Types, Operators & Coercion

Randy Shepherd
NYU
What is a variable?

→ A named location in memory that holds values
→ Moreover, a storage location and an associated symbolic name (an identifier)
→ What is the value? A bit pattern.
What is a type?

→ Information about a value

→ Moreover, an interpretation of the bit patterns at a memory location

→ char, int, short, long – Integral types

→ float, double – Floating point types

→ boolean
# Java's Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>What is it?</th>
<th>Range of values</th>
<th>Amount of memory allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8-bit, signed, two's compliment integer</td>
<td>-128 .. 127</td>
<td>1 byte</td>
</tr>
<tr>
<td>short</td>
<td>16-bit, signed, two's compliment integer</td>
<td>-32,768 .. 32,767</td>
<td>2 bytes</td>
</tr>
<tr>
<td>int</td>
<td>32-bit, signed, two's compliment integer</td>
<td>-2,147,483,648 .. 2,147,483,647</td>
<td>4 bytes</td>
</tr>
<tr>
<td>long</td>
<td>64-bit, signed, two's compliment integer</td>
<td>-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807</td>
<td>8 bytes</td>
</tr>
<tr>
<td>float</td>
<td>Single precision (32-bit) IEEE 754 floating point</td>
<td>see: <a href="http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3">http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3</a></td>
<td>4 bytes</td>
</tr>
<tr>
<td>double</td>
<td>Double precision (64-bit) IEEE 754 floating point</td>
<td>see: <a href="http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3">http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3</a></td>
<td>8 bytes</td>
</tr>
<tr>
<td>char</td>
<td>A single 16-bit Unicode character</td>
<td>'u0000' .. 'uffff' (Decimal equivalent: 0 .. 65,535)</td>
<td>2 bytes</td>
</tr>
<tr>
<td>boolean</td>
<td>Simple flag to track true/false conditions</td>
<td>true, false</td>
<td>Not precisely defined, but a boolean value represents one bit of information.</td>
</tr>
</tbody>
</table>
Values are bit patterns

→ Internally, the computer stores all data as 0s and 1s.
  - example: 42  --> 101010
  - example: "hi"  --> 0110100001101001

→ Two's compliment representation
Operators

→ Operators are used to apply operations to types

   Ex. int i = 5 + 5;

   String s = "5" + "5";

→ Operators have precedence.

   The order of evaluation of operations.

→ Operators have associativity

   How are operators of equal precedence evaluated?
Operators

→ Arithmetic

+  additive operator (also used for string concatenation)
-  subtraction operator
*  multiplication operator
/  division operator
%  remainder operator
Operators

→ Unary

+  Unary plus operator; indicates positive value
-  Unary minus operator; negates an expression
++ Increment operator; increments a value by 1
-- Decrement operator; decrements a value by 1
!  Logical complement operator; inverts the value of a boolean
<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>postfix</td>
<td>$expr++$ $expr--$</td>
</tr>
<tr>
<td>unary</td>
<td>$++expr$ $--expr$ $+expr$ $-expr$ $~!$</td>
</tr>
<tr>
<td>multiplicative</td>
<td>$*$ $/$ $%$</td>
</tr>
<tr>
<td>additive</td>
<td>$+$ $-$</td>
</tr>
<tr>
<td>shift</td>
<td>$&lt;&lt;$ $&gt;&gt;$ $&gt;&gt;&gt;$</td>
</tr>
<tr>
<td>relational</td>
<td>$&lt;$ $&gt;$ $&lt;=$ $&gt;=$ $instanceof$</td>
</tr>
<tr>
<td>equality</td>
<td>$==$ $!=$</td>
</tr>
<tr>
<td>bitwise AND</td>
<td>$&amp;$</td>
</tr>
<tr>
<td>bitwise exclusive OR</td>
<td>$^$</td>
</tr>
<tr>
<td>bitwise inclusive OR</td>
<td>$</td>
</tr>
<tr>
<td>logical AND</td>
<td>$&amp;&amp;$</td>
</tr>
<tr>
<td>logical OR</td>
<td>$</td>
</tr>
<tr>
<td>ternary</td>
<td>$?$ $:$</td>
</tr>
<tr>
<td>assignment</td>
<td>$=+$ $-=$ $*=$ $/$ $%=$ $&amp;=$ $^=$ $</td>
</tr>
</tbody>
</table>
Precedence example

1 * 2 + 3 * 5 / 4

2 + 3 * 5 / 4

2 + 15 / 4

2 + 3

5

1 + 2 / 3 * 5 - 4

1 + 0 * 5 - 4

1 + 0 - 4

1 - 4

-3
Coercion & Casting

→ Coercion is implicit type conversion
   ex. float x = 9;

→ Casting is explicit type conversion
   ex int i = (int) 9.0;

→ Casting is required when doing 'narrowing' conversions.
Coercion & Casting

7 / 3 * 1.2 + 3 / 2

2 * 1.2 + 3 / 2

2.4 + 3 / 2

2.4 + 1

3.4