Practice Questions

a) A static method within a class can be invoked without having to instantiate an object of that class.

   True    False

b) A Java interface should never contain a constructor.

   True    False

c) Generic types are specified at compile time.

   True    False

d) A Java interface never contains an instance variable declaration.

   True    False

e) Polymorphism allows a reference variable to point to objects of unrelated types.

   True    False

f) A new class $X$ derived from a class $Y$ establishes an is-a relationship between classes $X$ and $Y$.

   True    False

g) Any method in a Java class $X$ can call public and private methods from $X$.

   True    False

h) An interface can be used to create objects even when it is not implemented by any class.

   True    False

i) The reserved word super can be used in a class to refer to its parent class.

   True    False

j) The objects in a Java program are created when the program is being executed, as opposed to when it is compiled.

   True    False
Compute the time complexity of the following algorithms.

I did not have time to write explanations of how the time complexities of these algorithms are computed. In the midterm exam you must explain how you computed the time complexity of each algorithm.

1. $O(n)$
   ```java
   public static void print1( int[] list, int n ) {
       for (int i = 0; i < n; i++) {
           System.out.println(list[i]);
           n = n - 1;
       }
   }
   ```

2. $O(n^2)$
   ```java
   int k = 0;
   for (int i = n; i >= 1; i--) {
       for (int j = n; j >= 1; j--) {
           k = i * j;
           System.out.println(k);
       }
   }
   ```

3. $O(\log n)$
   ```java
   int i = n;
   while (i > 1) {
       System.out.println(i);
       i = i / 2;
   }
   ```

4. $O(n^2)$
   ```java
   for (int i = 0; i < n; i++) {
       for (int j = 0; j < n; j++)
           for (int k = 0; k < n; k++)
               System.out.println(i,j,k);
       ++i;
   }
   ```

5. $O(1)$
   ```java
   int sum = 0;
   for (int k = 0; k < n; k++)
       if (sum < n) break;
       else sum = sum + table[n-k-1][n-k-1];
   return sum
   ```
The following main program uses classes `LinkedStack.java` and `LinkedQueue.java` to create a stack of queues, in which `Integer` objects are enqueued:

```java
public static void main(String args[]) throws Exception {
    final int MAX = 3;
    LinkedQueue<Integer> queue;
    LinkedStack<LinkedQueue<Integer>> stack;
    stack = new LinkedStack<LinkedQueue<Integer>>();
    for (int j = 0; j < MAX; j++) {
        queue = new LinkedQueue<Integer>();
        for (int k = j; k < MAX; k++) {
            queue.enqueue(new Integer(j + k + 1));
        }
        stack.push(queue);
    }
    System.out.println("Size of stack: \n" + stack.size() + \n);
    int count = 1;
    while (!stack.isEmpty()) {
        queue = stack.pop();
        System.out.println("Queue " + count);
        while (!queue.isEmpty())
            System.out.println(queue.dequeue().toString());
        ++count;
    }
}
```

Show the output produced by running this main method.

```
Size of stack:
3
Queue 1
5
Queue 2
3
4
Queue 3
1
2
3
```

7. Consider a circular array implementation of the Queue ADT. Suppose that the array used to store the elements of the queue has length 4. Suppose that, into an initially empty queue, we enqueue the three integer values 3, 5, and 7, in this order. Then, two objects
are dequeued, and three more integer values 8, 6, and 9 are enqueued. Draw the resulting array. Indicate where the front and the rear elements of the queue are.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rear</td>
<td></td>
<td>front</td>
<td></td>
</tr>
</tbody>
</table>

8. Give the output generated by each of the following Java code segments:

- **LinkedQueue<Integer> a = new LinkedQueue<Integer>();**
  ```java
  final int MAX = 5;
  for (int i = 0; i < MAX; i++) a.enqueue(MAX - i);
  System.out.println(a.dequeue());
  Answer: 5
  ```

- **LinkedQueue<Integer> a = new LinkedQueue<Integer>();**
  **LinkedStack<Integer> b = new LinkedStack<Integer>();**
  ```java
  final int MAX = 5;
  for (int i = 0; i < MAX; i++) {
    a.enqueue(i);
    b.push(i);
  }
  for(int i = 0; i < MAX; i++)
    System.out.println(a.dequeue() + b.pop() + ", ");
  Answer: 4, 4, 4, 4, 4,
  ```

- **LinearNode<Integer> a = new LinearNode<Integer>(1);**
  **LinearNode<Integer> b = new LinearNode<Integer>(2);**
  ```java
  a.setNext(b);
  b.setNext(a);
  System.out.println((a.getNext().getNext()).getElement());
  Answer: 1
  ```

- **public class BuildLinkedList {**
  ```java
  public static void main(String[] args) {
    final int MAX = 5;
    LinearNode<Integer> head = null;
    LinearNode<Integer> intNode;
    for (int i = MAX; i >= 1; i--){
      intNode = new LinearNode<Integer>(MAX - i + 1);
      intNode.setNext(head);
      head = intNode;
    }
  }
  ```
LinearNode<Integer> current = head;
for (int i = 1; i <= MAX; i++) {
    System.out.print(current.getElement() + " , ");
    current = current.getNext();
}

Answer: 5, 4, 3, 2, 1,

9. Consider a singly linked list where each node stores an integer value. Write an algorithm `numLarger(front, x)` that receives as input a reference to the first node of the list and an integer value x and it returns the number of nodes of the list that store values larger than x. Given a node p, method `p.getValue()` returns the value stored in p and method `p.getNext()` returns the next node in the list after p.

   Algorithm `numLarger(front, x)` {
       current = front
       num = 0;
       while current != null do {
           if current.getValue() > x then num++
           current = current.getNext();
       }
       return num;
   }

10. Write an algorithm `attach(queue1, queue2)` that attaches the second queue to the first one, so the resulting queue has the same elements of queue1, in the same order in which they appear in queue1, followed by the elements in queue2, in reverse order in which they appear in queue2. You can use an auxiliary stack `s`. You cannot use any other data structures. The algorithm must return the resulting queue.

   Algorithm `attach(queue1, queue2)` {
       s = new Stack()
       while !queue2.isEmpty() do
           s.push(queue2.dequeue())
       while !s.isEmpty() do
           queue1.enqueue(s.pop())
       return queue1
   }