THE IF STATEMENT IN JAVA

THE FIRST OF THE TWO BRANCHING STATEMENTS in Java is the `if` statement, which you have already seen in Section 3.1. It takes the form

```
if (boolean-expression)
     statement-1
else
     statement-2
```

As usual, the statements inside an `if` statement can be blocks. The `if` statement represents a two-way branch. The `else` part of an `if` statement -- consisting of the word "else" and the statement that follows it -- can be omitted.

3.5.1 The Dangling else Problem

Now, an `if` statement is, in particular, a statement. This means that either **statement-1** or **statement-2** in the above `if` statement can itself be an `if` statement. A problem arises, however, if **statement-1** is an`if` statement that has no `else` part. This special case is effectively forbidden by the syntax of Java. Suppose, for example, that you type

```
if ( x > 0 )
    if (y > 0)
       System.out.println("First case");
else
    System.out.println("Second case");
```

Now, remember that the way you've indented this doesn't mean anything at all to the computer. You might think that the `else` part is the second half of your "`if (x > 0)`" statement, but the rule that the computer follows attaches the `else` to "`if (y > 0)`", which is closer. That is, the computer reads your statement as if it were formatted:

```
if ( x > 0 )
    if (y > 0)
       System.out.println("First case");
    else
        System.out.println("Second case");
```

You can force the computer to use the other interpretation by enclosing the nested `if` in a block:

```
if ( x > 0 ) {
    if (y > 0)
       System.out.println("First case");
}
else
    System.out.println("Second case");
```

These two `if` statements have different meanings: In the case when `x <= 0`, the first statement doesn't print anything, but the second statement prints "Second case".

3.5.2 The if...else if Construction

Much more interesting than this technicality is the case where **statement-2**, the `else` part of the `if` statement, is itself an `if` statement. The statement would look like this (perhaps without the final else part):

```
if (boolean-expression-1)
     statement-1
else
     if (boolean-expression-2)
         statement-2
     else
         statement-3
```

However, since the computer doesn't care how a program is laid out on the page, this is almost always written in the format:

```
if (boolean-expression-1)
     statement-1
else if (boolean-expression-2)
     statement-2
else
     statement-3
```

You should think of this as a single statement representing a three-way branch. When the computer executes this, one and only one of the three statements -- **statement-1**, **statement-2**, or **statement-3** -- will be executed. The computer starts by evaluating **boolean-expression-1**. If it is `true`, the computer executes **statement-1** and then jumps all the way to the end of the outer if statement, skipping the other two**statement**s. If **boolean-expression-1** is `false`, the computer skips **statement-1** and executes the second, nested if statement. To do this, it tests the value of **boolean-expression-2** and uses it to decide between **statement-2** and **statement-3**.

Here is an example that will print out one of three different messages, depending on the value of a variable named `temperature`:

```
if (temperature < 50)
   System.out.println("It's cold.");
else if (temperature < 80)
   System.out.println("It's nice.");
else
   System.out.println("It's hot.");
```

If `temperature` is, say, 42, the first test is `true`. The computer prints out the message "It's cold", and skips the rest -- without even evaluating the second condition. For a temperature of 75, the first test is`false`, so the computer goes on to the second test. This test is `true`, so the computer prints "It's nice" and skips the rest. If the temperature is 173, both of the tests evaluate to `false`, so the computer says "It's hot" (unless its circuits have been fried by the heat, that is).

You can go on stringing together "else-if's" to make multi-way branches with any number of cases:

```
if (boolean-expression-1)
     statement-1
else if (boolean-expression-2)
     statement-2
else if (boolean-expression-3)
     statement-3
  .
  . // (more cases)
  .
else if (boolean-expression-N)
     statement-N
else
     statement-(N+1)
```

The computer evaluates boolean expressions one after the other until it comes to one that is `true`. It executes the associated statement and skips the rest. If none of the boolean expressions evaluate to `true`, then the statement in the `else` part is executed. This statement is called a multi-way branch because only one of the statements will be executed. The final `else` part can be omitted. In that case, if all the boolean expressions are false, none of the statements are executed. Of course, each of the statements can be a block, consisting of a number of statements enclosed between { and }. (Admittedly, there is lot of syntax here; as you study and practice, you'll become comfortable with it.)

3.5.3 If Statement Examples

As an example of using `if` statements, lets suppose that `x`, `y`, and `z` are variables of type int, and that each variable has already been assigned a value. Consider the problem of printing out the values of the three variables in increasing order. For examples, if the values are 42, 17, and 20, then the output should be in the order 17, 20, 42.

One way to approach this is to ask, where does `x` belong in the list? It comes first if it's less than both `y` and `z`. It comes last if it's greater than both `y` and `z`. Otherwise, it comes in the middle. We can express this with a 3-way `if` statement, but we still have to worry about the order in which `y` and `z` should be printed. In pseudocode,

```
if (x < y && x < z) {
    output x, followed by y and z in their correct order
}
else if (x > y && x > z) {
    output y and z in their correct order, followed by x
}
else {
    output x in between y and z in their correct order
}
```

Determining the relative order of `y` and `z` requires another `if` statement, so this becomes

```
if (x < y && x < z) {        // x comes first
    if (y < z)
       System.out.println( x + " " + y + " " + z );
    else
       System.out.println( x + " " + z + " " + y );
}
else if (x > y && x > z) {   // x comes last
    if (y < z)
       System.out.println( y + " " + z + " " + x );
    else
       System.out.println( z + " " + y + " " + x );
}
else {                       // x in the middle
    if (y < z)
       System.out.println( y + " " + x + " " + z);
    else
       System.out.println( z + " " + x + " " + y);
}
```

You might check that this code will work correctly even if some of the values are the same. If the values of two variables are the same, it doesn't matter which order you print them in.

Note, by the way, that even though you can say in English "if x is less than y and z," you can't say in Java "`if (x < y && z)`". The `&&` operator can only be used

between boolean values, so you have to make separate tests, x<y and x<z, and then combine the two tests with &&.

There is an alternative approach to this problem that begins by asking, "which order should x and y be printed in?" Once that's known, you only have to decide where to stick in z. This line of thought leads to different Java code:

```
if ( x < y ) {  // x comes before y
   if ( z < x )   // z comes first
      System.out.println( z + " " + x + " " + y);
   else if ( z > y )   // z comes last
      System.out.println( x + " " + y + " " + z);
   else   // z is in the middle
      System.out.println( x + " " + z + " " + y);
}
else {          // y comes before x
   if ( z < y )   // z comes first
      System.out.println( z + " " + y + " " + x);
   else if ( z > x )  // z comes last
      System.out.println( y + " " + x + " " + z);
   else  // z is in the middle
      System.out.println( y + " " + z + " " + x);
}
```

Once again, we see how the same problem can be solved in many different ways. The two approaches to this problem have not exhausted all the possibilities. For example, you might start by testing whether x is greater than y. If so, you could swap their values. Once you've done that, you know that x should be printed before y.

Source : http://math.hws.edu/javanotes/c3/s5.html