```java
public class UnorderedLinkedList extends LinkedListClass
{

    public UnorderedLinkedList()
    {
        super();
    }

    public UnorderedLinkedList(UnorderedLinkedList otherList)
    {
        super(otherList);
    }

        //Method to determine whether searchItem is in
        //the list.
        //Postcondition: Returns true if searchItem is found in the list; false otherwise.
    public boolean search(DataElement searchItem)
    {
        LinkedListNode current; //variable to traverse the list
        boolean found;
        current = first;    //set current to point to the first node in the list
        found = false;    //set found to false

        while(current != null && !found) //search the list
            if(current.info.equals(searchItem))  //item is found
                found = true;
            else
               current = current.link; //make current point to the next node
        return found;
    }

        //Method to delete deleteItem from the list.
        //Postcondition: If found, the node containing
        //               deleteItem is deleted from the
        //               list. Also first points to the first
        //               node, last points to the last
        //               node of the updated list, and count
        //               is decremented by 1.
    public void deleteNode(DataElement deleteItem)
    {
        LinkedListNode current; //variable to traverse the list
        LinkedListNode trailCurrent; //variable just before current
        boolean found;

        if(first == null)    //Case 1; the list is empty
           System.err.println("Cannot delete from an empty list.");
        else
        {
           if(first.info.equals(deleteItem)) //Case 2
           {
              first = first.link;
              if(first == null)    //the list had only one node
                 last = null;
              count--;
           }
           else  //search the list for the node with the given info
           {
              found = false;
              trailCurrent = first;   //set trailCurrent to point to the first node
              current = first.link;   //set current to point to the second node
              while(current != null && !found)
              {
                  if(current.info.equals(deleteItem))
                     found = true;
                  else
                  {
                      trailCurrent = current;
                      current = current.link;
                  }
              }//end while
```

```
71
72              if(found) //Case 3; if found, delete the node
73              {
74                  count--;
75                  trailCurrent.link = current.link;
76                  if(last == current)   //node to be deleted was the last node
77                     last = trailCurrent;  //update the value of last
78              }
79              else
80                 System.out.println("Item to be deleted is not in the list.");
81           }//end else
82       }//end else
83   }//end deleteNode
84
85   public void deleteAll(DataElement deleteItem)
86   {
87       LinkedListNode current; //variable to traverse the list
88       LinkedListNode trailCurrent = null; //variable just before current
89       if(first == null)    //Case 1; list is empty.
90           System.err.println("Can not delete from an empty list.");
91       else
92       {
93           current = first;
94           while(current != null)
95           {
96               if(current.info.equals(deleteItem))
97               {
98                   if(current == first)
99                   {
100                      count--;                    // fixed by instructor
101                      first = first.link;
102                      current = first;
103                      if(first == null) last = null;
104                  }
105                  else
106                  {
107                      count--;                    // fixed by instructor
108                      trailCurrent.link = current.link;
109                      if(current == last) last = trailCurrent;
110                      current = trailCurrent.link;
111                  }
112              }
113              else
114              {
115                  trailCurrent = current;
116                  current = current.link;
117              }
118          } //end while
119      }
120  } //end deleteAll
121
122  public void deleteSmallest()
123  {
124      LinkedListNode current; //variable to traverse the list
125      LinkedListNode trailCurrent; //variable just before current
126      LinkedListNode small;
127      LinkedListNode trailSmall = null;
128
129      if(first == null)
130          System.err.println("Can not delete from an empty list.");
131      else
132          if(first.link == null)
133          {
134              count--;                            // fixed by instructor
135              first = null;
136              last = null;
137          }
138          else
139          {
140              count--;                            // fixed by instructor
```

```
141                 small = first;
142                 trailCurrent = first;
143                 current = first.link;
144                 while(current != null)
145                 {
146                     if(small.info.compareTo(current.info) > 0)
147                     {
148                         trailSmall = trailCurrent;
149                         small = current;
150                     }
151                     trailCurrent = current;
152                     current = current.link;
153                 }
154                 if(small == first) first = first.link;
155                 else
156                 {
157                     trailSmall.link = small.link;
158                     if(small == last) last = trailSmall;
159                 }
160             }
161     }//end deleteSmallest
162 }
```