CS212
Java Practicum

Fall 2004
Lecture 1
Introduction

Announcements
- [http://courses.cs.cornell.edu/cs212/](http://courses.cs.cornell.edu/cs212/)
- partner list (need partners?) (we'll explain in lecture)
- CMS?
- Part 1 posted tomorrow

Overview
- the course
  - course stuff
  - course overview
  - project overview
  - teams and CMS
- computer model
  - computer architecture
  - machine model
  - JVM model
  - SaM

Staff
- Instructors:
  - Prof. Keshav Pingali <pingali@cs.cornell.edu>
  - Prof. David I. Schwartz <dis@cs.cornell.edu>
  - Prof. Jai Shanmugasundaram <jai@cs.cornell.edu>
- TAs:
  - Ivan Gyurdiev <ivg2@cornell.edu>
  - David Levitan <dbl24@cornell.edu>
  - Daria Sorokina <ds373@cornell.edu>
  - Warren Wong <ww58@cornell.edu>
- See Staff on CS212 website
Classes

- Lecture
  - here
- Lecture notes? see Notes & Examples on website

- Section
  - sporadic—will hold based on demand
  - sometimes once before each homework

Reading

- Usually just current CS211 and CS212 notes
- Optional:

Software

- JDK 1.4, but not 1.5!
- IDE is optional
- See current CS211’s website ("Obtain Java" link)

Why are you here?

- CS212: A project course that introduces students to the ways of software engineering using the Java programming language. The course requires the design and implementation of several large programs.
- So, become better designers!
Why are we here?

- We want and need you to do the following:
  - Improve your programming skills.
  - Implement principles of software engineering, which include top-down and bottom-up design, software reuse, abstraction, and testing.
  - Develop interpersonal and project management skills, which you need for later courses.
  - Learn about the field of computer science.
- Lecture schedule on website

When to Take CS212?

- Take CS212 now?
  - connection with CS211 very tight
  - concepts expanded upon in CS212
  - coordination of assignments and instructors
  - just-in-time learning
- Take CS212 later?
  - more experience under your belt
  - need to balance with CS312
  - less connection (possibly) with your CS211
- Take CS212 before CS211?
  - No!

Groups and CMS

- partners allowed:
  - why? practice for later team-projects
  - learn about code integration
  - teams of 1, 2, 3
- the gist:
  - you choose team
  - once start for part must continue
  - can redo groups for each part
  - more detail in Syllabus
- CMS:
  - all work submitted on CMS
  - form groups early!
  - follow CS211 format (we'll alert you to changes)
  - regrades by meeting

Grades and Coursework

- Things to do:
  - four "Parts"
  - group evaluations
- Breakdown:
  - Part 1: 10%
  - Part 2: 25%
  - Part 3: 35%
  - Part 4: 30%
The Project

- build a compiler:
  - translate C-like language (Bali++) to assembly-like code (SaM)
  - use techniques you are learning in CS211
- SaM code:
  - resembles JBC
  - runs in simulation of JVM (SaM)
  - where’s SaM? website (careful of versions!)
  - currently http://www.csuglab.cornell.edu/~dbl24/sam/

Bits and Bytes

- Why SaM? Start at beginning!
- computer:
  - “programming electronic device that can store, retrieve, and process data”
  - digital: stores limited number of digits
- binary digits:
  - stick to 0 and 1 (bits)
  - byte: 8 bits

Storage of Bits

- boolean operations: AND, OR, XOR, …
  - e.g., \( \text{AND}(1, 0) \rightarrow 0 \)
- gate:
  - device that produces the output of Boolean operation
  - computers implement gates as small electronic circuits in which bits are represented as voltage values
- circuits:
  - gates provide building-blocks to create computers
  - can store bits, which means we can store info!
- integrated circuit:
  - embed multiple gates on chip
  - need to discuss kinds of memory

Von Neumann Model

- We can stores bits, so where do we put programs and data? together!
- Von Neumann Model of computer architecture:
Computer Memory

- **MU**: "Memory Unit"
  - think RAM
  - composed of circuits that provide main memory
- **main memory**
  - cells that store bytes
  - each memory cell has an address that stores bytes
  - start with 0 and work up to storage capacity
  - programs and data converted to bits to store
- **CPU** has some memory: **registers**
  - similar to main memory
  - for small amounts of data
  - PC: program counter
  - IR: instruction register (current instruction)
  - SP: stack pointer
  - more

Assembly Language

- **instruction set**:
  - complete set of instructions for machine
  - has two parts:
    - opcode
    - operand
  - needs 2 bytes, or “binary strings”
- **assembly language**:
  - symbolic representation of machine language of specific processor
  - mnemonics:
    - write in human form
    - e.g., **PUSH IMM 3**

Fetch-and-Decode Cycle

- CU fetches next instruction from main memory at the address in the program counter (PC)
- CU places the instruction into the instruction register
- CU increments the PC to prepare for the next cycle
- CU decodes the instruction to see what to do
- CU activates the correct circuitry to carry out the instruction (such as getting the ALU to perform an operation)

Stack Machine Model

- **Main Memory Reminder**
  - stack of cells that can hold information as bits
  - instructions and data can be translated as bit patterns
- **Java Bytecode (JBC)**
  - Java code gets compiled into class files which contain sequences of bits
  - not usually readable characters ("binary files")
  - a byte-code interpreter runs each instruction of byte-code in a similar fashion as machine code
Java Bytecode Example

```java
public class repeat {
    public static void main(String[] args) {
        int x = 1;
        int y = 10;
        while (x < y) x = x+1;
    }
}
```

- Use `javap -c classfile` to see JBC (and next page)
- See also Programming for JVM book

Why Java?

- JVM is “average” of all computers
- JBC is “average” of all instruction sets to run on JVMs
- interpreter runs JBC for JVM to run on computer
- JVMs programmed for specific architectures, so Java can be compiled and run “everywhere”
- “write once, run everywhere”

The JVM

- JVM is only a specification (see optional books)
- programmers write for specific architecture
- certain features are prevalent, like a stack:
  - data structure for storing items which are to be accessed in last-in first-out order
  - adding something to the stack: items are PUSHED onto a stack
  - taking something out of the stack: items are POPPED off of a stack
- instruction set architectures typically use stacks to store arguments and return addresses

SaM

- Stack Machine
- Stack (methods), Heap (objects), registers
- approximate a JVM: learn about assembly programming with our own instruction set