502045
Software Engineering

Chapter 05

Lesson 05: UML

Topics covered

- ✧ UML
  - What is the UML?
  - Ways of Using the UML
- ✧ Common Diagrams
  - Use case diagram
  - Sequence diagram
  - Activity diagram
  - Class diagram
- ✧ Other diagrams

What Is the UML?

- ✧ The Unified Modeling Language (UML) is a family of graphical notations that help in describing and designing software systems, particularly software systems built using the object-oriented (OO) style
- ✧ The UML is a relatively open standard, controlled by the Object Management Group (OMG), an open consortium of companies
- ✧ The UML was born out of the unification[thống-nhất] of the many object-oriented graphical modeling languages that thrived[phát-triển-mạnh] in the late 1980s and early 1990s (its appearance in 1997)

What Is the UML? (Extra)

✧ Phân tích và thiết kế hệ thống theo hqớng truyền thống

- System development life Cycle (SDLC) hoặc Structured Systems Analysis and Design (SSAD)
- Mô hình thác nước, ...
- Hai vấn đề chính trong việc phát triển hệ thống thông tin đó là các **xử lý** và **dữ liệu** được xây dựng độc lập với phương pháp này. (Quy trình: DFD Dữ liệu: ERD)

✧ Phân tích thiết kế hệ thống hướng đối tượng (OOAD)

- Dựa trên: **abstraction, encapsulation, modularity, hierarchy**
- Một hệ thống hướng đối tượng sẽ có một số đối tượng, mỗi đối tượng này sẽ kết hợp với các đối tượng khác để hoàn thành một nhiệm vụ.

Ways of Using the UML

- ✧ At the heart of the role of the UML in software development are the different ways in which people want to use it.
- ✧ Characterization of the three modes in which people use the UML: sketch, blueprint, and programming language .
- ✧ By far the most common of the three is UML as sketch. In this usage, developers use the UML to help communicate some aspects of a system. As with blueprints, you can use sketches in a forward-engineering or reverse-engineering direction

UML as Sketches **(Self study)** (Forward engineering)

- The essence of sketching is **selectivity**. With forward sketching, you rough out some issues in code you are about to write, usually **discussing them with a group of people on your team**
- Your aim is to use the sketches to help communicate ideas and alternatives about what you're about to do . You don't talk about all the code you are going to work on, only important issues that you want to run past your colleagues first or sections of the design that you want to visualize before you begin programming
- Sessions like this can be very short

UML as Blueprint (Self Study) (Forward engineering)

- ✧ **In contrast, UML as blueprint is about <u>completeness</u>**
- ✧ **The idea is that blueprints are developed by a designer whose job is to build a detailed design for a programmer to code up .**
- ✧ That design should be sufficiently complete in that all design decisions are laid out, and the programmer should be able to follow it as a pretty straightforward activity that requires little thought
- ✧ The designer may be the same person as the programmer, but usually the designer is a more senior developer who designs for a team of programmers

UML as Programming Language

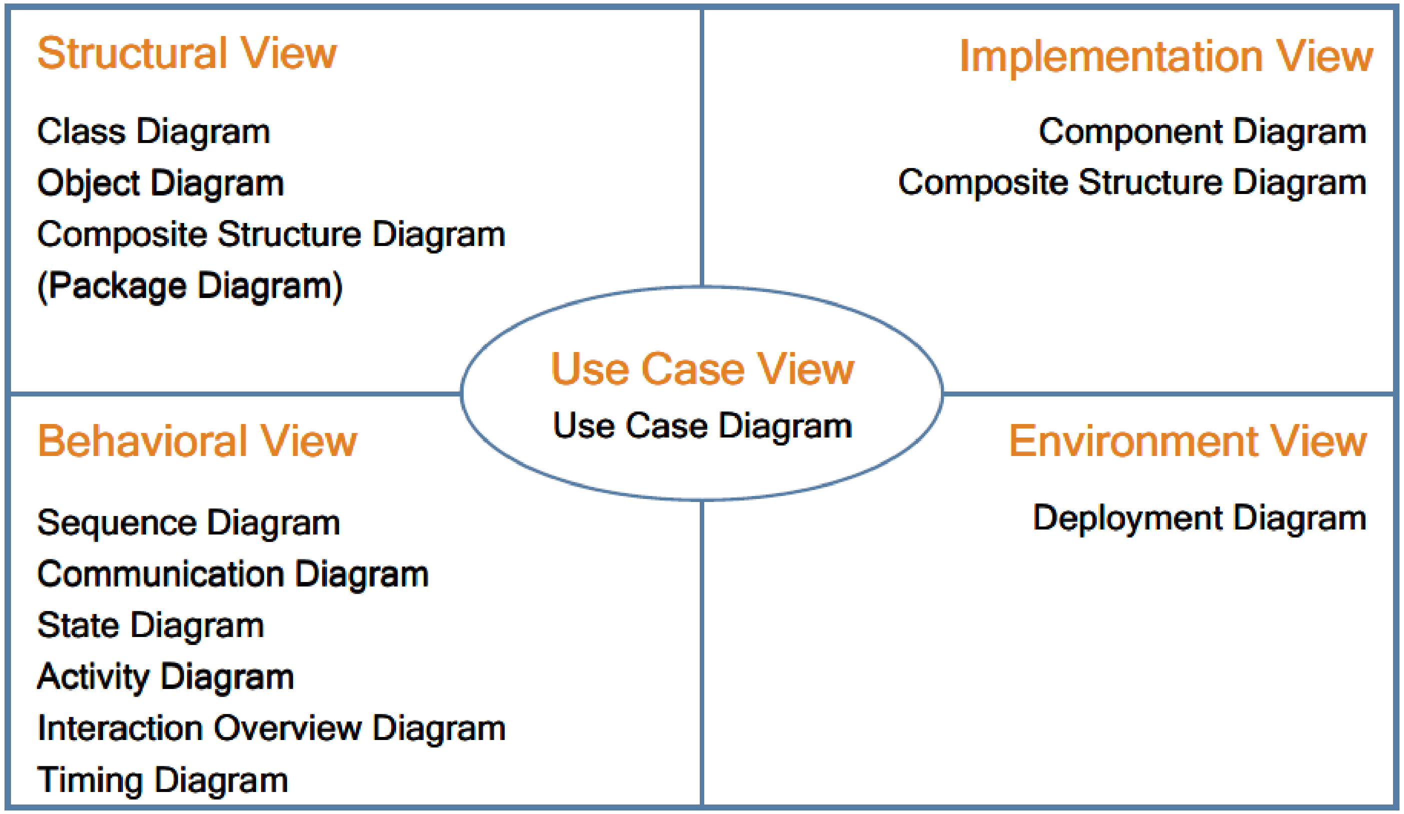
✧ In this environment, developers draw UML diagrams that are compiled directly to executable code, and the UML becomes the source code.

✧ Obviously, this usage of UML demands particularly sophisticated tooling

✧ One of the interesting questions around the UML as programming language is how to model behavioral logic. UML 2 offers three ways of behavioral modeling : **interaction diagrams, state diagrams, and activity diagrams.**

UML Diagrams

- UML defines 13 diagrams that describe 4+1 architectural views



UML (Extra)

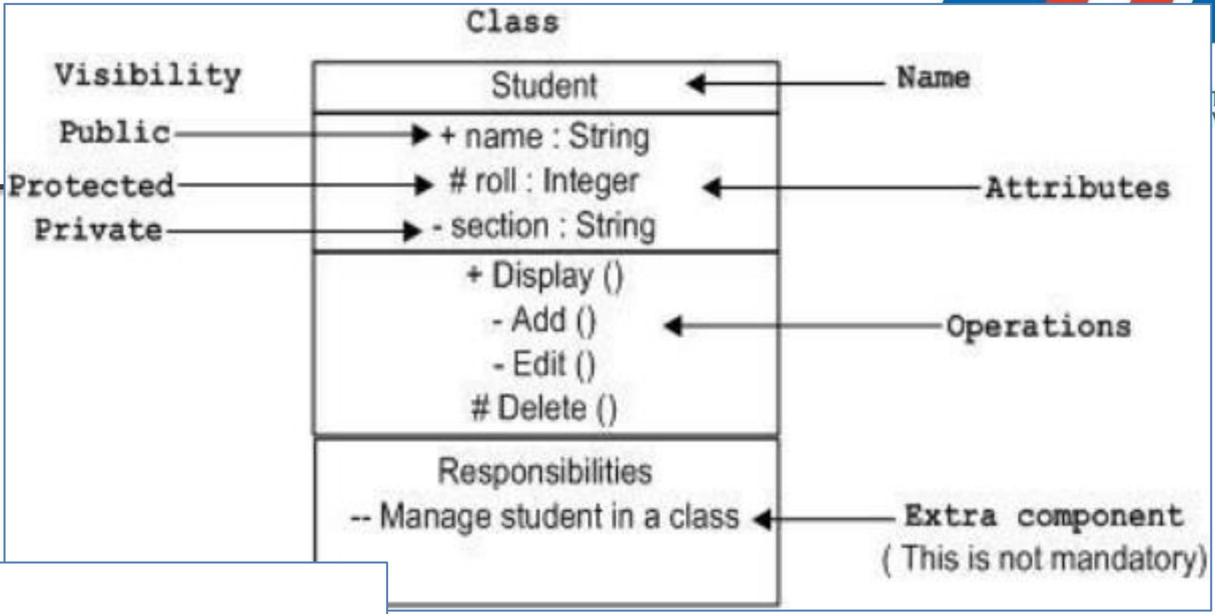
✧ Các khối của UML

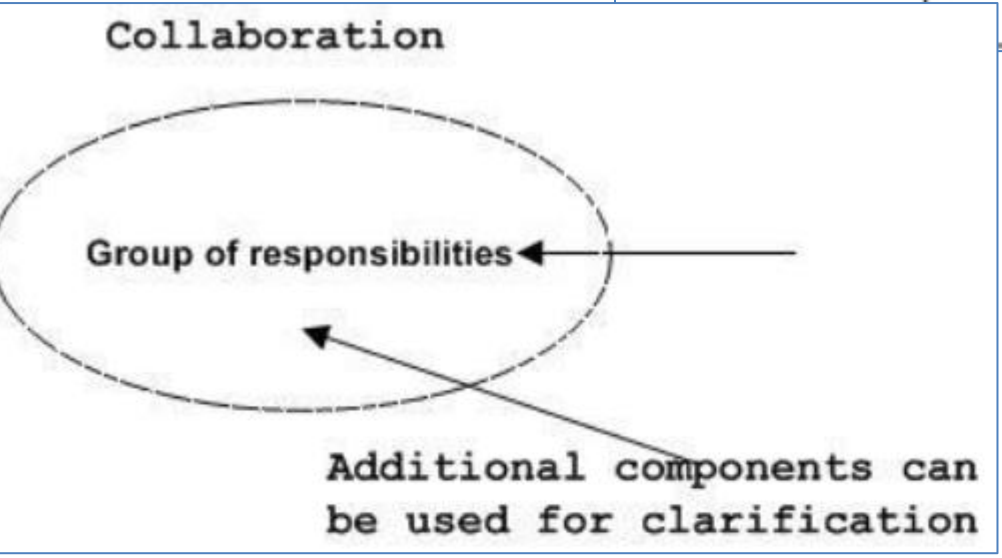
- Things
- Relationships
- Diagrams

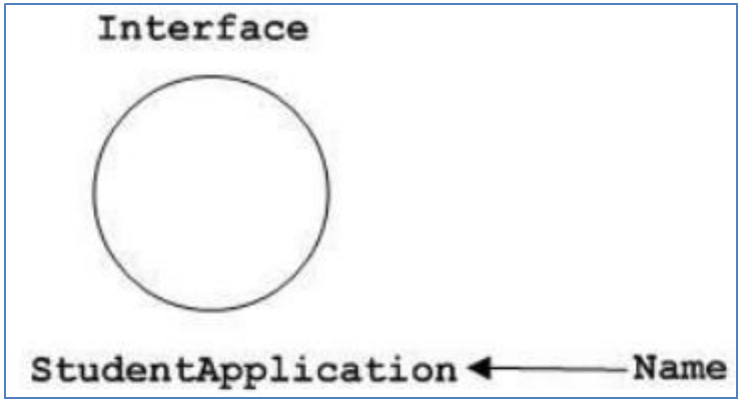
UML (Extra)

✧ Thing

- Class
- Interface
- Collaboration



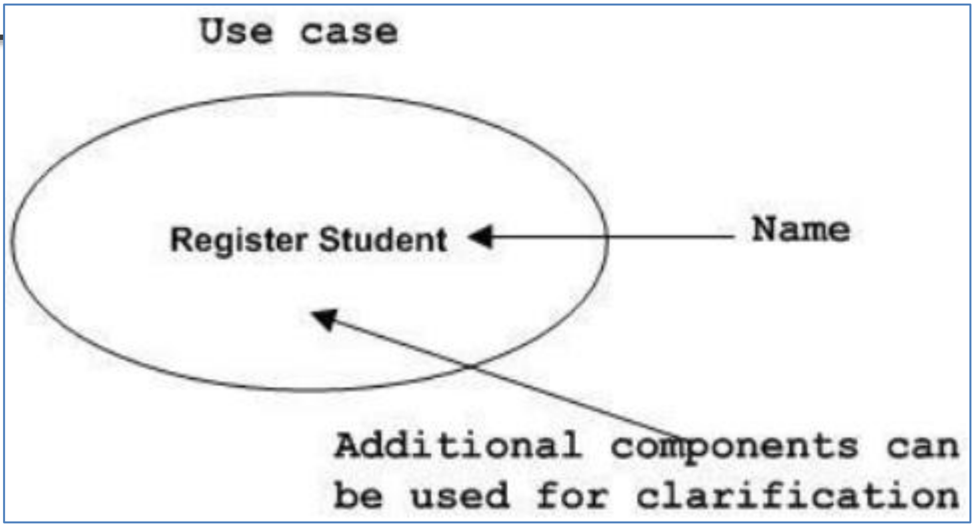

Collaboration
Group of responsibilities
Additional components can be used for clarification

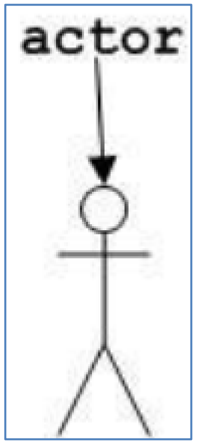


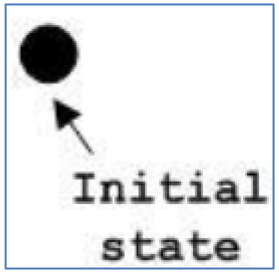
UML (Extra)

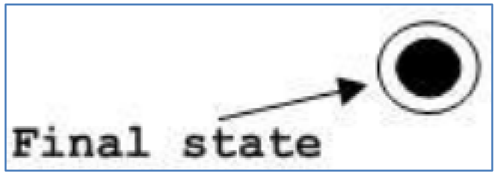
✧ Thing

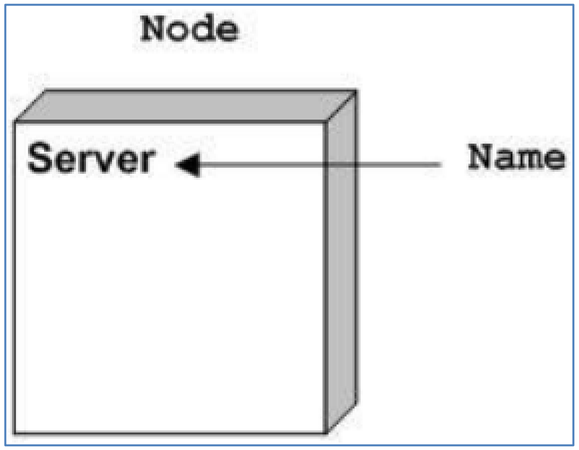
- Use case
- Actor
- Ký hiệu bắt đầu, kết thúc
- Component
- Node

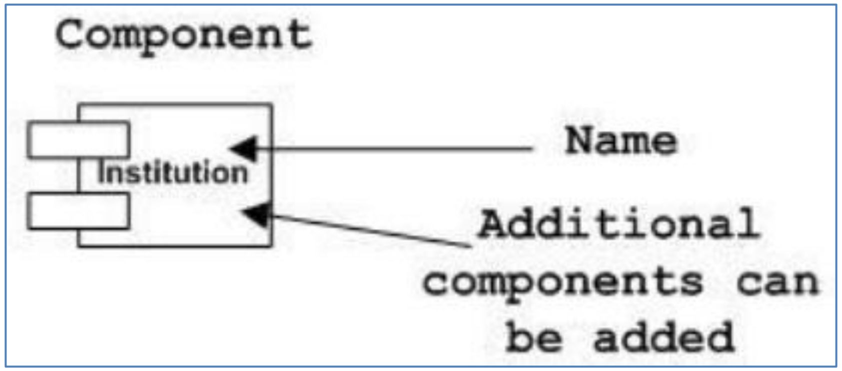








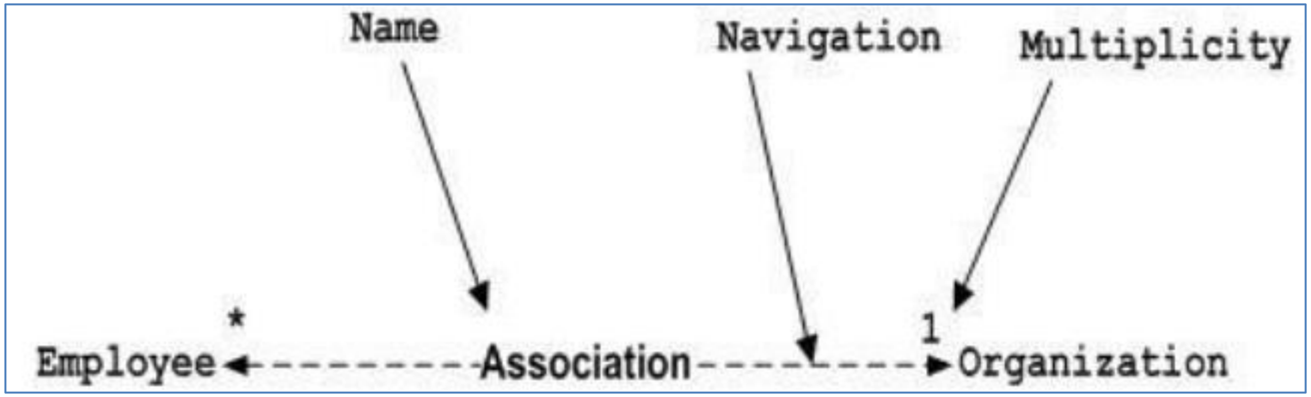


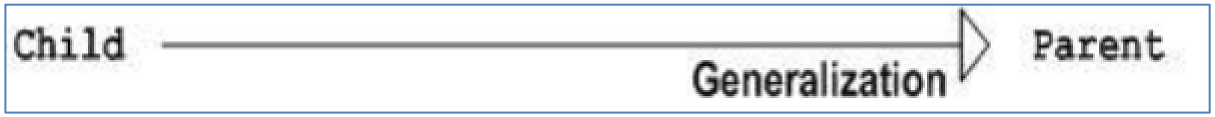


UML (Extra)

✧ Relationship

- Dependency
- Association
- Generalization/Realization





Topics covered

- ✧ Introduction
  - What is the UML?
  - Ways of using the UML
- ✧ Common diagrams
  - **Use case diagram**
  - Sequence diagram
  - Activity diagram
  - Class diagram
- ✧ Other diagrams

Use Cases

- Use cases are a technique for capturing the **functional requirements** of a System. Use cases work by describing the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used
- **A scenario**: "***The customer browses the catalog and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale. The system checks the authorization an the credit card and confirms the sale both immediately and with a follow-up e-mail***"

Use Cases

- ✧ This scenario is one thing that can happen. However, the credit card authorization might fail, and this would be a separate scenario.
- ✧ In another case, you may have a regular customer for whom you don't need to capture the shipping and credit card information, and this is a third scenario
- ✧ In all these three scenarios, the user has the same goal : to buy a product .The user doesn't always succeed, but the goal remains. This user goal is the key to use cases : A use case is a set of scenarios tied together by a common user goal

Actor

- Each use case has a primary actor, which calls on the system to deliver a service.
- The primary actor is the actor with the goal the use case is trying to satisfy and is usually, but not always, the Initiator of the use case.
- There may be other actors which the system communicates while carrying out the use case. These are known as secondary actors

Content of a Use Case

- There is no standard way to write the content of a use case, and different formats work well in different cases
- You begin by picking one of the scenarios as the main success scenario.
- You start the body of the use case by writing the main success scenario as a sequence of numbered steps .
- You then take the other scenarios and write them as extensions.

Buy a Product

Goal Level: Sea Level

Main Success Scenario:

1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming e-mail to customer

Extensions:

3a: Customer is regular customer

   .1: System displays current shipping, pricing, and billing information

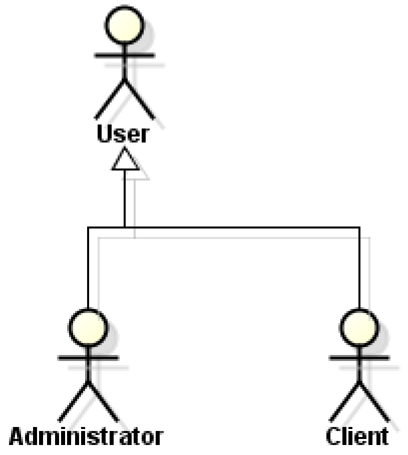
   .2: Customer may accept or override these defaults, returns to MSS at step 6

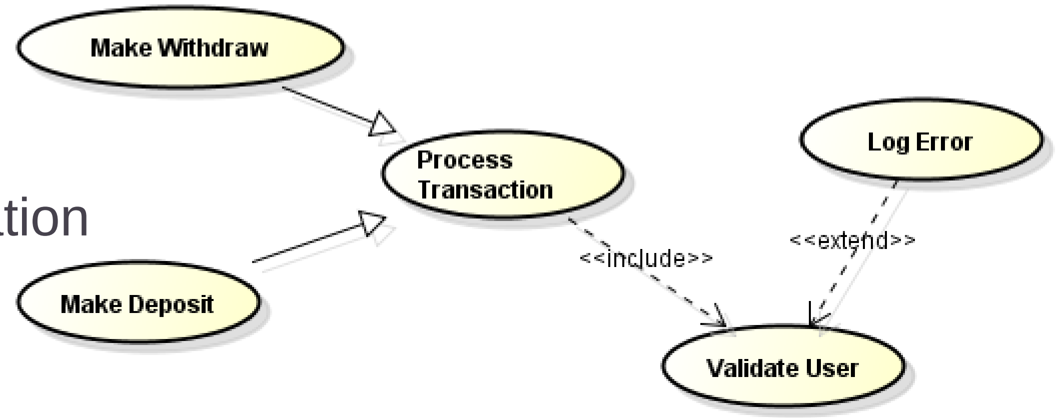
6a: System fails to authorize credit purchase

   .1: Customer may reenter credit card information or may cancel

Use case diagram – Relationship

- ✧ Actor Relationship: Generalization
  - Used to define overlapping roles between actors
- ✧ Actor – Use case relationship
  - Association
- ✧ Use case Relationship
  - Include
  - Extend
  - Generalization/Specification

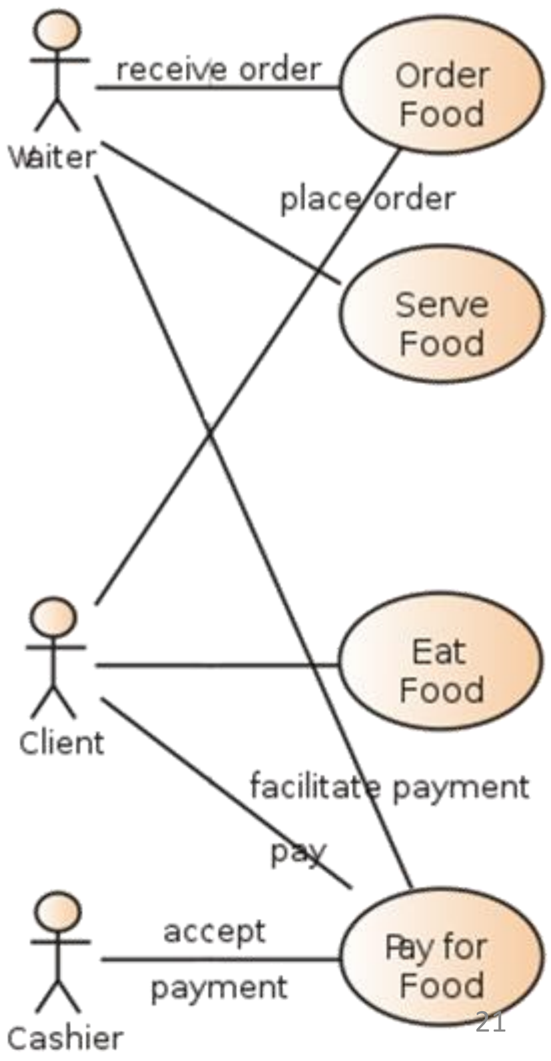




Use case diagram – Association Relationship

✧Association

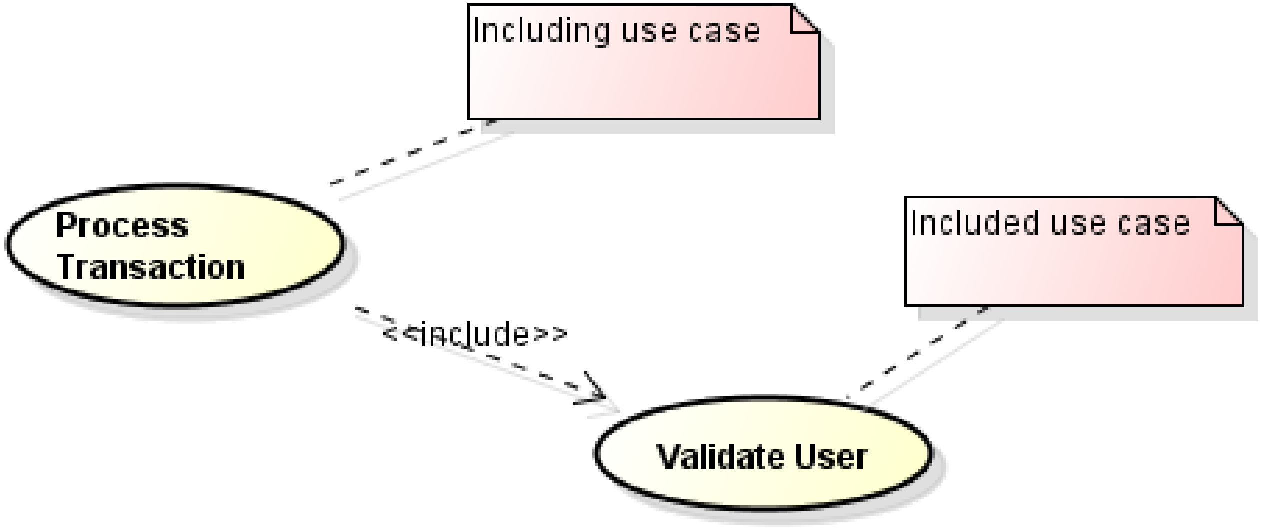
- **Relationship between Actors & use cases**
- Actor is involved in interaction described by use case
- If association line has Arrow head:
  - Indicate direction of invocation, primary actor
  - Indicate control flow (not data flow)



Use case diagram – Include Relationship

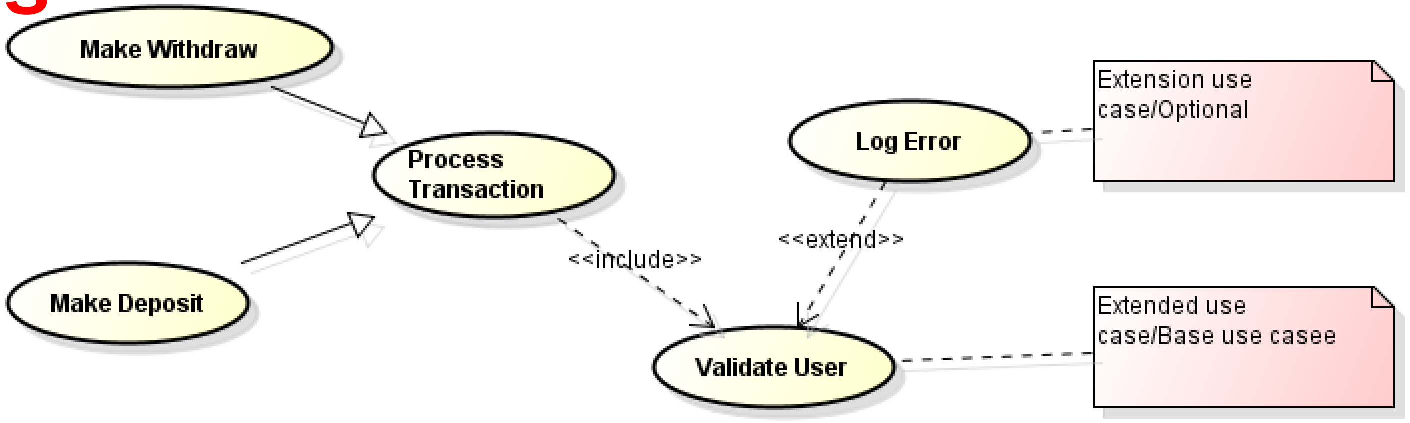
✧**Include**:

- Directed Relationship between 2 use cases



Use case diagram – Extend Relationship

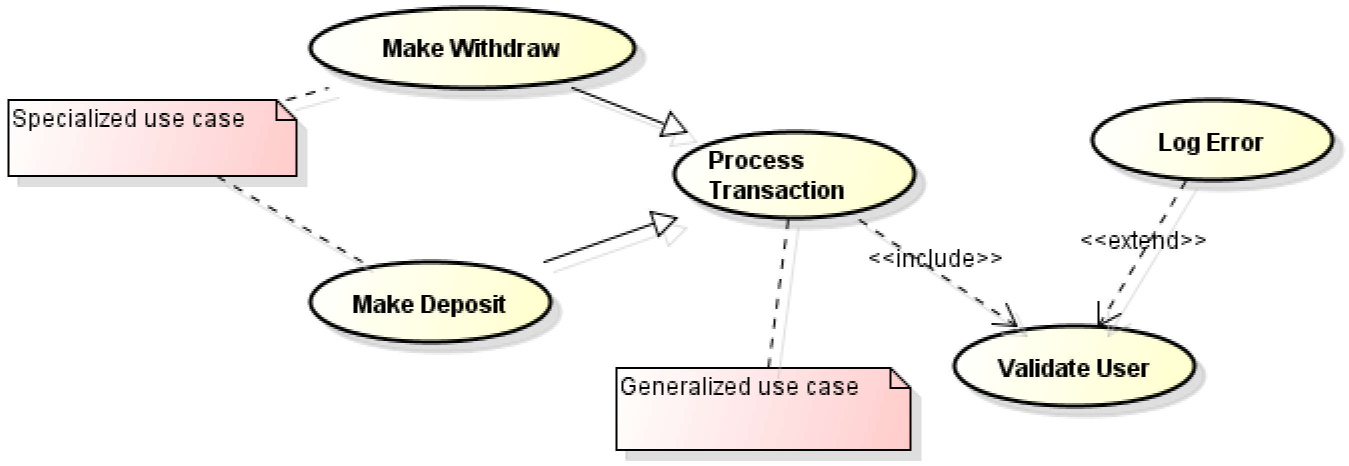
✧Extends



- Behavior of extension use case may be inserted in extended use case ***under some conditions.***

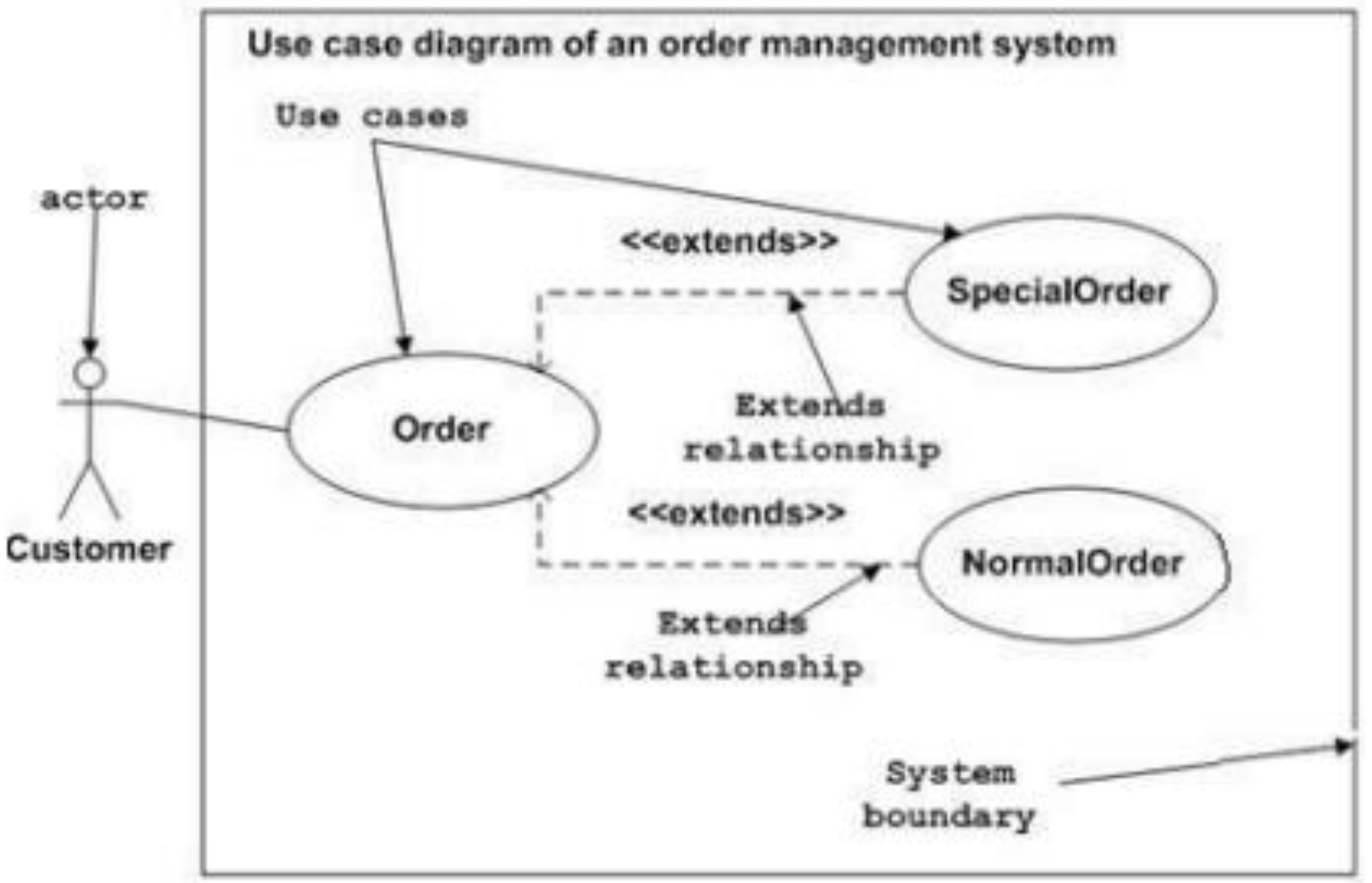
Use case diagram – Generalization Relationship

✧Generalization/Specialization



- Specialized use cases have common behaviors, requirements, constraints, assumptions.

Use Case Diagrams – Other example



Topics covered

- ✧ Introduction
  - What is the UML?
  - Ways of using the UML
- ✧ Common diagrams
  - Use case diagram
  - **Sequence diagram**
  - Activity diagram
  - Class diagram
- ✧ Other diagrams

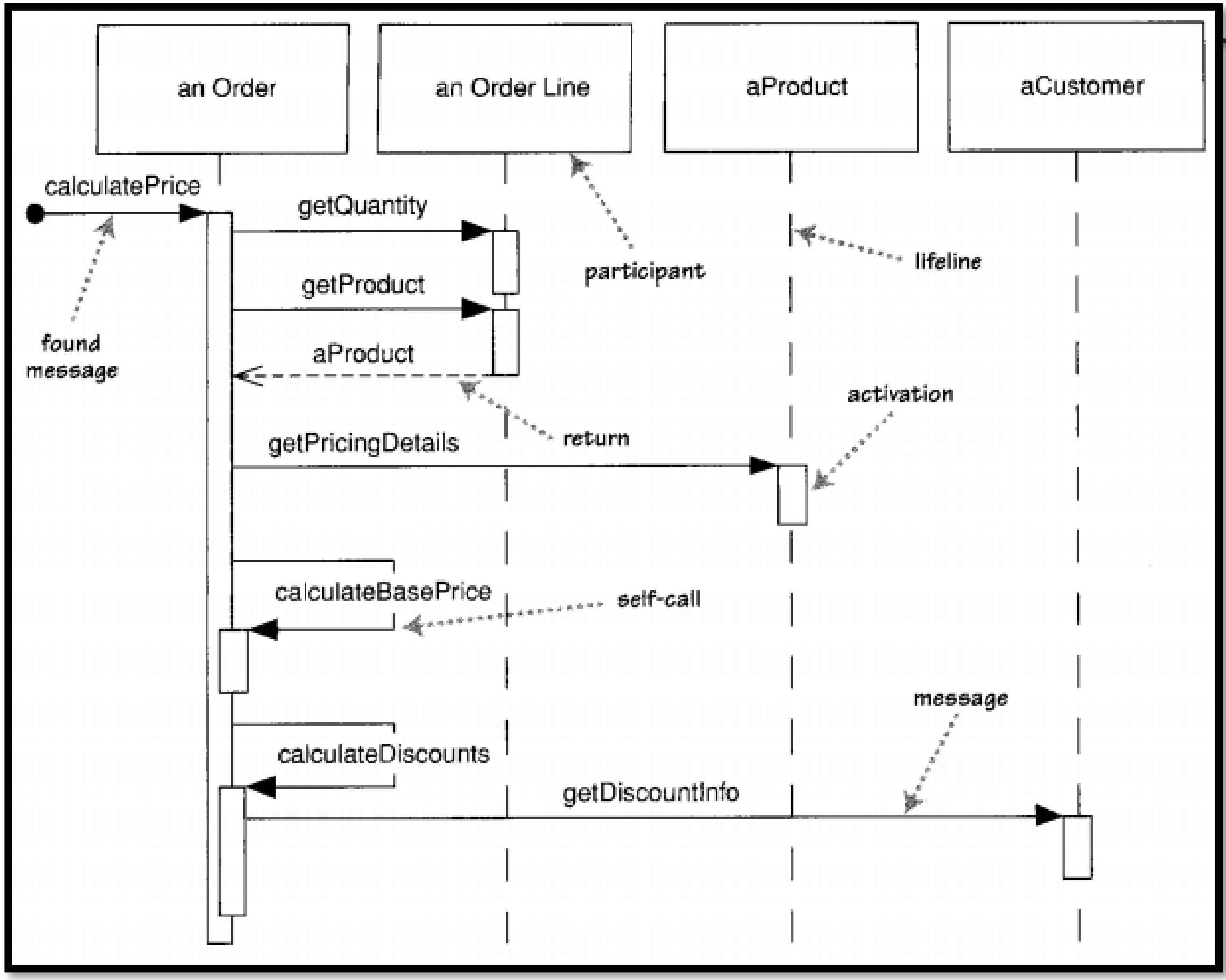
Sequence Diagrams

- ✧ ***Interaction diagrams describe <u>how groups of objects collaborate in some behavior</u>***. The UML defines several forms of interaction diagram, of which the most common is the sequence diagram
- ✧ Typically, a sequence diagram captures the ***<u>behavior of a single scenario</u>***. The diagram shows a number of example objects and the messages that are passed between these objects within the use case

Sequence Diagrams (Case Study)

- ✧ We have an order and are going to invoke a command on it to calculate its price
- ✧ To do that, the order needs to look at all the line items on the order and determine their prices, which are based on the pricing rules of the order line's products. Having done that for all the line items, the order then needs to compute an overall discount, which is based an rules tied to the customer

A sequence diagram for centralized control



A sequence diagram for centralized control

- ✧ Sequence diagrams show the interaction by showing each participant with a lifeline that runs vertically down the page and the ordering of messages by reading down the page
- ✧ You can see that an instance of order sends getQuantity and getProduct messages to the order line. You can also see how we show the order invoking a method an itself and how that method sends getDiscountInfo to an instance of customer .

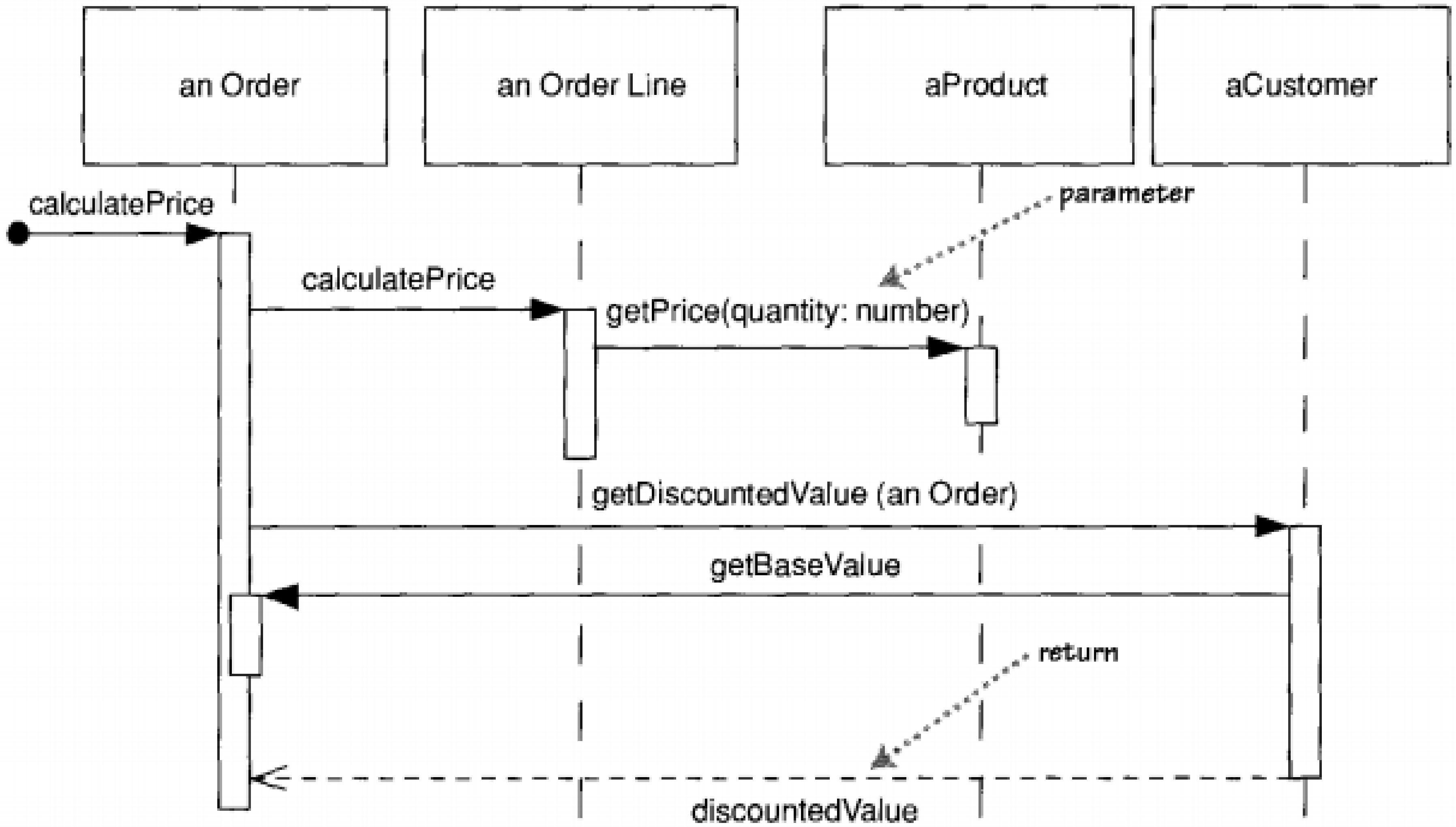
A sequence diagram for centralized control

- ✧ Most of the time, you can think of the participants in an interaction diagram as objects, as indeed they were in UML 1. But in UML 2, their roles are much more complicated
- ✧ I use the term participants, a word that isn't used formally in the UML spec. In UML 1, participants were objects and so their **names were underlined**, but in UML 2, they should be shown without the underline, as I've done here
- ✧ In these diagrams, I've named the participants using the style **anOrder**. This works well most of the time. A fuller Syntax is **name : Class**, where both the name and the class are optional

A sequence diagram for centralized control

- ✧ Each lifeline has an **activation bar** that shows when the participant is active in the interaction. This corresponds to one of the participant's methods being on the stack. Activation bars are **optional** in UML, but I find them extremely valuable in clarifying the behavior
- ✧ Naming often is useful to correlate participants an the diagram. The call **getProduct** is shown returning **aProduct**, which is the same name, and therefore the same participant.
- ✧ Some people use returns for all calls, but I prefer to use them only where they add Information

A sequence diagram for distributed control



Centralized vs distributed control

- The clear difference in styles between the two interactions. Figure 4 .1 is centralized control, with ***one participant pretty much doing all the processing*** and other participants there to supply data. Figure 4 .2 uses ***distributed control***, in which the processing is ***split among many participants***, each one doing a little bit of the algorithm

Centralized vs distributed control

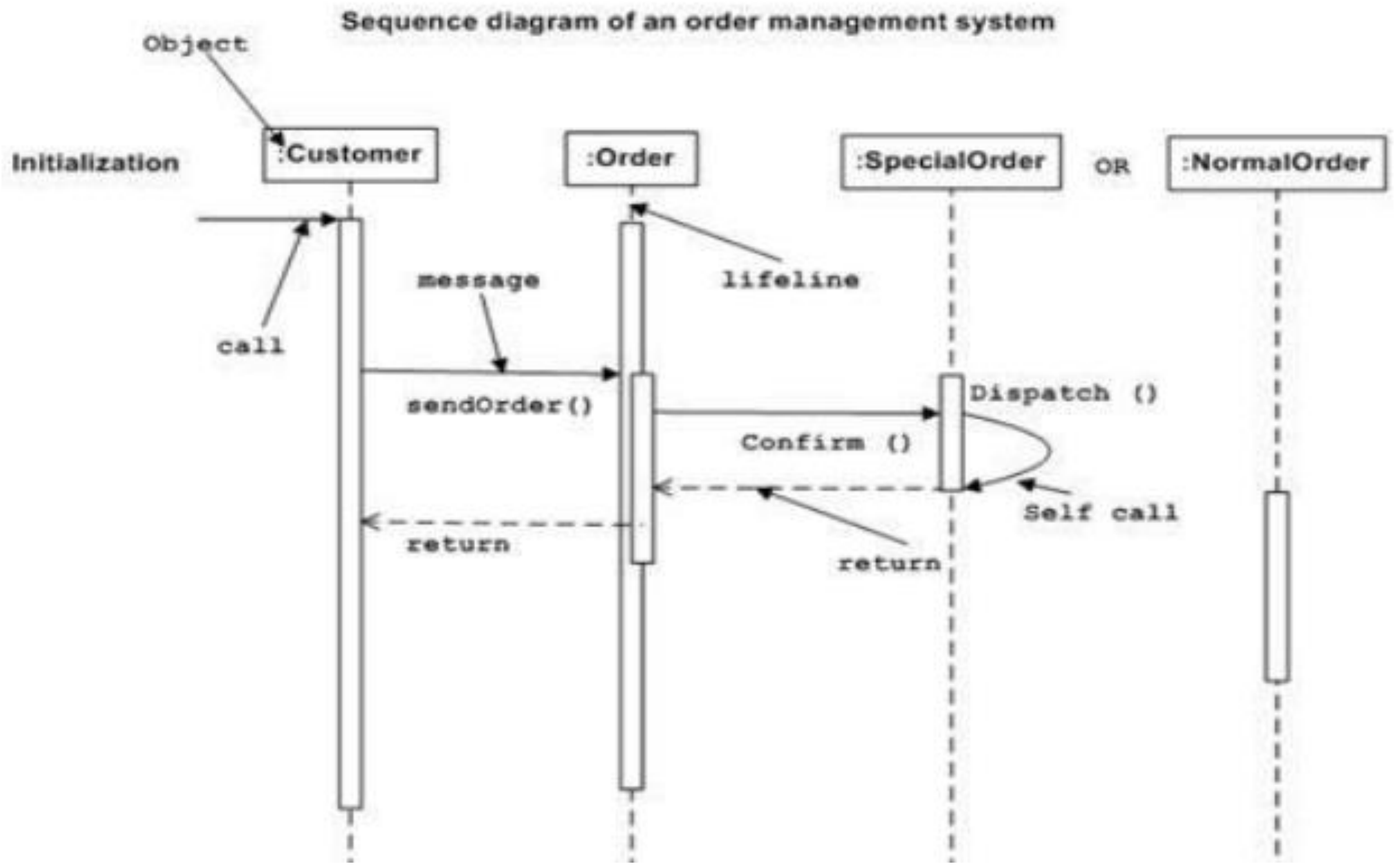
- ✧ Both styles have their strengths and weaknesses .
- ✧ Most people, particularly those new to objects, are more used to centralized control. In many ways, it‘s <u>simpler</u>, as all the processing is in <u>one place</u>
- ✧ With distributed control, in contrast, you have the sensation[cảm-giác] of chasing[chạy-theo] around the objects, trying to find the program

Centralized vs distributed control

- ✧ One of the main goals of good design is to localize the effects of change. Data and behavior that accesses that data often change together. So putting the data and the behavior that uses it together in one place is the first rule of object-oriented design.
- ✧ By distributing control, you create more opportunities for using polymorphism rather than using conditional logic . If the algorithms for product pricing are different for different types of product, the distributed control mechanism allows us to use subclasses of product to handle these variations

Sequence Diagrams – Another Example

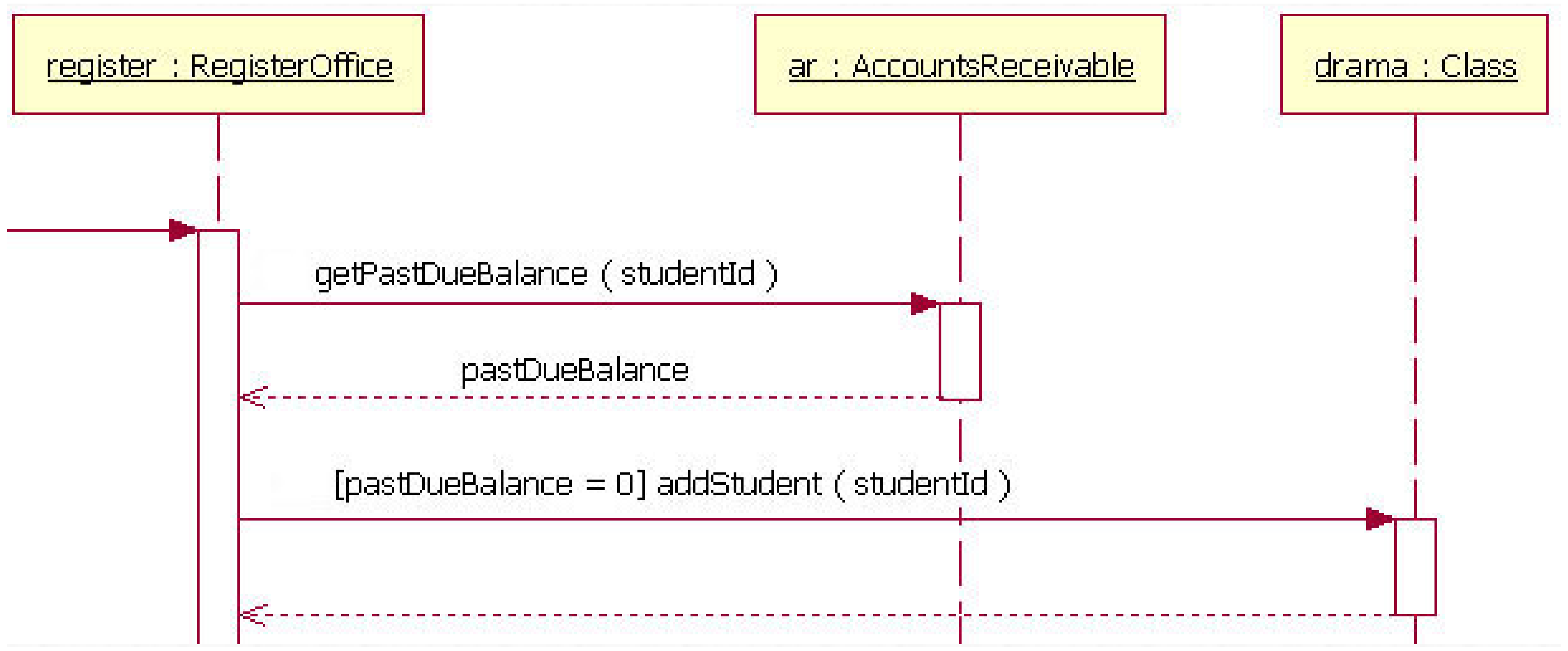
✧ Bài toán hệ thống quản lý đơn đặt hàng



When to Use Sequence Diagrams

- ✧ You should use sequence diagrams when you want to look at the behavior of several objects within a single use case.
- ✧ Sequence diagrams are good at showing collaborations among the objects; they are not so good at precise definition of the behavior.
- ✧ If you want to look at the behavior of a single object across many use cases, use a state diagram
- ✧ If you want to look at behavior across many use cases or many threads, consider an activity diagram

sequence diagram with conditions


register : RegisterOffice
ar : AccountsReceivable
drama : Class
getPastDueBalance (studentId)
pastDueBalance
[pastDueBalance = 0] addStudent (studentId)