

Instructor: Alex Ryba

These problems were given on exams for this course. Some older problems did not make use of generics in Java, but generic implementations are now required in this course.

**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:

```
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.**

(b) **Implement insertFirst.**

(c) **Implement removeLast.**

**Problem 2** Suppose that a doubly linked list is implemented as a generic class *DoublyLinkedList*<T> that uses an instance variable *size* and 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 node(s) 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.

**Problem 3** Consider the following partial implementation of a circular list of singly linked nodes. The nodes of the list are arranged into a big circle. One special node is marked as the cursor. All changes to the list are made at the cursor node. This cursor node can be moved forward through the list with the advance method to allow for all data to be accessed.

```
public class CircularList<T> {
    private Node<T> cursor;
    public CircularList() { cursor = null; }
    public boolean isEmpty() {return cursor == null;}
    public void advance() { cursor = cursor.getNext(); }
    public void addAfter(T d) { CODE OMITTED TO SAVE SPACE }
    public void addBefore(T d) {
        addAfter(d);
        swapData(cursor, cursor.getNext());
        cursor = cursor.getNext();
    }
    private void swapData(Node<T> n, Node<T> m) { // helper method for addBefore and remove
        T temp = n.getData();
        n.setData(m.getData()); m.setData(temp); }
    public T 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, storing data A, B, C and with the cursor positioned at item A):

A -> B -> C ->

(Extra credit) Write an implementation of the method *remove*. (Hint: Use the trick applied in the *addBefore* method. First apply the *swapData()* method to switch the data at the cursor and its follower node. Then remove the follower node.)

**Problem 4** Suppose that a doubly linked list is made of *DNodes* implemented with the following instance variables and constructor:

```
public class DNode<T> {
    private T data;
    private DNode<T> prev, next;
    public DNode(T d, DNode<T> p, DNode<T> n) { data = d; next = n; prev = p; }
    // standard getter and setter methods for all instance variables omitted
}
```

A doubly linked list class is implemented to use sentinels called *header* and *trailer* using the following instance variables and constructor:

```
public class DList<T> {
    private DNode<T> header, trailer;
    private int size;
    public DList() {
        size = 0;
        header = new DNode<T>(null, null, null);
        trailer = new DNode<T>(null, header, null);
        header.setNext(trailer);
    }
}
```

Write a method with the following title line

```
public T removeBefore(DNode<T> n) throws Exception
```

It should modify the list by removing the node immediately before the *DNode* *n*. It should return the data that was contained in the removed node. The method should not make any loop through the nodes of the list. You should throw an appropriate exception in case the removal is not allowed or is impossible.