Lecture 05: Languages

Sierra College
CSCI-12
Spring 2015
Mon 02/09/15

Announcements

- **Schedule**
  - Reminder: no classes NEXT Monday 2/16 (President's Day)
- **Past due assignments**
  - HW03: Why Code, accepted thru Fri 2/13 @ 11pm
- **Current assignments**
  - LAB04: Hello Again, due Tues 2/10 @ 11pm
- **New assignments**
  - HW05: Numbers, due Fri 2/13 @ 11pm
    - Perform several number conversions (decimal/binary/hex)
    - Don't just give the answer, show all the steps you took

Lecture Topics

- **Last time**:
  - Models
  - Number systems
  - Number conversion examples
- **Today**:
  - More definition of “programming”
  - A short history of computing language evolution
  - Initial overview of object-orientation
  - An intro to Java (history, features, advantages)

What Is Programming?

- So, here we are in an introductory programming class
- But, have we formally defined that term yet??

- Truthfully, we can't!
  - Formal CS definitions inevitably vary among sources
  - But, all definitions will likely use similar words to dance around some common ideas

- ***What comes to mind when <u>you</u> think of "programming"??***

Programming Stereotypes

Some Definitions of Programming

- CS-10 textbook *(Computer Science Illuminated, Dale & Lewis, 3rd ed.)*:
  - **Programming language**: "A set of grammar rules, symbols, and special words, used to construct a program – that is, to express a sequence of instructions for a computer"
  - **Program**: "A sequence of instructions written to perform a specified task"
- Wikipedia:
  - **Computer programming**: "… the comprehensive process that leads from an original formulation of a computing problem to executable programs… The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solve a given problem"
- Instructor (from last lecture):
  - "Programming is the science/art of describing, in very specific terms, WHAT you want done, in such terms that the computer itself can understand and execute"

Some Common Aspects of Programming

- Programming uses **clear, logical thinking** to attack a problem
  - Part of what makes CS-trained individuals so valuable in current economy
- Programming develops precise, repeatable, step-by-step instructions to solve that problem (**algorithms**)
- Programming expresses those instructions in the **syntax** of a **particular programming language**, to instruct a computer to solve that problem
- Programming is part methodology, part creativity
  - Art and science
- Programming uses **existing bodies of knowledge** and certain **standard, proven approaches** (Computer Science)
- Programming **reuses working solutions and components**, where appropriate (software reuse, object-orientation)

Some Techniques of Programming

- **Problem definition**: clearly identify what you KNOW versus what you DON'T KNOW (or, what you are trying to ACHIEVE)
- Turn a large, complicated problem into lots of smaller, simpler ones **("divide and conquer")**
- **Transform** new problems into familiar, solved ones
- **Reuse past, known solutions** intelligently ("don't reinvent the wheel")
- **Code reuse**: leverage code that already exists and is known to work (use existing classes/objects)
- Develop new, step-by-step approaches (**algorithms**)

Some Loose Terminology Equivalence

For the end result:

- Programs
- Software
- Code
- Applications
- Apps

For the process of creating it:

- Programming
- Software development
- Software engineering
- Coding
- Writing code
- Writing a program
- Application development
- App development

Type of Programming Languages

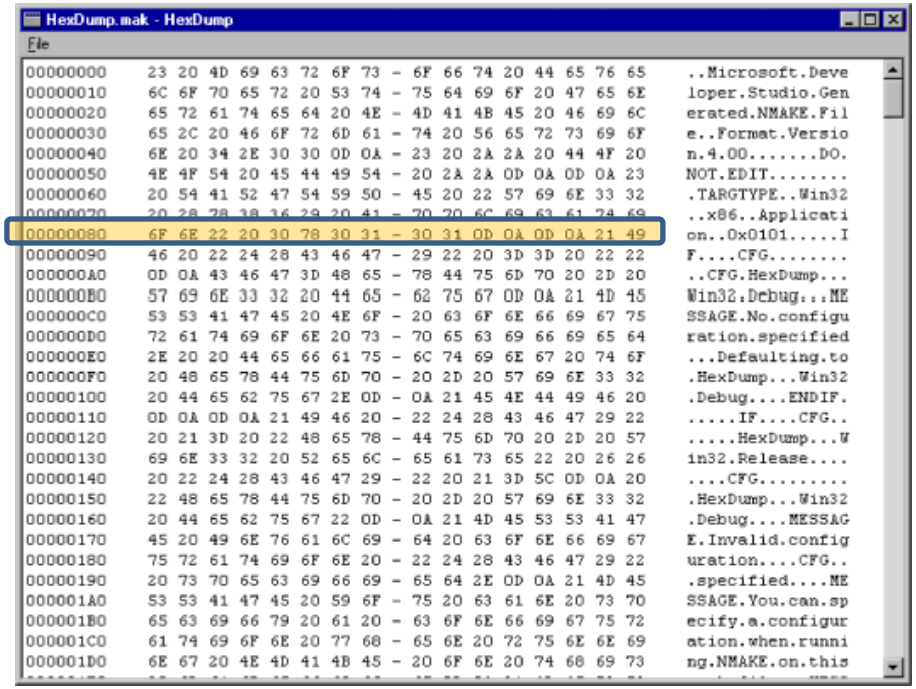
- Applications are written in (one or more) programming languages
- There are 3 types of programming languages:
  - Machine language
  - Assembly language
  - High-level language
- The 3 types represent the historical progression of programming languages over time

Machine Language

- All that was available to programmers in the early days of computing
- Written specifically for a particular CPU's native instruction set
- All instructions and data are simply 1s and 0s
- Often viewable as a "hex dump" of memory contents
- Difficult to write and not portable between computers
- No longer frequently used for programming
- But always present at the lowest hardware (CPU) level

Machine Language Example

Sample hex dump



Equivalent memory depiction

0x00000080	0110 1111	6F
0x00000081	0110 1110	6E
0x00000082	0010 0010	22
0x00000083	0010 0000	20
0x00000084	0011 0000	30
0x00000085	0111 1000	78
0x00000086	0011 0000	30
0x00000087	0011 0001	31
0x00000088	0011 0000	30
0x00000089	0011 0001	31
0x0000008A	0000 1101	0D
0x0000008B	0000 1010	0A
0x0000008C	0000 1101	0D
0x0000008D	0000 1010	0A
0x0000008E	0010 0001	21
0x0000008F	0100 1001	49

Assembly Language

- One step up from machine language on the computing ladder
- Uses CPU-specific, short mnemonics for instructions
- Allows use of symbolic names for variables and memory addresses
- Programs called assemblers translate assembly language into machine language
  - One-to-one statement correspondence between machine ←→ assembly
- Easier to program in than machine language, but still not portable

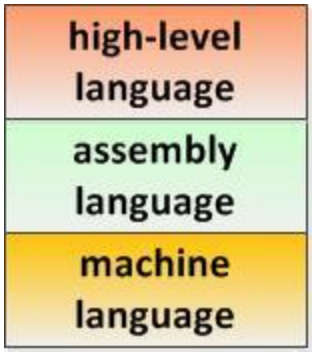
Assembly Language Example

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2


 C000                   ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START   LDS    #STACK

                *************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                *************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CARRY
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                *************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

- This example is for the Motorola 6800 processor
- Right side is the assembly language
- Left side shows memory addresses and their contained assembly instructions
- Some (likely) instructions?
  - LDA = load
  - BLE = branch if less than/equal to
  - JMP = jump to
  - CMP = compare

High-Level Languages

- Final evolutionary improvement upon assembly language
- Highly symbolic, much more human-friendly ("English-like")
  - **MUCH** easier to develop in than machine or assembly
- Single, readable statements replace multiple lines of assembly or machine language
- Often portable across CPUs, or at least nearly so
  - A program called a compiler turns the high-level language into assembly and then into machine language
- May be compiled, interpreted, or some of both (Java)
- Hundreds have been developed, some have achieved more widespread usage than others
  - Try googling: ***high-level languages list*** (or see Wikipedia URL in lecture module)



High-Level Language Examples

- All languages have strengths and weaknesses, and certain niches
- Here is an incomplete list of some you may have heard of:
  - **Java ,**C++ : general-purpose applications (Java → Android programming)
  - C: embedded systems, foundation of UNIX
  - Objective-C: iPhone programming
  - PHP, Perl, Python, ASP: back-end web programming
  - JavaScript: web page back-end programming (NO RELATION TO JAVA!)
  - Fortran: scientific applications
  - COBOL: business applications
  - Ada: sometimes a DOD contracting standard
  - Pascal: (was?) a common teaching language
  - Lisp, Prolog: artificial intelligence
  - See also: http://en.wikipedia.org/wiki/Lists_of_programming_languages
- Shortcomings in languages (real or perceived) often drive new language development

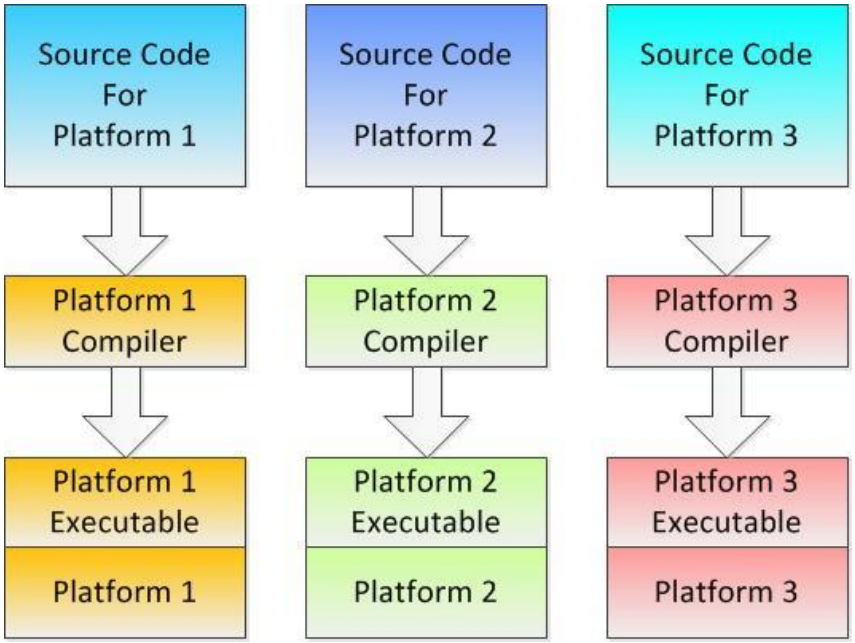
Some High-Level Languages Lineage

- Smalltalk (mid-70s)
  - First O-O language
- C (early 1970s)
  - The development language of UNIX
- C++ (early 1980s)
  - Often thought of as “C with objects”
  - Similar to C, added objects but also complexity
- Java (1995)
  - Originated at Sun Microsystems (now Oracle, as of 2010)
  - Originally named “Oak”, but that name was already in use
  - The “Java” name arose after a visit to a coffee shop

High-Level Language Types

- **Compiled**
  - A **compiler** pre-converts source code (instructions) into machine language
  - Then, that machine language is executed
  - Examples: C, C++, Objective-C
- **Interpreted**
  - An **interpreter** converts source code into machine language at run time
  - Then, that machine language is executed
  - Usually executes more slowly than a compiled program
  - Examples: PHP, Ruby, JavaScript, Python
- Java is a **hybrid** language
  - Java source code is compiled into **bytecode**
  - The **bytecode** is then executed by the **JVM** (Java Virtual Machine)
  - This helps solve the problem of **platform dependence**

Platform Dependence



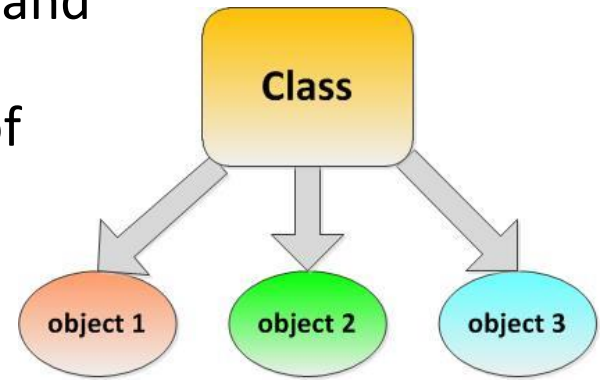
- Source code may be same for each platform, but usually it may have slight variations
- A different compiler is required for each platform
- Result is platform-dependent executables for each platform
- This is a nightmare from a support standpoint!

Object-Oriented Principles

- Object-oriented programming (OOP) is a different way of looking at software
  - Traditional languages: more focused on linear, procedural programming
  - Do this, then this, then this…
  - Large, monolithic programs are difficult to adapt or reuse
- In OOP, <u>everything</u> is an object
  - A software representation of some real-world “thing”
  - Think of them as reusable, software “Lego building blocks”
  - Objects themselves can be comprised of other objects

Object-Oriented Key Ideas

- **Class**
  - The means of encapsulating **data** and operations upon that data **(methods)** into one self-contained unit
  - Defines a template, or model, for creating and manipulating **objects**
  - A class is the “blueprint” for any number of new objects
- **Objects**
  - New “things” created using the class
  - An object is an **instance** of the class
  - Creating a new object is called **instantiation**
  - Each object has the same properties, but distinct values for them



Classes Versus Objects

- **Classes** and **objects** are the two dual, fundamental ideas in OOP
  - Classes are the general blueprints
  - Objects are the specific instances
- For example:
  - The same house **blueprint** can be used to build multiple houses in a new development, but each **house** has its unique address, paint scheme, and landscaping
  - The same **elementary school plan** can be used to start-up many identical schools, but each **school** has its own principal, teachers, student roster, and mascot
  - The same **cookie cutter** can stamp out dozens of cookies, but each **cookie** has its own flavor, frosting, and decorations

Characteristics of a Class

What is "is"

- Fields, or instance variables
- Attributes
- Descriptions
- "Nouns"

What it "does"

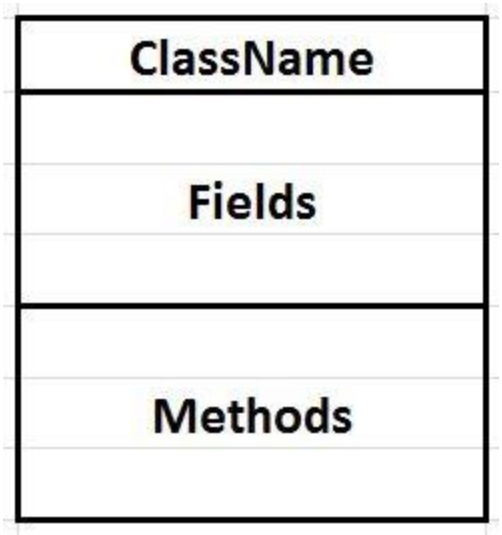
- Methods
- Behaviors
- Actions
- "Verbs"

- What IT can do
- What can be done TO IT

Examples of a Class

Class	Sample Attributes	Sample Behaviors
Person	first name, last name, birthdate, height, weight, …	eat, sleep, walk, breathe, speak, …
Dog	name, age, breed, weight, …	eat, sleep, run, bark, wag tail, fetch, give bath, …
Car	make, model, license plate, VIN, color, odometer, …	start, stop, accelerate, brake, wash, …
Course	course ID, department, subject, units, schedule, instructor, students, …	give exam, grade assignments, add student, evaluate course, …
Computer	OS name, OS level, type, manufacturer, memory, screen size, CPU, …	start up, shut down, change settings, upgrade OS, install app, connect to network, …

Class Representation



- Classes are frequently depicted using UML class diagrams
- UML = Unified Modeling Language
  - Formal treatment of UML is beyond the scope of this course
- This lets us consider/design the needs of a class, without getting lost in its details quite yet
- This is sometimes referred to as “Object Modeling”

Software Modularity

- Everything you'd ever need to know about an object is contained within that object ("**encapsulation**")
- Example: a box of IKEA furniture
  - Parts, tools
  - Instructions for assembly
- Example: a plumber's or electrician's service truck
  - Spare parts, tools
  - Expertise with which to use the parts and tools

Object-Oriented Advantage: Reuse

- Well-written classes can be reused in new applications
- Development time is shortened, because programmers don't need to rewrite that code
- Programs are more robust, because the class code is already well-tested
- Existing classes can be extended (“tweaked”) to accommodate the specific needs of a new situation
- Existing software can be more readily extended or adapted for new features or market opportunities

Java History

- Originated in 1995 by Sun Microsystems
  - Arose with the dawn of the Internet
  - HTML and web pages originally very static
  - There was need to add dynamic content, such as interactivity and animations, to web pages
- Platform independence
  - Internet content had to be provided in a platform-neutral way
  - Freely downloadable by developers
  - Original mantra: “Write Once, Run Anywhere”
- Now administered by Oracle
  - Acquired Sun Microsystems (and Java) in 2010

Java Now

- Remains a common, popular, general-purpose object-oriented programming language
  - 2nd in TIOBE Programming Community Index for January 2015 (C is #1)
  - Java Runtime Environment (JRE) is found on over 850 million PCs (source: Oracle)
  - As of 2015, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million users (source: Wikipedia)
- Familiar applications:
  - The Android mobile OS: all its applications are written in Java
  - Some familiar websites using Java: LinkedIn, Ebay, Paypal (source: W3Techs)
  - My son's Minecraft game
  - Most computers these days are "Java-enabled"

Other Devices Which Use Java

- A splash screen you may have seen during your Java JDK installation:

Usage vs. Understanding

- What happens when you…
  - point, click, drag, swipe, search, download, etc.
- Do you care what happens “under the hood”?
  - Probably not, as long as the desired end result happens quickly, accurately, and reliably
- But someone had to know
  - That someone was a software engineer or programmer
  - That someone has good marketable skills
  - Those skills center upon programming, such as we will be learning in this class

Java Opportunities

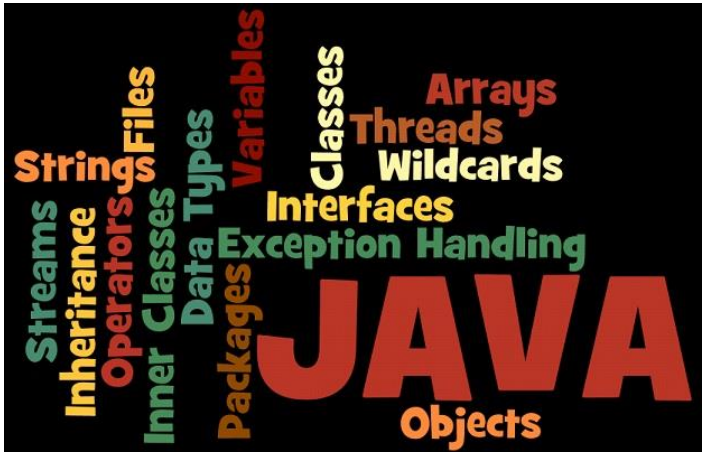
- Some results from quick, crude keyword searches on some online job posting sites:

Job board	Date	Keywords	Location	# Postings
SimplyHired.com	2/5/15	Java	Rocklin	606
SimplyHired.com	2/5/15	Java	Sacramento	655
SimplyHired.com	2/5/15	Java	San Jose	11,414
Careerbuilder.com	2/5/15	Java	Rocklin	36 (last 30 days)
Careerbuilder.com	2/5/15	Java	Sacramento	39 (last 30 days)
Careerbuilder.com	2/5/15	Java	San Jose	368 (last 30 days)
Dice.com	2/5/15	Java	Rocklin	55 (last 30 days)
Dice.com	2/5/15	Java	Sacramento	65 (last 30 days)
Dice.com	2/5/15	Java	San Jose	1171 (last 30 days)
Indeed.com	2/5/15	Java	Rocklin	248
Indeed.com	2/5/15	Java	Sacramento	270
Indeed.com	2/5/15	Java	San Jose	6544

- There are opportunities out there for what you are learning

Java Key Features

- Syntax is based on C/C++
- Object-oriented
- Inherently supports Internet applications
- Provides an extensive library of classes
  - Classes are pre-built, pre-provided capabilities
  - GUI building, database connectivity, networking, …
- Extremely well-documented online
  - The Java API
- Portable among platforms



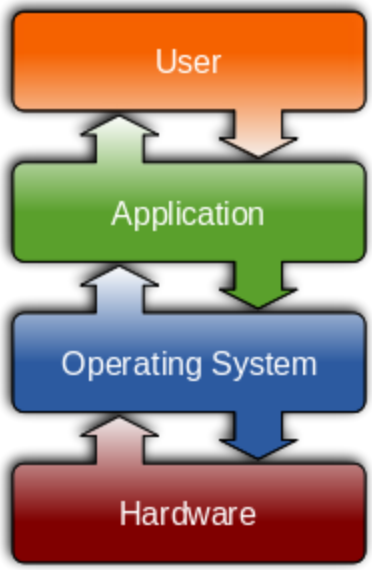
Java Program Types

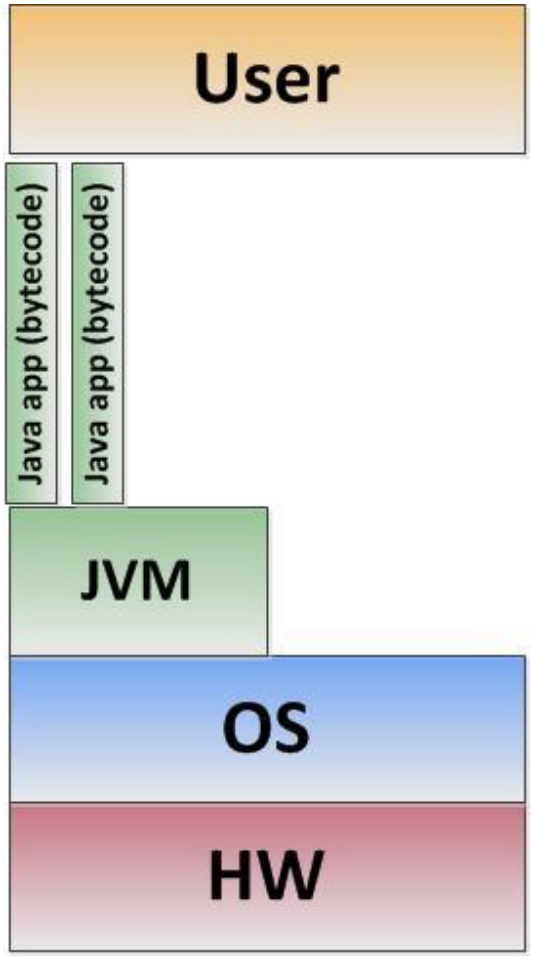
- **Applets** (client-side)
  - Small programs designed to add interactivity to websites
  - Downloaded with a web page and launched by an Internet browser
- **Servlets** (server-side)
  - Run by a web server, on the server
  - Typically generate web content
- **Applications** (standalone)
  - Programs that run standalone on a client machine
  - **The focus of THIS course**

Running Java

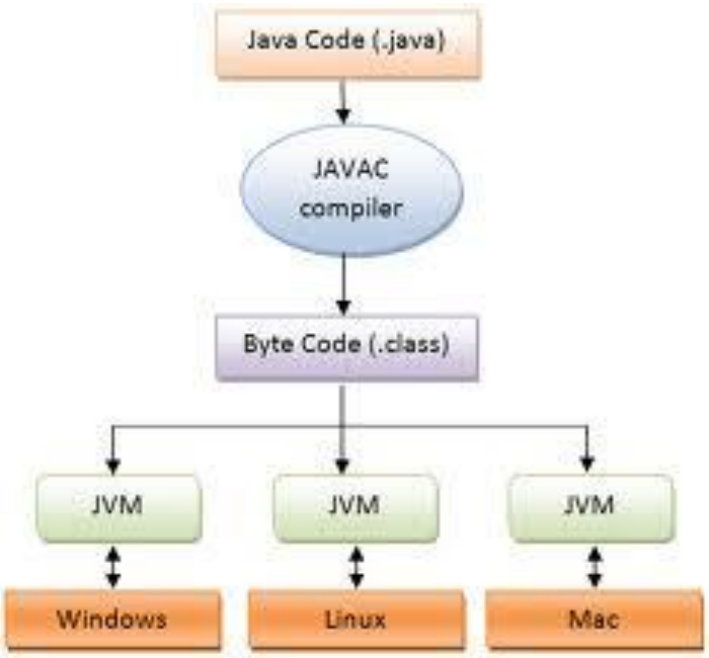
- Combination of a **compiler** and an **interpreter**
  - The Java compiler (javac) converts source code into **byte codes** (an instruction set for a virtual, machine-independent processor)
  - At run time, the **Java Virtual Machine (JVM)** interprets the byte codes and converts them into the machine language for the platform on which the program is running.
- So: Java applications run on top of another application (the JVM), which itself runs on top of the OS
  - See next slide…

Java As A “Layered” Application





Java Compile/Execute Process



- Java source code is compiled into **bytecode**
- The resulting bytecode is **interpreted** by the **JVM**
  - JVM = **Java Virtual Machine**
  - JVM is an application for each target platform/OS
- The same bytecode can be run on any platform's JVM
  - The Java mantra is "Write Once, Run Anywhere"
  - There is still platform dependence, but now it's a one-time cost

Java “Under The Hood”

- When we use the jGRASP IDE, we are simply using it as a convenient front-end to some underlying OS operations:

```
----jGRASP exec: javac -g HelloWorldRL.java


----jGRASP: operation complete.

 ----jGRASP exec: java HelloWorldRL

Hello Rob Lapkass, good to have you in class

 ----jGRASP: operation complete.
```

- We could achieve the same thing using the OS command line (assuming our search path is set properly):

For Next Time

- **Lecture Prep**
  - Text readings and lecture notes
- **Assignments**
  - See slide 2 for new/current/past due assignments