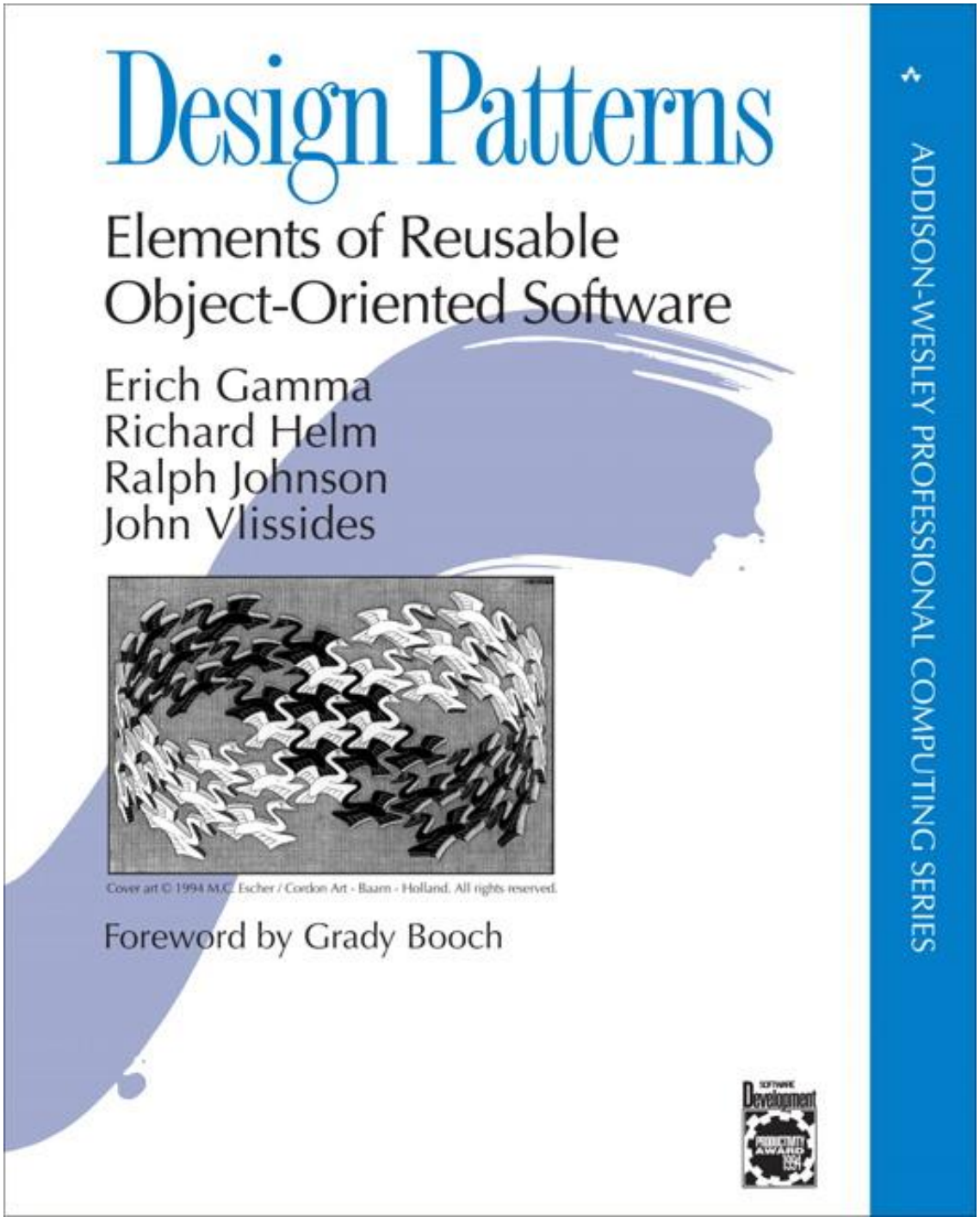
EAPLI

# Mais Alguns Padrões GoF

Paulo Gandra de Sousa
pag@isep.ipp.pt

| Topic | Principles and patterns |
| --- | --- |
| Which class should a responsibility be assigned to? | Information Expert, Tell, don't ask<br>Single Responsibility Principle<br>Interface Segregation Principle<br>Intention Revealing Interfaces |
| How to organize the system's responsibilities? | Layers,<br>High Cohesion, Low Coupling<br>Facade<br>Controller |
| How to model the domain? | Persistence Ignorance<br>Entity, Value Object, Aggregate<br>Domain Service<br>Domain Event<br>Observer |
| How to handle an object's lifecycle? | Factories<br>Repositories |
| How to prepare the code for modification? | Protected Variation<br>Open/Close Principle<br>Dependency Inversion Principle<br>Liskov Substitution Principle<br>Template Method<br>Strategy<br>Decorator |

# Introdução


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

1995

# Rules of thumb

- Programar para uma interface e não para uma implementação
  - Diminuir o acoplamento/dependências entre classes
- Favorecer a composição em vez da herança
  - i.e. preferível herança apenas no conceito de especialização conceptual e não funcional
- Separação de responsabilidades
  - "cada macaco no seu galho"

# 3 Tipos de Padrões GoF

- Padrões criacionais
  - Respeitantes à inicialização e configuração de objectos
- Padrões estruturais
  - Composição de classes ou objectos
  - Desacoplar interface e implementação das classes
- Padrões comportamentais
  - Respeitantes à dinâmica de interacções entre sociedades de objectos
  - Como se distribui a responsabilidade pelos objectos

# Alguns Padrões

# Conteúdo

- Adapter
- Command
- Composite
- Visitor

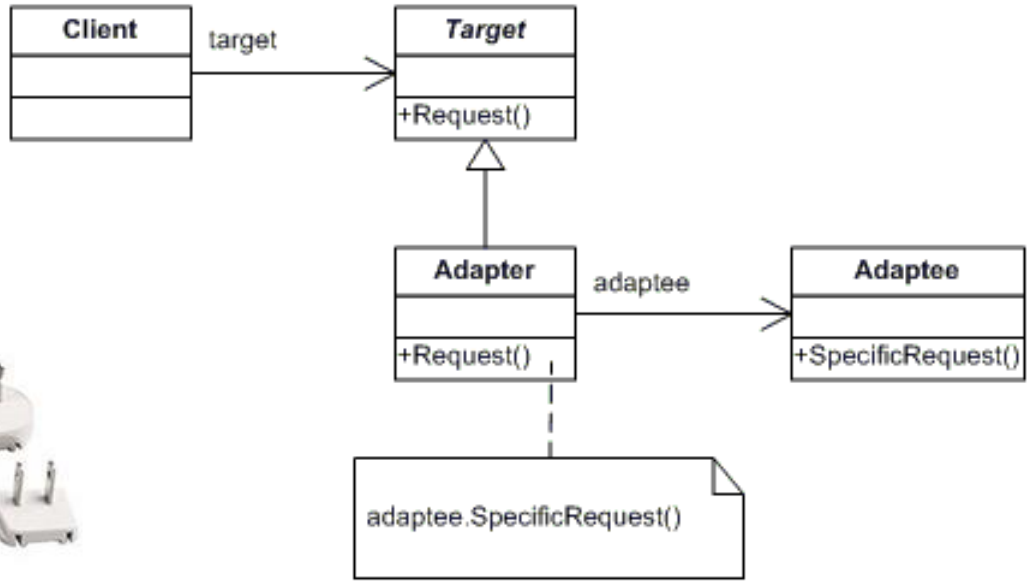
# Adapter

# Adapter

- Problema:
  - Como ligar um cliente a um fornecedor de serviços quando a interface fornecida é diferente da esperada
- Solução:
  - Criar uma classe que forneça a interface esperada pelo cliente e que utilize a interface do fornecedor de serviço

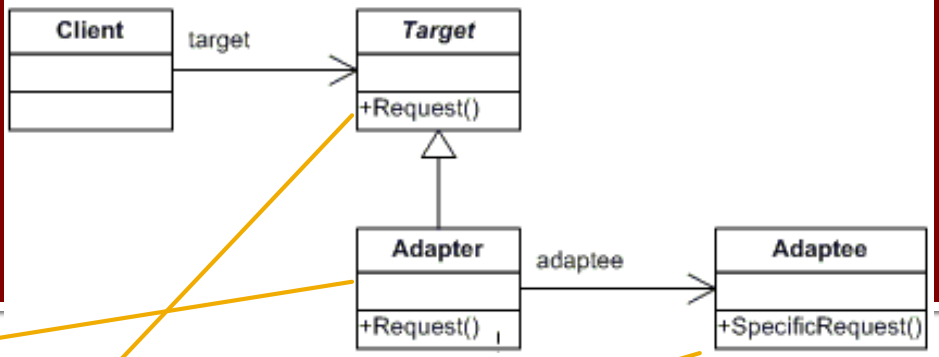
# Adapter

- Converter a interface duma classe numa interface expectável pelo cliente. Adapter permite que classes trabalhem em conjunto que de outro modo não seria possível devido a interfaces incompatíveis.



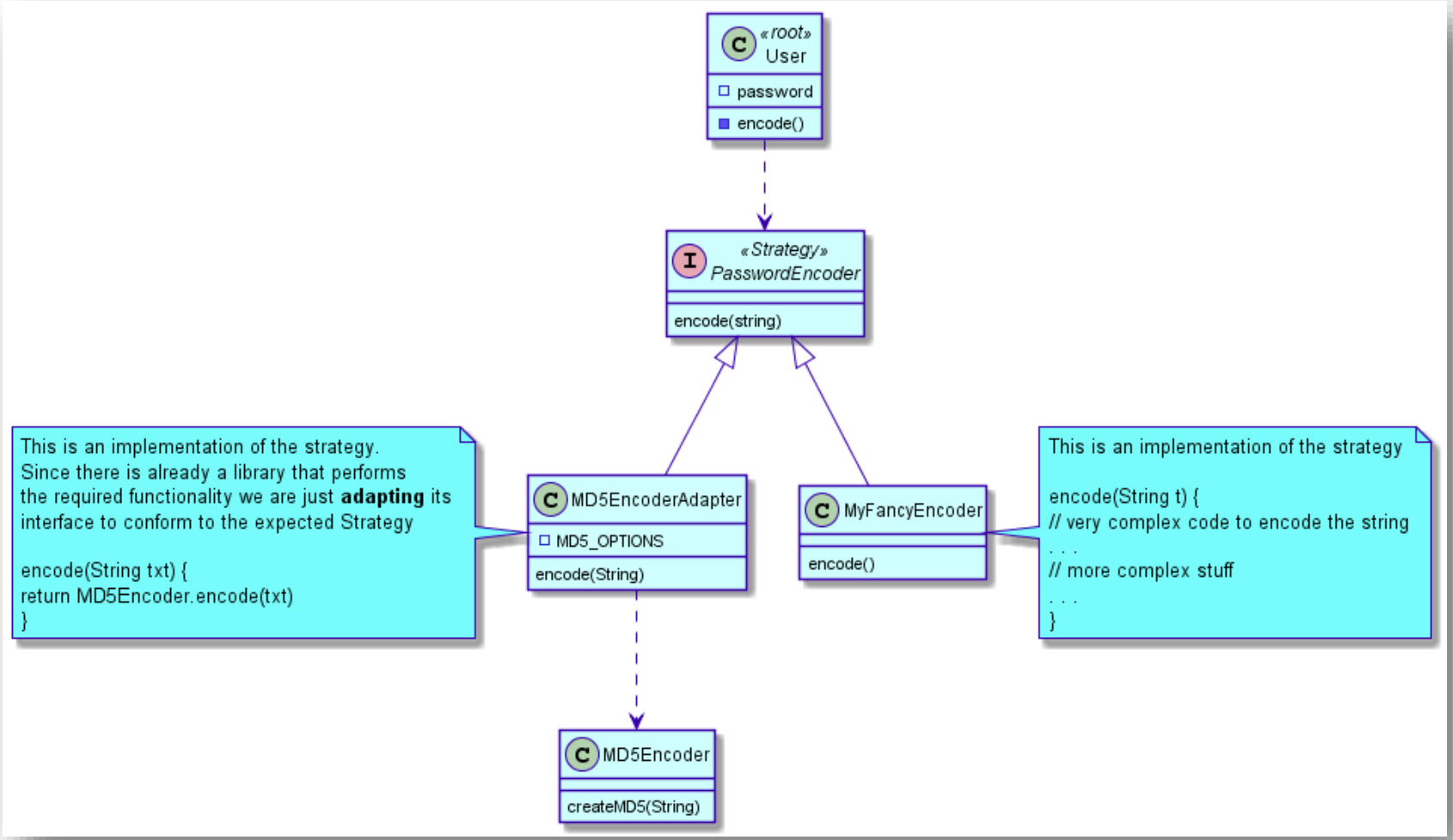

*fonte*: Design Patterns: Elements of Reusable Object-Oriented Software

# Exemplo (Java)




```
class DataBoxAdapter implements Collection {
   // DataBox is some collection that does not support the
      Collection interface
   private DataBox data = new DataBox();

   public boolean isEmpty () { return data.count() == 0; }

   public int size ()        { return data.count(); }

   public void addElement (Object newElement)
      {data.add(newElement); }

   public boolean containsElement (Object test)
      { return data.find(test) != null; }

   public Object findElement (Object test)
      { return data.find(test); }
}
```

# Strategy + Adapter



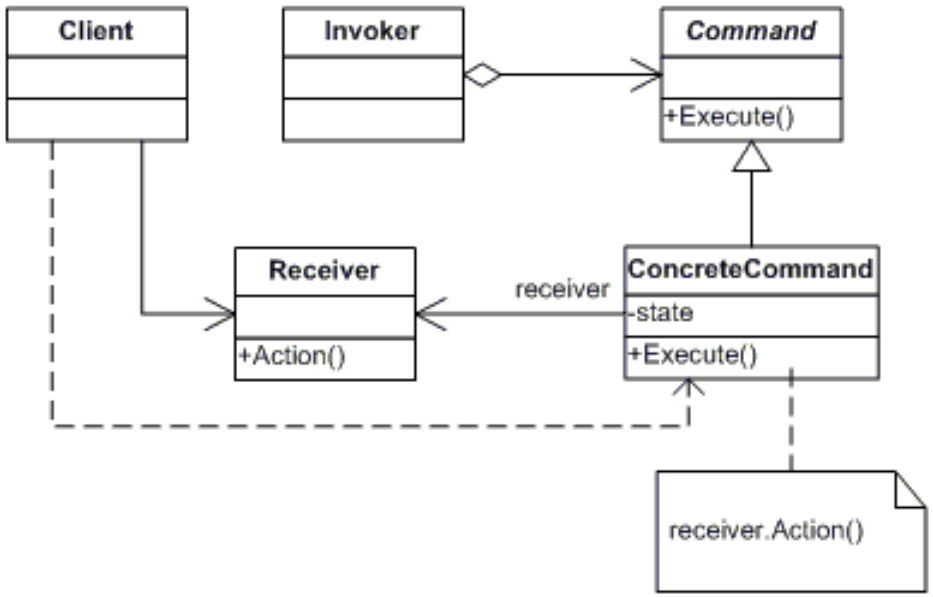
# Command

# Command

- **Problema:**
  - Como executar acções em objectos que só serão conhecidos posteriormente (ex., *toolkit* de janelas) e/ou permitir que essas acções sejam tratadas por vários objectos
- **Solução:**
  - Criar o conceito de comando e de receptor

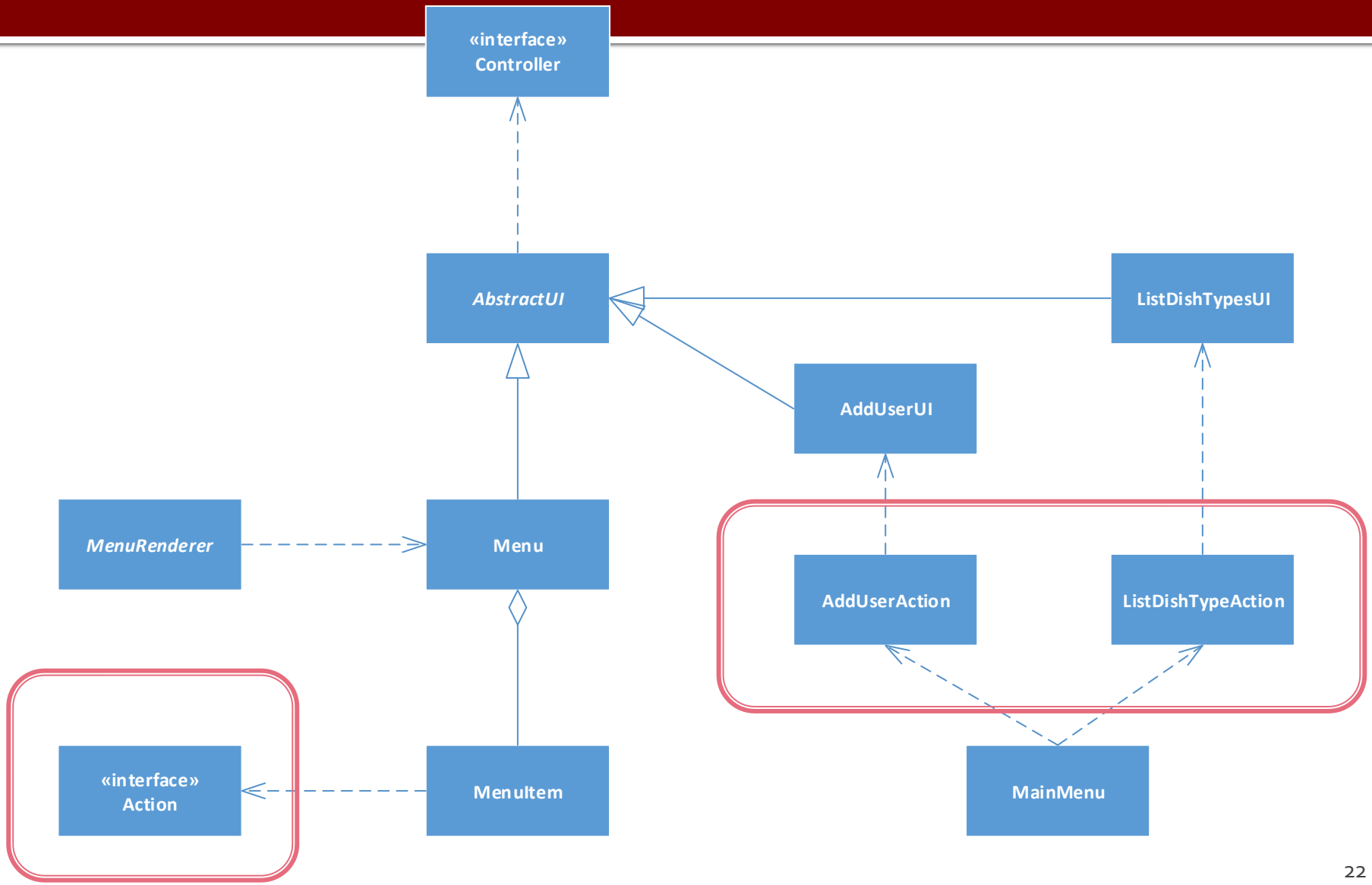
# Command

- Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations




*fonte*: Design Patterns: Elements of Reusable Object-Oriented Software

# EAPLI Framework: Presentation

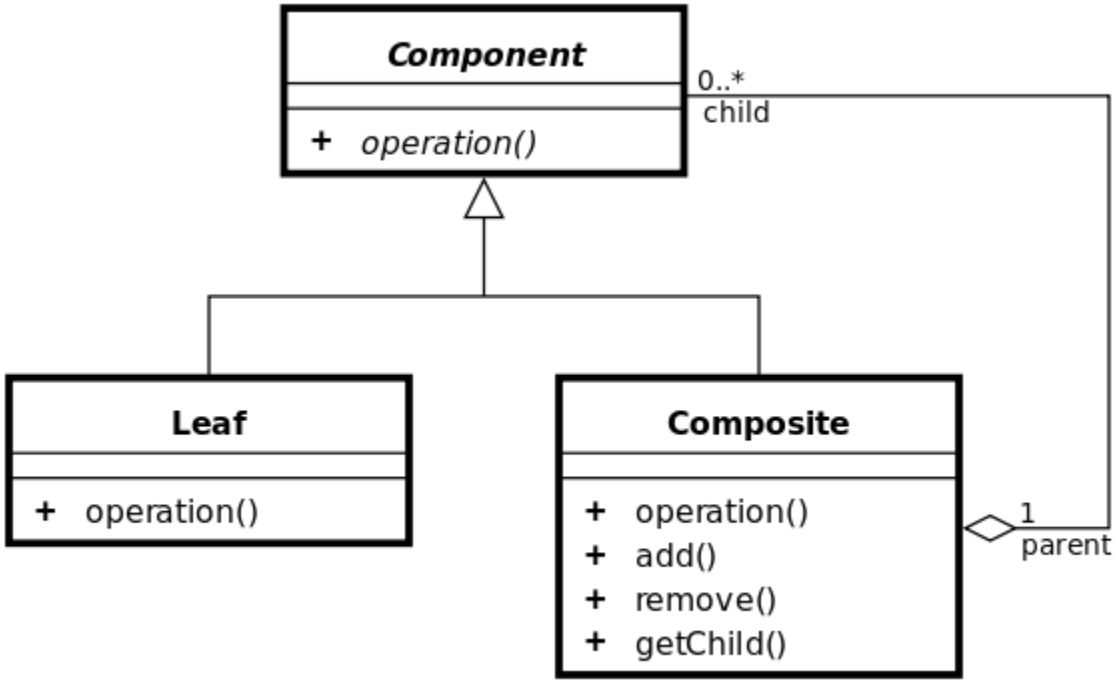


# Composite

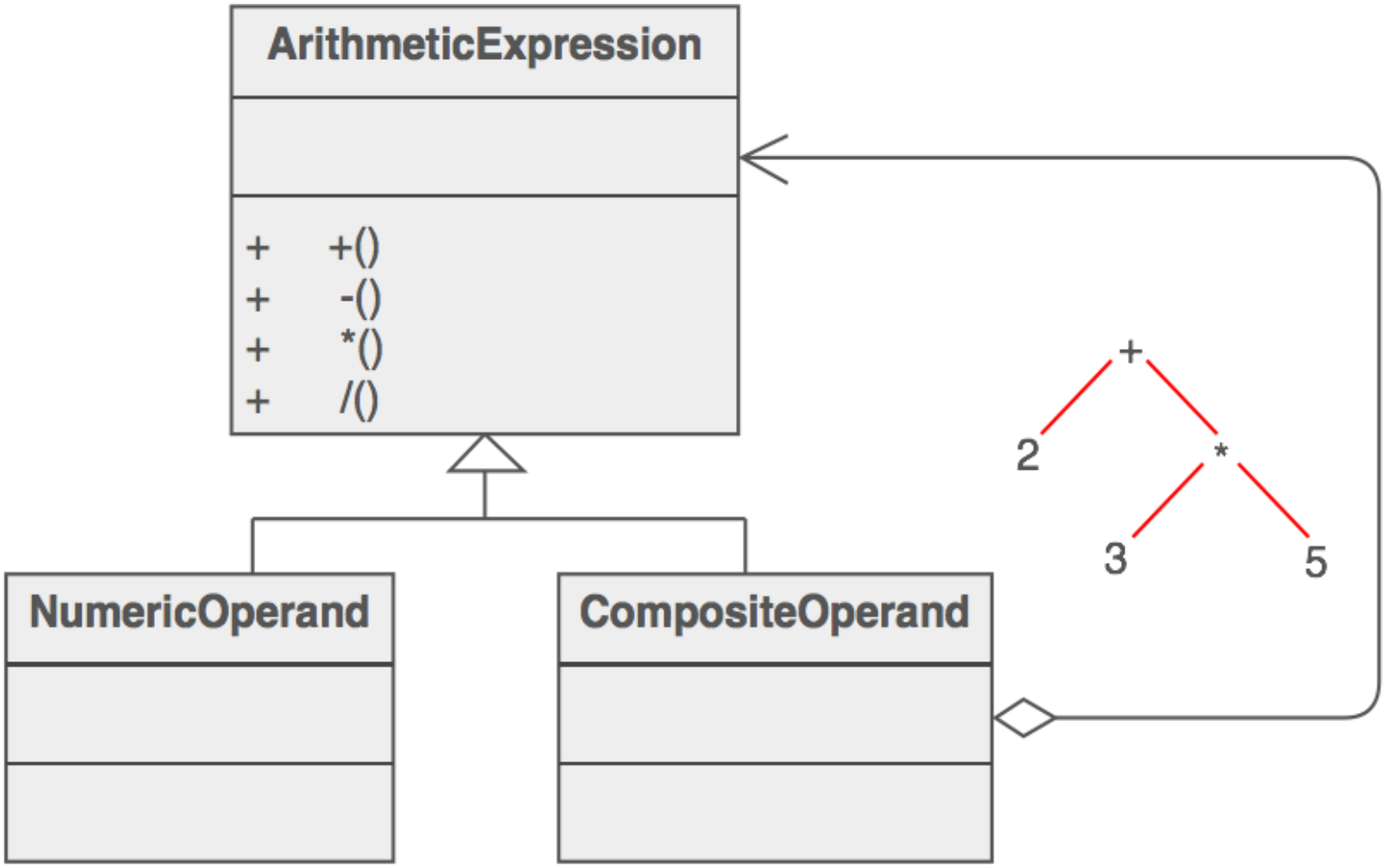
# Composite

- Problem
  - How to treat individual objects and compositions of objects uniformly
- Solution
  - compose objects into tree structures to represent part-whole hierarchies.

# Composite

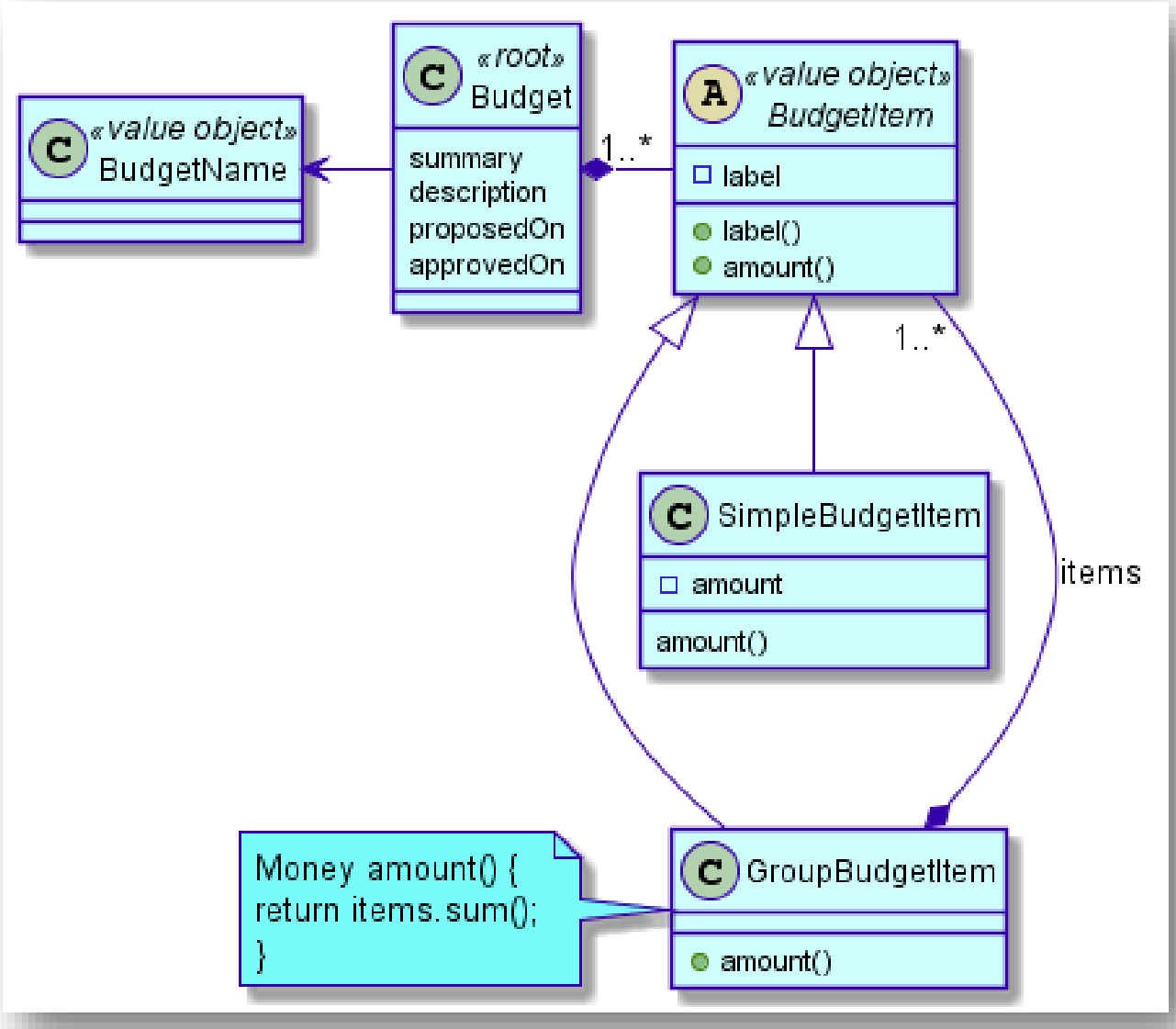

Component
+ operation()
0..*
child
Leaf
+ operation()
Composite
+ operation()
+ add()
+ remove()
+ getChild()
1
parent

# Example




https://sourcemaking.com/design_patterns/composite

# Example


«root»
Budget
summary
description
proposedOn
approvedOn
«value object»
BudgetName
1..*
«value object»
BudgetItem
label
label()
amount()
1..*
items
SimpleBudgetItem
amount
amount()
GroupBudgetItem
amount()
Money amount() {
return items.sum();
}

# Composite and others

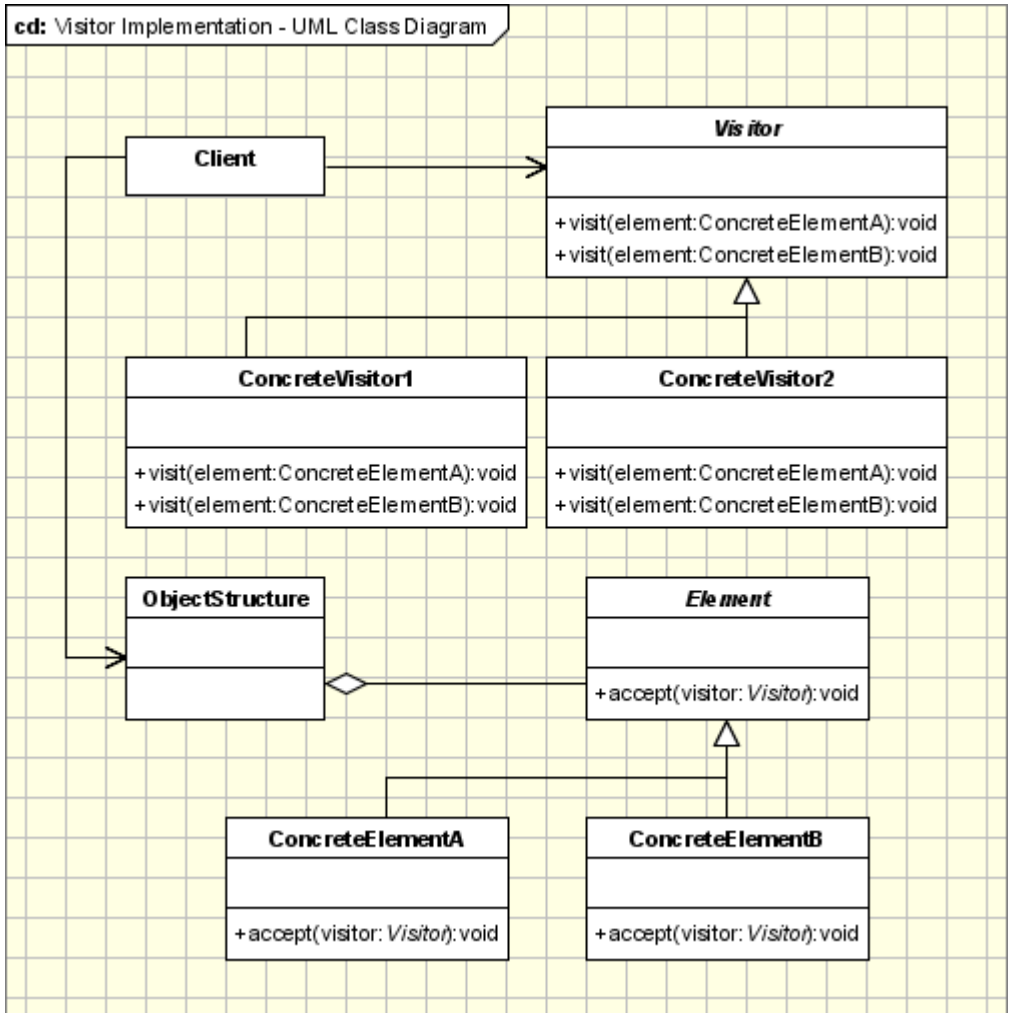
- Visitor
  - Can be used to traverse a composite
- Decorator
  - The base structure for decorator is composite
- Strategy
  - (small) Strategies can be reused and combined using a composite

# Visitor

# Visitor

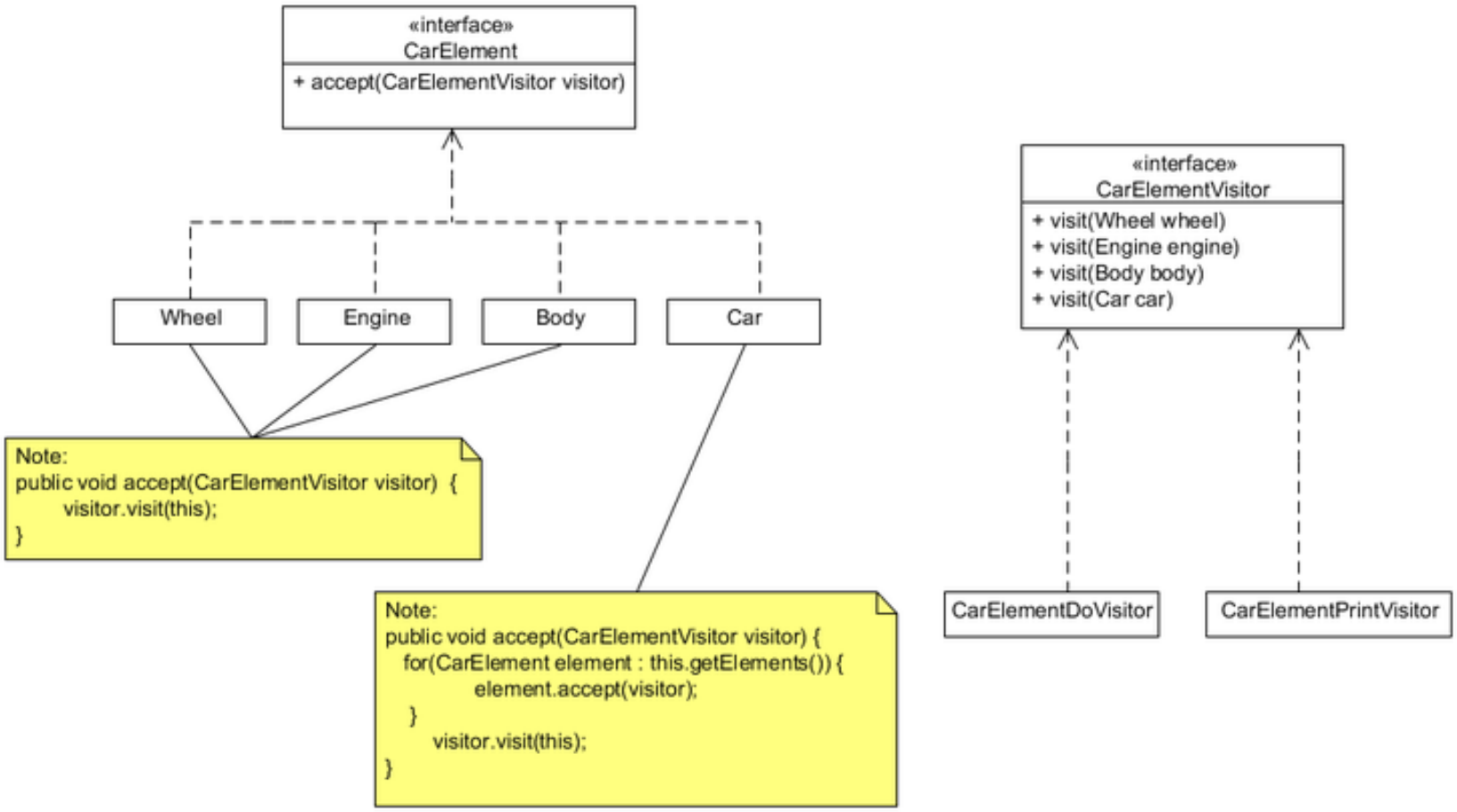
- Problem
  - How to dinamically add new operations to existing data structures?
- Solution
  - Separate the algorithm from the data structure it operates on by creating a visitor that traverses the desired object.

# Visitor




Source: http://www.oodesign.com/visitor-pattern.html

# Example




Source: http://en.wikipedia.org/wiki/Visitor_pattern

# Example

```
interface Node{
        void accept(Visitor v);
        int value();
}

class IntNode implements Node {
        private int value;
        public int value() { return value; }
        public accept(Visitor v) { v.visit(this); }
}

class SumVisitor implements Visitor {
        private int sum;
        public void visit(Node n) { sum += n.value(); }
}

class PrintVisitor implements Visitor {
        private int sum;
        public void visit(Node n) { System.out.println(n.value());
        }
}
```

# Example (cont.d)

```
class IntList implements Node{
      List<Node> elems = ...

      public void accept(Visitor v) {
            for (Node n: elems) {
                  n.accept(v);
            }
      }

      public int value() {
            SumVisitor adder = new SumVisitor();
            this.accept(adder);
            return adder.getSum();
      }
}
```

# Sinopse

# Sinopse de Padrões GoF

- Práticas vindas da experiência
- Soluções comprovadas para problemas recorrentes
- Utilizam-se em conjunto
- Há padrões muito semelhantes entre si
- Deve-se identificar os problemas e com base nisso aplicar o padrão
- Não se deve aplicar um padrão apenas por aplicar, ou por ser moda

# Rules of thumb

- Programar para uma interface e não para uma implementação
  - Diminuir o acoplamento/dependências entre classes
- Favorecer a composição em vez da herança
  - Herança apenas no conceito de especialização conceptual e não funcional
- Separação de responsabilidades
  - "cada macaco no seu galho"

# Padrões Estruturais

- Adapter
  - Converte interface duma classe numa expectável pelo cliente
- Bridge
  - Liga abstracção com muitas implementações possíveis
- Composite
  - Representa hierarquias parte-todo em estruturas em árvore
- Decorator
  - Associa dinamicamente responsabilidades adicionais a objectos
- Facade
  - Simplifica a interface dum sistema
- Flyweight
  - Partilha eficientemente muitos objectos de granularidade fina
- Proxy
  - Fornece um substituto ("surrogate") ou "placeholder" para um objecto para controlar o acesso a ele próprio

# Padrões Criacionais

- Singleton
  - Garante acesso a uma única instância duma classe
- Simple Factory
  - Cria objectos especializados e complexos
- Abstract Factory
  - Cria uma família de fábricas especializadas
- Factory Method
  - Define uma interface para a criação dum objecto, mas permite que a subclasse decida que classe instanciar
- Builder
  - Constrói um objecto complexo passo a passo
- Prototype
  - Cria (duplica) instâncias a partir dum protótipo
- Lazy initialization
  - Atrasa a criação cara até que o objecto seja necessário

# Padrões Comportamentais

- Chain of Responsibility
  - O pedido é delegado ao serviço responsável
- Command
  - Request ou Action é um objecto duma classe de primeira, logo armazenável
- Iterator
  - Agrega e acede sequencialmente a elementos
- Interpreter
  - Interpretador duma linguagem para uma gramática pequena
- Mediator
  - Coordena interacções entre os seus associados
- Memento
  - Captura e repõe imagem do objecto (Snapshot) privadamente

# Padrões Comportamentais (cont.)

- Observer
  - Actualiza automaticamente quando os objectos observados se alteram
- State
  - Objecto cujo comportamento depende do seu estado
- Strategy
  - Abstracção para a selecção de um ou mais algoritmos
- Template Method
  - Algoritmo com alguns passos fornecidos pela classe derivada
- Visitor
  - Operações aplicadas a elementos duma estrutura de objectos heterogéneos

# Catálogos de padrões

- GoF Design patterns (em C#)
  http://www.dofactory.com/Patterns/Patterns.aspx

- GoF & POSA Design patterns (em Java)
  http://www.vico.org/pages/PatronsDisseny.html