SI@UCF Java GUI Test #2 Solutions

6/29/2016

1) (12 pts) Consider creating a CoffeeCup class that implements the Comparable interface (so CoffeeCup objects can be sorted). Each CoffeeCup object has a currentAmt and capacity, both integers measured in ounces. We define a CoffeeCup object to come before another one if it has more coffee in it. If two CoffeeCup objects contain the same amount of coffee, then the one with larger capacity comes before the other. For example, if we had a CoffeeCup A with 10 ounces of coffee with a capacity 20 ounces, a CoffeeCup B with 12 ounces of coffee with a capacity of 16 ounces and a CoffeeCup C with 10 ounces of coffee with a capacity of 24 ounces, the first coffee cup would be B, followed by C, followed by A. (Thus, A.compareTo(B) should return a negative integer and B.compareTo(C) should return a negative integer, while the opposite calls, B.compareTo(A) and C.compareTo(B) should return positive integers.) If two coffee cups have the same amount of coffee in them and have the same capacity, then they are equal.

```
public class CoffeeCup implements Comparable<CoffeeCup> {

    private int currentAmt;
    private int capacity;

    public CoffeeCup(int size) {
        currentAmt = size;
        capacity = size;
    }

    public int compareTo(CoffeeCup other) {

          if (this.currentAmt != other.currentAmt)
               return other.currentAmt - this.currentAmt;

          return other.capacity - this.capacity;

// Grading 4 pts for amount comparison
//         4 pts for looking at capacity if amounts are equal
//         4 pts for capacity comparison




    }
}
```

2) (6 pts) Add a drink method to the CoffeeCup class that takes in an amount of coffee to drink. If the current object has enough coffee, remove that many ounces. If the current object doesn't have enough coffee, empty the object. (Thus, if we called A.drink(10) and CoffeeCup A had 12 ounces of coffee in it, it should have 2 ounces afterwards. If a CoffeeCup B has 10 ounces of coffee in it and we call B.drink(16), the object B should have 0 ounces afterwards.)

```
public void drink(int amt) {

     if (amt > currentAmt)    // 3 pts for this adjustment
          amt = currentAmt;

     currentAmt -= amt;       // 3 pts regular case
}
```

3) (20 pts) The game Treasure Hunt is played on a square board of squares, where each square has some amount of treasure. The player may choose any 10 x 10 subsquare that appears on the board. Complete the method below so that it takes in the whole board (guaranteed to be square and at least 10 x 10) and returns the largest sum of the values among all possible 10 x 10 subsquares of that board. You may assume that all entries of board are non-negative and that their sum is less than 10^9. (Hint: Your code will have 4 nested loops.)

```
public static int max10SqSum(int[][] board) {

    int n = board.length;

    int res = 0;                                 // 1 pt

    for (int i=0; i<=n-10; i++) {                // 3 pts
        for (int j=0; j<=n-10; j++) {            // 3 pts

            int sum = 0;                         // 1 pt
            for (int x=i; x<i+10; x++)           // 3 pts
                for (int y=j; y<j+10; y++)       // 3 pts
                    sum += board[x][y];          // 2 pts

            if (sum > res) res = sum;            // 3 pts
        }
    }

    return res;                                  // 1 pt

}
```

4) (10 pts) Write a ***recursive function*** that takes in a positive real number, first, a positive real number, ratio and a positive integer, n, and returns the sum of a geometric sequence of n terms, starting with first, with a common ratio of ratio. For example, geoSum(2, 3, 4) should return 80, since 2 + 6 + 18 + 54 = 80.

```
public static double geoSum(double first, double ratio, int n) {

    if (n == 0)                       // 1 pt   (okay if 1 with
        return 0;                     // 2 pts   return adjusted)

    return first + geoSum(first*ratio, ratio, n-1);
    // 1 pt return, 2 pts adding first (or last)
    // 4 pts recursive call
}
```

5) (15 pts) Fill in the draw function below to do the following: using a for loop, draw 5 cyan squares, each with a random side length (in between 50 and 250, inclusive), and an x position and y position for the top left corner of each square with both coordinates in between 0 and 399, inclusive. You can assume that all necessary Java libraries have been imported.

```
public void draw(Graphics g){

    g.setColor(Color.CYAN);                     // 1 pt
    Random rand = new Random();                 // 2 pts
    for(int i = 0; i < 5; i++){                 // 1 pt
        int side = 50 + rand.nextInt(201);      // 3 pts
        int x = rand.nextInt(400);              // 2 pts
        int y = rand.nextInt(400);              // 2 pts
        g.drawRect(x, y, side, side);           // 4 pts
    }

}
```

6) (10 pts) In the draw method, load an image called “`boom.wav.jpg`” and display it in the top-left corner of the screen. Assume that the image is in the project’s root directory and all necessary Java libraries are imported.

```
public void draw(Graphics g){
    BufferedImage img;
    try{

        img = ImageIO.read(new File("boom.wav.jpg")); // 5 pts

    }catch(Exception e){}

    g.drawImage(img, 0, 0, null); // 5 pts
}
```

7) (5 pts) Consider the following code segment:

```
// assume this is initialized and non-empty
ArrayList<Rectangle> list;

public void draw(Graphics g){
    for(Rectangle r : list){
        g.drawRect(r.getX(), r.getY(), r.getWidth(), r.getHeight());
        r.setX(r.getX() + r.getWidth());
        r.setY(r.getY() - r.getHeight());
    }
}
```

What does this code do?

This code draws all of the rectangles in the ArrayList, moving them each time draw is called. The rectangles move in the x direction right one length of their width and in the y direction up one length of their height. (Grading: 1 pts for saying all rectangles get drawn, 2 pts for describing left-right movement, 2 pts for describing up-down movement.)

8) (20 pts) Fill in the missing code for the cball constructor, then make it so that in the draw function all of the cballs in the ArrayList move until they move off the screen of the 500x500 application window, then remove them. Draw each ball as a filled in circle.

```
public static ArrayList<cball> items = new ArrayList<cball>();

class cball{

    private int x;
    private int y;
    private int vx;
    private int vy;
    private int size;

    public cball(int myx, int myy, int myvx, int myvy, int mysize){
        this.x = x;
        this.y = y;                       // 5 pts - 1 pt per line
        this.vx = vx;
        this.vy = vy;
        this.size = size;
    }

    public void draw(Graphics g) {

        for(int i = 0; i < items.size(); i++){          // 5 pts
            cball c = items.get(i);
            g.fillOval(c.x, c.y, c.size, c.size);       // 5 pts
            c.x += c.vx;                                // 2 pts
            c.y += c.vy;                                // 2 pts
            if(c.x > 500 || c.y > 500 || c.x < 0 || c.y < 0){ // 4 pts
                items.remove(i);                                // 2 pts
                i--;                // grade is part of loop control
            }
        }

    }
}
```

9) (2 pts) In how many dimensions do 3D printers print objects? **3 (Give to all)**

Graphics Class Methods

```
// Sets the color of this graphics object to color c.
void setColor(Color c);

// Draws a rectangle with the current color with the top
// left corner at (topLeftX, topLeftY), with width width and
// a height of height. The x axis is horizontal and the y axis
// is vertical, with y coordinates increasing going down the
// screen. The top left corner of the display is (0, 0)
void  drawRect(int  topLeftX,  int  topLeftY,  int  width,  int
height);

// Fills an oval with the current fill color with the top
// left corner at (topLeftX, topLeftY), with width width and
// a height of height. The x axis is horizontal and the y axis
// is vertical, with y coordinates increasing going down the
// screen. The top left corner of the display is (0, 0)
fillOval(int topLeftX, int topLeftY, int width, int height);


// Draws as much of the specified image as possible with the top
// left corner at (x, y). If ob is null, no object is notified
// as more of the image is converted.
boolean drawImage(Image img, int x, int y, ImageObserver ob);
```

Color Constants

```
// Constant for the color cyan.
Color.CYAN
```

Random Class Methods

```
// Returns a random integer in between 0 and n-1, inclusive.
int nextInt(int n);
```

ImageIO Class Methods

```
// Returns a BufferedImage as the result of decoding a supplied
// File with an ImageReader chosen automatically from among
// those currently registered.
void read(File input);
```