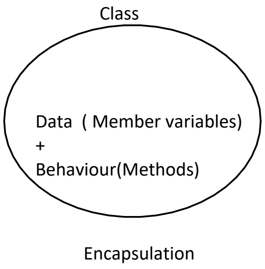
OOPS Pillars

Thursday, January 30, 2025 8:57 AM

Encapsulation -- Data (information) -- Secure you data
It hides the internal state and functionality of an object and only allows access through a public set of functions


Class

Data (Member variables)
+
Behaviour(Methods)

Encapsulation

Set --> Data Members
Behaviour - Member functions
When we wrap up in one single unit(class , interface , struct etc.)

Real Examples :

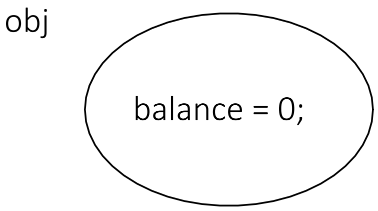
1. Bluetooth
2. HR(Salary - increment , Basic Pay)
3. Elections -- (Voting -- Age Limit)
4. Account -- deposit ---

   An IT or job related Companys, There will check bgv and if we apply another role they will show our details that we can not access to change

   Social Media, like some info are hidden like contact,email, dob but can see the posts.
   MFA--- Authenticator

   Here hiding means to hide internal data from outside the world. Main purpose is to protect the data from misuse by the outside world.

Abstraction ---> services (100) ---- 50 to the services -- ATM machine


obj

balance = 0;

Advantages of Encapsulation:

1. Data protection : we can validate the data before storing it in a variable
2. Flexibility : allowing developers to easily modify , update the code
3. Security
4. Control
5. Data Hiding : no idea about the inner implementation

Encapsulation and abstraction in C#

Encapsulation	Abstraction
Data hiding	Implementation hiding

Protect our data (Wrapping the data + Members in a single unit)	Exposing the services (Exposing the interface or Abstract class to the user and hiding the implementation (child class implementation)
Access specifiers / get & Set	Abstract class or Interface

Inheritance : when we want to inherit the methods defined in base class and consumed by child class.

Class A (Parent) Class B (Child)

Members → consumed by class B

Syntax:

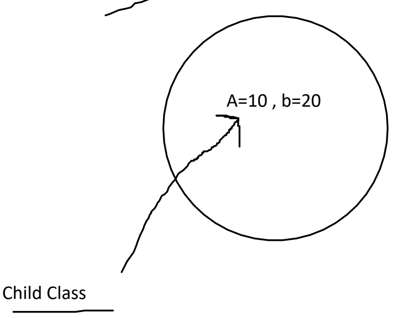
```
Class A
{
Members

}
Class B : A
{
//can consume the members of A from here

}
```

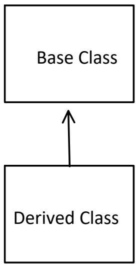
1. Why child class cannot consume Private members of Parent

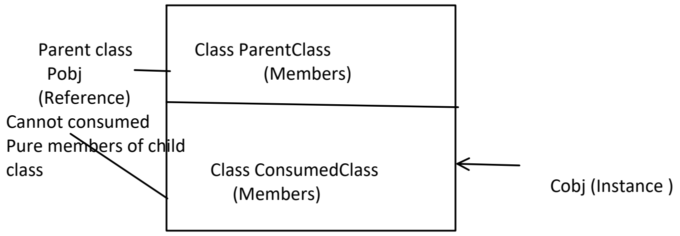
Car // Car license
Property // Property Papers(Aadhar card)

```
Parent class
{

int a ,b;


}

Child class : Parent
{

Child c  = new Child()

}
```

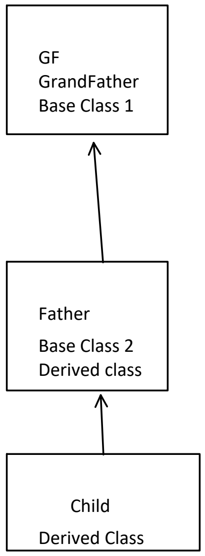
Implicitly the parent class constructor will be created to initialize the value



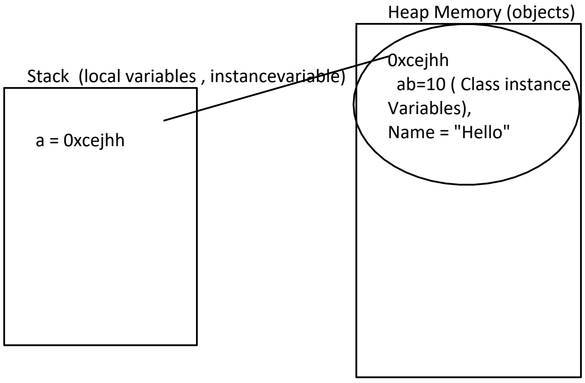




Multilevel



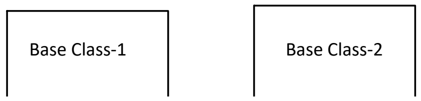
Abc a = null;

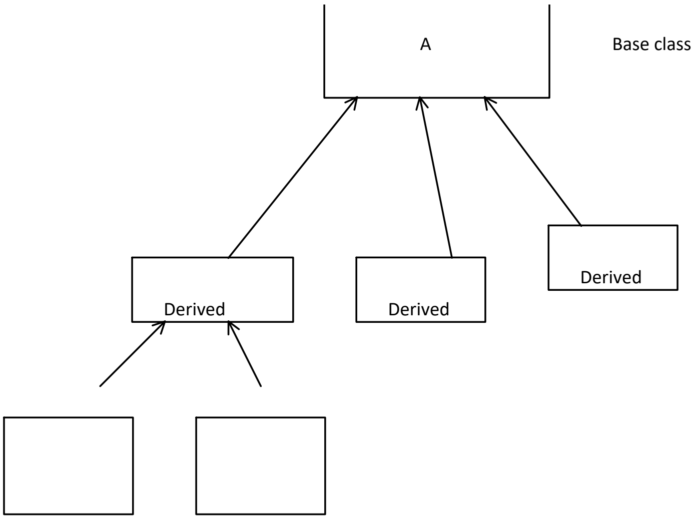


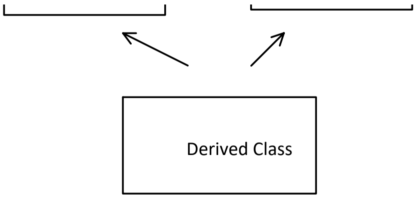
Multiple Inheritance

Hierarchical Inheritance

Bas







Polymorphism : Many Forms .

Operators or functions

Eg : It is a concept by which we can perform a single task in different ways .. A single entity behaves differently in different cases ..

Eg : Behaving in different ways depending on the input received which is known as polymorphism i.e. when the input changes , automatically the output or the behaviour also changes.

In organization --> 1. visitor - id card - add some details in a register.
2 . Employee -- he welcome
3. CEO -

Types of Polymorphism :

Static : early binding / compile - time ---> method overloading , Operator Overloading ,Method Hiding
Dynamic : late binding / Run time --- > Method overriding

Virtual Keyword in Polymorphism

1. We can declare it in base class
2. Overridden in the derived class using override keyword
3. Supports runtimepolymorphsim , which allows the calling of method and to be resolved at runtime
4. If not overridded the base class implementation is used

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using EncapsulationExample;

namespace EncapsulationExample
{
  internal class PolyProgram
  {

    //Method Overloading
    public void Add(int a , int b)
    {
      Console.WriteLine("The sum is :" + (a + b));
    }

    public void Add(float a, float b)
    {
      Console.WriteLine("The sum is :" + (a + b));
    }

    public void Add(string a, string b)
    {
      Console.WriteLine(a + " " + b);
    }

    public void Add(int a, int b , int c)
    {
        Console.WriteLine(a+b+c);
    }

    public void Add(float a, int b)
    {
        Console.WriteLine(a + b);
    }

    public void Add(int a, float b)
    {
        Console.WriteLine(a + b);
    }



}

//Method Overriding
class Class1
{
   public  void interest()
    {
        //Parent class logic is same for all child classes
        Console.WriteLine("Parent class interest calculated  :");

    }



}

class savingAccount : Class1
{

    public  void interest()
    {
        //redefining the method show()
     int a=20 , b=20;
       base.interest();
        Console.WriteLine(" child class interest calculated as 5%");


    }




}

class LoanAccount :savingAccount
{

    public  void interest()
    {
        //redefining the method show()
        int a = 20, b = 20;
        Console.WriteLine("10%");


    }

}










class Program
{
    public static void Main(string[] args)
    {
        /* PolyProgram polyProgram = new PolyProgram();
         //static polymorphism // compile-time polymorphism
         polyProgram.Add(2.9F, 3.5F);*/


        savingAccount obj = new savingAccount();
        obj.interest(); // Run time polymorphism
    }
  }









}
```