Overview of the Course on Java

• Part 1: Java as strong language for:
  • Cross platform development
  • Graphical user interface development
  • Object-oriented coding
• Part 2: Java for Web development
  • Applets
  • Servlets
  • JSP
• Part 3: Java for client-server development
Overview of Part 1 of the Course

- Demystifying Java: Simple Code
- Introduction to Java
- An Example of OOP in practice
- Object Oriented Programming Concepts
- OOP Concepts -- Advanced
- Hints and for Java
- I/O (Streams) in Java
- Graphical User Interface Coding in Java
- Exceptions and Exception handling
Overview of this Presentation

- Background understandings
- How to set your system up for Java
- How to prepare your editor
- How to Compile and Run a Java application
  - A “hello world” type application
  - A program with a graphical user interface
  - Gathering systems information
- Some Java History and Terminology
Background
Background Overview

• This course anticipates that you:
  • Have knowledge of at least one a structured programming language
  • Have some understanding of object oriented design
  • Have some familiarity with Graphical User Interface (GUI) programming
Language Evolution

• The history of programming languages is a history of efforts to deal with every more complex programming task
• Linear atomic programming (machine coding) gave way to structured, and then modular, and then object based and oriented coding as mechanisms to deal with complexity.
• Object oriented languages use data hiding, attached methods, and inheritance to simplify the programming task
Object Oriented Design

- Object oriented design makes use of objects, and actions on objects as a natural way of thinking about large complex objects.
- In addition, because graphical interfaces are naturally thought of as a large collection of objects, OOAD and OOP are naturals when working with interface intensive applications.
- Object oriented design requires a shift from procedurally controlled thinking about programs to an event driven model of programming where messages control the flow of information.
- Keep in mind that in OOP, you need to assume your application exists in a sandbox that dispatches events.
When program involve the development of a Graphical User Interface (GUI), several new concerns are presented to the programmer:

- The program is represented as a set of windows
- Windows are either components or containers
  - Components generate events which must be responded to
  - Containers contain other containers or components and must have rules about layout
- A mapping must be created which describes how events generated by objects are to be handled
Demystifying Java
Setting up your environment (1)

- Load the JDK -- The JDK has two releases, Java1 and Java2
  - Java 1 is JDK 1.1.x
  - Java 2 is JDK 1.2.x and 1.3
  - You want Java 2 which is radically different than Java 1
- a self extracting zip file can be obtained from
  - http://www.javasoft.com/j2se/1.3/
- Extract the file to a directory like d:\jdk1.2
- Follow the installation instructions available at the site:
  - On NT and 2000 -- right click on MyComputer, select properties, and set the path variable to include the JDK bin directory -- in our example d:\jdk1.2\bin
  - Win 98 -- edit c:\autoexec.bat and add the path to the end of the PATH command there
  - Do the same for the CLASSPATH variable, also, be sure to add “.” to the CLASSPATH
Setting up your environment (2)

• Load the JDK Documentation
  • extract the file using win zip to the Java Directory -- in our example d:\jdk1.2
  • This will extract all the files to a docs subdirectory
  • Provide for easy access to the documentation
    • add the file d:\jdk1.2.2\docs\api\index.html to the bookmarks of your browser, or
    • Add a shortcut to the document to your desktop – this will allow quick access to the voluminous documentation
Setting up an Editor (1)

You have three choices for editing and compiling Java files
- Basic command line
- Context sensitive editor
- Full IDE

Using the command line:
- edit a .java file (e.g. myclass.java) using notepad
- Open a command/dos window on the directory of the file
- compile the files using “javac”
  - e.g. javac myclass.java will produce myclass.class
- interpret the resulting class file using “java”
  - e.g. java myclass  will run the program
Setting up an Editor (2)

- Install your own editor (e.g. Realj)
  - Follow the installation instruction
    - It is best to install Realj after the JDK
    - Make sure your classpath includes “.”
    - Make sure Realj picked up the JDK location
      - Check preferences
  - Use the editor to write a program
    - Note that it is context sensitive
  - Compile and run the program from within the editor
- Sun makes an integrated development environment available
  - It is called “forte” and available at:
  - Its use is beyond the scope of this introduction.
A First Code Example

public class start{
    public static void main(String[] args){
        System.out.println("Starting Java");
    }
}

• one and only one main per application
• file must be named to match main class (start.java)
  • the naming is case sensitive
• each class in .java file become separate .class files
• args is an object, thus no argc, arg.len instead
• arg[0] is the first parameter, not the executable
• System.out is a printstream (to the console)
A Second Code Example

- Note the use of imports to access predefined classes and methods
- Note the subclassing of the JFrame from Swing
- Note the "implements" to get a "window listener"
- Note the use of methods
- Question
  - How many class files will be produced?
  - Where did the system object come from?
  - What do you think setSize does and what do the numbers mean?
  - What pattern do you see in capitalization?
import java.awt.event.*; // required for WindowListener
import javax.swing.*; // required for JFrame
public class ClosingFrame{
    public static void main(String[] args)
    {
        BFrame x = new BFrame();
        x.show();
    }
}

class BFrame extends JFrame implements WindowListener
{
    BFrame()
    {
        setTitle("A frame that listens");
        setSize(200,200);
        addWindowListener(this);
    }
    public void windowClosing(WindowEvent e) {
        System.exit(0); }[some code omitted here]
Learning About this Platform

- Java is a cross platform language, system information is made available through encapsulation.
- You should keep in mind the objects that encapsulate information.
- One of the places where you can find system information is the Properties class:
  - The Java version and classpath
  - The operating system name architecture, and version
  - User's account name, home directory, and current working directory
- The Toolkit has some information about the screen.
- Finally, the Graphics Environment has some information about fonts.
import java.util.*; //for the Properties class
import java.awt.*; //for GraphicsEnv and Toolkit

public class properties{
public static void main(String[] args){
    Properties p = System.getProperties();
    p.list(System.out);
    String[] fnts = GL.getAvailableFontFamilyNames();
    for (int i=0;i<fnts.length;i++)
    {
        System.out.println(fnts[i]);
    }
    Toolkit TK = Toolkit.getDefaultToolkit();
    Dimension scr_dim = TK.getScreenSize();
    System.out.println(scr_dim.height);
    System.out.println(scr_dim.width);
    System.exit(0);
}
}
A Challenge Exercise

• Can you improve this piece of code?
• See if you can find and read the documentation on the classes
  • Toolkit in java.awt
  • Properties in java.util
  • GraphicsEnvironment in java.awt

• Is there any other environment data you can find?
• Can you figure out how to organize the data?
• Can you add some headers and formatting information?
Java Terminology
Overview of Terminology

- The history and roots and future of Java
- Concepts and Terminology
  - Concept of class
  - The Java Virtual Machine (JVM)
  - Containers and components
- Other features of Java
History

• The Java language has roots and four branches.
• The roots are
  • C/C++
  • Embedded computing
  • The Web
• The branches are
  • Web Applications
  • Cross Platform Development
  • Advanced Network applications
Roots

- The original project at Sun was called the Green project
  - it was aimed at the development of a small language to control devices
  - it used a C-like language and depended on a virtual machine at the target
  - as it ended up, intelligent consumer appliances have been slow to develop
- Java was designed to be a true object-oriented language with C syntax
  - most of the weaknesses of C have been removed -- e.g. pointers
  - C++ structure has been improved upon -- interfaces
- The Web offered a new opportunity to market Java
  - Java provided an opportunity for dynamic pages
  - It offloaded the growing server burden
  - Clients (browsers) were just right for virtual machines
Branches

- Java the Application Development Language
  - a cross platform language -- write once, run anywhere
  - a mature object oriented language with a widely known syntax
  - a rich class library to support sophisticated applications
    - Remote Method Invocation
    - Java DataBase Connectivity (JDBC)
    - Java Beans
- Java the Web Language
  - secure applets
  - authenticated servlets
  - secure communications via SSL
- Java the Page Script
- Java the Next Generation -- JINI
The Concepts of Class

- OOPL are based on Classes
  - classes can be thought of as templates and as datatypes
  - an instance of a class is an object
  - the object is a complex variable an instance of the template or datatype in which specifics can be filled in

- Classes can be broadly or narrowly defined
  - a "account" class is broader than an "interest account" class
  - an "interest account" class is broader than a "money market savings account" or "CD account"
  - a "CD account" class is broader than a "CD - six month compounded interest account" class or a "CD-yearly compounded interest account" class
  - a subclass inherits the methods and fields of the parent and add additional specialize fields and methods
Java Terminology related to Classes

• The data associated with a class are called “fields”
• The associated functions are called “methods”
• The methods which create new instances (objects) from a class are called “constructors”
• A Class is generally created by “extending” an existing class
• A Class may “extend” only one class.
• A Class may “implement” one or more special classes, called interfaces to add additional features.
• Classes are collected in groups called packages.
• A class or group of classes is made available to a program by “importing” the package
The Java Virtual Machine

- The Java Virtual Machine (JVM) is a kind of “software cpu” that runs the compiled java program, translating the instructions into native CPU instructions.
- As a result, several things occur:
  - Java programs are compiled and have the potential to run very efficiently depending on the Java Virtual Machine.
  - Java programs can run on multiple platforms because the virtual machine translates the compiled program to native language.
  - The programs are considered secure in that the virtual machine can be restricted in which instructions it will translate.
Java Terminology related to the JVM

• The JDK is the Java Development Kit and it is distributed free of charge by SUN.
• The JDK consists of all the packages and tools needed to make Java programs.
• The JDK tools used most frequently are:
  • javac which is the java compiler which converts source code to byte code
  • java which is the JVM distributed with the JDK -- also called the java interpreter
  • appletviewer which is the JVM distributed with the JDK for interpreting applets
  • jar which is the program to create and manage java archives -- collections of compiled classes
Graphical User Interfaces (GUIs)

- Java is used to develop graphical user interfaces
- The interface is an important component of many applications
- Java has a rich set of classes and to support development of the user interface
- Because of the speed with which Java was adopted, both the classes that are used for the interface objects and the events which occur have undergone very rapid and dramatic evolution
- Some of what was developed historically is no longer considered to be valid -- it is deprecated
Java Terms related to the GUI

• The Java package for graphical user interface components prior to Java 1.2 was “java.awt”
• While some classes from java.awt continue to be used, GUI classes are now found in “javax.swing”
• Events that occur in interface objects are handled by methods that are called “listeners”
• There are about a dozen listeners defined in “java.awt.event”. They include “WindowListener” and “ActionListener”
• These event classes are “implemented” for a class that is defined in Java
Other Java Developments and Terms

- the fact that Java source code compiles to a separate bytecode file made it hard to share code (especially for applets)
- "JAR” file -- Java ARchive makes sharing sets of compiled files easier. (An archive is simply a Unix name for a zip file)
- "JavaBeans” are visual components that can be used in a standard way in apps. Traditionally, they are visual components that can simply be “dropped” into an application.
“Remote Method Invocation” or “RMI” is the Java name for a set of classes that enable the user to develop client-server applications with a greater focus on the functional rather than the communications design.

JDBC allows database connections much like those allowed by ODBC.

JINI, pronounced "Genie" is the newest development. It is intended to extend RMI so that devices can find each other and register services in a network environment. It is like a glorified plug and play capability.