1. (25 points) Consider the following Java classes:

```java
1. class ParentCo {
2.  protected double netWorth;
3.  protected String orgName;
4.  ParentCo(double dollars, String name) {
5.    netWorth = dollars;
6.    orgName = name;
7.  }
8.  public void printMe() {
9.    System.out.println("Current value of "+orgName+": "+netWorth); }
10. }
11. class SpinOff extends ParentCo {
12.  SpinOff(double p, String n) {
13.    super(p,n);
14.  }
15.  public void printMe() {
16.    super.printMe();
17.  }
18. }
19. class SmallBusiness extends SpinOff {
20.  protected int count = 0;
21.  SmallBusiness(double val, String name) {
22.    super(val, name);
23.  }
24.  public void printMe() {
25.    if (count > 0) {
26.      System.out.println(orgName+" first year profits: "+netWorth);
27.      super.printMe();
28.    } else {
29.      System.out.println(orgName+" first year profits not in yet.");
30.      count++;
31.    }
32.  }
33. }
34. }
35. public class TestOrg {
36.  public static void main(String[] args) {
37.    SpinOff mid = new SpinOff(250000.0, "My Cat’s Choice");
38.    ParentCo netscape = new SmallBusiness(600000.0,"Moo’s Icecream");
39.    Object obj = new ParentCo(2000000.0, "Scuba Unlimited");
40.    ParentCo startup = new SmallBusiness(80000.0, "Organics Inc.");
41.    mid.printMe();
42.    netscape.printMe();
43.    ((ParentCo) obj).printMe();
44.    obj = startup;
45.    ((SpinOff) obj).printMe();
46.    startup = netscape;
47.    startup.printMe();
48.    ((ParentCo) obj).printMe();
49.  }
50. }
```

(a) (10 pts.) What is the output of the TestOrg application? (Assume there are no syntax errors.)

```
Current value of My Cat’s Choice: 250000.0
Moo’s Icecream first year profits not in yet.
Current value of Scuba Unlimited: 2000000.0
Organics Inc. first year profits not in yet.
Moo’s Icecream first year profits: 600000.0
Current value of Moo’s Icecream: 600000.0
Organics Inc. first year profits: 80000.0
Current value of Organics Inc.: 80000.0
```

(b) (10 pts.) Draw a class inheritance diagram to show the relationship between classes in the code above (show all inheritance relationships that apply).

Will show result in class on Monday, April 11th.
(cont. from problem 1)

(c) (1 pt.) True or False: After line 36 is executed, there are 4 objects on the heap.
False. There are only 2 because obj is set to startup and then startup is set to netscape. So all three references obj, startup, and netscape reference the same object and mid references a different object on the heap.

(d) (1 pt.) True or False: The classes SpinOff and SmallBusiness overload the method printMe in the class ParentCo.
False. They override the printMe method in the class ParentCo.

(e) (1 pt.) True or False: The class SmallBusiness is a derived class of the classes SpinOff and ParentCo.
True

(f) (1 pt.) True or False: Line 35 could have been written “obj.printMe();” and the program would still compile with no syntax errors. Explain your answer.
False. It would create a syntax error because the object class does not have a printMe() method.

(g) (1 pt.) Short answer: Describe the purpose of the statement on line 21.
Calling the printMe method of the superclass.

2. (5 points) Consider a data set whose size may change during a program execution. Which is a more efficient container for this data set, in terms of space usage, the array or the singly-linked list implementation? If there is no difference, state “No difference”. Justify your answer for full credit. The singly-linked list would be more efficient because the array could be only partially full.

3. (15 pts.) Let A be an array containing n integers. Consider the following pseudocode:

1. for i := 0 to n-1 do
2. for j := n-1 down to (i + 1) do

(a) (1 pt.) State the approximate number of primitive operations performed for line 1: n
(b) (1 pt.) Suppose you could execute line 2 outside of the for loop defined in line 1. State the approximate number of primitive operations performed for line 2 if i = 0: n
(c) (1 pt.) Suppose you could execute line 3 outside of either for loop. State the approximate number of primitive operations performed for line 3: one
(d) (2 pts.) Approximate the running time of the entire block of pseudocode using big-oh notation: O(n^2)
(e) (10 pts.) If you implemented the above algorithm using the following Java code, what would be the output?

```java
public class TestArray {
    public static void main (String[] args) {
        int[] A = {1, 1, 1, 1, 1, 1, 1, 1, 1};
        for (int i = 0; i < A.length; i++) {
            for (int j = A.length - 1; j > i; j--) {
            }
        }
        System.out.println("Position "+ i +" = "+ A[i]);
    }
}
```

Position 0 = 9
Position 1 = 8
4. (5 pts.) Write numbers to the right of the following functions to indicate the slowest (1) to fastest (6) growing as \( n \), the size of the data set, increases to infinity.

\[
\begin{align*}
\text{nlog}_2 n & \quad 3 & \quad n^3 & \quad 5 & \quad 100n & \quad 2 & \quad 4^n & \quad 6 & \quad n^2 & \quad 4 & \quad 2^{100} & \quad 1 \\
\end{align*}
\]

5. (5 pts.) Explain the difference between **overriding** and **overloading**.

**Overriding** occurs when a subclass has a method with the same name as the superclass. **Overloading** occurs when there is more than 1 method in the class with the same name but different numbers or types of parameters.

6. (20 pts.) The following questions refer to exception handling. Assume there are no syntax errors in any of the code fragments (A through D).

(A) (5 pts.) What is the output of the following code fragment?

```java
try {
    int num1 = Integer.parseInt("14");
    System.out.println("Okay 1");
    int num2 = Integer.parseInt("one");
    System.out.println("Okay 2");
} catch (NumberFormatException nfe) {
    System.out.println("Error");
}
```

**Okay 1**

**Error**

(B) (5 pts.) What is the output of the following code fragment?

```java
try {
    int number = Integer.parseInt("-30");
    if (number < 0) {
        throw new Exception("No negative numbers please.");
    }
} catch (Exception e) {
    System.out.println("Error: "+ e.getMessage());
} catch (NumberFormatException nfe) {
    System.out.println("Cannot convert to int");
}
```

**Error: No negative numbers please.**

(C) (5 pts.) Consider the following code fragment:

```java
try {
    int number = Integer.parseInt(JOptionPane.showInputDialog(null, "Input:"));
    if (number != 0) {
        throw new Exception("Number not zero.");
    }
} catch (NumberFormatException nfe) {
    System.out.println("Cannot convert to int");
} catch (Exception e) {
    System.out.println("Error: "+ e.getMessage());
}
```

The output “Input:” would be written in a JOptionPane in all cases. Determine the output when the input is:
a) -1       Number not zero.

b) 0

c) 12XY     Cannot convert to int

(D) (5 pts.) Short answer: What, if anything, is wrong with the following code?

```java
public void check (int num) {
    if (num < 0) {
        throw new Exception();
    }
}
```

The code has an error:
unreported exception java.lang.Exception; must be caught or declared to be thrown

7. (5 pts.) The following method is intended to find the last node of a singly-linked list and return a pointer to that node. This code throws an exception when it is executed. Show how the method could be fixed so that it does not throw an exception.

Assume that when the method is called on a non-empty list, head is a pointer to the first node in the list, and that when the method is called on an empty list, head is a reference to the null object. Assume there are no syntax errors in this method.

```java
public SinglyLinkedListNode lastNode() {
    SinglyLinkedListNode n = head;
    if (n != null) {
        do {
            n = n.getNext();
        } while (n.getNext() != null);
    }
    return n;
}
```

The code throws an exception because n might be null before it gets to the n.getNext() on the last line. To fix the code, you could write it as shown below:

```java
public SinglyLinkedListNode lastNode() {
    SinglyLinkedListNode n = head;
    if (n != null) {
        while (n.getNext() != null) {
            n = n.getNext();
        }
    }
    return n;
}
```

8. (5 points) At times pseudocode can look more like a natural language than a programming language. Consider the following “English-like” pseudocode:
declare an empty queue of characters
declare an empty stack of characters
while ( there are more characters to read )
    read a character
    enqueue the character on the queue
while ( the queue is not empty )
    dequeue a character from the queue
    push the character on the stack
while ( the stack is not empty )
    pop a character off the stack
    print the character to the screen

What is written to the screen for the input “RoseAnne Rosannadanna”?
“annadannasor ennAesoR”

9. (10 pts.) Suppose the ListStack and ListQueue classes have been written so that they conform to the stack and queue abstract data type specifications, respectively. For which values of n would the following method return true and for which values would it return false?

```java
public static boolean mysteryMethod(int n) {
    ListStack stack = new ListStack();
    ListQueue queue = new ListQueue();
    for (int i = 0; i < n; i++) {
        stack.push(new Integer(i));
        queue.enqueue(new Integer(i));
    }
    if (n % 2 == 0) {
        for (int i = 0; i < n/2; i++) {
            stack.pop();
            queue.dequeue();
        }
    } else {
        for (int i = 0; i < (n - 1)/2; i++) {
            stack.pop();
            queue.dequeue();
        }
    }
    return (stack.toString().equals(queue.toString()));
}
```

This method would return false for any n because the values on the stack would be the lower half and the ones on the queue would be the upper half.

10. (10 points) Write pseudocode to remove the penultimate (second to the last) node in a singly linked list of INodes. Assume there is a reference to the head of the list that is null if the list is empty and that points to the first node in the list if the list is non-empty. Assume there is no reference to the tail of the list.

Hint: Think about how many nodes a list has to have in order to remove the penultimate node.

The list must have at least 2 nodes to have a second to the last node. As a result the pseudocode for the method is:

1. if ((head == null) || (head.next == null))
2. System.out.println("penultimate does not exist.");
3. INode current = head.next;
4. INode previous = head;
5. INode prevprevious = head;
6. if (current.next == null) {
7.   head = current;  \ head is penultimate, set head to current
8. } else {
9.   while (current.next != null) {
10.      prevprevious = previous
11.      previous = current
12.      current = current.next }
13. prevprevious.next = current; }