

# 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

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.. اتضمن 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

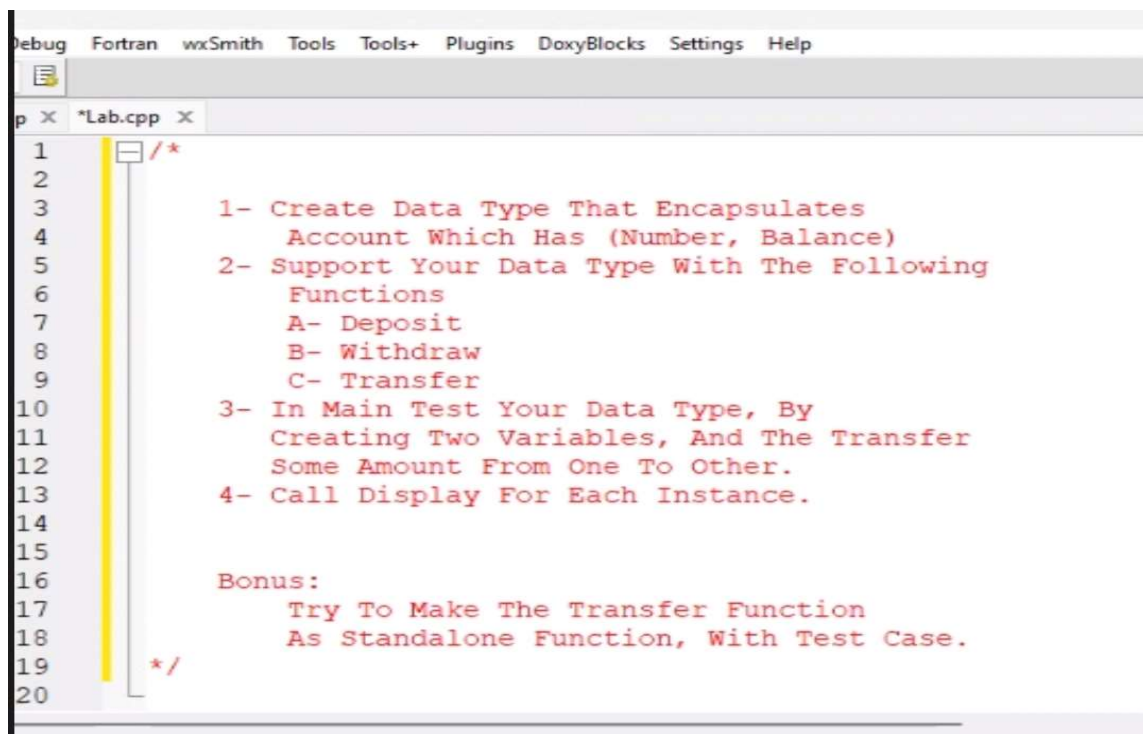
Inside datatype

Can access private data

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

**This ->** holds the address of the caller variable.

## Lab2



The screenshot shows a code editor window with a menu bar (Debug, Fortran, wxSmith, Tools, Tools+, Plugins, DoxyBlocks, Settings, Help) and a tab labeled '\*Lab.cpp'. The code is a multi-line comment in red text, numbered 1 through 20 on the left margin. The comment describes the requirements for a C++ lab: creating a data type for an account, supporting deposit, withdraw, and transfer functions, and testing it in main. A bonus task is also listed.

```
1  /*
2
3      1- Create Data Type That Encapsulates
4          Account Which Has (Number, Balance)
5      2- Support Your Data Type With The Following
6          Functions
7          A- Deposit
8          B- Withdraw
9          C- Transfer
10     3- In Main Test Your Data Type, By
11         Creating Two Variables, And The Transfer
12         Some Amount From One To Other.
13     4- Call Display For Each Instance.
14
15
16     Bonus:
17         Try To Make The Transfer Function
18         As Standalone Function, With Test Case.
19  */
20
```

## Answer

```
Start here X *main.cpp X
1  #include <iostream>
2  #include <stdio.h>
3  using namespace std;
4  struct account{
5      private:
6          int Number;
7          float balance;
8
9      public:
10         float Deposite (float _amount)
11         {
12             balance += _amount;
13             return balance;
14         }
15         bool withdraw(float _amount)
16         {
17             if (balance >= _amount)
18             {
19                 balance = balance - _amount;
20                 return true;
21             }
22         }
23         void transfer(account &b, float _amount){
24             if (this->withdraw (_amount) == true){
25                 b.Deposite(_amount);
26             }
27         }
28         void display()
29         {
30             cout << "balance" << balance << endl;
31         }
32     };
33 }
```

```
Start here X *main.cpp X
31 }
32 };
33 void transfer(account &a, account &b, float _amount){
34     if (a.withdraw(_amount)) //bonus
35         b.Deposite (_amount);
36 }
37
38 int main()
39 {
40     account a;
41     account b;
42
43
44
45     a.Deposite(3000);
46     b.Deposite(1000);
47
48     a.withdraw(200);
49
50     a.transfer(b ,300 );
51
52     a.display();
53     b.display();
54
55     transfer(a,b,100); //bonus
56     a.display();
57     b.display();
58
59
60     return 0;
61 }
62 }
```