CMPS310

Software engineering

Lecture- 4

Introduction to UML and Requirements Analysis with Use Cases

Topics covered

- System Modeling
- Use case Models
- Use case Diagram
- Use Case Relationships
- Use Case specification
- How to develop a Use-Case Model

Unified Modeling Language (UML)

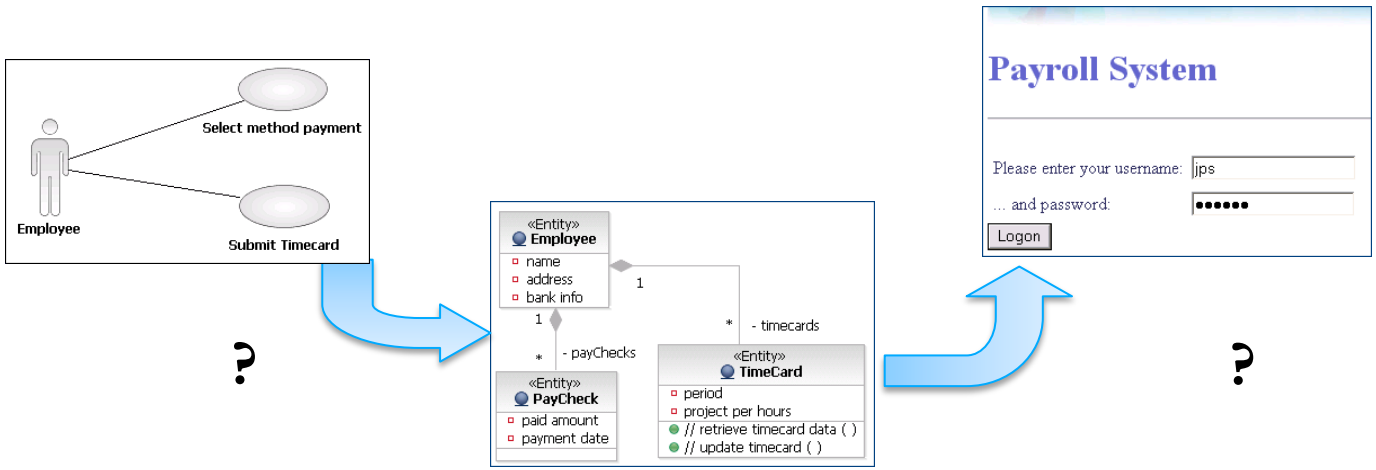
UML Is a Language for Specifying

UML Is a Language for Constructing

UML Is a Language for Documenting

UML Is a Language for Visualizing

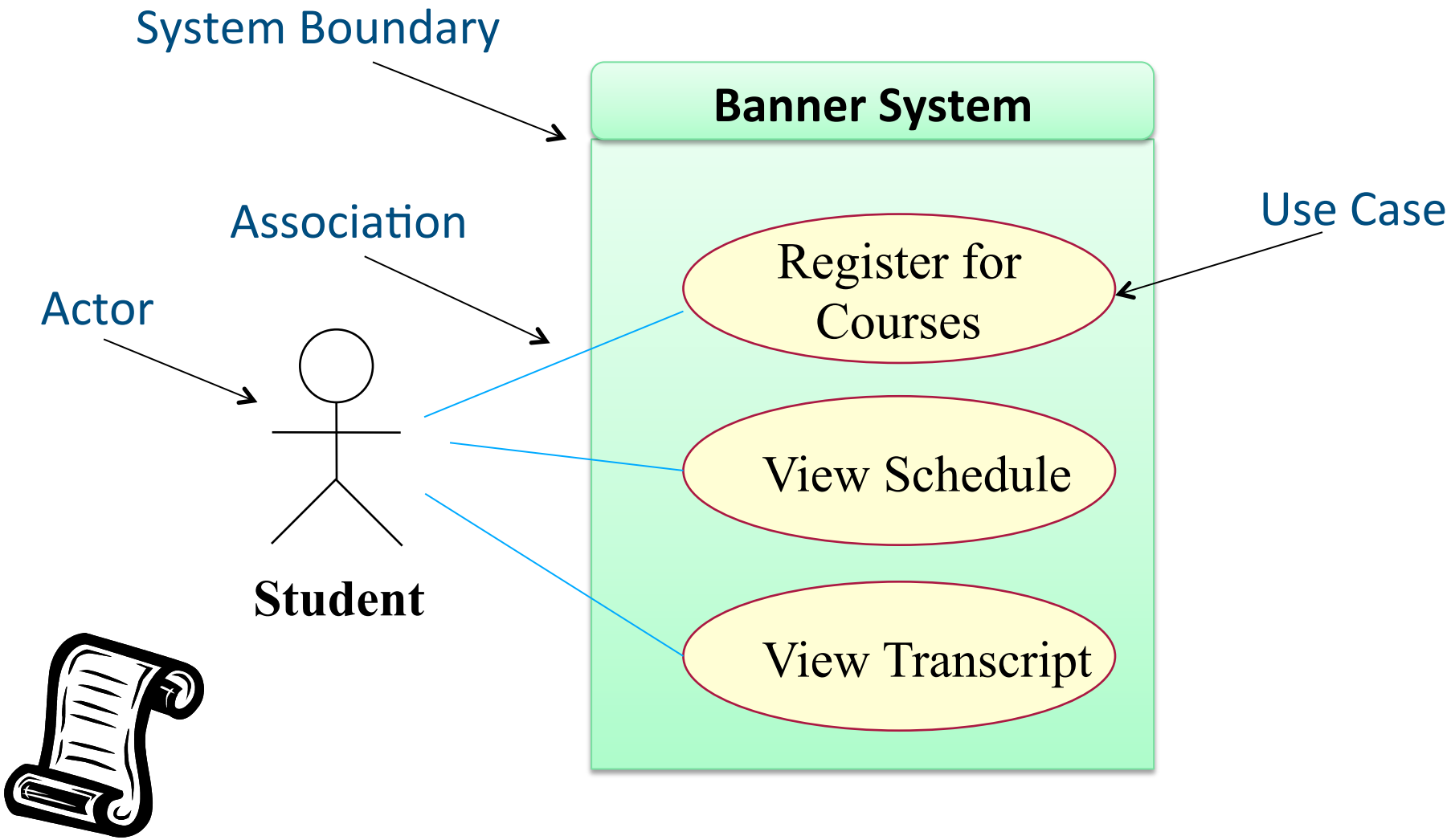
A picture is worth a thousand words!



Use-Case Model and System's Functionality

- A model that describes a system's **functional requirements** in terms of **use cases**
  - shows the **system's intended functionality** (use cases) and **its environment** (actors)
  - shows how the users can **use the system** to **achieve their goals**
  - **A dialog** between actors and the use case (system)

Simplified Use case model for Banner



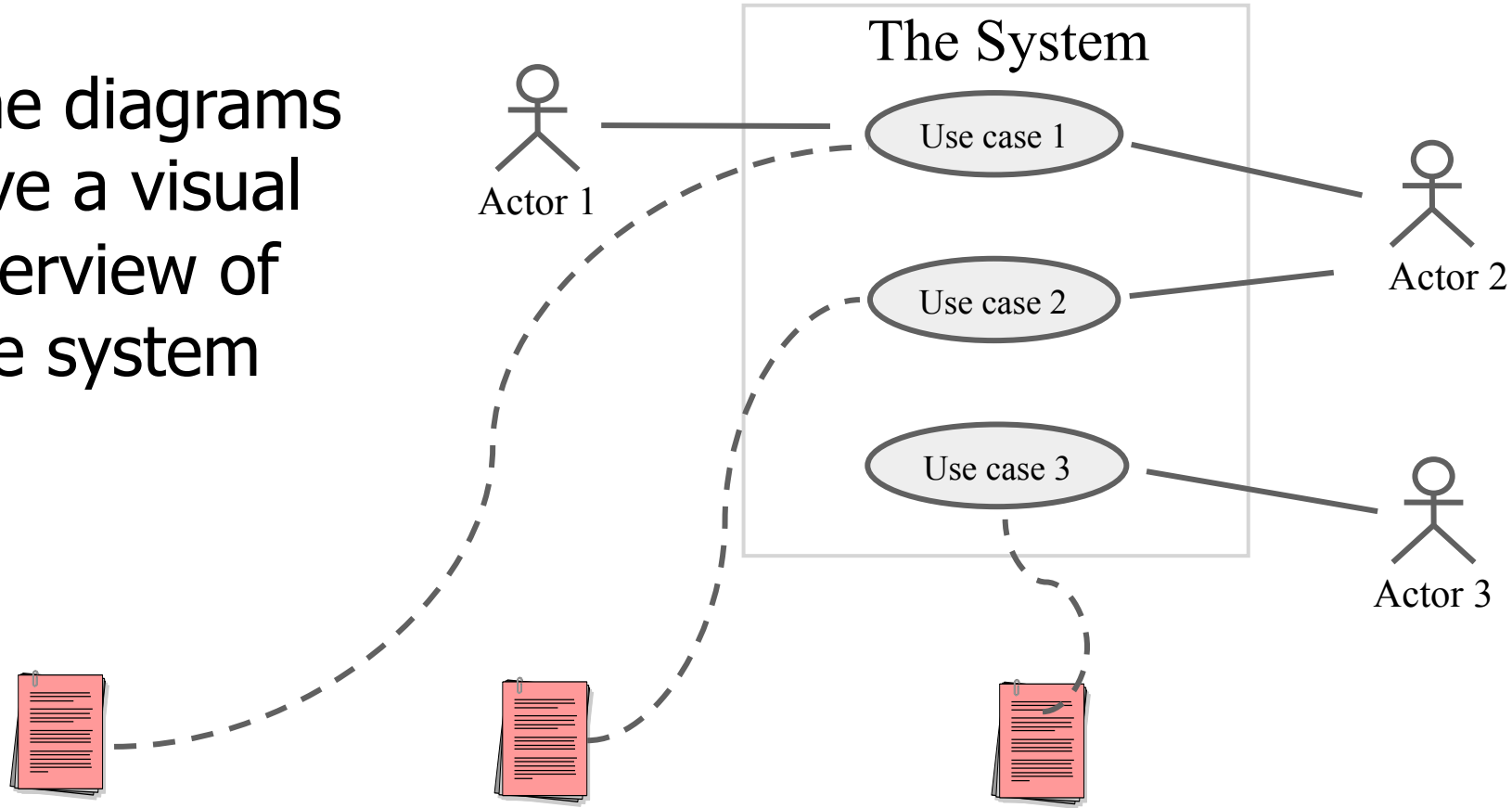
+ A document describing the use case scenarios in details

What is Use Case

- *A use-case is a sequence of actions a system performs that yields an observable result of value to a particular actor.*
- 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 collection of related success and failure scenarios that describe actors using a system to support a goal
- A set of use cases should describe all possible interactions with the system
- A use case tells a story of actors using a system. (e.g.“Rent Videos”)
- System is a ‘black box’ that provides functionality to the user. How it does it is not relevant (at this stage)

A Use-Case Model is Mostly Text

The diagrams give a visual overview of the system



Use-Case 1 Specification
- brief description
- flow of events

Use-Case 2 Specification
- brief description
- flow of events

Use-Case 3 Specification
- brief description
- flow of events

Define Use Cases

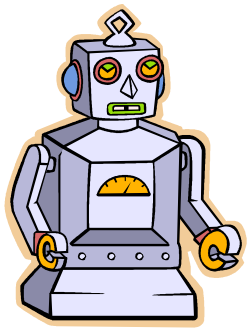
Use Cases: Identify Actors

- Who or what (the role) is/will be providing input to the system?
- Who or what (the role) is/will be receiving output from the system?
- Who or what (the role) is right next to the system boundary passing input into the system?
- Who or what (the role) is right next to the system boundary getting output from the system?

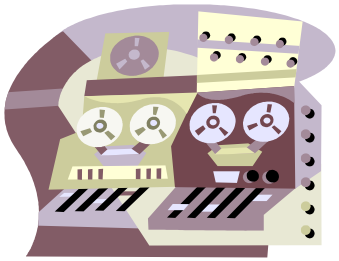
Finding Actors

- External objects that produce/consume data:
  - Must serve as sources and destinations for data
  - Must be external to the system
  - Usually nouns, such as student, staff, camera, external computer, car, etc.

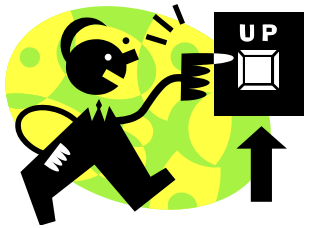
Human



Machine



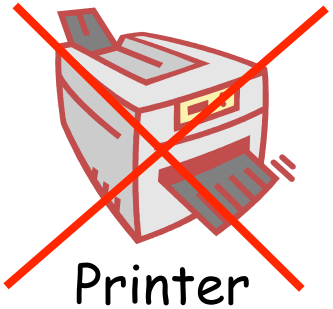
External SW system



Sensor

Camera

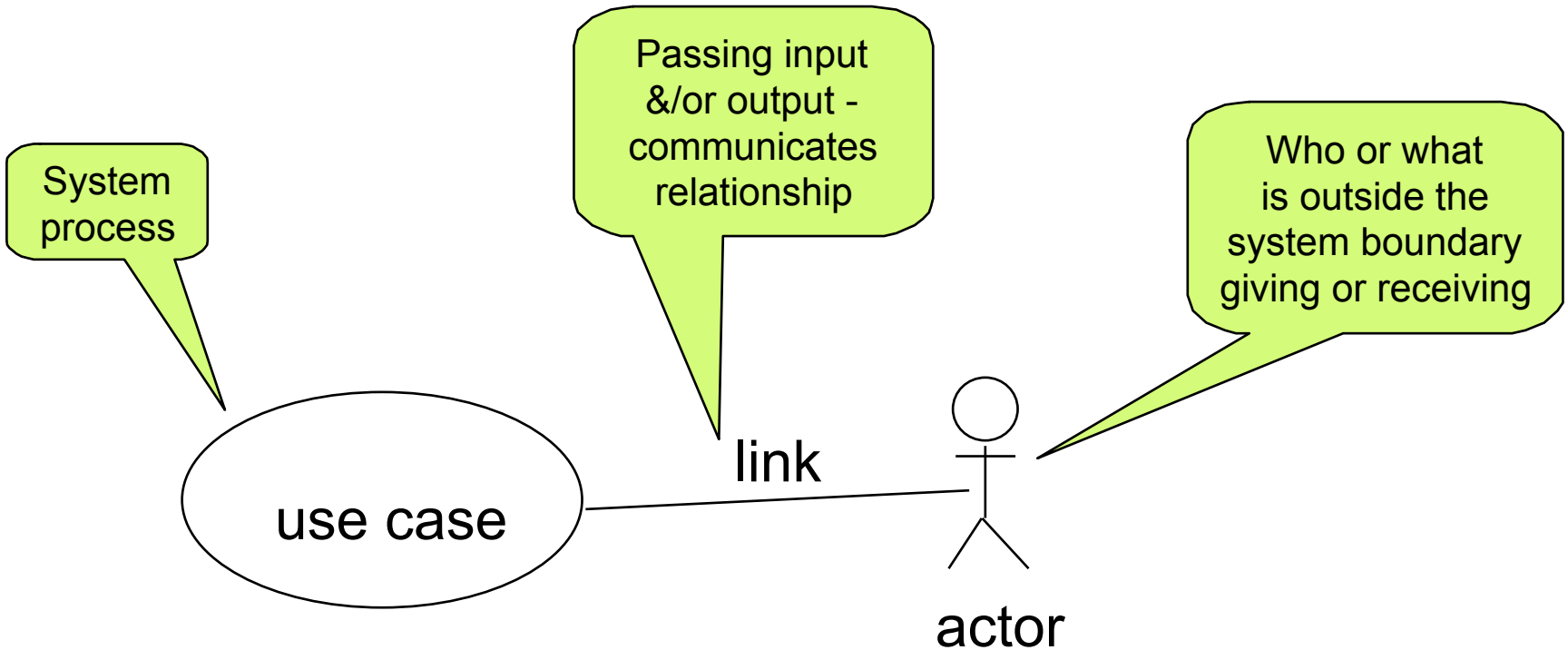
Organizational Units



Printer

Draw a Use Case Diagram

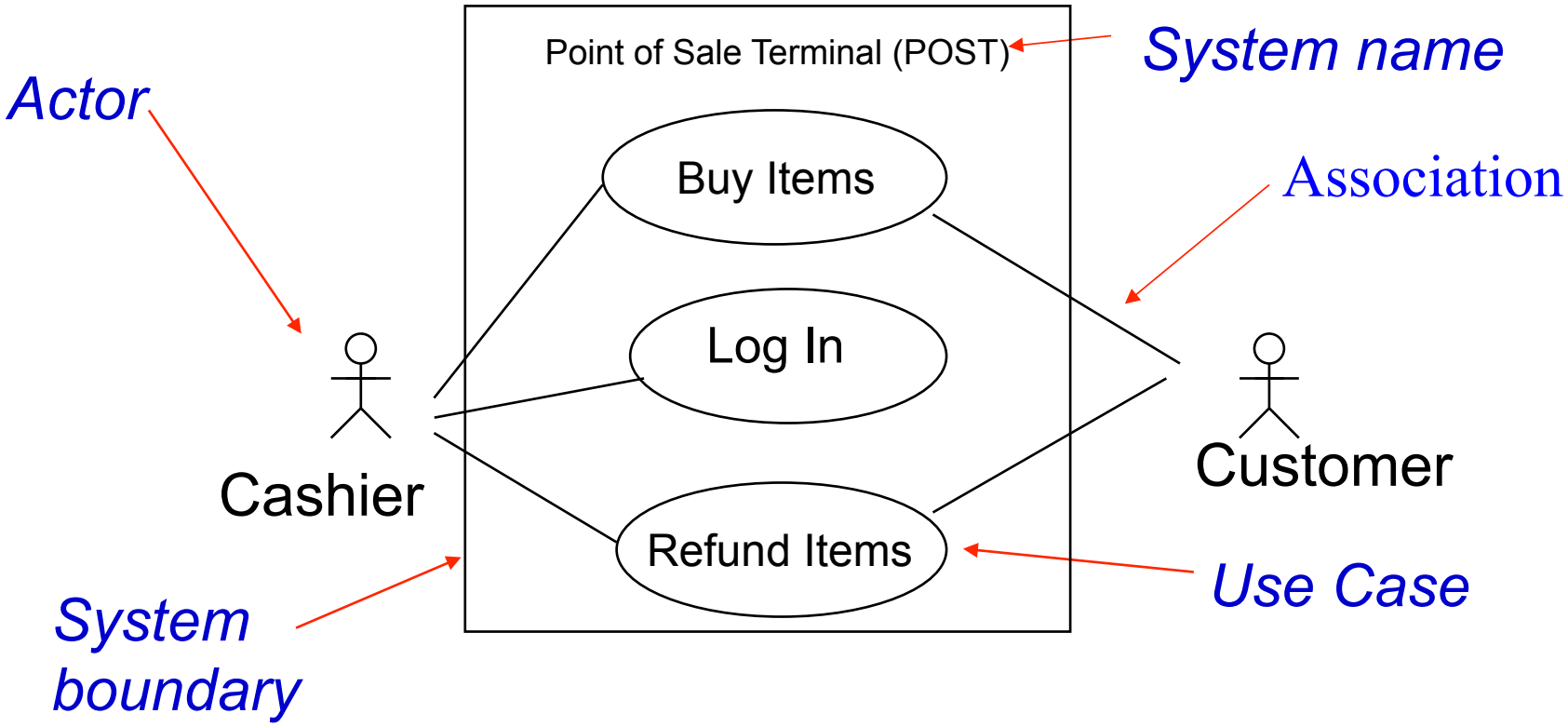
- Shows all the use cases, actors and associations



The UML Icons for a use case, actor and association

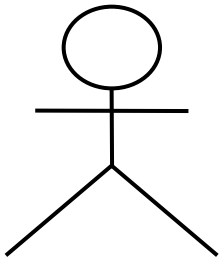
A Sample Use Case Diagram

- A shopping system example where customers buy item and pay to the cashier



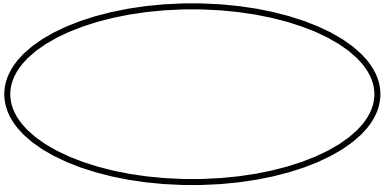
- Five types of basic elements in a use case diagram: Actor, Use case, System name,
- System boundary and Association (relationship)
- Association can have three types: Generalization, <<include>> and <<extend>>

Checkpoints: Actors



- Have all the actors been identified?
- Is each actor involved with at least one use case?
- Is each actor really a role? Should any be merged or split?
- Do two actors play the same role in relation to a use case?
- Do the actors have intuitive and descriptive names? Can both users and customers understand the names?

Checkpoints: Use-Cases

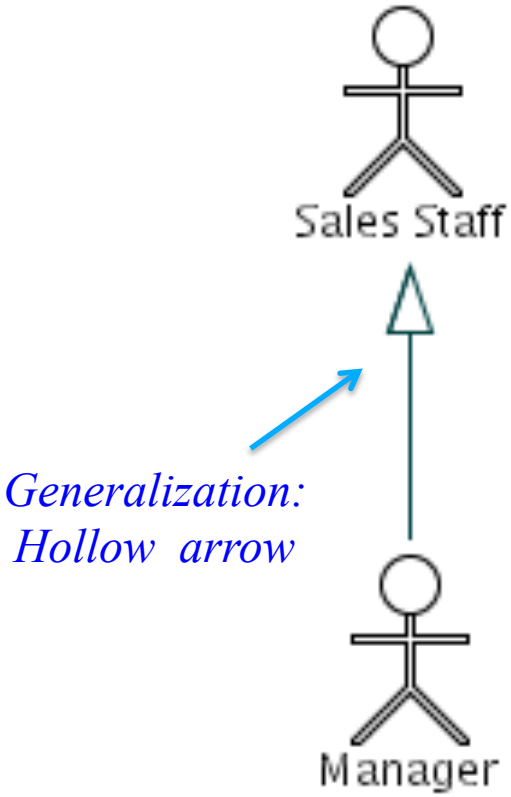


- Is each use case involved with at least one actor?
- Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage?
- Do any use cases have very similar flows of events?
- Do customers and users alike understand the names and descriptions of the use cases?

Use Case Relationships in UML

- **Generalization**
  - One actor is a special kind of another actor (specialized actor)
  - One use case is a special case of another use case (specialized use case)
- **<<include>>**
  - One use case invokes the other use case
  - Included use case represents common behavior
  - Represents a common behavior among several use cases
    - Decompose complicated use case
    - Centralize common behavior
- **<<extend>>**
  - One use case is a variation of the other
  - Extending use case adds behavior
  - Use cases representing exceptional flows can extend more than one use case.

Generalization of Actors

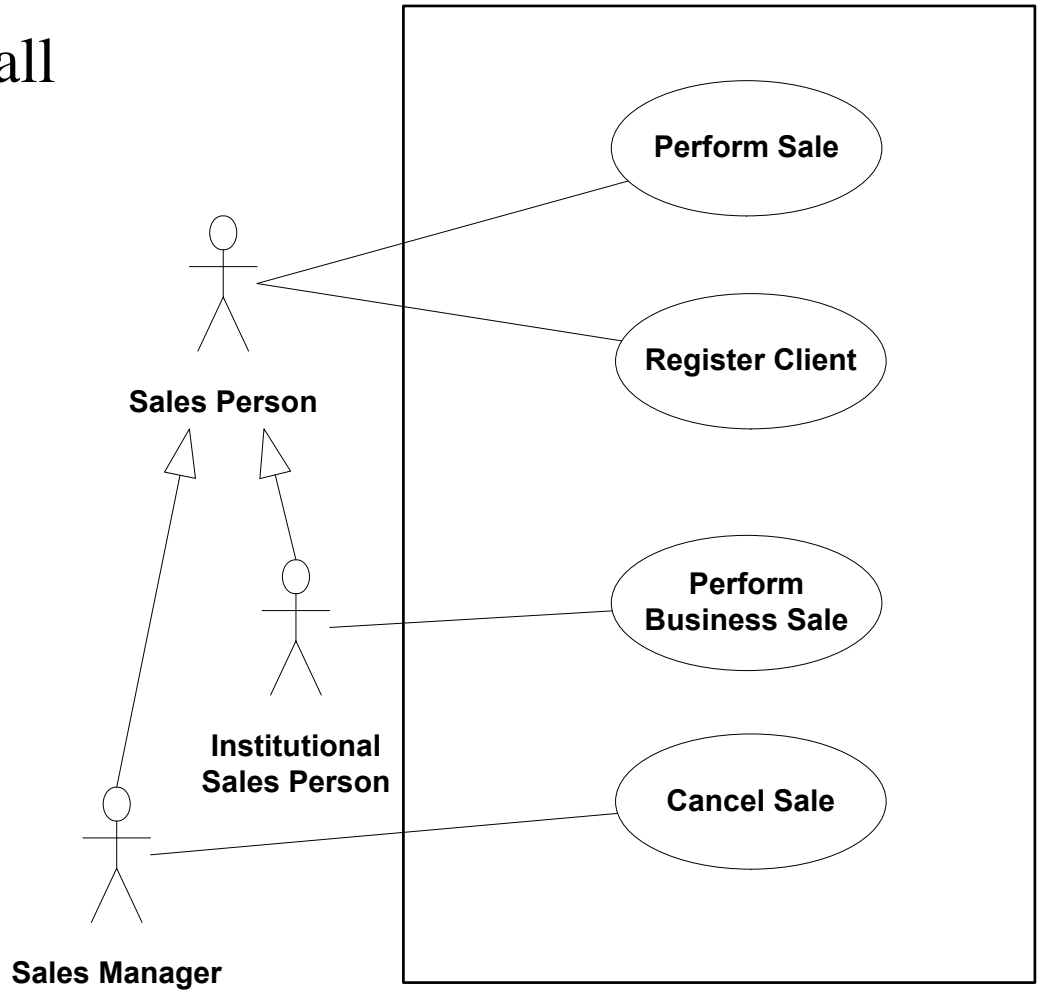


- One actor can be a specialization of another.
- Arrow points to the more general (base) actor
- Manager actor is the specialized actor of Sales Staff
- Notice the arrow type for generalization.

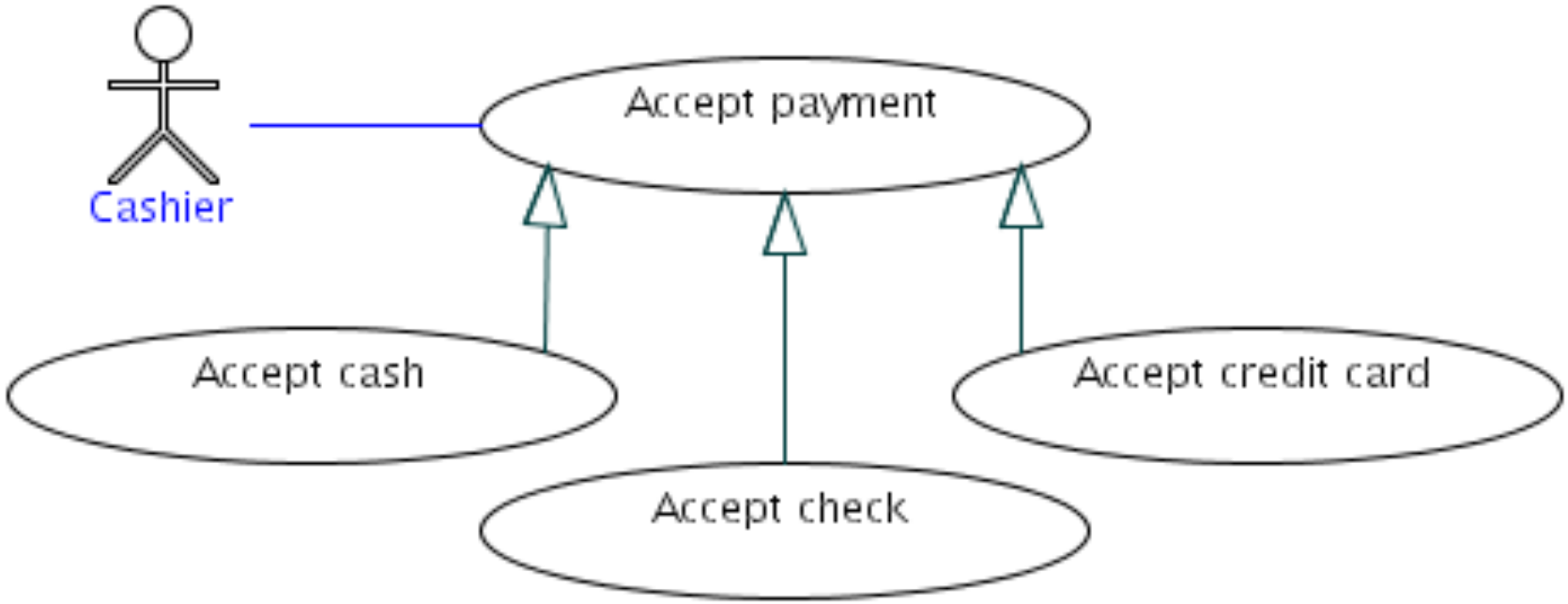
Example: Generalization of Actor

•The child actor inherits all use-cases associations

Should be used if (**and only if**), the specific actor has more responsibility than the generalized one (i.e., associated with more use-cases)



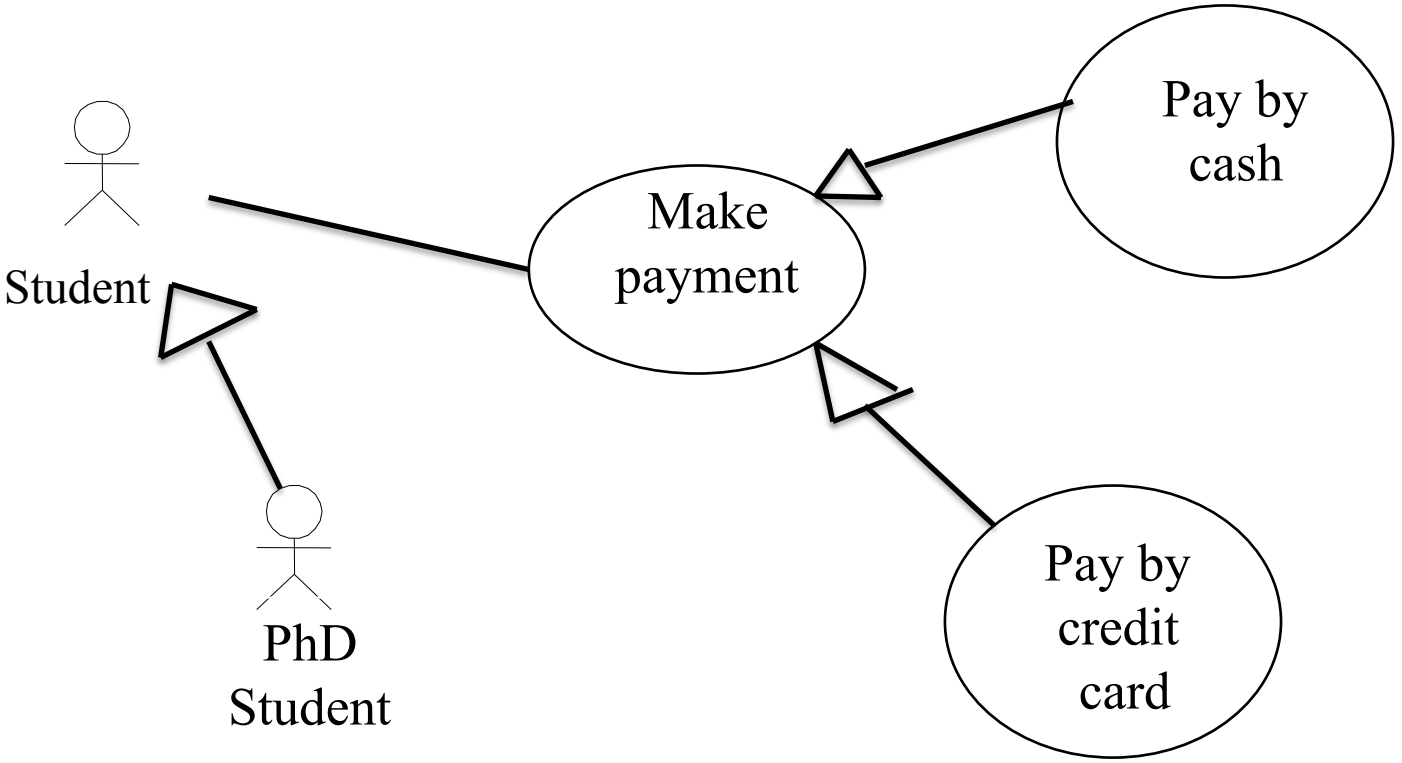
Use case Generalization



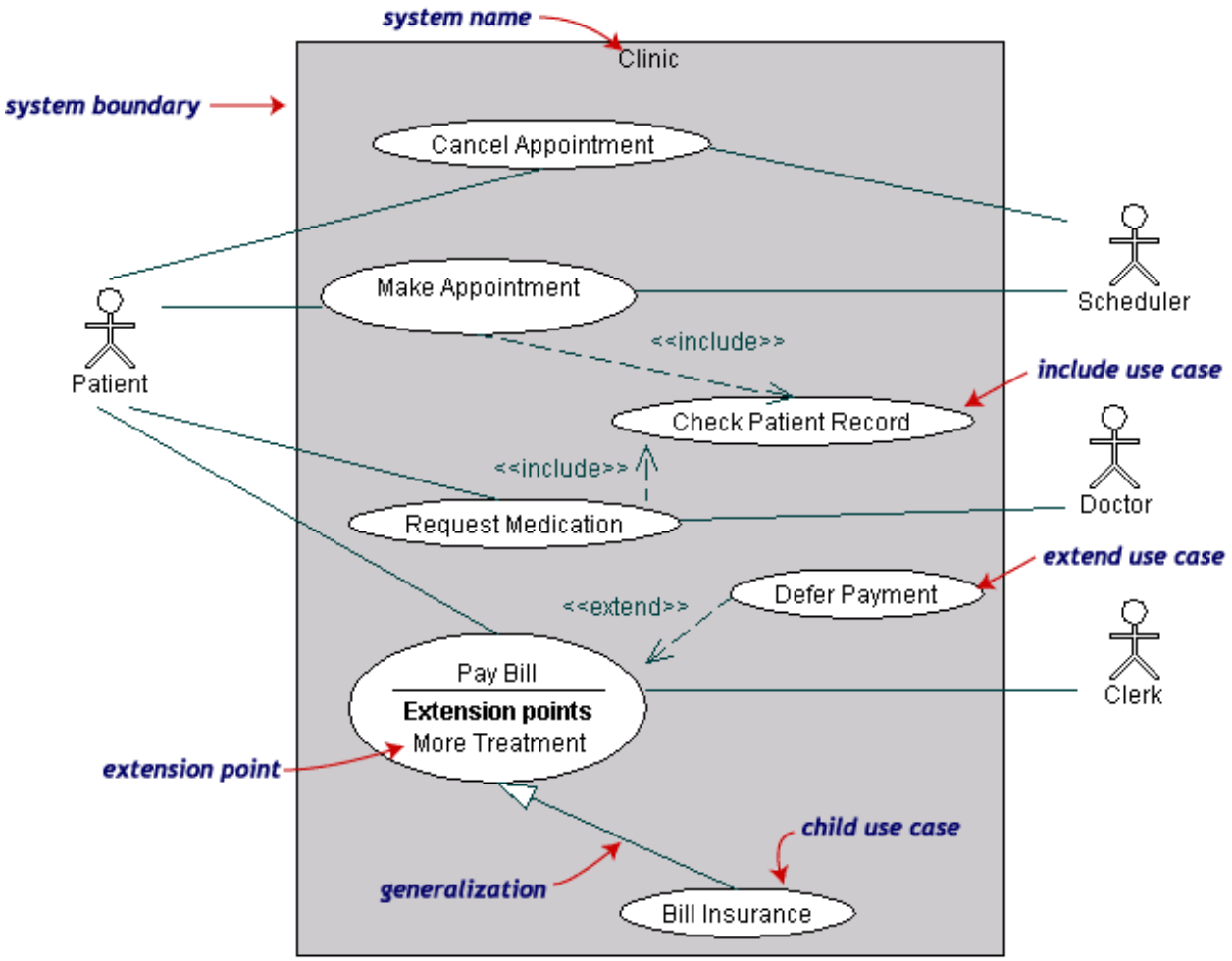
- One use case is simply a special kind of another
- Shows inheritance and specialization. The child use case inherits:
  - The interaction (described in the textual description)
  - Use case links (associations, include, extend, generalization)

Generalization

- Generalization means specialisation of a use case or an actor
  - *PhD student* actor is a specialised one of the actor *Student*
  - *Pay by credit card* is a specialised use case of the *Make payment* use case.
- It has inheritance relationship
  - The specialised ones inherit everything from the parent, not other way around



- Generalization = one is a special kind of the other
- Includes = **one invokes the other (must call the included use case))**
- Extend = **one is a variation of the other (conditional call)**



Notice the direction of arrows and their type of <<extend>> and <<include>>

<<extend>> and <<include>> (1)

- **Extend** is used when a use case adds steps to another use case depending on one or more condition(s).
- For example,
  - imagine "Withdraw Cash" is a base use case of an ATM.
  - "Assess Fee" would extend “Withdraw Cash” and describe the *conditiona*l "extension point" that is instantiated when the ATM user doesn't have bank at the bank that owns this ATM.
  - Notice that the base use case "Withdraw Cash” stands on its own, without the extension.

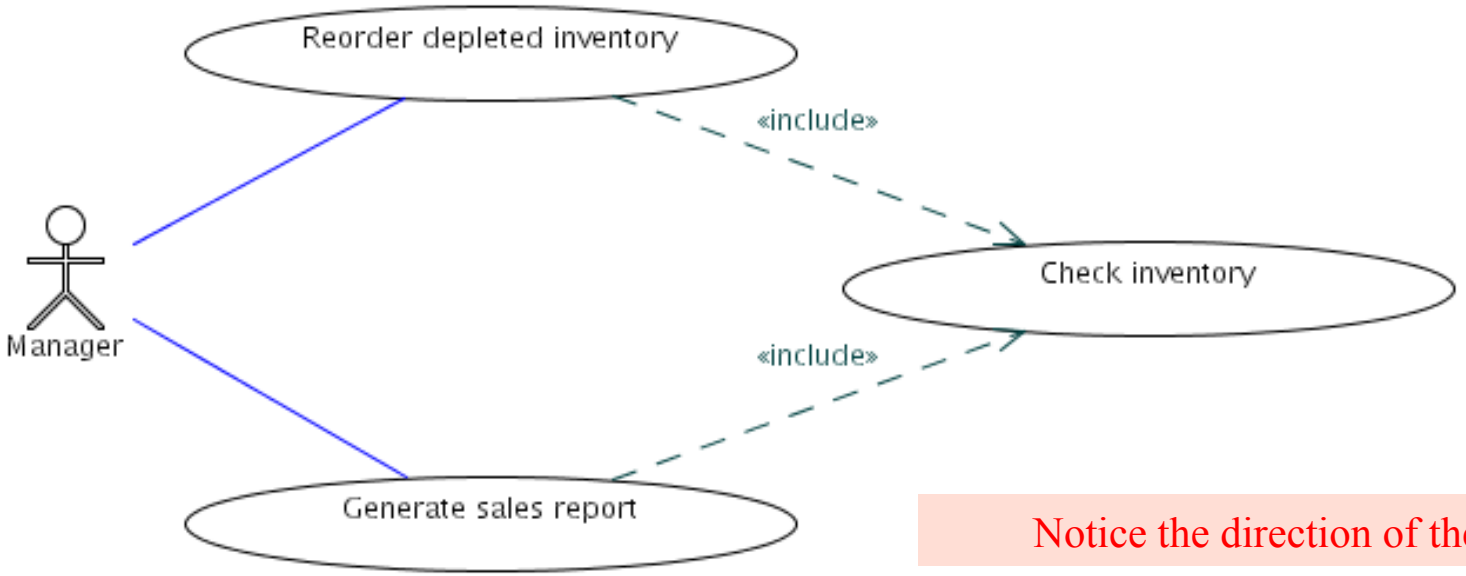
- **<<Include>>** is used to extract use case fragments that are *duplicat*ed in multiple use cases.
- The included use case cannot stand alone and the base use case is not complete without the included use case.
- Base use case **must** call the included use case
- This should be used only in cases where the duplication or commonality occurs.
- To avoid the duplication of a use case, we use <<include>>.
- It exists by design (rather than by coincidence).
- For example, the flow of events that occurs at the beginning of every ATM use case (when the user puts in their ATM card, enters their PIN, and is shown the main menu) would be a good candidate for an include.

<<extend>> and <<include>> (2)

- You can use an **<<extend>>** relationship to specify that one use case (extension) extends the behavior of another use case (base).
- The <<extend>> relationship specifies that the incorporation of the extension use case is dependent on what happens when the base use case executes.
- The extension use case owns the extend relationship. You can specify several extend relationships for a single base use case.
- You can add <<extend>> relationships to a model to show the following situations:
  - A part of a use case that is optional system behavior
  - A subflow is executed only under certain conditions
  - A set of behavior segments that may be inserted in a base use case
- <<extend>> relationships usually do not have names.

- An **<<include>>** relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case).
- The << include>> relationship supports the reuse of functionality in a use-case model.
- You can add include relationships to your model to show the following situations:
  - The behavior of the inclusion use case is common to two or more use cases.
  - The result of the behavior that the inclusion use case specifies, not the behavior itself, is important to the base use case.
- <<Include>> relationships usually do not have names.

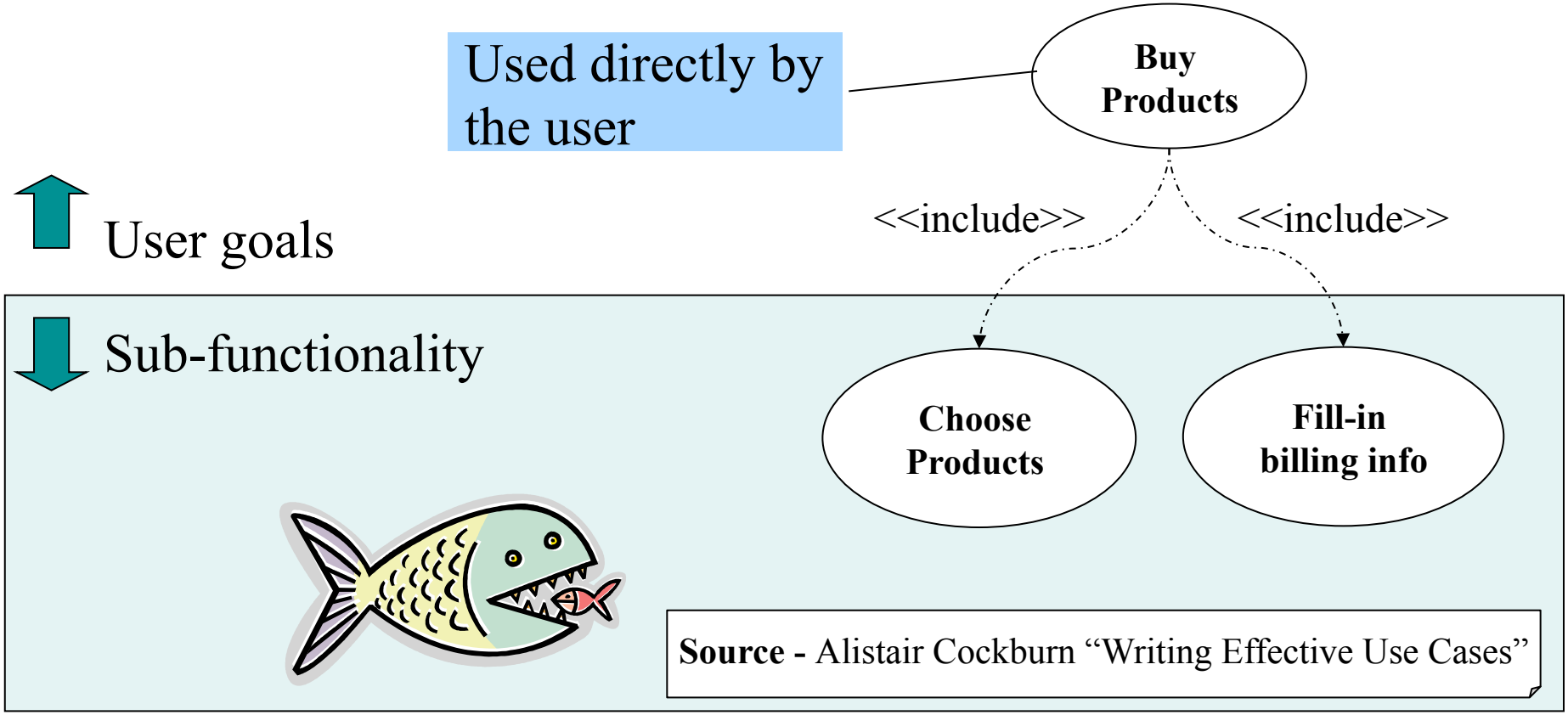
The <<include>> Relationship



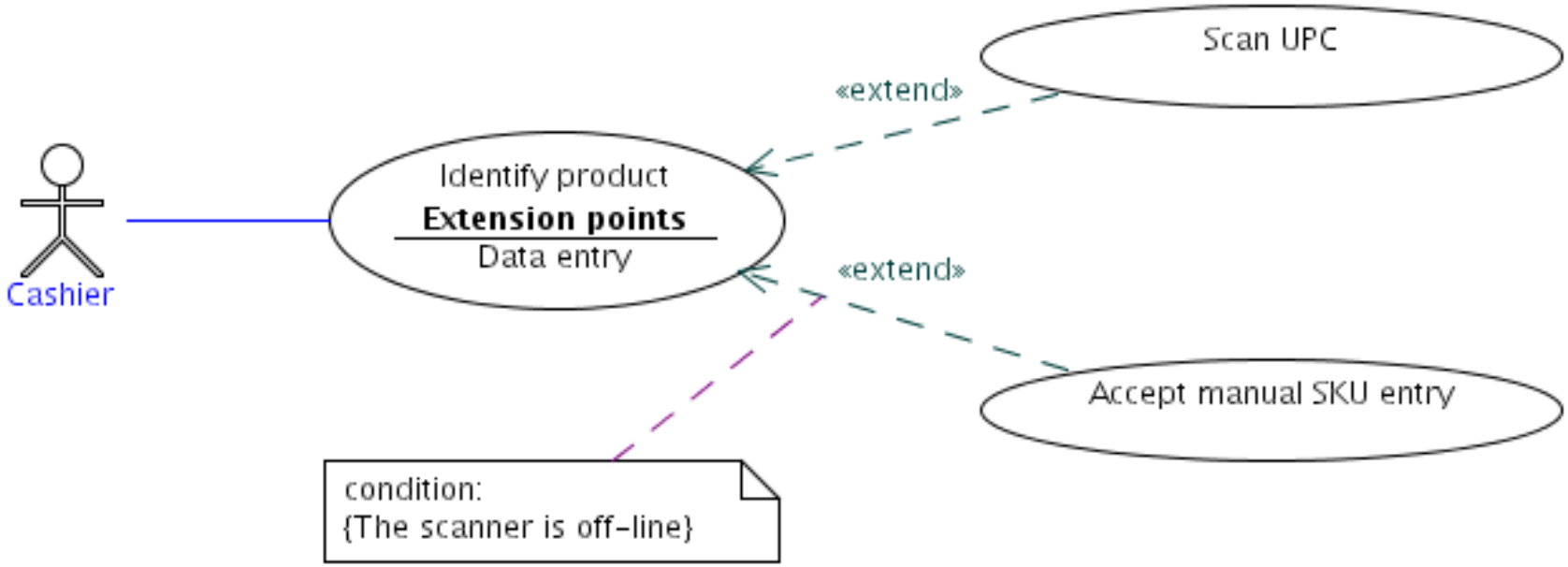
Notice the direction of the arrow

- <<include>> relationship represents a **common behavior among several use cases**
  - To decompose complicated use case
  - To centralize common behavior
  - To simplify large use case by splitting it into several use cases,
  - To extract **common parts** of the behaviors of two or more use cases.

The <<include>> Relationship



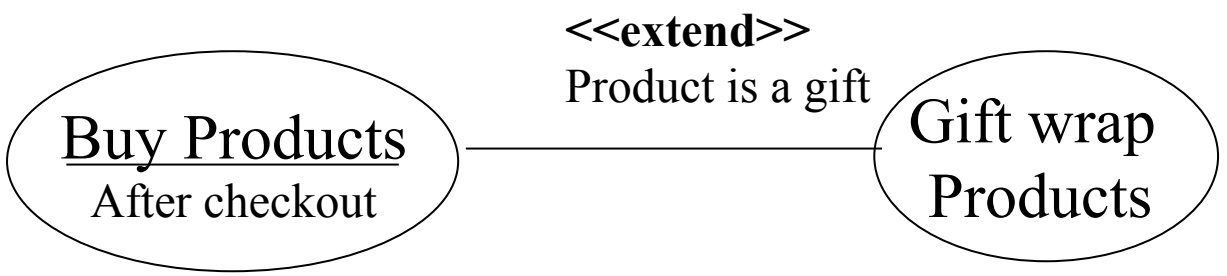
The <<extend>> Relationship



- <<extend>> relationship represent an **exceptional case**
  - The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extend>> relationship is to the extended use case

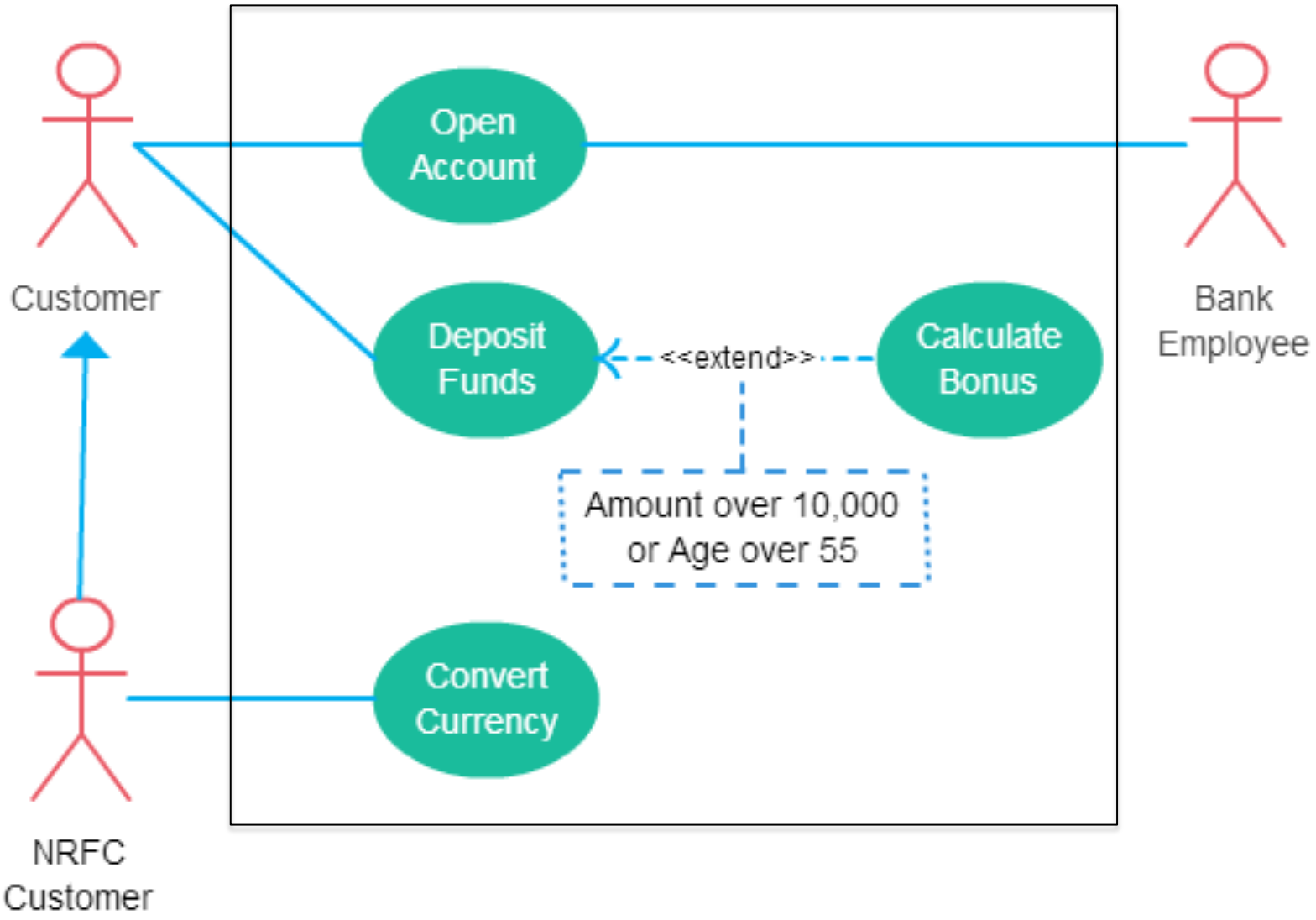
Example: <<extend> Relationship

- The base use case can incorporate another use case at certain points, called extension points.
- Note the direction of the arrow
  - The base use-case does not know which use-case extends it

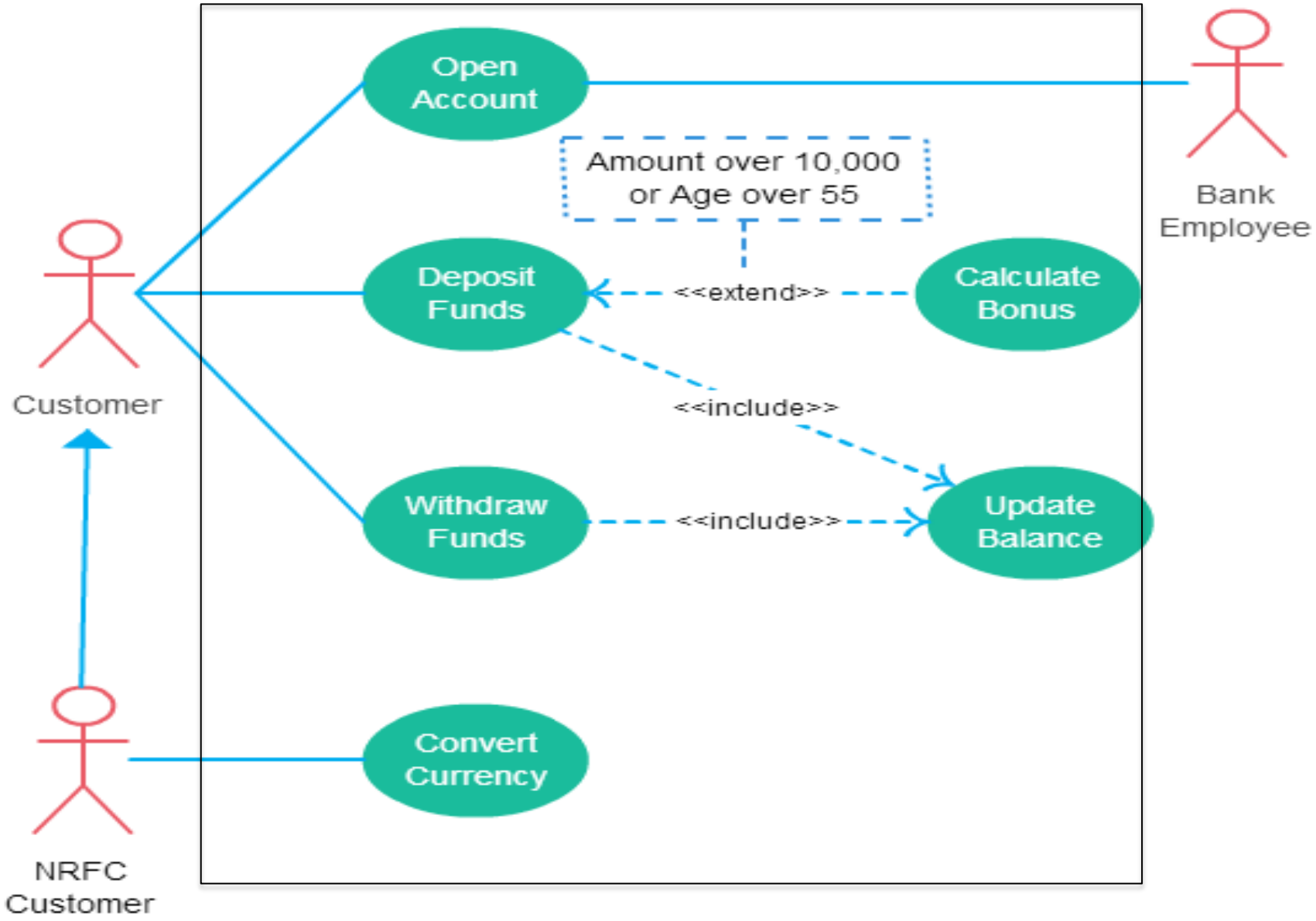


Example

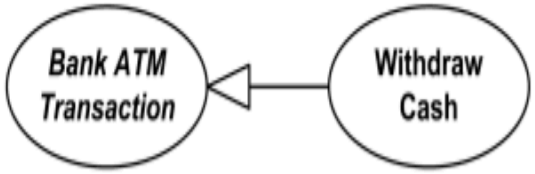
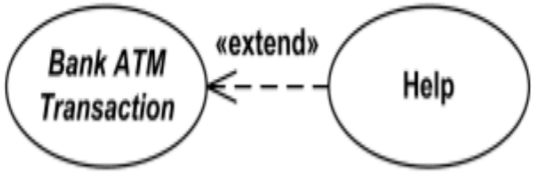
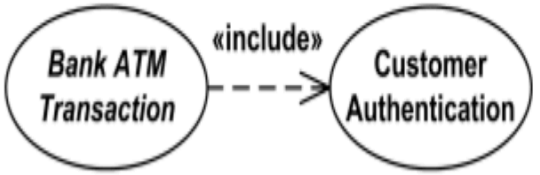
<<extend>> Example with Condition: Bank



Use Case Diagram: Library System



Example: Generalization, <<extend>>, <<include>>

Generalization	Extend	Include
		
Base use case could be **abstract use case** (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (**abstract use case**).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

An Exercise on Use Case Diagram

Exercise: Use Case Diagram

The QU wants to computerize its registration system

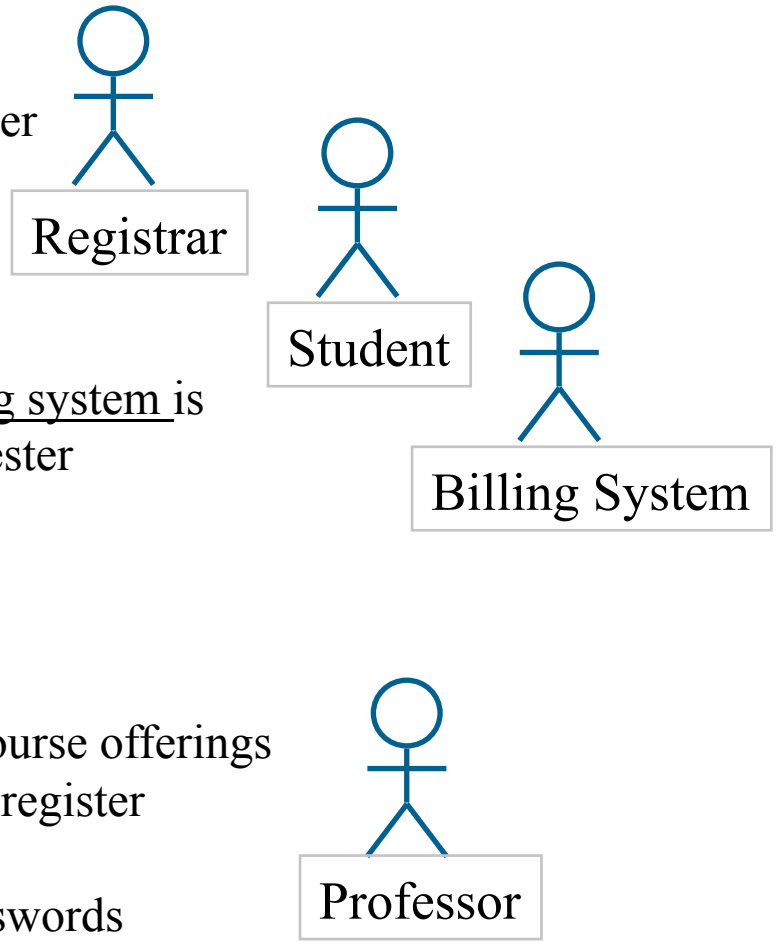
- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course class lists after students register
- Users of the registration system are assigned passwords which are used at logon validation

Actors in Use Case Diagram

- An actor is someone or some thing that must interact with the system under development

The QU wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course class lists after students register
- Users of the registration system are assigned passwords which are used at logon validation

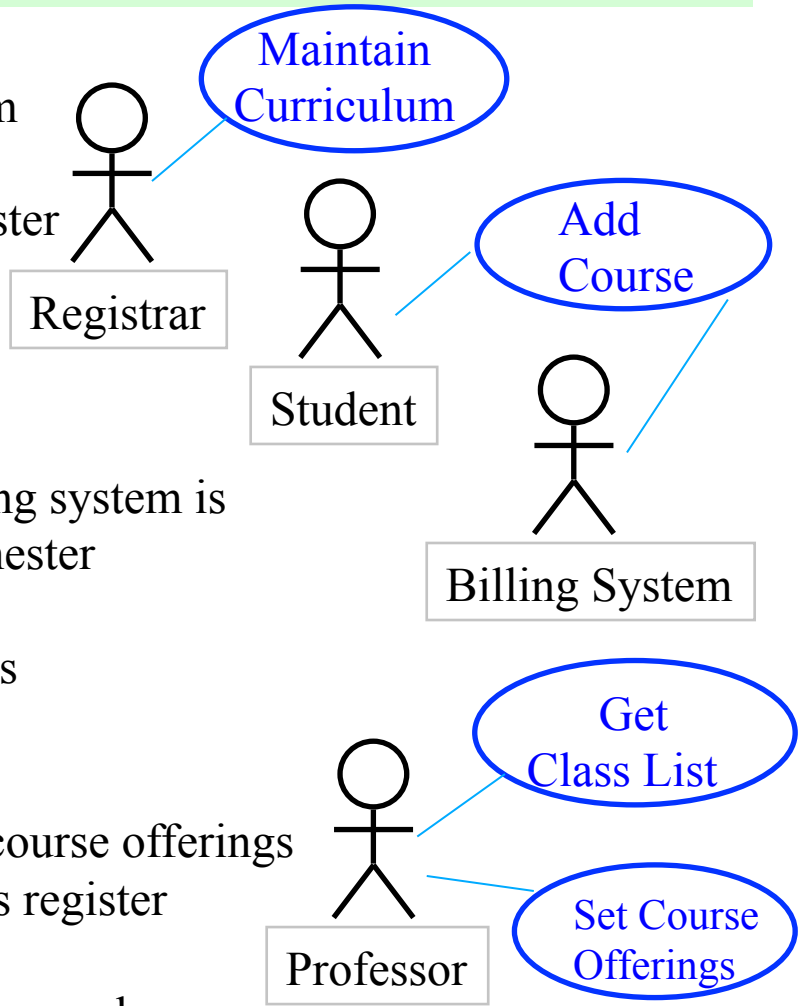


Use Cases

A use case is a sequence of interactions between an actor and the system

The QU wants to computerize its registration system

- The Registrar sets up the curriculum for a semester
- Students select 3 core courses and 2 electives
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course class lists after students register
- Users of the registration system are assigned passwords which are used at logon validation



Use Case Diagram: Basic Rules

- A use case must be associated either with an actor or another use case
- An actor cannot be directly connected with another actor unless it is an association relationship (Generalization)
- A use case name must be **active verb**
- An actor name must be generic **noun**
- An actor cannot exist without being associated with at least either one use case or another actor in Generalization relationship.

Summary of Rules for Actors

- **Give meaningful business relevant names for actors** – For example if your use case interacts with an outside organization its much better to name it with the function rather than the organization name. (e. g.: Airline Company is better than PanAir)
- **Primary actors should be to the left side of the diagram** – This enables you to quickly highlight the important roles in the system.
- **Actors model roles (not positions)** – In a hotel both the front office executive and shift manager can make reservations. So something like "Reservation Agent" should be used for actor name to highlight the role.
- **External systems are actors** – If your use case is send email and if interacts with the email management software then the software is an actor to that particular use case.
- **Actors don't interact with other actors** – In case actors interact within a system you need to create a new use case diagram with the system in the previous diagram represented as an actor.
- **Place inheriting actors below the parent actor** – This is to make it more readable and to quickly highlight the use cases specific for that actor.

Summary of Rules for Use Cases

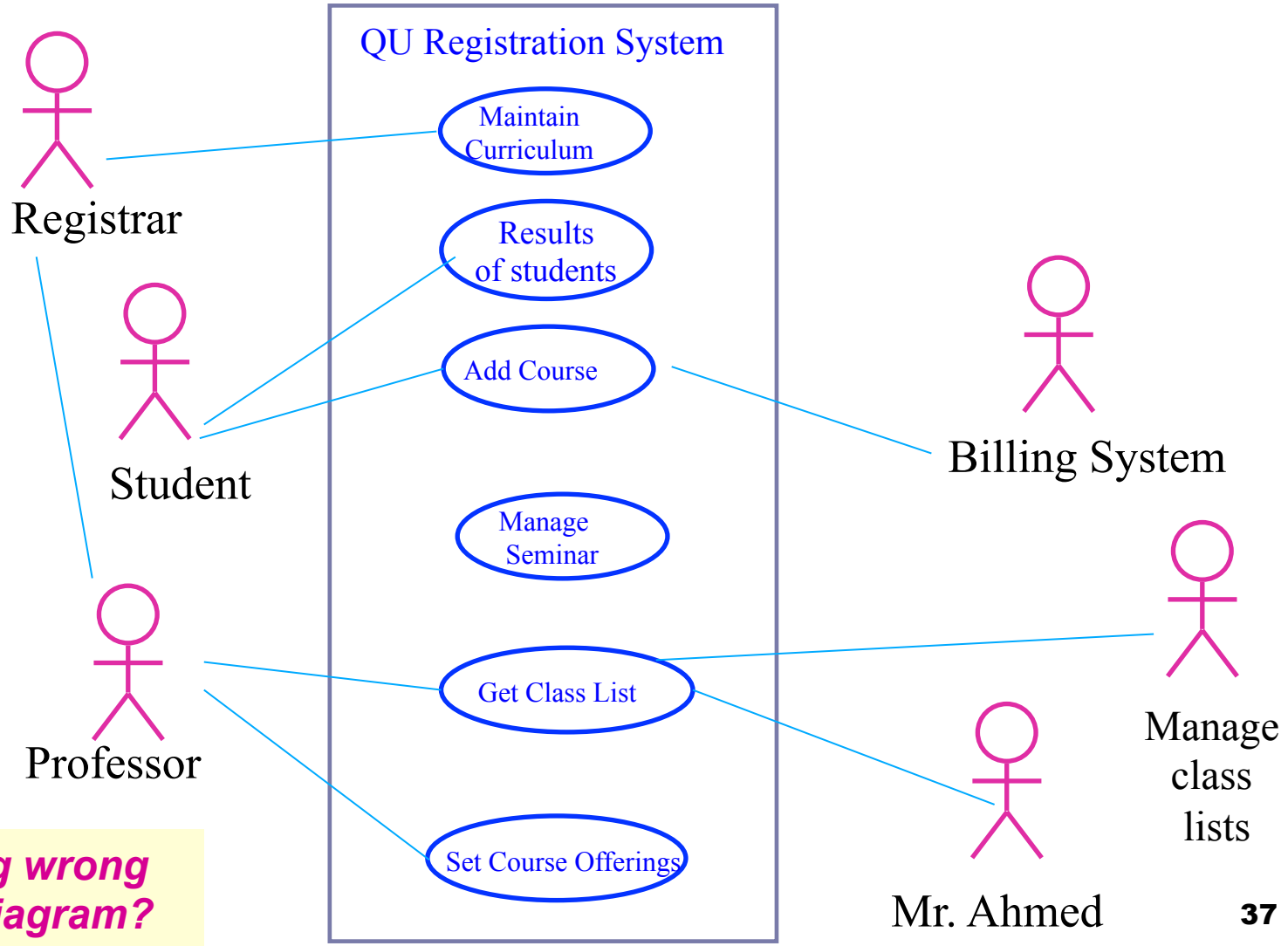
- **Names begin with a verb** – An use case models an action so the name should begin with a verb.
- **Make the name descriptive** – This is to give more information for others who are looking at the diagram. For example “Print Invoice” is better than “Print”.
- **Highlight the logical order** – For example if you’re analyzing a bank customer typical use cases include open account, deposit and withdraw. Showing them in the logical order makes more sense.
- **Place included use cases to the right of the invoking use case** – This is done to improve readability and add clarity.
- **Place inheriting use case below parent use case** – Again this is done to improve the readability of the diagram.

Summary of Rules for Relationships/ Associations

- There can be 5 relationship types in a use case diagram.
  - Association between actor and use case
  - Generalization of an actor
  - Extend between two use cases
  - Include between two use cases
  - Generalization of a use cases
- Arrow points to the base use case when using <<extend>>
- <<extend>> can have optional extension conditions
- Arrow points to the included use case when using <<include>>
- Both <<extend>> and <<include>> are shown as dashed arrows.
- Actor and use case relationship doesn't show arrows.

Use Case Diagram: University System

- Use case diagrams depict the relationships between actors and use cases. The diagram has also some mistakes, it violated the use case rules



Anything wrong in this diagram?

Use Case Specification

Structure of a Use Case Specification

Name (use case name exactly as in the diagram)

Brief Description: (max 3-4 sentences)

Actors: (List all actors as in the diagram)

Trigger (In one sentence, write what triggers this use case)

Preconditions (What conditions must be met to start this use case)

Post conditions (What changes are made after the execution of this use case

Normal Scenario: (Actor's actions, and the use case's response to those)

Alternatives Flows (Conditional system response, if there is any)

Non-Functional (optional) List of Non-Functional Requirements (NFRs)

Use Case Specification Template

Use case Id:	<Use case Title>
Brief Description	
Primary actors	
Trigger(s)	
Preconditions:	
Post-conditions:	
Normal Scenario	
Actor Action	System Response
1.	2.
Alternative flows: ?.a.	

Sometimes it is also called: Actor action – system response table

Triggers

- What starts the use-case?
- Examples:
  - Customer reports a claim
  - Customer inserts card
  - System clock is 10:00
- Usually the first step of a user case in the trigger event that starts the scenario
  - E.g., customer arrives to a checkout with items to purchase

Preconditions vs. Post-Conditions

- **Preconditions** = what *must always* be true before **beginning** a scenario. Preconditions are *not* tested within the use case but they are conditions that are *assumed to be true*. E.g. :
  - **Student identified and authenticated**
  - **User account exists**
  - **User has enough money in her account**
  - **There is enough disk space**
- **A post-condition** = success guarantee = the outcome of the use-case.

=> How to test that the use case was successful, E.g.:

  - **Money was transferred to the user account**
  - **User was logged in**
  - **The file was saved to the hard-disk**
  - **The file was saved; Money was transferred**
  - **Sale was saved. Tax was correctly calculated. Inventory updated. Receipt was generated**.

=> Declarations about the system state changes or outcomes rather than a description of actions to execute

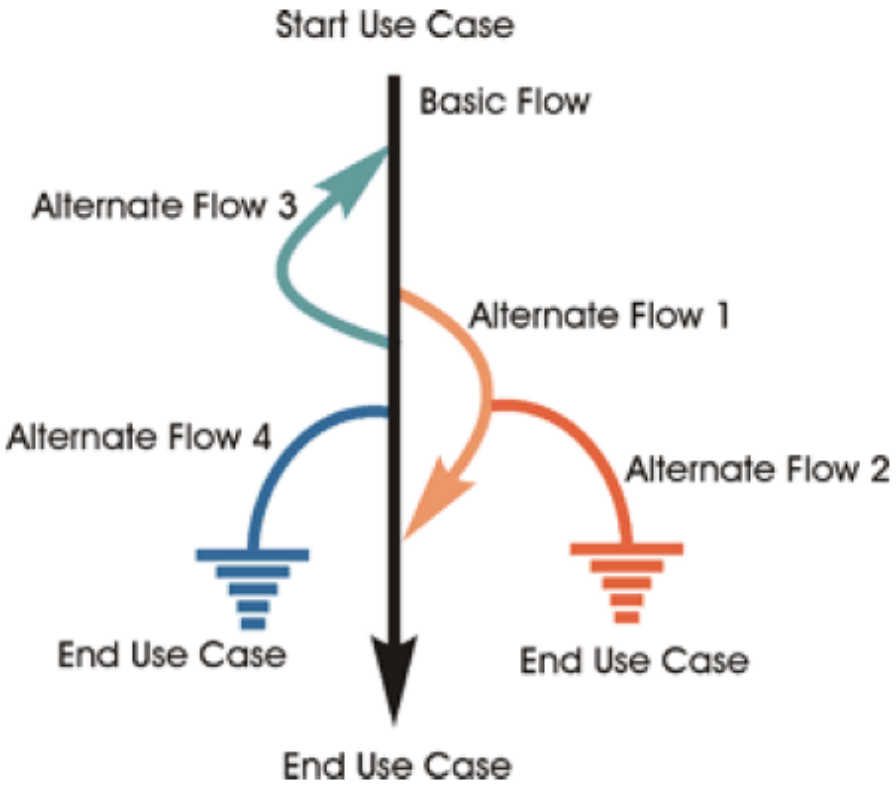
Use Case Scenarios

- A scenario is a sequence of steps describing the **interaction (a dialog) + the exchanged information between the actor and the system**
  - A **story of using the system** to achieve a user goal
- A use case has **one normal scenario** (happy day scenario) and several alternative flows:

  - Normal scenario describes **what "normally" happens** when the use case is performed

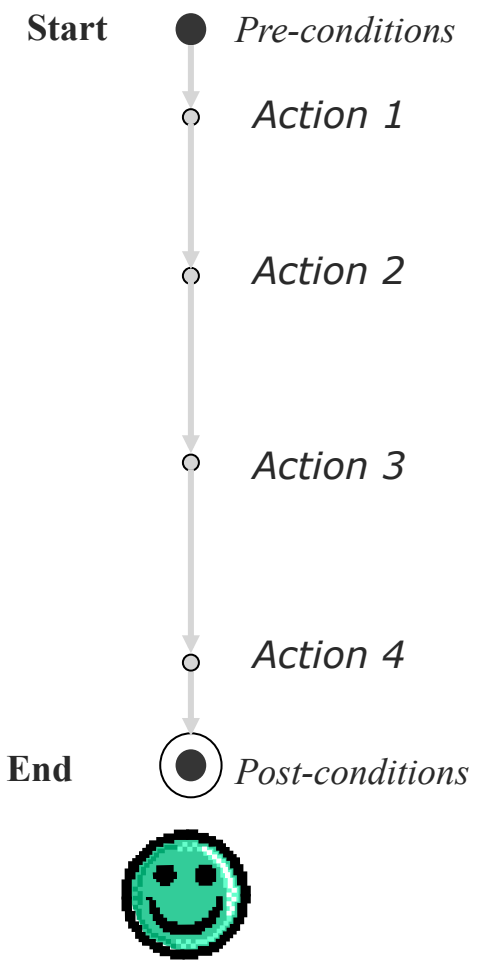
  - Alternative flows describe **optional or exceptional behavior** for handling errors or exceptions during the normal flow of events.

Normal Flow of Events and Alternate Flows of Events for a Use Case



Writing the Normal Scenario (The most Important component)

- The normal scenario is written under the assumption that everything is okay, no errors or problems occur. It describes:
  - Pre-conditions : what must be true before the use case starts
  - *The **interaction** and* what **data are exchanged** between *the actor and the system*
  - The data **validation** performed by the system
  - **State change** by the system (e.g., recording or modifying something)
  - Post-conditions = what will be true upon successful completion

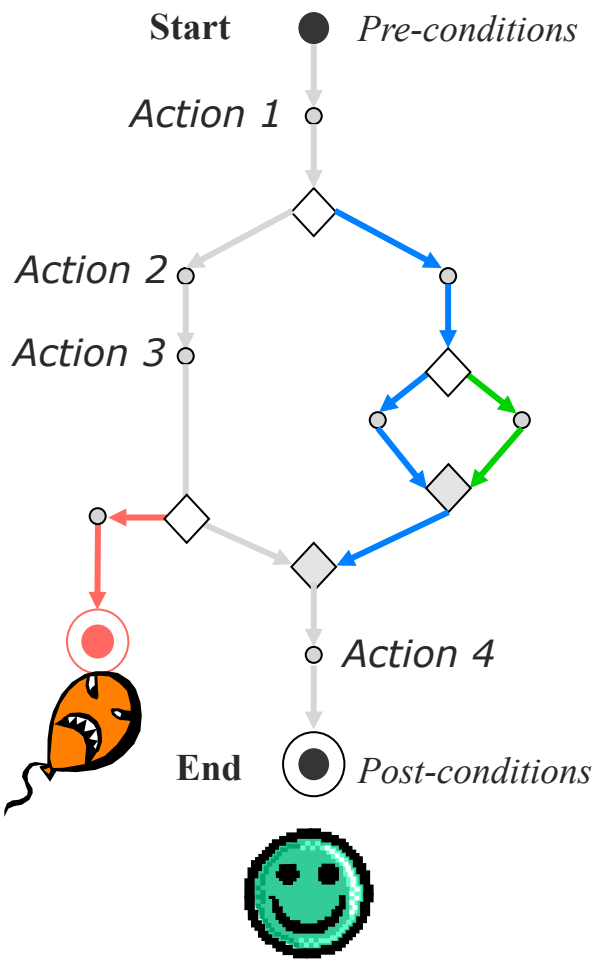


Writing Alternative Flows (conditional Flow)

- List what can go wrong in the normal flow. e.g.:
  - *course registration closed*
  - *invalid studentID*
- Describe **what to do to handle** the identified exceptions?
  - Sometimes the exception is recoverable i.e., the **alternative flow rejoins the normal flow**

  e.g., if the course is full the system can display alternative courses then the normal flow resumes
  - Or the exception could be non-recoverable and ends the use case

  e.g., if the registration is close, display a message and the use case ends.



Use Case Specification: Basic Rules

- There will be no alternative flow for the actor action
- You can define alternative flow ONLY for the system response
- Do not write “The system” beginning of every system response -- start directly with a verb.
- Make actor action and system response compact and concise --do not just copy from the requirement document and paste in the table.
- The flow of system response and actor actions should be in logical order
- Specify the name of the actor in each Actor Action.
- Alternative flow usually starts with “If”
- Use one sequence numbering system for all actor actions and system responses.
- If there is an <<extend>> relationship, that means the base use case calls another use case **if the condition is met**,
  - in the normal scenario, at the system side, start with this statement: <**extend: Use case name> then the condition**.
- For the <<include> relationship, that means the base use case calls another use case with no condition,
  - in the normal scenario, at the system side, start with this statement: <**include: Use case name>**.

Tips for Writing Effective Use Case Specification

- Actors should represent **roles** (not persons)
- Actor names should singular (not plural)
- Actor names should be consistent (e.g., if you use *Faculty* always use the same name and not *Professor*)
- Use case name should be a **verb** followed by a direct object.
- Make sure that each use case describes a significant chunk of system usage
- Do not represent communication between actors. Actors may collaborate through a use case.
- Reuse common Use Cases using <<include>> and <<extend>>
- Create detailed Use Case scenarios

Writing <<Include>> and <<extend>> in the Specification

- If a normal scenario includes another use-case, we will describe this as a step within the normal flow:
- Examples:

System response

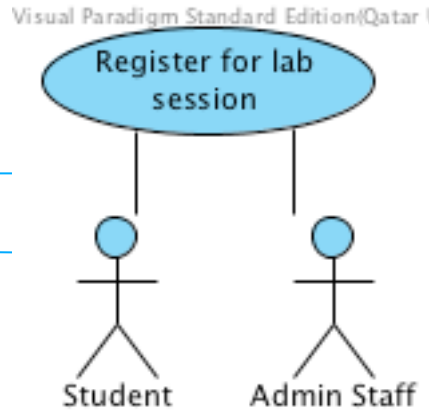
1. <include: Login use case>
2. <extend: Compute bonus use case> condition(s)
3. <extend: Register use case> if customer has no login info

Example 1: Buy Item - A Normal Scenario

Actor Action	System Response
1. A customer arrives at a POST checkout with items to purchase	
2. The Cashier records the identifier from each item, and the quantity	3. Find the item price and adds the item information to the sale
4. The cashier indicates that the item entry is complete	5. Calculate and presents the sale total
6.Cashier tells the Customer the total	
7. The Customer gives a cash payment, possibly greater than the sale total	
8. The Cashier records the cash received amount	9. Show the balance, and change
	10. Generate a receipt
11.The Cashier extracts the balance owing, gives to Customer	12. Log and complete the sale

Use case: Buy item

Example 2: A Normal Scenario with Alternative Flow



Use case: Register for lab session

Actor Action	System Response
1. Student chooses a lab from the available list	
2. Student requests allocation to choice of lab from Admin staff.	
3. Admin Assistant enters Student Name and ID Number into lab List	4. Check vacancy and records student in lab (see 4a for alternative flow)
5. Student leaves with allocation to a lab	

Alternative flow:

4a. If there is no vacancy, ask to choose another lab

Example 3: Use Case Specification

Use case: Book airline ticket

Actor Action	System Response
1. The customer enters flight information	2. Search for the flights
	3. Display the flight information with ticket price without airport charges
4. The customer selects the flights if options are available	5. Record the selection
	6. Request for the passenger details
7. The customer provides passengers details and frequent flyer number of there is any	8. Store the passenger information and frequent flyer number (See 8a)
	9. Check for the frequent flyer validity (See 9a)
	10. Compute the price of the ticket
	11. Provide the seat map of all flights
12. Select the seats of the flights	13. Reserves the seats
	14. Contact "World Airport Charges"
15. World Airport Charges responds with the charges of the airport	16. Compute the total price of the ticket
	17. Ask the customer for payment

Alternative flows:

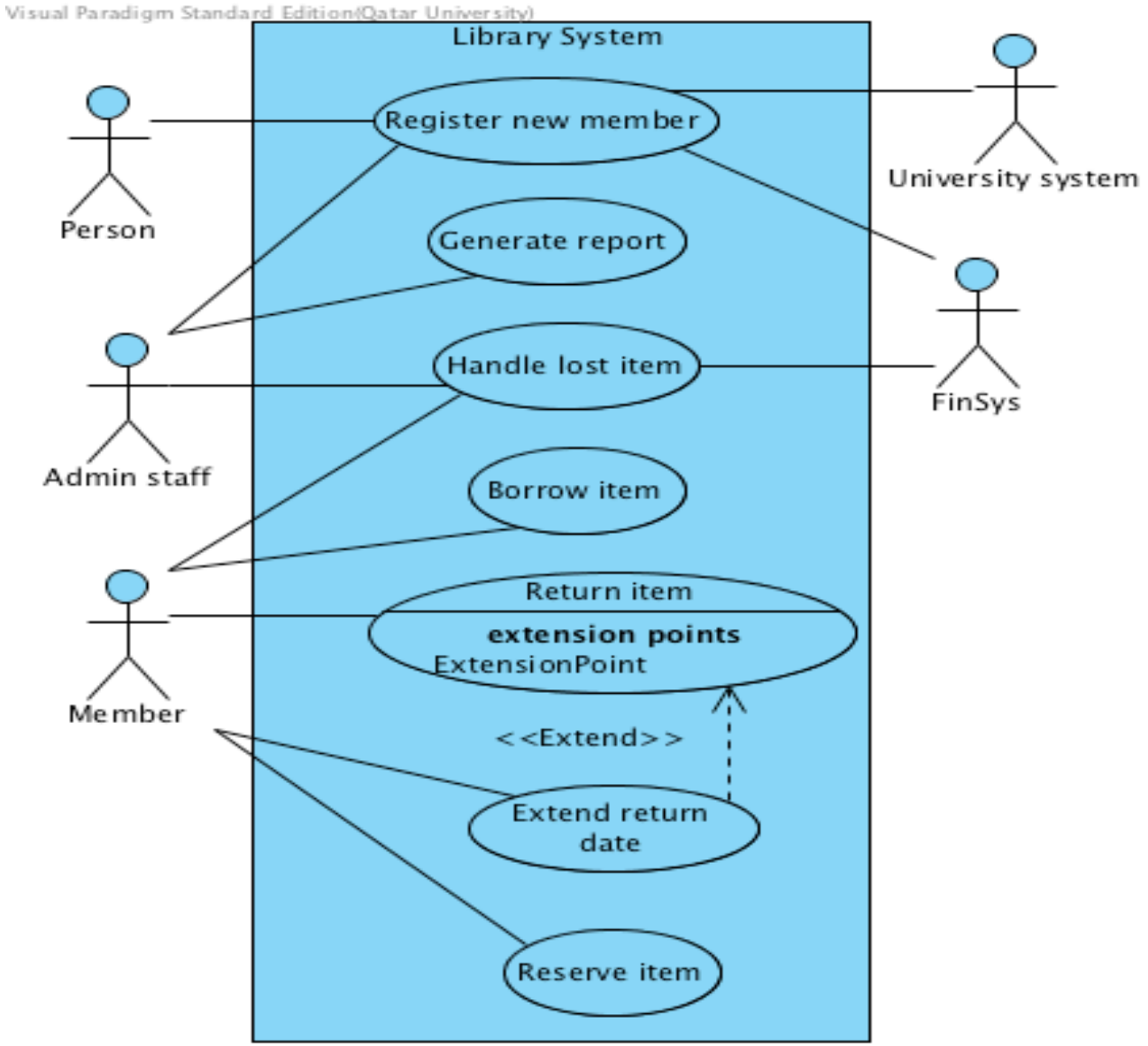
- **8a. If the customer does not have a frequent flyer number, asks the customer to register for a such number.**
- **9a. If the frequent flyer number is not valid, generates an error message**.

Example 4: Complete Use Case Specification
Use Case: "Create Registration"

- Below is an example on the use case "*Create Registration*" specification template.

Use case Id: UC001	**Create Registration**
Brief Description	The client creates a new registration with APS and PoB is informed about the new client.
Primary actors	Client, PoB.
Preconditions: 1. The client must not exist.	
Post-conditions: **1. A client registration was created**	
Main Success Scenario: A client registration has been created.	
Actor Action	**System Response**
1. The client provides details	2. Check if the client exits
	3. Create a registration with the client details if the registration does not exist. (See 3.a. for alternative flow)
	4. Assign a unique number to the registration
	5. Advise PoB about the new client .
	6. Inform the client with the registration number.
Alternative flows: 3.a. If the client already exists, inform the client that a new registration cannot be created.	

A Use Case Diagram of a Library System



Example 5: Complete Use Case Specification
Use Case: "Return Item"

Use case Name:	**Return item**
Brief Description:	This use case begins when a member returns an item to the library. The system accepts the item, update the member account.
Primary actors:	Member
Trigger: The member places the item into the scanner to return it	
Preconditions: 1. The item is recognized 2. The item must be already borrowed.	
Post-conditions: (a) The item was made available (b) The total item borrowed by the member was decremented by 1. (c) The return of the item was recorded in the member account.	
Main Success Scenario: The return of the item was accepted.	

Actor Action	System Response
(a) This use case begins when the member wants to return an item to the library.	
2. The member puts the item into the scanner	3. Retrieve loan details of the item
	4. Check the due date and record the returned date (see 4a)
	5. Update the member account
	6. Make the item available and set the security code of the item
	7. Decrement the total items borrowed of the member by 1
	8. Create a receipt of the return.

Alternative flows (if any):
4a. If the due date has already passed, calculate the fine based on the extra days, and assign the fine to the member account

Example 6: Complete Use Case Specification
Use Case: Extend Return Date

Use case Name:	**Extend return date**
Brief Description:	The member requests to extend the due date of an item. The system grants the extension.
Primary actors:	Member

Trigger: The member

Preconditions:

1. The member must exist
2. The borrow of the item exists with the member item
3. The return date has not passed.
4. The item was not reserved by other member.

Post-conditions:

(a) The return due date of the item was extended
(b) The member account was updated.

Main Success Scenario: The return date was extended sucessfully

Actor Action	System Response
1. The member enters the membership card to the card reader	2. Find the list of all borrowed items
3. The member selects the item	4. Check the current due date of the selected item (see 4a)
	5. Find the reservation status of the item
	6. Extend the due date to one week (see 6a)
	7. Update the member account with the new due date
	8. Prepare a receipt with the new return date.

Alternative flows (if any):

4a. If the due date has already passed, terminates the use case with a message that it cannot extend
6a. If the item has been reserved, terminates with the message that it cannot be extended.

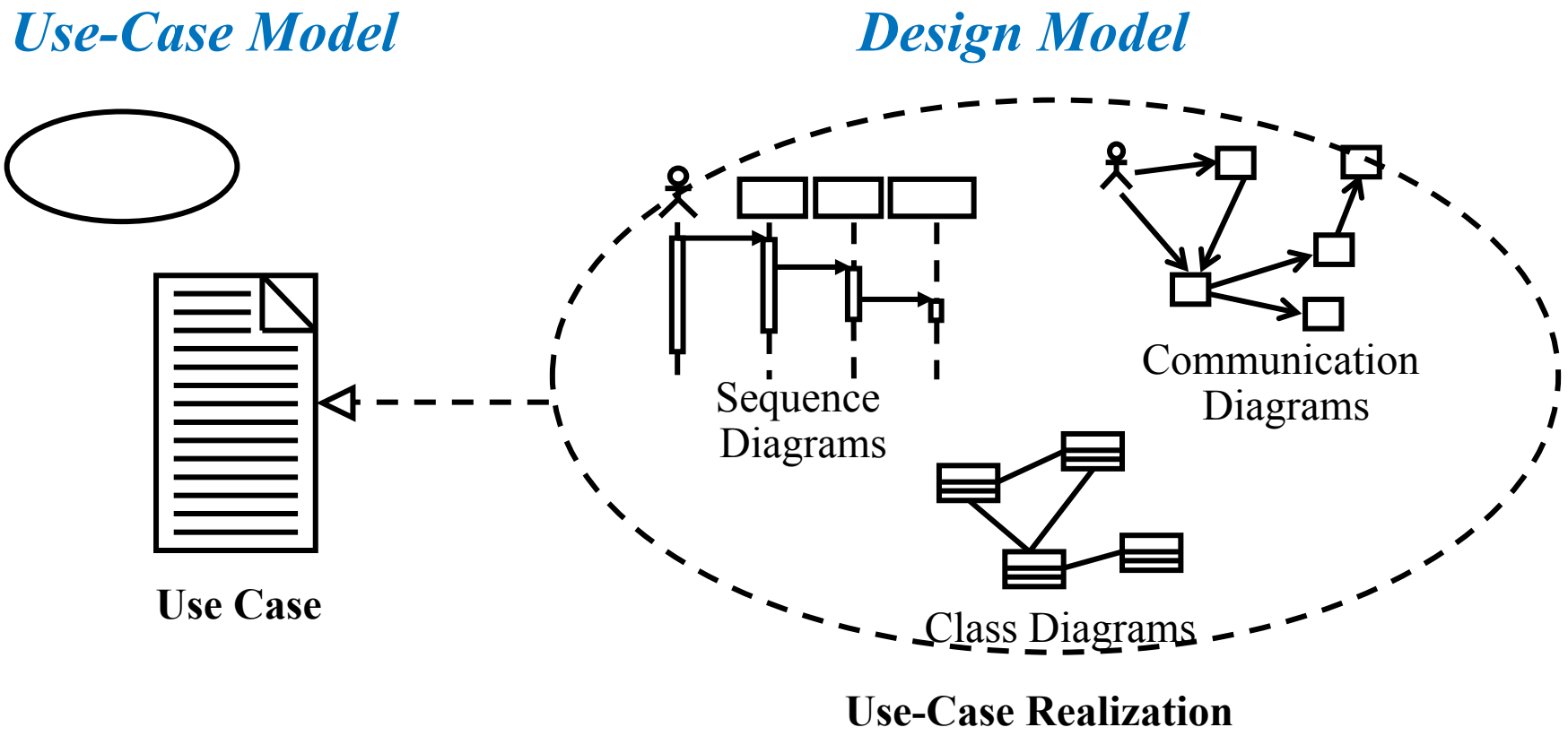
Example 7: “Handle Lost Item” use case

Use case Name:	**Handle lost item**
Brief Description :	The use case deals with lost item. If an item was not returned for more than 10 days from the due date, the item is considered lost, the admin staff is informed, and an amount of QR 500 is deducted from the member account.
Primary actors:	Member, Admin staff, FinSys.
Trigger: Once an item is not returned for more than 10 days of the due date.	
Preconditions: 1. Due date has passed 10 days ago	
Post-conditions: 2. Amount was deducted from the member account 3. The status of the item was changed to lost.	
Main Success Scenario: The lost library item has been processed	

Actor Action	**System Response**
	(a) Check at 10 am everyday if any item was not returned after 10 days of the due date
	2. Inform the admin staff with the item call number (see 2a)
3. The admin staff enters the call number	4 Retrieve the loan details of the item
	5 Change the status of the item to “lost”
	6 Deduct QR. 500 from the member account
	7. Request FinSys to transfer money
8. FinSys forwards transfer information	9. Receive advice from FinSys
	10. Update the member account with the transferred money
	11. Inform the member.

Alternative flows (if any):
2a. If no lost item was identified, terminate this use case.

Use Cases Drive the Development Process



Common Mistakes with Use Cases

Mistake

- Represent individual steps/operation
  - print name
  - read id
  - set initial value
  - check user ID
  - enter name
- Use name starts with noun
- Actor name starts with verb

In the first instance, always check the use case starts and ends at the system boundary.

Common Use Case Pitfalls

- Unclear system boundary
- The Use Case is written from the system view (e.g., describing how and not what)
- The actor names are inconsistent
- There are too many Use Cases (should not be more than 15 use cases for a very software system)
- The use-case scenarios are too long or confusing
- Incorrect description of the Use Case functionality
- The customer doesn't understand the use cases

Summary

- UML introduction
- Use Case diagram
  - <<include>>, <<extend>>
- Use Case Specification
- Pre-condition, post-condition
- Normal scenario
- Alternative flows
- Actor-System Use Case Specification Table