```
/**
Code for the creation and manipulation of a Data Structure called a LinkedList which is essentially a
@author Yanni Angelides
@version 11/30/15
*/

import java.util.Iterator;
import java.lang.Iterable;
import java.util.NoSuchElementException;

public class LinkedList<E> implements Iterable<E>//Stack<E>, Queue<E>
{
    private ListNode head;
    private ListNode tail;
    private int size;
    //singley link
    //head pointer
    //tail pointer

    /**
    Creates a completely empty LinkedList
    */
    public LinkedList()
    {
        head = null;
        tail = null;
        size = 0;
    }

    /**
    Creates a LinkedList that contains one ListNode
    @param the ListNode that is used to create the LinkedList
    */
    //Precondition that h.getNext() == null
    public LinkedList(ListNode<E> h)
    {
        head = h;
        tail = h;
        size = 1;
    }

    /**
    Creates a LinkedList that is
    @param LinkedList that is copied to create a new LinkedList
    */
    public LinkedList(LinkedList<E> list)
    {
        if (list == null)
        {
            throw new IllegalArgumentException(); //cannot copy a LinkedList that is null
        }
        else
        {
            for (ListNode<E> i = list.get(0); i != list.get(list.size()); i = i.getNext()) //starts at
            {
                if(size == 0) //if size == 0 then you have to set the head to whatever ListNode you ar
                {
                    i.setNext((list.get(0)).getNext());
                    head = i;
                    size++;
                }
                else
                {
                    i.setNext(i.getNext()); //sets the hext of the ListNode being created to the next
                    size++;
                }
            }
            tail = list.get(list.size());
            size++;
        }
    }

    /**
    A helper method that can get the ListNode at a specific index in the LinkedList
```

????

???

Isn't this method supposed to be copying the contents of list?

```java
    @param int index of the ListNode that is needed
    @return ListNode the ListNode that is needed
    */
    public ListNode<E> get(int indx)
```

get returns E, not ListNode<E>

```java
    {
        if (index >= size || index < 0)
        {
            throw new IndexOutOfBoundsException();
        }
        else
        {
            ListNode trav = head;
            for (int i = 0; i < indx; i++) //The for loop will start at the head of the LinkedList and
            {
                trav = trav.getNext();
            }
            return trav;
        }
    }

    /**
    Finds the index of a specified ListNode within the LinkedList
    @param ListNode<E> that needs to be found
    @return int index of the ListNode<E> passed in as a parameter
    */
    public int indexOf(ListNode<E> n)
```

E, not ListNode<E>

```java
    {
        ListNode trav = head;
        for (int i = 0; i < size; i++)
        {
            if(trav == n) //same for loop as the one in the above method except that each time it goes
            {
                return i;
            }
            trav = trav.getNext();
        }
        return 0;
    }

    /**
    @return int size of the LinkedList
    */
    public int size()
    {
        return size;
    }

    /**
    Completely empties the LinkedList so that there are no ListNodes in it

    */
    public void clear()
    {
        head = null;
        tail = null;
        size = 0;
    }

    /**
    Sets a specific index of the LinkedList to an E passed in as a paramater
    @param int index that needs to be changed, E item that you want to set the specified index to
    */
    public void set(int indx, E item)
    {
        if (index >= size || index < 0)
        {
            throw new IndexOutOfBoundsException();
        }
        ListNode<E> n = new ListNode<E>(item); //Need to make the item into a ListNode so that it can
        if(indx == 0)
        {
            n.setNext(head.getNext()); //if indx == 0 then the item parameter must become the head
            head = n;

        }
        else if(indx == size)
        {
            get(size - 1).setNext(n); //if indx == size the item parameter must become the new tail
            tail = n;
        }
        else
        {
            (get(indx - 1)).setNext(n);
            n.setNext(get(indx+1));
        }
    }
```

This won't work.

```java
    /**
    Checks if the item passed in as a parameter is contained in the LinkedList
    @param E item that the LinkedList needs to be checked for
    @return boolean indicating if the object is contained within the LinkedList
    */
    public boolean contains(E item)
    {
        for(ListNode<E> i = head; i != tail; i = i.getNext())
        {
            if(i.getValue() == item)
            {
                return true;
            }
        }
        return false;
    }
```

Need to
use .equals

Remember, == compares memory
address. The .equals method actually
compares the values.

```java
    /**
    Takes out a specific item from the LinkedList
    @param E item that needs to be removed
    @return boolean indicating if the item was removed or not
    */
    public boolean remove(E item)
    {
        if(head.getValue() == item)
        {
            removeFirst();
        }
        else if(tail.getValue() == item)
        {
            removeLast();
        }
        else
        {
            for(ListNode<E> i = head; i != tail; i = i.getNext())
            {
                if((i.getNext()).getValue() == item) //finds the item within the LinkedList
                {
                    i.setNext((i.getNext()).getNext()); //sets the next class field of the ListNode be
                    size--;
                    return true;
                }
            }
            return false;
        }
        return false;
    }

    /**
    Removes the ListNode at a certain index in the LinkedList
    @param int index that needs to be removed
    @return boolean indicating whether or not the index was removed
    */
    public boolean remove(int indx)
    {
        if (index >= size || index < 0)
        {
            throw new IndexOutOfBoundsException();
        }
        if (indx == 0)
        {
            this.removeFirst();
            return true;
        }
        if (indx == size)
        {
            this.removeLast();
            return true;
        }
        else
        {
            (get(indx - 1)).setNext((get(indx)).getNext());
            size--;
            return true;
        }
    }

    /**
    Adds a specific item to the LinkedList at a specified index
    @param int indx, index where the item needs to be added, E item that needs to be added
    */
    public void add(int indx, E item)
    {
        if (index > size || indx < 0)
        {
            throw new IndexOutOfBoundsException();
        }
        if (indx == 0)
        {
            this.addFirst(item);
        }
        if (indx == size)
        {
            this.addLast(item);
        }
        else
        {
            ListNode<E> n = new ListNode<E>(item);
            ListNode<E> node = head;
            for(int i = 0; i < indx - 1; i++)
            {
                node = node.getNext();
            }
            n.setNext(node.getNext());
            node.setNext(n);
        }
        size++;
    }

    /**
    Adds the specified item to the end of the LinkedList
    @param E item that needs to be added
    */
    public void add(E item)
    {
        if (head == null) //Just in case the LinkedList is empty
        {
            this.addFirst(n);
        }
        else
        {
            this.addLast(n);
        }
        size++;
    }

    /**
    Creates a String representation of the LinkedList
    @return String that represents the LinkedList
    */
    public String toString()
    {
        String str = " ";
        for (ListNode<E> i = head; i != tail; i = i.getNext())
        {
```

There is no n in this method. Do you mean item?

```java
            str += i.toString(); //this to String method is for ListNode because i is a ListNode, met
        }
        return str;
    }

    /**
    Removes the first ListNode in the LinkedList
    @return ListNode removed from the LinkedList
    */
    public ListNode<E> removeFirst()
    {
        ListNode<E> save = head; //have to save head here because once it is removed from the LinkedI
        head = head.getNext();
        size--;
        return head;
    }
```

Check to make sure head isn't null

```java
    /**
    Removes the last ListNode from the LinkedList
    @return ListNode that was removed
    */
    public ListNode<E> removeLast()
    {
        ListNode<E> trav = head; //have to traverse to the back of the LinkedList because you can't j
        for(int i = 0; i < size - 1; i++)
        {
            trav = trav.getNext();
        }
        ListNode<E> save = tail;
        tail = trav;
        size--;
        return tail;
    }

    /**
    Adds an item to the beginning of the LinkedList
    @param E item that needs to be added
    */
    public void addFirst(E item)
    {
        ListNode n = new ListNode(item); //item must be turned into a ListNode before being added to
        n.setNext(head);
        head = n;
        size++;
    }
```

Good.

```java
    /**
    Adds a specified item to the beginning of the LinkedList
    @param E item that needs to be added
    */
    public void addLast(E item)
    {
        ListNode n = new ListNode(item);
        tail.setNext(n);
        tail = n;
        size++;
    }

    /**
    Checks if the LinkedList is empty (does not have any ListNodes in it)
    @return boolean indicating whether or not the LinkedList is empty
    */
    public boolean isEmpty()
    {
        if(size == 0) //size will always be zero if the LinkedList is empty
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    /**
    Method that allows for the creation of an Iterator of LinkedList so for loops can be used with it
    @return Iterator<E> created in the LinkedListIterator class
    */
    public Iterator<E> iterator()
    {
        return new LinkedListIterator(head); //parameter is head because that is where the LinkedList
    }

    public static void main(String [] args)
    {
        LinkedList arr = new LinkedList();
        ListNode<Integer> n1 = new ListNode<Integer>(5);
        ListNode<Integer> n2 = new ListNode<Integer>(9);
        ListNode<Integer> n3 = new ListNode<Integer>(234);
        arr.add(n1);
        arr.add(n2);
        arr.add(1, n3);
        System.out.println(arr.toString());
    }
    //add
    //remove

    //STACK

    /**
    Adds an item to the top of the stack
    @param E item to be added to the stack
    */
    public void push(E item)
    {
        this.addFirst();
    }

    /**
    Gets whatever ListNode is at the top of the stack
    @return ListNode<E> at the top of the stack
    */
    public ListNode<E> peek()
    {
        return head;
    }

    /**
    Removes the item at the top of the Stack
    @return E item removed from the Stack
    */
    public E pop()
    {
        ListNode<E> n = new ListNode<E>()
        n = this.removeFirst();
        this.removeFirst();
        return n.getValue();
    }

    //QUEUE

    /**
    Adds item to the end of the Queue
    @param E item to be added to the Queue
    */
    public void offer(E item)
    {
        this.addLast();
    }

    /**
    gets the last ListNode in the Queue
    @return ListNode<E> that is at the end of the Queue
    */
    public ListNode<E> peek()
    {
        return tail ;
    }
```

Did this actually run when you tested?

You're missing a semi-colon. Also, why do you need a ListNode?

```

    /**
    Removes the last ListNode in the Queue
    @return E value of the last ListNode in the Queue
    */
    public E poll()
    {
        ListNode<E> n = new ListNode<E>()
        n = this.removeLast();
        this.removeLast();
        return n.getValue();
    }

/*
removeFirst()
E item = list.get(0)
list.add(0, item)
if(list.removeFirst() == item)

LinkedListIterator(ListNode<E> head)
    curr = head

Declare stack and queue as a Linked List and test pull, pop, peek...

*/
}

/**
This is the class that allows for the creation of an object that will act as the base of
the LinkedList
@author Yanni Angelides
@version 11/30/15
*/

public class ListNode<E>
{
    private E item;
    private ListNode<E> next;

    /**
    Constructs a ListNode<E> with a specified item
    */
    public ListNode(E e)
    {
        item = e;
        next = null;
    }

    /**
    Constructs a ListNode with a specified item and pointer that points to the next object in the Lin
    */
    public ListNode(E e, ListNode<E> ln)
    {
        item = e;
        next = ln;
    }

    /**

    @return E value of the item in the ListNode
    */
    public E getValue()
    {
        return item;
    }

    /**
    @return the next ListNode in the LinkedList
    */
    public ListNode<E> getNext()
    {
        return next;
    }

```

```java
    /**

    @param E the value that the item class field in the ListNode needs to be set to
    */
    public void setValue(E e)
    {
        item = e;
    }

    /**

    @param the ListNode that the current ListNode is set to point to
    */
    public void setNext(ListNode<E> ln)
    {
        next = ln;
    }

    /**
    @return String representation of the ListNode
    */
    public String toString()
    {
        String node = " ";
        node += "Item: " + item.toString();
        return node;
    }
}

/**
Iterface of the Queue class which is an offshoot of a LinkedList
@author Yanni Angelides
@version 11/30/15
*/
public interface Queue<E>
{
    void offer(E item);
    // offer adds to the end of the Linked list so that the end pointer is always the same thing
    E poll();
    E peek();
    boolean isEmpty();
}

/**
Iterface of the Queue class which is an offshoot of a LinkedList
@author Yanni Angelides
@version 11/30/15
*/
public interface Stack<E>
{
    void push(E item);
    // adds to the beginning of the list so that the end pointer is always the same thing
    E pop();
    E peek();
    boolean isEmpty();
}

/**
LinkedListIterator class which is called by the iterator method in LinkedList so that for loops can b
@autor Yanni Angelides
@version 11/30/15
*/
import java.util.NoSuchElementException;
import java.util.Iterator;
import java.lang.Iterable;

public class LinkedListIterator<E> implements Iterator<E>
{
    private ListNode<E> curr;

    public LinkedListIterator(ListNode<E> head)
    {
        curr = head;
    }

    /**
    This is a method from the Iterator Interface that returns the next object in the Vector
    @return E the next object of type E in the Vector
    */
    public E next()
    {
        if(hasNext() == false)
        {
            throw new NoSuchElementException(); //Because if hasNext() == false then there us no such
        }
        else
        {
            E item = curr.getItem();
            curr = curr.getNext();
            return item;
        }
    }

    /**
    @return boolean determining whether or not there is another object in the Vector
    */
    public boolean hasNext()
    {
        return curr != null;
    }
}

```

I'd like you to update this code. I tried fixing your various compile errors, but they were too many. How did you actually test this code? Of particular concern is when you use ListNode instead of E and == instead of .equals. I'll commit the .java files I created out of your readme file. I fixed a few compile errors, but you should do the rest.