CS1007: Object Oriented Design and Programming in Java

Lecture #10

Feb 16

Shlomo Hershkop
shlomo@cs.columbia.edu

Outline

- Advanced recursive tricks
- Divide and conquer
- Interfaces
- More graphics

- Reading : background book on recursion and chapter 4

Announcements

- Homework 2 out….the earlier you start the better prepared you will be for the midterm

Issues

From last class

- Can use recursion to solve some problems
  - Rule: there is nothing special about recursion…we can solve the same problem using a set of loops
- Should be aware of overhead of running the problem in a recursion fashion
- Remember 4 rules

Reminder

- Factorial recursion definition:

```
public int fact(n) {
If(n <1)
   return 1;
else
        return n*fact(n-1);
}
```

Can we do better?

- Idea, instead of waiting for the results so we can complete the call, end the recursion with a complete line of code i.e. that is free from local values

```
public int fact2(int n , int results){
   if(n<1)
        return results
   else
        return fact2(n-1,n*results)
}
```

Start with: fact2(n,1) (would use helper function with one arg for encapsulation

- How to do fib2?

- Will show fib1, and fib2 code

Divide and conquer

- We discussed bubble sort in last class
- Time to run = n^2
- Reason: we need to compare everything pair n times
- Can we do better? Is there a way of not having to compare everything like that

Question

- Has anyone heard of RANDSORT?

Random sort

- Take the item to sort
- Throw them in the air
- Pick them up and check if they are in order
- Actually very useful for specific tasks

Quick sort

- Idea: choose a random item
- Make 2 piles, everything less is on one side, everything greater on other side.
- When done 1st step, have found the spot fir the current item.
- Recursively call on each pile

No code!

- Quicksort(list)
- Choose pivot
- Run through list
- Have left, pivot, right piles
- Quicksort(left) + pivot + quicksort(right) is the sorted list.

Interface

- We've mentioned interfaces before
- We want to define some behavior that anyone who wants to behave in a certain way will indicate it by saying

Impliments interfaceYYYY

- It defines methods which need to be implemented that is nothing is implemented in interface class definition file
- Implementing class must supply implementation of all methods

Idea of an icon

- Want something which hints at some idea
- Small picture

The Icon Interface Type

```
public interface Icon
{
 int getIconWidth();
 int getIconHeight();
 void paintIcon(Component c,
  Graphics g, int x, int y)
}
```

Designing classes

- Remember that generally the interface is based on some design we have in mind
- Design can change

- What is the consequence: if you add a method OP() to this interface?

JOptionPane

- In general we work with window like containers and components
- Very useful to have a prepackaged class to quick display or fetch information

- JOptionPane

Example

- Use JOptionPane to display message:

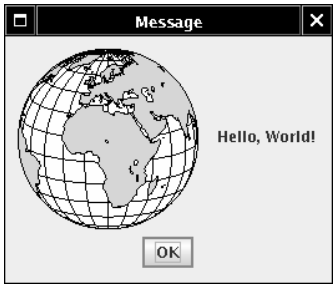
```
JOptionPane.showMessageDialog(null, "Hello,
  World!");
```

- Can you find the icon??

More complex

- Can specify an image file

```
JOptionPane.showMessageDialog(
 null,
 "Hello, World!",
 "Message",
 JOptionPane.INFORMATION_MESSAGE,
 new ImageIcon("globe.gif"));
```



Displaying an Image

- What if we don't want to generate an image file?
- Fortunately, can use any class that implements Icon interface type
- ImageIcon is one such class
- Easy to supply your own class

Graphics

- 2 approaches to using graphics in Java

1. Objects

   Use JLabel and place on the screen, Java takes care of all drawing

2. Raw Canvas

   Java hands you a way of drawing on a raw canvas object, use Graphics object to manipulate.

Marsicon.java

```
import java.awt.*;
02: import java.awt.geom.*;
03: import javax.swing.*;
04:
05: /**
06:    An icon that has the shape of the planet Mars.
07: */
08: public class MarsIcon implements Icon
09: {
10:    /**
11:  Constructs a Mars icon of a given size.
12:  @param aSize the size of the icon
13:    */
14:    public MarsIcon(int aSize)
15:    {
16:       size = aSize;
17:    }
18:
19:    public int getIconWidth()
20:    {
21:       return size;
22:    }
23:
24:    public int getIconHeight()
25:    {
26:       return size;
27:    }
28:
29:    public void paintIcon(Component c, Graphics g, int x, int y)
30:    {
31:       Graphics2D g2 = (Graphics2D) g;
32:       Ellipse2D.Double planet = new Ellipse2D.Double(x, y,
33:             size, size);
34:       g2.setColor(Color.RED);
35:       g2.fill(planet);
36:    }
37:
38:    private int size;
39: }
```