# ***App. Java***

```
package app;

import java.time.Duration;
import java.time.Instant;
import java.util.List;

/**
 *
 * <p>
 * <strong><em>Application Name: </em></strong>Class_Projects-Linked_List
 * </p>
 * <p>
 * <strong><em>Class Name: </em></strong>App
 * </p>
 *
 * <p>
 * <strong><em>Application Notes: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Class Notes: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Pre-Conditions: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Post-Conditions: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Author: </em></strong>Daniel C. Landon Jr.
 * </p>
 * <p>
 * <strong><em>Instructor: </em></strong>Dr. Robert Walsh
 * </p>
 * <p>
 * <strong><em>Course: </em></strong>SP20-SE-CSCI-C202-17057
 * </p>
 * <p>
 * <strong><em>Start Date: </em></strong>04.20.2020
 * </p>
 * <p>
 * <strong><em>Due Date: </em></strong>04.23.2020
 * </p>
 *
```

```
 */
public class App {

    /**
     *
     * <p><strong><em>Description: </em></strong>entry point for application</p>
     *
     * <p><strong><em>Method Name: </em></strong>main</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
     *
     * @param args not used
     * @throws Exception catch error so program ends gracefully
     */
    public static void main(String[] args) throws Exception {

        // variables
        Instant _tStart = null;
        Instant _tEnd = null;
        Duration _tElapsed = null;

        try {

            _tStart = Instant.now();

            // variables
            LinkedList<String> _list = new LinkedList<String>();

            System.out.println();
            lineSeperator(80, '*');
            System.out.println();
            System.out.println("\tSTART TIME: " + _tStart);
            System.out.println();

            // current status of list
            lineSeperator(80, '*');
            System.out.println();
            System.out.println(">>>>>>>>>> STATE OF LIST <<<<<<<<<<");
            System.out.println();
            System.out.println("Is the list Empty: " + _list.isEmpty());
            System.out.println("Number of elements in list: " + _list.getSize());
            System.out.println();
            System.out.println("\n" + _list.show());
```

```
        System.out.println();

        // delete from list
        lineSeperator(80, '*');
        System.out.println();
        System.out.println(">>>>>>>>>>> DELETE FROM AN EMPTY LIST <<<<<<<<<<");
        System.out.println();
        _list.delete("Bob");
        System.out.println();
        System.out.println("Is the list Empty: " + _list.isEmpty());
        System.out.println("Number of elements in list: " + _list.getSize());
        System.out.println();


        // add the captains of the U.S.S. Enterprise, based on cannon, in order
        lineSeperator(80, '*');
        System.out.println();
        System.out.println(">>>>>>>>>>> ADD DATA TO LIST <<<<<<<<<<");
        System.out.println();
        _list.add("NX-01: Archer");
        _list.add("NCC-1701: April");
        _list.add("NCC-1701: Pike");
        _list.add("NCC-1701: Kirk");
        _list.add("NCC-1701: Decker");
        _list.add("NCC-1701: Spock");
        _list.add("NCC-1701-A: Kirk");
        _list.add("NCC-1701-B: Harriman");
        _list.add("NCC-1701-C: Garrett");
        _list.add("NCC-1701-D: Picard");
        _list.add("NCC-1701-D: Riker");
        _list.add("NCC-1701-D: Jellico");
        _list.add("NCC-1701-E: Picard");
        System.out.println();

        // current status of list
        lineSeperator(80, '*');
        System.out.println();
        System.out.println(">>>>>>>>>>> STATE OF LIST <<<<<<<<<<");
        System.out.println();
        System.out.println("Is the list Empty: " + _list.isEmpty());
        System.out.println("Number of elements in list: " + _list.getSize());
        System.out.println();
        System.out.println("\n" + _list.show());
        System.out.println();

        // delete from random point in list
        lineSeperator(80, '*');
        System.out.println();
        System.out.println(">>>>>>>>>>> DELETE FROM RANDOM POINT IN LIST <<<<<<<<<<");
        System.out.println();
```

```
System.out.println("Is the list Empty: " + _list.isEmpty());
System.out.println("Number of elements in list: " + _list.getSize());
System.out.println();
System.out.println("\n" + _list.show());
System.out.println();
_list.delete("NCC-1701-C: Garrett");
System.out.println();
System.out.println("Is the list Empty: " + _list.isEmpty());
System.out.println("Number of elements in list: " + _list.getSize());
System.out.println();
System.out.println("\n" + _list.show());
System.out.println();

// delete the head
lineSeperator(80, '*');
System.out.println();
System.out.println(">>>>>>>>>> DELETE THE HEAD <<<<<<<<<<");
System.out.println();
System.out.println("Is the list Empty: " + _list.isEmpty());
System.out.println("Number of elements in list: " + _list.getSize());
System.out.println();
System.out.println("\n" + _list.show());
System.out.println();
_list.delete("NX-01: Archer");
System.out.println();
System.out.println("Is the list Empty: " + _list.isEmpty());
System.out.println("Number of elements in list: " + _list.getSize());
System.out.println();
System.out.println("\n" + _list.show());
System.out.println();

// add new element to head of list and shift everything down
lineSeperator(80, '*');
System.out.println();
System.out.println(">>>>>>>>>> ADD NEW HEAD ELEMENT <<<<<<<<<<");
System.out.println();
System.out.println("Is the list Empty: " + _list.isEmpty());
System.out.println("Number of elements in list: " + _list.getSize());
System.out.println();
System.out.println("\n" + _list.show());
System.out.println();
_list.addFront("Bob's Your Uncle");
System.out.println();
System.out.println("Is the list Empty: " + _list.isEmpty());
System.out.println("Number of elements in list: " + _list.getSize());
System.out.println();
System.out.println("\n" + _list.show());
System.out.println();

// does list contain element
```

```
        lineSeperator(80, '*');
        System.out.println();
        System.out.println(">>>>>>>>>> DOES LIST CONTAIN <<<<<<<<<<");
        System.out.println();
        System.out.println("Is the list Empty: " + _list.isEmpty());
        System.out.println("Number of elements in list: " + _list.getSize());
        System.out.println();
        System.out.println("\n" + _list.show());
        System.out.println();
        String _searchValue = "NCC-1701: Pike";
        List<Object> _retVal = _list.contains(_searchValue);
        if((boolean) _retVal.get(0)) {
            // found
            System.out.println("List contains, " + _searchValue + ", at index:  " + _retVal.get(1) + ".");
        } // end if
        else {
            // not found
            System.out.println("List does not contain: " + _searchValue + ".");
        } // end if
        // System.out.println("Does list contain, " + _searchValue + ": " + _list.contains(_searchValue));
        System.out.println();

        // clear the list
        lineSeperator(80, '*');
        System.out.println();
        System.out.println(">>>>>>>>>> CLEAR THE LIST <<<<<<<<<<");
        System.out.println();
        System.out.println("Is the list Empty: " + _list.isEmpty());
        System.out.println("Number of elements in list: " + _list.getSize());
        System.out.println();
        System.out.println("\n" + _list.show());
        System.out.println();
        _list.clear();
        System.out.println();
        System.out.println("Is the list Empty: " + _list.isEmpty());
        System.out.println("Number of elements in list: " + _list.getSize());
        System.out.println();
        System.out.println("\n" + _list.show());
        System.out.println();

    } // end try
    catch (Exception e) {

        // error handeling so the program will terminate gracefully regardless

        System.out.println("***** ERROR *****\n");
        System.out.println(e.getMessage());

    } // end catch
    finally {
```

```
            lineSeperator(80, '*');
            _tEnd = Instant.now();
            _tElapsed = Duration.between(_tStart, _tEnd);
            System.out.println();
            System.out.println("\tEND TIME: " + _tEnd);
            System.out.println("\tTime for completion (milliseconds): " + _tElapsed.toMillis());
            System.out.println();

            System.out.println(">>>>>>>>>> PROGRAM TERMINATED <<<<<<<<<<\n");
            System.out.println("END OF LINE");

        } // end finally

    } // end main

    /**
    * <p><strong><em>Description: </em></strong>Displays Character N times</p>
    *
    * <p><strong><em>Method Name: </em></strong>Show</p>
    *
    * <p><strong><em>Method Notes: </em></strong>recursive display of character</p>
    *
    * <p><strong><em>Pre-Conditions: </em></strong>none</p>
    *
    * <p><strong><em>Post-Conditions: </em></strong>none</p>
    *
    * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
    * <p><strong><em>Start Date: </em></strong>03.02.2020</p>
    *
    * @param N number of times to display character
    * @param ch character to show
    */
    public static void lineSeperator(int N, char ch) {

        if(N > 1) {
            System.out.print(ch);
            lineSeperator(N - 1, ch);
        } // end if
        else { System.out.println(""); } // end else

    } // end lineSeperator

} // end App
```

# ***Node.Java***

```
package app;

public class Node<T> {

    Node<T> next;

    T element;

    static int counter = 0;

    public Node(T element) {
        this.element = element;
    }

}
```

# ***LinkedList.java***

```
package app;

import java.util.Arrays;
import java.util.List;

/**
 *
 * <p>
 * <strong><em>Application Name: </em></strong>Class_Project-Linked_List
 * </p>
 * <p>
 * <strong><em>Class Name: </em></strong>LinkedList
 * </p>
 *
 * <p>
 * <strong><em>Application Notes: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Class Notes: </em></strong>noone
 * </p>
 *
 * <p>
 * <strong><em>Pre-Conditions: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Post-Conditions: </em></strong>none
 * </p>
 *
 * <p>
 * <strong><em>Author: </em></strong>Daniel C. Landon Jr.
 * </p>
 * <p>
 * <strong><em>Instructor: </em></strong>Dr. Robert Walsh
 * </p>
 * <p>
 * <strong><em>Course: </em></strong>SP20-SE-CSCI-C202-17057
 * </p>
 * <p>
 * <strong><em>Start Date: </em></strong>04.20.2020
 * </p>
 * <p>
 * <strong><em>Due Date: </em></strong>04.23.2020
 * </p>
 *
 */
```

```
public class LinkedList<T> {

    // class properties
    private Node<T> _head;

    /**
     *
     * <p><strong><em>Description: </em></strong>manipulates list for display</p>
     *
     * <p><strong><em>Method Name: </em></strong>show</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
     *
     * @return manipulated string for display
     */
    public String show() {

        System.out.println("ENTER: show(), displays list.");

        //variables
        StringBuilder _results = new StringBuilder("");
        int _nodeSize = Node.counter;

        if(_nodeSize == 0) {

            // list is empty
            _results.append("Nothing to display, List is Empty!");

        } // end if
        else {

            // reset to head
            Node<T> _current = _head;

            _results.append("[");

            // loop the list
            for(int _lC = 0; _lC < _nodeSize; _lC++) {

                // add current element of list to string
                _results.append("\"" + _current.element);

                // advance the list
```

```
            _current = _current.next;

            // some fancy string manipulation
            if(_current != null) { _results.append("\", "); } // end if
            else { _results.append("\"]"); } // end else

        } // end _lC

    } // end else

    return _results.toString();

} // end show

/**
 *
 * <p><strong><em>Description: </em></strong>adds element to list</p>
 *
 * <p><strong><em>Method Name: </em></strong>add</p>
 *
 * <p><strong><em>Method Notes: </em></strong>adds element to list, if list does not exist it creates one</
p>
 *
 * <p><strong><em>Pre-Conditions: </em></strong>none</p>
 *
 * <p><strong><em>Post-Conditions: </em></strong>none</p>
 *
 * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
 * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
 *
 * @param element
 */
public void add(T element) {

    System.out.println("ENTER: add(T element), Adding: " + element);

    if(isEmpty()) {

        // the list is empty so create a new list
        _head = new Node<T>(element);

    } // end if
    else {

        // add element to the list

        // reset the head
        Node<T> _current = _head;

        // loop through the list till we get to the end
```

```
            while(_current.next != null) { _current = _current.next; } // end while

            // the above loop got us to the end of the list so we add the new lement to the list
            _current.next = new Node<T>(element);

        } // end else

        // increment the node counter
        Node.counter++;

    } // end add

    /**
     *
     * <p><strong><em>Description: </em></strong>delete element based on value, from any position in list</p
>
     *
     * <p><strong><em>Method Name: </em></strong>delete</p>
     *
     * <p><strong><em>Method Notes: </em></strong>for this method I do it a bit differently from the code su
pplied. instead of deleting the element i simply reposition the next value so it points to the node after the one i w
ant to delete. i let garbage collection clear up the node that has now been skipped and no longer available. if we
delete the first node i simply reposition the head to the original heads next node.</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>Pre-Conditions</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>Post-Conditions</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>Start Date</p>
     *
     * @param element
     */
    public void delete(T element) {

        System.out.println("ENTER: delete(T element), Deleting: " + element);

        // variables
        int _nodeSize = Node.counter;

        // is the list empty?
        if(_nodeSize == 0) {

            // list is empty
            System.out.println("Nothing to delete, List is Empty!");

        } // end if
        else {

            // list is not empty
```

```
        // are we trying to delete the head node?
        if(_head.element == element) {
            // list = {1, 2, 3, 4, 5}
            // we want to delete 1, the head
            // next value for 1 currently equals 2
            // we change the current head to where it equals its current next value which is 2
            // 1 is now gone and 2 is the new head

            _head = _head.next;

        } // end if
        else {

            // we are deleteing something other than the head

            // variables
            Node<T> _current = _head;

            // loop the list
            while(_current.next != null) {
                // list = {1, 2, 3, 4, 5}
                // we want to delete 3
                // next value for 2 currently equals 3
                // we change the next value of 2 from 3 to 4
                // 3 is now gone

                // if the value of the next node is equal to what we want to delete
                if(_current.next.element == element) {

                    // point the next counter for the current node to the next counter for the node we want to delete
                    _current.next = _current.next.next;
                    break; // get out of the list...we are done

                } // end if

                // advance the list
                _current = _current.next;

            } // end while

        } // end else

        // decrement the node count to reflect a removal
        Node.counter--;

    } // end else

} // end delete
```

```
    /**
     *
     * <p><strong><em>Description: </em></strong>icheck to see if the list is empty</p>
     *
     * <p><strong><em>Method Name: </em></strong>isEmpty</p>
     *
     * <p><strong><em>Method Notes: </em></strong>did not include sysout echo in this method as we will be
 in and out like a revolving door...will flood the console</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
     *
     * @return true if list is empty, false if it contains data
     */
    public boolean isEmpty() {

        if(getSize() == 0) { return true ;} // end if
        else { return false; } // end else

    } // end isEmpty

    /**
     *
     * <p><strong><em>Description: </em></strong>returns current sizer of list</p>
     *
     * <p><strong><em>Method Name: </em></strong>getSize</p>
     *
     * <p><strong><em>Method Notes: </em></strong>did not include sysout echo in this method as we will be
 in and out like a revolving door...will flood the console</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
     *
     * @return returns size of list
     */
    public int getSize() {

        return Node.counter;

    } // end getSize

    public void clear() {
```

```
        System.out.println("ENTER: clear(), clear entire list.");

        // clear the list
        _head = null;

        // rest the counter
        Node.counter = 0;


    } // end clear

    /**
     *
     * <p><strong><em>Description: </em></strong>adds element to front of list</p>
     *
     * <p><strong><em>Method Name: </em></strong>addFront</p>
     *
     * <p><strong><em>Method Notes: </em></strong>none</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
     *
     * @param element item to add to front of list
     */
    public void addFront(T element) {

        System.out.println("ENTER: addFront(T element), add element to front of list: " + element);

        // create a new node to contain the new head
        Node<T> _newHead = new Node<T>(element);

        //set the next for the _newHead to the current head
        _newHead.next = _head;

        // set the head to the _newHead
        _head = _newHead;

    } // end addFront

    /**
     *
     * <p><strong><em>Description: </em></strong>checks to see if the list contains a specific value</p>
     *
     * <p><strong><em>Method Name: </em></strong>contains</p>
     *
```

```
     * <p><strong><em>Method Notes: </em></strong>doing something a little differnt on the return value</p>
     *
     * <p><strong><em>Pre-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Post-Conditions: </em></strong>none</p>
     *
     * <p><strong><em>Author: </em></strong>Daniel C. Landon Jr.</p>
     * <p><strong><em>Start Date: </em></strong>04.20.2020</p>
     *
     * @param element value to search for in list
     * @return list contains boolean value for value found and index for where it can be found
     */
    public List<Object> contains(T element) {

        // variables
        boolean _found = false;
        int _index = 0;
        int _nodeSize = Node.counter;

        // reset the head
        Node<T> _current = _head;

        // loop the list
        for(int _lC = 0; _lC < _nodeSize; _lC++) {

            if(_current.element.equals(element)) {
                _found = true;
                _index = _lC;
                break; // bounce out
            } // end if

            // advance the list
            _current = _current.next;

        } // end _lC

        return Arrays.asList(_found, _index);

    } // end contains

} // end LinkedList
```

# ***Console Output***

```
********************************************************************************

    START TIME: 2020-04-20T10:40:04.641071500Z

********************************************************************************

>>>>>>>>>> STATE OF LIST <<<<<<<<<<

Is the list Empty: true
Number of elements in list: 0

ENTER: show(), displays list.

Nothing to display, List is Empty!

********************************************************************************

>>>>>>>>>> DELETE FROM AN EMPTY LIST <<<<<<<<<<

ENTER: delete(T element), Deleting: Bob
Nothing to delete, List is Empty!

Is the list Empty: true
Number of elements in list: 0

********************************************************************************

>>>>>>>>>> ADD DATA TO LIST <<<<<<<<<<

ENTER: add(T element), Adding: NX-01: Archer
ENTER: add(T element), Adding: NCC-1701: April
ENTER: add(T element), Adding: NCC-1701: Pike
ENTER: add(T element), Adding: NCC-1701: Kirk
ENTER: add(T element), Adding: NCC-1701: Decker
ENTER: add(T element), Adding: NCC-1701: Spock
ENTER: add(T element), Adding: NCC-1701-A: Kirk
ENTER: add(T element), Adding: NCC-1701-B: Harriman
ENTER: add(T element), Adding: NCC-1701-C: Garrett
ENTER: add(T element), Adding: NCC-1701-D: Picard
ENTER: add(T element), Adding: NCC-1701-D: Riker
ENTER: add(T element), Adding: NCC-1701-D: Jellico
ENTER: add(T element), Adding: NCC-1701-E: Picard

********************************************************************************

>>>>>>>>>> STATE OF LIST <<<<<<<<<<
```

Is the list Empty: false
Number of elements in list: 13

ENTER: show(), displays list.

["NX-01: Archer", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-C: Garrett", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico", "NCC-1701-E: Picard"]

**********

>>>>>>>>>> DELETE FROM RANDOM POINT IN LIST <<<<<<<<<<

Is the list Empty: false
Number of elements in list: 13

ENTER: show(), displays list.

["NX-01: Archer", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-C: Garrett", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico", "NCC-1701-E: Picard"]

ENTER: delete(T element), Deleting: NCC-1701-C: Garrett

Is the list Empty: false
Number of elements in list: 12

ENTER: show(), displays list.

["NX-01: Archer", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico", "NCC-1701-E: Picard"]

**********

>>>>>>>>>> DELETE THE HEAD <<<<<<<<<<

Is the list Empty: false
Number of elements in list: 12

ENTER: show(), displays list.

["NX-01: Archer", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico", "NCC-1701-E: Picard"]

ENTER: delete(T element), Deleting: NX-01: Archer

Is the list Empty: false
Number of elements in list: 11

ENTER: show(), displays list.

["NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D:
Jellico", "NCC-1701-E: Picard"]

**********

>>>>>>>>>> ADD NEW HEAD ELEMENT <<<<<<<<<<

Is the list Empty: false
Number of elements in list: 11

ENTER: show(), displays list.

["NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D:
Jellico", "NCC-1701-E: Picard"]

ENTER: addFront(T element), add element to front of list: Bob's Your Uncle

Is the list Empty: false
Number of elements in list: 11

ENTER: show(), displays list.

["Bob's Your Uncle", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico",

**********

>>>>>>>>>> DOES LIST CONTAIN <<<<<<<<<<

Is the list Empty: false
Number of elements in list: 11

ENTER: show(), displays list.

["Bob's Your Uncle", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico",

List contains, NCC-1701: Pike, at index:  2.

**********

>>>>>>>>>> CLEAR THE LIST <<<<<<<<<<

Is the list Empty: false
Number of elements in list: 11

ENTER: show(), displays list.

["Bob's Your Uncle", "NCC-1701: April", "NCC-1701: Pike", "NCC-1701: Kirk", "NCC-1701: Decker", "NCC-1701: Spock", "NCC-1701-A: Kirk", "NCC-1701-B: Harriman", "NCC-1701-D: Picard", "NCC-1701-D: Riker", "NCC-1701-D: Jellico",

ENTER: clear(), clear entire list.

Is the list Empty: true
Number of elements in list: 0

ENTER: show(), displays list.

Nothing to display, List is Empty!

**********

END TIME: 2020-04-20T10:40:04.882590100Z
Time for completion (milliseconds): 241

>>>>>>>>>> PROGRAM TERMINATED <<<<<<<<<<

END OF LINE