Introduction to Computer Programming

Using the Java Programming Language

3. Conditional Statements

Conditional statements in Java are a form of control statement in that it can control different paths that your program can take. There are two main types of conditional statements: the If statement (comprising of If, If-Else, and If-Else-If statements) and the Switch statement.

Relational Operators

Relational operators are used to compare two or more values in a conditional statement. If the relation is true, then it will return the boolean true. If the relation is false, then it will return the boolean false. The following operators are available in Java.

Operator	Usage	Description	Example
>	`a > b`	a is greater than b	`5 > 2` returns `true`
<	`a < b`	a is less than b	`5 < 2` returns `false`
>=	`a >= b`	a is greater than or equal to b	`5 >= 2` returns `true`
<=	`a <= b`	a is less than or equal to b	`5 <= 2` return `false`
==	`a == b`	a is equal to b	`5 == 2` returns `false`
!=	`a != b`	a is not equal to b	`5 != 2` returns `true`

Important: These relational operators are not used with `String` variable types. You must use the `String` methods to compare strings. An example of comparing strings is shown here:

```
if (string1.equals(string2))
```

If Statements

In its most basic form, an if statement executes a single statement or a block of statements if a boolean expression evaluates to true. Here's the syntax:

```
if (boolean-expression)
   statement
```

The boolean expression must be enclosed in parentheses. If you use only a single statement, it must end with a semicolon. However, the statement can also be a statement block enclosed by braces. In that case, each statement within the block needs a semicolon, but the block itself doesn't.

Example:

```
double commissionRate = 0.0;

if (salesTotal > 10000.0)
    commissionRate = 0.05;
```

In this example, a variable named `commissionRate` is initialized to 0.0 and then set to 0.05 if `salesTotal` is greater than 10000.0.

Example Using a Block

```
double commissionRate = 0.0;

if (salesTotal > 10000.0) {
    commissionRate = 0.05;
    commission = salesTotal * commissionRate;
}
```

In this example, the two statements within the braces are executed if `salesTotal` is greater than $10,000. Otherwise, neither statement is executed.

Remember that comparisons use two equal signs, not one! Although this code will compile, it will not compare the values as you expect:

```
if (x = 3)
     System.out.println("The value of x is 3");
```


```
if (x == 3)
     System.out.println("The value of x is 3");
```

If-else Statements

An if statement can include an else clause that executes a statement or block if the boolean expression is not true. Its basic format is

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

Example

```
double commissionRate;

if (salesTotal <= 10000.0)
    commissionRate = 0.02;
else
    commissionRate = 0.05;
```

In this example, the commission rate is set to 2% if the sales total is less than or equal to $10,000. If the sales total is greater than $10,000, the commission rate is set to 5%.

Example using a Block

```
double commissionRate;
if (salesTotal <= 10000.0) {
    commissionRate = 0.02;
    level1Count++;
}
else {
    commissionRate = 0.05;
    level2Count++;
}
```

If-else-if Statements

You can test for more than two choices. For example, what if we wanted to test for different age ranges, say 19 to 39, and 40 and over? For more than two choices, the *if ... else if* statement can be used. The structure of an *if ... else if* is:

```
if ( condition_one )
   statement;
else if ( condition_two )
   statement;
else
   statement;
```

Example

```
int user = 21;

if (user <= 18)
     System.out.println("User is 18 or younger");
else if (user > 18 && user < 40)
     System.out.println("User is between 19 and 39");
else
     System.out.println("User is older than 40");
```

Example using a Block

```
int user = 21;

if (user <= 18) {
     System.out.println("User is 18 or younger");
     System.out.println("They are very young.");
}
else if (user > 18 && user < 40) {
     System.out.println("User is between 19 and 39");
     System.out.println("They are middle aged.");
}
else {
     System.out.println("User is older than 40");
     System.out.println("User is older than 40");
}
```

Nested If Statements

The statement that goes in the if or else part of an if-else statement can be any kind of Java statement, including another if or if-else statement. This arrangement is nesting, and an if or if-else statement that includes another if or if-else statement is a nested if statement.

The general form of a nested if statement is this:

```
if (expression-1)
    if (expression-2)
        statement-1
    else
        statement-2
else
    if (expression-3)
        statement-3
    else
        statement-4
```

In this example, expression-1 is the first to be evaluated. If it evaluates to true, expression-2 is evaluated. If that expression is true, statement-1 is executed; otherwise, statement-2 is executed. But if expression-1 is false, expression-3 is evaluated. If expression-3 is true, statement-3 is executed; otherwise, statement-4 is executed.

Here's an example that implements a complicated commission structure based on two variables, named salesClass and salesTotal:

```
if (salesClass == 1)
    if (salesTotal < 10000.0)
        commissionRate = 0.02;
    else
        commissionRate = 0.04;
else
    if (salesTotal < 10000.0)
        commissionRate = 0.025;
    else
        commissionRate = 0.05;
```

The trick of using nested if statements is to know how Java pairs else keywords with if statements. The rule is actually very simple: Each else keyword is matched with the most previous if statement that hasn't already been paired with an else keyword.

Switch Statements

A switch statement is useful when you need to select one of several alternatives based on the value of an integer, a character, or a String variable. The basic form of the switch statement is this:

```
switch (expression) {
  case constant:
        statements;
        break;
  case constant-2:
        statements;
        break;   ...
   default:
        statements;
        break;
}
```

The expression must evaluate to an int, short, byte, or char. It can't be a long or a floating-point type.

Each grouping of code lines that starts with the case keyword and ends with a break statement is a case group. You can code as many case groups as you want or need. Each group begins with the word case, followed by a constant (usually, a numeric, character, or string literal) and a colon.

Then you code one or more statements that you want executed if the value of the switch expression equals the constant. The last line of each case group is a break statement, which causes the entire switch statement to end.

The default group, which is optional, is like a catch-all case group. Its statements are executed only if none of the previous case constants matches the switch expression.

The case groups are not true blocks marked with braces. Instead, each case group begins with the case keyword and ends with the case keyword that starts the next case group. All the case groups together, however, are defined as a block marked with a set of braces.

The last statement in each case group usually is a break statement. A break statement causes control to skip to the end of the switch statement. If you omit the break statement, control falls through to the next case group. Accidentally leaving out break statements is the most common cause of trouble with using a switch statement.

Here's an example of a switch statement that assigns a value to a variable named `commissionRate` based on the value of an integer variable named `salesClass`:

```
double commissionRate;
switch (salesClass) {
    case 1:
        commissionRate = 0.02;
        break;
    case 2:
        commissionRate = 0.035;
        break;
    case 3:
        commissionRate = 0.05;
        break;
    default:
        commissionRate = 0.0;
        break;
}
```

The switch statement can also evaluate char data. In the following example, a char variable named `salesCategory` is evaluated to assign commission rates. The possible sales categories are A, B, or C. However, the category codes may be uppercase or lowercase:

```
double commissionRate;
switch (salesCategory) {
    case 'A':
    case 'a':
        commissionRate = 0.02;
        break;
    case 'B':
    case 'b':
        commissionRate = 0.035;
        break;
    case 'C':
    case 'c':
        commissionRate = 0.05;
        break;
    default:
        commissionRate = 0.0;
        break;
}
```

The key to understanding this example is realizing that you don't have to code any statements at all for a case group, and that if you omit the break statement from a case group, control falls through to the next case group. Thus, the case 'A' group doesn't contain any statements, but it falls through to the case 'a' group.

Beginning with Java 7, you can also use string values in a switch statement. For example:

```
double commissionRate;
switch (salesCategoryName) {
    case "Category A":
        commissionRate = 0.02;
        break;
    case "Category B":
        commissionRate = 0.035;
        break;
    case "Category C":
        commissionRate = 0.05;
        break;
    default:
        commissionRate = 0.0;
        break;
}
```

Logical Operators

A logical operator (sometimes called a "boolean operator") in Java programming is an operator that returns a boolean result that's based on the boolean result of one or two other expressions.

Sometimes, expressions that use logical operators are called "compound expressions" because the effect of the logical operators is to let you combine two or more condition tests into a single expression.

Operator	Name	Description
!	Not	Returns true if the operand to the right evaluates to false. Returns false if the operand to the right is true.
&	And	Returns true if both of the operands evaluate to true. Both operands are evaluated before the And operator is applied.
|	Or	Returns true if at least one of the operands evaluates to true. Both operands are evaluated before the Or operator is applied.
^	Xor	Returns true if one-and-only-one of the operands evaluates to true. Returns false if both operands evaluate to true or if both operands evaluate to false.
&&	Conditional And	Same as &, but if the operand on the left returns false, it returns false without evaluating the operand on the right.
||	Conditional Or	Same as |, but if the operand on the left returns true, it returns true without evaluating the operand on the right.

Learning about the logical operators means that you too will need to brush up on your logic. I think the easiest way to put it is like this:

AND is when you need everything to be true

OR is when you need only one thing to be true

Example 1

```
public class Test {

   public static void main(String args[]) {
      boolean a = true;
      boolean b = false;

      System.out.println("a && b = " + (a&&b));
      System.out.println("a || b = " + (a||b) );
      System.out.println("!(a && b) = " + !(a && b));
   }
}
```

Produces the following output:

```
a && b = false
a || b = true
!(a && b) = true
```

Example 2 – And Operator

Let's say you wanted your program to output "You Win!" when num1 is equal to 3 and num2 is equal to 5.

```
if (num1 == 3) {
     If (num2 == 5)
          System.out.println(“You win!”);
}
```

First, Java will look at the first if statement. Then, if it's true, it will go on to the second if statement. If that one is also true, then it will print out "You win!" on the screen. Seems simple enough.

But there has to be a better way than putting an if statement inside of another if statement!

There is, by using the && operator.

```
if (num1 == 3 && num2 == 5)
     System.out.println(“You win!”);
```

Do you see how much cleaner that code looks? It even sounds better when you say it in English. If num1 is equal to 3 and num2 is equal to 5, then output "You win!".

Example 3 – Or Operator

Up until this point, it was ridiculously difficult to make a bunch of if statements so that the program will execute a piece of code if only one out of several conditions is true. The Or operator allows just this, the ability to have an if statement be true if only one out of several conditions is met.

```
if (num1 == 3 || num2 == 5)
     System.out.println(“You win!”);
```

Do you see how this example of code is different from the previous? The code will run if num1 is equal to 3 OR num2 is equal to 5. Only one of them has to be true for that to happen. If both are true, that's fine also. The only time that if statement will be false is when num1 is not equal to 3 AND num2 is not equal to 5.

String Comparisons Reviewed

Although == does correctly test two values, such as two numbers, to see if they are equal, it has a different meaning when applied to strings.

All strings are in the `String` class, so == applied to two strings does not test to see whether the strings are equal!

To test two strings to see if they have equal values, you must use the method `equals()` or the method `equalsIgnoreCase()` rather than ==.

`string1.equals(string2)` checks to see if string `string1` is equal to string `string2`

The expressions `string1.equals(string2)` and `string2.equals(string1)` are equivalent.

The method `equalsIgnoreCase()` behaves similarly to `equals()`, except that with `equalsIgnoreCase()` the uppercase and lowercase versions of the same letter are considered the same. Bob and bob would be considered equal.

Example

```
String string1 = "Bob";
if (string1.equalsIgnoreCase("bob"))
     System.out.println("The same.");
else
     System.out.println("Not the same.");
```

This code results in the following displayed to the console:

```
The same
```

Remember that the following statement is an <u>incorrect</u> way of comparing String data types:

```
if (string1 == string2)
     System.out.println("The strings are the same");
```