

* OOP [Object Oriented Programming]

① Y OOP?

- helps us to think in terms of Real world objects

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; }
    }
}
```

✓ organises the code

Practical Ex: Doctor attending Patients
Hospital management
Payroll
accounting system.

② Important Pillars in OOP?

- A — Abstraction : shows only what 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
or add something more in child
- E — Encapsulation : Hide complexity
whatever shouldn't be shown outside obj hide it

③ What is a class & Object

is a type,
blueprint

is a instance
of a class

Ex :-

```
static void main()
```

```
{
```

```
Employee e1 = new Employee();
```

```
e1.name = "Kanya";
```

```
Console.WriteLine("Hello");
```

```
}
```

```
class Employee
```

```
{
```

```
public string name {get; set;}
```

```
public string address {get; set;}
```

```
}
```

```
}
```

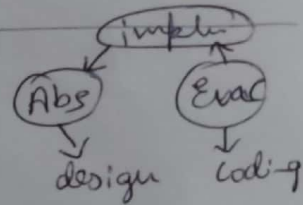
obj
instance

property

④ Abstraction vs Encapsulation

shows what is
necessary/required.

Hide complexity



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

Encapsulation, during execution/coding phase developers use it like access-modifiers (private, public, protected) to implement thought process. [ex imp abs] [they complement 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()
```

```
{
```

```
}
```

checkName checks both internal things but shouldn't be set outside so -> private

⑤ Explain Inheritance

↓
defines a Parent & child Relationship
b/w 2 class,

Ex:-

```
static void main()
{
    Employee e1 = new Employee();
    Manager m;
    m.Management();
}

public class Employee
{
    public String name {get; set;}
    public String address {get; set;}
    public void validate()
    {
        checkName();
        checkAddress();
    }
    private void checkName() --
    private void checkAddress() --
}

public class Manager : Employee
{
    void Management()
    {
    }
}
```

creating obj

creating obj

here Parent & child property & method

child is a child of Employee

Parent

NOTE: It is is-a relationship.

- manager is a child of Employee (Parent)

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

NOTE

we use
 P class logic
 in class

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

✓ 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 → child : Employee
{
    // 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?

Add (inta, intb)
 Add (double a)
 Add (double a, double b)

- * no of parameters are diff
- * type of parameters are diff
- * order of parameter diff

Add (inta, intb)
 Add (inta, intb, intc)

Ex:- static void Main ()

```
{
    Employee e1 = new Employee ( );
    Manager m;
    m.Validate ( )
    }
    ↳ ③ types.
```

without i/p
with i/p 1
with i/p 2

```
public class Employee
{
    ...
}
```

```
public class Manager : Employee
{
    public override void Validate ( )
    {
    }
    public override void Validate (bool strict)
    {
    }
    public override void Validate (bool strict, int a)
    {
    }
}
```

without i/p

with i/p 1

with i/p 2

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 ( );
    e = new Supervisor ( );
}
```

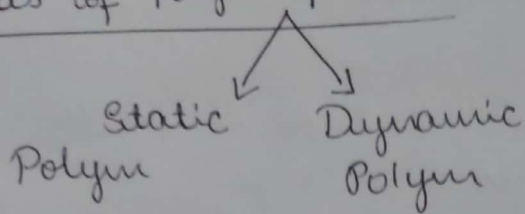
Employee act like Manager
Employee act like Supervisor

```
public class Employee {
}
```

```
public class Manager : Employee {
}
```

```
public class Supervisor : Employee {
}
```

⑨ 2 Kinds of Polymorphism



- Static Polymorphism / Compile Time :- is implemented by Method Overloading
[Build]
- Dynamic Polymorphism / Runtime :- Method Over-riding (p-c RV)
 - ✓ 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 = "Kanya" + "Shree";
    var O1 = new SC(10);
    var O2 = new SC(20);
    var O3 = O1 + O2;
}
```

~~public class SC~~
~~{~~
~~public void~~

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

Q: How to do
custom
operator
Overloading
?

```
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);
    }
```

```
}
```

SC = Some
Class
Variable
name

Keyword 1st instance of SC

30 p/p

II) 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 ();
}
```

error X instance

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

```
public abstract class Customer {
```

```
    public string name { get; set; }
    public string address { get; set; }
    public string productName { get; set; }
    public string productAmount { get; set; }
```

fully defined

this is partially defined as we are not

```
    public abstract / virtual decimal CalDis ( )
    {
        throw new NotImplementedException ("Child class");
    }
```

```

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 Exception Error

* is it Compulsary to implement Abstract methods in Child class

↓

Yes, we have to over-ride it in C class

* 7 simple base / parent class replace Abstract class?

lack logic / not good programming / no clean code.

Ex:-

```

public class Customer
{
    public decimal CalculateDiscount()
    {
        throw new NotImplementedException();
        // or return 0;
    }
}

```

so we
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 bounding this.

change management
& breaking change } Control.

Ex :

```
static void Main () {  
    ICustomer x = new GoldCustomer ();  
    x.name = 'Kavya';  
    x.productAmount = 100;  
    x.CalDiscount ();  
    ICustomer x1 = new SilverCustomer ();  
    x1.CalDiscount ();  
}
```

ICustomer
Points to
GC & SC
∴ Polymorphism
in action.

```
{  
    public interface ICustomer  
    {  
        string name {get; set; }  
        address  
        -- product name --  
        decimal productAmount  
        decimal calculatedDiscount ();  
    }  
}
```

```
public abstract class Customer : ICustomer
```

```
{  
    public string name {get; set; }  
    address  
    -- PN --  
    -- PA --  
    public abstract decimal CalDiscount ();  
}
```

```
public class GoldCustomer : Customer ---
```

```
public class SilverCustomer : Customer ---
```

we X can't
write logic
inside
interface
only pure
signature

promise to follow he follow
property & method

if this not
then there
no alert
if broken
high impact

ICustomer follows Customer
GC SC same
all property & method, as all coming in 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
{
```

—||—

```
}
```

added
another
property
/ method
in Interface

```
public interface ICustomerWithInterest : ICustomer
{
```

```
    decimal CalculateInterest();
}
```

multiple
inherit

```
public class Customer : IC, ICWI
{
```

—||—

```
    public decimal CalculateInterest()
    {
```

```
        return 0;
    }
```

```
}
```

```
}
```

```
static void M()
{
```

```
    ICustomer x = new EC();
```

```
    x.name = 'Kanya';
```

```
    x.CalculateDiscount();
```

```
    ICustomer x1 = new SC();
```

```
    x1.CalculateDis();
```

```
    ICWI c = new SC();
```

```
    c.CalculateDiscount();
```

```
    c.CalculateInterest();
```

old instance

new instance

```
return 0;
```