CS 335
Graphics, Image Processing, User Interface Design

2-3:15 TR  207 RGAN

Brent Seales
Course Goals

- Programming with *Java* and associated APIs
- Graphical User Interfaces (GUIs)
- Introduction to Image Processing
- Introduction to 2D Computer Graphics
Administrative Issues

- Course Webpage – check early and OFTEN
  http://dmn.netlab.uky.edu/~seales/cs335.html
- Mailing List
- Course Work
  - 5 Programming assignments
  - 4 Exercises (Problem Sets)
  - 2 Exams
- Assistant
  TBA
Introduction to Java
The Java Programming Language: Selected Web Resources

Java Homepage: www.java.sun.com
Java JDK 6 Update 2 java.sun.com/javase
Java Advanced Imaging (JAI):
   jai.dev.java.net

The Java Tutorials:
   java.sun.com/docs/books/books/tutorial/

On-line trade magazines, etc:
   www.javaside.com/
   www.javaworld.com/
Additional Materials

- The Java Advanced Imaging (JAI) libraries
- The Java 3D environment
- Java tutorial materials
- Java API documentation
Program Development Cycle

edit  compile  load  verify  execute

What does this mean for Java, which is an interpreted language?
Java is Interpreted

Source Code

Java source is text saved in a file with a .java extension. Java looks like C++.

Compile source using Java compiler

```
javac Myprogram.java
```

Compiler produces an output file, which ordinarily would be executable code (machine instructions).

Low-level "bytecode" file

```
Myprogram.class
```
Executing Java Programs

Standalone java program:

invoke the Java interpreter:

```
java Myprogram
```

(no extension; assumes .class)

Loader finds `Myprogram.class`, loads it into local memory, verifies it, and interprets (executes) it.

*(run examples)*
Example: A Complete Java Program

```java
import java.io.*;

public class Testclass
{
    public static void main( String args[] ) throws IOException
    {
        int count = 0;

        while ( count < 10 )
        {
            System.out.println("counter is " + count );
            count++;
        }
    }
}
```
New concepts with Java?
Interpreter/Emulator/API

Interpreter/emulator is an old idea:
   WINE: Windows interpreter for Linux OS
   SoftWindows: Windows interpreter for Silicon Graphics
   Executor: Macintosh emulator for Wintel PCs

Application Programmer Interface (API) is an old idea
   OpenGL: Graphics language
   Renderman
   etc.
New Possibilities

Combine API and Interpreter: network transparency via HTML

- Each hardware platform has specific implementation of API for local hardware
- Each platform can run interpreter
- Interpreter gives security from programs coming over network
- Applications can run anywhere
The Robust Java API

- The interface contains classes which can be declared directly or extended which do complex tasks:
  - Manage buttons
  - Manage text input windows
  - Display images
  - Read audio files
  - Run multiple threads in parallel

- Short Java programs can accomplish complex tasks via the API.
Interpreted Java: What about speed?

- API classes can perform well when implemented locally
- Most applets end up being a series of API calls
- Computers are faster
- Network is still the bottleneck for many applications
Why is Java the best?

- It isn't, necessarily!
- Includes powerful ideas
- First to get API + secure interpreted "platform independence" to be widely accepted
Summary

Java Development Environment:
edit, compile, load, verify, execute

Applets are different from standalone Java programs

Java combines powerful API (via complex classes) with interpreter and network (HTTP) interfaces.
Programming in Java

- Define data
- Calculate using data
- Output result

Java is object-oriented:
- Merge data and functions into object
- Invoke functions to operate on data

Java program must:
- Define data and functions (in a class)
- Invoke functions to compute things
Object-Oriented Programming: Classes

A **class** is an object definition, and includes data and functions on that data:

```java
public class MyCourseGrade {
    int pset1;
    int pset2;
    int final_exam;

    computeAverage() {
        // Method implementation
    }
}
```
Classes

Class: code which defines an object
Object: a variable (data + methods) which is an instance of a class
Java program: a bunch of class definitions, variables, etc.
Classes

One special class (the "mother of all classes") contains main(), and this is where flow of control begins:

```
class Test
    main
        Kernel of execution is here!

class AnotherClass (like MyCourseGrade)
    memberFunc1()
    memberFunc2()
```
Notes

- Java flow of control starts in `main()`, in whichever class `main()` is defined
- There can only be one class per file (unless you are defining subclasses)
- The filename must match the class name in a Java source file!
Example 1
A Java program with one class and one member called main():

```java
import java.io.*;

public class Test
{
    public static void main( String[] str )
    throws IOException
    {
        System.out.println("That’s it, folks!");
    }
}
```
Example 1: Scoping

```cpp
class Test {
    data (none defined)
    member functions
    main()
}
```
Example 2

Add another member function:

```java
import java.io.*;
public class Test {
    public static void main( String[] str )
        throws IOException {
        System.out.println("That it, folks!" );
    }
    public void doStuff() {
        System.out.println("doing stuff. ");
    }
}
```
Example 2: Scoping

class **Test**

- data (none defined)
- member functions
  - `doStuff()`
  - `main()`
Example 3

Define an object of class Test and make a function call

```java
import java.io.*;
public class Test
{
    public static void main( String[] str )
        throws IOException
    {
        Test t;  // t is of type "Test"
        t = new Test();  // allocate object
        t.doStuff();  // call member function
        System.out.println( "That it, folks!" );
    }
```
Example 3 (continued)

```java
public void doStuff()
{
    System.out.println( "I'm doing stuff." );
}
}
```

Notes:
• Static methods cannot access nonstatic class members directly
• `main()` must always be static
Test.java:
import java.io.*;
public class Test
{
    public static void main( String[] str )
    throws IOException
    {
        Stuff t;
        t = new Stuff();
        t.doStuff();
        System.out.println( "That it, folks!" );
    }
}

Put main class and a different class in separate files:
Example 4

**Stuff.java:**
```java
public class Stuff {
    public void doStuff()
    {
        System.out.println( "I'm doing stuff." );
    }
}
```

**Notes**
- One class per file
- To compile: `javac Test.java`
Java I/O

The **System** object provides a way to manage I/O from a more traditional "stream" (terminal window).

GUI-based I/O requires the `action()` method to deal with GUI mouse events.

The System object requires no `action()` method

But terminal I/O is inadequate in a browser-based (GUI) environment.
Summary of Some Basic Java Constructs

Everything is related to objects:

Data declaration:

```java
int i;    // declare i to be an int
Test t;  // declare t to be
         // an object of type Test
i = 0;   // set the int i equal to 0
t = new Test();
         // initialize t and allocate space
         // using a constructor
```
Java Constructs

Flow of Control:

Traditional, but with object-oriented syntax for function calls and member functions

Where control starts in the Applet class is important

Executable statements

Similar to C/C++: while, for, if/else, switch, etc.