Sample Problem: Write code

The Confusion class (below) implements both Comparable and OppComparable. It is incomplete because it does not implement Comparable. Do something to complete Confusion.

```
public interface OppComparable<T> {
    /* Returns a negative integer, zero, or a positive integer
     * as this object is greater than, equal to, or less than
     * the specified object (o)
     */
    public int compareTo(T t);

}

public interface Comparable<T> {
    /* Returns a negative integer, zero, or a positive integer
     * as this object is less than, equal to, or greater than
     * the specified object (o)
     */
    int compareTo(T o)
}


public class Confusion<T> implements Comparable<T>,
OppComparable<T> {

}
```

Sample Problem: Complexity and Data Structures

Fill in every cell of the following table with big O worst case time estimated for each cell. In the first column, just indicate Y or N. In the final column give the Big O space usage.

	Size Adjust-able Y/N	Read Time	Replace Time	Add time	Remove Time	Size Time	Space
Array							
ArrayList							
Singly linked list							
Doubly linked list							
Stack Array-based							
Queue link-list based							

	Size Adjust-able Y/N	Read Time	Replace Time	Add time	Remove Time	Size Time	Space
Array	N	1	1	1	1	1	N (see comment)
ArrayList	Y	1	1	N	N	1	N
Singly linked list	Y	1 (or N if have to find first)	1 (or N is you have to find first)	1	1 (or N is you have to find first)	1 (if list tracks size, else N)	N
Doubly linked list	Y	1 (or N if have to find first)	1 (or N is you have to find first)	1	1 (or N is you have to find first)	1 (if list tracks size, else N)	N
Stack Array-based	Y but only up to the array size	1 but only for top item	N/A	1	1	1	N (see comment)
Queue link-list based	Y	1 but only for top item	N/A	1	1	1 (if list tracks size, else N)	N

Comment about arrays and space. Suppose I have an array of size 1,000,000 that has 5 items of data in it. Then the space used is 1000000*M + 5*Z (where M is the space used by an empty spot the array and Z is the space used by the data item). If I have 10000 items of data in the same array, then the space used is 1000000*M + 10000*Z. Importantly, the only this that changes is the multiplier of Z. Hence, the space used by an array is O(N).

Sample Problem: Linked Lists

The BubbleSwapList class (below) has a private Node class and exactly one predefined method, `public void add(E element)`. Add a new method, `void bubbleSwap(Node n)`, to BubbleSwapList that takes one node and swaps its location in the list with the next node. For example, suppose a list contains

[z w t a f h]

in that order. Calling bubbleSwap with the node containing "a" should change the list to now be

[z w t f a h]

`bubbleSwap` must swap the nodes, not just their contents. Note that this is a different linked list implementation than you may have seen elsewhere and you do not have access to any methods other than those listed here.

```
public class BubbleSwapList<E>  {
    private class Node<E> {
     public final E element;
     public Node<E> next;
     public Node<E> previous;
     public Node(E element) {
         this.element = element;
         this.next = null; this.previous = null;
    }}
    private Node<E> head;
    public BubbleSwapList() {
     head = null;

    }
    public void add(E element) {
     Node<E> toAdd = new Node<E>(element);
     if (head != null)
         head.previous=toAdd;
     toAdd.next = head;
     head = toAdd;
     }}
```

Sample Problem: Queues

Write a Java class that implements the QMerge interface given below. The class should only have the method queueMerge. Also give a Big-O time bound for your merge algorithm.

The queues both implement the QueueInterface given below (and nothing else). Elements in the queues all implement the Comparable interface (also given below):

```
public interface QueueInterface<E> {
    int size();          // the number of items in q
    boolean isEmpty();   // return true iff q is empty
    E first();           // return the first item in q
    void enqueue(E e);   // add the item to q
    E dequeue();         // remove from, and return the first item in q
}

public interface Comparable<T> {
    /* Returns a negative integer, zero, or a positive integer
     * as this object is less than, equal to, or greater than
     * the specified object (o)
     */
    int compareTo(T o)
}

public interface QMerge<E> {
    /* takes two sorted queues q1 and q2 containing Objects that
     * implement Comparable. Creates and returns a new queue
     * that is sorted and contains all elements from q1 and
     * q2.
     * @param q1 a sorted queue to be merged
     * @param q2 a sorted queue to be merged
     * @return the merger of the two queues, still sorted.  The
     * number of elements in the returned queue should be equal
     * to the sum of the number of elements in q1 and q2
     **/
    public QueueInterface<E> queueMerge(QueueInterface<E> q1,
QueueInterface<E> q2);
}
```

Sample Problem: Analysis and Style

The program below functions correctly. What does it do? Stylistically it is a disaster. Identify and correct style faults

```
// Imports not shown (to save space)
public class DoMany {
    private String f;
    public DoMany(String fn)
    {
        f = fn;
    }
    public void Dave() {
        try (BufferedReader br = new BufferedReader(new
FileReader(f))) {
            int hungry = 1;
            String line;
            while (null != (line = br.readLine())) {
                int i = 0;
                try {
                    i = Integer.parseInt(line);
                    int j = i / hungry;
                    System.out.println("Result " + j);
                }
                catch (ArithmeticException dze)
                {
                    System.err.println("You divided by zero!!!
");
                }
                catch (NumberFormatException nfe) {
                }
                hungry = i;
            }
        } catch (FileNotFoundException fnf) {
            System.out.println("Could not open ||" + f + "||");
            return;
        } catch (IOException ioe) {
            System.err.println("Could not read ||" + f + "||");
            return;
        }}
    public static void main(String[] args) {
        DoMany mm = new DoMany("data.txt");
        mm.Dave();
    }}
```

Sample Problem: Class Design

Transportation devices vary according to their mode of travel. There are three modes: land, sea and air. For each mode there are multiple types: Air has planes and helicopters; Land has cars and trucks; while water has boats and hovercraft. Finally, among cars there a special case — the Yugo — which is unable to move.

The class below defines `TransportDevice`. Write class definitions that build upon `TransportDevice` for up to 10 classes to efficiently capture information above. Following the pattern of `TransportDevice` any methods you write should just print a string.

```
public abstract class TransportDevice {
    public void printType() { System.out.println("XXX"); }
    public void printMode() { System.out.println("All"); }
}
```