Very important interview questions

**Abstraction**  
  
Abstraction allows us to represent complex real world in simplest manner. It is process of identifying the relevant qualities and behaviors an object should possess, in other word represent the necessary feature without representing the back ground details. Abstraction is a process of hiding work style of an object and showing only those information which are required to understand the object. Abstraction means putting all the variables and methods in a class which are necessary.  
  
**Encapsulation**  
  
It is a process of hiding all the internal *details* of an object from the outside real world. The word Encapsulation, like Enclosing into the capsule. It restrict client from seeing its internal view where behavior of the abstraction is implemented. In Encapsulation, generally to hide data making it private and expose public property to access those data from outer world. Encapsulation is a method for protecting data from unwanted access or alteration. Encapsulation is the mechanism by which Abstraction is implemented.  
  
**Difference between Abstraction and Encapsulation**  
  
Abstraction is a process. It is the act of identifying the relevant qualities and behaviors an object should possess. Encapsulation is the mechanism by which the abstraction is implemented.

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| Abstraction solves the problem in the design level. | Encapsulation solves the problem in the implementation level. |
| Abstraction is used for hiding the unwanted data and giving only relevant data. | Encapsulation is hiding the code and data into a single unit to protect the data from outer world. |
| Abstraction is set focus on the object instead of how it does it. | Encapsulation means hiding the internal details or mechanics of how an object does something. |
| Abstraction is outer layout in terms of design.  For Example: - Outer Look of a iPhone, like it has a display screen. | Encapsulation is inner layout in terms of implementation. For Example: - Inner Implementation detail of a iPhone, how Display Screen are connect with each other using circuits |

Introduction

In this article along with the demo project I will discuss Interfaces versus Abstract classes. The concept of Abstract classes and Interfaces is a bit confusing for beginners of Object Oriented programming. Therefore, I am trying to discuss the theoretical aspects of both the concepts and compare their usage. And finally I will demonstrate how to use them with C#.

Background

An Abstract class without any implementation just looks like an Interface; however there are lot of differences than similarities between an Abstract class and an Interface. Let's explain both concepts and compare their similarities and differences.

What is an Abstract Class?

An abstract class is a special kind of class that cannot be instantiated. So the question is why we need a class that cannot be instantiated? An abstract class is only to be sub-classed (inherited from). In other words, it only allows other classes to inherit from it but cannot be instantiated. The advantage is that it enforces certain hierarchies for all the subclasses. In simple words, it is a kind of contract that forces all the subclasses to carry on the same hierarchies or standards. (Maintain code integrity)

What is an Interface?

An interface is not a class. It is an entity that is defined by the word Interface. An interface has no implementation; it only has the signature or in other words, just the definition of the methods without the body. As one of the similarities to Abstract class, it is a contract that is used to define hierarchies for all subclasses or it defines specific set of methods and their arguments. The main difference between them is that a class can implement more than one interface but can only inherit from one abstract class. Since C# doesn’t support multiple inheritance, interfaces are used to implement multiple inheritance.

Both Together

When we create an interface, we are basically creating a set of methods without any implementation that must be overridden by the implemented classes. The advantage is that it provides a way for a class to be a part of two classes: one from inheritance hierarchy and one from the interface.

When we create an abstract class, we are creating a base class that might have one or more completed methods but at least one or more methods are left uncompleted and declared abstract. If all the methods of an abstract class are uncompleted then it is same as an interface. The purpose of an abstract class is to provide a base class definition for how a set of derived classes will work and then allow the programmers to fill the implementation in the derived classes.

There are some similarities and differences between an interface and an abstract class that I have arranged in a table for easier comparison:

| **Feature** | **Interface** | **Abstract class** |
| --- | --- | --- |
| Multiple inheritance | A class may inherit several interfaces. | A class may inherit only one abstract class. |
| Default implementation | An interface cannot provide any code, just the signature. | An abstract class can provide complete, default code and/or just the details that have to be overridden. |
| Access Modfiers | An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public | An abstract class can contain access modifiers for the subs, functions, properties |
| Core VS Peripheral | Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface. | An abstract class defines the core identity of a class and there it is used for objects of the same type. |
| Homogeneity | If various implementations only share method signatures then it is better to use Interfaces. | If various implementations are of the same kind and use common behaviour or status then abstract class is better to use. |
| Speed | Requires more time to find the actual method in the corresponding classes. | Fast |
| Adding functionality (Versioning) | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. | If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. |
| Fields and Constants | No fields can be defined in interfaces | An abstract class can have fields and constrants defined |

**Constructor**  
  
Constructors are special methods called when a class is instantiated.

* Constructor will not return anything.
* Constructor name is same as class name.
* By default C# will create default constructor internally.
* Constructor with no arguments and no body is called default constructor.
* Constructor with arguments is called parameterized constructor.
* Constructor by default public.
* We can create private constructors.
* A method with same name as class name is called constructor there is no separate keyword.

To use a private constructor we should have main function in the same class, generally we will define constructors in different classes so defining private constructors is not that much useful.  
  
**Static Constructor**  
  
A static constructor has the same name as the class name but preceded with the static keyword; it will be called at the time of class load.

* No access specifier for static constructor.
* Static constructor will not return anything.
* Static constructor will accept only static members.
* Static constructor will call at the time of class loading.
* Static constructor will not allow overloading, so there is no parameterized static constructor.

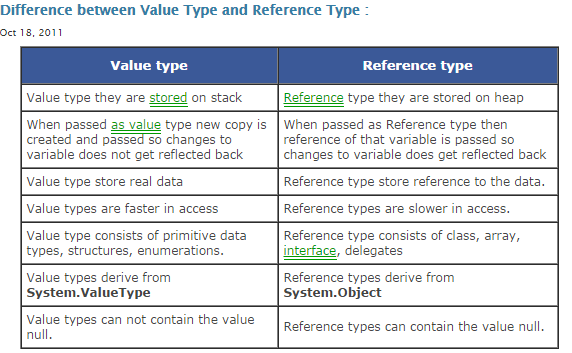
**Destructors**  
  
.Net will clean up the un-used objects by using garbage collection process. It internally uses the destruction method to clean up the un-used objects. Some times the programmer needs to do manual cleanup.  
  
**Syntax:**  
  
~<ClassName>

{}

Since C# is garbage collected, meaing that the framework will free the objects that you no longer use, there may be times where you need to do some manual cleanup. A destructor, a method called once an object is disposed, can be used to cleanup resources used by the object. Destructors doesn't look very much like other methods in C#. Here is an example of a destructor for our Car class:

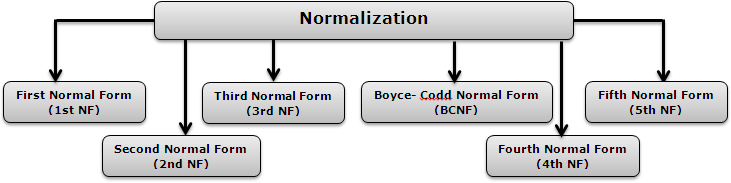
The main difference between Union and Union ALL operator is

Union operator will return distinct values but Union ALL returns all the values including duplicate values



**NORMALIZATION:**  
  
Normalization is the process of organizing data into a related table; it also eliminates redundancy and increases the integrity which improves performance of the query. To normalize a database, we divide the database into tables and establish relationships between the tables.  
  
Database normalization can essentially be defined as the practice of optimizing table structures. Optimization is accomplished as a result of a thorough investigation of the various pieces of data that will be stored within the database, in particular concentrating upon how this data is interrelated.  
  
**Normalization Avoids:**

* **Duplication of Data** - The same data is listed in multiple lines of the database
* **Insert Anomaly**- A record about an entity cannot be inserted into the table without first inserting information about another entity - Cannot enter a customer without a sales order
* **Delete Anomaly** - A record cannot be deleted without deleting a record about a related entity. Cannot delete a sales order without deleting all of the customer's information.
* **Update Anomaly** - Cannot update information without changing information in many places. To update customer information, it must be updated for each sales order the customer has placed

**DE- NORMALIZATION:**  
  
Denormalization is the process of adding redundant data to speed up complex queries involving multiple table JOINS. One might just go to a lower form of Normalization to achieve Denormalization and better performance. Data is included in one table from another in order to eliminate the second table which reduces the number of JOINS in a query and thus achieves performance.  
  
Normalization is a Six stage process - After the first stage, the data is said to be in first normal form, after the second, it is in second normal form, after the third, it is in third normal form and so on.  
  
  
  
**First Normal Form (1st NF)**  
  
In 1st NF:

* The table cells must be of single value.
* Eliminate repeating groups in individual tables.
* Create a separate table for each set of related data.
* Identify each set of related data with a primary key.

**Definition:** An entity is in the first normal form if it contains no repeating groups. In relational terms, a table is in the first normal form if it contains no repeating columns. Repeating columns make your data less flexible, waste disk space, and make it more difficult to search for data.  
  
**IMP:** In 1NF relation the order of tuples (rows) and attributes (columns) does not matter.   
  
**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Order** | **Customer** | **Contact Person** | **Total** |
| 1 | Rishabh | Manish | 134.23 |
| 2 | Preeti | Rohan | 521.24 |
| 3 | Rishabh | Manish | 1042.42 |
| 4 | Rishabh | Manish | 928.53 |

The above relation satisfies the properties of a relation and is said to be in first normal form (or 1NF). Conceptually it is convenient to have all the information in one relation since it is then likely to be easier to query the database.   
  
**Second Normal Form (2nd NF)**  
  
In 2nd NF:

* Remove Partial Dependencies.
* Functional Dependency: The value of one attribute in a table is determined entirely by the value of another.
* Partial Dependency: A type of functional dependency where an attribute is functionally dependent on only part of the primary key (primary key must be a composite key).
* Create separate table with the functionally dependent data and the part of the key on which it depends. Tables created at this step will usually contain descriptions of resources.

**Definition:** A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation.   
  
**Example:**  
  
**The following relation is not in Second Normal Form:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Order** | **Customer** | **Contact Person** | **Total** |
| 1 | Rishabh | Manish | 134.23 |
| 2 | Preeti | Rohan | 521.24 |
| 3 | Rishabh | Manish | 1042.42 |
| 4 | Rishabh | Manish | 928.53 |

In the table above, the order number serves as the primary key. Notice that the customer and total amount are dependent upon the order number -- this data is specific to each order. However, the contact person is dependent upon the customer. An alternative way to accomplish this would be to create two tables:

|  |  |
| --- | --- |
| **Customer** | **Contact Person** |
| Rishabh | Manish |
| Preeti | Rohan |

|  |  |  |
| --- | --- | --- |
| **Order** | **Customer** | **Total** |
| 1 | Rishabh | 134.23 |
| 2 | Preeti | 521.24 |
| 3 | Rishabh | 1042.42 |
| 4 | Rishabh | 928.53 |

The creation of two separate tables eliminates the dependency problem. In the first table, contact person is dependent upon the primary key -- customer name. The second table only includes the information unique to each order. Someone interested in the contact person for each order could obtain this information by performing a Join Operation.  
  
**Third Normal Form (3rd NF)**  
In 3rd NF:

* Remove transitive dependencies.
* Transitive Dependency A type of functional dependency where an attribute is functionally dependent on an attribute other than the primary key. Thus its value is only indirectly determined by the primary key.
* Create a separate table containing the attribute and the fields that are functionally dependent on it. Tables created at this step will usually contain descriptions of either resources or agents. Keep a copy of the key attribute in the original file.

A relation is in third [normal](http://www.watchesn.com/) form, if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on each candidate key of the relation.   
  
**Non-transitive dependency:**  
  
Let A, B and C be three attributes of a relation R such that Aïƒ B and Bïƒ C. From these FDs, we may derive Aïƒ C. This dependence Aïƒ C is transitive.   
  
**Example:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Company** | **City** | **State** | **ZIP** |
| ABC Ltd. | Mumbai | MH | 10169 |
| XYZ Ltd. | Noida | UP | 33196 |
| ASD Ltd. | Chennai | TN | 21046 |

The above table is not in the 3NF.  
  
In this example, the city and state are dependent upon the ZIP code. To place this table in 3NF, two separate tables would be created -- one containing the company name and ZIP code and the other containing city, state, ZIP code pairings.

|  |  |
| --- | --- |
| **Company** | **ZIP** |
| ABC Ltd. | 10169 |
| XYZ Ltd. | 33196 |
| ASD Ltd. | 21046 |

|  |  |  |
| --- | --- | --- |
| **City** | **State** | **ZIP** |
| Mumbai | MH | 10169 |
| Noida | UP | 33196 |
| Chennai | TN | 21046 |

This may seem overly complex for daily applications and indeed it may be. Database designers should always keep in mind the tradeoffs between higher level normal forms and the resource issues that complexity creates.   
  
**Boyce-Codd Normal Form (BCNF)**  
  
In BCNF:

* When a relation has more than one candidate key, anomalies may result even though the relation is in 3NF.
* 3NF does not deal satisfactorily with the case of a relation with overlapping candidate keys
* i.e. composite candidate keys with at least one attribute in common.
* BCNF is based on the concept of a determinant.
* A determinant is any attribute (simple or composite) on which some other attribute is fully functionally dependent.
* A relation is in BCNF is, and only if, every determinant is a candidate key.

**Definition:** A relation is in Boyce-Codd Normal Form (BCNF) if every determinant is a candidate key. (See the links in the box at right for definitions of determinant and candidate key.)  
  
The difference between 3NF and BCNF is that for a functional dependency A ïƒ  B, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key,  
   
Whereas BCNF insists that for this dependency to remain in a relation, A must be a candidate key.  
  
**Example:**  
  
**CLIENT INTERVIEW:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ClientNo** | **InterviewDate** | **InterviewTime** | **StaffNo** | **RoomNo** |
| CR76 | 13-may-11 | 10:30 | SG5 | G101 |
| CR76 | 13-may-11 | 12:00 | SG5 | G101 |
| CR74 | 13-may-11 | 12:00 | SG37 | G102 |
| CR56 | 02-july-11 | 10:30 | SG5 | G102 |

â€¢ FD1 ClientNo, InterviewDate -> InterviewTime, StaffNo, RoomNo (Primary Key)  
â€¢ FD2 StaffNo, InterviewDate, InterviewTime -> ClientNo (Candidate key)  
â€¢ FD3 RoomNo, InterviewDate, InterviewTime -> ClientNo, StaffNo (Candidate key)  
â€¢ FD4 StaffNo, InterviewDate -> RoomNo (not a candidate key)  
  
As a consequece the ClientInterview relation may suffer from update anomalies.  
  
To transform the ClientInterview relation to BCNF, we must remove the violating functional dependency by creating two new relations called Interview and StaffRoom as shown below,  
  
**Interview (ClientNo, InterviewDate, InterviewTime, StaffNo)  
StaffRoom (StaffNo, InterviewDate, RoomNo)**  
**INTERVIEW**

|  |  |  |  |
| --- | --- | --- | --- |
| **ClientNo** | **InterviewDate** | **InterviewTime** | **StaffNo** |
| CR76 | 13-may-11 | 10:30 | SG5 |
| CR76 | 13-may-11 | 12:00 | SG5 |
| CR74 | 13-may-11 | 12:00 | SG37 |
| CR56 | 02-july-11 | 10:30 | SG5 |

**STAFFROOM**

|  |  |  |
| --- | --- | --- |
| **StaffNo** | **InterviewDate** | **RoomNo** |
| SG5 | 13-may-11 | G101 |
| SG37 | 13-may-11 | G102 |
| SG5 | 02-july-11 | G102 |

BCNF Interview and StaffRoom relations.  
  
An entity is in Fourth Normal Form (4NF) when it meets the requirement of being in Third Normal Form (3NF) and additionally:

* Has no multiple sets of multi-valued dependencies. In other words, 4NF states that no entity can have more than a single one-to-many relationship within an entity if the one-to-many attributes are independent of each other.
* Many:many relationships are resolved independently.

**Fourth Normal Form (4th NF)**  
  
In 4th NF:  
  
An entity is in Fourth Normal Form (4NF) when it meets the requirement of being in Third Normal Form (3NF) and additionally:

* Has no multiple sets of multi-valued dependencies. In other words, 4NF states that no entity can have more than a single one-to-many relationship within an entity if the one-to-many attributes are independent of each other.
* Fourth Normal Form applies to situations involving many-to-many relationships.

In relational databases, many-to-many relationships are expressed through cross-reference tables.  
  
**Definition:** A table is in fourth normal form (4NF) if and only if it is in BCNF and contains no more than one multi-valued dependency.   
  
**Example:**  
  
Take an example of Employee Table  
info(Employee, Skills, Hobbies)

|  |  |  |
| --- | --- | --- |
| **Employee** | **Skills** | **Hobbies** |
| 1 | Programming | Golf |
| 1 | Programming | Bowling |
| 1 | Analysis | Golf |
| 1 | Analysis | Bowling |
| 2 | Analysis | Golf |
| 2 | Analysis | Gardening |
| 2 | Management | Golf |
| 2 | Management | Gardening |

This table is difficult to maintain since adding a new hobby requires multiple new rows corresponding to each skill. This problem is created by the pair of multi-valued dependencies EMPLOYEE -> SKILLS and EMPLOYEE -> HOBBIES. A much better alternative would be to decompose INFO into two relations:

|  |  |
| --- | --- |
| **Employee** | **Skills** |
| 1 | Programming |
| 1 | Analysis |
| 2 | Analysis |
| 2 | Management |

Hobbies(Employee, Hobby)

|  |  |
| --- | --- |
| **Employee** | **Hobbies** |
| 1 | Golf |
| 1 | Bowling |
| 2 | Golf |
| 2 | Gardening |

**Fifth Normal Form (5th NF)**  
  
In 5th NF:

* A relation that has a join dependency cannot be decomposed by a projection into other relations without spurious results
* A relation is in 5NF when its information content cannot be reconstructed from several smaller relations i.e. from relations having fewer attributes than the original relation

**Definition:** A table is in fifth normal form (5NF) or Project-Join Normal Form (PJNF) if it is in 4NF and it cannot have a lossless decomposition into any number of smaller tables.  
  
Fifth normal form, also known as join-projection normal form (JPNF), states that no non-trivial join dependencies exist. 5NF states that any fact should be able to be reconstructed without any anomalous results in any case, regardless of the number of tables being joined. A 5NF table should have only candidate keys and it's primary key should consist of only a single column.  
  
**Example:**  
  
Take an example of a buying table. This is used to track buyers, what they buy, and from whom they buy. Take the following sample data:

|  |  |  |
| --- | --- | --- |
| **Buyer** | **Vendor** | **Item** |
| Shalley | Kashmir House | Jeans |
| Mary | Kashmir House | Jeans |
| Shalley | Radhika Sarees | Saree |
| Mary | Radhika Sarees | Saree |
| Shalley | Radhika Sarees | Suit |

The problem with the above table structure is that if Claiborne starts to sell Jeans then how many records must you create to record this fact? The problem is there are pair wise cyclical dependencies in the primary key. That is, in order to determine the item you must know the buyer and vendor, and to determine the vendor you must know the buyer and the item, and finally to know the buyer you must know the vendor and the item.  
  
And the solution is to break this one table into three tables; Buyer-Vendor, Buyer-Item, and Vendor-Item. So following tables are in the 5NF.  
  
**BUYER-VENDOR**

|  |  |
| --- | --- |
| **Buyer** | **Vendor** |
| Shalley | Kashmir House |
| Mary | Kashmir House |
| Shalley | Radhika Sarees |
| Mary | Radhika Sarees |

**BUYER-ITEM**

|  |  |
| --- | --- |
| **Buyer** | **Item** |
| Shalley | Jeans |
| Mary | Jeans |
| Shalley | Saree |
| Mary | Saree |
| Shalley | Suit |

**VENDOR- ITEM**

|  |  |
| --- | --- |
| **Vendor** | **Item** |
| Kashmir House | Jeans |
| Radhika Sarees | Saree |
| Radhika Sarees | Suit |

1, HTTPModule:

    It's just like a filter. The Modules are called before and after the handler executes.

    For example: BeginRequest, AuthenticationRequest event, EndRequest event etc. You may intercept , participate and modify each request.

    Modules implement the IHttpMudule interface located in the System.Web.System.

2, HTTPHandler:

    You may think it as a  program.(or handler, or module), it execute some code when the user send some request. An .aspx page can be thought as a HTTPHandler too, which implements more functions.

There are two type of cookies we use in ASP.Net...

1- Persistent Cookie:-  A cookie which store your information in your hard disk, and it will stored till time either you will not delete that or cookie time period does not expire. it is browser independent cookie.

2- Non - Persistent Cookie(Temprary cookie):- A cookie which will alive till that time untill your browser is alive, As you will logout to your browser or close your browser, cookie will get expire and user information would also be lost. So it is a browser dependent cookie,

The Difference between**Dictionary** and **Hash table** can be represented as follows:

**1.Dictionary** is a generic type whereas **Hashtable** is not.

**Dictionary** is a generic type means

a)We get type safety with **dictionary** ,bcoz we can't insert any random object into it and don't have to cast the value when take out whereas in **hashtable** we can dynamically  add or remove elements without indexing.

b)Generic collection is faster as there is no boxing or unboxing here.

c)Hashtable have to box/unbox so again there is memory consumption as well as performance penalties.

d)Hash table uses object to hold the things internally(only non generic way to do it).

**2.Hashtable** is a collection of name/value pairs that are organised on the basis of hash code of the key being specified. whereas  In**Dictionary** we need to specify the types of both the key and the corresponding value.the value represents the actual object stored and the key represents a means to identify a particular object.

**3 Hashtable** is thread safe for use by multiple reader threads and a single writing thread whereas in **Dictionary** public static members are type safe but any instance members are not type safe.

4.We cannot use **Dictionary** with Web Services The reason is no web service standard supports generic standard.

How can we return multiple value from function?

There are a number of ways to return two object from one function. You can return one as usual return and another one as out parameter. You can create some type (class or structure) with two members (object and numeric) and return an object of this type. Alternatively, you can use an already defined generic type with two members.

|  |
| --- |
| **Components of .Net Framework** |
| There are many articles are available in the web on this topic; I just want to add one more article over the web by explaining Components of .Net Framework.  **Components of .Net Framework**  Components.jpg  Net Framework is a platform that provides tools and technologies to develop Windows, Web and Enterprise applications. It mainly contains two components,  1.   Common Language Runtime (CLR)  2.    .Net Framework Class Library.  1**. Common Language Runtime**(CLR) **.Net Framework** provides runtime environment called **Common Language Runtime** (CLR).It provides an environment to run all the .Net Programs. The code which runs under the CLR is called as **Managed Code**. Programmers need not to worry on managing the memory if the programs are running under the CLR as it provides memory management and thread management.  Programmatically, when our program needs memory, CLR allocates the memory for scope and de-allocates the memory if the scope is completed.  Language Compilers (e.g. C#, VB.Net, J#) will convert the Code/Program to **Microsoft Intermediate Language** (MSIL) intern this will be converted to **Native Code** by CLR. See the below Fig.  MSILCode.jpg  There are currently over 15 language compilers being built by Microsoft and other companies also producing the code that will execute under CLR.   2.    **.Net Framework Class Library**(FCL)  This is also called as Base Class Library and it is common for all types of applications i.e. the way you access the Library Classes and Methods in VB.NET will be the same in C#, and it is common for all other languages in .NET.   The following are different types of applications that can make use of .net class library.  1.                   Windows Application.  2.                   Console Application  3.                   Web Application.  4.                   XML Web Services.  5.                   Windows Services.  In short, developers just need to import the BCL in their language code and use its predefined methods and properties to implement common and complex functions like reading and writing to file, graphic rendering, database interaction, and XML document manipulation.  Below are the few more concepts that we need to know and understand as part of this .Net framework.    3.   **Common Type System**(CTS)  It describes set of data types that can be used in different .Net languages in common. (i.e), CTS ensures that objects written in different .Net languages can interact with each other.  For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level.  The common type system supports two general categories of types:   **Value types:**  Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations. **Reference types:**  Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.    4**. Common Language Specification**(CLS)  It is a sub set of CTS and it specifies a set of rules that needs to be adhered or satisfied by all language compilers targeting CLR. It helps in cross language inheritance and cross language debugging. **Common language specification Rules:**  It describes the minimal and complete set of features to produce code that can be hosted by CLR. It ensures that products of compilers will work properly in .NET environment.  **Sample Rules:**  1.       Representation of text strings  2.       Internal representation of enumerations  3.       Definition of static members and this is a subset of the CTS which all .NET languages are expected to support.  4.   Microsoft has defined CLS which are nothing but guidelines that language to follow so that it can communicate with other .NET languages in a seamless manner. |

**What is a .Net Assembly?**  
The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

There are two kind of assemblies in .NET;

* private
* shared

**Private assemblies** are simple and copied with each calling assemblies in the calling assemblies folder.

**Shared assemblies** (also called strong named assemblies) are copied to a single location (usually the Global assembly cache). For all calling assemblies within the same application, the same copy of the shared assembly is used from its original location. Hence, shared assemblies are not copied in the private folders of each calling assembly. Each shared assembly has a four part name including its face name, version, public key token and culture information. The public key token and version information makes it almost impossible for two different assemblies with the same name or for two similar assemblies with different version to mix with each other.

An assembly can be a single file or it may consist of the multiple files. In case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a sub part of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes, but unfortunately very few people take interest in learning such theoretical looking topics.

**What are the basic components of .NET platform?**

The basic components of .NET platform (framework) are:

|  |
| --- |
| **.Net Applications**  **(***Win Forms,Web Applications,Web Services***)** |
| **Data(ADO.Net) and XML Library** |
| **FrameWork Class Library(FCL)**  *(IO,Streams,Sockets,Security,Reflection,UI)* |
| **Common Language Runtime(CLR)**  *(Debugger,Type Checker,JITer,GC)* |
| **Operating System**  *(Windows,Linux,UNIX,Macintosh,etc.,)* |

**Common Language Runtime (CLR):**

The most important part of the .NET Framework is the .Net Common Language Runtime (CLR) also called .Net Runtime in short. It is a framework layer that resides above the Operating System and handles/manages the execution of the .NET applications. Our .Net programs don't directly communicate with the Operating System but through CLR.

**MSIL (Microsoft Intermediate Language) Code:**

When we compile our .Net Program using any .Net compliant language like (C#, VB.NET, C++.NET) it does not get converted into the executable binary code but to an intermediate code, called MSIL or IL in short, understandable by CLR. MSIL is an OS and H/w independent code. When the program needs to be executed, this MSIL or intermediate code is converted to binary executable code, called native code. The presence of IL makes it possible the Cross Language Relationship as all the .Net compliant languages produce the similar standard IL code.

**Just In Time Compilers (JITers):**

When our IL compiled code needs to be executed, CLR invokes JIT compilers which compile the IL code to native executable code (.exe or .dll) for the specific machine and OS. JITers in many ways are different from traditional compilers as they, as their name suggests, compile the IL to native code only when desired e.g., when a function is called, IL of function's body is converted to native code; just in time of need. So, the part of code that is not used by particular run is not converted to native code. If some IL code is converted to native code then the next time when its needed to be used, the CLR uses the same copy without re-compiling. So, if a program runs for sometime, then it won't have any just in time performance penalty. As JITers are aware of processor and OS exactly at runtime, they can optimize the code extremely efficiently resulting in very robust applications. Also, since JITer knows the exact current state of executable code, they can also optimize the code by in-lining small function calls (like replacing body of small function when its called in a loop, saving the function call time). Although, Microsoft stated that C# and .Net are not competing with languages like C++ in efficiency, speed of execution, JITers can make your code even faster than C++ code in some cases when program is run over extended period of time (like web-servers).

**Framework Class Library (FCL):**

.NET Framework provides huge set of Framework (or Base) Class Library (FCL) for common, usual tasks. FCL contains thousands of classes to provide the access to Windows API and common functions like String Manipulation, Common Data Structures, IO, Streams, Threads, Security, Network Programming, Windows Programming, Web Programming, Data Access, etc. It is simply the largest standard library ever shipped with any development environment or programming language. The best part of this library is they follow extremely efficient OO design (design patterns) making their access and use very simple and predictable. You can use the classes in FCL in your program just as you use any other class and can even apply inheritance and polymorphism on these.

**Common Language Specification (CLS):**

Earlier we used the term '.NET Compliant Language' and stated that all the .NET compliant languages can make use of CLR and FCL. But what makes a language '.NET compliant language'? The answer is Common Language Specification (CLS). Microsoft has released a small set of specification that each language should meet to qualify as a .NET Compliant Language. As IL is a very rich language, it is not necessary for a language to implement all the IL functionality, rather it meets the small subset of it, CLS, to qualify as a .NET compliant language, which is the reason why so many languages (procedural and OO) are now running under .Net umbrella. CLS basically addresses to language design issues and lays certain standards like there should be no global function declaration, no pointers, no multiple inheritance and things like that. The important point to note here is that if you keep your code within CLS boundary, your code is guaranteed to be usable in any other .Net language.

**Common Type System (CTS):**

.NET also defines a Common Type System (CTS). Like CLS, CTS is also a set of standards. CTS defines the basic data types that IL understands. Each .NET compliant language should map its data types to these standard data types. This makes it possible for the 2 languages to communicate with each other by passing/receiving parameters to/from each other. For example, CTS defines a type Int32, an integral data type of 32 bits (4 bytes) which is mapped by C# through int and VB.Net through its Integer data type.

**Garbage Collector (GC):**

CLR also contains Garbage Collector (GC) which runs in a low-priority thread and checks for un-referenced dynamically allocated memory space. If it finds some data that is no more referenced by any variable/reference, it re-claims it and returns the occupied memory back to the Operating System; so that it can be used by other programs as necessary. The presence of standard Garbage Collector frees the programmer from keeping track of dangling data.

WHY .NET

The .NET Framework has been developed to cater to the following objectives and requirements:

* To provide a consistent object-oriented environment to develop applications.
* To provide a code execution environment that simplifies deployment and versioning.
* To provide a code execution environment that guarantees the safety of the code that is executing. This includes both code developed internally by an organization or for code developed by 3rd party vendors.
* To provide a code execution environment that eliminates the issues faced by scripted environments with respect to performance.
* To provide a common programming model where the choice of a programming language becomes a matter of choice.

Microsoft SQL Server Profiler is a graphical user interface to SQL Trace for monitoring T-SQL Statements of Database Engine. We can save and reuse the state at a later point of time.

* **We can do the following using SQL Server Profiler**
  + Create a trace
  + Watch the trace results as the trace runs
  + Store the trace results in a table
  + Start, stop, pause, and modify the trace results as necessary
  + Replay the trace results
* Use SQL Server Profiler to monitor only the events in which you are interested.

It has two selection tabs:

* **General:**It is used for general setting for Trace Database Engine.
* **Event:** It is used to add or remove some selected event for monitor.

In General Section (as given in Figure 1.1), it is divided into four sections.

**Section 1:**In this section, you have to just specify the name of your trace, Trace provider name and server name are predefined and based upon your SQL Server.

And it is not editable.

**Section 2:**It is the template section. You can choose different type of Templates based upon your requirements. It is the configuration for trace. By default, it is "Standard (Default)" templates. Others templates are T-SQL, T-SQL Duration, T-SQL Reply, T-SQL SPs, etc. You can create your own custom Templates by selecting different Events and Event Class. It is saved as "*.tdf*" Extension.

**Section 3:**This section is related to save your trace. Either as File (*.trc*) or in a database. as table. While clicking on Save to file check box, File save dialog box should open and you can save that file (with*.trc* extension).

If you check the *"Save to Table",*it will connect with your server and ask you to which database you want to save that trace table information.

**Protected Member**  
   
 Protected Member of a class in only available in the contained class (in which it has been declared) and in the derived class within the  assembly and also outside the assembly.  
   
 Means if a class that resides outside the assembly can use the protected member of the other assembly by inherited that class only.  
   
 We can exposed the Protected member outside the assembly by inherited that class and use it in the derived class only.  
  
**Note:**Protected members are not accessible using the object in the derived class.  
   
  **Internal Member**  
   
 Internal Member of a class is available or access within the assembly either creating object or in a derived class or you can say it is  accessible across all the classes within the assembly.  
   
**Note**: Internal members not accessible outside the assembly either using object creating or in a derived class.  
   
   
 **Protected Internal**  
   
 Protected Internal access modifier is combination Protected or Internal.   
   
 Protected Internal Member can be available within the entire assembly in which it declared either creating object or by inherited that  class. And can be accessible outside the assembly in a derived class only.

**Note**: Protected Internal member works as Internal within the same assembly and works as Protected for outside the assembly.

How many web.config files : 1 per folder

Reflection is the ability of a managed code to read its own metadata for the purpose of finding assemblies, modules and type information at runtime. In other words, reflection provides objects that encapsulate assemblies, modules and types. A program reflects on itself by extracting metadata from its assembly and using that metadata either to inform the user or to modify its own behavior. Reflection is similar to C++ RTTI (Runtime Type Information), but much broader in scope and capability. By using Reflection in C#, one is able to find out details of an object, method, and create objects and invoke methods at runtime. The System.Reflection namespace contains classes and interfaces that provide a managed view of loaded types, methods, and fields, with the ability to dynamically create and invoke types. When writing a C# code that uses reflection, the coder can use the typeof operator to get the object's type or use the GetType() method to get the type of the current instance. Here are sample codes that demonstrate the use of reflection:

#### Example 1

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

using System;

using System.Reflection;

public class MyClass

{

public virtual int AddNumb(int numb1,int numb2)

{

int result = numb1 + numb2;

return result;

}

}

class MyMainClass

{

public static int Main()

{

Console.WriteLine ("\nReflection.MethodInfo");

*// Create MyClass object*

MyClass myClassObj = new MyClass();

*// Get the Type information.*

Type myTypeObj = myClassObj.GetType();

*// Get Method Information.*

MethodInfo myMethodInfo = myTypeObj.GetMethod("AddNumb");

object[] mParam = new object[] {5, 10};

*// Get and display the Invoke method.*

Console.Write("\nFirst method - " + myTypeObj.FullName + " returns " +

myMethodInfo.Invoke(myClassObj, mParam) + "\n");

return 0;

}

}

Firstly, the code snippet below will get the type information:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

Type myTypeObj = Type.GetType("MyClass");

And myTypeObj will now have the required information about MyClass. Therefore we can now check if the class is anabstract class or a regular class by using either of these statements:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

myTypeObj.IsAbstract

or:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

myTypeObj.IsClass

The code snippet below will get the method's information. And the method that we are interested in this case isAddNumb:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

Methodinfo myMethodInfo = myTypeObj.GetMethod("AddNumb");

The following code snippet will invoke the AddNumb method:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

myMethodInfo.Invoke(myClassObj, mParam);

*//Example2: In this example, we will use the typeof keyword to obtain the*

*// System.Type object for a type.*

Public class MyClass2

{

int answer;

public MyClass2()

{

answer = 0;

}

public int AddNumb(intnumb1, intnumb2)

{

answer = numb1 + numb2;

return answer;

}

}

The code snippet below gets the System.Type object for the MyClass2 type.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

Type type1 = typeof(MyClass2);

So we will now be able to create an instance of the type1 object by passing the type1 object to theActivator.CreateInstance(type1) method.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

object obj = Activator.CreateInstance(type1);

Then we can now invoke the AddNumb method of the MyClass2 class by first creating an array of objects for the arguments that we would be passing to the AddNumb(int, int) method.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

object[] mParam =newobject[] {5, 10};

Finally, we would invoke the AddNumb(int, int) method by passing the method name AddNumb toSystem.Type.InvokeMember() with the appropriate arguments.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial)

int res = (int)type1.InvokeMember("AddNumb", BindingFlags.InvokeMethod,null,

obj, mParam);

*//Here is the complete code:*

using System;

using System.Reflection;

namespace Reflection

{

class Class1

{

public int AddNumb(int numb1, int numb2)

{

int ans = numb1 + numb2;

return ans;

}

[STAThread]

static void Main(string[] args)

{

Type type1 = typeof(Class1);

*//Create an instance of the type*

object obj = Activator.CreateInstance(type1);

object[] mParam = new object[] {5, 10};

*//invoke AddMethod, passing in two parameters*

int res = (int)type1.InvokeMember("AddNumb", BindingFlags.InvokeMethod,

null, obj, mParam);

Console.Write("Result: {0} \n", res);

}

}

}

How to find second largest salary from query

**Protected Member**  
   
 Protected Member of a class in only available in the contained class (in which it has been declared) and in the derived class within the  assembly and also outside the assembly.  
   
 Means if a class that resides outside the assembly can use the protected member of the other assembly by inherited that class only.  
   
 We can exposed the Protected member outside the assembly by inherited that class and use it in the derived class only.  
  
**Note:**Protected members are not accessible using the object in the derived class.  
   
  **Internal Member**  
   
 Internal Member of a class is available or access within the assembly either creating object or in a derived class or you can say it is  accessible across all the classes within the assembly.  
   
**Note**: Internal members not accessible outside the assembly either using object creating or in a derived class.  
   
   
 **Protected Internal**  
   
 Protected Internal access modifier is combination Protected or Internal.   
   
 Protected Internal Member can be available within the entire assembly in which it declared either creating object or by inherited that  class. And can be accessible outside the assembly in a derived class only.

**Note**: Protected Internal member works as Internal within the same assembly and works as Protected for outside the assembly.

[C# - Constructors in C# with Example, Types of Constructor in C# with Example](http://www.aspdotnet-suresh.com/2013/09/csharp-constructor-example-types-of-constructor-in-csharp.html)

By: Suresh Dasari Sep 25, 2013

Categories: [C#.Net](http://www.aspdotnet-suresh.com/search/label/C%23.Net), [Interview Questions](http://www.aspdotnet-suresh.com/search/label/Interview%20Questions), [OOPS Concepts](http://www.aspdotnet-suresh.com/search/label/OOPS%20Concepts)

**Introduction:**

Here I will explain what is constructor in [c#](http://www.aspdotnet-suresh.com/search/label/C%23.Net) with example, uses and types of constructors in [c#](http://www.aspdotnet-suresh.com/search/label/C%23.Net) default constructor, static constructor, copy constructor, parameterized constructor and private constructor examples in [c#.net](http://www.aspdotnet-suresh.com/search/label/C%23.Net).

**Description:**

In previous posts I explained [use of virtual, override and new keyword with examples in c#](http://www.aspdotnet-suresh.com/search/label/C%23.Net) [method overloading and overriding](http://www.aspdotnet-suresh.com/2013/09/c-sharp-overloading-and-overriding-differences-with-example.html), [delegates example in c#](http://www.aspdotnet-suresh.com/2013/09/C-Sharp-delegates-example-use-of-delegates-in-C-Sharp.html), [sealed class in c#](http://www.aspdotnet-suresh.com/2011/11/what-are-sealed-classes-in-c-uses-of.html), [using statement in c#,](http://www.aspdotnet-suresh.com/2013/08/c-using-statement-example-uses-of-using.html) [OOPS examples in c#](http://www.aspdotnet-suresh.com/2010/04/introduction-to-object-oriented.html) and many articles relating to [interview questions](http://www.aspdotnet-suresh.com/search/label/Interview%20Questions) in [c#](http://www.aspdotnet-suresh.com/search/label/C%23.Net), [asp.net](http://www.aspdotnet-suresh.com/search/label/Asp.net), [SQL server](http://www.aspdotnet-suresh.com/search/label/SQL%20Server), [JavaScript](http://www.aspdotnet-suresh.com/search/label/Javascript" \t "_blank),[jQuery](http://www.aspdotnet-suresh.com/search/label/JQuery). Now I will explain constructor in [c#.net](http://www.aspdotnet-suresh.com/search/label/C%23.Net) with example and different types of constructor in[c#.net](http://www.aspdotnet-suresh.com/search/label/C%23.Net) with example.

Constructor is a special method of a class which will invoke automatically whenever instance or object of class is created. Constructors are responsible for object initialization and memory allocation of its class. If we create any class without constructor, the compiler will automatically create one default constructor for that class. There is always at least one constructor in every class.

Here you need to remember that a class can have any number of constructors and constructors don’t have any return type, not even void and within a class we can create only one static constructor.

Generally constructor name should be same as class name. If we want to create constructor in a class we need to create a constructor method name same as class name check below sample method for constructor

|  |
| --- |
| class SampleA  {  public SampleA()  {  Console.WriteLine("Sample A Test Method");  }  } |

**Types of Constructors**

Basically constructors are 5 types those are

      1.    Default Constructor

      2.    Parameterized Constructor

      3.    Copy Constructor

      4.    Static Constructor

      5.    Private Constructor

**Default Constructor**

A constructor without having any parameters called default constructor. In this constructor every instance of the class will be initialized without any parameter values like as shown below

|  |
| --- |
| using System;  namespace ConsoleApplication3  {  class Sample  {  public string param1, param2;  public Sample()     // Default Constructor  {  param1 = "Welcome";  param2 = "Aspdotnet-Suresh";  }  }  class Program  {  static void Main(string[] args)  {  Sample obj=new Sample();   // Once object of class created automatically constructor will be called  Console.WriteLine(obj.param1);  Console.WriteLine(obj.param2);  Console.ReadLine();  }  }  } |

When we run above program it will show output like as shown below

**Output**

|  |
| --- |
| Welcome  Aspdotnet-Suresh |

**Parameterized Constructors**

A constructor with at least one parameter is called as parameterized constructor. In parameterized constructor we can initialize each instance of the class to different values like as shown below

|  |
| --- |
| using System;  namespace ConsoleApplication3  {  class Sample  {  public string param1, param2;  public Sample(string x, string y)     // Declaring Parameterized constructor with Parameters  {  param1 = x;  param2 = y;  }  }  class Program  {  static void Main(string[] args)  {  Sample obj=new Sample("Welcome","Aspdotnet-Suresh");   // Parameterized Constructor Called  Console.WriteLine(obj.param1 +" to "+ obj.param2);  Console.ReadLine();  }  }  } |

When we run above program it will show output like as shown below

**Output**

|  |
| --- |
| Welcome to Aspdotnet-Suresh |

**Constructor Overloading**

In c# we can overload constructor by creating another constructor with same method name and different parameters like as shown below

|  |
| --- |
| using System;  namespace ConsoleApplication3  {  class Sample  {  public string param1, param2;  public Sample()     // Default Constructor  {  param1 = "Hi";  param2 = "I am Default Constructor";  }  public Sample(string x, string y)     // Declaring Parameterized constructor with Parameters  {  param1 = x;  param2 = y;  }  }  class Program  {  static void Main(string[] args)  {  Sample obj = new Sample();   // Default Constructor will Called  Sample obj1=new Sample("Welcome","Aspdotnet-Suresh");   // Parameterized Constructor will Called  Console.WriteLine(obj.param1 + ", "+obj.param2);  Console.WriteLine(obj1.param1 +" to " + obj1.param2);  Console.ReadLine();  }  } |

When we run above program it will show output like as shown below

**Output**

|  |
| --- |
| Hi, I am Default Constructor  Welcome to Aspdotnet-Suresh |

**Copy Constructor**

A parameterized constructor that contains a parameter of same class type is called as copy constructor. Main purpose of copy constructor is to initialize new instance to the values of an existing instance. Check below example for this

|  |
| --- |
| using System;  namespace ConsoleApplication3  {  class Sample  {  public string param1, param2;  public Sample(string x, string y)  {  param1 = x;  param2 = y;  }  public Sample(Sample obj)     // Copy Constructor  {  param1 = obj.param1;  param2 = obj.param2;  }  }  class Program  {  static void Main(string[] args)  {  Sample obj = new Sample("Welcome", "Aspdotnet-Suresh");  // Create instance to class Sample  Sample obj1=new Sample(obj); // Here obj details will copied to obj1  Console.WriteLine(obj1.param1 +" to " + obj1.param2);  Console.ReadLine();  }  }  } |

When we run above program it will show output like as shown below

**Output**

|  |
| --- |
| Welcome to Aspdotnet-Suresh |

**Static Constructor**

When we declared constructor as static it will be invoked only once for any number of instances of the class and it’s during the creation of first instance of the class or the first reference to a static member in the class. Static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.

|  |
| --- |
| using System;  namespace ConsoleApplication3  {  class Sample  {  public string param1, param2;  static Sample()  {  Console.WriteLine("Static Constructor");  }  public Sample()  {  param1 = "Sample";  param2 = "Instance Constructor";  }  }  class Program  {  static void Main(string[] args)  {  // Here Both Static and instance constructors are invoked for first instance  Sample obj=new Sample();  Console.WriteLine(obj.param1 + " " + obj.param2);  // Here only instance constructor will be invoked  Sample obj1 = new Sample();  Console.WriteLine(obj1.param1 +" " + obj1.param2);  Console.ReadLine();  }  }  } |

When we run above program we will get output like as shown below

**Output**

|  |
| --- |
| Static Constructor  Sample Instance Constructor  Sample Instance Constructor |

**Importance points of static constructor**

-      Static constructor will not accept any parameters because it is automatically called by CLR.

-      Static constructor will not have any access modifiers.

-      Static constructor will execute automatically whenever we create first instance of class

-      Only one static constructor will allowed.

**Private Constructor**

Private constructor is a special instance constructor used in a class that contains static member only. If a class has one or more private constructor and no public constructor then other classes is not allowed to create instance of this class this mean we can neither create the object of the class nor it can be inherit by other class. The main purpose of creating private constructor is used to restrict the class from being instantiated when it contains every member as static.

|  |
| --- |
| using System;  namespace ConsoleApplication3  {  public class Sample  {  public string param1, param2;  public Sample(string a,string b)  {  param1 = a;  param2 = b;  }  private Sample()  // Private Constructor Declaration  {  Console.WriteLine("Private Constructor with no prameters");  }  }  class Program  {  static void Main(string[] args)  {  // Here we don't have chance to create instace for private constructor  Sample obj = new Sample("Welcome","to Aspdotnet-Suresh");  Console.WriteLine(obj.param1 +" " + obj.param2);  Console.ReadLine();  }  }  } |

**Output**

|  |
| --- |
| Welcome to Aspdotnet-Suresh |

In above method we can create object of class with parameters will work fine. If create object of class without parameters it will not allow us create.

|  |
| --- |
| // it will works fine  Sample obj = new Sample("Welcome","to Aspdotnet-Suresh");  // it will not work because of inaccessability  Sample obj=new Sample(); |

**Important points of private constructor**

-      One use of private construct is when we have only static member.

-      Once we provide a constructor that is either private or public or any, the compiler will not allow us to add public constructor without parameters to the class.

-      If we want to create object of class even if we have private constructors then we need to have public constructor along with private constructor