Assignment 6

1\

public Node<E> findMiddleNode(DoublyLinkedList<E> list) {  
 Node<E> slow = list.getHeader().getNext();  
 Node<E> fast = list.getHeader().getNext();  
  
 while (fast != list.getTrailer() && fast.getNext() != list.getTrailer()) {  
 fast = fast.getNext().getNext();  
 slow = slow.getNext();  
 }  
  
 return slow;  
 }

2\

public int size() {  
 int count = 0;  
 Node<E> current = header.getNext();  
  
 while (current != trailer) {  
 count++;  
 current = current.getNext();  
 }  
  
 return count;  
 }

3\

public boolean equals(Object obj) {  
 if (this == obj) {  
 return true;   
 }  
  
 if (!(obj instanceof DoublyLinkedList)) {  
 return false;   
 }  
 DoublyLinkedList<?> other = (DoublyLinkedList<?>) obj;  
  
 if (this.size() != other.size()) {  
 return false;   
 }  
  
 Node<E> currentThis = header.getNext();  
 Node<?> currentOther = other.getHeader().getNext();  
  
 while (currentThis != trailer) {  
 if (!currentThis.getElement().equals(currentOther.getElement())) {  
 return false;   
 }  
 currentThis = currentThis.getNext();  
 currentOther = currentOther.getNext();  
 }  
  
 return true;   
 }

5\

private static class Node<E> {  
 private E element;  
 private Node<E> next;  
 private Node<E> previous;  
  
 public Node(E element, Node<E> next, Node<E> previous) {  
 this.element = element;  
 this.next = next;  
 this.previous = previous;  
 }  
  
 public E getElement() {  
 return element;  
 }  
  
 public Node<E> getNext() {  
 return next;  
 }  
  
 public void setNext(Node<E> next) {  
 this.next = next;  
 }  
  
 public Node<E> getPrevious() {  
 return previous;  
 }  
  
 public void setPrevious(Node<E> previous) {  
 this.previous = previous;  
 }  
 }  
}

6\

public class CircularDoublyLinkedList<E> {  
 private Node<E> head;  
 private int size;  
  
 public CircularDoublyLinkedList() {  
 head = null;  
 size = 0;  
 }  
  
 public void addFirst(E element) {  
 Node<E> newNode = new Node<>(element);  
  
 if (isEmpty()) {  
 newNode.setNext(newNode);  
 newNode.setPrevious(newNode);  
 } else {  
 newNode.setNext(head);  
 newNode.setPrevious(head.getPrevious());  
 head.getPrevious().setNext(newNode);  
 head.setPrevious(newNode);  
 }  
  
 head = newNode;  
 size++;  
 }  
  
 public void addLast(E element) {  
 if (isEmpty()) {  
 addFirst(element);  
 } else {  
 Node<E> newNode = new Node<>(element);  
 newNode.setNext(head);  
 newNode.setPrevious(head.getPrevious());  
 head.getPrevious().setNext(newNode);  
 head.setPrevious(newNode);  
 size++;  
 }  
 }  
  
 public E removeFirst() {  
 if (isEmpty()) {  
 throw new NoSuchElementException("List is empty");  
 }  
  
 E removedElement = head.getElement();  
  
 if (size == 1) {  
 head = null;  
 } else {  
 Node<E> tail = head.getPrevious();  
 head = head.getNext();  
 tail.setNext(head);  
 head.setPrevious(tail);  
 }  
  
 size--;  
 return removedElement;  
 }  
  
 public E getFirst() {  
 if (isEmpty()) {  
 throw new NoSuchElementException("List is empty");  
 }  
  
 return head.getElement();  
 }  
  
 public int size() {  
 return size;  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 public void rotate() {  
 if (size > 1) {  
 head = head.getNext();  
 }  
 }  
  
 public void rotateBackward() {  
 if (size > 1) {  
 head = head.getPrevious();  
 }  
 }  
  
 // Private Node class  
 private static class Node<E> {  
 private E element;  
 private Node<E> next;  
 private Node<E> previous;  
  
 public Node(E element) {  
 this.element = element;  
 this.next = null;  
 this.previous = null;  
 }  
  
 public E getElement() {  
 return element;  
 }  
  
 public Node<E> getNext() {  
 return next;  
 }  
  
 public void setNext(Node<E> next) {  
 this.next = next;  
 }  
  
 public Node<E> getPrevious() {  
 return previous;  
 }  
  
 public void setPrevious(Node<E> previous) {  
 this.previous = previous;  
 }  
 }  
}

7\

public class DoublyLinkedList<E> implements Cloneable {  
 private Node<E> header;  
 private Node<E> trailer;  
 private int size;  
  
 public DoublyLinkedList() {  
 header = new Node<>(null, null, null);  
 trailer = new Node<>(null, null, null);  
 header.setNext(trailer);  
 trailer.setPrevious(header);  
 size = 0;  
 }  
  
 public void addFirst(E element) {  
 Node<E> newNode = new Node<>(element, header.getNext(), header);  
 header.getNext().setPrevious(newNode);  
 header.setNext(newNode);  
 size++;  
 }  
  
 public void addLast(E element) {  
 Node<E> newNode = new Node<>(element, trailer, trailer.getPrevious());  
 trailer.getPrevious().setNext(newNode);  
 trailer.setPrevious(newNode);  
 size++;  
 }  
  
 public E removeFirst() {  
 if (isEmpty()) {  
 throw new NoSuchElementException("List is empty");  
 }  
  
 Node<E> firstNode = header.getNext();  
 E removedElement = firstNode.getElement();  
  
 header.setNext(firstNode.getNext());  
 firstNode.getNext().setPrevious(header);  
  
 firstNode.setNext(null);  
 firstNode.setPrevious(null);  
  
 size--;  
  
 return removedElement;  
 }  
  
 public E getFirst() {  
 if (isEmpty()) {  
 throw new NoSuchElementException("List is empty");  
 }  
  
 return header.getNext().getElement();  
 }  
  
 public int size() {  
 return size;  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 public DoublyLinkedList<E> clone() {  
 DoublyLinkedList<E> newList = new DoublyLinkedList<>();  
  
 Node<E> currentNode = header.getNext();  
 while (currentNode != trailer) {  
 newList.addLast(currentNode.getElement());  
 currentNode = currentNode.getNext();  
 }  
  
 return newList;  
 }  
  
 // Other methods...  
  
 private static class Node<E> {  
 private E element;  
 private Node<E> next;  
 private Node<E> previous;  
  
 public Node(E element, Node<E> next, Node<E> previous) {  
 this.element = element;  
 this.next = next;  
 this.previous = previous;  
 }  
  
 public E getElement() {  
 return element;  
 }  
  
 public Node<E> getNext() {  
 return next;  
 }  
  
 public void setNext(Node<E> next) {  
 this.next = next;  
 }  
  
 public Node<E> getPrevious() {  
 return previous;  
 }  
  
 public void setPrevious(Node<E> previous) {  
 this.previous = previous;  
 }  
 }  
}