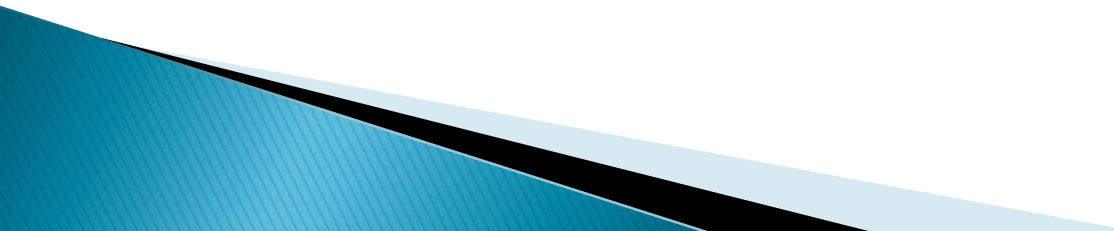# Object Oriented Analysis and Design Using the UML

Object Oriented Analysis

Object Oriented Analysis
Process of Object Modeling

1. Modeling the functions of the system.

2. Finding and identifying the business objects.

3. Organizing the objects and identifying their relationships.

Object Oriented Analysis
Process of Object Modeling

- **Identifying objects:**
  - Using concepts, CRC cards, stereotypes, etc.
- **Organising the objects:**
  - classifying the objects identified, so similar objects can later be defined in the same class.
- **Identifying relationships between objects:**
  - this helps to determine inputs and outputs of an object.
- **Defining operations of the objects:**
  - the way of processing data within an object.
- **Defining objects internally:**
  - information held within the objects.

Goals of OO analysis

▶ What are the two main goals of OO analysis?
  1) Understand the customer's requirements
  2) Describe problem domain as a set of classes and relationships

▶ What techniques have we studied for the 1st goal?
  ◦ Develop a requirements specification
  ◦ Describe scenarios of use in user's language as use cases

▶ What techniques have we studied for the 2nd goal?
  ◦ CRC cards discover classes and run simulations
  ◦ UML class diagrams represent classes & relationships
  ◦ Sequence diagrams model dynamic behavior of a system

Construction the Analysis Use-Case Model

System analysis use case – a use case that documents the interaction between the system user and the system. It is highly detailed in describing what is required but is free of most implementation details and constraints.
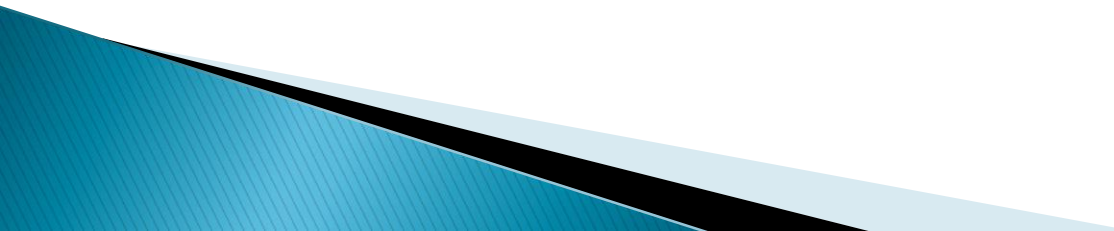
1. Identify, define, and document new actors.

2. Identify, define, and document new use cases.

3. Identify any reuse possibilities.

4. Refine the use-case model diagram (if necessary).

5. Document system analysis use-case narratives.

Use case diagrams

Use Case Diagrams describe the functionality of a system and users of the system.

Describe the functional behavior of the system as seen by the user.
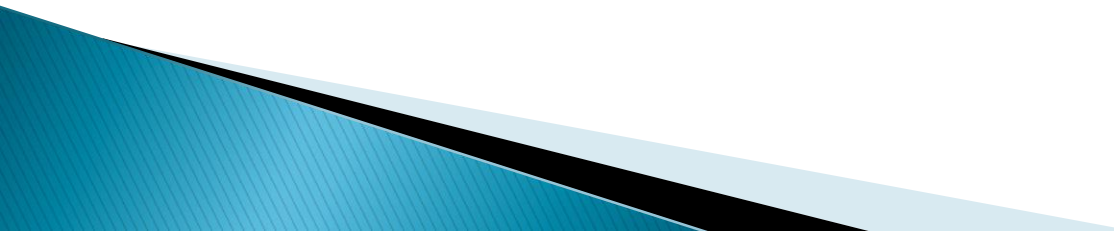
These diagrams contain the following elements:

- **Actors**, which represent users of a system, including human users and other systems.

- **Use Cases**, which represent functionality or services provided by a system to users.

Use case diagrams

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases.

To draw an use case diagram we should have the following items identified. Functionalities to be represented as an use case Actors Relationships among the use cases and actors.

Use case diagrams

Use case diagrams are drawn to capture the functional requirements of a system. So after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

The name of a use case is very important. So the name should be chosen in such a way so that it can identify the functionalities performed.

- ❑   Give a suitable name for actors. Show relationships and dependencies clearly in the diagram.

- ❑   Do not try to include all types of relationships.

- ❑   Because the main purpose of the diagram is to identify requirements.

- ❑   Use note when ever required to clarify some important points.

Use Case Diagrams

Passenger

PurchaseTicket
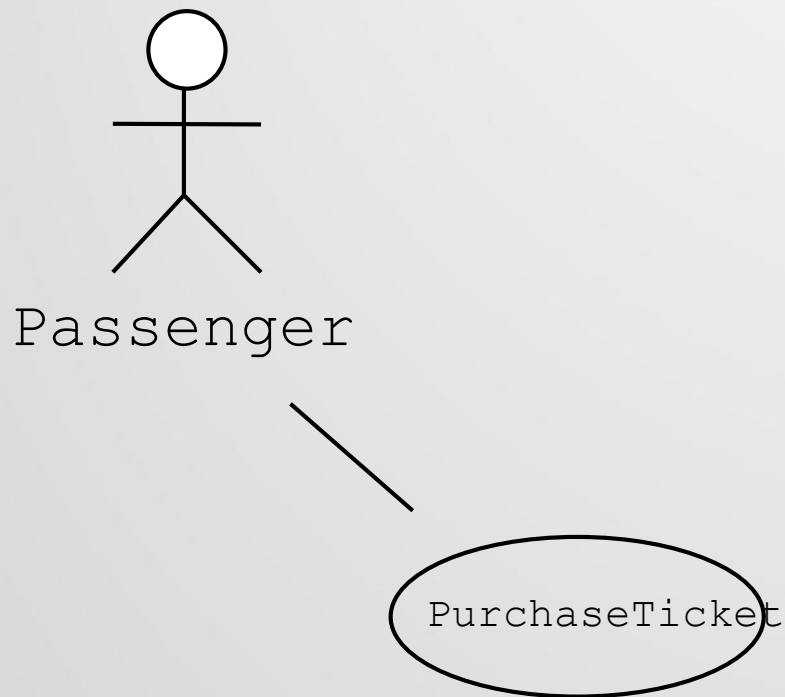
- Used during requirements elicitation to represent external behavior

- *Actors* represent roles, that is, a type of user of the system
- *Use cases* represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

Actors
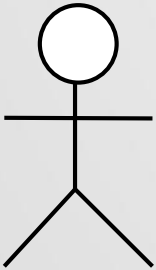
Passenger

- An actor models an external entity which communicates with the system:
  - User
  - External system
  - Physical environment
- An actor has a unique name and an optional description.
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates
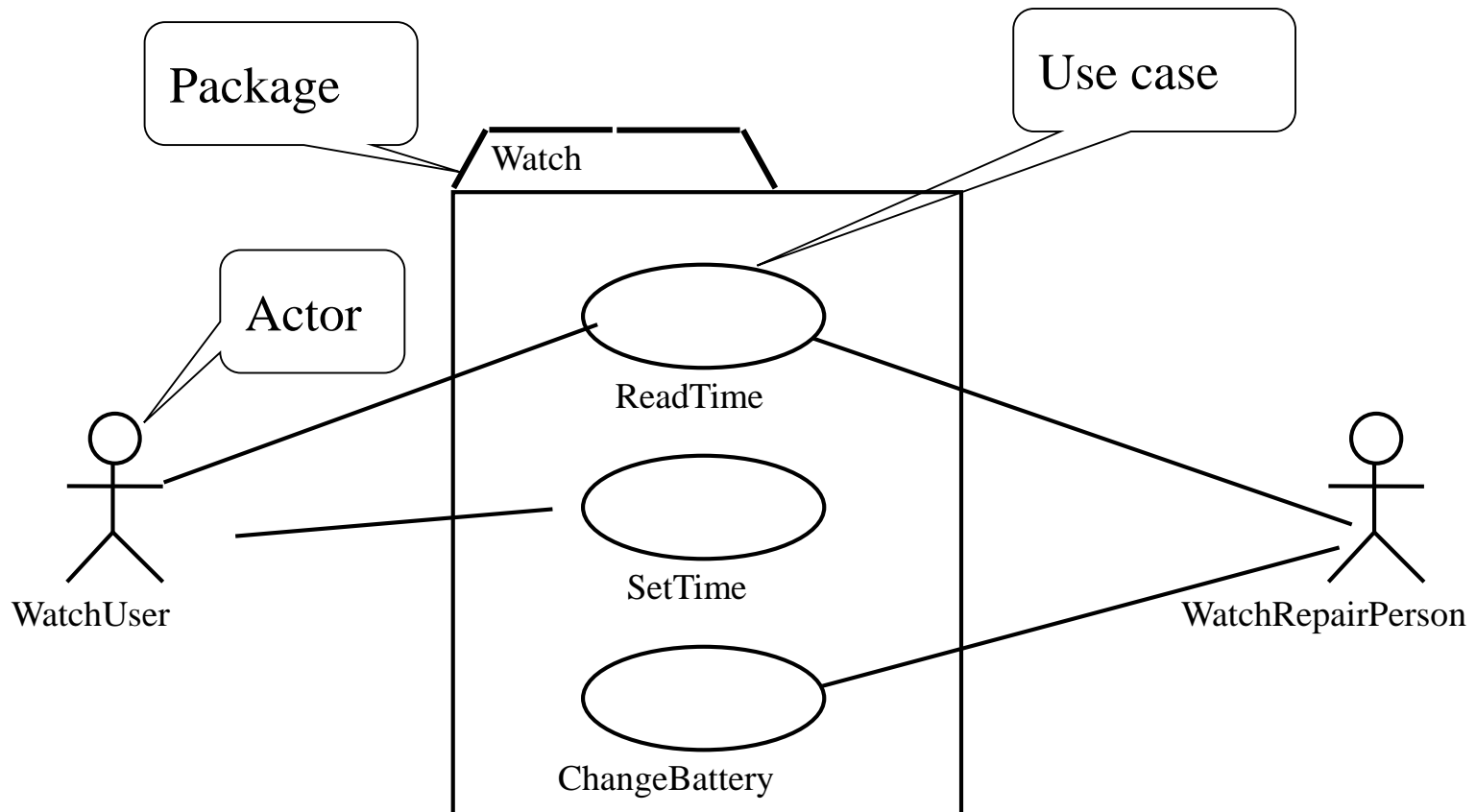
Use Case

A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:
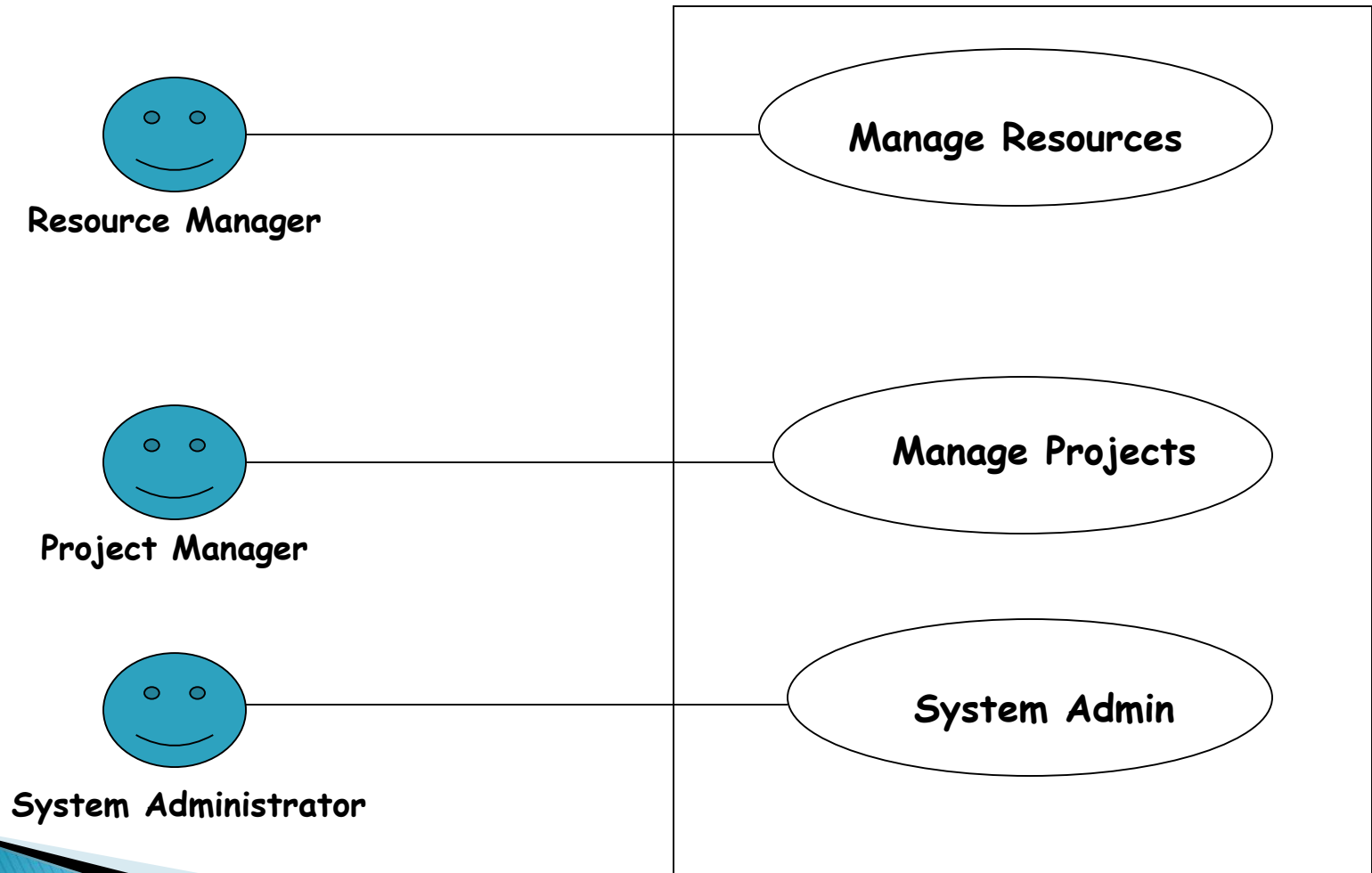
▸ Unique name

▸ Participating actors

▸ Entry conditions

▸ Flow of events

▸ Exit conditions

▸ Special requirements

PurchaseTicket
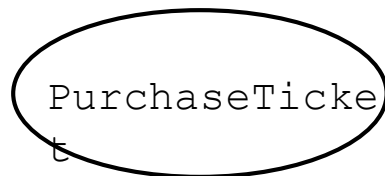
Use case diagrams

Example: High Level Use Case Diagram

Example: Managing Resources Use Case Diagram
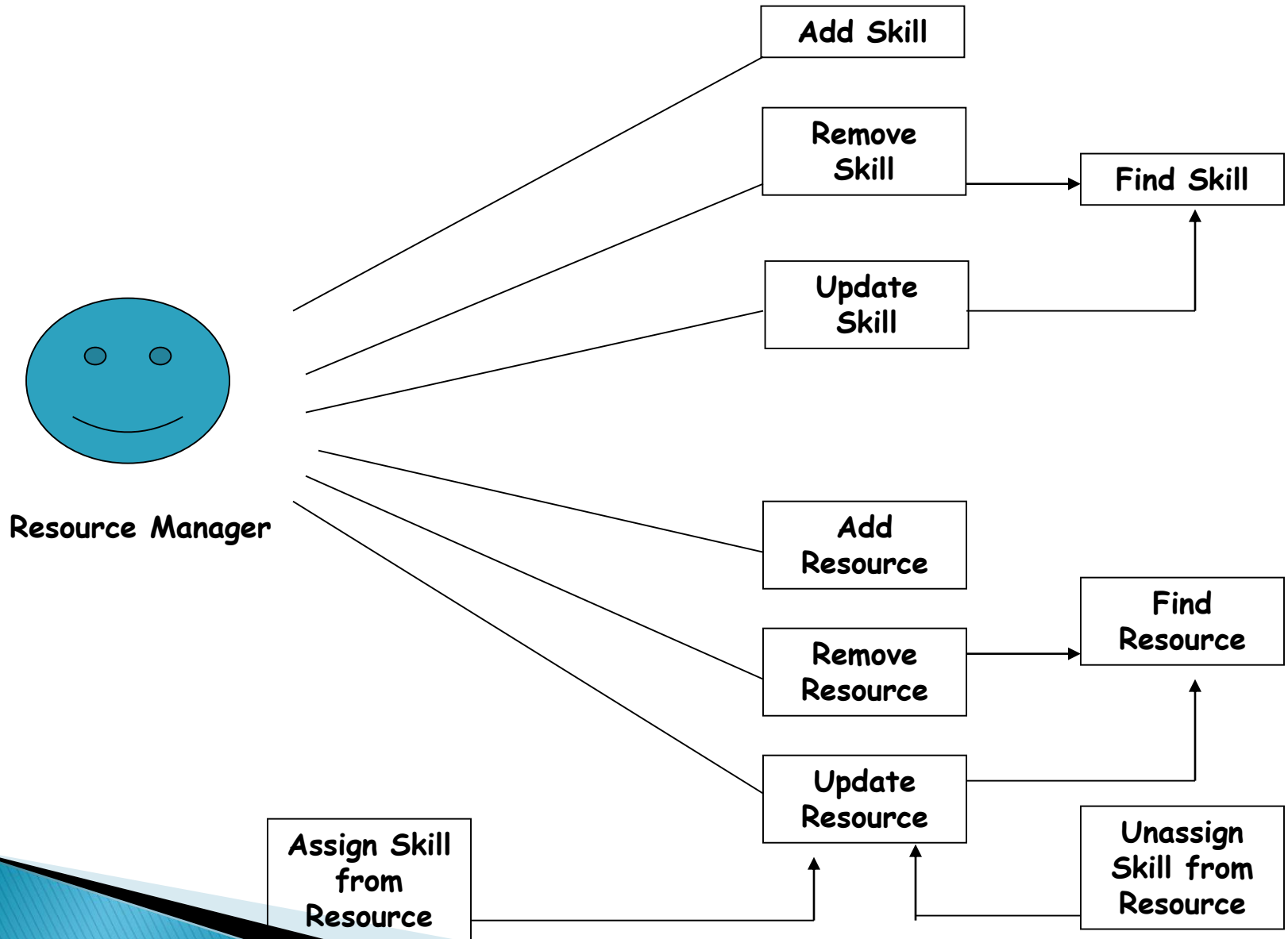
Example: Order management system

The following is a sample use case diagram representing the order management system. So if we look into the diagram then we will find three use cases (Order, SpecialOrder and NormalOrder) and one actor which is customer.

The *SpecialOrder and NormalOrder use cases are extended from Order use case. So they have extends relationship.*

Use case diagram of an order management system
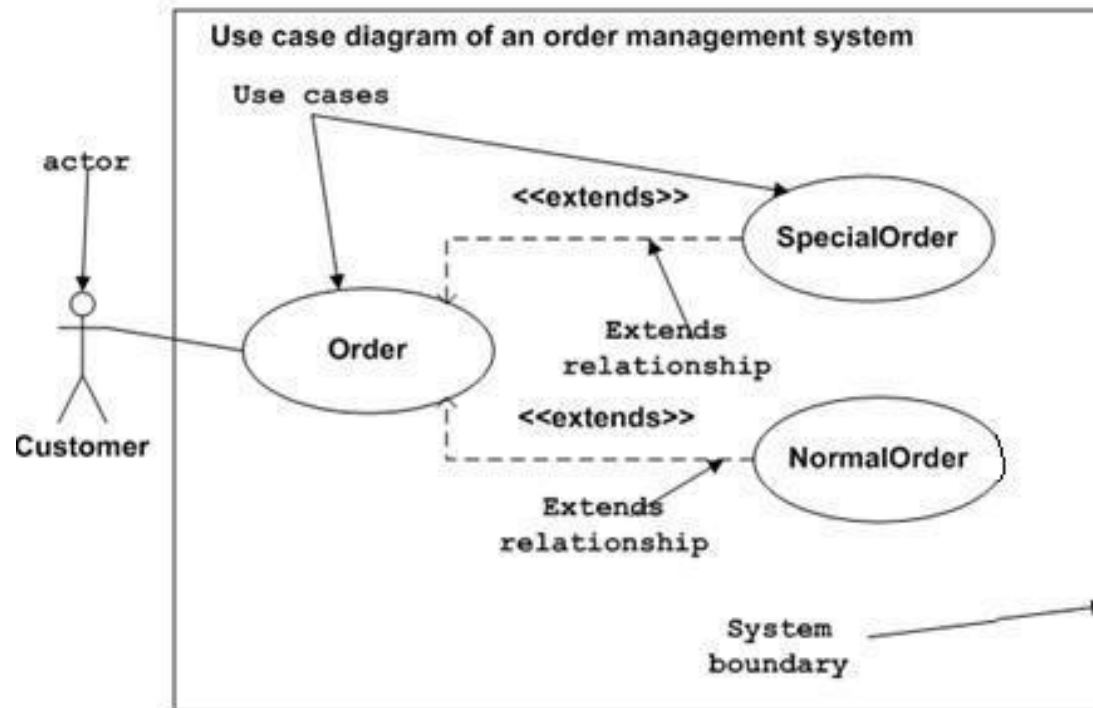
Dependences

Include Dependencies

An include dependency from one use case (called the base use case) to another use case (called the inclusion use case) indicates that the base use case will include or call the inclusion use case. A use case may include multiple use cases, and it may be included in multiple use cases.

A

includes

B
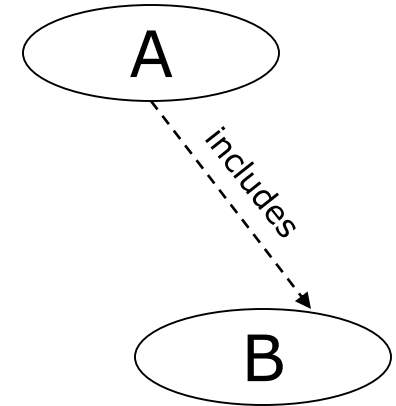
Extend Dependencies

An extend dependency from one use case (called the extension use case) to another use case (called the base use case) indicates that the extension use case will extend (or be inserted into) and augment the base use case. A use case may extend multiple use cases, and a use case may be extended by multiple use cases.

A

extends

B

Generalizations

❑ Actors may be similar in how they use a system; for example, project managers, resource managers, and system administrators may log in and out of our project management system.
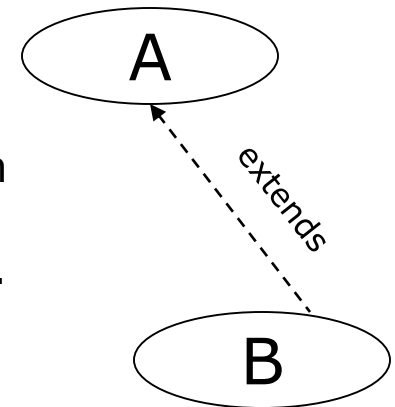
❑ Use cases may be similar in the functionality provided to users; for example, a project manager may publish a project's status in two ways: by generating a report to a printer or by generating a web site on a project web server.
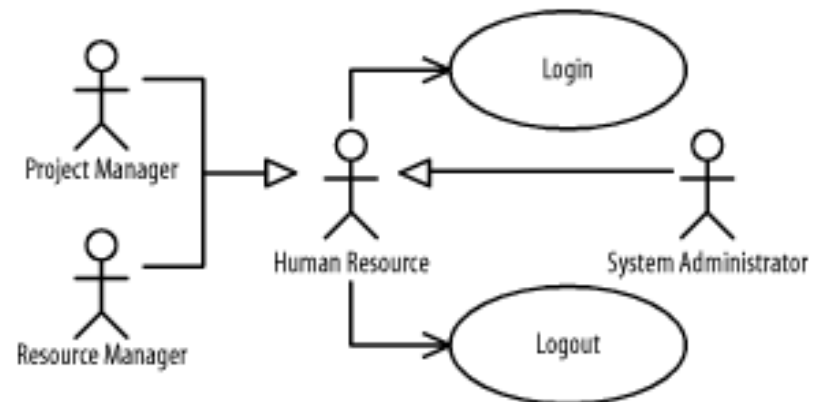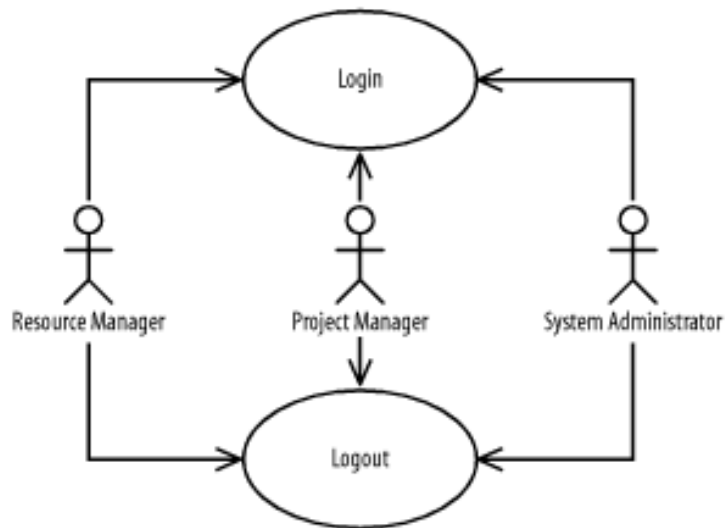
Example

Redraw the given use case diagram after applying:
- Inheritance between actors
- Extend between use cases;
- Include relationship between use cases.
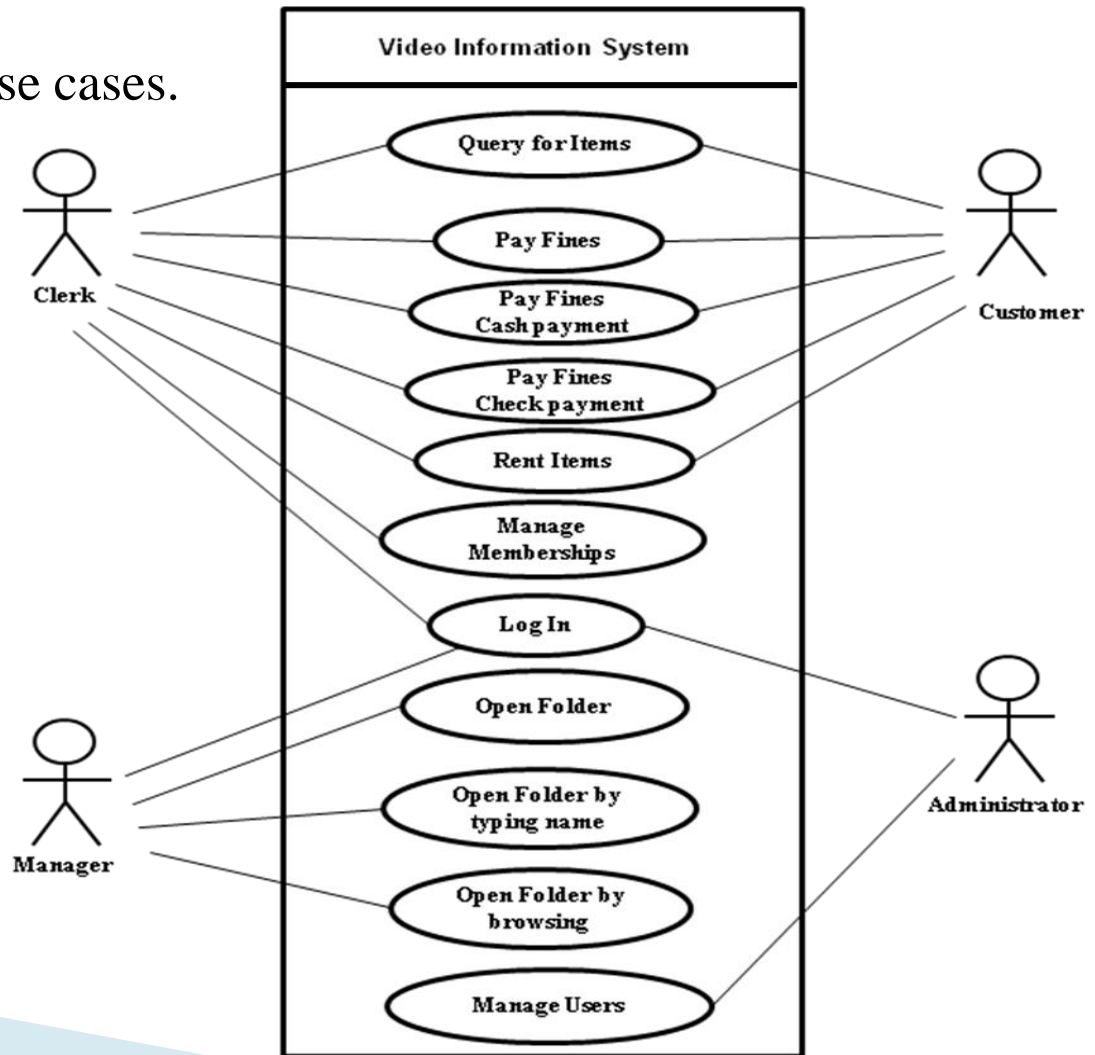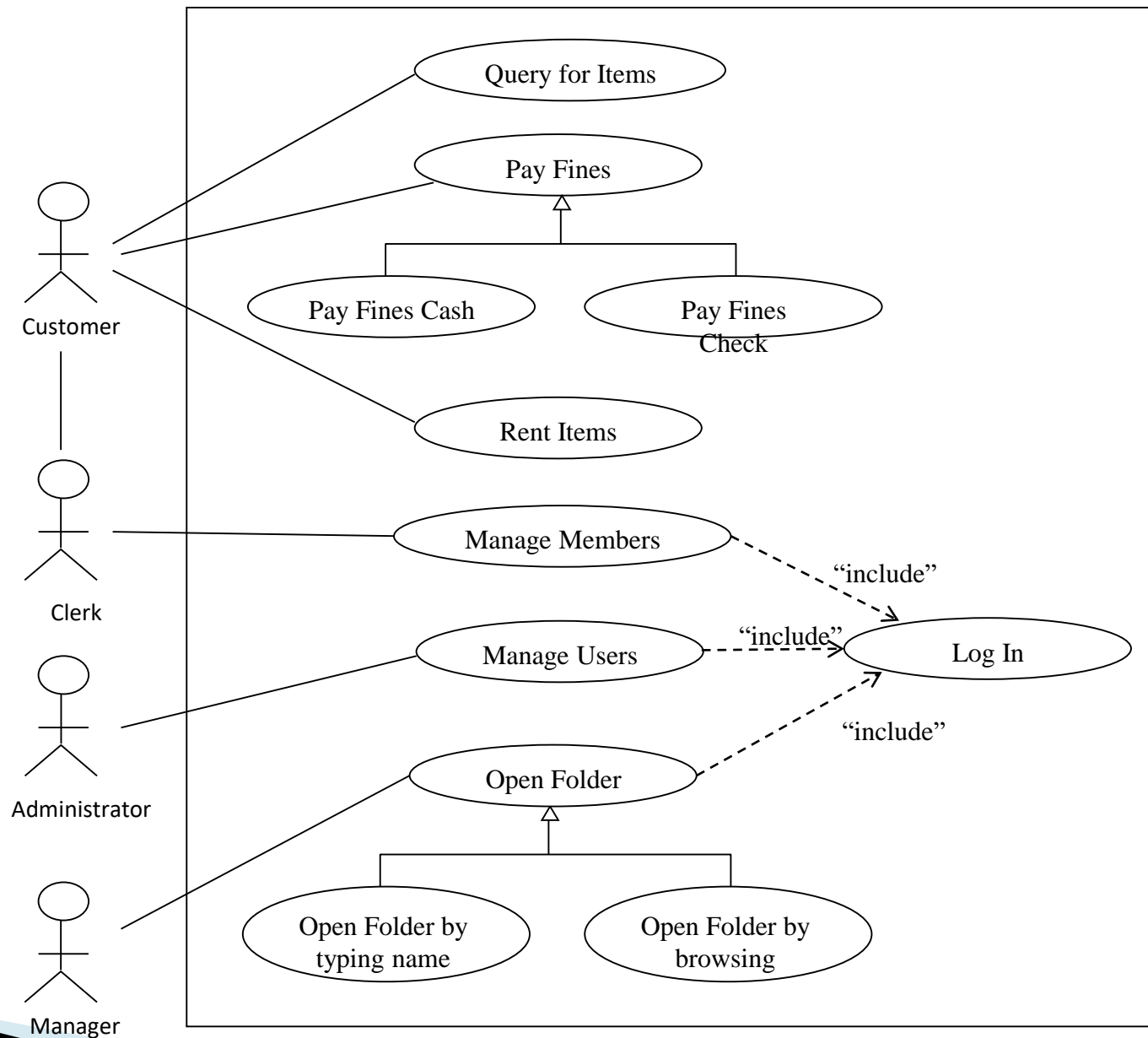
The main objective of this Section is to know how to model users' tasks using use case diagrams and detailed descriptions of use cases. In this section you will learn about the following:

- Scenarios: used to formulate the actual system requirements

- Use-Cases: describing how the user will use the system

- Explain the sections of a use case text

- Provide the student with a template for writing the use case description

- Introduce the use case and context diagrams

- Describe the artifacts used with a Use Case

- Explain a logic via artifacts decision tables or decision tree

- Explain use cases relations, e.g., include, extend, generalise

Scenarios

- Scenarios are real-life examples of how a system can be used.

- They should include

  - A description of the starting situation;

  - A description of the normal flow of events;

  - A description of what can go wrong;

  - Information about other concurrent activities;

  - A description of the state when the scenario finishes.

Scenarios

❑ Scenario

  ○ be a specific sequence of actions and interactions between actors and the system; it is also called a use case instance.

  ○ It is one particular story of using a system, or one path through the use case.

  ○ for example, the scenario of successfully purchasing items with cash, or the scenario of failing to purchase items because of a credit payment denial.

I would like a Book of stamps, please.

Ok. Will that be all?

Yes.

That will be SR7

Here is SR10

Thanks. Here are your stamps and your change

Use cases:

<u>describing how the user will use the system</u>

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.

- A use case is a typical sequence of actions that a user performs in order to complete a given task.

- A set of use cases should describe all possible interactions with the system.

- Explains everything in the user's language

Use case answers the question: *What is the system supposed to do for the user?*

Scenarios

- A <u>scenario</u> is an *instance* of a use case

  - A *specific occurrence* of the use case

    - a specific actor …
    - at a specific time …
    - with specific data.

Use cases:
describing how the user will use the system

- The objective of use case analysis is to model the system from the point of view of

  - … how users interact with this system

  - … when trying to achieve their objectives.

  - It is one of the key activities in requirements analysis

- A use case model consists of

  - a set of use cases

  - an optional description or diagram indicating how they are related

Use case model = the set of all use cases

Example: LIBSYS use cases
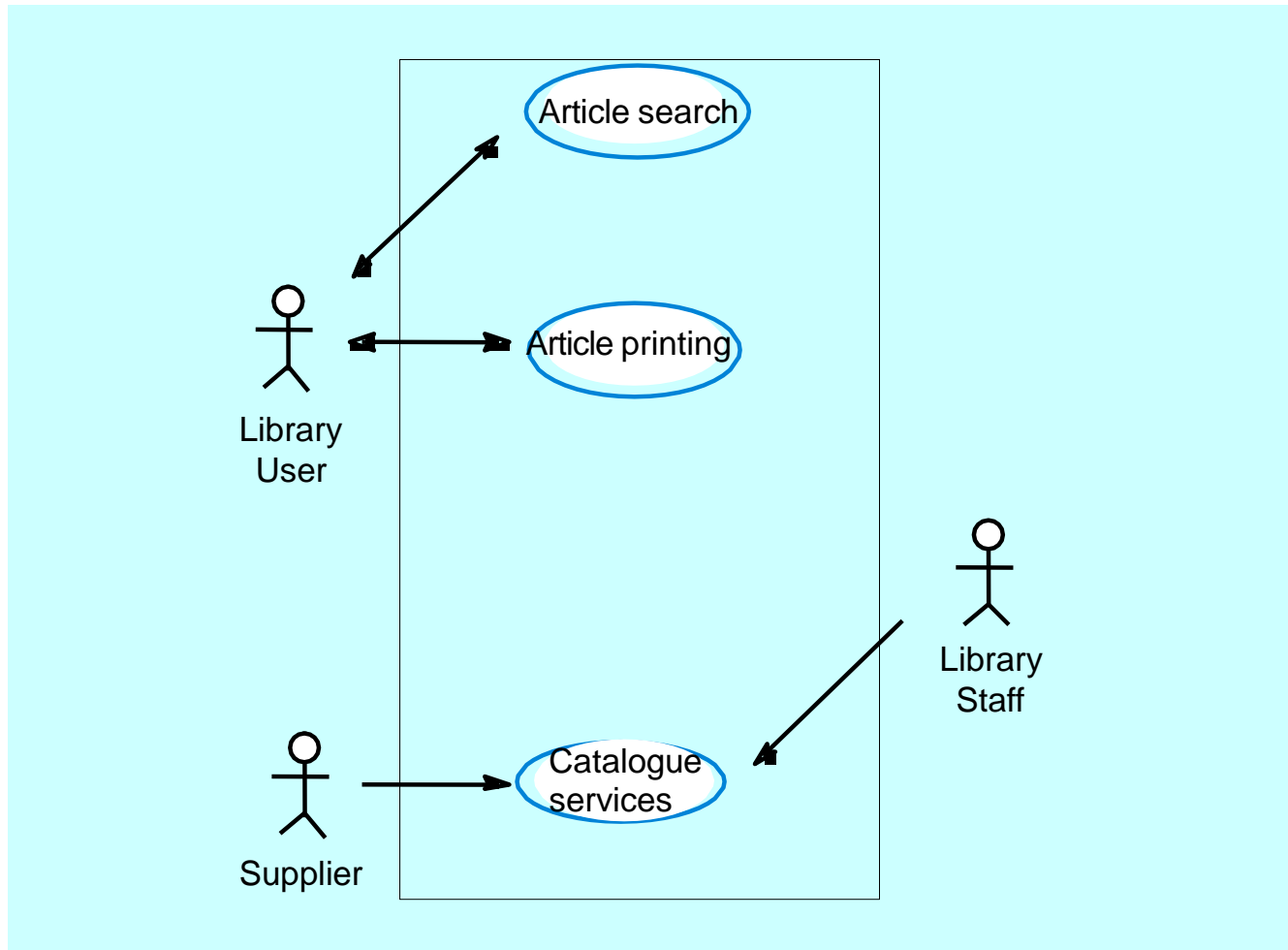
Use Case Description

Use case *text* provides the detailed description of a particular use case

Use Case ID:	Give each use case a unique integer sequence number identifier.		
Use Case Name:	Start with a verb.		
Created By:		Last Updated By:	
Date Created:		Date Last Updated:	

Actors:	Calls on the system to deliver its services.
Description:	"user-goal" or "sub-function"
Stakeholders and Interests:	Who cares about this use case, and what do they want?
Trigger:	Identify the event that initiates the use case.
Pre-conditions:	What must be true on start, and worth telling the reader?
Post-conditions:	Describe the state of the system at the conclusion of the use case execution.
Normal Flow:	A typical, unconditional happy path scenario of success.
Alternative Flows (Extensions):	Alternative scenarios of success or failure.
Priority:	Indicate the relative priority of implementing the functionality required to allow this use case to be executed.
Technology and Data Variations List	Varying I/O methods and data formats.
Special Requirements:	Related non-functional requirements.
Notes and Issues:	Such as open issues.

Use Case Description

- **Use Case Identification**
  - **Use Case ID**
    Give each use case a unique integer sequence number identifier.
  - **Use Case Name**
    State a concise, results-oriented name for the use case. These reflect the tasks the user needs to be able to accomplish using the system. Include an action verb and a noun.
  - **Use Case History**
    - Created By
    - Date Created
    - Last Updated By
    - Date Last Updated
  - **Actors**
    An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks. Name the actor that will be initiating this use case and any other actors who will participate in completing the use case.
  - **Description**
    Provide a brief description of the reason for and outcome of this use case.
  - **Stakeholders and Interests**
    Who cares about this use case, and what do they want?
  - **Trigger**
    Identify the event that initiates the use case. This could be an external business event or system event that causes the use case to begin, or it could be the first step in the normal flow.

Use Case Description

Use Case Definition

- **Pre-conditions**

  List any activities that must take place, or any conditions that must be true, before the use case can be started. Number each precondition. Examples:
  - User's identity has been authenticated.
  - User's computer has sufficient free memory available to launch task.

- **Post-conditions**

  Describe the state of the system at the conclusion of the use case execution. Number each post-condition. Examples:
  - Price of item in database has been updated with new value.

- **Normal (basic) Flow of events – Happy path – Successful path – Main Success Scenario**

  Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions. This dialog sequence will ultimately lead to accomplishing the goal stated in the use case name and description.

- **Alternative Flows (Extensions): Alternate scenarios of success or failure**

  Document other, legitimate usage scenarios that can take place within this use case separately in this section. State the alternative flow, and describe any differences in the sequence of steps that take place. Number each alternative flow in the form "X.Y", where "X" is the Use Case ID and Y is a sequence number for the alternative flow. For example, "5.3" would indicate the third alternative flow for use case number 5.

Use Case Description

- **Priority**

  Indicate the relative priority of implementing the functionality required to allow this use case to be executed. The priority scheme used must be the same as that used in the software requirements specification.

- **Technology and Data Variations List**

  Varying I/O methods and data formats.

- **Special Requirements**

  Identify any additional requirements, such as nonfunctional requirements, for the use case that may need to be addressed during design or implementation. These may include performance requirements or other quality attributes.

- **Notes and Issues**

  List any additional comments about this use case or any remaining open issues or TBDs (To Be Determineds) that must be resolved. Identify who will resolve each issue, the due date, and what the resolution ultimately is.

Text and Diagrams

- Use case *text* provides the detailed description of a particular use case

- Use case *diagram* provides an overview of interactions between actors and use cases

Example: Point of Sale – Problem Description

❑ The Point-of-Sale terminal is a computerized system used to record sales and handle payments; it is typically used in a retail store. It includes hardware components such as a computer and bar code scanner, and software to run the system.

❑ It interfaces to various service applications, such as a third-party tax calculator and inventory control. These systems must be relatively fault-tolerant; that is, even if remote services are temporarily unavailable (such as the inventory system), they must still be capable of capturing sales and handling at least cash payments (so that the business is not crippled).

Example: Point of Sale – Problem Description

❏ A POS system increasingly must support multiple and varied client-side terminals and interfaces. These include a thin-client Web browser terminal, a regular personal computer with something like a Java Swing graphical user interface, touch screen input, wireless PDAs, and so forth.

❏ Furthermore, we are creating a commercial POS system that we will sell to different clients with disparate needs in terms of business rule processing. Each client will desire a unique set of logic to execute at certain predictable points in scenarios of using the system, such as when a new sale is initiated or when a new line item is added.

❏ Therefore, we will need a mechanism to provide this flexibility and customization. Using an iterative development strategy, we are going to proceed through requirements, object-oriented analysis, design, and implementation.

- <u>Actors</u>:
  - Cashier
  - Customer
  - Supervisor

- <u>Choosing actors</u>:
  - Identify system boundary
  - Identify entities, human or otherwise, that will interact with the system, from outside the boundary.
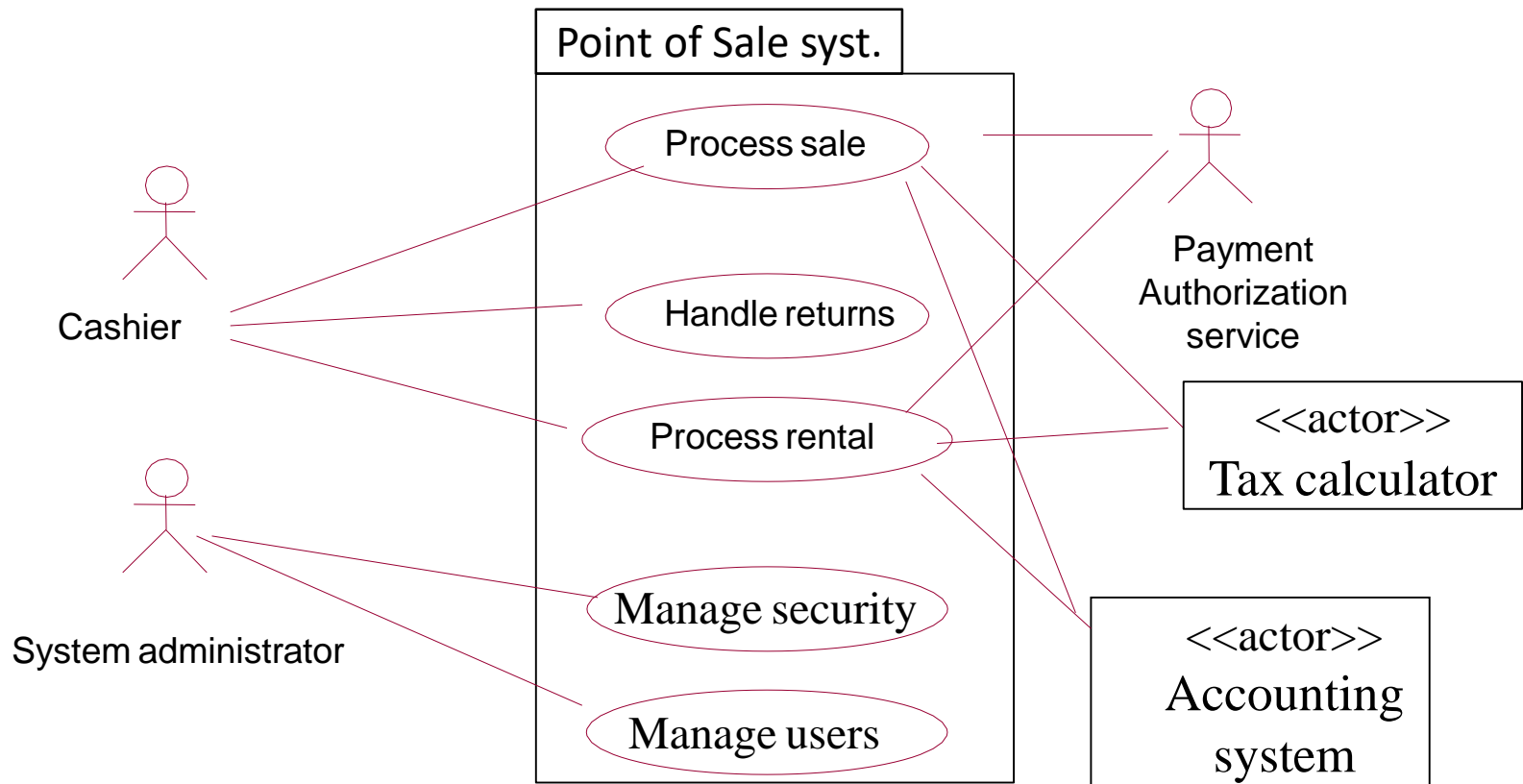
Example: Point of Sale – Actor-Goal List

Actor-Goal List

Sales activity system is a remote application that will frequently request sales data from each POS node on the network

Actor	Goal
Cashier	• Process sales • Process returns • Cash in • Cash out
Manager	• Start up • Shut down
System administrator	• Add users • Modify users • Delete users • Manage security
Sales activity system **(external computer sys)**	• Analyse sales & performance data

Example: Point of Sale – Use Case Diagram

Point of Sale syst.

- Process sale
- Handle returns
- Process rental
- Manage security
- Manage users

Cashier

System administrator

Payment Authorization service

<<actor>>
Tax calculator

<<actor>>
Accounting system

Use Case Diagram: illustrates a set of use cases for a system.

Exercise: movie ticket machine

- Problem Description

Implement a simple movie ticket vending machine. The movie theater that will use the machine has only one movie and one show time each day. Every morning, the theater manager will turn on the ticket machine, and it will ask him for the name of the movie and the ticket price that day. It will also ask how many seats are in the theater (so it won't sell too many tickets).

When a customer walks up to the ticket machine, he will see the name of the movie, the time, and the ticket price displayed. There is a slot to insert money, a keypad of buttons to enter a number into the "Number of Tickets" field, and a "Buy" button.

Printed tickets come out of a slot at the bottom of the machine. Above the ticket slot is a message display (for error messages like "Please enter more money or request fewer tickets" or "SOLD OUT!"). An additional display shows the customer's balance inside the machine.

Finally, there is a "Return Change" button so the customer can get his unspent money back.

- Who or what are the actors?

- What are the use cases (goals of actors)?

Use case diagram for Movie Ticket Machine
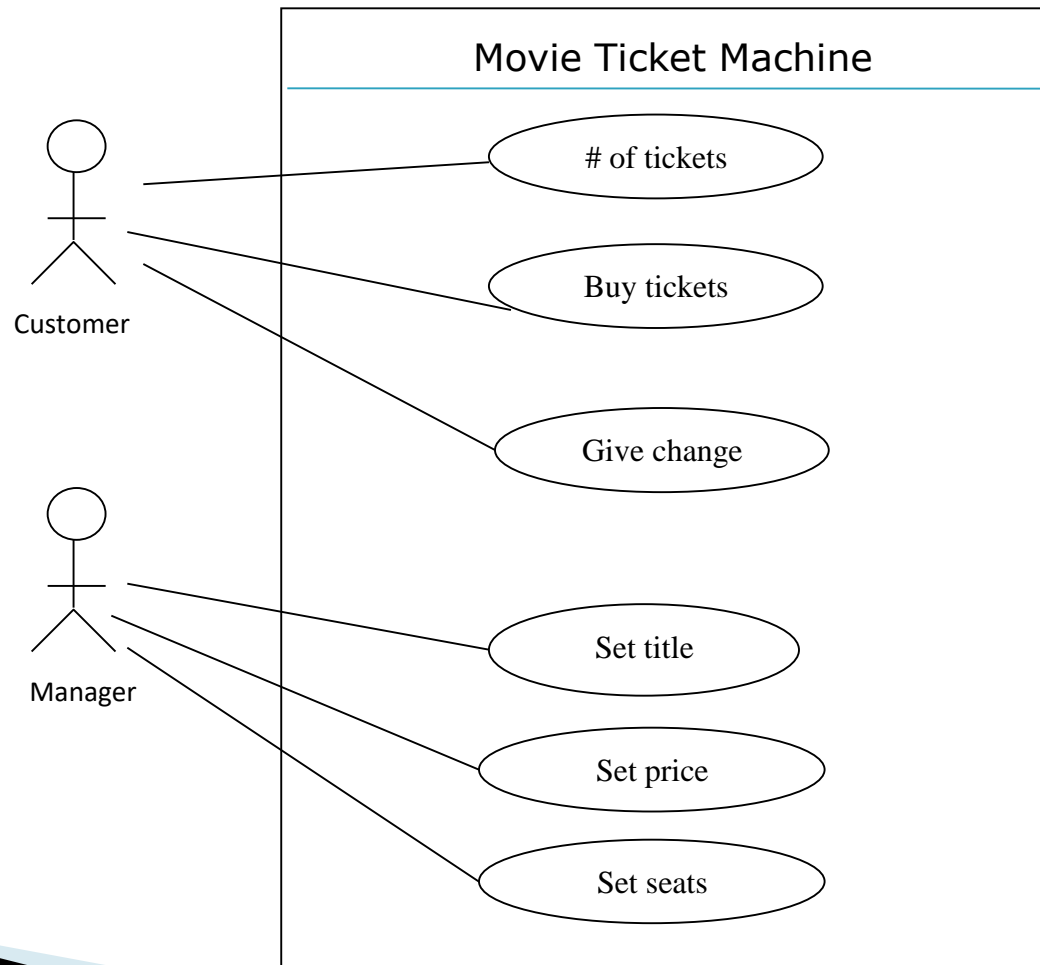
Identification of Use Cases

Use cases for Manager

- Use case: Set title
- Actors: Manager, Machine
- 1. Manager requests a change of movie title
- 2. Machine asks manager for new movie title
- 3. Manager enters movie title

- Use case: Set price
- Actors: Manager, Machine
- 1. Manager requests a change of ticket price
- 2. Machine asks manager for new price for movie title
- 3. Manager enters ticket price
- Alternatives: Invalid price
- If manager enters price below SR5 or greater than SR50
- 3a. Machine asks manager to reenter price

- Use case: Set seats
- Actors: Manager, Machine
- 1. Manager requests a change in number of seats
- 2. Machine asks manager for number of seats in theatre
- 3. Manager enters number of seats
- Alternatives: Invalid number of seats
- If manager enters number less than 20 or greater than 999
- 3a. Machine asks manager to reenter number of seats

Identification of Use Cases

Use cases for Customer

- Use case: # of tickets
- Actors: Customer, Machine
- 1. Customer enters number of tickets
- 2. Machine displays total balance due
- Alternative: Customer wants zero tickets
- At step 1, customer enters zero tickets
- 1a. Display thank you message
- 1b. Set balance to $0.0

- Use case: Return change to customer
- Actors: Customer, Machine
- 1. Customer requests change
- 2. Machine dispenses money
- 3. Machine updates customer balance

- Use case: Buy tickets
- Actors: Customer, Machine
- 1. Customer requests tickets
- 2. Machine tells customer to put balance due in money slot
- 3. Customer enters money in money slot
- 4. Machine updates customer balance
- 5. Customer requests tickets
- 6. Machine prints tickets
- 7. Machine updates number of seats
- Alternative: Insufficient seats
- At step 1, if number of tickets requested is less than available seats,
- 1a. Display message and end use case
- Alternative: Insufficient funds
- At step 5, if money entered < total cost,

UML Use Case Driven Object

Use case diagram for Club Sport

The ClubRiyadh Sport has decided to implement an electronic card system for its subscriber, so that subscribers can use their K-cards to access secure areas, and also as a debit card, linked to an account into which subscribers can deposit money to be used to pay club fees. For the initial release of the system, this will be limited to a few club usages: equipment rental at the sports centre, beverage fees, and library fees at club libraries. The system will keep a usage record for each K-card.

Identify use cases by providing the actors, use case names. Draw the use case diagram.

Use case diagram for Club Sport
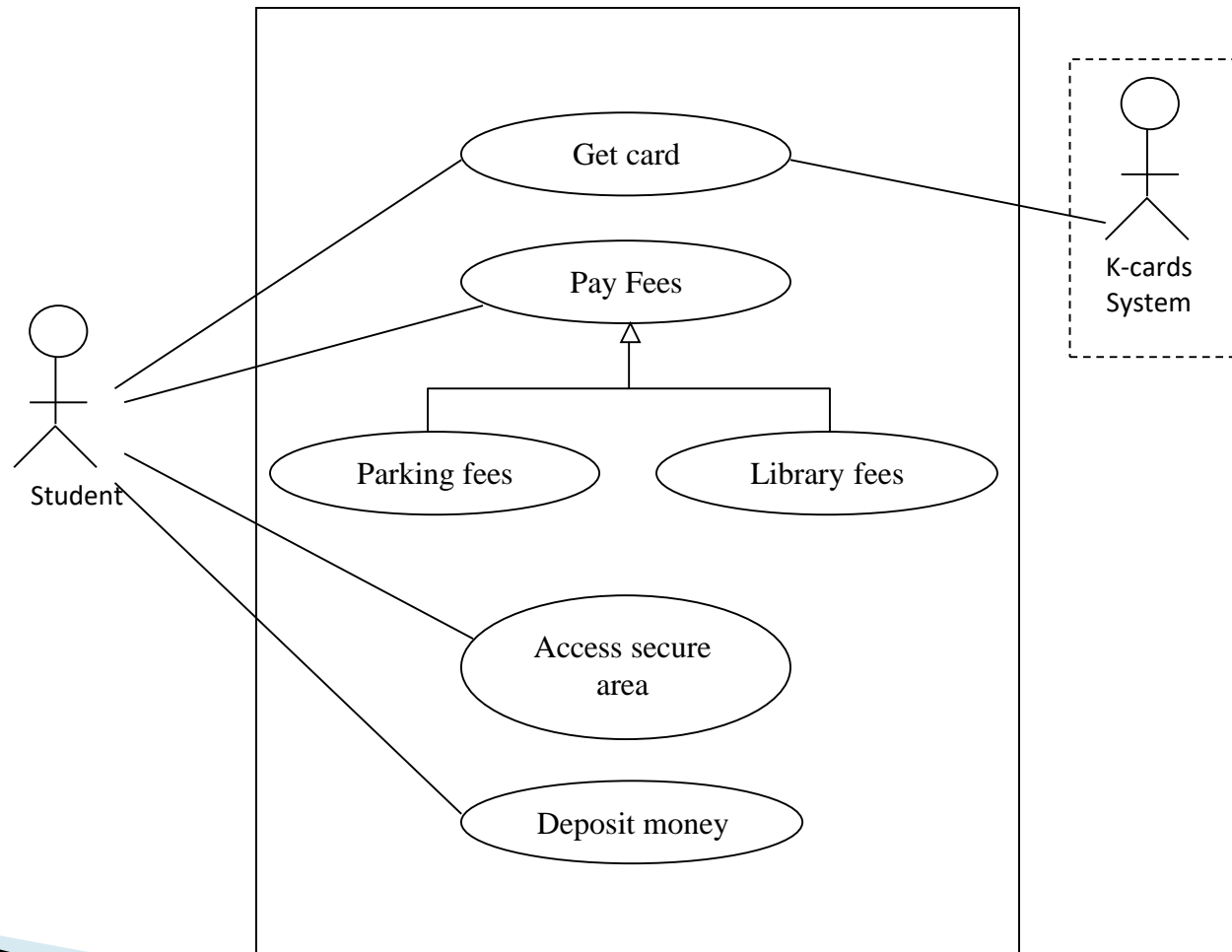
Identifying Actors and Use Cases

- **Observation**
  - Read documents and discuss requirements with users
  - Shadowing important potential users as they do their work
    - ask the user to explain everything he or she is doing
  - Session video taping
- **Interviewing**
  - Conduct a series of interviews
    - Ask about specific details
    - Ask about the stakeholder's vision for the future
    - Ask if they have alternative ideas
    - Ask for other sources of information
    - Ask them to draw diagrams

Use Case Best Practices

- **Method 1 - Actor based**
  - Identify the actors related to the system
  - Identify the processes these actors initiate or participate in

- **Method 2 - Event based**
  - Identify the external events that a system must respond to
  - Relate the events to actors and use cases

- **Method 3 – Goal based**
  - [Actors have goals.]
  - Find user goals. [Prepare actor-goal list.]
  - Define a use case for each goal.

Use Case Best Practices

Identify user goal-level UCs-

- that serve each goal of the primary actor

- Use EBP guide lines if a UC is at a suitable level: goals hierarchy and sub goals

Find the user goals:

- Ask: what are your goals?

- Do not ask what are the use cases

- Do not ask what do you do?

Use Case Best Practices

- Define a use case for each goal

- Answers to the question **'what are your goals ?'**

combined with a question to move higher up the goal hierarchy **('what is the goal of this goal') will:**

  - open up the vision for new & improved solutions
  - focus on the business
  - get to the heart of what the stakeholders want from the system

- Goals may be composed of many sub goals (sub functional goals) leading to a primary UC and sub UCs

Finding Primary Actors Goals & Use Cases

Use cases are defined to satisfy the goals of the primary actors:
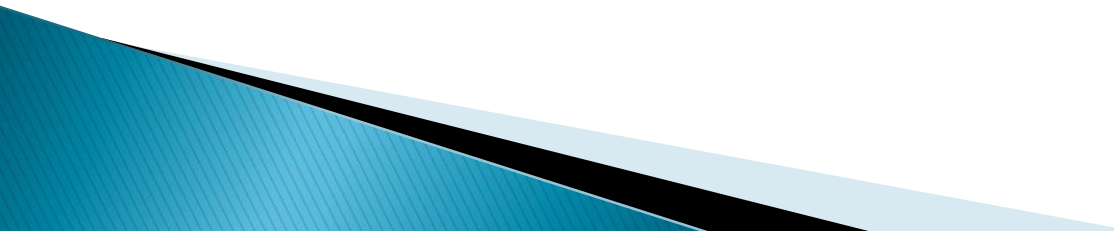
1) Choose the system boundary

2) Identify primary actors

3) For each primary actor, identify their user goals.

- Raise goals to the highest goal level that satisfy the EBP guidelines

4) Name a UC according to its goal:
- usually, a 1:1 relationship
- common exception to 1:1 relationship: **CRUD** (create, retrieve, update, delete) goals into one CRUD UC called "manage"

OO domain modeling with UML class diagrams and CRC cards

What is a Domain Model?

- ❑ Illustrates meaningful conceptual classes in problem domain

- ❑ Represents real-world concepts, not software components

- ❑ A diagram (or set of diagrams) which represents real world *domain objects*
    - *'conceptual classes'*

- ❑ *Not a set of diagrams describing software classes, or* software

  objects with responsibilities

Why make a Domain Model?

- to understand what concepts need to be modelled by our system, and how those concepts relate

- a springboard for designing software objects

What Domain Model should it show?

- conceptual classes

- associations between conceptual classes

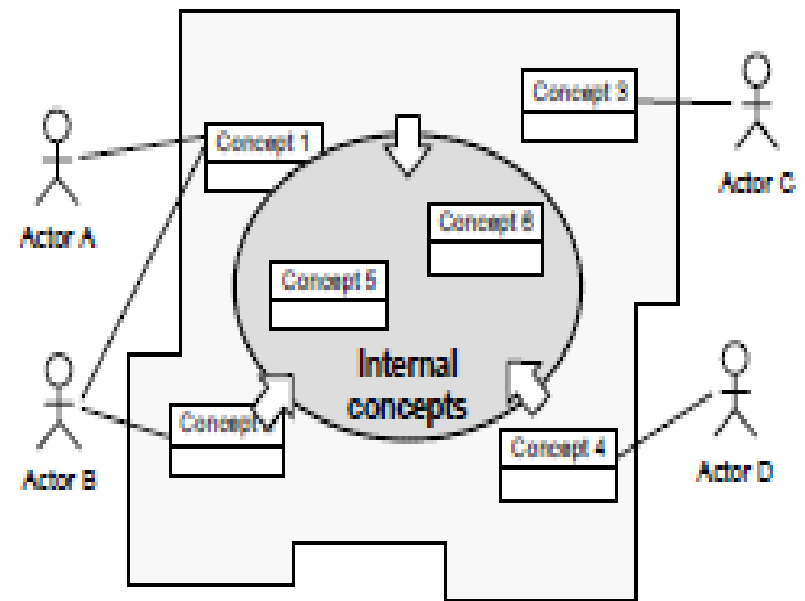- attributes of conceptual classes

Building the Domain Model

A useful strategy for building a domain model is to start with:

▸ the "boundary" concepts that interact directly with the actors

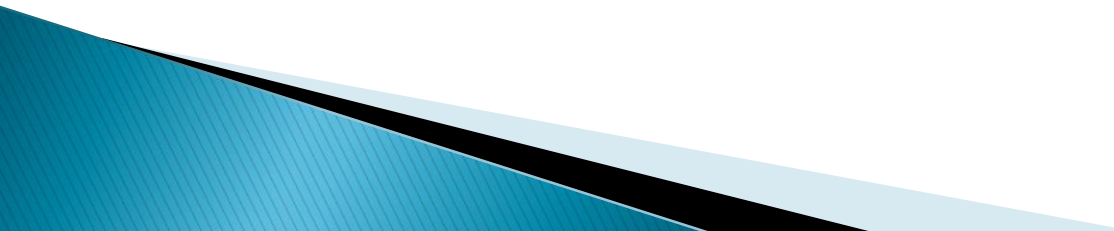▸ and then identify the internal concepts

Step 1: Identifying the boundary concepts
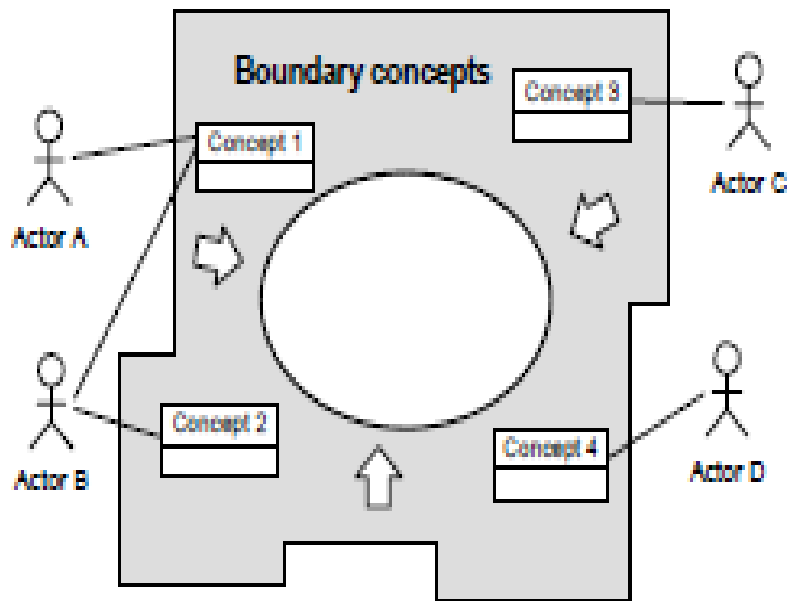
Step 2: Identifying the internal concepts
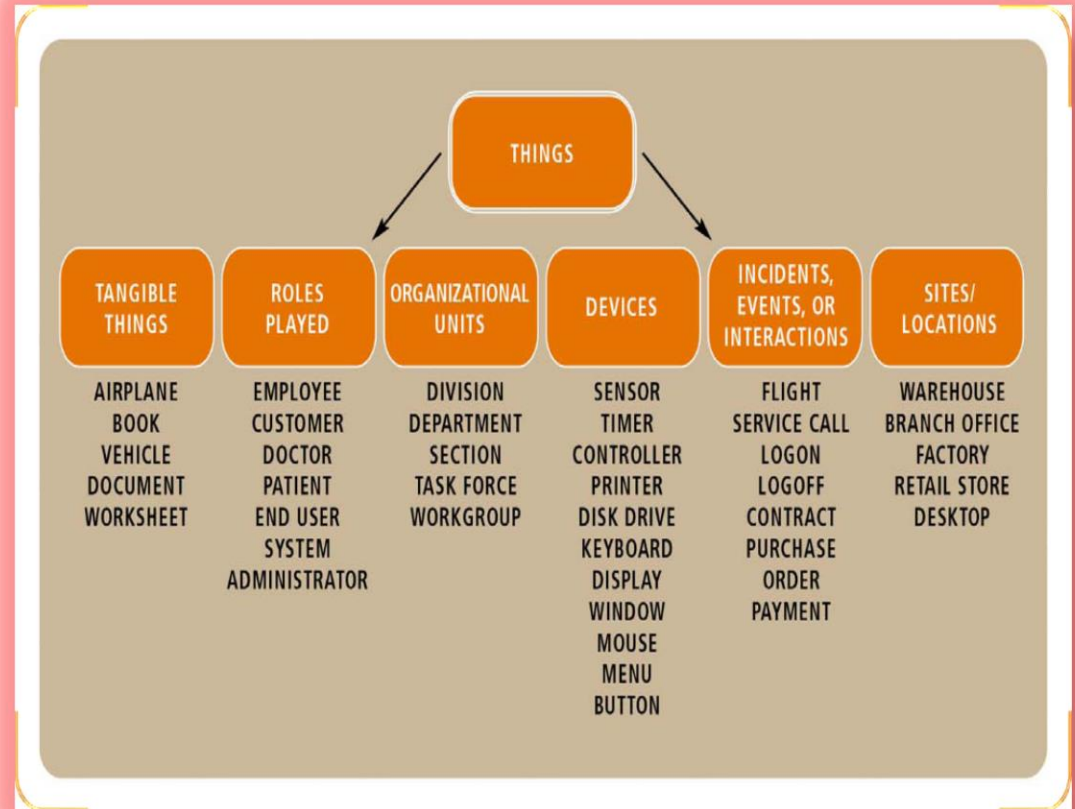
Steps to create a Domain Model

- ❑ Identify Candidate Conceptual classes

- ❑ Draw them in a Domain Model

- ❑ Add associations necessary to record the relationships that must be retained

- ❑ Add attributes necessary for information to be preserved

Identify conceptual classes

- Three strategies to find conceptual classes
  - Reuse or modify existing models
    - There are published, well-crafted domain models and data models for common domains: inventory, finance, health..
    - Books: *Analysis patterns* by Martin Fowler, *Data Model Patterns* by David Hay, *Data Model Resource Book* by Len Silverston
  - Use a category list
  - Identify noun phrases

Use a category list

- Finding concepts using the concept category list :

  - Physical objects: register, airplane, blood pressure monitor

  - Places: airport, hospital

  - Catalogs: Product Catalog

  - Transactions: Sale, Payment, reservation

THINGS

TANGIBLE THINGS	ROLES PLAYED	ORGANIZATIONAL UNITS	DEVICES	INCIDENTS, EVENTS, OR INTERACTIONS	SITES/ LOCATIONS
AIRPLANE	EMPLOYEE	DIVISION	SENSOR	FLIGHT	WAREHOUSE
BOOK	CUSTOMER	DEPARTMENT	TIMER	SERVICE CALL	BRANCH OFFICE
VEHICLE	DOCTOR	SECTION	CONTROLLER	LOGON	FACTORY
DOCUMENT	PATIENT	TASK FORCE	PRINTER	LOGOFF	RETAIL STORE
WORKSHEET	END USER	WORKGROUP	DISK DRIVE	CONTRACT	DESKTOP
	SYSTEM ADMINISTRATOR		KEYBOARD	PURCHASE	
			DISPLAY	ORDER	
			WINDOW	PAYMENT	
			MOUSE		
			MENU		
			BUTTON		

Identify conceptual classes from noun phrases

❏ Finding concepts using Noun Phrase identification in the textual description of the domain :

Identify the nouns and noun phrases in textual descriptions of a domain, and consider them as candidate conceptual classes or attributes.

❏ Noun phrases may also be attributes or parameters rather than classes:

➢ If it stores state information or it has multiple behaviors, then it's a class
➢ If it's just a number or a string, then it's probably an attribute

Identify conceptual classes from noun phrases

The domain model is a visualization of noteworthy ▸ domain concepts and vocabulary. Where are those terms found?

Some are in the use cases. ▸

Others are in other documents, or the minds of ▸ experts. In any event, use cases are one rich source to mine for noun phrase identification.

Identifying objects

- Look for **nouns** in the SRS (System Requirements Specifications) document

- Look for **NOUNS** in use cases descriptions

- A **NOUN** may be
  - Object
  - Attribute of an object

Identifying Operations 'methods'

- Look for verbs in the SRS (System Requirements Specifications) document

- Look for **VERBS** in use cases descriptions

- A **VERB** may be
  - translated to an **operation** or set of operations

  - A method is the code implementation of an operation.

Example POS

For example, the current scenario of the *Process Sale* use case can be used.

Main Success Scenario (or Basic Flow):

•*Customer* arrives at *a POS checkou*t with *goods* and / or **services** to purchase.
•*Cashier* starts a new *sale***.**
•**Cashier** enters *item identifier***.**
•System records *sale line item* and presents *item description***,** *price***,** and running *total***.** Price calculated from a set of price rules.
Cashier repeats steps 2 - 3 until indicates done.
•System presents total with *taxes* calculated.
•Cashier tells Customer the total, and asks for *payment***.**
•Customer pays and System handles payment.
•System logs the completed *sale* and sends sale and payment information to the external *Accounting* (for accounting and *commissions***)** and *Inventory* systems (to update inventory).
•System presents *receipt***.**
•Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

Paying by cash:
•Cashier enters the cash *amount tendered***.**
•System presents the *balance due***,** and releases the *cash drawer***.**
•Cashier deposits cash tendered and returns balance in cash to Customer.
•System records the cash payment.

Case Study: POS Domain

From the category list and noun phrase analysis, a list is generated of candidate conceptual classes for the domain. Since this is a business information system, I'll focus first on the category list guidelines that emphasize business transactions and their relationship with other things.

Sale	Cashier
CashPayment	Customer
SalesLineItem	Store
Item	ProductDescription
Register	ProductCatalog
Ledger	

Example: Identify conceptual classes from noun phrases

Consider the following problem description, analyzed for Subjects, Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.
    SC   V   R O   O A   O A

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.
              S R   V   O A   V   O A   V   O A

Checking the balance simply displays the account balance.
    S M   O A   V   O A

Depositing asks the customer to enter the amount, then updates the account balance.
    M S   V   O R   V   O A   V   O A

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,
    S M A O   V   O R   O A   V   S C V   O A

the account balance is updated.   The ATM prints the customer's account balance on a receipt.
        O A   V   C S   V   O A   O

Analyze each **subject and object** as follows:
- Does it represent a person performing an action? Then it's an actor, '**R**'.
- Is it also a verb (such as 'deposit')? Then it may be a method, '**M**'.
- Is it a simple value, such as 'color' (string) or 'money' (number)?
  Then it is probably an attribute, '**A**'.
- Which NPs are unmarked? Make it '**C**' for class.

Verbs can also be classes, for example:
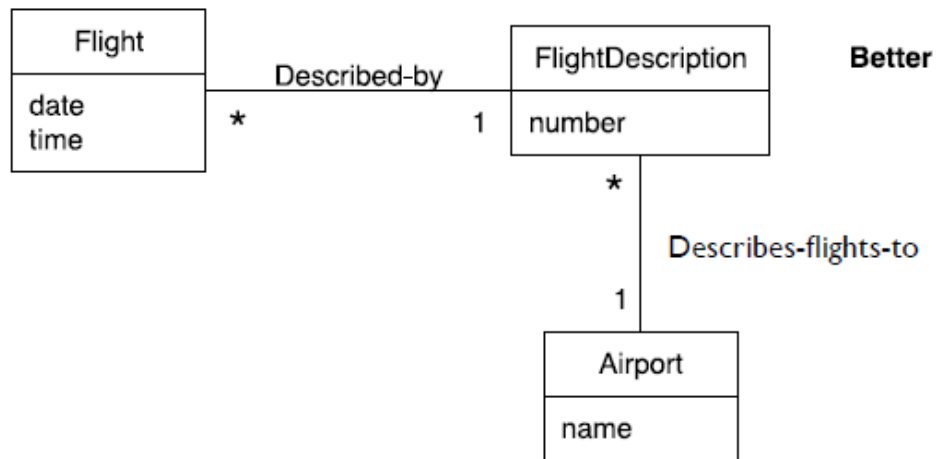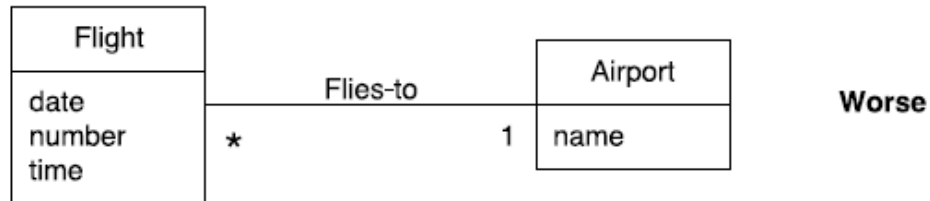- Deposit is a class if it retains state information

Example

Sale		Sale		Store
store	→			location

Flight		Flight		Airport
Airport	→			name
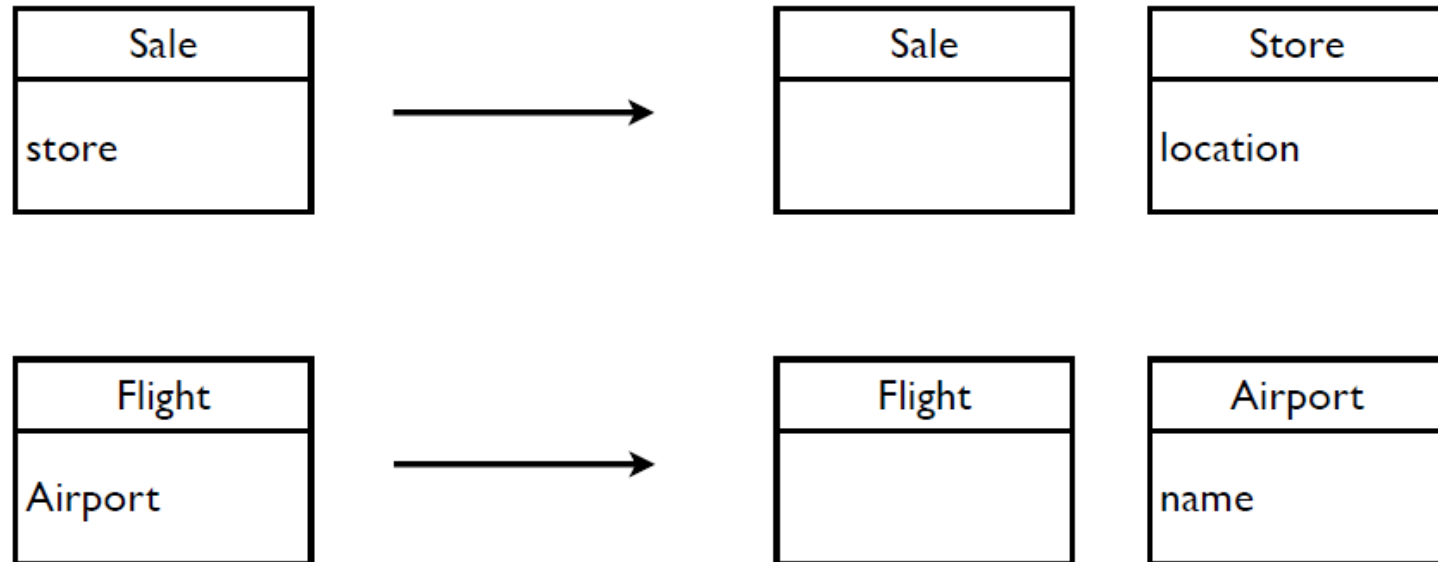
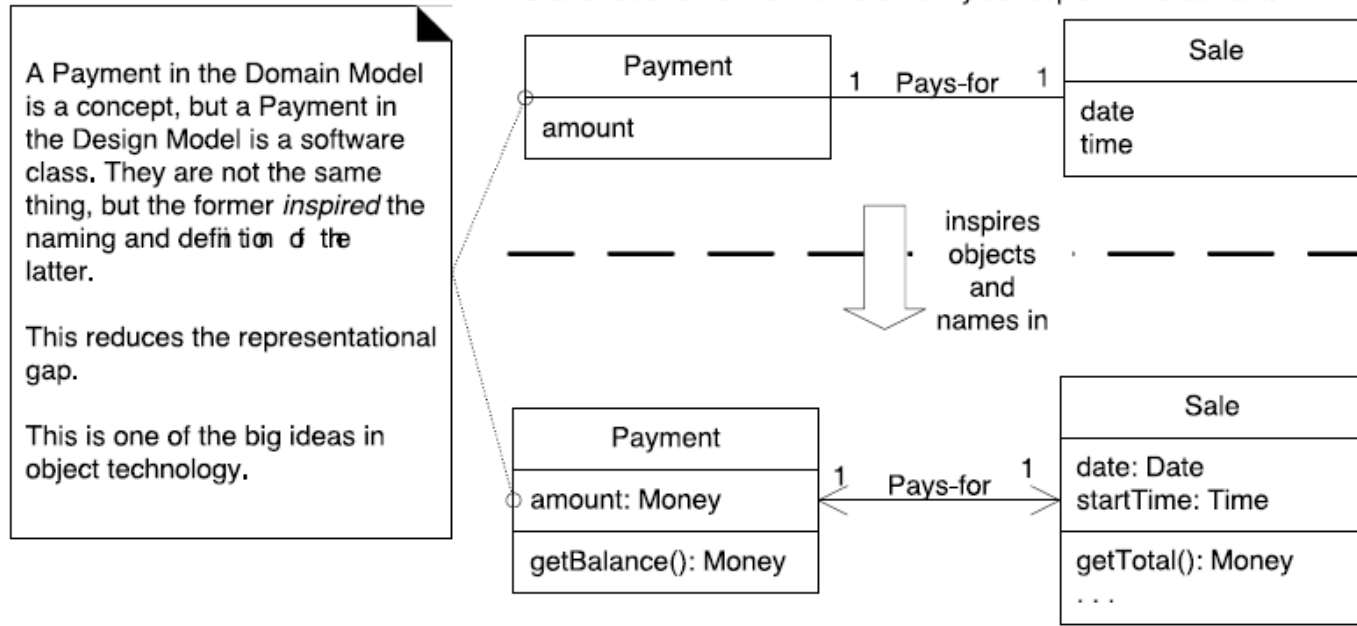from Ch 9 *Applying UML & Patterns (Larman 2004)*

both Store and Airport represent concepts of interest in their own right, so we model them as conceptual classes rather than attributes

Example

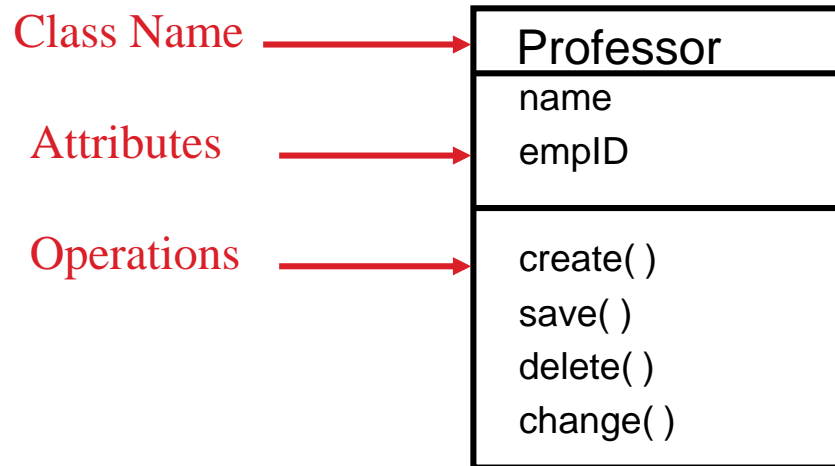from Ch 9 *Applying UML & Patterns (Larman 2004)*

Domain versus Design Models

UP Domain Model
Stakeholder's view of the noteworthy concepts in the domain.

A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and defini tion of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

Payment
amount

1   Pays-for   1

Sale
date
time

inspires
objects
and
names in

Payment
amount: Money
getBalance(): Money

1   Pays-for   1

Sale
date: Date
startTime: Time
getTotal(): Money
. . .

UP Design Model
The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

from Ch 9 *Applying UML & Patterns (Larman 2004)*

Class Compartments

- A class is comprised of three sections
  - The first section contains the class name
  - The second section shows the structure (attributes)
  - The third section shows the behavior (operations)

Class Name ────────▶

Attributes ────────▶

Operations ────────▶

Professor
name
empID
create( )
save( )
delete( )
change( )

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Relationships

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Relationships

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Relationships

What is Polymorphism?

```
                    ┌─────────────┐  name
                    │    Shape    │  getName( )
                    └─────────────┘  calculateArea( )
                           △
               ┌───────────┴───────────┐
  radius  ┌─────────┐         ┌─────────┐  side
calculateArea( ) │ Circle │         │ Square │  calculateArea( )
          └─────────┘         └─────────┘
```
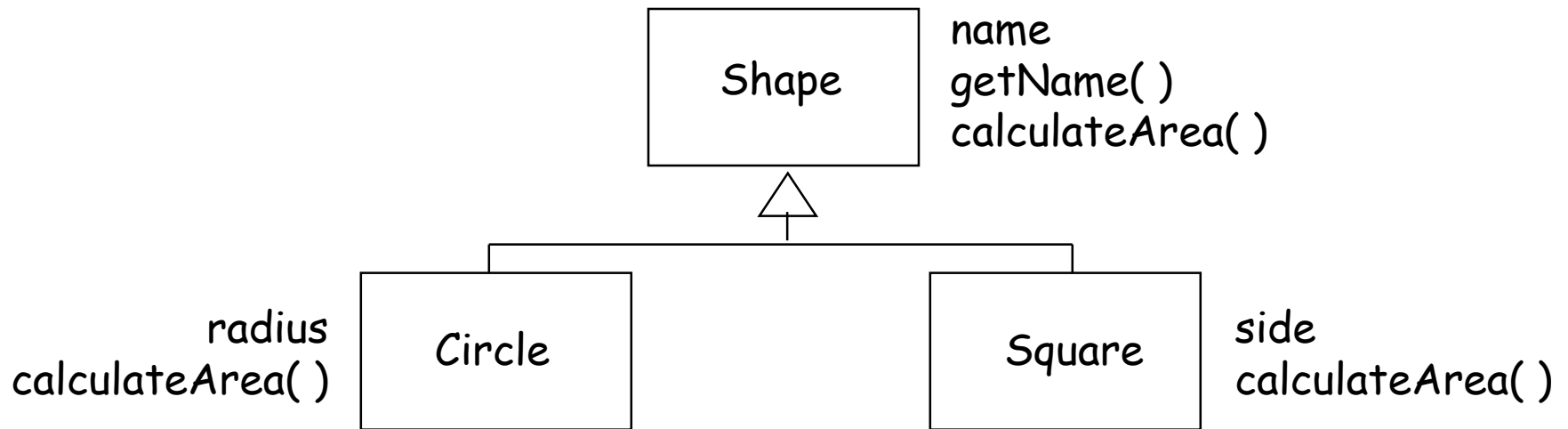
What is an Interface?

▸ Interfaces formalize polymorphism
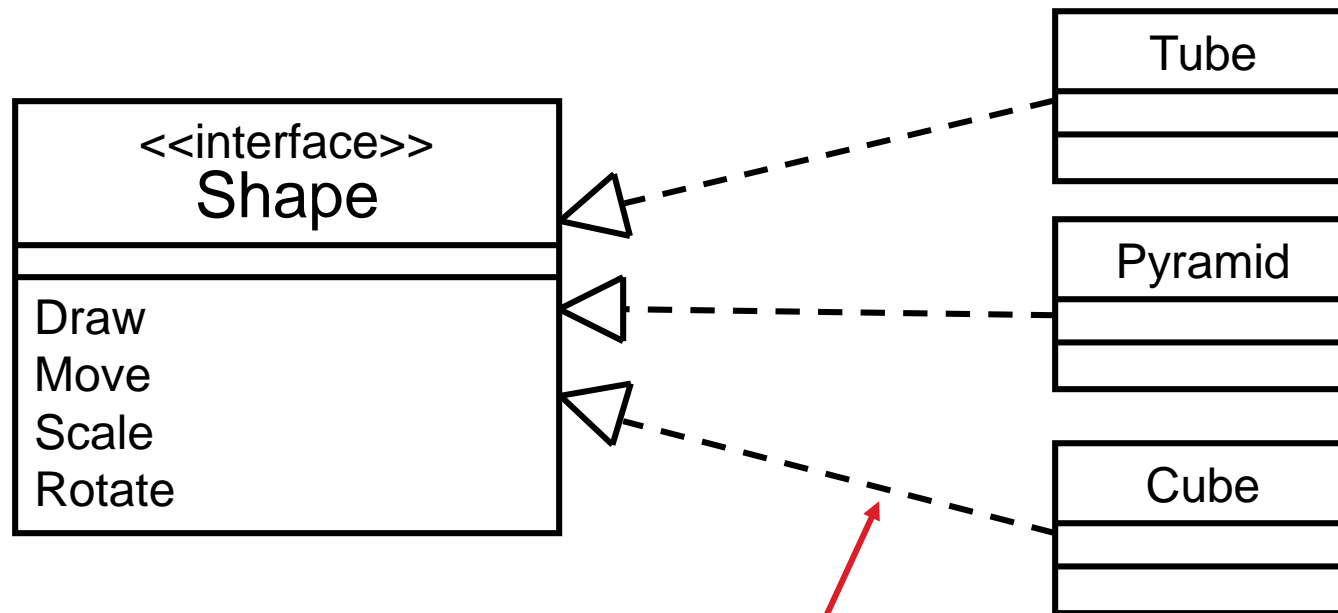
Realization relationship

(stay tuned for realization relationships)

Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Interface (Polymorphism)
- Relationships

Relationships

- Association
  - Aggregation
  - Composition
- Dependency
- Generalization
- Realization
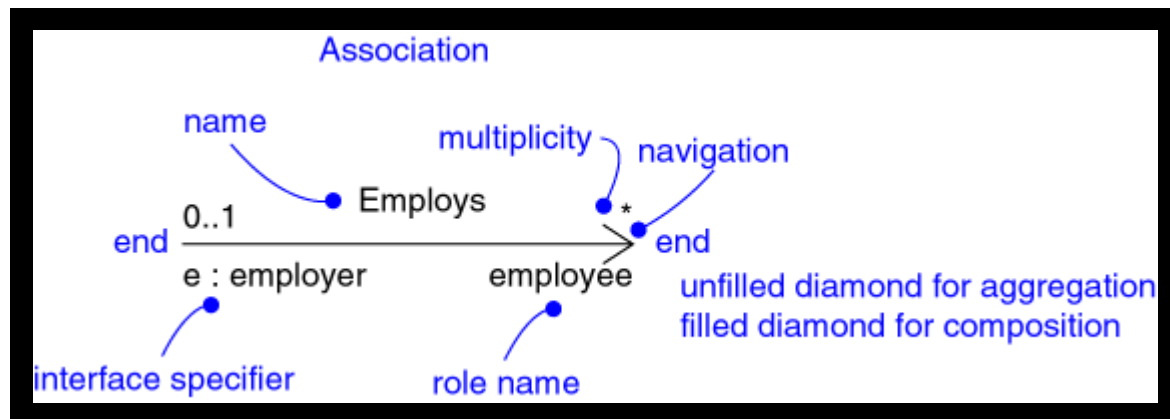
Relationships: Association

▸ Models a semantic connection among classes

Association Name

Professor — *Works for* — University

Association

Class

Role Names

Professor — Employee   Employer — University
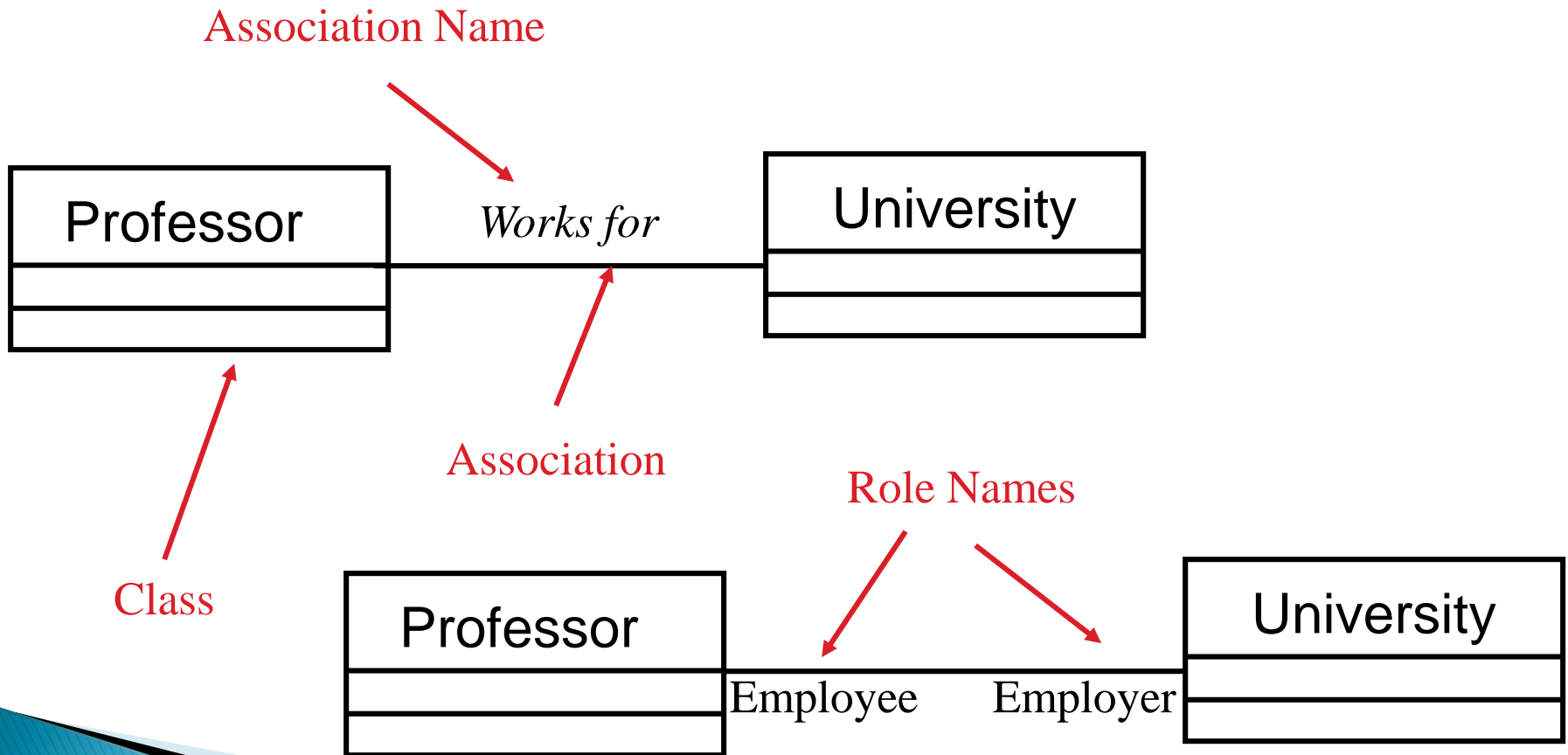
Relationships: Composition

▸ A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts
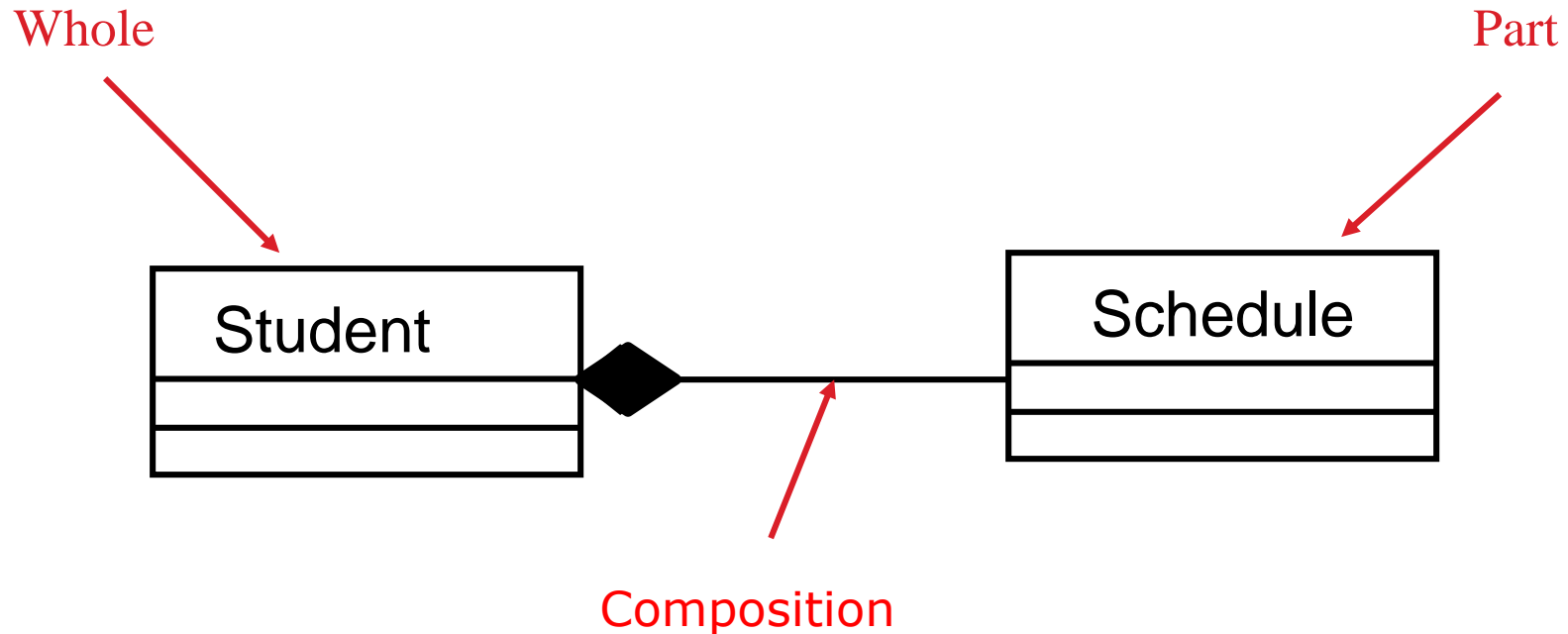
Relationships: Composition

Composition:

Chess Board | I --- 64 | Square
consists of ▸ | colour
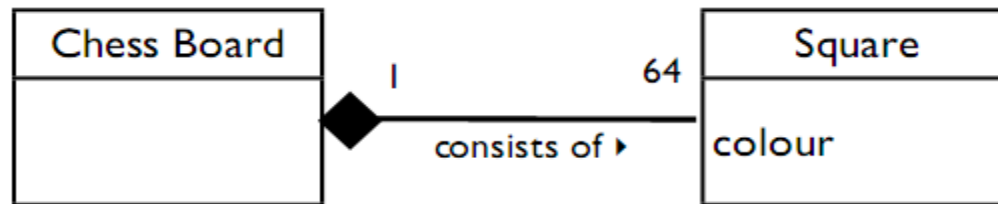
without the chess board, the square wouldn't exist...

Aggregation:

Class List | 0..3 --- 8..* | Student
contains ▸ | name
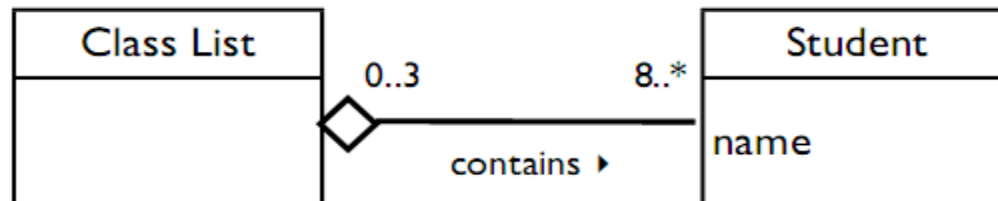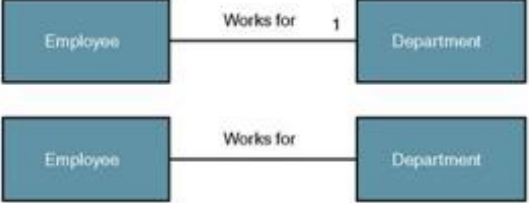
...but without the class list the student would

Association: Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationships
  ◦ The number of instances of one class related to ONE instance of the other class
  ◦ Specified for each end of the association
- Associations and aggregations are bi-directional by default, but it is often desirable to restrict navigation to one direction
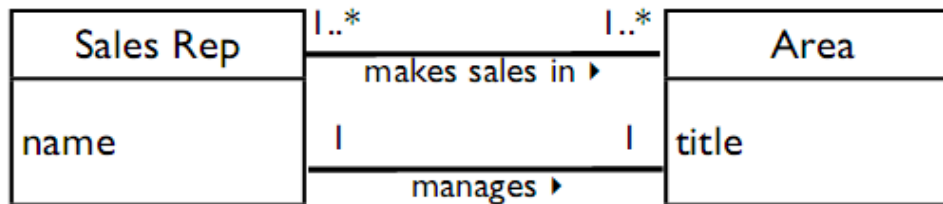 ◦ If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

Type of Multiplicity

Multiplicity – the minimum and maximum number of occurrences of one object/class for a single occurrence of the related object/class.

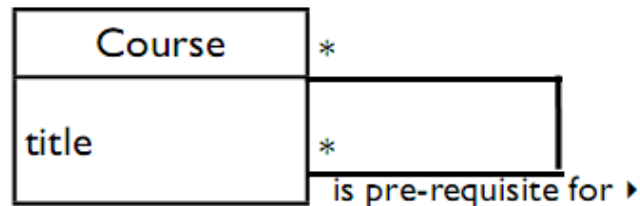| Multiplicity	UML Multiplicity Notation	Association with Multiplicity	Association Meaning
Exactly 1	1 or *leave blank*	Employee —Works for— 1 —Department Employee —Works for— Department	An employee works for one and only one department.
Zero or 1	0..1	Employee —Has— 0..1 —Spouse	An employee has either one or no spouse.
Zero or more	0..* or *	Customer —Makes— 0..* —Payment Customer —Makes— * —Payment	A customer can make no payment up to many payments.
1 or more	1..*	University —Offers— 1..* —Course	A university offers at least 1 course up to many courses.
Specific range	7..9	Team —Has scheduled— 7..9 —Game	A team has either 7, 8, or 9 games scheduled

Multiple & Reflexive Associations

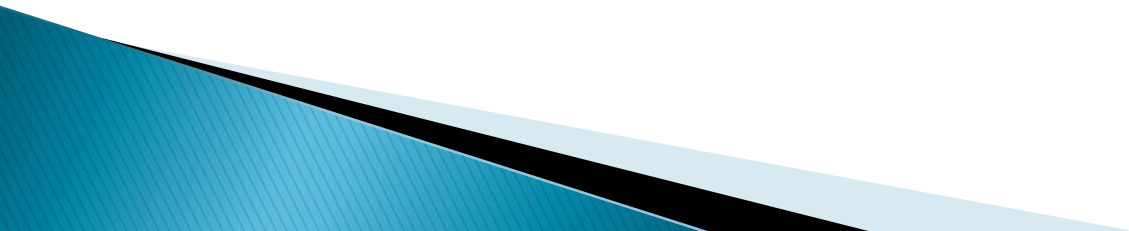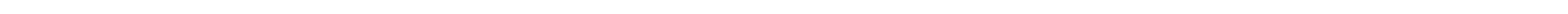- can two conceptual classes have multiple associations with each other, and can a class associate with itself?

yes,

Sales Rep	1..* makes sales in ▸ 1..*	Area
name	I manages ▸ I	title

and yes

Course	*
title	* is pre-requisite for ▸

but in each case it might be better to use generalisation and/or to add further conceptual classes to the model

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 in which a subclass inherits from one or more superclasses
  ◦ Single inheritance
  ◦ Multiple inheritance

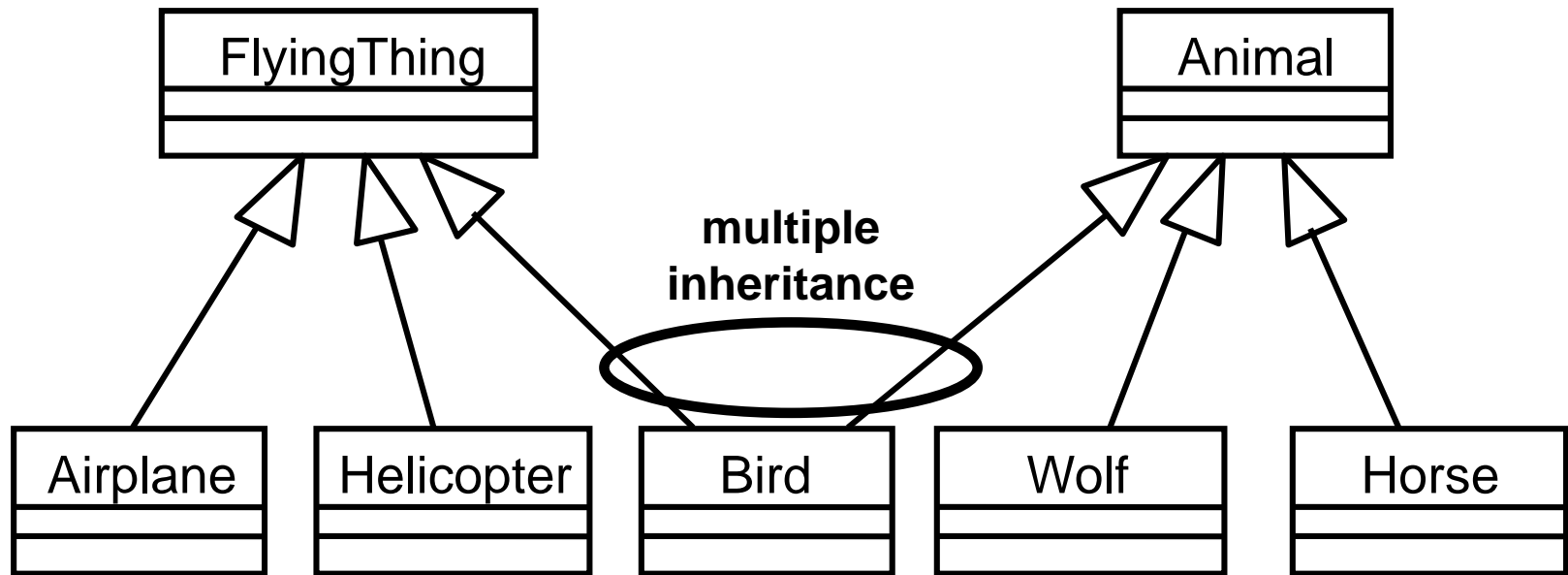▸ Generalization is an "is-a-kind of" relationship

Example: Single Inheritance

- sometimes conceptual classes are (sub) types of another class:
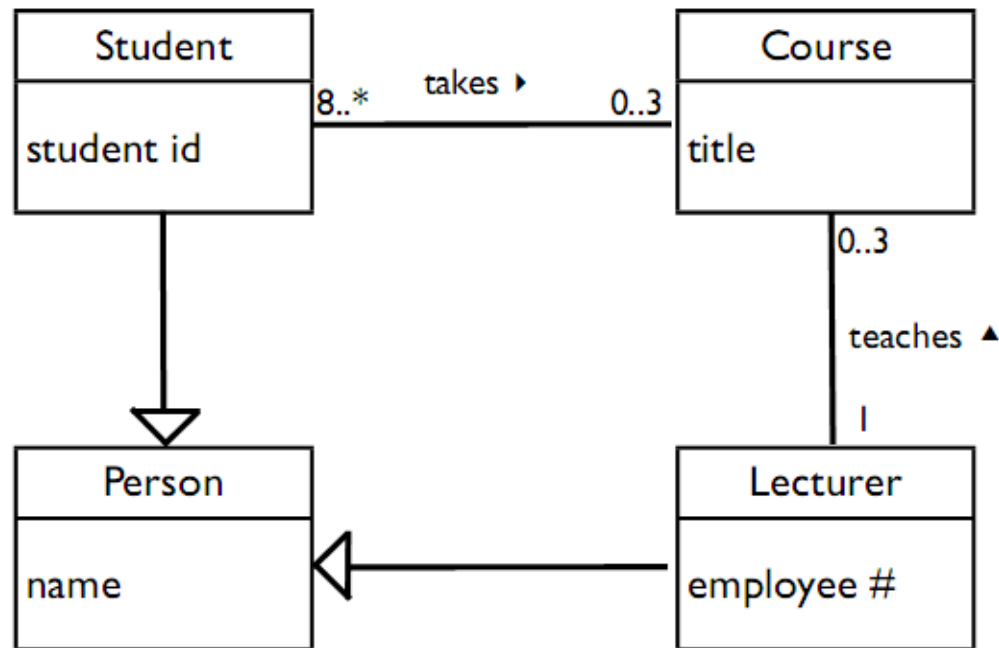
Example: Multiple Inheritance

A class can inherit from several other classes ▸

Use multiple inheritance only when needed, and always with caution !

What Gets 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

Associations

- we should model the important relationships between conceptual classes
  - not *every relationship, that would create clutter and confusion*
  - not too few, we want a useful model

- for each association, provide:
  - a short yet meaningful text label
  - the *multiplicity*

Associations

- Shows relationship between classes
- A class diagram may show:

Relationship	
Generalization (inheritance)	↑ "is a" "is a kind of"
Association (dependency)	does ▶ "Who does What" "uses"
Aggregation	◇ "has" "composed of"
Composition: Strong aggregation	◆

Example: Library System

- Consider the world of libraries. A library has books, videos, and CDs that it loans to its users. All library material has a id# and a title. In addition, books have one or more authors, videos have one producer and one or more actors, while CDs have one or more entertainers. The library maintains one or more copies of each library item (book, video or CD). Copies of all library material can be loaned to users. Reference-only material is loaned for 2hrs and can't be removed from the library. Other material can be loaned for 2 weeks. For every loan, the library records the user, the loan date and time, and the return date and time. For users, the library maintains their name, address and phone number.

- Define the two main actors.
- Identify use cases by providing the actors, use case names. Draw the use case diagram.
- Create the conceptual class diagram.

Example: Digital Music players

Draw a UML Class Diagram representing the following elements from the problem domain for digital music players: An artist is either a band or a musician, where a band consists of two or more musicians. Each song has an artist who wrote it, and an artist who performed it, and a title. Therefore, each song is performed by exactly one artist, and written by exactly one artist. An album is composed of a number of tracks, each of which contains exactly one song. A song can be used in any number of tracks, because it could appear on more than one album (or even more than once on the same album!). A track has bitrate and duration. Because the order of the tracks on an album is important, the system will need to know, for any given track, what the next track is, and what the previous track is.

Draw a class diagram for this information, and be sure to label all the associations (relationships) with appropriate multiplicities.

References & Further Reading

- *Applying UML & Patterns (Larman 2007), Chapters 6, 9.*

- *Object-Oriented Systems Analysis and Design (Bennett et al, Third Edition, 2006), Chapter 6, 7.*