Programming Manual for C#

Trainer: Mr. C.Santhosh Kumar, 329469

INDEX

1.Basic Concepts Of C#

Structure of C# program:

- Classes are main blocks of a C# program.
- A program in C# may contain many classes of which only one class defines a main method.
- Classes contain data members and methods that operate on the data members of the class.
- Methods may contain data type declaration and executable statements.
- To write a C# program, we first define a class and then put them together.
- A C# program may contain one or more sections as given below
    - Documentation Section
        - It is a best Suggested practice to include them
    - Import Library Files(using)
        - It is Optional
    - Namespace,Class and Type definitions
        - Essential (atleast one class containing main method)

Basic syntax of a C# Program is given below:

```
using namespaces;

namespace projectName
{
        class class_name1
        {
                .....
        }
        class class_name2
        {
                .....
        }

        ....
        class class_nameN
        {
                .....
        }
}
```

Datatypes:

There are three main categories in datatypes.

- Value types
- Reference types
- Implicitly typed local variable (var)

Let us see what are all the datatypes available in each types.

a)Value types:

- **Predefined types:**
  - **(a) Numeric types:**
    - ***(i) Integer***
      - **1) Signed int**
        - a) Sbyte (8 bit signed int)
        - b) Short(16 bit signed int)
        - c) Int (32 bit signed int)
        - d) Long (64 bit signed int)
      - **2) Unsigned int**
        - a) Byte (8 bit unsigned int)
        - b) Ushort (16 bit unsigned int)
        - c) Uint (32 bit unsigned int)
        - d) Ulong (64 bit unsigned int)
    - ***(ii) Float***
    - ***(iii) Double***
  - **(b) Char**
  - **(c) Decimal**
  - **(d) Boolean**
    - ♦ True
    - ♦ False

- **User defined types:**
  - 1) Struct
  - 2) Enum

b) Reference type:

- **Predefined types:**
  - 1) String
  - 2) Object

- **User defined types:**
  1) Array
  2) Classes
  3) Interfaces
  4) List
  5) Array list
  6) Stack
  7) Hash table
  8) Queue

  And so on..

Note: Value types are stored in stack memory and reference types are stored in Heap memory in C#.

Type Conversion:

There are 2 types of conversion available in C#.

a)Using Conversion functions

a. Convert.ToInt32() - converts values to integer
b. Convert.ToDouble()- converts values to double
c. Convert.ToSingle()- converts values to float
d. Convert.ToDecimal()- converts values to decimal

b)Using parser methods

a. int.parse()
b. float.parse()
c. double.parse()

and so on…

Implicit and explicit type conversion:

a) Implicit Type Conversion:

Conversion from value type variables to reference type variable is called **Boxing** (Implicit type conversion).

Example: Boxing

```
int i=10;
object Obj;
Obj=i; //Implicit type conversion occurs from integer type to object type
```

b) Explicit Type Conversion:

Conversion from reference type variables to value type variable is called **Unboxing**(Explicit type conversion).

Example: Unboxing

```
object Obj = 5;
float j;
j=(float) Obj; //Explicit type conversion from Object type to float type is needed.
```

Operators:

It is used to perform operations/Calculations on the operands. There are several types of operators. They are namely:

a)Arithmetic Operator: +,-,*,/,%
b)Assignment Operator: =
c) Relational Operator: <,>,<=,>=,!=,==
d)Logical operator: && ,||,!
e)Unary Operator: ++,--
f)ShortHand Operators: *=,+=,/=,%=

example for using shorthand operator:

a*=1 is equivalent to use a=a*1.

2.Control flow constructs and Looping

C# has several control flow constructs like switch, Continue, Break. Let us see more in detail about those control flow constructs in this section.

1.Switch case:

Note:

- In C#, The switch expression must be of an integer type, such as char, byte, short, or int, or of type string .
- The default statement sequence is executed if no case constant matches the expression.
- The default is optional.
- If default is not present, no action takes place if all matches fail.
- When a match is found, the statements associated with that case are executed until the break is encountered.

Syntax for switch case in C#:

```
switch(expression)
{
  case constant1:
     statement sequence
     break;
  case constant2:
     statement sequence
     break;
  case constant3:
     statement sequence
     break;
  .
  .
  .
  default:
    statement sequence
    break;
}
```

Sample program using switch:

```
using System;

class MainClass
{
  public static void Main()
{
    int i;

    for(i=0; i<10; i++)
      switch(i)
     {
        case 0:
          Console.WriteLine("i is zero");
          break;
        case 1:
          Console.WriteLine("i is one");
          break;
        case 2:
          Console.WriteLine("i is two");
          break;
        case 3:
          Console.WriteLine("i is three");
          break;
        case 4:
          Console.WriteLine("i is four");
          break;
        default:
          Console.WriteLine("i is five or more");
          break;
      }
  }
}
```

Output:

```
i is zero
i is one
i is two
i is three
i is four
i is five or more
i is five or more
i is five or more
i is five or more
i is five or more
```

2.Break:

- The break keyword alters control flow.
- Break statements are used in certain parts of a program, including for-loops, foreach-loops, while loops, do- while loops and switch-statements.
- Break will exit from the loop without waiting until given condition for the loop is false.

Sample program using break:

```
using System;

namespace ProgramCall
{
   class BreakPrime
   {
      static void Main()
      {
         int N, i;

         Console.Write("Enter An Integer : ");
         N = int.Parse(Console.ReadLine());

         for (i = 2; i < N; i++)
         {

            if (N % i == 0)
               break;

         }

         if (i == N)
            Console.WriteLine("Prime  Number");
         else
            Console.WriteLine("Not  A  Prime  Number");

         Console.Read();
      }
   }
}
```

Output:

```
Enter An Integer : 6543
Not  A  Prime  Number
```

3.Continue:

- Continue alters control flow.
- The continue keyword allows you to skip the execution of the rest of the iteration.
- It jumps immediately to the next iteration in the loop.
- This keyword is most useful in while loops.

Sample program using Continue:

```
using System;
class ContinueTest
{
  public static void Main()
  {
    for (int i = 1; i <= 10; i++)
    {
      if (i < 9)
        continue;
      Console.WriteLine(i);
    }
  }
}
```

Output:

```
9
10
```

Looping in C#:

- Like java, C# also has several looping statements like while, do while, for, for each and normal if, if else statements.

Example for all the looping construct:

```
using System;
class LoopsExample
{
  public static void Main()
```

```
  {
    for (int i = 1; i <= 10; i++)
    {
      if (i%2==0)
        Console.WriteLine(i+" is even");
      else
        Console.WriteLine(i+" is odd");
    }
    int j=0;
    while(j<5)
    {
       Console.WriteLine("Using While Loop");
       j++;
    }
    do
    {
        Console.WriteLine("Using Do While loop");
        j++;
    }while(j<7);

  }
}
```

output:

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
10 is even
Using While Loop
Using While Loop
Using While Loop
Using While Loop
Using While Loop
Using Do While loop
```

3.Parameter Types In C#

There are four types of parameters in C#. Namely:

- **Value parameter(Call by value):**
  - Passes the copy of the actual parameter from the main method to sub methods.
  - When we change the value in the sub method, the changes made will remain only in the copy and it will not reflect in the actual value.

- **Out parameter(Call by Reference):**
  - Passes the address of the actual parameter from the main method to sub methods.
  - When we change the value in the sub method, the actual value in the main method will also be changed . That is, Changes made inside the sub method will reflect in main method also.

- **Ref parameter**
  - Same as out parameter. The difference and similarity is explained in the upcoming section

- **Params Parameter**
  - Used with large set of values (when using arrays). We can avoid method overloading with the params. We can avoid the usage of various signatures to same function name with the help of params. It accepts any number of values provided to it.

Example for value parameter:

```
static void Change(int x, int y)
 {
      x = 30;
      y = 20;
    Console.WriteLine(x + " " + y);    // Note: changes made with x and y will not affect the actual values of a and b
 }

 static void Main(string[] args)
 {
    int a = 20;
    int b = 30;

    Console.WriteLine(a + " " + b);
    Change(a, b);
```

```
        Console.WriteLine(a + " " + b);
        Console.Read();
    }
```

Output:
20 30
30 20
20 30

Example for out parameter:

***Note**: Use **out** keyword while passing the values.*
It is not mandatory to initialize the value of a and b before passing it to the submethod but we should assign some value for the out variables x and y in the submethod.

```
    static void Change(out int x, out int y)
    {
        x = 30;
        y=20;
     Console.WriteLine(x + " " + y);    // Note: changes made with x and y will be affected in the actual values of a and b
    }

    static void Main(string[] args)
    {
        int a = 20;
        int b = 30;

        Console.WriteLine(a + " " + b);
        Change(out a, out b);
        Console.WriteLine(a + " " + b);
        Console.Read();
    }
```

Output:
20 30
30 20
30 20

Example for ref parameter:

```
static void Change(ref int x, ref int y)
  {
     x = 30;
     y=20;
   Console.WriteLine(x + " " + y);    // Note: changes made with x and y will be affected in the actual values of a and b
  }


static void Main(string[] args)
  {
     int a = 20;
     int b = 30;

     Console.WriteLine(a + " " + b);
     Change(ref a, ref b);
     Console.WriteLine(a + " " + b);
     Console.Read();
  }
```

Output:
20 30
30 20
30 20

***Note**: Use **ref** keyword while passing the values.*
we should assign some value for x and y in the submethod.

Difference between out parameter and ref parameter is :

- For ref parameter, The value should be initialized before passing. For out parameter, it is not mandatory.

Similarity between out and ref parameter:

- The values cannot be manipulated but they can only be used in the left side of the expression but not on the right side.

Example:
A = 20 VALID(where A is a out/ref parameter. A is used in the left side of the expression)
A= A+20 INVALID(where A is a out/ref parameter. A is used also in the right side of the expression)

Example for Params parameter:

```
static void passdata(params int[] arrObj)
{
        foreach (int x in arrObj)
          {
              Console.Write(x + " ");
          }
  }


static void Main(string[] args)
  {
           int[] arr=new int[5]{10,20,30,40,50};
           int[] arr1 = new int[3] { 60, 70, 80 };
          passdata(arr); //note: The same function can accepts the two different arrays with different
size.
          passdata(arr1);
  }
```

4.Arrays in C#

Syntax:

a) **Single Dimensional array**
Datatype[] arrayObj1= new Datatype[Size];

b) **Rectangular array**
Datatype[1] arrayObj2 = new Datatype[size1,size2];

c) **Jagged Array(Array of Array)**
Datatype[][] arrayObj3=new Datatype[Size][];

Note: array index starts from [0] and extends till [size – 1]. If we try to access any index that exceeds size, itwill show array out of bound error.

a)Single dimensional array:

Array initialization:

Single step declaration and initialization:

Ex: int[] arr=new int[5]{10,20,30,40,50}

Note: we have to provide all the 5 values explicitly ***during this type of declaration and initialization.***

Individual initialization:

```
Ex:   int [] arr = new int[5];
      arr[0]=10;
      arr[1]=20;
      arr[2]=30;
      arr[3]=40;
      arr[4]=50;
```

Loop initialization:

```
int[] arr=new int[5];
for(int i=0;i<arr.length;i++)
{
        arr[i]=Covert.ToInt32(Console.ReadLine());
        //Note:we can also use int.parse() method to retrieve integer values.
}
```

For-Each loop declaration:

Note:

- It automatically gets the count of the no. of elements and assigns it to the local variable.
- It automatically increments at iterations (implicit incrementation).
- If the type of the array is String, then the local variable declared inside the foreach loop should be of type String.

Syntax:

```
foreach(typeOfTheArray  variablename in array_name)
{
//body
}
```

Example:

```
int[] arrObj = new int[5]{10,20,30,40,50};
foreach(int VALUE in arrObj)
{
        Console.Write(VALUE+"  " )
}
```

Output:

10 20 30 40 50

Example:

Below is the two types of loops which does the same action. It displays all the strings which has length greater than 5.

Using For-each loop:

```
public void printAllStringsWithLengthFiveWithForEachLoop(String[] array)
{
   foreach (String name in array)
   {
      if (name.Length > 5)
         Console.WriteLine(name);
   }
}
```

Using For loop:

```
public void printAllStringsWithLengthFiveWithForLoop(String[] array)
{
   for (int i = 0; i < array.Length; i++)
   {
      if (array.Length > 5)
         Console.WriteLine(array);
   }
}
```

Method call:

```
String[] array = new String[5] { "lena", "chanakya", "prasanna", "kishore", "harsha" };
 obj.printAllStringsWithLengthFiveWithForLoop(array);
 obj.printAllStringsWithLengthFiveWithForEachLoop(array);
```

Output for both method calls:

chanakya
prasanna
kishore

Exercise: Write a function to print all the characters greater than “m”.

```
public void printMoreThanM()
 {
        char[] arrchar = new char[4];

        Console.WriteLine("enter the character values");

         for (int i = 0; i < arrchar.Length; i++)
        {
                arrchar[i] = char.Parse(Console.ReadLine()); //getting individual characters
        }
      for (int i = 0; i < arrchar.Length; i++)
      {
         if ((arrchar[i]) > 'm')    //check for greater than m
            Console.WriteLine(arrchar[i]);

      }
```

}

b)Rectangular array:

Note:

- Rectangular array is a multi dimensional array.
- Syntax for declaration of a 2 dimensional array is:
    - Int [,] arrObj=new int[2,3]
- In Jagged array, we can have different no. of columns for each row where as in Rectangular array, we have same no. of columns for each row. For the above declaration, each row of arrObj will have 3 columns.

Array initialization:

Single step declaration and initialization:

Ex: int[,] arrRectObj=new int[2,3]{ {10,20,30},{400,500,600}};

Individual initialization of two dimensional array :

arrRectObj[0,0] = 10; ***// Value for row 0 and column 0 is 10***
arrRectObj[0,1] = 20;
arrRectObj[0,2] = 30; ***// Value for row 0 and column 2 is 30***
arrRectObj[1,0] = 400;
arrRectObj[1,1] = 500;
arrRectObj[1,2] = 600; ***// Value for row 1 and column 2 is 600***

Loop initialization:

Using foreach loop:

int[,] arrRectObj = new int[2, 3]; //***declaring a 2 dimensional array***

foreach(int val in arrRectObj) ***// Note: since the type of arrRectObj is of type int, the variable val should also be declared as an integer inorder to fetch the data from the array***

```
{
        Console.Write(val)
}
```

Using for loop:

```
int[ , ] arrRectObj = new int[2, 3]; //declaring a 2 dimensional array
for (int rows = 0; rows < 2; rows++)
{
        for (int columns = 0; columns < 3; columns++)
         {
                arrRectObj[rows, columns] = int.Parse(Console.ReadLine());
              // Note: getting individual values
         }
}
```

C)Jagged array:

Note:

- Here,arrJagged[0], arrJagged[1], arrJagged[2] are individual arrays. Grouping all the arrays into a single group becomes a jagged array.
- Any number of values can be stored into those arrays.

Array initialization:

Single step declaration and initialization:

Note: call explicit new keyword to allocate memory for individual elements in the array.

```
int[][] arrJagged = new int[3][]; // jagged array declaration


  arrJagged[0]=new int[3]{10,20,30};//Explicit array initialization using new keyword is required.
  arrJagged[1]=new int[2]{40,50};
  arrJagged[2]=new int[4]{60,70,80,90};
```

Using foreach loop:

```
   foreach(int[] outerval in arrJagged)
    {
```

```
        foreach (int innerval in outerval)
        {
            Console.WriteLine(innerval);
        }
    }
```

Output:

10
20
30
40
50
60
70
80
90

Note:*Incase the elements are of char type, no need to use nested foreach loop. It is sufficient to use a single foreach loop to get the entire elements from an array.*

```
Char[][] arrJagged=new char[3][];
arrJagged[0]=new char[2]{'B','V'};
arrJagged[1]=new char[3]{'T','N','P'};
arrJagged[2]= new char[1]{'L'}; // we should do in this manner to initialize the values explicitly

foreach (char[] value in arrJagged)
 {
        Console.Write(value);
 }
```

We can also use for loop for char type jagged array instead of foreach loop. Below is the syntax to get the values in the char jagged array.
Note: *This will* ***work only for char type jagged array*** *and not for int or any other type.*

```
for(int i =0; i < 3;i++)
{
        Console.Write(arrJagged[i]);
}
```

5.Constructors and Destructors

Constructor:

- Constructor is the special member function which is used for object initialization.
- It will be automatically invoked when objects are instantiated.
- Constructor name will be similar to the class name.
- Constructor will not return any value.
- A class can have multiple constructors.
- Constructors can be overloaded.
- Constructors can be inherited.
- If constructor is overloaded, we need to provide default constructor also.

Constructor Types:

- Default Constructor(or) Parameterless constructor
- Parameterized Constructor
- Copy Constructor
- Static Constructor

Syntax for Constructor:

```
classname()  //Example default constructor
{
//body
}

Classname(parameters) //Example parameterized constructor
{
//body
}
```

Destructor:

- Destructor cannot be overloaded.
- Destructor is used to release or free up the resources.
- Only one Destructor is allowed for a class.
- Destructor name is same as the class name, but a tiled operator is prefixed to the class name.
- Destructors cannot be inherited.

Syntax for destructor:

```
~classname()  //Example destructor
{
}
```

Example program for Constructor and destructor:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConstructorDestructor
{
   class Person
   {
      public int pid;
      private string pName;
      private int pAge;

      public Person() //Default constructor
      {
         Console.WriteLine("Entering Default Constructor!");
         pid = 0;
         pName = string.Empty;
         pAge = 18;
      }

      public Person(int id) //Parameterised constructor with one argument.
      {
         Console.WriteLine("Entering parameterised Constructor with one argument!");
         pid = id;
         pName = "Kishore";
         pAge = 21;
      }
      public Person(int id, string name, int age)//Parameterised constructor with three arguments.
      {
         Console.WriteLine("Entering parameterised Constructor with three arguments!");
         pid = id;
         pName = name;
         pAge = age;
      }
      public Person(Person m) //copy constructor
      {
         pid = m.pid;
```

```
        pName = m.pName;
        pAge = m.pAge;
    }

    public void showDetails() // Method
    {
        Console.WriteLine("Person Id=" + pid);
        Console.WriteLine("Name=" + pName);
        Console.WriteLine("Age=" + pAge+"\n");
    }

    ~Person()  //Destructor
    {
        Console.WriteLine("Destructor is called");
    }
    static void Main(string[] args)
    {
        Console.WriteLine("Default Constructor");
        Person Obj1 = new Person(); //Default constructor will be called automatically
        Obj1.showDetails();

        Console.WriteLine("Parameterized Constructor");
        Person Obj2 = new Person(100); //parameterised constructor will be called automatically
        Obj2.showDetails();

        Console.WriteLine("Parameterized Constructor");
        Person Obj3 = new Person(100, "Lenaprasanna", 22);
        Obj3.showDetails();

        Console.WriteLine("Copy Constructor  Obj4 <--- obj3 \nNote:Reference is not shared with copy
constructor");
        Person Obj4= new Person(Obj3); //copy constructor is called.
        Obj4.showDetails();

        Obj4.pid = 500;

        Console.WriteLine("Obj4 data");
        Obj4.showDetails(); //Displays the changed value of pid = 500.

        Console.WriteLine("Obj3 data");
        Obj3.showDetails(); //actual Value of pid will not be changed in Obj3. Reason: Only the copy of
Obj3 is passed to Obj 4 and not the reference of Obj 3. Hence changes made with Obj4 remains local to
Obj4 alone.So, it does not reflect in obj3.

        Console.WriteLine("Passing the reference itself. Obj5 <--Obj2");
        Person Obj5 = Obj2; // Copy constructor is not called.Here, Only the reference of Obj2 is shared
with Obj5
```

```
        Obj5.showDetails();

        Obj5.pid = 22;

        Console.WriteLine("Obj5 data");
        Obj5.showDetails();

        Console.WriteLine("Obj2 data");
        Obj2.showDetails();

        Console.Read();
    }
  }
}
```

Output:

```
Default Constructor
Entering Default Constructor!
Person Id=0
Name=
Age=18

Parameterized Constructor
Entering parameterised Constructor with one argument!
Person Id=100
Name=Kishore
Age=21

Parameterized Constructor
Entering parameterised Constructor with three arguments!
Person Id=100
Name=Lenaprasanna
Age=22

Copy Constructor  Obj4 <--- obj3
Note:Reference is not shared with copy constructor
Person Id=100
Name=Lenaprasanna
Age=22

Obj4 data
Person Id=500
Name=Lenaprasanna
Age=22

Obj3 data
Person Id=100
Name=Lenaprasanna
Age=22

Passing the reference itself. Obj5 <--Obj2
Person Id=100
```

```
Name=Kishore
Age=21

Obj5 data
Person Id=22
Name=Kishore
Age=21

Obj2 data
Person Id=22
Name=Kishore
Age=21

Press any key to continue . . .
```

Note:

The above program is an example of **constructor overloading**. In the above example, Obj1 calls default constructor, obj2 calls parameterized constructor with one argument, Obj3 calls parameterized constructor with three arguments.

Overloading is briefly explained under the Polymorphism topic.

6.Structure members and classes

- Structures are user defined value type.
- They are similar to classes.
- They can be instantiated as a reference type.
- Structures can have constructors, methods.
- Constructors in structure can be overloaded.
- Structure can inherit interface.
- Structure cannot inherit class. But classes can inherit both the structures and interface.

Note: Structure does not allow default or parameterless constructor.

Important points in nutshell:

Struct	Class
1)Struct is a Value type which stores in the stack memory.	1)Class is the reference type which stores in the heap memory.
2)Can inherit interface	2)Can inherit interface
3)Cannot inherit class	3)Can inherit class
4)Cannot support default constructor	4)Default constructor is allowed.
5) No destructor is allowed	5)can have destructor

Structures can be created in 2 ways in C#:

- structName instanceName;
- structName instanceName=new structName();

Syntax for creating a structure:

```
AccessSpecifier struct structName
{
        AccessSpecifier DataMember
Declaration;
As Constructor (parameters)
{
//statements;
}
As returnType methodname()
{
//Method body;
}
```

Sample program that illustrates structure:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace StructureDemo
{
   class PersonStructure
   {
      struct Person
      {
         public int pid;
         public string pName;
         public int pAge;


         public Person(int id, string name, int age) //Parameterized constructor with three arguments.
         {
            Console.WriteLine("Entering parameterized Constructor with three arguments!");
            pid = id;
            pName = name;
            pAge = age;
         }
         public void showDetails() // Method
         {
            Console.WriteLine("Person Id=" + pid);
            Console.WriteLine("Name=" + pName);
            Console.WriteLine("Age=" + pAge + "\n");
         }
    }
      static void Main(string[] args)
      {
         Person Obj1 = new Person(); //This will work. In this case, Default constructor will be called
automatically. But a structure definition should not contain the default constructor.
         Obj1.pid = 100;
         Obj1.pName = "Lena";
         Obj1.pAge = 22;
         Obj1.showDetails();
```

```
        Person Obj2 = new Person(100, "Chanakya", 22);
        Obj2.showDetails();

        Person Obj3; //Structures can be instantiated without new operator. But classes cannot.
        Obj3.pid = 130;
        Obj3.pName = "Lenaprasanna";
        Obj3.pAge = 22;
        Obj3.showDetails();
        Person Obj4 = Obj3; //object assignment. A copy of Obj3 is made and it is assigned to Obj4.
        Obj4.showDetails();
        //Reassign the value is replaced and check the details
        Obj4.pAge = 25;
        Obj3.showDetails(); // Since obj4 and obj3 are structure types, changes made with Obj4 will not
affect the Obj3's data. But in the case of class, changes made with obj4 will affect obj3's data also.
This is because the memory has been shared in class where as a copy of the data is shared in the case
of structure.
        //Hence we conclude, structure is a value type and class is a reference type.
        Obj4.showDetails();
        Console.Read();
      }
   }
}
```

Output:

```
Person Id=100
Name=Lena
Age=22

Entering parameterized Constructor with three arguments!
Person Id=100
Name=Chanakya
Age=22

Person Id=130
Name=Lenaprasanna
Age=22

Person Id=130
Name=Lenaprasanna
Age=22

Person Id=130
Name=Lenaprasanna
Age=22

Person Id=130
Name=Lenaprasanna
Age=25
```

7.Enumerations

- Enumerations are named integer constants.
- Enumerations consists of large amount of data with unique integer values for each data.
- Default value for enum will start from 0.

Example:

```
Enum enumName
(
        Apple,
        Grapes,
        Banana,
        Orange
)
```

- By default, apple has the default value 0, Grapes will have 1, Banana will have 2, Orange will have 3.

- We can also assign an integer constant for an enum constant.

Example:

```
Enum enumName
(
        Apple,
        Grapes = 11,
        Banana,
        Orange
)
```

Apple will have default value 0 , whereas for Grapes, Banana and Orange, it will differ. Here, Grapes will have 11, Banana will have 12, Orange will have 13.

Sample Program using enumeration:

```
using System;

namespace ConsoleApplication1
{
   public enum Days { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }
```

```
    class Program
    {
        static void Main(string[] args)
        {
            Days day = Days.Monday;
            Console.WriteLine((int)day);
            Console.ReadLine();
        }
    }
}
```

Output:

0

8.OOPs Concepts

- **Classes**
  - A class is a blueprint for all things of that type
  - Instance of a class is a thing, an *object*
  - Classes have three main types of members
    - Methods (functions in other languages)
    - Fields (the data, sometimes called member variables)
    - Properties (accessed like fields)
- **Objects**
  - An object is a self-contained piece of functionality that can be easily used, and re-used as the building blocks for a software application.
  - Objects consist of data variables and functions (called *methods*) that can be accessed and called on the object to perform tasks. These are collectively referred to as *members*.

Declaring a C# Class:

A C# class is declared using the *public class* keywords followed by the name of the class. Although the C# compiler will accept just about any name for a class, programming convention dictates that a class name begin with a capital letter:

```
public class BankAccount
{
// class body
}
```

Creating class members:

- Class members or properties are essentially variables and methods embedded into the class. Members can be *public*, *private* or *protected*.
- *public* members can be accessed from outside the object and are also visible in classes derived from the current class.
- *private* members can only be accessed by methods contained in the class and are not accessible to derived classes.
- *protected* classes are only available to derived classes.

- This is the key to what is called ***data encapsulation***. Object-oriented programming convention dictates that data should be encapsulated in the class and accessed and set only through the methods of the class (typically called *getters* and *setters*).

- **Abstraction**
  - It is the process of providing only essential information to the outside word and hiding their background details ie. to represent the needed information in program without presenting the details.

- **Inheritance**
  - Inheritance is where one class (child class) inherits the members of another class (parent class).
  - The benefit of inheritance is that the child class doesn't have to redeclare and redefine all the members which it inherits from the parent class.
  - It is therefore a way to re-use code.

- **Interface**
  - An interface looks like a class, but has no implementation.
  - Interfaces in C# are provided as a replacement of **multiple inheritance**. Because C# does not support multiple inheritance, it was necessary to incorporate some other method so that the class can inherit the behavior of more than one class, avoiding the problem of name ambiguity that is found in C++.

- **Polymorphism**
  - Polymorphism is the ability for classes to provide different implementations of methods that are called by the same name.
  - Polymorphism allows a method of a class to be called without regard to what specific implementation it provides.

Access Specifiers in C#:

- **Private**
  - Can be accessed only within the class.
  - The default access specifier for classes and object data members is **private.**
- **Protected**
  - Can be accessed within the class and derived class.
- **Public**
  - Can be accessed anywhere
- **Internal**
  - Specifies that data members is accessed within the assembly.
  - Default access specifier for objects is **Internal.**
- **Protected Internal**
  - Inherits the properties of classes during Assembly.

Other Specifiers:

- Static
- Const
- Sealed
- Read only
- Virtual

Note:

- **Static** methods can access only the static fields. It can also access any fields that are locally declared within the static method. It does not need an object to be called. It is called using its name explicitly without the object.
- No body can inherit the class /method if the class or method is declared as **sealed**.
- If we are overriding a method in the derived class, we use "**virtual**" keyword in the base class and we use "override" keyword in the derived class.

Example Scenarios:

Case 1) Using a local non static variable inside the static method - VALID

```
Static void fun()
{
    int a;
    a++; //valid
}
```

Case 2) Using a non static variable inside the static method - INVALID

```
    int a;    //Assume a is declared inside the class.
  Static void fun()
  {
    a++; //invalid
  }
```

Case 3) Using a static variable inside the static method -VALID

```
   Static int a;    //Assume a is declared inside the class.
  Static void fun()
  {
    a++; //valid
  }
```

Case 4) Using a static variable inside the non static method - VALID

```
Static int a;
Void fun()
{
    a++; //valid
}
```

Syntax to access the methods of the class that belong to another project:

Namespace1.classname1.method()

where Namespace1 is the namespace within which the classname1 resides and within which the actual method to be called is written. The method() we are calling should be **Public.**

9.Polymorphism

Polymorphism is one of the fundamental concepts of OOP.

Polymorphism provides following features:

- It allows you to invoke methods of derived class through base class reference during runtime.
- It has the ability for classes to provide different implementations of methods that are called through the same name.
-

Polymorphism is of two types:

1. Compile time polymorphism/Overloading
2. Runtime polymorphism/Overriding

Type 1: Compile time Polymorphism/ Overloading:

compile time polymorphism can call the functions during compile time.

Note: In general, Operator Overloading, Method Overloading and Constructor overloading is an example of compile time polymorphism.

- **Constructor Overloading** :
  When a class contains several constructors but of different number of parameters, is called Constructor Overloading. The example of Constructor overloading is available under the Constructor and destructor topic.

- **Method overloading:**
  When a class contains methods of the same name but of different number of parameters, is called Method Overloading.
- **Operator Overloading** :
  Process of adding new features to an existing operators.

  Need For Operator Overloading:
  Although operator overloading gives us syntactic convenience, it also help us greatly to generate more readable and intuitive code in a number of situations.

Syntax for Operator Overloading:

```
Public static classname operator op (arglist)
{
Method body // task defined
}
```

Example for Method overloading:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PolyMorphism
{
   class MethodPolymorphism
   {
      public int Add(int a, int b)
      {
         Console.WriteLine("Entering add() with 1 arg!!");
         return a + b;
      }

      public int Add(int a, int b, int c)
      {
         Console.WriteLine("Entering add() with 3 args!!");
         return a + b + c;
      }


      public int Add(int a, int b, int c, int d)
      {
         Console.WriteLine("Entering add() with 4 args!!");
         return a + b + c + d;
      }


      static void Main(string[] args)
      {
         MethodPolymorphism obj = new MethodPolymorphism();
         int sum = obj.Add(4, 5);
         Console.WriteLine(sum);
         sum = obj.Add(4, 5, 7);
         Console.WriteLine(sum);
         sum = obj.Add(4, 5, 7, 9);
```

```
        Console.WriteLine(sum);

    }
  }
}
```

Output:

```
Entering add() with 1 arg!!
9
Entering add() with 3 args!!
16
Entering add() with 4 args!!
25
Press any key to continue . . .
```

Example for Operator Overloading:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OperatorOverloading
{

      class ThreeDShape
      {
      private int height;

      private int breadth;

     public ThreeDShape()
      {
         this.height = 5;
         this.breadth = 10;
      }
      public ThreeDShape(int height,int breadth)
      {
         this.height = height;
         this.breadth = breadth;
      }
      public static ThreeDShape operator -(ThreeDShape t1)
      {
         ThreeDShape result = new ThreeDShape();
         result.height = -t1.height;
         result.breadth = -t1.breadth;
         return result;
```

```
    }
    public static ThreeDShape operator +(ThreeDShape t1, ThreeDShape t2) //Binary operator
overloading with 2 objects
    {
      ThreeDShape result = new ThreeDShape();
      result.height = t1.height + t2.height;
      result.breadth = t1.breadth + t2.breadth;
      return result;
    }

    public static ThreeDShape operator +(ThreeDShape t1, int value) //Binary operator overloading
with an object and integer
    {
      ThreeDShape result = new ThreeDShape();
      result.height = t1.height + value;
      result.breadth = t1.breadth +value;
      return result;
    }

    public static ThreeDShape operator --(ThreeDShape t1) // Unary decrement Operator overloading
for both pre decrement and post decrement.
    {
      ThreeDShape result = new ThreeDShape();
      result.height = t1.height - 1;
      result.breadth = t1.breadth - 1;
      return result;
    }

    public void show()
    {
      Console.WriteLine("\nHeight=" + height);
      Console.WriteLine("breadth=" + breadth+"\n");
    }
    static void Main(string[] args)
    {
      ThreeDShape obj1 = new ThreeDShape(10,20);
      Console.WriteLine("***Obj1***");
      obj1.show();

      ThreeDShape obj2 = new ThreeDShape(30,40);
      Console.WriteLine("***Obj2***");
      obj2.show();

      ThreeDShape obj3 = new ThreeDShape();
      Console.WriteLine("***Obj3=Obj1+Obj2***");
      obj3 = obj1 + obj2; //binary Operator overloading for + operator is done here
      obj3.show();
```

```
        ThreeDShape obj4 = new ThreeDShape();
        Console.WriteLine("***Obj4=-Obj3***");
        obj4 = -obj3; //unary Operator overloading for - operator is done here
        obj4.show();

        ThreeDShape obj5=new ThreeDShape();
        Console.WriteLine("***Obj5=Obj3+10***");
        obj5= obj3 +10;
        obj5.show();

        ThreeDShape obj6 = new ThreeDShape();
        Console.WriteLine("***Obj4=--Obj3***");
        obj6 = --obj3; //Pre decrement Operator overloading is done here
        obj6.show();

        ThreeDShape obj7 = new ThreeDShape();
        Console.WriteLine("***Obj4=--Obj3***");
        obj7= obj3--; //Post decrement Operator overloading is done here
        obj7.show();

    }
  }
}
```

Output:

```
***Obj1***

Height=10
breadth=20

***Obj2***

Height=30
breadth=40

***Obj3=Obj1+Obj2***

Height=40
breadth=60

***Obj4=-Obj3***

Height=-40
breadth=-60

***Obj5=Obj3+10***

Height=50
breadth=70
```

```
***Obj4=--Obj3***

Height=39
breadth=59

***Obj4=--Obj3***

Height=39
breadth=59

Press any key to continue . . .
```

Type 2: runtime polymorphism/Overriding:

- By runtime polymorphism we can point to any derived class from the object of the base class at runtime that shows the ability of runtime binding.

Sample program for RunTime polymorphism/overriding:

```
class Shape
  {
     public virtual void Draw()
     {
     }
  }
  class Ractangel:Shape
  {
     public override void Draw()
     {
        Console.WriteLine("Rectangle Drawn ");
     }

  }
   class Circle:Shape
  {
     public override void  Draw()
     {
        Console.WriteLine("Circle Drawn ");

     }
  }
   class Traingle:Shape
  {
     public override void Draw()
     {
        Console.WriteLine("Triangle Drawn ");
```

```
    }
  }
 static void Main(string[] args)
    {
      /* Testing Polymorphism */

      Shape[] s = new Shape[3];
      /* Polymorphic Objects */
      /* creating Array with Different types of Objects */
      s[0] = new Circle();
      s[1] = new Ractangel();
      s[2] = new Traingle();

   Console.WriteLine("\n\nRuntime polymorphism \n\n");
      for (int i = 0; i < 3; i++)
      {
        s[i].Draw();
      }
      Console.Read();
    }
```

Output:

```
Runtime polymorphism

Circle Drawn
Rectangle Drawn
Triangle Drawn
```

10.Inheritance

Note:

- The class from which an another class derives is called Base class.
- Base class can be also called as parent class, superclass or ancestors.
- The class which is derived from the base class is called derived class
- Derived class can be also called as child class or subclass.
- Derived class follows "IS A kind of" relationship with the base class.

Syntax:

```
class A
{
        ....
        Public void method1()
        {
                ...
        }
}

class  B : A
{
        Public void method2()
        {
                ...
        }
}

Class pgm
{
Static void main(string[] args)
{
B obj;
Obj.method1();
Obj.method2();
}
}
```

Inheritance - Types:

1. Single inheritance

When a single derived class is created from a single base class then the inheritance is called as single inheritance.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstProgram
{
   class BaseClass
   {
      public void displayBase()
      {
         System.Console.WriteLine("I am Base class");
      }
   }
   class DerivedClass : BaseClass //DerivedClass is child of BaseClass
   {
      public void displayDerived()
      {
         System.Console.WriteLine("I am Derived class");
      }
   }

   class MainClass
   {
      public static void Main()
      {
         DerivedClass x = new DerivedClass();//Normally object of child
         x.displayBase();
         x.displayDerived();
      }
   }

}
```

Output:

```
I am Base class
I am Derived class
Press any key to continue . . .
```

2.Multi level inheritance

When a derived class is created from another derived class, then that inheritance is called as multi level inheritance.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstProgram
{
   class BaseClass
   {
      public void display()
      {
         System.Console.WriteLine("Base class!!");
      }
   }
   class Derived1 : BaseClass //Derived1 is child of BaseClass
   {
      public void display1()
      {
         System.Console.WriteLine("Derived class1");
      }
   }
   class Derived2 : Derived1 //Derived2 is child of Derived1
   {
      public void display2()
      {
         System.Console.WriteLine("Derived Class2");
      }
   }

   class MainClass
   {
      public static void Main()
      {
         Derived2 x = new Derived2();//Normally object of child
         x.display();
         x.display1();
         x.display2();
      }
   }
}
```

Output:

```
Base class!!
Derived class1
Derived Class2
Press any key to continue . . .
```

3.Multiple inheritance

when a derived class is created from more than one base class then that inheritance is called as multiple inheritance.
Multiple inheritance is not supported by C# But it can be achieved using interfaces.
Example is given in the interfaces topic.

4.Multipath inheritance

When a class is derived from many other class which are all derived from the common base class. This is not supported in C#.

5.Hierarchial inheritance

when more than one derived class are created from a single base class,
then that inheritance is called as hierarchical inheritance.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstProgram
{
   class BaseClass
   {
      public void display()
      {
         System.Console.WriteLine("Calling Base class method using derived class object!!");
      }
   }
   class Derived1 : BaseClass //Derived1 is child of BaseClass
   {
      public void display1()
      {
         System.Console.WriteLine("Inside Derived class1");
         display();
      }
   }
   class Derived2 : BaseClass//Derived2 is child of BaseClass
```

```
  {
    public void display1()
    {
      System.Console.WriteLine("Inside Derived Class2");
      display();
    }
  }
  class Derived3 : BaseClass//Derived3 is child of BaseClass
  {
    public void display1()
    {
      System.Console.WriteLine("Inside Derived Class3");
      display();
    }
  }

  class MainClass
  {
    public static void Main()
    {
      Derived1 x = new Derived1();
      Derived2 y = new Derived2();
      Derived3 z = new Derived3();
      Console.WriteLine("\n********Hierarchical Inheritance example*******\n");
      x.display1();
      y.display1();
      z.display1();
    }
  }
}
```

Output:

```
********Hierarchical Inheritance example*******

Inside Derived class1
Calling Base class method using derived class object!!
Inside Derived Class2
Calling Base class method using derived class object!!
Inside Derived Class3
Calling Base class method using derived class object!!
Press any key to continue . . .
```

6.Hybrid Inheritance

Any combination of single, hierarchical and multi level inheritances is called as hybrid inheritance.

11.Abstract Class

Note:

- An Abstract class cannot be instantiated.
- Abstract class should be having the keyword “abstract” prefixed with the class keyword.
- Abstract methods should be prefixed with the “abstract” keyword.
- Class that extends the abstract class should provide definitions for all the abstract methods and the abstract methods that are defined should be prefixed with “override” keyword.
- Once the class has defined all the abstract methods, then we can create an object instance for that class.

Need for abstract class:

- In many applications ,we would like to have one base class and a number of different derived classes.
- A Base Class simply act as a base for others ,we might not create its objects.
- Under such situations, abstract classes can be used.

Syntax:

```
abstract class Base
{
        abstract void method1();

        void method2()
        {
                ...
                ...
        }
}

class Derived : Base
{
        Method1()
        {
                ...
        }

        method3()
        {
                ...
        }
}
class Program
{
static void main(String[] args)
{
```

```
        Base b1= new Base();  // it is invalid
        Derived d1=new Derived(); // it is valid
}
}
```

Note: In the above syntax, class B extends an abstract class. Hence class B should provide the method definition for the method which is not defined in class A. If the method definition is not provided, then class B will also be an abstract class and the keyword "abstract" should be prefixed before the class B.

Example for Abstract class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AbstractClass
{
   abstract class baseclass
   {
      protected int num1, num2,num3;

      public baseclass()
      {
         Console.WriteLine("\n****Assigning 0 to num1 and num2******\n");
         num1 = 0;
         num2 = 0;

      }
      public baseclass(int a, int b)
      {
         Console.WriteLine("\n*******Assigning  {0},{1} to num1 and num2*****\n",num1,num2);
         num1 = a;
         num2 = b;
      }
      public abstract void add();
       public abstract void display();
   }
   class derivedclass : baseclass
   {
       public derivedclass():base()
       {
          Console.WriteLine("\nDefault base class constructor is called!!!!\n");
       }
       public derivedclass(int n1, int n2): base(n1, n2)
       {
          Console.WriteLine("\nParameterized base class constructor is called!!!!\n");
       }
```

```
    public override void add()
    {
      num3=num1+num2;
    }
    public override void display()
    {
      Console.WriteLine("\nSum of{0} and {1} is:{2}\n", num1, num2, num3);
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("\n*******Calling default constructor from derived class*******\n");
      derivedclass obj = new derivedclass();
      obj.add();
      obj.display();
      Console.WriteLine("\n********Calling parameterized constructor from derived class******\n");
      derivedclass obj1 = new derivedclass(30,50);
      obj1.add();
      obj1.display();


    }
  }
}
```

Output:

```
*******Calling default constructor from derived class*******

****Assigning 0 to num1 and num2******

Default base class constructor is called!!!!

Sum of0 and 0 is:0

********Calling parameterized constructor from derived class******

*******Assigning  0,0 to num1 and num2*****

Parameterized base class constructor is called!!!!
```

```
Sum of30 and 50 is:80

Press any key to continue . . .
```

12.Interface

Notes to remember:

- It is a contract/ a rule.
- Interface can be considered as a **pure abstract class.**
- By default its access specifier is public.
- It provides only method declaration.
- It does not allow the declaration of data members. But we can declare a data member as an argument / parameter in a method declaration.
- Multiple inheritance can be achieved using interface.
- Class which implements the

Syntax:

```
Interface A
{
    void method1();
    void method2(int x);
}

class B : A
{
    void method1()
    {
            ....
            ....
    }

    void method2(int x)
    {
            ....
            ....
    }
}
```

Nested Interface:

```
Interface A
{
    void method1();
}
```

Interface B : A ***//Note: interface B extends the interface A. Now, Interface B will contain all the method definitions present in interface A and also its own method definitions.***

```
{
    void method2(int x);
}
```

class C : B ***//Note: Class C extends interface B . Hence Class C should provide definitions for all the methods which are declared in the interface B. If not, The class C becomes an Abstract class.***

```
{
    void method1()
    {
        ....
        ....
    }

    void method2(int x)
    {
        ....
        ....
    }
}
```

<u>Multiple inheritance using interface:</u>

```
Interface A
{
    void method1();
}
Interface B
{
    void method2(int x);
}

class C : A,B
 {
    void method1()
    {
        ....
        ....
    }

    void method2(int x)
    {
        ....
        ....
    }
```

}

Example program for Multiple interfaces – achieving multiple inheritance

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MyFirstProgram
{

   interface Inter1
   {
      void display(); //it cannot be public
      //By default it is abstract and public
   }
   interface Inter2
   {
      void display1();
   }
   class InterfaceClass : Inter1, Inter2
   {
      public void display()
      {
         System.Console.WriteLine("Interface one!!!");
      }
      public void display1()
      {
         System.Console.WriteLine("Interface two!!!");
      }
   }
   class d
   {
      public static void Main()
      {
         InterfaceClass t = new InterfaceClass();
         t.display();
         t.display1();
      }
   }
}
```

Output:

```
Interface one!!!
Interface two!!!
Press any key to continue . . .
```

Difference between abstract class and Interface:

Abstract class	Interface
1) It can contain method definition for all the methods except atleast one method which is left undefined.	1)No Method definitions are allowed.
2) Access modifiers are private	2)Access modifiers are Public

13.Partial class

Notes to remember:

- Using the **partial** indicates that other parts of the class, struct, or interface can be defined within the namespace.
- All of the parts must be available at compile time to form the final type.
- All the parts must have the same accessibility, such as public, private, and so on
- If any parts are declared abstract, then the entire type is considered abstract
- If any parts are declared sealed, then the entire type is considered sealed.
- If any of the parts declare a base type, then the entire type inherits that class.

Syntax:

```
partial class A
{
    public void method1()
    {
            ....
    }

    partial void method2(); //note: This method is not mandatory to be implemented here. But
atleast once  it should be declared in the program.
}


partial class A
{
    pubic void method1()
    {
            ....
    }

    public void method2()
    {
            ....
    }
}
```

Sample Program that uses partial class

```
class Program
{
  static void Main()
  {
    A.A1();
    A.A2();
  }
```

```
}
```

<Contents of file A1.cs >

```
using System;

partial class A
{
   public static void A1()
   {
      Console.WriteLine("A1");
   }
}
```

<Contents of file A2.cs >

```
using System;

partial class A
{
   public static void A2()
   {
      Console.WriteLine("A2");
   }
}
```

Output:

```
A1
A2
Press any key to continue . . .
```

Note:

- Partial is required here. If you remove the partial modifier, you will get an error containing this text: [The namespace '<global namespace>' already contains a definition for 'A'].
- To fix this, you can either use partial, or change one of the class names.

14.Properties & Indexers

- Properties are used to access private fields of class in applications.
- More flexible and maintenance
- Provides get and set accessors.
- Properties cannot have void. It should definitely have a return type.
- Indexes are also called as “smart arrays”.
- Sharing the private arrays can be done with indexes.

Syntax for property:

```
returntype propertyName
{
        get
        {
                ...
        }

        set
        {
                ...
        }
}
```

Example:

```
class A
{
        private int _id; //Note: _id is a data member
        public int Id //Note: Id is a property of the data member _id.
        {
                get
                {
                        return _id;
                }
                set
                {
                        _id=value; //Note: value is a keyword here.
                }
        }
}
```

Example for using get and set accessor:

Note: get and set accessor enables us to access private variables outside the class.

<StudentInfo.cs> //implements get and set accessors

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace StudentInfo
{
   class StudentInfo
   {
      private int _id;
      private string _name;
      private int _age;

      public int Id
      {
         get  //We use get accessor. so we can get/retrieve this variable value in main method also.
         {
            return _id;
         }
         set //We use set accessor. so we can set/change this variable value in main method also.
         {
            _id = value;
         }
      }
      public string Name
      {
         get
         {
            return _name;
         }
         set
         {
            _name = value;
         }
      }
      public int Age
      {
         get
         {
            return _age;
```

```
        }
        set
        {
            _age = value;
        }
    }

  }
}
```

<u><program.cs></u> //creating objects for the above class and accessing using get and set accessors.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace StudentInfo
{
  class Program
  {
    static void Main(string[] args)
    {
        Console.WriteLine("\n*****Initialization of object using set accessors*****\n");
        StudentInfo[] obj = new StudentInfo[5];
        for (int i = 0; i < 5; i++)
        {
            obj[i] = new StudentInfo();
            Console.WriteLine("ID? ");
            obj[i].Id = int.Parse(Console.ReadLine());

            Console.WriteLine("Name? ");
            obj[i].Name = Console.ReadLine();

            Console.WriteLine("Age? ");
            obj[i].Age = int.Parse(Console.ReadLine());
        }

        Console.WriteLine("\n*****displaying using get accessors*****\n");

        foreach (StudentInfo obj1 in obj)
        {
            Console.WriteLine("Name={0}\tID={1}\tAge={2}", obj1.Name, obj1.Id, obj1.Age);
        }

    }
  }
```

}

Output:

```
*****Initialization of object using set accessors*****

ID?
100
Name?
Lenaprasanna
Age?
22
ID?
200
Name?
Chanakya
Age?
21
ID?
300
Name?
Kishore
Age?
21
ID?
400
Name?
Harsha
Age?
21
ID?
500
Name?
Saishivaprasad
Age?
22

*****displaying using get accessors*****

Name=Lenaprasanna       ID=100  Age=22
Name=Chanakya   ID=200  Age=21
Name=Kishore    ID=300  Age=21
Name=Harsha     ID=400  Age=21
Name=Saishivaprasad     ID=500  Age=22
Press any key to continue . . .
```

Note:

- properties does not end with parenthesis().
- Getter method should return the value, and setter method should assign a value.
- Atleast one accessor like get or set must be present if we use a property. It is not mandatory to use both get and set.
- If the property has only get, no one can assign a value to the variable to which the property is created. It can only read the value and returns it. If it has set, then we can assign the value.

Syntax for Auto Implemented Property:

```
class A
{
public int Id
{
get;
set;
}
}
```

Note: Auto implemented property can be given only if both get and set accessor are used together. If we need to make Auto Implemented property just to return values, we can declare set as private.

(ie.)

```
Public int Id
{
get;
private set;  // This will not allow us to set the value for Id. We need to use constructor initialization if we need to set the value for Id.
}
```

Indexers:

Note:

- An indexer provides array-like syntax.
- It allows a type to be accessed the same way as an array.
- Properties such as indexers often access a backing store.
- With indexers you often accept a parameter of int type and access a backing store of array type.
- An indexer is a member that enables an object to be indexed in the same way as an array.

Sample program for Indexers:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Indexer
{
   class IndexerClass
   {
      string[] _values = new string[100]; // Backing store
```

```
    public string this[int number]
    {
        get
        {
            // This is invoked when accessing Layout instances with the [ ].
            if (number >= 0 && number < _values.Length)
            {
                // Bounds were in range, so return the stored value.
                return _values[number];
            }
            // Return an error string.
            return "Error";
        }
        set
        {
            // This is invoked when assigning to Layout instances with the [ ].
            if (number >= 0 && number < _values.Length)
            {
                // Assign to this element slot in the internal array.
                _values[number] = value;
            }
        }
    }
}

class Program
{
    static void Main()
    {
        // Create new instance and assign elements
        // ... in the array through the indexer.
        IndexerClass objIndexer = new IndexerClass();
        objIndexer[1] = "Lenaprasanna";
        objIndexer[3] = "Kishore";
        objIndexer[10] = "Chanakya";
        objIndexer[11] = "Telugu";
        objIndexer[-1] = "Error";
        objIndexer[1000] = "Error";

        // Read elements through the indexer.
        string value1 = objIndexer[1];
        string value2 = objIndexer[3];
        string value3 = objIndexer[10];
        string value4 = objIndexer[11];
        string value5 = objIndexer[50];
        string value6 = objIndexer[-1];
```

```
        // Write the results.
        Console.WriteLine(value1);
        Console.WriteLine(value2);
        Console.WriteLine(value3);
        Console.WriteLine(value4);
        Console.WriteLine(value5); // Is null
        Console.WriteLine(value6);
    }
  }
}
```

Output:

```
Lenaprasanna
Kishore
Chanakya
Telugu

Error
Press any key to continue . . .
```

15.Collections

Points to remember:

In C#, Data structures are called as collections. It provides functionality from IEnumerable interface. It supports any type of values. It supports large collection of datas.

- ArrayList, HashTable,name value collection are **"pure non generic collection".**
- List is a "**pure generic collection**".
- Stack has both **"Non generic collection"** and **"generic collection".**
- Stack follows Last In First Out(LIFO) Strategy.
- Queue follows First In First Out(FIFO) Strategy.
- Non generic collections are normally reference types. It will be automatically incremented with the size of the values.
- Hashtable and Name value collection stores value in key-value pairs.
- Common built in methods used in all collections are listed below. They are:
    - Add()
    - AddRange()
    - Sort()
    - remove()
    - removeAt()
- Stack has push(), pop()
- User defined methods are also allowed.

Notes to remember:

If we want to use Non-generic collections like array list , hast table,etc., we have to include "system.collections" to our program.
(ie.) use "using system.collections" before the namespace in the program.
Similarly if we want to use Generic collections like list , stack,etc., we have to include "system.collections.generic" to our program.
(ie.) use "using system.collections.generic" before the namespace in the program.

Important note on generic programming:

- There is no property called length. we use count property to find the number of collections present. It is similar to Length property.

Collections is classified into 2 types:

- Non-Generic Collections(system.collections)
    - Built in types
        - Array list
        - Hash table
        - Stack
        - Queue
        - Name Value Collection

        - Sorted List
        - And so on..
    - Use Class Collection
        - User defined classes as arrays (Implemented more in LinQ requirements)
- Generic Collections(system.collections.generic)
    - Built in generics
        - List
        - Stack
        - Queue
        - Sorted List
        - Dictionary
        - Linked List
        - And so on..
    - Generic class (class classname <T>)
    - Generic method (return_type methodname <T>)
    - Generic constraints
        - Where T:class
        - Where T:struct
        - Where T:InterfaceName
        - Where T:new()
        - Where T:U
    - User defined Generic classes.

Note: Generic constraints are the set of rules that are given

<u>Example program for Built in generic and non generic collections:</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections; //This should be included as we use ArrayList,stack etc.

namespace CollectionsDemo
 {
   static class UserMainCode
    {

      public static void ArrayListDatas()
      {
         Console.WriteLine("\n***Entering ArrayListDatas()****\n");
         ArrayList objArrList = new ArrayList(); //Pure non generic collection

         Console.WriteLine("\n***Adding 3 elements****\n");
         objArrList.Add(10);
```

```
objArrList.Add("Lenaprasanna");
objArrList.Add('g');

Console.WriteLine("\n***Inserting a new element 100.45 at 0th index****\n");
objArrList.Insert(0, 100.45); //Inserts 100.45 in the 0th index.

ArrayList objArrList2 = new ArrayList();
objArrList2.Add("Mani");
objArrList2.Add(234);

ArrayList objArrList3 = new ArrayList();
objArrList3.Add("Telugu");
objArrList3.Add("Tamil");
objArrList3.Add("English");


Console.WriteLine("\n***Initial objArrList ***\n");
foreach (var value in objArrList)
{
   Console.WriteLine(value);
}

Console.WriteLine("\n***Initial objArrList2 ***\n");
foreach (var value in objArrList2)
{
   Console.WriteLine(value);
}

Console.WriteLine("\n***Initial objArrList3 ***\n");
foreach (var value in objArrList3)
{
   Console.WriteLine(value);
}

Console.WriteLine("\n***Appending the value of objArrList to objArrList2 using
AddRange()****\n");

objArrList2.AddRange(objArrList);

Console.WriteLine("\n***printing objArrList2 after AddRange()***\n");
foreach (var value in objArrList2)
{
   Console.WriteLine(value);
}

Console.WriteLine("\nAppending the value of objArrList3 at the specified index of objArrList2
to objArrList2\n");
```

```
objArrList2.InsertRange(1, objArrList3);

Console.WriteLine("\nremoving the object \"English\" from the list\n");
objArrList3.Remove("English");

Console.WriteLine("\n***printing objArrList3 after Remove()***\n");
foreach (var value in objArrList3)
{
   Console.WriteLine(value);
}

Console.WriteLine("\n**printing objArrList2 after InsertRange()***\n");
foreach (var value in objArrList2) //note: var datatype accepts any type of value. If val a=10
is given, a will be of type int. if we give val a="Telugu", then a will be of type string.
{
   Console.WriteLine(value);
}

//removing the element at 3rd position of objArrList2

objArrList2.RemoveAt(3);

Console.WriteLine("\n***printing objArrList2 after RemoveAt()***\n");
foreach (var value in objArrList2)
{
   Console.WriteLine(value);
}

//sorting the elements and reverse them.

//objArrList2.Sort(); - This function will work only if the array list has the elements of same
datatype.

Console.WriteLine("\nPrinting the elements using for loop before reverse()\n");
for (int i = 0; i < objArrList.Count; i++)
{
   Console.WriteLine(objArrList[i]);
}

objArrList.Reverse();


Console.WriteLine("\nPrinting the elements using foreach loop after reverse()\n");
foreach (var value in objArrList)
{
   Console.WriteLine(value);
}
```

```
        Console.WriteLine("\n***checking whether the element in arraylist exists or not using
Contains()***\n");
        if (objArrList.Contains("Lenaprasanna"))
        {
           Console.WriteLine("\nCheck for \"Lenaprasanna\" in objArrList\n");
           Console.WriteLine("\nLenaprasanna exists in objArrList\n");
        }
        if (objArrList.Contains(10))
        {
           Console.WriteLine("\nCheck for \"10\" in objArrList\n");
           Console.WriteLine("10 exists in objArrList");
        }

        if (objArrList.Contains(500))
        {
           Console.WriteLine("\nCheck for \"500\" in objArrList\n");
           Console.WriteLine("500 exists in objArrList");
        }
        else
        {
           Console.WriteLine("\nCheck for \"500\" in objArrList\n");
           Console.WriteLine("500 not exist in objArrList");
        }
     }

      public static void StackDatas()
      {
         Console.WriteLine("\n\n****Entering StackDatas()******");
         Stack objstack1 = new Stack();
         objstack1.Push(452);
         objstack1.Push("prasanna");
         for (int i = 0; i < 15; i++)
         {
            objstack1.Push(i * 2);
         }
         Console.WriteLine("Elements before poping");
         foreach (var value in objstack1)
         {
            Console.WriteLine(value + " ");
         }

         if (objstack1.Count > 6)
         {
            objstack1.Pop();
         }
         Console.WriteLine("Elements after poping");
```

```
        foreach (var value in objstack1)
        {
            Console.WriteLine(value + " ");
        }
    }

    public static void HashTableDatas()
    {
        //syntax to create a hash table:
        //Hashtable<keypair,valuepair> objHashtable = new Hashtable<keypair,valuepair>
        Console.WriteLine("***Entering HashTableDatas()****\n");
        Hashtable objHashtbl = new Hashtable();
        objHashtbl.Add(101, "Tamil");
//Or the above statement can also be given as objHastbl[101]="Tamil";
        objHashtbl.Add(102, "Hindi");
        objHashtbl.Add(103, "English");
        objHashtbl.Add(104, "Telugu");

//Note:checking whether the total number of entries in ObjHashtbl is less than 5

        if (objHashtbl.Count < 5)
        {
            foreach (DictionaryEntry dicEntry in objHashtbl)
            {
                Console.WriteLine("KeyPair = " + dicEntry.Key + "\nValuePair= " + dicEntry.Value);
                //Key and Value are the properties of the type DictionaryEntry.
            }
        }

        Console.WriteLine("\n***Retreiving only the keys****\n");
        foreach (var value in objHashtbl.Keys)
        {
            Console.WriteLine("Keys = " + value+"\tHashCode ="+value.GetHashCode());
        }

        Console.WriteLine("\n***Retreiving only the values****\n");
        foreach (var value in objHashtbl.Values)
        {
            Console.WriteLine("Value = " + value+"\tHashCode ="+value.GetHashCode());
        }

        Console.WriteLine("\nRemoving the key 102\n");
        objHashtbl.Remove(102);

        foreach (DictionaryEntry dicEntry in objHashtbl)
        {
            Console.WriteLine("KeyPair = " + dicEntry.Key + "\tValuePair= " + dicEntry.Value);
```

```
            }

            Console.WriteLine("\nClearing the all the entries from the hash table\n");
            objHashtbl.Clear();
            Console.WriteLine("\nCleared Successfully\n");
        }
    }
      public static  void listDatas() //pure generic collections
{
    Console.WriteLine("\n***Entering Generic listDatas()****\n");
    List<char> listObj = new List<char>();
    listObj.Add('a');
    listObj.Add('e');
    listObj.Add('i');
    listObj.Add('o');
    listObj.Add('u');
     listObj.Add('b');

    foreach (var value in listObj)
    {
        switch (value)
        {
            case 'a':
                Console.WriteLine("Vowel a exists");
                break;
            case 'e':
                Console.WriteLine("Vowel e exists");
                break;
            case 'i':
                Console.WriteLine("Vowel i exists");
                break;
            case 'o':
                Console.WriteLine("Vowel o exists");
                break;
            case 'u':
                Console.WriteLine("Vowel u exists");
                break;
            default:
                Console.WriteLine(value + " is not a vowel");
                break;
        }
    }

}

public static void stackData() // generic collections
{
```

```
    Console.WriteLine("\n***Entering stackDatas()****\n");
    Stack<string> stackObj = new Stack<string>();
    stackObj.Push("Lena");
    stackObj.Push("Prasanna");
    stackObj.Push("Mani");
    stackObj.Push("Kishore");
    stackObj.Push("Chanakya");

    for(int i=0;i<5;i++)
    {
       Console.WriteLine("Popped element is :" + stackObj.Pop());
    }
 }

 public static void dictionaryData() // pure generic collection
 {
    Console.WriteLine("\n***Entering DictionaryDatas()****\n");
    Dictionary<int, string> dict1 = new Dictionary<int, string>();

    dict1.Add(101, "Santhosh");
    dict1.Add(102, "Kishore");
    dict1.Add(103, "Lenaprasanna");
    dict1.Add(104, "Harsha");
    dict1.Add(105, "Sivasaiprasad");

    for (int i = 101; i < 106; i++)
    {
       Console.WriteLine(dict1[i]);
    }

 }

}
```

*******Method call for the above program is made from another class.**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace CollectionsDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         UserMainCode.ArrayListDatas(); //Note: Since CollectionDatas() is a static method, we call it
using its class name instead of its object name.
```

```
            UserMainCode.StackDatas();
            UserMainCode.HashTableDatas();
            UserMainCode.listDatas();
            UserMainCode.stackData();
            UserMainCode.dictionaryData();
            Console.Read();
        }
    }
}
```

Note: Since ArrayListDatas(),StackDatas(),hashTableDatas() is a static method, we call it using its class name instead of its object name.

Note: To run the program, build the UserMainCode.cs before running Program.cs
Build can be done by pressing ctrl+shift+B in Visual Studio 2010.

Output:

```
***Entering ArrayListDatas()****


***Adding 3 elements****


***Inserting a new element 100.45 at 0th index****


***Initial objArrList ***

100.45
10
Lenaprasanna
g

***Initial objArrList2 ***

Mani
234

***Initial objArrList3 ***

Telugu
Tamil
English

***Appending the value of objArrList to objArrList2 using AddRange()****


***printing objArrList2 after AddRange()***

Mani
234
100.45
```

```
10
Lenaprasanna
g

Appending the value of objArrList3 at the specified index of objArrList2 to
objArrList2


removing the object "English" from the list


***printing objArrList3 after Remove()***

Telugu
Tamil

**printing objArrList2 after InsertRange()***

Mani
Telugu
Tamil
English
234
100.45
10
Lenaprasanna
g

***printing objArrList2 after RemoveAt()***

Mani
Telugu
Tamil
234
100.45
10
Lenaprasanna
g

Printing the elements using for loop before reverse()

100.45
10
Lenaprasanna
g

Printing the elements using foreach loop after reverse()

g
Lenaprasanna
10
100.45

***checking whether the element in arraylist exists or not using Contains()***


Check for "Lenaprasanna" in objArrList
```

```
Lenaprasanna exists in objArrList


Check for "10" in objArrList

10 exists in objArrList

Check for "500" in objArrList

500 not exist in objArrList


****Entering StackDatas()******
Elements before poping
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
prasanna
452
Elements after poping
26
24
22
20
18
16
14
12
10
8
6
4
2
0
prasanna
452
***Entering HashTableDatas()****

KeyPair = 104
ValuePair= Telugu
KeyPair = 103
ValuePair= English
KeyPair = 102
```

```
ValuePair= Hindi
KeyPair = 101
ValuePair= Tamil

***Retreiving only the keys****

Keys = 104      HashCode =104
Keys = 103      HashCode =103
Keys = 102      HashCode =102
Keys = 101      HashCode =101


***Retreiving only the values****

Value = Telugu  HashCode =-804559291
Value = English HashCode =-794798841
Value = Hindi   HashCode =-341291271
Value = Tamil   HashCode =1162861271

Removing the key 102

KeyPair = 104   ValuePair= Telugu
KeyPair = 103   ValuePair= English
KeyPair = 101   ValuePair= Tamil

Clearing the all the entries from the hash table


Cleared Successfully


***Entering Generic listDatas()****

Vowel a exists
Vowel e exists
Vowel i exists
Vowel o exists
Vowel u exists
b is not a vowel

***Entering stackDatas()****

Popped element is :Chanakya
Popped element is :Kishore
Popped element is :Mani
Popped element is :Prasanna
Popped element is :Lena

***Entering DictionaryDatas()****

Santhosh
Kishore
Lenaprasanna
Harsha
Sivasaiprasad
```

Example program for user defined generic collections:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CollectionsDemo
{

   class genericCollections //Normal Class with generic  method
   {
      public static T GetData<T>(T x) // Type parameterized user defined generic method. T is the type
parameter. x can handle any datatype.
      {
         Console.WriteLine("\n*****Entering getdata()******\n");
         Console.WriteLine("\ninside Function: "+x);
         return x;
      }
      public static void swap<T>(ref T x,ref T y)
      {
         Console.WriteLine("\n******Entering swap()********\n");
         T temp;
         temp = x;
         x = y;
         y = temp;

      }

   }
   class GenericClassExample<T>  //generic class with generic method
   {
      private static  T num1, num2;
      public static void getdatas(T x, T y) //since method is static, the variables should be also declared
as static
      {
         num1 = x;
         num2 = y;
      }
      public static void display()
      {
         Console.WriteLine("datas are {0} and {1}" ,num1,num2);
      }

   }

}
```

*******Another class with main method to call the above methods.**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace CollectionsDemo
{
   class Program
   {
      static void Main(string[] args)
      {

         //calling generic functions.

         Console.WriteLine("\nValue in main:" + genericCollections.GetData<int>(200));
         Console.WriteLine("\nValue in main:" + genericCollections.GetData<char>('a'));
         Console.WriteLine("\nValue in main:" + genericCollections.GetData<float>(200.05f));
         Console.WriteLine("\nValue in main:" + genericCollections.GetData<string>("lenaprasanna"));

         int a, b;
         a = 50;
         b = 60;
          Console.WriteLine("\nBefore calling swap():" +"a="+ a+ "\tb="+ b);
         genericCollections.swap<int>(ref a, ref b);
         Console.WriteLine("\nafter calling swap():" + "a=" + a + "\tb=" + b);

        //calling Generic class
         GenericClassExample<int>.getdatas(20, 40);
         GenericClassExample<int>.display();

         GenericClassExample<char>.getdatas('a', 'b');
         GenericClassExample<char>.display();
         GenericClassExample<string>.getdatas("lena", "Prasanna");
         GenericClassExample<string>.display();

         GenericClassExample<object>.getdatas(100, "adf");
         GenericClassExample<object>.display();



         Console.Read();
      }
   }
}
```

Output:

```
*****Entering getdata()******

inside Function: 200

Value in main:200

*****Entering getdata()******

inside Function: a

Value in main:a

*****Entering getdata()******

inside Function: 200.05

Value in main:200.05

*****Entering getdata()******

inside Function: lenaprasanna

Value in main:lenaprasanna

Before calling swap():a=50      b=60

******Entering swap()********

after calling swap():a=60       b=50
datas are 20 and 40
datas are a and b
datas are lena and Prasanna
datas are 100 and adf
```

Brief explanation on the usage of get and set accessor is given in the below example with the help of user defined person class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CollectionsDemo
{
   class Persons
   {
```

```
int _id;
string _name;
int _age;
public int Id
{
   get  //We use get accessor. so we can get/retrieve this variable value in main method also.
   {
      return _id;
   }
   set //We use set accessor. so we can set/change this variable value in main method also.
   {
      _id = value;
   }
}
public string Name
{
   get
   {
      return _name;
   }
   set
   {
      _name = value;
   }
}
public int Age
{
   get
   {
      return _age;
   }
   set
   {
      _age = value;
   }
}
public Persons()
{
   _id = 0;
   _name = String.Empty;
   _age = 0;
}
public Persons(int no, string pname, int pAge)
{
   _id = no;
   _name = pname;
   _age = pAge;
```

```
      }
   }
}
```

***********Another class with main method calling the above class***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace CollectionsDemo
{
   class Program
   {
      static void Main(string[] args)
      {

         Console.WriteLine("\n********EXPLICIT INITIALIZATION USING CONSTRUCTOR ********\n");
         Persons personObj = new Persons(100, "Lenaprasanna Mani", 21);
         Console.WriteLine("Id=" + personObj.Id + "\tName=" + personObj.Name + "\tAge=" +
personObj.Age);
```

//Note: Properties like Id,Name,Age are accessed from the main. Properties of private variables in a class enables to access them outside the class. In general, any private variables cannot be used/accessed outside the class.

```
         int PERSON = 0; //Variable that is used to print the count of the person.

         //Person class initialized as Array with constructor initialization
         Persons[] objPersons = new Persons[5];
         objPersons[0] = new Persons(101, "Chanakya Mukku", 21);
         objPersons[1] = new Persons(102, "Sri Harsha Chilla", 21);
         objPersons[2] = new Persons(103, "Sai Shiva Prasad", 21);
         objPersons[3] = new Persons(104, "Kishore Sivagnanam", 21);
         objPersons[4] = new Persons(105, "Bharath Arunagiri", 21);
```

//person class initialized as Array with object initialization

```
         Console.WriteLine("\n********EXPLICIT INITIALIZATION USING PROPERTIES ********\n");
         Persons[] p = new Persons[4];
         p[0] = new Persons() { Id = 106, Name= "Tamil language", Age = 100 };
         p[1] = new Persons() { Id = 107, Name = "Telugu language", Age = 100 };
         p[2] = new Persons() { Id = 108, Name = "Hindi language", Age = 50 };
         p[3] = new Persons() { Id = 109, Name = "English language", Age = 75 };
```

//or

```
 //we can also initialize each object like p[0].Id = 200;
 //p[0].Name ="Andhrapradesh";
//p[0].Age = 22;

 //or

 Persons[] objPersonAlt=new Persons[4]
 {
 new Persons() { Id = 106, Name="Tamil", Age = 100 },
 new Persons() { Id = 107, Name="Telugu", Age = 100 },
 new Persons() { Id = 108, Name="Hindi", Age = 75 },
 new Persons() { Id = 109, Name="English", Age = 90}
  };

 //displaying all the persons details

 foreach (Persons obj in objPersons)
 {
    PERSON++;
    Console.WriteLine("\nPerson "+PERSON+" Details:");
    Console.WriteLine("Id=" + obj.Id + "\tName=" + obj.Name + "\tAge=" + obj.Age);
 }

 foreach (Persons obj in p)
 {
    PERSON++;
    Console.WriteLine("\nPerson " + PERSON + " Details:");
    Console.WriteLine("Id=" + obj.Id + "\tName=" + obj.Name + "\tAge=" + obj.Age);
 }

 foreach (Persons obj in objPersonAlt)
 {
    PERSON++;
    Console.WriteLine("\nPerson " + PERSON + " Details:");
    Console.WriteLine("Id=" + obj.Id + "\tName=" + obj.Name + "\tAge=" + obj.Age);
 }


 //Using list to hold and display the User Defined Object

 List<Persons[]> UserDefinedList = new List<Persons[]>();
 UserDefinedList.Add(objPersons);
 UserDefinedList.Add(p);
 UserDefinedList.Add(objPersonAlt);

 foreach (Persons[] Individualobj in UserDefinedList)
 {
```

```
            foreach (Persons Arrayobj in Individualobj)
            {
               Console.WriteLine("\nId=" + Arrayobj.Id + "\tName=" + Arrayobj.Name + "\tAge=" +
Arrayobj.Age);
            }
        }

        Console.Read();

    }
  }
}
```

Output:

```
********EXPLICIT INITIALIZATION USING CONSTRUCTOR ********

Id=100  Name=Lenaprasanna Mani  Age=21

********EXPLICIT INITIALIZATION USING PROPERTIES ********


Person 1 Details:
Id=101  Name=Chanakya Mukku     Age=21

Person 2 Details:
Id=102  Name=Sri Harsha Chilla  Age=21

Person 3 Details:
Id=103  Name=Sai Shiva Prasad   Age=21

Person 4 Details:
Id=104  Name=Kishore Sivagnanam Age=21

Person 5 Details:
Id=105  Name=Bharath Arunagiri  Age=21

Person 6 Details:
Id=106  Name=Tamil language     Age=100

Person 7 Details:
Id=107  Name=Telugu language    Age=100

Person 8 Details:
Id=108  Name=Hindi language     Age=50

Person 9 Details:
Id=109  Name=English language   Age=75

Person 10 Details:
Id=106  Name=Tamil      Age=100
```

```
Person 11 Details:
Id=107  Name=Telugu     Age=100

Person 12 Details:
Id=108  Name=Hindi      Age=75

Person 13 Details:
Id=109  Name=English    Age=90

Id=101  Name=Chanakya Mukku     Age=21

Id=102  Name=Sri Harsha Chilla  Age=21

Id=103  Name=Sai Shiva Prasad   Age=21

Id=104  Name=Kishore Sivagnanam Age=21

Id=105  Name=Bharath Arunagiri  Age=21

Id=106  Name=Tamil language     Age=100

Id=107  Name=Telugu language    Age=100

Id=108  Name=Hindi language     Age=50

Id=109  Name=English language   Age=75

Id=106  Name=Tamil      Age=100

Id=107  Name=Telugu     Age=100

Id=108  Name=Hindi      Age=75

Id=109  Name=English    Age=90
```

16.Exception Handling in C#

Note:

- Exception provides 2 types of classes:
  1.Application.Exception
  2.System.Exception
- These both class are extended from a common base class called "Exception".
- Some of the Exceptions are:
  - IndexOutOfRangeException
  - FileNotFoundException
  - FormatException
  - UserDefinedException
  - DivideByZeroException
  - And so on..

Exception in C# has 3 blocks:

1.**Try block** – try block has set of codes which arises exception. A module can have only one try block. It can be nested.
2.**Catch block**- it will be executed if the try block got any exception. There can be a number of catch blocks for a single try block. Only one catch block will be executed at a time. Catch block will have parameters which will find the type of exceptions and it will throw the exception using "throw" keyword. It is not always mandatory to have catch.
3.**Finally block** – There can be one finally block. It is used to release the memory, closing the resources, etc. Finally will be executed all the time even if exceptions are found or not, but catch block will be executed only when exception is occurred.

Syntax:

```
try
{
...   //try block has set of exceptions occuring statements
...
}
catch(...)
{
... // Catch block has set of exception handling codes
...
}
catch(...)
{
...
...
}
```

```
finally
{
... //Finally block has set of codes for freeing up the used resources
...
}
```

Points to remember in Catch block:

Catch(exception e1)
{
e1.Message;// ***Message Property gives the type of exception occurred***
e1.Source; // ***Source Property gives the namespace name where the exception has occurred***
e1.StackTrace;// ***StackTrace Property gives the method or sub method name where the exception has occurred***
e1.TargetSite;//***TargetSite property gives the name of the method that cause the exception***
}

Points to remember in throw block:

- Exceptions are thrown by “throw” keyword.
- We throw new exceptions by throw new Exception(“Exception Occurred”,e1);

Example for Exception handling:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExceptionHandling
{
   class ExceptionProgram
   {
      public void ExceptionHandling()
      {

         try
         {
            int value = 1 / int.Parse("0");
         }
         catch (Exception ex)
         {
            Console.WriteLine("\nHelpLink = "+ ex.HelpLink);
            Console.WriteLine("\nMessage = "+ ex.Message);
            Console.WriteLine("\nSource = "+ ex.Source);
            Console.WriteLine("\nStackTrace = "+ ex.StackTrace);
            Console.WriteLine("\nTargetSite = "+ ex.TargetSite);
```

```
            Console.WriteLine();
        }

    }
    static void Main(string[] args)
    {
        ExceptionProgram obj = new ExceptionProgram();
        obj.ExceptionHandling();
    }
  }
}
```

Output:

```
HelpLink =

Message = Attempted to divide by zero.

Source = ExceptionHandling

StackTrace =    at ExceptionHandling.ExceptionProgram.ExceptionHandling() in C:\
Users\346238\Documents\Visual Studio 2010\Projects\ExceptionHandling\ExceptionHa
ndling\ExceptionProgram.cs:line 15

TargetSite = Void ExceptionHandling()

Press any key to continue . . .
```

Note:

- HelpLink:
  This is empty because it was not defined on the exception.

- Message:
  This is a short description of the exception's cause.

- Source:
  This is the application name.

- StackTrace:
  This is the path through the compiled program's method hierarchy that the exception was generated from.

- TargetSite:
  This is the name of the method where the error occurred.

User Defined Exception Handling:

Using Data property:

It is possible to store structured data on an exception that is thrown in one part of our program, and then later read in this data. Using the Data dictionary on the Exception instance, we can store associative keys and values.

Sample Example Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections; //this should be included in the case of user Defined Exceptions inorder to use Data Property.


namespace ExceptionHandling
{
   class ExceptionProgram
   {

      public void userDefinedException()
      {
         try
         {
            // Create new exception.
            var ex = new DivideByZeroException("Message");
            // Set the data dictionary.
            ex.Data["Time"] = DateTime.Now;
            ex.Data["Flag"] = true;
            // Throw the exception
            throw ex;
         }
         catch (Exception ex)
         {
            // Display the exception's data dictionary.
            foreach (DictionaryEntry pair in ex.Data)
            {
               Console.WriteLine("{0} = {1}", pair.Key, pair.Value);
            }
         }
      }

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

```
        ExceptionProgram obj = new ExceptionProgram();
        obj.userDefinedExceptionuserDefinedException();
      }
    }
}
```

Output:

```
Time = 10/11/2012 1:04:16 PM
Flag = True
Press any key to continue . . .
```

17.File handling in C#

Methods for handling Files and directories:

Note:

- These file methods are defined in system.IO
- So it is mandatory to include System.IO in the program inorder to use these methods.
- In Visual Studio 2010, we have to include system.IO by adding "using System.IO" in the preprocessor directives and In case of METTL, It already has System.IO by default. If we add this system.IO to the METTL, it shows ambiguity error. So Use system.IO in the Visual studio 2010 alone and when submitting to METTL, please comment the line "using System.IO".
- Files with DAT extensions are Binary files.

Below are the file handling and directory handling methods:

- **Basic methods of FILE class:**
    - open()
    - create()
    - move()
    - copy()
    - exists()
    - delete()
    - ReadAllLines() – *It reads the datas line by line and returns the data in string[] format. Each lines of data will be in a single string.*
    - ReadAllText() – *The entire content in a file will be taken as a single string*
    - WriteAllLines()
    - WriteAllText()
    - ReadBytes()
    - AppendAllLines()
    - And so on..

- **Basic methods of DIRECTORY class:**
    - createDirectory()
    - exists()
    - delete()
    - and so on..

- **File Types** – *Uses FileStream for Data Manipulation*

    - **Text Files**

        - **Writing File**
            - TextWriter, StringWriter
            - StreamWriter (important)

- **Reading File content**
  - TextReader,StringReader
  - StreamReader (important)

- **<u>Binary Files</u>** – *used to share datas in the networks*

  - **Writing data to file**
    - BinaryWriter

  - **Reading datas**
    - BinaryReader

  - **Binary File Serialization and Deserialization**
    - BinaryFormatter

<u>Sample program illustrating default inbuilt file methods:</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO; // this should be included inorder to handle files or directories

namespace FileHandling
{
   class Program
   {
      static void Filemethod()
      {
         Console.WriteLine("******Check for file existance!!!********");
         if (File.Exists(@"D:\link.txt"))
            Console.WriteLine("file exists!!!\n The contents of the file is:\n");
         string[] linemsg = File.ReadAllLines(@"D:\link.txt");
         foreach (string line in linemsg)
         {
            Console.WriteLine(line);
         }

         string[] filecontent = new string[4] { "this is c# class", "I am Lenaprasanna", "Kishore is my Best
Friend", "I Love telugu" };
         File.AppendAllLines(@"D:\link.txt", filecontent);
         Console.WriteLine("******Using ReadAllLines() method********");
         string[] linemsg1 = File.ReadAllLines(@"D:\link.txt");
         Console.WriteLine("\nContents in the file is:\n");
         foreach (string line in linemsg1)
         {
```

```
        Console.WriteLine(line);
    }

    File.Copy(@"D:\link.txt", @"D:\link_copy.txt");
    File.Move(@"D:\link_copy.txt", @"D:\link_moved.txt");

    Console.WriteLine("******Using ReadAllText() method********");
   String totalContent = File.ReadAllText(@"D:\link_moved.txt");
   Console.WriteLine(totalContent);
   Console.WriteLine("******deleting the file********");
    File.Delete(@"D:\link.txt");
    bool exists = File.Exists(@"D:\link.txt");

    if (!exists)
    {
        Console.WriteLine("File deleted");

    }

}
static void DirectoryMethod()
{
    string[] files= new string[100];
    string[] docxfiles = new string[100];
    Directory.CreateDirectory(@"D:\Lenaprasanna");
    Console.WriteLine("Directory created successfully!!!");
    if(Directory.Exists(@"D:\Hands on programs"))
    {
       files= Directory.GetFiles(@"D:\Hands on programs");
       docxfiles = Directory.GetFiles(@"D:\Hands on programs", "*.cpp");
    }
    foreach (string file in files)
    {
        Console.WriteLine(files);
    }
    foreach (string file in docxfiles)
    {
        Console.WriteLine(files);
    }

    Directory.Delete(@"D:\Lenaprasanna");
    Console.WriteLine("Directory deleted successfully!!!");


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

```
        Filemethod();
        DirectoryMethod();
      }

   }
}
```

Output:

```
******Check for file existance!!!********
file exists!!!
 The contents of the file is:

I love telugu a lot!!!this is c# class
I am Lenaprasanna
Kishore is my Best Friend
I Love telugu

******Using ReadAllLines() method********

Contents in the file is:

I love telugu a lot!!!this is c# class
I am Lenaprasanna
Kishore is my Best Friend
I Love telugu

******Using ReadAllText() method********
I love telugu a lot!!!this is c# class
I am Lenaprasanna
Kishore is my Best Friend
I Love telugu
******deleting the file********
File deleted
Directory created successfully!!!
Directory deleted successfully!!!
Press any key to continue . . .
```

Example program for handling binary and text files is given below:

<TextFile.cs> ***//this C sharp program handles text files.***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO; //this should be included inorder to use File methods
namespace FileStreamManipulationDemo
{
   class TextFile
   {
      private FileStream fs1; // we need filestream obj for both read and write
      private StreamWriter sw;
      private StreamReader sr;
```

```
public void WriteFile()
{
   try
   {
      fs1 = new FileStream(@"D:\StreamFile.txt", FileMode.Create, FileAccess.Write,
FileShare.None);
      //Note: instantiation of filestream object. We are creating a file in the specified path and
opening it with write access.
      //Note: File format is need not be txt always. It can be anything like docx, etx.
      //FileMode.Create, FileAccess.Write are enumerations. the fourth parameter is optional.

      sw = new StreamWriter(fs1);
      sw.WriteLine("Hi! I am in C# File handling concept now!!!");
      sw.Write("My Mark is: ");
      sw.WriteLine(100);
      sw.WriteLine("I love telugu a lot!!!");
      sw.WriteLine("I know Tamil!!!");


      Console.WriteLine("File has been created!!!");
   }
   catch (FileNotFoundException e1)
   {
      throw new FileNotFoundException("Exception Found", e1);
   }
   finally
   {
      Console.WriteLine("Closing the resources!!");
      sw.Close();
      fs1.Close();
      fs1.Dispose();
   //Note: destroy objects using Dispose() method. It also releases all the resources used by the
object.
   }

}
public void ReadFile()
{
   try
   {
      fs1 = new FileStream(@"D:\StreamFile.txt", FileMode.Open, FileAccess.Read); //opening the
stored file  with read access.
      sr = new StreamReader(fs1);
      string msg = sr.ReadToEnd();
//Note: ReadToEnd() method takes entire data available in the file. If we use sr.ReadLine(). It takes
only one line in a file.
      Console.WriteLine("Content of Text file is:" + msg);
```

```
        }
        catch (FileNotFoundException e1)
        {
            Console.WriteLine("Exception Message is: " + e1.Message);
            Console.WriteLine("Exception source is: " + e1.Source);
        }
        finally
        {
            Console.WriteLine("Closing the resources!!");
            sr.Close();
            fs1.Close();
            fs1.Dispose();
        }
    }
    public void LineCount(string filename) //counts the number of lines in the given file
    {
        int count = 0;
        StreamReader r = new StreamReader(filename);

            string line;

            while ((line = r.ReadLine()) != null)
            {
                count++;
            }
            Console.WriteLine("\nNumber of Lines:" + count);

    }
  }
}
```

<BinaryFile.cs> ***//this c sharp program handles binary files***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace FileStreamManipulationDemo
{
  class BinaryFile
  {


    private FileStream fs1;
```

```
    private BinaryWriter bw;
    private BinaryReader br;
    public void BinaryFileWrite()
    {
      try
      {
        fs1 = new FileStream(@"D:\BinaryFile.DAT", FileMode.CreateNew, FileAccess.ReadWrite,
FileShare.None);
        bw = new BinaryWriter(fs1);
        bw.Write("Lenaprasanna");
        bw.Write(1000);
        Console.WriteLine("THE BINARY FILE HAS BEEN CREATED");
      }
      catch (Exception e1)
      {
        Console.WriteLine("Exception msg is  " + e1.Message);
      }
      finally
      {
        bw.Close();
        fs1.Close();
        fs1.Dispose();
      }
    }
    public void BinaryFileRead()
    {
      try
      {
        fs1 = new FileStream(@"D:\BinaryFile.dat", FileMode.Open);
        br = new BinaryReader(fs1);
        string name;
        int id;
        name = br.ReadString();
        id = br.ReadInt32();
        Console.WriteLine("Binary File Content is:");
        Console.WriteLine("Person name is {0} and id is {1}",name,id );
      }

      catch (FileNotFoundException e1)
      {
        Console.WriteLine("Exception msg is  " + e1.Message);
      }

      finally
      {
        br.Close();
      }
```

```
      }

   }
}
```

******** Another program that creates the instances of the above two classes and calls them***

<MainProgram.cs>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FileStreamManipulationDemo
{
   class MainProgram
   {
      static void Main(string[] args)
      {
         Console.WriteLine("\n*****HANDLING TEXT FILES*******\n");
         TextFile objTxtFile = new TextFile();
         Console.WriteLine("\n*****Writing datas to streamFile.txt*******\n");
         objTxtFile.WriteFile();
         Console.WriteLine("\n******Reading datas from streamFile.txt*******\n");
         objTxtFile.ReadFile();
         objTxtFile.LineCount("D:\\StreamFile.txt");
         Console.WriteLine("\n*****HANDLING BINARY FILES*******\n");
         BinaryFile objBinFile = new BinaryFile();
         Console.WriteLine("\n*****Writing datas to BinaryFile.txt*******\n");
         objBinFile.BinaryFileWrite();
         Console.WriteLine("\n******Reading datas from BinaryFile.txt*******\n");
         objBinFile.BinaryFileRead();

         Console.Read();


      }
   }
}
```

<u>Output:</u>

```
*****HANDLING TEXT FILES*******


*****Writing datas to streamFile.txt*******
```

```
File has been created!!!
Closing the resources!!

******Reading datas from streamFile.txt*******

Content of Text file is:Hi! I am in C# File handling concept now!!!
My Mark is: 100
I love telugu a lot!!!
I know Tamil!!!

Closing the resources!!

Number of Lines:4

*****HANDLING BINARY FILES*******


*****Writing datas to BinaryFile.txt*******

THE BINARY FILE HAS BEEN CREATED

******Reading datas from BinaryFile.txt*******

Binary File Content is:
Person name is Lenaprasanna and id is 1000
```

18.XML with C#

XML namespaces:

- System.xml
- System.xml.xsl
- System.xml.Schema
- System.xml.Path

XML Classes:

- XmlDocument
- XmlNode
- XmlElement
- XmlAttribute
- XmlNodeList

XML File Manipulation:

- XmlWriter -> XmlTextWriter (Text stream based class from XML Writer).
- XmlReader -> XmlTextReader(Test stream based Reader and Read Only Class).
- XmlValidatingReader validates in Auto, XSD,DTD technologies.

Note:

- XSL(XML Stylesheet Language), XSLT(XML Stylesheet Language Transformation), XML DOM(XML- Document Object Model), XML are some of the XML technologies.
- All the xml classes are inherited from System.Xml
- XMLNode is a parent class for all other classes.
- XMLAttribute is used to create elements or attributes
- XMLNodeList is used to create collection of XML nodes.

Example program for XML DOM :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml; //This should be included handling XML.

namespace XMLDomDemo
{
   class Program
   {
      static void CreateSimpleDocument()
      {
         //we are going to create a simple xml document in the following structure:
<EmployeeDetails>TeluguLanguage</EmployeDetails>

         //Note:The XML is basically parsed into an XmlNode which is the root element and then you can
access the child elements using the ChildNodes property. However, the XmlNode class gives you access
to a lot of other information as well, for instance the name of the tag, the attributes, the inner text
and the XML itself

          XmlDocument xDoc = new XmlDocument();

         //creating a node element in the xml document
          XmlNode xNode = xDoc.CreateElement("EmployeeDetails");

         xNode.InnerText = "TeluguLanguage";
          xDoc.AppendChild(xNode); //appending xNode on to the xml document

         // xDoc.Load(); //loads xml file into c#

         //saves the xml document on to the file in the specified
          xDoc.Save(@"D:\XMLFiles\XMLDemo1.xml");

          Console.WriteLine("\n****SIMPLE XML FILE IS CREATED by XML DOM TECHNOLOGY
CLASSES****\n");
      }

      static void CreateDocument()
      {

         XmlDocument xDoc = new XmlDocument();
         XmlNode xRoot = xDoc.CreateElement("EmployeeDetails"); //creating a root element in the xml
document
         xDoc.AppendChild(xRoot); //appending xRoot on to the xml document
```

```
        XmlNode xChild1 = xDoc.CreateElement("Employee1");
        XmlNode xChild1Name = xDoc.CreateElement("Name");
        xChild1Name.InnerText = "Lenaprasanna";
        XmlNode xChild1ID = xDoc.CreateElement("ID");
        xChild1ID.InnerText = "1001";
        XmlNode xChilds1Lang = xDoc.CreateElement("Language");
        xChilds1Lang.InnerText = "Telugu";
        xChild1.AppendChild(xChild1Name);
        xChild1.AppendChild(xChild1ID);
        xChild1.AppendChild(xChilds1Lang);



        XmlNode xChild2 = xDoc.CreateElement("Employee2"); //If we dint give any element or inner
text, it will produce an empty element tag <Employee2/>
        XmlNode xChild2Name = xDoc.CreateElement("Name");
        xChild2Name.InnerText = "Kishore";
        XmlNode xChild2ID = xDoc.CreateElement("ID");
        xChild2ID.InnerText = "1002";
        XmlNode xChilds2Lang = xDoc.CreateElement("Language");
        xChilds2Lang.InnerText = "Tamil";
        xChild2.AppendChild(xChild2Name);
        xChild2.AppendChild(xChild2ID);
        xChild2.AppendChild(xChilds2Lang);



        xRoot.AppendChild(xChild1); //appending xChild below the xRoot.
        xRoot.AppendChild(xChild2);

        xDoc.Save(@"D:\XMLFiles\XMLDemo2.xml"); //saves the xml document on to the file in the
specified
        Console.WriteLine("\n****XML FILE1 IS CREATED by XML DOM TECHNOLOGY CLASSES****\n");
    }

    static void CreateDocumentUsingElement()
    {//NOTE: Using XmlNode or XmlElement will produce same result.

        //The xml structure this method is going to produce the same as the above method had
produced.

        XmlDocument xDoc = new XmlDocument();
        XmlElement xRoot = xDoc.CreateElement("EmployeeDetails"); //creating a root element in the
xml document
        xDoc.AppendChild(xRoot); //appending xRoot on to the xml document
```

```
        XmlElement xChild1 = xDoc.CreateElement("Employee1");
        XmlElement xChild1Name = xDoc.CreateElement("Name");
        xChild1Name.InnerText = "Lenaprasanna";
        XmlElement xChild1ID = xDoc.CreateElement("ID");
        xChild1ID.InnerText = "1001";
        XmlElement xChilds1Lang = xDoc.CreateElement("Language");
        xChilds1Lang.InnerText = "Telugu";
        xChild1.AppendChild(xChild1Name);
        xChild1.AppendChild(xChild1ID);
        xChild1.AppendChild(xChilds1Lang);



        XmlElement xChild2 = xDoc.CreateElement("Employee2"); //If we dint give any element or inner
text, it will produce an empty element tag <Employee2/>
        XmlElement xChild2Name = xDoc.CreateElement("Name");
        xChild2Name.InnerText = "Kishore";
        XmlElement xChild2ID = xDoc.CreateElement("ID");
        xChild2ID.InnerText = "1002";
        XmlElement xChilds2Lang = xDoc.CreateElement("Language");
        xChilds2Lang.InnerText = "Tamil";
        xChild2.AppendChild(xChild2Name);
        xChild2.AppendChild(xChild2ID);
        xChild2.AppendChild(xChilds2Lang);

        xRoot.AppendChild(xChild1); //appending xChild below the xRoot.
        xRoot.AppendChild(xChild2);

        xDoc.Save(@"D:\XMLFiles\XMLDemo3.xml"); //saves the xml document on to the file in the
specified
        Console.WriteLine("\n****XML FILE2 IS CREATED by XML DOM TECHNOLOGY CLASSES****\n");


    }
    static void CreateDocumentUsingSingleVariableInSameLevel()
    {

        //Note:
        // After creating a new instance immediately we append that element. After appending the
element, we Can use the same instance again to Create another elements.

        XmlDocument xDoc = new XmlDocument();
        XmlNode xRoot = xDoc.CreateElement("EmployeeDetails"); //creating a root element in the xml
document
        xDoc.AppendChild(xRoot); //appending xRoot on to the xml document

        XmlNode xChild = xDoc.CreateElement("Employee1");
```

```
    //creating <Name>, <ID>and <laguage> under <Employee1>

    XmlNode xChildName = xDoc.CreateElement("Name");
    xChildName.InnerText = "Lenaprasanna";
    XmlNode xChildID = xDoc.CreateElement("ID");
    xChildID.InnerText = "1001";
    XmlNode xChildsLang = xDoc.CreateElement("Language");
    xChildsLang.InnerText = "Telugu";
    xChild.AppendChild(xChildName);
    xChild.AppendChild(xChildID);
    xChild.AppendChild(xChildsLang);

    //appending xChild below the xRoot.
    xRoot.AppendChild(xChild);

    xChild = xDoc.CreateElement("Employee2");

    //creating <Name>, <ID>and <laguage> under <Employee2>

    xChildName = xDoc.CreateElement("Name");
    xChildName.InnerText = "Kishore";
    xChildID = xDoc.CreateElement("ID");
    xChildID.InnerText = "1002";
    xChildsLang = xDoc.CreateElement("Language");
    xChildsLang.InnerText = "Tamil";
    xChild.AppendChild(xChildName);
    xChild.AppendChild(xChildID);
    xChild.AppendChild(xChildsLang);

    //appending xChild below the xRoot.
    xRoot.AppendChild(xChild);

    xDoc.Save(@"D:\XMLFiles\XMLDemo4.xml");
    Console.WriteLine("\n****XML FILE3 IS CREATED by XML DOM TECHNOLOGY CLASSES****\n");
}

static void createAttributesinXMLDoc()
{
    XmlDocument xDoc = new XmlDocument();
    XmlNode xRoot = xDoc.CreateElement("EmployeeDetails");
    xDoc.AppendChild(xRoot);

    //creating child node employee1
    XmlNode xChild = xDoc.CreateElement("Employee1");
    XmlNode xChildName = xDoc.CreateElement("Name");
    xChildName.InnerText = "Lenaprasanna";
```

```
        XmlAttribute xchildAttr = xDoc.CreateAttribute("City");
        xchildAttr.Value = "Tenali";

        xChild.AppendChild(xChildName);
        xChild.Attributes.Append(xchildAttr);
        xRoot.AppendChild(xChild);

        //creating child node employee2

        xChild = xDoc.CreateElement("Employee2");
        xChildName = xDoc.CreateElement("Name");
        xChildName.InnerText = "Kishore";
        xchildAttr = xDoc.CreateAttribute("City");
        xchildAttr.Value = "Chennai";

        xChild.AppendChild(xChildName);
        xChild.Attributes.Append(xchildAttr);
        xRoot.AppendChild(xChild);

        xDoc.Save(@"D:\XMLFiles\XMLDemoAttributes1.xml");
        Console.WriteLine("\n****XML FILE WITH ATTRIBUTES IS CREATED by XML DOM TECHNOLOGY
CLASSES****\n");

    }

    static void XMLTextWriterClass()
    {
        // Note:XmlTextWriter is the fastest writer class, and we proceed in the forward only way of
writing xml data. Similarly XmlTextReader is the fastest reader class

        // XmlTextWriter writer = new XmlTextWriter(Console.Out); // prints the result in the console

        XmlTextWriter writer = new
XmlTextWriter(@"D:\XMLFiles\XMLDemoAttributes1.xml",Encoding.UTF8);
        writer.Formatting= Formatting.Indented;
        writer.WriteStartDocument(true);

        writer.WriteStartElement("Employees");

            writer.WriteStartElement("Employee");//If we have a necessity to include more child elements
to it,then we can use this method. Using WriteElementString() method, we cant able to add childs
later.
                writer.WriteElementString("Medium","Telugu"); //the content will be added as a local
string name.
                writer.WriteElementString("Salary","20000$");
            writer.WriteEndElement();
```

```
        writer.WriteEndElement();
        writer.Close();
        Console.WriteLine("XmlTextWriter class is used");
    }
    static void XMLTextReaderClass()
    {
        XmlTextReader reader = new XmlTextReader(@"D:\XMLFiles\XMLDemoAttributes1.xml");
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
            //if the node is an element
            case XmlNodeType.Element:
                Console.WriteLine("<" + reader.Name+ ">");
                break;
            //if the node is text
            case XmlNodeType.Text:
                Console.WriteLine(reader.Value);
                break;
            //if the node is ending element
            case XmlNodeType.EndElement:
                Console.WriteLine("</" + reader.Name + ">");
                break;
            }
        }
        reader.Close();
    }
    static void Main(string[] args)
    {
        CreateSimpleDocument();
        CreateDocument();
        CreateDocumentUsingElement();
        CreateDocumentUsingSingleVariableInSameLevel();
        createAttributesinXMLDoc();
        XMLTextWriterClass();
        XMLTextReaderClass();
    }
  }
}
```

Output:

```
****SIMPLE XML FILE IS CREATED by XML DOM TECHNOLOGY CLASSES****

****XML FILE1 IS CREATED by XML DOM TECHNOLOGY CLASSES****
```

```
****XML FILE2 IS CREATED by XML DOM TECHNOLOGY CLASSES****


****XML FILE3 IS CREATED by XML DOM TECHNOLOGY CLASSES****


****XML FILE WITH ATTRIBUTES IS CREATED by XML DOM TECHNOLOGY CLASSES****

XmlTextWriter class is used
<Employees>
<Employee>
<Medium>
Telugu
</Medium>
<Salary>
20000$
</Salary>
</Employee>
</Employees>
Press any key to continue . . .
```

XML file contents:

XMLDemo1.xml:

```
<EmployeeDetails>TeluguLanguage</EmployeeDetails>
```

XMLDemo2.xml,XmlDemo3.xml.XmlDemo4.xml: All these three files holds same outputs.

```
<EmployeeDetails>
 <Employee1>
  <Name>Lenaprasanna</Name>
  <ID>1001</ID>
  <Language>Telugu</Language>
  </Employee1>
 <Employee2>
  <Name>Kishore</Name>
  <ID>1002</ID>
  <Language>Tamil</Language>
  </Employee2>
  </EmployeeDetails>
```

XMLDemoAttributes1.xml:

```
<Employees>
  <Employee>
     <Medium>Telugu</Medium>
     <Salary>20000$</Salary>
  </Employee>
</Employees>
```

XMLValidatingReader:

- The XmlValidatingReader class is used to validate an XML file against a DTD or a schema (either XDR or XSD).
- This class is used along with the XmlTextReader class and provides the same access to the document contents.
- The properties and methods of these two classes are essentially identical.
- The differences between them lie in two properties of the XmlValidatingReader class that are related to validation.

Sample program that Explains XMLValidatingReader usage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;
using System.Xml.Schema;

namespace XMLValidatingReaderClass
{
   class Program
   {
      private Boolean m_success = true;

      public Program()
      {
         //Validate the document using an external XSD schema.  Validation should fail.
         Validate(@"D:\notValidXSD.xml");

         //Validate the document using an inline XSD. Validation should succeed.
         Validate(@"D:\inlineXSD.xml");
      }
      private void Validate(String filename)
      {
         m_success = true;
         Console.WriteLine("\n******");
         Console.WriteLine("Validating XML file " + filename.ToString());
         XmlTextReader txtreader = new XmlTextReader(filename);
         XmlValidatingReader reader = new XmlValidatingReader(txtreader);

         // Set the validation event handler
         reader.ValidationEventHandler += new ValidationEventHandler(ValidationCallBack);
```

```
        // Read XML data
        while (reader.Read()) { }
        Console.WriteLine("Validation finished. Validation {0}", (m_success == true ? "successful!" :
"failed."));

        //Close the reader.
        reader.Close();
    }


    //Display the validation error.
    private void ValidationCallBack(object sender, ValidationEventArgs args)
    {
        m_success = false;
        Console.WriteLine("\n\tValidation error: " + args.Message);
    }

    static void Main(string[] args)
    {
        Program obj = new Program();
    }
  }
}
```

Output:

```
******
Validating XML file D:\notValidXSD.xml

        Validation error: The element 'book' in namespace 'urn:bookstore-schema'
 has invalid child element 'author' in namespace 'urn:bookstore-schema'. List of
 possible elements expected: 'urn:bookstore-schema:title'.
Validation finished. Validation failed.

******
Validating XML file D:\inlineXSD.xml
Validation finished. Validation successful!
Press any key to continue . . .
```

19.Delegates and events

Delegates:

- It is a type which act as a reference to methods.
- It is similar to functional pointer available in the C and C++ programming languages.
- Delegates are used to call/invoke the methods in the run time. (ie.) passing methods as a parameter in run time.
- Delegates are defined in System.Delegate
- Delegate is declared as similar to the normal method but it is prefixed with the keyword "delegate".
- Once a delegate is created, then the same delegate can be used to call any number of methods which has the same signature.
- Delegates can be declared and defined in any place within the namespace.
- Delegate should be declared with the same return type as of the method that is about to call.

Syntax for delegates:

delegate returntype delegateName(parameters);

Note: *A pointer that refers a function is called a functional pointer.*

Example :

Consider a method called void display() in the class "DelegateSimple". To access the method we need to create object to that class.

DelegateSimple d=new DelegateSimple(display); ***//where display is the method name which is passed as a parameter.***

Note: *it is not mandatory to use new operator explicitly on delegate objects. We can also use DelegateSimple d=display;*

Types of Delegates:

There are two types of delegates:

- **Single cast** –If a single delegate instance is referenced to a single method, then it is called as a Single cast delegate.
- **Multi cast** – If a single delegate instance is referenced to multiple methods, , then it is called as a Multi cast delegate.

Note:

Return type for multicast delegates should be **void** always, for better practice. If any other return type is given, the return value of the last method that is executed will be retained.

Syntax for events:

```
event DelegateName instance;
```

Anonymous methods:

- It is a set of codes referenced by delegates.
- They will not have any explicit method name.

Syntax for anonymous methods:

```
DelegateName instance=delegate(parameter)
                    {
                        .....
                        .....
                    };
```

Lambda expression:

Similar to anonymous methods, lambda expressions are also used. Anonymous methods consumes more space due to lines of code and hence we move on to lamba expression. *This topic is important.*

Types of Lamba expressions:

There are two types of Lambda expression. Namely:

- Expression Lambda
- Statement Lambda

Syntax for lambda expression:

1.**Expression Lambda:**

```
DelegateName instance = (Lambda Variables) => expression evaluation;
```

2.**Statement Lambda:**

```
DelegateName instance  = (Lambda variables) => {
                                                .....
                                                .....//statements;
                                                .....
                                                 }
```

- ***Note:*** *Expression Lambda will have only one expression to the right side of Lambda.*
  - *In C#, the symbol => is called as "Lambda".*
- *Variables declared in Expression lambda need not be declared with the datatype. Datatype of the variable will be automatically taken.*
- *If there are many variables, each variable is separated by comma.*

- *A void return type of delegate cannot call the Expression lambda but it can call the statement lambda*
- *If variables are declared outside the lambda expression, then those variables are called as "Outer Variables".*

Example for single cast delegate:

```
namespace DelegateDemo
{
        class Program
        {
                Delegate void DelegateSimple(string x);

                static void Main(String[] args)
                {
                        DelegateSimple objDelegate = new DelegateSimple(display);
                        ……
                        ……
                        objDelegate(); //This will call the display method at runtime.
                }
                static void display(string msg)
                {
                        Console.WriteLine("String passed:"+msg);
                }
        }
}
```

Where DelegateSimple is a delegate that can be used to refer all the method which has return type void and having string as the only parameter according to this example.

Example of single cast and multicast delegates:

<Program.cs>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DelegatesSimpleDemo
{

   class Program
   {
      static void Addition(int number1, int number2)
      {
         Console.WriteLine("\nAddition() method is called");
```

```
    Console.WriteLine("{0} + {1} is:{2}",number1,number2,number1+number2);
}

static void Multiplication(int number1, int number2)
{
    Console.WriteLine("\nMultiplication() method is called");
    Console.WriteLine("{0} * {1} is:{2}", number1, number2, number1 * number2);
}

static void display(string x)
{
    Console.WriteLine("\nstring is: "+x);
}

static void Main(string[] args)
{

   // simpleDelegate objSimpleDelegate = Addition;

   //or

    simpleDelegate objDelegate1 = new simpleDelegate(Addition);

    Console.WriteLine("\n****Calling Addition() using objDelegate1****\n");
   objDelegate1(10, 20); //addition method will be called.


    //We can also use objDelegate1 again to call the multiplication method or create a fresh object
for delegate.

    Console.WriteLine("\n****Calling Multiplication() using objDelegate1****\n");
    objDelegate1 = Multiplication;
    objDelegate1(10, 20); //single cast delegate
    //or
    //objDelegate1 = new SimpleDelage(Multipication); // This is similar to the above statement.
Addition method is overrided by th Multiplication Method. Now call using this object will call
Multiplication Method only and not the Addition method.


    simpleDelegate objDelegate2 = new simpleDelegate(Multiplication);

    Console.WriteLine("\n****Calling Multiplication() using objDelegate2****\n");
    objDelegate2(10, 20); //Singlecast delegate

    //objDelegate1 = display(); //Assignment of display method is NOT VALID. eventhough
returntype , the type of parameters and the number of parameters of the delegate and display() are
different.
```

```
        simpleDelegate objDelegate3 = new simpleDelegate(Addition);
      //If we also need addition method, we need to use += rather than = in the above statement.
        //(ie.) objDelegate3 += new SimpleDelage(Multipication);


        objDelegate3 += new simpleDelegate(Multiplication); //Multicast delegate
        //or
        //objDelegate3=Multiplication; // adds Multiplication() to the obj which already has Addition()
        Console.WriteLine("\n****Calling both addition() and Multiplication() using
objDelegate3****\n");

        objDelegate3(30, 50); //calls both the methods at the same time.
      }
   }
}
```

<codeFile.cs>

//Steps to follow: *right click projectName in Visual studio 2010, select Add and then select New Item.. and then select CodeFile.*

```
delegate void simpleDelegate(int a,int b);
```

Output:

```
****Calling Addition() using objDelegate1****


Addition() method is called
10 + 20 is:30

****Calling Multiplication() using objDelegate1****


Multiplication() method is called
10 * 20 is:200

****Calling Multiplication() using objDelegate2****


Multiplication() method is called
10 * 20 is:200

****Calling both addition() and Multiplication() using objDelegate3****


Addition() method is called
30 + 50 is:80

Multiplication() method is called
```

```
30 * 50 is:1500

****Removing Addition() and calling only the multiplication() using objDelegate3
****


Multiplication() method is called
30 * 50 is:1500
Press any key to continue . . .
```

Example for multicast Delegates:

<Program.cs>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DelegatesSimpleDemo
{

   class Program
   {

      static void Hello(string s)
      {
         System.Console.WriteLine(" Hello {0}!", s);
      }

      static void Goodbye(string s)
      {
         System.Console.WriteLine("  Goodbye, {0}!", s);
      }

      static void Welcome(string s)
      {
         System.Console.WriteLine("  Goodbye, {0}!", s);
      }

      static void Main(string[] args)
      {
         Console.WriteLine("\n****MULTICAST DELEGATES******\n");
         myDelegate obj = new myDelegate(Hello);
         Console.WriteLine("\n****Calling Hello() alone******\n");
         obj("lenaprasanna");

         obj += Welcome; //or obj=new myDelegate(Welcome);
         Console.WriteLine("\n****Calling Hello() and Welcome() alone******\n");
         obj("lenaprasanna");
```

```
        obj += Goodbye;//obj call
        Console.WriteLine("\n****Calling Hello(),Welcome() and Goodbye()******\n");
        obj("lenaprasanna");

        obj -= Goodbye;
        Console.WriteLine("\n****Calling Hello(),Welcome() after removing
Goodbye(),Welcome()******\n");
        obj("lenaprasanna");

        obj -= Welcome;
        Console.WriteLine("\n****Calling Hello() after removing Goodbye(),Welcome()******\n");
        obj("lenaprasanna");
        obj -= Hello;
       // Console.WriteLine("\n****Calling after removing Goodbye(),Welcome(),Hello() ******\n");
        //obj("lenaprasanna"); // This is INVALID as we removed all the methods we included to it and
now it has no method to call.

       }
   }
}
```

< codeFile.cs>

```
delegate void myDelegate(string a);
```

Output:

```
****MULTICAST DELEGATES******


****Calling Hello() alone******

 Hello lenaprasanna!

****Calling Hello() and Welcome() alone******

 Hello lenaprasanna!
  Goodbye, lenaprasanna!

****Calling Hello(),Welcome() and Goodbye()******

 Hello lenaprasanna!
  Goodbye, lenaprasanna!
  Goodbye, lenaprasanna!

****Calling Hello(),Welcome() after removing Goodbye(),Welcome()******

 Hello lenaprasanna!
  Goodbye, lenaprasanna!
```

```
****Calling Hello() after removing Goodbye(),Welcome()******

 Hello lenaprasanna!
Press any key to continue . . .
```

Example for Anonymous methods, Expression lambda and Statement lambda:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AnonymousMethods
{

   class Program
   {

      public delegate int DelegateMethod(int x); // prototype declaration

      static int display(int y)
      {
         Console.WriteLine("\nDisplay() method is invoked");
         return y;
      }


      static void Main(string[] args)
      {
         DelegateMethod del = new DelegateMethod(display); // del has display method

         Console.WriteLine("\n****Calling delegate method****\n");
         int disvalue = del(10); //calls display method

         Console.WriteLine("\nDisplay() method value is " + disvalue);

         Console.WriteLine("\n****Anonymous method****\n");
         del = delegate(int n)  //anonymous method - A method with no name. Here, del is overrided
with the anonymous method.
         {
            int sum = 0;
            for (int i = 0; i < n; i++)
            {
               sum = sum + i;
               i++;

            }
            return sum;
         };
```

```
int anonvalue = del(22); // call for anonymous method
Console.WriteLine("\nAnonymous method result is: " + anonvalue);

Console.WriteLine("\n***Expression Lambda***\n");
del = (count) => count * 2; // Overriding del with expression Lambda. Note: Count is not
declared as int. (ie.) Variables of expression lambda need not be declared.
//The above statement doubles the value that is passed to del.

int exLambValue = del(5); // it takes count value to be 5 and it multiplies 5 with 2 and the result
will be 10, that is assigned to exLambValue

Console.WriteLine("\nExpression Lambda value is " + exLambValue);

int sumOfThree = 0;
del = (number) =>
{
   Console.WriteLine("\n****Statement Lambda*****\n");
   for (int i = 0; i < number; i++)
   {
      if (i % 3 == 0)
      {
         sumOfThree = sumOfThree + i;
      }

   }
   return sumOfThree;
};
int statval = del(20);
Console.WriteLine("the value of statement lambda is " + statval);
```

//Function delegate

//In general, if a function has n parameters inside Func<>, then first n-1 parameters will be inputs and nth parameter will be output.

```
Console.WriteLine("\n******Function Delegates******\n");
Func<string, string> convertMethod = uppercaseString;
Console.WriteLine(convertMethod("telugu"));

//Anonymous delegate
Console.WriteLine("\n******Anonymous Delegates******\n");
Func<string,string> convert=delegate(string s)
                    {
                       return s.ToUpper();
                    };
Console.WriteLine(convert("telugu"));
```

```
        //replacing  anonymous delegate using lambda
        Console.WriteLine("\n******Lambda Expression******\n");
         Func<string,string> convertUsingLambda = s=>s.ToUpper();
          Console.WriteLine(convertUsingLambda("telugu"));
     }

   }
}
```

Output:

```
****Calling delegate method****

Display() method is invoked

Display() method value is 10

****Anonymous method****

Anonymous method result is: 110

***Expression Lambda***

Expression Lambda value is 10

****Statement Lambda*****

the value of statement lambda is 63

******Function Delegates******

TELUGU

******Anonymous Delegates******

TELUGU

******Lambda Expression******

TELUGU
Press any key to continue . . .
```

Example for Delegates – little complex:

// A simple delegate example.

```
using System;
```

// Declare a delegate.

```
delegate string strMod(string str);

public class DelegateTest
 {
  // Replaces spaces with hyphens.
  static string replaceSpaces(string a)
  {
    Console.WriteLine("Replaces spaces with hyphens.");
    return a.Replace(' ', '-');
  }

  // Remove spaces.
  static string removeSpaces(string a)
 {
    string temp = "";
    int i;

    Console.WriteLine("Removing spaces.");
    for(i=0; i < a.Length; i++)
      if(a[i] != ' ') temp += a[i];

    return temp;
  }

  // Reverse a string.
  static string reverse(string a) {
    string temp = "";
    int i, j;

    Console.WriteLine("Reversing string.");
    for(j=0, i=a.Length-1; i >= 0; i--, j++)
      temp += a[i];

    return temp;
  }

  public static void Main()
 {
    // Construct a delegate.
    strMod strOp = new strMod(replaceSpaces);
    string str;

    // Call methods through the delegate.
    str = strOp("Telugu Language");
    Console.WriteLine("Resulting string: " + str);
    Console.WriteLine();
```

```
    strOp = new strMod(removeSpaces);
    str = strOp("Telugu Language");
    Console.WriteLine("Resulting string: " + str);
    Console.WriteLine();

    strOp = new strMod(reverse);
    str = strOp("Telugu Language");
    Console.WriteLine("Resulting string: " + str);
  }
}
```

Output:

```
Resulting string: Telugu-Language
Resulting string: TeluguLanguage
Resulting string: egaugnaL uguleT
```

Arrays of Delegates:

We can also create array of delegates. Below is the self explanatory program which explains the entire concept of arrays of delegates.

Sample example:

```
using System;

public delegate void Task();

public class Starter {
    public static void Main() {
        Task[] tasks = { MethodA, MethodB, MethodC };
        tasks[0]();
        tasks[1]();
        tasks[2]();
    }

    public static void MethodA() {
        Console.WriteLine("Doing TaskA");
    }

    public static void MethodB() {
        Console.WriteLine("Doing TaskB");
    }

    public static void MethodC() {
        Console.WriteLine("Doing TaskC");
    }
}
```

Output:

```
Doing TaskA
Doing TaskB
Doing TaskC
Press any key to continue . . .
```

Events in C#:

- Events and delegates are somewhat analogous. An event is something *in the model*.
- A button is a thingthat can inform you when it is clicked, so the Button class has a "Click" event.
- The delegate that actually does the informing is the implementation detail of the event.

Sample example illustrating events in C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EventsDemo
{

  //Events are the delegate applications always.
  delegate void MyEventClass();

  class MyEvent
  {

    public event MyEventClass someevent; //someevent is the instance name.

    public void onSomeEvent() //An Event handler
    {
      if (someevent != null)
      {
        someevent(); //calling the delegate.
      }
    }
```

```
    }
    class X
    {
        public void Xhandler()
        {
            Console.WriteLine("Xhandler() method is invoked!!!");
        }
    }

    class Y
    {
        public void Yhandler()
        {
            Console.WriteLine("Yhandler() method is invoked!!!");
        }
    }
}
```

*******Class that invokes Events***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace EventsDemo
{
    class Program
    {
        static void Handler()
        {
            Console.WriteLine("Handler() method is called!!");
        }
        static void Main(string[] args)
        {
            MyEvent objMyEvent = new MyEvent();
```

```
        X objX = new X();
        Y objY = new Y();

        objMyEvent.someevent += new MyEventClass(objX.Xhandler); //this is an user defined event. it can be used to call any number of handlers.
        objMyEvent.someevent += new MyEventClass(objY.Yhandler);
        objMyEvent.someevent += new MyEventClass(Handler);
        objMyEvent.onSomeEvent();
        Console.WriteLine("\n***Removing Xhandler from someEvent***\n");
        objMyEvent.someevent -= new MyEventClass(objX.Xhandler);
        objMyEvent.onSomeEvent();
    }
  }
}
```

Output:

```
Xhandler() method is invoked!!!
Yhandler() method is invoked!!!
Handler() method is called!!

***Removing Xhandler from someEvent***

Yhandler() method is invoked!!!
Handler() method is called!!
Press any key to continue . . .
```

20.Reflection

Note:

- Collecting information about types in runtime.
- Reflection (Information about information)
- All classes are inherited from **System.Reflection**
- Creates new object in runtime and collects information about all modules or types such as classes, interface, structs etc in assembly (executable file (.exe) and class library (.dll)) files.
- Reflection gathers information about our c# source code files. Its a collection of fields, methods, properties, indexes , events, delegates etc. It happens during runtime.
- It is for the metadata purpose.To improve the performance of the application, the project group should know and hence reflection is carried out.

AppDomain : collection of assembly files waiting for execution.

Assembly files: it contains a number of modules.

Module: A module contains a number of types.

Type: A type can have many fields & methods(members), constructors, parameters, properties, Indexers

Basic Syntaxes used under reflection:

AppDomain Ad=Thread.Current; *//Getting the thread instance*

Assembly asm= assembly.LoadFile();
Assembly asm= Assembly.LoadFrom();

Modules[] mod=asm.GetModules(); *// Each and every modules will be retrieved. So, it is of type module array. For example mod[0] retrieves 1st module.*

Types[] types=mod.GetTypes(); *//collects all the class information*.

Type obj= type[0]; *//to collect a first type information.*

Type obj= typeof(classname); *//Collect the information about the types in a particular class*

Inorder to collect specific informations, we can use:

- MembersInfo[] *//collects info about members*
- MethodInfo[] *//collects info about methods*
- FieldInfo[] *//collects info about fields*
- ConstructorInfo[] *//collects info about constructors*

- PropertiesInfo[] *//collects info about properties*
- parametersInfo[] *//collects info about parameters in a particular methods.*
- and so on..

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection; //This should be included for reflection

namespace ReflectionDemo
{
   class Program
   {
      public string _name;
      public int number;

      public Program()
      {
         Name = string.Empty;
         number = 0;
      }

      public Program(string s, int a)
      {
         Name = s;
         number = a;
      }

      public string Name
      {
         get
         {
            return _name;
         }
         set
         {
            _name=value;
```

```
    }
}

public static long addition(int number1, int number2)
{
    Console.WriteLine("addition method is called");
    return number1 + number2;
}
public static long multiplication(int number1, int number2)
{
    Console.WriteLine("multiplication method is called");
    return number1 * number2;
}
public static void display(string x)
{
    Console.WriteLine("display message is" + x);
}
public static void Hello(string s)
{
    System.Console.WriteLine(" Hello {0}!", s);
}

public static void Goodbye(string s)
{
    System.Console.WriteLine("  Goodbye, {0}!", s);
}

public static void Welcome(string s)
{
    System.Console.WriteLine("  Goodbye, {0}!", s);
}
void display()
{
    Console.WriteLine("Name={0}\nNumber={1}", Name, number);
}
~Program()
{
    Console.WriteLine("Destructor is called");

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

```
{

   Type t = typeof(Program);
   Console.WriteLine("\n***Collecting Field Info***\n");
   FieldInfo[] fieldInfo = t.GetFields();

   foreach (FieldInfo fi in fieldInfo)
   {
      Console.WriteLine("Field Name={0}", fi.Name);
      Console.WriteLine("Field type={0}", fi.FieldType);
      Console.WriteLine("Field Module={0}", fi.Module);
      Console.WriteLine("\n");

   }

     Console.WriteLine("\n***Collecting Properties Info***\n");
     PropertyInfo[] propInfo = t.GetProperties();

     foreach (PropertyInfo fi in propInfo)
    {
      Console.WriteLine("Property Name={0}", fi.Name);
      Console.WriteLine("Property type={0}", fi.PropertyType);
       Console.WriteLine("Property Attributes={0}", fi.Attributes);
      Console.WriteLine("Property Module={0}", fi.Module);
      Console.WriteLine("\n");
    }

   Console.WriteLine("\n***Collecting Member Info***\n");

   MemberInfo[] memberInfo = t.GetMembers(); //retrieves all info about methods.
   foreach (MemberInfo m in memberInfo)  //for individual methods m, we are about to extract info.
   {
     //member Info retrieves all the details about properties, methods and constructors.
   //Note: Member info cannot have parameter Info
      Console.WriteLine("Member Name=" + m.Name); //Method name
      Console.WriteLine("member Module=" + m.Module);
      Console.WriteLine("Member type=" + m.MemberType);
      Console.WriteLine("\n");
   }

  Console.WriteLine("\n***Collecting Method Info***\n");
```

```
        MethodInfo[] methInfo =t.GetMethods();
        foreach (MethodInfo mi in methInfo)
        {


          Console.WriteLine("Method Name=" + mi.Name);
          Console.WriteLine("Return type=" + mi.ReturnType.Name);
          Console.WriteLine("\n***Collecting parameters Info of {0}***\n", mi.Name);

          ParameterInfo[] pi = mi.GetParameters();

          foreach (ParameterInfo p in pi) // collects parameters list in the method that mi currently
points to.
          {
            Console.WriteLine("parameter Name=" + p.Name);
            Console.WriteLine("Return type=" + p.ParameterType);
          }
        }
         Console.WriteLine("\n***Collecting Constructors Info***\n");
         ConstructorInfo[] ci = t.GetConstructors();

         foreach (ConstructorInfo c in ci)
         {
          //Note: constructor name will be .ctor always
           Console.WriteLine("Constructor Name=" + c.Name);
           Console.WriteLine("\n***Collecting parameters Info of {0}***\n", c.Name);

           ParameterInfo[] pi = c.GetParameters();

           foreach (ParameterInfo p in pi)
           {
             Console.WriteLine("parameter Name=" + p.Name);
             Console.WriteLine("Return type=" + p.ParameterType);
             Console.WriteLine("\n");
           }
         }
      }
   }
}
```

Output:

```
***Collecting Field Info***

Field Name=_name
Field type=System.String
Field Module=ReflectionDemo.exe


Field Name=number
Field type=System.Int32
Field Module=ReflectionDemo.exe



***Collecting Properties Info***

Property Name=Name
Property type=System.String
Property Attributes=None
Property Module=ReflectionDemo.exe



***Collecting Member Info***

Member Name=get_Name
member Module=ReflectionDemo.exe
Member type=Method


Member Name=set_Name
member Module=ReflectionDemo.exe
Member type=Method


Member Name=addition
member Module=ReflectionDemo.exe
Member type=Method


Member Name=multiplication
member Module=ReflectionDemo.exe
Member type=Method


Member Name=display
member Module=ReflectionDemo.exe
Member type=Method
```

```
Member Name=Hello
member Module=ReflectionDemo.exe
Member type=Method


Member Name=Goodbye
member Module=ReflectionDemo.exe
Member type=Method


Member Name=Welcome
member Module=ReflectionDemo.exe
Member type=Method


Member Name=ToString
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=Equals
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=GetHashCode
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=GetType
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=.ctor
member Module=ReflectionDemo.exe
Member type=Constructor


Member Name=.ctor
member Module=ReflectionDemo.exe
Member type=Constructor


Member Name=Name
member Module=ReflectionDemo.exe
Member type=Property
```

```
Member Name=_name
member Module=ReflectionDemo.exe
Member type=Field


Member Name=number
member Module=ReflectionDemo.exe
Member type=Field



***Collecting Method Info***

Method Name=get_Name
Return type=String

***Collecting parameters Info of get_Name***

Method Name=set_Name
Return type=Void

***Collecting parameters Info of set_Name***

parameter Name=value
Return type=System.String
Method Name=addition
Return type=Int64

***Collecting parameters Info of addition***

parameter Name=number1
Return type=System.Int32
parameter Name=number2
Return type=System.Int32
Method Name=multiplication
Return type=Int64

***Collecting parameters Info of multiplication***

parameter Name=number1
Return type=System.Int32
parameter Name=number2
Return type=System.Int32
Method Name=display
Return type=Void

***Collecting parameters Info of display***

parameter Name=x
```

```
Return type=System.String
Method Name=Hello
Return type=Void

***Collecting parameters Info of Hello***

parameter Name=s
Return type=System.String
Method Name=Goodbye
Return type=Void

***Collecting parameters Info of Goodbye***

parameter Name=s
Return type=System.String
Method Name=Welcome
Return type=Void

***Collecting parameters Info of Welcome***

parameter Name=s
Return type=System.String
Method Name=ToString
Return type=String

***Collecting parameters Info of ToString***

Method Name=Equals
Return type=Boolean

***Collecting parameters Info of Equals***

parameter Name=obj
Return type=System.Object
Method Name=GetHashCode
Return type=Int32

***Collecting parameters Info of GetHashCode***

Method Name=GetType
Return type=Type

***Collecting parameters Info of GetType***


***Collecting Constructors Info***

Constructor Name=.ctor

***Collecting parameters Info of .ctor***
```

```
Constructor Name=.ctor

***Collecting parameters Info of .ctor***

parameter Name=s
Return type=System.String


parameter Name=a
Return type=System.Int32


Press any key to continue . . .
```

Collecting info in Assembly level:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection; //This should be included for reflection

namespace ReflectionDemo
{
   class Program
   {

      public string _name;
      public int number;

      public Program()
      {
         Name = string.Empty;
         number = 0;
      }

      public Program(string s, int a)
      {
         Name = s;
         number = a;
      }
```

```
public string Name
{
   get
   {
      return _name;
   }

   set
   {
      _name = value;
   }
}

public static long addition(int number1, int number2)
{
   Console.WriteLine("addition method is called");
   return number1 + number2;
}
public static long multiplication(int number1, int number2)
{
   Console.WriteLine("multiplication method is called");
   return number1 * number2;
}
public static void display(string x)
{
   Console.WriteLine("display message is" + x);
}
public static void Hello(string s)
{
   System.Console.WriteLine(" Hello {0}!", s);
}

public static void Goodbye(string s)
{
   System.Console.WriteLine("  Goodbye, {0}!", s);
}

public static void Welcome(string s)
{
   System.Console.WriteLine("  Goodbye, {0}!", s);
}
```

```
void display()
{
   Console.WriteLine("Name={0}\nNumber={1}", Name, number);
}
~Program()
{
   Console.WriteLine("Destructor is called");

}
static void Main(string[] args)
{

   Assembly asm = Assembly.LoadFrom("ReflectionDemo.exe");
   Type[] types = asm.GetTypes();
   foreach (Type t in types)
   {
      Console.WriteLine("Assembly Name="+t.Name);
      MemberInfo[] memberInfo = t.GetMembers();
      Console.WriteLine("\n****Getting Members of {0}****\n", t.Name);

      foreach (MemberInfo m in memberInfo)  //for individual methods m, we are about to extract
info.
      {
         //member Info retrieves all the details about properties, methods and constructors.
         //Note: Member info cannot have parameter Info
         Console.WriteLine("Member Name=" + m.Name); //Method name
         Console.WriteLine("member Module=" + m.Module);
         Console.WriteLine("Member type=" + m.MemberType);
         Console.WriteLine("\n");
      }

      Console.WriteLine("\n****Printing Fields Info***\n");
      FieldInfo[] fieldInfo = t.GetFields();
      foreach (FieldInfo fi in fieldInfo)
      {
         Console.WriteLine("Field Name={0}", fi.Name);
         Console.WriteLine("Field type={0}", fi.FieldType);
         Console.WriteLine("Field Module={0}", fi.Module);
         Console.WriteLine("\n");

      }
```

```
            Console.WriteLine("\n***Collecting Properties Info***\n");
            PropertyInfo[] propInfo = t.GetProperties();

            foreach (PropertyInfo fi in propInfo)
            {
               Console.WriteLine("Property Name={0}", fi.Name);
               Console.WriteLine("Property type={0}", fi.PropertyType);
               Console.WriteLine("Property Attributes={0}", fi.Attributes);
               Console.WriteLine("Property Module={0}", fi.Module);
               Console.WriteLine("\n");
            }
           }
      }
   }
}
```

Output:

```
Assembly Name=Program

****Getting Members of Program****

Member Name=get_Name
member Module=ReflectionDemo.exe
Member type=Method


Member Name=set_Name
member Module=ReflectionDemo.exe
Member type=Method


Member Name=addition
member Module=ReflectionDemo.exe
Member type=Method


Member Name=multiplication
member Module=ReflectionDemo.exe
Member type=Method


Member Name=display
member Module=ReflectionDemo.exe
Member type=Method
```

```
Member Name=Hello
member Module=ReflectionDemo.exe
Member type=Method


Member Name=Goodbye
member Module=ReflectionDemo.exe
Member type=Method


Member Name=Welcome
member Module=ReflectionDemo.exe
Member type=Method


Member Name=ToString
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=Equals
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=GetHashCode
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=GetType
member Module=CommonLanguageRuntimeLibrary
Member type=Method


Member Name=.ctor
member Module=ReflectionDemo.exe
Member type=Constructor


Member Name=.ctor
member Module=ReflectionDemo.exe
Member type=Constructor


Member Name=Name
member Module=ReflectionDemo.exe
Member type=Property
```

```
Member Name=_name
member Module=ReflectionDemo.exe
Member type=Field


Member Name=number
member Module=ReflectionDemo.exe
Member type=Field



****Printing Fields Info***

Field Name=_name
Field type=System.String
Field Module=ReflectionDemo.exe


Field Name=number
Field type=System.Int32
Field Module=ReflectionDemo.exe



***Collecting Properties Info***

Property Name=Name
Property type=System.String
Property Attributes=None
Property Module=ReflectionDemo.exe


Press any key to continue . . .
```

<u>Collecting information in appDomain level for main thread:</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Reflection;// this should be used for reflection
using System.Threading; //this should be used for threading


namespace ReflectionDemo2
{
   class Person
```

```
{
   public int Id;

   public string Name;

   public string Address
   {
      get;
      set;

   }

   public string city
   {
      get;
      set;

   }

   public Person()
   {
      Address = "Nandula peta,Tenali";
      city = "Tenali";

   }

   public Person(string Address, string city)
   {
      this.Address = Address;
      this.city = city;

   }


   public void getDetails()
   {

      Console.WriteLine("Enter person id");
      Id = Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("Enter person name");
      Name = Console.ReadLine();
   }
   public void getDetails(int x, string y)
   {

      Id = x;
      Name = y;
```

```
    }


}
class Program
{

    static void Main(string[] args)
    {



        AppDomain ads = Thread.GetDomain();

        Assembly[] assemblies = ads.GetAssemblies();


        foreach (Assembly assembly in assemblies)
        {

            Type[] types = assembly.GetTypes();

            foreach (Type type in types)
            {

                //type level information
                Console.WriteLine(type.Name);
                Console.WriteLine("\n***Collecting Member Info***\n");
                MemberInfo[] memInfo = type.GetMembers();
                for (int i = 0; i < memInfo.Length; i++)
                {
                    Console.WriteLine("\n\nMember name is:" + memInfo[i].Name);

                    Console.WriteLine("\n\nMember module is:" + memInfo[i].Module);
                    Console.WriteLine("\n\nMember type is:" + memInfo[i].MemberType);

                }

                foreach (MemberInfo mi in memInfo)
                {

                    Console.WriteLine("Member name is:" + mi.Name);

                }
```

```
            //fields information in a type
            FieldInfo[] fieldInf = type.GetFields();
            Console.WriteLine("\n***Collecting Field Info***\n");
            foreach (FieldInfo fi in fieldInf)
            {
               Console.WriteLine(fi.Name);

            }

            //properties information in a type
            Console.WriteLine("\n***Collecting Properties Info***\n");
            PropertyInfo[] propInfo = type.GetProperties();
            foreach (PropertyInfo pi in propInfo)
            {
               Console.WriteLine("propery name is:" + pi.Name);
            }

            Console.WriteLine("\n***Collecting Method Info***\n");
            MethodInfo[] methInfo = type.GetMethods();
            foreach (MethodInfo mi in methInfo)
            {
               //method level information in a type

               Console.WriteLine("method name is:" + mi.Name);
               ParameterInfo[] paramInfo = mi.GetParameters();
               foreach (ParameterInfo pi in paramInfo)
               {
                  Console.WriteLine("parameter name:" + pi.Name + " its type is:" + pi.ParameterType);
               }
            }

         }
      }
   }
}
```

Output:

```
Member type is:Method

Member name is:ToString
```

```
Member module is:CommonLanguageRuntimeLibrary


Member type is:Method


Member name is:GetType


Member module is:CommonLanguageRuntimeLibrary


Member type is:Method
Member name is:Equals
Member name is:GetHashCode
Member name is:ToString
Member name is:GetType

***Collecting Field Info***


***Collecting Properties Info***


***Collecting Method Info***

method name is:Equals
parameter name:objits type is:System.Object
method name is:GetHashCode
method name is:ToString
method name is:GetType
Person

***Collecting Member Info***



Member name is:get_Address


Member module is:ReflectionDemo2.exe


Member type is:Method


Member name is:set_Address


Member module is:ReflectionDemo2.exe


Member type is:Method


Member name is:get_city
```

```
Member module is:ReflectionDemo2.exe

Member type is:Method

Member name is:set_city

Member module is:ReflectionDemo2.exe

Member type is:Method

Member name is:getDetails

Member module is:ReflectionDemo2.exe

Member type is:Method

Member name is:getDetails

Member module is:ReflectionDemo2.exe

Member type is:Method

Member name is:ToString

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:Equals

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:GetHashCode

Member module is:CommonLanguageRuntimeLibrary
```

```
Member type is:Method

Member name is:GetType

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:.ctor

Member module is:ReflectionDemo2.exe

Member type is:Constructor

Member name is:.ctor

Member module is:ReflectionDemo2.exe

Member type is:Constructor

Member name is:Address

Member module is:ReflectionDemo2.exe

Member type is:Property

Member name is:city

Member module is:ReflectionDemo2.exe

Member type is:Property

Member name is:Id

Member module is:ReflectionDemo2.exe

Member type is:Field
```

```
Member name is:Name


Member module is:ReflectionDemo2.exe


Member type is:Field
Member name is:get_Address
Member name is:set_Address
Member name is:get_city
Member name is:set_city
Member name is:getDetails
Member name is:getDetails
Member name is:ToString
Member name is:Equals
Member name is:GetHashCode
Member name is:GetType
Member name is:.ctor
Member name is:.ctor
Member name is:Address
Member name is:city
Member name is:Id
Member name is:Name

***Collecting Field Info***

Id
Name

***Collecting Properties Info***

propery name is:Address
propery name is:city

***Collecting Method Info***

method name is:get_Address
method name is:set_Address
parameter name:valueits type is:System.String
method name is:get_city
method name is:set_city
parameter name:valueits type is:System.String
method name is:getDetails
method name is:getDetails
parameter name:xits type is:System.Int32
parameter name:yits type is:System.String
method name is:ToString
method name is:Equals
parameter name:objits type is:System.Object
method name is:GetHashCode
method name is:GetType
Program

***Collecting Member Info***
```

```
Member name is:ToString

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:Equals

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:GetHashCode

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:GetType

Member module is:CommonLanguageRuntimeLibrary

Member type is:Method

Member name is:.ctor

Member module is:ReflectionDemo2.exe

Member type is:Constructor
Member name is:ToString
Member name is:Equals
Member name is:GetHashCode
Member name is:GetType
Member name is:.ctor
***Collecting Field Info***

***Collecting Properties Info***

***Collecting Method Info***
```

```
method name is:ToString
method name is:Equals
parameter name:objits type is:System.Object
method name is:GetHashCode
method name is:GetType
Press any key to continue . . .
```

21.Serialization

Note:

- Serialization is Converting object types into bytes.
- DeSerialization is from bytes to object types.
- It can be done for binary files, xml files, custom files and SOAP (Simple ObjectAccess Protocol) serialization
- SOAP Serialization -> converts soap objects into bytes
- *using System.Runtime.Serialization.Formatters.Binary;* - it is used for binary serialization
- using *System.*Runtime.Serialization.Formatters.Soap.dll – it is used for SOAP serialization

- ***IMPORTANT NOTE:*** class should be declared [Serializable] inorder to serialize its objects

Types of serializations:

There are four types of serialization. Namely:

- Binary Serialization - BinaryFormatter is used
- XML Serialization
- SOAP Serialization – SoapFormatter is used
- Custom Serialization

Serialize method of BinaryFormatter class has 2 parameters. Return type is void

1.File
2.Object

Similarly, deSerialize method of BinaryFormatter class has only one parameter. Return type is the Object. The parameter is for deSerialize method is the object.

Example:

```
BinaryFormatter bin=new BinaryFormatter();
bin.Serialize(FileStreamObject,ObjectToBeSerialized);
string msg= (string)bin.Deserialize(Object);
```

Example for Binary Serialization:

Serializing and Deserializing Predefined Object:

The below program serializes a string object and deserializes the same.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.IO; // to use filestream, we need to include this. Note: serialization needs FileStream.
using System.Runtime.Serialization.Formatters.Binary; //use this for binary serialization
namespace SerializationDemo
{
   class Binary
   {
      private FileStream fs;
      private BinaryFormatter bfr1,bfr2;

      public void SerializeMethod()
      {
         fs = new FileStream(@"D:\Binary.dat", FileMode.Create, FileAccess.Write);
         bfr1 = new BinaryFormatter();

         //serialization is done with serialize()
         bfr1.Serialize(fs, "Hi... Welcome to Binary Serialization");
         Console.WriteLine("Object Serialized!!!\n");

         fs.Close();
      }
      public void DeserializeMethod()
      {
         fs = new FileStream(@"D:\Binary.dat", FileMode.Open, FileAccess.Read);
         bfr2 = new BinaryFormatter();

          //Deserialization with Deserialize()
         string msg = (string)bfr2.Deserialize(fs);
         Console.WriteLine("message:"+msg);
         Console.WriteLine("Object DeSerialized!!!\n");
      }
   }
}
```

******Calling the above methods from main method() ********

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace SerializationDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         Binary binaryObj = new Binary();
         binaryObj.SerializeMethod();
         binaryObj.DeserializeMethod();
      }
   }

}
```

Output:

```
Object Serialized!!!

message:Hi... Welcome to Binary Serialization
Object DeSerialized!!!

Press any key to continue . . .
```

Serializing and Deserializing UserDefined Objects:

Let us now serialize and deserialize the object of an user defined class. Look at this example:

****A Class which has the employee class, whose object is to be serialized**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SerializationDemo
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
        Binary binaryObj = new Binary();
        binaryObj.SerializeMethod();
        binaryObj.DeserializeMethod();
      }
   }

   [Serializable]  //Objects of the Class to be serialized should be declared as "Serializable"
   class Employee
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public string City { get; set; }
      public void display()
      {
        Console.WriteLine("Id={0},Name={1},City={2}", Id, Name, City);
      }
   }
}
```

******Class which has serialization and deserialization methods***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO; // to use filestream, we need to include this. Note: serialization needs FileStream.
using System.Runtime.Serialization.Formatters.Binary; // to use Binary serialization, include this.
namespace SerializationDemo
{
   class Binary
   {
      private FileStream fs;
      private BinaryFormatter bfr1,bfr2;

      public void SerializeMethod()
      {
        Employee employeeObj = new Employee() { Id = 100, Name = "Lenaprasanna", City = "Tenali" };
        fs = new FileStream(@"D:\Binary.dat", FileMode.Create, FileAccess.Write);
        bfr1 = new BinaryFormatter();
```

```
        //serialization is done with serialize()
        bfr1.Serialize(fs, employeeObj);
        Console.WriteLine("Employee Object Serialized!!!\n");

        fs.Close();
    }
    public void DeserializeMethod()
    {
        fs = new FileStream(@"D:\Binary.dat", FileMode.Open, FileAccess.Read);
        bfr2 = new BinaryFormatter();

        //Deserialization
        Employee empObj = (Employee)bfr2.Deserialize(fs);
        Console.WriteLine("\n***After applying Deserialization***\nUsing the method display() of
employee class, displaying Id,Name,and City\n");
        empObj.display();
        Console.WriteLine("\n\n***Using get accessor of the object, displaying Id,Name,City****\n");
        Console.WriteLine("Id={0},Name={1},City={2}", empObj.Id,empObj.Name,empObj.City);
        Console.WriteLine("\nObject DeSerialized!!!\n");
    }
  }

}
```

Output:

```
Employee Object Serialized!!!


***After applying Deserialization***
Using the method display() of employee class, displaying Id,Name,and City

Id=100,Name=Lenaprasanna,City=Tenali


***Using get accessor of the object, displaying Id,Name,City****

Id=100,Name=Lenaprasanna,City=Tenali

Object DeSerialized!!!

Press any key to continue . . .
```

Example for SOAP serialization:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap; //it should be included for  SOAP serialization

namespace SerializationDemo
{
   class SOAP
   {
      //First add the Dll file. Below is the step in visual studio 2010 to be followed to add the DLL file
      //right click the projectName in the Solution Explorer window(Right side pane),select Add
reference.. Under .NET tab, check for System.Runtime.Serialization.Formatters.Soap. Select it and click
"OK".

       private FileStream fs;
      private SoapFormatter sfr1,sfr2;

      public void SerializeMethod()
      {
         EmployeeDetail employeeObj = new EmployeeDetail() { Id = 100, Name = "Lenaprasanna", City =
"Tenali" };
         //Note: file extension is .soap
         fs = new FileStream(@"D:\SOAP.soap", FileMode.Create, FileAccess.Write);
         sfr1 = new SoapFormatter();

         //serialization is done with serialize()
         sfr1.Serialize(fs, employeeObj);
         Console.WriteLine("Employee Object Serialized!!!\n");

         fs.Close();
      }
      public void DeserializeMethod()
      {
         fs = new FileStream(@"D:\SOAP.soap", FileMode.Open, FileAccess.Read);
         sfr2 = new SoapFormatter();
```

```
        //Deserialization
        EmployeeDetail empObj = (EmployeeDetail)sfr2.Deserialize(fs);

        Console.WriteLine("\n***After applying Deserialization***\nUsing the method display() of
employee class, displaying Id,Name,and City\n");
        empObj.display();
        Console.WriteLine("\n\n***Using get accessor of the object, displaying Id,Name,City****\n");
        Console.WriteLine("Id={0},Name={1},City={2}", empObj.Id,empObj.Name,empObj.City);
        Console.WriteLine("\nObject DeSerialized!!!\n");
      }
   }
   }
```

*******Another class which has Main Method, calling those two above methods***

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SerializationDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         SOAP soapObj = new SOAP();
         soapObj.SerializeMethod();
         soapObj.DeserializeMethod();

      }
   }

   [Serializable]  //Objects of the Class to be serialized should be declared as "Serializable"
   class EmployeeDetail
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public string City { get; set; }
      public void display()
      {
```

```
        Console.WriteLine("Id={0},Name={1},City={2}", Id, Name, City);
    }
  }
}
```

Output:

```
Employee Object Serialized!!!


***After applying Deserialization***
Using the method display() of employee class, displaying Id,Name,and City

Id=100,Name=Lenaprasanna,City=Tenali


***Using get accessor of the object, displaying Id,Name,City****

Id=100,Name=Lenaprasanna,City=Tenali

Object DeSerialized!!!

Press any key to continue . . .
```

Example for XML serialization:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml.Serialization; // This should be included for XML serialization
using System.IO;

namespace SerializationDemo
{
   [XmlRoot("Class_Person")] //This should be added
   public class PersonXML
   {

      //string m_sName;
      // int m_iAge;
      public PersonXML()
      {
```

```
    }

    [XmlElement("Property_Name")]
    public string Name
    {
      get;
      set;
    }
    [XmlElement("Property_Age")]
    public int Age
    {
      get;
      set;
    }
  }

  class XMLSerialize
  {
    //First add the Dll file.
    //right click the projectName in the Solution Explorer window(Right side pane),select Add
reference.. Under .NET tab, check for System.Runtime.Serialization.Formatters.Soap. Select it and click
"OK".

     private FileStream fs;
     private XmlSerializer objXmlSer;
     private StreamWriter objStrWrt;
     private StreamReader objStrRdr;

    public void SerializeMethod()
    {
        objXmlSer = new XmlSerializer(typeof(PersonXML));

        PersonXML personObj = new PersonXML() { Name = "Lenaprasanna", Age = 21 };
        //Note: file extension is .xml
        fs = new FileStream(@"D:\XMLSerialization.xml", FileMode.Create, FileAccess.Write);

        objStrWrt = new StreamWriter(fs);
        //serialization is done with serialize()
        //Important Note: In xml serialization, we have to pass the object of StreamWriter and not
the Object of FileStream
        objXmlSer.Serialize(objStrWrt, personObj);
```

```
            Console.WriteLine("Employee Object Serialized!!!\n");

            fs.Close();


        }
        public void DeserializeMethod()
        {
            fs = new FileStream(@"D:\XMLSerialization.xml", FileMode.Open, FileAccess.Read);

            objXmlSer = new XmlSerializer(typeof(PersonXML));
            //Deserialization
            objStrRdr = new StreamReader(fs);
            PersonXML personObj = (PersonXML)objXmlSer.Deserialize(objStrRdr);
            Console.WriteLine("\n\n***Using get accessor of the object, displaying Id,Name,City****\n");
            Console.WriteLine("Age={0},Name={1}", personObj.Age,personObj.Name);
            Console.WriteLine("\nObject DeSerialized!!!\n");
        }
    }

}
```

****Another class which calls the above two methods:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SerializationDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            XMLSerialize xmlObj = new XMLSerialize();
            xmlObj.SerializeMethod();
            xmlObj.DeserializeMethod();
        }
    }
}
```

Output:

```
Employee Object Serialized!!!


***Using get accessor of the object, displaying Id,Name,City****

Age=21,Name=Lenaprasanna

Object DeSerialized!!!

Press any key to continue . . .
```

22.Multi-Threading

Note:

- System.Threading is the class which provides Threading.
- Threading is a light weight process. It does not use lot of memory.
- Threading has Parallel execution path.
- Threading enables independent execution.
- There will be one default main thread.
- DotNet provides default thread in the form of main method.
- Thread scheduler is used to create main thread with the help of CLR.
- User can create any number of threads in the applications. Those threads are called as **Worker threads** or **Child threads**.
- Threads shares memory where as a process does not.
- Main thread is automatically started, but Child threads should be started manually by the users.
- There is a built in delegate called **ThreadStart** which is used to create threads.

Methods of Threads:

1. sleep(int milliseconds) - to stop the process of a thread for the given milli seconds of time.
2. Resume() – restarts the thread.
3. Start() – starts the thread.
4. Abort() – to kill the thread.

States of Threads:

A Thread can be in one the following state.

- **Unstarted**
  - In this state,Thread is Created within the CLR but not Started still.
- **Running**
  - After a Thread calls Start method, it enters this state.
- **WaitSleepJoin**
  - After a Thread calls its wait or Sleep or Join method, it enters this state.
- **Suspended**
  - In this state, Thread Responds to a Suspend method call.
- **Stopped**
  - In this state,The Thread is Stopped, either normally or Aborted.

Thread priority:

The Thread class's ThreadPriority property is used to set the priority of the Thread. A Thread may have one of the following values as its Priority: Lowest, BelowNormal, Normal, AboveNormal, Highest. The default property of a thread is Normal.

Sample program for multi threading:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading; // this should be included for threading
namespace ThreadingDemo
{
   class Program
   {
      static void ChildMethod1()
      {
         Console.WriteLine("Child thread1 is started for Execution");
         //Thread.Sleep(4000);
         for (char i = 'N'; i <= 'Z'; i++)
         {
            Thread.Sleep(1000);
            Console.WriteLine("Alphabet: " + i + "");
         }
         Console.WriteLine("Child thread1 ends");
      }
      static void Main(string[] args)
      {
         Thread t = Thread.CurrentThread; //instance of main thread is taken here. Note: Thread
t = new Thread() is INVALID as main thread cannot be created
         Console.WriteLine("Main thread is started for Execution");

         Thread objThread = new Thread(new ThreadStart(ChildMethod1)); //creating a child
thread

         objThread.Start(); // Child threads should be started manually.
         Console.WriteLine("Threadstate is :" + t.ThreadState);
         Console.WriteLine("Thread priority is " + t.Priority);
         for (char i = 'A'; i <= 'M'; i++)
```

```
            {
              Thread.Sleep(1000);
              Console.WriteLine("Alphabet: "+i + "");
            }
            Console.WriteLine("\n main thread ends");
          }
        }
      }
```

Output:

```
Main thread is started for Execution
Threadstate is :Running
Child thread1 is started for Execution
Thread priority is Normal
Alphabet: N
Alphabet: A
Alphabet: O
Alphabet: B
Alphabet: C
Alphabet: P
Alphabet: D
Alphabet: Q
Alphabet: E
Alphabet: R
Alphabet: F
Alphabet: S
Alphabet: G
Alphabet: T
Alphabet: H
Alphabet: U
Alphabet: V
Alphabet: I
Alphabet: W
Alphabet: J
Alphabet: X
Alphabet: K
Alphabet: Y
Alphabet: L
Alphabet: Z
Child thread1 ends
Alphabet: M

 main thread ends
Press any key to continue . . .
```

Array of threads:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace MultithreadingDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         MyClass[] objMyClass = new MyClass[4];
         objMyClass[0]=new MyClass("Child thread1");
         objMyClass[1]=new MyClass("Child thread2");
         objMyClass[2]=new MyClass("Child thread3");
         objMyClass[3]=new MyClass("Child thread4");
         foreach (MyClass obj in objMyClass)
         {
            do
            {
               Thread.Sleep(1000);
               Console.WriteLine("\nLoop Iteration " + obj.count + "in Main Thread");
               obj.count++;
            } while (obj.count < 10);

         }
      }
   }
   class MyClass
   {
      public int count;
      public Thread thrd;
      public MyClass(string name)
      {
         thrd = new Thread(this.run);
         thrd.Name = name;
         thrd.Start(); //starting the thread
         count = 0;
```

```
    }
    public void run()
    {
      Console.WriteLine(thrd.Name + " is started");
      do
      {
        Thread.Sleep(1000);
        Console.WriteLine("Loop Iteration " + count + "in" + thrd.Name);
        count++;
      } while (count < 10);

    }

  }

}
```

Output:

```
Child thread1 is started
Child thread2 is started
Child thread3 is started
Child thread4 is started
Loop Iteration 0inChild thread4
Loop Iteration 0inChild thread3
Loop Iteration 0inChild thread1

Loop Iteration 0in Main Thread
Loop Iteration 0inChild thread2
Loop Iteration 1inChild thread2
Loop Iteration 1inChild thread3
Loop Iteration 2inChild thread1

Loop Iteration 2in Main Thread
Loop Iteration 1inChild thread4
Loop Iteration 2inChild thread3
Loop Iteration 4inChild thread1

Loop Iteration 5in Main Thread
Loop Iteration 2inChild thread4
Loop Iteration 2inChild thread2
Loop Iteration 3inChild thread3
Loop Iteration 6inChild thread1
```

```
Loop Iteration 7in Main Thread
Loop Iteration 3inChild thread4
Loop Iteration 3inChild thread2
Loop Iteration 4inChild thread3

Loop Iteration 8in Main Thread
Loop Iteration 4inChild thread4
Loop Iteration 4inChild thread2
Loop Iteration 8inChild thread1
Loop Iteration 5inChild thread3

Loop Iteration 10in Main Thread
Loop Iteration 5inChild thread4
Loop Iteration 5inChild thread2
Loop Iteration 6inChild thread3

Loop Iteration 6in Main Thread
Loop Iteration 6inChild thread4
Loop Iteration 7inChild thread2
Loop Iteration 7inChild thread3

Loop Iteration 8in Main Thread
Loop Iteration 7inChild thread4
Loop Iteration 9inChild thread2
Loop Iteration 8inChild thread3

Loop Iteration 10in Main Thread
Loop Iteration 8inChild thread4
Loop Iteration 9inChild thread3

Loop Iteration 10in Main Thread
Loop Iteration 9inChild thread4

Loop Iteration 10in Main Thread
Press any key to continue . . .
```

Suspending and Resuming threads:

<u>Sample program:</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Suspend_And_Resume_In_Threads
{
```

```
  class mythread
  {
     public Thread thrd;
     public mythread(string name)
     {
        thrd = new Thread(new ThreadStart(this.run));
        thrd.Name = name;
        thrd.Start();
     }
     //this is the entry point of thread
     void run()
     {
        Console.WriteLine("\n"+thrd.Name + " starting.\n");
        for (int i = 1; i <= 100; i++)
        {
           Console.Write(i +" ");
           if (i % 10 == 0)
           {
              Console.Write("\n");
              Thread.Sleep(500);
           }
        }
        Console.WriteLine("\n"+thrd.Name + " exiting.\n");
     }
  }

  class Program
  {
     static void Main(string[] args)
     {
        mythread objmythread = new mythread("my thread");
        objmythread.thrd.Suspend();
        Console.WriteLine("suspending");
        objmythread.thrd.Resume();
        Console.WriteLine("started");
     }
  }
}
```

Output:

```
suspending
started
my threadstarting.
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
```

```
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
my threadexiting.
Press any key to continue . . .
```

Thread synchronization:

Example without lock() method :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;


namespace ThreadSynchronization
{
   class MyThread
   {
      public Thread thrd;
      int[] a;
      int answer;
      //create a sum array object for all instances of MyThread
      static SumArray sa = new SumArray();

      public MyThread(string name, int[] nums)
      {
         a = nums;
         thrd = new Thread(new ThreadStart(this.run));
         thrd.Name = name;
         thrd.Start(); //calls run method()
      }
      public void run()
      {
         Console.WriteLine(thrd.Name + " is starting!!");
         answer = sa.sumIt(a);
         Console.WriteLine("sum is: " + answer);
```

```
        Console.WriteLine(thrd.Name + " is completed!!");
      }
    }

    class SumArray
    {
      int sum;

      public int sumIt(int[] nums)
      {
          sum = 0; //reset sum
          for (int i = 0; i < nums.Length; i++)
          {
            sum += nums[i];
            Console.WriteLine("Running total for " + Thread.CurrentThread.Name + " is " + sum);
            Thread.Sleep(1000);
          }
          return sum;
      }

      class Program
      {
        static void Main(string[] args)
        {
          int[] arr = new int[5] { 1, 2, 3, 4, 5 };
          MyThread objMyThread = new MyThread("ChildThread1",arr);
          MyThread objMyThread1 = new MyThread("ChileThread2", arr);

        }
      }
    }
}
```

Output:

```
ChildThread1 is starting!!
ChileThread2 is starting!!
Running total for ChildThread1 is 1
Running total for ChileThread2 is 1
Running total for ChileThread2 is 3
Running total for ChildThread1 is 5
Running total for ChildThread1 is 8
```

```
Running total for ChileThread2 is 8
Running total for ChildThread1 is 16
Running total for ChileThread2 is 16
Running total for ChileThread2 is 26
Running total for ChildThread1 is 26
sum is: 26
ChileThread2 is completed!!
sum is: 26
ChildThread1 is completed!!
Press any key to continue . . .
```

Example with lock() method :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;


namespace ThreadSynchronization
{
   class MyThread
   {
      public Thread thrd;
      int[] a;
      int answer;
      //create a sum array object for all instances of MyThread
      static SumArray sa = new SumArray();

      public MyThread(string name, int[] nums)
      {
         a = nums;
         thrd = new Thread(new ThreadStart(this.run));
         thrd.Name = name;
         thrd.Start(); //calls run method()
      }
      public void run()
      {
         Console.WriteLine(thrd.Name + " is starting!!");
         answer = sa.sumIt(a);
         Console.WriteLine("sum is: " + answer);
         Console.WriteLine(thrd.Name + " is completed!!");
```

```
    }
  }

  class SumArray
  {
    int sum;

    public int sumIt(int[] nums)
    {
      lock (this)
      {
        //Lock the entire method
        sum = 0; //reset sum

        for (int i = 0; i < nums.Length; i++)
        {
          sum += nums[i];
         Console.WriteLine("Running total for " + Thread.CurrentThread.Name + " is " + sum);
          Thread.Sleep(1000);
        }
        return sum;
      }

    }

    class Program
    {
      static void Main(string[] args)
      {
        int[] arr = new int[5] { 1, 2, 3, 4, 5 };
        MyThread objMyThread = new MyThread("ChildThread1",arr);
        MyThread objMyThread1 = new MyThread("ChileThread2", arr);

      }
    }
  }
}
```

Output:

```
ChildThread1 is starting!!
ChileThread2 is starting!!
```

```
Running total for ChildThread1 is 1
Running total for ChildThread1 is 3
Running total for ChildThread1 is 6
Running total for ChildThread1 is 10
Running total for ChildThread1 is 15
sum is: 15
ChildThread1 is completed!!
Running total for ChileThread2 is 1
Running total for ChileThread2 is 3
Running total for ChileThread2 is 6
Running total for ChileThread2 is 10
Running total for ChileThread2 is 15
sum is: 15
ChileThread2 is completed!!
Press any key to continue . . .
```

Note:

- From the above two examples, we conclude, lock() method ensures consistency in final result.
- Once a method is locked, it is not held to any other threads which requests for accessing the same method.

23.LINQ

Note:

- LINQ is expanded as Language Integrated Query.
- It has lots of standard Query operators.
- It is defined in System.Linq
- LINQ is used to filter the data.
- LINQ queries data from various data sources with the help of querying operator.
- The two interfaces that returns data are given below:
    - IEnumerable<type>
    - IQueryable<type>

Basic syntax for LINQ expression should contain two mandatory clauses. They are listed below:

- From clause
- Select clause

Other optional clauses are:

- Where
- Order by

Basic Syntax for LINQ expression:

```
IEnumerable <type> ExpressionVariable = from rangeVariable in DataSource
                                        where condition
                                        select rangeVariable;
```

LINQ contains the below categories:

- LINQ to Objects
    - LINQ to classes
    - LINQ to Arrays
    - LINQ to Collections
- LINQ to ADO.net
- LINQ to XML
- LINQ to SQL

There is a two type of query expression. Namely,

- **Immediate Execution** - query data will be immediately converted into collection type. Collection typce may be ToArray(),ToCount(),ToList(), etc..
  - *Syntax:*

```
var QueryVar = (QueryExpression).ToArray();
Console.WriteLine(QueryVar)
```

- **Defered Execution**
  - *Syntax:*

```
var QueryVar =QueryExpression;
foreach(var LocalVar in QueryVar)
{
        Console.WriteLine(LocalVar);
}
```

Syntax using "Let" keyword in Linq Expression:

```
var QueryVar = from rangeVariable in DataSource
               let ExpressionStatements
               where condition
               select rangeVariable;
```

For detailed understanding of "Let" keyword, refer the program given in this section.

Program that explains LINQ:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml; //Include this for handling XML
using System.Xml.Linq; //Include this for LinqToXML()
namespace LinQtoObjectDemo
{
   class Program
   {
      static void linqToObjects()
      {
         int[] intArr = new int[5] { -10, 7, -2, 20, 35 };
         int[] intArr1 = new int[5] { -12, 7, 50, 210, 35 };
```

```
Console.WriteLine("\n----Displaying all entries of intArr----\n");
var QueryVar = from n in intArr
            select n;

foreach (int value in QueryVar)
{
   Console.WriteLine(value);
}


Console.WriteLine("\n1.Displaying positive values using Foreach loop\n");
Console.WriteLine("_____________________________________________\n");
foreach (int value in QueryVar)
{
   if (value > 0)
   {
      Console.WriteLine(value);
   }
}

Console.WriteLine("\n2.Displaying positive values using LinqExpression\n");
Console.WriteLine("_____________________________________________\n");

var QueryVar1 = from n in intArr
                where n > 0
                select n;
foreach (int value in QueryVar1)
{
   Console.WriteLine(value);
}

Console.WriteLine("\n3.Displaying positive even values using LinqExpression\n");

Console.WriteLine("_____________________________________________________\n");

  var QueryVar2 = from n in intArr
                  where n > 0 && n % 2 == 0
                   select n;
```

//Note:Any number of where clauses can be used. We can replace this statement with two different where clauses like "where n>0 where n%2==0" which will yield the same result.

```
        foreach (int value in QueryVar2)
        {
           Console.WriteLine(value);
        }

        string[] strArr = new string[5] { "Lena", "Prasanna", "Telugu", " TamilNadu",
"Andhrapradesh" };
        var QueryVar3 = from n in strArr
                    select n;
        Console.WriteLine("\n---- Displaying all entries of strArr ----\n");

        foreach (string value in QueryVar3)
        {
           Console.WriteLine(value);
        }

        Console.WriteLine("\n1.Displaying strings that starts with T\n");
        Console.WriteLine("_____________________________________________\n");
        var QueryVar4 = from n in strArr
                    where n.StartsWith("T")
                    select n;
//Note: n.StartsWith("T")  statement can be replaced with where  n.Substring(0,1).Equals("T")

        foreach (string value in QueryVar4)
        {
           Console.WriteLine(value);
        }

        Console.WriteLine("\n2.Displaying strings which contains a char \"a\" showing in
descending order");

Console.WriteLine("_____________________________________________________________\n");
        var QueryVar5 = from n in strArr
                    where n.Contains('a')
              //this statement can be replaced with  where n.IndexOf('a') != -1
                    orderby n descending
            //order by clause orders the result in the asc/desc order
                    select n;
        foreach (string value in QueryVar5)
        {
```

```
    Console.WriteLine(value);
}

Console.WriteLine("\n3.Displaying strings that has length of 6\n");
Console.WriteLine("_____________________________________\n");

var QueryVar6 = from n in strArr
            where n.Length == 6
            select n;
foreach (string value in QueryVar6)
{
    Console.WriteLine(value);
}

Console.WriteLine("\n4.Displaying strings after replacing 'a' with 'n' in the string");
Console.WriteLine("_____________________________________________________\n");

var QueryVar7 = from n in strArr
                select n.Replace('a', 'n');

foreach (string value in QueryVar7)
{
    Console.WriteLine(value);
}

Console.WriteLine("\n5.Displaying strings in lower case");
Console.WriteLine("_______________________________\n");

var QueryVar8 = from n in strArr
                select n.ToLower();

foreach (string value in QueryVar8)
{
    Console.WriteLine(value);
}

Console.WriteLine("\n6.Displaying only the first 3 characters of the strings");
Console.WriteLine("_____________________________________________________\n");

var QueryVar9 = from n in strArr
            select n.Remove(3);
```

```
        foreach (string value in QueryVar9)
        {
           Console.WriteLine(value);
        }
        Console.WriteLine("\n7.Truncating all the leading and trailing whitespaces in the string");

Console.WriteLine("_____________________________________________________________\n");
        var QueryVar10 = from n in strArr
                         select n.Trim();
        foreach (string value in QueryVar10)
        {
           Console.WriteLine(value);
        }

        Console.WriteLine("\n8.Calling multiple functions within a single select statement - using
anonymous method");

Console.WriteLine("_____________________________________________________________\n");

        var QueryVar11 = from n in strArr
                 select new
                      {
                         a = n.PadLeft(10, '*'),
                         b = n.PadRight(10, '*')
                      //Note:local variables for an anonymous class need not be declared. It can
be used as and then inside the anonymous method.
                      };

        foreach (var value in QueryVar11)
        {
           Console.WriteLine("Left padded string: " + value.a + "\nRight Padded String: " +
value.b);
        }

        Console.WriteLine("\n9.Display common values in intArr1 and intArr2");
        Console.WriteLine("_____________________________________________\n");

        var QueryVar12 = intArr.Intersect(intArr1);
        foreach (int value in QueryVar12)
        {
```

```
            Console.WriteLine(value);
        }

        Console.WriteLine("\n10.Display numbers in intArr1 if it is less than the average of
intArr1");

Console.WriteLine("_____________________________________________________________\n");

        var QueryVar13 = from n in intArr1
                         where n < intArr1.Average()
                        select n;
        foreach (int value in QueryVar13)
        {
            Console.WriteLine(value);
        }

        Console.WriteLine("\nUsing Let Keyword");
        var QueryVar14 = from n in intArr1
                         let avgValue = intArr1.Average()
                         where n < avgValue
                         select n;
        foreach (int value in QueryVar14)
        {
            Console.WriteLine(value);
        }

    }
    static void linqToClasses()
    {

        Person[] objPersonArray = new Person[4]{
            new Person{Id=101,IdRole=346238,FirstName="Lena",LastName="Prasanna"},
            new Person{Id=102,IdRole=346239,FirstName="Kishore",LastName="Sivagnanam"},
            new Person{Id=103,IdRole=346240,FirstName="Telugu",LastName="Language"},
            new Person{Id=104,IdRole=346241,FirstName="Tamil",LastName="Nadu"}
        };

        Console.WriteLine("\n1.Printing datas using IEnumerable interface");
        Console.WriteLine("_____________________________________________\n");
        IEnumerable<Person> QueryVar = from n in objPersonArray
                                       where n.Id > 102
```

```
                                        select n;
```

//Note:IEnumerable Interface is of Collection type. Hence we can select the class details on the whole. If we give "select n.Id or any other properties defined inside it, then it will show error.

```
        Console.WriteLine("\nID   IDRole   FirstName   LastName");
        Console.WriteLine("---------------------------------\n");

        foreach (Person value in QueryVar)
        {
           Console.WriteLine(value.Id + "   " + value.IdRole + "  " + value.FirstName + "   " +
value.LastName);
        }

        Console.WriteLine("\n2.Printing datas without using IEnumerable interface");
        Console.WriteLine("_____________________________________________________\n");

        var QueryVar1 = from n in objPersonArray
                    where n.Id > 102
                    select n;
```

// Note, Here, using select n.Id or anyother property defined in the Person class can be used and it is valid.

```
        Console.WriteLine("\nID   IDRole   FirstName   LastName");
        Console.WriteLine("----------------------------------");
        foreach (var value in QueryVar1)
        {
           Console.WriteLine(value.Id + "   " + value.IdRole + "  " + value.FirstName + "   " +
value.LastName);
        }
        Console.WriteLine("\n3.Printing only Id, FirstName and LastName");
        Console.WriteLine("_____________________________________________\n");

        var QueryVar2 = from n in objPersonArray
                        where n.Id > 102
                        select new
                       {
                            n.Id,
                            n.FirstName,
                            n.LastName
                        };
```

//Note: anonymous method that extracts id, first name and lastname.In general, inorder to extract specific fields, use anonymous method

```
Console.WriteLine("\nID   FirstName LastName");
Console.WriteLine("----------------------");

foreach (var value in QueryVar2)
{
   Console.WriteLine(value.Id + "  " + value.FirstName + "   " + value.LastName);
}


Role[] objRoleArray = new Role[4]{ new Role{Id = 101,RollName="ProgrammerAnalyst"},
                 new Role{Id = 102,RollName="Software Analyst"},
                 new Role{Id = 103,RollName="C# programmer"},
                 new Role{Id = 104,RollName="Java Programmer"}
                };

Console.WriteLine("\n4.Joining two user defined class objects using on clause");
Console.WriteLine("_____________________________________________\n");

var QueryVar3 = from p in objPersonArray
                 join r in objRoleArray
                 on p.Id equals r.Id
                select new
                {
                     p.IdRole,
                     p.FirstName,
                     p.LastName
                 };
```

//Note: when we use on clause, use equals keyword

```
Console.WriteLine("\nRole    FirstName    LastName");
Console.WriteLine("----------------------------");
foreach (var value in QueryVar3)
{
   Console.WriteLine(value.IdRole + "   " + value.FirstName + "   " + value.LastName);
}

Console.WriteLine("\n5.Joining two user defined class objects using where clause");
```

```
Console.WriteLine("_____________________________________________________\n");

    var QueryVar4 = from p1 in objPersonArray
                    from r1 in objRoleArray
                    where p1.Id == r1.Id
                    select new
                    {
                        p1.IdRole,
                        p1.FirstName,
                        p1.LastName
                    };
     //Note: when we use where clause, use == operator.

    Console.WriteLine("\nRole    FirstName    LastName");
    Console.WriteLine("----------------------------");
    foreach (var value in QueryVar4)
    {
       Console.WriteLine(value.IdRole + "   " + value.FirstName + "   " + value.LastName);
    }
  }

  static void linqToCollections()
  {
    IEnumerable<int> QueryVar = from n in ListIntCollectionMethod()
                         select n;

   //Note:Calling the method that returns List of Integer objects

    Console.WriteLine("1.Integer Objects");
    Console.WriteLine("___________________\n");
    foreach (var i in QueryVar)
    {
       Console.Write(i + " ");
    }
    Console.Write("\n");

    //Array of objects

    Employee[] objEmployee = new Employee[3];
   objEmployee[0] = new Employee() { Id = 346238, FirstName = "prasanna", City = "Tenali" };
```

```
        objEmployee[1] = new Employee() { Id = 346365, FirstName = "anu", City = "chennai" };
        objEmployee[2] = new Employee() { Id = 346534, FirstName = "chanakya", City =
"hyderabad" };

        Console.WriteLine("\n2.Displaying from Array of objects");
        Console.WriteLine("_________________________________\n");

        IEnumerable<Employee> listEmpQuery = from n in objEmployee
                                             where n.FirstName.StartsWith("p")
                                             select n;

        Console.WriteLine("FirstName   City");
        Console.WriteLine("----------------");
        foreach (Employee e in listEmpQuery)
        {
           Console.Write(e.FirstName + "    " + e.City);
        }

        Console.Write("\n\n3.Displaying from Collection of Employee Class Objects\n");
        Console.WriteLine("_____________________________________________________");
        //or

        //collection Initializer

        IEnumerable<Employee> QueryVar1 = from n in ListCollectionOfEmployee()
                                          select n;
//Note:Calling the ListCollectionOfEmployee() method that returns the collection object

        Console.WriteLine("\nFirstName   ID     City");
        Console.WriteLine("------------------------");
        foreach (var i in QueryVar1)
        {
           Console.WriteLine(i.FirstName +"   "+ i.Id+"   " + i.City);
        }

        Console.Write("\n\n4.Displaying from Collection of Role Class Objects - Type1\n");
        Console.WriteLine("_____________________________________________________\n");

        IEnumerable<Role> QueryVar3 = from n in ListCollectionOfRole()
                                      select n;
```

//calling Method 1 of initialization.In which ListCollectionOfRole() creates a List object, initializes it and then returning the object

```
        Console.WriteLine("ID   RollName");
        Console.WriteLine("-------------");
        foreach (var i in QueryVar3)
        {
           Console.WriteLine(i.Id +"    "+ i.RollName);
        }

        Console.Write("\n\n5.Displaying from Collection of Role Class Objects - Type2\n");

Console.WriteLine("_____________________________________________________________\n");

        IEnumerable<Role> QueryVar4 = from n in ListCollectionOfRoleClass()
                                      select n;
```

//Note: calling Method 2 of initialization.In which ListCollectionOfRoleClass() creates a nameless list object, initializes it and then returning the object

```
        Console.WriteLine("ID   RollName");
        Console.WriteLine("-------------");
        foreach (var i in QueryVar4)
        {
           Console.WriteLine(i.Id + "    " + i.RollName);
        }

        Console.Write("\n\n5.Displaying from Collection of Salary Class Objects\n");
        Console.WriteLine("_____________________________________________________\n");

        Console.WriteLine("ID      Year   SalaryPaid");
        Console.WriteLine("-----------------------");
        IEnumerable<Salary> QueryVar5 = from n in ListCollectionOfSalary()
                          select n;

        foreach (var i in QueryVar5)
        {
           Console.WriteLine(i.Id + "   " + i.Year + "   " + i.Salarypaid);
        }

        Console.Write("\n\n6.Displaying join of IEnumerable<Employee> object and  Salary class
object\n");
        Console.WriteLine("_____________________________________________\n");
```

```
        Console.WriteLine("Note:IEnumerable<Employee> has all the entries that has the
FirstName starting with 'P'");

        var QueryVar6=from e in listEmpQuery
                join s in ListCollectionOfSalary()
                on e.Id equals s.Id
                select new{e.Id,e.FirstName,s.Salarypaid};

        Console.WriteLine("ID      FirstName   SalaryPaid");
        Console.WriteLine("------------------------------");
        foreach (var i in QueryVar6)
        {
           Console.WriteLine(i.Id + "   " + i.FirstName + "   " + i.Salarypaid);
        }

        Console.WriteLine("\n\nRefining the above result with FirstName and Displaying total
salary and Name");

        var resultQuery = from q in QueryVar6
                            group q by q.FirstName into gp
                           select new
                    {
                       totalsalary = gp.Sum(q => q.Salarypaid),
                       gp.Key
                    };
//Note: gp stores all the entries that are grouped by FirstName.

//Note: q=>q.Salarypaid is a lambda expression. Here we have two records with the ID
346238(Refer Output). Consider a scenario that there is an another user with ID 346200, in
that case, q points to 346238 at 1st iteration and 346200 in the 2nd iteration. We extract the
salaryPaid of 346238 in the first Iteration and calculating the sum for the same. Similarly we
proceed with second ID too.

//Note:Key will have FirstName with which we grouped.

        Console.WriteLine("\nName      Total Salary");
        Console.WriteLine("-----------------------");
        foreach (var i in resultQuery)
        {
           Console.WriteLine(i.Key+"   "+i.totalsalary);
        }
```

```
}

static List<int> ListIntCollectionMethod()  //returns Int list object
{
   List<int> objList = new List<int>() { 111, 222, 333, 444 };
   return objList;
}


static List<Employee> ListCollectionOfEmployee()
{

   List<Employee> objlistemployee = new List<Employee>(){
   new Employee() { Id = 346200, FirstName = "Kishore", City = "chennai" },
   new Employee() { Id = 346238, FirstName = "Prasanna", City = "Tenali" },
   new Employee() { Id = 346197, FirstName = "Lakshman", City = "salem" }
};
   return objlistemployee;
}

//method 1 of Initialization
static List<Role> ListCollectionOfRole()
{
   List<Role> objlistrole = new List<Role>()
   {
      new Role(){Id=345,RollName="PAT"},
      new Role(){Id=346,RollName="CAT"},
      new Role(){Id=347,RollName="ELT"}
   };
   return objlistrole;
}

//Method 2 of Initialization
static List<Role> ListCollectionOfRoleClass()
{
   return new List<Role> //Creating nameless List object of type Role
   {
      new Role(){Id=345,RollName="PAT"},
      new Role(){Id=346,RollName="CAT"},
      new Role(){Id=347,RollName="ELT"}
```

```
        };
    }


    static List<Salary> ListCollectionOfSalary()
    {
        List<Salary> objlistsalary = new List<Salary>()
        {
            new Salary(){Id=346238,Year=2008,Salarypaid=20000},
             new Salary(){Id=346200,Year=2012,Salarypaid=21000},
             new Salary(){Id=346197,Year=2012,Salarypaid=22000},
              new Salary(){Id=346238,Year=2012,Salarypaid=22000}

        };
        return objlistsalary;
    }


    static void linqToXML()
    {
        var doc = XDocument.Load("D:\\Customers.xml");

        var results = from c in doc.Descendants("Customer")
                      where c.Attribute("City").Value == "London"
                      select c;
        Console.WriteLine("\n1.Displaying the XML elements of Customer whose City is
London");
        Console.WriteLine("\n-------------------------------------------------------------\n");
        Console.WriteLine("Results:\n");

        foreach (var contact in results)
            Console.WriteLine("{0}\n", contact);

        XElement transformedResults = new XElement("Londoners",
                                from customer in results
                                select new XElement("Contact",
                                     new XAttribute("ID", customer.Attribute("CustomerID").Value),
                                     new XAttribute("Name", customer.Attribute("ContactName").Value),
                                     new XElement("City", customer.Attribute("City").Value)));
```

```
        Console.WriteLine("\n\n2.Manipulating the above documents and appending a new
element tag <Londoners>");
        Console.WriteLine("\n-----------------------------------------------------------------------------\n");
        Console.WriteLine("Results:\n{0}", transformedResults);
        transformedResults.Save("D:\\Customers_Saved.xml");
    }

    static void Main(string[] args)
    {
        Console.WriteLine("\n***Entering LinqToObjects****\n");
        linqToObjects();


        Console.WriteLine("\n***Entering LinqToClasses***\n");
        linqToClasses();

        Console.WriteLine("\n***Entering LinqToCollections***\n");
        linqToCollections();

        Console.WriteLine("\n***Entering LinqToXML***\n");
        linqToXML();
    }

    class Person
    {
        public int Id
        {
            get;
            set;
        }
        public int IdRole
        {
            get;
            set;
        }
        public string FirstName
        {
            get;
            set;
        }
        public string LastName
```

```
        {
            get;
            set;
        }
    }
    class Role
    {
        public int Id { get; set; }
        public string RollName { get; set; }
    }
    class Employee
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string City { get; set; }
    }

 class Salary
    {
        public int Id { get; set; }
        public int Year { get; set; }
        public int Salarypaid { get; set; }
    }
  }
}
```

Output:

```
***Entering LinqToObjects****


----Displaying all entries of intArr----

-10
7
-2
20
35
```

```
1.Displaying positive values using Foreach loop

_____________________________________________

7
20
35

2.Displaying positive values using LinqExpression

_____________________________________________

7
20
35

3.Displaying positive even values using LinqExpression

_____________________________________________________

20

---- Displaying all entries of strArr ----

Lena
Prasanna
Telugu
 TamilNadu
Andhrapradesh

1.Displaying strings that starts with T

_____________________________________

Telugu

2.Displaying strings which contains a char "a" showing in descending order
_____________________________________________________________________________

Prasanna
Lena
Andhrapradesh
 TamilNadu

3.Displaying strings that has length of 6

_____________________________________________

Telugu
```

```
4.Displaying strings after replacing 'a' with 'n' in the string
_____________________________________________________________

Lenn
Prnsnnnn
Telugu
 TnmilNndu
Andhrnprndesh

5.Displaying strings in lower case
__________________________________

lena
prasanna
telugu
 tamilnadu
andhrapradesh

6.Displaying only the first 3 characters of the strings
_____________________________________________________

Len
Pra
Tel
 Ta
And

7.Truncating all the leading and trailing whitespaces in the string
_____________________________________________________________

Lena
Prasanna
Telugu
TamilNadu
Andhrapradesh

8.Calling multiple functions within a single select statement - using anonymous
method
_____________________________________________________________________________

Left padded string: ******Lena
Right Padded String: Lena******
Left padded string: **Prasanna
Right Padded String: Prasanna**
Left padded string: ****Telugu
Right Padded String: Telugu****
Left padded string:  TamilNadu
Right Padded String:  TamilNadu
Left padded string: Andhrapradesh
Right Padded String: Andhrapradesh
```

```
9.Display common values in intArr1 and intArr2
_____________________________________________

7
35

10.Display numbers in intArr1 if it is less than the average of intArr1
_____________________________________________________________________

-12
7
50
35

Using Let Keyword
-12
7
50
35

***Entering LinqToClasses***

1.Printing datas using IEnumerable interface
____________________________________________


ID   IDRole   FirstName   LastName
---------------------------------

103   346240  Telugu   Language
104   346241  Tamil   Nadu

2.Printing datas without using IEnumerable interface
____________________________________________________


ID   IDRole   FirstName   LastName
-----------------------------------
103   346240  Telugu   Language
104   346241  Tamil   Nadu

3.Printing only Id, FirstName and LastName
__________________________________________


ID   FirstName LastName
-----------------------
103  Telugu   Language
104  Tamil   Nadu
```

```
4.Joining two user defined class objects using on clause
_____________________________________________________


Role    FirstName    LastName
-----------------------------
346238   Lena   Prasanna
346239   Kishore   Sivagnanam
346240   Telugu   Language
346241   Tamil   Nadu

5.Joining two user defined class objects using where clause
____________________________________________________________


Role    FirstName    LastName
-----------------------------
346238   Lena   Prasanna
346239   Kishore   Sivagnanam
346240   Telugu   Language
346241   Tamil   Nadu



***Entering LinqToCollections***

1.Integer Objects
_________________

111 222 333 444

2.Displaying from Array of objects
__________________________________

FirstName   City
----------------
prasanna    Tenali

3.Displaying from Collection of Employee Class Objects
______________________________________________________

FirstName   ID     City
------------------------
Kishore   346200   chennai
Prasanna   346238   Tenali
Lakshman   346197   salem


4.Displaying from Collection of Role Class Objects - Type1
```

```
_____________________________________________________

ID   RollName
-------------
345    PAT
346    CAT
347    ELT


5.Displaying from Collection of Role Class Objects - Type2
_____________________________________________________

ID   RollName
-------------
345    PAT
346    CAT
347    ELT


5.Displaying from Collection of Salary Class Objects
_____________________________________________

ID       Year   SalaryPaid
-----------------------
346238   2008   20000
346200   2012   21000
346197   2012   22000
346238   2012   22000


6.Displaying join of IEnumerable<Employee> object and  Salary class object
_____________________________________________________________________

Note:IEnumerable<Employee> has all the entries that has the FirstName starting w
ith 'P'
ID       FirstName   SalaryPaid
-----------------------------
346238   prasanna   20000
346238   prasanna   22000


Refining the above result with FirstName and Displaying total salary and Name

Name      Total Salary
-----------------------
prasanna   42000
```

```
***Entering LinqToXML***


1.Displaying the XML elements of Customer whose City is London

------------------------------------------------------------

Results:

<Customer CustomerID="CONSH" City="London" ContactName="Elizabeth Brown" />

<Customer CustomerID="EASTC" City="London" ContactName="Ann Devon" />


2.Manipulating the above documents and appending a new element tag <Londoners>

-----------------------------------------------------------------------------

Results:
<Londoners>
  <Contact ID="CONSH" Name="Elizabeth Brown">
    <City>London</City>
  </Contact>
  <Contact ID="EASTC" Name="Ann Devon">
    <City>London</City>
  </Contact>
</Londoners>
Press any key to continue . . .
```

24. DotNet InterOperability

Note:

- Interoperability is the ability of content or system to work together through the use of agreed standards and specifications.
- It is defined in **System.runtime.Interoperable**
- Interoperablity of **DotNet code with COM components** is called as **COM interopability**
- Interoperability of **COM components with DotNET** is called as **DotNET interoperability**
- Interoperability of **DotNet code with Win32 DLL files** are called as **P/Invoke.**
- Runtime callable Wrapper is generated by **RegAsm.exe**
- COM callable Wrapper is generated by **Tlbimp.exe**
- **IUnknown** and **Idispatch** interfaces enables interaction between COM and DotNet Interactions.
- **coCreateInstance()** method is used to create ojects using **COM Components** where as **new operator** is used to create objects in **DotNet Components**.
- DotNet components provides **object based** Communication
- COM components provides **interface based** Communication
- COM components uses **reference count** to deallocate objects whereas DotNet components use garbage Collector to deallocate objects
- GC.collect() is the method that is used to release te objects which are not used in the program.
- Obj.Dispose() is used to destroy the particular object Obj from the scope. This method will not release the memory.

Difference between Managed code and Un managed code :

Managed code:
The source code which can be executed **within** the CLR environment.

Un managed code:
The source code which can be executed **outside** the CLR environment.

Important points in nutshell:

DotNet	COM
1.Object based communication	1.Interface based communication
2.Garbage Collector to manage memory	2.Reference count will be used to manage memory
3.Follows Type Standard objects	3.Follows Binary Standard objects
4.Objects are created by normal new operator	4.Objects are created by coCreateInstance
5.Exceptions will be returned	5.HRESULT will be returned
6.Object information resides in assembly files	6.Object information resides in Type library

DotNet Marshalling:

- The process of converting an object between types when sending it across contexts is known as marshalling.
- The DotNet runtime automatically generates code to translate calls between managed code and unmanaged code.
- While transferring calls DotNet handles the data type conversion from the server data type to client data type. This is known as marshalling.
- Marshaling occurs between managed heap and unmanaged heap.

Types of Marshalling:

There are 2 types of marshaling. Namely:

- **Interop marshaling**
  - When the server object is created in the same apartment of client, all data marshaling is handled by Interop marshaling.
  - When type conversion happens in a small environment, we go for interop marshaling.
- **COM marshaling**
  - COM marshaling involved whenever the calls between managed code and unmanaged code are in different apartment.
  - We go for COM marshaling to communicate with wide range of platforms.

25.Web Services

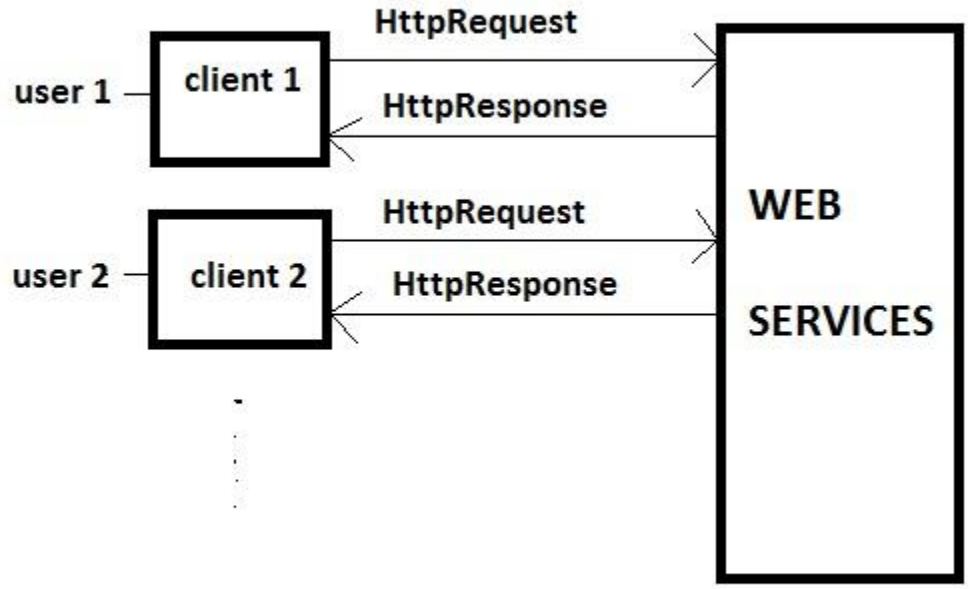

Figure 1. Web Services

Explanation to figure 1 is self explanatory. Web services responds to the requests sent by multiple clients at the same time. Client raises HttpRequest and Web services responds with the corresponding HttpResponse.

Note:

- Communication can be done by
  - **HttpRequest Protocol**
  - **HttpResponse protocol**
  - **SOAP**

  } **Data sharing in XML type format**
- **WSDL** (Web Service Description Language) which **describes how services defined** with web methods and parameters and how to use it.
- **SOAP**(Simple Object Access Protocol)
  - Tells how UDDI directories and web service classes are identified and communicated.
  - Shares datas in SOAP messages.

- **Note:** UDDI is expanded as **Universal Description Discovery and Integration**. It provides directory path for accessing web services.

- A Web service is a **programmable application logic** which are **accessible via standard web protocols** like SOAP,etc.
- Inorder to transfer the datas, it should be in a specific format. So explicit serialization has to be done on datas.

Note: *To create a webservice in visual studio 2010, select "Web" under "Visual C#" and click "ASP.NET Empty Web Application". Rename project name and click ok. Right click on project name, goto add , New Item… click "WebService", rename the webservice name and click OK.*

- If we have many methods in a webservice, those methods are displayed in the alphabetical ascending order.
- [WebMethod] is to be inserted before writing any method. It is very important.
- Any method should return any value. We cannot print any message using Console class.

<u>Sample program that demonstrates Web services:</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace WebClientApplication
{
   /// <summary>
   /// Summary description for WebServiceClass
   /// </summary>
   [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
   [System.ComponentModel.ToolboxItem(false)]
   // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following
line.
   // [System.Web.Script.Services.ScriptService]

   public class WebServiceClass : System.Web.Services.WebService //WebServices class extends
System.Web.Services.WebService
   {
```

//Note: Web method attribute should be given inorder to provide webservice. If not given, the method given next to it will not be used at the client side.

```
    [WebMethod]
    public string HelloWorld()
    {
      return "Hello World";
    }

    [WebMethod]
    public string Display(string msg)
    {
      return "Message from service: " + msg;
    }

    [WebMethod]
    public long AreaOfTriangle(int breadth, int height)
    {
      return (breadth * height) / 2;
    }
  }
}
```

26. Extension Methods in C#

Note:

- Extension methods can be created for any types under a static class only.
- An extension method uses “this” keyword in the parameters list.
- Extension Methods with the optional parameter is a new feature available only in DotNet 4.0 framework.
- Default parameters are to given in the right extreme in the argument list.

Sample program that illustrates extension methods:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NewFeaturesDemo
{
   static class Program //class should be static to use extension methods
   {
      static void Main(string[] args)
      {
        Console.WriteLine("\n****Capitalizing first letter of a string****\n");
         string s = "telugu is a beautiful language";
         Console.WriteLine("string: "+s.UpperCaseFirstLetter()); //the string s is implicitly passed as an argument.

         Console.WriteLine("\n***Printing Factorial***\n");
         int i = 5;
         Console.WriteLine("Factorial of {0} is :{1}",i,i.Factorial()); // integer i is passed as an argument.

         Console.WriteLine("\n****Calling Optional Records with default argument***\n");
         optionalRecords(100);
         Console.WriteLine("\n****Calling Optional Records with normal arguments***\n");
         optionalRecords(100, "bye!!!");

         Console.WriteLine("\n***Calling Named Arguments***\n");
         NamedRecords( z:100, x:50, y:20); //This passes 50 to x, 20 to y and 100 to z
      }
```

```
//Extension method
    static string UpperCaseFirstLetter(this string value) //It is mandatory to use "this" keyword in the
parameter list in case of an extension method.
    {
      if (value.Length > 0)
      {
        char[] array = value.ToCharArray(); //converts string into char[]
        array[0] = char.ToUpper(array[0]);//capitalizes the first character.
        return new string(array); // returns a new string object ( conversion of char[] to string)
      }
      return value;
    }

    static int Factorial(this int number)
    {
      int fact = 1;

       for (int i = 2; i <= number; i++)
       {
          fact = fact * i;
       }

      return fact;
    }

     //optional parameters or default parameters

    //note: the default parameters are place at the atmost right only.
    static void optionalRecords(int id,string msg="Welcome!!")
     {
       Console.WriteLine("Hello {0} \t {1}", id, msg);
     }

    //Named Parameters

    static void NamedRecords(int x, int y, int z)
    {
      int result = (x * y) / z;
      Console.WriteLine("Added results is {0}", result);
    }
  }
```

}

Output:

```
****Capitalizing first letter of a string****

string: Telugu is a beautiful language

***Printing Factorial***

Factorial of 5 is :120

****Calling Optional Records with default argument***

Hello 100        Welcome!!

****Calling Optional Records with normal arguments***

Hello 100        bye!!!

***Calling Named Arguments***

Added results is 10
Press any key to continue . . .
```