University of Hertfordshire

7COM1025
Programming for Software Engineers
Lecture 10

Dr. Renato C. de Amorim

THE ONE HUNDRED UNDER FIFTY

PROXY (STRUCTURAL PATTERN)

Provides a one-to-one forwarding interface to another class.
It is useful to modify the behaviour of the Original class while still preserving its interface.
The above is particularly true if the Original class is third-party and hence not easily modifiable directly.

The proxy class and the original class have the same interface.
FunctionA() in the proxy class will call FunctionA() in the original class

Generally the proxy class has a copy (or more often a pointer) to the original class.

Advantages
- It allows to implement lazy instantiation of the original object. The original object is not instantiated until it is actually needed. This is particularly useful if the instantiation consumes a lot of resources.
- Implements access control to the Original object. You can verify the the user has permission to access the method in the original object.
- Supports debug. It allows you to insert debugging statements into the proxy methods.
- Make the Original class thread safe.
- Support resource sharing. Multiple proxy objects could share the same underlying original class.
- Protects against changes in the original class.

A downside of this technique is the need to re-expose functions in the original object (some code duplication)

ADAPTER (STRUCTURAL PATTERN)

It translates the interface for one class into a compatible but different interface.
Similar to the proxy pattern (a single component wrapper), however, the interface of the adapter class and the original class may be different.

This pattern is useful to expose a different interface for an existing API to allow it to work with other code.

Advantages
- Enforce consistency across your API.
- Wrap a dependent library of your API.
- Transform data type. If your dependent library needs angles in degrees as input, you can write an adaptor that receives angles in radians, converts them to degrees and calls the dependent library.
- Expose a different calling convention for your API. You can write a C++ adaptor to wrap code in C (no classes).

As in the proxy pattern, you need to be careful with code duplication.

FAÇADE (STRUCTURAL PATTERN)

Presents a simplified interface for a larger collection of classes.
It defines a higher-level interface that makes the underlying subsystem easier to use.

Facade is different from Adapter. The former simplifies a class structure while the latter maintains the same class structure.

A Facade might provide an improved interface while still allowing access to the underlying subsystem (it is not necessary to provide access to the underlying classes).

Advantages
- Hide legacy code.
- Create convenience APIs.
- Support reduced or alternate functionality.

OBSERVER (BEHAVIOURAL PATTERN)

It is very common for objects to call methods in other objects.

- In fact any non-trivial task normally requires several objects collaborating together.
- An object A must know about the existence and interface of an object B in order to call its methods.
- The above introduces a compile-time dependency between A and B, forcing them to become tightly coupled.

eg. Model – View – Controller (MVC) is an architectural pattern which requires the isolation of the business logic (the model) from the user interface (the view). The controller receives the user input and coordinates the other two.

Benefits

- Segregation of Model and View components makes it possible to implement several user interfaces that reuse the common business logic core.
- Duplication of low-level Model code is eliminated across multiple US implementations.
- Decoupling of Model and View code results in an improved ability to write unit tests for the core business logic code.
- Modularity of components allows core logic developers and GUI developers to work simultaneously without affecting each other.

Note: View can call Model (so it can show its state), but Model cannot call View.
The observer pattern is a specific instance of the Publish/Subscribe paradigm.

ITERATOR (BEHAVIOURAL PATTERN)

Provides a way to access the elements of an aggregate object sequentially.

You have seen it before!

```
for(vector<int>::iterator item =vec.begin(); item!=vec.end();item++)
    cout<<*item<<' '
```

It decouples algorithms from containers.

You can sort the values of a vector with:

```
sort(myvector.begin(), myvector.end())
```

Sort is defined in the namespace std from #include <algorithm>

EXERCISE

In small group choose one design pattern from each of the main categories (creational, Structural and behavioural) and provide an example of when it would be beneficial to use the chosen design pattern.