

Re: A Design Problem

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2007-08/msg00130.html>

- *From:* Nick Keighley <nick_keighley_nospam@xxxxxxxxxxxxx>
 - *Date:* Fri, 24 Aug 2007 09:12:21 -0700
-

On 22 Aug, 23:24, "H. S. Lahman" <h.lah...@xxxxxxxxxxxxx> wrote:

Responding to Mousam...

I'm really asking H.S.Lahman questions because I'd like to improve my OO design skills (should be easy to improve on where I am now :-)

First of all forgive me for the length of this post.

I am developing one library for some text editor. In this library

I want to provide a set of C++ classes which will enable client (of the library) to add tool bars to the text editor. The requirements are:

- > Client can add as many tool bars as he wants to add.
- > In each toolbar there will be some tools. Tools can be of button

type or list box type like in MS word a tool bar can contain either buttons or list box (For simplicity consider these two types of tool only).

I took following approach to design the solution of above problem.

Re: A Design Problem

-> There will be a class say ToolBarMgr which will manage the collection of all tool bars.

Anything with 'manager' on the end makes me nervous because the vast majority of the time is it just a high level node in a procedural functional decomposition tree or it is some kind of god object with it fingers in everything.

-> There will be a class say Toolbar which will manage collection of all tools of this toolbar. This class will also contain information particular to a toolbar.

Now I am bit confused about the design of Tool class because there will be several functionalities which will be valid for button type of tool but does not make sense for list box type of tool and vice versa. Say e.g button tool can have one API to change the image of button but this API does not make any sense to have in list box tool.

I am not sure I see the problem. So far you have described:

```
[Window]
A
|R1
+-----+-----+----...
||
[MainWindow] [Dialog]
| 1
|
|R2
|
| contains
| 0..*
[Toolbar]
| 1
|
|R3
|
| contains
| * 1 represents R5 1
[Tool] ----- [Icon]
+ iconRelativePosition + modify()
+ isGrayedOut
...
A
|R6
+-----+----....
```

Re: A Design Problem

```
||  
[ListBox] [Button]  
+ elementCount + command()
```

seemed ok to me

There are two approaches which I can think of to design the Tool class.

1st Approach:

→ There will be a class called Tool which will contain information common to all type of tools. Say e,g it will contain toolbar ID to which this tool is associated.

OK.

→ There will be two more classes say ButtonTool and ListBoxTool which will inherit from Tool class and contain information specific to their type.

So far, so good.

→ Tool class will have all the methods which can be called on at least one type of tool and have the default implementation which will just throw some exception.

ah. that frightened me as well. I was thinking a do-nothing method instead was a good idea, but apparently not.

Not so good. The [Tool] is a superclass that identifies properties that /all/ tools share by definition. So every subclass /must/ provide a valid implementation of those properties and any client invoking the [Tool] property should be indifferent to which implementation is actually provided. It should not be possible to invoke a responsibility

Re: A Design Problem

a Tool does not have so the exception is unnecessary.

I suppose the question is why is his design trying to apply an inapplicable method to an object. Why try and change the image for a list box?

Would it be better for the GUI to ask the tool what it can do. Eg. right click the tool to get a list of applicable operations then you are never offered the option of doing something the tool can't do. Yeah I know I'm confusing GUI design with a discussion about basic OO principals

-> ButtonTool and ListBoxTool class will override their specific methods.

Implementation inheritance was one of those ideas that seemed like a good idea at the time but did not turn out well. It just opens up another dimension of foot-shooting with respect to LSP violations. Modern practice is to avoid implementation inheritance by making all superclass methods virtual whenever possible.

I think I've heard this before. Are there any good books that discuss modern OOD?

-> In this approach if client calls a method which is invalid for this tool object, he will get an exception.

If the client needs to invoke a method that only exists in a subclass, then the client needs to navigate a relationship that is directly to the subclass, not the superclass. Then the client can be sure that whoever is on the other end of the relationship has the needed properties. It is the responsibility of whoever instantiates that relationship to ensure that it is only instantiated to Tools having the right properties.

is what I said above a way of doing this?

<snip>

--

Nick keighley

Re: A Design Problem

Re: A Design Problem