Subject: Object-Oriented Programming Question
Posted by Peter Stoltz on Wed, 17 Dec 1997 08:00:00 GMT
View Forum Message <> Reply to Message

> Hi everyone-

I have finally gotten around to trying out object-oriented programming
with IDL, and I have a question about data encapsulation.

I define an object structure A that has as one of its data members
another object B (A has a B). So far as I can tell, when one creates an

instance of A, one cannot invoke the methods of class B through the
syntax

IDL> a=obj_new('A')
IDL> a.b->some_method

% Object instance data is not visible outside class methods
% Execution halted

I got around this in the way one would with any other piece of member
data, by defining a method function in A that accesses the private data,
b.
Writing access methods is fine for most types of data, but I find it
pretty clumsy
when the member data happens to be another object. What I would like is
to have the
member data, b, defined as public. Before I write tons of code,
I wanted to
see if someone else had a better way of mocking up access to member data
class
methods as though that member class were public.

Thanks for any help. Also, is there a mailing list or anything
specifically about
object-oriented programming in IDL?

Peter

Subject: Re: Object-Oriented Programming Question
Posted by Peter Stoltz on Fri, 19 Dec 1997 08:00:00 GMT
View Forum Message <> Reply to Message

> There are mainly two ways to put toghether two (or more) classes:
>
> A) Composition: One component of a class is an Object.

> A 'Leg' is a part of an 'Animal':
> pro Animal__Define
> tmp = {ANIMAL, ..., LeftFrontLeg : OBJ_NEW(),...}
> ...
> end
> ...
> function Animal::Init
> ...
> self.LeftFrontLeg = OBJ_NEW('Leg')
> ...
> end
>
> B) Inheritance: One class is a special case of another one.
> A 'Reptil' is a kind of 'Animal':
> pro Reptil__Define
> tmp = {Reptil, ..., INHERITS ANIMAL}
> ...
> end
>
> [some stuff cut out]
>
> In your case, maybe what you need is to define Class A as a Subclass of
> Class B, i.e.,

But, using your example above, you would never inherit Animal from Leg.
You definitly want Leg as a member of the class Animal. An Animal
has a Leg, but an Animal is not a Leg, to use the "has a" versus
"is a" way of looking at it.

I don't think I want to tamper with this. In my case, I want class B
to be part of the member data of class A (not for A to inherit from B).
What I *really* want is from B to be public member data. You can define
public member data in C++, for instance. Then an instance of A would have
access to B's methods, and I wouldn't have to mess up the 'has a' versus
'is a' relationship.

So, my question is whether anyone has a good (i.e. not clumsy) way to
mock the technique of having public data members in a class when the
data member in question is another class. I did the obvious thing of
defining a member function in A that returns the class B, but then one has
to make an extra copy of the member class B every time you want to call
one of its methods via an instance of A. Plus, this way takes extra
lines of code, and aesthetically speaking, is not great.

Thanks for the answer, by the way, Evilio...

Subject: Re: Object-Oriented Programming Question
Posted by Evilio del Rio on Fri, 19 Dec 1997 08:00:00 GMT
View Forum Message <> Reply to Message

On Wed, 17 Dec 1997, Peter Stoltz wrote:

> I define an object structure A that has as one of its data members
> another object B (A has a B). So far as I can tell, when one creates an
> instance of A, one cannot invoke the methods of class B through the
> syntax
>
> IDL> a=obj_new('A')
> IDL> a.b->some_method
>
> % Object instance data is not visible outside class methods
> % Execution halted
>

There are mainly two ways to put toghether two (or more) classes:

A) Composition: One component of a class is an Object.
A 'Leg' is a part of an 'Animal':

```
pro Animal__Define
  tmp = {ANIMAL, ..., LeftFrontLeg : OBJ_NEW(),...}
  ...
end
...
function  Animal::Init
 ...
 self.LeftFrontLeg = OBJ_NEW('Leg')
 ...
end
```

B) Inheritance: One class is a special case of another one.
A 'Reptil' is a kind of 'Animal':

```
pro Reptil__Define
 tmp = {Reptil, ..., INHERITS ANIMAL}
 ...
end
```

As a general rule, in OOP the implementation (the components and some methods) of an object must be hiden from the user(*) of the Object. The user must only know how to use of what kind of things an object does. This is the "Need to Know" rule.
For example, to use a class representing Complex numbers you don't need to know if its internal representation is cartesian (x,y) or polar (r,theta) or any other convenient way. You just need to know that a complex is 'something' you can add, multiply, etc..., or calculate its module:

```
; Use (never changes)
 oZ = OBJ_NEW('Complex')
 ...
 r = oZ->Module()


; Polar Implementation (r, th)   ; Cartesian Implementation (x,y)
function Complex::Module | function Complex::Module
 return,self.R   |  return, SQRT(self.X^2 + self.Y^2)
end    | end
```

Doing it this way, the code that uses the Complex class is stable against implementation changes.

In your case, maybe what you need is to define Class A as a Subclass of Class B, i.e.,

```
 pro a__define
  tmp = {A ,...(whatever)..., INHERITS B }
  return
 end
```

and then use it as

```
 a->some_method (equivalent to a->b::some_method)
```

multiple inheritance is allowed (as far as no conflicting structure member definitions are found)

```
 pro a__define
  tmp = {A ,...(whatever)..., INHERITS B, INHERITS C }
  return
 end
```

Hope this helps.
Cheers,

(*) Note: The user of an Object is the rutine(s) that uses this object but is not a method of the Object class.

Evilio Jose del Rio Silvan Institut de Ciencies del Mar
E-mail: edelrio@icm.csic.es URL: http://www.bodega.org/
"Anywhere you choose,/ Anyway, you're gonna lose"- Mike Oldfield