

# Comp 110 Introduction Algorithms and Programming

- 500 BC Abacus was first used by the Babylonians
- 850 AD Indian numbers with 0 (possibly 400 BC)
- 1300 Chinese abacus
- 1600s Mechanical calculators by Pascal, Leibniz's, Mueller,...
- 1822 – 1871 Difference engine and Analytical Engine, Babbage  
Analytical Engine conceptually modern computer, programmable.  
first programmer, Ada Lovelace
- 1848 Boolean algebra, Boole
- 1880s Mechanical, desktop, printing calculators.
- 1896 IBM (The Tabulating Machine Co.) founded by Hollerith.
- 1937 Turing machine formulated, Turing
- 1938 Electro/mechanical programmable binary calculator, Zuse
- 1940 Vacuum tube based calculators

1943 - 1959    **First generation computers** -- punched cards & vacuum tubes

1951    UNIVAC-1 (all purpose commercial computer)  
High level language compiler, Hopper

1957    FORTRAN (FORmula TRANslation science, engineering).

1958    LISP (LISt Processing -- AI language)

1959 - 1964    **Second generation computers** -- transistors & printed circuit boards

1960    ALGOL (ALGOrithmic Language) structured, procedural  
Operating Systems with time sharing (many interactive users)

1961    COBOL (COmmon Business Oriented Language) Hopper

1963    Interactive graphics, Sutherland

1964 - 1972    **Third generation computers** -- integrated circuits

1964    Digital's PDP-8 mini computer (12 bits)

- 1965 BASIC (Beginners All Purpose Symbolic Instruction Code),  
Kurtz & Kemeny  
Mouse, hyper text, interactive editor, Englebart  
Control Data's CD6600 supercomputer
- 1967 Simula-67, Dahl, Object Oriented Programming (OOP) language
- 1970 ARPANET – internet  
UNIX, Bell Labs, Thomson & Ritchie
- 1971 Intel 4004 4 bit microcomputer, Hoff
- 1972 **Fourth generation computers** – LSL/VLSL chips  
C language, Ritchie. Unix kernel re-written in C
- 1973 Pong video arcade game, Bushnell  
Ethernet – wired networked computers
- 1975 MITS Altair 8800 personal computer ( Intel 8088 8 bits)  
Microsoft, BASIC for Altair 8800
- 1976 Apple Co., 1977 Apple II ( CPU 6502 8 bit)

- 1981 IBM PC (Intel 8088) with MS DOS  
Xerox Star, bit map graphics – GUI mouse, Windows, Icons, ...
- 1983 C++, Stroustrup (Object Oriented Programming)  
Smalltalk-80, Goldberg (OOP language)
- 1984 Apple Macintosh ( Motorola 68000 - 16 bit)  
Laser printers, Hewlet Packard
- 1985 Microsoft Windows
- 1986 Computer virus: boot, Basit & Amjad, Virdem, Burger
- 1989 World Wide Web (http), Berners-Lee  
MP3, Fraunhofer-Gesellschaft – Grill et. al.
- 1991 Web Berners-Lee – http, no GUI ...  
Linux Torvalds (pre dated by Tannebaum's Minix)  
WIFI NCR - Hayes

- 1993 Doom, game, Id Co.  
NCSA Mosaic, graphical web browser
- 1994 Java, Gosling  
Amazon, Bezos, e-commerce  
Yahoo, Filo & Yang – search engine, portals  
eBay, Omidyar, on-line auction e-commerce
- 1996 Nokia Communicator, smart phone – cell phone + computer
- 1997 IBM's Deep Blue beats human chess champion  
Ultima, Garriott, Ultima OnLine – internet games
- 1999 Napster Fanning, peer – to – peer file sharing  
craigslist, Newmark, people – to – people personals
- 2002 C#, Hejlsberg  
Friendster, Abrams, social networking
- 2004 WOW, Blizzard, World of Warcraft
- 2005 You Tube, Chen, Hurley & Karim – streaming video
- 2007 iPhone, Apple – cell phone, cell/wifi browser

# Algorithms solve a problem (more than one solution)

Algorithm = steps (instructions) + resources (data)

complete      all data required, all steps needed

specific      sequence / control of steps, amount / types  
of steps

Algorithm formats (one is not enough)

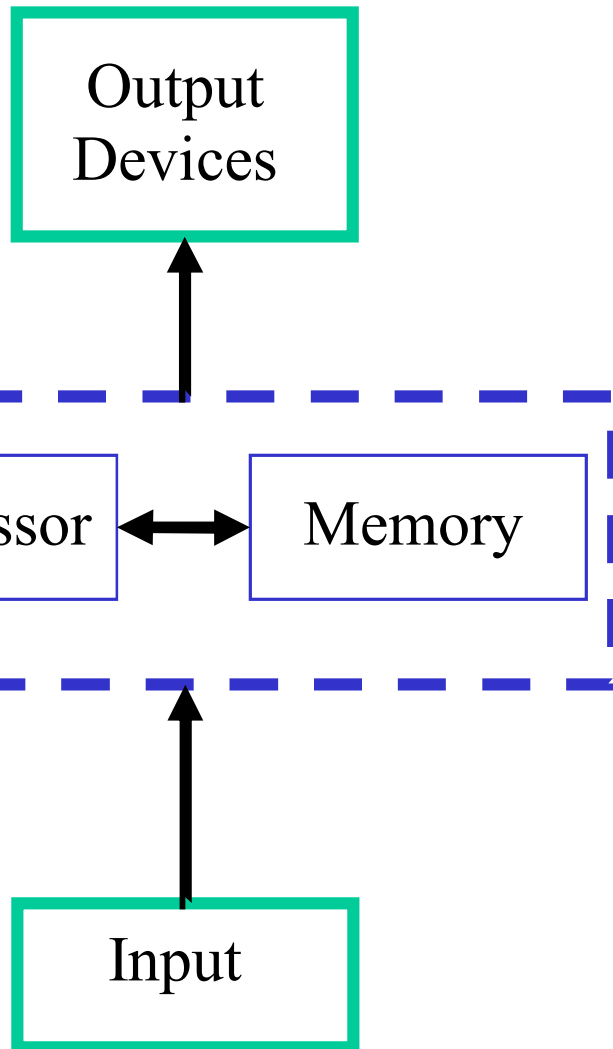
natural language (ie English)

programming language (Java)

diagram, flow chart, block flow, UML

pseudocode -- mix of natural and programming language

# Standard Hardware Organization



Processor (CPU)

Central Processing Unit

Interprets, executes, instructions

Memory

main & auxiliary

holds data and instructions

CPU and memory are physically housed together

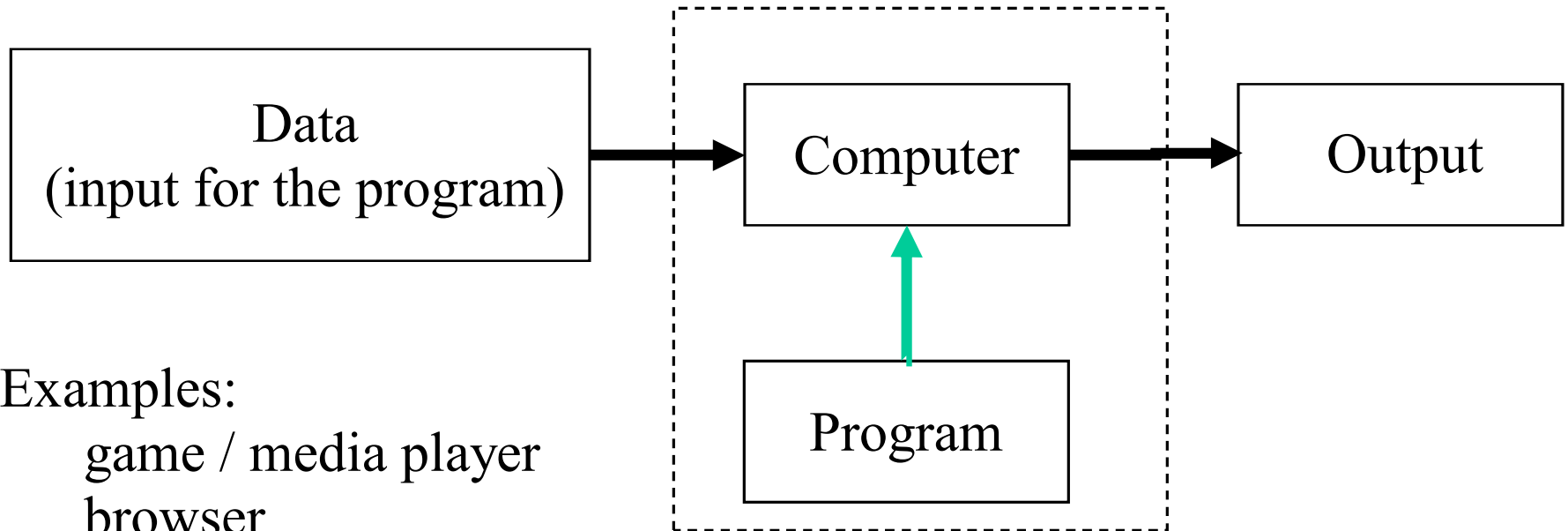
Input device(s)

mouse, keyboard, etc.

Output device(s)

video display, printer, etc.

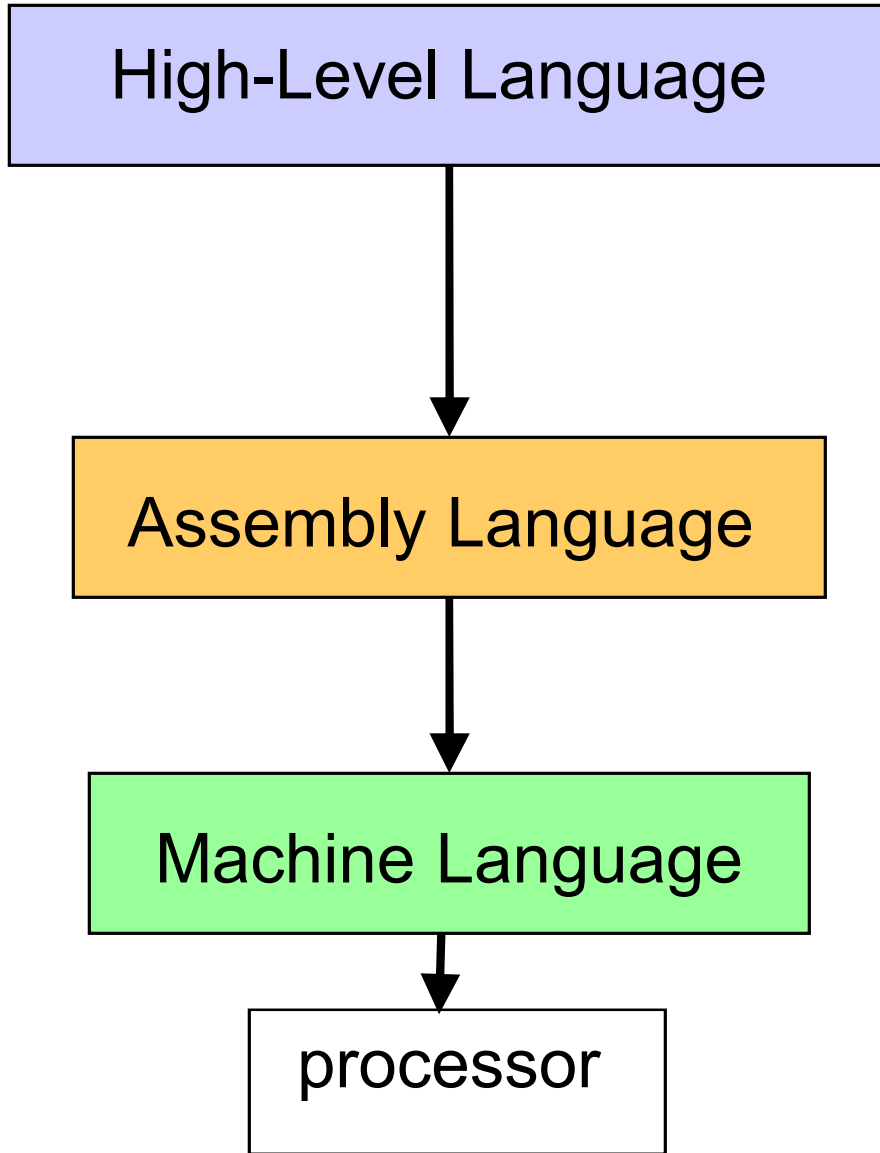
Program solves problem with computer  
a set of instructions for computer to follow  
a set of resources (variables) for computer to use



Examples:

game / media player  
browser  
spreadsheet  
text editor / word processor  
compiler / interpreter  
operating system

# Level of Programming Languages



Readable, natural / mathematical language.

Not executable, portable (text)

Java, C++, Ada.

High programmer productivity.

Readable, abbreviated, text.

Not executable system dependent.

Intermediate stage of translation.

Low programmer productivity.

Executable, system dependent.

Not very readable,

binary (...00101...).

Lowest programmer productivity.

# Translators: Compilers and Interpreters

Most compilers convert source code into object code in one step. C++ compiler.

Compile once, execute many times.

Optimizes execution speed.

Most interpreters convert source code into object code and execute the object code, line by line.

Easier to debug.

Need translator for every machine language

```
graph TD; A[Source code high level language] -- Compile --> B[Object code machine language]; A -- Interpret --> B;
```

Source code  
high level language

Compile

Interpret

Object code  
machine language

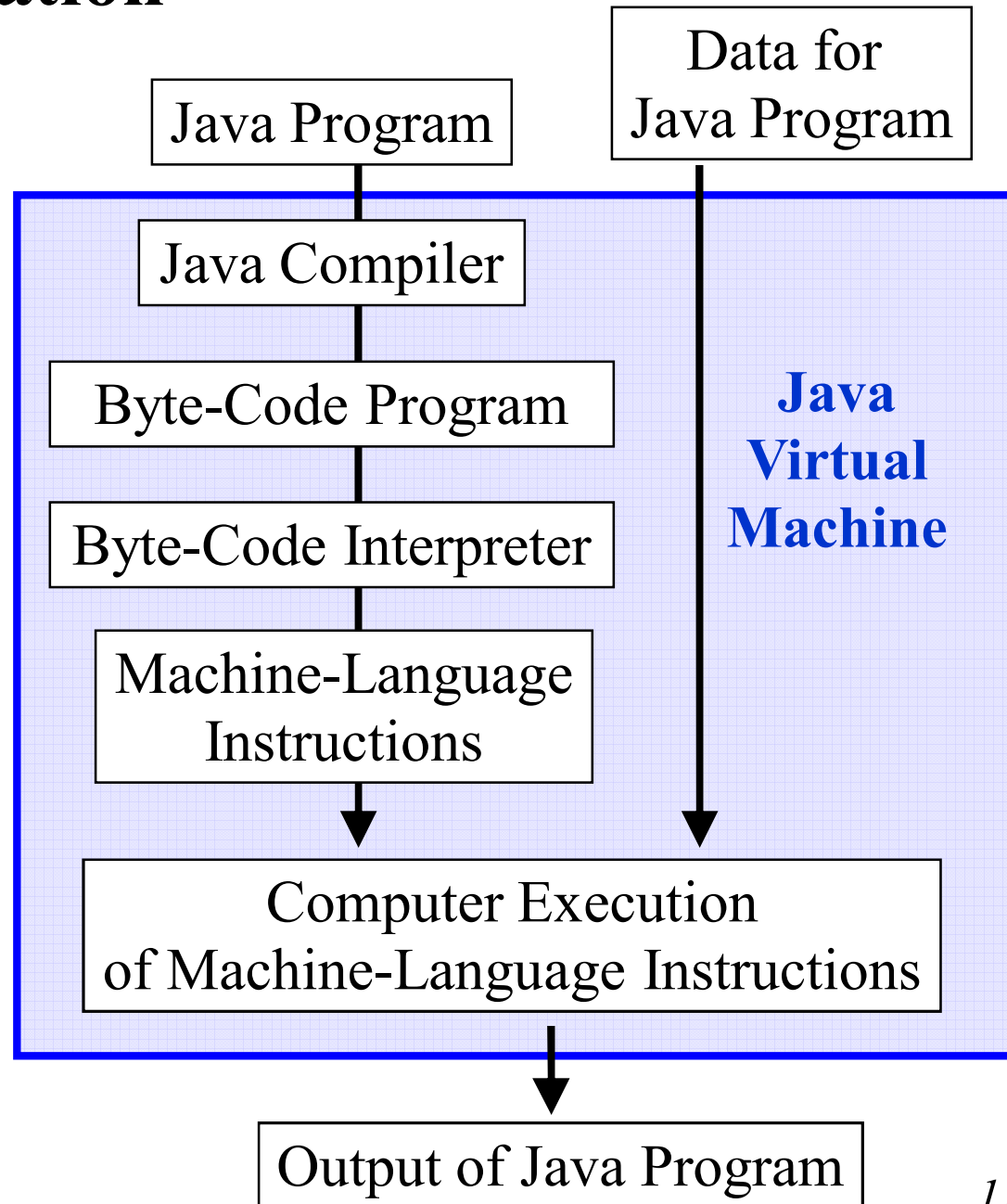
# Java Program Translation

Both Compilation and Interpretation

Compiler translate to Intermediate Code:  
“Byte Code”

portable low-level code  
like assembly code,  
hardware independent  
not readable

Interpreter translates from generic byte code to hardware-specific machine code and executes the code.



# Java Properties

High Level Language

readable, high programmer productivity

Portable, system independent

compile once -- optimize byte code (class files)

interpret often for execution on Virtual Machine (system dependent)

Object Oriented Programming (OOP) language

Exception Handling -- can catch and respond to run time errors

Multimedia programs – graphics and sound capabilities

Graphical User Interface components (AWT / Swing)

Distributed, networked programs

applets for web browsers

features (classes) for client / server applications

# Object-Oriented Programming

A design and programming technique

Some terminology:

**object** - usually a person, place or thing (a noun)

**method** - an action performed by an object (a verb)

*type* or **class** - a category of similar objects (such as automobiles)

Objects have both data (variables) and methods (“functions”)

Objects of the same class have the same data elements and methods

Saves development time (and cost) by reusing code

once an object class is created it can be used in other applications

Easier debugging

classes can be tested independently

reused objects have already been tested

# Program Design Process

**Design, then code**    avoid the rush to judgment!

Design process

- define the problem clearly

- design objects your program needs

- develop algorithms for the methods of objects

- describe the algorithms, usually in pseudocode

- write the code

- test the code

- Fix any errors and retest and often redesign.

Solve a Simpler Problem First

- Break larger task into smaller parts

- Solve each part, program each part, test each part

- Combine the parts.

Think of another approach or design solution

# Syntax Errors

A “grammatical” error

Caught by compiler (“compiler-time error”)

Automatically found, usually the easiest to fix

Cannot run code until all syntax errors are fixed

Error message may be misleading

error is near or before line specified by error message

Example:

Misspelling a command, for example “rturn” instead of “return”

# Run-Time Errors

An execution error (during run-time)

Not always so easy to fix

Hard to know where the error occurred

Error message usually not be helpful

Example:

```
sum / count;
```

Division by zero - variable count == 0

If your program attempts to divide by zero it automatically terminates and prints an error message.

# Logic Errors

**Just because a program compiles and runs without getting an error message does not mean the code is correct !**

An error in the design (the algorithm) or its implementation  
code compiles without errors  
no run-time error messages  
but incorrect action or data occurs during execution

Generally the most difficult to find and fix

You solved the wrong problem correctly

Need to be alert and test thoroughly  
think about test cases and predict results  
*before* executing the code

# General Benefit of Learning to Program

Designing and solving problems

Planning what sequence of steps are needed to transform resources (data) into desired solution

Structured, ordered, thinking.

Applicable to other fields:

Creative writing – novels or screenplays (plot sequence, character development).

Business process planning and analysis; logistics, operations

Engineering design and implementation

Mathematic problem perception, approach and solving

Teaching – lesson plans and curriculum development