```
import java.util.NoSuchElementException;

public class LinkedStack <T>  {
    private int N;
    private Node first;

    private class Node {
        private T item;
        private Node next;
    }

    public LinkedStack() {
        first = null;
        N = 0;
    }

    public boolean isEmpty() {
        return first == null;
    }

    public int size() {
        return N;
    }

    public void push(T item) {
        Node tempNode = first;
        first = new Node();
        first.item = item;
        first.next = tempNode;
        N++;
    }

    public T pop() {
        if (isEmpty()) throw new NoSuchElementException("The Stack is already
empty!");
        T item = first.item;
        first = first.next;
        N--;

        return item;
    }

    public T peek() {
        if (isEmpty()) throw new NoSuchElementException("The Stack is already
empty!");
        return first.item;
    }

    public static void main(String[] args) {

        LinkedStack<String> s = new LinkedStack<String>();
        s.push("1");
        s.push("2");

        System.out.println(s.pop());

    }
}
```

```
class MyStack {
    private int maxSize;
    private int[] data;
    private int top;

    public MyStack(int s) }
        maxSize = s;
        data = new int[maxSize];
        top = -1;
    }

    public void push(int v) {
        top++;
        data[top] = v;
    }

    public int pop() {
                top--;
        return data[top+1];
    }

    public long peek() {
        return data[top];
    }

    public boolean isEmpty() {
        if (top == -1) return true;
        return false;
    }
    public boolean isFull() {
        if (top == maxSize - 1) return true;
        return false;
    }
    public void clear() {
        top = -1;
    }
}
```

```
class MyQueue {
    private int maxSize;
    private int[] data;
    private int head;
    private int tail;
    private int size;

    MyQueue(int s) {
        maxSize = s;
        data = new int[maxSize];
        head = 0;
        tail = -1;
        size = 0;
    }

    public void clear() {
        head = 0;
        tail = -1;
        size = 0;
    }

    public void offer(int v) {
        if(tail == maxSize - 1)
            tail = -1;
        tail++;
        data[tail] = v;
        size++;
    }

    public int poll() {
        int temp = data[head];
        head++;
        if(head == maxSize)
            head = 0;
        size--;
        return temp;
    }

    public int peek() {
        return data[head];
    }
    public  boolean isEmpty() {
        if (size == 0) return true;
        return false;
    }

    public boolean isFull() {
        if (size == maxSize) return true;
        return false;
    }
}
```

```
public class Node {


    public String courseName;
    public int grade;

    public Node next;

    public Node(String courseName, int grade) {

        this.courseName = courseName;
        this.grade = grade;

    }

    public void display() {

        System.out.println(courseName + " - " + grade);

    }

    public String toString() {

        return courseName;

    }

    public static void main(String[] args) {

        LinkedList l = new LinkedList();

        l.addFirst("Calculus",2);
        l.addFirst("Geometry",2);
        l.addFirst("Algebra",3);

        l.display();

        l.removeFirst();

        l.display();

        l.deleteNodeByCourseName("Algebra");

        l.display();

    }

}
```

```
package treebylinkedlist;
import java.util.LinkedList;

public class TreeByLinkedList<T> {

public static class TreeNode<T>{
    private T value;
    private boolean hasParent;
    private LinkedList<TreeNode<T>> children;
    public TreeNode(T value){
        if (value==null){
            throw new IllegalArgumentException("Cannot insert null value");

        }
        this.value = value;
        this.children = new LinkedList<TreeNode<T>>();

    }
    public T getValue(){
        return this.value;

    }
    public void setValue(T value){
        this.value = value;

    }
    public void addChild(TreeNode<T> child){
        if(child == null){
            throw new IllegalArgumentException(
                    "Cannot insert null value");

    }
        if (child.hasParent){
            throw new IllegalArgumentException("The node already has a parent");

        }
        child.hasParent = true;
        this.children.add(child);

}
    public TreeNode<T> getChild(int index){
        return this.children.get(index);

    }
    public int getChildrenCount(){
        return this.children.size();

    }
}

private TreeNode<T> root;
public TreeByLinkedList(T value){
    if(value ==null){
        throw new IllegalArgumentException("Cannot insert null value");

    }

    this.root = new TreeNode<T>(value);
}
public TreeByLinkedList( T value, TreeByLinkedList<T> ...children){
    this(value);
    for(TreeByLinkedList<T> child:children){
        this.root.addChild(child.root);
    }
}
public TreeNode<T> getRoot(){
    return this.root;
        return this.root.children;
```

```
    }
    return new LinkedList<TreeNode<T>>();
}
private void printDFS(TreeNode<T> root, String spaces){
    if(this.root==null) return;
    System.out.println(spaces + root.getValue() );
    TreeNode<T> child = null;
    for(int i = 0;i<root.getChildrenCount();i++){
        child = root.getChild(i);
        printDFS(child,spaces + " ");

    .}

}
public void printDFS(){
    this.printDFS(this.root,new String());
}
    public static void main(String[] args) {

    }
}
```

```
class LinkedList {

    public Node firstNode;

    LinkedList() {

        firstNode = null;

    }

    public boolean isEmpty() {

        return(firstNode == null);

    }

    public void addFirst(String courseName, int grade) {

        Node tempNode = new Node(courseName, grade);

        tempNode.next = firstNode;

        firstNode = tempNode;

    }

    public Node removeFirst() {

        Node tempNode = firstNode;

        if(!isEmpty()) {

            firstNode = firstNode.next;

        } else {

            System.out.println("The LinkedList is already empty!");

        }

        return tempNode;

    }

    public void display() {

        Node tempNode = firstNode;

        while(tempNode != null){

            tempNode.display();

            tempNode = tempNode.next;

            System.out.println();

        }

    }
```

```
public Node searchByCourseName(String courseName) {

    Node tempNode = firstNode;

    if(!isEmpty()) {

        while(!(tempNode.courseName.equals(courseName))) {


            if(tempNode.next == null) {

                return null;

            } else {

                tempNode = tempNode.next;

            }

        }

    } else {

        System.out.println("The LinkedList is empty!");

    }

    return tempNode;

}

public Node deleteNodeByCourseName(String courseName) {

    Node currentNode = firstNode;
    Node previousNode = firstNode;

    while(!(currentNode.courseName.equals(courseName))) {

        if(currentNode.next == null) {

            return null;

        } else {

            previousNode = currentNode;

            currentNode = currentNode.next;

        }

    }

    if(currentNode == firstNode) {

        firstNode = firstNode.next;

    } else {

        previousNode.next = currentNode.next;
return currentNode;
    }

}
```

```
LinkedQueue<String> q = new
LinkedQueue<String>();
q.enqueue("1");
System.out.println(q.dequeue());
}
}
```

```
import java.util.NoSuchElementException;

public class LinkedQueue<T> {
    private int N;
    private Node first;
    private Node last;


    private class Node {
        private T item;
        private Node next;
    }

    public LinkedQueue() {
        first = null;
        last  = null;
        N = 0;
    }

    public boolean isEmpty() {
        return first == null;
    }

    public int size() {
        return N;
    }

    public T peek() {
        if (isEmpty()) throw new NoSuchElementException("The Queue is already
empty!");
        return first.item;
    }

    public void enqueue(T item) {
        Node tempLast = last;
        last = new Node();
        last.item = item;
        last.next = null;
        if (isEmpty()) {
            first = last;
        }
        else {
            tempLast.next = last;
        }
        N++;
    }

    public T dequeue() {
        if (isEmpty()) throw new NoSuchElementException("The Queue is already
empty!");
        T item = first.item;
        first = first.next;
        N--;
        if (isEmpty()) {
            last = null;
        }
        return item;
    }

    public static void main(String[] args) {
```