LinkedList Class Development Report

1. Class Declaration and Member Variables

```
package utils;

public class LinkedList<E> {
    private Node<E> first;
    private Node<E> last;
    private int size;

    public LinkedList() {
        first = null;
        last = null;
        size = 0;
    }
```

In the class declaration, defined a generic class LinkedList with three member variables: first, last, and size. Here, first and last represent the head and tail nodes of the linked list, while size indicates the number of elements in the list. In the constructor LinkedList(), initialize first and last to null and size to 0.

2. Adding Elements

```
public boolean add(E item) {
    append(item);
    size++;
    return true;
}
```

The add method is used to add an element to the end of the linked list. In this method, i call the private method append to perform the addition of the element and increment the size after adding the element.

```
private void append(E item) {
    Node<E> newNode = new Node<>(last, item, null);
    if (last == null) {
        first = newNode;
    } else {
        last.next = newNode;
    }
    last = newNode;
}
```

The append method is used to add a new node at the end of the linked list. i first create a new node, newNode, and then decide whether to set it as the head node or append it after the tail node based on whether the current list is empty. Finally, i update the tail node to be the new node.

3. Inserting Elements

```
public void add(int index, E item) {
    checkIndex(index);
    insertBefore(index, item);
    size++;
}
```

The add method is used to insert an element at a specified position in the linked list. In this method, i first call the checkIndex method to validate the index and then call the insertBefore method to insert the element at the specified position, folloid by incrementing the size.

```
private void insertBefore(int index, E item) {
    if (index == size) {
        append(item);
        return;
    }
    Node<E> node = node(index);
    Node<E> newNode = new Node<>(node.prev, item, node);
    if (node == first) {
        first = newNode;
    } else {
        node.prev.next = newNode;
    }
    node.prev = newNode;
}
```

The insertBefore method is used to insert a new node before the specified position in the linked list. If inserting at the end, i call the append method. Otherwise, i locate the node at the insertion position, create a new node, newNode, and update the relationships with adjacent nodes accordingly.

4. Removing Elements

```
public E remove(int index) {
    checkIndex(index);
    E removed = detach(index);
    size--;
    return removed;
}
```

```
public boolean remove(E item) {
    int index = indexOf(item);
    if (index != -1) {
        remove(index);
        return true;
    }
    return false;
}
```

The remove method is used to remove a node at a specified position or with a specified element from the linked list. In the remove(int index) method, i call the private method detach to remove the node and decrement the size afterward. In the remove(E item) method, i first find the index of the specified element and then call remove(int index) to remove it.

```
private E detach(int index) {
    Node<E> node = node(index);
    if (node == first) {
        first = node.next;
    } else {
        node.prev.next = node.next;
    }
    if (node == last) {
        last = node.prev;
    } else {
        node.next.prev = node.prev;
    }
    return node.data;
}
```

The detach method is used to remove the node at the specified position from the linked list. i locate the node to be removed, update the relationships with adjacent nodes based on its position, and return the data of the removed node.

5. Other Methods

In addition to the above methods, the LinkedList class also implements other common methods such as get, indexOf, set, etc., for retrieving node data at a specific position, finding the index of a specific element, replacing node data at a specific position, and so on.

6. Summary

By implementing the LinkedList class, i can perform operations such as adding, inserting, and removing elements from the linked list, as ill as providing convenient methods for accessing and manipulating the elements in the list. The implementation of this class is based on a doubly linked list structure, which effectively supports element insertion and removal operations, offering good performance and flexibility.

The result as follows:

```
After adding 3 elements: [A, B, C]
Element at index 1 is B
After setting element at index 1 to 'C': [A, C]
Size of the list is 2
After inserting 'C' at index 1: [A, C, B]
'B' is found at index 1
After removing element at index 1: [A, C]
```

The junits code as follows:

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

import java.util.LinkedList;

public class LinkedListTest {
	private LinkedList<String> list;

	@BeforeEach
	public void setup() {
		list = new LinkedList<>();
	}

	@Test
	public void testAdd() {
		list.add("A");
		list.add("B");
		list.add("C");

		System.out.println("After adding 3 elements: " + list);
	}

	@Test
	public void testAddAtIndex() {
		list.add("A");
		list.add("B");
		list.add(1, "C");

		System.out.println("After inserting 'C' at index 1: " + list);
	}
```

```
@Test
public void testRemove() {
	list.add("A");
	list.add("B");
	list.add("C");
	list.remove(1);

	System.out.println("After removing element at index 1: " + list);
}

@Test
public void testSet() {
	list.add("A");
	list.add("B");
	list.set(1, "C");

	System.out.println("After setting element at index 1 to 'C': " + list);
}

@Test
public void testIndexOf() {
	list.add("A");
	list.add("B");
	list.add("C");
	int index = list.indexOf("B");

	System.out.println("'B' is found at index " + index);
}

@Test
public void testGet() {
	list.add("A");
	list.add("B");
	String element = list.get(1);

	System.out.println("Element at index 1 is " + element);
}

@Test
public void testSize() {
	list.add("A");
	list.add("B");
	int size = list.size();
```

```
            System.out.println("Size of the list is " + size);
        }
    }
```