Computer Science 235
Test 2
Nov 17, 2006

Based on the code found at the end of the test:

1. Give an example of a local variable, with a line number. (2 points)

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

3. Give an example of a formal parameter, with a line number. (2 points)

4. Give an example, in line numbers, of a constructor (not a contructor *call*). (2 points)

5. Give an example of subtyping. Name a type and say what other type it is a subtype of. (2 point)

6. Which of the following are found on line 13? (Circle all that apply). (4 points)

Declaration Assignment Initialization Instantiation

7. Which of the following are found on line 74? (Circle all that apply). (4 points)

Declaration Assignment Initialization Instantiation

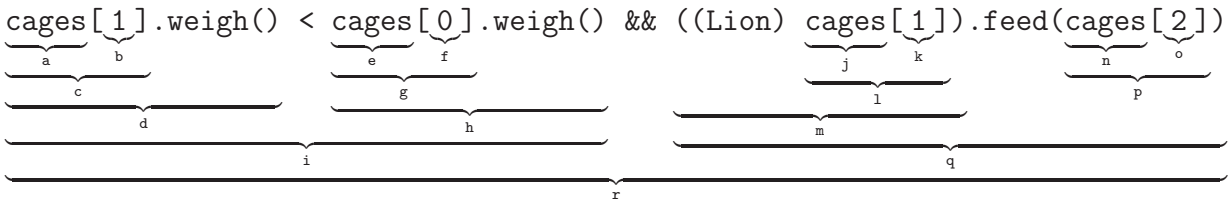
8. Which of the following are found on line 75? (Circle all that apply). (4 points)

Declaration Assignment Instantiation

9. In the invocation of the method `equals` on line 81, `cages[j]` is the _____ and

`"worms"` is the _____. (2 points)

10. Give the static type of each labelled expression from line 87-88. (1 point each)



a.

b.

c.

d.

e.

f.

g.

h.

i.

j.

k.

l.

m.

n.

o.

p.

q.

11. A *queue* is a data structure similar to a stack except that elements are extracted in the same order they are put in, so you extract the *earliest inserted* element instead of the *most recently inserted*. Queues exhibit First-In-First-Out order, whereas stacks exhibit Last-In-First-Out order. Write a class that implements the following interface. Using a `String` is recommended. (20 points)

```
public interface Queue {
     void pushBack(char c);      // add a char to the back of the queue
     char front();                    // return the char at the front
     char popFront();               // return and remove the char at the front
     boolean isEmpty();         // is this queue empty?
}
```

12. Recall that a *stack* is a container for data (a data structure) to which data can be added (*pushed*) or removed (*popped*) only at the top.

A stack to contain `int`s could implement the following interface.

```
public interface Stack {
     boolean isEmpty();    // is the stack empty?
     void push(int x);     // add an int to the top of the stack
     int pop();            // return and remove the top element
}
```

a. Write a class that implements this interface using an array. (You may assume `pop()` will never be called when the stack is empty, and that `push()` will never be called when it is full. The constructor should have a formal parameter that indicates the maximum number of elements the stack can hold.) (15 points.)

```
public class ArrayStack implements Stack {









}
```

13. You are writing a program for a college library to keep track of checked out books, compute fines, etc. In this library, students are fined 10 cents per day for a late book; faculty are not fined until a book is at least 5 days late, after which they are fined 5 cents per day.

Assume your system will have an interface `Book` (with several classes implementing it, representing various kinds of books), an interface `Patron` (as a supertype for classes representing students and faculty), and a program `Library` which contains (among other things) an array of `Book`s and a method which, given a `Patron`, computes the total fines for all of that patron's late books.

Given the following interfaces for `Book` and `Patron`, write (1) a class `Student` implementing `Patron`, (2) a class `Faculty` implementing `Patron`, and (3) the method `computeFine()` in class `Library`.

Do not do any more than you are asked to do. You do not need to write any `Book` classes, nor anything else in the `Patron` classes besides the method `computeFineOnBook()`. (10 points)

```
public interface Book {
  /**
     * Is this book checked out by the given
     * patron?
     * @param p The patron who may have
     *checked out this book
     * @return true if p holds this book, false
     * otherwise.
     */
     boolean isCheckedOutBy(Patron p);

     /**
      * How many days are left until this book
      * is due?
      * @return The number of days until the
      * book is due, a negative number if the
      * book is late.
      */
      int daysUntilDue();
}

public class Library {
    public static Book[] shelf;

    public static int computeFine(Patron p) {









    }
}
```

```
public interface Patron {
    /**
     * How much fine should this patron pay on
     * account of the given book?
     * @param b The book this patron may have
     * a fine for
     * @return The amount of money this patron
     * owes for this book, which would be 0 if this
     * patron does not hold this book, the book is
     * not late, or the book is in grace period.
     */
     int computeFineOnBook(Book b);
}
```

```
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 }
```