```java
/*****************************************************************/

                /**
                * @author              Ehsan Kourkchi
                * @lecture             ICS 211 November 2013
                * @date                December 9, 2013
                * @class type          public generic class
                */
/*****************************************************************/

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.util.Scanner;


public class OrderedLinkedList<E> {

private KeyedNode<E> head = null;
private int size=0;


///////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////

private class KeyedNode<E> {

    private String key;
    private E item;
    private KeyedNode<E> next;


       private KeyedNode(String key, E value) {
         item = value;
         this.key = key;
         next = null;
       }

       private KeyedNode(String key, E value, KeyedNode<E> reference) {
         item = value;
         this.key = key;
         next = reference;
       }

    }

///////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////

    // A no-arguments constructor public OrderedLinkedList()
    // class to create an empty ordered linked list.
     public OrderedLinkedList() {
          head = null;
          size = 0;
     }


    // Returns the number of elements in the linked list.
    int size() {
         return size;
    }
```

```java


///////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////

    // If the list does not already contain the given key,
    // the key and value are added to the linked list, and add returns null.
    // Otherwise, the existing object with the given key is replaced with
    // the new object (and the new key), and add returns the old (removed) object.
    E add(String key, E value) {

       int i;
       KeyedNode<E> nodepointer = head;
       KeyedNode<E> nodeprevious = null;

       if (nodepointer == null)
       {
          head = new KeyedNode<E> (key, value, head);
          size ++;
          return null;
       }
       else
       {
          while(nodepointer != null && key.compareToIgnoreCase(nodepointer.key) > 0)
          {
              nodeprevious = nodepointer;
              nodepointer = nodepointer.next;
          }

          if (nodeprevious == null)
          {
              head = new KeyedNode<E> (key, value, nodepointer);
              size ++;
              if (head.next.key.compareToIgnoreCase(key) == 0)
              {
                  E temp = head.next.item;
                  head.next = head.next.next;
                  size --;
                  return temp;
              }
          return null;
          }
          else
          {
              nodeprevious.next = new KeyedNode<E> (key, value, nodepointer);
              size ++;

              if (nodepointer != null && nodeprevious.next.key.compareToIgnoreCase(nodepointer.key) == 0)
              {
                  nodeprevious.next.next = nodepointer.next;
                  size --;
                  return nodepointer.item;
              }


          }

       }

    return null;
    }  // The end of 'add' method


///////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////

    // This method prints the database on the screen.
    void printList() {

       int i;
       KeyedNode<E> nodepointer = head;

       System.out.println("\n  ************************************");
       while(nodepointer != null)
       {
          System.out.printf("  %-15s  Tel: %s \n"  , nodepointer.key,
          nodepointer.item);
          nodepointer = nodepointer.next;
       }

       System.out.println("  ************************************");
       System.out.println("  Size of the list: " + size );
       System.out.println("  ************************************\n");

    }  // the end of printList method

///////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////

    // Returns the matching object if the corresponding key is
    // in the ordered linked list, and null otherwise.
     E find(String key) {

       KeyedNode<E> nodepointer = head;

       while(nodepointer != null &&
       key.compareToIgnoreCase(nodepointer.key) != 0)
          {
              nodepointer = nodepointer.next;
          }

          if (nodepointer != null)
            return nodepointer.item;
               else
                  return null;


     }


///////////////////////////////////////////////////
///////////////////////////////////////////////////
///////////////////////////////////////////////////

      // Removes the matching object if the corresponding key
      E remove(String key) {
       KeyedNode<E> nodepointer = head;
       KeyedNode<E> nodeprevious = null;


       while(nodepointer != null &&
       key.compareToIgnoreCase(nodepointer.key) != 0)
          {
              nodeprevious = nodepointer;
              nodepointer = nodepointer.next;

          }

          if (nodepointer != null)
          {
             if (nodepointer == head)
                head = nodepointer.next;
                    else
                        nodeprevious.next = nodepointer.next;
             size--;
             return nodepointer.item;
          }
          else
            return null;


     }


//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////

    // Returns the object at the given position, as long as the index is valid.
    // Otherwise, it returns null (this is unlike the standard get,
    // which throws an exception if the index is invalid).
     E get(int position) {

       KeyedNode<E> nodepointer = head;
       int index = 1;

       if (position > size) return null;
       else
       {

           while(index != position)
          {
              System.out.println(index);
              nodepointer = nodepointer.next;
              index++;
          }
       return nodepointer.item;
       }

     }

//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////

    // Prints the entire database into a file
    // it gets the ponter to the destination file
    // it does not care if the file already exists. It overwrites the previously available file
    // it's better the availability of the in the User Interface
     int write_file(BufferedWriter bw) {
     KeyedNode<E> nodepointer = head;

     try {

                        bw.write("************************" + "\n");
                        bw.write("Size of the list: " + (Integer) size + "\n");
                        bw.write("************************" + "\n\n");

```

```

                          while(nodepointer != null)
                          {
                          bw.write("Name: " + nodepointer.key+ "\n");
                          bw.write("Tel: " + nodepointer.item+ "\n\n");
                          nodepointer = nodepointer.next;
                          }


             }
             catch (IOException e) {
               e.printStackTrace();
               return 1;
             }

         return 0;
         }



} //the end of the main class (i.e OrderedLinkedList<E>)


////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
```