



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

INFORME DE LABORATORIO

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	11:59		
INTEGRANTE (s): Davila Flores Mathias Dario				NOTA:	
DOCENTE(s): <ul style="list-style-type: none"> Mg. Ing. Rene Alonso Nieto Valencia. ENLACE GITHUB: https://github.com/mathiasddf/LabsEDA 					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <p>a. Ejercicios Resueltos:</p> <p>i. 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>

	<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: 2</p>

```

Lab4 > Ej_resueltos > LinkedList1.java > LinkedList1 > insert(LinkedList1, int)
1 //Un programa java para implementar una simple lista enlazada
2 public class LinkedList1 {
3     Node head; // cabecera de la lista
4     // Nodo de lista enlazada.
5     // Esta clase interna se hace estática
6     // para que main() pueda acceder a ella
7     static class Node {
8         int data;
9         Node next;
10        // constructor
11        Node(int d){
12            data = d;
13            next = null;
14        }
15    }
16
17    // Método para insertar un nuevo nodo
18    public static LinkedList1 insert(LinkedList1 list, int data) {
19        // Crea un nuevo nodo con los datos dados
20        Node new_node = new Node(data);
21        // Si la lista enlazada está vacía,

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

Run: LinkedList1 +

```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCode
ptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspacestorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'Link
LinkedList: 1 2 3 4 5 6 7 8
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

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

```

DoubleLinkedList.java ...Ejercicio6 Java.java ...Ejercicio7 CircleLinkedList.java ...Ejercicio7 Main.java ...Ejercicio7 Main.java ...Ejercicio6 LinkedList2.java
Lab4 > Ej_resueltos > LinkedList2.java > ...
4 public class LinkedList2 {
114     public static void main(String[] args) {
115         /* Inicia con una lista vacía. */
116         LinkedList2 list = new LinkedList2();
117
118         //
119         // *****INSERCIÓN*****
120         //
121         // Inserta los valores
122         list = insert(list, data:1);
123         list = insert(list, data:2);
124         list = insert(list, data:3);
125         list = insert(list, data:4);
126         list = insert(list, data:5);
127         list = insert(list, data:6);
128         list = insert(list, data:7);
129         list = insert(list, data:8);
130         // Imprime la LinkedList
131         printList(list);
132
133         //

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

Run: LinkedList2 +

```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+Sho
ptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspacestorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\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\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```



```
LinkedList.java ...Ejercicio7 Main.java ...Ejercicio6 LinkedList2.java X LinkedList4.java X LinkedListExample.java X LinkedList3.java X
Lab4 > Ej. resueltos > LinkedList4.java > main(String[])
2 public class LinkedList4 {
178     public static void main(String[] args) {
179
198         //
199         // *****Eliminación por dato *****
200         //
201
202         // Elimina el nodo con el valor 1
203         // En el caso el dato 1 está en ***La cabeza***
204         deleteByKey(list, key:1);
205
206         // Imprime La LinkedList
207         printList(list);
208
209         // Borramos el nodo con el valor 4
210         // En este caso el dato esta presente ***en el
211         // medio***
212         deleteByKey(list, key:4);

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: LinkedList4 + v

```
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

```

	<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: 4</p>

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

```

AddElements.java x  Node.java  ...Ejercicio1  Main.java  ...Ejercicio1  Node.java  ...Ejercicio2  CircleLinkedList.java  ...Ejercicio2  Main.java  ...Ejercicio2
Lab4 > Ej_resueltos > AddElements.java > ...
1 // Un programa java para añadir elementos a una LinkedList
2 import java.util.LinkedList;
3
4 public class AddElements {
5     // Metodo principal
6     public static void main(String[] args) {
7         // Creando unaLinkedList
8         LinkedList<String> l = new LinkedList<String>();
9         // Añadiendo los elementos a la LinkedList usando el método add()
10        l.add(e:"Uno");
11        l.add(e:"Dos");
12        l.add(e:"Tres");
13        l.add(e:"Cuatro");
14        l.add(e:"Cinco");
15        // Imprimiendo la LinkedList
16        System.out.println(l);
17    }

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: AddElements

```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeOnExceptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'AddElements'
[Uno, Dos, Tres, Cuatro, Cinco]
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

- vi. 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.

```

GFG1.java x  AddElements.java  Node.java  ...Ejercicio1  Main.java  ...Ejercicio1  Node.java  ...Ejercicio2  CircleLinkedList.java  ...Ejercicio2  Main.java  ...Ejercicio2
Lab4 > Ej_resueltos > GFG1.java > ...
1 // Programa que demuestra la
2 // implementación de la LinkedList
3 // class
4 import java.util.*;
5
6 public class GFG1 {
7     public static void main(String args[]) {
8         // Creando el objeto de la
9         // clase lista enlazada
10        LinkedList<String> ll = new LinkedList<>();
11        // Añadido de elementos a la lista enlazada
12        ll.add(e:"A");
13        ll.add(e:"B");
14        ll.addLast(e:"C");
15        ll.addFirst(e:"D");
16        ll.add(index:2, element:"E");
17        System.out.println(ll);

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: GFG1

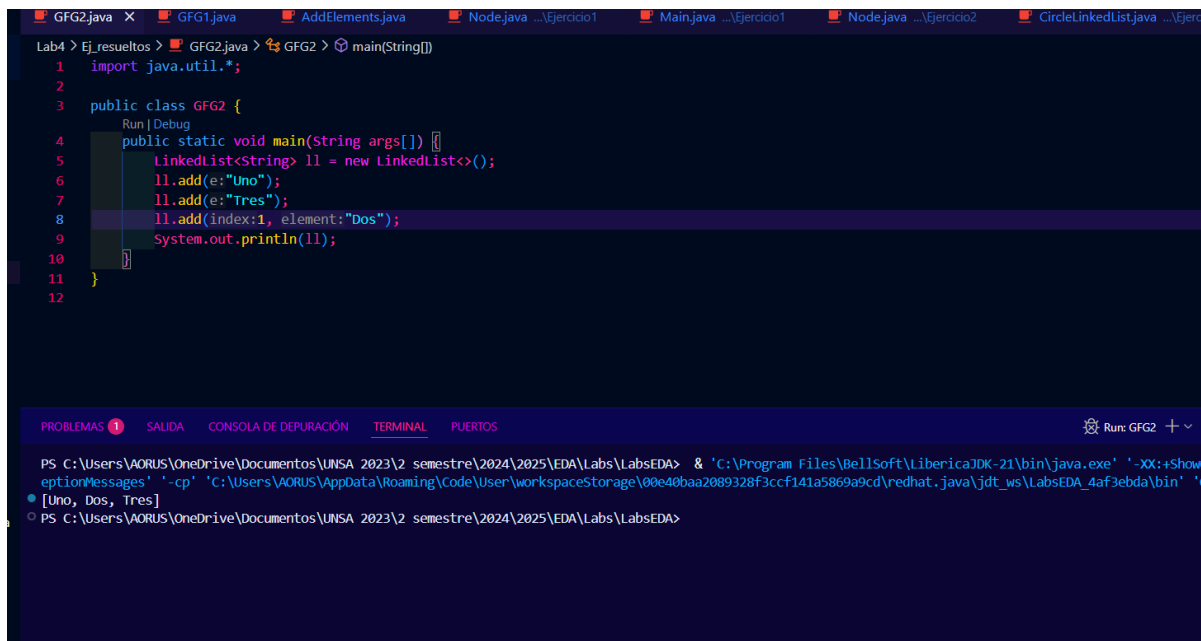
```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeOnExceptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'GFG1'
[D, A, E, B, C]
[A]
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

	<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: 5</p>

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

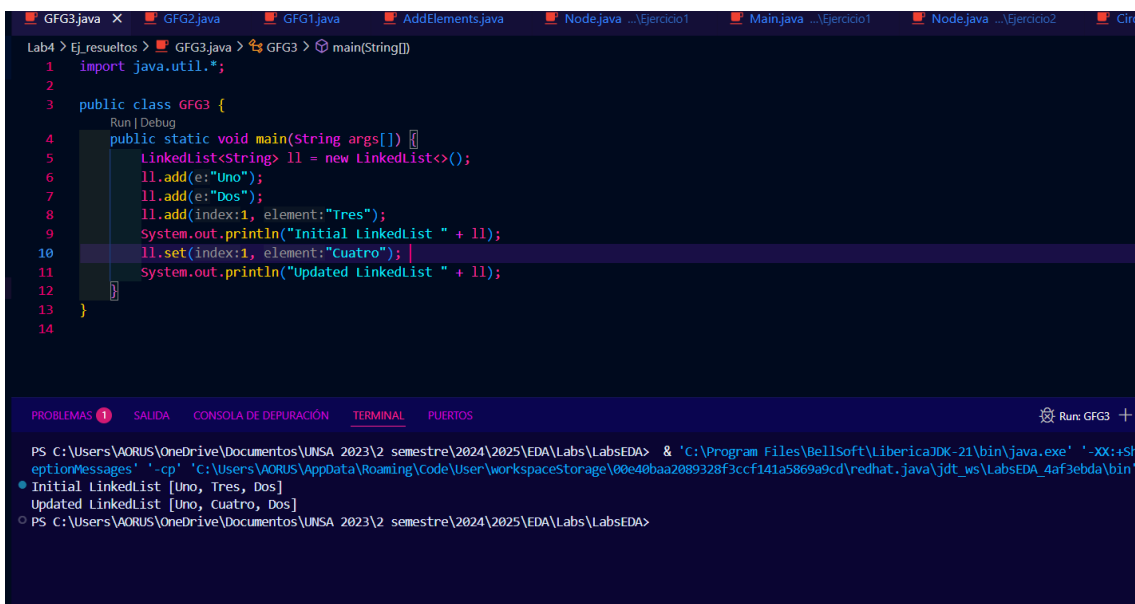


```

GFG2.java X GFG1.java AddElements.java Node.java ...Ejercicio1 Main.java ...Ejercicio1 Node.java ...Ejercicio2 CircleLinkedList.java ...Ejercicio2
Lab4 > Ej_resueltos > GFG2.java > GFG2 > main(String[])
1  import java.util.*;
2
3  public class GFG2 {
4      Run | Debug
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
PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: GFG2 + v
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowOptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'GFG2'
[Uno, Dos, Tres]
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

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



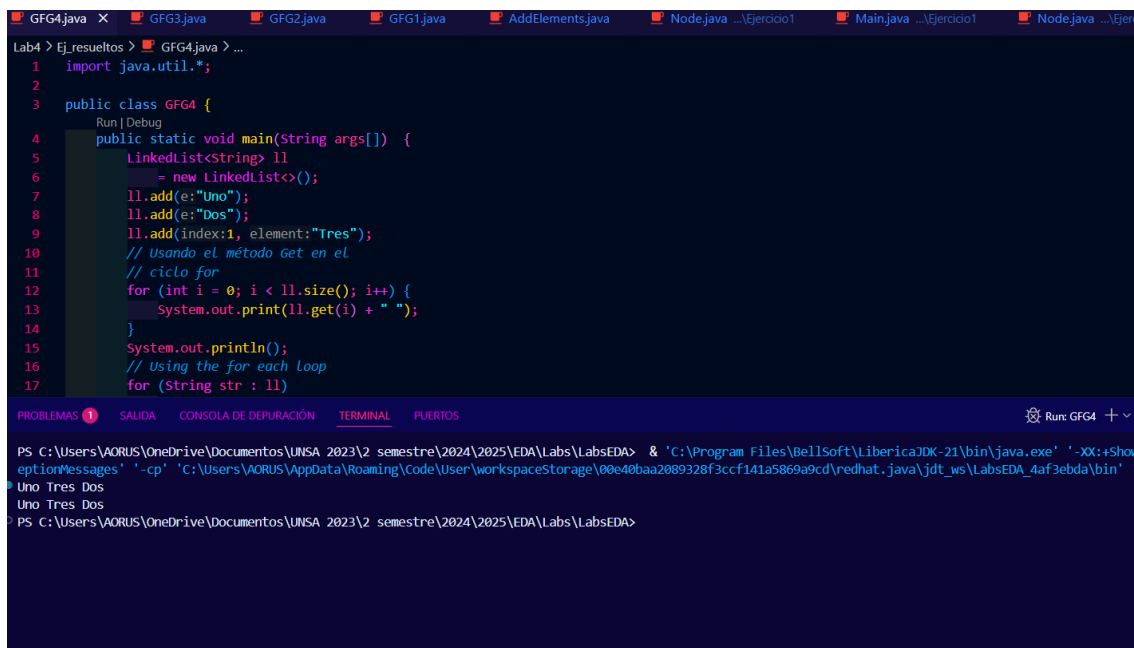
```

GFG3.java X GFG2.java GFG1.java AddElements.java Node.java ...Ejercicio1 Main.java ...Ejercicio1 Node.java ...Ejercicio2 CircleLinkedList.java ...Ejercicio2
Lab4 > Ej_resueltos > GFG3.java > GFG3 > main(String[])
1  import java.util.*;
2
3  public class GFG3 {
4      Run | Debug
5      public static void main(String args[]) {
6          LinkedList<String> ll = new LinkedList<>();
7          ll.add(e:"Uno");
8          ll.add(e:"Dos");
9          ll.add(index:1, element:"Tres");
10         System.out.println("Initial LinkedList " + ll);
11         ll.set(index:1, element:"Cuatro");
12         System.out.println("Updated LinkedList " + ll);
13     }
14
PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: GFG3 + v
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowOptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'GFG3'
Initial LinkedList [Uno, Tres, Dos]
Updated LinkedList [Uno, Cuatro, Dos]
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

	<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: 6</p>

- ix. 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.



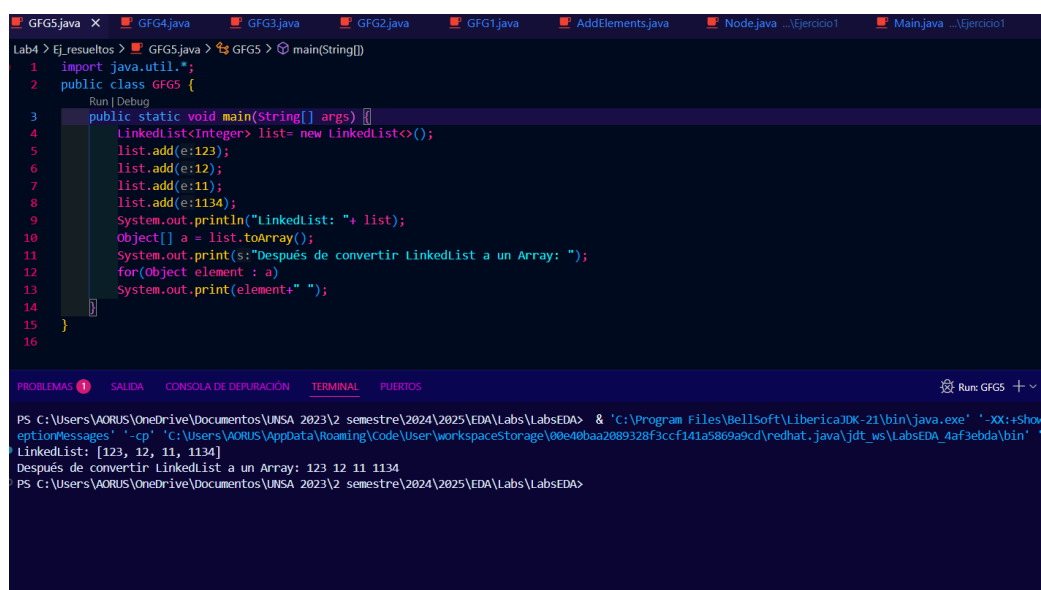
```

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

```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowOptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'G
Uno Tres Dos
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>
    
```

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



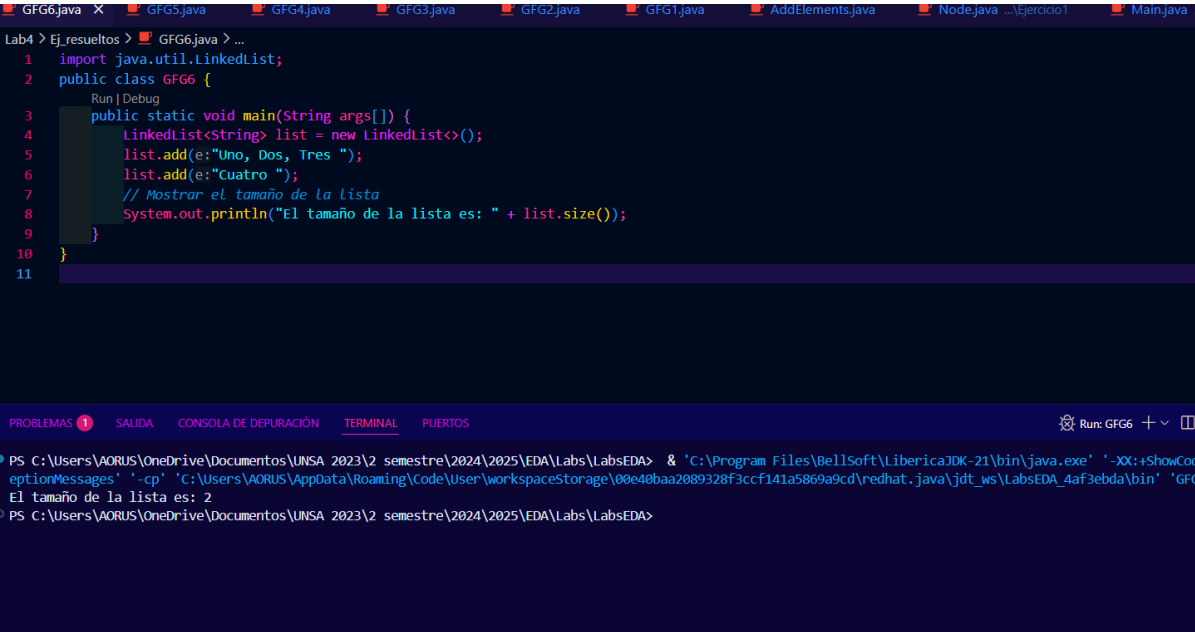
```

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

```

PS C:\Users\AORUS\OneDrive\documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowOptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'G
LinkedList: [123, 12, 11, 1134]
Después de convertir LinkedList a un Array: 123 12 11 1134
PS C:\Users\AORUS\OneDrive\documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>
    
```

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



```

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

```

Terminal Output:

```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDetailMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'GFG6'
El tamaño de la lista es: 2
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

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



```

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



```

Terminal Output:

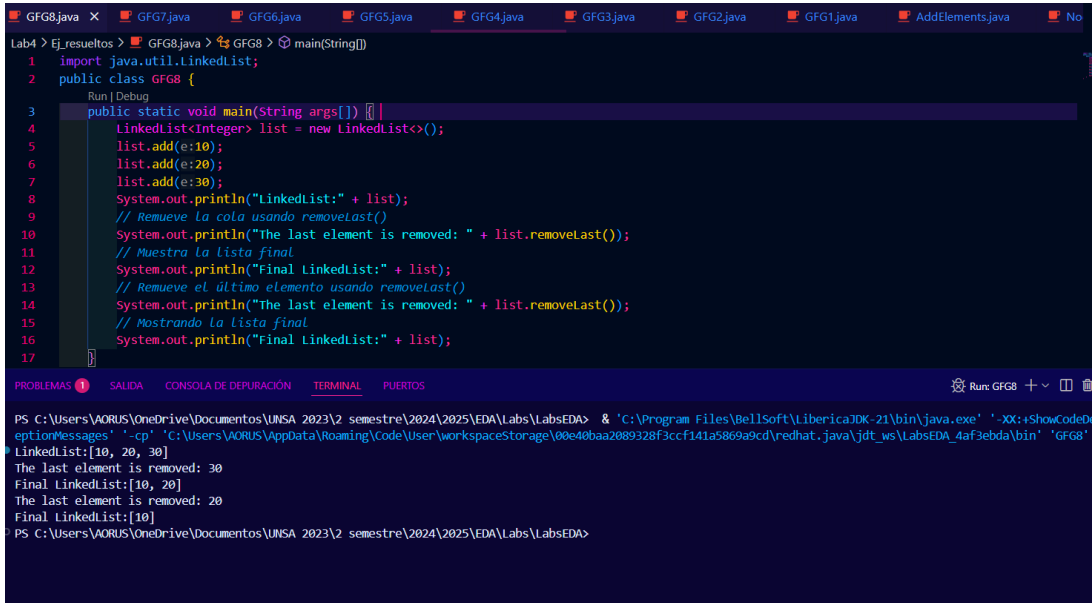
```

PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDetailMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'GFG7'
LinkedList:[10, 20, 30]
El primer elemento removido es: 10
Final LinkedList:[20, 30]
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

	<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: 8</p>

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



```

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

```

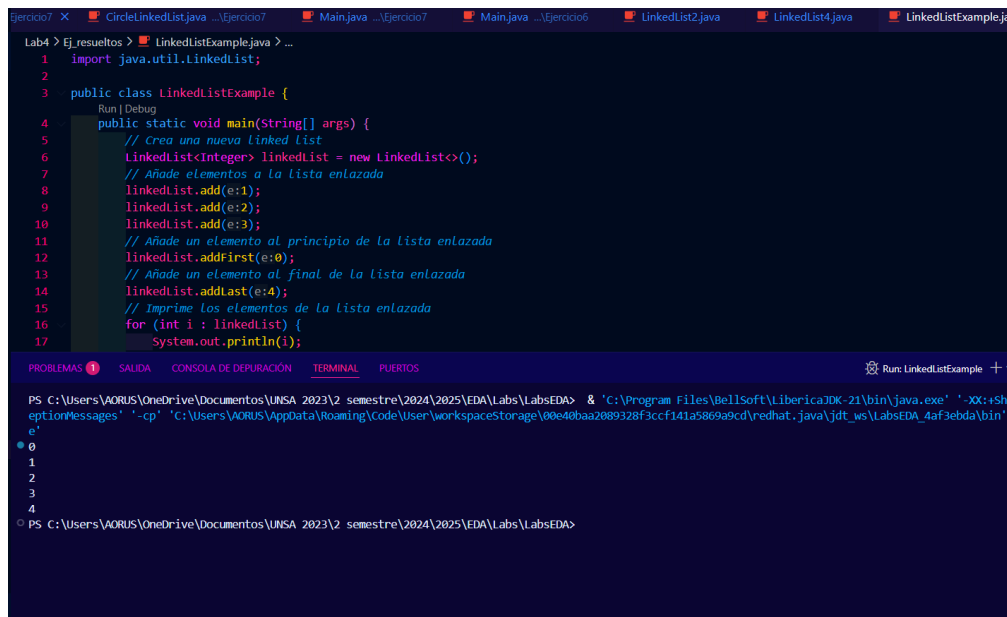
Terminal output:

```

PS C:\Users\AORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDe
eptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2889328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'GFG8'
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\AORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

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



```

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

```

Terminal output:

```

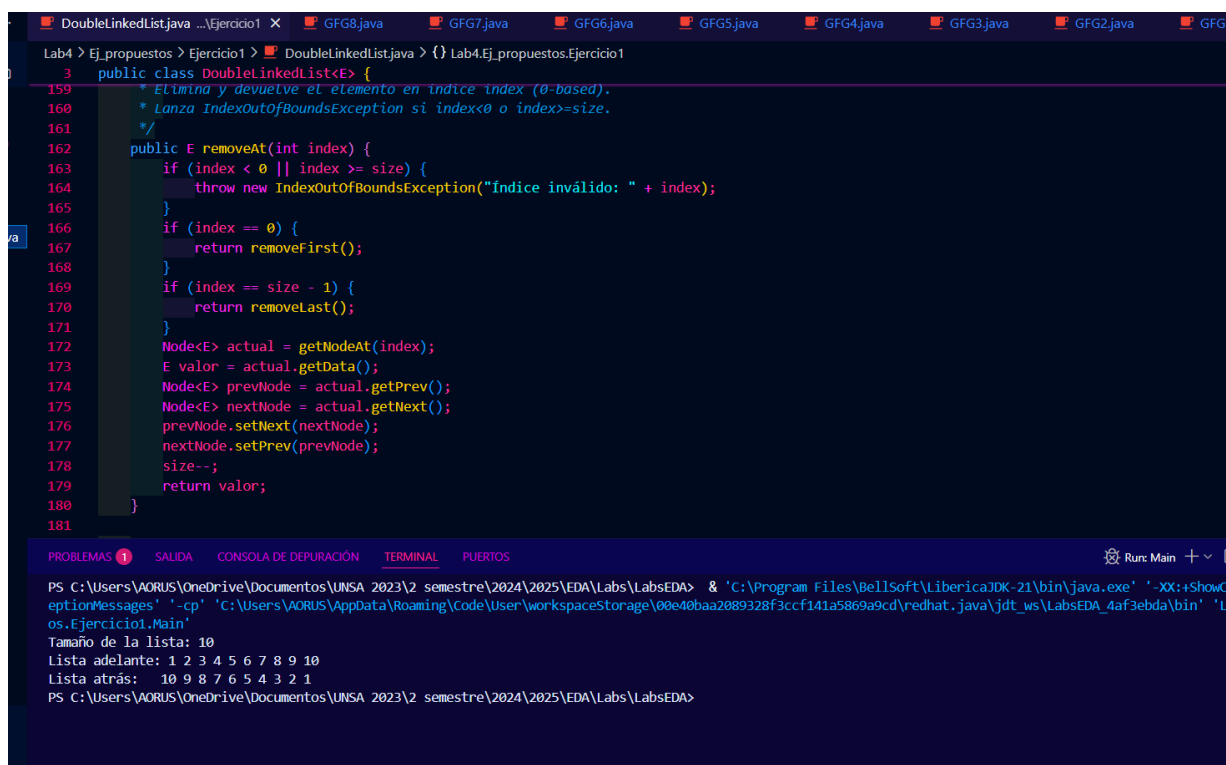
PS C:\Users\AORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+Sh
eptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2889328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin'
e'
0
1
2
3
4
PS C:\Users\AORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```


	<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: 9</p>

b. Ejercicios Propuestos:

- i. **Lista doblemente enlazada propia (elementos 1–10):** En este ejercicio se creó desde cero una clase genérica `Node<E>` (con atributos `data`, `prev` y `next`) y una clase `DoubleLinkedList<E>` que mantiene referencias a `head`, `tail` y un contador `size`. Todos los métodos trabajaron con la firma genérica `public E método(...)` cuando debían devolver datos (por ejemplo, `getFirst()`, `getLast()`, `removeFirst()`, `removeLast()` o `remove(E key)`) y otros devolvían valores primitivos (por ejemplo, `size()`). Para probarla, en `Main.java` se instanció `DoubleLinkedList<Integer>`, se insertaron los enteros del 1 al 10 con `addLast()`, y luego se mostraron el tamaño, los recorridos hacia adelante y hacia atrás, así como llamadas a `getFirst()`, `getLast()`, `contains(...)`, `indexOf(...)` y a los métodos de eliminación (`removeFirst()`, `removeLast()`, `remove(E)`, `removeAt(int)`), demostrando la corrección de todos los casos.



```

DoubleLinkedList.java ...Ejercicio1 x  GFG8.java  GFG7.java  GFG6.java  GFG5.java  GFG4.java  GFG3.java  GFG2.java  GFG
Lab4 > Ej_propuestos > Ejercicio1 > DoubleLinkedList.java > {} Lab4.Ej_propuestos.Ejercicio1
3  public class DoubleLinkedList<E> {
159      * Elimina y devuelve el elemento en indice index (0-based).
160      * Lanza IndexOutOfBoundsException si index<0 o index>=size.
161      */
162      public E removeAt(int index) {
163          if (index < 0 || index >= size) {
164              throw new IndexOutOfBoundsException("Índice inválido: " + index);
165          }
166          if (index == 0) {
167              return removeFirst();
168          }
169          if (index == size - 1) {
170              return removeLast();
171          }
172          Node<E> actual = getNodeAt(index);
173          E valor = actual.getData();
174          Node<E> prevNode = actual.getPrev();
175          Node<E> nextNode = actual.getNext();
176          prevNode.setNext(nextNode);
177          nextNode.setPrev(prevNode);
178          size--;
179          return valor;
180      }
181  }

PROBLEMAS 1  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowC
eptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2889328f3ccf141a5869a9cd\redhat_java\jdt_ws\LabsEDA_4af3ebda\bin' 'L
os.Ejercicio1.Main'
Tamaño de la lista: 10
Lista adelante: 1 2 3 4 5 6 7 8 9 10
Lista atrás: 10 9 8 7 6 5 4 3 2 1
PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

	<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: 10</p>

ii. **Lista circular propia (elementos 1–12):** En este caso también se tripartió el código en una clase genérica `Node<E>` (con atributos `data` y `next`, apuntando inicialmente a sí mismo) y una clase `LinkedList<E>` que implementa una lista circular simple. Los métodos genéricos `addFirst(E)` y `addLast(E)` insertan en el frente o al final, y al buscar el último nodo (aquel cuyo `next == head`) se asegura que el nuevo elemento recicle correctamente hacia el principio. Los métodos de eliminación `removeFirst()`, `removeLast()`, `remove(E key)` y `removeAt(int)` ajustan punteros de manera circular y devuelven el dato afectado. Para las operaciones de acceso, se incluyeron `getFirst()`, `getLast()`, `get(int)`, `indexOf(E)`, `contains(E)` y los dos recorridos `printOneCycle()` y `printN(int n)`. El `Main.java` precarga 1..12 usando `addLast()`, muestra el tamaño y un ciclo completo con `printOneCycle()`, imprime 15 elementos seguidos con `printN(15)` para evidenciar la circularidad, y luego prueba los métodos de búsqueda y eliminación, mostrando la lista y su tamaño tras cada operación.

```

3 public class CircleLinkedList<E> {
    // ...
    300 * obediendo la naturaleza circular (si n > size, continua dando vueltas).
    301 * Si n <= 0 o la lista está vacía, imprime mensaje apropiado.
    302 */
    303 public void printN(int n) {
    304     if (isEmpty() || n <= 0) {
    305         System.out.println(x:"Nada que imprimir.");
    306         return;
    307     }
    308     System.out.print("Imprimo " + n + " elementos: ");
    309     Node<E> actual = head;
    310     for (int i = 0; i < n; i++) {
    311         System.out.print(actual.getData() + " ");
    312         actual = actual.getNext();
    313     }
    // ...
}

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: Main + -

```

Tamaño de la lista circular: 12
Lista circular (1 ciclo): 1 2 3 4 5 6 7 8 9 10 11 12
Imprimo 15 elementos: 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3
Primer elemento (getFirst): 1
Último elemento (getLast): 12
¿Contiene 5? true
¿Índice de 7? 6

removeFirst() elimina: 1
removeLast() elimina: 12
Lista circular (1 ciclo): 2 3 4 5 6 7 8 9 10 11
Tamaño ahora: 10

remove(6) ? true
removeAt(4) ? 7
Lista circular (1 ciclo): 2 3 4 5 8 9 10 11
Tamaño final: 8
○ PS C:\Users\AORUS\OneDrive\Documentos\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

	<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: 11</p>

iii. Lista doblemente enlazada usando `java.util.LinkedList` (elementos 1–10):

Para aprovechar la librería de Java, en este ejercicio no se definieron nodos propios: basta con declarar `LinkedList<Integer> lista = new LinkedList<>()`. Los métodos genéricos preexistentes de `java.util.LinkedList<E>` (por ejemplo, `addLast(E)`, `getFirst()`, `getLast()`, `get(int)`, `contains(Object)`, `indexOf(Object)`, `removeFirst()`, `removeLast()`, `size()`) sirvieron para insertar los enteros del 1 al 10 y recorrer la lista tanto de adelante a atrás (con un bucle `for-each`) como de atrás a adelante (con `descendingIterator()`). De este modo quedó demostrada la implementación de una lista doblemente enlazada sin escribir código de nodos, y se mostraron ejemplos de cada método genérico sobre los valores del 1 al 10.



```

10 public class Main {
11     public static void main(String[] args) {
34         System.out.print(it.next() + " ");
35     }
36     System.out.println();
37
38     // 6. Otros métodos genéricos de LinkedList<E>
39     System.out.println("Primer elemento (getFirst): " + lista.getFirst());
40     System.out.println("Último elemento (getLast): " + lista.getLast());
41     System.out.println("Elemento en índice 4 (get): " + lista.get(index:4)); // valor 5
42     System.out.println("¿Contiene 7? (contains): " + lista.contains(o:7));
43     System.out.println("Índice de 3 (indexOf): " + lista.indexOf(o:3));
44
45     // 7. Eliminar el primer y último con removeFirst() y removeLast()
46     Integer eliminadoPrimero = lista.removeFirst();
47     Integer eliminadoUltimo = lista.removeLast();

```

```

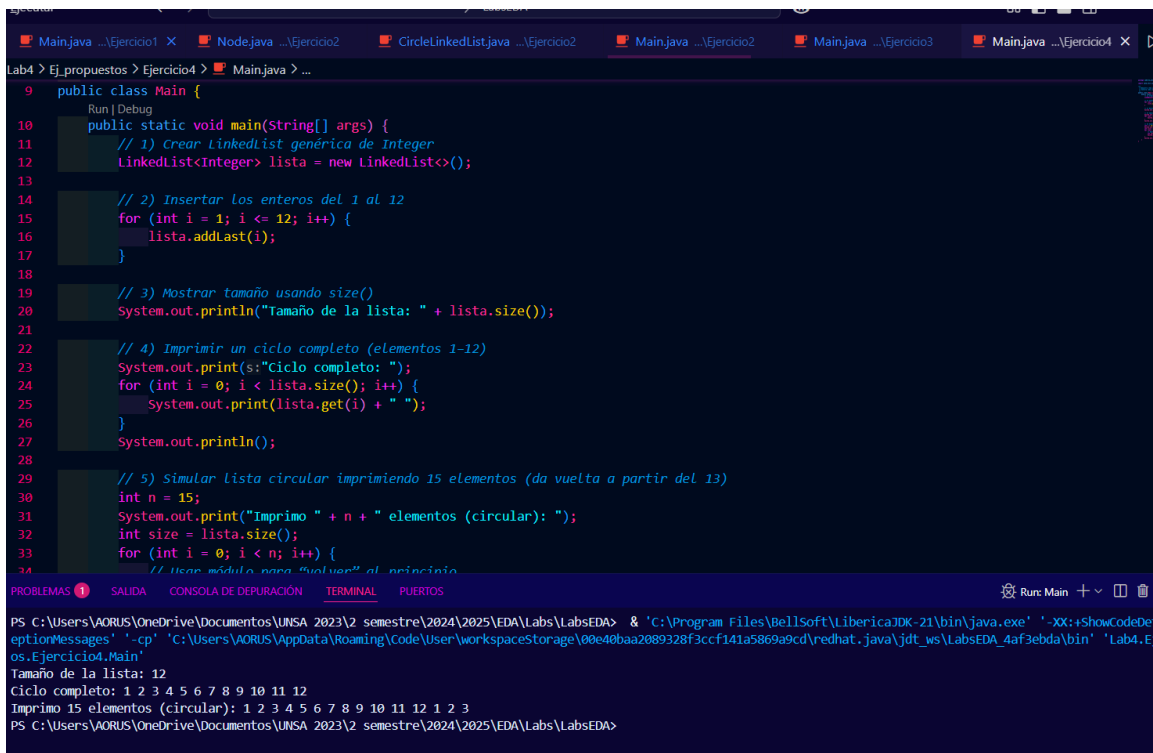
PS C:\Users\VAORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDetails -cp' 'C:\Users\VAORUS\AppData\Roaming\Code\User\workspaceStorage\00e40ba22089328f3ccf141a5869a9cd\redhat.java\jdt_ws\LabsEDA_4af3ebda\bin' 'Lab4.Ejercicio3.Main'
Tamaño de la lista: 10
Lista adelante: 1 2 3 4 5 6 7 8 9 10
Lista atrás: 10 9 8 7 6 5 4 3 2 1
Primer elemento (getFirst): 1
Último elemento (getLast): 10
Elemento en índice 4 (get): 5
¿Contiene 7? (contains): true
Índice de 3 (indexOf): 2

Se eliminó el primer elemento: 1
Se eliminó el último elemento: 10
Lista actual (adelante): 2 3 4 5 6 7 8 9
Tamaño actual: 8
PS C:\Users\VAORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

```

	<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: 12</p>

- iv. **Lista circular “simulada” con java.util.LinkedList (elementos 1–12):** En lugar de implementar nodos propios, se reutilizó LinkedList<Integer> y se accedió a sus elementos mediante índices con el operador módulo para imitar la circularidad. Después de insertar los valores 1..12 con addLast(), se mostró un ciclo completo recorriendo los índices 0..11 con lista.get(i), y a continuación se imprimieron 15 elementos seguidos usando lista.get(i % size) para que, al superar el índice 11, volviera al inicio. De esta forma se “implementó” una lista circular con los enteros del 1 al 12 usando únicamente la clase genérica proporcionada por Java, sin definir nodos ni punteros manualmente.



```

9 public class Main {
10     public static void main(String[] args) {
11         // 1) Crear LinkedList genérica de Integer
12         LinkedList<Integer> lista = new LinkedList<>();
13
14         // 2) Insertar los enteros del 1 al 12
15         for (int i = 1; i <= 12; i++) {
16             lista.addLast(i);
17         }
18
19         // 3) Mostrar tamaño usando size()
20         System.out.println("Tamaño de la lista: " + lista.size());
21
22         // 4) Imprimir un ciclo completo (elementos 1-12)
23         System.out.print(s:"Ciclo completo: ");
24         for (int i = 0; i < lista.size(); i++) {
25             System.out.print(lista.get(i) + " ");
26         }
27         System.out.println();
28
29         // 5) Simular lista circular imprimiendo 15 elementos (da vuelta a partir del 13)
30         int n = 15;
31         System.out.print("Imprimo " + n + " elementos (circular): ");
32         int size = lista.size();
33         for (int i = 0; i < n; i++) {
34             // Usar módulo para "volver" al principio
35         }
36     }
37 }

```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

PS C:\Users\AORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA> & 'C:\Program Files\BellSoft\LibericaJDK-21\bin\java.exe' '-XX:+ShowCodeDet eptionMessages' '-cp' 'C:\Users\AORUS\AppData\Roaming\Code\User\workspaceStorage\00e40baa2089328f3ccf141a5869a9cd\redhat.java\jdk_ws\LabsEDA_4af3ebda\bin' 'Lab4.Ej os.Ejercicio4.Main'

Tamaño de la lista: 12

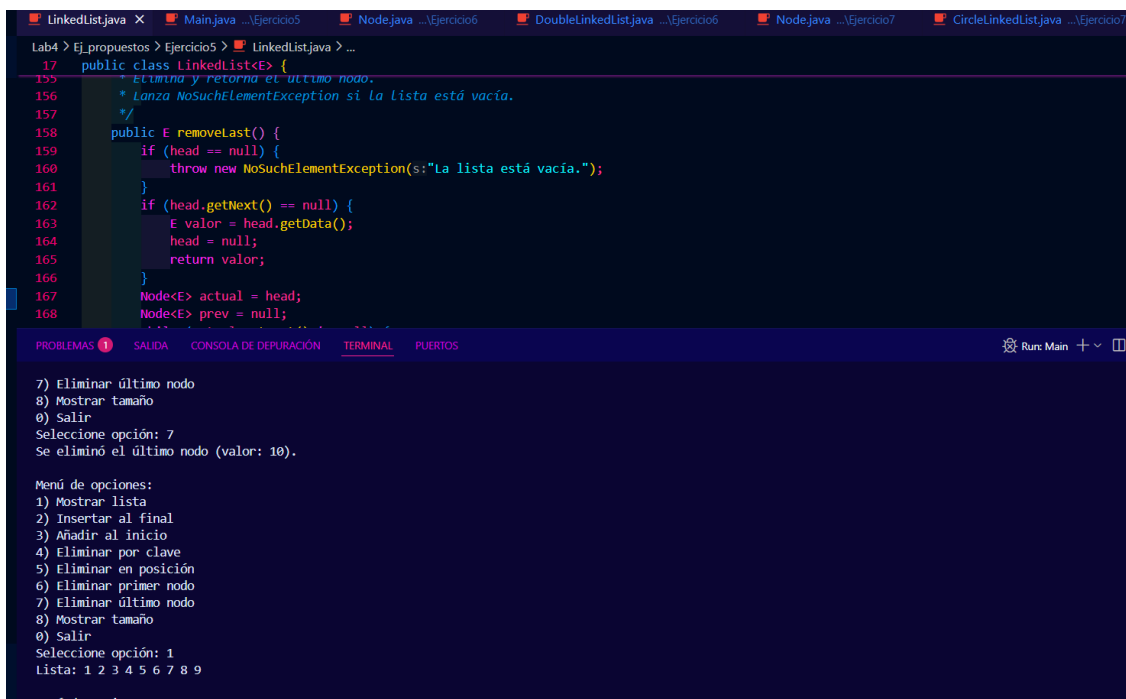
Ciclo completo: 1 2 3 4 5 6 7 8 9 10 11 12

Imprimo 15 elementos (circular): 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3

PS C:\Users\AORUS\OneDrive\Documents\UNSA 2023\2 semestre\2024\2025\EDA\Labs\LabsEDA>

- v. **Lista simplemente enlazada propia (elementos 1–10) con menú:** En este ejercicio se definió Node<E> con data y next, y LinkedList<E> (lista simplemente enlazada) con métodos genéricos: insert(E) o addLast(E), addFirst(E), printList(), deleteByKey(E), deleteAtPosition(int), size(), removeFirst() y removeLast(). El Main.java inicializa la lista con los enteros 1..10 y ofrece un menú interactivo donde el usuario puede elegir mostrar la lista, insertar un valor al final o al inicio, eliminar por clave o por posición, eliminar primer o último nodo, y ver el tamaño. Así se demostró la capacidad de cada método genérico para trabajar sobre la lista pre-cargada con 1..10.

	<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: 13</p>



```

LinkedList.java X Main.java ...Ejercicio5 Node.java ...Ejercicio6 DoubleLinkedList.java ...Ejercicio6 Node.java ...Ejercicio7 CircleLinkedList.java ...Ejercicio7
Lab4 > Ej_propuestos > Ejercicio5 > LinkedList.java > ...
17 public class LinkedList<E> {
155     * Elimina y retorna el ultimo nodo.
156     * Lanza NoSuchElementException si la lista está vacía.
157     */
158     public E removeLast() {
159         if (head == null) {
160             throw new NoSuchElementException(s:"La lista está vacía.");
161         }
162         if (head.getNext() == null) {
163             E valor = head.getData();
164             head = null;
165             return valor;
166         }
167         Node<E> actual = head;
168         Node<E> prev = null;
169         while (actual.getNext() != null) {
170             prev = actual;
171             actual = actual.getNext();
172         }
173         prev.setNext(null);
174     }
175 }

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS Run: Main + - □
7) Eliminar último nodo
8) Mostrar tamaño
0) Salir
Seleccione opción: 7
Se eliminó el último nodo (valor: 10).

Menú de opciones:
1) Mostrar lista
2) Insertar al final
3) Añadir al inicio
4) Eliminar por clave
5) Eliminar en posición
6) Eliminar primer nodo
7) Eliminar último nodo
8) Mostrar tamaño
0) Salir
Seleccione opción: 1
Lista: 1 2 3 4 5 6 7 8 9

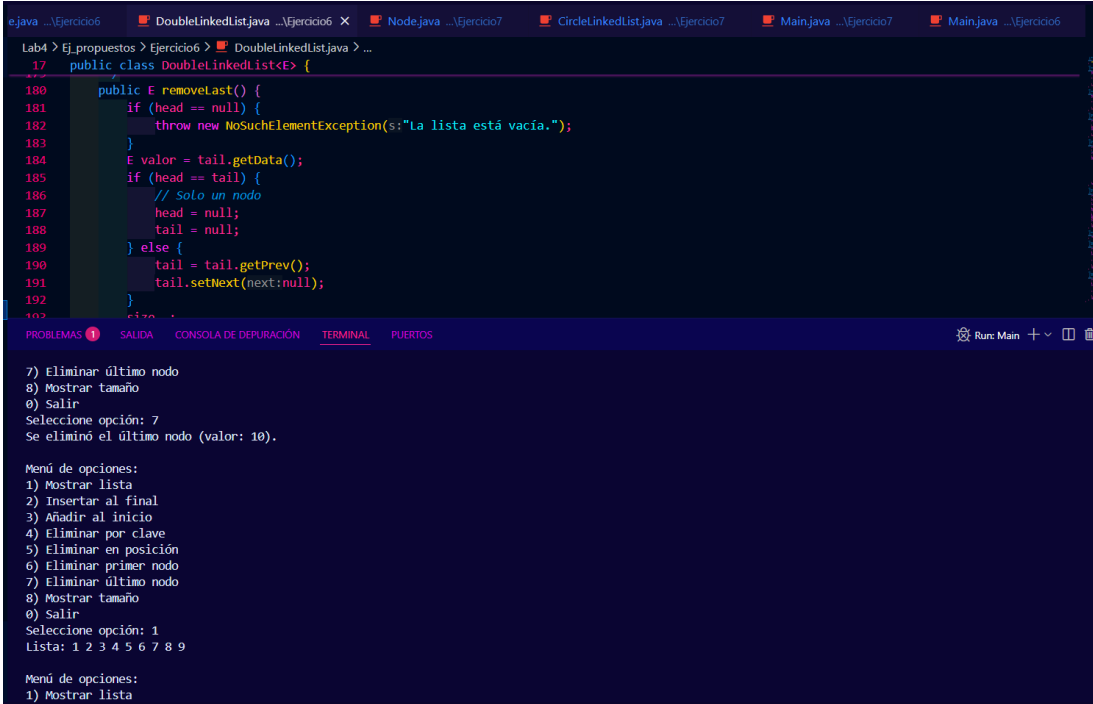
Menú de opciones:

```

vi. Lista doblemente enlazada completa (genérica) con todos los métodos:

Aquí se construyó una implementación desde cero de `DoubleLinkedList<E>` con nodos genéricos `Node<E>` que contienen data, prev y next. Se incluyeron todos los métodos comunes de una lista doblemente enlazada: inserciones (`addFirst(E)`, `addLast(E)`, `add(int, E)`), eliminaciones (`removeFirst()`, `removeLast()`, `remove(E key)`, `removeAt(int)`), accesos (`getFirst()`, `getLast()`, `get(int)`), búsqueda (`indexOf(E)`, `contains(E)`), recorridos (`printForward()`, `printBackward()`), así como utilidades `isEmpty()`, `size()`, `clear()` y el método privado `getNodeAt(int)`. En `Main.java` se mostró un ejemplo paso a paso: insertar varios valores al inicio, al final y en posiciones intermedias; imprimir la lista en ambos sentidos; consultar elementos por índice; eliminar primero, último, por valor o por índice; y finalmente limpiar la lista.

	<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: 14</p>



```

17 public class DoubleLinkedList<E> {
18     public E removeLast() {
19         if (head == null) {
20             throw new NoSuchElementException(s:"La lista está vacía.");
21         }
22         E valor = tail.getData();
23         if (head == tail) {
24             // Solo un nodo
25             head = null;
26             tail = null;
27         } else {
28             tail = tail.getPrev();
29             tail.setNext(next:null);
30         }
31     }
32 }

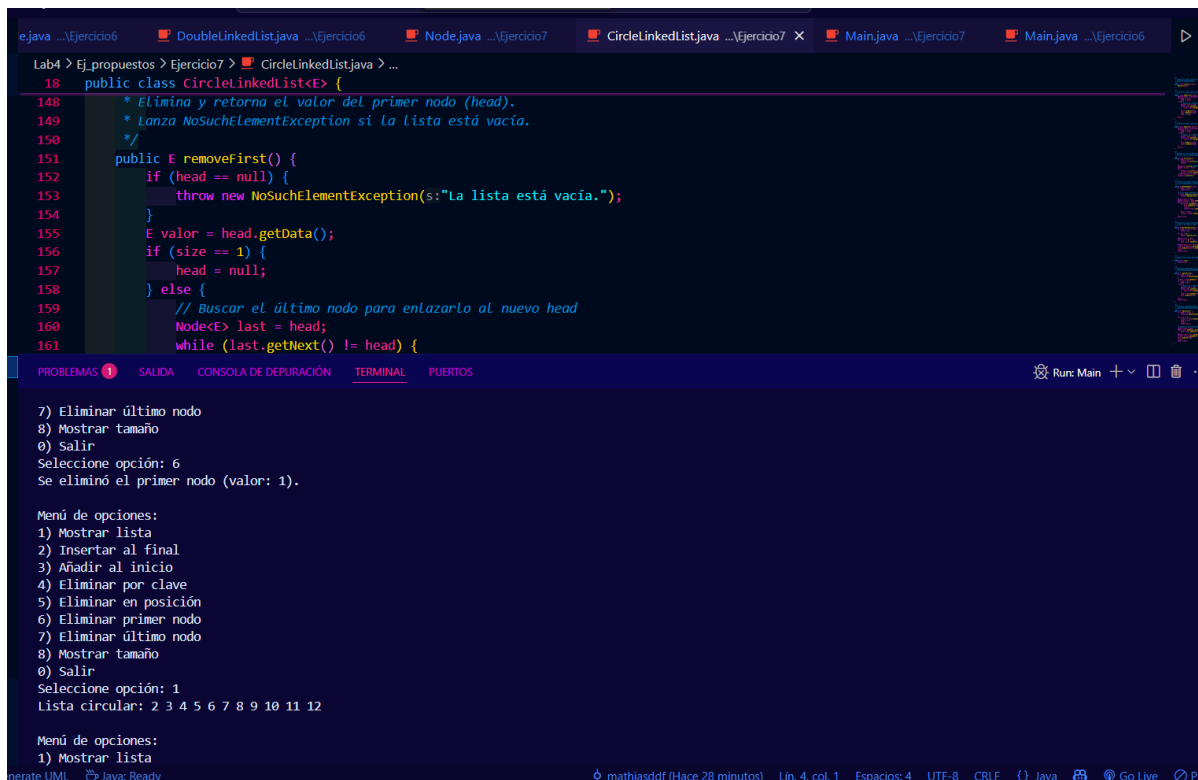
```

7) Eliminar último nodo
8) Mostrar tamaño
0) Salir
Selecione opción: 7
Se eliminó el último nodo (valor: 10).

Menú de opciones:
1) Mostrar lista
2) Insertar al final
3) Añadir al inicio
4) Eliminar por clave
5) Eliminar en posición
6) Eliminar primer nodo
7) Eliminar último nodo
8) Mostrar tamaño
0) Salir
Selecione opción: 1
Lista: 1 2 3 4 5 6 7 8 9

vii. Lista circular propia completa (genérica) con menú (elementos 1–12): En este último punto volvimos a implementar `Node<E>` (con data y next) y `CircleLinkedList<E>` como lista circular. Además de los métodos genéricos estándar (`add(int, E)`, `get(int)`, `indexOf(E)`, `contains(E)`, `clear()`, `size()`), agregamos las versiones adaptadas a una lista circular: `addFirst(E)` y `addLast(E)` que deben localizar el último nodo para mantener el bucle; `removeFirst()` y `removeLast()` que actualizan correctamente head y los enlaces circulares; `remove(E key)` y `removeAt(int index)` que eliminan nodos en un contexto circular; `printOneCycle()` que recorre exactamente un bucle y `printN(int n)` que imprime N elementos consecutivos posiblemente dando vueltas; así como acceso a primero/último con `getFirst()` y `getLast()`. El `Main.java` arranca precargando 1..12, luego despliega un menú interactivo con opciones para mostrar la lista, insertar al inicio o al final, eliminar por clave o por posición, eliminar primer o último nodo, y ver el tamaño. Cada operación demuestra el uso de métodos genéricos `public E método(...)` para manipular la lista circular con los valores 1–12.

	<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: 15</p>



```

eJava ...Ejercicio6 DoubleLinkedList.java ...Ejercicio6 Node.java ...Ejercicio7 CircleLinkedList.java ...Ejercicio7 X Main.java ...Ejercicio7 Main.java ...Ejercicio6
Lab4 > Ej_propuestos > Ejercicio7 > CircleLinkedList.java > ...
18 public class CircleLinkedList<E> {
148     * Elimina y retorna el valor del primer nodo (head).
149     * Lanza NoSuchElementException si la lista está vacía.
150     */
151     public E removeFirst() {
152         if (head == null) {
153             throw new NoSuchElementException(s:"La lista está vacía.");
154         }
155         E valor = head.getData();
156         if (size == 1) {
157             head = null;
158         } else {
159             // Buscar el último nodo para enlazarlo al nuevo head
160             Node<E> last = head;
161             while (last.getNext() != head) {
162
7) Eliminar último nodo
8) Mostrar tamaño
0) Salir
Seleccione opción: 6
Se eliminó el primer nodo (valor: 1).

Menú de opciones:
1) Mostrar lista
2) Insertar al final
3) Añadir al inicio
4) Eliminar por clave
5) Eliminar en posición
6) Eliminar primer nodo
7) Eliminar último nodo
8) Mostrar tamaño
0) Salir
Seleccione opción: 1
Lista circular: 2 3 4 5 6 7 8 9 10 11 12

Menú de opciones:
1) Mostrar lista

```

II. SOLUCIÓN DEL CUESTIONARIO

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

Al implementar desde cero las distintas variantes de listas enlazadas (simple, doble, circular) en Java, surgieron varias dificultades prácticas y conceptuales:

- **Comprender la circularidad y actualizar punteros correctamente:** En la lista circular, cada vez que insertamos o eliminamos un nodo (especialmente en `addFirst`, `addLast`, `removeFirst` y `removeLast`), es necesario rastrear cuidadosamente cuál es el “último” nodo (aquel cuyo `next` apunta de vuelta a `head`) para mantener la invariancia circular. Si olvidamos reasignar alguno de esos punteros, la lista deja de ser circular o se rompe el enlace, provocando `NullPointerException` o bucles infinitos al imprimir.
- **Mantener la doble referencia (prev/next) en la lista doblemente enlazada:** Al borrar un nodo intermedio, hay que actualizar tanto `prev.next` como `next.prev`. Un descuido en cualquiera de esos dos enlaces

	<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: 16</p>

puede dejar la lista en un estado inconsistente (p. ej., un nodo huérfano o referencias colgantes). Además, al eliminar el primer o último elemento, hay que revisar cuidadosamente si el nodo eliminado es tanto head como tail, y en ese caso establecer ambas referencias a null.

- **Gestión del tamaño y comprobaciones de índices:** Para métodos que reciben un índice (como `add(int, E)` o `removeAt(int)`), es fácil pasar por alto las validaciones `index < 0 || index > size` o `index >= size`. Si no validamos antes, podemos intentar recorrer más allá de los límites, lo que genera `NullPointerException` o índices inválidos. Llevar un contador `size` actualizado en cada inserción o eliminación ayuda, pero exige disciplina: olvidar `size++` o `size--` en un método provoca errores de validación posteriores.
- **Falta de documentación unificada para estructuras “homemade”:** Aunque Java provee `java.util.LinkedList<E>` con toda la lógica interna resuelta, al crear nuestras propias clases no existe una referencia oficial paso a paso: hubo que investigar en foros (Stack Overflow) y en la *Java Language Specification* (JLS) para entender bien conceptos como “null type” o la definición formal de subtipado. La documentación de Oracle (p. ej., la sección Generics (The Java Tutorials)) es útil, pero no detalla explícitamente cómo enlazar nodos manualmente. Esto incrementó el tiempo dedicado a depurar punteros “prev” y “next”.
- **Complejidad del lenguaje en torno a genéricos y null:** Java introduce restricciones en tiempo de compilación: no se puede instanciar un `new E` ni crear un arreglo `E[]` directamente, y el tipo “null” es un tipo especial (sin nombre) que puede asignarse a cualquier referencia [Stack Overflow](#). Comprender qué null pertenece a un “null type” (subtipo de todas las referencias) y cómo eso interactúa con genéricos (por ejemplo, `LinkedList<String>` puede contener null) supuso un punto extra de complejidad.

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

Sí. La clase `Node<E>`, al ser genérica y autónoma, puede emplearse en múltiples estructuras más allá de listas enlazadas:

- **Pilas (Stacks) implementadas con nodos enlazados:** Basta con mantener solo el puntero `top` (o `head`), donde `push(E value)` crea un `Node<E>` que apunta al nodo anterior, y `pop()` extrae `head.getData()` y reasigna `head = head.getNext()`. El enlace “prev” puede omitirse si solo necesitamos comportamiento LIFO.

	<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: 17</p>

- **Colas (Queues) basadas en nodos enlazados:** Manteniendo referencias a head (frente) y tail (final), enqueue(E value) agrega un nodo nuevo tras tail, y dequeue() hace head = head.getNext(). El mismo Node<E> sirve para apuntar al siguiente.
- **Árboles binarios o n-arios:** Si extendemos Node<E> a incluir referencias left y right (o incluso List<Node<E>> children), podemos armar jerarquías. Con ligeras modificaciones, nuestra Node<E> sirve de base para sumar nodos de subárboles.
- **Grafos (listas de adyacencia):** Al representar cada vértice como un Node<E> y mantener, por ejemplo, un List<Node<E>> neighbors, el mismo patrón genérico facilita el almacenamiento de datos arbitrarios en cada vértice.
- **Tablas de dispersión (Hash Tables) con buckets enlazados:** Una implementación clásica sujeta colisiones en una cadena enlazada: cada bucket apunta al primer Node<Entry<K,V>>, donde Node enlaza pares clave/valor y cada next cruza colisiones.

c. ¿Qué tipo de dato es NULL en java?

De acuerdo con la *Java Language Specification (JLS)*, existe un “null type” especial cuyo único valor posible es justamente null. A diferencia de cualquier otra clase o interfaz, el “null type” no tiene nombre y no se puede declarar directamente.

En la práctica, null es un literal que representa la ausencia de objeto. Cuando asignas null a cualquier variable de tipo referencia (por ejemplo, String s = null; o MyClass obj = null;), el compilador entiende que esa variable no apunta a ningún objeto en el montón (heap). Se suele decir que el “null reference” es de un tipo “vacío” que puede convertirse a cualquier tipo de referencia mediante una conversión de subtipado (widening reference conversion).

- null no es un objeto. No se encuentra en el heap ni tiene un “header” de objeto.
- En memoria, un campo de tipo referencia cuyo valor es null se representa con todos bits en cero (aunque esto no aparece explícito en el JLS, lo describe la implementación de la JVM) Stack OverflowStack Overflow.
- null no puede asignarse a tipos primitivos (p. ej., int, boolean, double), ya que éstos no pertenecen a la jerarquía de tipos de referencia. Solo los

wrappers (Integer, Boolean, Double, etc.) pueden contener null.

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

El uso de generics (public class LinkedList<E>) en Java proporciona varias ventajas importantes al diseñar y consumir estructuras de datos como listas enlazadas:

1. **Seguridad en tiempo de compilación:** Al declarar `LinkedList<Integer> lista = new LinkedList<>();`, el compilador garantiza que solo se puedan insertar Integer en esa lista. Si intentamos hacer `lista.add("texto")`, el compilador lanzará un error de tipo. Esto evita `ClassCastException` en tiempo de ejecución y reduce bugs relacionados con casting manual (antes de Java 5, era común tener que convertir (Integer) obj al extraer de una LinkedList no genérica).
2. **Eliminación de casts explícito:** Sin genéricos, sacar un elemento de una lista enlazada (`Object o = lista.removeFirst();`) exige convertirlo a su tipo real (`Integer x = (Integer) o;`). Con genéricos, `Integer x = lista.removeFirst();` ya está tipado, y no es necesario un casteo. Esto simplifica el código y mejora su legibilidad.
3. **Reutilización del mismo código:** Con una sola implementación de `LinkedList<E>`, es posible crear listas de String, Persona, BigDecimal o cualquier tipo de referencia sin duplicar clases. El compilador genera (internamente) la plantilla necesaria para cada tipo concreto, pero el programador escribe solo una versión genérica.
4. **Documentación implícita del tipo de elemento:** Cuando alguien lee `LinkedList<Empleado>`, queda claro que esa lista almacena instancias de Empleado; no hay ambigüedad. Con colecciones sin tipo, había que consultar comentarios o documentaciones adicionales para saber qué clase de objetos se guardan.
5. **Menor probabilidad de errores en tiempo de ejecución:** Al forzar el chequeo de tipos en tiempo de compilación, se evitan lanzamientos inesperados en ejecución. Por ejemplo, métodos genéricos como `public E removeFirst()` devuelven un E concreto (no un Object), evitando errores de "cast".
6. **Interoperabilidad y consistencia con la API de Java:** Desde Java 5, gran parte de la librería estándar (collections, streams, etc.) usa genéricos. Al seguir el mismo patrón en nuestras propias listas (`LinkedList<E>`), nos alineamos con la API y podemos, por ejemplo, pasar una `LinkedList<Empleado>` a métodos que consumen un `Collection<Empleado>` sin órtesis de tipo.

	<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: 19</p>

III. CONCLUSIONES

- **Reutilización de nodos genéricos:** Una sola clase `Node<E>` sirve para diversas estructuras (listas simples, dobles, circulares, pilas, colas, árboles), evitando duplicación de código.
- **Invariantes críticas:** En listas circulares y dobles, actualizar con precisión los punteros `next/prev` garantiza consistencia y previene bucles infinitos o referencias nulas.
- **Seguridad y claridad con genéricos:** `LinkedList<E>` o `LinkedList<E>` propio evita casts, detecta errores de tipo en compilación y trabaja con cualquier tipo de objeto.
- **Validación de null:** Tratar `head == null` antes de operar previene `NullPointerException` y asegura que las operaciones de inserción/eliminación sean robustas.

REFERENCIAS Y BIBLIOGRAFÍA

- Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Java* (6th ed.). John Wiley & Sons.
- Oracle. (n.d.). *The Java™ Language Specification, Java SE 17 Edition*. Retrieved May 31, 2025, from <https://docs.oracle.com/javase/specs/jls/se17/html/index.html>
- Oracle. (n.d.). *Java™ Tutorials: Generics*. Retrieved May 31, 2025, from <https://docs.oracle.com/javase/tutorial/java/generics/index.html>
- Oracle. (n.d.). *LinkedList (Java Platform SE 17)*. Retrieved May 31, 2025, from <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/LinkedList.html>
- Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.
- Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in Java* (3rd ed.). Pearson.