6.S096 Lecture 6 – Design Patterns

Higher-level program design

Andre Kessler
Outline

1. Code Review
2. Design Patterns
3. Wrap-up
Don’t overuse this->

- No need for `this->_member`, just write `_member`
- (that’s why we use a leading underscore - to distinguish member variables)

How not to do it:

```cpp
void Rational::normalize() {
    auto abs_num = std::abs( this->num );
    auto abs_den = std::abs( this->den );
    auto theSign = this->sign();
    // ..etc, we don’t need ‘this’!
}
```
Don’t overuse this->

- No need for this->_member, just write _member
- (that’s why we use a leading underscore - to distinguish member variables)

Much better:

```cpp
void Rational::normalize() {
    auto abs_num = std::abs(_num);
    auto abs_den = std::abs(_den);
    auto theSign = sign();
    // ..etc, ^^^ better
}
```
Scope issues

SomeClass::whatIsThis()?
What are design patterns?

- “Distilled wisdom” about object-oriented programming
- Solutions to common problems that arise
- Anti-patterns: bad solutions to common problems that arise.
Gang of Four (GoF)

Image of book cover removed due to copyright restrictions.
The 23 standard patterns

DESIGN PATTERN RELATIONSHIPS

Builder
- Creating Composites
- Changing skin versus guts

Decorator
- Adding responsibilities to objects
- Changing skin versus guts

Strategy
- Sharing strategies
- Defining algorithm’s steps

Flyweight
- Sharing states
- Sharing terminal symbols

Composite
- Defining traversals
- Adding operations
- Sharing composites

Memento
- Saving state of iteration

Adapter
- Avoiding hysteresis

Command
- Composed using

Observer
- Complex dependency management

Proxy
- Defining the chain

Bridge
- Defining the chain

Chain of Responsibility
- Adding responsibilities to objects

Facade
- Single instance

Abstract Factory
- Implement using

Template Method
- Often uses

Visitor
- Defining grammar

Interpreter
- Enumerating children

State
- Sharing states

Mediator
- Defining the chain

Composite
- Adding operations

Strategy
- Defining the chain

Singleton
- Single instance

Prototype
- Configure factory dynamically

Image by MIT OpenCourseWare.
Source: Gamma, Erich, Richard Helm, et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.
We’ll be covering:

- Strategy (behavioral)
- Composite (structural)
- Factory Method (creational)
class IndexingScheme {
public:
    virtual size_t idx( size_t r, size_t c ) = 0;
    virtual ~IndexingScheme() {};
};

class RowMajor : public IndexingScheme {
    size_t _nCols;
    RowMajor() = delete;
public:
    RowMajor( size_t, size_t numCols ) :
        _nCols{numCols} {}
    size_t idx( size_t r, size_t c ) {
        return c + r * _nCols;
    }
};
class ColMajor : public IndexingScheme {
    size_t _nRows;
    ColMajor() = delete;

public:
    ColMajor( size_t numRows, size_t ) :
        _nRows{numRows} {}
    size_t idx( size_t r, size_t c ) {
        return r + c * _nRows;
    }
};

Let’s look at the example code...
Composite Pattern

**Component**

- operation()

**Leaf**

- operation()

**Composite**

- operation()
- add()
- remove()
- getChild()
Composite Pattern

We’ll consider the example of a file system.

- Need to represent directories and files
- Directories can contain other files or directories
- Files are “leaf” nodes, probably contain pointers to data.
- This example will also use the factory pattern.
Composite Pattern

class Node {
    public:
        virtual ~Node() {}
        virtual Directory* getDirectory() { return nullptr; }
        // ...etc
};
class Directory : public Node {
    std::string _name;
    std::vector<NodePtr> _child; // ...etc
    public: // ...etc
        virtual Directory* getDirectory() { return this; }
        void add(NodePtr item) { _child.push_back(item); }
        static NodePtr create(const std::string &dirname);
};
Composite Pattern

// the ‘‘leaf’’ class
class File : public Node {
    std::string _name;
    File() = delete;
    void lsIndented( int indent ) const;

public:
    File( std::string filename ) : _name(filename) {}
    void ls() const;

    static NodePtr create( const std::string &filename );
};

Let’s look at the example code...
Let’s see some examples...
Wrap-up & Friday

Second assignment due tonight at midnight

Third assignment (small) due Saturday at midnight

Class on Fri.

Will cover …

- Design patterns and anti-patterns

Questions?

- Office hours Mon, Tues
6.S096 Effective Programming in C and C++
IAP 2014

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.