For Loop

Now we will look at a second type of loop, the for loop. Keep in mind that anything you can do with a for loop can also be done with a while loop. However, certain concepts are more clearly coded using a for loop instead of a while loop.

The basic syntax of a for loop is the following:

```
for (<init stmt>;<bool exp>;<inc. stmt>) {
        <several stmts>
}
```

If you only have one statement inside of your for loop, the braces({}) are not necessary.

Here is how to execute a for loop:

1) execute the <init stmt> - this typically initializes some variable.
2) Check the boolean expression. If it is true, execute all the statements in the body of the for loop. If the expression is false, skip all of the statements in the for loop, and also skip over steps 3 and 4.
3) Execute the <inc. stmt> - this typically increments or decrements the value of the variable initialized by the <init stmt>
4) Go back to step 2.

It will be easy to see how a for loop works with an example:

```
for (int i=0; i < 10; i=i+1) {
   System.out.println(“Line Number: ” + (i+1));
}
```

To execute this code, you first declare and initialize i to 0. Then you check to see that i is less than 10. Since it is, you go ahead and execute the body of the for loop, which will print out the following:

Line Number: 1

Next, the increment statement, i=i+1 will be executed. This changes the value of i from 0 to 1. Then we will check the boolean statement again. Since i is still less then 10, we continue with the loop, printing out:

Line Number: 2.

This continues until we print out the following:

Line Number: 10.

At this point, i has a value of 9, and then we execute the increment statement. Subsequent to that, we will have i=10. When we check whether or not i<10, we find the boolean expression is false, and then we are done executing the for loop.

It is worth it to note that setting your initial variable to 0 is the most commonly used practice when using a for loop, however, it does not mean that we are limited to using 0 as an initial value. It is perfectly fine to put;

for(int i = 1; i <= 10; i++)

instead of

for(int i = 0; i < 10; i++)

However, Due note that in order for your loop to actually loop 10 times you would need to either change the Boolean statement of:

i < 10

to :

i <=10 or i < 11

This is because the loop will terminate on 10 if the condition stays the same and because you moved the iteration of the loop forward by 1 you must account for that in your Boolean expression by either making it less than or equal to 10 which means on the 11^{th} run time the loop will end to end it as long as the loop runs less than 11 times.

As I mentioned before, any for loop can be turned into a while loop. Here is an essentially equivalent while loop for the preceding example:

```
int i=0;
while (i<10) {
   System.out.println(“Line Number: ” + (i+1));
   i = i+1;
}
```

(The only minor difference here is that after the while loop is over, the variable i will still be declared, but in our for loop it will not be. The deals with the scope of variables – we will talk about that in detail later.)

The reason people typically prefer the for loop to the while loop for a situation like the one in the example is that it is very easy for someone to read the loop and determine it will run exactly 10 times. All the pertinent information is in the one line, in a concise form. There will be many situations you will want to repeat some code a specific number of times. Let’s say n times. The following for loop allows you to do that:

```
for (int i=0; i < n; i=i+1) {
  <whatever code you want to repeat n times>
}
```

Sum of Squares Example

Consider trying to find the following sum:

$1^2 + 2^2 + ... + 100^2$.

How can we use a for loop to print out this sum in a relatively simple manner?

Since we are adding values up, we will need an accumulator variable. Let's call this sum. Since we are doing something exactly 100 times, (adding into our accumulator variable), we can use a for loop. With this in mind we produce the following code segment:

```
int sum = 0;
for (int i=1; i < 100 ; i=i+1) {
   sum = sum + i*i;
}
System.out.println("The sum of the first 100 squares is " + sum);
```

Another note: it seems like we are incrementing variables a lot. In fact, this statement is so common in Java, there is a shorthand for it. Anytime you want to add 1 to a variable, instead of doing the standard

```
i = i + 1;
```

You can do the shorthand:

```
i++;
```

The ++ translates to incrementing the variable by 1. Similarly, if you want to subtract 1 from a variable you can do it as follows:

```
i--;
```

However, let's say for example, that you want to add a different value to a variable, then you could do the following:

```
i+=2;
```

This gets translated to

i = i + 2;

Similarly,

i–=2;

gets translated to

i = i – 2;

You are certainly not required to use any of these shorthand statements, but they are used normally by programmers and you must be able to understand them when reading other peoples' code.

As one final note, it is possible to increment and decrement by multiples however you cannot do this in a direct manner. Incrementing by multiples is not commonly used however is still possible. For example:

i = i * 3

is translated into

i*=3

Compounding Interest Example

The most common form of this will be found if you are multiplying similar variables together. If you were trying to calculate the interest on an investment over a certain number of years you would have a fixed interest rate of, say, 1.05 a year and multiply that times the money you have in the account each year. This type of program would look like:

```
double total = 1000.00;
int years = 10;
double interest = 1.05

for(int i = 0; i < years; i++ )
{
        total *= interest;
}

System.out.println("The full investment yield would be $" + total);
```

The first iteration would get you:

1000 * 1.05 = 1050

The second iteration would then become:

1050 * 1.05 = 1102.50

This would continue until you had your last year of interest included:

1551.33 * 1.05 = 1628.89;
The full investment yield would be $1628.89

This is a nice handy little method that doesn't always see much use but when it does can save you many lines of code and make things much easier to read and decipher.

Perfect Number Example

A perfect number is one such that the sum of its proper divisors equals itself. A proper divisor of an integer is a number that divides into it evenly and is strictly less than the number. For example, 6 is a perfect number. Its proper divisors are 1, 2 and 3 and 1 + 2 + 3 = 6. The next perfect number is 28, which has proper divisors 1, 2, 4, 7 and 14.

First, we must figure out how to check if one number is a divisor of another number. To do this, we must use the mod operator. If one integer divides evenly into another, then the corresponding mod operation equals zero. (Namely, n%i is equal to 0 if and only if i is a divisor of n.)

Now, we must simply check each possible number smaller than n is a divisor. If it is, we should add that number to an accumulator variable. At the end, we can just check to see if the accumulator variable is equal to the number or not. Here is the full Java program:

```
public class Perfect {
	public static void main( String args[] )
	{
		// Get the user input.
		Scanner stdin = new Scanner(System.in);
		int n = stdin.nextInt();

		int sumdiv = 0;

		// Try each integer, add it if it's a divisor.
		for (int i=1; i<n; i++)
			if (n%i == 0)
				sumdiv += i;

		// Check the sum and print the result accordingly.
		if (sumdiv == n)
			System.out.println(n+" is a perfect number.");
		else
			System.out.println(n+" is NOT a perfect number.");
	}
}
```