Memory allocation to methods

When a method is called, the state of the method is placed on the call stack along with the information about where it was called from. This tells the run-time where it should return to when the current method finishes executing. Each recursive call pushes a new stack frame. A stack frame consists of information such as the return address, argument variables, and local variables related to that method call.

When a method calls itself, the new method call gets added to the top of the call stack and execution of the current method pauses while the recursive call is being processed. When the base case is reached the stack frames start popping from the stack until the stack becomes empty.

Example

Computing Factorials

What is a Factorial?

A factorial is the product of an integer and all the positive integers below it. It is denoted by the symbol: !

For example, 4! (read as four factorial) is denoted as follows:

$4! = 4 \times 3 \times 2 \times 1 = 24$

$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

Code

```
1  class Factorial {
2      // Recursive function
3      private static int factorial(int n) {
4        // Base case
5        if (n == 1) {
6          return 1;
7        }
8        // Recursive case
9        else {
10         return (n * factorial(n-1));
11       }
12     }
13
14     public static void main( String args[] ) {
15         // Calling from main
16         int result = factorial(5);
17         System.out.println("Factorial of 5 is: " + result);
18     }
19 }
```

Run Save Reset

We will now briefly discuss the two main parts of a recursive method, the base case and the recursive case, implemented in the code above.

The Base Case

We have defined the base case on **line 5** where it states that when the variable `n` equals 1, the method should terminate and start popping frames from the stack.

The Recursive Case

This is similar to the formula given above for the factorial. For example, in case of 3 factorial, $3! = 3 \times 2 \times 1 = 6$ we multiply the original number `n` with the integer that precedes it, `n-1`, and the recursive call, `factorial(n-1)`, is made until `n` reaches 1.

Visualizing through Stack

The following illustration may help you to understand the code through a stack: