Computer Programming
Exception Handling & Design Patterns

23rd Lecture

엄현상 (Eom, Hyeonsang)
컴퓨터공학부
서울대학교
순서

- Design Patterns
  - Visitor
- Q&A
Design Patterns

Definition
- Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context

Essential Elements
- Pattern name
- Problem
- Solution
- Consequences
  - Results and trade-off of applying the pattern

Design Patterns, E. Gamma, R. Helm, R. Johnson & J. Vlissides, Addison Wesley, '95
Visitor: A Design Pattern

The operation that gets executed depends on both the type of Visitor and the type of Element it visits.

Adds an operation to a class without modifying the class

- Every class has a virtual method `Accept(Visitor& v)`
- For every concrete class `S` that has `Accept`, the Visitor has a method `VisitS(S* s)`
- An object of class `Visitor` is passed to the `Accept` method
- `Accept` immediately calls `VisitS`, passing the `this` pointer as an argument
Visitor and ConcreteVisitor

Visitor

- Declares a Visit operation for each class of ConcreteElement in the object structure

ConcreteVisitor

- Implements each operation declared by Visitor
- Each operation implements a fragment of the algorithm defined for the corresponding class of object in the structure
- ConcreteVisitor provides the context for the algorithm and stores its local state
Element and ConcreteElement

- **Element**
  - Defines an Accept operation that takes a visitor as an argument

- **ConcreteElement**
  - Implements an Accept operation that takes a visitor as an argument

- **ObjectStructure**
  - Can enumerate its elements
  - May provide a high-level interface to allow the visitor to visit its elements
  - May either be a composite or a collection such as a list or a set

Design Patterns, E. Gamma, R. Helm, R. Johnson & J. Vlissides, Addison Wesley, ‘95
Visitor Class

class Visitor
{
    public:
        virtual void VisitElementA(ElementA*);
        virtual void VisitElementB(ElementB*);
        virtual void VisitCompositeElement(CompositeElement*);
    protected:
        Visitor();
};
ConcreteVisitor Class

class ConcreteVisitor : public Visitor
{
    public:
        ConcreteVisitor();
        virtual void VisitElementA(ElementA*);
        virtual void VisitElementB(ElementB*);
        virtual void VisitCompositeElement(CompositeElement*);
};
class Element
{
  public:
    virtual ~Element();
    virtual void Accept(Visitor&) = 0;
  protected:
    Element();
};
Element Class

class ElementA : public Element
{
    public:
        ElementA();
        virtual void Accept(Visitor& v) {
            v.VisitElementA(this);
        }
};

class ElementB : public Element
{
    public:
        ElementB();
        virtual void Accept(Visitor& v) {
            v.VisitElementB(this);
        }
};

Design Patterns, E. Gamma, R. Helm, R. Johnson & J. Vlissides, Addison Wesley, '95
CompositeElement Class

```cpp
class CompositeElement : public Element
{
  public:
    virtual void Accept(Visitor&);
  private:
    List<Element*>* _children;
};

void CompositeElement::Accept(Visitor& v)
{
  ListIterator<Element*>* i(_children);
  for (i.First(); !i.IsDone(); i.Next()) {
    i.CurrentItem()->Accept(v);
  }
  v.VisitCompositeElement(this);
}
```

Design Patterns, E. Gamma, R. Helm, R. Johnson & J. Vlissides, Addison Wesley, ‘95
How to Use?

CompositeElement* e;

Visitor v;
...

e->Accept(v);

Or
ConcreteVisitor cv;
...

e->Accept(cv);
Consequences

- Visitor makes adding new OPs easy
- A Visitor gathers related operations and separates unrelated ones
  - Related behavior is localized in a visitor while unrelated sets are partitioned in subclasses
- Adding new ConcreteElement classes is hard
- Visiting across class hierarchies
- Accumulating state
- Breaking encapsulation

Design Patterns, E. Gamma, R. Helm, R. Johnson & J. Vlissides, Addison Wesley, ‘95
Reference for Design Patterns

“Design Patterns: Elements of Reusable Object-Oriented Software,” Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison Wesley, 1995