


	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 1



GUÍA DE LABORATORIO

(formato docente)

INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	LISTAS ENLAZADAS				
NÚMERO DE PRÁCTICA:	04	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	TERCERO III
TIPO DE PRÁCTICA:	INDIVIDUAL	X			
	GRUPAL	—	MÁXIMO DE ESTUDIANTES	00	
FECHA INICIO:	26/05/2025	FECHA FIN:	30/05/2025	DURACIÓN:	90 minutos.
RECURSOS A UTILIZAR: <ul style="list-style-type: none"> • Github. • Lenguaje de Programación Java. • Ide Java Eclipse/Visual Studio Code. 					
DOCENTE(s): <ul style="list-style-type: none"> • Mg. Ing. Rene Alonso Nieto Valencia. 					

OBJETIVOS/TEMAS Y COMPETENCIAS	
OBJETIVOS: <ul style="list-style-type: none"> • Aprenda Listas Enlazadas. • Aplicar conceptos elementales de programación a resolver utilizando POO en problemas de algoritmos. • Desarrollar pruebas. 	
TEMAS: <ul style="list-style-type: none"> • Introducción. • Listas Enlazadas. • Operaciones de las Listas Enlazadas. 	
COMPETENCIAS	C.a
	C.b
	C.c
	C.d

CONTENIDO DE LA GUÍA

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 2</p>

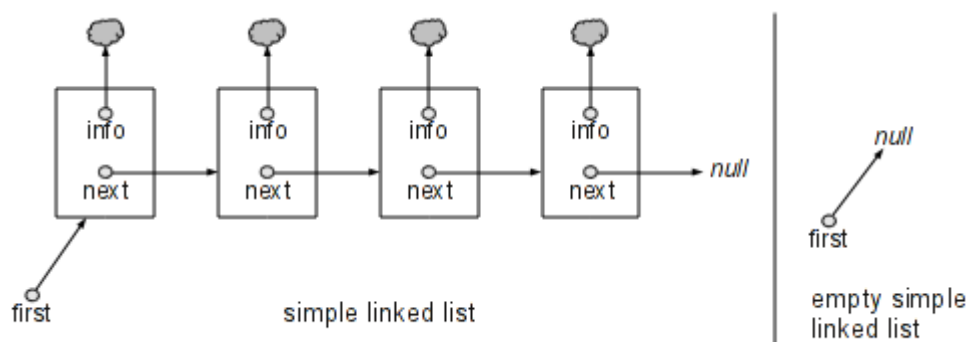
I. MARCO CONCEPTUAL

- <https://www.w3schools.com/java/>
- <https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-ide-enterprisejava-and-web-developers>
- <https://javaplot.panayotis.com/>
- <https://sourceforge.net/projects/gnuplot/files/gnuplot/5.4.3/>

Una **lista enlazada** es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

Una **lista enlazada** es un tipo de dato **autorreferenciado** porque contienen un puntero o enlace (en inglés link, del mismo significado) a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

Las listas enlazadas pueden ser implementadas en muchos lenguajes. Lenguajes imperativos u orientados a objetos tales como C o C++ y Java, respectivamente, disponen de referencias para crear listas enlazadas.



Lista enlazada es una colección de elementos.

Constructores de listas enlazadas



Volviendo al código fuente, podemos descubrir que **LinkedList** tiene dos constructores:

- `LinkedList()` sin parámetros se usa para construir una lista vacía.
- `LinkedList(Collection<? extends E> c)` es para crear una lista que contiene los elementos de la colección especificada, en orden, son devueltos por el iterador de la colección.

Declaración LinkedList

De hecho, una lista enlazada (Java o en cualquier otro lenguaje) consta de una secuencia de nodos. Cada nodo está diseñado para almacenar un objeto de un tipo definido al crear. Entonces, para crear **LinkedList**, el código Java es el siguiente:

```
LinkedList<Integer> myList = new LinkedList<>();
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 3</p>

Tenemos un objeto para mantener una secuencia de enteros y enlaces a los vecinos. Sin embargo, está vacío en este momento.

Operaciones principales de LinkedList

Como de costumbre, en el caso de las colecciones, puede colocar elementos en **LinkedList** (al final o en el medio), eliminarlos de allí y obtener un elemento por índice. Así que aquí están:

1. **add(E elemento)** Agrega el elemento especificado al final de esta lista;
2. **add(int index, elemento E)** Inserta el elemento en la posición especificada index ;
3. **get(int index)** Devuelve el elemento en la posición especificada en esta lista;
4. **remove(int index)** Elimina el elemento que está en la posición index;
5. **remove(Object o)** Elimina la primera aparición de ? o elemento de esta lista si está allí.
6. **remove()** Recupera y elimina el primer elemento de la lista.

II. EJERCICIO/PROBLEMA RESUELTO POR EL DOCENTE

En un editor Java, realizar la integración de los siguientes ejercicios, revisar y mostrar los resultados obtenidos y realizar una explicación del funcionamiento de forma concreta y clara.

1. **Ejercicio 1:** Crear una lista enlazada utilizando una clase **LinkedList** y una clase nodo e **ingresar** los elementos **1, 2, 3, 4, 5, 6, 7 y 8**.

```
import java.io.*;

// Un programa java para implementar una simple lista enlazada
public class LinkedList {
    Node head; // cabecera de la lista
    // Nodo de lista enlazada.
    // Esta clase interna se hace estática
    // para que main() pueda acceder a ella
    static class Node {
        int data;
        Node next;
        // Constructor
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // Método para insertar un nuevo nodo
    public static LinkedList insert(LinkedList list, int data) {
        // Crea un nuevo nodo con los datos dados
        Node new_node = new Node(data);
        // Si la lista enlazada está vacía,
        // entonces convierte el nuevo nodo en la cabeza
        if (list.head == null) {
            list.head = new_node;
        } else {
            // De lo contrario recorra hasta el último nodo
            // e inserte el nuevo nodo allí
            Node last = list.head;
            while (last.next != null) {
```

```
        last = last.next;
    }

    // Inserta el nuevo nodo al último nodo
    last.next = new_node;
}
// Retorna la lista desde la cabeza
return list;
}

// Metodo para imprimir la lista enlazada LinkedList.
public static void printList(LinkedList list) {
    Node currNode = list.head;
    System.out.print("LinkedList: ");
    // Recorre la lista enlazada (LinkedList)
    while (currNode != null) {
        // Imprime el dato en el nodo actual
        System.out.print(currNode.data + " ");
        // Va al siguiente nodo
        currNode = currNode.next;
    }
}

// Código principal
public static void main(String[] args) {
    /* Inicia con una lista vacia. */
    LinkedList list = new LinkedList();
    //
    // *****INSERCIÓN*****
    //
    // Inserta los valores
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);
    // Imprime la LinkedList
    printList(list);
}
}
```

2. Ejercicio 2: Implementa una lista enlazada donde se pueda **borrar** un **elemento** por el **elemento**.

```
import java.io.*;

// Un programa java para implementar
// Una simple lista enlazada
public class LinkedList {
    Node head; // Cabecera de la lista
    // Nodo de lista enlazada.
    // Esta clase interna se hace estática
    // para que main() pueda acceder a ella
    static class Node {
```

```
int data;
Node next;
// Constructor
Node(int d) {
    data = d;
    next = null;
}
}



// Método para insertar un nuevo nodo
public static LinkedList insert(LinkedList list, int data) {
    // Crea un nuevo nodo con los datos dados
    Node new_node = new Node(data);
    new_node.next = null;
    // Si la lista enlazada está vacía,
    // entonces convierte el nuevo nodo en la cabeza
    if (list.head == null) {
        list.head = new_node;
    } else {
        // De lo contrario recorra hasta el último nodo
        // e inserte el nuevo nodo allí
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }
        // Inserta el nuevo nodo al último nodo
        last.next = new_node;
    }
    // Retorna la lista desde la cabeza
    return list;
}

// Metodo para imprimir la lista enlazada
public static void printList(LinkedList list) {
    Node currNode = list.head;
    System.out.print("LinkedList: ");
    // Recorre la lista enlazada (LinkedList)
    while (currNode != null) {
        // Imprime el dato en el nodo actual
        System.out.print(currNode.data + " ");
        // Va al siguiente nodo
        currNode = currNode.next;
    }
    System.out.println();
}

// *****DELETION BY KEY*****
// Metodo para eliminar un nodo en LinkedList por dato
public static LinkedList deleteByKey(LinkedList list, int key) {
    // Aloja el nodo cabecera
    Node currNode = list.head, prev = null;
    //
    // CASO 1:
    // Si el nodo principal tiene el dato que se va a eliminar
    if (currNode != null && currNode.data == key) {
        list.head = currNode.next; // Cambia la cabeza
```

```
// Muestra el mensaje
System.out.println(key + " found and deleted");
// Retorna la lista actualizada
return list;
}
//
// CASO 2:
// Si el dato está en otro lugar que no sea la cabecera
//
// Busca el dato para ser borrado,
// realiza un seguimiento al nodo anterior
// ya que es necesario cambiar currNode.next
while (currNode != null && currNode.data != key) {
    // si currNode no tiene el dato
    // continua con el siguiente nodo
    prev = currNode;
    currNode = currNode.next;
}
// si el dato esuviera presente, sería el currNode
// Por lo tanto, el currNode no debe ser nulo
if (currNode != null) {
    // Desde que el dato está en currNode
    // Desenlaza currNode de la linked list
    prev.next = currNode.next;
    // muestra el mensaje
    System.out.println(key + " found and deleted");
}
//
// CASO 3: El dato no está presente
//
// Si la el dato no está presente en linked list
// el nodo actual podría ser nulo
if (currNode == null) {
    // Muestra el mensaje
    System.out.println(key + " not found");
}
// devuelve la lista
return list;
}

// *****Metodo principal*****
// método para crear una simple lista enlazada con n nodos
public static void main(String[] args) {
    /* Inicia con una lista vacia. */
    LinkedList list = new LinkedList();
    //
    // *****INSERCIÓN*****
    //
    // Inserta los valores
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);
}
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 7</p>

```

// Imprime la LinkedList
printList(list);
//
// *****Eliminación por dato *****
//
// Elimina el nodo con el valor 1
// En el caso el dato 1 está en ***la cabeza***
deleteByKey(list, 1);
// Imprime la LinkedList
printList(list);
// Borraremos el nodo con el valor 4
// En este caso el dato esta presente ***en el
// medio***
deleteByKey(list, 4);
// Imprime la LinkedList
printList(list);
// Borrar el nodo con el valor 10
// En este caso el dato esta ***no presente***
deleteByKey(list, 10);
// Imprime la LinkedList
printList(list);
}
}

```

3. Ejercicio 03: Implementa una lista enlazada donde se pueda borrar un elemento por posición.

```

import java.io.*;

// Un programa java para implementar
// Una simple lista enlazada
public class LinkedList {
    Node head; // cabecera de la lista
    // Nodo de lista enlazada.
    // Esta clase interna se hace estática
    // para que main() pueda acceder a ella
    static class Node {
        int data;
        Node next;
        // Constructor
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // Método para insertar un nuevo nodo
    public static LinkedList insert(LinkedList list, int data) {
        // Crea un nuevo nodo con los datos dados
        Node new_node = new Node(data);
        new_node.next = null;
        // Si la lista enlazada está vacía,
        // entonces convierte el nuevo nodo en la cabeza
        if (list.head == null) {
            list.head = new_node;
        } else {
            // De lo contrario recorra hasta el último nodo

```

```
// e inserte el nuevo nodo alli
Node last = list.head;
while (last.next != null) {
    last = last.next;
}
// Inserta el nuevo nodo al último nodo
last.next = new_node;
}
// Retorna la lista desde la cabeza
return list;
}

// Metodo para imprimir la lista enlazada LinkedList.
public static void printList(LinkedList list) {
    Node currNode = list.head;
    System.out.print("LinkedList: ");
    // Recorre la lista enlazada (LinkedList)
    while (currNode != null) {
        // Imprime el dato en el nodo actual
        System.out.print(currNode.data + " ");

        // Va al siguiente nodo
        currNode = currNode.next;
    }
    System.out.println();
}

// Metodo para eliminar un nodo en la LinkedList por POSITION
public static LinkedList deleteAtPosition(LinkedList list, int index) {
    // Guarda el nodo cabecera
    Node currNode = list.head, prev = null;
    //
    // CASE 1:
    // si el índice es 0, entonces el nodo principal debe ser
    // eliminado

    if (index == 0 && currNode != null) {
        list.head = currNode.next; // Cambia la cabecera
        // Muestra el mensaje
        System.out.println(index + " position element deleted");
        // Retorna la lista actualizada
        return list;
    }
    //
    // CASO 2:
    // Si el índice es mayor que 0 pero menor que el
    // tamaño de LinkedList
    //
    // El contador
    int counter = 0;
    // Conteo del índice a ser eliminado
    // seguimiento del nodo anterior
    // ya que es necesario cambiar currNode.next
    while (currNode != null) {

        if (counter == index) {
            // Dado el currNode es la posición requerida
```



```
// se desenlaza currNode de la lista enlazada
prev.next = currNode.next;



// Muestra el mensaje
System.out.println(index + " position element deleted");
break;
} else {
    // si la posición actual no es el índice
    // continua con el siguiente nodo
    prev = currNode;
    currNode = currNode.next;
    counter++;
}
}

// Si se encontro el elemento en la posición, debería estar
// en currNode Por lo tanto, currNode no debe ser
// null
//
// CASE 3: El índice es mayor que el tamaño de la
// LinkedList
//
// En el caso, el valor de currNode puede ser nulo
if (currNode == null) {
    // Muestra el mensaje
    System.out.println(index + " position element not found");
}

// retorna la lista
return list;
}

// *****METODO PRINCIPAL*****

// método para crear una simple lista enlazada con n nodos
public static void main(String[] args) {
    /* Inicia con una lista vacia. */
    LinkedList list = new LinkedList();
    //
    // *****INSERCIÓN*****
    //
    // Inserta los valores
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);
    // Imprime la LinkedList
    printList(list);
    //
    // *****BORRADO POR LA POSICIÓN *****
    //
    // Eliminar nodo en la posición 0
    // En este caso, la clave es ***la cabecera***
    deleteAtPosition(list, 0);
}
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 10</p>

```
// Imprime la LinkedList
printList(list);
// Eliminar nodo en la posición 2
// En este caso, la clave está presente ***en el
// medio***
deleteAtPosition(list, 2);
// Imprime la LinkedList
printList(list);
// Eliminar el nodo en la posición 10
// En este caso el dato esta ***no presente***
deleteAtPosition(list, 10);
// Imprime la LinkedList
printList(list);
}
}
```

4. Ejercicio 04: Implemente una **lista enlazada** donde se pueda **borrar** un **elemento** por el **elemento** y la **posición**.

```
import java.io.*;

// Un programa java para implementar
// Una simple lista enlazada
public class LinkedList {
    Node head; // cabecera de la lista
    // Nodo de lista enlazada.
    // Esta clase interna se hace estática
    // para que main() pueda acceder a ella
    static class Node {
        int data;
        Node next;
        // Constructor
        Node(int d) {
            data = d;
            next = null;
        }
    }

    // *****INSERCIÓN*****

    // Método para insertar un nuevo nodo
    public static LinkedList insert(LinkedList list, int data) {
        // Crea un nuevo nodo con los datos dados
        Node new_node = new Node(data);
        new_node.next = null;
        // Si la lista enlazada está vacía,
        // entonces convierte el nuevo nodo en la cabeza
        if (list.head == null) {
            list.head = new_node;
        } else {
            // De lo contrario recorra hasta el último nodo
            // e inserte el nuevo nodo allí
            Node last = list.head;
            while (last.next != null) {
                last = last.next;
            }
        }
    }
}
```

```
// Inserta el nuevo nodo al último nodo
last.next = new_node;
}
// Retorna la lista desde la cabeza
return list;
}

// *****Recorrido*****

// Metodo para imprimir la lista enlazada
public static void printList(LinkedList list) {
    Node currNode = list.head;
    System.out.print("\nLinkedList: ");
    // Recorre la lista enlazada (LinkedList)
    while (currNode != null) {
        // Imprime el dato en el nodo actual
        System.out.print(currNode.data + " ");
        // Va al siguiente nodo
        currNode = currNode.next;
    }
    System.out.println("\n");
}

// *****BORRADO POR DATO*****

// Metodo para eliminar un nodo en LinkedList por dato
public static LinkedList deleteByKey(LinkedList list, int key) {
    // Aloja el nodo cabecera
    Node currNode = list.head, prev = null;
    //
    // CASO 1:
    // Si el nodo principal tiene el dato que se va a eliminar
    if (currNode != null && currNode.data == key) {
        list.head = currNode.next; // Changed head
        // Muestra el mensaje
        System.out.println(key + " found and deleted");
        // Retorna la lista actualizada
        return list;
    }
    //
    // CASO 2:
    // Si el dato está en otro lugar que no sea la cabecera
    //
    // Busca el dato para ser borrado,
    // realiza un seguimiento al nodo anterior
    // ya que es necesario cambiar currNode.next
    while (currNode != null && currNode.data != key) {
        // si currNode no tiene el dato
        // continua con el siguiente nodo
        prev = currNode;
        currNode = currNode.next;
    }
    // si el dato esuviera presente, sería el currNode
    // Por lo tanto, el currNode no debe ser nulo
    if (currNode != null) {
        // Desde que el dato está en currNode
        // Desenlaza currNode de la linked list
    }
}
```

```
        prev.next = currNode.next;
        // muestra el mensaje
        System.out.println(key + " found and deleted");
    }
    //
    // CASO 3: El dato no está presente
    //
    // Si la el dato no está presente en linked list
    // el nodo actual podría ser nulo
    if (currNode == null) {
        // Muestra el mensaje
        System.out.println(key + " not found");
    }
    // devuelve la lista
    return list;
}
// *****Borrado por posicion*****

// Metodo para eliminar un nodo en la LinkedList por POSITION
public static LinkedList deleteAtPosition(LinkedList list, int index) {
    // Guarda el nodo cabecera
    Node currNode = list.head, prev = null;
    //
    // CASE 1:
    // si el índice es 0, entonces el nodo principal debe ser
    // eliminado
    if (index == 0 && currNode != null) {
        list.head = currNode.next; // Cambia la cabecera
        // Muestra el mensaje
        System.out.println(index + " position element deleted");
        // Retorna la lista actualizada
        return list;
    }
    //
    // CASO 2:
    // Si el índice es mayor que 0 pero menor que el
    // tamaño de LinkedList
    //
    // El contador
    int counter = 0;
    // Conteo del índice a ser eliminado
    // seguimiento del nodo anterior
    // ya que es necesario cambiar currNode.next
    while (currNode != null) {
        if (counter == index) {
            // Dado el currNode es la posición requerida
            // se desenlaza currNode de la lista enlazada
            prev.next = currNode.next;
            // Muestra el mensaje
            System.out.println(index + " position element deleted");
            break;
        } else {
            // si la posición actual no es el índice
            // continua con el siguiente nodo
            prev = currNode;
            currNode = currNode.next;
            counter++;
        }
    }
}
```

```
    }  
}  
  
// Si se encontro el elemento en la posición, debería ser  
// el currNode por lo tanto el currNode no debe ser  
// null  
// CASO 3: El dato no está presente  
//  
// Si la el dato no está presente en linked list  
// el nodo actual podría ser nulo  
if (currNode == null) {  
    // Muestra el mensaje  
    System.out.println(index + " position element not found");  
}  
// devuelve la lista  
return list;  
}  
  
// *****Metodo principal*****  
// método para crear una simple lista enlazada con n nodos  
public static void main(String[] args) {  
    /* Inicia con una lista vacia. */  
    LinkedList list = new LinkedList();  
    //  
    // *****INSERCIÓN*****  
    //  
    // Inserta los valores  
    list = insert(list, 1);  
    list = insert(list, 2);  
    list = insert(list, 3);  
    list = insert(list, 4);  
    list = insert(list, 5);  
    list = insert(list, 6);  
    list = insert(list, 7);  
    list = insert(list, 8);  
    // Imprime la LinkedList  
    printList(list);  
    //  
    // *****Eliminación por dato *****  
    //  
    // Elimina el nodo con el valor 1  
    // En el caso el dato 1 está en ***la cabeza***  
    deleteByKey(list, 1);  
    // Imprime la LinkedList  
    printList(list);  
    // Borramos el nodo con el valor 4  
    // En este caso el dato esta presente ***en el  
    // medio***  
    deleteByKey(list, 4);  
    // Imprime la LinkedList  
    printList(list);  
    // Borrar el nodo con el valor 10  
    // En este caso el dato esta ***no presente***  
    deleteByKey(list, 10);  
    // Imprime la LinkedList  
    printList(list);  
    //  
    // *****BORRADO POR LA POSICIÓN *****  
    //  
    //
```

```
// Eliminar nodo en la posición 0
// En este caso, la clave es ***la cabecera***
deleteAtPosition(list, 0);
// Imprime la LinkedList
printList(list);
// Eliminar nodo en la posición 2
// En este caso, la clave está presente ***en el
// medio***
deleteAtPosition(list, 2);
// Imprime la LinkedList
printList(list);
// Eliminar el nodo en la posición 10
// En este caso el dato esta ***no presente***
deleteAtPosition(list, 10);
// Imprime la LinkedList
printList(list);
}
}
```

5. **Ejercicio 05:** Crear una lista enlazada utilizando `java.util.linkedList`, que tenga los elementos **uno, dos, tres, cuatro y cinco**.

```
// Un programa java para añadir elementos a una LinkedList
import java.util.LinkedList;

public class AddElements {
    // Metodo principal
    public static void main(String[] args) {
        // Creando unaLinkedList
        LinkedList<String> l = new LinkedList<String>();
        // Añadiendo los elementos a la LinkedList usando el método add()
        l.add("Uno");
        l.add("Dos");
        l.add("Tres");
        l.add("Cuatro");
        l.add("Cinco");
        // Imprimiendo la LinkedList
        System.out.println(l);
    }
}
```

6. **Ejercicio 06:** Crear una lista enlazada utilizando la librería `java.util` que implemente el añadido de elementos, de letras del abecedario de la **A a la E** y también el borrado de elementos, por posición , por dato, que remueva el primero y el último.

```
// Programa que demuestra la
// implementación de la LinkedList
// class
import java.util.*;

public class GFG {
    public static void main(String args[]) {
        // Creando el objeto de la
        // clase lista enlazada
```

```
LinkedList<String> ll = new LinkedList<String>();  
// Añadido de elementos a la lista enlazada  
ll.add("A");  
ll.add("B");  
ll.addLast("C");  
ll.addFirst("D");  
ll.add(2, "E");  
System.out.println(ll);  
ll.remove("B");  
ll.remove(3);  
ll.removeFirst();  
ll.removeLast();  
System.out.println(ll);  
}  
}
```

7. **Ejercicio 07:** Crear una **lista enlazada** utilizando la librería **java.util** que implemente el **añadido** de elementos por **posición**.

```
// Programa java para añadir elementos  
// a la LinkedList  
  
import java.util.*;  
  
public class GFG {  
    public static void main(String args[]) {  
        LinkedList<String> ll = new LinkedList<>();  
        ll.add("Uno");  
        ll.add("Tres");  
        ll.add(1, "Dos");  
        System.out.println(ll);  
    }  
}
```

8. **Ejercicio 08:** Crear una **lista enlazada** utilizando la librería **java.util** que implemente el **cambio** de elemento usando el método **set()**.

```
// Programa en java para cambiar los elementos  
// en una LinkedList  
  
import java.util.*;  
  
public class GFG {  
    public static void main(String args[]) {  
        LinkedList<String> ll = new LinkedList<>();  
        ll.add("Uno");  
        ll.add("Dos");  
        ll.add(1, "Tres");  
        System.out.println("Initial LinkedList " + ll);  
        ll.set(1, "Cuatro");  
        System.out.println("Updated LinkedList " + ll);  
    }  
}
```

9. **Ejercicio 09:** Mostrar un programa en java que utilice la librería **java.util** para crear una **lista enlazada** y hacer el **recorrido** de sus **elementos**.

```
// Un programa java para iterar los elementos
// en una LinkedList

import java.util.*;

public class GFG {
    public static void main(String args[]) {
        LinkedList<String> ll = new LinkedList<>();
        ll.add("Uno");
        ll.add("Dos");
        ll.add(1, "Tres");
        // Usando el método Get en el
        // ciclo for
        for (int i = 0; i < ll.size(); i++) {

            System.out.print(ll.get(i) + " ");

        }
        System.out.println();
        // Using the for each loop
        for (String str : ll)
            System.out.print(str + " ");
    }
}
```

10. **Ejercicio 10:** Mostrar un programa en java que utilice la librería **java.util** y muestre el uso del método **toArray()**.

```
import java.util.*;

public class GFG2 {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<Integer>();
        list.add(123);
        list.add(12);
        list.add(11);
        list.add(1134);
        System.out.println("LinkedList: " + list);
        Object[] a = list.toArray();
        System.out.print("Después de convertir LinkedList a un Array: ");
        for (Object element : a)
            System.out.print(element + " ");
    }
}
```

11. **Ejercicio 11:** Mostrar un programa en java que utilice la librería **java.util** y muestre el uso del método **size()**.

```
import java.io.*;
import java.util.LinkedList;

public class GFG2 {
```



```
public static void main(String args[]) {  
    LinkedList<String> list = new LinkedList<String>();  
    list.add("Uno, Dos, Tres ");  
    list.add("Cuatro ");  
    // Mostrar el tamaño de la lista  
    System.out.println("El tamaño de la lista es: " + list.size());  
}  
}
```

12. Ejercicio 12: Mostrar un programa en java que utilice la librería **java.util** y muestre el uso del método **removeFirst()**.

```
import java.io.*;  
import java.util.LinkedList;  
  
public class GFG2 {  
    public static void main(String args[]) {  
        LinkedList<Integer> list = new LinkedList<Integer>();  
        list.add(10);  
        list.add(20);  
        list.add(30);  
        System.out.println("LinkedList:" + list);  
        System.out.println("El primer elemento removido es: " + list.removeFirst());  
        // Mostrando la lista final  
        System.out.println("Final LinkedList:" + list);  
    }  
}
```

13. Ejercicio 13: Mostrar un programa en java que utilice la librería **java.util** y muestre el uso del método **removeLast()**.

```
import java.io.*;  
import java.util.LinkedList;  
  
public class GFG2 {  
    public static void main(String args[]) {  
        LinkedList<Integer> list = new LinkedList<Integer>();  
        list.add(10);  
        list.add(20);  
        list.add(30);  
        System.out.println("LinkedList:" + list);  
        // Remueve la cola usando removeLast()  
        System.out.println("The last element is removed: " + list.removeLast());  
        // Muestra la lista final  
        System.out.println("Final LinkedList:" + list);  
        // Remueve el último elemento usando removeLast()  
        System.out.println("The last element is removed: " + list.removeLast());  
        // Mostrando la lista final  
        System.out.println("Final LinkedList:" + list);  
    }  
}
```

14. Ejercicio 14: Mostrar un programa en java que utilice la librería **java.util** y muestre el uso del método **addFirst()** y **addLast()**.

```
import java.util.LinkedList;

public class LinkedListExample {
    public static void main(String[] args) {
        // Crea una nueva linked list
        LinkedList<Integer> linkedList = new LinkedList<>();
        // Añade elementos a la lista enlazada
        linkedList.add(1);
        linkedList.add(2);
        linkedList.add(3);
        // Añade un elemento al principio de la lista enlazada
        linkedList.addFirst(0);
        // Añade un elemento al final de la lista enlazada
        linkedList.addLast(4);

        // Imprime los elementos de la lista enlazada
        for (int i : linkedList) {
            System.out.println(i);
        }
    }
}
```

III. EJERCICIOS/PROBLEMAS PROPUESTOS



De acuerdo a los ejercicios propuestos desarrollar los algoritmos y mostrar las siguientes indicaciones:

- Enunciado del ejercicio.
- Código en java desarrollado.
- Resultados obtenidos.
- Explicación breve y concreta del código implementado.

Referencia: <https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

Usando la clase genérica: `public class LinkedList<E>` usar métodos genéricos `public E Metodo()`

1. Implementar una **lista doblemente enlazada** que tenga los elementos del 1 al 10, usando la clase nodo en java.
2. Implementar una **lista circular** que tenga los elementos del 1 al 12 utilizando la clase nodo en.
3. Implementar una **lista doblemente enlazada** que tenga los elementos del 1 al 10, usando la librería java.util.
4. Implementar una **lista circular** que tenga los elementos del 1 al 12 utilizando la librería java.util.
5. Implementar una **lista enlazada simple** que tenga los elementos del 1 al 10, usando la clase nodo en java y los **métodos** vistos en los **ejercicios propuestos** (insert, printList, deleteByKey, deleteAtPosition, size, removeFirst, removelast, addFirst y addLast) y probar una clase **Principal** con un menú de opciones para probar los métodos.
6. Implementar una **lista doblemente enlazada** que tenga los elementos del 1 al 10, usando la clase nodo en java y modificar los **métodos** vistos en los **ejercicios propuestos** (insert, printList, deleteByKey,

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 19</p>

deleteAtPosition, size, removeFirst, removeLast, addFirst y addLast) y probar una clase **Principal** con un menú de opciones para probar los métodos.

7. Implementar una **lista circular** que tenga los elementos del 1 al 12 utilizando la clase nodo en java y modificar los **métodos** vistos en los **ejercicios propuestos** (insert, printList, deleteByKey, deleteAtPosition, size, removeFirst, removeLast, addFirst y addLast) y probar una clase **Principal** con un menú de opciones para probar los métodos.

IV. CUESTIONARIO

1. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.
2. ¿Es posible reutilizar la clase nodo para otras estructuras de datos?
3. ¿Qué tipo de dato es NULL en java?
4. ¿Cuáles son los beneficios de utilizar tipos genéricos en las listas enlazadas?

V. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS:

- Weiss M., Data Structures & Problem Solving Using Java, 2010, Addison-Wesley.
- Weiss M., Data Structures and Algorithms Analysis in Java, 2012, Addison-Wesley.
- Cormen T., Leiserson C., Rivest R., Stein C., Introduction to Algorithms, 2022, The MIT Press
- The Java™ Tutorials - <https://docs.oracle.com/javase/tutorial/>
- Sedgewick, R., Algorithms in Java, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, Part 5: Graph Algorithms, Addison-Wesley.
- Malik D., Data Structures Usign C++, 2003, Thomson Learning.
- Knuth D., The Art of Computer Programming, Vol. 1 y 3, Addison - Wesley.

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN

<p>TÉCNICAS: <i>Actividades Resueltas</i> <i>Ejercicios Propuestos</i></p>	<p>INSTRUMENTOS: <i>Rubricas</i></p>
---	--

CRITERIOS DE EVALUACIÓN

Los criterios de evaluación se encuentran en el silabo DUFA ANEXO en la sección EVOLUCIÓN CONTINUA