Notes:

- Had to remove the word “public” from the class MyLinkedList declaration as this was generating an error because of the two class declarations in one file.
- I moved “private static class Node<E>” to the top of the file as indicated at the bottom of the file????
- Added the package line to the top of file so it would work.
- All code for project assignment is at the end of the file.
- Added some extra spacing to the output for TestLinkedList.java so the project “results” would be easier to identify

MyLinkedList.java

```
// added line for package
package app;

class MyLinkedList<E> extends MyAbstractList<E> {

   // ////////  this should be first!!!!
   private static class Node<E> {
      E element;
      Node<E> next;
      public Node(E element) {
         this.element = element;
      }//constructor
   }//node

   private Node<E> head, tail;

   /** Create a default list */
   public MyLinkedList() {
   }

   /** Create a list from an array of objects */
   public MyLinkedList(E[] objects) {
      super(objects);
   }

   /** Return the head element in the list */
   public E getFirst() {
     if (size == 0) {
         return null;
      } else {
         return head.element;
      }
   }
```

```
    /** Return the last element in the list */
    public E getLast() {
        if (size == 0) {
            return null;
        } else {
            return tail.element;
        }
    }

    /** Add an element to the beginning of the list */
    public void addFirst(E e) {
        Node<E> newNode = new Node<E>(e); // Create a new node
        newNode.next = head; // link the new node with the head
        head = newNode; // head points to the new node
        size++; // Increase list size

        if (tail == null) // the new node is the only node in list
        {
            tail = head;
        }
    }
    /** Add an element to the end of the list */
    public void addLast(E e) {
        Node<E> newNode = new Node<E>(e); // Create a new for element e
        if (tail == null) {
            head = tail = newNode; // new node is the only node in list
        } else {
            tail.next = newNode; // Link the new with the last node
            tail = tail.next; // tail now points to the last node
        }
        size++; // Increase size
    }
    /** Add a new element at the specified index in this list
     * The index of the head element is 0 */
    public void add(int index, E e) {
        if (index == 0) {
            addFirst(e);
        } else if (index >= size) {
            addLast(e);
        } else {
            Node<E> current = head;
            for (int i = 1; i < index; i++) {
                current = current.next;
            }
            Node<E> temp = current.next;
            current.next = new Node<E>(e);
            (current.next).next = temp;
            size++;
        }
```

```
}

/** Remove the head node and
 *  return the object that is contained in the removed node. */
public E removeFirst() {
    if (size == 0) {
        return null;
    } else {
        Node<E> temp = head;
        head = head.next;
        size--;
        if (head == null) {
            tail = null;
        }
        return temp.element;
    }
}

/** Remove the last node and
 * return the object that is contained in the removed node. */
public E removeLast() {
    if (size == 0) {
        return null;
    } else if (size == 1) {
        Node<E> temp = head;
        head = tail = null;
        size = 0;
        return temp.element;
    } else {
        Node<E> current = head;
        for (int i = 0; i < size - 2; i++) {
            current = current.next;
        }
        Node<E> temp = tail;
        tail = current;
        tail.next = null;
        size--;
        return temp.element;
    }
}



/** Remove the element at the specified position in this list.
 *  Return the element that was removed from the list. */
public E remove(int index) {
    if (index < 0 || index >= size) {
        return null;
    } else if (index == 0) {
        return removeFirst();
        } else if (index == size - 1) {
            return removeLast();
        } else {
            Node<E> previous = head;
            for (int i = 1; i < index; i++) {
                previous = previous.next;
            }
            Node<E> current = previous.next;
            previous.next = current.next;
            size--;
            return current.element;
        }
    }

    /** Override toString() to return elements in the list */
    public String toString() {
        StringBuilder result = new StringBuilder("[");
        Node<E> current = head;
        for (int i = 0; i < size; i++) {
            result.append(current.element);
            current = current.next;
            if (current != null) {
                result.append(", "); // Separate two elements with a comma
            } else {
                result.append("]"); // Insert the closing ] in the string
            }
        }
        return result.toString();
    }

    /** Clear the list */
    public void clear() {
        head = tail = null;
    }







    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>contains</p>
     *
     * <p><strong><em>Method Notes: </em></strong>Returns true if this linked list contains the element e,
otherwise returns false.</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>04.17.2020</p>
     *
     * @param e item to check against the list
     * @return true if item is in the list, false if item is not in list
     */
    public boolean contains(E e) {

        // variables
        boolean _found = false;

        // set list to the beginning
        Node<E> _current = head;

        // loop until end of list
        while (_current != null) {

            if (_current.element.equals(e)) {
                // found it
                _found = true;
                break; // bounce out
            } // end if

            // advance the list
            _current = _current.next;

        } // end while

        return _found;

    } // end contains

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>get</p>
     *
     * <p><strong><em>Method Notes: </em></strong>Returns the element at specified index of this list,
returns null if index is invalid.</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>04.17.2020</p>
     *
     * @param index index if item to find
     * @return value at specified index, return null if index is invalid
     */
    public E get(int index) {

        // varialbes
        int _counter = 0;

        // repostion linked list at head
        Node<E> _current = head;

        // loop the list
        while (_current != null) {

            // do we have a match
            if(_counter == index) { return (_current.element); } // end if

            _counter ++; // increment counter

            _current = _current.next; // advance the list

        } // end while

        // if we get here the assumption is the index is invalid
        return null;

    } // end get

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>indexOf</p>
     *
     * <p><strong><em>Method Notes: </em></strong>Returns the index of the first matching element in this
linked list, return -1 if no match.</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>04.17.2020</p>
     *
     * @param e the item to look for in list
     * @return returns index position of first item to match e, otherwise returns -1 for no match
     */
    public int indexOf(E e) {
        int index = -1;
        Node<E> current = head;
        for (int i = 0; i < size; i++) {
            if (current.element.equals(e)) {
                index = i;
                break;
            }
            current = current.next;
        }
        return index;
    } // end indexOf

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>lastIndexOf</p>
     *
     * <p><strong><em>Method Notes: </em></strong>Returns the index of the last matching element in this list, returns -1 if no match.</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>04.17.2020</p>
     *
     * @param e item to search for in list
     * @return index of last matching element, -1 if no match
     */
    public int lastIndexOf(E e) {

        // varialbes
        int _index = -1;
        int _counter = 0;

        // reset list to head
        Node<E> _current = head;

        //loop the list
        while(_current != null) {

            // do we have a match
            // instead of breaking out of the loop we let it keep running just in case there is another element with the matching value
            if(_current.element.equals(e)) { _index = _counter; } // end if

            _counter ++; // increment counter
            _current = _current.next; // advance the list

        } // end while

        return _index;

    } // end lastIndexOf

    /**
     *
     * <p><strong><em>Description: </em></strong>Description</p>
     *
     * <p><strong><em>Method Name: </em></strong>set</p>
     *
     * <p><strong><em>Method Notes: </em></strong>Replaces the element at specified index in this linked list with the specified element.  Returns the old element at specified index, otherwise returns null if index is invalid.</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>04.17.2020</p>
     * @param index index position to modify
     * @param e new value for above index
     * @return old value that was replace otherwise returns -1 if index is invalid
     */
    public E set(int index, E e) {

        // varialbes
        int _counter = 0;
        E _oldValue = null;

        // repostion linked list at head
        Node<E> _current = head;

        // loop the list
        while (_current != null) {

            // do we have a match
            if(_counter == index) {

                _oldValue = _current.element;

                _current.element = e;

                return _oldValue;
            } // end if
            _counter ++; // increment counter

            _current = _current.next; // advance the list

        } // end while

        // if we get here the assumption is the index is invalid
        return null;

    } // end set



    // ////////  this should be first!!!!
    // private static class Node<E> {
    //     E element;
    //     Node<E> next;
    //     public Node(E element) {
    //         this.element = element;
    //     }//constructor
    // }//node
}//class
```

Console Output

```
(1) [America]
(2) [Canada, America]
(3) [Canada, America, Russia]
(4) [Canada, America, Russia, France]
(5) [Canada, America, Germany, Russia, France]
(6) [Canada, America, Germany, Russia, France, Norway]
(7) [Poland, Canada, America, Germany, Russia, France, Norway]
(8) [Canada, America, Germany, Russia, France, Norway]
(9) [Canada, America, Russia, France, Norway]
(10) [Canada, America, Russia, France]


The following is for Lab 6


(11) The list does not contain Germany
(12) Invalid position
(13) The list element France is at position 3
(14) [India, Canada, America, Russia, France]
(15) [India, Canada, America, Russia, France, America]
(16) The list element America occurs last at 5
(17) [India, Canada, America, Russia, France, China]
```