```
::::::::::::::
DNode.java
::::::::::::::

public class DNode extends Node {

    private DNode back;

    public DNode(Object newItem) {
        super(newItem);
        super.setNext(this);
        back = this;
    }

    public DNode(Object item, DNode nextNode, DNode backNode)
    {
        super(item, nextNode);
        this.setBack(backNode);
    }

    public DNode getBack()
    {
        return back;
    }

    public void setBack(DNode backNode) {
        back = backNode;
    }

    public DNode getNext()
    {
        return (DNode) super.getNext();
    }

    public void setNext(DNode nextNode)
    {
        super.setNext(nextNode);
    }

}
::::::::::::::
Driver.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested

 * Last update: 02/11/20

 * Submitted:  02/11/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;

public class Driver {
    static BufferedReader stdin = new BufferedReader (new InputStreamReader(System
.in));

    public static void main(String[] args) throws NumberFormatException, IOExcepti
on {


        LinkedList<Object> items = new LinkedList<Object>();
        boolean switchOn = true;

        System.out.println("\nSelect from the following menu:"
                           + "\n\t1. Insert item to list\n\t2. Remove item from li
st"
                           + "\n\t3. Get item from list"
                           + "\n\t4. Clear list\n\t5. Print size and content of li
st"
                           + "\n\t6. Reverse list" + "\n\t7. Exit program");

        while(switchOn == true)
        {
            System.out.print("\nMake your selection now: ");
            int selection = Integer.parseInt(stdin.readLine().trim());
            System.out.println(selection);

            switch(selection)
            {
            case 0:
                switchOn = false;
                System.out.println("Exiting program...Good Bye");
                break;
            case 1:
                System.out.print("\nYou are now inserting an item into the list.\n
\tEnter item: ");
                Object item = stdin.readLine().trim();
                System.out.println(item);

                System.out.print("\tEnter position to insert item in: ");
                int index = Integer.parseInt(stdin.readLine().trim());
                System.out.println(index);

                items = MenuHandling.add(items, index, item);
                break;
            case 2:
                System.out.print("\tEnter position to remove item from: ");
                int toRemove = Integer.parseInt(stdin.readLine().trim());
                System.out.println(toRemove);

                items = MenuHandling.remove(items, toRemove);

                break;
            case 3:
                System.out.print("\t\nEnter position to retrieve item from: ");
                int toRetrieve = Integer.parseInt(stdin.readLine().trim());
                System.out.println(toRetrieve);

                MenuHandling.get(items, toRetrieve);
```

```
                break;
            case 4:
                items.removeAll(items);
                break;
            case 5:

                MenuHandling.printAll(items);

                break;

            case 6:

                items = MenuHandling.reverseList(items);

                break;

            default:

                break;
            }
        }
    }
}


::::::::::::::
ListCDLSBased.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested

 * Last update: 02/11/20

 * Submitted:  02/11/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

// Please note that this code is slightly different from the textbook code
//to reflect the fact that the Node class is implemented using data encapsulation


// ************************************************
// Reference-based implementation of ADT list.
// ************************************************
public class ListCDLSBased implements ListInterface
{
    // reference to linked list of items
    private DNode head;
    private int numItems;

    public ListCDLSBased()
    {
        head = null;
        numItems = 0;
    }  // end default constructor

public boolean isEmpty()
{
    return size() == 0;
}  // end isEmpty

public int size()
{
    return numItems;
}  // end size

private Node find(int index)
{
    // --------------------------------------------------
    // Locates a specified node in a linked list.
    // Precondition: index is the number of the desired
    // node. Assumes that 0 <= index <= numItems
    // Postcondition: Returns a reference to the desired
    // node.
    // --------------------------------------------------
    DNode curr = head;
    if(index <= numItems/2)
    {
        for (int skip = 0; skip < index; skip++)
        {
            curr = curr.getNext();
        }
    }
    else
    {
        for(int skip = numItems-1; skip > index; skip--)
        {
            curr = curr.getBack();
        }
    }

    return curr;
} // end find

public Object get(int index)
throws ListIndexOutOfBoundsException
{
    if (index >= 0 && index < numItems)
    {
        // get reference to node, then data in node
        Node curr = find(index);
        Object dataItem = curr.getItem();
        return dataItem;
    }
    else
    {
        throw new ListIndexOutOfBoundsException(
            "List index out of bounds exception on get");
    } // end if
} // end get

public void add(int index, Object item)
throws ListIndexOutOfBoundsException
{
    if (index >= 0 && index < numItems+1)
```

```
        {
            numItems++;
            if(numItems == 0)
            {
                DNode newNode = new DNode(item);
                head = newNode;
            }
            else if(index == numItems)
            {
                DNode prev = head.getBack();
                DNode newNode = new DNode(item, head, prev);
                head.setBack(newNode);
                prev.setNext(newNode);
            }
            else
            {
                DNode toFind = (DNode) find(index);
                DNode toAdd = new DNode(item, toFind, toFind.getBack());

                toFind.getBack().setNext(toAdd);
                toFind.setBack(toAdd);
            }
        }

        else
        {
            throw new ListIndexOutOfBoundsException(
                "List index out of bounds exception on add");
        } // end if
    }  // end add

    public void remove(int index)
    throws ListIndexOutOfBoundsException
    {
        if (index >= 0 && index < numItems)
        {
            numItems--;

            if(index == 0)
            {
                head = (DNode) find(index).getNext();
            }
            else if (index == numItems)
            {
                find(index-1).getNext().setNext(null);
            }
            else
            {
                DNode nodeBack = (DNode) find(index-1);
                DNode nodeFront = (DNode) find(index+1);

                nodeBack.setNext(nodeFront);
                nodeFront.setBack(nodeBack);
            }

        }
        else
        {
            throw new ListIndexOutOfBoundsException(
                "List index out of bounds exception on remove");
        } // end if
    }   // end remove

    public void removeAll()
    {
        // setting head to null causes list to be
        // unreachable and thus marked for garbage
        // collection
        head = null;
    } // end removeAll

    public String toString()
    {
        Node next = head;
        StringBuilder builder = new StringBuilder();
        String toReturn = "";

        while(next != null)
        {
            String name = next.getItem().toString() + " ";
            builder.append(name);
            next.getNext();
        }
        toReturn = builder.toString();

        return toReturn;
    }

} // end ListReferenceBased::::::::::::::
ListIndexOutOfBoundsException.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested

 * Last update: 02/11/20

 * Submitted:  02/11/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

public class ListIndexOutOfBoundsException
    extends IndexOutOfBoundsException
{
    public ListIndexOutOfBoundsException(String s)
    {
        super(s);
    }  // end constructor
}  // end ListIndexOutOfBoundsException::::::::::::::
ListInterface.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested
```

```
 * Last update: 02/11/20

 * Submitted:  02/11/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

// ******************************************************
// Interface ListInterface for the ADT list.
// ******************************************************
public interface ListInterface
{
    boolean isEmpty();
    int size();
    void add(int index, Object item) throws ListIndexOutOfBoundsException;
    Object get(int index) throws ListIndexOutOfBoundsException;
    void remove(int index) throws ListIndexOutOfBoundsException;
    void removeAll();
    String toString();
}  // end ListInterface::::::::::::::
MenuHandling.java
::::::::::::::
import java.util.LinkedList;

public class MenuHandling {

    public static LinkedList<Object> add(LinkedList<Object> items, int index, Obje
ct item)
    {
        if(index >= items.size()+1)
        {
            System.out.println("\nPosition specified is out of range!");
        }
        else
        {
            items.add(index, item);
            System.out.println("\nItem " + item + " inserted at position " + index
 + " in the list.");
        }

        return items;
    }

    public static LinkedList<Object> remove(LinkedList<Object> items, int toRemove
) {
        if(items.size() == 0)
        {
            System.out.println("List is empty.");
        }
        else if((toRemove >= items.size()) || (toRemove < 0))
        {
            System.out.println("\nPosition specified is out of range!");
        }
        else
        {
            System.out.println("\nItem " + items.get(toRemove) + " removed from po
sition " + toRemove + " in the list.");
            items.remove(toRemove);
        }

        return items;
    }

    public static void get(LinkedList<Object> items, int toRetrieve) {
        if((toRetrieve >= items.size()) || (toRetrieve < 0))
        {
            System.out.println("\nPosition specified is out of range!");
        }
        else
        {
            System.out.println("\nItem " + items.get(toRetrieve) + " retrieved fro
m position " + toRetrieve + " in the list.");
        }
    }

    public static void printAll(LinkedList<Object> items) {
        if(items.size() == 0)
        {
            System.out.println("List is empty.");
        }
        else
        {
            System.out.print("List of size " + items.size() + " has the following
items: ");

            for(int i = 0; i < items.size(); i++)
            {
                System.out.print(items.get(i) + " ");
            }
        }
    }

    public static LinkedList<Object> reverseList(LinkedList<Object> items) {
        if(items.size() == 0)
        {
            System.out.println("List is empty... nothing to reverse!");
        }
        else
        {

            for(int i = 0, k = items.size()-1; i < items.size()/2; i++, k--)
            {
                Object toFront = items.get(k);
                Object toBack = items.get(i);

                items.add(i, toFront);
                items.remove(i+1);

                items.add(k, toBack);
                items.remove(k+1);
            }

            System.out.println("List reversed");
        }
        return items;
    }
```

```
}
::::::::::::::
Node.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested

 * Last update: 02/11/20

 * Submitted:  02/11/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

//please note that this code is different from the textbook code, because the data
 is encapsulated!

public class Node
{
    private Object item;
    private Node next;
    private Node back;

    public Node(Object newItem)
    {
        item = newItem;
        next = null;
    } // end constructor

    public Node(Object newItem, Node nextNode)
    {
        item = newItem;
        next = nextNode;
    } // end constructor

    public void setItem(Object newItem)
    {
        item = newItem;
    } // end setItem

    public Object getItem()
    {
        return item;
    } // end getItem

    public void setNext(Node nextNode)
    {
        next = nextNode;
    } // end setNext

    public Node getNext()
    {
        return next;
    } // end getNext


} // end class Node::::::::::::::
output.txt
::::::::::::::

Select from the following menu:
        1. Insert item to list
        2. Remove item from list
        3. Get item from list
        4. Clear list
        5. Print size and content of list
        6. Reverse list
        7. Exit program

Make your selection now: 5
List is empty.

Make your selection now: 6
List is empty... nothing to reverse!

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Pikachu
        Enter position to insert item in: 0

Item Pikachu inserted at position 0 in the list.

Make your selection now: 5
List of size 1 has the following items: Pikachu
Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Bulbasaur
        Enter position to insert item in: 0

Item Bulbasaur inserted at position 0 in the list.

Make your selection now: 5
List of size 2 has the following items: Bulbasaur Pikachu
Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Charizard
        Enter position to insert item in: 4

Position specified is out of range!

Make your selection now: 5
List of size 2 has the following items: Bulbasaur Pikachu
Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Charizard
        Enter position to insert item in: 2

Item Charizard inserted at position 2 in the list.

Make your selection now: 6
List reversed
```

```
Make your selection now: 5
List of size 3 has the following items: Charizard Pikachu Bulbasaur
Make your selection now: 6
List reversed

Make your selection now: 5
List of size 3 has the following items: Bulbasaur Pikachu Charizard
Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Mew
        Enter position to insert item in: 1

Item Mew inserted at position 1 in the list.

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Abra
        Enter position to insert item in: 3

Item Abra inserted at position 3 in the list.

Make your selection now: 5
List of size 5 has the following items: Bulbasaur Mew Pikachu Abra Charizard
Make your selection now: 2
        Enter position to remove item from: 7

Position specified is out of range!

Make your selection now: 2
        Enter position to remove item from: 3

Item Abra removed from position 3 in the list.

Make your selection now: 5
List of size 4 has the following items: Bulbasaur Mew Pikachu Charizard
Make your selection now: 2
        Enter position to remove item from: 0

Item Bulbasaur removed from position 0 in the list.

Make your selection now: 5
List of size 3 has the following items: Mew Pikachu Charizard
Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Kadabra
        Enter position to insert item in: 1

Item Kadabra inserted at position 1 in the list.

Make your selection now: 6
List reversed

Make your selection now: 5
List of size 4 has the following items: Charizard Pikachu Kadabra Mew
Make your selection now: 3

Enter position to retrieve item from: 2

Item Kadabra retrieved from position 2 in the list.

Make your selection now: 3

Enter position to retrieve item from: 0

Item Charizard retrieved from position 0 in the list.

Make your selection now: 3

Enter position to retrieve item from: 4

Position specified is out of range!

Make your selection now: 4

Make your selection now: 5
List is empty.

Make your selection now: 6
List is empty... nothing to reverse!

Make your selection now: 0
Exiting program...Good Bye
:::::::::::::::
ComplexityAnalysis.txt
:::::::::::::::

Space Complexity Analysis

1. Critical Operations
   Critical operations for this program are find(), get(), add(), and remove().

2. Count # of Critical Operations
   4 total.

3. Express # of Critical Operations as a Function
   numItems = q
   Each item has a front and back, therefore making it 3 times bigger, and the
   equation:

   4 * 3(q) = 12 bytes

Time Complexity Analysis

isEmpty() - 0
size() - 0
removeAll - 0
toString - 0

get()
BC - 0
WC - (n-1)/2 (because of the ability to go backwards)
AC - (n-1)/4

add()
BC - 0
WC - (n-1)/2
AC  (n-1)/4

remove()
BC - 0
```

WC - n/2
AC - n+1/4

::::::::::::::
Lab4Conclusions.txt
::::::::::::::
I feel kinda shakey on the analysis but it is what it is at **this** point.

What I got out of **this** lab was a deeper understanding of how the LinkedArray really works. Its so simple to make it circular *at the cost* of memory, which is an important lesson to take out of things.

Efficiency isn't just something you get - the give and take between efficiency and simplicity isn't 1:1, so you need to carefully measure things out.