OOP

oop : is a programming paradigm , which uses objects and thier interactions for Building computer programs (easy To understand)

Why use oop :- *save devoplment Time and cosT by reusing code
* EaSier debugging

fundamenTal & oop:-

1- inherTance
2- abSTracTion
3- encapSulaTion
4- Poly morPhiSm
5- aggregaTion

oop is an aPProach basedon objecTs interacTing with each otherTo make TaskK

objecTs have data & meThods if they in The Same class both will be same

class :- is a category of similer objects and doesn't Hold any values of the object ATTribuTe

object :- abSTract DaTa TyPe created by a DeVeloPer can ~~incutde~~ include PropertieS and meThods

MeThods : is a Procedure asSociaTed with a message and aTTaning an object

Variable : is a reserved Place To Stove DaTa in it ~~until~~ till needed

conSTanT :- is JuST LiKe Variable but it's Value always doesn't chang

InTeger TyPes :- sByTe, byTe, ShorT, uShorT, inT, uinT, Long, uLong
STring :- String
floaT TyPes :- floaT, double
character TyPes :- decimal
Boolean TyPes :- BooL

Loop :- is a basic programming construct That allows Repeated execution of a fragment of Source code

class Modifiers :- change The way a class can be used

Access Modifiers : Describe How a class can be accessed

NON-Access modifiers : Describ How a class can be Manipulated

No Modifier :- class is accessible with its Package only

Public :- class is accessible by any class in any Package

Static :- class can't be instainted

ATTributes : class Variables in it's Body

Interface :- is Defined as a Syntactical contract That all The classes inherting The interface should follow

Abstraction: Working With SomeThing we know To use without Knowing HoW it works internally

VirTual instead of abSTract BefOr The name of the meThod

inSide VirTual There is a body of meThod conTains insTrucTions we can overRide for This VirTual meThod in oTher classes

EncapSulaTion :- Prossec of enclosing one or more items within a PhySical or Logical Package and Prevent access To implemention DeTails

Polymorphism : Ability of Two Diffrent object To Respond To Take Same Request message in their own unique way

Static Polymorphism :-
Function overLoading :- you can have maltiple Definations for the same function name in The Same Scope

operator overLoading :- you can Re-define or overLoad most of the built-in operators available in C#

Dynamic Polymorphism :-
- you can't create aninstance of an Abstract class
- you can't Declare an Abstrad method outSide an AbsTract class

Behavior of objects is modeled by the Definition of methods in class

get and set modifier:- get and set modifier or Accessor mostly used for storing and Retreving value of Property Type

Garbage collector Takes care of Releasing Unused objects

object when ~~Released~~ Released we Have Destroy the Refrence ~~of it~~ of it

class is at stack, Attributes and methods at the Heap

Inheritance :- It's a fundemental principle of it allows class To "invert" Behavior or chracteristic of another

Protected : Defines class members which are not visible To users of the class But Visible To the inherting class

Protected internal : Defines class members which are Both internal visible within the entire Assembly and Protected NOT visible out Side The Assembly but visible To the class who inhrit it.

- When a class is Declared Sealed it cannot be inherited Abstract classes can't Be Declared Sealed

Aggregation : is when an object consists of a composite of other objects That Work Together

Composition :- A filled rhombus Represent composition which is an aggregation where The ~~co~~ ~~composition~~ components can't exist without the aggregate.

class : Human

objects
ATTribute=
Variables=

first Name
LasT name
age
Salary
adress

ACTion = methods

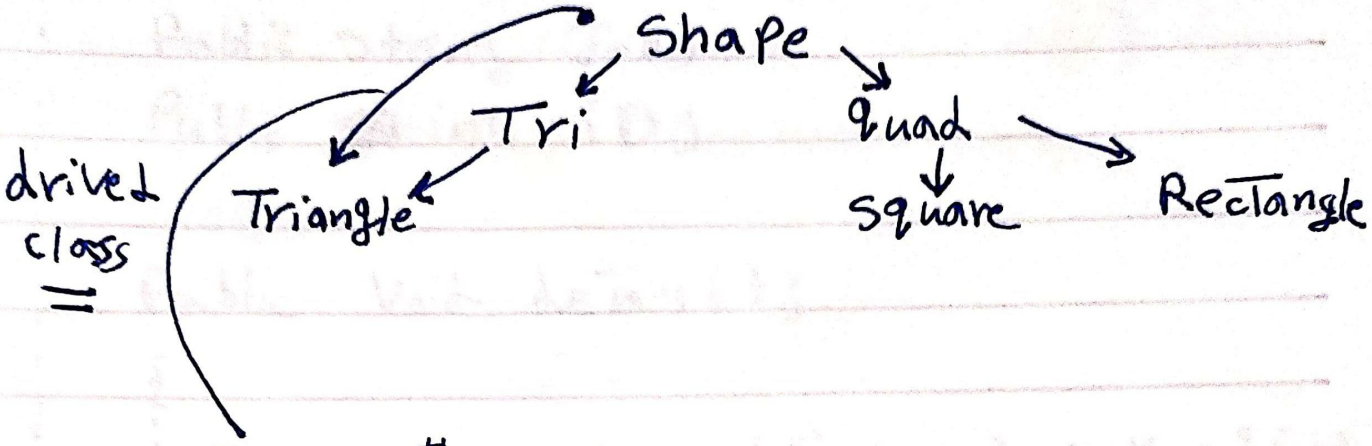
eaT
Drink
sleeP
walk
geT Salary

objects, ATTribute : is the characteristics of the class. (element)

Method : is the Procedure associated with a message

Interhance

Base class



Since there is one Type in Tri classes Some can inhrit from the Base class

```
class Human
{
    Public String name;
    Public String Job;
    Public int iD;

    Public Void deTails();
    {
        Console.WriteLine($"{name} work as {job}
        and his ID : {iD});
    }
}

class Human_is_noT_intellegent : human
{
    Public Void anoTher_deTails()
    {
    Console.WriteLine ($"{name} works as {job} and
    his ID:{iD})
    }
}
```

```
class Program
{
    STaTic Void Main (String[] args)
    { Var Person1 : new human();
      Person1. name =      ;
      Person1. iD =        ;
      Person1. Job =       ;
    } Person1. DeTails();
}
```