

# A Design Problem

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.object/2007-08/msg00118.html>

---

- *From:* Mousam <[mousam.dubey@xxxxxxxxxx](mailto:mousam.dubey@xxxxxxxxxx)>
  - *Date:* Wed, 22 Aug 2007 07:26:13 -0000
- 

Hi All,

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.

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

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.
- > There will be two more classes say `ButtonTool` and `ListBoxTool` which

## A Design Problem

will inherit from Tool class and contain information specific to their type.

-> 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.

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

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

Main Advantages of this approach:

-> Client will always deal with Tool class. So, addition/deletion of any tool type will not affect existing client code.

Main Disadvantages of this approach:

-> Methods which do not make sense to be defined for some classes

will still be inherited from Tool class. Say e.g API to change the button image does not make sense to be in ListBoxTool class but still this method will get inherit from Tool class.

-> Addition/Deletion of tool type class will mean to add/remove some APIs from Tool class also.

2nd Approach:

---

-> There will be a class say Tool which itself will not be templated but will contain member templates similar to the class boost::any. This class will contain information which will be common for all type of tools. Also this class will contain the object of ButtonTool class or ListBoxTool class or anything.

-> Tool class will also contain one method say GetObject() which will also be templated and will return the object it contains. Before returning object the GetObject() method will make sure that it is returning correct type of object. If it can not return correct type of object it will throw some exception. So its implementation will be something like this

```
template <typename ValueType>
ValueType* Tool::GetObject()
{
    If(the ValueType
    typeid is same as that
    of type id of the
    object Tool class contains)
    {
        return address
        of the object
    }
}
```

## A Design Problem

```
else
{
throw some
exception
}
}
```

→ So now if client is having object of the Tool class and if he knows the exact type of tool; Tool class contains, he will call the GetObject() method of the Tool class to get the exact type of tool and then can call methods on that tool directly. But if the client does not know the exact type of tool; Tool class contains, still he can call the common methods (methods which are applicable to all type of tools).

Main Advantages of this approach:

→ Addition/Deletion of tool type class will not affect any class in the library. We just need to add new class for new tool type or delete the existing class for that tool type.  
→ Methods which are specific to a tool type will belong to its class only which makes sense.

Main Disadvantages of this approach:

→ Addition/Deletion of tool type class may affect client code bcoz now client code will deal with concrete classes.  
→ I am using Visual Studio6 as my IDE which has lots of bugs with the templates. So in some cases I have to use some ugly workarounds.

It would be great if some one can please suggest me as to which approach is better or can suggest me a new better approach.

Thanks a lot for your help and patience :)

.