```
public class Node<T> {
	T data;
	Node<T> next;

	Node(T data) {
		this.data = data;
	}
}

public class LinkListMethods<T> {
	Scanner sc = new Scanner(System.in);
	// test method
	public static Node<Integer> createLinkedList() {
		Node<Integer> n1 = new Node<>(50);
		Node<Integer> n2 = new Node<>(30);
		Node<Integer> n3 = new Node<>(10);
		Node<Integer> n4 = new Node<>(60);
		Node<Integer> n5 = new Node<>(40);
		n1.next = n2;
		n2.next = n3;
		n3.next = n4;
		n4.next = n5;
		return n1;
	}

	// 1st mehtod
	public Node<Integer> takeInput() {
		int data = sc.nextInt();
		// ref of Node
		Node<Integer> head = null;
		while (data != -1) {
			Node<Integer> currentNode = new Node<>(data);
			if (head == null) {
				head = currentNode;
			}
			else {
				// keeping tail node to keep track of the last node
				Node<Integer> tail = head;
				while (tail.next != null) {
					tail = tail.next;
				}
				tail.next = currentNode;
			}
			data = sc.nextInt();
		}
		return head;
	}

	// 2nd method
	public Node<Integer> takeInputBetter() {
		int data = sc.nextInt();
		Node<Integer> head = null, tail = null;
		while (data != -1) {
			Node<Integer> currentNode = new Node<>(data);
			if (head == null) {
				head = currentNode;
				tail = currentNode;
			}
```

```
            else {
                  tail.next = currentNode;
                  // we have to update the tail to the last node
                  tail = currentNode; // or
                  //tail = tail.next;
            }
            data = sc.nextInt();
      }
      return head;
}

// 3rd mehtod
public void print(Node<T> head) {
            if (head == null) {
                  return;
            }
            Node<T> temp = head;
            System.out.print(temp.data + " ");
            temp = temp.next;
            print(temp);
      }

// 4th method
public int length(Node<T> head) {
      int count = 0;
      while (head != null) {
            count++;
            head = head.next;
      }
      return count;
}

// 5th mehtod
public int lengthRec(Node<T> head) {
      int count = 1;
      if (head == null)
            return 0;
      return count + lengthRec(head.next);
}

// 6th mehtod
public Node<Integer> insertLast(Node<Integer> head) {
      int data = sc.nextInt();
      Node<Integer> insert = new Node<>(data);
      Node<Integer> tail = head;
      while (tail.next != null) {
            tail = tail.next;
      }
      tail.next = insert;
      return head;
}

// 7th method
public void insert(Node<Integer> head, int pos, int data) {
      Node<Integer> nodeToInsert = new Node<>(data);
      if (pos == 0) {
            nodeToInsert.next = head;
      }
      else {
            int count = 0;
```

```
                Node<Integer> prev = head;
                // iterating till position -1 so to make space
                while (count < pos - 1 && prev != null) {
                        count++;
                        prev = prev.next; // putting nodes in variable
                }
                if (prev != null) {
                // appending the last list to the node to insert
                        nodeToInsert.next = prev.next;
                // now connecting the inserted node to list
                        prev.next = nodeToInsert;
                }
        }
    }

    // 8th method
    public Node<Integer> deleteNode(Node<Integer> head, int pos) {
        // If the list is empty or position is invalid, return head as it is
        if (head == null || pos < 0) {
            return head;
        }
        // If position is 0, delete the first node
        if (pos == 0) {
            return head.next;
        }
        // Traverse the list to find the node at position pos-1
        Node<Integer> previous = head;
        for (int i = 0; previous != null && i < pos - 1; i++) {
            previous = previous.next;
        }
        // If position is greater than or equal to the length of the list,
        // return the list as it is
        if (previous == null || previous.next == null) {
            return head;
        }
        // Skip the node at position pos
        previous.next = previous.next.next;
        return head;
    }
}
```