Lecture 6: Numeric Operation Details and Casting

Jan 27 2015

Partial Homework 1 is up; full assignment will be up by midnight.
Exponent operations

System.out.println(Math.pow(2, 3));
   // Displays 8.0
System.out.println(Math.pow(4, 0.5));
   // Displays 2.0
System.out.println(Math.pow(2.5, 2));
   // Displays 6.25
System.out.println(Math.pow(2.5, -2));
   // Displays 0.169

Demonstrate in AreaOfCircle2.java
How an expression is evaluated

Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.

\[
\begin{align*}
3 &+ 4 \times 4 + 5 \times (4 + 3) - 1 \\
3 &+ 4 \times 4 + 5 \times 7 - 1 \\
3 &+ 16 + 5 \times 7 - 1 \\
3 &+ 16 + 35 - 1 \\
19 &+ 35 - 1 \\
54 &- 1 \\
53
\end{align*}
\]

(1) inside parentheses first
(2) multiplication
(3) multiplication
(4) addition
(5) addition
(6) subtraction
Assignment operators (shortcuts)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>• +=</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>• -=</td>
<td>f -= 8.0</td>
<td>f = f -8.0</td>
</tr>
<tr>
<td>• *=</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>• /=</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>• %=</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
</tbody>
</table>

Demonstrate in TimeDisplay2.java
Increment and decrement operators

- `++var` (preincrement)
  The expression `(++var)` increments `var` by 1 and evaluates to the *new* value in `var` *after* the increment.

- `var++` (postincrement)
  The expression `(var++)` evaluates to the *original* value in `var` and increments `var` by 1.

- `--var` (predecrement)
  The expression `(--var)` decrements `var` by 1 and evaluates to the *new* value in `var` *after* the decrement.

- `var--` (postdecrement)
  The expression `(var--)` evaluates to the *original* value in `var` and decrements `var` by 1.
Numeric type conversion rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

- If one of the operands is double, the other is converted into double.
- Otherwise, if one of the operands is float, the other is converted into float.
- Otherwise, if one of the operands is long, the other is converted into long.
- Otherwise, both operands are converted into int.
Type Casting

• Implicit casting
  For example: double x = 3; (type widening)

• Explicit casting
  (datatype) variableName:
  int i = (int)3.0; // type narrowing
  int i = (int)3.9; // Fraction part is truncated

What is wrong with the following?
int x = 5/2.0;