



深圳大学
Shenzhen University

第17-2讲

访问者模式

软件体系结构与设计模式
Software Architecture & Design Pattern

深圳大学计算机与软件学院



主要内容

- ◆ 访问者模式动机与定义
- ◆ 访问者模式结构与分析
- ◆ 访问者模式实例与解析
- ◆ 访问者模式效果与应用

访问者模式动机

■ 医院处方单处理示意图



人民医院					
处方笺					
费别: 现金	医疗证 / 医保卡号: _____				
姓名: 李四	性别: 男	年龄: 26岁			
门诊 / 住院病历号: 475868	科室: 药房				
诊断: 肺炎	开具日期: 2010年09月10日				
电话地址: 广州大道中路129号309					
Rp					
收费名称	规格	单位	数量	用法	用量
1 青霉素	80万u	瓶	2	birp tb	成人一次二片, 一日三次, 儿童一日一至三片
2 黄连素针	50ml	支	2	birp tb	成人一次二片, 一日三次, 儿童一日一至三片
3 葡萄糖盐水	250ml	瓶	1	RIT	250ml
医师: 00 _____					
审核药师: _____		调配药师: _____		核对、发药药师: _____	
收费员: _____					



访问者模式动机

- 处方单：

- 药品信息的集合，包含一种或多种不同类型的药品信息
- 不同类型的工作人员（例如划价人员和药房工作人员）在操作同一个药品信息集合时将提供不同的处理方式
- 可能会增加新类型的工作人员来操作处方

- 软件开发：

- | | |
|--------|------|
| □ 处方单 | 对象结构 |
| □ 药品信息 | 元素 |
| □ 工作人员 | 访问者 |



访问者模式动机

- 对象结构中存储了多种不同类型的对象信息
- 对同一对象结构中的元素的操作方式并不唯一，可能需要提供多种不同的处理方式
- 还有可能需要增加新的处理方式
- 以不同的方式操作复杂对象结构 → 访问者模式

访问者模式定义

■ 对象行为型模式

访问者模式：表示一个作用于某对象结构中的各个元素的操作。访问者模式让你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

Visitor Pattern: Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

- 它为操作存储不同类型元素的对象结构提供了一种解决方案
- 用户可以对不同类型的元素施加不同的操作

访问者模式结构

■ 访问者模式包含如下5个角色：

□ Visitor (抽象访问者)

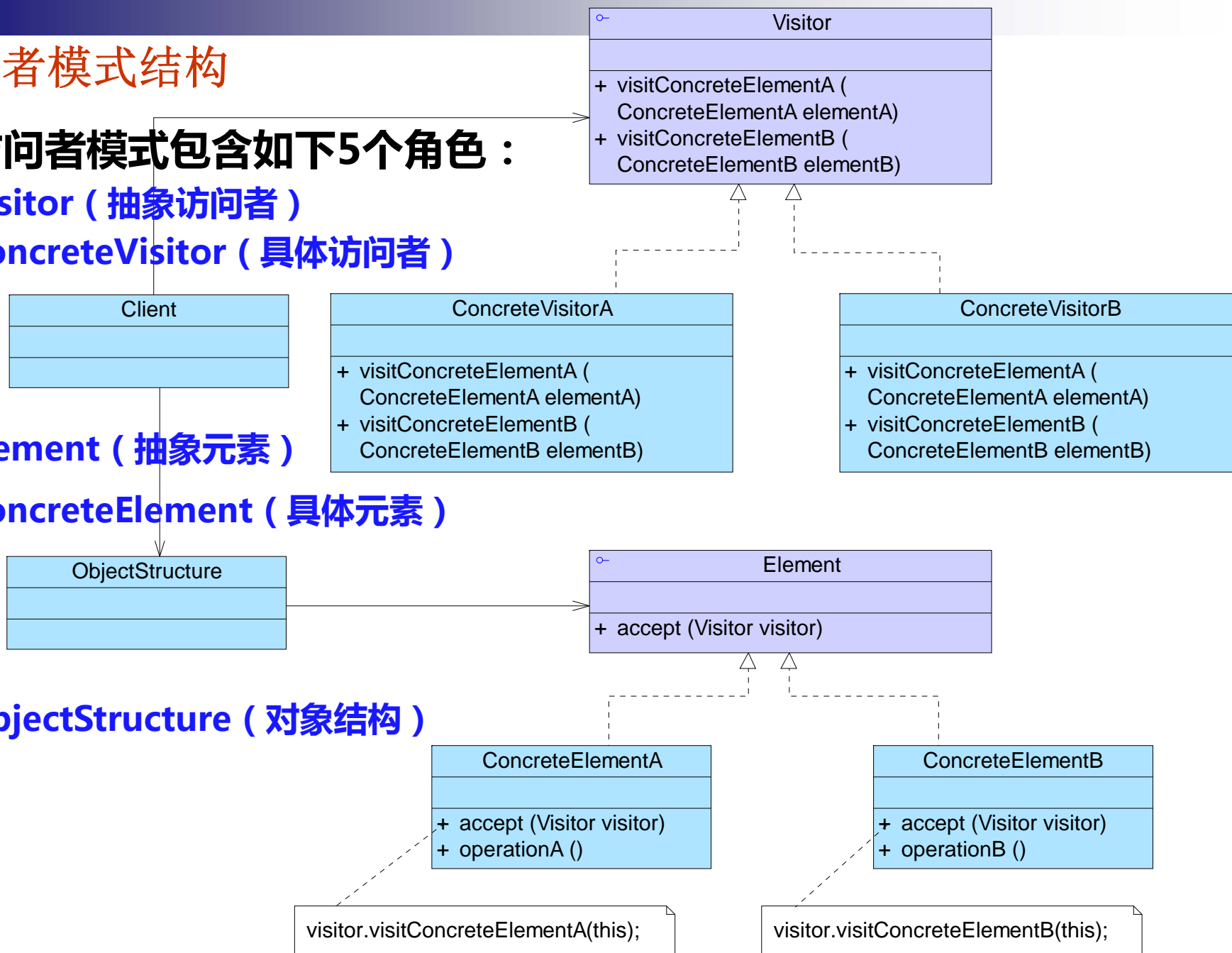
□ ConcreteVisitor (具体访问者)

□ Element (抽象元素)

□ ConcreteElement (具体元素)

□ ObjectStructure (对象结构)

□





访问者模式分析

- 抽象访问者类示例代码：

```
public abstract class Visitor {  
    public abstract void visit(ConcreteElementA elementA);  
    public abstract void visit(ConcreteElementB elementB);  
  
    public void visit(ConcreteElementC elementC) {  
        //元素ConcreteElementC操作代码  
    }  
}
```




访问者模式分析

- 具体访问者类示例代码：

```
public class ConcreteVisitor extends Visitor {  
    public void visit(ConcreteElementA elementA) {  
        //元素ConcreteElementA操作代码  
    }  
  
    public void visit(ConcreteElementB elementB) {  
        //元素ConcreteElementB操作代码  
    }  
}
```

访问者模式分析

- 抽象元素类示例代码：

```
public interface Element {  
    public void accept(Visitor visitor);  
}
```

- 具体元素类示例代码：

```
public class ConcreteElementA implements Element {  
    public void accept(Visitor visitor) {  
        visitor.visit(this);  
    }  
  
    public void operationA() {  
        //业务方法  
    }  
}
```

访问者模式分析

■ 对象结构示

例代码：

```
import java.util.*;
public class ObjectStructure
{ // 定义一个集合用于存储元素对象
    private ArrayList<Element> list = new ArrayList<Element>();
    // 接受访问者的访问操作
    public void accept(Visitor visitor) {
        Iterator i=list.iterator();

        while(i.hasNext()) { // 遍历访问集合中的每一个元素
            ((Element)i.next()).accept(visitor); }
        }

    public void addElement(Element element) {
        list.add(element);
    }
    public void removeElement(Element element) {
        list.remove(element);
    }
}
```



访问者模式分析

■ 双重分派机制

- (1) 调用具体元素类的accept(Visitor visitor)方法，并将Visitor子类对象作为其参数
- (2) 在具体元素类accept(Visitor visitor)方法内部调用传入的Visitor对象的visit()方法，例如
visit(ConcreteElementA elementA)，将当前具体元素类对象(this)作为参数，例如visitor.visit(this)
- (3) 执行Visitor对象的visit()方法，在其中还可以调用具体元素对象的业务方法



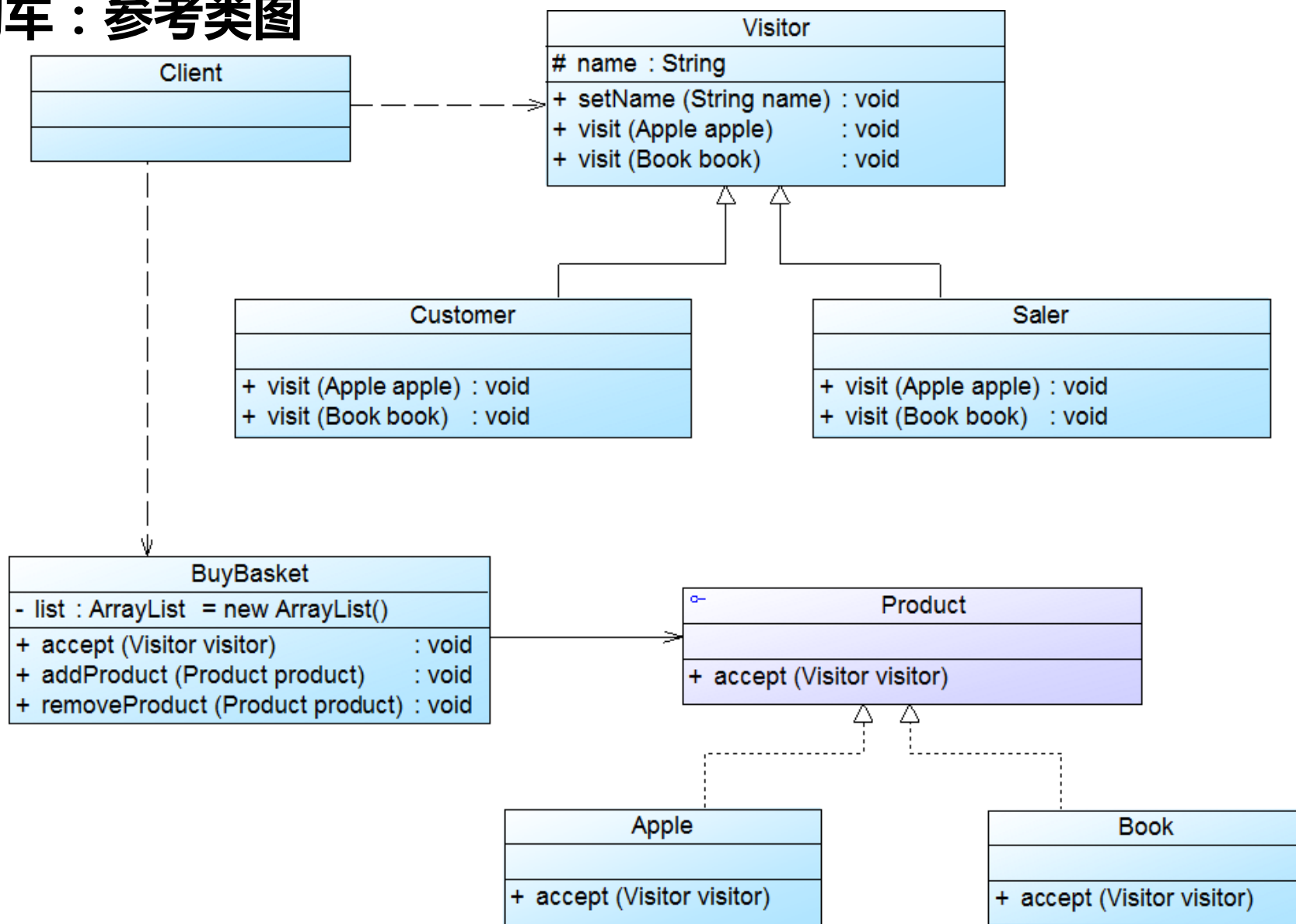
访问者模式实例

■ 购物车：实例说明

- 顾客在超市中将选择的商品，如苹果、图书等放在购物车中，然后到收银员处付款。在购物过程中，顾客需要对这些商品进行访问，以便确认这些商品的质量，之后收银员计算价格时也需要访问购物车内顾客所选择的商品。此时，购物车作为一个 ObjectStructure（对象结构）用于存储各种类型的商品，而顾客和收银员作为访问这些商品的访问者，他们需要对商品进行检查和计价。不同类型的商品其访问形式也可能不同，如苹果需要过秤之后再计价，而图书不需要。使用访问者模式来设计该购物过程。

访问者模式实例与解析

■ 购物车：参考类图



访问者模式实例

- 购物车：参考代码
- DesignPatterns之visitor包

Product.java

```
3 public interface Product
4 {
5     void accept(Visitor visitor);
6 }
```

Visitor.java

```
3 public abstract class Visitor
4 {
5     protected String name;
6
7     public void setName(String name)
8     {
9         this.name=name;
10    }
11
12    public abstract void visit(Apple apple);
13
14    public abstract void visit(Book book);
15 }
```

访问

BuyBasket.java

```
1 package visitor;
2
3 import java.util.*;
4 public class BuyBasket
5 {
6     private ArrayList list=new ArrayList();
7
8     public void accept(Visitor visitor)
9     {
10         Iterator i=list.iterator();
11
12         while(i.hasNext())
13         {
14             ((Product)i.next()).accept(visitor);
15         }
16     }
17
18     public void addProduct(Product product)
19     {
20         list.add(product);
21     }
22
23     public void removeProduct(Product product)
24     {
25         list.remove(product);
26     }
27 }
```


Customer.java

```
3 public class Customer extends Visitor
4 {
5     public void visit(Apple apple)
6     {
7         System.out.println("顾客" + name + "选苹果。");
8     }
9
10    public void visit(Book book)
11    {
12        System.out.println("顾客" + name + "买书。");
13    }
14 }
```

Saler.java

```
3 public class Saler extends Visitor
4 {
5     public void visit(Apple apple)
6     {
7         System.out.println("收银员" + name + "给苹果过秤，然后计算其价格。");
8     }
9
10    public void visit(Book book)
11    {
12        System.out.println("收银员" + name + "直接计算书的价格。");
13    }
14 }
```

Apple.java

```
1 package visitor;
2
3 public class Apple implements Product
4 {
5     public void accept(Visitor visitor)
6     {
7         visitor.visit(this);
8     }
9 }
```

Book.java

```
1 package visitor;
2
3 public class Book implements Product
4 {
5     public void accept(Visitor visitor)
6     {
7         visitor.visit(this);
8     }
9 }
```

```
7 public class XMLUtil
8 {
9     //该方法用于从XML配置文件中提取具体类名，并返回一个实例对象
10    public static Object getBean()
11    {
12        try
13        {
14            //创建文档对象
15            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
16            DocumentBuilder builder = dFactory.newDocumentBuilder();
17            Document doc;
18            doc = builder.parse(new File("Visitorconfig.xml"));
19            //获取包含类名的文本节点
20            NodeList nl = doc.getElementsByTagName("className");
21            Node classNode=nl.item(0).getFirstChild();
22            String cName=classNode.getNodeValue();
23            Class c=Class.forName(cName); //通过类名生成实例对象并将其返回
24            Object obj=c.newInstance();
25            return obj;
26        }
27        catch(Exception e)
28        {
29            e.printStackTrace();
30            return null;
31        }
32    }
33 }
```

Visitorconfig.xml

```
1 <?xml version="1.0"?>
2 <config>
3     <className>visitor.Customer</className>
4 </config>
```

Client.java

```
1 package visitor;
2
3 public class Client
4 {
5     public static void main(String a[])
6     {
7         Product b1=new Book();
8         Product b2=new Book();
9         Product a1=new Apple();
10        Visitor visitor;
11
12        BuyBasket basket=new BuyBasket();
13        basket.addProduct(b1);
14        basket.addProduct(b2);
15        basket.addProduct(a1);
16
17        visitor=(Visitor)XMLUtil.getBean();
18
19        visitor.setName("张三");
20
21        basket.accept(visitor);
22    }
23 }
```



访问者模式效果与应用

■ 访问者模式优点：

- 增加新的访问操作很方便
- 将有关元素对象的访问行为集中到一个访问者对象中，而不是分散在一个个的元素类中，类的职责更加清晰
- 让用户能够在不修改现有元素类层次结构的情况下，定义作用于该层次结构的操作



访问者模式效果与应用

■ 访问者模式缺点：

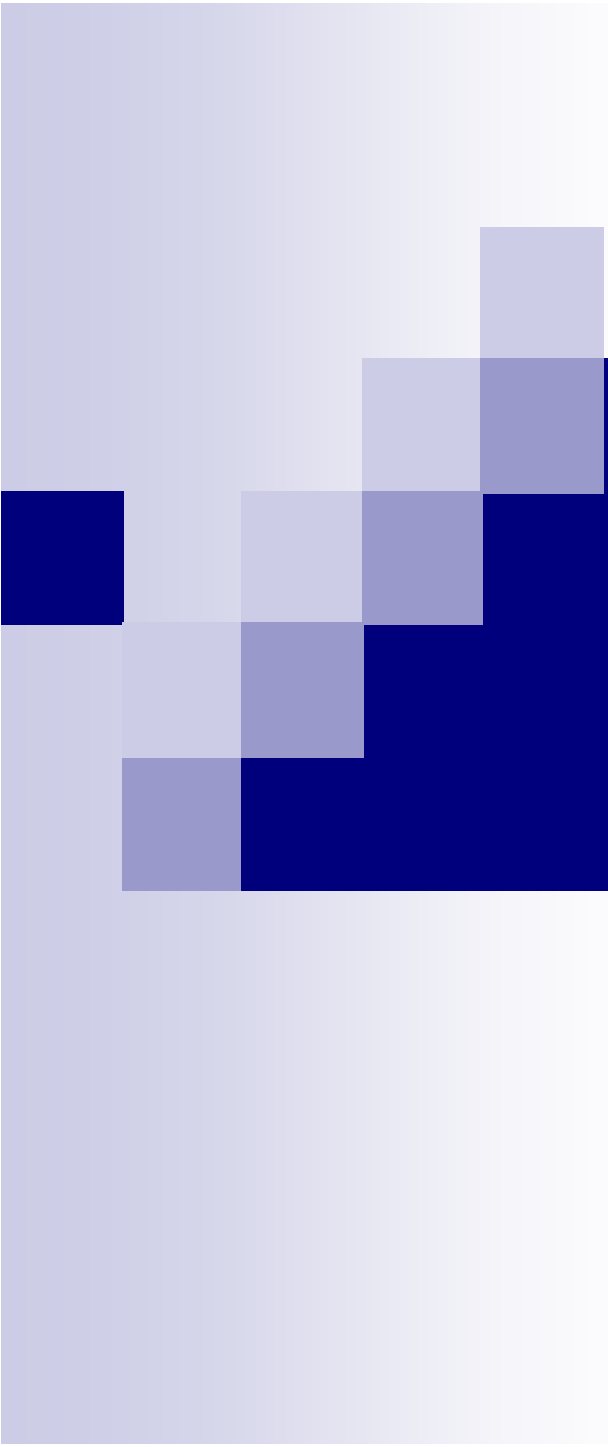
- 增加新的元素类很困难
- 破坏了对象的封装性



访问者模式效果与应用

■ 在以下情况下可以使用访问者模式：

- 一个对象结构包含多个类型的对象，希望对这些对象实施一些依赖其具体类型的操作
- 需要对一个对象结构中的对象进行很多不同的且不相相关的操作，并需要避免让这些操作“污染”这些对象的类，也不希望在增加新操作时修改这些类
- 对象结构中对象对应的类很少改变，但经常需要在此对象结构上定义新的操作



谢谢