Linked Lists Assignment

Part I:

Create a program that prompts the user for a string. Create a Linked List, store the string in the list, and again prompt the user for another string. Continue to add nodes into your list for each string that the user provides. When the user types “QUIT”, stop collecting strings and print the contents of your linked list. Each node on the list should be printed on a separate line. Exit the program.

Part 2:

Update your program, so that your linked list can allow nodes to be deleted. After the list has been populated from Part I, prompt the user to enter a string to be removed from the list. You will have to compare each node to find the one to delete. When the user types “QUIT”, stop deleting strings and print the contents of your linked list. Each node on the list should be printed on a separate line. Exit the program.

Some advice: Make sure you understand how to:

- <u>create</u> a list
- <u>add</u> an element into your list,
- <u>read</u> an element in your list,
- <u>delete</u> an element from your list.

Solution1:

```
import java.util.LinkedList;
import java.util.*;
import java.util.Scanner;
import java.lang.String;

public class LinkedListProject {

        public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

          //create LinkedList object
          LinkedList lList = new LinkedList();

          //add nodes to LinkedList
        //... Read a list of words.
        String word = in.next();
        while ( !word.equalsIgnoreCase("quit") ) {
            lList.add(word);
            System.out.println ("saved " + word );
            word = in.next();
        }
        System.out.println ("received " + word );

        System.out.println("LinkedList contains : " + lList);


        /* Insert a node into the list at index 3. */
        lList.add(3,"a");
        System.out.println("LinkedList contains : " + lList);

        /* Remove a node from the list at index 5. */
        lList.remove(5);
        System.out.println("LinkedList contains : " + lList);

        }
      }
```

(untested) Solution2:

```
import java.util.Scanner;

public class SimpleSingleLinkedList {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        Elem front = null;    // First element of list.
        Elem back  = null;    // Last element of list.

        //... Read a list of words.
        while (in.hasNext()) {
            String word = in.next();
            Elem e = new Elem();      // Create a new list element.
            e.data = word;            // Set the data field.

            //... Two cases must be handled differently
            if (front == null) {
                  //... When the list is empty,
                  // we have to set the front pointer.
                front = e;
                  // Back element will be set below.
            } else {
                  //... When we already have elements,
                  // we need to link to it.
                back.next = e;      // Link last elem to new element.
            }
            back = e;          // Update back to link to new element.
        }

        //... While loop to print list in forward order.
        System.out.println("*** Print words in order of entry");
        Elem curr = front;
        while (curr != null) {
            System.out.println(curr.data);
            curr = curr.next;
        }

        System.out.println("*** Print words in order of entry");
        for (Elem e = front; e != null; e = e.next) {
            System.out.println(e.data);
        }

        //... Printing list in backward order is an interesting
exercise.
        //    But too much for here.
    }
}

class Elem {
    public Elem next;    // Link to next element in the list.
    public String data;  // Reference to the data.
}
```