A deque, or a double-ended queue, is a linear data structure in which elements can be added, inspected, and removed from either end of the structure, but not from the middle. Thus the deque is a generalization of both stacks (in which only the first element can be accessed) and queues (in which the first element can be removed, but insertions must happen at the end of the queue).

Write a class `DequeFromList` as a subclass of the class `ListReferenceBased` we discussed in class, and which implements the following interface:

```java
public interface DequeInterface{
    public boolean isEmpty();
    public Object peekFirst();
    public Object removeFirst();
    public void addFirst(Object o);
    public Object peekLast();
    public Object removeLast();
    public void addLast(Object o);
    public void removeAll();
}
```
2. HOW TO DO IT

Use the StackFromList discussed in class as a guide. 
Fill in the sections of the code indicated below:

```java
public class DequeFromList extends ListReferenceBased
    implements DequeInterface{
    public DequeFromList(){
        super();
    }
    public Object peekFirst(){
        return get(0);
    }
    public Object removeFirst(){
        Object ret = get(0);
        remove(0);
        return ret;
    }
    // fill in other methods
}
```

To access the tail of the list, you should use the `size` method of the `ListInterface`, in order to gain access to the index at the end.

You will need to copy to your current directory the files `ListReferenceBased.java`, `ListInterface.java` and `Node.java` (found in the `linkedList` directory in the programs discussed in class).

3. DRIVER CLASS

Your program should work with the following driver class:

```java
public class Driver{
    public static void main (String[] args){
        DequeFromList d = new DequeFromList();
        d.addFirst("apple");
        d.removeLast();
        d.addLast("peach");
        d.addFirst("cherry");
        d.addLast("plum");
        String s = (String) d.peekFirst();
        System.out.println(d.removeLast());
        d.addLast("melon");
        d.addFirst("grape");
        d.addLast(s);
    }
}
```
while (! d.isEmpty())
    System.out.println(d.removeFirst());
}
}

4. SAMPLE OUTPUT

plum
grape
cherry
peach
melon
cherry

5. DUE DATE

Monday, April 19, 2010.

6. HOW TO TURN IT IN

Email your file DequeFromList.java to loftin@rutgers.edu as an attachment. Be sure to put your name in a comment line at the beginning of your file.

7. BONUS

Write a class DequeReferenceBased which, instead of inheriting the methods from ListReferenceBased, implements DequeInterface from scratch. You should use a doubly-linked list (so that each node has references to the next node and the precede node), equipped with both head and tail references, as the basic data structure. Your class should implement Java generics and throw a DequeException if a peek or remove is attempted on an empty deque. You’ll need to pay special attention to updating the references in the cases where the queue is close to empty.

You will also need to rewrite the Node class to a DoubleNode class.

The DequeException class is as follows:

```java
public class DequeException extends RuntimeException{
    public DequeException (String s){
        super(s);
    }
}
```
To turn in the bonus, send an email containing your files `DequeReferenceBased` and `DoubleNode` as attachments. (You should also turn in the program `DequeFromList` separately.)

### 8. Driver class for bonus

```java
public class BonusDriver{
    public static void main (String[] args){
        DequeReferenceBased<Integer> d = new DequeReferenceBased<Integer>();
        d.addLast(5);
        d.addFirst(13);
        for (int i=0; i<10; i++){
            d.addFirst(i);
            d.addLast(3*i);
        }
        Integer a = d.removeLast();
        Integer b = d.removeLast();
        try{
            for (int j=0; j<25; j++)
                System.out.print(d.removeFirst() + " ");
        }
        catch (DequeException c){
            c.printStackTrace();
        }
        d.addLast(a);
        d.addFirst(b);
        while (! d.isEmpty())
            System.out.println(d.removeLast());
    }
}
```