Introduction to Computers, Programs, and Java

CSE 114, Computer Science 1
Stony Brook University

http://www.cs.stonybrook.edu/~cse114
What is a Computer?

A computer consists of a CPU, memory, hard disk, monitor, printer, input and communication devices.
CPU

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

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

• What’s binary?
  • a base-2 number system

• What do humans use?
  • base-10

• Why?

• 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>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>01001010</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>01100001</td>
<td>Encoding for character ‘a’</td>
</tr>
<tr>
<td>2002</td>
<td>01110110</td>
<td>Encoding for character ‘v’</td>
</tr>
<tr>
<td>2003</td>
<td>01100001</td>
<td>Encoding for character ‘a’</td>
</tr>
<tr>
<td>2004</td>
<td>00000011</td>
<td>Encoding for number 3</td>
</tr>
</tbody>
</table>
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.
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

- 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)
  - A number or a text inside a computer is stored as a sequence of 0s and 1s.
- Binary numbers are not intuitive, 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.
Binary numbers tend to be very long and cumbersome:
  - For example: $(1010 1010 1010)_2$

Hexadecimal and octal numbers are often used to abbreviate binary numbers:
  - For example: $(1010 1010 1010)_2 = (AAA)_H$
    and $(101 010 101 010)_2 = (5252)_8$

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

The octal number system has 8 digits:
  0, 1, 2, 3, 4, 5, 6, and 7
  - Each octal digit corresponds to 3 bits
Binary Numbers => Decimals

Given a binary number \((b_n b_{n-1} b_{n-2} \ldots b_2 b_1 b_0)_2\)
the equivalent decimal value is

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

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

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

\((10101011)_2\) in binary \(1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = 171\) in decimal
Decimals => 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:

\[
\begin{array}{cccccccccc}
& & & & & & 0 & \downarrow 61 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
2 & \overline{123} & \div 2 & = & 61 & \div 2 & = & 30 & \div 2 & = & 15 & \div 2 & = & 7 & \div 2 & = & 3 & \div 2 & = & 1 \\
& & & & & & 0 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 61 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
\hline
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
& & & & & & 1 & \downarrow 122 & \downarrow 60 & \downarrow 30 & \downarrow 15 & \downarrow 7 & \downarrow 3 & \downarrow 1 & \downarrow 0 \\
\end{array}
\]

\(b_6, b_5, b_4, b_3, b_2, b_1, b_0\)
Hexadecimals <=> 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,

\[(38D)_H = (1110001101)_2\]

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,

\[(1110001101)_2 \rightarrow (38D)_H\]
Hexadecimals => Decimals

- The hexadecimal number system has sixteen 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.
- Given a hexadecimal number \((h_n h_{n-1} h_{n-2} ... h_2 h_1 h_0)_H\)
  The equivalent decimal value is
  \[
h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + ... + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0
  \]

- \((7F)\text{H}\) in hex is \(7 \times 16^1 + 15 = 127\) in decimal
- \((FFFF)\text{H}\) 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.
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 $h_0, h_1, h_2, ..., h_{n-2}, h_{n-1}, h_n$

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.
Octal <=> Binary

<table>
<thead>
<tr>
<th>Binary</th>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

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,

\[
\begin{array}{c}
\text{(1 1 1 0 0 0 1 1 0 1)}_2 \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\text{(1 6 1 5)}_8 \\
\end{array}
\]
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:
Memory: What goes in each memory segment?

- **Stack Segment**
  - temporary variables declared inside methods
  - removed from memory when a method returns

- **Heap Segment**
  - for dynamic data (whenever you use new)
  - data for constructed objects
  - persistent as long as an existing object variable references this region of memory

- **Global Segment**
  - data that can be reserved at compile time
  - global data (like static data)
So Hardware stores 0s & 1s

• **How do we store text?**
  • Numerically (using a numeric code)
    • Each character is stored in memory as a number
  • Standard character sets: ASCII & Unicode
    • ASCII uses 1 byte per character
      • For example: ‘A’ is 65
### ASCII Table

[Link to ASCII Table Image](http://enteos2.area.trieste.it/russo/IntroInfo2001-2002/CorsoRetiGomezelo/ASCII_EBIC_files/ascii_table.jpg)

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>HTML</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0</td>
<td>NUL</td>
<td>#32</td>
<td>Space</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>0</td>
<td>SOH</td>
<td>#33</td>
<td>!</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0</td>
<td>STX</td>
<td>#34</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>0</td>
<td>ETX</td>
<td>#35</td>
<td>#</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>0</td>
<td>EOT</td>
<td>#36</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>0</td>
<td>ENQ</td>
<td>#37</td>
<td>%</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>0</td>
<td>ACK</td>
<td>#38</td>
<td>&amp;</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>0</td>
<td>BEL</td>
<td>#39</td>
<td>'</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>0</td>
<td>BS</td>
<td>#40</td>
<td>(</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>0</td>
<td>TAB</td>
<td>#41</td>
<td>)</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>0</td>
<td>LF</td>
<td>#42</td>
<td>*</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>0</td>
<td>VT</td>
<td>#43</td>
<td>+</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>0</td>
<td>FF</td>
<td>#44</td>
<td>,</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>0</td>
<td>CR</td>
<td>#45</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>0</td>
<td>SO</td>
<td>#46</td>
<td>.</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>0</td>
<td>SI</td>
<td>#47</td>
<td>/</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>2</td>
<td>DLE</td>
<td>#48</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>2</td>
<td>DC1</td>
<td>#49</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>2</td>
<td>DC2</td>
<td>#50</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>2</td>
<td>DC3</td>
<td>#51</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>2</td>
<td>DC4</td>
<td>#52</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>2</td>
<td>NAK</td>
<td>#53</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>2</td>
<td>SYN</td>
<td>#54</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>2</td>
<td>ETB</td>
<td>#55</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>0</td>
<td>CAN</td>
<td>#56</td>
<td>8</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>0</td>
<td>EM</td>
<td>#57</td>
<td>9</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>0</td>
<td>SUB</td>
<td>#58</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>0</td>
<td>ESC</td>
<td>#59</td>
<td>;</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>0</td>
<td>FS</td>
<td>#60</td>
<td>&lt;</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>0</td>
<td>GS</td>
<td>#61</td>
<td>=</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>0</td>
<td>RS</td>
<td>#62</td>
<td>&gt;</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>0</td>
<td>US</td>
<td>#63</td>
<td>?</td>
</tr>
</tbody>
</table>

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
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:

```
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.

A program called assembler is used to convert assembly language programs into machine language.

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

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

...
• Example: 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: movl %ebx, (%esp)
call putint
movl -4(%ebp), %ebx
leave
ret
D: subl %ebx, %eax
jmp B
```
Programming Languages

Machine Language    Assembly Language    High-Level Language

**Assembly:** Far 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.
– more user friendly, easy to use
– more flexible
– platform independent
Popular High-Level Languages

- FORTRAN (FORmula TRANslation)
- LISP
- COBOL (CCommon 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)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (an object-oriented language, based on C)
- Java
- C# (a Java-like language developed by Microsoft)
- python
Compiling Source Code

What’s a compiler?

- A software program
- Input: High Level Language source code
- Output: Machine Language or Assembly Code
- It is typically integrated with an assembly
- together they can make an executable or binary program
The **operating system** (OS) is a program that manages and controls a computer’s activities.

- Windows
- Mac OsX
- Android
- Linux
Why Java?

Java is somewhat different from older languages
Java started a principle, "**write once, run anywhere**"
What does that mean?
   Platform independence for compiled Java code
How?
   The Java Virtual Machine
Java programs are compiled into Java bytecode
   Bytecode is then executed by the
   **Java Virtual Machine (JVM)**
• 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.)
JDK Versions

- JDK 1.02 (1995)
- JDK 1.1 (1996)
- J2SE 1.2 (1998)
- J2SE 1.3 (2000)
- J2SE 5.0 (2004)
- Java SE 6 (2006)
- Java SE 7 (2011)
- Java SE 8 (2014) Long Term Support (LTS)
- Java SE 9 (2017)
- Java SE 10, 11 (LTS) (2018)
- Java SE 12, 13 (2019)
- Java SE 14 (March 2020)
JDK Editions

- **Java Standard Edition (J2SE)**
  - J2SE can be used to develop client-side standalone applications or applets.

- **Java Enterprise Edition (J2EE)**
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.

- **Java Micro Edition (J2ME)**
  - J2ME was used to develop applications for mobile devices such as cell phones.

Our textbook uses J2SE to introduce Java programming.
// 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

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

Saved on the disk
stored on the disk

Source code (developed by the programmer)

```java
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)
Running Programs from command line

pfodor@sparky ~$ emacs Welcome.java

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

pfodor@sparky ~$ javac Welcome.java

pfodor@sparky ~$ java Welcome
Welcome to Java!
Compiling and Running Java from the Command Window

- Set path to JDK bin directory
  ```
  set PATH=c:\Java\jdk1.8.0\bin
  ```
- Set classpath to include the current directory
  ```
  set CLASSPATH=.
  ```
- Compile your source code:
  ```
  javac Welcome.java
  ```
- Run your bytecode:
  ```
  java Welcome
  ```
Running Programs in Eclipse
//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 a Java Program

- 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.
Comments

- The code that explains itself let it be (no need to comment). Just use good meaningful names for your identifiers (variables, methods).

Good programmers can always figure out what something is done from the code. But it is much more difficult to figure out why or how it was done.

```java
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.

```java
System.out.println("Welcome to Java!");
```

is a statement to display the greeting "Welcome to Java!"

• Every statement in Java ends with a semicolon (;)
A pair of braces in a program forms a block that groups components of a program.
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**

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

**Next-line style**

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

- What’s an API?
  - Application Programming Interface
  - a library of code / names to use

- What are Names / Identifiers used for?
  - For Variables, Classes, and Methods
  - From 2 sources:
    - the Oracle/Sun (or someone else’s) API
    - your own classes, variables, and methods

- Your Identifiers (Names) – Why name them?
  - they are your data and commands, and you’ll need to reference them elsewhere in your program

```java
int myVariable = 5; // Declaration
myVariable = myVariable + 1; // Using the variable
```
Rules for Identifiers

• Should contain only letters, numbers, & '_'
  • '$' is allowed, but only for special use
• Cannot begin with a digit!
• Although it is legal, do not begin with ‘_’ (underscore)
• Uppercase and lowercase letters are considered to be different characters (Java is case-sensitive)
• Examples:
  • Legal: myVariable, my_class, my4Var
  • Illegal: 4myVariable, my class, my!Var, @$myClass
Common Java Naming Conventions

- Variables & Methods start with lower case letters: `radius`, `getRadius`
- Classes start with upper case letters: `Circle`
- Variables and Class identifiers should generally be nouns: `radius`, `Circle`
- Method identifiers should be verbs: `getRadius`
- Use Camel notation: `GeometricObject`, `getRadius`
- Use descriptive names: `Circle`, `radius`, `area`

```
area = PI * radius * radius;
```
Programming Errors

• Syntax Errors
  • Detected by the compiler
• Runtime Errors
  • Causes the program to abort
• Logic Errors
  • Produces incorrect result
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.
Runtime Error

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.
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:

- Set breakpoints where the execution pauses when we are debugging.
- Execute a single statement at a time.
- 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")
        rounds++;
    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();
    b6.disable();
} else if(option.equals("b52")) {
    // implement second step of baccarat