Lecture 5
Statements and variables

- [http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/03/statements+vars.html](http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/03/statements+vars.html)
The **building blocks** of a Java program

- [http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/03/intro-java2.html](http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/03/intro-java2.html)
## Numeric Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>$-2^7$ to $2^7-1$ ($-128$ to $127$)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>$-2^{15}$ to $2^{15}-1$ ($-32768$ to $32767$)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>$-2^{31}$ to $2^{31}-1$ ($-2147483648$ to $2147483647$)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>$-2^{63}$ to $2^{63}-1$</td>
<td>64-bit signed</td>
</tr>
<tr>
<td></td>
<td>(i.e., $-9223372036854775808$ to $9223372036854775807$)</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>Negative range: $-3.4028235E+38$ to $-1.4E-45$</td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: $1.4E-45$ to $3.4028235E+38$</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Negative range: $-1.7976931348623157E+308$ to $-4.9E-324$</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: $4.9E-324$ to $1.7976931348623157E+308$</td>
<td></td>
</tr>
</tbody>
</table>
Floating point numbers

• A floating point number is a number where the number of decimal digits *before* and *after* the decimal point is not fixed.
Floating point data types

• There are 2 kinds of floating point data types in Java:
  – `float` : Single precision floating numbers.

- uses 4 consecutive bytes of memory as a single 32 bit memory cell

- A single precision floating point variable can represent a floating point number:
  - in range of from $-10^{38}$ to $10^{38}$
  - and with about 7 decimal digits accuracy
- **double**: 8 consecutive bytes in memory.

- **Uses** 8 consecutive bytes of memory as a single 64 bit memory cell.

- A double precision floating point variable can represent a floating point number:
  - in range of from $-10^{308}$ to $10^{308}$
  - and with about 15 decimal digits accuracy.
## Integer data types

- Integer types in Java

<table>
<thead>
<tr>
<th>Type name (keyword)</th>
<th>Number of bytes</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1</td>
<td>−128 . . . 127</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>−32768 . . . 32767</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>−2,147,483,648 . . . 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>−9,223,372,036,854,775,808 . . . 9,223,372,036,854,775,807</td>
</tr>
</tbody>
</table>
public class Var02
{
    public static void main(String[] args)
    {
        double x;
        x = 0.31415e1;
        System.out.println("Hello Class");
        System.out.println("The variable x contains:");
        System.out.println(x);
    }
}
public class Var02
{
    public static void main(String[] args)
    {
        double x;
        x = 0.31415e1;
        System.out.println("Hello Class");
        System.out.println("The variable x contains..");
        System.out.println(x);
    }
}

public class Var02 {
    public static void main(String[] args) {
        double x;
        x = 0.31415E1;
        System.out.println("Hello Class");
        System.out.println("The variable x contains.. ");
        System.out.println(x);
    }
}
public class Demo {
    public static void main(String[] args) {
        // Define floating point variable with name "x"
        double x;
        // Assign 3.1415 to x
        x = 0.31415e1;
        System.out.println("Hello Class");
        System.out.println("The variable x contains this number:");
        // Print the content in variable "x"
        System.out.println(x);
    }
}
Numeric Literals

- Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a `double` type value. For example, `5.0` is considered a `double` value, not a `float` value. You can make a number a `float` by appending the letter `f` or `F`, and you can make a number a `double` by appending the letter `d` or `D`. For example, you can use `100.2f` or `100.2F` for a `float` number, and `100.2d` or `100.2D` for a `double` number.

- An integer literal is assumed to be of the `int` type. To denote an integer literal of the `long` type, append the letter `L` or `l` to it.
Scientific Notation

\[ a \times 10^b \]

- Normalized notation \( 1 \leq |a| < 10 \)
- e.g. 0.5 is written as \( 5 \times 10^{-1} \)
- 350 is written as \( 3.5 \times 10^2 \)

\[
\begin{array}{ll}
0.31415e1 & (= 0.31415 \times 10^1 = 3.1415) \\
31.415e-1 & (= 31.415 \times 10^{-1} = 3.1415)
\end{array}
\]
## Numeric Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
<td>34.0 − 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>
## Order of operations

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( ) [ ] -&gt; . ::</td>
<td>Grouping, scope, array/member access</td>
</tr>
<tr>
<td>2</td>
<td>! ~ - + * &amp; sizeof type cast ++x --x</td>
<td>(most) unary operations, sizeof and type casts</td>
</tr>
<tr>
<td>3</td>
<td>* / % MOD</td>
<td>Multiplication, division, modulo</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>Addition and subtraction</td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise shift left and right</td>
</tr>
<tr>
<td>6</td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Comparisons: less-than, ...</td>
</tr>
<tr>
<td>7</td>
<td>== !=</td>
<td>Comparisons: equal and not equal</td>
</tr>
<tr>
<td>8</td>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td>9</td>
<td>^</td>
<td>Bitwise exclusive OR</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Bitwise inclusive (normal) OR</td>
</tr>
<tr>
<td>11</td>
<td>&amp;&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>?: = += -= *= /= %= &amp;=</td>
<td>= ^= &lt;&lt;= &gt;&gt;=</td>
</tr>
<tr>
<td>14</td>
<td>,</td>
<td>Comma operator</td>
</tr>
</tbody>
</table>
public class Average
{
    public static void main(String[] args)
    {
        double a, b, c, avg;  // Define 4 variables

        a = 3.0;
        b = 4.0;
        c = 6.0;

        avg = (a + b + c)/3.0;

        System.out.print("The average = ");
        System.out.println(avg);
    }
}
Left-associative operators

• A left-associative operator is an operator that groups its operands from left to right:
• Example of left-associative operators:

![Diagram showing binary arithmetic operators](image)

Floating point expression: 7.0 - 2.0 - 1.0
Evaluated as follows: 7.0 - 2.0 - 1.0  (group from left to right)
5.0 - 1.0
4.0
Right-associative operators

• A right-associative operator is an operator that groups its operands from right.

• Example of right-associative operators

Floating point expression: \(- - - - 1.0\)

Evaluated as follows:

\[- - - - 1.0\]
\[- - - (-1.0)\]  \(\text{Note: \((-1.0)\) denotes negative 1.0}\)
\[- - (1.0)\]
\[- (-1.0)\]
\[= 1.0\]
public class RightAssoc {

    public static void main(String[] args) {

        System.out.print("- - - - 1.0 = ");
        System.out.println("- - - - 1.0");

    }

}
Rules for evaluation an arithmetic expression

1. First, apply the operator priority rules

   I.e.: perform the operation of the highest priority first

2. Then, apply the operator associativity rules
### Mathematical functions

- Java's Mathematical functions (methods)

<table>
<thead>
<tr>
<th>Java's method name</th>
<th>Corresponding Mathematical function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sin(x)</td>
<td>Sine function of value $x$</td>
</tr>
<tr>
<td>Math.cos(x)</td>
<td>Cosine function of value $x$</td>
</tr>
<tr>
<td>Math.tan(x)</td>
<td>Tangent function of value $x$</td>
</tr>
<tr>
<td>Math.asin(x)</td>
<td>Arc sine (inverse of sine) function of value $x$</td>
</tr>
<tr>
<td>Math.acos(x)</td>
<td>Arc cosine function of value $x$</td>
</tr>
<tr>
<td>Math.atan(x)</td>
<td>Arc tangent function of value $x$</td>
</tr>
<tr>
<td>Math.exp(x)</td>
<td>$e^x$</td>
</tr>
<tr>
<td>Math.log(x)</td>
<td>Natural log function of value $x$</td>
</tr>
<tr>
<td>Math.log10(x)</td>
<td>10-based log function of value $x$</td>
</tr>
<tr>
<td>Math.pow(a, b)</td>
<td>$a^b$</td>
</tr>
<tr>
<td>Math.sqrt(x)</td>
<td>Squared root of the number $x$</td>
</tr>
</tbody>
</table>
• A real-life example: programming the $a, b, c$-formula

**Quadratic equation:**

$$ax^2 + bx + c = 0,$$

**Solutions:**

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$