

Newbie Question

Source:

<http://www.tech-archive.net/Archive/VB/microsoft.public.vb.general.discussion/2006-04/msg01894.html>

- *From:* "GFM GToeroe" <gfm@xxxxxxxxxx>
 - *Date:* 21 Apr 2006 01:16:53 -0700
-

Hi!

I'm new to vb.net (and to programming as well) and last night I tried to make my own picture of the difference of value and reference types.

What I find out is that the main difference is how the CLR handles the memory for this types. If a new instance of a value type is generated then for this instance an address exist which points to a formatted chunk of data on the stack. This adress which points on the stack can be regarded as the name of the instance which is connected to the name of the instance in the source code.

When ever the program execution leaves the scope of the instance the used memory on the stack will immediatly be reclaimed and the instance no longer exists.

If a new instance of a reference type is generated then the same happens with the stack with the difference that the data on the stack is an adresse which points to the heap where the real data of the instance is. If the instance variable is set to nothing or if the instance is out of scope then the used memory on the stack is again immediatly reclaimed but the instance still lives due to the data on the heap. First if the garbage collector decides to clean up instances which no longer are reachable by the application then the instance will be removed totally from the memory.

If an instance is passed to an implementation then one has to distinct the four cases:

- 1) ByVal Value Type
- 2) ByVal Reference Type
- 3) ByRef Value Type
- 4) ByRef Reference Type

If an instance is passed by value semantics then a copy of the data on the stack is made and the adress of this copy is passed to the implementation while by reference semantics pass the adress of the original data on the stack. If the instance is a value type then the

Newbie Question

implementation has no access to the original instance at all as the implementation only get the name (address) of the copy on the stack.

But if the instance is a reference type then again the implementation gets a address of the copy on the stack but the data on the stack points to the real data on the heap. So this way the implementation has access to the data of original instance. What the implementation can do not is to change the target of the reference of the instance. That means it can not change the data on the stack of the original instance. The pointer to the place to the heap can not be changed.

Last but not least one have the case where a reference type is passed by reference. Then the implementation not only can modify the data on the heap. As the implementation gets the original address on the stack where the location of the heap's data is found the implementation can change this value and therefore can the reference variable point to a totally different place on the heap. That means the reference variable can point to a different instance.

Correct?

If so then I have another question: If a reference type is passed by reference to a procedure and within this procedure a new instance of the same type is generated and assigned to the passed reference variable what happens to the instance on the heap if the program execution leaves the procedure? I think the original instance on the heap is now in danger to be killed by the garbage collector as no reference points to it anymore:

```
Sub Main()  
....  
Dim a As New Object  
....  
Test(a)  
.....  
End Sub  
....  
Sub Test(ByRef a As Object)  
Dim b As New Object  
a=b  
End Sub
```

Correct?

GFM GToeroe

.