Lecture 7.
Visitor Pattern (访问者模式)
(Behavioral)

- 行为模式关心算法和对象之间的责任分配。
- 它关心的不是仅仅描述对象或类的模式，而是要更加侧重描述它们之间的通信模式。
- 行为模式刻画了很难在运行时跟踪的复杂的控制流。该模式将软件开发者的注意力从控制流转移到对象相互关联的方式方面。

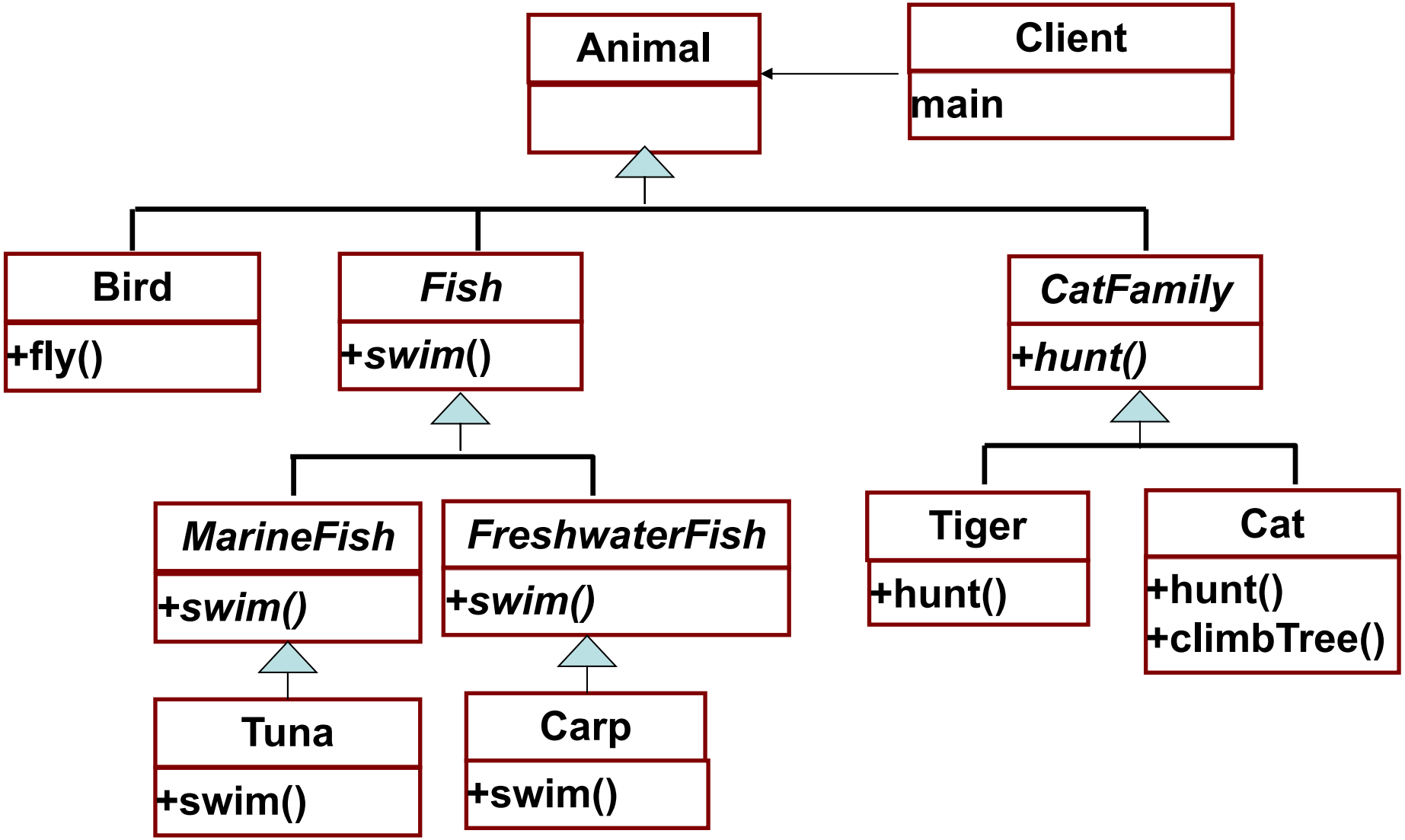
Professor:
Yushan (Michael) Sun
Fall 2020

Contents of the lecture

1. Introduction of the Visitor Pattern
2. Official Diagram of the Visitor Design Pattern
3. Design Example Using the Visitor Pattern
4. Further discussions

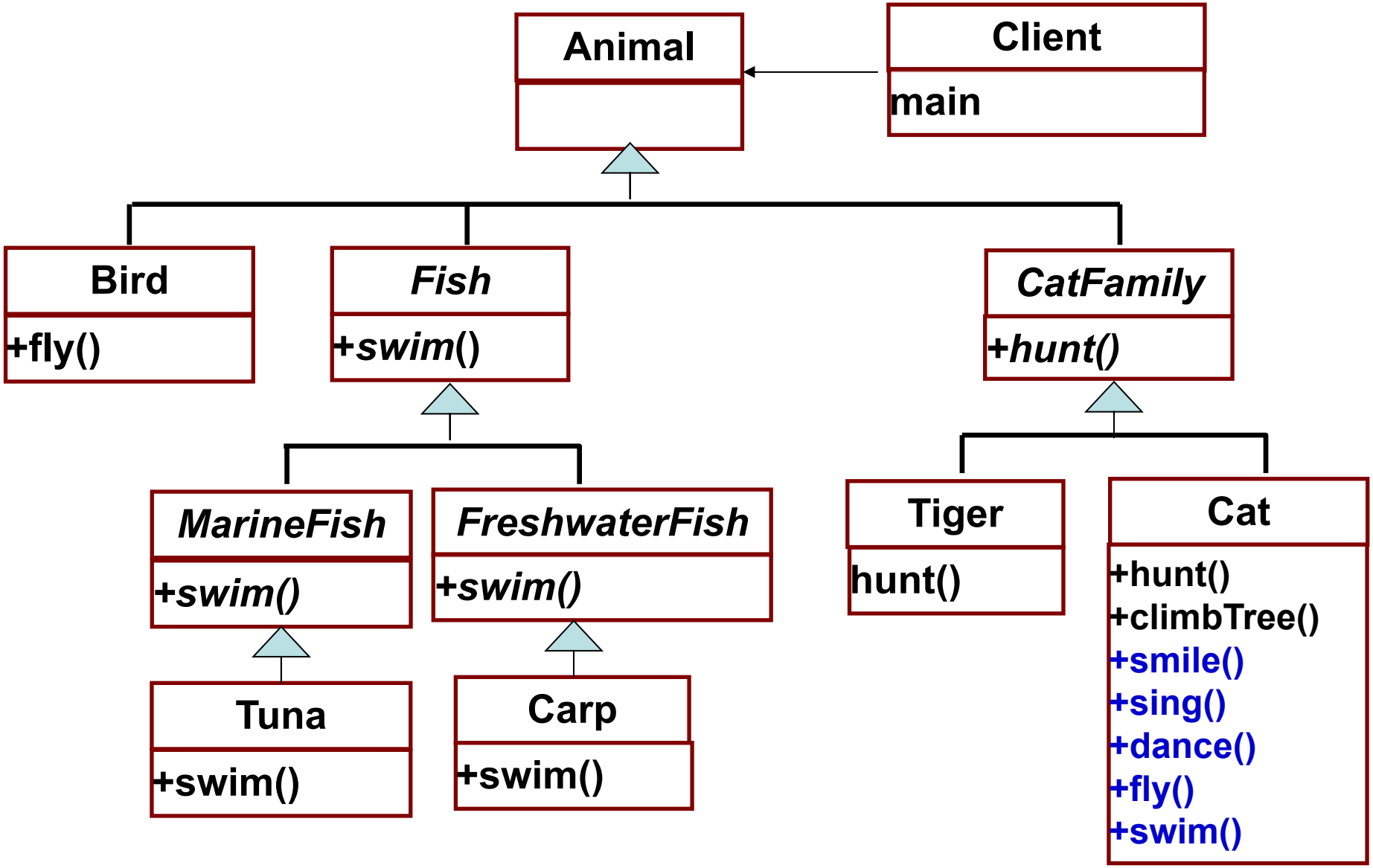
Introduction of the Visitor Pattern

[例1] Animal class



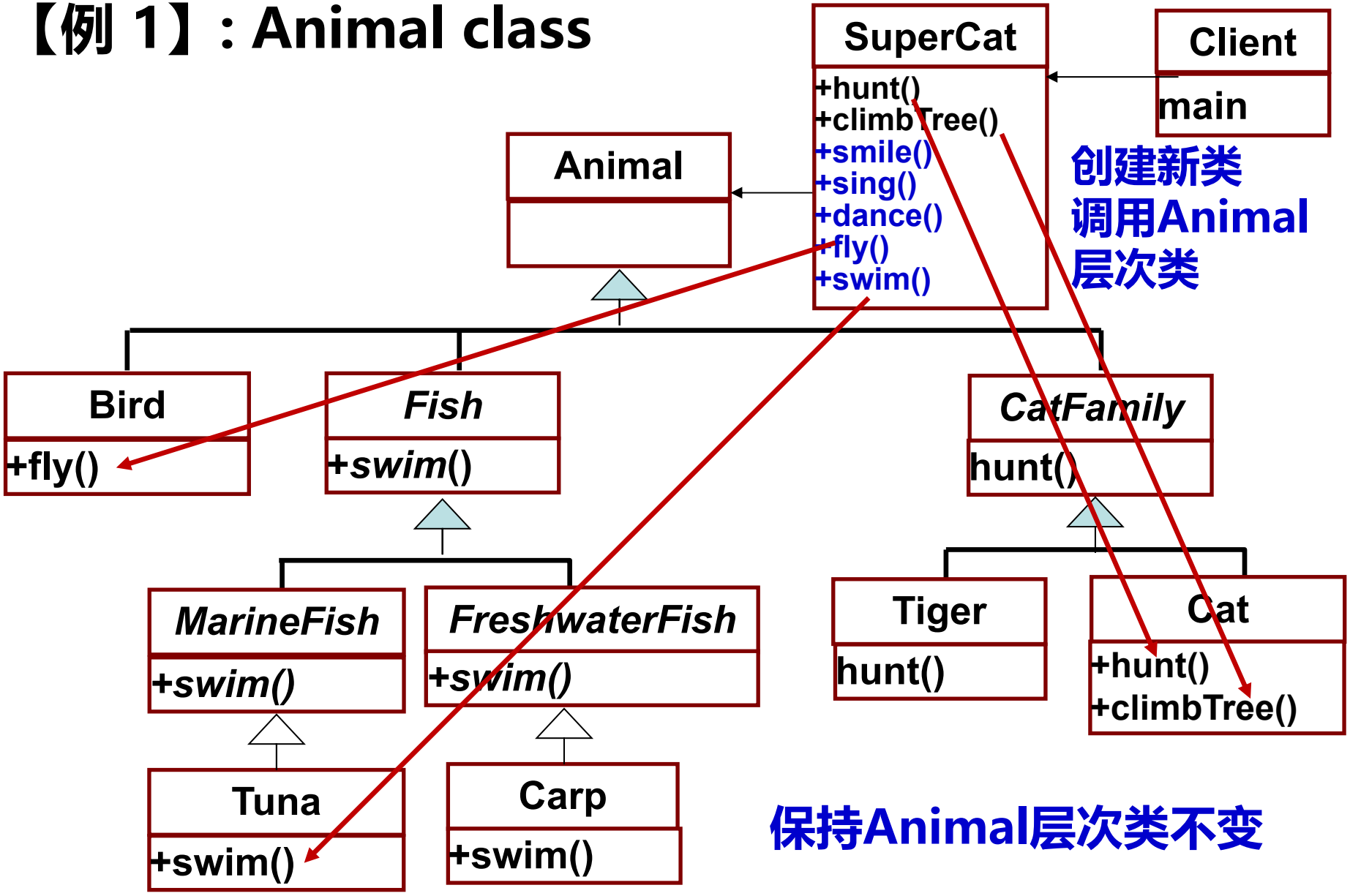
生物项目团队写的Animal层次类

[例1] Animal class



网络游戏开发团队修改Animal层次类-造成了接口污染

【例 1】: Animal class



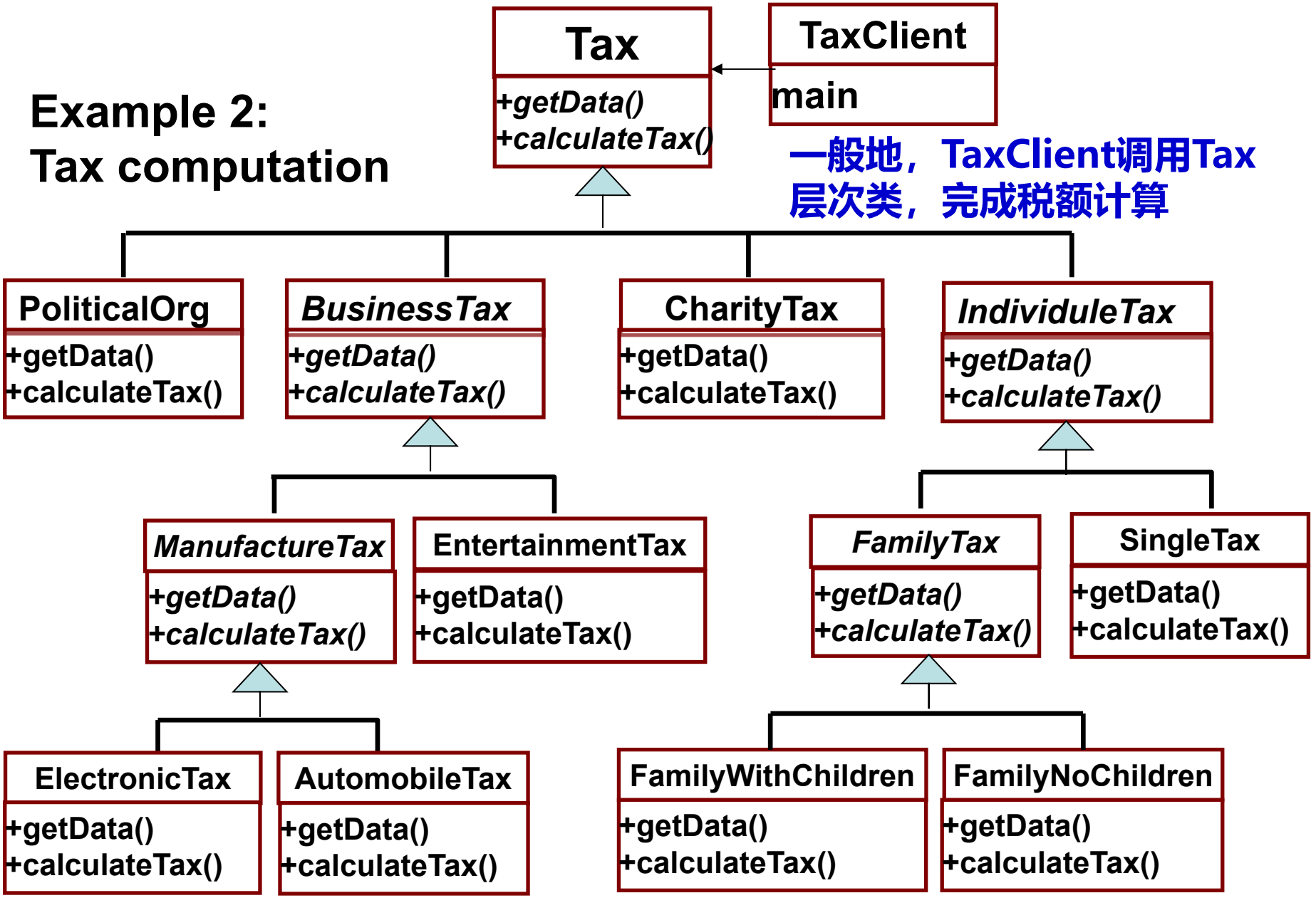
网络游戏开发团队决定合理地利用Animal层次类

Example 2: tax computation problem (税收问题)

【例2】: tax computation problem (税收问题)

- In the US, there are many kinds of taxes
- The taxes can be expressed by using a hierarchy of classes as below.
- For each class, the tax rate and the algorithm to calculate tax are different

Example 2:
Tax computation



First design- use a class to call the tax hierarchy to calculate tax

Example 2: tax computation problem (税收问题)

本设计的缺点。

Problems in this design (drawbacks):

- Distributing all these operations across the various node classes leads to a system that's hard to
  - 理解understand,
  - 维护maintain, and
  - 改变change.

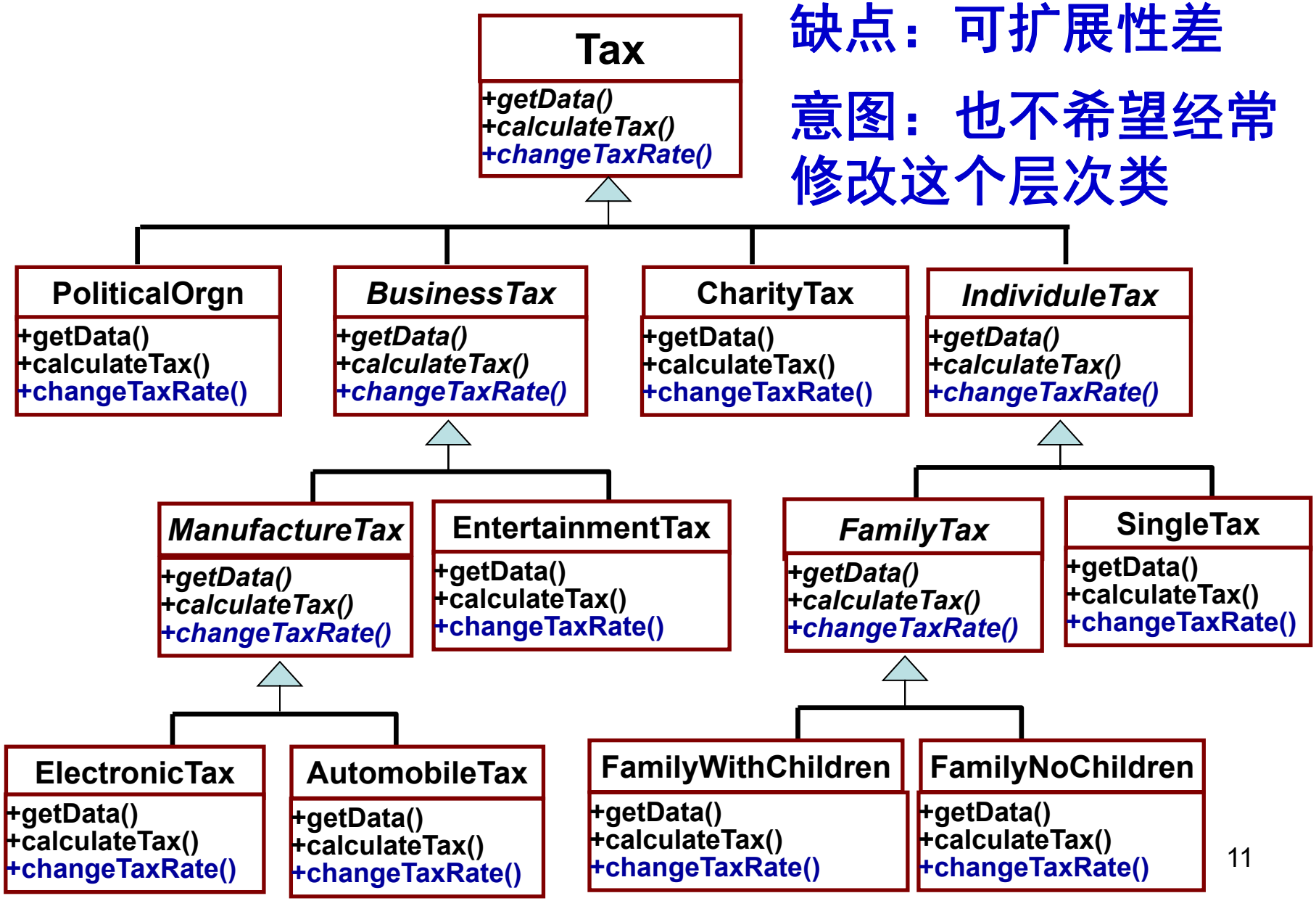
Example 2: tax computation problem (税收问题)

Problems (drawbacks) (cont):

For example,

- **可维护性差. Poor maintainability**: It will be confusing to have a lot of **various kinds** of code in so many nodes.
- **可扩展性差. Poor extendibility** : adding a new operation usually requires recompiling all of these classes,
- **See next page**

例如，在每个类中添加一个方法changeTaxRate()



Example 2: tax computation problem (税收问题)

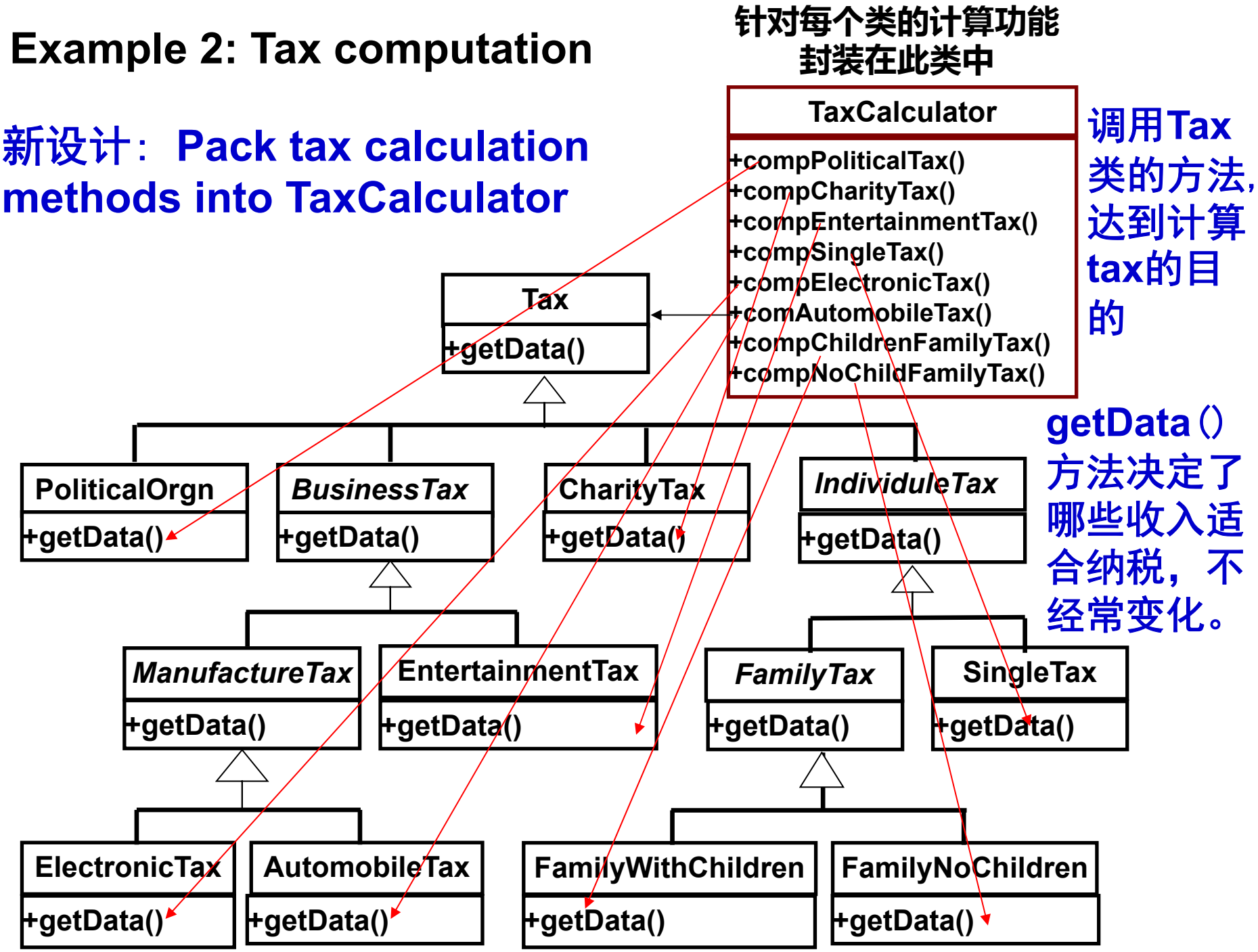
改善设计：将具体的计算税收的方法从层次类中分离出来，而在层次类中保留数据维护方法。

One possible solution to improve these:

- **Separate the main functionalities of the classes, i.e., let the "Tax" class be independent of the operations that apply to them.**

Example 2: Tax computation

新设计：**Pack tax calculation methods into TaxCalculator**

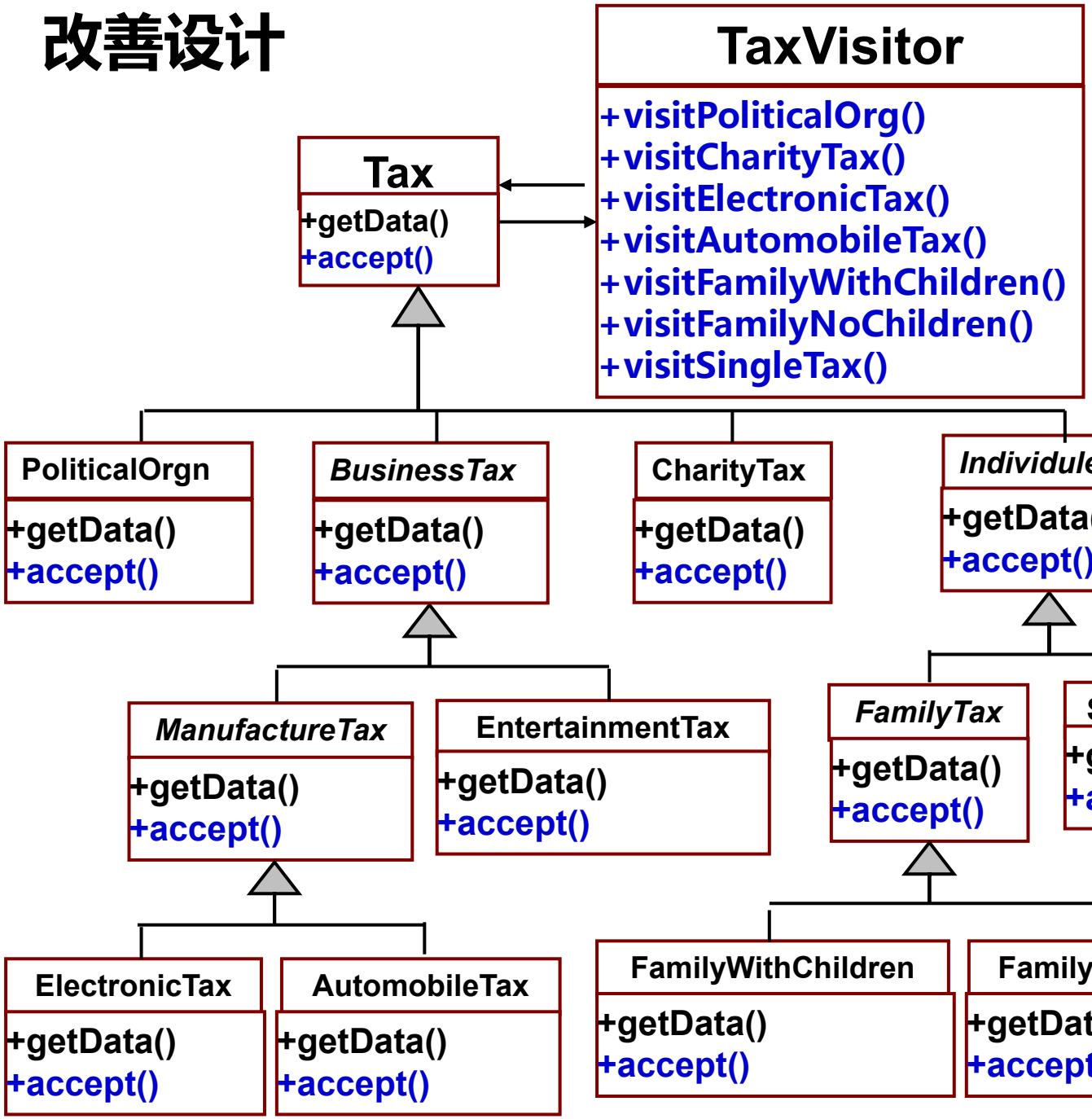


Example 2: tax computation problem (税收问题)

- 进一步改善设计 (More Improved design)

- In order for the above design more effective, we change the above design a little bit
- The new design is in the next page

改善设计



改名：1) 将类名改为TaxVisitor

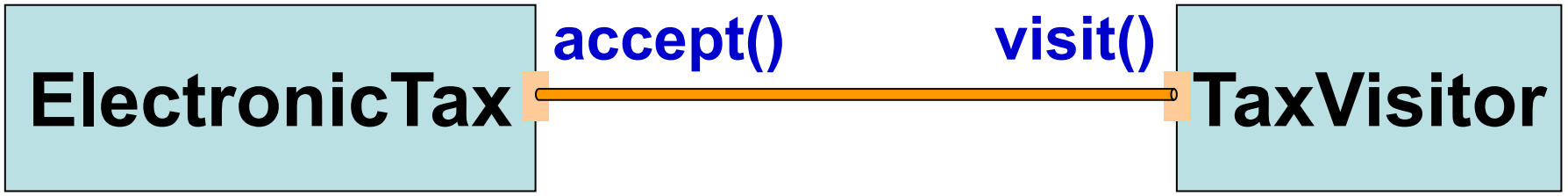
2) 方法名改为visitxxx()，即为每个Tax类都单独设计一个访问方法，以获得相应的Tax信息。

3) 在Tax层次类中增加accept()方法

Example 2: tax computation problem (税收问题)

为什么要引入访问者方法与接受方法：

意图：利用accept()与visit()方法，在Tax层次类与TaxVisitor中都精心地设计接口



Example 2: tax computation problem (税收问题)

怎样设计接口呢？

四人帮利用accept方法建立两个类之间的接口。

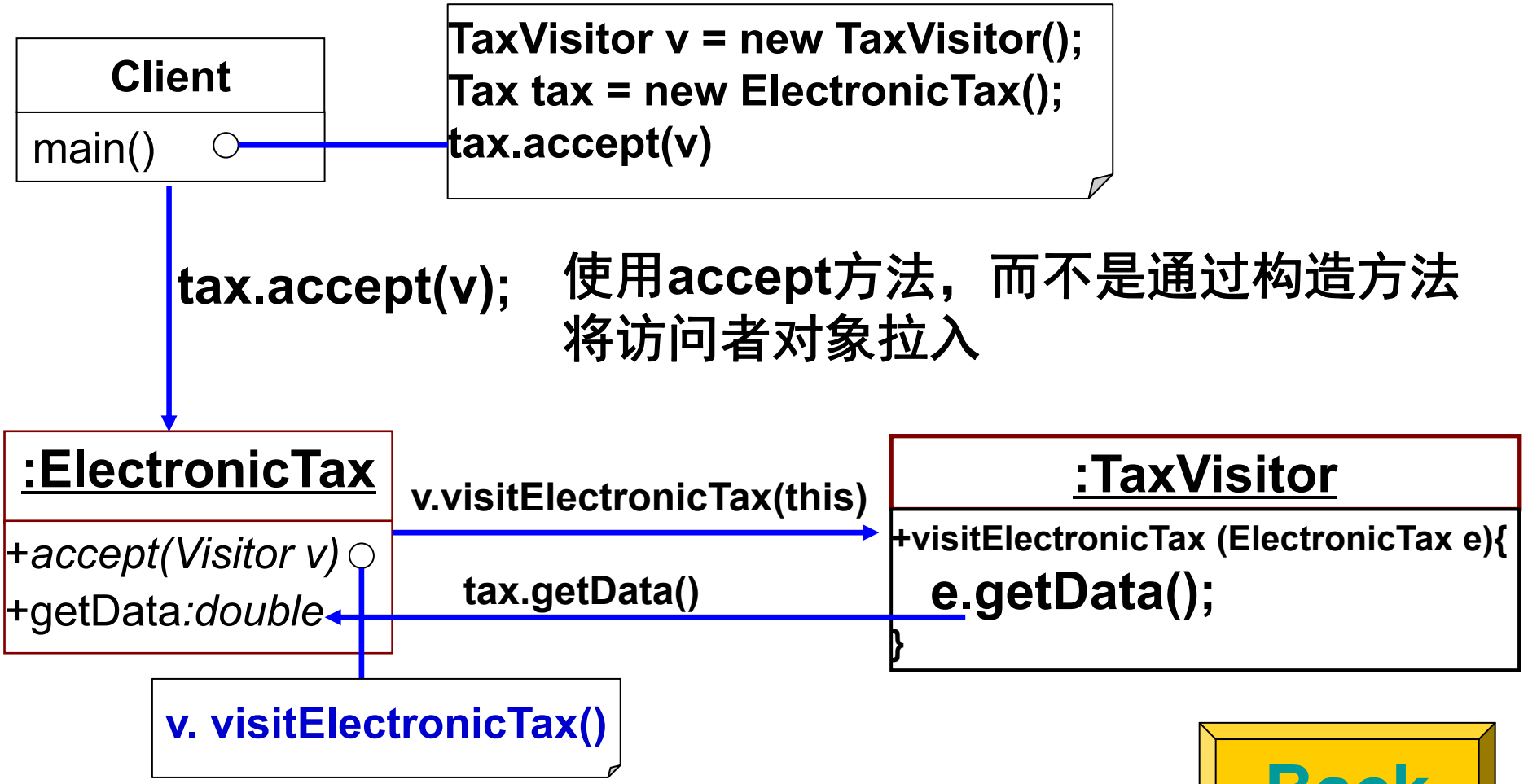
四个软件天才--四人帮（Gof）照片

类ElectronicTax 中的accept 方法

```
accept(TaxVisitor v)
    v. visitElectronicTax()
}
```

Example 2: tax computation problem (税收问题)

两个对象是怎样被链接在一起的呢？

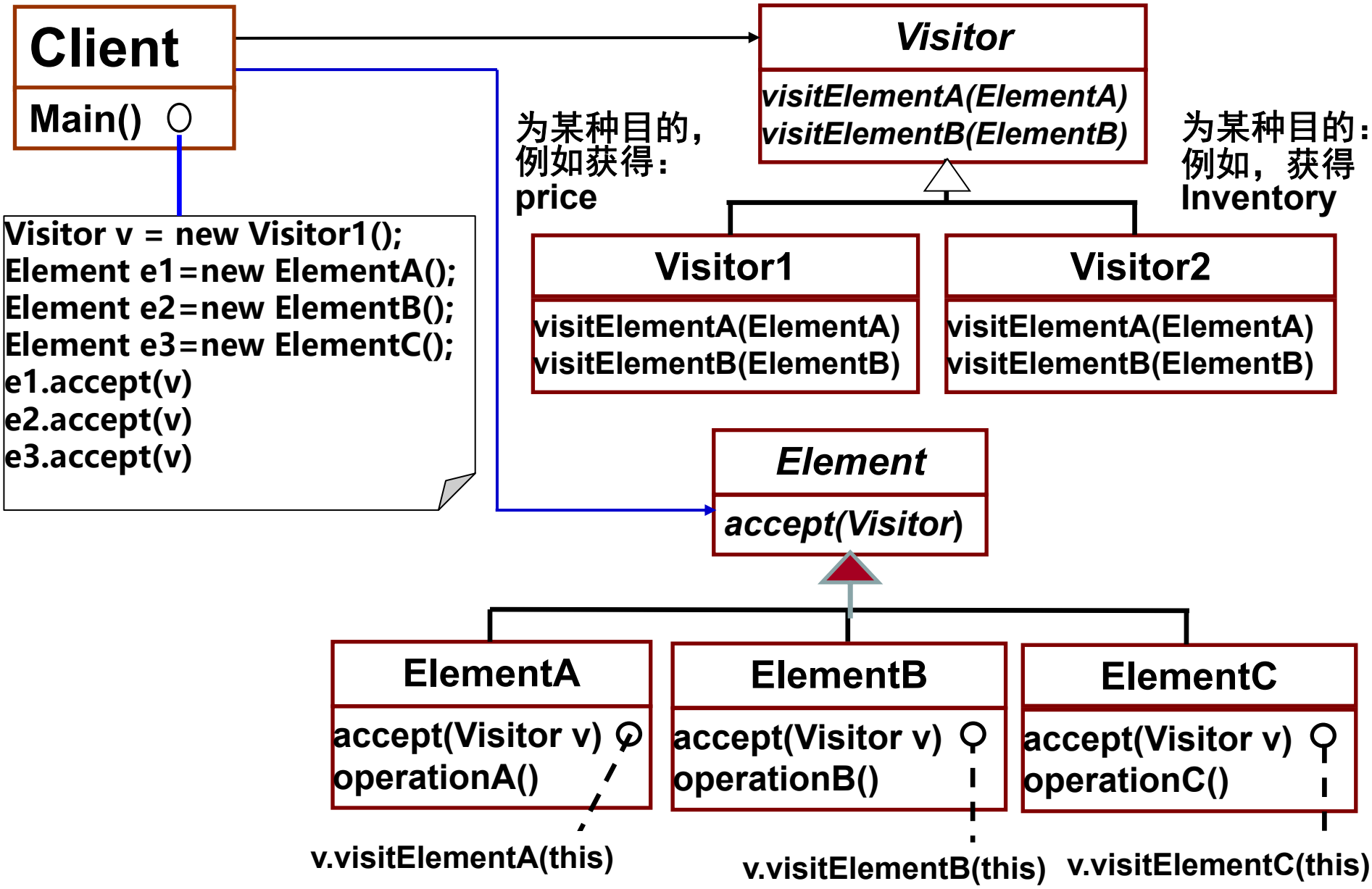


Back

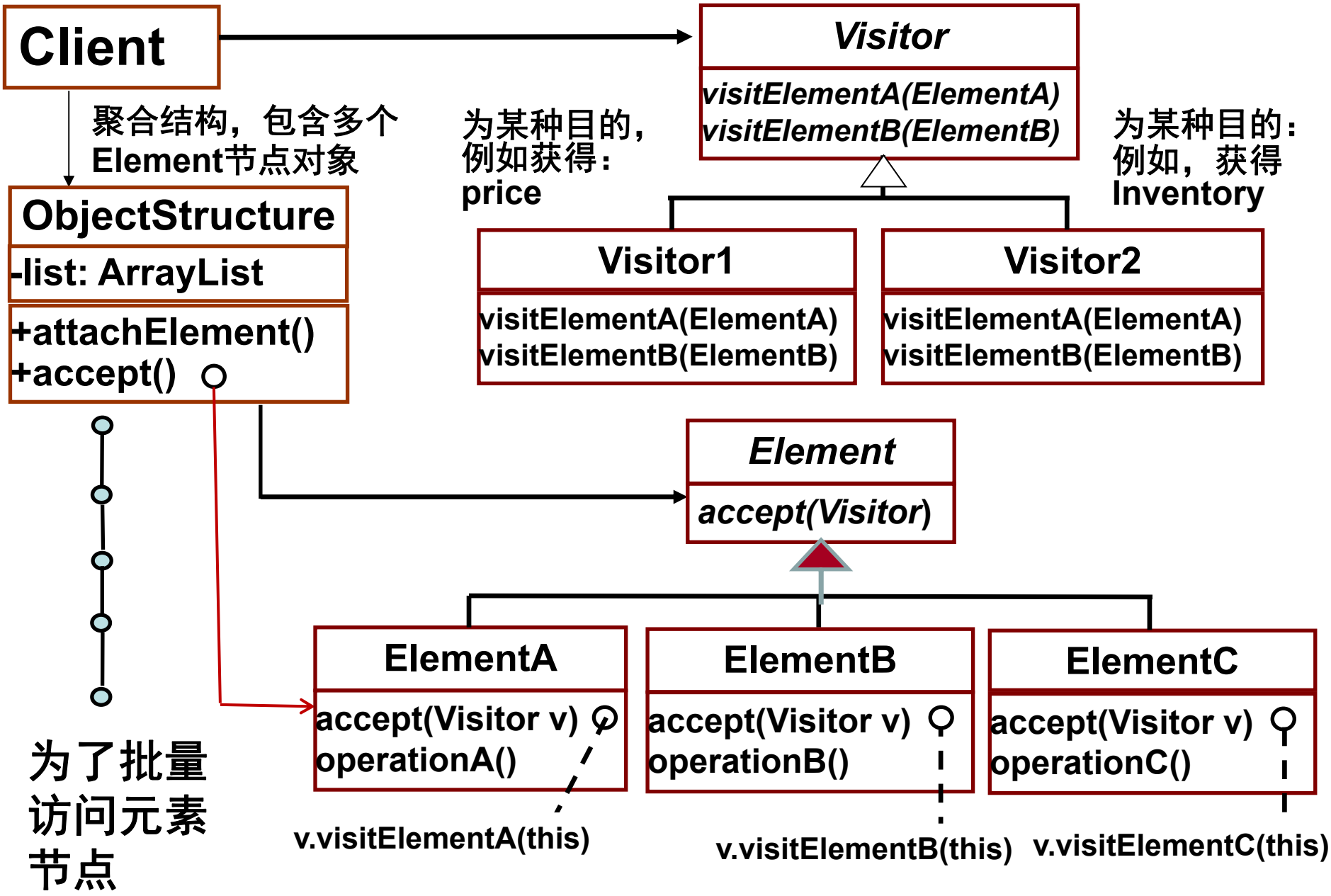
Interactions involved in the above design

Official Diagram of the Visitor Pattern

访问者模式类图



Visitor pattern class diagram – 无聚合结构的情况



Visitor pattern class diagram

Visitor Pattern

访问者模式组件的解释

- 访问者类为每个Element子类准备了一个访问者方法。
  Visitor class declares a Visit operation for each class of Element in the object structure.
  - The operation's name and signature identifies the class that sends the Visit request to the visitor. For example, method visitHardDisk
- 访问者类决定要访问哪个类。Let the visitor determine the concrete class of the element being visited. Then the visitor can access the element directly through its particular interface.

Visitor Pattern

访问者模式构件的解释

Visitor1, Visitor2

- 实现Visitor类的接口中所声明的所有操作.
  Implements each operation declared by Visitor.
  - ◆Each operation implements a fragment of the algorithm defined for the corresponding class of object in the structure.
- 提供算法环境、存储局部状态、积累遍历结构体的结果.
  Provides the context for the algorithm and stores its local state, which often accumulates results during the traversal of the structure.

Visitor Pattern

Element

- **定义accept()方法. Defines an Accept operation that takes a visitor as an argument.**

ElementA, ElementB

- **实现accept()方法. Implements an "accept" operation that takes a visitor as an argument.**
- **还可以实现一些其它方法。May also implement some methods to help the visitor to realize some functionalities. For example, getData in Tax hierarchy**

Visitor Pattern

ObjectStructure (Program) (can be omitted)

- **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.**

- **该类可以使用一个聚合结构例如 ArrayList或者Vector将被访问对象集合在一起，然后再使用一个循环语句批量调用accept方法。**

Back

Design Examples Using the Visitor Pattern

利用访问者模式进行设计的例子

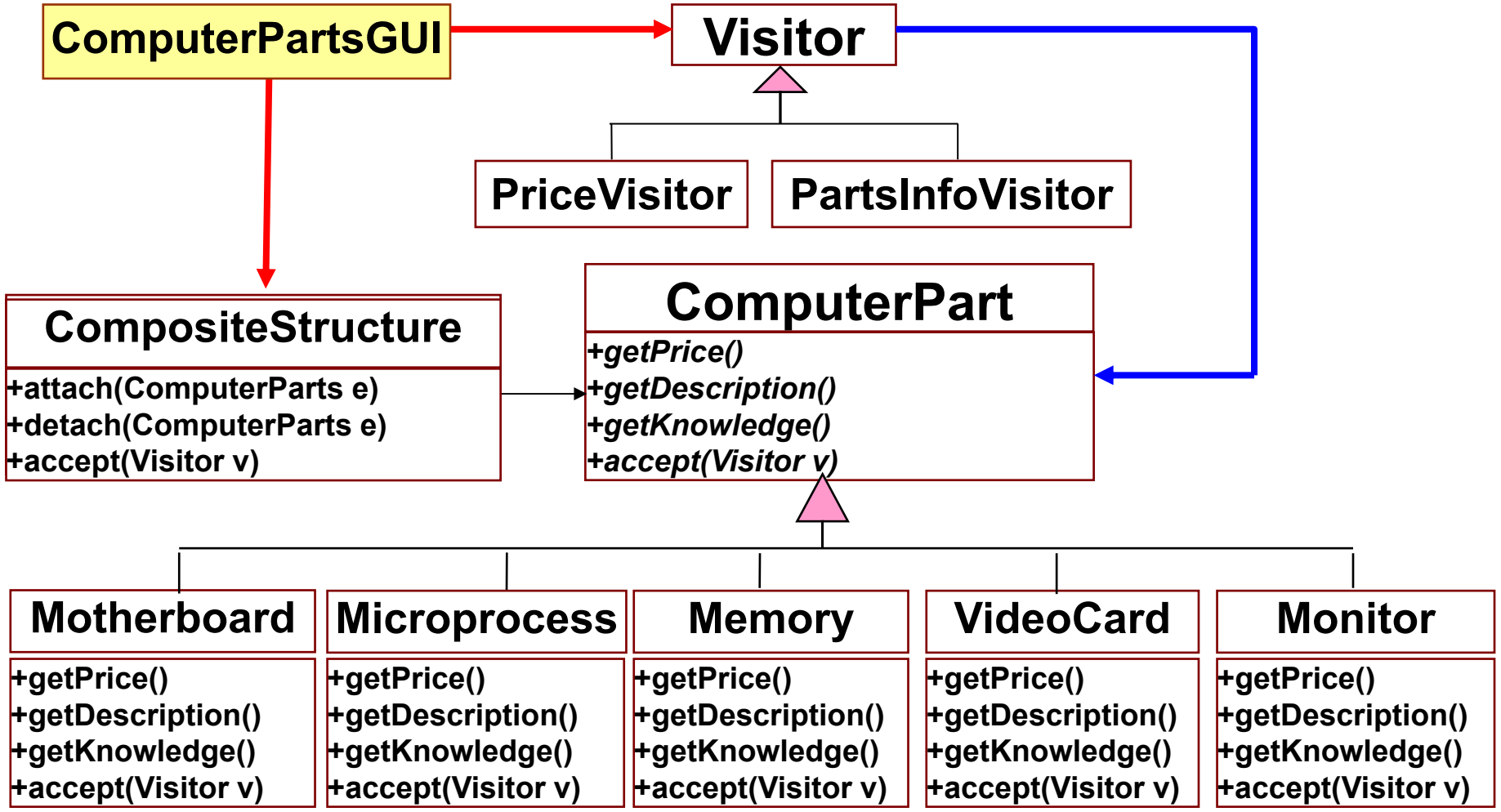
计算机部件销售的例子

【例3】需求：计算机部件销售的例子。

- 假设要写一个为计算机商店使用的计算机部件销售软件。
- 该软件要求用户使用GUI选择要购买的部件，提交，然后程序分别列出各个部件的名称与单价，以及总价格。

设计：考虑到计算机部件的种类相对固定，所以我们采取使用访问者模式进行设计。设计类图如下。

计算机部件销售的例子



利用访问者模式设计的计算机部件销售软件系统

设计要点：

ComputerPart类

- getPrice()：返回一个部件的价格
- getDescription(): 返回一个部件的规格、型号
- getKnowledge(): 解释一个部件的功能
- accept(Visitor v)：接受某个特定的访问者访问

Visitor层次类

- PriceVisitor通过accept()方法，调用getPrice()方法，达到计算价格的目的。
- PartsInfoVisitor类负责调用getDescription方法，实现获取部件的规格、型号

计算机部件销售的例子

- 本设计使用了CompositeStructure类。
- ComputerPartsGUI类根据用户请求提交的计算机部件名称，
  - 创建各个相应的部件对象，然后，
  - 将这些对象添加到复合对象CompositeStructure中。

- CompositeStructure对象负责接受访问者PriceVisitor对象，对该聚合对象所包含的各个对象访问。

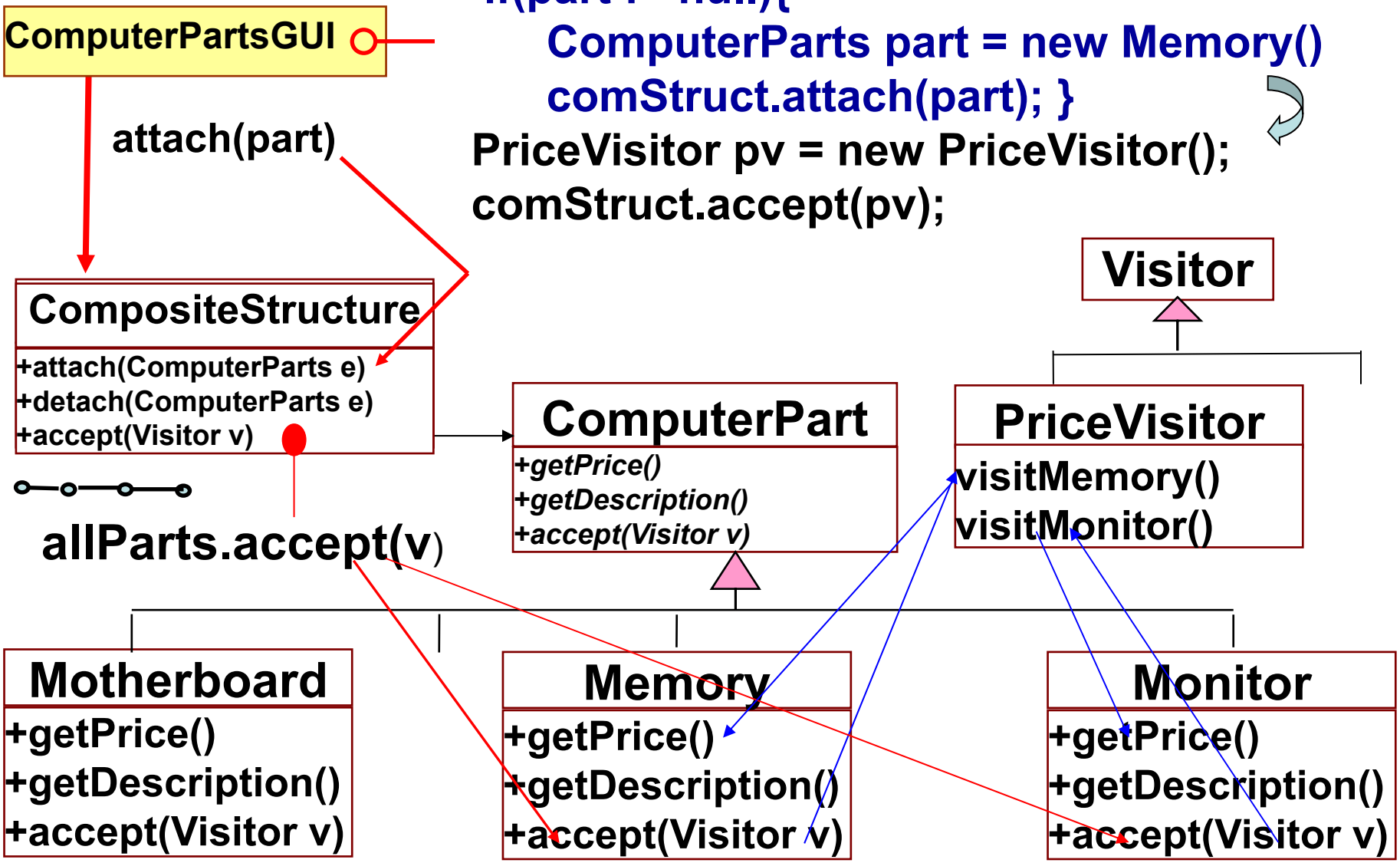
计算机部件销售的例子

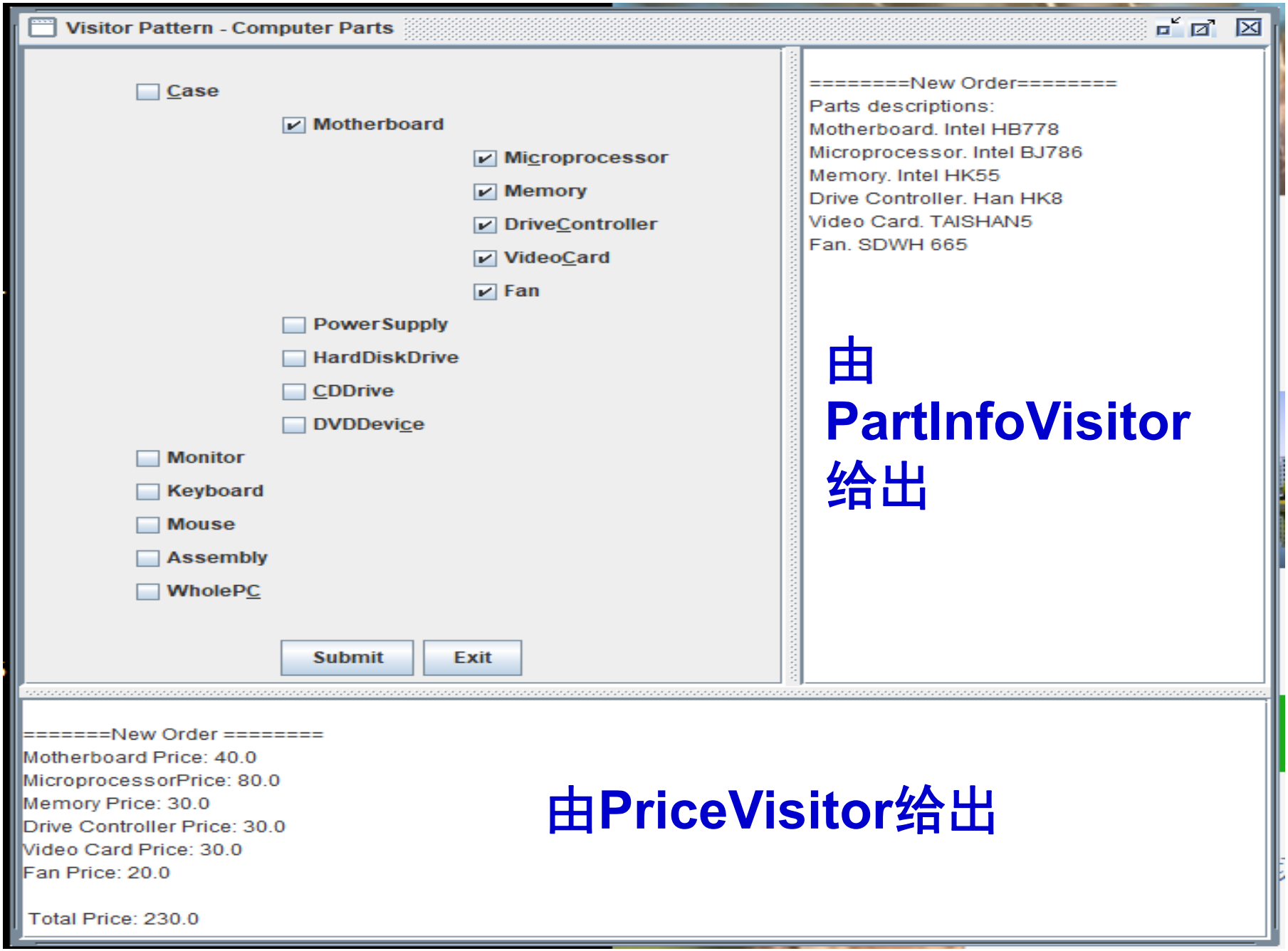
关于CompositeStructure：

- 在CompositeStructure类中，封装了长度可以弹性增长的数据结构ArrayList，也可用Vector。
- 在CompositeStructure对象中，可以添加许多等待访问的对象，而不必关心数组越界问题。
- 类CompositeStructure负责“批量”接受访问者访问所需要被访问的对象。

典型交互

```
comStruct = new CompositeStructure();
While(true){
 if(part != null){
     ComputerParts part = new Memory()
     comStruct.attach(part); }
PriceVisitor pv = new PriceVisitor();
comStruct.accept(pv);
```




Visitor Pattern - Computer Parts
Case
Motherboard
Microprocessor
Memory
DriveController
VideoCard
Fan
PowerSupply
HardDiskDrive
CDDrive
DVDDevice
Monitor
Keyboard
Mouse
Assembly
WholePC
Submit
Exit
========New Order========
Parts descriptions:
Motherboard. Intel HB778
Microprocessor. Intel BJ786
Memory. Intel HK55
Drive Controller. Han HK8
Video Card. TAISHAN5
Fan. SDWH 665
由
PartInfoVisitor
给出
=======New Order ========
Motherboard Price: 40.0
MicroprocessorPrice: 80.0
Memory Price: 30.0
Drive Controller Price: 30.0
Video Card Price: 30.0
Fan Price: 20.0
Total Price: 230.0
由PriceVisitor给出

- 在以上客户程序的代码实现的中，首先创建了CompositeStructure类的对象，
- 然后调用了该类的方法accept(PriceVisitor pv) 与accept(InventoryVisitor iv)。
- 在以上的accept方法中，采用了一个循环语句，但是没有条件语句。也就是说，CompositeStructure类的对象负责将等待访问的Part对象保存，但是并不关心确切地哪个对象被保存了。
- 当需要接受访问者的时候，使用一个循环语句，遍历所有的待接受访问的对象，使得每个对象都接受访问。

计算机部件销售的例子-代码

在下面用户图形界面源代码中，省略了大量的语句

```
public class ComputerPartsGUI extends JFrame implements ItemListener{
  private void createPartObjAndVisitParts(ActionEvent e){
        ComputerParts part = null;
        PriceVisitor pv = new PriceVisitor();
        PartsInfoVisitor iv = new PartsInfoVisitor();
        CompositeStructure comStruct = new CompositeStructure();

        //循环语句
        if(part != null){
           comStruct.attach(part);


       comStruct.accept(pv);
       comStruct.accept(iv);
    }
}
```

计算机部件销售的例子-代码

```
public class CompositeStructure{
    private ArrayList<ComputerParts> parts;
    public CompositeStructure(){
            parts = new ArrayList<ComputerParts>();
    }
    public void attach(ComputerParts equip){
      if(equip != null)
            parts.add(equip);
    }
     public void detach(ComputerParts equip){
        if(equip != null)
                parts.remove(equip);
    }
    public void accept(Visitor v){
            int len =parts.size();
            for (int i=0; i < len; i++){
                        ComputerParts part = parts.get(i);
                        part.accept(v);
      }
    }
}
```

计算机部件销售的例子-代码

```
public abstract interface ComputerPart {
   public abstract void accept(Visitor vis);
   public abstract String getName();
   public abstract double getPrice();
   public abstract String getDescription();
}
```

计算机部件销售的例子-代码

```
public class Microprocessor implements ComputerPart{
    public static final String NAME = "Microprocessor";
    private final double PRICE = 80.00;
    public static final String FEATURES = "Microprocessor. Intel BJ786";

    public String getName(){
        return NAME;
    }
    public double getPrice(){
        return PRICE;
    }
    public String getDescription(){
            return FEATURES;
    }
    public void accept(Visitor v){
       System.out.println("Microprocessor has been visited.");
       v.visitMicroprocessor(this);
    }
}
```

计算机部件销售的例子-代码

```
public abstract interface Visitor{
    public abstract void visitComputerCase(ComputerCase e);
    public abstract void visitPowerSupply(PowerSupply e);
    public abstract void visitMotherboard(Motherboard e);
    public abstract void visitMicroprocessor(Microprocessor e);
    public abstract void visitMemory(Memory e);
    public abstract void visitDriveController(DriveController e);
    public abstract void visitHardDiskDrive(HardDiskDrive e);
    public abstract void visitCDDrive (CDDrive  e);
    public abstract void visitDVDDevice (DVDDevice  e);
    public abstract void visitMonitor (Monitor  e);
    public abstract void visitKeyboard (Keyboard  e);
    public abstract void visitMouse (Mouse  e);
    public abstract void visitFan (Fan  e);
    public abstract void visitVideoCard(VideoCard e);
}
```

计算机部件销售的例子-代码

本类省略了大量代码

```
public class PriceVisitor implements Visitor{
    private double total =0 ;
    public double price = 0;
    ArrayList<Double> partsPrices;
    public PriceVisitor()  {
         partsPrices = new ArrayList<Double>();
    }
    public void visitComputerCase(ComputerCase e) {
         price = e.getPrice();
         partsPrices.add(new Double(price));
         total += price;
   }
  public ArrayList<Double>  getPartsPrices() {
      return partsPrices;
    }
   public double getPriceTotal(){
      return total;
    }
}
```

Back

Further discussion of the Visitor Pattern

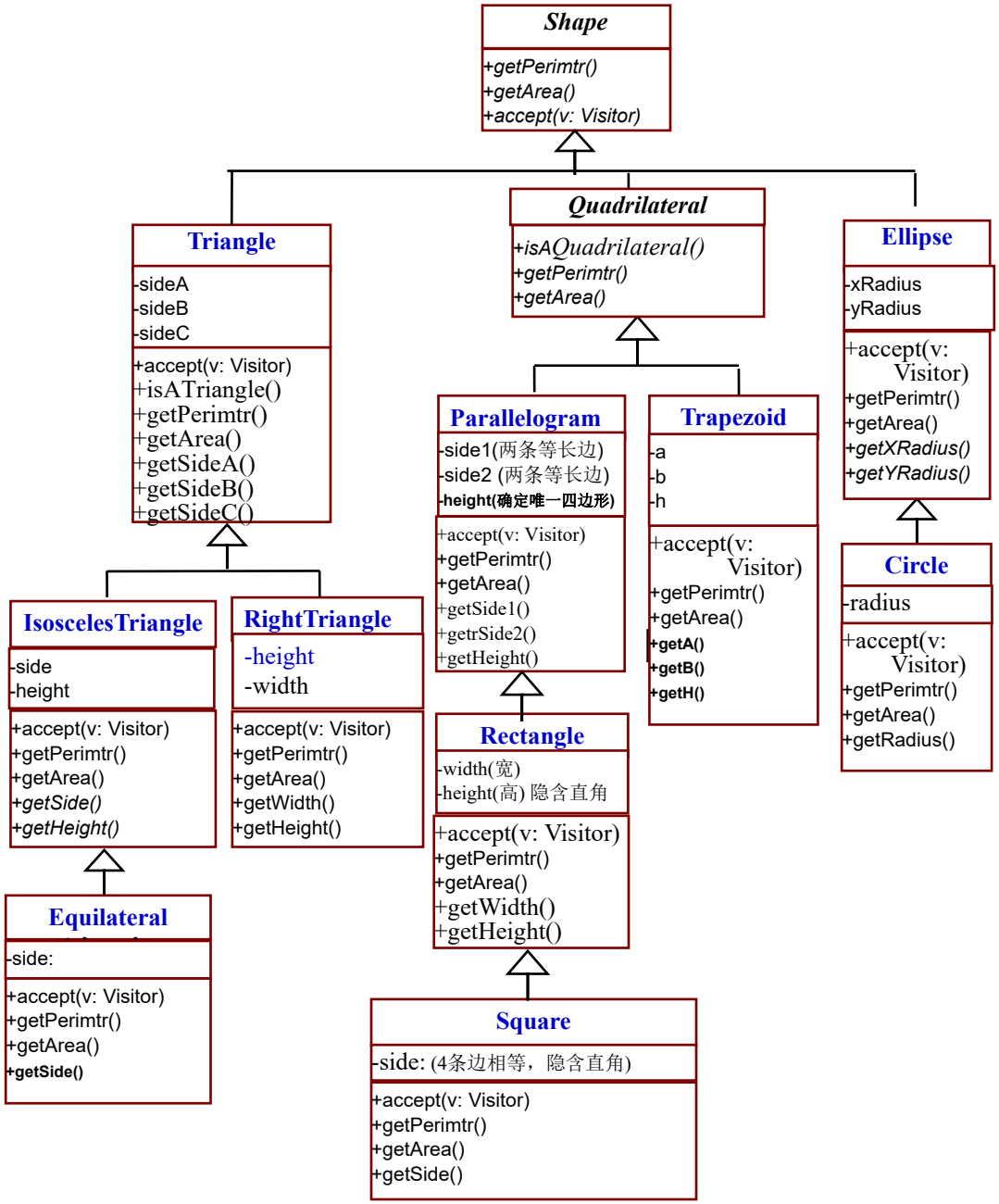
Visitor Pattern-when to use？

何时使用访问者模式？

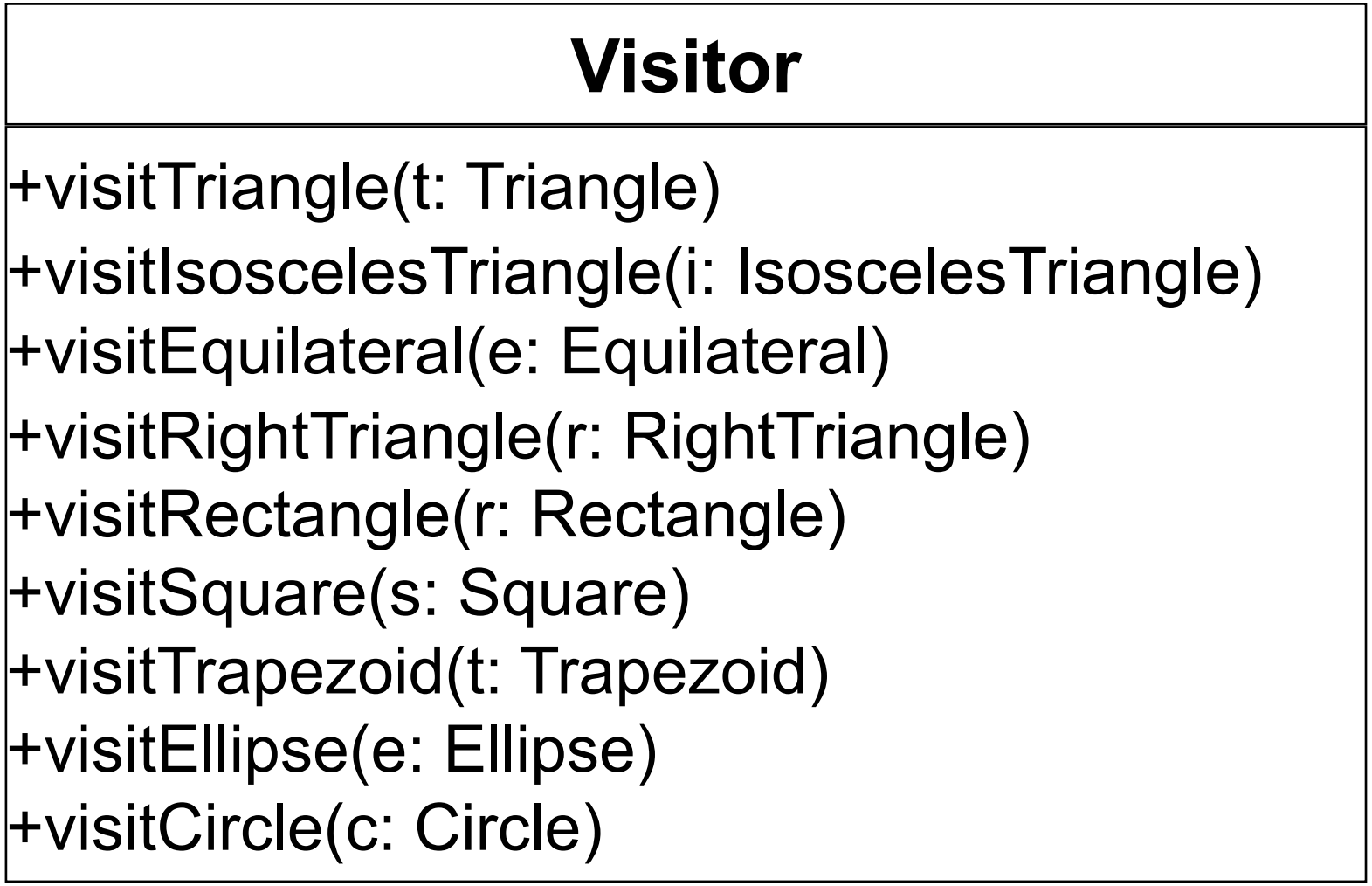
Use the Visitor pattern when

1) 对象结构包含很多个子类，而这些类有不同的接口。你需要针对不同的类进行不同的操作。

   An object structure contains many classes of objects with differing interfaces, and you want to **perform operations** on these objects that depend on their concrete classes.



- 对象结构包含很多个子类，
- 这些类有不同的接口。
- 你需要针对不同的类进行不同的操作。



- **在这些Visit方法中，你可以随心所欲地调用层次类中的相关子类的任何方法。**

- 若你需要调用层次类的共同接口的时候，例如，若要调用

  +getPerimtr()

  +getArea()

  方法的时候，
- 你可以在Visitor类中只使用一个Visit方法即可，层次类子类对象可以通过参数传入，然后，利用传入的对象，调用以上的两个方法即可。

Visitor
+visit(s: Shape)

其中，参数s可以是任何形状的对象，例如：Square对象。

Visitor Pattern-when to use？

2)许多不同的且不相干的操作需要施加于对象结构，而你想要避免这些不同的操作污染这些类。

Many distinct and unrelated operations need to be performed on objects in an object structure, and you want to avoid "polluting" their classes with these operations.

- **Visitor lets you keep related operations together by defining them in one class (e.g., compute total price).**
- **When the object structure is shared by many applications, use Visitor to put operations in just those applications that need them.**

Visitor Pattern-when to use？

3)对象结构类很少改变(例如税收的种类很少改变)，但是你需要经常地在这些结构体上增加新的运算(税率可能经常改变)。

The classes defining the object structure rarely change, but you often want to define new operations over the structure.

– Changing the object structure classes requires redefining the interface to all visitors, which is potentially costly.

Visitor Pattern-Collaborations

注:

1. A client that uses the Visitor pattern must create a ConcreteVisitor object and then traverse the object structure, visiting each element with the visitor.
2. When an element is visited, it calls the Visitor operation that corresponds to its class. The element supplies itself as an argument to this operation to let the visitor access its state, if necessary.

Visitor Pattern-Consequences

访问者模式的优点 （Advantages）

1. **Visitor makes adding new operations (in the Visitor) easy**. Visitors make it easy to add operations that depend on the components of complex objects. You can define a new operation over an object structure simply by adding a new visitor.
   - **In contrast, if you spread functionality over many classes, then you must change each class to define a new operation.**

Visitor Pattern-Consequences

Advantages

2. **A visitor gathers related operations and separates unrelated ones**. (例如,将计算总价格放在一个具体的visitor类中)
   - **Related behavior isn't spread over the classes defining the object structure; it's localized in a visitor.**
   - **Unrelated sets of behavior are partitioned in their own visitor subclasses.**

Visitor Pattern-Consequences

访问者模式的缺点（Disadvantages）

1. **Adding new ConcreteElement classes is hard.**

- Each new ConcreteElement gives rise to a new abstract operation on Visitor and a corresponding implementation in every ConcreteVisitor class.

ConcreteElement➔functionalities

讨论：访问者模式的重点

- **什么是访问者模式的重点?**
- 回答: 用于使得被访问者与访问者之间建立自动访问的accept方法是重点。该方法建立了两个层次类之间的关联。该方法使得自动访问成为可能。

- 注: 另外,类objectStructure类对于批量访问很重要。另外使用此类，可以使得客户类包含较少的条件语句。能够有效地做到责任分离。

- 结论: 要写好**accept方法**与objectStructure

Back