

CSE 143, Winter 2009

Section Handout #20 (Sample Final Exam #3) Key

1.

<u>Statement</u>	<u>Output</u>
var1.method1();	Nose 1
var2.method1();	Eye 1/Mouth 1
var3.method1();	Eye 1/Mouth 1
var1.method2();	Mouth 2/Nose 1
var2.method2();	Ear 2
var3.method2();	Mouth 2/Eye 1/Mouth 1
var4.method2();	error
var5.method2();	Mouth 2/Nose 1
var6.method2();	Ear 2
var1.method3();	error
var2.method3();	Ear 3
var3.method3();	error
((Nose) var5).method3();	Nose 3
((Eye) var1).method1();	Nose 1
((Eye) var4).method1();	error
((Nose) var1).method3();	Nose 3
((Mouth) var4).method1();	Mouth 1
((Ear) var5).method3();	error
((Eye) var6).method3();	error
((Mouth) var4).method2();	Mouth 2/Mouth 1

2.

```
public class HistoryList extends ArrayList implements Comparable<HistoryList> {
    private List<String> history;

    public HistoryList() {
        this(DEFAULT_CAPACITY);
    }

    public HistoryList(int capacity) {
        super(capacity);
        history = new ArrayList<String>();
        history.add(toString());
    }

    public void add(int index, int value) { // the other add method calls this one,
        super.add(index, value);          // so we don't need to override it
        history.add(toString());
    }

    public void remove(int index) {
        super.remove(index);
        history.add(toString());
    }

    public void set(int index, int value) {
        super.set(index, value);
        history.add(toString());
    }

    public int historySize() {
        return history.size();
    }

    public String getHistory(int index) {
        return history.get(index);
    }

    public int compareTo(HistoryList other) {
        if (historySize() != other.historySize()) {
            return historySize() - other.historySize();
        } else {
            return size() - other.size();
        }
    }
}
```

3.

```

public void removeDuplicates() {
    ListNode current = front;
    while (current != null && current.next != null) {
        ListNode current2 = current;
        while (current2.next != null) {
            if (current2.next.data == current.data) {
                current2.next = current2.next.next;
            } else {
                current2 = current2.next;
            }
        }
        current = current.next;
    }
}

```

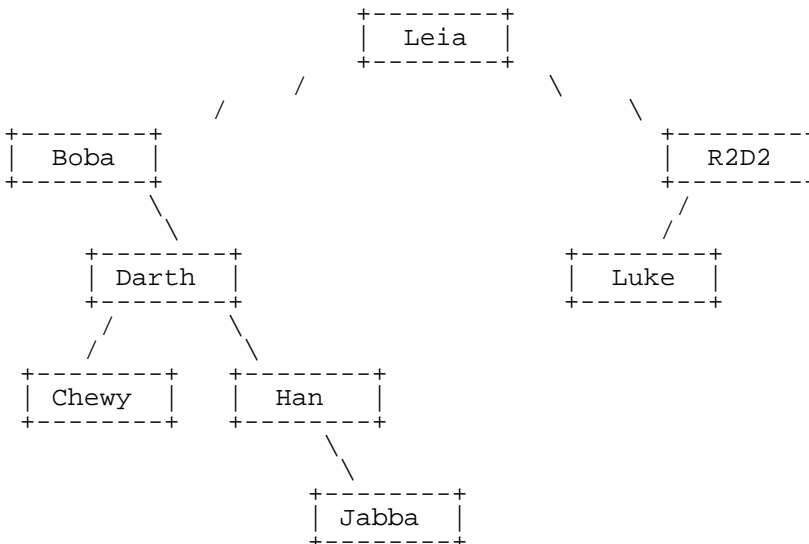
4.

(a) Indexes examined: 6, 9, 7, 8 Value returned: -9

(b) {-4, 56, 24, 5, 39, **15**, 27, 10}
 {-4, **5**, 24, **56**, 39, 15, 27, 10}
 {-4, 5, **10**, 56, 39, 15, 27, **24**}

(c) {15, 56, 24, 5, 39, -4, 27, 10}
 {15, 56, 24, 5} {39, -4, 27, 10} split
 {15, 56} {24, 5} {39, -4} {27, 10} split
 {15} {56} {24} {5} {39} {-4} {27} {10} split
 {15, 56} {5, 24} {-4, 39} {10, 27} merge
 {5, 15, 24, 56} {-4, 10, 27, 39} merge
 {-4, 5, 10, 15, 24, 27, 39, 56} merge

5. (a)



(b)

Pre-order: Leia, Boba, Darth, Chewy, Han, Jabba, R2D2, Luke

In-order: Boba, Chewy, Darth, Han, Jabba, Leia, Luke, R2D2

Post-order: Chewy, Jabba, Han, Darth, Boba, Luke, R2D2, Leia

6.

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

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

7.

```
public IntTree combineWith(IntTree other) {
    IntTree result = new IntTree();
    result.overallRoot = combine(overallRoot, other.overallRoot);
    return result;
}

private IntTreeNode combine(IntTreeNode root1, IntTreeNode root2) {
    if (root1 == null) {
        if (root2 == null) {
            return null;
        } else { // root2 != null
            return new IntTreeNode(2, combine(null, root2.left),
                combine(null, root2.right));
        }
    } else { // root1 != null
        if (root2 == null) {
            return new IntTreeNode(1, combine(root1.left, null),
                combine(root1.right, null));
        } else {
            return new IntTreeNode(3, combine(root1.left, root2.left),
                combine(root1.right, root2.right));
        }
    }
}
```