```java
// Author: Tyerone Chen
// Create Date: 4/15/2025

import java.util.AbstractList;

/**
 * A generic singly linked list that extends Java's AbstractList<T>.
 * This version supports dynamic sizing and allows the user to:
 * - Add, remove, and access elements at any index
 * - Use this class anywhere a List<T> would be accepted
 */
public class OurLinkedList<T> extends AbstractList<T> {

    /**
     * Private inner Node class to represent each item in the list.
     * Each node holds data and a reference to the next node.
     */
    private static class Node<T> {
        T data;           // The value stored in this node
        Node<T> next;     // A reference to the next node in the list

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

    private Node<T> head;  // Points to the first node in the list

    /**
     * Default constructor. Creates an empty list.
     */
    public OurLinkedList() {
        head = null;
    }

    /**
     * Returns the element at the specified index.
     * Must walk the list from the head until the target index is reached.
     */
    @Override
    public T get(int index) {
        if (index < 0 || index > size()) throw new IndexOutOfBoundsException();

        Node<T> current_node = head;

        for (int i = 0; i < index; i++){
            current_node = current_node.next;
        }

        return current_node.data;
    }

    /**
     * Replaces the element at the specified index with the given element.
     * Returns the old value that was replaced.
     */
    @Override
    public T set(int index, T element) {
        if (index < 0 || index > size()) throw new IndexOutOfBoundsException();

        Node<T> current_node = head;

        for (int i = 0; i < index; i++){
            current_node = current_node.next;
        }

        T old_data = current_node.data;
```

```
 69         current_node.data = element;
 70
 71         return old_data;
 72     }
 73
 74     /**
 75      * Inserts an element at the specified index.
 76      * Shifts the current node at that index (and everything after) forward.
 77      */
 78     @Override
 79     public void add(int index, T element) {
 80       if (index < 0 || index > size()) throw new IndexOutOfBoundsException();
 81
 82       Node<T> new_node = new Node<T>(element);
 83
 84       // Case 1: The LinkedList is empty or we are adding at 0, were we will add the value to the beggining and p
ush any head value up one
 85       if (index == 0){
 86          new_node.next = head;
 87          head = new_node;
 88          return;
 89       }
 90
 91       // Case 2: The LinkedList is filled with data, there we will place the value ther and push other value up
 92       Node<T> current_node = head;
 93
 94       for (int i = 0; i < index - 1; i++){
 95          current_node = current_node.next;
 96       }
 97
 98       new_node.next = current_node.next;
 99       current_node.next = new_node;
100     }
101
102     /**
103      * Removes and returns the element at the specified index.
104      * Relinks the list so the removed node is skipped over.
105      */
106     @Override
107     public T remove(int index) {
108       if (index < 0 || index > size()) throw new IndexOutOfBoundsException();
109       Node<T> removed_node;
110
111       // Case 1:
112       if (index == 0){
113          removed_node = head;
114          head = head.next;
115          return removed_node.data;
116       }
117
118       // Case 2:
119       Node<T> current_node = head;
120
121       for (int i = 0; i < index - 1; i++){
122          current_node = current_node.next;
123       }
124
125       removed_node = current_node.next;
126       current_node.next = removed_node.next;
127
128       return removed_node.data;
129     }
130
131     /**
132      * Returns the number of elements in the list.
133      * Walks through the list and counts nodes.
134      * This is calculated dynamically — no size variable is stored.
135      */
```

```
136     @Override
137     public int size() {
138         Node<T> current_node = head;
139         int count = 0;
140 
141         while (current_node != null){
142             count++;
143             current_node = current_node.next;
144         }
145 
146         return count;
147     }
148 
149     public String toString(){
150       String temp_str = "";
151       Node<T> current_node = head;
152 
153       while (current_node != null){
154          temp_str += current_node.data + " -> ";
155          current_node = current_node.next;
156       }
157 
158       temp_str += "null";
159 
160       return temp_str;
161    }
162 
163    public static void main(String[] args){
164       OurLinkedList oll = new OurLinkedList();
165       oll.add(1);
166       oll.add(2);
167       System.out.println(oll.toString());
168       oll.add(0, -33);
169       System.out.println(oll.toString());
170       oll.add(2, 99);
171       System.out.println(oll.toString());
172       oll.remove(0);
173       System.out.println(oll.toString());
174       System.out.println(oll.get(0));
175       oll.set(0, 5);
176       System.out.println(oll.toString());
177 
178       try {
179          oll.add(35, 5);
180          System.out.println(oll.toString());
181       }
182       catch (IndexOutOfBoundsException err){
183          System.out.println("[ERROR] | Attempted to add value out of LinkedList Bounds");
184       }
185 
186    }
187 
188 }
```