11. Modeling a Class's Internal Structure: Composite Structures

Shaoning Zeng, http://zsn.cc

What we learnt?

- 8. Focusing on Interaction Links: Communication Diagrams
- 9. Focusing on Interaction Timing: Timing Diagrams
- 10. Completing the Interaction Picture:
  Interaction Overview Diagrams

11. Modeling a Class's Internal Structure: Composite Structures

- 11.1. Internal Structure 内部结构
- 11.2. Showing How a Class Is Used 类的使用
- 11.3. Showing Patterns with Collaborations

Composite Structures 复合结构

- Composite structures show how objects create a big picture.
  - They model how objects work together inside a class,
  - or how objects achieve a goal. 协作完成一个目标
- Composite structures are fairly advanced, but they're good to have in your bag of tricks because they are perfectly suited for specific modeling situations, such as showing: 适合用于一些特定的场合
  - Internal structures 内部结构
  - Ports 端口
  - Collaborations 协作

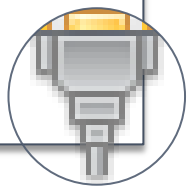
Specific modeling situations 特殊建模情况

- Show the parts contained by a class and the relationships between the parts; this allows you to show context-sensitive relationships

Internal structures

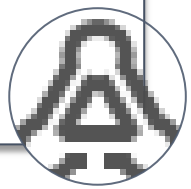
- Show how a class is used on your system with ports
  如何通过端口使用各个类

Ports



- Show design patterns in your software and, more generally, objects cooperating to achieve a goal
  设计模式

Collaborations



11.1. Internal Structure

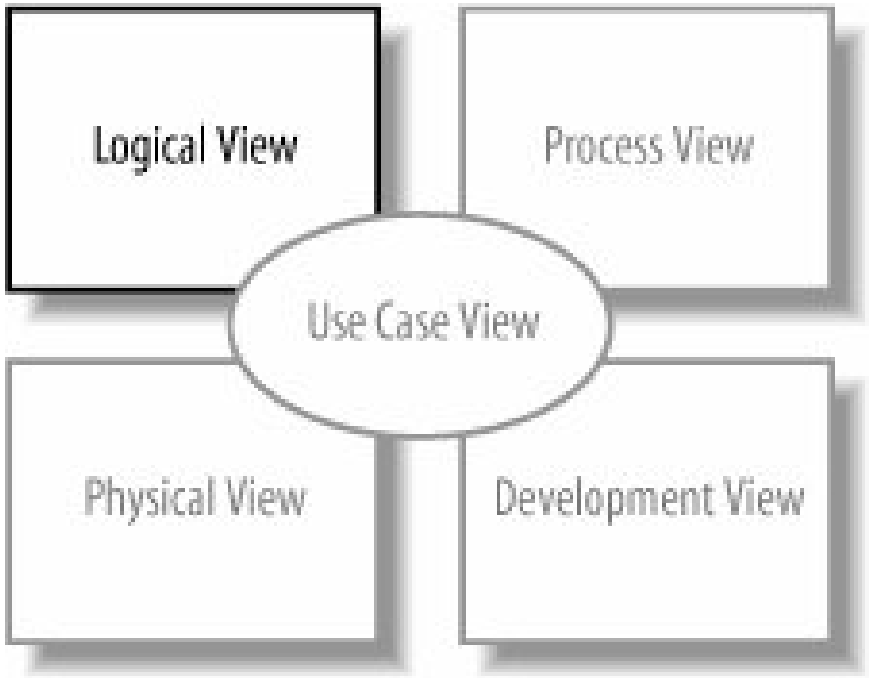


Figure 11-1.The logical view captures the abstract descriptions of a system's parts, including composite structures

11.1.1. When Class Diagrams Won't Work

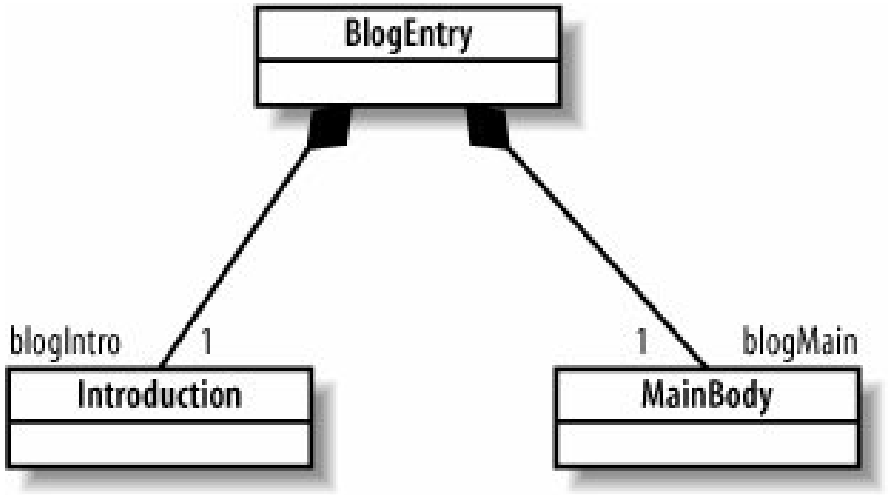


Figure 11-2. Class diagram showing that BlogEntry contains Introduction and MainBody

Figure 11-3. This first pass at showing that a blog entry's introduction introduces its main body doesn't quite work

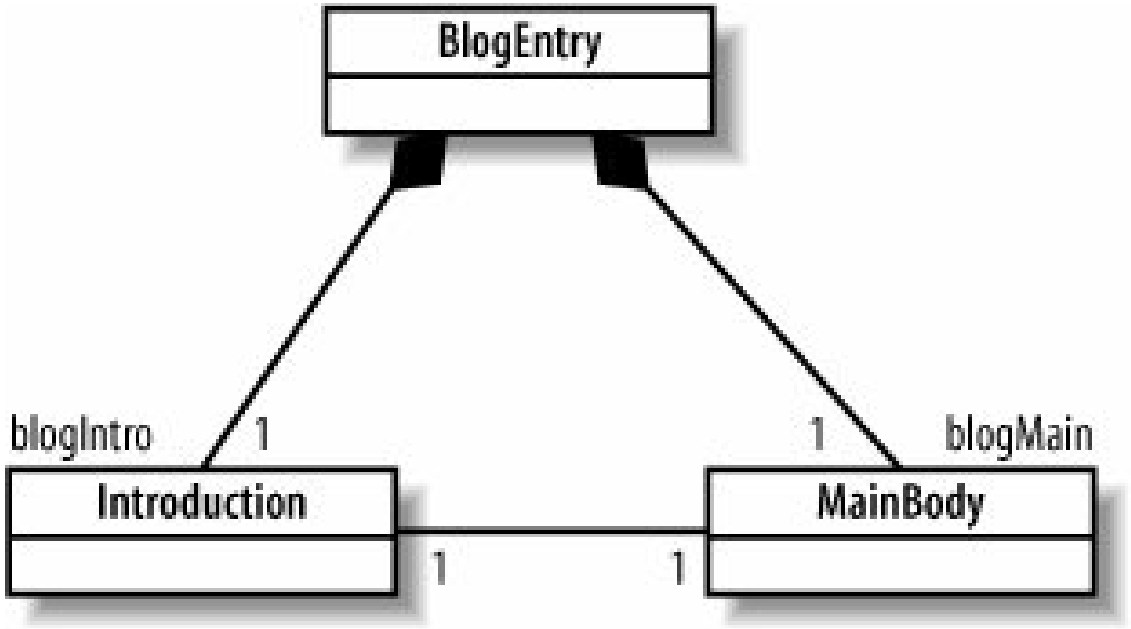


Figure 11-4. Unintended but valid object structure

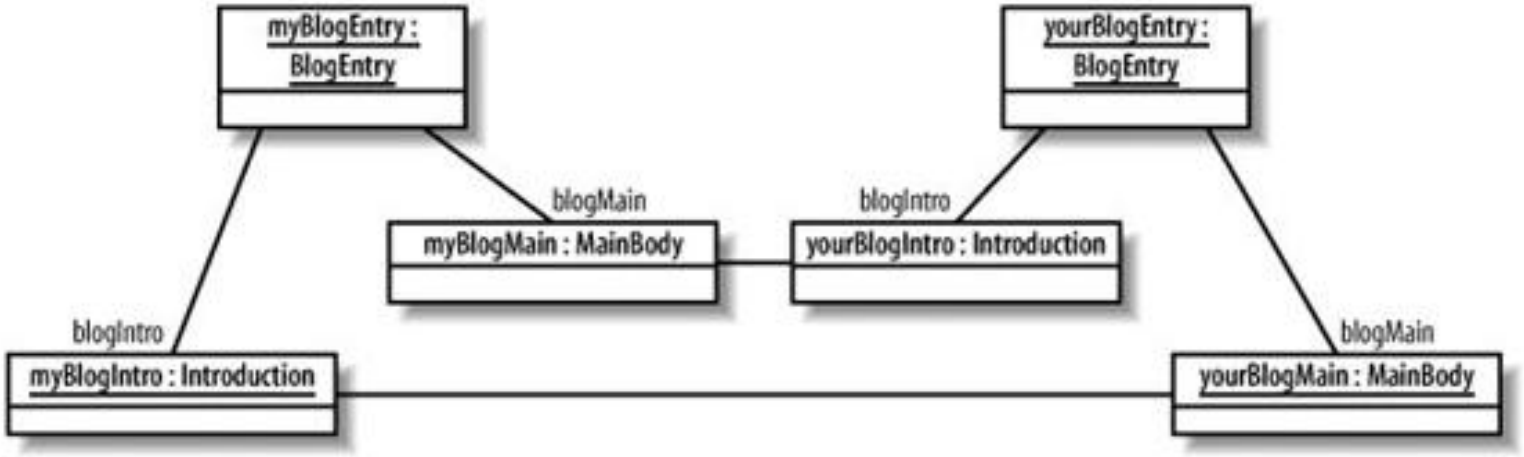
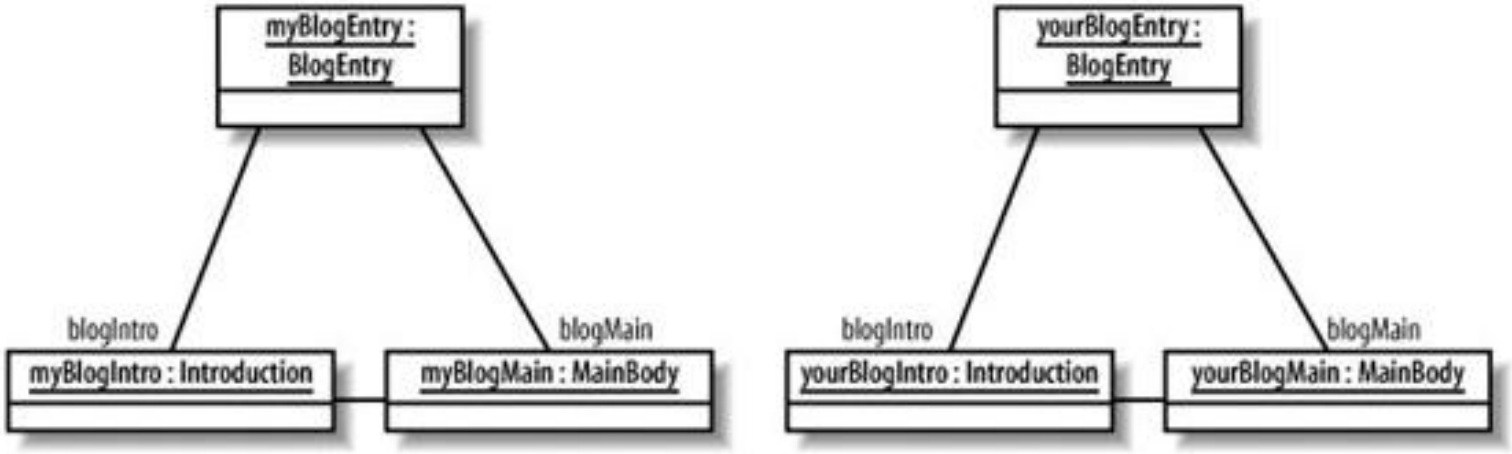


Figure 11-5. This was the intended object structure



11.1.1. When Class Diagrams Won't Work

- It turns out that class diagrams are not good at expressing how items contained in a class relate to each other. 不擅长表示类中元素与其他类中元素的关系
- This is where internal structure comes in: it allows you to specify relationships in the context of the class that contains them. 内部结构：类内部元素的关系
- But keep reading the internal structure notation is a convenient and simple way to show relationships between contained items, especially when the contained items have complex relationships.

11.1.2. Parts of a Class

- Its contained items are now drawn directly inside, instead of connected through filled diamond arrowheads.

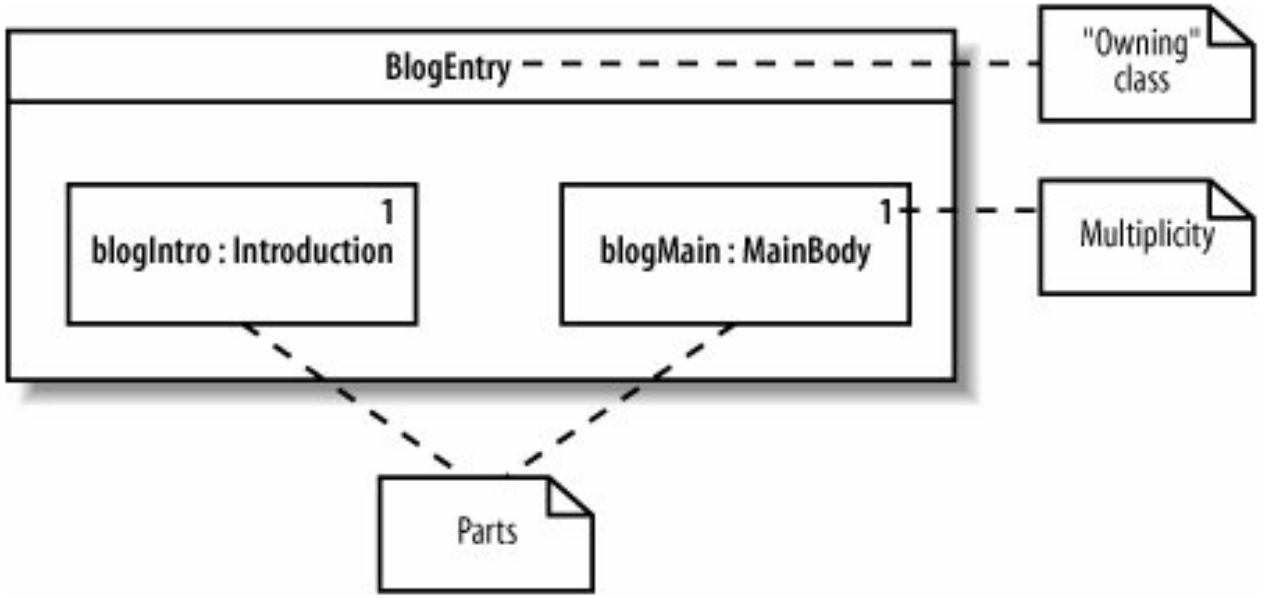


Figure 11-6. The internal structure of the BlogEntry class

11.1.2. Parts of a Class

- When showing the internal structure of a class, you draw its *parts*, or items contained by composition, inside the containing class. 位于类之内
- Parts are specified by the role they play in the containing class, written as *<roleName> : <type>*.
  - In Figure 11-6, the part of type Introduction has the role blogIntro and the part of type MainBody has the role blogMain.
- The *multiplicity*, or number of instances of that part, is written in the upper righthand corner of the part.
- A part is a set of instances that may exist in an instance of the containing class at runtime. 运行时概念

Figure 11-7. How the internal structure of BlogEntry matches up to the class diagram

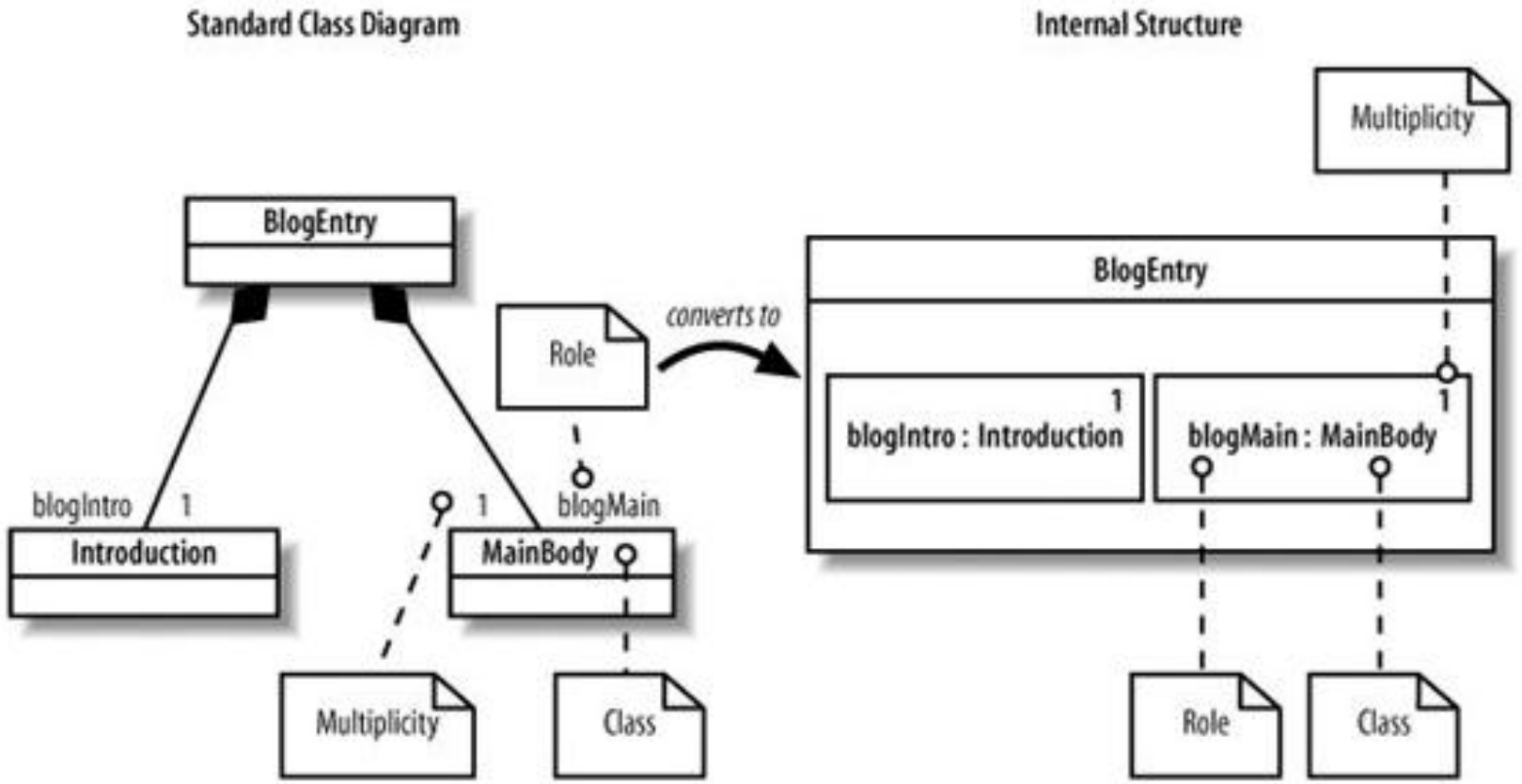
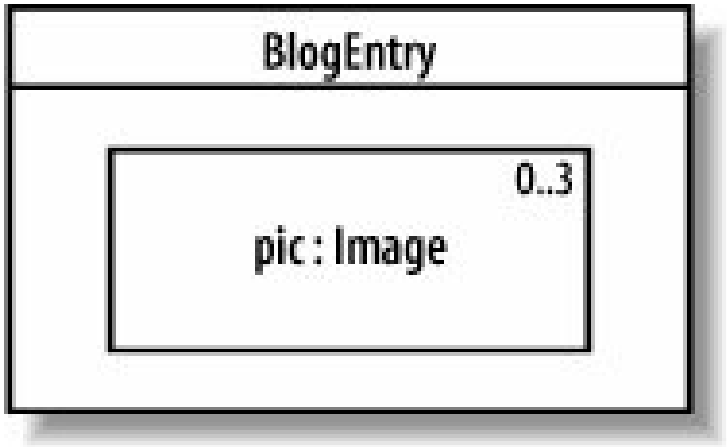


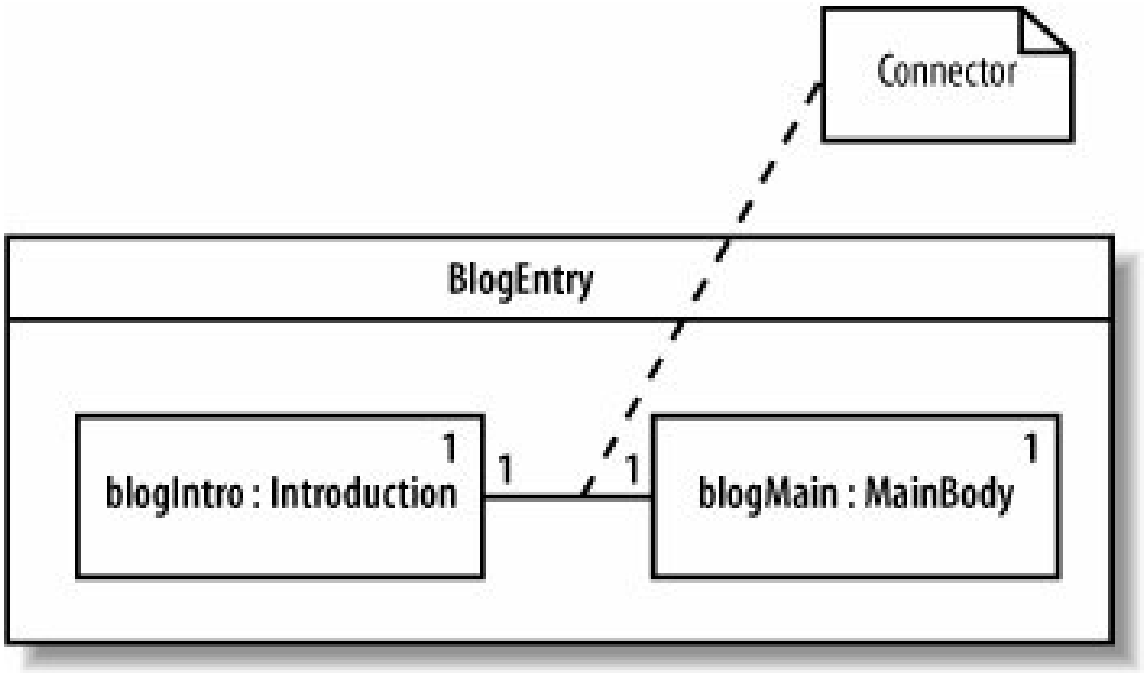
Figure 11-8. In the internal structure of BlogEntry, the part with role pic has a multiplicity of zero to three



11.1.3. Connectors 连接器

- A connector is a link that enables communication between parts. 各部分通信的链路
  - A connector simply means that runtime instances of the parts can communicate.
  - A connector can be a runtime instance of an association or a dynamic link established at runtime, such as an instance passed in as a parameter. 关联/动态链接
- A connector applies only to the parts it's connected to in Figure 11-9, that means only the set of instances that will exist in an instance of BlogEntry.
  - You can now be certain that an introduction introduces the main body in the same blog entry as the introduction.

Figure 11-9. Using connectors to link parts in the internal structure of a class



11.1.4. Alternate Multiplicity Notations

Figure 11-10. Equivalent notations for multiplicity

11.1.5. Properties 属性

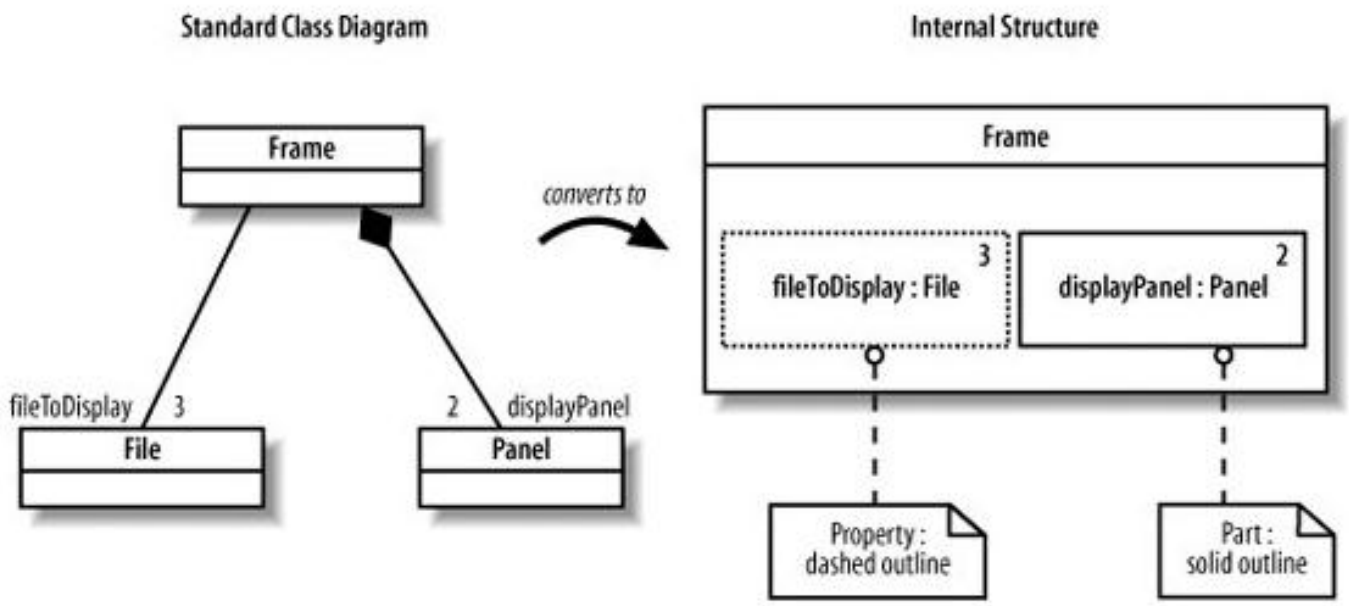
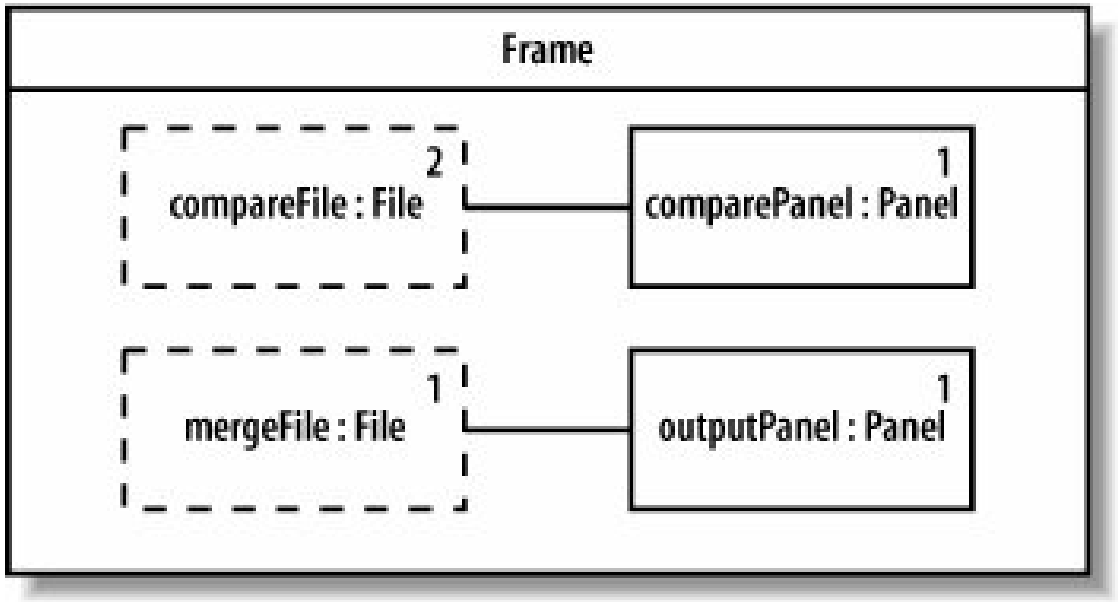


Figure 11-11. Parts and properties in the internal structure of a class

11.1.6. Showing Complex Relationships Between Contained Items 复杂关系

- Showing a class's internal structure is especially useful when its contained items relate to each other in unusual ways.
  - Revisiting the merge tool example in Figure 11-11, suppose you want to explicitly model one panel displaying the two files being compared and the other panel displaying the merged file.
  - You can do this by defining more detailed roles for files and panels to show how they relate to each other within a frame, as shown in Figure 11-12.
  - Figure 11-2 demonstrates that there can be parts (or properties) of the same type playing different roles. Internal structures help make these roles and their relationships explicit. 同一种类型的部分，发挥不同的作用

Figure 11-12. A more detailed internal structure diagram that specifies how files and panels relate to each other within a frame



11.1.7. Internal Structure Instances

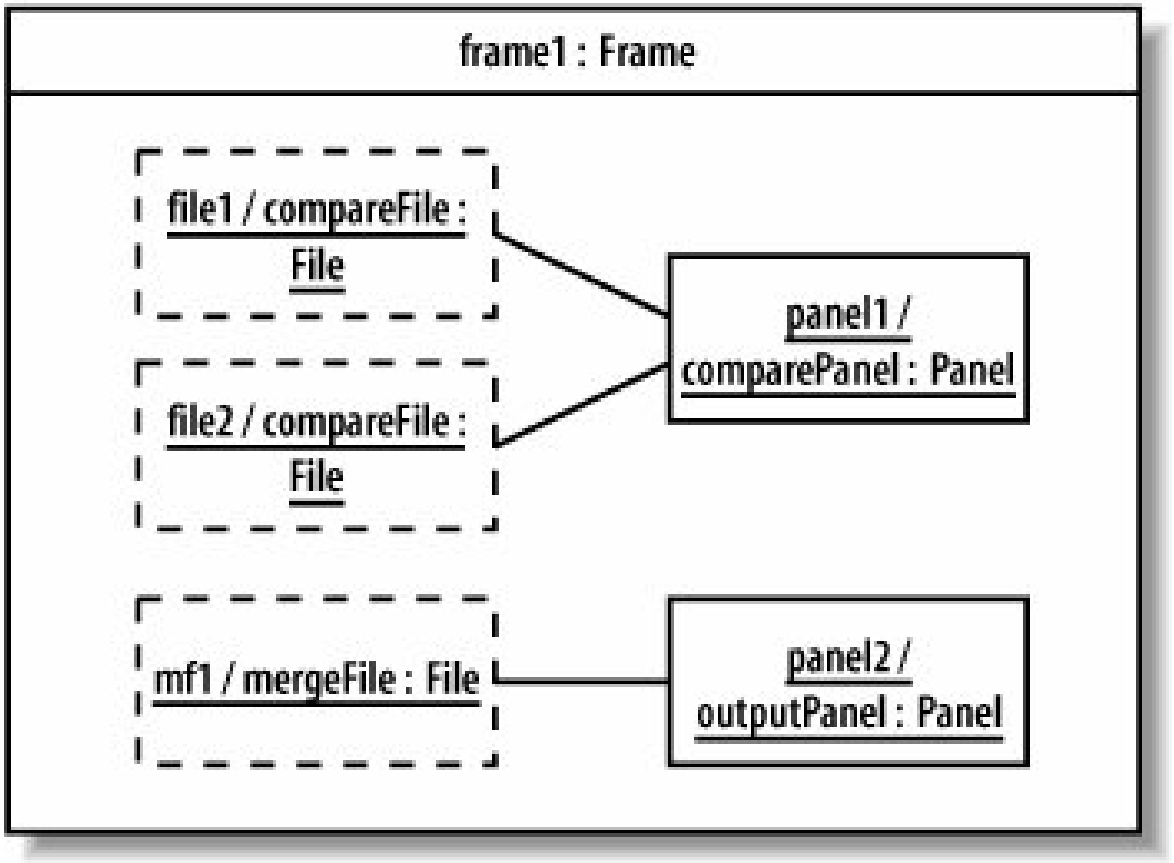


Figure 11-13. An instance of Frame with instances of its contained parts

11.2. Showing How a Class Is Used

- The internal structure of a class focuses on the contents of a class; ports focus on the outside of a class, showing how a class is used by other classes.
- A port is a point of interaction between a class and the outside world. It represents a distinct way of using a class, usually by different types of clients.
- For example, a Wiki class could have two distinct uses:
  - Allowing users to view and edit the Wiki
  - Providing maintenance utilities to administrators who want to perform actions such as rolling back the Wiki if incorrect content is provided

11.2. Showing How a Class Is Used

- Each distinct use of a class is represented with a port, drawn as a small rectangle on the boundary of the class, as shown in Figure 11-14. 不同用途用端口表示
  - Write a name next to the port to show the purpose of the port.
- It's common for classes to have interfaces associated with ports. You can use ports to group related interfaces to show the services available at that port.
  使用端口分组相关的接口

11.2. Showing How a Class Is Used

- Recall from Chapter 5 that a class can realize an interface, and this relationship can be shown using the ball interface symbol.
  - When a class realizes an interface, the interface is called a *provided interface* of the class. 提供的接口
  - A provided interface can be used by other classes to access the class through the interface.
- Similarly, classes can have *required interfaces*. 需要的接口
  - A required interface is an interface that the class requires to function.
  - More precisely, the class needs another class or component that realizes that interface so it can do its job.
  - A required interface is shown with an open lollipop, or the socket symbol.

Figure 11-14. A class with two ports showing that the class provides UserServices and Maintenance capabilities

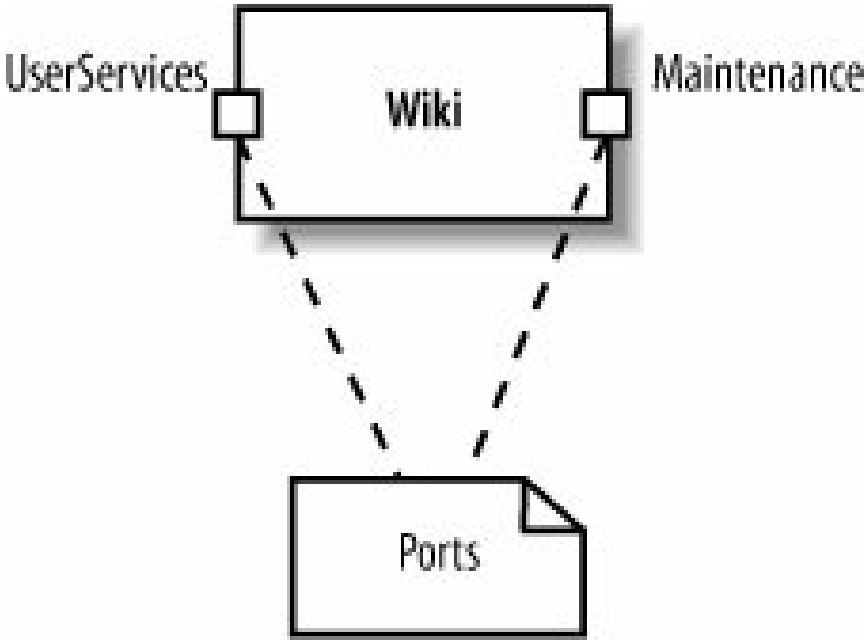
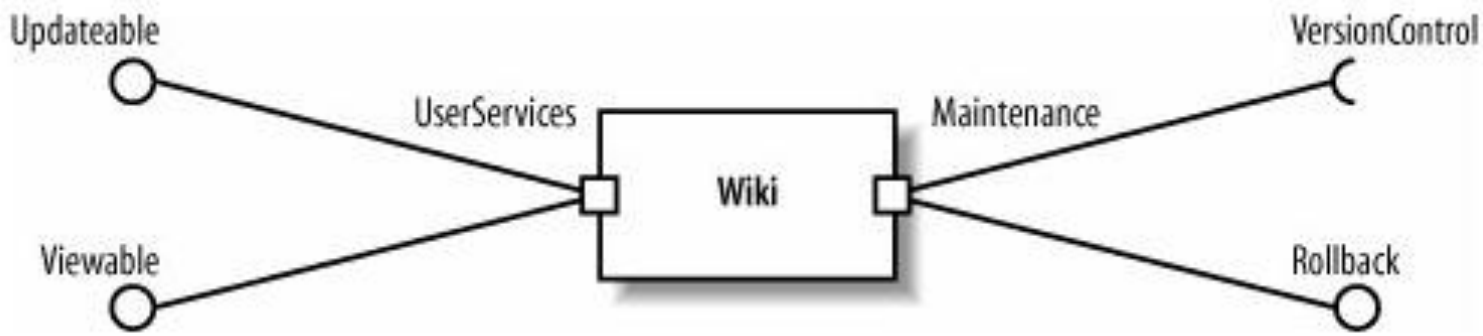


Figure 11-15. Ports can be used to group "like" interfaces



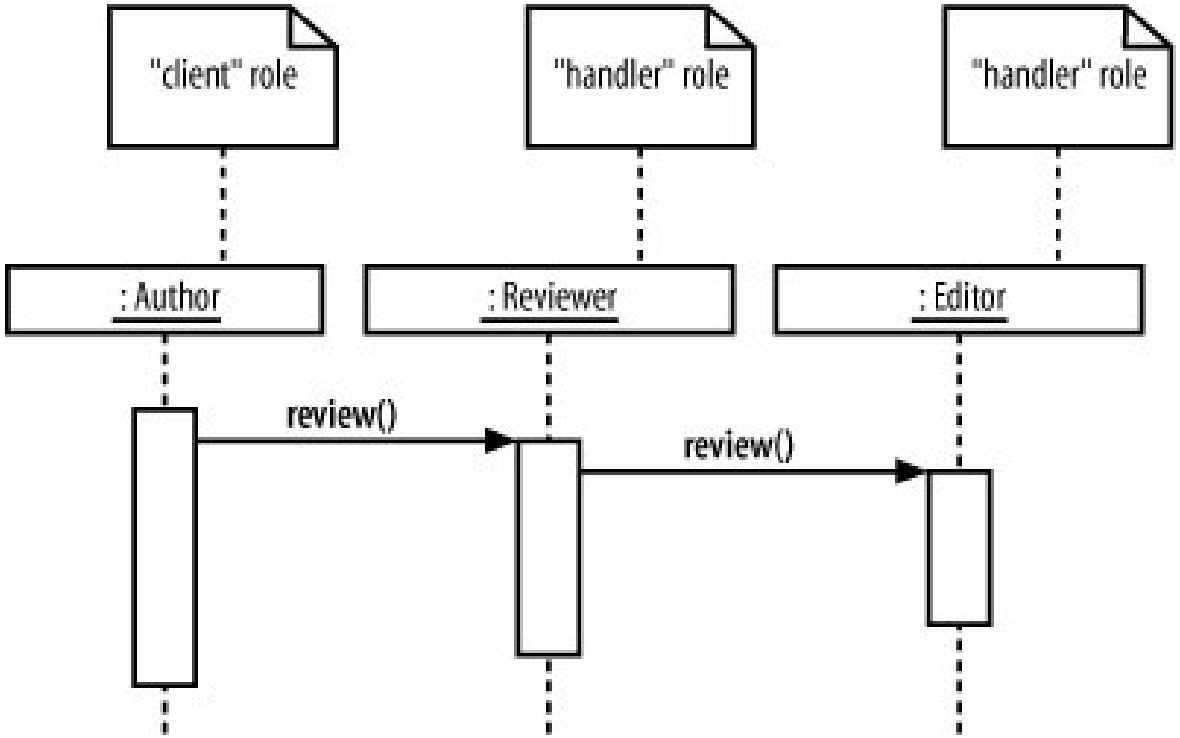
11.3. Showing Patterns with Collaborations

- Collaborations show objects working together, perhaps temporarily, to get something done. 协作
  - This may sound like object diagrams (see Chapter 6), but collaborations have a different focus: describing objects by the role they play in a scenario and providing a high-level textual description of what the objects are doing.
- Collaborations are a good way to document *design patterns* , which are solutions to common problems in software design. 设计模式
  - Observer and Observable in the Java API are an implementation of the Observer design pattern a way for an object to receive notification that another object changed.

Example

- Let's consider a problem in the CMS design that can be solved with a design pattern, which we'll then model using collaborations.
  - Suppose the CMS requires a content approval process: the author submits content, the reviewer may reject the content or pass it on to the editor, and the editor may reject or accept the content.
  - You decide to implement this flow with the Chain of Responsibility (COR) design pattern. 责任链
  - The COR design pattern allows an object to send a request without worrying about which object will ultimately handle the request. In the COR pattern, the *client* submits the request, and each *handler* in the chain decides whether to handle the request or to pass the request on to the next handler.
  - In the content approval process, the author will play the role of client, while the reviewer and editor will each play the role of handler.

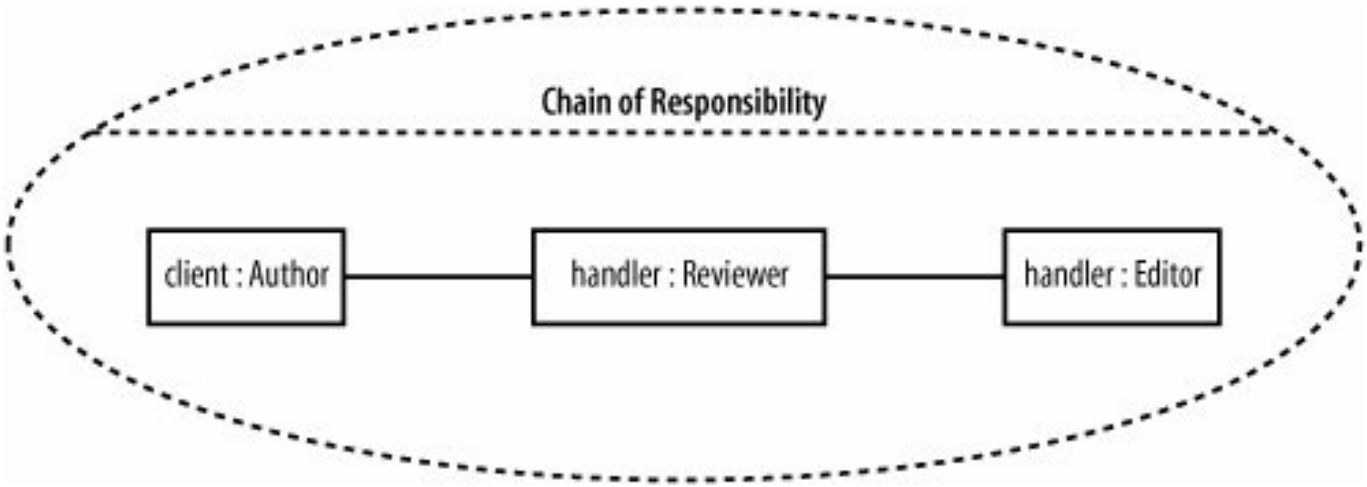
Figure 11-16. Sequence diagram showing how the COR pattern is used in the content approval process



There are two ways to model this pattern using collaborations.

- The first way uses a large dashed-lined oval with the collaboration participants drawn inside the oval.
  - You name a participant by the role it plays in the collaboration and its class or interface type, written as *<role> : <type>*.
  - The participants are linked together using connectors to show how they communicate.
  - The name of the collaboration is written inside the oval above a dashed line.
- Figure 11-17 shows a Chain of Responsibility collaboration using the first notation.
  - In this COR collaboration, the participant of type Author has the role client, and the other participants have the role handler.

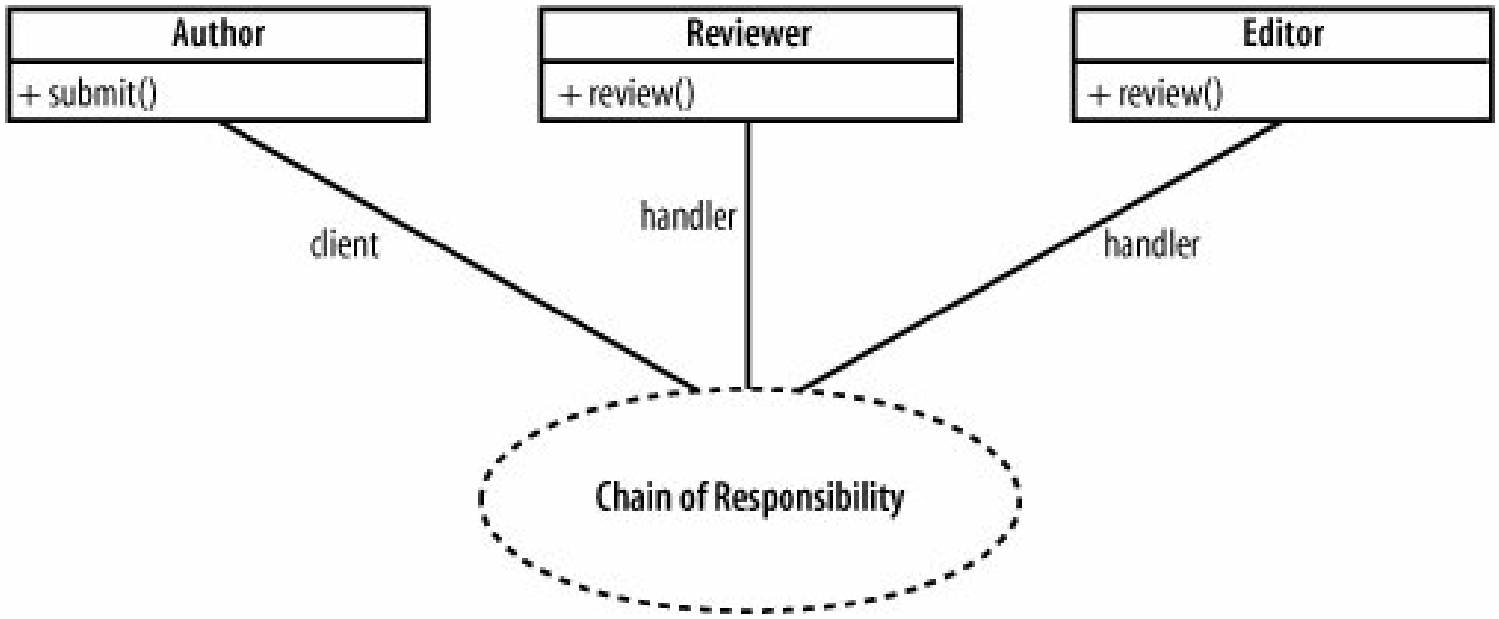
Figure 11-17. Collaboration showing the COR pattern in the content approval process



There are two ways to model this pattern using collaborations.

- The second way to draw a collaboration is shown in Figure 11-18.
  - In this notation, you show the class (or interface) rectangles of the participants, connecting each to a small collaboration oval.
  - Write the participants' roles along the lines.
  - This notation is useful for showing details of the class or interface, such as its operations.

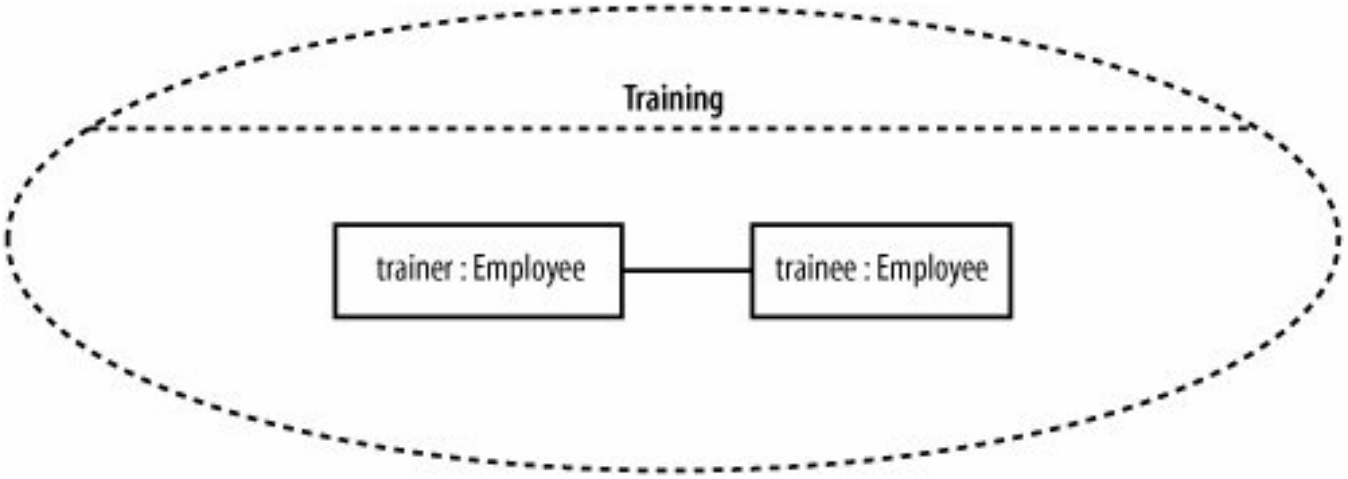
Figure 11-18. Alternate representation of the COR design pattern



Without Collaborations

- Collaborations may not look particularly useful, but their strength is in their ability to express patterns that may not be obvious from other diagrams, such as class or sequence diagrams. 主要用于描述模式
  - Without collaborations, you'd have to come up with your own technique to describe what's going on, such as the attached notes in Figure 11-16.
- Because collaborations are only temporary relationships, they have some interesting runtime properties that are best described with an everyday example of a collaboration.
  - Suppose a company has monthly training sessions in which the topic changes every session, and in every session the resident expert on the topic performs the training.
  - This is modeled as a Training collaboration that has participants with roles trainer and trainee, as shown in Figure 11-19.

Figure 11-19. This Training collaboration shows that the objects participating in a collaboration at runtime can interact with different collaborations in different ways



This training example illustrates the following points:

- An object isn't bound to its role in a collaboration; it can play different roles in different collaborations. 不同作用
  - Employee Ben and employee Paul remain the same objects; they're just playing different roles in different training collaborations.
- The objects in a collaboration aren't owned by the collaboration; they may exist before and after. 对象独立
  - Ben and Paul have a life outside of training.
- Even though objects in a collaboration are linked, they don't necessarily communicate outside the collaboration. 内部协作的对象，在外部不一定会通信
  - Ben and Paul may not talk to each other unless they absolutely have to during the training sessions.

Summary

- 11. Modeling a Class's Internal Structure: Composite Structures
  - 11.1. Internal Structure
  - 11.2. Showing How a Class Is Used
  - 11.3. Showing Patterns with Collaborations

Next …

- 12. Managing and Reusing Your System's Parts: Component Diagrams
  - 12.1. What Is a Component?
  - 12.2. A Basic Component in UML
  - 12.3. Provided and Required Interfaces of a Component
  - 12.4. Showing Components Working Together
  - 12.5. Classes That Realize a Component
  - 12.6. Ports and Internal Structure
  - 12.7. Black-Box and White-Box Component Views

See you …