

```

1  /*****
2
3      /**
4      * @author          Ehsan Kourkchi
5      * @lecture         ICS 211 November 2013
6      * @date            December 9, 2013
7      * @class type     public generic class
8      */
9  *****/
10
11 import java.io.BufferedWriter;
12 import java.io.File;
13 import java.io.FileWriter;
14 import java.io.FileOutputStream;
15 import java.io.IOException;
16 import java.io.OutputStreamWriter;
17 import java.io.Writer;
18 import java.util.Scanner;
19
20
21 public class OrderedLinkedList<E> {
22
23     private KeyedNode<E> head = null;
24     private int size=0;
25
26
27     //////////////////////////////////////
28     //////////////////////////////////////
29     //////////////////////////////////////
30
31     private class KeyedNode<E> {
32
33         private String key;
34         private E item;
35         private KeyedNode<E> next;
36
37
38         private KeyedNode(String key, E value) {
39             item = value;
40             this.key = key;
41             next = null;
42         }
43
44         private KeyedNode(String key, E value, KeyedNode<E> reference) {
45             item = value;
46             this.key = key;
47             next = reference;
48         }
49     }
50
51
52     //////////////////////////////////////
53     //////////////////////////////////////
54     //////////////////////////////////////
55
56     // A no-arguments constructor public OrderedLinkedList()
57     // class to create an empty ordered linked list.
58     public OrderedLinkedList() {
59         head = null;
60         size = 0;
61     }
62
63
64     // Returns the number of elements in the linked list.
65     int size() {
66         return size;
67     }

```

```
68
69
70 ///////////////////////////////////////////////////
71 ///////////////////////////////////////////////////
72 ///////////////////////////////////////////////////
73
74 // If the list does not already contain the given key,
75 // the key and value are added to the linked list, and add returns
76 null.
77 // Otherwise, the existing object with the given key is replaced
78 // with
79 // the new object (and the new key), and add returns the old
80 // (removed) object.
81 E add(String key, E value) {
82
83     int i;
84     KeyedNode<E> nodepointer = head;
85     KeyedNode<E> nodeprevious = null;
86
87     if (nodepointer == null)
88     {
89         head = new KeyedNode<E> (key, value, head);
90         size ++;
91         return null;
92     }
93     else
94     {
95         while(nodepointer != null &&
96             key.compareToIgnoreCase(nodepointer.key) > 0)
97         {
98             nodeprevious = nodepointer;
99             nodepointer = nodepointer.next;
100         }
101         if (nodeprevious == null)
102         {
103             head = new KeyedNode<E> (key, value, nodepointer);
104             size ++;
105             if (head.next.key.compareToIgnoreCase(key) == 0)
106             {
107                 E temp = head.next.item;
108                 head.next = head.next.next;
109                 size --;
110                 return temp;
111             }
112             return null;
113         }
114         else
115         {
116             nodeprevious.next = new KeyedNode<E> (key, value,
117                 nodepointer);
118             size ++;
119             if (nodepointer != null &&
120                 nodeprevious.next.key.compareToIgnoreCase(nodepointer.key)
121                 == 0)
122             {
123                 nodeprevious.next.next = nodepointer.next;
124                 size --;
125                 return nodepointer.item;
126             }
127         }
128     }
129 }
```

```

128     return null;
129 } // The end of 'add' method
130
131
132 //////////////////////////////////////
133 //////////////////////////////////////
134 //////////////////////////////////////
135
136 // This method prints the database on the screen.
137 void printList() {
138
139     int i;
140     KeyedNode<E> nodepointer = head;
141
142     System.out.println("\n *****");
143     while(nodepointer != null)
144     {
145         System.out.printf(" %-15s Tel: %s \n" , nodepointer.key,
146             nodepointer.item);
147         nodepointer = nodepointer.next;
148     }
149
150     System.out.println(" *****");
151     System.out.println(" Size of the list: " + size );
152     System.out.println(" *****\n");
153 } // the end of printList method
154
155 //////////////////////////////////////
156 //////////////////////////////////////
157 //////////////////////////////////////
158
159 // Returns the matching object if the corresponding key is
160 // in the ordered linked list, and null otherwise.
161 E find(String key) {
162
163     KeyedNode<E> nodepointer = head;
164
165     while(nodepointer != null &&
166         key.compareToIgnoreCase(nodepointer.key) != 0)
167     {
168         nodepointer = nodepointer.next;
169     }
170
171     if (nodepointer != null)
172         return nodepointer.item;
173     else
174         return null;
175
176 }
177
178
179 //////////////////////////////////////
180 //////////////////////////////////////
181 //////////////////////////////////////
182
183 // Removes the matching object if the corresponding key
184 E remove(String key) {
185     KeyedNode<E> nodepointer = head;
186     KeyedNode<E> nodeprevious = null;
187
188
189     while(nodepointer != null &&
190         key.compareToIgnoreCase(nodepointer.key) != 0)
191     {
192         nodeprevious = nodepointer;

```

```

192         nodepointer = nodepointer.next;
193     }
194 }
195
196 if (nodepointer != null)
197 {
198     if (nodepointer == head)
199         head = nodepointer.next;
200     else
201         nodeprevious.next = nodepointer.next;
202     size--;
203     return nodepointer.item;
204 }
205 else
206     return null;
207
208 }
209
210
211
212 ///////////////////////////////////////////////////////////////////
213 ///////////////////////////////////////////////////////////////////
214 ///////////////////////////////////////////////////////////////////
215
216 // Returns the object at the given position, as long as the index
217 // is valid.
218 // Otherwise, it returns null (this is unlike the standard get,
219 // which throws an exception if the index is invalid).
220 E get(int position) {
221     KeyedNode<E> nodepointer = head;
222     int index = 1;
223
224     if (position > size) return null;
225     else
226     {
227         while(index != position)
228         {
229             System.out.println(index);
230             nodepointer = nodepointer.next;
231             index++;
232         }
233         return nodepointer.item;
234     }
235 }
236
237 }
238
239 ///////////////////////////////////////////////////////////////////
240 ///////////////////////////////////////////////////////////////////
241 ///////////////////////////////////////////////////////////////////
242
243 // Prints the entire database into a file
244 // it gets the pointer to the destination file
245 // it does not care if the file already exists. It overwrites the
246 // previously available file
247 // it's better the availability of the in the User Interface
248 int write_file(BufferedWriter bw) {
249     KeyedNode<E> nodepointer = head;
250
251     try {
252
253         bw.write("*****" + "\n");
254         bw.write("Size of the list: " + (Integer) size
255             + "\n");
256         bw.write("*****" + "\n\n");

```

```
256
257         while(nodepointer != null)
258         {
259             bw.write("Name: " + nodepointer.key+ "\n");
260             bw.write("Tel: " + nodepointer.item+ "\n\n");
261             nodepointer = nodepointer.next;
262         }
263
264
265     }
266     catch (IOException e) {
267         e.printStackTrace();
268         return 1;
269     }
270
271     return 0;
272 }
273
274
275
276 } //the end of the main class (i.e OrderedLinkedList<E>)
277
278
279 //////////////////////////////////////
280 //////////////////////////////////////
281 //////////////////////////////////////
```