UCLA PIC 20A
Java Programming

- **Instructor:** Ivo Dinov,
  Asst. Prof. In Statistics, Neurology and Program in Computing
- **Teaching Assistant:** Yon Seo Kim, PIC

University of California, Los Angeles, Summer 2002
http://www.stat.ucla.edu/~dinov/

PIC 20 A – Course Organization

- *http://www.stat.ucla.edu/~dinov/courses_students.html*
- Summer2002\HTMLs\PIC20A.html
- Java Demo: C:\Ivo.dir\LONI_Viz\LONI_Viz_MAP_demo\run.bat

Chapter 1 – Getting Started

- What is the Java platform and what it can do?
- How to compile and run two simple programs on the Windows, the UNIX/Linux or the Mac OS X platforms.
- Software Development Kits (SDKs) provided by Sun Microsystems include a minimal set of tools to let you run and compile your programs.
- Advanced developers are encouraged to use a professional Integrated Development Environment (IDE).

Chapter 1 – Getting Started

- Java is a high-level language that is characterized by:-
  - Simple
  - Robust
  - High performance
  - Object oriented
  - Secure
  - Multithreaded
  - Distributed
  - Architecture neutral
  - Dynamic
  - Interpreted
  - Portable
- Most programming languages, you **either** compile **or** interpret a program to run it on your computer. In Java a program is both **compiled** and **interpreted**.

Chapter 1 – Getting Started

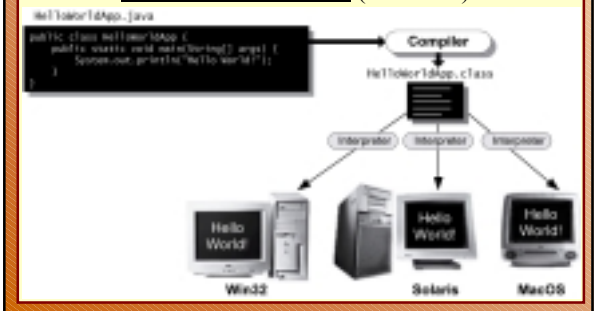
- Bytecodes is the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter is an implementation of the Java VM.
- **Write once, run anywhere**. As long as a computer has a Java VM, the same Java program should run on Windows, Solaris, or Mac.

Java Platform

- A **platform** is the hardware or software environment in which a program runs (W2K, Linux, Solaris, MacOS). Most platforms can be described as a combination of the operating system and hardware.
- Java platform is **software-only platform** that runs on top of other, hardware-based platforms and has these 2 components:
  - The *Java Virtual Machine* (Java VM)
  - The *Java Application Programming Interface* (API)

Java Platform

- The *Java Virtual Machine* (Java VM)



Java VM & Java API

- The **Java VM** is the base for the Java platform and is ported onto various hardware-based platforms.
- The **Java API** is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*.

Native vs. Portable Code

- **Native code** is code that, after you compile it, runs on a specific hardware platform.
- As a platform-independent (**portable**) environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

Java Applets & Applications

- The most common types of programs written in the Java programming language are **applets** and **applications**.
- An **applet** is a program that adheres to certain conventions that allow it to run within a Java-enabled web browser.
- Example: C:/Ivo.dir/UCLA_Classes/Applets.dir/DiceApplet.htm
- An **application** is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network.
- Servers Ex: Web servers, Mail servers, Print servers.
- Servlets, similar to applets, are run-time extensions of applications. Instead of working in browsers, though, **servlets run within Java Web servers**, configuring or tailoring the server.

Java 2 Platform SE v 1.4

First Java Program – HelloWorldApp

- Your first program, HelloWorldApp , will simply display the greeting "Hello World!" To create this program, you complete each of the following steps.
- **Create a source file.** A source file contains text, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and to edit source files.
- **Compile the source file into a bytecode file.** The compiler takes your source file and translates the text into instructions that the Java VM can understand. The compiler converts these instructions into a bytecode file.

First Java Program – HelloWorldApp

```
/**
*    The HelloWorldApp class implements an application
*    that displays "Hello World!"to the standard output.
*/
public class HelloWorldApp  {
      public static void main(String [] args))  {
                //Display "Hello World!"
                System.out.println("Hello World!");
      }
}
```

Case Sensitive

Do this example in a DOS shell or Explorer
javac ***HelloWorldApp.java***
java ***HelloWorldApp***

First Java Applet – HelloWorld

- **Applet** called HelloWorld , displaying the greeting "Hello World!" Unlike HelloWorldApp , however, the applet runs in a Java-enabled Web browser, such as the HotJava TM browser, Netscape Navigator, or Microsoft Internet Explorer.
- Create a source file, compile the source file, and run the program. However, unlike for an application, you must also create an HTML file.

First Java Applet – HelloWorld.java

```
import    java.applet.Applet;
import    java.awt.Graphics;
public class HelloWorld extends    Applet     {
      public void paint(Graphics g)     {
                //Display "Hello world!"
            g.drawString("Hello world!",50,25);
      }
}
```


First Java Applet – HelloWorld

- **HTML file to accompany your applet**. HelloWorld.html
- <HTML>
- <HEAD>
- <TITLE>A Simple Program</TITLE>
- </HEAD>
- <BODY>
- Here is the output of my program:
- **<APPLET** CODE="**HelloWorld.class**"
  WIDTH=150 HEIGHT=25 >
- **</APPLET>**
- </BODY>
- </HTML>

Running Java Applet – HelloWorld

- Compiling:
- **javac HelloWorld.java**
- The compiler should generate a Java bytecode file, **HelloWorld.class**
- Although you can use a Web browser to view your applets, you may find it easier to test your applets by using the simple appletviewer application that comes with the Java platform. To view the elloWorld applet using appletviewer enter at the prompt:
- **appletviewer** HelloWorld.html
- **netscape** HelloWorld.html

Exceptions …

- **Exception in thread "main " java.lang.NoClassDefFoundError:HelloWorldApp**
- If you receive this error, the interpreter cannot find your bytecode file, HelloWorldApp.class .
- One of the places java tries to find your bytecode file is your current directory. So, if your bytecode file is in C drive, you should change your current directory to that. To change your directory, type the following command at the prompt and press Enter :
- **cd c:**
- The prompt should change to C:. If you enter dir at the prompt, you should see your **.java** and **.class** files. Now again enter
- **java HelloWorldApp**
- Still have problems, change your CLASSPATH variable.
- **set CLASSPATH=**

First Java Application Program

- **HelloWorldApp** , displays the greeting "Hello World!" To createthis program:
- **Create a source file** – text written in the Java that others can understand. Use any text editor to generate/edit source files – ***.java**.
- **Compile the source** file into a bytecode file – **javac** translates the source/text into instructions that the Java Virtual Machine can understand, bytecode file – ***.class**.
- **Run the program** contained in the bytecode file via Java interpreter – carries out the instructions by translating them into instructions that your computer can understand.
- **E.g.,**C:\Ivo.dir\UCLA_Classes\Summer2002\PIC20A\Examples\JavaTutorialExamples\getStarted\application\example\HelloWorldApp.java

First Java Applet

- **HelloWorldA**, displays the greeting **Hello World!** in a java enabled browser
- **Create a source file** – notepad HelloWorld**.java**
  notepad Hello**.html**.
- **Compile the source** – **javac** HelloWorld**.java**
  makes – HelloWorld**.class**.
- **Run the program** – appletviewer Hello**.html**.
- **E.g.,**C:\Ivo.dir\UCLA_Classes\Summer2002\PIC20A\Examples\JavaTutorialExamples\getStarted\applet\example\HelloWorld.java && Hello.html

HelloWorld – Program description

- Kinds of Comments:
- /* *text* */
- Compiler ignores everything from opening **/*** to closing ***/**.
- /** *documentation* */
- This style indicates a documentation comment *(doc comment*, for short). Compiler ignores all the text within the comment. The SDK **javadoc** tool uses doc comments to automatically generate docs.
- **Example**:C:\Ivo.dir\UCLA_Classes\Summer2002\PIC20A\Examples\JavaTutorialExamples\getStarted\applet\example\doc\index.html
- // *text*
- The compiler ignores everything from the //to the end of the line.

HelloWorld – Program description

- Defining a Class:
- **class HelloWorldApp {**

```
    public static void main(String [] args)){
        System.out.println("Hello World!");   // Display the string
    }
}
```

- A ***class*** is the basic building block of an object-oriented language, such as Java. A class is a blueprint that describes the state and the behavior associated with ***instances*** of that class. When you ***instantiate*** a class, you create an ***object*** that has the same states and behaviors as other instances of the same class.
- The state associated with a class or an object is stored in ***member variables***.

HelloWorld – Program description

- Defining a Class:
- The behavior associated with a class or an object is implemented with ***methods***, which are similar to the functions or procedures in procedural languages, such as C/FORTRAN/Pascal.
- Ex. Of a class is a recipe—for making a cake. It's a blueprint for making a specific **instance** of the recipe. Your attempt to make a cake is one **instance** of the recipe, and someone else's attempt to bake the same cake is another **instance**.
- Ex. of a class that represents a **rectangle**: The class defines variables for the **origin**, **width**, and **height**. The class might also define a method that calculates its area/circumference.

  An instance of the rectangle class, a **rectangle object**, contains the information for a specific rectangle, such as the dimensions of the floor of your office or the dimensions of this page.

HelloWorld – Program description

- Main method of a Class:
- The entry point of every Java application is its **main** method. The interpreter first invokes the **main** method defined in that class. The main method controls the initial flow of the program.
- /** The HelloWorldApp class implements an application that
  * simply displays "Hello World!"to the standard output.
  */
  class HelloWorldApp {
  public static void main(String [] args)){
  System.out.println("Hello World!"); // Display the string.
  }
  }

HelloWorld – Program description

- Main method of a Class:
- Every application must contain a main method declared like this:

public static void main(String [] args))

- 3 modifiers:
- **public** : Allows any class to call the main method
- **static** : Means that the main method is associated with the HelloWorldApp class as a whole instead of operating on an instance of the class
- **void** : Indicates that the main method does not return a value
- Each application must contain a main method. When invoked, the interpreter starts by calling the class's main method, which then calls all the other methods.

HelloWorld – Program description

- Main method of a Class:
- The main method accepts a single argument: an array of elements of type String , like this:

public static void main(**String [ ] args**)

- This array is the mechanism through which the Java Virtual Machine passes information to your application. Each String in the array is called a ***command-line argument***. It let users affect the operation of the application at run-time, without recompiling it.
- The HelloWorldApp application, is very simple and ignores its command-line arguments.

HelloWorld – Program description

- Using Classes & Objects:
- Because HelloWorldApp is such a simple program, it doesn't need to define any classes except HelloWorldApp . However, the application does ***use*** another class, **System**, that is part of the Java API. The System class provides system-independent access to system-dependent functionality.
- The System class contains *standard output stream*—a place to send text, usually refers to the terminal window the Java interpreter runs.
- **Caution!** Using the standard output stream isn't recommended in 100% Pure Java programs. However, it's fine to use during the development cycle.

HelloWorld – Program description

- Using Classes & Objects:
- **System.out.println("Hello World!");**
- Ex. of a *class variable* and an *instance method*.

System.out .println("Hello World!");

- The construct System.out is the full name of the **out** variable in the System class. The application never instantiates the System class but instead refers to out directly through the class. The reason is that **out** is a *class variable*—a variable associated with a class rather than with an object.
- The Java Virtual Machine allocates a class variable once per class, no matter how many instances of that class exist.

HelloWorld – Program description

- Using Classes & Objects:
- **System.out.println("Hello World!");**
- **System**'s **out** variable is a ***class variable***, it refers to an instance of the Print-Stream class (another Java API-provided class that implements an easy-to-use output stream). When it is loaded into the application, the System class instantiates PrintStream and assigns the new **PrintStream** object to the **out** class variable.
- System.**out.println** ("Hello World!");
- Calls one of **out**'s instance methods. An instance method implements behavior specific to a particular object—an instance of a class.

HelloWorld – Program description

- Using Classes & Objects:

System.out.println("Hello World!");

- Java also has instance variables. An **instance variable** is a member variable associated with an object rather than with a class. Each time you instantiate a class, the new object gets its own copy of all the instance variables defined in its class.
- We'll discuss these further in ch. 4 …

HelloWorld – Applet description

- Importing Classes & Packages:
- **HelloWorld.java** begins with two import statements that import the Applet and Graphics classes:

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld extends Applet    {
        public void paint(Graphics g)       {
                g.drawString("Hello world!",50,25);
        }
}
```

- By importing classes or packages, a class can easily refer to classes in other packages. Packages are used to group classes, similar to the way libraries group C functions.

HelloWorld – Applet description

- Importing Classes & Packages:

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorld extends Applet    {
        public void paint(Graphics g)       {
                g.drawString("Hello world!",50,25);
        }
}
```

- Importing the Applet and Graphics classes lets the program refer to them later without any prefixes. The **java.applet.** and **java.awt.** prefixes tell the compiler which packages it should search for the Applet and Graphics classes.

HelloWorld – Applet description

- Defining an Applet Sub-Class:
- import java.applet.Applet;
  import java.awt.Graphics;

```
public class HelloWorld extends Applet {
    public void paint(Graphics g){
        g.drawString("Hello world!",50,25);
    }
}
```

- Every applet must define a subclass of the Applet class. Applets inherit a great deal of functionality from the Applet class, ranging from the ability to communicate with the browser to the ability to present a graphical user interface (GUI).

HelloWorld – Applet description

- Implementing Applet Methods:
- import java.applet.Applet;
  import java.awt.Graphics;
  public class HelloWorld extends Applet {
  public void **paint**(Graphics g){
  g.drawString("Hello world!",50,25);
  } }
- There's a single method here: **paint** . Every applet should implement at least one of the following methods: **init , start , or paint** . Applets do *not* need to implement a **main** method.
- Applets are designed to be included in HTML pages. Using the <APPLET>HTML tag, you specify (at a minimum) the location of the Applet subclass and the dimensions of the applet's on-screen display area. The applet's coordinate system starts at (0,0), which is at the upper-left corner of the applet's display area.

HelloWorld – Applet description

- Running an Applet:
- When it encounters an <APPLET>tag, a Java-enabled browser reserves on-screen space for the applet, loads the Applet subclass onto the computer on which it is executing, and creates an instance of the Applet subclass.
- <HTML> <HEAD> <TITLE>A Simple Program </TITLE> </HEAD>
  <BODY>
  Here is the output of my program:
  <APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
  </APPLET>
  </BODY></HTML>