Data Structure & Algorithms.

By Ramana Sir.

Q1 :- ArrayList.

```
public class Node
{
   Object ele;
   Node next;
   public Node(Object ele,Node next)
   {
      this.ele = ele;
      this.next = next;
   }
   public Node(Object ele)
   {
      this.ele = ele;
   }
}
public class ArrayList
{
   Object [] a = new Object[5];
   int count = 0;
   public void add(Object ele)
   {
      if(count==a.length)increase();
         a[count] = ele;
         count++;
   }
```

```
public void increase()
{
   Object [] a1 = new Object[a.length + 5];
   for(int i = 0;i<size();i++)
   {
      a1[i] = a[i];
   }
   a = a1;
}
public int size()
{
   return count;
}
public Object get(int index)
{
   if(index<0||index>size())
   throw new IndexOutOfBoundsException();
   return a[index];
}
public int indexOf(Object ele)
{
   for(int i = 0;i<size();i++)if(a[i]==ele)return i;
   return -1;
}
public void set(Object ele,int index)
{
   if(index<0||index>size())
   throw new IndexOutOfBoundsException();
   a[index] = ele;
```

```
    }
    public void addPos(Object ele,int index)
    {
        if(index<0||index>size())
        throw new IndexOutOfBoundsException();
        if(index==count)increase();
        for(int i = size();i>index;i--)a[i] = a[i-1];
        a[index] = ele;
        count++;
    }
    public Object remove(int index)
    {
        if(index<0||index>size())
        throw new IndexOutOfBoundsException();
        Object ele = a[index];
        for(int i = index+1;i<size();i++)a[i-1] = a[i];
        count--;
        return ele;
    }
}
```

Q2 :- LinkedList.

```
public class Node
{
    Object ele;
    Node next;
    public Node(Object ele,Node next)
    {
        this.ele = ele;
```

```
        this.next = next;
    }
    public Node(Object ele)
    {
        this.ele = ele;
    }
}
public class LinkedList
{
    Node head;
    Node tail;
    int count = 0;
    public void insertFirst(Object ele)
    {
        Node node = new Node(ele);
        node.next = head;
        head = node;
        count++;
        if(tail==null)tail = head;
    }
    public int size()
    {
        return count;
    }
    public void display()
    {
        Node temp = head;
        while(temp!=null)
        {
```

```
        System.out.print(temp.ele + " ");
        temp = temp.next;
    }
}
public void add(Object ele)
{
    if(count==0)
    {
        insertFirst(ele);
        return;
    }
    Node temp = head;
    while(temp.next!=null)temp = temp.next;
    temp.next = new Node(ele,null);
    count++;
}
public void inserLast(Object ele)
{
    Node temp = head;
    while(temp.next!=null)temp = temp.next;
    Node node = new Node(ele);
    temp.next = node;
    tail = node;
    tail.next = null;
    count++;
}
public void insertInBetween(Object ele,int index)
{
    Node temp = head;
```

```
    for(int i = 1;i<index;i++)temp = temp.next;
    Node node = new Node(ele,temp.next);
    temp.next = node;
    count++;
}
public Object getEle(int index)
{
    if(index<0||index>size())
    throw new IndexOutOfBoundsException();
    Node temp = head;
    for(int i = 0;i<index;i++)temp = temp.next;
    return temp.ele;
}
public Object removeFirst()
{
    Object ele = head.ele;
    head = head.next;
    if(head==null)tail = null;
    count--;
    return ele;
}
public Object removeInBetween(int index)
{
    if(index<0||index>size())
    throw new IndexOutOfBoundsException();
    if(index==0)return removeFirst();
    Node temp = head;
    for(int i = 0;i<index;i++)temp = temp.next;
    Object ele = temp.next.ele;
```

```
        temp.next = temp.next.next;
        count--;
        return ele;
    }
    public Node getAdd(int index)
    {
        Node temp = head;
        for(int i = 0;i<index;i++)temp = temp.next;
        return temp;
    }
    public Object removeLast()
    {
        if(count==1)return removeFirst();
        Object ele = tail.ele;
        Node secondLast = getAdd(size()-2);
        tail = secondLast;
        tail.next = null;
        count--;
        return ele;
    }
}
public class LinkedListDriver
{
    public static void main(String[] args)
    {
        LinkedList a = new LinkedList();
        a.add(10);
        a.add(20);
        a.add(30);
```

```
        a.inserLast(40);
        a.display();
        a.insertInBetween(25, 2);
        System.out.println();
        a.removeInBetween(3);
        a.display();
    }
}
```

Q3 :- Double LinkedList.

```
public class Node {
        Node pre;
        Object ele;
        Node next;

        public Node(Node pre, Object ele, Node next) {
                this.pre = pre;
                this.ele = ele;
                this.next = next;
        }

        Node(Object ele) {
                this.ele = ele;
        }

}
```

==========

```
public class DoubleLinkedList {
        Node head;
```

```
int count = 0;
Node last;

public void insertFirst(Object ele) {
        Node node = new Node(ele);
        node.next = head;
        node.pre = null;
        if (head != null) {
                head.pre = node;
        }
        head = node;
        last = head;
        count += 1;
}

public int size() {
        return count;
}

public void display() {
        Node temp = head;
        while (temp != null) {
                System.out.print(temp.ele + "->");
                last = temp;
                temp = temp.next;
        }
        System.out.println("END");
}
```

```
public Object get(int index) {
	if (index < 0 || index > size())
		throw new IndexOutOfBoundsException();
	Node temp = head;
	for (int i = 0; i < index; i++) {
		temp = temp.next;
	}
	return temp.ele;
}

public Object removeLast() {
	if (count == 1) {
		return removeFirst();
	}
	Object ele = last.ele;
	Node secondLast = getAdd(size() - 2);
	last = secondLast;
	last.next = null;
	count--;
	return ele;
}

public Node getAdd(int index) {
	if (index < 0 || index > size())
		throw new IndexOutOfBoundsException();
	Node temp = head;
	for (int i = 0; i < index; i++) {
		temp = temp.next;
	}
```

```
        return temp;
}

public void addAfter(Object after, Object ele) {
        Node node = new Node(ele);
        Node p = find(after);
        if (p == null) {
                System.out.println("Ele Not Found");
                return;
        }
        node.next = p.next;
        p.next = node;
        node.pre = p;
        node.next.pre = node;

        count++;
}

public Node find(Object ele) {
        Node temp = head;
        while (temp != null) {
                if (temp.ele == ele)
                        return temp;
                temp = temp.next;
        }
        return null;
}

public void insertLast(Object ele) {
```

```
        if (head == null) {
                insertFirst(ele);
                return;
        }
        Node temp = head;
        while (temp.next != null) {
                temp = temp.next;
        }
        Node node = new Node(ele);
        node.next = null;
        temp.next = node;
        node.pre = temp;
        last = node;
        count++;

}

public Object removeFirst() {
        if (head != null) {
                Object ele = head.ele;
                head = head.next;
                if (head != null)
                        head.pre = null;
                count--;
                return ele;
        }
        return null;

}
```

```
public Object remove(int index) {
	if (index < 0 || index > size())
		throw new IndexOutOfBoundsException();
	if (index == 0) {
		return removeFirst();
	}
	if (index == size()) {
		return removeLast();
	}
	Node temp = head;
	for (int i = 1; i < index; i++) {
		temp = temp.next;
	}
	Object ele = temp.next.ele;
	temp.next = temp.next.next;
	temp.next.pre = temp;
	count--;
	return ele;
}

public void insert(Object ele, int index) {
	if (index < 0 || index > size())
		throw new IndexOutOfBoundsException();
	if (index == 0) {
		insertFirst(ele);
		return;
	}
	if (index == size()) {
```

```
                insertLast(ele);
                return;
        }
        Node temp = head;
        for (int i = 1; i < index; i++) {
                temp = temp.next;
        }
        Node node = new Node(ele);
        node.next = temp.next;
        node.pre = temp;
        temp.next = node;
        node.next.pre = node;
        count++;
}

public void add(Object ele) {
        if (count == 0) {
                insertFirst(ele);
                return;
        }
        Node temp = head;
        while (temp.next != null) {
                temp = temp.next;
        }
        temp.next = new Node(last, ele, null);
        last = temp.next;
        count++;
}
```

```
	public void reverse() {
		Node temp = last;
		while (temp != null) {
			System.out.print(temp.ele + "->");
			temp = temp.pre;
		}
		System.out.println("START");
	}

}
============================================================
public class UserProg {
	public static void main(String[] args) {
		DoubleLinkedList d = new DoubleLinkedList();
		d.add(10);
		d.add(20);
		d.add(30);
		d.add(40);
		d.add(50);
		d.insertFirst(5);
		d.insertLast(60);
		d.insert(25, 3);
		d.addAfter(30, 35);
		System.out.println(d.removeFirst());
		System.out.println(d.removeLast());
		System.out.println(d.remove(5));
		System.out.println(d.get(2));
		d.display();
		d.reverse();
```

```
		System.out.println("====================");
		System.out.println(d.size());

	}

}
```

Q4 :- Circular LinkedList.

```
public class Node {
	Object ele;
	Node next;

	Node(Object ele) {
		this.ele = ele;
	}

	public Node(Object ele, Node next) {
		this.ele = ele;
		this.next = next;
	}

}
```

```
public class CircularLinkedList {
	Node head;
	int count = 0;
	Node tail;

	public void insert(Object ele) {
```

```
        Node node = new Node(ele);
        if (head == null) {
                head = node;
                tail = node;
                count++;
                return;
        }
        node.next = head;
        tail.next = node;
        tail = node;
        count++;
}

public int size() {
        return count;
}

public void delete(Object ele) {
        if (head == null) {
                throw new NoSuchElementException();
        }
        if (head.ele == ele) {
                head = head.next;
                tail.next = head;
                count--;
                return;
        }
        Node temp = head;
        do {
```

```
                Node n = temp.next;
                if (n.ele == ele) {
                        temp.next = n.next;
                        count--;
                        break;
                }
                temp = temp.next;

        } while (temp != head);

    }

    public void display() {
        Node temp = head;
        do {
            if (temp != null)
                System.out.print(temp.ele + "->");
            temp = temp.next;

        } while (temp != head);
        System.out.println("START");
    }

}
```

```
public class UserProg {
    public static void main(String[] args) {
        CircularLinkedList c = new CircularLinkedList();
        c.insert(10);
```

```
		c.insert(20);
		c.insert(30);
		c.insert(40);
		c.insert(50);

		System.out.println(c.size());
		c.display();
		System.out.println("-----------");
		c.delete(50);
		c.display();
		System.out.println(c.size());
	}

}
```

Q5 :- Queue.

```
public class Node {
	Object ele;
	Node next;

	public Node(Object ele, Node next) {
		this.ele = ele;
		this.next = next;
	}

	Node(Object ele) {
		this.ele = ele;
	}
```

```
}
-------------------------
public class Queue {
	Node head;
	int count = 0;
	Node tail;

	public void add(Object ele) {
		Node node = new Node(ele);
		if (head == null) {
			head = node;
			tail = node;
			count++;
			return;
		}
		tail.next = node;
		tail = tail.next;
		count++;
	}

	public int size() {
		return count;

	}

	public boolean isEmpty() {
		return count == 0;
	}
```

```
	public Object peek() {
		if (isEmpty()) {
			System.out.println("No Ele Found");
			return null;
		}
		return head.ele;
	}

	public Object poll() {
		if (isEmpty()) {
			throw new NoSuchElementException();
		}
		Object ele = head.ele;
		head = head.next;
		count--;
		if (head == null)
			tail = null;
		return ele;
	}
}
------------------------------------------------------------
public class UserProg {
	public static void main(String[] args) {
		Queue q=new Queue();
		q.add(10);
		q.add(20);
		q.add(30);
		q.add(40);
		q.add(50);
```

```
            System.out.println(q.size());
            System.out.println(q.isEmpty());
            System.out.println(q.peek());

            while(!q.isEmpty())System.out.println(q.poll());
            System.out.println(q.isEmpty());
            System.out.println(q.size());
        }

}
```

Q6 Stack :-

```
public class Node {
        Object ele;
        Node next;

        public Node(Object ele, Node next) {
            this.ele = ele;
            this.next = next;
        }

        Node(Object ele) {
            this.ele = ele;
        }

}
```

```
public class Stack {
	Node first;
	int count = 0;

	public void push(Object ele) {
		if (first == null) {
			first = new Node(ele, null);
			count++;
			return;
		}
		first = new Node(ele, first);
		count++;
	}

	public int size() {
		return count;
	}

	public boolean isEmpty() {
		return count == 0;
	}

	public Object peek() {
		if (isEmpty()) {
			System.out.println("No Elements Found");
			return null;
		}
		return first.ele;
	}
```

```
	public Object pop() {
		if (isEmpty()) {
			throw new NoSuchElementException();
		}
		Object ele = first.ele;
		first = first.next;
		count--;
		return ele;
	}

}
-------------------------------------------------
public class UserProg {
	public static void main(String[] args) {
		Stack s = new Stack();
		s.push(10);
		s.push(20);
		s.push(30);
		s.push(40);
		s.push(50);

		System.out.println(s.isEmpty());
		System.out.println(s.size());
		System.out.println(s.peek());
		System.out.println("-----------------");
		while (!s.isEmpty())
			System.out.println(s.pop());
```

```
            System.out.println(s.size());
            System.out.println(s.isEmpty());


        }

}
```