What does the modifier protected internal in C# mean?

The **Protected Internal** can be accessed by Members of the Assembly or the inheriting class, and of course, within the class itself.   
  
In VB.NET, the equivalent of protected internal is **protected friend**.  
  
The access of this modifier is limited to the current assembly or the types derived from the defining class in the current assembly. To know more on different types of access specifiers, [Click Here](http://www.dotnetuncle.com/Difference/5_public_private_protected_friend_internal.aspx)

Can multiple data types be stored in System.Array?

So whats an **array** all about? An array is a collection of items of the same type, that is grouped together and encompassed within an array object. The array object, or the **System.Array** object to be precise, is derived from the System.Object class. It is thus, stored in the form of a heap in the memory.   
An array may be of single dimensional, multi-dimensional or jagged (a jagged array means an array within an array).   
  
A group of items when assigned values within braces implicitly derive from System.Array class. See example below written in C#...

int[] testIntArray = new int[4] { 2, 3, 4, 5 };  
Object[] testObjArray = new Object[5] { 32, 22, 23, 69, 75 };

Ideally an array should contain a single data type. But still in case there is a requirement to place data of different data types in a specific array, then in such a scenario, the data elements should be declared as an **object** type. When this is done, then each element may point ultimately to a different data type. See code example below written in VB.NET...

Dim allTypes As Object() = New Object() {}  
  
'In this kind of scenario, the performance may tend to slow down, as data conversions may take place.   
'In case a value type is converted to reference type, then boxing and unboxing occurs  
'To know more on Value Types & Reference Types, [Click Here](http://www.dotnetuncle.com/Difference/25_value_reference_type.aspx)  
  
Dim studentTable(2) As Object  
studendTable(0) = "Vishal Khanna"  
studentTable(1) = 28  
studentTable(2) = #9/1/1978#  
  
'To get these values of these varying datatypes, their values are converted to their original data type  
  
Dim myAge As Integer = CInt(studentTable(1))  
Dim myBirthDay as Date = CDate(studentTable(2))

How to sort array elements in descending order in C#?

Elements of an array may not be sorted by default. To sort them in descending order, the Sort() method is first called. Next, to descend the order, call the Reverse() method.

Whats the use of "throw" keyword in C#?

The **throw** keyword is used to throw an exception programatically in C#. In .NET, there is an in-built technique to manage & throw exceptions. In C#, there are 3 keyword, that are used to implement the Exception Handling. These are the **try, catch and finally** keywords. In case an exception has to be implicitly thrown, then the **throw** keyword is used. See code example below, for throwing an exception programatically...

class SomeClass  
{  
public static void Main()  
{  
 try  
 {  
   throw new DivideByZeroException("Invalid Division Occured");  
 }  
 catch(DivideByZeroException e)  
 {  
   Console.WriteLine("Exception - Divide by Zero" );  
 }  
}  
}

Can we put multiple catch blocks in a single try statement in C#?

Yes. Multiple catch blocks may be put in a try block. See code example below, to see multiple catch blocks being used in C#.

class ClassA  
{  
public static void Main()  
{  
 int y = 0;  
try  
{  
 val = 100/y;  
 Console.WriteLine("Line not executed");  
}  
catch(DivideByZeroException ex)  
{  
  Console.WriteLine("DivideByZeroException" );  
}  
catch(Exception ex)  
{  
  Console.WritLine("Some Exception" );  
}  
finally  
{  
  Console.WriteLine("This Finally Line gets executed always");  
}  
Console.WriteLine("Result is {0}",val);  
}  
}

How to achieve polymorphism in C#?

In C#, polymorphism may be achieved by overloading a function, overloading an operator, changing the order of types, changing the types using the same name for the member in context. To see some code examples of polymorphism, [Click Here](http://www.dotnetuncle.com/OOPS/13_polymorphism.aspx).

What is polymorphism?

**Polymorphism** means allowing a single definition to be used with different types of data (specifically, different classes of objects). For example, a polymorphic function definition can replace several type-specific ones, and a single polymorphic operator can act in expressions of various types. Many programming languages implement some forms of polymorphism.   
  
The concept of polymorphism applies to data types in addition to functions. A function that can evaluate to and be applied to values of different types is known as a polymorphic function. A data type that contains elements of different types is known as a polymorphic data type.   
  
Polymorphism may be achieved by **overloading a function, overloading an operator, changing the order of types, changing the types** using the same name for the member in context.

Example:  
Public Class Calc  
{  
public void fnMultiply(int x, int y)  
{ return x \* y; }  
public void fnMultiply(int x, int y, int z)  
{ return x \* y \* z; }  
}  
**...  
...**Calc obj;  
int Result;  
Result = obj.fnMultiply(2,3,4); // The second fnMultiply would be called  
Result = obj.fnMultiply(3,4); // The first fnMultiply would be called  
//Here, the call depends on the number of parameters passed, polymorphism is achieved using overloading

How to achieve inheritance in C#?

When a class derives features of another class, this feature is called **Inheritance**.   
  
To know more on Inheritance, [Click Here](http://www.dotnetuncle.com/OOPS/11_inheritance.aspx)

What is inheritance?

**Inheritance** - is the concept of passing the traits of a [class](http://www.dotnetuncle.com/Difference/14_class_object.aspx) to another class.   
  
A class comprises of a collection of types of encapsulated instance variables and types of methods, events, properties, possibly with implementation of those types together with a constructor function that can be used to create objects of the class. A class is a cohesive package that comprises of a particular kind of compile-time metadata. A Class describes the rules by which objects behave; these objects are referred to as "instances" of that class. ([Reference](http://en.wikipedia.org/wiki/Inheritance_(programming)))  
  
Classes can inherit from another class. This is accomplished by putting a colon after the class name when declaring the class, and naming the class to inherit from—the base class—after the colon. ([Reference](http://msdn2.microsoft.com/en-us/library/ms173149(VS.80).aspx))

Compare Session state and Viewstate in .NET?

**Session state** is the feature of ASP.NET based web applications using which values of a variable may be persisted and then posted to another page.  
  
**Viewstate** property of a page or a control, or a viewstate object for a variable value, may also be created to persist its value across a postback. However, the viewstate value are for page level, i.e. they cannot be posted across to another page.   
  
To know more on Viewstate, [Click Here](http://www.dotnetuncle.com/Aspnet/75_viewstate.aspx)  
  
To know more on Session state, [Click Here](http://www.dotnetuncle.com/Aspnet/72_global_variables.aspx) The datagrid or the gridview has a property called **AutoGenerateColumns**. Make this property **false**. Once false, the columns may then be added to the gridview manually.

How to add gridview/datagrid columns manually? Autogenerate in .NET?

The datagrid (of ASP.NET 1.1) and gridview (of ASP.NET 2.0) has a property called **AutoGenerateColumns**. This property is set to **false**. After this, columns may be set manually. Sample source below...

<asp:GridView ID="GridView1" DataSourceID="SqlDataSource1" AutoGenerateColumns="False" runat="server">  
<Columns>  
<asp:BoundField HeaderText="ID" DataField="au\_id" ReadOnly="true" />  
<asp:BoundField HeaderText="Last Name" DataField="au\_lname" />  
<asp:BoundField HeaderText="First Name" DataField="au\_fname" />  
<asp:BoundField HeaderText="Phone" DataField="phone" />  
<asp:BoundField HeaderText="Address" DataField="address" />  
</Columns>  
</asp:GridView>  
<asp:SqlDataSource ID="SqlDataSource1" runat="server"   
SelectCommand="SELECT [au\_id], [au\_lname], [au\_fname], [phone], [address] FROM [authors]"  
ConnectionString="<%$ ConnectionStrings:Pubs %>" />

How to add a ReadOnly property in C#?

**Property** - A property is an entity that describes the features of an object. A property is a piece of data contained within a class that has an exposed interface for reading/writing. Looking at that definition, you might think you could declare a public variable in a class and call it a property. While this assumption is somewhat valid, the true technical term for a public variable in a class is a field. The key difference between a field and a property is in the inclusion of an interface. We make use of Get and Set keywords while working with properties. We prefix the variables used within this code block with an underscore. Value is a keyword, that holds the value which is being retrieved or set. See code below to set a property as **ReadOnly**. If a property does not have a set accessor, it becomes a ReadOnly property.

public class ClassA  
{  
private int length = 0;  
public ClassA(int propVal)  
{  
length = propVal;  
}  
public int length  
{  
get  
{  
return length;  
}  
}  
}

How to prevent a class from being inherited? Sealed in C#?

In order to prevent a class in C# from being inherited, the **sealed** keyword is used. Thus a sealed class may not serve as a base class of any other class. It is also obvious that a sealed class cannot be an abstract class. Code below...

sealed class ClassA  
{  
public int x;   
public int y;  
}

No class can inherit from ClassA defined above. Instances of ClassA may be created and its members may then be accessed, but nothing like the code below is possible...

class DerivedClass: ClassA {} // Error

To know whats a sealed method in C# [Click Here](http://www.dotnetuncle.com/csharp/131_sealed_method.aspx)

How to inherit the class, but not the method inside it in C#? What is a sealed method in C#? Can a method in C# be sealed? How to create a sealed method in C#?

If a method is not to be inherited, but the class is, then the method is sealed. It becomes a **sealed method** of a class. It is important to note here that in C#, a method may not be implicitly declared as sealed. This means that a method cannot be sealed directly. A method in C# can be sealed only when the method is an overriden method. Once the overriden method is declared as sealed, it will not be further overriding of this method. See code sample below, where an overriden method is sealed.

class SomeClass   
{  
   public int a;   
   public int b;  
  
  public virtual void SomeMethod()  
  {  
    Console.WriteLine("This is a virtual method");  
  }  
}  
class SomeClass : DerivedClass  
{  
   public override sealed void SomeMethod()  
   {  
     Console.WriteLine("This is a sealed method");  
   }   
}

Can we inherit multiple interfaces in C#?

Yes. Multiple interfaces may be inherited in C#.   
  
Note that when a class and multiple interfaces are to be inherited, then the class name should be written first, followed by the names of the interfaces. See code example below, on how to inherit multiple interfaces in C#.

class someclass : parentclass, IInterface1, IInterface2  
{  
//...Some code in C#  
}

What are the different ways of overloading methods in C#? What is function overloading in C#?

Before knowing the different methods of overloading in C#, lets first clear out what exactly overloading is. **Overloading** is the OOPs concept of using a method or a class in different styles by modifying the signature of the parameters in it. To know more on overloading, [Click Here](http://www.dotnetuncle.com/OOPS/17_overloading.aspx).  
  
In order to achieve overloading, there may be several techniques applied. There are different types of overloading like Operator Overloading, Function Overloading etc.   
  
Function overloading may be achieved by changing the order of parameters in a function, by changing the types passed in the function, and also by changing the number of parameters passed in a function. See code sample below to see types of overloading.

//Define a method below  
public void fnProcess(int x, double y)  
{  
......  
}  
//change the order of parameters  
public void fnProcess(double x, int y) {  
//.......Some code in C#  
}  
//Similarly, we may change the number of parameters in fnProcess   
//and alter its behaviour

|  |  |
| --- | --- |
| **How to call a specific base constructor in C#?**  What is a **Constructor?** - It is a method that gets invoked when an instance of a class is created. In case a class has plenty of constructors, i.e. there are plenty of overloaded constructors, in such a scenario, it is still possible to invoke a specific base constructor. But there is a special way, as explicit calls to a base constructor is not possible in C#. See code below:  public class dotnetClass { public dotnetClass() { // The constructor method here } // Write the class members here }  //Sample code below shows how to overload a constructor public class dotnetClass { public dotnetClass() { // This constructor is without a parameter // Constructor #1  }  public dotnetClass(string name) { // This constructor has 1 parameter. // Constructor #2  } }  This constructor gets executed when an object of this class is instantiated. This is possible in C#. Calling a specific constructor will depend on how many parameters, and what parameters match a specific constructor. Note that a compile time error may get generated when 2 constructors of the same signature are created.   We may make use of the **this** keyword and invoke a constructor. See code example below.  this("some dotnet string"); //This will call Constructor #2 above  What is the use of the **base** keyword. Suppose we have a derived class named dotnetderivedclass. If this derived class is to invoke the constructor of a base class, we make use of the base keyword. See code example below on how to use a base keyword to invoke the base class constructor.  public class dotnetClass { public dotnetClass() { // The 1st base class constructor defined here }  public dotnetClass(string Name) { // The 2nd base class constructor defined here } }  public class dotnetderivedclass : dotnetClass // A class is being inherited out here { public dotnetderivedclass() { // dotnetderivedclass 1st constructor defined here }  public dotnetderivedclass(string name):base(name) { // dotnetderivedclass 2nd constructor defined here } }  Note that we have used the base keyword in the sample code above. The sequence of execution of the constructors will be as follows: public dotnetClass() method -> public dotnetderivedclass() method  The above sequence triggers when there is no initializer to the base class, and thus it triggers the parameterless base class constructor. The other base class constructor may also get invoked when we pass a parameter while defining it.  **What is a static constructor in C#?**  To know more on what is a static constructor in C#, [Click Here](http://www.dotnetuncle.com/OOPS/27_static_constructor.aspx). |  |

|  |  |
| --- | --- |
|  |  |

What is a static constructor?

**Static Constructor** - It is a special type of constructor, introduced with C#. It gets called before the creation of the first object of a class(probably at the time of loading an assembly). See example below.

Example:  
public class SomeClass()  
{  
  static SomeClass()  
  {  
     //Static members may be accessed from here  
     //Code for Initialization  
   }  
}

While creating a static constructor, a few things need to be kept in mind:  
\* There is no access modifier require to define a static constructor  
\* There may be only one static constructor in a class  
\* The static constructor may not have any parameters  
\* This constructor may only access the static members of the class  
\* We may create more than one static constructor for a class  
  
**Can a class be created without a constructor?**   
No. In case we dont define the constructor, the class will access the no-argument constructor from its base class. The compiler will make this happen during compilation.

What are generics in C#?

**Generics** in C# is a new innovative feature through which classes and methods in C# may be designed in such a way that the rules of a type are not followed until it is declared. The generics feature in C# has been introduced with version 2.0 of the .NET Framework. Using generics, a class template may be declared that may follow any type as required at runtime. The class behavior is later governed by the type that we pass to the class. Once the type is passed, the class behaves depending on the type passed to this generic class.  
  
More on generics in C-Sharp, read this article [Click Here](http://msdn2.microsoft.com/en-us/library/512aeb7t(vs.80).aspx).

What is the use of the main() function in C#?

Every executable C# application must contain a class defining a Main() method that signifies the entry point of the application. Note that the Main() method is of the access type **public** by nature. Moreover, it is also **static**. See example below:

using System;  
class Question   
{  
 public static int Main(string[] args)  
 {  
  Console.Writeline("Another Question");  
  Console.Readline();  
  return 0;  
 }  
}

A public member is accessible from other types. The main() method is set as static so that it may be invoked at class level itself, without the need of creating an instance of it. The single parameter is an array of strings. that may contain any number of incoming command line arguments.

Are C# Keywords in lowercase or uppercase?

All C# keywords are in lowercase.  
  
Namespaces, Types and Members have their first character in uppercase

What is the use of Console.Readline() in C#?

Using the Console.Readline() method means that the command prompt launched by Visual Studio remains visible during an application session until the ENTER key is pressed by the user.

Can we set the specifier of the main() method in C# as private?

Yes. When the access specifier is set as private for the Main() method, then other assemblies may not invoke this class' Main() method as the starting point of the application. The startup scope gets limited to the class in context itself. See code below:

private static void Main()  
{  
  //This code isn't invoked automatically by other assemblies  
}

Can we set different types of parameters & return-types in main() method"?

Yes. The Main() method may easily be played around with by developers by passing different parameters and setting different return types. See the code below, that demonstrates different ways the Main() method may be implemented:

(1)  
public static void Main(string[] args)  
{  
 //NO return type, the argument is an array of strings  
}  
  
(2)  
public static int Main(string[] args)  
{  
 //Return type is int, argument is an array of strings  
}  
  
(3)  
public static int Main()  
{  
 //Return type is int, NO arguments  
}  
  
(4)  
public static void Main()  
{  
 //Return type is void, NO arguments  
}

What is the use of GetCommandLineArgs() method?

The **GetCommandLineArgs()** method is used to access the command line arguments. The return value of this method is an array of strings. It is a method of the **System.Environment** class. See the code example below:

public static int Main(string[] args)  
{  
 string[] strArgs = System.Environment.GetCommandLineArgs();  
 Console.WriteLine("Arguments {0}", strArgs[0]);  
}

What is the use of System.Environment class?

The class **System.Environment** is used to retrieve information about the operating system. Some of the static members of this class are as follows:  
1) Environment.OSVersion - Gets the version of the operating system  
2) Environment.GetLogicalDrives() - method that returns the drives  
3) Environment.Version - returns the .NET version running the application  
4) Environment.MachineName - Gets name of the current machine  
5) Environment.Newline - Gets the newline symbol for the environment  
6) Environment.ProcessorCount - returns number of processors on current machine  
7) Environment.SystemDirectory - returns complete path to the System Directory  
8) Environment.UserName - returns name of the entity that invoked the application

Why is the new keyword used for instantiating an object in .NET?

The **new** keyword instructs the .NET compiler to instantiate a new object, with appropriate number of bytes (depending on the type) for the object and gather required memory from the managed heap.

**What are the default values for bool, int, double, string, char, reference-type variables?**

When the objects of the following types are declared, then they have a default value during declaration. The following table shows the default value for each type:

|  |  |
| --- | --- |
| **Type** | **Default Value** |
| bool | false |
| int | 0 |
| double | 0 |
| string | null |
| char | '\0' |
| Reference Type | null |

Why does the compiler throw an error when variables are declared with an initial value?

The compiler throws an error when a variable is declared without an initial value, when the declaration is within a member scope. The variables do not recieve default value when they are declared within a scope.   
Note that if the variable is used as an output parameter, this variable does not need to be assigned an initial value.

How to declare a constant variable in C#? What is the use of the const keyword?

If a variable needs to have a fixed value, that may not be changed across the application's life, then it may be declared with the **const** keyword. The value assigned to a constant variable (using the const keyword) must be known at the time of compilation.

In C#, can we create an object of reference type using const keyword?

No. A constant member may not be created of an object that is of reference type, because its value is decided dynamically at runtime

What is the difference between const and readonly in C#?

When using the const keyword to set a constant variable, the value needs to be set at compile time. The **const** keyword may not be used with an object of reference type.  
  
In case a reference type needs to be assigned a non-changeable value, it may be set as **readonly**

**What are the different parameter modifiers available in C#?**

**What is a parameter modifier?**  
  
Parameter modifiers in C# are entities that controls the behaviour of the arguments passed in a method. Following are the different parameter modifiers in C#:  
1) None - if there is NO parameter modifier with an argument, it is passed by value, where the method recieves a copy of the original data.   
2) **out** - argument is passed by reference. The argument marked with "out" modifier needs to be assigned a value within this function, otherwise a compiler error is returned.

public void multiply(int a, int b, out int prod)  
{  
 prod = a \* b;  
}

Here, note that prod is assigned a value. If not done so, then a compile time error is returned.   
3) **params**- This modifier gives the permission to set a variable number of identical datatype arguments.  
Note that a method may have **only one** "params" modifier. The params modifier needs to be in the **last argument.**

static int totalruns(params int[] runs)  
{  
  int score = 0;  
  for(int x=0; x<RUNS,LENGTH;X++)  
 &nsbp;score+=runs[x];  
 return score;  
}

Further, from the calling function, we may pass the scores of each batsman as below...

score = totalruns(12,36,0,5,83,25,26);

4) **ref** - The argument is given a value by the caller, where data is passed by reference. This value may optionally be reset in the called method. Note that even if NO value is set in the called method for the **ref** attribute, no compiler error is raised.

What is the difference between out and ref in C#?

**1)** **out** parameters return compiler error if they are not assigned a value in the method. Not such with **ref** parameters.  
  
**2)** **out** parameters need not be initialized before passing to the method, whereas **ref** parameters need to have an initial value before they are passed to a method.