CS 121 – Intro to Programming:Java - Lecture 5

Professor Robert Moll (+ TAs)
CS BLDG 276 - 545-4315
moll@cs.umass.edu course web page below

http://twiki-edlab.cs.umass.edu/bin/view/Moll121/WebHome

Announcements

Third OWL assignment up, due next Tuesday; (GET GOING!)

Second Programming assignment up, due next Friday (9/19). Tip: read 2nd worked example at end of 3.5 VERY CAREFULLY

Midterm posted (W evening 10/15)

TA Office hours: M 12-4; TU 1-3:40; W 4-6; TH 1-4; F 12:30-4:30, in (back room next to) LGRT 223
Old MadDonald does Java
Old Macdonald had a farm // first two lines make chorus
ei, ei, o;
and on that farm he had a pig
ei ei o
With an oink oink here
And a oink oink there
Here a oink there a oink
Everywhere a oink oink
Old Macdonald had a farm
ei, ei, o;
- 
Old Macdonald had a farm
ei, ei, o;
and on that farm he had a dog
ei ei o (etc...)
Overall structure

Chorus
Pig verse
Chorus

Chorus
Dog verse
Chorus

Who are the players?
Chorus object
Verse objects - it’s parameterized (pig, dog, etc)
A coordinating “song” class - the driver
public class MacSong {
    public static void main(String[] args) {
        MacChorus m = new MacChorus();
        MacVerse p = new MacVerse("pig", "oink");
        MacVerse d = new MacVerse("dog", "woof");
        m.chorus();
        p.verse();
        m.chorus();
        m.chorus();
        d.verse();
        m.chorus();
    }
}

Three objects made - a chorus object, and two verse objects. The verse objects differ in that their attributes hold different values. The verse method exploits this to give different verses.

What about flow-of-control?
public class MacChorus{
    public void chorus(){
        // a method that serves MacChorus objects
        System.out.println("Old Macdonald had a farm");
        System.out.println("ei, ei, o;");
    }
}
public class MacVerse{
private String name;
private String noise; // attributes

public MacVerse(String animalName, String animalNoise){
    name = animalName;
    noise = animalNoise; }
public String getName(){return name;}
public String getNoise(){return noise;};

public void verse(){
    System.out.println("and on that farm he had a " + name);
    System.out.println("ei ei O");
    System.out.println("With a " + noise + " " + noise + " here");
    System.out.println("And a " + noise + " " + noise + "there");
    System.out.println("Here a " + noise + " there a " + noise);
    System.out.println("Everywhere a " + noise + " " + noise); } }
When a class definition is written, we (of course!) don’t know exactly which objects will be created according the definition - that is, we almost always have a blueprint with unspecified parts.

Example: the MacVerse blueprint does not identify an animal, nor does it identify a noise.

We use instance variables (attribute names) and parameters as place holders for these to-be-specified features.

“When she got home, Jill hung up her coat.”
Primitive Data Types

• objects are Java’s main currency
• Too tedious for them to be the only currency - making numbers a kind of object is a pain, however.
• Primitive data types integers (4), floats(2), char, boolean.

That these aren’t actually objects will turn out to be a pain, too.

Statement like these are fairly common:

```java
int count = 0;

boolean chewsTobacco = false;

char averageGrade = ‘C’; // note the single quotes
```
Strings - A very important class.

String greeting;
greeting = new String(“ola”);
greeting2 = new String(“howdy”);
greeting = greeting2;
System.out.println(greeting); // greeting now holds howdy

Some caveats:
1) Strings are not primitives (unlike float, double, int, etc). String is a class in the package java.lang

2) There’s a shorthand for String creation:
   greeting = “ola”; // works fine

3) As a class, String comes with extensive functionality
String pupName = “spot”;

int len = pupName.length(); // len assigned 4

char what = pupName.charAt(1); // what assigned ‘p’

char what = pupName.charAt(0); // what assigned ‘s’

String huh = pupName.concat(“less”); // spotless

String bigHuh = pupName.toUpperCase(); // SPOT

Where do I find out about the String class… (hold on)
Multiple views of a class

Most obvious view: the class source code:

```java
public class Infant{
    private String name;
    private int age; // in months

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

    public String getName(){return name;}

    public int getAge(){return age;}

    public void anotherMonth(){age = age + 1;}
}
```
Underlying principle: **Encapsulation**

Hide information (there’s too much of it - only keep track of what you really need to know)

And all you really need to know, in many cases, is how use a class - not the details of how a class is implemented (how it does what it does).
For example, we can get by just fine building and manipulating Infant objects if we just use the Infant class API, and don’t bother explicitly with the class at all.

(API: Application Program Interface - this simply gives the “what” of a class, not the “how” of a class)

(Example in text: TreeHouse class)

This IS big
The entire Java API is online, at http://java.sun.com/j2se/1.5.0/docs/api/

This is the link we use in the online book, but you can look at it any time, for example to look up the String class’s API, etc.
### Java™ 2 Platform Standard Edition 5.0

#### API Specification

This document is the API specification for the Java 2 Platform Standard Edition 5.0.

**See:**
- [Description](#)

## Java 2 Platform Packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>java.applet</strong></td>
<td>Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.</td>
</tr>
<tr>
<td><strong>java.awt</strong></td>
<td>Contains all of the classes for creating user interfaces and for painting graphics and images.</td>
</tr>
<tr>
<td><strong>java.awt.color</strong></td>
<td>Provides classes for color spaces.</td>
</tr>
<tr>
<td><strong>java.awt.datatransfer</strong></td>
<td>Provides interfaces and classes for transferring data between and within applications.</td>
</tr>
<tr>
<td><strong>java.awt.dnd</strong></td>
<td>Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.</td>
</tr>
<tr>
<td><strong>java.awt.event</strong></td>
<td>Provides interfaces and classes for dealing with different types of events fired by AWT components.</td>
</tr>
<tr>
<td><strong>java.awt.font</strong></td>
<td>Provides classes and interface relating to fonts.</td>
</tr>
<tr>
<td><strong>java.awt.geom</strong></td>
<td>Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.</td>
</tr>
<tr>
<td><strong>java.awt.im</strong></td>
<td>Provides classes and interfaces for the input method framework.</td>
</tr>
<tr>
<td><strong>java.awt.im.spi</strong></td>
<td>Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.</td>
</tr>
<tr>
<td><strong>java.awt.image</strong></td>
<td>Provides classes for creating and modifying images.</td>
</tr>
<tr>
<td><strong>java.awt.image.renderable</strong></td>
<td>Provides classes and interfaces for producing rendering-independent images.</td>
</tr>
<tr>
<td><strong>java.awt.print</strong></td>
<td>Provides classes and interfaces for a general printing API.</td>
</tr>
<tr>
<td><strong>java.beans</strong></td>
<td>Contains classes related to developing beans -- components based on the JavaBeans™ architecture.</td>
</tr>
</tbody>
</table>
A package: a bundle of classes with a common general purpose

There are dozens of standard packages in Java

You can make your own packages

When you run standard Java, only the package `java.lang` is automatically loaded (automatically available)

Classes in other packages are loaded as needed

This “load by need” feature makes Java the recycling engine we touted at beginning of term
The Scanner class is in the package java.util. You need to load it in - it’s not automatically available, as the classes in java.lang are.

The import statement loads classes (makes them available to your program):

```java
import java.util.Scanner;
```

Import statements go outside and before your class definitions.
```java
import java.util.Scanner;

public class Adder {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter 2 decimal numbers");
        double num1 = scan.nextDouble();
        double num2 = scan.nextDouble();
        System.out.println("Sum of " + num1 + " " +
                           num2 + " is ");
        System.out.println(num1 + num2);
    }
}
```
import java.util.*;

public class Paste{
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("enter first string");
        String s = scan.next();
        System.out.println("enter second string");
        String t = scan.next();
        System.out.println("paste them together");
        System.out.println(s + t);
    }
}

import java.util.Scanner;

public class VerseMaker{
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("enter animal name");
        String name = scan.next();
        System.out.println("enter animal noise");
        String noise = scan.next();
        MacVerse m = new MacVerse(name,noise);
        m.verse();
    }
}

java VerseMaker

enter animal name (donkey entered)
enter animal noise (hee-haw entered)

and on that farm he had a donkey
ei ei o
With an hee-haw hee-haw here
And a hee-haw hee-haw there
Here a hee-haw there  a hee-haw
Everywhere a hee-haw hee-haw