Java Introduction

First, Course Info

Nick Parlante
nick.parlante@cs.stanford.edu
(650) 725-4727
Feel free to call or stop by -- often in the office M-F
Detailed office hours on the course page
Course: Java, OOP design, Java Libraries
Prerequisite: Experience in C -- we won't cover basic programming
Course Page http://www.stanford.edu/class/cs193j
Questions: cs193j@cs.stanford.edu
Readings
No required text, but you may want a book for reference
Lecture outlines, source code -- available in PDF off course page
Platform -- Java
Mostly platform independent
Codewarrior Java license for Mac and PC
Must test your work on 1.3 JVM on leland
Grading
4-5 Homeworks (60%) + 1 final exam (40%)
Must pass both the homeworks and the exam to pass the course
CR/NC Teams
People taking the class CR/NC may work in teams of 2 on each assignment if they wish
Honor Code
See handout #1

Java Doc Links

A great deal of Java documentation is available online -- here's some links to get you started
http://java.sun.com/docs/
lots of docs about Java.
http://java.sun.com/docs/books/tutorial/
Java tutorial -- includes separate tracks for many topics.
http://java.sun.com/j2se/1.3/docs/api/
The "API" section is a reference for the many built-in library classes (String, ArrayList, ...)
http://www.bruceeckel.com/
Bruce Eckel's Book, Thinking in Java, in a free online form
http://www.afu.com/javafaq.html
The comp.lang.java FAQ -- maintained by Peter van der Linden
http://www.javaworld.com/
lots of Java articles

Java -- Buzzword Enabled

From the Sun Java whitepaper: "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multi-threaded, and dynamic language."

Simple

Simpler than C++ -- no operator overloading
Mimics C/C++ syntax, operators, etc. where possible
To the programmer, Java's garbage collector (GC) memory model is much simpler than C/C++

Object-Oriented

Java is fundamentally based on the OOP notions of classes and objects
Java uses a formal OOP type system that must be obeyed at compile-time and run-time. This is helpful for larger projects, where the structure helps keep the various parts consistent. Contrast to Perl, which as more of a quick-n-dirty feel.

Distributed / Network Oriented

Java is network friendly -- both in its portable, threaded nature, and because its libraries support common network operations

Interpreted

Java bytecode is interpreted by the Java Virtual Machine on each platform

Robust / Secure

Java is very robust -- both vs. unintentional memory errors and vs. malicious code such as viruses. Java makes a tradeoff of robustness vs. performance.
1. The JVM uses a verifier on each class at runtime to verify that it has the correct structure
2. The JVM checks certain runtime operations, such as pointer and array access, to make sure they are touching only the memory they should
3. The Security Manager can check which operations a particular piece of code is allowed to do at runtime

Architecture Neutral / Portable

Java is designed to "Write Once Run Anywhere", and for the most part this works. Not even a recompile is required -- a Java executable should work, without change, on any Java enabled platform.

High-performance

Java performance has gotten a lot better. It can approach the speed of C -- say within a factor of 2. However memory use and startup time are both significantly worse than C, although those problems may be fixable.

Multi-Threaded

Java has a notion of concurrency wired right in to the language itself. This works out more cleanly than languages where concurrency is bolted on after the fact.

Dynamic

Class and type information is kept around at runtime. This enables runtime loading and inspection of code in a very flexible way.

Java Compiler Structure

Compile classes in .java files -- produce bytecode in .class files

Bytecode

A compiled class stored in a .class files or .jar file
Represent a computation in a portable way -- as PDF is to an image

Java Virtual Machine

Loads and runs the bytecode for a program + the library classes
The JVM runs the code with the various robustness / safety checks in place -- robustness vs. performance tradeoff

JITs and Hotspot

Just In Time compiler -- the JVM may compile the bytecode to native code at runtime (with the robustness checks still in). (This is one reason why java programs have slow startup times.)
The "hotspot" project tries to do a sophisticated job of which parts of the program to compile. In some cases, hotspot can do a better job of optimization than a C++ compiler, since hotpsot is playing with the code at runtime and so has more information.

Java Lang + Its Libraries

The core java language is not that big
However, it is packaged with an enormous number of "library" or "off the shelf" classes that solve common problems for you
    e.g. String, ArrayList, HashMap, StringTokenizer, HTTPConnection, Date, ...
Java programmers are more productive in part because they have access to a large set of standard, well documented library classes. Code re-use at last!

Java: Programmer Efficiency

Faster Development
    Building an application in Java takes about 30% less time than in C or C++
    Faster time to market
Memory errors
    Memory errors largely disappear because of the safe pointers and garbage collector. I suspect the lack of memory errors accounts for much of the increased programmer productivity.
Libraries
    Code re-use at last -- String, ArrayList, ... (C++ can also do this to an extent)

Java Is For Real

Java has a lot of hype, but much of it is deserved.
Java is very well matched for many modern problem
    Using more memory and CPU time but less programmer time is an increasingly appealing tradeoff.
    Robustness and portability can be very useful features
I suspect we will be using some derivative of the Java language for a long time