Subject: Re: What about real polymorphism ??
Posted by Antonio Santiago on Thu, 09 Dec 2004 16:34:51 GMT
View Forum Message <> Reply to Message

Sorry, i think my english is little poor :)

Here are class C1 with a method called "datos" to print object data. It
prints: name and num.

C2 is a subclass of C1 and overrides "datos" method, that prints: name,
num and think.

In a language like java i can "cast" C2 object to a C1 object and
invoque "datos". The result is the execution of method "datos" of the C2
object.

How can i do the same with IDL? It posible in IDL to cast to superclass?

Example:

```
IDL> o1 = obj_new('c1')
IDL> o2 = obj_new('c2')
IDL> o1->datos
clase1
       1

IDL> o2->datos
class2
       2
thing 2
```

I want to "cast" o2 from C2 to C1 class and invoque "datos". A real
polymorphism "detects" that o2 really is an C2 object and that it had
overriden "datos" method and invoque it.
 --- ---

Really i have a class called VOLUM that draws some kind of data. My data
can be en cartesian or polar data, then my idea is to create two derived
clases VOLUM_CART and VOLUM_POLAR that overrides some methods of VOLUM
(for exmaple: "draw_data") and extend other news.

I want my application has some number of objects VOLUM, that can be
VOLUM_POLAR or VOLUM_CART. From application point of view they are only
VOLUM object. Then when executes the method "draw_data" depends of type
of object VOLUM (VOLUM_CART or VOLUM_DATA) i want IDL executes

VOLUM_CART::draw_data or VOLUM_POLAR::draw_data.

In Java, C++, ... it is easy but i think it is not possible in IDL.

Mmm... by other hand... while i write this message :) I supose that like
you say i can create an object array, assign different type object and
invoque the xxx method on every object.

The problem is that IDL can't brings me the possibility of abstract the
concept of VOLUM_CART and VOLUM_POLAR to a more generic class VOLUM.

To finshing, i think i answer my self :)

Thanks.

```
PRO c1__define

   struct = { $
     c1, $
     name: '', $
     num: 0 $
     }
END

FUNCTION c1::init

   self.name = 'clase1'
   self.num = 1
   return, 1
END

pro c1::datos
```

```
   print, self.name
   print, self.num
END




PRO c2__define

   struct = { $
     c2, $
     INHERITS c1, $
     thing: '' $
     }
END

FUNCTION c2::init

   r=self->c1::init()

   self.name = 'class2'
   self.num = 2
   self.thing = 'thing 2'
   return, 1
END

pro c2::datos
   print, self.name
   print, self.num
   print, self.thing
END
```

File Attachments

1) c1__define.pro, downloaded 69 times
2) c2__define.pro, downloaded 65 times