* OOP [Object Oriented Programming]

① Y OOP?

- helps us to think in terms of Real world objects

✓ organises the code

Ex

```
using System;
namespace ConsoleApp
{
 class Program
 {
  static void Main (string[] args)
  {

  }
 }
 class Patient
 {
  public string name { get; set; }
  public string address { get; set; }
  // Patient allocated with dr
  public Doctor doctorWhoTreat { get; set; }
 }
 class Doctor
 {
  public string name { get; set; }
 }
}
```

Practical Ex : Doctor attending Patients
Hospital management
Payroll
accounting system

② Important Pillar's in OOP?

A - Abstraction : show only wt is necessary

P - Polymorphism : Object acts differently under diff condition

I - Inheritance : Parent - Child relationship
some common thing in Parent that is inherited by Child & add something more in child

E - Encapsulation : Hide complexity
whatever shouldn't be shown outside obj hide it

③ wt is a class & Object

class → is a type, blueprint

Object → is a instance of a class

Ex :-
```
                                                  obj
                                                  instance
static void Main ( )
{
   Employee e1 = new Employee ( );
   e1 . name = "Kavya ";
   Console . WriteLine ('Hello!');

}
Class Employee                          property
{
   public string name { get; set ;}
   public string address { get; set ;}

}
}
```

④ Abstraction vs Encapsulation

Abstraction → shows wt is necessary/required.

Encapsulation → Hide complexity

$$\begin{tikzcd} & \boxed{\text{impln}} \arrow[dl] \arrow[dr] & \\ \text{Abs} \arrow[d] & & \text{Enc} \arrow[d] \\ \text{design} & & \text{coding} \end{tikzcd}$$

Abstraction is used in design phase, wt has to be shown in public

Encapsulation, during implimentation execution/coding phase developer use it like access-modifiers (private, public, protected) to implement thought process [En imp abs] [they complient each other]

Ex :
```
static void Main ( )
{
   Employee e1 = new Employee ( );
   e1 . Validate ( ) ;

}
public class Employee
{
    public string name {get; set; }
    public string address {get; set; }
}
public void Validate ( )
{ checkName ( );
  checkAddress ( );
}
```

```
private void checkName ( )
{

}
private void checkAddress()
{

}
}
```

checkN & checkA both Internal things fu'n but shouldn't be see outside so → private

(5) Explain Inheritance

$\downarrow$ defines a Parent & Child Relationship b/w 2 Class.

Ex:-

```
static void Main()
{
    Employee e1 = new employee();   → creating obj
    Manger m;   → creating obj
    m.Management();   → here Parent & Child perperty & method ✓
}
public class Employee
{
    pub str name {get; set;}
    —"— address —"—
    public void Validate()
        {
            checkName();
            checkAddress();
        }
    ~~private~~ void checkName()..
    private void checkAddress()..
}
public class Manger : Employee
{
    p void Management()
        {
        }
}
}
```

Child is a child of Employee

Parent

NOTE: it is is-her/a relationship.
- Manager is a child of Employee (Parent)

Scanned with CamScanner

(6) Explain Virtual Keyword
(or)
Virtual Method
(or)
Over-riding.

NOTE we use P class logic in C Class

↳ helps us to define some logic in the parent class which can be overridden in the child c

✓ must have Parent & Child R/n

- we use Virtual Keyword - in Parent Class
  override Keyword - in Child Class

Ex :

```
public class Employee  → Parent
{
  public virtual void Validate ( ) - -
}
public class Manager : Employee  → child
{
  //override validate
  public override void Validate ( )
  {
    //own logic
  }
}
```

(7) Overloading Method : same method names with different signature in the same class.

Compile Time Polymorphism

Diff ways of method overloading?

- ✱ no of parameters are diff — Add (int a, int b) → Add (int a, int b, int c)
- ✱ type of parameters are diff — Add (int a) / Add (double a)
- ✱ Order of parameter diff — Add (int a, int b) / Add (double b, int a)

```
Ex:- static void Main ( )
     {
       Employee e1 = new Employee ( );
       Manage m;
       m. Validate ( )
              ↳ (3) types.  ── without i/p
                            ── with i/p 1
                            ── with i/p 2
     }

   public class Employee  ┐
   {                      │ ✗
        ...               │
   }                      ┘
   public class Manger : Employee      without
   {                                     i/p
     public override void Validate ( )  ↗
     {
     }                                  with
                                        i/p1
      public ~~override~~ void Validate (bool strict)
      public ~~override~~ void Validate (bool strict, int a)
                                                  ↓
   }                                            with i/p2
```

(8) Polymorphism Poly = many
morph = change as per situation

↓

ability of a object to act differently under diff condition.

```
Ex:-  static void Main ( )
      {
       Employee e = new Manager ( );    → Employee act
                                          like Manager
        e = new Supervisor ( );         → Employee act
      }                                   like supervisor
      public class Employee {
      }
      public class Manager : Employee {
      }
      public class Supertvisor : Employee {
 } }
```

⑨ 2 Kinds of Polymorphism

$$\begin{tikzcd} & \text{2 Kinds of Polymorphism} \arrow[dl] \arrow[dr] & \\ \text{Static Polym} & & \text{Dynamic Polym} \end{tikzcd}$$

- Static [Build] Polymorphism / Compile Time :- is implemented by Method overloading
- Dynamic Polymorphism / Runtime :- Method Over-riding (p-c R/n)
  - ✓ virtual in Parent class use
  - ✓ override in Child class

Ex:-
```
static void Main()
{
Employee e = new Manager();
  e.validate();          → Dynamic P
  e = new SuperVisor();
  e.Validate();          ~~→ Static P~~
}
```

⑩ Explain operator overloading
↓
concept of Polymorphism

where we can re-define operator like +, -, x with additional functionalities.

Ex:-
```
static void Main()
{
  var x = "Kavya" + "Shree";
  var O1 = new SC(10);
  var O2 = new SC(20);
  var O3 = O1 + O2;
}
~~public class~~ SC
~~{~~ ~~pu~~
~~public~~ ~~[illegible]~~
```

→ string concatination
but // var x = 1+2
↓
arithmatic addition
[default Polym work. Net]

Q: How to do custom operator OLoading?

SC = Sony Class Variable name

```
public class SC
{
  private int SV;
  public SC (int val)
  {
    SV = val ;
  }
  public static SC operator + (SC arg1, SC arg2)
  {
    return new SC (arg1.SV + arg2.SV);
  }
}.
```

operator → Keywork

SC arg1 → 1st instance of SC

$\boxed{30}$ o/p

(11) Abstract class
↓
is a 1/2 / Partially defined Parent class

* where some implementation is defined & some implementation is left to the child classes to be defined.

ex:-

```
static void Main ( )
{
  Customer x = new Customer() ;
  x. CalculateDiscount ( );
}
public abstract class Customer {
  public string name {get; set; }
  public string address { get; set; }
  public string productName {get; set;}
  public string productAmount { get; set;}
  public abstract /virtual decimal CalDis ( )
  { throw new NotImplementedException ("Child Class")
  }
```

new Customer() → Error × instance

as it is 1/2 class × cannot create instance of it.

fully defined

this is partially defined as we override it

```
public class GoldCustomer : Customer
{
  public override decimal CalculateDiscount()
  {
    return productAmount -10;        → 10% Dis
  }
}
public class SilverCustomer : Customer
{
  public override decimal CalculateDiscount( )
  {
    return productAmount -5;        → 5% Dis
  }
 }
}
```

---

* are abstract methods virtual?
↓
Yes , so we can directly override in CC

---

* can we create a instance of Abstract class?
↓
No , Compiler throw up Expection Error

---

* is it Compulsary to implement Abstract methods in Childclass
↓
Yes , we have to over-ride it in C class

---

* Y simple base / Parent class replace Abstract class?

lack logic / not good programming / no clean code.

Ex:-

```
public class Customer
{
   public decimal calculateDiscount ( ) {
       throw new NotImplementException();
       or return 0;
}
```

so use abstract
so pure base class

(12) Interface
↓
is a contract, i.e a legal binding b/w Developer & Consumer

always public

i.e Developer who created the class & Consumer who is using the class

* we can make better impact analysis as bonding thr.
change management & breaking change] Control.

Ex :

ICustomer points to GC & SC ∴ Polymorphism in action.

```
static void Main ( ) {
    ICustomer  x = new GoldCustomer ( );
    x.name = 'Kavya';
    x.productAmount = 100;
    x.CalDiscount ( );
    ICustomer  x1 = new SilverCustomer( );
    x1.CalDiscount ( );
}
public interface ICustomer
{
  properties [ string name {get; set; }
             [        address
             [ -11-   productName       -11-
             [ decimal productAmount
  method  ←  decimal calculateDiscount ( );
}
public abstract class Customer : ICustomer
{  public string name {get; set; }
                  address
      -11-        PN         -11-
                  PA
   public abstract decimal CalDiscount ( );
}
public class GoldCustomer : Customer ---

public class SilverCustomer : Customer ---
```

contract

we ✗ can't write logic inside Interface
o/y pure signature

promise to follow i.e follow property & method

if this not thr then no alert, if breaked high impact

$$ICustomer \xrightarrow{follow} Customer$$
GC SC

all property & method, as all coming in same inherit

Multiple Inheritance (or) if I want to change interface wts the best practice?

if the interface has to be changed use Multiple - Inheritance

```
public interface ICustomer
{

    —||—

}
public interface ICustomerWithInterest : ICustomer
{
    decimal CalculateInterest ();
}
public class Customer : IC , ICWI
{

    —||—
    public decimal CalculateInterest ( )
        { return 0;
        }
    }
}
```

added another property / method in Interface

Multiple inherit

```
static void M( )
{
    ICustomer x = new CC ( )
      x.name = 'Kavya';
      x.CalculateDiscount ();
    ICustomer x1 = new SC ( );
      x1.CalculateDis ( );

    ICWI c = new SC ( );
        c.CalcuteDiscount ();    → old interface
        c.CalcuteInterest ();    → new interface
}
```

is?