#.Net

C# is a new computer programming language developed by Microsoft Corporation, USA. C# is a fully object-oriented language like Java and is the first Component oriented language. It has been designed to support the key features of .NET framework, the new development platform of Microsoft for building component-based software solutions. It is a simple, efficient, productive and type-safe language derived from the popular C and C++ languages. Although it belongs to the family of C/C++, it is purely object-oriented, modern language suitable for developing Web-based applications.

C# is designed for building robust, reliable and durable components to handle real-world applications.

Characteristics of C#

1) Simple : C# simplifies C++ by eliminating irksome operators such as ->, :: and pointers. C# treats integer and Boolean data types as two entirely different types. This means that the use of = in place of == in if statements will be caught by the compiler.

2) Consistent : C# supports an unified type system which eliminates the problem of varying ranges of integer types. All types are treated as objects and developers can extend the type system simply and easily.

3) Modern : It is modern language due to a number of features it supports like Automatic garbage collection, Rich intrinsic model for error handling, Decimal data type for financial applications, Modern approach to debugging and Robust security model

4) Object-Oriented : C# is truly object-oriented. It supports all the three tenets of object-oriented systems, namely, Encapsulation, Inheritance, Polymorphism. The entire C# class model is built on top of the Virtual Object System (VOS) of the .NET framework. In C# everything is an object. There are no more global functions, variables and constants.

5) Type-safe : It promotes robust programs. C# incorporates a number of type-safe measures.
   - All dynamically allocated objects and arrays are initialized to zero.
   - Use of any uninitialised variables produces an error message by the compiler.
   - Access to arrays are range-checked and warned if it goes out-of-bounds.
   - C# does not permit unsafe casts.
   - C# enforces overflow checking in arithmetic operations.
   - Reference parameters that are passed are type-safe.
   - C# supports automatic garbage collection.

6) Versionable : Making new versions of software modules work with the existing applications is known as versioning. C# provides support for versioning with the help of new and override keywords. With this support, a programmer can guarantee that his new class library will maintain binary compatibility with existing client applications.

7) Compatible : C# enforces the .NET common language specifications and therefore allows inter-operation with other .NET languages.

   - C# provides support for transparent access to standard COM and OLE automation.
   - C# also permits interoperation with C-style APIs.

8) Flexible : Although C# does not support pointers, we may declare certain classes and methods as 'unsafe' and then use pointers to manipulate them. However, these codes will not be type-safe.

9) Inter-operability : C# provides support for using COM objects, no matter what language was used to author them. C# also supports a special feature that enables a program to call out any native API.

Features of C# or new in C#

1. It follows OOPS.
2. It is RAD.
3. Both windows & web applications can be created.
4. Derived from C++.
5. Productivity of VB, elegance of Java and the power of C++.
6. C# is submitted for ratification by ECMA.
7. C# allows native code usage. i.e. C++ code can be used in C#.

New Concepts in C#

1. Common Language Runtime
2. Intermediate Language (IL)
3. Code Management
4. Just In Time Compilation
5. .Net base class library

Applications of C#

1. Console Applications
2. Windows Applications
3. Web Services & Applications
4. ASP.NET
5. Web Services
6. Components
7. Windows & Web Controls

Differences between C# & C++

1. C# supports native code but it has to be managed by the programmer.
2. C# is simpler than C++.
3. C# is safer than C++.
4. Tight syntax checking
5. No Macros, Pointers, Switch Case, Memory leaks

Differences between C# & VB.NET

1. VB.Net supports parameterized constructors
2. VB.Net supports operator overloading.
3. VB.Net supports attributes.

Differences between C# & Java

1. Both derived from C++.
2. C# suports Operator Overloading.
3. C# code can interoperate with code written in other .NET language.

NOTE :

1. The C# Program starts execution from Main() function and the first letter shall be always Capital.
2. Each and every statement should end with ;
3. C# is structured & strongly typed language.
4. .cs : Extension Name for C# Files.

It is pure object oriented programming language. Also it is Event Driven Programming Language.

It is case sensitive.

It is part of Microsoft Visual Studio .Net.

Microsoft Visual Studio .Net comprises of the following :

1) VB.Net
2) C#.Net
3) ASP.Net
4) VC++.Net
5) VJ++.Net

.NET Framework

The .NET framework is a collection of common services provided to perform various tasks or achieve various goals. These may include

1) Designing,
2) Developing,
3) Deploying,
4) Debugging,
5) Run applications

1) .NET Framework software
   a) BCL & FCL (Base class Library & Framework class Library)

2) .Net Framework Tools
   a) Visual Studio .NET
   b) Web matrix (specific for web applications)

BCL :

→ Collection of built-in library Files
→ Specially designed for a particular language
→ Contains those files only which are required by the specific language and not others.

FCL

→ Collection of built-in library files.
→ Designed to be utilized by all language
→ Contains only those files which are required by all the tools.

Goals of .NET Framework

1. Unite isolated web application. (Connecting various web applications using a Web Service, so that data can be shared by different users.)

2. Make information available every time, anytime.

3. Simplify development & deployment.

How does .NET achieve the goals (Features of .NET) :

1. Web Services (Web services are called as Health storm service thru which ones data can be utilized by other people without re-entry of the data of the client). The centre of the .NET architecture.

2. ADO.Net Datasets & XML support through out the platform (Allows access to disconnected distributed databases. It creates Dataset containing database schemas).

3. Rich tools, runtime services & XCOPY deployment. (Because of XCOPY deployment no need to stop the server to deploy the upgraded application. The logged-in clients will get the old information and new clients will get the new information.)

4. In .NET framework common data types are defined using this data type you can share data between different applications and Microsoft has told to all the software vendors to develop compilers according to these common data types, so that their compilers become .NET Complaint.

5. Threading is built-in.

6. Cross language inheritance is one of the best feature of .NET.

7. Under .NET after compilation all languages will generate Intermediate Language Code (IL).

8. Visual Studio.NET IDE is the GUI editor for all .NET languages.

9. Cross language debugging is also possible.

10. Profiling of applications also possible.

11. No need of registration, GUIDs.

12. Multiple versions of the same component can be run side-by-side.

13. No “DLL Hell”.

14. Applications will run directly from CD. However, transferring project is not possible.

15. Applications install only the core logic.

16. No need to install runtime libraries.

17. No need to copy the .dll files with the code.

18. XCOPY deployment : Made possible with the introduction of metadata.

19. Metadata : It is generated by compiler and stored automatically in binary format. It contains Name, Version, Culture, Public key, Types exposed by the assembly, Dependencies on other assemblies, Security permissions needed to run, Base classes & interfaces used by the assembly, Custom attributes (User, Compiler defined).

20. Metadata is used by Designers, Debuggers, Profilers, Proxy Generators, Object browsers.

.NET is not platform independent.

Features of .Net

1) Object Oriented Paradigm.
   a) Classes
   b) Objects
   c) Data Abstraction
   d) Encapsulation
   e) Inheritance
   f) Polymorphism
   g) Interfaces

2) Common Language Runtime (CLR)
3) Common Language Specification (CLS) → Common Language Specifications for Cross Language Implementation.
4) Common Type System (CTS) → For compatible data type support.
5) Garbage Collection
6) Multithreading
7) Interoperability
8) Cross Language Inheritance
9) Cross Language Debugging
10) Web Applications (ASP.NET)
11) Connections to Data Objects (ADO.NET)

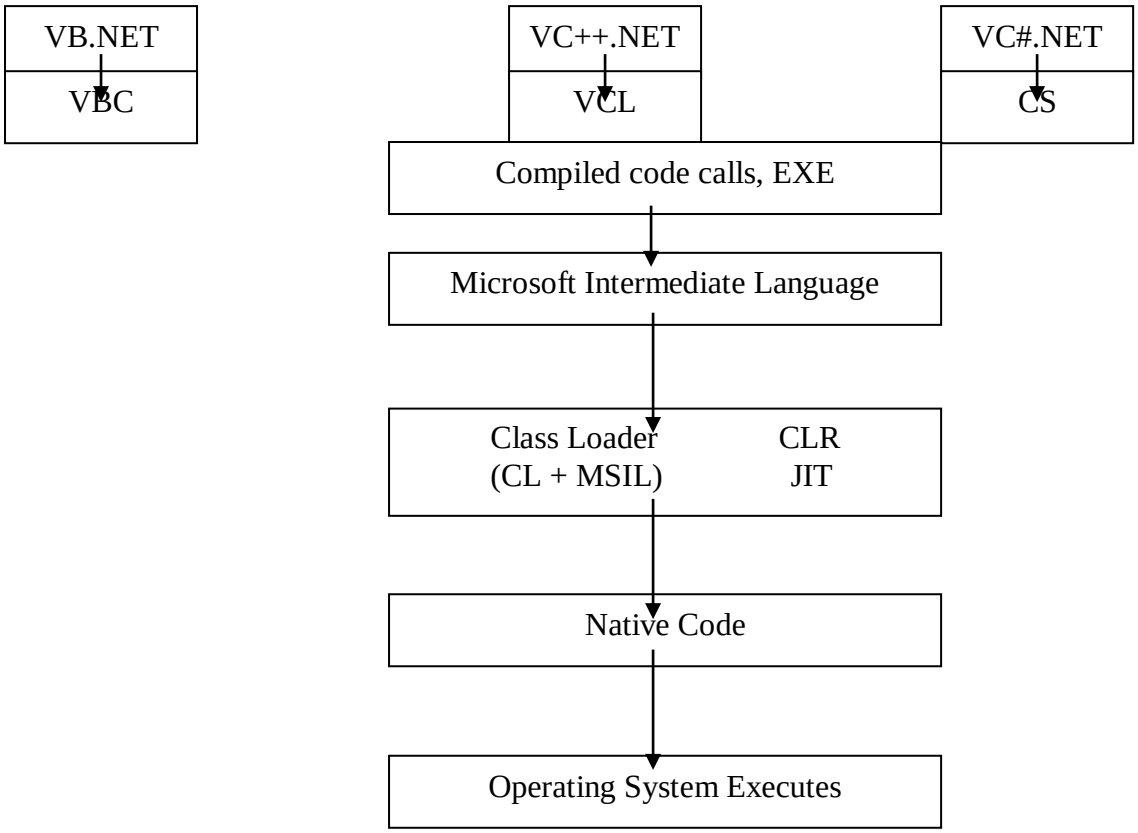
Types of applications

1) Console Applications
2) Windows Applications
3) Web Applications
4) Mobile Applications
5) Smart Device Applications

Compiling .Net Programs

Runtime Engine (CLR → Common Language Runtime) : It converts the .Net specific IL (Intermediate Language) Code into machine specific code (Native code).

Conversion & Compilation :



Console Applications

Structure of C# Program :

```
using namespacename;
namespace namespacename
{
        class classname
        {
                static void Main(string[] args)
                {
                        Declaration statements
                        Input statements
                        Processing statements
                        Output statements
                }
        }
}
```

Input statements

1) Console.ReadLine() / Console.Read() : Reads one line of data from keyboard.

Syntax : variable=Console.ReadLine();

Example : a=Console.ReadLine();

Output statements

1) Console.Write() : Displays the data on screen and cursor stays on the same line.

Example :
```
Console.Write(15);
Console.Write("Enter any no");
Console.Write("The no is " +a);
```

2) Console.WriteLine() : Displays the data on screen and cursor goes to next line.

Example :
```
Console.WriteLine(15)
Console.WriteLine("Enter any no")
Console.WriteLine("The no is " +a)
```

System.Console.WriteLine("{0},{1},{2}",variabel1/data1,variable2/data2);

Displays the data or value of variable on the screen. Here we have to specify a no. in { } separated by comma starting from 0. Each number will be replaced with the variable value or data.

Declaring Variables :

data type variable1, variable2……v ariableN;

Data Types :

1) byte 2) short 3) int 4) long 5) float 6) double
7) bool 8) char 9) string

Remark Statements

1) Single line

// statement

2) Multi-line

```
/*
statements
statements
statements
statements
/*
```

Escape Sequences and Strings :

\a : alert
\b : back space
\f : form feed
\n : New Line
\r : carriage return
\t : Horizontal tab
\v : Vertical tab
\' : Displays single quote

\" : Displays double quote
\\ : Displays \
\0 : null

Using aliases for namespace classes :

Syntax : using alias-name=class-name;

Example : using A=System.Console;

Command line arguments

We can also pass arguments to a C# program at run time and they are called as command line arguments.

```
class abc
{
        public static void main(string[] args)
        {
                Console.WriteLine("The arguments are : " + args[0]);
        }
}
```

System.Console.WriteLine(@ "string1\nstring2\tstring3") : Displays the string as it is without replacing escape sequence characters.

This type of string is called as Verbatim string.

We can't write a string in two lines like the below :

System.Console.WriteLine("hi Bye");

If you do C# gives an error. To write the statement as above and without getting error, write the statement as below :

System.Console.WriteLine(@"hi Bye ");

Type Casting

char a=(char) 65; -> stores A in variable a.

int p=(int) 'D'; -> stores 68 in variable p.

C# understands Unicode and thus the char data type store characters internally as Unicode. Previously in every language only ASCII characters are used but with this there is a limitation of 256 characters only to avoid this limitation Unicode character set was created.

```
int a;      long b; short c;byte d; char e;
string f;   float g=90.0F; double h;

Console.WriteLine ("Enter any no ");
a=int.Parse(Console.ReadLine());
```

```
                                                    or
a=System.Int32.Parse(Console.ReadLine());

Console.WriteLine ("Enter any no ");
b=long.Parse (Console.ReadLine());
                                                    or
b=System.Int64.Parse(Console.ReadLine());

Console.WriteLine ("Enter any no ");
c=short.Parse (Console.ReadLine());
                                                    or
c=System.Int16.Parse (Console.ReadLine());

Console.WriteLine ("Enter any no upto byte range ");
d=byte.Parse (Console.ReadLine());
                                                    or
d=System.Byte.Parse (Console.ReadLine());


Console.WriteLine ("Enter any character ");
e=char.Parse (Console.ReadLine());
                                                    or
e=System.Char.Parse (Console.ReadLine());

Console.WriteLine ("Enter any string ");
f=Console.ReadLine();

Console.WriteLine ("Enter any no ");
g=float.Parse (Console.ReadLine());
                                                    or
g=System.Single.Parse (Console.ReadLine());

Console.WriteLine ("Enter any no ");
h=double.Parse(Console.ReadLine());
                                                    or
h=System.Double.Parse(Console.ReadLine());
```

Boxing

Any type, value or reference can be assigned to an object without an explicit conversion. When the compiler finds a value type where it needs a reference type, it creates an object “box” into which it places the value of the value type.

Example :

```
int m=100;
object m1=m;  // creates a box to hold m.
```

When executed, this code creates a temporary reference_type “box” for the object on heap. We can also use a C-style cast for boxing :

```
int m=100;
object m1=(object) m;
```

Note that the boxing operation creates a copy of the value of the m integer to the object m1. Now both the variables m and m1 exist but the value of m1 resides on the heap. This means that the values are independent of each other.

```
int m=100;
object m1=m;  // creates a box to hold m.
m=20;
Console.WriteLine("m="+m);          // wii give output as 20.
Console.WriteLine("m1="+m1);        // wii give output as 100.
```

When a code changes the value of variable m, the value of m1 is not affected.

Unboxing

Unboxing is the process of converting the object type back to the value type. Remember that we can only unbox a variable that has previously been boxed. In contrast to boxing, unboxing is an explicit operation using C-style casting.

```
int m=10;
object m1=m;  // box m
int n=(int)m1;  // unboxing m1 back to int
```

When performing unboxing, C# checks that the value type we request is actually stored in the object under conversion. Only if it is, the value is unboxed. Also we have to ensure that the value type is large enough to hold the value of the object. Otherwise, the operation may result in a runtime error. For example, the code

```
int m=500;
object m1=m;
byte n=(byte)m1;
```

will produce runtime error.

Using Reserved/Keywords as variables

Specify @before the keyword.

```
int @if;
for (@if=1;@if<=5;@if++)
        A.WriteLine (@if );
```

Operators

1) Arithmetic : +, -, *, /
2) Logical : AND, OR, NOT
3) Relational : >, <, ==, !=, >=, <=

```
string a="bye", b="bye";
System.Console.WriteLine(a==b); returns true.
```

Conditional Statements

1)
```
if (condition)
        statement;
```

2)
```
if (condition)
```

```
        {
                statement1;
                statement2;
        }

3)      if (condition)
                Statement;
        else
                statement;

4)      if (condition)
        {
                statement1;
                statement2;
                statementN;
        }
        else
        {
                statement1;
                statement2;
                statementN;
        }
```

switch..case :

```
switch (variable)
{
        case expression :
        {
                statements;
                break;
        }
        case expression :
        {
                statements;
                break;
        }
        default :
        {
                statements;
        }
}
```

Control Statements (Loops)

1) for loop

```
        for (initialization;condition;increment/decrement)
        {
                statement1;
                statement2;
                statementN;
                [break;] [continue;]
```

```
}
```

Example :

```
for (int I=1;I<=10;I++)
        System.Console.WriteLine("{0}",I);
```

2) while loop

```
while (condition)
{
        statement1;
        statement2;
        statementN;
        [break;] [continue;]
}
```

Example :

```
int I=1;
while (I<=10)
{
        System.Console.WriteLine("{0}",I);
        I++;
}
```

3) do..while loop

```
do
{
        statement1;
        statement2;
        statementN;
        [break;] [continue;]
}
while (condition);
```

Example :

```
int I=1;
do
{
        System.Console.WriteLine("{0}",I);
        I++;
}
while (I<=10);
```

4) foreach

```
foreach(datatype variable in arrayname or objectname)
{
        Statements;
}
```

Arrays

An array is a group of similar elements having logical relationship. Arrays are two types.

1) Single Dimensional 2) Two Dimensional

Arrays dimension starts from 0 for single dimensional array and 0,0 for two dimensional array.

Declaration :

1) datatype arrayvariable[]=new datatype[no. of cells];
2) datatype arrayvariable[][]=new datatype[no. of rows][no. of cols];

Example :
1) int[] a=new int[3];
2) int[][] a=new int[3][3];

Storing values into array :

1) arrayvariable[cell no]=data/variable
2) arrayvariable[row no][col no]=data/variable

Example :
1) a[0]=13;
2) a[1][2]=33;
3) int[] a=new int[3] {9,8,7};
4) int[] b={9,7,8};

Printing the array :

1) System.Console.WriteLine(array variable[cell no]);
2) System.Console.WriteLine(array variable[row no][col no]);

Example :
1) System.Console.WriteLine(a[0]);
2) System.Console.WriteLine(a[1][2]);

Creating Functions

access modifier return type function name(data type variable1, data type variable2, data type variable3….argN)

```
{
        statement1;
        statement2;
        statement3;
        return data/variable;
}
```

Example :

```
static void abc()
{
        System.Console.WriteLine("Welcome to C#");
}
```

Calling Functions

1) variable = function name([arg1,arg2…argN]);
2) function name([arg1,arg2…argN]);
3) To call a function present in another class specify class name.function name(args).
4) To call a function present in another namespace specify namespacename.classname.function name(args).

Parameters Passing to Function

We can pass parameters to functions in 2 ways. Using **out & ref.**

out : While sending the data of variable **prefix the variable with out** and in the same way **prefix with out** at function declaration. When you say out it means whatever changes you will make in a function, they will be visible outside the function also.

For e.g. :

```
public static void Main(String a[])
{
        int p=20;
        sum(out p);
        System.Console.WriteLine(p);
}

public void sum(out int p)
{
        p=50;
}
```

Even though we stored 20 in p output will be 50 because we are telling to complier that whatever changes you make to variable p in function sum should available in function Main.

We can write the above program using **ref** keyword also.

Difference between **out & ref** is :

Using **out** without initialising we can send variable to a function.
Using **ref** without initialising we can't send variable to a function. If send you will get error.

Actually **out** key word is used to write data to a variable from one function to another function and **ref** keyword is used for reading and writing data of variable between two functions.

It is must and should to write a value in variable before leaving the function otherwise you get error.

Namespaces

It is just like package of java. It is used to group logically various classes. Collection of classes is called namespace. They are two types.

1) User-defined namespace (e.g. System)
2) pre-defined namespace

Creating Namespace

```
namespace namespace name
{
    class class name
    {
        access modifier return type function name(data type variable1, data type variable2, data type variable3….argN)
        {
            statement1;
            statement2;
            statement3;
            return data/variable;
        }
    }
}
```

Calling function available in namespace

namespace name.class name.function name(arguments);

Example :

```
namespace sekhar
{
    class asset
    {
        static void totstudents()
        {
            System.Console.WriteLine(35);
        }
    }
}
```

Calling the above namespace in below program

```
class abc
{
    static void Main()
    {
        sekhar.asset.totstudents();
    }
}
```

Importing namespace

using namespacename;

If you import the namespace no need to write everytime while calling the function the namespace name.

We can even nest the namespaces upto the level you desire. For e.g.

```
namespace sekhar
{
        namespace shailesh
        {
                class abc
                {
                        static void print()
                        {
                                System.Console.WriteLine("ASSET");
                        }
                }
        }
}
```

To use the function print() you have to write : sekhar.shailesh.abc.print();

We can also write the above thing in the below manner also.

```
namespace sekhar.shailesh
{
        class abc
        {
                static void print()
                {
                        System.Console.WriteLine("ASSET");
                }
        }
}
```

To use the function print() you have to write : sekhar.shailesh.abc.print();

Object Oriented Programming

Object Oriented Paradigm (OOP) features :

1) Class : A class is a template or blueprint that defines an objects attributes, operations and that is created at design time.

It contains set of functions to access, manipulate the data and a set of access restrictions on the data and on the functions.

2) Object : An object is a running instance of a class that consumes memory and has a finite life span. It is instance of class.

Car

Attributes	Operations (methods)	
No. of wheels, color, make, model	Unlock door, Opendoor,Start engine	
Objects exhibit 3 characteristics :		

1) Identity : One object must be distinguishable from another object of the same class.
2) Behavior
3) State : It refers to the attributes or information that an object stores. These attributes are often manipulated by an objects operations.

3) Encapsulation : Encapsulation is the process of hiding the details about how an object performs its duties when asked to perform those duties by a client.

4) Abstraction : Is the practice of focusing only on the essential aspects of an object.

5) Inheritance (Reusability) : It is the concept of reusing common attributes and operations from a base class in a derived class.

6) Interfaces : Interfaces are similar to abstract classes. They define the method signatures used by other classes but do not implement any code themselves.

7) Polymorphism : It is the ability to call the same method on multiple objects that have been instantiated from different sub classes and general differing behavior.

Creating Class

Type 1 :

```
namespace namespacename
{
        class classname
        {
                variable declaration;
                methods();
        }
}
```

Type 2 :

```
namespace namespacename
{
        class classname
        {
                variable declaration;
                methods();
        }

        static void Main(string[] args)
        {
        }
}
```

Creating Object

```
classname objectname = new classname(arg1,arg2,argN);
```

Calling Static method or variable

```
classname.method name();
classname.variable=data;
```

Calling Non-Static method or variable

First create object of the class then use that object while calling method or variable.

```
objectname.methodname();
objectname.variable=data;
```

Example :

```
using System;
namespace n1
class abc1
{
        int a,b,c=0;
        public void input()
        {
                Console.WriteLine("Enter first no ");
                a=int.Parse(Console.ReadLine());
                Console.WriteLine("Enter second no ");
                b=int.Parse(Console.ReadLine());
        }
        public void calc()
        {
                c=a+b;
        }
        public void print()
```

```
        {
                Console.WriteLine("Sum of two nos. "+c);
        }
}
using System;
namespace n1
class abc2
{
        static void Main(string[] args)
        {
                abc1 a1=new abc1();
                a1.input();
                a1.calc();
                a1.print();
                Console.ReadLine();
        }
}
```

Constructor

A constructor looks just like function but works in different way, whenever an object is created the constructor shall be called automatically and what ever written in the constructor is executed. The keyword new first allocates memory for the functions and the variables. After this it calls the constructor.

A constructor can be used in cases where every time an object gets created and you want some code to be automatically executed.

Example :

```
class abc
{
        public abc()
        {
                System.Console.WriteLine("asset");
        }
}
```

We can also send parameters to the constructor. We can also write 'N' no. of constructors as desired by you.

```
class abc
{
        public abc(string s)
        {
                System.Console.WriteLine("asset"+s);
        }
}
```

Private constructors

C# does not define global variables or constants. All declarations must be contained in a class. In many situations, we may wish to define some utility classes that contain only static members.

Such classes are never required to instantiate objects. Creating objects of such classes may be prevented by adding private constructor to the class.

Copy constructor

A copy constructor creates an object by copying variables from another object. For example, we may wish to pass an Item object to the Item constructor so that the new Item object has to same values as the old one.

Since C# does not provide a copy constructor we must provide it ourselves if we wish to add this feature to the class. A copy constructor is defined as follows :

```
public Item()
{
}
public Item(Item item1)
{
        code=item1.code;
        price=item1.price;
}
public Item(int c, float d)
{
        code=c;
        price=d;
}
```

The copy constructor is invoked by instantiating an object of type Item and passing it the object to be copied.

Example :

```
Item item1=new Item(15,55.56);
Item item2=new Item(item1);
```

Static constructor

A static constructor is called before any object of the class is created. This is useful to do any housekeeping work that needs to be done once. It is usually used to assign initial values to static data members.

```
static Item()
{
}
```

Creating Destructor

You can controls what happens during the destruction of objects by using procedures called destructors.

Destructor use : Deallocate resources, disconnect from the internet, inform other objects that the current object is going to be destroyed.

```
~class name()
{
        statements;
}
```

Example :

```
~abc()
{
        Console.WriteLine("Destructor is called");
}
```

Polymorphism

1) Method/Function Overloading
2) Method/Function Overriding
3) Operator Overloading

1) Method/Function Overloading

If we write several functions with same name having different signatures, then it is called Function Overloading.

The function signature includes

1) different type of arguments or
2) different no. of arguments or
3) different order of arguments

Note : Function signature does not include return type.

Example :

```
using System;
namespace n2
{
        class overload
        {
                public int sum(int a, int b)
                {
                        return a+b;
                }
                public long sum(long a, long b)
                {
                        return a+b;
                }
                public double sum(double a, double b)
                {
                        return a+b;
                }
```

```
        public string sum(string a, string b)
        {
            return a+b;
        }
        static void Main(string[] args)
        {
            overload o1=new overload();
            Console.WriteLine(o1.sum(15,20));
            Console.WriteLine(o1.sum(15324324,203243243));
            Console.WriteLine(o1.sum("sekhar ","rao"));
        }
    }

}
```

Method/Function Overriding

If you re-write base class function in derived class it is called function overriding.

1) Define the base class function using virtual keyword.
2) Define the derived class function using override keyword.

Example :

```
public class class1
{
    public virtual void display()
    {
        Console.WriteLine("SSI");
    }
}
public class class2 : class1
{
    public override void display()
    {
        Console.WriteLine("APTECH");
    }
}
```

Hiding method

Some times we don't want to use base class method or we don't want to override it, at that time we can hide that method in derived class.

Example :

```
public class class1
{
    public void display()
    {
        Console.WriteLine("SSI");
    }
}

public class class2 : class1
{
```

```
        public new void display()
        {
                Console.WriteLine("APTECH");
        }
}
```

Define the base class method simple.
Define the derived class method using new keyword.

Abstract classes

In a number of hierarchical applications, we would have one base class and a number of different derived classes. The top-most base class simply acts as a base for others and is not useful on its own. In such situations, we might not want any one to create its objects. We can do this by making base class abstract.

```
abstract class abc
{
        declarations;
        methods();
}
```

Abstract methods

Similar to abstract classes, we can also create abstract methods. An abstract method is implicitly a virtual method and does not provide any implementation.

Example :

```
public abstract void display();
```

After inheriting the class in which abstract method is there we have to override the abstract function, otherwise the class become abstract class.

Sealed Classes

A class that can't be sub-classed is called sealed class. It is needed when a class to be prevented being further sub-classed for security reasons.

```
sealed class abc
{

}
```

A sealed class cannot also be an abstract class.

Sealed methods

To prevent overriding of methods in derived classes we can create sealed methods.

```
class A
{
        public virtual void fun()
        {
        }
}
```

```
class B : A
{
        public sealed override void fun()
        {
        }
}
```

Now you can't override method fun() in a class which is derived from class B.

Operator Overloading

Although operator overloading gives us syntactical convenience, it also help us greatly to generate more readable and intuitive code in a number of situations. These include :

Mathematical or physical modeling where we use classes to represent objects such as co-ordinates, vectors, matrices, tensors, complex numbers and so on.

Graphical programs where co-ordinate-related objects are used to represent positions on the screen.

Financial programs where a class represents an amount of money.

Text manipulations where classes are used to represent strings and sentences.

Overloadable operators

Binary arithmetic : +, *, /, -, %,
Unary arithmetic : +, -, ++, --
Binary bitwise : &, |, ^, <<, >>
Unary bitwise : true, false, !, ~
Logical operators : ==, !=, >=, <=, >, <

Operators that can't be overloaded

Conditional operators : &&, ||
Compound assignments : +=, -=, *=, /=, %=
Other operators : [ ], { }, =, ?:, ->, new, sizeof, typeof, is, as

Syntax :

```
public  static returntype operator operatorsymbol(arg1, arg2, argN)
{
        Statements;
}
```

Interfaces

An interface can contain one or more methods, properties and events but none of them are implemented in the interface itself. It is the responsibility of the class that implements the interface to define the code for implementation of these members.

Syntax :

```
interface abc
{
        void display();
}
```

Using interface in a class

```
class bbc : abc
{
        void display()
        {
                Console.WriteLine("APTECH");
        }
}
```

Inheriting an interface

```
interface bbc : abc
{
        void show();
}

interface ccc
{
        void list();
}
```

Using multiple interfaces

```
class ddd : abc, ccc
{
        statements;
}
```

Nesting of classes

Class in a class is called nested class.

```
public class outer
{
        public class inner
        {
                Statements;
        }
        Statements;
}
```

Creating object of outer class

```
outer o1=new outer();
```

Creating object of inner class :

```
outer.inner o2=new outer.inner();
```

Constant members

C# allows you to create constant members whose value will not be changed in the program anywhere once it is initialized with a value.

```
public const int a=100;
public const int b;          error
a=90;                                  error
```

Read-only members

C# allows you to create read-only members whose value will not be changed in the program anywhere. once it is initialized with a value.

```
public readonly int a;
```

Inside the class constructor or in any function we can initialize variable a with a value.

Creating Property

```
int a2;
public int an1
{
        get
        {
                return a2;
        }
        set
        {
                a2=value;
        }
}
```

Storing value in property

```
objname.property=data;
```

In OOP's creating a public member variable is dis-allowed because it avoids the Data hiding feature of OOP. To use a private variable we have to create two functions for storing/reading data from the private variable, instead of this we can create a property which has standard functions get (to return data) and set (to store data). Also you may remove get/set functions. If you remove set function it is called read-only property, if you remove get function it is called write-only property (it is rarely used).

Modifiers

1) static : It implies free. It signifies that you can access a member or a function without creating an object. A variable or a method can be static.

2) public : To make a method or constructor public. If we create method or constructor using public that method can be called by any class without creating object.

3) private : Member is accessible only within the class containing the member.

4) protected : Member is visble only to its own class and its derived classes.

5) internal : Member is available within the assembly or component that is being created but not to the clients of that component.

6) protected internal : Available in the containing program or assembly and in the derived classes.

Keyword	Containing class	Derived class	Containing program	Anywhere outside the containing program
private	√			
protected	√	√		
internal	√		√	
protected internal	√	√	√	
public	√	√	√	√

Delegates

The dictionary meaning of delegate is "a person acting for another person". In C#, it really means a method acting for another method. As pointed out earler, a delegate in C# is a class type object and is used to invoke a method that has been encapsulated into it at the time of its creation. Creating and using delegate involves four steps. They are,

Delegate declaration :

Syntax :

public/private/protected delegate void delegatename([arg1,arg2,argN]);

Example :
public delegate void show();
public delegate void show1(int x, int y);

Delegate methods

The method whose references are encapsulated into a delegate instance are known as delegate methods or callable entities. The signature and return type of delegate methods must exactly match the signature and return type of the delegate.

Example :

```
void display()
{
        statements;
}
void sum(int a, int b)
{
        statements;
}
```

Delegate instantiation

Delegates are of class types and behave like classes, they are also instantiated just like class.

delegatename objectname=new delegatename(methodname);

Delegate invocation

objectname([arg1,arg2,argN]);

Threading

C# supports concept of multithreading which enables us to execute two or more "parts" of a program concurrently. Each part is known as a thread. A thread is basically a separate sequence of instructions designed for performing a "specific task" in the program. A task may represent an operation such as reading data, sending a file over the Internet, printing some results or performing certain calculations. Multithreading, means performing multiple tasks at the same time during the execution of a program.

The execution of a C# program starts with a single thread called the main thread that is automatically run by the Common Language Runtime (CLR) and the operating system. From the main thread, we can execute other threads for performing desired tasks in the program. The process of developing a program for execution with multiple threads is called multithreaded programming and the process of execution is called multithreading.

The main advantage of multithreading is that it enables us to develop efficient programs that could optimize the use of computer resources such as memory, I/O devices and time.

C# defines a namespace known as System.Threading containing classes and interfaces that are required for developing and running multithreaded programs. The following are the classes :

1) Thread : It helps us to perform task such as creating and setting the priority of a thread. We can use this class to control a thread and obtain its status. It provides the following properties :

CurrentThread : To retrieve the name of the thread, which is currently running.

IsAlive : To retrieve a value to indicate the current state of thread execution. The value of the IsAlive property is true if the thread has been started or has not terminated, otherwise the value is false.

IsThreadPoolThread : Returns true if the thread is part of a thread pool, otherwise false.

Name : To specify a name for a thread.

Priority: To obtain a value, which indicates the scheduling priority of a thread. By default, the value is Normal. We can assign either Highest, AboveNormal, BelowNormal, Normal or Lowest. The Highest value indicates that the thread must be scheduled for running before any other thread.

ThreadState : Returns state of a thread. By default, the value of the ThreadState property is Unstarted. The values can be Running, Stopped, Suspended, Unstarted or WaitSleepJoin.

The Thread class also provides the following methods :

Interrupt : To interrupt the thread, which is in the WaitSleepJoin state.

Join : To block a thread until another thread has terminated.

Resume : To resume a thread, which has been suspended earlier.

Sleep : To block the current thread for a specified time period.

SpinWait : To make a thread wait the number of times specified in Iterations parameter.

Start : To start a thread.

Suspend : To suspend a thread.

2) ThreadPool : It provides a pool of threads that help us to perform tasks such as processing of asynchronous I/O and waiting on behalf of another thread. The following are the methods :

Equals : To determine whether two thread pools are equal or not.

GetType : To obtain the type for the current thread pool.

QueueUserWorkItem : To allow a method to be queued for execution. The method, which has been queued, executes when a thread pool thread is made available.

SetMaxThreads : To specify the number of requests to the thread pool that can be concurrently active.

SetMinThreads : To specify the number of idle threads that can be maintained by a thread pool for new requests.

3) Monitor : It provides control access to an object by granting a lock for the object to a single thread. When an object is locked for a thread then access to a specific program code is restricted. The Monitor class defines a few methods that allows us to perform tasks such as acquiring and releasing a lock for an object.

Enter : Allows a thread to obtain a lock on a specified object.

Exit : Allows a thread to release a lock on a specified object.

TryEnter : Allows a thread to try and obtain a lock on a specified object.

Wait : Allows a thread to release the lock on an object and block the thread for the time period until which it again acquires the lock.

GetType : Allows us to obtain the type for the current instances of the Monitor class.

Theread Pooling : In this a thread pool is created to perform multiple tasks simultaneously. A thread pool is basically a group of threads that can be run simultaneously to perform a number of tasks in the background. This feature of C# is mainly used in server applications. In server applications, a main thread receives the request from the client computers and passes it to a thread in the thread pool for processing of the request. In this manner, the main thread functions asynchronously and is free to receive the requests from client computers. The main thread does not have to process the request. Instead, the request is transferred to a thread in the thread pool for processing. A delay in receiving the requests from the client computer does not occur because of implementation of thread pooling in server applications. After the thread in the thread pool completes the task of processing the client request, it waits in a queue for performing another task. In this way, the thread in the thread pool can be reused for performing different tasks. The reusability of the thread because of thread pooling enables a server application to avoid creating a new thread for every task.

System.Array Class

In C#,every array we create is automatically derived from the System.Array class.This class defines a number of methods and properties that can be used methods to manipulate arrays more efficiently. Table lists some of the commonly used methods and their purpose.

Method/Property	Purpose
Clear()	Sets a range of elements to empty values.
Copy to()	Copies elements from the source array into the destination array
Getlength()	Gives the number of elements in a given dimension of the array.
Getvalue()	Gets the value for a given index in the array
Length()	Gives the length of the array.
Setvalue()	Sets the value for a given index in the array
Reverse()	Reverses the contents of a 1-d array
Sort()	Sorts the elements in a 1-d array.

```
int[] a=int[5];
a[0]=20; a[1]=12; a[2]=5; a[3]=4;
System.Array.Sort(a);
```

ArrayList Class

System.Collections namespace defines a class known as Arraylist that can store a dynamically sized array of objects. The Arraylist class includes a number of methods to support operations such as sorting, removing and enumerating its contents. It also supports a property Count that gives the number of objects in an array list and a property Capacity to modify or read the capacity.

An array list very similar to an array, except that it has the ability to grow dynamically. We can create an array list by indicating the initial capacity we want.

Example :

```
ArrayList cities = new ArrayList(30);
```

It creates cities with a capacity to store thirty objects. If we do not specify the size, it defaults to sixteen. i.e.

```
ArrayList cities = new ArrayList();
```

will create a cities list with the capacity to store sixteen objects. We can now add elements to the list using the Add() method:

```
cities.Add("Bombay");
cities.Add("Anand");
```

We can also remove an element:

```
cities.RemoveAt(1);
```

This will remove the object in position1. We can modify the capacity of the list using the property Capacity:

```
cities.Capacity=20;
```

We may obtain the actual number of objects present in the list using the property Count as follows:

```
int n = cities.Count;
```

An array list can be really useful if we need to create and array of objects but we do not know in advance how big the array would be. Futher, an array list can contain any object reference.

Table lists some of the most important methods and properties supported by the ArrayList class.

Method/Property	Purpose
Add()	Adds an object to a list
Clear()	Remove all the elements from the list
Contains()	Determines if an element is in the list
CopyTo()	Copies a list to another
Insert()	Inserts an element into the list
Remove()	Removes the first occurrence of an element
RemoveAt()	Removes the element at a specified place
RemoveRange()	Removes a range of elements
Sort()	Sorts the elements
Capacity	Gets or sets the number of elements in the list
Count	Gets the number of elements currently in the list

String Class methods

Method/Property	Operation
Compare()	Compares two strings
CompareTo()	Compares the current instance with another instance.
ConCat()	Concatenates two or more strings
Copy()	Creates a new string by copying another
CopyTo()	Copies a specified number of characters to an array of Unicode characters.
EndsWith()	Determines whether a substring exists at the end of the string
Equals()	Determine if two strings are equal
IndexOf()	Returns the position of the first occurrence of a substring.
Insert()	Returns a new string with a substring inserted at a specified location
Join()	Joins an array of strings together
LastIndexOf()	Returns the position of the last occurrence of a substring
PadLeft()	Left-aligns the strings in a field
PadRight()	Right-aligns the strings in a field
Remove()	Deletes characters from the string
Replace()	Replaces all instances of a character with a new character

Split()	Creates an array of strings by splitting the string at any occurrence of one or more characters.
StartsWith()	Determines whether a substring exists at the beginning of the string
Substring()	Extracts a substring
ToLower()	Returns a lower-case version of the string
ToUpper()	Returns an upper-case version of the string
Trim()	Removes white space from the string
TrimEnd()	Removes a string of characters from the end of the string
TrimStart()	Removes a string of characters from the beginning of the string

Example :

```
string s=”Aptech”;
s.ToUpper();
```

Exception Handling

Exception handling means Error handling. Errors are two types :

1) Compile time : The errors occur during compile time of a program.
2) Run time : The errors occur during runtime of a program.

Exception handling key words

1) Try : It is used to monitor statements for errors.
2) Catch : It catches error thrown.
3) Finally : This block is executed irrespective of error occurs or not,
4) Throw : To throw user defined exception.

Exception class is the super/base class for all the exception classes.

Exception Classes :

1) DividebyZeroException
2) OverflowException
3) ConstraintException
4) Exception

Catching Errors using try & catch

```
try
{
        statement1;
        statement2;
        statementN;
}
catch(System.Exception/Exception Name Object Name)
{
        statements;
}
finally()
{
```

```
        statements;
}
```

Throwing Errors

throw new System.Exception(); : throws the System.Exception().

object name.ToString() : Converts the contents of the object into a string.

Specify **return** statement in catch block to stop execution of statements present in the function.

.NET Framework : The .NET framework is a collection of common services provided to perform various tasks or achieve various goals. These may include

1) Designing,
2) Developing,
3) Deploying,
4) Debugging,
5) Run applications

3) .NET Framework software
   a) BCL & FCL (Base class Library & Framework class Library)

4) .Net Framework Tools
   a) Visual Studio .NET
   b) Web matrix (specific for web applications)

BCL :

→ Collection of built-in library Files
→ Specially designed for a particular language
→ Contains those files only which are required by the specific language and not others.

FCL

→ Collection of built-in library files.
→ Designed to be utilized by all language
→ Contains only those files which are required by all the tools.

Goals of .NET Framework :

1. Unite isolated web application. (Connecting various web applications using a Web Service, so that data can be shared by different users.)

2. Make information available every time, anytime.

3. Simplify development & deployment.

How does .NET achieve the goals (Features of .NET) :

1. Web Services (Web services are called as Health storm service thru which ones data can be utilized by other people without re-entry of the data of the client). The centre of the .NET architecture.

2. ADO.Net Datasets & XML support through out the platform (Allows access to disconnected distributed databases. It creates Dataset containing database schemas).

3. Rich tools, runtime services & XCOPY deployment. (Because of XCOPY deployment no need to stop the server to deploy the upgraded application. The logged-in clients will get the old information and new clients will get the new information.)

4. In .NET framework common data types are defined using this data type you can share data between different applications and Microsoft has told to all the software vendors to develop compilers according to these common data types, so that their compilers become .NET Complaint.

5. Threading is built-in.

6. Cross language inheritance is one of the best feature of .NET.

7. Under .NET after compilation all languages will generate Intermediate Language Code (IL).

8. Visual Studio.NET IDE is the GUI editor for all .NET languages.

9. Cross language debugging is also possible.

10. No need of registration, GUIDs.

11. Multiple versions of the same component can be run side-by-side.

12. No "DLL Hell".

13. Applications will run directly from CD.

14. Applications install only the core logic.

15. No need to install runtime libraries.

16. No need to copy the .dll files with the code.

17. XCOPY deployment : Made possible with the introduction of metadata.

18. Metadata: It is generated by compiler and stored automatically in binary format. It contains Name, Version, Culture, Public key, Types exposed by the assembly, Dependencies on other assemblies, Security permissions needed to run, Base classes & interfaces used by the assembly, Custom attributes (User, Compiler defined).

19. Metadata is used by Designers, Debuggers, Profilers, Proxy Generators, Object browsers.

.NET is not platform independent.

Features of .Net :

1) Object Oriented Paradigm.
   a) Classes
   b) Objects
   c) Data Abstraction
   d) Encapsulation
   e) Inheritance
   f) Polymorphism
   g) Interfaces

2) Common Language Runtime (CLR)
3) Common Language Specification (CLS) → Common Language Specifications for Cross Language Implementation.
4) Common Type System (CTS) → For compatible data type support.
5) Garbage Collection
6) Multithreading
7) Interoperability
8) Cross Language Inheritance
9) Cross Language Debugging
10) Web Applications (ASP.NET)
11) Connections to Data Objects (ADO.NET)

Types of applications :

1) Console Applications
2) Windows Applications
3) Web Applications
4) Mobile Applications
5) Smart Device Applications