⊙ Features of C Sharp

1) object oriented programming ⇒ security & Reusability
2) Platform Independent
   ↳ Applicn that you are developing using C#, you can run them on Multiple platforms
   As of current version of .NET, there is direct support for building C Sharp applicn to run on windows, android, iphones (earlier 3rd party suppor
3) Language Independent
   ↳ C# is one lang. but like this 30+ lang. are in .NET
   programmer can develop applicn using any language
   ↳ CROSS Language Reusability
   eg. C# code can be Reuse in other .Net languages as well.
   but in C++ it can be reuse in C++ only or java can be reuse in java only

⊙ Visual Studio.Net

⇒ it's an IDE provided by Microsoft for developing .Net applicn
↳ provide support for developing :
   i) console Applicn
   ii) windows Applicn
   iii) web Applicn
   iv) web services
   v) windows services

① .Net components (3)

i) CLR ii) CTS iii) CLS iv) MSIL v) JIT

i) CLR (common language @ Runtime)

→ Responsible for :

a) Debugging → CLR sends corresponding compiler to compile corresponding code

* eg. In your project, C# code, VB. code, ASP.Net code is there ⇒ all are .Net Framework languages then CLR is going to send corresponding compiler to compile corresponding lang. code

b) exception Handling : to handle Runtime issues w.r.t error pages or Log File entry

c) security

d) version support : ensures the application which implemented in lower version can be used and upgraded automatically to higher version

ii) CTS : (common Type System) / (standard type system)

↳ All .Net Framework languages needs to undergone one common state w.r.t Data types and ranges

eg. integer in vb.net takes 4 bytes
then C#.net integer also takes 4 bytes

iii) CLS (common language specifications)

↳ language interoperability

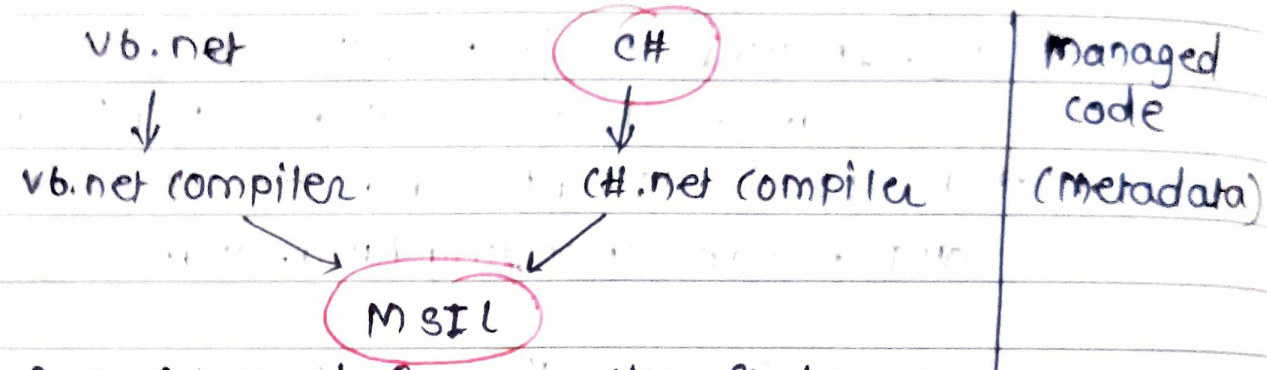
↳ code which is in one .Net Framework lang. can be used in another .Net Framework lang.

eg. same project VB person & C# person can work

↳ CPU specific code to lang. Neutral MSIL code

iv) MSIL (Microsoft Intermediate language)

↳ when you compile any .NET application at first instead of converting that into CPU specific code first it converted into MSIL.
Then it becomes language Neutral

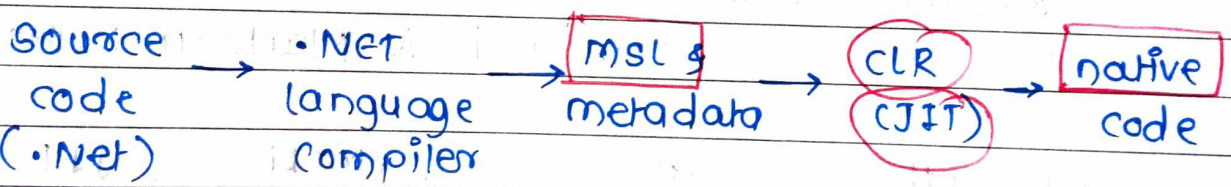


Once it is in MSIL format, then it becomes language neutral, the code will run in any .Net framework language

v) JIT (Just in Time) : used to convert MSIL into CPU understandable format

(only those classes will be loaded into memory which are there at runtime & Not all classes)

- .NET framework to develop software applications developed by Microsoft → first beta version (2000)
- This framework contains large no. of class libraries known as framework class libraries (FCL)
- Software prog. written in .NET executed in environment which is called CLR (common lang. RunTime)
- CLR: It is an prog. execution engine that loads & executes program. [It converts prog. into native code]
  - Acts as ~~framework~~ bet^n interface bet^n framework and operating system
  - i) it does exception handling
  - ii) Memory Management
  - iii) garbage collection
  - iv) it provides security, interoperability



- Framework class libraries (FCL): standard library i.e collection of thousands of classes and used to build an application.
  BCL (Base class Library) is core of FCL
  - ↳ provides: system.web
    system.windows.form
    system.xml

1) common type system (CTS)
↳ CTS provides guideline for declaring, using & managing data types at Runtime
↳ offers cross lang. communication
eg. VB.Net has integer datatype
C# has int datatype
after compilation Int32 is used by both data types
↳ CTS provides data types using managed code
↳ CTS helps in writing language-independent code

(I) Value type ⇒ value type stores data in memory allocated on stack or inline in structure
if one variable value copied into another both variables stored independently

(2) Reference type ⇒ Ref. type stores a reference to value of memory address and is allocated on heap. (Heap → For dynamic mem. Alloc^n)
⇒ it copies address not actual data

2) Common lang specifications (CLS)
↳ CLS contains rules to be followed by all .NET languages
↳ easy to implement lang. & Cross Language inheritance and debugging
↳ CLS ensures interoperability

3) Garbage Collection: It works as an automatic memory manager
↳ manage memory by automatically allocating memory according to requirement
↳ allocates heap mem. to objects
↳ when object of not in use, it reclaims memory
↳ it ensures safety of objects by not allowing one object to use content of another object

4) Just-in-Time compiler (JIT)

↳ it converts MSIL code into native code (machine specific-code)

↳ In Implicit compiln → twice compiled program

source code → compiled into → MSIL → during

MSIL code → to → native code → during 2nd

This process is called JIT compilation

JIT → Pre → before execution

→ Econo → specific part execution

→ Normal → cache mem. for fut

5) metadata: Binary info about program either stored in CLR Portable executable file or with MSIL code

→ during execution along with MSIL, metadata also get loaded in memory for proper interpretation of classes & related info.

→ metadata helps in implement code in lang. neutral manner for interoperability

extra information about types being used into your code/ Applicn & this information read by CLR

6) Assemblies: Fundamental unit of physical code grouping (assembly manifest + metadata + MSIL code + set of resources like image files)

↳ also consider as basic deployment unit, reuse, security, version control

⊙ .Net Framework Class Library

It is collection of classes, namespaces, interfaces, value types that used for .Net application.

FCL contains thousands of classes that supports.

1) Base & user-defined datatypes
2) exception handling
3) I/O stream operations
4) Access to data
5) Ability to create windows based GUI apps
6) create web-client & server apps
7) support for creating web services

⊙ CLR is basic virtual Machine component of .Net Framework
Internally CLR implements the VES (virtual execution system) which is defined in microsoft's implementation of CLI (common lang. Infra)

⊙ Code that runs under CLR is → managed code

C# code → C# compiler → converts to MSIL (CIL) or IL + metadata

(MSIL Machine Independent)

CLR provides runtime env. to MSIL code
Internally CLR includes JIT (Just In Time) compiler which converts MSIL code to machine code

⊙ **Constructor**: It is a special method present under a class responsible for initializing the variables of that class

⇒ name of constructor method is exactly the same name of class in which it present

⇒ it's a non-value returning method

⇒ each & every class require this constructor if we require to make instance of the class

But ***

```
Class Test {
        int i;
    }
Test obj = new Test(); // valid
```

* It's a responsibility of a programmer to define a constructor under class, if he fails on behalf of programmer an implicit constructor gets defined in that class by the compiler

Implicit constructors → Default constructor
(Parameterless constructor)
[initialized values with default values]
→ They are public *

Explicit constructor → user Defined constructor

⊙ What is Assemblies in .NET ?

Collection of types & resources that are built to work together & forms a logical unit of functionality

Type : 1) .exe (executable)
it contains main function

2) .DLL (Dynamic Link Library)

Assemblies provide CLR in .NET

CLR can : 1) implemented as .exe or .dll
2) they are loaded into memory when needed
3) info. about assembly can be found using Reflection
4) you can also share assemblies betn applications using GAC (Global Assembly cache)

Types of Assemblies

i) Private : they requires to be copied seperately in all appl^n Bin folder

ii) public : need not to be copied
↳ only one copy present at system level
↳ it is also called shared Assembly

iii) satellite : they are used for deployment of language & culture specific resources
Foreg. 1 app can be made with multiple lang.

⊙ Assembly

⇒ Fundamental unit of deployment

⇒ Assembly is precompiled .NET code which can run by CLR

⇒ Files with extention .exe/.DLL are known as asse

⇒ Assemblies include metadata including assembly manifest which will store information about source file, dependencies of that .exe/.DLL file

⊙ Memory management in Java

⇒ stack is linear data structure which stores a collection of objects.

↳ Follows LIFO Algorithm

① Push
② Pop
③ Top

} operations of stop

⊙ In Java automatic allocation & ~~dell~~ deallocation mem unlike c/c++ where manual allocn

⊙ Java mem. mgmt done by JVM

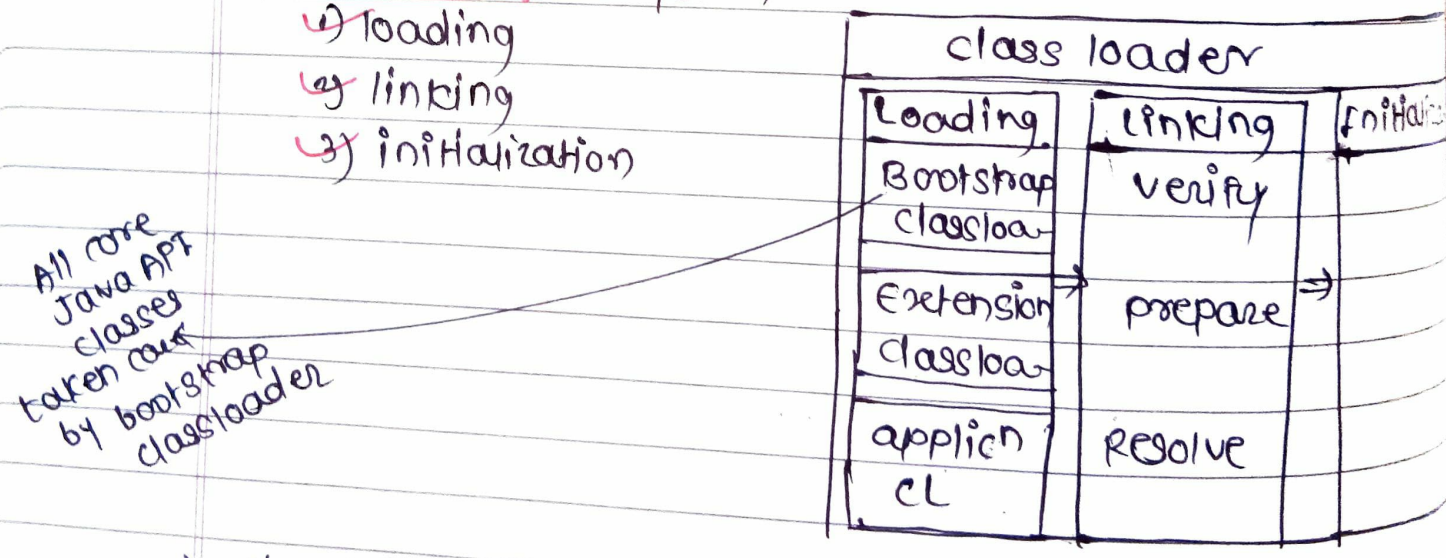
JVM Architecture

Java source file
(.java) file
↓
Java compiler
(javac command)
↓
This compiler will generate
Java class file
(.class) file

Byte code

↳ This .class file will be given as input to JVM then JVM will responsible to load & execute .class file
(it will be input to classloader subsystem)

classloader subsystem
1) loading
2) linking
3) initialization



⇒ classloader subsystem will load our classes so to load some memory bytes required

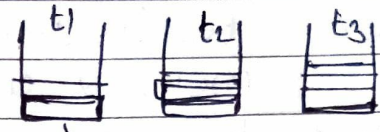
Various Memory Areas of JVM

① Method Area
⇒ Class level data will be there & all static variable stored

② Heap Memory Area
⇒ object / instance data will be stored

every Array is also object only so arrays also will be in Heap area

③ Stack Area
⇒ For every thread seperate runtime stack will be created



each entry in stack is called stackframe

ⓐ stackframe again consists of 3 parts :

①	②	③
Local variable Array	Frame data	operand stack

④ PC Registers : to hold address of current executing instruct
⇒ For every thread seperate PC registers created

⑤ Native Method stacks
⇒ For every thread seperate stack
⇒ to hold native method information

Execution engine of JVM

Ⅰ Interpreter

Ⅱ JIT compiler
i) Intermediate code generator produces Interm. code (IC)
ii) code optimizer : it is responsible to optimize IC code
iii) Target code generator : is responsible to generate machine code or native code

profiler (hotspot) in JIT compiler

⊙ garbage collector
⊙ security engine
} part of execute engine

Java Native Interface (JNI)
⇒ responsible to provide Native Method libraries

⊙ Garbage Collection in Java
⇒ when prog. executes in java, it uses memory
⇒ heap is part of memory where objects stored
⇒ heap is only part of memory involved in garbage collection process.
Also known as Garbage collection heap

when object created → allocates
⇒ JRockit JVM checks its size → small objects stored in Thread Local Area (TLA)

⇒ JVM controls Garbage Collector
⇒ we can even java program request for Garbage coll

⊙ Mark & sweep Algorithm
⇒ JRockit JVM uses this Algorithm for performing the garbage collection

⊙ .NET Memory Management : Garbage Collection

⇒ .Net Support automatic mem. mgmt.

⇒ Automatic mem. mgmt. means whenever objects which we are declaring, no longer in use .NET automatically remove that and save memory, so that mememory can be other object in use

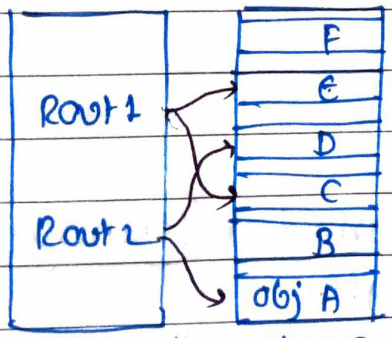
⇒ This procedure of Garbage collection done by Garbage Collector which is part of CLR

∴ CLR provides automatic Memory mgmt. with help of Garbage collector

⇒ Every Applicn has roots ; roots are pointing to memory which is taken by objects

⇒ Heap : it is part of memory where we store program which are into runtime

⇒ Root identify storage location which refered to object on Managed Heap



root 1 pointing → obj E, C

~~so~~ obj root 2 → obj A, D

so objects : A D C E are active objects which are taking memory as applicn is running

⇒ info. of active roots & active objects is collected by JIT Compiler and CLR and JIT provide this information to garbage collector

① .Net manage memory automatically.
- creates objects onto managed memory blocks

→ Allocates objects onto one of two heaps
i) small object Heap (SOH) - objects < 85K
ii) large object Heap (LOH) - objects >= 85K

→ you Allocate onto a heap whenever you use the "new" keyword in code
→ objects references held on
stack
Globals
statics
CPU Registers
→ objects not in use are Garbage Collected

② Garbage collector groups objects into
Gen 0 → short lived
Gen 1 → medium
Gen 2 → long lived

③ Garbage collection always worked on Managed Heap
internally it has optimization engine
1) marking phase : roots refers live objects
list is of live objects prepared
2) Relocating phase : updated list
3) compacting phase : space occupied by dead objects
is released

⊙ console.ReadKey() ⇒ this makes program wait for a key press and prevents screen from running & closing quickly

⊙ C# Data types

① Value Data type : derived class → System.ValueType
i) signed & unsigned Integrals ⇒ 8 types
ii) Floating point types ⇒ float, double
(32bit) (64bit)
iii) decimal → System.Decimal ⇒ 128 bits 16 Byte
iv) character ⇒ System.Char ⇒ 16 bits 2 Byte
v) Boolean

② Reference Data types: it contains memory address of variable value because reference types won't store variable value directly in memory
Built in ref. types: string, object, user-defined (Cl

③ Pointer Data type: '&' and '*'
address operator ↲ ↳ indirection operator

C# If Statement

```
Public class Demo {
        Public static void Main (____)
        { int num = 10;
          if (num % 2 == 0)
          { Console.WriteLine ("It is even no.");
          }
        }
```

*

```
Console.WriteLine (" enter number: ");
int num = Convert.ToInt32(Console.ReadLine());
```

Copy Constructor

If we want to create multiple instances with same value then we use Copy Constructor

→ Here same class is taken as parameter to it

```
namespace DemoProject
{
    class CopyConDemo
    {
        int x;
        Public CopyConDemo (int i)
        { x = i;
        }
        Public void Display ()
        {
            CopyConDemo cd1 = new CopyConDemo (10);
            cd1.Display ();  Console.WriteLine ("val: "+x);
        }
        static void Main ()
        { CopyConDemo cd1 = new CopyConDemo (10);
          cd1.Display ();
        }
    }
}
```

pass class as arg. type
↓ because class is user-d... data t...

```
public CopyConDemo ( CopyConDemo obj)
{
        x = obj. x ;
}
public CopyConDemo (int i)
   { x = i ;
   }
public void Display ()
   { Console.WriteLine (" value: " + x);
   }
public static void Main()
   {
     CopyConDemo cd1 = new CopyConDemo(10);
val: 10 ←——— cd1.Display();
     CopyConDemo cd2 = new CopyConDemo( cd1);
val: 10 ←——— cd2. Display();
   }
```

↳ obj of Copy— ty...

Seperate memory for cd_1 & cd_2 but val. will be same

⊙ **Static constructor**:

If a constructor is explicitly declared by using static modifier

```
class Test
{
    Static Test() {
                  }
}
```

Static constructor can defined Implicitly only if class contains any static variables other wise we must define it explicitly

⇒ static constructors are implicitly called
they are first block to execute
they can not be paramatrized ✓
so no overloading possible ✓ ★

⊙ Need of constructor

⇒ Every class requires a constructor to be present init if we want to create instance of that class

⇒ Every class contains implicit constructor if not defined explicitly and with the help of that implicit constructor instance of that class can be created

If constructor is implicitly present, why do we need to explicitly ~~con~~ define it ?

⇒ Implicit constructors will initialize variables of class ~~class~~ with same value even if we create multiple instances of class

⇒ If we define constructors explicitly with parameters then we can initialize fields of class with new value every time we create instance/ object of class

```
class Test {
    int x = 100;
— main (———)
{
Test t1 = new Test();→
— t2 = ———;
— t3 = ———;
Console.WriteLine(t1.x + t2.x + t3.x);
```

100 100 100

```
class Test {
int x;
public Test(int i) {
    x = i;
}
——— main —— ( )
{
Test t1 = new ———(100);
Test t2 = ———(200);
Test t3 = ———(300);
Conso———(t1.x) → 100
(t2.x) → 200
(t3.x) → 300
```

⊙ Static constructors are implicitly called whereas non-static constructors must be explicitly called

⊙ Static constructors executes immediately once execution of class starts & more over it's the first block of code to run under a class

eg. Public class Test {

```
    static Test()          1st
    {  Console.WriteLine(" This is static constr
    }
    Static void Main()
    {
      Console.Writeline("Main Method is called")   2nd
    }
}
```

1st to execute

Op ① This is static constructor
② Main Method is called

⊙ Non static constructors executes only after creating instance of class as well as each & every time the instance of class created.

⊙ In life cycle of class, static constructor executes one & only one time whereas Non-static constructor executes for zero times if no instances are created and "n" times if 'n' no. of instances created

eg. static int y;
int x;

⊙ static constructor must be parameterless, can't have parameters because static constructors are implicitly called so values can not be sent to it.

⊙ static constructors can not be overloaded unlike Non-static parameterized constructors

⊙ Every class except static class contains an implicit non-static constructor if not defined with explicit constructor

⊙ static constructors are implicitly defined only if that class contains static fields or else that construct. will not be present at all

⊙ class : user-defined type
String is predefined class in library

⊙ Object Initializer syntax

eg.

```
public Customer(int Id, String n)
{ this.Id = Id;
  this._name = n;
}
pub. void print()
{
("Id={0} & Name={1}",
  this.id, this._name);
}
```

```
class Test { — Main()
{
Customer C1 = new Customer
                (101, "Mark");
* one way (passing value in constru
* 2nd way is object Initializer
Customer C2 = new Customer
  {
    ID = 103,
    Name = "Sawabb",
  };
C2.print();
```

⇒ object Initializer syntax introduced in C# 3.0 can initialize either struct or class

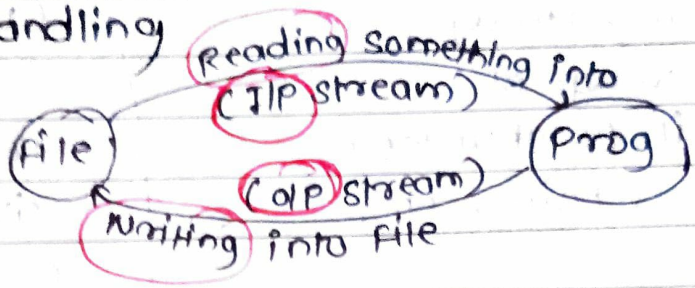
File handling in C#

→ various operations like creating, Reading, writting, appending content of file

→ File is for storing data ; operations of File ⇒ File handling

→ File is treated as stream (sequence of bytes) during read / write operation

→ System.IO namespace contains classes for IP/OP stream handling



FileStream Class

eg.

```
using system;
using System.IO;
class File_example
{
  Public static void Main()
{
FileStream fr = new FileStream ("c:\\a.txt", FileMode.
                                    OpenOrCreate);

    fr.WriteByte (66);  // Writing
    fr.close();
}
}
```

open file or create file

//mode

//location

```
// For Reading File
  FileStream fr = new FileStream("c:\\a.txt", FileMode.
                                    OpenOrCreate);

    int r = 0 ;
while ((r = fr.ReadByte()) != -1)        // end of file
  {
    Console.Write ((char) r);
  }
```

⊙ **Stream** : When file opened for read/write
↳ Sequence of bytes passing through communication path
↳ i/p stream & o/p stream
(Reading) (writing)

⊙ Classes present in System.IO namespace & used for both desktop app & web app

1) FileStream
2) BinaryReader
3) BinaryWriter
4) StreamReader
5) StreamWriter
6) StringReader
7) StringWriter
8) DirectoryInfo
9) FileInfo

```
using System.IO;
namespace FileDemo
{
  class Program
  {
    ——— main (——)
    {
      String path = @"F:\data.txt";
                    ↳ // verbatim literal
      if (File.Exists(path))
      {
        Console.WriteLine("File exists");
      }
      else {
        ——("File Not found.");
      }
    }
```

File path

// File class present in System.IO

method of File class

⊙ File compare

```
byte[] hashBytesA = hashAlg.computeHash(A);
_____ B = _____ (B);
if ( BitConverter.ToString(hashBytesA) ==
        BitConverter.ToString(hashBytesB))
{
Console.WriteLine("File match");
}
```

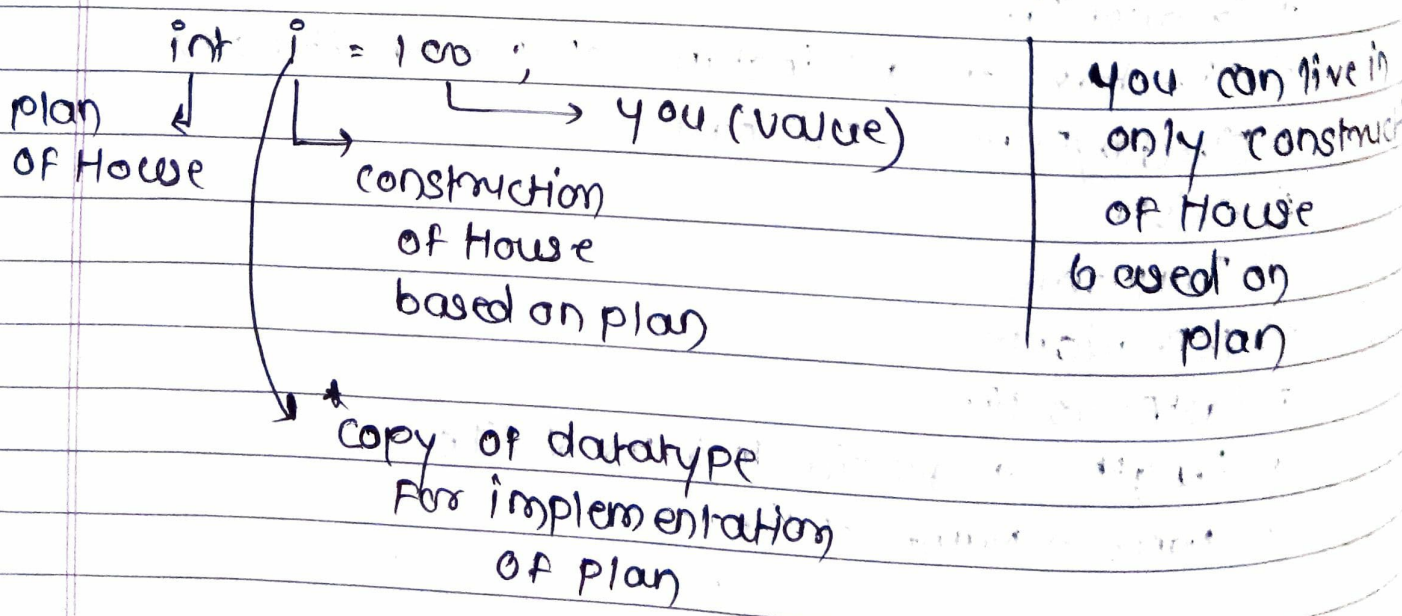
⊙ Variable, Instance, Reference

Class → user defined type
String → predefined class (made of characters)

int = 100; ✗ invalid (int is only blueprint, No mem
int i = 100; ✓ valid (i copy of type int, have mem

String = "Hello"; ✗
String S = "Hello"; ✓ (S is copy of type string)

- Every House requires a plan



⊙ Every class is new data type, to consume it you should create copy of it.
Until & unless you create copy of it memory will not be allocated

eg.
```
class A
{
    int x = 100;
pub. static void Main (—)
    {
      Console.Write (x);
    }
}
```
↓ O/P: f.x ⇒ 100

O/P: Error because x is instance member or Non-static member & you can not access Non-static member of class from static block.

// to access val. of x
Create instance of your class
First f = new First ( );
↳ instance of class / copy of class first

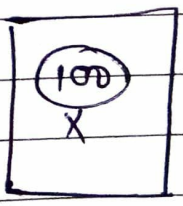
like ↓
String s

Instance of class created using → "new" keyword

✓ If no "new" keyword it is just variable
eg. First f; // unassigned variable f

First f; → [null]

f = new First();



⊙ Variable of class: copy of class that is Not Initialized

⊙ Instance of class: copy of class that is initialized by using new keyword which has its own memory & never shared with another instance

① Reference of class:

eg. First F1 = new First(); // F1 instance of class
First F2 = new First(); // F2 instance of class
(2 instances, 2 mem

Every instance has seperate memory
Any modifications performed on members on one instance will never reflect to 2nd instance

eg. f1.x = 200;
Console.WriteLine(f1.x + " " + f2.x);
↓ 200 ↓ 100

Security: Every instance is unique to itself (OOP) concept

First F1 = new First(); // F1 instance of class
First F2 = F1; // F2 is reference of class
(i.e F2 is pointer to F1)
→ F2 will Not have seperate Memory Allocation
→ F1 & F2 consume same Memory Address

Note*: variable can Not use to calling members
eg. First F2; f2.x → Invalid

But Reference can call members
eg. First F2 = F1;
f2.x → valid

* A Reference is copy of class that is created by using existed instance.
Ref. will have No Memory, only pointer to instance
but Ref. can used Like instances

⊙ Access Specifiers

special kind of modifiers to define scope of a type & it's members

C# supports 5 Access specifiers:

Default scope ← 1) private → only within ~~in~~ the class it is defi
2) internal
3) protected
4) Protected internal
5) public → Accessible anywhere

eg.

```
public class program
{
    private void m1() {
                        ("private method");
                    }
    internal void m2() {
                        ("internal Method");
                    }
    protected void m3() {   ("          ");
                    }
    protected internal void m4() {
                                    } ("          ");
    public void m5() {
                    } ("          ");
}
  static void Main (————)
{
    Program p = new Program();
}
}
```

✓ You can not declare class as private, protected, pro-int

✓ All 5 specifiers are Accessible in same class

```
class Two : Program          // Two extends Program
{
   static void Main()
   {  Two t = new Two();
      t.m1(); // compiler error because m1() is
              private can not accessible
      t.m2(); ✓
      t.m3(); ✓
      t.m4(); ✓
   }
}
```

✓ Members declared as protected are only accessible in child class & same class

✓ Protected internal can accessible → Non-child class

✓ Non-child class can Access → Public ✓
(in same project) ↳ internal ✓
↳ Protected internal ✓

✓ Child class in diff. Project can Access → protected ✓
↳ protected internal
↳ public ✓

✓ If you declare member as internal you can access it in child / non-child class in same project

✓ If protected OR internal any one is accessible then Protected internal accessible

✓ If class Non child & diff. project → only Access public

Case	private	internal	protected	protected-internal	
① same class	✓	✓	✓	✓	
② child class same project	✗	✓	✓	✓	
③ non-child same project	✗	✓	✗	✓	
④ child class Diff. project	✗	✗	✓	✓	✓
⑤ Non-child diff. project	✗	✗	✗	✗	✓

① In lifecycle of class ⇒ start execution to end exec
A static var. is initialized only one time
instance var initialized for 0 time if No inst
created & 'n' time if n instances created
eg. (P1.x); } 2 instances
(P2.x); }

② Non static variables / instance variables can be initialized through constructor also

```
eg.  public Prog (int x)
     { this.x = x;
     }
     static void main (——)
     { Program P1 = new program (50);
       ——— (P1.x); // 50
     }
```

④ static var can be initialized through constructors but everytime value will be overridden instead seperate copies

③ Constant variables ⇒ " const " keyword
↳ can not be modified once after initialization
↳ must initialize constant variables at time of declaration only
eg. const float pi; // Invalid (must init
const float Pi = 3.14f; ✓ // valid

↳ only one copy of memory similar to static
↳ static variables can be Modified ✓
but Constant var. Can Not Modified ✓

④ **Readonly** : "Readonly" keyword

↳ can not be modified like constants but after initialization

↳ It is Not compulsory to initialize Readonly var. at time of declaration

↳ can be initialized under constructors

eg. readonly bool flag; // valid

constructor { this.flag = true } // valid

→ Behaviour of readonly similar to Non-static var. that is initialized only after creating the instance of class & once for each instance of class created

(जितने objects बनते हैं उतने ही memory copies allocated but value will be constant no modified)

this will lead to memory wastage, then why multiple copies required :

eg. constructor { this.flag = flag }

Program P_1 = new Program (true); valid

——— P_2 = ——— (False); valid

X P_1.flag = false; // Invalid

(Readonly var. can not be modified after initializ)

→ only diff. betn readonly & Non-static

↓ No Modify — ↓ Can Modify after initializ

→ Constant variable is fixed value for whole class

readonly fixed value specific to an instance of class

Single copy
→ static ✓
→ constant ✓

Multiple copy
→ non static
→ Readonly

~~Modifiable~~ ~~Non-Modify~~

→ Single copy modifiable : static ✓
→ Single copy Non-Modify : constant
→ Multiplecopy Modifiable : Non-static / Instance
→ Multiple copy Non-Modify : Readonly ✓

INHERITANCE

⇒ members of one class can be consumed from Another class by establishing Parent-child rel^n

⇒ Reusability : Instead Rewritting / copy paste code
& increasing length of code &
Performance down
Use Inheritance for Reusability

Class B : A // Class B extends A

⇒ Child can inherit Parent except private members

① Parent classes constructor must be accessible to child class if Not inheritance is not possible

② child class constructor will implicitly call parent class constructor.

③ Parent class can not access purely child class member

eg. class parent {
public parent() {
("parent constructor");
}
public void m1() {
("method 1");
}
——— m2() { ("method 2");
}
}

// Parent class constructor must be public in order to get accessible by child

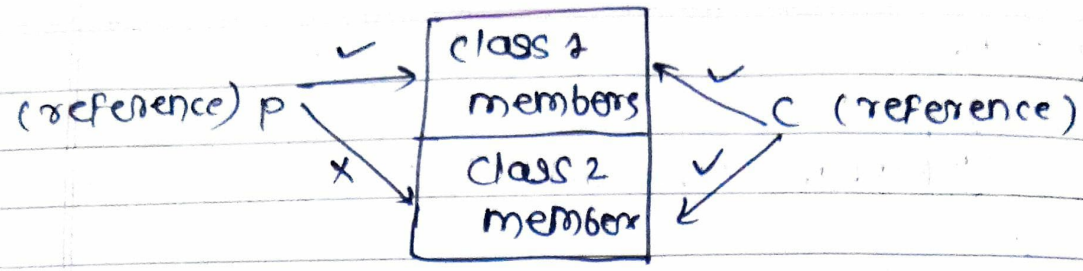
class child {
~~pub static voi~~ child c) { ("child constructor");
}
public void m3() { ("method 3")
}
child c = new child(); // Implicitly parent class constructor first get called then child class const-

o/p: parent constructor
child constructor

c.m1(); ✓ o/p: method 1
c.m2(); ✓ o/p: method 2
c.m3(); ✓ o/p: method 3

parent p = new parent(); // Implicit call to parent const
p.m3(); ✗ compiler error
p.m1(); ✓
p.m2(); ✓

④ We can initialize parent classes variable by using child class instance to make it as reference so that p = c; // p is ref. of parent class created using child class instance ref. consuming child class instance. We can not call pure child class members.
p.m3() ✗ Invalid



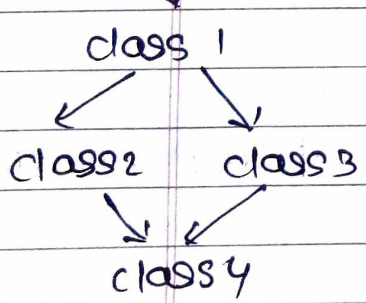
⑤ every class (user defined / predefined) has default parent class i.e object class of system namespace

Object class
- Equals()
- GetHashCode()
- GetType()
- ToString()

object → C# type
Object → IL type

⊙ Inheritance types C++

1. Single (parent - child)
2. Multi-Level (Parent → child → ~~it's~~ Grand child → Great child)
3. Hierarchical (one parent → multiple children)
4. Hybrid (multilevel + Hierarchical)
5. Multiple (one child → multiple parent)



Immediate one parent → single Inheritance
Immediate multiple parent → multiple Inheritance

⑥ In C# we do not have support for multiple Inheritance through classes.
We are provided only single Inheritance through classes

eg.
```
Public class parent {
    public parent (int i)
    { ——— (" parent constructor" + i);
    }
}
class child : parent
{
    Public child () : base (10)
    { ——— ("child constructor");
    }
}
```

child class constructor will implicitly call parent constructor if we run Blue color code get CE.

o/p: Parent Constructor 10
child Constructor

Because parent const. needs formal arg. of int type which is not passed by child constr.
∴ use child () : base (10) ✓

(7) Whenever child class instance is created, child class constructor will implicitly call its parent class constructor But only if parent class const. is parameterless.

If parent class const. is parameterized then child const. can not implicitly call its parent class constructor.
∴ explicitly call to parent const. From child class Const. & pass values to parameters.

To call parent's const. From child class we need to use " base" keyword

eg. Public child (int a) : base (a) { ——— }

child c = new child (10);

: Define classes representing entities in the

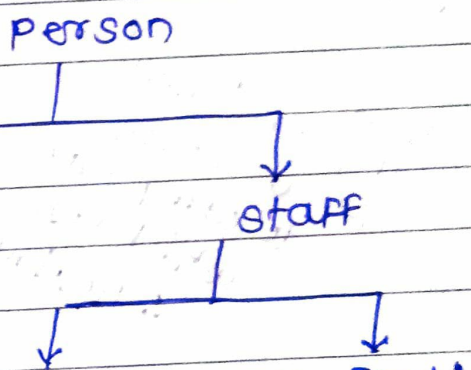
```
public class student : person
{   int class;
    char Grades;
    float marks, fees;
}

public class staff : person
{   public string designation;
    public double salary;
}

public class Teaching : staff
{   string qualification, subject;
}

public class NonTeaching : staff
{   int MgrId;
    string Dname;
}
```

} common in teach & non-teach staff



Diamond Problem Abiguity

A class B & C inherits class A
class D inherits both B & C
C if method in D calls a method
defined A (and does not override method) Teaching
and B & C overridden that method differently then from which
does in inherit: B OR C?
mbiguity is Diamond problem

⊙ Method overloading : same method name
diff. arguments

eg. Public void Test () ✓ No Ambiguity
Public void Test (int i) ✓
——— (int i, String s) ✓
——— (String s, int i) ✓

same no. of argum. & same name

Public string Test () → No overloading because return type change but Compiler error (Ambiguity)

Public void Test ()
{ —— (" 1st Method");
}

Public string Test () → Compiler Error
{ return "Hello";
}

return comes in picture at end of execn

but

Static void Main (———)
{ Program P = new ———();
P.Test(); → which Test method to call
this is Ambiguity
(2 methods same name & same parameters)

here compiler doesn't know where to start from which Test() ∴ Ambiguity

⇒ overloading comes under polymorphism (changing behaviour based on inputs)

⊙ Why Method overloading?
⇒ Method overloading is approach of defining multiple behaviours to method
where behaviour is changing with parameters
Method (Input changes → output changes)

eg. String s = "Hello World";

char ⟶ s.IndexOf('o'); 4 // 1st occurance

char,in ⟶ s.IndexOf('o',5); 7 // next occurance

Same method but
diff. parameters

If you don't use overloading
For each method programmer should give diff. name.

String parameter ⟶ s.IndexOf("ll"); 2

⊙ Method overriding : Reimplement parent class metho
in child class with same signat
and same name

overloading	vs.	overriding
→ multiple methods with same name & changing their parameters		→ multiple methods with same name & same parameters
→ can perform in same class as well as in parent-child class		→ can perform only within parent-child classes → can not perform in same class
→ while overloading a parent class method under child class, child class doesn't require to take any permission from parent class		→ while overriding pa method in child class, child class requires permission from it's parent (virtual keyword)

eg.
```
public class Parent {
    Public void show ()
    {
    }
    public virtual void Test ()
    {
    }
}
```
overloading in same class

→ virtual keyword means method is ove
—— ("Parent's Test Method");

```
Public void show ( string s)
{
}
```

```
class child : parent
{
    static void Main()
    {   child c = new child();
        c.show();
        c.Test();      // child's Test method
        c.show(10);
    }
    public void show(int i)
    {   ____("child show:" + i);
    }
}
```

overloading (parent-child)

Note: If we want to override parent's method under child class first that method should be declared using virtual modifier in parent class.
means parent class giving permission to child to override method.
(Any virtual() of parent can overriden by child class by using override modifier)

```
public override void Test()   // overriding in child
{
    ____("child's Test Method");
}
```

① overloading → Multiple behaviours as per input
overriding → changing behaviour of parent class method under child class

⊙ Method Hiding : approach of Re-implementing a parent class's method under the child class exactly with same name & same signature

⇒ child class can implement any parent's method even if the method is not declared as virtual

i.e. overriding → only virtual parent method
Hiding → any parent method

⇒ 2nd way to change behaviour of Parent Method is Method Hiding

eg.
```
Public class parent
{   Public virtual void m1() { ("parent");
                             }

    Public void m2() {  ("parent m2");
                     }
}
class child : Parent
    {
// Method override   Public override void m1() { ("child m1");
                                                }

// Method Hiding     Public new void m2() { ("child m2");
                                         }

    Public static void Main()
    {   child c = new child();
        c.m1(); // child m1
        c.m2(); // child m2
```

⑤ "new" keyword is optional
↳ it gives info. to compiler that programmer is intentionally trying to define another method with same name present in parent class

valid ✓ public void $m_2()$ { — ("child m_2");
but get }
warning if yours are intented to Hide parent. $m_2()$

⑥ we can Re-implement parent class method under child class :
1) Method overriding : permission (virtual)

2) Method Hiding / shadowing : No permission

⑦ child class instances ('c') starts calling local methods only that is re-implemented methods
⑧ But if required in any case we can also call parent classes methods from child classes by 2 approaches
↳① create parent class instance under child class
eg. parent P = new Parent();
P.$m_1()$ → "Parent m_1"
P.$m_2()$ → "parent m_2"
child C = new child();
C.$m_1()$ → "child m_1"
C.$m_2()$ → "child m_2"

↳② using "base" keyword we can call parent's method from child class but "this" & "base" keyword can not be used from static blocks
eg. → public void ParentTest1()
{ base.$m_1()$;
}
call this method from child instance i.e C.ParentTest1() ✓

```
child c= new child();
parent p = c;
p.m1();   // child m1
p.m2();   // parent m2
```

invokes child m1()

invokes parent m2()

If Parent class ref. can not call child class method, How it is called in this case?
↳ But overriden members of child class ✓

In Hiding parent Ref. does not recognize child class without permission defined parent's method in child class i.e $m_2()$

⇒ overriding method is Not Purely child members (permission taken From parent)

⇒ Hiding method is Purely child member (No permission taken From parent ∴ parent ref. does not recognize it)

⊙ Operator Overloading : multiple behaviours to an operator and those behaviours will vary based on operand types bcz which operator is used

eg. + operator

Num + Num ⇒ Addition 3 + 3 = 6

string + string ⇒ concatenation "Sau" + "preet" = Saupreet

public static int operator + (int a, int b)

_____ - (int a, int b)

_____ bool operator > (int a, int b)

} predefined library methods get invoked

⊙ Structures in C#

⇒ class is user defined data type

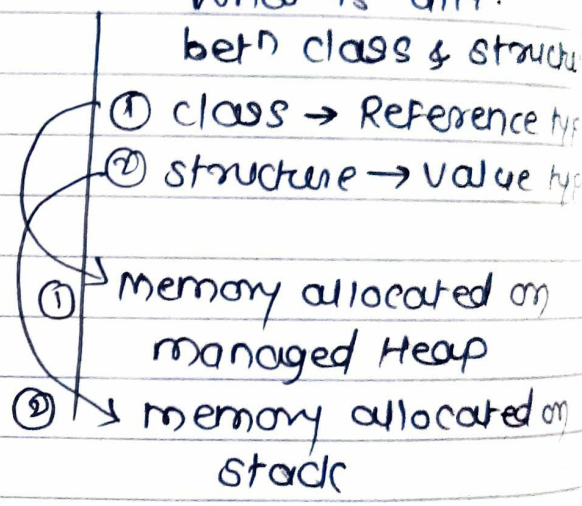
⇒ structure is use defined type

→ structures in C can contain only fields in it
structures in C# contains most of members like class
can contain like fields, methods, const., Indexer
operator methods etc

[<modifiers>] Struct <name>
{
}

eg. open code file Template

```
using System;
namespace Demo {
    struct MyStruct
    {
    }
}
```



⇒ memory allocation for instances of class is performed on managed Heap whereas Memory allocation for instance of structure is performed on stack

⇒ we use classes representing entity with larger volume of data.
structures for representing smaller volumes of data

⇒ All predefined reference type data types under library of our language which come under reference type category eg. string & object are classes
Pre-defined value types eg. int, float, bool, are structures

⇒ In case of class 'new' keyword mandatory for instance
In structure it is not mandatory

eg. MyStruct m1; ✓
m1.Display(); ✓

⇒ Fields of a class can be initialized at time of declarat
whereas it's not possible with fields in structure.

eg. Struct MyStruct { int i;
——— void Main (——)
{ MyStruct m1;
✓ m1.i = 10;
m1.Display();
}

MyStruct m1 = new MyStr ✓
OR (10)

⇒ In structure you can define only parameterized constru

⇒ Class can inherited from other classes
but structure does not support inheritance

⇒ structure can implement interface.

⊙ Properties in C#

: Property is member of class using which we can expose values associated with a class to outside environment

```
Public class circle
{
    double Radius = 12.34;
}
```

Radius is private type because no public keyword used ∴ Radius = 12.34 value not going to accessed by outside class

How to Restrict Scope & usability of variable outside class then ?

- ↳ don't declare it public
- ↳ use Getters & setters if variable is private

If I will make variable public it will get accessible outside class

You will loose control because Anyone can get & set the new value

eg. C.Radius = 30.54;

eg.

```
Public class circle
{   double Radius = 12.34;

    Public double GetRadius()
    {   return Radius;
    }
// provides only get access
// Provides set Access
    public void setRadius (double val)
    {   Radius = val;
    }
}
```

```
public class Test
{
  static void Main()
  {
  Circle C = new Circle();

  doule Rad = C.GetRadius();
  // you can only get value

  C.setRadius (56.78);
  // setting value
```

⊙ property ⇒ combination of two methods

[<modifiers>] <type> <Name>

* No Requirement of Getters & Setters Now ☺

{
[get { <Statements> }] // Get Accessor
[set { <Statements> }] // Set Accessor
}

eg. Public double Radius ~~Property~~

{
get { return _Radius ; }
set { _Radius = value ; }
}

value → Implicitly declared
No need like setter ()
double val

Class Test {

Main ()

{ circle C = new circle ();
double Radius = C.Radius ~~Property~~ ;

C.Radius ~~Property~~ = 56.78 ;

⊙ Industry standard

variable Name : _Radius eg. double _Radius
Property Name : Radius

⊙ Conditional Access to set value

eg.

public double Radius
{ get { return _Radius ; }

set { if (value > _Radius) // Assume
_Radius = value ;
}

}

⊙ Indexers : member of class, class starts behaving like virtual array

↳ it is a member which gives access to values of class just like an array

eg. Public class ~~IndexeProject~~ Employee

{ int eno;
double salary;
string job;

This are private 3 ways
1) make public
2) Property
3) Indexers but you can not directly index class

Public Employee (int Eno ———)
{ Eno = Eno;
}

eg. Employee E = new Em..
E[0] → X
~~E.Eno~~ →

write here

Indexers: [<modifier>] <type> this [int index]
{
[get { —— }] // Get Accessor
[set { —— }] // set Accessor
}

<parameter list>

eg. public object this [int index]
{ get { if (index == 0)
return Eno;
elseif (index == 1)
return salary;
else if (index == 2)
return job;
}
// get Accessor for Returning value

set { if (index == 0)
~~Eno = value;~~
else if (index == 1)
Salary = (double) value;
else if (index == 2)
job = (string) value;
}

Typecast
Eno = (int) value;
// value Implicit because data type of Indexer object value is also of type object

}

→ User Defined Type (Reference Type)

⊙ **Delegates** : it's type safe functional pointer
↳ holds ref. of method & calls method for execution

Method calls can be : 1) Creating instance (Non stat
2) using class name (static m
3) Using Delegate

```
eg.  class Program
     {
       public void Add (int a, int b)
         {  Console.WriteLine (a + b);
         }
       Public static String SayHello ( String name)
         {
           return "Hello" + name ;
         }
       Static void Main ( ——— )
       {   Program P = new Program ();
```

Calling Non-Static Method → `P. Add ( 100 , 50) ;`

```
→ String str = Program. SayHello ( " saurabh");
```

calling static method `Console. Writeline (str) ;`

```
       }
```

⊙ To call method using delegates we have 3 steps:

① Define a delegate

[<modifier>] delegate <type> <name>([parameter

```
eg.  Public delegate void AddNum ( int a, int b);
```

* & parameters types also match

* Return type of delegate should be exactly same as Method we want to call.

eg. Public delegate String SayDelegate (String name);
// Define delegates under namespace or class also fine

② Instantiate Delegate : method name passed as Parameter to delegate Constructor
eg. AddNum ad = new AddNum (P.Add) ;
ad holds address of method Add()
As Add (———) is Non-static method
∴ we need to give instantiated address i.e P.Add to ad

SayDelegate sd = new SayDelegate (Program.SayHello) ;
static method directly written with class name

③ Now, call the delegate by passing required Parameter value, so that internally the method which is bound with delegate gets executed
eg. ad (100, 50) ; OR ad.Invoke (100, 50) ;
String str = sd ("Raju") ;
i.e sd → executes → program.SayHello → code inside SayHello method executed

* delegate has No Body internally the method with definition is executed

⊙ Type Safe : Signature of delegate Must match with fun^n that delegates points to, otherwise you will get compiler error. That is why delegate is called Type safe function pointer

⊙ **Multicast Delegate** : A delegate holding reference of more than 1 method

Suppose → Multiple methods with same signature we can call them with 1 delegate

```
namespace ————
{ Public delegate void RectDelegate (double width, double height);

  class Rectangle
  {
    Public void GetArea (double width, double height)
    {
      Console.WriteLine (width * height);
    }
    Public void GetPerimeter (double width, double height)
    {
      ————(2 * (width + height));
    }
  static void Main (————)
  { Rectangle rect = new Rectangle ();
    rect.GetArea (12.34, 56.78);
    rect.GetPerimeter (12.34, 56.78);

    RectDelegate obj = new RectDelegate (rect.GetArea);
                    OR
    RectDelegate obj = rect.GetArea;
          // obj.Invoke (12.34, 56.78); // 700.38
    obj += rect.GetPerimeter;
          obj.Invoke (12.34, 56.78);
    o/p: 700.3852
         138.24
```

Same Return type, Same parameter

Normal way Creating instance

Multicast Delegate

Only 1 Method call for executing 2 methods

A method without a method body which can be bound directly to delegate & can be called

⊙ Anonymous Method : unnamed codeblock ~~binded~~ bound to delegate

eg.

```
namespace ———
{
    public delegate string GreetDelegate (string name);

    Class Anonymous_Demo{}
        static void Main ( )
        {
            GreetDelegate obj = delegate (string name)
                                    ↑ method defined by delegate keyword
            {
                return "Hello" + name + " Good Day!";
            };

            String str = obj.Invoke ("saurabh");
            Console.WriteLine (str);
        }
```

Anonym. method return type same as delegate defined in namespace

Adv: less typing

↳ For small volume / Few Lines of code then only go for Anonymous methods

⊙ Lambda Expressions ⇒ short form for writing Anonymous methods

eg.

```
GreetDelegate obj = (name) =>
    { return "Hello" + name + "Good Day!";
    };
```

⊙ Generic Delegates : ① Func ② Action ③ Predicate

① Func : Func delegate is used when method is returning value. (1 + 16 generic types) Func <i/p> (17)

② Action : action delegate is used when void method (16)

③ Predicate : Predicate delegate is used when return type of method is boolean

i/p type, o/p type

eg. `Func <int, float, double, double> obj1 = AddNum1;`
`double result = obj1.Invoke(100, 34.5f, 193.465);`

eg. `Action <int, float, double> = AddNum2;`
`obj2.Invoke(100, 34.5f, 193.46);`
(i/p)

eg. `Predicate <string> obj3 = checklength;` → method name
`bool status = obj3.Invoke("Hello world");`
var. ↳ Invoking delegate object method by passing "Hello World" parameters

You can write lambda form as well

```
Func <int, double float, double, double> obj1 = (x, y, z) =>
    { return x+y+z;
    };
```

orginal [10()]

modify [+5()]

⊙ Extension Methods : Added in C# 3.0

⇒ Inheritance one method we can extend functionalities of class

↳ Drawback: 1) we can Not apply Inherit^n on sealed class

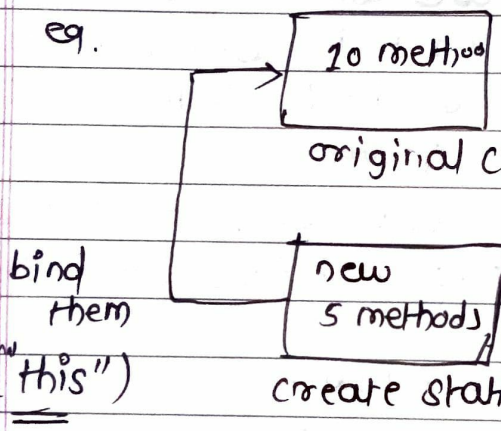
2) we can not apply Inheritance on structures

⇒ Extension Methods is Mechanism using which you can add new methods in existing class without Modifying source code of original type

⇒ No permission from original require

⇒ No Recompilation of original

eg.



"this" binding parameter

eg.

```
Static class statclass
{
    Public Static void Test3 (this Program p)
    {
        Console.W_____("Method 3");
    }
}
class Test
{   Static void Main ()
    {   Program P = new Program ();
        P.Test3();
        P.Test1();
    }
}
```

Assume class program is original code

System → System.Text

⊙ String vs StringBuilder

⇒ Strings are Immutable (Non Modifiable)

eg. String str = "Hello";

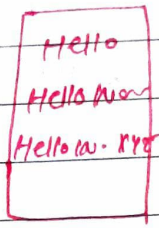
5 Byte memory on the

str = str + "world";

→ seperate memory allocation

Heap

str = str + "XYZ"



→ new copy

* Immutable
⇒ you can not manipulate value of string

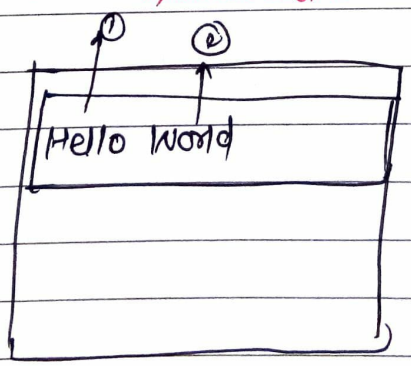
⇒ When you need static value or less modification use String

⇒ Lang Modifications ⇒ StringBuilder Class
↳ (Mutable) → you can do Modific to data

StringBuilder sb = new StringBuilder ("Hello"); → ①

// 16 characters memory will be allocated
Hello → 5 char Remaining → 11 chars
(Future Modific)

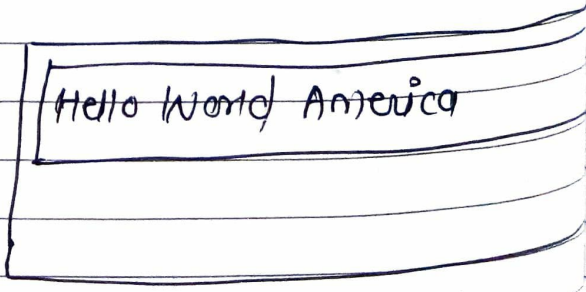
// StringBuilder Automatically Resizes to 32 (16+16)
Growable



eg.
5+1+5 = 11
Remaining 5

sb.Append ("World"); -③

sb.Append ("America"); -④



⊙ Method parameters : (4 types in C#)

i) value ② Reference ③ out ④ Parameter Arrays

↳ ① value parameter: creates copy of parameter passed, so modifications does not affect each other

② Reference parameters : "ref" keyword on method parameter causes a method to refer to same variable that was passed into a method

↳ Any changes made to parameter in method will reflected in that variable when control passes back to calling method

③ out parameter : If you want a method to return more than one value

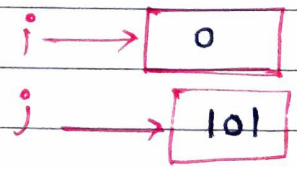
④ Parameter Arrays : "params" keyword lets you specify a method parameter that takes a variable No. of arguments.

You can sent comma seperated list of arguments or an Array or No Arguments.

"params" keyword should be last in method declaration & only 1 params keyword in method declaration

eg.

```
class Program
{ ——— Main()
  { int i=0;
    SimpleM(i); i=0
    ——.WriteLine(i); // o/p: 0
  }
Public static void SimpleM (int j) j=0
  {
    j = 101;   j=101
  }
}
```



I & J are pointing to different memory location operation on one variable not affect value of other

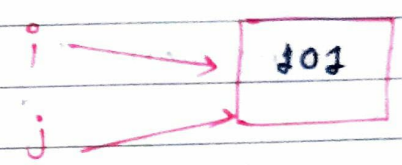
[Pass by VALUE]

```
class program {          ——— Main ()
                         { int i = 0;
                           SimpleM (ref i)
                           ———.WriteLine (i); → // 101
                         }
Public static void SimpleM (ref int j)
                         { j = 101;
                         }
}
```

we ref keyword in calling →

ref type param

flow: first i will be 0
passing ref i
∴ j will also be 0



i & j are pointing to same memory location. operⁿ on one affects other

because both pointing same memory location

Now in method called,
j is modified to 101, this will also modify i
∴ O/P = 101

③ out: in normal value returning method we can return only 1 value. eg. int sum (— —) {
return a+b;
}

out if we want Multiple return values we out parameters

eg. class Program {
——— Main ()
{ int Total = 0;
int Product = 0;
Calculate (10, 20, out Total, out Product);
———print ("sum = {0} && product = {1}", Total, Produ
}

Public static void Calculate (int num1, int num2,
{ Sum = num1 + num2; out int sum, out int produ
Product = num1 * num2;
}

④ Parameter Arrays

```
eg. class Program {
        ——— : ——— Main ()
    {   int [] numbers = new int [3];
        numbers [0] = 101;
        ——————[1] = 102;
        ——————[2] = 103;
        ParamsDemo ();
    }
    public static void ParamsDemo ( Params int[] Numbers)
    {
        foreach ( int i in Numbers)
        { ———— .WriteLine (i);
        }
    }
}
```

This will show No compiler error

Generally if we have not use params keyword, we need to pass argument to method calling

Though ParamsDemo() will not show output we want but atleast we dont get CE

If we Remove "Params" we can see compiler error at ParamsDemo() that we ~~need to~~ must pass arguments as per signature

Now,
if ParamsDemo (numbers);
then we get desired output

Also
ParamsDemo(1, 2, 3, 4, 5);

You can pass as many as you want this flexibility due to Params

⊙ Method parameters ?
What we write while defining method ✓
eg. ——— void ParamsDemo (params int [] nums)

⊙ Method Argument ?
What we pass during calling Method
eg. ParamsDemo (1, 2, 3, 4, 5);

⊙ Eg.

```
class Program
{
    ____ Main (____)
    {
        int i=5;
        int j;
        method1 (ref i);
        method2 (out j);
        Console.WriteLine (i + " " + j);   → 10, 36
    }
    static void method1 (ref int x)
    {
        x = x + x;      // 5 + 5 = 10
    }
    static void method2 (out int x)
    {
        x = 6;
        x = x * x;      // 6 * 6 = 36
    }
}
```

Ref	vs.	out
⇒ Parameters must be initialized before it passed to ref.		⇒ Not compulsory to initialize
⇒ Not require to assign value of parameters in called method		⇒ it is required
⇒ useful when modifications on given parameter required		⇒ when multiple values to be returned

⇒ Both ref & out are treated differently at run time & they are treated same at compile Time

eg. `public ____ m1 (ref int id) { }`

`Public ____ m1 (out int id) { }`

Methods can NOT be overloaded

⊙ Nullable Types in C# : C# 8.0 Feature

Null coalescing operator ??

C# → Value Type : int, float, double, struct, enums etc

↘ Ref. Type : interface, class, Delegate, arrays

⇒ By default value types are Non-Nullable

to make them Nullable we ?

eg. int i = 0 (i is non-nullable, so

I cannot set i = null (X) → CE

∴ int? j = 0 (j is Nullable int, so j = null is legal) ✓

⇒ Nullable types bridge difference betn C# types & Database types

database doesn't have concepts like value type / ref type

⇒ when it comes to work with C# & database we have to have some transformation betn Null & non-value types

eg. Assum a Form

Name :
Age :
Are you Married :

eg. bool AreYouMarried = null;

→ Compiler error

∴ bool? A—M = null; ✓

then we can work on database

```
if (AreYuMarr == true)
{
}
else {
}
```

⊙ DataTypes Conversion

① Implicit : If no possibility of throwing exception
↳ done by compiler
↳ eg. converting int into float
(No loose of any data with such type of conversions)

② Explicit : where data can be lossed
where Exception can be thrown
eg. converting float into int
(Fractional part will be loosed)
↳ we use cast operator or Convert class

eg.

```
float f = 123.45f;
int i = (int) f;
```
↑ type cast

OR

```
int i = Convert.toInt32(f);
```

if very large number

eg. `float f = 12345678933456.45f;`
`int i = (int) f;`
`print(i);` O/P: -2147483648

`int i = Convert.toInt32(f);` O/P: OverflowException value was too large

⊙ `string s = "100";`
`int i = int.Parse(s);`

⊙ TryParse() → to check if possible to convert
↳ returns bool success/fail

⊙ Exception Handling

1. compile Time Error → Syntactical ~~or logical~~ mistake during writing code → comiler checks only syntaxes

2. Run Time Error → at time of execution
   - eg. wrong implementation of logic, wrong inputs, missing required resou
   - ↳ Abnormal Termination

who is Responsible for abnormal termination → Exception (class)

Exception → logic for abnormal Termination
- ↳ contains Readonly property to display error message which is declared as virtual
  - Property ⇒ Messege()
- 1) Application Exception → Non-Fatal errors → Program
- 2) System Exception → Fatal Errors
  - ↳ exceptions caused by CLR

eg.
```
int x = int.Parse(Console.ReadLine());
int y = int.parse( ____________ );
int z = x/y
writeline(z);
```

If x = 100	
y = 5	
o/p: z = 20 ✓	
x = 100	
y = 0	
o/p: DivideByZero Exception	

RunTime exception will occur on this line
CLR is execution engine,
CLR is analyse the what is wrong with this line any no. divide by 0 is against rule
∴ this operation should not be permitted
Now CLR will pick the class which is associated with this error [i.e class DivideByZeroException]
CLR will create instance of this class & this object will be thrown to abnormal Termination of program.
↳ program Terminates on same line where error occured

if x = 100
y = 4 → o/p: FormatException
(because String/char can not parse into integer)

Runtime Error

⇒ whenever Exception occurs CLR will create instance of Matching Exception class & throws it.
Instance which thrown by CLR will perform abnormal Termination

⇒ Exception Handling : is process of stopping the abnormal termination of prog. whenever Runtime Error occuring inside prog.

→ i) we can uses user friendly errors msgs to the end users so tha we can describe about error & graceful Termination

→ 2) we can peoform Corrective action to resolve the problems that may come into picture due to error
eg. Rollback

suppose in Bank Account transaction
if First step of Amount deduction happens but at 2nd step ~~excep~~ error occurs ∴ Amount Failed to deposited in desired account
In this case if 2nd step does not execute so 1st statement should also be cancelled

⊙ Try & Catch Block

⇒ try { ____ code ____ (statements may caus Runtime errors)
}
catch (<Exception Class Name> <variable>) :
{ ____ code (statements which should execute only when there is runtime error)
}

eg.
```
static void Main ()
{
    try {  int x = int.Parse ( Console.ReadLine() );
           int y = ------------ " ------------ ;
           int z = x/y ;
           console.writeline (z) ;
    }
    catch (DivideByZeroException ex1)
    {
        console.Writeline ( ex1.Message) ;
    }
    Console.Writeline (" End of program ") ;
}
```

Block: statements that may cause error & those not require execution if got error

exceptn class → DivideByZeroException
variable → ex1

if any corrective Action to be taken or error Message to print write in catch Block

message is property under class DivideByExceptn

This will print Error msg Associated with DivideZeroExceptn class

o/p :

x = 100	x = 100
y = 0	y = y
Attempted to divide by zero	Format Exception
End of program	(to catch this Exception you can Add one more catch Block)

i/p by user

* Note: if y = y on the line int y = int.parse(———) error will occur, CLR will create instance of DivideByZeroException class & throws it to terminate execution abnormally but what happens is By Catch ( ) { } block, it will catch instance under ex1, once instance is caught abnormal termination stop

specific catch blocks important for specific corrective actions
∴ It is recommended to write default catch Block at last mentioning after those catch blocks

```
catch (Exception ex)          // Write this Block at last
{ (ex.message);               this catch Block will catch
}                                all the exception that are
                                 left / unhandled
                                 (Default Block)
```

- Catch blocks execute only if there is exception
- If on any line error occurs from there control directly goes to matching exception ~~the~~ catch block that can handle exception ⇒ then abnormal termination stops there ⇒ execute code inside catch block ⇒ then control jumps to first statement after all catch blocks.

- Finally Block : If control enters in try compulsory Finally execute No escape to it.
  - ↳ For code to execute if error occurs or Not occurs

eg.
```
try {  open file on Hard drive   // error may occur
       Write into file   // if file Readonly error may occur
}
catch (     ) {
}
Finally {
        close file
}
```

// If file opened but while writing exception occurs then next lines of code will not execute, so control will come in Finally Block & file will get closed.

Finally Block
if mandatory execution Block

Foreach Loop: No initialization and No conditions.
↳ sequential iteration from first to last.

For loop	Foreach
① For (int i=0; i<arr.Length; i++)	① Foreach (int i in arr)
② loop variable refers index of any array element	② loop variable refers value of array element
③ Irrespect of datatype of Arr. element i will be int because index is int	③ i is type of type Array element eg. (string i in arr)
* ④ Both For accessing values from array as well as Assigning value int array	④ ONLY used for accessing values from Array BUT NOT to Assign value

⇒ Array class ⟶ X System.collection namespace
↳ ✓ System namespace

⇒ Array implements IList interface.

⇒ Array Methods: Sort()
Reverse()
Copy (src, destination, n) ← no. of elements to be copied

eg. arr.GetLength(0); ✓ GetLength (int) : 32 bit integer no. of e

eg. arr.Length; ✓ Length : total No. of elements

Keyword

⊙ Implicitly typed Arrays : var arr[] = new[] {10,20,30,40};

⊙ Jagged Array : Column Size vary for row to row

↳ Arrays of Array (each Row is Single dimension Array)

eg.

1	10	14		
5	16	100	200	71
21	51	40		
15	63	38	190	

* Combination of multiple Single Dimension Arrays with diff. Column size forms Jagged Array

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

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

↳ row size need to specify

```
arr[0] = new int[5] // 5 coln in 1st row
arr[1] = new int[3] // 3 coln in 2nd row
arr[2] = new int[4] // 4 coln in 3rd row

// Print values
For (int i=0; i < arr.GetLength(0); i++)
    {
      For (int j=0 ; j < arr[i].length ; j++)
        {
          _____ (arr[i][j] + " ");
        }
    }

// Assign values of Jagged Array at time of Declaration
int [][] arr = {
                new int[5] {11, 22, 33, 44, 55},
                new int[3] {21, 31, 41},
                new int[4] {15, 25, 35, 45}
               };
```

⊙ Collection : Group of Records treated as 1 logical unit.

eg. Product → Collection of Features & id, name, size, ...

Collection

Index Based	Key-value	Prioritized	Specialized
→ Array → List	→ Hashtable → SortedList	→ Stack LIFO → Queue FIFO	→ String collection → Hybrid Dictionaries
When we Add any element to collection Array List, or Hashtable it maintains it's own internal index no. (which is Auto-generated) eg. List[0] = India	Keys to identify specific record Hashtable('IND') = India	Help you to access elements in particular sequence	Designed for specific purpose

eg.

Inter_Index	Country_code	Country_name
0	IND	India
1	UK	United Kingdom
2	AUS	Australia

⇒ Collections predefined classes present in System.Collections namespace

⇒ Dynamic Array

⇒ Collection classes represent group of objects we can perform different operations on objects

(HT)

⊙ **Hashtable** : Array & Arraylist → elements access using key which is inder no.
(difficult to remember which index for
- → stores elements in form of "key-value" pair
- → The data in HT organized based on Hashcode of k
- → key can be of any datatype unlike array inde
- → HT computes hash code for each key, Then it uses that hash code to look up elements very quickly which increases performance of applic

⇒ implement IDictionary interface
⇒ keys can be of Any Data type, Unique & Non-Null
⇒ values can be Null or Duplicate
⇒ hash Function Provided by each → key object

How HT Works?

Add object to Hashtable
like string, int, etc
⇓
Hashtable converts the long key Data which can be of any type into simple hash integer
⇓
then Data will be added to Hashtable collection

eg. keys	Hash Integ
Saurabh	0001
Nish	0002
Preetam	0003

* Note: performance of HT is less as compared to Arraylist because of this conversion key into Hashcode
* Al Faster than HT

```
Hashtable HT = new Hashtable()
    HT.Add("EID", 1001);
    HT.Add("Name", "Saurabh");
    Add(object key, object? value)
```