

Assumptions in our last lecture

Lecture 12 Adapters

In the last lecture, we introduced the concepts of inheritance and polymorphism, and use it to make an abstract class that can deal with multiple types.

Sometimes that mechanism does not work well. This happens if the source of the library is not readily available. We will see what we can do in such situations.

References:

- 3rd Edition: Section 15.2, Section 24.3.

PMOOP(0396A)

PMOOP(0396A)-12.1

What we actually need

Before going any further, let's check what we really need. We want the following code to work:

```
int main(int argc, char *argv[]) {
    SoundOutput *so = 0;
    if (argc == 1)
        so = new OSS_CLASS;
    else
        so = new WavFile(argv[1]);
    ...
}
```

Of course, it is okay for us to use a name other than *OSS* for *OSS_CLASS*. The only point is that the "so" pointer must be able to handle both a *OSS* class and a *WavFile* class.

Thus ***OSS_CLASS* must be derived from *SoundOutput***, no compromise!

PMOOP(0396A)-12.2

PMOOP(0396A)-12.3

One solution

- Now, *OSS_CLASS* must be derived from *SoundOutput*, while we cannot make *OSS* a derived class from *SoundOutput*.
- That is, *OSS_CLASS* and *OSS* must be two different classes.
- But is it just the solution? We just make a new class *OSS_CLASS*, different from *OSS*. Within *OSS_CLASS*, we call the functions of *OSS* to achieve all the needed operations.
- In general, we call *OSS_CLASS* an adapter class. It adapts the class *OSS* into the *SoundOutput* interface. Let's just call the new class *OSSAdapter*.
- To do this, *OSSAdapter* defines a data member of class *OSS*. The *OSS* instance is created when we create the *OSSAdapter* instance.
- In *OSSAdapter*, we write methods to redirect requests to the *OSS*-class data member.

OSSAdapter, version 1 (lecture-12-a)

```
#include "SoundOutput.h"
#include "OSS.h"
class OSSAdapter: public SoundOutput {
public:
    OSSAdapter() {};
    void set_sample_rate(int sr);
    int get_sample_rate();
    ...
private:
    OSS driver;
};
```

```
#include "OSSAdapter.h"
void OSSAdapter::set_sample_rate(int sr) {
    driver.set_sample_rate(sr); // Delegate to driver
}
int OSSAdapter::get_sample_rate() { return driver.get_sample_rate(); }
...
```

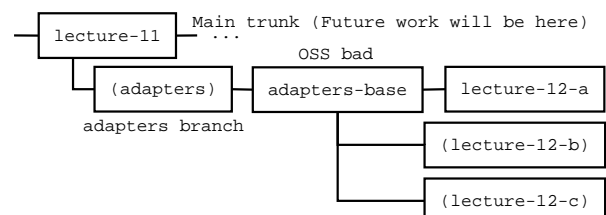
PMOOP(0396A)-12.4

PMOOP(0396A)-12.5

About the source

For your own study, it is the best if we have a completed program rather than just a small fragment of it, so that you can play with it.

A CVS repository can be found in the source of this lecture, so go checkout "note2wav" from it. It is structured like this.



PMOOP(0396A)-12.4

PMOOP(0396A)-12.5

There's More Than One Way To Do It: take 1

Interestingly, there are at least two other ways to achieve the same effect, and the solution we have seen is the **least** popular among the three.

Alternative 1

- Instead of an OSS data field, we can have a OSS* data field.
- **Benefit:** With *driver* being a pointer, it has polymorphic behaviour. So this allows *OSSAdapter* to use any derived class of OSS, and also to create a new OSS object in the middle.
- **Benefit:** This also insulates our program from any changes to the OSS class: The header file of the OSSAdapter only need to know that OSS is a class, without knowing the exact structure. Only OSSAdapter.o need to be recompiled, the main program don't.
- **Cost:** The cost is one extra level of pointer, i.e., slower to call functions.

PMOOP(0396A)-12.6

OSSAdapter: version 2 (lecture-12-b)

```
#include "SoundOutput.h"
class OSS; // Declare a class
class OSSAdapter: public SoundOutput {
public:
    OSSAdapter();
    void set_sample_rate(int sr);
    ...
private:
    OSS *driver;
};
```

```
#include "OSSAdapter.h"
#include "OSS.h" // Only in implementation
OSSAdapter::OSSAdapter() { driver = new OSS; }
void OSSAdapter::set_sample_rate(int sr) {
    driver->set_sample_rate(sr); // Delegate to driver
}
...
```

PMOOP(0396A)-12.7

There's More Than One Way To Do It: take 2

For non-polymorphic classes like the original OSS, another implementation prevails.

Alternative 2

- Instead of an OSS data field, *OSSAdapter* can derive from OSS.
- Thus *OSSAdapter* is derived from **both** *SoundOutput* and OSS. This is called multiple inheritance, and is allowed in C++.
- An *OSSAdapter* object thus have member functions of both classes, and inherits the member functions of both classes.
- The derivation from OSS is done in **private**, so that the user cannot see it. The user only see that *OSSAdapter* is a sub-class of *SoundOutput*.
- **Benefit:** Since *OSSAdapter* is derived from OSS, this allows *OSSAdapter* to easily override operations of OSS.

PMOOP(0396A)-12.8

OSSAdapter: version 3 (lecture-12-c)

```
#include "SoundOutput.h"
#include "OSS.h"
class OSSAdapter: public SoundOutput, private OSS {
public:
    OSSAdapter() {};
    void set_sample_rate(int sr);
    int get_sample_rate();
    ...
};
```

```
#include "OSSAdapter.h"
void OSSAdapter::set_sample_rate(int sr) {
    OSS::set_sample_rate(sr); // Delegate to OSS
}
int OSSAdapter::get_sample_rate() { return OSS::get_sample_rate(); }
...
```

PMOOP(0396A)-12.9

Conclusion

- If we can use a class maintained by another programmer, typically we don't want to modify the code.
- If we want to modify the interface of such a class, we cannot do it directly. For example, we cannot rename the member functions, and we cannot turn some non-virtual classes to virtual or vice-versa.
- Instead, we can make an adapter class to adapt the interface.
- Adapter classes are very simple and common. However, to make the best design, we still need to consider many issues.

Exercise

What is the circumstances under which an adapter can be used? Find all your assumptions.

PMOOP(0396A)-12.10