Building Java Programs

Chapter 2
Lecture 2-1: Expressions and Variables

reading: 2.1 - 2.2
...you know, I could teach you to process your own data!
Data and expressions

reading: 2.1
Data types

• Internally, computers store everything as 1s and 0s
  
  104  $\rightarrow$  01101000
  "hi"  $\rightarrow$  0110100001101001
  h  $\rightarrow$  01101000

• How are h and 104 differentiated?

• **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types
  - Examples: integer, real number, string
Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers</td>
<td>42, -3, 0, 926394</td>
</tr>
<tr>
<td>double</td>
<td>real numbers</td>
<td>3.1, -0.25, 9.4e3</td>
</tr>
</tbody>
</table>
| char      | single text characters       | 'a', 'X', '?', '
'     |
| boolean   | logical values               | true, false            |

- Why does Java distinguish integers vs. real numbers?
Integer or real number?

- Which category is more appropriate?

<table>
<thead>
<tr>
<th>integer (int)</th>
<th>real number (double)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Temperature in degrees Celsius
2. The population of lemmings
3. Your grade point average
4. A person's age in years
5. A person's weight in pounds
6. A person's height in meters
7. Number of miles traveled
8. Number of dry days in the past month
9. Your locker number
10. Number of seconds left in a game
11. The sum of a group of integers
12. The average of a group of integers

Expressions

• **expression**: A value or operation that computes a value.
  
  • Examples:  
    \[ 1 + 4 \times 5 \]
    \[ (7 + 2) \times 6 / 3 \]
    \[ 42 \]

  • The simplest expression is a *literal value*.
  • A complex expression can use operators and parentheses.
Arithmetic operators

• **operator**: Combines multiple values or expressions.
  
  +   addition 
  −   subtraction (or negation) 
  *   multiplication 
  /   division 
  %   modulus (a.k.a. remainder)

• As a program runs, its expressions are *evaluated*.
  
  • 1 + 1 evaluates to 2
  • `System.out.println(3 * 4);` prints 12
    • How would we print the text 3 * 4?
Integer division with \(/

- When we divide integers, the quotient is also an integer.
  - $14 \div 4$ is 3, not 3.5

\[
\begin{array}{c}
\text{3} \\
4 \quad 14
\end{array}
\quad \quad
\begin{array}{c}
\text{4} \\
10 \quad 45
\end{array}
\quad \quad
\begin{array}{c}
\text{52} \\
27 \quad 1425
\end{array}
\]

- 4 ) 14
  12
  2

- 10 ) 45
  40
  5

- 27 ) 1425
  135
  75
  54
  21

- More examples:
  - $32 \div 5$ is 6
  - $84 \div 10$ is 8
  - $156 \div 100$ is 1

- Dividing by 0 causes an error when your program runs.
Integer remainder with %

- The % operator computes the remainder from integer division.
  - $14 \% 4$ is 2
  - $218 \% 5$ is 3

\[
\begin{array}{c}
4 \overline{) 14} \\
 \underline{12} \\
2
\end{array}
\quad \begin{array}{c}
5 \overline{) 218} \\
 \underline{20} \\
18 \\
15 \\
3
\end{array}
\]

- Applications of % operator:
  - Obtain last digit of a number: $230857 \% 10$ is 7
  - Obtain last 4 digits: $658236489 \% 10000$ is 6489
  - See whether a number is odd: $7 \% 2$ is 1, $42 \% 2$ is 0

What is the result?

- $45 \% 6$
- $2 \% 2$
- $8 \% 20$
- $11 \% 0$
Precedence

**precedence**: Order in which operators are evaluated.

- Generally operators evaluate left-to-right.
  \[ 1 - 2 - 3 \text{ is } (1 - 2) - 3 \text{ which is } -4 \]

- But \[ * / \% \] have a higher level of precedence than \[ + - \]
  \[ 1 + 3 * 4 \text{ is } 13 \]
  \[ 6 + 8 / 2 * 3 \]
  \[ 6 + 4 * 3 \]
  \[ 6 + 12 \text{ is } 18 \]

- Parentheses can force a certain order of evaluation:
  \[ (1 + 3) * 4 \text{ is } 16 \]

- Spacing does not affect order of evaluation
  \[ 1+3 * 4-2 \text{ is } 11 \]
Precedence examples

1 * 2 + 3 * 5 % 4

2

1 + 8 % 3 * 2 - 9

2

2 + 3 * 5 % 4

15

1 + 2 * 2 - 9

4

2 + 3

3

5

5

2

1 + 4 - 9

5

-4

Copyright 2010 by Pearson Education
Precedence questions

What values result from the following expressions?

- $9 / 5$
- $695 \% 20$
- $7 + 6 \times 5$
- $7 \times 6 + 5$
- $248 \% 100 / 5$
- $6 \times 3 - 9 / 4$
- $(5 - 7) \times 4$
- $6 + (18 \% (17 - 12))$
Real numbers (type `double`)

- **Examples:** $6.022$, $-42.0$, $2.143e17$
  - Placing `.0` or `.` after an integer makes it a `double`.

- **The operators** `+`, `-`, `*`, `/`, `%`, `()` all still work with `double`.
  - `/` produces an exact answer: $15.0 / 2.0$ is $7.5$
  - Precedence is the same: `()` before `*`, `/`, `%` before `+`, `-`
Real number example

\[ 2.0 \times 2.4 + 2.25 \times 4.0 / 2.0 \]
\[ \underline{4.8} + 2.25 \times 4.0 / 2.0 \]
\[ \underline{4.8} + \underline{9.0} / 2.0 \]
\[ \underline{4.8} + \underline{4.5} \]
\[ \underline{9.3} \]
Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`

- The conversion is per-operator, affecting only its operands.

```plaintext
7 / 3 * 1.2 + 3 / 2
  |  
  2 * 1.2 + 3 / 2
    |  
  2.4 + 3 / 2
    |  
  2.4 + 1
      |  
  3.4
```

- `3 / 2` is `1` above, not `1.5`.

```plaintext
2.0 + 10 / 3 * 2.5 - 6 / 4
  |  
  2.0 + 3 * 2.5 - 6 / 4
    |  
    2.0 + 7.5 - 6 / 4
      |  
      2.0 + 7.5 - 1
        |  
        8.5
```
String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

  - "hello" + 42 is "hello42"
  - 1 + "abc" + 2 is "1abc2"
  - "abc" + 1 + 2 is "abc12"
  - 1 + 2 + "abc" is "3abc"
  - "abc" + 9 * 3 is "abc27"
  - "1" + 1 is "11"
  - 4 - 1 + "abc" is "3abc"

- Use + to print a string and an expression's value together.

```
System.out.println("Grade: " + (95.1 + 71.9) / 2);
```

- **Output**: Grade: 83.5
Variables

reading: 2.2
What's bad about the following code?

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println((38 + 40 + 30) +
                            (38 + 40 + 30) * .08 +
                            (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements
Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:
  
- Steps for using a variable:
  - *Declare* it - state its name and type
  - *Initialize* it - store a value into it
  - *Use* it - print it or use it as part of an expression
Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- **Syntax**:
  
  \[ \text{type name;} \]
  
  - The name is an *identifier*.

  - `int zipcode;`

  - `double myGPA;`
Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

**Syntax:**

\[
\text{name = expression;}
\]

- `int zipcode;`  
  - `zipcode = 90210;`  
  - ` zipcode | 90210`

- `double myGPA;`  
  - `myGPA = 1.0 + 2.25;`  
  - ` myGPA | 3.25`
Using variables

- Once given a value, a variable can be used in expressions:

  ```java
  int x;
  x = 3;
  System.out.println("x is " + x); // x is 3
  System.out.println(5 * x - 1); // 5 * 3 - 1
  ```

- You can assign a value more than once:

  ```java
  int x;
  x = 3;
  System.out.println(x + " here"); // 3 here
  x = 4 + 7;
  System.out.println("now x is " + x); // now x is 11
  ```
Declaration-initialization

- A variable can be declared/initialized in one statement.

- Syntax:
  
  \[
  \text{type name} = \text{value};
  \]

- double myGPA = 3.95;

- int x = (11 % 3) + 12;
Assignment and algebra

- Assignment uses =, but it is not an algebraic equation.
  - = means, "store the value at right in variable at left"

- The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

\[
\begin{align*}
\text{int } x &= 3; \\
x &= x + 2; & \text{// } ???
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
x & 5 \\
\hline
\end{array}
\]
Assignment and types

- A variable can only store a value of its own type.
  - \( \text{int } x = 2.5; \quad // \text{ERROR: incompatible types} \)

- An int value can be stored in a double variable.
  - The value is converted into the equivalent real number.
  - \( \text{double myGPA} = 4; \)
  - \( \text{double avg} = 11 / 2; \)
    - Why does avg store 5.0 and not 5.5?
Compiler errors

- A variable can't be used until it is assigned a value.
  
  ```java
  int x;
  System.out.println(x);   // ERROR: x has no value
  ```

- You may not declare the same variable twice.
  
  ```java
  int x;
  int x;       // ERROR: x already exists
  ```

  ```java
  int x = 3;
  int x = 5;   // ERROR: x already exists
  ```

  - How can this code be fixed?
Printing a variable's value

- Use `+` to print a string and a variable's value on one line.

```java
double grade = (95.1 + 71.9 + 82.6) / 3.0;
System.out.println("Your grade was "+ grade);

int students = 11 + 17 + 4 + 19 + 14;
System.out.println("There are "+ students + " students in the course.");
```

- Output:

  Your grade was 83.2
  There are 65 students in the course.
Receipt question

Improve the receipt program using variables.

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:" + 38 + 40 + 30);
        System.out.println("Tax:" + (38 + 40 + 30) * .08);
        System.out.println("Tip:" + (38 + 40 + 30) * .15);
        System.out.println("Total:" + (38 + 40 + 30) * .15 + 
                                    (38 + 40 + 30) * .08);
    }
}
```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}