CS193J: Programming in Java
Summer Quarter 2003

Lecture 1
Course Overview
Introduction to OOP/Java

Manu Kumar

sneaker@stanford.edu

Agenda

- Introductions
  - Instructor
  - TA
- Course Overview
  - Administrivia and Logistics
- Student Introductions
- Introduction to Java
- OOP/Java

Handouts

- 4 Handouts for today!
  - #1: CS193j – Programming in Java
  - #2: Java 1
  - #3: OOP
  - #4:Java 2

Staff Introductions

- Intructor: Manu Kumar
  - sneaker@stanford.edu
  - 650-723-1923
  - Office: Gates 266
  - Office hours: Monday 4:15 PM – 6:15 PM
- Teaching Assistant: Shankar Ponnekanti
  - pshankar@cs.stanford.edu
  - 650-725-3053
  - Office: Gates 252
  - Office hours
    - Tuesday 2:15 PM – 4:15 PM in Gates 252
    - Wednesday 8:00 PM – 10:00 PM in Sweet Hall

CS193J Course Overview

- Java as a second language course
  - Teaches programming in Java for people who already know how to program in C or similar language
- Pre-requisites
  - Basic programming background (C/C++/Pascal)
  - Problem solving techniques
  - Debugging skills
  - Basic HTML
- Flavor
  - A hands-on course for practitioners
    - Requires considerable hands-on development time.

Credits and Acknowledgements

- Summer Quarter 2003
  - Based on previous versions of the course
  - Will closely mirror Winter 2003 course
  - Re-use materials from previous instructors
    - Evolve material based on prior feedback
  - Credits
    - Nick Parlante (Winter 2003)
    - Julie Zelenski (assignments)
    - Prior instructors

Class Structure

- Lectures
  - 4:15 PM – 6:05 PM
  - Two 50 minute sessions
    - 4:15 PM – 5:05 PM
    - 5:15 PM – 6:05 PM
- Lectures will provide background for progamming assignments
- Office hours
  - Intended to be hands on office hours
    - Theoretical principles
    - Programming issues

Text Book

- Text Book
  - No official text
    - Java is an open language which is very popular on the Web and so everything you need is available on the Web!
  - If you *must* have a text book
    - Readers
      - Just Java by vad der Linder
      - Core Java 2 series by Horstmann
    - Reference
      - Java in a Nutshell series

Handouts

- Re-using handouts by Nick Parlante
  - Printed copies will be handed out in class
  - Posted on course website about 30-60 minutes prior to lecture
- Slide Sets
  - No printed copies
  - Will mirror handouts
  - Will be posted on course website

Assignments

- 4 individual programming assignments

Homework	Assigned On	Due Date	Percentage
#1	Thursday, June 26th	Wednesday, July 9th by 11:59 PM	20%
#2	Thursday, July 10th	Wednesday, July 23rd by 11:59 PM	30%
#3	Thursday, July 24th	Wednesday, August 6th by 11:59 PM	40%
#4	Thursday, August 7th	Wednesday, August 13th by 11:59 PM	10%

- *Note: This means there is no final exam!* ☺

P/NC / SCPD Students

- P/NC Students
  - Same criteria as students taking the class for a regular grade except
    - May not have to do *all* parts of the assignment
    - Will be indicated on the assignment handouts
  - Please indicate that you are a P/NC student
- SCPD Students
  - All course information will be on the course website
  - We will actively respond to emails to the course alias (coming up next…)
  - Please indicate that you are a SITN student

Late Submissions

- Deadlines are hard deadlines
  - If an assignment is due at 11:59 PM, it will be considered late if submitted at 12:01 AM.
- BUT…. Three free “late-days”
  - Each late day is exactly 24 hours.
  - Use your late days wisely.
  - Relative weight of assignments indicates their level of difficulty/work required.
- ½ letter grade penalty for each day late

Honor Code

- Stanford uses the Honor code system
  - http://www.stanford.edu/dept/vpsa/judicialaffairs/guiding/honorcode.htm
- The fine print…
  - The Honor Code is an undertaking of the students, individually and collectively:
    - that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
    - that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.
  - The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.
  - While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

Collaboration Policy

- Okay to discuss ideas and problem approaches
- All work must be your own creation
  - Not okay to share code or look at other people’s code
- Always give credit where credit is due
- See handout for details.

Disability Resource Center

- Stanford Disability Resource Center
  - http://www.stanford.edu/group/DRC/

- Please note: no special accommodations can be made without having the official forms from the DRC first

Course Outline

- Introduction to Java
- Arrays, Strings, static
- OOP – encapsulation, inheritance
- OOP in Java – inheritance, abstract superclasses, Interfaces, inner classes, packages.
- Building GUIs with Swing. Components, drawing, layouts, graphics
- Listeners, buttons, mouse-tracking
- Threads and concurrency. Threads, runnables, critical sections, synchronization
- Exceptions, I/O, Streams
- Networking
- MVC structure
- Advanced topics: XML, JDBC performance

Guest Speakers

- Sun’s J2EE Team
  - George Grigoryev
    - J2EE Senior Product Manager, Java Software Technologies, Sun Microsystems, Inc.
  - August 7th, 4:15 – 6:05 PM

Student Introductions

- Please say your:
  - Name
  - Where you are from
  - Stanford / non-Stanford affiliation
    - eg: Stanford department, undergrad HS Summer visitor, visiting from another university, company
  - What is your objective in taking this class
- SCPD Students
  - Please email your response to sneaker@stanford.edu with the subject: cs193j introduction

Request for constant feedback!

- Your feedback is needed to make sure that the course is on track
  - Please feel free to email or talk with us about the course and let us know how you think it is going often!

10-minute Break!

- Please be back promptly!

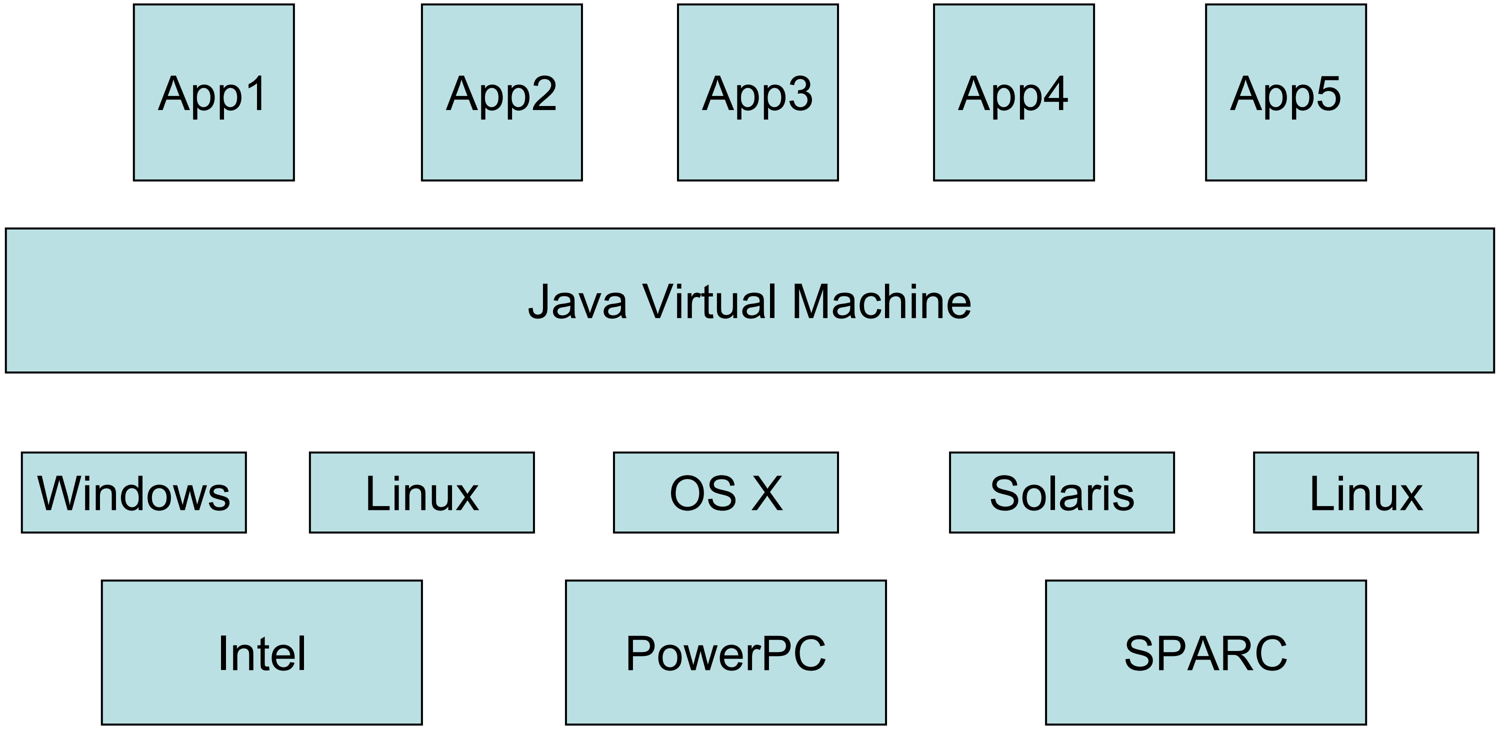
Introduction to Java (Handout #2)

- The Java buzzword-bingo!
  - Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high performance, multi-threaded, and dynamic language.
- Right Language, Right Time
  - Kept all the good features of C/C++
  - Dumped a lot of the ugliness
  - Timing
    - Internet boom
    - Moore's Law.

The Java Virtual Machine

- The Java Language runs on a “Java Virtual Machine”
  - Java Virtual machine abstracts away the details of the underlying platform and provides a uniform environment for executing Java “byte-code”
- The Java compiler (javac) compiles Java code into byte-code
  - Bytecode is an intermediate form which can run on the JVM
  - JVM does the translation from byte-code to machine-code in a platform dependent way.

The Java Platform



Language + Libraries

- Core language
  - Ints, array, objects, loops and conditionals
  - Moderately sized language
    - Can run on small devices
- Libraries
  - This is where the power of Java really emerges
    - String, ArrayList, HashMap, String Tokenizer
  - Networking, Graphics, XML, Database connectivity, Web Services….
  - Re-use at it's best (so far).

Simple

- Similar to C/C++ in syntax
- But eliminates several complexities of
  - No operator overloading
  - No direct pointer manipulation or pointer arithmetic
  - No multiple inheritance
  - No malloc() and free() – handles memory automatically
  - Garbage Collector
- Lots more things which make Java more attractive.

Object-Oriented

- Fundamentally based on OOP
  - Classes and Objects
  - Uses a formal OOP type system
  - Lends an inherent structure/organization for how we write Java programs
    - Unlike spaghetti code in languages like Perl
  - Efficient re-use of packages such that the programmer only cares about the interface and not the implementation
- OOP will be covered in a little more detail later.

Distributed / Network Oriented

- Java grew up in the days of the Internet
  - Inherently network friendly
  - Original release of Java came with Networking libraries
  - Newer releases contain even more for handling distributed applications
    - RMI, Transactions

Robust / Secure / Safe

- Designed with the intention of being secure
  - No pointer arithmetic or memory management!
  - The JVM "verifier"
    - Checks integrity of byte-codes
  - Dynamic runtime checking for pointer and array access
    - No buffer overflow bugs!
  - SecurityManager to check which operations a piece of code is allowed to do
  - "Sandbox" operation for applets and other untrusted code
    - Limited set of operations or resources made available
    - Contrast to ActiveX

Portable

- "Write-Once Run-Anywhere"
- The Java Virtual Machine becomes the common denominator
  - Bytecodes are common across all platforms
  - JVM hides the complexity of working on a particular platform
    - Difficult to implement a JVM
    - But simpler for the application developer
- Java does this well

High-Performance

- Honestly – thanks to Moore's Law
- Java performance IS slower than C
    - Tradeoff between development time vs. run time
    - Additional checks in Java which make is secure and robust and network aware etc, all have a small cost.
- BUT
    - JIT compilation and HotSpot
        - Dynamic compilation of bytecode to native code at runtime to improve performance
    - HotSpot optimizes code on the fly based on dynamic execution patterns
        - Can sometimes be even faster than compiled C code!

Multi-Threaded

- Native support for threading
  - We will cover this in a lot of detail
- Basic concept
  - The ability to have multiple flows of control/programs which appear to run at the same time
    - Processes - application level
    - Threads – within the application
  - JVM uses native threads on operating system but provides a consistent abstraction for the developer.

Dynamic

- Java is “self-aware”
  - Java code can look at itself and tell what interfaces it exports (Introspection)
  - Can dynamically load new classes/code at runtime

Java Programmer Efficiency

- Faster Development
  - More programmer friendly
  - Less error prone
- OOP
  - Easier to manage large development projects
- Robust memory system
  - No pointer arithmetic and manual memory management. Garbage collector!
- Libraries
  - Re-use of code

Microsoft vs. Java

- Java is platform independent
  - Was considered a threat to Microsoft’s dominance
  - Sun vs. Microsoft Law Suit
- Microsoft’s latest response to Java
  - C#
    - Very similar to Java in structure and style
    - Some improvements over Java (which have now emerged in Java 1.5)
    - Some questionable features

Java is for Real

- The initial hype has died down and things are more realistic today
  - Similar to the economy.
- Java is maturing into a mainstream language for development
  - Here to stay

Canonical Example

- HelloWorld Application in Java

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

- Compile: javac HelloWorldApp.java
- Run: java HelloWorldApp

OOP (Handout #3)

- Object Oriented Programming
  - Map your problem in the real world
  - Define “things” (objects) which can either do something or have something done to them
  - Create a “type” (class) for these objects so that you don’t have to redo all the work in defining an objects properties and behavior

- Some examples
  - Classroom
  - Car
  - Person

Procedural vs. OOP

- Nouns refer to Data
- Verbs refer to Operations

- Procedural Languages
  - C/Pascal etc.
  - Verb-oriented
  - No formal noun-verb structure (not enforced by language or compiler)
- OOP languages
  - Operations are performed by/on “Actors” (objects) which have names and store data (nouns)

Objects

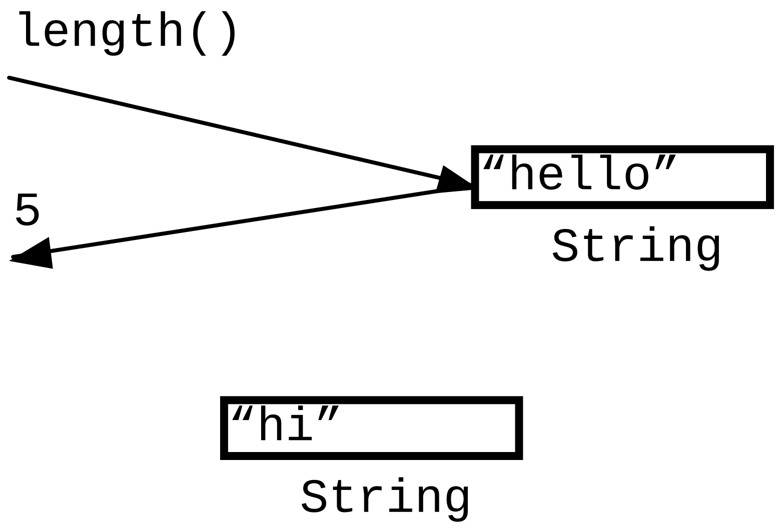
- Storage
  - Properties
  - Runtime state
- Behavior
  - Set of operations that can be performed by an object, usually on itself
- Class
  - Every Object belongs to a class
- Objects are anthropomorphic
  - Think of them as being alive (actors) that can do something

Class

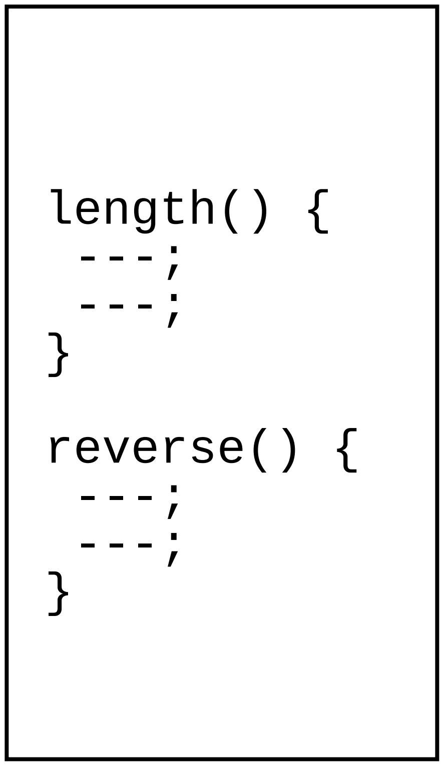
- Exists once
  - The Class is the template for the object
  - Defines the storage and behavior of the objects
- Every object must belong to a Class
  - The object is an instance of a class
- Example
  - Person is a class
  - Bart Simpson is an instance of class Person

String Example

Sending the length() message to a String object...



String class



```
length() {
 ---;
 ---;
}

reverse() {
 ---;
 ---;
}
```

Message / Receiver

- Objects are manipulated by sending them messages
  - The object itself is therefore a receiver of messages
  - The message is usually a “method invocation”
- Example:

```
    String name = “bart simpson”;
    name.reverse();
// name is receiver, reverse is the message
```

Methods / Message->Method Resolution

- Method
  - Executable code defined in class
  - It is the behavior/operation
  - Objects of a class can execute all the methods that class defines
- Suppose a message is sent to an object --- x.reverse();
  - 1. The receiver, x, is of some class -- suppose x is of the String class
  - 2. Look in that class of the receiver for a matching reverse() method (code)
  - 3. Execute that code "against" the receiver-- using its memory (instance variables)

OOP Design – Anthropomorphic/Modular

- Objects are responsible for their own state
- Objects can send messages to each other
- The object/message paradigm makes the program more modular
  - Each class deals with it's own implementation details
  - The other classes only need to know the interface it exposes

OOP Design Process

- Think about the objects that make up an application
- Think about the behaviors or capabilities those objects should have
- Endow the objects with those abilities as methods
- If a capability does not occur to you in the initial design, that's ok. Add it to the appropriate class when needed – it just needs to go in the right class
- Co-operation
  - Objects send each other messages to co-operate
- Tidy style
  - Experience shows that having each object operate on its own state is a pretty intuitive and modular way to organize things.

OOP Design Exercise

- You are asked to design the game of Chess. What are some of the classes you would create and what properties would they have?

OOP Design Exercise Trivial Solution

- The following is a *very high level and trivial solution*

- Board
  - Black squares and white squares
- Pieces
  - Position on the board
  - Dead or alive
  - Valid moves
  - Move to new location
- Players
  - 2 instances (black and white)

Summary!

- Today
  - Course Introduction
  - Student Introductions
  - Introduction to Java
  - OOP concepts
- To Dos
  - Write a HelloWorld program in Java, compile it and run it on Leland machines.
  - SITN students: email introductions