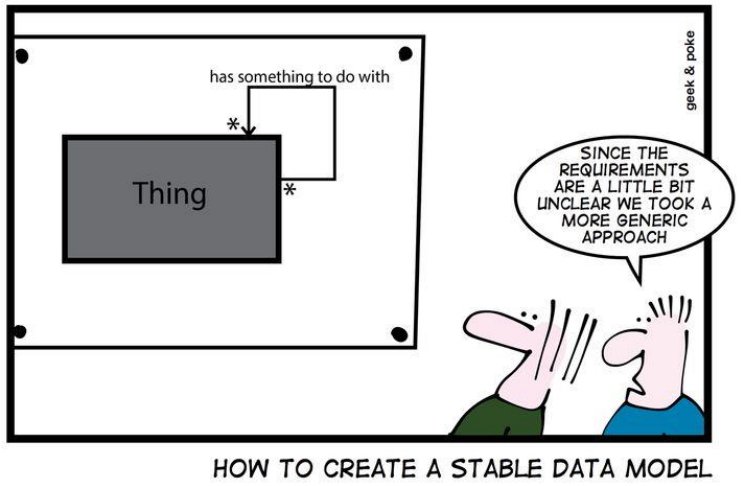
University
Mohammed VI
Polytechnic



Introduction To UML

Pr. Imane Fouad

Course Objectives

Understand	Understand the importance of modeling in software development
Introduce	Get introduced to the UML standard (Unified Modeling Language)
Discover	Discover the different types of UML diagrams

Course Outline

Section 1: Modeling

Section 2: Introduction to UML

Section 3: Use Case Diagram

Modeling

Section 1

What is a Model?

A **model** is a **representation of the real world**

Models describe a system with **different levels of detail**

A model is an **abstraction**

It provides a **description or analogy** that helps us understand something **difficult to observe directly**.

Models often use a **simple and graphical notation** to make ideas easy to read and share.

What is Modeling?

The act of **designing or creating a model**

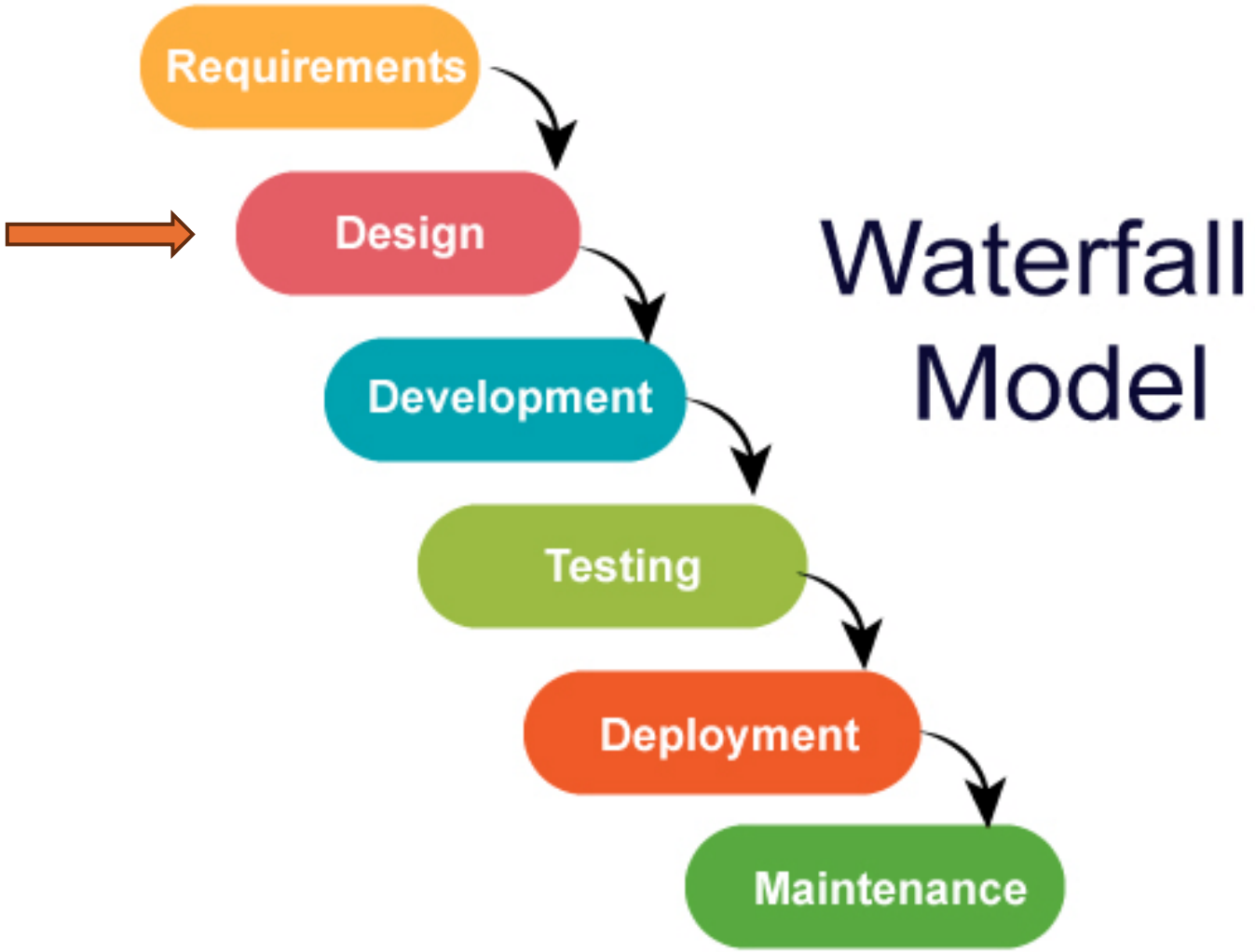
Done using a **dedicated modeling language** (such as UML)

Helps represent the **structure, behavior, and interactions** of a system clearly before implementation

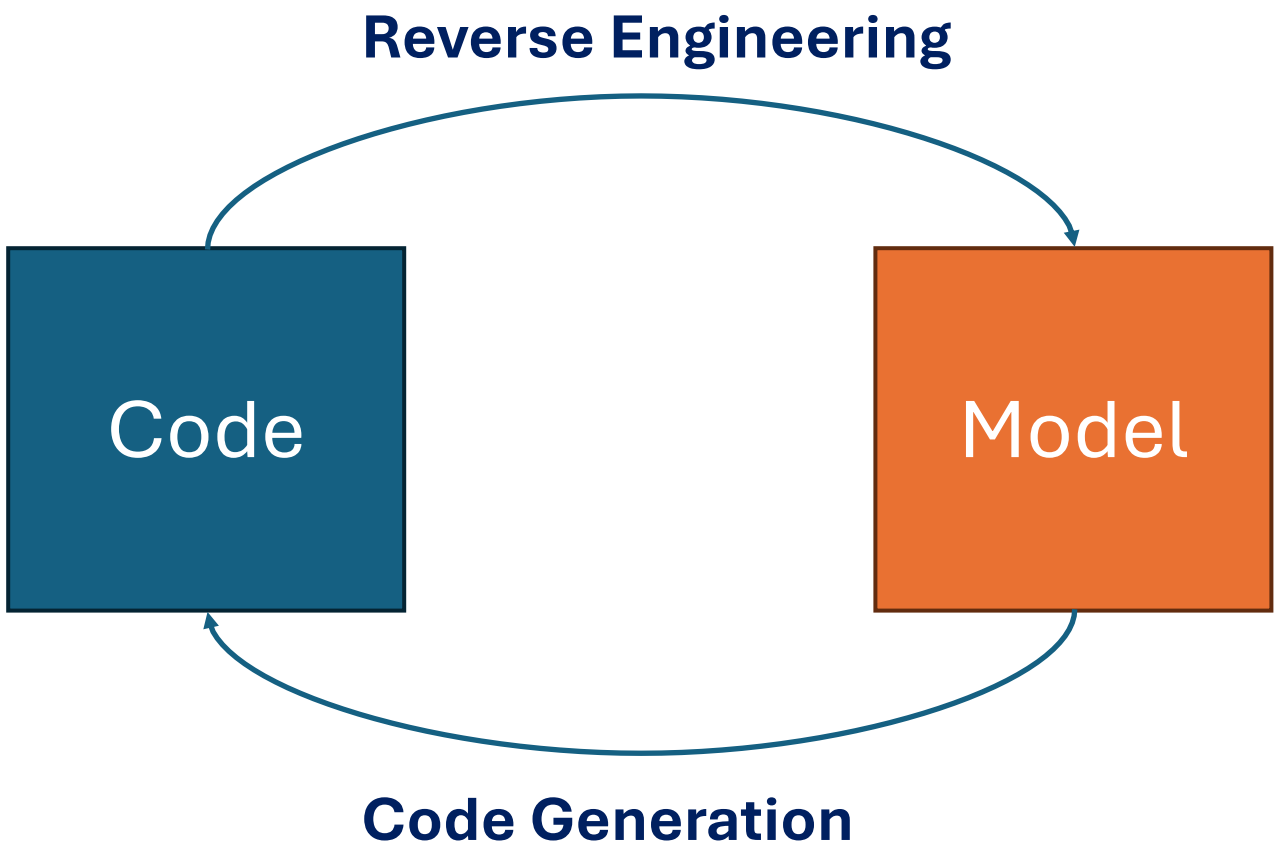
Why ?

Represent	Represent the system effectively
Reduce	Reduce costs
Enable	Enable different perspectives
Make	Make observation easier
Use	Use simple graphical notation
Facilitate	Facilitate communication
Simplify	Simplify complex aspects
Standardize	Standardize the language

Software Development Process



Transformations Between Code and Model



Introduction to UML

Section 2

Origin of UML

There were several modeling methods, and there was a need for standardization.

UML is the **fusion of the work of several modeling experts**.

UML was **standardized by the OMG**.

UML is a **modeling language, not a methodology**

Why UML?

UML is **graphical**

UML is **simple**

UML is a **standard**

Diagrams in UML

A **diagram** allows you to **visualize a model** from a specific perspective.

A diagram is a **partial view of the model**.

A model can contain **1, or many diagrams**.

A model contains **different types of diagrams**.

Each type of diagram focuses on a **specific aspect of the system**.

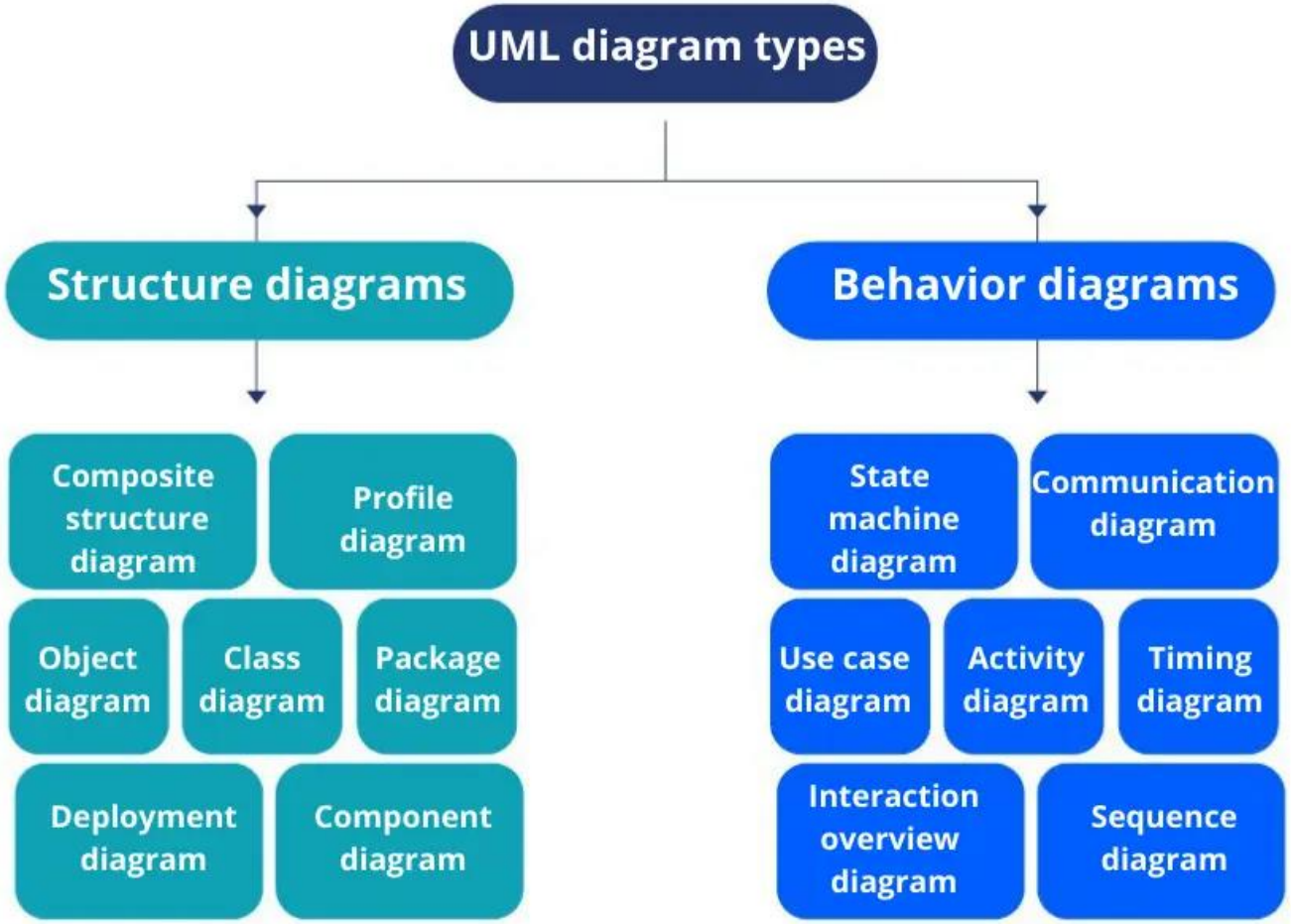
A model can contain **multiple diagrams of the same type**.

Each diagram is used in **one or more stages of the software life cycle**.

A diagram should be **compact, readable, and expressive**.

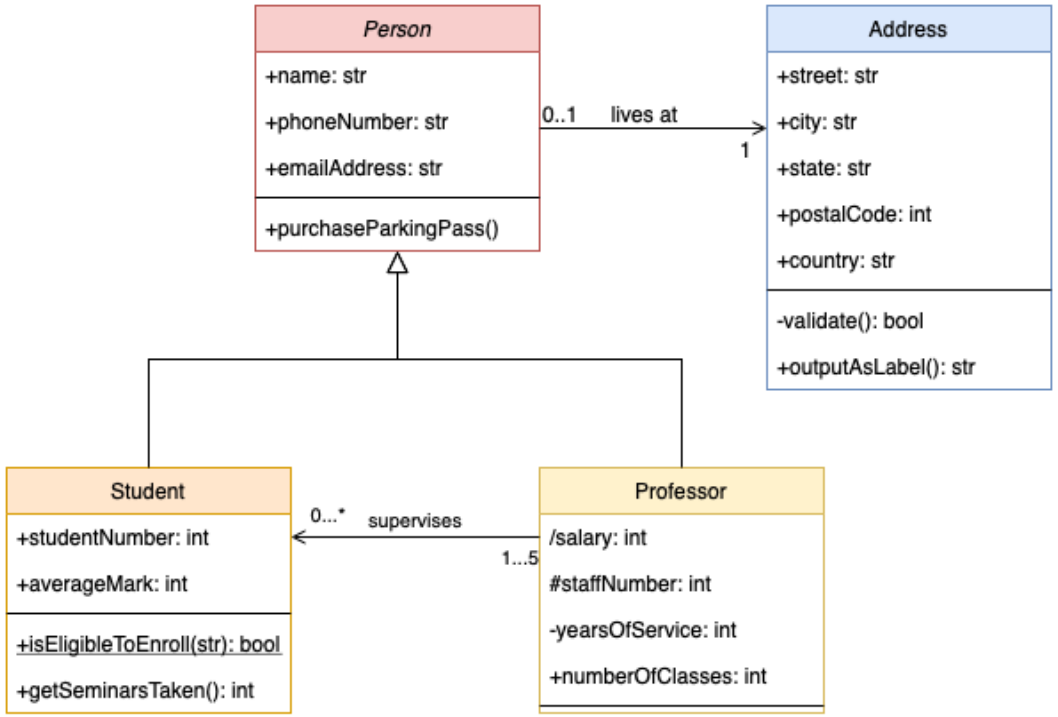
UML 2.5 contains 22 diagrams

Types of UML Diagrams



Class Diagram

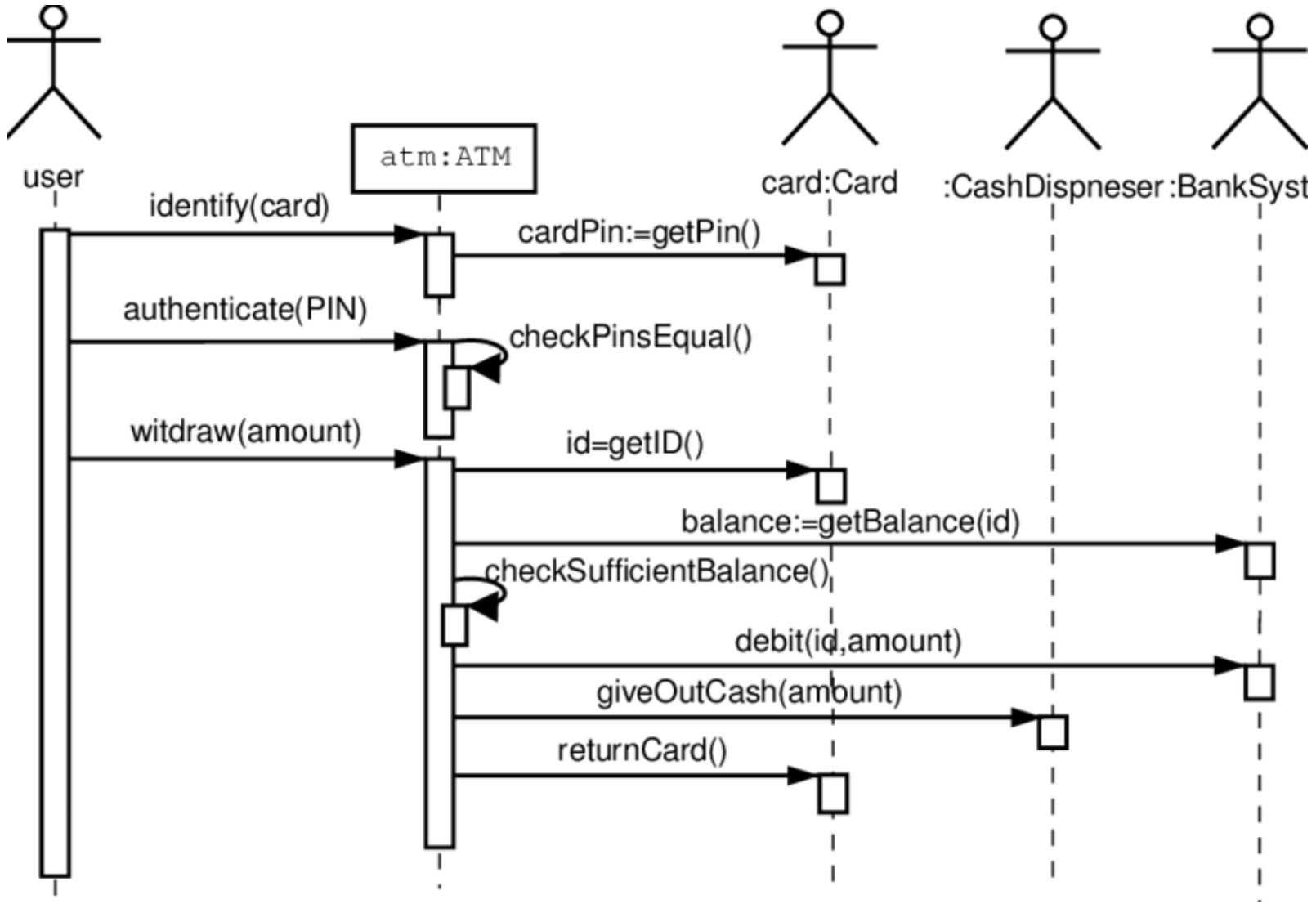
- Often considered the **most important UML diagram**
- Defines the **classes**, their **attributes**, and their **relationships**
- Describes the **overall design and structure** of the system



Sequence Diagram

- A sequence diagram is an interaction diagram that shows how each process (or object) interacts with others and in what order.
- The Sequence Diagram (SD) illustrates interactions along a time axis.
- It lists the objects or participants involved in the interaction to achieve a specific goal.
- The diagram focuses on the temporal order of messages exchanged between participants.

Sequence Diagram



Use Case Diagram

Provides a **view of the system** in terms of **actors and their goals**.

The main purpose of a **Use Case Diagram** is to **identify which functions are performed by each actor**

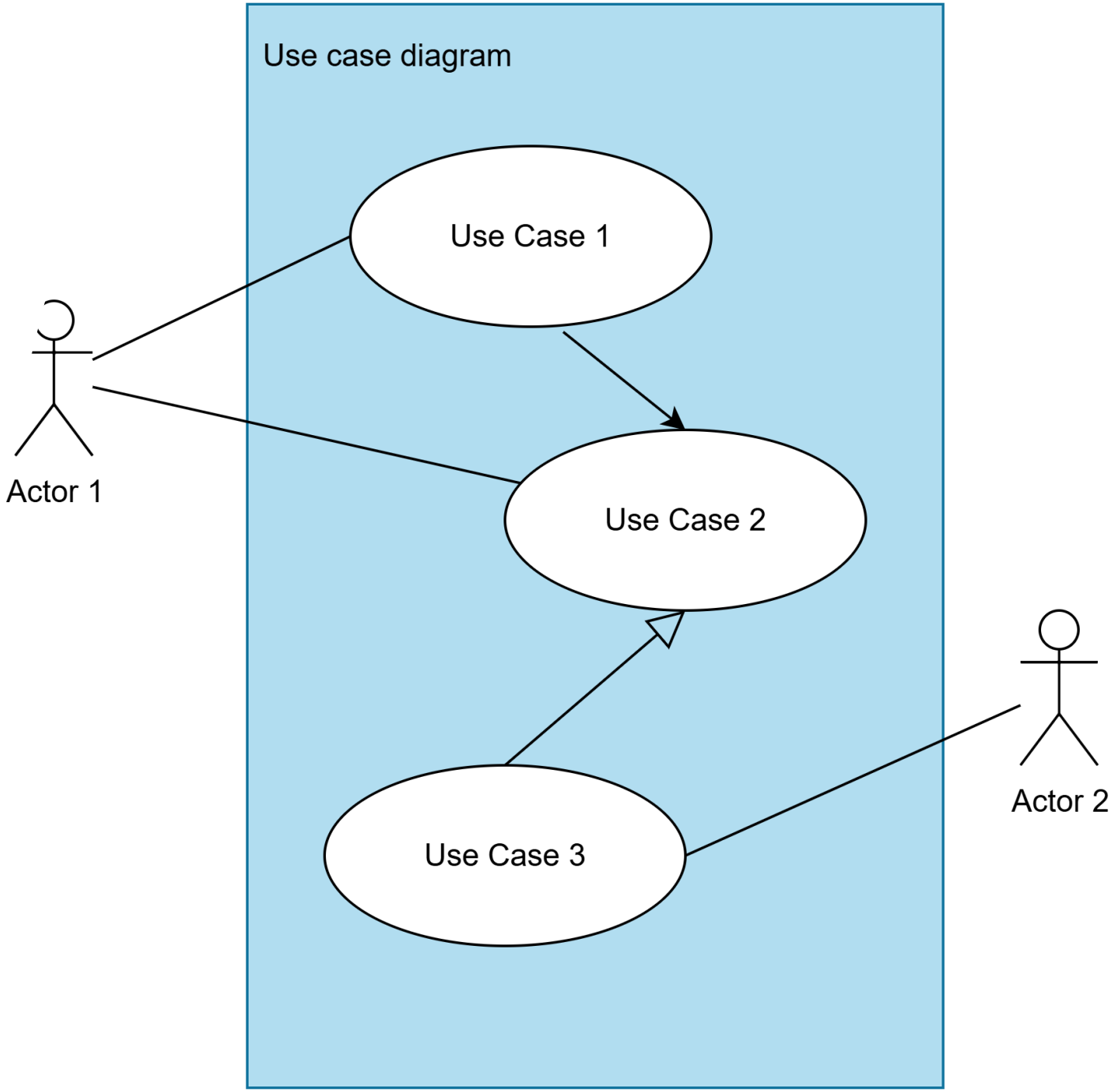
Use Case Diagram

Section 3

Use Case Diagram (Notation Recap)

A use case diagram represents:

- **Actors** – external roles interacting with the system
- **Use Cases** – functions or services the system provides
- **Interactions** – communication between actors and use cases



Case Study – ATM System

This example concerns a simplified **(ATM) system**. The ATM provides the following services:

1. Cash Withdrawal for any credit card holder, via a card reader and cash dispenser.

2. Checking account balance, cash deposits, and check deposits for customers holding a credit card from the bank associated with the ATM.

Case Study – ATM System

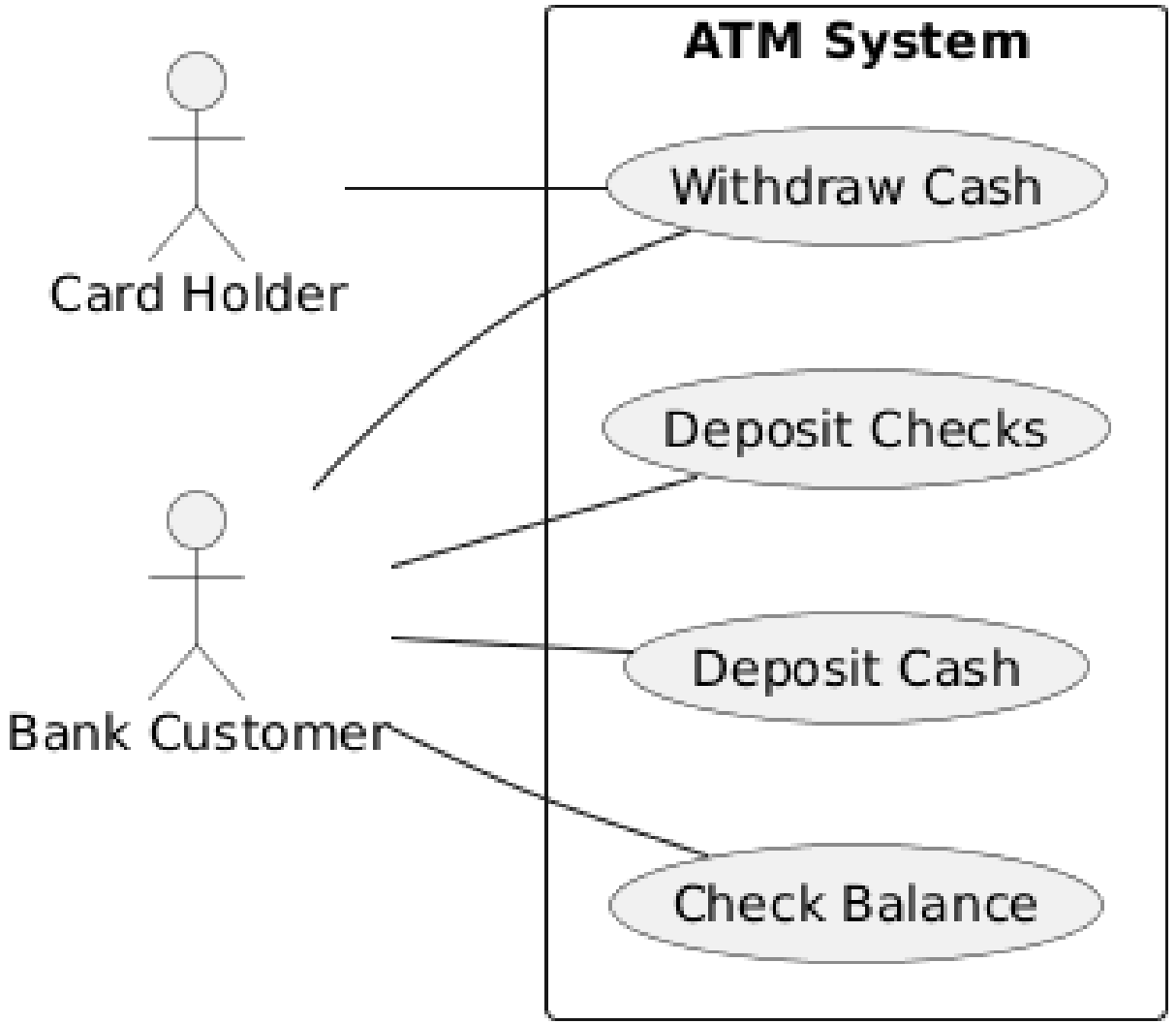
Card Holder:

- Withdraw cash

Bank Customer:

- Withdraw cash (don’t forget this!)
- Check their account balance
- Deposit cash
- Deposit checks

Case Study – ATM System



Generalization in Use Case Diagrams

- Actors can share many common use cases
- An actor may differ from another by just a few additional use cases
- Generalization helps reduce clutter in use case diagrams
- It simplifies both the presentation and the semantics of use cases

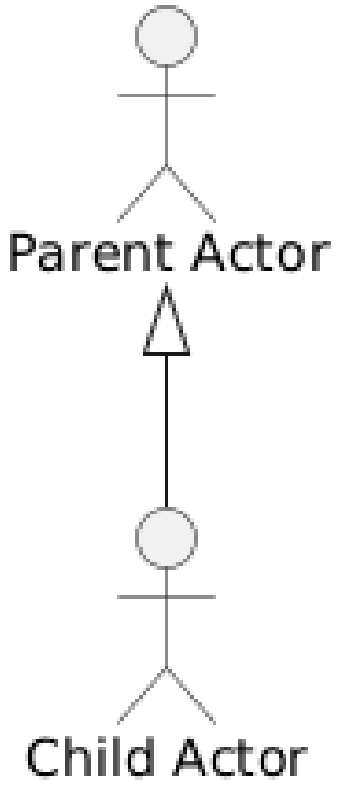
Relations Between Actors

- Only one type of relation exists between actors in UML: **Generalization**.

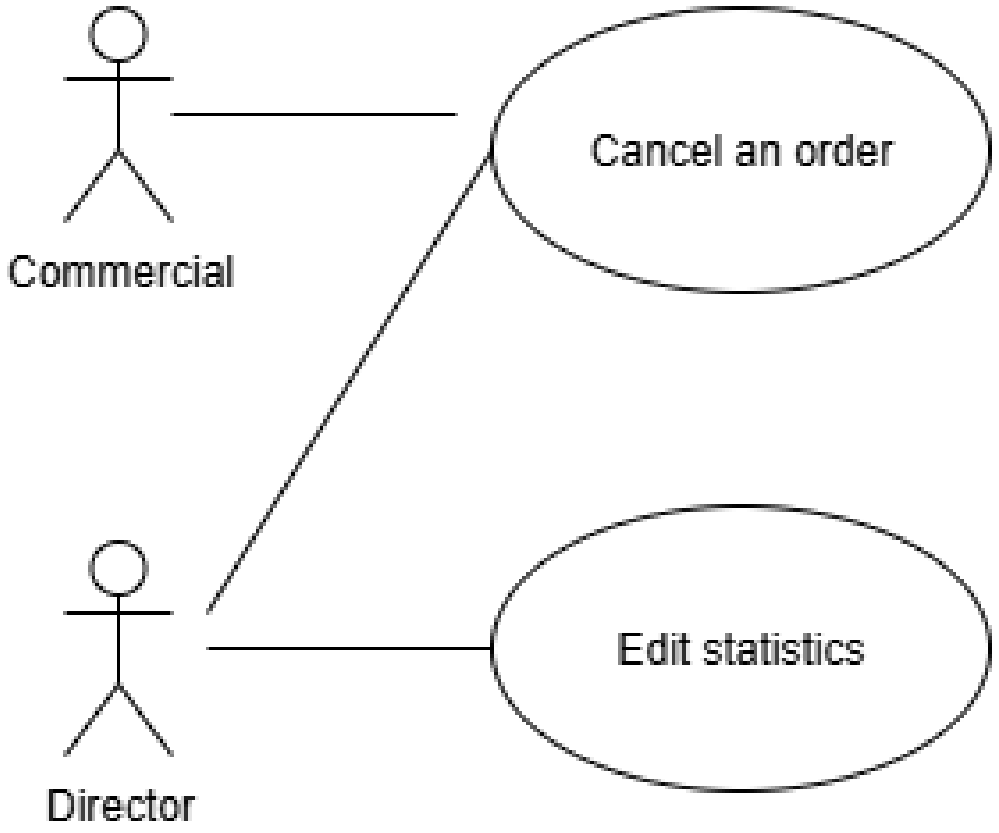
- A **child actor inherits the roles, permissions, and interactions** of a **parent actor**.

- The child actor can also have its **own specific interactions** in addition to the inherited ones.

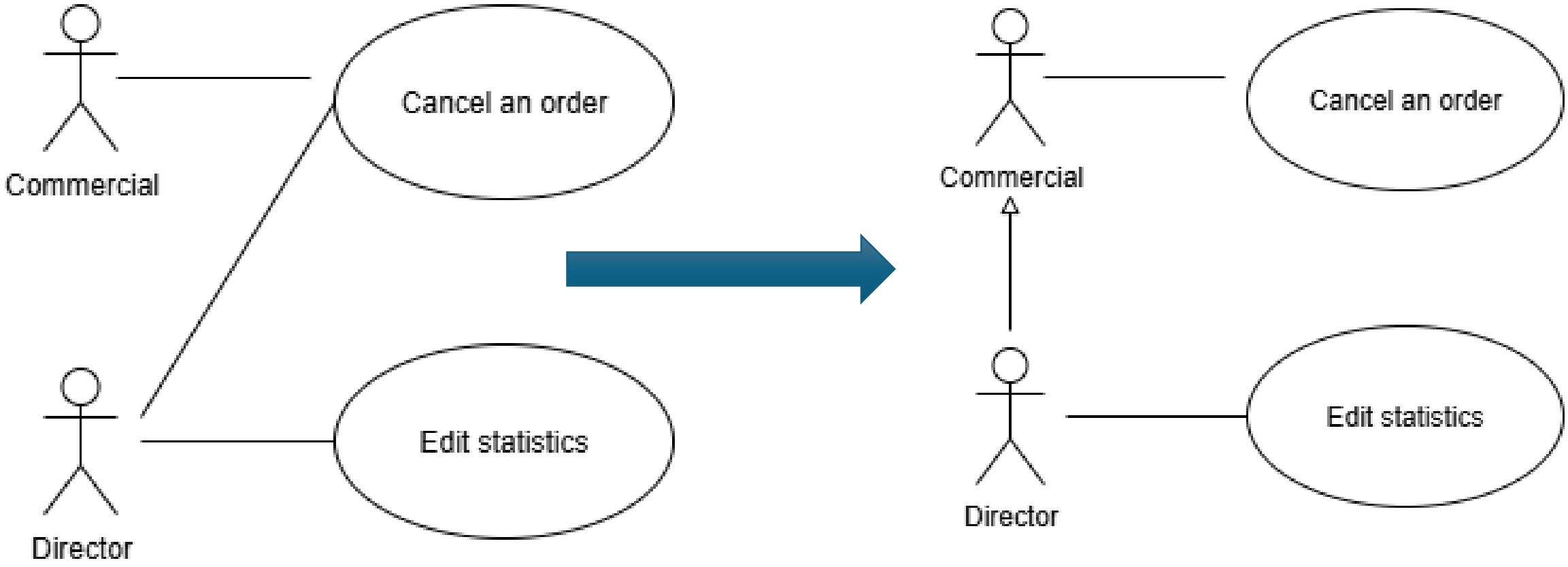
- **Generalization** is represented by a **solid line with a hollow triangle arrowhead** pointing from the **specialized element** (child) to the **general element** (parent).



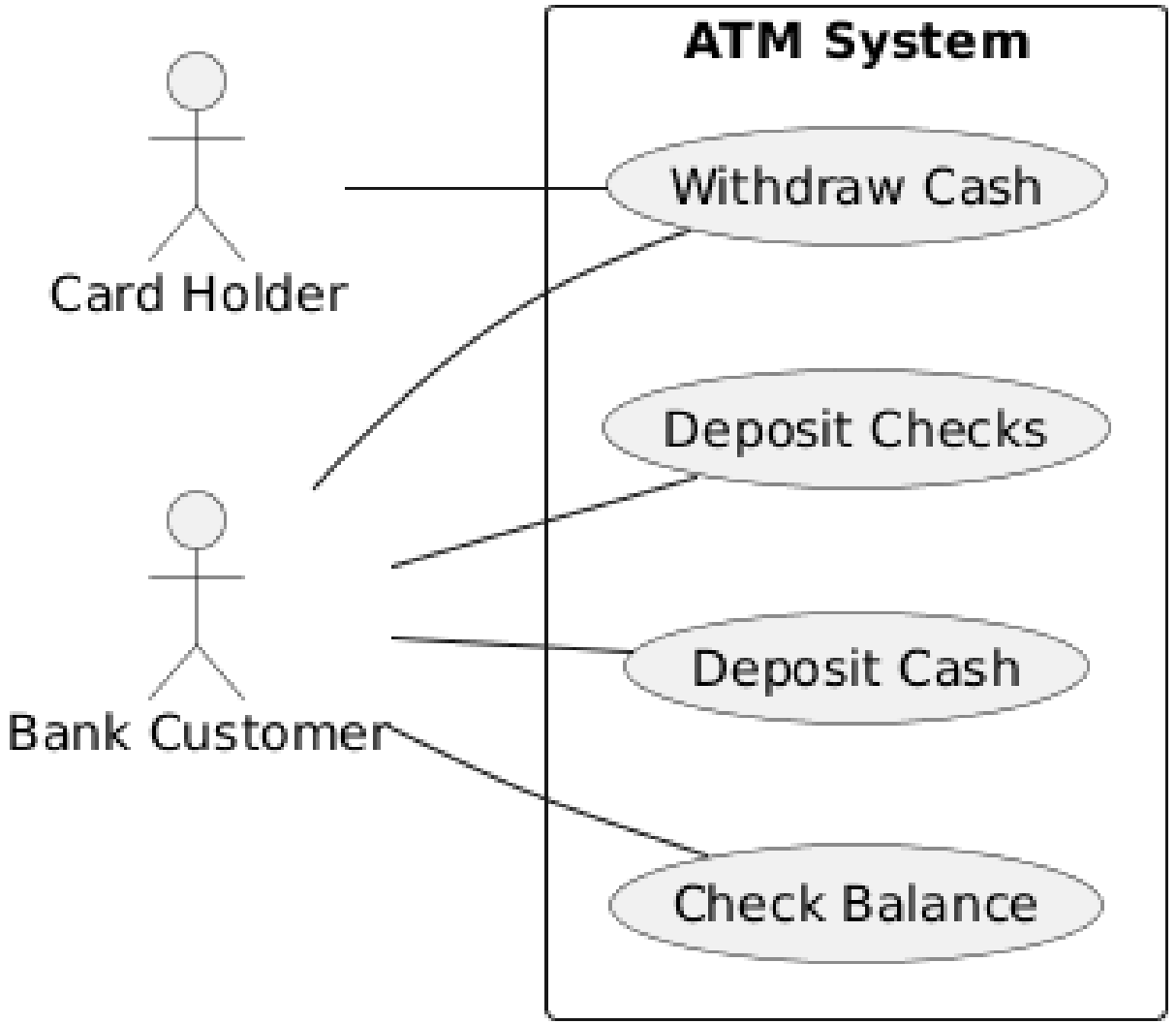
Generalization in Use Case Diagrams



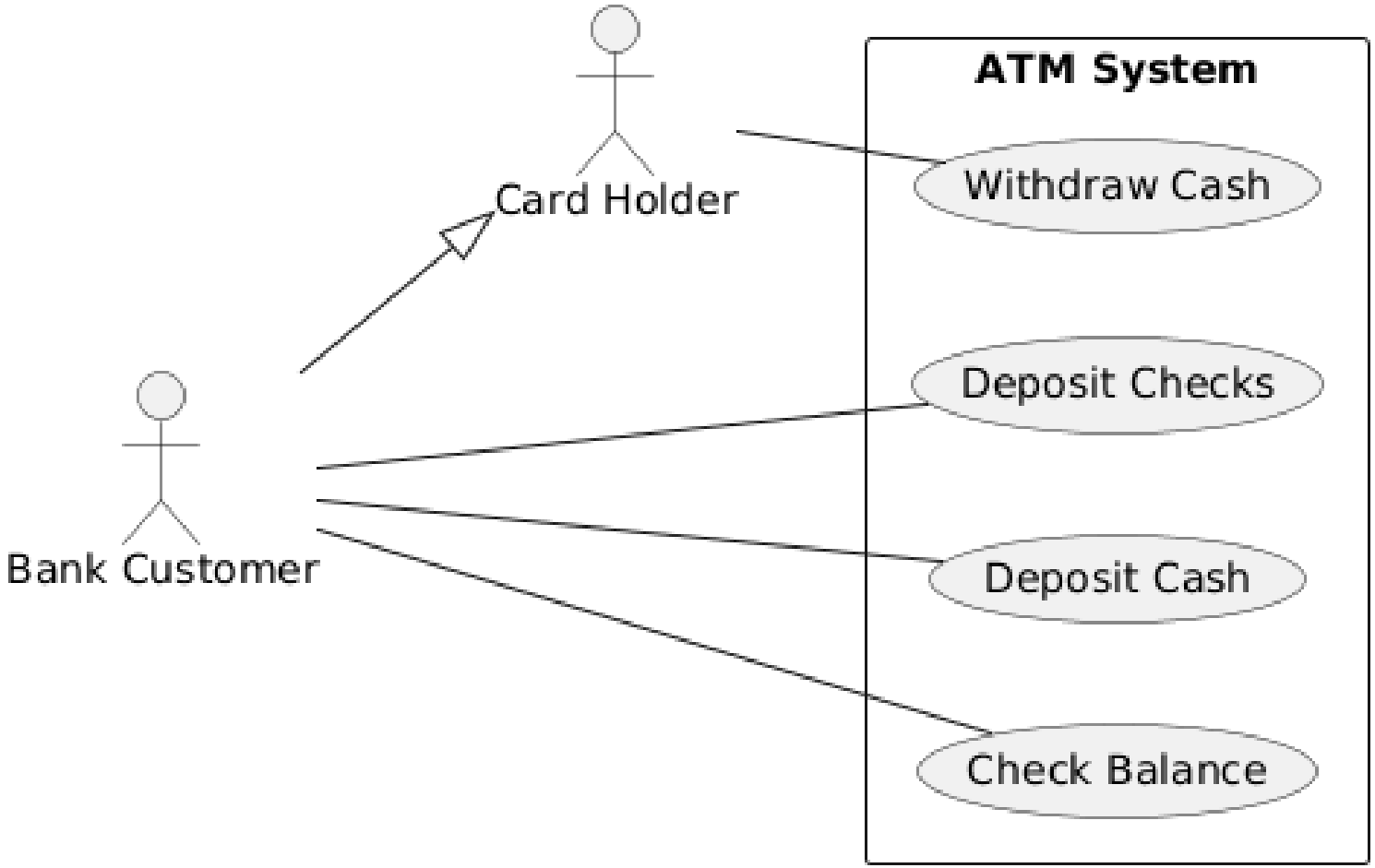
Generalization in Use Case Diagrams



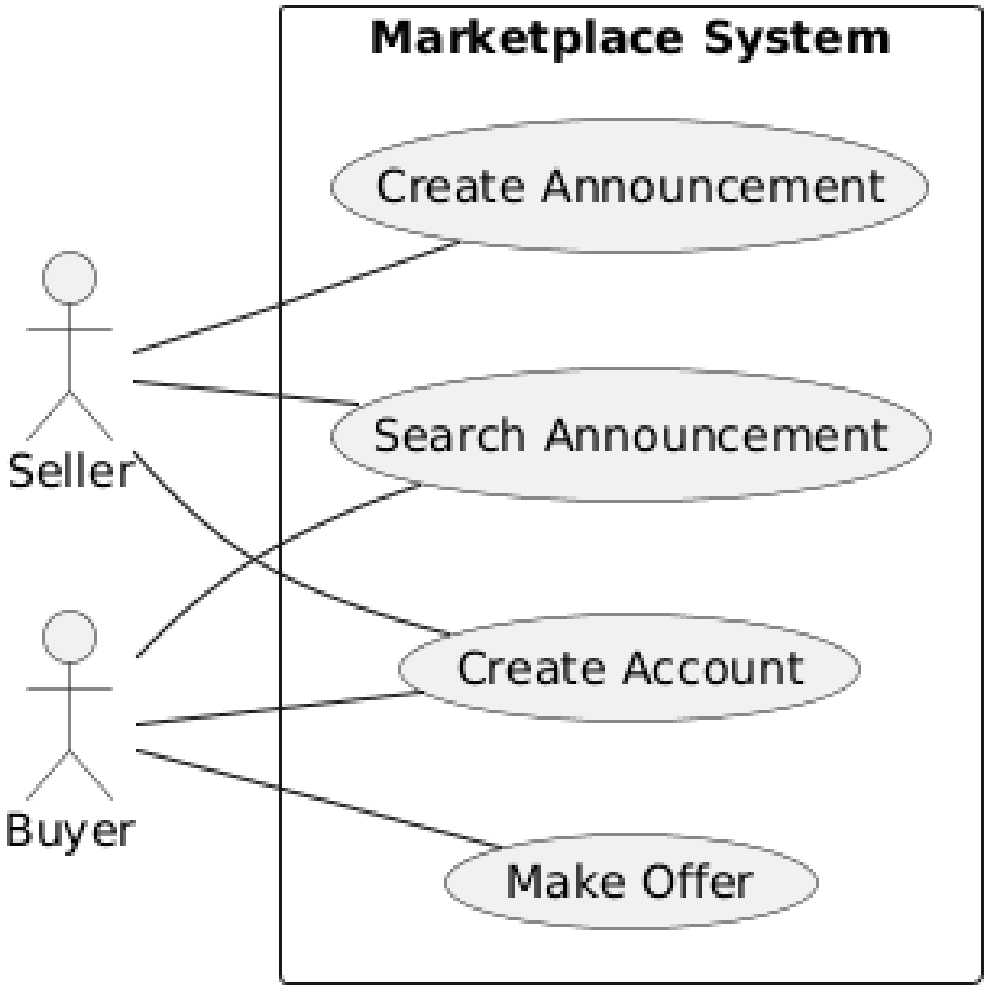
Case Study – ATM System



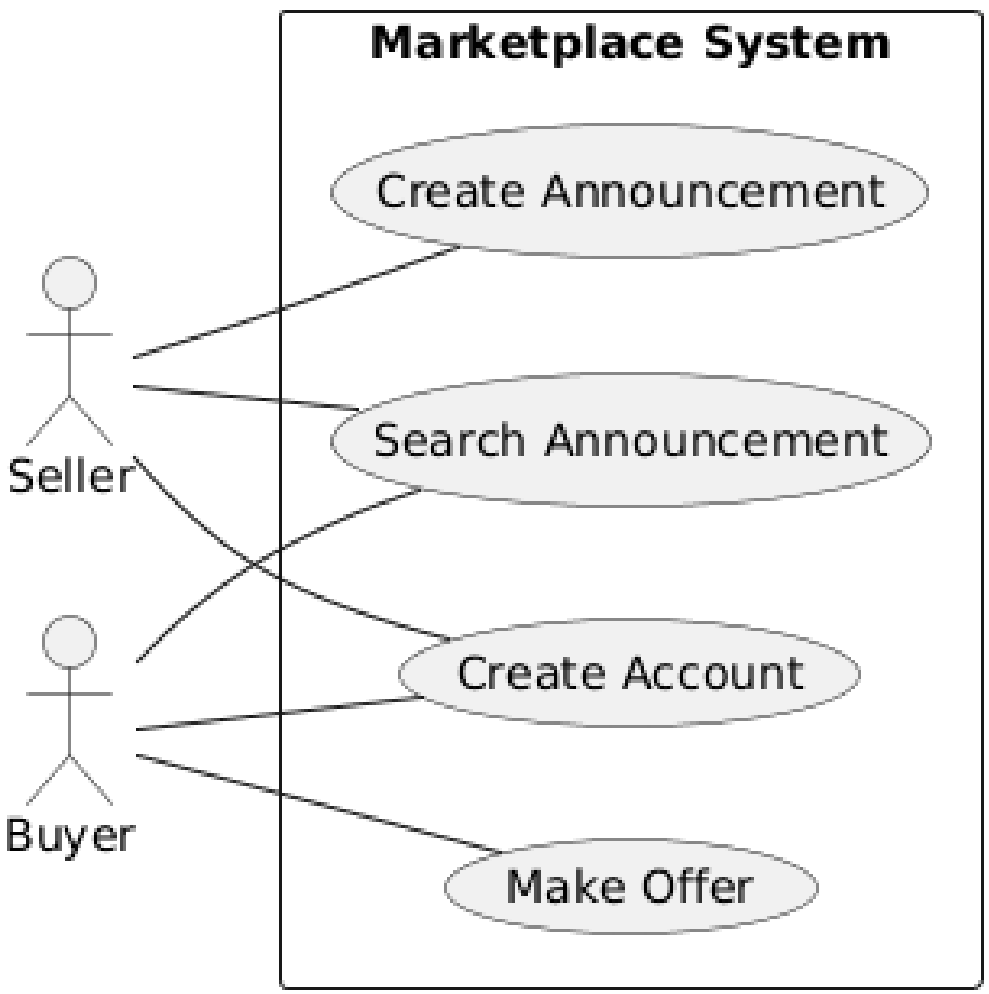
Case Study – ATM System

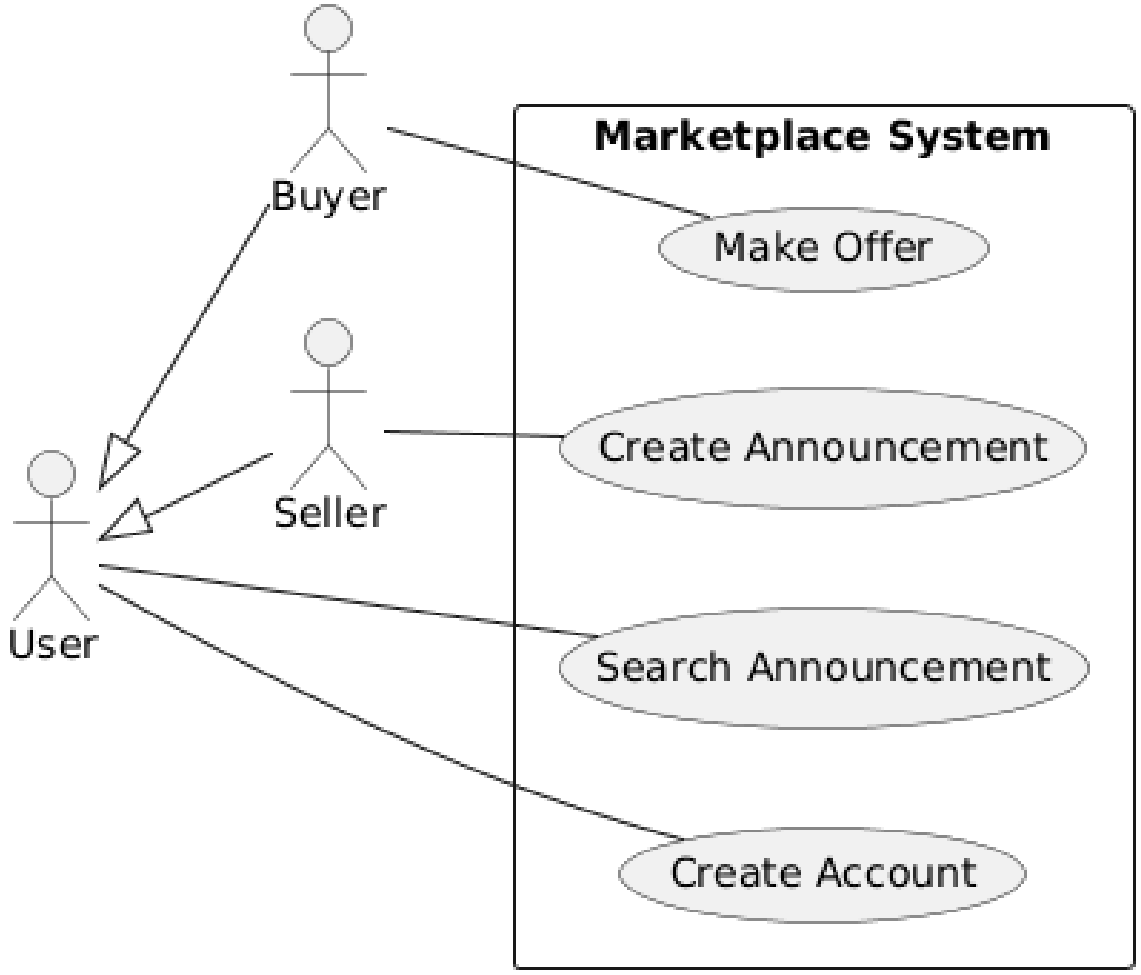


Exercise


Marketplace System
Create Announcement
Search Announcement
Create Account
Make Offer
Seller
Buyer

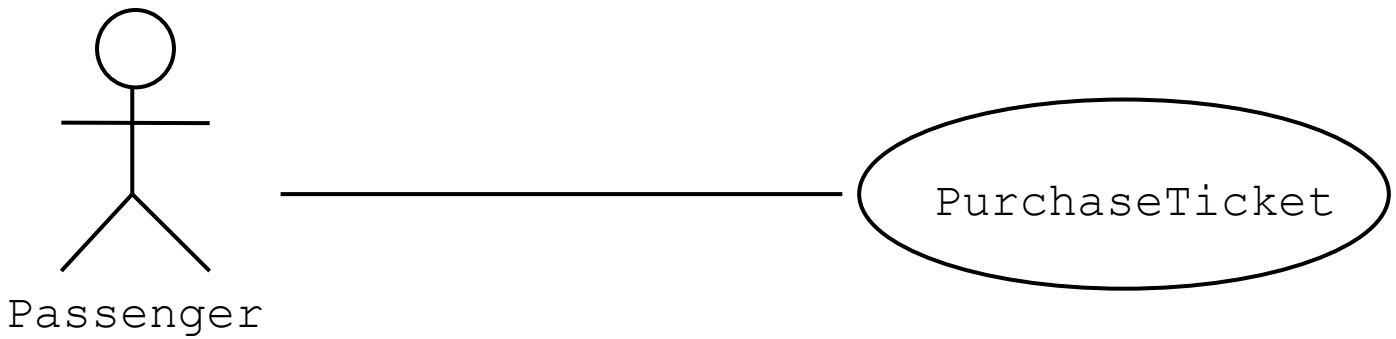
Exercise


Marketplace System
Create Announcement
Search Announcement
Create Account
Make Offer
Seller
Buyer


Buyer
Marketplace System
Make Offer
Seller
Create Announcement
User
Search Announcement
Create Account

Association

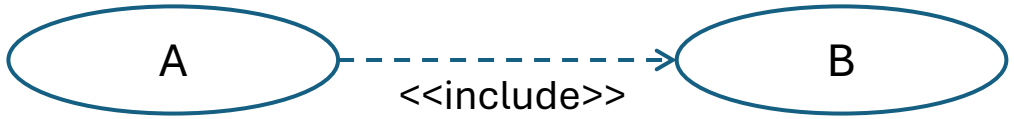
- Actors involved in a use case are **linked by an association**
- An actor can **use the same use case multiple times**
- Represented as a **solid line**
- Shows **who uses what functionality**



Relationship

Relationship is an association between use case and actor.
There are several use case relationships:

- **Inclusion («include»):** Use case A includes use case B (B is a mandatory part of A)

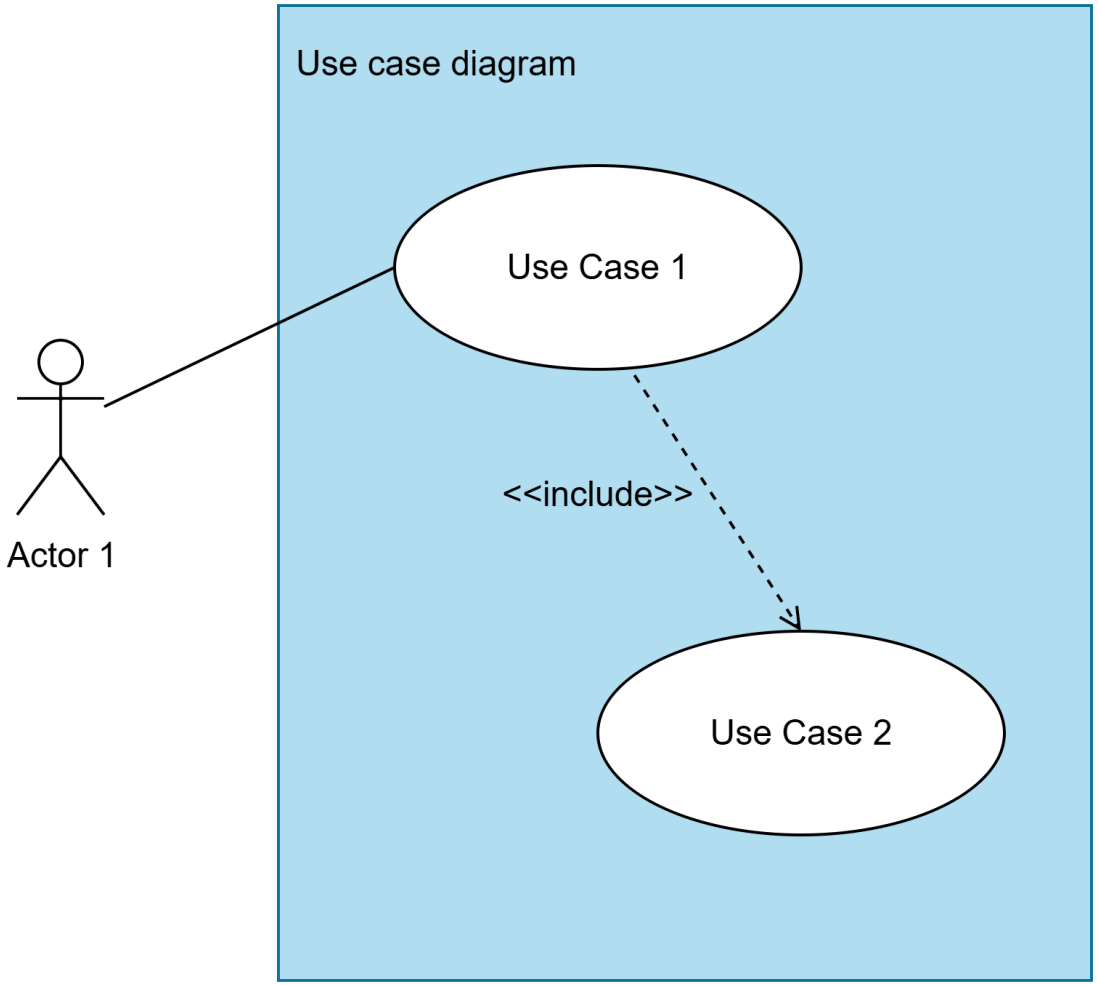


- Extension («extend»): Use case B extends use case A (B adds optional or conditional behavior to A)

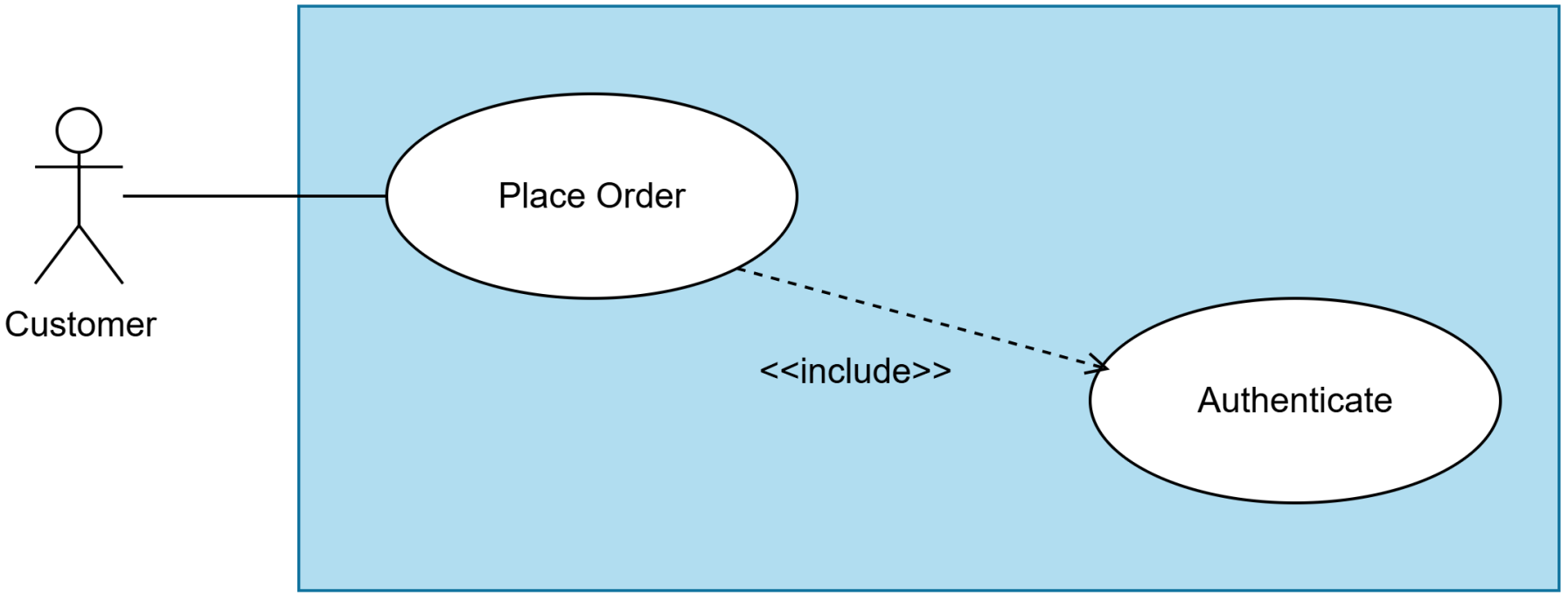
- Generalization: Use case A is a generalization of use case B (B is a specialized version of A)

Include Relationship

- The «include» relationship shows that one use case **always** uses another use case.
- It represents **mandatory** behavior.
- Represented by a dashed arrow with the «include» label.



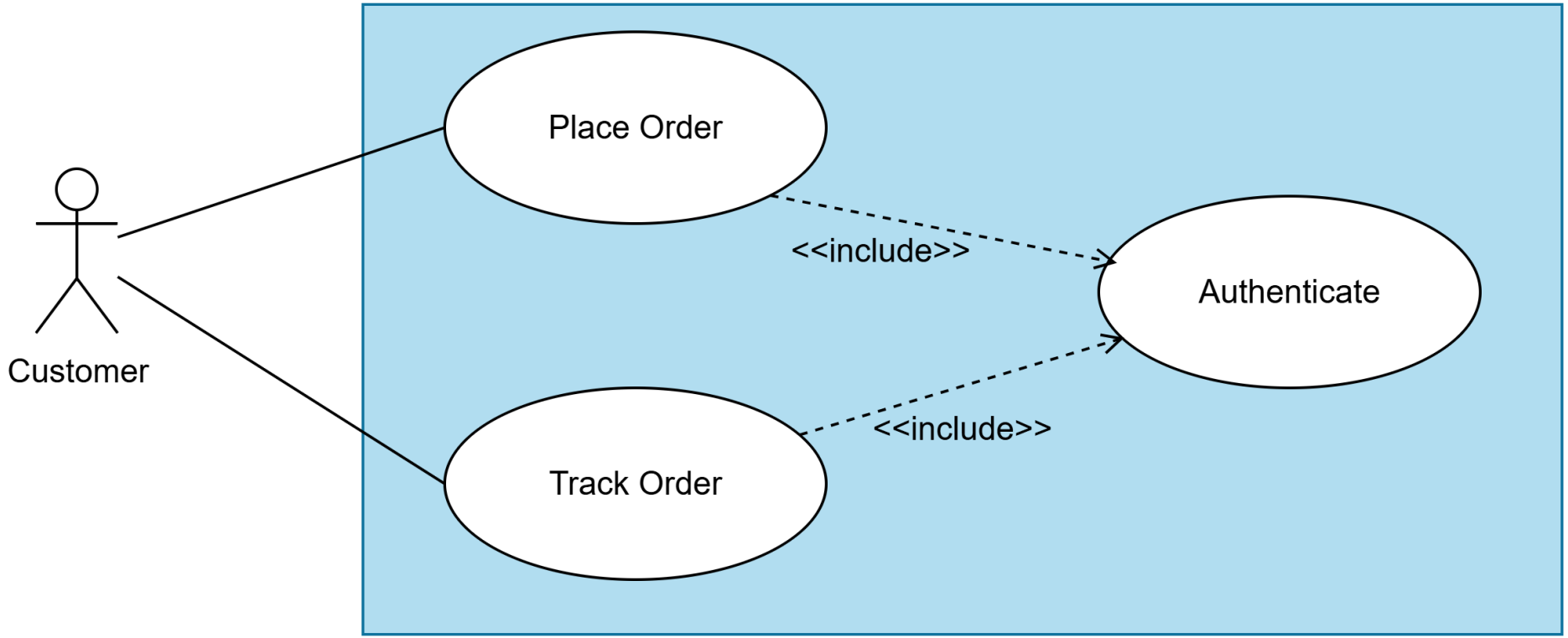
Include Relationship



Placing an order always requires authentication!

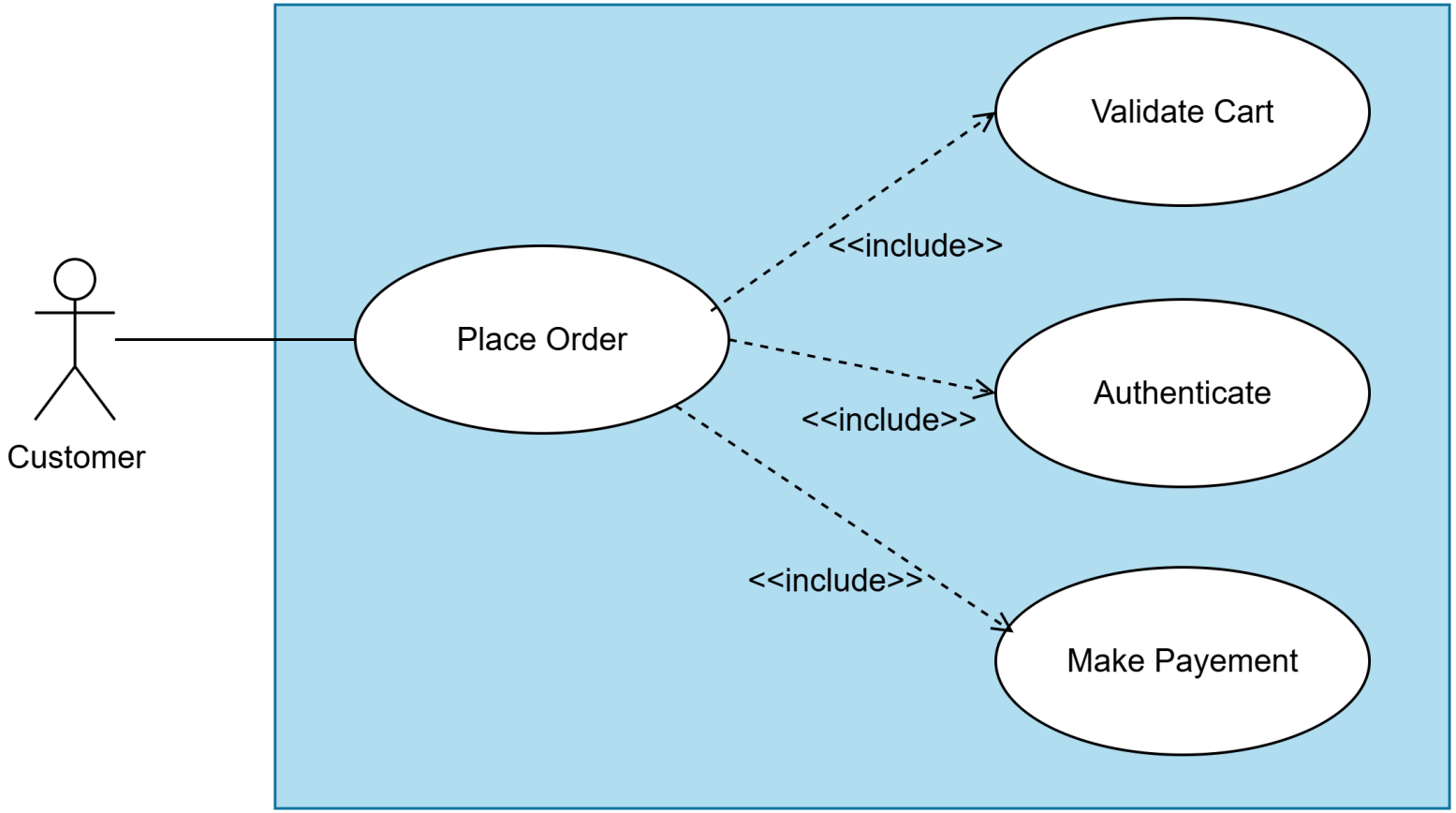
Reusability with Includes and Extends

- Relationships between use cases allow reuse of functionality. For example, the Authenticate use case means it is unnecessary to develop the authentication module multiple times.



Decomposition Using Includes and Extends

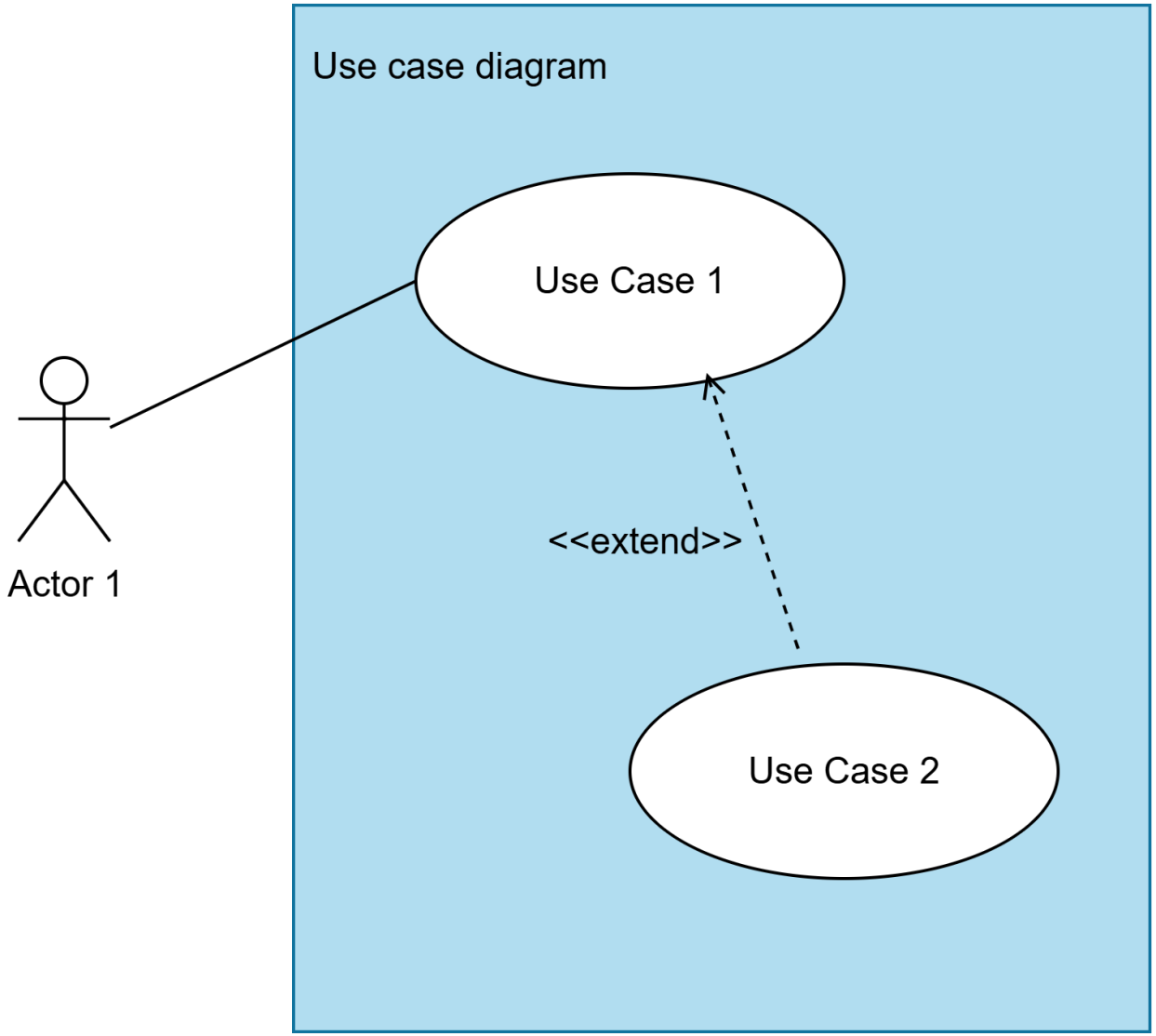
- When a use case is too complex, involving too many actions, it can be **decomposed into simpler use cases**.



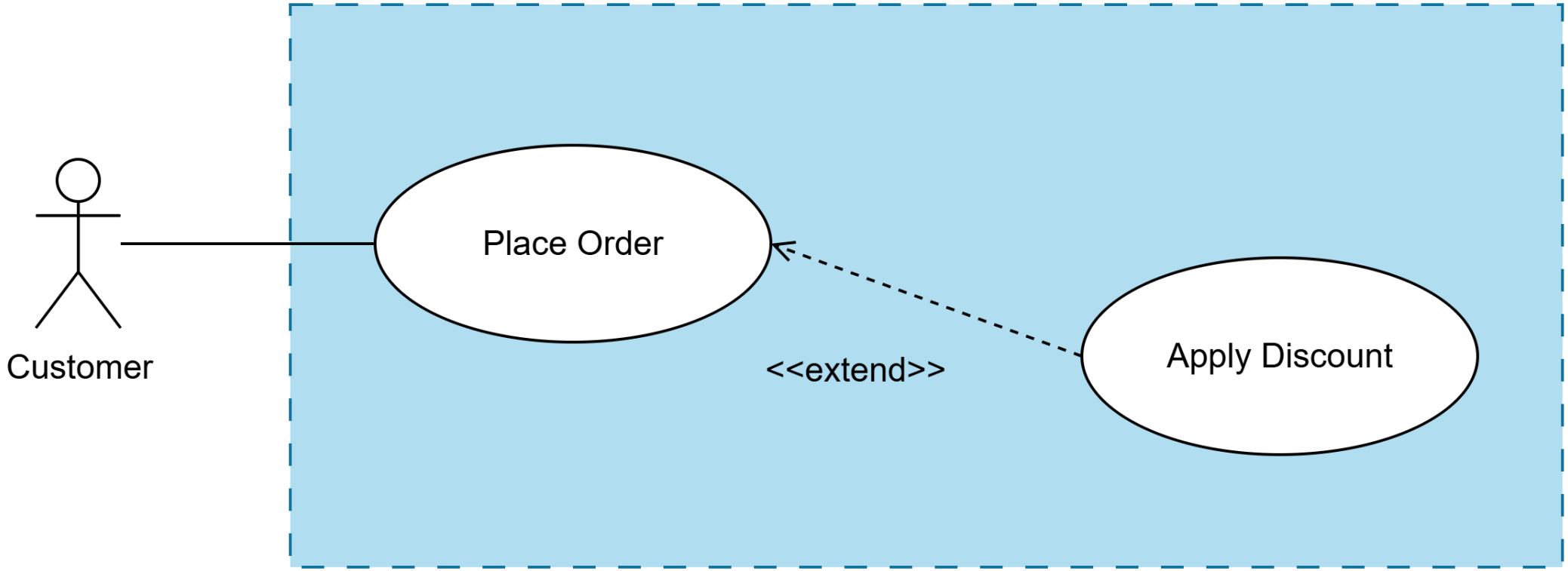
Extend Relationship

- The «extend» relationship Indicates that a use case can **optionally** or **under certain conditions** use the sequence of activities described in another use case.

- Represented by a dashed arrow labeled «extend» from the optional use case to the base use case.



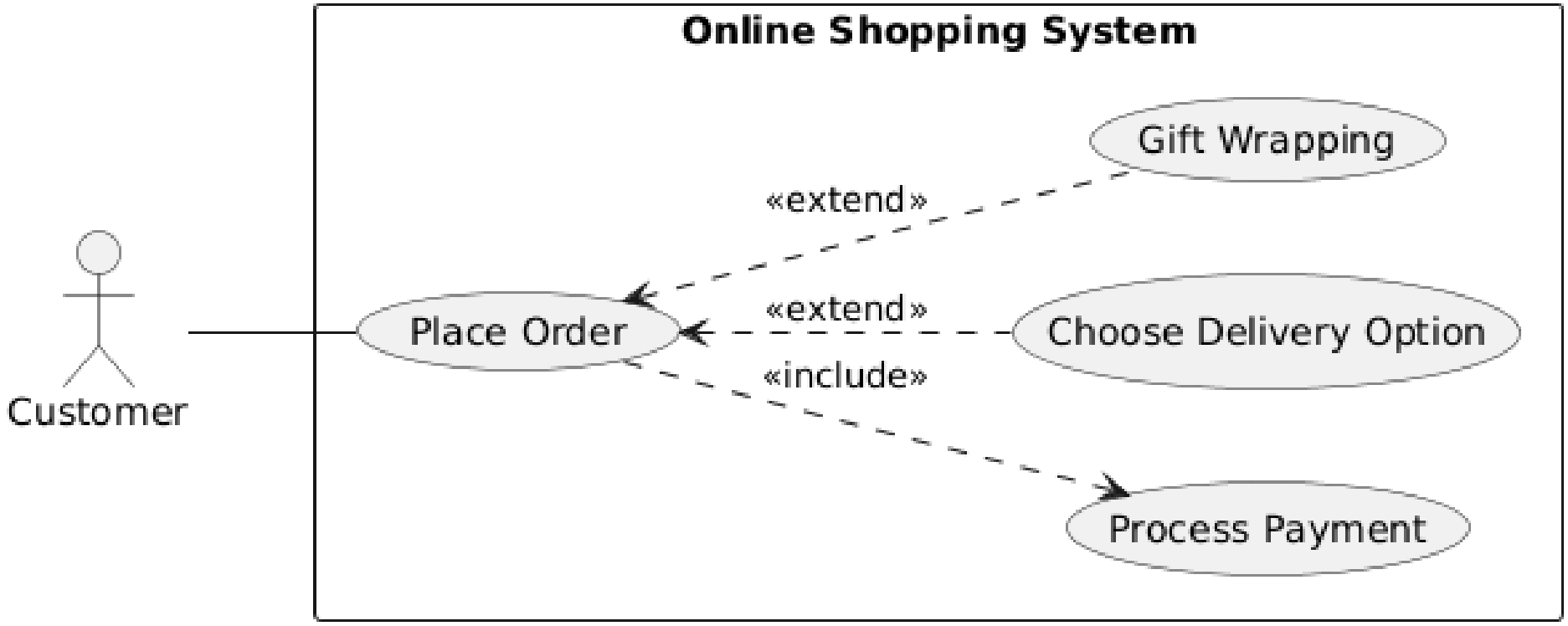
Extend Relationship



Exercise – Online Shopping System

1. A customer can place an order.
2. Every order requires payment processing.
3. Optionally, the customer can choose a delivery option, or request gift wrapping

Exercise – Online Shopping System



Case Study – ATM System

This example concerns a simplified **Automated Teller Machine (ATM) system**. The ATM provides the following services:

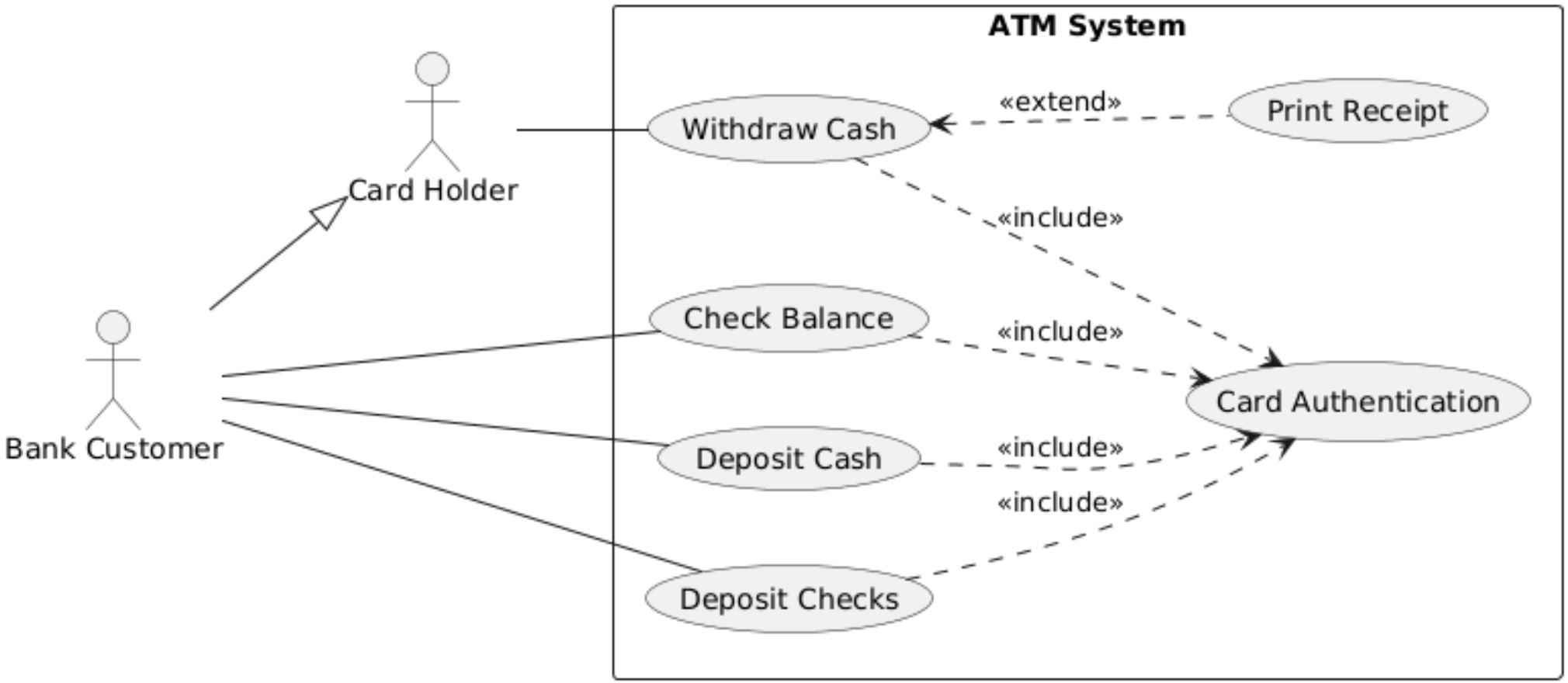
1. Cash Withdrawal for any credit card holder, via a card reader and cash dispenser.

2. Checking account balance, cash deposits, and check deposits for customers holding a credit card from the bank associated with the ATM.

3. Before performing any transaction, the client must authenticate their card. This step is common to all services.

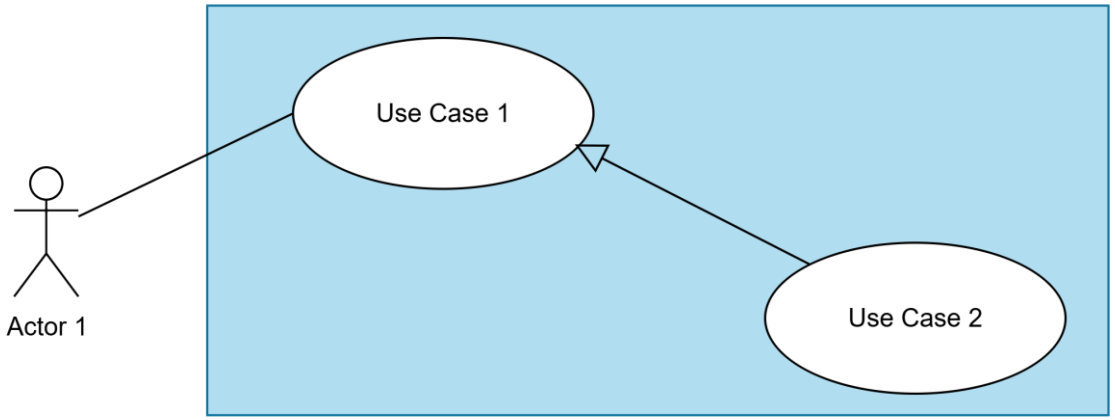
4. After performing a cash withdrawal, the client may choose to print a receipt.

Case Study – ATM System

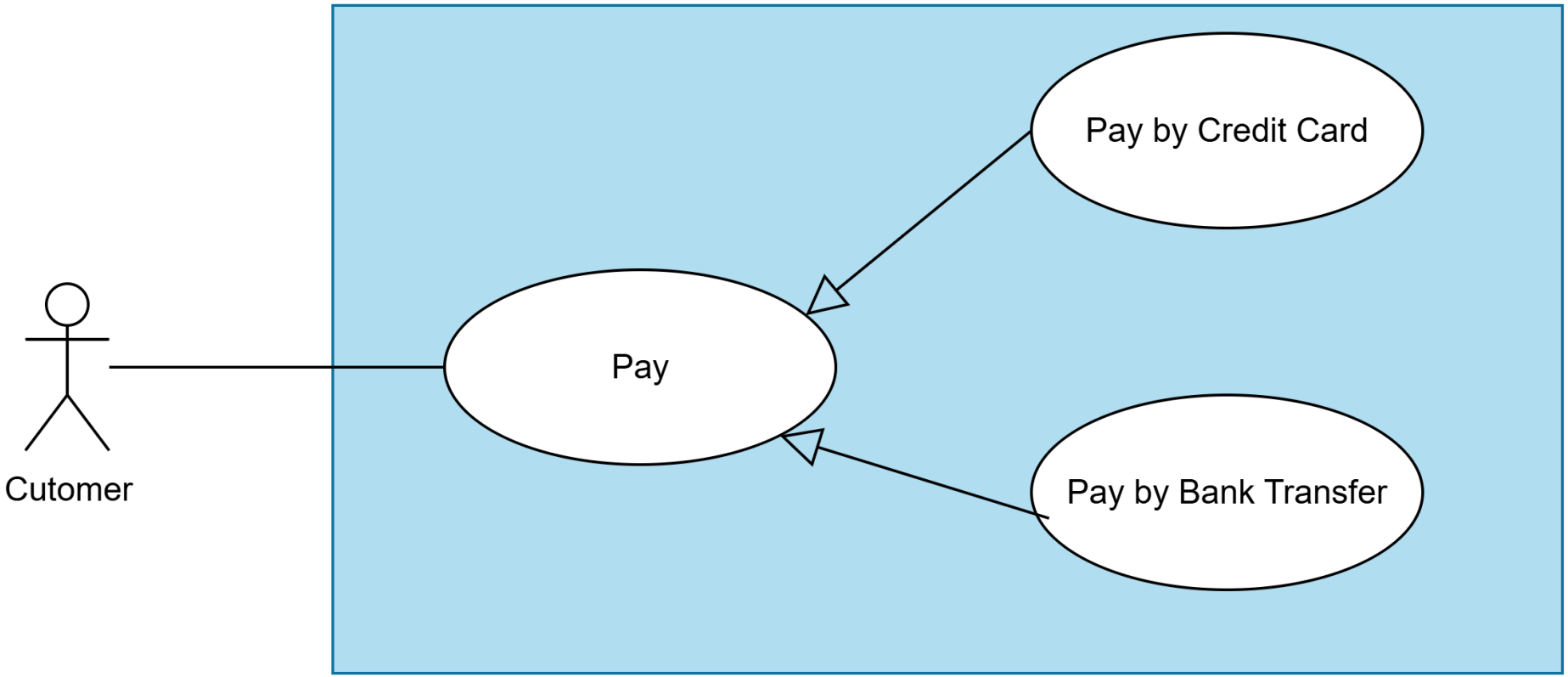


Generalization Relationship

- Generalization shows that one use case inherits the behavior of another.
- The child (specialized) use case can reuse or extend the behavior of the parent (general) use case.
- Represented in UML by a solid line with a hollow triangle pointing toward the parent.
- It is present in most UML diagrams and corresponds to the inheritance concept in object-oriented languages.



Generalization Relationship



The base use case Pay defines general payment behavior, while Pay by Credit Card and Pay by Bank Transfer inherit this behavior and add their specific steps

Case Study – ATM System

This example concerns a simplified **Automated Teller Machine (ATM) system**. The ATM provides the following services:

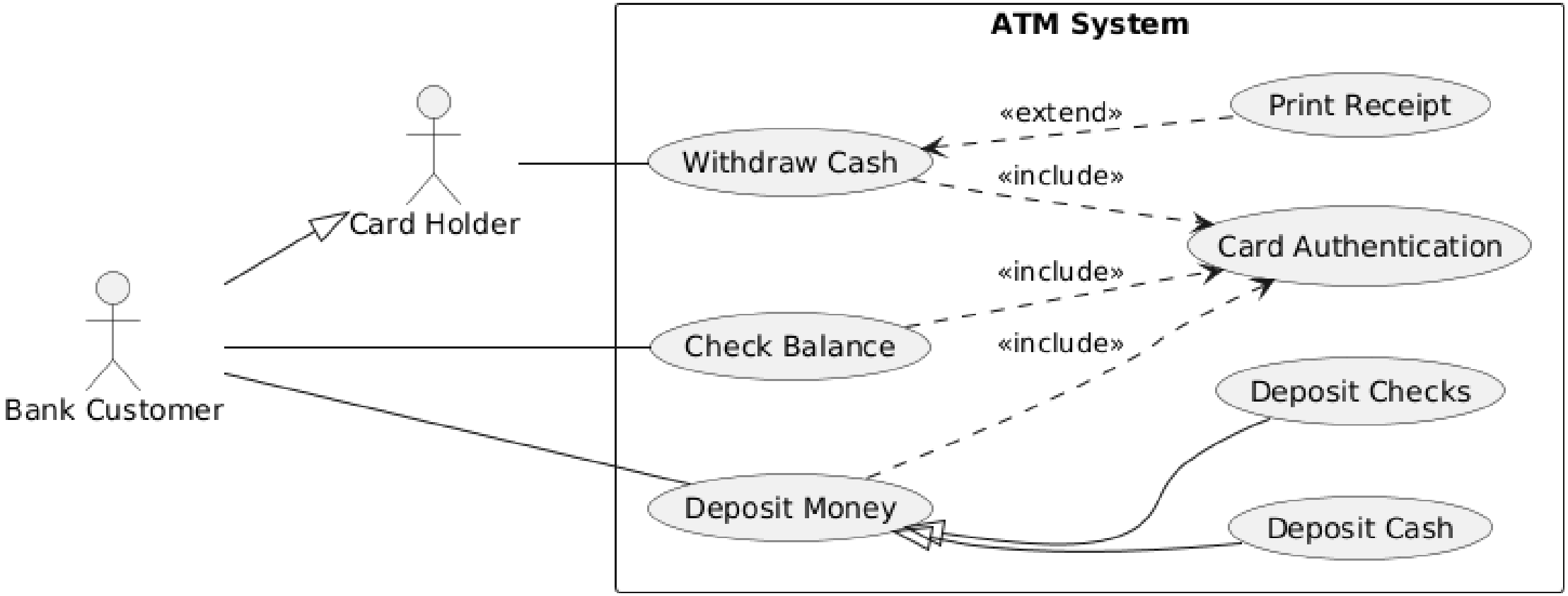
1. Cash Withdrawal for any credit card holder, via a card reader and cash dispenser.

2. Checking account balance, cash deposits, and check deposits for customers holding a credit card from the bank associated with the ATM.

3. Before performing any transaction, the client must authenticate their card. This step is common to all services.

4. After performing a cash withdrawal, the client may choose to print a receipt.

Case Study – ATM System



Use Case Description Attributes

Attribute	Description
Use Case Name	- Describes a behavior (use verbs) - Name should be short, meaningful, and **unique**
ID	Numeric identifier, **unique** for each use case
Brief Description	One paragraph summarizing the use case
Primary Actors	Actors who **initiate** the use case
Secondary Actors	Actors who **interact** with the use case
Preconditions	- System state **before** execution - Must be true before starting the use case
Postconditions	- System state **after** execution - Must be true once the use case is completed
Basic Path / Main Flow	- Typical sequence of steps when everything goes as expected during the execution of the use case
Alternative Paths / Exceptions	- What happens if something goes wrong or optional branches occur

Main Flow

- **Main flow** describes the **typical, successful sequence of actions**.
- Steps are **numbered**, starting with the **trigger**:
  - The use case starts when <actor> <performs action>
- Each step describes **what happens next**, either by the actor or the system.
- Optional steps may be indicated, but exceptions go in **alternative paths**.

Example: User Login Use Case

1. **The use case starts when** the client clicks the “Login” button.
2. **The client** enters their username in the “login” field.
3. **The client** enters their password in the “password” field.
4. **The system** verifies the validity of the username and password.

Avoid Vague Formulations

Bad Example: “Client information is entered and verified”

Problems:

- Who enters the information?
- Where is it entered?
- What specific information is entered?
- Written in **passive voice** → unclear who performs the action

ATM-Use Case Description

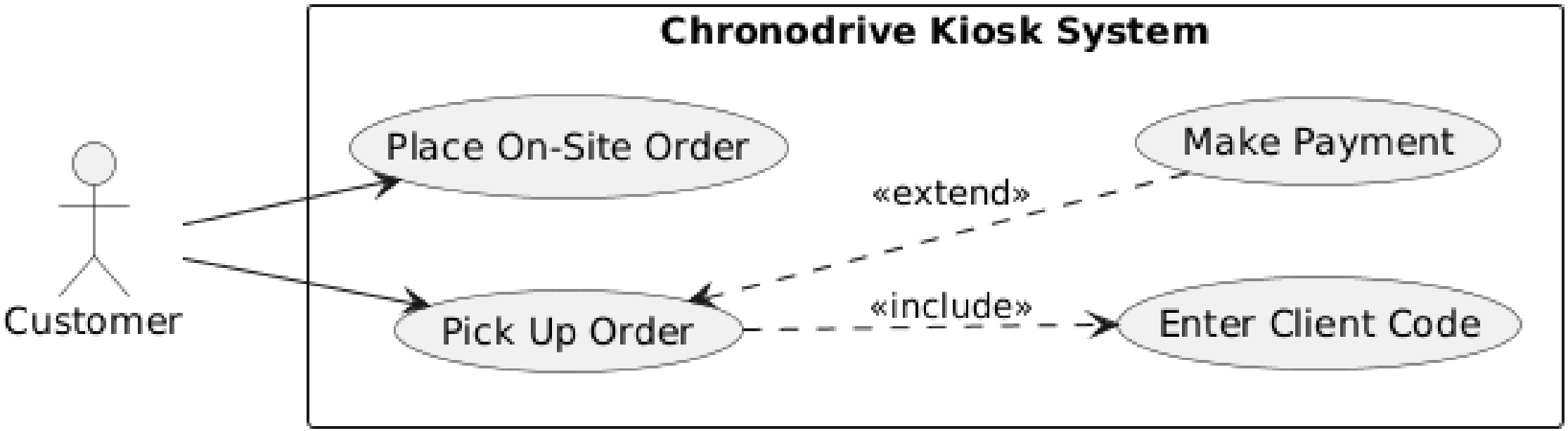
Attribute	Description
Use Case Name	Withdraw Cash
ID	UC1
Brief Description	Allows a card holder to withdraw cash from their account using an ATM. The system authenticates the card, checks the balance, and dispenses the requested amount if sufficient funds are available.
Primary Actor	Card Holder
Secondary Actor	_
Preconditions	- Card holder has a valid ATM card - Card holder **must be authenticated** (via included `Card Authentication`) - Account has sufficient funds - ATM is operational
Postconditions	- Cash is dispensed to the card holder - Account balance is updated - Transaction recorded in bank system
Basic Path / Main Flow	**1. The use case starts when** the Card Holder inserts their ATM card 2. **The system** authenticates the card (`Card Authentication` included). **3. The Card Holder** enters the withdrawal amount. 4. **The system** checks the account balance 5. **The system** dispenses the requested cash. **6. The system** optionally prints a receipt (`Print Receipt` extends). 7. **The transaction ends**.
Alternative Paths / Exceptions	- **Authentication fails:** System displays error, transaction canceled, card returned. - **Insufficient funds:** System displays error, transaction canceled, card returned. - **User cancels transaction:** Card returned, no cash dispensed. - **ATM out of cash:** Transaction canceled, card returned.

Exercise 1

Chronodrive is a commercial website that allows customers to shop online and then pick up their goods at the store via the drive. On the automatic kiosk screen in the store, there are two options: Place an on-site order and Pick up order. To pick up an order, the customer must enter their client code and make payment if it has not already been completed online.

Model the functionalities offered by the automatic kiosk using a **use case diagram**.

Exercise 1


Chronodrive Kiosk System
Place On-Site Order
Make Payment
«extend»
Customer
Pick Up Order
«include»
Enter Client Code

Exercise 3

- An Agent at a travel agency helps a customer plan and organize a trip.
- Planning a trip involves multiple steps:
  - Reserving accommodation at a hotel or other lodging.
  - Arranging transportation for the trip, which could include train, flight, or bus tickets.
  - Booking local transportation such as taxis or rental cars.
- During the booking of transport tickets, the type of ticket depends on the customer's preference:
  - Train Ticket for rail travel
  - Flight Ticket for air travel
  - Bus Ticket for short-distance trips
- Optionally, the agent can generate a detailed receipt if the customer requests one.
- Additionally, if the customer chooses, the agent can add travel insurance for the trip