Introduction to Computers, Programs, and Java

CSE160: Computer Science A: Honors
Paul Fodor
Stony Brook University

http://www.cs.stonybrook.edu/~cse160
Contents

- How computers work (CPU, memory, hard disk, input and output)?
  - How Data is Stored and Processed? Number Systems.
  - How do we store text?
  - Memory: What goes in each memory segment?
- Programming Languages
  - Source Code
  - Compilation vs. Interpretation
- Operating Systems
- Java History and Basics
  - A Simple Java Program
  - Anatomy of Java Programs
- Programming Errors
What is a Computer?

- A computer consists of a CPU, memory, hard disk, monitor, printer, input and output devices.
Central Processing Unit (CPU)

• Central Processing Unit (CPU, processor):
  • retrieves instructions from memory and executes them
  • the CPU speed is measured in cycles per second = hertz (Hz)
    • 1 MegaHertz (MHz) = 1 million pulses per second
    • 1 GigaHertz (GHz) = 1 billion pulses per second
(Main) Memory

- Stores data and program instructions for CPU to execute
  - ordered sequence of bytes (i.e., 8 bits – a binary base unit)
How Data is Stored and Processed?

- In binary
- What’s binary?
  - the base-2 **number system**
- What do humans use?
  - base-10
  - Why? 10 fingers.
- Why do computers like binary?
  - electronics
    - easier to make hardware that stores and processes binary numbers than decimal numbers
  - more efficient: space & cost

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Memory content</th>
<th>Encoding for character ‘J’</th>
<th>Encoding for character ‘a’</th>
<th>Encoding for character ‘v’</th>
<th>Encoding for character ‘a’</th>
<th>Encoding for number 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>01001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>01100001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>01110110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>01100001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td>00000011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Number Systems

Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binary: 0, 1

Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Octal: 0, 1, 2, 3, 4, 5, 6, 7
Number Systems: Decimal

- The digits in the **decimal number system** are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
  - A decimal number is represented using a sequence of one or more of these digits.
    - The value that each digit in the sequence represents depends on its position.
    - A position in a sequence has a value that is an integral power of 10.
      - e.g., the digits 7, 4, 2, and 3 in decimal number 7423 represent 7000, 400, 20, and 3, respectively:
        \[
        \begin{array}{cccc}
        7 & 4 & 2 & 3 \\
        10^3 & 10^2 & 10^1 & 10^0 \\
        \end{array}
        \]
        \[= 7 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0\]
        \[= 7000 + 400 + 20 + 3 = 7423\]
- We say that 10 is the **base** or **radix** of the decimal number system.
  - The base of the binary number system is 2 since the binary number system has two digits: 0 and 1.
  - The base of the hex number system is 16 since the hex number system has sixteen digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
  - The base of the octal number system is 8 with digits: 0,1,2,3,4,5,6,7.
Binary

• Computers use binary numbers internally because storage devices like memory and disk are made to store 0s and 1s.
• Each 0 and 1 is called a bit (short for binary digit)
• Binary numbers are not intuitive to us, since we use decimal numbers in our daily life.
  • When you write a number like 20 in a program, it is assumed to be a decimal number.
• Internally, computer software is used to convert decimal numbers into binary numbers, and vice versa.
  • A number or a text (see character encodings later) inside a computer is stored as a sequence of 0s and 1s.
Binary Numbers => Decimals

Given a binary number \((b_nb_{n-1}b_{n-2}...b_2b_1b_0)_2\) the equivalent decimal value is

\[
b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + ... + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0
\]

Examples:

\((10)_2\) in binary is \(1 \times 2^1 + 0\) = \(2\) in decimal

\((1010)_2\) in binary is \(1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 0\) = \(10\) in decimal

\((10101011)_2\) in binary is \(1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1\) = \(171\) in decimal
Common Binary Powers

\[ 2^0 = 1 \]
\[ 2^1 = 2 \]
\[ 2^2 = 4 \]
\[ 2^3 = 8 \]
\[ 2^4 = 16 \]
\[ 2^5 = 32 \]
\[ 2^6 = 64 \]
\[ 2^7 = 128 \]
\[ 2^8 = 256 \]
\[ 2^9 = 512 \]
\[ 2^{10} = 1024 \]
Decimal => Binary

• To convert a decimal number \( d \) to a binary number is to find the binary digits \( (b_n, b_{n-1}, b_{n-2}, \ldots, b_2, b_1, b_0)_2 \) such that

\[
d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0
\]

• These numbers can be found by successively dividing \( d \) by 2 until the quotient is 0. The remainders are \( b_0, b_1, b_2, \ldots, b_{n-2}, b_{n-1}, b_n \)

For example, the decimal number 123 is \((1111011)_2\) in binary. The conversion is conducted as follows:

\[
\begin{array}{cccccccc}
0 & 1 & 3 & 7 & 15 & 30 & 61 & 123 \\
0 & 1 & 6 & 14 & 30 & 60 & 122 & \text{Quotient} \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & \text{Remainder} \\
b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
\end{array}
\]
Hexadecimal and Octal

• Binary numbers tend to be very long and cumbersome:
  • For example: \((11\ 1010\ 1011\ 1110)_2\)
• Hexadecimal and octal numbers are often used to abbreviate binary numbers:
  • For example: \((11\_1010\_1011\_1110)_2 = (3ABE)_H\)
    and \((11\_101\_010\_111\_110)_2 = (35276)_8\)
• The hexadecimal number system has 16 digits:
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
  • The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15.
  • Each hex digit corresponds to 4 bits (grouped from the end)
• The octal number system has 8 digits:
  0, 1, 2, 3, 4, 5, 6, and 7
  • Each octal digit corresponds to 3 bits (grouped from the end)
Hexadeimals <-> Binary

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

To convert a hexadecimal number to a binary number, simply convert each digit in the hexadecimal number into a four-digit binary number. For example,

\[(38D)_H = (11\_1000\_1101)_2\]

To convert a binary number to a hexadecimal, convert every four binary digits from right to left in the binary number into a hexadecimal number. For example,

\[\begin{align*}
&\underline{1} \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \\
&\downarrow \quad \downarrow \quad \downarrow \\
&(3 \ 8 \ D)\ H
\end{align*}\]
Octal <=> Binary

To convert an octal number to a binary number, simply convert each digit in the octal number into a three-digit binary number. For example,

\[(1615)_8 = (1_110_001_101)_2\]

To convert a binary number to an octal number, convert every three binary digits from right to left in the binary number into an octal digit. For example,

\[(1110001101)_2 \downarrow \downarrow \downarrow \downarrow \downarrow = (1615)_8\]
Hexadecimals => Decimals

• Given a hexadecimal number \((h_nh_{n-1}h_{n-2}...h_2h_1h_0)_H\)

The equivalent decimal value is

\[h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + ... + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0\]

\((7F)_H\) in hex is \(7 \times 16^1 + 15 = 127\) in decimal

\((FFFF)_H\) in hex \(15 \times 16^3 + 15 \times 16^2 + 15 \times 16 + 15 = 65535\) in decimal

• Octal number system is similar, but base is 8.

\((12)_8\) in octal is \(1 \times 8^1 + 2 = 10\) in decimal
Decimals => Hexadecimals

To convert a decimal number $d$ to a hexadecimal number is to find the hexadecimal digits $(h_n, h_{n-1}, h_{n-2},..., h_2, h_1, h_0)_H$ such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + ... + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

These numbers can be found by successively dividing $d$ by 16 until the quotient is 0. The remainders are $h_0, h_1, h_2,\ldots, h_{n-2}, h_{n-1}, h_n$

For example, the decimal number 123 is $(7B)_H$ in hexadecimal. The conversion is conducted as follows:

- Octal number system is similar, but *base* is 8, so we divide by 8 instead of 16.
Windows Calculator

The Windows Calculator is a useful tool for performing number conversions. To run it, choose *Programs, Accessories, and Calculator* from the Start button, and switch to Programmer View:
So Hardware stores 0s & 1s

• **How do we store text?**
  • Numerically (i.e., using numeric codes)
    • Each character is stored in memory as a number
  • **Standard character encoding sets**: ASCII, Unicode
    • ASCII uses 1 byte per character (128 chars)
      • For example: ‘A’ is 65
    • Unicode: ~65K different characters
      • Multiple encodings (UTF-8, UTF-16, UTF-32, …)
        o short encodings use the first bit for continuation
          (variable length encodings) and may be more
efficient for communication (shorter encoding)
## ASCII Table

| Dec | Hx | Oct | Char       | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|-----|----|-----|-----------|-----|----|-----|------|-----|-----|----|-----|------|-----|-----|----|-----|------|-----|-----|----|-----|------|-----|-----|
| 0   | 0  | 000 | NUL (null)| 32  | 20 | 040 | #32 | Space| 64  | 40 | 100 | #64 | @   | 96 | 60 | 140 | #96 | \   |
| 1   | 1  | 001 | SOH (start of heading)| 33  | 21 | 041 | #33 | !    | 65  | 41 | 101 | #65 | A   | 97 | 61 | 141 | #97 | a   |
| 2   | 2  | 002 | STX (start of text) | 34  | 22 | 042 | #34 | "    | 66  | 42 | 102 | #66 | B   | 98 | 62 | 142 | #98 | b   |
| 3   | 3  | 003 | ETX (end of text)  | 35  | 23 | 043 | #35 | #    | 67  | 43 | 103 | #67 | C   | 99 | 63 | 143 | #99 | c   |
| 4   | 4  | 004 | EOT (end of transmission) | 36  | 24 | 044 | #36 | $    | 68  | 44 | 104 | #68 | D   | 100| 64 | 144 | #100| d   |
| 5   | 5  | 005 | ENQ (enquiry)     | 37  | 25 | 045 | #37 | %    | 69  | 45 | 105 | #69 | E   | 101| 65 | 145 | #101| e   |
| 6   | 6  | 006 | ACK (acknowledge) | 38  | 26 | 046 | #38 | &    | 70  | 46 | 106 | #70 | F   | 102| 66 | 146 | #102| f   |
| 7   | 7  | 007 | BEL (bell)       | 39  | 27 | 047 | #39 | ;    | 71  | 47 | 107 | #71 | G   | 103| 67 | 147 | #103| g   |
| 8   | 8  | 010 | BS (backspace)   | 40  | 28 | 050 | #40 | (    | 72  | 48 | 110 | #72 | H   | 104| 68 | 150 | #104| h   |
| 9   | 9  | 011 | TAB (horizontal tab) | 41  | 29 | 051 | #41 | )    | 73  | 49 | 111 | #73 | I   | 105| 69 | 151 | #105| i   |
| 10  | A  | 012 | LF (NL line feed, new line) | 42  | 30 | 052 | #42 | *    | 74  | 4A | 112 | #74 | J   | 106| 6A | 152 | #106| j   |
| 11  | B  | 013 | VT (vertical tab) | 43  | 31 | 053 | #43 | +    | 75  | 4B | 113 | #75 | K   | 107| 6B | 153 | #107| k   |
| 12  | C  | 014 | FF (NP form feed, new page) | 44  | 32 | 054 | #44 | ,    | 76  | 4C | 114 | #76 | L   | 108| 6C | 154 | #108| l   |
| 13  | D  | 015 | CR (carriage return) | 45  | 33 | 055 | #45 | -    | 77  | 4D | 115 | #77 | M   | 109| 6D | 155 | #109| m   |
| 14  | E  | 016 | SO (shift out)   | 46  | 34 | 056 | #46 | \   | 78  | 4E | 116 | #78 | N   | 110| 6E | 156 | #110| n   |
| 15  | F  | 017 | SI (shift in)    | 47  | 35 | 057 | #47 | 0   | 79  | 4F | 117 | #79 | O   | 111| 6F | 157 | #111| o   |
| 16  | 10 | 020 | DLE (data link escape) | 48  | 36 | 060 | #48 | 0    | 80  | 50 | 120 | #80 | P   | 112| 70 | 160 | #112| p   |
| 17  | 11 | 021 | DC1 (device control 1) | 49  | 37 | 061 | #49 | 1    | 81  | 51 | 121 | #81 | Q   | 113| 71 | 161 | #113| q   |
| 18  | 12 | 022 | DC2 (device control 2) | 50  | 38 | 062 | #50 | 2    | 82  | 52 | 122 | #82 | R   | 114| 72 | 162 | #114| r   |
| 19  | 13 | 023 | DC3 (device control 3) | 51  | 39 | 063 | #51 | 3    | 83  | 53 | 123 | #83 | S   | 115| 73 | 163 | #115| s   |
| 20  | 14 | 024 | DC4 (device control 4) | 52  | 40 | 064 | #52 | 4    | 84  | 54 | 124 | #84 | T   | 116| 74 | 164 | #116| t   |
| 21  | 15 | 025 | NAK (negative acknowledge) | 53  | 41 | 065 | #53 | 5    | 85  | 55 | 125 | #85 | U   | 117| 75 | 165 | #117| u   |
| 22  | 16 | 026 | SYN (synchronous idle) | 54  | 42 | 066 | #54 | 6    | 86  | 56 | 126 | #86 | V   | 118| 76 | 166 | #118| v   |
| 23  | 17 | 027 | ETB (end of trans. block) | 55  | 43 | 067 | #55 | 7    | 87  | 57 | 127 | #87 | W   | 119| 77 | 167 | #119| w   |
| 24  | 18 | 030 | CAN (cancel)     | 56  | 44 | 070 | #56 | 8    | 88  | 58 | 130 | #88 | X   | 120| 78 | 170 | #120| x   |
| 25  | 19 | 031 | EM (end of medium) | 57  | 45 | 071 | #57 | 9    | 89  | 59 | 131 | #89 | Y   | 121| 79 | 171 | #121| y   |
| 26  | 1A | 032 | SUB (substitute) | 58  | 46 | 072 | #58 | 0    | 90  | 5A | 132 | #90 | Z   | 122| 7A | 172 | #122| z   |
| 27  | 1B | 033 | ESC (escape)     | 59  | 47 | 073 | #59 | ;    | 91  | 5B | 133 | #91 | [   | 123| 7B | 173 | #123| {   |
| 28  | 1C | 034 | FS (file separator) | 60  | 48 | 074 | #60 | <    | 92  | 5C | 134 | #92 | \   | 124| 7C | 174 | #124| }   |
| 29  | 1D | 035 | GS (group separator) | 61  | 49 | 075 | #61 | =    | 93  | 5D | 135 | #93 | I   | 125| 7D | 175 | #125| (   |
| 30  | 1E | 036 | RS (record separator) | 62  | 50 | 076 | #62 | >    | 94  | 5E | 136 | #94 | ~   | 126| 7E | 176 | #126| ~   |
| 31  | 1F | 037 | US (unit separator) | 63  | 51 | 077 | #63 | ?    | 95  | 5F | 137 | #95 | ]   | 127| 7F | 177 | #127| DEL |

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Memory: What goes in each memory segment?

- **Stack Segment**
  - variables declared inside methods (called local)
    - removed from memory when the method returns

- **Heap Segment**
  - for dynamic data (whenever you use the operator `new`)
    - i.e., the data for constructed objects
      - persistent as long as an existing a variable references this region of memory (because garbage collection)

- **Global Segment**
  - data that can be reserved at compile time
    - the program
    - global data (like `static` data)
    - constants (e.g., interned `String`)
A machine language is a set of instructions executed directly by a computer's central processing unit (CPU).

At the beginning there was only machine language: a sequence of bits that directly controls a processor, causing it to add, compare, move data from one place to another.

Example: GCD program in x86 machine language (represented in hexadecimal here):

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00
00 00 39 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```
Assembly languages were invented to allow operations to be expressed with **mnemonic** abbreviations.

For example, to add two numbers, you might write an instruction in assembly code like this:

```
ADDF3 R1, R2, R3
```

A program called **assembler** is used to convert assembly language programs into machine language.
Machine Language    Assembly Language    High-Level Language

• Example: the GCD program in x86 assembly:

```
pushl  %ebp
movl  %esp, %ebp
pushl %ebx
subl  $4, %esp
andl  $-16, %esp
call  getint
movl  %eax, %ebx
call  getint
cmpl  %eax, %ebx
je    C
A:     cmpl  %eax, %ebx
       jle   D
       subl  %eax, %ebx
B:     cmpl  %eax, %ebx
       jne  A
C:     novl  %ebx, (%esp)
call  putint
       novl  -4(%ebp), %ebx
leave
ret
D:     subl  %ebx, %eax
       jmp   B
```
Programming Languages

Machine Language  Assembly Language  High-Level Language

**Assembly** was easier to use than binary machine language, **BUT:** not very user friendly, very low-level operations, machine language dependent, programming is very time consuming.

*High Level programming Languages:* languages with strong abstraction from the details of the computer: methods, classes, etc.

– platform independent
– can use previously-developed libraries
– more user friendly, easy to use
Popular High-Level Languages

• FORTRAN (FORmula TRANslation)
• LISP
• COBOL (COnmon Business Oriented Language)
• BASIC (Beginner All-purpose Symbolic Instructional Code)
• Pascal (named for Blaise Pascal)
• Ada (named for Ada Lovelace)
• C (whose developer designed B first)
• Visual Basic (Basic-like visual language developed by Microsoft)
• C++ (an object-oriented language, based on C)
• Java
• C# (a Java-like language developed by Microsoft)
• python
• …
Source Code

- You write programs in text, they are called *source code*
- *Source code,* or simply *code* or *source,* is a program, with or without comments, written by a human in plain text (i.e., human readable alphanumeric characters).

**Brief history (of source code):**
- The earliest programs for stored-program computers were entered in binary through the front **panel switches** of the computer.
- Punched cards replaced binary switches.
- IBM started distributing high-level programming languages source code printed or stored in persistent memory.

**Legal aspects:**
- In 1974, the US Commission on New Technological Uses of Copyrighted Works (CONTU) decided that "computer programs, to the extent that they embody an author's original creation, are proper subject matter of copyright".
Source Code

How does a program run?

- Compilers and interpreters.

What’s a compiler?

- A software program that translates the high-level source program into an equivalent target program (typically in machine language), and then goes away:
Interpretation

• Pure Interpretation

• Interpreter stays around for the execution of the program

  • Interpreter is the locus of control during execution

[Diagram of source program, input, interpreter, output]
Why Java?

Java is somewhat different from previous languages.
Java started a principle, "write once, run anywhere".
What does that mean?
- **Platform (and operator system) independence** for compiled Java code

How?
- The Java Virtual Machine (JVM)
  - Java programs are compiled into Java bytecode
  - Bytecode is then executed by the JVM on any OS and any platform
Compilation and Interpretation

- Most modern language implementations (starting with Java) include a mixture of both compilation and interpretation
- Compilation followed by interpretation:

```
Source program
    ↓
  Translator
    ↓
Intermediate program
    → Virtual machine
        Input
        Output
```
Java Virtual Machine

- A program that runs Java programs and manages memory for Java programs.
- Why?
  - Each platform is different (Mac, PC, Linux, Android, etc.)

- The Java Development Kit (JDK) is a distribution of Java Technology, i.e., the Java Application Programming Interface (API), the Java compiler and the Java Virtual Machine, to compile and execute Java programs.
Java Development Kit (JDK)

- JDK 1.02 (1995)
- J2SE 5.0 (2004) (Major Refactoring)
- Java SE 8 (2014) (Major Refactoring: Language-level support for lambda expressions)
  - Long Term Support (LTS)
  ...
- Java SE 21 (LTS, 2023)
- Java SE 22, 23 (2024)
The operating system (OS) is a program that manages and controls a computer’s activities.

Windows
Unix
Linux
Mac OsX
Android
A Simple Java Program

// Welcome.java
// This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
Creating, Compiling, and Running Programs

Source Code
Create/Modify Source Code
i.e., javac Welcome.java

Bytecode
Run Bytecode
i.e., java Welcome

Result
If compilation errors
If runtime errors or incorrect result

Source code (developed by the programmer)

```java
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Byte code (generated by the compiler for JVM to read and interpret, not for you to understand)

```plaintext
... Method Welcome()
    0 aload_0
    ...
Method void main(java.lang.String[])
    0 getstatic #2 ...
    3 ldc #3 <String "Welcome to Java!">
    5 invokevirtual #4 ...
    8 return
...`
Running Programs in Eclipse
Running Programs from command line

pfodror@sparky ~$ emacs Welcome.java

```java
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

pfodror@sparky ~$ javac Welcome.java

pfodror@sparky ~$ java Welcome
Welcome to Java!
• Set the path to JDK bin directory, so that the console knows where \texttt{javac} and \texttt{java} are:
  \begin{verbatim}
  set PATH=$PATH;c:\Java\jdk1.8.0\bin
  \end{verbatim}

• Set Classpath to include the current directory, so that javac and java know where used libraries are:
  \begin{verbatim}
  set CLASSPATH=...
  \end{verbatim}
Trace a Program Execution

//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

Enter main method
Trace a Program Execution

//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

Execute statement
//This program prints Welcome to Java!
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

print a message to the console
Anatomy of Java Programs

- Comments
- Reserved words
- Modifiers
- Statements
- Blocks
- Classes
- Methods
- The main method
Comments

- Three types of comments in Java:
  - *Line comment*: A line comment is preceded by two slashes (//) in a line.
  - *Paragraph comment*: A paragraph comment is enclosed between /* and */ in one or multiple lines.
  - *javadoc comment*: javadoc comments begin with /** and end with */.
    - They are used for documenting classes, data, and methods.
    - They can be extracted into an HTML file using JDK's javadoc command.
public static int baseX2decimal(int base, String s) {  
    int dec = 0;  
    for (int i = 0; i < s.length(); i++) {  
        char c = s.charAt(i);  
        // extract the decimal digit from the character 0..9 or A..Z for 10,11,...
        int e = ('0' <= c && c <= '9')  
                 ? c - '0'  
                 : ('a' <= c && c <= 'z')  
                     ? c - 'a' + 10  
                     : c - 'A' + 10;  
        dec = dec * base + e;  
    }  
    return dec;  
}
Reserved Words (Keywords)

- **Reserved words** or **keywords** are words that have a **specific meaning to the compiler**
- Cannot be used for other purposes in the program
- Example: **class**
  - the word after **class** is the name for the class
Java Keywords

abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, false, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, true, try, void, volatile, while

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html
Modifiers

• Java uses certain reserved words called **modifiers** that specify the **properties** of the data, methods, and classes and how they can be used

• Examples: **public, static, private, final, abstract, protected**
  
  • A **public** datum, method, or class can be accessed by other programs
  
  • A **private** datum or method cannot be accessed by other programs
Statements

• A statement represents an action or a sequence of actions.
• Every statement in Java ends with a semicolon (;)
• Examples:
  
  `System.out.println("Welcome to Java!");`
  is a statement to display the greeting "Welcome to Java!" followed by a new line.

  `System.out.print("Welcome to Java!");`
  is a statement to display the greeting "Welcome to Java!" without moving to the new line.
Statements

- Printing is overloaded for all types in Java:
  
  ```java
  System.out.println(1);
  System.out.println(1.2);
  System.out.println(true);
  ```

- Java is weakly typed:
  
  ```java
  System.out.print("result is "+123);
  ```

  is a statement to displays "result is 123"
Blocks

A pair of braces in a program forms a block that groups components of a program.

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```
Block Styles

- We use end-of-line style for braces:

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-line style

Next-line style

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```
Programming Errors

• Syntax Errors
  • Detected by the compiler

• Runtime Errors
  • Causes the program to abort

• Logic Errors
  • Produces incorrect result (may or may not run into a runtime error)
Syntax Errors

public class ShowSyntaxError {
    public static void main(String[] args) {
        i = 30; // Detected by the compiler
        System.out.println(i + 4);
    }
}

The program does not compile.
public class ShowRuntimeError {
    public static void main(String[] args) {
        int i = 1 / 0;
        // Runtime error: Division with 0
    }
}

The program compiles (because it is syntactically correct), but it crashes at runtime.
public class ShowLogicError {

    // Determine if a number is between 1 and 100 inclusively
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number = input.nextInt();
        // Display the result
        System.out.println("The number is between 1 and 100, \textit{inclusively}: "+
            ((1 < number) && (number < 100)));
        // Wrong result if the entered number is 1 or 100
        System.exit(0);
    }
}

The program compiles and may run without a crash, but the results are incorrect.
Logic Errors  **Debugging**

- Logic errors are also called **bugs**
- The process of finding and correcting errors is called **debugging**

Methods of debugging:

- hand-trace the program (i.e., catch errors by reading the program),
- insert print statements in order to show the values of the variables
- for a large, complex program, the most effective approach for debugging is to use a **debugger utility**
Debugger

Debugger is a program that facilitates debugging. You can use a debugger to:

- **Step 1**: Set breakpoints where the execution pauses when we are debugging.
- **Step 2**: Start the debugger and execute a single statement at a time from the first encountered breakpoint.
  - Trace into or stepping over a method.
  - Display variables.
public void refreshDisplay(String option) {
    System.out.println("Option: "+option);
    if (option.equals("b41") || option.equals("b42") || option.equals("b43") || option.equals("b44") || option.equals("b51") || option.equals("b52")){
        bet = 1;
        141.setText("Round "+rounds+" "+Action);
        if (option.equals("b41") ){
            bet = 1;
        } else if (option.equals("b42") ){
            bet = 2;
        } else if (option.equals("b43") ){
            bet = 3;
        } else if (option.equals("b44") ){
            bet = 4;
        } else bet = 5;
        142.setText("Bet: "+bet);
        b41.disable();
        b42.disable();
        b51.enable();
        b62.disable();
    } else if(option.equals("b52")){
        // implement second step of baccarat
    }
}