CS193J: Programming in JAVA

January 6, 1999

Lecture #1

Overview

- CS193J, the course
  - Format, audience, materials, policies
- Java, the technology
  - A language
  - A set of libraries
  - A virtual machine
- Java, the phenomenon
  - Its quick rise to stardom
  - Present-day realities
  - Outlook for the future

Positioning

- The 193 concept
  - A bit outside the core curriculum
  - Practical, focused on skill acquisition
  - Majors and non-majors
- CS193J is "Java as a second language"
- Prerequisites
  - Solid C programmer, experienced in design, data structures, algorithms, debugging
  - No assumptions about objects or C++ (therefore overlaps with CS107, 108, 193D, etc.)

Content

- Language (3-4 weeks)
  - Syntax, objects, inheritance, interfaces, exceptions, inner classes, concurrency
- Core packages (5-6 weeks)
  - lang, util, io, network, awt, applet
- Implementation (1 week)
  - VM, security, efficiency
- CS193K for those more experienced
  - "Advanced Java" to be offered in Spring
  - Swing, beans, RMI, serialization, ...

Format

- Lectures Mon & Wed 12:50-2:05 pm
- Optional section Fri 1:15pm, Gates B01, ch. E2
- Web site has info, handouts, schedules, etc.
  - http://cse.stanford.edu/class/cs193j
  - Webclass used for on-line grade database, must register to reserve your spot -- see link on web site
- E-mail questions go to cs193j@cs.stanford.edu
- Book: Java 1.1, The Complete Reference
  - Delayed at bookstore, link from FAQ page to order on-line
  - Can use other Java books to supplement/replace if you like

Coursework

- Using JDK version 1.1
  - Metrowerks CodeWarrior Pro 4 or Solaris JDK 1.1.5
  - Others most likely will work as well
- Programming assignments
  - About one every 1-2 weeks
  - Somewhat substantial, but not intended to be grueling
  - Done individually, except CR/NC can work in pairs
  - 5% per 24-hour late, 4 days of self-granted extensions
- Final exam
  - 3-hour in-class exam in official slot during finals week
- Honor Code taken very seriously

Looking backward: C, 1970

- Original design for operating systems programming (relatively low-level)
- Strengths
  - Efficiency
  - Lack of restrictiveness
  - Portability
- Weaknesses
  - Safety
  - Expressiveness
  - Only a small amount of built-in functionality

Ten years later, C++

- Complete superset of C
- Changes include:
  - Objects
  - Parametrized types
  - Unification of user-defined and built-in types
- Strengths
  - Improved data abstraction and encapsulation
  - More extensive built-ins (STL)
- Weaknesses
  - Considered by many to be over-complicated
  - Contains all of C's warts
- Design gives ability, doesn't mandate policy

Another ten years, Java

- Similar to C/C++ but not a superset
  - Conscious decision to not be backward compatible with C++ or even C
- Not originally designed as general-purpose programming language
- Strong emphasis on safety and portability
- Unique features make it particularly suitable for dynamic content delivery over the web

Java design

- Start with C syntax and semantics
- Add just a few features from C++
  - Objects, exceptions
- Leave out parts unneeded, unsafe, complex
  - Gosling: "Java omits many rarely used, poorly understood, confusing features of C++ that in our experience bring more grief than benefit."
- Add a few facilities not present in C or C++
  - Garbage collection, concurrency, runtime error checking
- Strengthen portability and security

Java: buzzword enabled

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

Simple

- Streamline C language
  - No typedefs, unions, enums, goto, comma operator
  - No preprocessor
  - No header files
- Simplify problematic areas
  - No pointers, function pointers
  - Add garbage collection
- Provide better utility support
  - Built-in array and string objects, as well as hashtables, trees, dates, etc.

Object-oriented

- Combine data and behavior into one unit, the ***object***
- Perceived advantages of objects
  - Provides strong data abstraction and encapsulation
  - Gives framework for design
  - Allows independent development and testing
  - Facilitates system extension and maintenance
  - More opportunity for code re-use
- The object-oriented paradigm
  - Decompose program into objects
  - Object relationships: cooperate, contain, and inherit
  - Programs are collections of interacting, cooperating objects

Platform-independent

- Portable
  - Java programs behave predictably, no matter what platform
  - Mandated type sizes, byte ordering, order of sub-expression evaluation, behavior on exceptional conditions
  - Compliance with library specification verified by large test suite
- Architecture-neutral
  - Not just source-compatible, but binary-compatible
  - Java compiled to byte-code, not object code
  - Virtual machine interprets byte codes during execution
- "Write once, run anywhere"

Robust and secure

- Fewer language loopholes
  - No pointers, typecasts limited, no variable argument functions
- Compilers are strict
  - Require initialization of variables, enforce type-consistency, prototypes required
- Runtime error checking
  - Array bounds checking, null pointer access
- Security Manager
  - System to control permissions for high-level actions
- Runtime verifier checks untrusted byte code
  - Avoids havoc from hand-constructed byte-code

The Java libraries

- A language and its libraries should be considered as one unit
- At one end of the spectrum, a language like C
  - A little I/O, string handling, qsort, not much else
- At the other, a language such as Common Lisp
  - Large collection of built-in routines
- Java pushes that extreme even farther
  - Extensive range of included functionality
  - Objects lend themselves toward packaging for re-use

Package highlights

- Collections of classes grouped in "packages"
  - Java equivalent of a C library
- java.lang
  - String, Math, Exception, Thread, Runtime, etc.
- java.util
  - Vector, Stack, Hashtable, Date, Tokenizer
- java.io
  - All varieties of input/output processing
- java.net
  - Networking, client/server sockets, URLs
- java.awt and java.swing
  - Windows, buttons, drawing, images, events

And more...

- java.security
  - Encryption, digital signature, message digest
- java.text
  - Formatting and parsing
- java.sql
  - Database connectivity
- java.rmi
  - Remote method invocation, distributed objects
- The list goes on...
  - And it is continuing to expand, Java is still young

Libraries = leverage

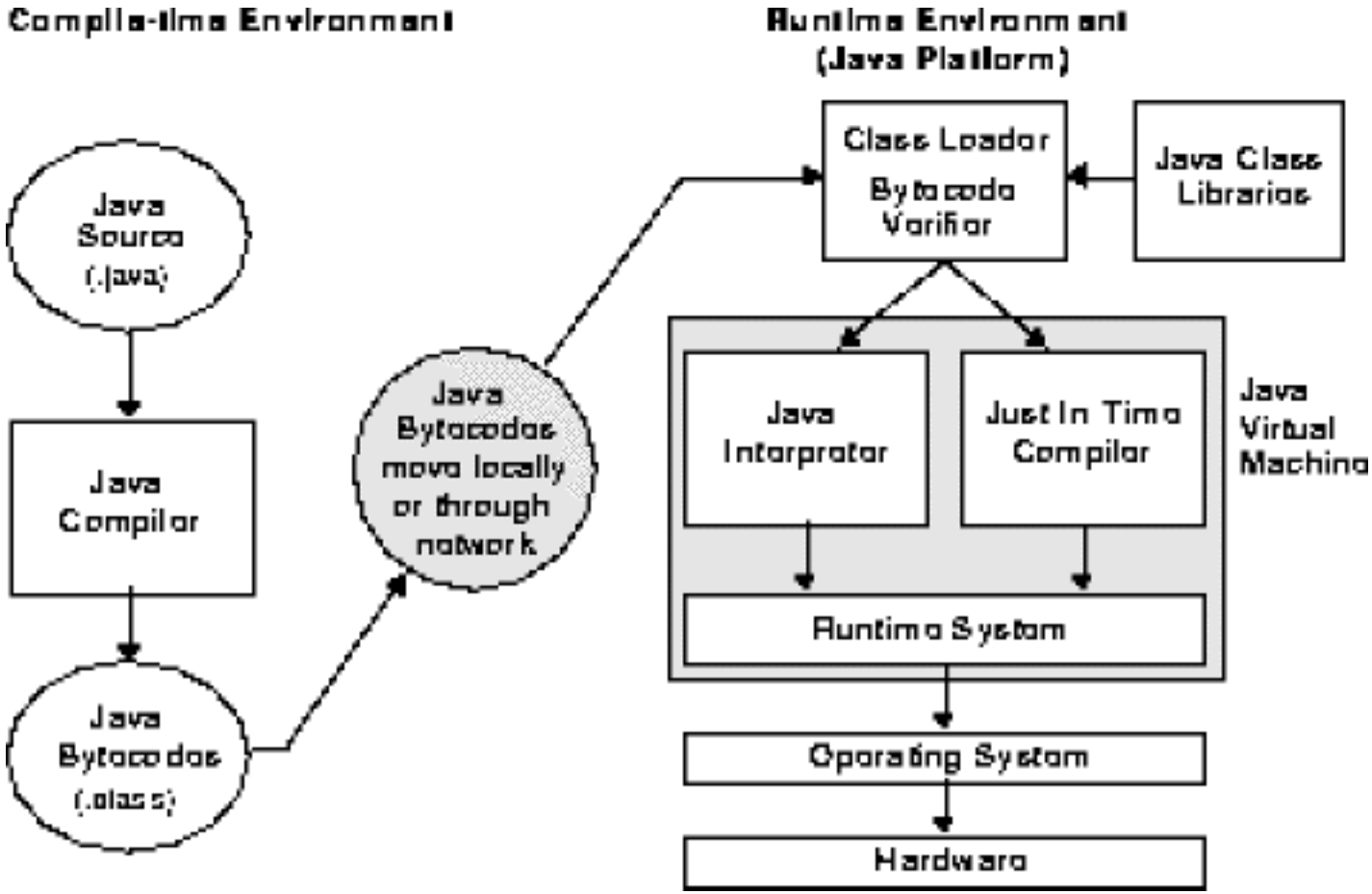
- Java has far expanded traditional scope of a language's libraries
- Much less effort required to accomplish common tasks
- Platform-specific details are handled
- All programs improve when common core updated

The virtual machine (VM)

- VM is the execution environment for Java code
  - Java source compiles to byte code
  - Byte codes are instructions for virtual machine
  - VM is a platform-specific tool to interpret the byte codes and translate to commands for certain processor and OS
- Porting a VM (and libraries) is the hard part
  - But porting done once by the vendor instead of again and again by each application author

Compile-interpret-execute cycle

(Diagram from Javasoft docs)



VM characteristics

- Adds a level of robustness
  - Execution occurs in "artificial" context, not on raw hardware
  - Byte codes are verified before execution to ensure correctness
- Adds inefficiency
  - Extra work to interpret byte codes as well as safety checks
    - Interpreted Java can be 5-10x slower than comparable compiled C
  - "Just in time" (JIT) code generators can compile byte codes to machine code on the fly
    - Have brought us within factor of 2-3 of C
  - Sun developing next generation "HotSpot" VM with aggressive code optimization and compilation
    - Amazing results predicted, have to wait and see

Java's quick rise to stardom

- C and C++ each took over a decade to gain favor
- Java still in its infancy and yet so popular -- why?
- Possible theories
  - Things happen faster in "internet time"
  - Tremendous market pressure for dynamic web content
  - Cross-platform development is very desirable
  - Java plays well to mentality of programmers
    - Looks like C (familiar & comfortable) but acts like Lisp (safe and expressive)
  - Even simpler: Java is just a good tool at the right time

Promise versus reality

- So in theory, Java is amazing
- In reality, there have been some issues
  - Efficiency concerns
  - Language and library immaturity and instability
  - Platform inconsistencies
  - Unreliable tools
  - Politics (in particular with regard to Microsoft)

Current Java activity

- On-going development
  - Lots of small applets, games, tools available today on web
  - Significant Java development on internet sites
  - Some in-house applications being done in Java
  - Not many full-featured commercial apps attempting Java yet
- Education
  - Much use of Java for educational tools and demos
  - Some schools switching to Java for intro course

Looking to the future

- VM performance will improve
  - So will the hardware, which always helps
- Language and libraries should stabilize
  - More companies than Sun need to be involved though
- Platform inconsistencies should lessen
- Political relationships need work
  - How to bring Microsoft fully on board?

Assignment #0

- Your first Java program
  - Take simple program on p.25 in text and figure out what it takes to edit, compile, and execute it in your development environment of choice
- Friday's section is demo of Mac & UNIX tools
- Reading for next Monday
  - Skim ch. 3, 4, & 5 on types, operators, and control structures (skip bit operators)
    - Most of this is same as C, should be quick to go through
- Register with Webclass!