1. Behavior of Stacks and Queues (20 Points)
   a. Show the lines printed by the code below. Draw pictures of the data structure states to be eligible for partial credit. Note: There is a summary of the API for the java.util.Stack class and the java.util.Queue interface on the next to last page of this exam.

   ```java
   import java.util.Stack;
   import java.util.Queue;
   import java.util.LinkedList;

   Stack<String> myStack = new Stack<String>();
   Queue<String> myQueue = new LinkedList<String>();

   myStack.push("World");
   myStack.push("Hello");
   myStack.push("Salve");
   myStack.push("Mondo");

   myQueue.offer(myStack.peek());
   myQueue.offer(myStack.pop());
   myQueue.offer(myStack.pop());
   myQueue.offer(myStack.peek());

   myQueue.offer(myStack.peek());

   while (!myStack.isEmpty())
       System.out.println(myStack.pop());

   while (!myQueue.isEmpty())
       System.out.println(myQueue.poll());
   ```
2. (30 Points) Performance with Recursion
Study the recursive methods below and indicate the performance in big-O notation.
Explain your answers.

a. Based on the number n of elements in the linked list
   \[ O(\quad) \]
   ```java
   public void mysteryPrint(LinearNode<String> next)
   {
     if (next != null) {
       mysteryPrint(next.getNext());
       System.out.println(next.getElement());
     }
   }
   ```

b. Based on the number n of elements in the linked list
   \[ O(\quad) \]
   ```java
   public void mysteryPrint(LinearNode<String> next)
   {
     if (next != null) {
       System.out.println(next.getElement());
       mysteryPrint(next.getNext());
     }
   }
   ```

c. Based on the number n
   \[ O(\quad) \]
   ```java
   public int calculate(int m, int n)
   {
     if (m < n)
     { n = calculate(m + 1, n / 10); return m * n; }
   }
   ```
3. Class/Interface Hierarchy (20 Points)

For each of the following, indicate if it is (a) valid Java statement(s) or not and explain. (Assume that we have included any needed import statements for the java.util classes.)

<table>
<thead>
<tr>
<th>Statement(s)</th>
<th>Valid(Y/N)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Stack&lt;String&gt; myStack = new Vector&lt;String&gt;();</td>
<td>______</td>
<td>___________________</td>
</tr>
<tr>
<td>b. Stack&lt;Vector&gt; myStack = new Stack&lt;Vector&gt;();</td>
<td>______</td>
<td>___________________</td>
</tr>
<tr>
<td>c. Queue&lt;boolean&gt; myQueue = new LinkedList&lt;boolean&gt;();</td>
<td>______</td>
<td>___________________</td>
</tr>
<tr>
<td>d. OrderedListADT&lt;String&gt; myList = new . . . ; myList.addAfter(&quot;Hello&quot;, &quot;World&quot;);</td>
<td>______</td>
<td>___________________</td>
</tr>
<tr>
<td>e. public class SomeThing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ public static void main(String [] args)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{ Integer [] newArgs = new Integer[10]; // some valid code here . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>main(newArgs);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>______</td>
<td>___________________</td>
</tr>
</tbody>
</table>
4. Stack Application (30 Points)
In each plastic container of Pez candies (a stack), the colors are stored in random order. Your younger brother only likes the red ones, so he takes out all the candies one at a time and eats the red ones. He keeps the others in order and returns them to the container in exactly the original order (minus the red ones). Use the java.util.Stack class to simulate this process.

```java
import java.util.Stack;
public class Pez {
    public static void main(String[] args) {
        // setup the original container of candies
        Stack<Candy> container = new Stack<Candy>();
        container.push(new Candy("Red"));
        container.push(new Candy("Green"));
        // additional initialization not shown here

        // write your code here to remove all the candies from
        // the container, "eat" the red ones, and return all the
        // other candies to the container in the original order
    }
}

public class Candy {
    private String color;
    public Candy(String color) {
        this.color = color;
    }
    public String getColor() {
        return color;
    }
}
```
The only methods in the java.util.Stack class you should use:

• push is done using:
  T push(T element)  // pushes and returns the element T

• pop is done using:
  T pop()  // pops and removes the element T

• peek is done using:
  T peek()  // returns an alias of the element T

• isEmpty is done using:
  boolean isEmpty()  // returns true if stack is empty

The methods in the java.util.Queue interface:

• Enqueue is done using:
  boolean offer(T element)  // returns false if full

• Dequeue is done using either:
  T poll()  // returns null value if empty
  T remove()  // throws an exception if empty

• Peek is done using either:
  T peek()  // returns null value if empty
  T element()  // throws an exception if empty

• isEmpty is done using:
  boolean isEmpty()  // returns true if queue is empty
For reference in Problem 2:

```java
public class LinearNode<T> {
    private LinearNode<T> next;
    private T element;

    public LinearNode()  // create an empty node
    {
        next = null;
        element = null;
    }
    public LinearNode(T element)
    {
        next = null;
        this.element = element;
    }
    public LinearNode<T> getNext()
    {
        return next;
    }
    public void setNext(LinearNode<T> next)
    {
        this.next = next;
    }
    public T getElement()
    {
        return element;
    }
    public void setElement(T element)
    {
        this.element = element;
    }
} // end class LinearNode<T>
```
Answer Key:
1. Behavior of Stacks and Queues
   > java StackQueue
   Stack contents are:
   Hello
   World
   Queue contents are:
   Mondo
   Mondo
   Salve
   Hello
   Hello
   >

2. Performance with Recursion
   a. $O(n)$  The recursive call is made once per element in the list

   b. $O(n)$  The recursive call is still made once per element in the list.
      This method will merely print the list contents in the reverse order of a.
      BTW: This method exhibits tail recursion.

   c. $O(\log n)$  The recursive call is made as the value of $n$ is decreased by division by ten.
      Hence, the number of times $m$ is incremented by 1 to exceed the value of $n$ is $\log_{10} n$.
      BTW: This method does not calculate any significant function. It is just a recursion.
3. Class/Interface Hierarchy

<table>
<thead>
<tr>
<th>Statement(s)</th>
<th>Valid(Y/N)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Stack&lt;String&gt; myStack = new Vector&lt;String&gt;();</td>
<td>No</td>
<td>Incompatible types - Stack is a subclass of Vector</td>
</tr>
<tr>
<td>b. Stack&lt;Vector&gt; myStack = new Stack&lt;Vector&gt;();</td>
<td>Yes</td>
<td>A stack of vector class objects is valid</td>
</tr>
<tr>
<td>c. Queue&lt;boolean&gt; myQueue = new LinkedList&lt;boolean&gt;();</td>
<td>No</td>
<td>boolean is a primitive data type not a reference data type</td>
</tr>
<tr>
<td>d. OrderedListADT&lt;String&gt; myList = new . . .; myList.addAfter(&quot;Hello&quot;, &quot;World&quot;);</td>
<td>No</td>
<td>addAfter method is not valid in OrderedList&lt;ADT&gt;</td>
</tr>
<tr>
<td>e. public class SomeThing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public static void main(String [] args)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer [] newArgs = new Integer[10];</td>
<td></td>
<td></td>
</tr>
<tr>
<td>// some valid code here . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>main(newArgs);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>The main method must be called with a String [] parameter</td>
<td></td>
</tr>
</tbody>
</table>
4. Stack Application
import java.util.Stack;
public class Pez {
    public static void main(String[] args) {
        // setup the original container of candies
        Stack<Candy> container = new Stack<Candy>();
        container.push(new Candy("Red"));
        container.push(new Candy("Green"));
        container.push(new Candy("Yellow"));
        container.push(new Candy("Red"));
        container.push(new Candy("Blue"));

        // write your code here to remove all the candies from
        // the container, "eat" the red ones, and return all the
        // other candies to the container in the original order
        Stack<Candy> myStack = new Stack<Candy>();

        while(!container.isEmpty()) {
            Candy myCandy = container.pop();
            if (!myCandy.getColor().equals("Red"))
                myStack.push(myCandy);
        }

        while(!myStack.isEmpty())
            container.push(myStack.pop());

        while(!container.isEmpty())
            System.out.println(container.pop().getColor());
    }
}

Sample Output:
> java Pez
Blue
Yellow
Green
>