```java


import java.util.Scanner;


class Node {
  protected int data;  // Almacena el valor del nodo
  protected Node next, prev;  // Referencias al nodo siguiente y anterior

  /* Constructor sin parámetros */
  public Node() {
    next = null;
    prev = null;
    data = 0;
  }

  /* Constructor con parámetros */
  public Node(int d, Node n, Node p) {
    data = d;
    next = n;
    prev = p;
  }

  /* Función para establecer la referencia al siguiente nodo */
  public void setLinkNext(Node n) {
    next = n;
  }

  /* Función para establecer la referencia al nodo anterior */
  public void setLinkPrev(Node p) {
    prev = p;
  }

  /* Función para obtener la referencia al siguiente nodo */
  public Node getLinkNext() {
    return next;
  }

  /* Función para obtener la referencia al nodo anterior */
  public Node getLinkPrev() {
    return prev;
  }

  /* Función para establecer el valor del nodo */
  public void setData(int d) {
    data = d;
  }

  /* Función para obtener el valor del nodo */
  public int getData() {
    return data;
  }
}


/* Clase linkedList */
class linkedList {
```

```
  protected Node start;  // Referencia al primer nodo de la lista
  protected Node end;  // Referencia al último nodo de la lista
  public int size;  // Tamaño de la lista

  /* Constructor */
  public linkedList() {
    start = null;
    end = null;
    size = 0;
  }

  /* Función para verificar si la lista está vacía */
  public boolean isEmpty() {
    return start == null;
  }

  /* Función para obtener el tamaño de la lista */
  public int getSize() {
    return size;
  }

/* Función para insertar un elemento al inicio */
public void insertAtStart(int val) {
    Node nptr = new Node(val, null, null); // Crear un nuevo nodo con el valor dado
    if (start == null) { // Si la lista está vacía
        start = nptr; // Establecer el nuevo nodo como el primer nodo
        end = start; // Establecer el nuevo nodo como el último nodo también (ya que es el único nodo en la lista)
    } else { // Si la lista no está vacía
        start.setLinkPrev(nptr); // Establecer el enlace previo del nodo existente en el inicio para que apunte al nuevo nodo
        nptr.setLinkNext(start); // Establecer el enlace siguiente del nuevo nodo para que apunte al nodo existente en el inicio
        start = nptr; // Establecer el nuevo nodo como el primer nodo
    }
    size++; // Incrementar el tamaño de la lista
}
/*
En insertAtStart, esta función inserta un nuevo nodo al inicio de la lista
enlazada doble. Si la lista está vacía, el nuevo nodo se convierte tanto
en el primer nodo como en el último nodo de la lista. Si la lista no está
vacía, el nuevo nodo se enlaza correctamente con el nodo existente en el
inicio y se establece como el nuevo primer nodo. Al final, se incrementa
el tamaño de la lista en uno.
*/


/* Función para insertar un elemento al final */
public void insertAtEnd(int val) {
    Node nptr = new Node(val, null, null); // Crear un nuevo nodo con el valor dado
    if (start == null) { // Si la lista está vacía
        start = nptr; // Establecer el nuevo nodo como el primer nodo
        end = start; // Establecer el nuevo nodo como el último nodo también (ya que es el único nodo en la lista)
    } else { // Si la lista no está vacía
        nptr.setLinkPrev(end); // Establecer el enlace previo del nuevo nodo para que apunte al nodo existente en el final
        end.setLinkNext(nptr); // Establecer el enlace siguiente del nodo existente en el final para que apunte al nuevo nodo
        end = nptr; // Establecer el nuevo nodo como el último nodo
```

```
    }
    size++; // Incrementar el tamaño de la lista
}


/* Función para insertar un elemento en una posición */
/*
Dentro del `if(i == pos)` se realizan los cambios necesarios para insertar un nuevo nodo en una posición
específica de la lista. Veamos cómo funciona paso a paso:

1. `Node tmp = ptr.getLinkNext();`: Se guarda una referencia al nodo siguiente al nodo actual en la variable
`tmp`. Esto es necesario para mantener la continuidad de la lista después de insertar el nuevo nodo.

2. `ptr.setLinkNext(nptr);`: Se establece el enlace siguiente del nodo actual (`ptr`) para que apunte al
nuevo nodo (`nptr`). Ahora el nodo actual apunta al nuevo nodo.

3. `nptr.setLinkPrev(ptr);`: Se establece el enlace previo del nuevo nodo (`nptr`) para que apunte al nodo
actual (`ptr`). Ahora el nuevo nodo apunta al nodo anterior.

4. `nptr.setLinkNext(tmp);`: Se establece el enlace siguiente del nuevo nodo (`nptr`) para que apunte al nodo
siguiente (`tmp`). Ahora el nuevo nodo apunta al nodo que originalmente seguía al nodo actual.

5. `tmp.setLinkPrev(nptr);`: Se establece el enlace previo del nodo siguiente (`tmp`) para que apunte al nuevo
nodo (`nptr`). Ahora el nodo siguiente apunta al nuevo nodo.

En resumen, estos cambios realizados dentro del `if` se encargan de ajustar los enlaces de los nodos para insertar
correctamente el nuevo nodo en la posición especificada. Se aseguran de que el enlace previo y el enlace siguiente
del nuevo nodo y los nodos adyacentes estén correctamente establecidos, manteniendo la estructura de la lista
doblemente enlazada.

Recuerda que la variable `ptr` se utiliza como un puntero para recorrer la lista y encontrar el nodo en la posición
`pos`.
*/
public void insertAtPos(int val, int pos) {
    Node nptr = new Node(val, null, null); // Crear un nuevo nodo con el valor dado

    if (pos == 1) { // Si la posición es 1, insertar el nuevo nodo al inicio de la lista
        insertAtStart(val);
        return;
    }

    Node ptr = start; // Puntero para recorrer la lista
    for (int i = 2; i <= size; i++) { // Comenzando desde la posición 2 hasta el tamaño de la lista
        if (i == pos) { // Cuando se alcanza la posición deseada
            Node tmp = ptr.getLinkNext(); // Obtener el nodo siguiente al nodo actual
```

```
159             ptr.setLinkNext(nptr); // Establecer el enlace siguiente del nodo actual
    para que apunte al nuevo nodo
160             nptr.setLinkPrev(ptr); // Establecer el enlace previo del nuevo nodo para
    que apunte al nodo actual
161             nptr.setLinkNext(tmp); // Establecer el enlace siguiente del nuevo nodo
    para que apunte al nodo siguiente
162             tmp.setLinkPrev(nptr); // Establecer el enlace previo del nodo siguiente
    para que apunte al nuevo nodo
163         }
164         ptr = ptr.getLinkNext(); // Mover el puntero al siguiente nodo
165     }
166     size++; // Incrementar el tamaño de la lista
167 }
168
169
170 /* Función para eliminar un nodo en una posición */
171 public void deleteAtPos(int pos) {
172     if (pos == 1) { // Si la posición es 1 (primer nodo)
173         if (size == 1) { // Si la lista solo contiene un nodo
174             start = null; // Se establece el inicio y fin a null, eliminando el nodo
175             end = null;
176             size = 0; // Se actualiza el tamaño a 0
177             return; // Se finaliza la función
178         }
179         start = start.getLinkNext(); // Se actualiza el inicio para que apunte al
    siguiente nodo
180         start.setLinkPrev(null); // Se establece el enlace previo del nuevo inicio a
    null
181         size--; // Se reduce el tamaño de la lista
182         return; // Se finaliza la función
183     }
184     if (pos == size) { // Si la posición es igual al tamaño (último nodo)
185         end = end.getLinkPrev(); // Se actualiza el fin para que apunte al nodo
    previo
186         end.setLinkNext(null); // Se establece el enlace siguiente del nuevo fin a
    null
187         size--; // Se reduce el tamaño de la lista
188     }
189     Node ptr = start.getLinkNext(); // Puntero para recorrer la lista, se inicia
    desde el segundo nodo
190     for (int i = 2; i <= size; i++) { // Comenzando desde la posición 2 hasta el
    tamaño de la lista
191         if (i == pos) { // Cuando se alcanza la posición deseada
192             Node p = ptr.getLinkPrev(); // Obtener el nodo previo al nodo actual
193             Node n = ptr.getLinkNext(); // Obtener el nodo siguiente al nodo actual
194             p.setLinkNext(n); // Establecer el enlace siguiente del nodo previo para
    que apunte al nodo siguiente
195             n.setLinkPrev(p); // Establecer el enlace previo del nodo siguiente para
    que apunte al nodo previo
196             size--; // Se reduce el tamaño de la lista
197             return; // Se finaliza la función
198         }
199         ptr = ptr.getLinkNext(); // Mover el puntero al siguiente nodo
200     }
201 }
202
203
204   /* Función para mostrar el estado de la lista */
205   public void display() {
206     System.out.print("\nListas doblemente enlazadas(GRUPO 1) = ");
```

```
207     if (size == 0) {
208       System.out.print("empty\n");
209       return;
210     }
211     if (start.getLinkNext() == null) {
212       System.out.println(start.getData());
213       return;
214     }
215     Node ptr = start;
216     System.out.print(start.getData() + " <-> ");
217     ptr = start.getLinkNext();
218 
219     while (ptr.getLinkNext() != null) {
220       System.out.print(ptr.getData() + " <-> ");
221       ptr = ptr.getLinkNext();
222     }
223     System.out.print(ptr.getData() + "\n");
224   }
225 }
226 
227 /* Class DoublsyLinkedList */
228 
229 public class ListasDobles {
230   public static void main(String[] args) {
231     Scanner scan = new Scanner(System.in);
232 
233     /* Creating object of linkedList */
234     linkedList list = new linkedList();
235     System.out.println("PRUEBA DE LISTAS DOBLEMENTE ENLAZADAS\n");
236     char ch;
237 
238     /* Perform list operations */
239 
240     do {
241       System.out.println("\nListas doblemente enlazadas(GRUPO 01)\n");
242       System.out.println("1. Insertar en el inicio");
243       System.out.println("2. Insertar al final");
244       System.out.println("3. Insertar en la posicion");
245       System.out.println("4. Eliminar la posicion");
246       System.out.println("5. Revisar si esta vacio");
247       System.out.println("6. Obtener tamaño");
248 
249       int choice = scan.nextInt();
250 
251       switch (choice) {
252       case 1:
253         System.out.println("Escribe el elemento entero para insertarlo");
254         list.insertAtStart(scan.nextInt());
255         break;
256 
257       case 2:
258         System.out.println("Escribe el elemento entero para insertarlo");
259         list.insertAtEnd(scan.nextInt());
260         break;
261 
262       case 3:
263         System.out.println("Escribe el elemento entero para insertarlo");
264         int num = scan.nextInt();
265         System.out.println("Escribe la posicion");
266         int pos = scan.nextInt();
```

```
        if (pos < 1 || pos > list.getSize())
          System.out.println("Posicion invalida\n");
        else
          list.insertAtPos(num, pos);
        break;

      case 4:
        System.out.println("Escribe la posicion");
        int p = scan.nextInt();
        if (p < 1 || p > list.getSize())
          System.out.println("Posicion invalida\n");
        else
          list.deleteAtPos(p);
        break;

      case 5:
        System.out.println("Estado vacio = " + list.isEmpty());
        break;

      case 6:
        System.out.println("Tamaño = " + list.getSize() + " \n");
        break;

      default:
        System.out.println("Entrada incorrecta \n ");
        break;

      }

      /* Display List */
      list.display();
      System.out.println("\nDeseas continuar (Escribe S o N) \n");
      ch = scan.next().charAt(0);

    } while (ch == 'S' || ch == 's');
  }
}
```