CONCEPT EXERCISES

7.1 In the SinglyLinkedList class, define the following method without using an iterator.

```
/**
 *  Finds the element at a specified position in this LinkedList object.
 *  The worstTime(n) is O(n).
 *
 *  @param index – the position of the element to be returned.
 *
 *  @return the element at position index.
 *
 *  @throws IndexOutOfBoundsException – if index is less than 0 or greater
 *                  than or equal to size().
 */
public E get (int index)
{
        if (index < 0 || index >= size( ))
                throw new IndexOutOfBoundsException( );
        Entry<E> current = head;
        while (index > 0)
        {
                current = current.next;
                index--;
        } // while
        return current.element;
} // method get
```

7.2 Re-do Concept Exercise 7.1 by using an iterator.

```
public E get (int index)
{
        if (index < 0 || index >= size( ))
                throw new IndexOutOfBoundsException( );
        Iterator<E> itr = iterator();
        while (index > 0)
        {
                itr.next();
                index--;
        } // while
        return itr.next();
} // method get
```

7.3 Suppose we added each of the following methods to the ArrayList class:

```
public boolean addFirst (E element)
public boolean addLast (E element)
public E getFirst()
public E getLast()
public E removeFirst()
public E removeLast()
```

Estimate worstTime(*n*) for each method.

addFirst: linear in *n*
addLast: linear in *n* (because of the possibility of resizing)
getFirst: constant
getLast: constant
removeFirst: linear in *n*
removeLast: constant

7.7 Explain how to remove "Don" from the LinkedList object in Figure 7.18. Explain why, for the definition of the method remove (E element), worstTime(*n*) is linear in *n*?

Starting at header, search through the LinkedList object names to find a reference to an entry, e, whose element is "Don". Then set e.next.previous to e.previous, set e.previous.next to e.next, and increment size.

The worstTime(n) is linear in *n* because of the search to find an entry whose element is the element to be removed.

PROGRAMMING EXERCISES

7.4 Rewrite the code in Exercise 7.2 with a native array. For example, you would start with:

```
char [ ] letters = new char [10];

letters [0] = 'f';
```

Test your revision with a project that includes the above code in a main method.

```
public static void main (String[ ] args)
{
        char [ ] letters = new char [10];

        letters [0] = 'f';
        letters [1] = 't';
        System.arraycopy (letters, 0, letters, 1, 2);
        letters [0] = 'e';
        System.arraycopy (letters, 1, letters, 2, 2);
```

```
                letters [1] = 'r';
                System.arraycopy (letters, 3, letters, 4, 1);
                letters [3] = 'e';
                System.arraycopy (letters, 4, letters, 5, 1);
                letters [4] = 'c';
                System.arraycopy (letters, 0, letters, 1, 6);
                letters [0] = 'p';

                for (int i = 0; i < 7; i++)
                        System.out.print (letters [i]);
        } // method main
```

7.5 Hypothesize the error in the following code:

```
LinkedList<Double> duesList = new LinkedList<Double>();

ListItr<Double> itr = duesList.listIterator();
```

Test your hypothesis with a project that includes the above code in the `main` method.

Error: "cannot find class `ListItr`" because `ListItr` is not a public class. The public interface `ListIterator` should be used instead.

7.10 Define and test the following method:

```
/**
 *   Removes the first and last 4-letter words from a given LinkedList<String>
               object.
 *  Each word consists of letters only.
 *  The worstTime(n) is O(n).
 *
 *  @param list – the LinkedList<String> object.
 *
 *  @throws NullPointerException – if list is null.
 * @throws NoSuchElementException - if list is not null, but list has no 4-letter
 *                                   or only one 4-letter word.
 *
 */
public static void bleep (LinkedList<String> list)
```

```
import org.junit.*;
import static org.junit.Assert.*;
import org.junit.runner.Result;
import static org.junit.runner.JUnitCore.runClasses;
import java.util.*;

public class BleepTest
{
    public static void main(String[ ] args)
    {
        Result result = runClasses (BleepTest.class);
        System.out.println ("Tests run = " + result.getRunCount() +
                            "\nTests failed = " + result.getFailures());
    } // method main

    protected LinkedList<String> list;

    @Before
    public void runBeforeEachTest()
    {
        list = new LinkedList<String>();
    } // method runBeforeEachTest

    @Test (expected = NullPointerException.class)
    public void nullListTest()
    {
         list = null;
         bleep (list);
    } // methoe nullListTest

    @Test (expected = NoSuchElementException.class)
    public void noBleepsTest()
    {
        list.add ("wow");
        bleep (list);
    } // noBleepsTest

    @Test (expected = NoSuchElementException.class)
    public void oneBleepTest()
    {
        list.add ("wow");
        list.add ("help");
        bleep (list);
    } // oneBleepTest

    @Test
    public void twoBleepsTest1()
    {
        list.add ("help");
        list.add ("flop");
        bleep (list);
        assertEquals ("[]", list.toString());
    } // method twoBleepsTest1

    @Test
    public void twoBleepsTest2()
    {
        list.add ("wow");
        list.add ("help");
        list.add ("yes");
        list.add ("flop");
        list.add ("never");
        bleep (list);
        assertEquals ("[wow, yes, never]", list.toString());
    } // method twoBleepsTest2

    @Test
    public void twoBleepsTest3()
    {
        list.add ("help");
        list.add ("ecru");
        list.add ("flop");
        bleep (list);
        assertEquals ("[ecru]", list.toString());
    } // method twoBleepsTest3

    @Test
    public void twoBleepsTest4()
    {
        list.add ("wow");
        list.add ("maybe");
        list.add ("true");
        list.add ("several");
        list.add ("help");
        list.add ("flop");
        list.add ("some");
        list.add ("several");
        bleep (list);
        assertEquals ("[wow, maybe, several, help, flop, several]", list.toString());
    } // method twoBleepsTest4

    public static void bleep (LinkedList<String> list)
    {
        ListIterator<String> itr = list.listIterator();

        String s;
```

```
        int listSize = list.size();

        while (itr.hasNext())
        {
            s = itr.next();
            if (s.length() == 4)
            {
                itr.remove();
                break;
            } // 4-letter word
        } // while itr.hasNext()
        itr = list.listIterator (list.size());
        while (itr.hasPrevious())
        {
            s = itr.previous();
            if (s.length() == 4)
            {
                itr.remove();
                break;
            } // 4-letter word
        } // while itr.hasPrevious()
        if (listSize != list.size() + 2)
            throw new NoSuchElementException();
    } // method bleep

} // class BleepTest
```