

23/12/13

## C# .Net

→ What is software app?

Software application is collection of software prgms which are developed by some by some software technologies to fulfill the end user requirement.

Ex: MS-office, Gmail.com

→ What is Software prgm?

It is collection of logical instructions to the computer

→ Ex: printf, scanf. (Logical instructions.)

Example for Software Technologies

.Net ; Java, PHP

→ Example for End users.

all gmail users, all MS. office users

Types of Software applications:

→ Software app are classified into various types based on application behaviour.

→ Some of them are

1. Windows applications → Ex: MS-office

2. Web application → Ex: Gmail.com, facebook etc.

24/12/13

What is .Net

→ .Net is a software technology which was introduced by Microsoft in the year 2002.

→ Microsoft is a leading organisation, which was established by Bill Gates in U.S.A.

- .Net is a framework technology which is integrated with multiple technologies like below
1. .Net windows technology (Windows Forms)
  2. web technology (ASP.NET)
  3. web service technology (WCF)
  4. Mobile technology
- 26
- ↓  
windows communication  
foundation

→ .Net is integrated with multiple technologies, due to that reason using single .Net a programmer can develop various type of software applications like below

1. windows applications
2. web applications
3. web services
4. mobile applications
5. windows Service - - -

→ .Net is supporting for multiple programming languages which are called as .Net languages they are

1. C# .Net
2. VB .Net
3. VC++ .Net
4. VJ# .Net
5. VF# .Net
6. C++ .Net - - -

→ To develop any type of application using .Net, we require one .Net technology and one .Net language for ex if you want to develop a windows application we have to use windows forms and C# .Net.

→ If you want to develop web application we have to use ASP.NET and C# .NET.

→ If we want to develop web service, we have to use WCF and C# .NET.

## 26/12/13 History of .Net (8) Challenges faced by Microsoft to introduce .Net :-

→ Before .NET technology i.e., from 1990-98 under Microsoft family we have 3 popular technologies. They are

1. VB (Visual Basic)
2. VC++ (Visual C++)
3. ASP (Active Server Pages)

→ These technologies are having some of the drawbacks as below:

### 1. Platform dependency:-

→ These technologies will work only on windows operating system.

### 2. Not supporting to develop multi type of app's

→ Using VB and VC++ we can develop only windows applications.

→ Using ASP we can develop only web applications.

→ At the same time in the yr Sun Microsystems has introduced a technology called JAVA.

→ JAVA is a platform Independent technology and using JAVA we can develop various types of software applications.

→ Because of above advantages in JAVA, Microsoft existing technology clients are attracting towards JAVA.

→ To overcome this Microsoft has announced about .NET first time in the year 1998 in the meeting called PDC (Professional Developers Conference).

→ In PDC Microsoft has announced about .Net features like below.

1. .Net will be platform Independent technology.
2. .Net is supporting to develop various type of applications.
3. .Net ~~will~~ will be supporting for multi languages.

→ From 1998 - 2002 Microsoft R&D team has designed .Net technology and finally they have released first version of .Net i.e., .Net 1.0 in the year 2002.

27/12/13

### Versions of .Net technology:

I <sup>st</sup> release —	.Net framework 1.0 → 2002
II <sup>nd</sup> release —	1.1 → 2003
III <sup>rd</sup> release —	2.0 → 2005
IV <sup>th</sup> release —	3.0 → 2006
V <sup>th</sup> release —	3.5 → 2008
VI <sup>th</sup> release —	4.0 → 2010
VII <sup>th</sup> release —	4.5 → 2012

### What is .Net framework?

→ .Net framework is an important integral component in software, when we install .Net software this component will be installed directly.

### Definition:

→ .Net framework is a common platform for developing various types of applications by using .Net technologies and .Net languages.

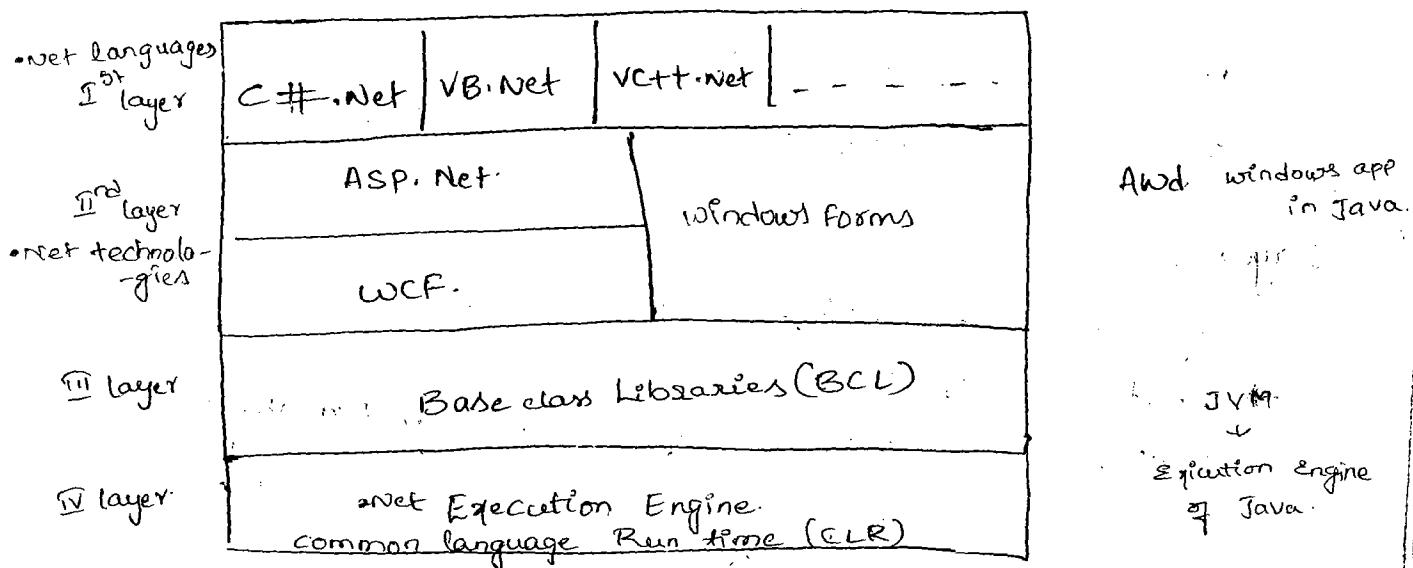
- .Net framework is integrated with 4 items as below.
1. .Net languages
  2. .Net technologies
  3. .Net base class libraries
  4. .Net execution Engine.

28/12/13

### .Net frame work architecture diagram:

- Net frame work will be divided into 4 layers

#### .Net Framework Architecture



→ Layer 1 is representing .Net languages.

→ To develop any type of application by using .Net, .Net programmers require one .Net language i.e., C#-Net (or) VB-.Net.

→ Layer 2 is representing .Net Technologies.

#### 1. Windows Forms :-

~~& ASP-.Net~~ → it is a .Net windows technology

→ Using windows forms we can develop a windows application

→ windows application can be

→ Using .Net if we want to develop windows forms

We have to use <sup>.Net</sup> windows technology and C#-Net.

### d. ASP.NET :-

- It is a .Net web technology.
- By using ASP.NET we can develop a web application.
- In .Net if we want to develop a web application, we have to use .Net web technology called ASP.NET and .Net language called C# .Net language.

### 3. WCF :-

- WCF stands for "Windows Communication Foundation".
- It is a .Net web service technology.

→ In .Net if we want to develop a web service by using .Net we have to use WCF and C# .Net.

30/12/13

### → Layer 3 :-

- Layer 3 is representing Base class libraries
- ★ What is a class?
- Class is a collection of variables and methods
- Classes are of types.
  - a. Pre defined class
  - b. User defined class

#### Pre defined class :

A class which is defined by Microsoft programmer can be called as pre defined class.

#### User defined class :

A class which is defined by the .Net programmer can be called as user defined class.

What is

→ What is class Libraries?

→ Class library is a collection of classes

→ Class libraries are of types.

1. Predefined class libraries

2. User defined class libraries.

### Pre-defined class libraries:

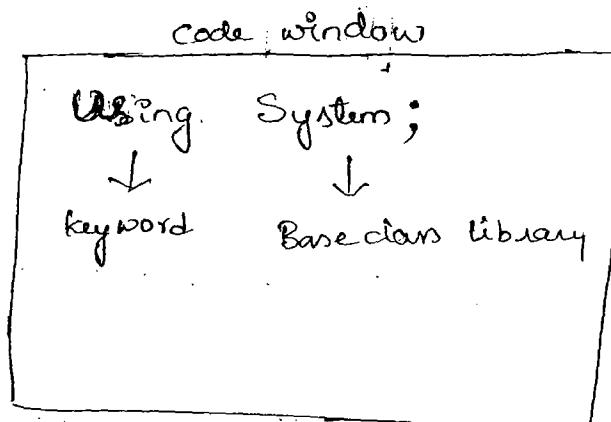
→ A class library which is defined by the Microsoft programmer, can be called as pre-defined class libraries.

→ Pre-defined class libraries contain collection of pre-defined classes.

→ Pre-defined class libraries are called as Base class libraries, which are provided by the Microsoft and used by the .NET programmers.

### How to use Base class library:

With the help of 'Using' keyword like below.



### User defined class libraries:

→ A class library which is defined by .NET programmer can be called as user defined class library.

Note: Base class libraries can be called as .NET framework class libraries.

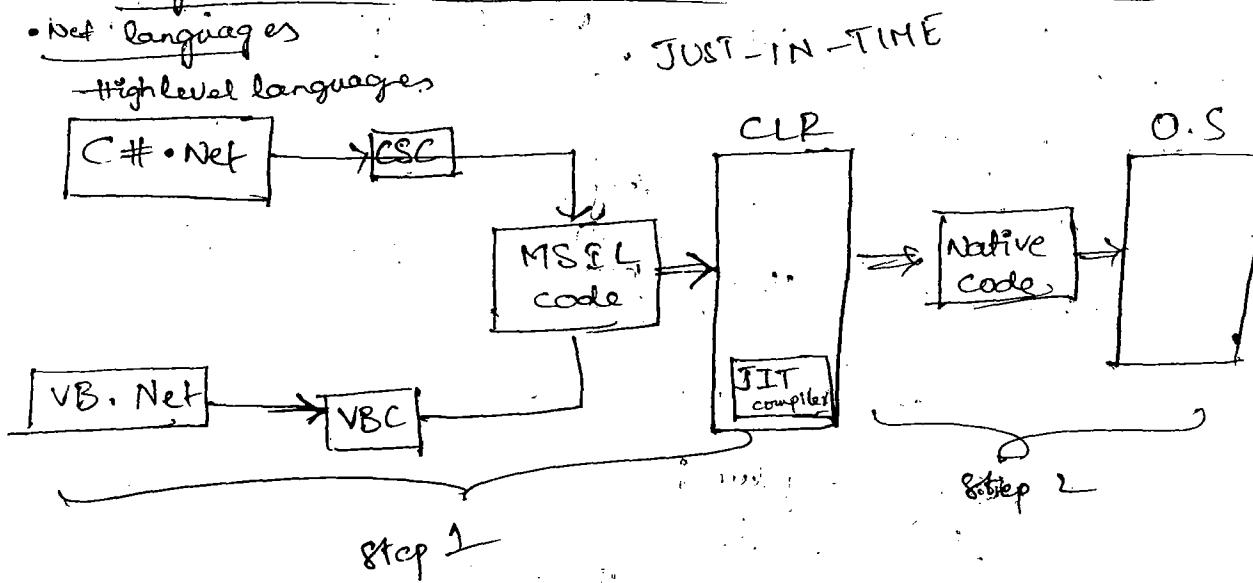
## Layer 4:

- Layer 4 is representing .Net execution Engine i.e., CLR
- CLR is a common execution engine for all .Net languages.
- Every .Net language developed application has to execute with the help of CLR.

31/12/13

## Execution process of .Net applications:-

### Diagram for .Net application Execution process



.Net application process will be divided into 2 steps

### Step 1 :

\* Language compilers will be converting high level language code (~~C#.Net compiler~~<sup>Code</sup>) in to MSIL code (Microsoft Intermediate Language).

\* Becoz .Net execution engine can understand only MSIL code.

## Step 2:

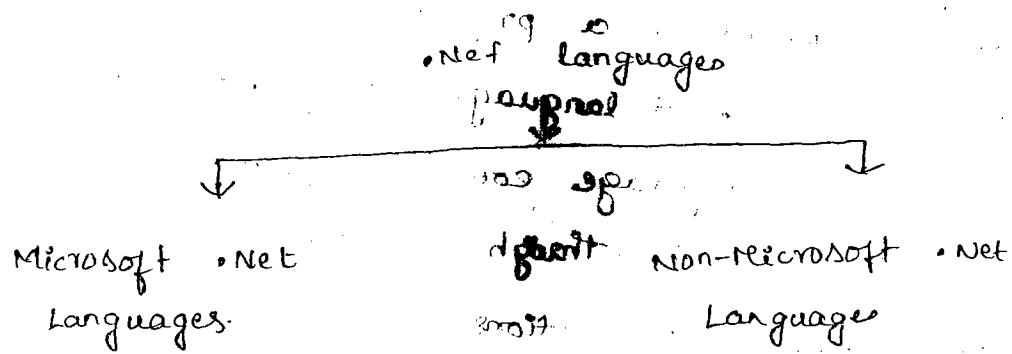
→ JIT compiler [Just In Time] will convert MSIL code to Native code becoz operating Systems can understand only native code.

01/01/14

.Net Languages: (18)

→ .Net is supporting for multiple programming languages which are called as .Net languages (18) .Net supporting languages

→ These .Net languages are classified into two types like below



### 1. Microsoft .Net Languages:-

→ Microsoft .Net languages are the languages which are introduced by Microsoft organisation.

Ex: C# .Net

VB .Net

VC++ .Net

C++ .Net

→ Microsoft .Net languages compilers are inbuilt by the microsoft software . So the .Net programme is not in need to install them again.

## d. Non - Microsoft .Net languages:

- Non - Microsoft .Net languages are the languages introduced by other than Microsoft organisation. These are Non - Microsoft .Net languages (2) third party .Net languages (3) Vendor .Net languages.

Ex : cobol .Net

Pascal .Net

Perl .Net

Delphi .Net

- These language compilers are not inbuilt by Microsoft.
- That means whenever a programmer wants to use non - Microsoft .Net <sup>Brand</sup> languages, programmer has to purchase the language compiler from the concerned organisation website through online.

21/14

## Types of Software applications:

- Software applications are classified into various types based on application behaviour.
1. Console application
  2. Windows application.
  3. Web application.
  4. web Service
  5. Windows Service
  6. Mobile application

## 1. Console application:

- Console application is a single user application
- To consume this application end user doesn't require internet connectivity
- No user interface i.e., input and output will be within the command prompt window.
- Will not have real time applications but, to learn C# .Net we are going to develop these applications.
- To develop console applications by using .Net we have to use console technology and C# .Net

## 2. Windows applications:

- Windows applications can be called as desktop applications
  - (a) Standard alone applications (b) GUI based application (graphical user interface), (c) windows applications & so on ..
- Single user applications
- To consume windows application end user doesn't require internet connectivity.
- It has User Interface will be there which is called as windows forms.
- Application should be installed at end user machine.
- When we will develop windows applications?

Ans: whenever an application should be available for single user at a time we will go for windows applications

Ex: Father business accounts applications, Mobile Shoppee day to day transactions, college library management, Medical store stock application, MS office.

- To develop a windows application by using a .Net, we have to use .Net windows technology called windows forms & .Net language called C# .Net.

### 3.13. web application:

- Web applications are web enabled applications.
- Web applications are multi user applications.
- Web application will have a user interface which can be called as web page.
- Web application should be installed at remote machine called web server.
- ⇒ When we will go for web application?

Ans: whenever an application should be available for single or multiple users at ~~that~~ <sup>a</sup> time we will go for web application

Ex: www.facebook.com

www.gmail.com

- To develop a web application by using .Net we have to use ASP.NET and a language called C# .NET

### Comparison between Console application and windows application:

Console application	windows application
→ Single user	→ Single user
→ No user interface	→ Interface is there which is called windows forms
→ No internet connectivity	→ No internet connectivity
→ No real time application	→ We will develop Real time applications.
→ Light weight applications	→ Heavy weight application.

## Comparision b/w windows and web applications :

### windows Application

- Single user
- User Interface is there called windows forms
- Internet connectivity is not needed
- To develop we have to use windows forms and C# • Net
- whenever the application should be available for single user we go for windows application.  
Ex: MS office

### web application

- Multi User
- User interface is there called web page
- Internet connectivity is required
- To develop we have to use ASP.NET and C# • Net.
- whenever the application should be available for multi user we go for web application.  
Ex: Gmail.com  
facebook.com

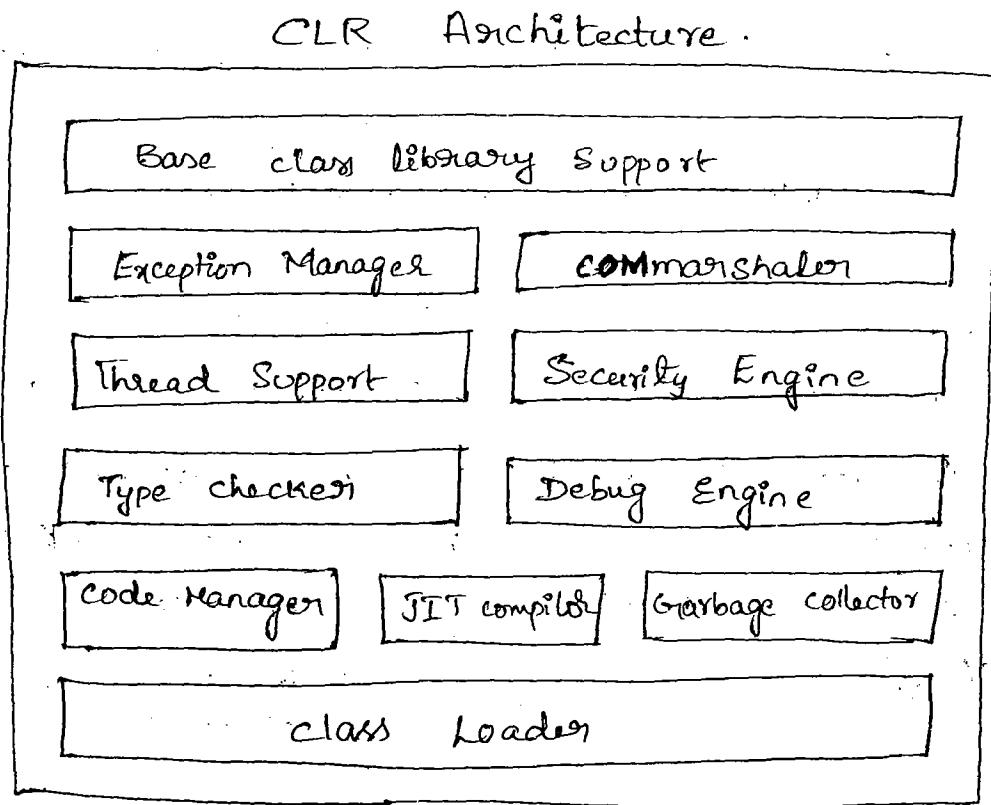
6/1/14

## CLR Architecture :

- CLR is part of .Net frame work.
- .Net frame work is a part of .Net Software
- The main responsibility of CLR is executing .Net applications, it will provide some other services such as code management, memory management, exception handling support, multi threading support, security, debugging and so on..

## Diagram for CLR architecture :-

→ CLR architecture is representing the responsibilities of the CLR in .Net.



### Class Loader :

→ It loads classes from application to CLR one by one according to execution requirement.

### Code Manager :

→ which manages the code during execution of the application.

### JIT Compiler : (Just-In-Time)

→ which converts MSIL code to Native code.

### Garbage collector :

→ In .Net, memory management is performing by garbage collector.

→ To perform this memory management garbage collector

is doing 2 duties.

a) allocating memory :-

→ when an object is creating by the application garbage collector will allocate memory for that particular object within the heap data structure.

b) Deallocating memory :

→ when an object is not using by the application garbage collector will recognize particular object as unused object and it will destroy that unused object.

→ To allocate memory and to de-allocate memory programmer need not to write single line of code because which is internal process. Due to that reason a .NET memory management is called as "Automatic memory management"

Ques. → Exception Manager:

→ Errors are two types:

1. Compile time error
2. Run time error

Compile time errors:

→ An error which is occurred at the time of compilation is called compile time error.

Ex: Syntax errors.

Run time error:

→ An error which is occurred at the time of executing program is called as run time error.

Ex:

→ Run time error will occur because of invalid input or improper logic.

→ what is an exception?

Ans: An exception can be defined as runtime error.

→ what is exception handling?

Ans: Exception handling is a mechanism to handle runtime errors by using try, catch and finally blocks.

→ Purpose of Exception handling?

Ans: To avoid the abnormal termination when the error is occurred

→ Exception manager will provide exception handling facility for .Net applications.

### Thread Support:

→ what is a thread?

Ans: Thread is an independent execution path, it able to run simultaneously with other execution paths.

→ what is multi threading?

Ans: Implementing multiple execution paths simultaneously is called as multi threading.

→ what is purpose of multi threading?

Ans: To improve the performance of application will go for multithreading.

→ Thread Support is providing multithreading facility for .Net application.

### Type checker:

→ which provides strict data type checking.

### COM Marshaler:-

- COM Stands for "Component Object Model."
- COM is a Microsoft old traditional technology.
- Using COM we can develop reusable components (DLL files).
- COM marshaler allows .Net applications to consume COM components.

08/11/14.

### Security Engine:

- which provides Security to .Net applications.

### Debug Engine:

- which provides various types of debugging facilities to .Net application.

### Base class library Support:

- which provides predefined class library support to .Net applications.

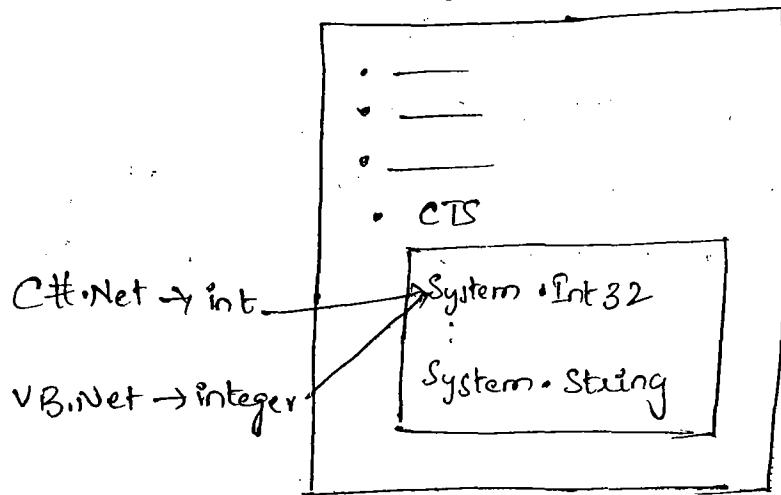
### CLS (Common Language Specification):-

- CLS is a set of common language standards defined by Microsoft for all .Net languages.
- whenever a programming language wants to recognize as .Net language it has to follow CLS Standards.

### CTS (common Type System) :-

- CTS is a subset of CLS.
- CTS is a set of common base datatypes defined by Microsoft for all .Net languages.
- Every .Net language data types will have base data types in CTS.
- Ex: `System.int32` is the CTS type which is base type for VB.NET integer and C# .NET int like below.

CLS



C#

## 9/11/14 C# .Net

→ what is C# .Net ?

Ans It is a .Net language.

→ why C# .Net ?

Ans whenever we want to develop any type of Software application .Net programmers require one .Net language.

→ what type of programming language is C# .Net ?

Ans: C# .Net is a object oriented <sup>Programming</sup> language

### Features of C# .Net :

→ C# .Net is a case sensitive language

Ex: Int a;

A = 10; // error

→ It supports block level programming.

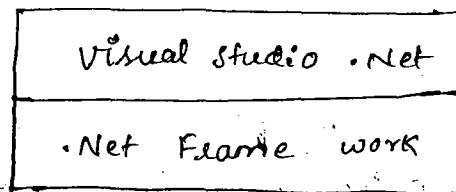
Ex: for block :-

```
{  
    St 1;  
    St 2;  
}
```

→ every statement should end with Semicolon.

### Visual Studio .Net :

→ Visual Studio .Net is a predefined tool providing by Microsoft with .Net software.



→ Visual Studio .Net is providing development environment for .Net Programmers to develop various types of applications by using .Net technologies and .Net languages.

→ .Net framework is a runtime environment which is required to run the .Net application.

### Order to install .Net Software

- 1) Install SQL Server
- 2) Configure & install IIS
- 3) Install .Net Software

How to create a new console application?

→ Start → programs → Microsoft Visual Studio 2010 → Visual Studio 2010

→ It will open visual studio IDE [Integrating development environment]

→ Click on New Project

→ It will open new project window

→ here Select language visual c# and type of application as console application and rename it as myconsole application, location as D:\. Then click OK.

→ Console application development environment will have mainly 2 windows.

1) Solution Explorer window

2) Code window

### 1) Solution Explorer window:

→ This window will display all the project related files.

→ By default it will come with one file called program.cs

Program.cs

→ Program.cs is a C#.Net class file.

→ Every C#.Net class file extension will be .cs and VB.NET class file extension will be .vb.

What is class file?

- Ans A class file can contain collection of classes, by default every class file will come with a single class.
- Program.cs file will come with a single class called Program and program class will come with a special method called Main method.

### Main() :-

- Main is a special type of function, console application execution starts by main(), controlling by main() and will stop by main().
- This main() is also controlling by .Net execution engine called CLR.

### 2) Code Window :-

- This window will display the selected class file code.

### Structure of Program.cs file:

```
keyword      ↗ Base class
using System;    ↗ library
key word ↙
namespace MyconsoleApplication;    ↗ project name
{           ↗
    keyword      ↗ class name
    ↗
    class Program
    {
        keyword      ↗ method name
        keyword      ↗
        static void Main(String[], args)
        {
            ↗
            ↗
        }
    }
}
```

1) write a console programme to print welcome msg.

```
using System;
using namespace System::ConsoleApplication;
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("welcome to C# .Net");
            Console.ReadLine();
        }
    }
}
```

Compiling the program :-

F6

O/P

Welcome to C# .Net

Executing the program:

F5

- write line is a predefined method.
- Read line is a predefined method.
- console is predefined class
- class Program is user defined class
- static void Main is user defined. but it is not be change.  
becoz it
- System → it is a base class library.

2) write a console prgm to print two welcome msgs?

O/P

welcome to C#.Net  
welcome to console Appl.

Prgm.

Static void Main( )

{

    Console.WriteLine ("welcome to C#.Net");  
    Console.WriteLine ("welcome to console Appl");  
    Console.ReadLine();

}

3) write a console prgm to print 3 welcome msgs

O/P

welcome to C#.Net  
welcome to console Appl  
welcome to satya.

Prgm

Static void Main( )

{

    Console.WriteLine ("Welcome to C#.Net");  
    Console.WriteLine ("welcome to console Appl");  
    Console.WriteLine ("welcome to Satya");  
    Console.ReadLine();

}

## System :

→ It is a base class library.

## Console :

→ Console is a predefined class which is a part of System Base class Library

## WriteLine() :-

→ Write Line is a predefined member method of Console class.

→ Write Line method will print the given value, after printing the value, it will move the cursor to the next line.

ReadLine() :- It holds the command Prompt window until we

4) Write a prgm to print one string value and one numerical value.

Prgm

Static Void Main()

{

```
    console.WriteLine("rama");
    console.WriteLine("25");
    console.ReadLine();
```

}

O/P
rama
25
-

## Variable:

Variable is representing a value which can be changed.

### Syntax to declare a variable:

<data type> <variable name> = <value>;

Eg: int a = 10;

String s = "Sathya";

- 5) Write a Console program to declare 1 int variable and 1 string variable?

### Prgm

```
Static void Main()
```

```
{
```

O/P

```
String s = "rama";
```

rama  
25

```
int age = 25;
```

```
String name = "rama";
```

```
Console.WriteLine(name);
```

```
Console.WriteLine(age);
```

```
Console.ReadLine();
```

```
}
```

7

9

- 6) Write Example to display variable values with userdefined strings?

O/P

Your name is : rama

Your age is : 25

### Prgm

```
Static void Main()
```

```
{
```

```
String a = "Your name is : rama";
```

```
String b = "Your age is : 25";
```

Prgm

Static Void Main()

{

int age = 25;

String name = "Rama";

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

Console.WriteLine("Your Age is: " + age);

Console.ReadLine();

}

7) Write a Console program to display employee info. i.e., employee no, employee name, salary?

Prgm

Static void Main()

{

int enumber = 9981263440;

int esalary = 50000;

String name = "Amrutha";

Console.WriteLine("Employee name is: " + name);

Console.WriteLine("Employee number is: " + enumber);

Console.WriteLine("Employee Salary is: " + esalary);

Console.ReadLine();

}

O/P

Employee name is: Amrutha.

Employee number is: 9981263440.

Employee Salary is: 50000

8) Write a Console program to display student info i.e.,  
Student ID, name, location?

Prgm      Static Void Main()

{

    int stuid = 123;

    String stuname = "Pallavi";

    String stulocation = "Hyderabad";

    Console.WriteLine("Student ID is: " + stuid);

    Console.WriteLine("Student name is: " + stuname);

    Console.WriteLine("Student location is: " + stulocation);

    Console.ReadLine();

}

O/p

Student ID is: 123

Student name is: Pallavi

Student location is: Hyderabad

Shortcut to open visual studio.net

Start → Run → type as devENV → OK.

2/1/14

7) Write a console prgm to store student Marks. They are

M1, M2, M3 & calculate total marks and average marks & finally display

O/P

Total marks are : 190

Average marks are : 63

Prgm

```
void Main()
{
    int m1 = 70;
    int m2 = 65;
    int m3 = 55;
    int totmarks = m1+m2+m3;
    double avg = totmarks/3;
    Console.WriteLine("Total marks are :" + totmarks);
    Console.WriteLine("Average marks are :" + avg);
    Console.ReadLine();
}
```

10) Write a console prgm to represent customer info  
i.e., customer ID, customer name, customer location, customer  
ph no, customer mail ID & display.

11) Write a console prgm to store product information  
i.e., product ID, product name, product company, product  
per cent & display?

HW

10) Prgm void Main()
{
 int cusID = 123;
 String cusname = "sandhy";
 String custloc = "kukatpally";
 double cusnum = 9848032211;
 String cusemail = "sandhya@gmail.com";
 Console.WriteLine("customer ID is :" + cusID);
 Console.WriteLine("customer name is :" + cusname)

```
console.WriteLine("Customer location is:" + custLoc);
console.WriteLine("Customer number is:" + cusnum);
console.WriteLine("Customer emailID is:" + cusemail);
console.ReadLine();
```

Q.

O/P

Customer ID is : 123

Customer name is : Sandhya.

Customer location is : Kukatpally.

Customer number is : 9848032211.

Customer emailID is : Sandhya@gmail.com.

H/W 11)

Prgm

```
Void Main()
```

```
{
```

```
int proID = 123;
```

```
String proname = "TV";
```

```
String procom = "Sony";
```

```
int procost = 30000;
```

```
Console.WriteLine("Product ID is :" + proID);
```

```
Console.WriteLine("Product name is :" + proname);
```

```
Console.WriteLine("Product Company is :" + procom);
```

```
Console.WriteLine("Product Cost is :" + procost);
```

```
Console.ReadLine();
```

```
}
```

O/P

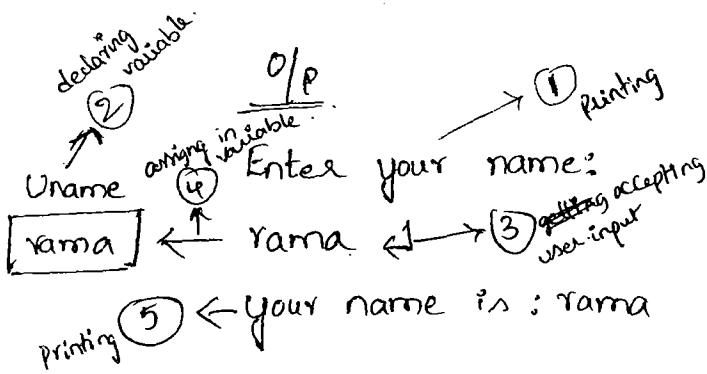
Product ID is : ~~123~~ 123

Product name is : TV

Product company is : Sony

Product cost is : 30000.

22) Write a console program to accept user name & display



Program . Void Main( )  
{

Console.WriteLine ("Enter your name : ");

String <sup>uname</sup> s = Console.ReadLine();

Console.WriteLine ("Your name is : "+uname);

Console.ReadLine();

}

13) Write a console program to accept user name and user location & display. to user

O/P

Enter your name:

uname

rama ← rama

Enter your location:

loc

ameerpet ← Ameerpet

your name is : rama

your location is : Ameerpet

Prgm      void Main()

{

    Console.WriteLine("Enter your name : ");

    String uname = Console.ReadLine();

    Console.WriteLine("Enter your location : ");

    String Uloc = Console.ReadLine();

    Console.WriteLine("your name is :" + uname);

    Console.WriteLine("your location is :" + Uloc);

    Console.ReadLine();

}

14) Write a console prgm to accept user age and display to the user.

ent  
↑

age  
25

O/P

Enter your age :

← 25

your age is : 25

Prgm

void Main()

{

    Console.WriteLine("Enter your age : ");

    int age = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("your age is :" + age);

    Console.ReadLine();

}

### (Console.ReadLine):-

- ReadLine is a predefined member method of console class.
- This method will accept input from the user, until user will press an enter key, once user will press the enter key that accepted value this method will be written as String value becoz ~~as~~ ReadLine() written type.  
Return type is String.

### Convert.ToInt32():-

- Convert is a predefined class which is a part of System base class library.
  - ToInt32 is a predefined member method of Convert class.
  - This method will convert given value in to int datatype
- Ques 15) Write a console program to accept employee no & display

O/P

Enter Employee number :

eno  
123  
your empno is : 123

Prgm    Void Main()

{

    console.WriteLine("Enter Employee number");

    int eno = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("your empno is :" + eno);

    Console.ReadLine();

}

16) Write a console Prgm to accept user name, age and display.

O/P

String uname  
Enter your name:

Sandhya  
Sandhya

int age  
Enter your age:

25  
25

your name is: Sandhya.

your age is: 25

Prgrm

Void Main()

{

Console.WriteLine("Enter your name:");

String uname = Console.ReadLine();

Console.WriteLine("Enter your age:");

int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("your name is :" + uname);

Console.WriteLine("your age is :" + age);

Console.ReadLine()

?

17) Write a console prgm. to accept employee number, name, salary & display.

O/P

Enter emp number:

123

Enter emp name:

Anneetha

Enter emp salary:

25000

Employee number is: 123

Employee name is: Anneetha

Prgm

Void Main( )

{

Console.WriteLine("Enter empnumber:");

int eno = ~~float~~ Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter empname:");

String ername = Console.ReadLine();

Console.WriteLine("Enter empSalary:");

int esal = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Employee number is: " + eno);

Console.WriteLine("Employee name is: " + ername);

Console.WriteLine("Employee Salary is: " + esal);

Console.ReadLine();

}

18) Write a Console Prgm to accept Student name, Stu 3 sub marks they are m<sub>1</sub>, m<sub>2</sub>, m<sub>3</sub>. and calculate the Student total marks and ~~average~~ display total marks with student name. calculate Student avg marks & display avg marks of student with name?

O/P

Enter Student name:

Pavithra.

Enter m<sub>1</sub> marks:

70

Enter m<sub>2</sub> marks:

80

Enter m<sub>3</sub> marks:

75

Pavithra ~~total~~ total marks is:

Pavithra avg. is:

Program

```
Void Main()
{
    Console.WriteLine("Enter student name:");
    String Sname = Console.ReadLine();
    Console.WriteLine("Enter m1 marks:");
    int m1 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter m2 marks:");
    int m2 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter m3 marks:");
    int m3 = Convert.ToInt32(Console.ReadLine());
    int totmarks = m1 + m2 + m3;
    Console.WriteLine("Sname + " + totmarks);
    Console.WriteLine("average is:" + avg);
    Console.ReadLine();
}
```

a) console prg train name & train no: ~~interval~~.

```
Void Main()
{
    Console.WriteLine("Enter train name:");
    String tname = Console.ReadLine();
    Console.WriteLine("Enter train number:");
    int tno = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Train name is:" + tname);
```

```
Console.WriteLine("Train name is :" + tname);  
Console.ReadLine();  
}
```

O/p

Enter train name:

Amaravathi

Enter train number:

12344

Train name is : Amaravathi

Train number is : 12344.

Note : → To move cursor to next line we have to use  
" \n ". Always it should be in double quotes.  
→ To print the tab space we can use " \t ".

24/1/14

Q) Write a Console prgm to accept 2 nos & perform  
the addition & display the addition result like below.

O/p

a Enter the first num:

← 10

b Enter the second num:

← 5

c addition result is: 15.

Program

Void Main()

{

    Console.WriteLine("Enter the first num:");

    int ~~a~~<sup>a</sup> = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Enter the second num:");

    int ~~b~~<sup>b</sup> = Convert.ToInt32(Console.ReadLine());

    int ~~c~~<sup>c</sup> = ~~a+b~~<sup>a+b</sup> + ~~b~~<sup>b</sup>;

    Console.WriteLine("Addition result is "+c);

    Console.ReadLine();

}

- Q1) Write a Console prgm to accept a number & calculate the square of the given number & display.

O/P

Enter the number:

2

2 Square is: 4.

-

prgm

Void Main()

{

    Console.WriteLine("Enter the number:");

    int a = Convert.ToInt32(Console.ReadLine());

    int b = a\*a;

    Console.WriteLine("Square is: "+b);

    Console.ReadLine();

}

22) console prgm to find the cube of given no;

O/P

Enter the number:

2

2 Cube is : 8

Program

Void Main()

{

Console.WriteLine("Enter the number:");

int a = Convert.ToInt32(Console.ReadLine());

int b = a\*a\*a;

Console.WriteLine("a" + "cube is : " + b);

Console.ReadLine();

}

Datatypes :-

→ Datatypes specifies type of the data and size of the data.

→ In C#.Net datatypes are classified into 5 types based on type of the data and size of the data.

1. Numerical datatypes
2. Floating Point datatypes
3. Character Related datatypes
4. Logical datatypes
5. General datatypes.

## 1. Numerical datatypes:

→ A number which is not having any fractional part will come under Numerical datatype.

Eg: 10, 20 . . .

→ Again numerical datatypes are classified into 2 types

1. Signed datatype

2. Unsigned datatype

## 1. Signed numerical datatype :-

→ These datatypes will allow positive and negative values

→ According to sizes signed Numerical datatypes are classified into 4 types.

- |          |           |           |
|----------|-----------|-----------|
| 1. Sbyte | → 1 byte  | → 3 no's  |
| 2. Short | → 2 bytes | → 5 no's  |
| 3. Int   | → 4 bytes | → 10 no's |
| 4. Long  | → 8 bytes | → 19 no's |

### 1. Sbyte :

→ Here 'S' stands for signed.

→ Predefined size of Sbyte is 1 byte i.e.,  $1 \times 8 = 8$  bits of signed integers it can hold maximum.

→ Sbyte base type in CTS is → System.SByte

### 2. Short :

→ predefined size 2 bytes i.e.,  $2 \times 8 = 16$  bits

→ Base type of short is → System.Int16.

### 3. Int :

→ predefined size is 4 byte i.e.,  $4 \times 8 = 32$  bits

→ Base type of int is → System.Int32

#### 4. long:

- predefined size is 8 bytes ie.,  $8 \times 8 = 64$  bits
- Base type of long is → System.Int64.

#### 2. Unsigned Numerical datatypes:-

- These data types will allow only positive values.
- According to the sizes, these data types are classified into 4 types.

1. byte
2. ushort  $\rightarrow 5$
3. uint  $\rightarrow 10$
4. ulong  $\rightarrow 20$

#### 1. byte:

- Predefined size is 1 byte i.e.,  $1 \times 8 = 8$  bits of unsigned integers
- Base type is → System.Byte

#### 2. ushort:

- 'u' stands for unsigned.
- 2 bytes  $\rightarrow 2 \times 8 = 16$  bits of unsigned integers
- Base type is → System.UInt16

#### 3. uint:

- 4 bytes  $\rightarrow 4 \times 8 = 32$  bits of unsigned integers
- Base type is → System.UInt32

#### 4. ulong:

- 8 bytes  $\rightarrow 8 \times 8 = 64$  bits of unsigned integers
- Base type is → System.ULong64. System.UInt64

## 2. Floating Point data types:

- A number which is having fractional part will come under floating point data.
- Ex: 10.5, 20.5
- Floating point data types will have only signed which will not support unsigned data types.
- These are 3 types.

1. Float
2. Double
3. Decimal

### 1. Float:

- Pre defined size is 4 bytes i.e.,  $4 \times 8 = 32$  bits of signed floating values
- Base type is System.Single

### 2. Double:

- pre defined size is 8 byte i.e.,  $8 \times 8 = 64$  bits of signed floating values.
- Base type is System.Double.

### 3. Decimal:

- predefined size is 16 byte i.e.,  $16 \times 8 = 128$  bits of signed floating values.
- Base type is System.Decimal

## 23) Example for float variable :

Program    Void Main()

{

    float a = 4.5;

    Console.WriteLine("a");

    Console.ReadLine();

}

O/P

4.5

- In the above prgm will generate a compile tym error bcoz by default C#-Net compiler will treat floating value as double.
- whenever we want to assign a floating value into float variable we should postfix with "f" like below.

float a = 4.5f;

(08)

float a = 4.5F;

24) Example for double variable -

Prgrm Void Main()

{

double a = 4.5;

Console.WriteLine(a);

Console.ReadLine();

}

25) Example for decimal .

Prgrm Void main()

{

decimal a = 4.5m;

Console.WriteLine(a);

Console.ReadLine();

}

### 3. Character Related datatypes :

#### 1) char

→ Pre defined size is 1 byte i.e.,  $2 \times 8 = 16$  bits of unicode characters.

→ Base type is System.Char

## 25) Example for char

```
Void Main()
{
    char a1 = 'b'; // valid
    Console.WriteLine(a1);
    char a2 = 'ab'; // invalid
    char a3 = '2'; // valid
    char a4 = '@'; // valid
```

## 4. Logical Datatypes :

### 1) bool

- ~~Base~~ Bool variable can contain either True (or) False value
- True will be representing as '1' and False will be representing as '0'
- Predefined size of bool is 1 bit.
- Base type is System.Boolean

## 26) Example for bool variable

```
Program
Void Main()
{
    bool b1 = true; // valid
    Console.WriteLine(b1);

    bool b2 = "true"; // invalid

    bool b3 = 1; // invalid

    Console.ReadLine();
}
```

## 5) General data types :

- In general data types we have 2 types

- 1) String
- 2) object

## 1. String :

→ To represent collection of characters we will go for String variable

Ex: String s = "Satya";

→ There is no predefined size.

→ Base type is System.String

## 2. Object :

→ Base type is System.Object

Note : → The above 5 datatypes are called primitive data types

→ In these primitive data types 1, 2, 3 & 4 are supporting predefined sizes but 5<sup>th</sup> category i.e., general data types will not support predefined sizes.

## MinValue and MaxValue :

→ These two are constants

→ MinValue is written in the given data type starting range

→ MaxValue is written in the ending range of given data type

Q) Example for MinValue & MaxValue of byte & Sbyte datatypes

A) void Main()

{

Console.WriteLine("byte min value is :" + byte.MinValue);

Console.WriteLine("byte Max value is :" + byte.MaxValue);

Console.WriteLine("Sbyte min value is :" + sbyte.MinValue);

Console.WriteLine("Sbyte Max Value is :" + sbyte.MaxValue);

Console.ReadLine();

}

O/P

byte min value is : 0

byte max value is : 255

Sbyte min value is : -128

Sbyte max value is : 127

Note :

- In primitive datatype 1,2,3 & 4 data types are providing by the microsoft as predefined structures.
- And general datatypes ie., String and object are providing by the microsoft as predefined classes with in a base class library called System like below.

Skeleton of System Base class Library :

namespace System

{

    class console

{

        writeLine()

{

}

        ReadLine()

{

}

}

    class convert

{

        ToInt32()

{

}

}

struct int

{

}

struct long

{

}

struct double

{

}

struct char

{

}

struct bool

{

}

class string

{

}

class object

{

}

}

use of  
typeid() and sizeof():

typeid()

→ This will return the base type of given data type

sizeof()

→ This will return the size of the given data type in bytes.

## 28) Ex of sizeof() & typeid()

prog

```
Void Main()
{
    Console.WriteLine(typeof(int));
    Console.WriteLine(typeof(float));
    Console.WriteLine(sizeof(int));
    Console.WriteLine(sizeof(long));
    Console.ReadLine();
}
```

O/P

System.Int32

System.Single

4

8

-

## What is Local Variable?

- A variable which is declared within a method is called as Local Variable.
- Local Variable should be initialised with some value before accessing, otherwise compiler will generate an error.

## 29) Ex:

```
Void Main()
{
    int a;
    Console.WriteLine(a); → error
    Console.ReadLine();
}
```

- To overcome above error we can write the code like below.

```
Void Main()
{
    int a=10;
    Console.WriteLine(a);
    Console.ReadLine();
}
```

(or)

```
int a; // declaration  
a = 10; // assigning.  
Console.WriteLine(a); // accessing.  
Console.ReadLine();
```

Q) Ex to declare multiple variables in a single line.

```
int a = 10, b = 20; // valid  
int a = 10, string s = "Satya"; // invalid  
int a = 10; string s = "Satya"; // valid
```

### Implicit typed Variables:-

→ Using var keyword we can declare implicit typed variables.

→ Implicit typed variables concept was introduced by with .Net framework 3.0 version

→ Implicit typed variable datatype will be deciding based on the value which we are assigning

Q) Ex of implicit typed variables.

```
program Void Main()  
{  
    Var a = 10  
    Var b = "rama";  
    Var c = true;  
    Console.WriteLine(a.GetType());  
    Console.WriteLine(b.GetType());  
    Console.WriteLine(c.GetType());  
    Console.ReadLine();  
}
```

O/P

System.Int32

System.String

System.Boolean

GetType():

→ This method will return the base type of given variable.

Difference b/w typeof() and GetType():

→ Using GetType() we can get the base type of given variable.

→ Using typeof() we can get the base type of given data type.

31) Ex for implicit typed variable.

```
Void Main()
{
    Var a = 6.5;
    Var b = 4.5f;
    Var c = 6.5m;
    Int d = 10;

    Console.WriteLine(a.GetType());
    Console.WriteLine(b.GetType());
    Console.WriteLine(c.GetType());
    Console.WriteLine(d.GetType());

    Console.ReadLine();
}
```

O/P

System.Double

System.Single

System.Decimal

System.Int32

when will we use Implicit Variables.

→ We will use Implicit typed variable at the time of implementing ~~king~~ concept.

→ According to data storage location C#-Net datatypes are classified into two types.

1. Value types
2. Reference types

1. Value types :-

→ In value types data will be storing into stack memory, due to that reason value types are called as Stack based datatypes.

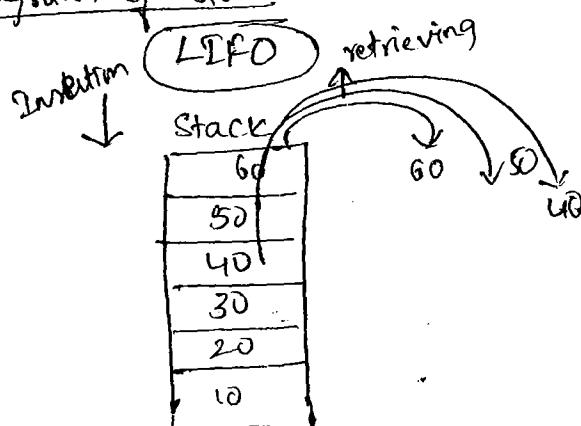
Stack:

→ Stack is a data structure which can contain collection of elements

→ Stack will support only one end i.e., Topend due to that reason, insertion & retrieval we can perform from Topend.

→ Stack is following an approach called LIFO [Last In First Out] that means last inserted element we can retrieve first

Diagram of stack



## 2. Reference Types:

→ Reference type data will be stored into heap memory.

Due to that reason, reference types are called heap based data types.

### heap

→ Heap is a data structure which can contain collection of elements.

→ Heap will be representing in binary tree structure format.

→ Heap is a collection of nodes, which are two types.

1. Parent node

2. Child node.

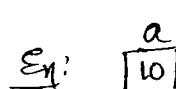
→ Heap is not following any approach called LIFO or FIFO, which is following its own approach called Random approach that means within the heap we can reach the an element randomly.

## Difference b/w value types & Reference types.

### Value Types

1. Data is storing in to stack.

2. Value type variable will contain the actual data like below.

Ex:  a  
10

3. Value types are:-

1. Structure

2. In primitive datatypes following are value types:-

a) Numerical data types

b) Floating point data types

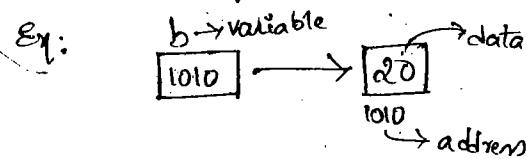
c) Character related data types

d) Logical data types.

### Reference Types

1. Data is storing in to heap.

2. Reference type variable will contain the address of the data like below

Ex:  b → variable  
1010 → 20  
1010 → address

3. Reference types are:-

1. class

2. In primitive datatypes following are Reference type

a) General data types

Ex: String, object

Ex: Int, float, double, char,  
bool.....

### 3) Enums(Enumerator)

4)

### 3) Interface

### 4) delegates

Can we assign Null value into Value type variable?

→ No

Ex: Int a = null; // invalid

Can we assign null value into reference type variable?

→ Yes

Ex: String s = null; // valid.

→ Till .Net 1.1 we cannot assign null value into value type.

Variable.

→ From .Net 2.0 we can assign null value into value type variable with the help of nullable value types.

### nullable value types:

→ This was introduced with .Net 2.0 version.

→ Using nullable value types we can assign a null value into value type variable.

→ Example for nullable value types.

32) `int? a = null;`

{

`float? b = null;`

`y`

when we will go for nullable value types?

→ whenever the input is optional. We will go for declare particular variable as nullable value type.

30/01/14

Example to perform division.

```
Void Main()
{
    int a = 4;
    int b = /2;
    div c = a/b;
    console.WriteLine("Enter first no:");
    Console.ReadLine();
    console.WriteLine("Enter second no:");
    console.WriteLine("Division is: " + c);
    console.WriteLine
```

Void Main()

{

```
    Console.WriteLine("Enter first no:");
    int a = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter second no:");
    int b = Convert.ToInt32(Console.ReadLine());
    div c = a/b;
    console.WriteLine("division is: " + c);
    Console.ReadLine();
```

}

Note: → In the above program when user enter second no as zero it is throwing runtime error.

→ We can handle these runtime errors in two ways

i. Using logic

ii. Exception handling mechanism.

→ To solve the runtime errors by using logic we should go for control statements.

## Control Statements:-

→ In C# .Net Control statements are 3 types.

1. Conditional statements
2. Loop
3. Jump statements

### 1. Conditional statements.

1. If condition
2. Switch condition

→ When we will go for conditional statements.

Ans whenever we want to execute a single statement or block of statements based on condition we will go for conditional statements.

### If Condition :-

→ We can implement if condition in 4 ways

1. If (Simple if)
2. If else
3. Multi if else if (multiple if)
4. Nested if

#### 1. Simple if:

Syntax: if(<condition>)

{

  Stm 1;

  Stm 2;

}

33) → Example to handle the above division prgrm error by using simple if.

O/P

Enter first no:

a  
10 ← 10

Enter second no:

b  
0 ← 0

Please enter other than Zero:

b  
2 ← 2

div

a/b → 5

Division result is : 5

Program

Void Main()

{

Console.WriteLine("Enter first no:");

int a = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter second no:");

int b = Convert.ToInt32(Console.ReadLine());

If (b == 0)

{

Console.WriteLine("Please Enter other than Zero:");

b = Convert.ToInt32(Console.ReadLine());

Console.ReadLine();

}

Console.WriteLine("Enter second no:");

int b = Convert.ToInt32(Console.ReadLine());

div c = a / b;

Console.WriteLine("division result is :" + c);

Console.ReadLine();

→ when we will go for simple if?

Answ → Whenever we have a single option, which we want to execute based on condition we can go for simple if.

## 2. if else :

Syntax: if (<condition>)

  st 1; a

~~st 2;~~

else

  st2; b

34) Write a console prgm to compare two no's by using if else

O/p

Enter first no:

10

↙

Enter second no:

5

↙

i is greater than j.

Prgm

Void Main()

{

Console.WriteLine("Enter first no:");

int i = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter second no:");

int j = Convert.ToInt32(Console.ReadLine());

```
if (i > j)
{
    console.WriteLine("i is greater than j");
}
else
    console.WriteLine("j is greater than i");
    console.ReadLine();
}
```

### 3. If else if:

```
Syntax : if (<condition>)
{
    st1;
    else if (<condition2>)
        st2;
    else if (<condition3>)
        st3;
    else
        st4;
```

34) Implement comparing two no's with the help of if else if.

```
Void Main()
{
    console.WriteLine ("Enter first no:");
    int i = Convert.ToInt32(console.ReadLine());
    console.WriteLine ("Enter second no:");
    int j = Convert.ToInt32(console.ReadLine());
    if (i > j)
        console.WriteLine ("it is big");
    else if (i > j)
        console.WriteLine ("it is big");
```

else

Console.WriteLine("it is equal to " + j);

Console.ReadLine();

}

Q) whenever we will go for if else?

An whenever we have two options, want to execute one option at a time among two options based on condition then we can go for if else.

Q) when we will go for if else if?

An whenever we have more than two options, at a time if we want to execute one option among multiple options based on condition then we can go for if else if.

35) Implement the above example with if else.

prog

Void main

{

    Console.WriteLine("Enter first no.");

    int a = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Enter second no.");

    int b = Convert.ToInt32(Console.ReadLine());

    if (a >= b)

{

        if (a > b)

            Console.WriteLine("a is greater than b");

    else

        Console.WriteLine("a is equal to b");

}

else

```
Console.WriteLine ("a is less than b");  
Console.ReadLine();
```

g.

→ In the above example we have implemented nested if

Q) When we will go for nested if?

An whenever we have multiple options which we want to execute 1 option based on ~~execution~~ condition first choice is if else if , second choice is nested if .

36) Implement the above prgm using simple if conditions

prgm void Main()

{

```
Console.WriteLine ("Enter first no:");  
int a = Convert.ToInt32 (Console.ReadLine());  
Console.WriteLine ("Enter second no:");  
int b = Convert.ToInt32 (Console.ReadLine());  
if (a > b)  
    Console.WriteLine ("a is greater than b");  
else if (a < b)  
    Console.WriteLine ("b is greater than a");  
else if (a == b)  
    Console.WriteLine ("a is equal to b");
```

Console.ReadLine();

g.

## 2/14. 2. Switch :-

Q) When will we go for Switch?

Ans Whenever we want to execute single case among multiple cases we can go for switch.

Syntax:

Switch(<expression>)

{       $\nearrow$  Label

case 1:

st1;

break;

case 2 :

st2;

break;

case 3 :

st3;

break;

default:

st4

break;

}

37) Example for switch condition to display month name, based on month no -

O/P

month

$\leftarrow$  1

Jan

—

Program

Void Main()

{

Console.WriteLine("Enter your month number : ");

Console. Int a = Convert.ToInt32(Console.ReadLine());

Switch( $\frac{a}{month}$ )

{

Case 1:

```
Console.WriteLine("January");  
break;
```

Case 2:

```
Console.WriteLine("February");  
break;
```

Case 3:

```
Console.WriteLine("March");  
break;
```

Case 4:

```
Console.WriteLine("April");  
break;
```

Case 5:

```
Console.WriteLine("May");  
break;
```

Case 6:

```
Console.WriteLine("June");  
break;
```

Case 7:

```
Console.WriteLine("July");  
break;
```

Case 8:

```
Console.WriteLine("August");  
break;
```

Case 9:

```
Console.WriteLine("September");  
break;
```

Case 10:

```
Console.WriteLine("October");  
break;
```

Case 11:

```
Console.WriteLine("November");  
break;
```

Case 12 :

```
Console.WriteLine ("December");
break;
default:
    Console.WriteLine ("Invalid input");
    break;
}
```

```
Console.ReadLine();
```

}

Q8) Write a Console prgm to compare 2 nos by using switch

Program

```
Void Main()
```

{

```
    Console.WriteLine ("Enter first no:");
    int a = Convert.ToInt32 (Console.ReadLine());
```

```
    Console.WriteLine ("Enter Second no:");
    int b = Convert.ToInt32 (Console.ReadLine());
```

```
    Switch (a > b)
```

{

Case True :

```
    Console.WriteLine ("a is greater than b");
    break;
```

Case False :

```
    Console.WriteLine ("b is greater than a");
    break;
```

default :

```
    Console.WriteLine ("both are equal");
    break;
```

}

```
, Console.ReadLine();
```

- Note :> Switch expression can have a condition and it can have  
 bool, char, string, integral, enum (or) corresponding type values  
 but it cannot contain floating type value.
- Switch case label can be integer value, String value, char value,  
 bool value, enum value and it cannot be floating value.
- Switch case label cannot have condition.

39) void Main()

{

Console.WriteLine("Enter first no:");

int a = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter second no:");

int b = Convert.ToInt32(Console.ReadLine());

switch (a > b)

{

case true :

Console.WriteLine("a is greater than b");

break;

case false :

switch (b > a)

{

case true :

Console.WriteLine("b is greater than a");

break;

case false :

Console.WriteLine("a is equal to b");

break;

} // internal switch close

break;

} // external switch close

Console.ReadLine();

}

Q) Comparison b/w if else if and switch condition.

- Ans → Both are executing one option among multiple options.  
→ Execution wise switch is faster than if else if because switch is direct branching and if else if is ladder wise branching.  
→ Switch is execution wise faster than if else if, but which is having some limitations like below
1. expression of switch cannot be a floating value
  2. Switch label cannot be a floating value & condition.
- Q) → whenever we want to execute multiple options among multiple options first priority is to switch then next priority is if else.

40) Write console prgm to compare two nos with the combination of switch & if else

program

Void Main()

{

Console.WriteLine("Enter first no:");

int a = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter second no:");

int b = Convert.ToInt32(Console.ReadLine());

switch (a > b)

{

case true:

Console.WriteLine("a is greater than b");

break;

Case false:

if (b > a)

Console.WriteLine("b is greater than a");

else

Console.WriteLine("both are equal");

```
break;
```

3

```
Console.ReadLine();
```

4

5/2/14

- 4-) Write a Console prgm to accept a letter and display whether it is vowel or not:

Prgrm

```
void Main()
```

```
{
```

```
Console.WriteLine("Enter your letters");
```

ch  
i

← ?

O/p

Enter your letters:

it is a vowel.

```
Char ch = Convert.ToChar(Console.ReadLine());
```

```
Switch (ch)
```

```
{
```

case 'a':

```
Console.WriteLine("it is a vowel");
```

```
break;
```

case 'e':

```
Console.WriteLine("it is a vowel");
```

```
break;
```

case 'i':

```
Console.WriteLine("it is a vowel");
```

```
break;
```

case 'o':

```
Console.WriteLine("it is a vowel");
```

```
break;
```

case 'u':

```
Console.WriteLine("it is a vowel");
```

```
break;
```

default:

```
Console.WriteLine("it is not vowel");
```

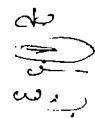
```
break;
```

g

Console.ReadLine();

g

Ho



## LOOPS :

Q) When we will go for loops?

Ans Whenever we want to execute a single statement or block of statements multiple times we will go for loops.

Ex: Write a console prgm to print Satya Technologies 5 times.

Prgm      void Main()

{

```
    Console.WriteLine("Satya Technologies");
    Console.WriteLine("Satya Technologies");
    Console.WriteLine("Satya Technologies");
    Console.WriteLine("Satya Technologies");
    Console.WriteLine("Satya Technologies");
    Console.ReadLine();
```

}

→ In the above prgm we have written WriteLine statement 5 times, by using loops we can write it as a single statement.

## Benefit of loops:-

→ We can reduce the number of lines of codes with the help of loops.

→ In C# .Net we can implement loops in 4 ways.

- 1) for loop
  - 2) while loop
  - 3) do while loop
  - 4) for each loop

## for loop :

Syntax:    `for(<initialization>; <condition>; <+ + (or) - ->)`

{  
    St 1;  
    St 2;  
}

43) write a console prgm to print 1 to 10 no's using <sup>for</sup> loop

Program      Void Main ()  
                  { }

```
Console.WriteLine(" Numbers are :");
```

```
int i = Convert.ToInt32(Console.ReadLine());
```

$(\theta \leq 10^\circ)$   
(d)

for(  $i = 1$ ;  $i < 11$ ;  $i++$ )

1

```
console.writeLine (#i + "\n");
```

```
Console.ReadLine();
```

2

alp

44) write a program to print no's like below by using for loop.

10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Program      void Main()

```
Console.WriteLine("Number are:");
```

```
int f = Convert.ToInt32(Console.ReadLine());
```

```
for(i=10; i>=1; i--)
```

10

```
Console.WriteLine(i + "\n");
```

```
Console.ReadLine();
```

```
}
```

Note : For debugging F10 or F11

(4) Write prgm to print no's like below.

O/p

```
1  
3  
5  
7  
9  
11  
15  
17  
19
```

Prgrm:

```
Void Main():
```

```
{
```

```
for (int i = 1; i < 20; i++)
```

$i = i + 2$

```
Console.WriteLine(i + "\n");
```

```
Console.ReadLine();
```

```
}
```

while loop:

Syntax : while (<condition>)

```
{
```

```
St 1;
```

```
St 2;
```

```
}
```

(45) write a prgm to print 1 to 10 no's like below

Prgm    Void Main()  
{  
    int i=1;  
    while (int p <= 10)  
    {  
        Console.WriteLine(i);  
        i++  
    }  
    Console.ReadLine();  
}

O/P  
1 2 3 4 5 6 7 8 9 10 -

for (i=1; i <= 10; i++)

Console.WriteLine(); :-

→ Write is a predefined method of Console class

→ This method will print the given value on output window, after printing the value it will keep the cursor in the same line of next character.

Do while loop: -

Syntax :    Do  
              {  
              st 1;  
              st 2;  
              i++ (or) i--;  
              }  
              while (<condition>);

O/P

(46) write a Console prgm to print 1 to 100 nos like below.

O/P    1, 5, 10, 15, 20 - - - 100,

Prgm    Void Main()  
{  
    int i=1;  
    Do  
    {  
        Console.WriteLine(" Numbers are: ");  
        Console.Write(p + ",");  
    }  
}

```
while (i < 20);  
{  
    if (i == 1)  
        console.write(i);  
    i++;  
    int a = (i - 1) * 5;  
    console.write(a);  
}  
Console.ReadLine();  
}
```

Q7) Prgm to print 1 to 10 no's like below using Do while

O/P

1, 2, 3, 4, 5, 6, 7, 8, 9, 10. —

Prgm void main()

{

int i = 1;

Do

{

console.write(i + " ")  
 i++;

while (i <= 10)

{  
 console.write(i + ".");

i++;

}

Console.ReadLine();

}

```

void main()
{
    int i=1
    do
    {
        console.write(i + ",");
        i++
    }
    while (i <= 10)
    {
        console.write(i + ".");
        console.ReadLine();
    }
}

```

Q) When we will go for while loop and when we will go for for loop?

Ans → When the no. of iterations are fixed or decided we will go for for loop.

→ When the no. of iterations are not fixed we will go for while loop.

Purposeful

(f) Example for while loop.

division by zero prgm by using while.

Prgm

```

void main()
{
    console.WriteLine("first no:");
    int a = Convert.ToInt32(console.ReadLine());
    console.WriteLine("second no:");
    int b = Convert.ToInt32(console.ReadLine());
    while (b == 0)
    {
        console.WriteLine("Enter 'b' other than zero");
        b = Convert.ToInt32(console.ReadLine());
    }
    int div = a / b;
    console.WriteLine(div);
    console.ReadLine();
}

```

(a) Implement the above prgm by using for loop

```
Void Main()
{
    // first 4 lines are same as above prgm
    for( ; b==0; )
    {
        Console.WriteLine("pls enter other than zero");
        b = Convert.ToInt32(Console.ReadLine());
    }

    int div = a/b;
    Console.WriteLine("Result is: " + div);
    Console.ReadLine();
}
```

→ we can implement a for loop without initialization & increment (or) decrement. By default we can work only with condition also. So for the above requirement abov. best choice is while loop

(b) Can we implement a for loop without initialization, condition, increment (or) decrement?

Ans Yes possible like below but will be like infinite loop.

```
for(;;)
{
    Console.WriteLine("hi");
    Console.ReadLine();
}
```

(c) When we will go for Do while loop?

Ans whenever the no.of iterations are not fixed and even though condition is false it has to execute atleast one iteration then we can go for Do while loop.

50) Implement above division prgm using Do while.

51) Write a purposeful console prgm for Do while.

for each loop :-

→ for each is one of the looping concept which we will use to fetch the elements from collections like array, stack, queue.

Jump Statements :-

→ Using jump statements we can transfer the prgm control.

→ we have 5 types

1. break..

2. Continue

3. goto

4. return

5. throw

1. break :

→ break statement will transfer the prgm ctrl to out of nearest closing brace and out of switch block.

52) Write a prgm to print 1 to 100 nos but it should print only 1 to 5 nos like below.

<u>prgm</u>	Void main()	<u>o/p</u>
	{	1
	int i;	2
	for(i=1; i<=100, i++)	3
	if(i==5)	4
	Console.WriteLine(i); if(i==5);	5
	break;	-
	}	
	Console.ReadLine();	
	}	10

## 2. Continue :-

→ Continue will skip the current iteration

- 53) prgm to print 1 to 10 nos but it has to print the o/p like below

O/P

1  
2  
3  
4  
6  
7  
8  
9  
10

Void Main( )

{

for (i=1; i<=10; i++)  
{  
 Console.WriteLine(i);  
 if (i == 5)  
 {  
 continue;  
 }  
 Console.WriteLine(i);  
}

Console.ReadLine();

}

## 3. goto :-

→ goto will transfer the prgm control to the given label as well as to the switch case.

→ label should be defined by the prgm programmer like below

Satya: → label

54) Example for goto.

```
prog void main()
{
    Console.WriteLine("I like C#-Net");
    Console.WriteLine("I like ASP-.Net");
    goto skip;
    Console.WriteLine("I like Java");
skip:
    Console.ReadLine();
}
```

Op

I like C#-Net

I like ASP-.Net

55) Example for division program by using if condition with the help of go to.

```
prog void main()
{
    Console.WriteLine("first no.");
    int a = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("second no.");
    int b = Convert.ToInt32(Console.ReadLine());
    if (b == 0)
    {
        goto consoleWL("Enter other than zero");
        b = Convert.ToInt32(Console.ReadLine());
    }
    Console.ReadLine();
    int div = a / b;
    Console.WriteLine("Result is:" + div);
    Console.ReadLine();
}
```

46) To print 1 to 100 no's as below using do while loop

Op

1, 5, 10, 15, 20, 25, - - - 99, 100,

Program

```
Void main()
{
    int i = 1;
    Console.WriteLine("Numbers are:");
    do
    {
        if (i == 1)
            Console.WriteLine(i, " ");
        i++;
        int a = (i - 1) * 5;
        Console.WriteLine(a + ", ");
    }
    while (i <= 20);
    Console.ReadLine();
}
```

50)

```
Void Main()
{
    // first 4 times are same
    do
```

56) Example to transfer program control to the targeted Switch case  
using go to:

```
Void Main()
{
    Console.WriteLine ("Enter your choice 1 or 2");
    int i = Convert.ToInt32(Console.ReadLine());
    Switch (i)
    {
        Case 1:
            Console.WriteLine ("This is first case.");
            break;
        Case 2:
            Console.WriteLine ("This is second case");
            break;
            goto case 1;
        Default:
            Console.WriteLine ("This is default case");
            break;
    }
    Console.ReadLine();
}
```

#### 4. return :

- Using return statement we can return the ~~program control~~ value.
- Whenever we are defining a user defined function to return a value. we can use return statement.
- Once return statement is executed control will transfer the out of the method block.

## 5. throw:

→ Using throw keyword we can throw an exception explicitly

57) Console prgm to store 5 employee no's. They are 111, 222, 333, 444, 555.

→ To implement above prgm we have to go for five variables

→ whenever we want to store multiple elements which are same type we will go for a concept called array.

## Arrays

→ Array is a collection of elements which are similar datatype

→ every element will be representing with one index value.

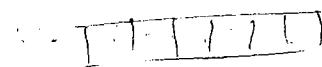
first element index value will be '0' and 2<sup>nd</sup> element index value is 1. and so on..

→ C# .Net will support 3 types of arrays.

1. One dimensional array

2. Multi dimensional array

3. Jagged array.



### 1. One dimensional array:

→ Arranging collection of elements in a single row is called as one dimensional array.

→ Syntax: <int, string> <sub>datatype</sub> <sup>Subscription operator</sup> <sup>→ undefined</sup> <sup>keyword</sup> <sup>no. of elements</sup>  
<datatype>[ ] <arrayname> = new <datatype> [size];

→ Syntax to declare an array with initializations -

<datatype>[ ] <arrayname> = new <datatype> [ ] { <element1>, <element2>,  
<element3>, ... -- };

→ Syntax to access array element:-

array name [ <index> ] ;

Ex: a

10	20	30	40	50
0	1	2	3	4

$$\begin{aligned} n &= 5 - 1 \\ &= 4 \end{aligned}$$

a[3] → 40

a[0] → 10

- 57) Console prgm to create one dimensional array like above and display the array elements like below

prgm void Main()

{

Wrong

O/p

10 20 30 40, 50.

int [] a = new int [] { 10, 20, 30, 40, 50 };

X console.WriteLine( a[0] + " " + a[1] + " " + a[2] + " " + a[3] + " " + a[4] );

console.ReadLine();

}

→ For the above prgm if we have more elements so we go for loops

58) void Main()

{

int [] a = new int [] { 10, 20, 30, 40, 50 };

for ( i = 0; i < a.Length; i++ )

10 20 30 40 50

console.WriteLine( a[i] + " " );

console.ReadLine();

}

## Length :

- length is a predefined member property of array class.
- length property will return the no of elements with in the no. of elements with in the given one dimensional array.

Q.W.

59) Implement above prgm using while & do while loops.

60) Console prgm to pr create string one dimensional array.

## Multi Dimensional array:

→ Multi di

→ Representing Collection of elements in matrix format is called as Multi dimensional array.

→ Multi dimensional array is a collection of rows and collection of columns.

→ every row should have the same no. of elements.

→ Syntax to declare multi dimensional array :-

<datatype>[ , ]<array>=new<datatype>[size, size]; ↑ no. of columns

Ex) Example for Multi dimensional array. ↑ no. of rows

a	0	1	2
0	10	20	30
1	40	50	60

Void main()

{  
int i, j;

int [ , ] a=new int [2][3] {{10,20,30}, {40,50,60}};

for(i=0; i<=2; i++)

{  
for(j=0; j<=3; j++)

a[0,0] → 10

a[0,2] → 30

a[1,2] → 60

op

10 20 30

{40,50,60}; 40 50 60

```
a[i,j]  
console.WriteLine(a[i,j] + " ");  
console.WriteLine();
```

```
Console.ReadLine();
```

y

→ now

a.GetLength(0); :-

→ It will return no. of rows with in multi dimensional array.

a.GetLength(1); :-

→ It will return no. of columns with in multi dimensional array.

Q2) Implement above prgm by using while & while loop.

Q3) above prgm using while & for

Q4) for - while

Q5) while - do while

Q6) dowhile - while

Q7) for - do while

Q8) do while - for

HW

Q9) using while

O/p  
10 20 30 40 50 -

Program

```
Void Main
```

```
{
```

```
    int i = 0;
```

```
    int[] a = new int[5] {10, 20, 30, 40, 50};
```

```
    while (i < 5)
```

```
{
```

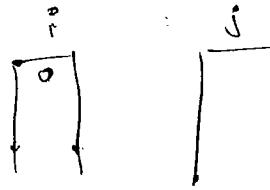
```
        console.WriteLine(a[i] + " ");
```

```
        i++;
```

```
}
```

```
    console.ReadLine();
```

```
}
```



## Using do while

```

Void Main()
{
    int i = 0;
    int [ ] a = new int [ ] { 10, 20, 30, 40, 50 };
    do
    {
        console.write(a[i] + " ");
        i++;
    }
    while (i < 5);
    console.ReadLine();
}

```

(60)

Void Main()

{

String int i

String [ ] a = new String [ ] { "rama", "Sita", "Satya" };

for (i = 0; i < a.length; i++)

{ index (3)

console.write(a[i] + " ");

}

console.ReadLine();

}

O/P.

rama sita satya -

(62)

Void Main()

{

int i = 0, j = 0;

int [ , ] a = new int [ , ] { { 10, 20, 30 }, { 40, 50, 60 } };

int p = 0;

while (i < 2)

{ int j = 0;

while (j < 3)

S < 2

console.write(a[i, j] + " ");

j += 1;

}

```
        console.WriteLine();
        i++;
    }
}
```

```
63) int void Main()
{
    int i=0;
    int[,] a = new int[,] {{10, 20, 30}, {40, 50, 60}};
    while (i < 2)
    {
        for (j=0; j < 3; j++)
        {
            a[i,j] = a[i,j].GetLength(0);
            console.WriteLine(a[i,j] + " ");
        }
        console.WriteLine();
        i++;
    }
    console.ReadLine();
}
```

```
64) void Main()
{
    int i=0;
    int[,] a = new int[,] {{10, 20, 30}, {40, 50, 60}};
    for (i=0; i < 2; i++) → row
    {
        while (j < 3) → col.
        {
            console.WriteLine(a[i,j] + " ");
            j++;
        }
        console.WriteLine();
    }
    console.ReadLine();
}
```

0	1	2
0	10	20 30
1	40	50 60

a[0,1]

65) void main()

```
{  
    int i=0;  
    int[,] a = new int[,] {{10,20,30},{40,50,60}};  
    while (i<2)  
    {  
        int j=0;  
        do  
        {  
            if (j<3)  
                Console.WriteLine(a[i,j] + " ");  
            j++;  
        }  
        while (j<3);  
        Console.WriteLine();  
        i++;  
    }  
    Console.ReadLine();  
}
```

67) void Main()

```
{  
    int[,] a = new int[,] {{10,20,30},{40,50,60}};  
    for(int i=0; i<2 ; i++)  
    {  
        int j=0;  
        do  
        {  
            if (j<3)  
                Console.WriteLine(a[i,j] + " ");  
            j++;  
        }  
        while (j<3);  
        Console.WriteLine();  
    }  
    Console.ReadLine();  
}
```

66)

```
int[,] a = new int[,] {{ {10, 20, 30}, {40, 50, 60} };
```

```
int i=0 → i
```

do

d

```
int j=0; → co
while (j < a.GetLength(1))
{
    console.write(a[i,j] + " ");
    j++;
}
```

```
console.WriteLine();
i++;
```

```
while (i < a.GetLength(0));
    console.ReadLine();
```

y.

68)

```
int[,] a = new int[,] {{ {10, 20, 30}, {40, 50, 60} };
```

```
int i=0
```

do

{

```
for (int j=0; j < a.GetLength(1); j++)
    console.write(a[i,j] + " ");
```

y,

```
while (i < a.GetLength(0))
```

```
    console.WriteLine();
```

```
i++;
```

```
while (i < a.GetLength(0));
    console.ReadLine();
```

y

0  
1  
2  
3

Q) How to get the number of elements with in Multidimensional array?

Ans: Using Length property we can get the no.of elements using multi dimensional array.

69) Write a console prgm to create Multidimensional array like below

a	0	1	2	3
0	10	15	20	25
1	30	35	40	45
2	50	55	60	65

and print this array elements like below

Prgm

Void Main()

{

```
int[,] a = new int[,] {{10,15,20,25},{30,35,40,45},{50,55,60,65}};
```

O/P

10 15 20 25  
30 35 40 45  
50 55 60 65

O/P ①

10 15 20 25  
30  
35  
40  
45  
50 55 60 65

⑨  
1  
2

35 35  
25 25

O/P ②

10 15 20 25  
30 35 40 45  
50  
55  
60  
65

O/P ③

10  
15  
20  
25  
30 35 40 45  
50 55 60 65

11  
22

O/P ④

10  
15  
20  
25  
30  
35  
40  
45  
50 55 60 65

O/P ⑤

10  
15  
20  
25  
30 35 40 45  
50  
55  
60  
65

21 Implement above 5 o/p with diff loop combinations.

### 3. Jagged Array:

- Jagged array is a collection of rows which may not contain same no. of elements.
- That means in jagged array every row may contain different no. of elements.
- Finally we can say jagged array is a collection of one dimensional arrays.

#### Advantage:

- We can save the memory.

- Syntax for jagged array :-

`<datatype> E [][] <arrayname> = new <datatype> [size] [];`

No. of rows

70) Example for jagged array.

a	0	1	2	3
0	10	25	16	17
1	20	30		
2	40	50	60	

`a[0][0] → 10`

`a[1][1] → 30`

`a[2][2] → 60`

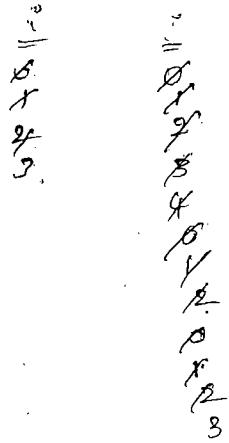
program void Main  
{  
 int [][] a = new int [3] [];  
 a[1][2] = 3  
 a[2][3] = 4  
 for (int i = 0; i < a.GetLength(0); i++)  
 {  
 for (int j = 0; j < a.GetLength(1); j++)  
 {  
 Console.WriteLine(a[i][j] + " ");  
 }  
 Console.WriteLine();  
 }  
 Console.ReadLine();  
}

```

Void Main()
{
    int[][] a = new int[3][];
    // Step 2
    // 1st row
    a[0] = new int[] {10, 15, 16, 17};
    // 2nd row
    a[1] = new int[] {20, 30};
    // 3rd row
    a[2] = new int[] {40, 50, 60};

    for (int i = 0; i < a.length; i++)
    {
        for (int j = 0; j < a[i].length; j++)
        {
            Console.WriteLine(a[i][j] + " ");
        }
        Console.WriteLine();
    }
    Console.ReadLine();
}

```



### a.Length:

- Here a is jagged array.
- It will return no. of rows within the jagged array.

### a[i].Length:

→ 1st iteration

$$a[0].Length \rightarrow 4$$

↳ 1st one dimensional array

→ 2nd iteration

$$a[1].Length \rightarrow 2$$

↳ 2nd one dimensional array

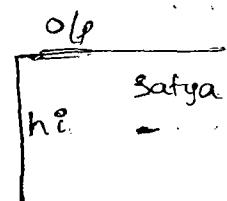
→ 3rd iteration

$$a[2].Length \rightarrow 3$$

↳ 3rd one dimensional array.

- 71) Implement the above prgm by using diff loops combinations
- 72) Write a jagged array to store customer address & implement that array using diff loops combination & display customer address which is available in jagged array with diff o/p formats.
- 73) write a console prgm for 1t and 1n

```
Void Main()
{
    console.write("1t Satya\n");
    console.write("Ht 1t");
    console.ReadLine();
}
```



- 74) Write a console prgm to accept elements from the user and store into one dimensional array and display to user.

O/P

Enter your Element : 10

⑧

Enter your Element : 20

a

10	20	30	40	50
----	----	----	----	----

Enter your Element : 30

Enter your Element : 40

Enter your Element : 50

Array elements are :

10 20 30 40 50

Program:

Void Main()

```
{
    int[] a = new int[5];
    Console.WriteLine("Enter ur element : ");
    a[0] = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter your element : \n");
    a[1] = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter your element : \n");
    a[2] = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter your element : \n");
    a[3] = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Enter your element : \n");
}
```

a[4] = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter your element : \n");

```
for (int i=0; i<a.length; i++)
{
    Console.WriteLine(a[i] + " ");
    a[i] = Convert.ToInt32(Console.ReadLine());
}
Console.WriteLine
```

Program

```
Void Main()
{
    int[] a = new int[5];
    for (int i=0; i<a.Length; i++)
    {
        Console.Write("Enter your element : ");
        a[i] = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine();
    }
    Console.WriteLine("Array elements are : ");
    for (int j=0; j<a.Length; j++)
    {
        Console.Write(a[j] + " ");
        Console.ReadLine();
    }
}
```

75) Write a console program to create two arrays ① names ② ages  
Accept user name, user age store into concern arrays. Accept the user  
name and display the concern user age. If user entered invalid  
name display err msg.

O/P

Enter ur name : ramu

Enter ur age : 20

Enter ur name : sita

Enter ur age : 18

Enter ur name : venkata

Enter ur age : 25

Enter ur name : Gopi

Enter ur age : 26

Enter ur name : ramesh

Enter ur age : 28

find

Enter searching user name : Gopi

Gopi age is : 26

Prgrm

Void Main()

{

//Step 1 declaring arrays

String [ ] names = new String [5];

int [ ] ages = new int [5];

//Step 2 accepting & storing into arrays.

for (int i = 0; i < 5; i++)

{

Console.WriteLine ("Enter ur name : ");

names [i] = Console.ReadLine();

Console.WriteLine ("Enter ur age : ");

ages [i] = Convert.ToInt32 (Console.ReadLine());

Console.WriteLine ("\\n\\n");

}

Console.WriteLine ("Enter searching user name : ");

String find = Console.ReadLine();

int ctr = 0;

for (int i = 0; i < 5; i++)

{ if (names [i] == find)

{ names [i] + " age is : " + ages [i]);

Console.WriteLine ("Valid user ");

ctr++

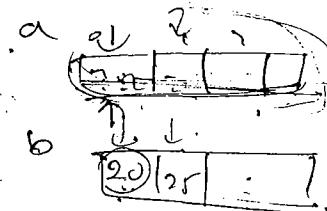
if (ctr == 0) { Console.WriteLine ("Invalid user "); }

Console.ReadLine();

}

if (ctr == 0)

{ Console.WriteLine ("



## Type Casting: (Type Converting)

→ It is a process of converting from one data type to another data type.

Q) When we will go for type casting?

Ans: → whenever we want to assign one datatype value into another data type variable we go for type casting

→ In C# .Net will support type casting in 3 ways.

1) Implicit type casting.

2) Explicit type casting

3) Boxing and unboxing.

1. Implicit type casting:-

→ By default C# .Net will support some of the type conversions which are called as implicit type casting.

→ To implement implicit type casting programmer doesn't require to implement any special syntax except assignment operation.

Following table describing the possible implicit type conversions in C# .Net.

From	To
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double

char      ushort, int, uint, long, float, ulong, double, decimal  
String     object

76) Example for implicit type casting from int to long

```
Void Main()
{
    int i = 10;
    long j = i;
    Console.WriteLine("j value is : " + j);
    Console.ReadLine();
}
```

77) Example for implicit type casting to convert from char to int & double

```
Void Main()
{
    char c = 'a';
    int i = c;
    Console.WriteLine("int i value is : " + i);
    double d = c;
    Console.WriteLine("double d value is : " + d);
    char e = '1';
    int j = e;
    Console.WriteLine("Int j value is : " + j);
    Console.ReadLine();
}
```

O/P  
int i value is : 97

double d value is : 97

int j value is : 48

78) Implement above all possible implicit type conversions.

## 2. Explicit type casting:

(Q) When we go for explicit type casting?

Ans → whenever we are assigning one type value into another type variable which is not possible by implicit type casting then we have to go for explicit type casting.

→ while implementing explicit type casting programmer has to follow some special type conversion Syntax.

→ In C# .Net explicit type casting we can implement by using 3 techniques

1. ~~C++~~ C++ style of type casting technique
2. Parsing Technique
3. Converting Technique.

1. C++ Style of type casting technique:

→ Syntax

<To datatype><To variable> = <(To datatype)> <from variable>;

Example for C++ style of type casting from converting from int to byte

Ans) 

```
int p=100;
byte b=(byte)i;
Console.WriteLine("byte b value is "+b);
Console.ReadLine();
```

2. Parsing:-

→ To implement parsing we will use a predefined method called Parse()

Parse() :-

→ It is a predefined member method of every primitive datatype structure.

→ Parse method will accept string value and, it will convert string value into given calling datatype and that converted value it will return.

79) Example for parsing:

```
Void Main()
{
    Console.WriteLine("Enter ur id:");
    Int Id = int.Parse(Console.ReadLine());
    Console.WriteLine(" Enter ur weight:");
    float weight = float.Parse(Console.ReadLine());
    Console.WriteLine(" Ur id is: " + id);
    Console.WriteLine(" Ur weight is: " + weight);
    Console.ReadLine();
}
```

80) Example:2 for parsing:

```
Void Main()
{
    String s1 = " 123 ";
    int i1 = int.Parse(s1); // valid
    String s2 = " rama ";
    int i2 = int.Parse(s2); // format exception
    String s3 = " 4.5 ";
    float f1 = float.Parse(s3); // valid
    int i3 = int.Parse(s3); // format exception
    String s4 = " 123456785543&14 ";
    int i4 = int.Parse(s4); // overflow
    String s5 = null;
    int i5 = int.Parse(s5); // Argument exception.
```

81) Implement multiple programs for different pairings

### 3. Converting :-

→ To implement converting we have to use a predefined class called Convert.

#### Convert :-

→ Convert is a predefined class which is part of system base class library.

→ Within convert class we have all the data type conversion methods

#### Difference between Pairing and converting :-

##### Pairing

1) Pairing will convert from String to any other datatype

1) Converting will convert from any data type to any other data type

#### Difference b/w int.Parse & Convert.ToInt32 :-

##### int.Parse

1) It will convert string to int.  
2) when string variable contains null value int.Parse will throw argumentnullexception;

##### Convert.ToInt32

1) It will convert any type to int.

2) when string variable contains null value convert.ToInt32 will not throw any exception & will convert the null value into zero.

### 3. Boxing and UnBoxing :

→ Boxing is a process of converting from value type to Reference type

Ex: Converting from int to object.

→ UnBoxing is a process of converting from Reference type to value type

Ex: Converting from object to int.

Note :-

→ To do Boxing and UnBoxing we can use C++ style of type casting Syntax.

Q1) Example for Boxing and unBoxing.

```
Void Main()
{
    int i = 100; // value type
    object obj = (object) i; // boxing
    Console.WriteLine("obj value is :" + obj);
    int j = (int) obj; // unboxing
    Console.WriteLine("j value is :" + j);
    console.ReadLine();
}
```

Q2) Write a console prgm to accept a range and display the numbers up to that range.

```
Program
Void Main()
{
    Console.WriteLine("Enter ur range:");
    int n = int.Parse(Console.ReadLine());
    for(int i = 1; i <= n; i++)
    {
        Console.WriteLine(i);
    }
    Console.ReadLine();
}
```

83) Console prgm to print sum of given range.

Program

ulong  
int  
n

O/P

Enter ur range:

ulong n  ← 5

ulong sum  ← Sum is : 15

Program

Void Main()

{

Console.WriteLine (" Enter ur range: ");

ulong n = ~~Convert~~.ToInt32(Console.ReadLine());

ulong sum = 0;

for (long i=1; i<=n; i++)

{

sum = sum+i;

}

Console.WriteLine (" sum is : " +sum);

Console.ReadLine();

}

84) Console prgm to accept a range & display multiplication of range.

Void Main()

{

Console.WriteLine (" Enter ur range: ");

ulong n = ulong.Parse (Console.ReadLine());

ulong mul = 1;

for (ulong i=1; i<=n; i++)

{

mul = mul \* i;

}

Console.WriteLine (" multiplication is : " +mul);

Console.ReadLine();

}

85) Console prgm to find factorial of given no;

```
Void Main()
{
    Console.WriteLine("Enter the range:");
    ulong n = ulong.Parse(Console.ReadLine());
    ulong fac = 1;
    for(ulong i = n; i >= 1; i--)
    {
        fac = fac * i;
    }
    Console.WriteLine("factorial is :" + fac);
    Console.ReadLine();
}
```

86) Console prgm to display table for given no.

int      Op  
[2]      Enter ur number:  
2  
 $2 \times 1 = 2$   
 $2 \times 2 = 4$   
 $2 \times 3 = 6$   
 $2 \times 4 = 8$   
 $2 \times 5 = 10$   
 $2 \times 6 = 12$   
 $2 \times 7 = 14$   
 $2 \times 8 = 16$   
 $2 \times 9 = 18$   
 $2 \times 10 = 20$

Program

```
Void Main()
{
    Console.WriteLine("Enter the number:");
    int n = int.Parse(Console.ReadLine());
    int tab = 1;
    for(int i = 1; i <= 10; i++)
    {
        tab = tab * i;
        Console.WriteLine(n + " * " + i + " = " + n * i);
    }
}
```

3  
    Console.ReadLine();

87) Console prgm to display no's like below.

1  
2 2  
3 3 3  
4 4 4 4  
5 5 5 5 5

Prog

Void Main()

{

    Console.WriteLine("i");

    {  
        Console.WriteLine("j");

        {  
            Console.WriteLine("i+j");

        }  
    }

    }

    {  
        Console.WriteLine("k");

    }

    Console.ReadLine();

88

Pro

88) Implement above prgm by using diff loops combinations.

89) Console prgm to print no's like below with in given range.

n  
19 Enter O/P or range:

2 → 9.

1  
2 2  
3 3 3  
4  
5  
6  
7  
8  
9 9 9 9 9 9 9 9

Void Main()

{

    Console.WriteLine("Enter the range:");

    int n = int.Parse(Console.ReadLine());

```

    foo(int i=1; i<=5; i++)
    {
        for(int j=1; j<=i; j++)
        {
            Console.WriteLine(i + " ");
        }
        Console.WriteLine();
    }
    Console.ReadLine();
}

```

Q90) Console prgm to print the no's like below within given range.

O/P

n	5
2	1 2
	1 2 3
	1 2 3 4
	1 2 3 4 5

prgm

```

void main()
{
    Console.WriteLine("Enter ur range:");
    int n = int.Parse(Console.ReadLine());
    for(int i=1; i<=n; i++)
    {
        for(int j=1; j<=i; j++)
        {
            Console.WriteLine(j + " ");
        }
        Console.WriteLine();
    }
    Console.ReadLine();
}

```

n=5

i	1	2	3	4	5
j	1	2	3	4	5

Q1) Console prgm to print no's like below within given range.

O/P

Enter ur range:

n  
5 ← s

5 5 5 5 5  
4 4 4 4  
3 3 3  
2 2  
1

Pr

Void Main()

{

Console.WriteLine("Enter ur range:");

int n = int.Parse(Console.ReadLine());

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= i; j++)

{

Console.Write(i);

y

Console.WriteLine();

y

Console.ReadLine();

}

n=5

92) Console prgm to display no's like below within given range

O/P

Enter ur range

s

1 2 3 4 5  
1 2 3 4  
1 2 3  
1 2  
1

Void Main()

{

Console.WL("Enter ur range:");

int n = int.Parse(Console.ReadLine());

```

for(int i=n; i<=1; i--)
{
    for(int j=0; j>=i; j++)
    {
        console.WriteLine(j+" ");
    }
    console.WriteLine();
}
console.ReadLine();

```

→ Console prgm to display the no's like below.

```

main()
{
    int k=1;
    for(int i=1; i<=4; i++)
    {
        for(int j=1; j<=i; j++)
        {
            console.WriteLine(k+" ");
            k++;
        }
        console.WriteLine();
    }
    console.ReadLine();
}

```

O/P

1
2 3
4 5 6
7 8 9 10

→ console program to display the no's like below within the range

```

main()
{
    console.WriteLine("enter your range");
    long n = long.Parse(console.ReadLine());
    int k=1;
    for(long i=1; i<=n; i++)
    {
        for(long j=1; j<=i; j++)
        {
            if(k<=n)
                console.WriteLine(k+" ");
            k++;
        }
    }
}

```

```
console.WriteLine();
```

{

```
console.ReadLine();
```

{

HW

1.

```
for(i=1; i<=4; i++)
    for(j=1; j<=i; j++)
        Console.Write("x");
    Console.WriteLine();
```

```
x x x x
x x x
x x x
x x x x
```

3.

```
x x x
x x x
x x x x
```

```
4. x x x x x
      x x x x
      x x x
      x x
      x
```

```
5.   1
      2 2
      3 3 3
      4 4 4 4
      5 5 5 5
```

→ console prgm to accept student name and accept  
3 sub marks they are  $m_1, m_2, m_3$  and calculate the  
total and avg, If  $\text{avg} \geq 60$  first class, if  $\text{avg}$  in  
between ( $50$  to  $< 60$ ) <sup>second class</sup>, if  $\text{avg} \geq 35$  and  $< 50$  third class  
If student is getting in any one of the sub  $< 35$  then  
result is fail.

```
main()
```

{

```
Console.WriteLine("enter student name:");
```

```
String Sname = (Console.ReadLine());
```

```
Console.WriteLine("enter m1 marks:");
```

```
int m1 = int.Parse(Console.ReadLine());
```

```
Console.WriteLine("enter m2 marks:");
```

```
int m2 = int.Parse(Console.ReadLine());
```

```

console.wL ("enter m1 marks:");
int m1 = int.Parse (console.ReadLine ());
int total = m1 + m2 + m3;
float avg = total / 3;
else if (m1 <= 35 || m2 <= 35 || m3 <= 35)
{
    console.wL (Sname + " result is fail");
}
else if (avg >= 60)
{
    console.wL (Sname + " result is 1st class");
}
else if (50 >= avg)
{
    console.wL (Sname + " result is 2nd class");
}
else
{
    console.wL (Sname + " result is 3rd class");
}
then
}

```

O/P

enter student name:

Sneha.

enter m<sub>1</sub> marks:

70

enter m<sub>2</sub> marks:

70

enter m<sub>3</sub> marks:

70

sneha result is 1st class.

Q1) write a program to print numbers between given range.

console program to accept user age & display user status

→ Write a console program to accept user age and display user status.

age  $\geq 58 \rightarrow$  senior citizen

age  $\rightarrow 25-57 \rightarrow$  working

age  $\rightarrow 16-24 \rightarrow$  college student

3-15  $\rightarrow$  school boy

1-3  $\rightarrow$  playing kid

main()

{

    Console.WriteLine("enter ur age");

    int age = int.Parse(Console.ReadLine());

    if (age  $\geq 58$ )

        Console.WriteLine("senior citizen");

    else if (age  $\geq 25$ )

        Console.WriteLine("working citizen");

    else if (age  $\geq 16$ )

        Console.WriteLine("college student");

    else if (age  $\geq 3$ )

        Console.WriteLine("school student");

    else if (age  $\geq 1$ )

        Console.WriteLine("playing kid");

    else

        Console.WriteLine("invalid age");

}

→ Write a console program to accept customer name and customer consumed units then accept customer type and calculate the electricity bill and display to the user?

Day 1 - Industry

cost per unit  $w = 7$  RS/-

Residential

cost per unit  $w = 6$  RS/-

Agricultural

cost per unit  $w = \text{free}$

Total bill =  $w \times t \times 7$

$$= 120 \times 7$$

$$= \boxed{840}$$

name

O/P

↑  
enter customer name:

← Jhon

↑  
enter customer units:

← 120

↑  
enter customer type:

press 1 for Industry:

press 2 for Residential

ctype press 3 for agricultural:

← 1 ←

Jhon total bill is 840

~~City Pe~~  
1

John total bill is: 840

Industry cost per unit = 7 Rs

Residential cost per unit = 6 Rs

Agriculture cost /unit = free.

Total bill = cunit \* ctype

$$840 = 120 \times 7$$

(20) (7)

Prgrm      void Main()

{

Console.WriteLine ("Enter customer name : ");

string cname = Console.ReadLine();

Console.WriteLine ("Enter customer units : ");

int cunits = Int.Parse (Console.ReadLine());

restart:

Console.WriteLine ("Enter customer type : " + "\n" + "press 1 for Industry" +

"\n" + "press 2 for residence" + "\n" + "Press 3 for Agriculture");

char ctype = char.Parse (Console.ReadLine());

if

switch (ctype)

{

case '1':

Console.WriteLine (cname + " John total bill is : " + ctype \* 7);  
break;

case '2':

Console.WriteLine (cname + " John total bill is : " + ctype \* 6);  
break;

case '3':

Console.WriteLine (cname + " John total bill is : " + ctype \* 0);  
break;

default:

Console.WriteLine ("Invalid input !!");

~~break~~; goto restart;

y

Console.ReadLine();

g

Q3) Implement the above prgm with diff requirement like below  
for industry

up to 100 cost/unit = 7 Rs

if ( $> 100$ )  $\rightarrow$  cost/unit = 8 Rs

for residential

up to 100 cost/unit = 6 Rs

if ( $> 100$ ) cost/unit = 7 Rs

for agriculture cost/unit = 0/-

Program

// above lines are same

Switch (ctype)

{

Case '1':

If (cunits > 100)

{ console.write (cname + " total bill is : " + cunits \* 8);  
}

else

{ console.write (cname + " total bill is : " + cunits \* 7);  
}

}

break;

case '2': if (cunits > 100){

if (console.write (cname + " total bill is : " + cunits \* 7);  
{

console.write

else

console.write (cname + " total bill is : " + cunits \* 6);  
break;

case '3':

If (cunits > 100)

console.write (cname + " total bill is : zero");  
break;

default:

console.write (cname + " Invalid type");  
goto restart;

console.ReadLine();  
}



110) Implement above prgm without temp variable.

```
Void Main()
{
    Console.WriteLine("Enter i value :");
    int i = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter j value :");
    int j = int.Parse(Console.ReadLine());
    i = i + j;
    j = i - j;
    i = i - j;
    Console.WriteLine("i value is : " + i);
    Console.WriteLine("j value is : " + j);
    Console.ReadLine();
}
```

$i = 5$   
 $j = 10$   
 $\underline{i = i + j}$   
 $= 5 + 10 \Rightarrow 15$

$j = i - j$   
 $= 15 - 10 \Rightarrow 5$

$i = i - j$   
 $= 15 - 5 \Rightarrow 10$

111) Console prgm to find biggest among 3 no's.

```
Void Main()
{
    Console.Write("Enter first no:");
    int a = int.Parse(Console.ReadLine());
    Console.Write("Enter second no:");
    int b = int.Parse(Console.ReadLine());
    Console.Write("Enter third no:");
    int c = int.Parse(Console.ReadLine());
    if(a > b & a > c)
        Console.WriteLine("a is biggest");
    else if(b > c)
        Console.WriteLine("b is biggest");
    else
        Console.WriteLine("c is biggest");
    Console.ReadLine();
}
```

Note:- Above prgm is giving wrong op when user entered same no's.

112) write the prgm to this <sup>above</sup> ~~prgm~~ problem . using nested if and also switch

113) Implement above prgm using simple if

114) console prgm to display smallest no among 3 no's.

```
Void Main()
{
    Console.WriteLine("Enter first no:");
    int a = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter second no:");
    int b = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter third no:");
    int c = int.Parse(Console.ReadLine());

    if (a > b && a > c)
    {
        if (a > b)
            Console.WriteLine("a is biggest");
        else if (a == b)
            Console.WriteLine("a & b is equal");
    }
    else if (a < b)
```

a = 5 5 8  
b = 5 8 5  
c = 8 5 5

115) Implement above smallest no prgm using nested if

116) console prgm to print biggest among 4 no's

117) smallest among 4 no's

118) second biggest among 4 no's

119) second smallest among 4 no's

120) Biggest no among 5 no's

121) Smallest no among 5 no's

122) Bigg second biggest & second smallest no

123) third biggent no.

} if else if  
nested if  
nested switch  
switch with if else

124) Console prgm is even or odd.

Example for even

2, 4, 6 - - -

Example for odd

1, 3, 5 - - -

% → Modular operator

/ → division operator.

dividend  
divisor  $\frac{2}{4}$  (2 → Quotient)  
4  
0 → remainder

→ with the help of modular operator(%) we get remainder

→ with the help of division operator(/) we get quotient.

Ex: void Main()

```
Console int i = 4%2;
Console.WriteLine(i);
int j = 4/2;
Console.WriteLine(j);
Console.ReadLine();
```

}

124)

O/p

Enter ur no:

← 4

It is even number.

Program

```
Void main()
```

{

```
Console.WriteLine("Enter your no:");
```

```
int n = int.Parse(Console.ReadLine());
```

```
If (n % 2 == 0)
```

```
Console.WriteLine("It is even number");
```

else

Console.WriteLine("It is odd number");

Console.ReadLine();

y

Q25) Console prgm to display the list of even no's within given range & display the sum of even no's & no. of even no's within that range.

O/p

n Enter the range:

← 10

List of Even no's within in 10

2.

4

6

8

10.

1 → 10

2

Sum	ctr	i
0	0	2
2	1	4
4	2	6
6	3	8
8	4	10
30		

← Sum of even numbers are : 30  
 ctr no. of even numbers are : 5

Void Main()

{

Console.WriteLine("Enter ur range:");

int n = int.Parse(Console.ReadLine());

Console.WriteLine("List of even no's within in "+n);

int sum=0; ctr=0;

for(int i=2; i<=n; i=i+2)

{

Console.WriteLine(i);

}

sum = sum + i;

ctr++;

Console.WriteLine("Sum of even numbers are :" + sum);

Console.WriteLine("No. of even numbers are :" + ctr);

Console.ReadLine();

y

(Q) 126) Implement above prgm using ~~logical~~ <sup>Modular</sup> operator.

```
Void Main()
{
    Console.WriteLine ("Enter your range:");
    int n = int.Parse (Console.ReadLine ());
    Console.WriteLine ("List of even no's are:");
    int sum=0, ctr=0;
    for (int i=2; i<=n; i+=2)
    {
        Console.WriteLine (i);
        if (n%2 == 0)
        {
            Console.WriteLine (i);
            sum = sum + i;
        }
    }
    Console.WriteLine ("Sum of even no's are: " + sum);
    Console.ReadLine ();
}
```

O/p

Enter ur range:

10

List of even no's are:

2

4

6

8

10

Sum of even no's are: 30.

(Q) 127) console prgm to display the odd no's & with in given range  
and display the sum & list of odd no's. with in range

1) without using modular operator 2) By using modular operators.

(Q) 128) console prgm to check the given no is a prime no or not.

Prime no: Number which is divisible by 1 and it self

Void Main()

{

Console.WriteLine("Enter ur number:");

int n = int.Parse(Console.ReadLine());

If (~~n <= 1~~ & n != 1)

10/10

int ctr = 0;

for (int i = 2; i < n; i++)

If (n % i == 0)

{ Console.WriteLine(

ctr++;

break;

} If (ctr == 0)

{

Console.WriteLine(" prime no");

}

else

Console.WriteLine(" not a prime no");

Console.ReadLine();

4

ff

129) Console prgm to display list of prime no's, sum of prime no's,  
no. of prime no's with in given range

2, 3, 5, 7, 11, 13, (5)

1/2 1/2 i <= n i++

Void Main()

{

Console.WriteLine("Enter ur range:");

int n = int.Parse(Console.ReadLine());

int sum = 0, ctr = 0;

for (int i = 2; i <= n; i++)

If (n % i == 0)

{ Console.WriteLine(i);

ctr++;

break;

}

15:

i <= n

2 3 5 7

sum = 0

130) console pgm to reverse the given no

O/p  
n Enter ur no:  

123
-----

 ← 123  
reverse no is : 321

Program

```
Void Main()
{
    Console.WriteLine("Enter your no:");
    int n = int.Parse(Console.ReadLine());
    int rem, rev=0;
    while(n!=0)
    {
        rem = n % 10;
        rev = rev * 10 + rem;
        n = n / 10;
    }
    Console.WriteLine("Reverse number is :" + rev);
    Console.ReadLine();
}
```

1st iteration

$$123 \rightarrow 0 \rightarrow T$$

$$\text{rem} = 123 \% 10$$

3
---

$$\text{rev} = 0 * 10 + \text{rem}$$

$$\hookrightarrow 0 = 0 * 10 + 3$$

$$\text{rem} = 3$$

$$n = 123 / 10$$

12
----

2nd iteration

$$12 \rightarrow 0 \rightarrow T$$

$$\text{rem} = 12 \% 10$$

2
---

$$\text{rev} = 3 * 10 + \text{rem}$$

$$\hookrightarrow 3 = 30 + 2$$

$$\text{rev} = 32$$

$$n = 12 / 10$$

1
---

3rd iteration

$$1 \rightarrow 0 \rightarrow T$$

$$\text{rem} = 1$$

1
---

$$\text{rev} = 32 * 10 + \text{rem}$$

$$32 = 32 * 10 + 1$$

$$\text{rev} = 321$$

$$n = 1 / 10$$

0
---

4th iteration

$$0 \rightarrow 0$$

false

131) console prgm to check whether given no's a palindrome or not

Void Main()

{

console.WriteLine("Enter your no:");

int n = int.Parse(console.ReadLine());

int rem, rev=0, temp=n;

while(n!=0)

{

rem = n % 10;

rev = rev \* 10 + rem;

n = n / 10;

}

console.WriteLine("Reverse number is: " + rev);

if(temp == rev)

console.WriteLine("It is palindrome");

else

console.WriteLine("It is not palindrome");

console.ReadLine();

y

$$n = 121$$

$$\text{temp} = n$$

$$= 121$$

$$\text{temp} = \text{rev} \rightarrow T$$

$$121 \rightarrow 121$$

while( $n \neq 0$ )

$$\text{rem} \quad \text{rev} = 0$$

$$\text{rem} = 121 \% 10$$

$$\text{rev} = \text{rev} \times 10 + \text{rem}$$

$$n = n / 10$$

132) Console prgm to reverse the given string.

```
Void Main()
{
    console.WL("Enter a name:");
    String a = console.ReadLine();
    String b = null;
    for (int i = a.length - 1; i >= 0; i--)
        console.WriteLine(a[i]);
    {
        b = b + a[i];
    }
    console.WL(b);
    console.ReadLine();
}
```

O/P

Enter a name:

rama

Reverse name is: amar

a = rama

b = 4    b <= 4

133) Console prgm to check given string is a polyndrome or not.

```
Void Main()
{
    console.WL("Enter a name:");
    String a = console.ReadLine();
    String b = null;
    for (int i = a.length - 1; i >= 0; i--)
    {
        b = b + a[i];
    }
    if (a == b)
        console.WriteLine("It is polyndrome");
    else
        console.WL("not a polyndrome");
    console.ReadLine();
}
```

mam

a = rama

b =

i = 4 - 1, i >= 0, i --

b = ram

amar

mam?

134) console program to generate fibonacci series

```
void Main()
{
    int a=0, b=1, c=0;
    while (a <= 100)
    {
        Console.WriteLine(a + " ");
        c = a+b;
        a = b;
        b = c;
    }
    Console.ReadLine();
}
```

at

$$\begin{array}{lll} a=0 & b=1 & c=1 \\ 1 & 1 & 2 \\ 2 & 3 & 5 \\ 3 & 5 & 8 \\ 5 & 8 & 13 \\ 8 & 13 & 21 \\ 13 & 21 & 34 \\ 21 & 34 & 55 \\ 34 & 55 & 89 \\ 55 & 89 & 89 \end{array}$$

Op

0 1 1 2 3 5 8 13 21 34 55 89 -

0+1 1+1 1+2=3 2+3=5 3+5=8

## Object oriented programming Systems :-(OOPS)

Q) What is Object Oriented approach?

Ans: Object oriented approach is a methodology to develop computer programs by using classes and objects.

Q) what do you mean by object <sup>oriented</sup> programming language?

Ans: A programming language which will follow object oriented approach is called as object oriented programming language  
Ex: C++, C# .Net, VB .Net.....

Q) what do you mean by Object oriented programming principles?

Ans → OOPS principles are 4

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism.

→ Every object oriented programming language has to follow the above 4 principles.

→ To achieve the OOPS principles every object oriented programming language will use two concepts

1. class
2. object

### 1. Class :-

→ class is a collection of data members and member functions.

→ Data member can be called as state (or) variable (or) field.

Q) what is state?

Ans State will represent some value.

→ Member function can be called as method (or) function. (or) behaviour.

Q) what is behaviour?

Ans Behaviour is nothing but functionality (or) logic.

- Finally we can say class is a collection of states and behaviours
- To define a class we have to use class keyword
- <Syntax to define class>:-

↗ Keyword

```

<access modifier> class <classname>
{
    // states // variables
    // behaviours // methods
}

```

→ Class is a logical representation, when we define class no memory will be allocated for class members.

→ Q) List out the states & behaviours of human being:

States:

- name
- age
- height
- weight

methods:

- eating
- speaking
- walking

→ Q) List out the states & behaviours of employee & student & customer

Employee

states:-

- Salary
- Position

methods

Student

states

- Marks
- class

methods

- fee payment
- attendance calc.

Customer

states

methods

- bill payment

Q) When we will go for state?

Ans: Whenever we want to represent some value we will go for state.

Ex: empno, emp name, empage - - -

### Types of Variables:-

→ C# .Net will support 3 types of variables

1. Local variable.
2. Instance variable (or) Non static variable
3. Static variable.

#### 1. Local Variable :-

→ A variable which is declared within a method is called as Local Variable.

→ Local Variable should be initialized with some value before accessing otherwise compiler will generate an error.

#### 2. Instance Variable (or) Non-static variable :-

→ A variable which is declared inside class and outside the method without static keyword is called instance variable.

#### 3. static variable :-

→ A variable which is declared inside class and outside the method by using static keyword is called as static variable.

Q) When we will go for method (or) behavior?

Ans: Whenever we want to implement some functionality we will go for method or behaviour.

Ex: calculating salary, calculating age (or) calculating marks - - -

→ Syntax to define a method:

<Access modifier> <return type> <method name <type> <arg1>, <type>  
<arg2>>



Note: → Argument and parameters both are same.

→ According to parameters, methods are classified into 2 types.

1. Parameter less method
2. Parameterized method

### 1. Parameter less method

→ while defining a method if we don't declare any parameters which is called as Parameter less method.

Q) When we will go for parameter less method?

Ans → According to method functionality if the method doesn't required any value then we will go for parameter less method.

Ex: class Program  
 {  
 void Display()  
 {  
 Console.WriteLine("Welcome to C# Net");  
 Console.WriteLine("Welcome to ASP.NET");  
 }  
 void Main()
 }

### 2. Parameterized method

→ While defining a method if we have declared parameter which is called as parameterized method.

Q) When we will go for parameterized method?

Ans According to requirement whenever a method is required some value for its functionality then we go for parameterized method.

Ex: class Program  
 {  
 void add(int a, int b)  
 {  
 int c = a + b;  
 Console.WriteLine("Result is :" + c);
 }
 }

→ According to method return types methods are classified into 2 types

1. Void Method

2. Non-Void Method.

1. Void Method:

→ A method which is not returning any value according to its functionality is called as void method

→ void Method return type should be void.

Q) When we will go for void Method?

Ans According to the requirement if the method doesn't require to return any value we will go for void Method.

2. Non-Void Method:

→ A method which is returning some value is called Non-Void method.

Q) When we will go for non-Void Method?

Ans → According to method functionality if it requires to return some value we will go for non-Void method.

→ The return type of method will be depending on type of the value which is returning by the method.

→ If the method is returning requires to return numerical value then method return type can be any one of the numerical data type like int, long, --

→ If the method is returning value is floating value then method return type can be float, double, decimal .

Eg: int fun1 (int a)

{

    return a;

}

```
char fun2(char a)
{
    return a;
}
```

→ Again Method are classified into 2 types

1. Instance Method (a) Non-static Method
2. Static Method

### 1. Instance Method:

- While defining a method if we don't use static keyword which is called as instance method.
- If we want to access instance methods we have to access with the help of object

```
Ex: bool fun3(bool a)
{
    return a;
}
```

### 2. Static Method

- While defining a method if we are using static keyword which is called as static method.

- If we want to access static method we have to access with the help of class name.

### Memory allocation:-

- No memory will be allocated for class
- No memory will be allocated for methods
- Memory will be allocated for variables

### Memory allocation for instance variable:-

- When the object is created for a class concern class instance variables will be allocating memory.
- When an object is destroyed concern class instance variables memory will be deallocated ~~memory~~.

- In .Net object will be destroyed in 2 ways
  - 1) by garbage collector
  - 2) by Programmer.

### Memory allocation for local variables :-

- As part of method execution memory will be allocated for local variable.
- When the method execution is completed local variable are destroyed.

### Structure of a class with instance variables, instance method, local variables :

```

* class Myclass
{
    int a; } → instance variables
    int b; } → instance method
    void add(int x, int y)
    {
        a = x;
        b = y;
        int res = a + b;
    } } → local variable
}

```

#### Note:-

- Parameter will be acting like a local variable.
- In the above class we have a method called add which is consuming 2 instance variables which are a, b
- If we want to access above my class add method, it requires to allocate memory for instance variables. They are a, b. For this we require Object.

## Purpose of object:

- To allocate memory for instance variables.

## What is an Object:-

- An object is a instance of a class which is physical representation of a class.
- For example. human being is a class , Ravi is a object of human being class.
- When we create an object for a class memory will be allocated for concern class instance variables.

## What object will contain:

- An object can contain concern class instance variable and concern class instance methods references.
- Syntax to create a object :-

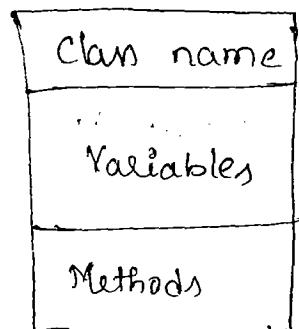
`<classname> <objectname> = new <classname()>;`

- Syntax to invoke instance methods:-

`<Objectname> • <methodname()>;`

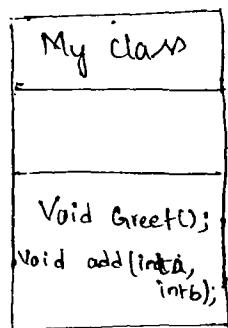
## Programming in oops:-

Structure of the UML class diagrams.



Q) Write a console program to define a class with two instance methods.

- 1) Void Method without parameters
- 2) Void Method with parameters



O/P

Welcome to OOPS.

Addition result is: 15

Prog

namespace class Example1

{

class MyClass

{

internal Void Greet()

{

Console.WriteLine("Welcome to OOPS");

}

internal Void add(int a, int b)

{

int a=5;

b=10;

int res=a+b;

Console.WriteLine("Result is:" + res);

}

y

Class program

{

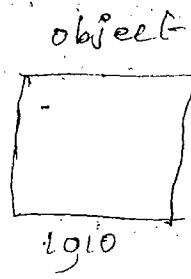
Void Main()

{

```

My class obj = new MyClass();
obj.Greet();
obj.Add(10,5);
console.ReadLine();
}
// main
}
// class
}
// namespace.

```



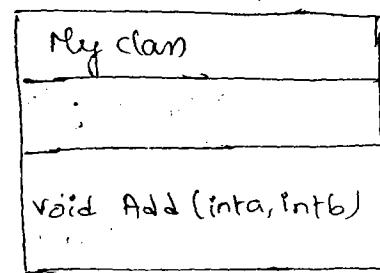
2) Console program Example to above program by passing a,b values by user.

Op

Addition result is :

X      20

Enter first no : 20



Enter second no :

Y      10

Addition result is : 30

Program

class MyClass

{

    internal void Add (int a, int b)

{

        int res = a+b;

        Console.WriteLine("Addition result is :" + res);

}

}

class program

{

    Void Main()

{

        Console.WriteLine("Enter first no: ");

        int x = int.Parse (Console.ReadLine());

```

        Console.WriteLine("Enter second no:");
        int y = int.Parse(Console.ReadLine());
        Myclass obj = new Myclass();
        obj.Add(x, y);
        Console.ReadLine();
    }
}
}

```

3) Implement the above Add method as a non-void method.

```

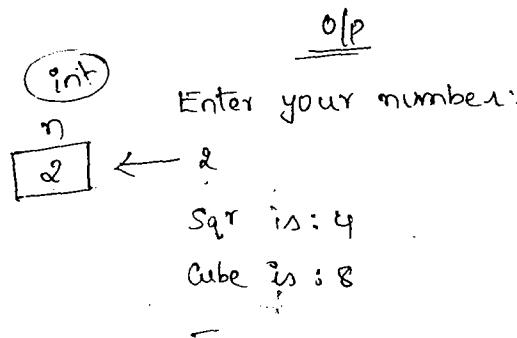
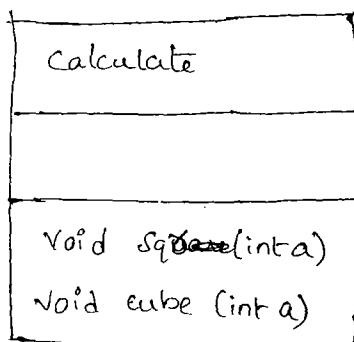
class Myclass
{
    internal int add(int a, int b)
    {
        int res = a + b;
        return res;
    }

    void Main()
    {
        Console.WriteLine("Enter first no:");
        int x = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter second no:");
        int y = int.Parse(Console.ReadLine());
        int z =
        Myclass obj = new Myclass();
        int i = obj.Add(x, y);
        Console.WriteLine("Addition result is: " + i);
        Console.ReadLine();
    }
}

```

4) Console prgm to calculate the following two.

- 1) Calculating Square
- 2) calculating cube.



```
namespace ConsoleExample4.  
{  
    class calculate  
    {  
        internal void Sqr(int a)  
        {  
            int b = a * a;  
            Console.WriteLine("Sqr is :" + b);  
        }  
  
        internal void cube(int a)  
        {  
            int c = a * a * a;  
            Console.WriteLine("Cube is :" + c);  
        }  
    }  
  
    class program  
    {  
        void Main()  
        {  
            Console.WriteLine("Enter ur number:");  
            int n = int.Parse(Console.ReadLine());  
            calculate obj = new calculate();  
            obj.Sqr(n);  
            obj(cube(n));  
            Console.ReadLine();  
        }  
    }  
}
```

5) Implement above 2 methods as non void Methods

namespace Examples  
{

Calcel

internal class calculate  
{

internal int Sqr(int a)  
{

int b = a\*a;

return b;

}

internal int cube(int a)

{

int c = a\*a\*a;

return c;

}

class program  
{

void Main()

{

Console.WriteLine("Enter ur no:");

int n = int.Parse(Console.ReadLine());

calculate obj = new calculate();

int i = obj.Sqr(n);

Console.WriteLine("Square result is:" + i);

int j = obj(cube(n));

Console.WriteLine("cube result is:" + j);

Console.ReadLine();

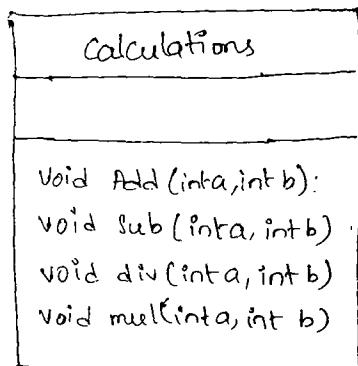
}

}

}

6) Console prgm to perform following operations

1. addition
2. Subtraction
3. division
4. multiplication



DIP

Enter first no:

10

Enter second no:

5

Addition result is : 15

Sub. result is : 5

Div result is : 2

Mul result is : 50

Program namespace Example6

{

class Calculations

{

internal void Add(int a, int b)

{

int res = a+b;

Console.WriteLine("Addition result is :" + res);

y

internal void Sub(int a, int b)

{

int res = a-b;

Console.WriteLine("Sub result is :" + res);

}

internal void Div(int a, int b)

{

int res = a/b;

Console.WriteLine("Div result is :" + res);

}

internal void Mult(int a, int b)

{

int res = a\*b;

Console.WriteLine("Mul result is :" + res);

}

class programs

```
void Main()
{
    Console.WriteLine("Enter first no:");
    int x = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter second no:");
    int y = int.Parse(Console.ReadLine());
    Calculations obj = new Calculations();
    obj.add(x,y);
    obj.Sub(x,y);
    obj.div(x,y);
    obj.Mult(x,y);
    Console.ReadLine();
}
```

6) Implement above prgm to define 4 methods as non-void method.

namespace Example6

{

class Calculations

{

internal int add(int a, int b)

{

int res = a+b;

return res;

}

internal int Sub(int a, int b)

{

int res = a-b;

return res;

}

internal int div(int a, int b)

{

int res = a/b;

return res;

}

```

internal int real(int a, int b)
{
    int res = a * b;
    return res;
}

class program
{
    void Main()
    {
        Console.WriteLine("Enter first no.");
        int x = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter second no.");
        int y = int.Parse(Console.ReadLine());
        Calculations obj = new Calculations();
        int res = obj.add(x, y);
        Console.WriteLine("Addition is :" + res);
        int res = obj.Sub(x, y);
        Console.WriteLine("Sub is :" + res);
        int res = obj.div(x, y);
        Console.WriteLine("Div is :" + res);
        int res = obj.real(x, y);
        Console.WriteLine("Real is :" + res);
        Console.ReadLine();
    }
}

```

7) Implement above prgm by defining div method for validating the second number should not be zero.

Prgrm namespace example7
{
 class Calculations
 {
}

```

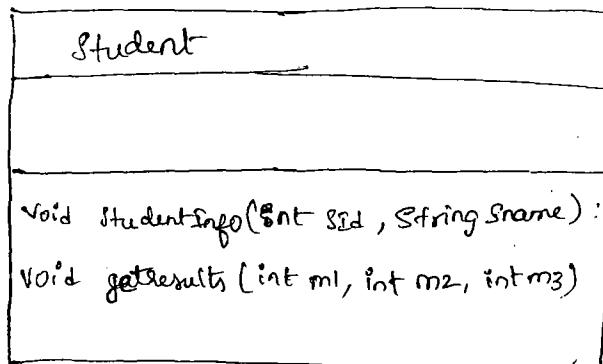
internal int div(int a, int b)
{
    while (b == 0)
    {
        console.WriteLine("Enter other than zero");
        b = int.Parse(console.ReadLine());
    }
    return a / b;
}

class program
{
    void main()
    {
        console.WL("Enter first no:");
        int x = int.Parse(console.ReadLine());
        console.WL("Enter second no:");
        int y = int.Parse(console.ReadLine());
        calculations obj = new calculations();
        int res = obj.div(x, y);
        Console.WriteLine("div result is :" + res);
        console.ReadLine();
    }
}

```

8) Define class called student with two behaviors:

1. Display student info
2. calculating student result.



## namespace Example 8:

```
class Student
{
    internal void StudentInfo(int sid, string sname)
    {
        Console.WriteLine("Student id is:" + sid);
        Console.WriteLine("Student name is :" + sname);
    }

    internal void studentResults(int m1, int m2, int m3)
    {
        Console.WriteLine("M1 marks is :" + m1);
        Console.WriteLine("M2 Marks is :" + m2);
        Console.WriteLine("M3 marks is :" + m3);
        if (m1 < 35 || m2 < 35 || m3 < 35) int tot = m1 + m2 + m3;
        else int avg = tot / 3;

        if (avg <= 60)
            Console.WriteLine("Fail");
        else if (avg >= 60)
            Console.WriteLine("First class");
        else if (avg >= 50)
            Console.WriteLine("Second class");
        else
            Console.WriteLine("Third class");
    }
}

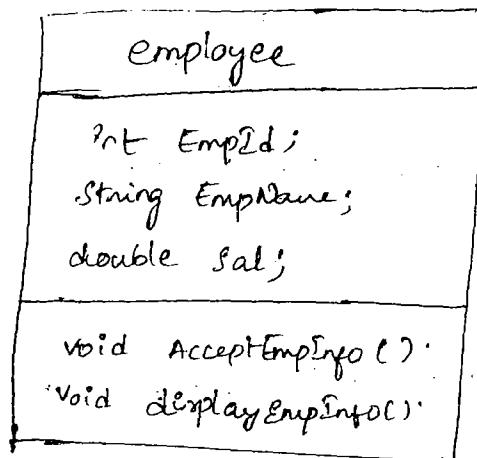
class program
{
    void Main()
    {
        Student obj = new Student();
        Console.WriteLine("Enter student id : ");
        int sid = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter student name : ");
        string sname = Console.ReadLine();
        obj.StudentInfo(sid, sname);

        // Entering m1, m2, m3 marks by user
        obj.studentResults(m1, m2, m3);
        (Console.ReadLine());
    }
}
```

Q) Define a class called employee with two functions.

1. accepting empId, empName, empSal.
2. Displaying empId, empName, empSal.

Q) what is



Program

namespace instanceEx1

{

Class employee

{

    int EmpId;

    String EmpName;

    double Sal;

    internal void AcceptEmpInfo()

{

        Console.WL("Enter Employee ID:");

~~int~~ EmpId = int.Parse(Console.ReadLine());

        Console.WL("Enter Empname:");

        Empname = Console.ReadLine();

        Console.WL("Enter empSal:");

        double

        sal = ~~int~~.Parse(Console.ReadLine());

}

    internal void displayEmpInfo()

{

        Console.WL("Employee Id is :" + EmpId);

        Console.WL("Employee name is :" + empname);

        Console.WL("Salary is :" + sal);

}

```

class program
{
    void Main()
    {
        employee obj = new employee();
        obj.AcceptEmpInfo();
        obj.DisplayEmpInfo();
        employee obj2 = new employee();
        obj2.AcceptEmpInfo();
        obj2.DisplayEmpInfo();
        Console.ReadLine();
    }
}

```

- In the above example we have declared 3 instance variables
  - 1. empid & empname & empSal.
- Instance variable will be related to instance object.
- That means memory allocation and memory deallocation will be at the time of creating object and deallocation is at the time of destroying object.
- Similarly instance variable should be initialized as part of object creation.
- But in the above example instance variables are initializing after creating the object because we have used a mechanism called method to initialize instance variables.
- At the time of creating object to initialize instance variable we have a separate mechanism called constructor.

### Constructor:

- Constructor is a member of a class which is a special type of function
- Constructor will invoke automatically when an object is created
- Constructor will not return any value so it will not have return type.

- class name and constructor name should be same.
- Constructor can contain parameters
- Syntax to define constructor :-

<access modifier> <classname()>

{

// initialization code.

}

### Q) Example to understand constructor Execution

namespace constructorEx

{

class MyClass

{ // Instance Variables

int a;

int b;

internal MyClass()

{

Console.WriteLine("constructor is calling");

a=10;

b=20;

}

// Instance Method

internal void display()

{

Console.WriteLine("a value is :" + a);

Console.WriteLine("b value is :" + b);

}

Class program

{

Void Main()

{

Console.WriteLine("Before object is created");

`MyClass obj = new MyClass();`  
 Step 1      Step 2      Step 3  
 Console.WriteLine("After object creation");  
 Obj. Display();  
 Console.ReadLine();

3  
4

dp

Before object is created,

Constructor is calling

After object creation

a value is \$10.

b value is \$20.

`MyClass obj = new MyClass();`  
 Step 1      Step 2      Step 3      Step 4

The above statement will execute in 4 steps.

Step 1: Creating reference variable

Step 2: Creating object [as part of this memory is allocating for instance variables]

Step 3: Invoking constructor & will execute constructor [as part of constructor execution it will initialize default values to instance variables & after that it will initialize the given values to instance variables].

Step 4: Assigning object address into reference variable.

## Types of Constructors:-

→ Constructors are mainly 2 types

1. Instance constructor (or) non-static constructor
2. Static constructor.

### 1. Instance Constructor:

→ While defining a constructor if we don't use static keyword it is called as instance constructor.

→ Purpose of instance constructor is to initialize instance variables.

→ Within instance constructor, we can initialize instance & static variables.

→ But if we will initialize static variables within instance constructor that static variables will lose the static nature.

→ Instance constructor can contain parameters.

→ Instance constructors are classified into 4 types.

1. Default constructor (or) parameter less constructor
2. Parameterized constructor
3. Copy constructor
4. Private constructor.

### 1. Default Constructor:

→ If a constructor is not having any parameters, which is called as parameter less constructor.

→ A class can contain only one default constructor.

→ Default constructors are two types

1. User defined default constructor
2. System defined default constructor.

### 1. User defined default constructor:

→ If a programmer defines a constructor which is not having any parameters is called as user defined default constructor.

## ii) Example for userdefined default constructor

name space UserDefined Default Constructor

{

class Employee

{

int eno;

String ename;

double esal;

internal Employee()

{

eno = 111;

ename = "rama";

esal = 1000;

}

internal void Display()

{

Console.WriteLine("Employee no is :" + eno);

Console.WriteLine("Ename is :" + ename);

Console.WriteLine("Esal is :" + esal);

}

}

Class program

{

void Main()

{

Employee obj = new Employee();

obj.Display();

Console.ReadLine();

}

}

### drawback :

- Default constructor will initialize the same values to all the objects.
- To overcome this we will go for parameterised constructor.

### Purpose of default constructor

- In real time we will not use default constructor to initialize instance variables.
- we will use default constructor to declare the connection string and soon.

### 2. System defined default constructor:

- when we compile the program within a class, if programmer didn't define any instance constructor, then compiler will generate one default constructor which is called as system defined default constructor.

Ex: namespace Systemdefineddefaultconstructor

{

Class Employee  
{

int eno;  
string ername;  
double esal;

External void Display()

{

Console.WL("Emp No is :" + eno);  
Console.WL("Ename is :" + ername);  
Console.WL("Esal is :" + esal);

}

}

Class Program

{

Void Main()

{

internal Employee()  
{  
eno = 0;  
esal = 0;  
ername = null;

→ system defined default constructor

```
Employee emp1 = new Employee();
emp1.Display();
Console.ReadLine();
```

o/p

Emp no is : 0

Ename is : ~~Abhi~~

Esal is : 0

Purpose of System defined default constructor:

→ To Initialize default values to the instance variables

2. Parameterized Constructors:

→ While defining a constructor if we have declared parameters which is called as parameterized constructor.

→ A class can contain multiple Parameterized Constructors but we should differentiate with no. of parameters (a) Order of type of Parameters (b) Order of parameters is nothing but diff

→ For Example signature like below.

Class employee:

{

int eno;

String ename;

double esal; // 1. Parameterized constructor

internal Employee (int Eno)

{ eno = Eno;

}

internal Employee (String Ename)

{ ename = Ename;

}

internal Employee (double Esal)

{ esal = Esal;

}

(Q) What signature will represent?

Ans Signature will represent no. of parameters, type of parameters, order of parameters.

### What is constructor overloading

→ Defining multiple constructors with different constructors with different signature in a single class is called as constructor overloading.

Purpose of parameterized constructor (Q) When we will go for parameterized constructor:-

→ To initialize different values, Object to object we will go for parameterized constructor.

→ Example for parameterized constructor

namespace parameterconst

{

class parameterconst

{

int eno;

String ename;

double esal;

internal parameterconst(int eno, String ename, double esal)

{ eno = eno; ename = ename; esal = esal; }

internal void display(<sup>Employee</sup>)

{ Console.WL("Enter Employee no: " + eno); }

Console.WL("Employee name: " + ename);

Console.WL("Employee salary: " + esal);

}

}

Class program

{

void Main()

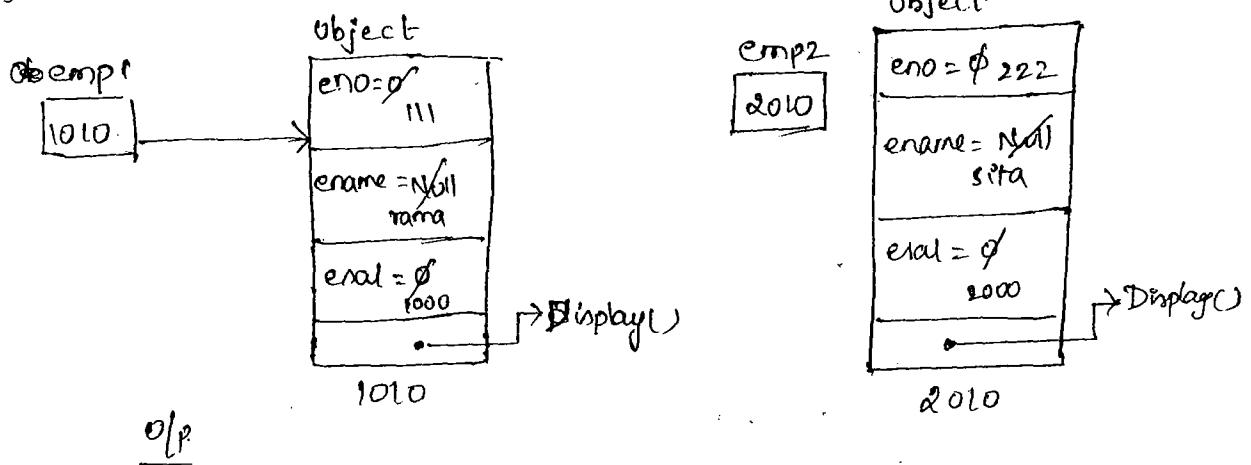
}

```

Employee emp1 = new Employee(111, "rama", 1000);
emp1.Display();
Console.WriteLine("Object address is :" + emp1.GetHashCode());
Employee emp2 = new Employee(222, "sita", 2000);
emp2.Display();
Console.WriteLine("Object address is :" + emp2.GetHashCode());
Console.ReadLine();

```

y  
y  
y



O/p

Employee no : 111

Employee name : rama.

Employee salary : 1000

Q) How to get the address of object.

Ans → By using GetHashCode().

→ GetHashCode is a predefined member method of Object class

→ This method will return the given object address

Q) What is the Super class for all .Net classes.

Ans Object

```
class object
{
    int GetHashCode()
    {
        y
    }

    class Employee : object
    {
        y

        class Program : object
        {
            y
        }
    }
}
```

### 3. Copy Constructor:

→ Using copy constructor we can copy values from one object to another object.

→ Example for copy constructor.

```
namespace copy constructor
{
    class Employee
    {
        int eno;
        string ename;
        double esal;

        internal Employee(int Eno, String Ename, double Esal)
        {
            eno = Eno;
            ename = Ename;
            esal = Esal;
        }
    }
}
```

//Copy Constructor

Internal Employee(Employee obj)

{  
eno = obj.eno;

ename = obj.ename;

esal = obj.esal;

}

Internal void Display()

{

Printing values

}

}

Class program

{

void Main()

{

Employee emp1 = new Employee(111, "Rama", 1000);

emp1.Display();

Employee emp2 = new Employee(emp1);

emp2.Display();

Console.RL();

}

}

}

Q) When we will go for copy constructor?

Ans whenever we want to copy the values from one object to another object we will go for copy constructor.

#### 4. Private Constructor :-

- while defining a constructor if we have used private access modifier that constructor is called private constructor.
- If a class is having private constructor, we cannot create an object for concern class.
- Example for private constructor.

```
namespace PrivateConstructor  
{
```

```
    class Employee
```

```
    {
```

```
        private Employee()
```

```
    {
```

```
    }
```

```
    class Program
```

```
{
```

```
    void Main()
```

```
{
```

```
        Employee empl = new Employee();
```

```
        Console.ReadLine();
```

```
}
```

```
}
```

```
}
```

Note : The above code cannot create an object, because we have an private constructor. So it will throw a runtime error.

## Static Constructor :-

- while defining a constructor if we have used static keyword which is called as static constructor.
- static constructor purpose is to initialize static variables when the memory will be allocated for static variables:-
- when the class is loading memory will be allocated for static variables.
- Memory will be deallocated for static variables when the class is unloaded.
- When the class is loading first memory will be allocated for static variable then static constructor will execute.
- If a class is having one static constructor and one instance constructor, first control will execute static constructor then will execute instance constructor becoz class will load first then object will create.
- Static constructor cannot contain parameters which is by default parameter less constructor.
- A class can contain maximum one static constructor.
- While defining static constructor we should not use access modifier.
- Within the static constructor we can initialize only static variables we cannot initialize instance variables becoz static constructor will execute before creating the object.

→ Example for static constructor.

```
namespace staticConstructor
{
    class Employee
    {
        int Eno;
        string Ename;
        double Esal;
        static string CompanyName;

        static Employee()
        {
            CompanyName = "Microsoft";
        }

        internal Employee(int EmpNo, string EmpName, double EmpSal)
        {
            Eno = EmpNo;
            Ename = EmpName;
            Esal = EmpSal;
        }

        internal void Display()
        {
            Console.WriteLine("Employee No is :" + Eno);
            Console.WriteLine("Employee Name is :" + Ename);
            Console.WriteLine("Employee Salary is :" + Esal);
            Console.WriteLine("Company name is :" + CompanyName);
        }
    }

    class Program
    {
        void Main()
        {
            Employee emp1 = new Employee(111, "John", 1000);
            emp1.Display();
        }
    }
}
```

```

Employee emp2 = new Employee(222, "David", 2000);
emp2.Display();
Console.ReadLine();

```

3  
3

O/p

Employee number is: 111

Employee name is: John

Employee salary is: 1000

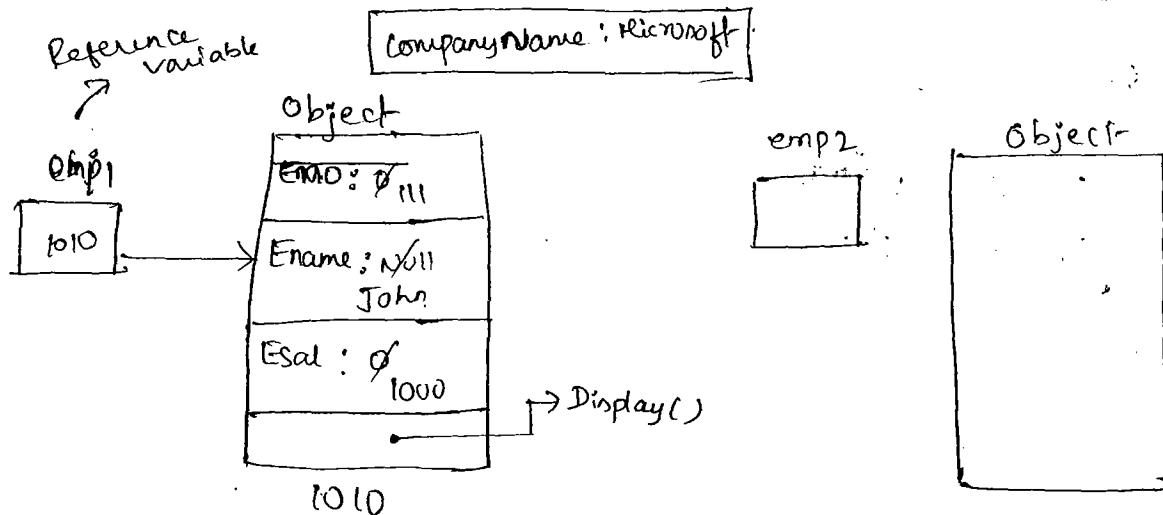
Company name is: Microsoft

Employee number is: 222

Employee name is: David

Employee salary is: 2000

Company name is: Microsoft



- When we will go for static variable?
- Whenever a value is common for all the objects then we can declare particular variable as static variable.
- What is the advantage of static variable?  
We can save the memory.

→ Example to understand Static Constructor & instance constructor execution

```
program
namespace ConstructorEx1
{
    class MyClass
    {
        int a;
        static int b;
        static MyClass()
        {
            Console.WriteLine("Static constructor is calling");
            b = 10;
        }
        internal MyClass(int x)
        {
            Console.WriteLine("Instance constructor is calling");
            a = x;
        }
        internal void Display()
        {
            Console.WriteLine("a value is :" + a);
            Console.WriteLine("b value is :" + b);
        }
    }
    class Program
    {
        void Main()
        {
            //Console.WriteLine("Before object is created");
            MyClass mc = new MyClass(10);
            mc.Display();
            Console.ReadLine();
        }
    }
}
```

O/P

Static constructor is calling.

Instance constructor is calling

a value is: 10

b value is: 20

→ Example to initialize static variable & instance variable within the instance constructor.

Program

namespace ConstructorEx2

{

class MyClass

{

int a;

static int b;

internal MyClass(int x)

{

a = x;

b = 20;

}

// internal void Display()

{

Console.WriteLine("a value is:" + a);

Console.WriteLine("b value is:" + b);

}

}

class Program

{

Void Main()

{

MyClass obj1 = new MyClass(10);

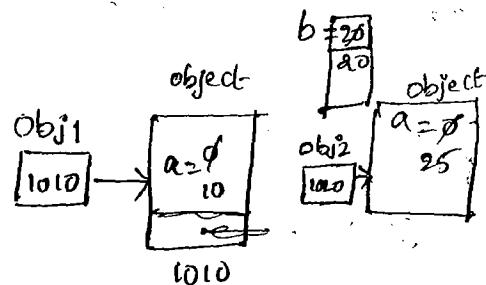
MyClass obj2 = new MyClass(20);

Console.ReadLine();

}

}

}



- In the above example we have initialized a static variable called b within instance constructor, because of that for two objects two times the value is initializing to static variable.
- But the static variable nature is it has to create once & it have to initialize once.
- Because of initializing static variable within the instance constructor we are losing the static nature.
- Example for static constructor with parameter.

```

class MyClass
{
    static int a;

    static MyClass (int x)
    {
        a = x
    }
}

class program
{
    void Main()
    {
        MyClass obj = new MyClass();
        Console.ReadLine();
    }
}

```

Note: The above code will generate an error because static constructor ~~is~~ should not have parameters

## Importance of this Keyword:-

→ this is a keyword which is representing current class instance or object.

Q) When we will use this keyword?

Ans Whenever we want to access current class instance members within the class by using object we can use this keyword

→ Syntax :-

this. <instance member>;

→ Example for this keyword.

Program  
namespace thiskeyword  
{

class myclass

{

    int a; } → instance variables  
    int b; } → instance variables

    internal myclass (int a, int b)

        { local variables }

            {  
                a = a;  
                b = b; } → local variables  
                {  
                    instance  
                    variables  
                {  
                        y  
                    }

    internal void add()

        { local variable }

        int c = a + b;

        {  
            instance  
            variables  
        }

    Console.WriteLine("Addition result is :" + c);

    {  
        y  
    }

class Program

{

    void Main()

{

```
MyClass obj = new MyClass(10, 5);
```

```
obj.add();
```

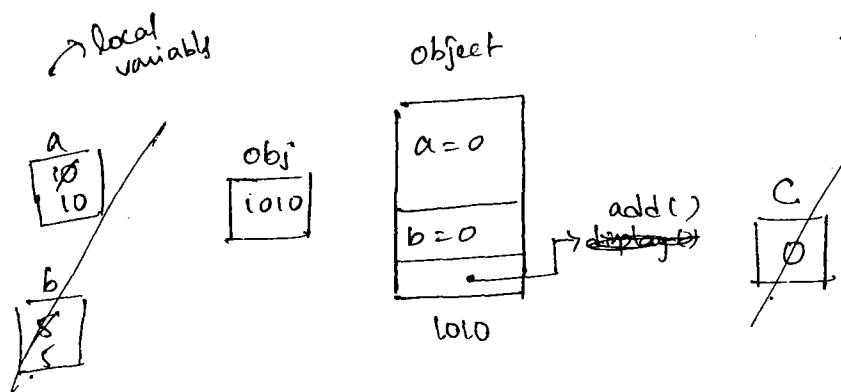
```
Console.ReadLine();
```

```
}
```

```
3
```

O/P

Addition result is : 0



Note : To overcome above problem we have to access a and b instance variables by using this keyword like below.

```
internal MyClass (int a, int b)  
{  
    this.a = a;  
    this.b = b;  
}
```

Note :-

- While initializing the instance variable with the help of Parameterized constructor, according to coding standards instance Variable name and parameter name should be same.
- To differentiate instance variable with parameters we should use this keyword like above.

(Q) When object class default constructor will invoke?

Ans → Object is a predefined class and which is Super class for all .Net classes.

→ Within object class Microsoft defined a default constructor like below.

```
class Object
{
    public Object()
    {
        // Body of constructor
    }
}
```

→ In .Net when we will invoke any instance constructor, automatically it will invoke object class default constructor.

→ For example

```
namespace ObjectClassConstructor
{
    class MyClass
    {
        int a;

        internal MyClass()
        {
            Console.WriteLine("Default constructor is calling");
        }

        internal MyClass(int a)
        {
            this.a = a;
            Console.WriteLine("Parameterized constructor is calling");
        }
    }

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

```

Void Main()
{
    Myclass Obj1 = new Myclass();
    Myclass Obj2 = new Myclass(10);
    Console.ReadLine();
}
}
}

```

→ When we compile the above prgm compiler will add base of keyword within every instance constructor declaration like below

```

internal myclass():base()
{
}
internal myclass(int a):base()
{
}

```

base :- → base represents Super class instance.

→ base()

base() :-

→ It represents Super class default constructor.

→ For all .Net classes Super class is object.

→ When we run CLR is executing below statement

```
Myclass Obj1 = new Myclass();
```

→ Here control will go to Myclass default constructor like below

```

internal Myclass():base()
{
}

```

- In declaration because of base() keyword it will go to object class and will execute object class default constructor
- After executing object class default constructor it will come back to Myclass and will execute myclass default constructor.
- While executing below statement
  - Myclass obj2 = new Myclass(10);
- It will go to myclass parameterized constructor like below
  - internal Myclass (int a) : base()
  - {
  - }
- Here within the constructor declaration because of base() keyword it will go to execute first object class default constructor then it will execute Myclass parameterized constructor.
- With the above constructors calling and execution we can say that constructor calling will be bottom to top and constructor execution will be top to bottom.
- Example to create object in two lines.

```
namespace      objectex1
{
    class Myclass
    {
        internal int a=10;
    }
    class Program
    {
    }
```

```
Void Main()
```

```
{  
    MyClass
```

```
    MyClass Obj;
```

```
    Obj = new MyClass();
```

O/P

```
    Console.WriteLine(Obj.a);
```

:10

```
    Console.ReadLine();
```

```
}
```

```
}
```

```
}
```

→ Example to hold one object address by two reference variables.

```
namespace ObjectEx2
```

```
{
```

```
    class MyClass
```

```
{
```

```
    int a;
```

```
    internal int a { get; set; }
```

```
}
```

```
class program
```

```
{
```

```
    Void Main()
```

```
{
```

```
    MyClass Obj1 = new MyClass();
```

```
    MyClass Obj2 = Obj1;
```

```
    Console.WriteLine("Obj1 value is :" + Obj1.a);
```

```
    Console.WriteLine("Obj2 value is :" + Obj2.a);
```

```
    Console.ReadLine();
```

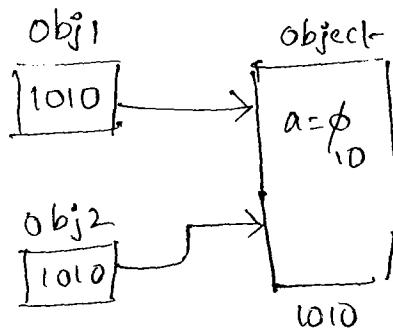
```
}
```

```
}
```

O/P

Obj1 value is :10

Obj2 value is :10



→ Example for object

```

Program namespace objectex3
{
    class Myclass
    {
        internal int a=10;
    }

    class program
    {
        void main()
        {
            Myclass obj1 = new Myclass();
            Myclass obj2 = obj1;
            obj1.a=20;
            console.WL ("obj1 value is :" +obj1.a);
            console.WL ("obj2 value is :" +obj2.a);
            console.RL();
        }
    }
}

```

O/P

obj1 value is : 20  
 obj2 value is : 20

→ Example for object

namespace objectex4

{

class Myclass

{

internal int a=10;

}

class program

{

void Main()

{

Myclass obj1 = new Myclass();

Myclass obj2 = ~~obj1~~ new Myclass();

obj1 = null;

// Console.WriteLine("obj1 value is :" + obj1.a); // it will throw null  
Console.WriteLine("obj2 Value is :" + obj2.a); reference exception

Console.ReadLine();

}

g

3

Q) When Null reference exception will throw.

Ans whenever we are accessing an object with the help of a reference variable which contains Null value.

→ Example for object

namespace objectex5

{

class Myclass

{

internal int a=10;

}

class program

```
{  
    void Main()  
    {  
        Myclass obj1;  
        Console.WL ("obj1 value is :" + obj1.a);  
        Console.RL();  
    }  
}
```

→ obj is compile time error becoz we are not initialized object for reference variable which is local.

## Static Method :-

Q) When we will go for static method?

Ans → According to the requirement whenever we want to access all static variables we will go for static method.

→ According to the requirement whenever we want to display static text we will go for static method.

→ for ex help(), we will go for static method.

→ Example for staticmethod.

namespace StaticmethodEx.

```
{  
    class employee  
    {  
        int Eno;  
        String Ename;  
        double Esal;  
        static String CompanyName;  
        static String Comploc;  
    }  
}
```

```
static Employee()
```

```
{
```

```
    CompanyName = "Microsoft";
```

```
    CompLoc = "Manikonda";
```

```
}
```

```
internal Employee(int Eno, string Ename, double Esal)
```

```
{
```

```
    this.Eno = Eno;
```

```
    this.Ename = Ename;
```

```
    this.Esal = Esal;
```

```
}
```

```
internal void Employeeinfo()
```

```
{
```

```
    Console.WriteLine("Employee No is :" + Eno);
```

```
    Console.WriteLine("Employee name is :" + Ename);
```

```
    Console.WriteLine("Salary is :" + Esal);
```

```
}
```

```
internal static void CompInfo()
```

```
{
```

```
    Console.WriteLine("Company name is :" + CompanyName);
```

```
    Console.WriteLine("Company location is :" + CompLoc);
```

```
}
```

```
class program
```

```
{
```

```
    void main()
```

```
{
```

```
        Employee Emp1 = new Employee(111, "david", 1000);
```

```
        Emp1.Employeeinfo();
```

```
        Employee.CompInfo();
```

```
        Console.ReadLine();
```

```
}
```

→ Example for predefined static methods

WriteLine()

Write()

ReadLine()

ToInt32()

Parse()

→ Example to access instance variable with in static method.

namespace StaticMethodEx2

{

class Myclass

{

~~static~~ int a = 10;

internal void show()  
in console.WL("instance method  
constructor is calling");

internal static void display()

{ Myclass obj = new Myclass();

obj.show();

console.WL("a value is "+a);

}

}

class program

{

void main()

{

Myclass.display();

console.ReadLine();

}

O/p

Instance method is Calling

a value is : 10

Note:

→ With the above example we can say that while accessing instance members of the same class within the static method we have required object.

## Advantage of static method :-

→ We can save the memory. because for static members we don't require to create an object.

Q) Why main method is static method?

→ To run console application CLR has to invoke main method

→ If main method is instance method every time object has to create it is wastage of memory and main method is an entry point for program execution which is not using any instance variables due to that reason main is declared as static method.

## Accessing Instance members :-

### 1. Accessing instance members within the class

```
class MyClass
{
    int a = 10;
    internal void Print()
    {
        Console.WriteLine(a);
    }
    internal void Show()
    {
        Print();
    }
}
```

### 2. Accessing instance members from outside the class

```
class MyClass
{
    int a = 10;
    internal void Print()
    {
        Console.WriteLine(a);
    }
}
```

```
class class2
{
    internal void show()
    {
        Myclass obj = new Myclass();
        obj.print();
    }
}
```

### ② Accessing static Members:

#### 1. Accessing static members within the class

```
Class Myclass
{
    static int a=10;
    static internal void print()
    {
        Console.WriteLine(a);
    }
    static internal void show()
    {
        print();
    }
}
```

#### 2. Accessing static members from outside the class

```
class Myclass
{
    static int a=10;
    static internal void print()
    {
        Console.WriteLine(a);
    }
}
```

```

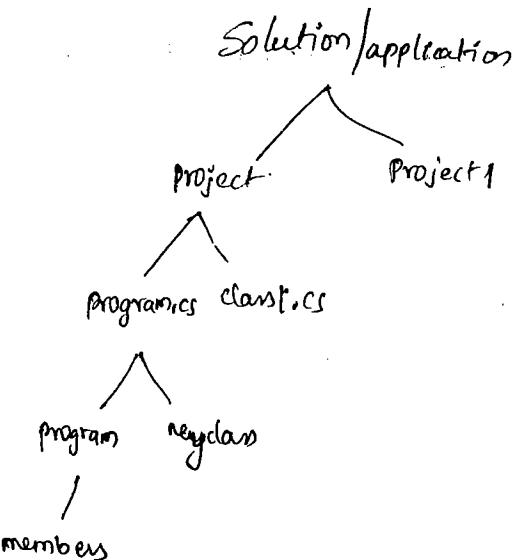
class class2
{
    static internal void Show()
    {
        Myclass.Print();
    }
}

```

## Access Modifiers (or) Access Specifiers

- Access Modifiers (or) Access Specifiers are keywords which are specifying the accessibility (or) access level of class (or) class members.
- While defining a class (or) while declaring a class member we can define the accessibility with the help of access modifiers.
- In C# .Net we have 5 access modifiers ; they are
  1. Public
  2. Private
  3. Protected
  4. Internal
  5. Protected Internal

~~1. Public~~



- Net solution/application is a collection of projects
- Project is a collection of class files
- class file is collection of classes
- class is a collection of members.

### 1. Public

- If we declare a class (or) class members as public which can be accessed by all the classes of current project & all the projects of application.
- To access a public member there is no restriction within the application
- Ex of access modifier public.

namespace Publicmodifier

class class1

{

Public int a=10;

Public void method1()

{

Console.WriteLine("method1 value is :" + a);

}

y

class class2

{

internal

Public void method2()

{ class1 obj = new class1();

Console.WriteLine("method2 value is :" + obj.a);

}

y

class Program

{

```

void Main()
{
    class1 objc1 = new class1();
    Console.WriteLine("method1 value");
    main method value is :" + objc1.a);
    Obj1.method1();
    class2 objc2 = new class2();
    Obj2.method2();
    Console.ReadLine();
}

```

Output

```

main method value is :10
method1 value is :10
method2 value is :10

```

### do Private

- If we declare class (or) class member access modifier as private , it which can be accessed only "within" that class.
- Example for private access modifier

```

namespace privateEx
{
    class class1
    {
        private int a=10;
        private public void method1()
        {
            Console.WriteLine("method1. Value is :" + a);
        }
    }
    class Program
    {
    }
}

```

```

Void Main()
{
    Class obj = new Class1();
    obj.method1();
    obj.method2();
    Console.ReadLine();
}
}

```

### 3. Protected :-

- If we declare class or class member access modifier as protected which can be accessed by current class members as well as derived class members.
- Protected is depending on inheritance concept.
- Example for protected.

Namespace protectedEx

```

{
    Class Class1
    {
        Protected int a=10;
        Public void Method1()
        {
            Console.WriteLine("Method1 value is :" + a);
        }
    }

    Class Class2 : Class1
    {
        Public void Method2()
        {
            Console.WriteLine("Method2 value is :" + a);
        }
    }
}

```

class program

{

void main()

{

class1 obj1 = new class1();

obj1.method1();

class2 obj2 = new class2();

obj2.method2();

obj2.method1();

Console.WriteLine("Main");

Console.ReadLine();

}

#### 4. Internal :-

→ If we declare class or class member access modifier as internal which can be accessed by all the classes of current project.

→ Ex for internal

namespace InternalEx

{

class class1

{

internal int a = 10;

public void method1()

{ Console.WriteLine("Method1 value is :" + a); }

}

}

class class2

{

public void method2()

{ class1 obj = new class1();

Console.WriteLine("Method2 value is :" + obj.a);

}

}

class Program

{

```
    void Main()
    {
        class1 obj2 = new class1();
        Console.WriteLine("Main method value is :" + obj2.a);
        class1 obj2 = Method1();
        class2 Obj3 = new class2();
        Obj3.Method2();
        Console.ReadLine();
    }
}
```

Difference b/w public and internal access modifiers:-

- Public members can be accessed by all the projects of the application.
- Internal members can be accessed only with in that project.

Default access modifiers in c#.Net:-

- Default access modifier of a class is internal.
- Default access modifier of a method is private.
- Default access modifier of a variable is private.
- Default access modifier of a constructor will be private.
- Finally we can say default access modifier of a class member will be private.

Why static constructor cannot contain access modifier?

- We don't require to access static constructor from outside the class.

## Default values :-

1. Numerical datatype default value is Zero.
2. Default value of string is null.
3. Default value of char is null.
4. Default value of floating datatype is zero.
5. Default value of bool is false.
6. Default value of Object is null.

## Passing Parameter mechanism :-

→ Passing a value to a function is called as passing parameter mechanism.

what is a parameter :-

→ A variable which is declared within method signature is called as parameter.

Why Parameter :-

→ Whenever a method is required some value for its functionality from outside we will go for parameters.

## Types of parameters :-

→ Two types

1. Formal parameter
2. Actual Parameter

### 1. Formal Parameter

Called function parameter is formal parameter.

### 2. Actual Parameter:

Calling function parameter is actual parameter.

→ For Example

```
class MyClass
{
    internal void add(int a, int b)
    {
        int c = a + b;
        Console.WriteLine("Result is: " + c);
    }
}

class Program
{
    void Main()
    {
        MyClass obj = new MyClass();
        int x = 10;
        int y = 20;
        obj.add(x, y);
    }
}
```

Annotations:

- called function
- ↑ formal parameters
- ↑ actual parameters
- calling function

16

Note: Argument & Parameter both are same.

→ C# .Net will support three types of passing Parameter mechanism.

1. Call by value  
(a) Pass by value.
2. Call by reference (b) Pass by reference
3. Call by out (c) Pass by out.

## 1. call by value :-

- In call by value when formal parameters are modified the modifications will not be reflected back to actual parameters.
- In call by value while passing the parameters we will pass the actual values.
- Example for call by value.

namespace callbyValueEx

{

class MyClass

{

    internal void PayFee(int a) //formal Parameter

{

    a = a + 1500;

    Console.WriteLine("a value is :" + a);

}

class Program

{

    Void Main()

{

        MyClass obj = new MyClass();

        int x = 1000;

        obj.PayFee(x);

        Console.WriteLine("x value is :" + x);

        Console.ReadLine();

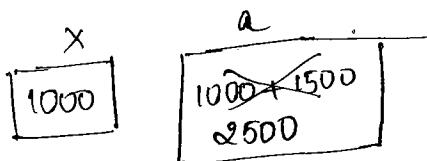
}

Q1P

a value is : 2500

x value is : 1000

-



when can I go for call by value?

→ whenever we want to pass some value to the function and that called function modification should not reflect back to calling function then we will go for call by value. Means we call particular parameter is called call by value.

2. Call by reference :

- In call by reference when formal parameters are modified those modifications will be reflecting back to actual parameters.
- In call by reference while passing the parameters we will pass the address of the value.
- Example for call by reference.

Note: Call by value parameter and call by reference parameter should be initialize with some value before passing otherwise compiler will generate an error. Because which are local variables.

namespace call by reference Ex

{

class MyClass

{

internal void Add (ref int a)

{

a = a + 1500;

Console.WriteLine("a value is :" + a);

}

y

class Program

{

void Main()

{

MyClass obj = new MyClass();

obj.Add

int x = 1000;

obj.Add (ref x);

Console.WriteLine ("x value is :" + x);

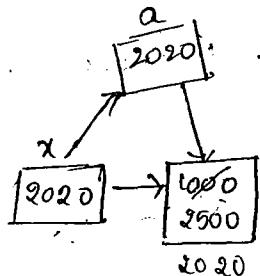
Console.ReadLine();

}

O/P

a value is : 2500

x value is : 25000



Q) When we will go for call by reference?

Ans whenever we want to pass some value to the function and if we want to get back the modified value then we will pass it as a ~~as~~ call by reference.

3. Call by out :-

- Call by out is same as call by reference, <sup>i.e.</sup> in call by out also when formal parameter is modified those modifications will be reflected back to actual parameter.
- While passing the parameter & while catching the parameter we should use out keyword.
- In call by out also we will pass th. with the help of Parameter we will pass the address.
- Out parameter is not required to initialize if we initialize also CLR will ignore that value.

Q) When we will go for call by out?

Ans whenever we dont want to pass some value to function, but we want to get back the modified value we will go for call by out.

→ Example for call by out.

namespace callbyoutex  
{

class Myclass

{

internal void BuFee(out int totFee)

{

```
int admFee = 1500;  
int semFee = 1200;  
totFee = admFee + semFee;  
Console.WriteLine("Total fee is :" + totFee);
```

{  
}}  
class program.

{

void Main()

{

MyClass obj = new MyClass();

int x;

obj.PayFee(out x);

Console.WriteLine("x value is :" + x);

Console.ReadLine();

}  
  
O/p

Total fee is : 2700

x value is : 2700

→ Comparison b/w call by value, call by reference, call by out

### call by value

1. formal parameter  
modification will not be  
reflected to actual  
parameters

### call by reference

1. will be reflected

### call by out

1. will be reflected

2. Passing the value

2. Passing address

2. Passing address

3. No keyword

3. ref Keyword

3. Out Keyword

4. Should be initialized

4. ref parameter  
should be initialized

4. out parameter is  
not required to  
initialize.

Note : → Default passing parameter in c# is call by value.

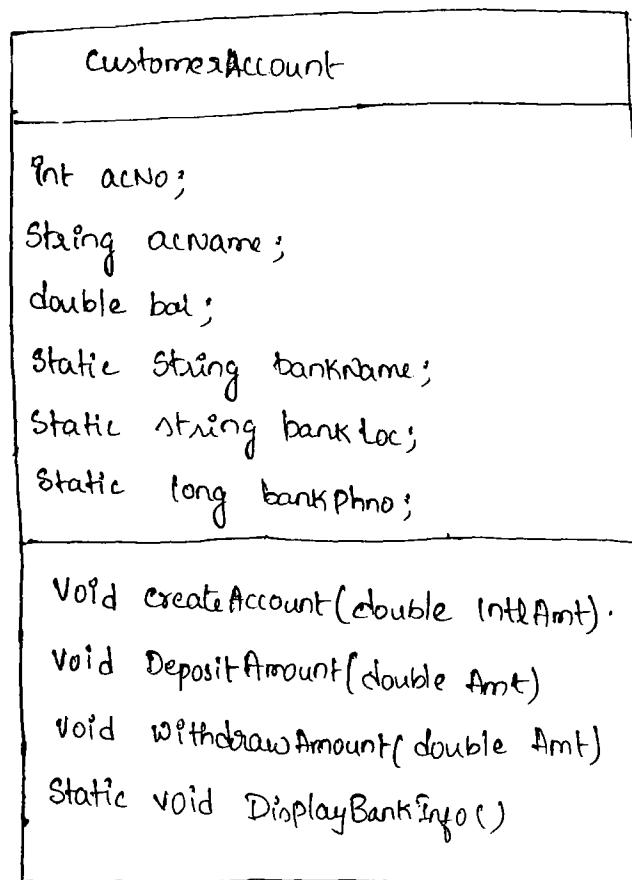
→ VB.NET will support only call by value & call by reference  
and will not support call by out.

## Oops Case Study :-

### Requirement :-

we have a bank for that bank we have require following functionalities

1. Creating account.
2. Depositing amount.
3. Withdraw amount.
4. Displaying the bank information



Program

namespace customer & BankEx

{

class customer

{

int actNo;

String actName;

double bal;

static String bankName;

static String bankLoc;

static long bankPhno;

internal static customer()

{

bankName = "ICICI";

bankLoc = "Mumbai";

bankPhno = 0402361452;

}

internal Customer(int actNo, String actName)

{

this.actNo = actNo;

this.actName = actName;

}

internal void createAcct(double InitAmnt)

{

bal = bal + InitAmnt;

Console.WriteLine("Your acct is created successfully");

Console.WriteLine("Account no is :" + actNo);

Console.WriteLine("Account name is :" + actName);

Console.WriteLine("Account bal is :" + bal);

}

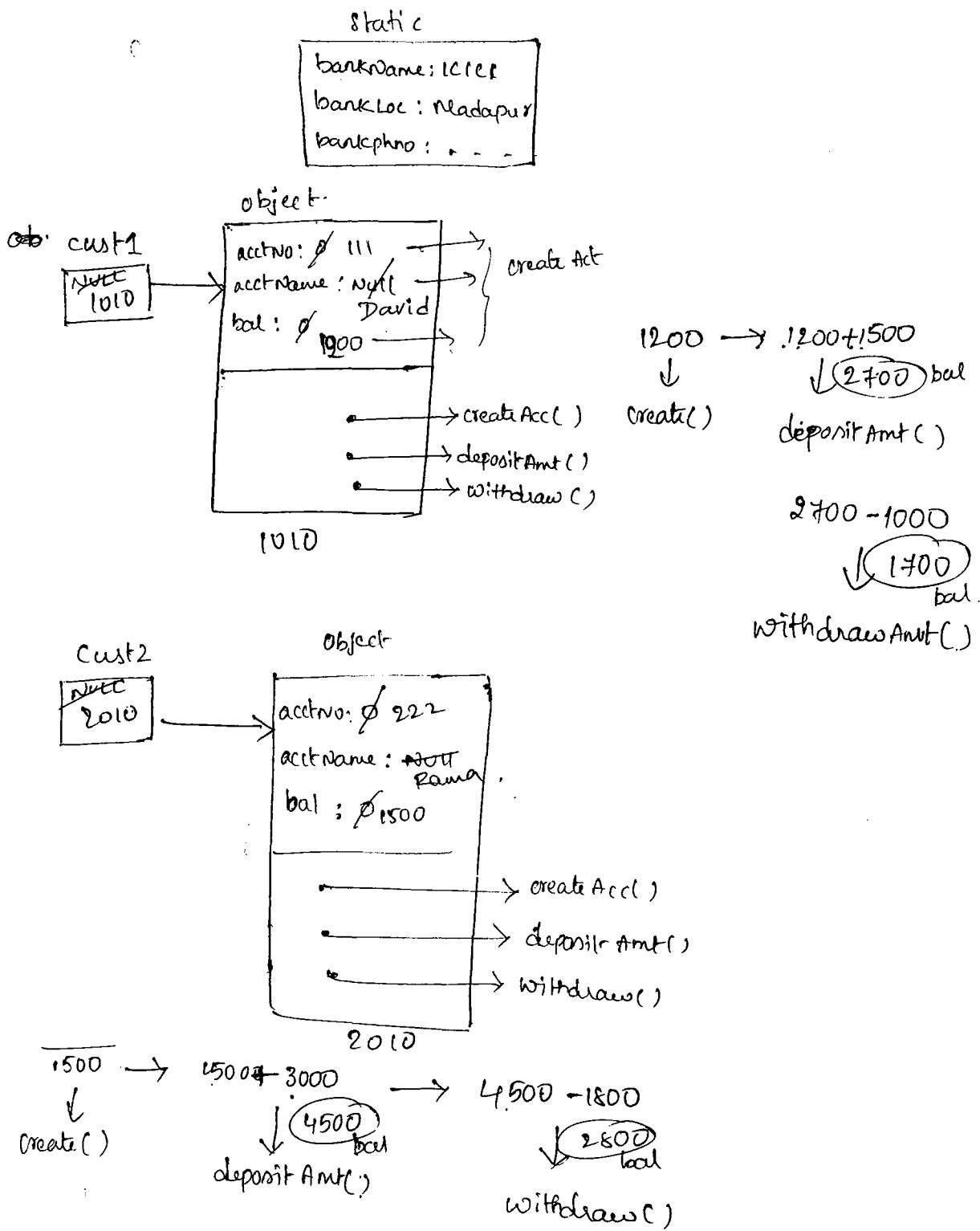
```
else
{
    Console.WL("Account is not created");
}

internal void depositAmt(double Amt)
{
    bal = Amt + bal;
    Console.WL("Your total balance is :" + bal);
}

internal void withdrawAmt(double Amt)
{
    if(bal >= Amt)
    {
        Console.WL("Your withdrawn amount is :" + Amt);
    }
    else
    {
        Console.WL("Insufficient funds");
    }
}

static void DisplayBankInfo()
{
    Console.WL("Bank Name is :" + bankName);
    Console.WL("Bank Location is :" + bankLoc);
    Console.WL("Bank Ph No is :" + bankPhno);
}

class Program
{
    void Main()
    {
        Customer obj1 = new Customer(111, "Rama");
        obj1.CreateAcct(1000);
        obj1.depositAmt(1000);
        obj1.withdrawAmt(500);
        Customer.DisplayBankInfo();
        Console.ReadLine();
    }
}
```



Q. We have a requirement for a company. In that company we have to perform following operations.

1. Employee joining and display employee details
2. Increment the employee salary.
3. Displaying the Employee Company info like Company name, Company location, Company phno.

```
employee
int eno;
string ename;
double esal;
double hikkesal;
static companyName;
static comploc;
static compno;

void empinfo(int eno, string ename, double esal)
void emphike(int
static compinfo(string empname, string emploc, long CompNO)
```

name.space employee.infoEx

{

class employee

{

int eno;

String ename;

double esal;

Static String cmpName;

Static String cmpLoc;

Static long cmpNo;

Static employee()

{

CmpName = "Microsoft";

CmpLoc = "Hitechcity";

CmpNo = 123456;

}

Internal employee(int eno, String ename, double esal)

{

this.eno = eno;

this.ename = ename;

this.esal = esal;

Console.WriteLine("Employee No is;" + eno);

Console.WriteLine("Employee Name is;" + ename);

Console.WriteLine("Employee Salary is;" + esal);

}

```
internal void hike()
{
    float i = 0.0f;
    double tot = esal + (esal * i);
    Console.WriteLine("Employee incremented salary is :" + tot);
}

static void compInfo()
{
    Console.WriteLine("Employee Company Name is :" + cmpName);
    Console.WriteLine("Company Location is :" + cmpLoc);
    Console.WriteLine("Company Phno is :" + cmpPho);
}
```

class program

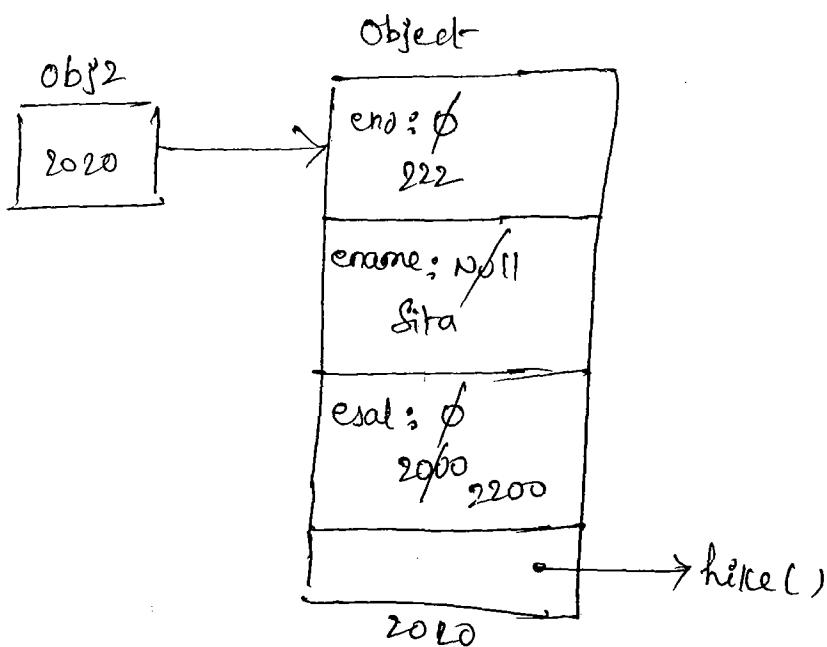
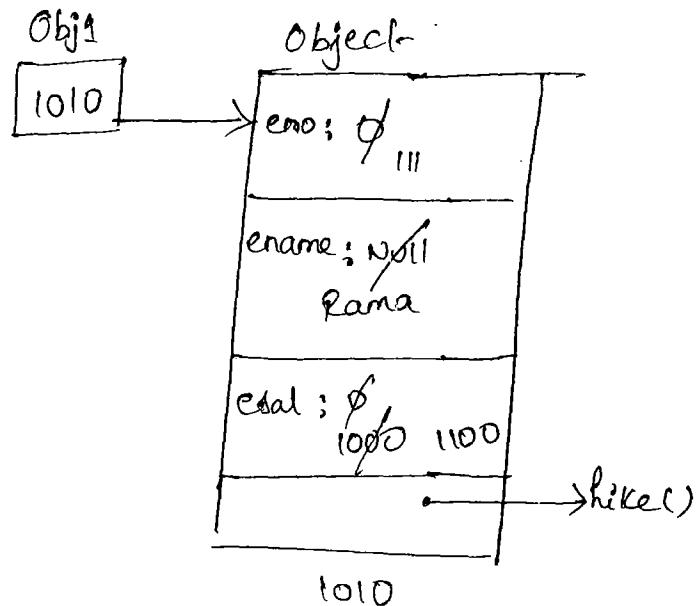
```
void Main()
{
    Employee obj1 = new Employee(111, "Rama", 1000);
    obj1.hike();
    obj1.compInfo();
    Employee.CompInfo();

    Employee obj2 = new Employee(222, "Sita", 2000);
    obj2.hike();
    Employee.CompInfo();

    Console.WriteLine(ReadLine());
}
```

Company Name : MicroSoft  
 Company Loc : Hitech City  
 Company No : 12345

Static Variables



## Constant :-

- To declare a constant we have to use const keyword.
- By default constant is static
- Constant should be initialized at the time of declaration that is in design time. and the value will be never changed.

(Q) When we will go for constant?

Ans

Example for constant:

```
namespace Company
{
    class Employee
    {
        internal const int MinWorkHrs = 9;
        internal static void display()
        {
            Console.WriteLine("Minimum working hours is :" + MinWorkHrs);
        }
    }
    class Program
    {
        void Main()
        {
            Employee.Display();
            Console.ReadLine();
        }
    }
}
```

- In the above example we have declared MinWichrs as constant, but, there is a chance of changing this value for this requirement we cannot declare as constant.
- Finally we can say in business applications we will declare constants very rarely.
- Especially in scientific applications, to represent universal fixed values like below we will declare constants.

Ex:- const double Pi = 3.14;  
 const long Speedoflight = 300000; // km per sec

- Example for predefined constants are

- a. MinValue
- b. Max Value

- Differences between constant and static variable.

<u>Constant</u>	<u>Static Variable</u>
1. For using Constant we use const keyword.	1. For using static variable we use static keyword.
2. Const can be used, when a field value is not to be changed forever.	2. Static <del>can</del> value can <del>be</del> also be fixed value, but that value can be changed by static method.
3. Const can be used mostly, to represent universal fixed values.	

3. Constant should be initialized 4. Static can be initialized while declaration.

using constructor (C) during declaration

5. By default it is static

6. Static keyword should be declared explicitly.

7. Whenever we want to have same value to all the objects but not required to change forever.

8. whenever we want to represent common value for all the objects but required to change in future.

9. Declared during design time.

10. Declared during run time.

### readonly :-

- To declare readonly we have to use readonly keyword
- By default readonly is a instance member.
- readonly can be initialized at the tym of declaration (design tym) as well as readonly can be initialized in runtime but only within the constructor but not within the method.
- readonly value cannot be changed.

When we will go for readonly?

Ans Whenever we want to have a field for multiple objects with different values but values should not be changed. then we can go for readonly field.

→ Example for readonly.

namespace readonlyEx

{

class employee

{

readonly int eno;

readonly string ename;

double esal;

internal employee( int eno, string ename, double esal )

{

this.eno = eno;

this.ename = ename;

this.esal = esal;

}

internal void hike()

{

double incsal = esal \* 10/100;

Console.WriteLine("Your increment salary is :" + incsal);

esal = esal + incsal;

Console.WriteLine("Updated salary is :" + esal);

}

}

Class Program

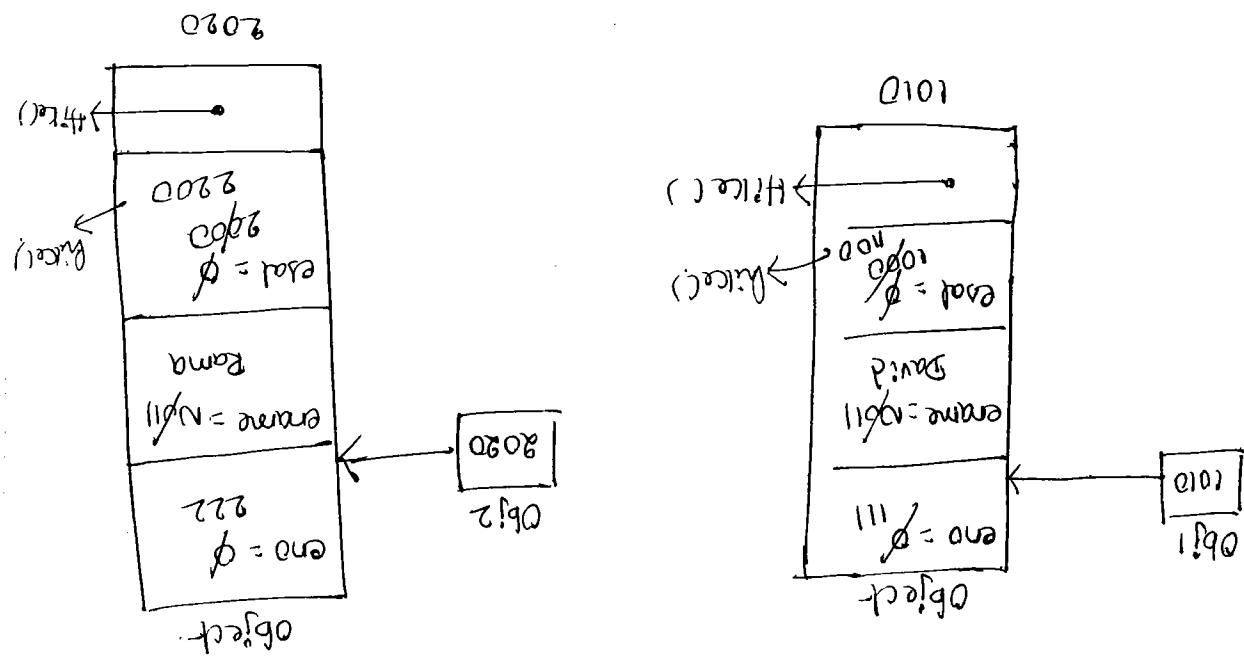
{

void Main()

{

employee obj1 = new employee( 111, "John", 1000 );

obj1.hike();



updated salary is 2000

your increment salary is : 200

updated salary is : 1100

your increment salary is : 100

OP

5

Console.ReadLine();

obj2.write();

Employee obj2 = new Employee(222, "Pama", 2000);

## readonly Vs instance variable

### readonly

1. readonly keyword.
2. Initialization can be done during declaration and also in runtime but only in constructor.
3. Value cannot be changed.
4. when?

Whenever we require a field for all the objects with different values but, value doesn't required to change then we will go for readonly.

Eg:: Empno, empname, SId, Sname...

### readonly

1. readonly Keyword.
2. By default instance.
3. Readonly can be initialized at the time of declaration as well as runtime i.e., in constructor.
4. Readonly value ~~can~~ will be differ from one object to another object.
5. when?

### instance variable

1. No keyword.
2. Initialization can be done during declaration and <sup>can</sup> also in runtime i.e., in constructor (d) in method also.
3. value can be changed.
4. when?

Whenever we require a field for all the objects with different values but, value is required to change in future then we will go for instance variable.

Eg:: esal, sloc, sm1, sm2 - - -

### Constant

1. const Keyword.
2. By default static
3. Constant can be initialized <sup>only</sup> at the time of declaration i.e., during design time. Cannot be initialized in runtime.
4. Constant value will be same for all the objects.
5. when?

## Differences b/w Instance variable & static variable.

### Instance Variable

1. No keyword.
2. Initialization is done in instance constructor.
3. Memory is allocating at the time of object is creating.
4. Value will be different from one object to another object.
5. memory will be deallocated when the object is destroyed.
6. when?

### Static Variable

1. Static keyword.
2. for initialization is done in static constructor.
3. memory is allocating when the class is loading.
4. Value is same for all the objects.
5. memory will be deallocated when the class is unloading.
6. when?

### Instance Method

1. No keyword
2. To access instance variables we go for instance method.
3. Purpose of instance method is to change the instance variable values.
4. instance method we have to invoke by using object.
5. when?

### Static Method

1. static keyword.
2. To access static variables
3. Purpose of static method is to change the static variable values.
4. static method we have to invoke with the help of class name.
5. when?

## Instance Constructor

## Static Constructor

- 1. No keyword.
- 2. It can contain access modifiers.
- 3. It can contain parameters.
- 4. Within instance constructor we can initialize instance and also static variables.
- 5. When?
- 6. Instance constructor will invoke when the object is creating.
- 7. Within a single class we can have multiple instance constructors.
- 1. Static keyword.
- 2. It cannot contain access modifiers.
- 3. It cannot contain parameters.
- 4. In static constructor we can initialize only static variables.
- 5. When?
- 6. Static constructor will invoke when the class is loading.
- 7. Within a single class we can have only one static constructor.

## Constructor

## Method

- 1. Constructor name same as class name.
- 2. It will contain initialization logic.
- 3. No return types.
- 4. Constructor will invoke automatically if instance at the time of creating according to the object and if static at the time of loading the class.
- 5. When?
- 1. Method name will be user defined.
- 2. It will contain modification logic.
- 3. Method can contain return types.
- 4. Method can be invoked by programmer according to his requirement.
- 5. When?

## Properties :-

- Property is a member of a class which we will use
- to assign the value to <sup>a variable,</sup> as well as which we will use to retrieve the value from variable.
- Property name and Variable name should be same.
- But Variable name should start with small letter and property name should start with capital letter.
- For Ex

eno → variable name

Eno → property name

## Syntax to define a property:

<access modifier> <return type> <property name>

{

get → retrieving the value

{

Set → assigning <sup>given</sup> value

{

}

y

→ Properties are 3 types.

### 1. Readonly Property

It will contain only get access

## 2. Write only:

→ Contains only Set access.

## 3. Read and write properties:

→ Contain get access and set access.

→ Again properties are 2 types.

1. Instance property

2. Static Property

### 1. Instance property:-

→ While defining a property if we don't use static keyword is called as instance property.

→ The purpose of instance property is to initialize and to retrieve values to instance variables.

### 2. Static Property:-

→ While defining a property if we have used static keyword is called as static property.

→ Purpose of static property is to initialize as well as to retrieve values from static variables.

→ Example for static and instance property.

Namespace property ex.

{

class Employee

{

int eno;

String ename;

```
int age;  
static string compName;  
internal int End
```

```
{  
    get  
    {  
        return end;  
    }  
}
```

```
set  
{  
    end = value;  
}
```

```
}  
internal string Ename
```

```
{  
    get  
    {  
        return ename;  
    }  
}
```

```
set  
{  
    ename = value;  
}
```

```
}  
internal int Age
```

```
{  
    get  
    {  
        return age;  
    }  
}
```

```
set  
{  
    while (value < 1 || value > 150)  
    {  
        Console.WriteLine ("Please enter valid age.");  
        value = int.Parse (Console.ReadLine());  
    }  
    age = value;  
}
```

```
internal static string CompanyName
```

```
{
```

```
    get  
    {
```

```
        return CompanyName;
```

```
}
```

```
    set
```

```
{
```

```
    CompanyName = value;
```

```
}
```

```
}
```

```
}
```

```
class Program
```

```
{
```

```
    void Main()
```

```
{
```

```
    Employee emp1 = new Employee();
```

```
    Console.WriteLine("Enter ur EmpNo:");
```

```
    emp1.Age =
```

```
    emp1.EmpNo = int.Parse(Console.ReadLine());
```

```
    Console.WriteLine("EmpNo is :" + emp1.EmpNo);
```

```
    Console.WriteLine("Enter ur EmpName:");
```

```
    emp1.Ename = Console.ReadLine();
```

```
    Console.WriteLine("EmpName is :" + emp1.Ename);
```

```
    Console.WriteLine("Enter your age:");
```

```
    emp1.Age = int.Parse(Console.ReadLine());
```

```
    Console.WriteLine("Emp age is :" + emp1.Age);
```

```
    Console.WriteLine("Enter your Company Name :");
```

```
Employee.ComName = Console.ReadLine();
Console.WriteLine("Company name is :" + Employee.ComName);
Console.ReadLine();
```

{

}

}

### Purpose of Property :-

- Whenever we want to assign the value to a class level variable or whenever we want to retrieve the value from class level variable from outside the class we can go for property.
- As well as whenever we want to perform validations while assigning the value to class level variable from outside the class we can use property.

→ Console prgm within the Notepad.

Step 1: Open note pad.

Step 2: Write below code within notepad.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        for(int i=0; i<=10; i++)
        {
            Console.WriteLine("Welcome to Satya");
        }
        Console.ReadLine();
    }
}
```

Step 3: Save Program file with .cs.

Step 4:

Start → Program → Visual Studio 2010 → Visual Studio 2010 → Visual Studio command prompt.

→ change to d drive (d:) ↴

D:\>csc Program.cs ↴

With this process compilation is completed it will generate Program.exe file

Step 5: Executing the Program

D:\> Program.exe ↴

Welcome to Satya

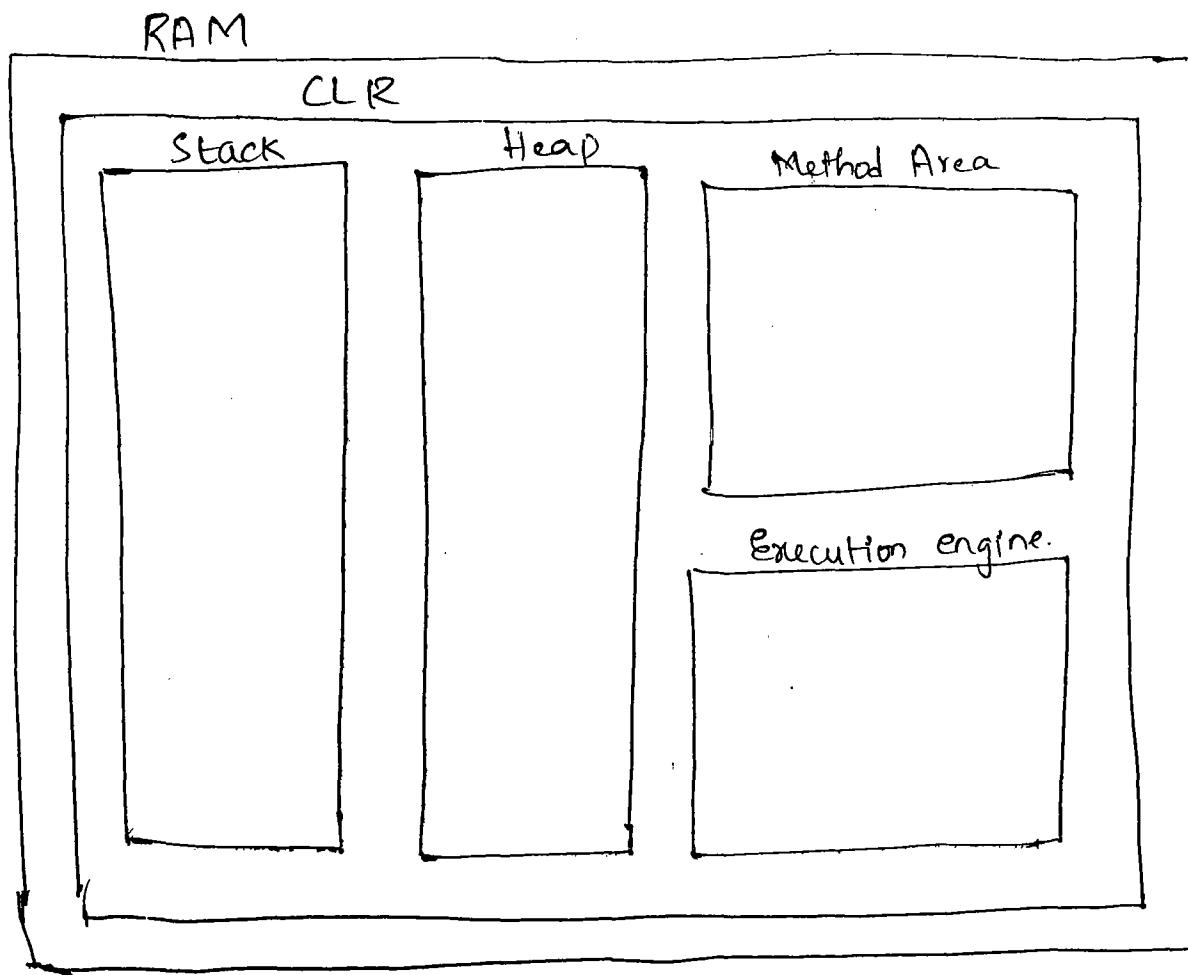
Welcome to Satya.

HW.

Implement all above programs using Notepad.

## CLR Inside (or) CLR Architecture (or) Memory allocation in CLR :-

- When we run the .Net application, .Net execution engine called CLR will be loading in to RAM memory.
- When we run Java application, Java execution engine JVM will load into RAM memory.
- CLR will maintain various blocks for memory allocation or memory management while executing the application.
- Among the various blocks of the CLR, 4 are important:
  1. Stack
  2. Heap
  3. Method Area
  4. Execution Engine.like below.



### Stack :

→ Stack CLR will use to store local variables and reference variables.

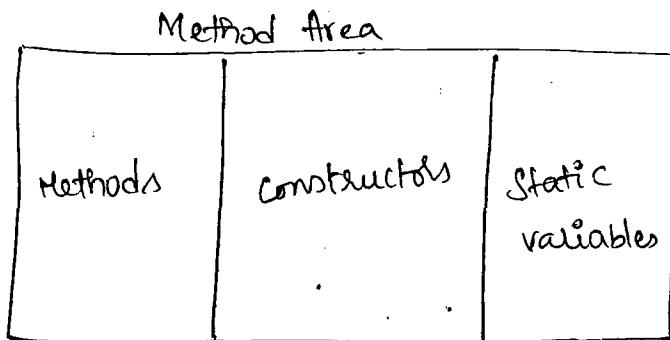
### Heap:

→ CLR will use heap to store objects.

→ Object will contain instance variables & reference of instance methods.

### Method Area :

→ CLR will use method area to store maintain static variables and methods and constructors like below.



### Execution Engine:

→ CLR will use execution engine to maintain the block which is executing currently , it can be a method (or) it can be a constructor.

○ → Example to understand Execution engine.

```
class MyClass
{
    static Method1()
    {
    }

    static Method2()
    {
    }

}

class Program
{
    void Main()
    {
        MyClass.Method1();
        MyClass.Method2();
    }
}
```

→ When we run the above program, first CLR required

Program class, due to that reason CLR will load program class into method area means, main method will place into method area.

→ Now CLR has to start execution with main method, due to that reason main method will be moving from method area to execution engine.

→ Now CLR will has to execute main method first statement that is MyClass.Method1. Now it requires MyClass, due to

that means CLR will load Myclass methods i.e., Method1 and Method2 into Method area.

→ Now CLR will execute Method1 for this it will move method1 from method area to execution engine.

→ Now CLR will execute Method1 which is in execution engine

→ Once method1 execution is completed CLR will remove method1 block from execution engine. and now CLR will execute Method2 like below.

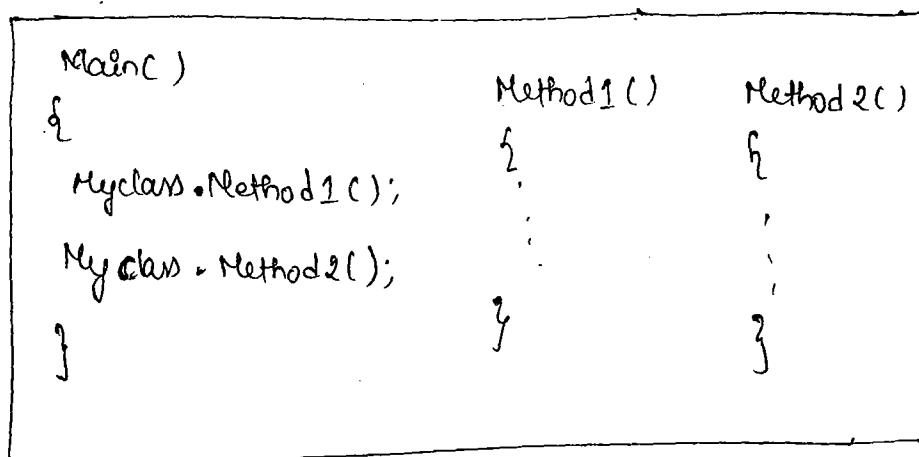
Myclass.Method2();

→ Due to that reason CLR will move the Method2 ~~block~~ from method area to execution engine and so on.

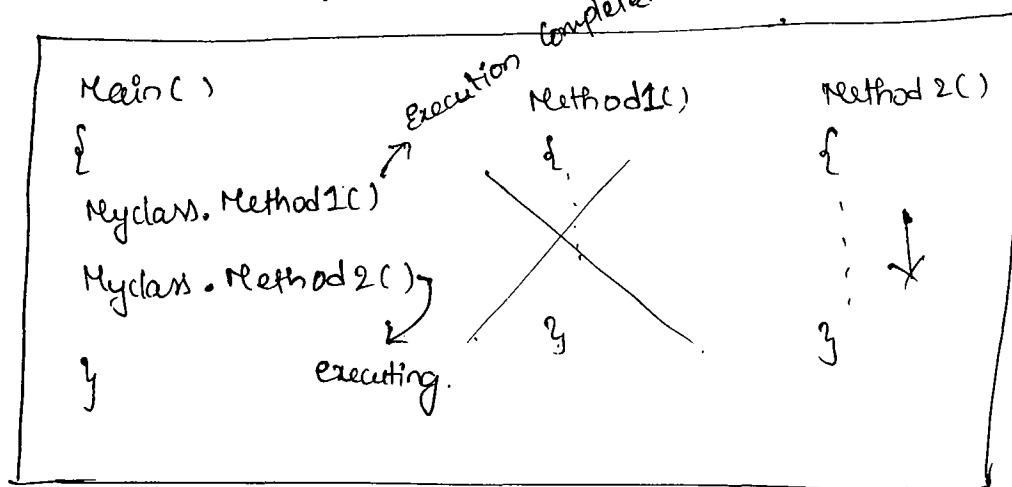
→ Once main method execution is completed main block will be removed from the execution engine.

→ Once application execution is completed (2) closed, all the classes will unload nothing but method area will be cleared.

### Method Area



## Execution Engine



→ Example for understanding the memory allocations within CLR.

namespace memoryallocationEx  
{

class Employee  
{

int eno;

String ename;

double esal;

Static String CompName;

Static Employee()

{

CompName = "Microsoft";

}

internal Employee(int eno, String ename, double esal)

{

this.eno = eno;

this.ename = ename;

this.esal = esal;

}

```
internal void Display()
{
    Console.WriteLine("EmpNo is :" + empno);
    Console.WriteLine("Ename is :" + ename);
    Console.WriteLine("EmpSal is :" + esal);
    Console.WriteLine("CompName is :" + CompName);
}

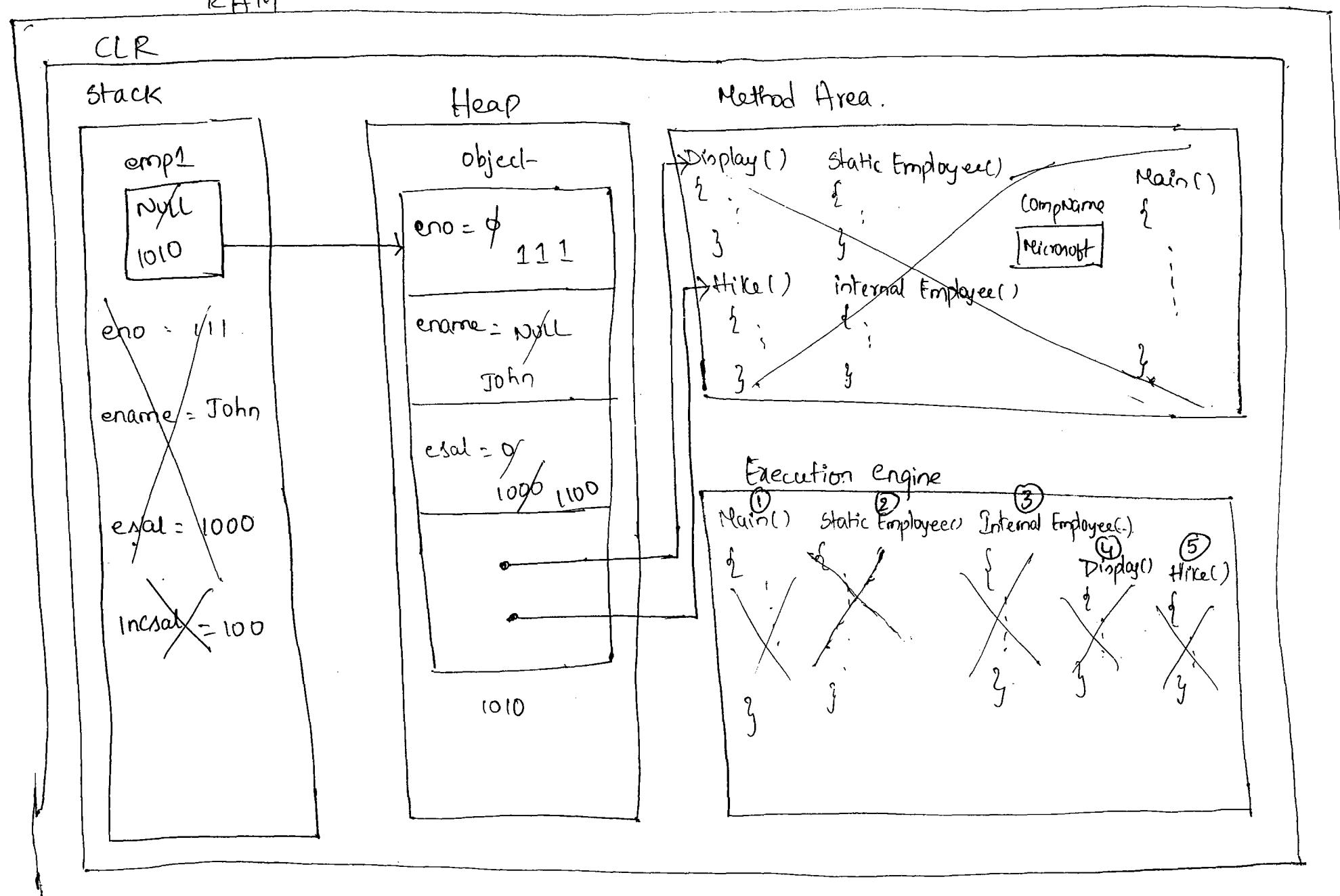
static

internal void Hike()
{
    double incsal = esal * (10 / 100);
    Console.WriteLine("Incremented salary is :" + incsal);
    esal = incsal + esal;
    Console.WriteLine("Updated salary is :" + esal);
}

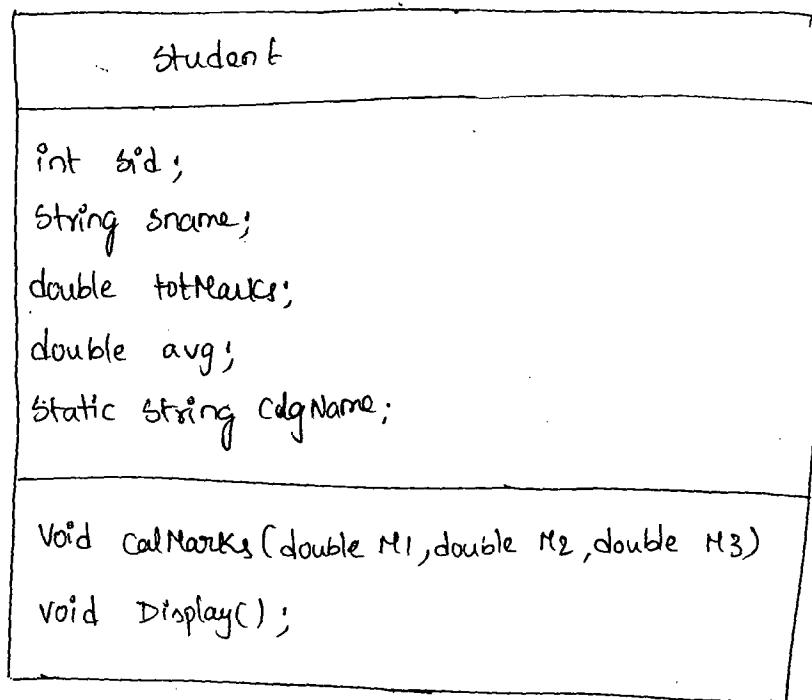
}

class Program
{
    void Main()
    {
        Employee emp1 = new Employee(111, "John", 1000);
        emp1.Display();
        emp1.Hike();
        Console.ReadLine();
    }
}
```

RAM



→ Example 2 for memory allocation in CLR.



name class

namespace memoryallocationEx2

{

Class Student

{

int Sid;

String Sname;

double totMarks;

double avg;

Static String colgName;

Static Student()

{

colgName = " Karunya University";

}

```
internal Student(int sid, string sname)
{
    this.sid = sid;
    this.sname = sname;
}

internal void calMarks(double M1, double M2, double M3)
{
    totMarks = M1 + M2 + M3;
    avg = totMarks / 3;
}

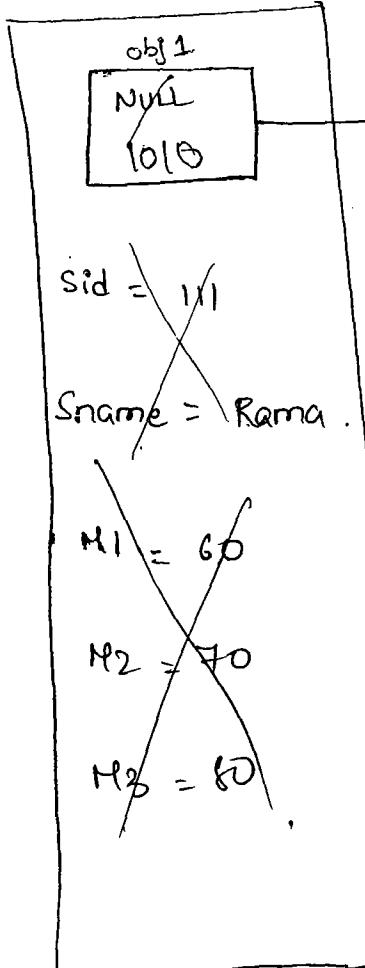
internal void Display()
{
    Console.WriteLine("Student Id is :" + sid);
    Console.WriteLine("Student name is :" + sname);
    Console.WriteLine("Student total Marks is :" + totMarks);
    Console.WriteLine("Student average is :" + avg);
    Console.WriteLine("Student College is :" + clgName);
}
```

```
class Program
{
    void Main()
    {
        Student obj1 = new Student(111, "Rama");
        obj1.calMarks(60, 70, 80);
        obj1.Display();
        Console.ReadLine();
    }
}
```

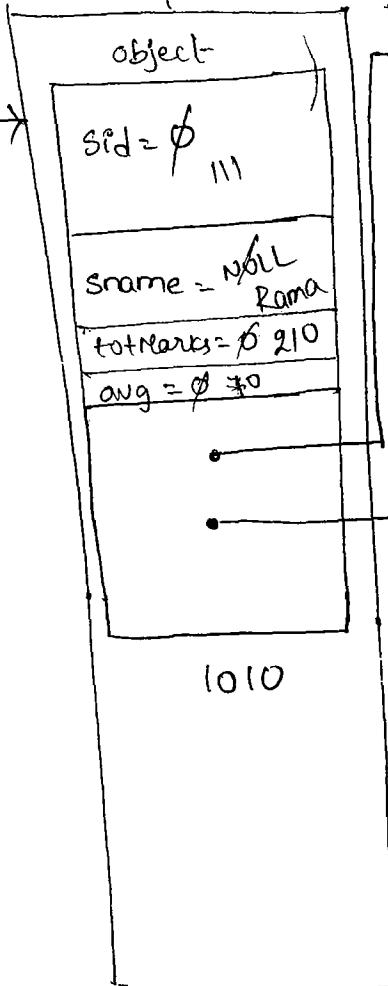
RAM

CLR

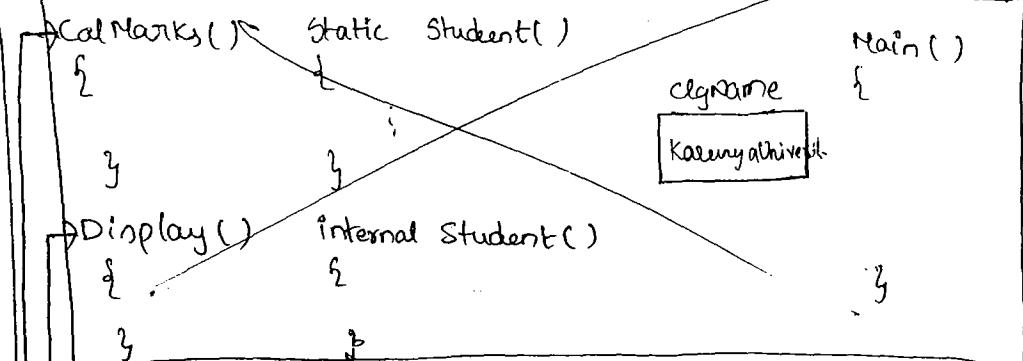
Stack



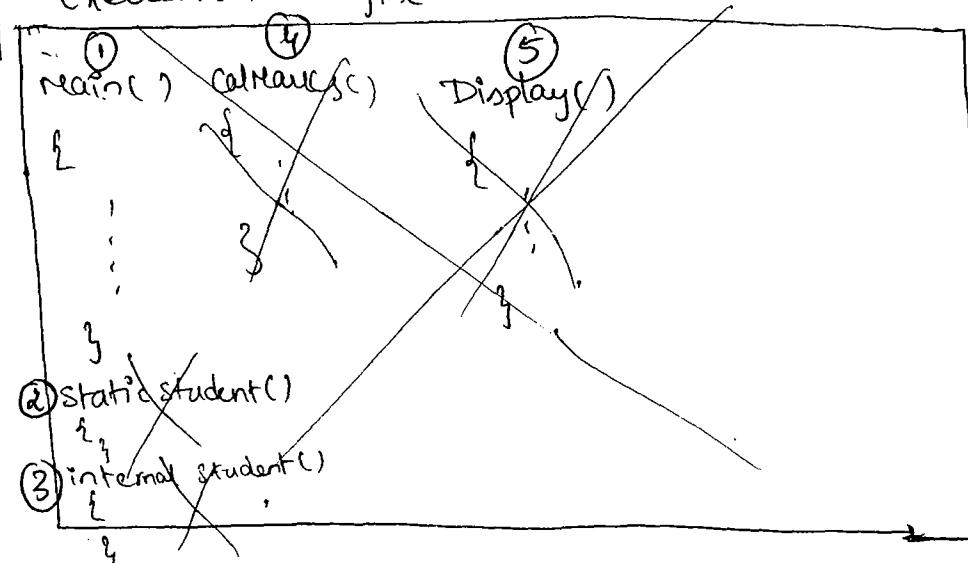
Heap



Method Area



Execution Engine



- Draw the CLR diagram for Bank case study.
- Draw the CLR diagrams for all the above oops examples.

## Oops Principles

### 1. Encapsulation :-

- Wrapping states and behaviors into a single unit is called Encapsulation.
  - Binding variables and methods is called as Encapsulation
- How to encapsulate?

By using class.

### 2. Abstraction :-

- Hiding the unwanted data is called as Abstraction.
- Abstractions are two types.

1. Data Abstraction

2. Method Abstraction.

#### 1. Data abstraction :-

Hiding the unwanted data is nothing but data abstraction

#### 2. Method Abstraction:-

Hiding unwanted method is called as method abstraction.

→ Example data abstraction.

```
namespace dataAbstractionEx  
{  
    class Employee  
    {  
        int empNo;  
        String empName;  
        String designation;  
        double esal;  
  
        internal Employee(int empNo, String empName, String designation, double esal)  
        {  
            this.empNo = empNo;  
            this.empName = empName;  
            this.designation = designation;  
            this.esal = esal;  
        }  
  
        internal void display()  
        {  
            Console.WriteLine("EmpNo is :" + empNo);  
            Console.WriteLine("EmpName is :" + empName);  
            Console.WriteLine("Empdesignation is :" + designation);  
            Console.WriteLine("Empsalary is :" + esal);  
        }  
  
        internal void Hike()  
        {  
            double Incsal = esal * (10 / 100);  
            esal = Incsal + esal;  
            Console.WriteLine("EmpNo : " + empNo + " EmpName : " + empName +  
                " updated salary is :" + esal);  
        }  
    }  
}
```

## Class Program

```
void Main()
{
    Employee emp1 = new Employee( 111, "John", "Software Engineer",
                                  1000);
    emp1.Display();
    emp1.Hike();
    Console.ReadLine();
}
```

Note : In the above program, Display method does not have any data abstraction, but Hike method has data abstraction i.e., designation of employee is not needed in Hike method. So we are hiding designation in Hike(). It is nothing but data abstraction.

### 3. Inheritance :

→ Inheritance is nothing but deriving (or) inheriting the members from one class to another class.

→ A class which is giving the members is called as Super class (or) parent class (or) base class.

→ A class which is receiving the members is called as Sub class (or) derived class (or) child class.

→ Because of inheritance sub class can access Super class members as well as using sub class object we can access Super class members.

→ Because of inheritance Super class cannot access sub class members as well as using super class object we cannot access sub class members

→ Types of inheritance.

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hybrid inheritance
5. Hierarchical inheritance.

1. Single inheritance :-

→ Inheriting from one class to another class is called single inheritance.

→ Syntax.

```
class C1  
{  
}  
  
class C2 : C1  
{  
}  
  
}
```

→ Example for single inheritance.

```
namespace SingleInheritanceEx  
{
```

```
class Branch  
{  
    int branchid;  
    String branchname;  
    String branchloc;
```

```
B internal Branch( int branchid, String branchname, String branchloc)
```

```
C {
```

```
    this.branchid = branchid;
```

```
    this.branchname = branchname;
```

```
    this.branchloc = branchloc;
```

```
D }
```

```
E internal void BranchDisplay()
```

```
F {
```

```
    Console.WL("Branch Id is :" + branchid);
```

```
    Console.WL(" Branch name is :" + branchname);
```

```
    Console.WL(" Branch location is :" + branchloc);
```

```
G }
```

```
H }
```

```
I class Employee : Branch
```

```
J {
```

```
K     int eno;
```

```
L     String ename;
```

```
M     String designation;
```

```
N     internal Employee( int eno, String ename, String designation) : base(
```

```
O     {
```

```
P         (11, "HydBranch", "Aman") .
```

```
Q     this.eno = eno,
```

```
R     this.ename = ename;
```

```
S     this.designation = designation;
```

```
T }
```

```
U     internal void EmployeeDisplay()
```

```
V {
```

```
W     Console.WL("Emp No is :" + eno);
```

```
X     Console.WL("Emp Name is :" + ename);
```

```
Y     Console.WL(" Emp designation is :" + designation);
```

```
Z     base.BranchDisplay();
```

```
a }
```

```
b }
```

```

class Program
{
    Void Main()
    {

```

```

        Employee emp1 = new Employee(111, "Rama", "SoftwareEngg");
        emp1.EmployeeDisplay();

```

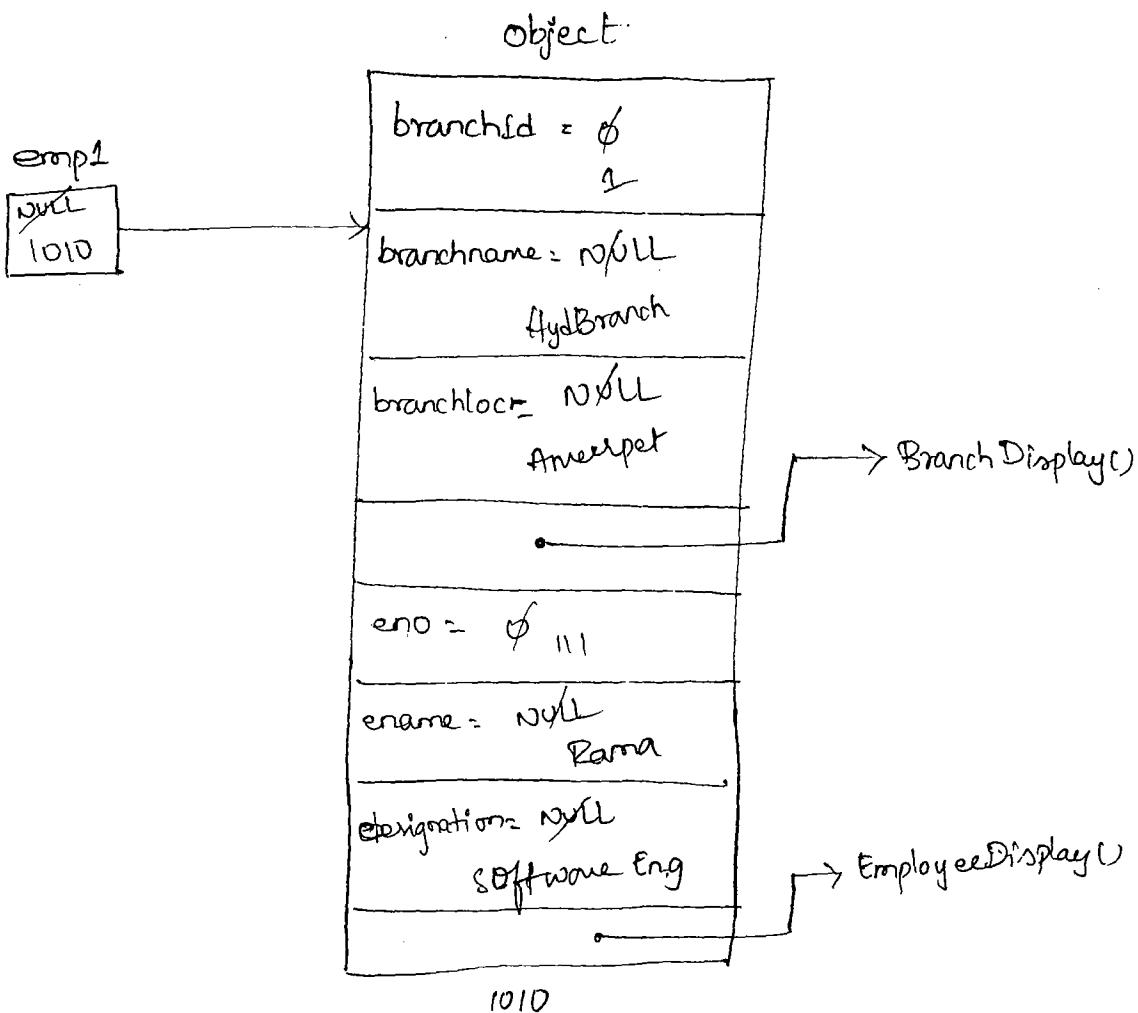
```

        Console.ReadLine();
    }
}
```

```

}

```



## Q. Multi Level Inheritance :-

- Inheriting from one class to another class, from that class to some other class is called as MultiLevel inheritance

→ Syntax :

class C1

{

}

class C2 : C1

{

}

class C3 : C2

{

}

→ Example for multilevel inheritance.

namespace MultiLevelInheritanceEx

{

class HeadOffice

{

String headLoc;

ulong headPhNo;

internal HeadOffice (String headLoc, ulong headPhNo)

{

this. headLoc = headLoc;

this. headPhNo = headPhNo;

}

```
internal void HeadoffDisplay()
{
    Console.WL("HeadOffice location is :" + headloc);
    Console.WL("HeadOffice Phone No is :" + headPhNo);
}

class Branch : HeadOffice
{
    int branchid;
    String branchName;
    String branchloc;

    internal Branch(int branchid, String branchName, String branchloc)
        base("Delhi", 123444)

    {
        this.branchid = branchid;
        this.branchname = branchname;
        this.branchloc = branchloc;
    }

    internal void BranchDisplay()
    {
        Console.WL("Branch id is :" + branchid);
        Console.WL("Branch Name is :" + branchName);
        Console.WL("Branch location is :" + branchloc);
        base.HeadoffDisplay();
    }
}
```

```
class Employee : Branch
{
    int eno;
    string ename;
    string designation;
```

```
internal Employee(int eno, string ename, string designation) : base
    (1, "HydBranch", "Ameerpet");
{
```

```
    this.eno = eno;
    this.ename = ename;
    this.designation = designation;
```

```
}
```

```
internal void EmployeeDisplay()
```

```
{
```

```
    Console.WriteLine("Emp NO is :" + eno);
    Console.WriteLine("Emp Name is :" + ename);
    Console.WriteLine("Emp designation is :" + designation);
```

```
    base.BranchDisplay();
```

```
}
```

```
}
```

```
class Program
```

```
{
```

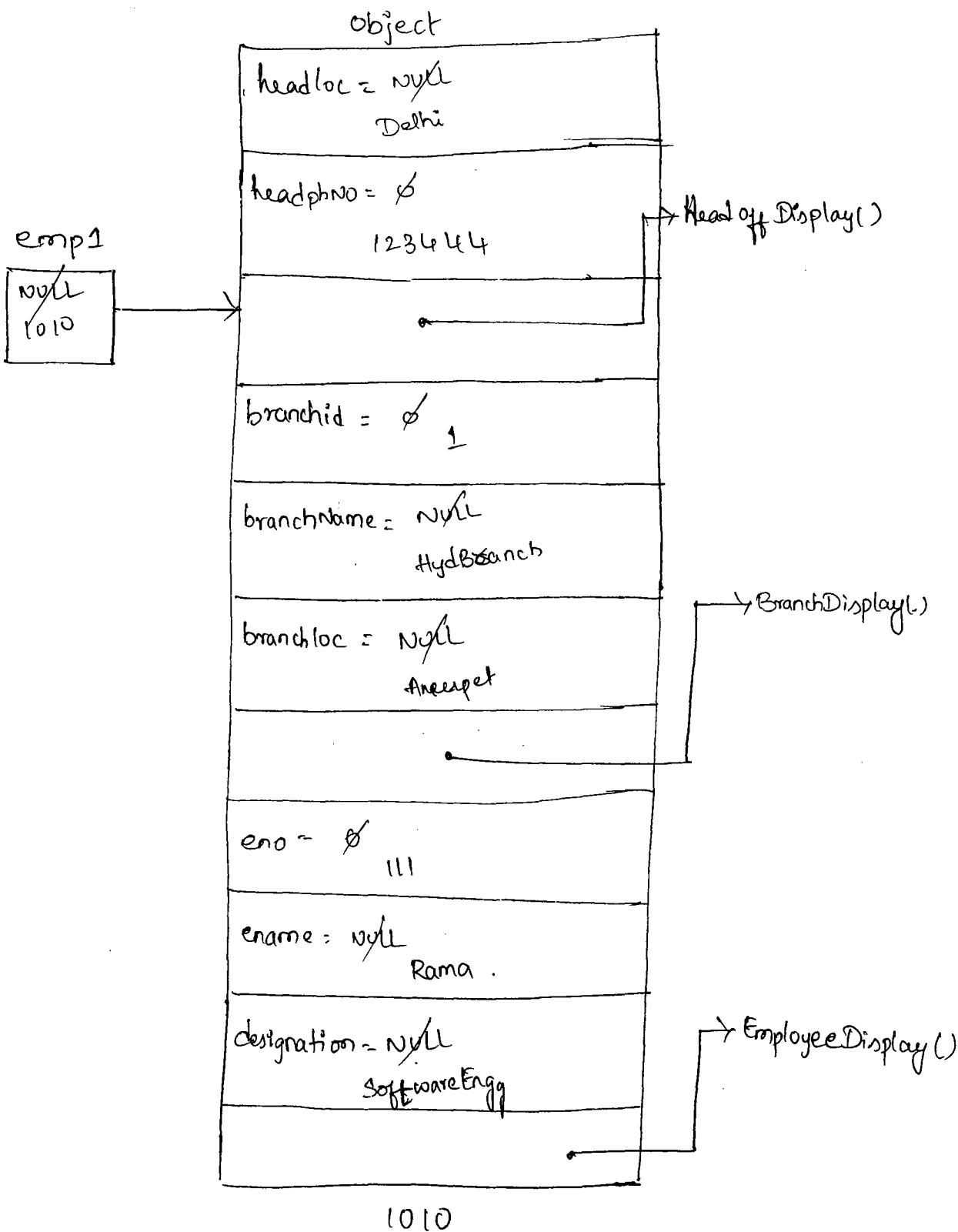
```
    void Main()
```

```
{
```

```
    Employee emp1 = new Employee(111, "Rama", "SoftwareEng");
```

```
    emp1.EmployeeDisplay();
```

```
    Console.ReadLine();
```



### 3. Multiple Inheritance:-

→ Inheriting multiple classes into single class is called as multiple inheritance.

→ Syntax :

```
Class C1
{
}

Class C2
{
}

Class C3 : C1, C2
{
}
```

Q) Can we implement multiple inheritance in C# .Net?

Ans Yes. But not possible by using classes, which is possible with the help of interfaces.

Q) Why multiple interface inheritance is not possible by using C# .Net?

Ans

HW 1) Implement one more real time case study by using all the oops concepts

2) Implement one real time case study by using single inheritance

3) Implement one real time case study by using MultiLevel inheritance

4) Hybrid Inheritance :-

→ A combination of two inheritance is nothing but Hybrid inheritance

5) Hierarchical Inheritance :-

→ Inheriting ~~one~~ single class into multiple classes is called as Hierarchical inheritance.

→ Syntax :

class C1 .

{

  class C2 : C1

{

  class C3 : C1

{

### Example 1 :-

→ Object is Super class for all .Net classes.

### Example 2 :-

→ In ASP.NET Page is the super class for all web page classes like Webform1, Webform2 ---, Webform n.

### Example 3:-

→ In Windows Forms, Form is the super class for all the forms like form1, form2 --- Form n.

→ In all the above 3 examples, Microsoft has implemented Hierarchical inheritance.

### Advantages of inheritance.

Re-

→ code Reusability.

### Reusability:

→ Because of inheritance a method which is defined in the Super class can be accessed by derived classes. That means here we are achieving code reusability with inheritance.

#### 4. Polymorphism :-

- Polymorphism means one name many forms.
- Implementing multiple functionalities with the same name can be called as polymorphism.

##### A. Implementation

Def:

- Implementing multiple methods with the same name with different behaviour is called as polymorphism.
- Polymorphism can be classified into two ways

1. Static Polymorphism (or) Compile time Polymorphism
2. Dynamic Polymorphism (or) run time polymorphism.

##### 1. Static Polymorphism:-

- In static polymorphism, a method which will be bound at compile time will execute.
- Compile time polymorphism we are achieving in function overloading.

##### 2. Dynamic Polymorphism:-

- In dynamic polymorphism a method which is binding at runtime will execute.
- Run time polymorphism we are achieving in function overriding  
When we will go for Polymorphism?

Ans: Whenever we want to implement multiple methods with the same name in a single class (or) a combination of base & derived class we will go for polymorphism.

C Q) when we will define multiple methods with same name.

## Function Overloading :-

Def:

→ Implementing multiple methods with the same name but with the different signature in a single class (or) a combination of base and derived class is called as function overloading.

Q) what do u mean by different signature?

Ans 1. Differentiating with no. of parameters.

2. Differentiating with type of parameters.

3. Differentiating with order of parameters.

Example 1 for function Overloading.

namespace FunctionOverloadingEx1.

{

class calculate

{

internal void Add (int a, int b)

{

    int res = a+b;

    Console.WriteLine("Addition result of two integers is:" + res);

}

internal void Add (int a, int b, int c)

{

    int res = a+b+c;

```
Console.WriteLine("Addition result of 3 integers is :" + res);
```

```
}
```

```
internal void Add(double a, double b) // 3rd Method
```

```
{
```

```
    double res = a + b;
```

```
    Console.WriteLine("Addition result of 2 double values is :" + res);
```

```
}
```

```
internal void Add(double a, double b, double c)
```

```
{
```

```
    double res = a + b + c;
```

```
    Console.WriteLine("Addition result of 3 double values is :" + res);
```

```
}
```

```
}
```

## Class Program

```
{
```

```
void Main()
```

```
{
```

```
    calculate obj1 = new calculate();
```

```
    obj1.Add(10, 5);
```

```
    obj1.Add(10, 5, 2);
```

```
    obj1.Add(0.002, 0.003);
```

```
    obj1.Add(0.001, 0.002, 0.0004);
```

```
    Console.ReadLine();
```

```
}
```

- In the above program we have implemented function overloading
- When we compile the above program, ~~the~~ compiler will bind a method based on reference variable type like below.

obj. Add(10, 5); → 1st Add

obj. Add(10, 5, 2); → 2<sup>nd</sup> Add

obj. Add(0.002, 0.003); → 3rd Add

obj. Add(0.001, 0.002, 0.004); → 4th Add

→ When we run the above prgm, because of new calculate(); it is creating object of calculate.

→ Due to that reason in runtime also which are invoking the methods which are binded at compile time because here reference variable type and object type will be same.

→ Due to that reason function overloading is called as compile time binding.

→ Function overloading will depend on following 3 things

1. No. of Parameters

2. Type of Parameters

3. Order of Parameters

→ Example for no. of Parameters

```
namespace noofParameters
{
    class MyClass
    {
        internal void Print()
        {
            Console.WriteLine("zero parameter function is calling");
        }

        internal void Print(int a)
        {
            Console.WriteLine("one parameter function value is :" + a);
        }
    }

    class Program
    {
        void Main()
        {
            MyClass obj = new MyClass();
            obj.Print();
            obj.Print(10);
            Console.ReadLine();
        }
    }
}
```

→ Example for type of parameter to a function

```
namespace typeofParameters
{
    class MyClass
    {
        internal void Print(string s)
        {
            Console.WriteLine("s value is :" + s);
        }

        internal void Print(int a)
        {
            Console.WriteLine("a value is :" + a);
        }
    }
}
```

```
class Program
{
    void Main()
    {
        Myclass obj = new Myclass();
        obj.Print("satya");
        obj.Print(10);
        Console.WriteLine();
    }
}
```

→ Example of for Order of Parameters

```
namespace OrderofParameters
{
    class Myclass
    {
        internal void Print(int a, string s)
        {
            Console.WriteLine("a value is: {0}, s value is: {1}", a, s);
        }

        internal void Print(string s, int a)
        {
            Console.WriteLine("s value is: {0}, a value is: {1}", s, a);
        }
    }

    class Program
    {
        void Main()
        {
            Myclass obj = new Myclass();
            obj.Print(10, "satya");
            obj.Print("satya", 10);
            Console.WriteLine();
        }
    }
}
```

→ Function overloading will not depend on following 2 things

1. Return type of a function.
2. Parameter name.

→ Example for return type of a function.

namespace ReturnTypeoffunction  
{

class Myclass

{

internal void Print()

{

Console.WriteLine("void function is calling");

}

internal int Print()

{

return 10;

}

}

class Program

}{

→ The above code will generate a compile time error because , we cannot overload two functions because of only diff return types with this we can say function overloading will not depend on return types .

→ Example for parametername

namespace ParameterNameEx  
{

class Myclass

{ int x;

internal void Print(int a)

{

```
Console.WriteLine("a value is :" + a);
```

```
}
```

```
internal void Print(int b)
```

```
{
```

```
Console.WriteLine("b value is :" + b);
```

```
}
```

```
{
```

→ The above code will generate a compile time error because function overloading will not depend on parameter names.

→ Realtime ex for function Overloading.

```
namespace functionOverloading
```

```
{
```

```
class Employee
```

```
{
```

```
    double basic, hra, ta, da, gross;
```

```
    internal void calGrossSal(double basic, double hra, double da)
```

```
{
```

```
        gross = basic + hra + da;
```

```
        Console.WL("Software Engineer gross salary is :" + gross);
```

```
}
```

```
    internal void calGrossSal(double basic, double hra, double da,
```

```
{
```

```
        double ta)
```

```
        gross = basic + hra + da + ta;
```

```
        Console.WL("Manager gross salary is :" + gross);
```

```
}
```

```
}
```

Class Program

{

    Void Main()

{

        Employee ser~~er~~ = new Employee();

        ser.CalGrossSal(10000, 400, 300);

        Employee mgr = new Employee();

        mgr.CalGrossSal(20000, 800, 600, 400);

        Console.RL();

}

List of the Predefined overloaded methods in C# .Net.

1. WriteLine() → 19

2. Write() → 18

3. ToInt32() → 19

:

### Questions in OOPS

1. Why class?

2. Procedural oriented approach vs Object oriented approach

3. Why object?

4. Why instance variable?

5. Why static variable

6. When we will go for local variables

7. Why method return type?

8. Why parameter

9. What is the role of access modifier

10. What is the drawback of default access modifier

11. when will go for parameterized constructor
12. why static constructor does not have access modifier and parameters.
13. why we cannot change the main method name.
14. what is access modifier of object class and object class  
Object class default constructor and GetHashCode()
15. When we will declare internal and when <sup>we</sup> will go for public,  
Protected.
16. what is access modifier of console class and WriteLine()
17. Why main method is static?
18. when we will go for instance and static methods.
19. can we initialize values with the help of a method if yes  
why constructor
20. when readonly.
- 21. Can we declare static readonly? when?
22. when call by value, when call by reference, when call by out?
23. what is the importance of properties in C#-Net.
24. when we will go for constant.
- 25. what is the importance of this and base and this()  
and base() ?

→ Example for method Abstraction.

→ Hiding unwanted method in called method Abstraction

namespace method Abstraction

{

class MyClass

{

internal void Print()

{

Console.WriteLine("Zero parameter method is calling");

}

internal void Print(int a)

{

Console.WriteLine("One parameter method is calling");

}

internal void Print(int a, int b)

{

Console.WriteLine("Two parameter method is calling");

}

}

class Program

{

void Main()

{

MyClass mc = new MyClass();

mc.Print();

(Console.ReadLine());

}

→ In the above program when control is executing below statement

mc.Point();

→ It will invoke first printmethod, but will hide second and third methods which is called as method Abstraction.

→ With this we can say method hiding can be achieved with function overloading.

## 2. Function Overriding :-

→ Having multiple methods with same name, same signature in a combination of base and derived class, while overriding for base class method we should use Virtual keyword and for derived class method we should use override keyword.

Q) Can we override in single class?

Ans No.

Q) What do you mean by same signature?

Ans Same no. of Parameters <sup>and</sup> Same type of parameters and same order of parameters.

→ In function overriding both methods return type should be same.

→ Function overriding is depending on inheritance concept.

→ Example to create Super class reference variable and subclass object.

namespace inheritance Ex

{

class bc

{

int a = 10;

internal void bcmethod()

{

Console.WriteLine("bcmethod value is :" + a);

}

}

class dc : bc

{ int b = 20;

internal void dcmethod()

{

Console.WriteLine("dcmethod value is :" + b);

}

}

class Program

{

void Main()

{

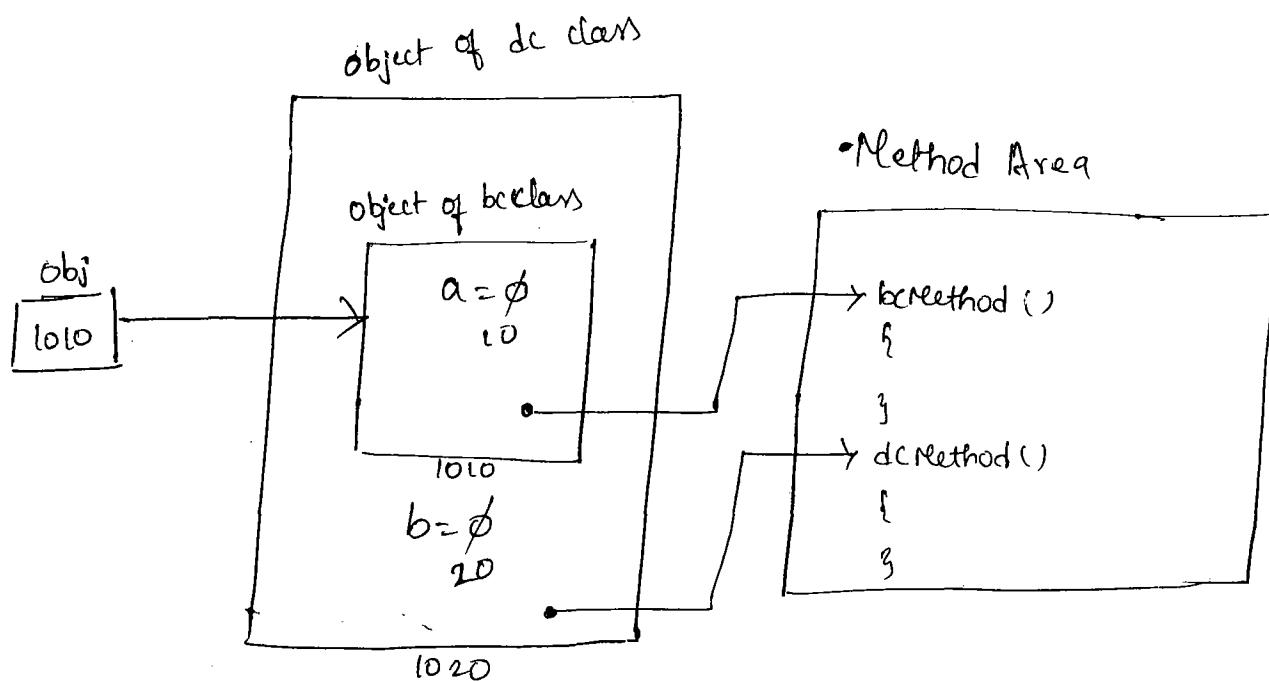
bc obj = new dc();

obj.bcmethod();

obj.dcmethod(); //error

Console.ReadLine();

}



→ Example for <sup>function</sup> overriding

namespace functionoverriding1.

```

class bc
{
internal virtual void Display()
{
    Console.WriteLine(" bc display is calling");
}

class dc : bc
{
internal void override void Display()
{
    Console.WriteLine(" dc display is calling");
}

class Program
{
    Void Main()
  
```

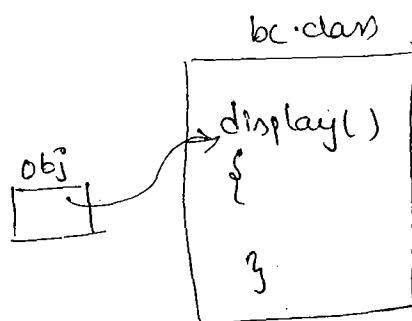
```

        {
            obj = new dc();
            obj.Display();
            Console.ReadLine();
        }
    
```

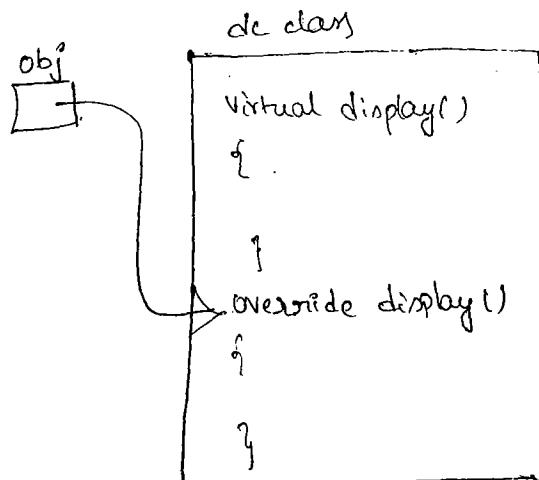
→ when we compile the above program, compiler will bind the method based on reference variable type. According to that here obj is a reference variable of bc class. Due to that reason below statement

```
obj.Display();
```

will bind to display method of bc class like below. in compile time



→ when we run the above program because of new dc(); it will bind to dc class display()



→ Here we have to identify that override method is hiding the virtual method.

→ In runtime below statement

obj.Display();

will invoke the, a method which is binding at runtime that is nothing but dc display.

→ finally we can say in function overriding a method which is binding at ~~is~~ compile time is not executing. Instead of that a method which is binding at runtime is executing. Due to that reason function overriding is called as Dynamic Polymorphism

Q1P

dc display is calling.

(Q) Can we say below is dynamic Polymorphism.

```
bc obj = new BC();  
obj.Display();
```

→ No. It is no dynamic

(Q) Can we say below statement is dynamic Polymorphism.

```
dc Obj = new dc();  
Obj.Display();
```

→ NO.

→ Real time example for function overriding.

namespace RealtimeEx

{

class Employee

{

double grossSal;

internal virtual void calGrossSal(double basic)

{

doublehra = 0.4 \* basic;

grossSal = basic +hra;

Console.WriteLine ("Employee Gross Salary is :" + grossSal);

}

}

class Manager : Employee

{

internal override void calGrossSal(double basic)

{

doublehra = 0.4 \* basic;

double da = 0.3 \* basic;

base.grossSal = basic +hra +da;

Console.WriteLine ("Manager gross salary is :" + grossSal);

}

}

class Program

{

Void Program

{

```
Employee emp1 = new Manager();  
emp1.calGrossSal(1000);  
Console.ReadLine();
```

{

O/P

Manager Gross Salary is: 1400

Q) In function overriding why super class reference variable and sub class object.

→ In function overriding we <sup>can</sup> ~~will~~ override a method in multiple derived classes like below.

namespace RealTimeEx2

{

class Employee

{

int emp;

String

Protected double grossSal;

Internal Virtual void <sup>Ex1</sup>GrossSal(double basic)

{

double hra = 0.4 \* basic;

grossSal = hra + basic;

Console.WriteLine("Employee Gross Salary is:" + grossSal);

}

```
class Manager : Employee
```

```
{
```

```
    internal override void CalGrossSal(double basic)
```

```
{
```

```
        double hra = 0.4 * basic;
```

```
        double da = 0.3 * basic;
```

```
        base.grossSal = hra + basic + da;
```

```
    Console.WL("Manager Gross Salary is :" + grossSal);
```

```
}
```

```
}
```

```
class CEO : Employee
```

```
{
```

```
    internal override void CalGrossSal (double basic)
```

```
{
```

```
        double hra = 0.4 * basic;
```

```
        double da = 0.3 * basic;
```

```
        double ta = 0.2 * basic;
```

```
        base.grossSal = hra + da + ta + basic;
```

```
    Console.WL("CEO Gross Salary is :" + grossSal);
```

```
y
```

```
z
```

```
class Program
```

```
{
```

```
    void Main()
```

```
{
```

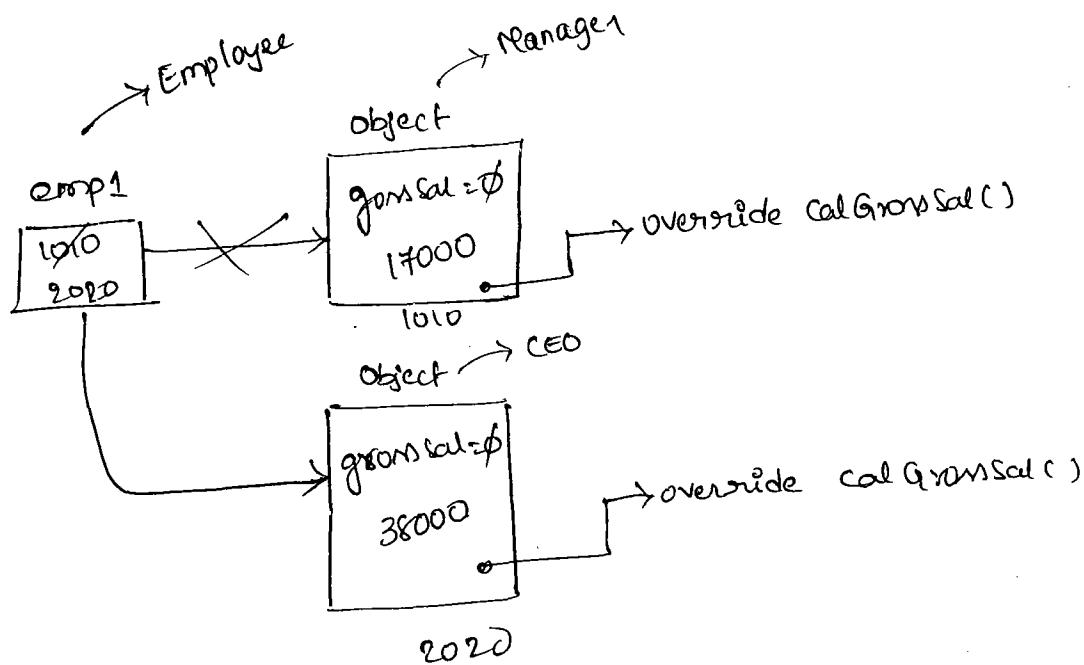
```
        Employee emp1 = new Manager();
```

```
        emp1.CalGrossSal(10000);
```

```

    emp1 = new CEO();
    emp1.CalGrossSal(20000);
    Console.ReadLine();
}

```



→ In function overriding we will create the super class reference variable and sub class object like above. Because of this we can unrefernce the previous objects always only current object will be in live and with this we can save the memory.

### Method Hiding :-

- Method Hiding and function overriding will be similar.
- In function overriding for base class method we will use **virtual** keyword for derived class method we use **override** keyword.
- But in method hiding for base class method we will not use any keyword, but for derived class method we have to use **new** keyword.

→ Hiding Superclass method in Subclass by using new keyword is called as method hiding.

→ Example for method hiding.

namespace MethodHidingEx1.

{

class bc

{

internal void display()

{

Console.WL("bc display is calling");

}

}

class dc : bc

{

internal new void display()

{

Console.WL("dc display is calling");

}

}

class Program

{

void Main

{

dc obj = new dc(); obj.display();

Console.RL();

}

qp

dc display is calling.

○ → Example 2.

namespace MethodHidingEx2

{

// defining bc, dc classes as above

class program

{

void Main()

{

bc obj = new dc();

obj.display();

console.RL();

}

olp

bc display is calling

→ whenever we want to implement superclan method in one sub class we can use method hiding.

→ whenever we want to implement superclan method in multiple derived classes better to go for function overriding.

namespace MethodHidingEx3

{

// defining bc, dc as above

olp

class tc : bc

{

new internal void display()

{

console.WL("tc display is calling.");

bc display calling

bc display calling

class Program

{ void Main()

{ bc obj = new dc(); obj.display();

obj = new tc(); obj.display(); c.RL(); }

c.td...

## Exception handling

Q) what is an Exception?

Ans An exception is nothing but runtime error.

Q) what is exception handling?

Ans Exception handling is a <sup>Predefined</sup> mechanism to handle runtime errors by using try, catch and finally blocks.

Q) Purpose of exception handling?

Ans → To handle runtime errors we will go for exception handling.

→ Whenever an error is occurred while executing the program, program execution will not terminate if we have implemented exception handling mechanism and it will display user friendly error messages.

→ To implement exception handling mechanism Microsoft is providing a collection of predefined classes for .Net programmers like below.

1. DivideByZeroException class

2. OverflowException class

3. FormatException class

4. ArithmeticException class

5. ... and so on

→ All the above predefined classes are part of System base class library.

C → Syntax for try, catch, finally blocks

Try  
{

}      ↗ Predefined  
Catch(<ExceptionClass>, <object name>)  
{

}  
finally  
{

}

→ Within the Try block we have to write the statements which may throw an error.

→ Within the Catch block we have to write the code to handle the error by displaying the user friendly message.

→ Within the finally block we have to write the error free code (Ø) the code which we want to execute irrespective of error occurrence.

Execution Flow of Try, catch and finally:

→ When there is an error

→ While executing try block when there is an error control will come out from the try block and it will execute appropriate catch block then it will execute finally block.

→ While executing try block if there is no error then it will execute full try block and it will skip catch block and it will execute finally block.

- catch block will execute only when there is an error, but finally block will execute always irrespective of error occurrence.
- A try block can follow with one catch block (or) multiple catch blocks and with finally (or) without finally
- Write a console program by using try, catch handle the divide by zero error.

```

Void Main()
{
    Console.WriteLine ("Enter first no:");
    int a = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine ("Enter second no:");
    int b = Convert.ToInt32 (Console.ReadLine());

    try
    {
        int div = a/b;
        Console.WL("div result is :" + div);
    }
    catch (DivideByZeroException de)
    {
        Console.WL("A value which cant divide by zero");
    }
    Console.ReadLine();
}

```

- D → Example to handle divide by zero error by using try, catch and finally blocks.

```
Void Main()
{
    Console.WriteLine ("Enter first no.");
    int a = int.Parse (Console.ReadLine ());
    Console.WriteLine ("Enter second no.");
    int b = int.Parse (Console.ReadLine ());

    try
    {
        int div = a/b;
        Console.WriteLine ("Division result is :" + div);
    }
    catch (DivideByZeroException de)
    {
        Console.WL ("A value which cannot be divided by zero");
    }
    finally
    {
        Console.WL ("finally block is calling");
    }
    Console.ReadLine ();
}
```

→ With the above program we can say that finally block will execute always irrespective of error occurrence.

→ Example for only try block,

```
Void main()
{
    Console.WL("Enter first no:");
    int a = int.Parse(Console.RL());
    Console.WL("Enter second no:");
    int b = int.Parse(Console.RL());
    try
    {
        int div = a/b;
        Console.WL("Division result is :" + div);
    }
    Console.RL();
}
```

→ O/p is compile time error. try block should follow with either catch block or finally block.

→ Example for try and finally.

```
Void Main()
{
    // first 4 statements same as above
    try
    {
        int div = a/b;
        Console.WL("Division result is :" + div);
    }
    finally
    {
        Console.WL("finally block is calling");
    }
}
```

Console.WriteLine();

}

→ With the help of only try and finally we cannot handle the exceptions.

### Multiple Catch blocks :-

when we will go for multiple catch blocks.

→ Whenever we are expecting more than one error from the try block, we require to implement multiple catch blocks.

→ Example for multiple catch blocks.

Void Main()

{ int a, b, div;

try

{

Console.WriteLine("Enter first no.");

a = int.Parse(Console.ReadLine());

Console.WriteLine("Enter second no.");

b = int.Parse(Console.ReadLine());

div = a / b;

Console.WriteLine("division result is :" + div);

}

Catch (OverflowException oe)

{

Console.WriteLine(oe.Message);

}

Catch (FormatException fe)

{

Console.WriteLine(fe.Message);

}

```

    Catch ( DivideByZeroException de )
    {
        Console.WriteLine (de.Message);
    }
    Console.ReadLine();
}

} Catch ( Exception ee )
{
    Console.WriteLine (ee.Message);
}
Console.ReadLine();
}

```

Example to handle all the errors by using single catch block.

→ To handle all the errors by using single catch block we will write generalized catch block, in generalised catch block we will write a class called Exception.

Exception class :-

→ It is a predefined class, which is capable to handle all the errors because Exception class is the Super class for all .Net exception classes like OverflowException class, FormatException class, DivideByZeroException class ...

Message:

→ It is a predefined member property of exception class which will display the concern error predefined message.

```

Void Main()
{
    int a, b, div;

    try
    {
        Console.WL("Enter first no:");
        a = int.Parse(Console.RL());
        Console.WL("Enter second no:");
        b = int.Parse(Console.RL());
        div = a / b;
        Console.WL("Division result is:" + div);
    }

    catch (Exception ee)
    {
        Console.WL(ee.Message);
    }

    Console.RL();
}

```

Example for empty catch block.

```

Void Main()
{
    int a, b, div;

    try
    {
        Console.WL("Enter first no:");
        a = int.Parse(Console.RL());
    }

```

```

        console.WL("Enter second no:");
        b = int.Parse(Console.ReadLine());
        div = a / b;
        Console.WriteLine("division result is :" + div);
    }
    catch {
        Console.WriteLine("error is occurred");
    }
    finally {
        Console.ReadLine();
    }
}

```

Note: empty catch block can also handle all the type of errors because it will act like a generalized catch block.

What <sup>should be the</sup> is the order of catch blocks?

→ When we are implementing multiple catch blocks first we have to write appropriate catch blocks and last catch block should be generalized catch block.

Execution flow of multiple catch blocks?

→ Whenever there is an error is occurred in try block, control will come out of the try block then it will search for the appropriate catch block, and it will execute appropriate catch block then finally....

→ If the appropriate catch block is not available then control will execute generalized catch block and finally block....

- Example to throw an exception explicitly.
- → Whenever a programmer wants to throw an exception explicitly he can throw by using throw keyword.

```
Void Main ()  
{  
    try  
    {  
        throw new DivideByZeroException();  
    }  
    catch (DivideByZeroException de)  
    {  
        Console.WriteLine(de.Message);  
    }  
    Console.ReadLine();  
}
```

List out the available predefined Exception classes in .Net.

Implement multiple examples with exception handling mechanism to handle various types of exceptions.

ctd.

namespace MethodHidingEx4

{

class bc

{

    internal virtual void display()

{

        Console.WriteLine("bc display is calling");

}

}

class dc : bc

{

    internal override void display()

{

        Console.WriteLine("dc display is calling");

}

}

class tc : bc

{

    internal override void display()

{

        Console.WriteLine("tc display is calling");

}

}

class program

{

    void Main()

{

        // as above

}

}

class bc

{

// Same as above with virtual keyword.

}

class dc : bc

{

internal new void display()

{

CWL(

}

}

class tc : dc

{

internal override void display()

{

CWL(

)

}

class program

{

void main()

{

bc obj = new tc();

obj.display();

Console.RL();

}

O/P

Compile time error.

Differences between function loading and function overriding.

### Over Loading

### over riding

→ ~~Stat~~

→ Multiple methods with same name but with different signature. → Can have multiple methods with same name and with same signature.

→ To implement in overloading we don't require any key words

→ To implement function overriding we use virtual and override keywords.

→ It is static polymorphism.

→ It is dynamic polymorphism

→ This can be implemented in single class and also in combination on base and derived class.

→ It is not possible in a single class. We have to go for base and derived class

→ Static methods can be overloaded.

→ Static methods cannot be overloaded.

### Advanced

#### Abstract class

→ While defining a class if we have used abstract ~~class~~ keyword which is called as Abstract class.

→ Abstract class is a collection of Abstract members and non-Abstract members.

what do you mean by Abstract member?

- While declaring a class member if we have used Abstract keyword which is called as Abstract member
- Abstract members are empty members which will have only declaration within the abstract class, we have to define (8) we have to implement these abstract members within the derived class by using override keyword.

what do you mean by declaration and what do mean by definition

Ex:

internal void showC()

→ declaration

definition

{  
    Console.WriteLine("show method is calling");  
}

→ A field cannot be abstract.

→ We can have abstract Methods.

→ Example to declare Abstract Method.

internal abstract void show();

→ We can have Abstract properties.

Ex:

internal abstract int x;  
{  
    get;  
    set;  
}

- We cannot have abstract constructors.
- Abstract class can contain a constructor which is non abstract
- Abstract class members access modifiers should not be private because we should implement these abstract members in derived class by using override keyword.
- static member cannot be abstract.
- By default abstract member is a virtual member.
- A class which contains normal members (Ø) non-abstract members (Ø) concrete members is called as normal class (Ø) non-abstract class (Ø) concrete class and it is fully implemented class.
- A class which contains abstract members and non-abstract members is called as abstract class which is partially implemented class.
- Abstract class cannot be instantiated but we can create reference variable.
- Example for abstract class

```
namespace AbstractClassEx1
{
    Abstract class Vehicle
    {
        internal void Start()
        {
            Console.WriteLine("Vehicle has started");
        }
    }
}
```

```
internal void Drive()
{
    Console.WL("Vehicle has been driving");
}

// abstract members

internal abstract void Park();
internal abstract string Colour
{
    get;
    set;
}

internal abstract void Stop();

class DC : Vehicle
{
    // implementing abstract members in derived class using
    // override keyword.

    internal override void Park()
    {
        Console.WL("Vehicle has parked");

        internal string colour;
        internal override string Colour
        {
            get
            {
                return colour;
            }
            set
            {
                colour = value;
            }
        }
    }
}
```

```
internal override void Stop()
{
    Console.WriteLine("Vehicle has stopped");
}

class program
{
    void Main()
    {
        Vehicle obj = new DLL();
        Obj.Start();
        Obj.Drive();
        Obj.Park();
        Obj.Colour = "Red";
        Console.WriteLine("Vehicle colour is :" + colour);
        Obj.Stop();
        Console.ReadLine();
    }
}
```

When we will declare Abstract class?

→ According to the requirement whenever we want to implement some methods in the current class and some methods required to implement in derived classes in future then we will declare particular class as a Abstract class

Example.

namespace Electricity

{

abstract class Customer

{

internal void AboutInfo()

{

Console.WL("Electricity department information");

}

internal abstract void CalBill(int totunits);

}

class IndustryCustomer : Customer

{

internal override void CalBill(int ~~totunits~~ bill)

{

int bill = totunits \* 7;

Console.WL("Your total bill is :" + bill);

}

}

class ResidentialCustomer : Customer

{

internal override void CalBill(int totunits)

{

int bill = totunits \* 5;

Console.WL("Your total bill is :" + bill);

}

}

```
class Program
{
    void Main()
    {
        customer cust1 = new ResidentialCustomer();
        cust1.AboutInfo();
        cust1.CallBill(100);

        customer cust1 = new IndustryCustomer();
        cust1.AboutInfo();
        cust1.CallBill(200);
        Console.ReadLine();
    }
}
```

### O/p

Electricity department Information

Your total bill is : 500

Electricity department Information

Your total bill is : 1400

## Interface

- To define a Interface we have to use interface keyword.
- Interface looks like a class but it has no implementation.
- An interface is a collection of abstract members, by default interface members are public and abstract.
- Syntax to define Interface:-

Interface <Interface name>

{

// abstract members

}

Note: Interface name should start with 'I'

Ex: IEmployee, IStudent - - -

- Interface members we should implement within derived class without using override keyword.
- An interface cannot contain fields.
- Interface cannot contain constructor.
- We cannot implement a property (or) method in interface.
- Interface cannot contain static members.
- Using interface we can implement multiple inheritance.
- We cannot create an object for interface but we can create reference variable for interface.

→ Example to implement Single Inheritance by using interface.

Interface IMyInterface

{

Void Print();

Int x

{

get;

Set;

}

}

Class Dc : IMyInterface

{

//Implementing Interface members

Public Void Print()

{

Console.WriteLine ("Print is calling");

}

Int x;

Public Int X

{

get

{

return x;

}

Set

{

x = value;

}

}

}

class Program

{

void Main()

{

MyInterface obj = new dc();

obj.Point();

obj.X = 100; Console.WriteLine("X value is :" + obj.X);

Console.ReadLine();

}

Output

Point is calling

X value is : 100

→ Example for Multiple inheritance by using interfaces.

Namespace InterfaceEx2

{

Interface INokia1

{

void calling();

void Receiving();

void sendReq();

void Endcall();

}

Class Nokia1100 : INokia1

{

Public void calling

{

```
Console.WL("Nokia1100 is calling");
}

public void Receiving
{
    Console.WL("Nokia1100 is receiving call");
}

public void SendMessage
{
    Console.WL("Nokia1100 is sending msg");
}

public void Endcall
{
    Console.WL("Nokia 1100 is ending call");
}

interface INokia2
{
    void WiFi();
    void Vediocalling();
}

class NokiaAsha : INokia1, INokia2
{
    public void calling
    {
        Console.WL("NokiaAsha is calling");
    }

    public void Receiving
    {
        Console.WL("NokiaAsha is receiving call");
    }
}
```

```
Public void sendMsg  
{  
    Console.WL("NokiaAsha is sending msg");  
}
```

```
Public void Endcall  
{  
    Console.WL("NokiaAsha is ending call");  
}
```

```
Public void WiFi  
{  
    Console.WL("NokiaAsha WiFi is on");  
}
```

```
Public void videoCalling  
{  
    Console.WL("NokiaAsha videoCall is supporting");  
}
```

```
class Program  
{  
    void Main()  
{  
        Nokia1 obj1 = new Nokia1100();  
        obj1.calling();  
        obj1.Receiving();  
        obj1.sendMsg();  
        obj1.Endcall();  
  
        obj1 = new NokiaAsha();  
        obj1.calling();  
        obj1.Receiving();  
    }  
}
```

```
obj1.SendMsg();
obj1.EndCall();
Inokia2 obj2 = (Inokia2) obj1;
obj2.WiFi();
obj2.VideoCalling();
Console.ReadLine();
```

{

Can we implement multiple inheritance with a combination of one class and multiple interfaces.

Yes. But first we have to write class name then we have to write interfaces names like below.

```
interface I1
```

{

}

```
interface I2
```

{

}

```
class bc
```

{

}

```
class dc : bc, I1, I2
```

{

}

when we will go for interface?

→ whenever we want to implement all the behaviours in future derived classes, we will go for interface. As well as to implement multiple inheritance, we will go for interface.

Realtime Example for interface.

namespace BankEx

{

Interface IAccount

{

Void OpenAccount (int acno, String acName, double IntAmt);  
Void withdraw (double amt);

}

class CurrentAccount : IAccount

{

int acno;

String acName;

double bal;

Internal CurrentAccount (int acno, String acname)

{

this.acno = acno;

this.acname = acname;

}

// implementing interface methods

Public Void OpenAccount (double IntAmt)

{

this.bal = IntAmt;

Console.WL ("Your account is created successfully");

Console.WL ("Your account details are:");

Console.WL ("Acct No is :" + acno);

```
Console.WL ("Acct holder name is :" + acName);
Console.WL ("Your acct current balance is :" + this.bal);
}

Public void WithDraw(double amt)
{
    if (this.bal >= amt)
    {
        this.bal = this.bal - amt;
        Console.WL ("Your withdrawn is successful");
        Console.WL ("Your current balance is :" + this.bal);
    }
    else
    {
        Console.WL ("Insufficient funds");
    }
}

class SavingAccount : Iaccount
{
    int acNo;
    String acName;
    double bal;
    Internal SavingAccount (int acno, String acname)
    {
        Console.WL ("Account is created");
        Console.WL ('');
        Console.WL
    }
}
```

```
Public void openAccount (double InAmt)
{
    if (InAmt >= 1000)
        this. bal = InAmt;
    Console.WL ("Your Account is created");
    Console.WL ("Current bal is :" + this. bal);
}
else
{
    Console.WL ("Your account cannot be created, min bal
should be 1000");
}
```

```
Public void set withdraw (double amt)
{
    if (this. bal - amt >= 1000)
    {
        this. bal = this. bal - amt;
        Console.WL ("withdraw is successful");
        Console.WL ("Your balance is :" + this. bal);
    }
    else
    {
        Console.WL ("Insufficient funds");
    }
}
```

Class Program

```
{
```

```
Void Main()
```

```
}
```

```
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z  
Iaccount cust1 = new CurrentAccount(111, "Rama");  
cust1.OpenAccount(2000);  
cust1.WithDraw(2000);  
  
cust1 = new SavingAccount(222, "Ravi");  
cust1.OpenAccount(4000);  
cust1.WithDraw(2000);  
Console.ReadLine();
```

{

}

}

## Difference between Abstract class and interface.

### Abstract class

- Abstract keyword.
- Should use override keyword in derived classes.
- Can have non-abstract members also.
- Cannot implement multiple inheritance in abstract class we give same access modifiers in base and derived class.
- Partially implemented.
- Can contain field.
- Can contain constructor.

### interface

- Interface keyword.
- No need of override keyword in derived classes.
- By default all members are abstract.
- Used mainly to implement multiple inheritance.
- In interface no need of access modifiers, by default it is public.
- No implementation.
- Cannot contain fields.
- Cannot contain constructor.

- Can implement methods which is non abstract.
- Cannot implement methods.
- Can implement a property which is non abstract.
- Cannot implement a property.
- By default access modifier of abstract class members will be private.
- By default access modifier of interface members will be public.
- By default abstract class abstract members will be virtual.
- By default interface members will be abstract.
- Can contain static members but it should be non abstract.
- Cannot contain static members.

## Benefits of OOPS

- Modularity (achieving with class).
- Reusability (achieving with inheritance).
- Extensibility (achieving with inheritance).
- Reimplementation (achieving with polymorphism).
- Security (Access modifiers)

## Memory Management

What is memory management.

→ Memory management is a process of allocating the memory and deallocating the memory.

→ In .Net memory management is handled by garbage collector.

→ Garbage collector is an integral component of CLR.

→ CLR is an integral component of .Net framework.

## Managed Heap

→ Heap is a data structure which can contain collection of objects.

→ Garbage collector is using the heap data structure for memory management due to that reason we will call it as a managed heap.

→ To perform memory management garbage collector will do 2 duties

1. Allocating the memory

2. Deallocating the memory. (i) releasing the memory.

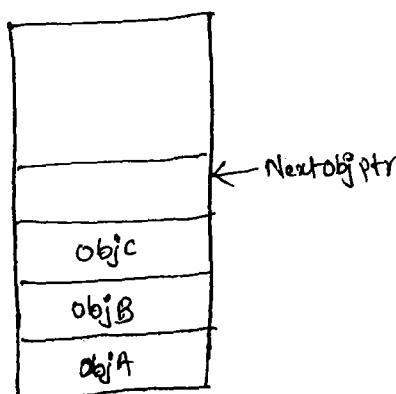
### 1. Allocating the memory :

→ When a new object is created by the application, garbage collector

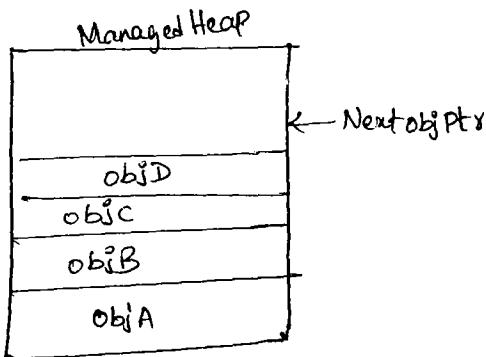
will allocate memory for newly created object within the heap data structure where exactly next object pointer is pointing.

→ Next object pointer will be pointing to the next adjacent empty location within the heap.

→ Let us assume the heap will be occupied by couple of objects like below.



→ Let us assume our application is created a new object called objD, then garbage collector will allocate memory for objD within the heap data structure where exactly next object pointer is pointing and next object ptr will be moving to the next adjacent empty location like below.



→ In this way garbage collector will allocate the memory for newly created objects within the heap data structure.

→ In this way garbage collector allocating memory for objects within the heap continually, there is a chance of out of memory.

→ To overcome this garbage collector is doing the second duty called deallocating memory (Ø) releasing memory

### a. Deallocating Memory :-

→ Deallocation is performing by the garbage collector in two steps

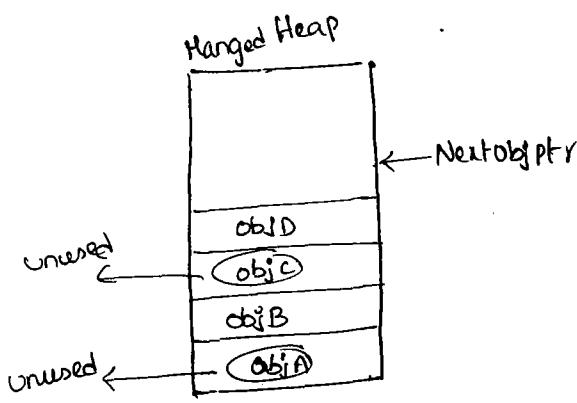
1. Recognizing the unused objects / unreferenced objects
2. Collection process.

#### 1. Recognizing the unused objects.

What is unreferenced object?

→ An object which is not pointing by any reference variable will be called as unreferenced (Ø) unused object.

→ As part of deallocation garbage collector will identify the unused objects within the heap and it will give a tag called unused object like below.



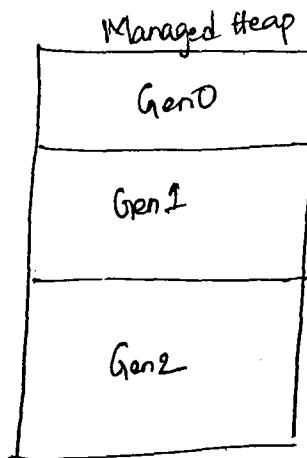
- In the above heap diagram we have 4 objects. Among 4 objA, objC are unused objects and objB and objD are used objects.
- In this way garbage collector will recognize unused objects

### a. Collection Process:-

- To implement collection process garbage collector will depend on generation algorithm.
- According to generation algorithm, managed heap will be divided into 3 generations. They are

1. Generation 0: will contain just now created objects.
2. Generation 1: will contain just before created objects.
3. Generation 2: will contain long before created objects.

- Managed heap will be divided in to 3 blocks.
- Generation 0 block space < Generation 1 block space < Generation 2 block space.



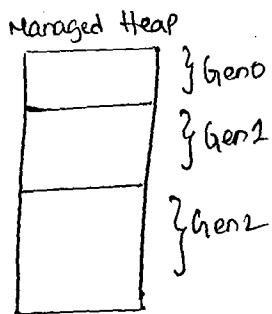
what is collection process?

→ collection process is nothing but destroying all unused objects within current generation and moving the all used objects from current generation to next generation.

Eg: generation 0 to generation 1.

How garbage collector will implement collection process?

→ Whenever an application is initialized all the generations of the managed heap will be empty like below



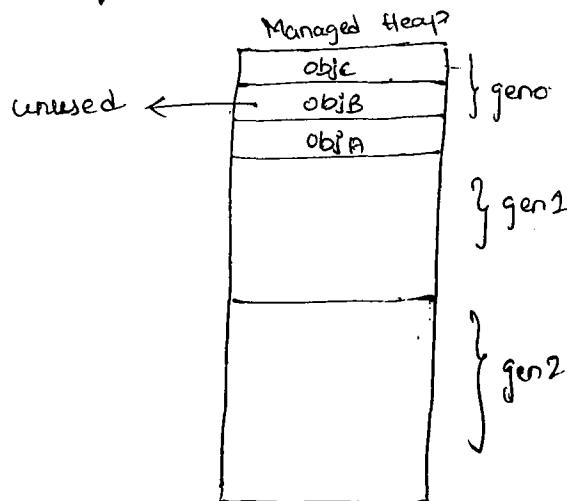
→ Whenever new objects are created by the application garbage collector will allocate memory for newly created objects within the generation 0 continually till the generation 0 is full.

→ Whenever generation 0 has no empty space then garbage collector will perform the collection process within the generation 0 is nothing but destroying the all unused objects of generation 0 and moving all used objects of generation 0 to generation 1.

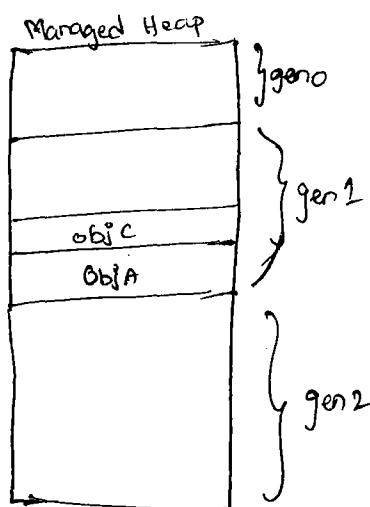
→ Whenever generation 0 and generation 1 is full then garbage collector will (and gen0 second) perform the collection process in generation 1 first means destroying all unused objects within the generation 1 and moving all used objects of generation 1 to generation 2.

→ Whenever generation 0, generation 1 and generation 2 are full then garbage collector will perform the collection process within generation 2 first here most of the objects are unused objects due to that all unused objects of generation 2 will be destroyed.

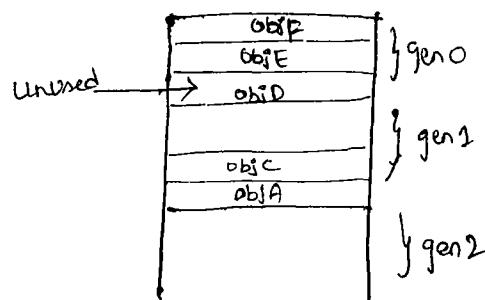
- After this it will perform a collection process in generation 1 and generation 2.
- This collection process is a cycling process.
- Finally with this we can say all new objects are allocating the memory within generation 0.
- For example let us assume we have 3 objects like below in gen 0



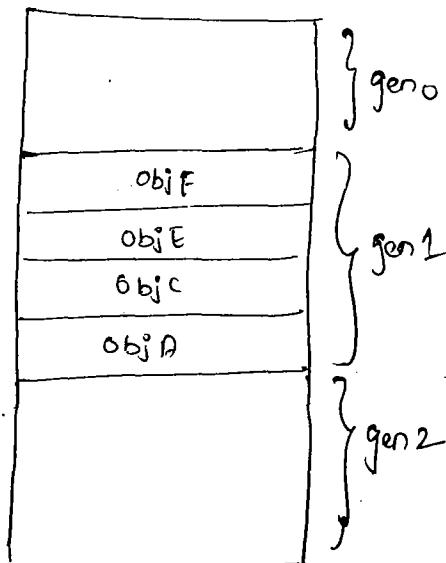
→ According to above diagram generation 0 has no space, due to that reason garbage collector will perform collection process within gen 0 like below.



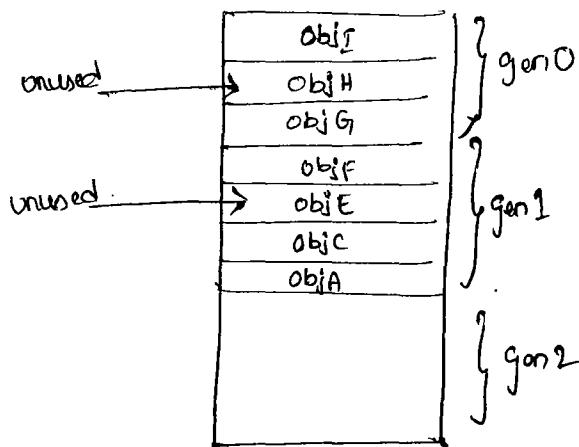
→ Now newly created objects will be allocated into generation 0 like below.



→ According to above diag gen0 is full due to that reason again garbage collector will perform collection process in gen0



→ Let us assume newly created objects will be allocated within gen0 like below



→ According to the above diagram gen1 and gen0 are full due to that reason garbage collector will implement the collection process first in gen1 and gen0

→ Whenever 3 generations are full first garbage collector will implement collection process in generation 2, generation 1, generation 0.

→ It is a cycling process

- Q Why .Net memory management is called automatic memory management.
- Memory management is performing by garbage collector internally.
- Here programmer will not involve in allocation and deallocation.
- Is there any way to invoke the garbage collector by programmer.
- Yes, like below

GC.Collect();  
 ↴      ↴  
 Class    predefined method.

→ Collect method will invoke the garbage collector for collection process.

Is there any way to destroy an object by the programmer.

→ Yes, Using dispose() method.

Ex: <Objectname>.Dispose();

### Delegates:-

→ Delegates are similar like C++ function pointers.

→ To define a delegate we have to use delegate keyword, delegates are reference types.

→ Delegate can hold the address of a function, using delegate <sup>object</sup> we can invoke a method without using ~~the~~ method name.

→ Delegates are two type

1. Single cast delegate :- It can hold address of a single method.

2. Multi cast delegate :- It can hold addresses of multiple methods.

→ Using multi cast delegate we can invoke multiple methods with a single call.

Implementation of delegate concept can be divided into following steps

## Step 1 : Defining a class

## Step 2 : Defining a method .

### Step 3 : Defining a delegate

## Syntax

Step 4 : creating object for delegate and initializing the method name.

`<delegatename> <delegate objname> = new <delegatename>(<Method name>);`

Step 5: Invoking the method by using delegate object.

<delegateobjname &gt; ;

→ Delegates are called as type function pointers because delegate declaration should follow the method declaration which is holding by the delegate object. that means method return type and signature should be same for delegate.

Note : Delegate can hold the address of static methods as well as instance methods.

Example to invoke instance methods by using delegate object.  
(single cast delegate)

## namespace SingleCast Delegate Ex1

१

class calculate

१

```

internal int add(int a, int b)
{
    return a+b;
}

internal delegate int MyDelegate (int x, int y);

class Program
{
    void Main()
    {
        calculate obj = new calculate();
        MyDelegate delobj = new Mydelegate (obj.add);

        int res = delobj(10, 5);
        Console.WriteLine ("Addition result is:" + res);
        Console.ReadLine();
    }
}

```

HW Implement single cast delegate to invoke static methods.

### Multicast delegate

→ Multi cast delegate return type should be void because it can hold ~~any~~ only void methods.

Example for multicast delegate to invoke instance methods.

namespace multicastDelegateEx1

```

class Myclass
{

```

```
    internal void Method1()
    {

```

```
        Console.WriteLine("Method1 is calling");
    }
}
```

```
internal void Method2()
{
    Console.WriteLine("Method2 is calling");
}

internal delegate void MultiDelegate();

class Program
{
    void Main()
    {
        Myclass obj = new Myclass();
        MultiDelegate delobj = new MultiDelegate(obj.Method1);
        delobj += new MultiDelegate(obj.Method2);
        delobj();
        delobj -= new MultiDelegate(obj.Method1);
        delobj();
        delobj += new MultiDelegate(obj.Method1);
        delobj();
        delobj -= new MultiDelegate(obj.Method2);
        delobj();
        Console.ReadLine();
    }
}
```

Output

Method1 is calling  
Method2 is calling  
Method2 is calling  
Method2 is calling  
Method1 is calling  
Method1 is calling

## Advantages of delegates

- Using delegates with a single call we can invoke multiple methods.
- Implementation of delegates will improve the performance of the application.

## Multiple class files

- By default every project will come with single class file but it can contain multiple class files.

Example for multiple class files

Step 1: Open a console application and rename it as multiple class files example

Step 2: Add one more class file to the solution explorer i.e., class1.cs

Step 3: Write the below code within class1.cs

namespace MultipleClassFilesExample  
{

    class Employee  
    {

        internal void Induction()

    {

        Console.WriteLine("Every employee has to attend induction training");  
    }

        internal void Appraisal()

    {

        Console.WriteLine("Every employee will have appraisal for an year");  
    }

```
internal void Nop()
```

```
{
```

```
Console.WriteLine("Every Employee has to serve notice period while  
leaving company");
```

```
}
```

```
}
```

```
}
```

Step 4 : Write below code program.cs

```
namespace MultipleClassFilesExample
```

```
{
```

```
class Program
```

```
{
```

```
void Main()
```

```
{
```

```
Employee obj = new employee();
```

```
obj.Induction();
```

```
obj.Appraisal();
```

```
obj.Nop();
```

```
Console.ReadLine();
```

```
}
```

```
}
```

```
y
```

## Multithreading :-

what is a thread?

- Thread is an independent execution path, it able to run simultaneously with other execution paths.

What is multithreading?

- Executing multiple threads simultaneously is called as multithreading.

Purpose of multithreading

- Whenever we want to execute multiple functionalities simultaneously we can go for multithreading.

Benefits of multithreading.

- We can improve the performance of the application

What we can do using a thread.

- Using thread we can start method execution, we can send method for sleep, we can suspend the method execution as well as we can call back the suspended method as well as we can terminate the method execution permanently.

What is the base class library for multithreading?

System.Threading.

How to create a user defined thread?

Thread thr1 = new Thread();



thread object

(or)

user defined thread.

## Thread

→ Thread is a predefined class which is part of `System.Threading` base class library.

→ Within Thread class Microsoft defined all the Thread related required methods called.

1. `Start()`
2. `Sleep()`
3. `Suspend()`
4. `Abort()`
5. `Resume()` and so.. on

How to initialize method to Thread

We can initialize method to Thread in two ways

1. With the help of ThreadStart delegate

→ It can be in two steps

Step 1: Creating object for ThreadStart delegate and initializing method name

`ThreadStart tstart1 = new ThreadStart (<methodname>);`

↓              ↓  
Predefined delegate    delegate  
                    object

Step 2: Creating object for Thread class and initializing delegate object like below.

`Thread thr1 = new Thread (tstart1);`

↓  
Delegate object

## 2. Without using ThreadStart delegate object

Thread thr1 = new Thread(<methodname>);

How to invoke a thread?

thr1.start();

### Start()

→ It is a predefined member method of Thread class.

→ This method will invoke the given method.

Implementation of MultiThreading will be divided into following steps

Step 1 : Defining a class

Step 2 : Defining methods.

Step 3 : Creating ThreadStart delegate object..

Step 4 : Creating Thread object.

Step 5 : Invoking method by using Thread.

What is ThreadStart()?

→ ThreadStart is a predefined delegate which is part of System.Threading base class library.

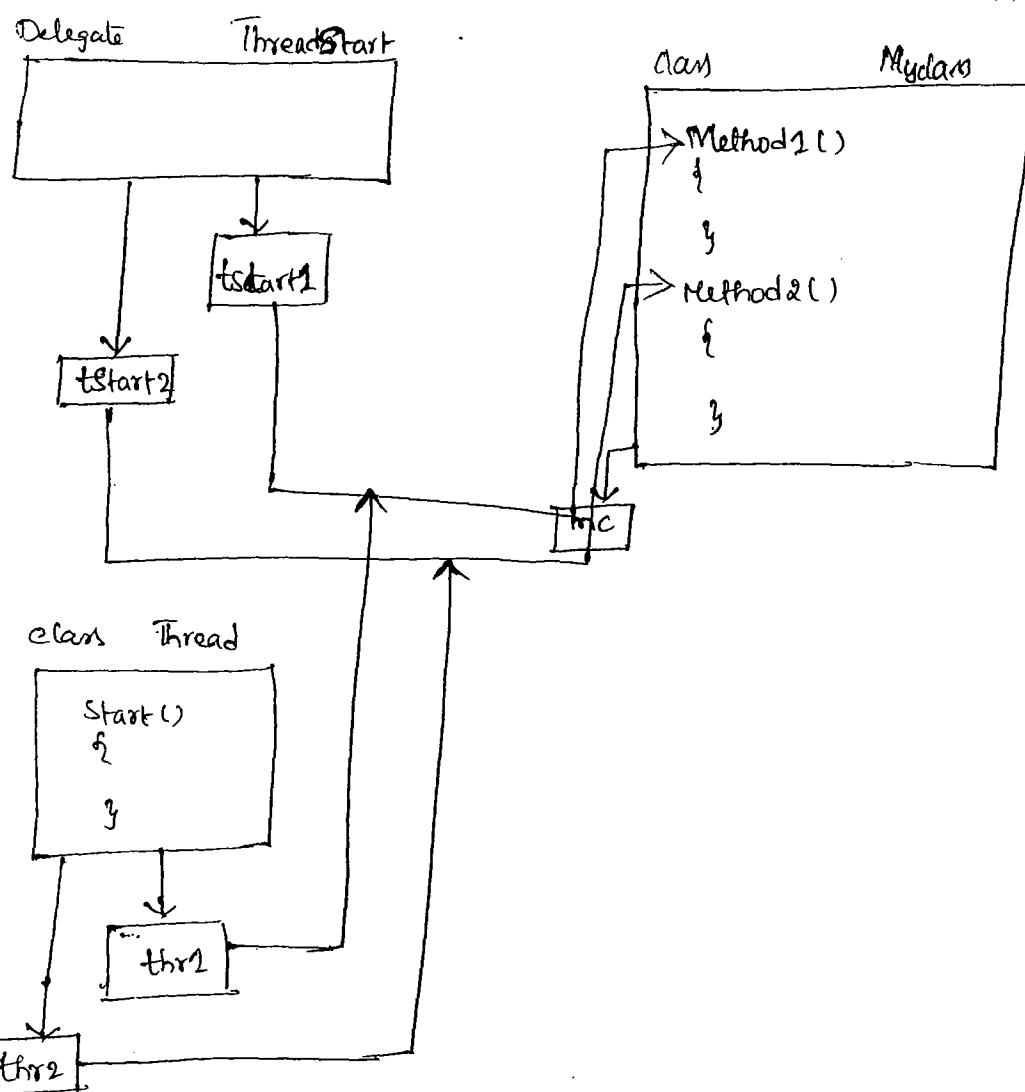
Note: Using Thread we can control static methods as well as instance method.

Example to invoke a instance method by using Thread.

```
class MyClass
{
    internal void Method1()
    {
        for(int i=1; i<=10; i++)
        {
            Console.WriteLine("Method1 i = " + i);
        }
    }

    internal void Method2()
    {
        for(int i=1; i<=10; i++)
        {
            Console.WriteLine("Method2 i = " + i);
        }
    }
}

class Program
{
    void Main()
    {
        MyClass obj = new MyClass();
        ThreadStart tstart1 = new ThreadStart(obj.Method1);
        Thread thr1 = new Thread(tstart1);
        thr1.Start();
        ThreadStart tstart2 = new ThreadStart(obj.Method2);
        Thread thr2 = new Thread(tstart2);
        thr2.Start();
        Console.ReadLine();
    }
}
```



→ Implement the above prgm as static methods.

→ Example to invoke one method by using two threads

using System;

using System.Threading;

namespace ThreadEx2

{

class MyClass

{

internal void Method()

{

for(int i=0; i<=10; i++)

{

Console.WriteLine("Method = " + i);

}

}

```

class Program
{
    void main()
    {
        Myclass mc = new Myclass();
        ThreadStart tstart = new ThreadStart(Obj. Method);
        Thread thr1 = new Thread(tstart);
        Thread thr2 = new Thread(tstart);
        thr1.start();
        thr2.start();
        Console.ReadLine();
    }
}

```

### Name :-

- It is a predefined member property of Thread class.
- Using this property we can assign the user defined name to given Thread as well as we can retrieve the userdefined name of the given thread.

### CurrentThread

static

- It is a predefined member property of Thread class which ↗
- CurrentThread property is representing the ~~curr~~ instance of the current executing Thread.

- Current executing Thread will ~~return to~~
- CurrentThread property will return the current executing Thread Object.
- CurrentThread is a readonly property.

→ Example to assign a userdefined names to threads as well as retrieving the current executing Thread name

Using System.Threading;

namespace ThreadEx3

{

class MyClass

{

internal void Method()

{

for (int i = 1; i <= 10; i++)

{

    Thread currentThread = ThreadEx3.CurrentThread;

    currentThread.Name = "Thread" + i;

}

}

}

{

void Main()

{

    MyClass obj = new MyClass();

    ThreadStart tstart = new ThreadStart(obj.Method);

    Thread thr1 = new Thread(tstart); thr1.Name = "MTR";  
    thr1.Start();

    Thread thr2 = new Thread(tstart);

    thr2.Name = "ANR";

    thr2.Start();

    Console.WriteLine();

}

## Thread.Sleep():-

→ Sleep is a static member method of Thread class.

→ We have two Sleep overloaded methods.

### 1. First Method

Thread.Sleep(int milliseconds);

→ This method will send a Thread for sleep according to given milliseconds.

### 2. Second Method

→ This method will take input as a TimeSpan object.

Thread.Sleep(TimeSpan obj);

→ We can consume in 5 ways.

1. Thread.Sleep(new TimeSpan());

2. Thread.Sleep(new TimeSpan(10));

3. Thread.Sleep(new TimeSpan(10, 20));

4. Thread.Sleep(new TimeSpan(1, 2, 20, 10));

5. Thread.Sleep(new TimeSpan(10, 3, 10, 20, 30));

→ This Second sleep method will send a Thread for long sleep.

→ Example for first sleep method.

namespace firstsleepmethodex.

{

class MyClass

{

    internal void Method1()

{

        for(int i=9; i<=10; i++)

}

```

        (Console.WriteLine(Thread.CurrentThread.Name + " is going for sleep");
        Thread.Sleep(new TimeSpan(0, 0, 20));
    }

    class Program
    {
        void Main()
        {
            // same as above prgm
        }
    }
}

```

Why sleep() is static and why start() is instance?

### Suspend()

→ This method will Suspend the given Thread temporarily.

### Resume()

→ This method will call back the Suspended Thread.

### Abort()

→ This method will terminate the Thread Execution permanently.

### Example for Suspend()

~~using~~ using System.Threading

namespace ThreadEx6

{

class MyClass

{

    internal void Method1()

{

```

Console.WriteLine(Thread.CurrentThread.Name + " = " + i);
    Thread.Sleep(20);
}
}

class Program
{
    void Main()
    {
        MyClass obj = new MyClass();
        ThreadStart tstart = new ThreadStart(obj.Method1Method1);
        Thread thr1 = new Thread(tstart);
        Thread thr2 = new Thread(tstart);
        thr1.NameName = "NTR";
        thr2.NameName = "ANR";
        thr1.Start();
        thr2.Start();
        Console.ReadLine();
    }
}

```

→ Example for second ~~no~~ Sleep method

```

namespace ThreadEx5
{
    class MyClass
    {
        Internal void Method1()
        {
            for(int i=1; i<=10; i++)
            {
                Console.WriteLine(Thread.CurrentThread.Name + " = " + i);
                if(i==5)
                {

```

```

for(int i=1; i<=10; i++)
{
    Console.WriteLine(Thread.CurrentThread.Name + "=" + i);
    if(i==5)
    {
        Console.WriteLine(Thread.CurrentThread.Name + " is going to Suspend");
        Thread.CurrentThread.Suspend();
    }
}
}

class Program
{
    void Main()
    {
        // Same as above program
    }
}

```

→ Example to abort()

Using System.Threading;

namespace ThreadExit.

```

class Myclass
{

```

internal void Method1()

```

{
    for(int i=1; i<=10; i++)
    {

```

Console.WriteLine(Thread.CurrentThread.Name + "=" + i);

if(i==5)
 {

```
Console.WriteLine("Thread.CurrentThread.Name + " is going to Abort");
Thread.CurrentThread.Abort();
}
}
class program
{
    void Main()
    {
        // Same as above prgm
    }
}
```

→ Example to call back suspended Thread.

O/P

NTR=1  
!  
NTR=5  
NTR is going to Suspend  
ANR=1  
!  
ANR=5  
ANR is going to Suspend  
NTR=6  
!  
NTR=10  
ANR=6  
!  
ANR=10

```
using System.Threading;  
namespace ThreadEx8  
{  
    class Myclass  
    {  
        internal void Method1()  
        {  
            for(int i=0; i<=10; i++)  
                Console.WriteLine(Thread.CurrentThread.Name + " = " + i);  
            if(i==5)  
            {  
                Console.WriteLine(Thread.CurrentThread.Name + " is going to Suspend");  
                Thread.CurrentThread.Suspend();  
            }  
        }  
    }  
    class Program  
    {  
        void Main()  
        {  
            Myclass obj = new Myclass();  
            ThreadStart tstart = new ThreadStart(obj.Method1);  
            Thread thr1 = new Thread(tstart);  
            Thread thr2 = new Thread(tstart);  
            thr1.Name = "NTR";  
            thr2.Name = "ANR";  
            thr1.Start();  
            thr2.Start();  
            Console.ReadLine();  
        }  
    }  
}
```

→ C# .Net will support following types of classes

1. Normal class
2. Abstract class
3. Static class
4. Sealed class
5. Partial class
6. generic class - - -

### Static class

- While defining a class if we have used static keyword which can be called as static class.
- Static class can contain only static members.
- Static class cannot be inherited.
- We cannot create an object for static class because not required.

When we will go for static class?

→ whenever a class is having all static members then we can declare particular class as a static class.

→ Example for static class.

namespace StaticClassEx.

{

Static class Myclass

{

    internal static int a;

    Static Myclass()

{

        a=10;

}

```
internal static void Show()
{
    console.WriteLine("a value is:" + a);
}
```

Class Program

```
{ void Main()
{
    Myclass.Show();
    Console.ReadLine();
}}
```

}

### Sealed class

- While defining a class if we have used sealed keyword which is called as sealed class.
- Sealed class cannot be inherited.
- We can create an object for sealed class.

Example for sealed class : Thread

When can i go for sealed class ?

→ Whenever a class required all the static and non-static members and cannot be inherited , then we will go for sealed class.

→ Example to implement sealed class.

```
Sealed class Myclass
{ }
```

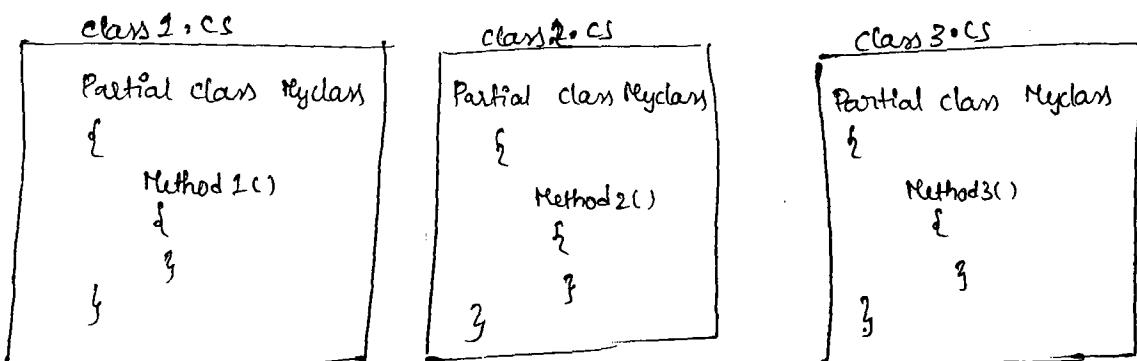
```
class C2:Myclass
{ }
```

y

→ Above code will generate an error bcoz sealed class cannot inherit.

## Partial class

- Partial class is introduced in 2.0 version.
- Partial class will split a single class into multiple class files, but class name will be same but class files name should be differ.
- For example.



When we will go for partial class?

- Whenever multiple resources want to work on single class we will go for Partial class.
- Example for Partial class

Step 1 : Open a Console application & name it as partial class Example.

Step 2 : Add 3 class files they are `class1.cs`, `class2.cs`, `class3.cs`

Step 3 : `class1.cs` code.

namespace PartialClassEx

{

Partial class MultiMedia

{

internal void `VedioRecording()`

{

`Console.WriteLine("Vedio Recording is calling");`

Step 4: class2.cs code.

Partial class Multimedia

{

internal void AudioRecording()

{

C.WL("Audio Recording is calling");

}

}

Step 5: class3.cs code

Partial class Multimedia

{

internal void ImageProcessing()

{

C.WL("Image Processing is calling");

}

}

Step 6 : program.cs code.

class Program

{

void Main()

{

Multimedia obj = new Multimedia();

obj.VedioRecording();

obj.AudioRecording();

obj.ImageProcessing();

Console.RLC();

}

}

→ In ASP.NET every webpage class is a partial class

Ex: WebForm1, WebForm2 ... WebFormN are partial classes.

→ In WindowsForms every windowsForm class is a Partial class.

Ex: Form1, Form2 ... FormN are partial classes.

for each loop :-

→ It is one of the looping concept to fetch the elements from collection.

→ Using foreach we can fetch the elements from collection without using initialization, condition, increment/decrement.

Syntax :-

```
foreach(<datatype> <Variable name> in <collection>)
{
}
```

→ Example to fetch the elements from array by using ~~as~~ foreach.

class Program

{

void Main()

{ ~~int~~ ~~for~~

int[] a = new int[5] {10, 20, 30, 40, 50};

foreach(int i int a)

{

Console.WriteLine(i);

}

Console.ReadLine();

}

## Advantage of foreach

→ Faster execution.

Implement appropriate realtime examples for for loop, while loop, do while loop and foreach loop and identify the differences b/w each loop.

Array class:- Array is predefined class defined by microsoft.

Members of predefined class are of mainly two types 1. Instance  
Instance Members 2. Static.

1. Length : Number of elements in an array.

2. Rank : This property will return no. of dimensions within given array.

3. CopyTo() :- Using this method we can copy the elements from one array to another array.

Syntax: <Objectname>.CopyTo (otherarrayname, Index)

static Members

Eg: a.CopyTo(b, 1);

1. Array.Reverse() :- Reversing the elements. Syntax: Array.Reverse (Arrayname);

2. Array.Sort() :- Sort the elements. Eg: Array.Sort (a);

3. Array.Clear() :- Clear the elements Eg: Array.Clear (a);

4. Array.BinarySearch() :- This method will Search the given elements within the array.

→ If the element is identified it will return index value of the element

→ If the element is not identified it will return negative value.

Eg: Array.BinarySearch (a, "Marry");

## Example for array class Members:

```
using System;
namespace ArrayExample
{
    class Program
    {
        void Main()
        {
            string[] names = new string[5] {"John", "David", "Marry", "Phillips",
                "francis"};
            Console.WriteLine("Array elements are:");
            foreach (string i in names)
            {
                Console.WriteLine(i);
            }
            Array.Reverse(names);
            Console.WriteLine("Reversed array elements");
            foreach (string i in names)
            {
                Console.WriteLine(i);
            }
            Array.Sort();
            Console.WriteLine("Sorted array elements");
            int index = Array.BinarySearch(names, "Marry");
            if (index >= 0)
            {
                Console.WriteLine("Searching elements name is :" + names[index]);
            }
            else
            {
                Console.WriteLine("elements are not found");
            }
        }
    }
}
// Copy to method.
String[] Enames = new String[6];
names.CopyTo(enames, 1);
```

```

B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
}

console.wL ("enames elements are:");
foreach (string i in ename)
{
    console.wL(i);
}

```

Console.wL("no.of Dimensions are :" + enames.Rank);

O/P

Array elements are:

John  
David  
Marry  
phillips  
francis

0	1	2	3	4
John	David	Marry	phillips	francis

Reversed array are:

francis  
phillips  
Marry  
David  
John

0	1	2	3	4
francis	phillips	Marry	David	John

Sorted array are:

David  
Francis  
John  
Marry  
phillips

0	1	2	3	4
David	Francis	John	Marry	phillips

No. of dimensions are : 1

Searching for an element in array : Marry

enames elements are:

—  
David  
Francis  
John  
Marry  
phillips

0	1	2	3	4	5
	David	Francis	John	Marry	phillips

No. of Dimensions are : 1.

## Assemblies

Assemblies (class libraries): class libraries are the collection of classes.

→ Assemblies are of two types.

1. Predefined class assemblies (Base class libraries)
2. User defined class assemblies.

Predefined class assemblies :

→ The assemblies which are providing by Microsoft and used by .Net programmer is called as predefined class assemblies.

e.g.: System.dll, System.Threading.dll, System.Web.dll etc...

Userdefined class assemblies:

→ These class libraries are developed by .Net programmer as part of application development.

→ In .Net class libraries are called as assemblies. An assembly can be in the form of .exe file (or) .dll file.

Difference b/w exe file and dll file.

exe file

→ exe stands for executable file

→ extension of exe file is .exe.

→ exe is self executable, exe is itself an application

→ collection of classes which contains main method will produce exe file.

dll file

→ dll stands for dynamic link library.

→ extension of dll file is .dll.

→ dll is not self executable. It will depend on some other application(exe) for execution.

→ collection of classes which doesn't have main() method will produce a dll file.

Note:- In .Net if we want to develop a dll we have to go for class library project.

## How to develop DLL's :-

→ An assembly (DLL) is a unit of code which provides versioning and deployment.

→ An assembly content can be divided into three parts.

Part 1: Manifest: Manifest is representing complete info about the assembly in the form of metadata.

→ Metadata is representing in three ways:

1. Assembly Name
2. Assembly Version
3. Public key (strong name)

Part 2: MSIL code

Part 3: References: References are representing the links to other files which are consuming by assemblies.

→ According to the scope of the assembly, assemblies are of two types.

1. Private assembly.

2. Shared assembly.

1. Private assembly: An assembly which providing services to a single client application at a time is called as private assemblies.

## Inside CLR

When we run dotnet application, .Net execution engine called CLR will load into RAM memory. As part of program execution CLR will maintain various blocks.

Among various blocks 4 are important. They are

1. Stack
2. Heap
3. Method Area
4. Execution Engine.

Stack: CLR will use stack to allocate memory for local variables.

For every method within the stack one frame will be maintaining by CLR.

Heap: CLR will use heap to allocate the memory for objects.

Method Area :- It will be divided into 3 parts like below.

Static Variables	Methods	Constructors
------------------	---------	--------------

Note: In method area only the memory will be allocated for static variables. No memory will be allocated for methods & constructors i.e., when the class is loading concern class static variables will be allocating within method area & concern class methods & constructors will be loaded into method ~~area~~.

Execution Engine: Execution engine will maintain the current executing block.

→ Example for memory allocations within CLR

```
namespace MemoryAllocationEx
```

```
class MyClass
```

```
int a = 10;
```

```
int b = 20;
```

```
internal void display()
```

```
    Console.WriteLine(a);
```

```
    Console.WriteLine(b);
```

```
}
```

```
class Program
```

```
{
```

```
void Main()
```

```
{
```

```
    MyClass obj = new MyClass();
```

```
    obj.Display();
```

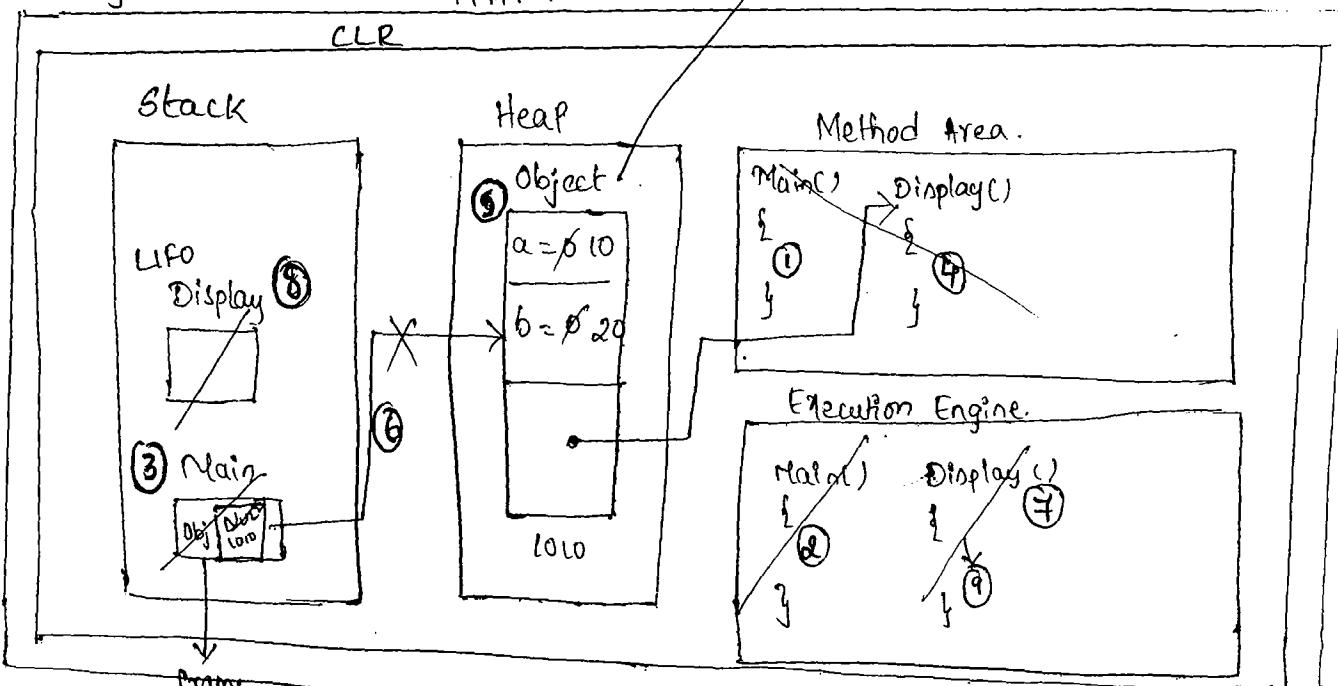
```
    Console.ReadLine();
```

```
}
```

```
}
```

RAM

MyClass



namespace CLRInside.

```
{  
    class employee  
    {  
        int eno;  
        String ename;  
        double esal;  
        static String compName;  
        static employee()  
        {  
            compName = "Microsoft";  
        }  
  
        internal employee(int eno, String ename, double esal)  
        {  
            instance variables {  
                this.eno = eno;  
                this.ename = ename;  
                this.esal = esal; } local variables  
        }  
  
        internal void display()  
        {  
            Console.WriteLine("Emp No is "+eno);  
            Console.WriteLine("Emp Name is "+ename);  
            Console.WriteLine("Emp Salary is "+esal);  
            Console.WriteLine("Company name is "+compName);  
        }  
    }  
}
```

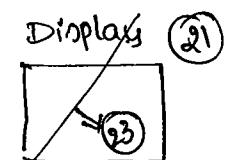
class program

```
{  
    void Main()  
    {  
        employee obj = new employee(111, "rama", 30000);  
        obj.Display();  
        Console.ReadLine();  
    }  
}
```

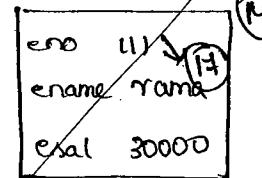
RAM

CLR

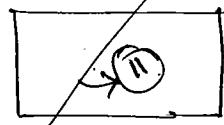
Stack



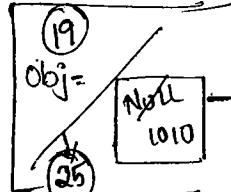
instance employee (14)



Static employee (8)



Main frame (2)



Heap

employee class

object (13)

eno = 0  
111

ename = null  
rama

esal = 0  
30000

Method Area.

Main() , Display() static employee

{ (1)

{ (5)

{ (9)

{ (3)

{ (7)

{ (11)

{ (13)

{ (15)

{ (17)

{ (19)

{ (21)

{ (23)

{ (25)

{ (27)

{ (29)

{ (31)

{ (33)

{ (35)

{ (37)

{ (39)

{ (41)

{ (43)

{ (45)

{ (47)

{ (49)

{ (51)

comp Name (4)

null (10)  
Microsoft (10)

internal employee (--)

{ (6)

{ (28)

Execution Engine

Main (3)

{ (2)

{ (4)

{ (6)

{ (8)

{ (10)

{ (12)

{ (14)

{ (16)

{ (18)

{ (20)

{ (22)

{ (24)

{ (26)

{ (28)

Static employee (9)

{ (1)

{ (3)

{ (5)

{ (7)

{ (9)

{ (11)

{ (13)

{ (15)

{ (17)

{ (19)

{ (21)

{ (23)

{ (25)

{ (27)

{ (28)

display (22)

{ (1)

{ (3)

{ (5)

{ (7)

{ (9)

{ (11)

{ (13)

{ (15)

{ (17)

{ (19)

{ (21)

{ (23)

{ (25)

{ (27)

{ (28)

HW Implement Student class Example for CLR inside Student class  
members are student id ; name , location , college name ,  
totalmarks , avg marks . Behaviours are calculate totalMarks  
(m<sub>1</sub>, m<sub>2</sub>, m<sub>3</sub>) & avgmarks in same method & dont display.  
Display student info ()

## Class libraries :-

class libraries are two types

1. Base class library. Ex: System
2. User defined class library Ex: MyConsoleApplication123

→ In .Net ~~Assem~~ class library is nothing but assembly .  
→ An assembly can be in the form of either dll file or exe file.

## Differences b/w exe and dll

### exe

- Stands for executable.
- Extension of the exe file will be .exe.
- Collection of classes which contain main method will produce exe file .
- exe is a self executable which itself an application.

### dll

- Stands for dynamic link library .
- Extension of the dll file will be .dll
- Collection of classes which doesn't have main method will produce dll file .
- dll is not a self executable which will depend on other application to execute .

## Assembly

- An assembly is a collection of classes.
- An assembly is a unit of code which provides versioning and deployment.
- Content of the assembly can be divided into 3 parts.

### Part 1:

- Manifest: → Manifest is representing the complete information about assembly in the form of metadata.
- Metadata is representing like below.
1. Assembly Name.
  2. Assembly Version
  3. Public Key (strong name)

### Part 2:

### MSIL code:

### Part 3:

References: Links to other assemblies which are consuming by this assembly.

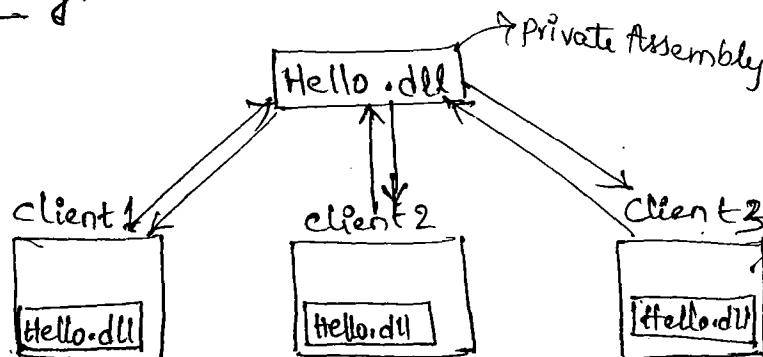
How to develop a dll file by using .Net?

Ans With the help of class library project.

### Types of Assemblies

- According to scope of the assembly, assemblies are classified into 2 types.
1. Private Assembly
  2. Shared Assembly.

### 1. Private Assembly:

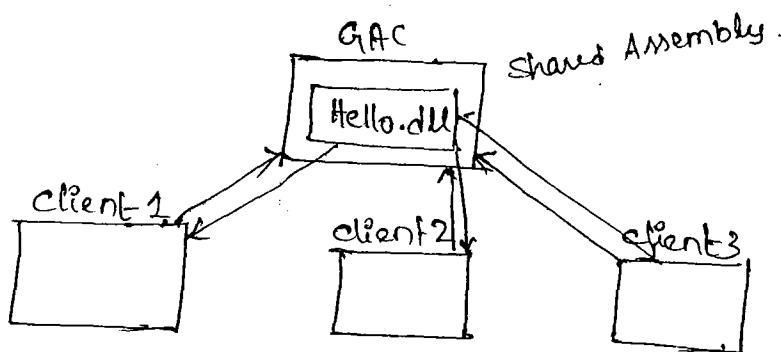


- An assembly which is providing services to single client application at a time is called as private assembly.
- Private assembly will create a local copy within client application folder, that local copy will provide the services to concern client.

## 2. Shared Assemblies:-

- An assembly which is providing services to multiple client applications at a time is called as shared assembly.
- Shared assembly will not create any local copy within client folder.
- Shared assembly will be providing services to multiple client applications from a common shared secured folder called GAC (Global Assembly Cache).

Diagram for Shared Assembly.



Implementation of private assembly divided into 4 steps:

- Step 1: Creating private assembly (i.e., class library)
- Step 2: Creating client appn (Console Appn)
- Step 3: Adding private assembly reference to client appn.
- Step 4: Consuming private assembly within client appn.

Example to implement private assembly in step by step process:

Step 1 : creating private assembly.

- open class library project rename it as my assembly location as 'D' drive.
- Rename class1.cs as calculate.cs write the code in calculate.cs.

```

namespace MyAssembly
{
    public class calculate
    {
        public int add(int a, int b)
        {
            return a+b;
        }

        public int sub(int a, int b)
        {
            return a-b;
        }
    }
}

```

→ Build Solution

→ with this process private assembly will create with in following Path :

D:\MyAssembly\MyAssembly\bin\debug.

→ With in debug folder we can identify a file called MyAssembly.dll is nothing but private assembly.

Step 2 : creating client appn.

→ open a console appn & name it has Myclient location it has D.

Step 3 : Adding MyAssembly Reference to Myclient .

→ open Myclient → select References → add → browse → D drive

→ My assembly → MyAssembly → bin → debug → MyAssembly.dll .

Step 4 : Consuming private Assembly from client appn.

→ Program.cs code .

using MyAssembly

namespace Myclient

{

class Program

{

static void Main()

{

calculate obj = new calculate();

int res = obj.add(10, 5);

Console.WriteLine("addition result is :" + res);

res = obj.sub(10, 5);

Console.WriteLine("Sub result is : " + res);

C.RLC;

y y y

→ Private assembly will create a local copy with in the every client folder. That we can check with in the following path.

D:\Myclient\Myclient\bin\Debug.

→ With in debug folder we can identify a file called Myassembly.dll is nothing but local copy of private assembly.

Drawbacks :-

→ It will create multiple local copies.

Implementing Shared Assemblies :-

→ By default an assembly is a private assembly if we want to make our assembly as a shared assembly we have to install into "GAC" folder, then Particular assembly will be converting as shared Assembly.

→ Converting from private assembly to shared assembly will be divided in to 3 steps.

Step 1 :- Creating a strong name by using strong name utility.

Q) What is strong name?

→ Strong name can be called as Public Key.

→ Strong name ~~can~~ will give the unique identity to assembly among collection of assemblies with in the GAC folder.

Q) What is strong name utility?

→ It is one of the .Net framework utility using this utility we can create a strong name to give assembly like below.

## Syntax to create strong name;

Note: we have to execute below command through visual studio .Net command prompt.

D:\>sn -k MyKeys.snk

↓      ↓  
key      file name.  
Strong name utility.

→ The above command will create a strongname, that created strong name will be storing into MyKeys.snk.

Step 2 : Signing the assembly

→ Informing abt strong name to an assembly is called as signing the assembly.

Step 3 : Installing an assembly into GAC folder by using GAC utility.

Q) What is GAC utility?

→ It is one of the .Net framework utility.

→ Using this utility we can install an assembly in to GAC folder.

→ GAC utility is represented with a file called ~~gacutil~~ gacutil.exe.

## Syntax to install an assembly to GAC folder:

D:\> gacutil : MyAssembly.dll ↴  
installing.

Implementation of shared Assembly will be divided into following steps

Step 1 : Create private assembly

Step 2 : Creating strong name.

Step 3 : signing the assembly

Step 4 : Installing an assembly into GAC.

Step 5 : creating client appn

Step 6 : Adding shared assembly reference to client appn.

Step 7 : Consuming shared assembly from client appn.

## Example to implement shared assembly:

Step 1 : creating private assembly.

→ open a class library prjt rename it as "MyCalculate" location as "E" drive.

→ write the below code in class1.cs

```
namespace MyCalculate
{
    public class calculate
    {
        Public int Mul(int a, int b)
        {
            return a * b;
        }

        Public int Div(int a, int b)
        {
            while (b != 0)
            {
                C.WFC("please enter other than 0");
                b = int.Parse(C.RLC());
            }
            return a / b;
        }
    }
}
```

→ Build Solution.

Step 2 : Creating strong name.

E:\MyCalculate\MyCalculate>sn -k keys.snk

Note : Strongname keyfile we have to create beside bin folder of one assembly.

Step 3 : Signing the Assembly.

→ write the below code in AssemblyInfo.cs file.

→ Goto the end of the and add below statement.

```
[assembly: AssemblyKeyFile("keys.snk")]
```

Rebuild the Assembly (F5) Solution.

Step 4 : Installing an assembly into GAC folder.

E:\cd Mycalculate\Mycalculate\bin\Debug

E:\Mycalculate\Mycalculate\bin\Debug\gacutil\Mycalculate.dll

Step 5 : creating client application.

→ open a console appn rename it as Myclient location as E drive

Step 6 : adding stored assembly reference to client application

Same as above Ex.

Step 7 : consuming shared assembly from client appn.

Program.cs

using Mycalculate

namespace Myclient

{ class Program

{

void Main()

{

calculate obj = new calculate();

C.WL("Enter 1st number");

int x = int.parse(C.RL());

int y = int.Parse(C.RL());

int res = obj.Mul(x,y);

C.RL("Multiplication result is;" + res);

res = obj.Div(x,y);

C.WL("Division result is;" + res);

C.RL();

y

y

→ shared Assembly will not create local copy with in the client folder  
that we can check with below path.

E:\Myclient\Myclient\bin\Debug

→ With debug folder there is no dll file bcoz here Myclient  
Mycalculate.dll is shared assembly which will provide services to

## Protected Internal :-

→ If we declare a class (8) class members access modifiers as protected Internal which can be accessed by all the classes of current prjt classes. and derived class of another prjt.

Ex : ① open a console application. rename it as protected Internal Example .

② Add a class library project to solution Explorer. rename as MyProject .

③ class.cs code .

namespace MyProject

{ Public class MyClass

{ Internal void Myfun()

{ C.WL("Internal method is calling");

}

Protected Internal void Print()

{ C.WL("Protected internal method is calling");

}

→ Build MyProject ↴

Protected Internal Ex .

Step 4 : Adding. MyProject reference to ~~protected Internal~~

Step 5 : Consume MyProject from protected Internal Ex .

## Program.cs code

using MyProject;

Namespace ProtectedInternalEx

{ class dc : MyClass

{ Internal void display()

}

```

base.Print(); //possible becoz of protected Internal
base.MyFunc(); //Not possible becoz of Internal
}
}

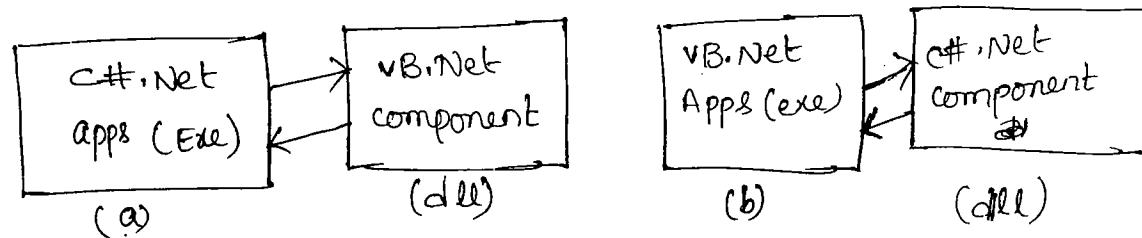
class Program
{
    void Main()
    {
        DC obj = new DC();
        obj.Display();
        C.RLC();
    }
}

```

## Language Independence :-

One .Net Language code we can use in another .Net language application due to that reasons .Net can be called as language independent platform.

→ That means while developing c#.Net application we can use VB.Net component as well as while developing VB.Net application we can consume c#.Net component.



→ Example to consume VB.NET component from C#.Net appn in step by step process.

Step 1 : Creating a VB.NET component.

→ open a class library project select language VB.NET and rename it has VBComponent location as "E" drive.

→ Below code in class1.vb.

```
Public class vbclass  
    Public sub Print()  
        Console.WriteLine("Welcome to VB.net")  
        Console.WriteLine("Welcome to VB.net");  
    End Sub  
  
    Public Function Add(ByVal a As Integer, ByVal b As Integer)  
        Return a+b  
    End Function.  
End Class.
```

→ Build solution with this process VBcomponent.dll will generate.

Step 2 :- creating c# .Net client.

→ Open a C# .Net console appn rename it as c# client  
location as "E" drive.

Step 3 : Adding VB component reference to c# client.

Step 4 : Consuming VB component from c# client.

Program.cs code.

```
using vbcomponent;  
namespace chartclient  
{  
    class program  
    {  
        static void Main()  
        {  
            vbclass objvb = new vbclass();  
            objvb.print();  
            Console.WriteLine("Enter 1st number:");  
            int x = int.parse(Console.ReadLine());  
            Console.WriteLine("Enter 2nd number:");  
            int y = int.parse(Console.ReadLine());  
        }  
    }  
}
```

```

int res = objvb.Add(x,y);
Console.WriteLine("addition result is:" + res);
Console.ReadLine();
}

```

} O/P

Welcome to VB.net.

Welcome to VB.net.

- Example to consume a C# .NET component from VB.NET client.
- Implement the all the above console program by using VB.NET.
- Implement all ASP.NET example by using VB.NET

### Generics: (Introduced in 2.0)

- Generics is similar like C++ templates.
- Generics was introduced with .NET framework .NET 2.0.
- Generics will allow the programmer to decide parameter type at the time of consumption. i.e., in declaration we will declare it as a generic type  $\langle T \rangle$  which can be consumed has  $\text{int } \langle T \rangle$ ,  $\text{float } \langle T \rangle$ ,  $\text{char } \langle T \rangle$ ,  $\text{double } \langle T \rangle$  or  $\text{string } \langle T \rangle$  according to our requirement.
- In C# .NET we can have Generic variable, Generic property, Generic constructor, generic method, Generic class.
- To implement generics we will use place holder operator ( $< >$ ) will type parameter like below  $\langle T \rangle$

Generic Function :- When we implement generics on method ( $\langle T \rangle$ ) functions then we call it as a Generic function.

Syntax :-

$\langle \text{am} \rangle \langle \text{it} \rangle \langle \text{function name} \rangle \langle T \rangle \{ T \langle \text{arg1} \rangle, - \langle \text{arg2} \rangle \}$

{  
=

?

Note: We can declare static and instance members as generic members.

Ex for Generic function :-

namespace Generic Example.

{

class MyClass //instance Method

{

internal void Display<T>(T u)

{

Console.WriteLine(u);

}

internal static void Show<T>(T d, T b)

{

C.WL(a + " + b);

}

internal void print<T, K>(T a, K b)

{

C.WL(a + " + b);

}

y

class Program

{

MyClass obj = new MyClass();

obj.Display<int>(10);

obj.Display<string>"satya");

MyClass.Show<int>(10, 20);

MyClass.Show<string>"satya", "dotnet");

obj.print<int, string>(10, "satya");

C.RL();

y

}

Note: Arithmetic operation can not be performed generic function.

→ we can declare function return type as T.

→ we can pass & different values to generic function.

## // Non void generic function

```
internal int add<T>(Ta)
```

```
{
```

```
    C.WL(a);
```

```
    return 15;
```

```
}
```

## // Generic return type

```
internal T Myfun<T>(Ta)
```

```
{
```

```
    return a;
```

```
}
```

```
class program
```

```
{
```

```
static void Main()
```

```
{
```

```
    int res = add<int>(100);
```

```
    C.WL(res);
```

```
    int res = obj.Myfun<int>(125);
```

```
    C.WL(res);
```

```
}
```

```
}
```

→ We can pass different value to generic function.

→ We can declare function return type as generic type.

→ We can have non void generic function.

## Generic class :

When we are implemented generics in class level which is called as generic class.

Syntax : class <class name> <T>

{

// generic members

}

// Non generic members.

→ In case of generic class at the time of creating object we will decide the type that object is called generic object.

→ we cannot create a normal object for generic class.

→ For every time to requirement we have to create generic object.

→ Using every generic object we can invoke non-generic methods.

## Syntax to create generic object :

<classname><Type> <object name> = new <classname><Type>();

Example for generic class with generic & non generic members :  
namespace GenericClassEx1.

{  
class Myclass<T>

{  
T a;

internal Myclass(T a)

{  
this.a=a;

y

Tx;

internal T x

{  
set

{  
x=value;

g

```

Internal void display()
{
    C.WL ("a value is :" + a);
    C.WL ("x value is :" + x);
}
class Program
{
    void Main()
    {
        Myclass<int> moint = new Myclass<int>(10);
        moint = 100;
        moint.display();
        Myclass<string> mcString = new Myclass<string>("satya");
        mcString. ds = "dotnet";
        mcString.Display();
        C.RLC();
    }
}

```

O/P

10  
100  
satya  
dotnet

### Collections :

- collections is representing collection of elements.
- Every collection will be provided by Microsoft as predefined class.
- Before .net 2.0 which have normal collections class.
- Normal class all predefined within a base class called System.Collections.

### Normal collections class: (up to 1.1)

1. Stack
2. Queue
3. ArrayList
4. Hashtable.

## Generic collections:

→ Base class library is `System.Collections.Generic`;

→ Stack < >

→ Queue < >

→ List < >

→ Dictionary < >

→ In .Net we have 2 types of collections

1. Normal collection (before 2.0)

2. Generic collection (After)

## Example to implement stack generic collection:

Class Program

{  
    void Main()

        Stack<int> mystack = new Stack<int>();

        mystack.Push(10);

        mystack.Push(20);

        mystack.Push(30);

        mystack.Push(40);

        mystack.Push(50);

        C.WL(" mystack elements are: ");

        foreach (int i in mystack)

            {  
                C.WL(i);

            }

return the top most element  
and remove it.

        C.WL(" Pop elements is: " + mystack.Pop());

        C.WL(" Peak element is: " + mystack.Peek());

→ returns the topmost element  
and without removing.

        C.WL(" mystack elements are: ");

```
foreach(int i in mystack)
```

```
{
```

```
    C.wL(i);
```

```
}
```

```
int [] arr = new int[5];
```

```
mystack.copyTo(arr, 1);
```

foreach (int i in arr) ↳ copies the stack to an existing one dimensional array.

```
{
```

```
    C.wL(i);
```

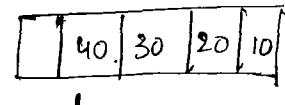
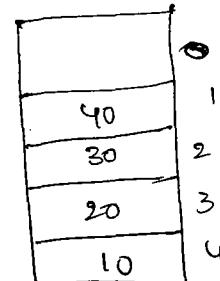
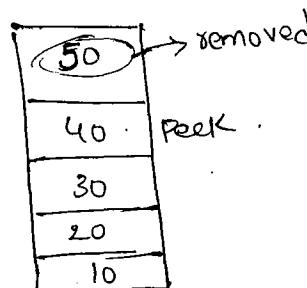
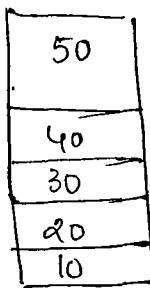
↳ removes all the objects  
mystack.clear(); from stack .

```
C.wL("No. of mystack elements are :" + mystack.count);
```

```
C.RL();
```

```
}
```

```
}
```



O/p : mystack element is: 50  
40  
30  
20  
10

Pop element is: 50

Peek Stack element is: 40

Mystack elements are : 40  
30  
20  
10

Myarray elements are : 40  
30  
20  
10

Myarray elements are ;  
0  
40  
30  
20  
10

## Generic list:

→ Generic list collection will be representing collection of elements, every element will be representing with one index value first element index value zero, second index value as 1 .....

### Example

class Program

{

Void Main()

{

list <int> mylist = new list <int>();

mylist .Add(10);

mylist .Add(20);

mylist .Add(30);

mylist .Add(40);

mylist .Add(50);

C.WL(" Mylist elements are:");

foreach(int i in mylist)

{

C.WL(i);

y → Removes the object which is first item occurrence

Mylist .Remove(10);

Mylist .RemoveAt(1);

int [ ] arr = new int[5]; ← remove the element with  
the specific index value.

Mylist .CopyTo (arr, 2);

C.WL(" My array elements are:");

foreach(int i in arr)

{

C.WL(i);

y

C.WL(" Mylist elements are:");

foreach (int i in mylist)

{

C.WL(i);

y

Mylist.clear();

C.wL ("Number of mylist elements are :" + Mylist.Count);

C.RLC;

O/P

Mylist elements are.

10

20

30

40

50

Myarray elements are.

0

0

20

40

50

Mylist elements are

20

40

50

mylist of elements : 0

### Dictionary Generic collection.

Dictionary is a collection of pairs each pair will have two values

1. key value

2. Item value.

### Example for generic Dictionary:

→ EmpDic

Key	Item
111	Rama
222	John
333	Mary
444	David
555	philips

→ namespace genericDic Ex

```
class Program
{
    void Main()
    {
        Dictionary<int, string> EmpDict = new Dictionary<int, string>();
        EmpDict.Add(111, "rama");
        EmpDict.Add(222, "John");
        EmpDict.Add(333, "mang");
        EmpDict.Add(444, "David");
        EmpDict.Add(555, "philips");

        C.WL("empdict elements are : ");
        foreach (int i in EmpDict.Keys) → property
        {
            C.WL(i + " " + EmpDict[i]);
        }

        bool res = EmpDict.Remove(333);
        C.WL("empdict elements after remove");
        foreach (int i in EmpDict.Keys)
        {
            C.WL(i + " " + EmpDict[i]);
        }

        EmpDict.Clear();
        C.WL("No . of dictionary pairs are :" + EmpDict.Count);
        C.RL();
    }
}
```

## Indexer:

- Indexer will treat an object as an array.
- Indexer will be acting as a mediator b/w array and object.
- Indexer definition will be similar like property.

## Syntax:

<am> <returntype> this [<type> <arg1>] <int> [<index>]

{  
  get  
  {  
    Current class  
    object  
  }

= // returning values from array.

  y  
  set  
  {  
    y

// assigning the values to array.

  y

## Example for Indexer..

namespace indexerEx

{  
  // Step1

  int [] EmpAges = new int [5];

  // Step2

  internal int this[int i]

  {  
    get  
    {  
      y

        return EmpAges[i];

      y

    get  
    {  
      while (value < 1 || value > 120)

      {  
        C.WL ("Plz enter the age b/w 1 to 120");

      value = (int)Console.ReadLine();  
    }

      ↑ will represent index  
      value of array

```

value = int.Parse(c.RL());
}
empAges[i] = value;
}

class Program
{
    void Main()
    {
        Myclass obj = new myclass();
        c.wL("enter emp ages:");
        for (int i = 0; i < obj.employees.length; i++)
        {
            obj[i] = int.Parse(c.RL());
        }
        c.wL("employee Ages are ");
        for (int i = 0; i < obj.empages.length; i++)
        {
            c.wL(obj[i]);
        }
        c.RL();
    }
}

```

When we will go for indexer?

→ whenever we want to assign the values to an array or retrieving values from an array from outside the class by implementing validations we will go for indexer.

What do you mean by smart array in C# .Net?

→ Indexers are called as Smart array. which makes faster assigning to array faster retrievals from arrays

Can we have static indexers in C# .NET?

→ No, because indexer concept is depending on object.



Q) Can we implement indexer when a class is having two arrays?

→ No.

Q) Can we implement indexer on multidimensional array?

→ Yes

Q) Can we implement indexer on jagged array

→ Yes.

Enum (Enumerator); -

→ Enum is value type.

→ To define enum we have to use a keyword called "enum".

→ Using enum we can create our own user defined types.

→ Enum is a collection of string constants which represents integer constraints.

Example for enum

namespace EnumEx

{  
enum weekdays

{ Monday = 1;

Tuesday = 2;

Wednesday = 3;

Thursday = 4;

Friday = 5;

Saturday = 6;

Sunday = 7;

class Myclass

{ internal static void Display(weekdays d) yesterM

{ switch(d)

{ case weekdays.Monday :

c.wL ("Today is first day of the week");  
break;

case weekdays.Tuesday :

c.wL ("Today is 2nd day of week");  
break;

case weekdays.Wednesday :

c.wL ("Today is 3rd day of week");  
break;

case weekdays.Thursday :

c.wL ("Today is 4th day of week");  
break;

case weekdays.Friday :

c.wL ("Today is 5th day of week");  
break;

case weekdays.Saturday :

c.wL ("Today is 6th day of week");  
break;

case weekdays.Sunday :

c.wL ("Today is 7th day of week");  
break;

class program

{

```
void Main()
```

```
{  
    weekdays day = weekdays.Wednesday;  
    Myclass.Display(day);  
    CRLC();  
}
```

→ Here enum will be loaded like a class.

→ enum doesn't have any return type.

→ It will create memory in stack memory.

→ Default datatype of enum is int.

Q) Can we change the datatype of Enum?

→ Yes, but it should be any one of the numerical datatype like below.

```
enum<enum name> : Long  
{  
}
```

Structure :

→ Structure is value type, when we create an object <sup>or</sup> structure it will be allocated into stack memory.

→ To define structure we have to use struct keyword.

→ Structure is a collection of members like variables, Properties, Constructors, methods and so on..

Syntax : <am> Struct <structurename>

```
{  
    // members.  
}
```

- Default access modifier of structure is internal.
- Default access modifier of structure members will be private.
- We can create an object for structure without using new keyword only when the structure does not have instance variables.
- We cannot define explicit parameterless constructor within the structure.
- We cannot initialize instance variables of a structure by using instance field initializers for this we have to go for either parameterized constructor or property.
- Structure can contain static members.
- We cannot have static structure, we cannot have abstract structure, we cannot have sealed structure.
- But we can have partial structure and generic structure.
- Structure will not support inheritance due to the reason using structure we cannot implement overriding but we can implement function overloading.
- Access Modifier of structure and structure member can not be protected and protected internal.

### Example

```
namespace structureEx.  
{  
    struct Employee  
    {  
        int eno;  
        string ename;  
        static string compName;  
        static Employee()  
        {  
            compName = "Satyam Computers";  
        }  
    }  
}
```

```
internal Employee (int eno, string ename)
```

```
{  
    this.eno = eno;  
    this.ename = ename;
```

```
}
```

```
internal void Display()
```

```
{
```

```
    C.WL ("EmpNo is :" + empNo);
```

```
    C.WL ("EmpName is :" + ename);
```

```
    C.WL ("compName is :" + compName);
```

```
}
```

```
class Program
```

```
{
```

```
    void Main()
```

```
{
```

```
    Employee emp1 = new Employee (111, "Rama");
```

```
    emp1.display();
```

```
    C.RL();
```

```
} } }
```

### Differences b/w structure and class

<u>Class</u>	<u>Structure</u>
<ul style="list-style-type: none"><li>1. class keyword</li><li>2. object will be creating with in heap memory.</li><li>3. Instance initializers are allowed.</li><li>4. To create object we should use new operator.</li><li>5. Explicit default constructor we can have.</li></ul>	<ul style="list-style-type: none"><li>1. Struct keyword.</li><li>2. object will be creating with in stack memory.</li><li>3. Instance initializers not allowed.</li><li>4. without using new operator also we can create object.</li><li>5. we cannot have explicit default constructor.</li></ul>

6. we can have static class,  
abstract class, sealed class.

7. class will support inheritance

8. class will support function  
overriding.

9. Access modifier of a class  
& class member can be any  
file.

10. class is reference type.

6. We can not have static;  
abstract class, sealed class

7. will not support inheritance

8. will not support function over-  
riding.

9. Access Modifier of a structure  
cannot be protected & protected  
internal.

10. Structure is value type.