上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

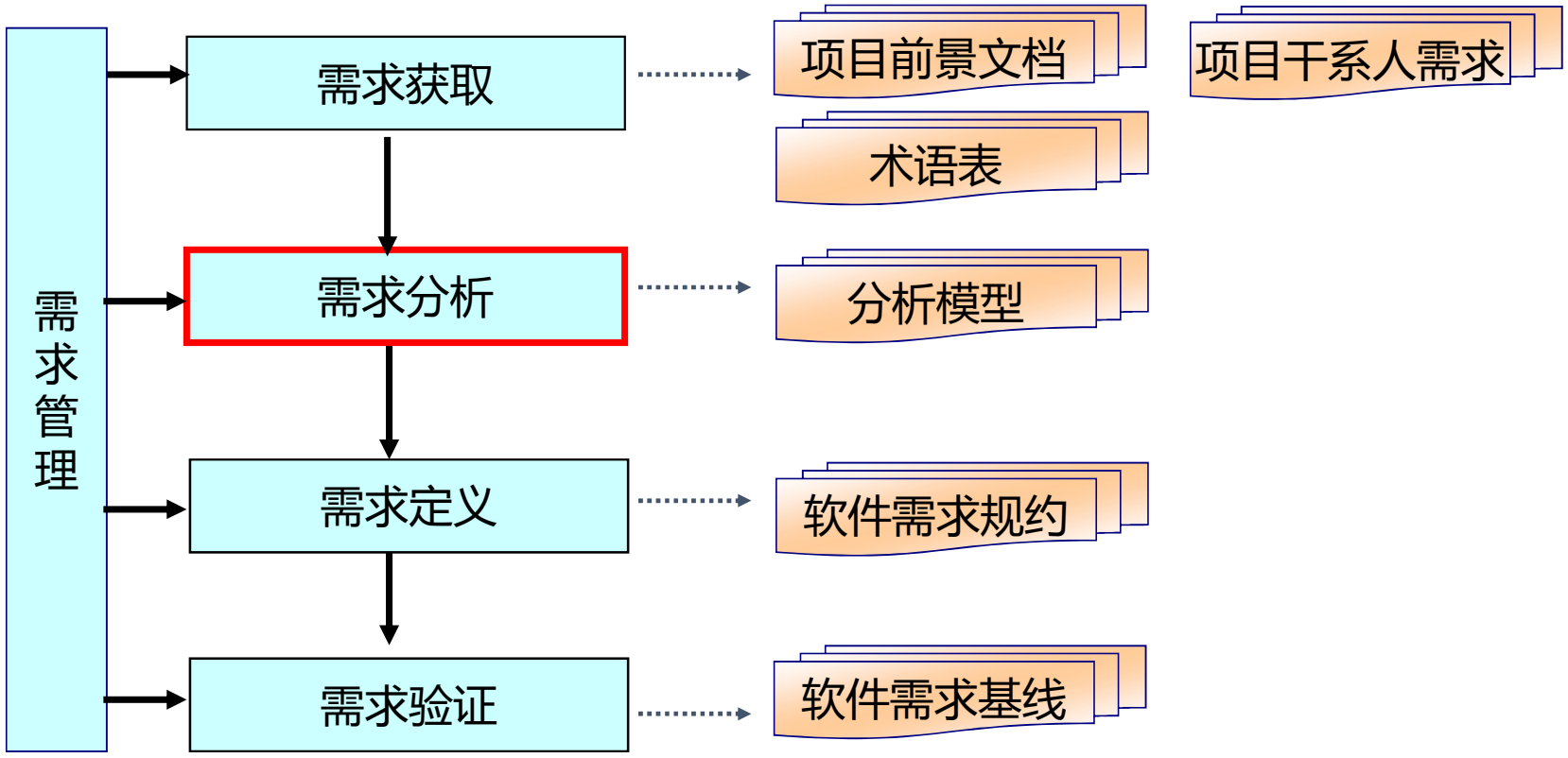
软件工程原理与实践
SOFTWARE ENGINEERING

面向对象分析

沈备军

饮水思源 · 爱国荣校

需求工程



发现、获取、组织、 分析、编写和管理需求的系统方法，让客户和项目组之间达成共识。

构建分析模型

- 需求工程中的重要环节
- 分析模型是平台无关模型
- 关注 What, Not How
- 分析建模方法
  - 面向对象分析
  - 结构化分析
  - ……

大 纲

01-面向对象方法概述

02-面向对象的基本概念

03-用例建模

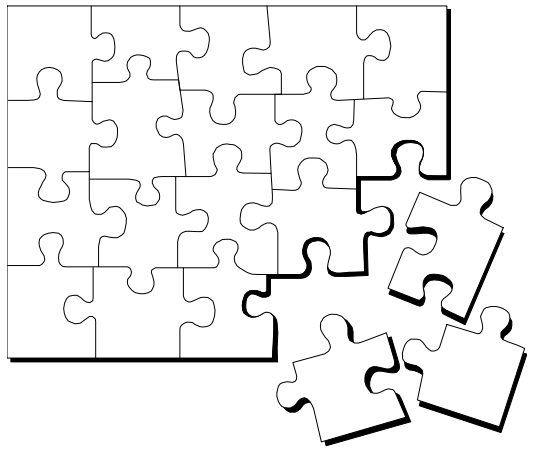
04-建立概念模型

05-用例分析

@第3.3节,第5章.教材

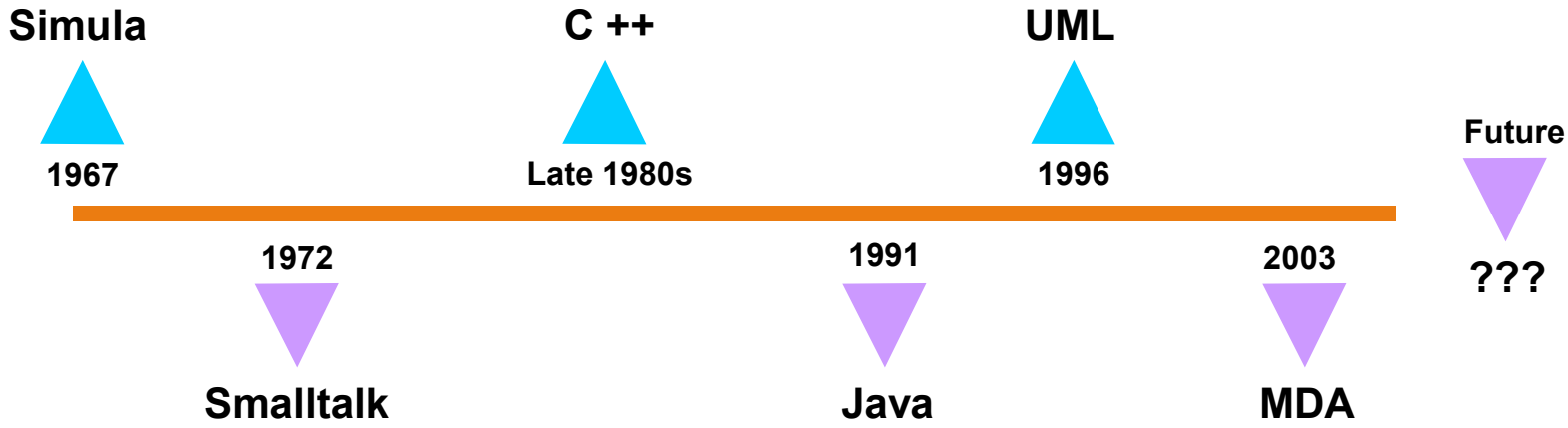
Object Technology

- What Is Object Technology?
  - A set of principles (abstraction, encapsulation, polymorphism) guiding software construction, together with languages, databases, and other tools that support those principles.
- The Strengths of Object Technology
  - Reflects a single paradigm
  - Facilitates architectural and code reuse
  - Reflects real world models more closely
  - Encourages stability
  - Is adaptive to change

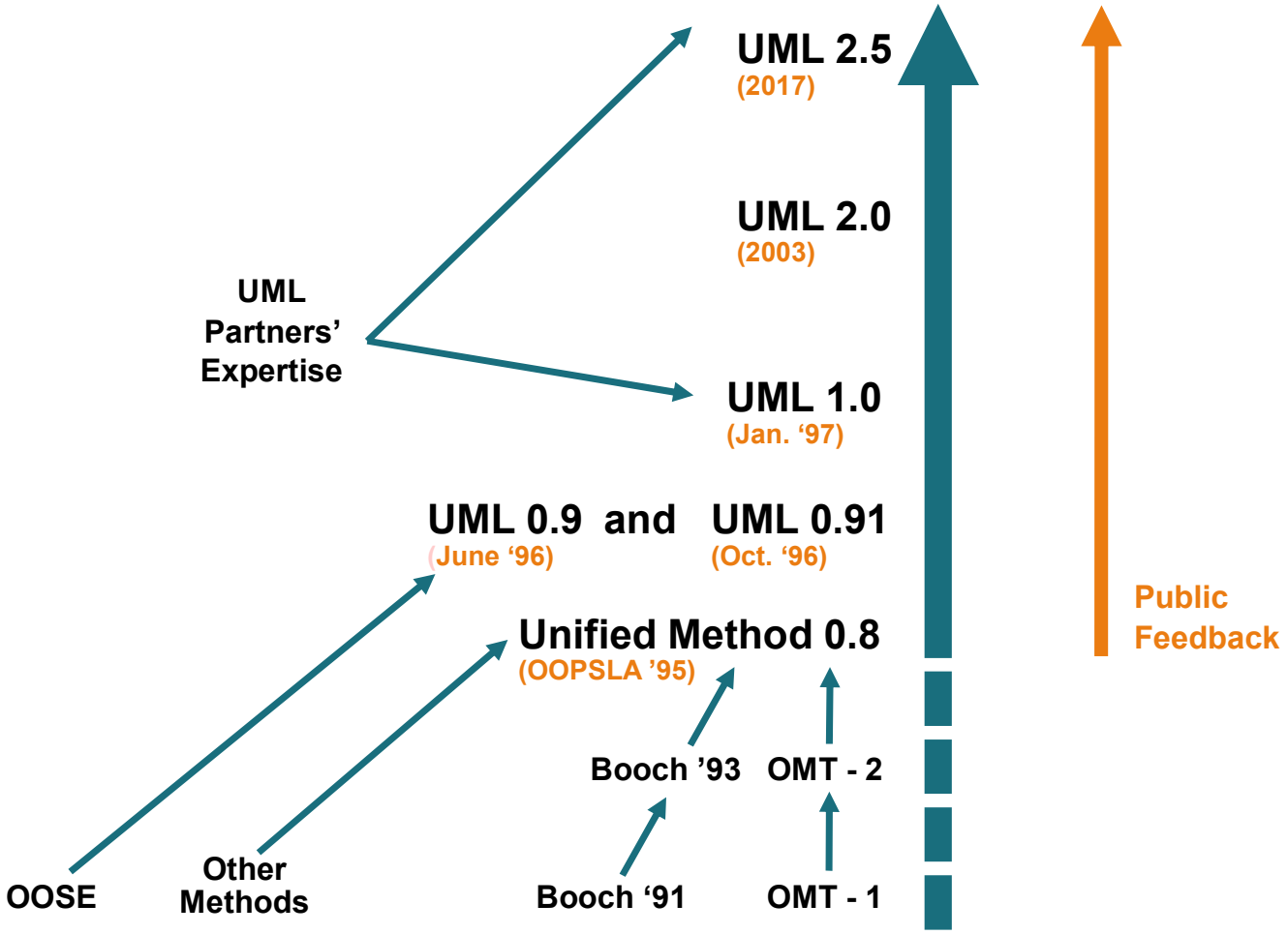


The History of Object Technology

- Major object technology milestones

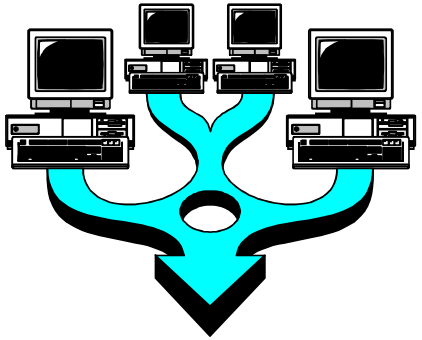


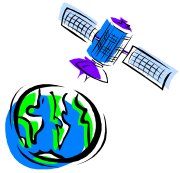
UML (Unified Modeling Language)



Where Is Object Technology Used?

- Client/Server Systems and Web Development
- Real-time Systems
- Embedded System
- Multimedia System
- Middleware
- ……





面向对象方法的四个步骤

- 面向对象分析
  - Object Oriented Analysis, OOA
- 面向对象设计
  - Object Oriented Design, OOD
- 面向对象编程
  - Object Oriented Programming, OOP

大 纲

01-面向对象方法概述

02-面向对象的基本概念

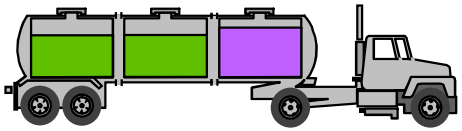
03-用例建模

04-建立概念模型

05-用例分析

What Is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software.
  - Physical entity

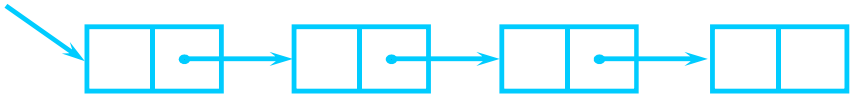


Truck

  - Conceptual entity

Chemical Process

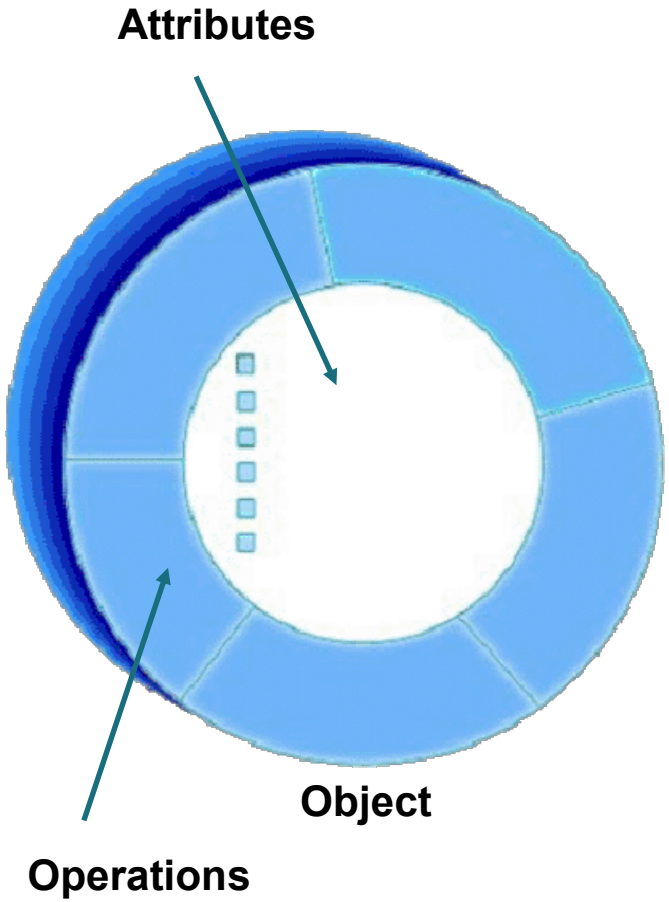
  - Software entity



Linked List

A More Formal Definition

- An object is an entity with a well-defined boundary and identity that encapsulates state and behavior.
  - State is represented by attributes and relationships.
  - Behavior is represented by operations, methods, and state machines.



Representing Objects in the UML

- An object is represented as a rectangle with an underlined name.

Professor J Clark

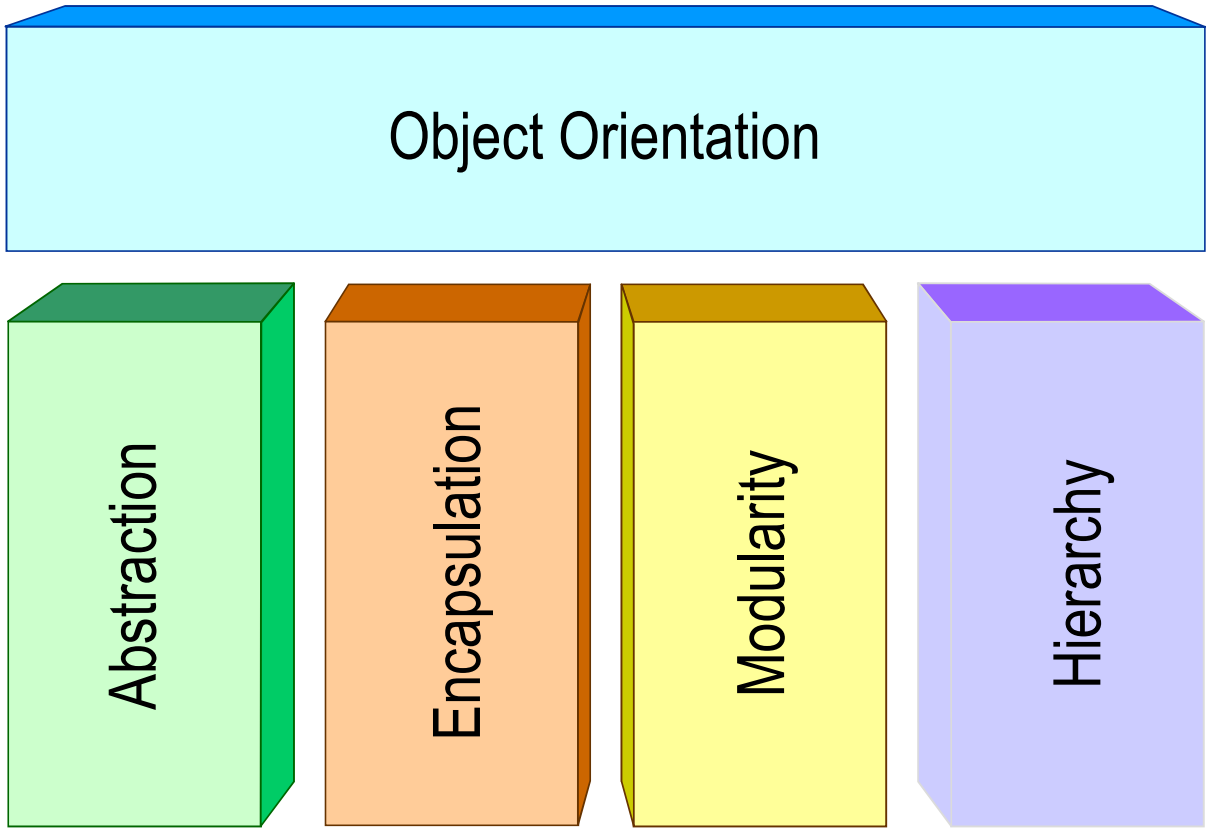
J Clark : Professor

Named Object

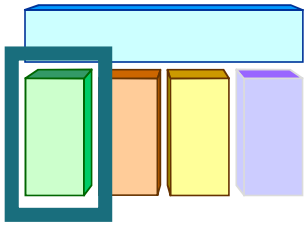
: Professor

Anonymous Object

Basic Principles of Object Orientation

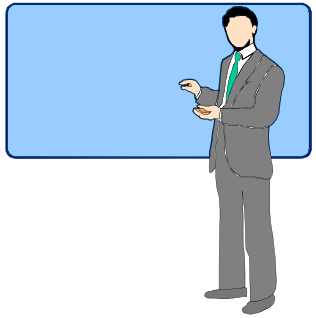


What Is Abstraction?

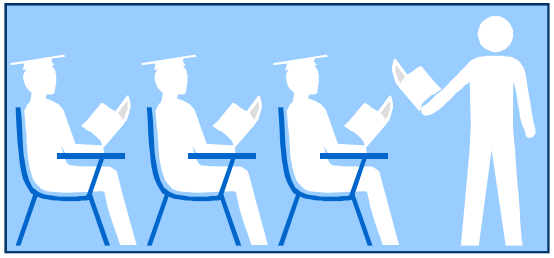


- The essential characteristics of an entity that distinguishes it from all other kinds of entities.
- Defines a boundary relative to the perspective of the viewer.
- Is not a concrete manifestation, denotes the ideal essence of something.

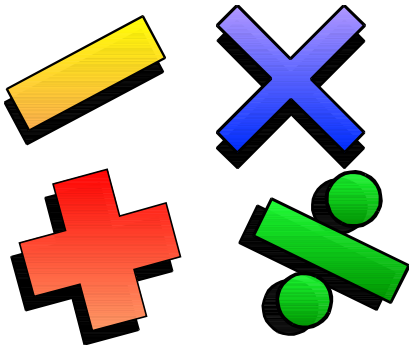
Student



Professor

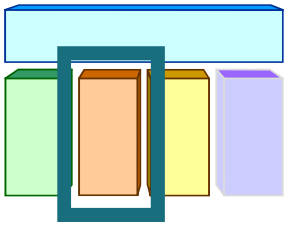


Course Offering (9:00 a.m., Monday-Wednesday-Friday)

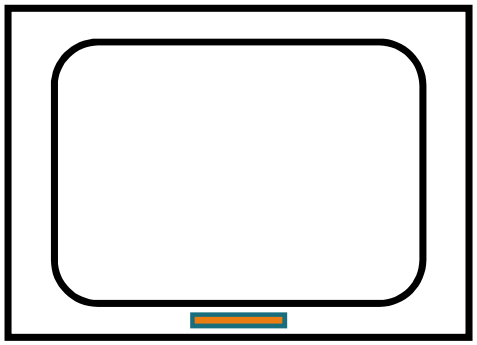


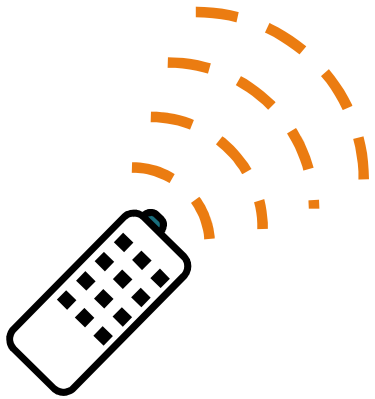
Course (e.g. Algebra)

What Is Encapsulation?



- Hides implementation from clients.
  - Clients depend on interface.

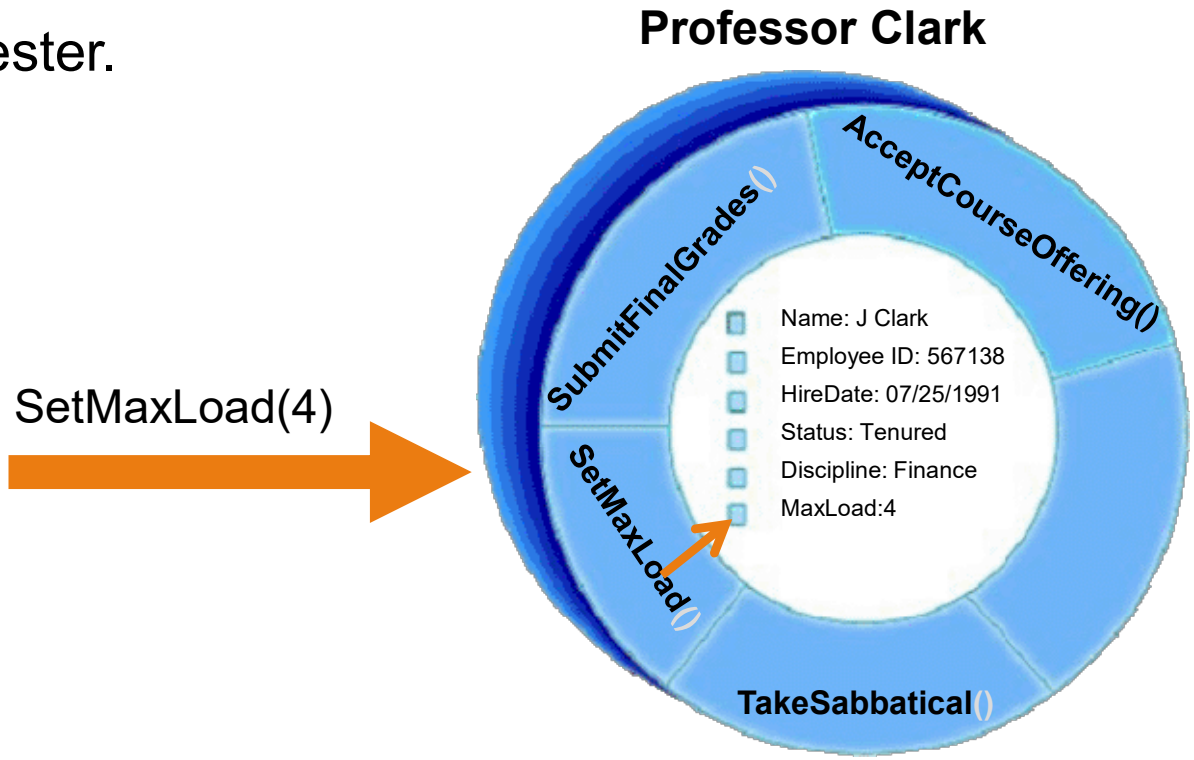




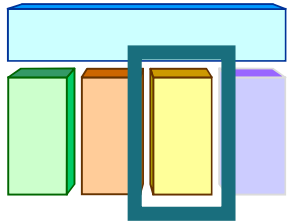
Improves Resiliency

Encapsulation Illustrated

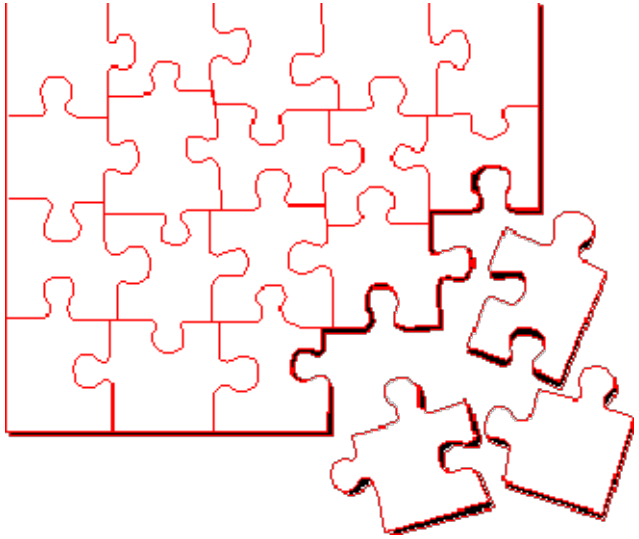
- Professor Clark needs to be able to teach four classes in the next semester.



What Is Modularity?

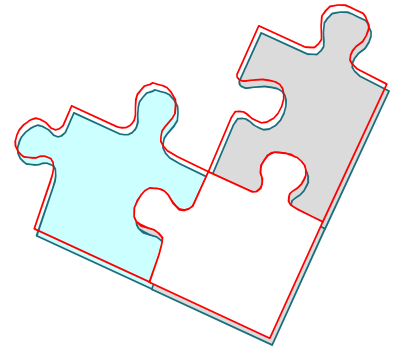


- Breaks up something complex into manageable pieces.
- Helps people understand complex systems.

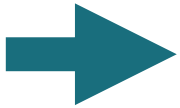


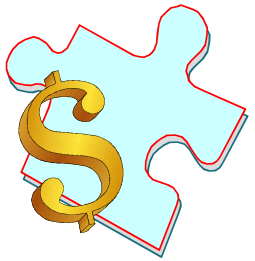
Example: Modularity

- For example, break complex systems into smaller modules.

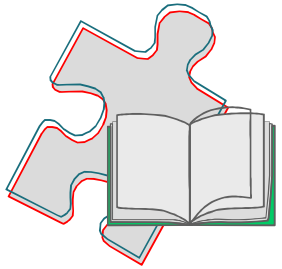


Course Registration System

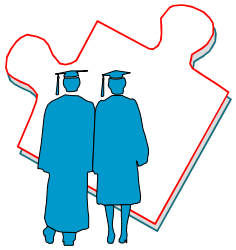




Billing System

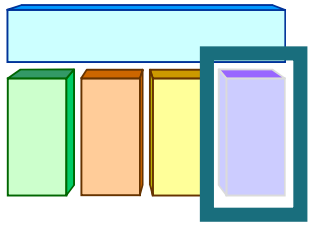


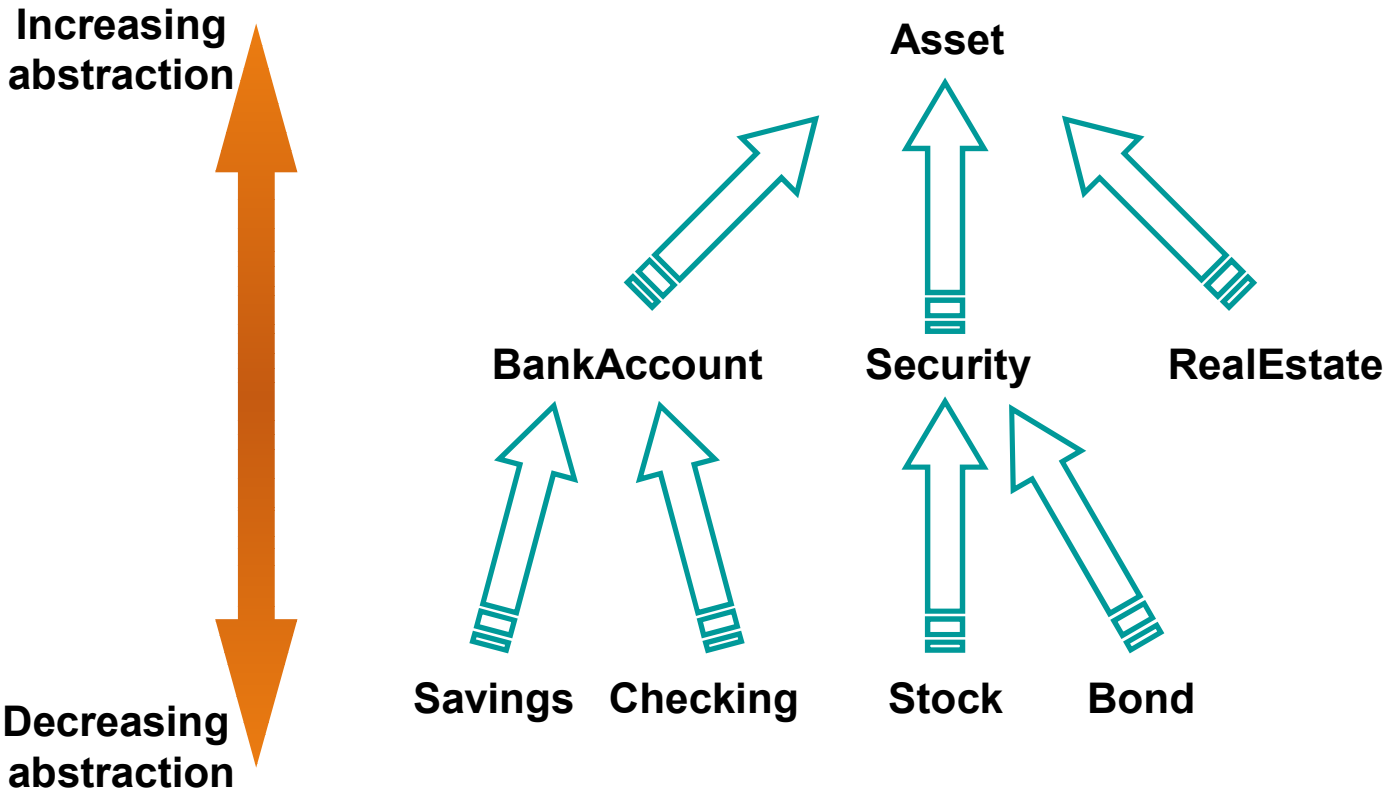
Course Catalog System



Student Management System

What Is Hierarchy?





Elements at the same level of the hierarchy should be at the same level of abstraction.

What Is a Class?

- A class is a description of a set of objects that share the same *attributes*, *operations*, *relationships*, and *semantics*.
  - An object is an instance of a class.
- A class is an abstraction in that it
  - Emphasizes relevant characteristics.
  - Suppresses other characteristics.

Representing Classes in the UML

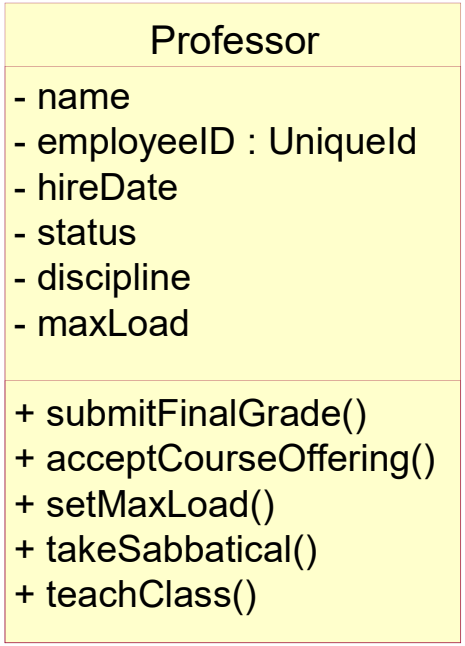
- A class is represented using a rectangle with three compartments:
  - The class name
  - The structure (attributes)
  - The behavior (operations)

Visibility:

Public: +

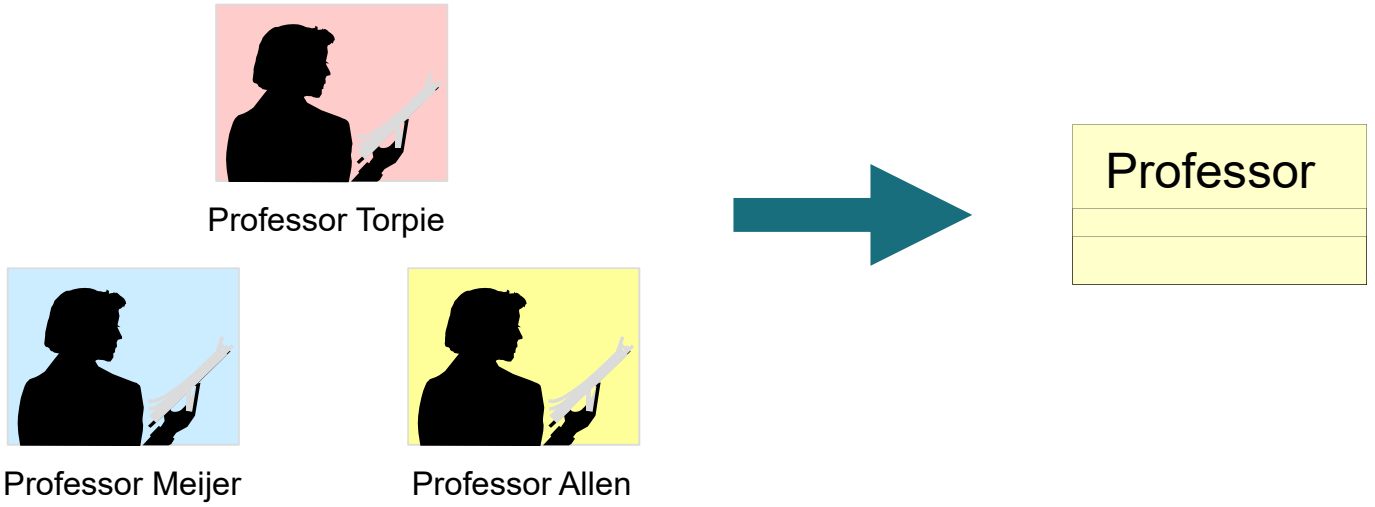
Private: -

Protected: #



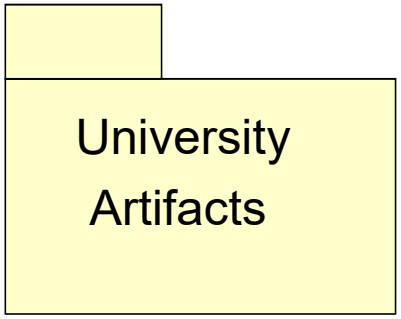
The Relationship between Classes and Objects

- A class is an abstract definition of an object.
  - It defines the structure and behavior of each object in the class.
  - It serves as a template for creating objects.
- Classes are not collections of objects.



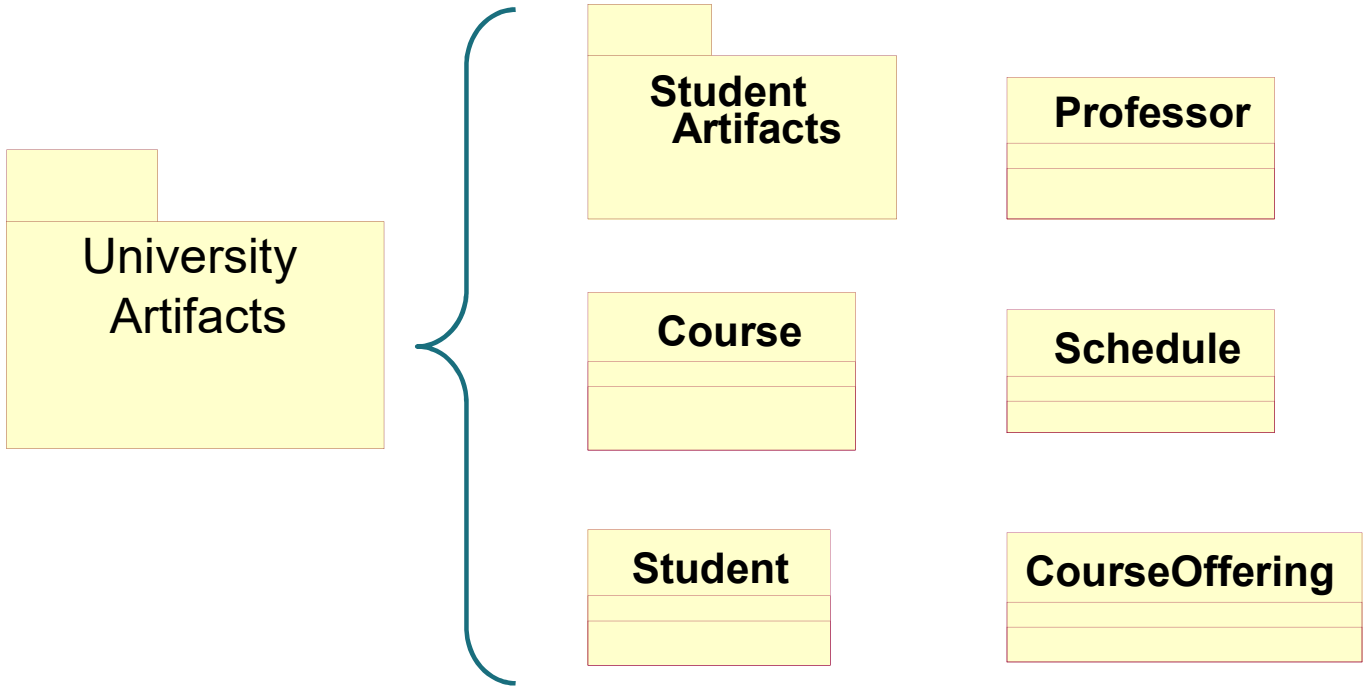
What Is a Package?

- A general purpose mechanism for organizing elements into groups.
- A model element that can contain other model elements.
- A package can be used:
  - To organize the model under development.
  - As a unit of configuration management.



Package 示例

- The package, University Artifacts, contains one package and five classes.



面向对象分析的步骤

1. 用例建模

2. 建立概念模型

3. 用例分析

大 纲

01-面向对象方法概述

02-面向对象的基本概念

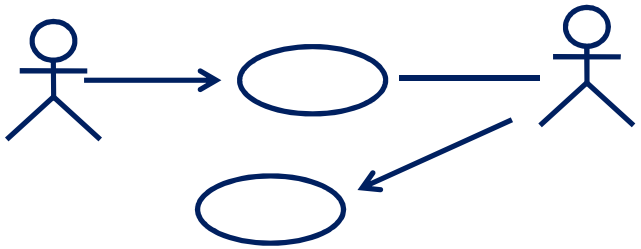
03-用例建模

04-建立概念模型

05-用例分析

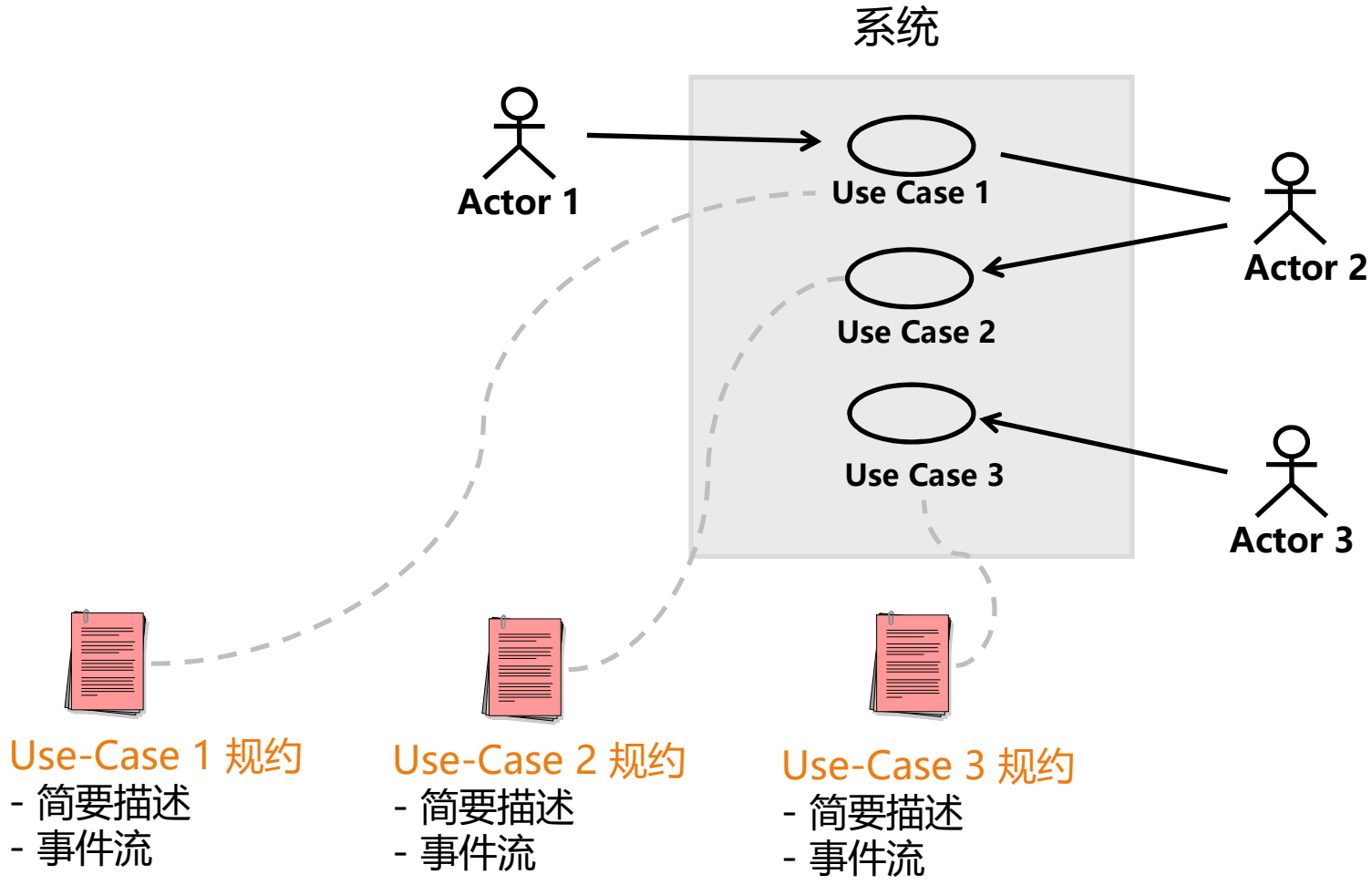
Use Case技术

- 提供涉众的观点
- 定义功能需求
- 促进理解和讨论
  - 为什么需要系统?
  - 谁和系统交互 (actors)?
  - 用户希望如何使用系统 (use cases)?
  - 系统应该有什么接口?



Use-Case 模型

Use-Case 模型的组成



面向对象分析的步骤

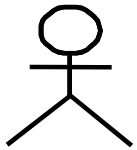
1. 用例建模

- **1.1识别actor和use case，画Use-Case图**
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

3. 用例分析

Actor 和 Use Case

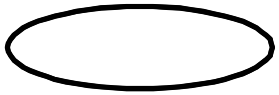


Actor

Actor

和系统交互的系统外的某些人或某些东西：

- 最终用户
- 外界软件系统
- 外界硬件设备



Use Case

Use case

Actor想使用系统去做的事

如何识别Actor

- 谁需要在系统的帮助下完成自己的任务?
- 需要谁去执行系统的核心功能?
- 需要谁去完成系统的管理和维护?
- 系统是否需要和外界的硬件或软件系统进行交互?

如何识别Use Case

- 每个actor的目标和需求是什么?
  - actor希望系统提供什么功能?
  - actor 将创建、存取、修改和删除数据吗?
  - actor是否要告诉系统外界的事件?
  - actor 需要被告知系统中的发生事件吗?

避免功能分解

现象

- use case太小
- use cases太多
- Uses case没有价值回报
- 命名以低层次的操作
  - “Operation” + “object”
  - “Function” + “data”
  - Example: “Insert Card”
- 很难理解整个模型

纠正措施

- 寻找更大的上下文
  - “为什么你要构建本系统?”
- 让自己站在用户的角度

“用户想得到什么”

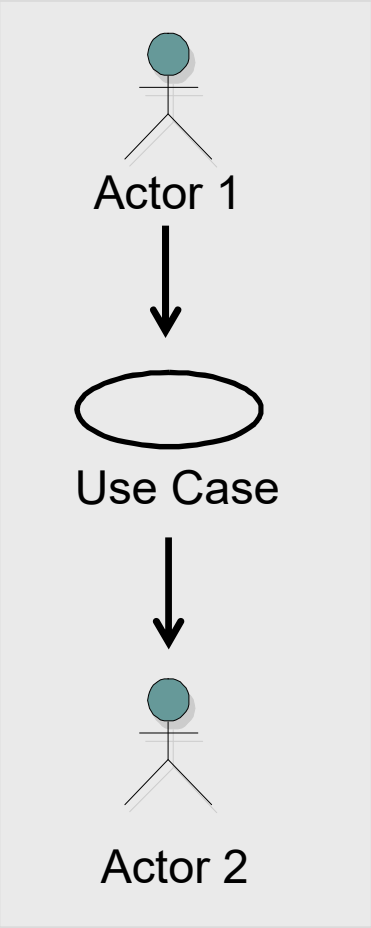
“这个 use case 能满足谁的要求?”

“这个use case 能增值什么?”

“这个use case 背后有什么故事?”

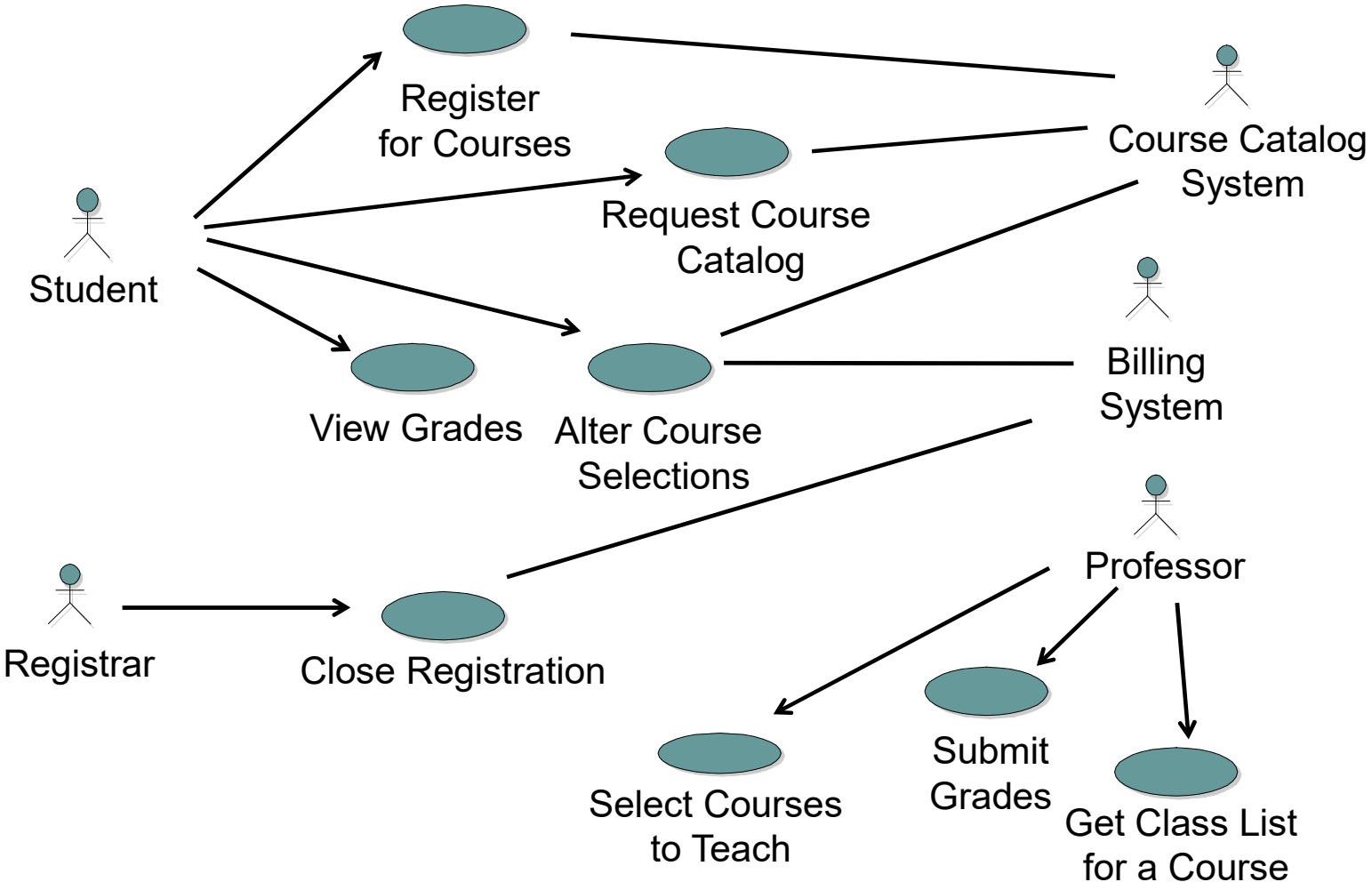
一个用例定义了和actor之间的一次完整对话。

通信-关联 Communicates-Association



- Actor和use case 间的通信渠道
- 用一条线表示
- 箭头表示谁启动通信

Course Registration System的用例图



面向对象分析的步骤

1. 用例建模

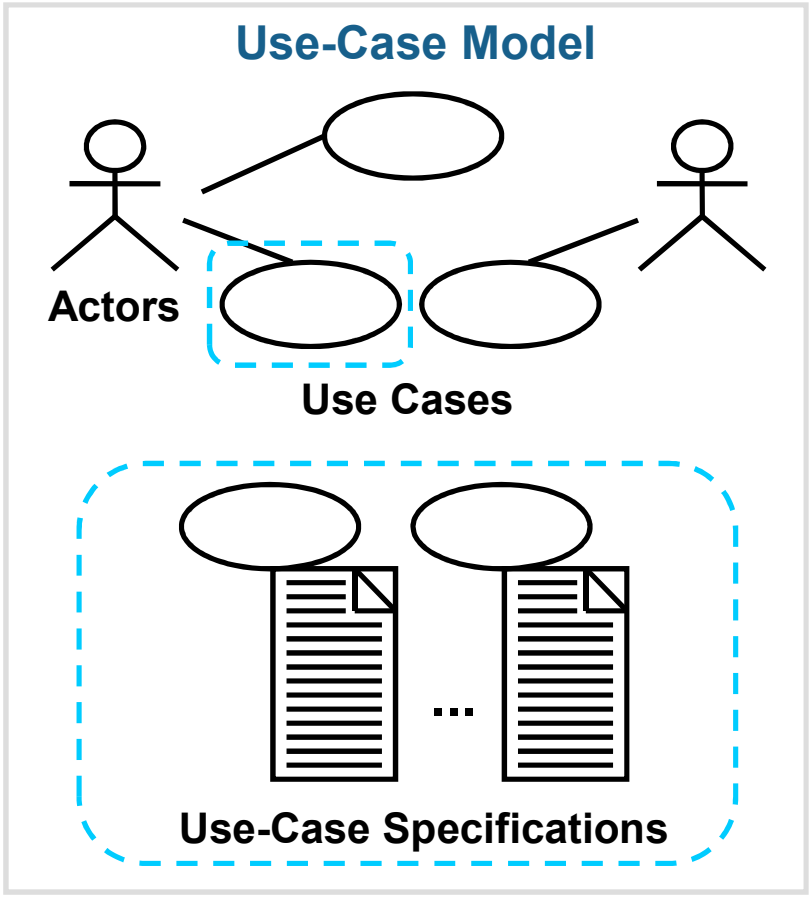
- 1.1识别actor和use case，画Use-Case图
- **1.2 编写Use-Case Spec.**
- 1.3 优化Use-Case图的结构

2. 建立概念模型

3. 用例分析

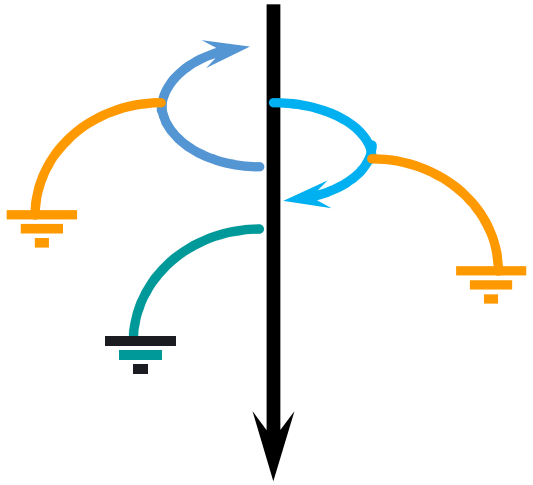
Use-Case Specifications

- Name
- Brief description
- Flow of Events
- Relationships
- Activity diagrams
- Use-Case diagrams
- Special requirements
- Pre-conditions
- Post-conditions
- Other diagrams



事件流（基本流和备选流）

- 一个基本流
  - Happy day scenario
  - Successful scenario from start to finish
- 多个备选流
  - Regular variant
  - Odd cases
  - Exceptional (error) flows



事件流举例: Get Quote

基本流 Basic Flow

1. Customer logs on
2. Customer selects ‘Get Quote’ function
3. Customer selects stock trading symbol
4. Get desired quote from Quote System
5. Display quote
6. Customer gets other quotes
7. Customer logs off

备选流 Alternative Flows

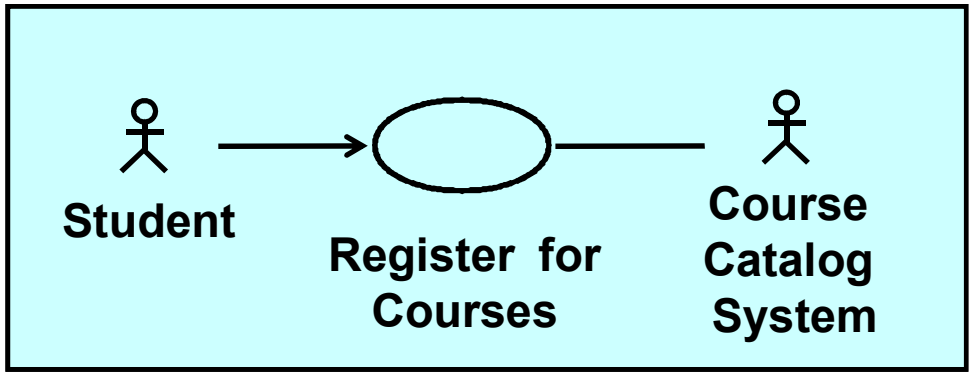
A1. Unidentified Trading Customer

A2. Quote System Unavailable

A3. Quit

What are other alternatives?

Scenario 是Use-Case 的实例



Scenario 1
Log on to system
Approve log on
Enter subject in search
Get course list
Display course list
Select courses
Confirm availability
Display final schedule

Scenario 2
Log on to system
Approve log on
Enter subject in search
Invalid subject
Re-enter subject
Get course list
Display course list
Select courses
Confirm availability
Display final schedule

细化基本事件流

Get Quote

Basic Flow

1. Customer Logs On

The use case starts when the Trading Customer logs on. The system validates the customer id and password. …

2. Customer Selects "Get Quote" Function

The Trading Customer chooses to "Get Quote." The system displays the list of securities on which it has quotes.

3. Customer Gets Quote

The Trading Customer selects from the list of securities or enters the trading symbol for a security. The system sends the trading symbol to the Quote System, and receives the Quote System Response.

4. Customer Gets Other Quotes

If the Trading Customer wishes to get additional quotes, the use case resumes at Step 3.

5. Customer Logs Off

The Trading Customer logs off the system. The use case ends.

将流分成几步

每步标识编号和名字

描述每步 **(1-3 句)**

每步是一个来回的事件

细化通用备选流

Identify Customer

Alternative Flows

A1. Unidentified Trading Customer

In Step 1 of the Basic Flow, if the system determines that the customer password is not valid, the system displays a “Sorry, not a valid customer….” message. The use case ends.

A2. Wrong Password

In Step 1 of the Basic Flow, if the system determines that the customer password id is not valid, the system displays a “Sorry, not….”

A3. Suspect Theft

In step 1 of the Basic Flow, if the customer id is on the system’s list of stolen identification, the system displays a “Sorry, not …” message. The system also records the date, time. And computer address…

A4. Quit

The RU e-st System allows the Trading Customer to quit at any time during the use case. The use case ends.

A5. No Reply From User

At any time during the use case, if the system asks for input from the Trading Customer….

在任何步骤都可能发生

细化特殊备选流

Get Quote

Alternative Flows

A1 Unidentified Trading Customer

In Step 1, Customer Logs On, in the Basic flow, if the system determines that the customer id and/or password are not valid,…

A2 Quote System Unavailable

In Step 3, Customer Gets Quote, in the basic flow, if the system is unable to communicate with the Quote System, the system…

A3 Quit

The RU e-st System allows the Trading Customer to quit at any time during the use case. The use case ends.

A4 Unknown Trading Symbol

In Step 3, Customer Gets Quote, in the Basic Flow, if the system cannot recognize the trading symbol the system notifies the Trading Customer that the trading symbol is not recognizable. The use case continues at the beginning of Step 3...

A5 Quote System Cannot Locate Information

In Step 3, Customer Gets Quote, in the Basic Flow, if the Quote System responds that it does not have the requested information…

描述发生了什么

开始

条件

动作

继续

举例：备选流的另一种编号方法

基本流 Basic Flow

1. 采购员在初始申购单中添加申购子项。
2. 采购员输入供应商、交货地点、最终价格、返利、加价和费用明细。
3. 采购员增加、删除和修改申购单（包括第1、2步）直到满意为止。
4. 采购员在输入所有必要信息后，保存并完成申购单。
5. 系统检验申购单，分配申购单号，设置申购单和申购单子项状态为“未提交”
6. 采购员提交申购单。
7. 系统设置申购单状态为“待审批”，通知采购经理审批申购单。

备选流 Alternative Flows

1-3a 退出：

1) 系统提示用户保存。

2) 用户选择不保存，系统放弃临时信息，申购单状态不变。

4a. 申购单信息不完整：

系统提示不能保存，回到步骤3。

What Is an Activity Diagram?

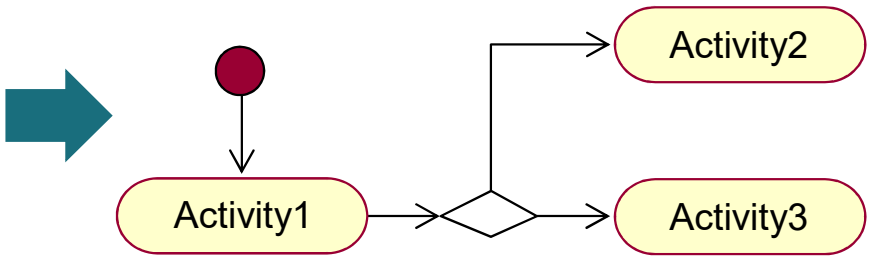
- An activity diagram in the Use-Case Model can be used to capture the activities in a use case.
- It is essentially a flow chart, showing flow of control from activity to activity.

Flow of Events

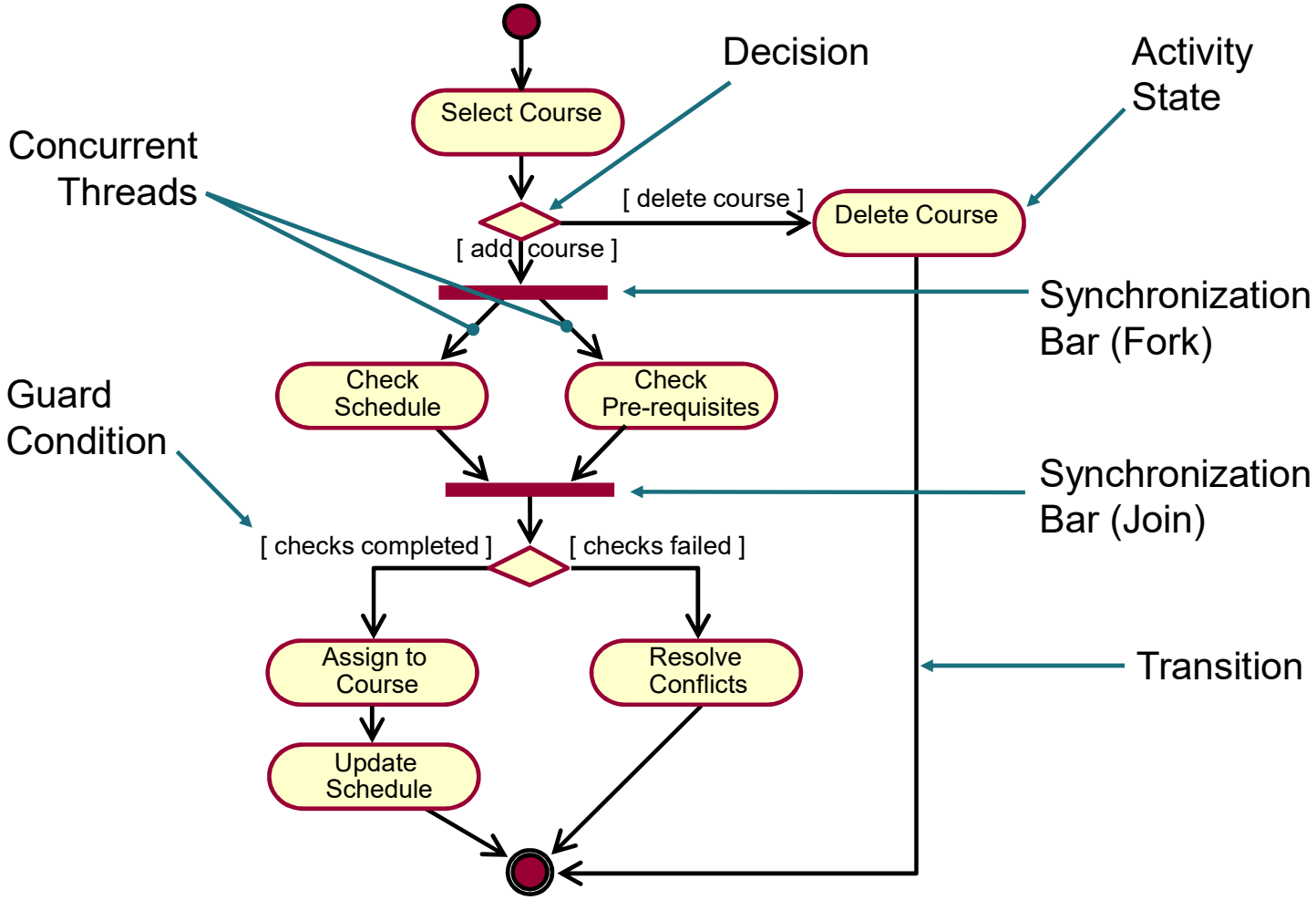
This use case starts when the Registrar requests that the system close registration.

1. The system checks to see if registration is in progress. If it is, then a message is displayed to the Registrar and the use case terminates. The Close Registration processing cannot be performed if registration is in progress.

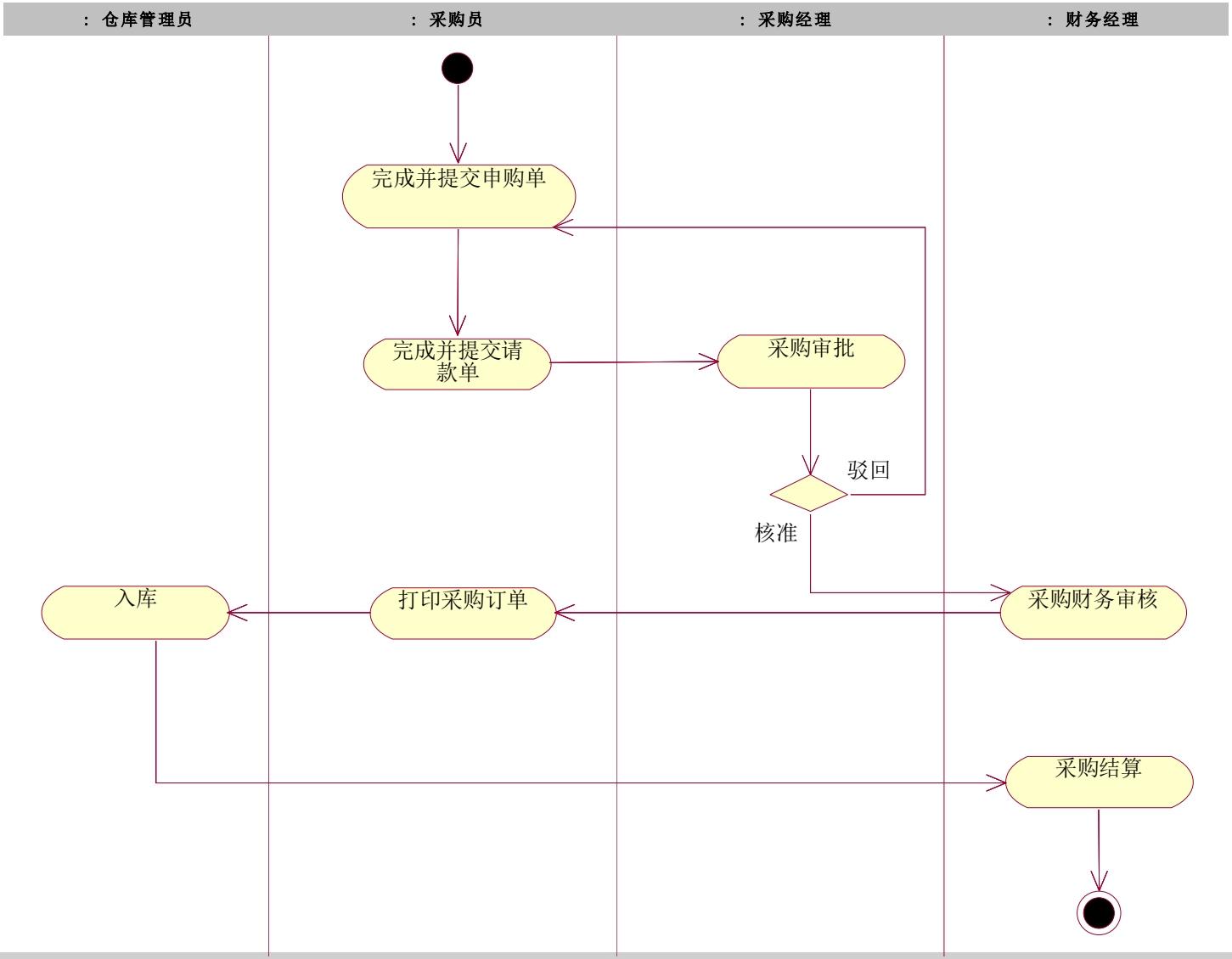
2. For each course offering, the system checks if a professor has signed up to teach the course offering and at least three students have registered. If so, the system commits the course offering for each schedule that contains it.



Example: Activity Diagram



Partitions


：仓库管理员
：采购员
：采购经理
：财务经理
完成并提交申购单
完成并提交请款单
采购审批
驳回
核准
入库
打印采购订单
采购财务审核
采购结算

前置条件

- Use-case 启动的约束条件
- 不是触发Use-case 的事件
- 可选的: 仅当需要时才用

示例:

- 必须满足下面的前置条件，ATM 系统才能出钞:
  - 必须能进入 ATM 网络。
  - ATM 必须处于准备接受交易的状态。
  - ATM 中必须备有一些现金可供出钞。
  - ATM 必须备有足够的纸张打印至少一次交易的收据

后置条件

- 当use case结束时,后置条件一定为真
- 用于降低用例事件流的复杂性并提高其可读性
- 还可用来陈述在用例结束时系统执行的动作
- 可选的: 仅当需要时才用

示例:

- ATM 在用例结束时总是显示“欢迎使用”消息，可将此消息记录在用例后置条件中。
- 与此类似，如果 ATM 在如提取现金这样的用例结束时总会停止客户交易，则不管事件进程如何，将这种情况记录为用例后置条件。

其它Use Case属性

- 非功能需求
  - 有关本use-case的URPS+
- 业务规则
  - 在执行事件流时，使用到的重要业务规则或计算公式
- 扩展点
  - use-case可以通过另一use-case进行扩展
- 关系
  - 和actors及use-case的关联
- Use-case 图
  - 涉及本use-case的关系的可视化模型
- Other diagrams or enclosures
  - 交互、活动或其它图
  - 用户界面框图

面向对象分析的步骤

1. 用例建模

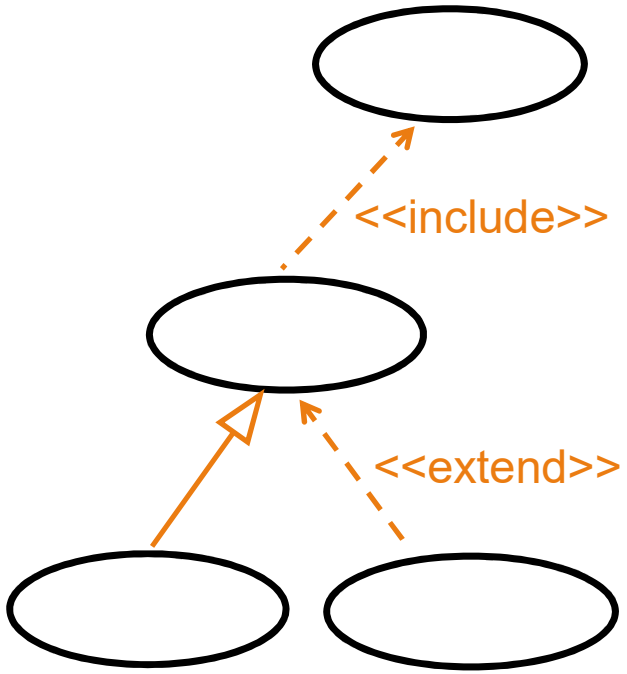
- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- **1.3 优化Use-Case图的结构**

2. 建立概念模型

3. 用例分析

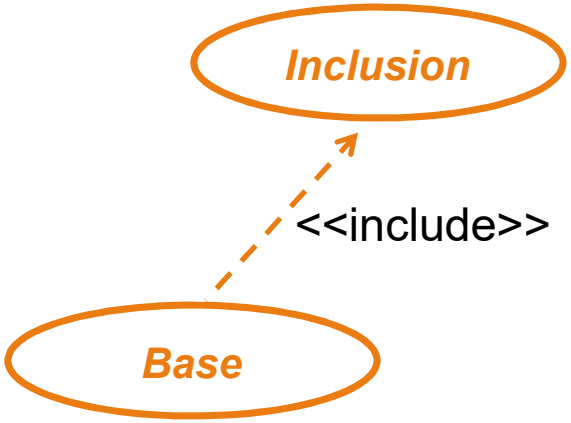
Use Case间的关系

- Include 包含
- Extend 扩展
- Generalization 泛化

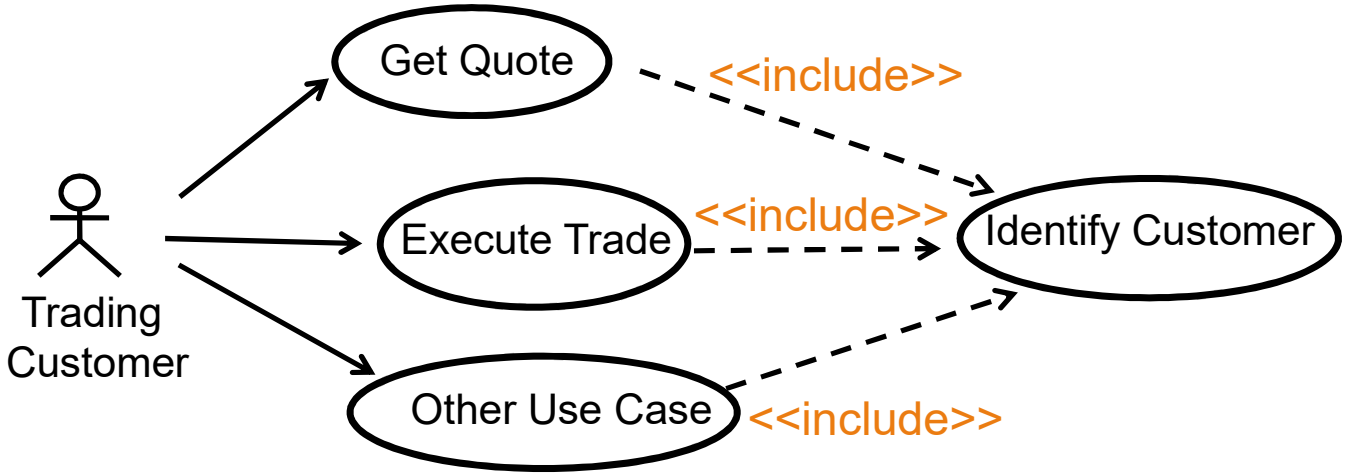


什么是Include 关系?

- 基本用例包含某个包含用例
- 包含用例定义的行为将显式插入基本用例



Include 关系示例

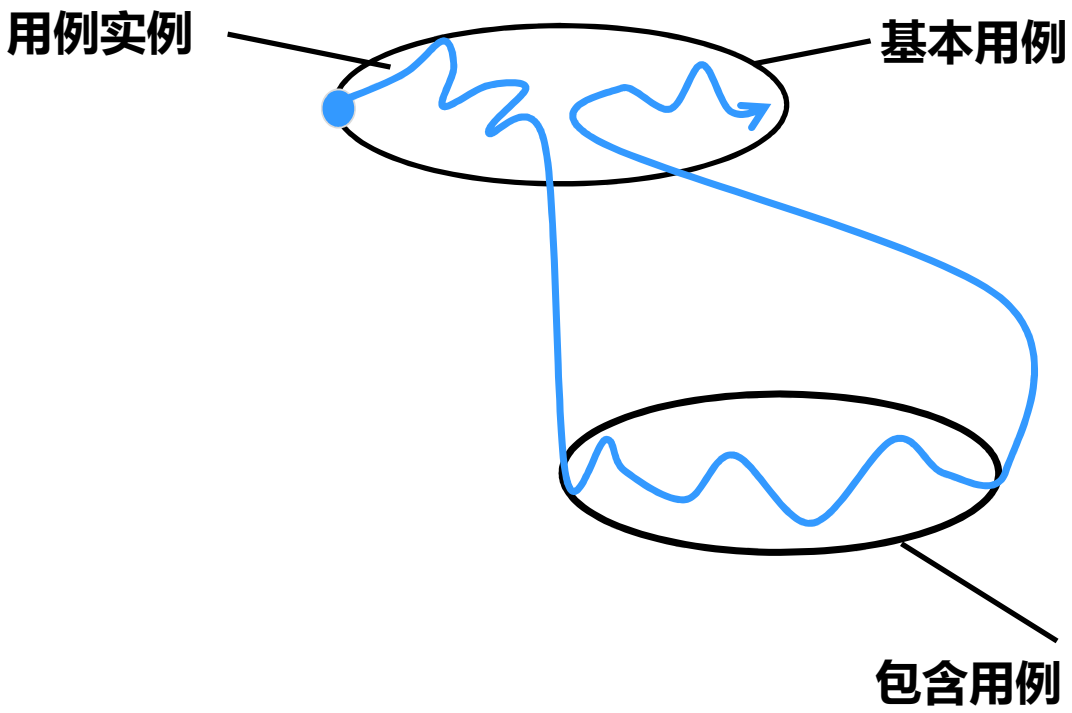


Get Quote Use Case
1. Include “Identify Customer” to verify customer’s identity
2. Display options. Customer selects “Get Quote”
3. ...

Identify Customer Use Case
1. Log on
2. Validate logon
3. Enter password
4. Verify password
A1: Not valid logon
A2: Wrong password
A3: ...

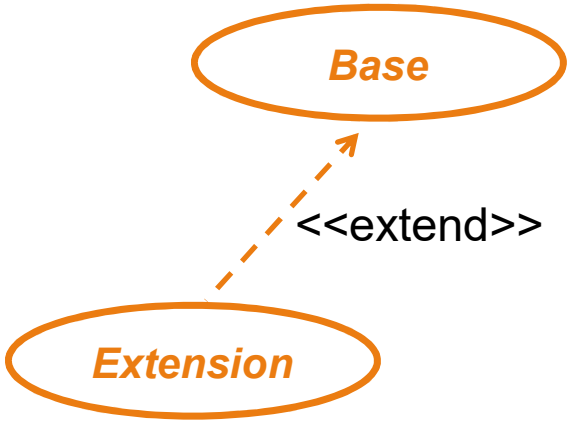
执行Include

- 到达插入点时执行包含用例

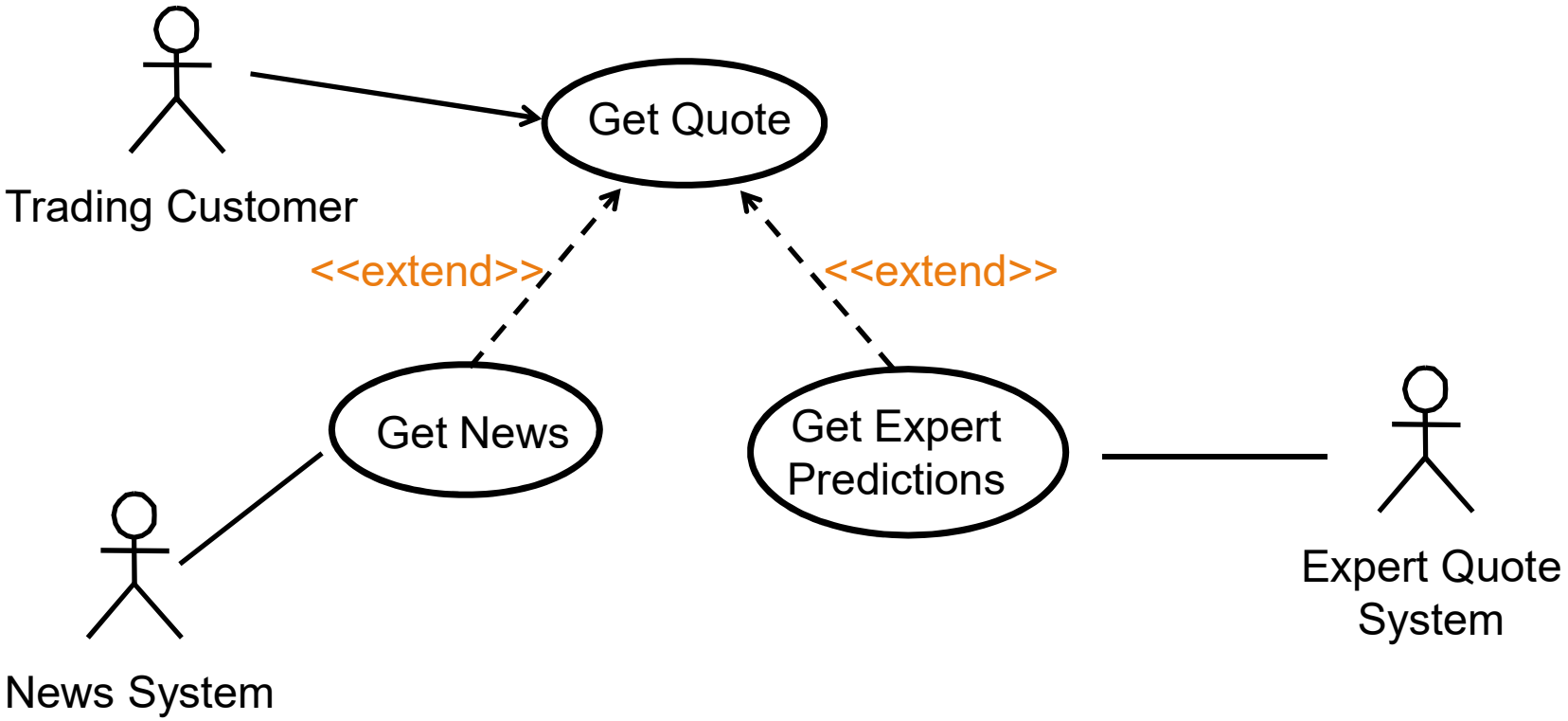


什么是Extend关系?

- 从基本用例到扩展用例的联接
  - 将扩展用例的行为插入基本用例
  - 只有当扩展条件为真是才进行插入
  - 在一个命名的扩展点插入基本用例



Extend 关系示例


Get Quote
Trading Customer
<<extend>>
<<extend>>
Get News
Get Expert
Predictions
Expert Quote
System
News System

Extend 关系示例(续)

Get Quote Use Case

Basic Flow:

1. Include "Identify Customer" to verify customer's identity.
2. Display options.
3. Customer selects "Get Quote."
4. Customer gets quote.
5. Customer gets other quotes.
6. Customer logs off.

A1. Quote System unavailable

…

Extension Points:
The "Optional Services" extension point occurs at Step 3 in the Basic Flow.

Get News Use Case

This use case extends the Get Quote Use Case, at extension point "Optional Services."

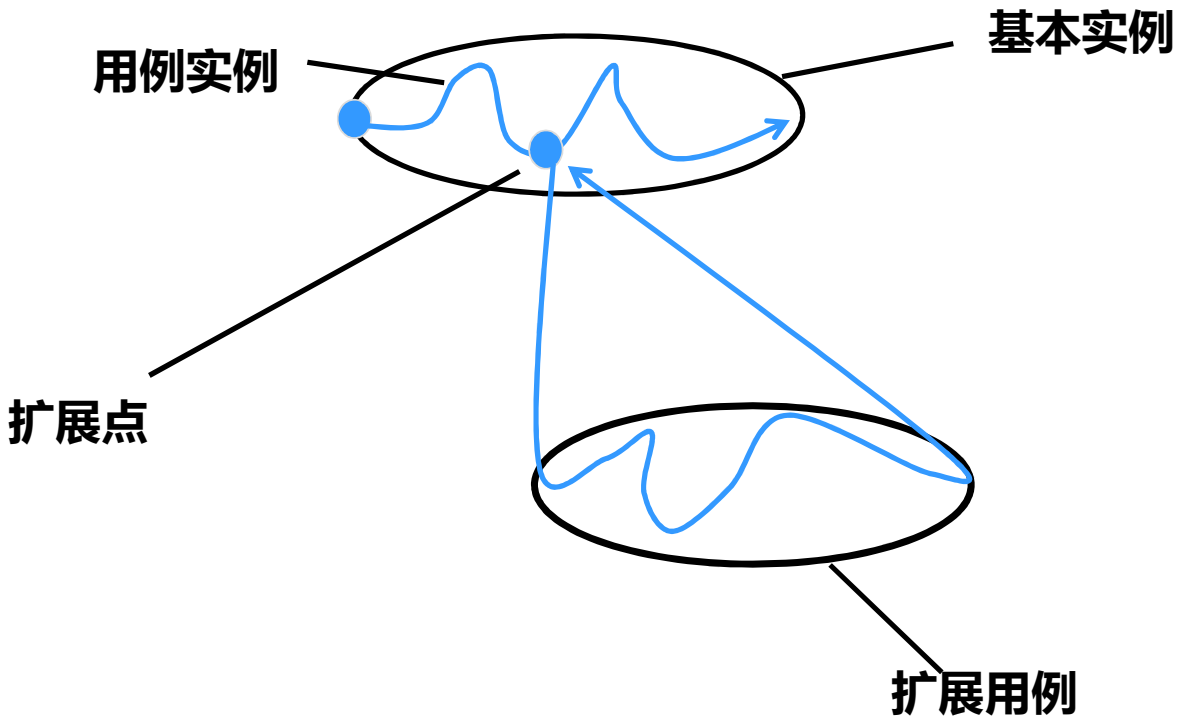
Basic Flow:

1. **If Customer selects "Get News," the system asks customer for time period and number of news items.**
2. **Customer enters time period and number of items. The system sends stock trading symbol to News System, receives reply, and displays the news to the customer.**
3. **The Get Quote Use Case continues.**

A1: News System Unavailable
A2: No News About This Stock
…

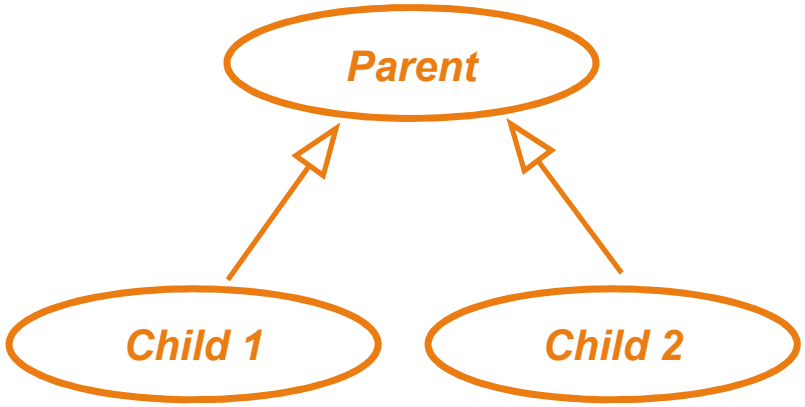
执行Extend

- 当到达扩展点并且扩展条件为真时执行

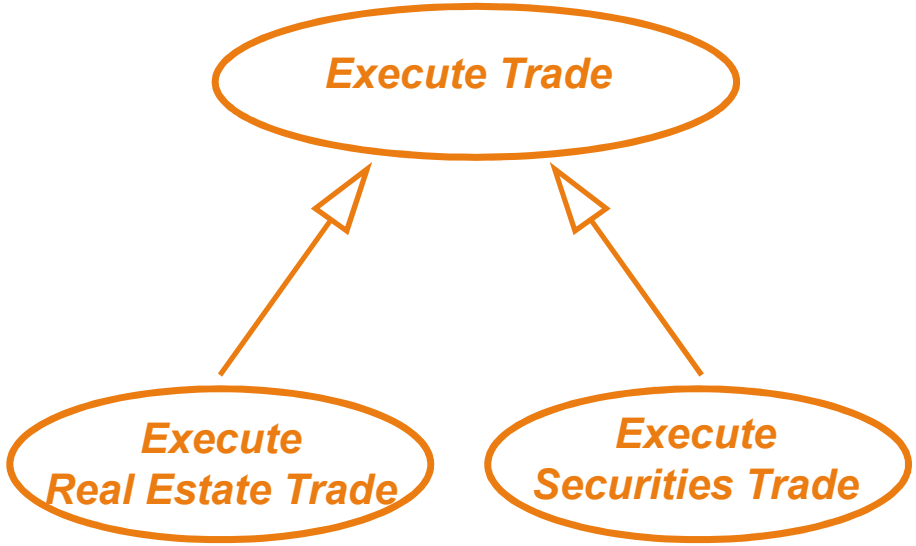


什么是 Use-Case Generalization?

- 从子用例到父用例的关系
  - 在父用例中描述通用的共享的行为
  - 在子用例中描述特殊的行为
  - 共享共同的目标



Generalization示例



Generalization 关系示例

Execute Trade Use Case

Basic Flow:

1. Customer Logs On
   Include “Identify Customer” to verify customer identity …
2. Customer Selects “Trade.”
   Customer chooses trade …
3. Customer Selects Account
   Customer selects account.
   System displays account …
4. *Perform Trade*

5. Customer Begins New Trade
   If customer wants other trades, repeat from Step 3 …
6. Display Summary
   System displays summary …

A1. Account Not Operational…

Execute Securities Trade Use Case

This use case is a child of the Execute Trade Use Case.

Basic Flow:

1. Customer Logs On

2. Customer Selects “Trade”

3. Customer Selects Account

4. Perform Trade

   If customer selects trade order type. The system performs…Limit Sell, Limit Buy, ... The system sends…

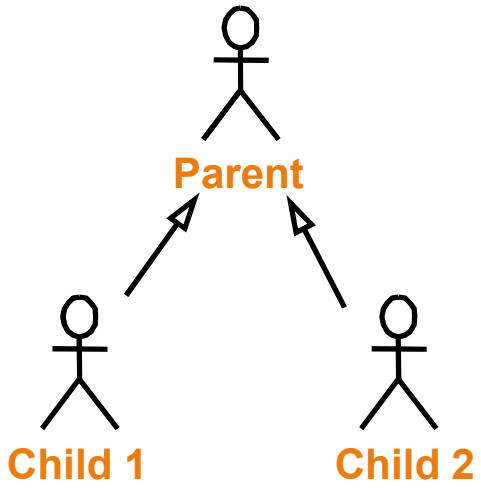
5. Customer Begins New Trade

6. Display Summary

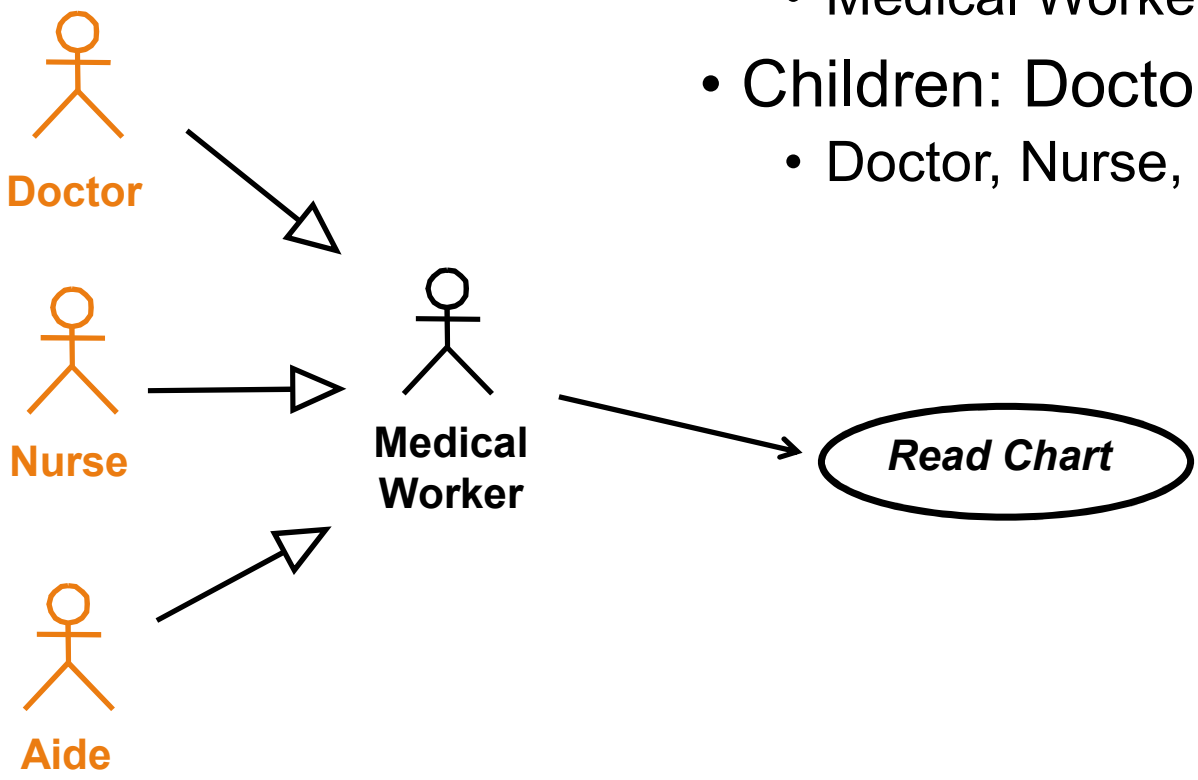
A1: Limit Sell Order …

什么是Actor Generalization?

- Actors 可能有公共的属性
- 多个actors 和Use-Case交互时可能有公共的角色或目的



Actor Generalization示例



- Parent: Medical Worker
  - Medical Worker can read charts
- Children: Doctor, Nurse, Aide
  - Doctor, Nurse, and Aide can read charts

大 纲

01-面向对象方法概述

02-面向对象的基本概念

03-用例建模

04-建立概念模型

05-用例分析

参考附录：UML到C++的映射
UML到Java的映射
UML到VB的映射

面向对象分析的步骤

1. 用例建模

- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

- **2.1 识别Conceptual Class**
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性，画状态图

3. 用例分析

概念类图的作用

- When modeling the static view of a system, class diagrams are typically used in one of three ways, to model:
  - The vocabulary of a system
  - Collaborations
  - A logical database schema

Identify Conceptual Class

- Strategies to Identify Conceptual Classes
  - Use a conceptual class category list
  - Finding conceptual classes with Noun Phrase
  - Use analysis patterns, which are existing conceptual models created by experts
    - using published resources such as Analysis Patterns [Fowler96] and Data Model Patterns [Hay96].

Use a conceptual class category list

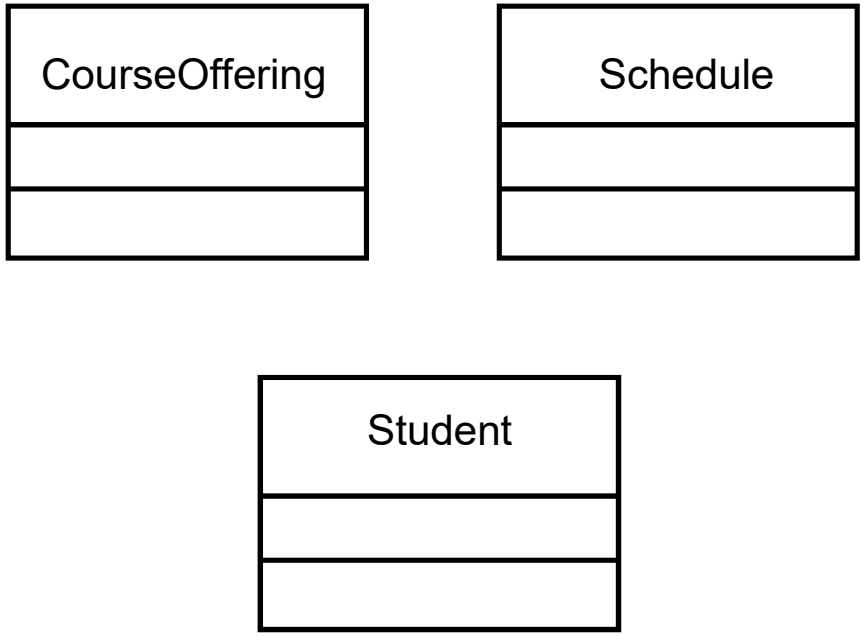
Category	Conceptual Classes
physical or tangible objects	Student Professor FulltimeStudent ParttimeStudent
form or abstract noun concepts	Course Course Offering Schedule
organizations	Dept

Finding conceptual classes with Noun Phrase

- Use use-case flow of events as input
- Underline noun clauses in the use-case flow of events
- Remove redundant candidates
- Remove vague candidates
- Remove actors (out of scope)
- Remove implementation constructs
- Remove attributes (save for later)
- Remove operations

Example: Candidate Conceptual Classes

- Register for Courses (Create Schedule)



A Common Mistake in Identifying Conceptual Classes

- Perhaps the most common mistake when creating a domain model is to represent something as an attribute when it should have been a concept.
- A rule of thumb to help prevent this mistake:
  - If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute.
- Example
  - Should store be an attribute of Sale, or a separate conceptual class Store?
  - In the real world, a store is not considered a number or text - the term suggests a legal entity, an organization, and something occupies space.
  - Therefore, Store should be a concept.

面向对象分析的步骤

1. 用例建模

- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

- 2.1 识别Conceptual Class
- **2.2 建立Conceptual Class之间的关系**
- 2.3 增加 Conceptual Class的属性，画状态图

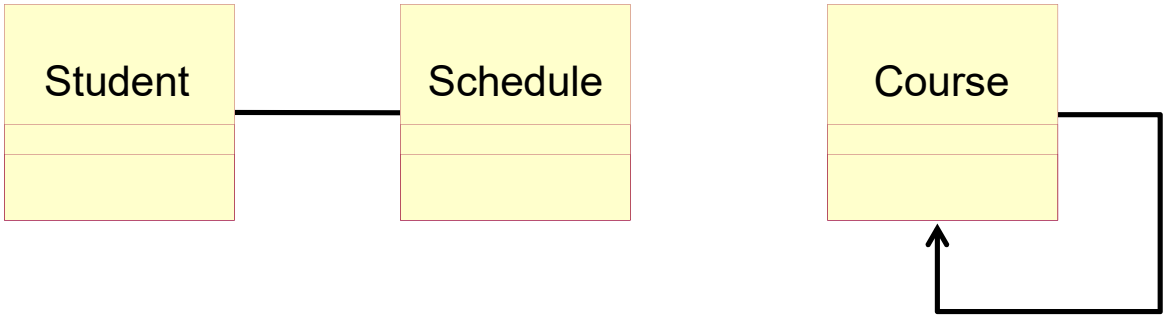
3. 用例分析

Class relationships

- Association
  - Aggregation
  - Composition
- Generalization
- Dependency

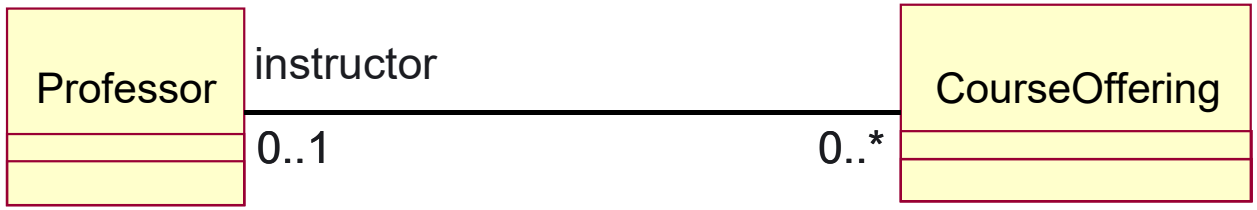
Relationships: Association

- The semantic relationship between two or more classifiers that specifies connections among their instances.
- A structural relationship specifying that objects of one thing are connected to objects of another thing.



What Is Multiplicity?

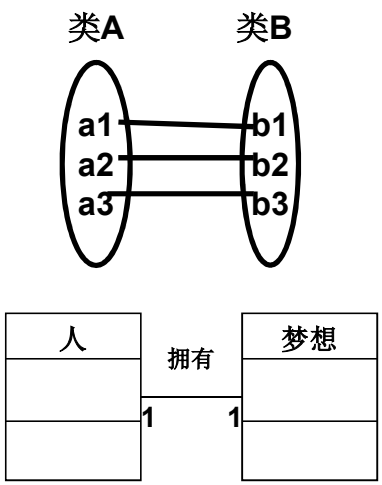
- Multiplicity is the number of instances one class relates to ONE instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.
  - For each instance of Professor, many Course Offerings may be taught.
  - For each instance of Course Offering, there may be either one or zero Professor as the instructor.

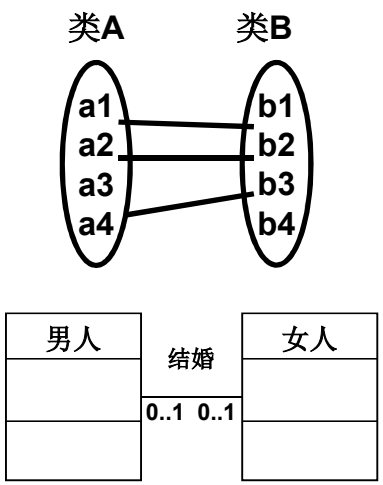


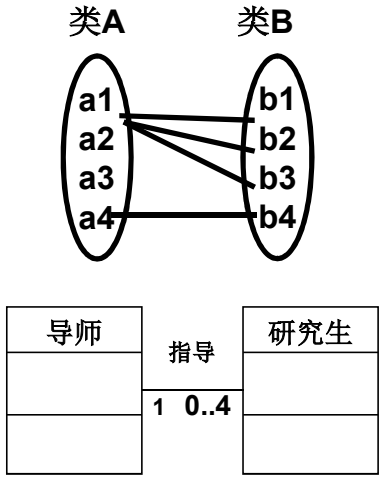
Multiplicity Indicators

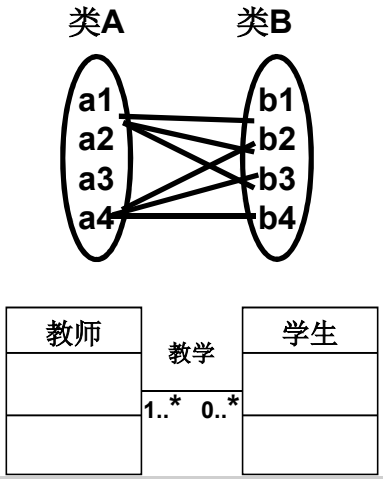
Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional value)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

举例

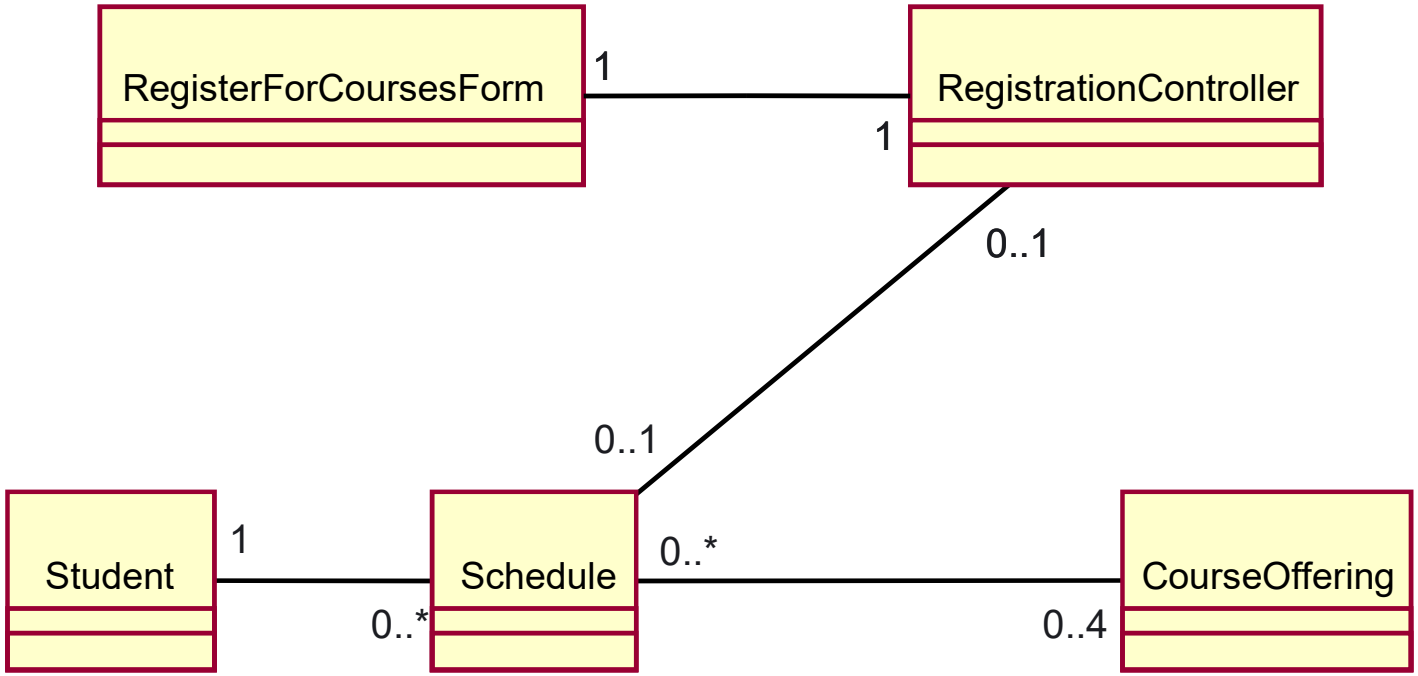

类A
类B
a1
a2
a3
b1
b2
b3
人
拥有
梦想
1
1


类A
类B
a1
a2
a3
a4
b1
b2
b3
b4
男人
结婚
女人
0..1
0..1


类A
类B
a1
a2
a3
a4
b1
b2
b3
b4
导师
指导
研究生
1
0..4

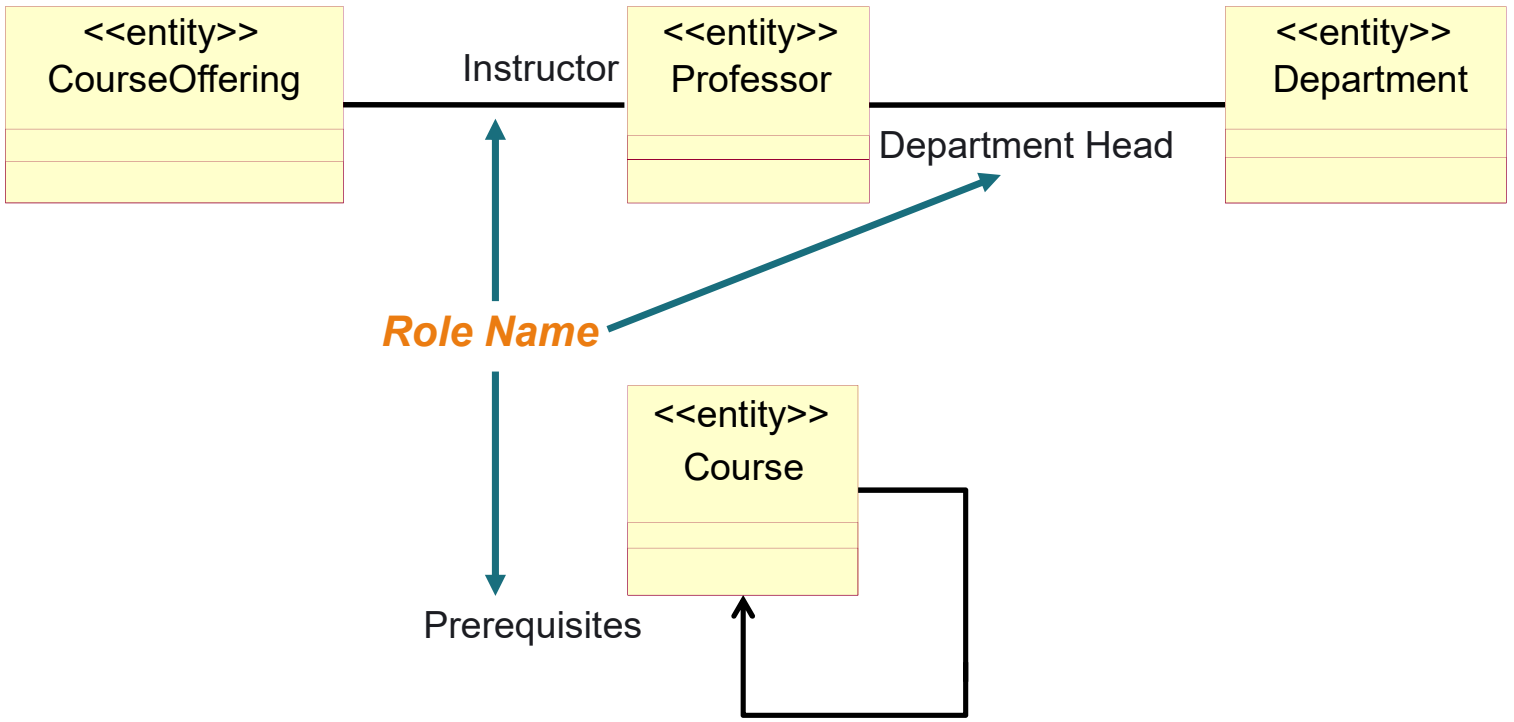

类A
类B
a1
a2
a3
a4
b1
b2
b3
b4
教师
教学
学生
1..*
0..*

Example: Multiplicity



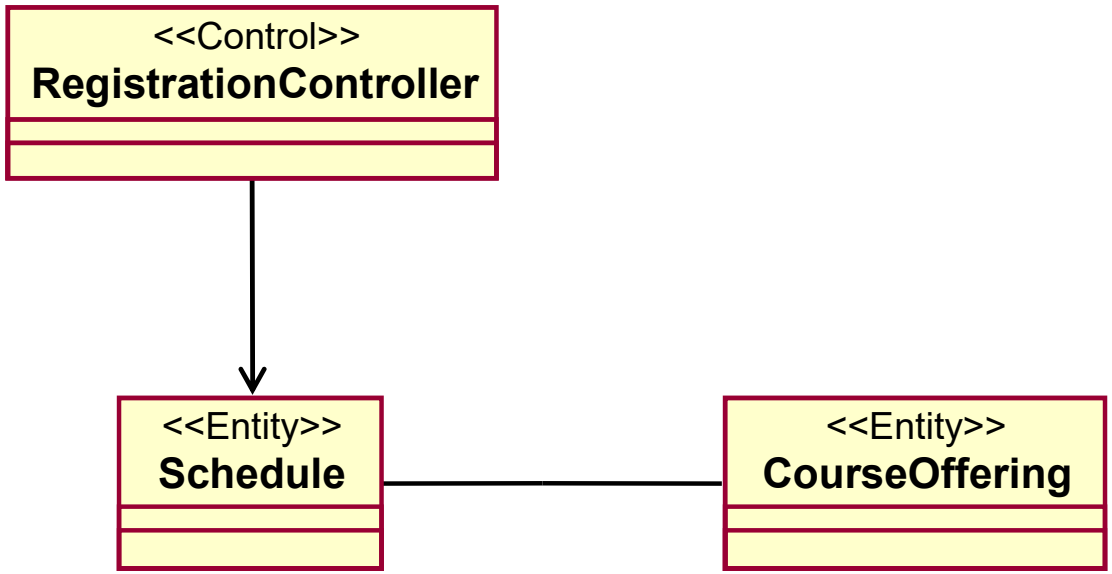
What Are Roles?

- The “face” that a class plays in the association

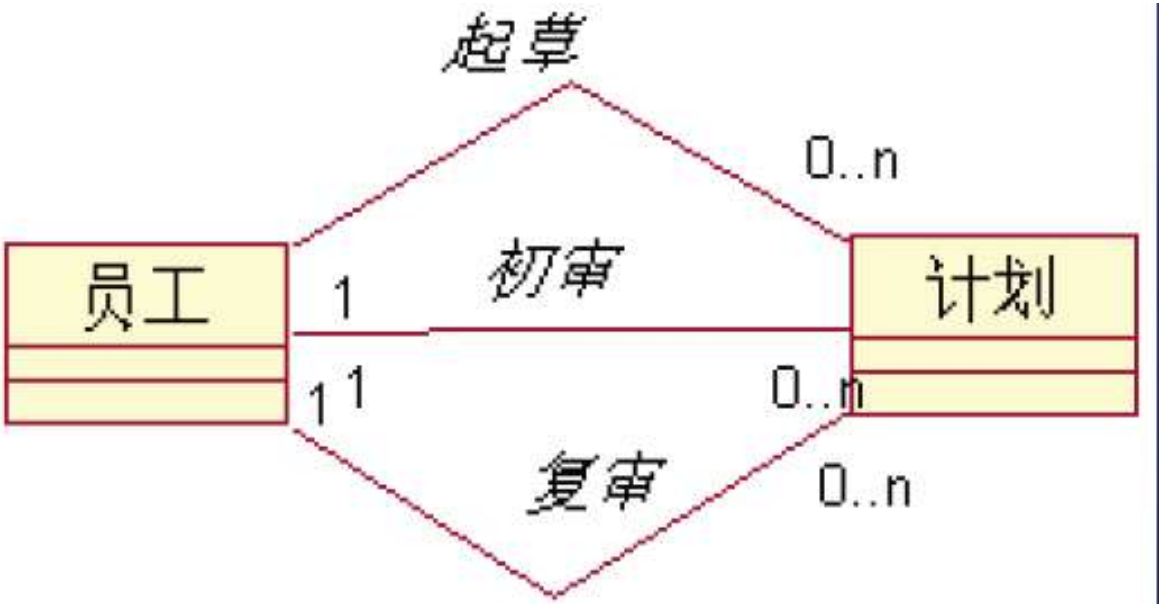


What Is Navigability?

- Indicates that it is possible to navigate from a associating class to the target class using the association

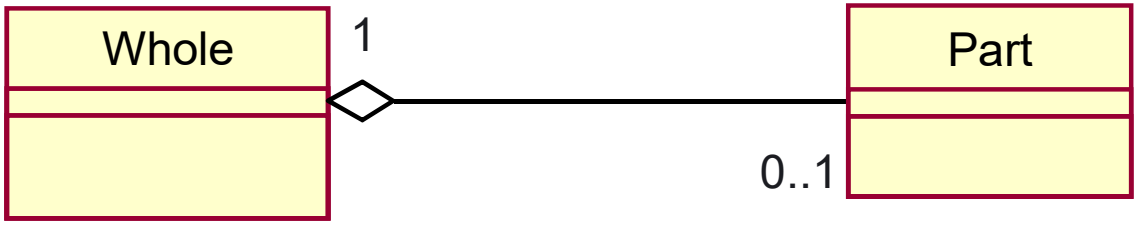


两个类之间关联同时可以有多条

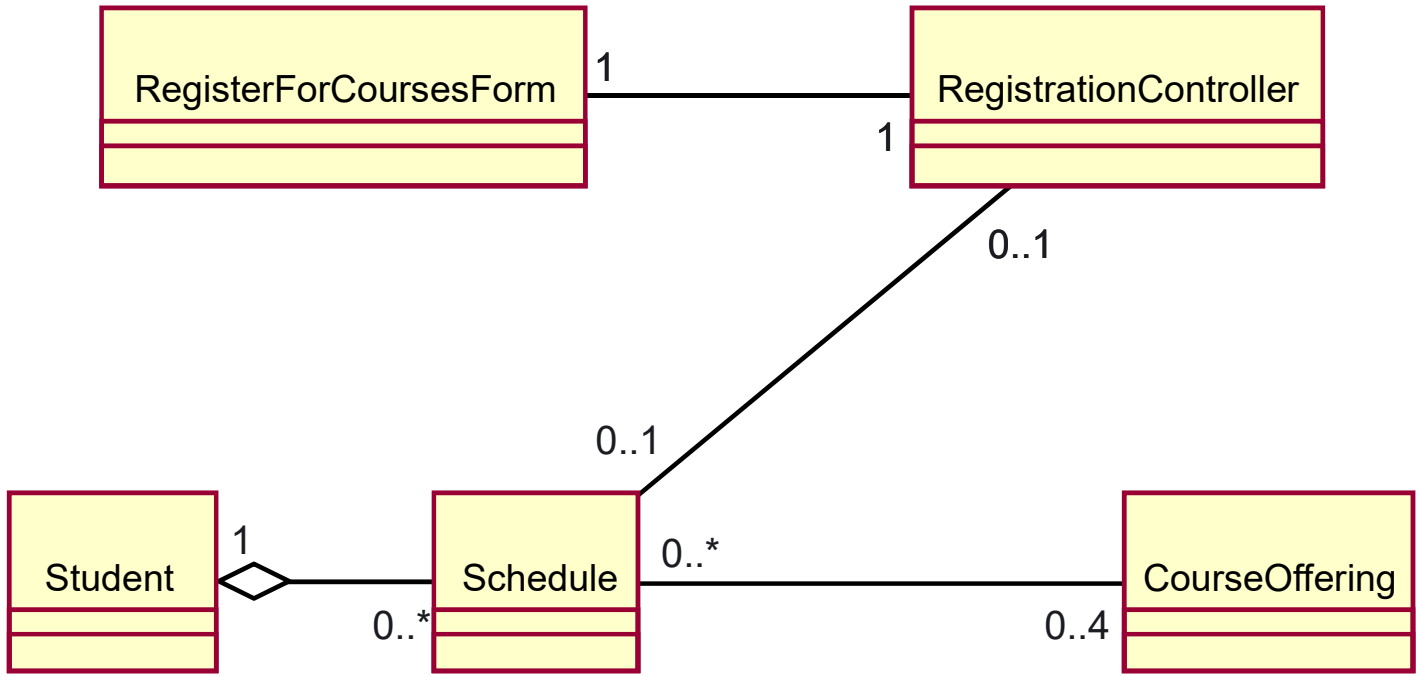


What Is an Aggregation?

- A special form of association that models a whole-part relationship between the aggregate (the whole) and its parts.
  - An aggregation is an “is a part-of” relationship.
- Multiplicity is represented like other associations.

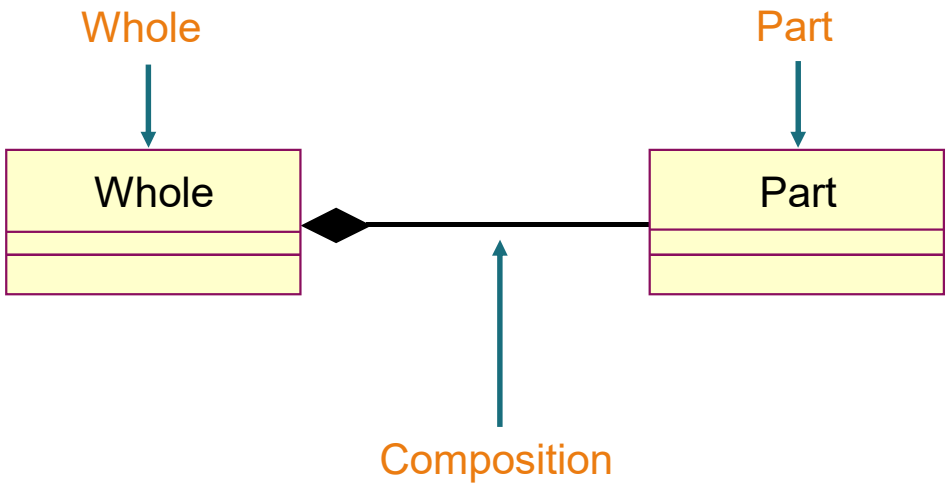


Example: Aggregation

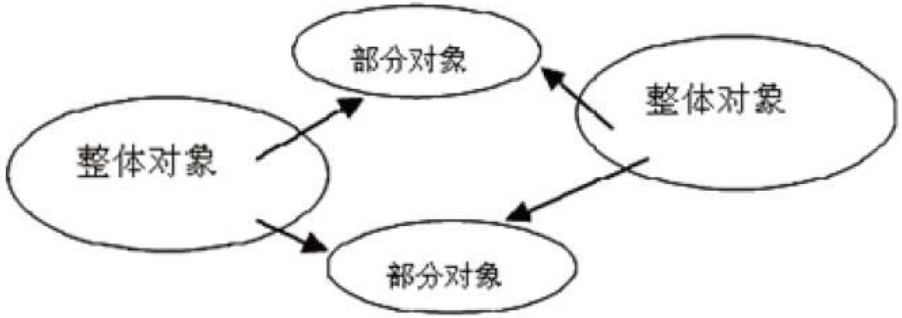


What Is Composition?

- A form of aggregation with strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate

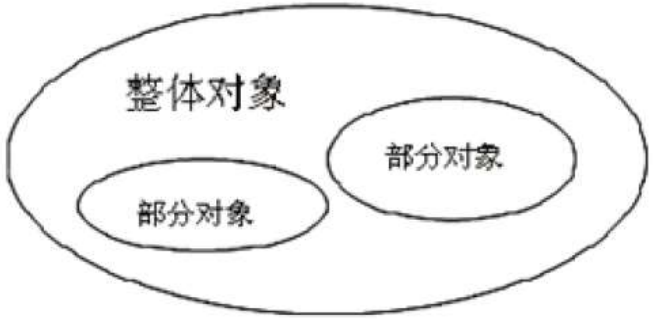


Aggregation Vs. Composition



聚合

例：公司和雇员

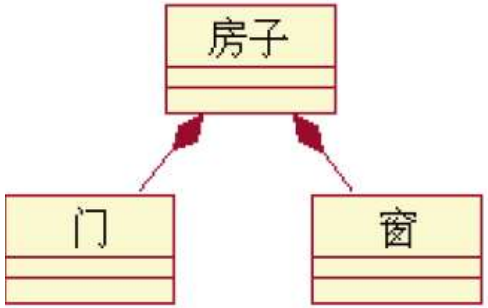


组合

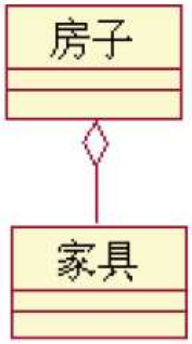
例：订单和订单项

举例

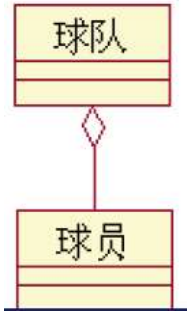
- 组合/部分



- 容器/内容



- 集合/成员



练习

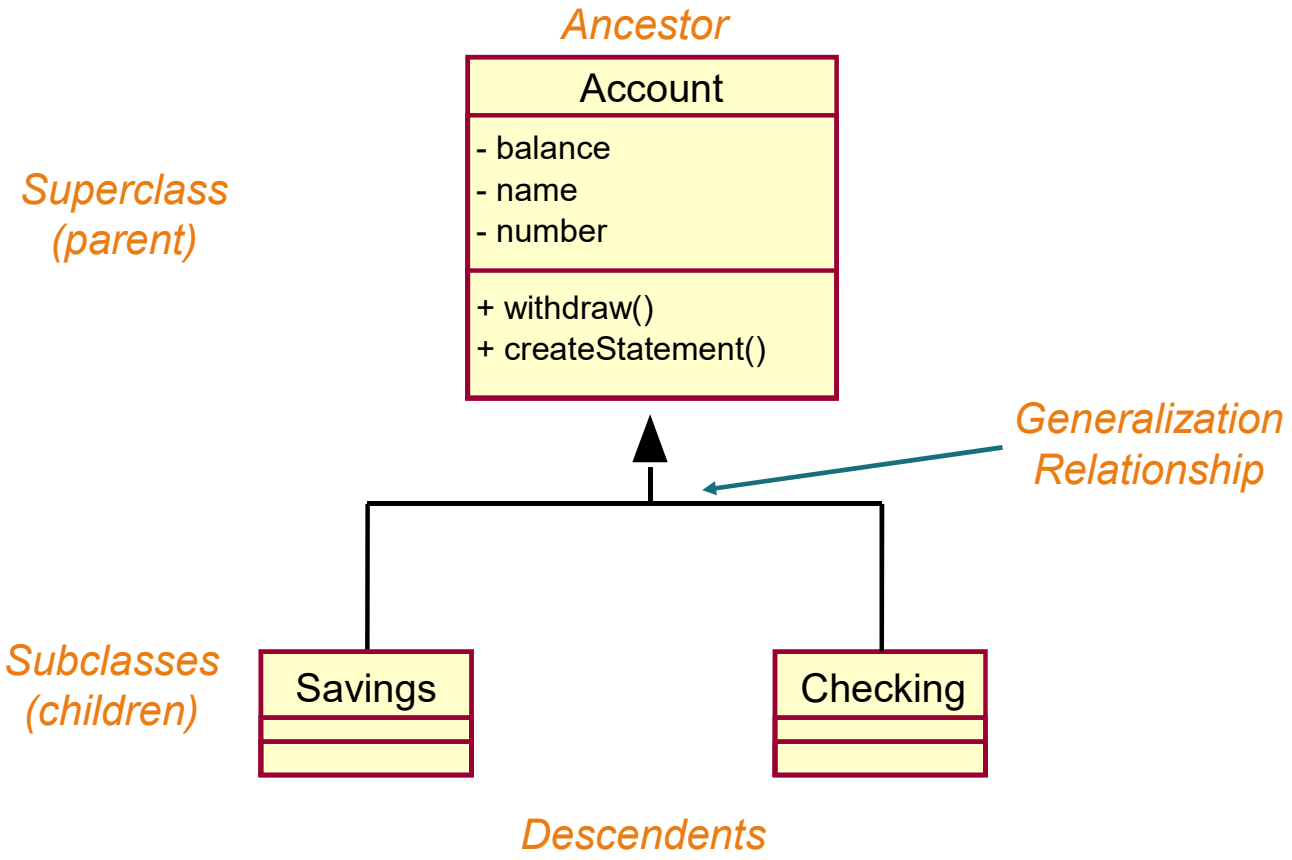
- 汽车和轮胎是关联、聚合、还是组合?
- 丈夫和妻子是关联、聚合、还是组合?

Relationships: Generalization

- A relationship among classes where one class shares the structure and/or behavior of one or more classes.
- Defines a hierarchy of abstractions where a subclass inherits from one or more superclasses.
  - Single inheritance
  - Multiple inheritance
- Is an “is a kind of” relationship.

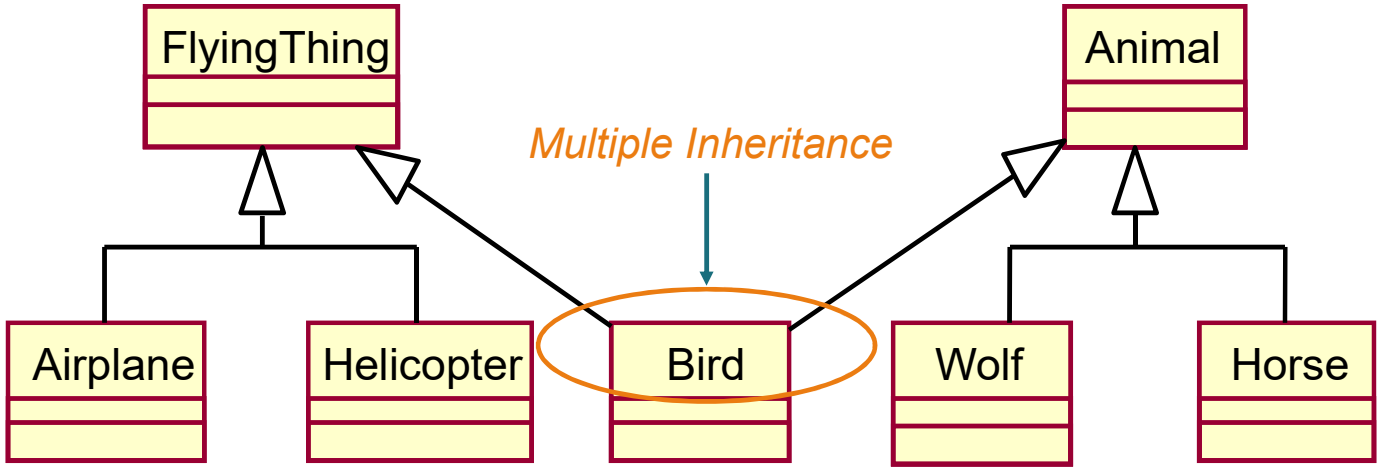
Example: Single Inheritance

- One class inherits from another.



Example: Multiple Inheritance

- A class can inherit from several other classes.



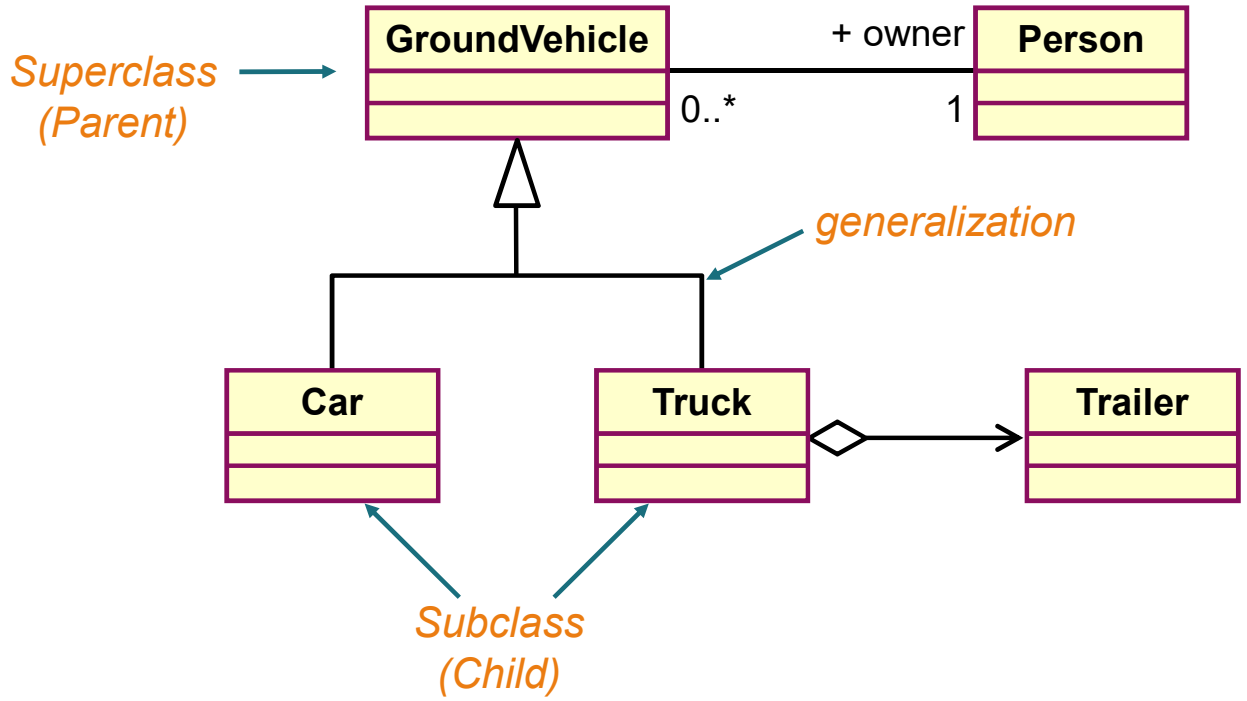
Use multiple inheritance only when needed and always with caution!

What Is Inherited?

- A subclass inherits its parent's attributes, operations, and relationships.
- A subclass may:
  - Add additional attributes, operations, relationships.
  - Redefine inherited operations. (Use caution!)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy.

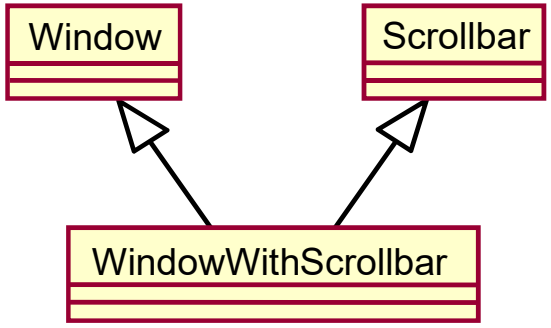
Inheritance leverages the similarities among classes.

Example: What Gets Inherited



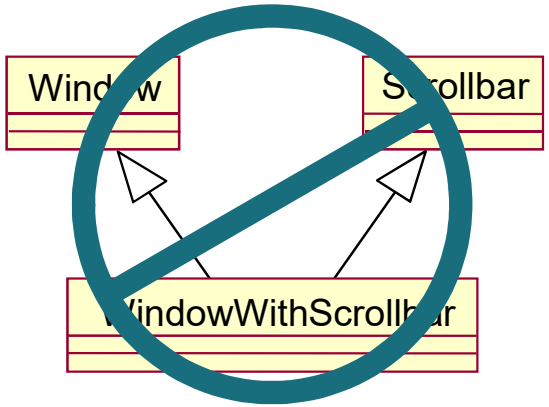
Generalization vs. Aggregation

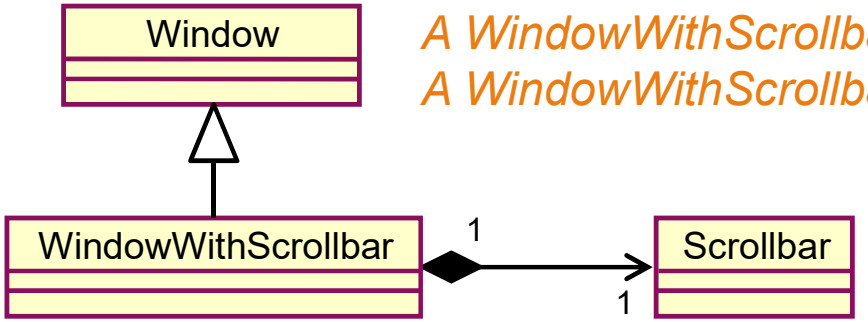
- Generalization and aggregation are often confused
  - Generalization represents an “is a” or “kind-of” relationship
  - Aggregation represents a “part-of” relationship



Is this correct?

Generalization vs. Aggregation





A WindowWithScrollbar "is a" Window
A WindowWithScrollbar "contains a" Scrollbar

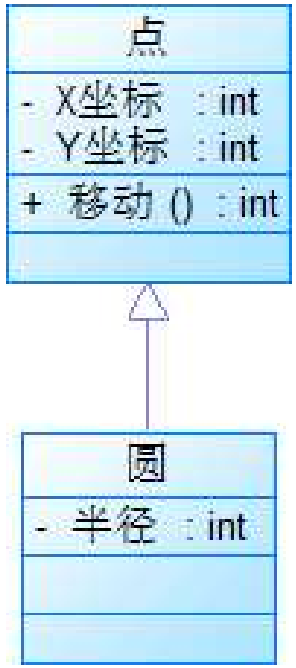
Liskov替换原则 —— LSP

- Liskov Substitution Principle (LSP)

子类型应该能替换基类型

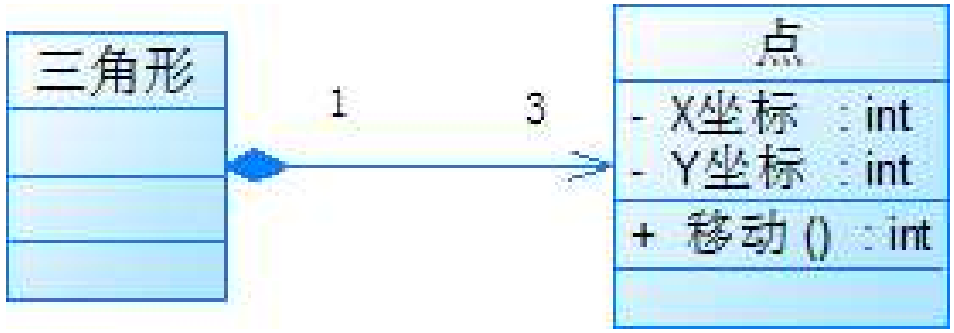
----Barbara Liskov

对吗?

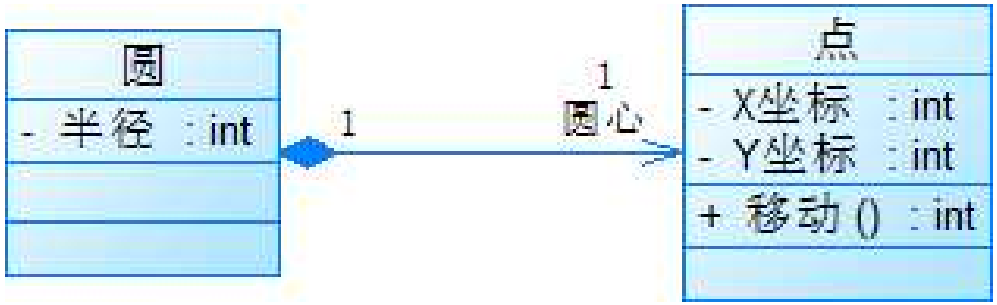


圆是一种有半径的点

子类能替换超类吗?



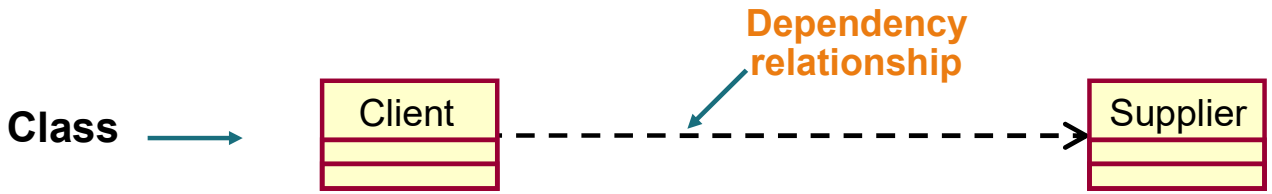
三个圆能组成三角形吗?



关联更能反映领域内涵

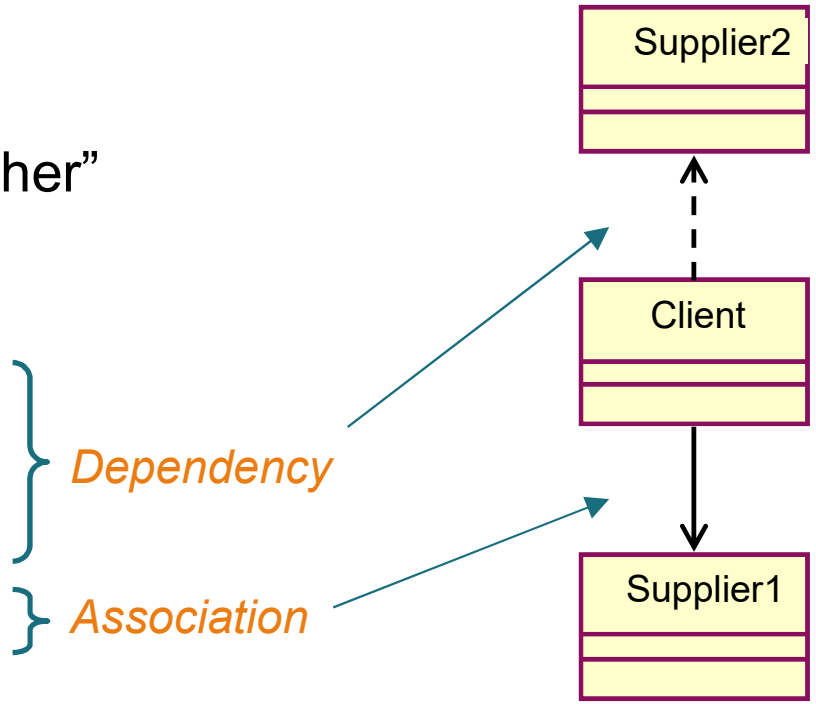
Relationships: Dependency

- A relationship between two classes where a change in one may cause a change in the other
- Non-structural, “using” relationship



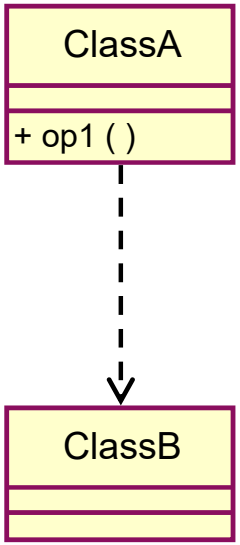
Dependencies vs. Associations

- Associations are structural relationships
- Dependencies are non-structural relationships
- In order for objects to “know each other” they must be visible
  - Local variable reference
  - Parameter reference
  - Global reference
  - Field reference



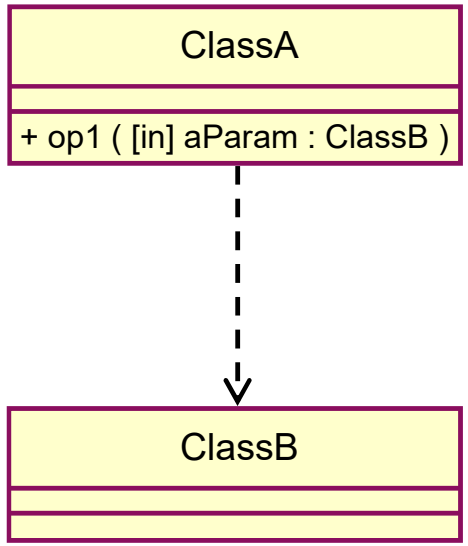
Local Variable Visibility

- The op1() operation contains a local variable of type ClassB



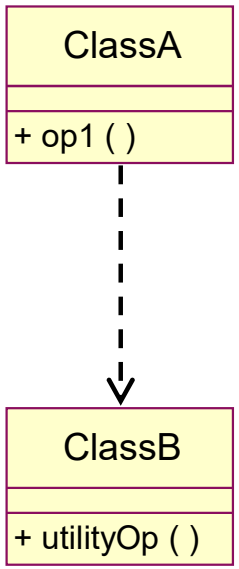
Parameter Visibility

- The ClassB instance is passed to the ClassA instance

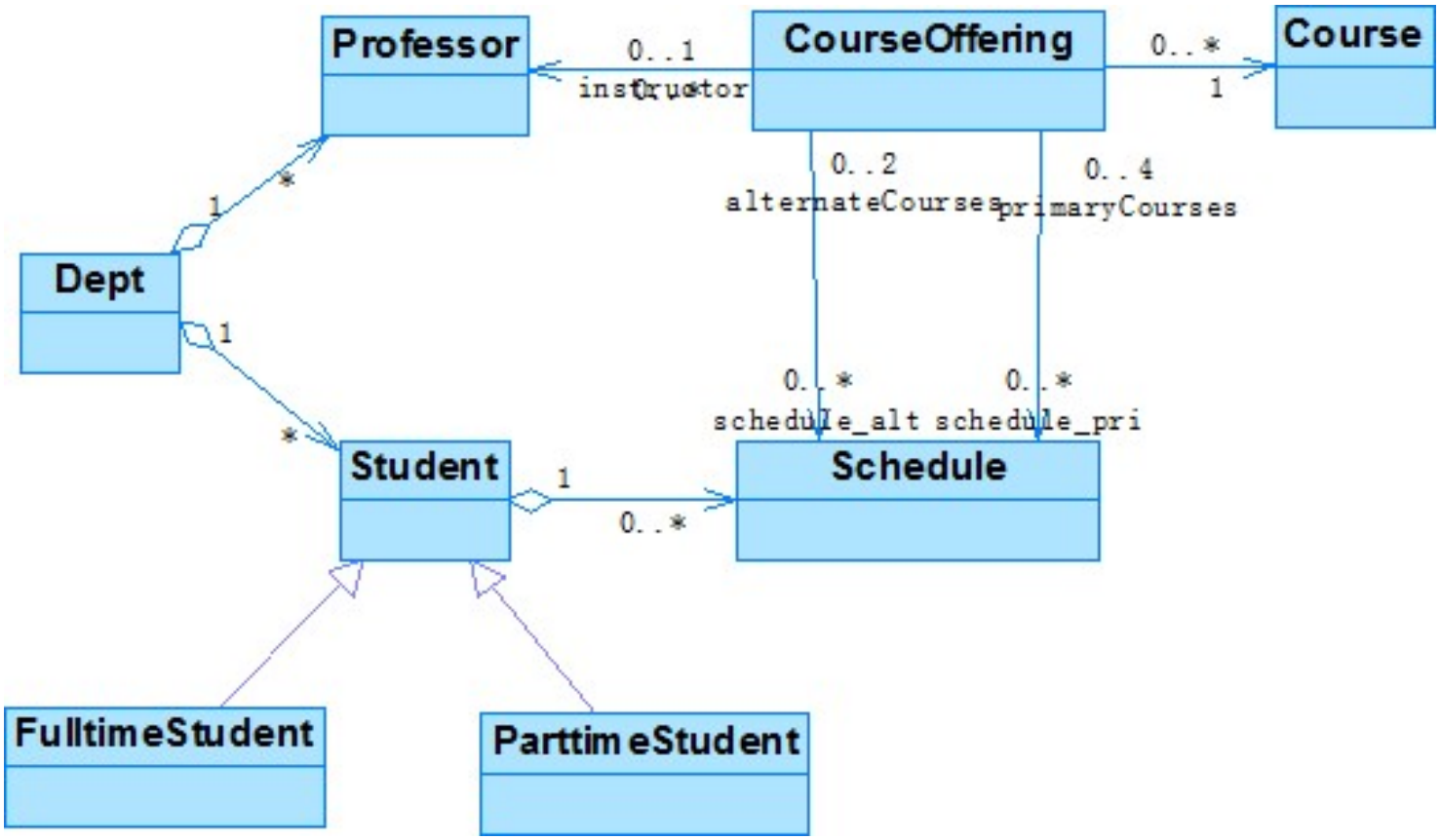


Global Visibility

- The ClassUtility instance is visible because it is global



Example: Conceptual Model



面向对象分析的步骤

1. 用例建模

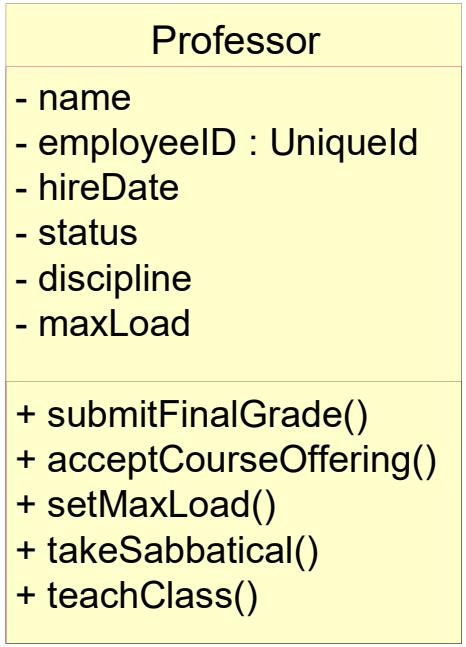
- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- **2.3 增加 Conceptual Class的属性，画状态图**

3. 用例分析

增加 Conceptual Class的属性



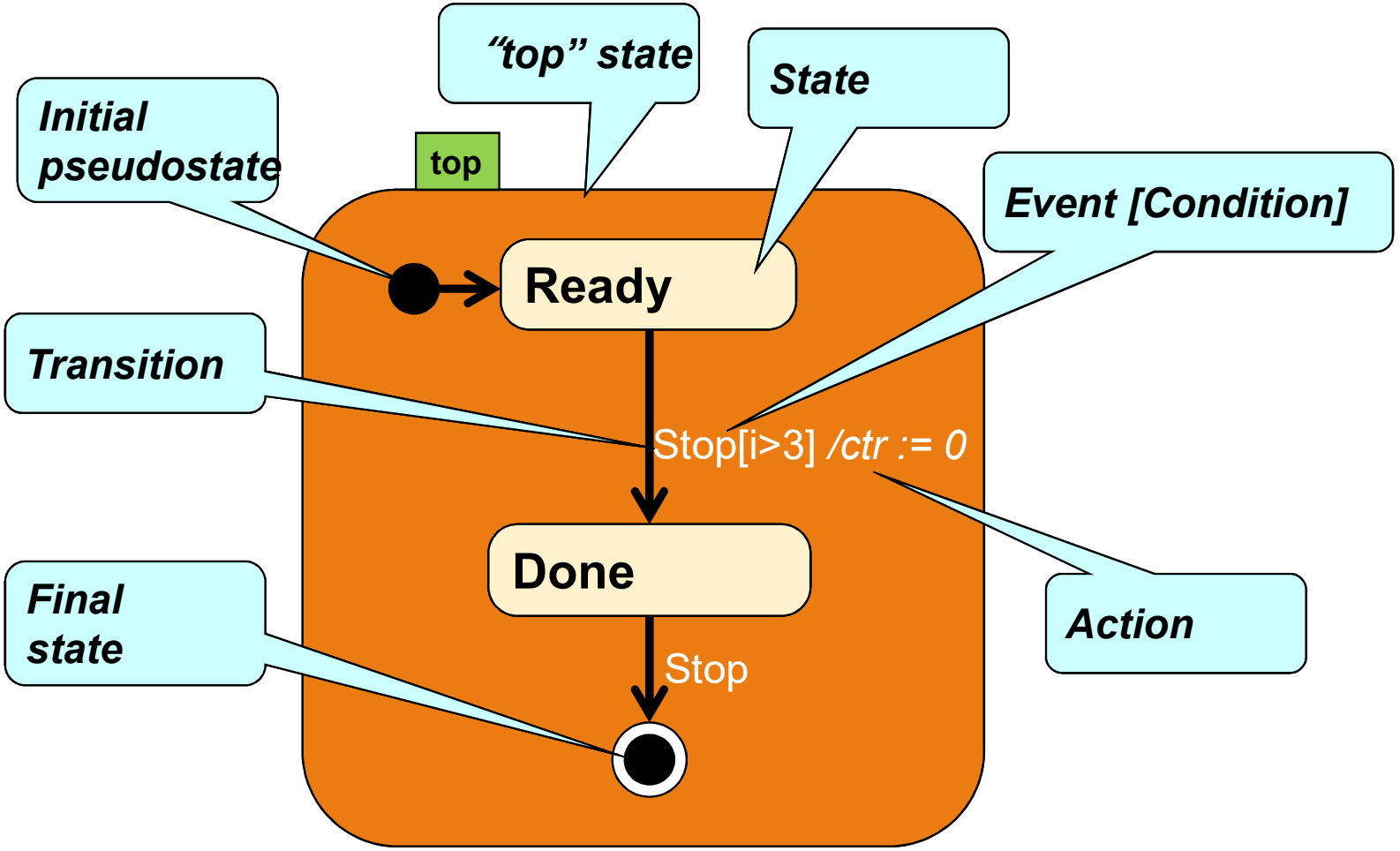
识别出概念类的主要属性，可以有遗漏，

在后续的分析与设计中，这些属性会逐渐补全。

同时可以识别出概念类的部分操作。

在概念模型中，类的属性比操作更为重要。

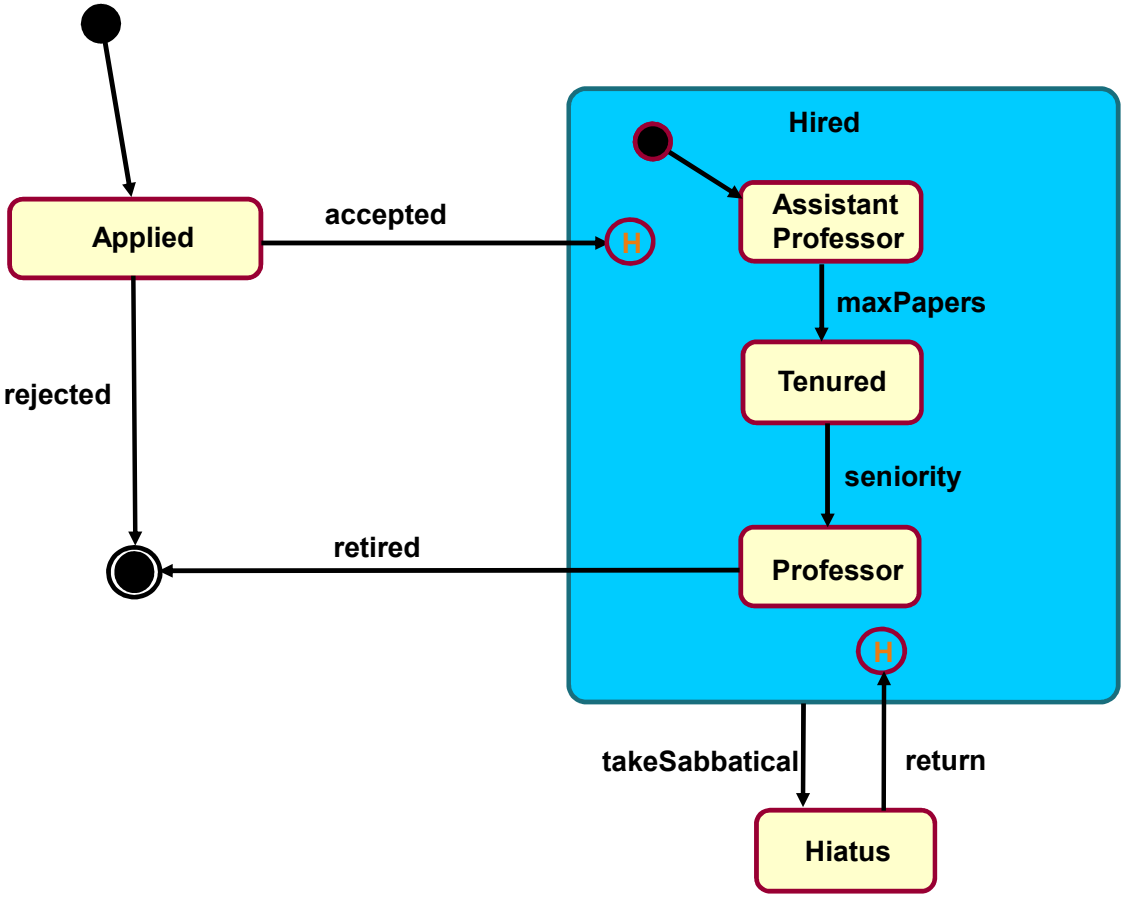
What Are State Machine Diagrams?



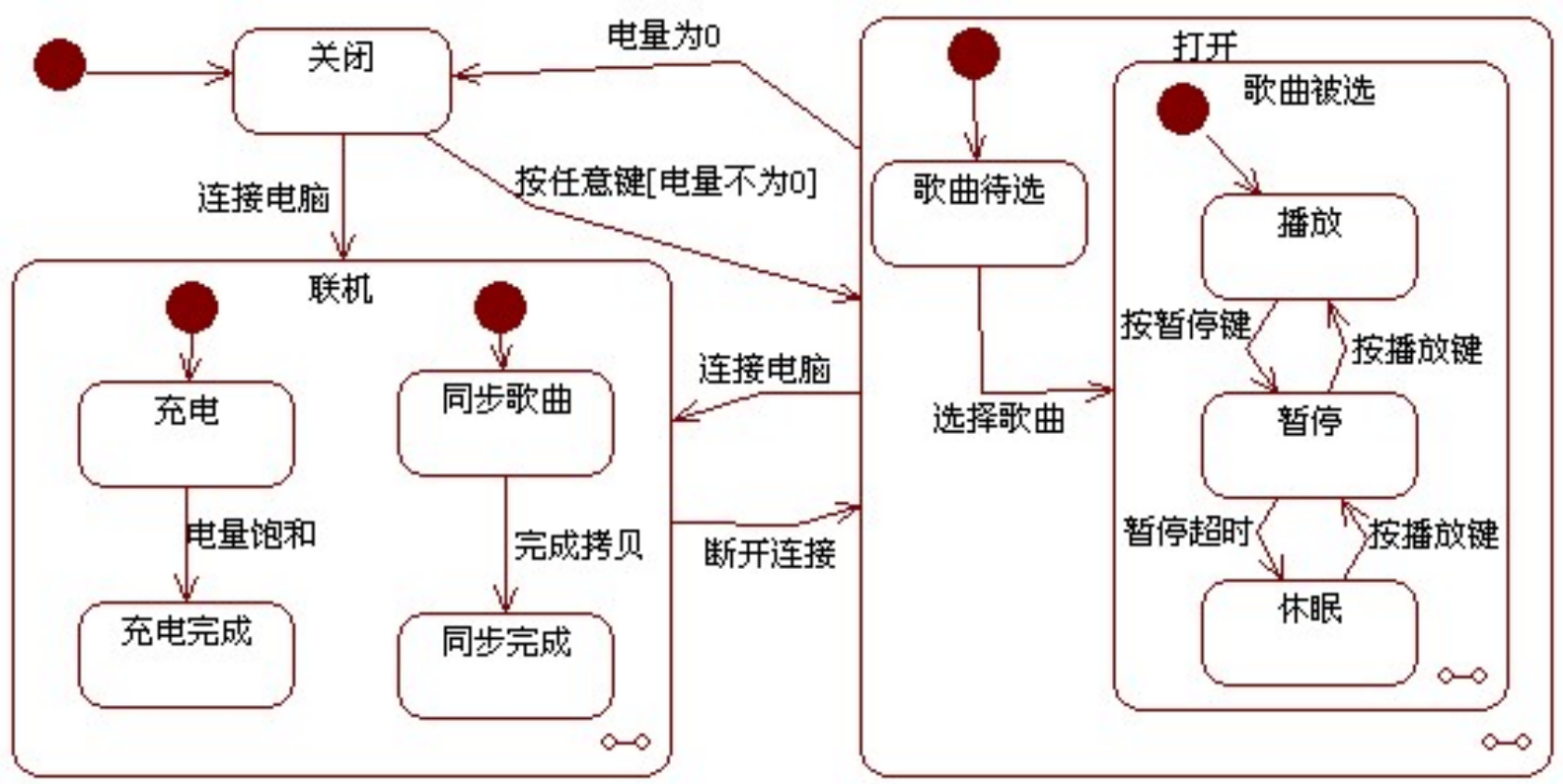
Special States

- The initial state is the state entered when an object is created.
  - An initial state is mandatory.
  - Only one initial state is permitted.
  - The initial state is represented as a solid circle.
- A final state indicates the end of life for an object.
  - A final state is optional.
  - A final state is indicated by a bull's eye.
  - More than one final state may exist.

Example: State Machine

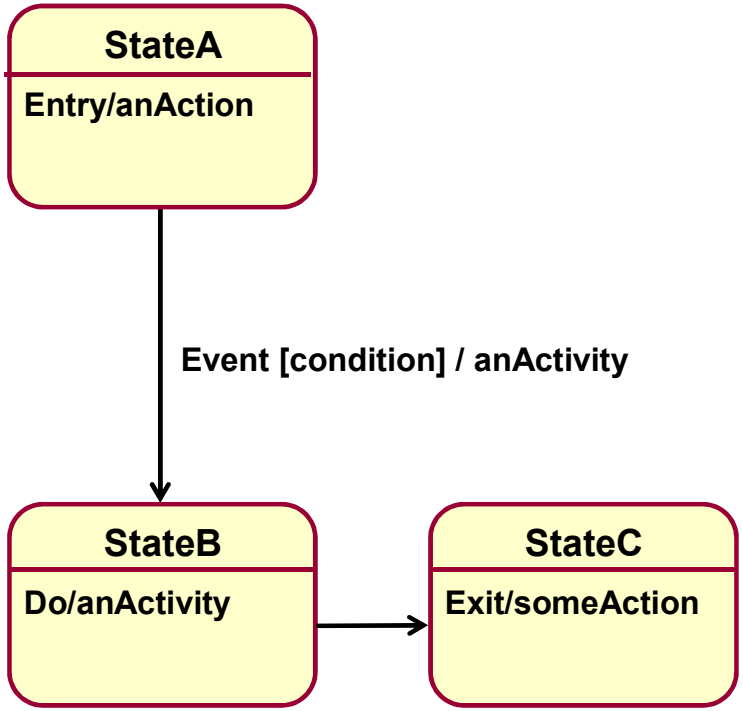


唱片播放器的状态图

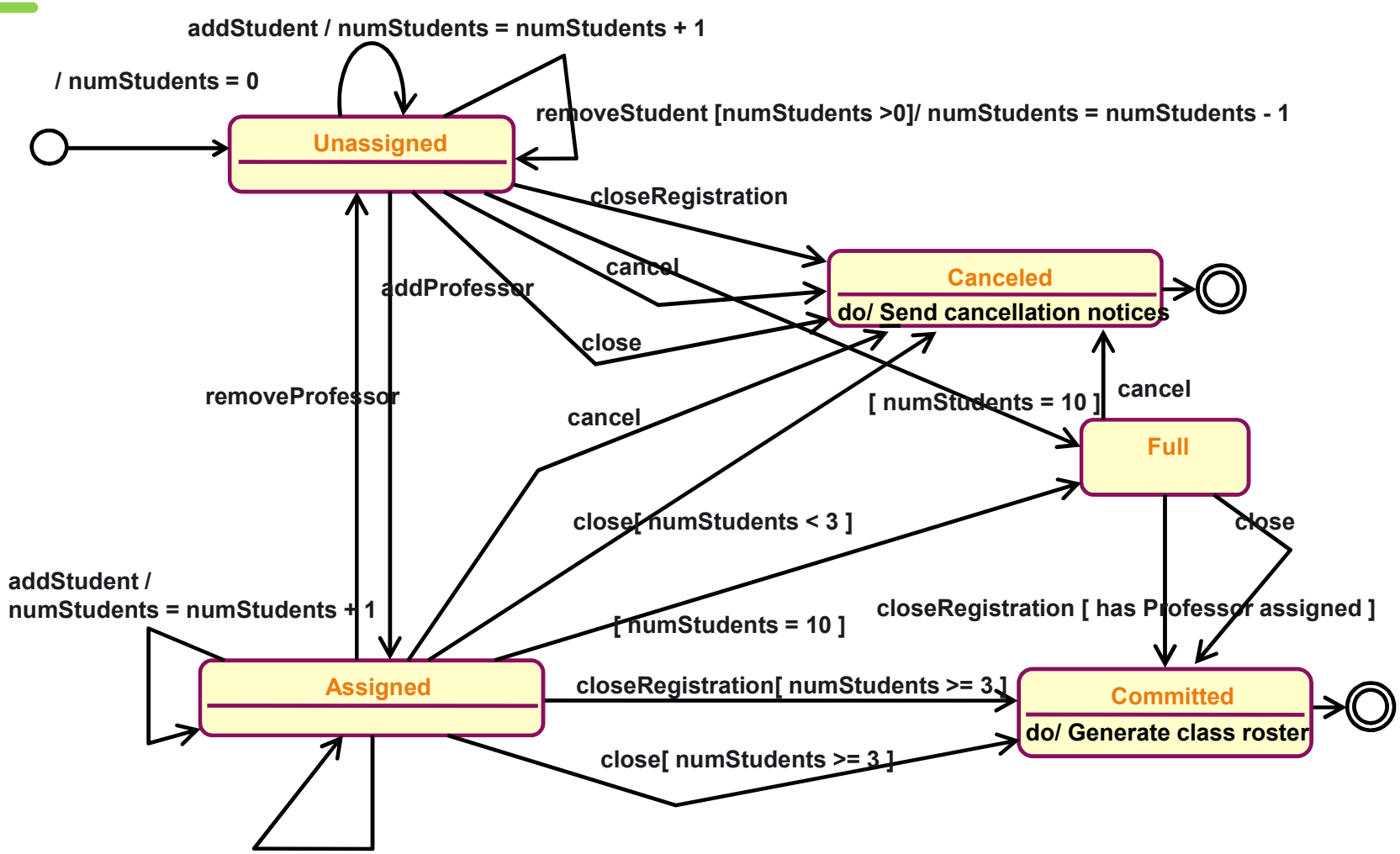

关闭
电量为0
打开
歌曲被选
歌曲待选
播放
连接电脑
按任意键[电量不为0]
联机
按暂停键
按播放键
充电
同步歌曲
连接电脑
选择歌曲
暂停
电量饱和
完成拷贝
断开连接
暂停超时
按播放键
充电完成
同步完成
休眠

Activities and Actions

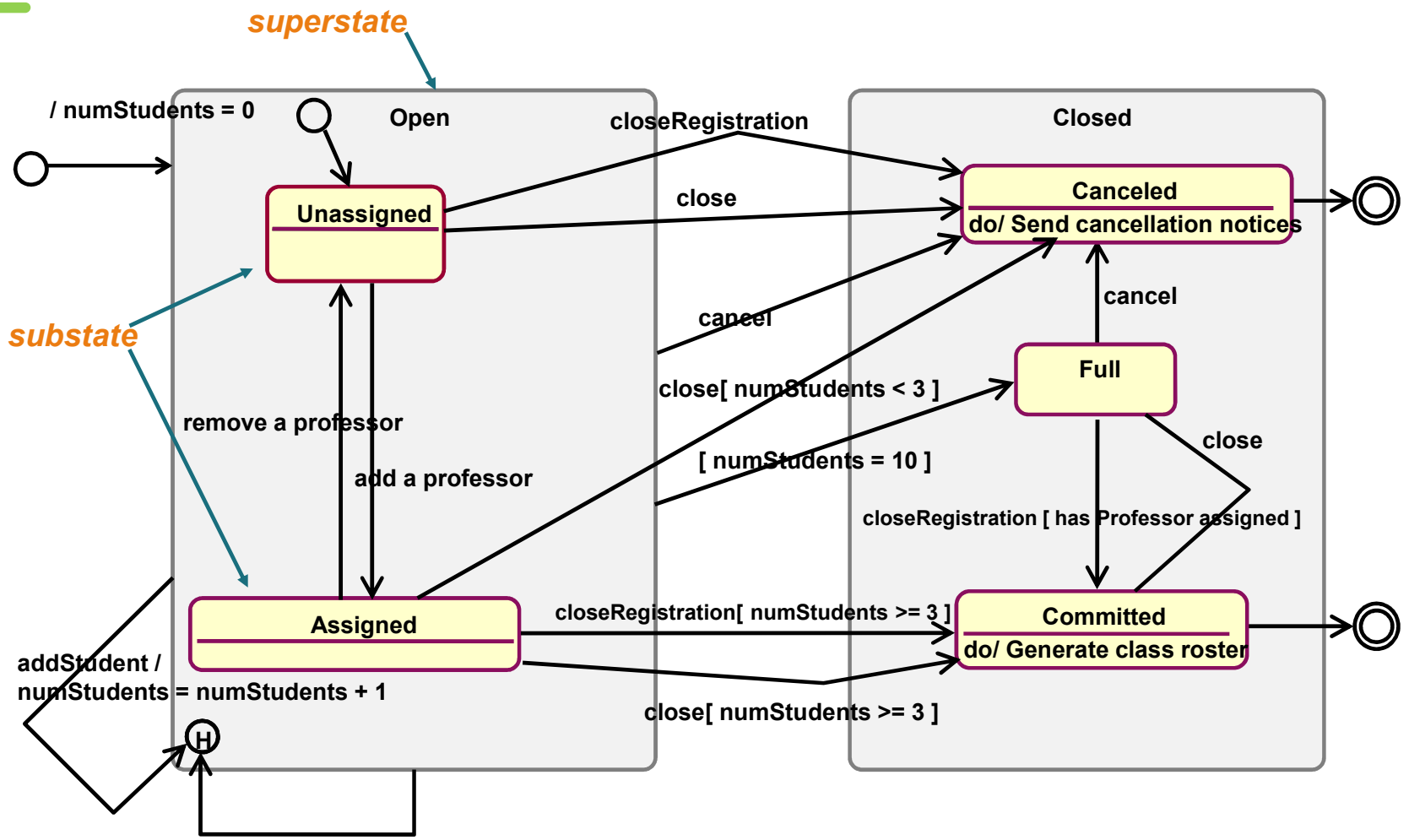
- Entry
  - Executed when the state is entered
- Do
  - Ongoing execution
- Exit
  - Executed when the state is exited
- Event [condition] / anActivity
  - Executed during transition



Example: State Machine



Example: State Machine with Nested States and History



Which Objects Have Significant State?

- Objects whose role is clarified by state transitions
- Complex use cases that are state-controlled
- It is not necessary to model objects such as:
  - Objects with straightforward mapping to implementation
  - Objects that are not state-controlled
  - Objects with only one computational state

讨论

- “女儿”和“父亲”是什么关系?
- “青蛙”和“蝌蚪”是什么关系? （Metamorphosis）

大 纲

01-面向对象方法概述

02-面向对象的基本概念

03-用例建模

04-建立概念模型

05-用例分析

哪些类相互合作实现用例模型中每个用例？

面向对象分析的步骤

1. 用例建模

- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

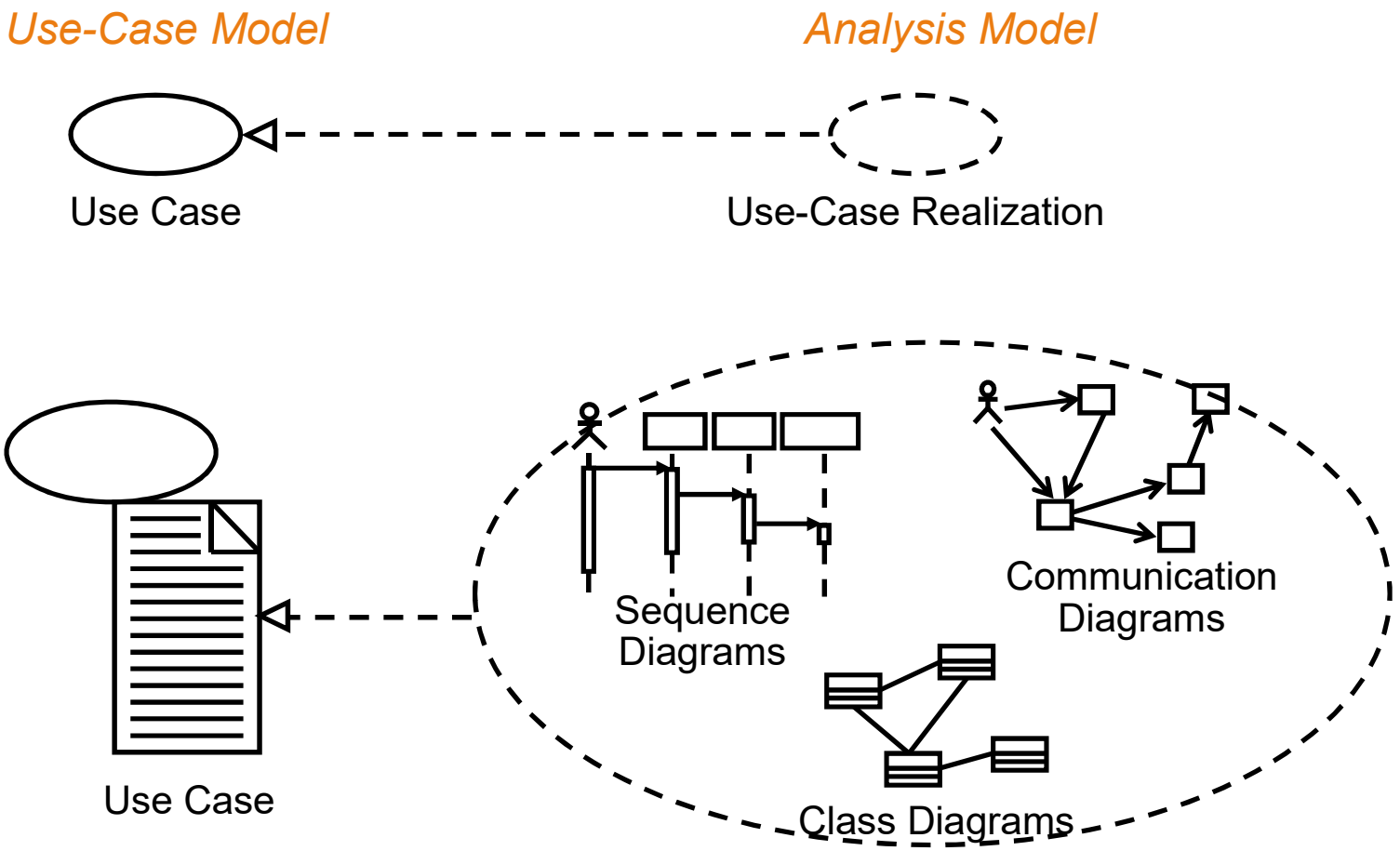
2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性，画状态图

3. 用例分析

- **3.1 识别出用例实现**
- 3.2 针对每个用例实现：
  - 识别出分析类
  - 建立时序图，生成通信图
  - 对照通信图建立类图，完善每个分析类的属性和操作

What is a Use-Case Realization?



面向对象分析的步骤

1. 用例建模

- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性，画状态图

3. 用例分析

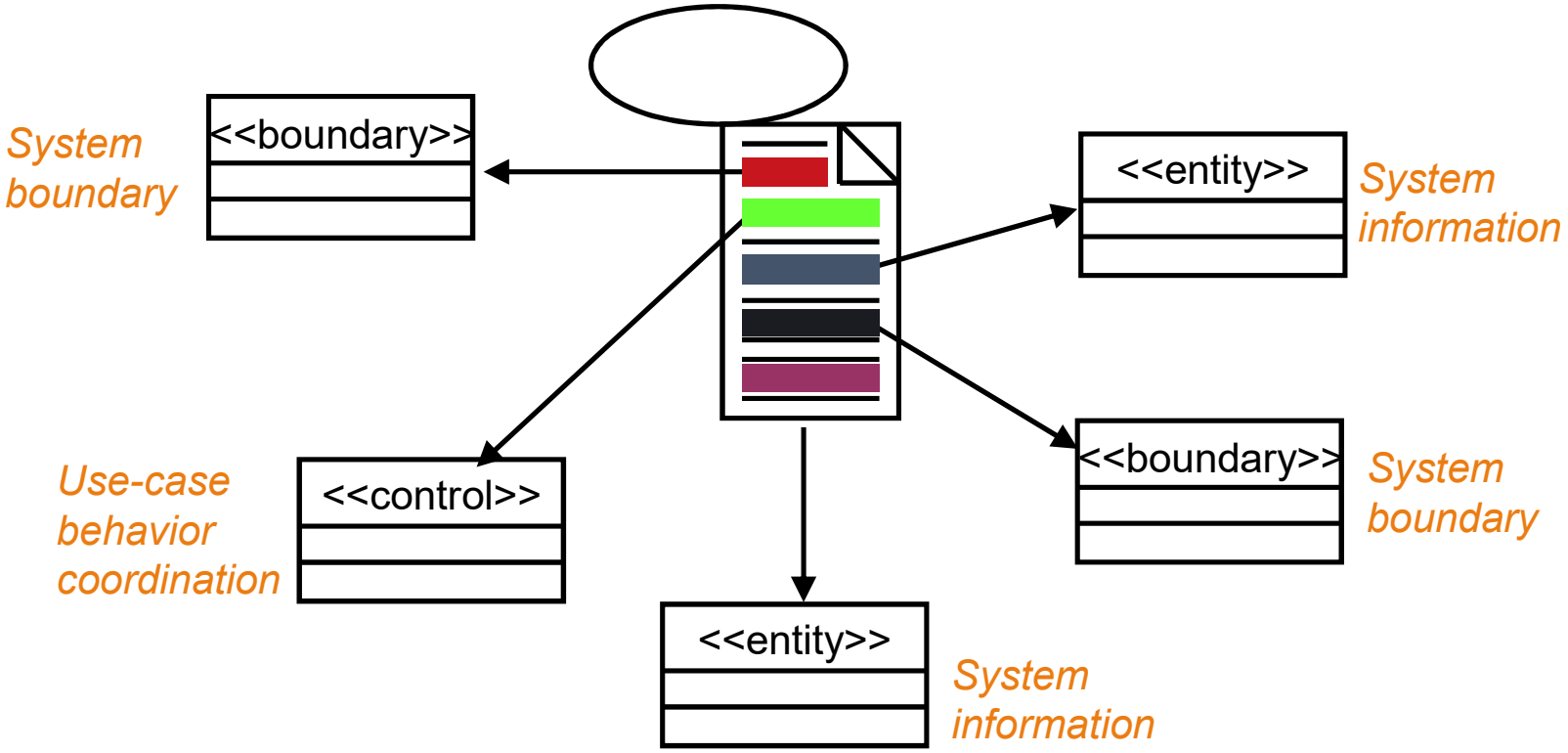
- 3.1 识别出用例实现
- 3.2 针对每个用例实现：
  - **识别出分析类**
  - 建立时序图，生成通信图
  - 对照通信图建立类图，完善每个分析类的属性和操作

三种Analysis Class

- 不同的衍型(stereotype)可采用不同的图标，也可用采用“《》”来区别。

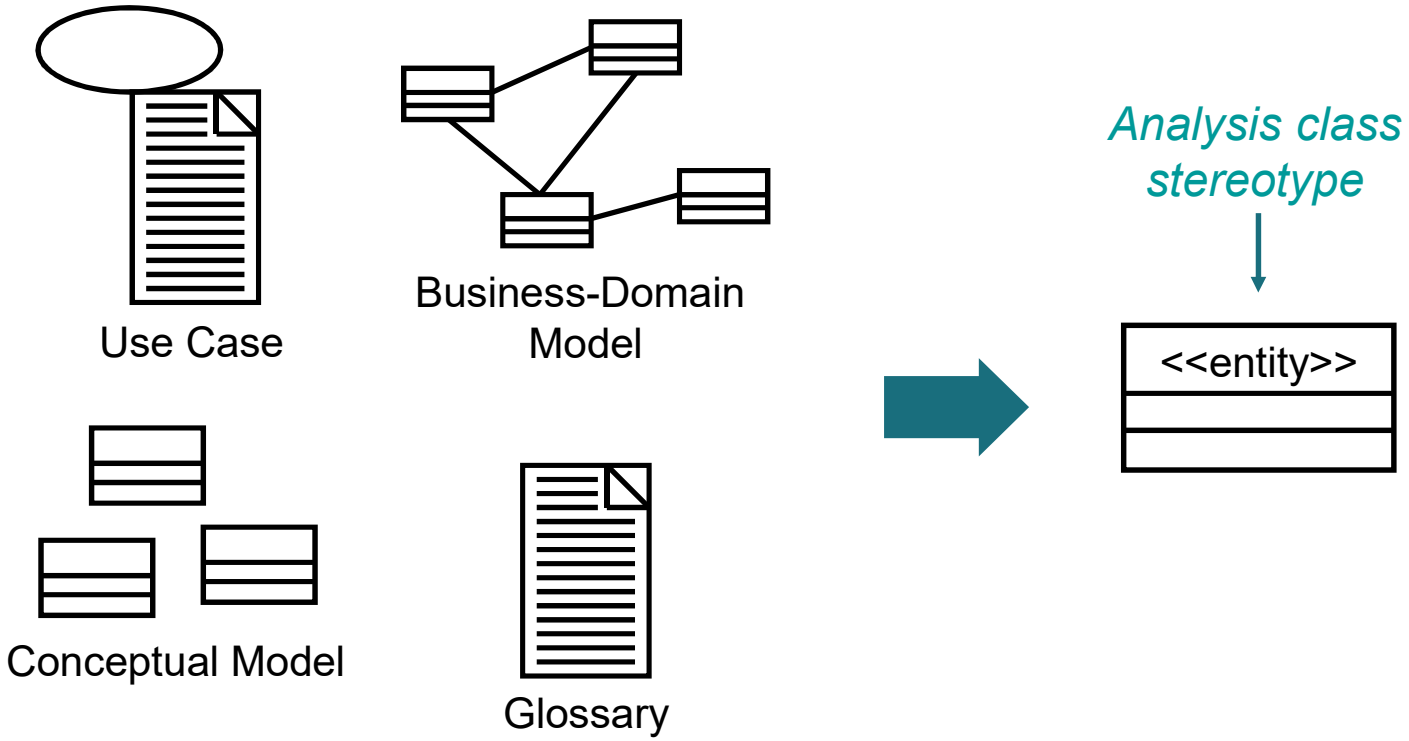
Find Analysis Classes

- The complete behavior of a use case has to be distributed to analysis classes



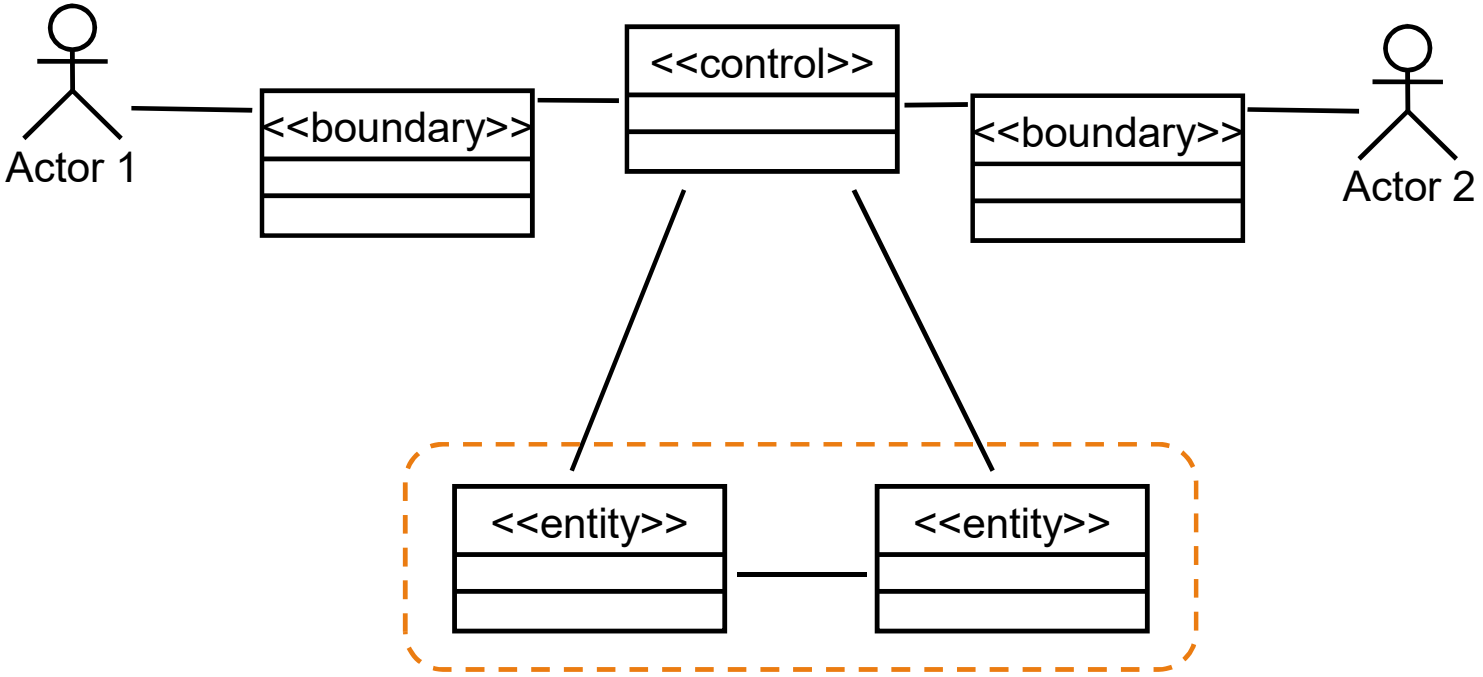
What Is an Entity Class?

- Key abstractions of the system



Environment independent.

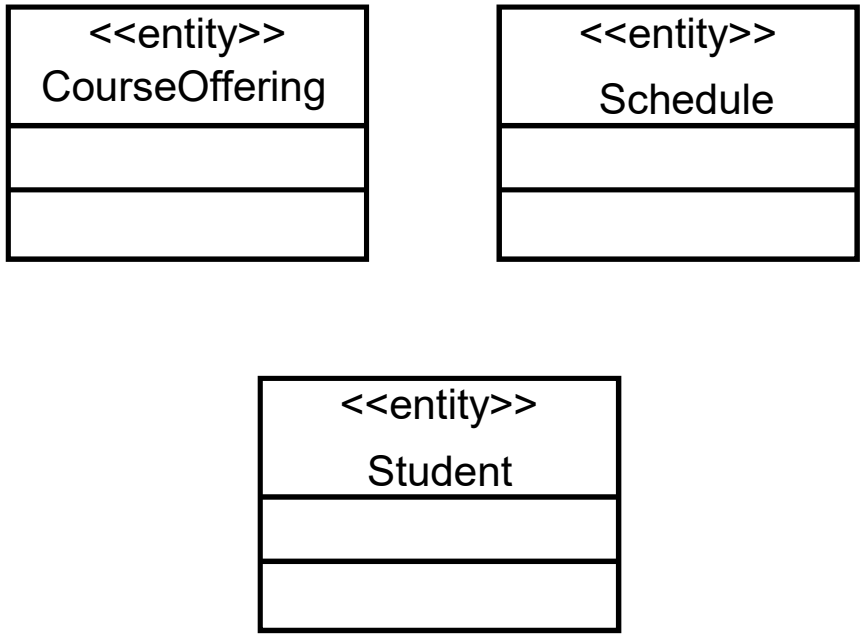
The Role of an Entity Class



Store and manage information in the system.

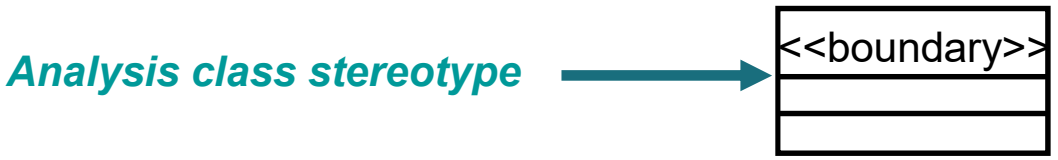
Example: Candidate Entity Classes

- Register for Courses (Create Schedule)



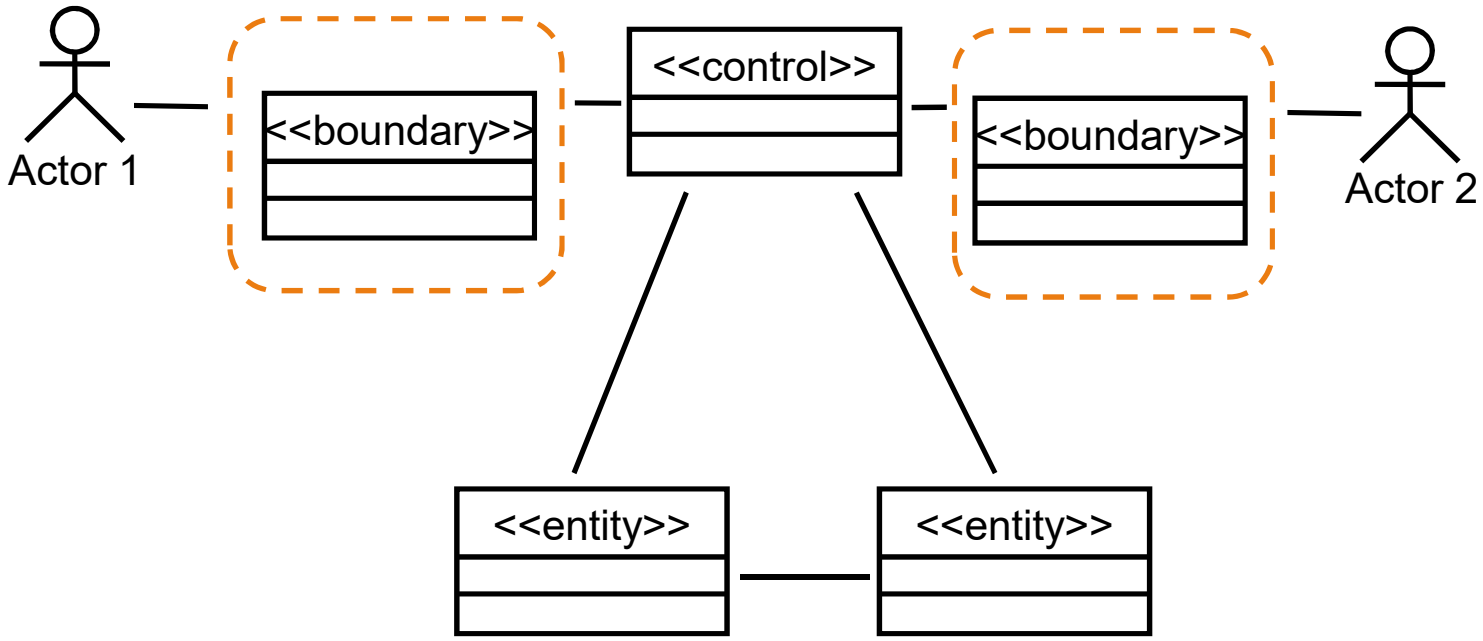
What Is a Boundary Class?

- Intermediates between the interface and something outside the system
- Several Types
  - User interface classes
  - System interface classes
  - Device interface classes
- *One boundary class per actor/use-case pair*



Environment dependent.

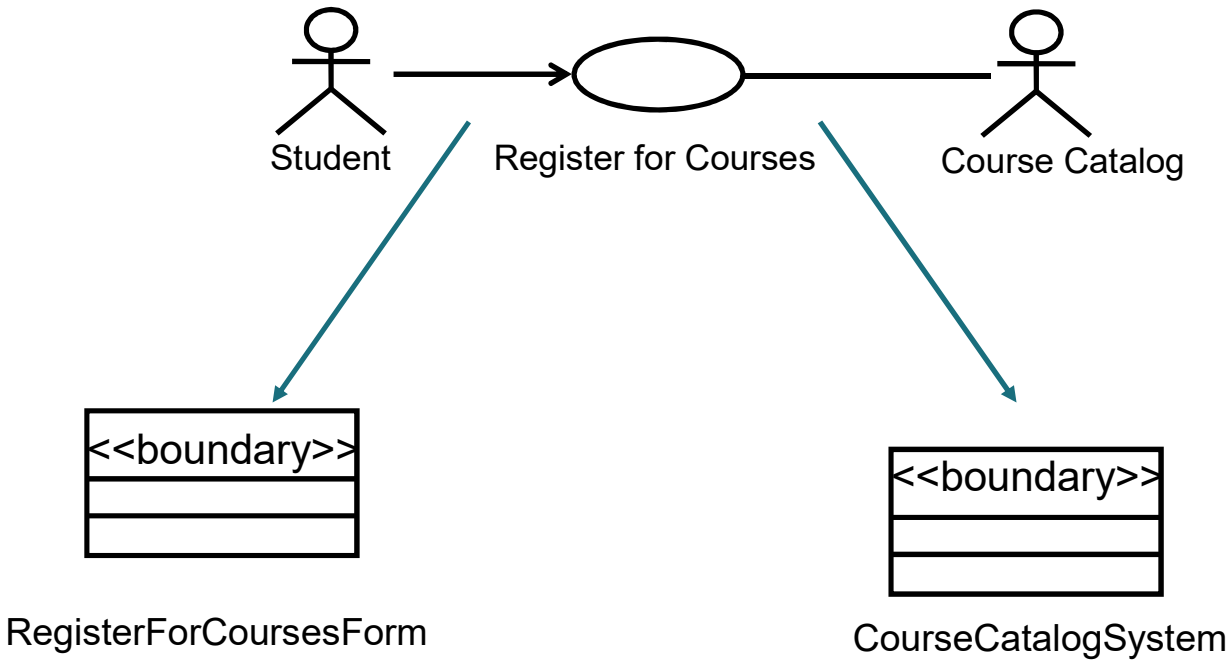
The Role of a Boundary Class



Model interaction between the system and its environment.

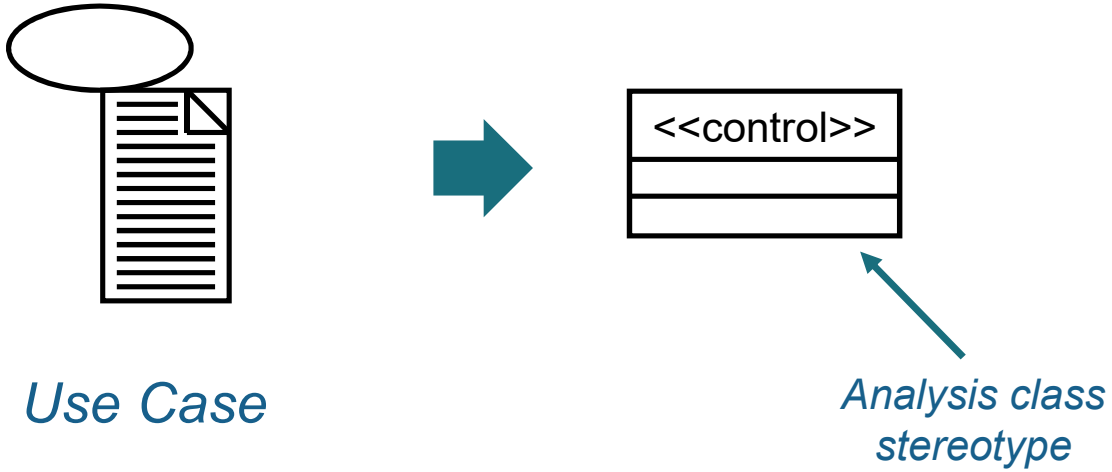
Example: Finding Boundary Classes

- One boundary class per actor/use case pair



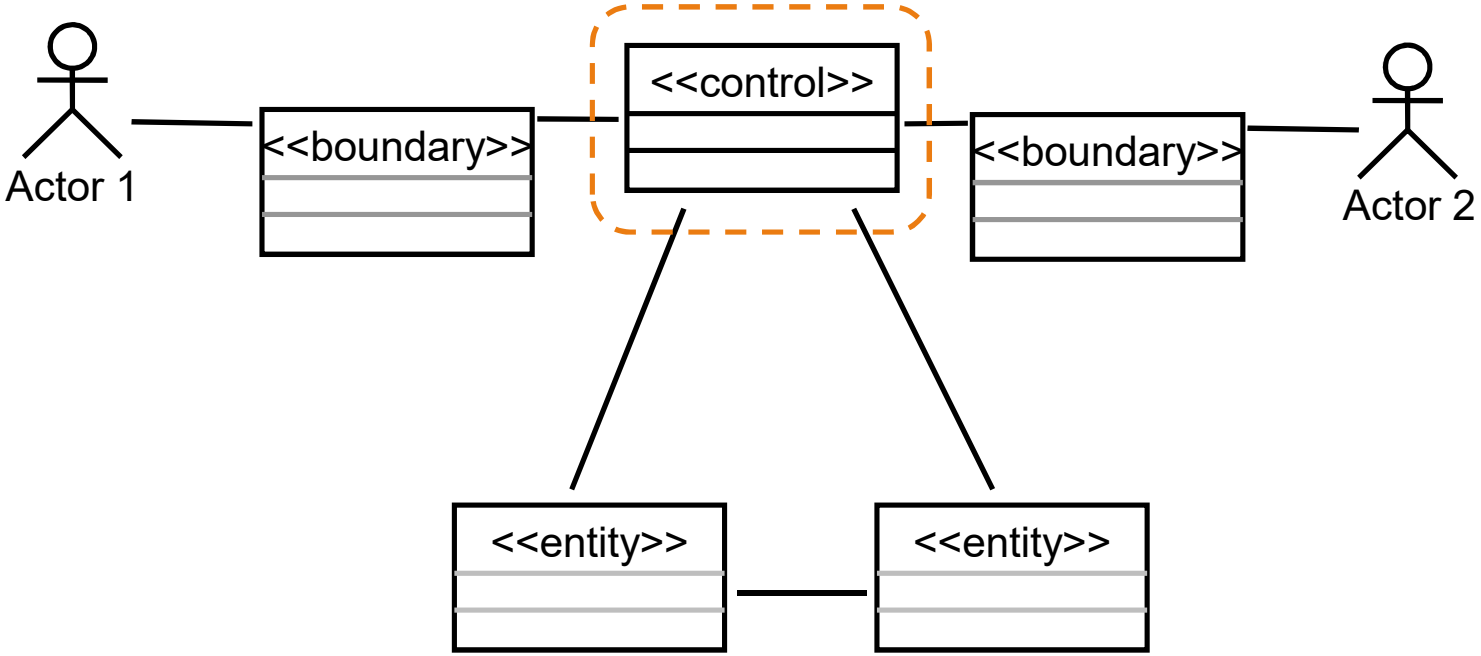
What Is a Control Class?

- Use-case behavior coordinator
  - More complex use cases generally require one or more control cases



Use-case dependent. Environment independent.

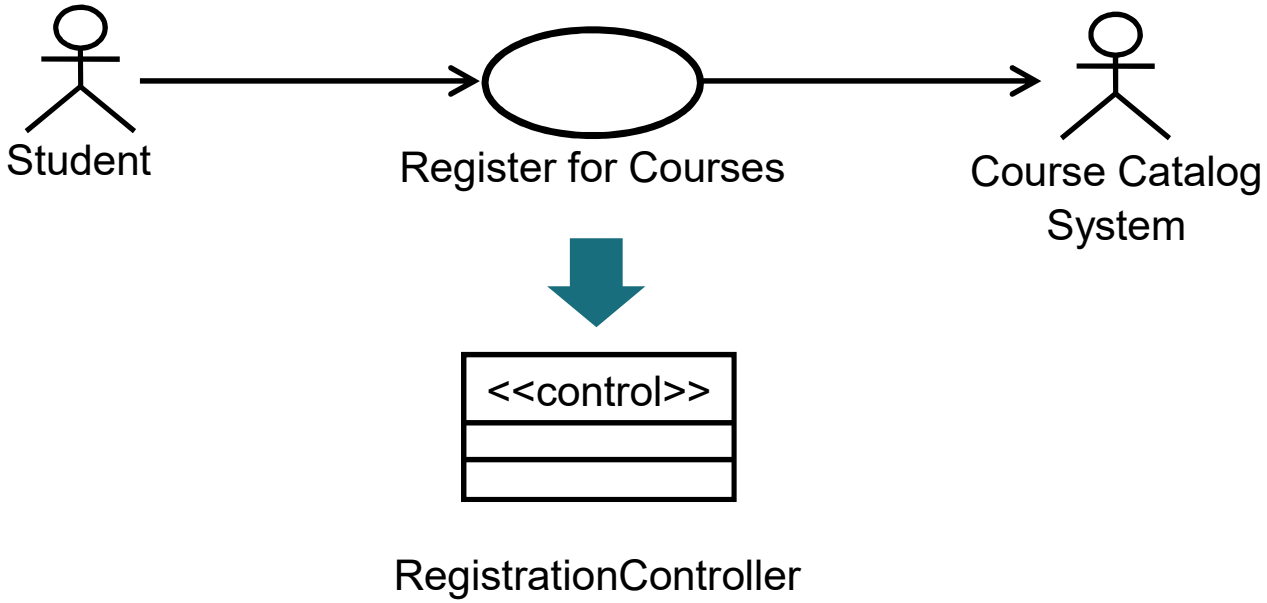
The Role of a Control Class



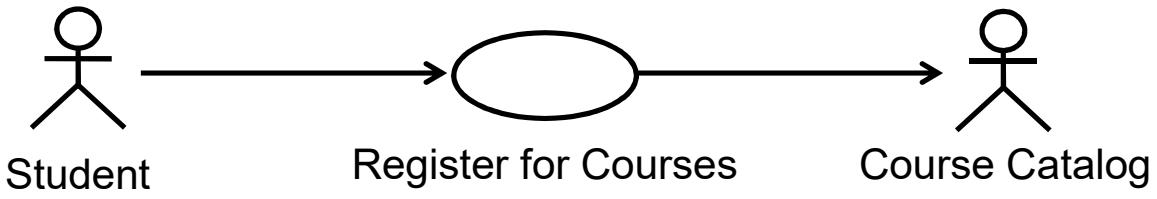
Coordinate the use-case behavior.

Example: Finding Control Classes

- In general, identify one control class per use case.
  - As analysis continues, a complex use case's control class may evolve into more than one class

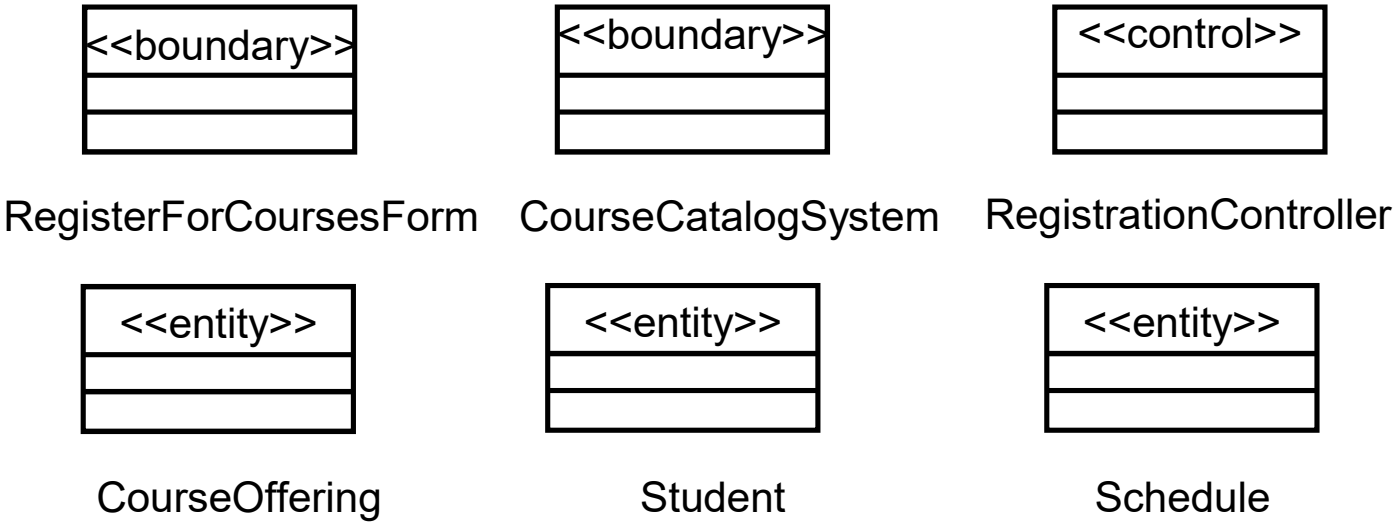


Example: Summary: Analysis Classes



Use-Case Model

Analysis Model



面向对象分析的步骤

1. 用例建模

- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性，画状态图

3. 用例分析

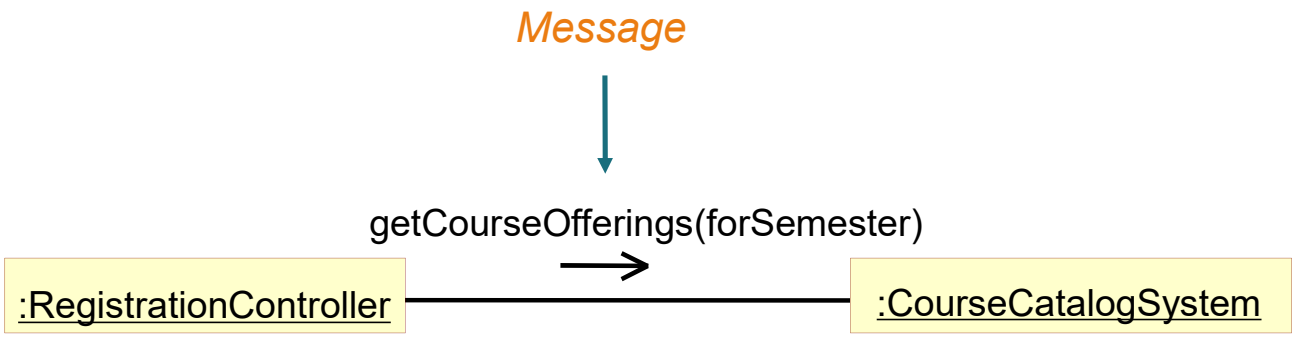
- 3.1 识别出用例实现
- 3.2 针对每个用例实现：
  - 识别出分析类
  - **建立时序图，生成通信图**
  - 对照通信图建立类图，完善每个分析类的属性和操作

Objects Need to Collaborate

- Objects are useless unless they can collaborate to solve a problem.
  - Each object is responsible for its own behavior and status.
  - No one object can carry out every responsibility on its own.
- How do objects interact with each other?
  - They interact through messages.

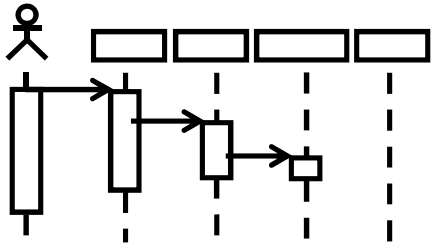
Objects Interact with Messages

- A message shows how one object asks another object to perform some activity.

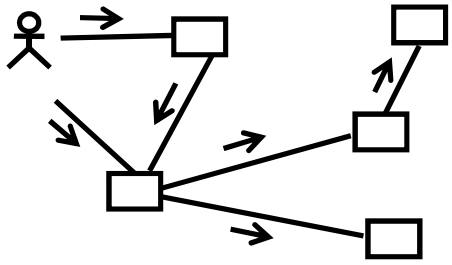


What is an Interaction Diagram?

- Generic term that applies to several diagrams that emphasize object interactions
  - Sequence Diagram
    - Time oriented view of object interaction
  - Communication Diagram
    - Structural view of messaging objects



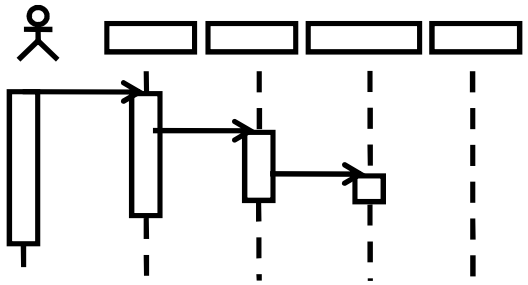
Sequence Diagrams



Communication Diagrams

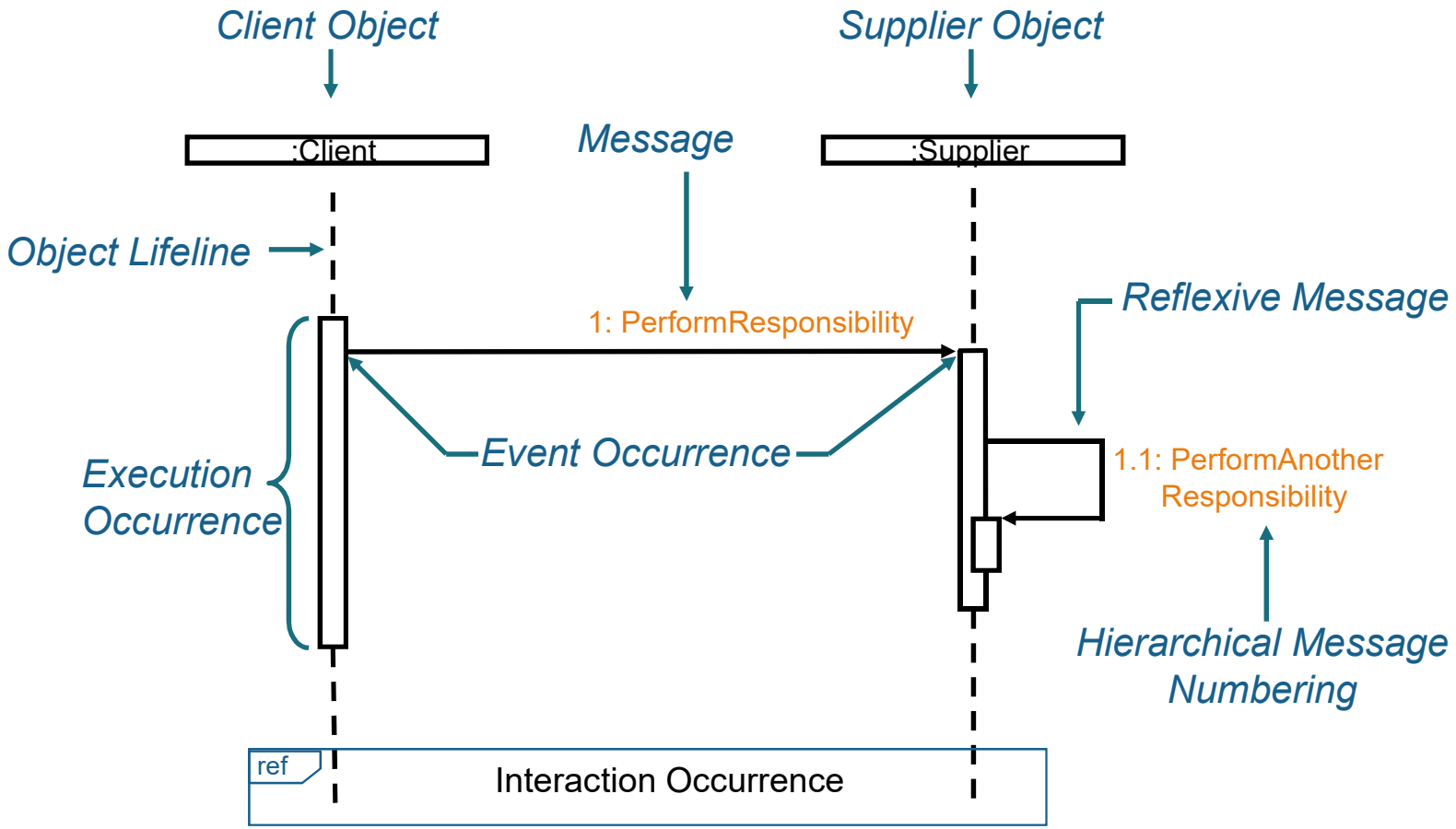
Create Sequence Diagrams

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.
- The diagram shows:
  - The objects participating in the interaction.
  - The sequence of messages exchanged.



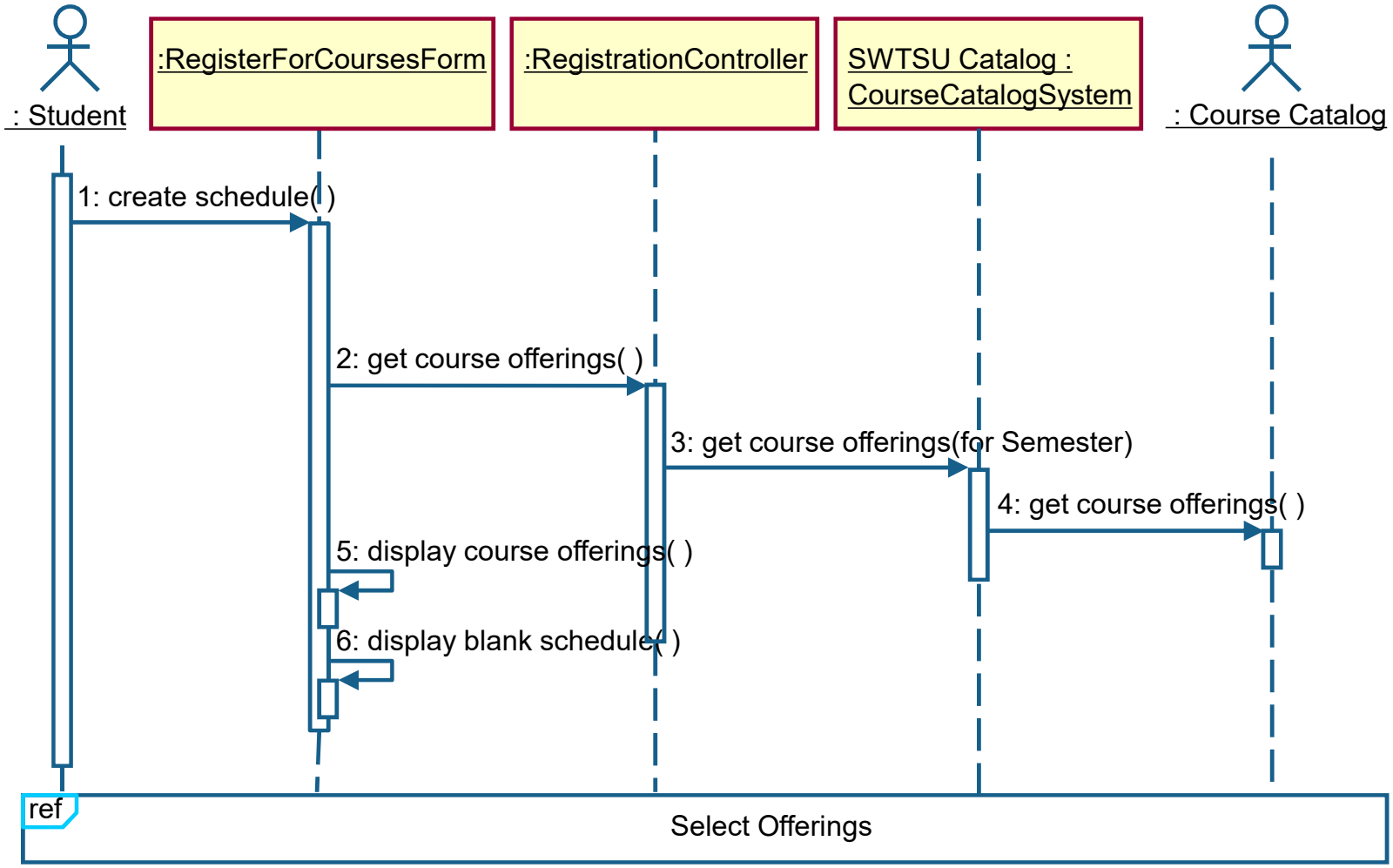
Sequence Diagram

The Anatomy of Sequence Diagrams



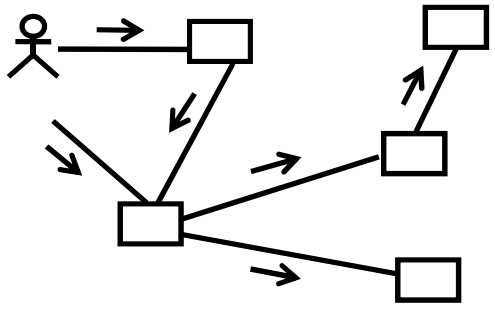
Example: Sequence Diagram

先分工，再协作！



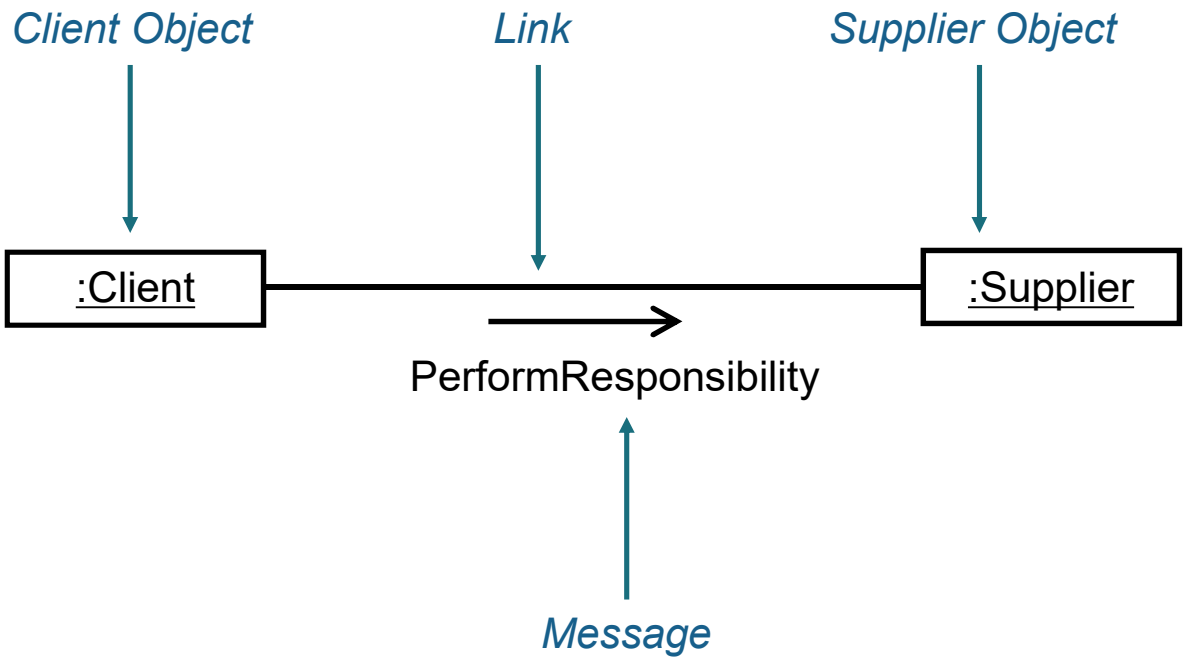
Generate Communication Diagrams

- A communication diagram emphasizes the organization of the objects that participate in an interaction.
- The communication diagram shows:
  - The objects participating in the interaction.
  - Links between the objects.
  - Messages passed between the objects.

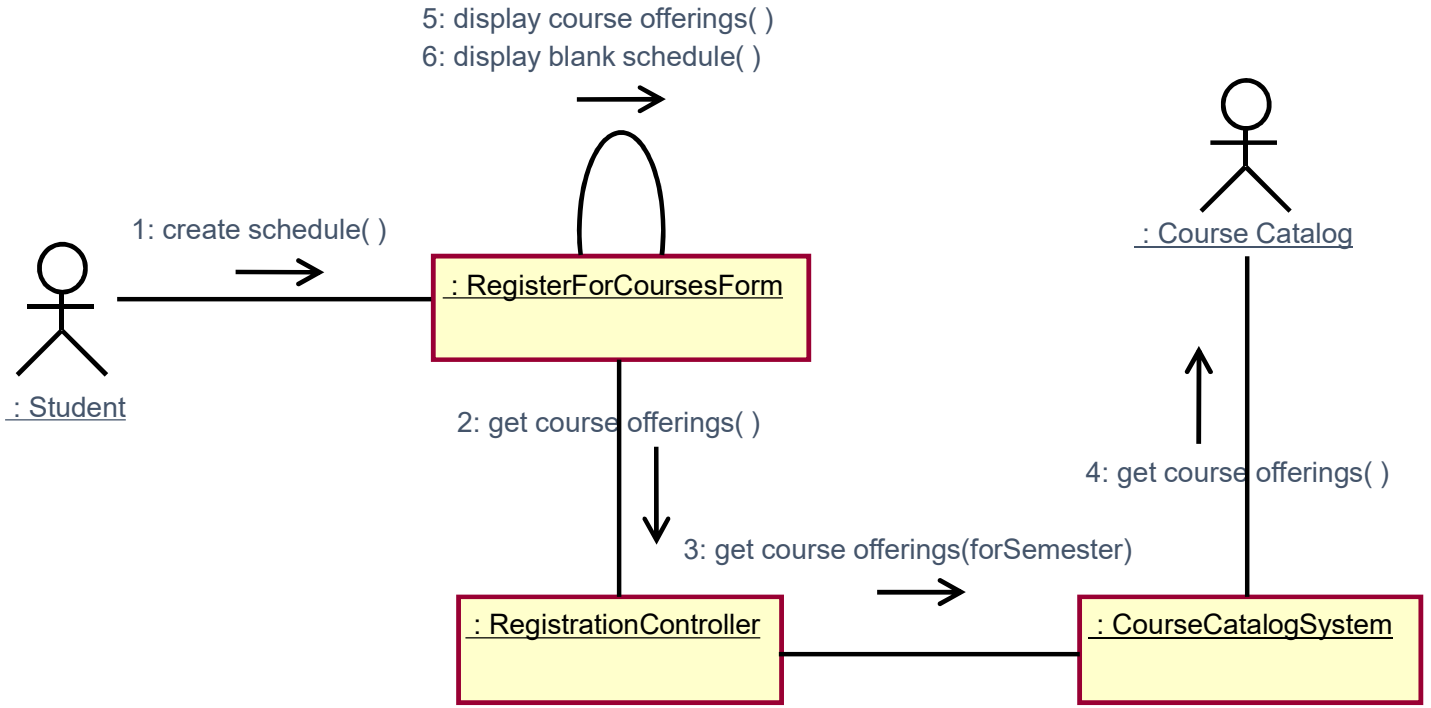


Communication Diagrams

The Anatomy of Communication Diagrams



Example: Communication Diagram



面向对象分析的步骤

1. 用例建模

- 1.1识别actor和use case，画Use-Case图
- 1.2 编写Use-Case Spec.
- 1.3 优化Use-Case图的结构

2. 建立概念模型

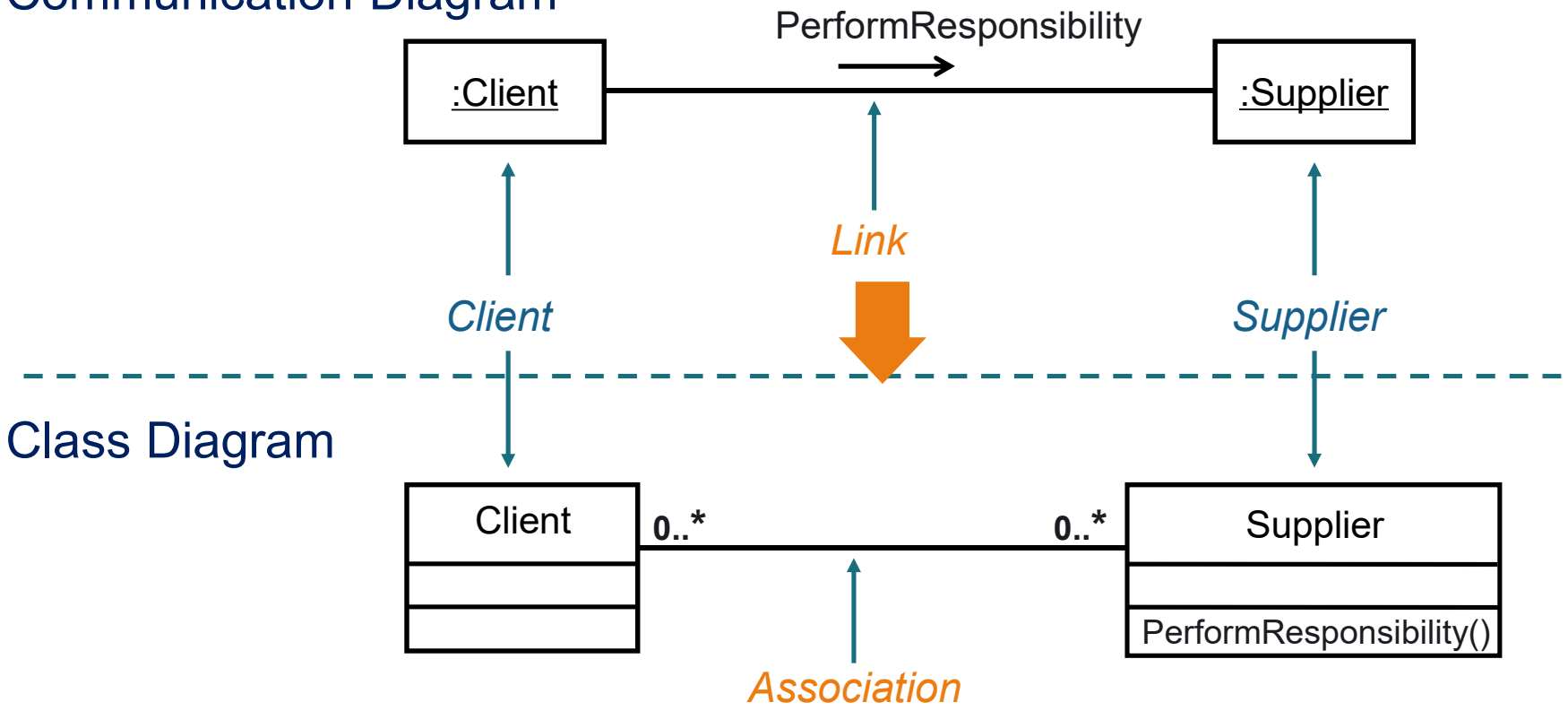
- 2.1 识别Conceptual Class
- 2.2 建立Conceptual Class之间的关系
- 2.3 增加 Conceptual Class的属性，画状态图

3. 用例分析

- 3.1 识别出用例实现
- 3.2 针对每个用例实现：
  - 识别出分析类
  - 建立时序图，生成通信图
  - **对照通信图建立类图，完善每个分析类的属性和操作**

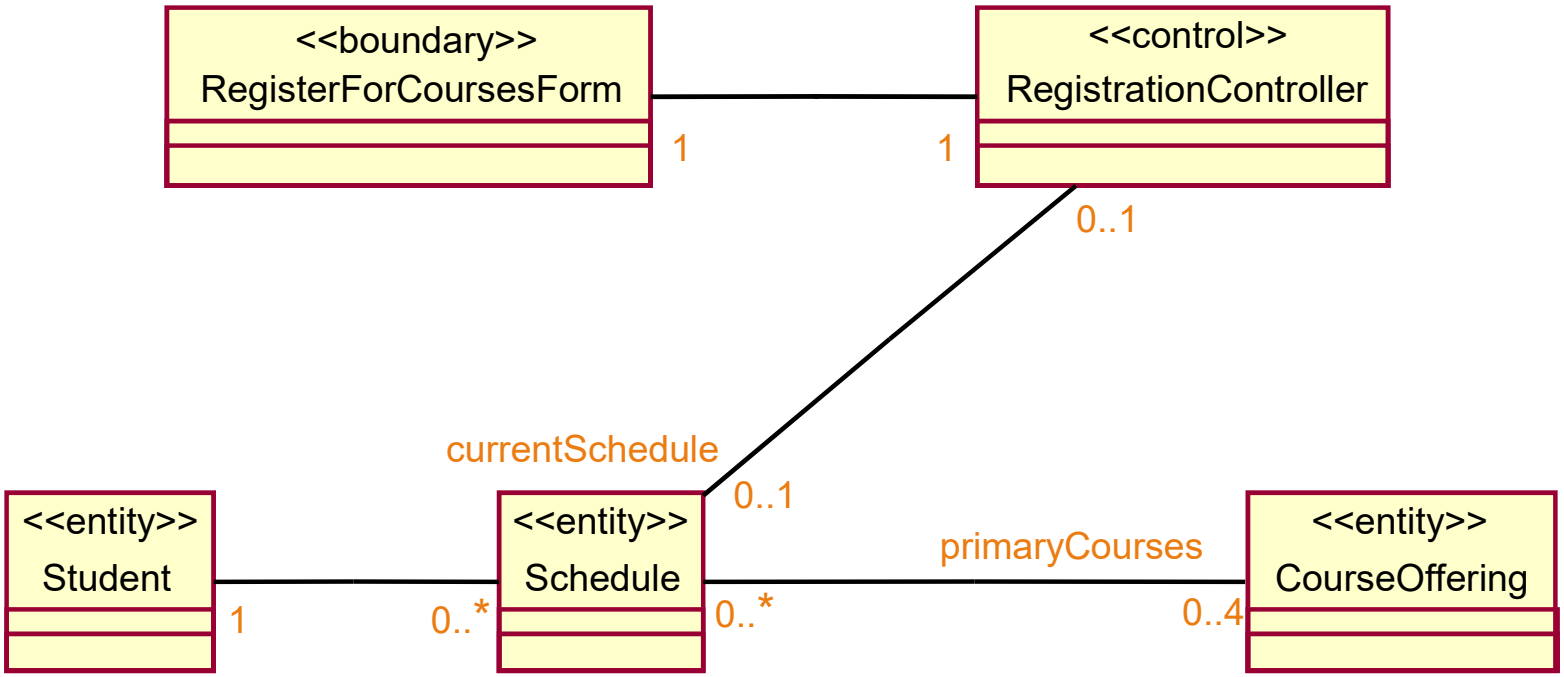
Finding Relationships and Operations

Communication Diagram

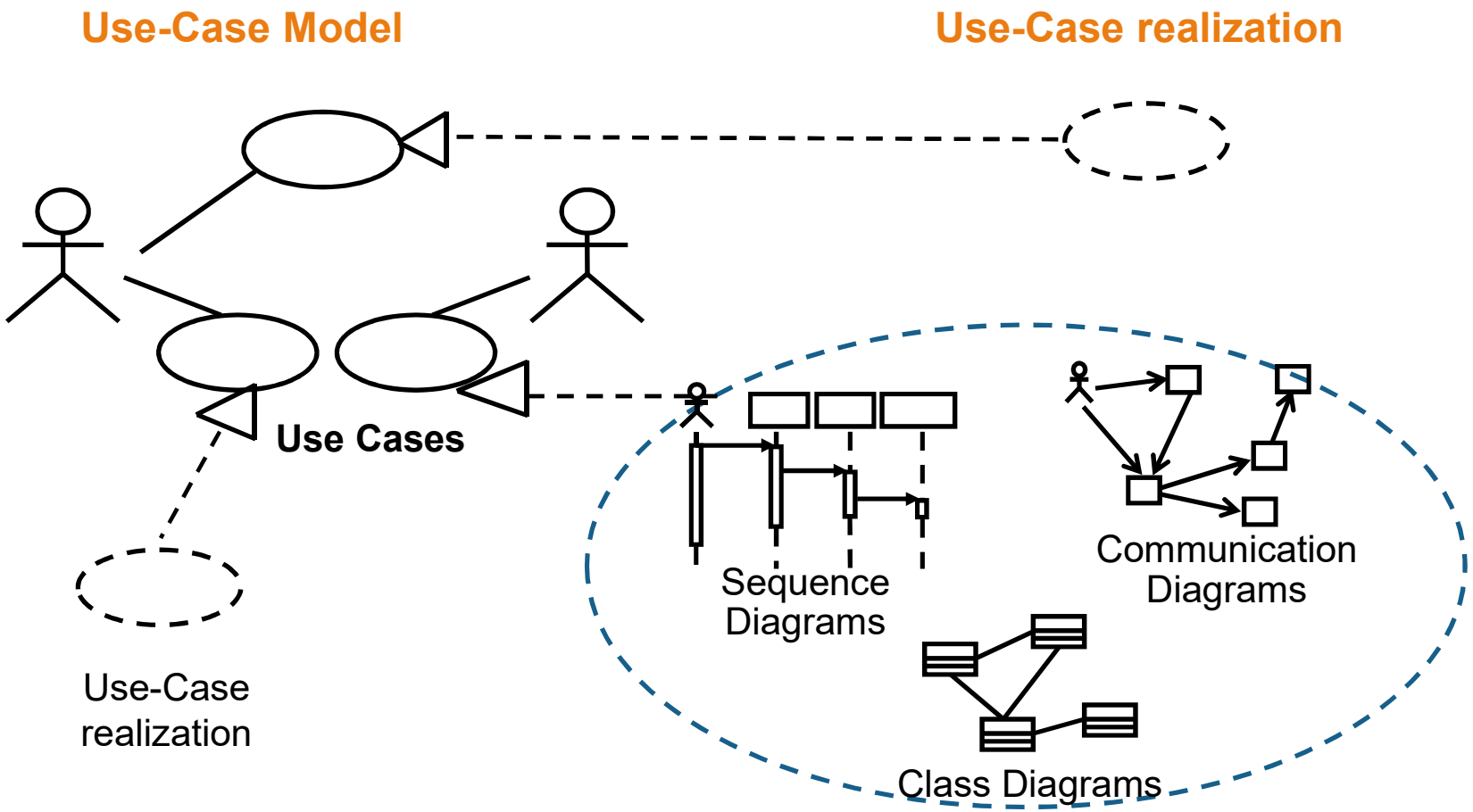


Relationship for every link!

Example: VOPC: Finding Relationships



完成用例分析


Use-Case Model
Use-Case realization
Use Cases
Use-Case
realization
Sequence
Diagrams
Communication
Diagrams
Class Diagrams