**Part 50 - C# Tutorial - Internal and Protected Internal Access Modifiers**

In this part we will understand internal and protected internal access modifiers. [If you want to learn about private, public and protected access modifiers, please watch this video](http://csharp-video-tutorials.blogspot.co.uk/2012/07/part-49-c-tutorial-access-modifiers.html)  
  
**Internal:**A member with internal access modifier is available any where with in the containing assembly. It's a compile time error to access, an internal member from outside the containing assembly.  
  
**To understand inernal access modifier, we need 2 assemblies. To generate, the 2 assemblies we need follow these steps.**  
1. Open Solution Explorer (From the Viiew menu, select Solution Explorer)  
2. From the Solution Explorer, right click on the project and select Add -> New Project  
3. In the Add New Project Dialog Box, Select Visual C# from the Installed Templates section, and select Class Library from the center pane, and then enter AssemblyOne for the Name of the project and click OK.  
4. Follow steps 2 & 3, to create AssemblyTwo project.  
5. If you have followed these steps correctly, you should now see three projects in the soultion explorer.

**Now, if we build the solution we should have 3 assemblies generated. Two dll's and one exe. To locate the physical assembly follow these steps.**  
1. Right click on AssemblyOne project, in solution explorer and select Open Folder in Windows Explorer.  
2. Open bin folder  
3. Now open Debug folder  
4. In the Debug folder you should see AssemblyOne.dll, which is the physical assembly.  
 **Copy and paste the following code in class1.cs file of AssemblyOne project**using System;  
namespace AssemblyOne  
{  
    public class AssemblyOneClassI  
    {  
        internal int ID = 999;  
    }

  public class AssemblyOneClassII  
    {  
        public void Test()  
        {  
            AssemblyOneClassI instance = new AssemblyOneClassI();  
            // Can access inetrnal member ID, AssemblyOneClassII and AssemblyOneClassI  
            // are present in the same assembly             
            Console.WriteLine(instance.ID);  
        }  
    }  
}  
  
In this example, **AssemblyOneClassI**has an **internal member ID**. We can access ID member from **AssemblyOneClassII**, because this class is also present in the same assembly as **AssemblyOneClassI**.  
  
**Now, Copy and Paste the following code, in Class1.cs of AssemblyTwo project.**using System;  
using AssemblyOne;  
namespace AssemblyTwo  
{  
    public class AssemblyTwoClassI  
    {  
        public void Test()  
        {  
            AssemblyOneClassI instance = new AssemblyOneClassI();  
            //Console.WriteLine(instance.ID);         
        }  
    }  
}  
  
**Note: You will get 3 compiler errors at this point. To solve this we need to add an assembly reference. Follow these steps.**  
1. Expand References folder under AssemblyTwo project, from Solution Explorer.  
2. Right Click on References folder and select Add Reference  
3. From the Add Reference dialog box, select Projects tab  
4. From the list, select AssemblyOne project and click OK.  
At, this point all the compiler errors should have gone.

**Uncomment the following line from Class1.cs file from AssemblyTwo project and rebuild the solution.**  
Console.WriteLine(instance.ID);  
  
Now, you will get a compiler error stating 'AssemblyOne.AssemblyOneClassI' does not contain a definition for 'ID' and no extension method 'ID' accepting a first argument of type 'AssemblyOne.AssemblyOneClassI' could be found (are you missing a using directive or an assembly reference?).  
  
This is because, **AssemblyTwoClassI** is not present in **AssemblyOne** assembly and hence cannot access the **internal ID member**defined in **AssemblyOne**assembly. **This proves that internal members are only accessible with in the same assembly. Code outside of the containing assembly cannot access internal members.**  
  
**Protected Internal:**Protected Internal members can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly. It is a combination of protected and internal. If you have understood protected and internal, this should be very easy to follow.  
  
Now, change the **access modifier**from **internal**to **protected internal**for **ID member**in **AssemblyOneClassI**of **class1.cs** file in **AssemblyOne**project.  
internal int ID = 999; to protected internal int ID = 999;  
  
**Finally modify the code in Class1.cs file in AssemblyTwo project as shown below.**using System;  
using AssemblyOne;  
namespace AssemblyTwo  
{  
    // Make AssemblyTwoClassI inherit from AssemblyOneClassI     
    public class AssemblyTwoClassI : AssemblyOneClassI  
    {  
        public void Test()  
        {  
            AssemblyOneClassI instance = new AssemblyOneClassI();  
            // Access the base class member using the base keyword             
            Console.WriteLine(base.ID);  
        }  
    }  
}

So, this shows protected internal **ID** member, defined in **AssemblyOne**is accessible in **AssemblyTwo**.

