Stack

So the stack is used to keep track of a method's data from every other method call. When a method is called, .NET creates a container (a stack frame) that contains all of the data necessary to complete the call, including parameters, locally declared variables and the address of the line of code to execute after the method finishes. For every method call made in a call tree (i.e. one method that calls another, which calls another, and so on) stack containers are stacked on top of each other. When a method completes, its container is removed from the top of the stack and the execution returns to the next line of code within the calling method (with its own stack frame). The frame at the top of the stack is always the one used by the current executing method.

Using this simple mechanism, the state of every method is preserved in between calls to other methods, and they are all kept separate from each other.

In Listing 1.1, Method1 calls Method2, passing an int as a parameter.

```
1  void Method1()
2  {
3       Method2(12);
4       Console.WriteLine("Goodbye");
5  }
6  void Method2(int testData)
7  {
8     int multiplier=2;
9     Console.WriteLine("Value is " + testData.ToString());
10    Method3(testData * multiplier);
11  }
12  void Method3(int data)
13  {
14     Console.WriteLine("Double " + data.ToString());
15  }
```

Listing 1.1: Simple method call chain.

To call Method2, the application thread needs first to record an execution return address which will be the next line of code after the call to Method2. When Method2 has completed execution, this address is used to call the next line of code in Method1, which is Line 4. The return address is therefore put on the stack.

Parameters to be passed to Method2 are also placed on the stack. Finally, we are ready to jump our execution to the code address for Method2.

If we put a break point on Line 13 the stack would look something like the one in Figure 1.1. Obviously this is a huge simplification, and addresses wouldn't use code line numbers for return addresses, but hopefully you get the idea.

In Figure 1.1, stack frames for Methods 1, 2, and 3 have been stacked on top of each other, and at the moment the current stack frame is Method3, which is the current executing method. When Method3 completes execution, its stack frame is removed, the execution point moves to Method2 (Line 9 in Listing 1), and execution continues.

A nice simple solution to a complex problem, but don't forget that, if your application has multiple threads, then each thread will have its own stack.

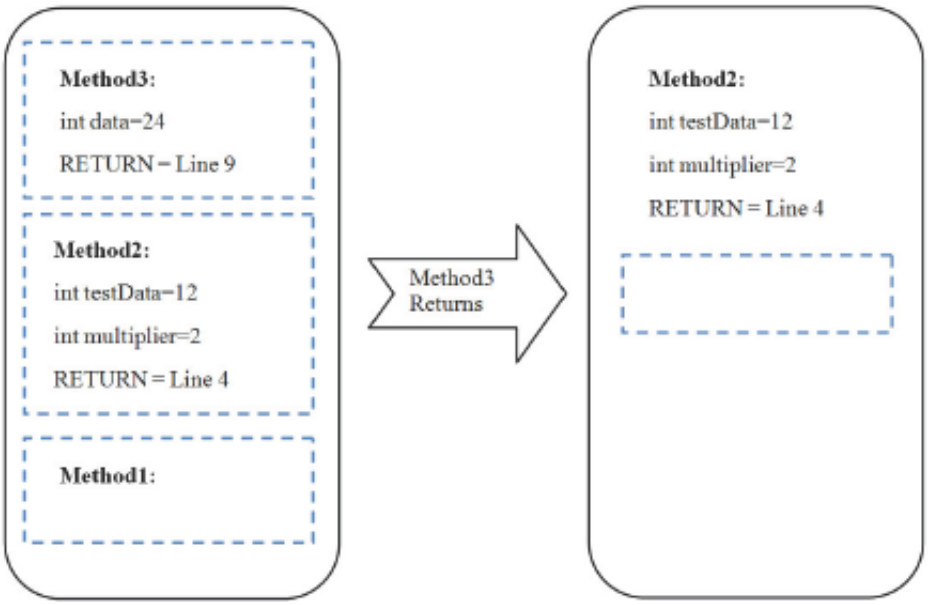

Figure 1.1: Example of a stack frame.