Subject: When should objects be used?
Posted by Richard G. French on Thu, 24 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Nearly all of my IDL programs started as interactive bits of code, saved
in
a journal file and then expanded into programs. The main power of IDL
for me is to be able to tinker with the code quite easily, try different
things out, and then finalize the code in the fashion that works best.
Most of these programs are the old-fashioned kind where you specify the
conditions at the start (the data to be
analyzed, the conditions of the analysis, the form of the result) and
then crank
through the processing in a fairly linear fashion, without a lot of
interaction along the way. In fact, when I produce PostScript files for
publications, I generally do
it by having a specific parameter file to go with each figure, such that
I can
reliably reproduce the figure exactly by running the code again. I had
found that
previously, I had often produced figures from highly interactive
programs but
had a hard time figuring out just exactly what was done to produce them.
All of this is by way of saying that this sort of programming seems to
be the opposite of object oriented code or widget programming, which
often seems to involve
lots of interactive choices along the way. Thus, I have never taken the
plunge to
learn object oriented programming, but I have a feeling that I am
missing the parade
and I'd like to know what I am missing.
Could some of you who might have started out doing things the way I do,
and
who have converted to object-oriented programming, tell me if there is
something to be gained by my revising my programming style to use
objects? I have developed a nice
library of IDL procedures and functions to do my kind of processing, but
I have not
done the same with widgets, and I have never tried doing it with
objects.
To provide a specific example, I have written some IDL code to
determine
the exact locations of small Saturnian satellites in Hubble Space
Telescope images,
to fit orbits to these positions, and to produce TeX tables of the final
fitted
results. Do you see obvious advantages to trying to do some of this
using objects?

Thanks for any insight you can provide. My problem is that so much of
the discussion of objects seems abstract that I have a hard time
figuring out how to
translate it to the problems I am trying to solve.

Dick French,
Astronomy Dept.
Wellesley College

Subject: Re: When should objects be used?
Posted by Struan Gray on Mon, 28 Jun 1999 07:00:00 GMT

View Forum Message <> Reply to Message

This thread seems to have split into two parts: IDL's own built-in object
routines and user-written objects. I think you either love or hate the
provided object graphics. I'm gaining more and more respect for them, but I
think that in general the provided routines are either too high-level or too
low-level, and reading the OpenGL specs I feel that RSI has unnecessarily
hamstrung their implementation. I like the way object graphics takes care of
any colour table juggling for me, and the way that data picking is made
substantially easier.

Some problems are intrinsically object-oriented. People have already
mentioned minor variations on a theme, such as multiple file-types for the
same sorts of data, but there are also data which are naturally hierarchical.
I have some crude routines for making and plotting crystal structures. Adding
atoms to a unit cell and then specifying a lattice and a bounding polygon is
an obvious OOP way to build up crystals. Because the objects keep track of
their own properties it is then easy to alter the size or colour of any the
individual atoms and the whole crystal instantly reflects the changes. It is
also easy to interact with the crystal: to add or remove bonds and
coordination polyhedra; to muck about creating point defects and lattice
distortions; and to add surface structures, adsorbates or the interface to
anther crystal. Toss in the fact that the whole thing can be rendered in 3D
with lights, perspective and depth-cueing and you have a pretty persuasive
arguement for OOP.

In IDL there are some language-based reasons for using objects. They are
persistent, globally available, and easier to clean up than rivals like
pointers. You can give novices templates which you know will work reliably
with the rest of your application, because you've hidden the dull but
necessary behaviour in a superclass that they can't edit. Best of all, you
don't *have* to use objects, so you can mix and match behaviour with normal
variables, widgets and the command line: you lose none of the prototyping
power of IDL.

The only downside is that objects give you a whole new way of writing

spaghetti code, since method definitions can be hidden in or distributed among
many-times abstracted superclasses. Also, I often get an uneasy feeling that
I'm the Sorcerer's Apprentice and the broomsticks are about to make their
autonomy felt, but being able to interact with and interrogate objects from
the command line helps calm me down.

Struan

Subject: Re: When should objects be used?
Posted by Martin Schultz on Mon, 28 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Richard G. French wrote:
>
> Nearly all of my IDL programs started as interactive bits of code, saved
> in
> a journal file and then expanded into programs. [...]

I've been thinking about using objects for a while now, but every time I
am ready for it there is something that has to be done right away so I
postpone again. From what I read about objects so far, I would think
that (perhaps) the most important
difference between object oriented (direct) graphics and old-style
sequential graphics programming is the decoupling between setting the
appearance of a plot and actually creating the plot (what David called
"sticky"). An old=style program might look something like this:

```
  pro myexcitingplot,data,color=color,line=line,...

  if (n_elements(color) ne 1) then color = 1
  if (n_elements(line) ne 1) then line = 0
  ; ...

  plot,data[0,*],data[1,*],color=color,line=line,...
  end
```

And you will have to store the color, line type etc externally and pass
it into this routine whenever you want to redraw the plot (e.g. on a
postscript printer). A little more sophisticated would be the use of a
stucture containing all the style parameters you want, e.g.

```
  thisstyle = { color:1, line:0 }
  myexcitingplot,data,thisstyle
```

where myexcitingplot would now "parse" the structure for relevant information:

```
   pro myexcitingplot,data,stru

   if (ChkStru(stru,"color")) then color=stru.color
   ; ... (this uses my ChkStru routine)
   plot,....

   end
```

In OOP you would instead write

```
   myexcitingplot->SetProperty,color=1,line=0
```

and you would want to store your data within the object as well (or at least a pointer):

```
   myexcitingplot->SetData,Data
```

and the Draw method would (re)produce your plot with the new properties:

```
   myexcitingplot->Draw
```

As long as you keep your object reference somewhere (and don't destroy the object ;-) you can repeat that Draw method anywhere and anytime you like. In the old-style approach you would have to keep the structure with all the plot parameters and the data instead which is more vulnerable to unforeseen changes. I would argue that it is possible to do (almost?) anything that you can do with objects without them as well, but it will occasionally be much more complicated. On the other hand I have a feeling that OOP requires more thinking in advance and lends itself less to a more experimental programming style -- but that may be a wrong impression because I am just not that familiar with it.

.... paused to think ;-) ....

But how about batch job processing? As mentioned before, you need to detroy objects after use shall they not leak your memory. Hence, if you want to process, say 1000 data files all in the same way your code would look like

```
  for i=0,n_elements(filename)-1 do begin
   thisplot = obj_new(filename[i])
   thisplot->SetProperty,.....
   set_plot,'PS'
   device,....
   thisplot->Draw
```

```
    device,/close
    obj_destroy,thisplot
  endfor
```

So not that much different from an old-style program. But the nice thing is you can reuse that object say within a widget program where you can mess around interactively. AND (what I think hasn't been mentioned intensively enough) you can inherit your object methods without having to rewrite everything. Thus, say myexcitingplot would produce a world map in a cylindrical projection, you could write myexcitingpolarplot which would inherit almost everything but take care of a few extra "features" that one needs to be aware of in polar plots (i.e. all that would have to change is your Draw method).

Hmm. That was long and maybe nothing new, but it helped me sort through some things (writing as psychotherapy ;-)

Cheers,
Martin

```
 |||||||||||||\\\\\\\\\\\\\-------------------//////////// //|||||||||||||
Martin Schultz, DEAS, Harvard University, 29 Oxford St., Pierce 109,
Cambridge, MA 02138          phone (617) 496 8318   fax (617) 495 4551
e-mail mgs@io.harvard.edu    web http://www-as/people/staff/mgs/
```

Subject: Re: When should objects be used?
Posted by davidf on Tue, 29 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith (jdsmith@astrosun.tn.cornell.edu) writes:

> Now David, I cannot sit idly by while you indict me on charges of
> misleading the uninitiated populace ;).

No, no. I think I accused you (falsely, I admit) of misleading *me*! :-)

> So my advice is: use whatever style of programming produces a program
> you can explain in 1 paragraph. Object oriented programming *is* easy.
> Especially in IDL. 10 minutes will suffice to learn the mechanics of
> it.

I couldn't agree with this more. The kinds of things I have been talking about lately *are* ridiculously easy. What is difficult for many people is the vocabulary of OOP. (I agree that calling something a "thingy" is not likely to help.) But there are only a handful of words you *really* have to

know. And if these can't be learned in 10 minutes someone
is not explaining them very well.

> However, in the great space of designs, that region labelled
> Obect-Oriented contains vastly more bad designs than good ones, compared
> to the more familiar regions.

Yes. This is a concern of mine too. As Struan was mentioning
yesterday, it *is* possible to develop great heaps of
spaghetti code if you are not careful. And I've found that
people who inherit some of my object code find it slow going
to digest it occasionally, although it seems extremely
straightforward to me. I can't tell yet if this is because
it is a new way of thinking for them or because these things
are inherently more difficult to learn to use. I suspect the
former, but I don't have enough experience using them to know
for sure.

I also haven't completely solved the problem of debugging
the darn things. If I put error handlers into the code too
soon, the darn things don't break when they should and an
error can be difficult to track down.

But I think most of these problems are problems that come
with anything new. It just takes some time and some experience
to learn how to use it properly.

> So, by all means, give it a try. You'll like it. Just
> be somewhat cautious before throwing all you eggs in that basket.

Good advice, there. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: When should objects be used?
Posted by J.D. Smith on Tue, 29 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Richard G. French (rfrench@wellesley.edu) writes:
>
>> My problem is that so much of the discussion of objects
>> seems abstract that I have a hard time figuring out how to
>> translate it to the problems I am trying to solve.
>
> To tell you the truth, I read JD's articles five or six
> times and I'm *still* not sure I have a clue. :-(
>
> Cheers,
>
> David
>
> P.S. Let's just say that object programming is a whole
> lot easier than a lot of people might lead you to believe. :-)
>

Now David, I cannot sit idly by while you indict me on charges of misleading the uninitiated populace ;). The framework I outlined really isn't specific to object-oriented programming, it's just most easily expressed as such. And it sounds complicated, not because it is, but because I haven't worked on it enough.

So my advice is: use whatever style of programming produces a program you can explain in 1 paragraph. Object oriented programming *is* easy. Especially in IDL. 10 minutes will suffice to learn the mechanics of it. However, in the great space of designs, that region labelled Obect-Oriented contains vastly more bad designs than good ones, compared to the more familiar regions.

Yes, OO is easy. Sometimes even easier than linear design. If you have modest goals of reuseability and interchange, it's not very different from other design styles. It can produce enticingly elegant solutions, or can lead you far astray of your original goals. So, by all means, give it a try. You'll like it. Just be somewhat cautious before throwing all you eggs in that basket.

JD

--
```
 J.D. Smith                            |*|      WORK: (607) 255-5842
 Cornell University Dept. of Astronomy  |*|           (607) 255-6263
 304 Space Sciences Bldg.              |*|      FAX: (607) 255-5875
 Ithaca, NY 14853                      |*|
```