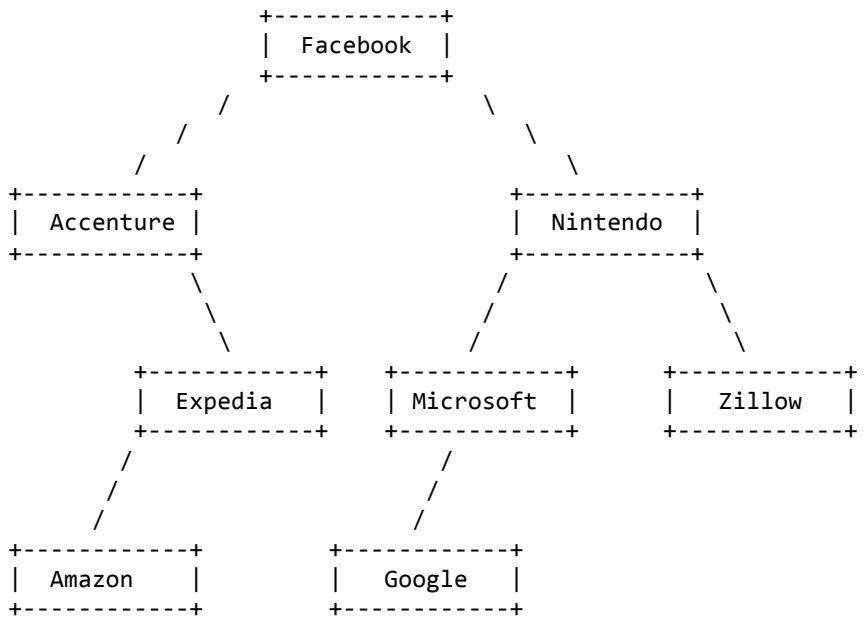
CSE 143, Spring 2014
Final Key

1. Binary Tree Traversal
   - Pre-order: 4, 1, 6, 3, 7, 0, 8, 5, 9, 2
   - In-order: 3, 7, 6, 1, 0, 8, 5, 4, 2, 9
   - Post-order: 7, 3, 6, 5, 8, 0, 1, 2, 9, 4

2. Binary Search Tree



3. Collections Mystery
   - [4, 8, 10, 12]
   - [12, 14, 23, 25]
   - [23, 25, 31, 37, 43, 47]

4. Inheritance/Polymorphism Mystery

```
var1.method1();                    Blues 1 Jazz 3 Blues 3
var4.method1();                    Blues 1 Punk 3
var5.method1();                    error
var2.method2();                    Punk 2
var3.method2();                    error
var1.method3();                    Jazz 3 Blues 3
var2.method3();                    Punk 3
((Punk) var5).method1();           error
((Jazz) var3).method2();           error
((Punk) var4).method2();           Punk 2
((Blues) var2).method2();          error
((Rap) var3).method2();            Rap 2
((Jazz) var4).method2();           error
((Jazz) var4).method3();           Punk 3
((Blues) var5).method3();          Jazz 3 Blues 3
```

5. Binary Tree Programming

```
public boolean sameStructure(IntTree other) {
   return sameStructure(overallRoot, other.overallRoot);
}

private boolean sameStructure(IntTreeNode root1, IntTreeNode root2) {
   if (root1 == null && root2 == null) {
      return true;
   } else if (root1 == null || root2 == null) {
      return false;
   } else {
      return sameStructure(root1.left, root2.left) &&
sameStructure(root1.right, root2.right);
   }
}

private boolean sameStructure(IntTreeNode root1, IntTreeNode root2) {
   if (root1 == null || root2 == null) {
      return root1 == null && root2 == null;
   } else {
      return sameStructure(root1.left, root2.left) &&
sameStructure(root1.right, root2.right);
   }
}

private boolean sameStructure(IntTreeNode root1, IntTreeNode root2) {
   if (root1 == null) {
      return root2 == null;
   } else if (root2 == null) {
      return false;
   } else {
      return sameStructure(root1.left, root2.left) &&
sameStructure(root1.right, root2.right);
   }
}
```

```
private boolean sameStructure(IntTreeNode root1, IntTreeNode root2) {
   if (root1 != null || root2 != null) {
      if (root1 == null || root2 == null) {
         return false;
      } else {
         return sameStructure(root1.left, root2.left) &&
sameStructure(root1.right, root2.right);
      }
   }
   return true;
}
```

6. Collections Programming

```
public static Map<String, Set<String>> birthdayMonths(Map<String, Date>
birthdays) {
  Map<String, Set<String>> months = new TreeMap<String, Set<String>>();

  for (String name : birthdays.keySet()) {
     String month = birthdays.get(name).getMonth();
     if (!months.containsKey(month)) {
        months.put(month, new TreeSet<String>());
     }
     months.get(month).add(name);
  }

 return months;
}
```

7. Comparable

```
public class Donation implements Comparable<Donation> {
   private String org;
   private double amount;
   private boolean deductible;

   public Donation(String org, double amount, boolean deductible) {
      if (amount <= 0) {
         throw new IllegalArgumentException();
      }
      this.org = org;
      this.amount = amount;
      this.deductible = deductible;
   }

   public int compareTo(Donation other) {
      if (deductible && !other.deductible) {
         return -1;
      } else if (!deductible && other.deductible) {
         return 1;
      } else if (amount != other.amount) {
         return (int) Math.signum(amount - other.amount);
      } else {
         return org.compareTo(other.org);
      }
   }
```

```
   public int compareTo(Donation other) {
      if (deductible == other.deductible) {
         if (amount > other.amount) {
            return 1;
         } else if (amount < other.amount) {
            return -1;
         } else {
            return org.compareTo(other.org);
         }
      } else {
         if (deductible) {
            return -1;
         } else {
            return 1;
         }
      }
   }

   public String toString() {
      String prefix = "";
      if (deductible) {
         prefix += "* ";
      }
      return prefix + "$" + amount + ": " + org;
   }
}
```

8. Binary Tree Programming

```
public void makeFull() {
    overallRoot = makeFull(overallRoot, 1);
}

private IntTreeNode makeFull(IntTreeNode root, int level) {
    if (root != null) {
        if (root.left == null && root.right != null) {
            root = new IntTreeNode(-level, root, root.right);
            root.left.right = null;
        } else if (root.left != null && root.right == null) {
            root = new IntTreeNode(-level, root.left, root);
            root.right.left = null;
        }
        root.left = makeFull(root.left, level + 1);
        root.right = makeFull(root.right, level + 1);
    }
    return root;
}
```

9. Linked List Programming

```
public LinkedIntList collapseDuplicates() {
   int removed = 0;
   LinkedIntList result = new LinkedIntList();

   if (front == null) {
      front = new ListNode(0);
   } else {
      ListNode current = front;
      while (current.next != null) {
         if (current.data == current.next.data) {
            ListNode temp = current.next;
            current.next = current.next.next;
            temp.next = result.front;
            result.front = temp;
            removed++;
         } else {
            current = current.next;
         }
      }
      current.next = new ListNode(removed);
   }

   return result;
}
```