Subject: subclassing LineEdit is ugly
Posted by hojtsy on Sun, 09 Apr 2006 19:46:24 GMT
View Forum Message <> Reply to Message

I am trying to subclass LineEdit, to create some kind of terminal window, where the user can only enter after the last character of the text. It seems quite much possible, but the resulting code would have lot of copy-paste from library code. It would be much easier to subclass LineEdit after the following desired modifications:

1) make these methods virtual, so that I can replace them:
 PlaceCaretNoG
 InsertChar
 DeleteChar
 Backspace
 AlignChar

2) Extract to a new virtual method this last part of LineEdit::Key, so that I can replace this code in a subclass.

```
if(IsReadOnly()) return false;
switch(key) {
case K_DELETE:
 DeleteChar();
 break;
case K_BACKSPACE:
 Backspace();
 break;
case K_SHIFT_TAB:
 AlignChar();
 break;
case K_CTRL_Y:
case K_CTRL_L:
 if(cutline) {
  CutLine();
  break;
 }
default:
 if(InsertChar(key, count, true))
  return true;
 return MenuBar::Scan(WhenBar, key);
}
return true;
```

Additionally please correct the error in the quoted code that WhenBar hotkeys are not working in read-only LineEdits.

Subject: Re: subclassing LineEdit is ugly
Posted by mirek on Sun, 09 Apr 2006 20:11:41 GMT
View Forum Message <> Reply to Message

hojtsy wrote on Sun, 09 April 2006 15:46I am trying to subclass LineEdit, to create some kind of terminal window, where the user can only enter after the last character of the text.

Well, I would rather solved that by putting LineEdit to readonly mode, used it as child of my real Ctrl and feeded characters from outside (using Insert/Remove). See how Console is done in TheIDE.... (but maybe look for pre-603 version, adding HYDRA support made it much more complicated).

Quote:
It seems quite much possible, but the resulting code would have lot of copy-paste from library code. It would be much easier to subclass LineEdit after the following desired modifications:

1) make these methods virtual, so that I can replace them:
PlaceCaretNoG
InsertChar
DeleteChar
Backspace
AlignChar

2) Extract to a new virtual method this last part of LineEdit::Key, so that I can replace this code in a subclass.

Ah, well, but where this should stop? Should we make all methods everywhere public and virtual?

Quote:
Additionally please correct the error in the quoted code that WhenBar hotkeys are not working in read-only LineEdits.
[/quote]

Hopefuly fixed.

Mirek

Subject: Re: subclassing LineEdit is ugly
Posted by hojtsy on Sun, 09 Apr 2006 21:29:16 GMT
View Forum Message <> Reply to Message

luzr wrote on Sun, 09 April 2006 16:11Ah, well, but where this should stop? Should we make all methods everywhere public and virtual? Lets not mix public and virtual, I was only asking for virtual. LineEdit is clearly made to be subclassed, because the fields and methods are protected

and not private. Providing a way to subclass a Ctrl but making it ugly doesn't seem to make sense. Are you trying to do speed optimization with keeping the inflexible non-virtual signatures? The selected methods are only called when a key is pressed, not 1000 times a second, so the minimal cost of virtual call would be unnoticable. I invested some time to find these methods of LineEdit where the behaviour could be most easily changed, not only for this Console thing, but other modifications of LineEdit.

Let me put an example. Somebody wants a LineEdit in which Shift-Tab unindents only if none of the selected lines are already fully unindented. It is clearly a subclassing situation: you are replacing only part of the behaviour. But AlignChar is non-virtual for whatever reason. So you need to do all these steps:
1) create a new method instead of AlignChar
2) override the method Key, copy-paste long code from LineEdit
3) call your new method from Key, instead of AlignChar
4) pray that no further subclasses, or methods of your subclass would accidentally call AlignChar instead of your new method
5) every time a new version of library comes out try to sync the changes to the copy-pasted Key method

Returning to your question whether all methods everywhere should be virtual: I think that complex library classes should be easy to subclass, not just possible, which means to me that
1) any non-speed-critical and non-trivial methods of complex classes should be virtual, and
2) long and complex methods implementing multiple aspects of the behaviour (such as the monster Paint in several Ctrls) should be broken up to multiple virtual methods, to enable overriding only one of them

My reasoning for this is that when you are developping an application yourself and need a subclass it is very easy for you to just make the needed method virtual in the base class, or just insert a branch in the library code itself. But for the clients of the library we are stuck with the amount of flexibility which is readily provided by the library. Imaging working in an environment where you can not change the library, but required to provide slighly different behaviour in some classes. This different working method places different requirements on the library, which may not be realized by you while working on one of your own applications.

Subject: Re: subclassing LineEdit is ugly
Posted by fudadmin on Sun, 09 Apr 2006 23:48:06 GMT
View Forum Message <> Reply to Message

hojtsy wrote on Sun, 09 April 2006 22:29luzr wrote on Sun, 09 April 2006 16:11Ah, well, but where this should stop? Should we make all methods everywhere public and virtual?
...
Returning to your question whether all methods everywhere should be virtual: I think that complex library classes should be easy to subclass, not just possible, which means to me that
1) any non-speed-critical and non-trivial methods of complex classes should be virtual, and
2) long and complex methods implementing multiple aspects of the behaviour (such as the monster Paint in several Ctrls) should be broken up to multiple virtual methods, to enable overriding only one of them

My reasoning for this is that when you are developping an application yourself and need a subclass it is very easy for you to just make the needed method virtual in the base class, or just insert a branch in the library code itself. But for the clients of the library we are stuck with the amount of flexibility which is readily provided by the library. Imaging working in an environment where you can not change the library, but required to provide slighly different behaviour in some classes. This different working method places different requirements on the library, which may not be realized by you while working on one of your own applications.

Yes, yes and yes.

Subject: Re: subclassing LineEdit is ugly
Posted by mirek on Mon, 10 Apr 2006 08:08:27 GMT
View Forum Message <> Reply to Message

Quote:
Let me put an example. Somebody wants a LineEdit in which Shift-Tab unindents only if none of the selected lines are already fully unindented. It is clearly a subclassing situation: you are replacing only part of the behaviour. But AlignChar is non-virtual for whatever reason. So you need to do all these steps:
1) create a new method instead of AlignChar
2) override the method Key, copy-paste long code from LineEdit
3) call your new method from Key, instead of AlignChar
4) pray that no further subclasses, or methods of your subclass would accidentally call AlignChar instead of your new method
5) every time a new version of library comes out try to sync the changes to the copy-pasted Key method

Well, let us discuss it:

1 - you have to add method anyway
2&3 - nope. Just override Key, react to Shift-Tab by calling your new method, for other keys call original LineEdit::Key.

```
bool MyLineEdit::Key(dword key, int count)
{
  if(key == K_SHIFT|K_TAB) {
     MyAlignChar();
     return true;
  }
  return LineEdit::Key(key, count);
}
```

4 - or pray that no other code does expect original AlignChar behaviour... See, by not makeing

some methods virtual, you enforce stronger contract - you guarantee single behaviour.

5 - nope because of 2.

Quote:
My reasoning for this is that when you are developping an application yourself and need a subclass it is very easy for you to just make the needed method virtual in the base class, or just insert a branch in the library code itself. But for the clients of the library we are stuck with the amount of flexibility which is readily provided by the library. Imaging working in an environment where you can not change the library, but required to provide slighly different behaviour in some classes. This different working method places different requirements on the library, which may not be realized by you while working on one of your own applications.

OK, I will think about it. However, all my experiences favor "black box" approach, with as narrow interfaces as possible. It means, if problem can be solved by composition rather than subclassing, I favor composition.

In this particular case (using LineEdit for terminal, I have no further description, so I may be wrong) I am pretty sure that things can be solved pretty easy without adding virtual methods. That said, if you want to push me into another direction, please provide more detailed description (adding characters at the end of console window only is too trivial to justice adding 10 virtual methods

Once again, it is not about speed concerns, but interface definition. I simply believe that for majority of methods it is much bettter when they are "final".

BTW, the real problem there is that AlignChar is protected. It should in fact be private...

Mirek

Subject: Re: subclassing LineEdit is ugly
Posted by hojtsy on Mon, 17 Apr 2006 21:38:28 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 10 April 2006 04:08

```
bool MyLineEdit::Key(dword key, int count)
{
  if(key == K_SHIFT|K_TAB) {
     MyAlignChar();
     return true;
  }
  return LineEdit::Key(key, count);
}
```

Fine! Now let's discuss this solution. When you are duplicating part of a method just to replace a call in it, you can make two mistakes. Mistake 1 is failure to reproduce the context of the original

call you are replacing. This was an easy case and you was familiar with it, yet you made this mistake. Now imagine this being attempted by somebody less familiar with library code AND less skilled than you, attempting the same. The error is that this overloaded method invokes MyAlignChar even if the LineEdit is read only. Another problem is that in the future the code you duplicated could change. In fact I hope that sometime it will change to configurable hotkeys. When such change happens you need to maintain, and change the duplicate version too. With lots of such duplication it becomes impossible to monitor all original locations for any possible change in the future.

luzr wrote on Mon, 10 April 2006 04:08pray that no other code does expect original AlignChar behaviour... See, by not makeing some methods virtual, you enforce stronger contract - you guarantee single behaviour.Expecting the default AlignChar behaviour is like expecting the default LeftDouble behaviour. Or the default behaviour of Key method for Shift-Tab as input paremeter. The point is that it is not a wise expectation. When you are calling AlignChar you mean that you want the selected text unindented. When I override this method in a subclass I am redefining what "unindent" means, not what Shift-Tab does. It is a conceptual difference - I would like my method called whenever "unindent" is needed, even if that is not when Shift-Tab is pressed. With this method as virtual I can choose from redefining "unindent" by overriding AlignChar, or changing the action triggered by Shift-Tab in writable mode by overriding Key.

luzr wrote on Mon, 10 April 2006 04:08adding characters at the end of console window only is too trivial to justice adding 10 virtual methods
So should every client of your library ask for each method to be virtual each time the need arises? That will be long story. And then the client need is weighted against what cost? The only cost is minimal CPU overhead. Almost any user benefit seem to justify that cost. BTW it is 6 virtual methods not 10, and they are not added just made virtual to enable more elegant subclassing.

Subject: Re: subclassing LineEdit is ugly
Posted by mirek on Mon, 17 Apr 2006 22:15:44 GMT
View Forum Message <> Reply to Message

hojtsy wrote on Mon, 17 April 2006 17:38
The only cost is minimal CPU overhead.

I do not care about virtual method overhead here (not a bit). But I am very concerned about fuzzy interfaces.

One problem with this approach is: where are limits?

String::Cat ?
Ctrl::HSizePos ?
DrawLabel::GetSize ?

Another problem is that introducing too much virtual methods often exposes too much of implementation details.

Now about our specific example: I see LineEdit as "blackbox" with defined (and final) operations, which has "default interface" represented by current "Key" method and default menu.

If you want more than "default interface", simply change the interface part (that one is well defined in Ctrl).

Mirek
