Problem 1  
A generic class List is to be programmed as a doubly linked list that begins and ends with a sentinel node. The sentinel is created in the constructor and can never be removed. Assume a standard implementation of a class DNode with private instance variables called data, next, prev that are accessed by getter and setter methods only.

The skeleton of code for the class List follows:

```java
public class List<T> {
    private DNode<T> sentinel;
    private int size;
    public List()
        // initializes the list: to be coded as (a)
    public void insertFirst(T x)
        // adds x at the front of the list, leaving other elements in place: coded as (b)
    public T removeLast() throws Exception
        // removes the last element from the list, returning its data: to be coded as (c)
}
```

(a) **Implement the constructor.**

**Answer:**

```java
public List() {
    size = 0;
    sentinel = new DNode<T>();
    sentinel.setNext(sentinel);
    sentinel.setPrev(sentinel);
}
```

(b) **Implement insertFirst.**

**Answer:**

```java
public void insertFirst(T x) {
    DNode<T> newNode = new DNode<T>(x, sentinel, sentinel.getNext());
    sentinel.getNext().setPrev(newNode);
    sentinel.setNext(newNode);
    size++;
}
```

(c) **Implement removeLast.**

**Answer:**

```java
public T removeLast() throws Exception {
    if (size == 0) throw new Exception("List Empty");
    size--;
    DNode<T> last = sentinel.getPrev();
    DNode<T> newLast = last.getPrev();
    newLast.setNext(sentinel);
    sentinel.setPrev(newLast);
    return last.getData();
}
```

Problem 2  
Suppose that a doubly linked list is implemented as a class DoublyLinkedList that uses sentinel doubly linked nodes header and trailer and no other instance variables. Write a method of the class called removeMiddle...
that removes either the middle node from a list of odd length, or the middle two nodes from a list of even length. The method should throw an exception if the required nodes do not exist. Give a $O$-estimate for the run time of your method in terms of the number $n$ of elements in the list.

**Answer:**

```java
public void removeMiddle() {
    if (header.getNext() == trailer)
        throw new RuntimeException("Empty");
    removeRecursive(header, trailer);
}

private void removeRecursive(DNode h, DNode t) {
    if (h == t || h.getNext() == t) {
        DNode downOne = h.getPrev();
        DNode upOne = t.getNext();
        downOne.setNext(upOne);
        upOne.setPrev(downOne);
    } else removeRecursive(h.getNext(), t.getPrev());
}
```

The run time of this method is $O(n)$.

**Problem 3** Consider the following partial implementation of a circular list of singly linked nodes.

```java
public class CircularList {
    private Node cursor;
    public CircularList() { cursor = null; }
    public boolean isEmpty() {return cursor == null;}
    public void advance() { cursor = cursor.getNext(); }
    public void addAfter(Object d) { CODE OMITTED TO SAVE SPACE }
    public void addBefore(Object d) {
        addAfter(d);
        swapData(cursor, cursor.getNext());
        cursor = cursor.getNext();
    }
    private void swapData(Node n, Node m) { // helper method for addBefore and remove
        Object temp = n.getData();
        n.setData(m.getData()); m.setData(temp); }
    public Object remove() { CODE OMITTED HERE }
    public String toString() { CODE OMITTED HERE }
}
```

Supply an implementation for the missing method `toString`. The output from your method should match the following format (which indicates a circular list with size 3, stored data A, B, C and the cursor at item A):

A -> B -> C ->

**Answer:**

```java
public String toString() {
    if (cursor == null) return "";
    String ans = cursor.getData() + " ->";
    Node n = cursor.getNext();
    while (n != cursor) {
        ans += n.getData();
        ans += " ->";
    }
}
```
n = n.getNext();
}
return ans;
}

(Extra credit) Write an implementation of the method remove. (Hint: Use the swapData() method, and apply the trick used in addBefore().)

Answer:

public Object remove() {
    if (cursor == null) throw new RuntimeException("Empty");
    Object answer = cursor.getData();
    if (cursor.getNext() == cursor) {
        cursor = null;
        return answer;
    }
    swapData(cursor, cursor.getNext());
    cursor.setNext(cursor.getNext().getNext());
    return answer;
}