Primitive Data Types

15-110 Summer 2010
Margaret Reid-Miller
Data Types

• Data stored in memory is a string of bits (0 or 1).
• What does 1000010 mean?
  66?
  'B'?
  9.2E-44?
• How the computer interprets the string of bits depends on the context.
• In Java, we must make the context explicit by specifying the \textit{type} of the data.
Primitive Data Types

• Java has two categories of data:
  • **primitive data** (e.g., number, character)
  • **object data** (programmer created types)

• There are 8 primitive data types:
  byte, short, int, long, float, double, char, boolean

• Primitive data are only single values; they have no special capabilities.
### Common Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example of Literals</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers (whole numbers)</td>
<td>42, 60634, -8, 0</td>
</tr>
<tr>
<td>double</td>
<td>real numbers</td>
<td>0.039, -10.2, 4.2E+72</td>
</tr>
<tr>
<td>char</td>
<td>single characters</td>
<td>'a', 'B', '&amp;', '6'</td>
</tr>
<tr>
<td>boolean</td>
<td>logical values</td>
<td>true, false</td>
</tr>
</tbody>
</table>
# Numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768 to 32,727</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-9x10^{18} to 9x10^{18}</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>±10^{-45} to ±10^{38}, 7 significant digits</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>±10^{-324} to ±10^{308}, 15 significant digits</td>
</tr>
</tbody>
</table>
Variables

- A variable is a name for a location in memory used to store a data value.
- We use variables to save and restore values or the results of calculations.
- The programmer has to tell Java what type of data will be stored in the variable’s memory location. Its type cannot change.
- During the program execution the data saved in the memory location can change; hence the term “variable”.
Variable Declaration

• Before you can use a variable, you must declare its type and name.
• You can declare a variable only once in a method.
• Examples:
  
  ```java
  int numDimes;
  double length;
  char courseSection;
  boolean done;
  String lastName;
  ```


Declaring Variables

• Declaring a variable instructs the compiler to set aside a portion of memory large enough to hold data of that type.

```c
int count;
double length;
```

• No value has been put in memory yet. That is, the variable is *undefined*. 
Assignment Statements

- An *assignment statement* stores a value into a variable's memory location:
  \[ \text{<variable>} = \text{<expression>}; \]
- An *expression* is anything that has a value: a literal value, a variable, or a more complex calculation.
- The result of the expression is *assigned* to the variable.

\[
\begin{align*}
\text{count} &= 3; \\
\text{length} &= 72.3 + 2.0;
\end{align*}
\]

- The first assignment to a variable *initializes* it.
Re-Assigning Variables

- A variable must be declared exactly once.
- A variable can be assigned and re-assigned values many times after it is declared.

Example:

```java
int x;
x = 4;
System.out.println(x);  // prints 4
x = x + 1;
System.out.println(x);  // prints 5
```
Declaration/Initialization

- Variables can be declared and initialized in one statement:

Examples:

```java
int numDimes = 4;
double length = 52.3;
char courseSection = 'J';
boolean done = true;
String lastName = "Reid-Miller";
int count = 3 + 2;
```
Expressions

- An *expression* is anything that result in a value.
- It must have a type. Why?

  Example: \((2 + 3) * 4\)

**Arithmetic operators:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>1 + 3</td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>12 - 4</td>
<td>8</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>3 * 4</td>
<td>12</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>2.2 / 1.1</td>
<td>2.0</td>
</tr>
<tr>
<td>%</td>
<td>modulo (remainder)</td>
<td>14 % 4</td>
<td>2</td>
</tr>
</tbody>
</table>
Division and Modulo

```c
int a = 40;
double x = 40.0;
int b = 6;
double y = 6.0;
int c;
double z;

c = a / b;  // 6
            c = a % b;  // 4
z = x / y;  // 6.666666667
            c = b % a;  // 6

c = b / a;  // 0
            c = 0 % a;  // 0
z = y / x;  // 0.15
            c = b % 0;  // error

c = 0 / a;  // 0
            c = a / 0;  // error
```
Operator Precedence

- The operators *, /, % are evaluated before the operators +, - because *, /, % have higher precedence than +, -.

Example: \[ 2 + 4 * 5 \]

\[
\begin{align*}
2 + 20 & \quad 22 \\
2 + 20 & \quad 22
\end{align*}
\]

- To change the order use parentheses:

Example: \((2 + 4) * 5\) evaluates to _______
Evaluating expressions

• When an expression contains more than one operator with the same level of precedence, they are evaluated from left to right.
  • $2 + 2 + 3 - 1$ is $((2 + 2) + 3) - 1$ which is 6
  • $2 \times 4 \% 5$ is $((2 \times 4) \% 5)$ which is 3
  • $2 * 3 - 2 + 7 / 4$
    $\left\{ \begin{array}{c}
    6 - 2 + 7 / 4 \\
    6 - 2 + 1 \\
    4 + 1 \\
    5
    \end{array} \right\} $
Other operators

- **Assignment operators**: =, +=, -=, *=, /=, %=  

**Example:**  
- Shortcut for `x = x + 2;` is `x += 2;`  
  (“add 2 to x”)  
- Shortcut for `y = y * 3;` is `y *= 3;`  
  (“multiply y by 3”)  

- **Increment / Decrement** operators: ++, --  
  - Shortcut for `x = x + 1;` is `x++;` (“increment x”)  
  - Shortcut for `y = y - 1;` is `y--;` (“decrement y”)
Data Conversion

- **Widening conversions** convert data to another type that has the same or more bits of storage. *E.g.*, 
  - short to int, long (safe)
  - int to long (safe)
  - int to float, double (magnitude the same but can lose precision)

- **Narrowing conversions** convert data to another type that has the fewer bits of storage and/or can lose information. *E.g.*, 
  - double or float to any integer type
  - double to float
Mixing Types

• When a Java operator is applied to operands of different types, Java does a widening conversion automatically, known as a promotion.

• Example:
  - \(2.2 \times 2\) evaluates to \(4.4\)
  - \(1.0 / 2\) evaluates to \(0.5\)
  - `double x = 2;` assigns \(2.0\) to `x`
  - `"count = " + 4` evaluates to `"count = 4"`

string concatenation
Mixing Types

• Conversions are done on one operator at a time in the order the operators are evaluated.

\[ \frac{3}{2} \times 3.0 + \frac{8}{3} = 5.0 \]

\[ 2.0 \times \frac{4}{5} + \frac{6}{4.0} = 3.2 \]
Mixing Types

• String concatenation has the same precedence as + – and is evaluated left to right.

\[
\begin{align*}
1 \ + \ "x" \ + \ 4 & \rightarrow "1x4" \\
"2+3=\ " \ + \ 2 \ + \ 3 & \rightarrow "2+3=23" \\
1 \ + \ 2 \ + \ "3" & \rightarrow "33" \\
"2*3=\ " \ + \ 2 \ * \ 3 & \rightarrow "2*3=6" \\
4 \ - \ 1 \ + \ "x" & \rightarrow "3x" \\
"x" \ + \ 4 \ - \ 1 & \rightarrow \text{error}
\end{align*}
\]
Type Casting

- **Type casting** tells Java to convert one type to another.

**Uses:**
- Convert an `int` to a `double` to force floating-point division.
- Truncate a `double` to an `int`.

**Examples:**
- `double average = (double) 12 / 5`
- `int feet = (int) (28.3 / 12.0)`
Type casting

- Because type casting has high precedence, it casts the operand immediately to its right only.

Example:

```java
double s = (double) 2 + 3 / 2;  // 3.0
double s2 = (double) (2 + 3) / 2;  // 2.5
double average = (double) 22 / 4;  // 5.5
double average2 = 22 / (double) 4;  // 5.5
double wrong = (double) (22 / 4);  // 5.0
```
char data type

- A variable of type `char` holds exactly one (Unicode) character/symbol.
- Every character has a corresponding integer value.
- The digit characters '0' ... '9' have consecutive integer values, as do the letters 'A' ... 'Z' and 'a' ... 'z'. We can use this ordering to sort alphabetically.
- Conversions:

  ```java
  String letter = "" + 'M';       // evaluates to "M"
  int aAsInt = 'a';              // evaluates to 97
  'a' + 2;                       // evaluates to 99
  char c = (char)('a' + 2);      // evaluates to 'c'
  ```
# Operator Precedence

<table>
<thead>
<tr>
<th>Operator type</th>
<th>Operator</th>
<th>Associates</th>
</tr>
</thead>
<tbody>
<tr>
<td>grouping</td>
<td>(expression)</td>
<td>Left to right</td>
</tr>
<tr>
<td>unary</td>
<td>++, --, +, -</td>
<td>Right to left</td>
</tr>
<tr>
<td>cast</td>
<td>(type)</td>
<td>Right to left</td>
</tr>
<tr>
<td>multiplicative</td>
<td>*, /, %</td>
<td>Left to right</td>
</tr>
<tr>
<td>additive</td>
<td>+, -</td>
<td>Left to right</td>
</tr>
<tr>
<td>assignment</td>
<td>=, +=, -=, *=, /=, %=</td>
<td>Right to left</td>
</tr>
</tbody>
</table>