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:

```
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:

```
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.
Reference: Gamma, Erich, Richard Helm, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

The 23 standard patterns

DESIGN PATTERN RELATIONSHIPS

Memento
Adapter
Proxy
Bridge
Builder
Creating Composites
Saving state of iteration
Avoiding hysteresis
Iterator
Command
Enumerating children
Defining traversals
Composed using
Decorator
Adding responsibilities to objects
Composite
Sharing composites
Adding operations
Defining the chain
Changing skin versus guts
Flyweight
Visitor
Chain of Responsibility
Defining grammar
Strategy
Sharing strategies
Sharing terminal symbols
Interpreter
Adding operations
Sharing states
Mediator
Complex dependency management
Observer
State
Defining algorithm's steps
Template Method
Often uses
Factory Method
Prototype
Configure factory dynamically
Abstract Factory
Implement using
Single instance
Singleton
Single instance
Facade

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)

Strategy

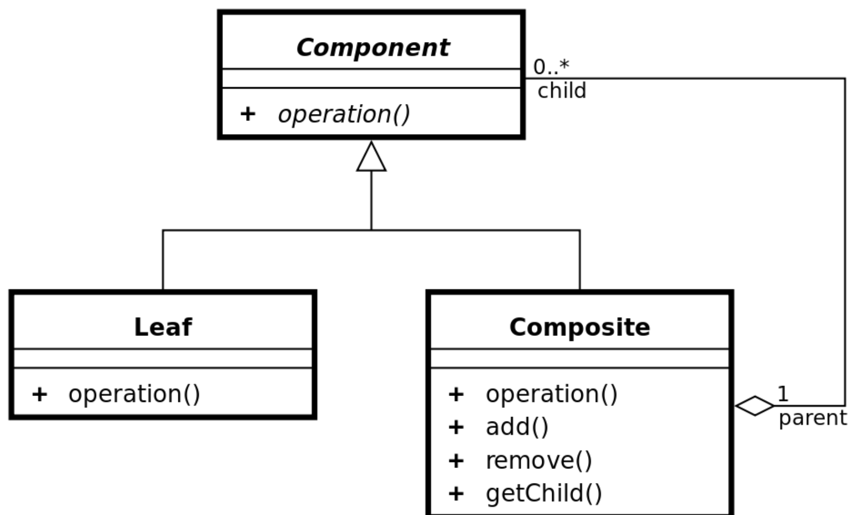
```
class IndexingScheme {
public:
  virtual size_t idx( size_t r, size_t c ) = 0;
  virtual ~IndexingScheme() {}
};
class RowMajor : public IndexingScheme {
  size_t _nCols;
  RowMajor() = delete;
public:
  RowMajor( size_t, size_t numCols ) :
    _nCols{numCols} {}
  size_t idx( size_t r, size_t c ) {
    return c + r * _nCols;
  }
};
```

Strategy

```
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



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...

Examples

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

MIT OpenCourseWare
http://ocw.mit.edu

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