Introduction to Recursion

In the science of computer programming it is quite possible that recursion may be the single hardest topic to both understand and master. In this lesson the conceptual basis for recursion will be introduced and a very simple mathematical problem will be solved using traditional loops **and** recursion to demonstrate the difference between the two programming algorithms.

A common definition of recursion in programming is the process of solving a problem by successively solving smaller instances of the same problem.

For our purposes at this stage recursion will take the place of the traditional loop structures `for` and `while`.

First let us examine a traditional solution to a simple math problem. A factorial is written in the form $n!$ where n is a non-negative integer. To solve $n!$ when n=4, one solves $4\times3\times2\times1 = 24$.

How would this be solved in a JAVA method?

The following method solves $n!$ using a simple for loop and accumulator. If n=5, the loop iterator variable `i` starts with the value `5`, and iterates down to `1`. In each iteration of the loop the accumulator `Result` is multiplied by the current value of `i` so one could write:

`Result=4*3*2*1;`

Now let's examine the same method using recursive techniques.

The first thing you should notice is that this solution doesn't have a traditional loop. The second item of note should be the line of code:

`return n*factorialR(n-1);`

It is this line of code that makes this a recursive algorithm!

```
public static int factorial(int n) {
  int result=1;
  for (int i=n; i>=1; i--) {
    result*=i;
  }
  return result;
}
```

```
public static int factorialR(int n) {
  if (n>1) {
    return n*factorialR(n-1);
  } else {
    return 1;
  }
}
```

Lets follow this code for the method call *factorialR*(4):