Computer Science 235
Final Examination Sample
Apr 20, 2011

Based on the code found at the end of the test (1–11, 1 point each; 12–15, 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 of a static variable, with a line number.

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

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

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

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

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

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

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

12. Which of the following are found on line 17? (Circle all that apply).

Declaration Assignment Initialization Instantiation

13. Which of the following are found on line 44? (Circle all that apply).

Declaration Assignment Initialization Instantiation

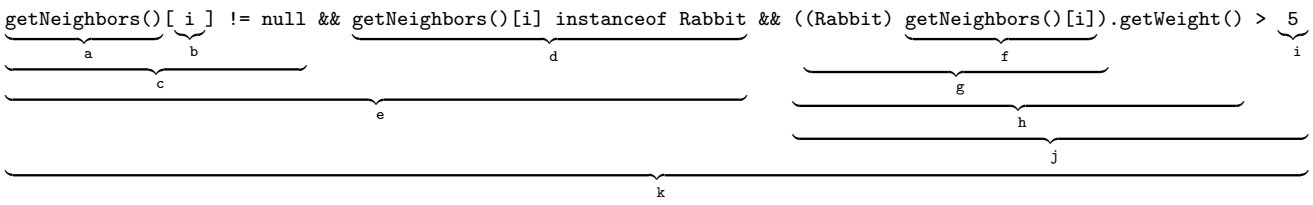
14. Which of the following are found on line 110? (Circle all that apply).

Declaration Assignment Initialization Instantiation

15. Which of the following are found on line 125? (Circle all that apply).

Declaration Assignment Initialization Instantiation

16. Give the static type of each labelled expression from lines 25–27. (6 points total.)



a. g.

b. h.

c. i.

d. j.

e. k.

f.

17. Write a static method which, given an array of `String`s and an `int`, will return the number of `String`s in the array that have that `int` as their length. (8 points.)

18. Write a static method which, given an array of `int`s, will return an array of `boolean`s indicating whether each element of the original array is odd. For example, given

3	5	4	12	0	23	16

should yield

true	true	false	false	false	true	false

. (8 points.)

19. a. Given the following `Node` class, write an iterative method `evenOut()` in the list class which modifies the list to one that has the same number of nodes as the original, but in which all odd data are doubled; even data are unchanged. For example, if the list is $5 \to 3 \to 2 \to 1 \to 4 \to 1 \dashv|$, it would mutate the list into $10 \to 6 \to 2 \to 2 \to 4 \to 2 \dashv|$. (Remember that you are not able to change a node's datum; instead, make new nodes.) (12 points)

```
public class Node {
     private int datum;
     private Node next;
     public Node(int datum, Node next) {
         this.datum = datum;
         this.next = 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 evenOut() {









   }
}
```

b. Now do the same thing recursively. Although it is now possible to change a node's datum, you still *should* not, in order to get full credit. (Think of this method as a message to a node that says, "Return the list that is like you but with each node made even.") (12 points)

```
public class List {
     private Node head;
     public void evenOut() {

            if (head != null)
                head = head.evenOut();

     }
}

public class Node {
       private int datum;
       private Node next;
       public Node(int datum, Node next) {
           this.datum = datum;
           this.next = next;
       }

       public Node evenOut() {









   }
}
```

c. Now re-write the method from your previous answer so that it never tests if a link is null; use exception handling instead. (3 points.)

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

        public Node(int datum, Node next) {
           this.datum = datum;
           this.next = next;
       }

       public Node evenOut() {









   }
}
```

20. The following signature defines a structure into which one can insert data and find the minimum, maximum, the range (the difference between the max and the min), and the average of those data.

```
public interface StatsDataSet {
     void addValue(double datum);
     double min();
     double max();
     double range();
     double average();
}
```

Write a class that implements this interface *without using an array, a linked list or any kind of collection (such as as* ***`ArrayList`***). The constructor should take the first datum as a parameter. (Hint: Notice that you are never asked to return a specific value, only the min, max, range, and average; think of a way that will make `min()` and `max()` trivial and `range()` and `average()` pretty easy.) (12 points)

21. Write a method that, given an `ArrayList<String>`, returns the count of distinct string values that it contains. (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 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 Agent {
2
3       void act();
4
5   }
6
7
8   public class Fox implements Agent {
9
10      private Agent[][] grid;
11
12      private int xPos, yPos;
13
14      private static int speed = 3;
15
16      public Fox(Agent[][] grid, int xPos, int yPos) {
17          this.grid = grid;
18          this.xPos = xPos;
19          this.yPos = yPos;
20      }
21
22      public void act() {
23
24          for (int = 0; i < getNeighbors().length; i++)
25              if (getNeighbors()[i] != null
26                  && getNeighbors()[i] instanceof Rabbit
27                  && ((Rabbit) getNeighbors()[i]).getWeight() > 5)
28                  ((Rabbit) getNeighbors()[i]).die()
29
30              }
31
32          // try to move in a random direction
33          int i = (int) Math.floor(speed * Math.random()) - 1;
34          int j = (int) Math.floor(speed * Math.random()) - 1;
35          if (grid[i + xPos][j + yPos] == null) {
36              grid[i + xPos][j + yPos] = this;
37              grid[xPos][yPos] = null;
38              xPos += i;
39              yPos += j;
40          }
41      }
42
43      public Agent[] getNeighbors() {
44          Agent[] toReturn = new Agent[8];
45
46          int k = 0;
47          for (int i = -speed; i <= speed; i ++)
48              for (int j = -speed; j <= speed; j++) {
49                  if (i != 0 || j != 0)
50                      toReturn[k] = grid[i][j];
51                  k++;
52              }
53
54          return toReturn;
55      }
56  }
57
58
59  public class Rabbit implements Agent {
60
61      private Agent[][] grid;
62
63      private int xPos, yPos;
64
65      private static int speed = 2;
66
67      private int weight;
68
69      public Rabbit(Agent[][] grid, int xPos, int yPos) {
70          this.grid = grid;
71          this.xPos = xPos;
72          this.yPos = yPos;
73          this.weight = 1;
```



```
74      }
75
76      public void act() {
77
78          // try to move in a random direction
79          int i = (int) Math.floor(speed * Math.random()) - 1;
80          int j = (int) Math.floor(speed * Math.random()) - 1;
81          if (grid[i + xPos][j + yPos] == null) {
82              grid[i + xPos][j + yPos] = this;
83              grid[xPos][yPos] = null;
84              xPos += i;
85              yPos += j;
86          }
87          weight++;
88      }
89
90      public int getWeight() { return weight; }
91
92      public void die() {
93          grid[xPos][yPos] = null;
94      }
95  }
96
97
98  public class PredPrey {
99
100     public static void main(String[] args) {
101
102         Scanner keyboard = new Scanner(System.in);
103
104         System.out.print("What size grid? --> ");
105         int n = keyboard.nextInt();  // size of the grid
106         keyboard.nextLine();
107
108         Agent[][] grid = new Agent[n][n];
109         PredPreyGrid pGrid = new PredPreyGrid(n);    // bio grid to work with.
110
111
112         System.out.print("How many rabbits? --> ");
113         int rabbits = keyboard.nextInt();
114         keyboard.nextLine();
115
116         for (int i = 0; i < rabbits; i++) {
117             int x = (int) Math.floor(50 * Math.random());
118             int y = (int) Math.floor(50 * Math.random());
119             Rabbit current = new Rabbit(grid, x, y);
120             grid[x][y] = current;
121         }
122
123         System.out.print("How many foxes? --> ");
124         int foxes = keyboard.nextInt();
125         keyboard.nextLine();
126
127         for (int i = 0; i < foxes; i++) {
128             int x = (int) Math.floor(50 * Math.random());
129             int y = (int) Math.floor(50 * Math.random());
130             Fox current = new Fox(grid, x, y);
131             grid[x][y] = current;
132         }
133
134
135         for(;;) {
136             for (int i = 0; i < n; i++)
137                 for (int j = 0; j < n; j++)
138                     if (grid[i][j] != null)
139                         grid[i][j].act();
140         }
141     }
142
143 }
144
```