



# Introduction to Computers and Java



MAGICAL







INPUT



INPUT











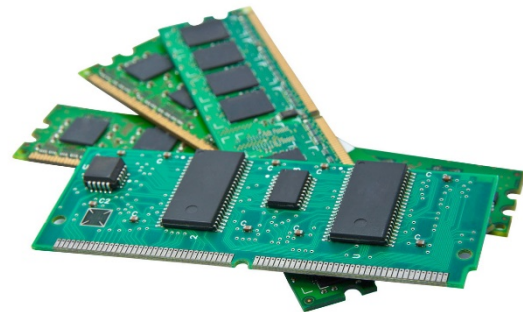


# HARDWARE

CPU



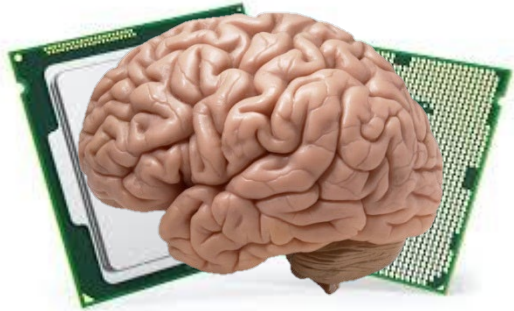
Memory



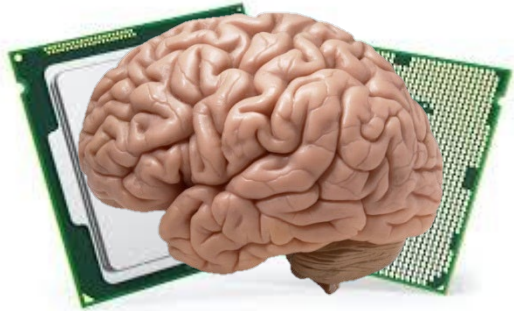
# HARDWARE



# HARDWARE

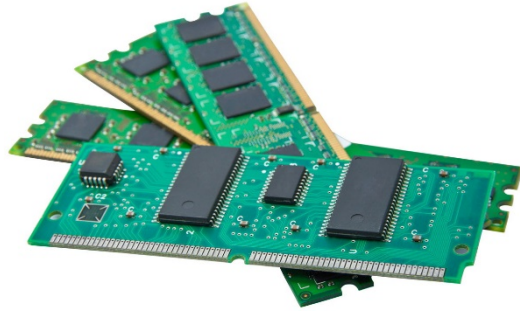


# HARDWARE

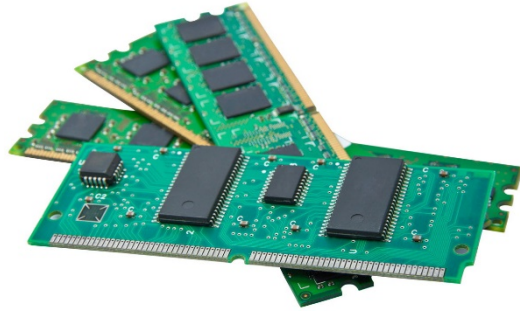




# HARDWARE



# HARDWARE



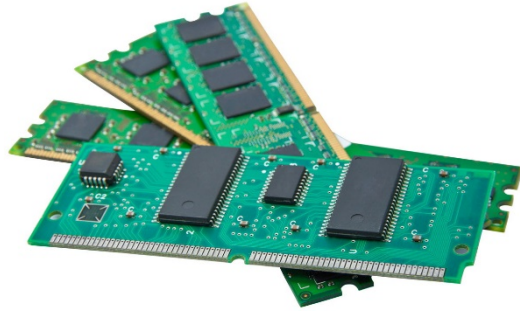
RAM

# HARDWARE

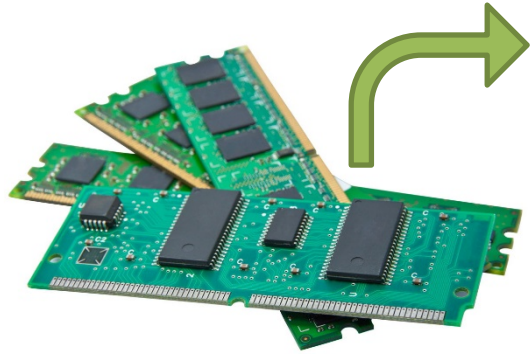


Secondary

# HARDWARE



# HARDWARE



Memory	
Addresses	Values
...	
256	01000001
260	01000010
264	01000010
268	01000001
...	

# Bits and Bytes

- Bit – a digit with a value of 0 or 1
- Byte – Consists of 8 bits
- Address – The numbered location where each byte resides
- All data is encoded as a 0 or 1
  - Everything is a number
- When more than 1 bytes is needed then several adjacent addresses are used

Memory	
Addresses	Values
...	
256	01000001
260	01000010
264	01000010
268	01000001
...	

# Files and Folders

- File – large group of bytes stored in secondary (auxiliary) memory
  - Files have names
  - Most files have extensions
- Folder (Directory) – group together multiple files
- Java programs are stored in files
  - Source code have the extension “.java”
  - Byte-Code have the extension “.class”

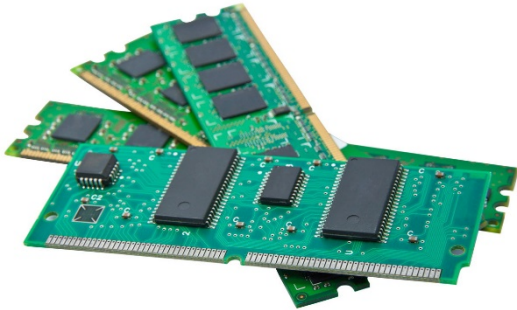
Memory	
Addresses	Values
...	
256	01000001
260	01000010
264	01000010
268	01000001
...	

# Running Software

Secondary



Main



CPU





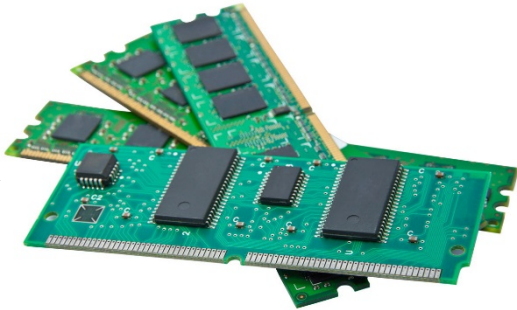
# Running Software

Secondary



Load →

Main



CPU

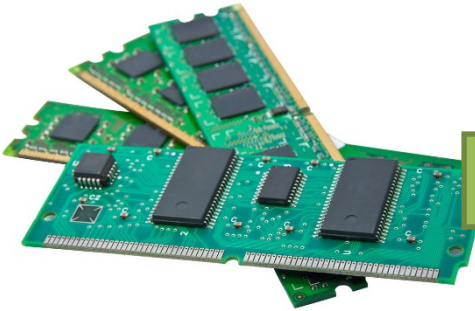


# Running Software

Secondary

Main

CPU



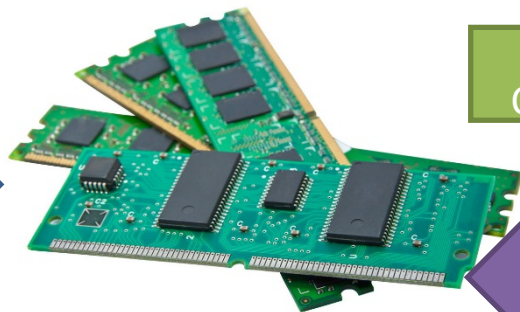
# Running Software

Secondary



Load

Main



CPU

Run Code

Store Info





SOFTWARE



Algorithm ('algə,riT\_Həm) noun  
A set of instructions to  
solve a problem



Program (prō, gram) *noun*  
A set of instructions for a computer to follow.



# Programming Languages

# Programming Languages







**LOW LEVEL**

# Programming Languages

# LOW LEVEL

```

0001 0001001011 01 0 01010 010110 01100 00010 01010101110 001010 00100 000111
01 1010 001 111010 001001010 101011000101010 010 10001001 010 001 01001 0 100010
00 01 1 01 1010 010 00101 110100 1 1000 010 11 0 0101000 10110 01011 010010010101
001 10101001011 10 01001001 00010111 00010100 100100 00100 01001 010010010 010
10 10 0 00 011001 0 010101011 010110010010110101011101 00 001 010 100 1000111
0 011100110 0 0010 0010 0 101000101010 001101010100001011010 01 100100 010101
00 010101000 0 1 010 00 00 010010 111000 0 0100100000100 01010 1001001 010 010
001011 1 11010 010 0101011000010100100011010 000 0110101 1100 0010 10100
0 010101000 1110 0100 0010 010111 0010 01000100100 1001000 0 100100 0010001
001011 01000 11 0100010 10100 0101100001010001 0 000100 000 0100100 0010000 0
0010111 011010001 0010 01101000101 01010 1 1010 0100001011010 01110 1001001010
100 010010 101 10 1 000 0 010 011 1 1001000101010 01111 1000 001 0100 010001110
0 1 10110001 1 101100 0 1010 0101 00 1 101000100001 0100 010000001 01 0010
0 10 1 10 110 0100 0100101 0 0001000 00 110101 100010 1101010 110 10 10010101
0010111 01101010010 001 011010001 00 0100110101 0 001 11 01010 10 10 100 0101
0 0 1 1010 11101000 00100101011100010 01000100 000100 001010010 100 0010 010
0 1 10110 0010111001 1001 0101 01 101110010 0100 100 000100100100010000 00 0 010
01 11 10 1010 0 10 0 1 1010 00010 0 10 0 110 010000 01110101 100 00010 1
0 0101010 011 101010 10 10 00001 10001010 0 1001 001001010 0 0 0010 0 10
0010111011101001000 01010 000100 0100 10 01010000 0110101011 01 00 10
10 1010100 111010 010 0 0111010100 00 10110 0101011101 01001001 100 00 11
010 01 010101 10101 100101 01010 000101010 1001000 00 00101001001010 1 0010
0 10 111011 010 001 100101 11 10 010 001 100 110 1 1 0001 11 010 11010 1001 101
100100100101110 00101010 11 01 1100 000 011 010 0111010 0100100 001 000011
001 11 101 0010101000 0 01101 0 10 010100 1010 010000 0110101 110 0 1 010 01
00 010101 0 111010 010010 0 0 0 11 00 01 1 001 0100010 00101 01 0 1000001

```

```

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

```

C010 B6 80 04	INCH	LDA A ACIA	GET STATUS
C013 47		ASR A	SHIFT RDRF FLAG INTO CARRY
C014 24 FA		BCC INCH	RECIEVE NOT READY
C016 B6 80 05		LDA A ACIA+1	GET CHAR
C019 84 7F		AND A #\$7F	MASK PARITY
C01B 7E C0 79		JMP OUTCH	ECHO & RTS

# Programming Languages

# LOW LEVEL

```

1001 100100101 01 0 01010 0101110 01100 00010 01010101110 001010 00100 000111
01 1010 001 111010 001001010 10101100001010 010 10001001 010 001 01001 0 10010
00 01 1 01 1010 010 00101 110100 1 1000 010 11 0101000 10110 0110 0101000101
001 10101001011 10 01001001 0001011 00 010100 100100 00100 01001 010010010 010
10 10 0 00 0110010 0 01010101 010110010010110101011101 00 0010100 1001000111
0 011100110 0 0010 0010 0 1010001010010 001101010100000111010101 100100 010101
00 010101000 0 010 00 00 010010 111000 0 0100100000100 01010 1001001 010 0010
001011 1 11010 010 01010110000100010001101010000 011010101 1100 001010 0
0 010101000 01110 0100 0101 0010 01001001000 1001000 00100100000001
00101 01000 11 0100 0 0100 01000001000 0000000000000000000000000000000000000
0010111 011010001 00000000000000000000000000000000000000000000000000000000000
100 010010 101 10100 0101 010 010 00010000001000 0001 0100 01000111
0 1 10110001 1 1 10100 0 101 0 101 0 1 010010010001000 0100000001 01 0010
0 10 1 10110 0100 01001010 0 0001000 00 110101 100010 1101010 110 10 100 0101
0010111 01101010010 001 0111 001 00 01001 0101 0 001 11 01010 10 10 100 0101
0 0 1 1010 11101000 001 00000000000000000000000000000000000000000000000000000
0 101010 001011101 1001 01 01 00000000000000000000000000000000000000000000000
01 11 10 1010 0 10 0 1 10 000000000000000000000000000000000000000000000000000
0 0101010 011 101010 10 10 000000000000000000000000000000000000000000000000000
00101110110101001100 01010 000 0100 01001 0 01010000 011010101 01 000 10
10 1010000 111010 010 0 0111010100 00 1010 0101011101 01001001 0100 00 11
010 01 010101 10101 100101 01010 000101010 1001000 00 0010100001010 1 0010
0 10 111011 010 001100101 11 10 01 001 100 110 1 1 000111 10 01 10010 1001 101
100100100101101 00101010 11 01 100 000 011 010 0111010 0101000001 000011
001 11 101 010101000 01101 0 10 010100 1010 010000 0110101 110 0 1 010 01
00 010101 0 111010 010101 0 010 11 00 01 10001 000010 00101 01 0 1000001

```

## Machine Code

```

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

```

## Assembly

```

C010 B6 80 04 INCH GET STATUS
C013 47 ASR A SHIFT RDRF FLAG INTO CARRY
C014 24 FA BCC INCH RECIEVE NOT READY
C016 B6 80 05 LDA A ACIA+1 GET CHAR
C019 84 7F AND A #$7F MASK PARITY
C01B 7E C0 79 JMP OUTCH ECHO & RTS

```



Programming  
Languages

*High Level*



Programming Languages



# High Level





Programming  
Languages

*High Level*



# *High Level*

**Nouns and Verbs**



# *High Level*

**Syntax**



# Programming Languages



# Programming Languages



Compiler







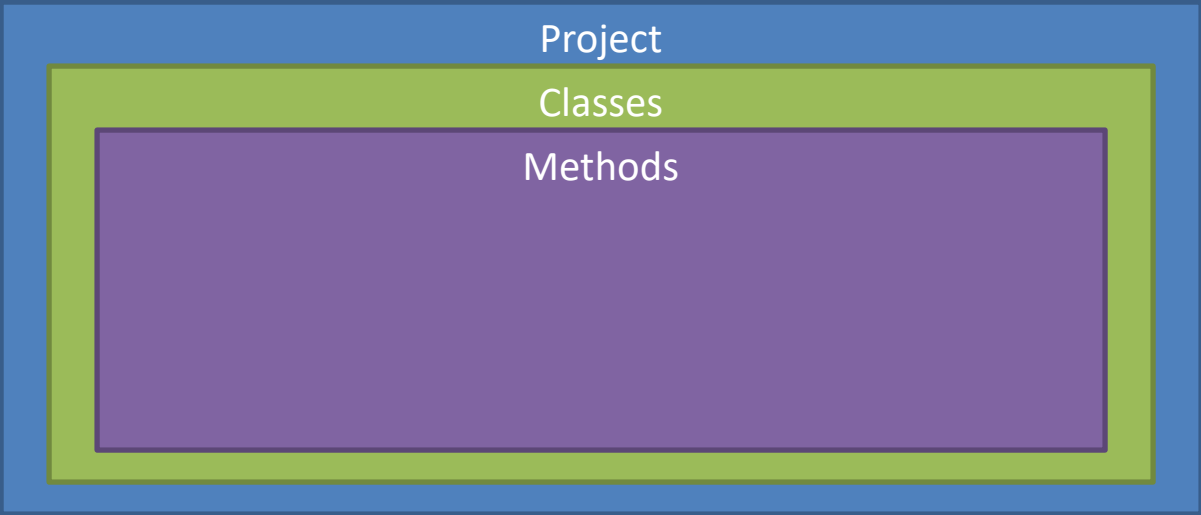
Project



Project

Classes

A diagram showing a blue rectangular frame containing a green rectangular area. The text "Project" is centered above the green area, and "Classes" is centered within the green area.





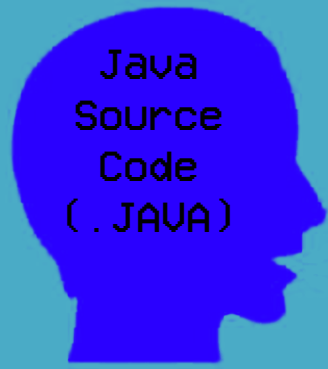
Classes

Methods

- Source Code in files with “.JAVA” extension
- The filename must MATCH the name of the class
- Everything is an “Object”



## Compilation



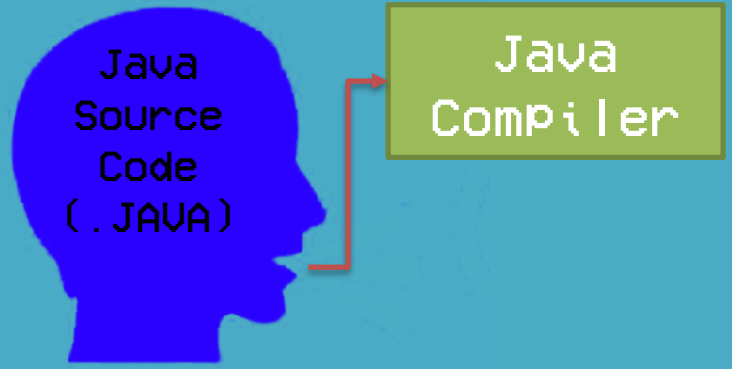
## Running







### Compilation

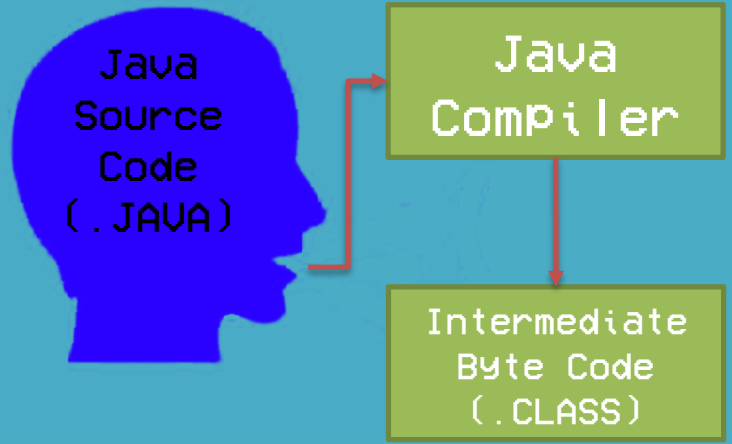


### Running





### Compilation

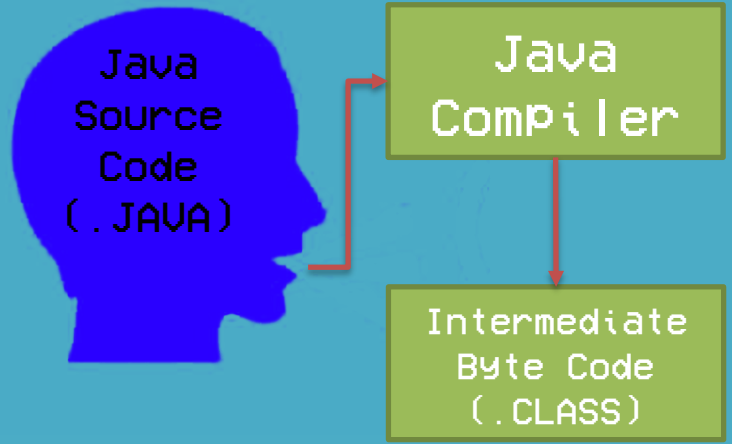


### Running





### Compilation



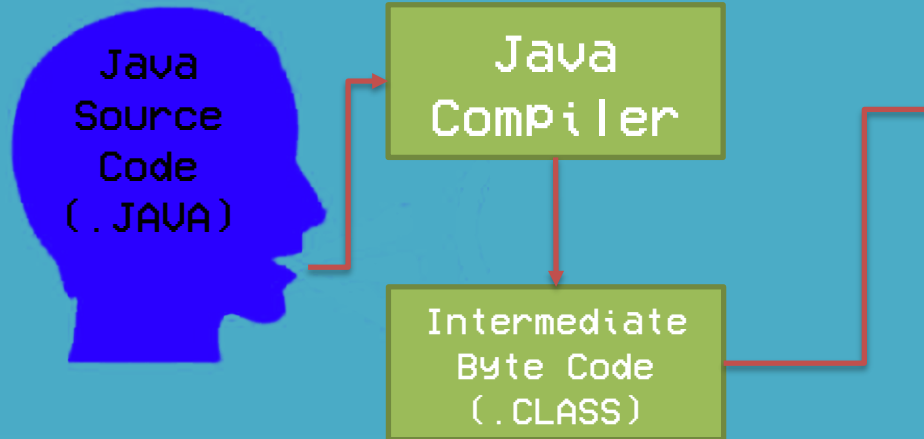
### Running

Java Virtual Machine (JVM)





### Compilation



### Running



## Other Terminology

- Running (Executing) – Is when the computer is following the instructions in a program
- Statement – An instruction to the computer. Most end with a semicolon “;”
  - int i;
  - double j;
- Syntax – The grammar rules for a programming language
- Comments – Code ignored by the compiler that is generally used to explain the code.
  - Single line comments use the “//”
  - Multiline comments use “/\* \*/”
- Arguments – Information found inside of parenthesis “()” that provide information for methods or other statements
  - if(<<ARGUMENT>>)
  - System.out.println(<<ARGUMENT>>)
- Bug – an error in a program
- Debugging – the process of removing errors
- There are 3 major classes of errors
  - Syntax
  - Runtime
  - Logic

# Types of Errors

- Syntax Errors– Grammatical mistakes in a program
  - Very common at first
  - These prevent a program from compiling and running
  - Common
    - Missing a Semicolon at the end of a statement
    - Using the wrong, misspelled, repeated, or incorrectly capitalized identifier
    - Mismatched curly braces “{}”, parenthesis “()”, single quotes “'””, double quotes “””, etc.

```
int i //Missing a semicolon
double j = 0.0;
j = 1.0; //Wrong identifier
System.out.println(i; //Missing parenthesis
```

# Types of Errors

- Runtime Errors – Errors detected when the program is running but not during compilation
  - The code will compile but crashes at some point
  - When this happens the computer detects the error and terminates the program
  - Common
    - Divide by 0
    - Calling a method from a NULL object
    - An index going outside the bounds of an Array

```
double j = 1.0 / 0.0; //Divide by 0
Scanner keyboard; //This has not been constructed so it is NULL
keyboard.hasNextLine(); //Calling method from NULL object
int[] a = {5,4,3,2,1}; //An array
a[5] = 2; //Index 5 is out of bounds
```

# Types of Errors

- Logic Errors – despite the program compiling and running, it produces incorrect results
  - Arguably the hardest to fix
  - Common:
    - Order of operations error
    - Round off mistakes
    - Incorrectly using types or methods

```
double f = 72.0; //72 degrees fahrenheit
double c = f - 32.0 * 5.0/9.0; //Order of operations error
double c2 = (f-32.0)*(5/9); // 5/9 = 0 so it will always be 0
double c3 = (f-32.0)*(5.0/9.0); //Correct
```



HELLO WORLD!