Lösningsförslag

Länkade listor

U 1.
```
SingleLinkedList<Integer> list = new SingleLinkedList<Integer>();
list.addFirst(3);
list.addFirst(2);
list.addFirst(1);
```

U 2. a)
```
/** Returns the first element in this list.
    Throws NoSuchElementException if this list is empty. */
public E getFirst() {
    if (first == null) {
        throw new NoSuchElementException();
    }
    return first.element;
}
```

b)
```
/** Returns the last element from this list.
    Throws NoSuchElementException if this list is empty. */
public E getLast() {
    if (first == null) {
        throw new NoSuchElementException();
    }
    ListNode<E> n = first;
    while (n.next != null) {
        n = n.next;
    }
    return n.element;
}
```

U 3.
```
/** Returns true if this collection contains the specified element. */
public boolean contains(Object x) {
    ListNode<E> n = first;
    while (n != null) {
        if (n.element.equals(x)) {
            return true;
        }
        n = n.next;
    }
    return false;
}
```

U 4.
```
/** Removes the first occurrence of the specified element from this list,
    if it is present. If this list does not contain the element, it is unchanged.
    Returns true if this list contained the specified element (or equivalently,
    if this list changed as a result of the call). */
public boolean remove(Object e)  {
    if (first == null) {
        return false;
    }
    if (first.element.equals(e)) {
        first = first.next;
```

```
            return true;
        }
        ListNode<E> pre = first;
        ListNode<E> n = first.next;
        while (n != null) {
            if (n.element.equals(e)) {
                pre.next = n.next;
                return true;
            }
            pre = n;
            n = n.next;
        }
        return false;
    }
```

U 5.

```
public class Dequeue<E> {
    private Node<E> first;  // refererence to the first element
    private Node<E> last;   // refererence to the last element

    /** Creates an empty dequeue. */
    public Dequeue() {
        first = last = null;
    }

    /** Inserts the specified element at the beginning of this dequeue. */
    public void addFirst(E x) {
        Node<E> n = new Node<E>(x);
        if (first == null) {
            first = last = n;
        } else {
            n.next = first;
            first = n;
        }
    }

    /** Inserts the specified element at the end of this dequeue. */
    public void addLast(E x) {
        Node<E> n = new Node<E>(x);
        if (first == null) {
            first = last = n;
        } else {
            last.next = n;
            last = n;
        }
    }

    /** Removes and returns the first element in this dequeue.
        Returns null if this dequeue is empty. */
    public E removeFirst() {
        if (first == null) {
            return null;
        } else {
            E temp = first.element;
            first = first.next;
            if (first == null) {
                last = null;
            }
            return temp;
        }
```

```
    }

    /** Removes and returns the last element in this dequeue.
        Returns null if this dequeue is empty. */
    public E removeLast() {
        if (first == null) {
            return null;
        } else {
            E temp = last.element;
            if (first == last) {
                first = last = null;
            } else {
                Node<E> p = first;
                while (p.next !=  last) {
                    p = p.next;
                }
                last = p;
                p.next = null;
            }
            return temp;
        }
    }

    private static class Node<E> {... given i uppgiftsformuleringen...}
}
```

U 6. a) En statisk klass deklarerad i en klass C har inte tillgång till typparametern i den omgivande klassen. Den har inte heller direkt tillgång till icke-statiska attribut eller icke-statiska metoder i C. Däremot kan den, komma åt statiska attribut och statiska metoder i C .

En inre klass har tillgång till allt i den omgivande klassen (även det som deklarerats `private`).

Nodklassen har i detta fall deklarerats statisk eftersom den inte behöver tillgång till någonting i den omgivande klassen.

När det är lämpligt att deklarera en klass inuti en annan finns det alltså två fall:

- Man har ett behov av att använda attribut eller metoder eller modifiera attribut i den omgivande klassen. Då kan den nästlade klassen inte vara `static`
- Man behöver inte accessa någonting i den omgivande klassen. Då kan den nästlade klassen deklareras `static`. Det är inte fel att även i dessa fall deklarera den icke-statisk. En icke-statisk nästlad klass (inre klass) har emellertid alltid en implicit referens till det objekt av den omgivande klassen som den hör ihop med. Dessa referenser är onödiga om man inte tänker använda något från omgivningen. Därför brukar man deklarera nästlade klasser som statiska när det är möjligt.

b) Ja det går bra. Se dock punkt 2 i föregående deluppgift.

Om vi låter nodklassen bli en inre klass behöver den inte längre vara generisk. Vi kan i så fall deklarera den på följande sätt:

```
private class Node {
    E element;
    ...
}
```

Det `E` som används i deklarationen av attributet `element` binds nu till typparametern `E` i den omgivande klassen `Dequeue`.