C# and .NET

Unit-1

A. M. Bhave

by Arvind M. Bhave

C# (Introduction)

❖ C# (pronounced "See Sharp") is a simple, modern, object-oriented, and type-safe programming language.

❖ C# has its roots in the C family of languages and will be immediately familiar to C, C++, and Java programmers.

❖ C# is an object-oriented language

❖ C# further includes support for ***component-oriented*** programming.

- The C# language was developed by a small team led by two distinguished Microsoft engineers
- Anders Hejlsberg and Scott Wiltamuth.
- Hejlsberg is also known for creating Turbo Pascal

Several C# features

- **robust and durable**
  - ***Garbage collection*** automatically reclaims memory occupied by unused objects.
  - ***exception handling*** provides a structured and extensible approach to error detection and recovery.
- **type-safe**
  - makes it impossible to have uninitialized variables.
  - to index arrays beyond their bounds
  - to perform unchecked type casts.

Several C# features

- C# has a ***unified type system***

All C# types, including primitive types such as int and double , inherit from a single root object type.

- C# supports both user-defined reference types and value types, allowing dynamic allocation of objects.

Hello World

```
using System;
class Hello
{
        public static void Main()
         {
        Console.WriteLine("Hello, World");
        }
}
```

file extension .cs ***save*** *hello.cs* ***run*** *csc hello.cs*

Hello World

❖ program starts with a *using* directive that references the *System* namespace.

❖ Namespaces provide a hierarchical means of organizing C# programs and libraries.

❖ Namespaces contain types and other namespaces

for example, the System namespace contains a number of types, such as the Console class referenced in the program, and a number of other namespaces, such as IO and Collections

using directive that references a given namespace enables unqualified use of the types that are members of that namespace. Because of the *using* directive, the program can use Console.WriteLine as shorthand for System.Console.WriteLine

.

Hello World

- The *Hello* class declared by the “Hello, World” program has a single member, the method named Main.

- The Main method is declared with the static modifier. Unlike instance methods, which reference a particular object instance using the keyword this , **static** methods operate without reference to a particular object.
- a static method named **Main** serves as the entry point of a program.

Hello World

- The output of the program is produced by the WriteLine method of the Console class in the System namespace.

This class is provided by the .NET Framework class libraries, which, by default, are automatically referenced by the Microsoft C# compiler.

Note that C# itself does not have a separate runtime library. Instead, the .NET Framework is the runtime library of C#.

The Main Method

- Must contain Main method in which control starts and ends
- Main method is a static method
- Resides inside a class
- public :- access modifier
  - It tells the compiler that the Main method is accessible by anyone.

The Main Method

- static :-
  - Declares that the Main method is a global and can be called without an creating instance of a class.

- void :-
  - states that the Main method does not return any value.

The Main Method

- Three ways to declare Main
- i) can be void

```
static void Main()
{
…….
}
```

The Main Method

- Three ways to declare Main
- ii) it can return int

```
static int Main()
{
…….
return 0;
}
```

The Main Method

- Three ways to declare Main
- iii) it can also take arguments

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

  - OR

```
static int Main(string[] args)
{
…….
return 0;
}
```

Program output

- C# programs use the input/outputservices provided by the run-time library of the .NET framework.
- The statement

System.Console.WriteLine("Hello, World"); uses the WriteLine () method in Console class in the runtime library. It displays the string parameter on the standard output stream followed by a new line.

Printing and Formatting Output

```
class sample {
   static void Main()
   {
   System.Console.WriteLine(“ 786  {0}”,
   786};
   }
   Output
   D:\> prg1
        786  786
   D:\>_
```

Printing and Formatting Output

- In the above program {0} means directing the computer to print the first value and {1} means the second value.

- C# treats the 0to be the first digit.

Printing and Formatting Output

```
class sample {
   static void Main()
   {
   System.Console.WriteLine(“ 786  {0} , {1} ”,
   787,788};
   }
   Output
   D:\> prg1
        786  787 788
   D:\>_
```

Printing and Formatting Output

- Another output method

```
using System;
class InputOutput {
        public static void Main()
        {
        Console.Write(“Enter your Name: ”);
        string strName= Console.ReadLine();
        Console.WriteLine(“Hello  {0} “,
strName);
        }
}
```

Printing and Formatting Output

- The statement

```
Console.WriteLine("Hello  {0} ", strName);
```

uses formatted string. The format string is "Hello {0}"

{0} is replaced for the first variable….

Ex:-

```
Console.WriteLine("Hello  {0} {1} ", firstName, lastName);
```

Output

```
D:\> prg2
Enter your Name:  Ram
Hello Ram
```

Printing and Formatting Output

- Input from Console

```
1.  using System;
2.  class InputOutput {
3.      public static void Main()
4.      {
5.      Console.Write(“Enter your Name: ”);
6.      string strName= Console.ReadLine();
7.      Console.WriteLine(“Hello  {0} “, strName);
8.      }
9.  }
```

Line 6 uses a method ReadLine() to accept string.

Compilation and execution

- Use Visual Studio IDE
- Use command Line
  - Create source file using any text editor
  - Save file with extension .cs
  - Enter csc prgName.cs (to invoke compiler)
  - Type prgName (to run)

- Set Path for your directory
  - D:\> path=C:\Windows\Microsoft.NET\Framework\v3.5

Namespace declaration

- Namespaces contain groups of code
- Consider the output statement

System.Console.WriteLine();

System :- namespace (Scope)

With using directive , you can import namespace System into the program.

Command Line Input

- Collection of command line input occurs in the Main method.

```
class Welcome {
    public static void Main(string [] args)
    {
        Console.WriteLine(“Hello, {0} !”,args[0]);
        Console.WriteLine(“Wel-Come to IICC”);
        }
        }
        D:\> csc wel.cs
        D:\> wel  Aditya Bhave
        Hello, Aditya !
        Wel-Come to IICC
```

Interactive input / output

```
using System;
class InputOutput2 {
        public static void Main()
        {
        Console.Write(“Enter your Name: ”);
        Console.Write(“Hello  {0} “,
Console.ReadLine());
        Console.WriteLine(“Wel-Come to IICC”);
        }
}
```

Command Line Outputs

```
using System;
public class hello2{
          public static void Main(string [] args)
          {
          Console.Write(“Hello, IICC!”);
          Console.Write(“You entered following   {0}
command line arguments :“, args.length);
For(int i=0;i<args.length;i++) {
Console.WriteLine(“{0},args[i]);
          }
} }
```

output

```
D:\> hello2  arg1 arg2 arg3
```

Common Language Platform (CLR)

- C# used for developing applications for MS.NET platform
- The .Net platform is designed to be language neutral.
- All .NET code is run under the CLR
- C# is one of the languages used to write classes for CLR

Output Formatting options

- Console.WriteLine("{0,-10} {1,-3}","Name", "Age");
- Console.WriteLine("{0,-10} {1,3}", "Ramm", Age);
- First number in { } 0,1for variables and second number for width
- – means left justified

General Structure of a C# program

- Contains one or more files
- Each file can contain one or more namespace
- A namespace can contain types (classes, structs, intrfaces, enumerations and delegates)

General Structure of a C# program

- **A skeleton of a C# program**

```
using System;
namespace MyNamespace1
{
    Class MyClass1{}
    struct MyStruct
    {}
            Interface IMyInterface{}
            Delegate int MyDelegate();
            Enum MyEnum {}
namespace MyNamesoace2{ }
    class MyClass2 {
            public static void Main(string [] args)
            {
            }
    }
}
```

General Structure of a C# program

- Documentation section
- Using directive section optional
- Interface section
- Classes section
- Main method section essential

Runtime Arguments

- Consider a program for factorial (clineFact.cs)

```
Using System;
public class Factorial {
          public static long Fac(long i)
          {           return ((i<=1)?1: (i*Fac(i-1))); }
}
class MainClass {
          public static int Main(string[] args) {
 if(args.length==0) {
          Console.WriteLine("Please enter a numeric argument");
          return 1; }
Long num=long.Parse(args[0]);
Console.WriteLine("The factorial of {0} is {1}", num, Factorial.Fac(num));
Return 0;
}
}
```

Types

- C# is divided into categories
- Value type (var. directly contain data)
- Reference type (var. stroes references to data)
- Pointers (available but not safe code)

Types

- Value Types
  - struct -type
  - enum –type
- struct-type
  - type-name
  - simple-type
- simple-type
  - numeric-type
  - Bool

Types

- numeric Type
  - integral -type
  - floating point–type
  - decimal
- integral-type
  - sbyte
  - byte
  - short
  - ushort
  - int
  - uint
  - long
  - ulong
  - char

- Signed integer type
  - sbyte (1)
  - Short (2)
  - Int (4)
  - long (8)
- Unsigned
  - byte (1)
  - ushort (2)
  - uint (4)
  - ulong (8)

Types

- By default :- all integers are int
- U- uint (123 U)
- L – long (123 L)
- UL- ulong (123 UL)
- Floating point
  - float
  - double
- enum-type
  - Type-name

Types

- All value types implicitly inherit from the class object.
- No type can derive from a value type.
- A variable of value type always contains a value of that type.

Default Constructor

- All value types implicitly declare a public parameter less constructor called default constructor.
- Returns a zero-initialized instance known as the default value for the value type.
- For all simple types , the default value is the value produced by the bit pattern of all zeros.

Default Constructor

- For all sbyte, byte, short , ushort, int, uint, long and ulong, the default value is 0 .
- For the char, the default value is ‘ \x0000’.
- For float, 0.0f .
- For double , 0.0d.
- For decimal, 0.0m.
- For bool, false.
- For an enum E, 0.
- For a struct type, the value produced by all setting all value types fields to their default value and all ref. type fields to null.

Struct type

- It is a value type
- It can declare
    - Constructor
    - Constants
    - Fields
    - Methods
    - Properties
    - Operators
    - nested types

The decimal type

- 128 bitdata type
- Used for financial calculations
- Range 1.0 x10^{-28} to 7.9 x10^{28} with 28 significant digits.
- Must append M or m to the value for decimal.
- Does not support signed zeros, infinities or NaN.

Enumeration type

- It has named constants
- It is a user defined integer type
- Syntax
  - enum Shape { circle, square, triangle }
  - Circle has a value 0 , square 1 and triangle 2.
  - enum Colour { Red, Blue, Green}

Reference type

- 2-groups
  - User-defined
  - Predefined
- User defined ref types are
  - Class type
  - Interface type
  - Array type
  - Delegate type
- Predefined ref types
  - Object type
  - String type

Class type

- Contains
  - data members (fields, constants, events)
  - Function members (methods, constructors, destructors)
- Support inheritance
- Instances are created using object-creation expression.

Object type

- Every type in C# is directly or indirectly derives from the object class type.
- It is the general base class of all other types.
- An alias for the predefined System.Object class.

String type

- Sealed class type that inherits directly from object.
- Instances represent Unicode character strings.
- Values of the string type can be written as string literals.
- Alias for System.String class

Interface Types

- An interface defines a contract.
- A class or struct that implements an interface must obey to its contract.
- An interface may inherit from multiple base interfaces.
- A class or struct may implement multiple interfaces

Array Types

- An array is a data structure that contains number of variables.
- These are accessed through computed indices.
- The elements of the array are all of same type (element type).

Delegate type

- It is a data structure refers to a static method or an object instance.
- Is a method acting for another method
- Is a class type object.
- Used to invoke a method that has been encapsulated into it at the time of creation.

Delegate type

- Creating and using involve following
  - Delegate Declaration
  - Delegate Methods definition
  - Delegate Instantiation
  - Delegate Invocation

Predefined Types

- The predefined reference types – object and string.
- Type object-- is the ultimate base type of all other types.
- String – used to represent Unicode string values.

Predefined Types

- The predefined value types – signed and unsigned integral types, floating point, bool, char and decimal.
- Signed integral type– sbyte, short, int and long.
- Unsigned integral type– byte, ushort, uint and ulong.
- Floating point– float and double.

Predefined Types

- Bool type- used to represent boolean values. (true or false)
- Char type– used to represent Unicode characters
- Decimal type– is appropriate for calculations in which rounding errors caused by floating point representations are unacceptable.

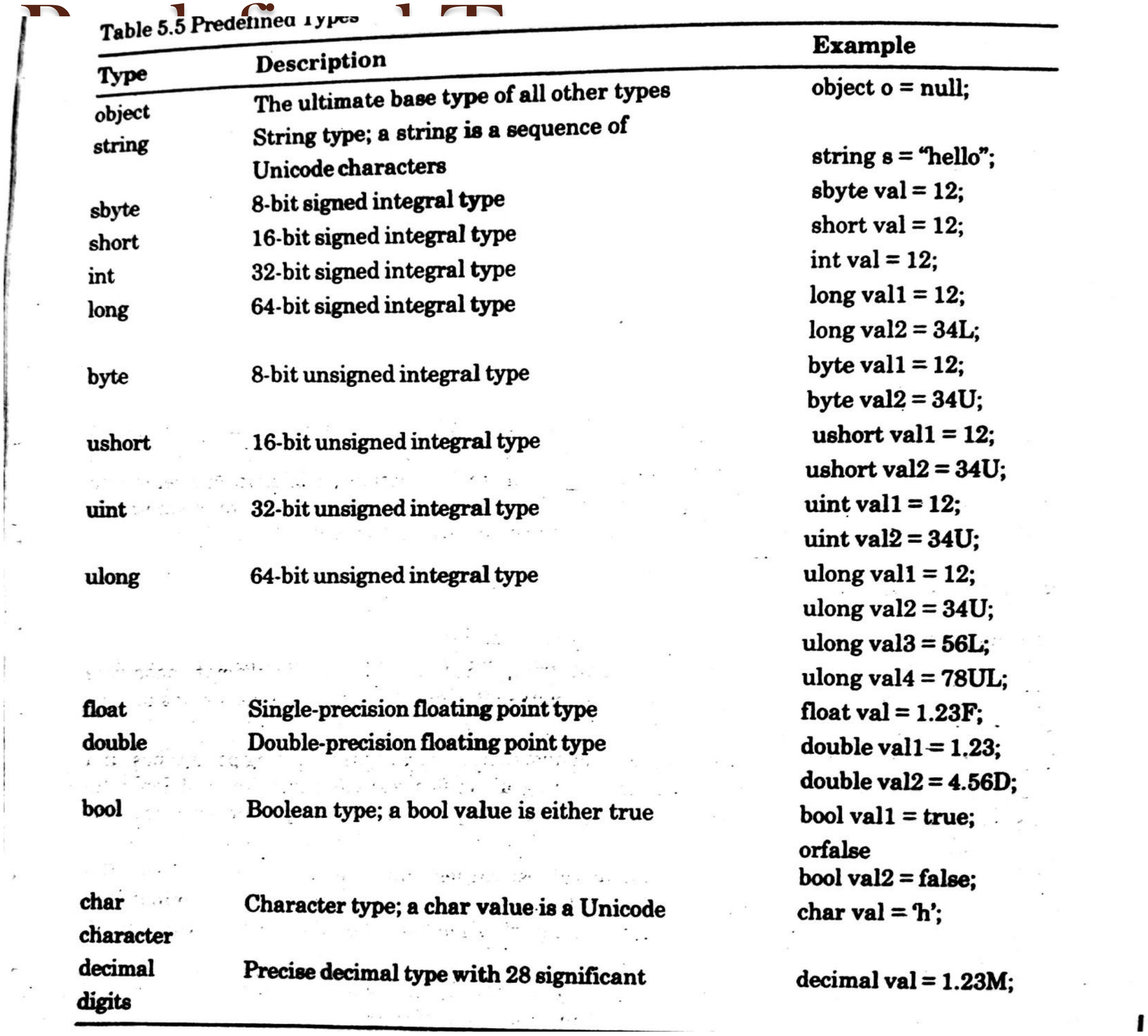

Table 5.5 Predefined Types

Type	Description	Example
object	The ultimate base type of all other types	object o = null;
string	String type; a string is a sequence of Unicode characters	string s = "hello";
sbyte	8-bit signed integral type	sbyte val = 12;
short	16-bit signed integral type	short val = 12;
int	32-bit signed integral type	int val = 12;

long	64-bit signed integral type	long val1 = 12;
		long val2 = 34L;

byte	8-bit unsigned integral type	byte val1 = 12;
		byte val2 = 34U;

ushort	16-bit unsigned integral type	ushort val1 = 12;
		ushort val2 = 34U;

uint	32-bit unsigned integral type	uint val1 = 12;
		uint val2 = 34U;

ulong	64-bit unsigned integral type	ulong val1 = 12;
		ulong val2 = 34U;
		ulong val3 = 56L;
		ulong val4 = 78UL;

float	Single-precision floating point type	float val = 1.23F;

double	Double-precision floating point type	double val1 = 1.23;
		double val2 = 4.56D;

bool	Boolean type; a bool value is either true	bool val1 = true;
		orfalse
		bool val2 = false;

char character	Character type; a char value is a Unicode	char val = 'h';
decimal digits	Precise decimal type with 28 significant	decimal val = 1.23M;

Boxing and unboxing

- Provide a link between value types and ref. types
- Permits any value of value-type to and from type object
- This gives a view of the type system wherein a value of any type can be treated as an object.
- Boxing :- value-type to object-type
- Unboxing :- object-type to value-type

Boxing

- Any type value or reference can be assigned to an object without explicit conversion.
- When a compiler finds value type where it need a reference type, it creates an object 'box' into which it places the value of the value types.
  - int m=100;
  - object ob=m;// creates a box to hold m
  - This code creates a temporary reference type box for the object on heap.

Boxing

- Boxing operation creates a copy of the value of the m integer to the object ob.
- Both m and ob are exist but the value of ob resides on the heap.
- It means are values are independent of each other.
  - int m=10;
  - object ob=m'
  - m=20;
  - Console.WriteLine(m); //m=20
  - Console.WriteLine(ob); //ob=10
  - When a code changes the value of m, the value of ob is not affected.

Unboxing

- Process of converting object type back to value type.
- Only unbox a variable which has been boxed.
- int m=10;
- object ob=m; //boxing m
- int n =(int) ob; //unbox ob to int
- When unboxing, we need to use explicit cast.

Boxing and unboxing

```
class Class1
{ public int Value=0;}
class test{
    Static void Main() {
        int val1=0;int val2=val1;
        val2= 321;
        Class1 ref1=new Class1();
        Class1 ref2=ref1;
        ref2.Value=321;
        Console.WriteLine(“Values:{0},{1}”,val1,val2);
        Console.WriteLine(“Refs:{0},{1}”,ref1.Value,ref2.Value);
        }
    }
```

o/p

```
Values 0,321
Refs 321,321
```

Variables and Parameters

- Variables: represents storage locations where specific variables are stored depending on the variable type.
- Parameter: used in methods
- 4 type of parameters
  - Value, reference, output , and parameter array

Variables and Parameters

- Value parameter refers to its own variable.
- A value parameter is used for in parameter passing, in which the value of an argument passed into a method, and modifications of the parameter do not affect the original argument.

Variables and Parameters

- A reference parameter is used for ’by reference’ parameter passing in which parameter acts as an alias for a caller provided argument.
- A reference parameter does not define a variable but refers to the variable of the corresponding argument.
- A reference parameter is declared by ref keyword.
- Ex. static void Swap(ref int a, ref int b)

Variables and Parameters

- Output parameter is similar to reference parameter, except that the initial value of the caller provided argument is unimportant.
- An output parameter is declared with an out modifier.
- Ex. static void Divide(int a, out int result)

Variables and Parameters

- A parameter array is declared with a params modifier.
- There can be only one parameter array for a given method.
- Alwas a single dimension array
- Ex.static void F(params int[] args)

Operands/Operators

- Three types of Operators
- Unary :-
  - Takes one operand
  - Either prefix or postfix (++x or x++)
- Binary:-
  - Takes two operands
  - Use Infix notations (x+y)
- Ternary:-
  - Only one ternary operator ?: exists
  - Takes three operands & uses infix
  - Eg. z?x:y

Operator precedence

Category	Operators
Primary	(x),f(x),a[x]
new	typeof, sizeof
unary	+,-,++,--, !
multiplicative	*, /,%
Additive	+,-
Shift	<< , >>
Relational	<, > ,<=,>=
Equality	==, !=
Logical	& .,^,| (logical AND/XOR/OR)
Conditional	&& , || , ?:
Assignment	= , =*=,

Operator precedence

- All binary operators are left associative except assignment and conditional(?:).
- Assignment and conditional(?:) are right associative.

Statement

- Statement list and block statement ({ })
- Labeled statement and goto
- Local constant declaration
- Local variable declaration
- Expression statement
  - static int Fun(int x, int y)
  - {return x*y;}
  - static void Main(){
  - Fun(11,22); // exp. St.}
- If/switch/while/for/do/foreach/break/continue/return/throw/using

Expression

- Is a sequence of operators and operands that states a computation.
- Exp. x+y+z evaluated as (x+y)+z
- x=y=z … x=(y=z)

Assign 1

1. Explain the structure of the C#.net program.
2. Explain compilation routine of C#. Net program. Explain command line argument with example.
3. Explain delegates with example. Write its significance.
4. Explain the difference between instance member and class members. Why Main () must be static ?
5. Explain the following operators in detail :
   (i) Relational Operators
   (ii) Logical Operators
   (iii) Ternary Operators
   (iv) Assignment Operators

Assign1

6. Explain the C# main method and the compilation of C# program.
7. Explain the String object and different printing formats in C#.net.
8. Explain array type objects with example.
9. Explain delegates with examples. How events are handled using delegates?
10. Discuss data enumeration in C# with suitable example.
11. Write an application in C# to simulate a die.
12. Explain data types with example. i) Reference type ii) Object type iii) String type v) Interface type
13. Develop C# program for Quick sort method.
14. What are value type and reference data? Explain boxing and unboxing of data with suitable example.
15. Write a program to change the case of all characters in an array. (Do not use in built methods)
16. Explain interfaces and delegates with example.
17. Explain precedance of operator in C#.NET for logical and mathematical operators. Write a program to execute the following series.
    - y = 1*2–2*3+3*4–4*5———

C#Environment

- Tools required for creating C# programming.
- C# is part of .Net framework and is used for writing .Net applications.
- let us understand how C# relates to the .Net framework.

- **The .Net Framework**
- The .Net framework is a revolutionary platform that helps you to write the following types of applications:
  - □ Windows applications
  - □ Web applications
  - □ Web services

C# Environment

- The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: C#, C++, Visual Basic, Jscript, COBOL, etc. All these languages can access the framework as well as communicate with each other.
- The .Net framework consists of an enormous library of codes used by the client languages like C#. Following are some of the components of the .Net framework:
- Common Language Runtime (CLR)
- The .Net Framework Class Library
- Common Language Specification
- Common Type System
- Metadata and Assemblies
- Windows Forms
- ASP.Net and ASP.Net AJAX
- ADO.Net

by Arvind M. Bhave