Object Modelling using UML

Courtesy:

Roger Pressman, Ian Sommerville &

Prof Rajib Mall

System modeling:

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

System perspectives:

- An **external perspective**, where you model the context or environment of the system.
- An **interaction perspective**, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural perspective**, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral perspective**, where you model the dynamic behavior of the system and how it responds to events.

Design Approaches:

- Two fundamentally different software design approaches:
  - **Function-Oriented Design**
  - **Object-Oriented Design**
- These two design approaches are radically different.
  - However, are complementary rather than competing techniques.
- Each technique is applicable at different stages of the design process.

Use of Graphical Models:

- As a means of facilitating discussion about an existing or proposed system.
- As a way of documenting an existing system.
- As a detailed system description that can be used to generate a system implementation.

Unified Modeling Language (UML):

- The Unified Modelling Language is a set of 13 different diagram types that may be used to model software systems
- The main aim of UML is to define a standard way to visualize the way a system has been designed
  - Acts as a blueprints
  - A visual language
- UML helps software engineers, businessmen and system architects with modelling, design and analysis.

Need of UML?

- A clear and concise way to communicate among teams/collaborators.
- To communicate with non programmers.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.
- When you're writing code, there are thousands of lines in an application, and it's difficult to keep track of the relationships and hierarchies within a software system. UML diagrams divide that software system into components and subcomponents.

Need of UML?

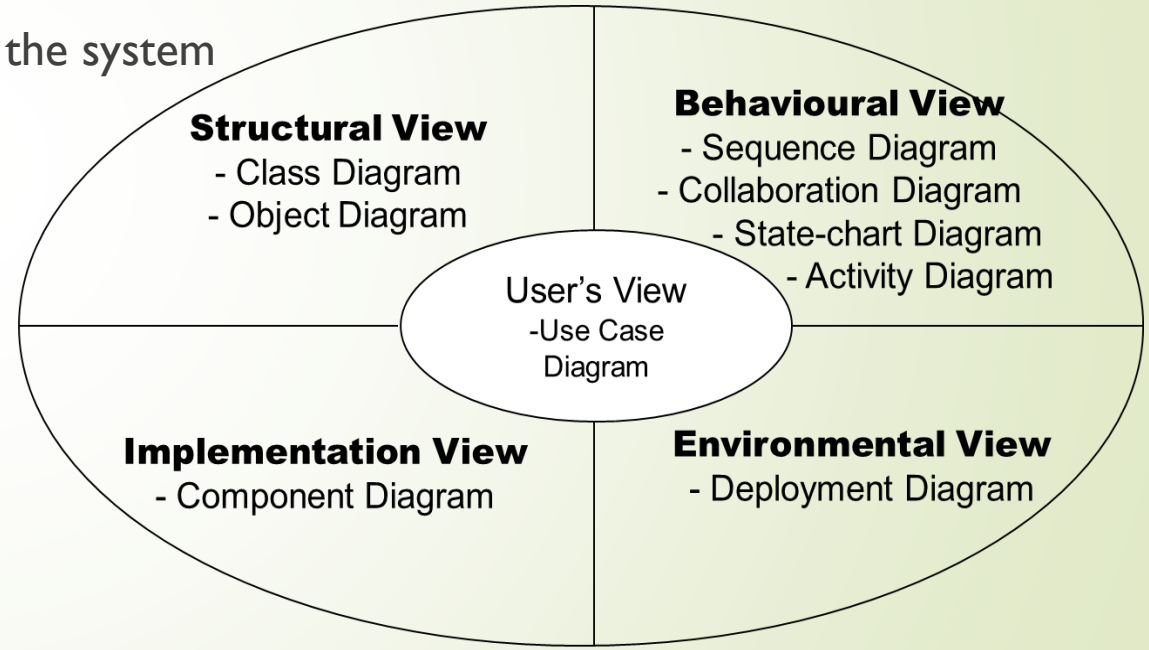
- UML is a modelling language
  - to document object-oriented analysis and design
  - it is independent of any specific design methodology
- Model is required to capture only important aspects
- UML a graphical modelling tool, easy to understand and construct
- Helps in managing complexity

UML Diagrams:

- UML is linked with object oriented design and analysis.
- Diagrams in UML can be broadly classified as:
  - **Structural Diagrams** – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
  - **Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

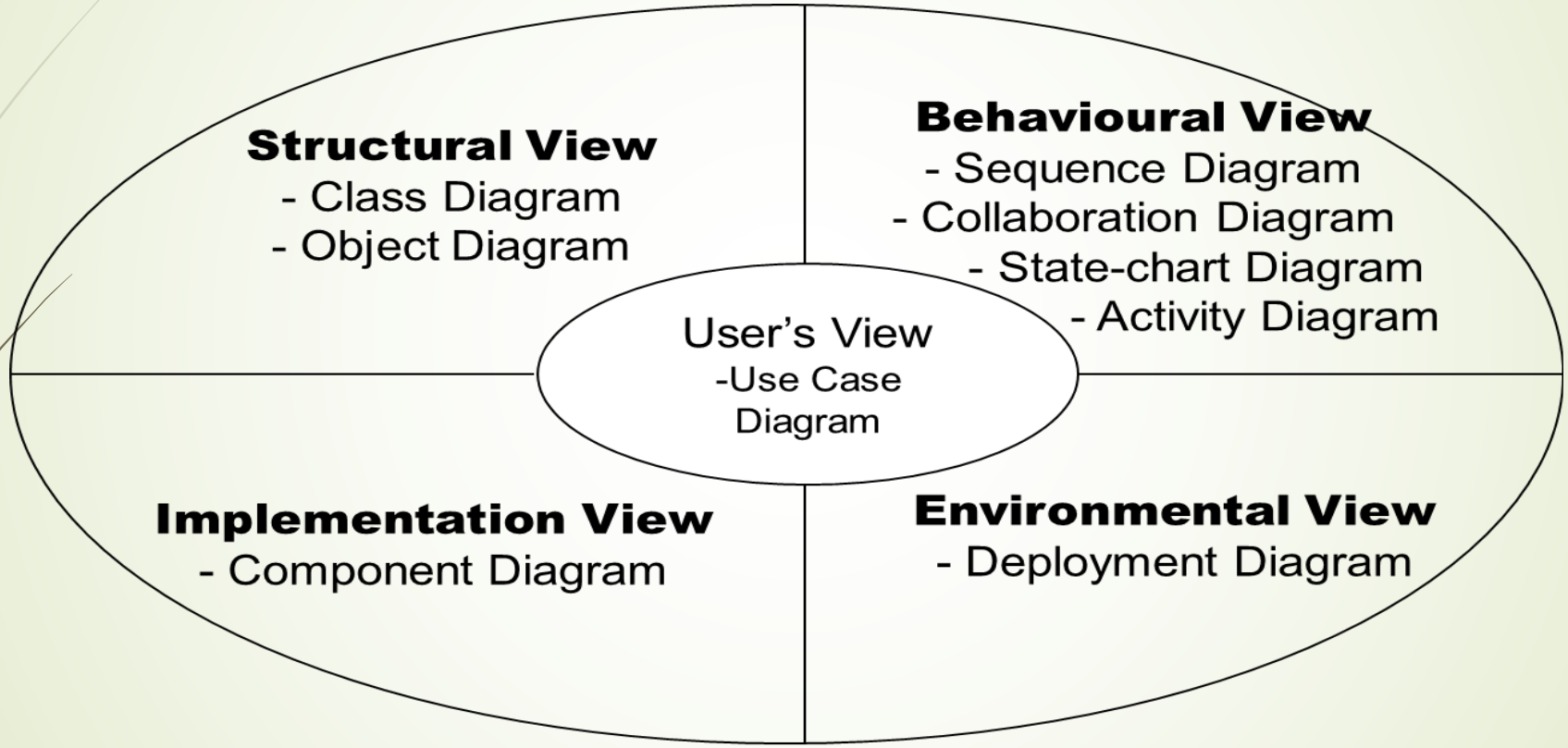
UML's Nine diagrams

- Nine diagrams
  - to capture different views of a system
  - Provide different perspectives of the software system
  - can be refined to get the actual implementation of the system
- Views of a system
  - User's view
  - Structural view
  - Behavioral view
  - Implementation view
  - Environmental view



Diagrams and views in UML

UML's Nine diagrams…



Diagrams and views in UML

Structural UML Diagrams:

- **Class Diagram**
  - It is the building block of all object oriented software systems.
  - We use class diagrams to depict the static structure of a system by showing system's classes, their methods and attributes.
  - Class diagrams also help us identify relationship between different classes or objects.
- **Object Diagram**
  - To study the behavior of the system at a particular instant
- **Component Diagram**
  - Component diagrams are used to represent the how the physical components in a system have been organized.
- **Deployment Diagram**

Behavior Diagrams:

- **State Machine Diagrams-** represent the condition of the system or part of the system at finite instances of time.
- **Activity Diagrams –** illustrate the flow of control in a system.
- **Use Case Diagrams –** Use Case Diagrams are used to depict the functionality of a system or a part of a system.
- **Sequence Diagram –** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.
- **Communication Diagram –** A Communication Diagram(known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects.

Are all the views required ?

- NO
- Use case model, class diagram and one of the interaction diagram for a simple system
- State chart diagram in case of many state changes
- Deployment diagram in case of large number of hardware components.

- What minimal set of the UML diagrams could be sufficient to describe a system completely ?

UML Diagrams:

- Which of the following is not a Behavior diagram in UML**?**
  - (a) State Diagram
  - (b) Object Diagram
  - (c) Use case Diagram
  - (d) Sequence Diagram
  - (e) None of these

Use Case Diagram:

- **Use case diagrams** give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.
- A Use case Model
  - consists of set of "use cases"
  - It is an important analysis and design artifact
  - other models must confirm to this model
  - not really an object-oriented model
  - represents a functional or process model

What are use cases ?

- Use cases represent
  - different ways in which system can be used by the users
  - correspond to the high-level requirements
  - represent the high-level transactions between the user and the system
  - define **behavior without revealing internal structure** of system
  - a set of related scenarios tied together by a common goal

What are use cases ?

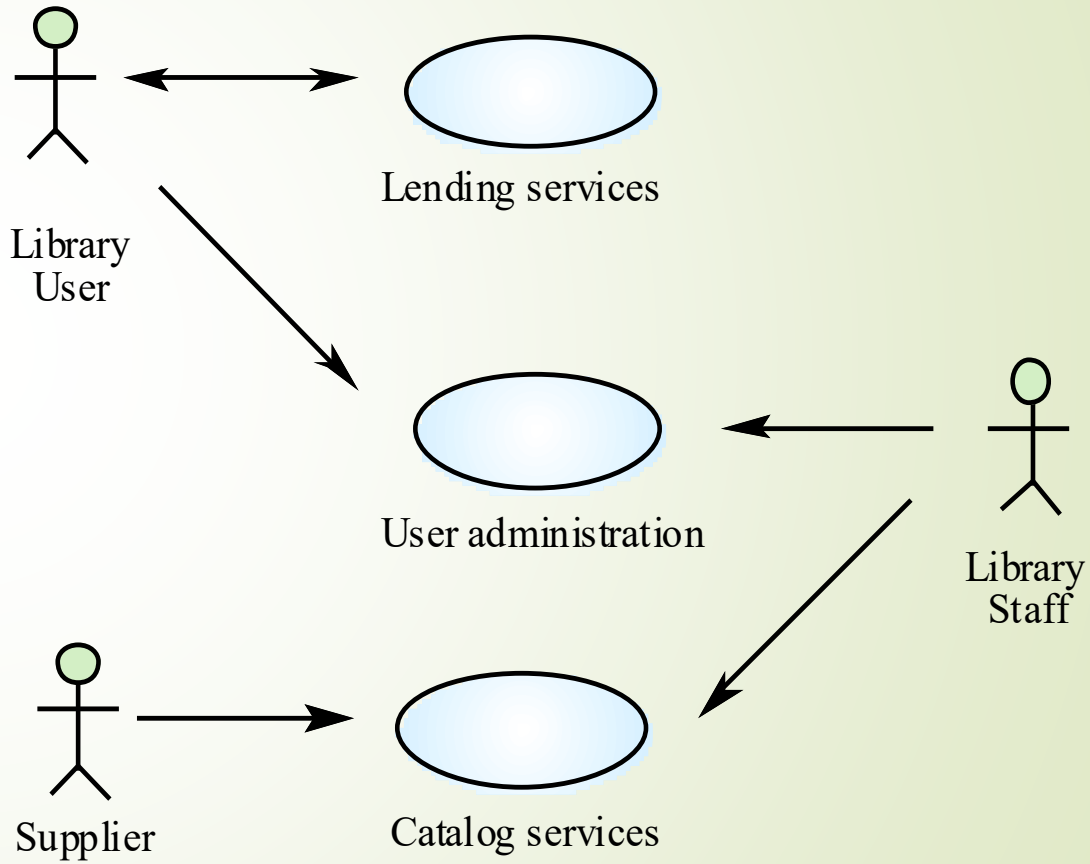
- def:
  - an interaction between the system & an external entity that has use for the external entity
  - drive all activities
  - start with analysis
  - used as a starting point in the requirements
  - drive acceptance tests

Usecase Diagram components

- Actor
- Use case
- Connector
- Boundary
- 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 case

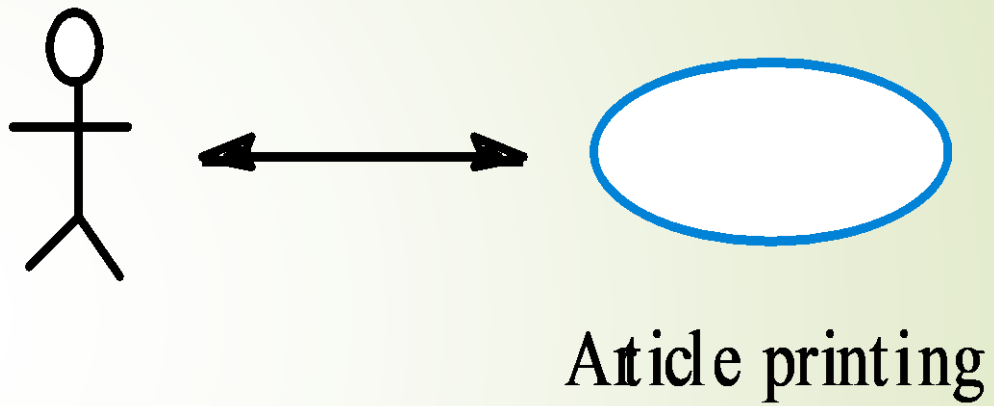
Usecase diagram components

- Usecase diagram consists of
  - actors and depict interactions amongst them to provide observable results.
    - Actors – stick figures
    - interaction – ellipses
- Use Case Diagram – context model
  - use actors and depict interactions amongst them to provide observable results.



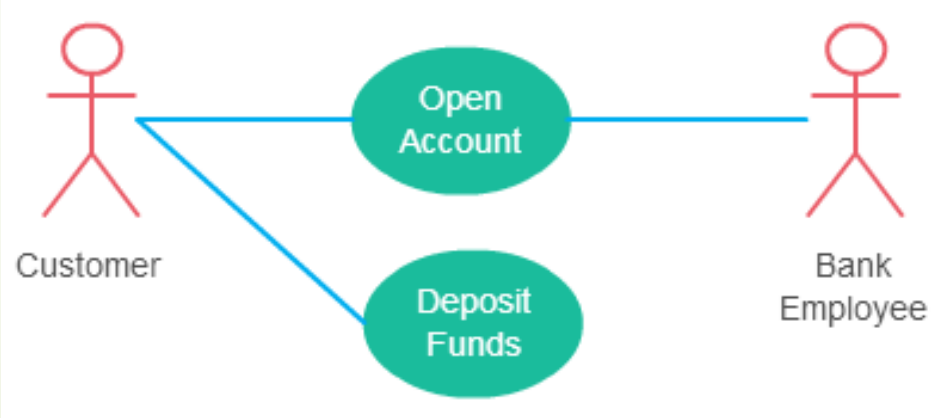
Usecase diagram components…

- Represented by use case diagram
- Use case is represented by ellipse
- System boundary is represented by rectangle
- Users are represented by stick person icon (actor)
- Communication relationship between actor and use case by line
- External system by stereotype



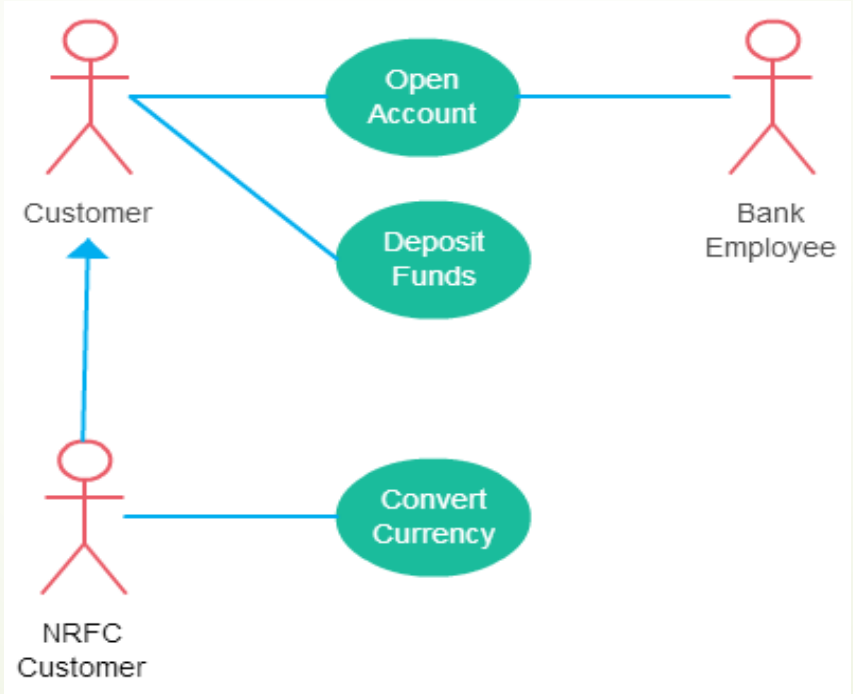
Association Between Actor and Use Case

- 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.



Generalization of an Actor:

- Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor. The descendant has one or more use cases that are specific to that role.



Use case diagram : Points to note

- Normally, use cases are independent of each other
- Implicit dependencies may exist
- Example:
  - In Library Automation System, renew-book & reserve-book are independent use cases.
  - But in actual implementation of renew-book, a check is made to see if any book has been reserved using reserve-book

An Example:

- For library information system, use cases are
  - issue-book
  - Query-book
  - Return-book
  - Create-member
  - Add-book, etc.

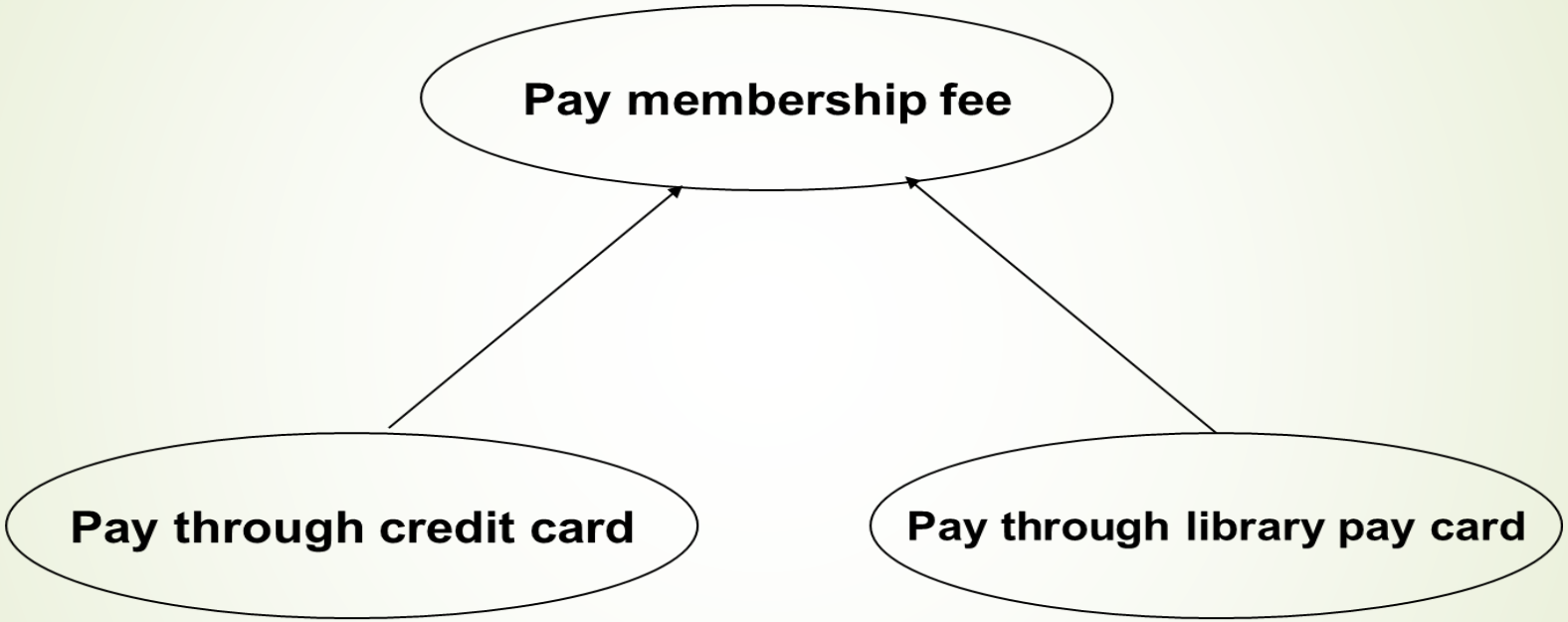
Factoring Use cases:

- Complex use cases need to be factored into simpler use cases
- Represent common behavior across different use cases
- Three ways of factoring
  - Generalization
  - Includes
  - Extends

Generalization of a Use Case

- This is similar to the generalization of an actor. The behavior of the ancestor is inherited by the descendant.
- This is used when there is common behavior between two use cases and also specialized behavior specific to each use case.

Factoring Using Generalization



Use case generalization

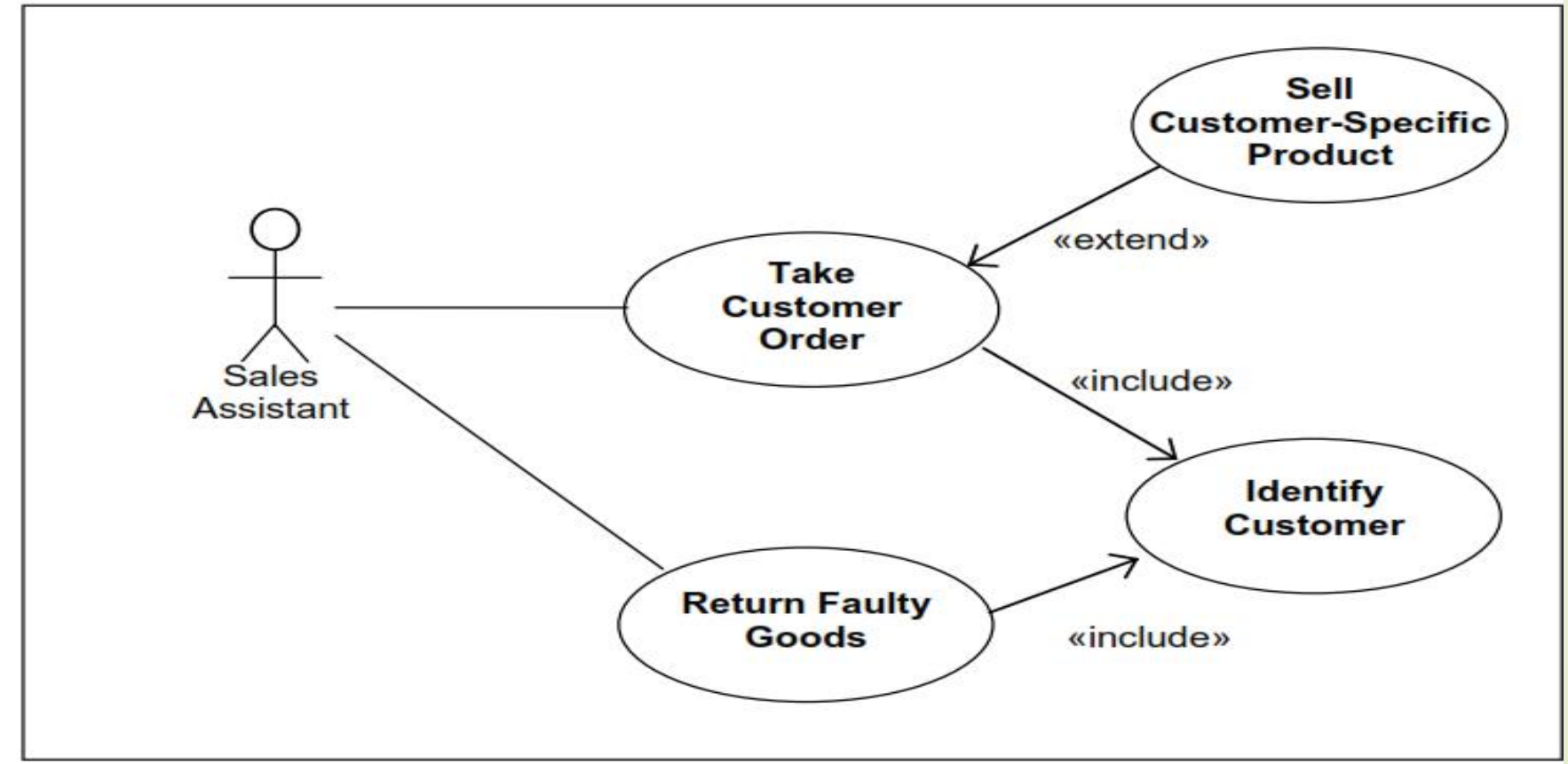
Extend Relationship Between Two Use Cases

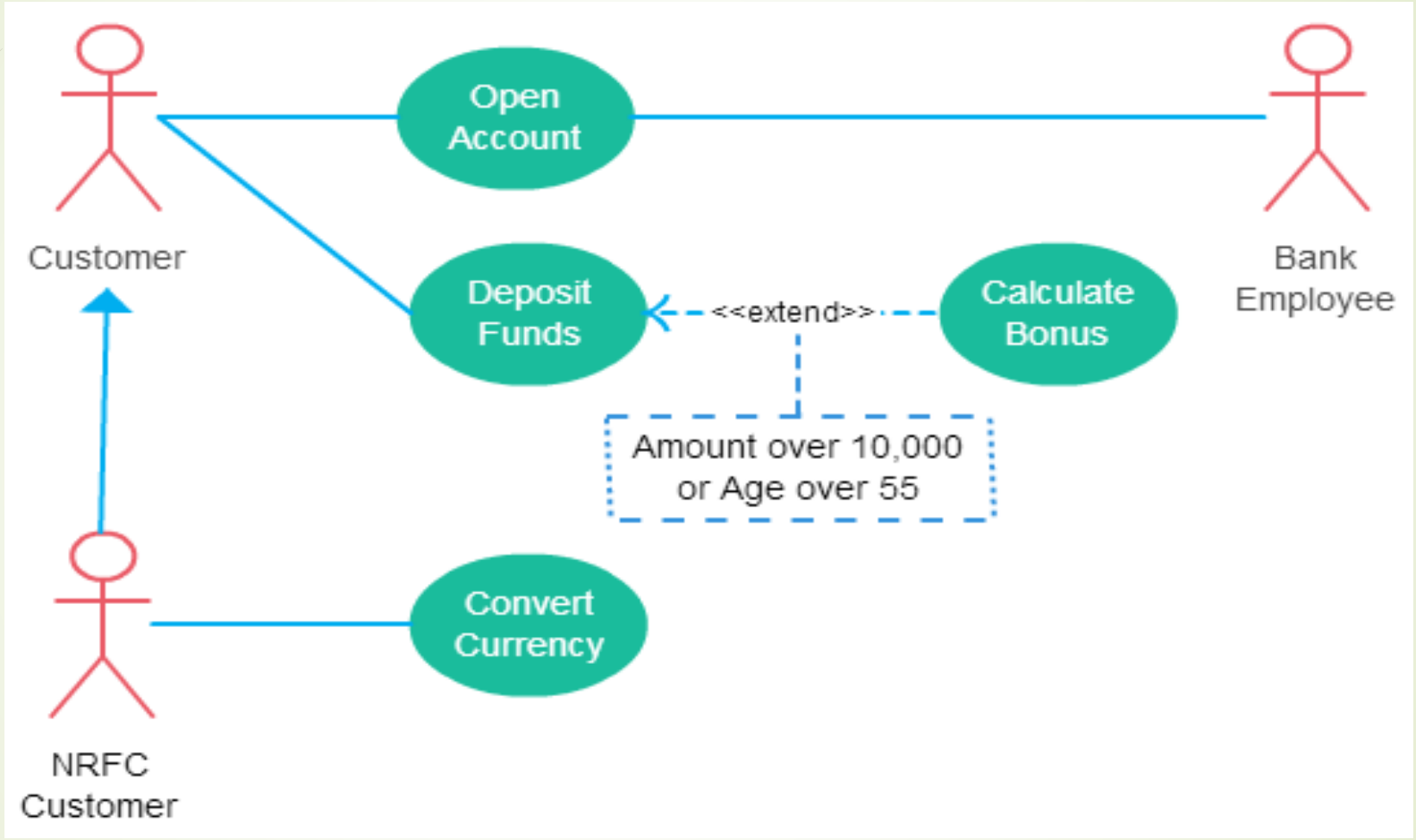
- It extends the base use case and adds more functionality to the system.
- **The extending use case is dependent on the extended (base) use case.** In the below diagram the "Calculate Bonus" use case doesn't make much sense without the "Deposit Funds" use case.
- **The extending use case is usually optional and can be triggered conditionally.** In the diagram, you can see that the extending use case is triggered only for 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 the extending use case.

Factoring using extends

Use case extension

Use of <<extends>> stereotype

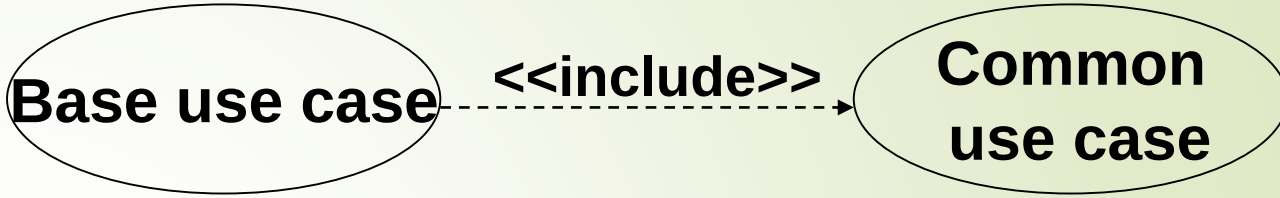



Customer
Open Account
Bank Employee
Deposit Funds
<<extend>>
Calculate Bonus
Amount over 10,000 or Age over 55
Convert Currency
NRFC Customer

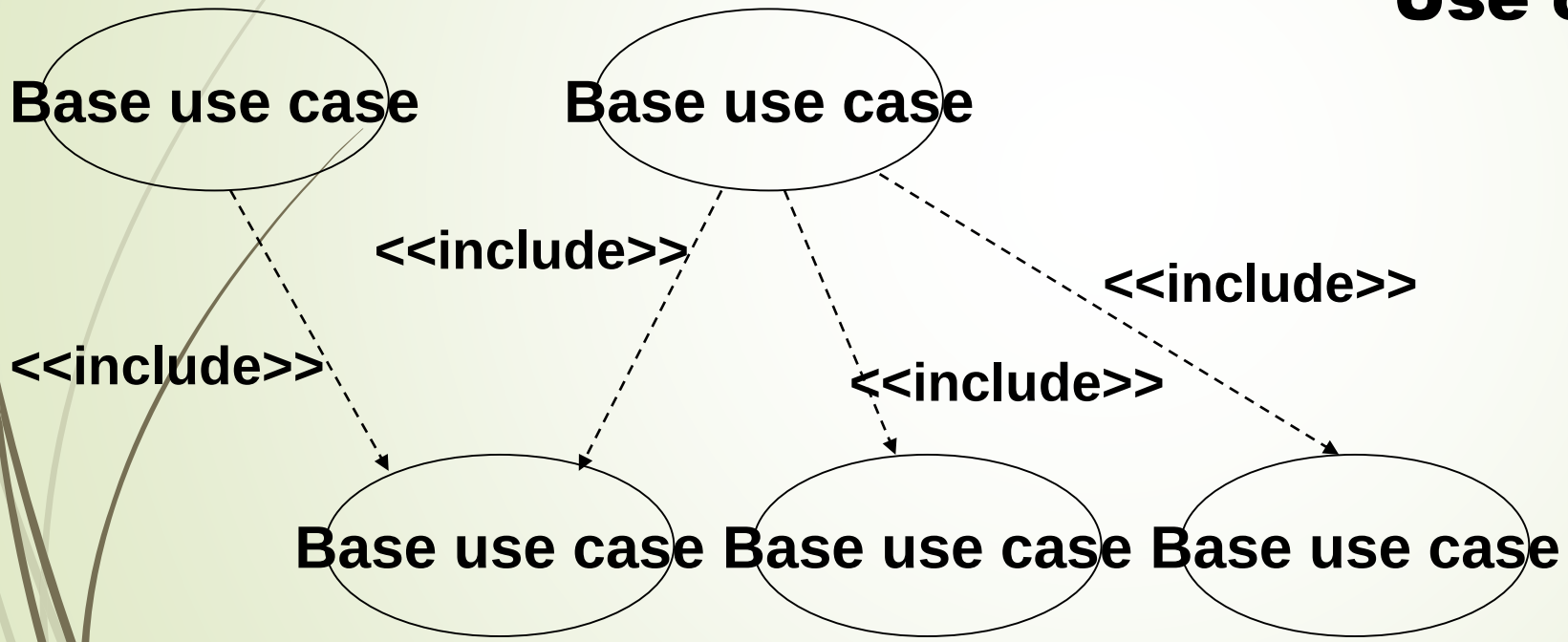
Include Relationship Between Two Use Cases

- The main reason for this is to reuse common actions across multiple use cases.
- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.

Factoring Using Includes

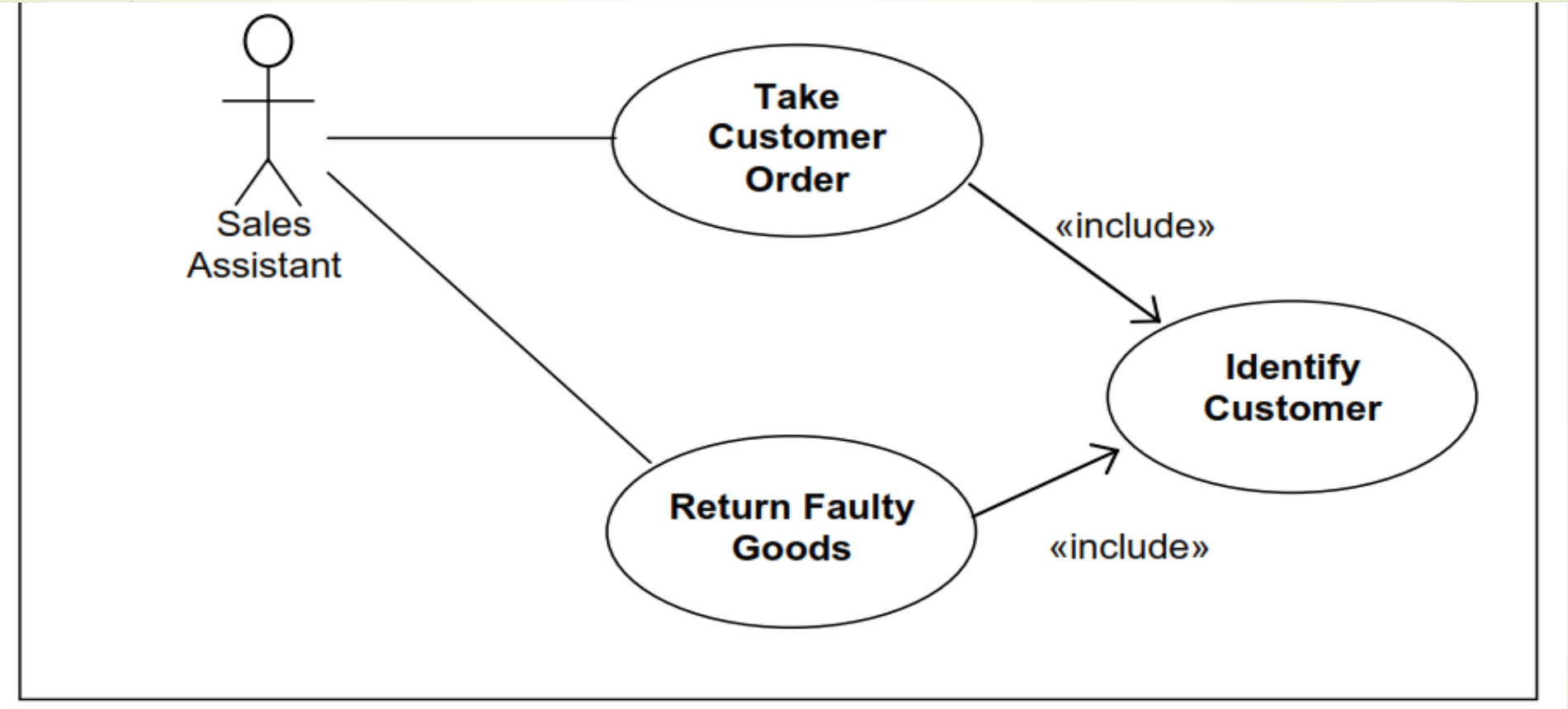


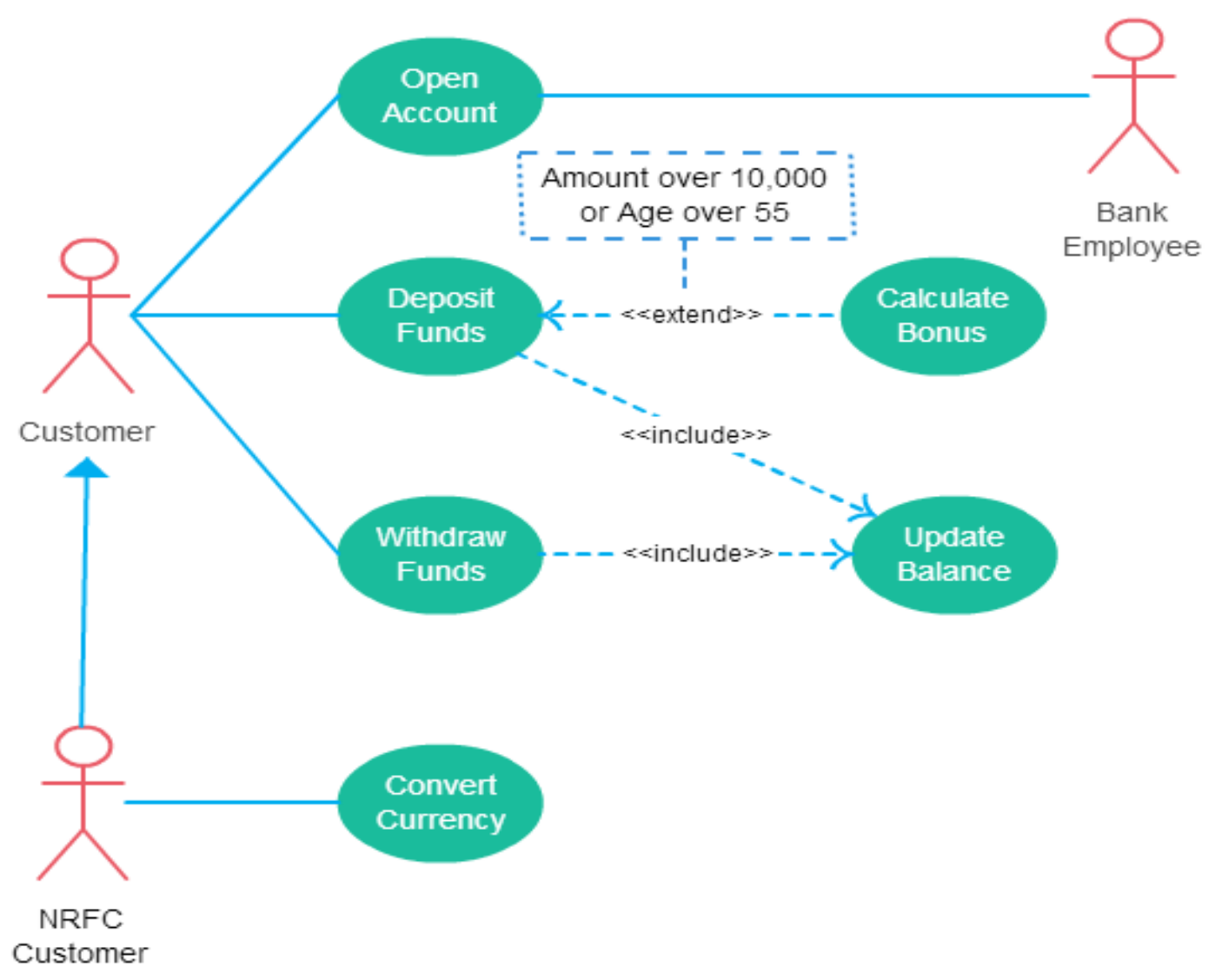
Use case inclusion



Paralleling model

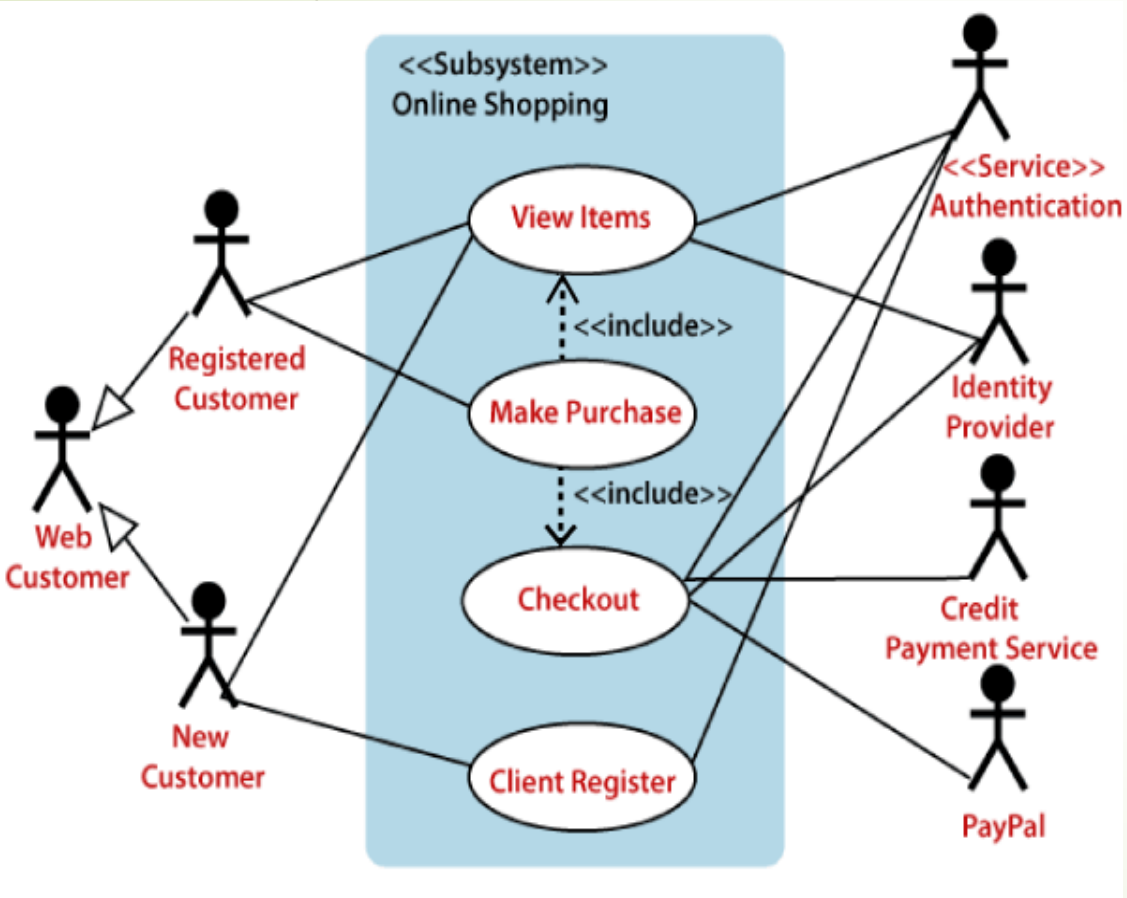
Use of <<includes>> stereotype



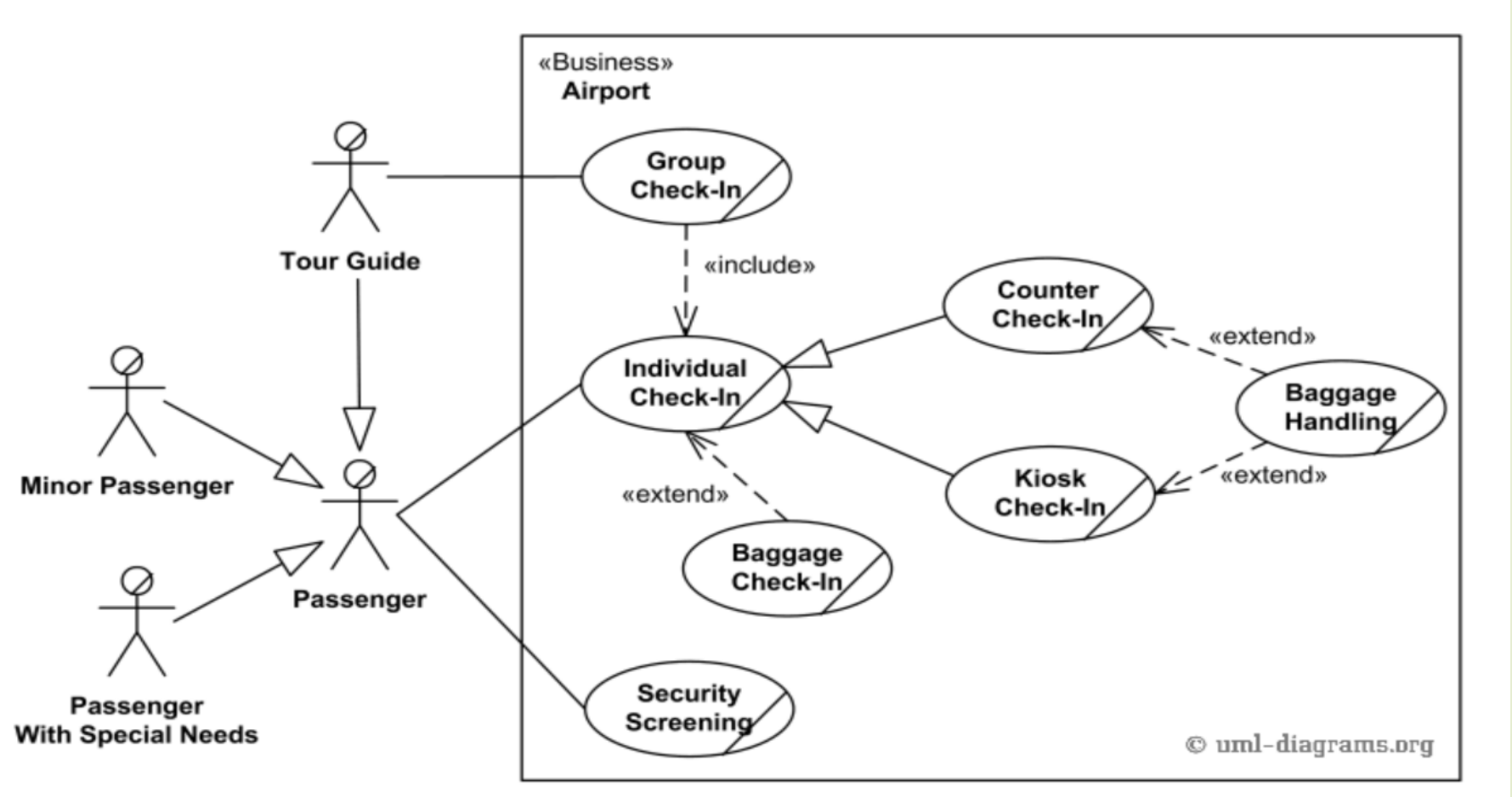

Open Account
Bank Employee
Amount over 10,000 or Age over 55
Customer
Deposit Funds
<<extend>>
Calculate Bonus
<<include>>
Withdraw Funds
<<include>>
Update Balance
Convert Currency
NRFC Customer

<<extends> vs <<includes>>

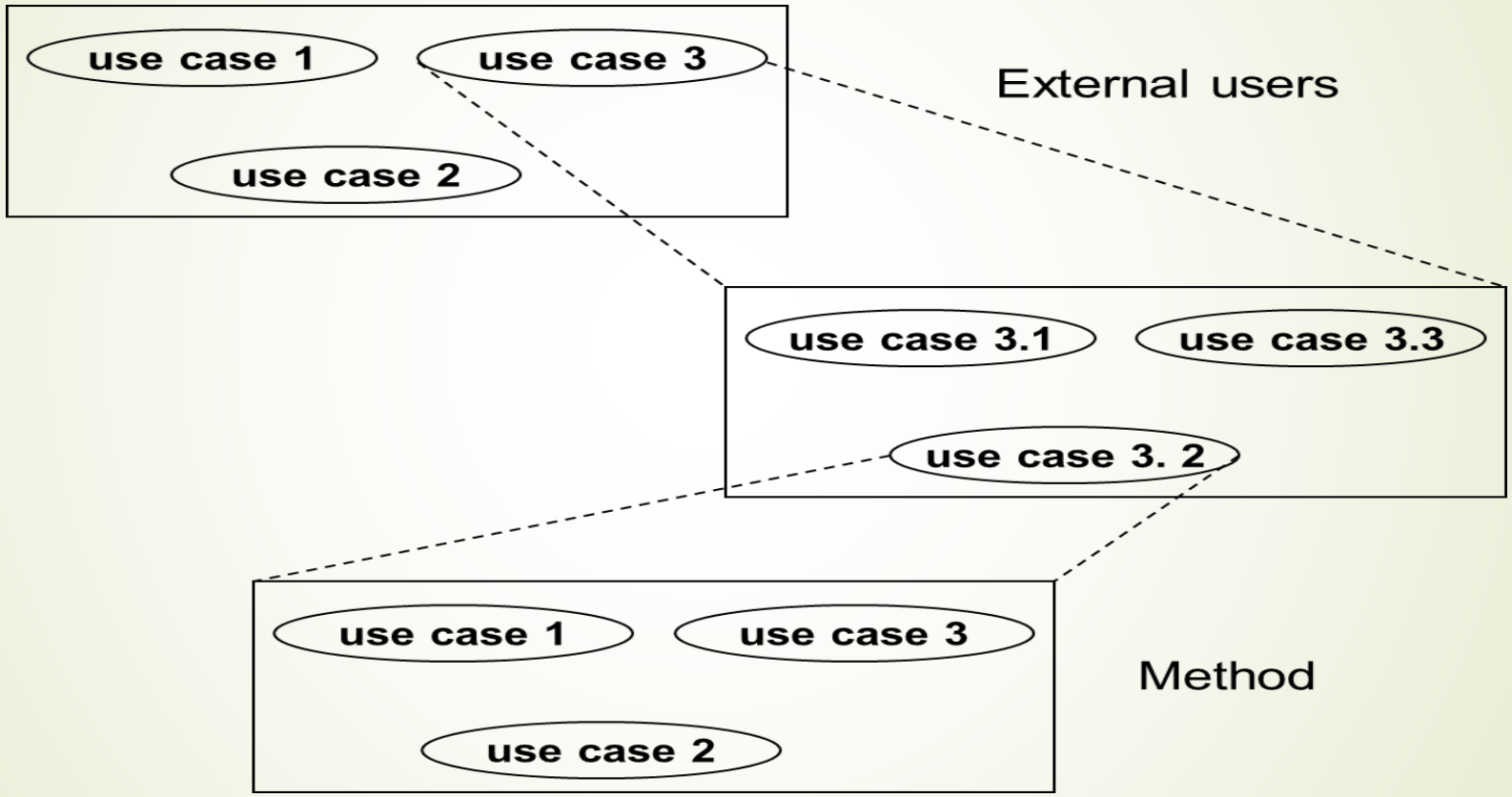
- When to use <<extends>> and when <<includes>> ?

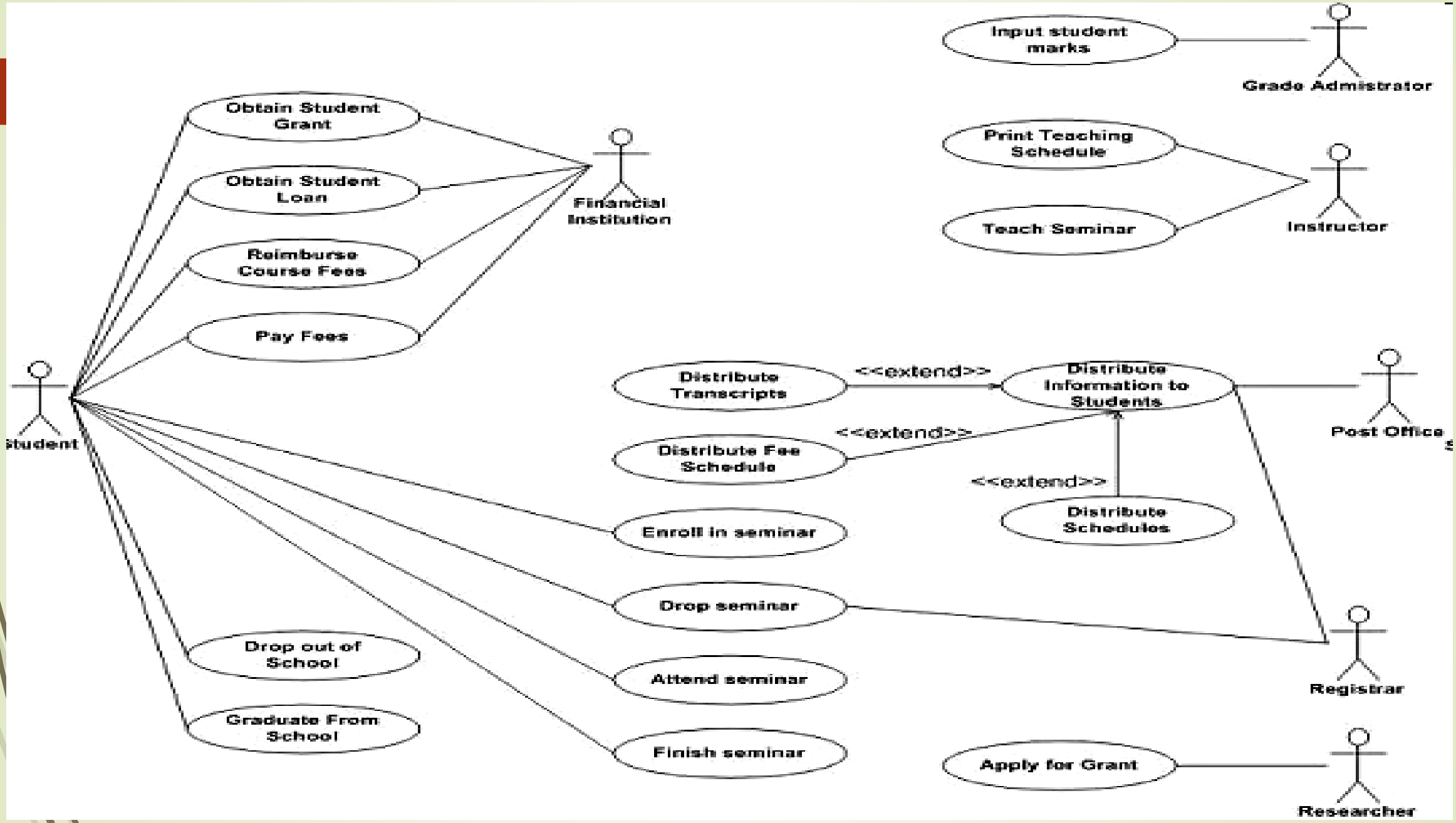


An example of use case diagram for airport check-in and security screening.

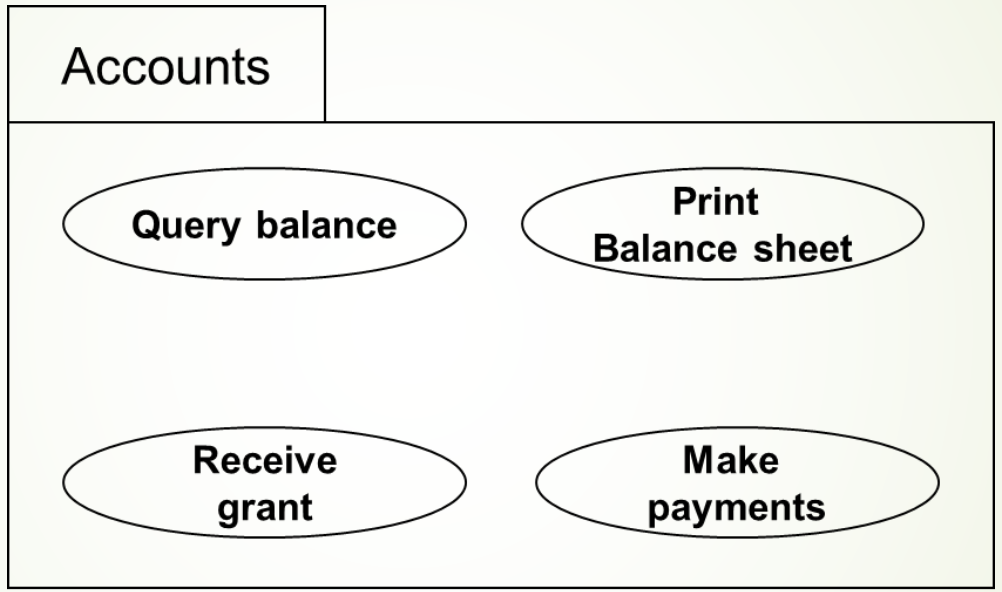


Hierarchical organization of use cases




Input student marks
Grade Administrator
Obtain Student Grant
Obtain Student Loan
Reimburse Course Fees
Pay Fees
Financial Institution
Print Teaching Schedule
Teach Seminar
Instructor
Distribute Transcripts
<<extend>>
Distribute Information to Students
Post Office
Distribute Fee Schedule
<<extend>>
<<extend>>
Distribute Schedules
Enroll in seminar
Drop seminar
Drop out of School
Attend seminar
Graduate From School
Finish seminar
Registrar
Apply for Grant
Researcher

Use case packaging:



Use case packaging

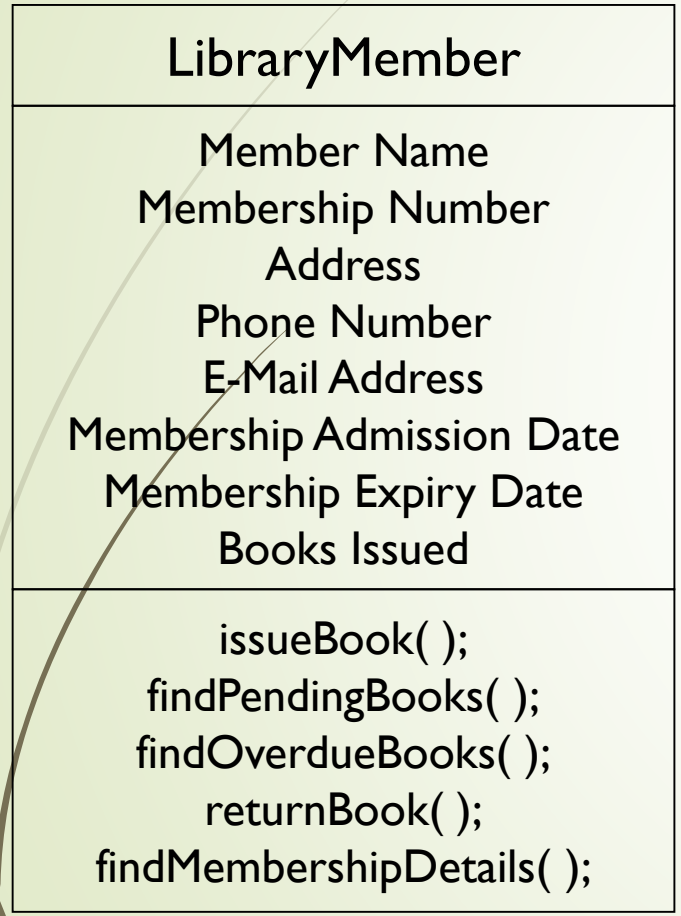
Class Diagram:

- Describes static structure of a system
- It shows how a system is structured rather than how it behaves.
- Main constituents are classes and their relationships:
  - Generalization
  - Association
  - Aggregation
  - Dependencies

Class Diagram:

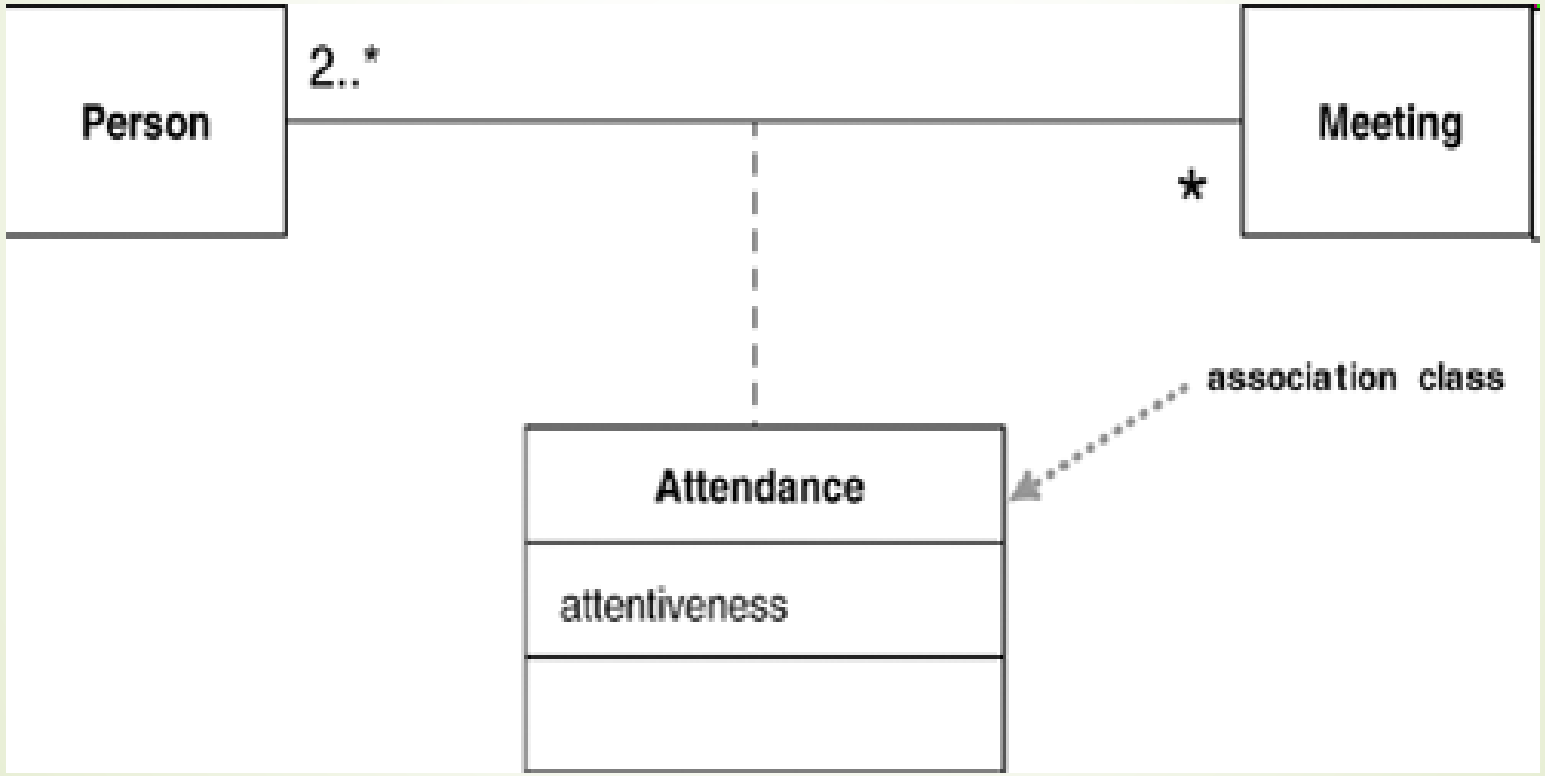
- Entities with common features, i.e. attributes and operations
- Classes are represented as solid outline rectangle with compartments
- Compartments for name, attributes & operations
- Attribute and operation compartment are optional for reuse purpose

Example of a Class diagram…



Different representations of the Library Member class

Class diagram



Association Relationships

- Enable objects to communicate with each other
- Usually binary but more classes can be involved
- Class can have relationship with itself (recursive association)
- Arrowhead used along with name, indicates direction of association
- Multiplicity indicates # of instances

- An association almost always implies that one object has – **owns** - the other object as a field/property/attribute i.e.
- Association -->
  A *has-a* C object (as a member variable)

Association between two classes

Many books may be borrowed by a LibraryMember and a book may be borrowed by exactly one member

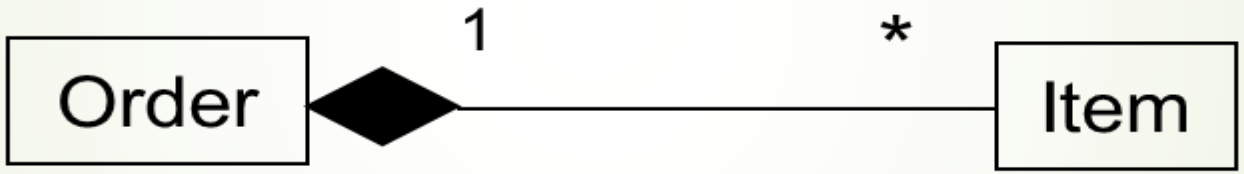
Aggregation Relationships

- Represent a whole-part relationship
- Represented by diamond symbol at the composite end
- Cannot be reflexive(i.e. recursive)
- Not symmetric
- It can be transitive

Representation of aggregation

Composition Relationship

- Stricter form of aggregation, in which the parts are existence-dependent on the whole.
- Life of item is same as the order

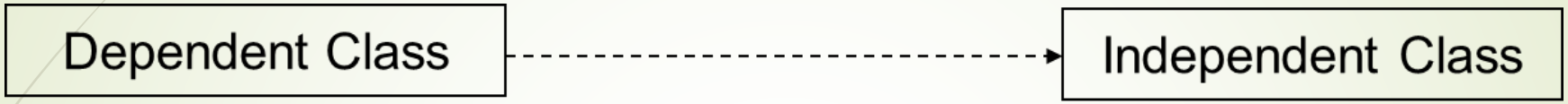


Representation of composition

Aggregation vs Composition Relationship

- Both aggregation and composition represents part/whole relationships.
- If components can dynamically be added to and removed from the aggregate, then the relationship is expressed as aggregation.
- If the components are not required to be dynamically added/delete, then the components have the same life time as the composite.

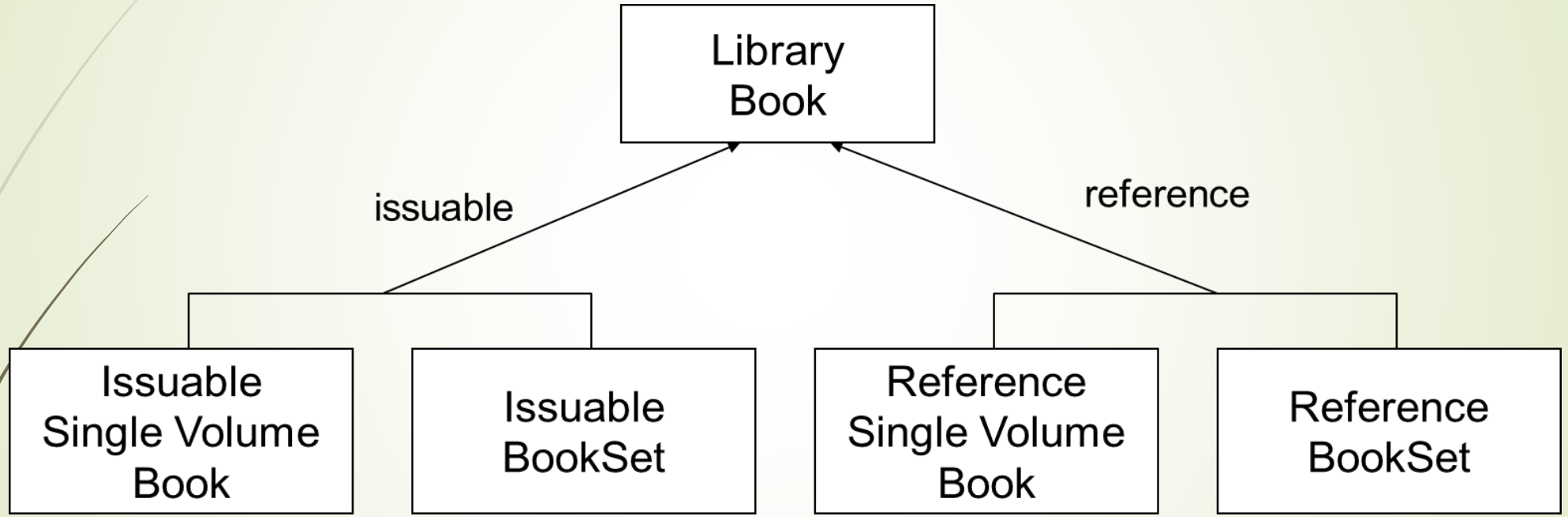
Class Dependency



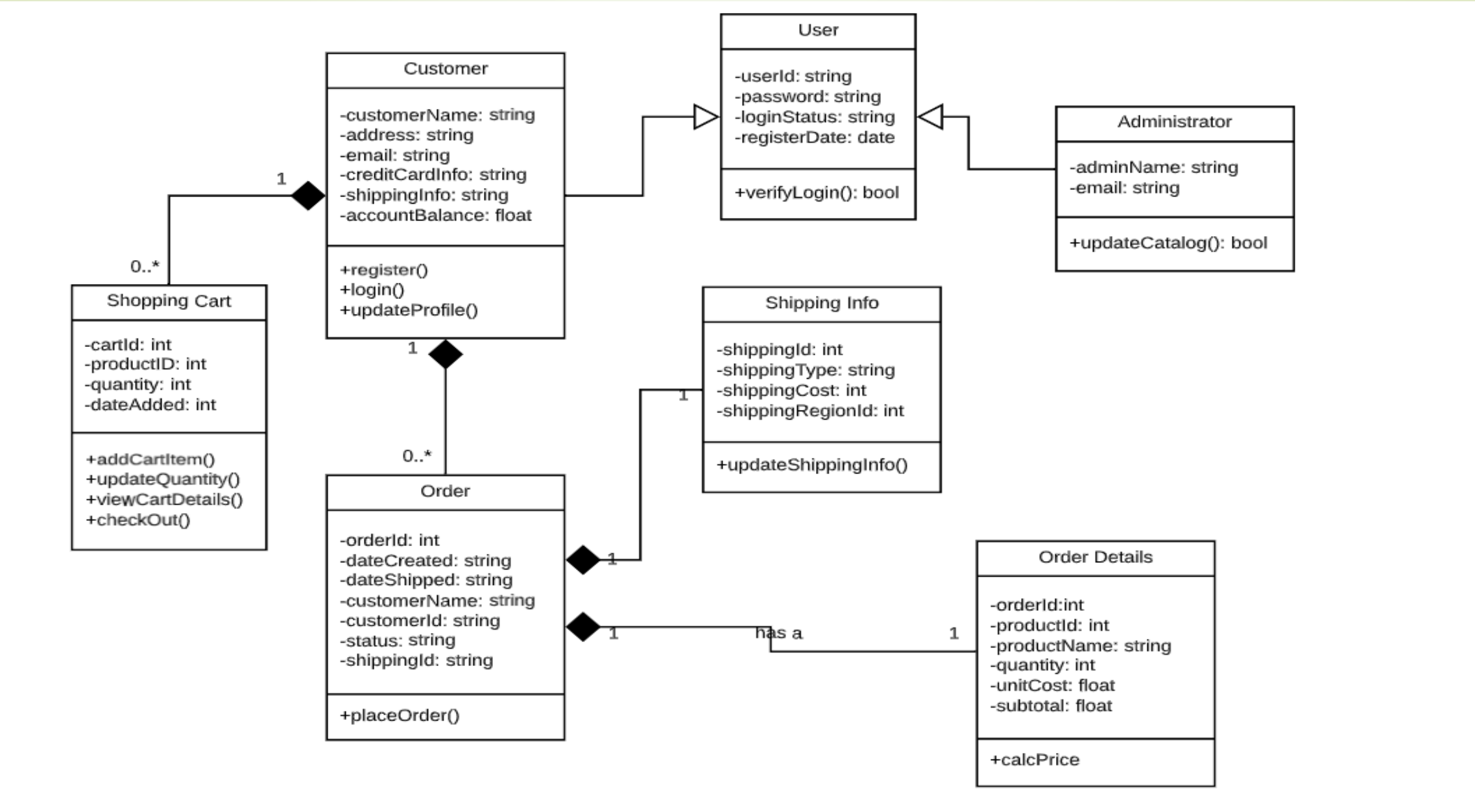
Representation of dependence between class

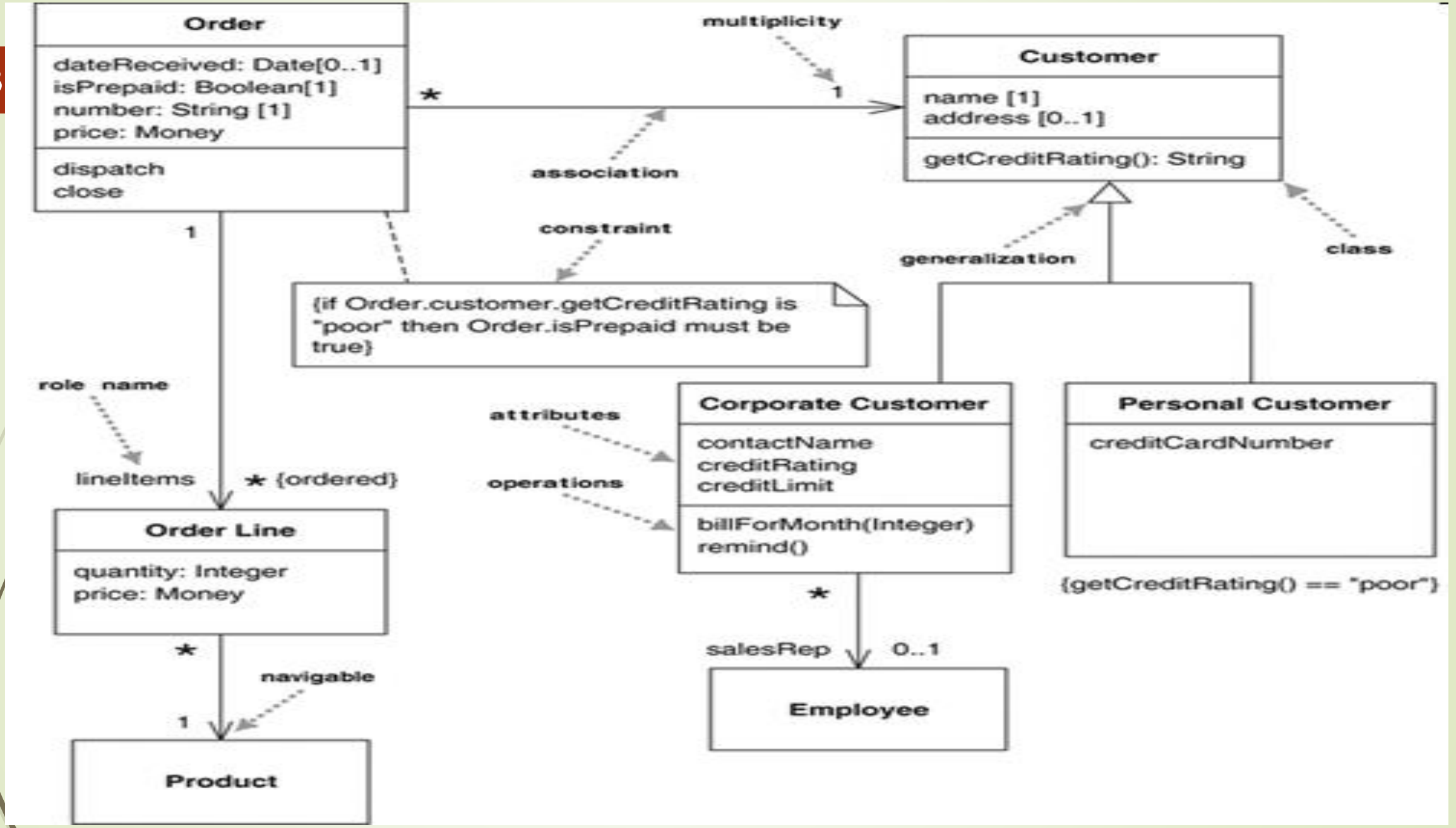
- Dependency is a weaker form of bond which indicates that one class depends on another because it uses it at some point in time.
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.

Inheritance Relationships



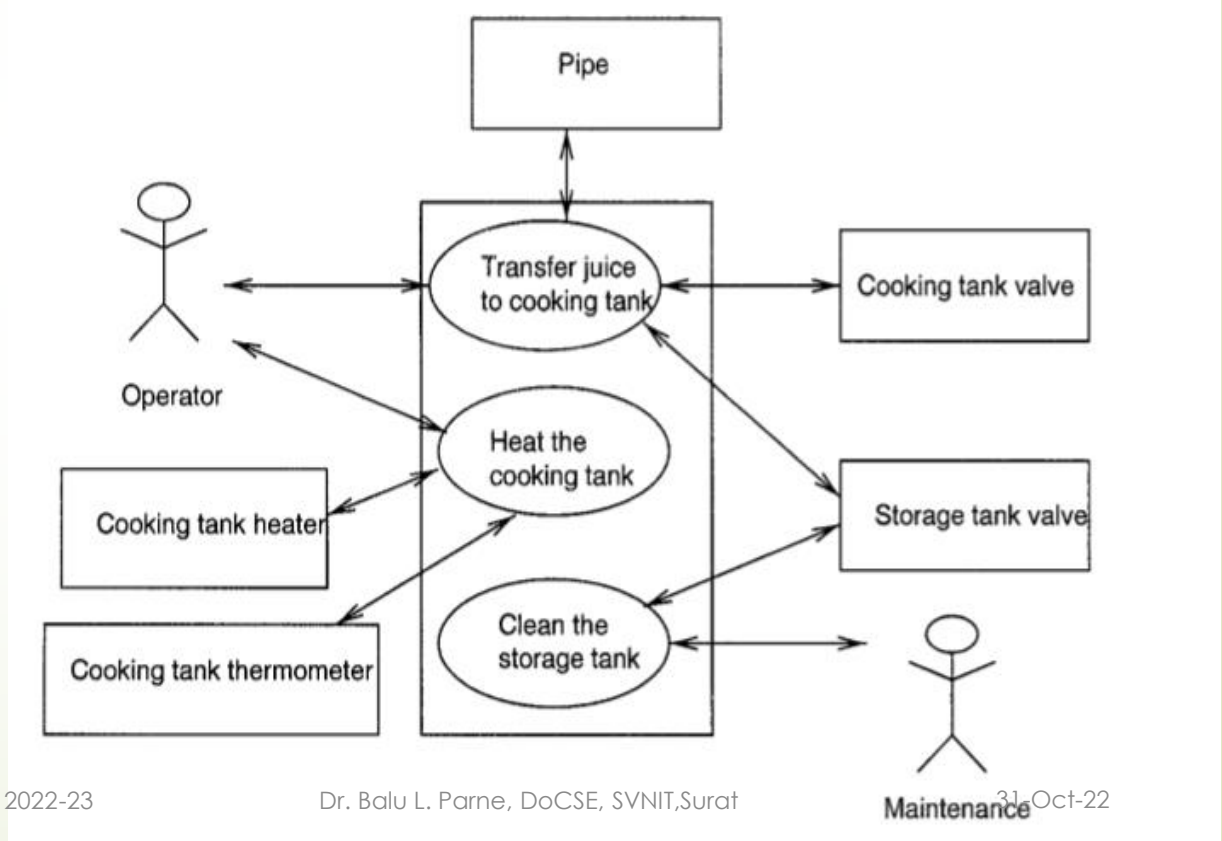
Representation of the inheritance relationship


User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
Administrator
-adminName: string
-email: string
+updateCatalog(): bool
1
0..*
Shopping Cart
-cartId: int
-productID: int
-quantity: int
-dateAdded: int
+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()
1
0..*
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
1
Order
-orderId: int
-dateCreated: string
-dateShipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
1
1
has a
1
Order Details
-orderId:int
-productId: int
-productName: string
-quantity: int
-unitCost: float
-subtotal: float
+calcPrice

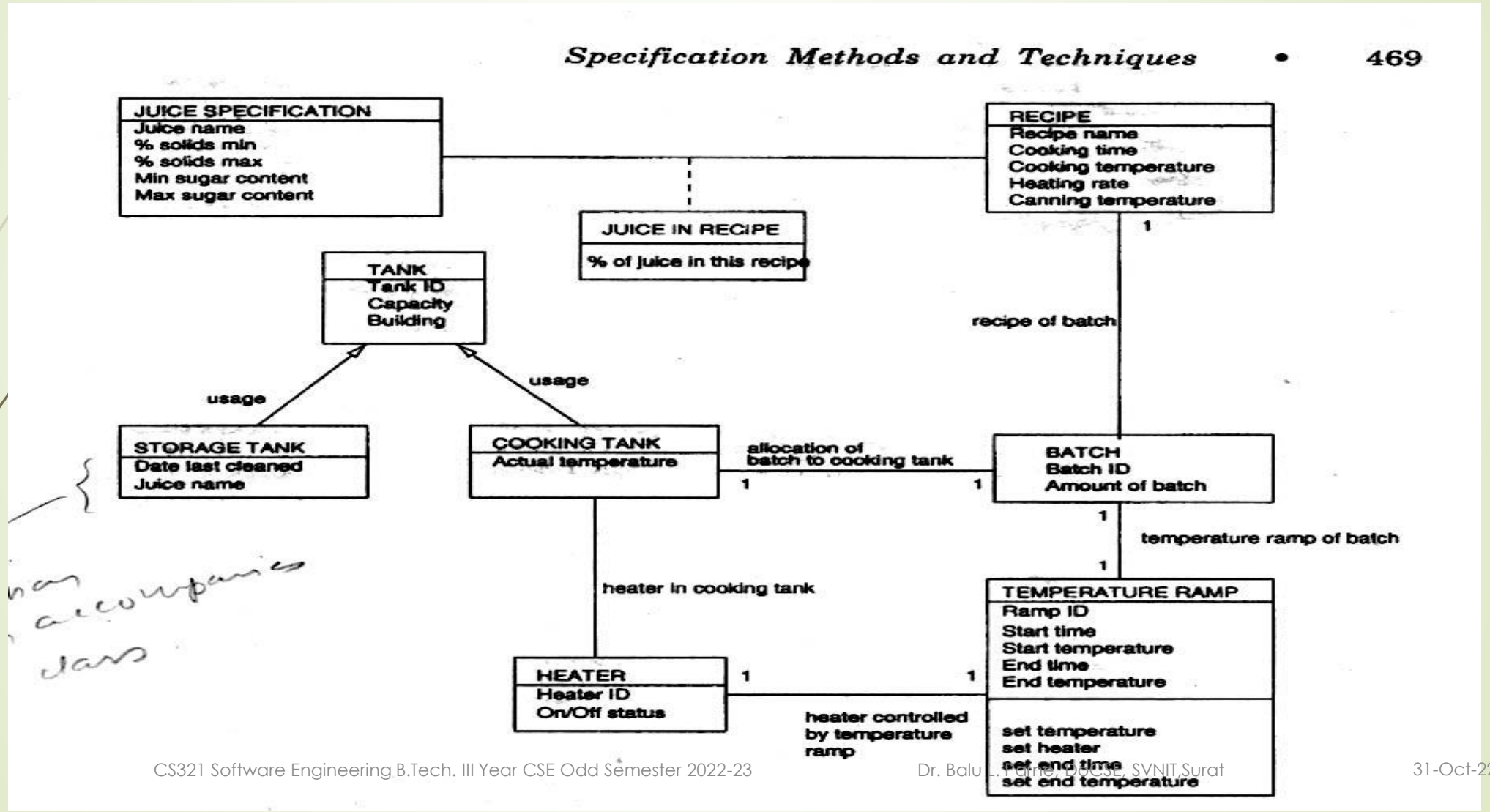

Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String [1]
price: Money
dispatch
close
multiplicity
1
*
Customer
name [1]
address [0..1]
getCreditRating(): String
association
constraint
generalization
class
1
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}
role name
lineItems
* {ordered}
attributes
operations
Corporate Customer
contactName
creditRating
creditLimit
billForMonth(Integer)
remind()
Personal Customer
creditCardNumber
{getCreditRating() == "poor"}
Order Line
quantity: Integer
price: Money
*
navigable
1
*
salesRep
0..1
Employee
Product

Use case diagram – juice plant system

- To design a Use case diagram for the juice plant system.
- What could be the actors **?**
- What could be some of the usecases **?**



Example of a Class diagram…



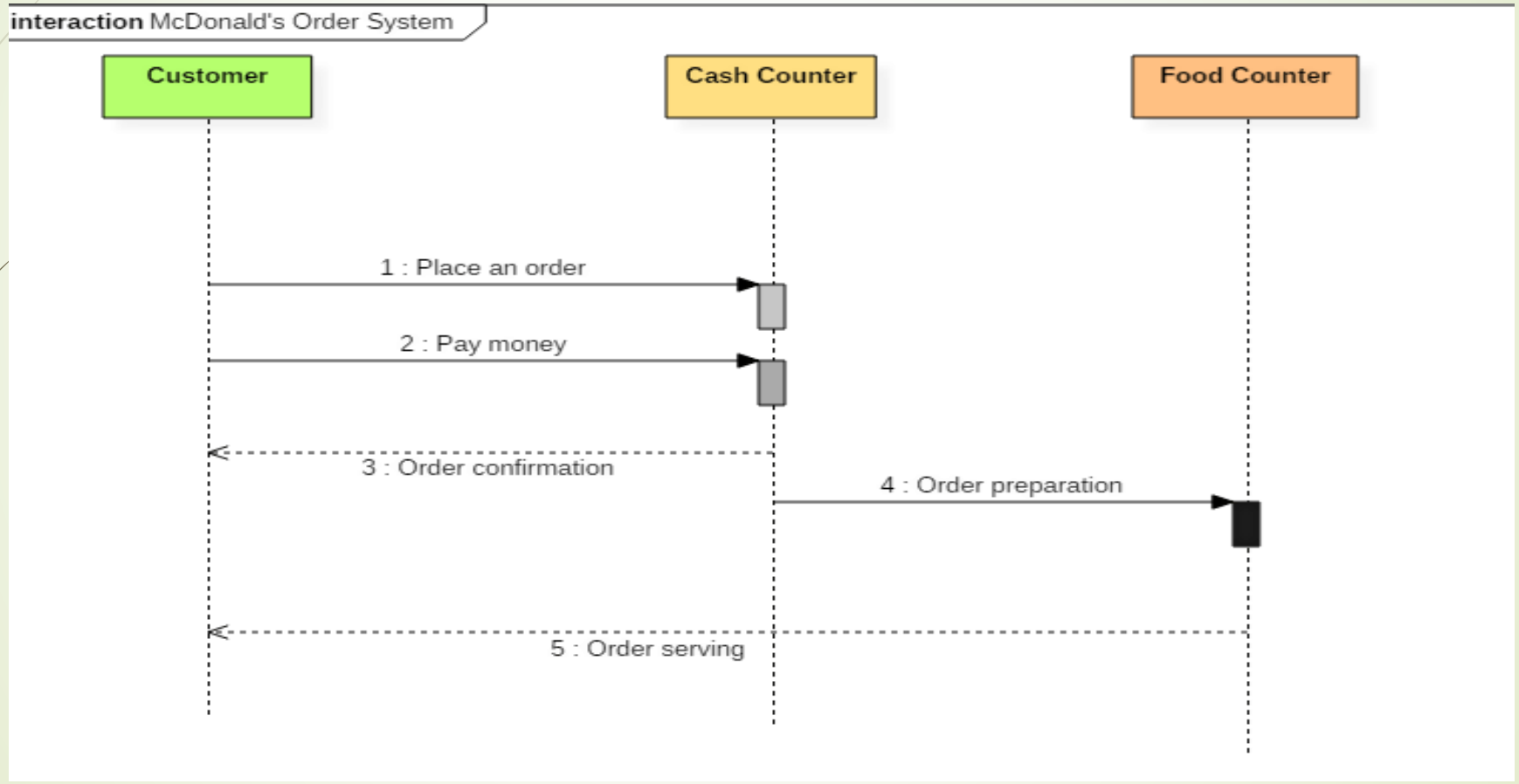
Interaction diagram

- Models how groups of objects collaborate to realize some behavior
- Typically each interaction diagram realizes behavior of a single use case
- Two kinds
  - Sequence &
  - Collaboration
- are equivalent but portrays different perspective
- these diagrams lay a very important role in the design process

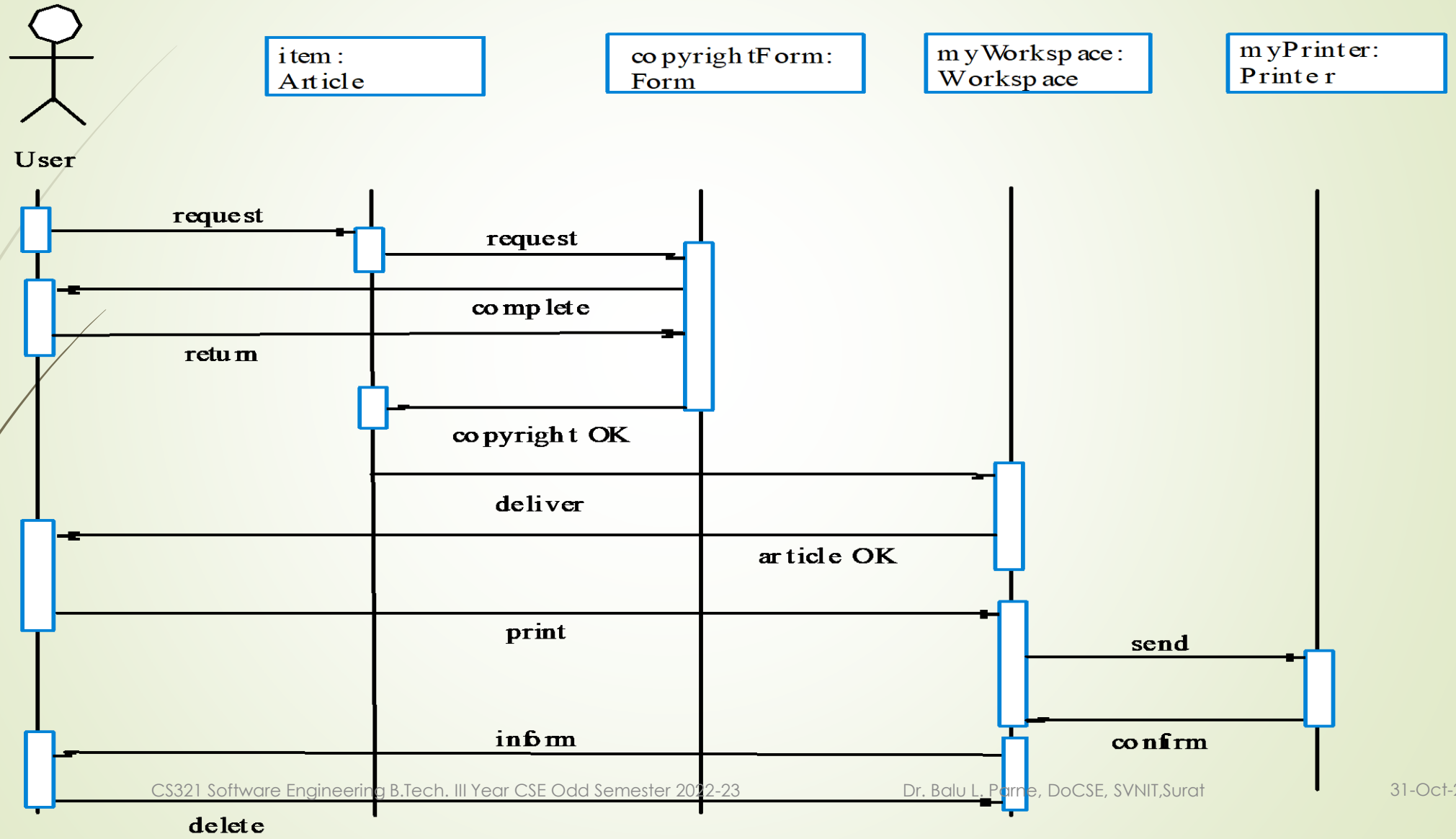
Sequence diagrams

- Sequence diagrams
  - show interaction among objects as two-dimensional chart
- Sequence diagrams may be used to add detail to use-cases. How **?**
  - describe how objects interact by exchanging messages
  - by providing a dynamic view
- Objects
  - are shown as boxes at top
  - if object created during execution then shown at appropriate place

Example of Sequence diagram



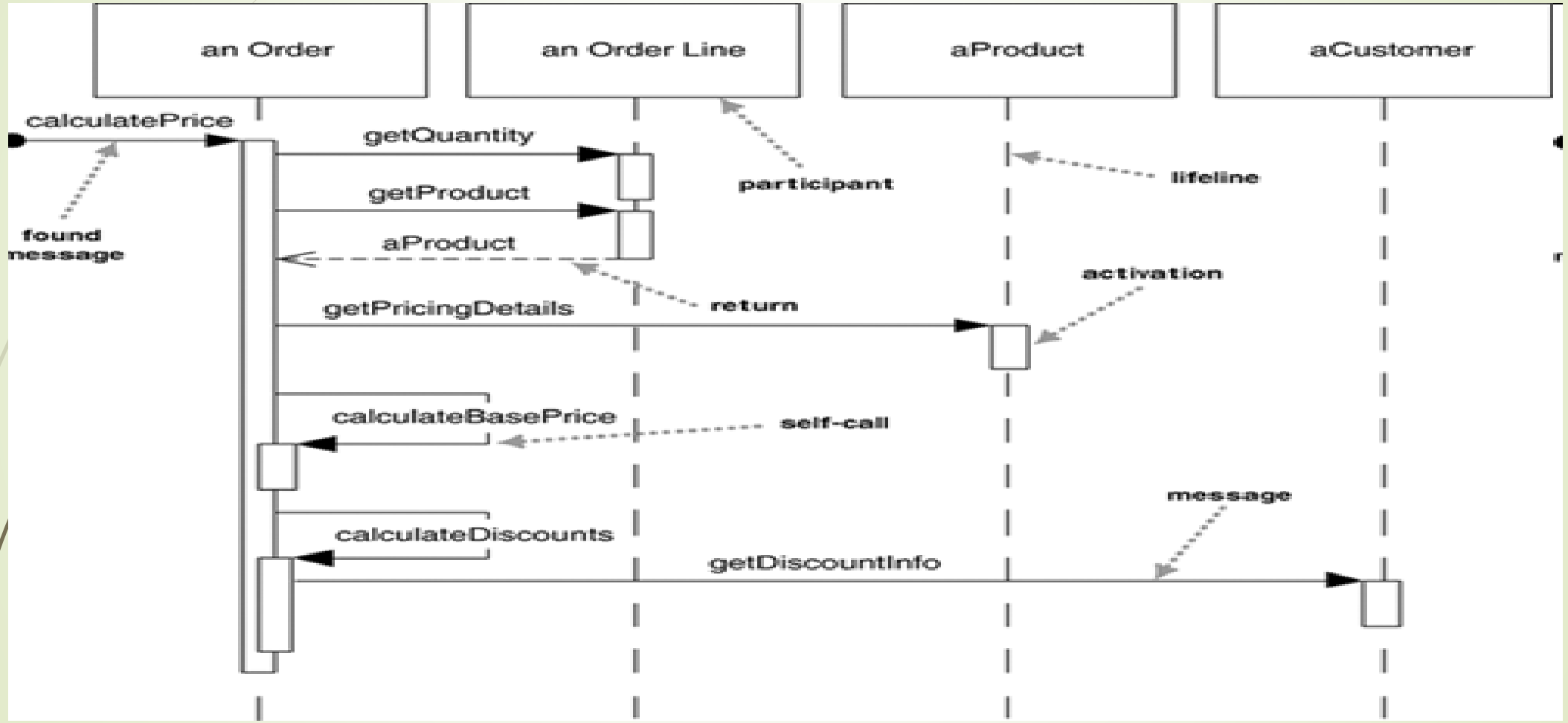
Sequence diagram: Print article sequence



Sequence diagrams – more details

- Objects
  - Objects existence are shown as dashed lines (lifeline)
  - Objects activeness, shown as rectangle on lifeline
  - Messages are shown as arrows
  - Message labelled with message name
  - Message can be labelled with control information
  - Two types of control information: condition ([]) & an iteration (*)

Sequence diagram

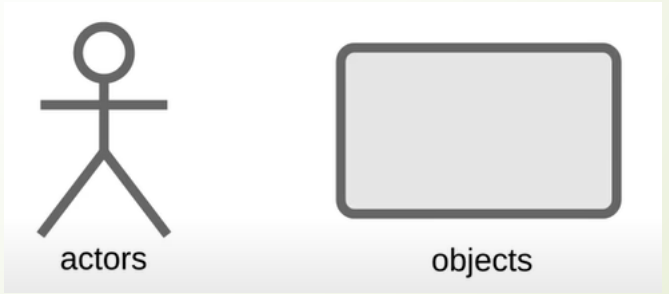


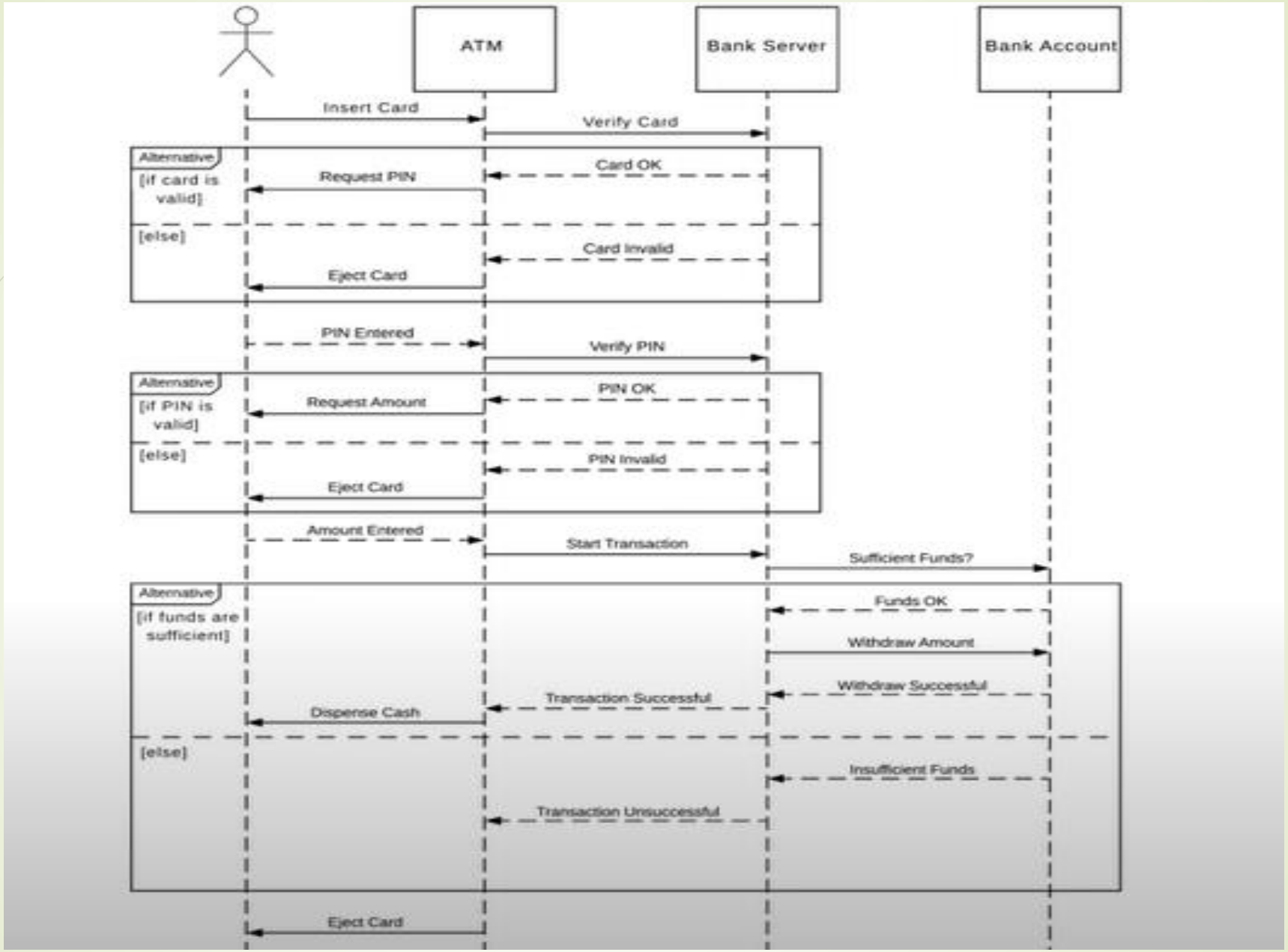
Ref: UML Distilled, Martin Fowler: Safari

Sequence Diagrams:

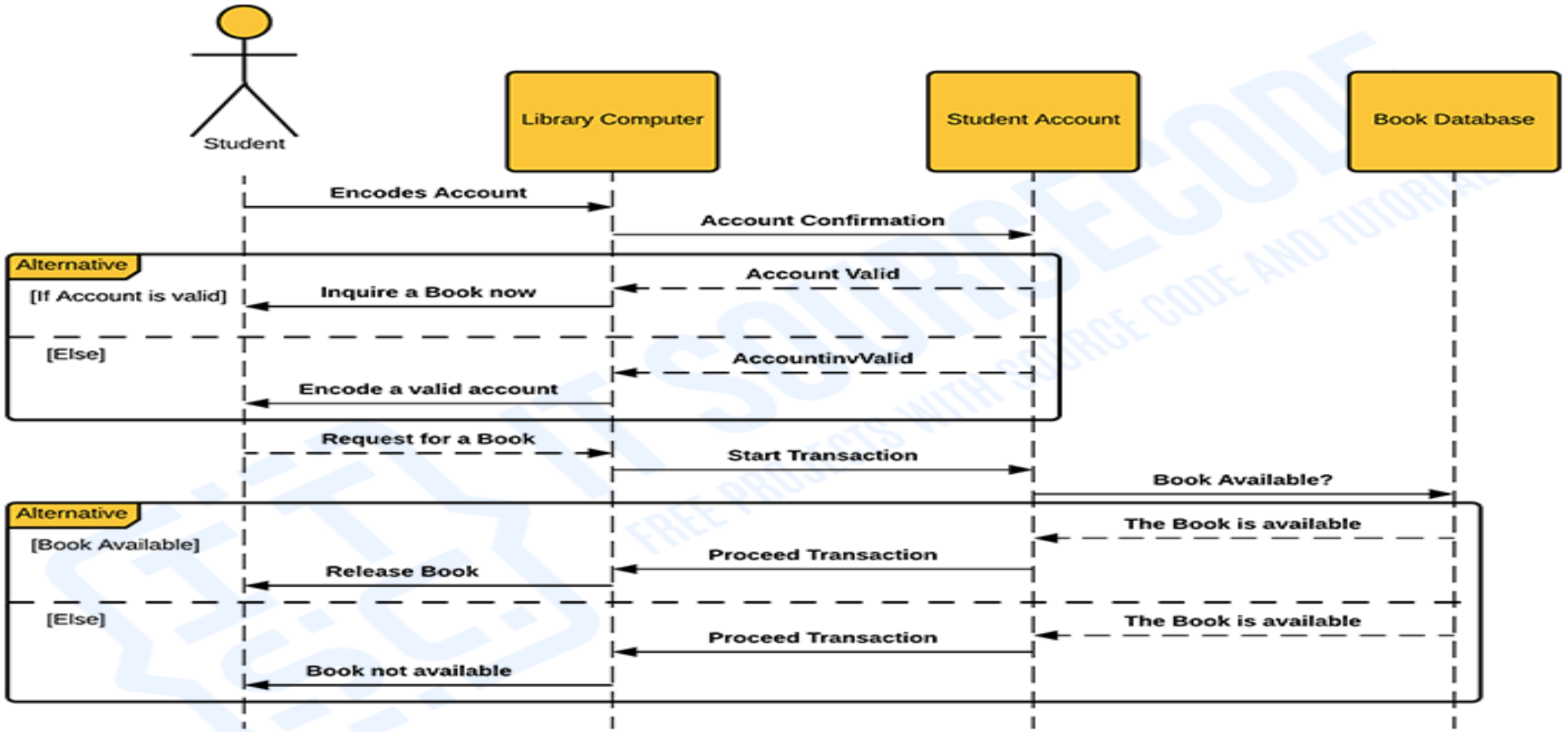
- Design a sequence diagram for “withdrawal” transaction for ATM application
- Design a sequence diagram for “book borrow” transaction.

Sequence Diagram:




ATM
Bank Server
Bank Account
Insert Card
Verify Card
Alternative
[if card is valid]
Request PIN
Card OK
[else]
Card Invalid
Eject Card
PIN Entered
Verify PIN
Alternative
[if PIN is valid]
Request Amount
PIN OK
[else]
PIN Invalid
Eject Card
Amount Entered
Start Transaction
Sufficient Funds?
Alternative
[if funds are sufficient]
Funds OK
Withdraw Amount
Withdraw Successful
Transaction Successful
Dispense Cash
[else]
Insufficient Funds
Transaction Unsuccessful
Eject Card

LIBRARY MANAGEMENT SYSTEM



SEQUENCE DIAGRAM

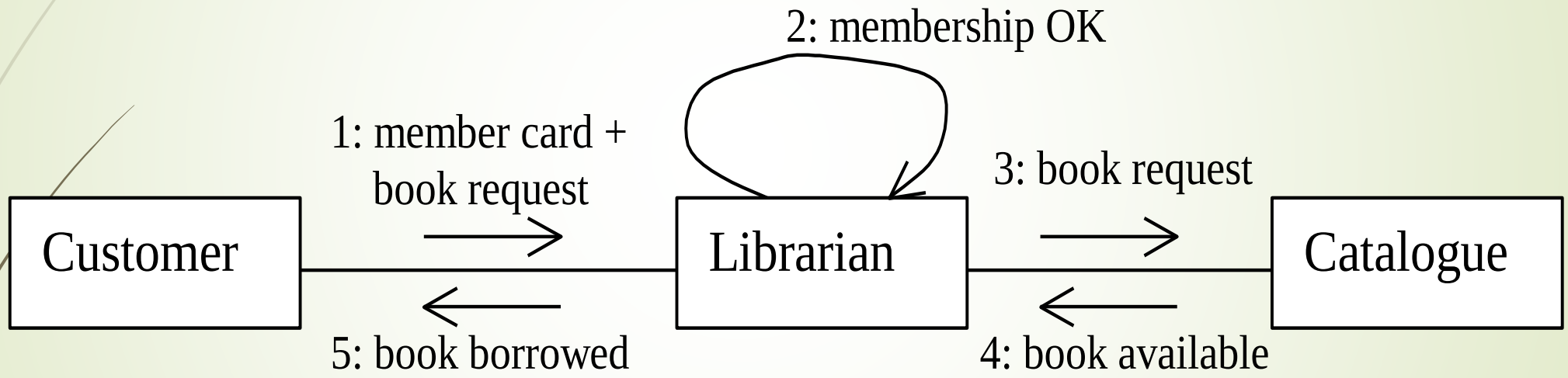
UML collaboration diagram

- def:
  - it is a directed graph in which nodes represent communicating entities & edges represent communications.
- UML collaboration diagrams
  - Shows both structural and behavioral aspects explicitly.
  - give object interactions and their order
  - semantically equivalent to sequence diagrams; syntactically different.

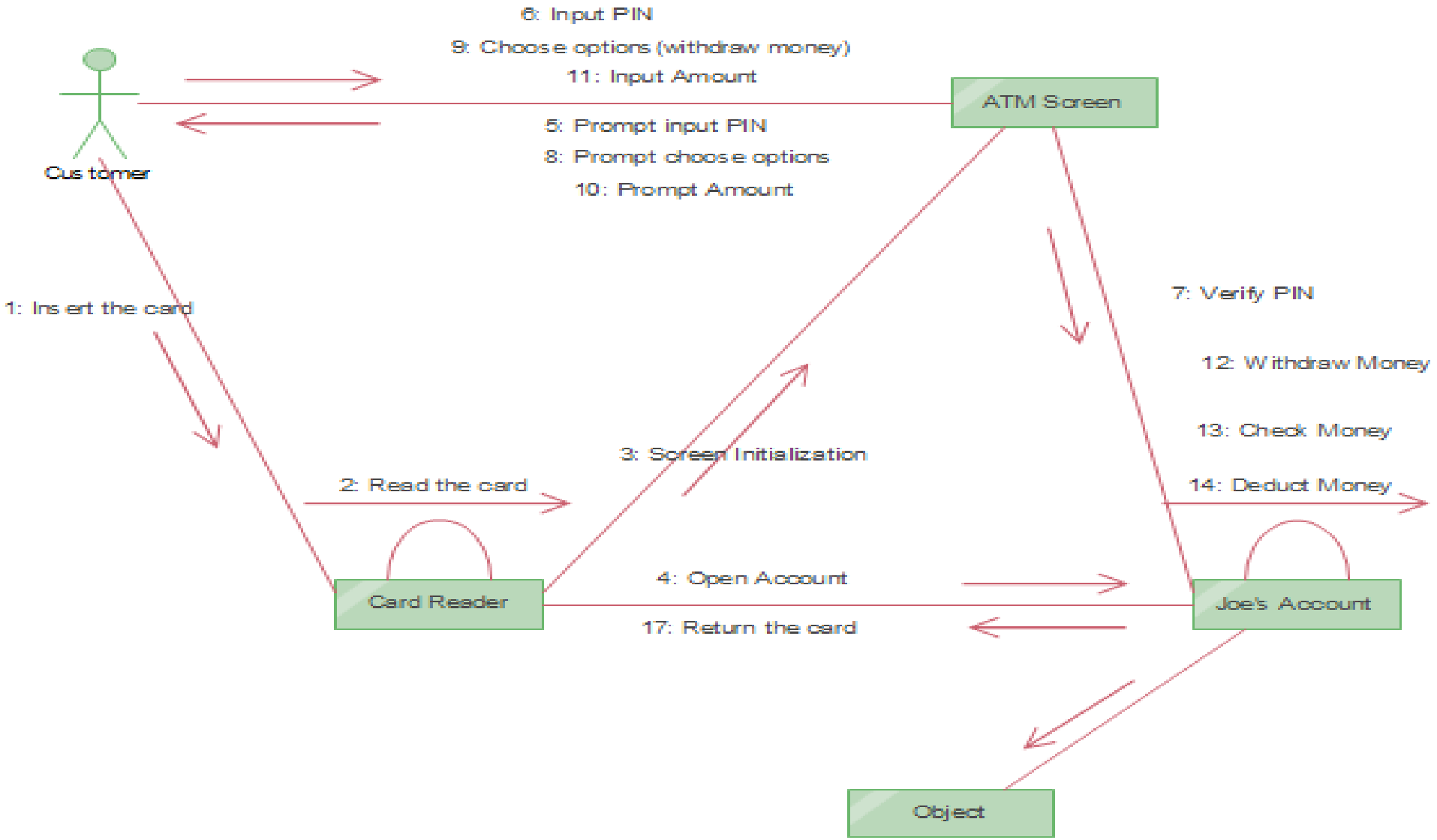
UML collaboration diagram…

- Collaboration diagrams
  - show both structural and behavioural aspects
  - explicitly shows which class is associated with other class.
  - make the structural properties of a collaboration more evident
  - while sequence diagrams make the temporal evolution of the scenarios, more evident.
- Objects are collaborator, shown as boxes
- Messages between objects shown as a solid line
  - Message is shown as a labelled arrow placed near the link
  - Messages are prefixed with sequence numbers to show relative sequencing

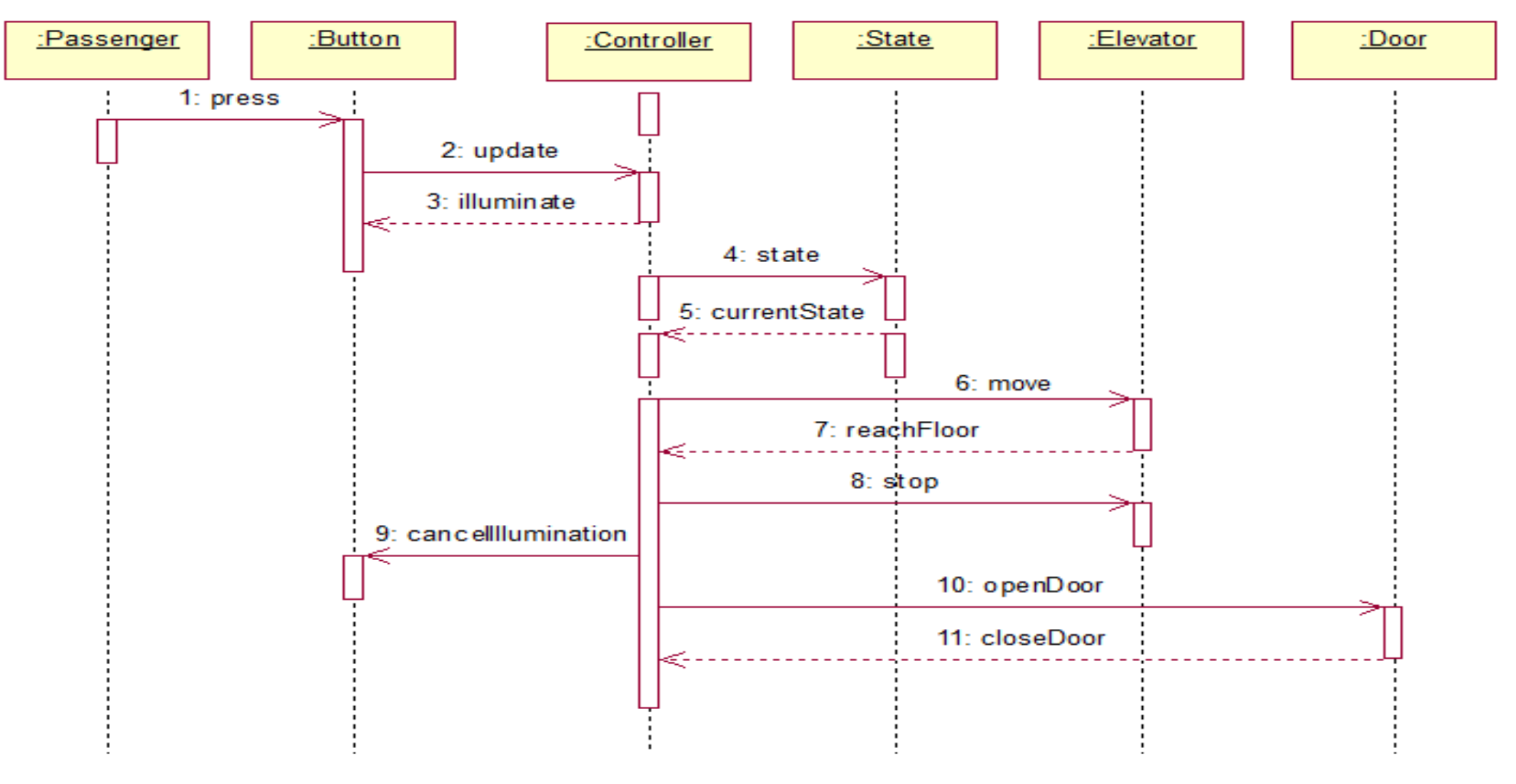
An example – collaboration diagram



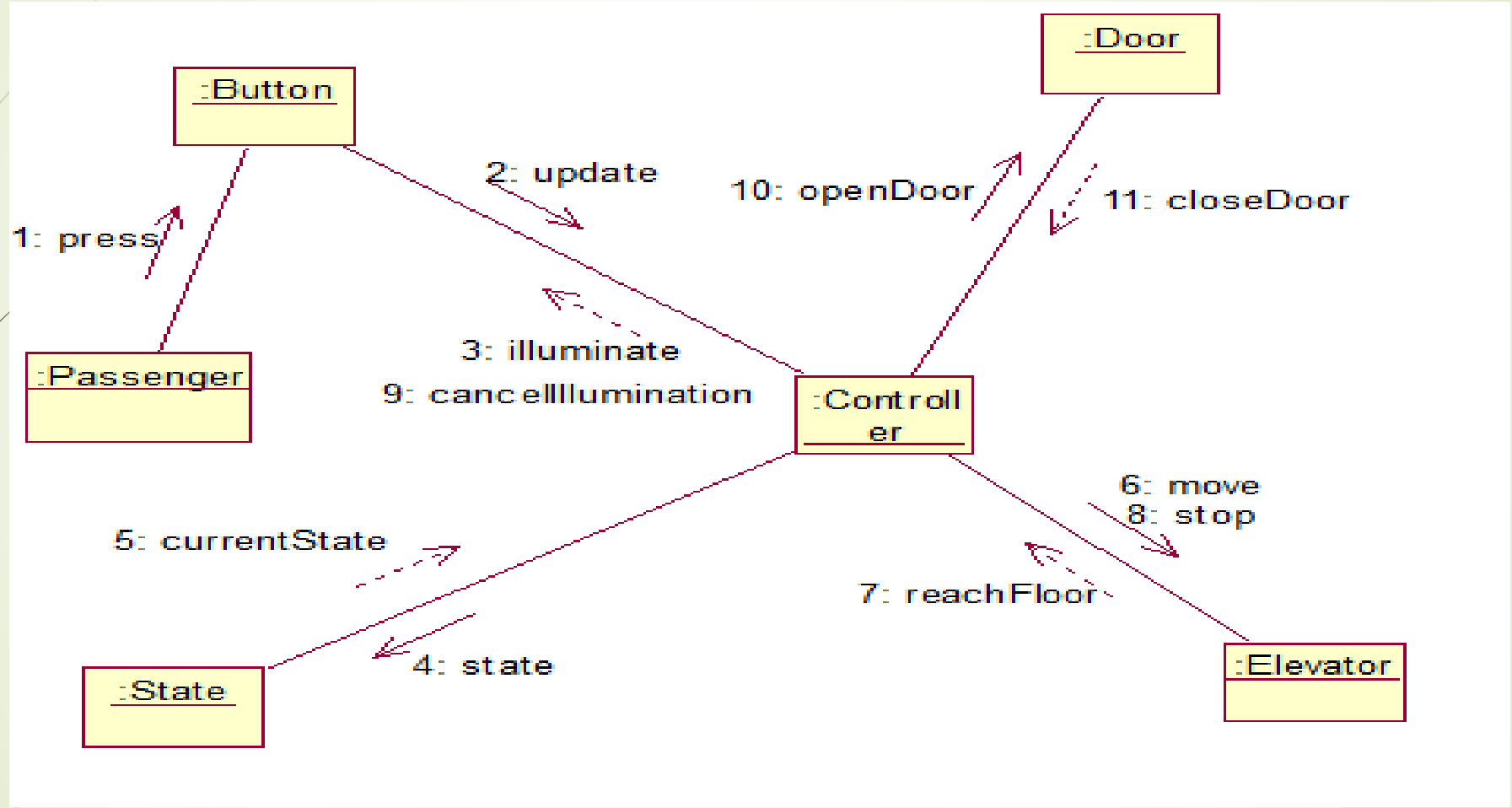
ATM UML Collaboration Diagram



Elevator Controller: Sequence Diagram:



Elevator Controller Collaboration Diagram:



Sequence Diagrams	Collaboration Diagrams
The sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality.	The collaboration diagram also comes under the UML representation which is used to visualize the organization of the objects and their interaction.
The sequence diagram are used to represent the sequence of messages that are flowing from one object to another.	The collaboration diagram are used to represent the structural organization of the system and the messages that are sent and received.
The sequence diagram is used when time sequence is main focus.	The collaboration diagram is used when object organization is main focus.
The sequence diagrams are better suited of analysis activities.	The collaboration diagrams are better suited for depicting simpler interactions of the smaller number of objects.

Activity diagram…

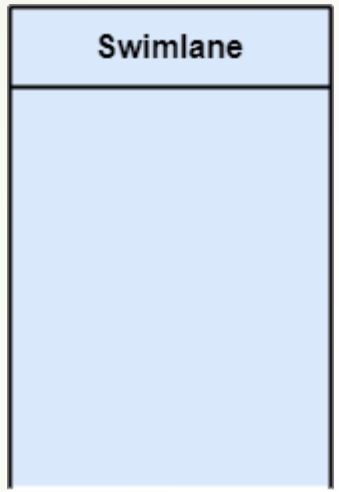
- Activity diagrams are
  - typically used for process modeling, specifically in business process modelling
    - such as for modeling the logic captured by an entire use case or set of use cases, or for modeling the detailed logic of a business rule.
  - carried out during requirement analysis and specification
  - can be used to develop interaction diagrams

Activity diagram…

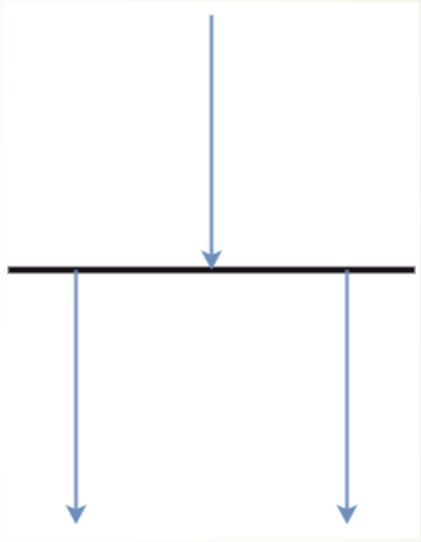
- In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation.
- It models the concurrent and sequential activities.
- The flow can be **sequential**, **branched**, **or concurrent**, and to deal with such kinds of flows, the activity diagram has come up with a **fork, join**, etc.
- It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

Components of an Activity Diagram

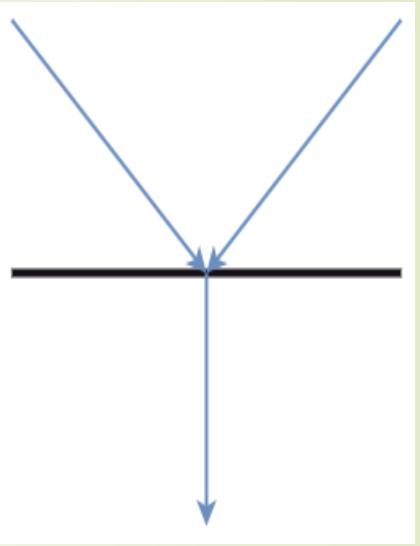
The categorization of behavior into one or more actions is termed as an activity.



The swimlane is used to cluster all the related activities in one column or one row.

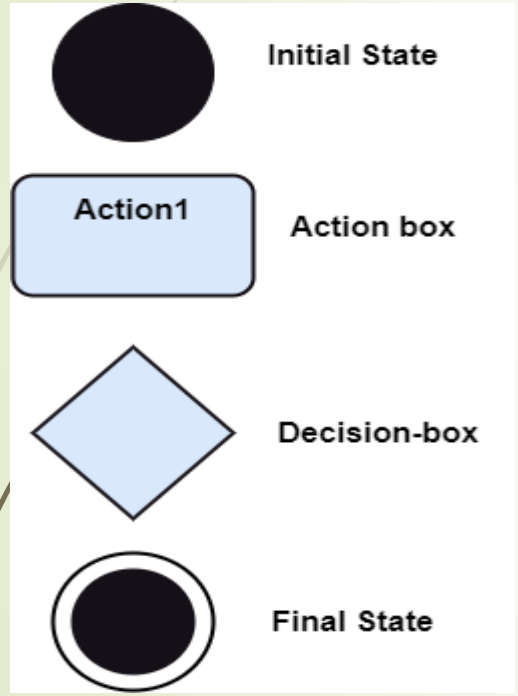


Forks and join nodes generate the concurrent flow inside the activity.



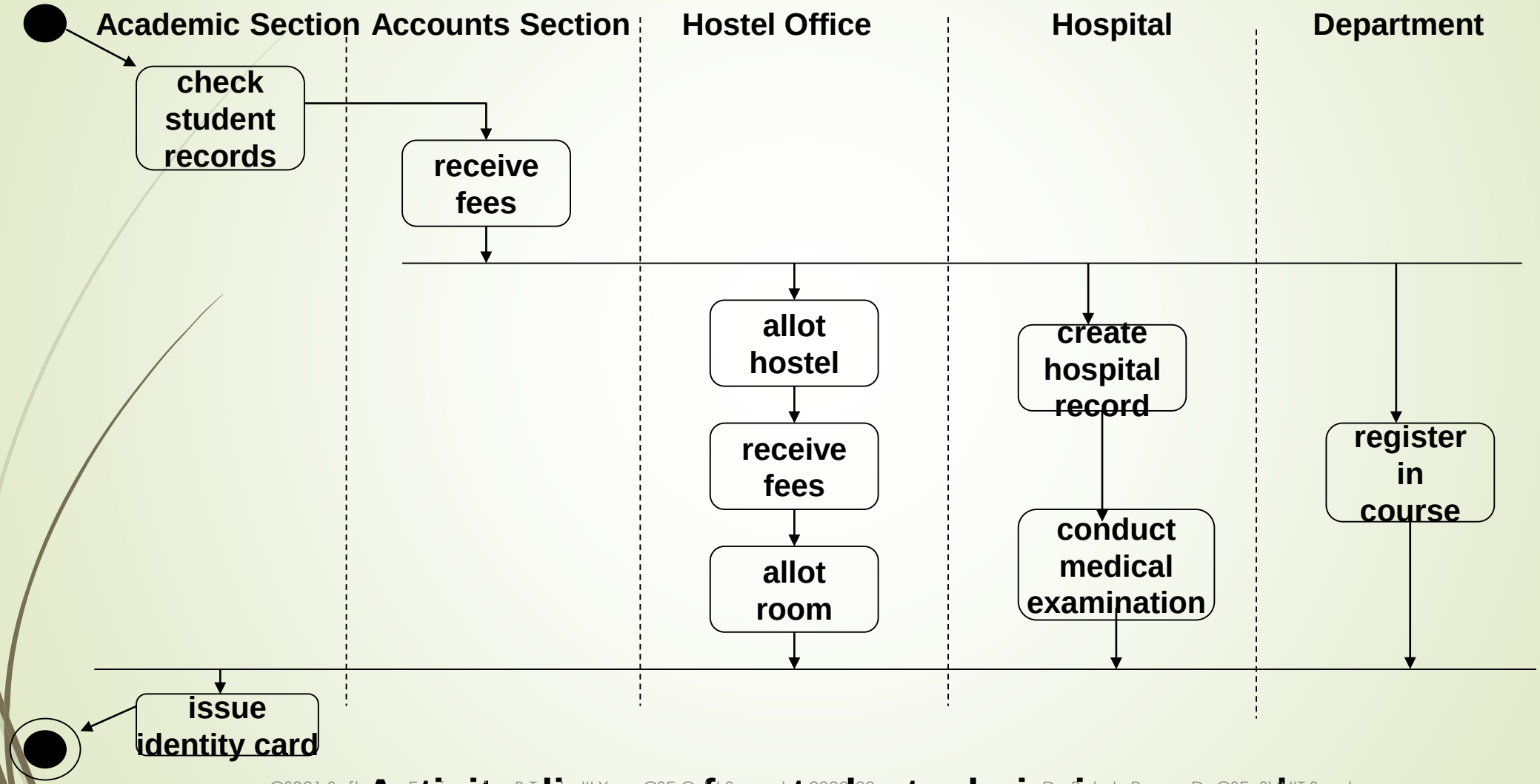
Components of an Activity Diagram

Activity diagram constitutes following notations:



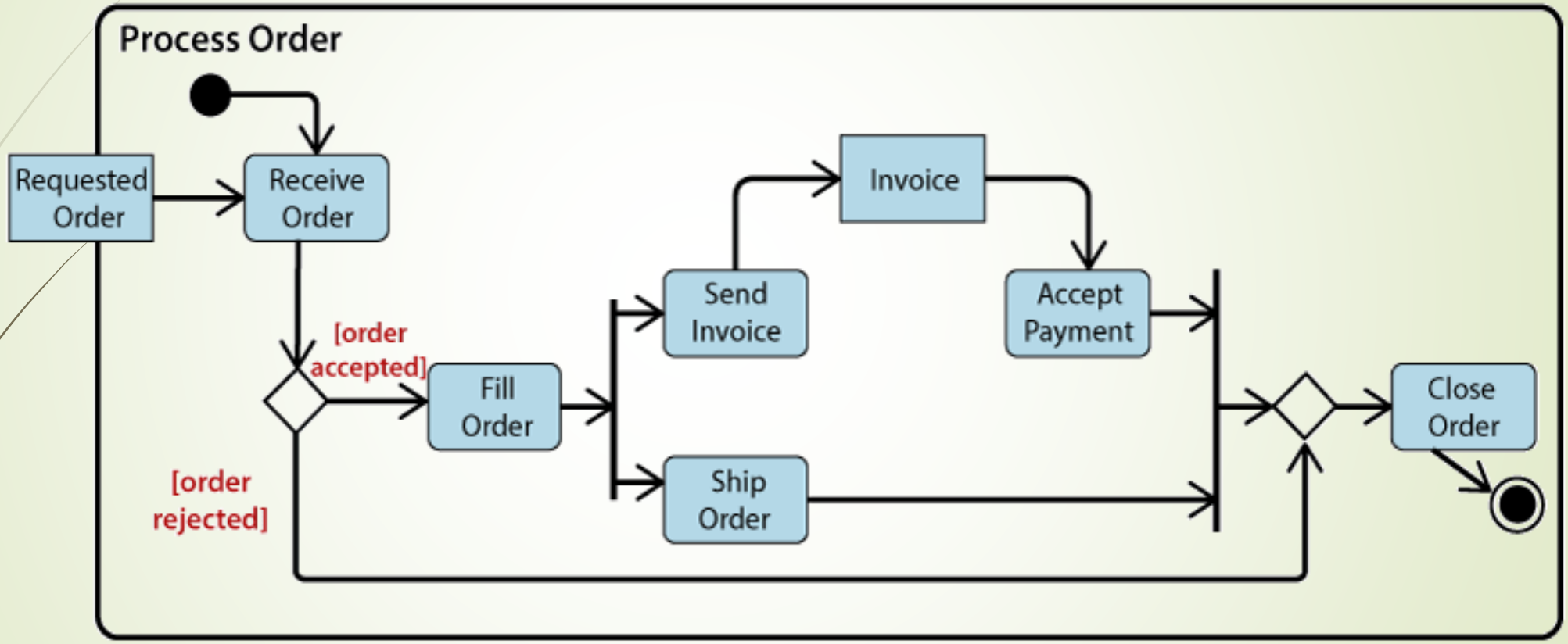
Initial State: It depicts the initial stage or beginning of the set of actions.
Final State: It is the stage where all the control flows and object flows end.
Decision Box: It makes sure that the control flow or object flow will follow only one path.
Action Box: It represents the set of actions that are to be performed.

UML Activity diagram....

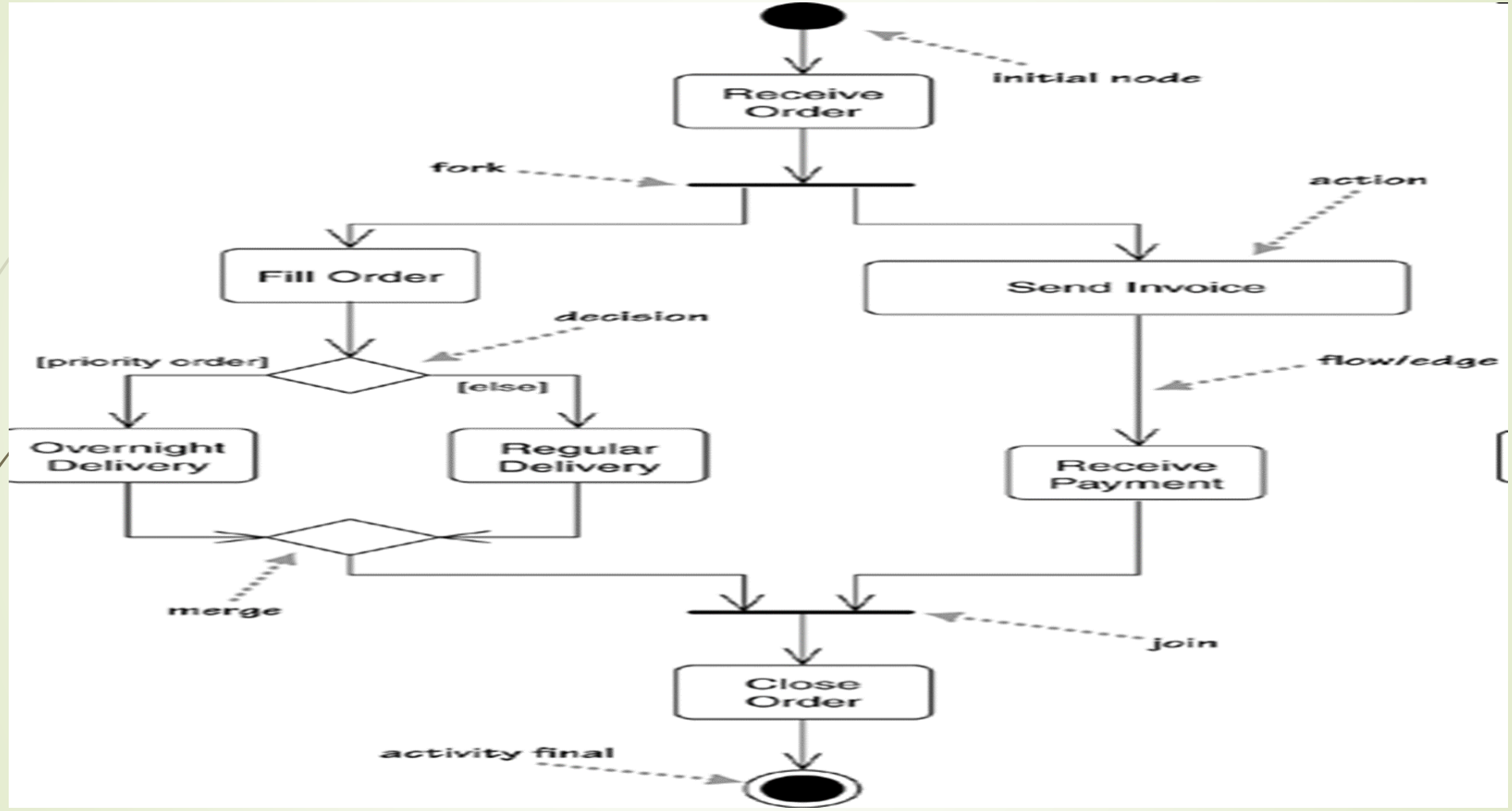


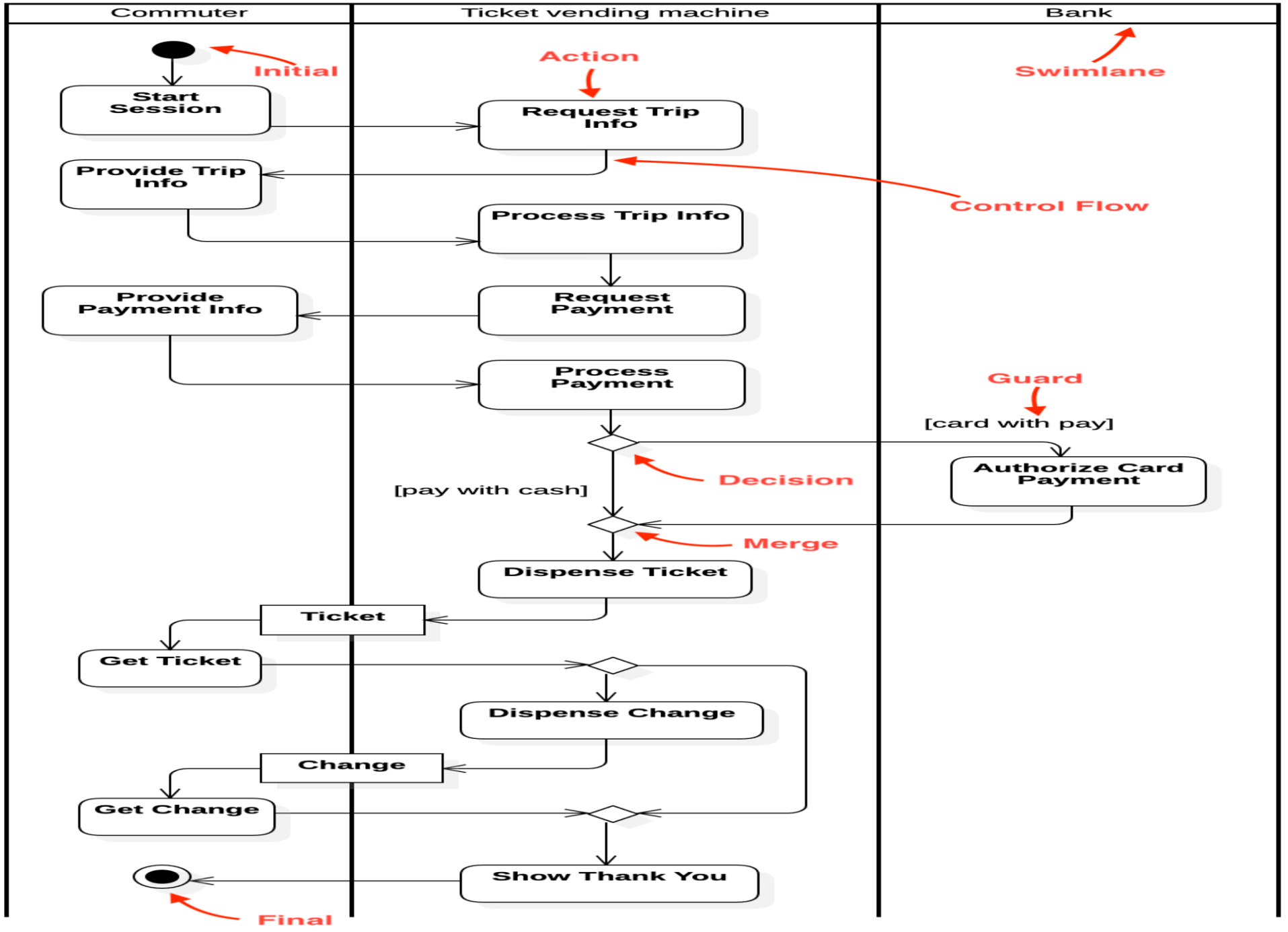
Activity diagram for student admission procedure

UML Activity Diagram…

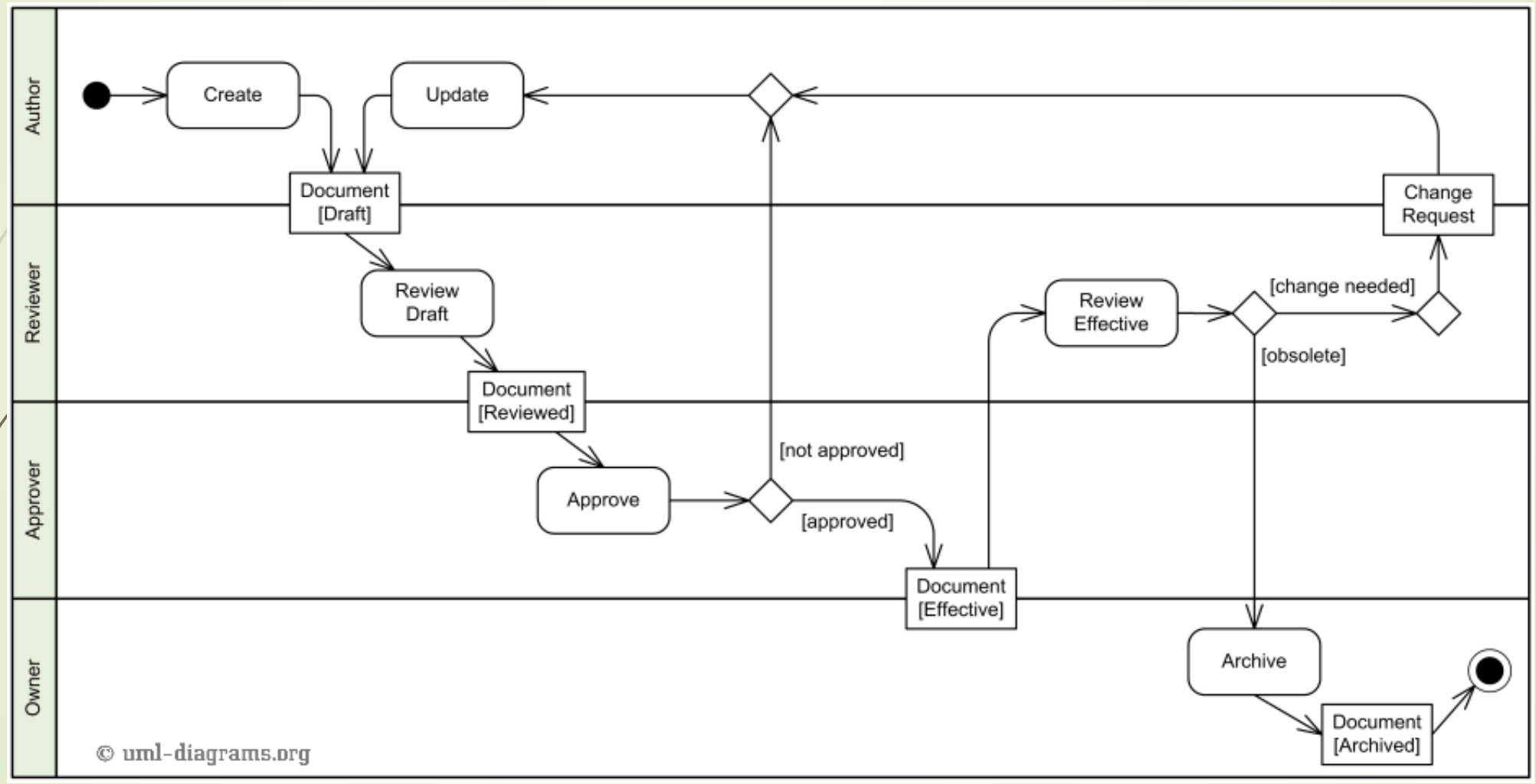


UML Activity diagram…




Commuter
Ticket vending machine
Bank
Initial
Action
Swimlane
Start Session
Request Trip Info
Provide Trip Info
Control Flow
Process Trip Info
Provide Payment Info
Request Payment
Process Payment
Guard
[card with pay]
Decision
[pay with cash]
Authorize Card Payment
Merge
Dispense Ticket
Ticket
Get Ticket
Dispense Change
Change
Get Change
Show Thank You
Final

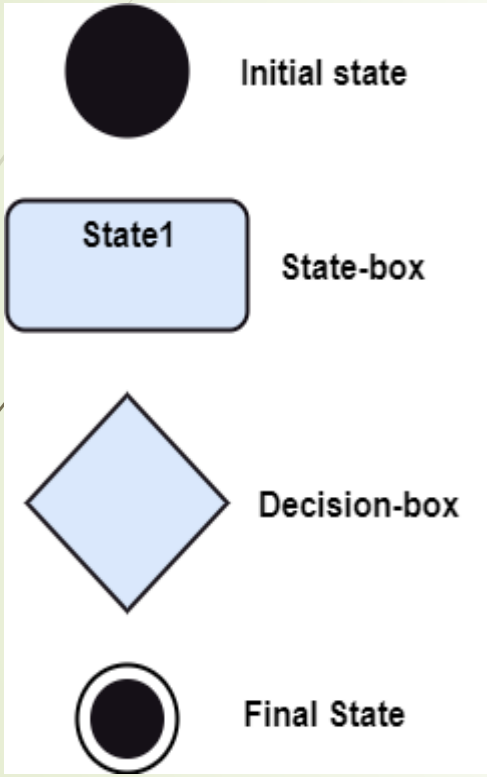
UML Activity Diagram...



UML State Machine Diagram

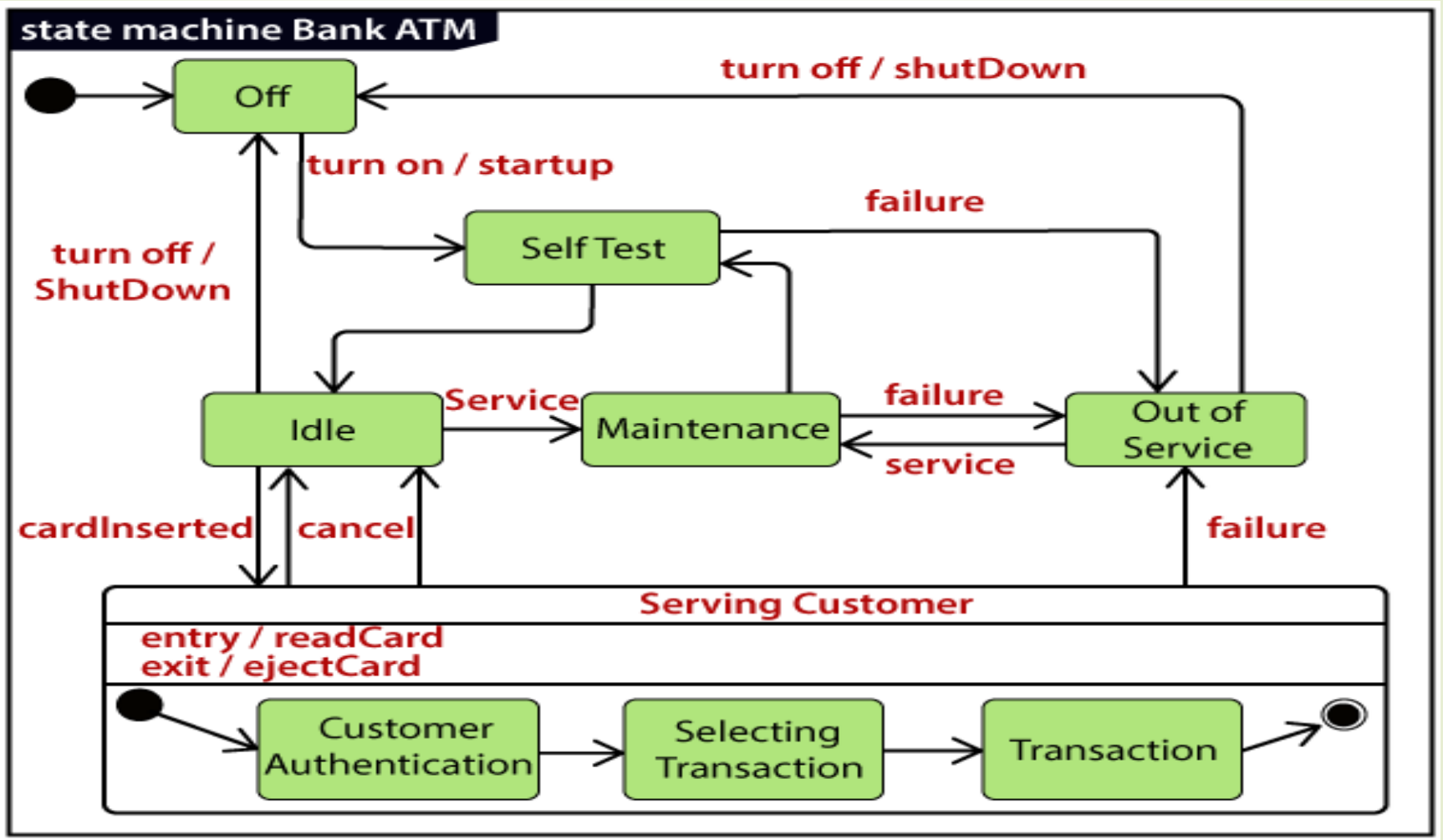
- The state machine diagram is also called the **Statechart or State Transition diagram**, which shows the order of states underwent by an object within the system.
- An efficient way of modeling the interactions and collaborations in the external entities and the system.
- During a lifespan, an object underwent several states, such that the lifespan exist until the program is executing. Each state depicts some useful information about the object.
- The execution flow from one state to another is represented by a state machine diagram. It visualizes an object state from its creation to its termination.

UML State Machine Diagram

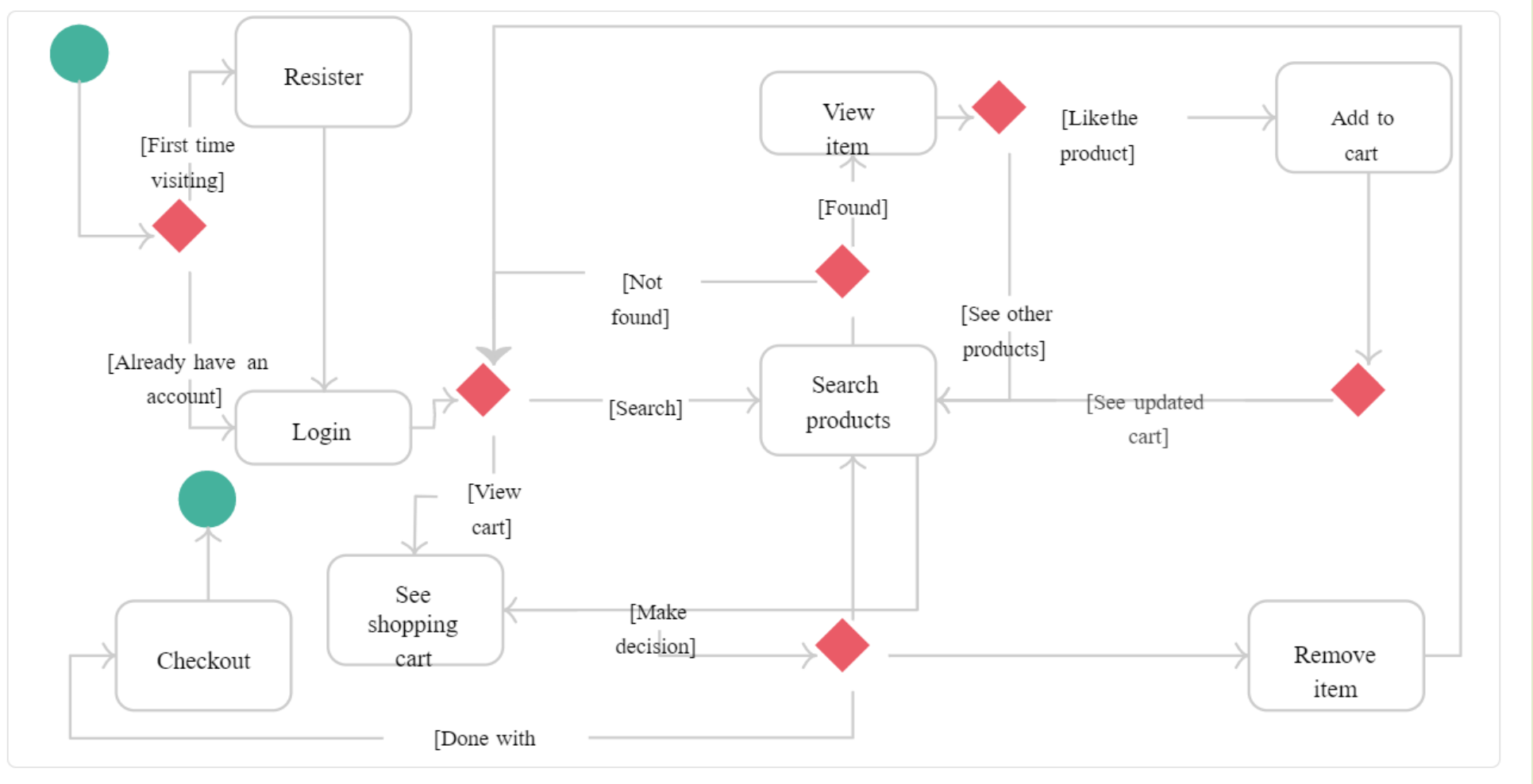


1.Initial state: It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2.Final state: It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3.Decision box: It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4.Transition: A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5.State box: It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

UML State Machine Diagram



UML State Machine Diagram



State Machine vs. Flowchart

State Machine	Flowchart
It portrays several states of a system.	It demonstrates the execution flow of a program.
It encompasses the concept of WAIT, i.e., wait for an event or an action.	It does not constitute the concept of WAIT.
It is for real-world modeling systems.	It envisions the branching sequence of a system.
It is a modeling diagram.	It is a data flow diagram (DFD)
It is concerned with several states of a system.	It focuses on control flow and path.

Thank You.