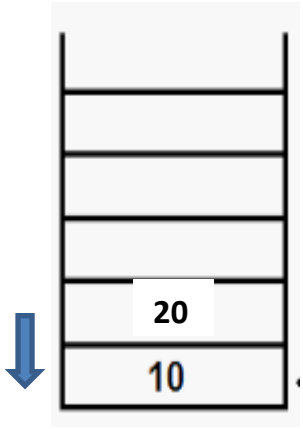
Stack- Linked list

Creating a stack with a linked list has the advantage of not worrying about the size as a linked list is dynamic. It is slight overkill as the only methods are again pop, push and peek.

This time **top** points to the current node. All we have to do is insert a node at the beginning to push and delete a node at the beginning to pop



←-top

In a normal linked list 10 would point to 20. However, for a stack it is the other way around

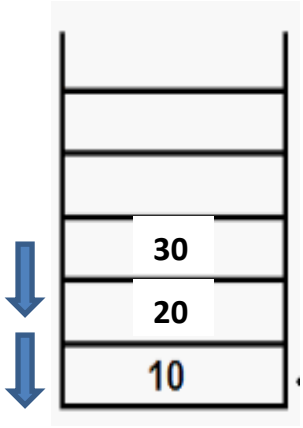
Push

temp = top

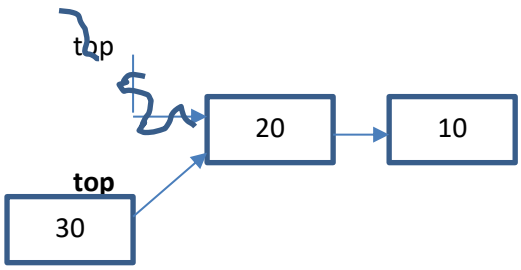
top = new node

top value = 30

top next = temp



←-top



```
public void push(int value) {
		Node temp= top;
		top= new Node();
		top.value = value;
		top.next =temp;
	}


public static void main(String args[])
	{
		LinkedListStack llstack=new LinkedListStack();
		lls.push(10);
		lls.push(20);
		lls.push(30);
```

POP (top will be given as a paremeter)

Check if list is empty

if not get top value

top becomes top.next

return top value

We get back to the first diagram.

```
public int pop() throws LinkedListEmptyException {
		if (top == null) {
			throw new LinkedListEmptyException();
		}
		int value = top.value;
		top = top.next;
		return value;
	}


System.out.println("Element removed from LinkedList:
"+llstack.pop());
```

In Java we can use the linked list java util which is unlikely to be asked for in Paper 2. There is no stack version of this. Note linked lists have more methods than this but for a stack we can only use these 3

```java
import java.util.LinkedList;

public class LinkedStack {

    private LinkedList<Employee> stack;

    public LinkedStack() {
        stack = new LinkedList<Employee>();
    }

    public void push(Employee employee) {
        stack.push(employee);
    }

    public Employee pop() {
        return stack.pop();
    }

    public Employee peek() {
        return stack.peek();
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }
```