

Chapter 5

Object Oriented Analysis and Design

Contents

- 5. OBJECT ORIENTED ANALYSIS AND DESIGN (OOAD) (5hrs)**
- a. Object Oriented Development Life Cycle
 - b. Difference between Object Oriented Development Life Cycle and Traditional SDLC
 - c. Unified Modeling Language (UML)
 - d. Use Case Modeling
 - e. Object Modeling: Class Diagrams
 - f. Dynamic Modeling: State Diagrams, Sequence Diagrams
 - g. Analysis vs Design

Introduction

- Object-oriented analysis and design (OOAD) is a technical approach used in the analysis and design of an application or system through the application of the object-oriented paradigm and concepts including visual modeling.
- Focuses on creating a model of objects from the real world and then to use this model to develop object oriented system.

- OOAD techniques are used to:
 - Study the existing objects to see if they can be reused or adopted to new user.
 - Define both object modeling and process modeling.
 - Develop complex system using the important properties of object modeling like inheritance ,encapsulation and polymorphism.

Object Oriented Development Life Cycle

- Comprises of **THREE** main phases:
 1. Analysis
 2. Design
 3. Implementation

Implementation

Programming database

System Design

System Architectural Design

Analysis

- **Application**
- **what**

Object Design

- data structure**
- algorithm**
- control**

1. Analysis:

- Starts with defining a problem statement.
- Requirements are collected ,analyzed and organized collecting from the customer.
- Based on requirements, a feasibility study is carried out to know whether the project is do-able or not.
- Different analysis techniques are used during this phase ,the mostly used THREE techniques are:
 - I. Object modeling
 - II. Dynamic modeling
 - III. Functional modeling

i. Object modeling

- Develops the static structure of the system in terms of objects.
- Identify the objects and group into classes.
- Identify the relationship among the classes.
- Define the user object attributes.
- Define the operations that should be performed on the classes .
- Review the glossary.

ii. Dynamic modeling

- After the static behaviors of the system is analyzed, its behavior with respect to time and external changes need to be examined.
- A way of describing how an individual objects responds to event.
- Identify the events and analyze the applicability of actions.
- Comprises of state transition diagrams.

iii. Functional modeling

- Final component of the object oriented analysis.
- Shows the processes that are performed within an object and the data changes as it moves between the methods.
- Corresponds to the data flow diagrams of the structure analysis.
- Identify all inputs and outputs of the system.
- Identify the constraints.
- Specify the optimization criteria.

2.Design

- The design phase of this development life cycle can be done into TWO subheadings:
 - System design
 - Object Design

System Design

- A model of the real world situation showing its important properties and domain is build in this phase.
- Architectural design is represented at high level of abstraction(overall system view) and subsequent refinements lead to more details system view.
- Divide and conquer approach is used where system is decomposed into number f sub-systems , modules ,components and units.
- System design can be done under following methods:
 - Data Design
 - Architectural Design
 - Component level design
 - Interface design

Object Design

- In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase.
- All the classes required are identified.
- The designer decides whether –
 - new classes are to be created from scratch,
 - any existing classes can be used in their original form, or
 - new classes should be inherited from the existing classes.
- The associations between the identified classes are established and the hierarchies of classes are identified.
- Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

3.Implementation and Testing

- In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool.
- The databases are created and the specific hardware requirements are ascertained.
- Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

Comparing SDLC and OODLC

S. No	Traditional approach	Object oriented approach
1	The system is viewed as collection of processes.	The system is viewed as collection of objects.
2	Data flow diagrams, ER diagrams, data dictionary and structured charts are used to describe the system.	UML models including use case diagram, class diagram, sequence diagrams, component diagrams etc are used to describe the system.
3	Reusable source code may not be produced.	The aim is to produce reusable source code.
4	Data flow diagrams depicts the processes and attributes	Classes are used to describe attributes and functions that operate on these attributes.
5	It follows top-down approach for modeling the system.	It follows bottom-up approach for modeling the system.
6	Non iterative	Highly iterative

TRADITIONAL APPROACH	OBJECT ORIENTED SYSTEM DEVELOPMENT
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step of process	Focuses on object, classes, modules that can be easily replaced, modified and reused.
Moving from one phase to another phase is complex.	Moving from one phase to another phase is easier.
Increases duration of project	decreases duration of project
Increases complexity	Reduces complexity and redundancy

Unified Modeling Language(UML)

- A language for specifying ,visualizing and constructing the artifacts of the software system as well as for business modeling.
- Set of modeling conventions that is used t describe a software system in terms of objects.
- The notation is useful for graphically representing the object oriented analysis and design.
- UML diagrams can be categorized into **TWO** types:
 - Structural diagrams
 - Behavioral diagrams

Structural diagrams

- Represents static aspect of the system.
- Represents those parts of the diagram which forms the main structure of the system in terms of classes ,objects ,interfaces , components and units.
- Class diagrams ,object diagrams,component diagrams and deployment diagram are the example of structural diagrams.

Behavioral diagrams

- Represents dynamic aspect of the system.
- Captures the behavior of the system with respect to time and external changes .
- Use case diagram ,sequence diagram ,collaboration diagram ,activity diagram and state diagram are the examples of behavioral diagram.

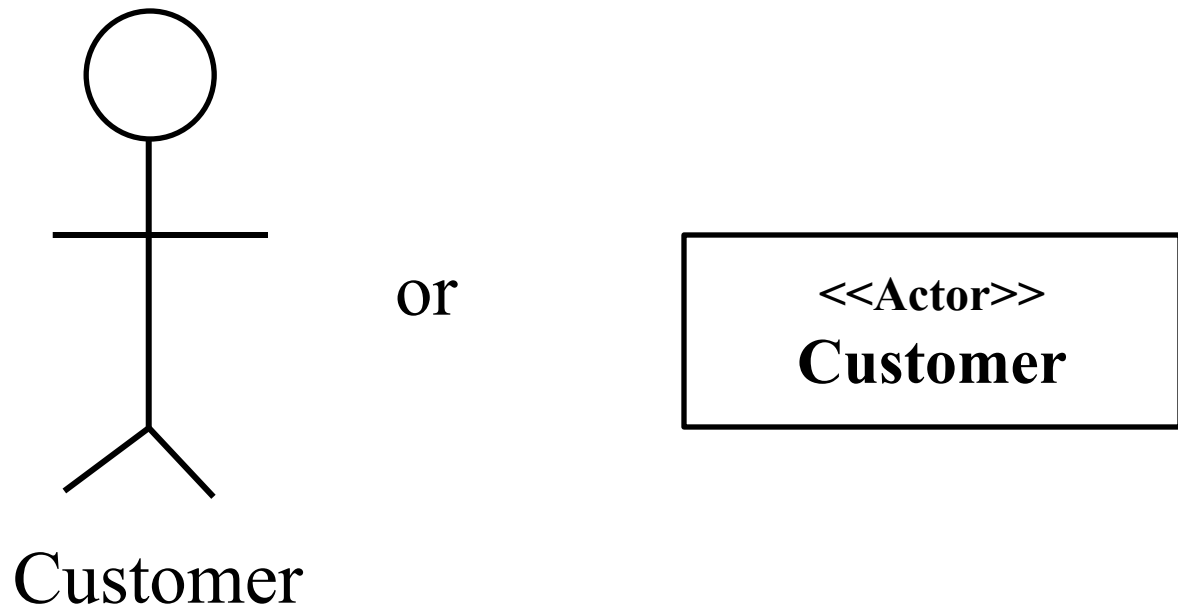
- UML was designed by THREE experts:
 - Grady Boach - Boach method.
 - James Rumbaugh - Object Modeling Technique(OMT)
 - Ivan Jacobson

Use case Diagram

- Summarizes the details of a system's user also known as actors and their interaction with the system.
- focuses on what the system does.
- Captures the requirements of the system ,if the system requirements changes during the life cycle , those changes will be visible in use case model.
- Consisting of **use cases** and **actors**.

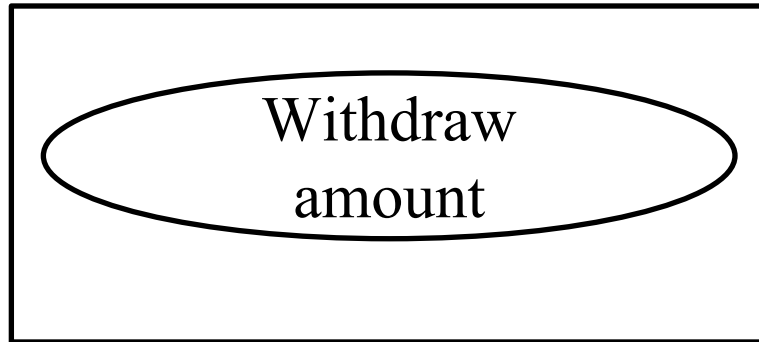
a. Actors

- An external entity that interacts with the system.
- Represents role which may include human users , external hardware or other system.
- Is usually drawn as a named stick figure or alternatively as a class rectangle with <<actor>> keyword.



b. Use cases

- A single unit of meaningful work.
- Provides a high level view of behavior observable someone or something outside the system.
- Can be represented by an ellipse /oval.



- The notation for using a use case is connecting line with an arrowhead showing direction control.

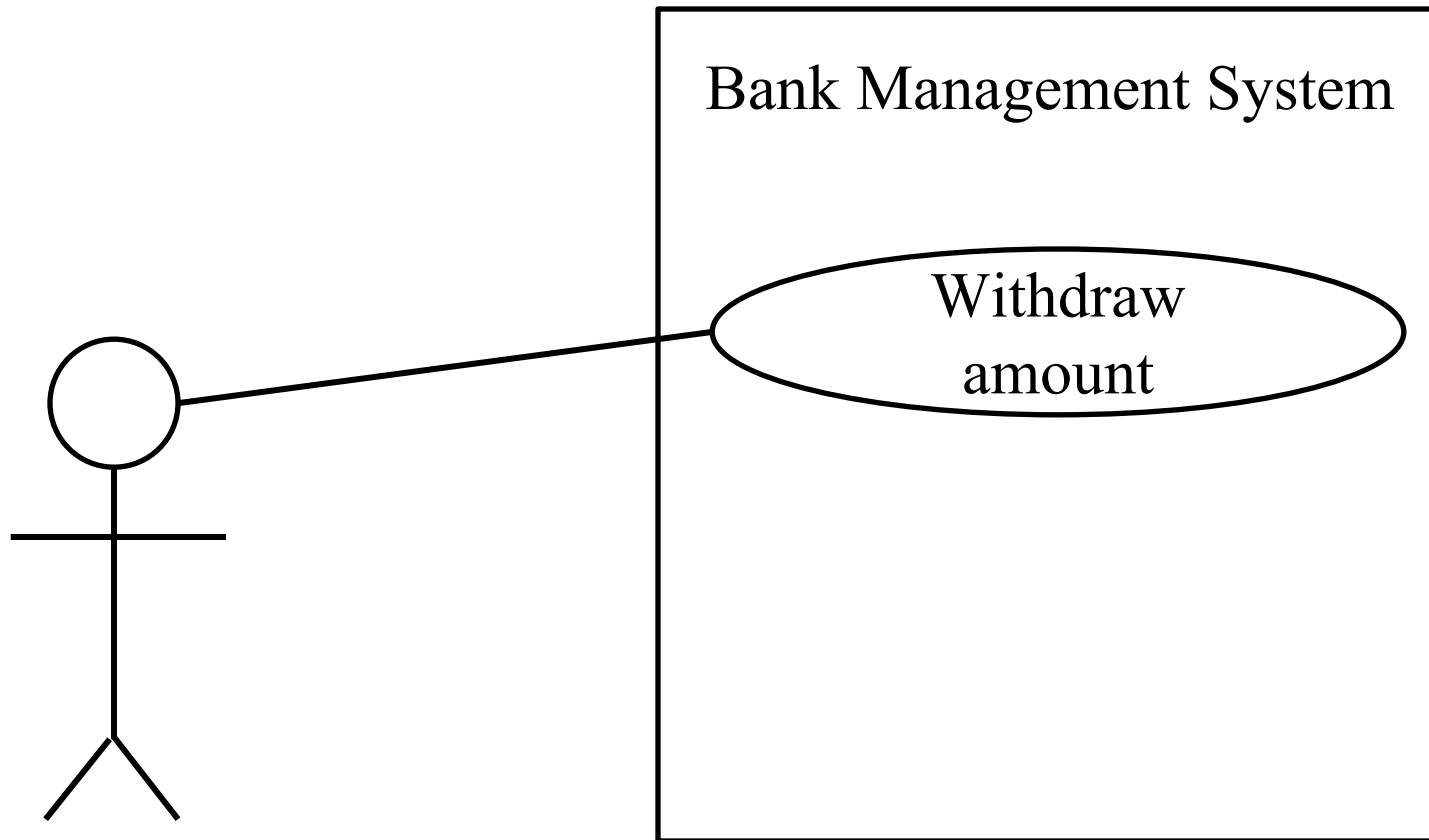
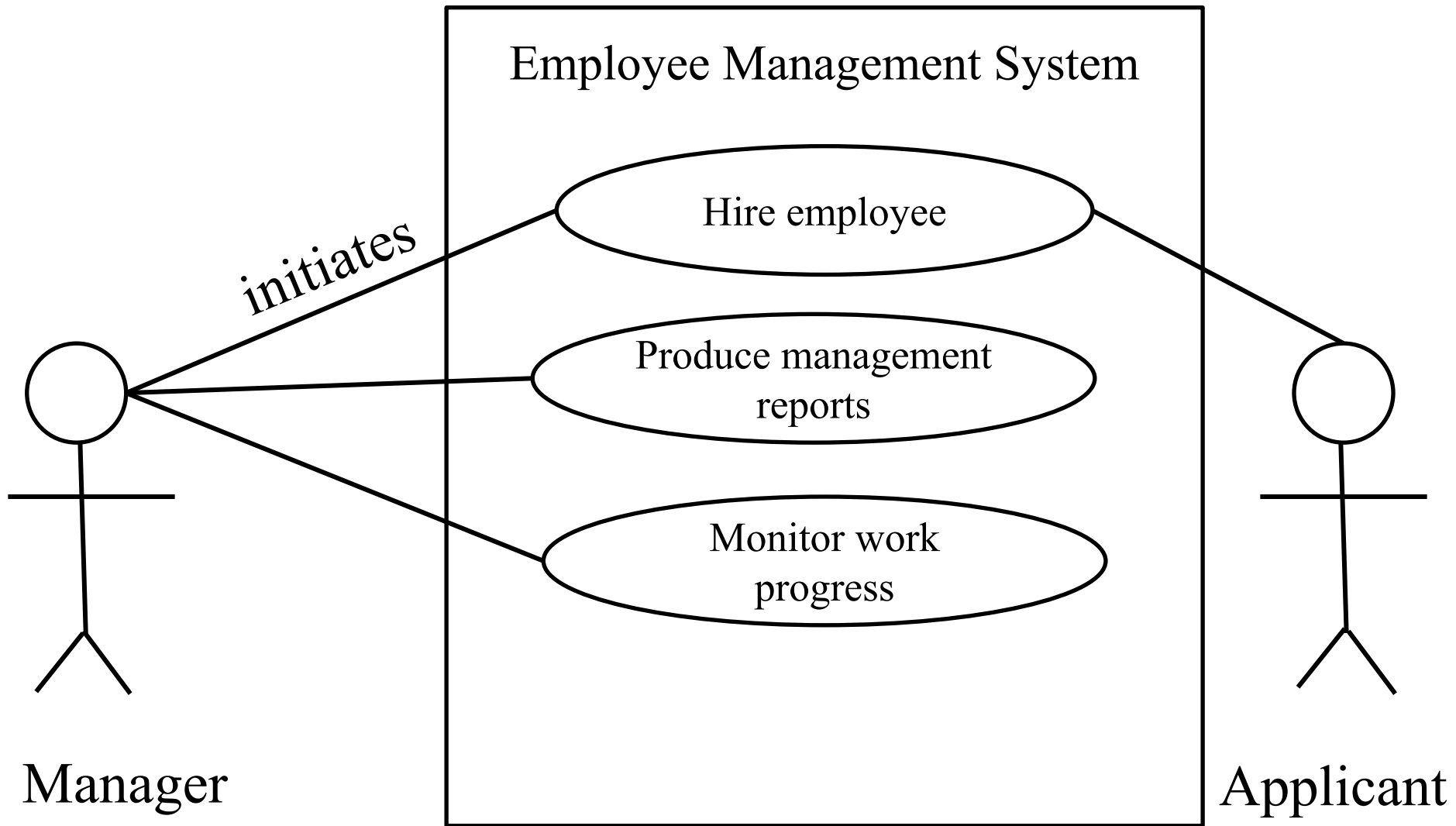
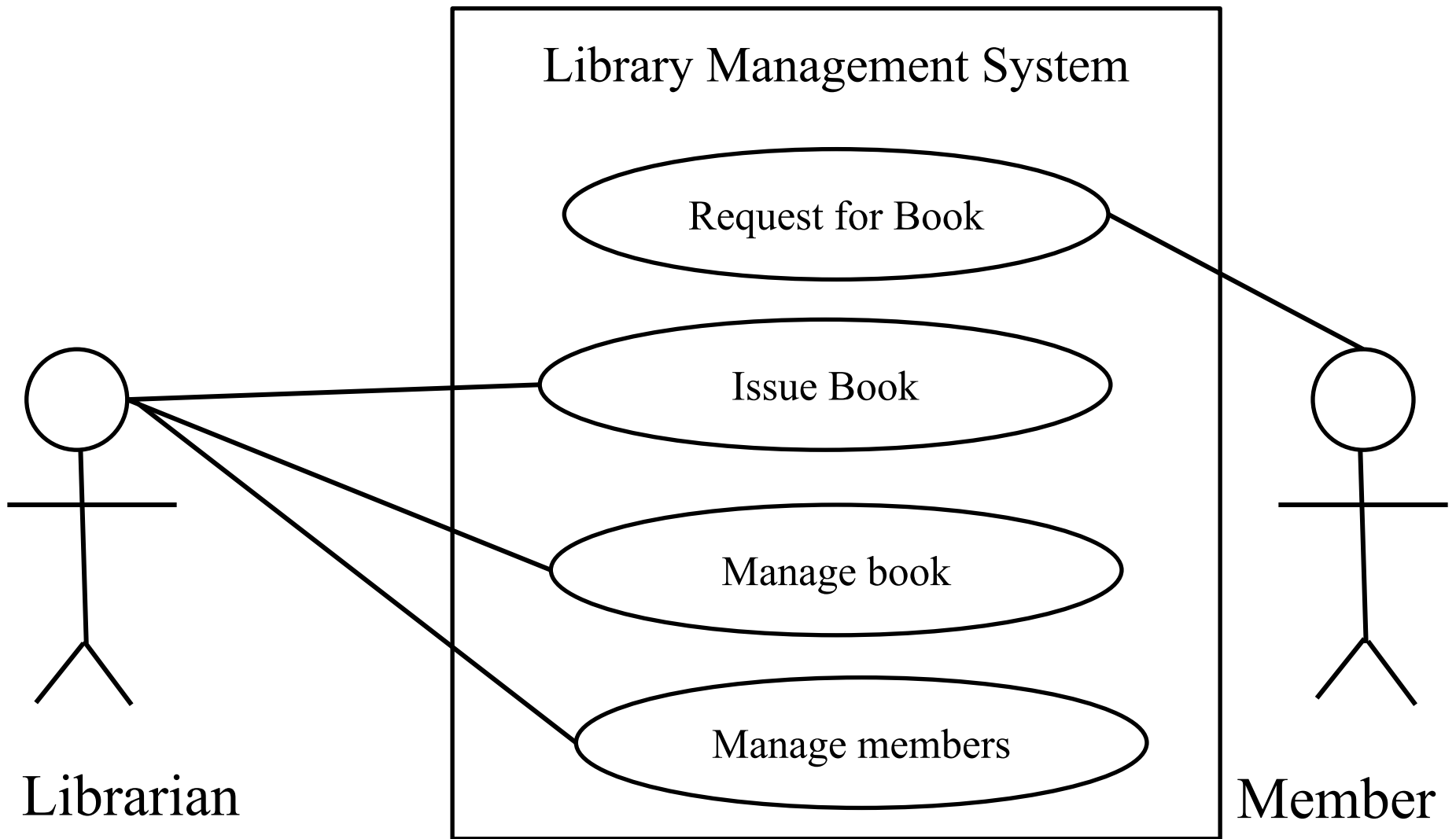


Fig: indicates that the actor customer uses the withdraw amount use case

Some Examples



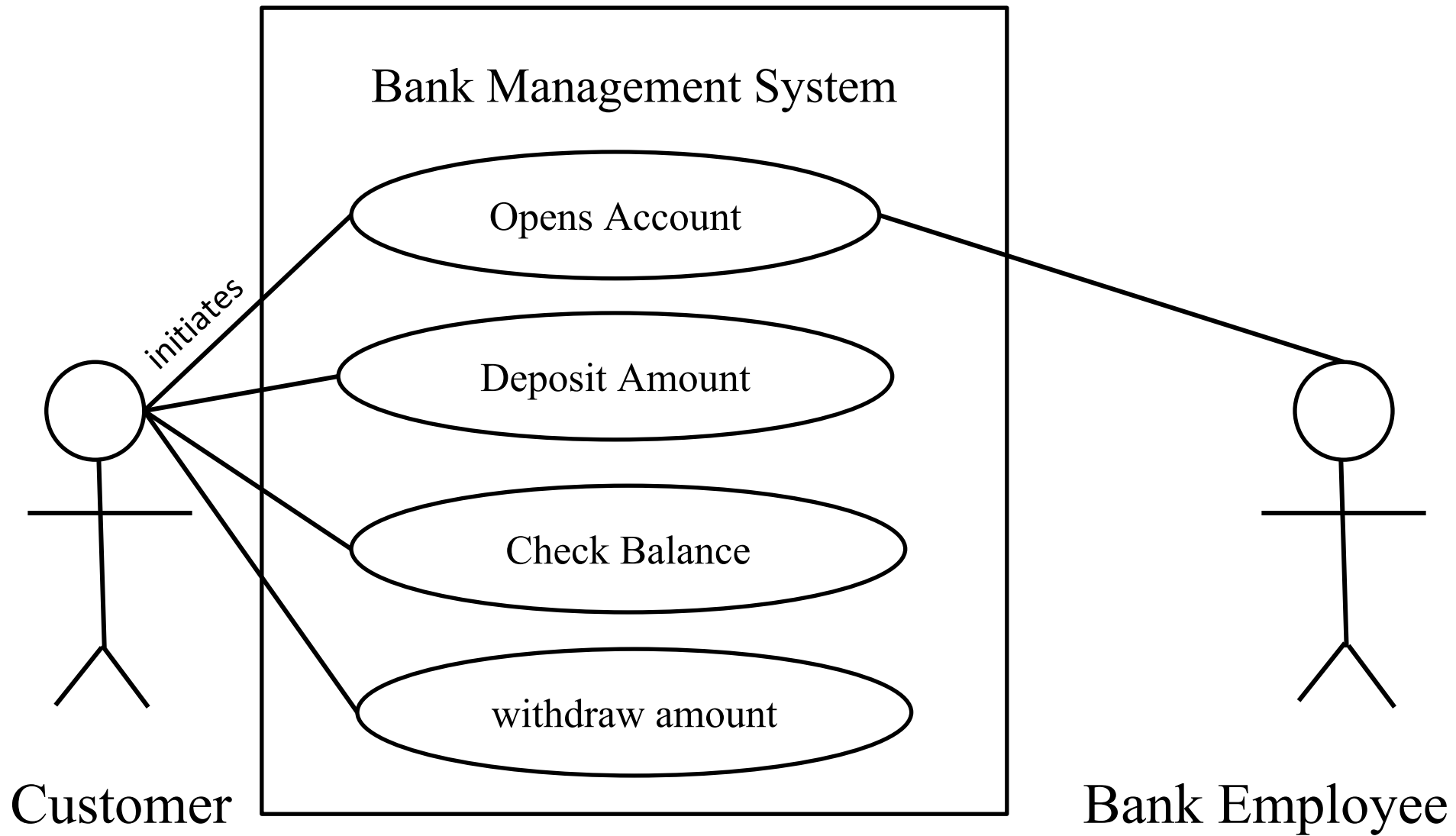


Relationship Between use cases

- Use cases many participate in relationship with other use cases.
 1. Association between actor and use case.
 2. Generalization of an actor.
 3. Extend between two use cases.
 4. Include between two use cases.
 5. Generalization of use case.

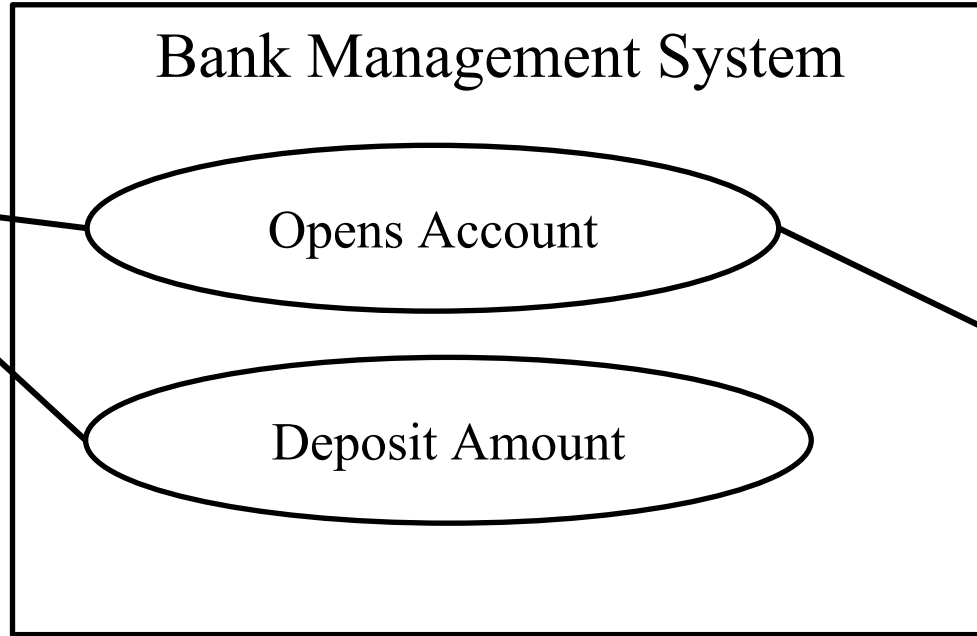
1. Association between the actor and use case

- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case.



2.Generalization of an actor

- One actor can inherit the role of other actor.
- The descendant inherits all the use cases of the ancestor.

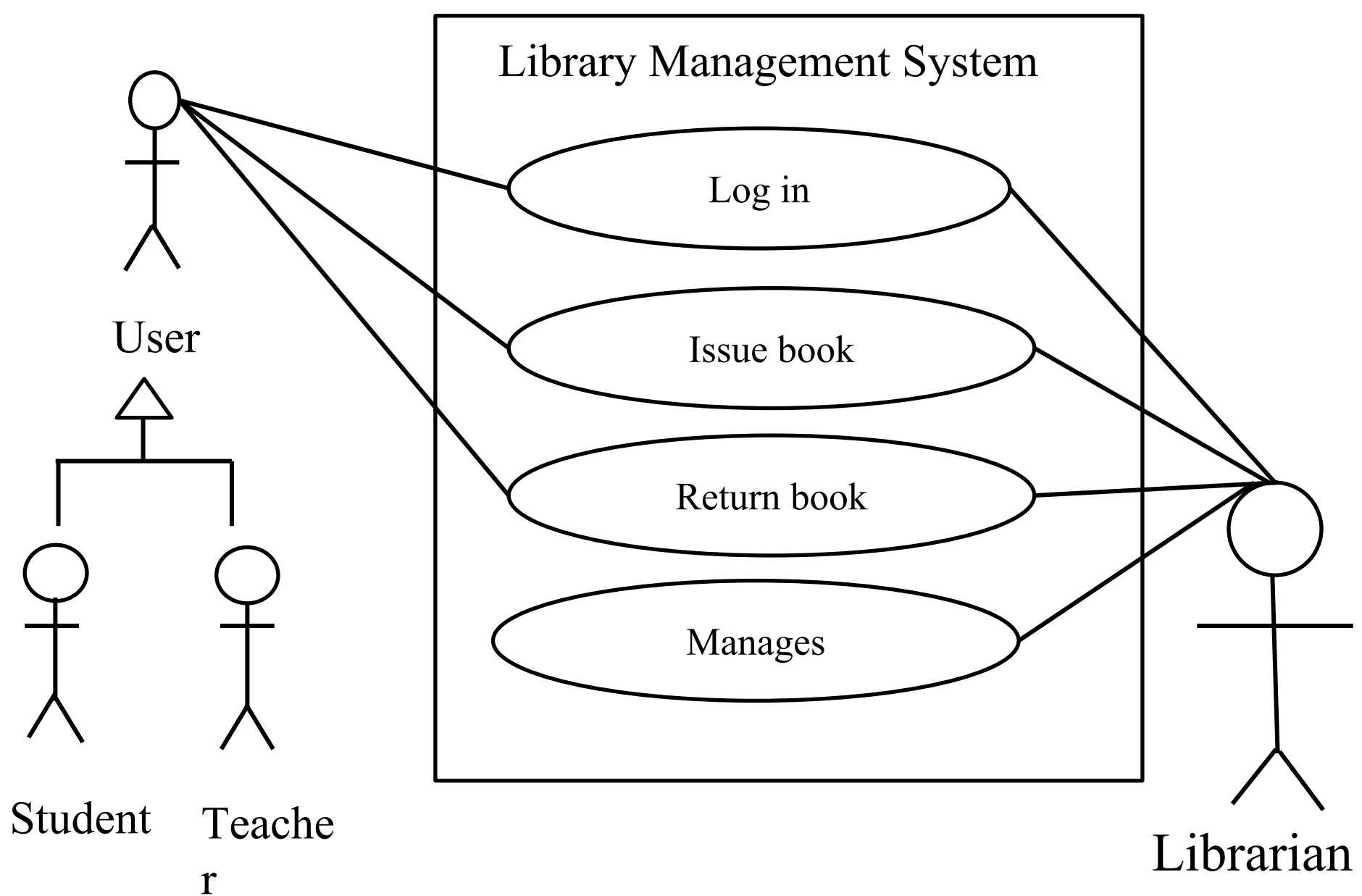


Customer

NRFC means Non-Resident Foreign Currency

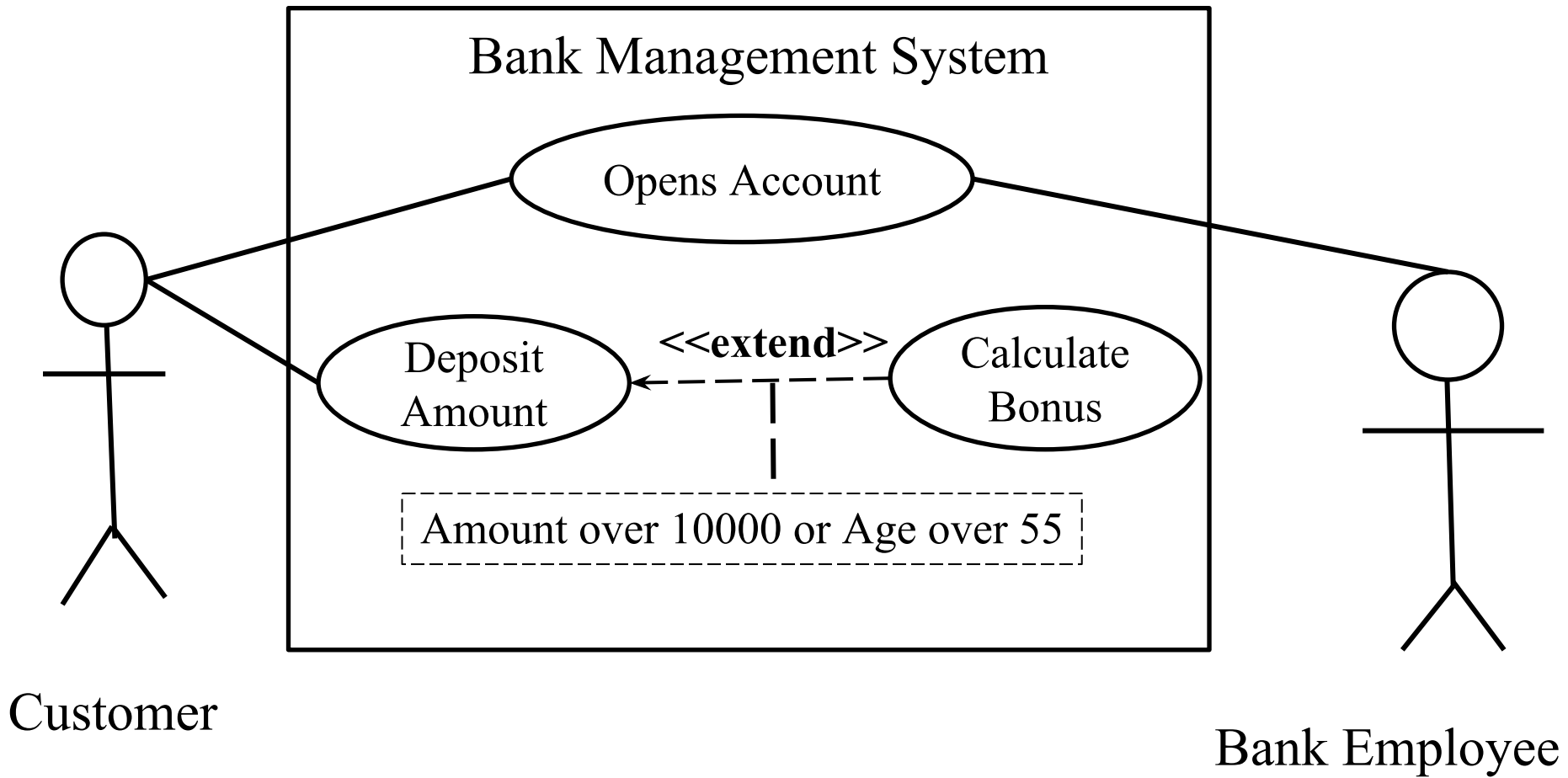
NRFC

Bank Employee



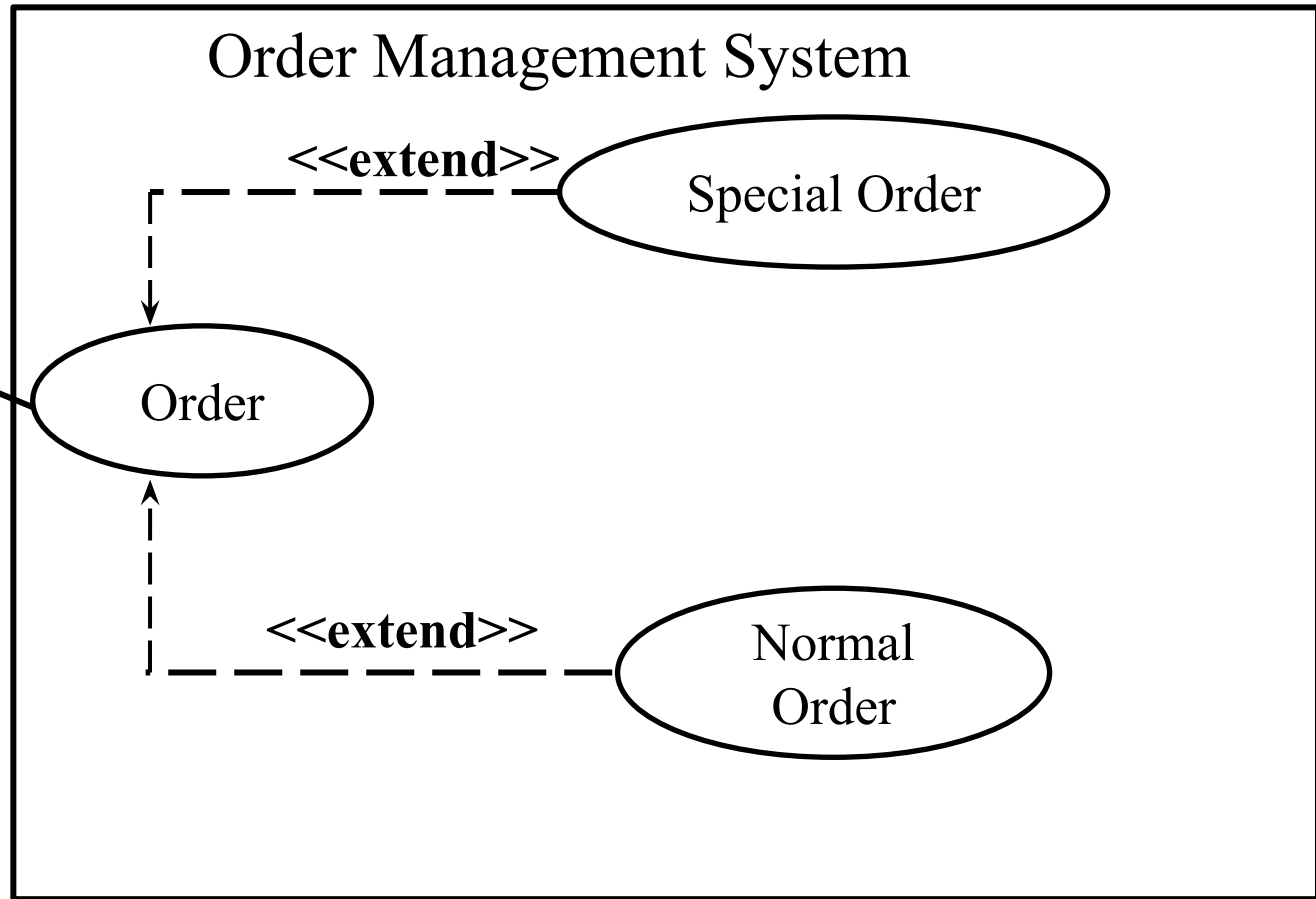
3.Extend relationship between the use cases

- As the name implies, it extends the base use case and adds more functionality to the system.
- Useful for a complex software system.



Few things need to be considered when using the <<extend>> relationship:

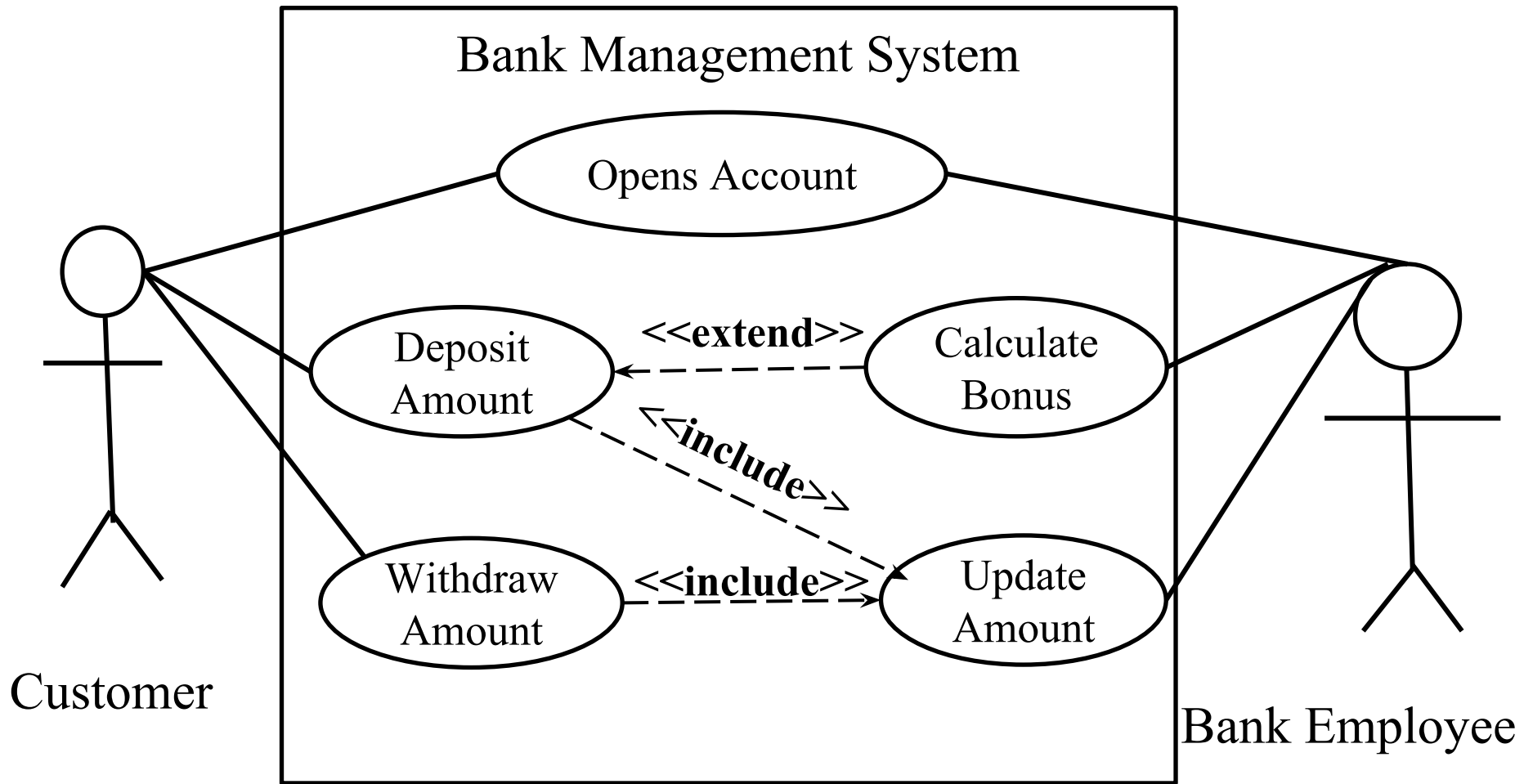
- The extending use case is dependent on the extended(base) use case. In the diagram above , the “Calculate Bonus” use case does not make much sense without the “Deposit Funds” use case.
- The extending use case is usually optional and can be triggered conditionally .In the diagram, we can see that the extending use case is triggered only for the deposits over 10,000 or when the age is over 55.
- The extended (base) use case must be meaningful on its own . This means it should be independent and must not rely on the behavior of extending use case.



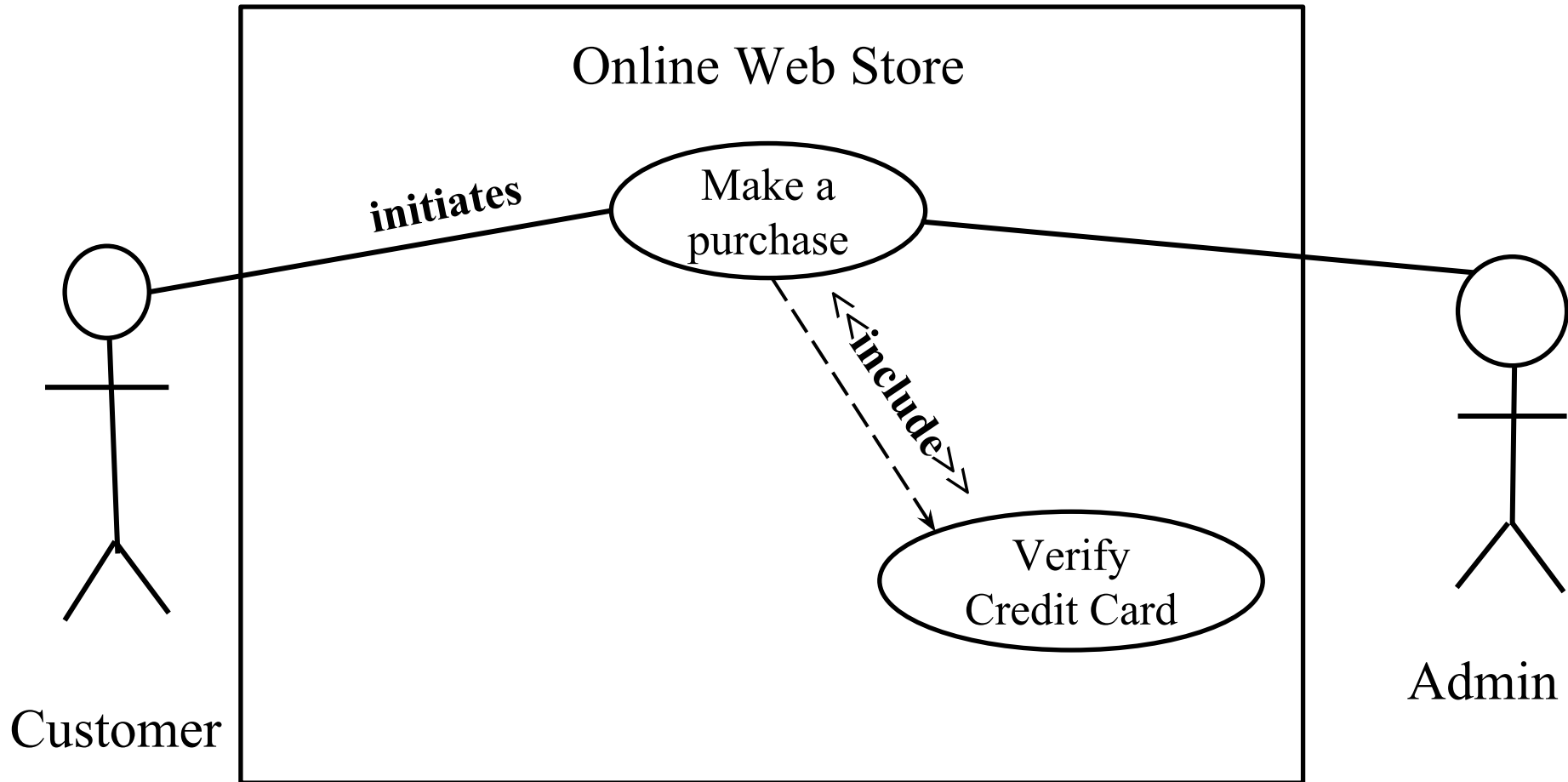
Customer

4.Include relationship between two use cases

- Shows that the behavior of the included use is part of the including(base) use case.
- Reuses the common actions across the multiple use cases.
- In some situations, this is done to simplify complex behaviors.

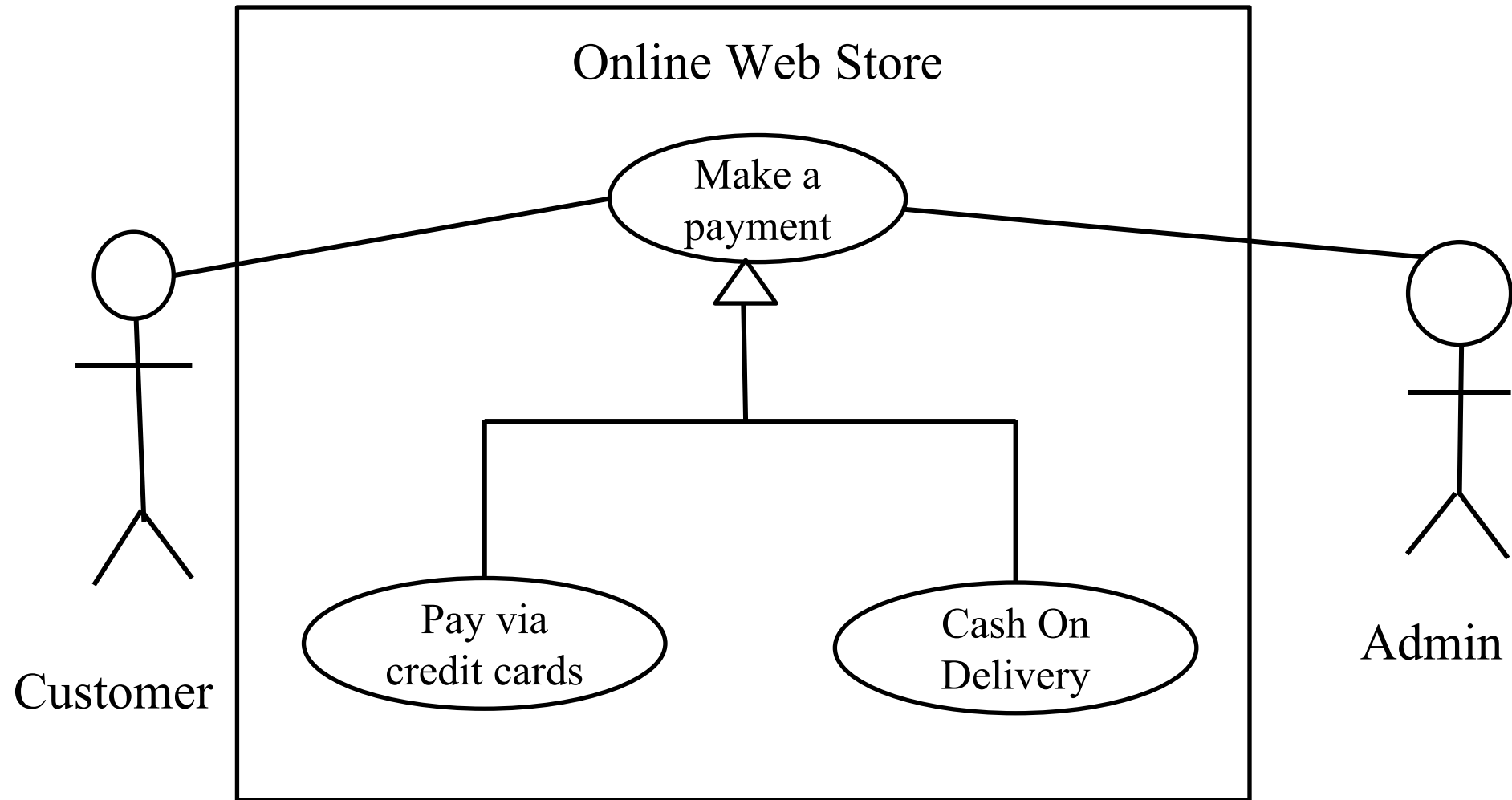


- Few things need to be considered when using the <<include>> relationship:
 - The base use case is incomplete without the included use case.
 - The included use case is mandatory and not optional.



5. Generalization of use case

- Similar to generalization of an actor.
- The behavior of ancestor is inherited by the descendant.
- Used when there is a common behavior between two use cases and also specialized behavior specific to each other.



Class Diagrams

- One of the most useful diagram in UML as they clearly map out the structure of particular system by modeling its classes ,attributes, operations and relationships among the classes.
- Class is represented as a rectangle with THREE components:
 1. The class name appears in the top compartment.
 2. The list of attributes in the middle compartment and
 3. The list of operations in the bottom compartment.

CLASS NAME	Centered ,Bold and Capitalized
List of attributes	Left aligned ,not bold and lowercase
List of operations	Left aligned ,not bold and lowercase

Visibility

- Defines the accessibility of attributes and operations to other classes.
- Can be:
 - a. Private**
 - » Denoted with a “-” sign.
 - » Hides information from anything outside the class partition.
 - b. Protected**
 - » Denoted with a “#” sign.
 - » Allow child classes to access information they inherited from a parent class.
 - c. Public**
 - » Denoted with a “+” sign.
 - » Allows all other classes to access the marked information.

Marker	Visibility
+	Public
-	Private
#	Protected
~	Package

STUDENT

-id

-name

-dateOfBirth

-gender

-address

-faculty

+getStdInfo()

+showStdInfo()

Associations

- Represents relationship between the classes.
- Place association name above , on or below the association line.
- Used a filled arrow to indicate the direction of relationship.

STUDENT

-stdid
-stdname
-dateofbirth
-gender
-address
-faculty

+getstdinfo()
+showstdinfo()

register for ►

COURSE

-crseid
-title
-description
-duration
-cost

+getcrseinfo()
+showcrseinfo()

Multiplicity(Cardinality)

- Indicates the number of instances of one class that can be associated to one or more instances of another class.
- Represents how many objects of one class can participate in a relationship/association.
- **Notation:**
lower bound.....upper bound
- Example
3.....9
 - Minimum no of objects is 3 and maximum is 9 that can participate in association.

Indicator	Meaning	Minimum	Maximum
0.....1	Zero or more	0	1
1	Only one	Exactly one	
0.....*	0 or more	0	many
1.....*	1 or more	1	many
1.....n	1 or more	1	many
*	1 or more	1	many

Degree of association

- Depending upon the number of classes participating in an association , degree of association can be classified as follows:
 1. Unary(1^0)
 2. Binary(2^0)
 3. Ternary(3^0)
 4. Many(higher degree)

1.Unary

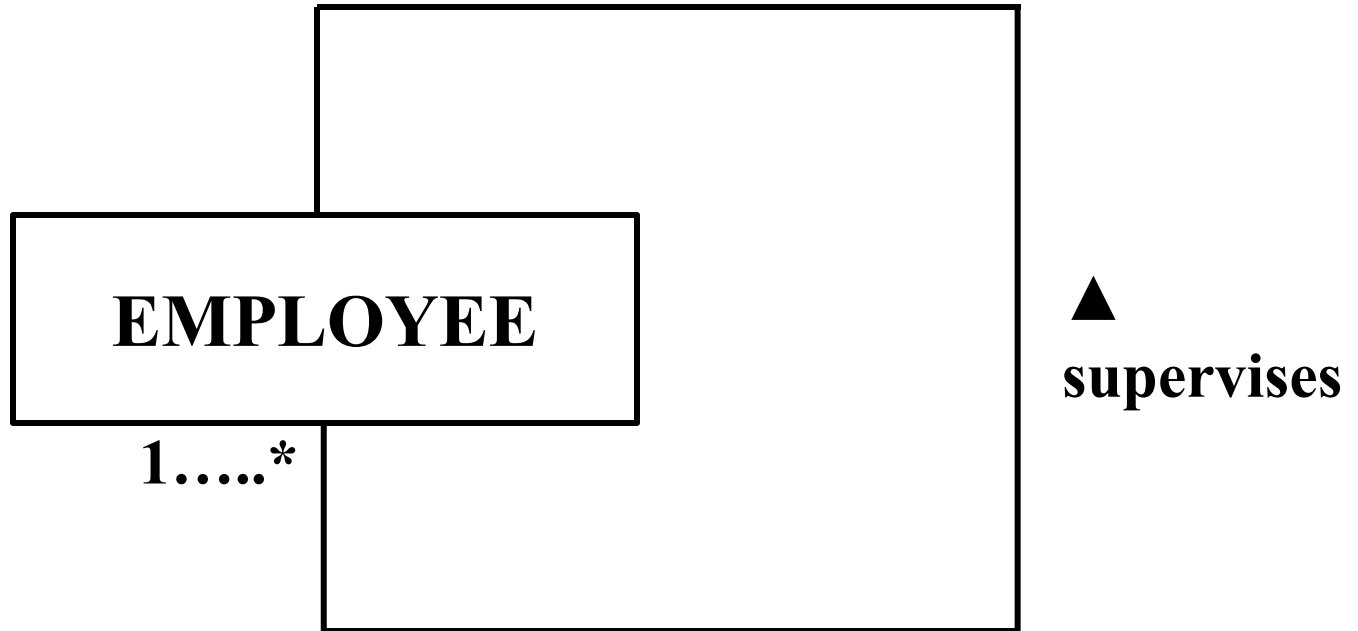


Fig : One employee supervises 1 or many employees

2.Binary

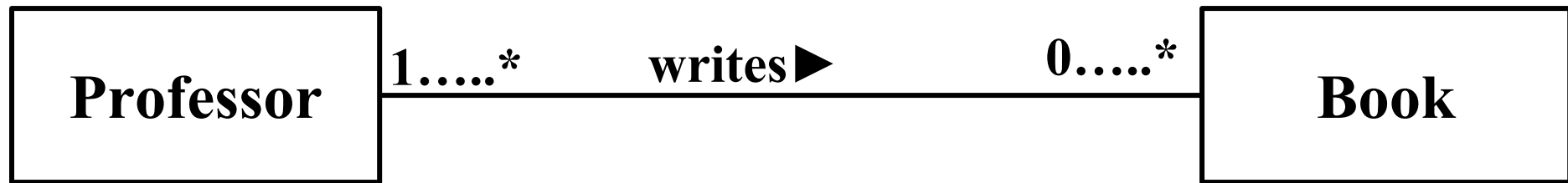


Fig : One professor writes no or many books

Or

One book is written by one or many professor

3.Ternary

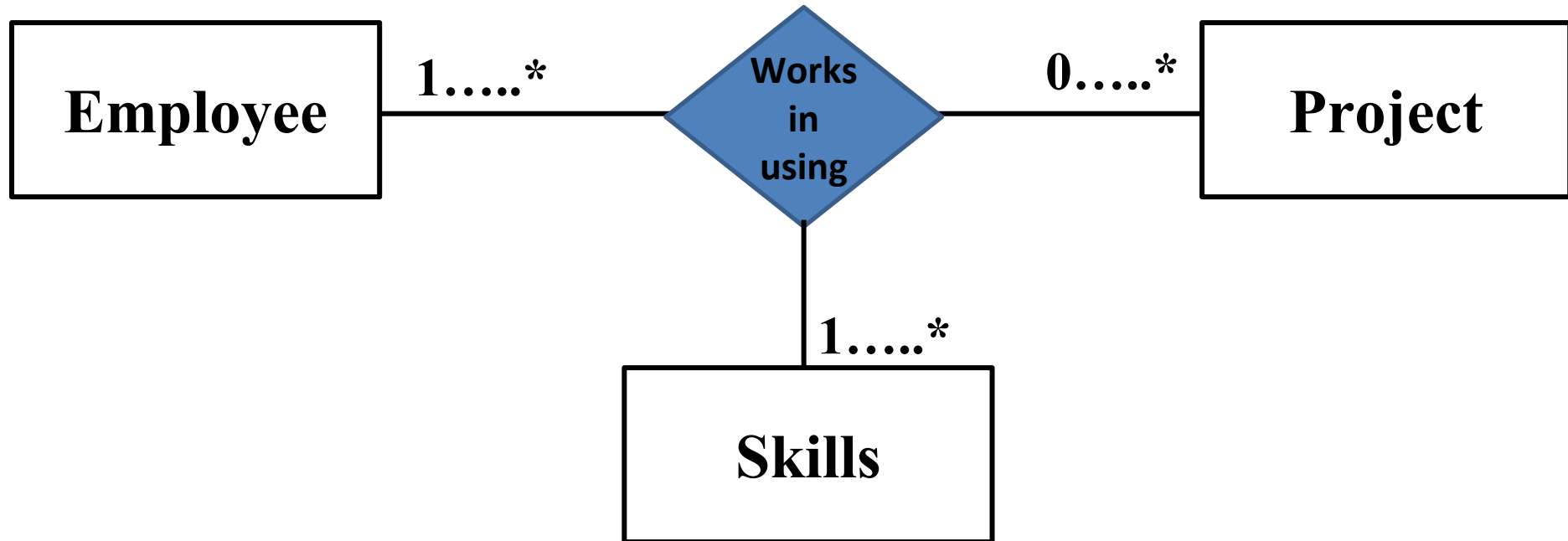
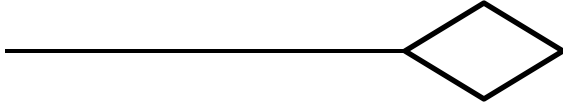
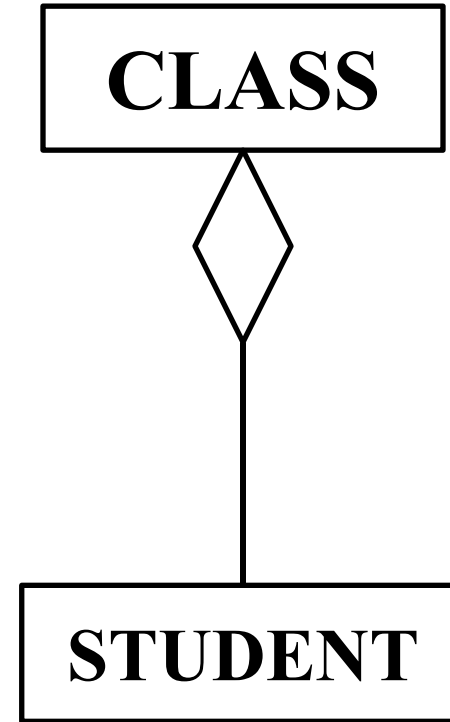
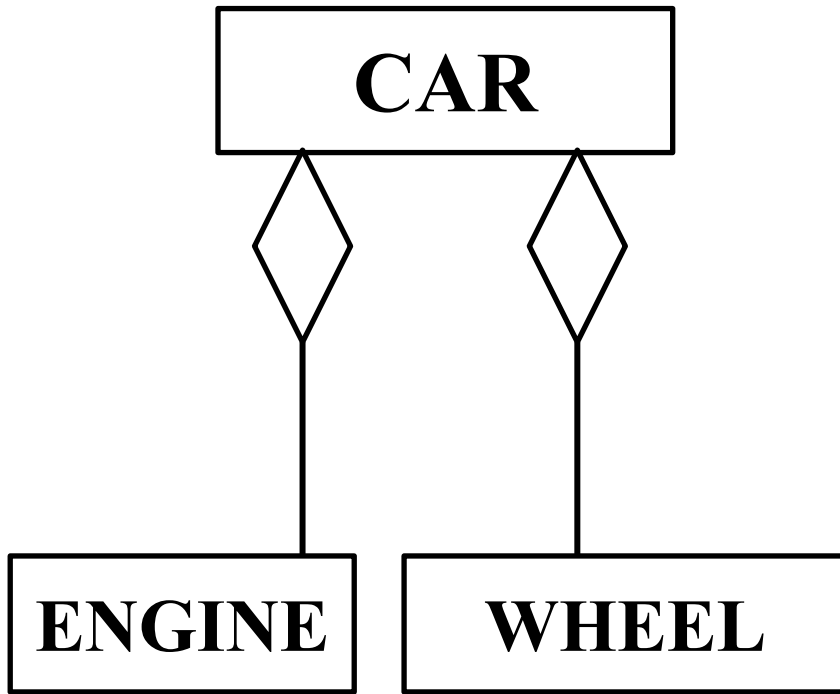


Fig : One employee works in zero or more projects using one or multiple skills

Aggregation

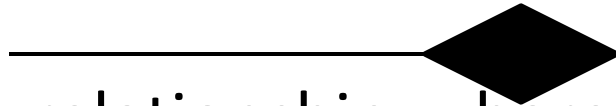
- Is a special case of association.
- A directional association between the classes.
- When an object class has another object class then there exist aggregation.
- Also called as “**has a**” relationship.
- Denoted by a line with attached hollow diamond shape. 
- implies **a part of** relationship.
- A relationship where the part classes can exist independently of the main class.



- Here ,ENGINE and WHEEL classes would be meaning less if there class CAR doesn't exist but they can survive independently.

Compositions

- A special case of aggregation.
- A simple restricted aggregation is called composition.
- When a class contains another class if the contained class cannot exist without the existence of container class then it is called composition.
- Represented by a line attached with filled diamond.



- Implies a relationship where the part classes cannot exist independent of the main classes.
- Example: House (parent) and Room (child). Rooms don't exist separate to a House.

HOUSE

```
graph TD; HOUSE[HOUSE] --- DIAMOND1{ }; DIAMOND1 --- ROOMS[ROOMS]; HOUSE --- DIAMOND2{ }; DIAMOND2 --- WINDOWS[WINDOWS]; PERSON[PERSON] --- DIAMOND3{ }; DIAMOND3 --- LEG[LEG]; PERSON --- DIAMOND4{ }; DIAMOND4 --- HAND[HAND]; PERSON --- DIAMOND5{ }; DIAMOND5 --- HEAD[HEAD];
```

ROOMS

WINDOWS

PERSON

LEG

HAND

HEAD

Inheritance

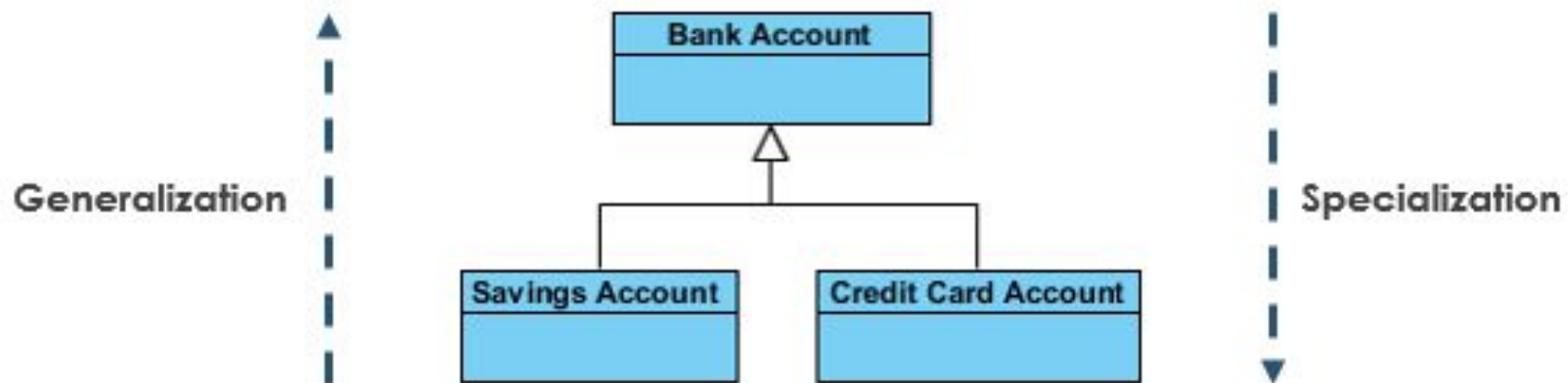
- Several classes may have the same attributes and/or methods. When this occurs, a general class is created containing the common attributes and methods.
- The specialized class inherits or receives the attributes and methods of the general class.
- In addition, the specialized class has attributes and methods that are unique and only defined in the specialized class.
- Represents **is a** relationship.
- Includes two terms:
 1. Generalization association
 2. Specialization association

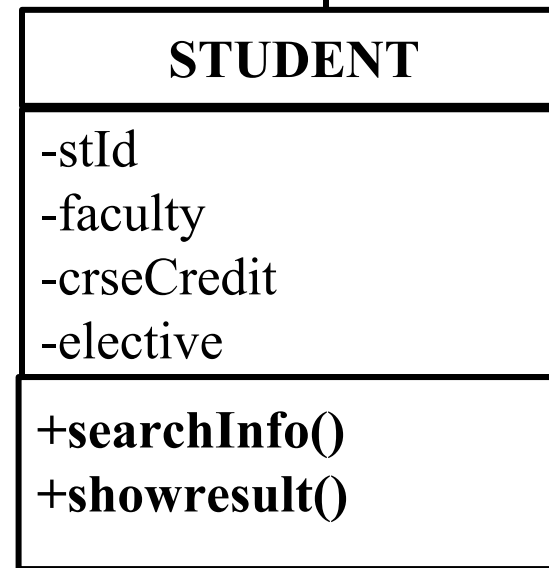
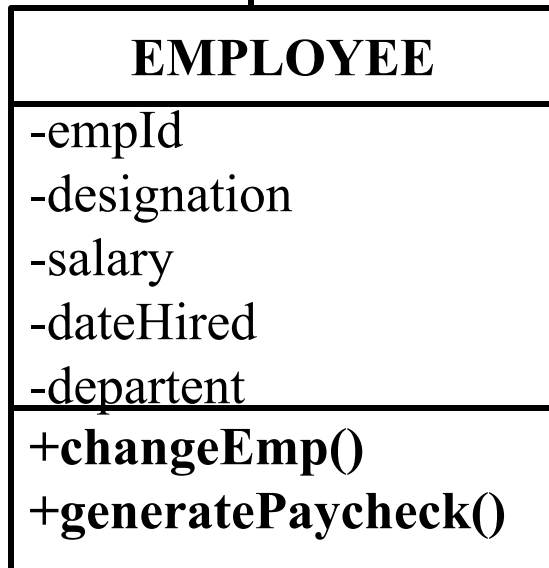
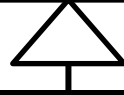
1.Generalization association

- The process of extracting shared characteristics from two or more classes and combining them into a generalized class called as **SUPER CLASS** is called generalization.
- Shared characteristics can be attributes , methods or association.
- The most generalized class is placed at the top of the diagram.

2.Specialization association

- The process of creating new sub classes from an existing class called generalization.
- If it turns out that certain attributes, methods or association applied to some of the object of the class, a new class called as **SUB CLASS**.
- The sub class is placed below the super class.





Object Diagram

- It is an instance of a class diagram.
- It shows a snapshot of the detailed state of a system at a point in time, thus an object diagram encompasses objects and their relationships at a point in time.
- It may be considered a special case of a class diagram or a communication diagram.
- Represented by TWO compartments:
 1. The name of the object and its class are underlined in the top compartment with syntax:
object name: class name
 2. The object's attributes and their values are in the second compartments.

Ram Sharma : STUDENT

stId :1

stdName : Ram Sharma

faculty:1996-07-21

address : Ramkot

elective : Java

emailID : ram.sharma@gmail.com

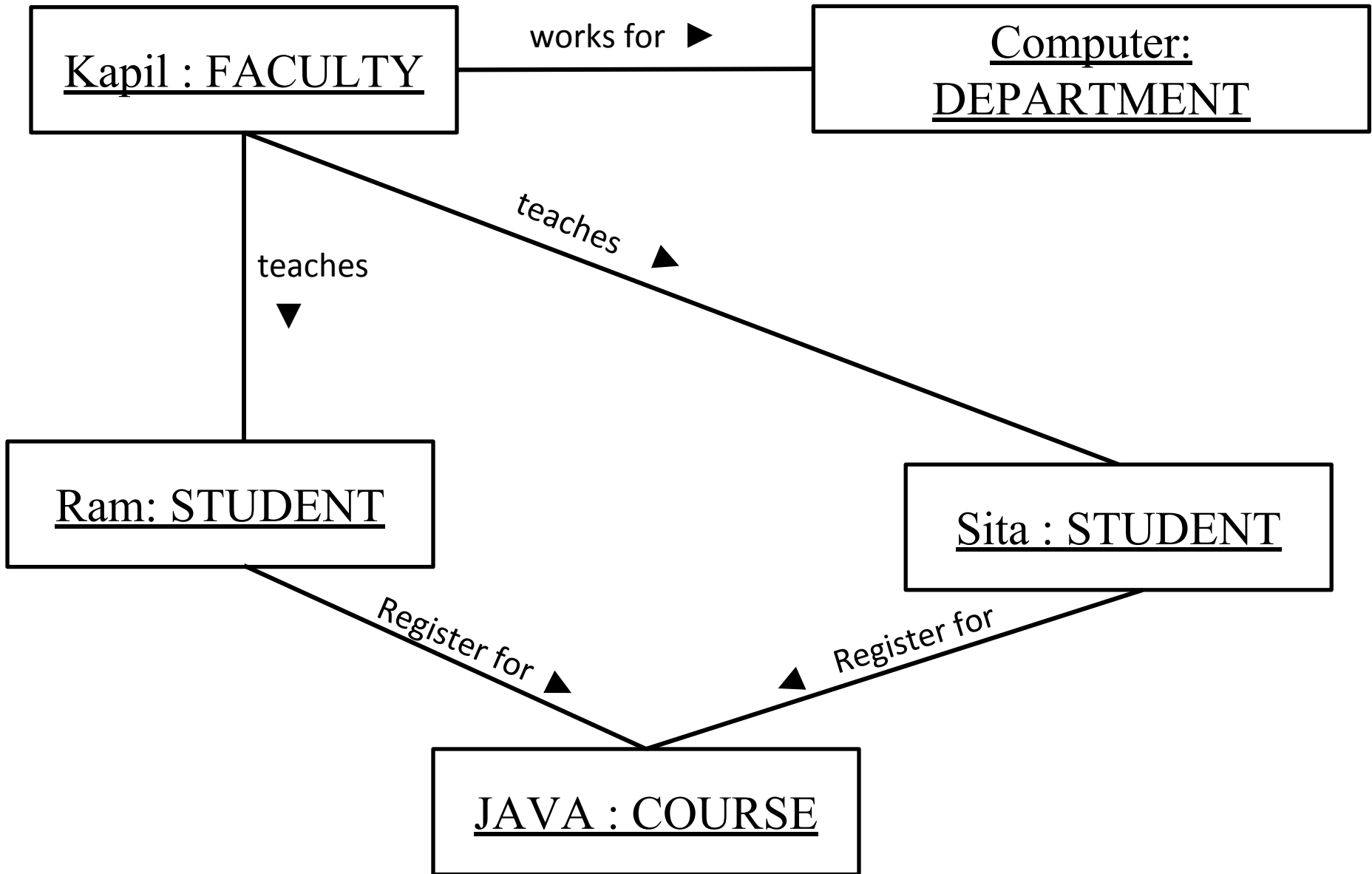


Fig: Object Diagram Of College

More examples:

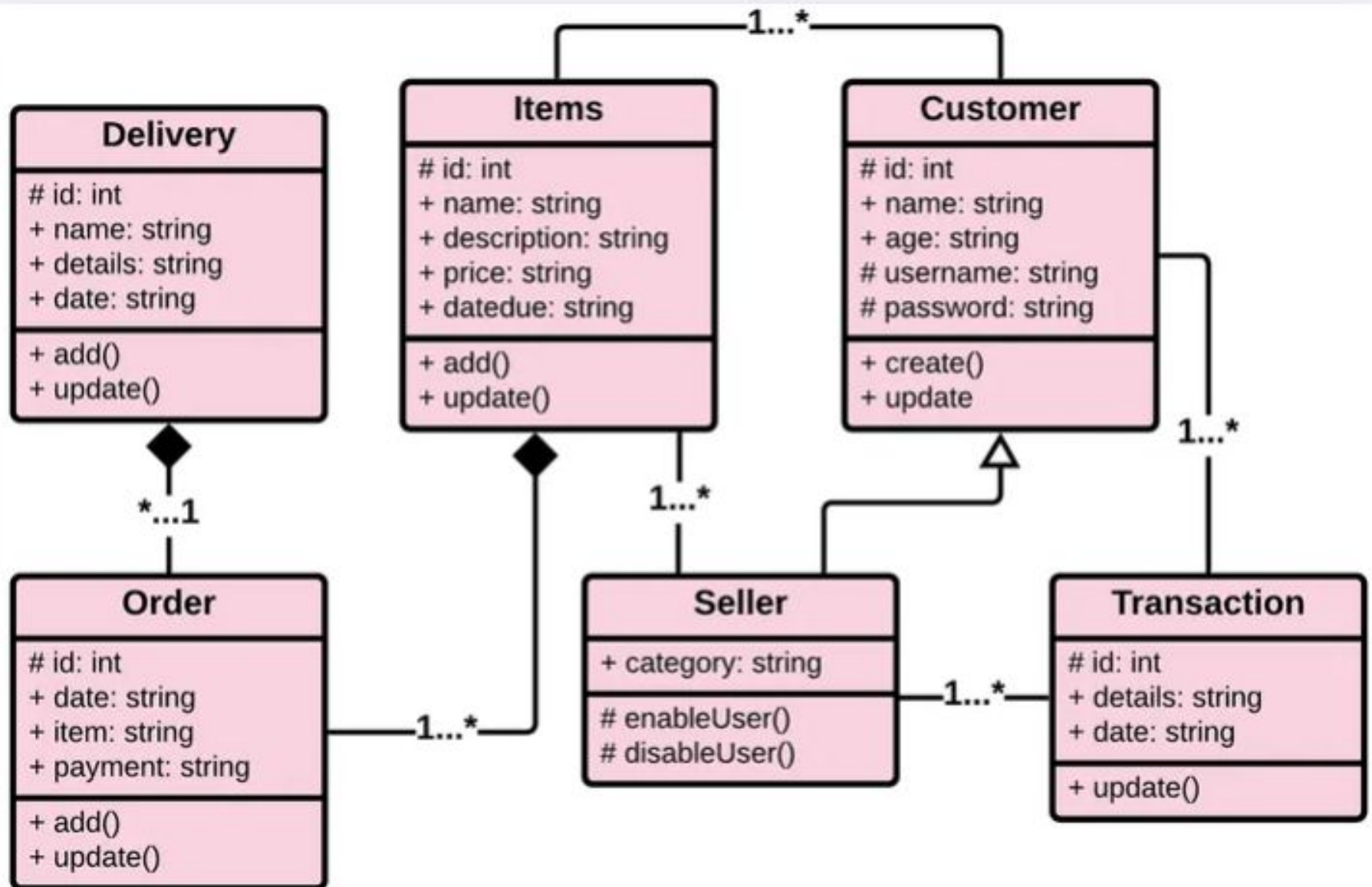


Figure: Class Diagram for Online Shopping System

Draw object diagram for online shopping system(above diagram).

State Diagram

- It depicts the life cycle of a single object by showing the different states that an object can have , the events that cause the object to change their state over the time and the rules that govern the objects transition between the states.
- The process of changing the state of an object at particular at state of time is called state transition.
- An object can be in particular state for sometimes before transitioning to next state . State depends upon the values of attributes and links with other objects.
- While determining the state of an object ,only those attributes that have direct effect on the object behavior are considered.
- For example ,address of a person may not be important attribute but if you want to classify a person as being in one of the TWO states.





At home

talk()
{ informally }

At work

talk()
{ formally }

UML notation in state diagram

1.  : A small ,solid ,filled circle represents initial state of an object.
2.  : a rounded cornered rectangle represents an intermediate state with some internal activities.
3.  : a bull's eye ,a small solid circle inside the hollow circle represents the final state.
4.  : an arrow line represents the state transition(movement from one state to another).Each transition is labelled with the event that causes the transition.

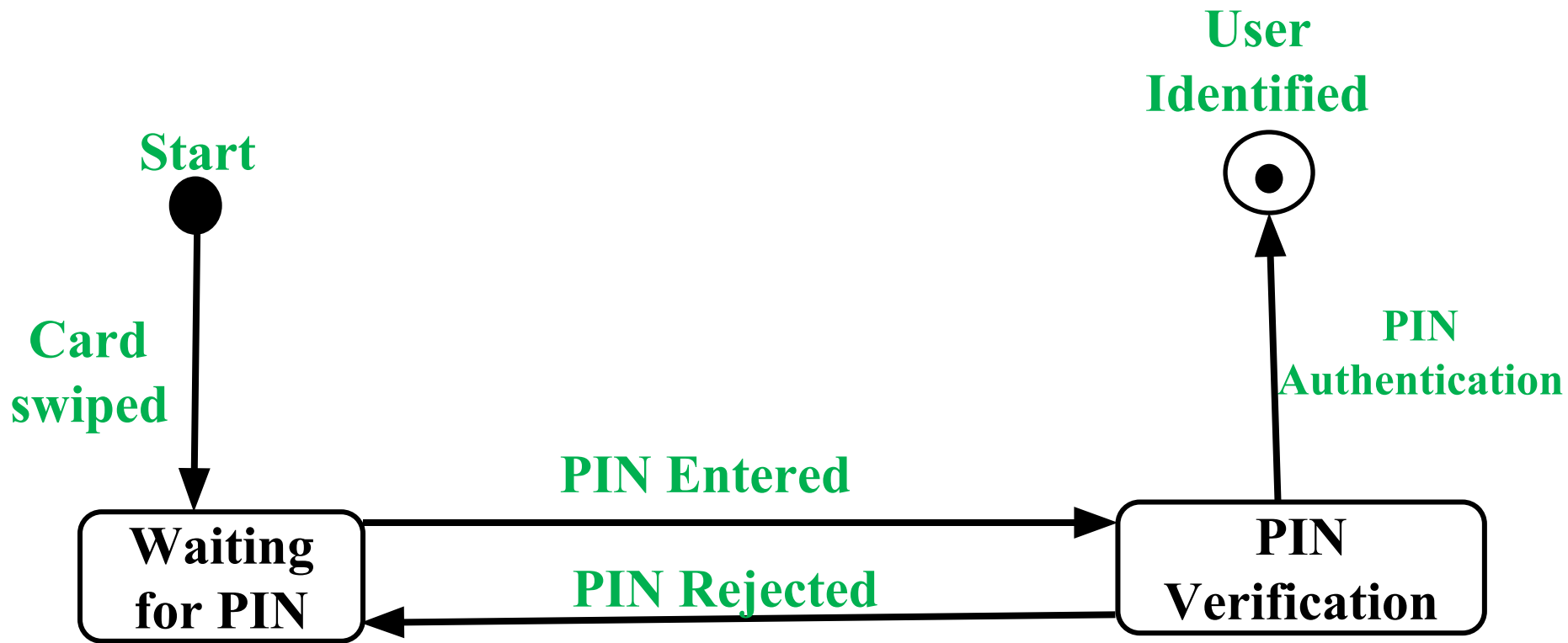


Fig : A state diagram for user verification

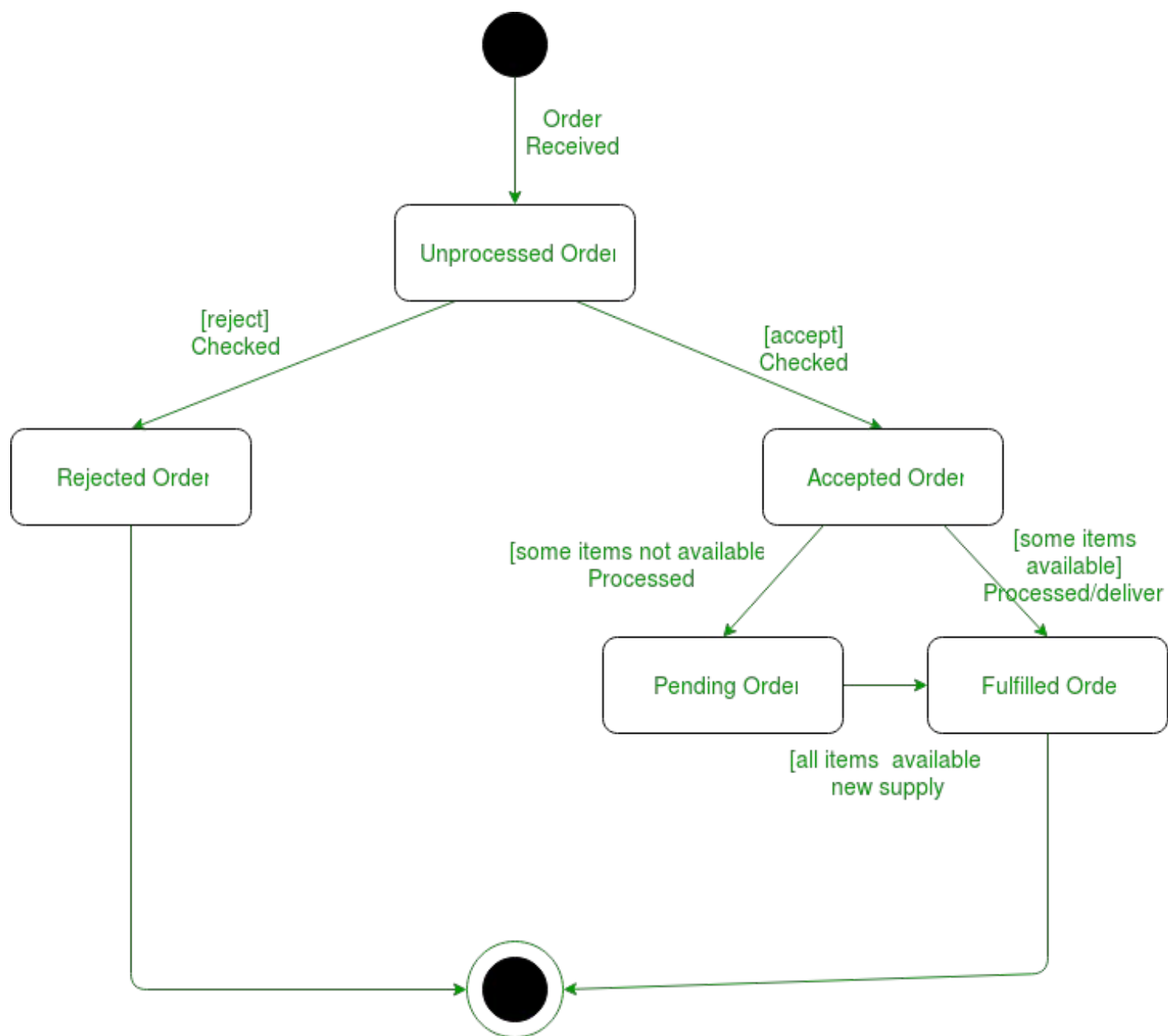
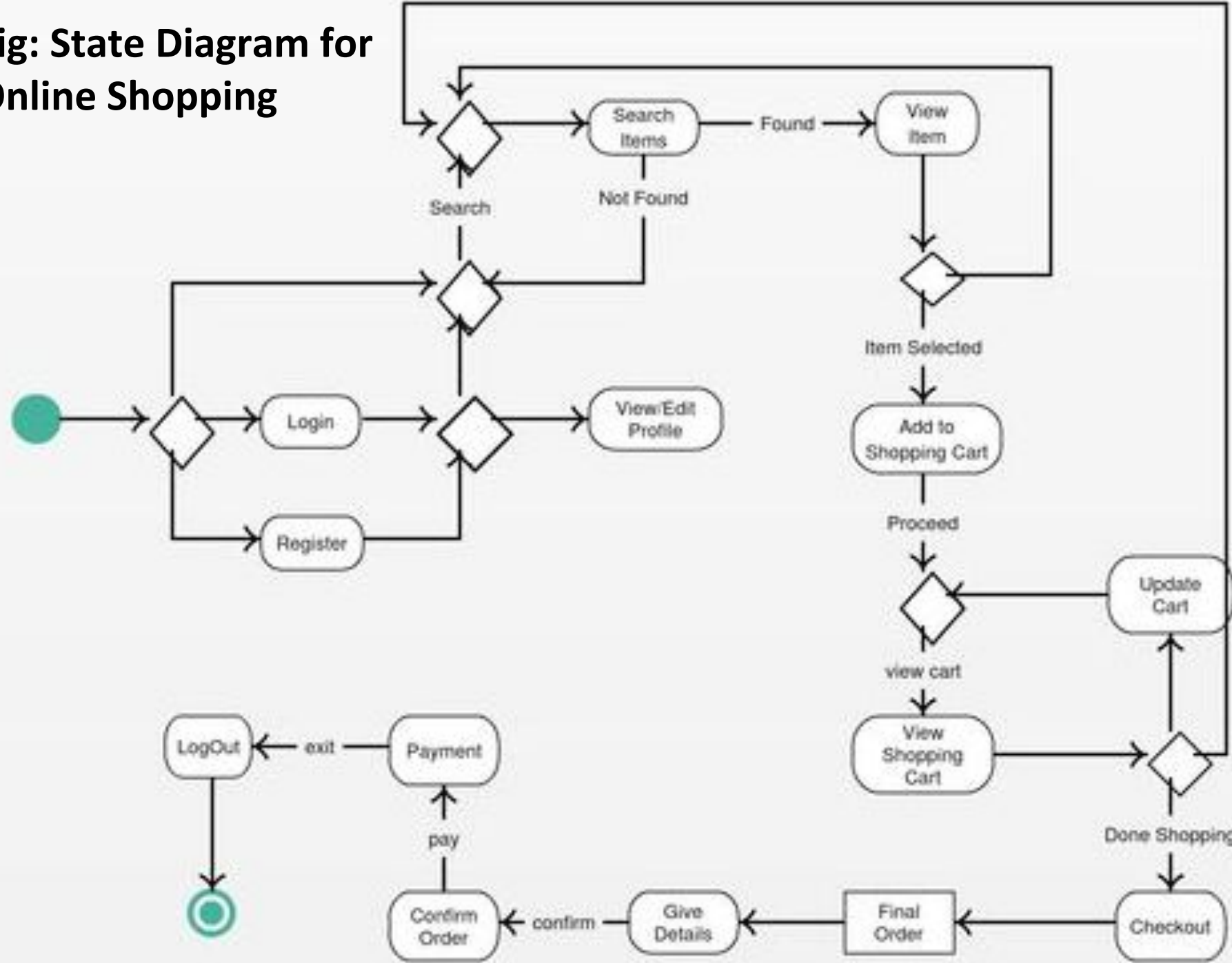


Fig: A state diagram for an online order

Fig: State Diagram for Online Shopping



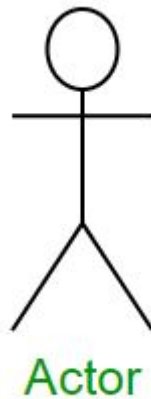
Sequence Diagram

- The most commonly used **interaction** diagram.
- It depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.
- It describes how and in what order the objects in a system function.
- These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

UML notation in Sequence diagram

1. Actors :

- Represents a type of role where it interacts with the system and its objects.
- An actor is always outside the scope of the system .



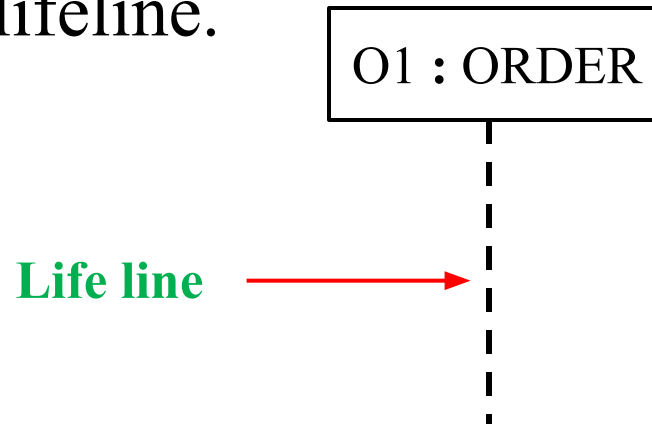
2. Class roles or participants or objects:

- A named element which depicts an individual participant in a sequence diagram.
- Represented by

Object name : Class name

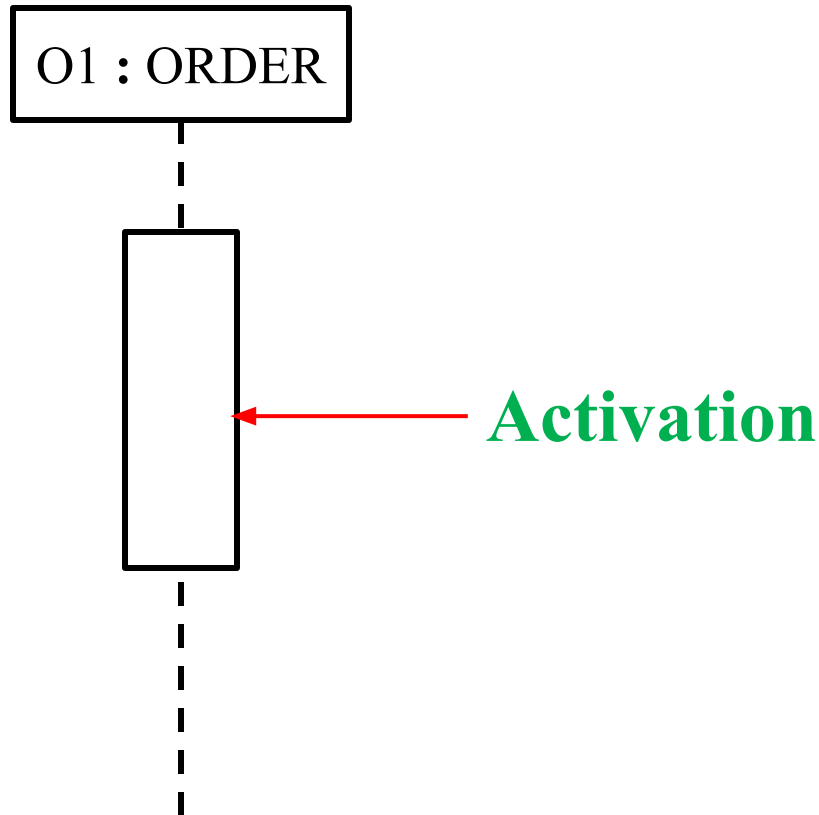
3. Lifeline :

- a vertical dashed line ,indicating the presence of an object over the time.
- each instance in a sequence diagram is represented by a lifeline.



4.Activation or execution occurrence:

- Represents the time period during which an object performs an operations.
- Represented by :



5.Messages:

- Communication between objects is depicted using messages.
- The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram .
- Messages can be broadly classified into the following **categories**:
 - » Synchronous
 - » Asynchronous Messages
 - » Create message
 - » Delete Message
 - » Self Message
 - » Reply Message
 - » Found Message
 - » Lost Message

Synchronous messages :

- A synchronous message waits for a reply before the interaction can move forward.
- We use a solid arrow head to represent a synchronous message.
- The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message.

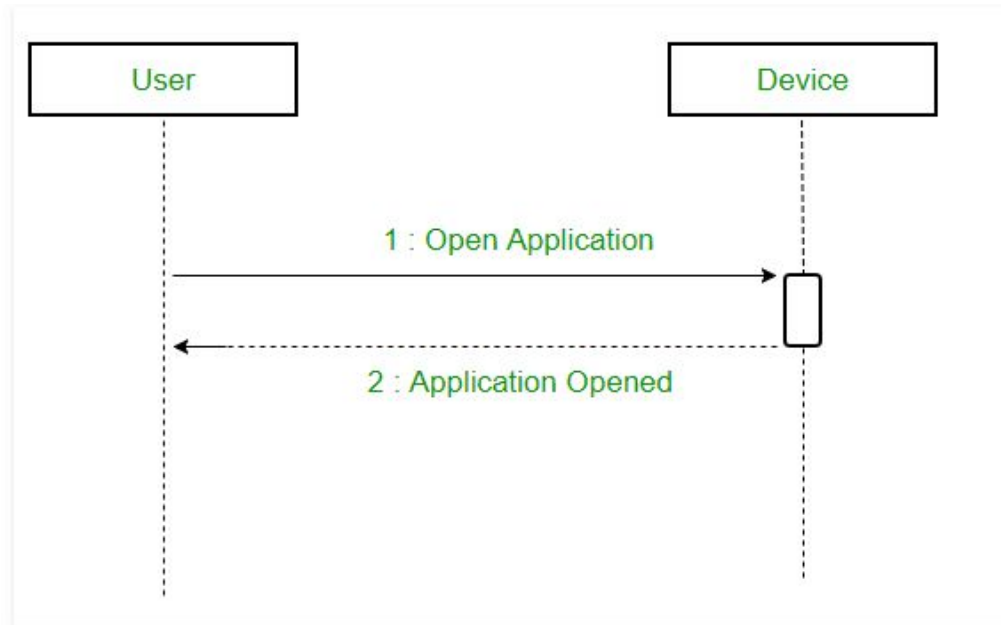
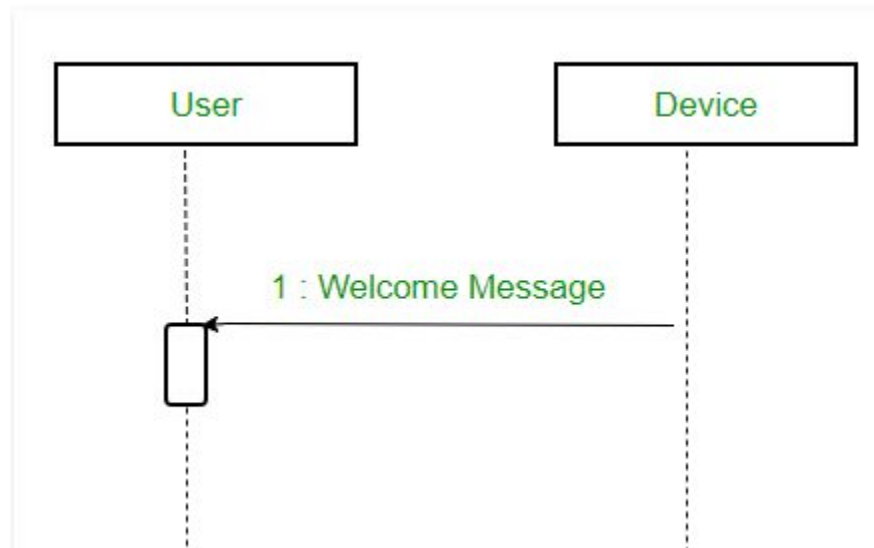


Figure – a sequence diagram using a synchronous message

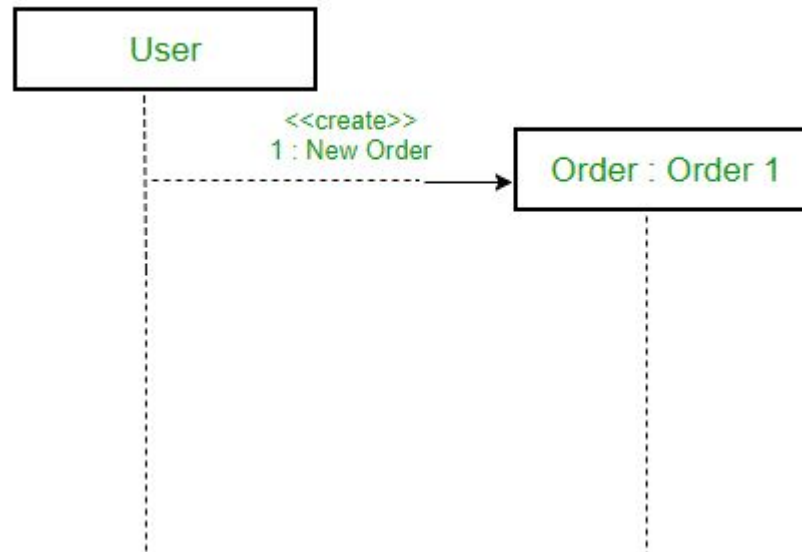
Asynchronous Messages :

- An asynchronous message does not wait for a reply from the receiver.
- The interaction moves forward irrespective of the receiver processing the previous message or not.
- We use a lined arrow head to represent an asynchronous message.



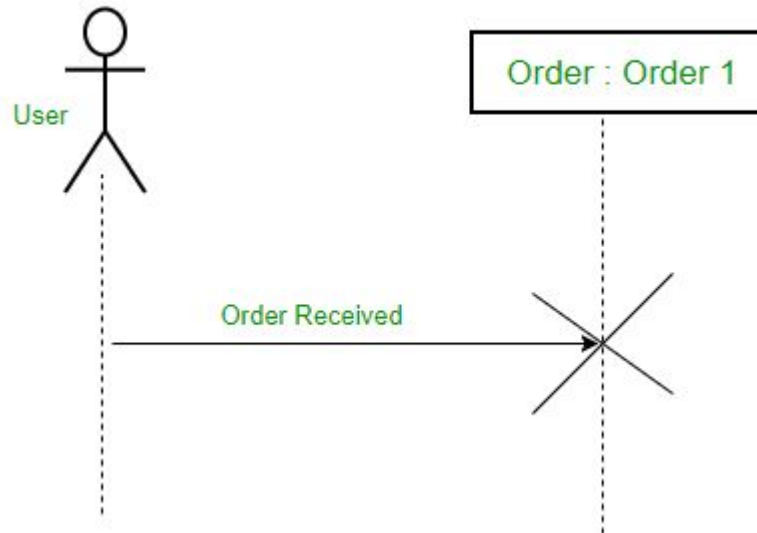
Create message :

- We use a Create message to instantiate a new object in the sequence diagram.
- There are situations when a particular message call requires the creation of an object.
- It is represented with a dotted arrow and create word labeled on it to specify that it is the create Message symbol.



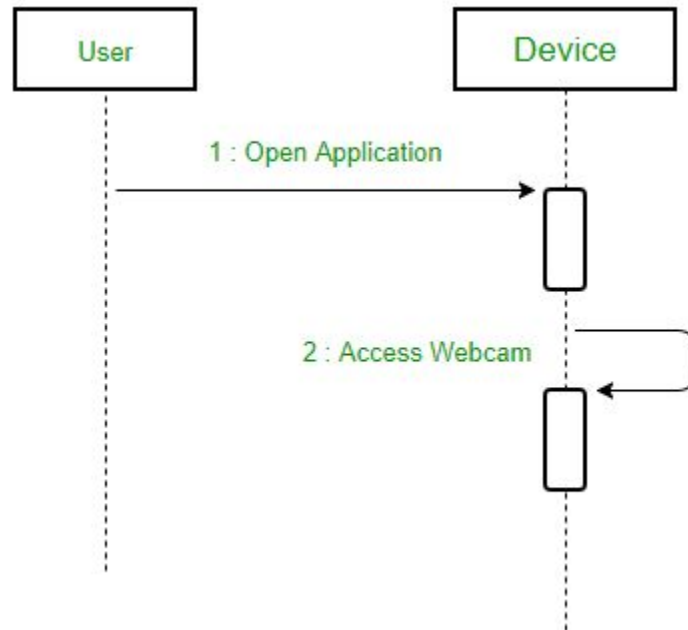
Delete Message :

- We use a Delete Message to delete an object.
- It destroys the occurrence of the object in the system.
- It is represented by an arrow terminating with a x.



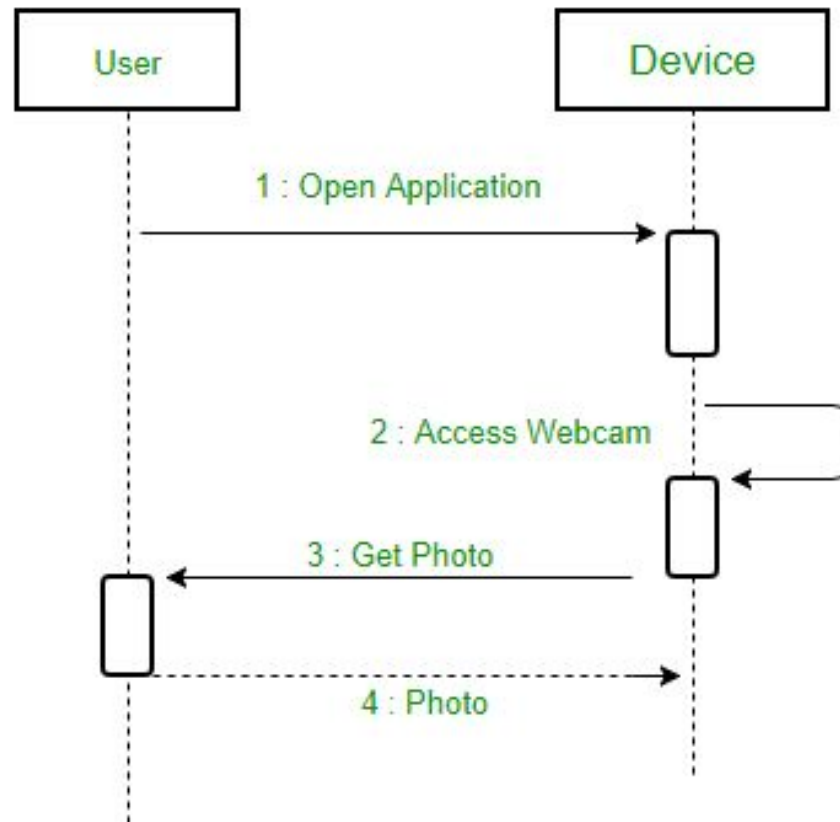
Self Message :

- Certain scenarios might arise where the object needs to send a message to itself.
- Such messages are called Self Messages and are represented with a U shaped arrow.



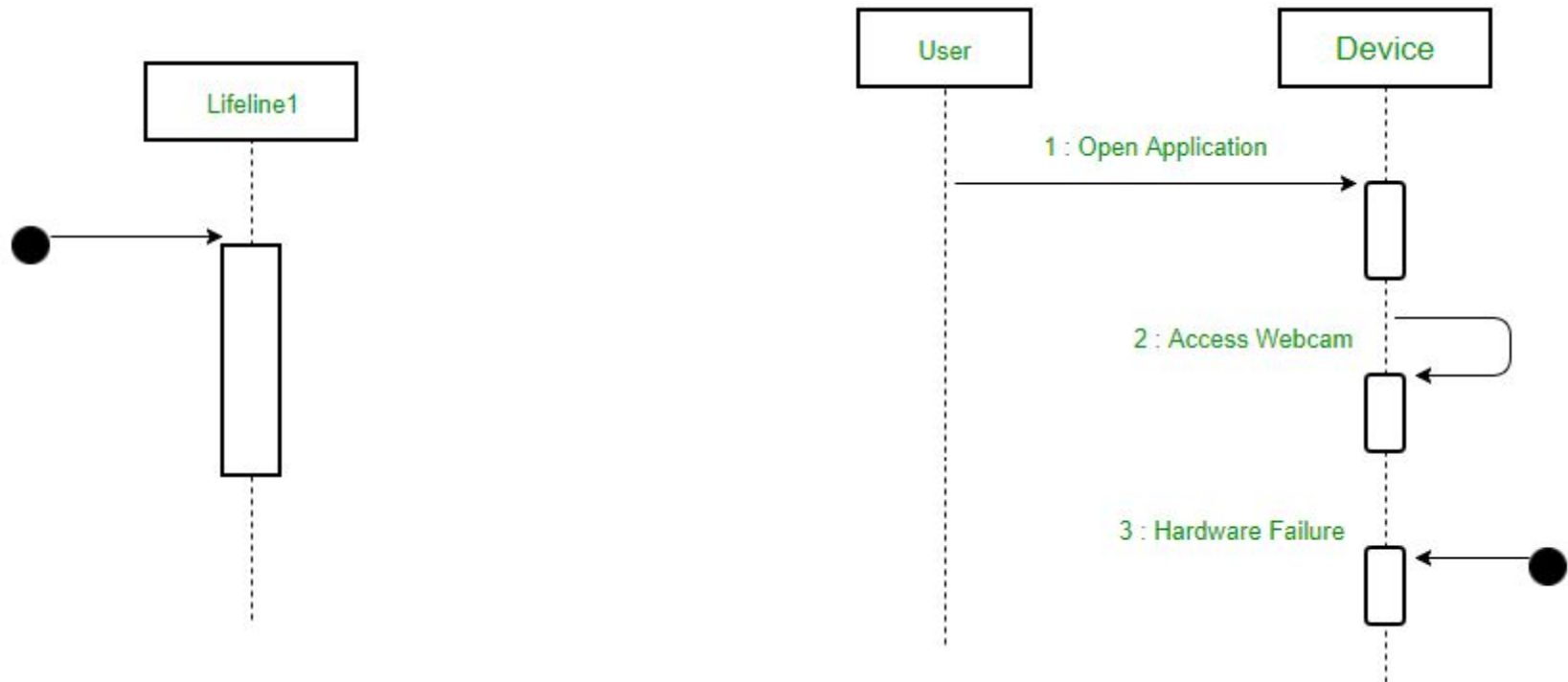
Reply Message :

- Reply messages are used to show the message being sent from the receiver to the sender.
- We represent a return/reply message using an open arrowhead with a dotted line.



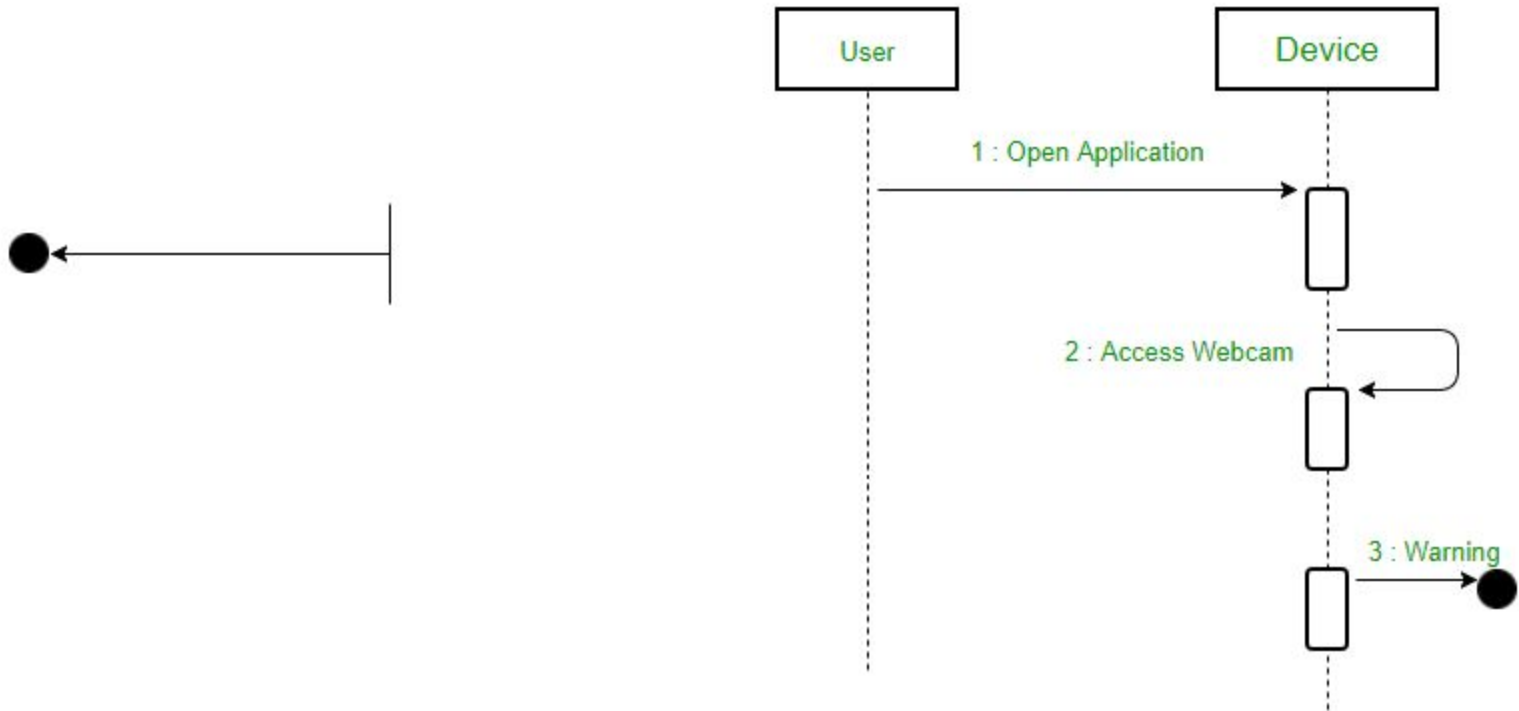
Found Message :

- A Found message is used to represent a scenario where an unknown source sends the message.
- It is represented using an arrow directed towards a lifeline from an end point.
- For example: Consider the scenario of a hardware failure.



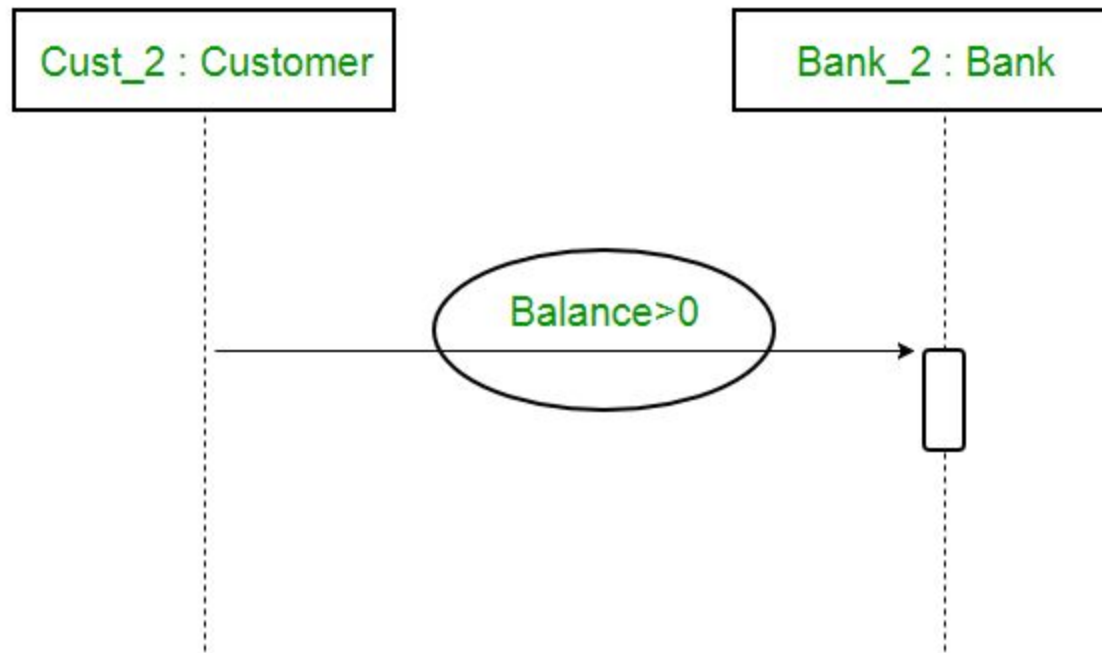
Lost Message :

- A Lost message is used to represent a scenario where the recipient is not known to the system.
- It is represented using an arrow directed towards an end point from a lifeline.
- For example: Consider a scenario where a warning is generated.



6. Guards :

- To model conditions we use guards in UML.
- They are used when we need to restrict the flow of messages on the pretext of a condition being met.



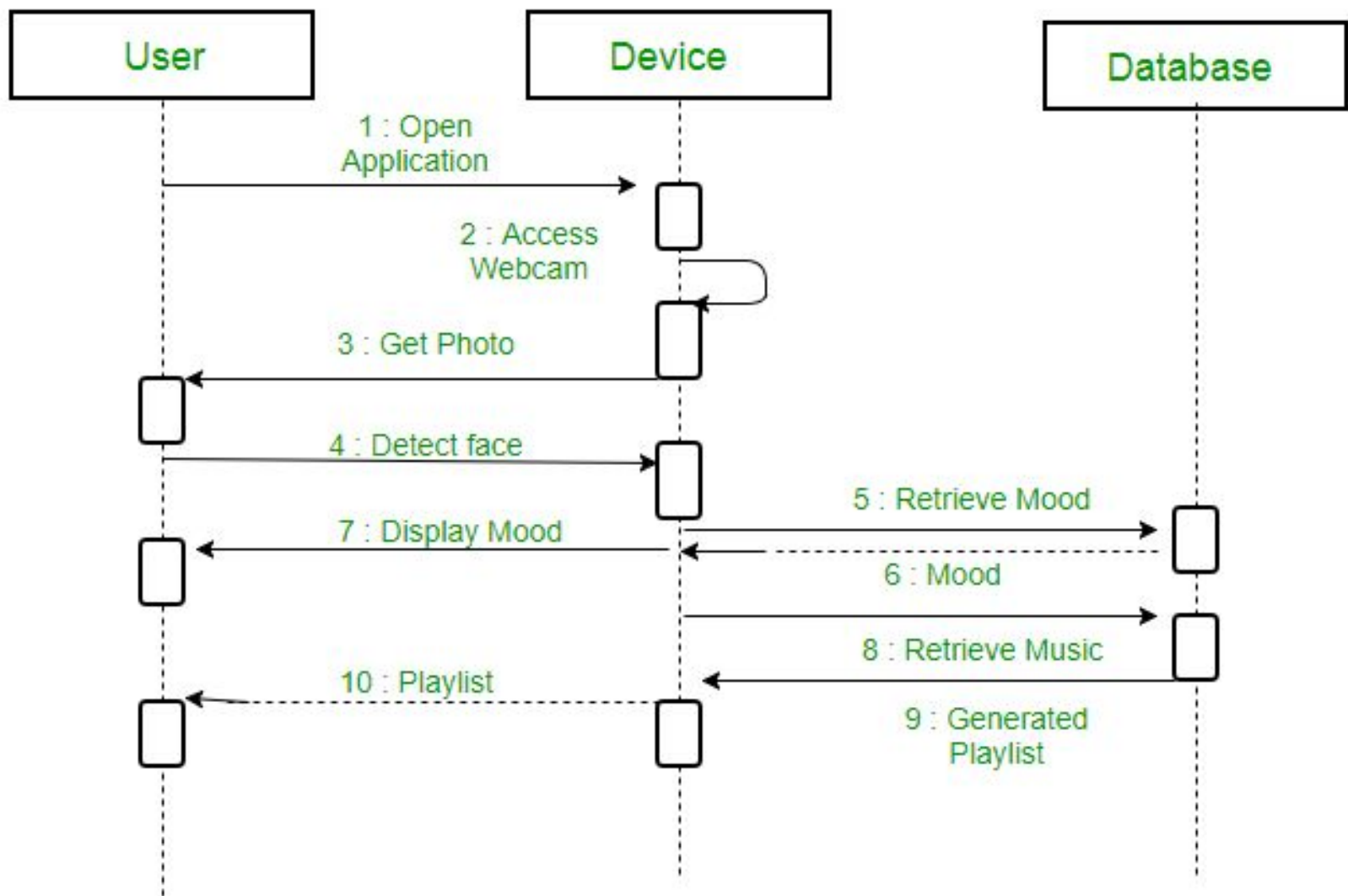


Figure : A sequence diagram for an emotion based music player.

online shopping

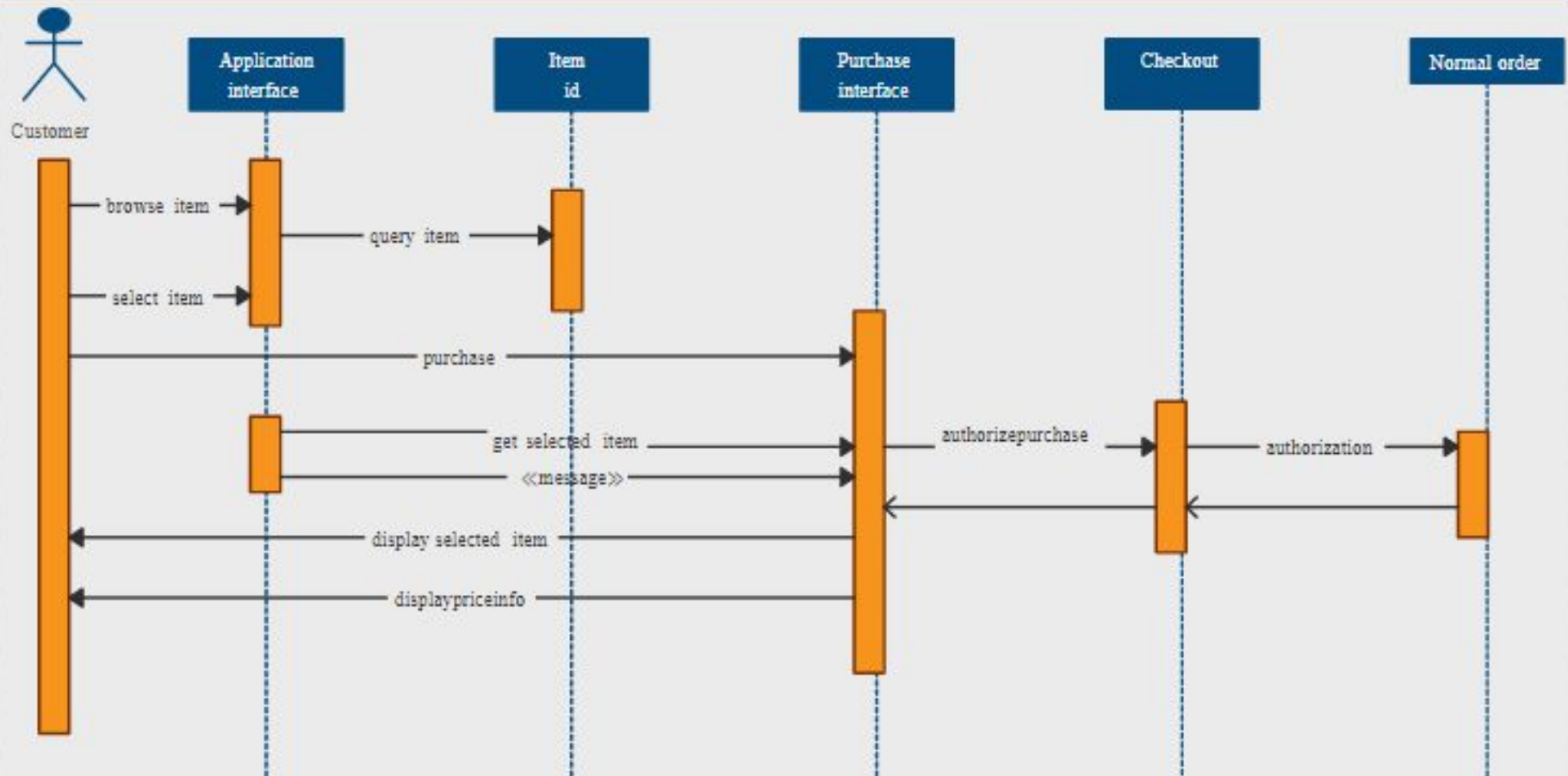


Fig: Sequence diagram for Online Shopping

Assignments(Not in syllabus)

1. Component diagram
2. Deployment Diagram
3. Collaboration Diagram
4. Activity Diagram

Analysis vs Design

- **Analysis** emphasizes an *investigation* of the problem and requirements, rather than a solution. For example, if a new online trading system is desired, how will it be used? What are its functions?
- "Analysis" is a broad term, best qualified, as in *requirements analysis* (an investigation of the requirements) or *object-oriented analysis* (an investigation of the domain objects).

- **Design** emphasizes a *conceptual solution* (in software and hardware) that fulfills the requirements, rather than its implementation. For example, a description of a database schema and software objects. Design ideas often exclude low-level or "obvious" details—obvious to the intended consumers.
- Ultimately, designs can be implemented, and the implementation (such as code) expresses the true and complete realized design.

- As with analysis, the term is best qualified, as in *object-oriented design* or *database design*.
- Useful analysis and design have been summarized in the phrase *do the right thing (analysis), and do the thing right (design)*.