Computer Science 5JA

Introduction to Computer Programming (Java™ Flavor)

- No pre-requisites
  - But primary goal is to *learn how to program* in Java – requires practice (and commitment)
- Designed for non-majors
  - CS pre-majors welcome to prepare for CS 10
    - But should skip to 10 if they already know how to program *in any language*

What CS 5JA is *not*

- *Not* for people with zero computer experience just wanting to know how to use computers
  - Attend short courses offered by IC instead (or first)
    - Word processing, spreadsheets, web browsing and e-mail, …
  - Such people are frustrated by CS 5JA's requirements
- *Not* a comprehensive course in the Java programming language
  - Text and lectures focus on a "strategic subset" of Java – to teach fundamental programming concepts
  - Must learn advanced Java on your own – but CS 5JA covers ways to go about learning such things, and CS 10, 20, 50, … cover more ways (but still not all of Java!)

Course structure

- Mostly follows the text, Chapters 1-8
  - Intro to computers, programming, Java: Ch. 1 & 2
  - Data, memory, operators, …: Ch. 3, and App. A-D

  - Control structures: Ch. 4 & 5, and App. I
  - Graphical programming (aside): probably supplement
  - Writing and using methods: Ch. 6, and App. G

  - Arrays and other collections: Ch. 7, and supplement
  - Designing classes & using objects: Ch. 8
- Special Java topics throughout – as time permits

Requirements

- Homework assignments – 30 percent of total grade
  - Mostly programming projects
- 2 midterm exams – 20 percent each
  - Wednesday, Jan. 28, and Friday, Feb. 20
- Cumulative final exam – 30 percent
  - Thursday, Mar. 19, 9-10:30am (1.5 hours, not 3 hours)
- Course web pages are mandatory reading
  - www.cs.ucsb.edu/~mikec/cs5ja/ - updated regularly
- Questions about the requirements?

To do – this week

- Read chapters 1-2 in the text
  - In general, *read ahead* of the lectures – see Syllabus
- Confirm access to JDK (SE), version 1.5 or higher
  - JDK is the Java Development Kit: necessary to create and execute Java programs (SE is standard edition)
  - Instructional Computing (IC) lab is a good backup
- Compile and execute at least one sample program
  - See chapter 2 examples
- Go to a discussion section Friday

What is a computer?

- Webster: “one that computes”
  - Compute: “to determine esp. by mathematical means”
  - Abacus?
  - Slide rule?
- Person?
  - Actually a 1940s job title!
    - Ballistics project for U.S. War Dept. – computed artillery trajectories by desk calculator – up to 30-40 hours each
  - Led to the first electronic computer – ENIAC

ENIAC – electronic numerical integrator and computer – 1945

- 100 feet long, by 10 feet high, by 3 feet deep
- 30 tons!
- 17,468 vacuum tubes, 70,000 resistors, and 6,000 switches
- Trajectories computed in 30 seconds instead of 40 hours

Electronic computer hardware

- Central processing unit – CPU
  - Controls the other components, performs arithmetic, directs the flow of all data
- Main memory – a.k.a. RAM (“random access”)
  - Fastest access, but short term – power must be on
  - States are binary – e.g., electronic pulse up or down
  - Also ROM (“read-only”) – mostly for starting up
- Secondary storage – disks, CDs, tapes, …
  - Long-term memory – usually magnetic, so no power
- Input/output – I/O – keyboard, mouse, monitor, …

Hardware evolution

- Vacuum tubes phased out long ago
  - Suggest: *Player Piano* by Kurt Vonnegut
  - Replaced by transistors – faster, smaller
  - Then by integrated circuits – "chips"
    - Currently tens of millions of transistors
    - Continually getting faster, smaller, cheaper, …
- I/O and storage improvements too
  - Direct wiring → IBM cards → keyboard → wireless
  - Line printer → dot-matrix → laser/color & more
  - Disk drums & 9-track tapes → 50 gigabyte drives …

Today: “Personal” Computers

PC hardware – schematic

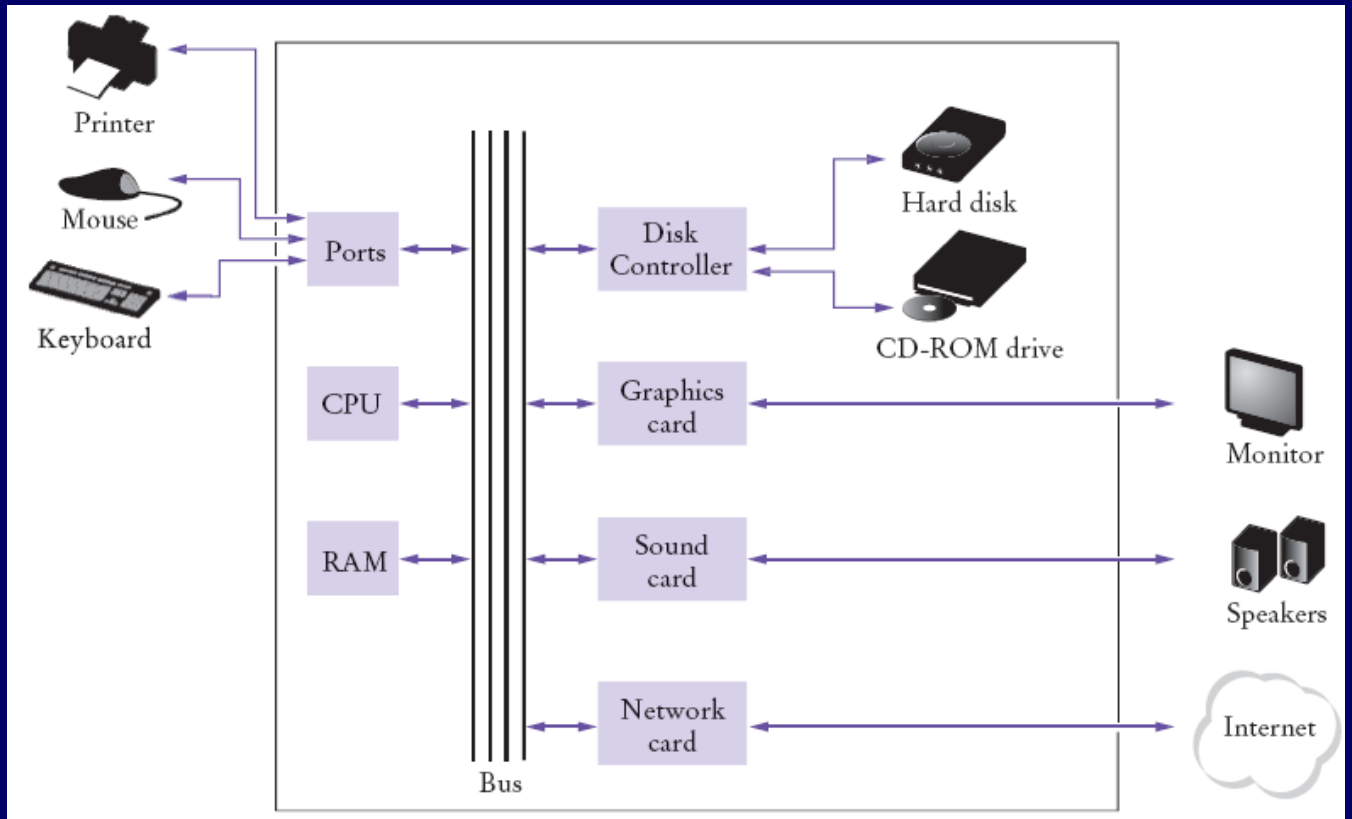

Figure 5 Schematic Diagram of a Computer

What is programming?

- Basically: instructing a computer what to do
- Programs – a.k.a. “Software”
  - Includes operating system, utilities, applications, …
  - Computer just sits there until instructions fed to CPU
- Machine language – basic CPU instructions
  - Completely numeric – i.e., computer “readable”
    - e.g., `43065932752`, might mean add (operation `43`) value at memory address `065` to value at address `932` and store result at address `752`
    - But in binary form, of course – `1001101…`
  - Specific to particular computer types – not portable

Programming languages

- Assembly language – 1st real advance
  - Human-readable instructions – translated to machine language by assembler programs
    - e.g., `ADD  X  Y  T`
    - Symbolic names represent operations and memory addresses
  - Very basic – lots of instructions to do simple things
  - Still processor-specific
- High-level languages – much bigger advance
  - Easier to write/read: `result = (first + second)`
  - Translated to assembly language (usually) by compiler programs
    - Same code works on many types of processors

High-level language history

- Procedural languages – focus on *functions*
  - Fortran (by IBM, 1957) – first high level language
    - Easy to learn – spawned thousands of new programmers
  - C, Pascal, others – developed through 1970s
    - Even easier to learn/use – ever more programmers into 1990s
- Object-oriented languages – focus on *objects*
  - C++ (early 1980s), …, Java (1996)
  - Idea is to build objects – then let them perform tasks
  - Many side benefits – facilitates team efforts, "software reuse", rapid application development, …

Java – became popular quickly

- Code looks like C (and C++) – familiar for many existing programmers
  - Object-oriented without complexities of C++
- Killer API (application programmers interface)
  - Built-in networking features
  - Graphical user interface (GUI) objects
  - Threads, media support, …
- Is free!
- Java virtual machine – JVM – "Write once, run anywhere."

A simple Java program

- Java “programs” are actually classes
  - A class defines a *type* of object
- A first java application: class Hello
  1. Create file called `Hello.java`
  2. Compile – `javac Hello.java`
     (creates bytecode file named `Hello.class` if successful)
  3. Execute – `java Hello`
     (invokes JVM)