## Interface

Interface is created by Inheritance. In C++, interface is actually made of ***pure virtual*** methods.

Eg: If class CFlyWithWings : public IFlyBehaviors, then

IFlyBehaviors\* flyBehaviors = new CFlyWithWings();

Invalid:

// Cannot new classes having pure virtual method(s)

ISubject\* pSubject = new ISubject;

An interface class should have NO member variable. It should only have pure virtual methods. But the concrete class that implements such interface can have other methods (those the interface has no def); these methods will be called within this concrete class (not called by other classes).

If in Beverage.h:

class Beverage

{

public:

Beverage();

~Beverage();

virtual std::string GetDescription() = 0;

But in HouseBlend.h:

class HouseBlend : public Beverage

{

public:

HouseBlend();

~HouseBlend();

// std::string GetDescription(); // Uncomment and the code will work!

then, WRONG syntax when Beverage\* pBeverage = new HouseBlend();

Reason: Abstract methods (pure virtual) must be overriden by concrete methods in inherited classes.

## Class Instance

**Delete class instance:**

Never: delete m\_pDuck;

Should: if (m\_pDuck) delete m\_pDuck;

Exception:

ISubject\* pSubject = NULL;

pSubject->RegisterObserver(pObserver); // NULL pointer cannot point to anything

Invalid:

ChocolateBoiler\* p;

p->IsEmpty(); // Error: uninitialized local variable 'ins' used

// Because the pointer points to a **random place** in the memory

Valid:

ChocolateBoiler i;

i.IsEmpty(); // Run well

// In this case, you won't create "pointer to i" object

// But you create "i" object directly. No need to allocate memory.

// Also, no need to free memory because it'll be done automatically

// when going out of scope since those are actual objects.

If class CurrentConditionDisplay: public ISubject, then:

Avoid: CurrentConditionDisplay\* pSubject = new CurrentConditionDisplay;

Should: ISubject\* pSubject = new CurrentConditionDisplay;

Instances (using pointers or not) of the same class are NOT data-related.

Eg1:

ChocolateBoiler i1;

ChocolateBoiler i2;

// IsEmpty() returns isEmpty - a class's private member variable initilized as TRUE.

cout << "before Fill(), isEmpty is " << i1.IsEmpty() << endl; // TRUE

i1.Fill();

cout << "After Fill(), isEmpty is " << i1.IsEmpty() << endl; // FALSE

cout << i2.IsEmpty() << endl; // TRUE

Eg2:

The same thing happens when declaring:

ChocolateBoiler\* p1 = new ChocolateBoiler;

ChocolateBoiler\* p2 = new ChocolateBoiler;