

OOAD

Introduction

Defining Requirements

- What the application
 - requires to do
 - What must the app. do
- Functional Requirements
 - What does app do? **Features / Capabilities**
- Non Functional Requirements
 - What else?
 - Help**
 - Performance**
 - Support**
 - Security**

Functional Requirements Example

- Phrase starts with like
 - System must or application must or program must

Program must allow receipts to be generated by via e-mail

System must display the heart rate, temperature and blood pressure of a patient connected to the patient monitor

Application must allow user to search by customer's last name or telephone number

System must automatically produce monthly comparative analysis report. **Report must** be in PDF format and automatically emailed to everyone on first day of the month.

Non-Functional Requirements Example

System must respond to searches within 2 seconds.

Help desk available by telephone. Mon – Fri 8 am – 6 pm

FURPS / FURPS

- **Functional** requirements →
- **Usability** requirements →
- **Reliability** requirements →
- **Performance** requirements →
- **Supportability** requirements →

+ Implementation requirements
Physical requirements, Interface requirements

We need
**absolute
minimum
set of
requirements**

Not what is nice
to have, not
what is
optional

Unified Modeling Language (UML)

- Not a programming language, a graphical notation – specifically for drawing diagrams for object oriented systems.
- UML diagrams are
 - quick, useful communication tool
 - Support system for brain

Use Cases

- A **set of scenarios** related by a common actor and a goal
- A description of sequences of actions performed by a given system to produce a result for an actor
- Use cases **specify the expected behaviour [what]**, and **not the exact method of making it happen [how]**
- Use Cases should have the following three things:

Title

What is the goal ?

Actor

Who desires it ?

Scenario

How it is accomplished ?

Use Cases : Title

- **Short Phrase, active verb**

Register new member

Transfer funds

Purchase items

Collect late payments

Use Cases : Actor

- Represent roles that humans, hardware devices, or external systems play while interacting with a given system

User

Customer

Member

Administrator

ATM Machine

Use Cases: Scenario as paragraph

Title: Purchase Items

Actor: Customer

Scenario(ঘটনাবিন্যাস): Customer reviews items in shopping cart. Customer provides payment and shipping information. System validates payment information and responds with confirmation of order and provides order number that Customer can use to check on order status. System will send Customer a confirmation of order details and tracking number in an email.

Use Cases: Scenario as Steps

Title: Purchase Items

Actor: Customer

Scenario:

1. Customer chooses to enter the checkout process.
2. Customer is shown a confirmation page for their order, allowing them to change quantities, remove items, or cancel.
3. Customer enters his/her shipping address.
4. System validates the customer address.
5. Customer selects a payment method.
6. System validates the payment details
7. System create an order number that can be used for tracking
8. System displays a confirmation screen to the customer
9. Email is sent to the customer with order details.

Use Cases: Scenario **Additional Details**

Title: Purchase Items

Actor: Customer

Scenario:

Scope:

Level:

Extensions: Describe steps for out-of-stock situations

Extensions: Describe steps for order never finalized

Preconditions: Customer has added at least **1** item to the shopping cart

Postconditions:

Stakeholders:

Example of Use Cases

Use case Name: User Login to the System

Actor: User

Scenario / Description:

1. Input user ID and password.
2. Check User ID and password (User authentication).
3. Save ID and password (Cookies).
4. Remember Password.

Exception:

1. Required page not found.
2. No database connection.

Precondition:

1. URL of login page from web browser.

Post Condition:

1. **Successful login** – Logging in to the system.
2. **Unsuccessful login** – Stay in same page showing error message.
3. **Exception** – Stay in same page if possible.

Example of Use Cases

Use case Name: Create New User Account

Actor: User

Description:

1. Input user ID, password, confirm password, name, address, date of birth.
2. Check User ID (Unique or not).
3. Store new User account into database.

Exception:

1. Required page not found.
2. No database connection.

Precondition:

1. URL of “New User” hyperlink page.

Post Condition:

1. **Successful** – Create user successfully.
2. **Unsuccessful**– Stay in same page, don't create any user showing failure message / correction message.
3. **Exception** - Stay in same page if possible.

Use Case Diagram

- Use Case diagram shows a set of use cases and actors and their relationships.
- A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case.
- These diagrams are important in organizing and modeling the behaviors of a system.

Use Case Diagram

- Use Case Diagrams have only 4 major elements:
 - The **actors** that the system you are describing interacts with,
 - the **system boundary** itself, Identify an implicit separation between actors (external to the system) and use cases (internal to the system)
 - the **use cases**, or services, that the system knows how to perform, and
 - the **lines** that represent **relationships** between these elements.

Use Case Diagram

- **Actor** is represented by a labeled stick figure, or a class rectangle.



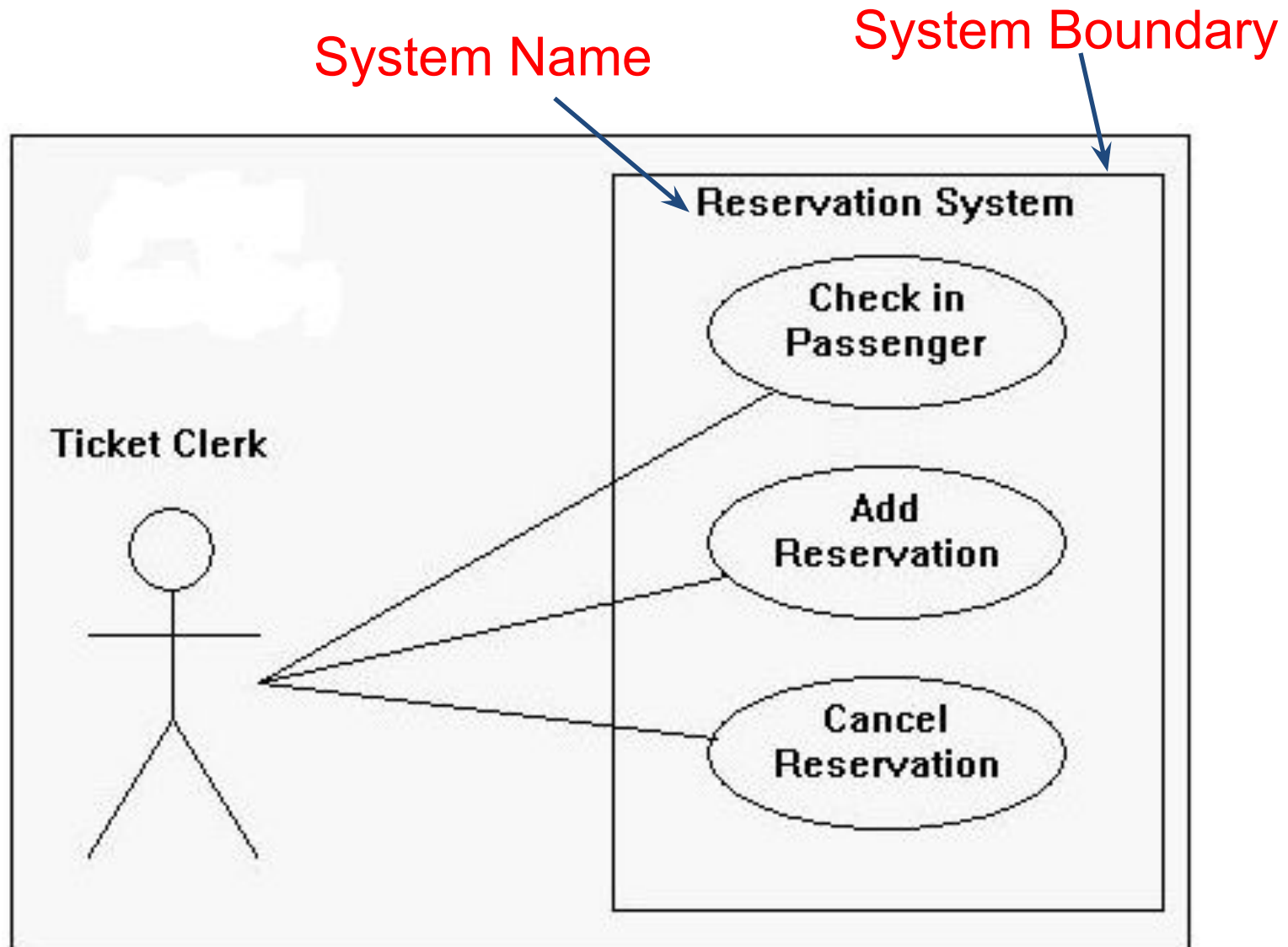
- **Use cases** are represented by a labeled ellipse.



Use Case Diagram

- A typical UML Use Case Diagram will be composed of many diagrams and sub-diagrams, and should contain use case ovals, one for each **top-level** service that the system provides to its actors. Any kind of internal behavior that the system may have that is only used by other parts of the system should **not** appear in the system boundary.
- The system boundary only appears on the top-level diagram.

Use Case Diagram



Use Case Relationships

- Include
- Extend
- Generalizations

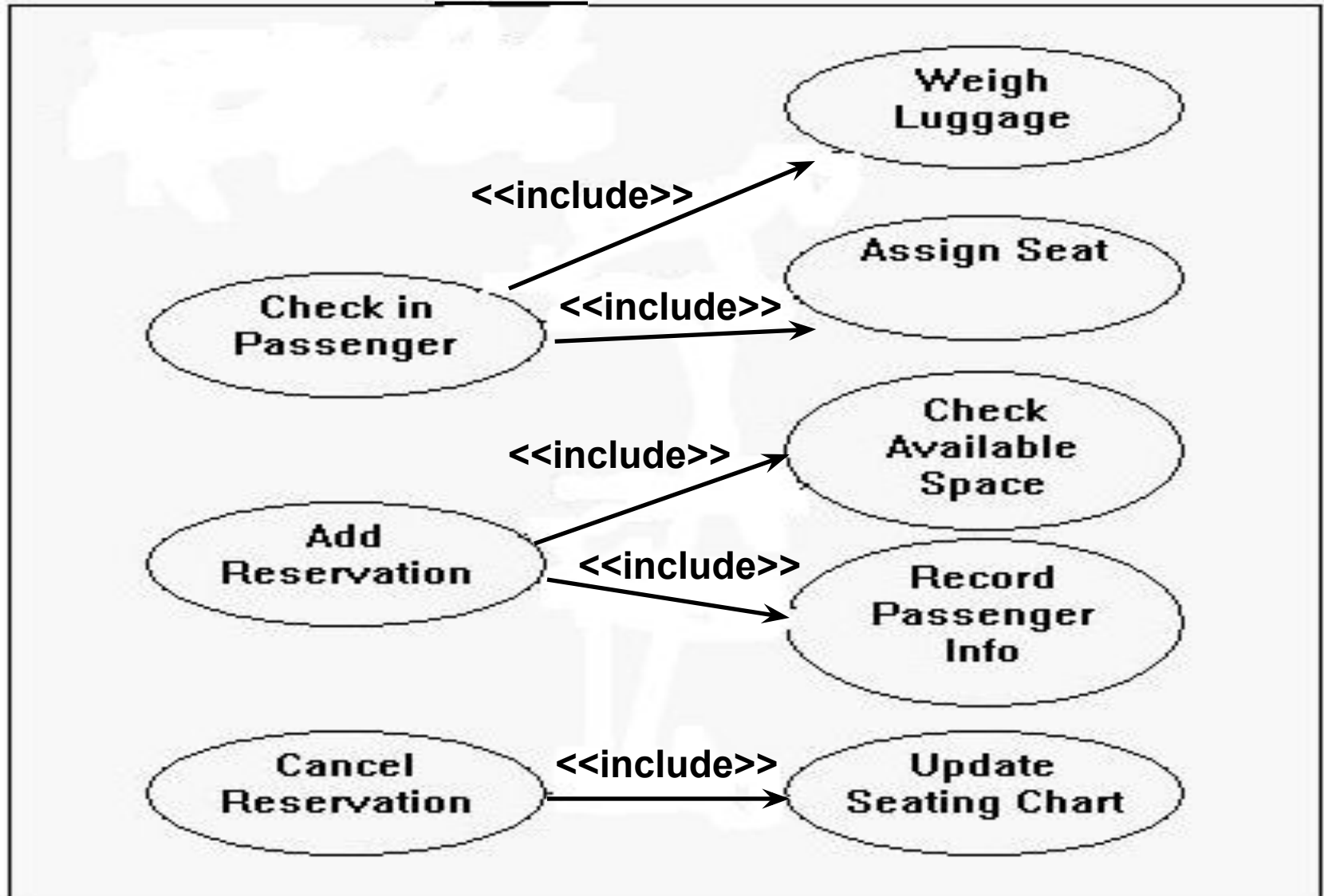


Use Case Relationships (**Include**)

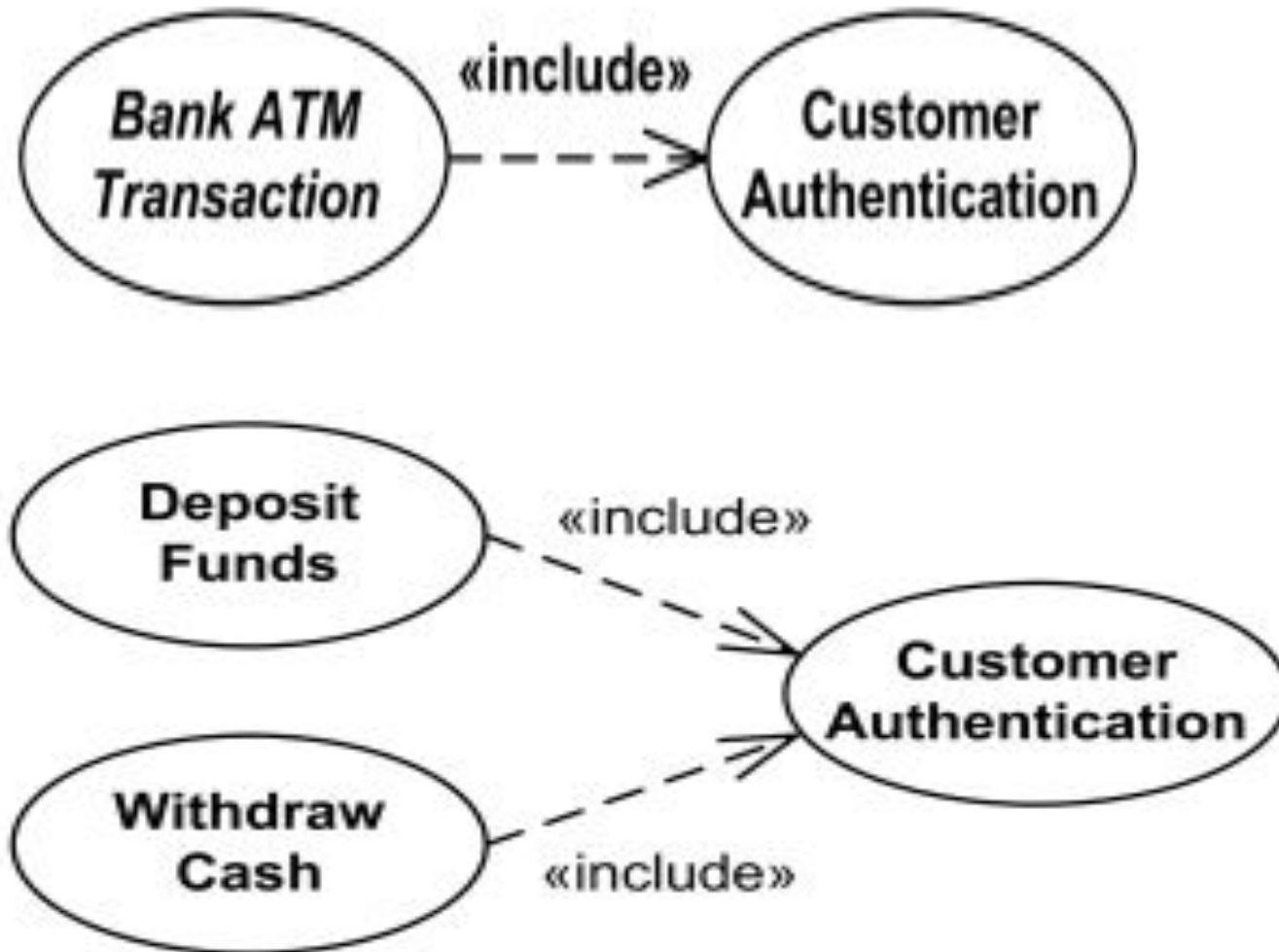
- **Include** relationship

- The *include relation* is drawn from a use case X to another use case Y to indicate that *the process of doing X always involves doing Y at least once*.
- If a certain use case *includes* several others, that means that all of the component use cases must be completed in the process of completing the aggregate use case (although there is no specification in Use Case Diagrams of the order in which these are completed).
- In brief, it can be read X uses Y means that "X *has a Y*" as part of it's behavior.

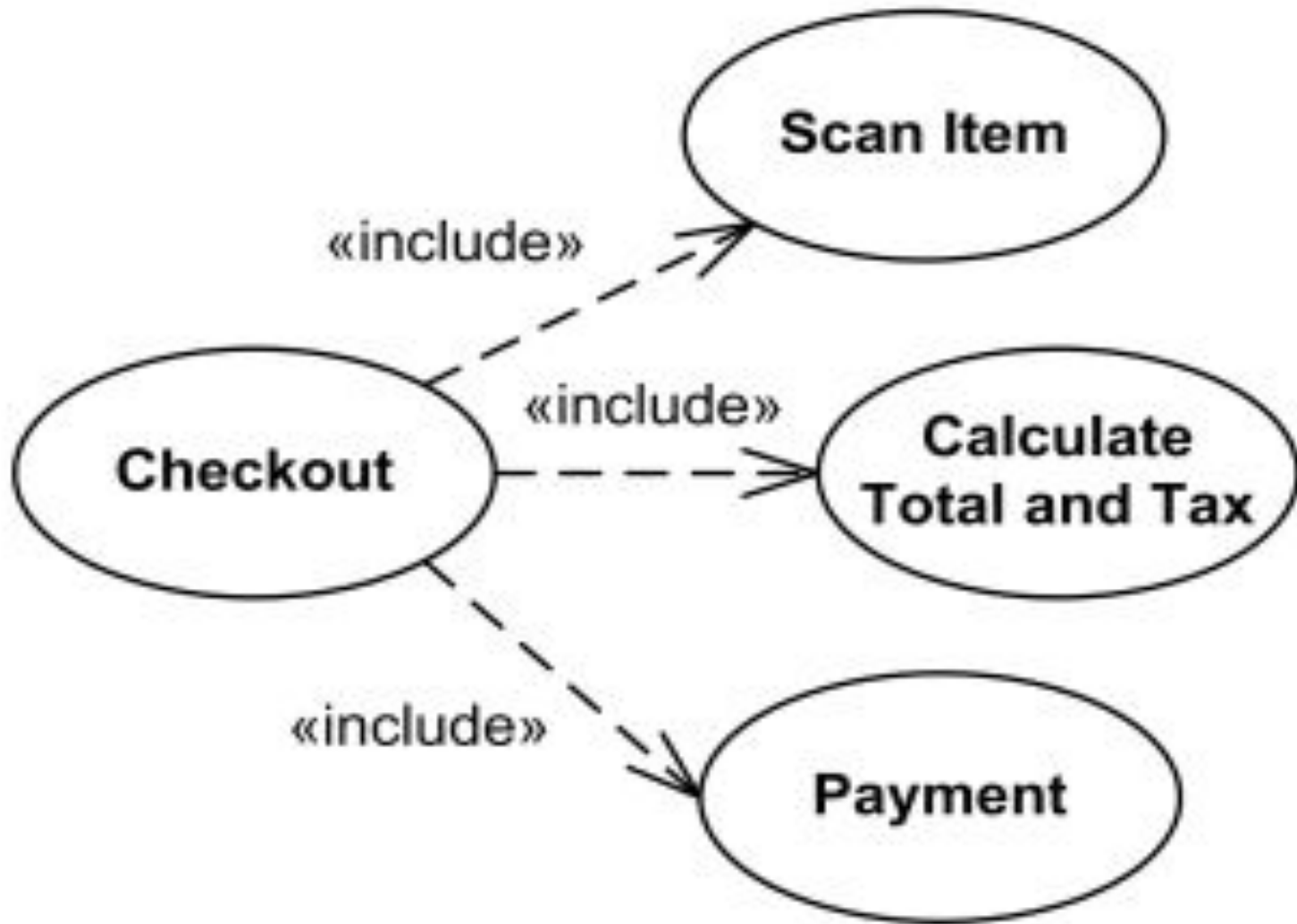
Use Case Relationships (**Include**)



Use Case Relationships (**Include**)



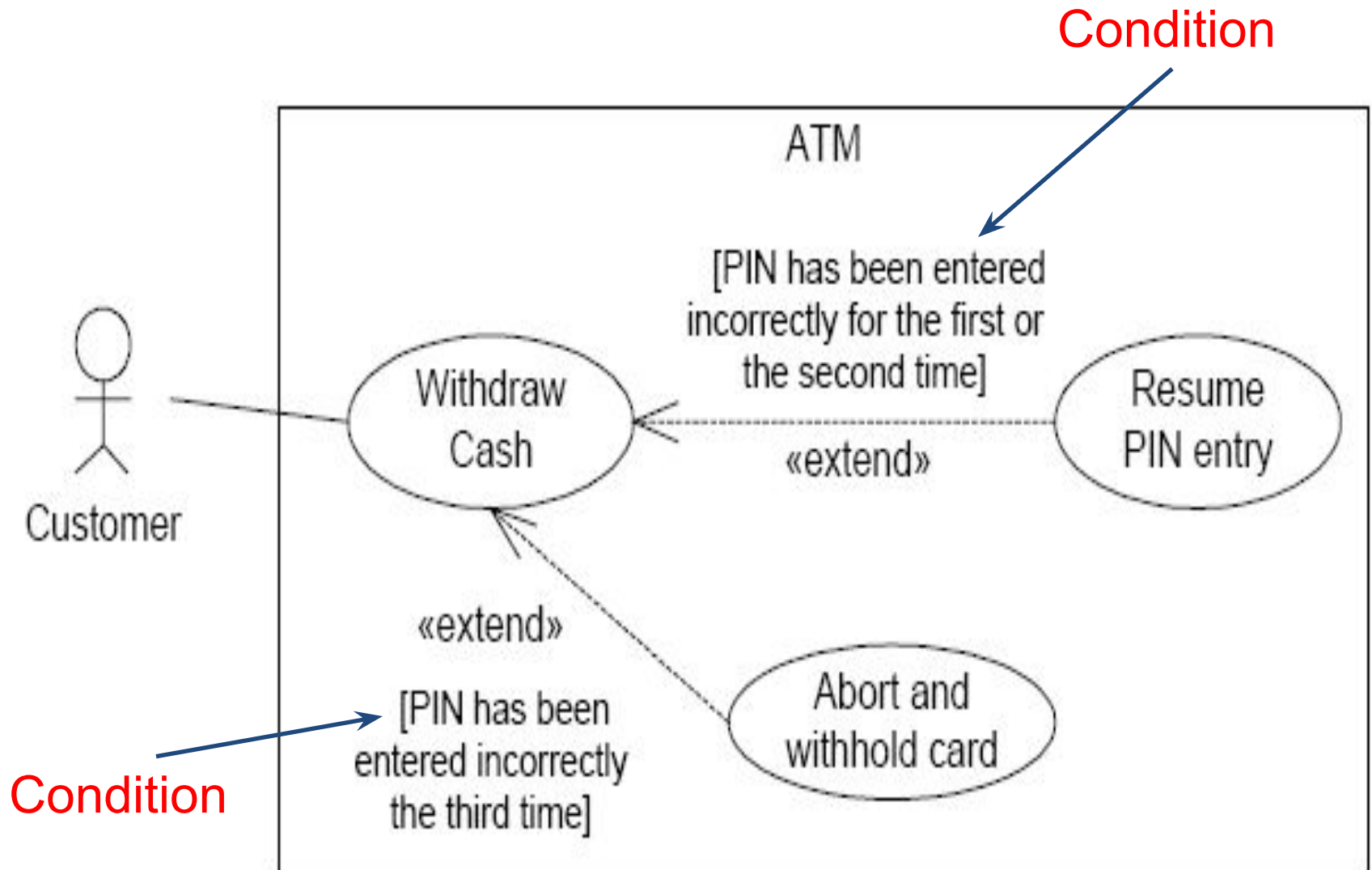
Use Case Relationships (**Include**)



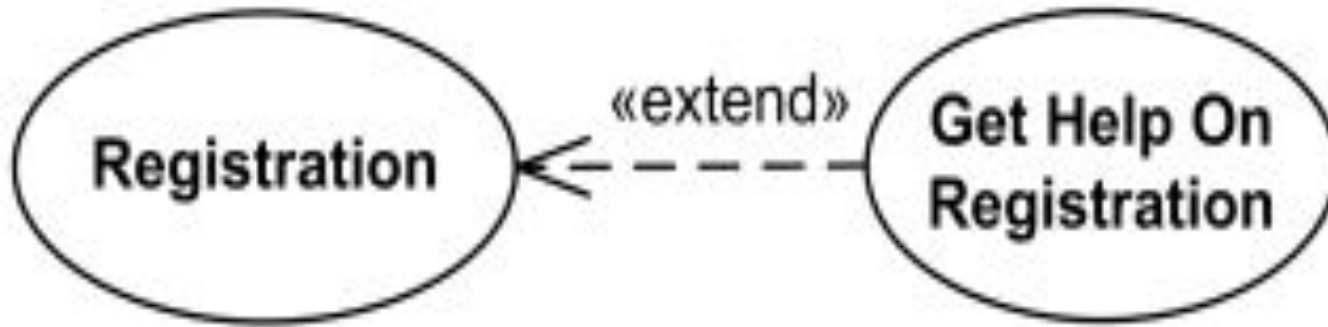
Use Case Relationships (**Extend**)

- **Extend** relationship
 - connects an extension use case to a base use case.
 - extending use case **continues the behavior of a base use case**. The extending use case accomplishes this by conceptually inserting additional action sequences into the base use-case sequence.
 - **Extending** use case typically defines **optional** behavior.

Use Case Relationships (**Extend**)



Use Case Relationships (**Extend**)



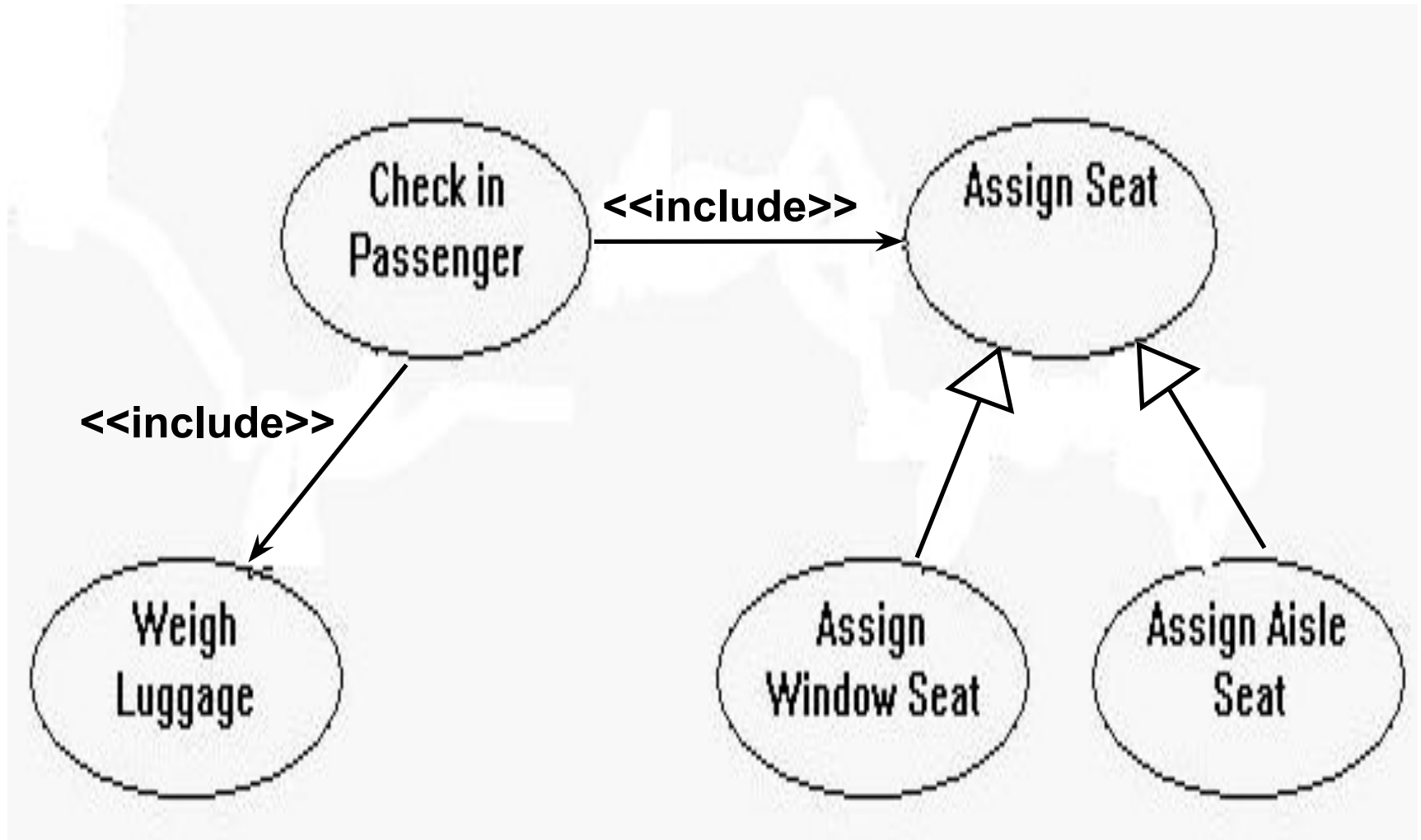
Registration use case is complete and meaningful on its own.

It could be **extended** with **optional** **Get Help On Registration** use case.

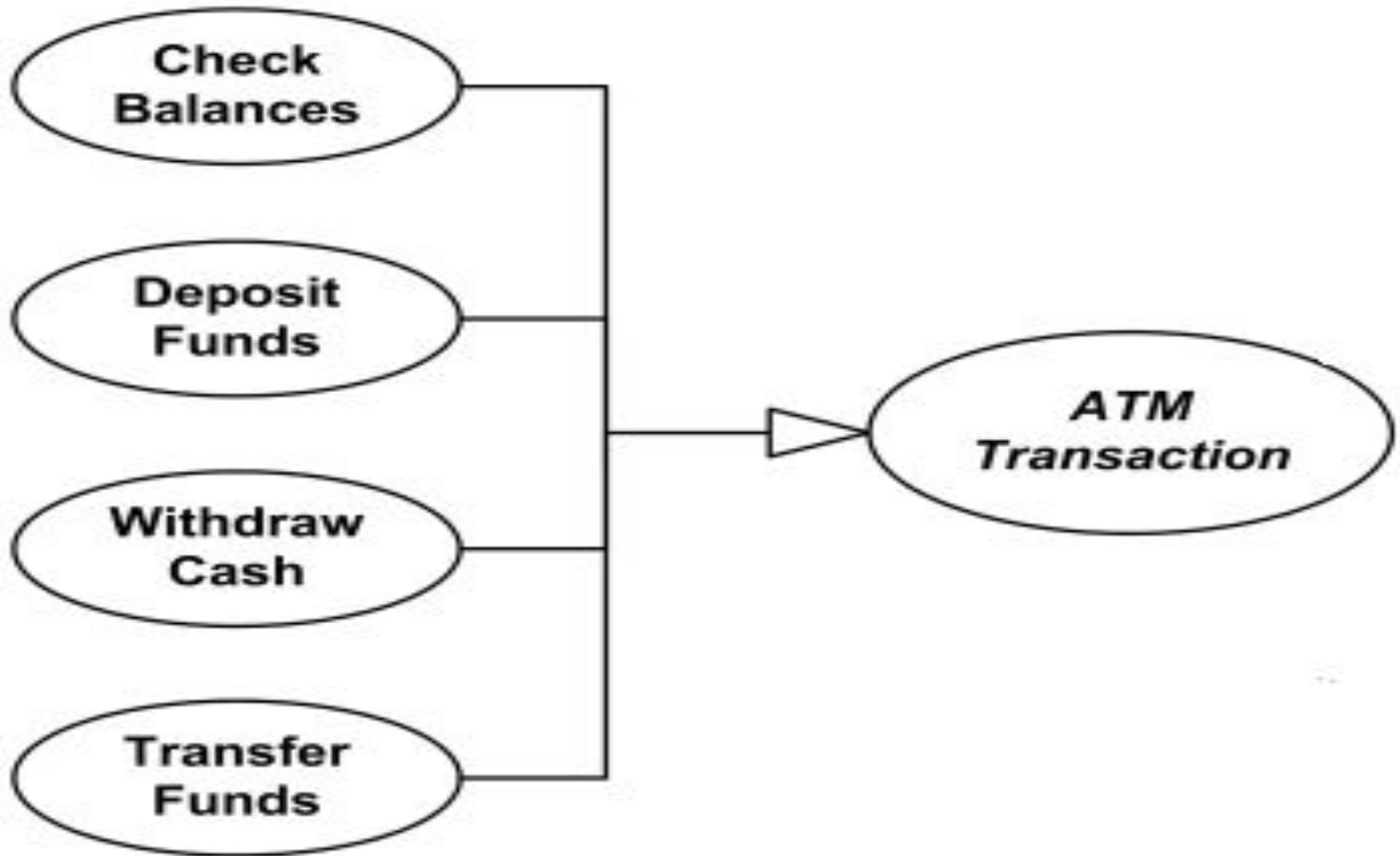
Use Case Relationships (**Generalizations**)

- **Generalizations** relationship
 - The *generalizations relationship* is drawn from a use case X to a use case Y to indicate that the process X is a special case behavior of the same type as the more general process Y.
 - A generalization relationship is a parent-child relationship between use cases in which the child use case is an enhancement of the parent use case.
 - Child use case is effectively, **an alternate course of the base use case.**
 - Base use case could be abstract use case

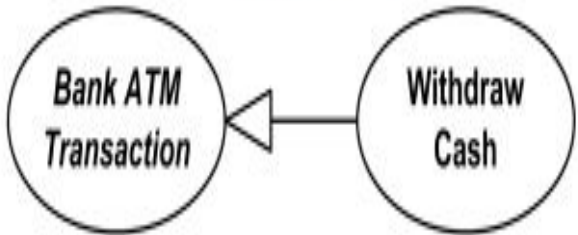
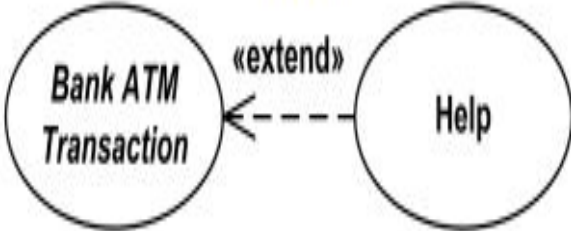
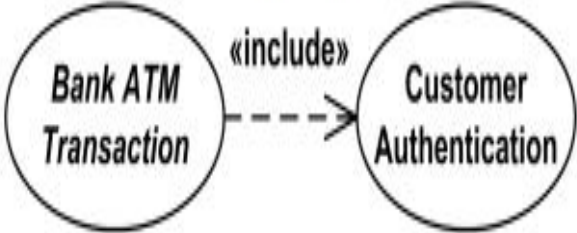
Use Case Relationships (Generalizations)



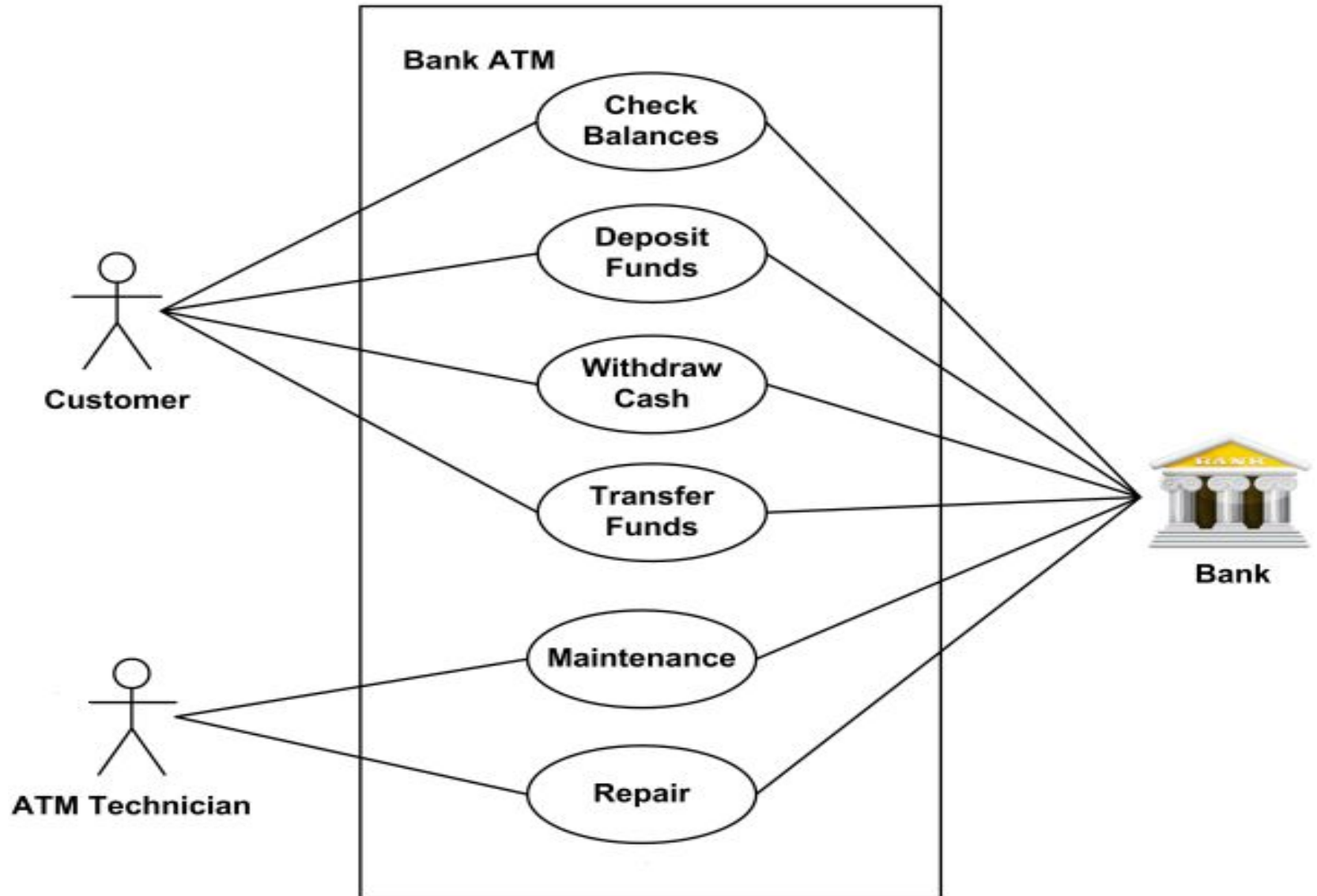
Use Case Relationships (**Generalizations**)



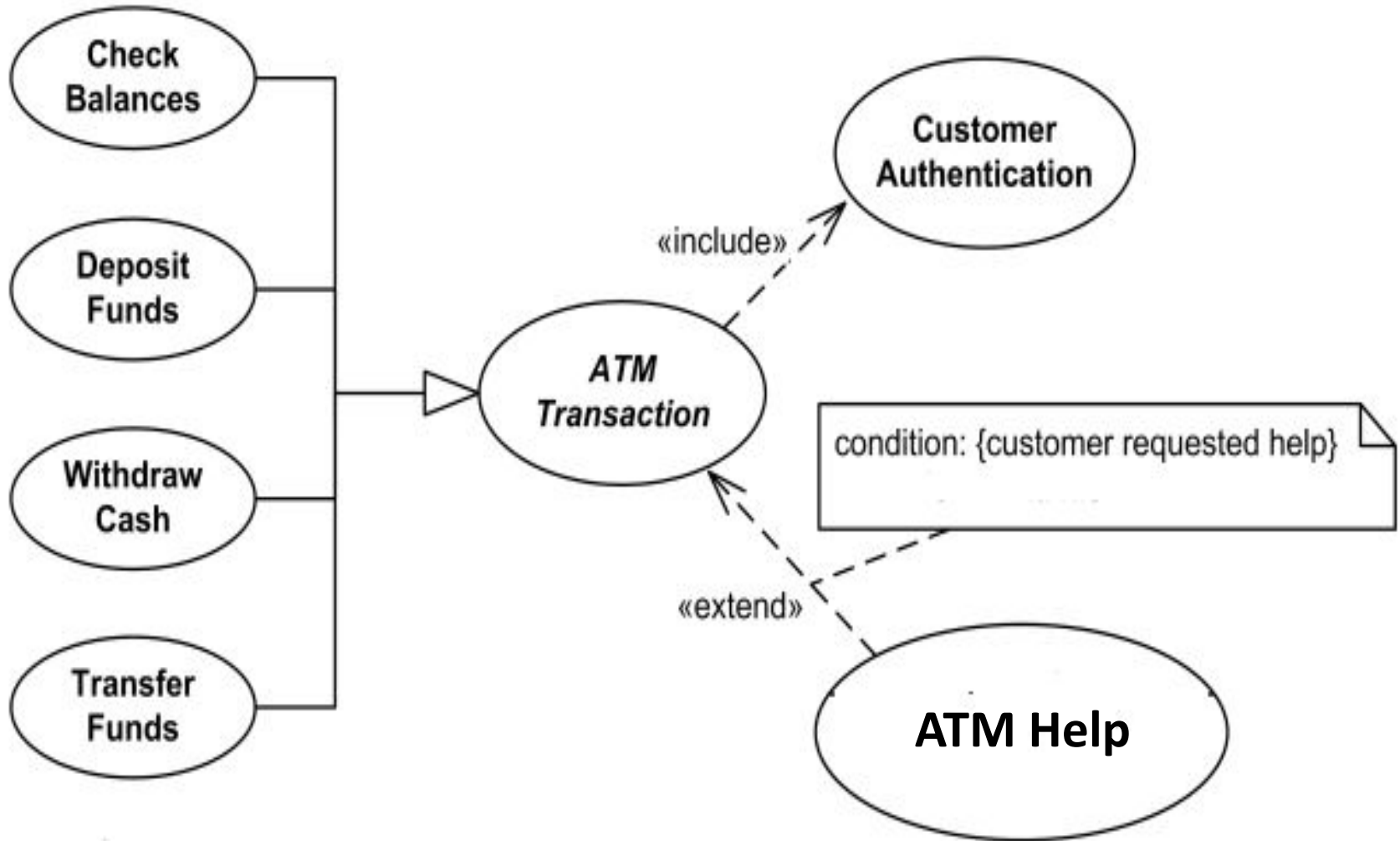
Difference between *use case* relationships

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.
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 condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

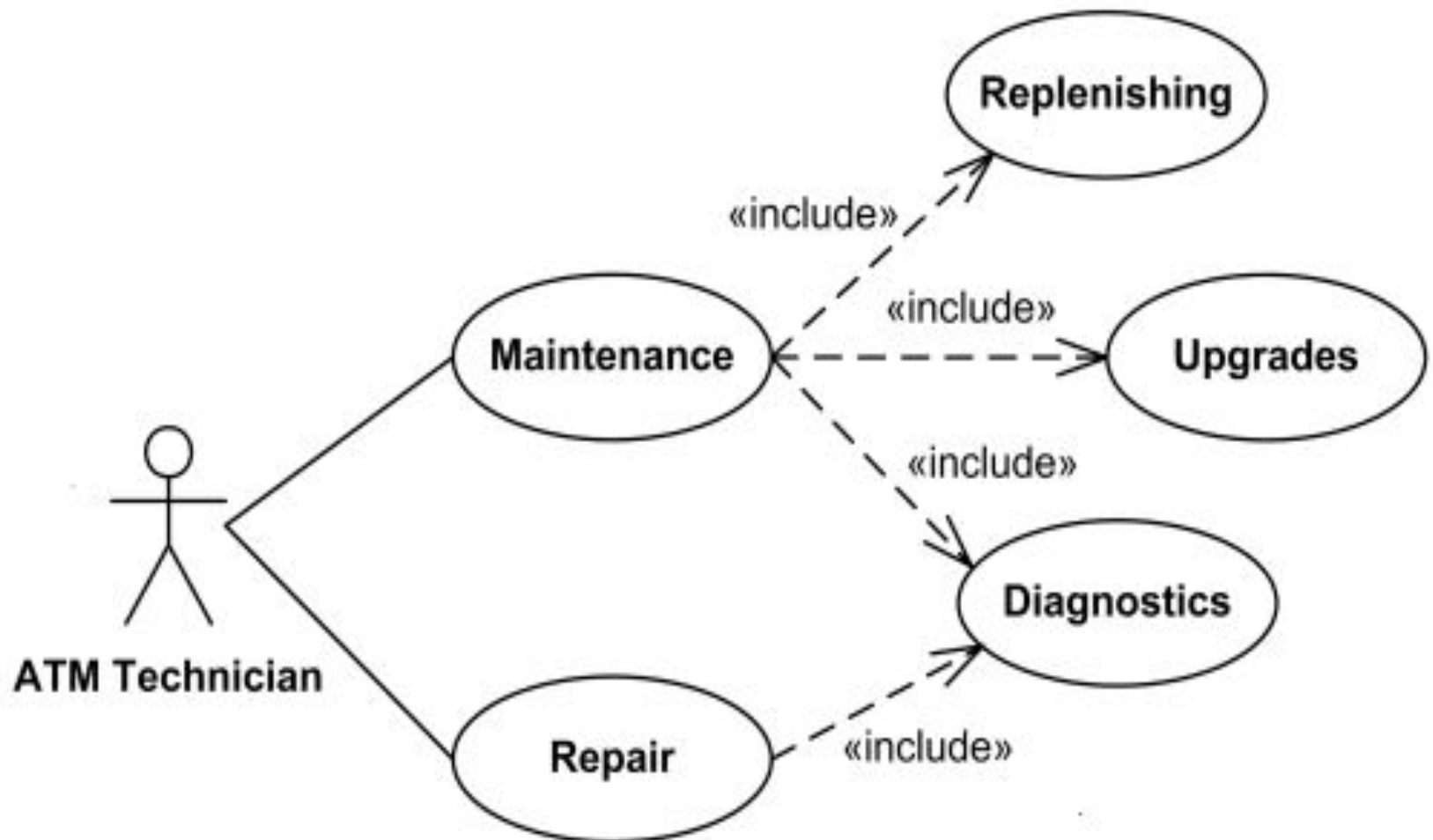
Bank ATM

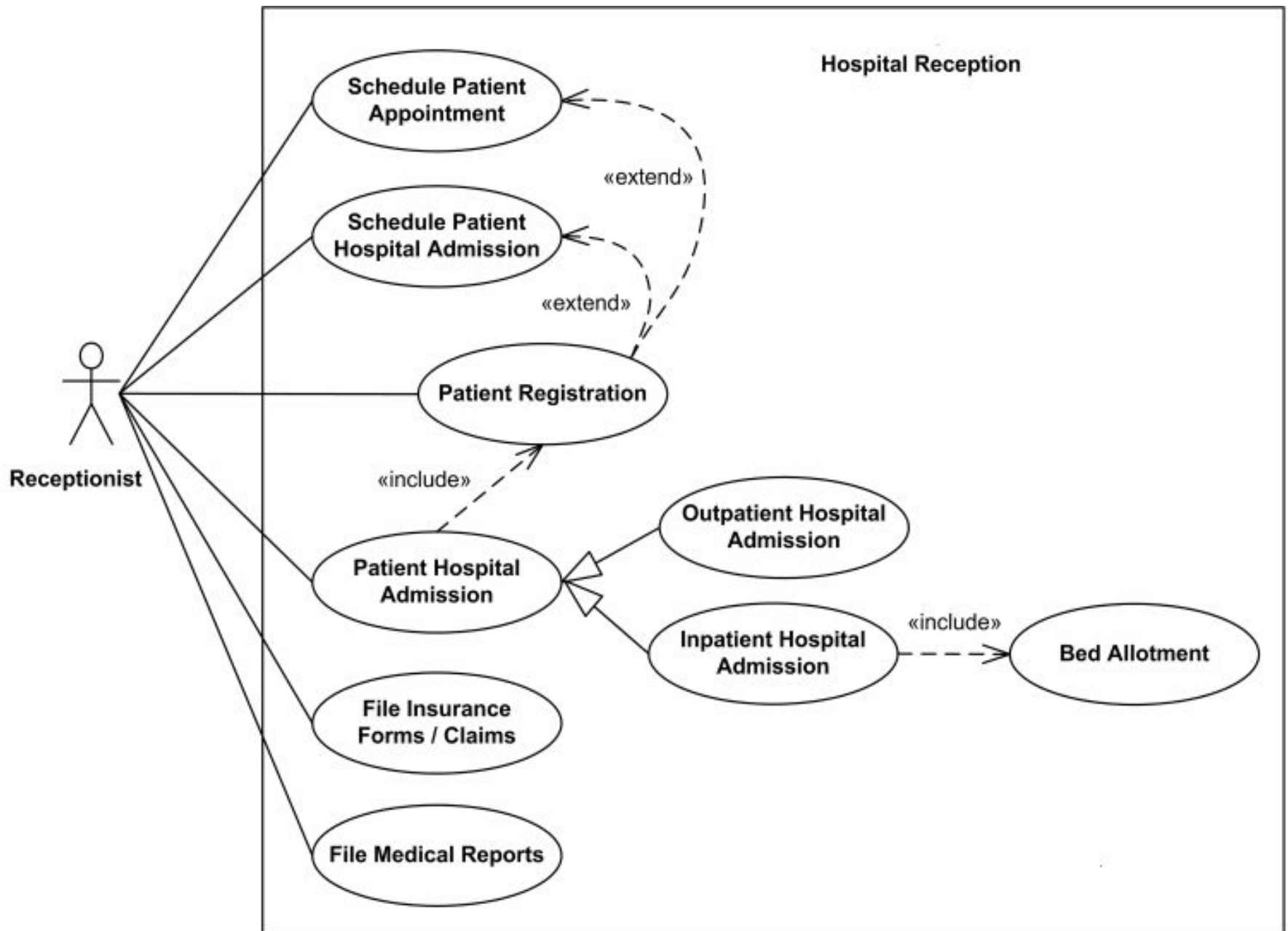


Bank ATM



Bank ATM





Identifying Classes



- Perform a “grammatical parse” on a description of the system to be built.
- Classes are determined by underlining each noun or noun clause and entering it in a simple list –these are candidate classes.
- Synonyms should be noted.

Identifying Classes

- Use Case Scenario

Customer confirms items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by email.

Identifying Classes

- Use Case Scenario

Customer confirms items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by email.

Identifying Classes

- Noun List

Customer Order

Item ~~Order Number~~

Shopping cart ~~Order Status~~

Payment ~~Order Details~~

Address Email

~~Sale~~ System

Identifying Classes

Consider the following requirements:

Req#	Requirement	Candidate Classes
1	The customer uses the manufacturer's online shopping web page to view the standard configuration of the chosen computer ...	Customer StandardConfiguration Computer
2	The customer chooses to view the details of the configuration , with the intention to buy it as is or to build a more suitable configuration ...	Customer ConfiguredItem ConfiguredComputer

Common Class Patterns Approach

- Example patterns are:
 - External entities that produce or consume information to be used by a computer-based system (e.g. other systems, devices, people)
 - Things that are part of the information domain for the problem (e.g. reports, displays, letters, signals)

Common Class Patterns Approach

- **Concepts** that are shared and agreed by a large community of people (e.g. Reservation in an airline reservation system).
- **Occurrences or events** that occur within the context of system operation (e.g. Arrival is an event class in an airline reservation system).
- **Roles** played by people who interact within the system (e.g. manager, engineer, salesperson).

Common Class Patterns Approach

- Consider the following functional requirement:
 - “After a *customer’s order* has been entered into the system, the *salesperson* sends an *electronic request* to the *warehouse* with *details of the ordered configuration*”
- Candidate classes:
 - External entity: customer, warehouse
 - Concept: order
 - Thing: configuredComputer
 - Occurrence: an electronic request
 - Role : salesperson