# Use Case Modeling
# UML Modeling
# Lecture 16

# Introduction

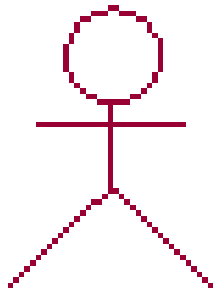
- **Use case Introduction**
  - Use-Cases diagram is an effective tool that models how the end user interact with the system.
  - Use-Cases define what exactly is outside the system(actor) and what should be performed by the system(Use-Cases)
  - Use-Cases modeling is very effective in modeling user interfaces

# Components of use-case diagram

- **Use case diagram components**
  - Actor
  - Use case
  - System Boundary
  - Link
  - Relationship
  - Generalization

# Actors

- **Actor**
  - An actor is ANYONE or ANYTHING that must interact with the system.
  - Actors are NOT part of the system
  - In the UML, an actor is represented as a stickman.



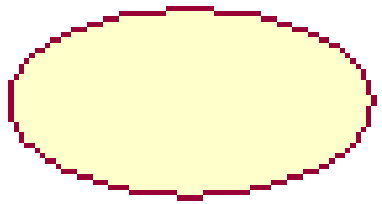
Actor

# Use-Cases

- **Use case**
  - The term use case means function, process or operation.
  - The operation performed by software system is use case
  - It represents the capability of the system that what a system can do. It may be any function, process or operation.
  - A Use-Case is a description of a set of sequences of actions that a system performs
  - A Use-Cases typically represents a major piece of functionality that is complete from beginning to end.

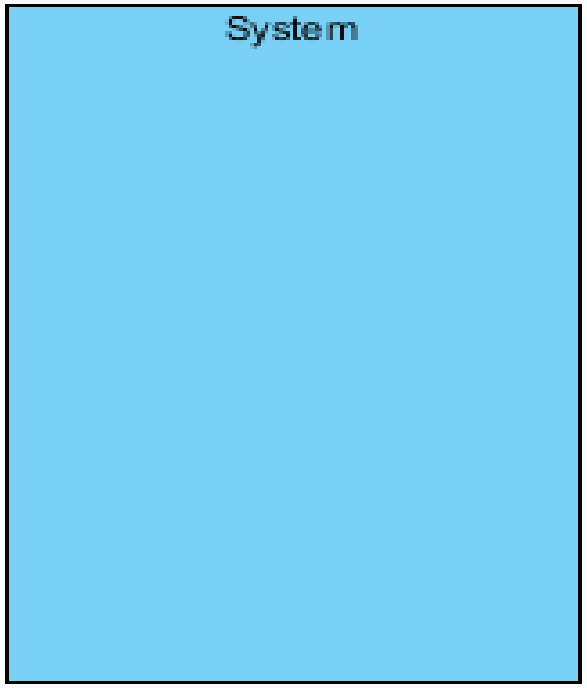
# Use-Cases *(Cont'd)*

- **Use case**
  - Use-Cases are tasks those are performed by the system
  - A Use-Cases must deliver something of value to an actor.
  - Use-Cases is represented as an oval.



# System Boundary

- **System boundary**
  - The system boundary is potentially the entire system as defined in the requirements document.



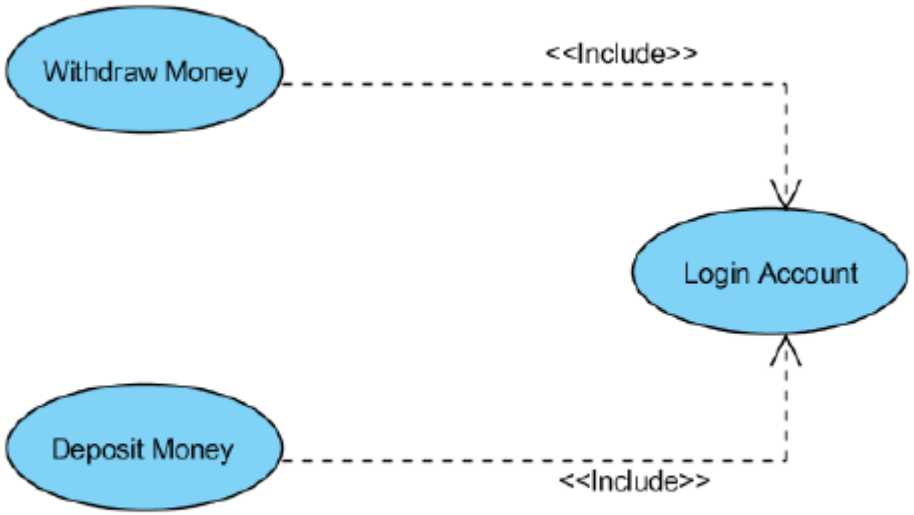
# Communication Link

- **Communication Link**
  - The participation of an actor in a use case is shown by connecting a actor to a use case by a solid link.
  - Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages

# Include Relationship

- **Include Relationship**
  - When a use case is using functionality of another use case, this relationship between the use cases is named as an include or uses relationship.

# Include Relationship

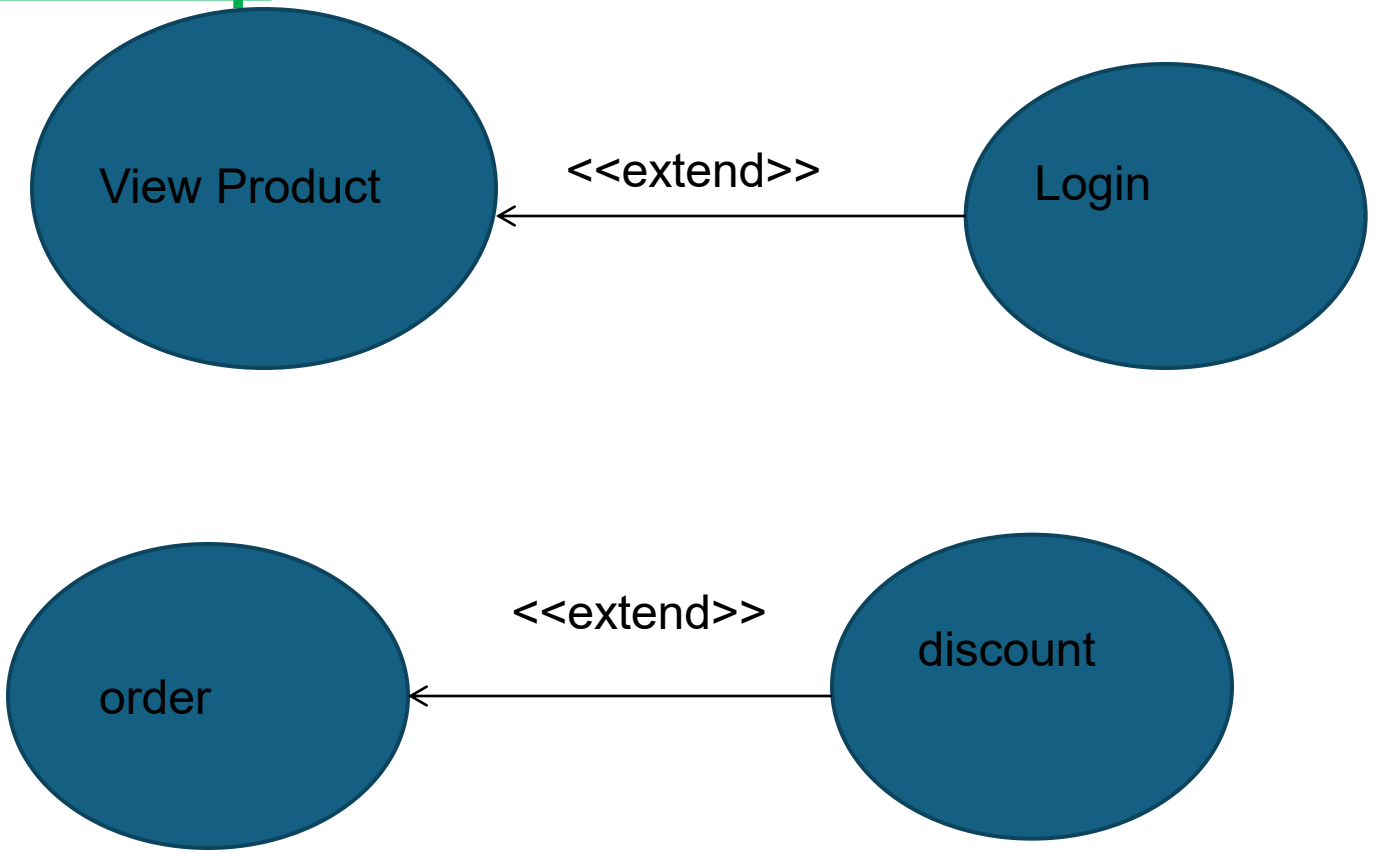


# Extend Relationship (Optional)

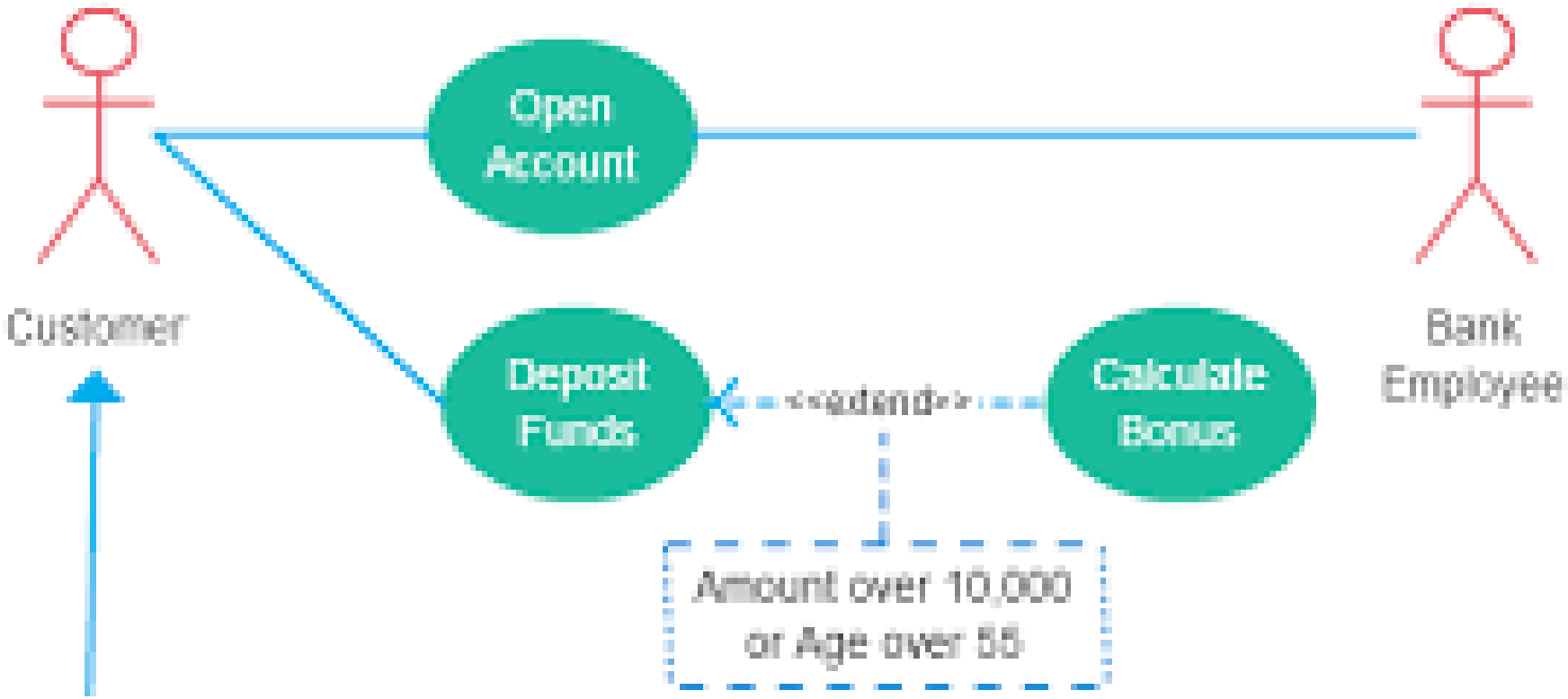
- **Extend Relationship**
  - Another kind of relationship between uses cases is called the extend dependency
  - An extend dependency shows optional relationship between two use cases
  - In this dependency we have two use cases one is called as ***base use case*** and the other is called ***extending use case***.
  - You can use an extend relationship to specify that one use case (extension) extends the behavior of another use case (base)

# Extend Relationship

- **Extend Relationship**



# Extend relationship



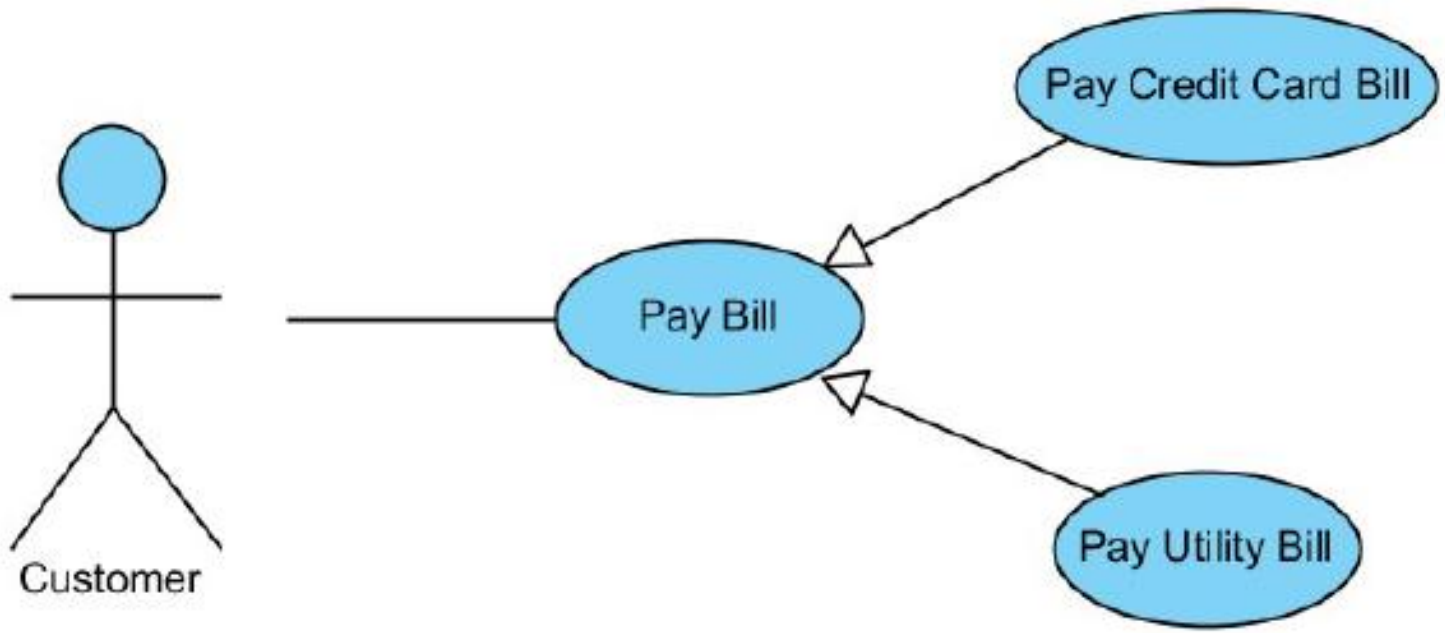
# Generalization Relationship

- **Generalization Relationship**

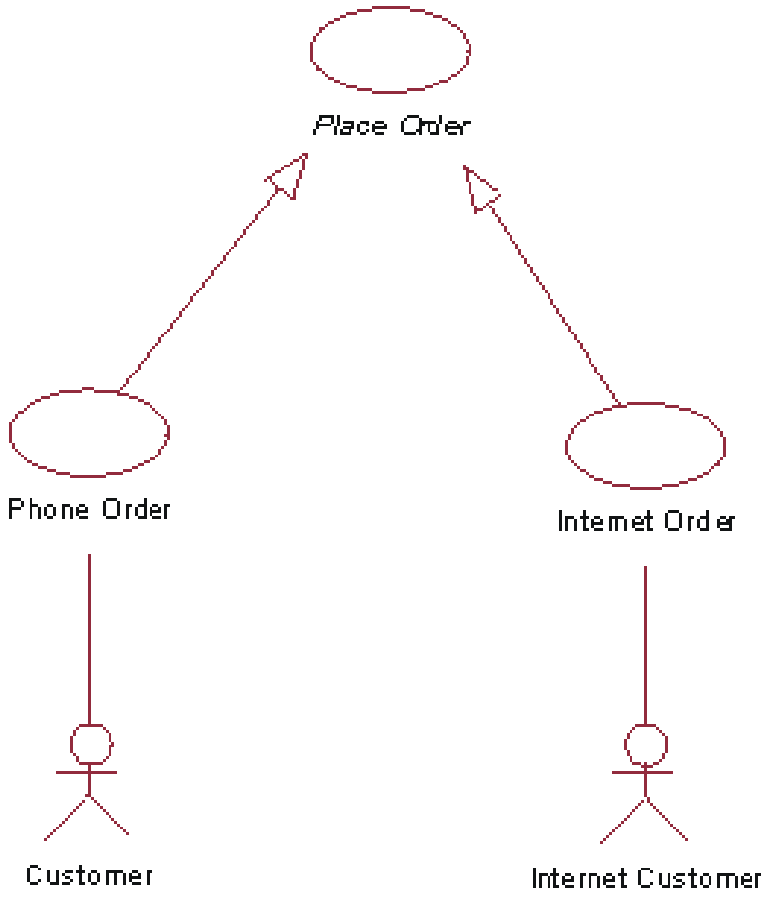
  - Generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent)
  - Generalization relationship can be used between actors or between use cases

  - *For example*: suppose the ATM system can be used to pay bills. Pay Bill has two child use cases:
    1. Pay Credit Card Bill
    2. Pay Utility Bill

# Generalization example



# Generalization Relationship



# ATM: Problem Description

- **ATM Example**
  - A bank has several automated teller machines (ATMs).
  - Each ATM machine has a card reader, a cash dispenser, a keyboard/display, and a receipt printer.
  - By using the ATM machine, a customer can withdraw cash from either a checking or savings account, query the balance of an account, or transfer funds from one account to another.
  - A transaction is initiated when a customer inserts an ATM card into the card reader.
  - The system validates the ATM card to determine that the expiration date has not passed, that the user-entered PIN (personal identification number) matches the PIN maintained by the system, and that the card is not lost or stolen.
  - The customer is allowed three attempts to enter the correct PIN - the card is removed if the third attempt fails.

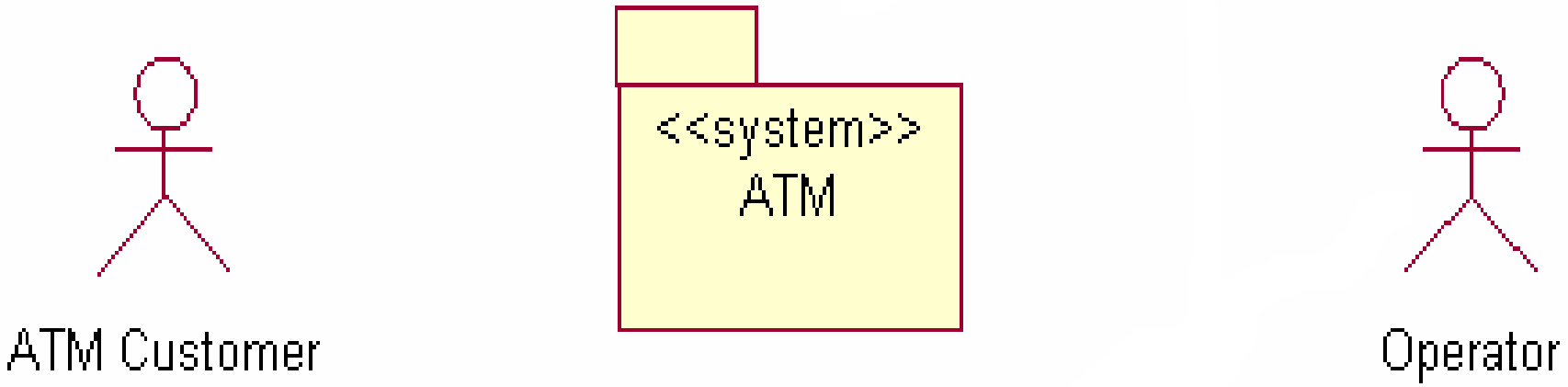
# ATM: Problem Description *(Cont'd)*

- **ATM Example**
  - If the PIN is validated satisfactorily, the customer is prompted for a withdrawal, query, or transfer transaction.
  - Before a withdrawal transaction can be approved, the system, determines that sufficient funds exist in the requested account, that the maximum daily limit will not be exceeded, and that there are sufficient funds at the local cash dispenser.
  - If the transaction is approved, the requested amount of cash is dispensed, a receipt is printed containing information about the transaction, and the card is ejected.
  - Before a transfer transaction can be approved, the system determines that the customer has at least two accounts and that there are sufficient funds in the account to be debited.
  - For approved query and transfer requests, a receipt is printed and the card is ejected.
  - A customer may cancel a transaction at any time; the transaction is terminated and the card is ejected.

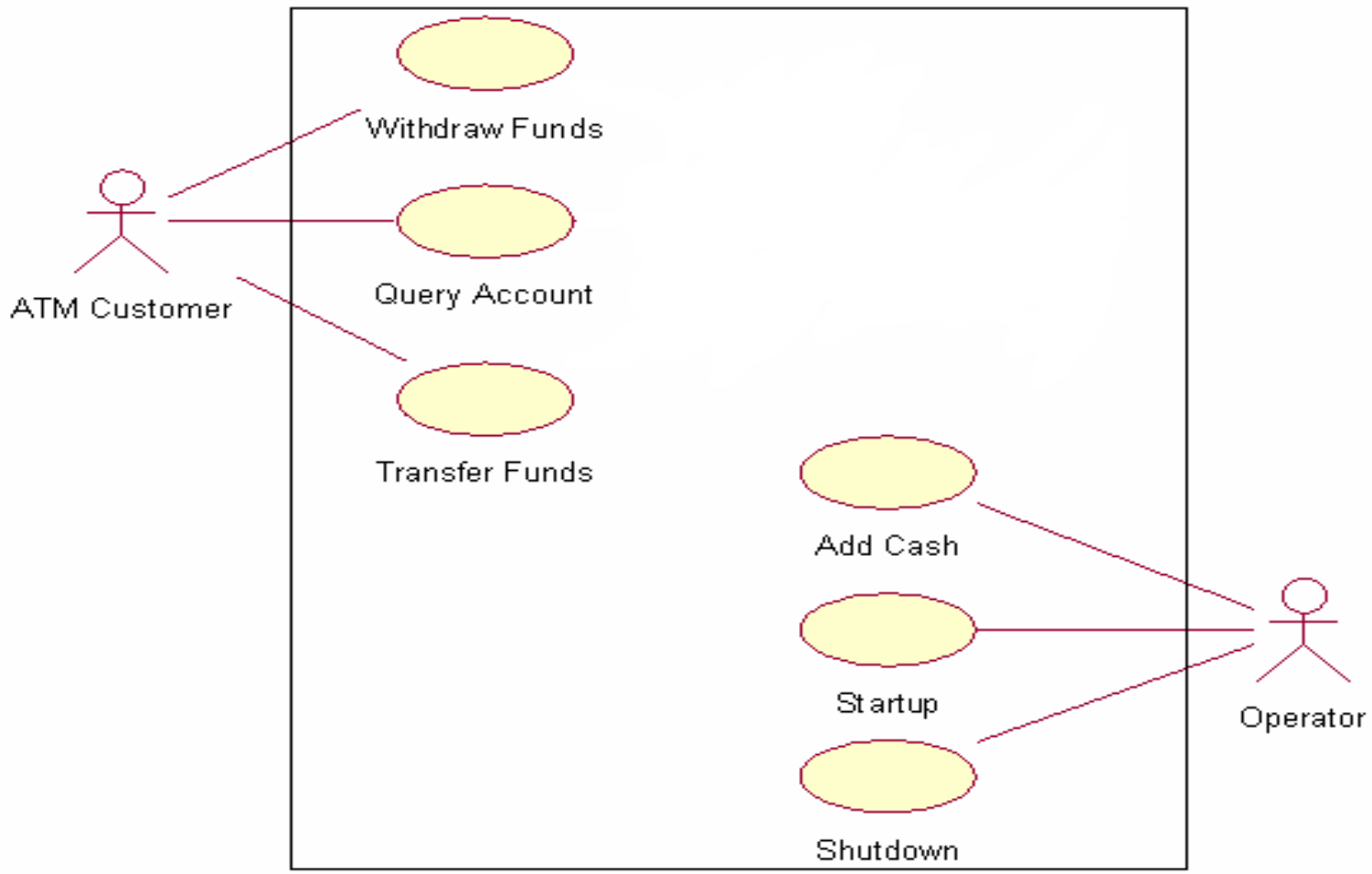
# ATM: Problem Description *(Cont'd)*

- **ATM Example**
  - An ATM operator may start up and close down the ATM to replenish the ATM cash dispenser and for routine maintenance.
  - It is assumed that functionally to open and close accounts and to create, update, and delete customer and debit card records is provided by an existing system and is not part of this problem.

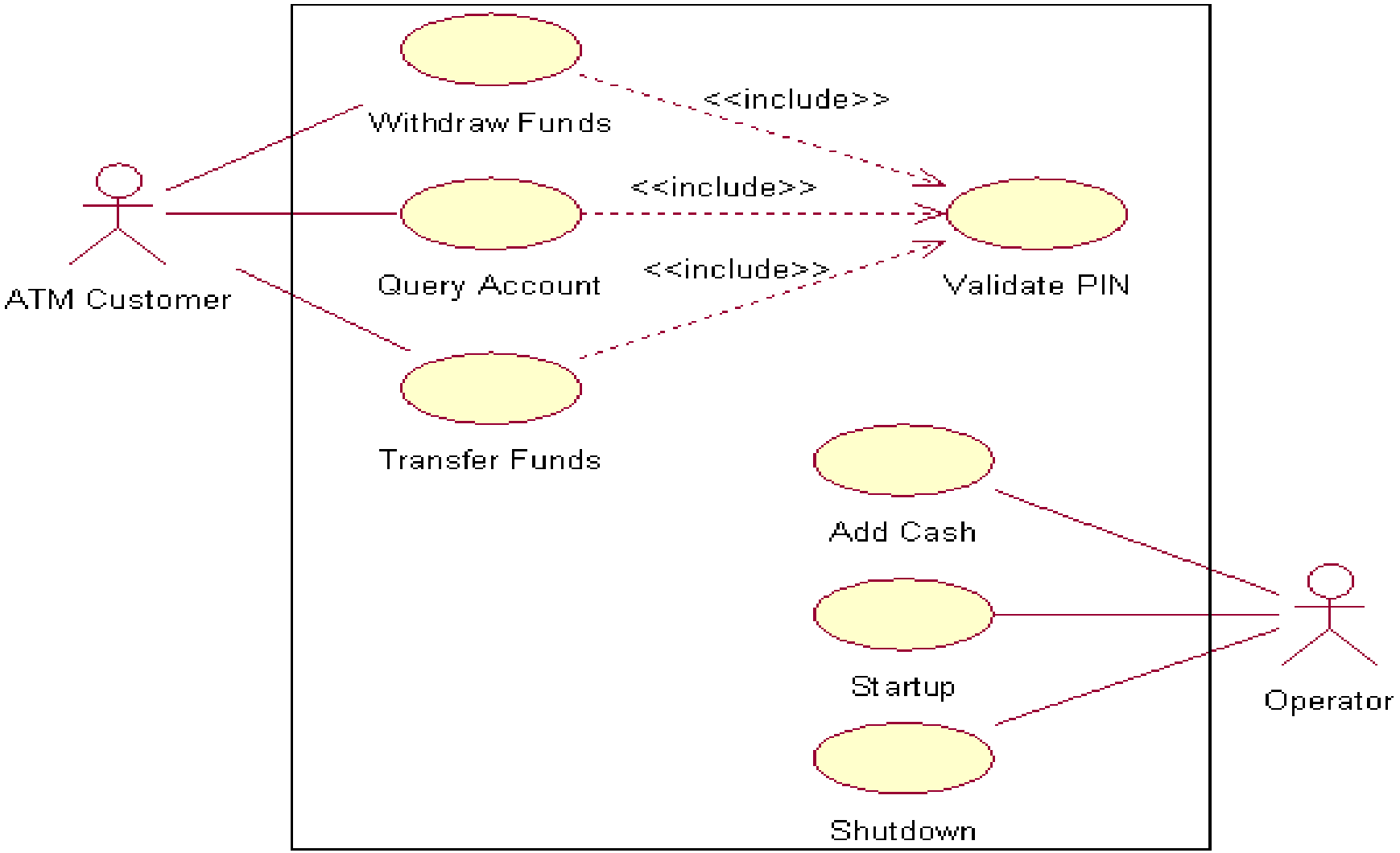
# ATM System



# ATM: Use-Case Model



# ATM: Use-Case Model



# Unified Modeling Language (UML) Notations

# Overview

- UML
- UML Notation

# UML

- ***What is UML***
  - UML diagram is a way to visualize systems and software using Unified Modeling Language (UML)
  - UML diagrams is used to diagram the **behavior and structure** of a system.
  - A Unified Modeling Language (UML) diagram provides a visual representation of an aspect of a system.

# UML Notations…

- ***UML Notations***
  - Graphical notations are most widely used in UML.
  - Following is a list of Object Oriented (OO) System contents that we will learn about their diagrams.

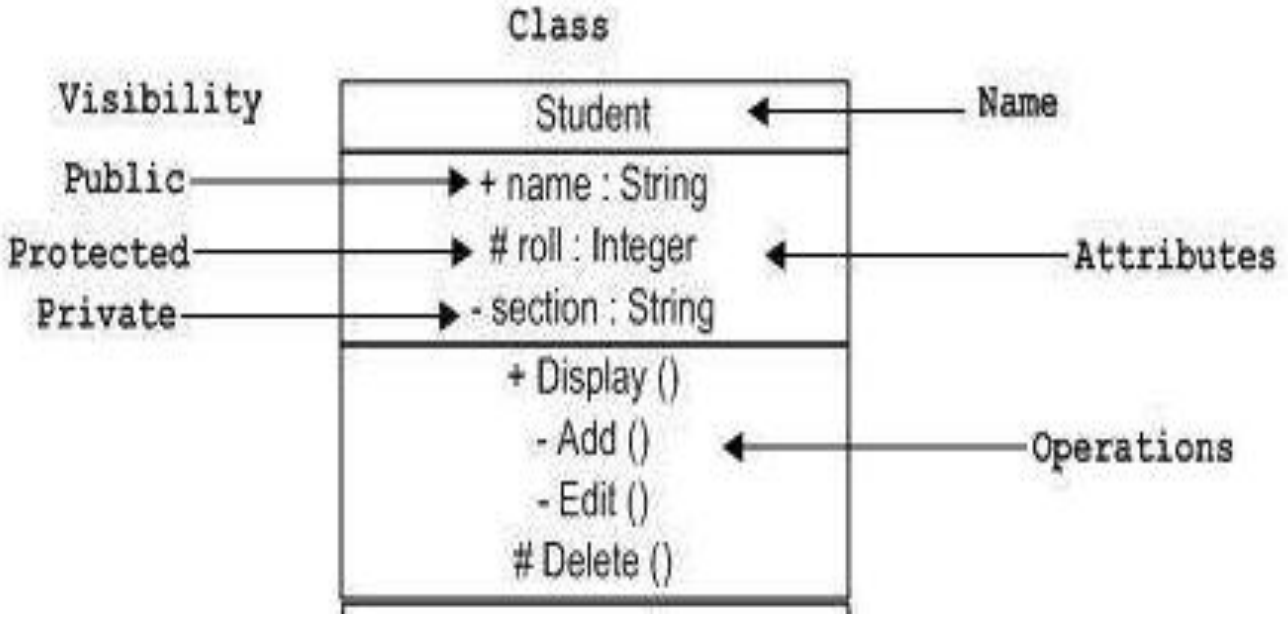
☞Classes
☞Object
☞Actor
☞Use case
☞Components
☞Node
☞Final State
☞Initial State

# Classes and Objects

- Class: An entity that has a well-defined role in the application domain, as well as state, behavior, and identity
    - Tangible: person, place or thing
    - Concept or Event: department, performance, marriage, registration
    - Artifact of the Design Process: user interface, controller, scheduler
- Object: a particular instance of a class

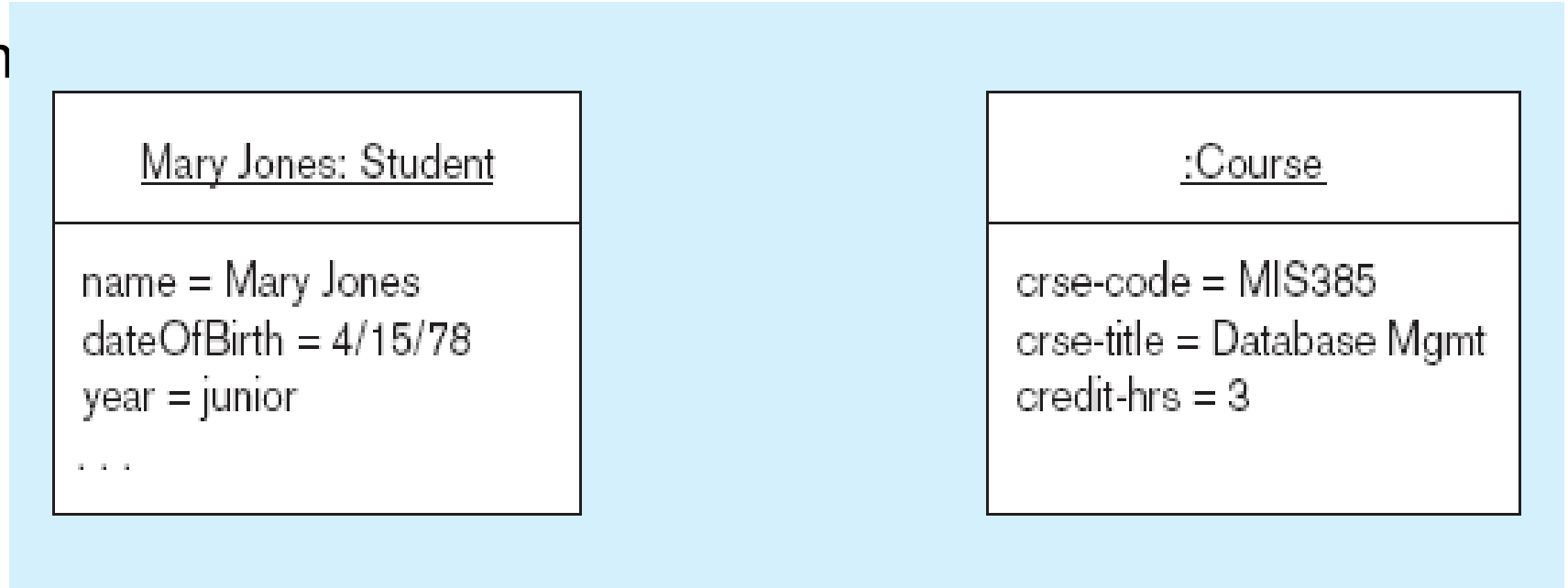
# UML Class Notation

- ***Class Notation***
- class diagram describes the structure of the system or the details of an implementation



# UML Object notation

- Object Diagram




- Object diagram shows instances that are compatible with a given class diagram.

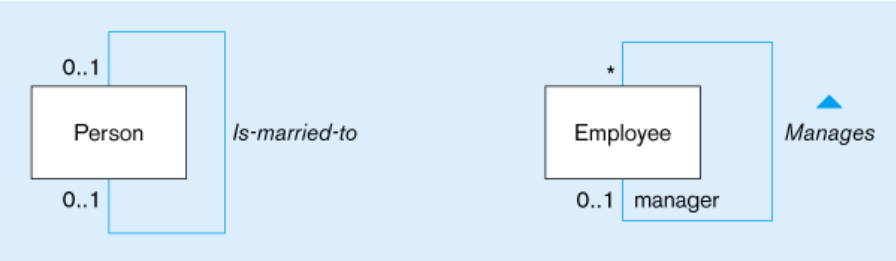
# Associations

- **Association**:
  - Relationship among object classes

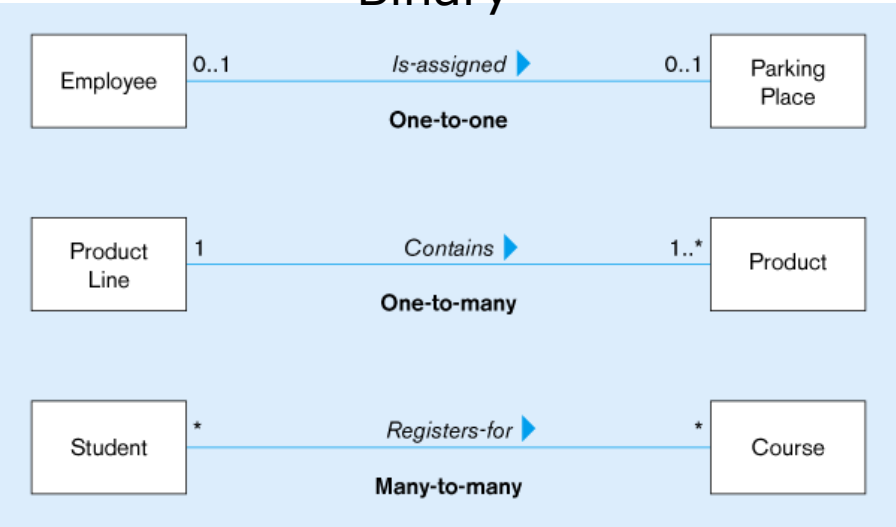
- **Multiplicity**:
  - How many objects participate in an association. Lower-bound..Upper bound (cardinality).

# Figure - Association relationships of different degrees

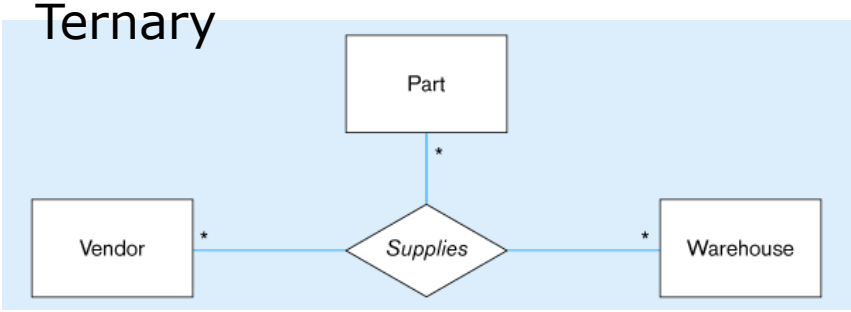
## Unary




## Binary

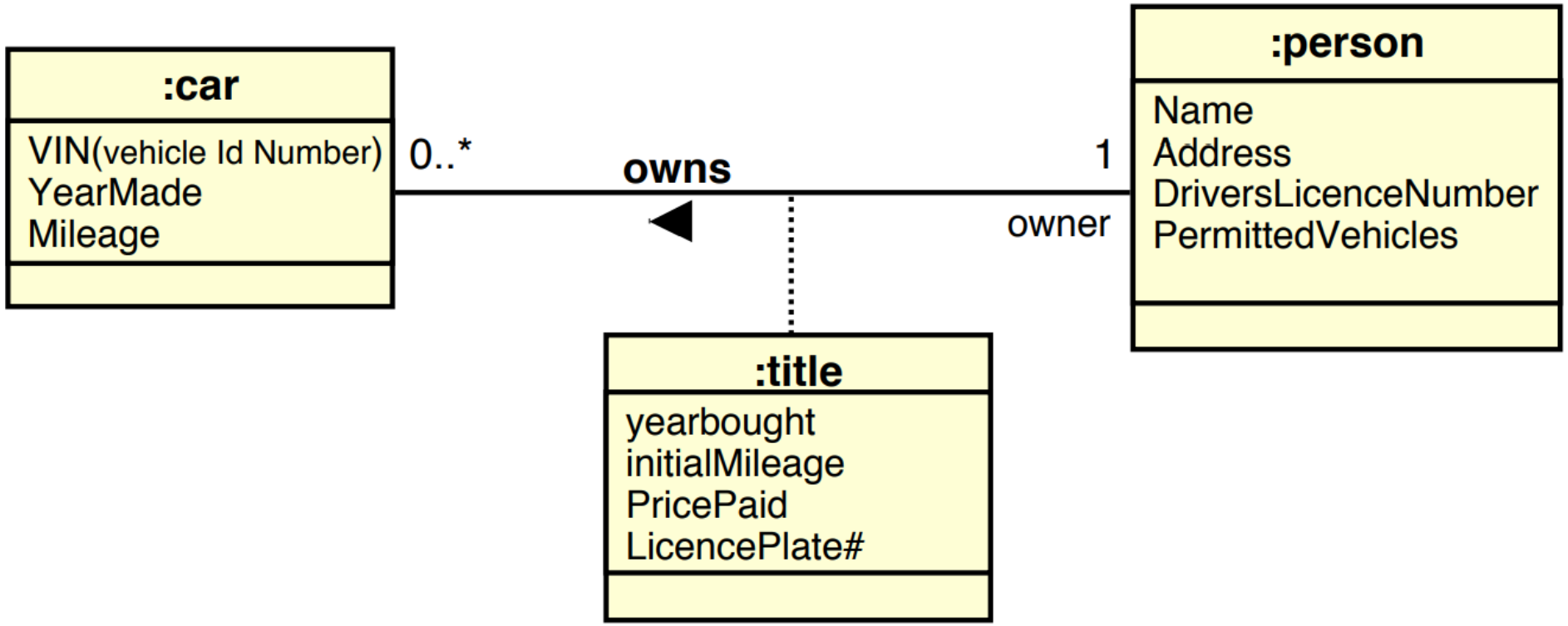



## Ternary



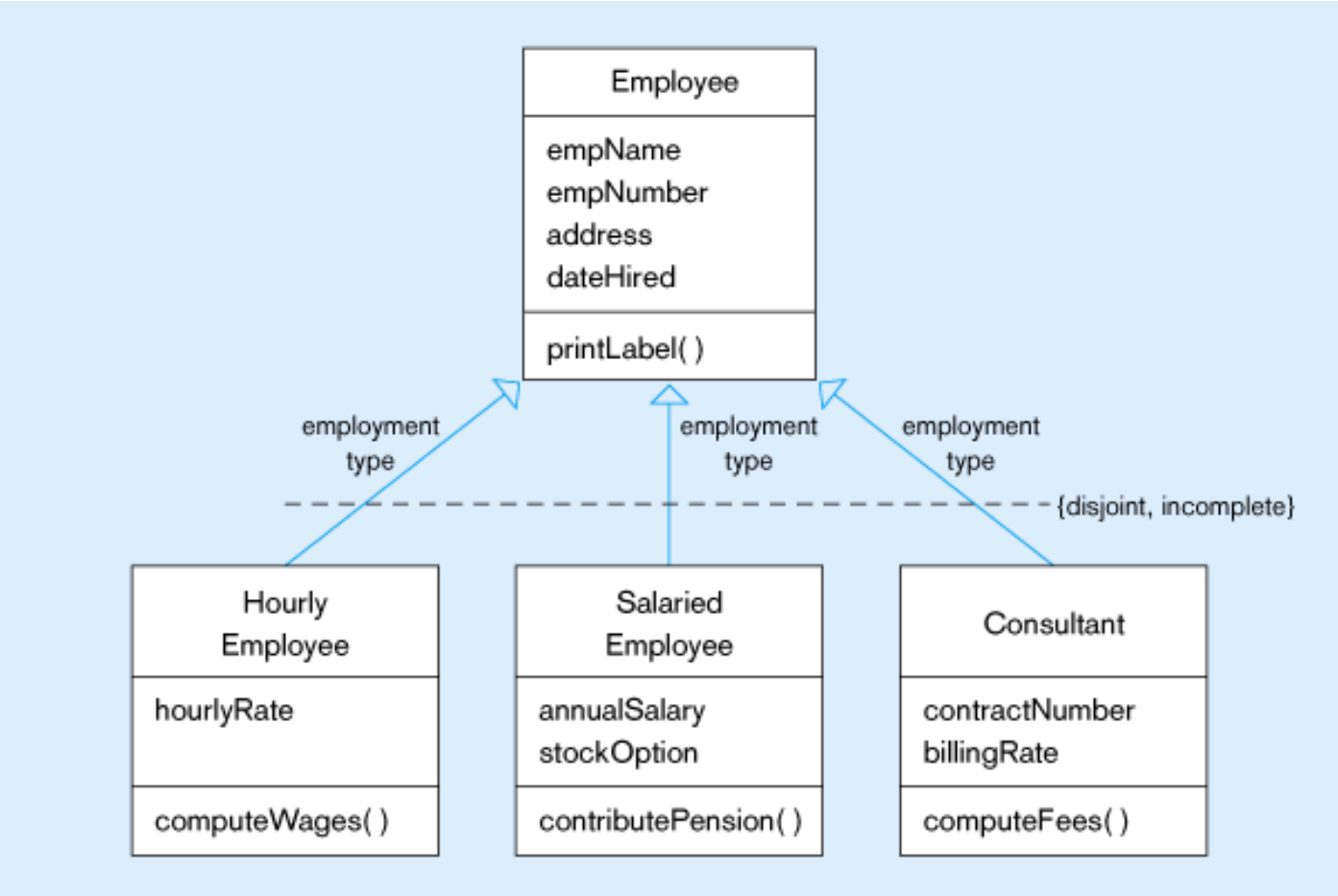
# Some association is it self a class

To keep information about relationship, we need association class



# Generalization / Specialization

- Subclass, superclass
  - Inheritance



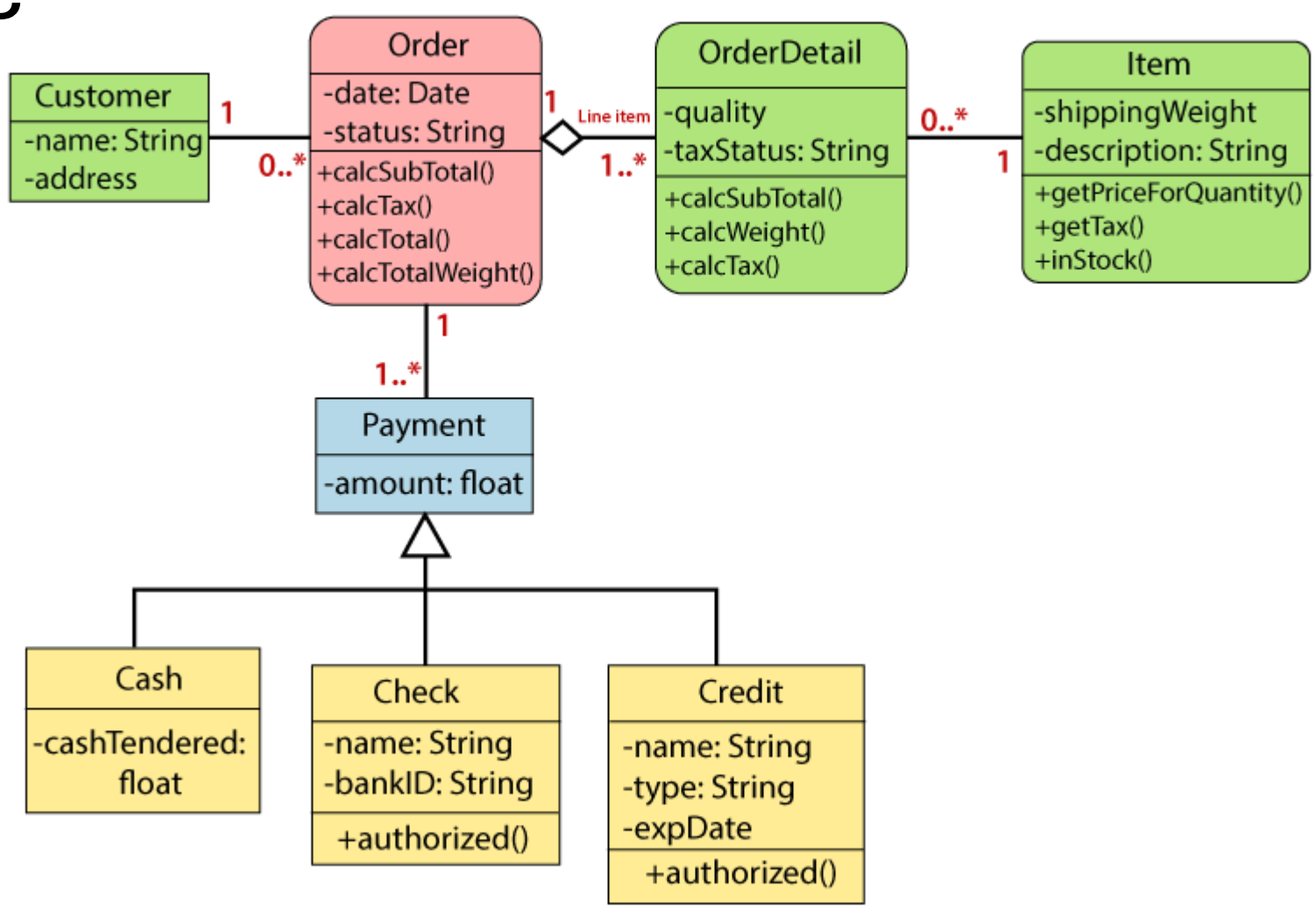
# Dependency

- **Aggregation**
  - The best way to understand this relationship is to call it a “has a” or “is part of” relationship.
  - For example, consider the two classes: Wallet and Money. A wallet “has” money. But money doesn’t necessarily need to have a wallet so it’s a one directional relationship.

**Composition:**

- a restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other.

# Example


Customer
-name: String
-address
1
0..*
Order
-date: Date
-status: String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
Line item
1..*
OrderDetail
-quality
-taxStatus: String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description: String
+getPriceForQuantity()
+getTax()
+inStock()
1
1..*
Payment
-amount: float
Cash
-cashTendered: float
Check
-name: String
-bankID: String
+authorized()
Credit
-name: String
-type: String
-expDate
+authorized()

Thank You