

CS212

Java Practicum

Spring 2006
Lecture 1
Introduction

1

Announcements

- Three ways to get website:
 - Link from CS211
 - <http://courses.cs.cornell.edu/cs212/>
 - <http://www.cs.cornell.edu/courses/cs212/2006sp>
- CMS:
 - check if you're on (see website)
 - e-mail patwell@cs.cornell.edu with "Need to be on CMS for CS212"
- Part 1 of project
 - posted soon
 - no partners
 - Due TBA
 - See *Chapter 1* in notes

2

Overview

- **The course:**
 - course stuff
 - course overview
 - project overview
 - teams and CMS
- **Computer models:**
 - computer architecture
 - machine model
 - JVM model
 - SaM

3

Staff

- **Instructors:**
 - Prof. David I. Schwartz
 - also, "Director of GDIAC"!
- **Grad TA:**
 - Jon Kaldor
- **Ugrad TAs:**
 - David Levitan
 - Aaron Sidford
 - Etan Bukiet
 - Kevin Hunton Foster
 - Oleg Sagalchik
- **See Staff on CS212 website**

4

Classes

- **Lecture:**
 - here!
 - Lecture notes? see **Notes & Examples** on website
- **Section:**
 - very sporadic—will hold based on demand
 - sometimes once before each homework

5

Reading

- Usually just current CS211 and CS212 notes
 - The CS212 book?
- **Optional:**
 - *Programming for the Virtual Machine*, J. Engel, Reading, Mass: Addison-Wesley, 2003.
 - *The Java Virtual Machine Specification*, Second Edition, T. Lindholm & F. Yellin, Boston: Addison-Wesley, 1999

6

Software

- **Java 5!** (also known as **jdk1.5**)
- *IDE is optional*
- See current CS211's website:
 - <http://www.cs.cornell.edu/courses/cs211>
 - Click on **Help & Software** under **Java Resources**
 - Follow downloading directions for JDK
- **SaM:**
 - runs in Java 5
 - will be used for all CS212 assignments
 - might show up in CS211, too

7

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, you want to become better designers!

8

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.

9

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! (well,....)

10

Groups and CMS

- **Partners allowed (Parts 2+):**
 - 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 **Course Info**
- **CMS:**
 - all work submitted on CMS
 - form groups early!
 - usually follow CS211 format
 - CS211 website: see **Info** (under CMS)
 - we'll alert you to changes
 - regrades usually by meeting

11

Grades and Coursework

- **Things to do:**
 - 3 "Parts"
 - programming, documentation, presentation
- **Breakdown:**
 - Part 1: 5%
 - Part 2: 35% (2 subparts)
 - Part 3: 59% (3 subparts)
 - see **Schedule** and **Course Info** on website

12

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? see CS212 course website

13

Bits and Bytes

- **Computer:**
 - “programming electronic device that can store, retrieve, and process data”
 - *digital*: stores limited number of digits
- **Data:**
 - *bits*: use binary digits: 0 and 1
 - *byte*: 8 bits
 - *cells* or words: groups of bytes

14

Storage of Bits

- **Boolean operations:**
 - **AND, OR, XOR, ...**
 - e.g., **AND (1, 0) → 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...

15

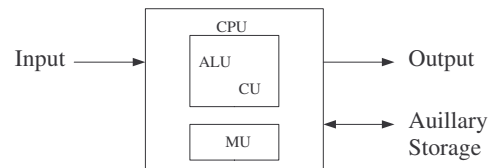
Computer Memory

- **Memory:**
 - large collection of cells
 - has location (*address*)
 - can access cells in any order
- **RAM** (Random Access Memory)
 - storage for programs you run
 - programs fill cells with data
 - what about instructions...?

16

Von Neumann Model

- We have ability to store bits, which can be....
 - data
 - programs
- **Von Neumann Model** of computer architecture:



17

Components of Model

- **MU:**
 - **memory unit**
 - hold data and programs
- **ALU:**
 - **Arithmetic/Logic Unit**
 - handle arithmetic and logic calculations
- **CU:**
 - **Control unit**
 - interprets instructions
 - controls ALU, Memory, I/O
- **I/O:**
 - **input/output**
 - some storage

18

CPU

- **CPU:**
 - central processing unit
 - CU, ALU, and registers
 - interact with memory
- **Registers:**
 - writable memory cells
 - hold small amounts of data
 - various types:
 - **PC:** program counter
 - **IR:** instruction register (current instruction)
 - **SP:** stack pointer
 - more...

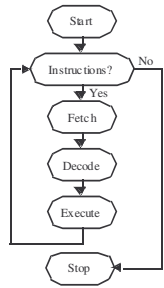
19

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., **PUSHIMM 3**

20

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)

21

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
- Use `javap -c classfile` to see JBC

22

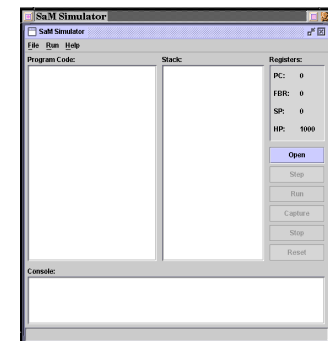
Why Java?

- 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!”**

23

JVM and SaM

- **JVM is only a specification!**
 - you can program your own!
 - certain features are prevalent, like stacks (see CS211: push and pop values onto stack)
- **SaM:**
 - approximate JVM with *stack machine*
 - has areas for stack, heap
 - download from course website



24