Java Syntax

- Statements
- Comments
- Literals
- Expressions
- Operators and Operations
- Data types
- Declaring variables
- Identifiers
Statements

• One can write several statements, one after the other
• Separate each statement by a semicolon (;)
• Indent your code to facilitate readability
Comments

• Comments are ignored by the compiler

• Java allows comments anywhere and they are started by ‘/\*’ and ended by ‘\*/’:
  – /* This is a traditional multi-line comment block. */

Comments cannot be nested

• Java also supports single-line comments:
  – // This is a single-line comment.

- recommended – can be surrounded by ‘/\*..\*/’

- javadoc is a tool for Java which is able to create html based documentation of your code. To use surround your comments with ‘/\*\*… */’
Literals

• A literal represents an exact value of a certain type
  – Integer Literal: 20

• Refer to next slide for possible types of literals

• Hexadecimal integer literals are distinguished from decimal integer literals by leading “0x”
  – Hex literal: 0xf000

• Octal (base 8) integer literals are distinguished by a leading “0” in Java, no octal in C#
  – Octal literal: 07020
<table>
<thead>
<tr>
<th>Literal Type</th>
<th>Can contain</th>
<th>Size (bits)</th>
<th>Min and Max Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>true or false</td>
<td>1</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Character</td>
<td>Unicode character</td>
<td>16</td>
<td>/u0000 to /uFFFFF</td>
</tr>
<tr>
<td>Byte</td>
<td>signed integer</td>
<td>8</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>Short</td>
<td>signed integer</td>
<td>16</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>Integer</td>
<td>signed integer</td>
<td>32</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>Long</td>
<td>signed integer</td>
<td>64</td>
<td>-</td>
</tr>
<tr>
<td>Float</td>
<td>IEEE754 flt.pt.</td>
<td>32</td>
<td>+/-3.4028E+38 to +/-1.4023E-45</td>
</tr>
<tr>
<td>Double</td>
<td>IEEE754 flt.pt.</td>
<td>64</td>
<td>+/-1.79E+308 to +/-4.90E-324</td>
</tr>
</tbody>
</table>
Literals (cont.)

• Integer literals may be designated *long* by appending ‘l’ or ‘L’ to the number:
  – Long integer:
    10000000000000000L;

• Floating point literals
  – contain a decimal point
  – are of type Double by default
  – Double literal: 1.0

• May be designated float by appending an ‘f’ or ‘F’:
  – Float literal: 22.6F
Character Literals

• A simple character literal is a character enclosed within single quotes:
  – Character Literal: ‘.’

• Java also supports the standard escape sequences:
  – ‘\n’ (newline)
  – ‘\t’ (tab)
  – ‘\b’ (bell)
  – ‘\’ (‘) etc.
  – also supports \xxx where xxx is 3 octal digits
Character Literals (cont.)

• A character literal is 2 bytes wide to be able to hold an element from the Unicode Worldwide Character Standard

• Unicode character literals:
  – accommodate international (non-Latin) characters:
  – note: not every font can display them
  - written as ‘\uxxxx’ where xxxx is 1 to 4 hex digits

• Note: ASCII printing characters \u0020 to \u007E map directly into Unicode i.e. have the same binary value
String Literals

• String literals can be a string of any length surrounded by double quotes
  – String literal “abcdef”
Operators

• Many Java/C# operators are self-explanatory:
  +, −, *, /, % arithmetic
  <, >, <=, >= relational
  = assignment
  ==, != equality
  +=, -=, *=, ... compound assignment

• Unary integer operators:
  - arithmetic negation
  + arithmetic negation
  ~ bitwise complement
  ++ increment
  -- decrement

• Binary integer operators:
  &, |, ^ bitwise AND, OR, XOR
  << left shift
  >> right shift with sign fill
  >>> right shift with zero fill

• Unary Integer Operator
  ! Not

• Binary Boolean Operators
  && And
  || Or
  ^ Xor
Expressions

• The simplest expression is a literal.

• An operator combines operand expressions into a larger expression.

• Usually translates to byte code to be executed

• An expression results in a new value and type

• examples:
  - 2+3
  - 1.2+4
  - 2.4l+1.2
Operations on Integers

- The result type of an integer operation depends on the types of the operands
- If one or more operands is of type long, the result is long
- Otherwise, the result is int
- The compiler automatically widens integer values
Operations on Booleans

• The usual boolean operators apply to booleans
  – true && true = true
  – true || false = true

• When one operand of && evaluates to false, evaluation of the expression stops
Floating Point Operations

• Unary operators:
  - arithmetic negation

Binary operators:
  +, -, *, / (addition, subtraction, multiplication, division)
  %  floating point remainder (modulo)

• Compound assignment operators apply

• Undefined operations on floating point values produce special result values (NaN), Inf, etc.) rather than interrupt the program
Floating Point Operations (cont.)

• Type of result depends on type of operands:
  – if all operands are float, result is float
  – otherwise, result is double

• Integer to floating point conversion is automatic since Float is same size
Exercises

Using the instruction
   – System.out.println(literal);
   – Console.WriteLine(literal);
will display the value of ‘literal’.

• Print out the values of some literals.

• Add an integer to a float, and multiply the result to a double and then print it out. Verify your results.

• Divide a float with the value 0 and print it out.