CS 121 – Intro to Programming:Java - Lecture 12
Announcements

Next Owl assignment up, due next week
Programming assignment up next week - will be a “work with a friend” assignment (two of these coming..)

Final exam: 12/19, 8 AM

Today: interfaces, abstract classes, (quick look at graphics?).

Last classes: graphics I, graphics II
The concept of an interface:

• A mechanism for specification

• We’ve seen them before: the Java API

• Encapsulation: working on a “need-to-know” basis

• An interface is a kind of contract

• Big principles -> reuse; hygiene
Here is the “Scoring” interface - it’s just two methods -- and not even that: the methods involved are “disembodied”

```java
public interface Scoring{
    public double getScore();
    public void setScore(double newScore);
}
```
public class CookieSeller implements Scoring {
    private String name;
    private double boxesSold;

    public CookieSeller(String n, double sold) {
        name = n;
        boxesSold = sold;
    }

    public String getName() {
        return name;
    }

    public double getBoxesSold() {
        return boxesSold;
    }
}
public void setName(String newName)
{
    name = newName;
}

public void setBoxesSold(double sold)
{
    boxesSold = sold;
}

public double getScore() // implements interface method
{
    return boxesSold;
}

public void setScore(double sold) // implements interface
{
    boxesSold = sold;
}
public class Scorefns {
   // contains methods that exploit the Scoring interface

   public static int scoreMax(Scoring[] theArray){
      // returns position of highest entry in array theArray

      int highPos = 0;
      for(int j = 1; j < theArray.length; j++){
         if (theArray[j].getScore() > theArray[highPos].getScore())
            highPos = j;
      }
      return highPos;
   }
}
public interface Directions{

    final int NORTH = 0;
    final int EAST = 1;
    final int SOUTH = 2;
    final int WEST = 3;

}

class BigTrip implements Directions {

    ......

    if (myDir() == NORTH) setDir(EAST);

Extremely important library interface: Comparable - it’s intended to model the “natural” ordering on elements in a class

A single method: compareTo

```java
public int compareTo(Object other);
```

Tricky:

It’s binary - you’re really comparing two objects, the calling object, and the parameter or argument object

Secondly: the parameter is completely general: it’s of type Object. This is something we’ll have to deal with.
The “meaning” of compareTo:

a, b are of some type (they’re cars, or strings, or tennis balls, or whatever)

a.compareTo(b) < 0  means -> a comes before b in natural ordering
a.compareTo(b) == 0 means a, b, equal in natural ordering
a.compareTo(b) > 0 means a comes after b in natural ordering.

Example: String implements Comparable; natural ordering - lexicographic

a = “cow”;  b = “snake”;  c = “walrus”;

a.compareTo(b) ->
b.compareTo(c) ->
c.compareTo(a) ->
c.compareTo(c) ->
public class Infant implements Comparable
{
    private String name;
    private int age; // in months

    public Infant(String who, int months){
        name = who;
        age = months;
    }

    ... 

    public int compareTo(Object other){ // natural ordering: age
        int b = ((Infant)other).getAge();
        int a = this.age;
        return (a-b);
    }
}
// now name, alphabetical, is natural ordering:

public int compareTo(Object other) {
    String b = ((Infant)other).getName();
    String a = this.name;
    return (a.compareTo(b));
}

In other words: we “hand off” the ordering decision to String..
Why is Comparable valuable?

It’s so pervasive - we’re always comparing things - that we can now build functionality around it:

Arrays.sort(words);

words: an array of Strings;

Arrays: a library class with array manipulation functionality

sort: put things in order

This combination works for any array for which the object type implements Comparable.

Note: there’s a version of sort, Arrays.sort(words, j , k), that sorts an array words from j through k.
Another notion: an abstract class

Notice: at one extreme -> full-blown, concrete classes
    at other extreme -> interfaces: everything disembodied

In the middle: abstract classes:

These are classes that

1) Cannot be instantiated

2) Generally have at least one method marked “abstract”

3) You make “real” classes out of them by extending them, providing bodies for the abstract methods.

4) One way to think of an abstract class: it’s a fancy stereo system, all ready to go, except that the component that provides the sound isn’t there, although the wire to the sound component is ready to be plugged in. You build a “concrete” system by extending what you have - you add a sound source.
import java.util.Scanner;
import java.io.*;

public abstract class LineReader{
    String fileName; // external file name
    Scanner scan; // Scanner for reading from file

    public LineReader(String f) throws IOException {
        fileName = f;
        scan = new Scanner(new FileReader(fileName));
    }

    public LineReader(File aFile) throws IOException {
        scan = new Scanner(aFile);
    }
}
public void readLines()
{
    while (scan.hasNext()){
        processLineLine(scan.nextLine());
    }
    scan.close();
}

public abstract void processLineLine(String line);
import java.io.*;
import java.util.*;

public class NewEcho extends LineReader {
    String fileName;
    public NewEcho(String f) throws IOException {
        super(f);
    }

    public void processLine(String line) {
        System.out.println(line);
    }
}

import java.util.*;
import java.io.*;

public class LineDriver{
    public static void main(String[] args){
        try{
            Scanner scan = new Scanner(System.in);
            System.out.println("Enter name of a text file");
            String fileName = scan.next();
            NewEcho r = new NewEcho(fileName);
            r.readLine();
        }catch(Exception e){e.printStackTrace();}
    }
}
public abstract class JobTimer {

    public abstract void doJob();

    // keeps track of time and calls doJob
    public void runJob() {
        // call the garbage collector to make more memory available
        System.gc();
        long s1 = System.currentTimeMillis();
        doJob();
        long s2 = System.currentTimeMillis();
        long runTime = (s2 - s1);
        System.out.println("running time in milliseconds: " +
                           runTime);
    }
}
public class AddTimer extends JobTimer {

    // how many times should the operation be performed
    public long numOperations = 10000000;

    // implementation of the abstract method
    public void doJob()
    {
        long k = 0;
        int result = 0, operand = 12345;
        while(k<numOperations)
        {
            result = operand + operand;
            k++;
        }
    }
}
public static void main(String[] args) {
    AddTimer a = new AddTimer();
    a.runJob();
}

running time in milliseconds: 186
And now let’s start graphics and GUIs, or Graphical User Interfaces..

GUIs are pervasive - think of almost any website, or restaurant computer system, or…

Java gets at GUIs in two ways: via Applets - these are programs that are suitable for transporting over the web, and displaying with a browser; and application programs, which you run directly (not through a browser).

The two concepts are pretty close. Here we’ll work on the latter.
good day!
import java.awt.*;
import javax.swing.*;

public class BabyGraphics{

    public static void main(String[] args){
        JFrame frame = new JFrame("Starter Work");
        Container c = frame.getContentPane();
        BabyPanel p = new BabyPanel();
        c.add(p); // add panel to frame's container
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

A JFrame is like a window frame..

It comes with a content pane, a container that holds things (surfaces..)

We’ll place on the surface something we can draw on / write on: a JPanel

Then we do just that: we write on it, draw on it, (and also color it).
import java.awt.*;
import javax.swing.*;

public class BabyPanel extends JPanel{

  public BabyPanel(){
    setPreferredSize(new Dimension(700,300));
    setBackground(Color.red);
  }

  public void paintComponent(Graphics g){
    super.paintComponent(g);
    g.drawLine(0,0,30,150);
    g.drawString("good day!",10,15);
  }
}