Design Patterns in C++: Structural Façade to Proxy

FAÇADE

Dmitri Nesteruk
QUANTITATIVE ANALYST

@dnesteruk http://activemesa.com

Course Overview

Third course in a series of courses on C++ Design Patterns

- Covers 2nd half of structural design patterns (Façade to Proxy)

Covers every pattern from GoF book

- Motivation
- Classic implementation
- Pattern variations
- Library implementations
- Pattern interactions
- Important considerations (e.g., testability)

Patterns demonstrated via live coding!

Demo

Uses modern C++ (C++11/14/17)

Demos use Microsoft Visual Studio 2015, MSVC, ReSharper C++

Some simplifications:

- Classes are often defined inline (no .h/.cpp separation)
- Pass by value everywhere
- Liberal import of namespaces (e.g., `std::`) and headers

Course Structure

Façade

Flyweight

Null Object

Proxy

Overview

Motivation

Scenario

Façade

Balancing complexity and presentation/usability

Typical home

- Many subsystems (electrical, sanitation)
- Complex internal structure (e.g., floor layers)
- End user not exposed to internals

Same with software!

- Many systems working together provide flexibility, but…
- API consumers want it to ‘just work’

Façade

Provides a simple, easy to understand/use interface over a large and sophisticated body of code.

Make a library easier to understand, use and test

Reduce dependencies of user code on internal APIs that may change

- Allows more flexibility in developing/refactoring the library

Wrap a poorly designed collection of APIs with a single well-designed API

Summary

Build a Façade to provide a simplified API over a set of classes

May wish to (optionally) expose internals though the façade

May allow users to 'escalate' to use more complex APIs if they need to