Computer Science 235
Final Examination
May 4, 2011

Based on the code found at the end of the test (1–10, 1 point each; 11–13, 2 points each):

1. Give an example of a local variable, with a line number.

2. Give an example of an instance variable, with a line number.

3. Give an example of a formal parameter, with a line number.

4. Give an example, in line numbers, of a constructor (not a contructor *call*).

5. Give an example of an explicit cast (say what is cast to what), with a line number.

6. Give an example of an automatic type conversion (say what is converted to what), with a line number.

7. Given an example of an array creation, with a line number.

8. Give an example of subtyping. Name a type and say what other type it is a subtype of.

9. Give an example of a literal, with a line number.

10. Give an example of a method invocation that uses polymorphism, with a line number.

11. Which of the following are found on line 74? (Circle all that apply).

Declaration Assignment Initialization Instantiation

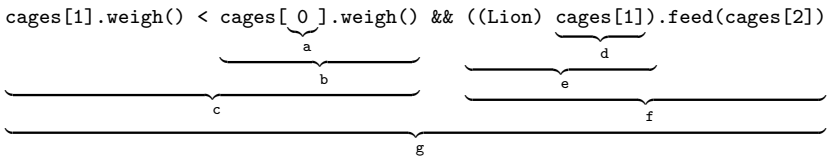
12. Which of the following are found on line 76? (Circle all that apply).

Declaration Assignment Initialization Instantiation

13. Which of the following are found on line 85? (Circle all that apply).

Declaration Assignment Initialization Instantiation

14. Give the static type of each labelled expression from lines 87–88. (7 points total.)



a.

b.

c.

d.

e.

f.

g.

15. Write a static method that, given an array of `int`s, returns an array containing the differences between each pair of adjacent elements in the original array. For example, given | 3 | 5 | 4 | 12 | 0 | 23 | 16 | should yield | 2 | -1 | 8 | -12 | 23 | -7 |. (8 points.)

16. Write a static method that, given an `ArrayList` of `String`s, will return an array of `int`s containing the lengths of the strings. 8 points.)

17. a. Given the following `TreeNode` class, write a method `nLeaves()` that returns the number of leaf nodes among its descendents. Recall that a *leaf* is a node that has no children. *Note that in this tree, the absence of a child is indicated by a* ***`null`*** *reference.* (12 points)

```
public class TreeNode {
    private int datum;
    private TreeNode leftChild;
    private TreeNode rightChild;
    public TreeNode(int d, TreeNode left, TreeNode right) {
        datum = d;
        leftChild = left;
        rightChild = right;
    }
    public TreeNode(int d) {
        datum = d;
        leftChild = null;
        rightChild = null;
    }

    public int nLeaves() {









   }
}
```

b. Now re-write the method from your previous answer so that it never tests if a link is null; use exception handling instead. (4 points.)

```
public class TreeNode {
    private int datum;
    private TreeNode leftChild;
    private TreeNode rightChild;
    public TreeNode(int d, TreeNode left, TreeNode right) {
        datum = d;
        leftChild = left;
        rightChild = right;
    }
    public TreeNode(int d) {
        datum = d;
        leftChild = null;
        rightChild = null;
    }

    public int nLeaves() {









   }
}
```

18. Write a method that, given an `ArrayList<String>`, returns the most frequently occurring string value. (10 points)

Class/`interface` interfaces for reference; these are not complete, but sufficient for the problems here.

```
interface Iterator<E> {
    boolean hasNext();
    E  next();
}

interface HashSet<E> {
    HashSet<E>();
    void add(E elem);
    void remove(E elem);
    boolean contains(E elem);
    int size();
    Iterator<E> iterator();
}

interface HashMap<K, V> {
    HashMap<K, V>();
    V get(K key); // returns null is not present
    void put(K key, V value);
    void remove(K key);
    boolean containsKey(K key);
    boolean isEmpty();
    int size();
    Iterator<K> keySet().iterator();
}

interface ArrayList<E> {
    ArrayList<E>();
    void add(int index, E);   // insert at position index
    E get(int index);
    E remove(int index);  // remove and return element at position index
    boolean contains(E elem);
    int size();
    Iterator<E> iterator();
}
```

```
1  public interface Animal {
2
3      public boolean feed(String food);
4
5      public int weigh();
6  }
7
8  public class Carp implements Animal {
9
10     private int weight;
11
12     public Carp() {
13         weight = 1;
14     }
15
16     public boolean feed(String food) {
17         weight *=2;
18         return true;
19     }
20
21     public int weigh() { return weight; }
22 }
23
24 public class Lion implements Animal {
25
26     private int weight;
27
28     public Lion() {
29         weight = 20;
30     }
31
32     public boolean feed(String food) { return false; }
33
34     public boolean feed(Animal prey) {
35         if (prey instanceof Lion)
36             return false;
37         else {
38             weight += prey.weigh();
39             return true;
40         }
41     }
42
43     public int weigh() {
44         return weight;
45     }
46 }
47
48 public class Parrot implements Animal {
49
50     public boolean feed(String food) {
51         if (food.equals("worms"))
52             return true;
53         else
54             return false;
55     }
56
57     public int weigh() { return 1 ; }
58 }
59
60
61
62
63
```



```
64
65
66
67
68
69
70 public class Zoo {
71
72     public static void main(String[] args) {
73
74         Animal[] cages = new Animal[3];
75         cages[0] = new Carp();
76         cages[1] = new Lion();
77         cages[2] = new Parrot();
78
79         for (int i = 0; i < 5; i++)
80             for (int j = 0; j < cages.length; j++)
81                 cages[j].feed("worms");
82
83         int totalWeight = 0;
84         for (int i = 0; i < cages.length; i++)
85             totalWeight += cages[i].weigh();
86
87         if (cages[1].weigh() < cages[0].weigh()
88             && ((Lion) cages[1]).feed(cages[2]))
89             System.out.println(cages[1].weigh());
90         else
91             System.out.println(totalWeight);
92     }
93 }
```