Lesson 16: The *for*-Loop

One of the most important structures in Java is the "*for*-loop". A loop is basically a block of code that is **repeated** with certain rules about how to start and how to end the process.

Simple example:

Suppose we want to sum up all the integers from 3 to 79. One of the statements that will help us do this is:

```
sum = sum + j;
```

However, this only works if we repeatedly execute this line of code, …first with *j = 3,* then with *j = 4, j = 5*, …and finally with *j = 79*. The full structure of the *for*-loop that will do this is:

```
int j = 0, sum = 0;
for (j = 3; j <= 79; j++)
{
     sum = sum + j;
     System.out.println(sum);
     //Show the progress as we iterate thru the loop.
}
System.out.println("The final sum is " + sum);
// prints 3157
```

Three major parts:

Now let's examine the three parts in the parenthesis of the *for*-loop.

Initializing expression:

j = 3 If we had wanted to start summing at 19, this part would have read, *j = 19*.

Control expression:

j <= 79 We continue looping as long as this *boolean* expression **is *true***. In general this expression can be **any** *boolean* expression. For example, it could be:

```
count == 3     s + 1 < alphB       s > m +19
```
etc.

Warning:

There is something really bad that can happen here. You must write your code so as to insure that this control statement will eventually become *false,* thus causing the loop to terminate. Otherwise you will have an endless loop which is about the worst thing there is in programming.

Step expression:

j++ This tells us how our variable changes as we proceed through the loop. In this case we are incrementing *j* each time; however, other

possibilities are:

```
j-- j = j + 4 j = j * 3 etc.
```

For our example above, exactly when does the increment …*j*++ occur? Think of the step expression being at the bottom of the loop as follows:

```
for (j = 3; j <= 79; . . . )
{
     … some code …
     j++;
     // Just think of the j++ as being the last line
     // of code inside the
} //braces.
```

Special features of the for-loop:

The *break* command:

If the keyword ***break*** is executed inside a *for*-loop, the loop is immediately exited (regardless of the control statement). Execution continues with the statement immediately following the closing brace of the *for*-loop.

Declaring the loop variable:

It is possible to declare the loop variable in the initializing portion of the parenthesis of a *for*-loop as follows:

```
for (int j = 3; j <= 79; j++)
{
     . . .
}
```

In this case the **scope** of *j* is confined to the interior of the loop. If we write *j* in statement outside the loop (without redeclaring it to be an *int*), it won't compile. The same is true of any other variable declared inside the loop. Its scope is limited to the interior of the loop and is not recognized outside the loop as is illustrated in the following code:

```
for (j = 3; j <= 79; j++)
{
     double d = 102.34;
     . . .
}
System.out.println(d);
//won't compile because of this line
```

No braces:

If there is only **one line of code** or just one basic structure (an *if*-structure or another loop) inside a loop, then the braces are unnecessary. In this case it is still correct (and highly recommended) to still have the braces…but you **can** leave them off.

```
for (j = 3; j <= 79; j++) sum = sum + j;

is equivalent to

for (j = 3; j <= 79; j++)
{ sum = sum + j; }
```

When the loop finishes:

It is often useful to know what the loop variable is after the loop finishes:

```
for (j = 3; j <= 79; j++)
{
     . . . some code . . .
}
System.out.println(j); //80
```

On the last iteration of the loop, *j* increments up to 80 and this is when the control statement *j <= 79* finally is *false*. Thus, the loop is exited.

Nested loops:

"Nested loops" is the term used when one loop is placed inside another as in the following example:

```
for(int j = 0; j < 5; j++)
{
     System.out.println("Outer loop");
     // executes 5 times
     for(int k = 0; k < 8; k++)
     {
          System.out.println("...Inner loop");
          // executes 40 times
     }
}
```

The inner loop iterates eight times for **each** of the five iterations of the outer loop. Therefore, the code inside the inner loop will execute 40 times.

Warning:

A very common mistake is to put a semicolon immediately after the parenthesis of a *for* loop as is illustrated by the following code:

```
for (j =3; j <= 79; j++);
{
     // This block of code is only executed once
     // because of the inappropriately
     // placed semicolon above.
     . . . some code . . .
}
```