Introduction to Programming

with Java, for Beginners

++ & -- operator
Switch statement
Passing arguments to Main
Number Representation & Storage

The increment operator

- ++ adds 1 to a variable
  - It can be used as a statement by itself, or within an expression
  - It can be put before or after a variable
  - If before a variable (pre-increment), it means to add one to the variable, then use the result
  - If put after a variable (post-increment), it means to use the current value of the variable, then add one to the variable

Examples of ++

```java
int a = 5;
> a++
5
> a // a is now 6
6
int b = 5;
> ++b
6
> b
6
```

The decrement operator

- -- subtracts 1 from a variable
  - Used similarly as ++ operator

```java
int e = 5;
int f = e++;
// e is 6, f is 5
int x = 10;
int y = 100;
int z = ++x + y++;
// x is 11, y is 101, z is 111
int c = 5;
int d = --c;
// c is 4
int x = 10;
int y = 100;
int z = --x + y--;
// x is 9, y is 99, z is 109
```
Syntax of the *switch* statement

- The syntax is:
  ```java
  switch (expression) {
  case value1 :
    statements ;
    break ;
  case value2 :
    statements ;
    break ;
  ...(more cases)...
  default :
    statements ;
    break ;
  }
  ```

- The *expression* must yield an integer or a character
- Each *value* must be a literal integer or character
- Notice that colons (:) are used as well as semicolons
- The last statement in every case should be a `break`;
  - If missing, fall through the next case
- The *default* case handles every value not otherwise handled

Example switch statement

```java
switch (cardValue) {
  case 1:
    System.out.print("Ace");
    break;
  case 11:
    System.out.print("Jack");
    break;
  case 12:
    System.out.print("Queen");
    break;
  case 13:
    System.out.print("King");
    break;
  default:
    System.out.print(cardValue);
    break;
}
```

Main

```java
public static void main (String [] args)
{
  Must have the exact signature
  - Only variation allowed is name of the input parameter
  - So main starts everything, how do we call main and provide inputs?

  To run a program recall
  - Command: `java ClassName`
    - This what calls the main method if the class has one
  - So we could pass arguments as follows:
    `java ClassName list-of-arguments`
}
```

Main Args Example

```java
public class ExampleArgs{
  public static void main(String [] args){
    System.out.println("Demo for Inputs args");
    for(int i = 0; i < args.length; i++){
      System.out.println(args[i]);
    }
  }
}
```

```java
> java ExampleArgs ESE 112
Demo for Inputs args
ESE 112
```

Note: Code works even if no arguments are passed to main() because JVM passes to main() a zero-length array of Strings and not a null
What does the Computer Understand?

- At the lowest level, a computer has electronic “plumbing”
- Operates by controlling the flow of electrons through very fast tiny electronic devices called transistors
- The devices react to presence or absence of voltage
- Could react actual voltages but designing electronics then becomes complex
- Symbolically we represent
  1. Presence of voltage as “1”
  2. Absence of voltage as “0”

What does the Computer process & store?

- An electronic device can represent uniquely only one of two things
  - Each “0” and Each “1” is referred to as a Binary Digit or Bit
  - Fundamental unit of information storage
- To represent more things we need more bits
  - E.g. 2 bits can represent four unique things: 00, 01, 10, 11
  - k bits can distinguish $2^k$ distinct items
- Combination binary bits together can represent some information or data.
  - E.g. 00101001 can be Decimal value 65
  - Interpretation of binary string is what distinguishes data

Unsigned Integers

- Like decimal numbers: “329”
  - “3” is worth 300, because of its position, while “9” is only worth 9

\[
\begin{align*}
\text{base-10 (decimal)} & \\
329 & = 3 \times 10^2 + 2 \times 10^1 + 9 \times 10^0 \\
\text{base-2 (binary)} & \\
101 & = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0
\end{align*}
\]

\[
3 \times 100 + 2 \times 10 + 9 \times 1 = 329
\]

\[
1 \times 4 + 0 \times 2 + 1 \times 1 = 5
\]

Unsigned Integers (cont.)

- An n-bit unsigned integer represents $2^n$ values
  - From 0 to $2^n-1$

<table>
<thead>
<tr>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
**Hexadecimal Notation**

- It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead.
- Fewer digits: four bits per hex digit.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Decimal</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>A</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>

**Hex to Decimal**

- $1A7 = 1 \times 16^2 + 10 \times 16^1 + 7 \times 16^0 = 423$ (base 10)

**char**

- The primitive type `char`.
- Just stored as numbers.
- Each char as a unique unsigned integer value (character code)
  - Based on Unicode standard.
  - Uses 16-bits to represent characters.
- You can use characters in arithmetic (they will automatically be converted to `int`).
  - `char ch = 'A';`
  - `ch + 1`
  - 66
  - `ch2 = 67`
  - C

**Two’s Complement Signed Integers**

- Range of an n-bit number: $-2^{n-1}$ through $2^{n-1} - 1$.
- Note: most negative number ($-2^{n-1}$) has no positive counterpart.

<table>
<thead>
<tr>
<th>$2^3$</th>
<th>2$^2$</th>
<th>2$^1$</th>
<th>2$^0$</th>
<th>$2^3$</th>
<th>2$^2$</th>
<th>2$^1$</th>
<th>2$^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
<td>1 0 0 0</td>
<td>-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>1</td>
<td>1 0 0 1</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>2</td>
<td>1 0 1 0</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>3</td>
<td>1 0 1 1</td>
<td>-5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>4</td>
<td>1 1 0 0</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>5</td>
<td>1 1 0 1</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>6</td>
<td>1 1 1 0</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>7</td>
<td>1 1 1 1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits to Bytes**

- Often storage is described in terms of bytes.
- 1 Byte = 8 bits.
- 16 bits = 2 Bytes (char in Java).
- 32 bits = 4 Bytes (int in Java).
- 64 bits = 8 Bytes (double in Java).
- 1024 bits = 1 Kilo-Byte (KB).
- 1024 x 1024 bits = 1 Mega-Byte (MB).
- 1024 x 1024 x 1024 = 1 Giga-Byte (GB).
More on Storage

- Stack: store temporary data within method or procedure
  - E.g. local variables within a method or variable declared in a for loop
    - After the method/block exits, the variables are no longer available
- Heap: remember data that can be accessed outside of methods and blocks
  - E.g. Objects are stored on heap
    - If you declare an array inside a method, you can access its contents even after you exit the method

Memory Management

- Memory is not infinite
- Stacks grow and shrink
- Heap
  - Grow when you dynamically allocate memory i.e. new Object()
  - If you do not manage the allocations then you will run out of this memory
    - Objects that will never be accessed or mutated again by application need to be reclaimed
      - This is known as Garbage Collection
- Some Languages like C/C++ leave it up to the programmer to do explicit memory management
- Java does automatic garbage collection
  - Done by JVM (Java Virtual Machine)