4.13 面向对象系统设计

4.13.7 模式（Patterns）

- 问题的提出：面向对象技术要求使用者有正确的观念，熟练掌握分析与设计技能，才能充分发挥对扩充和重用的作用，但这对于一般技术人员而言，有一段较长的过程。
- 与软件技术面临的问题相近、可以为软件技术领域所借鉴的一个传统工业领域：建筑。
  - 与环境融为一体（如中山陵）；
  - 造型与风格（如悉尼歌剧院、陕西省历史博物馆）；
  - 适应现有条件（如卢浮宫金字塔）；
  - 结构（如香港中国银行）；
  - 质量（反例如北京西客站、重庆彩虹桥）；
  - 造价与工期;
  - ……

4.13 面向对象系统设计

4.13.7 模式

- 1964 年，著名建筑学家 Christopher Alexander 出版了一本书：*Notes on the Synthesis of Form*。在此书中，他提出了 Form（建筑形式）的概念，认为设计师可创造 Form 来化解环境中互相冲突的需求，使冲突变成为和谐的景象。同时，他也提出了 Pattern（模式）的概念，可引导设计师逐步创造出 Form。1971 年，该书再版上市，此时正是结构化设计方法的萌芽阶段，该书对当时 Ed Yourdon 和Larry Constantine 的结构化观念的诞生具有相当大的影响力。

4.13 面向对象系统设计

4.13.7 模式

- **1972~1985** 年，**Alexander** 任教于 **UC Berkeley**，他和其同事共同研究模式，出版了四本书：
  - *The Timeless Way of Building*：完整地介绍了模式及模式语言的概念。
  - *A Pattern Language*：列举了 **253** 个建筑方面的模式。
  - *The Oregon Experiment*：叙述了在 **Oregon** 大学的实验过程。
  - *The Production of Houses*：叙述了在墨西哥的实验，也详述了这实验并未成功的原因。
- **Alexander** 发明的模式语言，用来描述一系列建筑方面的问题、这些问题的约束条件以及解决这些问题的方法，不仅为建筑设计提供了一种通用的语言，可用于描述以往的设计经验，以利于沟通，而且使得以往的设计经验可以被重复使用。

4.13 面向对象系统设计

4.13.7 模式

- 受到这一思想的启发，人们从 **1980** 年代末期开始尝试将模式这一概念用到软件工程中去。到 **1990** 年代之后，设计模式便渐渐成为软件界（尤其是面向对象领域）的一个热门话题。
- **1991** 年，**Gamma** 完成了他的博士论文－《*Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools*》。
- **1995**年，**Gamma**、**Helm**、**Johnson** 和 **Vlissides**（俗称四人帮）撰写了一部名著：《*Design Patterns: Elements of Reusable Object-Oriented Software*》，奠定了这个领域的基础。

4.13 面向对象系统设计

4.13.7 模式

- 模式在设计领域的成功，使之推广到分析领域 – 分析模式。实际上，二者只有细致程度的区别。
- 即使是在设计领域，模式也被细分为体系结构模式和设计模式:
  - 前者主要体现了体系结构的设计经验;
  - 后者则主要体现了类之间关联结构的设计经验。

4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

- 典型的体系结构模式（*Architectural Patterns*）
  - Distributed
  - Event-driven
  - Frame-based
  - Batch
  - Pipes and filters
  - Repository-centric
  - Blackboard
  - Interpreter
  - Rule-based
  - Layered
  - MVC
  - IR-centric（信息查询为中心）
  - Subsumption（包容式）

4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

- 体系结构模式的分类:

类别	特征	典型模式
Data flow	dominated by motion of data through the system	◆ batch sequential ◆ data flow network ◆ pipes and filters
Call and return	dominated by order of computation	◆ main program/subroutines ◆ abstract data types ◆ object ◆ call based ◆ client/server ◆ layered

4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

- 体系结构模式的分类:

类别	特征	典型模式
Independent components	dominated by communication patterns	• event systems • communicating processes
Data-centered	dominated by a complex central data store, manipulated by independent computations	• repository • blackboard
Virtual machine	characterized by translation of one instruction set into another	• interpreter

4.13 面向对象系统设计

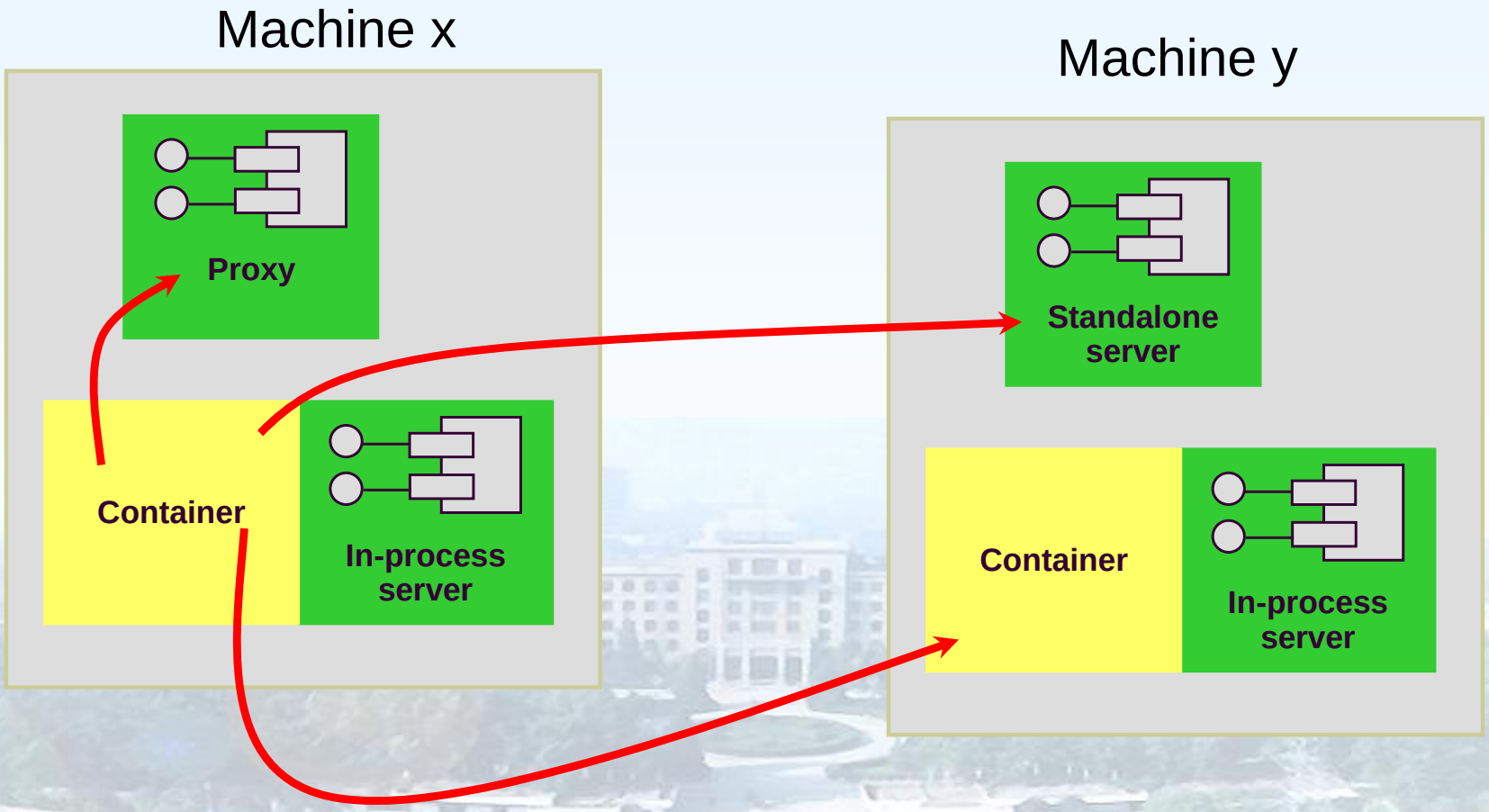
4.13.7 模式– 体系结构模式

- 一些商品化平台 / 构架已经提供了基本的体系结构支撑:
  - MTS / MSQS (Microsoft Transaction Server / Microsoft Message Queue)
  - CORBA
  - Enterprise Java Beans (EJB)
  - Domino
  - SAP R/3（**SAP** 公司的 **ERP** 产品）
  - Delphi
  - Forte（**Sun** 公司的 **Java** 开发平台）
  - Visual Basic

4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

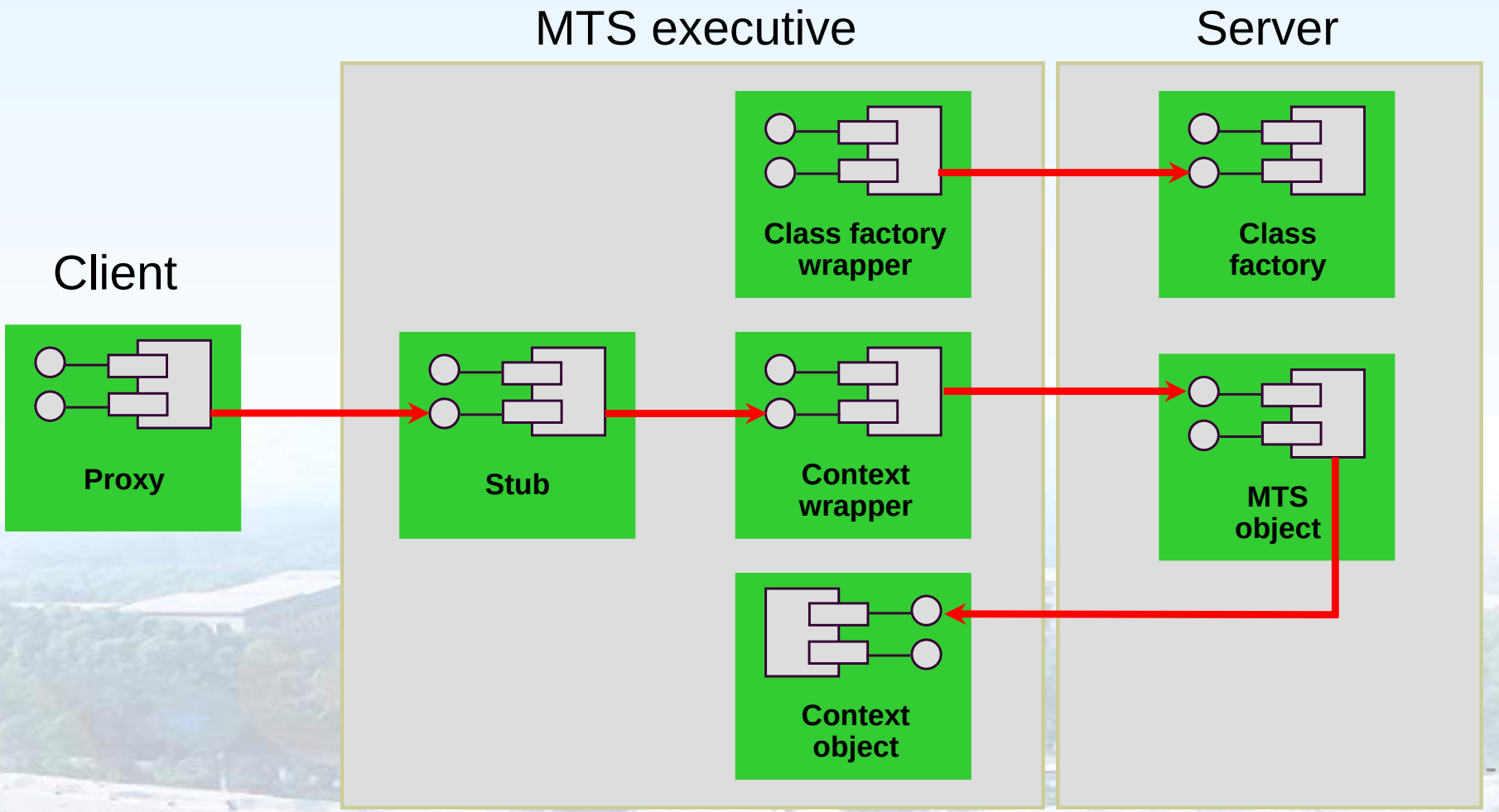
- DCOM的体系结构:



4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

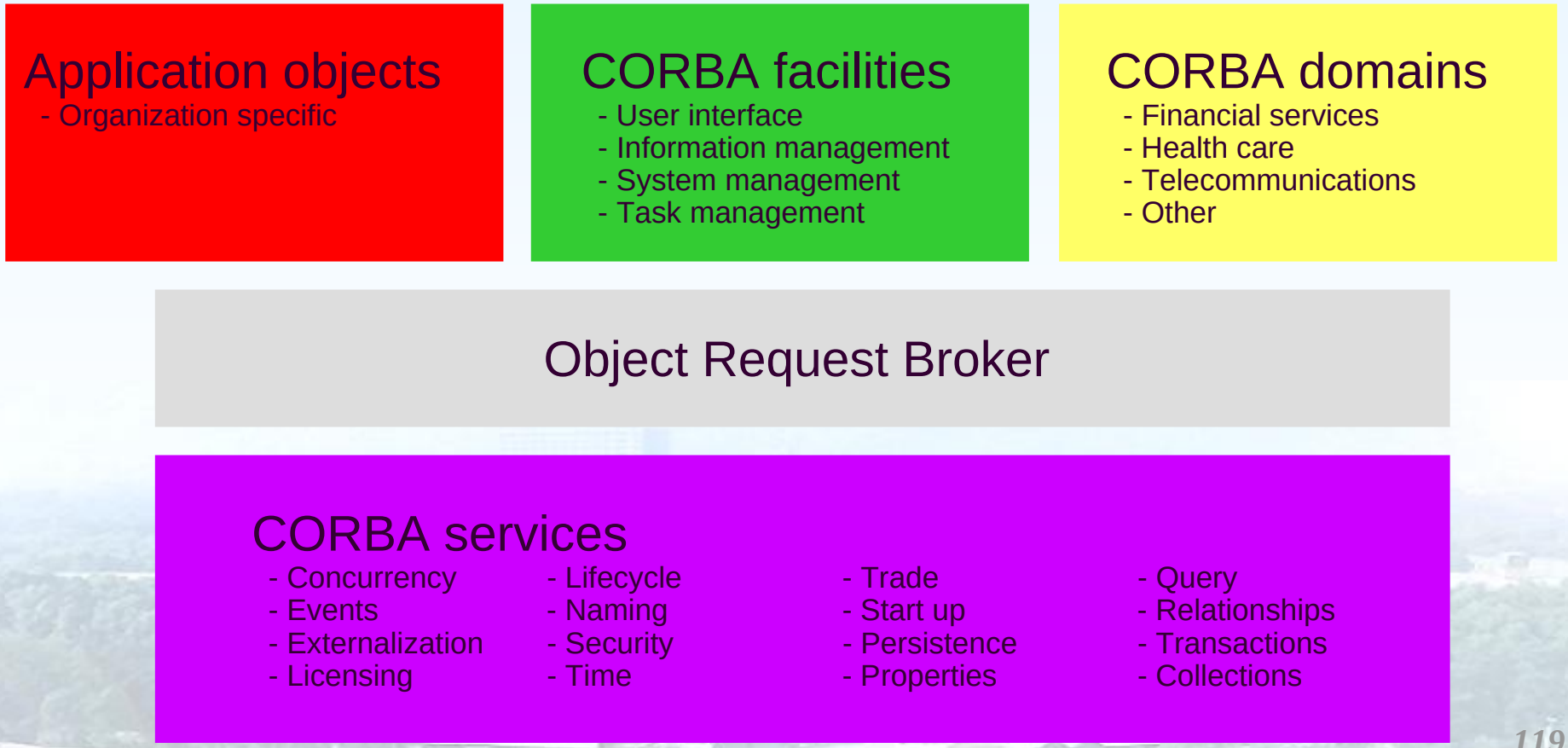
- MTS 的体系结构:



4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

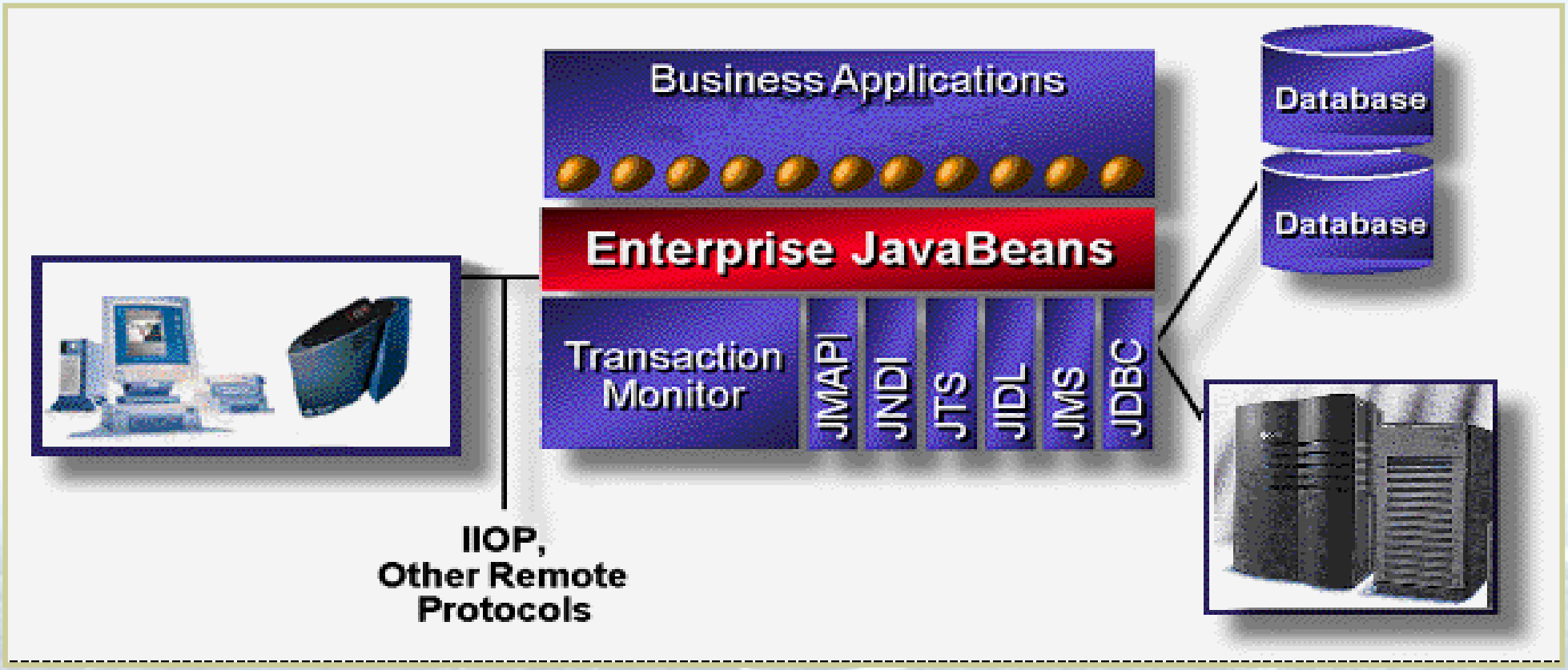
- **CORBA** 的体系结构:



4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

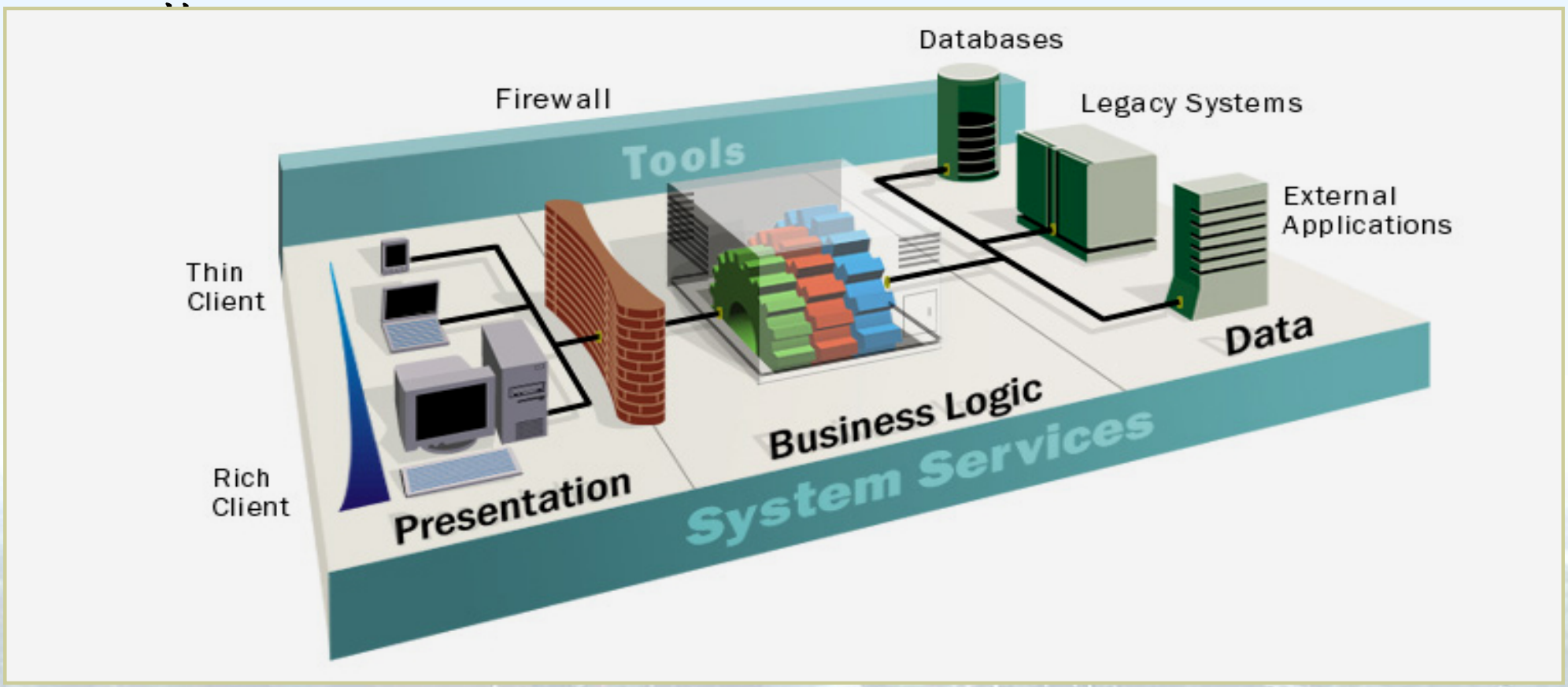
- EJB 的体系结构:



4.13 面向对象系统设计

4.13.7 模式– 体系结构模式

- Microsoft DNA (Distributed interNet Application) 的体系结



4.13 面向对象系统设计

4.13.7 模式– 设计模式

- 设计模式是一个抽象的方案，它可以解决一类问题。设计模式往往首先描述一类相似的问题，然后提出一个抽象而通用的解决方案，使用这一解决方案可以解决所描述的那些问题。
- 设计模式一般由三个主要部分组成:
  - 类的结构及对象交互的抽象描述;
  - 所涉及的问题（这决定了该设计模式的应用范围）;
  - 应用的后果（这决定了在某些约束下是否应使用该设计模式）。
- 设计模式比构架的规模小得多，易于组合与理解。
- 设计模式的应用难点在于如何将实际问题加以分解，使之与既定的设计模式匹配。

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 系统行为的设计 – 协同图是一种重要的工具:
  - What methods in what classes? How should objects interact?
    - These are critical questions in the design of behavior.
    - Poor answers lead to abysmal（糟透了的）, fragile（脆弱的） systems with low reuse and high maintenance.
  - Collaboration diagrams illustrate the design of behavior.
    - What is a good design?
    - What are the principles of a good design?

4.13 面向对象系统设计

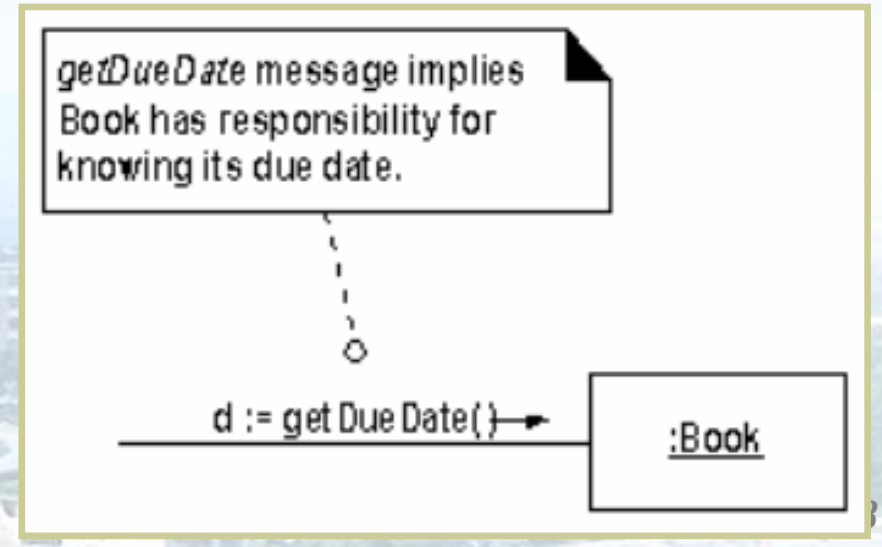
4.13.8 对交互图进行精制－基于模式的设计

- 先引入一个概念－责任（*Responsibility*）：
  - Responsibility is a contract or an obligation（义务） to meet certain agreed conditions.
  - Two kinds of responsibilities for an object - “a thing”: knowing and doing.
  - Responsibility of knowing:
    - privately owned information
    - about other related objects
    - about how to do something special (algorithm)
  - Responsibility of doing:
    - doing something by itself
    - initiating collaboration with other objects
    - controlling/coordinating activities in other objects

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

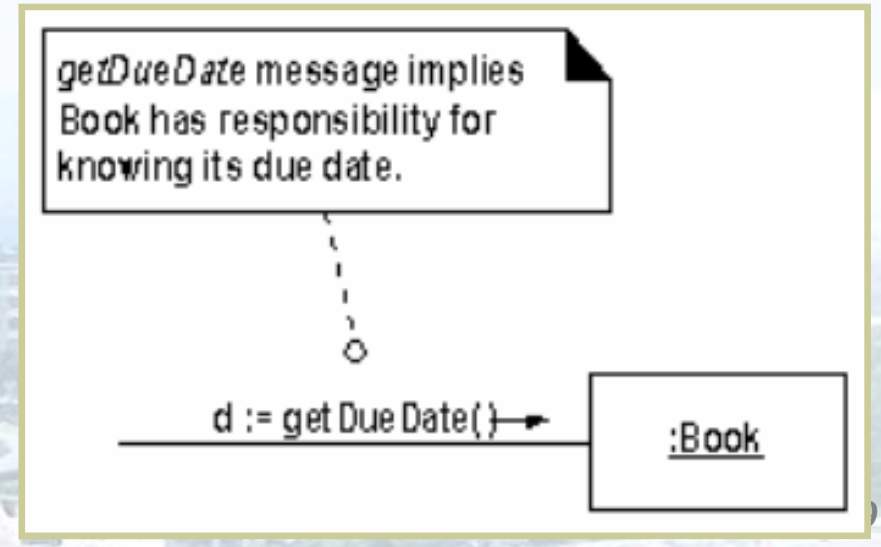
- 责任的指派:
  - Design of behavior implies assigning responsibilities to software classes.
  - Responsibilities.
    - Knowing
    - Doing
  - A message in a collaboration diagram suggests a related responsibility.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 责任的指派:
  - A responsibility is:
    - related to the methods and data of classes,
    - fulfiled with one or more methods, and
    - possibly spread across classes.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 责任的指派:
  - Appropriate assignment of responsibilities to classes is the key to successful design.
  - There are fundamental principles in assigning responsibilities that experienced designers apply.
  - These principles are summarized in the GRASP patterns.

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- **GRASP 模式:**
  - Acronym for General Responsibility Assignment Software Patterns.
  - Has nine core principles that object-oriented designers apply when assigning responsibilities to classes and designing message interactions.
  - Can be applied during the creation of collaboration diagrams.

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- **GRASP 模式:**
  - Memorization and application of these patterns are critical for successful object-oriented designs.
    1. Expert
    2. Creator
    3. Controller
    4. Low Coupling（低耦合）
    5. High Cohesion（高内聚）
    6. Polymorphism
    7. Pure Fabrication（纯装配）
    8. Indirection
    9. Don’t Talk to Strangers

4.13 面向对象系统设计

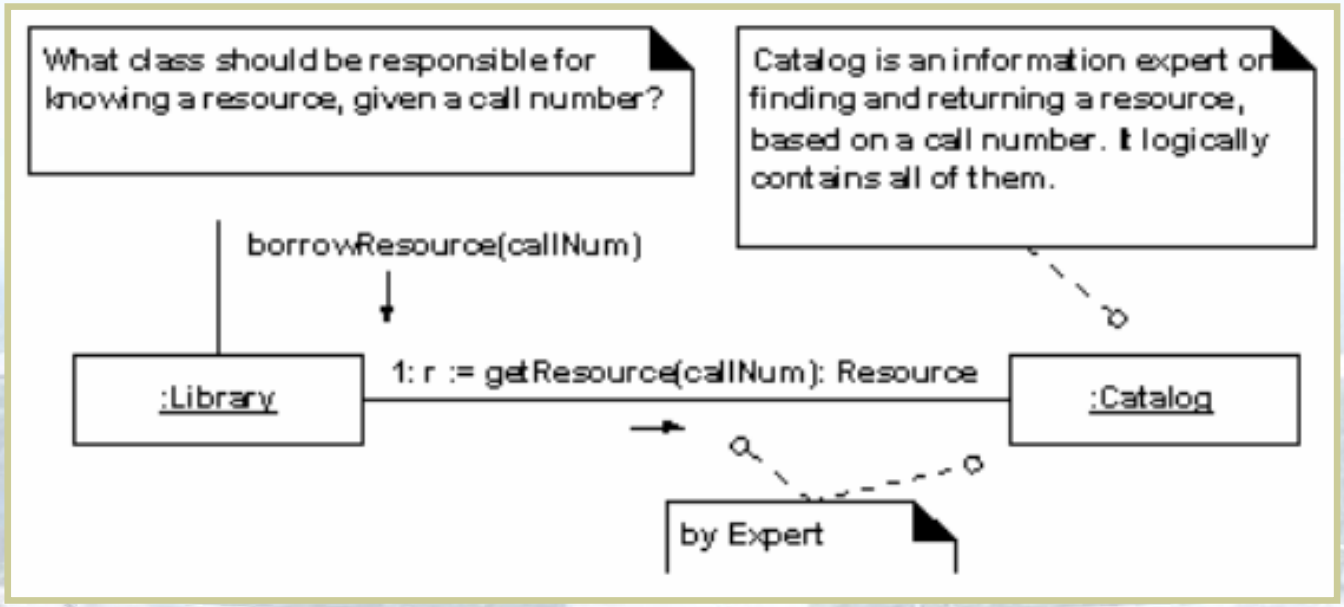
4.13.8 对交互图进行精制－基于模式的设计

- **GRASP 模式:**
  - Memorization and application of these patterns are critical for successful object-oriented designs.
    1. Expert
    2. Creator
    3. Controller
    4. Low Coupling（低耦合）
    5. High Cohesion（高内聚）
    6. Polymorphism
    7. Pure Fabrication（纯装配）
    8. Indirection
    9. Don’t Talk to Strangers

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- **Expert 模式:**
  - The most general purpose responsibility assignment principle.
    - Assign a responsibility to the information expert – the class with the information necessary to fulfill the responsibility.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- **Creator 模式:**
  - Who should create an instance of a particular class?
    - Consider assigning Class B the responsibility to create an instance of class A if one of the following is true.
      - B contains A.
      - B aggregate A.
      - B records A.
      - B closely uses A.
      - B has the initialization data needed for the creation of A (i.e., expert in creating A).
    - B is the "creator" of A instances.
    - The pattern supports low coupling.

4.13 面向对象系统设计

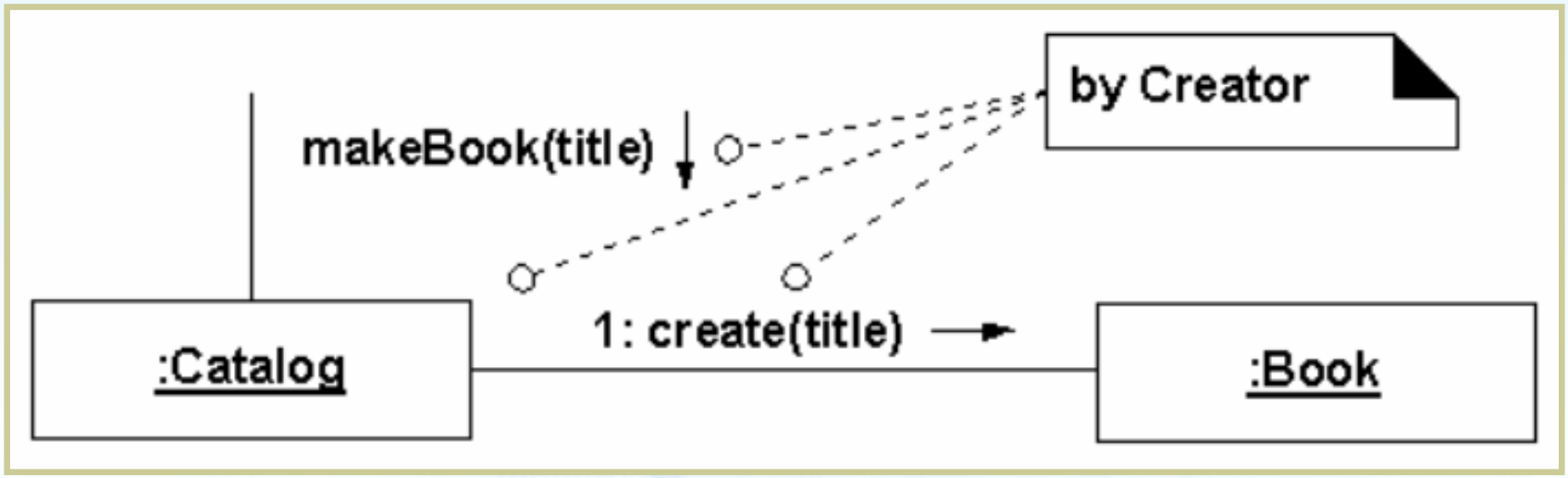
4.13.8 对交互图进行精制—基于模式的设计

Association Category	Examples
A is part of B	*Sentence – Paragraph*
A is a line item of B	*SalesLineItem – Sale*
A is contained with in B	*Book – Library*
A is a member of B	*Librarian – Library*
A is an organizational subunit of B	*Department – Company*
A is a policy related to B	*LoanPolicy – Book*
A is recorded in B	*Book – Catalog, Loan – Library*
A uses or manages B	*Patron – Library*
A is related to transaction of B	*Patron – ResouceLoan*
A communicates with B	*Patron – Librarian*
A is a transaction related to another transaction B	*Loan – Return*
A is next to B	*ChessSquare – ChessSquare*
A is related to B via a transaction	*Patron – Librarian, Husband – Wife*

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- **Creator 模式:**



4.13 面向对象系统设计

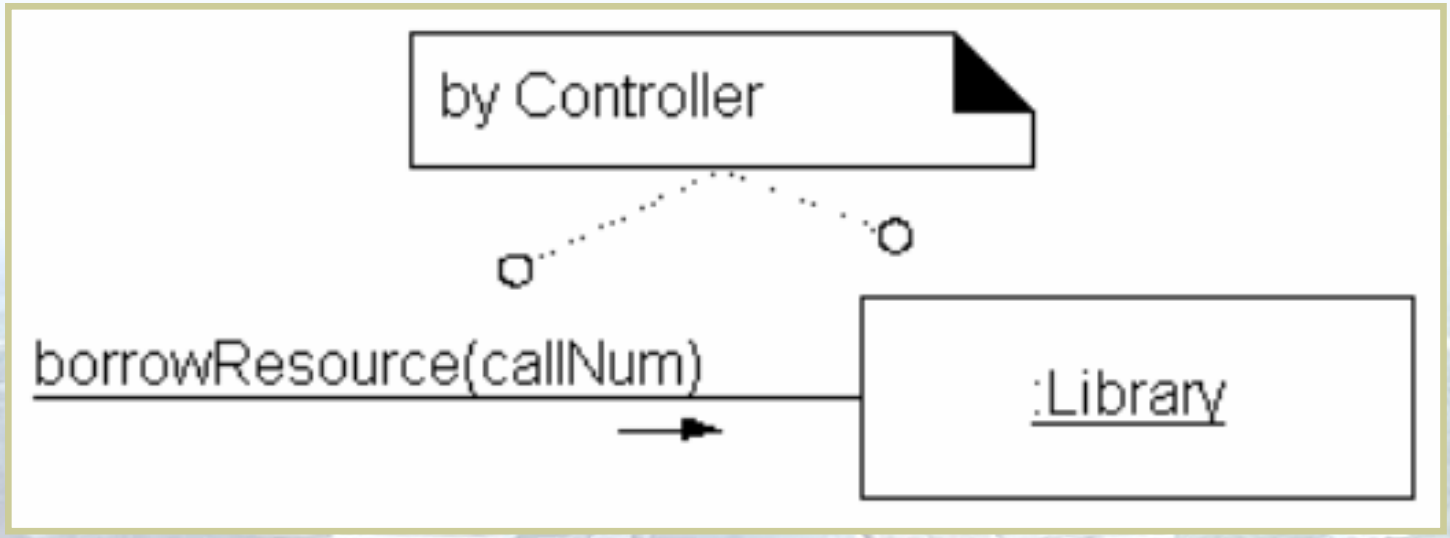
4.13.8 对交互图进行精制－基于模式的设计

- **Controller 模式:**
  - A controller is a class that handles a system event message.
  - What class should handle a system event message?
  - Assign the responsibility for handling a system operation message to one of these options:
    - The business or overall organization (a façade（正面的） controller).
    - The overall “system” (a façade controller).
    - An animate（活跃的） thing in the domain that would perform the work (a role controller).
    - An artificial class representing the use case (a use-case controller).

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

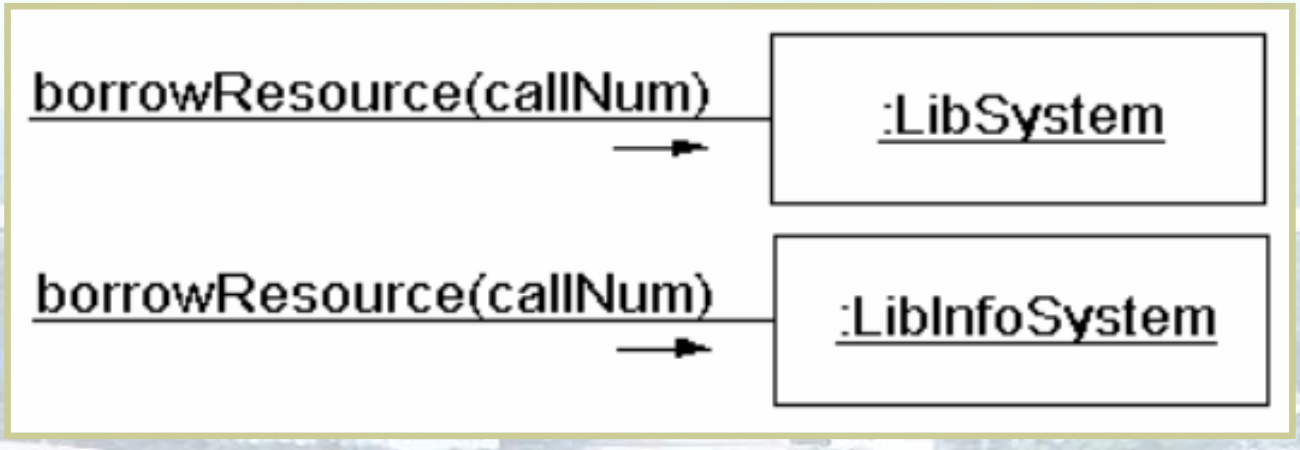
- **Controller 模式– Façades（*门面*）:**
  - Façades are “covers”.
  - Intent – A class that (in some way) represents an overall cover.
  - Many possibilities.
    - Example: The entire organization.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

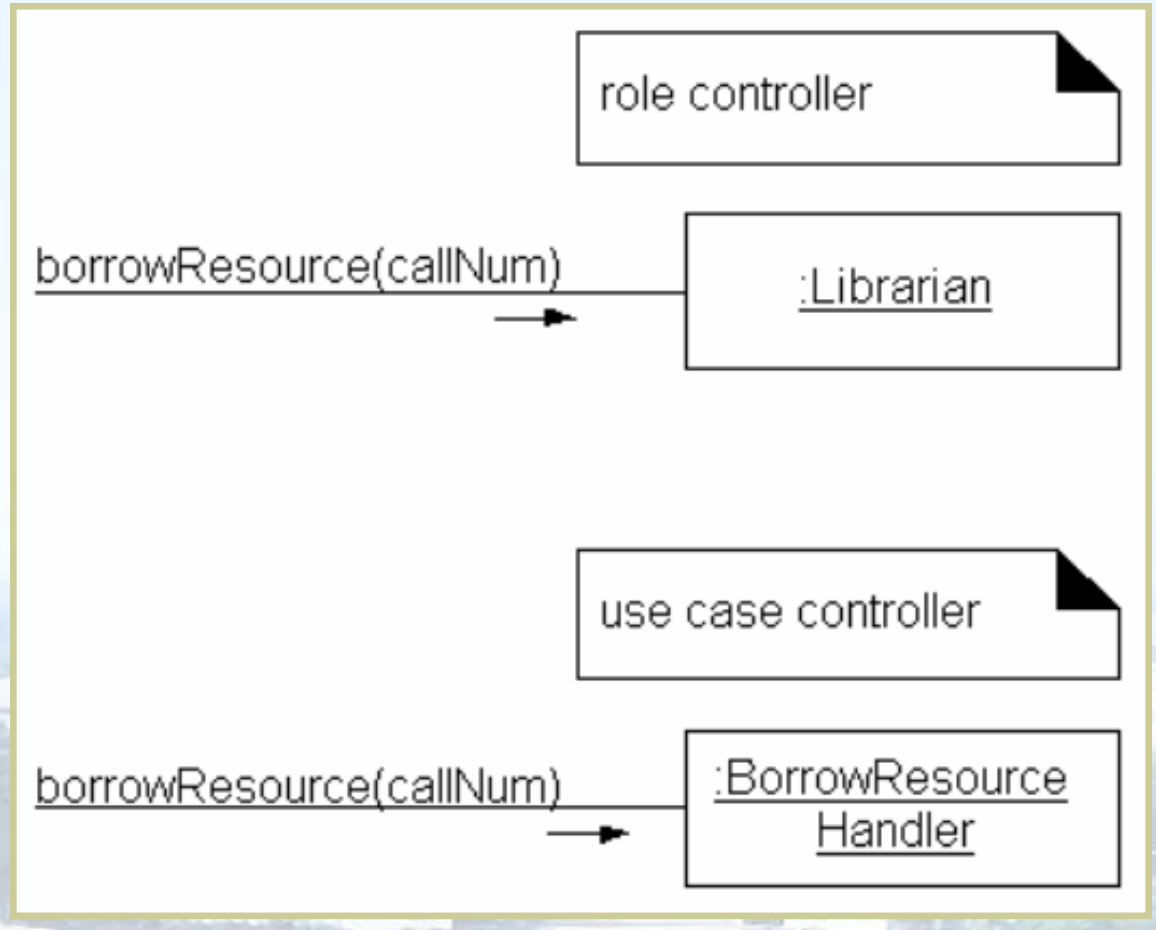
- **Controller 模式– Façades:**
  - Other façades? A class representing the “system”.
  - Examples:
    - The software information system.
    - The device that includes a computer and software such as an ATM.
    - Others.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

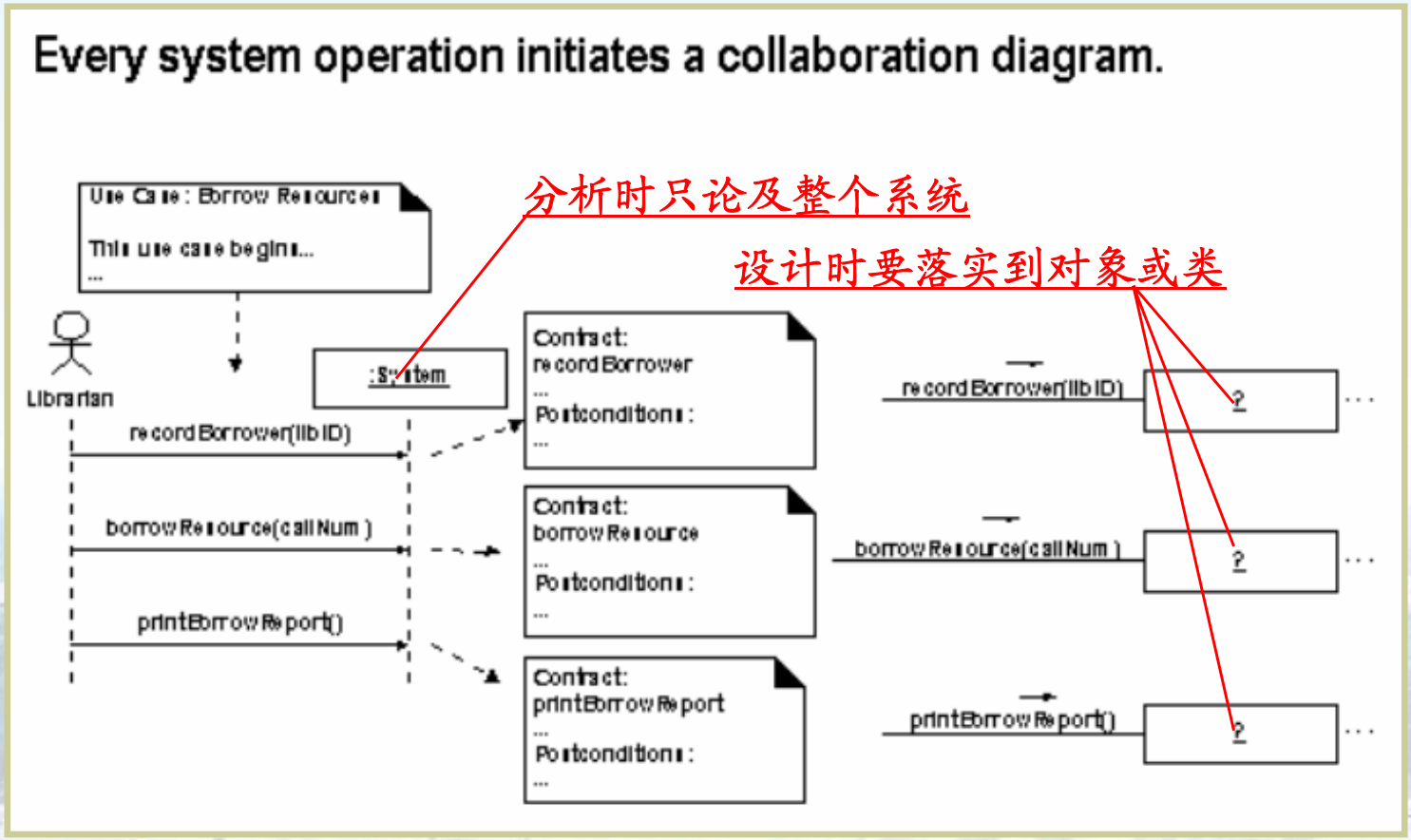
- **Controller 模式– 其他两种:**



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 将分析阶段得出的用例、操作合约与协同图的设计加以关联:



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 例（借书业务的用例 – **Borrow Books**）：

Actor Action	*System Response*
[1] Patron approaches circulation desk with books and library membership card, making a request to borrow the books from the Library.	
[2] Librarian checks the current status of the patron's membership – asking for the status and number of books on loan.	[3] System responds with the patron's membership status (active/inactive), and number of books currently on loan.
[4] If status is active and the total number of books on loan will not exceed a limit, Librarian approves request.	
[5] For each book, Librarian records the Call No. and the Copy No., along with the patron's membership ID No. in the system for the book to be on loan to the patron.	[6] System accepts the information and updates the circulation records of the loan for each book. System issues the due date for each book on loan.
[7] Librarian gives the books to the patron to take them out on loan.	

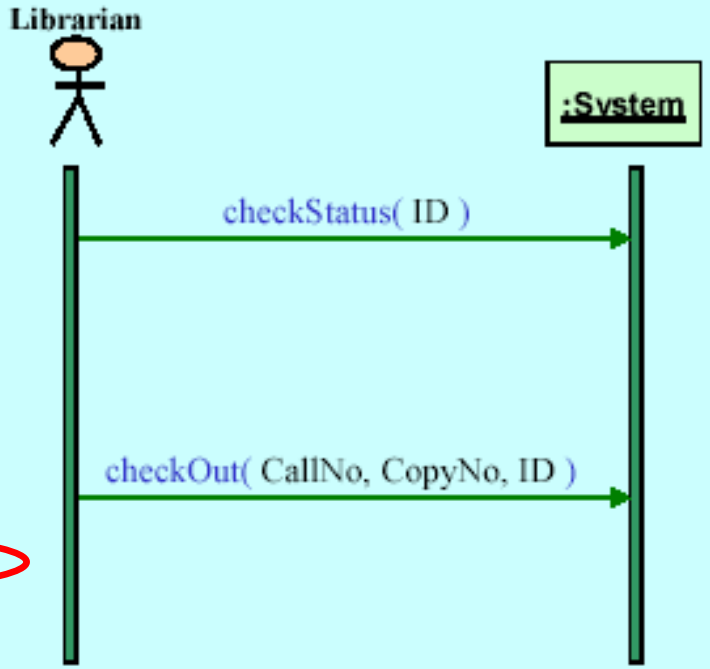
4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 例（**Borrow Books** 对应的系统序列图）：

Use Case: **Borrow Books**

(1) Patron requests to borrow books.
(2) Librarian checks the current status of the patron's membership – using the ID from the patron's membership card.
System event: checkStatus(ID).
(3) **System responds with the patron's membership status (active/inactive), and the number of books currently on loan.**
System displays information to Librarian.
(4) If status is active and the total number of books on loan will not exceed a limit, Librarian approves request.
(5) For each book, Librarian enters the Call No. and the Copy No., along with the patron's membership ID No. for loan.
System event: checkOut(Call#,Copy#,ID).
(6) **System updates the circulation records, and issues the due date for each book.**
(7) Librarian gives the books to the patron to take out on loan.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

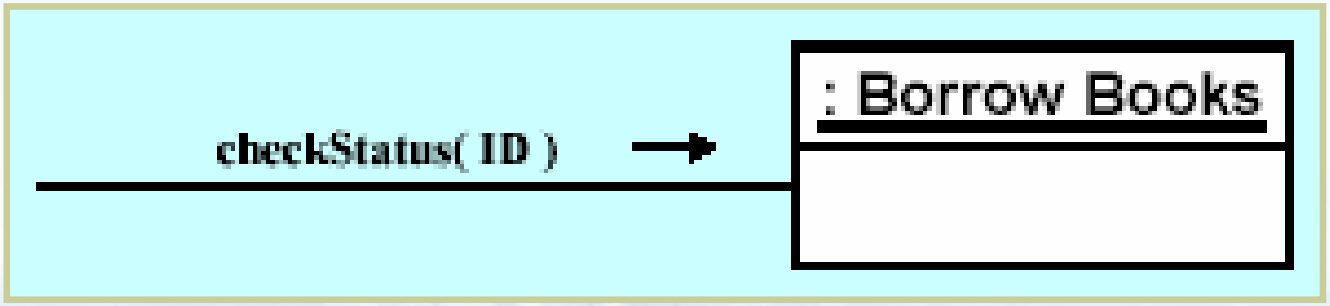
- 例（系统操作 **checkStatus(ID)** 的操作合约）：

CONTRACT	
Name:	*checkStatus(ID)*
Responsibilities:	*to verify Membership ID, and check # of books on loan.*
Type:	*System*
Use Cases:	*Borrow Books*
Exceptions:	*none*
Output:	*none* (Membership ID status will be on display.)
Pre-conditions:	*Access to Membership Records, Circulation Records.*
Post-conditions:	• *No change in system state*
	•
	•
	•
	•

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

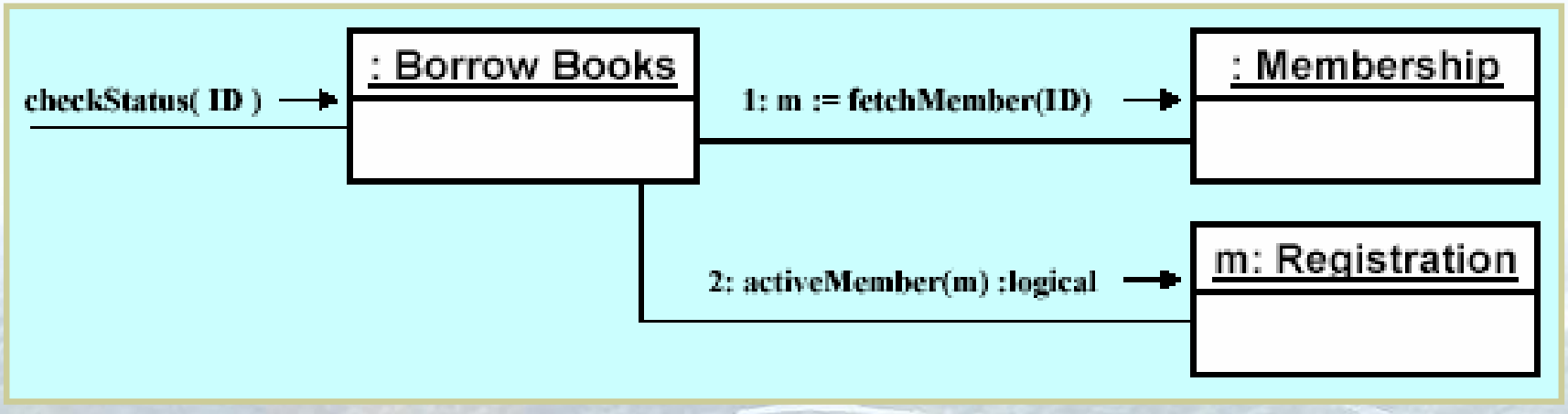
- 例（系统操作 **checkStatus(ID)** 的设计）：
  - The system event goes to the Business Process taking care of the current Use Case – that is, Borrow Books.
  - Application of Controller Pattern ...



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

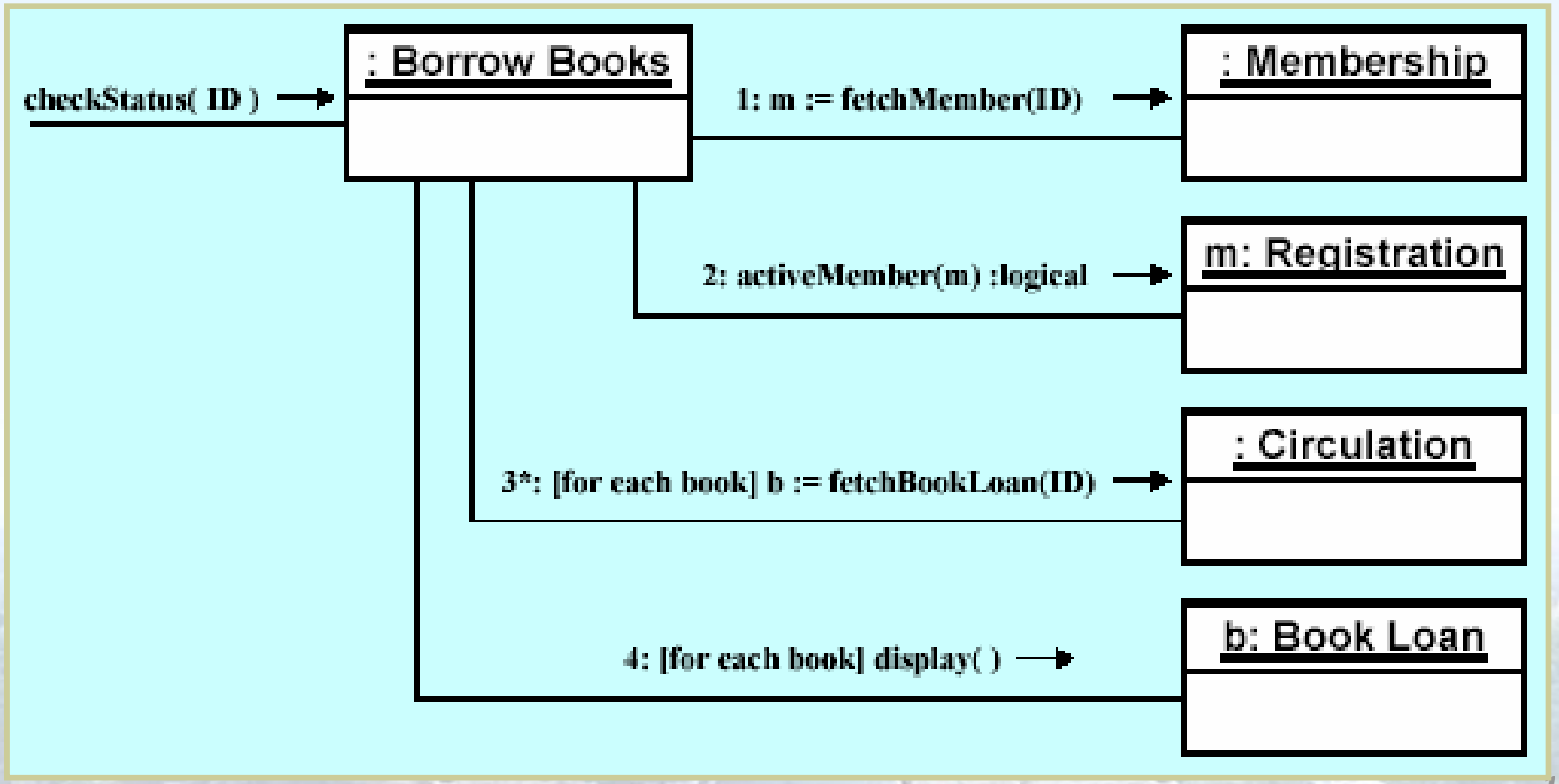
- 例（系统操作 **checkStatus(ID)** 的设计）：
  - To fetch member registration from ID, use the membership records.
  - To check for active status, ask the membership registration.
  - Application of Expert Pattern … in both cases.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

- 例（系统操作 **checkStatus(ID)** 的设计）：



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

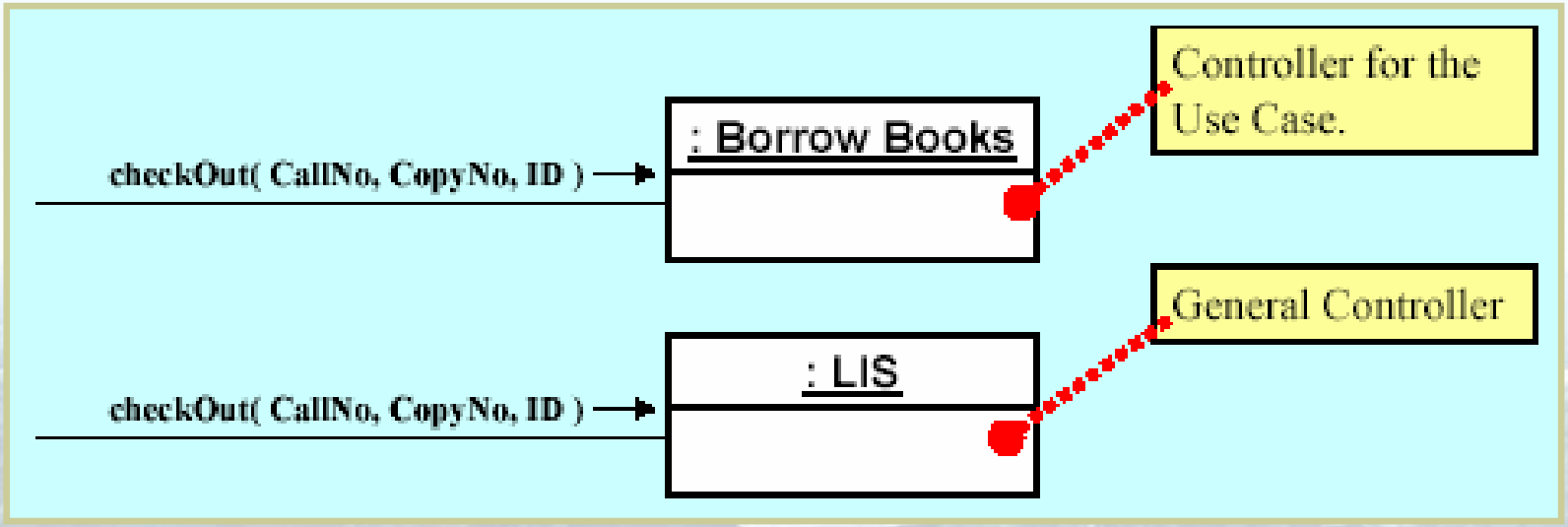
- 例（系统操作 **checkOut(CallNo, CopyNo, ID)** 的操作合约）：

CONTRACT	
Name:	*checkOut(CallNo, CopyNo, ID)*
Responsibilities:	*to record that the Member identified by the ID has taken the specified book (CallNo,CopyNo) on loan.*
Type:	*System*
Use Cases:	*Borrow Books*
Exceptions:	*Invalid Membership ID; No such book in the library.*
Output:	*none*
Pre-conditions:	*Access to Circulation Records.*
Post-conditions:	• *A new loan record is generated.*
	• *The loan record identifies the book on loan.*
	• *The loan record identifies the Member.*
	• *The loan record marks the appropriate due date.*
	• *The loan record is kept with the circulation records.*
	•

4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

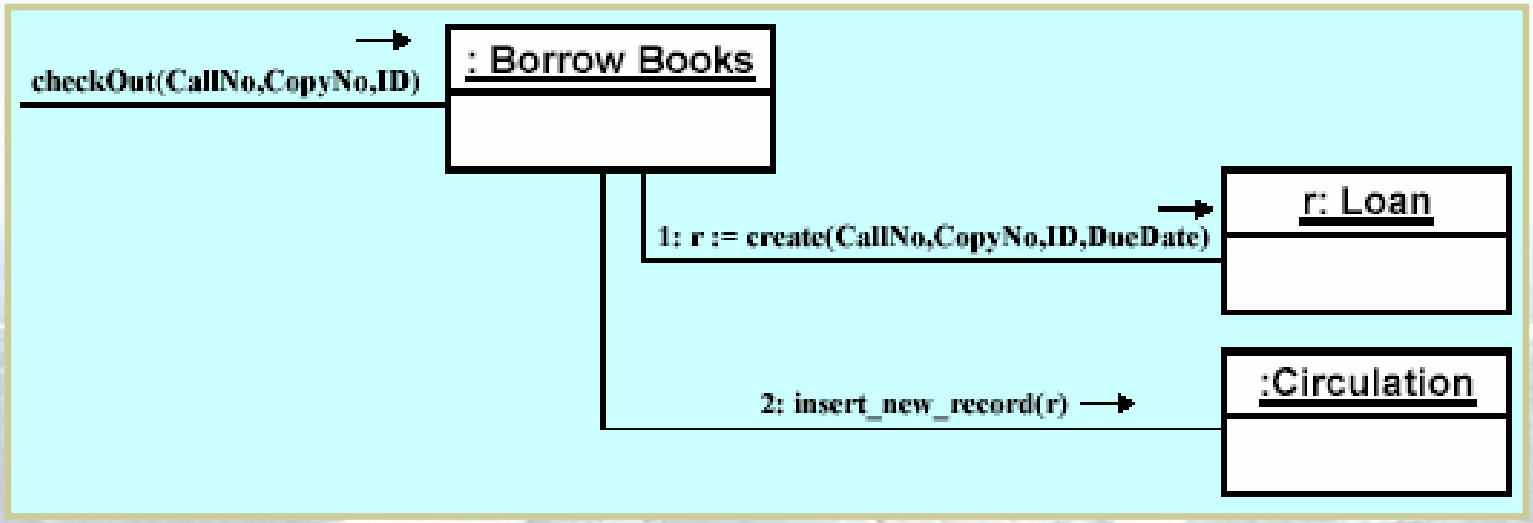
- 例（系统操作 **checkOut(CallNo, CopyNo, ID)** 的设计）：
  - Who should respond to checkOut(CallNo,CopyNo,ID)?
    - The system event goes to the Business Process taking care of the current Use Case – that is, Borrow Books.
    - Application of Controller Pattern ...



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

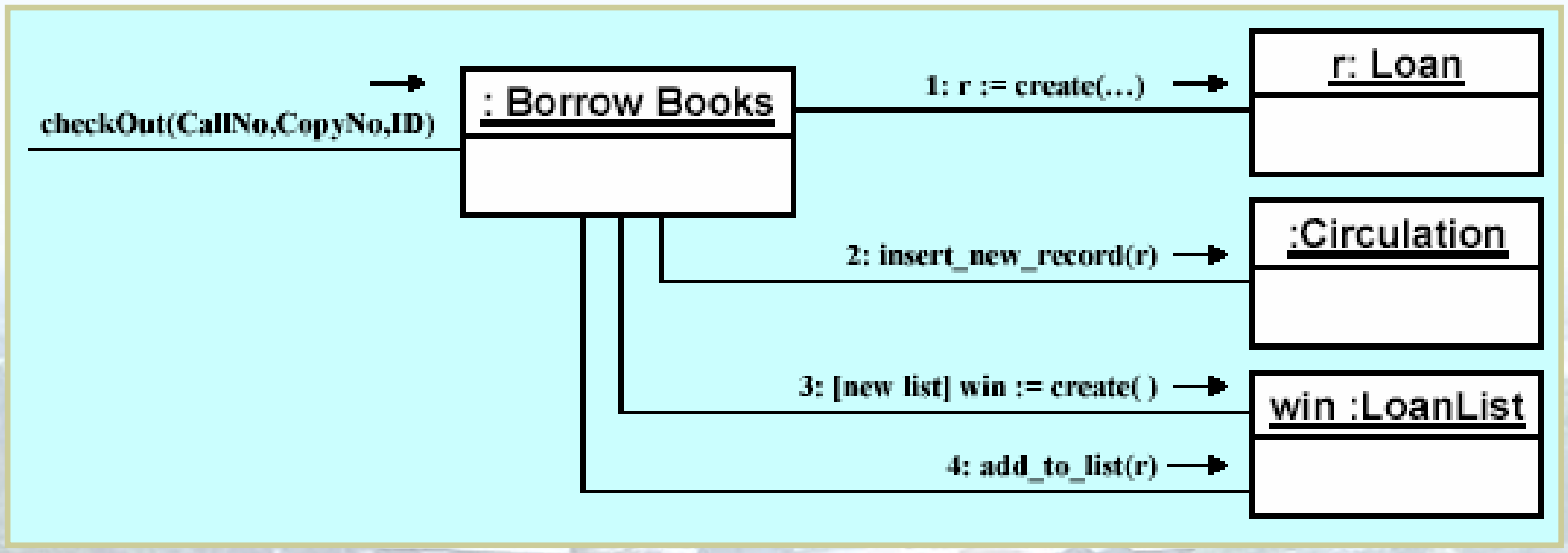
- 例（系统操作 **checkOut(CallNo, CopyNo, ID)** 的设计）：
  - To create a new loan record for the transaction ...
    - Create new loan record with appropriate attribute values for the current loan transaction.
    - Send the loan record to Circulation for keep sake（出于保存借书记录的缘故）.



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

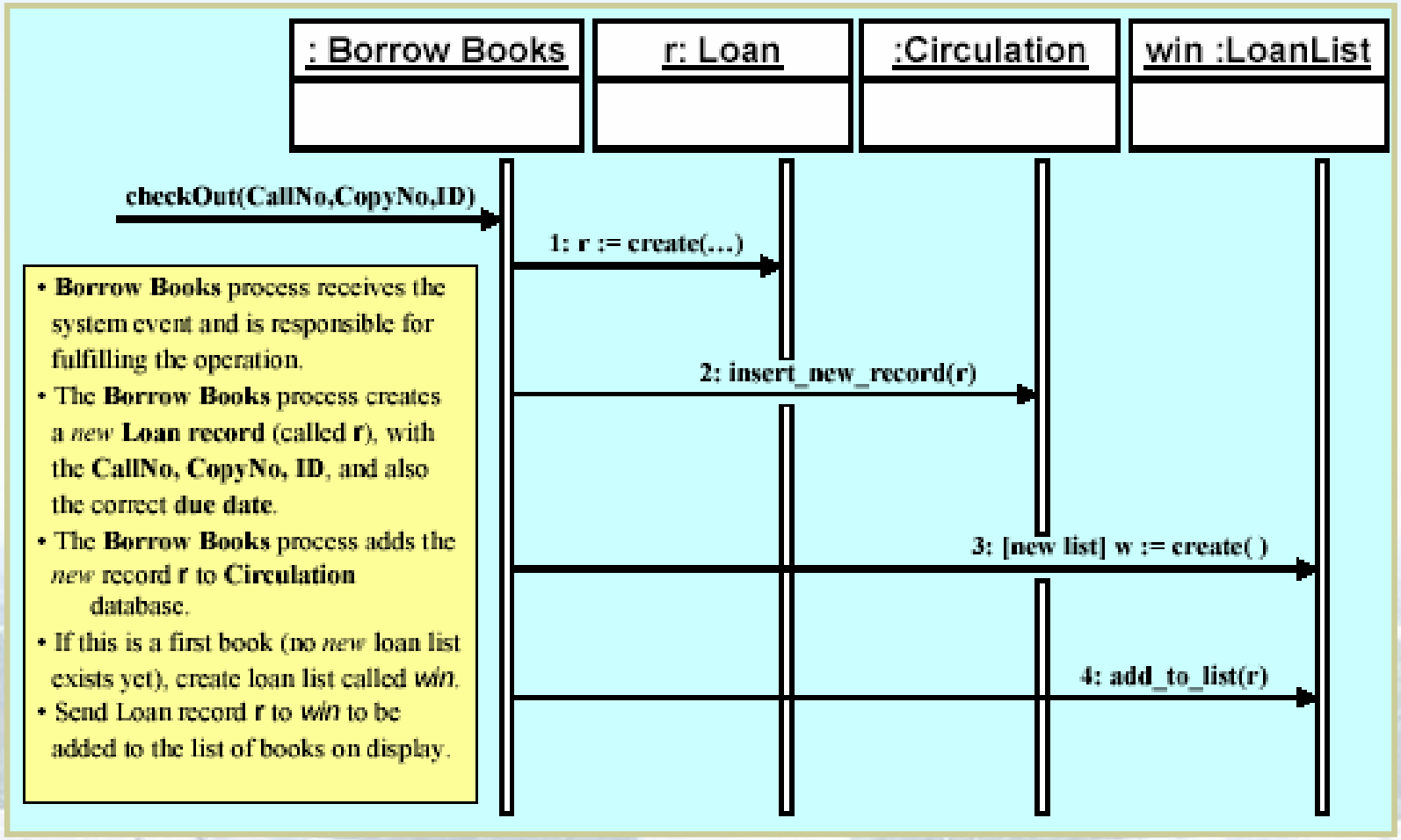
- 例（系统操作 **checkOut(CallNo, CopyNo, ID)** 的设计）：
  - To list the loan information in (the) pop-up window.
    - If the pop-up window is not already existing, create the window for Books Loaned Out.
    - Send the loan record to display (in a window).



4.13 面向对象系统设计

4.13.8 对交互图进行精制－基于模式的设计

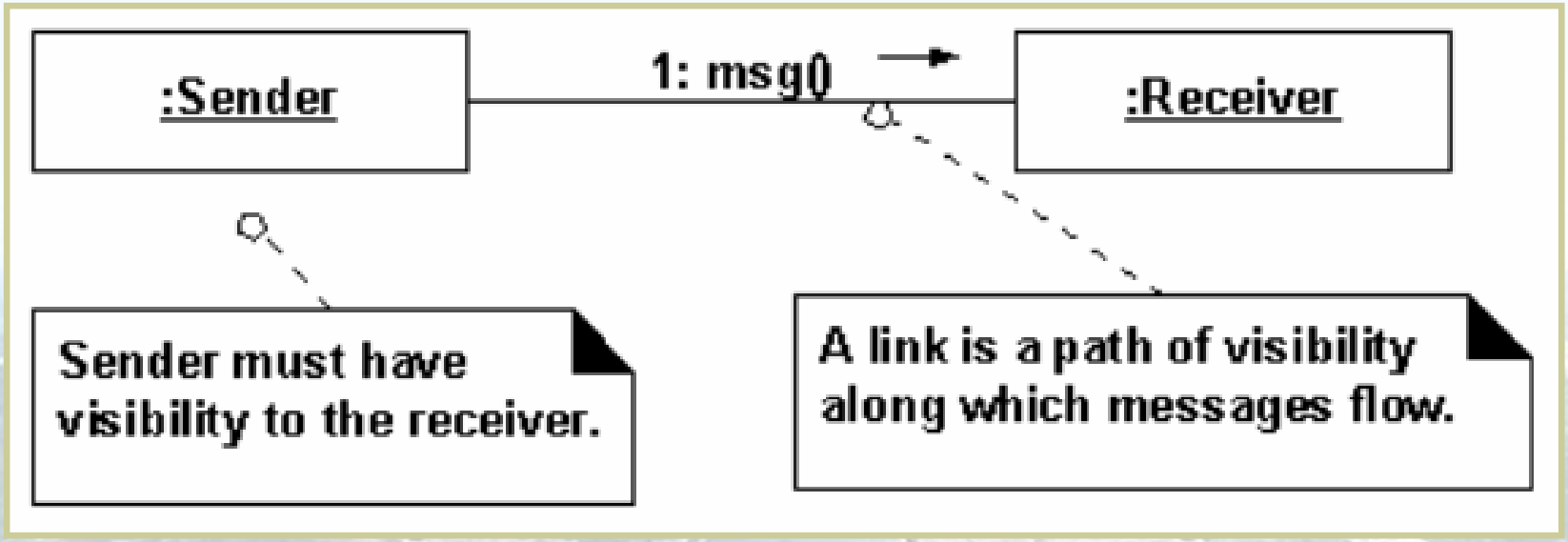
- 例（系统操作 **checkOut(CallNo, CopyNo, ID)** 的设计）：



4.13 面向对象系统设计

4.13.9 对象间关联方式的考虑

- 对象视域（*visibility*）：
  - Visibility is the ability of an object to “see” or have a reference to another object.
  - In order for a sender object to send a message to a receiver object, it must have visibility to the receiver.



4.13 面向对象系统设计

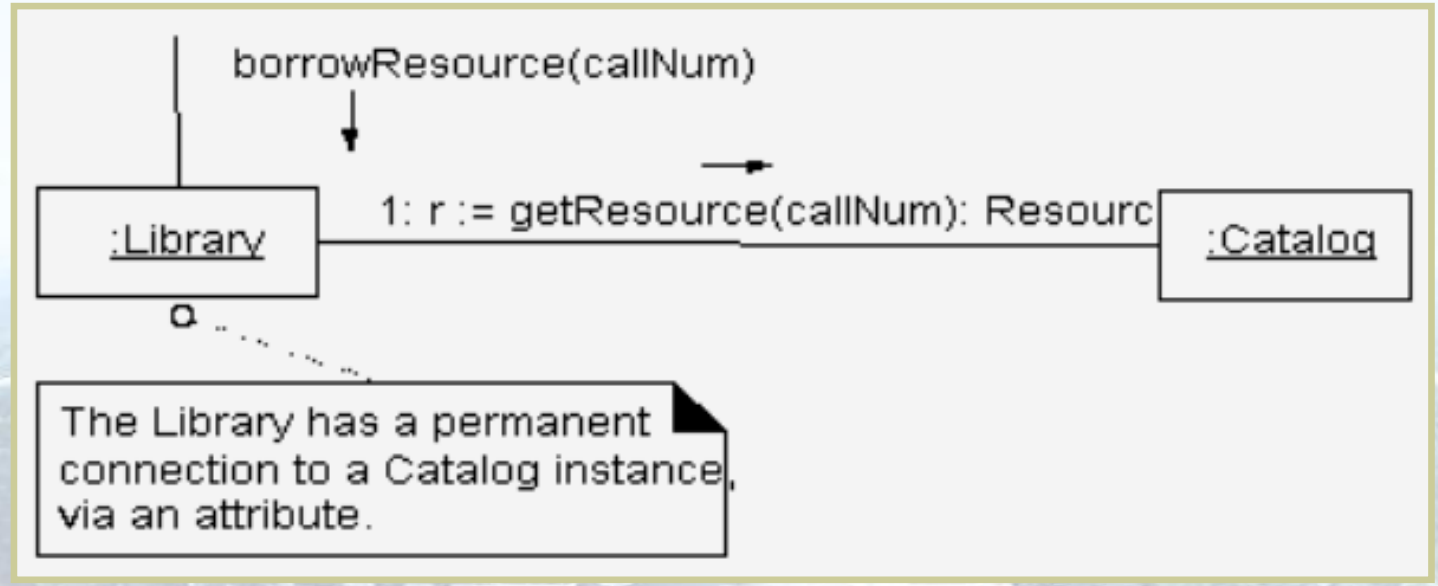
4.13.9 对象间关联方式的考虑

- Visibility must be estabilished; it doesn't happen automatically.
- There are four common types of visibility.
  - Attribute visibility（属性视域）.
  - Parameter visibility（参数视域）.
  - Local visibility（局部视域）.
  - Global visibility（全局视域）.

4.13 面向对象系统设计

4.13.9 对象间关联方式的考虑

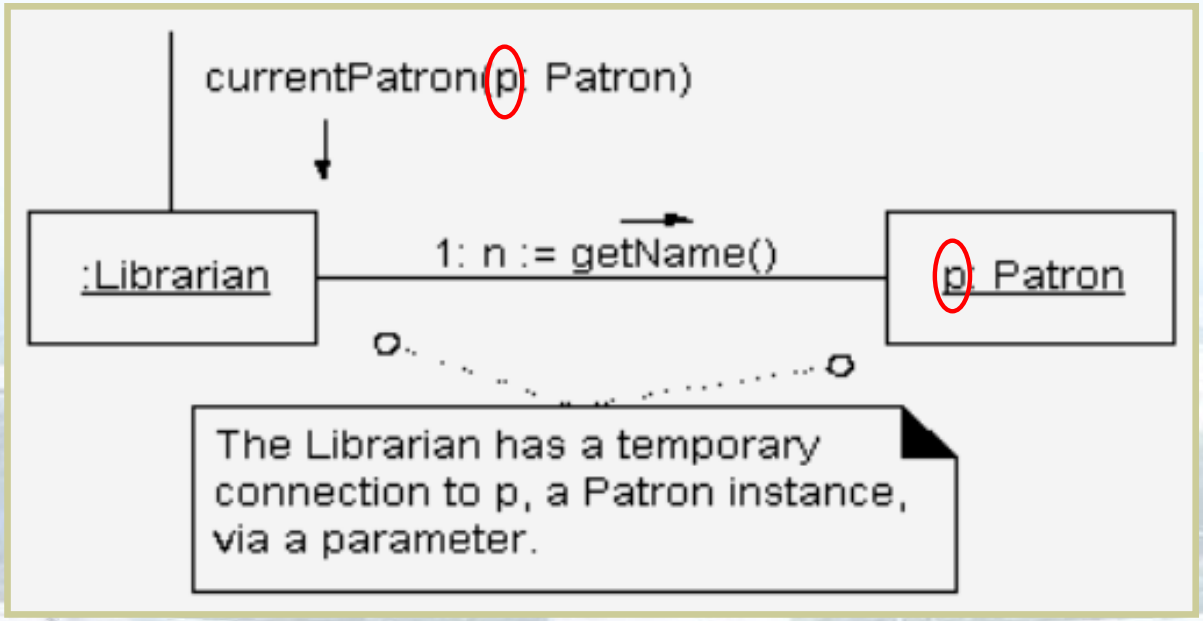
- 属性视域:
  - When a client object defines an attribute that references an instance of a server object.
  - Required if the client needs a permanent memory（持久记忆）of the server object.



4.13 面向对象系统设计

4.13.9 对象间关联方式的考虑

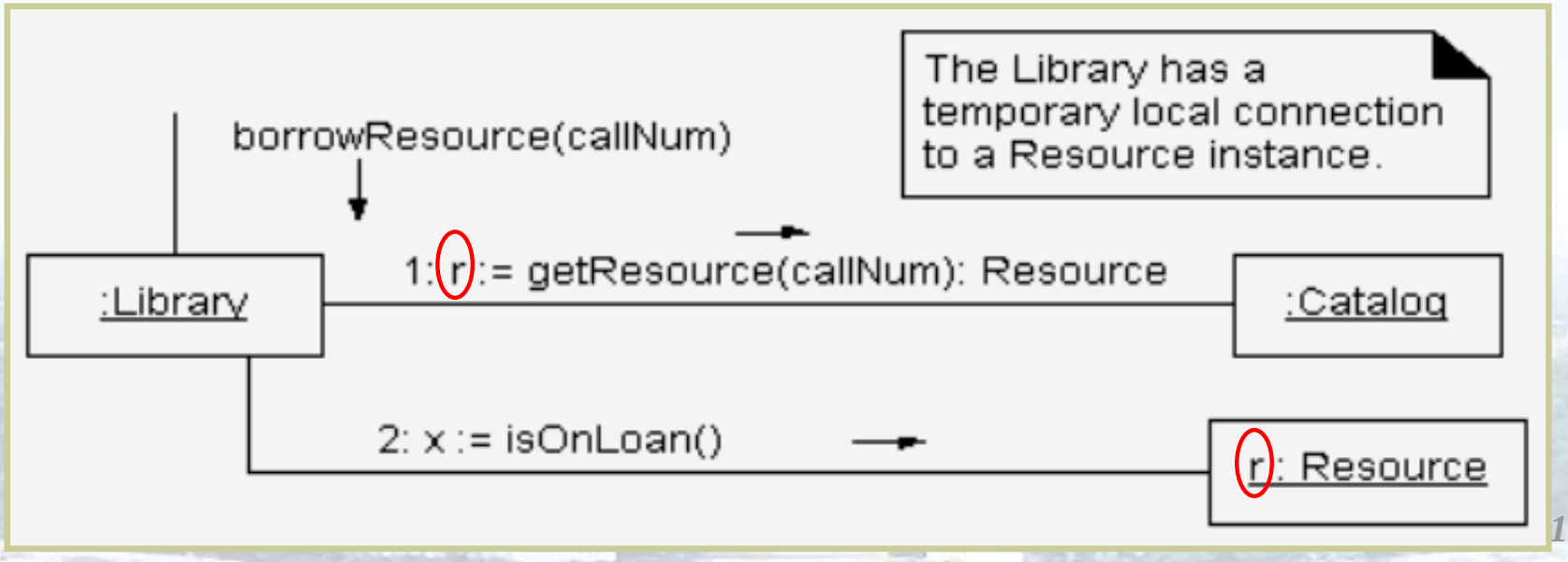
- 参数视域:
  - When a client object receives a parameter that refers to an instance of a server object.
  - Required if the client needs a temporary connection to the server object.



4.13 面向对象系统设计

4.13.9 对象间关联方式的考虑

- 局部视域:
  - In a method of the client object, when a local variable is assigned a new instance or an existing object.
  - Required if the client needs a temporary connection to the server object.



4.13 面向对象系统设计

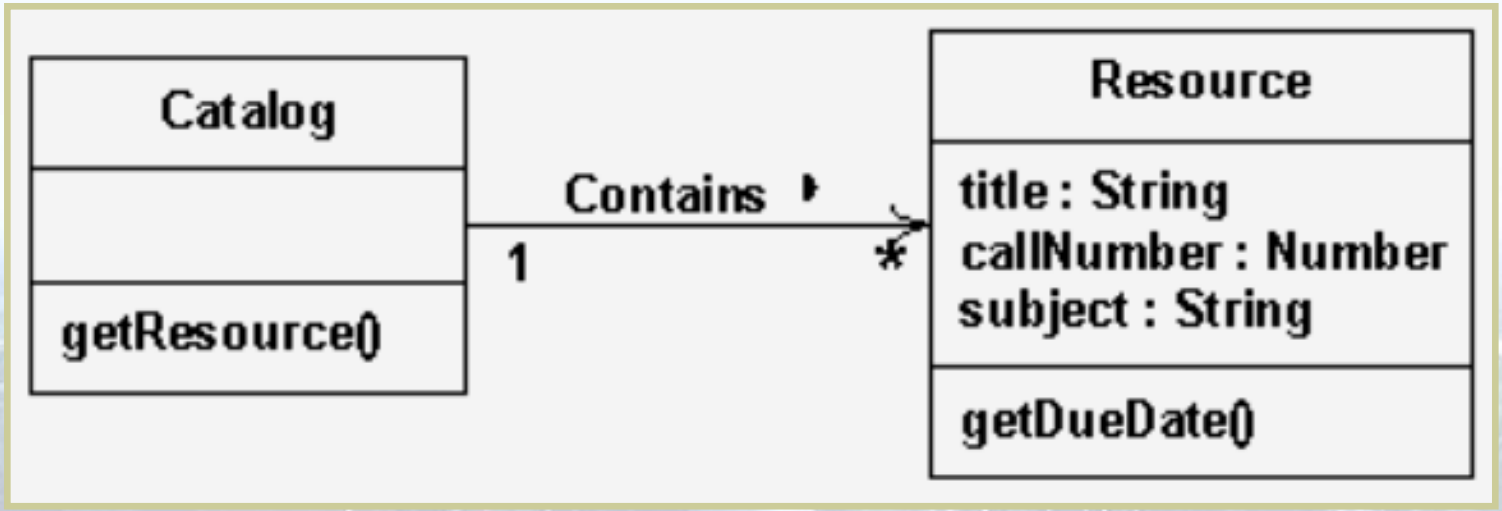
4.13.9 对象间关联方式的考虑

- 全局视域:
  - Permanent visibility to an object via (for example):
    - Global variable.
    - Singleton pattern（独身模式）.

4.13 面向对象系统设计

4.13.10 产生设计类图

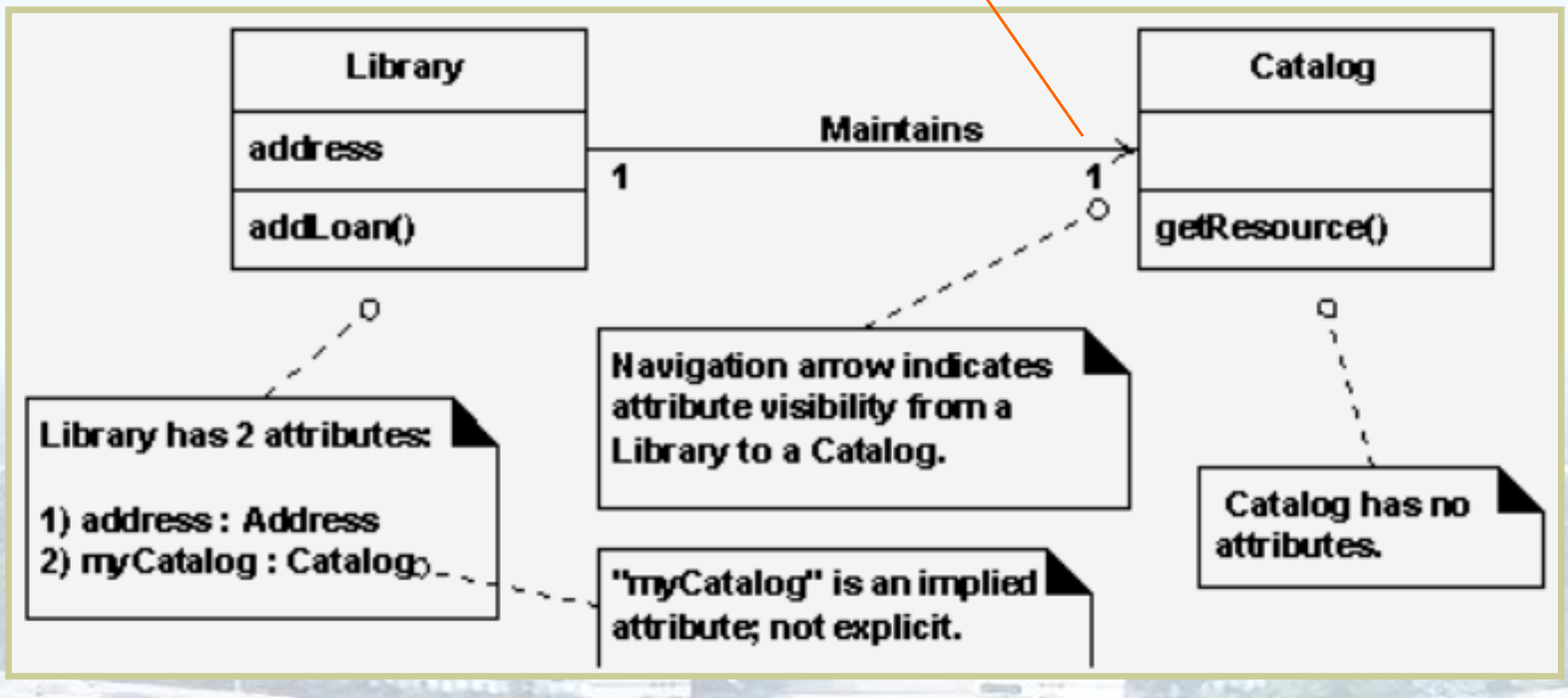
- 设计类图:
  - Design class diagrams depict software class definitions.
    - Methods.
    - Attribute visibility.
    - Simple attributes.



4.13 面向对象系统设计

4.13.10 产生设计类图

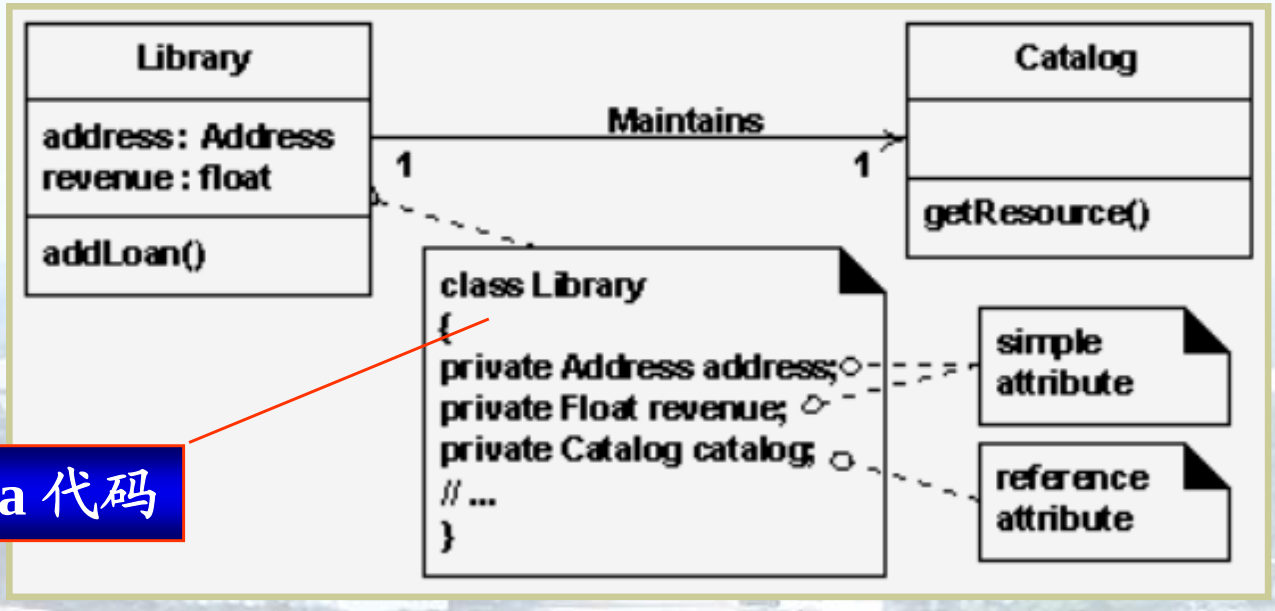
- 指示出属性视域:
  - Attribute visibility should be shown when a permanent connection is required.
    - How? Add a UML navigation arrow on the association.



4.13 面向对象系统设计

4.13.10 产生设计类图

- 简单属性和引用属性:
  - Simple attributes are relatively primitive types.
    - Number, string, boolean, address, …
  - Reference attributes refer to other complex objects.
    - Implied by associations in the diagram.

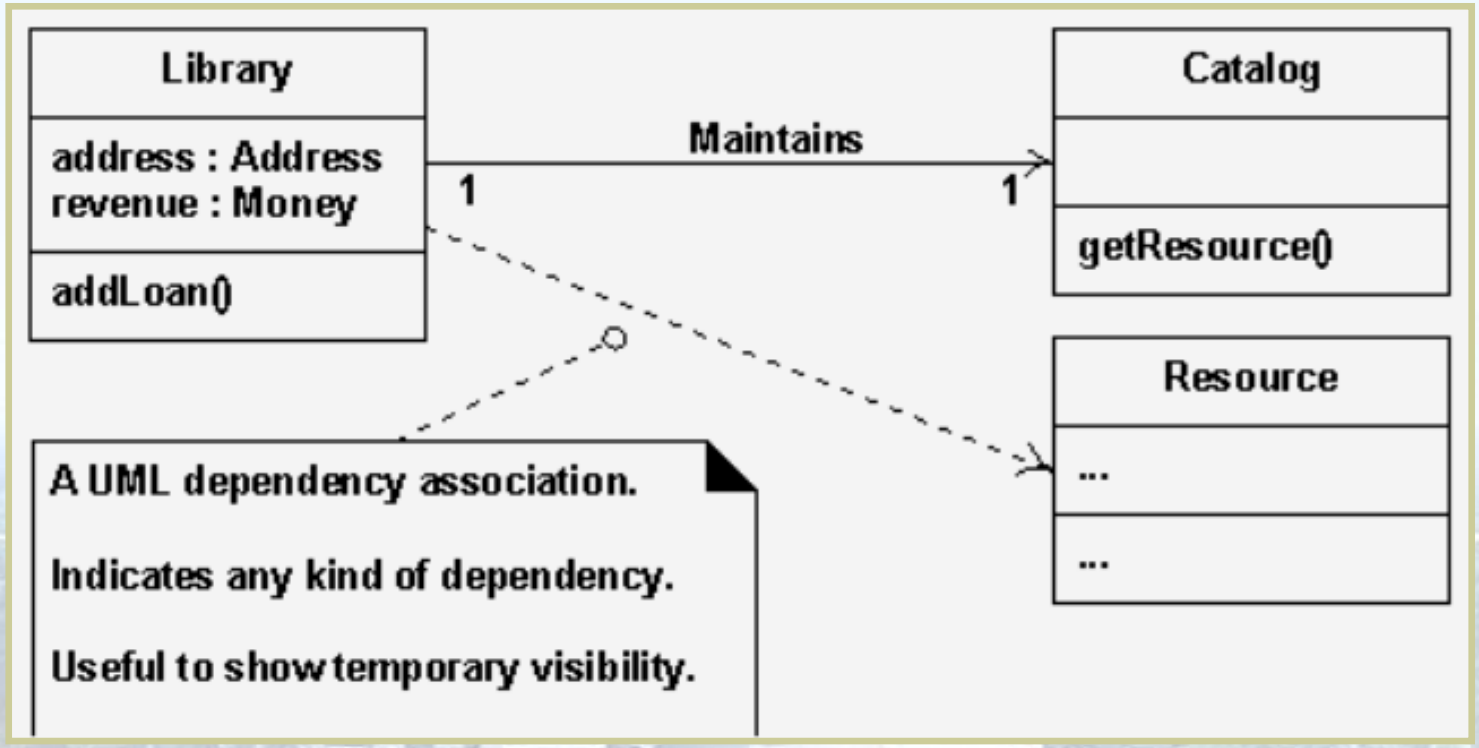


对应的 Java 代码

4.13 面向对象系统设计

4.13.10 产生设计类图

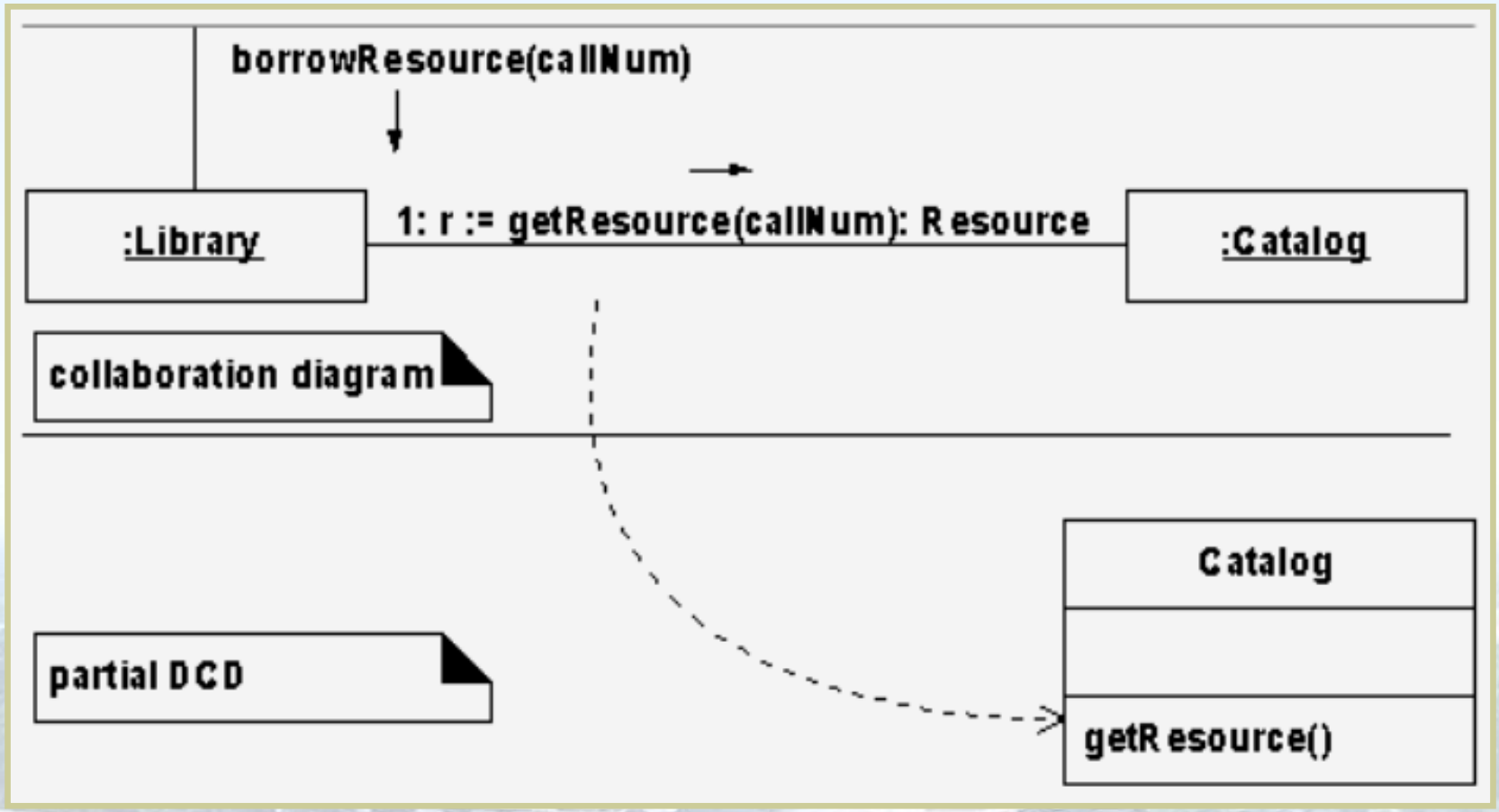
- 指示出临时性的视域:
  - Temporary visibility is associated with parameter or local visibility.



4.13 面向对象系统设计

4.13.10 产生设计类图

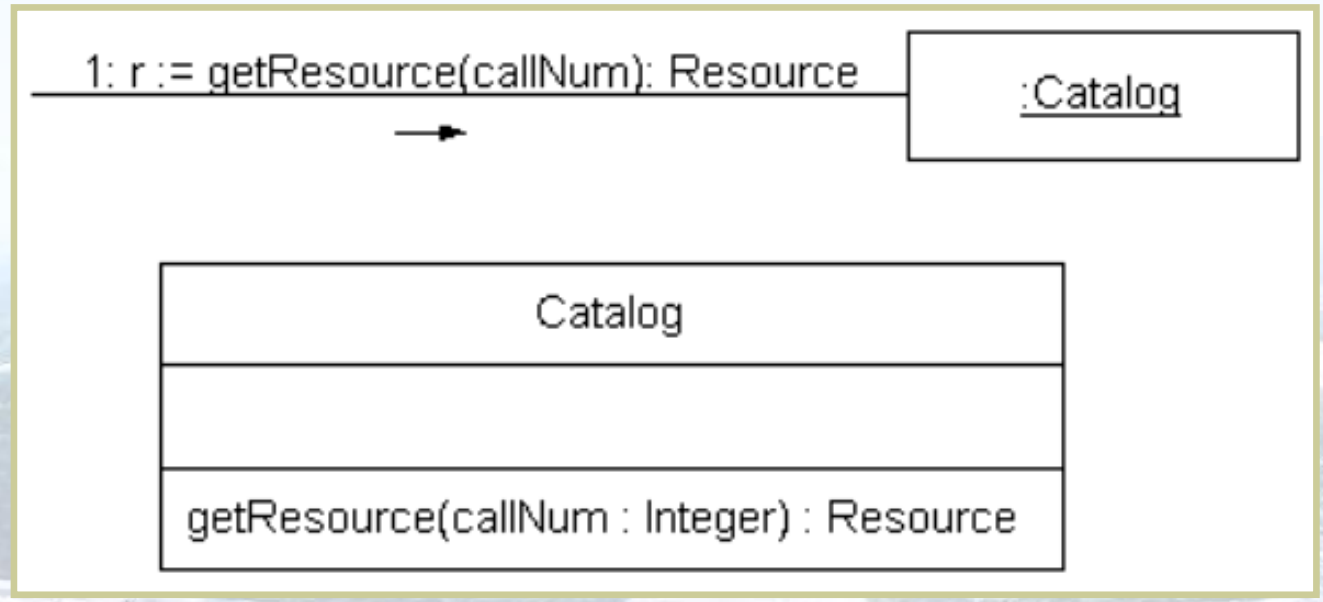
- 在类中加入方法:



4.13 面向对象系统设计

4.13.10 产生设计类图

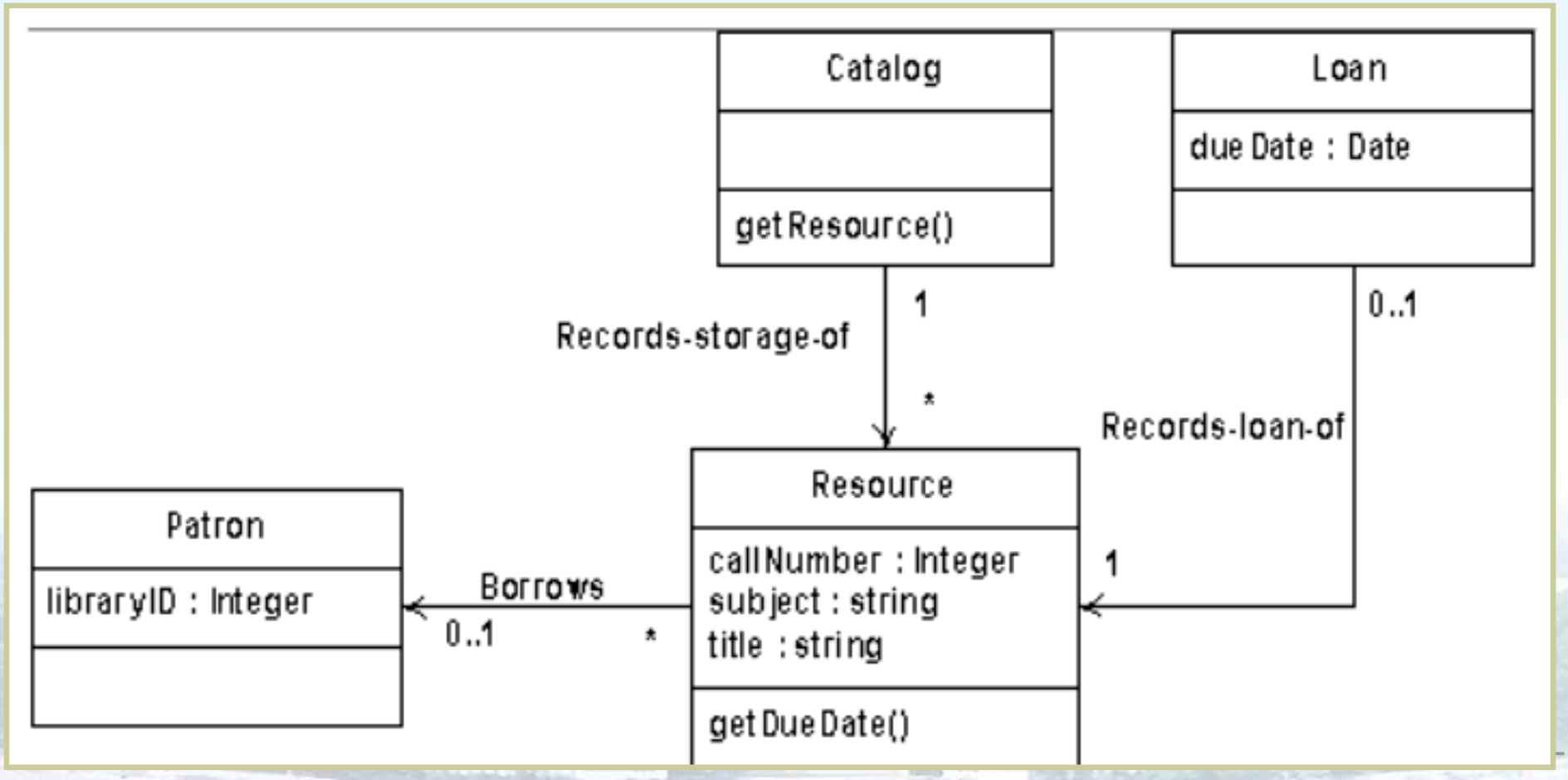
- 指示出方法的接口细节：
  - Method details are optional.
  - When are they useful?
    - Automatic code generation.
    - Communication of detailed specifications to developers.



4.13 面向对象系统设计

4.13.10 产生设计类图

- 设计类图示例:



4.13 面向对象系统设计

4.13.11 设计模式及其应用

- 模式的种类:
  - A major contribution of the work in design patterns has been the organization of design patterns into three categories:
    - Creational Patterns（生成模式）
    - Structural Patterns（结构模式）
    - Behavioral Patterns（行为模式）

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- 生成模式:
  - Creational Patterns address "lifecycle" operations, such as creation of new objects.
  - Creational Patterns also encapsulate knowledge about what classes are created and how instances are created.
  - Kinds of Creational Patterns:
    - Class-based creational patterns use *inheritance* to vary the kind of object that's created. (e.g., Factory Method)
    - Object-based creational patterns use *composition* to parameterize the kind of object the system creates. (e.g., Abstract Factory)

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- 典型的生成模式:
  - Factory Method
  - Abstract Factory
  - Singleton

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- 典型的生成模式:
  - Factory Method
  - Abstract Factory
  - Singleton

4.13 面向对象系统设计

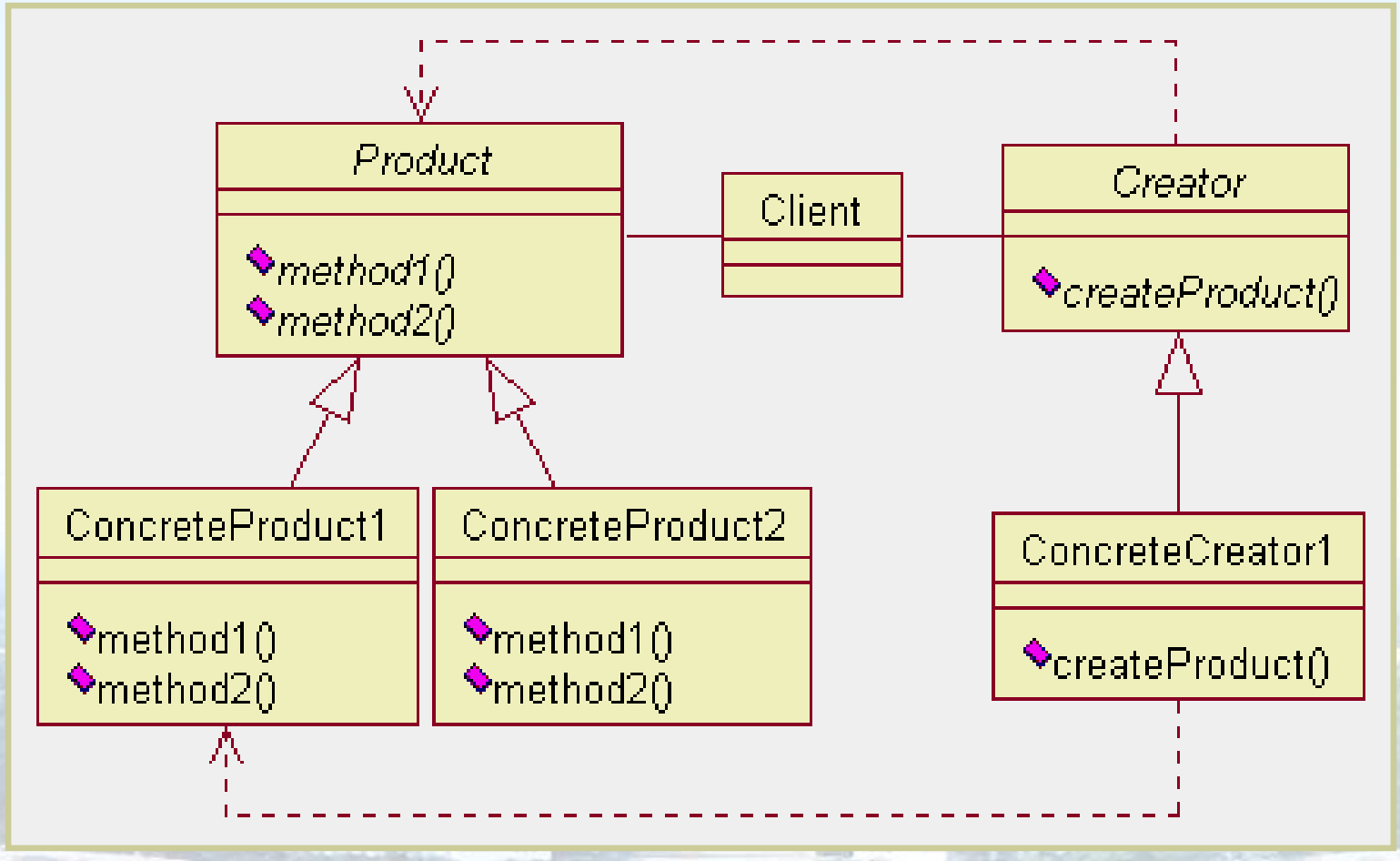
4.13.11 设计模式及其应用 – 生成模式

- **Factory Method 模式:**
  - The Problem:
    - We want to be able to define an interface for object creation so that we can have more flexibility in the kind of objects that are created. (e.g., from a specific set of subclasses)

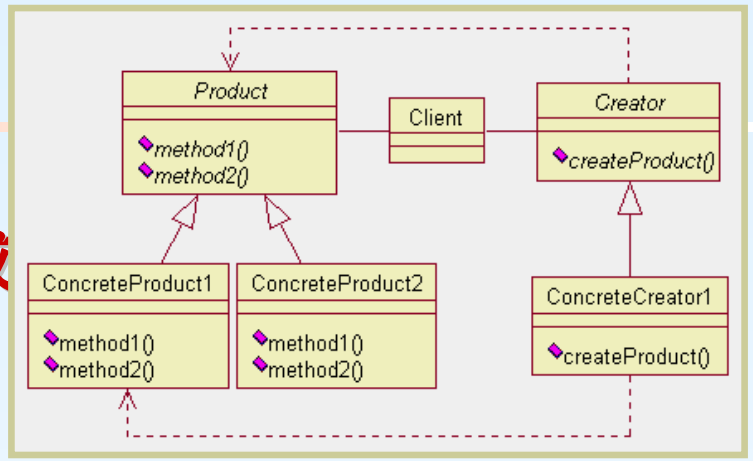
4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- **Factory Method** 模式的类图:



4.13 面向对象系统设计

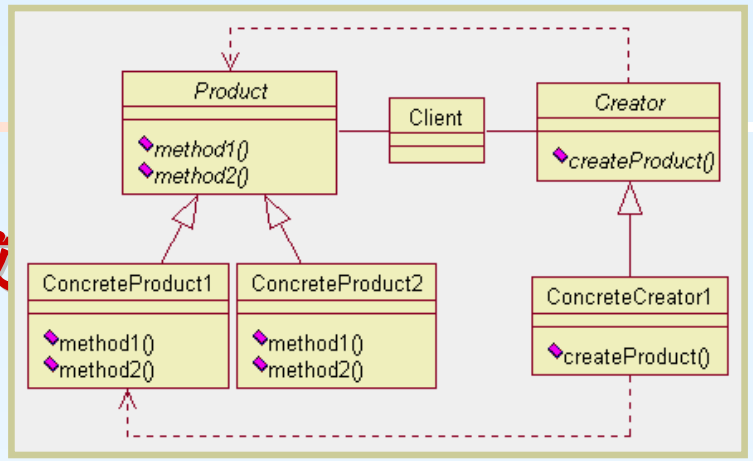


4.13.11 设计模式及其应用 – 生成

- **Factory Method 模式的内涵:**
  - We use the class Product to indicate some component which must be created in the system.
  - The Client object is insulated from the details of which subclass of Product is constructed, and is willing to use only the interface defined in the Product class (as shown, only method1() and method2()).
  - The methods in the abstract Product class are of no concern to the creational issues, only to the Product class created.
  - The Product class is abstract, but needs not be.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成



- **Factory Method 模式的内涵:**
  - The factory method is createProduct(), which could be declared as:

    Product createProduct(); // 实际生成的是 **Product**子类的实例
  - The Client object ultimately gets a reference to an instance of one of the Product's subclasses and can rely only on the public interface defined at the Product level.
  - The Creator class can be abstract or concrete.

4.13 面向对象系统设计

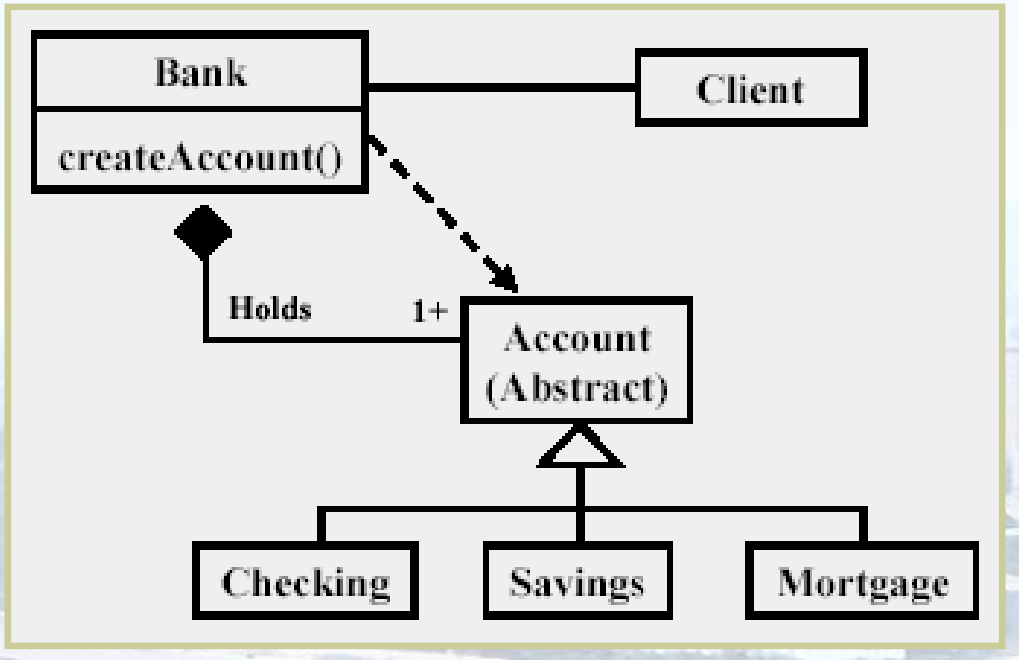
4.13.11 设计模式及其应用 – 生成模式

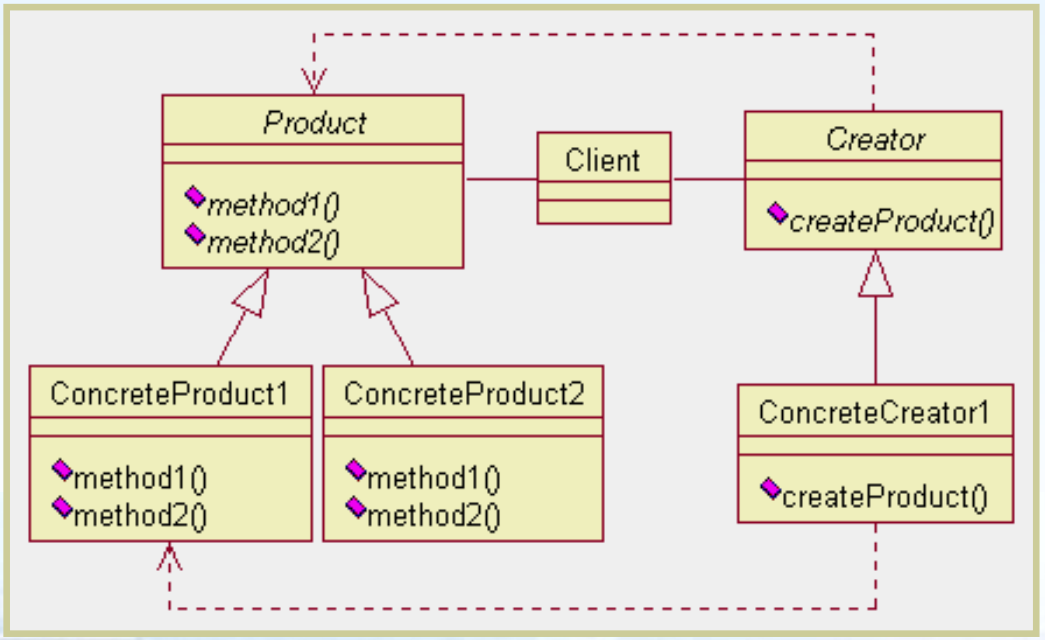
- 适用**Factory Method** 模式的场合:
  - When a client can‘t anticipate（无法预计） the class of the object it must create.
  - When an application has parallel hierarchies that must be created. (e.g., when there are two separate implementations of a set of windowing classes)

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- 使用Factory Method 模式的例子:

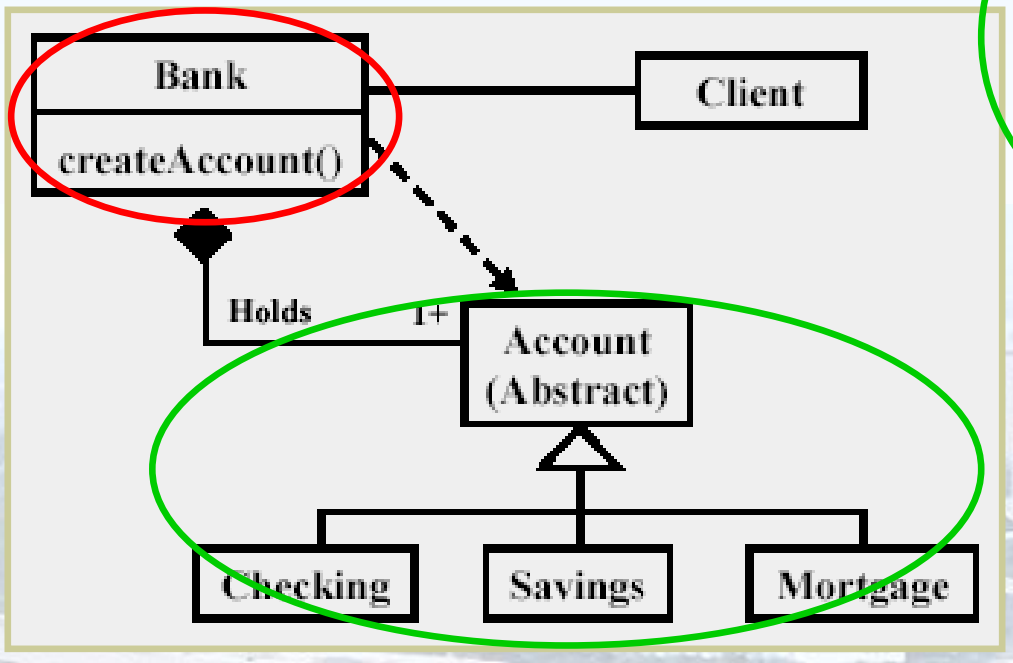


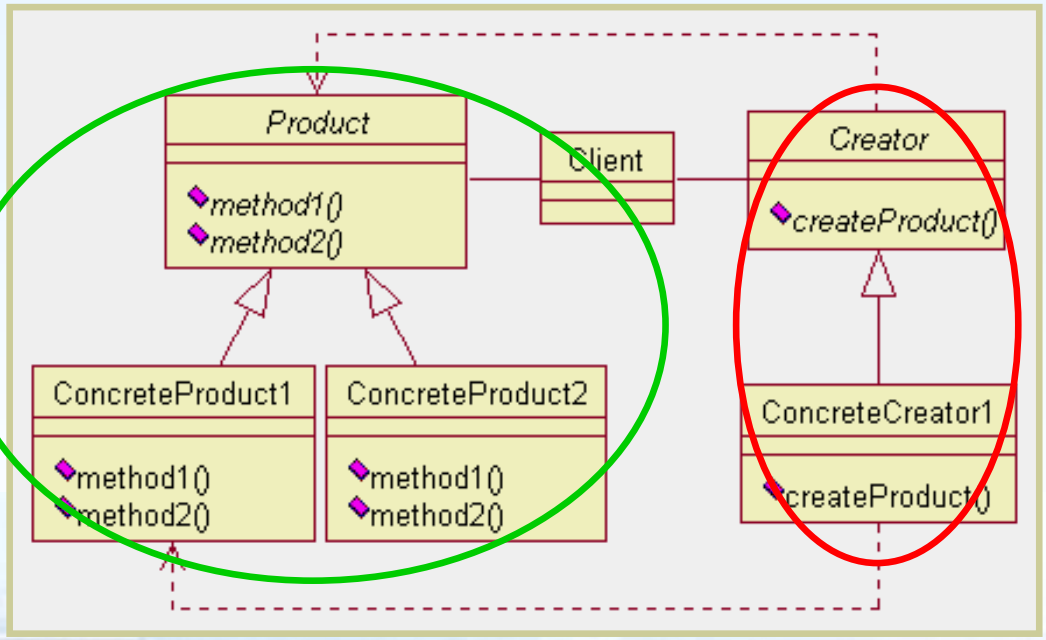


4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- 使用Factory Method 模式的例子:





4.13 面向对象系统设计

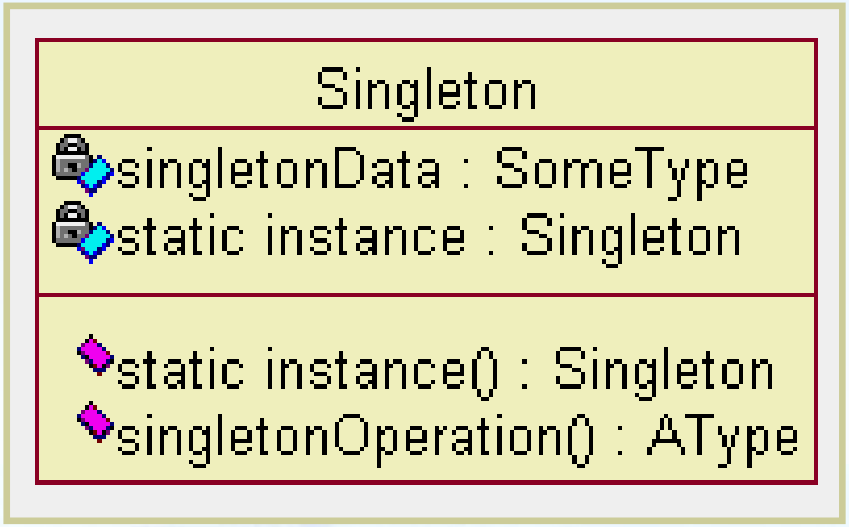
4.13.11 设计模式及其应用 – 生成模式

- **Singleton 模式:**
  - The Problem:
    - Ensure that only one instance of a given class can exist.
  - The problem can be varied slightly to ask how to create exactly n instances of a class. This can be accomplished by a slight modification to our solution.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- **Singleton** 模式的类图:



A Singleton class provides a static method, instance(), to be called when a client wants to access the unique instance of the class. If the unique instance does not yet exist, the instance() method creates it.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 生成模式

- 使用 **Singleton** 模式的例子:
  - In the banking system described earlier, we need to ensure that there is only one instance of class Bank in any given program.
  - One thing we'll want to do is to ensure that all constructors of class Bank are not public, so that no one can write code such as

    Bank myOwnBank;

    anywhere in the program they wish.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- 结构模式:
  - Often a single object can comprise many parts, or be an entry point into a network of related objects.
  - Structural Patterns address ways to compose classes and objects to form larger structures.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- 典型的结构模式:
  - Adapter
  - Bridge
  - Composite
  - Façade
  - Proxy

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- 典型的结构模式:
  - Adapter
  - Bridge
  - Composite
  - Façade
  - Proxy

4.13 面向对象系统设计

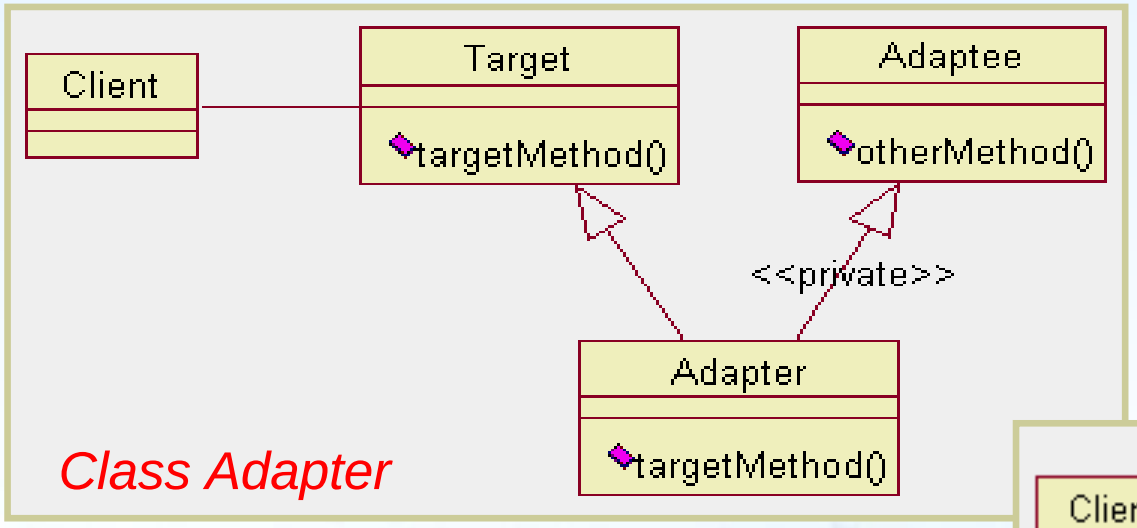
4.13.11 设计模式及其应用 – 结构模式

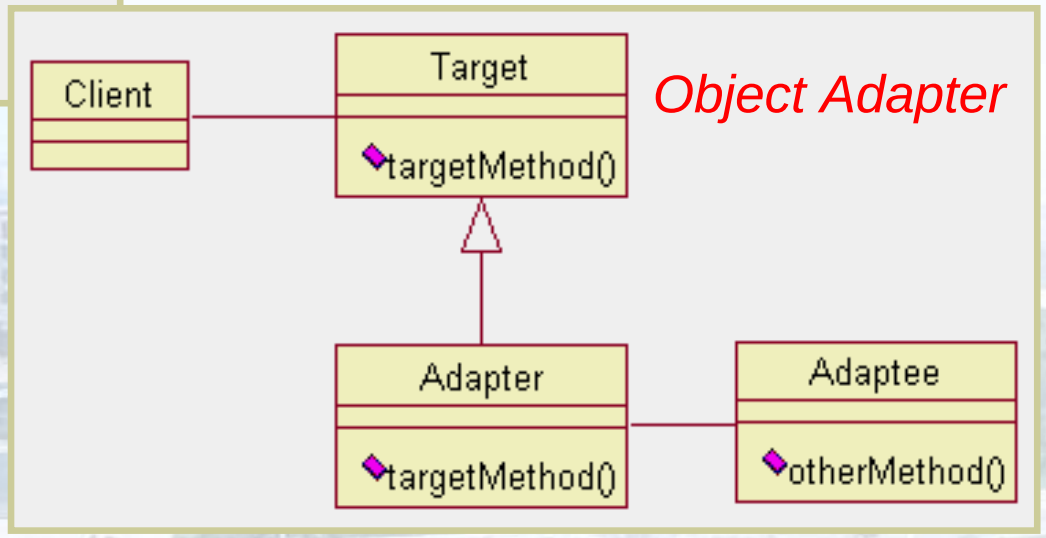
- **Adapter 模式:**
  - The Problem:
    - We want to convert the interface of a class into another interface that clients expect.
  - This will enable class interaction that would be inconvenient otherwise.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- **Adapter** 模式的类图:



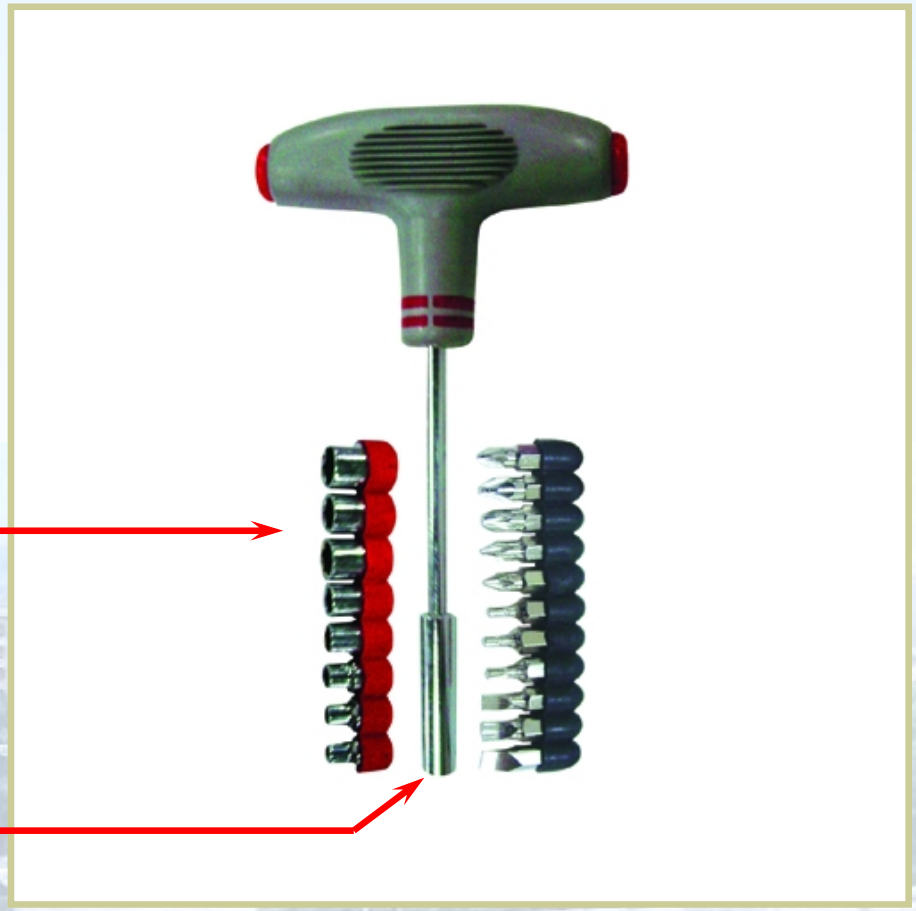


The goal is to make targetMethod() available to users of the Target class but use otherMethod() in Adaptee to provide the "real" service.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- 用生活中的例子来类比:



4.13 面向对象系统设计

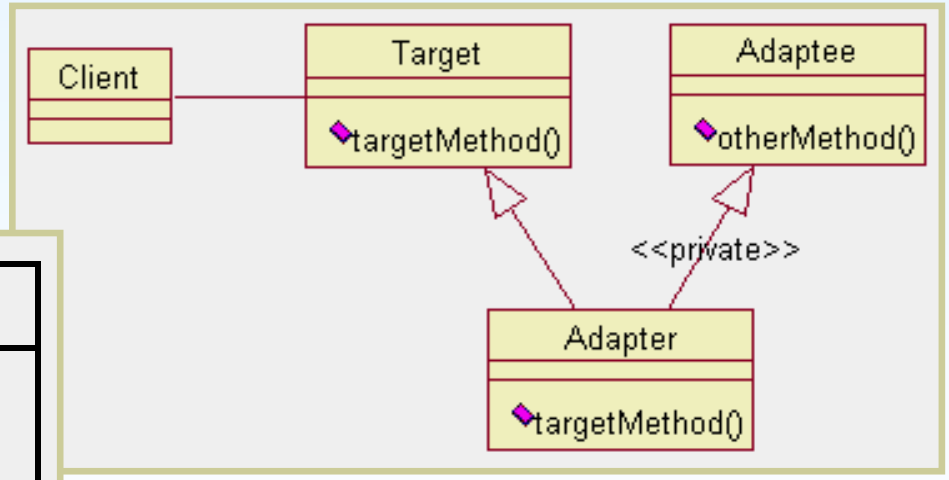
4.13.11 设计模式及其应用 – 结构模式

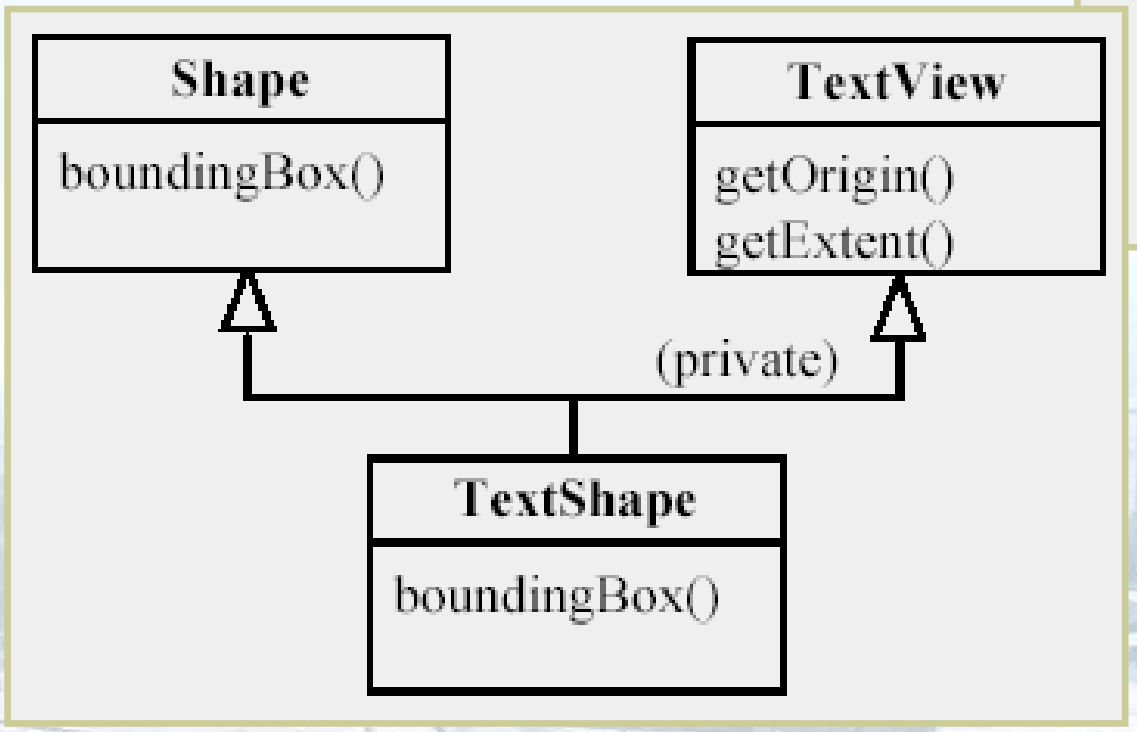
- 适用 **Adapter** 模式的场合:
  - When we have a coherent, consistent set of classes and want to add in a new class (say, from a third party) that has a very different style of interface.
  - This could happen when you have a set of GUI classes and buy a specialized class from another vendor.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- 使用 **Class Adapter** 模式的例子:

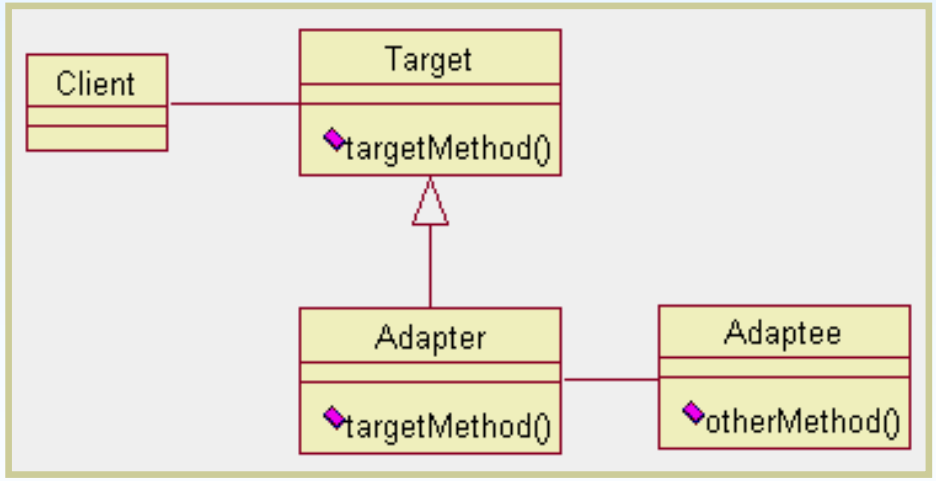


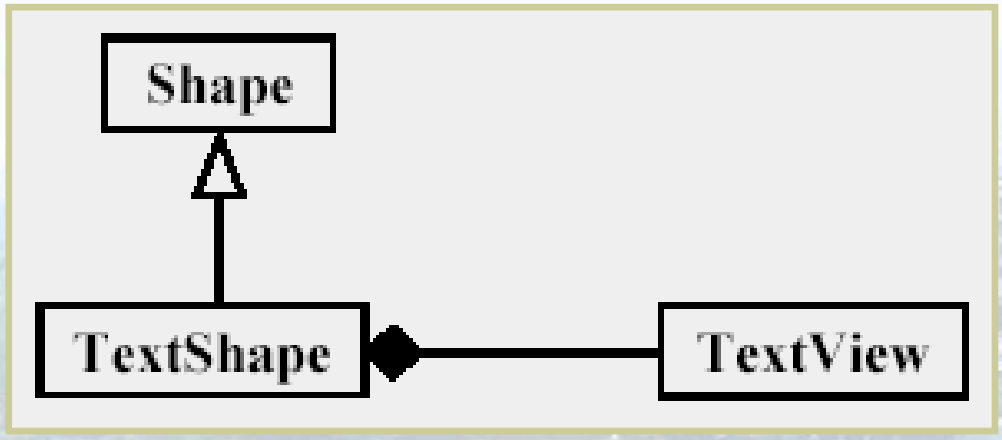


4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- 使用 **Object Adapter** 模式的例子:





4.13 面向对象系统设计

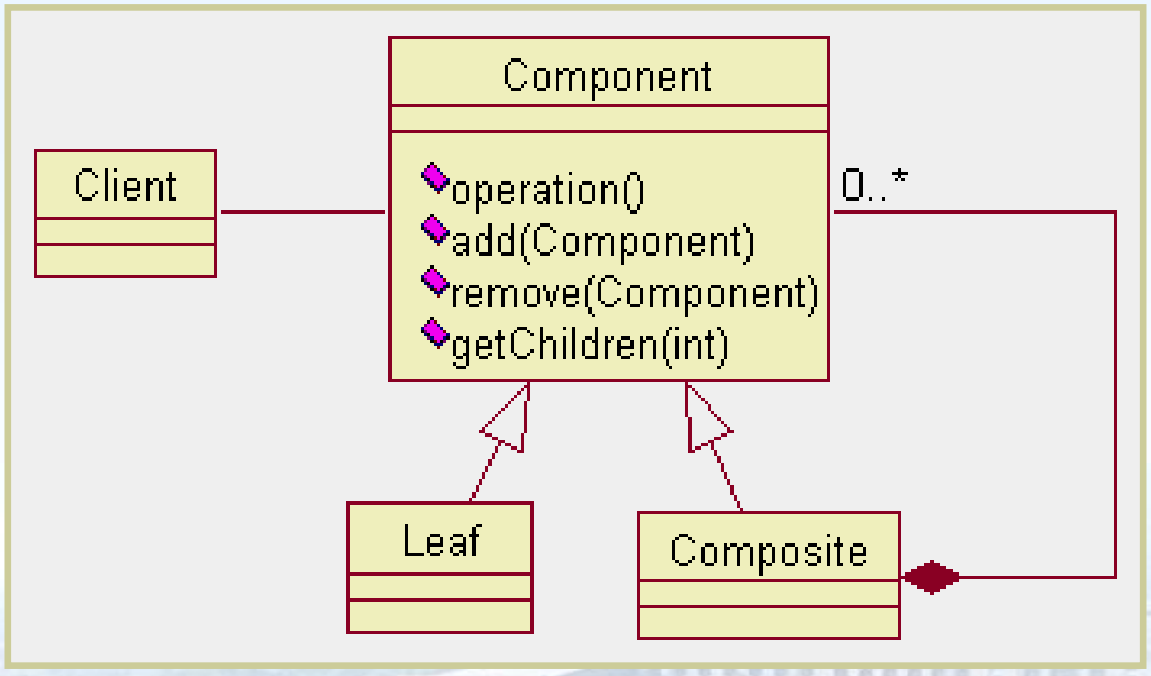
4.13.11 设计模式及其应用 – 结构模式

- **Composite 模式:**
  - The Problem:
    - We want to compose objects into tree structures representing whole-part hierarchies so that clients can treat individual objects and containers of objects uniformly.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 结构模式

- **Composite** 模式的类图:



The Client object uses the common protocol, indicated here by the method named operation(), which is a placeholder for the semantic methods of the class, whereas the add(), remove(), and getChildren() methods provide the interface to the container side of the class.

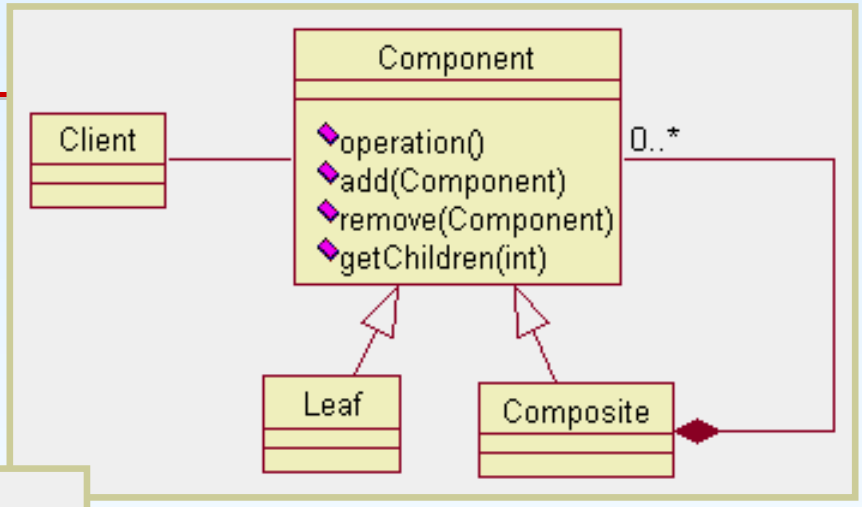
4.13.11 设计模式及其应用 – 结构模式

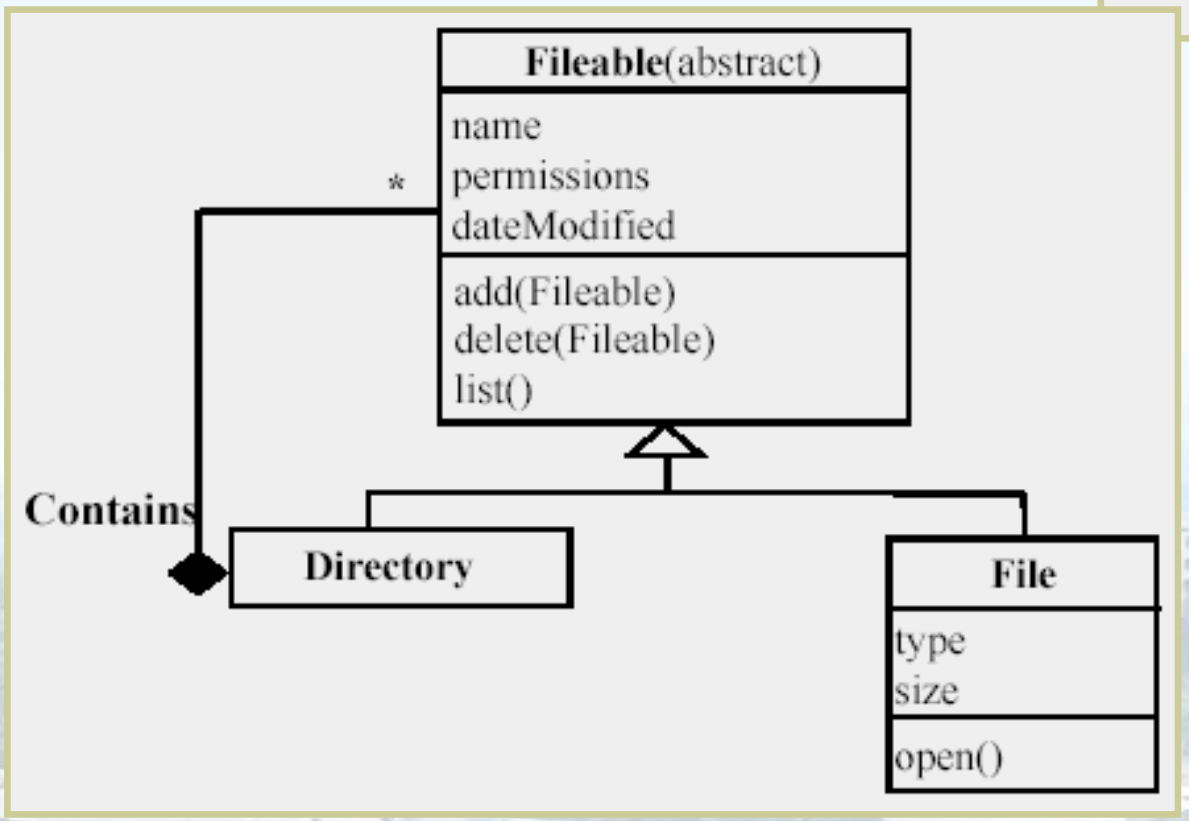
- 适用 **Composite** 模式的场合:
  - Use the Composite Pattern when you have a whole-part hierarchy that has significant similarities among the things which are containers and the things which are not.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 -

- 使用 **Composite** 模式的例子:





4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 行为模式

- 行为模式:
  - Behavioral Patterns are concerned with proper allocation of services to objects in a system.
  - This covers both the algorithms that must be executed in a program as well inter-object communication.
  - Kinds of Behavioral Patterns:
    - Behavioral class patterns, such as the Template Pattern, use inheritance to distribute behavior between classes.
    - Behavioral object patterns, such as the Command Pattern, use object composition to distribute behavior between objects.

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 行为模式

- 典型的行为模式:
  - Command
  - Mediator
  - Observer
  - State
  - Strategy
  - Template method

4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 行为模式

- 典型的行为模式:
  - Command
  - Mediator
  - Observer
  - State
  - Strategy
  - Template method

4.13 面向对象系统设计

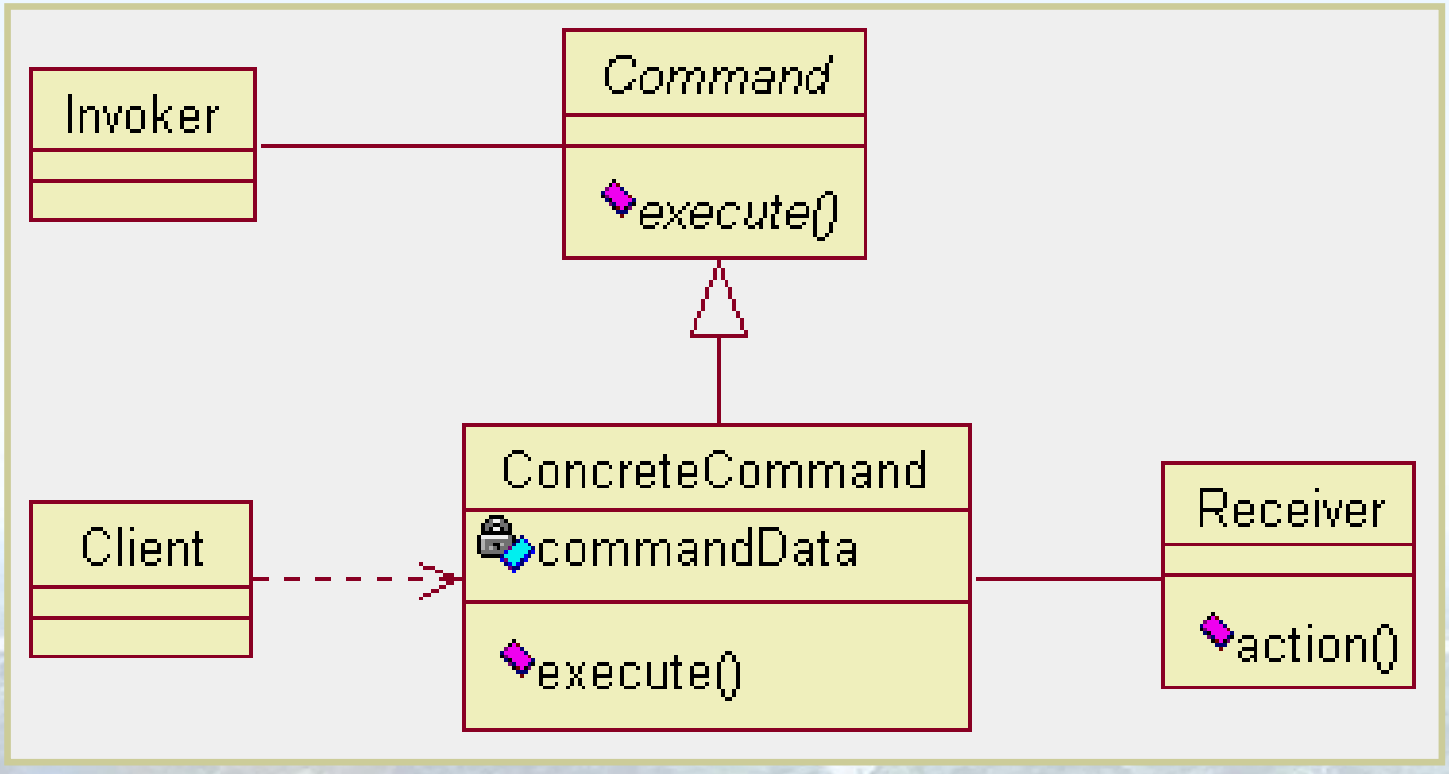
4.13.11 设计模式及其应用 – 行为模式

- **Command 模式:**
  - The Problem:
    - We want to be able to encapsulate service requests as objects, so we can pass them as parameters to other objects for execution.
  - This is one of two patterns that give an object-oriented way of encapsulating functions. The other pattern is the Strategy Pattern.
    - Both describe a means of replacing old-style C language code that relied on passing function pointers.
  - By using objects to encapsulate functions, we can more easily package the function with the data it needs to perform its task, and we can also gain some advantage through inheritance and polymorphic behavior.

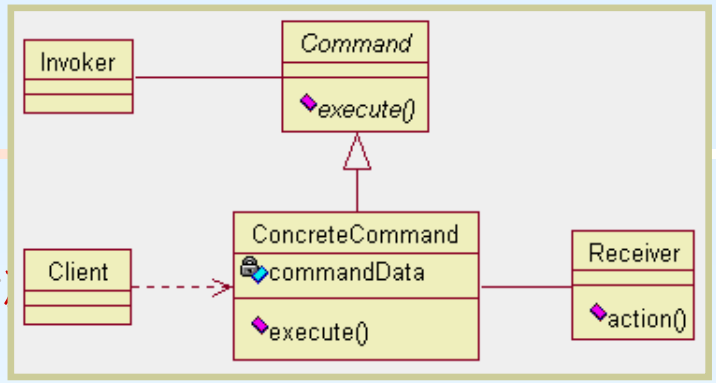
4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 行为模式

- **Command** 模式的类图:



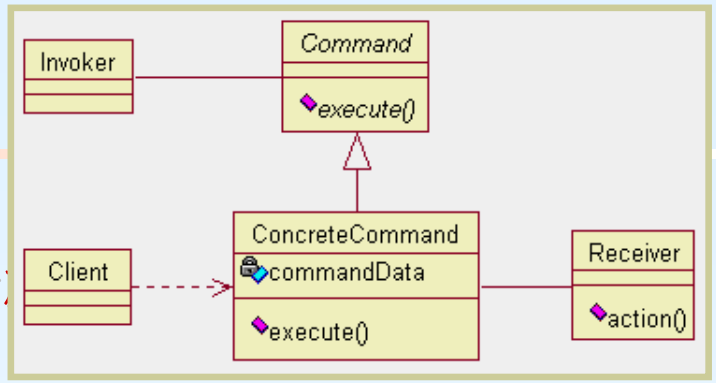
4.13 面向对象系统设计



4.13.11 设计模式及其应用 – 行为

- **Command 模式的内涵:**
  - The (abstract) Command class declares an interface for executing a command.
  - A ConcreteCommand binds the Receiver with an action.
  - A Client object creates the ConcreteCommand object and sets its Receiver.
  - The Invoker asks the Command to carry out its action. The Invoker could be a command processor (i.e., it queues Command objects and invokes execute() on each in turn).
  - A ConcreteCommand can have other methods, as needed.
  - Command objects are an object-oriented way to do callbacks.

4.13 面向对象系统设计



4.13.11 设计模式及其应用 – 行为

- 用生活中的例子来类比 Command 模式:
  - 场景: 某饭店中的招待行为。
  - 角色: **Client** – 吃饭的顾客; **Invoker** – 女服务员; **Receiver** – 厨师; **Command** – 顾客自己填写的订单（可能各不相同）;
    - 订单是厨师与action（按订单加工）之间的绑定。
    - 厨师接受订单并对之负责。
    - 顾客通过产生订单（形成commandData）来给出命令的内容。
    - 女服务员通过将订单放入待加工队列而具体地将订单激活。
  - 这种给出命令的方式不限于饭店，例如 … …

4.13 面向对象系统设计

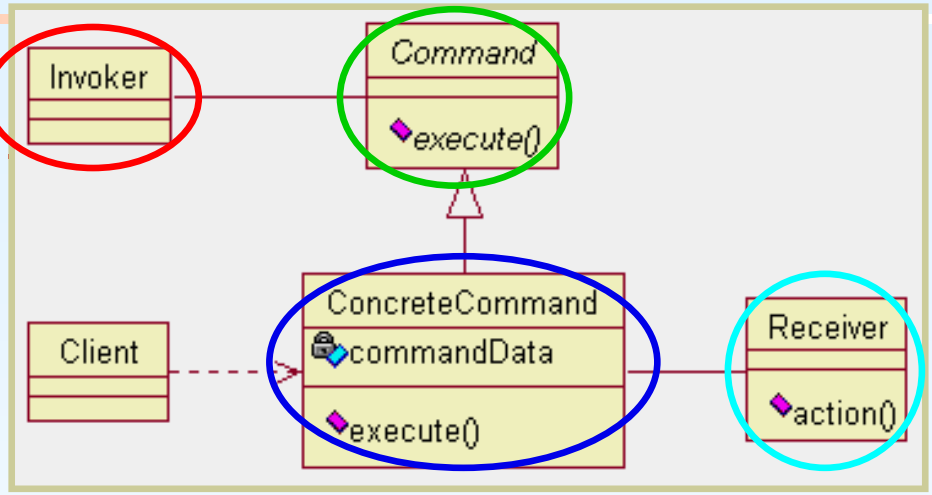
4.13.11 设计模式及其应用 – 行为模式

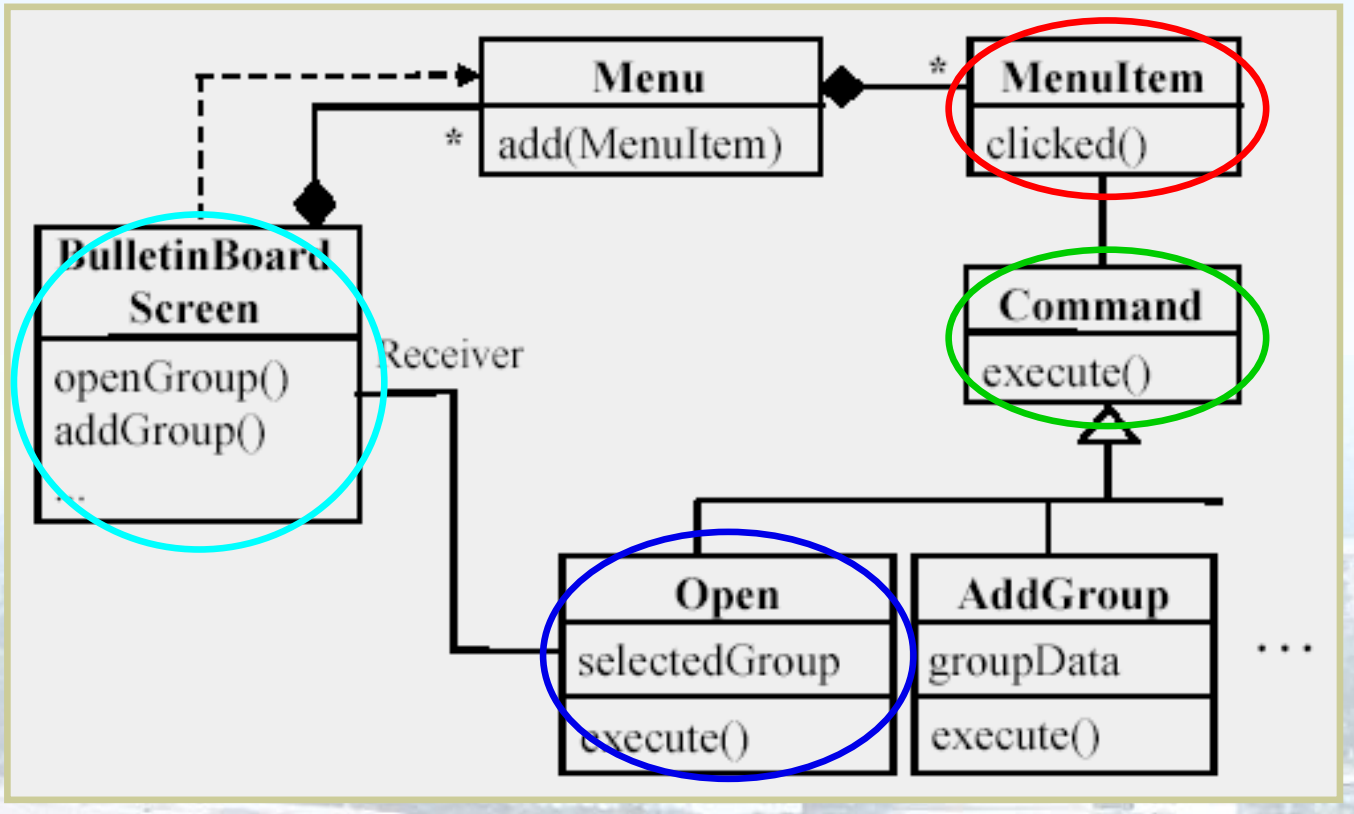
- 适用 **Command** 模式的场合:
  - When you want to have a means to encapsulate requests as objects with a standard execution convention.
    - One example is an action handler that wants to respond to selection of an item in a menu.
    - Another example is the use of callbacks in a distributed system.

4.13 面向对象系统设计

4.13.11 设计模式及其应用

- 使用 **Command** 模式的例子:





4.13 面向对象系统设计

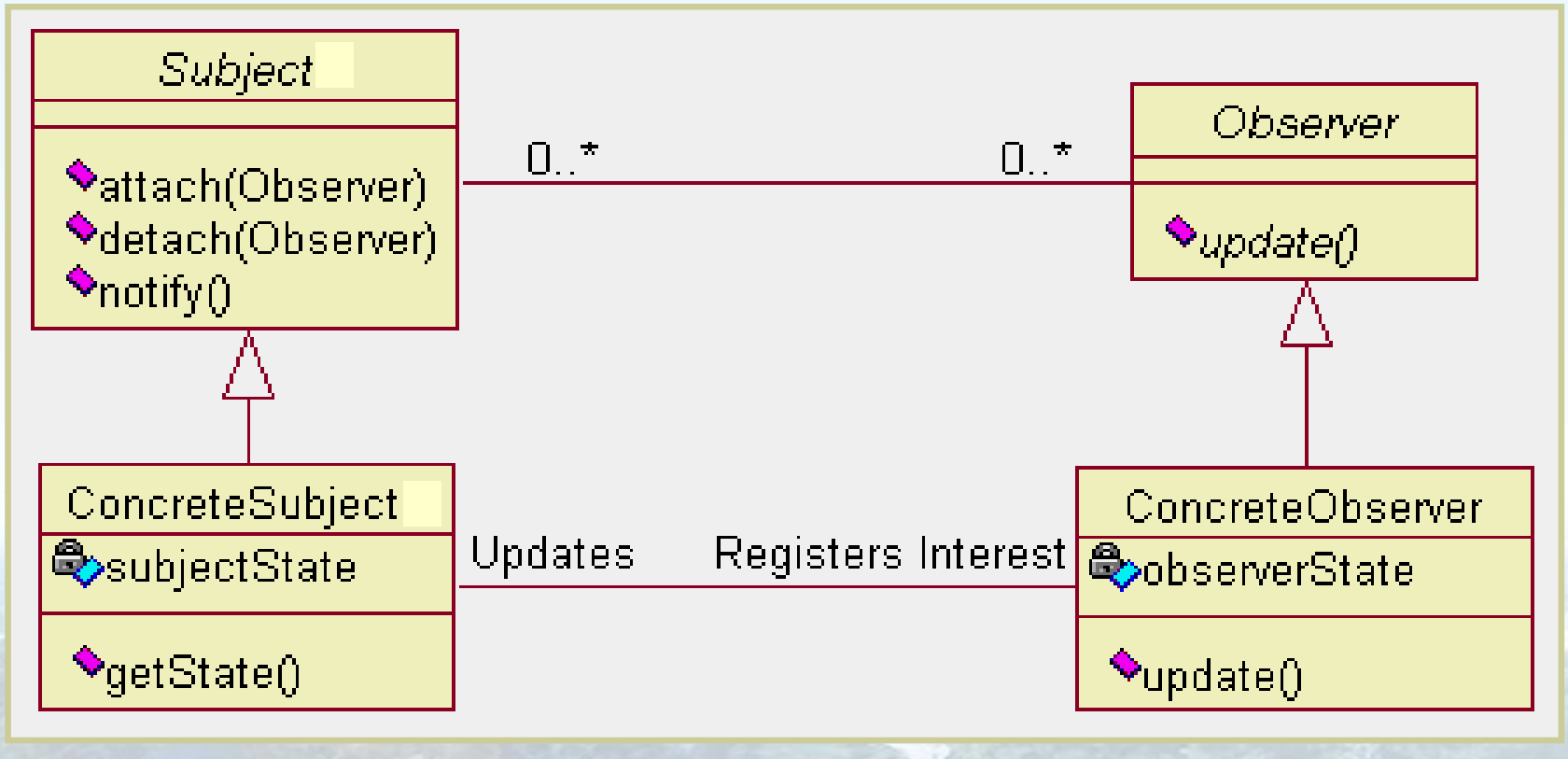
4.13.11 设计模式及其应用 – 行为模式

- **Observer 模式:**
  - The Problem:
    - We want to allow a one-to-many dependency among objects so that when one object changes, the others are notified.

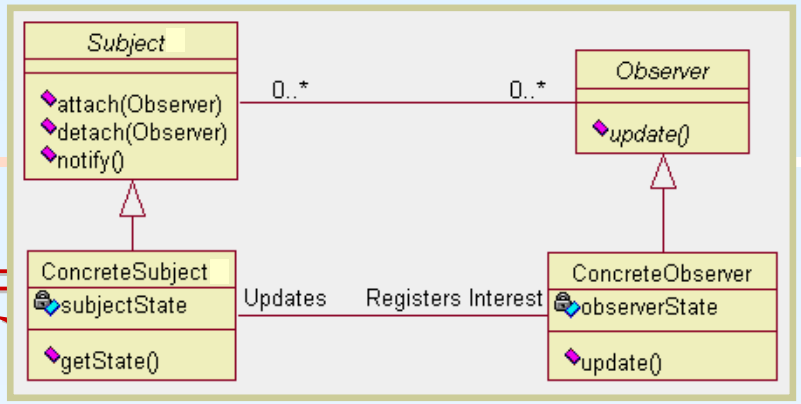
4.13 面向对象系统设计

4.13.11 设计模式及其应用 – 行为模式

- **Observer** 模式的类图:



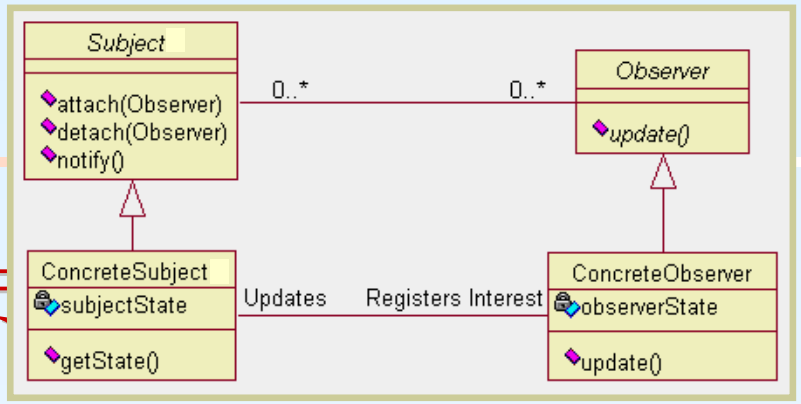
4.13 面向对象系统设计



4.13.11 设计模式及其应用 – 行

- **Observer 模式的内涵:**
  - A good name for Subject would be Observable（可观察量）.
  - Invoking the attach(Observer) method is sometimes referred to as registering interest.
  - Interest registration can be carried out by a third party object that invokes the attach(Observer) method on the Subject.
  - The notify() method loops through all the Observer objects who have registered interest in the change and calls their update() method.

4.13 面向对象系统设计



4.13.11 设计模式及其应用 – 行

- 用生活中的例子来类比 **Observer** 模式:
  - 场景: 某拍卖行中的拍卖行为。
  - 角色: **Subject** – 拍卖师的动作; **ConcreteSubject** – 当前的出价; **Observer** – 出价人的动作; **ConcreteObservers** – 不同的、有自己之容忍上限的出价人;
    - 由于出价人必须在拍卖前注册，所以拍卖师知道所有的出价人。
    - 当前出价的值（**subjectState**）是出价人最感兴趣的。
    - 出价人需要知道当前出价的变化，来决定自己下一步的行动。
  - 这种通知的方式不限于拍卖行，例如 …….

4.13 面向对象系统设计

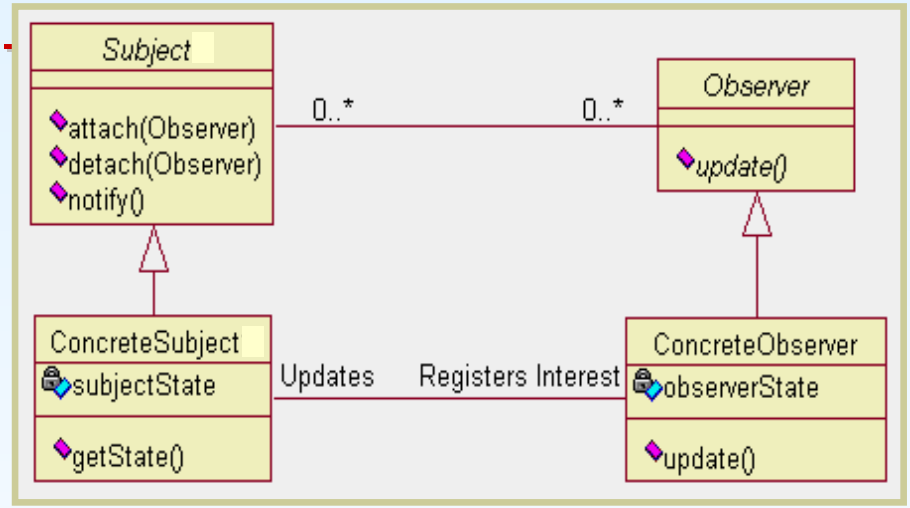
4.13.11 设计模式及其应用 – 行为模式

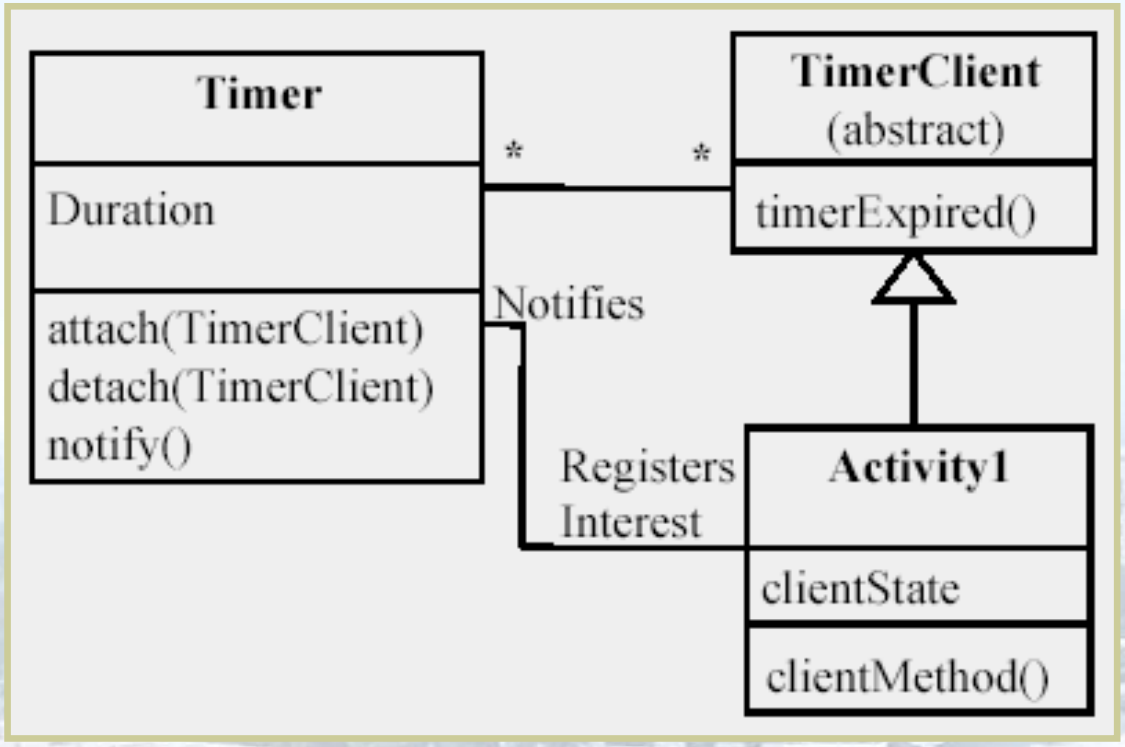
- 适用 **Observer** 模式的场合:
  - When you want a level of decoupling so that the changed object need not be aware at compile time as to who the Observers are.

4.13 面向对象系统设计

4.13.11 设计模式及其应用

- 使用 **Observer** 模式的例子:

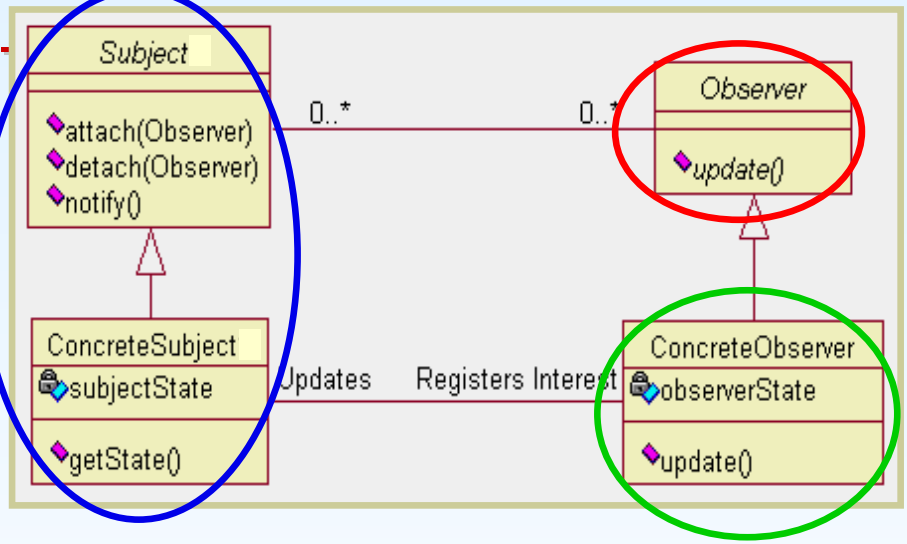


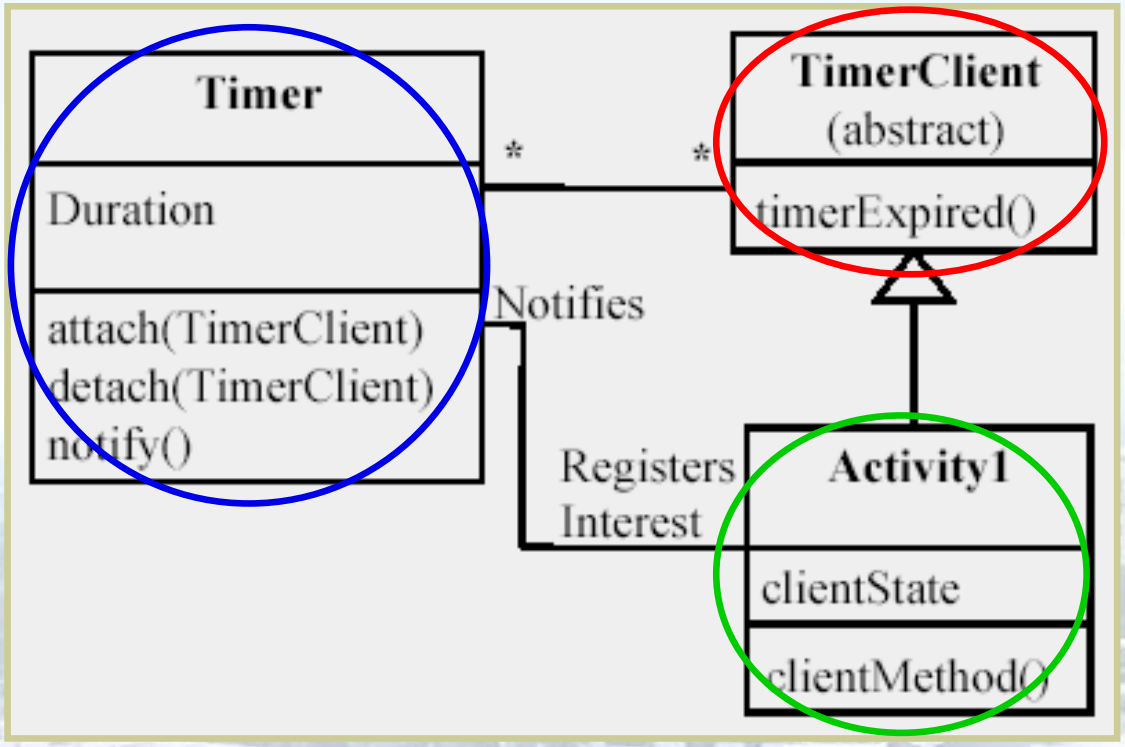


4.13 面向对象系统设计

4.13.11 设计模式及其应用

- 使用 **Observer** 模式的例子:





4.13 面向对象系统设计

4.13.12 分析模式简介

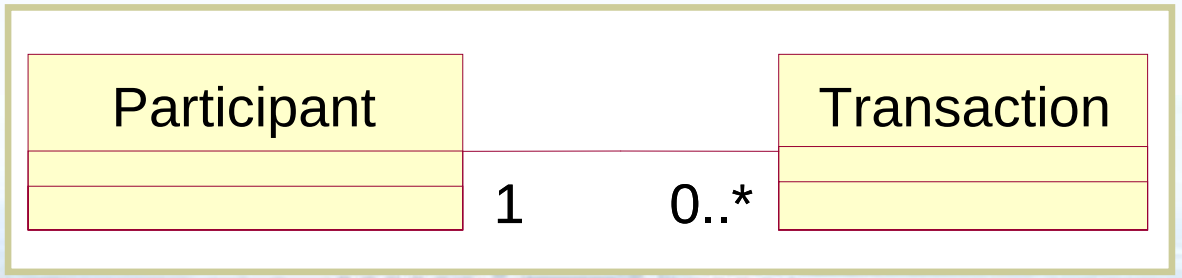
- 经典著作:
  - Martin Fowler, Analysis Patterns: Reusable Object Models, Addison Wesley, 1997（中国电力出版社2003年6月影印版）

4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

1. “参与者—交易”模式:

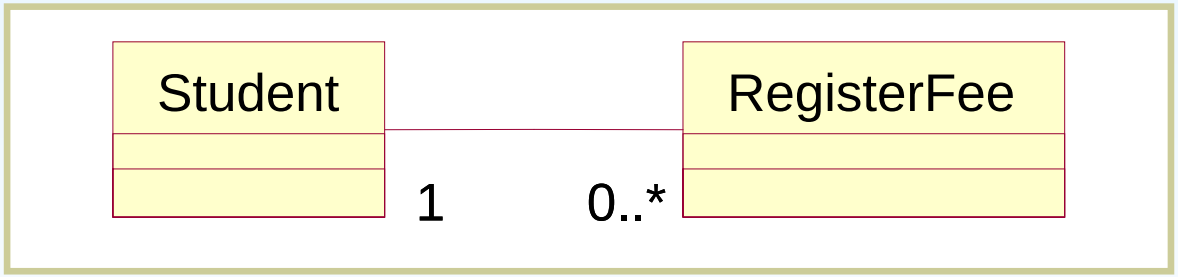
- 定义交易过程中参与交易的人或组织与交易之间的关系。
- 限定：参与者指有稳定的交易要求、固定的标识（而不是任意）的人或组织；交易则指的是发生单据往来的活动。



4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

“参与者—交易”模式的例子－学生与交纳注册费：



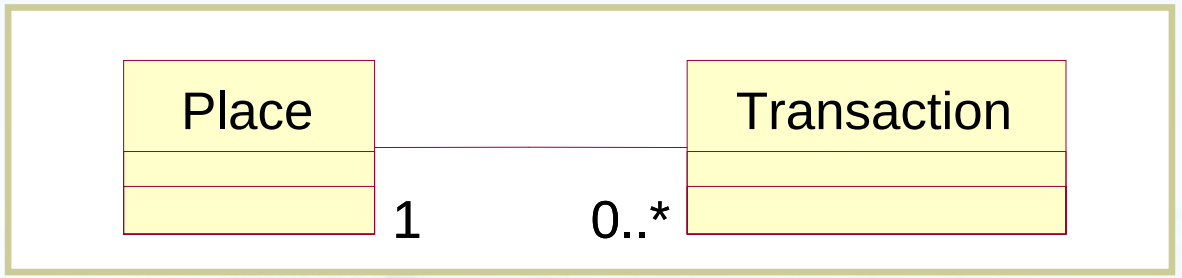
- 可采用的范例：储户与存取款；保户与保险/撤保；手机用户与缴纳话费；一般纳税人与缴税；……
- 不宜采用的范例：游客与购买门票 ……

4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

2. “位置—交易”模式:

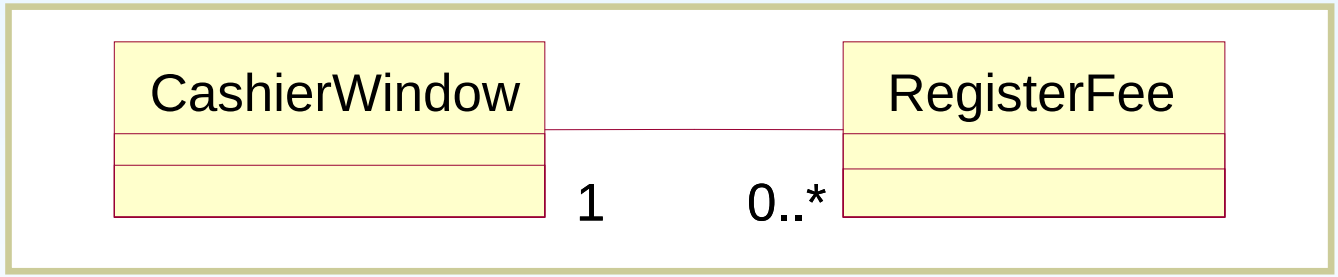
- 定义交易过程中发生交易的地点与交易之间的关系。
- 限定：位置指相对稳定、有固定标识的地点。



4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

“位置—交易”模式的例子－学校出纳窗口与交纳注册费：



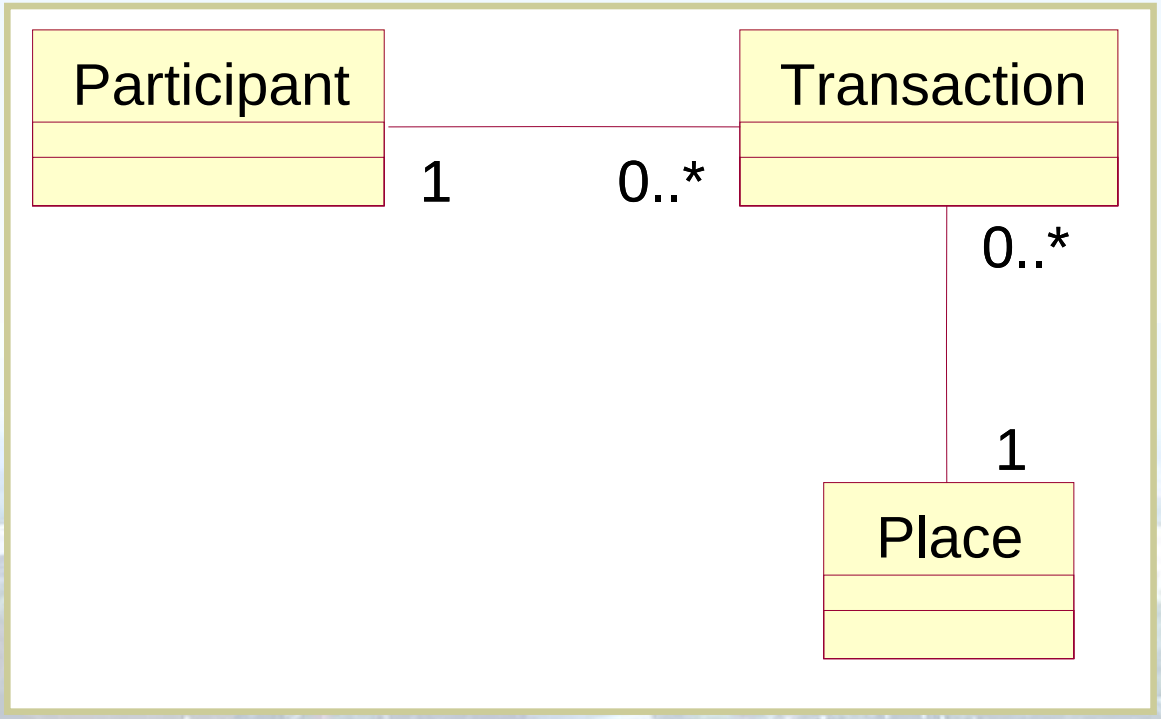
- 可采用的范例：储蓄网点与存取款；保险分理处与保险/撤保；电信营业厅与缴纳话费；税务分局与缴税；……
- 不宜采用的范例：网上/电话缴费；……

4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

3. “参与者—交易”与“位置—交易”模式的组合:

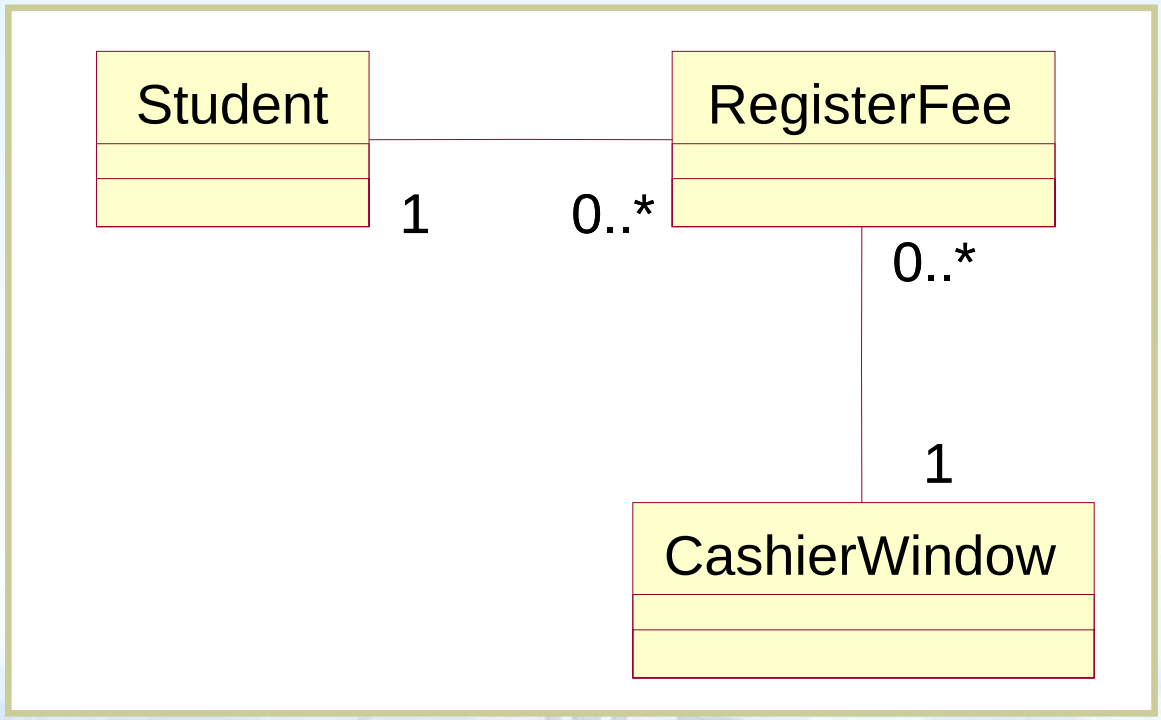
- 参与者与位置之间的关系，是由交易建立起来的。



4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

例:



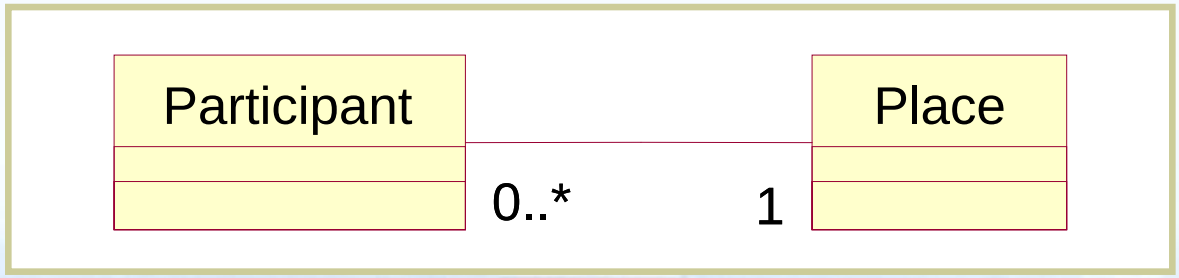
- 学生与出纳窗口之间的关系，是由于缴纳注册费建立起来的。

4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

4. “参与者—位置”模式:

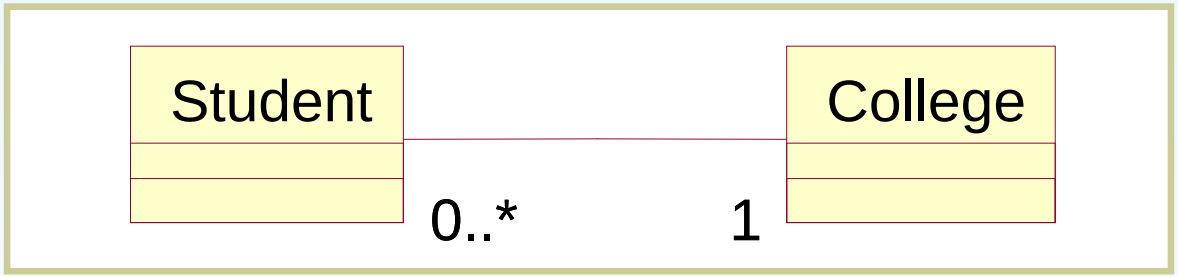
- 定义参与特定活动的人或组织与相关地点之间的关系。
- 限定：参与者与位置存在某种必然联系（但不一定是交易），而不是任意关系。



4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

“参与者—位置”模式的例子－学生与学校:



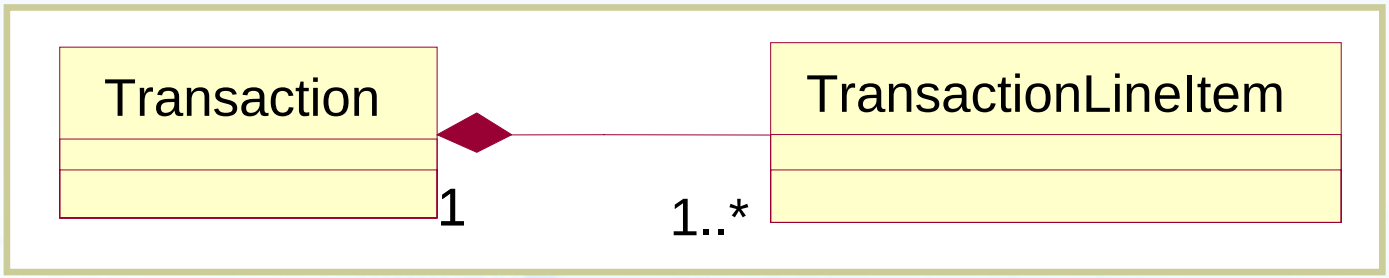
- 可采用的范例：储户与开户储蓄网点；保户与承保的保险分理处；一般纳税人与主管税务分局；……
- 不宜采用的范例：储户与通存通兑储蓄网点；……

4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

5. “交易—交易项目”模式:

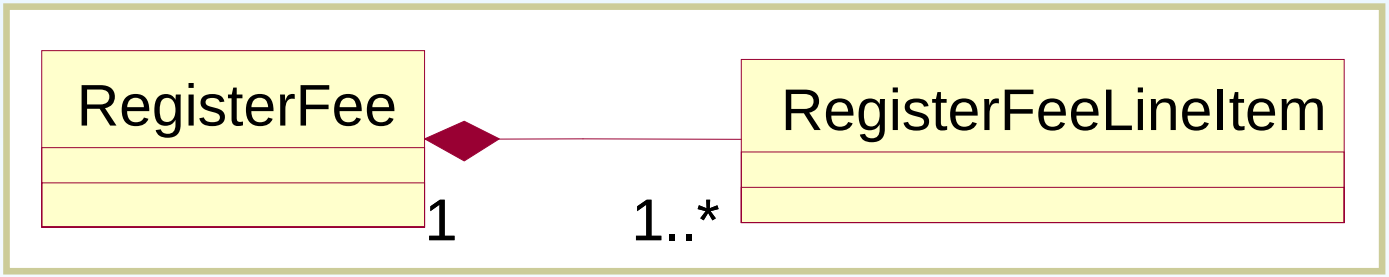
- 定义一个特定的交易与组成该交易的各交易子项之间的关系，后者是前者的有机成份。



4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

“交易—交易项目”模式的例子－缴注册费与缴费明细:



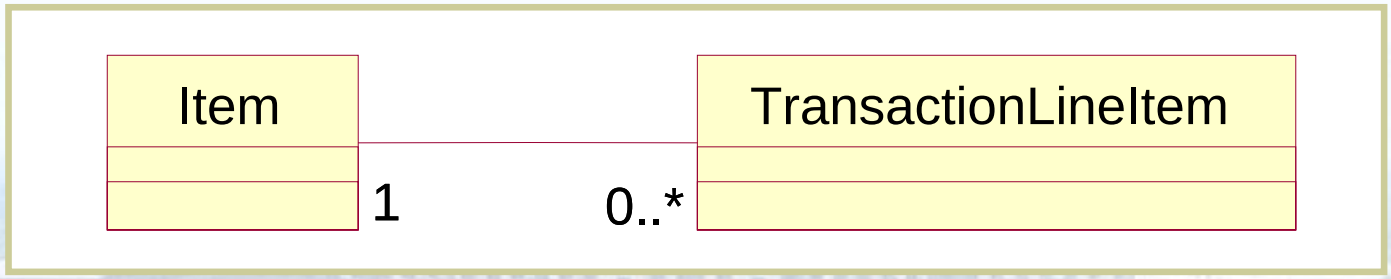
- 可采用的范例：缴税与税单；代付电话费与话费明细；购物与发票；领料与出库单；核实顾客信誉与近六个月付款明细；……

4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

6. “科目—交易项目”模式:

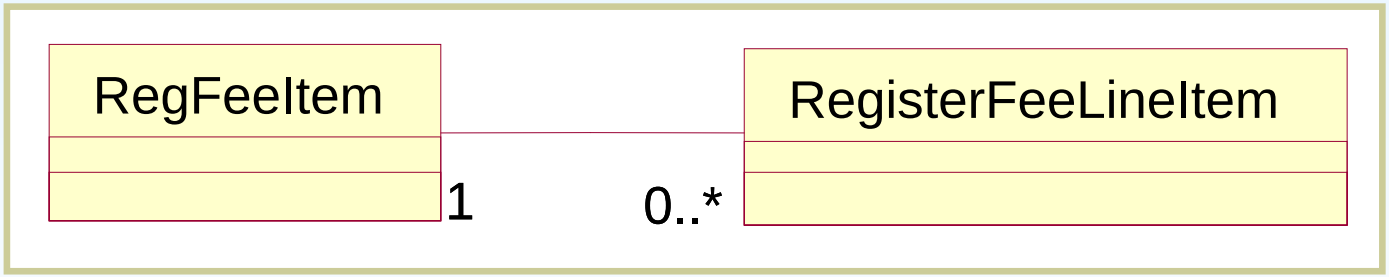
- 定义规定科目（或物品）和与它有关的多次交易之间的关系，例如商品与发票之间的关系。这种模式既可以避免交易项目中关于科目信息的冗余，又利于从特定科目得出所需的分类交易信息。



4.13 面向对象系统设计

4.13.12 分析模式简介－几种最简单的分析模式

“科目—交易项目”模式的例子－收费科目与分项缴费：

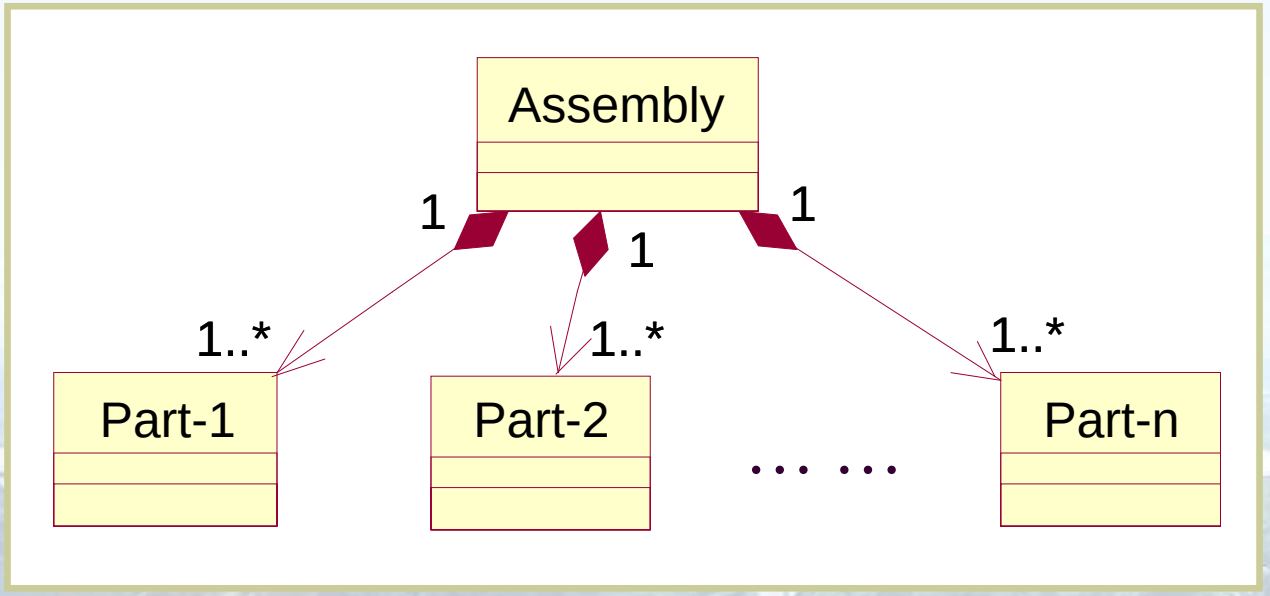


- 可采用的范例：险种与保单；税种与税单；电话费科目与话费分项清单；商品与发票交易明细；……
- 不宜采用的范例：汽车零件与整车销售发票；……

4.13 面向对象系统设计

4.13.12 分析模式简介—常见的聚集

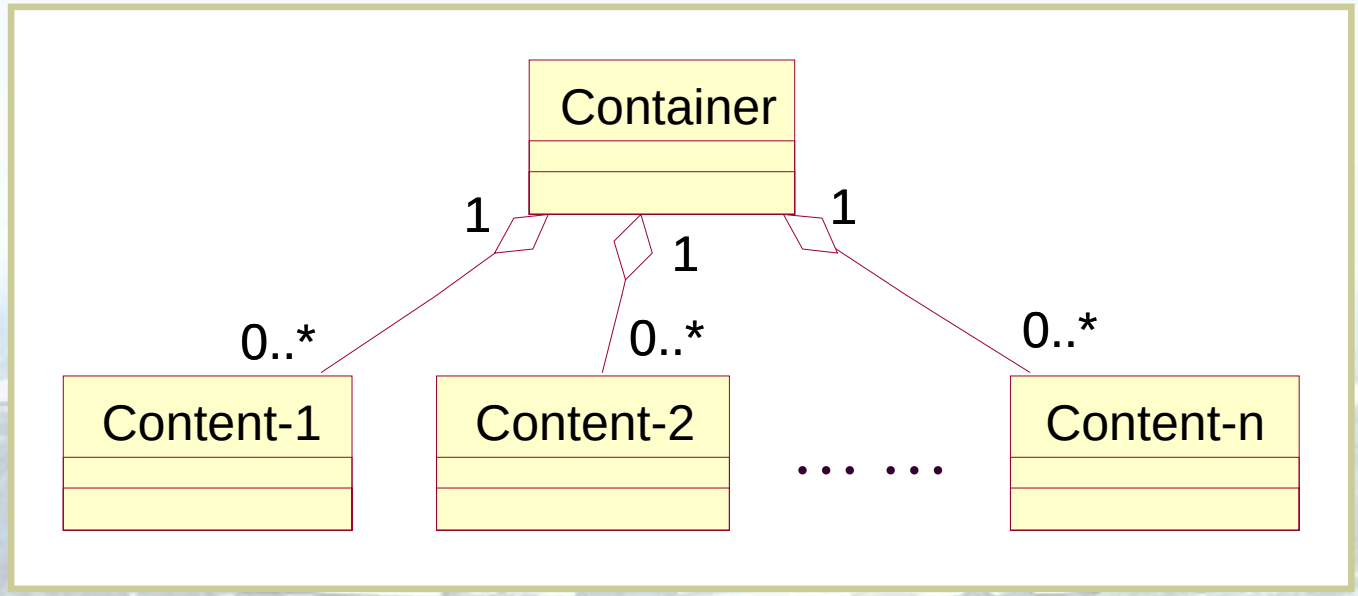
- “装置与零件”（*assembly and parts*）：
  - 经常用于表示那些制造出来的产品以及组成它的零件，例如个人计算机由电源、主板、磁盘驱动器、螺钉等组成；



4.13 面向对象系统设计

4.13.12 分析模式简介－常见的聚集

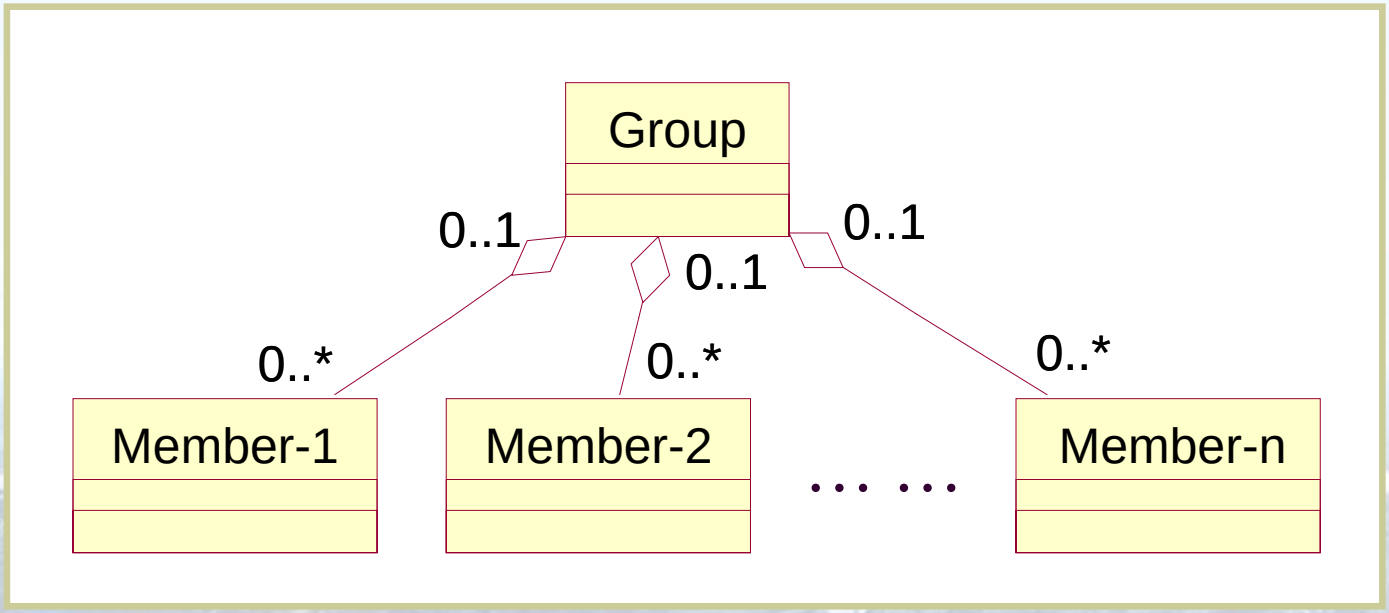
- “容器与内容”（*container and contents*）：
  - 没有明显的组装特征时可考虑这一种。例如，飞机的驾驶仓中有计量仪表、刻度盘、操纵杆、指示灯等，办公室里有办公桌、电话、书架、文件柜等。



4.13 面向对象系统设计

4.13.12 分析模式简介—常见的聚集

- “团体与成员”（*group and members*）：
  - 存在变动关系和（或）需要记录历史。例如，学会与会员；学校与学生等。



第三次作业（7 天后提交，10天后截止）

- 根据你对 **Template method** 模式的理解，设想并概要地描述两种可使用该模式的不同应用要求，再给出相应的设计类图。

要点与引伸

- 责任涉及:
  - **What “I” know?** – 每一个对象知道关于它自己的事情;
  - **Who “I” know?** – 每一个对象知道有关的其他对象;
  - **What “I” do?** – 每一个对象知道它必须履行的功能。
- 确定 **Expert**、**Creator** 和 **Controller**，分别是在确定:系统中既定信息的提供者、有利于降低耦合度的对象生成机制和处置系统事件的负责者。
- 协同图是确定责任的主要工具，然后再用类图具体落实到类。

要点与引伸（续）

- 对象视域的确定，实际上是在以不同的方式进行对象组织结构的确定。
- 生成模式给出了对象生成机制的典型范例。
- 结构模式给出了复杂的对象组织结构的典型范例。
- 行为模式给出了操作定位的典型范例。
- 这里给出的各种模式，都是在实际开发中经历了考验、能够充分发挥面向对象机制作用的成功经验。我们可以看出，在这些模式中经常使用着抽象类、重置等机制，其用意在于规范接口、适应变化和扩充。
- 课堂上没有讲的其他一些模式在后面给出，大家自己看。

下一次课的内容

- 对象技术的发展过程、现状与发展趋势
- 课程总结