Building Java Programs
Chapter 2
Lecture 2-1: Expressions and Variables

**reading:** 2.1 - 2.2

Data and expressions

**reading:** 2.1
self-check: 1-4
videos: Ch. 2 #1
Data types

- **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

- Internally, computers store everything as 1s and 0s
  
  \[
  \begin{align*}
  104 & \rightarrow 01101000 \\
  "h" & \rightarrow 01101000
  \end{align*}
  \]

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>
<tr>
<td>char</td>
<td>single text characters</td>
<td>'a', 'X', '?'</td>
</tr>
<tr>
<td>boolean</td>
<td>logical values</td>
<td>true, false</td>
</tr>
</tbody>
</table>
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 \ast 4`?
Integer division with /

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

\[
\begin{array}{ccc}
4 & 14 & 45 \\
12 & 40 & 135 \\
2 & 5 & 75 \\
\end{array}
\]

- More examples:
  - $32 / 5$ is 6
  - $84 / 10$ is 8
  - $156 / 100$ 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

\[
\begin{array}{ccc}
4 & 14 & 43 \\
12 & 20 & \text{What is the result?} \\
2 & 15 & \\
\end{array}
\]

- 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
• **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.
    - $1 - 2 - 3$ is $(1 - 2) - 3$ which is $-4$
    - But $*/\%$ have a higher level of precedence than $+-$
      - $1 + 3 \times 4$ is $13$
      - $6 + 8 / 2 \times 3$
      - $6 + 4 \times 3$ is $18$
  - Parentheses can force a certain order of evaluation:
    - $(1 + 3) \times 4$ is $16$
  - Spacing does not affect order of evaluation
    - $1 + 3 \times 4 - 2$ is $11$

**Precedence examples**

```
1 * 2 + 3 * 5 % 4
  \
2 + 3 * 5 % 4
  \n2 + 15 % 4
  \n2 + 3
  \n5
```

```
1 + 8 % 3 * 2 - 9
  \
2 + 8 % 3 * 2 - 9
  \n2 + 15 % 4
  \n2 + 4
  \n5
  \n5 - 9
  \n-4
```
Precedence questions

- What values result from the following expressions?
  - 9 / 5
  - 695 % 20
  - 7 + 6 * 5
  - 7 * 6 + 5
  - 248 % 100 / 5
  - 6 * 3 - 9 / 4
  - (5 - 7) * 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

\[ 2.0 \times 2.4 + 2.25 \times 4.0 / 2.0 \]

\[ \begin{array}{c}
4.8 \\
+ 2.25 \times 4.0 / 2.0 \\
\hline
4.8 \\
+ 9.0 / 2.0 \\
\hline
4.8 \\
+ 4.5 \\
\hline
9.3
\end{array} \]

Mixing types

- **When int and double are mixed, the result is a double.**
  - 4.2 \times 3 is 12.6
- The conversion is per-operator, affecting only its operands.

\[ \frac{7}{3} \times 1.2 + 3 / 2 \]

\[ \begin{array}{c}
2 \\
\times 1.2 + 3 / 2 \\
\hline
2.4 \\
+ 3 / 2 \\
\hline
2.4 \\
+ 1 \\
\hline
3.4
\end{array} \]

\[ \frac{2.0 + 10}{3} \times \frac{2.5 - 6}{4} \]

\[ \begin{array}{c}
2.0 \\
+ 3 \\
\hline
2.4 \\
+ 1 \\
\hline
3.4
\end{array} \]

\[ 2.0 + 7.5 - 6 / 4 \]

\[ \begin{array}{c}
2.0 \\
+ 7.5 \\
\hline
9.5 \\
- 1 \\
\hline
8.5
\end{array} \]

- 3 / 2 is 1 above, not 1.5.
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
- self-check: 1-15
- exercises: 1-4
- videos: Ch. 2 #2
Receipt example

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:
  
  ```
  type name;
  ```
  
  - The name is an identifier.

  ```
  int x;
  ```

  ```
  double myGPA;
  ```

Assignment

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

- Syntax:
  
  ```
  name = expression;
  ```

  ```
  int x;
  x = 3;
  ```

  ```
  double myGPA;
  myGPA = 1.0 + 2.25;
  ```
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:

```plaintext
    type name = 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"
  - x = 3; means "x becomes 3" or "x should now store 3"

- What happens here?
  ```
  int x = 3;
  x = x + 2;  // ???
  ```
  ```
  x
  5
  ```

Assignment and types

- A variable can only store a value of its own type.
  ```
  int x = 2.5;  // ERROR: incompatible types
  ```

- An int value can be stored in a double variable.
  - The value is converted into the equivalent real number.
  ```
  double myGPA = 4;
  ```
  ```
  myGPA 4.0
  ```
  ```
  double avg = 11 / 2;
  ```
  ```
  avg 5.0
  ```
  - 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);
  ```

  ```java
  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:");
        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) * .15 +
        (38 + 40 + 30) * .08);
    }
}
```

Receipt answer

```java
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);
    }
}
```