Linked Lists:

What is a linked list? What items is it always made up of?

Linked list - linear data structure where each element (node) is a separate object.

Be able to insert new items into a linked list

```
public class Links      {

        public static void main(String[] args)  {

                Node pos1 = null;
                Node pos2 = null;

                pos1 = new Node(new Integer(13));
                pos1.setNext(new Node(new Integer(15), null));

                pos2 = new Node(new Integer(11), null);
                pos2.setNext(pos1);

                printList(pos2);
        }
```

Node node = new Node("B");
Node front = new Node("A");
front.setNext(node);
node = front;

Be able to print out and count the number of items in a linked list

```
        private static void printList(Node head)        {
                if (head != null)       {
                        System.out.println(head.getItem());
                        printList(head.getNext());
                }
        }
```

Be able to delete items from a linked list

```
        // Recursive method
        private static int count (Node head)    {
                int count = 0;
                if(head == null)        {
                        return 0;
                }
                else
                        return 1 + count(head.getNext());
        }
```

node.setNext(node.getNext()...); // … means add as many "getNext" to reach node position that you want to delete.

Be able to compare and contrast arrays and linked lists

Arrays	Linked Lists
Applications with **constant** size	Applications with **varying** sizes
Easy to **find** nth element	**Search** starts from beginning for nth element
Hard to insert/delete from first slot	**Easy** to insert/delete from first slot
Easy to insert/delete from last slot	**Hard** to insert/delete from last slot since need to find RAM address
Easy to apply sorting algorithms to	

How to instantiate:
Array a[] = new Array[];
Node pos1 = null;

How to delete:
array = ArrayUtils.removeElement(array, element);
Node pos1.setItem(null);

The role of the copy-constructor clone method; the dangers of not including it within a class that manipulates dynamic data; the differences between a deep copy (clone) and a shallow copy (alias)

Copy constructor - a new instance method that returns a copy of a class.
Dangers of not including - you don't want to delete the original copy.

Shallow copy - duplicate as little as possible. A copy of collection structure but not elements.
The 2 copies share the elements.
Deep copy - duplicate everything.

Example:

```
Public static void colors(Stack s)      {       // Shallow Copy
        Stack a = s;

Public static void colors(Stack s)      {       // Deep Copy
        Stack temp = (Stack) s.clone();
        temp.push();
        temp.pop();
}
```

Know what lines of code like the following represent:

- **p.getNext().getNext() == q**
- **p.getNext().setItem(q.getItem())**
- **p.getNext() == q**

Be able to read and understand if true or false.

What is a "memory leak"? How is garbage collection of memory handled in Java?

Memory leak - failure to release memory. It is actually when you delete a node.
Garbage collection memory - automatically deleted in Java. RAM pointer is now not there, ready to be wiped over.

Stacks and Queues:

Definition of both and what distinguishes them

Stack - a LIFO object
Queue - a FIFO object

LIFO vs. FIFO

LIFO - Last in, First out -Queue
FIFO - First in, First out -Stack

Be able to name some applications of a stack and queue

Applications of Stack: Recursive function
Calling function
Expression evaluation
Expression conversion
Towers of Hanoi
Baggage Loading and Off Loading on airplanes

Applications of Queue: Storing pending work
Scheduling algorithms
Any application where data is processed in order of arrival

Be able to trace through a program which uses the methods of a stack or queue class

Linked List: setNext()
getNext()

Stack: push() // moves node to top of stack
pop() // deletes top node and returns the object
top() // gets top node

Queue: enqueue() // adds node to back of queue
dequeue() // deletes node from the front

Be able to write class methods for a driver program which needs to use the methods of a stack or queue class

"For example, in Lab 12, you wrote a class method called letterCount which using only the methods of the Queue class like front(), dequeue(), isEmpty(). You then wrote the same method as an instance method for a Queue built as a circular array." Digh

What are the two main ways a stack class can be implemented

Implemented using a linked list or a deep copy

Be able to write instance methods for a linked list implementation of a stack (see findMax instance methods from Lab11)

```
public void main (String[] args)        {
        Stack stack1 = new Stack();
        stack1.push("Andy");
        stack1.push("Allison");
        System.out.print(findMax(stack1));
}

public String findMax() throws CloneNotSupportedException        {
        Stack temp = (Stack) this.clone(); // deep copy
        String max = (String) temp.top(); // pushes temp's top value to the max string
        while(!temp.isEmpty())        {
                String curr = (String) temp.top(); //makes curr is shallow copy
                if(curr.compareTo(max) > 0)
                        max = curr;
                         temp.pop();
        }
        return max;
}

public static String findMax(Stack stack1) throws CloneNotSupportedException   {
        Stack temp = (Stack) stack1.clone();
        String max = (String) temp.top();
        temp.pop();
        while (! temp.isEmpty())      {
                String current = (String) temp.pop();
                if (current.compareTo(max) > 0)   {
                        max = current;
                        temp.pop();
                }
        }
        return max;
}
```

What are the two main ways queue class can be implemented? How does a circular array work? Be able to trace through some code which used a circular array.

Queue class implementation - circular linked list or circular array

Queue q = new Queue();
q.**en**queue("C");
q.**en**queue("B");
q.dequeue();

Circular - works because the last node is set to the beginning node
node.getNext(+ however many to reach the last node).**setNext(node)**