package zad4;  
  
public class Main {  
 public static void main(String[] args){  
 TwoWayLinkedList<Integer> list1 = new TwoWayLinkedList<>();  
  
 System.*out*.println("Is empty: " + list1.isEmpty());  
 list1.add(150);  
 list1.add(29);  
 list1.add(9312);  
 list1.add(253);  
 list1.add(350);  
 System.*out*.println("List 1:");  
 list1.printList();  
  
 //List 2  
 TwoWayLinkedList<Integer> list2 = new TwoWayLinkedList<>();  
 list2.add(9999);  
 list2.add(9559);  
 list2.add(9939);  
 list2.add(9534);  
 list2.add(9834);  
 System.*out*.println("List 2:");  
 list2.printList();  
  
 ListOperators<Integer> op = new ListOperators<>();  
 op.addToEnd(list1, list2);  
 System.*out*.println("After merging:");  
 list2.printList();  
  
 //List 3  
 TwoWayLinkedList<Integer> list3 = new TwoWayLinkedList<>();  
 list3.add(150);  
 list3.add(29);  
 list3.add(556);  
 list3.add(253);  
 list3.add(350);  
 System.*out*.println("List 3:");  
 list3.printList();  
  
 //List 4  
 TwoWayLinkedList<Integer> list4 = new TwoWayLinkedList<>();  
 list4.add(9999);  
 list4.add(9559);  
 list4.add(9939);  
 list4.add(9534);  
 list4.add(9834);  
 System.*out*.println("List 4:");  
 list4.printList();  
 op.addAddAt(list3, list4, 3);  
  
 System.*out*.println("After merging:");  
 list3.printList();  
 }  
}

package zad4;  
  
public class ListOperators<E> {  
 public void addToEnd(TwoWayLinkedList<E> list1,TwoWayLinkedList<E> list2){  
 TwoWayLinkedList<E> output = list1;  
  
 output.get(output.size() - 1).insertWithoutChangingNext(list2.get(0));  
 }  
  
 public void addAddAt(TwoWayLinkedList<E> list1, TwoWayLinkedList<E> list2, int index){  
 TwoWayLinkedList<E> output = list1;  
 TwoWayLinkedList.Element before = list1.get(index - 1);  
 TwoWayLinkedList.Element after = list1.get(index);  
 TwoWayLinkedList.Element list2Last = list2.get(list2.size() - 1);  
  
 before.insertWithoutChangingNext(list2.get(0));  
 list2Last.insertWithoutChangingNext(after);  
 }  
}

package zad4;  
  
public class TwoWayLinkedList<E> {  
 public class Element{  
 private E value;  
 private Element next;  
 private Element previous;  
  
 public Element(E value) { this.value = value; }  
  
 public Element getNext() { return next; }  
  
 public void setNext( Element next ) { this.next = next; }  
  
 public Element getPrevious() { return previous; }  
  
 public void setPrevious( Element previous ) { this.previous = previous; }  
  
 public E getValue() { return this.value; }  
  
 public void setValue(E value) { this.value = value; }  
  
 public void insertWithoutChangingNext(Element element){  
 element.setPrevious(this);  
 this.setNext(element);  
 }  
  
 public void insertAfter(Element element){  
 element.setNext(this.getNext());  
 element.setPrevious(this);  
 this.setNext(element);  
 }  
  
 public void insertBefore(Element element){  
 element.setNext(this);  
 element.setPrevious(this.getPrevious());  
 this.getPrevious().setNext(element);  
 this.setPrevious(element);  
 }  
  
 public void remove(){  
 this.getNext().setPrevious(this.getPrevious());  
 this.getPrevious().setNext(this.getNext());  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if(obj == null) return false;  
  
 if(obj.getClass() != this.getClass()) return false;  
  
 Element e = (Element)obj;  
  
 return this.getValue().equals(e.getValue());  
 }  
  
 @Override  
 public String toString() {  
 return value.toString();  
 }  
 }  
  
 private Element head;  
  
 public TwoWayLinkedList(){  
 head = null;  
 }  
  
 private Element getElement(int index){  
 Element element = head;  
  
 int counter = 0;  
  
 while(element.getNext() != null && counter < index){  
 element = element.getNext();  
 counter++;  
 }  
  
 if(index != counter){  
 throw new ArrayIndexOutOfBoundsException();  
 }  
  
 return element;  
 }  
  
 private Element getElement(Element e){  
 Element element = head;  
  
 while(element != null && !element.value.equals(e.value)){  
 element = element.getNext();  
 }  
  
 if(element.value.equals(e.value)){  
 return element;  
 }  
  
 return null;  
 }  
  
 public boolean isEmpty(){  
 return head == null;  
 }  
  
 public void clear() {  
 head = null;  
 }  
  
 public Element get(int index){  
 return getElement(index);  
 }  
  
 public E set(int index, E value){  
 Element element = getElement(index);  
  
 if(element != null){  
 E retValue = element.getValue();  
 element.setValue(value);  
  
 return retValue;  
 }  
  
 return null;  
 }  
  
 public boolean add(E value){  
 Element newElement = new Element(value);  
  
 if(head == null){  
 head = newElement;  
 return true;  
 }  
  
 Element current = head;  
 while(current.getNext() != null){  
 current = current.getNext();  
 }  
  
 current.insertAfter(newElement);  
  
 return true;  
 }  
  
 public boolean add(int index, E value){  
 Element newElement = new Element(value);  
  
 if(index == 0){  
 head.insertAfter(newElement);  
 } else{  
 Element element = getElement(index - 1);  
 element.insertAfter(newElement);  
 }  
  
 return true;  
 }  
  
 public int indexOf(E value){  
 Element element = head;  
  
 int counter = 0;  
 while(element != null && !element.getValue().equals(value)){  
 element = element.getNext();  
 counter++;  
 }  
  
 if(element != null && element.getValue().equals(value)){  
 return counter;  
 }  
  
 return -1;  
 }  
  
 public int size(){  
 Element element = head;  
  
 int counter = 0;  
 while(element != null){  
 counter++;  
 element = element.getNext();  
 }  
  
 return counter;  
 }  
  
 public boolean remove(E value) {  
 Element elem = getElement(new Element(value));  
  
 if(elem == null) return false;  
  
 elem.remove();  
  
 return true;  
 }  
  
 public void printList(){  
 Element element = head;  
 StringBuffer buffer = new StringBuffer();  
  
 while(element != null){  
 buffer.append(element.getValue() + ", ");  
 element = element.getNext();  
 }  
  
 System.*out*.println(buffer.toString());  
 }  
}

Wyniki działania programu:

Is empty: true

List 1:

150, 29, 9312, 253, 350,

List 2:

9999, 9559, 9939, 9534, 9834,

After merging:

9999, 9559, 9939, 9534, 9834,

List 3:

150, 29, 556, 253, 350,

List 4:

9999, 9559, 9939, 9534, 9834,

After merging:

150, 29, 556, 9999, 9559, 9939, 9534, 9834, 253, 350,