Encapsulation

One of the 4 pillars (support)

Readability _ Maintability _ Reusability_ Extensible

Each datatype with some functions in a capsule.

Access Modifiers:

- ➔ **Public** (put the functions in it)

طريقة تسمح لهم يتواصلوا معايا كده

- ➔ **Private** (put datatype _it's not accessible)
  Not from hackers but other developers [redacted]

That makes this data not accessible any time & everywhere (not global).

Maintability, can (edit) in one place.

Data validation.

اشوف الداتا عاجباني و لا لا.. اتطمن يعني

Effect of change.

انا بس اللي اقدر اعدل..

Standalone of function

من برة ..

Outside the scope

Cannot access Private

Display (instance) -> Display (a);

Member Function

من جوه

Inside datatype

Can access private data

Instance. Display () -> a. Display ();

This -> holds the address of the caller variable.

Lab2

```
/*

    1- Create Data Type That Encapsulates
        Account Which Has (Number, Balance)
    2- Support Your Data Type With The Following
        Functions
        A- Deposit
        B- Withdraw
        C- Transfer
    3- In Main Test Your Data Type, By
       Creating Two Variables, And The Transfer
       Some Amount From One To Other.
    4- Call Display For Each Instance.


    Bonus:
        Try To Make The Transfer Function
        As Standalone Function, With Test Case.
*/
```

Answer

Start here | *main.cpp

```cpp
#include <iostream>
#include <stdio.h>
using namespace std;
struct account{
  private:
     int Number;
     float balance;

  public:
    float Deposite (float _amount)
    {
        balance += _amount;
        return balance;
    }
    bool withdraw(float _amount)
    {
        if (balance >= _amount)
        {
            balance = balance - _amount;
            return true;
        }
    }
    void transfer(account &b, float _amount){
        if (this->withdraw (_amount) == true){
            b.Deposite(_amount);
        }
    }
    void display()
    {
            cout << "balance" << balance << endl;
    }
};
```

Start here | *main.cpp

```cpp
    }
};
void transfer(account &a, account &b, float _amount){
    if (a.withdraw(_amount))        //bonus
     b.Deposite (_amount);
}

int main()
{
     account a;
     account b;



     a.Deposite(3000);
     b.Deposite(1000);

     a.withdraw(200);

     a.transfer(b ,300 );

     a.display();
     b.display();

     transfer(a,b,100);    //bonus
     a.display();
     b.display();


     return 0;
}
```