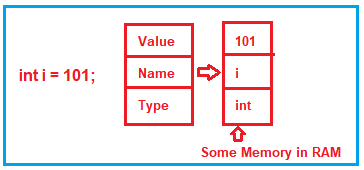
**What happens internally when we declare a variable in .NET Application?**

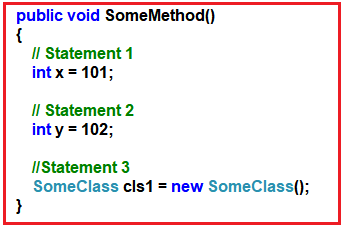
When we declare a variable in a .NET application, it allocates some memory in the RAM. The memory that it allocates in RAM has three things are as follows:

1. **Name of the variable,**
2. **The data type of the variable, and**
3. **Value of the variable.**



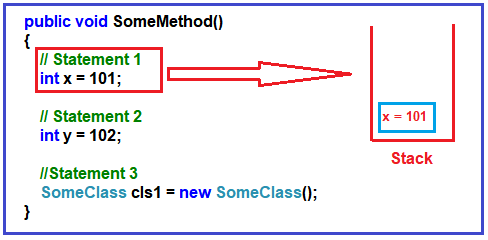
But depending on the data type (i.e. depending on the value type and reference type ), the memory may be allocated either in the stack or in the heap memory.

Consider bellow image :



##### ****Statement1:****

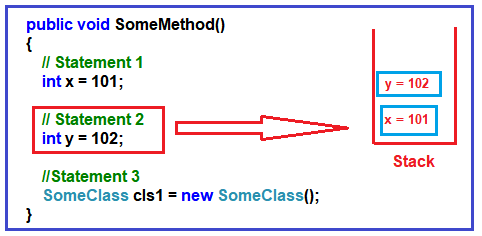
When the first statement is executed, the compiler allocates some memory in the stack. The stack memory is responsible for keeping track of the running memory needed in your application.



##### ****Statement2:****

When the second statement is executed, it stacks this memory allocation (memory allocation for variable y) on top of the first memory allocation (memory allocation for variable x).

You can think about the stack as a series of plates or dishes put on top of each other.

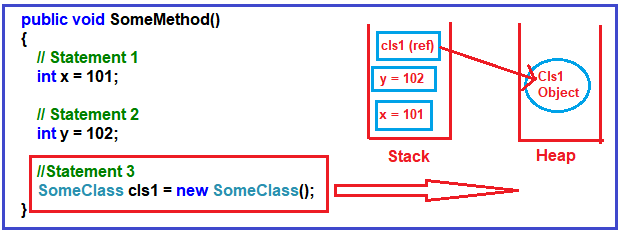


##### ****Statement3:****

In the 3rd statement, we have created an object of SomeClass.

When the 3rd statement is executed, it internally creates a pointer on the stack memory and the actual object is stored in a different memory location called Heap memory.

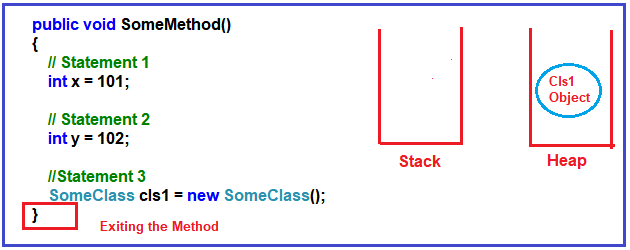
The heap memory location does not track running memory. Heap is used for dynamic memory allocation.



##### ****What happens when the method completes its execution?****

When the three statements are executed, then the control will exit from the method. When it passes the end control i.e. the end curly brace “}”, it will clear all the memory variables which are created on the stack.

It will de-allocate the memory in a ‘LIFO’ fashion from the stack. For a better understanding please have a look at the below image.



It will not de-allocate the heap memory.

Later, the heap memory will be de-allocated by the garbage collector.

##### ****Why do we have two types of memory?****

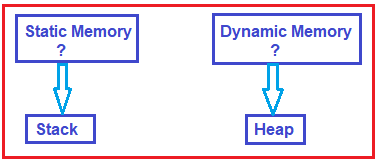
As we know, in C#, the primitive data types such as int, double, bool, etc. just hold a single value.

On the other hand, the reference data types or object data types are complex i.e. an object data type or reference data type can have reference to other objects as well as other primitive data types.

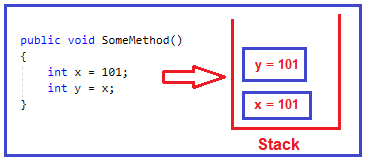
So, the reference data type holds references to other multiple values, and each one of them must be stored in memory.

Object types need dynamic memory while primitive data types need static memory.

Please have a look at the following image for a better understanding.



##### ****Understanding Value Type in C#:****



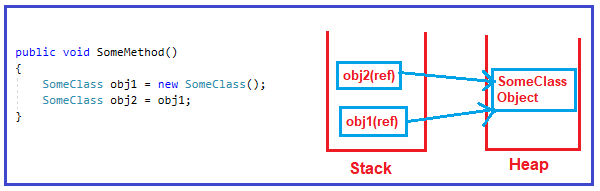
In .NET, when we assign one integer variable value to another integer variable, then it creates a completely different copy in the stack memory that’s what you can see in the above image.

So, if you change one variable value, then the other variable will not be affected.

In .NET these kinds of data types are called ‘Value types’.

So, bool, byte, char, decimal, double, enum, float, long, sbyte, int, short, ulong, struct, uint, ushort are examples of value types.

##### ****Understanding Reference Type in C#:****



In this case, when you change one of them, the other object is also gets affected.

These kinds of data types are termed as ‘Reference types’ in .NET.

So, **class**, **interface**, **object**, **string**, and **delegate** are examples of Reference Types.