

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

INFORME DE LABORATORIO

(formato estudiante)

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
FECHA DE PRESENTACIÓN	31/05/2025	HORA DE PRESENTACIÓN	23:59:59		
INTEGRANTE (s): Aragón Carpio Fredy José				NOTA:	
DOCENTE(s): • Mg. Ing. Rene Alonso Nieto Valencia.					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS https://github.com/DARSEN12/Laboratorio_EDA_E.git</p> <p>a. Problemas Desarrollados</p> <p>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.</p> <pre> public class ED1 { Node head; static class Node { int data; Node next; Node(int d) { data = d; next = null; } } public static ED1 insert(ED1 list, int data) { </pre>

```
Node new_node = new Node(data) ;

if (list.head == null) {
    list.head = new_node;
} else {
    Node last = list.head;
    while (last.next != null) {
        last = last.next;
    }
    last.next = new_node;
}
return list;
}

public static void printList(ED1 list) {
    Node currNode = list.head;
    System.out.print("LinkedList: ");
    while (currNode != null) {
        System.out.print(currNode.data + " ");
        currNode = currNode.next;
    }
}

public static void main(String[] args) {
    ED1 list = new ED1();

    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);

    printList(list);
}
}
```

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

El código implementa una lista enlazada simple en Java, donde se definen nodos que almacenan datos enteros y un puntero al siguiente nodo. Permite insertar nuevos elementos al final de la lista y recorrerla para imprimir sus valores en orden. En el `main`, se crea una lista vacía, se insertan los números del 1 al 8 secuencialmente, y luego se muestra la lista completa en consola. Este código es la base para manipular estructuras dinámicas lineales, y puede extenderse para agregar operaciones como eliminación, búsqueda o inserción en posiciones específicas.

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

```
public class ED2 {
    Node head;

    static class Node {
        int data;
        Node next;

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

    public static ED2 insert(ED2 list, int data) {
        Node new_node = new Node(data);
        new_node.next = null;

        if (list.head == null) {
            list.head = new_node;
        } else {
            Node last = list.head;
            while (last.next != null) {
                last = last.next;
            }
            last.next = new_node;
        }
        return list;
    }
}
```

```
}

public static void printList(ED2 list) {
    Node currNode = list.head;
    System.out.print("LinkedList: ");
    while (currNode != null) {
        System.out.print(currNode.data + " ");
        currNode = currNode.next;
    }
    System.out.println();
}

public static ED2 deleteByKey(ED2 list, int key) {
    Node currNode = list.head, prev = null;

    if (currNode != null && currNode.data == key) {
        list.head = currNode.next;
        System.out.println(key + " found and deleted");
        return list;
    }

    while (currNode != null && currNode.data != key) {
        prev = currNode;
        currNode = currNode.next;
    }

    if (currNode != null) {
        prev.next = currNode.next;
        System.out.println(key + " found and deleted");
    }

    if (currNode == null) {
        System.out.println(key + " not found");
    }

    return list;
}
```

```
public static void main(String[] args) {  
    ED2 list = new ED2();  
  
    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);  
  
    printList(list);  
  
    deleteByKey(list, 1);  
    printList(list);  
  
    deleteByKey(list, 4);  
    printList(list);  
  
    deleteByKey(list, 10);  
    printList(list);  
}
```

Este código implementa una lista enlazada simple con operaciones básicas de inserción, impresión y eliminación de nodos por valor. Inicialmente, inserta los números del 1 al 8 en la lista, luego imprime el contenido. Después elimina nodos específicos (valor 1 en la cabeza, valor 4 en medio, y trata de eliminar el valor 10 que no existe), mostrando mensajes que confirman si la eliminación fue exitosa o si el valor no se encontró. Finalmente, imprime la lista tras cada eliminación para reflejar los cambios.

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

```
public class ED3 {  
    Node head;  
  
    static class Node {
```

```
int data;
Node next;

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

}

public static ED3 insert(ED3 list, int data) {
    Node new_node = new Node(data);
    new_node.next = null;

    if (list.head == null) {
        list.head = new_node;
    } else {
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = new_node;
    }
    return list;
}

public static void printList(ED3 list) {
    Node currNode = list.head;
    System.out.print("LinkedList: ");
    while (currNode != null) {
        System.out.print(currNode.data + " ");
        currNode = currNode.next;
    }
    System.out.println();
}

public static ED3 deleteAtPosition(ED3 list, int index) {
    Node currNode = list.head, prev = null;
```

```
if (index == 0 && currNode != null) {
    list.head = currNode.next;
    System.out.println(index + " position element deleted");
    return list;
}

int counter = 0;

while (currNode != null) {
    if (counter == index) {
        prev.next = currNode.next;
        System.out.println(index + " position element
deleted");
        break;
    } else {
        prev = currNode;
        currNode = currNode.next;
        counter++;
    }
}

if (currNode == null) {
    System.out.println(index + " position element not
found");
}

return list;
}

public static void main(String[] args) {
    ED3 list = new ED3();

    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);  
  
printList(list);  
  
deleteAtPosition(list, 0);  
printList(list);  
  
deleteAtPosition(list, 2);  
printList(list);  
  
deleteAtPosition(list, 10);  
printList(list);  
}  
}
```

Este código implementa una lista enlazada simple con operaciones para insertar nodos al final, imprimir la lista y eliminar nodos según una posición dada. Primero inserta los números del 1 al 8, imprime la lista, luego elimina el nodo en la posición 0 (la cabeza), imprime la lista nuevamente, elimina el nodo en la posición 2 (un nodo en medio), imprime otra vez, y finalmente intenta eliminar un nodo en la posición 10 que no existe, mostrando un mensaje de error. Así se demuestra la gestión dinámica y el manejo de casos en eliminación por índice.

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

```
public class ED4 {  
    Node head;  
  
    static class Node {  
        int data;  
        Node next;  
  
        Node(int d) {  
            data = d;  
            next = null;  
        }  
    }  
}
```



```
public static ED4 insert(ED4 list, int data) {
    Node new_node = new Node(data);
    new_node.next = null;

    if (list.head == null) {
        list.head = new_node;
    } else {
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }
        last.next = new_node;
    }
    return list;
}

public static void printList(ED4 list) {
    Node currNode = list.head;
    System.out.print("\nLinkedList: ");
    while (currNode != null) {
        System.out.print(currNode.data + " ");
        currNode = currNode.next;
    }
    System.out.println("\n");
}

public static ED4 deleteByKey(ED4 list, int key) {
    Node currNode = list.head, prev = null;

    if (currNode != null && currNode.data == key) {
        list.head = currNode.next;
        System.out.println(key + " found and deleted");
        return list;
    }

    while (currNode != null && currNode.data != key) {
        prev = currNode;
```

```
        currNode = currNode.next;
    }

    if (currNode != null) {
        prev.next = currNode.next;
        System.out.println(key + " found and deleted");
    }

    if (currNode == null) {
        System.out.println(key + " not found");
    }

    return list;
}

public static ED4 deleteAtPosition(ED4 list, int index) {
    Node currNode = list.head, prev = null;

    if (index == 0 && currNode != null) {
        list.head = currNode.next;
        System.out.println(index + " position element deleted");
        return list;
    }

    int counter = 0;

    while (currNode != null) {
        if (counter == index) {
            prev.next = currNode.next;
            System.out.println(index + " position element
deleted");
            break;
        } else {
            prev = currNode;
            currNode = currNode.next;
            counter++;
        }
    }
}
```

```
        if (currNode == null) {  
            System.out.println(index + " position element not  
found");  
        }  
  
        return list;  
    }  
  
    public static void main(String[] args) {  
        ED4 list = new ED4();  
  
        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);  
  
        printList(list);  
  
        deleteByKey(list, 1);  
        printList(list);  
  
        deleteByKey(list, 4);  
        printList(list);  
  
        deleteByKey(list, 10);  
        printList(list);  
  
        deleteAtPosition(list, 0);  
        printList(list);  
  
        deleteAtPosition(list, 2);  
        printList(list);  
    }  
}
```

```
deleteAtPosition(list, 10);  
printList(list);  
}  
}
```

Este código implementa una lista enlazada simple con las operaciones básicas de inserción, impresión y eliminación de nodos por valor o por posición. Primero inserta los números del 1 al 8, imprime la lista, luego elimina nodos específicos buscando por valor (clave) mostrando mensajes si se encuentra o no el dato, e imprime tras cada eliminación. Después realiza eliminaciones por posición (índice) también con mensajes y vuelve a imprimir la lista. Así, demuestra manejo completo de nodos dinámicos en lista enlazada, control de casos especiales y actualización correcta de punteros para mantener la integridad de la estructura.

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

```
import java.util.LinkedList;  
  
public class ED5 {  
    public static void main(String[] args) {  
        LinkedList<String> l = new LinkedList<>();  
        l.add("Uno");  
        l.add("Dos");  
        l.add("Tres");  
        l.add("Cuatro");  
        l.add("Cinco");  
  
        System.out.println(l);  
    }  
}
```

Crea una lista enlazada (`LinkedList`) de tipo `String` usando la clase genérica de la biblioteca estándar de Java (`java.util.LinkedList`). Inserta cinco elementos en orden mediante el método `add()`. Finalmente, imprime la lista completa en consola, mostrando los elementos en secuencia. Es un ejemplo básico de uso de estructuras de datos listas enlazadas con las

clases nativas de Java, que simplifican la gestión interna de nodos y punteros.

- 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.**

```
import java.util.LinkedList;

public class ED6 {
    public static void main(String[] args) {
        LinkedList<String> l1 = new LinkedList<>();

        l1.add("A");
        l1.add("B");
        l1.addLast("C");
        l1.addFirst("D");
        l1.add(2, "E");

        System.out.println(l1);

        l1.remove("B");
        l1.remove(3);
        l1.removeFirst();
        l1.removeLast();

        System.out.println(l1);
    }
}
```

El programa crea una `LinkedList` de cadenas y realiza diversas operaciones: inserta elementos al final (`add`), al inicio (`addFirst`), y en una posición específica (`add(2, "E")`). Luego imprime la lista. Después elimina elementos por valor (`remove("B")`), por índice (`remove(3)`), el primero (`removeFirst()`) y el último (`removeLast()`), y vuelve a imprimir la lista actualizada. Demuestra el manejo dinámico y versátil de listas enlazadas con la API estándar de Java.

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

```
import java.util.LinkedList;

public class ED7 {
    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);
    }
}
```

El programa crea una lista enlazada de cadenas y añade tres elementos. Los dos primeros se añaden al final en orden ("Uno" y "Tres"). Luego, inserta el elemento "Dos" en la posición 1 (segunda posición). Finalmente, imprime la lista mostrando el orden modificado: [Uno, Dos, Tres]. Este ejemplo ilustra la inserción de elementos en posiciones específicas usando la API estándar de Java para LinkedList.

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

```
import java.util.LinkedList;

public class ED8 {
    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);
    }
}
```

El programa crea una lista enlazada de cadenas, añade los elementos "Uno", "Dos" y "Tres" (insertado en la posición 1). Imprime la lista inicial. Luego, reemplaza el elemento en la posición 1 por "Cuatro" usando el método

`set()`. Finalmente, imprime la lista actualizada, demostrando cómo modificar elementos existentes en una `LinkedList` usando la API estándar de Java.

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.

```
import java.util.LinkedList;

public class ED9 {
    public static void main(String[] args) {
        LinkedList<String> ll = new LinkedList<>();
        ll.add("Uno");
        ll.add("Dos");
        ll.add(1, "Tres");

        // Iteración usando un ciclo for con get()
        for (int i = 0; i < ll.size(); i++) {
            System.out.print(ll.get(i) + " ");
        }
        System.out.println();

        // Iteración usando un for-each
        for (String str : ll) {
            System.out.print(str + " ");
        }
    }
}
```

El programa crea una lista enlazada de cadenas, inserta los elementos "Uno", "Dos" y "Tres" (en la posición 1). Luego, muestra dos formas de recorrer la lista: con un ciclo `for` que accede a cada elemento por índice mediante `get()`, y con un ciclo `for-each` que itera directamente sobre los elementos. Ambas impresiones muestran los elementos en orden, demostrando métodos estándar para iterar listas enlazadas en Java.

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

```
import java.util.LinkedList;

public class ED10 {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();
        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 + " ");
        }
    }
}
```

El programa crea una `LinkedList` de enteros, inserta varios valores, y luego imprime la lista completa. Posteriormente convierte la `LinkedList` a un arreglo (`Array`) usando el método `toArray()`, y recorre el arreglo para imprimir sus elementos. Esto demuestra cómo transformar una lista enlazada en un array para manipulación o acceso diferente dentro de Java.

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

```
import java.util.LinkedList;

public class ED11 {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();
        list.add("Uno, Dos, Tres ");
        list.add("Cuatro ");
    }
}
```



```
        System.out.println("El tamaño de la lista es: " +  
list.size());  
    }  
}
```

El programa crea una `LinkedList` de cadenas, añade dos elementos, y luego imprime el tamaño (número de elementos) de la lista usando el método `size()`. Es una demostración sencilla para obtener la cantidad de elementos almacenados en una lista enlazada de Java.

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

```
import java.util.LinkedList;  
  
public class ED12 {  
    public static void main(String[] args) {  
        LinkedList<Integer> list = new LinkedList<>();  
        list.add(10);  
        list.add(20);  
        list.add(30);  
  
        System.out.println("LinkedList: " + list);  
        System.out.println("El primer elemento removido es: " +  
list.removeFirst());  
        System.out.println("Final LinkedList: " + list);  
    }  
}
```

Crea una lista enlazada de enteros, añade tres elementos, imprime la lista completa, luego elimina y muestra el primer elemento de la lista usando `removeFirst()`. Finalmente, imprime la lista actualizada, mostrando que el primer elemento fue removido correctamente.

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

```
import java.util.LinkedList;  
  
public class ED13 {  
    public static void main(String[] args) {  
        LinkedList<Integer> list = new LinkedList<>();
```

```
list.add(10);
list.add(20);
list.add(30);

System.out.println("LinkedList: " + list);
System.out.println("The last element is removed: " +
list.removeLast());
System.out.println("Final LinkedList: " + list);
System.out.println("The last element is removed: " +
list.removeLast());
System.out.println("Final LinkedList: " + list);
}
```

El programa crea una lista enlazada de enteros, añade tres elementos, imprime la lista completa, luego elimina y muestra el último elemento dos veces consecutivas usando `removeLast()`. Después de cada eliminación, imprime la lista actualizada, demostrando la manipulación dinámica del final de la lista enlazada.

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 ED14 {
    public static void main(String[] args) {
        LinkedList<Integer> linkedList = new LinkedList<>();
        linkedList.add(1);
        linkedList.add(2);
        linkedList.add(3);
        linkedList.addFirst(0);
        linkedList.addLast(4);

        for (int i : linkedList) {
            System.out.println(i);
        }
    }
}
```

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

El programa crea una lista enlazada de enteros, añade elementos al final (1, 2, 3), luego añade un elemento al inicio (0) y otro al final (4). Finalmente, recorre la lista imprimiendo cada elemento, demostrando la inserción dinámica en posiciones iniciales y finales usando la API estándar de Java para LinkedList.

b. Problemas Propuestos

1. Implementar una lista doblemente enlazada que tenga los elementos del 1 al 10, usando la clase nodo en java.

```
public class EP1 {

    // Clase Nodo
    static class Node {
        int data;
        Node next;
        Node prev;

        Node(int data) {
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }

    // Clase de la lista doblemente enlazada
    static class DoublyLinkedList {
        Node head;
        Node tail;

        // Método para insertar un nuevo nodo al final de la lista
        public void insert(int data) {
            Node newNode = new Node(data);
            if (head == null) {
                head = tail = newNode;
            } else {
                tail.next = newNode;
                newNode.prev = tail;
            }
        }
    }
}
```

```
        tail = newNode;
    }
}



// Método para imprimir la lista de cabeza a cola
public void printList() {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}

// Método para imprimir la lista de cola a cabeza
public void printListReverse() {
    Node current = tail;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.prev;
    }
    System.out.println();
}
}

// Método principal
public static void main(String[] args) {
    DoublyLinkedList list = new DoublyLinkedList();

    // Insertando los números del 1 al 10
    for (int i = 1; i <= 10; i++) {
        list.insert(i);
    }

    // Imprimir la lista de cabeza a cola
    System.out.print("Lista desde la cabeza hasta la cola: ");
    list.printList();
}
```

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 21</p>

```
// Imprimir la lista de cola a cabeza
System.out.print("Lista desde la cola hasta la cabeza: ");
list.printListReverse();
}
}
```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión 4_c6ea4efa\bin' 'EP1'

Lista desde la cabeza hasta la cola: 1 2 3 4 5 6 7 8 9 10

Lista desde la cola hasta la cabeza: 10 9 8 7 6 5 4 3 2 1

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código:

Este código implementa una lista doblemente enlazada utilizando una clase `Node` que define la estructura de cada nodo con tres campos: `data` para almacenar el valor del nodo, `next` como puntero al siguiente nodo y `prev` como puntero al nodo anterior. La clase `DoublyLinkedList` contiene métodos para insertar nodos al final de la lista (`insert`), imprimir la lista en orden ascendente desde la cabeza hasta la cola (`printList`), e imprimir la lista en orden descendente desde la cola hasta la cabeza (`printListReverse`). En el método `main`, se crea una lista doblemente enlazada, se insertan los elementos del 1 al 10 y luego se imprime la lista en ambas direcciones. Esto permite navegar por la lista en ambas direcciones gracias a los punteros a los nodos anteriores y siguientes.

2. Implementar una lista circular que tenga los elementos del 1 al 12 utilizando la clase `nodo` en.

```
public class EP2 {

    // Clase Nodo: Define la estructura de un nodo
```

```
static class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

// Clase CircularLinkedList: Contiene los métodos para la lista
circular
static class CircularLinkedList {
    Node head = null;

    // Método para insertar un nuevo nodo en la lista circular
    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            newNode.next = head;
        } else {
            Node temp = head;
            while (temp.next != head) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.next = head;
        }
    }

    // Método para imprimir los elementos de la lista circular
    public void printList() {
        if (head == null) {
            System.out.println("La lista está vacía.");
            return;
        }
    }
}
```

```
Node temp = head;
do {
    System.out.print(temp.data + " ");
    temp = temp.next;
} while (temp != head);
System.out.println();
}
}

// Método principal
public static void main(String[] args) {
    CircularLinkedList list = new CircularLinkedList();

    for (int i = 1; i <= 12; i++) {
        list.insert(i);
    }



    System.out.print("Lista circular: ");
    list.printList();
}
}
```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión 4_c6ea4efa\bin' 'EP2'

Lista circular: 1 2 3 4 5 6 7 8 9 10 11 12

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código:

	<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-PRLE-001</p>	<p>Página: 24</p>

Este código implementa una lista circular utilizando una clase **Node** que define la estructura del nodo con un campo **data** para almacenar el valor y un puntero **next** al siguiente nodo. La clase **CircularLinkedList** contiene métodos para insertar nodos y para imprimir la lista circular. En el método **main**, se crea la lista circular, se insertan los números del 1 al 12 y luego se imprime la lista, recorriéndola desde el primer nodo hasta regresar al inicio, mostrando todos los elementos insertados.

3. Implementar una lista doblemente enlazada que tenga los elementos del 1 al 10, usando la librería `java.util`.

```
import java.util.LinkedList;

public class EP3 {
    public static void main(String[] args) {

        // Crear una lista doblemente enlazada usando la clase
LinkedList de java.util
        LinkedList<Integer> list = new LinkedList<>();

        // Insertar los elementos del 1 al 10 en la lista
        for (int i = 1; i <= 10; i++) {
            list.add(i);
        }

        // Imprimir la lista completa
        System.out.print("Lista doblemente enlazada: ");
        for (int i : list) {
            System.out.print(i + " ");
        }
        System.out.println();

        // Imprimir la lista en orden inverso (reverse) usando un
iterador
        System.out.print("Lista en orden inverso: ");
        for (int i = list.size() - 1; i >= 0; i--) {
            System.out.print(list.get(i) + " ");
        }
        System.out.println();
    }
}
```


	<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-PRLE-001</p>	<p>Página: 25</p>

```
}
}
```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión 4_c6ea4efa\bin' 'EP3'

Lista doblemente enlazada: 1 2 3 4 5 6 7 8 9 10

Lista en orden inverso: 10 9 8 7 6 5 4 3 2 1

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código por sección:

Este código utiliza la librería `java.util.LinkedList` para crear una lista doblemente enlazada, donde cada elemento tiene punteros al siguiente y al anterior. Se inserta una secuencia de elementos del 1 al 10 mediante el método `add()`. Luego, la lista se imprime de manera secuencial, recorriéndola de izquierda a derecha. También se muestra cómo recorrer e imprimir la lista en orden inverso utilizando el método `get()` para acceder a los elementos por índice, permitiendo la manipulación y el recorrido en ambas direcciones.

4. Implementar una lista circular que tenga los elementos del 1 al 12 utilizando la librería `java.util`.

```
import java.util.LinkedList;

public class EP4 {
    public static void main(String[] args) {

        LinkedList<Integer> list = new LinkedList<>();

        // Insertar los números del 1 al 12 en la lista
```

```
for (int i = 1; i <= 12; i++) {  
    list.add(i);  
}  
  
    // Hacer que la lista sea circular, conectando el último  
nodo con el primero  
list.addFirst(list.getLast());  
  
    // Imprimir la lista circular  
System.out.print("Lista circular: ");  
for (int i : list) {  
    System.out.print(i + " ");  
}  
System.out.println();  
  
    // Imprimir la lista nuevamente (muestra la repetición del  
primer elemento al final)  
System.out.print("Lista circular nuevamente: ");  
for (int i = 0; i < list.size(); i++) {  
    System.out.print(list.get(i) + " ");  
    if (i == list.size() - 1) {  
        break;  
    }  
}  
}
```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión 4_c6ea4efa\bin' 'EP4'

Lista circular: 12 1 2 3 4 5 6 7 8 9 10 11 12

Lista circular nuevamente: 12 1 2 3 4 5 6 7 8 9 10 11 12

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 27</p>

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código:

Este código utiliza la librería `java.util.LinkedList` para crear una lista circular que contiene los elementos del 1 al 12. Primero, inserta los elementos del 1 al 12 en la lista utilizando el método `add()`. Luego, para hacer la lista circular, agrega el primer elemento al final de la lista, creando un ciclo entre el último nodo y el primero. Finalmente, imprime la lista mostrando los elementos en el orden insertado, con el último elemento repitiéndose al final, lo que simula el comportamiento circular.

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.

```
import java.util.Scanner;

public class EP5 {

    static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    static class LinkedList {
        Node head;

        // Insertar al final
        public void insert(int data) {
            Node newNode = new Node(data);
```

```
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }

    // Imprimir la lista
    public void printList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    // Eliminar por clave
    public void deleteByKey(int key) {
        Node current = head;
        Node previous = null;

        if (current != null && current.data == key) {
            head = current.next;
            System.out.println(key + " found and deleted");
            return;
        }

        while (current != null && current.data != key) {
            previous = current;
            current = current.next;
        }

        if (current == null) {
```

```
        System.out.println(key + " not found");
        return;
    }

    previous.next = current.next;
    System.out.println(key + " found and deleted");
}

// Eliminar en una posición específica
public void deleteAtPosition(int position) {
    if (head == null) {
        return;
    }

    Node current = head;

    if (position == 0) {
        head = current.next;
        return;
    }

    for (int i = 0; current != null && i < position - 1;
i++) {
        current = current.next;
    }

    if (current == null || current.next == null) {
        return;
    }

    Node next = current.next.next;
    current.next = next;
}

// Obtener el tamaño de la lista
public int size() {
    int size = 0;
    Node current = head;
```

```
        while (current != null) {
            size++;
            current = current.next;
        }
        return size;
    }

    // Eliminar el primer elemento
    public void removeFirst() {
        if (head != null) {
            head = head.next;
        }
    }

    // Eliminar el último elemento
    public void removeLast() {
        if (head == null) return;
        if (head.next == null) {
            head = null;
            return;
        }

        Node secondLast = head;
        while (secondLast.next != null && secondLast.next.next
!= null) {
            secondLast = secondLast.next;
        }

        secondLast.next = null;
    }

    // Añadir al principio
    public void addFirst(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }
}
```

```
// Añadir al final
public void addLast(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();
    Scanner scanner = new Scanner(System.in);
    int choice, value, position;

    // Insertar elementos del 1 al 10
    for (int i = 1; i <= 10; i++) {
        list.insert(i);
    }

    while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Mostrar lista");
        System.out.println("2. Eliminar por clave");
        System.out.println("3. Eliminar por posición");
        System.out.println("4. Obtener tamaño de la lista");
        System.out.println("5. Eliminar primer elemento");
        System.out.println("6. Eliminar último elemento");
        System.out.println("7. Añadir primer elemento");
        System.out.println("8. Añadir último elemento");
        System.out.println("9. Salir");
        System.out.print("Seleccione una opción: ");
        choice = scanner.nextInt();
    }
}
```



```
switch (choice) {
    case 1:
        list.printList();
        break;
    case 2:
        System.out.print("Ingrese el valor a eliminar: ");

        value = scanner.nextInt();
        list.deleteByKey(value);
        break;
    case 3:
        System.out.print("Ingrese la posición a
eliminar: ");

        position = scanner.nextInt();
        list.deleteAtPosition(position);
        break;
    case 4:
        System.out.println("El tamaño de la lista es: "
+ list.size());
        break;
    case 5:
        list.removeFirst();
        break;
    case 6:
        list.removeLast();
        break;
    case 7:
        System.out.print("Ingrese el valor a añadir al
principio: ");

        value = scanner.nextInt();
        list.addFirst(value);
        break;
    case 8:
        System.out.print("Ingrese el valor a añadir al
final: ");

        value = scanner.nextInt();
        list.addLast(value);
}
```


	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 33</p>

```

        break;
    case 9:
        System.out.println("Saliendo...");
        scanner.close();
        return;
    default:
        System.out.println("Opción inválida.");
    }
}
}
}

```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión 4_c6ea4efa\bin' 'EP5'

Menu:

1. Mostrar lista
2. Eliminar por clave
3. Eliminar por posición
4. Obtener tamaño de la lista
5. Eliminar primer elemento
6. Eliminar último elemento
7. Añadir primer elemento
8. Añadir último elemento
9. Salir

Seleccione una opción: 1

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 34</p>

1 2 3 4 5 6 7 8 9 10

Menu:

1. **Mostrar lista**
2. **Eliminar por clave**
3. **Eliminar por posición**
4. **Obtener tamaño de la lista**
5. **Eliminar primer elemento**
6. **Eliminar último elemento**
7. **Añadir primer elemento**
8. **Añadir último elemento**
9. **Salir**

Seleccione una opción: 9

Saliendo...

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código:

Este código implementa una lista enlazada simple en Java con una clase **Node** que tiene dos campos: **data** (valor del nodo) y **next** (puntero al siguiente nodo). La clase **LinkedList** proporciona varios métodos como **insert**, **printList**, **deleteByKey**, **deleteAtPosition**, **size**, **removeFirst**, **removeLast**, **addFirst**, y **addLast** para manipular la lista. En el método **main**, se crea una instancia de la lista, se insertan los elementos del 1 al 10, y se muestra un menú que permite al usuario probar estos métodos para agregar, eliminar y consultar la lista.

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

```
import java.util.Scanner;

public class EP6 {

    static class Node {
        int data;
        Node next;
        Node prev;

        Node(int data) {
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }

    static class DoublyLinkedList {
        Node head;

        // Insertar un nuevo nodo al final
        public void insert(int data) {
            Node newNode = new Node(data);
            if (head == null) {
                head = newNode;
            } else {
                Node temp = head;
                while (temp.next != null) {
                    temp = temp.next;
                }
                temp.next = newNode;
                newNode.prev = temp;
            }
        }
    }
}
```

```
}

// Imprimir la lista desde la cabeza hasta la cola
public void printList() {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}

// Eliminar por clave
public void deleteByKey(int key) {
    Node current = head;
    while (current != null && current.data != key) {
        current = current.next;
    }
    if (current == null) {
        System.out.println(key + " not found");
        return;
    }

    if (current.prev != null) {
        current.prev.next = current.next;
    } else {
        head = current.next;
    }

    if (current.next != null) {
        current.next.prev = current.prev;
    }

    System.out.println(key + " found and deleted");
}

// Eliminar por posición
public void deleteAtPosition(int position) {
    if (head == null) {
```

```
        return;
    }

    Node current = head;
    int count = 0;

    if (position == 0) {
        head = current.next;
        if (head != null) {
            head.prev = null;
        }
        return;
    }

    while (current != null && count != position) {
        current = current.next;
        count++;
    }

    if (current == null) {
        System.out.println("Position " + position + " not found.");
        return;
    }

    if (current.next != null) {
        current.next.prev = current.prev;
    }

    if (current.prev != null) {
        current.prev.next = current.next;
    }

    System.out.println("Node at position " + position + " deleted.");
}

// Obtener el tamaño de la lista
```

```
public int size() {
    int size = 0;
    Node current = head;
    while (current != null) {
        size++;
        current = current.next;
    }
    return size;
}

// Eliminar el primer nodo
public void removeFirst() {
    if (head != null) {
        head = head.next;
        if (head != null) {
            head.prev = null;
        }
    }
}

// Eliminar el último nodo
public void removeLast() {
    if (head == null) return;

    if (head.next == null) {
        head = null;
        return;
    }

    Node last = head;
    while (last.next != null) {
        last = last.next;
    }

    last.prev.next = null;
}

// Añadir al principio
```

```
public void addFirst(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
    } else {
        newNode.next = head;
        head.prev = newNode;
        head = newNode;
    }
}

// Añadir al final
public void addLast(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.prev = temp;
    }
}

public static void main(String[] args) {
    DoublyLinkedList list = new DoublyLinkedList();
    Scanner scanner = new Scanner(System.in);
    int choice, value, position;

    // Insertar elementos del 1 al 10
    for (int i = 1; i <= 10; i++) {
        list.insert(i);
    }



    while (true) {
```

```
System.out.println("\nMenu:");
System.out.println("1. Mostrar lista");
System.out.println("2. Eliminar por clave");
System.out.println("3. Eliminar por posición");
System.out.println("4. Obtener tamaño de la lista");
System.out.println("5. Eliminar primer elemento");
System.out.println("6. Eliminar último elemento");
System.out.println("7. Añadir primer elemento");
System.out.println("8. Añadir último elemento");
System.out.println("9. Salir");
System.out.print("Seleccione una opción: ");
choice = scanner.nextInt();

switch (choice) {
    case 1:
        list.printList();
        break;
    case 2:
        System.out.print("Ingrese el valor a eliminar: ");

        value = scanner.nextInt();
        list.deleteByKey(value);
        break;
    case 3:
        System.out.print("Ingrese la posición a
eliminar: ");

        position = scanner.nextInt();
        list.deleteAtPosition(position);
        break;
    case 4:
        System.out.println("El tamaño de la lista es: "
+ list.size());
        break;
    case 5:
        list.removeFirst();
        break;
    case 6:
        list.removeLast();
```


	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 41</p>

```

        break;
    case 7:
        System.out.print("Ingrese el valor a añadir al
principio: ");

        value = scanner.nextInt();
        list.addFirst(value);
        break;
    case 8:
        System.out.print("Ingrese el valor a añadir al
final: ");

        value = scanner.nextInt();
        list.addLast(value);
        break;
    case 9:
        System.out.println("Saliendo...");
        scanner.close();
        return;
    default:
        System.out.println("Opción inválida.");
    }
}
}
}
}

```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión 4_c6ea4efa\bin' 'EP6'

Menu:

1. Mostrar lista
2. Eliminar por clave
3. Eliminar por posición

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 42</p>

4. Obtener tamaño de la lista

5. Eliminar primer elemento

6. Eliminar último elemento

7. Añadir primer elemento

8. Añadir último elemento

9. Salir

Seleccione una opción: 1

1 2 3 4 5 6 7 8 9 10

Menu:

1. Mostrar lista

2. Eliminar por clave

3. Eliminar por posición

4. Obtener tamaño de la lista

5. Eliminar primer elemento

6. Eliminar último elemento

7. Añadir primer elemento

8. Añadir último elemento

9. Salir

Seleccione una opción: 2

Ingrese el valor a eliminar: 4

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 43</p>

4 found and deleted

Menu:

- 1. Mostrar lista**
- 2. Eliminar por clave**
- 3. Eliminar por posición**
- 4. Obtener tamaño de la lista**
- 5. Eliminar primer elemento**
- 6. Eliminar último elemento**
- 7. Añadir primer elemento**
- 8. Añadir último elemento**
- 9. Salir**

Seleccione una opción: 1

1 2 3 5 6 7 8 9 10

Menu:

- 1. Mostrar lista**
- 2. Eliminar por clave**
- 3. Eliminar por posición**
- 4. Obtener tamaño de la lista**
- 5. Eliminar primer elemento**

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 44</p>

6. Eliminar último elemento

7. Añadir primer elemento

8. Añadir último elemento

9. Salir

Seleccione una opción: 9

Saliendo...

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código:

Este código implementa una lista doblemente enlazada utilizando una clase `Node` que tiene tres campos: `data` para almacenar el valor, `next` para apuntar al siguiente nodo y `prev` para apuntar al nodo anterior. La clase `DoublyLinkedList` ofrece varios métodos como `insert`, `printList`, `deleteByKey`, `deleteAtPosition`, `size`, `removeFirst`, `removeLast`, `addFirst` y `addLast` para manipular la lista. En el método `main`, se crea una instancia de la lista, se insertan los números del 1 al 10 y luego se muestra un menú interactivo para que el usuario pueda probar estos métodos, permitiendo agregar, eliminar y consultar la lista en diferentes maneras.

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.

```
import java.util.Scanner;

public class EP7 {

    static class Node {
        int data;
```

```
Node next;

Node(int data) {
    this.data = data;
    this.next = null;
}

static class CircularLinkedList {
    Node head;

    // Insertar un nodo al final de la lista circular
    public void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            newNode.next = head; // El primer nodo apunta a sí mismo
        } else {
            Node temp = head;
            while (temp.next != head) {
                temp = temp.next;
            }
            temp.next = newNode; // El último nodo apunta al nuevo nodo
            newNode.next = head; // El nuevo nodo apunta al primer nodo
        }
    }

    // Imprimir la lista circular
    public void printList() {
        if (head == null) {
            System.out.println("La lista está vacía.");
            return;
        }
        Node current = head;
        do {
```

```
        System.out.print(current.data + " ");
        current = current.next;
    } while (current != head);
    System.out.println();
}

// Eliminar un nodo por clave
public void deleteByKey(int key) {
    Node current = head;
    Node previous = null;

    if (current != null && current.data == key) {
        if (current.next == head) { // Si es el único nodo
            head = null;
        } else {
            previous = current;
            while (current.next != head) {
                previous = current;
                current = current.next;
            }
            previous.next = head; // El último nodo apunta
al primer nodo
        }

        System.out.println(key + " encontrado y
eliminado.");
        return;
    }

    while (current != null && current.data != key) {
        previous = current;
        current = current.next;
    }

    if (current == null) {
        System.out.println(key + " no encontrado.");
        return;
    }
}
```

```
        previous.next = current.next; // Desenlazar el nodo
        System.out.println(key + " encontrado y eliminado.");
    }

    // Eliminar un nodo en una posición específica
    public void deleteAtPosition(int position) {
        if (head == null) return;

        Node current = head;
        if (position == 0) {
            head = current.next;
            return;
        }

        int counter = 0;
        while (current != null && counter < position - 1) {
            current = current.next;
            counter++;
        }

        if (current == null || current.next == null) {
            System.out.println("Posición " + position + " no
encontrada.");
            return;
        }

        Node next = current.next.next;
        current.next = next;
        System.out.println("Nodo en la posición " + position + "
eliminado.");
    }

    // Obtener el tamaño de la lista circular
    public int size() {
        int size = 0;
        if (head == null) return size;

        Node current = head;
```

```
do {  
    size++;  
    current = current.next;  
} while (current != head);  
  
return size;  
}  
  
// Eliminar el primer nodo  
public void removeFirst() {  
    if (head != null) {  
        if (head.next == head) { // Si es el único nodo  
            head = null;  
        } else {  
            head = head.next;  
        }  
    }  
}  
  
// Eliminar el último nodo  
public void removeLast() {  
    if (head == null) return;  
  
    Node current = head;  
    while (current.next != head) {  
        current = current.next;  
    }  
  
    if (current == head) { // Si es el único nodo  
        head = null;  
    } else {  
        Node temp = head;  
        while (temp.next != current) {  
            temp = temp.next;  
        }  
        temp.next = head;  
    }  
}
```



```
// Añadir un nodo al principio
public void addFirst(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        newNode.next = head;
    } else {
        Node temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.next = head;
        head = newNode;
    }
}

// Añadir un nodo al final
public void addLast(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        newNode.next = head;
    } else {
        Node temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.next = head;
    }
}

public static void main(String[] args) {
    CircularLinkedList list = new CircularLinkedList();
    Scanner scanner = new Scanner(System.in);
}
```

```
int choice, value, position;



// Insertar los elementos del 1 al 12
for (int i = 1; i <= 12; i++) {
    list.insert(i);
}

while (true) {
    System.out.println("\nMenu:");
    System.out.println("1. Mostrar lista");
    System.out.println("2. Eliminar por clave");
    System.out.println("3. Eliminar por posición");
    System.out.println("4. Obtener tamaño de la lista");
    System.out.println("5. Eliminar primer elemento");
    System.out.println("6. Eliminar último elemento");
    System.out.println("7. Añadir primer elemento");
    System.out.println("8. Añadir último elemento");
    System.out.println("9. Salir");
    System.out.print("Seleccione una opción: ");
    choice = scanner.nextInt();

    switch (choice) {
        case 1:
            list.printList();
            break;
        case 2:
            System.out.print("Ingrese el valor a eliminar: ");

            value = scanner.nextInt();
            list.deleteByKey(value);
            break;
        case 3:
            System.out.print("Ingrese la posición a eliminar: ");

            position = scanner.nextInt();
            list.deleteAtPosition(position);
            break;
        case 4:
```

	UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 51

```

        System.out.println("El tamaño de la lista es: "
+ list.size());



        break;
    case 5:
        list.removeFirst();
        break;
    case 6:
        list.removeLast();
        break;
    case 7:
        System.out.print("Ingrese el valor a añadir al
principio: ");

        value = scanner.nextInt();
        list.addFirst(value);
        break;
    case 8:
        System.out.print("Ingrese el valor a añadir al
final: ");

        value = scanner.nextInt();
        list.addLast(value);
        break;
    case 9:
        System.out.println("Saliendo...");
        scanner.close();
        return;
    default:
        System.out.println("Opción inválida.");
    }
}
}
}

```

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.7-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Usuario\AppData\Roaming\Code\User\workspaceStorage\1fa

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 52</p>

22d956528dafec054c584b459a3ec\redhat.java\jdt_ws\Sesión
4_c6ea4efa\bin' 'EP7'

Menu:

1. Mostrar lista
2. Eliminar por clave
3. Eliminar por posición
4. Obtener tamaño de la lista
5. Eliminar primer elemento
6. Eliminar último elemento
7. Añadir primer elemento
8. Añadir último elemento
9. Salir

Seleccione una opción: 1

1 2 3 4 5 6 7 8 9 10 11 12

Menu:

1. Mostrar lista
2. Eliminar por clave
3. Eliminar por posición
4. Obtener tamaño de la lista
5. Eliminar primer elemento

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 53</p>

6. Eliminar último elemento

7. Añadir primer elemento

8. Añadir último elemento

9. Salir

Seleccione una opción: 9

Saliendo...

PS C:\Users\Usuario\Desktop\Estructura de Datos y Algoritmos\Laboratorio\Sesión 4>

Resumen de lo que hace el código:

Este código implementa una lista circular utilizando una clase **Node** que tiene dos campos: **data** para almacenar el valor del nodo y **next** para apuntar al siguiente nodo. La clase **CircularLinkedList** proporciona métodos para insertar nodos al final (**insert**), imprimir la lista circular (**printList**), eliminar nodos por clave (**deleteByKey**), eliminar nodos por posición (**deleteAtPosition**), obtener el tamaño de la lista (**size**), eliminar el primer (**removeFirst**) y último nodo (**removeLast**), y agregar nodos al principio (**addFirst**) o al final (**addLast**). En el método **main**, se crea una instancia de la lista, se insertan los elementos del 1 al 12 y luego se presenta un menú interactivo para probar los métodos de manipulación de la lista circular.

II. SOLUCIÓN DEL CUESTIONARIO

Dificultades al desarrollar los ejercicios propuestos: Al desarrollar los ejercicios propuestos, la principal dificultad fue la gestión de referencias y punteros entre nodos, especialmente en listas más complejas como las doblemente enlazadas o circulares. La documentación a veces no era lo suficientemente detallada, lo que llevó a investigar más sobre cómo manejar correctamente los nodos y las conexiones entre ellos. Además, la complejidad de implementar métodos como **deleteByKey** o **deleteAtPosition** requiere un manejo minucioso de las relaciones entre los nodos, lo que aumenta el riesgo de errores.

	<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-PRLE-001</p>	<p>Página: 54</p>

¿Es posible reutilizar la clase nodo para otras estructuras de datos?: Sí, la clase Node es completamente reutilizable en diversas estructuras de datos. Dado que la clase define una estructura básica con un campo data y un puntero (next o prev), puede ser utilizada en diferentes implementaciones como listas enlazadas, pilas, colas, y árboles binarios. Esto permite reutilizar la lógica de manejo de nodos y simplifica el desarrollo de diferentes estructuras de datos sin tener que escribir nuevas clases desde cero.

¿Qué tipo de dato es NULL en Java?: En Java, null es un valor especial asignable a variables de tipo de referencia, como objetos, arrays y clases. null indica que una variable de referencia no apunta a ningún objeto o instancia en memoria. No es un tipo de dato en sí mismo, sino un valor que se puede asignar a tipos de referencia, mientras que los tipos primitivos como int o boolean no pueden ser asignados con null.



¿Cuáles son los beneficios de utilizar tipos genéricos en las listas enlazadas?: Los tipos genéricos en las listas enlazadas permiten crear estructuras de datos flexibles y reutilizables para cualquier tipo de dato sin necesidad de escribir código redundante. Aseguran la seguridad de tipo en tiempo de compilación, evitando errores de conversión o ClassCastException en tiempo de ejecución. Además, los genéricos mejoran la claridad y legibilidad del código, eliminan la necesidad de castings y facilitan la creación de colecciones más seguras y flexibles.

III. CONCLUSIONES

Durante el desarrollo de los ejercicios en Java, se presentaron diversas dificultades, principalmente relacionadas con la comprensión del lenguaje, la falta de documentación en español y la complejidad de implementar estructuras de datos como listas genéricas. Sin embargo, estas dificultades brindaron la oportunidad de reconocer la importancia de entender las diferencias entre List y ArrayList, lo cual es fundamental para aplicar de manera correcta el principio de programación orientada a interfaces. Asimismo, se valoró el uso de clases genéricas debido a los beneficios que proporcionan en términos de reutilización de código, seguridad en los tipos y claridad en la implementación. En conjunto, estos aprendizajes contribuyen al fortalecimiento de las habilidades en diseño y desarrollo de soluciones eficientes y escalables dentro del marco de la ingeniería de software.

RETROALIMENTACIÓN GENERAL

REFERENCIAS Y BIBLIOGRAFÍA

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 55</p>