**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)

$$\underbrace{\underbrace{\underbrace{\underbrace{\texttt{cages}}_{a}\texttt{[}\underbrace{\texttt{1}}_{b}\texttt{]}}_{c}\texttt{.weigh()}}_{d} \texttt{ < } \underbrace{\underbrace{\underbrace{\texttt{cages}}_{e}\texttt{[}\underbrace{\texttt{0}}_{f}\texttt{]}}_{g}\texttt{.weigh()}}_{h}}_{i} \texttt{ \&\& } \underbrace{\underbrace{\texttt{((Lion) }\underbrace{\underbrace{\texttt{cages}}_{j}\texttt{[}\underbrace{\texttt{1}}_{k}\texttt{]}}_{l}\texttt{)}}_{m}\texttt{.feed(}\underbrace{\underbrace{\texttt{cages}}_{n}\texttt{[}\underbrace{\texttt{2}}_{o}\texttt{]}}_{p}\texttt{)}}_{q}}_{r}$$

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. a. Given the following `Node` class, write an iterative method `deleteEveryNth()` in the `List` which, given $n$, deletes every $n$th element from the list. For example, if the list is $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow$ `null`, then `deleteEveryNth(3)` will result in $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow$ `null`. (You may assume that $n$ is positive. Note that if $n = 1$, then this deletes the entire list; note also that $n = 1$ is the only occasion when the head is deleted). (12 points)

```
public class Node {
     private int datum;
     private Node next;
     public int datum() { return datum; }
     public Node next() { return next; }
     public void setNext(Node next) { this.next = next; }
}

public class List {
     Node head;
     public void deleteEveryNth(int n) {
















     }
}
```

b. Now do the same thing recursively. You may decide the return t ype and parameters for the `deleteEveryNth` in the `Node` class; accordingly, you should fill-in the method in the `List` class as well. This time you may assume $n > 1$ (and so the head will never be deleted by this). (12 points)

```
public class List {
     private Node head;
     public void deleteEveryNth(int n) {




     }
}

public class Node {
       private int datum;
       private Node next;

       public                    deleteEveryNth(                                ) {


























       }
 }
```

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 `Books` 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 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);
}
```

```
public class Library {
    public static Book[] shelf;

    public static int computeFine(Patron p) {








    }
}
```

```java
public interface Animal {

    public boolean feed(String food);

    public int weigh();
}

public class Carp implements Animal {

    private int weight;

    public Carp() {
        weight = 1;
    }

    public boolean feed(String food) {
        weight *=2;
        return true;
    }

    public int weigh() { return weight; }
}

public class Lion implements Animal {

    private int weight;

    public Lion() {
        weight = 20;
    }

    public boolean feed(String food) { return false; }

    public boolean feed(Animal prey) {
        if (prey instanceof Lion)
            return false;
        else {
            weight += prey.weigh();
            return true;
        }
    }

    public int weigh() {
        return weight;
    }
}

public class Parrot implements Animal {

    public boolean feed(String food) {
        if (food.equals("worms"))
            return true;
        else
            return false;
    }

    public int weigh() { return 1 ; }
}
```

```java
public class Zoo {

    public static void main(String[] args) {

        Animal[] cages = new Animal[3];
        cages[0] = new Carp();
        cages[1] = new Lion();
        cages[2] = new Parrot();

        for (int i = 0; i < 5; i++)
            for (int j = 0; j < cages.length; j++)
                cages[j].feed("worms");

        int totalWeight = 0;
        for (int i = 0; i < cages.length; i++)
            totalWeight += cages[i].weigh();

        if (cages[1].weigh() < cages[0].weigh()
            && ((Lion) cages[1]).feed(cages[2]))
            System.out.println(cages[1].weigh());
        else
            System.out.println(totalWeight);
    }
}
```