Problem 1:
Use MyStringList class and implement the Sort method to sort the arraylist using the CompareTo method of the String.

```
public class MyStringList {
	private final int INITIAL_LENGTH = 4;
	private String[] strArray;
	private int size;

	public MyStringList() {
		strArray = new String[INITIAL_LENGTH];
		size = 0;
	}
	// Add element in last
	public void add(String s){

		if(s==null) return;
		if(size == strArray.length) resize();
		strArray[size++] = s;
	}

	public String get(int i){
		if(i < 0 || i >= size){
			return null;
		}
		return strArray[i];
	}

	public boolean find(String s){
		if(s==null) return false;
		for(String test : strArray){
			if(test.equals(s)) return true;
		}
		return false; // The element is not in the list
	}

	public void insert(String s, int pos){
		if(pos > size || pos<0 ) return;
		if(pos == strArray.length||size+1 > strArray.length) {
			resize();
		}
		String[] temp = new String[strArray.length+1];
```

```
                System.arraycopy(strArray,0,temp,0,pos); // src, spos,des,dspos,number of
elements
                temp[pos] = s;

                System.arraycopy(strArray,pos,temp,pos+1, strArray.length - pos);
                strArray = temp;
                ++size;
        }

/*      public void insert(String s, int pos) {
        if(pos<0 || pos > size) return;
                if(pos == strArray.length||size+1 > strArray.length) {
                        resize ();
                }
                String[] temp = new String[strArray.length+1];
    for(int i = 0; i < pos; i++)
     temp[i] = strArray[i];
  temp[pos] = s;
  for(int i = pos + 1; i < strArray.length; i++)
    temp[i] =strArray[i - 1];
  strArray = temp;
                ++size;
}*/

        public boolean remove(String s){
                if(size == 0) return false; // list is empty
                if(s==null) return false;
                int index = -1;
                for(int i = 0; i < size; ++i ){
                        if(strArray[i].equals(s)){
                                index = i;
                                break;
                        }
                }
                if(index==-1) return false; // s is not found in the list
                String[] temp = new String[strArray.length];
                System.arraycopy(strArray,0,temp,0,index);
                System.arraycopy(strArray,index+1,temp,index,strArray.length-(index+1));
                strArray = temp;
                --size;
                return true;
        }

        private void resize(){
```

```
                System.out.println("resizing");
                int len = strArray.length;
                int newlen = 2*len;
                String[] temp = new String[newlen];
                System.arraycopy(strArray,0,temp,0,len);
          // strArray = Arrays.copyOf(strArray, newlen);
                strArray = temp;
        }
        public String toString(){
                StringBuilder sb = new StringBuilder("[");
                for(int i = 0; i < size-1; ++i){
                        sb.append(strArray[i]+", ");
                }
                sb.append(strArray[size-1]+"]");
                return sb.toString();
        }
        public int size() {
                return size;
        }
        public boolean isEmpty(){
                return(size==0);
        }
   public Object clone()
   {
        String[] temp = Arrays.copyOf(strArray, size);
        return temp;

   }}
```

Problem 2:
Use the SinglyLinkedList class and implement RemoveLast method. This method should remove the last occurrence of an integer value that is passed as a parameter.

```
 //Represent a node of the singly linked list
public class SinglyLinkedList {
class Node{
   int data;
   Node next;

   public Node(int data) {
      this.data = data;
      this.next = null;
   }
}
```

```
//Represent the head and tail of the singly linked list
public Node head = null;
public Node tail = null;

//addNode() will add a new node to the list
public void addNode(int data) {
   //Create a new node
   Node newNode = new Node(data);

   //Checks if the list is empty
   if(head == null) {
      //If list is empty, both head and tail will point to new node
      head = newNode;
      tail = newNode;
   }
   else {
      //newNode will be added after tail such that tail's next will point to newNode
      tail.next = newNode;
      //newNode will become new tail of the list
      tail = newNode;
   }
}

public boolean find(int n) {
          Node current  = head;
          if(head==null)
          {return false;}

          while(current!=null) {
                    if (current.data==n) {
                              return true;
                    }
                    current = current.next;
          }
          return false;
}
//display() will display all the nodes present in the list
public void display() {
   //Node current will point to head
   Node current = head;

   if(head == null) {
      System.out.println("List is empty");
```

```
      return;
    }
    System.out.println("Nodes of singly linked list: ");
    while(current != null) {
      //Prints each node by incrementing pointer
      System.out.print(current.data + " ");
      current = current.next;
    }
    System.out.println();
}  }
```

Problem 3:

Use the DuoblyLinkedList class and implement Remove method. This method should remove the node that contains a String value that is passed as a parameter.

```
public class MyStringDLinkedList {
        Node header;
        MyStringDLinkedList(){
                header = new Node(null,null, null);
        }
        public void addFirst(String item){
                Node n = new Node(header,item,header.next);
                if(header.next != null){
                        header.next.previous = n;
                }
                header.next = n;
        }
        /** returns the index of the String s, if found;
         * -1 otherwise
         */
        public int find(String s){
                if(s == null) return -1;
                Node currentNode = header;
                int i = -1;
                while(currentNode.next != null){
                        ++i;
                        currentNode = currentNode.next;
                        if(s.equals(currentNode.value)) return i;
                }
                return -1;
        }
```

```
	public int size(){
		int count = 0;
		Node next = header.next;
		while(next != null){
			++count;
			next = next.next;
		}
		return count;
	}
	private Node getNode(int pos){
		if (pos >= size() && (pos<0)) throw new IndexOutOfBoundsException();
		Node next = header;
		for(int i = 0; i <= pos; ++i){
			next = next.next;
		}
		//next is the node we are seeking
		return next;
	}
	public String get(int pos){
		Node node = getNode(pos);
		return (node != null) ? node.value : null;
	}
	public void insert(String s, int pos) {
		//corrected to throw exception
		if(pos > size() && (pos<0)) {
			throw new IndexOutOfBoundsException("pos = "+pos +" but size =
"+size());
		}
		Node next = header;
		Node previous = null;
		for(int i = 0; i <= pos; ++i){
			if(i==pos){
				previous = next;
			}
			next = next.next;
		}
		Node insertNode = new Node(previous,s,next);
		if(next != null){
			next.previous = insertNode;
		}
		previous.next =insertNode;
	}
	public boolean isEmpty() {
		if(header.next == null || size()==0)
```

```
                return true;
        else
                return false;
}
/** remove object at specified index */
public boolean remove(int index){
        Node toBeRemoved = getNode(index);
        if(toBeRemoved == null) return false;
        Node previous = toBeRemoved.previous;
        Node next = toBeRemoved.next;
        previous.next = next;
        if(next != null){
                next.previous = previous;
        }
        toBeRemoved = null;
        return true;
}
/** remove by specifying object -- removes
 *  first occurrence of s
 */
public boolean remove(String s){
        int pos = find(s);
        if(pos == -1) return false;
        return remove(pos);

}
void displayNodes() {
        Node next = header.next;
        while(next.next != null){
                System.out.print(next.value + "-->");
                next = next.next;
        }
        System.out.println(next.value);
}

class Node {
        Node previous;
        String value;
        Node next;

        Node(Node previous, String value, Node next){
                this.previous = previous;
                this.value = value;
                this.next = next;
```

```
            }

        }
        public static void main(String[] args) {
            MyStringDLinkedList list = new MyStringDLinkedList();
            System.out.println("Is Empty : "+list.isEmpty());
            list.addFirst("Java");
            list.addFirst("C#");
            list.insert("Android", 0);
            list.displayNodes();
            System.out.println(list.size());
            System.out.println(list.get(0));
            System.out.println(list.find("Java"));
            System.out.println(list.remove(1));
            list.displayNodes();
            System.out.println(list.size());
            System.out.println("Is Empty : "+list.isEmpty());
        }
}
```

Write main methods to test problem 1, problem 2 and problem 3