

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Forma :o: 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	01	
FECHA INICIO:	26/05/2025	FECHA FIN:	30/05/2025	DURACIÓN:	90 minutos.
<b>RECURSOS A UTILIZAR:</b> <ul style="list-style-type: none"> <li>Repositorio GITHUB: <a href="https://github.com/JosueClaudioQP/EDA-Lab">https://github.com/JosueClaudioQP/EDA-Lab</a></li> <li>Lenguaje de Programación Java.</li> <li>Ide Java Eclipse/Visual Studio Code.</li> </ul>					
<b>DOCENTE(s):</b> <ul style="list-style-type: none"> <li>Mg. Ing. Rene Alonso Nieto Valencia.</li> </ul>					
<b>ALUMNO:</b> <ul style="list-style-type: none"> <li>Quispe Paucar, Josué Claudio</li> </ul>					

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



UNIVERSIDAD NACIONAL DE SAN AGUSTIN  
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS  
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA



Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Aprobación: 2022/03/01

Código: GUIA-PRLD-001

Página: 2

- **PROBLEMAS RESUELTOS**

- **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.

```
1  package Laboratorio4;
2  import java.io.*;
3
4  // Un programa java para implementar una simple lista enlazada
5  public class LinkedList1 {
6      Node head; // cabecera de la lista
7      // Nodo de lista enlazada.
8      // Esta clase interna se hace estática
9      // para que main() pueda acceder a ella
10
11     static class Node {
12         int data;
13         Node next;
14
15         Node(int d) {
16             data = d;
17             next = null;
18         }
19     }
20
21     // Método para insertar un nuevo nodo
22     public static LinkedList1 insert(LinkedList1 list, int data) {
23         // Crea un nuevo nodo con los datos dados
24         Node new_node = new Node(data);
25         // Si la lista enlazada está vacía,
26         // entonces convierte el nuevo nodo en la cabeza
27         if (list.head == null) {
28             list.head = new_node;
29         } else {
30             // De lo contrario recorra hasta el último nodo
31             // e inserte el nuevo nodo allí
32             Node last = list.head;
33             while (last.next != null) {
34                 last = last.next;
35             }
36             // Inserta el nuevo nodo al último nodo
37             last.next = new_node;
38         }
39         // Retorna la lista desde la cabeza
40         return list;
41     }
42 }
```

```
43 // Metodo para imprimir la lista enlazada LinkedList.
44 public static void printList(LinkedList1 list) {
45     Node currNode = list.head;
46     System.out.print("LinkedList: ");
47     // Recorre la lista enlazada (LinkedList)
48     while (currNode != null) {
49         // Imprime el dato en el nodo actual
50         System.out.print(currNode.data + " ");
51         currNode = currNode.next;
52     }
53 }
54
55 // Código principal
56 public static void main(String[] args) {
57     /* Inicia con una lista vacia. */
58     LinkedList1 list = new LinkedList1();
59     //
60     // *****INSERCIÓN*****
61     //
62     // Inserta los valores
63     list = insert(list, 1);
64     list = insert(list, 2);
65     list = insert(list, 3);
66     list = insert(list, 4);
67     list = insert(list, 5);
68     list = insert(list, 6);
69     list = insert(list, 7);
70     list = insert(list, 8);
71     printList(list);
72     // Imprime la LinkedList
73 }
74 }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab> &
256caf1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\bin'
LinkedList: 1 2 3 4 5 6 7 8
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
1 package Laboratorio4;
2
3 public class LinkedList2 {
4     Node head; // cabecera de la lista
5     // Nodo de lista enlazada.
6     // Esta clase interna se hace estática
7     // para que main() pueda acceder a ella
8
9     static class Node {
10         int data;
11         Node next;
12
13         Node(int d) {
14             data = d;
15             next = null;
16         }
17     }
18 }
```

```
18
19 // Método para insertar un nuevo nodo
20 public static LinkedList2 insert(LinkedList2 list, int data) {
21     // Crea un nuevo nodo con los datos dados
22     Node new_node = new Node(data);
23     // Si la lista enlazada está vacía,
24     // entonces convierte el nuevo nodo en la cabeza
25     if (list.head == null) {
26         list.head = new_node;
27     } else {
28         // De lo contrario recorra hasta el último nodo
29         // e inserte el nuevo nodo allí
30         Node last = list.head;
31         while (last.next != null) {
32             last = last.next;
33         }
34         // Inserta el nuevo nodo al último nodo
35         last.next = new_node;
36     }
37     // Retorna la lista desde la cabeza
38     return list;
39 }
40
41 // Metodo para imprimir la lista enlazada LinkedList.
42 public static void printList(LinkedList2 list) {
43     Node currNode = list.head;
44     System.out.print("LinkedList: ");
45     // Recorre la lista enlazada (LinkedList)
46     while (currNode != null) {
47         // Imprime el dato en el nodo actual
48         System.out.print(currNode.data + " ");
49         currNode = currNode.next;
50     }
51 }
52
53 // *****DELETION BY KEY*****
54 // Metodo para eliminar un nodo en LinkedList por dato
55 public static LinkedList2 deleteByKey(LinkedList2 list, int key) {
56     // Aloja el nodo cabecera
57     Node currNode = list.head, prev = null;
58     //
59     // CASO 1:
60     // Si el nodo principal tiene el dato que se va a eliminar
61     if (currNode != null && currNode.data == key) {
62         list.head = currNode.next;
63         // Muestra el mensaje
64         System.out.println(key + " found and deleted");
65         // Retorna la lista actualizada
66         return list;
67     }
68 }
```

```
69 //
70 // CASO 2:
71 // Si el dato está en otro lugar que no sea la cabecera
72 //
73 // Busca el dato para ser borrado,
74 // realiza un seguimiento al nodo anterior
75 // ya que es necesario cambiar currNode.next
76 while (currNode != null && currNode.data != key) {
77     // si currNode no tiene el dato
78     // continua con el siguiente nodo
79     prev = currNode;
80     currNode = currNode.next;
81 }
82
83 // si el dato estuviera presente, sería el currNode
84 // Por lo tanto, el currNode no debe ser nulo
85 if (currNode != null) {
86     // Desde que el dato está en currNode
87     // Desenlaza currNode de la linked list
88     prev.next = currNode.next;
89     // muestra el mensaje
90     System.out.println(key + " found and deleted");
91 }
92
93 //
94 // CASO 3: El dato no está presente
95 //
96 // Si la el dato no está presente en linked list
97 // el nodo actual podría ser nulo
98 if (currNode == null) {
99     // Muestra el mensaje
100     System.out.println(key + " not found");
101 }
102
103 // devuelve la lista
104 return list;
105 }
106
107 // Código principal
108 public static void main(String[] args) {
109     /* Inicia con una lista vacía. */
110     LinkedList2 list = new LinkedList2();
111     //
112     // *****INSERCIÓN*****
113     //
114     // Inserta los valores
115     list = insert(list, 1);
116     list = insert(list, 2);
117     list = insert(list, 3);
118     list = insert(list, 4);
119     list = insert(list, 5);
120     list = insert(list, 6);
121     list = insert(list, 7);
122     list = insert(list, 8);
123     printList(list);
124     // Imprime la LinkedList
125
126     //
127     // *****Eliminación por dato *****
128     //
129     // Elimina el nodo con el valor 1
130     // En el caso el dato 1 está en ***la cabeza***
131     deleteByKey(list, 1);
132     // Imprime la LinkedList
133     printList(list);
134     // Borrarnos el nodo con el valor 4
135     // En este caso el dato esta presente ***en el
136     // medio***
137     deleteByKey(list, 4);
138     // Imprime la LinkedList
139     printList(list);
140     // Borrar el nodo con el valor 10
141     // En este caso el dato esta ***no presente***
142     deleteByKey(list, 10);
143     // Imprime la LinkedList
144     printList(list);
145 }
146 }
```

## Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab> & '256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\bin'
LinkedList: 1 2 3 4 5 6 7 8 1 found and deleted
LinkedList: 2 3 4 5 6 7 8 4 found and deleted
LinkedList: 2 3 5 6 7 8 10 not found
LinkedList: 2 3 5 6 7 8
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

- **Ejercicios 3:** Implementa una lista enlazada donde se pueda borrar un elemento por posición.

```
3 public class LinkedList3 {
4     Node head; // cabecera de la lista
5     // Nodo de lista enlazada.
6     // Esta clase interna se hace estática
7     // para que main() pueda acceder a ella
8
9     static class Node {
10         int data;
11         Node next;
12
13         Node(int d) {
14             data = d;
15             next = null;
16         }
17     }
18
19     // Método para insertar un nuevo nodo
20     public static LinkedList3 insert(LinkedList3 list, int data) {
21         // Crea un nuevo nodo con los datos dados
22         Node new_node = new Node(data);
23         // Si la lista enlazada está vacía,
24         // entonces convierte el nuevo nodo en la cabeza
25         if (list.head == null) {
26             list.head = new_node;
27         } else {
28             // De lo contrario recorra hasta el último nodo
29             // e inserte el nuevo nodo allí
30             Node last = list.head;
31             while (last.next != null) {
32                 last = last.next;
33             }
34             // Inserta el nuevo nodo al último nodo
35             last.next = new_node;
36         }
37         // Retorna la lista desde la cabeza
38         return list;
39     }
40 }
```

```
41 // Metodo para imprimir la lista enlazada LinkedList.
42 public static void printList(LinkedList3 list) {
43     Node currNode = list.head;
44     System.out.print(s:"LinkedList: ");
45     // Recorre la lista enlazada (LinkedList)
46     while (currNode != null) {
47         // Imprime el dato en el nodo actual
48         System.out.print(currNode.data + " ");
49         currNode = currNode.next;
50     }
51 }
52 }
```

```
53 // Metodo para eliminar un nodo en la LinkedList por POSITION
54 public static LinkedList3 deleteAtPosition(LinkedList3 list, int index) {
55     // Guarda el nodo cabecera
56     Node currNode = list.head, prev = null;
57     //
58     // CASE 1:
59     // si el índice es 0, entonces el nodo principal debe ser
60     // eliminado
61     if (index == 0 && currNode != null) {
62         list.head = currNode.next; // Cambia la cabecera
63         // Muestra el mensaje
64         System.out.println(index + " position element deleted");
65         // Retorna la lista actualizada
66         return list;
67     }
```

```
69 // CASO 2:
70 // Si el índice es mayor que 0 pero menor que el
71 // tamaño de LinkedList
72 //
73 // El contador
74 int counter = 0;
75 // Conteo del índice a ser eliminado
76 // seguimiento del nodo anterior
77 // ya que es necesario cambiar currNode.next
78 while (currNode != null) {
79     if (counter == index) {
80         // Dado el currNode es la posición requerida
81         // se desenlaza currNode de la lista enlazada
82         prev.next = currNode.next;
83         // Muestra el mensaje
84         System.out.println(index + " position element deleted");
85         break;
86     } else {
87         // si la posición actual no es el índice
88         // continua con el siguiente nodo
89         prev = currNode;
90         currNode = currNode.next;
91         counter++;
92     }
93 }
94 // Si se encontro el elemento en la posición, debería estar
95 // en currNode Por lo tanto, currNode no debe ser
96 // null
97 //
98 // CASE 3: El índice es mayor que el tamaño de la
99 // LinkedList
100 //
101 // En el caso, el valor de currNode puede ser nulo
102 if (currNode == null) {
103     // Muestra el mensaje
104     System.out.println(index + " position element not found");
105 }
106 // retorna la lista
107 return list;
108 }
```



```

112 public static void main(String[] args) {
113     /* Inicia con una lista vacia. */
114     LinkedList3 list = new LinkedList3();
115     //
116     // *****INSERCIÓN*****
117     //
118     // Inserta los valores
119     list = insert(list, data:1);
120     list = insert(list, data:2);
121     list = insert(list, data:3);
122     list = insert(list, data:4);
123     list = insert(list, data:5);
124     list = insert(list, data:6);
125     list = insert(list, data:7);
126     list = insert(list, data:8);
127     // Imprime la LinkedList
128     printList(list);
129     // Eliminar nodo en la posición 2
130     // En este caso, la clave está presente ***en el
131     // medio***
132     deleteAtPosition(list, index:2);
133     // Imprime la LinkedList
134     printList(list);
135     // Eliminar el nodo en la posición 10
136     // En este caso el dato esta ***no presente***
137     deleteAtPosition(list, index:10);
138     // Imprime la LinkedList
139     printList(list);
140 }
141 }

```

Resultados:

```

PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab> &
256cafc1e53eeal597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\bin
LinkedList: 1 2 3 4 5 6 7 8 2 position element deleted
LinkedList: 1 2 4 5 6 7 8 10 position element not found
LinkedList: 1 2 4 5 6 7 8
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>

```

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

```

57 // *****BORRADO POR DATO*****
58 // Metodo para eliminar un nodo en LinkedList por dato
59 public static LinkedList4 deleteByKey(LinkedList4 list, int key) {
60     // Aloja el nodo cabecera
61     Node currNode = list.head, prev = null;
62     //
63     // CASO 1:
64     // Si el nodo principal tiene el dato que se va a eliminar
65     if (currNode != null && currNode.data == key) {
66         list.head = currNode.next; // Changed head
67         // Muestra el mensaje
68         System.out.println(key + " found and deleted");
69         // Retorna la lista actualizada
70         return list;
71     }
72     //
73     // CASO 2:
74     // Si el dato está en otro lugar que no sea la cabecera
75     //
76     // Busca el dato para ser borrado,
77     // realiza un seguimiento al nodo anterior
78     // ya que es necesario cambiar currNode.next
79     while (currNode != null && currNode.data != key) {
80         // si currNode no tiene el dato
81         // continua con el siguiente nodo
82         prev = currNode;
83         currNode = currNode.next;
84     }

```

```

85 // si el dato estuviera presente, sería el currNode
86 // Por lo tanto, el currNode no debe ser nulo
87 if (currNode != null) {
88     // Desde que el dato está en currNode
89     // Desenlaza currNode de la linked list
90     prev.next = currNode.next;
91     // muestra el mensaje
92     System.out.println(key + " found and deleted");
93 }
94 //
95 // CASO 3: El dato no está presente
96 //
97 // Si la el dato no está presente en linked list
98 // el nodo actual podría ser nulo
99 if (currNode == null) {
100     // Muestra el mensaje
101     System.out.println(key + " not found");
102 }
103 // devuelve la lista
104 return list;
105 }

106 // *****Borrado por posicion*****
107 // Metodo para eliminar un nodo en la LinkedList por POSITION
108 public static LinkedList4 deleteAtPosition(LinkedList4 list, int index) {
109     // Guarda el nodo cabecera
110     Node currNode = list.head, prev = null;
111     //
112     // CASE 1:
113     // si el índice es 0, entonces el nodo principal debe ser
114     // eliminado
115     if (index == 0 && currNode != null) {
116         list.head = currNode.next; // Cambia la cabecera
117         // Muestra el mensaje
118         System.out.println(index + " position element deleted");
119         // Retorna la lista actualizada
120         return list;
121     }
122     //
123     // CASO 2:
124     // Si el índice es mayor que 0 pero menor que el
125     // tamaño de LinkedList
126     //
127     // El contador
128     int counter = 0;
129     // Conteo del índice a ser eliminado
130     // seguimiento del nodo anterior
131     // ya que es necesario cambiar currNode.next
132     while (currNode != null) {
133         if (counter == index) {
134             // Dado el currNode es la posición requerida
135             // se desenlaza currNode de la lista enlazada
136             prev.next = currNode.next;
137             // Muestra el mensaje
138             System.out.println(index + " position element deleted");
139             break;
140         } else {
141             // si la posición actual no es el índice
142             // continua con el siguiente nodo
143             prev = currNode;
144             currNode = currNode.next;
145             counter++;
146         }
147     }

```

```

148 // Si se encontro el elemento en la posición, debería ser
149 // el currNode Por lo tanto el currNode no debe ser
150 // null
151 // CASO 3: El dato no está presente
152 //
153 // Si la el dato no está presente en linked list
154 // el nodo actual podría ser nulo
155 if (currNode == null) {
156     // Muestra el mensaje
157     System.out.println(index + " position element not found");
158 }
159 // devuelve la lista
160 return list;
161 }

```

Método principal aplicando las funciones creadas:

```

179 // Imprime la LinkedList
180 printList(list);
181 //
182 // *****Eliminación por dato *****
183 //
184 // Elimina el nodo con el valor 1
185 // En el caso el dato 1 está en ***la cabeza***
186 deleteByKey(list, key:1);
187 // Imprime la LinkedList
188 printList(list);
189 // Borrarnos el nodo con el valor 4
190 // En este caso el dato esta presente ***en el
191 // medio***
192 deleteByKey(list, key:4);
193 // Imprime la LinkedList
194 printList(list);
195 // Borrar el nodo con el valor 10
196 // En este caso el dato esta ***no presente***
197 deleteByKey(list, key:10);
198 // Imprime la LinkedList
199 printList(list);
200 //
201 // *****BORRADO POR LA POSICIÓN *****
202 //
203 // Eliminar nodo en la posición 0
204 // En este caso, la clave es ***la cabecera***
205 deleteAtPosition(list, index:0);
206 // Imprime la LinkedList
207 printList(list);
208 // Eliminar nodo en la posición 2
209 // En este caso, la clave está presente ***en el
210 // medio***
211 deleteAtPosition(list, index:2);
212 // Imprime la LinkedList
213 printList(list);
214 // Eliminar el nodo en la posición 10
215 // En este caso el dato esta ***no presente***
216 deleteAtPosition(list, index:10);
217 // Imprime la LinkedList
218 printList(list);
219 }
220 }

```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256caf1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b

LinkedList: 1 2 3 4 5 6 7 8

1 found and deleted

LinkedList: 2 3 4 5 6 7 8

4 found and deleted

LinkedList: 2 3 5 6 7 8

10 not found

LinkedList: 2 3 5 6 7 8

0 position element deleted

LinkedList: 3 5 6 7 8

2 position element deleted

LinkedList: 3 5 7 8

10 position element not found

LinkedList: 3 5 7 8
```

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

```
Laboratorio4 > AddElements.java > ...
1 package Laboratorio4;
2 import java.util.LinkedList;
3
4 public class AddElements {
5     // Metodo principal
6     Run | Debug
7     public static void main(String[] args) {
8         // Creando unaLinkedList
9         LinkedList<String> l = new LinkedList<String>();
10        // Añadiendo los elementos a la LinkedList usando el método add()
11        l.add(e:"Uno");
12        l.add(e:"Dos");
13        l.add(e:"Tres");
14        l.add(e:"Cuatro");
15        l.add(e:"Cinco");
16        // Imprimiendo la LinkedList
17        System.out.println(l);
18    }
19 }
```

**Resultados:**

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256caf1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
[Uno, Dos, Tres, Cuatro, Cinco]
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

- **Ejercicio 6:** 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.

```
Laboratorio4 > GFG.java > ...
1  package Laboratorio4;
2  import java.util.*;
3
4  public class GFG {
5      public static void main(String[] args) {
6          // Creando el objeto de la
7          // clase lista enlazada
8          LinkedList<String> ll = new LinkedList<>();
9          // Añadido de elementos a la lista enlazada
10         ll.add(e:"A");
11         ll.add(e:"B");
12         ll.addLast(e:"C");
13         ll.addFirst(e:"D");
14         ll.add(index:2, element:"E");
15         System.out.println(ll);
16         ll.remove(o:"B");
17         ll.remove(index:3);
18         ll.removeFirst();
19         ll.removeLast();
20         System.out.println(ll);
21     }
22 }
23
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
[D, A, E, B, C]
[A]
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > GFG2.java > GFG2 > main(String[])
1  package Laboratorio4;
2  import java.util.*;
3
4  public class GFG2 {
5      public static void main(String[] args) {
6          LinkedList<String> ll = new LinkedList<>();
7          ll.add(e:"Uno");
8          ll.add(e:"Tres");
9          ll.add(index:1, element:"Dos");
10         System.out.println(ll);
11     }
12 }
13
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
[Uno, Dos, Tres]
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > GFG3.java > {} Laboratorio4
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG3 {
5      Run | Debug
6      public static void main(String[] args) {
7          LinkedList<String> ll = new LinkedList<>();
8          ll.add(e:"Uno");
9          ll.add(e:"Dos");
10         ll.add(index:1, element:"Tres");
11         System.out.println("Initial LinkedList " + ll);
12         ll.set(index:1, element:"Cuatro");
13         System.out.println("Updated LinkedList " + ll);
14     }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
Initial LinkedList [Uno, Tres, Dos]
Updated LinkedList [Uno, Cuatro, Dos]
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > GFG4.java > ...
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG4 {
5      Run | Debug
6      public static void main(String[] args) {
7          LinkedList<String> ll = new LinkedList<>();
8          ll.add(e:"Uno");
9          ll.add(e:"Dos");
10         ll.add(index:1, element:"Tres");
11         // Usando el método Get en el
12         // ciclo for
13         for (int i = 0; i < ll.size(); i++) {
14             System.out.print(ll.get(i) + " ");
15         }
16         System.out.println();
17         // Using the for each loop
18         for (String str : ll)
19             System.out.print(str + " ");
20     }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\t
Uno Tres Dos
Uno Tres Dos
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > GFG5.java > GFG5 > main(String[])
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG5 {
5      Run | Debug
6      public static void main(String[] args) {
7          |   LinkedList<Integer> list = new LinkedList<Integer>();
8          |   list.add(e:123);
9          |   list.add(e:12);
10         |   list.add(e:11);
11         |   list.add(e:1134);
12         |   System.out.println("LinkedList: " + list);
13         |   Object[] a = list.toArray();
14         |   System.out.print(s:"Después de convertir LinkedList a un Array: ");
15         |   for (Object element : a)
16         |       |   System.out.print(element + " ");
17         |   }
18     }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafcl53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
LinkedList: [123, 12, 11, 1134]
Después de convertir LinkedList a un Array: 123 12 11 1134
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > GFG6.java > GFG6 > main(String[])
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG6 {
5      Run | Debug
6      public static void main(String[] args) {
7          |   LinkedList<String> list = new LinkedList<String>();
8          |   list.add(e:"Uno, Dos, Tres ");
9          |   list.add(e:"Cuatro ");
10         |   // Mostrar el tamaño de la lista
11         |   System.out.println("El tamaño de la lista es: " + list.size());
12         |   }
13     }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafcl53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
El tamaño de la lista es: 2
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```



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

```
Laboratorio4 > GFG7.java > {} Laboratorio4
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG7 {
5      Run | Debug
6      public static void main(String[] args) {
7          LinkedList<Integer> list = new LinkedList<Integer>();
8          list.add(e:10);
9          list.add(e:20);
10         list.add(e:30);
11         System.out.println("LinkedList:" + list);
12         System.out.println("El primer elemento removido es: " + list.removeFirst());
13         // Mostrando la lista final
14         System.out.println("Final LinkedList:" + list);
15     }
16 }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
LinkedList:[10, 20, 30]
El primer elemento removido es: 10
Final LinkedList:[20, 30]
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

- **Ejercicio 13:**

```
Laboratorio4 > GFG8.java > {} Laboratorio4
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG8 {
5      Run | Debug
6      public static void main(String[] args) {
7          LinkedList<Integer> list = new LinkedList<Integer>();
8          list.add(e:10);
9          list.add(e:20);
10         list.add(e:30);
11         System.out.println("LinkedList:" + list);
12         // Remueve la cola usando removeLast()
13         System.out.println("The last element is removed: " + list.removeLast());
14         // Muestra la lista final
15         System.out.println("Final LinkedList:" + list);
16         // Remueve el último elemento usando removeLast()
17         System.out.println("The last element is removed: " + list.removeLast());
18         // Mostrando la lista final
19         System.out.println("Final LinkedList:" + list);
20     }
21 }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\
LinkedList:[10, 20, 30]
The last element is removed: 30
Final LinkedList:[10, 20]
The last element is removed: 20
Final LinkedList:[10]
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```



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

```
Laboratorio4 > GFG9.java > ...
1  package Laboratorio4;
2  import java.util.LinkedList;
3
4  public class GFG9 {
5      Run | Debug
6      public static void main(String[] args) {
7          // Crea una nueva linked list
8          LinkedList<Integer> linkedList = new LinkedList<>();
9          // Añade elementos a la lista enlazada
10         linkedList.add(e:1);
11         linkedList.add(e:2);
12         linkedList.add(e:3);
13         // Añade un elemento al principio de la lista enlazada
14         linkedList.addFirst(e:0);
15         // Añade un elemento al final de la lista enlazada
16         linkedList.addLast(e:4);
17         // Imprime los elementos de la lista enlazada
18         for (int i : linkedList) {
19             System.out.println(i);
20         }
21     }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
0
1
2
3
4
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

### 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.

Laboratorio4 &gt; ProblemasPropuestos &gt; Ejercicio1.java &gt; Ejercicio1 &gt; LinkedList&lt;E&gt;

```
1 package Laboratorio4.ProblemasPropuestos;
2
3 public class Ejercicio1 {
4
5     // Clase Nodo doblemente enlazado
6     static class NodoDoble<E> {
7         E dato;
8         NodoDoble<E> anterior;
9         NodoDoble<E> siguiente;
10
11         public NodoDoble(E dato) {
12             this.dato = dato;
13             this.anterior = null;
14             this.siguiente = null;
15         }
16     }
17
18     // Clase LinkedList genérica
19     static class LinkedList<E> {
20         private NodoDoble<E> cabeza;
21         private NodoDoble<E> cola;
22
23         // Insertar al final
24         public void insertar(E valor) {
25             NodoDoble<E> nuevo = new NodoDoble<>(valor);
26             if (cabeza == null) {
27                 cabeza = cola = nuevo;
28             } else {
29                 cola.siguiente = nuevo;
30                 nuevo.anterior = cola;
31                 cola = nuevo;
32             }
33         }
34
35         // Imprimir la lista
36         public void imprimir() {
37             NodoDoble<E> actual = cabeza;
38             while (actual != null) {
39                 System.out.print(actual.dato + " ");
40                 actual = actual.siguiente;
41             }
42             System.out.println();
43         }
44     }
45 }
```

```
44
45     // Método genérico de ejemplo
46     public E obtenerPrimero() {
47         return cabeza != null ? cabeza.dato : null;
48     }
49
50
51     // Clase principal
52     Run | Debug
53     public static void main(String[] args) {
54         LinkedList<Integer> lista = new LinkedList<>();
55         for (int i = 1; i <= 10; i++) {
56             lista.insertar(i);
57         }
58
59         System.out.println(x:"Lista doblemente enlazada (1 al 10):");
60         lista.imprimir();
61
62         System.out.println("Primer elemento (método genérico): " + lista.obtenerPrimero());
63     }
64 }
```

- **NodoDoble:** es la clase nodo para lista doblemente enlazada. Tiene referencias al nodo anterior y siguiente.
- **LinkedList:** clase genérica que permite operaciones sobre listas dobles.
- **insertar:** agrega elementos al final de la lista, actualizando las referencias de anterior y siguiente.
- **imprimir:** recorre desde el nodo cabeza hasta el cola, mostrando los elementos.
- **obtenerPrimero:** método genérico que devuelve el primer dato de la lista.
- **Principal:** prueba la implementación creando una lista del 1 al 10.

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
es' '-cp' 'C:\Users\JOSUE\AppData\Roaming\Code\User\workspa
Lista doblemente enlazada (1 al 10):
1 2 3 4 5 6 7 8 9 10
Primer elemento (método genérico): 1
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > ProblemasPropuestos > Ejercicio2.java > ...
1 package Laboratorio4.ProblemasPropuestos;
2
3 public class Ejercicio2 {
4
5     // Clase Nodo para lista circular
6     static class Nodo<E> {
7         E dato;
8         Nodo<E> siguiente;
9
10        public Nodo(E dato) {
11            this.dato = dato;
12            this.siguiente = null;
13        }
14    }
15 }
```

```
16 // Lista circular genérica
17 static class LinkedList<E> {
18     private Nodo<E> cabeza;
19
20     // Insertar al final
21     public void insertar(E valor) {
22         Nodo<E> nuevo = new Nodo<>(valor);
23         if (cabeza == null) {
24             cabeza = nuevo;
25             cabeza.siguiente = cabeza; // Enlaza a sí misma
26         } else {
27             Nodo<E> actual = cabeza;
28             while (actual.siguiente != cabeza) {
29                 actual = actual.siguiente;
30             }
31             actual.siguiente = nuevo;
32             nuevo.siguiente = cabeza;
33         }
34     }
35
36     // Imprimir lista circular
37     public void imprimir() {
38         if (cabeza == null) {
39             System.out.println(x:"Lista vacía");
40             return;
41         }
42
43         Nodo<E> actual = cabeza;
44         do {
45             System.out.print(actual.dato + " ");
46             actual = actual.siguiente;
47         } while (actual != cabeza);
48         System.out.println();
49     }
50
51     // Método genérico: obtener cabeza
52     public E obtenerPrimero() {
53         return cabeza != null ? cabeza.dato : null;
54     }
55 }
56
57 // Método principal
58 Run | Debug
59 public static void main(String[] args) {
60     LinkedList<Integer> lista = new LinkedList<>();
61     for (int i = 1; i <= 12; i++) {
62         lista.insertar(i);
63     }
64
65     System.out.println(x:"Lista circular (1 al 12):");
66     lista.imprimir();
67
68     System.out.println("Primer elemento (método genérico): " + lista.obtenerPrimero());
69 }
70 }
```

- Nodo: contiene el dato y el puntero al siguiente nodo.
- LinkedList:
  - La inserción al final encuentra el último nodo y lo enlaza con el nuevo nodo, que apunta a la cabeza.
  - La lista es circular, así que último.siguiente = cabeza.
- imprimir(): usa un bucle do-while para imprimir todos los elementos una vez.

- obtenerPrimero(): devuelve el valor del primer nodo, usando un método genérico.
- main: crea la lista con los números del 1 al 12 y prueba los métodos.

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
es' '-cp' 'C:\Users\JOSUE\AppData\Roaming\Code\User\workspa
Lista circular (1 al 12):
1 2 3 4 5 6 7 8 9 10 11 12
Primer elemento (método genérico): 1
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > ProblemasPropuestos > Ejercicio3.java > Ejercicio3 > main(String[])
1 package Laboratorio4.ProblemasPropuestos;
2 import java.util.LinkedList;
3
4 public class Ejercicio3 {
5
6     public static <E> E obtenerPrimero(LinkedList<E> lista) {
7         return lista.isEmpty() ? null : lista.getFirst();
8     }
9
10    Run | Debug
11    public static void main(String[] args) {
12        LinkedList<Integer> lista = new LinkedList<>();
13        for (int i = 1; i <= 10; i++) {
14            lista.add(i);
15        }
16
17        System.out.println(x:"Lista doblemente enlazada (1 al 10) usando java.util.LinkedList:");
18        for (Integer num : lista) {
19            System.out.print(num + " ");
20        }
21
22        System.out.println("\nPrimer elemento (método genérico): " + obtenerPrimero(lista));
23    }
24 }
```

- Usamos `java.util.LinkedList`, que es una lista doblemente enlazada por defecto.
- En el método `main` se insertan los números del 1 al 10 con `.add()`.
- El bucle `for-each` imprime todos los elementos.
- Se define un método genérico `obtenerPrimero` que devuelve el primer elemento de cualquier `LinkedList<E>`.

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab> c:: c
ers\JOSUE\AppData\Roaming\Code\User\workspaceStorage\f6daf3d90ac
Lista doblemente enlazada (1 al 10) usando java.util.LinkedList:
1 2 3 4 5 6 7 8 9 10
Primer elemento (método genérico): 1
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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

```
Laboratorio4 > ProblemasPropuestos > Ejercicio4.java > Ejercicio4 > main(String[])
1  package Laboratorio4.ProblemasPropuestos;
2  import java.util.LinkedList;
3
4  public class Ejercicio4 {
5
6      public static <E> E obtenerPrimero(LinkedList<E> lista) {
7          return lista.isEmpty() ? null : lista.getFirst();
8      }
9
10     public static <E> void imprimirCircular(LinkedList<E> lista, int repeticiones) {
11         if (lista.isEmpty()) {
12             System.out.println(x:"Lista vacía");
13             return;
14         }
15
16         int size = lista.size();
17         for (int i = 0; i < repeticiones; i++) {
18             System.out.print(lista.get(i % size) + " ");
19         }
20         System.out.println();
21     }
22
23     Run | Debug
24     public static void main(String[] args) {
25         LinkedList<Integer> lista = new LinkedList<>();
26
27         for (int i = 1; i <= 12; i++) {
28             lista.add(i);
29         }
30
31         System.out.println(x:"Lista circular simulada (1 al 12, dos vueltas):");
32         imprimirCircular(lista, repeticiones:24);
33
34         System.out.println("Primer elemento (método genérico): " + obtenerPrimero(lista));
35     }
36 }
```

- Usamos java.util.LinkedList para almacenar los números del 1 al 12.
- La función imprimirCircular() imprime los elementos en modo circular accediendo con i % size, repitiéndolos tantas veces como se indique.
- Se imprime la lista dos veces (24 elementos).
- obtenerPrimero() es un método genérico que devuelve el primer elemento de cualquier lista.

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
Lista circular simulada (1 al 12, dos vueltas):
1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12
Primer elemento (método genérico): 1
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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.

```
Laboratorio4 > ProblemasPropuestos > Ejercicio5.java > Ejercicio5 > main(String[])
1  package Laboratorio4.ProblemasPropuestos;
2  import java.util.Scanner;
3
4  public class Ejercicio5 {
5
6      static class Nodo<E> {
7          E dato;
8          Nodo<E> siguiente;
9
10         public Nodo(E dato) {
11             this.dato = dato;
12             this.siguiente = null;
13         }
14     }
15
16     static class LinkedList<E> {
17         private Nodo<E> cabeza;
18
19         public void insert(E dato) {
20             Nodo<E> nuevo = new Nodo<>(dato);
21             if (cabeza == null) {
22                 cabeza = nuevo;
23             } else {
24                 Nodo<E> actual = cabeza;
25                 while (actual.siguiente != null) {
26                     actual = actual.siguiente;
27                 }
28                 actual.siguiente = nuevo;
29             }
30         }
31
32         public void addFirst(E dato) {
33             Nodo<E> nuevo = new Nodo<>(dato);
34             nuevo.siguiente = cabeza;
35             cabeza = nuevo;
36         }
37
38         public void addLast(E dato) {
39             insert(dato);
40         }
41
42         public void deleteByKey(E key) {
43             if (cabeza == null) return;
44
45             if (cabeza.dato.equals(key)) {
46                 cabeza = cabeza.siguiente;
47                 return;
48             }
49
50             Nodo<E> actual = cabeza;
51             while (actual.siguiente != null && !actual.siguiente.dato.equals(key)) {
52                 actual = actual.siguiente;
53             }
54
55             if (actual.siguiente != null) {
56                 actual.siguiente = actual.siguiente.siguiente;
57             }
58         }
59
60         public void deleteAtPosition(int posicion) {
61             if (posicion < 0 || cabeza == null) return;
62
63             if (posicion == 0) {
64                 cabeza = cabeza.siguiente;
65                 return;
66             }
67         }
68     }
69 }
```

```
68     Nodo<E> actual = cabeza;
69     for (int i = 0; actual != null && i < posicion - 1; i++) {
70         actual = actual.siguiente;
71     }
72
73     if (actual != null && actual.siguiente != null) {
74         actual.siguiente = actual.siguiente.siguiente;
75     }
76 }
77
78 public void removeFirst() {
79     if (cabeza != null) {
80         cabeza = cabeza.siguiente;
81     }
82 }
83
84 public void removeLast() {
85     if (cabeza == null || cabeza.siguiente == null) {
86         cabeza = null;
87         return;
88     }
89
90     Nodo<E> actual = cabeza;
91     while (actual.siguiente.siguiente != null) {
92         actual = actual.siguiente;
93     }
94
95     actual.siguiente = null;
96 }
97
98 public void printList() {
99     Nodo<E> actual = cabeza;
100     while (actual != null) {
101         System.out.print(actual.dato + " ");
102         actual = actual.siguiente;
103     }
104     System.out.println();
105 }
106
```

```
107 public int size() {
108     int count = 0;
109     Nodo<E> actual = cabeza;
110     while (actual != null) {
111         count++;
112         actual = actual.siguiente;
113     }
114     return count;
115 }
116
```

```
118 public static void main(String[] args) {
119     LinkedList<Integer> lista = new LinkedList<>();
120     Scanner scanner = new Scanner(System.in);
121     int opcion;
122
123     for (int i = 1; i <= 10; i++) {
124         lista.insert(i);
125     }
126
```



```

127         do {
128             System.out.println(x:"\n--- Menú Lista Enlazada Simple ---");
129             System.out.println(x:"1. Mostrar lista");
130             System.out.println(x:"2. Insertar al inicio");
131             System.out.println(x:"3. Insertar al final");
132             System.out.println(x:"4. Eliminar por valor");
133             System.out.println(x:"5. Eliminar por posición");
134             System.out.println(x:"6. Eliminar primero");
135             System.out.println(x:"7. Eliminar último");
136             System.out.println(x:"8. Tamaño de la lista");
137             System.out.println(x:"9. Salir");
138             System.out.print(s:"Elige una opción: ");
139             opcion = scanner.nextInt();
140
141             switch (opcion) {
142                 case 1 -> lista.printList();
143                 case 2 -> {
144                     System.out.print(s:"Dato a insertar al inicio: ");
145                     int valor = scanner.nextInt();
146                     lista.addFirst(valor);
147                 }
148                 case 3 -> {
149                     System.out.print(s:"Dato a insertar al final: ");
150                     int valor = scanner.nextInt();
151                     lista.addLast(valor);
152                 }
153                 case 4 -> {
154                     System.out.print(s:"Valor a eliminar: ");
155                     int valor = scanner.nextInt();
156                     lista.deleteByKey(valor);
157                 }
158                 case 5 -> {
159                     System.out.print(s:"Posición a eliminar: ");
160                     int pos = scanner.nextInt();
161                     lista.deleteAtPosition(pos);
162                 }
163                 case 6 -> lista.removeFirst();
164                 case 7 -> lista.removeLast();
165                 case 8 -> System.out.println("Tamaño: " + lista.size());
166                 case 9 -> System.out.println(x:"Saliendo...");
167                 default -> System.out.println(x:"Opción inválida.");
168             }
169
170         } while (opcion != 9);
171
172         scanner.close();
173     }
174 }
175

```

- Se define un nodo genérico con un campo dato y siguiente.
- La lista tiene todos los métodos pedidos para modificar la lista simple.
- El menú permite:
  - Ver la lista
  - Insertar al inicio o final
  - Eliminar por valor o por posición
  - Remover primero o último
  - Ver el tamaño
- Inicializa la lista con los números del 1 al 10.

Resultados:

```
--- Menú Lista Enlazada Simple ---
```

1. Mostrar lista
2. Insertar al inicio
3. Insertar al final
4. Eliminar por valor
5. Eliminar por posición
6. Eliminar primero
7. Eliminar último
8. Tamaño de la lista
9. Salir

```
Elige una opción: | |
```

```
Elige una opción: 1
```

```
1 2 3 4 5 6 7 8 9 10
```

```
Elige una opción: 2
```

```
Dato a insertar al inicio: -1
```

```
-1 1 2 3 4 5 6 7 8 9 10
```

```
Elige una opción: 3
```

```
Dato a insertar al final: 13
```

```
Elige una opción: 1
```

```
-1 1 2 3 4 5 6 7 8 9 10 13
```

```
Elige una opción: 4
```

```
Valor a eliminar: -1
```

```
Elige una opción: 1
```

```
1 2 3 4 5 6 7 8 9 10 13
```

```
Elige una opción: 5
```

```
Posición a eliminar: 5
```

```
Elige una opción: 1
```

```
1 2 3 4 5 7 8 9 10 13
```

```
Elige una opción: 6
```

```
Elige una opción: 1
```

```
2 3 4 5 7 8 9 10 13
```

```
Elige una opción: 7
```

```
Elige una opción: 1
```

```
2 3 4 5 7 8 9 10
```

```
Elige una opción: 8
```

```
Tamaño: 8
```

```
Elige una opción: 9
```

```
Saliendo...
```

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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.

```
Laboratorio4 > ProblemasPropuestos > Ejercicio6.java > Ejercicio6 > r
1 package Laboratorio4.ProblemasPropuestos;
2 import java.util.Scanner;
3
4 public class Ejercicio6 {
5
6     static class Nodo<E> {
7         E dato;
8         Nodo<E> anterior;
9         Nodo<E> siguiente;
10
11         public Nodo(E dato) {
12             this.dato = dato;
13             this.anterior = null;
14             this.siguiente = null;
15         }
16     }
17 }
```

```
18     static class LinkedList<E> {
19         private Nodo<E> cabeza;
20         private Nodo<E> cola;
21
22         public void insert(E dato) {
23             Nodo<E> nuevo = new Nodo<>(dato);
24             if (cabeza == null) {
25                 cabeza = cola = nuevo;
26             } else {
27                 cola.siguiete = nuevo;
28                 nuevo.anterior = cola;
29                 cola = nuevo;
30             }
31         }
32
33         public void addFirst(E dato) {
34             Nodo<E> nuevo = new Nodo<>(dato);
35             if (cabeza == null) {
36                 cabeza = cola = nuevo;
37             } else {
38                 nuevo.siguiete = cabeza;
39                 cabeza.anterior = nuevo;
40                 cabeza = nuevo;
41             }
42         }
43
44         public void addLast(E dato) {
45             insert(dato);
46         }
47
48         public void deleteByKey(E key) {
49             Nodo<E> actual = cabeza;
50             while (actual != null && !actual.dato.equals(key)) {
51                 actual = actual.siguiete;
52             }
53
54             if (actual == null) return;
55
56             if (actual == cabeza) {
57                 cabeza = cabeza.siguiete;
58                 if (cabeza != null) cabeza.anterior = null;
59             } else if (actual == cola) {
60                 cola = cola.anterior;
61                 if (cola != null) cola.siguiete = null;
62             } else {
63                 actual.anterior.siguiete = actual.siguiete;
64                 actual.siguiete.anterior = actual.anterior;
65             }
66         }
67     }
```

```
68     public void deleteAtPosition(int pos) {
69         if (pos < 0) return;
70
71         Nodo<E> actual = cabeza;
72         for (int i = 0; actual != null && i < pos; i++) {
73             actual = actual.siguiente;
74         }
75
76         if (actual == null) return;
77
78         if (actual == cabeza) {
79             cabeza = cabeza.siguiente;
80             if (cabeza != null) cabeza.anterior = null;
81         } else if (actual == cola) {
82             cola = cola.anterior;
83             if (cola != null) cola.siguiente = null;
84         } else {
85             actual.anterior.siguiente = actual.siguiente;
86             actual.siguiente.anterior = actual.anterior;
87         }
88     }
89
90     public void removeFirst() {
91         if (cabeza != null) {
92             cabeza = cabeza.siguiente;
93             if (cabeza != null) cabeza.anterior = null;
94             else cola = null;
95         }
96     }
97
98     public void removeLast() {
99         if (cola != null) {
100             cola = cola.anterior;
101             if (cola != null) cola.siguiente = null;
102             else cabeza = null;
103         }
104     }
105
106     public int size() {
107         int count = 0;
108         Nodo<E> actual = cabeza;
109         while (actual != null) {
110             count++;
111             actual = actual.siguiente;
112         }
113         return count;
114     }
115
116     public void printList() {
117         Nodo<E> actual = cabeza;
118         while (actual != null) {
119             System.out.print(actual.dato + " ");
120             actual = actual.siguiente;
121         }
122         System.out.println();
123     }
124 }
```

```

126 public static void main(String[] args) {
127     LinkedList<Integer> lista = new LinkedList<>();
128     Scanner scanner = new Scanner(System.in);
129     int opcion;
130
131     for (int i = 1; i <= 10; i++) {
132         lista.insert(i);
133     }
134
135     do {
136         System.out.println(x:"\n--- Menú Lista Doblemente Enlazada ---");
137         System.out.println(x:"1. Mostrar lista");
138         System.out.println(x:"2. Insertar al inicio");
139         System.out.println(x:"3. Insertar al final");
140         System.out.println(x:"4. Eliminar por valor");
141         System.out.println(x:"5. Eliminar por posición");
142         System.out.println(x:"6. Eliminar primero");
143         System.out.println(x:"7. Eliminar último");
144         System.out.println(x:"8. Tamaño de la lista");
145         System.out.println(x:"9. Salir");
146         System.out.print(s:"Elige una opción: ");
147         opcion = scanner.nextInt();
148
149         switch (opcion) {
150             case 1 -> lista.printList();
151             case 2 -> {
152                 System.out.print(s:"Dato a insertar al inicio: ");
153                 int valor = scanner.nextInt();
154                 lista.addFirst(valor);
155             }
156             case 3 -> {
157                 System.out.print(s:"Dato a insertar al final: ");
158                 int valor = scanner.nextInt();
159                 lista.addLast(valor);
160             }
161             case 4 -> {
162                 System.out.print(s:"Valor a eliminar: ");
163                 int valor = scanner.nextInt();
164                 lista.deleteByKey(valor);
165             }
166             case 5 -> {
167                 System.out.print(s:"Valor a eliminar: ");
168                 int valor = scanner.nextInt();
169                 lista.deleteByKey(valor);
170             }
171             case 6 -> {
172                 System.out.print(s:"Posición a eliminar: ");
173                 int pos = scanner.nextInt();
174                 lista.deleteAtPosition(pos);
175             }
176             case 7 -> lista.removeFirst();
177             case 8 -> lista.removeLast();
178             case 9 -> System.out.println("Tamaño: " + lista.size());
179             default -> System.out.println(x:"Saliendo...");
180             default -> System.out.println(x:"Opción inválida.");
181         } while (opcion != 9);
182     } while (opcion != 9);
183     scanner.close();
184 }

```

- Implementa una lista doblemente enlazada con nodos que contienen referencia al nodo anterior y siguiente.
  - Métodos como deleteByKey y deleteAtPosition manejan los enlaces en ambas direcciones.
  - El menú permite probar cada una de las operaciones de forma interactiva.
- Resultados:

```
--- Menú Lista Doblemente Enlazada ---
1. Mostrar lista
2. Insertar al inicio
3. Insertar al final
4. Eliminar por valor
5. Eliminar por posición
6. Eliminar primero
7. Eliminar último
8. Tamaño de la lista
9. Salir
Elige una opción:
```

```
Elige una opción: 1
1 2 3 4 5 6 7 8 9 10
```

```
Elige una opción: 2
Dato a insertar al inicio: -1  -1 1 2 3 4 5 6 7 8 9 10
```

```
Elige una opción: 3      Elige una opción: 1
Dato a insertar al final: 13  -1 1 2 3 4 5 6 7 8 9 10 13
```

```
Elige una opción: 4      Elige una opción: 1
Valor a eliminar: -1  1 2 3 4 5 6 7 8 9 10 13
```

```
Elige una opción: 5      Elige una opción: 1
Posición a eliminar: 5  1 2 3 4 5 7 8 9 10 13
```

```
Elige una opción: 6      Elige una opción: 1
2 3 4 5 7 8 9 10 13
```

```
Elige una opción: 7      Elige una opción: 1
2 3 4 5 7 8 9 10
```

```
Elige una opción: 8
Tamaño: 8
```

```
Elige una opción: 9
Saliendo...
```

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

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.

```
Laboratorio4 > ProblemasPropuestos > Ejercicio7.java > Ejercicio7 > LinkedList<E> > insert(E)
1 package Laboratorio4.ProblemasPropuestos;
2 import java.util.Scanner;
3
4 public class Ejercicio7 {
5
6     static class Nodo<E> {
7         E dato;
8         Nodo<E> siguiente;
9
10        public Nodo(E dato) {
11            this.dato = dato;
12            this.siguiente = null;
13        }
14    }
15
16    static class LinkedList<E> {
17        private Nodo<E> ultimo;
18
19        public void insert(E dato) {
20            Nodo<E> nuevo = new Nodo<>(dato);
21            if (ultimo == null) {
22                nuevo.siguiente = nuevo;
23                ultimo = nuevo;
24            } else {
25                nuevo.siguiente = ultimo.siguiente;
26                ultimo.siguiente = nuevo;
27                ultimo = nuevo;
28            }
29        }
30    }
31 }
```

```
31 public void addFirst(E dato) {
32     Nodo<E> nuevo = new Nodo<>(dato);
33     if (ultimo == null) {
34         nuevo.siguiente = nuevo;
35         ultimo = nuevo;
36     } else {
37         nuevo.siguiente = ultimo.siguiente;
38         ultimo.siguiente = nuevo;
39     }
40 }
41
42 public void addLast(E dato) {
43     insert(dato);
44 }
45
46 public void deleteByKey(E key) {
47     if (ultimo == null) return;
48
49     Nodo<E> actual = ultimo.siguiente;
50     Nodo<E> anterior = ultimo;
51     do {
52         if (actual.dato.equals(key)) {
53             if (actual == ultimo && actual == ultimo.siguiente) {
54                 ultimo = null;
55             } else {
56                 anterior.siguiente = actual.siguiente;
57                 if (actual == ultimo) {
58                     ultimo = anterior;
59                 }
60             }
61             return;
62         }
63         anterior = actual;
64         actual = actual.siguiente;
65     } while (actual != ultimo.siguiente);
66 }
```

```
68 public void deleteAtPosition(int pos) {
69     if (ultimo == null || pos < 0) return;
70
71     Nodo<E> actual = ultimo.siguiente;
72     Nodo<E> anterior = ultimo;
73
74     for (int i = 0; i < pos; i++) {
75         anterior = actual;
76         actual = actual.siguiente;
77         if (actual == ultimo.siguiente) return;
78     }
79
80     if (actual == ultimo && actual == ultimo.siguiente) {
81         ultimo = null;
82     } else {
83         anterior.siguiente = actual.siguiente;
84         if (actual == ultimo) {
85             ultimo = anterior;
86         }
87     }
88 }
89
90 public void removeFirst() {
91     if (ultimo == null) return;
92
93     Nodo<E> primero = ultimo.siguiente;
94     if (primero == ultimo) {
95         ultimo = null;
96     } else {
97         ultimo.siguiente = primero.siguiente;
98     }
99 }
```

```
90     public void removeFirst() {
91         if (ultimo == null) return;
92
93         Nodo<E> primero = ultimo.siguiente;
94         if (primero == ultimo) {
95             ultimo = null;
96         } else {
97             ultimo.siguiente = primero.siguiente;
98         }
99     }
100
101     public void removeLast() {
102         if (ultimo == null) return;
103
104         Nodo<E> actual = ultimo.siguiente;
105         Nodo<E> anterior = ultimo;
106
107         if (actual == ultimo) {
108             ultimo = null;
109             return;
110         }
111
112         while (actual.siguiente != ultimo.siguiente) {
113             anterior = actual;
114             actual = actual.siguiente;
115         }
116
117         anterior.siguiente = ultimo.siguiente;
118         ultimo = anterior;
119     }
120
121     public int size() {
122         if (ultimo == null) return 0;
123         int count = 1;
124         Nodo<E> actual = ultimo.siguiente;
125         while (actual != ultimo) {
126             count++;
127             actual = actual.siguiente;
128         }
129         return count;
130     }
131
132     public void printList() {
133         if (ultimo == null) {
134             System.out.println(x:"Lista vacía");
135             return;
136         }
137         Nodo<E> actual = ultimo.siguiente;
138         do {
139             System.out.print(actual.dato + " ");
140             actual = actual.siguiente;
141         } while (actual != ultimo.siguiente);
142         System.out.println();
143     }
144 }
```



```

146 Run | Debug
147 public static void main(String[] args) {
148     LinkedList<Integer> lista = new LinkedList<>();
149     Scanner scanner = new Scanner(System.in);
150     int opcion;
151
152     for (int i = 1; i <= 12; i++) {
153         lista.insert(i);
154     }
155
156     do {
157         System.out.println(x:"\n--- Menú Lista Circular ---");
158         System.out.println(x:"1. Mostrar lista");
159         System.out.println(x:"2. Insertar al inicio");
160         System.out.println(x:"3. Insertar al final");
161         System.out.println(x:"4. Eliminar por valor");
162         System.out.println(x:"5. Eliminar por posición");
163         System.out.println(x:"6. Eliminar primero");
164         System.out.println(x:"7. Eliminar último");
165         System.out.println(x:"8. Tamaño de la lista");
166         System.out.println(x:"9. Salir");
167         System.out.print(s:"Elige una opción: ");
168         opcion = scanner.nextInt();
169
170         switch (opcion) {
171             case 1 -> lista.printList();
172             case 2 -> {
173                 System.out.print(s:"Dato a insertar al inicio: ");
174                 int valor = scanner.nextInt();
175                 lista.addFirst(valor);
176             }
177             case 3 -> {
178                 System.out.print(s:"Dato a insertar al final: ");
179                 int valor = scanner.nextInt();
180                 lista.addLast(valor);
181             }
182             case 4 -> {
183                 System.out.print(s:"Valor a eliminar: ");
184                 int valor = scanner.nextInt();
185                 lista.deleteByKey(valor);
186             }
187             case 5 -> {
188                 System.out.print(s:"Posición a eliminar: ");
189                 int pos = scanner.nextInt();
190                 lista.deleteAtPosition(pos);
191             }
192             case 6 -> lista.removeFirst();
193             case 7 -> lista.removeLast();
194             case 8 -> System.out.println("Tamaño: " + lista.size());
195             case 9 -> System.out.println(x:"Saliendo...");
196             default -> System.out.println(x:"Opción inválida.");
197         }
198     } while (opcion != 9);
199
200     scanner.close();
201 }
202
203

```

- La lista es circular, es decir, el último nodo apunta nuevamente al primero.
- El nodo ultimo es la clave para controlar el ciclo.
- Se puede recorrer y modificar sin romper el ciclo.
- El menú permite probar todos los métodos.

Resultados:

```
--- Menú Lista Circular ---
```

```
1. Mostrar lista
2. Insertar al inicio
3. Insertar al final
4. Eliminar por valor
5. Eliminar por posición
6. Eliminar primero
7. Eliminar último
8. Tamaño de la lista
9. Salir
Elige una opción: |
```

```
Elige una opción: 1
1 2 3 4 5 6 7 8 9 10
```

```
Elige una opción: 2
Dato a insertar al inicio: -1 -1 1 2 3 4 5 6 7 8 9 10
```

```
Elige una opción: 3
Dato a insertar al final: 13 -1 1 2 3 4 5 6 7 8 9 10 13
```

```
Elige una opción: 4
Valor a eliminar: -1 1 2 3 4 5 6 7 8 9 10 13
```

```
Elige una opción: 5
Posición a eliminar: 5 1 2 3 4 5 7 8 9 10 13
```

```
Elige una opción: 6
2 3 4 5 7 8 9 10 13
```

```
Elige una opción: 7
2 3 4 5 7 8 9 10
```

```
Elige una opción: 8
Tamaño: 8
```

```
Elige una opción: 9
Saliendo...
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
```

#### IV. CUESTIONARIO

1. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.

Una de las dificultades fue asegurarse de manejar correctamente los enlaces entre nodos, especialmente en listas doblemente enlazadas y listas circulares, ya que un error puede romper la estructura completa. Otra complicación fue mantener el código limpio y reutilizable al implementar una clase genérica. Además, aunque Java tiene una amplia documentación, puede resultar confuso encontrar ejemplos específicamente orientados a estructuras de datos genéricas sin el uso de `java.util`, lo que puede representar un reto adicional para estudiantes o desarrolladores que recién inician en programación con estructuras personalizadas.

2. ¿Es posible reutilizar la clase nodo para otras estructuras de datos?

Sí, la clase nodo puede ser reutilizada para otras estructuras de datos como pilas, colas, árboles, grafos, o incluso tablas hash, dependiendo del diseño. Su flexibilidad radica en que simplemente representa un contenedor de datos con referencias a otros nodos, por lo tanto, con pequeñas modificaciones (como

	<p align="center"><b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b></p>	
<b>Forma :o:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
<b>Aprobación:</b> 2022/03/01	<b>Código:</b> GUIA-PRLD-001	<b>Página:</b> 35

incluir más referencias o campos adicionales), se puede adaptar fácilmente a diferentes estructuras. Usar una clase nodo genérica también facilita su integración con distintos tipos de datos sin tener que crear una nueva clase para cada caso.

**3. ¿Qué tipo de dato es NULL en java?**

En Java, null no es un tipo de dato, sino un literal que representa la ausencia de valor o de referencia a un objeto en memoria. Es utilizado para indicar que una variable de tipo objeto no apunta a ninguna instancia. No puede asignarse a tipos primitivos como int o boolean, pero sí a cualquier variable de tipo referencia, como String, Nodo, List, etc. Intentar acceder a métodos o atributos de una variable que tiene el valor null generará una excepción NullPointerException.

**4. ¿Cuáles son los beneficios de utilizar tipos genéricos en las listas enlazadas?**

El uso de tipos genéricos en listas enlazadas permite que la estructura de datos sea más flexible y reutilizable, ya que se puede trabajar con cualquier tipo de objeto sin necesidad de duplicar código para cada tipo. Esto mejora la mantenibilidad del código y reduce errores, ya que el compilador puede detectar incompatibilidades de tipo en tiempo de compilación. Además, mejora la legibilidad y evita conversiones innecesarias o el uso excesivo de casting, lo cual hace que el desarrollo sea más seguro y eficiente.

**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**

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

**CRITERIOS DE EVALUACIÓN**

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