Java Control Flow Statements

Source http://java.about.com/od/beginningjava/l/aa_control_1.htm

Java programs accomplish their tasks by manipulating program data using operators and making decisions by testing the state of program data. When a program makes a decision, it is determining, based on the state of the program data, whether certain lines of code should be executed. For example, a program may examine a variable called hasChanged to determine if it should execute a block of code that saves data to disk. If hasChanged is true, the data saving code is executed. If hasChanged is false, the data saving code is not executed.

At a simplistic level, a program executes statements sequentially in the order in which they appear. It starts with the first line of the "main()" method and ends with the last line, always moving from top to bottom. Most useful programs are vastly more complex than a simple linear sequence. Real programs contain numerous code blocks that must be executed only if certain conditions are present. Other code blocks must be executed repeatedly until certain conditions are met. A hypothetical example: A program loops through an array of employee models and increases the value stored in the salary variable by 10% for each employee that has a hire date of one year ago. (I said it was hypothetical).

Control flow statements are the tools programmers use to make decisions about which statements to execute and to otherwise change the flow of execution in a program. The four categories of control flow statements available in Java are selection, loop, exception, and branch.

Control Flow Statements

Selection and Loop

Selection and loop statements require a conditional expression that evaluates to true or false. If the conditional expression is true, the accompanying block of code is executed. If the conditional expression is false, the accompanying block of code is bypassed. Selection statements will execute the code block only once. Loop statements repeatedly execute the code block, testing the conditional statement each time. Once the conditional statement evaluates to false, the loop exits and execution of the program continues to the next statement following the loop.

Exception

Exception statements are used to gracefully handle unusual events or errors that arise while a program is running. For instance, if a program attempts to open a file but the file doesn't exist, this will cause an I/O error. Exception handling in Java is a two step process. First, an exceptional condition is detected during program execution by the JVM or application code. At this point a new exception object is created that describes the situation and is passed up the call stack using the throw statement. Each method in the call stack now has the option to handle it or let it continue to bubble up the call stack.

Branch

Branch statements explicitly redirect the flow of program execution. The most infamous branch statement in the history of computing is goto. In Java, goto is an unimplemented keyword, so it cannot be used in Java programs. In other languages, goto is used to redirect program execution to another line of code in the program by specifying a line number or other type of label. In Java, break, continue, and label: fill the role of goto, but without many of goto's negatives. return is a branching statement that redirects execution out of the current method and back to the calling method. It also returns a value to the calling method. This value is either a primitive, object, or void. void is used to return execution without returning a value. Also, whatever the return value is, its type must match the return type listed in the method declaration.

Let's take a look at control flow statements' general form:

```
statement_keyword(conditional_expression) {
  statements_to_execute
}
```

Notice there are three parts to the general form above. The statement_keyword is the Java language keyword that defines what type of control is being exerted on the statements_to_execute code block. The conditional_expression defines the conditions that must be present for statements_to_execute to be executed.

Notice statements_to_execute is wrapped in curly braces {}. If statements_to_execute consists of a single statement only, the curly braces are optional. Most Java programmers consider it to be good coding style to always include the curly braces. Leaving out curly braces can cause subtle bugs because most Java programmers expect the curly braces out of habit.

Here are some examples of control statements. We will cover all of these and more in upcoming tutorials.

```
//Example 1
if(conditional_expression) {
  statements_to_execute
}

//Example 2
if(conditional_expression) {
  statements_to_execute
}
else {
  statements_to_execute
}

//Example 3
```

```
while(conditional_expression) {
  statements_to_execute
}

//Example 4
for(int i = 0; conditional_expression; i++) {
  statements_to_execute
}
```

*~~~ End of Article ~~~*