what is .net?

.net is a framework tool---60+ prg language

.net framework--- is a collcetion class libraries

without .net framework there is no .net

9--- microsoft
Remaining---Non -microsft

1.C#.net
2.VB.net
3.c++.net
4.J#.net
5.F#.net
6.Jscript.net
7.windows power shell
8.iron Ruby
9.iron Python

1.C#.net----prg language---coding part for app develoment
2.ASP.net(Active server page)-----Web or serverside Technology
3.SQL-Server---Database design
4.ADO.NET(Active X data object)

Platform: is the combination of OS and CPU(processor)

Platform dependent:--- .net framework--- only windows based os---xp,7,vita
8,10
platform independent:--- mono framework(novell)---- All os

Code execution process in .net:
---

source code----- LC---- Bytecode---OS

Physical Location:
---

root drive:\os folder\assembly folder

c:\windows\assembly folder

.net framework------ Visual studio(IDE)

1.0	------	VS
1.1	------	VS-2003
2.0	------	VS-2005
3.0	------	N/A
3.5	------	VS-2008
4.0	-------	VS-2010
4.5	------	VS-2012
4.6	------	VS-2015

4.7	-------	Vs-2017
4.7.2	------	VS-2019
4.8	------	vs-2022

64 -bit-- windows
2GB RAM
HD---20 GB

1.Console Applications----CUI
2.Windows Forms Applications----GUI
3.web applications---distributed or online app

.net framework components:

1.CLR(common Language Runtime)
2.BCL(Base Class Libraries)

.net framework components:

1.CLR(common Language Runtime)

2.BCL(Base Class Libraries)
or
FCL(Framework Class Library)
or
assmblies

.Exe---main file---which is going to execute invidulay

.exe contains main() method i.e entry point of the program

Dll(Dynamic Link Library)---- supportive file

ddl dont have main method

CLR(common Language Runtime): it will provide runtime environment for .net applications

sub components of CLR:

1.CLS(Common Language specification)
2.CTS(common Type system)---Datatype system
1.valuetype---int a=10;---stack---compiletime
2.referencetype----int b=a;---heap---runtime

3.GC(Garbage collector)---Automatic memory management

optimized engine----1.objects in use
2.idled objects(Garbage)

4.JIT(just in time)

C#.net:
---

is a prg language--coding part

is a case sensitive

is object oriented prg language

is a powerful prg lang

c#.net=c++ + Java + Additional Features

Console Applications:

1.is a standalone or desktop app
2.CUI ---character user interface or command user interface
3.in console application ,console is class
4.console is used to work with input and output stream
5.console class is present system namespce
6.console class is having methods

1.Write()
2.WriteLine()
3.Read()---is used to read single character or next char
return type is integer
Read() will read ascii value of the character

4.ReadLine()---is used to read group of character
returntype is string
ReadLine() will read group of characters

How to accept values at runtime:

to accept values at runtime we use ReadLine()

by default in .net what ever the values accept at runtime i.e string

to convert from string to integer then we use conversion methods

1.convert.ToInt()
2.int.Parse()

convert.Toint() is convert from string to integer and also convert
from null to integer

int.Parse is convert from string to integer but it will not convert
from null to integer

Ex:

```
class Class3
    {
        static void Main(string[] args)
        {
            //string s = "100";
            string s = null;
            Console.WriteLine(Convert.ToInt32(s));
            Console.WriteLine(int.Parse(s));
            Console.Read();
        }
    }
```

Control statements in c#.net:

---

these statements are used to maintain control flow of prgogram execution.

1.conditional statements--- if,if else,nested if,switch

2.iterative statements--- for loop,while loop,do while loop,for each loop

3.transfer statements----break,continue

if: it is used to test specific condition
if the condition is True then only if block will execute
otherwise nothing will execute

syntax:

```
if(condition)
        {
            statements
        }
```

if else:

it is used to test specific condition
if the condition is True then only if block will execute
otherwise else block will execute

else block will execute only if condition is False

syntax:

```
if(condition)
        {
            statements
        }
        else
        {
           statements
        }
```

C# if-else

In C# programming, the if statement is used to test the condition. There are various types of if statements in C#.

- if statement
- if-else statement
- nested if statement

o if-else-if ladder

C# IF Statement
The C# if statement tests the condition. It is executed if condition is true.
Syntax:

```
if(condition){
    //code to be executed
    }
```

```
using System;
public class IfExample
    {
        public static void Main(string[] args)
         {
            int num = 10;
            if (num % 2 == 0)
            {
                Console.WriteLine("It is even number");
            }

         }
    }
```

C# IF-else Statement
The C# if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.
Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

```
using System;
public class IfExample
    {
        public static void Main(string[] args)
        {
            int num = 11;
            if (num % 2 == 0)
                {
                Console.WriteLine("It is even number");
            }
            else
            {
                Console.WriteLine("It is odd number");
            }

    }
}
```

C# If-else Example: with input from user
In this example, we are getting input from the user using Console.ReadLine() method. It returns string. For numeric value, you need to convert it into int using Convert.ToInt32() method.

```
using System;
public class IfExample
```

```
        {
            public static void Main(string[] args)
             {
                    Console.WriteLine("Enter a number:");
                    int num = Convert.ToInt32(Console.ReadLine());

                    if (num % 2 == 0)
                    {
                        Console.WriteLine("It is even number");
                    }
                    else
                {
                    Console.WriteLine("It is odd number");
                    }

                }
    }
```

C# IF-else-if ladder Statement

The C# if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
    if(condition1){
    //code to be executed if condition1 is true
    }else if(condition2){
    //code to be executed if condition2 is true
    }
    else if(condition3){
    //code to be executed if condition3 is true
    }

    else{
    //code to be executed if all the conditions are false
    }
    using System;
    public class IfExample
        {
            public static void Main(string[] args)
        {
                    Console.WriteLine("Enter a number to check grade:");
                    int num = Convert.ToInt32(Console.ReadLine());

            if (num <0 || num >100)
                {
                Console.WriteLine("wrong number");
            }
           else if(num >= 0 && num < 50){
                    Console.WriteLine("Fail");
            }
            else if (num >= 50 && num < 60)
            {
                    Console.WriteLine("D Grade");
                }
                else if (num >= 60 && num < 70)
                {
                    Console.WriteLine("C Grade");
                }
```

```
            else if (num >= 70 && num < 80)
            {
                Console.WriteLine("B Grade");
            }
            else if (num >= 80 && num < 90)
            {
                Console.WriteLine("A Grade");
            }
            else if (num >= 90 && num <= 100)
        {
            Console.WriteLine("A+ Grade");
            }
        }
    }
```

C# switch

The C# switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C#.

Syntax:

```
switch(expression){
case value1:
  //code to be executed;
  break;
case value2:
  //code to be executed;
  break;
.....

default:
  //code to be executed if all cases are not matched;
  break;
}
```

```
using System;
   public class SwitchExample
      {
         public static void Main(string[] args)
         {
                Console.WriteLine("Enter a number:");
            int num = Convert.ToInt32(Console.ReadLine());

            switch (num)
            {
                case 10: Console.WriteLine("It is 10"); break;
                case 20: Console.WriteLine("It is 20"); break;
                case 30: Console.WriteLine("It is 30"); break;
                default: Console.WriteLine("Not 10, 20 or 30");                    break;
            }
         }
      }
```

C# For Loop

The C# for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

The C# for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

```
for(initialization; condition; incr/decr){
//code to be executed
}
```

```
using System;
public class ForExample
    {
        public static void Main(string[] args)
        {
            for(int i=1;i<=10;i++){
                Console.WriteLine(i);
            }
        }
    }
```

C# Nested For Loop

In C#, we can use for loop inside another for loop, it is known as nested for loop. The inner loop is executed fully when outer loop is executed one time. So if outer loop and inner loop are executed 3 times, inner loop will be executed 3 times for each outer loop i.e. total 9 times.
Let's see a simple example of nested for loop in C#.

```
using System;
public class ForExample
    {
        public static void Main(string[] args)
        {
            for(int i=1;i<=3;i++){
                    for(int j=1;j<=3;j++){
                        Console.WriteLine(i+" "+j);
                    }
            }
        }
    }
```

C# Infinite For Loop
If we use double semicolon in for loop, it will be executed infinite times. Let's see a simple example of infinite for loop

```
using System;
public class ForExample
    {
        public static void Main(string[] args)
        {
            for (; ;)
            {
                    Console.WriteLine("Infinitive For Loop");
            }
        }
 }
```

C# While Loop
In C#, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.
Syntax:

```
while(condition){
//code to be executed
}
using System;
public class WhileExample
    {
    public static void Main(string[] args)
      {
            int i=1;
            while(i<=10)
            {
                Console.WriteLine(i);
                i++;
            }
       }
   }
```

C# Nested While Loop Example:
In C#, we can use while loop inside another while loop, it is known as nested while loop. The nested while loop is executed fully when outer loop is executed once.
Let's see a simple example of nested while loop in C# programming language.

```
using System;
public class WhileExample
    {
      public static void Main(string[] args)
      {
            int i=1;
            while(i<=3)
            {
                int j = 1;
                while (j <= 3)
                {
                    Console.WriteLine(i+" "+j);
                    j++;
                }
                i++;
            }
       }
   }
```

C# Infinitive While Loop Example:
We can also create infinite while loop by passing true as the test condition.

```
using System;
public class WhileExample
    {
      public static void Main(string[] args)
      {
            while(true)
            {
                    Console.WriteLine("Infinitive While Loop");
            }
      }
   }
```

C# Do-While Loop

The C# do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
The C# do-while loop is executed at least once because condition is checked after loop body.

```
do{
//code to be executed
}while(condition);
```

```
using System;
public class DoWhileExample
    {
        public static void Main(string[] args)
        {
            int i = 1;

            do{
                Console.WriteLine(i);
                i++;
            } while (i <= 10) ;

        }
    }
```

C# Nested do-while Loop

In C#, if you use do-while loop inside another do-while loop, it is known as nested do-while loop. The nested do-while loop is executed fully for each outer do-while loop.
Let's see a simple example of nested do-while loop in C#.

```
using System;
public class DoWhileExample
    {
        public static void Main(string[] args)
        {
            int i=1;

            do{
                int j = 1;

                do{
                    Console.WriteLine(i+" "+j);
                    j++;
                } while (j <= 3) ;
                i++;
            } while (i <= 3) ;
        }
    }
```

C# Infinitive do-while Loop
In C#, if you pass true in the do-while loop, it will be infinitive do-while loop.
Syntax:

```
do{
//code to be executed
}while(true);
using System;
public class WhileExample
```

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

        do{
            Console.WriteLine("Infinitive do-while Loop");
        } while(true);
    }
}
```

C# Break Statement

The C# break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

Syntax:

```
jump-statement;
break;
```

```
using System;
public class BreakExample
    {
        public static void Main(string[] args)
        {
            for (int i = 1; i <= 10; i++)
            {
                if (i == 5)
                {
                    break;
                }
                Console.WriteLine(i);
            }
        }

    }
```

C# Continue Statement

The C# continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

Syntax:

```
jump-statement;
continue;
```

```
using System;
public class ContinueExample
    {
        public static void Main(string[] args)
        {
            for(int i=1;i<=10;i++){
                if(i==5){
                    continue;
                }
                Console.WriteLine(i);
            }
        }
    }
```

Array:

Array is a collection of similar datatype elements

in c#.net array is userdefined datatype

Always array index starts with 0 and ends with size-1

types of arrays in c#.net:

1.According to physical size----1.single dimension
2.two dimension
2.According to nature of physical size----1.static
2.dynamic
3.According memory size------- 1.Normal---<=64 KB
2.Huge--->64 KB
4.According to specialtype----1.Jagged Array

working with sinle dimension array:
---

syntax: datatype[] Arrayname=new datatype[size];
ex: int[] A=new int[4];

intialize:

A[0]=10;
A[1]=20;
A[2]=30;
A[3]=40;

[or]

int[] A=new int[4]{10,20,30,40};

```
class SDArray
    {
        static void Main(string[] args)
        {
            int[] A = new int[4] { 10, 20, 30, 40 };
            //string[] A = new string[4] { "mohan", "durga", "sai", "raj" };
            Console.WriteLine("Elements of Array:");
            //for(int i=0;i<4;i++)
            //{
            //    Console.Write(A[i] + " ");
            //}
            foreach(int i in A)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine("Length is:" + A.Length);
            Console.WriteLine("Rank is:" + A.Rank);
            Console.Read();

        }
```

workng with two dimension Array:
---

syntax: datatype[,] Arrayname=new datatype[rowsize,columnsize];
ex: int[,] A=new int[3,3];

intialize:

A[0,0]=10;
A[0,1]=20;
A[0,2]=30;
A[1,0]=40;
A[1,1]=50;
A[1,2]=60;
A[2,0]=70;
A[2,1]=80;
A[2,2]=90;

[or]

int[,] A=new int[3,3]{{10,20,30},{40,50,60},{70,80,90}};

```
class TwoDArray
    {
        static void Main(string[] args)
        {
            int[,] A = new int[3, 3] { { 10, 20, 30 }, { 40, 50, 60 }, { 70, 80, 90 } };
            Console.WriteLine("Elements of Array are:");
            for(int r=0;r<3;r++)
            {
                for(int c=0;c<3;c++)
                {
                    Console.Write(A[r, c] + " ");
                }
               Console.WriteLine();
            }
            Console.WriteLine("Length is:" + A.Length);
            Console.WriteLine("Rank is:" + A.Rank);
            Console.Read();
        }
    }
```

working with jagged Array:
---

array of array is called jagged array

an array which contain one more array within it is called jagged Array

in jagged Array no of rows are fixed but columns are not fixed

syntax:
---

datatype[][] Arrayname=new datatyepe[rowsize][];
int [][] A=new int[3][];

intialize:

A[0]=new int[3]{10,20,30};
A[1]=new int[2]{10,20};
A[2]=new int[4]{10,20,30,40};

```
class JaggedArray
    {
        static void Main(string[] args)
        {
            int[][] A = new int[3][];
            A[0] = new int[3] { 10, 20, 30 };
            A[1] = new int[2] { 10, 20 };
            A[2] = new int[4] { 10, 20, 30, 40 };
            Console.WriteLine("Elements of Array are:");
            for(int r=0;r<A.Length;r++)
            {
                for(int c=0;c<A[r].Length;c++)
                {
                    Console.Write(A[r][c] + " ");
                }

                Console.WriteLine();
            }
            Console.WriteLine("Length is:" + A.Length);
            Console.WriteLine("Rank is:" + A.Rank);
            Console.Read();
        }
    }
```

Array properties:

1.Length: is used to represent size of the Array
2.Rank: is used to represent dimensions present in Array

single dimension Array length is size of the array and Rank is 1
two dimension array length is no of rows x no of clomuns and rank is 2
jagged Array length is no of rows and rank is 1

Dynamic Array:
---
an array whose size is not fixed ,array size can be change at runtime
by using resize() method.

Ex:

```
class DynamicArray
    {
        static void Main(string[] args)
        {
            int[] A = new int[8] { 10, 20, 30, 40, 50, 60, 70, 80 };
```

```
            Console.WriteLine("Elements of Array are:");
            foreach(int i in A)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine("\nEnter new size of array:");
            int N = int.Parse(Console.ReadLine());
            Array.Resize(ref A, N);
            Console.WriteLine("Elements of Array A After Resize:");
            foreach (int i in A)
            {
                Console.Write(i + " ");
            }
            Console.Read();
        }
```

Array object method:
---

when we create any array with any name ,the array name is called object.

CopyTo(): this method is array object method.

this method is use to copy the elements from one array to another

letus consider two Arrays A and B

A

```
0  1   2  3  4  5  6  7
10 20 30 40 40 50 60 70
```

B

```
0   1   2   3   4   5   6   7  8  9
15  5  35  25  55  45  75  65 95  85
```

syntax:

SourceArrayname.CopyTo(DestinationArrayname,int index);
Ex: A.CopyTo(B,2);

Array class methods:

Array is a predefined class and it is having builtin methods

1.Copy()
2.Sort()
3.Reverse()
4.clear()
5.Resize()

copy(): is used to copy the elements from one array to another
---

this method is having two syntaxes

syntax1:Array.Copy(SourceArray,int sourceindex,DestinationArray,int destinationindex,int length);
Ex: Array.Copy(A,3,B,6,3);

syntax2: Array.Copy(SourceArray,DestinationArray,int length);
ex: Array.Copy(A,B,5);

Sort(): is used to display array elements in sorting order.
---
by defalut sort() will display array elements in ascending order
to display array elements in descending order,first sort the array and then make it reverse by using reverse method.

syntax1: Array.Sort(Arrayname);
ex: Array.Sort(B);

syntax2: Array.Sort(Arrayname,int index,int length);
Ex: Array.Sort(B,2,5);

Reverse(): is used to display array elements in reverse order
---

syntax1: Array.Reverse(Arrayname);
ex: Array.Reverse(A);

syntax2: Array.Reverse(Arrayname,int index,int length);
Ex: Array.Reverse(B,2,5);

Clear(): is used to clear the array elements
---

syntax: Array.Clear(Arrayname,int index,int length);
Ex: Array.Clear(B,2,5);

Ex:

```
class Arraymethods
    {
        static void Main(string[] args)
        {
            int[] A = new int[7] { 10, 20, 30, 40, 50, 60, 70 };
            int[] B = new int[10] { 15,5,35,25,55,45,75,65,95,85 };
            Console.WriteLine("Elements of Array A:");
            foreach(int i in A)
            {
                Console.Write(i+ " ");
            }
            Console.WriteLine("\nElements of Array B:");
            foreach (int i in B)
            {
                Console.Write(i + " ");
            }
            //A.CopyTo(B, 2);
```

```
            // Array.Copy(A, 3, B, 6, 3);
            //Array.Copy(A, B, 5);
            //Array.Sort(B);
            //Array.Sort(B, 2, 5);
            //Array.Reverse(A);
            // Array.Reverse(A, 2, 5);
            //Array.Clear(B, 2, 5);
            Array.Clear(B, 0, 10); //to clear all elements
            Console.WriteLine("\nElements of Array B    After Clear:");
            foreach (int i in B)
            {
                Console.Write(i + " ");
            }
            Console.Read();
        }
    }
```

How to Accept elements at runtime for single dimension Array:

```
class Arrayruntime
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter lenth of the Array:");
            int n = Convert.ToInt32(Console.ReadLine());
            //creating array based on user length
            int[] A = new int[n];

            for(int i=0;i<n;i++)
            {
                Console.WriteLine("Enter element:");
                A[i] = Convert.ToInt32(Console.ReadLine());
            }
            Console.WriteLine("Elements of Array A are:");
            for(int j=0;j<n;j++)
            {
                Console.Write(A[j] + " ");
            }
            Console.Read();
        }
    }
```

How to Accept elements at runtime for two dimension Array:

```
class TwoDArrayRuntime
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter Row size:");
            int r = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter Column size:");
            int c = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter Array Values:"+r*c);
```

```
            //creating array by using rows and columns from user input
            int[,] A = new int[r, c];
            for(int i=0;i<r;i++)
            {
                for(int j=0;j<c;j++)
                {
                    Console.WriteLine("Enter Array Values:");
                    A[i, j] = Convert.ToInt32(Console.ReadLine());
                }
            }
            Console.WriteLine("Elements of Array A are:");
            for(int i=0;i<r;i++)
            {
                for(int j=0;j<c;j++)
                {
                    Console.Write(A[i, j] + " ");
                }
                Console.WriteLine();
            }
            Console.Read();


        }
    }
```

Object oriented Programming(oop's)

1.security
2.Reusability
3.App enhancement

class: is a collection of member variables and member methods

object: is an instance of class or it is used to represent a class

bird is a class
parrot is an object of bird class

apart from class and object any prg language wants to become an object oriented that prg lang must and should be satisfy the following features

1.Encapsulation
2.Abstraction
3.polymorphism
4.inheritance

Encapsulation: is the process of binding the member variables along with member functions

encapsulation can be achieved with the help of objects only

abstraction: is the process of hiding the implementation but providing service

polymorphism:poly means many and morph means behavior

an operator or method will show different behavior when we change datatypes of arguments or no of arguments.

1.static\compiletime\earlybinding----overloading---refinement techinique
2.dynamic\runtime\latebinding-------overriding----replacement technique

inheritance: is the process of creating new class from existing class

in inheritance process existing class is treated as base class or parent class

in inheritance process newly created class is treated as derived class or child class

in inheritance process child class will get all features from parent class

the main advantage of inheritance is reusability and extensibility

in c#.net class is userdefined type and it contain members

members of class is:

1.datafields
2.functions
3.constructors
4.destructors
5.properties
6.events
7.indexers

classdiagram is represented by rectangle symbol and that is divide into
3 parts

first part is consider as classname
second part is consider as datafields,proerties,indexers
third part is consider as functions,constructors,destructors,events

classname

datafields
properties
indexers

functions
constructors
destructors
events

Datafields: are used store the data
except datafields no other member can store the data

to declare datafields we should follow this syntax:

accessmodifier datatype datafieldname;

public int empid;

Note: default access modifier of the datafield is private

method : is a reusable of piece of code which can be called again and again when it is required.

syntax to create a class:
---

accessmodifier class classname

ex: public class clssample

Note: default accessmodifier of class is internal

syntax to create object to class:
---

syntax: classname objectname=new clssname();
ex: clssample obj1=new clssample();

Accessmodifiers:
---

are used to provide restrction to access variables and methods

c#.net supports 5 types of accessmodifiers

1.private
2.protected
3.internal
4.protected internal
5.public

private: these members are access within the same class,we cant access from outside the class

protected: these members are access within the same class and also in derived class,derived class might be present in same assmbly or in differet assembly.

internal: these members are access in any class within the same assembly these members cant access outside of the assembly.

protected internal: it is the combination of protected and internal these members are access in any class within the same assembly and outside of the assembly in derived class

public: these members are access in any class in any assembly.

How to create class and implement and access:
---

class diagram:
---

clsEmployee

int EmpId;
string EName;
string EAddress;
int EAge;

public void
GetEmpData()
public void
DisplayEmpData()

```
class ClsEmployee
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        public void GetEmpData()
        {
            Console.WriteLine("Enter Emp Details:");
            this.EmpId = Convert.ToInt32(Console.ReadLine());
            this.EName = Console.ReadLine();
            this.EAddress = Console.ReadLine();
            this.EAge = Convert.ToInt32(Console.ReadLine());
        }
        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
// Console.WriteLine("Emp id is:{0}\nEName is:{1}\nEAddress is:{2}\nEAge is:{3}", this.EmpId,
this.EName, this.EAddress, this.EAge);
        }
    }
    class ClsAccess
    {
        static void Main(string[] args)
        {
            ClsEmployee obj1 = new ClsEmployee();
            obj1.GetEmpData();
            obj1.DisplayEmpData();
            Console.Read();
        }
    }
```

in the above example this is a keyword which is used to access current class members

new is a keyword which is used to allot memory to the object

. is an operator which is used to access member variables and member methods

. is a member access operator

in the above example we are trying to access variables and methods with the help of object this process is called encapsulation.

Constructors:

1.constructor is a special method of class
2.constructor is a member method of class
3.constructor is used to declare and intialize the data to the datafields
4.constructor is execute automatically when we create object to the class.
5.constructor name should be same as classname.
6.constructor will keep something ready for the object when it is create to class.
7.constructor do not contain any return type even void also.

types of constructors:

1.instance : 1.Default ----1.userdefined 2.systemdefined
2.parameterized
3.copy
4.private

2.non instance: 1.static constructor

what is the difference b/w method and constructor:

1.method name can be of any name
2.constructor name should be same as classname
3.method will execute only when we call
4.constructor will execute automatically when we create object to class.
5.constructor dont have returntype but methods will have return type
6.inside the method we can write business logic code but inside the constructor we can declare and intialize the variables.
7.we can create any no of objects to the class
8.for every object constructor will execute once but for one object method can be called any no of times.

1.user defined default constructor:
---
is used to intialize the data to the datafields.
this constructor is create by programmer or user
this constructor will not accept any parameters

ClsEmployee1

int EmpId;
string EName;
struing EAddress;
int EAge;

Public ClsEmployee1()
public void DispayEmpData()

2.system defined default constructor:
---

is used to intialize the data to the datafields.
this constructor is not create by programmer or user
this constructor will not accept any parameters

when we execute the program,runtime will check inside the program whether constructor is present or not,if constructor is not present then system will create its own constructor is called system defined default constructor.

this constructor will assign default values to the datafields.

Ex:

```
class ClsEmployee1
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        //public ClsEmployee1()
        //{
        //    this.EmpId = 1234;
        //    this.EName = "ram";
        //    this.EAddress = "hyd";
        //    this.EAge = 35;
        //}
        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
        }
    }
    class ClsUDconstructor
    {
        static void Main(string[] args)
        {
            ClsEmployee1 obj1 = new ClsEmployee1();
            ClsEmployee1 obj2 = new ClsEmployee1();
            obj1.DisplayEmpData();
            obj2.DisplayEmpData();
            Console.Read();
        }
    }
```

Note: in the above example we created two objects ,for any no of objects same employee data is going to display ,to display different employee details for every object then we use parameterized constructor

3.Parameterized constructor:
---

is used to intialize the data to the datafields.
this constructor will accept any parameters

ClsEmployee2

int EmpId;
string EName;
struing EAddress;
int EAge;

Public ClsEmployee2(int Id,string s1,string s2,int Ag)
public void DispayEmpData()

Ex:

```
class ClsEmployee2
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        public ClsEmployee2(int Id,string s1,string s2,int Ag)
        {
            this.EmpId = Id;
            this.EName = s1;
            this.EAddress = s2;
            this.EAge = Ag;
        }
        public ClsEmployee2()
        {
            this.EmpId = 1234;
            this.EName = "ram";
            this.EAddress = "hyd";
            this.EAge = 35;
        }
        public ClsEmployee2(int Id, string s1)
        {
            this.EmpId = Id;
            this.EName = s1;

        }

        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
```

```
        }
    }
    class ClsparameterClass
    {
        static void Main(string[] args)
        {
            ClsEmployee2 obj1 = new ClsEmployee2(123, "sai", "hyd", 34);
            ClsEmployee2 obj2 = new ClsEmployee2(345, "ram", "hyd", 45);
            ClsEmployee2 obj3 = new ClsEmployee2();
            ClsEmployee2 obj4 = new ClsEmployee2(678, "mohan");
            obj1.DisplayEmpData();
            obj2.DisplayEmpData();
            obj3.DisplayEmpData();
            obj4.DisplayEmpData();
            Console.Read();
        }
    }
```

Note: by observing the above class we can say two things
1.a class can contain any no of constructors
2.a constructor can be overload

4.Private constructor:
---

the constructor whose accessmodifier is private is called private constructor
private constructor can not access from outside the class,it can access within the same class.

ClsEmployee3

```
int EmpId;
string EName;
struing EAddress;
int EAge;

private ClsEmployee3()
private void DispayEmpData
```

Ex:

```
class ClsEmployee3
    {

        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        private ClsEmployee3()
        {
            this.EmpId = 1234;
            this.EName = "ram";
            this.EAddress = "hyd";
            this.EAge = 35;
        }
```

```
        private void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
        }
        static void Main(string[] args)
        {
            ClsEmployee3 obj1 = new ClsEmployee3();
            obj1.DisplayEmpData();
            Console.Read();
        }
    }
```

Copy constructor:

1.is used to copy the data from existing object into newly created object

```
ClsEmployee3 obj1=new ClsEmployee3();
ClsEmployee3 obj2=new ClsEmployee3(obj1);
```

ClsEmployee4

```
int EmpId;
string EName;
string EAddress;
int EAge;

public ClsEmployee4()
public ClsEmployee4(ClsEmployee4 objTemp)
public void DisplayEmpData()
```

Ex:

```
class ClsEmployee4
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        public ClsEmployee4()
        {
            this.EmpId = 1234;
            this.EName = "ram";
            this.EAddress = "hyd";
            this.EAge = 35;
        }
        public ClsEmployee4(ClsEmployee4 objTemp)
        {
            this.EmpId = objTemp.EmpId;
            this.EName = objTemp.EName;
            this.EAddress = objTemp.EAddress;
```

```
            this.EAge = objTemp.EAge;
        }
        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
        }
    }
    class Clscopyconstructor
    {
        static void Main(string[] args)
        {
            ClsEmployee4 obj1 = new ClsEmployee4();
            ClsEmployee4 obj2 = new ClsEmployee4(obj1);
            obj1.DisplayEmpData();
            obj2.DisplayEmpData();
            Console.Read();
        }
    }
```

1.static constructor:

1.is used to intialize the data into static datafields
2.using static constructor we cant intialize the data into non static datafields
3.a class contain only one static constructor
4.static constructor do not accept any parameters
5.static constructor do not contain any accessmodifier
6.to make a datafield or constructor as static then we use static keyword

scenario:

```
int i ;              //non static datafield
static int j      // static datafield

public Clssample()      //instance constructor
{
  i=100;    //allowed
  j=100;    //allowed but it looses their static nature
}

static Clssample()    //static constructor
{
  j=100; //allowed
  i=100;    //not allowed
}
```

class diagram:

Clssample

int i;
static int j;

public Clssample()
static Clssample()
public void Display()

Ex:

```
class Clssample
 {
        int i;
        static int j;
        public Clssample()
        {
              i = 100;
        }
        static Clssample()
        {
              j = 100;
        }
        public void Display()
        {
              Console.WriteLine("i=" + i);
              i++;
              Console.WriteLine("j=" + j);
              j++;
        }
 }
 class clsstaticconstructor
 {
        static void Main(string[] args)
        {
              Clssample obj1 = new Clssample();
              obj1.Display();

              Clssample obj2 = new Clssample();
              obj2.Display();

              Clssample obj3 = new Clssample();
              obj3.Display();

              Console.Read();

        }
 }
```

Instance: means any no of objects we created to class each and every object is going to maintan its own reference

obj1---i=100
obj2---i=100

obj3---i=100

Non-Instance: means any no of objects we created to class first object only will maintain its own reference,and remaing objects will follow that.

obj1---j=100
obj2---j=101
obj3---j=102

i variable life time is within the method,once method execution completes i variable loose the data

j variable life time is entire the progarm,j variable will not loose the data until the program execution is finish.

Implementing Inheritance:

1.inheritance is the process of creating new class from existing class
2.in inheritance process new class is treated as Derived or child class
3.in inheritance process existing class is treated as Base or Parent class
4.the main advantage of inheritance is reusaility and Extensibility
5.in nheritance process child class will get all features from parent class

types of inheritance:
1.single
2.multilevel
3.multiple
4.hybrid

assume that there is a company called x
if the company is asking to computerised their Branch Details then the class diagram is

ClsBranch

int Bcode;
string BName;
string BAddress;

public void GetBData()
public void DisplayBData()

later if the company is asking to computerised their Employee Details also
then the class diagram is

ClsEmployee5

int EmpId;
string EName;
string EAddress;
int EAge;

public void GetEmpData()
public void DisplayEmpData()

later if the company is asking to know which Employee is belongs which branch.
based on the above two class diagrams we will not get that information
to know which employee is belongs which branch then we need to do Inheritance.

ClsBranch

int Bcode;
string BName;
string BAddress;

public void GetBData()
public void DisplayBData()
^
|
|
ClsEmployee5

int EmpId;
string EName;
string EAddress;
int EAge;

public void GetEmpData()
public void DisplayEmpData()

in the above class diagram ClsBrancg is a Base or Parent class
clsEmployee5 is a derived or Child class

after doing inheritance we should create object for child class,then we will get
information regarding which employee belngs to which branch

Ex:

```
class ClsBranch
    {
        int BCode;
        string BName;
        string BAddress;
        public void GetBData()
        {
            Console.WriteLine("Enter Branch Details:");
            this.BCode = Convert.ToInt32(Console.ReadLine());
            this.BName = Console.ReadLine();
            this.BAddress = Console.ReadLine();
        }
        public void DisplayBData()
        {
            Console.WriteLine("Branch Details are:");
            Console.WriteLine("Branch Code is:" + this.BCode);
            Console.WriteLine("Branch Name is:" + this.BName);
            Console.WriteLine("Branch Address is:" + this.BAddress);
        }
    }
    class ClsEmployee5:ClsBranch
    {
        int EmpId;
```

```
        string EName;
        string EAddress;
        int EAge;

        public void GetEmpData()
        {
            Console.WriteLine("Enter Emp Details:");
            this.EmpId = Convert.ToInt32(Console.ReadLine());
            this.EName = Console.ReadLine();
            this.EAddress = Console.ReadLine();
            this.EAge = Convert.ToInt32(Console.ReadLine());
        }
        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
        }
    }
    class ClsInheritance
    {
        static void Main(string[] args)
        {
            ClsEmployee5 obj1 = new ClsEmployee5();
            obj1.GetBData();
            obj1.GetEmpData();
            obj1.DisplayBData();
            obj1.DisplayEmpData();

            Console.Read();
        }
    }
```

later if the company is asking to know which Employee is belongs which branch
and they want to computerised their employee salary details also
then we should go for multi level inheritance

```
ClsBranch

int Bcode;
string BName;
string BAddress;

public void GetBData()
public void DisplayBData()
      ^
      |
      |
ClsEmployee5

int EmpId;
string EName;
```

```
string EAddress;
int EAge;

public void GetEmpData()
public void DisplayEmpData()
        ^
        |
        |
Clssalary

double Basic;
double DA;
double HRA,Gross;

public void Getsal()
public void Calculate()
public void Displaysal()
```

Ex:

```
class ClsBranch
    {
        int BCode;
        string BName;
        string BAddress;
        public void GetBData()
        {
            Console.WriteLine("Enter Branch Details:");
            this.BCode = Convert.ToInt32(Console.ReadLine());
            this.BName = Console.ReadLine();
            this.BAddress = Console.ReadLine();
        }
        public void DisplayBData()
        {
            Console.WriteLine("Branch Details are:");
            Console.WriteLine("Branch Code is:" + this.BCode);
            Console.WriteLine("Branch Name is:" + this.BName);
            Console.WriteLine("Branch Address is:" + this.BAddress);
        }
    }
    class ClsEmployee5:ClsBranch
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        public void GetEmpData()
        {
            Console.WriteLine("Enter Emp Details:");
            this.EmpId = Convert.ToInt32(Console.ReadLine());
            this.EName = Console.ReadLine();
            this.EAddress = Console.ReadLine();
            this.EAge = Convert.ToInt32(Console.ReadLine());
```

```
        }
        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
        }
    }
    class Clssalary:ClsEmployee5
    {
        double Basic, DA, HRA, Gross;
        public void Getsal()
        {
            Console.WriteLine("Enter Basic salary:");
            this.Basic = Convert.ToDouble(Console.ReadLine());
        }
        public void Calculate()
        {
            this.DA = this.Basic * 0.03;
            this.HRA = this.Basic * 0.4;
            this.Gross = this.Basic + this.DA + this.HRA;
        }
        public void Displaysal()
        {
            Console.WriteLine("Emp salary Details:");
            Console.WriteLine("Basic is:" + this.Basic);
            Console.WriteLine("DA is:" + this.DA);
            Console.WriteLine("HRA is:" + this.HRA);
            Console.WriteLine("Gross is:" + this.Gross);
        }
    }
    class ClsInheritance
    {
        static void Main(string[] args)
        {
            Clssalary obj1 = new Clssalary();
            obj1.GetBData();
            obj1.GetEmpData();
            obj1.Getsal();
            obj1.Calculate();
            obj1.DisplayBData();
            obj1.DisplayEmpData();
            obj1.Displaysal();

            Console.Read();
        }
    }
```

implementing polymorphism:

1.poly means many and morph means behavior
2.in polymorphism an operator or function will show different behavior according perticular situation.

3.polymorphism can be of two types
  *static\compiletime\earlybinding
  *dynamic\runtime\latebinding
4.static polymorphism can be achieve with the help of overloading
5.dynamic polymorphism can be achieve with the help of overriding

function signature:
---
it includes 4 things

1.functionname
2.accessmodifier
3.type of parameters
4.no of parameters

void is not include in function signature

Function overloading:
---

if a function is having already some work to do ,again if we assign some other work
to the same function then that function is said to be overload.

in general function can be overload in 2 different situations

1.when datatypes of arguments are changed
2.when no of arguments are changed

1.when datatypes of arguments are changed
---

let us consider the function

add(int a,int b)

to call add funtion we need to supply the values like add(10,20)
if we supply the values like add(10.5,20),then we need to overload the function
by changing its datatypes of arguments like add(float a,int b)

1.when no of arguments are changed
---

let us consider the function

add(int a,int b)

to call add funtion we need to supply the values like add(10,20)
if we supply the values like add(10,20,30),then we need to overload the function
by increasing no of arguments like add(int a,int b,int c)

definition of function overloading is:
---
providing new implementation to a function with same name with different signature.

Ex1:

```
class Clsoverload
    {
        public void add(int a,int b)
        {
            Console.WriteLine("sum is:" + (a + b));
        }
        public void add(float a, int b)
        {
            Console.WriteLine("sum is:" + (a + b));
        }
        public void add(int a,int b,int c)
        {
            Console.WriteLine("sum is:" + (a + b + c));
        }
        static void Main(string[] args)
        {
            Clsoverload obj1 = new Clsoverload();
            obj1.add(10, 20);
            obj1.add(10.5f, 30);
            obj1.add(10, 20, 30);
            Console.Read();
        }
    }
```

Ex2:

```
class Clsoverload
    {
        static void add(int a, int b)
        {
            Console.WriteLine("sum is:" + (a + b));
        }
        static void add(float a, int b)
        {
            Console.WriteLine("sum is:" + (a + b));
        }
        static void add(int a, int b, int c)
        {
            Console.WriteLine("sum is:" + (a + b + c));
        }
        static void Main(string[] args)
        {

            add(10, 20);
            add(10.5f, 30);
            add(10, 20, 30);
            Console.Read();
        }
    }
```

Note: all public methods we can access with the help of object
all static methods we can access directly without object

Ex3:

```
class Clsoverload
    {
        static int add(int a, int b)
        {
            return a + b;
        }
        static float add(float a, int b)
        {
            return a + b;
        }
        static int add(int a, int b, int c)
        {
            return a + b + c;
        }
        static void Main(string[] args)
        {
            Console.WriteLine("sum is:" + add(10, 20));
            Console.WriteLine("sum is:" + add(10.5f, 30));
            Console.WriteLine("sum is:" + add(10, 20, 30));
            Console.Read();
        }
    }
```

Ex4:

```
class Clsoverload
    {
        public int add(int a, int b)
        {
            return a + b;
        }
        public float add(float a, int b)
        {
            return a + b;
        }
        public int add(int a, int b, int c)
        {
            return a + b + c;
        }
    }
    class Clsoverload1
    {
        static void Main(string[] args)
        {
            Clsoverload obj1 = new Clsoverload();
            Console.WriteLine("sum is:" +obj1. add(10, 20));
            Console.WriteLine("sum is:" +obj1. add(10.5f, 30));
            Console.WriteLine("sum is:" +obj1. add(10, 20, 30));
            Console.Read();

        }
    }
```

Function overriding:

---

providing new implementation to a function with same name with same signature is called function overriding.

in general what ever the members are available in parent class that are by default available to the child class through inheritance.

but if child class is not satisfy with the parent class implementation then child class can redefine the parent class functions, this process is called overriding.

Ex:

```
class clsParent
    {
        public void property()
        {
            Console.WriteLine("Gold+Cash+Lands+Power");
        }
        public void Car()
        {
            Console.WriteLine("Alto car");
        }
    }
    class clsChild:clsParent
    {
        public void Car()
        {
            base.Car();
            Console.WriteLine("Benz Car");
        }
    }
    class Clsoverride
    {
        static void Main(string[] args)
        {
            clsChild obj1 = new clsChild();
            obj1.property();
            obj1.Car();
            Console.Read();
        }
    }
```

Note: if child class wants to access parent class method also,then inside the child class method we use base keyword.

the difference between overloading and overriding:

1.providing new implementation to a function with same name with different signature is called function overloading.

2.providing new implementation to a function with same name with same signature is called function overriding.

3.overloading is a refinement technique
4.overriding is a replacement technique

5.overloading can be used to achieve static polymorphism
6.overriding can be used to achieve dynamic polymorphism

Note: virtual and ovverride keywords are optional

Sealed class:

1.a class from which it is not possible to create or derive new class is called sealed class.
2.the main advantage of sealed class is to avoid further inehritance.
3.to make a class as sealed we use sealed keyword.
4.sealed class is completely opposite to abstract class.
5.in general abstract class is contain virtual and abstract functions
6.but sealed class do not contain virtual and abstract functions
7.sealed class may contain override functions.
8.abstract class is always be a base class
9.but sealed class can never be a base class.
10.sealed class is always bottom most of the class in inheritance heirarchy

ex:

```
class A
  ^
  |
class B ---sealed class
  ^
  |
class C
```

if B class is sealed class then from B class we cant extend c class.

Ex:

```
class A
    {
        public void F1()
        {
            Console.WriteLine("F1 function");
        }
    }
  sealed   class B:A
    {
        public void F2()
        {
            Console.WriteLine("F2 function");
        }
    }
    class C: B      //'C': cannot derive from sealed type 'B'
    {

    }
```

Partial class:

1.a class code can be split or divide into two or more parts is called partial class.
2.to make a class as partial we use partial keyword.
3.partial class will allow multiple developers to develop the code.
4.using partial class faster rate of application development is possible.
5.in general partial class can be used in very large scale projects
6.while implementing partial class all multiple developers should maintain the same class name.
7.all partial classes should be present in same assembly or application.
8.we can create object for partial class for accessing the methods.

part1:

```
partial   class Class8
    {
        //x developer
        int id;
        string name;
        public void getempdata()
        {
            Console.WriteLine("Enter EmpId:");
            this.id = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter EmpName:");
            this.name =Console.ReadLine();

        }
    }
```

part2:

```
partial   class Class8
    {
        //y developer
        public void displayempdata()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.id);
            Console.WriteLine("EName is:" + this.name);
        }
    }
```

Main program:

```
class ClsPartial
    {
        static void Main(string[] args)
        {
            Class8 obj1 = new Class8();
            obj1.getempdata();
            obj1.displayempdata();
            Console.Read();
        }
    }
```

Abstract Function and Abstract class:

Abstract Function:

1.a function which do not contain any implementation,which contain only declaration or signature is called abstract function.
2.to make a function as abstract we use abstract keyword.
3.abstract function should terminate.
4.abstract function should be override.

Ex:

```
public abstract void GetEmpData();
```

Abstract Class:

1.a class which contain one or more abstract functions is called abstract class.
2.to make a class as abstract we use abstract keyword
3.abstract class is also contain non abstract functions.
4.abstract class is contain all members of class.
5.abstract class can not instantiated directly,so that we need to create or derive new class from abstract class to provide functionality to its abstract functions.
6.partial implementation of the class is abstract class.
7.by default abstract class functions are not public and abstract
8.we can not create object for abstract class.

Ex:

```
abstract class classname
```

Note:
---

if a class is abstract then that class must and should contain atleast one abstract function.

if a class is non abstract class then that class will not allow any abstract functions.

how many no of abstract functions available in abstract class,that many abstract functions should implement in the derived classes.

Ex:

```
abstract class Vehicle
    {
        public abstract void wheels();
        public abstract void color();
        public void engine()
        {
            Console.WriteLine("Bs-VI Engine");
        }
    }
    class Car:Vehicle
    {
        public override void wheels()
        {
            Console.WriteLine("car is having 4 wheels");
```

```
        }
        public override void color()
        {
            Console.WriteLine("Car color is Red");
        }
    }
    class Bike:Vehicle
    {
        public override void wheels()
        {
            Console.WriteLine("Bike is having 2 wheels");
        }
        public override void color()
        {
            Console.WriteLine("Bike color is Black");
        }
    }
    class Auto : Vehicle
    {
        public override void wheels()
        {
            Console.WriteLine("Auto is having 3 wheels");
        }
        public override void color()
        {
            Console.WriteLine("Auto color is Yellow");
        }
    }
    class ClsAbstract
    {
        static void Main(string[] args)
        {
            Car c = new Car();
            Bike b = new Bike();
            Auto a = new Auto();
            c.wheels();
            c.engine();
            c.color();
            b.wheels();
            b.engine();
            b.color();
            a.wheels();
            a.engine();
            c.color();
            Console.Read();
        }
    }
```

Interface:

1.a class which contain all abstract functions is called interface.
2.to create an interface we use interface keyword
3.interface do not contain non abstract functions.
4.interface contains:
    abstract functions,properties,indexers,events

5.interface do not contain :
non abstract functions,datafields,constructors,destructors
6.interface can not instantiated directly,so that we need to create or derive
new class from interface to provide functionality to its abstract functions.
7.no implementation of the class is called interface.
8.by default interface functions are public and abstract.
9.we can not create object for interface.
10.the main purpose of interface is to implement multiple inheritance.

Ex: interface interfacename

```
interface Vehicle1
    {
        void wheels();
        void color();
    }
    class Car1:Vehicle1
    {
        public    void wheels()
        {
            Console.WriteLine("car is having 4 wheels");
        }
        public void color()
        {
            Console.WriteLine("car color is Red");
        }
    }
    class ClsInterface
    {
        static void Main(string[] args)
        {
            Car1 c1 = new Car1();
            c1.wheels();
            c1.color();
            Console.Read();
        }
    }
```

Multiple Inheritance:
---

is the process of creating new class from two or more base classes.

```
class A       class B
              ^
              |
         Class C
```

c#.net do not support multiple inheritance with base level two classes

let us consider the example

```
class A
{
void f1()
```

```
}
class B
{
void f1()
}
class C:A,B
{
code...
}

C obj1=new C();
obj1.f1();
```

when we call obj1.f1() ,runtime will get confuse to call f1() of class A or class B
this situation is known as ambiguity situation.

to avoid this, to implement multiple inheritance there should be a base level
one class or no class or there should be a base level atleast one interface or more interfaces.

```
class A       interface B
              ^
              |
         class C


interface A       interface B
                  ^
                  |
               class C
```

when we implement multiple inheritance at base level one class and one interface
first we have to mention classname then after interfacename.

class C:A,B ---right

class C:B,A ---wrong

Ex:

```
class ClsBranch1
      {
            int BCode;
            string BName;
            string BAddress;
            public void GetBData()
            {
                  Console.WriteLine("Enter Branch Details:");
                  this.BCode = Convert.ToInt32(Console.ReadLine());
                  this.BName = Console.ReadLine();
                  this.BAddress = Console.ReadLine();
            }
            public void DisplayBData()
            {
                  Console.WriteLine("Branch Details are:");
                  Console.WriteLine("Branch Code is:" + this.BCode);
```

```
            Console.WriteLine("Branch Name is:" + this.BName);
            Console.WriteLine("Branch Address is:" + this.BAddress);
        }
    }
    interface Clsemployee6
    {
        void GetEmpData();
        void DisplayEmpData();
    }
    class ClsManager : ClsBranch1, Clsemployee6
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        public void GetEmpData()
        {
            Console.WriteLine("Enter Emp Details:");
            this.EmpId = Convert.ToInt32(Console.ReadLine());
            this.EName = Console.ReadLine();
            this.EAddress = Console.ReadLine();
            this.EAge = Convert.ToInt32(Console.ReadLine());
        }
        public void DisplayEmpData()
        {
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + this.EmpId);
            Console.WriteLine("EName is:" + this.EName);
            Console.WriteLine("EAddress is:" + this.EAddress);
            Console.WriteLine("EAge is:" + this.EAge);
        }

    }
    class Clsmultiple
    {
        static void Main(string[] args)
        {
            ClsManager obj1 = new ClsManager();
            obj1.GetBData();
            obj1.GetEmpData();
            obj1.DisplayBData();
            obj1.DisplayEmpData();
            Console.Read();
        }
    }
```

Ex:

```
interface Interface1
    {
        void f1();
    }
    interface Interface2
    {
        void f2();
```

```
}
class C1:Interface1,Interface2
{
    public void f1()
    {
        Console.WriteLine("Hello");
    }
    public void f2()
    {
        Console.WriteLine("welcome");
    }
}
class Clsmultiple1
{
    static void Main(string[] args)
    {
        C1 obj1 = new C1();
        obj1.f1();
        obj1.f2();
        Console.Read();
    }
}
```

both interfaces should have same function i.e f1() and we need to provide different result for both interface functions then we use a concept called explicit interface implementation.

Ex: Explicit interface implementation

```
interface Interface1
{
    void f1();
}
interface Interface2
{
    void f1();
}
class C1:Interface1,Interface2
{
    void Interface1.f1()
    {
        Console.WriteLine("Hello");
    }
    void Interface2.f1()
    {
        Console.WriteLine("welcome");
    }

}
class Clsmultiple1
{
    static void Main(string[] args)
    {
        C1 obj1 = new C1();
        //explicit interface implementation
```

```
            Interface1 obj2 = (Interface1)obj1;
            Interface2 obj3 = (Interface2)obj1;

            obj2.f1();
            obj3.f1();

            Console.Read();
        }
    }
```

Hybrid Inheritance:
---
it is the combination of multiple and multivel inheritance

```
class A       interface B
        ^
        |
      class C
        ^
        |
      class D
```

Ex:

```
class classA
    {
        public void f1()
        {
            Console.WriteLine("f1 function of class A");
        }
    }
    interface interfaceB
    {
        void f2();

    }
    class classC:classA, interfaceB
    {
        public void f2()
        {
            Console.WriteLine("f2 function of class C");
        }
    }
    class classD:classC
    {
        public void f3()
        {
            Console.WriteLine("f3 function of class D");
        }
    }
    class clsHybrid
    {
        static void Main(string[] args)
        {
            classD obj1 = new classD();
            obj1.f1();
```

```
            obj1.f2();
            obj1.f3();
            Console.Read();
        }
    }
```

Abstraction:
---
is the process of hiding the implementation but providing service or result
in c#.net abstraction can be achieve with the help of properties

Properties:

1.properties are a member of class.
2.properties are used write the data into datafields and read the from datafields.
3.properties will never store the data but these are transfer the data.
4.for writing and reading purpose properties will use two accessors or methods
   1. set accessor
   2. get accessor

set accessor is used to write the data into datafields

syntax:

set{datafieldname=value;}

it contain default variable called value, when we supply the data to datafields
it will go and store in value variable.

get accessor is ised to read the data from datafields

syntax:

get{return datafieldname;}

c#.net supports 3 types of properties

1.Readonly property
2.Writeonly property
3.Readwrite property

1.Readonly property: it can be used to read the data from datafields
using this property we cant write the data into datafields.
it contain only one accessor i.e get

syntax:
Accessmodifier datatype propertyname
{
get{return datafieldname;}
}

2.Writeonly property: it can be used to write the data into datafields

using this property we cant read the data from datafields.

it contain only one accessor i.e set

syntax:
Accessmodifier datatype propertyname
{
set{datafieldname=value;}
}

3.Readwrite property: it cane be used for reading and writing

it contain only two accessors i.e get and set

syntax:
Accessmodifier datatype propertyname
{
set{datafieldname=value;}
get{return datafieldname;}
}

Note: propertyname and datafieldname should be different
property datatype and datafield datatype should be same

properties will provide abstraction to the datafields
properties will provide security to the datafields

Note: writing the code inside set or get is strictly prohibited

program for read write property
---

clsEmployee7

int EmpId;
string EName;
string EAddress;
int EAge;
public int PEmpId;
public string PEName;
public string PEAddress;
public int PEAge;

Ex:

```
class ClsEmployee7
    {
        int EmpId;
        string EName;
        string EAddress;
        int EAge;

        public int PEmpId
        {
            set { EmpId = value; }
```

```
            get { return EmpId; }
        }
        public string PEName
        {
            set { EName = value; }
            get { return EName; }
        }
        public string PEAddress
        {
            set { EAddress = value; }
            get { return EAddress; }
        }
        public int PEAge
        {
            set { EAge = value; }
            get { return EAge; }
        }
    }

 class ClsProperty
    {
        static void Main(string[] args)
        {
            ClsEmployee7 obj1 = new ClsEmployee7();
            Console.WriteLine("Enter Emp Details:");
            obj1.PEmpId = Convert.ToInt32(Console.ReadLine());
            obj1.PEName = Console.ReadLine();
            obj1.PEAddress = Console.ReadLine();
            obj1.PEAge = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Emp Details are:");
            Console.WriteLine("EmpId is:" + obj1.PEmpId);
            Console.WriteLine("EName is:" + obj1.PEName);
            Console.WriteLine("EAddress is:" + obj1.PEAddress);
            Console.WriteLine("EAge is:" + obj1.PEAge);
            Console.Read();

        }
    }
```

program for read only and write only property:

ClsArithmetic

```
int Num1;
int Num2;
int Result;
public int PNum1
public int PNum2
public int PResult

public void Add()
public void Sub()
public void Mul()
public void Div()
```

Ex:

```
class ClsArithmetic
    {
        int Num1;
        int Num2;
        int Result;
        public int PNum1
        {
            set { Num1 = value; }
        }
        public int PNum2
        {
            set { Num2 = value; }
        }
        public int PResult
        {
            get { return Result; }
        }
        public void Add()
        {
            Result = Num1 + Num2;
        }
        public void Sub()
        {
            Result = Num1 - Num2;
        }
        public void Mul()
        {
            Result = Num1 * Num2;
        }
        public void Div()
        {
            Result = Num1 / Num2;
        }

    }


class ClsProperty1
    {
        static void Main(string[] args)
        {
            ClsArithmetic obj1 = new ClsArithmetic();
            Console.WriteLine("Enter Two Numbers:");
            obj1.PNum1 = Convert.ToInt32(Console.ReadLine());
            obj1.PNum2 = Convert.ToInt32(Console.ReadLine());
            obj1.Add();
            Console.WriteLine("Sum is:" + obj1.PResult);

            obj1.Sub();
            Console.WriteLine("Sub is:" + obj1.PResult);

            obj1.Mul();
            Console.WriteLine("Mul is:" + obj1.PResult);
```

```
            obj1.Div();
            Console.WriteLine("Div is:" + obj1.PResult);

            Console.Read();


        }
    }
```

in the above example clsArithmetic is having implementation,we are hide the implementaion but providing result to the clsProerty1 class. this process is called abstraction.

Generics:

1.Generics are general datatypes like int,float,str,double
2.Generics are present in system.collections.generic namespace
3.Generics are used to avoid function overloading when datatypes of arguments are changed.
4.using Generics we will write generic methods and Generic classes
5.to work with generics we use two things.
  1.place holder represented by <>
  2.type parameter
6.Always type parameter should be enclosed within the place holder like <typeparameter>

Non-Generic method:

```
public void display(string s)
{
Console.WriteLine("value is:"+s);
}
public void display(int s)
{
Console.WriteLine("value is:"+s);
}
public void display(float s)
{
Console.WriteLine("value is:"+s);
}

display("sai");
display(100);
display(23.4);
```

Generic Method:

```
public void display<G>(G s)
{
Console.WriteLine("Value is:"+s);
}

display<string>("sai");
display<int>(100);
```

```
display<double>(34.5);
```

Ex with Generic method:

```
class ClsGeneric
    {
        static void display<G>(G s)
        {
            Console.WriteLine("value is:" + s);
        }
        static void Main(string[] args)
        {
            display<string>("sai");
            display<int>(100);
            display<double>(34.6);
            Console.Read();
        }
    }
```

Ex with Generic class:

```
class ClsGeneric1<G>
    {
        public void display(G s)
        {
            Console.WriteLine("value is:" + s);
        }
    }
    class ClsGeneric2
    {
        static void Main(string[] args)
        {
            ClsGeneric1<string> objs = new ClsGeneric1<string>();
            ClsGeneric1<int> obji = new ClsGeneric1<int>();
            ClsGeneric1<double> objd = new ClsGeneric1<double>();
            objs.display("mohan");
            obji.display(100);
            objd.display(34.5);
            Console.Read();
        }
    }
```

Ex with generic class by passing multile datatype values:

```
    class ClsGeneric1<G1,G2>
    {
        public void display(G1 s1,G2 s2)
        {
            Console.WriteLine("values are:" + s1+","+s2+"");
        }
    }
    class ClsGeneric2
    {
        static void Main(string[] args)
```

```
        {
            ClsGeneric1<string,int> objs = new ClsGeneric1<string,int>();
            ClsGeneric1<int,float> obji = new ClsGeneric1<int,float>();
            ClsGeneric1<double,string> objd = new ClsGeneric1<double,string>();
            objs.display("mohan",10);
            obji.display(100,56.7f);
            objd.display(34.5,"sai");
            Console.Read();
        }
    }
```

Ex with generic method with multiple datatype values:

```
 class ClsGeneric
    {
        static void display<G1,G2>(G1 s1,G2 s2)
        {
            Console.WriteLine("values are:{0},{1}",s1,s2);
        }
        static void Main(string[] args)
        {
            display<string,int>("sai",40);
            display<int,double>(100,56.7);
            display<double,string>(34.6,"ram");
            Console.Read();
        }
    }
```

Poninters:

1.pointer is a variable which strores address of another variable.
2.the code that has been written using pointer is called unsafe code.
3.for pointer code CLR do not provide any security.
4.unsafe code can write using unsafe keyword within the unsafe block or within the unsafe class.
5.unsafe code can only be compile and run with unsafe options.

synta to declare pointer variable:
---

datatype*variablename
int*x --- x is a pointer variable which stores adress of an integer value.

two operators are required here
1.* --- represent pointer
2.& ---- address operator

Ex with pointer using unsafe block

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

```
        {
            int a = 100;
            unsafe
            {
                int* x = &a;
                Console.WriteLine("Value of a is:" + *x);
                Console.WriteLine("Address of a is:" + (int)x);
            }
            Console.Read();
        }
    }
```

Ex with pointer using unsafe class

```
unsafe class Clspointer1
    {
        static void square(int*x)
        {
            Console.WriteLine("Square is:" + *x * *x);
        }
        static void Cube(int* x)
        {
            Console.WriteLine("Cube is:" + *x * *x**x);
        }
        static void Main(string[] args)
        {
            int a = 9;
            square(&a);
            Cube(&a);
            Console.Read();
        }
    }
```

Note: to run pointer code ---goto solution explorere---double click on properties
---click on build from LHS(left hand side)---activate the check box----
---allow unsafe code---then save---and then run.

Delegates:

1.delegates are used to represent or refer one or more functions.
2.delegates are userdefined types in c#.net
3.delegates in c#.net are similar to function pointers in c++
4.delegate is not a member of class,but it is similar to a class.
5.to consume any delegate,we need to create object for delegate
6.a delegate is a type that references a method
7.once a delegate is assigned to a method it behaves exctly a method.
8.delegates are backbone for events.
9.in delegates multiple methods can be called on single event.
10.in delegates we can also use generics
11.delegates are object oriented and secure.

types of delegates:

1.single cast delegate
2.multi cast delegate

*a delegate that represent only one function is called single cast delegate.
*a delegate that represent more than one function is called multi cast delegate.

to work with delegates we should follow these steps:

*creating a delegate
*instantiating a delegate
*invoking a delegate

let us consider a function

```
public void add(int a,int b)
{
code...
}
```

to refer this function,if we want to use delegate we use the above steps like.

step1: creating a delegate
syntax: accessmodifier delegate returntype delegatename(argumentslist);
ex: public delegate void mydelegate(int a,int b);

note: when we create a delegate ,accessmodifier,returntype,no of arguments and their datatypes of the delegate must and should be same as accessmodifier ,returntype,no of argumnents and their datatype of the function that we want to refer.

step2: instantiating a delegate
syntax:delegatename objectname=new delegatename(targetfunctionname);
ex: mydelegate objd=new mydelegate(add);

step3: invoking a delegate
syntax: delegateobjectname(argumentvalues);
ex: objd(10,20);

Ex: single cast delegate

```
//program to represent a function using delegate within same class
    class Clsdelegate
    {
        static void display(string s)
        {
            Console.WriteLine("value is:" + s);
        }
        //creating a delegate
          delegate void mydelegate(string s);
        static void Main(string[] args)
        {
            //instantiating a delegate
            mydelegate objd = new mydelegate(display);
            //invoking a delegate
            objd("sai");
```

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

Ex:

```
class Clsdelegate1
    {
        //program to represent a function using delegate outside of the class
        public void add(int a, int b)
        {
            Console.WriteLine("sum is:" + (a + b));
        }
    }
    //creating a delegate
    public delegate void sampledelegate(int a, int b);
    class clsdelegate2
    {
        static void Main(string[] args)
        {
            Clsdelegate1 obj1 = new Clsdelegate1();
            //instantiating a delegate
            sampledelegate objd = new sampledelegate(obj1.add);
            //invking a delegate
            objd(10, 20);
            Console.Read();
        }
    }
```

Ex: multi cast delegate

```
  class ClsArithmetic1
    {
        public void add(int x,int y)
        {
            Console.WriteLine("sum is:" + (x + y));
        }
        public void sub(int x, int y)
        {
            Console.WriteLine("sub is:" + (x - y));
        }
        public void mul(int x, int y)
        {
            Console.WriteLine("mul is:" + (x * y));
        }
        public void div(int x, int y)
        {
            Console.WriteLine("div is:" + (x / y));
        }
    }
    public delegate void mcdelegate(int x, int y);
    class clsdelegate3
    {
        static void Main(string[] args)
```

```
        {
            ClsArithmetic1 obj1 = new ClsArithmetic1();
            mcdelegate objd = new mcdelegate(obj1.add);

            objd += obj1.sub;
            objd += obj1.mul;
            objd += obj1.div;

            objd(5, 3);

            objd -= obj1.mul;
            objd -= obj1.div;

            objd(8, 7);

            Console.Read();
        }
    }
```

Note:

to add reference of more functions to a delegate object, use += operator like
objd+=obj1.add

to delete reference of any functions from a delegate object,use -= operator like
objd-=obj1.add

Exception Handling Mechanism:
---

in general when we type .net code and execute ,we may get three different types of errors

1.syntactical errors
2.compilation errors
3.runtime errors

*syntactical errors occur when miss "" ; () or writing wrong spelling to the syntax.
*syntactical errors do not cause any harm to the program execution
*these errors can be easily identify and rectify by the programmer at the time of writing code.

*the errors which will raise at the time of compilation is called compilation errors
*compilation errors do not cause any harm to the program execution
*these errors can be easily identify and rectify by the programmer at the time of compilation.

*runtime error is nothing but exception.
*these errors will cause abnormal termination of program execution.
*these errors cant be identity and rectify by the programmer at runtime

Def: an unwanted or unexpected event will disturb normal flow of program execution is called exception.

*to handle the exception we use Exception Handling mechanism.
*Exception Handling mechanism is only for runtime errors.
*Exception Handling mechanism is not the process of repairing the error,it is the process of to show alternative way of program execution successfully.

there are two types of exception handling mechanism:

1.logical implementation
2.try catch implementation

most of the cases programmers will give first preference to logical implementation
if any Exception is not possible to handle using logical statements then we should go for try catch.

in C#.net every exception is a class

the subclasses of Exception:

1.DivideByZeroException
2.FormatException
3.outofmemoryException
4.indexoutofrangeException
5.FilenotfoundException
6.memberAccessException
7.methodAccessException

Note: the super class of All sub class Exceptions is "Exception"

Ex with logical implementation:

```
class ClsException
    {
        static void Main(string[] args)
        {
            int a, b, c;
            Console.WriteLine("Enter two Numbers:");
            a = int.Parse(Console.ReadLine());
            b = int.Parse(Console.ReadLine());
            c = a / b;
            Console.WriteLine("Quotient is:" + c);
            Console.Read();
        }
    }
```

in the above example when we give second number as 0 then we will get DividebyzeroException.

Ex:

```
class ClsException
    {
        static void Main(string[] args)
        {
            int a, b, c;
            Console.WriteLine("Enter two Numbers:");
            a = int.Parse(Console.ReadLine());
            b = int.Parse(Console.ReadLine());
            if(b==0)
            {
```

```
                Console.WriteLine("second num cant be zero");
            }
            else
            {
                c = a / b;
                Console.WriteLine("Quotient is:" + c);
            }
            Console.Read();
        }
    }
```

in the above example when we give input as characters then we will get FormatException.

Note: all exceptions may not possible to handle by using logical statements

to handle any type Exceptions then we use try catch implementation.

try catch implementation:
---
syntax:

```
try
{
  statements
}
catch
{
  statements
}
finally
{
  statements
}
```

in general inside the try block we should keep risky code
inside the catch block we should keep handling code
inside the finally block we should keep mandatory messages

Note : inlcuding finaly block is optional ,if we include it must be execute.

case1: if there is an exception inside the try block
        try block is ignore,then catch block will execute and then finally will execute
case2: if there is no exception inside the try block
        try block is Execute,catch block will ignore and then finally will execute

Ex with try catch:

```
class ClsException1
    {
        static void Main(string[] args)
        {
            try
            {
                int a, b, c;
```

```
                Console.WriteLine("Enter two Numbers:");
                a = int.Parse(Console.ReadLine());
                b = int.Parse(Console.ReadLine());
                c = a / b;
                Console.WriteLine("Quotient is:" + c);
            }
            catch
            {
                Console.WriteLine("error is occured");
            }
            finally
            {
                Console.WriteLine("Code is excuted");
            }
            Console.Read();
        }
    }
```

Note: in the above example if any exception is raised we are showing same error message i.e error is occured.

to show proper exception error message then we use specific catch block.

catch blocks can be of two types

1.generic catch block

2.specific catch block

1.generic catch block:
---

a catch block without any exception class is called generic catch block

ex:

```
catch
{
Console.WriteLine("error is occured");
}
```

1.specific catch block:
---

a catch block with exception class is called specific catch block

ex:

```
catch(Exception ex)
{
Console.WriteLine(ex.message);
}
```

Ex:

```
    class ClsException1
```

```
{
    static void Main(string[] args)
    {
        try
        {
            int a, b, c;
            Console.WriteLine("Enter two Numbers:");
            a = int.Parse(Console.ReadLine());
            b = int.Parse(Console.ReadLine());
            c = a / b;
            Console.WriteLine("Quotient is:" + c);
        }
        //catch(Exception ex)
        //{
        //      Console.WriteLine(ex.Message);
        //}
        //instead of superclass we can multiple catch blocks
        catch(DivideByZeroException ex)
        {
            Console.WriteLine(ex.Message);
        }
        catch (FormatException ex)
        {
            Console.WriteLine(ex.Message);
        }

        finally
        {
            Console.WriteLine("Code is excuted");
        }
        Console.Read();
    }
}
```

Note: if existing Exception classes are not meet our requirement then we can write our own exception classes is called customized Exceptions or userdefined exceptions

to create our own exception class ,that class should be inherited from predefined Exception class ie. Exception.

Ex: to raise our own execeptions

to raise our own Exceptions inside the program we use throw keyword

```
class TooooldException : Exception
{
    public TooooldException(string message) : base(message)
    {

    }
}
class TooyoungException : Exception
{
    public TooyoungException(string message) : base(message)
    {
```

```
        }
    }
    class clsException3
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your Age:");
            int Age = Convert.ToInt32(Console.ReadLine());
            if(Age>=60)
            {
                throw new TooldException("You are too old");
            }
            else if(Age<=16)
            {
              throw new TooyoungException("you are too young");
            }
            else
            {
                Console.WriteLine("you are eligible to take policy");
            }
            Console.Read();
        }
    }
```

Ex: to raise our own execeptions and Handle:

```
 class TooldException : Exception
    {
        public TooldException(string message) : base(message)
        {

        }
    }
    class TooyoungException : Exception
    {
        public TooyoungException(string message) : base(message)
        {

        }
    }
    class clsException3
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Enter your Age:");
                int Age = Convert.ToInt32(Console.ReadLine());
                if (Age >= 60)
                {
                    throw new TooldException("You are too old");
                }
                else if (Age <= 16)
                {
```

```
                throw new TooyoungException("you are too young");
            }
            else
            {
                Console.WriteLine("you are eligible to take policy");
            }
        }
        catch(Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        finally
        {
            Console.WriteLine("code is executed");
        }
        Console.Read();
    }
}
```

Boxing and Unboxing:

1.Boxing is the process of converting a variable from
valuetype to reference type
2.Unboxing is the process of converting a variable from
reference type to value type

Difference b/w Boxing and Unboxing:

Boxing supports 2 types:
*Implicit Boxing
*Explicit Boxing

Unboxing supports 1 type:
*Explicit Unboxing

Boxing is 20 times costlier than normal intialiation
Unboxing is 4 times costlier than normal intialization

Ex:

```
class ClsBoxing
    {
        static void Main(string[] args)
        {
            int i = 10; //normal intialization
            object O = i; //implicit Boxing
            object x = (object)i;    //explicit Boxing
            int j = (int)O; //Explicit Unboxing

            Console.WriteLine("Value of i is:" + i);
            Console.WriteLine("Value of O after Implicit Boxing is:" + O);
            Console.WriteLine("Value of X after Explicit Boxing is:" + x);
            Console.WriteLine("Value of j after Implicit UnBoxing is:" + j);
            Console.Read();
        }
```

```
    }
```

Boxing is 20 times costlier than normal intialiation,
because whenever boxing is done
the following tasks will be performed internally.

1.Runtime will search for the respective data within the stack
2.A copy of this value is made into heap.
3.reference to this copy is mantained from object variable.

Unboxing is 4 times costlier than normal intialization,
because whenever Unboxing is done
the following tasks will be performed internally.

1.object reference value is searched within the heap
2.a copy of this is made into stack

Note: as boxing and unboxing costlier than normal intialization,
so that Boxing and Unboxing should be avoid in maximum situations.
use only if necessary,but we should use boxing and unboxing in case
other operations are costlier than boxing and Unboxing.

Enums in C#:

1.Enum stands for enumeration.
2.enum is a special class that represent a group of constants(not changeble/Read only)
3.to create enum we use enum keyword and separate the enum values by using comma.
4.use enums when we have values that you know are not going to change,like month,days,
colors.
5.enum can be used in any prg language to define a set of constant values.
6.enums consumes less memory space because enums are valuetype and that allocated
on stack.
7.enums are represented as string but processed as an integer.
8.the default enumeration type is int.

Ex1:

```
class ClsEnum
    {
        enum level
        {
            Low,
            Medium,
            High
        }
        static void Main(string[] args)
        {
            level myvalue = level.High;
            Console.WriteLine("Level is:" + myvalue);
            Console.Read();
        }

    }
```

Ex2: Enum values

by default,the first item of enum has value 0 and the second item has the value 1 and so on

to get integer value from an item,we must explicitly convert the item to an int.

```
class ClsEnum1
    {
        enum months
        {
            january,
            february,
            march,
            april,
            may,
            june,
            july
        }
        static void Main(string[] args)
        {
            //months m = months.january;
            //Console.WriteLine("the month is:" + m);
            int m = (int)months.march;
            Console.WriteLine(m);
            Console.Read();
        }
    }
```

Ex3: enum in switch statement

enums are often used in switch statements to check for corresponding values.

```
class ClsEnum
    {
        enum level
        {
            Low,
            Medium,
            High
        }
        static void Main(string[] args)
        {
            level myvalue = level.Medium;
            switch(myvalue)
            {
                case level.Low:
                    Console.WriteLine("Low Level");
                    break;
                case level.Medium:
                    Console.WriteLine("Medium Level");
                    break;
                case level.High:
                    Console.WriteLine("High Level");
                    break;
```

```
            }
            Console.Read();
        }

    }
```

Note: enums helps you to define,assign and maintain constants in a program in an effective way.

Indexers:
---

*Indexer is a special type of property that allows a class to be accessed like an array for its private collection.
*if user will define an indexer for a class,then the class will behave like an virtual array.
*there are 3 ways to access datafields into other class or outside the class

1. we can make datafields are public.
2. we can use properties
3. we can use indexers

*indexers are almost similar to the properties.
*the main difference b/w indexers and properties is that,
the accessors of the indexers will take parameters.

syntax:

```
[access_modifier] [return_type] this [arguments_list]
{
get
{
   //get block code
}
set
{
  //set block code
}
}
```

Ex:

```
 class ClsEmp
     {
         int eno;
         string ename;
         double salary;
         string eaddress;

         public ClsEmp(int eno,string ename,double salary,string eaddress)
         {
             this.eno = eno;
             this.ename = ename;
             this.salary = salary;
```

```
            this.eaddress = eaddress;
        }
        public object this[int index]
        {
            get
            {
                if (index == 0)
                    return eno;
                else if (index == 1)
                    return ename;
                else if (index == 2)
                    return salary;
                else if (index == 3)
                    return eaddress;
                return null;

            }
            set
            {
                if (index == 0)
                    eno = (int)value;
                else if (index == 1)
                    ename = (string)value;
                else if (index == 2)
                    salary = (double)value;
                else if (index == 3)
                    eaddress = (string)value;
            }
        }
    }

 class ClsTestEmployee
    {
        static void Main(string[] args)
        {
            ClsEmp obj1 = new ClsEmp(101, "sai", 45000.00, "hyd");
            Console.WriteLine("Eno is:" + obj1[0]);
            Console.WriteLine("Ename is:" + obj1[1]);
            Console.WriteLine("Esal is:" + obj1[2]);
            Console.WriteLine("Eaddress is:" + obj1[3]);

            obj1[1] = "mohan";
            obj1[3] = "sr nagar";
            Console.WriteLine("Eno is:" + obj1[0]);
            Console.WriteLine("Ename is:" + obj1[1]);
            Console.WriteLine("Esal is:" + obj1[2]);
            Console.WriteLine("Eaddress is:" + obj1[3]);
            Console.Read();

        }
    }
```

Partial methods:

---

1. a partial class can contain partial methods.
2. a partial method is created by using partial keyword.

3. a partial method declaration consists of two parts.
    1. the definition(only the method signature)
    2. the implementation

these may be are seperate part of the partial class or in the same part.

4.the implemenation of the partial methods are optional,
if you dont provide implementation then the compiler will remove signature.
5.partial methods are private by default. it raise compilation error when we use any accesmodifier including private.
6.declaration and implementation of the partial methods at a time will give compilation error.
7.a partial method reurntype must be void,if we include any other returntype then it will give compilation error.

Ex:

```
partial    class ClassPartial
     {
          partial void m1();

            partial void m1()
          {
               Console.WriteLine("partial method m1");
          }

          public void m2()
          {
               Console.WriteLine("non partial method m2");
               m1();
          }


     }

class Clssamplepartial
     {
          static void Main(string[] args)
          {
               ClassPartial obj1 = new ClassPartial();
               obj1.m2();
               Console.Read();
          }
     }
```

Designing .net components or Assemblies or User Defined Class Libraries or DLL's:
---

what is .net component:

1.a component is reusable piece of code and it is in the form of DLL.
2.once component is designed ,it can be reused from any kind of application like in console application,windows forms application etc.
3.once a component is designed in any prg language of .net that can be reused from any other prg languages of .net.
4. .net component will provide language interoperability.

5.to create .net component we use class library template.
6. End user or client will never interact with DLL directly.
7.Always end user will interact with some application and the application will interact with component or DLL.
8.the main advantage to create .net compnent or dll is security and reusability.

User------> Application -------> Component or DLL

steps to create .net component or DLL:
---

goto visual studio 2019----Click on New project---choose language as C#---platform as windows---application type as Library----choose ClassLibrary(.NET Framework)---click on Next----give projectname is ---- MyClassLibrary1---click on Create.

```
namespace MyClassLibrary1
{
    public class Class1
    {
        int Num1, Num2, Result;
        public int PNum1
        {
            set { Num1 = value; }
        }
        public int PNum2
        {
            set { Num2 = value; }
        }
        public int PResult
        {
            get { return Result; }
        }
        public void Add()
        {
            Result = Num1 + Num2;
        }
        public void Sub()
        {
            Result = Num1 - Num2;
        }
        public void Mul()
        {
            Result = Num1 * Num2;
        }
        public void Div()
        {
            Result = Num1 / Num2;
        }
    }
}
```

*Add one more class to the MyclassLibrary1 i.e class2

```
namespace MyClassLibrary1
{
```

```
    public class Class2
     {
          public int square(int x)
          {
               return x * x;
          }
          public int cube(int x)
          {
               return x * square(x);
          }

     }
}
```

goto build menu----click on build myclasslibrary1---build will be sucess----this will be create MyClassLibrary1.dll--- it is known as .net component.

*it can be reused in any application.

steps to consume dll into console application:

1.create a new console application
2. Add reference to the dll

steps to add reference to the dll:
---
click on view menu----click on objectbrowser----Expand Browse-----click on EditCustom component settings----click on Browse tab----choose location where your dll ----select MyclassLibrary1.dll---click on Add ---click on ok----select MyclassLibrary1----click on add to references in project solution explorer--this will add to your application as a reference.

then write the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MyClassLibrary1;

namespace ConsoleApp1
{
     class Program
     {
          static void Main(string[] args)
          {
               Class1 obj1 = new Class1();
               Class2 obj2 = new Class2();
               Console.WriteLine("Enter Two Numbers:");
               obj1.PNum1 = Convert.ToInt32(Console.ReadLine());
               obj1.PNum2 = Convert.ToInt32(Console.ReadLine());
               obj1.Add();
               Console.WriteLine("Sum is:" + obj1.PResult);
               obj1.Sub();
```

```
            Console.WriteLine("Sub is:" + obj1.PResult);
            obj1.Mul();
            Console.WriteLine("Mul is:" + obj1.PResult);
            obj1.Div();
            Console.WriteLine("Div is:" + obj1.PResult);
            Console.WriteLine("Enter Any Number:");
            int n = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Square value is:" + obj2.square(n));
            Console.WriteLine("Cube value is:" + obj2.cube(n));
            Console.Read();
        }
    }
}
```