GUÍA DE LABORATORIO

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.**

RECURSOS A UTILIZAR:

- **Repositorio GITHUB: https://github.com/JosueClaudioQP/EDA-Lab**
- **Lenguaje de Programación Java.**
- **Ide Java Eclipse/Visual Studio Code.**

DOCENTE(s):

- **Mg. Ing. Rene Alonso Nieto Valencia.**

ALUMNO:

- **Quispe Pauccar, Josué Claudio**

OBJETIVOS/TEMAS Y COMPETENCIAS

OBJETIVOS:

- Aprenda Listas Enlazadas.
- Aplicar conceptos elementales de programación a resolver utilizando POO en problemas de algoritmos.
- Desarrollar pruebas.

TEMAS:

- Introducción.
- Listas Enlazadas.
- Operaciones de las Listas Enlazadas.

CONTENIDO DEL LABORATORIO

- **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.

```java
package Laboratorio4;
import java.io.*;

// Un programa java para implementar una simple lista enlazada
public class LinkedList1 {
    Node head; // cabecera de la lista
// Nodo de lista enlazada.
// Esta clase interna se hace estática
// para que main() pueda acceder a ella

 static class Node {
  int data;
  Node next;

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

// Método para insertar un nuevo nodo
public static LinkedList1 insert(LinkedList1 list, int data) {
  // Crea un nuevo nodo con los datos dados
  Node new_node = new Node(data);
  // Si la lista enlazada está vacía,
  // entonces convierte el nuevo nodo en la cabeza
  if (list.head == null) {
   list.head = new_node;
  } else {
    // De lo contrario recorra hasta el último nodo
    // e inserte el nuevo nodo alli
    Node last = list.head;
    while (last.next != null) {
        last = last.next;
   }
   // Inserta el nuevo nodo al último nodo
   last.next = new_node;
  }
  // Retorna la lista desde la cabeza
  return list;
 }

```

```java
  // Metodo para imprimir la lista enlazada LinkedList.
  public static void printList(LinkedList1 list) {
   Node currNode = list.head;
   System.out.print("LinkedList: ");
   // Recorre la lista enlazada (LinkedList)
   while (currNode != null) {
    // Imprime el dato en el nodo actual
    System.out.print(currNode.data + " ");
    currNode = currNode.next;
   }
  }

  // Código principal
 public static void main(String[] args) {
   /* Inicia con una lista vacia. */
   LinkedList1 list = new LinkedList1();
   //
   // ******INSERCIÓN******
   //
   // Inserta los valores
   list = insert(list, 1);
   list = insert(list, 2);
   list = insert(list, 3);
   list = insert(list, 4);
   list = insert(list, 5);
   list = insert(list, 6);
   list = insert(list, 7);
   list = insert(list, 8);
   printList(list);
   // Imprime la LinkedList
  }
 }
```

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
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.

```java
package Laboratorio4;

public class LinkedList2 {
    Node head; // cabecera de la lista
// Nodo de lista enlazada.
// Esta clase interna se hace estática
// para que main() pueda acceder a ella

 static class Node {
  int data;
  Node next;

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

```

```
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 alli
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     // Muetra el mensaje
64     System.out.println(key + " found and deleted");
65     // Retorna la lista actualizada
66     return list;
67    }
68
```

```
    //
    // CASO 2:
    // Si el dato está en otro lugar que no sea la cabecera
    //
    // Busca el dato para ser borrado,
    // realiza un seguimiento al nodo anterior
    // ya que es necesario cambiar currNode.next
    while (currNode != null && currNode.data != key) {
     // si currNode no tiene el dato
     // continua con el siguiente nodo
     prev = currNode;
     currNode = currNode.next;
    }

    // si el dato esuviera presente, sería el currNode
   // Por lo tanto, el currNode no debe ser nulo
    if (currNode != null) {
     // Desde que el dato está en currNode
     // Desenlaza currNode de la linked list
     prev.next = currNode.next;
     // muestra el mensaje
     System.out.println(key + " found and deleted");
    }

    //
    // CASO 3: El dato no está presente
    //
    // Si la el dato no está presente en linked list
    // el nodo actual podría ser nulo
    if (currNode == null) {
     // Muestra el mensaje
     System.out.println(key + " not found");
    }

    // devuelve la lista
    return list;
   }

   // Código principal
  public static void main(String[] args) {
    /* Inicia con una lista vacia. */
    LinkedList2 list = new LinkedList2();
    //
    // ******INSERCIÓN******
    //
    // Inserta los valores
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);
    printList(list);
    // Imprime la LinkedList

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

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.

```java
public class LinkedList3 {
    Node head; // cabecera de la lista
// Nodo de lista enlazada.
// Esta clase interna se hace estática
// para que main() pueda acceder a ella

 static class Node {
  int data;
  Node next;

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

// Método para insertar un nuevo nodo
public static LinkedList3 insert(LinkedList3 list, int data) {
  // Crea un nuevo nodo con los datos dados
  Node new_node = new Node(data);
  // Si la lista enlazada está vacía,
  // entonces convierte el nuevo nodo en la cabeza
  if (list.head == null) {
   list.head = new_node;
  } else {
    // De lo contrario recorra hasta el último nodo
    // e inserte el nuevo nodo alli
    Node last = list.head;
    while (last.next != null) {
        last = last.next;
   }
   // Inserta el nuevo nodo al último nodo
   last.next = new_node;
  }
  // Retorna la lista desde la cabeza
  return list;
 }
```

```java
// Metodo para imprimir la lista enlazada LinkedList.
public static void printList(LinkedList3 list) {
 Node currNode = list.head;
 System.out.print(s:"LinkedList: ");
 // Recorre la lista enlazada (LinkedList)
 while (currNode != null) {
  // Imprime el dato en el nodo actual
  System.out.print(currNode.data + " ");
  currNode = currNode.next;
 }
}

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

    // CASO 2:
    // Si el índice es mayor que 0 pero menor que el
    // tamaño de LinkedList
    //
    // El contador
    int counter = 0;
    // Conteo del índice a ser eliminado
    // seguimiento del nodo anterior
    // ya que es necesario cambiar currNode.next
    while (currNode != null) {
        if (counter == index) {
            // Dado el currNode es la posición requerida
            // se desenlaza currNode de la lista enlazada
            prev.next = currNode.next;
            // Muestra el mensaje
            System.out.println(index + " position element deleted");
            break;
        } else {
            // si la posición actual no es el indice
            // continua con el siguiente nodo
            prev = currNode;
            currNode = currNode.next;
            counter++;
            }
    }
    // Si se encontro el elmento en la posición, debería estar
    // en currNode Por lo tanto, currNode no debe ser
    // null
    //
    // CASE 3: El indice es mayor que el tamaño de la
    // LinkedList
    //
    // En el caso, el valor de currNode puede ser nulo
    if (currNode == null) {
        // Muestra el mensaje
        System.out.println(index + " position element not found");
    }
    // retorna la lista
    return list;
}
```

```java
public static void main(String[] args) {
  /* Inicia con una lista vacia. */
  LinkedList3 list = new LinkedList3();
  //
  // ******INSERCIÓN******
  //
  // Inserta los valores
  list = insert(list, data:1);
  list = insert(list, data:2);
  list = insert(list, data:3);
  list = insert(list, data:4);
  list = insert(list, data:5);
  list = insert(list, data:6);
  list = insert(list, data:7);
  list = insert(list, data:8);
  // Imprime la LinkedList
  printList(list);
  // Eliminar nodo en la posición 2
  // En este caso, la clave está presente ***en el
  // medio***
  deleteAtPosition(list, index:2);
  // Imprime la LinkedList
  printList(list);
  // Eliminar el nodo en la posición 10
  // En este caso el dato esta ***no presente***
  deleteAtPosition(list, index:10);
  // Imprime la LinkedList
  printList(list);
 }
}
```

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 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.

```java
// *************BORRADO POR DATO*************
// Metodo para eliminar un nodo en LinkedList por dato
public static LinkedList4 deleteByKey(LinkedList4 list, int key) {
    // Aloja el nodo cabecera
    Node currNode = list.head, prev = null;
    //
    // CASO 1:
    // Si el nodo principal tiene el dato que se va a eliminar
    if (currNode != null && currNode.data == key) {
        list.head = currNode.next; // Changed head
        // Muetra el mensaje
        System.out.println(key + " found and deleted");
        // Retorna la lista actualizada
        return list;
    }
    //
    // CASO 2:
    // Si el dato está en otro lugar que no sea la cabecera
    //
    // Busca el dato para ser borrado,
    // realiza un seguimiento al nodo anterior
    // ya que es necesario cambiar currNode.next
    while (currNode != null && currNode.data != key) {
    // si currNode no tiene el dato
    // continua con el siguiente nodo
        prev = currNode;
        currNode = currNode.next;
    }
```

```java
        // si el dato esuviera presente, sería el currNode
        // Por lo tanto, el currNode no debe ser nulo
        if (currNode != null) {
            // Desde que el dato está en currNode
            // Desenlaza currNode de la linked list
            prev.next = currNode.next;
            // muestra el mensaje
            System.out.println(key + " found and deleted");
        }
        //
        // CASO 3: El dato no está presente
        //
        // Si la el dato no está presente en linked list
        // el nodo actual podría ser nulo
        if (currNode == null) {
            // Muestra el mensaje
            System.out.println(key + " not found");
        }
        // devuelve la lista
        return list;
    }
```

```java
    // *************Borrado por posicion*************
    // Metodo para eliminar un nodo en la LinkedList por POSITION
    public static LinkedList4 deleteAtPosition(LinkedList4 list, int index) {
        // Guarda el nodo cabecera
        Node currNode = list.head, prev = null;
        //
        // CASE 1:
        // si el índice es 0, entonces el nodo principal debe ser
        // eliminado
        if (index == 0 && currNode != null) {
            list.head = currNode.next; // Cambia la cabecera
            // Muestra el mensaje
            System.out.println(index + " position element deleted");
            // Retorna la lista actualizada
            return list;
        }
        //
        // CASO 2:
        // Si el índice es mayor que 0 pero menor que el
        // tamaño de LinkedList
        //
        // El contador
        int counter = 0;
        // Conteo del índice a ser eliminado
        // seguimiento del nodo anterior
        // ya que es necesario cambiar currNode.next
        while (currNode != null) {
            if (counter == index) {
                // Dado el currNode es la posición requerida
                // se desenlaza currNode de la lista enlazada
                prev.next = currNode.next;
                // Muestra el mensaje
                System.out.println(index + " position element deleted");
                break;
            } else {
                // si la posición actual no es el indice
                // continua con el siguiente nodo
                prev = currNode;
                currNode = currNode.next;
                counter++;
            }
        }
```

```
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         // Borramos 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>
256cafc1e53eea1597da9\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
        Run | Debug
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      }
18  }
19
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\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 {
        Run | Debug
5       public static void main(String[] args) {
6           // Creando el objeto de la
7           // clase lista enlazada
8           LinkedList<String> ll = new LinkedList<String>();
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\bi
[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 {
        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  }
```

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 {
        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  }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\bi
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 {
        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          // 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)
18          System.out.print(str + " ");
19      }
20  }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\b
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 {
       Run | Debug
5      public static void main(String[] args) {
6          LinkedList<Integer> list = new LinkedList<Integer>();
7      list.add(e:123);
8      list.add(e:12);
9      list.add(e:11);
10     list.add(e:1134);
11     System.out.println("LinkedList: " + list);
12     Object[] a = list.toArray();
13     System.out.print(s:"Después de convertir LinkedList a un Array: ");
14     for (Object element : a)
15         System.out.print(element + " ");
16     }
17 }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\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 {
       Run | Debug
5      public static void main(String[] args) {
6          LinkedList<String> list = new LinkedList<String>();
7          list.add(e:"Uno, Dos, Tres ");
8          list.add(e:"Cuatro ");
9          // Mostrar el tamaño de la lista
10         System.out.println("El tamaño de la lista es: " + list.size());
11     }
12 }
```

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\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

```java
package Laboratorio4;
import java.util.LinkedList;

public class GFG7 {
    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<Integer>();
        list.add(e:10);
        list.add(e:20);
        list.add(e:30);
        System.out.println("LinkedList:" + list);
        System.out.println("El primer elemento removido es: " + list.removeFirst());
        // Mostrando la lista final
        System.out.println("Final LinkedList:" + list);
    }
}

```

Resultados:

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

```java
package Laboratorio4;
import java.util.LinkedList;

public class GFG8 {
    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<Integer>();
        list.add(e:10);
        list.add(e:20);
        list.add(e:30);
        System.out.println("LinkedList:" + list);
        // Remueve la cola usando removeLast()
        System.out.println("The last element is removed: " + list.removeLast());
        // Muestra la lista final
        System.out.println("Final LinkedList:" + list);
        // Remueve el último elemento usando removeLast()
        System.out.println("The last element is removed: " + list.removeLast());
        // Mostrando la lista final
        System.out.println("Final LinkedList:" + list);
    }
}

```

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

Resultados:

```
PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab>
256cafc1e53eea1597da9\redhat.java\jdt_ws\EDA Lab_4ccd2262\bi
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 > ProblemasPropuestos > Ejercicio1.java > Ejercicio1 > LinkedList<E>

```java
package Laboratorio4.ProblemasPropuestos;

public class Ejercicio1 {

    // Clase Nodo doblemente enlazado
    static class NodoDoble<E> {
        E dato;
        NodoDoble<E> anterior;
        NodoDoble<E> siguiente;

        public NodoDoble(E dato) {
            this.dato = dato;
            this.anterior = null;
            this.siguiente = null;
        }
    }

    // Clase LinkedList genérica
    static class LinkedList<E> {
        private NodoDoble<E> cabeza;
        private NodoDoble<E> cola;

        // Insertar al final
        public void insertar(E valor) {
            NodoDoble<E> nuevo = new NodoDoble<>(valor);
            if (cabeza == null) {
                cabeza = cola = nuevo;
            } else {
                cola.siguiente = nuevo;
                nuevo.anterior = cola;
                cola = nuevo;
            }
        }

        // Imprimir la lista
        public void imprimir() {
            NodoDoble<E> actual = cabeza;
            while (actual != null) {
                System.out.print(actual.dato + " ");
                actual = actual.siguiente;
            }
            System.out.println();
        }

        // Método genérico de ejemplo
        public E obtenerPrimero() {
            return cabeza != null ? cabeza.dato : null;
        }
    }

    // Clase principal
    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>();
        for (int i = 1; i <= 10; i++) {
            lista.insertar(i);
        }

        System.out.println(x:"Lista doblemente enlazada (1 al 10):");
        lista.imprimir();

        System.out.println("Primer elemento (método genérico): " + lista.obtenerPrimero());
    }
}
```

- 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 > ...

```java
package Laboratorio4.ProblemasPropuestos;

public class Ejercicio2 {

    // Clase Nodo para lista circular
    static class Nodo<E> {
        E dato;
        Nodo<E> siguiente;

        public Nodo(E dato) {
            this.dato = dato;
            this.siguiente = null;
        }
    }

```

```java
    // Lista circular genérica
    static class LinkedList<E> {
        private Nodo<E> cabeza;

        // Insertar al final
        public void insertar(E valor) {
            Nodo<E> nuevo = new Nodo<>(valor);
            if (cabeza == null) {
                cabeza = nuevo;
                cabeza.siguiente = cabeza; // Enlaza a sí misma
            } else {
                Nodo<E> actual = cabeza;
                while (actual.siguiente != cabeza) {
                    actual = actual.siguiente;
                }
                actual.siguiente = nuevo;
                nuevo.siguiente = cabeza;
            }
        }

        // Imprimir lista circular
        public void imprimir() {
            if (cabeza == null) {
                System.out.println(x:"Lista vacía");
                return;
            }

            Nodo<E> actual = cabeza;
            do {
                System.out.print(actual.dato + " ");
                actual = actual.siguiente;
            } while (actual != cabeza);
            System.out.println();
        }

        // Método genérico: obtener cabeza
        public E obtenerPrimero() {
            return cabeza != null ? cabeza.dato : null;
        }
    }
```

```java
    // Método principal
    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>();
        for (int i = 1; i <= 12; i++) {
            lista.insertar(i);
        }

        System.out.println(x:"Lista circular (1 al 12):");
        lista.imprimir();

        System.out.println("Primer elemento (método genérico): " + lista.obtenerPrimero());
    }
}
```

- 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[])

```java
package Laboratorio4.ProblemasPropuestos;
import java.util.LinkedList;

public class Ejercicio3 {

    public static <E> E obtenerPrimero(LinkedList<E> lista) {
        return lista.isEmpty() ? null : lista.getFirst();
    }

    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>();
        for (int i = 1; i <= 10; i++) {
            lista.add(i);
        }

        System.out.println(x:"Lista doblemente enlazada (1 al 10) usando java.util.LinkedList:");
        for (Integer num : lista) {
            System.out.print(num + " ");
        }

        System.out.println("\nPrimer elemento (método genérico): " + obtenerPrimero(lista));
    }
}
```

- 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
sers\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[])

```java
package Laboratorio4.ProblemasPropuestos;
import java.util.LinkedList;

public class Ejercicio4 {

    public static <E> E obtenerPrimero(LinkedList<E> lista) {
        return lista.isEmpty() ? null : lista.getFirst();
    }

    public static <E> void imprimirCircular(LinkedList<E> lista, int repeticiones) {
        if (lista.isEmpty()) {
            System.out.println(x:"Lista vacía");
            return;
        }

        int size = lista.size();
        for (int i = 0; i < repeticiones; i++) {
            System.out.print(lista.get(i % size) + " ");
        }
        System.out.println();
    }

    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>();

        for (int i = 1; i <= 12; i++) {
            lista.add(i);
        }

        System.out.println(x:"Lista circular simulada (1 al 12, dos vueltas):");
        imprimirCircular(lista, repeticiones:24);

        System.out.println("Primer elemento (método genérico): " + obtenerPrimero(lista));
    }
}
```

- 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\bi
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[])

```java
package Laboratorio4.ProblemasPropuestos;
import java.util.Scanner;

public class Ejercicio5 {

    static class Nodo<E> {
        E dato;
        Nodo<E> siguiente;

        public Nodo(E dato) {
            this.dato = dato;
            this.siguiente = null;
        }
    }

    static class LinkedList<E> {
        private Nodo<E> cabeza;

        public void insert(E dato) {
            Nodo<E> nuevo = new Nodo<>(dato);
            if (cabeza == null) {
                cabeza = nuevo;
            } else {
                Nodo<E> actual = cabeza;
                while (actual.siguiente != null) {
                    actual = actual.siguiente;
                }
                actual.siguiente = nuevo;
            }
        }

        public void addFirst(E dato) {
            Nodo<E> nuevo = new Nodo<>(dato);
            nuevo.siguiente = cabeza;
            cabeza = nuevo;
        }

        public void addLast(E dato) {
            insert(dato);
        }

```

```java
        public void deleteByKey(E key) {
            if (cabeza == null) return;

            if (cabeza.dato.equals(key)) {
                cabeza = cabeza.siguiente;
                return;
            }

            Nodo<E> actual = cabeza;
            while (actual.siguiente != null && !actual.siguiente.dato.equals(key)) {
                actual = actual.siguiente;
            }

            if (actual.siguiente != null) {
                actual.siguiente = actual.siguiente.siguiente;
            }
        }

        public void deleteAtPosition(int posicion) {
            if (posicion < 0 || cabeza == null) return;

            if (posicion == 0) {
                cabeza = cabeza.siguiente;
                return;
            }
```

```
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         }
```

```
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         }
```

```
        do {
            System.out.println(x:"\n--- Menú Lista Enlazada Simple ---");
            System.out.println(x:"1. Mostrar lista");
            System.out.println(x:"2. Insertar al inicio");
            System.out.println(x:"3. Insertar al final");
            System.out.println(x:"4. Eliminar por valor");
            System.out.println(x:"5. Eliminar por posición");
            System.out.println(x:"6. Eliminar primero");
            System.out.println(x:"7. Eliminar último");
            System.out.println(x:"8. Tamaño de la lista");
            System.out.println(x:"9. Salir");
            System.out.print(s:"Elige una opción: ");
            opcion = scanner.nextInt();

            switch (opcion) {
                case 1 -> lista.printList();
                case 2 -> {
                    System.out.print(s:"Dato a insertar al inicio: ");
                    int valor = scanner.nextInt();
                    lista.addFirst(valor);
                }
                case 3 -> {
                    System.out.print(s:"Dato a insertar al final: ");
                    int valor = scanner.nextInt();
                    lista.addLast(valor);
                }
                case 4 -> {
                    System.out.print(s:"Valor a eliminar: ");
                    int valor = scanner.nextInt();
                    lista.deleteByKey(valor);
                }
                case 5 -> {
                    System.out.print(s:"Posición a eliminar: ");
                    int pos = scanner.nextInt();
                    lista.deleteAtPosition(pos);
                }
                case 6 -> lista.removeFirst();
                case 7 -> lista.removeLast();
                case 8 -> System.out.println("Tamaño: " + lista.size());
                case 9 -> System.out.println(x:"Saliendo...");
                default -> System.out.println(x:"Opción inválida.");
            }

        } while (opcion != 9);

        scanner.close();
    }
}
```

- 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

```java
package Laboratorio4.ProblemasPropuestos;
import java.util.Scanner;

public class Ejercicio6 {

    static class Nodo<E> {
        E dato;
        Nodo<E> anterior;
        Nodo<E> siguiente;

        public Nodo(E dato) {
            this.dato = dato;
            this.anterior = null;
            this.siguiente = null;
        }
    }

```

```java
static class LinkedList<E> {
    private Nodo<E> cabeza;
    private Nodo<E> cola;

    public void insert(E dato) {
        Nodo<E> nuevo = new Nodo<>(dato);
        if (cabeza == null) {
            cabeza = cola = nuevo;
        } else {
            cola.siguiente = nuevo;
            nuevo.anterior = cola;
            cola = nuevo;
        }
    }

    public void addFirst(E dato) {
        Nodo<E> nuevo = new Nodo<>(dato);
        if (cabeza == null) {
            cabeza = cola = nuevo;
        } else {
            nuevo.siguiente = cabeza;
            cabeza.anterior = nuevo;
            cabeza = nuevo;
        }
    }

    public void addLast(E dato) {
        insert(dato);
    }

```

```java
    public void deleteByKey(E key) {
        Nodo<E> actual = cabeza;
        while (actual != null && !actual.dato.equals(key)) {
            actual = actual.siguiente;
        }

        if (actual == null) return;

        if (actual == cabeza) {
            cabeza = cabeza.siguiente;
            if (cabeza != null) cabeza.anterior = null;
        } else if (actual == cola) {
            cola = cola.anterior;
            if (cola != null) cola.siguiente = null;
        } else {
            actual.anterior.siguiente = actual.siguiente;
            actual.siguiente.anterior = actual.anterior;
        }
    }

```

```java
    public void deleteAtPosition(int pos) {
        if (pos < 0) return;

        Nodo<E> actual = cabeza;
        for (int i = 0; actual != null && i < pos; i++) {
            actual = actual.siguiente;
        }

        if (actual == null) return;

        if (actual == cabeza) {
            cabeza = cabeza.siguiente;
            if (cabeza != null) cabeza.anterior = null;
        } else if (actual == cola) {
            cola = cola.anterior;
            if (cola != null) cola.siguiente = null;
        } else {
            actual.anterior.siguiente = actual.siguiente;
            actual.siguiente.anterior = actual.anterior;
        }
    }

    public void removeFirst() {
        if (cabeza != null) {
            cabeza = cabeza.siguiente;
            if (cabeza != null) cabeza.anterior = null;
            else cola = null;
        }
    }

    public void removeLast() {
        if (cola != null) {
            cola = cola.anterior;
            if (cola != null) cola.siguiente = null;
            else cabeza = null;
        }
    }
```

```java
    public int size() {
        int count = 0;
        Nodo<E> actual = cabeza;
        while (actual != null) {
            count++;
            actual = actual.siguiente;
        }
        return count;
    }

    public void printList() {
        Nodo<E> actual = cabeza;
        while (actual != null) {
            System.out.print(actual.dato + " ");
            actual = actual.siguiente;
        }
        System.out.println();
    }
}
```

```java
    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);
        int opcion;

        for (int i = 1; i <= 10; i++) {
            lista.insert(i);
        }

        do {
            System.out.println("\n--- Menú Lista Doblemente Enlazada ---");
            System.out.println("1. Mostrar lista");
            System.out.println("2. Insertar al inicio");
            System.out.println("3. Insertar al final");
            System.out.println("4. Eliminar por valor");
            System.out.println("5. Eliminar por posición");
            System.out.println("6. Eliminar primero");
            System.out.println("7. Eliminar último");
            System.out.println("8. Tamaño de la lista");
            System.out.println("9. Salir");
            System.out.print("Elige una opción: ");
            opcion = scanner.nextInt();

            switch (opcion) {
                case 1 -> lista.printList();
                case 2 -> {
                    System.out.print("Dato a insertar al inicio: ");
                    int valor = scanner.nextInt();
                    lista.addFirst(valor);
                }
                case 3 -> {
                    System.out.print("Dato a insertar al final: ");
                    int valor = scanner.nextInt();
                    lista.addLast(valor);
                }
                case 4 -> {
                    System.out.print("Valor a eliminar: ");
                    int valor = scanner.nextInt();
                    lista.deleteByKey(valor);
                }
                    System.out.print("Valor a eliminar: ");
                    int valor = scanner.nextInt();
                    lista.deleteByKey(valor);
                }
                case 5 -> {
                    System.out.print("Posición a eliminar: ");
                    int pos = scanner.nextInt();
                    lista.deleteAtPosition(pos);
                }
                case 6 -> lista.removeFirst();
                case 7 -> lista.removeLast();
                case 8 -> System.out.println("Tamaño: " + lista.size());
                case 9 -> System.out.println("Saliendo...");
                default -> System.out.println("Opción inválida.");
            }
        } while (opcion != 9);

        scanner.close();
    }
}
```

- 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
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>
```

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)

```java
package Laboratorio4.ProblemasPropuestos;
import java.util.Scanner;

public class Ejercicio7 {

    static class Nodo<E> {
        E dato;
        Nodo<E> siguiente;

        public Nodo(E dato) {
            this.dato = dato;
            this.siguiente = null;
        }
    }

    static class LinkedList<E> {
        private Nodo<E> ultimo;

        public void insert(E dato) {
            Nodo<E> nuevo = new Nodo<>(dato);
            if (ultimo == null) {
                nuevo.siguiente = nuevo;
                ultimo = nuevo;
            } else {
                nuevo.siguiente = ultimo.siguiente;
                ultimo.siguiente = nuevo;
                ultimo = nuevo;
            }
        }

```

```java
public void addFirst(E dato) {
    Nodo<E> nuevo = new Nodo<>(dato);
    if (ultimo == null) {
        nuevo.siguiente = nuevo;
        ultimo = nuevo;
    } else {
        nuevo.siguiente = ultimo.siguiente;
        ultimo.siguiente = nuevo;
    }
}

public void addLast(E dato) {
    insert(dato);
}

public void deleteByKey(E key) {
    if (ultimo == null) return;

    Nodo<E> actual = ultimo.siguiente;
    Nodo<E> anterior = ultimo;
    do {
        if (actual.dato.equals(key)) {
            if (actual == ultimo && actual == ultimo.siguiente) {
                ultimo = null;
            } else {
                anterior.siguiente = actual.siguiente;
                if (actual == ultimo) {
                    ultimo = anterior;
                }
            }
            return;
        }
        anterior = actual;
        actual = actual.siguiente;
    } while (actual != ultimo.siguiente);
}
```

```java
public void deleteAtPosition(int pos) {
    if (ultimo == null || pos < 0) return;

    Nodo<E> actual = ultimo.siguiente;
    Nodo<E> anterior = ultimo;

    for (int i = 0; i < pos; i++) {
        anterior = actual;
        actual = actual.siguiente;
        if (actual == ultimo.siguiente) return;
    }

    if (actual == ultimo && actual == ultimo.siguiente) {
        ultimo = null;
    } else {
        anterior.siguiente = actual.siguiente;
        if (actual == ultimo) {
            ultimo = anterior;
        }
    }
}

public void removeFirst() {
    if (ultimo == null) return;

    Nodo<E> primero = ultimo.siguiente;
    if (primero == ultimo) {
        ultimo = null;
    } else {
        ultimo.siguiente = primero.siguiente;
    }
}
```

```java
    public void removeFirst() {
        if (ultimo == null) return;

        Nodo<E> primero = ultimo.siguiente;
        if (primero == ultimo) {
            ultimo = null;
        } else {
            ultimo.siguiente = primero.siguiente;
        }
    }

    public void removeLast() {
        if (ultimo == null) return;

        Nodo<E> actual = ultimo.siguiente;
        Nodo<E> anterior = ultimo;

        if (actual == ultimo) {
            ultimo = null;
            return;
        }

        while (actual.siguiente != ultimo.siguiente) {
            anterior = actual;
            actual = actual.siguiente;
        }

        anterior.siguiente = ultimo.siguiente;
        ultimo = anterior;
    }

    public int size() {
        if (ultimo == null) return 0;
        int count = 1;
        Nodo<E> actual = ultimo.siguiente;
        while (actual != ultimo) {
            count++;
            actual = actual.siguiente;
        }
        return count;
    }
```

```java
    public void printList() {
        if (ultimo == null) {
            System.out.println(x:"Lista vacía");
            return;
        }
        Nodo<E> actual = ultimo.siguiente;
        do {
            System.out.print(actual.dato + " ");
            actual = actual.siguiente;
        } while (actual != ultimo.siguiente);
        System.out.println();
    }
}
```

```
    Run | Debug
    public static void main(String[] args) {
        LinkedList<Integer> lista = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);
        int opcion;

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

        do {
            System.out.println(x:"\n--- Menú Lista Circular ---");
            System.out.println(x:"1. Mostrar lista");
            System.out.println(x:"2. Insertar al inicio");
            System.out.println(x:"3. Insertar al final");
            System.out.println(x:"4. Eliminar por valor");
            System.out.println(x:"5. Eliminar por posición");
            System.out.println(x:"6. Eliminar primero");
            System.out.println(x:"7. Eliminar último");
            System.out.println(x:"8. Tamaño de la lista");
            System.out.println(x:"9. Salir");
            System.out.print(s:"Elige una opción: ");
            opcion = scanner.nextInt();

            switch (opcion) {
                case 1 -> lista.printList();
                case 2 -> {
                    System.out.print(s:"Dato a insertar al inicio: ");
                    int valor = scanner.nextInt();
                    lista.addFirst(valor);
                }
                case 3 -> {
                    System.out.print(s:"Dato a insertar al final: ");
                    int valor = scanner.nextInt();
                    lista.addLast(valor);
                }
                case 4 -> {
                    System.out.print(s:"Valor a eliminar: ");
                    int valor = scanner.nextInt();
                    lista.deleteByKey(valor);
                }
                case 5 -> {
                    System.out.print(s:"Posición a eliminar: ");
                    int pos = scanner.nextInt();
                    lista.deleteAtPosition(pos);
                }
                case 6 -> lista.removeFirst();
                case 7 -> lista.removeLast();
                case 8 -> System.out.println("Tamaño: " + lista.size());
                case 9 -> System.out.println(x:"Saliendo...");
                default -> System.out.println(x:"Opción inválida.");
            }

        } while (opcion != 9);

        scanner.close();
    }
}
```

- 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
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>
```

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

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 BIBLIOGRÁFIA 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

TÉCNICAS:	**INSTRUMENTOS:**
Actividades Resueltas *Ejercicios Propuestos*	*Rubricas*

CRITERIOS DE EVALUACIÓN

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