Lecture Set 1: Introduction

Today’s topics:
1. Course information
2. Tools needed for this course
3. Computer terminology basics

CMSC 131

- Name: “Object-Oriented Programming I”
- Instructor: Jan Plane
- Class meetings
  - Lecture sections
    - 4 lecture sections
    - 2 instructors
  - Lab sections
    - 8 lab sections
    - 10 teaching assistants
Coordination of Sections

- Five sections total of CMSC 131
  - two lectures taught by me
  - other two lectures taught by Fawzi Emad
  - Ten TAs in total for the 4 sections
- All sections will be closely coordinated:
  - Same lecture material on same day
  - Same projects
  - Same labs
  - Coordinated exams
- Lab/Discussion/Recitation Sections
  - exercises – laptops
  - quizzes
  - new material occasionally

What Is This Course?

- A fast-paced introduction to techniques for writing computer programs!
  - Skill Development in Programming
  - Conceptual Understanding of Programming
  - Not really “computer science”
- There will be quite a bit of work but assumes you are starting at level 0.
- Keys to success
  - Attend all classes and lab sections
  - Start assignments early – and continue until you truly understand
  - Get help early if you are having trouble – 2 instructors & 10 TAs
  - Study every day
    - it doesn’t work to cram for these exams
    - ask questions as soon as you realize you are confused
  - Check announcements on course web-page every day
Textbooks & Course Web-Page

- Check daily!
- Review:
  - Announcements
  - Syllabus
  - Contact
  - Schedule
  - Lecture slides - outlines

Study Questions

- Available on web-page
  - Login: study
  - Password: daily
- Look at them on evenings before class; they will help you keep up
Course Software

- Eclipse
  - An IDE (integrated development environment)
  - You will use it for writing Java™ programs
  - Access to Eclipse (it’s free!)
    - You can install it on your own machine: [http://www.cs.umd.edu/eclipse](http://www.cs.umd.edu/eclipse)
    - Also accessible in some Workstations at Maryland (WAM) labs around campus: [http://www.wam.umd.edu/](http://www.wam.umd.edu/)

- CVS (Concurrent Versions System)
  - A version-management system
  - You will use it for submitting your projects
  - We will talk more about this later

Tools for Writing Programs

- The old days
  - Text editor: used to create files of source code
  - Compiler: generate executables from source
  - Debugger: trace programs to locate errors

- Today: IDE = “integrated development environment”
  - Text editor / compiler / debugger rolled in one
  - Examples: Eclipse, Visual Studio, etc.
Basics of Eclipse

- Eclipse is used to:
  - Create
  - Edit
  - Compile
  - Run
  - Debug
  programs (for this class, Java programs).

Basics of Eclipse-speak

- **Project**: collection of related source files
  To create a program in Eclipse:
  - Create a new project
  - Create files in the project

- **Perspective**: framework for viewing and/or manipulating programs

- Important perspectives in this class:
  - **Java**: for creating, running programs
  - **Debug**: for tracing, removing errors in programs
  - **CVS repository**: for interacting with assignment-submission system
Eclipse Demo

Class Projects with CVS

- You will use Eclipse for Java programming in this course
- How will you:
  - obtain (check-out) files that are supplied to you
  - save (commit) the files for later work
  - turn in (submit) when you are finished class projects?
- **CVS (= Concurrent Versions System)**
  - Tool for project-file management
  - Maintains versions, etc.
  - Allows different sites to work on same project
CVS Worldview

- Files in "repository" (Server)
- Files (local copies) (Client 1)
- Files (local copies) (Client 2)

CVS in More Detail

- CVS server maintains current versions of files in project (= "repository")
- To access files from another machine ("client"), repository files must be "checked out"
- Changes to files on client may be "committed" to server, with changed files becoming new version
- (Once a repository is checked out by a client, subsequent versions may be accessed via "update")
What’s Needed for CVS?

- Server machine
  
  *For CMSC 131, CS linuxlab machines*

- User authentication
  
  *For CMSC 131, student linuxlab accounts*

How CMSC Project Submission Works

- Repository created for each student linuxlab account

- You check out repository to start work on project

- When you “save” changes in Eclipse, “commit” automatically invoked by plug-ins

- You “submit” when finished using Eclipse (UMD plug-in handles relevant CVS commands)
To Checkout a Project

1. Set repository location
   - Change to “CVS Repository Exploring” perspective in Eclipse (“Window -> Open Perspective” …)
   - Right-click in “CVS Repositories” panel and select “New -> Repository Location…”

Adding a CVS Repository

Common to everyone
Your linuxlab username
Your linuxlab password
Don’t forget to set this!
To Checkout a Project (cont.)

1. Open repository name, then “Head”
2. Right-click on project name to save
Working on Project

- When you switch back to “Java” perspective, your project is now there!
- When you save in “Java” perspective, changes are automatically committed to CVS repository.

Submitting the Project

- Edit the file
- Make sure it runs correctly
- Submit the project for grading
- Go to submit.cs.umd.edu to see test results
  - Public tests
  - Private tests
  - Release tests
    - give limited feedback (first two failed tests give more)
    - costs you “tokens” – usually 3 to start with
    - spent tokens regenerate in 24 hours
Study Questions

- Login: study
- Password: daily

Computer Organization

- Hardware: physical parts of computer
  - Monitor, mouse, keyboard
  - Chips, boards
  - Cables, cards
  - etc.
- Software: non-physical (“logical”) parts of computer
  - Programs = instructions for computer to perform
Hardware Overview

- **CPU** = central processing unit
  - Executes the "instructions" in programs
- **Main memory** = random-access memory = "RAM"
  - Stores data that CPU accesses, including instructions
  - FAST, but temporary; wiped out when computer is shut off!
- **Secondary memory**: Hard disks, CDs, DVDs, flash memory, etc.
  - Stores data that can be loaded into main memory
  - SLOWER, but permanent
- **I/O devices**
  - How you communicate with your machine
  - Keyboard, monitor, mouse, speakers, etc.
- **Networking equipment**
  - How others communicate with your machine
  - Networking "cards", cables, etc.

Main Memory

- Computer data consists of off and on pieces (often written as 0's and 1's)
- **bit**: A single cell in main memory that can hold either a 0 or 1
- **byte**: A sequence of 8 bits
- **word**: Smallest unit of addressable memory (often a sequence of 4 bytes)
- **Main memory**: table of bytes indexed by "addresses"

<table>
<thead>
<tr>
<th>Address</th>
<th>Byte value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 1 1 1 0 1</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 1 0 1 1</td>
</tr>
<tr>
<td>4</td>
<td>1 1 0 0 0 1 0 0</td>
</tr>
</tbody>
</table>
How Many Different Values in a...

- Bit?
  - 2
- Two bits?
  - $4 = 2 \times 2$
- Byte?
  - $256 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8$
- Word?
  - $4,294,967,296 = 2^{32}$

Other Standard Terminology

- 1 KB = 1 “kilobyte” = $2^{10}$ bytes = 1,024 bytes
- 1 MB = 1 “megabyte” = $2^{10}$ KB = 1,024 KB
- 1 GB = 1 “gigabyte” = $2^{10}$ MB = 1,024 MB
How Are Characters, Etc., Represented?

Via *encoding schemes*

Example: ASCII (American Standard Code for Information Interchange)

- Standard for encoding character values as bytes
- In ASCII:
  - ‘A’ 01000001
  - ‘a’ 01100001
  - ‘,’ 00101100
  - etc.

There are other character encoding schemes also: Shift-JIS, Unicode, etc.

Software Overview

1. **Operating system**: manages computer's resources; typically runs as soon as computer is turned on. Typical responsibilities:
   - Process management
     - Determines when, how programs will run on CPU time
   - Memory management
     - Controls access to main
   - I/O, window system, network control
     - Performs low-level drawing, communication operations
   - Security
     - Manages user IDs, passwords, file protections, etc.

2. **Applications**: programs users interact directly with; usually are explicitly run.
   - Examples:
     - Word processors
     - Games
     - Spreadsheets
     - Music software,
     - Etc
How Programs Are Executed

Program “foo” initially stored in secondary storage → Program copied into main memory → CPU executes program instruction-by-instruction

Programming Languages

- Used to write programs that run on computers

- Generations of programming languages
  - 1st (1GL): machine code
  - 2nd (2GL): assembly code
  - 3rd (3GL): procedural languages
  - 4th (4GL): application-specific languages
  - 5th (5GL): constraint languages
 Recall: computer data is 0’s and 1’s.
 In machine code, so are programs!
  - Program: sequence of instructions
  - Machine code: instructions consist of 0’s and 1’s
 Next slide: example machine code instruction from MIPS (= “Microprocessor without interlocked pipeline stages”) architecture
  - Popular in mid-, late 90s
  - Instructions are 4 bytes long

Example MIPS Instruction

“Add data in addresses 1, 2, store result in address 6”:
0000000001000100011000000100000

???

000000 00001 00010 00110 00000 100000

opcode  2nd address  shift amount

1st address destination address function specifier
Programming in 1GLs

Problem with 1GLs: Who can remember those opcodes, addresses, etc. as 0’s, 1’s?

Solution (1950s): assembly language
- Use mnemonics = descriptive character strings for opcodes
- Let programmers give descriptive names to addresses

MIPS example revisited:
- add $1, $2, $6
- instead of
- 00000000001000100011000000100000
- for “add contents of addresses 1, 2, store result in 6”
Assemblers

- Computers still only work on machine code (1GL)
- Assembly language is not machine code
- Assemblers are programs that convert assembly language to machine code (= “object code”)

3rd Generation: Procedural Languages

- Problems with 2GLs
  - Platform dependency
    - Different kinds (architectures) of computers use different instruction formats
      - E.g. x86, Pentium, 68K, MIPS, SPARC, etc.
    - 1GL / 2GL programs written for one kind of machine will not work on another
  - Low level: programs difficult to understand
- Solution (60s -- now): procedural languages
  - Higher-level, “universal” constructs
  - Examples: Cobol, Fortran, Algol, Pascal, C, C++, Java, C#
Compilers

- Computers can only execute machine code
- Compilers are programs for translating 3GL programs ("source code") into assembler / machine code

Interpreters

- Another way to execute 3GL programs
  - Interpreters take source code as input
  - Interpreters execute source directly
  - Much slower than compiled programs
- Debuggers are based on interpreters
  - Debuggers support step-by-step execution of source code
  - Internal behavior of program can be closely inspected
# Object Oriented Terminology

- **Procedural Languages**
  - have procedures that can be reused
- **Object Oriented Languages**
  - centered on the objects
- **object**
  - principal entities that are manipulated by the program (nouns)
- **class**
  - a “blueprint” that defines the structure for one or more objects
- **method**
  - java term for a “function”, a “procedure” or a “subroutine”
  - this is the code that does something (verbs)
- **main method**
  - a special method that defines where program execution begins
- **statements**
  - individual instructions