

Re: [CLOS] Ensuring a method exists

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2006-10/msg00364.html>

- *From:* Pascal Costanza <pc@xxxxxxxx>
 - *Date:* Thu, 05 Oct 2006 23:09:08 +0200
-

Rahul Jain wrote:

Pascal Costanza <pc@xxxxxxxx> writes:

Rahul Jain wrote:

Equality is not an OO concept. You should not be using a g-f to determine equality.

Says who?

Says me. ;)

;)

But seriously. Equality is not just inherited. Note that I qualified my use of OO below. It's fine to compare objects, just not in a way that gets inherited by subclasses.

What would it mean for me to compare a point to a color-point? And I mean OO in the polymorphic sense, not the encapsulation sense.

Comparing a point to a color-point could have different meanings depending on context. I can imagine situations in which a default color is assumed for non-color points. I can also imagine situations in which the color is simply ignored. Even throwing a dedicated exception can be a well-defined behavior, especially with the condition handling system in CL.

Right, I tried to make that point before when I described what `_I_` would

Re: [CLOS] Ensuring a method exists

expect a point= function to do. Of course, all such discussions are mental masturbation because what really matters is what a specific application wants to compare. But I don't see how that implies that g-fs are the way to implement such (a) function(s). In fact, I would think that it would indicate that g-fs are the wrong way to do it because it implies that such operators can be arbitrarily augmented by other libraries. This may not result in expected behavior in all applications.

I think generic functions can be appropriate, but I agree with you that they are probably not sufficient the way they are.

I said "depending on context" for a reason: I think something like dynamically scoped functions or ContextL can be of use here. For example, the different cases could be captured in different ContextL-layers and then selected appropriately.

I haven't tried this yet for comparison functions, so take this with a grain of salt.

But if it turns out that it could work like that, that would actually be an acknowledgment of Christophe's position as well. OOP may not be appropriate for such tasks yet, but this could be because we still don't have OOP systems that are general enough. (That's not what he said, but that's roughly how I understand him. And it resonates with my point of view... ;)

There's a reason why the equality operators in CL are not g-fs.

The equality operators in CL are generic in the sense that they are applicable for several types. You just can't define methods on them (and even that is not a given, since the CL spec allows implementations to provide all plain functions as CLOS generic functions).

Right. I don't like equal or equalp for that exact reason. They're just a mishmash of other equality operators on specific types. There's no reason that specific combination should be preferred over any other. I consider them to be historical artifacts.

OK, that's indeed arguable.

Pascal

--

My website: <http://p-cos.net>

Common Lisp Document Repository: <http://cdr.eurolisp.org>

Closer to MOP & ContextL: <http://common-lisp.net/project/closer/>

.