Subject: Controls & classes design questions
Posted by Mindtraveller on Wed, 15 Aug 2007 23:19:42 GMT
View Forum Message <> Reply to Message

First of all, thanks for such a promising framework.

My question comes from very common task.

My goal is to write an editor for some kind of graphics files. As for modern standard, editor is to be multi-tabbed, working with multiple files simultaneously.
It`s OK, I just added TabCtrl to main window.
Next move is to resolve design approach. My decision was to create a class representing single file in editor. This class contains main window Tab control`s tab, representing this file, and a GLCtrl object.
Something like this:

```
class MyFile
{
   MyFile (TabCtrl &tabs) { tab = tabs.Add("..."); }

   GLCtrl        gl;
   TabCtrl::Item &tab;
};
//...
class MainWindow
{
   TabCtrl       tabs;
   Vector<MyFile> files;
};
//...
void MainWindow::OnNew()
{

   files.Add(MyFile(tabs));
}
```

So my questions are:

1) Is this an optimal solution for decomposition controls on such a task?

2) I`ve read articles about moveable for a number of times, and just can`t understand, why application doesn`t crash since new MyFile object existed only within MainWindow::OnNew() scope?

3) Class MyFile contains GL control which has virtual methods for sure. Is it still moveable? How could that be?

4) What kind of copy constructor should I implement? What to do with gl and tab fields?

Thanx in advance.

Subject: Re: Controls & classes design questions
Posted by mrjt on Thu, 16 Aug 2007 09:46:55 GMT
View Forum Message <> Reply to Message

Hello and welcome,

I think your design is mixing up access to controls and data at different levels and will probably lead to complications later. For instance: why does MyFile need access to the TabCtrl? Shouldn't that be managed by the MainWindow?

Personally I would do it something like this:

```
class MyFile : public GLCtrl {
    MyFile();

    virtual void GLPaint();
    //...
};
//...
class MainWindow : public TopWindow
{
    TabCtrl       tabs;
    Array<MyFile> files;
};
//...
void MainWindow::OnNew()
{

    MyFile &file = files.Create<MyFile>();
    file.SizePos();
    tabs.Add(file, "filename");
}
```

You need to inherit from GLCtrl and overload GLPaint to be able to draw anything.

I'll try and answer your other questions about Moveable, but I'm not 100% so if I get something wrong hopefully someone will correct me:
2) I'm confused about this, because MSC8 won't even compile the code you posted, let alone run it.
3) No Ctrl or derived class is Moveable because they contain internal pointers to other Ctrls. However, what is not clear from the docs is that virtual methods are Moveable, but abstract methods are not (ie. virtual void Abstract() = 0; is not allowed). Array can store non-moveable types, but Vector et al. cannot.
4) As far as I can see you shouldn't need a copy-constructor.

Hope that helps,
James

Subject: Re: Controls & classes design questions
Posted by unodgs on Thu, 16 Aug 2007 10:06:19 GMT
View Forum Message <> Reply to Message

Quote:
You need to inherit from GLCtrl and overload GLPaint to be able to draw anything.

That's not true. GLCtrl call WhenGLPaint to which you can assign your painting routine:

```
GLCtrl gl;
gl.WhenGLPaint = THISBACK(MyGLPaint);
```

Subject: Re: Controls & classes design questions
Posted by mrjt on Thu, 16 Aug 2007 10:16:25 GMT
View Forum Message <> Reply to Message

I stand corrected

Subject: Re: Controls & classes design questions
Posted by Mindtraveller on Thu, 16 Aug 2007 12:00:54 GMT
View Forum Message <> Reply to Message

mrjt wrote on Thu, 16 August 2007 13:46I think your design is mixing up access to controls and data at different levels and will probably lead to complications later. For instance: why does MyFile need access to the TabCtrl? Shouldn't that be managed by the MainWindow?
I rearranged structure and now MyFile has local tab object only. Main TabCtrl object access is needed only for creating and destroying specific file in editor (adding & removing tabs from control). And this adding & removal should be done inside MyFile class, I suppose.
For now, I have MyFile::ctor and MyFile::Close using TabCtrl (but not storing it`s reference anymore).

mrjt wrotePersonally I would do it something like this:

```
class MyFile : public GLCtrl {
    MyFile();

    virtual void GLPaint();
    //...
};
```

You need to inherit from GLCtrl and overload GLPaint to be able to draw anything. I agree, in this case inheritance is better than inclusion. Thanks.

mrjt wroteclass MainWindow : public TopWindow

```
{
    TabCtrl       tabs;
    Array<MyFile> files;
```

```
};
//...
void MainWindow::OnNew()
{

   MyFile &file = files.Create<MyFile>();
   file.SizePos();
   tabs.Add(file, "filename");
}
...
```

2) I'm confused about this, because MSC8 won't even compile the code you posted, let alone run it.
It compiled, actually. There even was no rintime error(!).
I`m shocked too.
mrjt wrote
3) No Ctrl or derived class is Moveable because they contain internal pointers to other Ctrls. However, what is not clear from the docs is that virtual methods are Moveable, but abstract methods are not (ie. virtual void Abstract() = 0; is not allowed). Array can store non-moveable types, but Vector et al. cannot.
Wow... it`s just unclear moment. Suppose author should focus on it in manual. Proposing new conatiner classes and more important, new approach of handling with containers and elements - this must be explained much more than 2 articles. Examples must also be included. U++ is great, but it is unusual and complicated in moments, so good documentation is greatly needed.
mrjt wrote4) As far as I can see you shouldn't need a copy-constructor.
Hope that helps,
James If my class isn`t moveable, then of course not.
Thanks, James!

Subject: Re: Controls & classes design questions
Posted by mirek on Fri, 17 Aug 2007 22:14:32 GMT
View Forum Message <> Reply to Message

mrjt wrote on Thu, 16 August 2007 05:46
3) No Ctrl or derived class is Moveable because they contain internal pointers to other Ctrls. However, what is not clear from the docs is that virtual methods are Moveable, but abstract methods are not (ie. virtual void Abstract() = 0; is not allowed). Array can store non-moveable types, but Vector et al. cannot.

Actually, classes with virtual methods are explicitly NOT MOVEABLE.

http://www.ultimatepp.org/srcdoc$Core$Moveable$en-us.html

Note that while with both MSC and GCC on x86, technically there is a little problem, we thought it is better to stay on the safer side, while "moving" objects with virtual methods makes only a little benefit.

Mirek

Subject: Re: Controls & classes design questions
Posted by mrjt on Sun, 19 Aug 2007 09:22:38 GMT
View Forum Message <> Reply to Message

Then I must be misunderstanding something. I would be grateful if you could tell me why the following code works if virtual methods are non-moveable:

```
struct BaseClass
{
 BaseClass() { int1 = 1; }

 int int1;
 virtual int  GetInt()   { return int1; }
 virtual String  GetString()  { return "A String"; }
};

struct DerivedClass : public BaseClass, public Moveable<DerivedClass>
{
 DerivedClass() { int2 = 99; }

 int int2;
 virtual int  GetInt()   { return int2; }
 virtual String  GetString()  { return "This is a derived class"; }
};

GUI_APP_MAIN
{
 Vector<DerivedClass> v;

 v.Add(DerivedClass());
 v.Add(DerivedClass());

 for (int i = 0; i < v.GetCount(); i++)
  PromptOK(Format("Int: %d String: %s", v[i].GetInt(), v[i].GetString()));
}
```

James

Subject: Re: Controls & classes design questions
Posted by mirek on Sun, 19 Aug 2007 09:51:16 GMT
View Forum Message <> Reply to Message

mrjt wrote on Sun, 19 August 2007 05:22Then I must be misunderstanding something. I would be

grateful if you could tell me why the following code works if virtual methods are non-moveable:

```
struct BaseClass
{
 BaseClass() { int1 = 1; }

 int int1;
 virtual int  GetInt()   { return int1; }
 virtual String  GetString()  { return "A String"; }
};

struct DerivedClass : public BaseClass, public Moveable<DerivedClass>
{
 DerivedClass() { int2 = 99; }

 int int2;
 virtual int  GetInt()   { return int2; }
 virtual String  GetString()  { return "This is a derived class"; }
};

GUI_APP_MAIN
{
 Vector<DerivedClass> v;

 v.Add(DerivedClass());
 v.Add(DerivedClass());

 for (int i = 0; i < v.GetCount(); i++)
  PromptOK(Format("Int: %d String: %s", v[i].GetInt(), v[i].GetString()));
}
```

James

Sure it does work - at least with GCC and MSC.

The problem is that C++ standard allows memcpy only for POD types. U++ extends this to moveable types, but that is technically violating C++ standard - such thing is undefined by standard.

Therefore we try to keep such extension as thin as possible. There is really not much practical advantage to having types with vtable moveable and in the same time, it is not that much unlikely that some compiler would implement virtual methods in a way that would not work with U++.

Subject: Re: Controls & classes design questions
Posted by HenryXin on Wed, 12 Mar 2008 07:48:20 GMT
View Forum Message <> Reply to Message

I also read these code.

This means the struct Moveable actually provide a moveable interface with NTL, doesn't provide the valid check method? Is right?
Thanks.

Subject: Re: Controls & classes design questions
Posted by mirek on Wed, 12 Mar 2008 13:34:22 GMT
View Forum Message <> Reply to Message

HenryXin wrote on Wed, 12 March 2008 03:48I also read these code.
This means the struct Moveable actually provide a moveable interface with NTL, doesn't provide the valid check method? Is right?
Thanks.

Not quite sure I understand....

There is no way how to check moveabitity by either compiler or in runtime. Therefore type has to be marked as moveable by programmer, by inheriting from Moveable.

Actually, the alternative would be to avoid Moveable completely, but that still feels a little bit risky. Besides, all non-moveable types can still be stored in Array (including objects with virtual methods etc....)

Mirek
