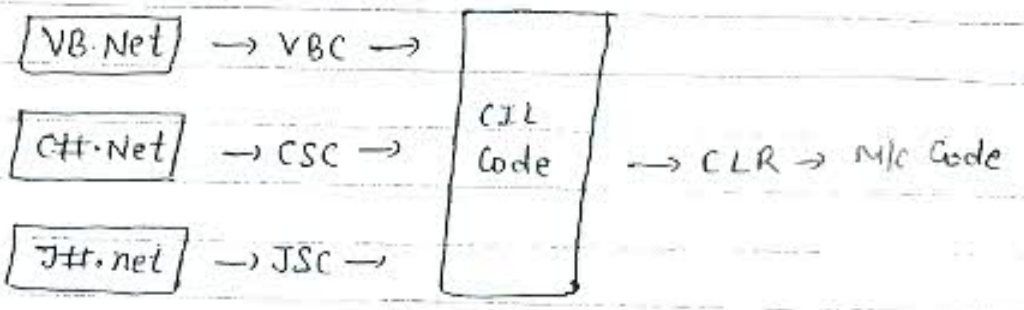
Rs. 115/-

C# By Bhargav (RaJu

Vasudev Singh
UIT-RGPV
Bhopal

Rs:- 115/-

Detailed Compilation and Execution of .Net language Application -:-



A .Net is a collection of language's. Programers have a flexibility of choosing a language according to their interest of choice and develop the application, but the code written in each different .Net language needs to be compiled using a different language compiler. But after compilation all compilers will still genrate the same type of output known as IL Code. The IL code that is genrated from any language program will be same or alike, that is why IL Code is refered as Common Intermediate language Code (CIL Code), which then taken by CLR to convert in to Machine Code.

Language Independency / Interoperability -:-

Object Oriented languages provides an interesting features under them known as code reuseability. That is the code written in one program can be consumed in other program. But in earlier object Oriented language's in C++/Java. The reuseability is avilable with in the language

only but in case of .Net language the reuseability is between all lang's of .Net because here after compilation all .Net lang's genrates the same output that what we call as CIL code.

This feature of consuming the code written in one .Net language from other .net language's is known as language independency or interoperability.

C#.Net → CIL code → Can be consumed from any .Net language's

VB.Net → CIL code → "

(.NET) Network Enabled Technology -

.Net Application -

(i) Desktop Application
→ CUI, GUI

(ii) Web Application

(iii) Mobilite Application

1. Language (C#, f#, J#, VB, VCPP etc.)

2. Technologies (ASP.net, ADO.net, etc.)

3. Server (Windows 2008 Server, SQL Server 2008, IIS, Sharepoint, Biztalk etc.)

Windows CE (Mobile OS)

.Net → It is a product of Microsoft that came into existence in 2000. Used in the development of various kinds of applications like - desktop App, web App, Mobile App etc.

To develop these application we are providing with the .Net are following -

1. Language
2. Technologies
3. Server

plateform Dependent —: C, C++, Cobol, Pascal, VB etc.

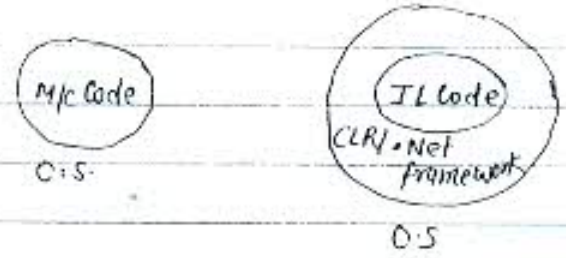
Source code → Compile → Machine Code

plateform Independent —: .Net language's

(C#, VB.net, J#, Cobol, Paskal, F#, VCPP etc.)

Source code → Compile → IL Code

IL Code → CLR → M/c Code

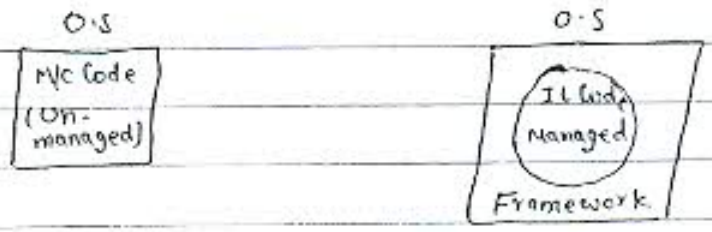


—: .NET Framework :—

⇒ To Execute .Net application on any machine, we required a s/w known as .Net Framework. Where all the .Net app. will execute under framework only.

⇒ Framework is a s/w which will masked the functionality of the O.S making the IL code to execute under it's control.
Providing the features like -

(i) Plalefoorm Independency
(ii) Security
(iii) Autometic memory Management



The code that runs under the .Net framework is refered as managed and the code that runs under O.S is unmanaged code.

-: Development of .Net Framework :-

The development of .Net has been started in late 90's with the development of .Net framework only originally under the name NGWS (next genration windows services).

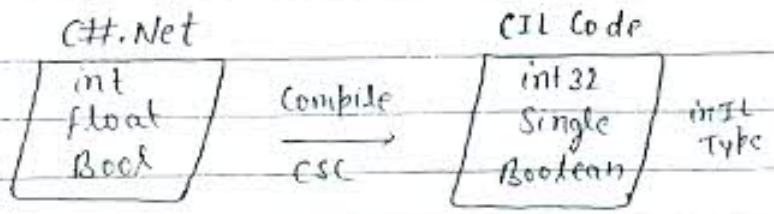
⇒ CLI Specification (Common Language Infrastructure) Specification can build or design and development the Framework.

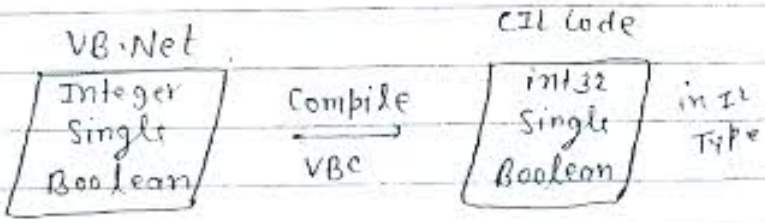
⇒ CLI speci's are openspecific that is standarised on ISO and ECMA (European Computer Manufacturer Association)

CLI spec's talks on four Major things like- CLS, CTS, BCL, and VES (Virtual exe. System)

1. CLS (Common language Specification) - A set of base rules all the .net lang's ha to adopt to interoperate or communicate with each other. Most importently after compilation, every .Net language has genrated CIL code.

2. CTS (Common Type System) - According to this all the language's of .Net have to adopt the same data type structure that is similar types should have Uniform sizing.





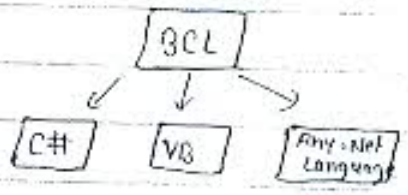
- Each .NET language is derived from some existing language so the data types name b/w the lang's will be different. But similar data types will be uniform in size. And any data type used in any language after compilation gets converted into IL Type. where in IL Types all are same.

- As any data Type is getting Converted in to IL type of the compilation, when we try to consume the IL Code From any languages. IL type is presented to that language in their understanable format only. As following-

CLS and CTS are the plateform / Foundation for language Independency or language interoperability. This is possible under .Net lang's.

3) BCL (Base class library) :- A library is a set of reusable functionality. In every Programming lang's we are provided with these libraries. But in our earlier lang's libraries of the lang's are specific to that language only.

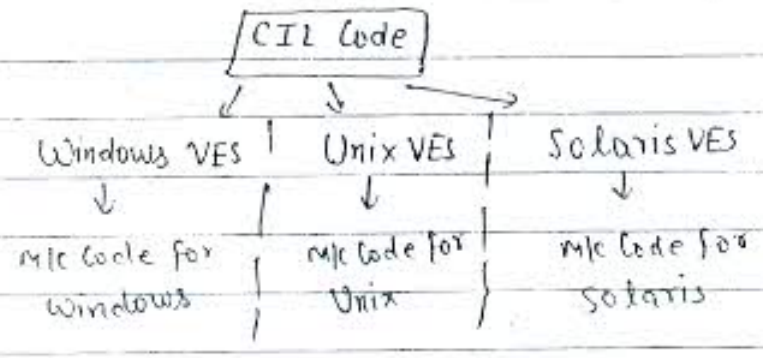
where as we are coming to .Net languages all languages are provided with the same set of libraries, what we call as BCL which can be consumed in any .Net language.



Base class libraries are implemented in C# language and being consumed from various diff. .Net lang's so this can be taken as a good example for language interoperability.

4) VES (Virtual Execution System) — This is nothing but CLR, which is responsible in converting IL code in to M/c code.

Beta → Trial version's
RTM → Released to Manufactures



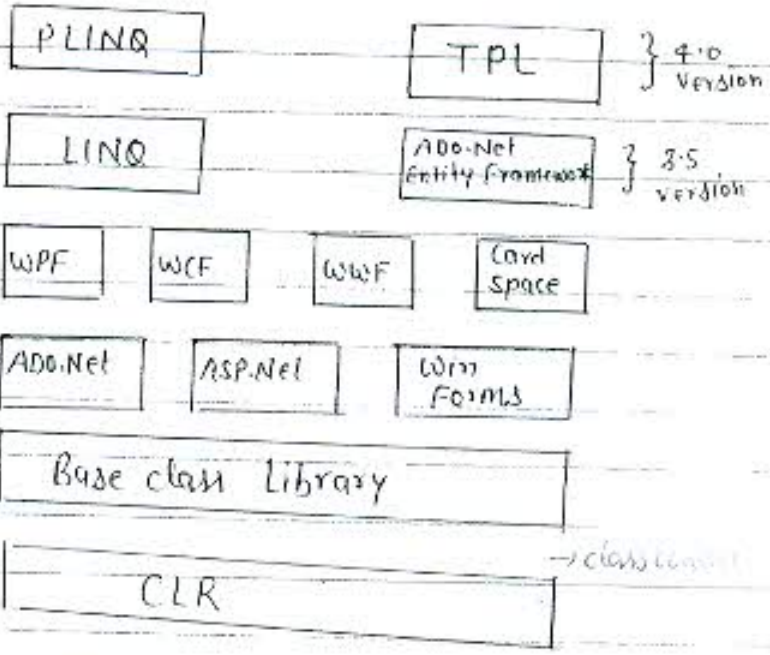
All .Net lang's after compilation gets converted in to CIL Code, which can be carried and executed on any m/c, with the help of a O.S specific VES which takes the responsibility of converting IL → M/c Code according to the OS and microprocessor.

Following the above CLI spec's, Microsoft has implimented the framework for it's windows OS only, where as the other O.S the framework's are implimented by 3rd party vendors like "MONO" to O.S's like Univ, linux, Mac, Solaries etc.

—:— .NET Framework Versions —:—

Microsoft has launched the 1st version of framework in the year 2000 as a Beta version and officiely in the year 2002. it has launched the RTM versions as Framework 1.0

2000	Framework 1.0	Beta
2002	"	1.0 RTM
2003	"	1.1
2005	"	2.0
2006	"	3.0
2007	"	3.5
2009	"	4.0



CLR (Common Language Runtime) :- It is the execution engine of .Net Framework where all .net app. Run under the super vision of this CLR. CLR guarantees various benifits to the code like Security, Memory Management, Plateform Independent etc.

1. Security Manager
2. Class Loader
3. JIT Compiler
4. Garbage Collector ← Implicit (Automatic) / Explicit
5. Exception Manager

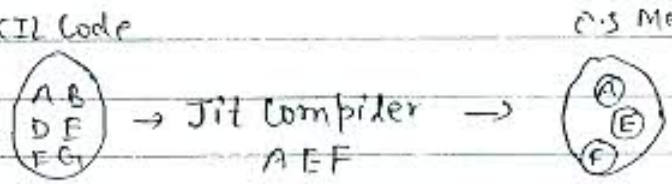
The CLR contains various things under it that are following.

1. Security Manager - which takes the responsibility of application Security.

2. Class loader - This is responsible in loading the required libraries for execution of the program from base class libraries.

(3) Jit Compiler –: It is responsible for Converting IL Code of .Net into O.S. understandable Machine / Native Code.

Adopting a process known as Conversions gradually during the program execution.



(4) Garbage Collector –: This is responsible for automatic memory management. Memory Management is a process of allocation and de-allocation of memory, that is required for a program, which is of two types-

(i) Explicit / Manual Memory Management
(ii) Implicit / Automatic Memory Management

In the 1st case programers are responsible for allocation and deallocation of memory for their program. Which has to be performed manually.

Where as in the 2nd case garbage collector is responsible for memory management, that is required in our program.

Garbage collector allocates the memory to our program when and where, it is required and de-allocates the memory that is allocated ences The object become unused in the program.

That is garbage collector reclaims the memory that is allocated to an object.

Garbage collector is designed By John McCarthy in the year 1959. To solve the problems with manual memory management.

(5) Exception Manager —: This is responsible for taking care of run-time Errors that occur in our programs.

Features of .Net —:-

(i) Language Independency :- All lang's of .Net are independent providing cross lang's communication that is the libraries of one language can be consumed from any other .Net language which follows the specification of .Net.

(ii) Base class Libraries :- These are a set of reusable functionalities, that have been provided in common for all lang's of .Net.

(iii) Common language Runtime Engine - It is the execution Engine of frame work, where all .Net applications run under the Super Vision of CLR giving you guarantie in the Area of security, memory management, Runtime Error handling etc.

4. Portability :- The design of .Net is meant to be plateform independent. A loving app. to own on different plateforms. (O.S + Microprocessor Independent)

Even Microsoft has not provided the frameworks for all O.S except windows. It has given the specifications of framework to the Industry making them open so that 3rd Party can provide compartible implementation of framework to other O.S also.

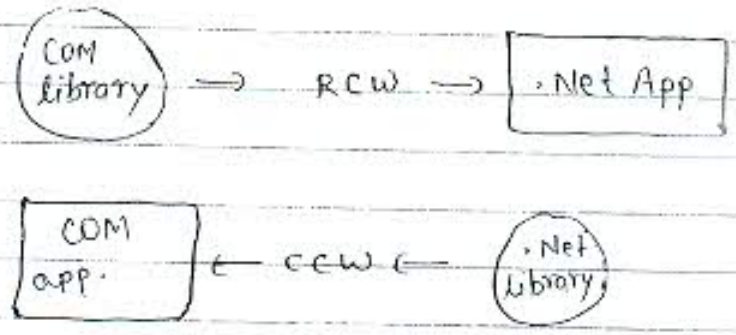
5. Simplified Deployment :- The .Net Framework includes design features and tools which help Manage the Installation of computer S/w Confirming to all Security requirments.

6. COM (Component Object Model):-

It is a specification from Microsoft introduce in 1990's which tells never build a S/w as a monolythic unit inspite it suggest building of the s/w by dividing it into smaller libraries and then integrate as a unit, which provides the benifits like S/w maintenence become eaiser and also reuseability of libraries under multiple S/w.

Microsoft .Net is also design on the same specification only. But the libraries of COM all are dependent on the O.S which can be used only on windows whereas libraries of .Net are plateform independent

6. COM Interoperability –: In Application development genrally communication between older and newer applications is commonly required so to provide access between old and new applications, we are provided with interoperability that allows using of COM libraries in .Net app. and .Net libraries in COM app.



RCW - Runtime callable Wrapper – It is a .Net compatible format of a COM library which exposes COM interfaces as .Net Interfaces to a .Net app.

CCW - COM callable Wrapper – This is opposite of RCW which exposes .Net interfaces as COM interfaces to a COM app.

obfuscation

Concerns and Criticisms related to .Net :-

(i) Managed App. which Runs under the .Net framework CLR or Java's JVM required more system resources, when compiled with Unmanaged App., which runs directly under Operating System's control.

(ii) Byte Code of Java and CIL Code of .Net app. can easily be reverse engine it in to source code then compilied with MIL code, when MIL code can not reverse engineit. Because of it they will be possible loss of trade Secreate. However to restrict this reverse engineri we are provided new tools and techniques, which can help you in restricting reverse engineering. And those tools and techniques are refered as "obfucation techinque"

(iii) When every garbage collector comes in to picture for reclaiming the memory of unused Objects, the execution of prog. gets suspended till the garbage collector complete it's work but this will not be ~~prog~~ more than milli seconds.

CSharp .Net language :- (C#.Net)

It is an object Oriented Prog. language developed by microsoft as a part of the .Net initiative and later approved by ECMA and ISO.

Anders Hejlsberg leads development of the language which has procedural object oriented syntax based on C++ and includes influences from several other programming languages most importent 'Delphi' and 'Java' with a particular emphasis on simplification.

History of the language – During the development of .Net the class libraries were originally written in a language called Simple Managed C (SMC), later the language had been renamed C# and the class libraries as well as Asp.Net runtime had been ported to C#.

C# principle designer and lead Architect Anders Hejlsberg has previously involved with the design of visual OJ++, Borland Delphi, and Turbo Paskal languages. In interviews and technical papers he has stated that flaws in most major programming language like CPP, JAVA, Delphi and small-talk drove the design of C# programming language.

Design Goals of C# -

The ECMA standard lists these design goals for C#:

1. It is intented to be a simple, mordern, genral purpose, and object Oriented programming language.

2. The language should include spong type checking, array bounds checking, detection of atlemts to use uninitilized variables, source code portability and autometic memory Mangement.

3. The language is intended for use in developing software components that can take advantage of distributed enviornments.

4. Programmer portability is very important in software industry so especially for those programmers already familier with C and C++, C# will be the best choice.

5. Support for internationalization is very important.

6. C# is intended to suitable for writing application both to hosted and Embedded system.

Versions of C# language-
1.0, 1.5, 2.0, 3.0, 4.0

Features of C# 2.0 -:-

→ Partial classes
→ Generic or parameterized types
→ Static classes
→ Anonymous Delegates
→ The accessebility of property accessors may be set independently.
→ Nullable value types.
→ Coalease operater (??) that returns the first of it's operand which is not null or null, if no such operand exists.

Features of C# 3.0 -:-

→ Language Integrated Query
→ Object initializers and collection initializers.
→ Anonymous types.
→ Implicitly-typed array and variables
→ lambda Expressions
→ Autometic properties.
→ Extension Methods.
→ Partial Methods

Features of C# 4.0 :-

→ Dynamic Programming and Lookups
→ Named and optional parameters
→ Convarience and Contra-Varience
→ Indexed properties
→ Com specific interop features.

Programming approch indifferent languages-

(i) Procedural Prog. language - In these lang's like C a program is a collection of members like variables and functions, that are defined in a program and then called from main function for execution. Because Main is the Entry point of any program.

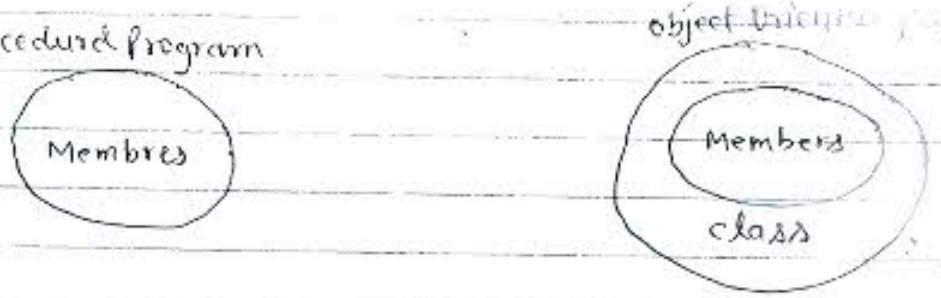
C Program

```
- collection of members
void main () <- entry point
{
call the members from here for
        execution
}
```

The draw back of procedural language is they do not provide Security and reusebility of code. So provide security and reusebility in 90's a new approch in app. development has been introduced, that is OOA (OOPS) object oriented approch.

(ii) Object Oriented Programming language's -

CPP is the 1st OOP Long's that came in to existence. In object Oriented Approch also a program is a collection of members like variables and function & the members of program must be enclosed under a special container or wrapper known as a class which provides the basic security for the content that is present inside it



A class is a user defined datatype much like a structure in our procedural programming lang's The basic diff. b/w the class and structure is a structure in procedural language's can contain only variables where a class can contain variables and function also.

Data types wheather they are predefined or user defined, we can never consume them directly

```
int = 100; //invalid
```

to consume a data type we must first create a copy of the data type which gets the required memory allocated for storing the values

```
int x = 100; // valid
```

The same rules applies in case of structure as well as a class also because there also data types only so to consume the members of class or structure first we need to create a copy of the class, which gets allocated with, the required memory for execution.

→ A copy of a simple type (like int/float) is known as a variable where as a copy of complex type like structure, class are known as objects.

CPP Program

```
class Example-
{ collection of members };
    void main() ← entry point
    {
- create the object of class
- using the object invoke members
      of class
    }
```

CPP lang's suffers from a critisism that it is not fully object Oriented because as per the rules of OOP, the code must be inside of the class only, but in CPP we never write main function inside of the class it will be outside of the class only.

If main is defined inside of the class we need to call main function also using object of class only because it becomes a member of the class but objects of class that's created inside main only so until main starts it's execution object cannot be created and until object is created main can not be called.

→ The solution for the above problem has been given in Java language with introduction of static members in a class.

→ The members of a class are now divided into static and nonstatic where static members does not required object of the class for any initilization or execution but nonstatic members required.

→ In Java language the main method will be defind as static which comes and sits inside of the class only. And now also this is my Entry point because it don't required object for execution. So once it starts execution objects get created inside of it for invoking other nonstatic members of the class.

Java Program / C# Program

```
class Example
{
Collection of members
Static void main () ← entry point
{
- create the object of the class
- using the object invoke non static
  members of class
}
}
```

Even in C# language also we follow the same approch and standards.

If a class contains only main method in it, we don't orequired to create object of the class to run the program.

Syntax to define a class -

optional

[<modifiers>] class <Name> any

{

- statements

}

Modifiers are some special key words that can be applied on a class like public, static, abstract, sealed, etc.

C# is a case sensitive language so keyword must be in lower case and if it all wants to consume any libraries they need to be used them in proper case (WriteLine) Mechanism that is one 1st character must be capital for every word.

Syntax to define Main Method -

static void [int] Main ([string[] args])

{

- statements

}

→ Main method must be declared explicitly as static if at all the execution should start from here.

→ Main method can be non value returning or it can a value return also but of type int only.

→ Main method can have parameter also but of type string Array only and passing it is also optional.

Requirements for programming -

1. 2.5 to 3.0 GB Hard disk
2. Minimum 1 GB RAM
3. Windows XP SP3
   Windows 7 (Home Premium, Ultimate)
4. Sql Server 2005 to 2008
5. Visual studio .Net 2010
6. Oracle
7. MS Office.

Books :

1. Professional C# [Wrox Press]
2. C# Unleashed [SAMS Publication]
3. C# for developers ["]
   [Ditel Serives]

Websites :

1. www.msdn.com
2. www.CSharpCorner.com
3. www.bangarraju.net
   m.bangarraju@gmail.com

iting a program in Csharp - Open Notepad and write the following code in it -

```
Class Example
{
    static void Main ()
    {
    System.Console.WriteLine ("My First C# Program");
    }
}
```

→ Save the program as Example.cs under your desired location that e.g. c:\sharp location.

→ To compile the program open Visual Studio Command Prompt first as following.

Goto start → Programs → MS Visual Studio → Visual Studio Tools → VS Command Prompt

→ After opening the command prompt go in to the folder where prog. has been saved that is C: drive csharp11 and compile the program using csharp Compiler as following.

Syntax : csc <filename>

eg : C:\csharp11 > csc Example.cs

→ Once the program is compiled successfully it genrates an output file Example.exe that contains IL code in it, run it from the command prompt as following.

C:\csharp11 > Example

Namespace is a logical container. Folder → logical, files → physical

Ildasm – Intermediate language Dis-Assembler

Using this tool we can view the content of an IL File use it from command prompt as following-

ildasm <name of .exe File>

C:\CSharp11> ildasm Example.exe

⇒ System.Console.Writeline –:-

System (Namespace) . Console (class) . Writeline (method, static)

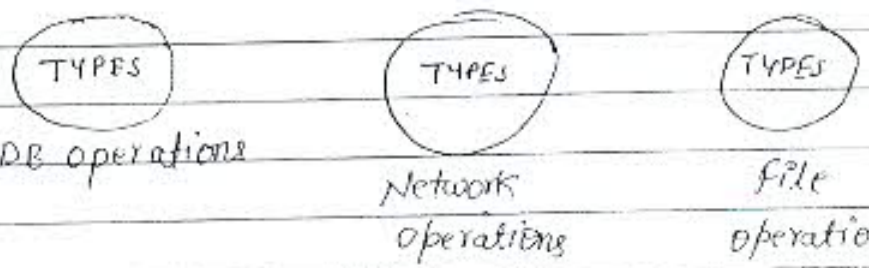
Writeline
Readline
~~function/objects/member var.~~

Console is a predefined class under the Base Class libraries that provides a set of members to perform I/O operations on standard I/O devices like Monitor and Keyboard. The class provides static member like write, writeline, Readline etc. to perform these operations.

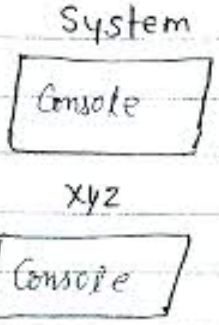
System - It is a namespace where a name-space is a logical container of types like class, structure, interface, enum and delegates.

A Namespace comes into picture two basic reasons.

(1) grouping of related types or items so that we can access them easily.



These are also used to overcome the problem of Naming Collision that is multiple types were the same name, which can be defined by placing them under diff. Namespaces.



Importing a NameSpace - If at all a class is defined under a NameSpace, we can consume that class anywhere only refering it, with it's Namespace always. To overcome this problem we are provided with a feature Importing a Namespace. If at all we import a NameSpace, we can consumes the types of that Namespace with in the program without Namespace prefix again. We need to import a Namespace in a program with the help of using statement on the top of the program.

using <namespace>

Ex. using System;
class Import Demo
{
Static void Main()
{
Console.WriteLine("Hello Vashu");
}
}

- Integer Types -

C# Type	IL Type	Size.
byte	System.Byte	0-255
Short	System.Int16	-32768 - 32767
int	System.Int32	$-2^{31} - 2^{31}-1$
long	System.Int64	$-2^{63} - 2^{63}-1$
sbyte	System.SByte	-128 - 127
ushort	System.UINT16	0-65535
uint	System.UInt32	$0 - 2^{32}-1$
ulong	System.UInt64	$0 - 2^{64}-1$

Byte, Ushort, UINT, Ulong types can store only un-signed value under them, whereas SByte, INT, long, short can store signed values under them.

- Decimal or Float Types -

float	System.Single	4 bytes
double	System.Double	8 bytes
decimal	System.Decimal	16 bytes

- Boolean Type -

bool	System.Boolean	True or False

- CHARACTER TYPES -

char	System.Char	2 bytes
string	System.String	

the size of char type has been increased to 2 byte to provide support for Unicode characters that is languages other than English like Hindi, Telgu, Tamil, frech, japanies etc.

String is a variable length type, which does not have any fixed size, the size of string depend upon the characters to assigned to it.

- Base Type

object	System.object

object is the parent of all datatypes which can store any type of value in it. And more above this is also a variable length type.

declaring a datatype -

[<modifiers>] [const] [Readonly] <type> <var> [= value] [,....n]

```
int x;
int Y = 100;
string S1, S2 = "hello", S3;
public bool flag;
const float Pi = 3.14f;
readonly double d = 3.14;
Const decimal de = 3.14m;
```

→ The default scope of any member in C# is private.

→ A variable declared here will not have any garbage values. where neumeric variable is initilized with zero, boolean variables with false and string and object with NULL.

→ The const and readonly variable are similar in behaviour whose value can not be modified once after initilization.

→ Every Decimal value is bydefault treated as Double so use f as suffix to represent it as a float and m is a suffix to represent it as a Decimal.

```
using system;

class AddNums
{
  static void Main()
  {
    int x,Y,Z;
    X= int.Parse(Console.Readline());
    Y= int.Parse(Console.Readline());
    Z= X+Y;
Console.Writeline("The sum of {0} and {1} is {2}",
                                    X,Y,Z);
  }
}
```

The Readline method of console class read the input given by the user and return it in the form of a string.

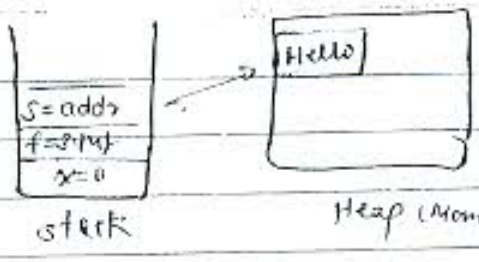
Parse is a method which convert a given string expression in to the type on the which mathood is called.

int.Parse (string) → Converts To Int
double.Parse (string) → Converts To Double
bool.Parse (string) → Converts To Bool

Calling the method Parse on incompatible value will give you a Error.

```
int.Parse ("100");  // Valid
int.Parse ("100A"); // Invalid (Error)
```

6/02/17



Data types are divided in to two catagorie -

(i) Value Types.

(ii) Reference Type.

Value Type stored the data on stack, which is a place where data stors in fixed length. such as int, float etc.

Stack is a data structure that works on a principle LIFO. And this data structure is in the control of OS. where every program has it's bathroom stack which is not shared with any other program.

STack can be deallocated only in the END OF Program execution. but faster in access.

Reference type is stored on heap memory, which is the other memory location where data stored, C# supports two predefind reference type object and string.

In .Net the heap is more manage and called as managed heap which will be under the control of garbage collector.

Ex- int x;
float f = 3.14f;
string s = "Hello";

Nullable value types :- This are new introduced in C# 2.0, which allows storing the Null value under value type. By default Null value can be stored only under reference type but not value type.

The advantage of storing Null values in a value type provides improved interaction while working with databases, because databases allow storing of Null values both under value types and reference type also.

To Declare a nullable value type, the type name suffix with a question mark.

```
string s = Null;  // valid
int x = null;     // invalid
int? x = null;    // valid
```

Implicitly typed variables :-

This feature has been added in C# 3.0 which allows to declare a variable using var keyword. where the data type of variable is implicitly recognized with the type of value that is assign to it.

```
var x = 100;      // x is of type int
var s = "Hello";  // s is of type string
var f = 3.14f;    // f is of type float
var y;            // invalid
```

We can not declare any variable with var variable without assinging a value.

```
Using System;

class TypeDemo
{
    Static void Main()
    {
    var f = 3.14f;
    Console.WriteLine (f.GetType());
    var d = 3.4;
    Console.WriteLine (d.GetType());
    var de = 3.14M;
    Console.WriteLine (de.GetType());
    var s = "Hello";
    Console.WriteLine (s.GetType());
    }
}
```

GetType is a predefined method which tells you or returns the Type of a given variable or object.

09/02/12

Boxing - And UnBoxing -:

If a value type is converted into a reference type, we call this process as Boxing.

Ex- int x = 100;
object obj = x; // Boxing

If the value type which is converted in to reference type is again converted back in to a value type, we called as UnBoxing. But here explicit typecasting is required.

Ex. int y = Convert.Int32 (obj); // Un-Boxing

Operators in c# :-

Comparision -
==, !=, <, >=, >, <=, is, as, like

Assignment -
=, +=, -=, %=, *=, /=

Airthmetic -
+, -, *, %, /

Concatenation -
+

Increment and decrement -
++, --

Logical
&&, ||, ^

A block of code that is executed basis on condition.

Conditional Statement in C#.

A block of code that gets executed basis on a condition is known as Conditional statement. These are of two types-

a) Conditional Braching (if/switch case)
b) Conditional looping (for/while)

(a) These statements allow you to branch your code depending on whether certain condition were met or not. C# has two constructs for branching code, the if statement which allows you to test whether a specific condition is met, and the switch statement which allows you to compare an expression with a number of different values.

```
if (<condition>)
        <stmts>;
else if [<condition>)
        <stmts>;
- - - - - -
else
        <stmts?;
```

Ex-

```
Using System;

Class IfDemo
{
  Static void Main ()
  {
    int x, y;
```

```
Console.Write ("Enter x value:");
x = int.Parse (Console.Readline());

Console.Write ("Enter y value:");
y = int.Parse (Console.Readline());

if (x>y)
  Console.Writeline ("x is greater");
else if (x<y)
  Console.Writeline ("y is greater");
else
  Console.Writeline ("Both are equal");
}
}
```

```
Switch (<expression>)
{
  case <value>:
    <stmts>;
    break;
  - - - - - -
  default:
    <stmts>;
    break;
}
```

Note - Using a break statement after a case block is optional in our traditional language like C++, whereas in case of C# Language it is mandatory to use break after every case. Even after default block also.

Example.

```
Using System;

    Class SwitchDemo
    {
    static void Main()
    {
Console.WriteLine("Enter the student No (1-3):");
int sno = int.Parse(Console.ReadLine());

switch (sno)
{
    case 1:
Console.WriteLine("Student no.1");
    break;

    case 2:
Console.WriteLine("Student no.2");
    break;

    case 3:
Console.WriteLine("student no.3");
    break;

    default:
Console.WriteLine("Invalid Code/Stu.No.");
    break;
}
}
}
```

(b) Conditional loops :- C# provides four different loops that allows us to execute a block of code repeatedly untill a certain condition is met, those are

- for loop
- while loop
- do while loop
- foreach loop

Every loops requires three things in common.

(i) Initilization - that sets the starting point of loop.

(ii) Condition - that sets the ending point of the loop.

(iii) Iteration - which takes you to next level of the cycle, either forword or backwork direction.

```
for (initilizer ; condition ; iterator)
    <stmts>;
for (int i=0 ; i<=10 ; i++)
    Console.WriteLine (i);
```

```
while (condition)
    <stmts>;
int x=1;
while (x<=100)
{
Console.Writeline (x);
    x++;
}
```

```
do { <stmts>; }
while (condition);
int x=1;
do {
Console.Writeline (x);
    x++;
} while (x<=100);
```

Note- In case of for / while the execution starts after checking the condition so if the condition gets statisfied then only the code gets executed. So the minimum number of execution will be zero

Whereas in case of do while loop after per completing the 1st execution then only, the condition gets verified so the minimum number of execution here will be one.

foreach loop- This is specially designed for processing the value's of Arrays and collections.

```
foreach ( type var in coll/array )
        {
        <stmts>;
        }
```

Jump Statements :- C# provides a number of statements that allows you to Jump immediately to another line in a program. Those are -

- → goto
- → break
- → Continue
- → Return

Array -
Set of similar type
value :
Collection - dynamic array

Goto - It allows you to jump directly to the another specified line in the program, indicated by a label which is an identifier followed by a colon.

```
goto xxx;
Console.Writeline ("Hello");
xxx:
Console.Writeline ("Go to statement Called");
```

output :-
Go to statement Called

break. It is used to exit from a case in a switch statement and also used to exit from any conditional loop statement which is with switch control to the statement immediately after end of the loop.

```
for (int i=0; i<=100; i++)
{
Console.WriteLine (i);
if (i==50)
  break;
}
Console.Writeline ("End of the loop");
```

Continue - This Jump Statement is mostly used in a for loop for skipping few iterations of a cycle. That is whenever Continue executes control directly transfer to the iteration part without executing any statement that is present after it.

```
for (int i=1 ; i<=100 ; i++)
{
  if (i==7)
    continue ;
  Console.WriteLine(i);
}
Console.WriteLine ("End of the loop");
```

Return :- It's a jump stmts that can jump out of a method or function in execution. while jumping out it can also carry a value out of the method or function.

Ex.

```
Using System;

  Static void Main()
{
Console.Write ("Enter a neumeric Num.");
int no. = int.Parse ("Console.ReadLine());
 if (no == 0)
   oreturn;
for (int i=1; i<=10; i++)
Console.WriteLine ("{0} * {1} = {2}", no., i, no*i);
}
```

Array :- An Array is a set of similar types values, that are stored in sequential order. We can arrage the values of an Array either in one dimensional, two Dimensional and Jagged.

→ We access the values of an Array, using index positions were the array index starts at zero, that means 1st item of an Array will be stored zero (0th) position and the position of last item of an Array will be total no of items -1 (n-1).

→ In C# array can be declared as fixed length or dynamic, fixed length array can stored predefined number of items while size of dynamic Arrays increase as you add, number of items / new items to the Array.

One dimensional Array - These array stores the data in the form of a row, which would be declared as following.

```
<type>[] <name> = new <type>[size];
int[] arr = new int[4];
or
int[] arr;
arr = new int[4];
or
int[] arr = {list of values};
```

The memory allocation of an array is performed only with the use of new operator or assignment of values.

```
Using System;

class SDArray.
{
   Static void Main()
  {
   int[] arr = new int[6];
   for (int i=0; i<6 ; i++)
   Console. Writeline (arr[i] + " ");
   Console. Write Line ();

arr[0] = 10 ; arr[1] = 20 ; arr[2]=30;
arr[3] = 40 ; arr[4]=50 ; arr[5]=60;

   for each ( int i in arr)
      console.Write (i+" ");
   }
}
```

For-each loop :- This is specially designed for accessing the value of an array or a collection. where for each iteration of the loop one value of the array is assigned to the loop variable in a sequence and the return to you.

diff. b/w For and foreach loop —

→ In case of a for loop the variable of the loop refers to index of the array. whereas in case of foreach variable of the loop refers to values of the array.

array is always a reference type.

→ In case of for the loop variable. be always in int type only. whereas in case of foreach the loop variable will be having the same type of value present in arrays.

→ for loop can be used both for accessing as well as assigning of values also whereas foreach loop can be used only for accessing the values but not assigning of values.

Note - An Array is a reference type whose memory is allocated on managed heap memory. Because in case of arrays also at the time of declaration if size is not specified, we will not know how much memory has to be allocated.

Array class :- This class is defined under the System Name space, which contains a set of members using which, we can perform operations on an array.

Array class:
- Sort (<array>)
- Reverse (<array>)
- Copy (src, dest, n)
- GetLength (int)
- Length

```
Using System;

Class SDArray2
{
static void Main()
 {
int[] arr = {12,96,16,25,21,28,85,63,15,4,2,89};
for (int i=0; i<arr.length ; i++)
    Console.Write (arr[i] + " ");
    Console.WriteLine();

    Array.Sort (arr);
    foreach (int i in arr)
     Console.Write (arr[i]+ " ");
     Console.WriteLine();

    Array.Reverse (arr);
    foreach (int i in arr)
    Console.Write (i + " ");
    Console.WriteLine();

    int[] brr = new int[10];
    Array.Copy (arr, brr, 5);
    foreach ( int i in brr)
     Console.Write (i + " ");
 }
}
```

Two Dimensional Array :- These Array stores the data in the form of rows and colums which has to be declared as following -

```
<type> [,] <name> = new <type> [rows, cols];
```

```
int [,] arr = new int [3,4];
```
or
```
int [,] arr ;
arr = new int [3,4];
```
or
```
int [,] arr = { list of values };
```

```
using System;

class TDArray
{
    static void Main ()
    {
    int [,] arr = new int [4,5];
    int a = 5;
    // Code for assinging values to 2D Array
    for (int i=0; i< arr.GetLength(0); i++)
    {
      for (int j= 0; j< arr.GetLength(1); j++)
      {
        arr [i,j] = a;
        a += 5;
      }
    }
```

```
// Code for printing values of a 2D array

for (int i=0; i<arr.Getlength(0); i++)
{
    for (int j=0; j<arr.Getlength(1); j++)
        Console.Write (arr[i,j] + " ");
    Console.Writeline ();
}
}
```

Note :- Get length Method can give both rows and Colums that are present in a two Dim-array, when we pass (0) it returns no. of row and when we pass (1) it returns no. of columns.

(Arrays of Array) -

Jagged Array :- These are arrays, which are also going to be 2D but the number of Colums to each row will be different.

In a 2D Array all rows will have the same number of colums but in a Jagged Array the colum size varies from row to row. These are known as Arrays of Array, because it is a combination of multiple Single Dimensional arrays.

20/02/12

Syntax :-

<type> [ ][ ] <name> = new <type> [rows][ ];

int [ ][ ] arr = new int [3][ ];

or

int [ ][ ] arr = { list of values };

Note- While declaring a jagged in it's initial declaration we can only specify the number of rows in the array, but not the colums.
After specifying the num. of rows, now pointing to each row you must specify colums for that row.

Ex. int[ ][ ] arr = new int [4][ ];
arr[0] = new int [5];
arr[1] = new int [6];
arr[2] = new int [8];
arr[3] = new int [4];

arr [row][column]

```
Using System;

Class JArray
{
  Static void Main()
  {
    int[][] arr = new int[4][];
    arr[0] = new int[5];
    arr[1] = new int[6];
    arr[2] = new int[8];
    arr[3] = new int[4];

// Printing values of a Jagged Array:

    for (int i=0; i<arr.GetLength(0); i++)
    {
    for (int j=0; j<arr[i].length; j++)
      Console.Write(arr[i][j] + " ");
      Console.Writeline();
    }

// Assigning values to a Jagged Array:
    for (int i=0; i<arr.GetLength(0); i++)
      {
      for (int j=0; j<arr[i].length; j++)
        arr[i][j] = j+2;
      }

// Printing values of a jagged Array:

    foreach (int i=0; i<arr.GetLength(0); i++)
    {
        foreach (int x in arr[i])
          Console.Write(x + " ");
    }}  Console.Writeline();
```

Assigning values to 2D Array at time of declaration

```
int [,] arr = {
                { 11,12,13,14 },
                { 21,22,23,24 },
                { 31,32,33,34 }
              };
```

Assigning values to Jagged array at the time of declaration;

```
int[][] arr = {
                new int[3] {11,12,13}
                new int[5] {21,22,23,24,25}
                new int[4] {31,32,33,34}
              };
```

Command Line parameters :- It is an option using which we can supplied input to a program from the command prompt at the time of program execution, which allows us to give any number of values as well as any type of values as an input. where all these values are captured under the program, with in the string array of main method.

```
Using System;

class Params
{
static void Main (string[] args)
  {
    foreach (string str in args)
      Console.WriteLine (str);
  }
}
```

after compiling the program you can run the program from Command Promt as following -

```
C:\Csharp11 > Params 100 Hello False 3.14
```

```
Using System;

class AddParams
  {
    Static Void Main (String[] args)
    {
      int sum = 0;
      foreach (string str in args)
```

24/02/12

Working With Visual Studio .NET :-

- It's an IDE used for development .net app's with any .net lang like C#, VB etc., as well as we can develop any kind of App like Console, windows, Web etc.

Version of VS.Net -

VS	(Framework 1.0)
VS 2003	(" 1.1)
VS 2005	(" 2.0)
VS 2008	(" 3.5)
VS 2010	(" 4.0)

→ To open VS go to start Menu → Programs → MS Visual Studio → MS visual studio and click on it to open.

→ App. development under VS are known as projects, where each project is collection of various files or Item. To create a project either click on "New Projects" option or go to File Menu and select New → Projects which opens "New project" etc.

→ Under New project window we need to specify the following details -

1. In the LHS choose the language in which we want to develop the project (VB/C#)

2. In the middle choose the Of App. we want to develop by selecting a project template.
Ex: Console App.

3- In the bottom specify to name the project. Ex. First project

4. Below project name specified location where to save the project.
Ex C:\Csharp\

5- Click on ok button which create the project with a default class program - Program.cs.

6. When project is developed in VS by default a Namespace get created with the same name of the project, that is demo.project. in our case, from now each and every class of the project comes within the same namespace only.

Note: As discussed earlier a Namespace is logical container of type.

7. Now under main method of Program class write the following code:
Console.WaiteLine ("My First Project");
Console.ReadLine();

B- To Run the class either press F5 or Ctrl + F5 or click on start debugging button on top of the studio which will save, compile and execute the program.

Adding New @items in the Project :-

→ Under VS we find a window in the RHS known as Solution Explorer used for organizing the complete app, which allow us to view, add and delete items under the projects.

Note :- If solution Explorer is not avilable on RHS go to view Menu and select solution Explorer.

→ To Add a new proj. class under project open solution explorer right click on the project → select Add→new item, which opens add new item window → select class template init → specify a name in the bottom and click on Add Button, which adds the class under project. Ex: class1.cs

Note - The New class added also comes under the same Namespace FirstProject.

- Now under the New class write following Code:

```
static void Main()
{
Console.WriteLine("Second Class");
Console.ReadLine();
}
```

→ To Run the above class open Solution Explorer right click on the project → select properties → which open project property window, under it we find an option startup object which lists all the classes of project that containg valid Main method in them, choose your class and Run.

Object Oriented Programming :-

- → Encapsulation - hiding data
- → Abastraction - hiding complexcity
- → Inheritance - Reusebility
- → Polymorphism - Output according to input

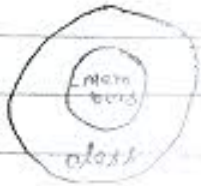
It's a new approch that came in to picture in 70's to resolve the drowbacks of procedural programing languege like security and reusability.

Object Oriented lang's are mainly designed to provide security and reuseability to the code. To call a language as object oriented it needs to stisfy a set of poincipal that are prescribed like. (i) Encapsulation
(ii) Abastraction
(iii) Inhesitence
(iv) Polymorphism

(i) Encapsulation - This is all about hiding of the code to provide security by wrapping the code under a container known as a class.

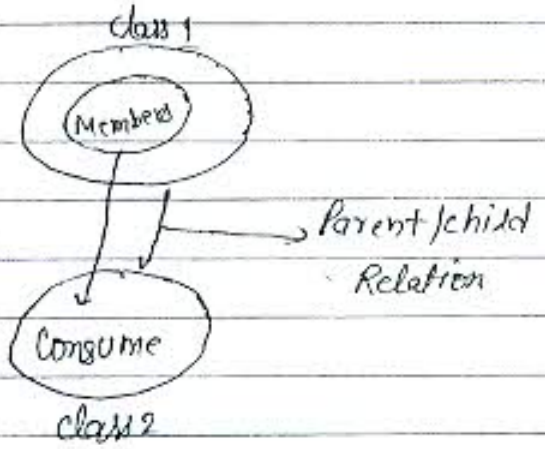
Procedural Program

OOP



(ii) Abastraction - This is all about hiding the complexity of code and providing with a set of interfaces to consume the functionalities where a fun or a Method never exposes the content of it but only performs a appropriate action when it is called.

(iii) Inheritence - According to this the property of parents belongs to children, applying the same principle in our language the members that are defined in a class can be consumed from other classes by establishing relationship between (Parent and child) the classes. Which provide the reuseablity of the code.



(iv) Polymorphism :- Changing the behaviour of entities according to the input that is supplied to them is known as polymorphism that is whenever the input changes automatically the output or behaviour also changes accordingly.

Sub-Programs:- A sub program is a named block of code which can be reused that is it gets executed when and where it is called. These Sub program are called with different name, and diff. language and approches like functions in lang's like C and CPP, methods in Java and .NET stored Procedure (SP) in databases.

Method:- A method is an action that has to be performed.

A method can be either a returning a value or non value returning also. where value returning methods performs an action and then return the status of the action. where as a non value returning method just performs an action. But will not return any result.

Syntax:

makes a method dynamic

```
[<modifiers>] void/Type <Name> ([<parame.defs>])
{
  - stmt(s)
}
```

Modifiers or some special key words that can be used on a method like public, private, static, virtual, abastract, ~~overwrite~~ override etc.

As we discussed a method can be either value returning or non value returning to define a non value returning method, we use void and to define a value returning method we specify the type of value it is going to return

We can make object of a class in any static method.

Note - The type of value a method return need not be a predefined type like, int, float, string etc. they can also be user defined type also.

→ If required we can pass parameters to a method to make them dynamic so to pass them we adopt the below process.

Syntax:

[ref/out] <type> <var> [= value] [,...n]

adding ov

→ As per the rule of encapsulation we defined methods under a class.

→ The methods what ever we define in the class if they are not static should be called if and all

We create object of a class as following -

<class> <obj> = new <class>([<list of values>])

program p = new program(); // P is a object

or

Program P; // P is a variable

P = new Program(); // P is a object

→ The object of a class can be created under any static block of a class as well as it can also be created in other classes also. When we want to create an object of a class in it self only that can be done from any static block but we use

Main() for creating the object, because it is an entry point of the program or class.

→ Open VS select new project choose the language as Visual C#, choose the project template as Console application name the project as OOPS project. and save it our folder.

→ Now under the default class program of the Object write the following code.

```
class Program
{
//Method without any Input or Return type
    Public void Test1()        //fixed Behaviour
    {
    int x = 5;
    for(int i=1 ; i<=10 ; i++)
    Console.Writeline ("{0} * {1} = {2}", x, i, x*i);
    }

// method which has input but no return type
public void Test2(int x, int n) // Dynamic Behaviour
  {
    for (int i=1; i<=n ; i++)
    Console.Writeline ("{0} * {1} = {2}", x, i, x*i);
    }

// method without any input but has return type
    Public string Test3()      //fixed Behaviour
    {
      string str = "Hello world";
```

```
        str = str.ToUpper();
        return str;
    }

    // Method with both input and return type also
    Public String Test4(string str) // Dynamic Behaviour
    {
        str = str.ToUpper();
        return str;
    }

    static void Main(string[] args)
    {
        Program P = new Program();

    // Calling non value returning methods:
        P.Test1();
        P.Test2(6,12);

    // Calling value returning methods:

        String str = P.Test3();
        Console.WriteLine(str);
    Console.WriteLine(P.Test4("hello how r you"));
        Console.ReadLine();
    }
}
```

5/12/12

Parameters :- These are used for a method, making them dynamic, there of two types -

(i) Input parameters
(ii) Output parameters

Input Parameters used for bringing a value into the method for execution of the method. Where as output parameters are used for carring a value out of the method after the execution

We can send a value out of a method after execution in two diff- ways.

(i) using return types -
(ii) using output parameters -

A Return type can carry only a single result out of the method, where as output parameters are capable Carring multiple value out of a method.

To Declare a parameter as output, we need to use ref or out Keyword on the parameter

```
public void Math1 ( int a, int b, ref int c, ref int d)
{
    c = a+b;
    d = a*b;
}

public void Math2 ( int a, int b, ref int c, ref int d)
{
    c = a-b;
    d = a/b;
}
```

ref int c → int *c

→ Pass by value — Input
→ Pass by reference [implicit Pointer] — Output
↳ garbage collector is responsible for that.

```
class Params
{
// Methods with Input and output parameters
  public void Math1 (int a, int b, ref int c, ref int d)
    {
      c = a+b;
      d = a * b;
    }

public void Math2 ( int a int b, out int c, out int d)
  {
      c = a - b;
      c = a / b;
  }

//Method with default values to parameters
    public void AddNums (int x, int y=50, int z=25)
    {
      Console.WriteLine (x+y+z);
    }

static void Main ()
{
    Params P = new Params ();

// Calling method with default value to
   parameters
```

```
p.AddNums (100);
p.AddNums (100,100);
p.AddNums (100,2:100);
p.AddNums (100,100,100);

// Calling methods with output parameters:
int x=0, y=0; // Initilization is Mandatory
p.Math1 (100, 50, ref x, ref y);
Console.WriteLine (x + " " + y);

int m,n;          // Initilization is optional
p.Math1 (100,50, out m, out n)
Console.WriteLine (m + " " + n);
Console.ReadLine ();
}
}
```

If A method is declared with a parameter using ref/out keyword, we called it as output parameter. If the parameter is an output para, it is going to store the addresses of variable that are used to calling the method but not the values.

In the case of our method the execution is starts as following.

```
void Math1 (int a, int b, ref int c, ref int d
{
c = a+b;
c = a*b;
}
```

(100) (50) (&m) (&n)

```
int m=0, n=0;
p.Math1 (100, 50
```

'In the Method c and d are reference parameters that is pointer variables in our "traditional language."' which can store the address of a variable so they are holding the address of m and n but not the values of m and n.

→ Once the Execution of method start the sum and product of variables a, b is assigned to the reference parameters c and d. which will internally redirect the values to the variable they are point too. So the result will be as following after execution.

```
int m=0;   (150) m   (1500) n
int n=0;
p.Maths (100, 50, ref m, ref n)
```

→ An Output parameter can be declared either by using ref / out keyword. but however they are declared they are execute in the same way the only difference is the variable being substituted in the place of ref should be initilized and optional in case of out to be initilized.

→ Passing default value to parameters (addNums method) is a new feature that has been added in C# 4.0 V. So if default value is given to any parameter, passing

24/02/12

value to the parameter while calling the method is only optional.

- Add a class First.cs and write the following code

```
class First
{
    int x = 100;
    static void Main()
    {
        first f;        // f is a variable
        f = new first(); // f is a object
        Console.Writeline(f.x);
        Console.Readline();
    }
}
```

→ Every Member of a class if it is nonstatic can be accessed only by using object of the class in which it defined.

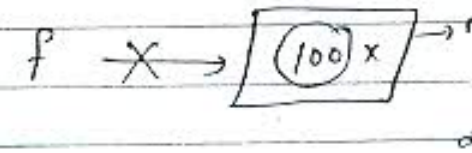
→ Object of the class can be created only with the help of new keyword that is if new is used then only the memory required for the object gets allocated.

First f; — [null]

f = new First(); — [0x]

→ If NULL is assigned to any object that is created the reference of this object, with the memory is removed and the memory get marked as unused, which can be deallocated by garbage collector at any particular point of time. But the memory once marked as unused can not be used any more so you can not use the member using object, to test this rewrite the code, under the main method of class First as following.

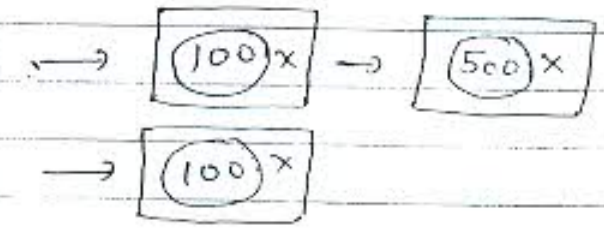
Ex:-
```
static void Main()
{
    First f = new first();
    Console.WriteLine (f.x);
    f = null;
    Console.WriteLine (f.x); // invalid Error
    Console.ReadLine();
}
```



→ We can create any Number of object to a class where each object we create will have it's own memory allocation storing it's members any changes that are performed on members of an object, will never be reflected to member of other object maintaining the security. To test this rewrite the code, under the main method

of Class First as following -

Ex. static void Main()
{
first F1 = new first();
first F2 = new first();
Console.WriteLine (f1.x + " " + f2.x);
f1.x = 500;
Console.WriteLine (f1.x + " " + f2.x);
Console.ReadLine();
}



→ If required the object of a class can assigned to the variable of some class. So that the variable now treated as a reference. where references of a class will not have any memory allocation. They will be consuming the memory of object that is assigned to it. So if at all any changes are made on the members through the object, those changes get reflected to reference also and vice versa. To test this we write the code under main method of class first as following -

Ex. static void Main()
{ first f1 = new first();
first f2; // is a variable
f2 = f1; // is a reference

```
Console.Writeline (f1.x + " " + f2.x);
f1.x = 500;
Console.Writeline (f1.x + " " + f2.x);
Console.Readline ();
}
```

Result :- 100, 100
500, 500

A copy of a class which is not initilized is known as a variable of the class. where as a copy of the class, which is initilized through new keyword is known as object of a class. And a copy of a class which is initilized through another object of the class is known as reference of the class.

→ In the above case we have been talking an object of the class and the references of the class will consume the same memory, in such cases if null is assigned to any one of them still other can consume or access the memory but for whom null is assinged the memory is not accessable and we can not access the member using that.

To test this rewrite the code on the main method of class first as following.

- link method
- blue variable
- white constant

Ex- Static Void Main ()
{
First f1 = new first ();
first f2 = f1;
f1 = null;
Console.WriteLine (f2.x); // valid
Console.WriteLine (f1.x); // Invalid (Error)
Console.ReadLine();
}

When an object of a class is created internally the following things takes place.

(i) Reads the classes to identify the member.

(ii) It invokes the constructors of the classes.

(iii) Allocates the memory that is required.

Constructor – It is a special method that is present in each and every class responsible for initilizing the variables of the class.

→ The Name of a Constructor method will be same as your class name and it is void value returing.

→ without a constructor defined in a class a class can not execute so it is mandatory a class should contain a constructor in it.

Note- If we don't define a constructor in a class explicitly while compilation of the class, an implicit constructor is defined in the class. This is how all our previous classes got executed even if we did not define a constructor in them.

If we defind a class as following -

```
class Test
{
  int x; string s; bool b;
}
```

after compilation of the class you will be finding it as following - with Implicit Constructor initilizing the value of variable with default value.

```
class Test
{
  int x; string s; bool b;
  public Test()
  {
    x=0; s=null; b=false;
  }
}
```

25/02/12

Note – Implicitly defined constructor will be always public.

Syntax for Constructor :-

[<modifiers>] <Name> ([<Param def's>])
{ stmts; }

We can also defined constructor in our class as above.

Add class condemo.cs and write the following code.

```
class ConDemo
{
    Public ConDemo()
    {
    Console.WriteLine("Constructor Called");
    }
    public void Demo()
    {
    Console.Writeline("Method called");
    }
static void Main()
{
    ConDemo cd1 = new ConDemo();
    ConDemo cd2 = new ConDemo();
    ConDemo cd3 = cd2;
cd1.Demo(); cd2.Demo(); cd3.Demo();
  Console.Readline();
}
}
```

If a method has to execute it should be explicitly called, in the same way if a constructor has to execute it must be called explicitly. We call a constructor while creatin object of a class as following.

```
ConDemo cd2 = new ConDemo();
```

calling the constructor explicitly.

Constructor are of two types.

(i) Zero Argument constructor
(ii) Parameterized Constructor.

A Cons. without any parameters is known as zero argument Cons., these consts. can be defined in a class either by a programmer explicitly or will be defind implicitly provided there is no constructor present in the class.

A Cons. with parameters is know as parameterized constructor, these can be defined only explicitly by programmers but can not be defined implicitly. If Cons. are parameterized in a class we need to send values to the parameters while creating object of the class.

Add a class Conparam.cs and write the following-

```
class Conparam
{
    public Conparam(x)
    {
        Console.WriteLine("Parametrized Constructer" + "x");
    }
    Static void Main()
    {
        Conparam. CP1 = new Conparam(10);
        Conparam. CP2 = new Conparam(20);
        Console.ReadLine();
    }
}
```

What is the need of a Constructor – or why do we define a constructor explicitly.

As we are aware that parameters make a method dynamic in the same way parameters to a constructor will make the completer class dynamic that is every class requires some initialization or value to execute the code to initialize

After calling the constructor variable initialization and method invocation gets performed

Add a class Maths.cs and write the following -

```
class Maths
{
    int x, y;
    public Maths (int x, int y)
{
    this.x = x;
    this.y = y;
}
    public void Add()
{
Console.Writeline (x+y);
}
    public void Sub()
    {
Console.Writeline (x-4);
    }
    Public void Mul ()
    {
      Console.Writeline (x*y);
    }
    Public void div ()
      {
      Console.Writeline (x/y);
      }
}
```

To Consume the above class add a new class in the project as TestMaths. And write the

llowing -

following code in it -

```
Class TestMaths
{
    static void Main()
    {
    Maths obj = new Maths (100,50);
    obj.Add();
    obj.Sub();
    obj.Mul();
    obj.Div();
     Console.ReadLine();
    }
```

Using parameterized Constructor we can initilized variables of a class with user defined value and make them dynamic where as zero argument Constructor will initilize variables of a class with default values so we refer them as default Constructor also.

Sav
ite the

27/02/12

Working of ATM Machines :-

Assume the below class has been defined for performing the transaction under in ATM Machine.

```
public class customer
{
   readonly int CustId;
    double bal;
   static int count = 0;
   public Customer ( int custid )
   {
     count += 1;
     this. custid = custid;
this.bal = < load bal from db table for given id >;
   }
public double FindBalance ()
 {
   return bal;
 }
public void Deposit ( double amt )
 {
     bal + = amt;
- Update bal back to db table for given id
 }
 public void Withdraw ( double amt )
 {
     bal - = amt;
- update bal back to db table for given id
 }
}
```

7/02/12

Assume the data of customer was stored in a DB table as following.

Customer

Cus_Id	Cus_Name	Balance
101	AAA	5000
102	BBC	10000
103	CCC	15600

Now whenever a customer tries to access the m/c with his card, the card reader on the m/c first reads the Cus.Id on the Card and making use of that Id an object of the class Customer gets created on the server so if at all multiple customer accessing the system at same time from diff m/c for each customer a seprate obj. gets created on the Server. Loading the data of that Customer. As following -

Machine 1:

```
Customer C1 = new Customer (Cus_Id);
                                  101
```

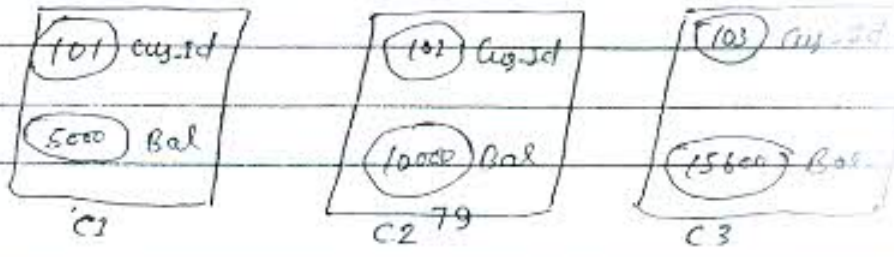
Machine 2:

```
Customer C2 = new Customer (102);
```

Machine 3;

```
Customer C3 = new Customer (103);
```

The objects are maintained on Server as following -



Here each object created on the server is associated with the diff Customer so using that object Customers can perform Transaction like Deposit, WithDraw etc.

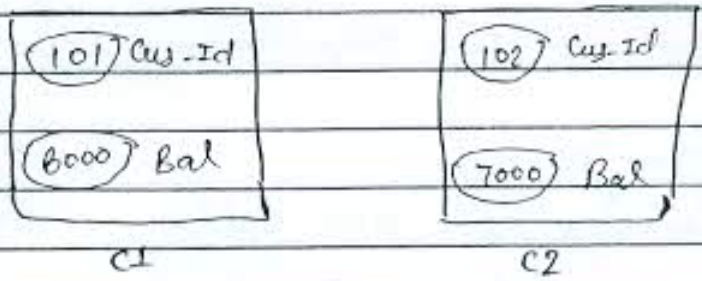
Machine 1:

c1.Deposit(3000);

Machine 2;

c2.WithDraw(3000);

As we are aware that in object Oriented programming the data of one object is never shared with the data of other object so transaction performed on one Customer object does not reflect to other Customer object so changes can perform on his object data only. As following



Static Methods Vs Instance Methods –

→ A method that is declared explicitly using static modifier is a static method rest of all are Instance only.

→ All the rule and regulation that we have discuss in case of variables applying these in methods also.

While defining methods under a class if the method is instance or non static we can directly consume in static members here. Where as if the method is static we can consume non static members here only with the help of class object.

Note - We can never consume non static member of a class from static block directly, we need to consume them. Compulsury with the obj. of class only.

```
class statmets
{
  int x = 100;
  static int y = 200;
  static void Main Add ()
{
  statmets obj = new statMets ();
  Console. WriteLine (obj.x + y);
}
  static void Main ()
  {
    statMets. Add ();
    Console. Readline ();
  }
}
```

Static Constructors Vs. Instance Constructors -

→ A Cons. that is explicitly declared using static Modifiers is a static constructor ones of all are Instance Constructor Only.

→ A static constructor is a first block of code which executes under a class and is executes one and only one time in the life cycle of a class.

→ Instance Cons. are called only when the obj. of class is created as well as each time the obj. is created.

→ A static Constructor can never have parameters because they are called implicitly so sending values to them will not be possible. whereas Instance Cons. can be parameterized as we are explicitly calling them.

```
class statCon ()
  {
    static statCon ()
   {
    Console.WriteLine ("static Constructor");
   }
    statCon ()
   {
   Console.WriteLine ("Instance Constructor");
  }
```

```
        static void Main()
        {
            Console.WriteLine("Main Method");

            StatCon S1 = new StatCon();
            StatCon S2 = new StatCon();
            Console.ReadLine();
        }
    }
```

Static classes :- These are classes which can contain only static member in it because it can contain, only static member, it is not possible to create object of the class.

Inheritance :- It is a process of establishing relationship b/w classes so that the member of one class can be consume in other classes. In b/w the classes there are Parent child relationship.

Once Parent-child relationship is estb. b/w classes, Parent class members can be consumed by child class as if the members of of it's own. So Inheritance provides reuseablity of code.

Syntax:

[<modifiers>] class <classname child> : <PCName>

Ex- class class1
{
- members
}

class class2 : class1
{
- Consume parent's members
}

Note: Private Members of a class can never be accessed by child classes.

⇒ Add a class class1.cs and write the following-

```
Class class1
{
    Public class1()
    {
Console.Writeline ("class1. Constructor");
    }
    Public void Test1()
    {
Console.writeline ("Method one");
    }
Public void Test2()
{
Console.writeLine ("Method Two");
}
}
```

→ Add a class class2.cs and write the following -

```
class class2 : class1
{
  Public class2()
  {
  Console.Writeline ("Class 2 Constructor");
  }
public void Test3()
 {
Console.Writeline ("Method Three");
}
  static void Main()
  {
   class2 c = new class2();
   c.Test1();
   c.Test2();
   c.Test3();
   Console.Readline();
  }
}
```

Rules and regulations that has to be followed while working with Inheritance.

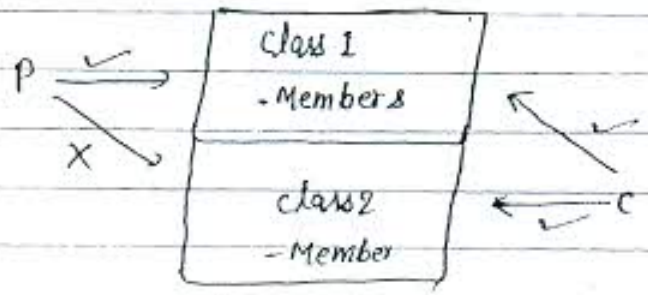
1. In Inheritance the execution of child class always starts by invoking the default Constructor of it's parent class so Constructor of the parent class must to accessible to the child, if not accessible Inheritance is not possible.

2- In Inheritance child classes can access parent class member but a parent class can never access child class member. You can checked this by the following child class main method.

```
class1 p = new class1();
p.Test1();
p.Test2();
p.Test3(); //invalid
```

3. Just like object of a class can be assigned to variable of the same class in the same way it can also be assigned to the variable of it's Parent class. To make it a reference. Where the reference will be consuming the memory of the object assined to it. But in this context also using parent's reference it's not possible to access child class members. To test this rewrite the code under the main method of child class class2 as following

```
class2 c = new class2();
class1 p = c;
p.Test1();
p.Test();
p.Test(); // invalid (It's not work)
Console.Readline();
```



Note → If required a parent class reference created using object of child. Can be converted back in to child reference again but not directly, we need to perform a explicit conversion.

Ex- Creating parent's reference using child object:

```
class 2 c = new class 2();
class 1 p = c;
```

Converting parent's reference back to child reference:

```
class2 obj = (class2) p;
```

or

```
class2 obj = p as class2;
```

4- Every class (predefined / user defined) has a default parent that is the class object defined under System Namespace. This class will be implicitly inherited for each and every class. So the members of this class like "equals, getType, gethashcode and tostring" can be accessed from any class.

To test this rewrite the code under child class Main Method as following-

```
object obj = new object();
Console.WriteLine(obj.GetType());
Class1 P = new Class1();
Console.WriteLine(P.GetType());
Class2 C = new Class2();
Console.WriteLine(C.GetType());
Console.ReadLine();
```

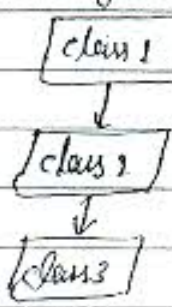
Types of Inheritance -:

This tells how many parent classes a child can have. As per the standard of OOP there are two diff. types of inheritance.

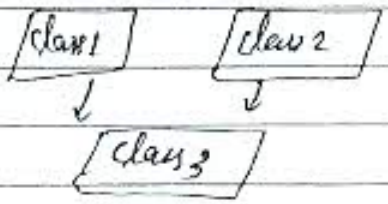
(i) Single Inheritance
(ii) Multiple Inheritance

If a class has one immediate parent class to it, we called as Single Inheritance whereas if a class has more than one immediate parent classes it, we called it as Multiple Inheritance.

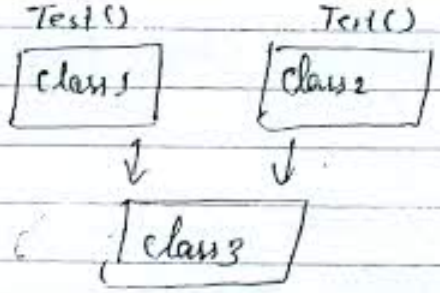
Single Inheritance



Multiple Inheritance



5. Under Java and .Net lang's Multiple Inheritance is not supported through classes what it support is only single inheritance through classes because multiple inheritance suffers from ambiguity problem. That is there can be a member with same name under multiple parents which becomes a confusion while invoking from child.



Note-: CPP lang support both single and multiple inheritance also through classes because it was the 1st OO lang's that came into picture so 'amiguity' problem was not anticipated while designing.

6. In the our 1st Rule we have discussed that execution of a child class always starts by invoking it's parent classes default Constructor implicitly, but in some situations parent classes may ~~not~~ contain parameterised Constructor in place of default so in that Senario implicit calling of the Constructor from child class will not be performed because the constructor requires a value for execution. So to resolve the problem it is the responsibility of developers to explicitly call the avilable parent class parameterised Constructor supplying the required value using

Base ↔ super

1/08/12

Base Keyword from child class constructor.

To check this do the following -
going to class class1 and rewrite it's Constructor as following.

```
public class1 (int x)
{
Console.WriteLine ("class 1 Constructor : " + x);
}
```

Now try to execute the child class class2 where you will getting an error under it's Cons. stating class1 doesn't contain a zero argu. Constr. to resolve the problem rewrite the constructor of class2 as following.

```
public class2 (int x) : Base (x)
{
Console.WriteLine ("class2 Constructor");
}
```

Now go to Main Method of class2 and set the value to the parameter x while creating the obj as following.

```
Static void Main()
{
    class2 c = new class2 (100);
    Console.ReadLine();
}
```

In the above case when we create the obj of child class class2, the parameter value first goes to it's constructor and from there transferred to parent class class1 Constructor.

Note - Using 'BASE' Keyword from child class we can access parent class member.

Consuming a class for other classes -

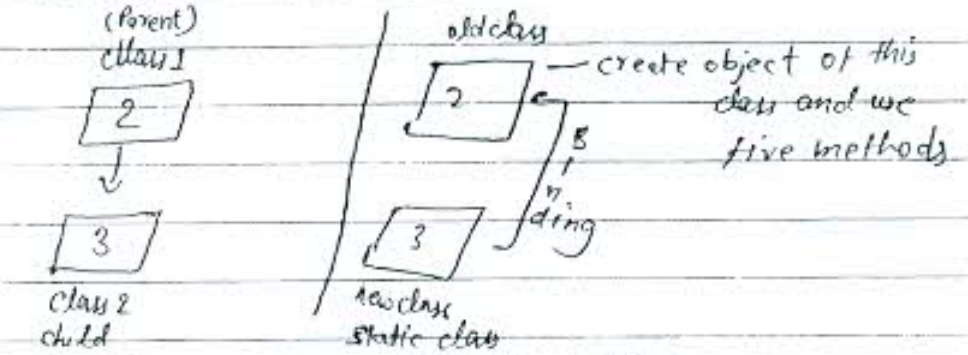
We can consume a class from other classes using two different approches.

(i) With help of Inheritance - as we have seen in class1 and class2 above

(ii) By Creating an object of a class also under another class we can consume the members of class to test this add a new class class3.cs and write the following.

```
class class3
{
    static void Main()
    {
        class1 obj = new class1(20);
        obj.Test1();
        obj.Test2();
        Console.ReadLine();
    }
}
```

Extension Methods:



Extension Method - It is a new feature that has been added in C# 3.0, which allows to add your methods in a class without Editing the source code of the class.

→ We can extend the functionalities of existing class using two different approches-

(i) Inheritance

(ii) Extension Methods

In the 1st Case to add new members in an existing class we can define a child class for it and then add newer methods in the child class and consume the old and new members using object of child class. in which

In the 2nd approch 1st define the methods we want to add to an existing class in a new class which must be static and then bind those methods to the required class the speciality in this approch is you can consume both old and newer methods using the object of old class.

defining Extension Method.

Extension method has to be defined under a static class as per the rule.

→ Because we have defining the method under a static class must and should you need to define them static method but once this method are bound with a class, they will be implicitly converted into non static methods under the class to which they are bound.

→ The first parameter of an extension method is refferred as a binding parameter, where this parameter will not be taken into consideration while invoking the method, A binding parameter is only specify to which class the method has to be added. Binding parameter should be prefixed with 'this' Keyword.

→ We can pass any formal parameter to an extension method beside the binding parameter. These parameter should be taken into consideration while invoking the parameter except binding parameter so if an extension method is defind with n parameter while invoking we will have only n-1 parameters.

→ An Extension Method can have only one binding parameter that is it can be associated with a single class.

Add a class Original.cs and write the following -

```
class Original
{
  public int x = 50;
  public void show1()
  {
  Console.Writeline ("first Method :" + this.x);
  }
    public void show2()
    {
     Console.Writeline ("second Method:");
    }
}
```

Add a class static.cs and write the following -

```
static class Statclass
  {
  public static void show3 (this Original o)
    {
      Console.Writeline ("3rd Method");
    }
public static void show4 (this Original o, int x)
    {
    Console.writeline ("4th method");
    }
Public static void show5 (this Original o)
    {
Console.WriteLine ("5th Method :" + o.x);
    }
}
```

Add a class TestOriginal.cs and write the following:-

```
class TestOriginal
{
    static void Main()
    {
     Original obj = new Original();
    obj.show1();
    obj.show2();
    obj.show3();
    obj.show4();
    obj.show5();
     Console.ReadLine();
     }
}
```

Polymorphism :- Behaving in diff. ways depending upon the input received is known as polymorphism. That is whenever the input changes automatically output and behaviour also changes.

Polymorphism can be implimented in a program using three diff. approches -

(i) Overloading
(ii) Overriding
(iii) Hiding

Overloading is again of three types -

(a) Method Overloading
(b) Constructor Overloading
(c) Operator Overloading

(c) Method Overloading - It's an approch which allows to define multiple methods in a class with the same name changing their Signature. Changing the sign. sense we can change the number of parameter & being passed or type of parameters being passed in to the method or order of parameter passed in to the method.

Ex.
```
Public void show()
Public void show (int x)
public void show (string x)
public void show (string x, int y)
public void show (int y, string s)

public string show ()   // Invalid
```

Note - A change in the return type of a method will not be taken into consideration in case of- overloading method. What it considered only parameter.

Add a class LoadDemo.cs and write following in it -

```
class LoadDemo
{
    Public void show()
    {
    Console.WriteLine();
    }
Public void show (int x)
    {
    Console.WriteLine (2);
    }
```

```
Public void show (string s)
{
 Console.WriteLine (s);
}
public void show ( string s, int x)
 {
  Console.WriteLine (4);
 }
public void show ( int x, string s)
 {
  Console.WriteLine (s);
 }

static void Main ()
{
    LoadDemo d = new LoadDemo();
 d.show();
 d.show(10);
 d.show("Vasu");
 d.show("Vasu", 10);
 d.show(10, "Vasu");
    Console.ReadLine ();
}
}
```

Note - Overloading is a feature that allows to define Multiple behaviours to a method.

Ex: Console.WriteLine () method have a 19 methods in it. That is overloads and print any type of value that is supplied as a input so programers are force to print anytype of value without knowing what type of value they are printing.

11.16- Constructor Overloading :- Just like we can overload methods under a class, we can also overload constructor of a class also.

→ If a class contains multiple Cons. in it, object of that class can be created using any constructor avilable.

→ Generally parameterised Cons. will be defined to initilize variables of a class dynamically with the values that are sent while creating object of the class where as zero argument constructors are used for initilizing variables of a class with default values so that if object of the class is created using this constructor variable get initilized with those default values, that is why zero argument constructors are also known as default constructors.

Add a class LoadCon.cs and write the following -

```
class LoadCon
{
    int x;
    public LoadCon()
    {
        this.x=10;
    }
    public LoadCon(int x)
    {
        this.x=x;
    }
```

```
    public void display()
    {
      Console.WriteLine("The value of x is :{0}", x);
    }

    static void Main()
    {
      LoadCon c1 = new LoadCon();
      LoadCon c2 = new LoadCon(20);
      c1.Display();
      c2.Display();
      Console.ReadLine();
    }
}
```

In the above class we have both default and parameterised Constructor also so object of the class can be created by using any constructor. If the object is created using default Constructor, the variable x is initilized with a default value that is 10. Whereas if object is created using parameterised Cons. we can send our own value for the variable x to initilize.

(i) Inheritance based overloading :-

If at all parent classes method is overloaded under child class we call it as inheritance based overloading.

Ex. class 1
public void Test()

class 2 : class 1
public void Test(int x)

(ii) Method overriding — If at all a parent classes method is reimplimented under the child class exactly with the same signature we call it as method overriding.

Note

diff. b/w Overloading and Overriding

Overloading	Overriding
1. It is implimenting multiple method with same name and diff. signature.	1- It is implimenting multiple method with the same name and signature.
2. This can be performed in one single class as parent/child classes also.	2- This can be performed only in parent/child classes.
3. To overload a parent class method under child class we do not any permission from parent.	3- To override a parent class method under child class 1st we required an explicit permision from parent

4. This is all about providing multiple behaviour to a method.

4 - This is all about changing the behaviour of a method.

How can we override parent classes method under child class - To override a parent class method under child class, 1st in the parent class the method must be declared as virtual then only child classes can override the method. Using override modifier.

Ex- class 1.
public virtual void show()

class 2 : class1
public override void show()

Note - Overriding virtual method of a parent class under child class is only optional for child classes.

Add a class loadParent.cs and write the following code -

```
class loadParent
{
  Public void Test()
{
Console.writeline ("Parent's Test Method:");
}
Public virtual void show()
{
 Console.Writeline ("Parent's show method:");
}
```

```
public void Display()
{
Console.WriteLine("Parent's Display Method");
}
}
```

Add a class Load childcs and write the following -

```
class Loadchild : Load Parent
{
// overloading Test method under child class

  public void Test (int x)
{
Console.WriteLine("child class Test Method");
}

Static void Main()
{
  LoadChild c = new Loadchild();
    c.Test();
    c.Test(10);
   c.show();
   c.Display();
   Console.ReadLine();
}
}
```

In the above class we are overloading parent classes Test() in child class then then calling both Parent and child class Test() using childclass object.

→ In the above case we did not override the virtual show method of parent class, which proves overriding is only optional for child.

→ Now if you want to override the virtual show method under the child class add a new method in the child class as following.

ex- public override void show()
{
Console.Writeline("child show method");
}

Now execute the child class again and see the o/p of show method where in this case a place of the parent show method child classes show method gets executed because we changing the behaviour in this context under the child class where the child class object gives preference to local methods.

Once after overriding parent classes Virtual Method under child class, child classes object can not call parents virtual method anymore because, it gives preference to it's own overriding method but if still we want to consume parent classes virtual method from a child classes, we have two options -

(i) By creating parent classes object under child class we can call parent classes virtual method from child. To test this rewrite the code under child classes main method as following.

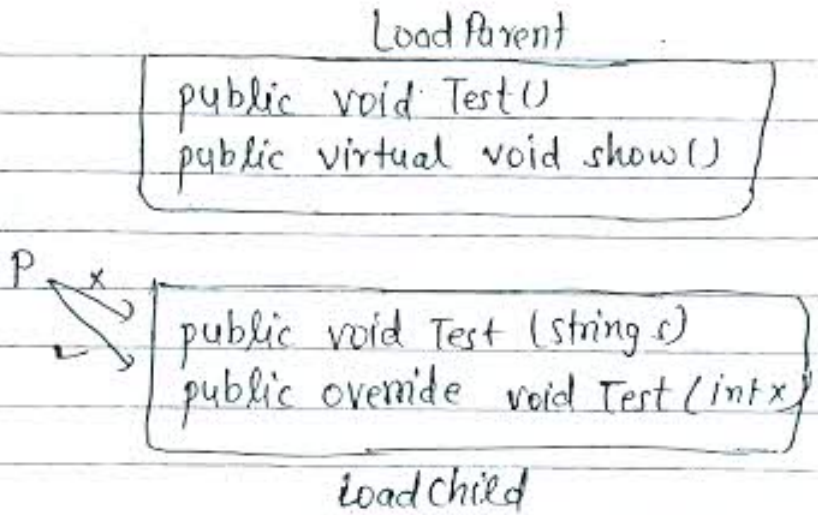
```
static void Main()
{
    LoadParent P = new LoadParent();
    LoadChild C = new LoadChild();

    P.show(); // Invoking Parent's method
    C.show(); // Invokes child's method
    Console.ReadLine();
}
```

Note:- Earlier we have In the 3rd Rule of Inheritance we have discussed that parent references that are created using child objects can not access any members of child class but we have an assumption for this rule in overriding that is using the reference we can call child classes overridden members because overriding is performed here with parent classes permission only.

To test this rewrite the code under child classes main method as following-

```
static void Main()
{
    LoadChild C = new LoadChild();
    LoadParent P = C;
    P.show(); // Invokes child's method only
    C.show(); // Invokes child's method
    Console.ReadLine();
}
```



(ii) Using Base Keyword also from ~~parent~~ child class we can access parent classes virtual methods after overriding But using Keywork like this, base is not possible from static blocks.

To test this add a new method under child class as following -

```
// Invokes parent classes virtual show method
public void Pshow()
{
    base.show();
}
```

Now under the Main method using child class object you can call either Parent's method or child clouses Methods -

```
static void Main()
{   LoadChild c = new LoadChild();
    c.Pshow(); // Invokes parent's method
    c.show(); // Invokes child's method
    Console.ReadLine();
}
```

5/05/12

Method hiding :- This is another approch using which parent classes method can be reimplimented under child class.

→ We can reimpliment parent classes method under child class using two different approches.

(i) Method overriding

(ii) Method Hinding / shadowing

In the first case reimplimentation of method is possible only when the method is declared as Virtual in the parent class. So that child classes can reimpliment that method using override modifier.

In the second case evenif the method is not declared as virtual also reimplimenting the method under child class is possible.

Ex. LoadParent

```
public void Display()

LoadChild : LoadParent
public new void Display()        // Hiding
```

Note. Using new keyword on the method while reimplimentation is only optional. Or else you get a warning message stating the child class methods hides parent classes Method.

All the rules we have discussed in case of method overriding applies to method hinding also exceptone in overriding we have discussed that Parent class reference created using object of child class, can invoke overridden method of child class because reimplimentation is performed in child class with parent classes permission, whereas it is not possible in hiding because reimplimentation is not performed here with parent classes permission.

To test this add new method in the child class LoadChild as following -

```
public new void display()
{
Console.WriteLine("child's Display Method");
```

Now in the main method of child class Rewrite the code as following -

```
LoadChild c = new LoadChild();
LoadParent p = c;
p.show();    // Invokes child class method
p.Display(); // Invokes parent class method
Console.Readline();
```

Ex. Loadparent

```
public void Test()
public virtual void show()
public void Display()

Loadchild : loadparent
public void Test (int x)        // overloading
public override void show()     // overriding
public new void display()       // Hiding / Shadowing
```

Polymorphism is of two types –

(i) Compile Time or Static or Early Binding
(ii) RUN Time or Dynamic or Late Binding

In the 1st case at the time of program compilation the object of class knows which polymorphic method it has to invoke and gets the method call bound with the method. All this happens only in case of Method overloading because here each Method will have different signature also known as early binding because binding is done with method call and method signature at compilation time only.

In the 2nd case object of the class recognizes which polymorphic method, it has to call in Runtime and gets bound with that method because this is happning at runtime we call it as late binding. Method overriding and Method

hiding comes under this catagory. Because here multiple methods will have same signature a find in parent and child.

Sealed class and Sealed Method – A class which can not be Inherited by any other class is known as Sealed class. To make a class as sealed we must explicitly declared it using sealed modifier. String is a sealed class defined in system Namespace so inheriting of this class is never possible. But a sealed class can still consumed from other classes by creating it's object.

Ex- Sealed class class1

```
{
}
```

class class2 : class1 // Invalid

A method which can be overridden is known as sealed method, we can not override any method of a class under child class untill and unless it is declared using virtual modifier so by default every method is a sealed method.

If at All a method is declared as Virtual in in class any child class of it in linear hirarchy has a chance to overide the method.

Ex- class 1
public virtual void show()

class 2 : class1
public override void show()

class 3 : class2
public override void show()

Note - If class 2 does'nt override the method also class 3 can still override.

⇒ If Required a child class while overriding it's parent classes Virtual Method can seal the method. So that further overriding of the method is not possible under it's child class.

Ex- class 1
public virtual void show()

class 2 : class1
public sealed override void show()

Class 3 : class 2
public override void show() // Invalid

Operator Overloading :-

Just like we can define multiple behaviours to a method in the same way we can define multiple behaviours to an operator also. which is known as operator overloading.

'+' is an existing overloaded operator which can be used both for addition as well as Concatination of values also.

In the same way we can also ~~a~~define newer behaviour to an existing operator so that we can be used in different ways. To define your behaviour to an operator, we must define an operator method as following -

Syntax:

```
[<modifiers>] static <type> operator <opt> (<param defns>)
  {
    -stmts
  }
```

operator Method must be static. The name of an operator Method is 'operator' keyword only and beside that you need to specify the overator you want to overload.

Ex. operator + (<param def'n>)
operator - (")
operator * (")

6/03/12

Add a class Matrix.cs and write the following code:-

```
class Matrix
{
  // Attributes of a 2*2 Matrix
  int a,b,c,d;
  public Matrix (int a, int b, int c, int d)
    {
      Initializing the value of the Matrix
  this.a = a; this.b = b; this.c = c;
  this.d = d;
  }

//Overloading + operator to add values of
  Matrix's
public static Matrix operator +
      (Matrix m1, Matrix m2)
  {
Matrix obj = new Matrix (m1.a + m2.a,
   m1.b + m2.b, m1.c + m2.c, m1.d + m2.d);
  return obj;
  }
//Overloads - operator to subtract value of
    2 Matrix's.
public static Matrix operator -
          (Matrix m1, matrix m2)
  {
    Matrix obj = new Matrix (m1.a - m2.a,
  m1.b - m2.b, m1.c - m2.c, m1.d - m2.d);
  return obj;
}
```

```
//Overloading the Tostring method to print
values of matrix

public override string Tostring()
{
 return string.Format(
"[a:{0}, b:{1}, c:{2}, d:{3}]", a,b,c,d);
}}
```

Add a class TestMatrix.cs and write the following:

```
class TestMatrix
 {
   static void Main()
 {
Matrix M1 = new Matrix(5,6,7,8);
Matrix M2 = new Matrix(1,2,3,4);
Matrix M3 = m1+m2;
Matrix M4 = m1-m2;
 Console.Writeline(M3);
 Console. Write line(M2);
 Console.Writeline(M3);
 Console.Writeline(m4);
 Cnsole.Readline();
  }
}
```

→ Whenever we print the object of a class implicitly using Write/Writeline methods implicitly ToString() method of class gets called (Inherited from Object class). That identifies the name of class to which the object belongs and returns it.

→ As the ToString() is a Virtual method, we can override the method according to our requirement and change the behaviour of that method.

→ In the above context, we are overriding the ToString() method under the Matrix class and returning values of the Matrix in the place of ClassName.

→ Substituting variables under string can be done either under Write/Writeline() methods of Console class and format method of string class. The main diff. is the 1st two prints the value whereas Format returns the values.

Abstract classes and Abstract Method :-

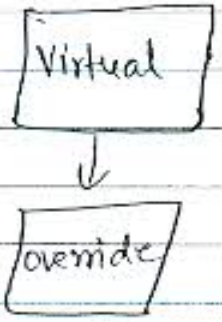
→ A method without any method body is known as 'Abastract Method' what it contains is only declaration of the method. To declare a method as abastract we need to use abastract Modifier.

→ The class under which we declare abastract methods is referred as an 'Abastract class'.

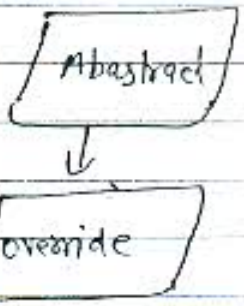
An abastract class also should be declared using abastract modifier.

Ex. abastract class Maths
{
public abastract void Add (int x, int y);
}

Method overriding



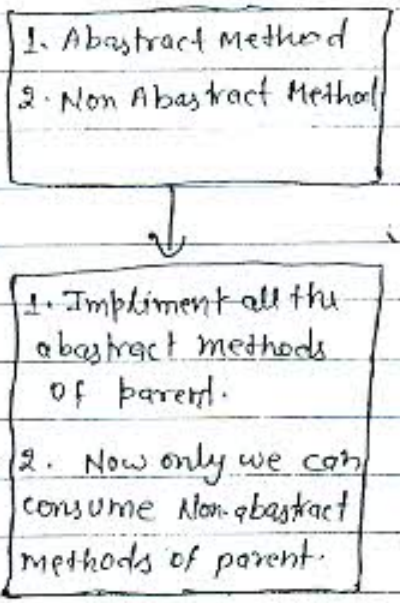
Abastract Methods



The Concept of Abastract class is near similar to the concept of method overriding, where in overriding if a method of parent class is declared Virtual we can oeimpliment the method under child class using override modifier (optional).

where as if a parent class contain any abastract method in it those methods must be even implimented under the child class using some override modifier but it is mandatory for the chilol elasses.

→ An Abastract class can contain both abastract and non abastract member in it. If any child class of abastract class wants to consume non abastracts method of it's parent, first they require to impliment all the abastract methods of parent otherwise consuming non abastract methods of parent will not be possible.



→ An abastract class is never useful to itself. because creating an object of abastract class. is not possible.

→ An abastract class can consume only by child classes that to after providing the implimentation for all the abastract method of Abastract class.

Add a class Absparent.cs and write the following -

```
abstract class Absparent
{
    public void Add (int x, int y)
    {
        Console.WriteLine (x+y);
    }
    public void sub (int x, int y)
    {
        Console.WriteLine (x-y);
    }
    public abstract void mul (int x, int y);
    {
        Console.WriteLine (x*y);
    }
public abstract void div (int x, int y);
    {
        Console.WriteLine (x/y);
    }
}
```

New child class Abschild —

```
class Abschild : Absparent
{
    public override void mul (int x, int y)
    {
        Console.WriteLine (x*y);
    }
public override void div (int x, int y)
    {
        Console.WriteLine (x/y);
    }
}
```

```
static void Main()
{
    AbsChild c = new AbsChild();
    c.Add(100,50);   c.Sub(25,4);
    c.Mul(25,25);    c.Div(500,25);
    Console.ReadLine();
}
}
```

→ Even we can not create an object of an abstract class, it is possible to create reference of an abstract class and using that reference we can invoke all it's non abstract methods and abstract method also, that are implimented in child class.

To Test this rewrite the code under Main Method of child class AbsChild as following-

```
AbsChild c = new AbsChild();
AbsParent p = c;
p.Add(100,50);
p.Sub(100,50);
p.Mul(100,50);
p.Div(100,50);
Console.ReadLine();
```

Figure.cs

```
Public double width, height, radius;
Constant float pi = 3.14f;
public abastract double GetArea();
public abastract double GetPerimeter();
```

Rectangle.cs

```
// Construct of Init
// Impliment the GetArea
// Impliment the
    GetPerimeter
```

Circle.cs

```
// Constructor for Init
// Impliment the GetArea
// Impliment the
    GetPerimeter
```

An Abastract class can provide properties to it's child classes as well as impose any restrictions on child classes also. In the above case figure is an abastract class which is defining the attributes that are commonly required in all it's child classes like Rectangle, Circle, Triangle, and also it contains a set of abastract methods, which must be implimented under all the child classes without failing.

Add a class Figure.cs and wrote -

```
public abastract class Figure
{
    public double width, height, radius;
    public const float pi = 3.14f;
    public abastrad double GetArea();
    public abastract double GetPerimeter();
}
```

Add a child clas Rectangle and write the following

```
public class Rectangle : Figure
{
public Rectangle (double width, doubl height)
{
    // Here this & base will be same
    this.width = width;
    base.height = height;
}
public override double GetArea()
    {
        return width * height;
    }
public override double GetPerimeter()
    {
        return 2*(width+height);
    }
}
```

Add a class Circle.cs and write the following -

```
public class Circle : Figure
{
  public Circle (double radius)
    {
      this.radius = radius;
    }
public override double GetArea()
    {
    return pi * radius * radius;
    }

public override double GetPerimeter()
    {
    return 2 * pi * radius;
    }
}
```

Add a class TestFigure and write the following -

```
public class TestFigure {
{       static void Main()
    Retangle r = new Rectangle (30.55, 32.06);
    r.GetArea();
  Console.WriteLine (r.GetArea());
  Console.WriteLine (r.GetPerimeter());

    Circle c = new Circle (22.25);
  Console.WriteLine (c.GetArea());
  Console.Writeline ((.GetPerimeter));
  Console.Readline();
}
}
```

Interface – It is also a type which can contain only abastract member in it.

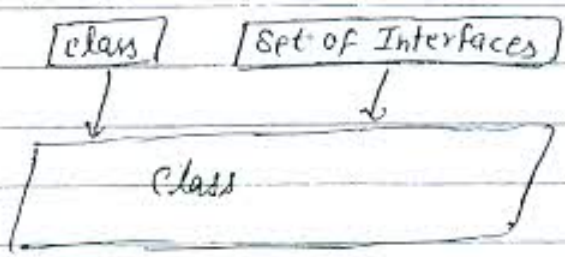
Non-Abastract class –
Only non-abastract method /Members

Abastract class-
Both Abastract and non abastract method /Members

Interface-
Only Abastract Members

The abastract member that are defined in the interface will be given implimentation under the child class as interface. We some discuss in Abastract members.

Multiple Inheritance through Interface –
We are aware that a class can be Inherited from only one single class whereas the class can inherit from any num. of interfaces at a time. that is multiple inheritance is supported through Interfaces.



The concept of Inheritance is divided in two different Catagories-
(i) Implimentation Inheritance
(ii) Interface Inheritance

If a class is inheriting from another class, we called is as Implementation Inheritance, whereas if a class is inheriting from an Interface we called is as Interface Inheritance.

Note- In Java and .Net lang's implimentation Inheritance is supported as single Inheritance only and Interface inheritance is supported as multiple. where as in C++ both are multiple.

Syntax:-

[<modifiers>] interface <Name>
{
- Abastract Member Declaration.
}

→ In Interface can not contain any variable in it.
→ Every Member of an interface is by default abastract. so we don't oreqire to declare them explicitly.
→ The default scope for an Interface members is public, where as it is private in case of a class
→ An Interface can inherit from another interface if oequired but not from any class.

Add a Interface in the project Name is Inter1.cs —

```
interface inter1
{
    void Add( int x, int y);
    void Sub ( int x, Int y);
}
```

Add a another Interface and —

```
interface inter2
{
    void Mul (int x, int y);
    void Div ( int x, int y);
}
```

To Impliment the Interfaces
Add a class Interclass.cs and —

```
class Interclass : Inter1, Inter2
{
    public void Add ( int x, int y)
    {
        return Console. Writeline (x+y);
    }
    public void Sub (int x, int y)
    {
        Console. Writeline (x-y);
    }
    public void Mul (int x, int y)
    {
        Console. Writeline (x * y);
    }
```

```
public void Div (int x, int y)
{
    Console.WriteLine (x / y);
}

static void Main ()
{
    Interclass c = new Interclass ();
    c.Add (10, 5);
    c.Sub (10, 5);
    c.Mul (10, 5);
    c.Div (10, 5);
    Console.ReadLine ();
}
}
```

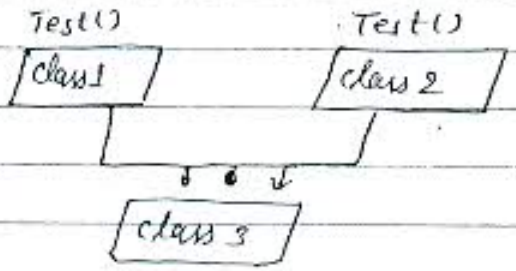
If a class is Inheriting from an interface it's main intension is to impliment the members of it's parent. But not to consume them. So Interface Inheritance does'nt provide 'Reuseoblity'.

We can not create an object of an interface, But we can create a reference of it using the object of child class. And invoke it's members that are implimented in the child. You can test this by rewriting the code under Main Method of above class.

```
{
    Interclass c = new Interclass ();
    Inter1 i1 = c;  Inter2 i2 = c;
    i1.Add (15, 25);  i1.Sub (25, 25);
    i2.Mul (5, 10);  i2.Div (15, 3);
    Console.Readline ();
}
```

Ambiguity Problem with Multiple Inheritance

Earlier we have been discussing that Multiple Inheritance not supp. through classes because of amiguity problem, that is a same member with same signature and same name can exists in multiple parents with implimentation so they can not be accomidated with in the single child class.



→ In case of Interfaces we don't have any amiguity problem because even if the similar type of problem (we have similar problem) with multiple interfaces 'amiguity' will never occured. Because the method will not have any implimentation under parent which we should be implemented under child. So the child class can provide the implimentation for the ambigious method in two diff. ways-

(i) Implimemt the method in child class only for one time. where abt the parent Interfaces, in which the method is declared assumes the implimented method was their method only.

(ii) The child class can also provide the implimentation for ambigious method separately for each interface by prefixing the method with Interface Name.