Problem #1

Statement	Output
var1.method1();	Morty 1
var2.method1();	Beth 1/Rick 1
var3.method1();	Rick 1
var4.method1();	compiler error
var5.method1();	Morty 1
var6.method1();	compiler error
var1.method2();	Rick 2/Morty 1
var2.method2();	Rick 2/Beth 1/Rick 1
var3.method2();	Rick 2/Rick 1
var4.method2();	compiler error
var5.method2();	Rick 2/Morty 1
var6.method2();	compiler error
((Beth)var1).method3();	compiler error
((Summer)var6).method3();	Summer 3
((Morty)var4).method1();	runtime error
((Rick)var6).method2();	Rick 2/Rick 1
((Rick)var4).method1();	Beth 1/Rick 1
((Beth)var6).method3();	compiler error
((Morty)var3).method3();	runtime error
((Morty)var5).method3();	Morty 3

Problem #2

One possible solution

```
    public boolean hasPathSum(int sum) {
        return hasPathSum(overallRoot, sum);
    }

    private boolean hasPathSum(IntTreeNode root, int sum) {
        if (root == null)
            return false;
        if (root.data == sum)
            return true;
        int sum2 = sum - root.data;
        return hasPathSum(root.left, sum2) || hasPathSum(root.right, sum2);
    }
```

Problem #3

One possible solution

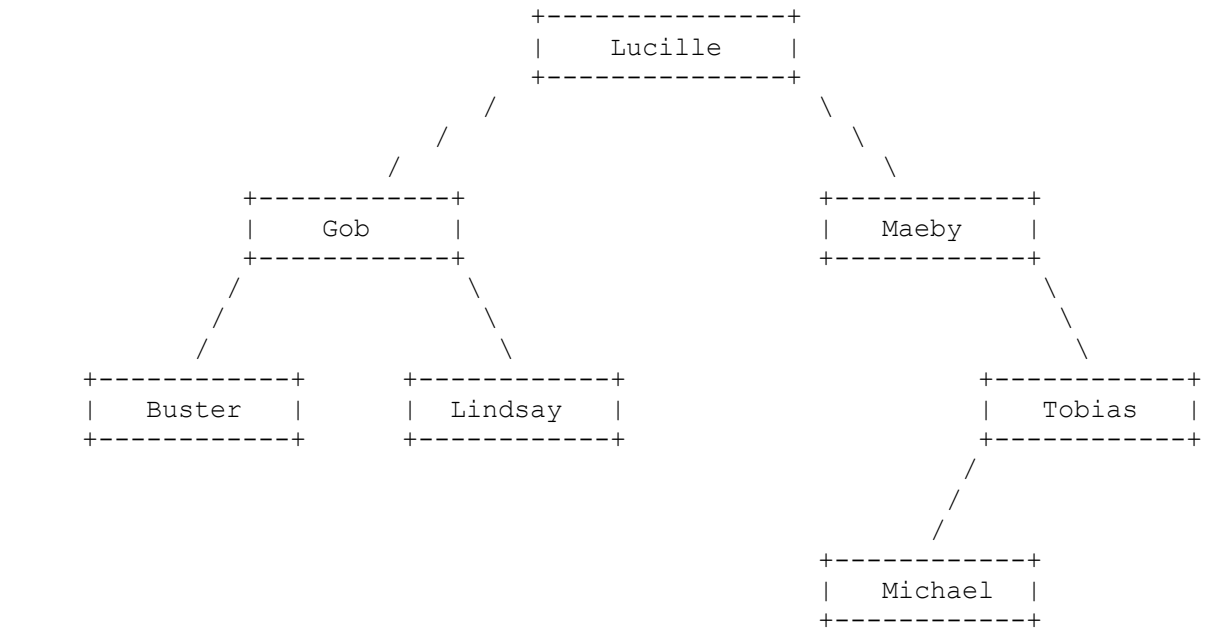
```
    public void weave(LinkedIntList other) {
        if (front == null) {
            front = other.front;
        } else if (other.front != null) {
            ListNode current1 = front;
            ListNode current2 = other.front;
            while (current1.next != null && current2.next != null) {
                ListNode temp = current1.next;
                current1.next = current2;
                current2 = current2.next;
                current1.next.next = temp;
                current1 = temp;
            }

            if (current1.next == null) {
                // we know curr2 might have more list, just attach it at end
                current1.next = current2;
            } else {
                // there is one node left in curr2, weave it in
                current2.next = current1.next;
                current1.next = current2;
            }
        }
        other.front = null;
    }
```

Problem #4

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

Problem #5



Problem #6

One possible solution

```
   public Map<Integer, Set<String>> split(Set<String> words) {
       Map<Integer, Set<String>> buckets = new TreeMap<Integer, Set<String>>();
       for (String word : words) {
           int n = word.length();
           if (!buckets.containsKey(n)) {
               buckets.put(n, new TreeSet<String>());
           }
           buckets.get(n).add(word);
       }
       return buckets;
   }
```

Problem #7

One possible solution

```
    public class AdmissionsEntry implements Comparable<AdmissionsEntry> {
        private String ID;
        private int ratings;
        private double total;
        private boolean discuss;

        public AdmissionsEntry(String ID) {
            this.ID = ID;
            this.ratings = 0;
            this.total = 0.0;
            this.discuss = false;
        }

        public void rate(double rating) {
            ratings++;
            total += rating;
            if (rating >= 4) {
                discuss = true;
            }
        }

        public void flag() {
            discuss = true;
        }

        public String getID() {
            return ID;
        }

        public double getRating() {
            if (ratings == 0) {
                return 0.0;
            } else {
                return total / ratings;
            }
        }

        public String toString() {
            double rating = Math.round(100 * getRating()) / 100.0;
            return ID + ": " + rating;
        }
```

```
        public int compareTo(AdmissionsEntry other) {
            if (discuss && !other.discuss) {
                return -1;
            } else if (!discuss && other.discuss) {
                return 1;
            } else if (getRating() > other.getRating()) {
                return -1;
            } else if (getRating() < other.getRating()) {
                return 1;
            } else {
                return ID.compareTo(other.ID);
            }
        }
    }
```

Problem #8

One possible solution

```
    public void add(IntTree other) {
        overallRoot = add(overallRoot, other.overallRoot);
    }

    private IntTreeNode add(IntTreeNode root1, IntTreeNode root2) {
        if (root2 != null) {
            if (root1 == null)
                root1 = new IntTreeNode(0);
            root1.data += root2.data;
            root1.left = add(root1.left, root2.left);
            root1.right = add(root1.right, root2.right);
        }
        return root1;
    }
```