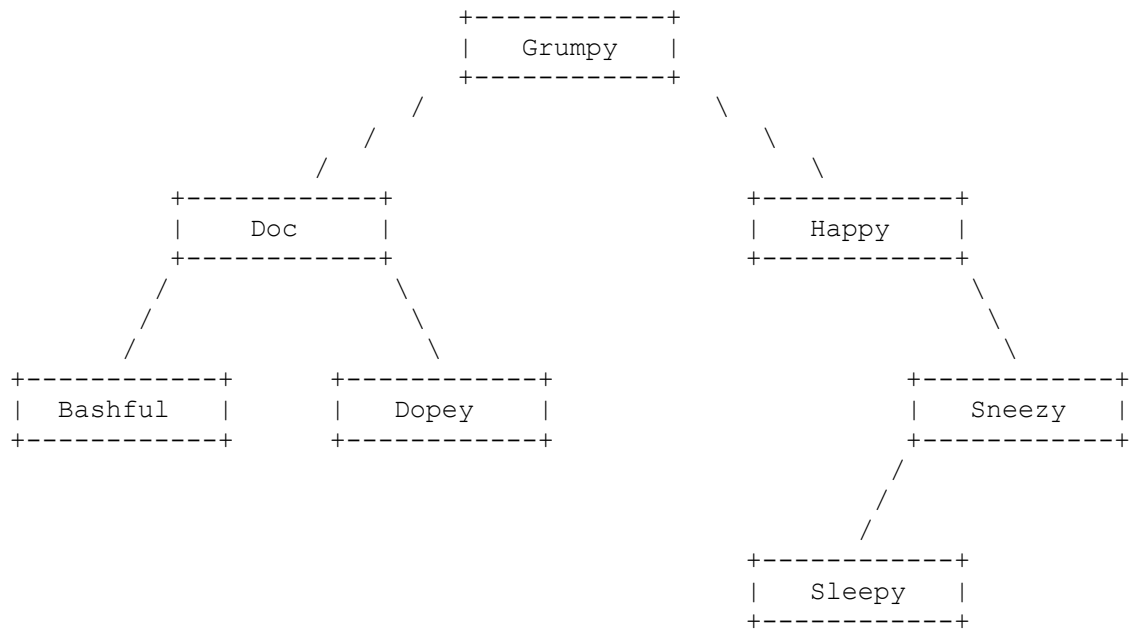
Problem #1

```
Preorder traversal:  8 7 2 4 9 5 1 3 6 0
Inorder traversal:   4 2 9 7 8 3 1 6 5 0
Postorder traversal: 4 9 2 7 3 6 1 0 5 8
```

Problem #2



Problem #3

Statement	Output
var1.method2()	Box 2
var2.method2()	Jar 2
var3.method2()	Cup 2 / Box 2
var4.method2()	Jar 2
var5.method2()	Compiler Error
var6.method2()	Pill 2
var1.method3()	Box 2 / Box 3
var2.method3()	Compiler Error
var3.method3()	Cup 2 / Box 2 /Box 3
var4.method3()	Jar 2 / Box 3
((Cup) var1).method1()	Runtime Error
((Jar) var2).method1()	Jar 1
((Cup) var3).method1()	Cup 1
((Cup) var4).method1()	Runtime Error
((Jar) var4).method2()	Jar 2
((Box) var5).method2()	Box 2
((Pill) var5).method3()	Compiler Error
((Jar) var2).method3()	Jar 2 / Box 3
((Cup) var3).method3()	Cup 2 / Box 2 /Box 3
((Cup) var5).method3()	Runtime Error

Problem #4

```
    public Map<String, List<Integer>> indexMap(List<String> data) {
        Map<String, List<Integer>> result = ew TreeMap<String, List<Integer>>();
        for (int i = 0; i < data.size(); i++) {
            String next = data.get(i);
            if (!result.containsKey(next)) {
               result.put(next, new ArrayList<Integer>());
            }
            result.get(next).add(i);
        }
        return result;
    }
```

Problem #5

```
   public class TeamData implements Comparable<TeamData> {
       private String name;
       private int solved;
       private int totalTime;
       private int problems;

       public TeamData(String name, int problems) {
           this.name = name;
           this.problems = problems;
           this.totalTime = 0;
           this.solved = 0;
       }

       public void success(int problem, int time) {
           solved++;
           totalTime += time;
       }

       public String toString() {
           return name + " solved " + solved + " of " + problems + " in "
                    + totalTime + " minutes";
       }


       public int solved() {
           return solved;
       }

       public int time() {
           return totalTime;
       }

       public double percentCorrect() {
           return 100.0 * solved / problems;
       }

       public int compareTo(TeamData other) {
           if (solved != other.solved){
               return other.solved - solved;
           } else {
               return totalTime - other.totalTime;
           }
       }
   }
```

Problem #6

```
    public int evenBranches() {
        return evenBranches(overallRoot);
    }

    private int evenBranches(IntTreeNode root) {
        if (root == null)
            return 0;
        else if (root.left == null && root.right == null)
            return 0;
        else if (root.data % 2 == 0)
            return 1 + evenBranches(root.left) + evenBranches(root.right);
        else
            return evenBranches(root.left) + evenBranches(root.right);
    }
```

Problem #7

```
    public int matches(IntTree other) {
        return matches(overallRoot, other.overallRoot);
    }

    private int matches(IntTreeNode root1, IntTreeNode root2) {
        if (root1 == null || root2 == null)
            return 0;
        else {
            int sum = matches(root1.left, root2.left) +
            matches(root1.right, root2.right);
            if (root1.data == root2.data)
                return 1 + sum;
            else
                return sum;
        }
    }
```

Problem #8

```
    public void markMultiples(int n) {
        if (n <= 0)
            throw new IllegalArgumentException();
        if (front != null) {
            ListNode current = front;  // tricky Stuart sets current before
            // resetting front
            if (front.data % n == 0)
                front = new ListNode(0, front);
            while (current.next != null) {
                if (current.next.data % n == 0) {
                    current.next = new ListNode(0, current.next);
                    current = current.next.next;
                } else
                    current = current.next;
            }
        }
    }
```