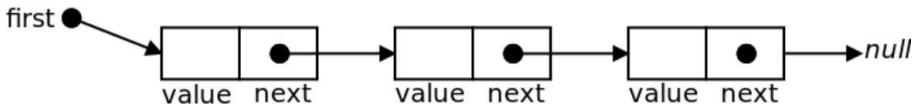
Todos excepto el último nodo, tienen una dirección en el campo **next**.

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

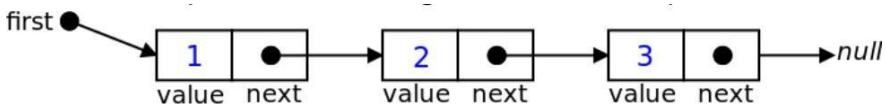
La diferencia principal de una lista enlazada con un arreglo es que en un arreglo los elementos deben de estar almacenados en posiciones continuas de la memoria, sin embargo, en una lista no.

Los siguientes son los términos importantes para entender el concepto de Lista Enlazada.

- Cada "**nodo**" de la secuencia contiene no solo el **valor** en esa posición, sino también una referencia al **siguiente** elemento de la secuencia
- Necesitamos saber dónde está el **primer** elemento de la lista
- El último elemento apunta al "**final**" de la lista (utilicemos **null** - *el objeto nulo* - para indicar el final de la lista)**.**

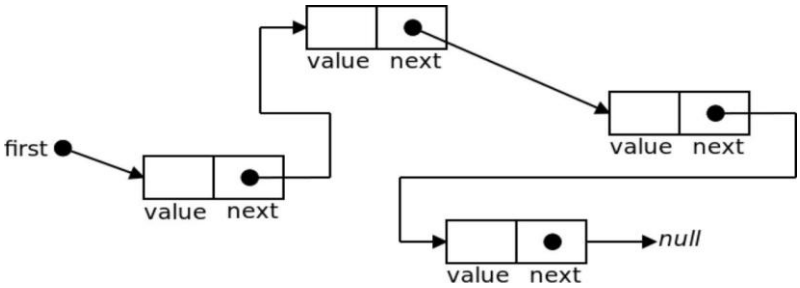


- En el atributo “valor” almacenamos el "contenido" de la secuencia. Por ejemplo, una lista enlazada de números enteros podría ser:



Representación en memoria

Las ubicaciones de memoria de lista no tienen que ser contiguas (porque cada nodo "apunta" al siguiente).



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.

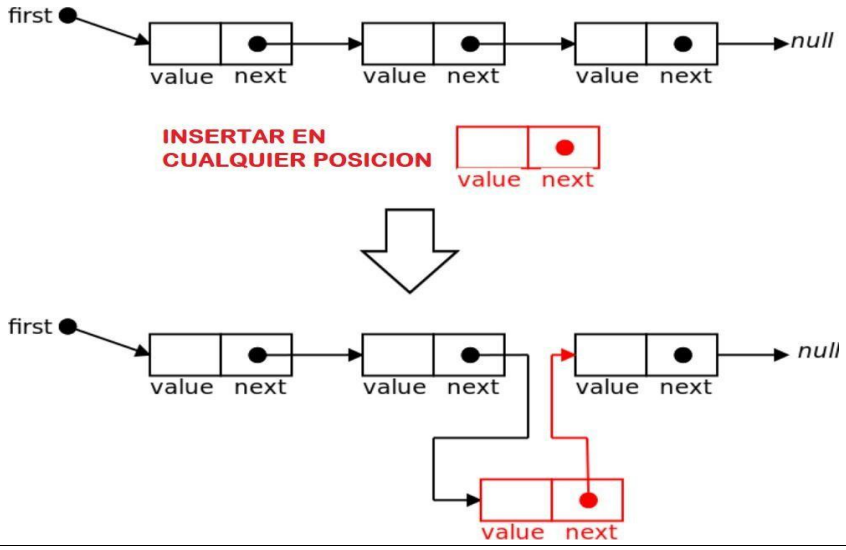
void insertFirst(x): inserta el nuevo elemento x al inicio de la lista

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

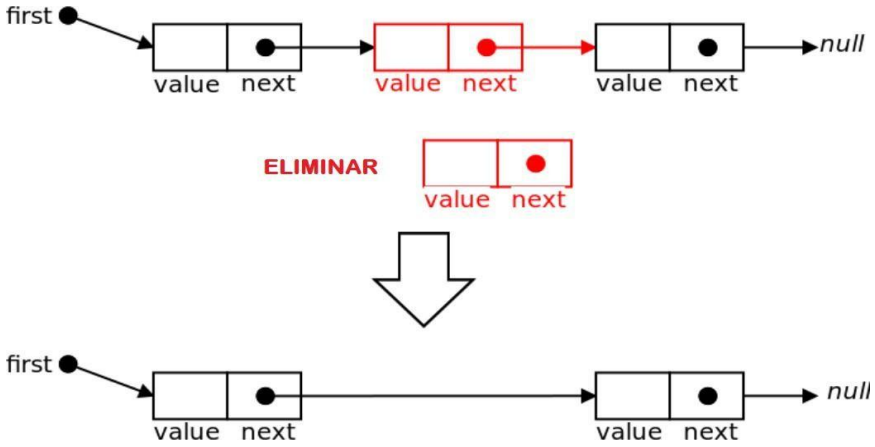
removeNode(x): elimina el elemento x de la lista.

Representación de inserción y eliminación

- **Inserción en cualquier posición**



- **Eliminando un elemento**



VI

ACTIVIDADES

El estudiante deberá implementar un **gestor de tareas** genérico usando tanto listas (Creadas por usted mismo) como listas enlazadas, aplicando operaciones fundamentales: insertar, eliminar, buscar, recorrer, contar y ordenar.

Contexto del problema:
Una empresa necesita una estructura para almacenar y gestionar tareas de diferentes tipos (pueden ser de tipo String, Integer, Tarea, etc.). Para ello, el estudiante debe:

1. Implementar una estructura de **lista enlazada genérica**.
2. Crear una clase genérica de **gestor de tareas** que pueda usar tanto las listas creadas como las Linked List.
3. Implementar operaciones clave para manipular las tareas.

Requisitos mínimos del proyecto:
1. Clase Node<T>
Implementa la estructura básica de nodo genérico.
2. Clase Tarea (si se quiere trabajar con objetos más complejos):

```
public class Tarea {
    private String titulo;
    private int prioridad;

    // Constructor, getters y toString
}
```

3. Clase GestorDeTareas<T> con métodos:

- void agregarTarea(T tarea) → agrega al final de la lista enlazada.
- boolean eliminarTarea(T tarea) → elimina la tarea si existe.
- boolean contieneTarea(T tarea) → busca una tarea.
- void imprimirTareas() → imprime todas las tareas.
- int contarTareas() → cuenta el total de tareas.
- T obtenerTareaMasPrioritaria() → si se trabaja con objetos Tarea, devuelve la de mayor prioridad.
- void invertirTareas() → invierte la lista enlazada.

4. Uso de una lista genérica adicional:
Simular una lista de tareas completadas usando una lista genérica.

Flujo sugerido del programa:

1. Crear una instancia de GestorDeTareas<T>.
2. Agregar tareas.
3. Eliminar alguna.
4. Imprimir todas las tareas actuales.
5. Verificar si cierta tarea existe.
6. Invertir la lista.
7. Transferir una tarea a una lista de “tareas completadas” (List<T>).
8. Mostrar ambas listas.

VII

EJERCICIOS

1. **Buscar un elemento genérico en una lista**

Crea un método genérico buscarElemento que reciba una lista de tipo genérico<T> y un valor T. El método debe retornar true si el valor está en la lista, o false en caso contrario.

2. **Invertir una lista genérica**

Escribe una función genérica invertirLista que reciba una lista<T> y retorne una nueva lista con los elementos en orden inverso.

3. **Insertar un nodo al final**

Implementa un método genérico insertarAlFinal que reciba un nodo head y un valor T, y agregue un nuevo nodo al final de la lista enlazada.

4. **Contar los nodos**

Escribe un método genérico contarNodos que reciba la cabeza de una lista enlazada y retorne el número total de nodos.

5. **Comparar dos listas**

Crea un método sonIguales que reciba dos listas enlazadas genéricas Node<T> y determine si contienen los mismos elementos en el mismo orden.

6. **Concatenar dos listas**

Desarrolla una función genérica concatenarListas que reciba dos listas enlazadas Node<T> y devuelva una nueva lista que combine ambas.

VIII

REFERENCIAS

- Weiss M., “Data Structures & Problem Solving Using Java”, Addison-Wesley, 2010.
- The Java™ Tutorials - https://docs.oracle.com/javase/tutorial/
- https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorith ms.htm

RUBRICA PARA CALIFICACION DE LABORATORIO

CRITERIO A SER EVALUADO		Nivel				
		A	B	C	D	NP
R1	Uso de estándares y buenas prácticas de programación	2.0	1.6	1.2	0.7	0.0
R2	Solución de las actividades a través de la definición y uso adecuado de List y Linked list, lo que se evidencia en el registro oportuno de los commits en los repositorios correspondientes, así como de las pruebas realizadas para su verificación.	4.0	3.2	2.4	1.4	0.0
R3	Solución de los ejercicios a través de la definición y uso adecuado de List y Linked list, lo que se evidencia en el registro oportuno de los commits en los repositorios correspondientes, así como de las pruebas realizadas para su verificación.	4.0	3.2	2.4	1.4	0.0
R4	Informe bien redactado, ordenado, bien detallado de las actividades y ejercicios resueltos, y de acuerdo a las normas establecidas. Mostrar evidencia suficiente de lo efectuado.	4.0	3.2	2.4	1.4	0.0
R5	Participación y contribución en el desarrollo del laboratorio del estudiante.	4.0	3.2	2.4	1.4	0.0
R6	Entrega oportuna de las actividades, ejercicios e informe.	2.0	1.6	1.2	0.7	0.0
Total		20				

(*) : de cada actvidad o ejercicio