Chapter 4

Making Choices

Two Java statements that allow you to create variety in your programs:

if statement:

A statement or block of statements are executed for as long as a conditional test turns out to be true;

Result is based on boolean expressions – true or false

switch statement:

A statement or several blocks of statements are executed depending on the value of an integer variable

Using Simple Boolean Expressions

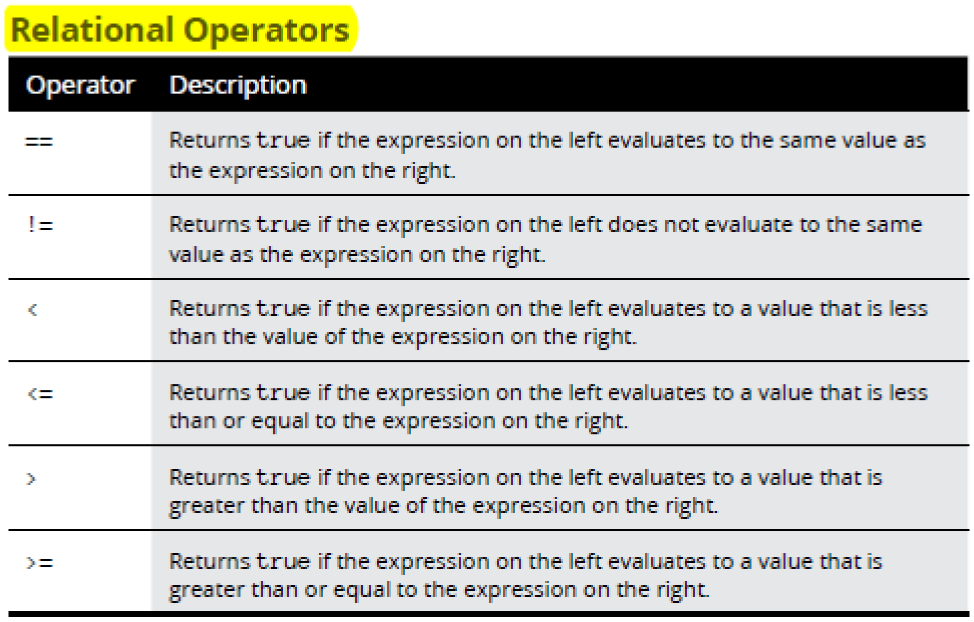
Control statements that use Boolean expressions:

if,

while,

do and

for.



Relational Operators

Operator	Description
==	Returns `true` if the expression on the left evaluates to the same value as the expression on the right.
!=	Returns `true` if the expression on the left does not evaluate to the same value as the expression on the right.
<	Returns `true` if the expression on the left evaluates to a value that is less than the value of the expression on the right.
<=	Returns `true` if the expression on the left evaluates to a value that is less than or equal to the expression on the right.
>	Returns `true` if the expression on the left evaluates to a value that is greater than the value of the expression on the right.
>=	Returns `true` if the expression on the left evaluates to a value that is greater than or equal to the expression on the right.

Boolean expression:

Expression relational-operator expression;

<u>Using if Statements</u>

Basic form of if statement:

if (Boolean-expression)

statement;

Can be a single statement and end with a semicolon;

<u>if-else statements (with an additional element)</u>

if (*boolean-expression*)

statement

else

statement

<u>Nested if statements</u>

if or if-else statement that includes another if or if-else statement is called a *nested if statement.*

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

Outer if statement testing expression-1

Inner if statement testing expression-2 and 3

Expression -1 is first evaluated, if true, expression-2 is evaluated, if both true, statement-1 is executed otherwise statement-2 is executed;

BUT if expression-1 is false, expression-3 is evaluated and statement-3 or statement-4 executed;

Special form of nested if statements called *else-if statements*

else-if statements

A common pattern for nested if statements is to have a series of if-else statements

with another if-else statement in each else part:

if (*expression-1*)

statement-1;

else if (*expression-2*)

statement-2;

else if (*expression-3*)

statement-3;

Logical Operators

! operator

AKA bang operator

AKA complement operator

Reverses the value of the Boolean expression.

Evaluates the expression first and then reverses it after evaluation;

Using the & and && operators

The & and && operators combine two boolean expressions and return true only

if both expressions are true.

If one expression is false, returns false;

The && operator evaluates the first expression and if true, is smart enough to stop when it knows what the outcome is.

Using the | and || operators

Or operators – return true or false

Return true if first or second expression is true;

Return true if both expressions are true;

<u>Example:</u>

if ((salesTotal < 1000.0) | (salesClass == 3))

commissionRate = 0.0;

Explanation: Expressions on either side of the | operator are evaluated first. Then if one of the expressions is true, the entire expression is true otherwise it is false.

In most cases, you should use the Conditional Or operator (||) instead of the regular Or operator (|), like this:

if ((salesTotal < 1000.0) || (salesClass == 3))

commissionRate = 0.0;

Explanation: Conditional Or operator stops evaluating as soon as it knows what the outcome is

<u>Combining logical operators</u>

<u>Using the Conditional Operator</u>

AKA ternary operator

AKA conditional operator

Code: *boolean-expression* ? *expression-1* : *expression-2*

Explanation: The boolean expression is evaluated first. If it evaluates to true, *expression-1* is evaluated, and the result of this expression becomes the result of the whole expression. If the expression is false, *expression-2* is evaluated, and its results are used instead.

<u>Comparing Strings</u>

Cannot make use of the double equal sign in order to compare strings as it's not a primitive data type;

Instead, make use of the equals method:

if (answer.equals("Yes"))

System.out.println("The answer is Yes.");

This method actually compares the value of the string object referenced by the variable with the string you pass as a parameter and returns a Boolean result to indicate whether the strings have the same value.

String class has another method, equalsIgnoreCase, that's also useful for comparing strings. It compares strings but ignores case.