Final Day 1

```
public void printLeaves() {
    if (overallRoot == null)
        System.out.println("no leaves");
    else {
        System.out.print("leaves:");
        printLeaves(overallRoot);
        System.out.println();
    }
}

private void printLeaves(IntTreeNode root) {
    if (root != null)
        if (root.left == null && root.right == null)
            System.out.print(" " + root.data);
        else {
            printLeaves(root.right);
            printLeaves(root.left);
        }
}


public class FoodData implements Comparable<FoodData> {
    private String name;
    private double fat;
    private double carbs;
    private double protein;

    public FoodData(String name, double fat, double carbs,
                    double protein) {
        if (fat < 0 || carbs < 0 || protein < 0)
            throw new IllegalArgumentException();
        this.name = name;
        this.fat = fat;
        this.carbs = carbs;
        this.protein = protein;
    }

    public String getName() {
        return name;
    }

    public double getCalories() {
        return 9 * fat + 4 * (carbs + protein);
    }

    public double percentFat() {
        return 100.0 * (9 * fat / getCalories());
    }

    public String toString() {
        return name + ": " + fat + "g fat, " + carbs +
               "g carbohydrates, " + protein + "g protein";
    }

    public int compareTo(FoodData other) {
        double difference = percentFat() - other.percentFat();
        if (difference < 0)
            return -1;
        else if (difference > 0)
```

```
            return 1;
        else
            return name.compareTo(other.name);
    }
}
```

```
public boolean bubble() {
    boolean swap = false;
    if (front != null && front.next != null) {
        if (front.data > front.next.data) {
            swap = true;
            ListNode temp = front;
            front = front.next;
            temp.next = front.next;
            front.next = temp;
        }
        ListNode current = front;
        while (current.next != null && current.next.next != null) {
            if (current.next.data > current.next.next.data) {
                swap = true;
                ListNode temp = current.next.next;
                current.next.next = temp.next;
                temp.next = current.next;
                current.next = temp;
            }
            current = current.next;
        }
    }
    return swap;
}
```

Final Day 2

Statement	Output
var1.method2();	Table 2/Chair 2
var2.method2();	Table 2/Chair 2
var3.method2();	Table 2/Chair 2
var4.method2();	Lamp 2
var5.method2();	Table 2/Chair 2
var6.method2();	compiler error
var1.method1();	Table 1
var2.method1();	compiler error
var3.method1();	Couch 1
var4.method1();	compiler error
var1.method3();	Table 3/Table 1
var2.method3();	compiler error
((Lamp)var4).method1();	Lamp 1
((Lamp)var2).method1();	runtime error
((Table)var5).method1();	Couch 1
((Couch)var1).method1();	runtime error
((Chair)var6).method2();	Chair 2
((Couch)var5).method3();	Table 3/Couch 1

```
public static int recordDate(Map<String, List<String>> dates,
                                String name1, String name2) {
        if (!dates.containsKey(name1)) {
            dates.put(name1, new LinkedList<String>());
        }
        if (!dates.containsKey(name2)) {
            dates.put(name2, new LinkedList<String>());
        }
        dates.get(name1).add(0, name2);
        dates.get(name2).add(0, name1);
        int n = 0;
        for (String s : dates.get(name1)) {
            if (s.equals(name2)) {
                n++;
            }
        }
        return n;
    }
```

```
// standard recursive solution
public void trim(int min, int max) {
    overallRoot = trim(overallRoot, min, max);
}
```

```
private IntTreeNode trim(IntTreeNode root, int min, int max) {
    if (root != null) {
        if (root.data < min) {
            root = trim(root.right, min, max);
        } else if (root.data > max) {
            root = trim(root.left, min, max);
        } else {
            root.left  = trim(root.left,  min, max);
            root.right = trim(root.right, min, max);

        }
    }
    return root;
```