

# Unit 3

## Interaction Modeling

# Interaction Modeling

- Interaction modeling describes how object interact to produce useful results. The term interact is a set of exchanged messages between the objects.
- The interaction model can be modelled at different levels :-
  - Use case diagram (at high level use case describe how system interacts with outside actors)
  - Sequence diagram (provides more detail and show the messages exchanged among set of objects over time.)
  - Activity diagram (provide more detail and show the flow of control among the steps of a computation.)

# Use Case Diagram

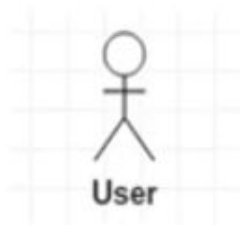
- A UML diagram that represents the relationship between actors and use cases, and among the use cases.
- Represents an “architectural” view of the requirements.
- **Actors :-**
  - External entities (e.g., object, user role, another system)
- **Relationship between actors and use cases :-**
  - Initiation
  - Communication
- **Relationship among different use cases :-**
  - Enables the decomposition of complex use cases into smaller ones

# Use Case Diagram

## COMPONENT OF USE CASE DIAGRAM

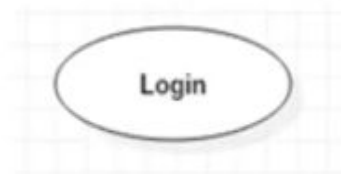
### ACTORS

WHO INTERACTS WITH  
THE SYSTEM



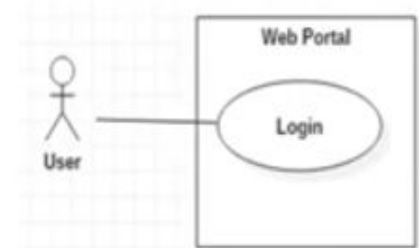
### USE CASE

FUNCTIONALITY OR  
SERVICES PROVIDED BY  
THE SYSTEM



### RELATION

RELATION BETWEEN  
ACTORS AND THE  
SYSTEM



**Stick man** icon denotes an actor

A name with in an **ellipse** denotes a use case.

**Solid line** connect use cases to participating actors.

# Use Cases

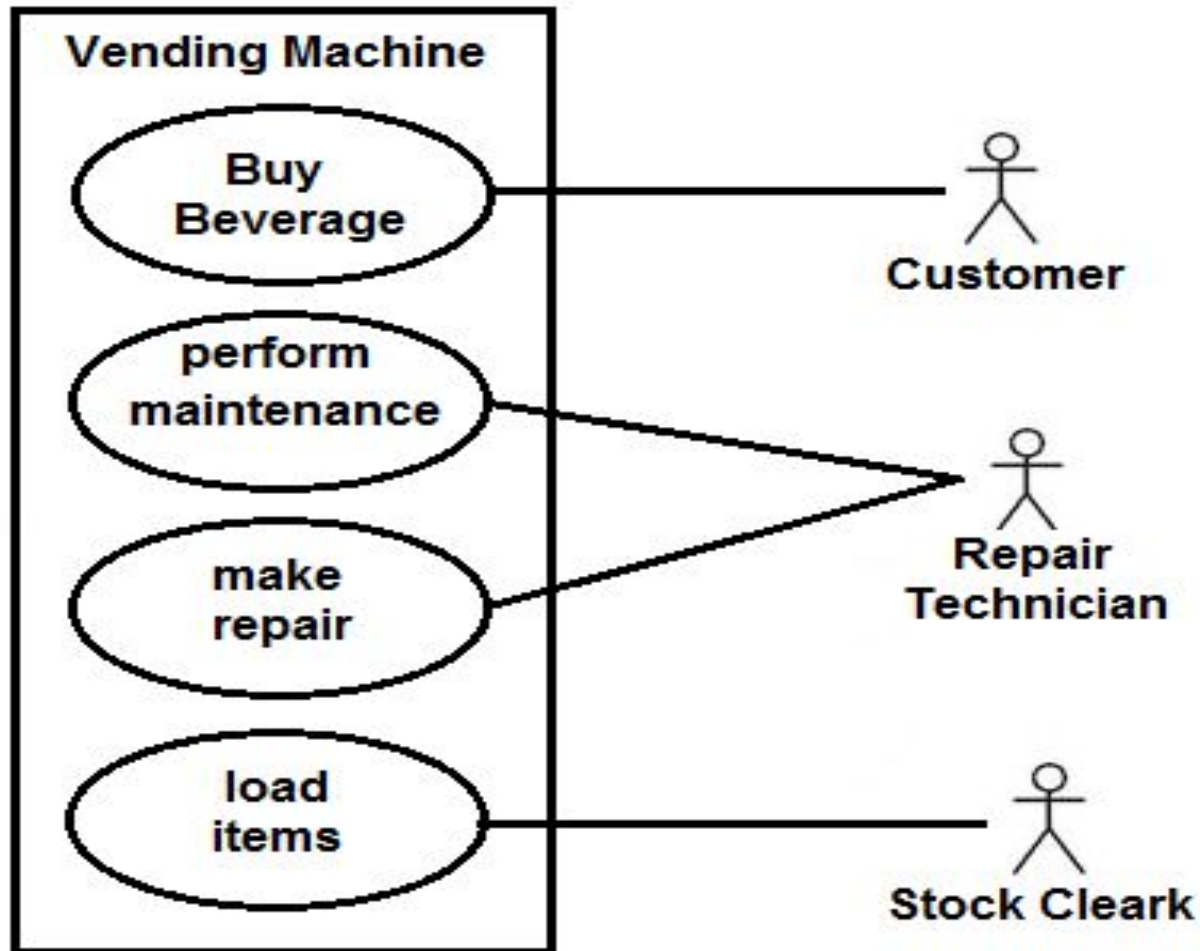
- A use case is a coherent piece of functionality that a system can provide by interacting with actors. For ex a customer actor can buy a beverage from a vending m/c ,repair tech can perform schedule maintenance on a vending m/c etc..
- Each use case involve one or more actors.
- A use case involves sequence of messages among system and its actor.
- Error condition is also a part of use case.

# Use Case summarization for a Vending Machine.

- **Buy a beverage.** The vending machine delivers a beverage after a customer selects and pays for it.
- **Perform scheduled maintenance.** A repair technician performs the periodic service on the vending machine necessary to keep it in good working condition.
- **Make repairs.** A repair technician performs the unexpected service on the vending machine necessary to repair a problem in its operation.
- **Load items.** A stock clerk adds items into the vending machine to replenish its stock of beverages.

**Figure 7.1 Use case summaries for a vending machine.** A use case is a coherent piece of functionality that a system can provide by interacting with actors.

## 7.1.3 Use Case Diagram for Vending Machine



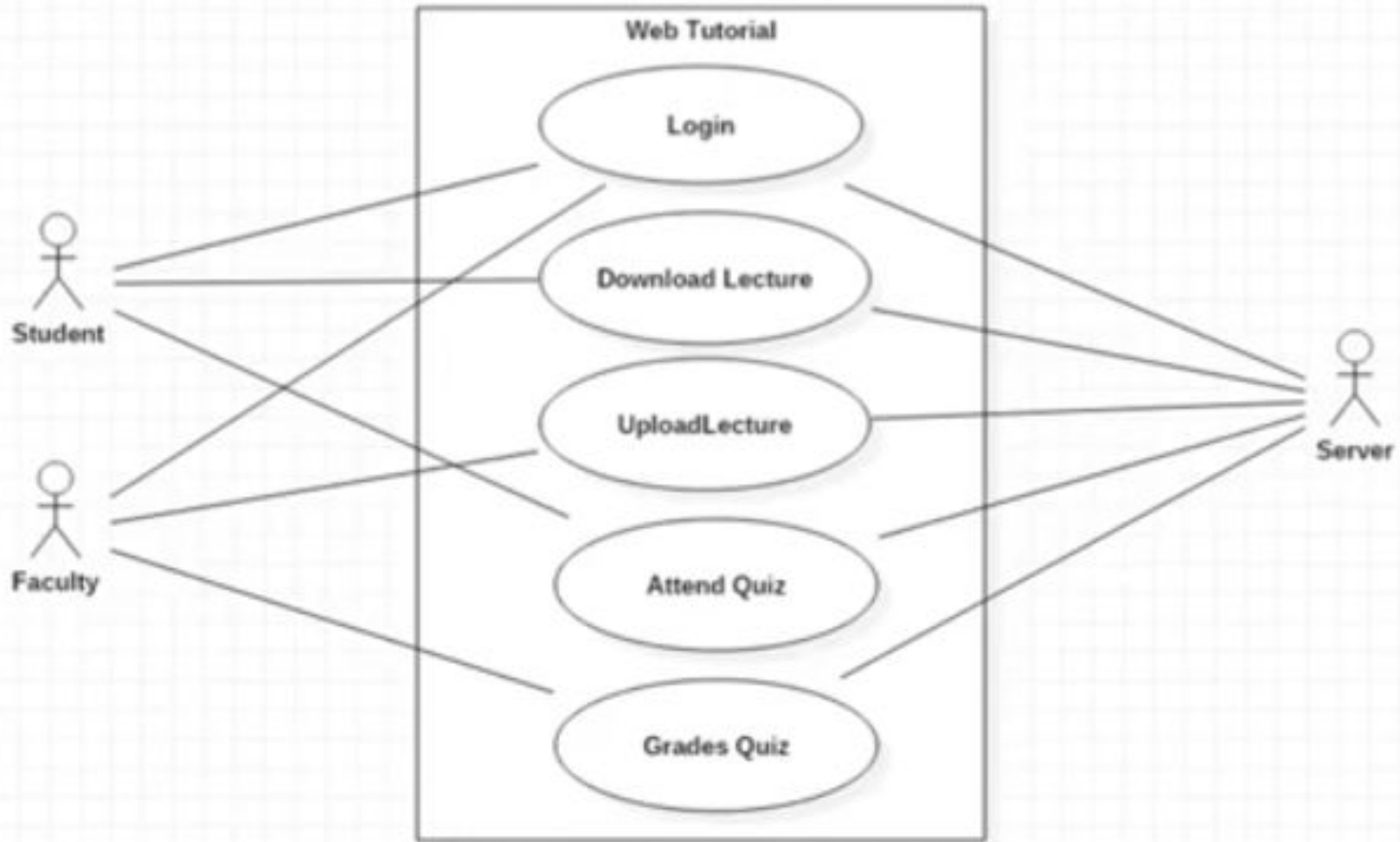
In the above figure ,actor repair technician participates in two use cases, the other in one each.

# Use case description.

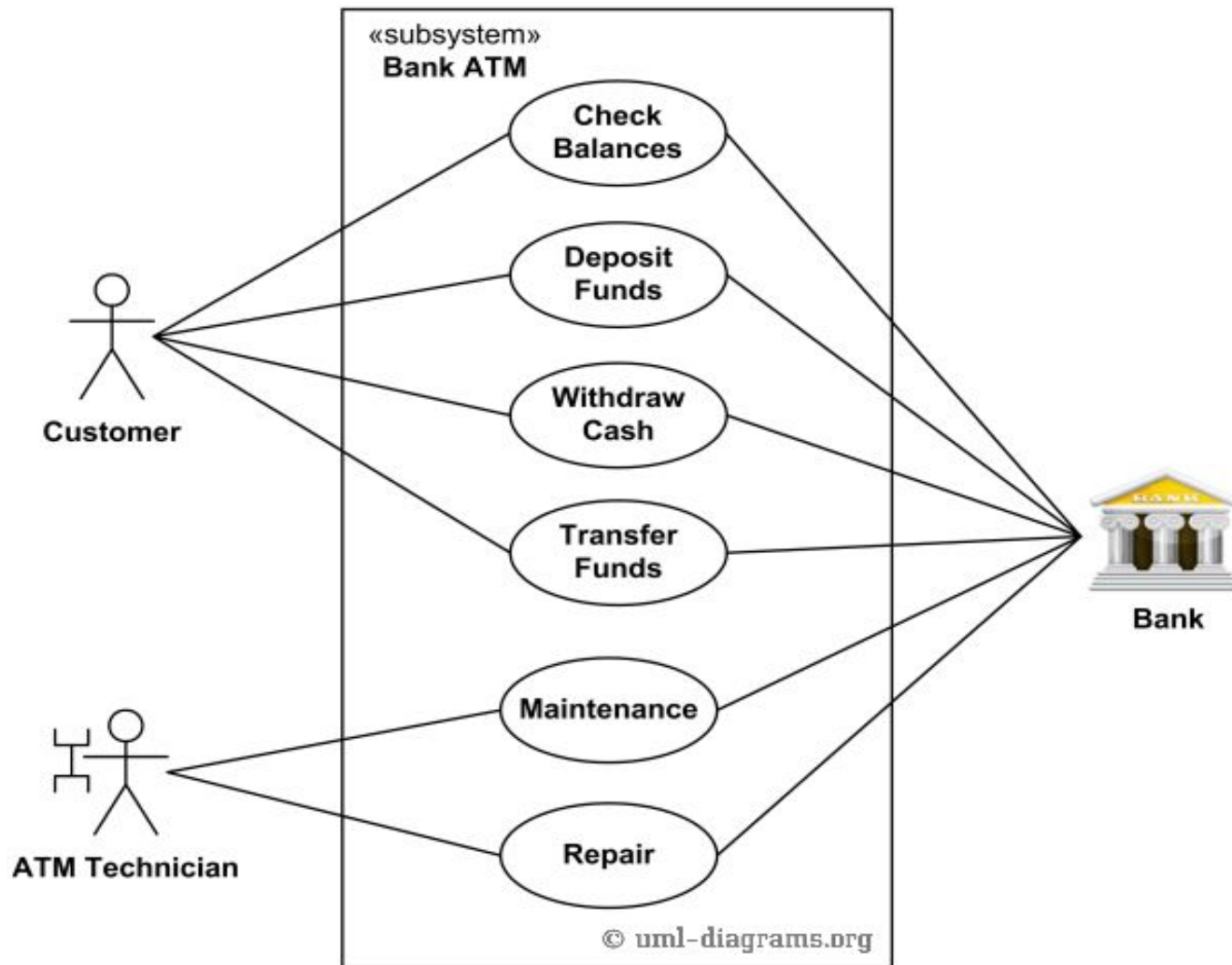
- **Use Case :**Buy a beverage
- **Summary:** A vending machine delivers a beverage after customers selects and pays for it.
- **Actors:** Customer
- **Pre Condition:** The machine is waiting for money to be inserted.
- **Description:** The machine starts and says “Enter Coins”. A customer inserts coin into the machine. The machine displays the total value of money entered and lights up the button for item to be selected. The customer selects the item . The machine dispenses the item and makes change ,if the cost of the item is less than the money inserted.
- **Exceptions:**
  - Canceled
  - Out of stock
  - Insufficient money
  - No change
- **Post Condition:** The machine is waiting for money to be inserted.



# Use Case Diagram for Web Tutorial.



# Example: Use case diagram for Bank ATM .



*An example of use case diagram for Bank ATM subsystem - top level use cases.*

## 7.1.4 Guidelines for Use Case Models

- Use case identify the functionality of a system and organize it according to the perspective of the users.
- Some guidelines for constructing Use Case Models are:
  1. **First determine the system boundaries-**
    - Identify use cases or actors
  2. **Ensure that actors are focused-**
    - Each actor should have a single, coherent purpose.
    - Real-world objects embodies multiple purpose may be with separate actors
  3. **Each use case must provide value to users-**
  4. **Relate use cases and actors-**
  5. **Remember that use cases are informal-**
  6. **Use cases can be structured-**

## 7.2 Sequence Models

Sequence models elaborates the themes of use cases.

There are two kinds of sequence models: **Scenarios** and more structured format called **Sequence diagrams**.

### 7.2.1 Scenarios:

Def: “A scenario is a sequence of events that occurs during one particular execution of a system, such as for a use case”.

A scenario can be displayed as a list of text statements as shown below

Withdraw money from ATM

Bank customer inserts debit card and enters PIN.  
Customer is validated.  
ATM displays actions available on ATM unit. Customer selects Withdraw Cash.  
ATM prompts account.  
Customer selects account.  
ATM prompts amount.  
Customer enters desired amount.  
Information sent to Bank, inquiring if sufficient funds/allowable withdrawal limit.  
Money is dispensed and receipt prints.

## 7.2 Sequence Models

John Doe logs in.  
System establishes secure communications.  
System displays portfolio information.  
John Doe enters a buy order for 100 shares of GE at the market price.  
System verifies sufficient funds for purchase.  
System displays confirmation screen with estimated cost.  
John Doe confirms purchase.  
System places order on securities exchange.  
System displays transaction tracking number.  
John Doe logs out.  
System establishes insecure communication.  
System displays good-bye screen.  
Securities exchange reports results of trade.

**Figure 7.4 Scenario for a session with an online stock broker.** A scenario is a sequence of events that occurs during one particular execution of a system.

Figure 7.4 illustrates A scenario can be displayed as a list of text statements. In this example, John Doe logs on with an online stock broker system, places an order for GE stock, and then logs off. Sometime later, after the order is executed, the securities exchange reports the results of the trade to the broker system. John Doe will see the results on the next login, but that is not part of this scenario.

## 7.2 Sequence Models

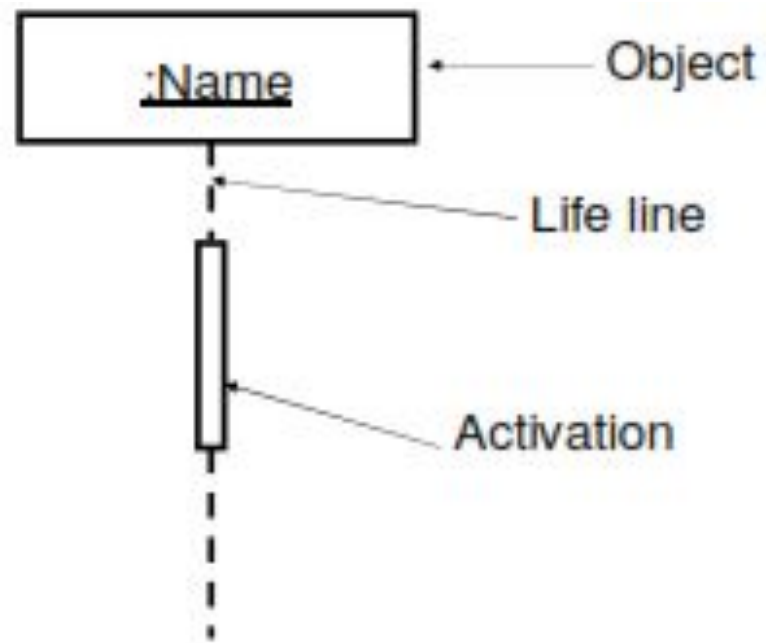
### 7.2.2 Sequence Diagrams:

Def: “A Sequence diagram shows the participants in an interaction and the sequence of messages among them”.

- A Sequence diagram shows the interaction of a system with its actor to perform all or part of a use case.
- Each **Actor** or **System** is represented by **Vertical Line called Lifeline**

Sequence diagrams consists of :

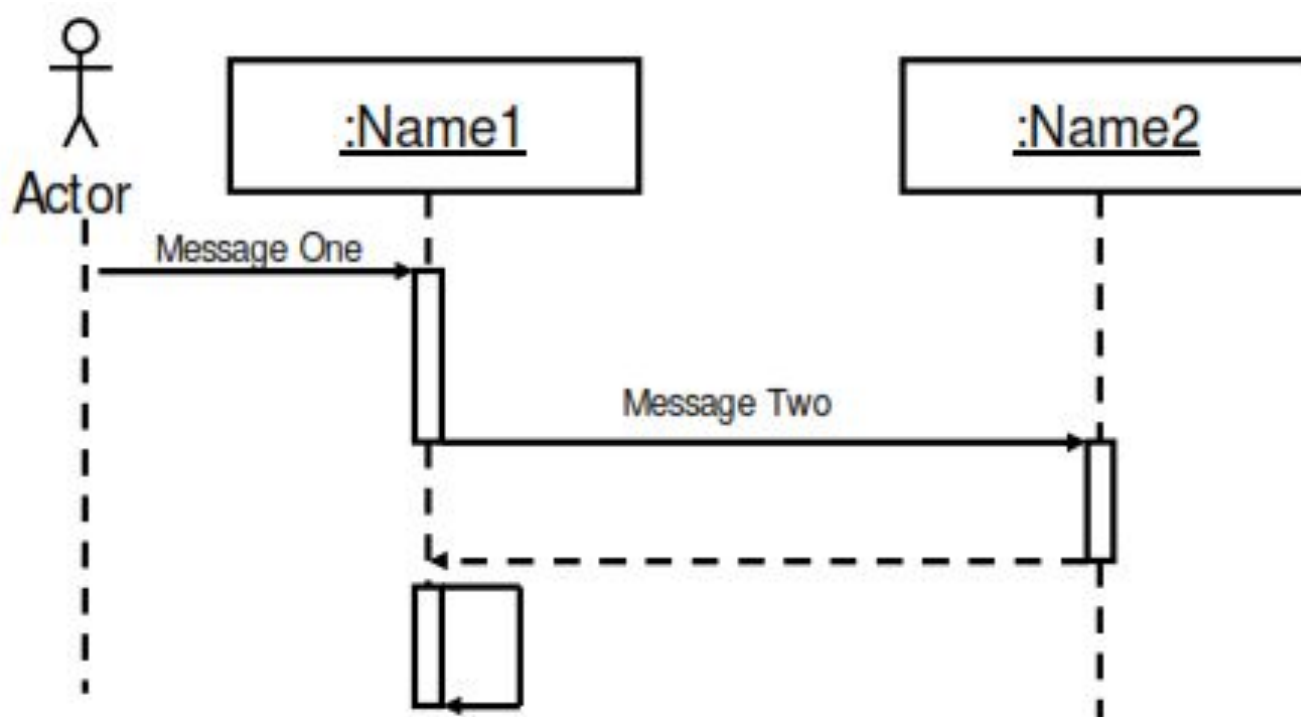
1. Active Objects,
2. Messages and
3. Time



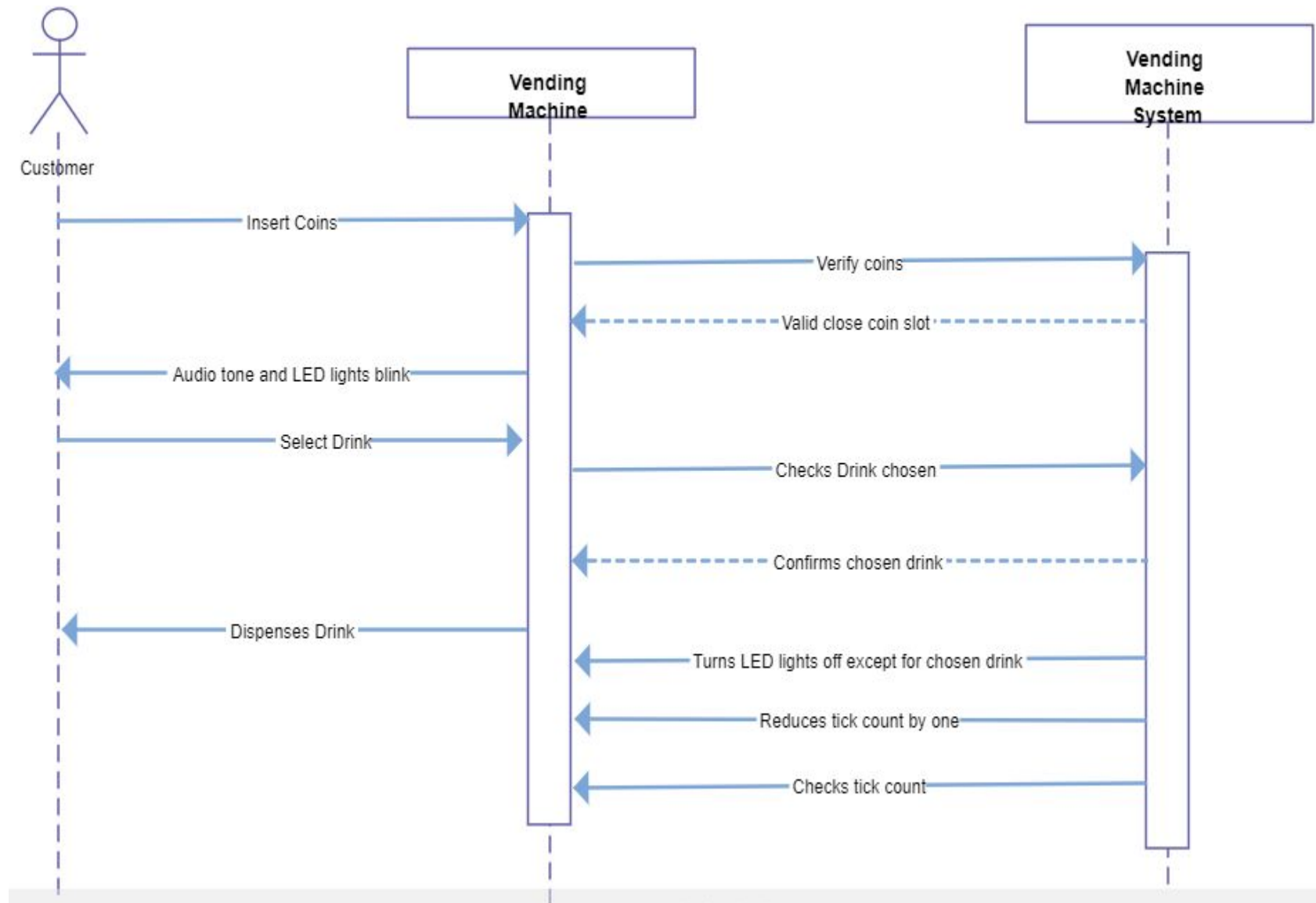
## 7.2 Sequence Models

### 7.2.2 Sequence Diagrams:

#### Sequence Diagram – Time and Message

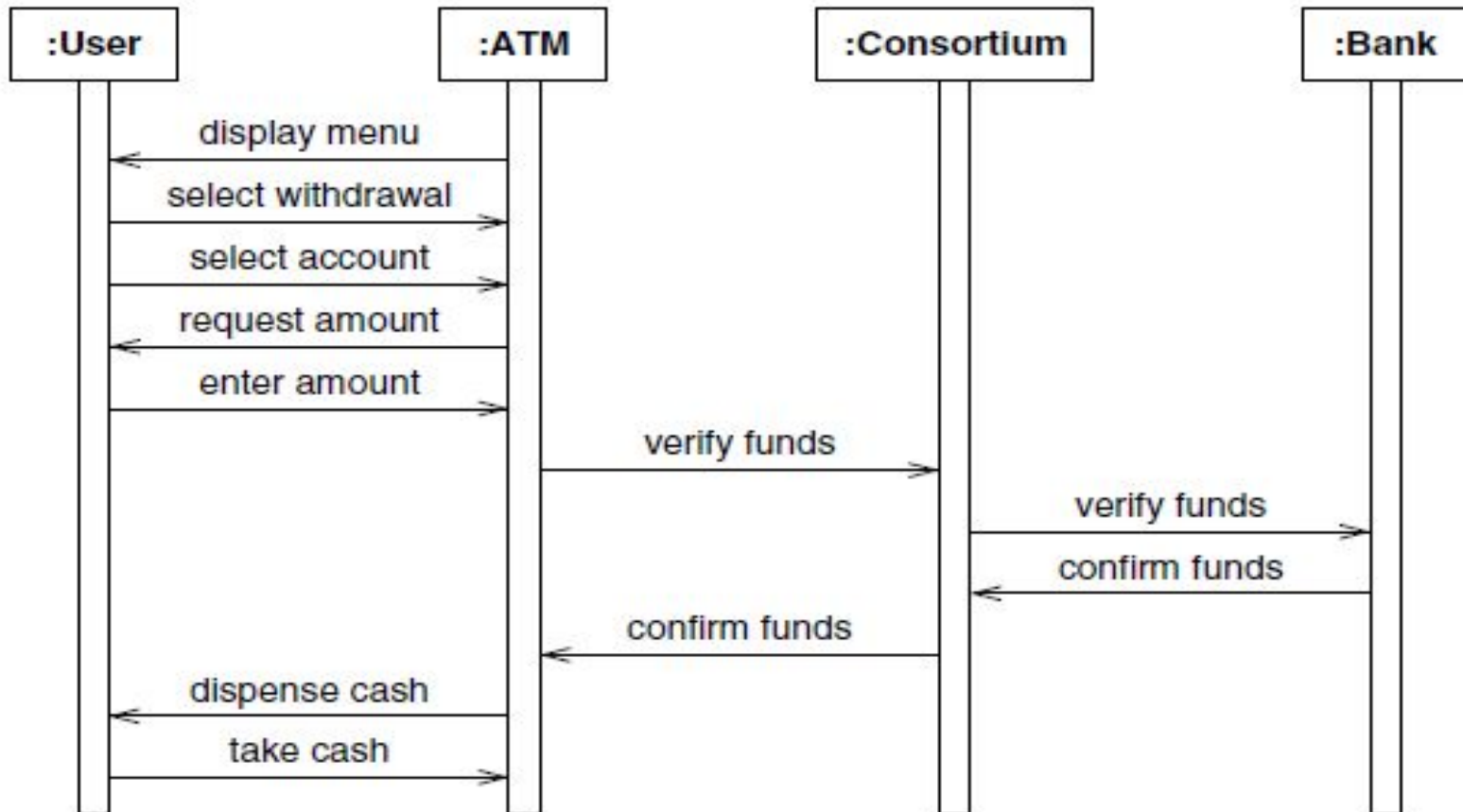


# Vending machine sequence diagram.





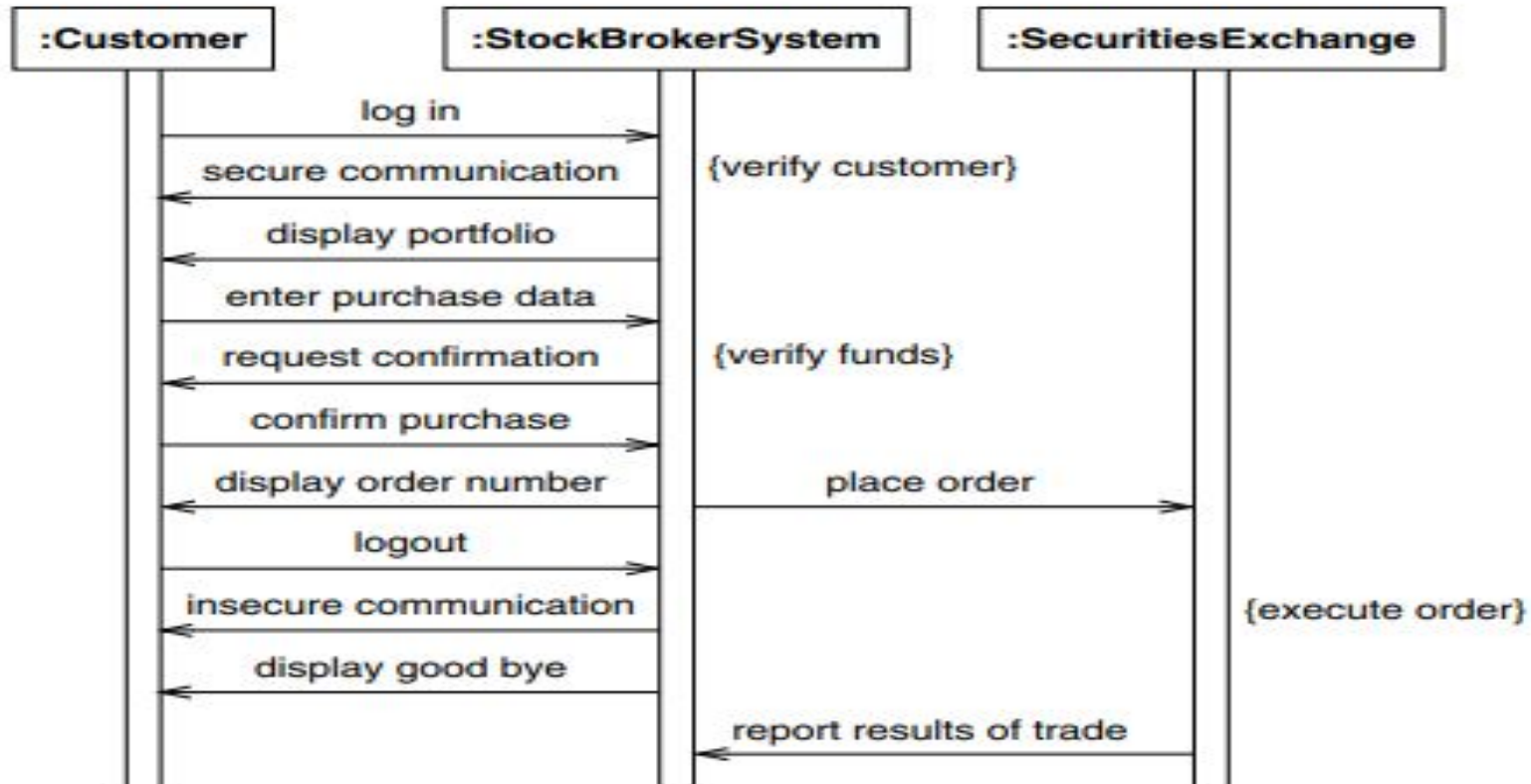
# ATM Example



# SEQUENCE DIAGRAM

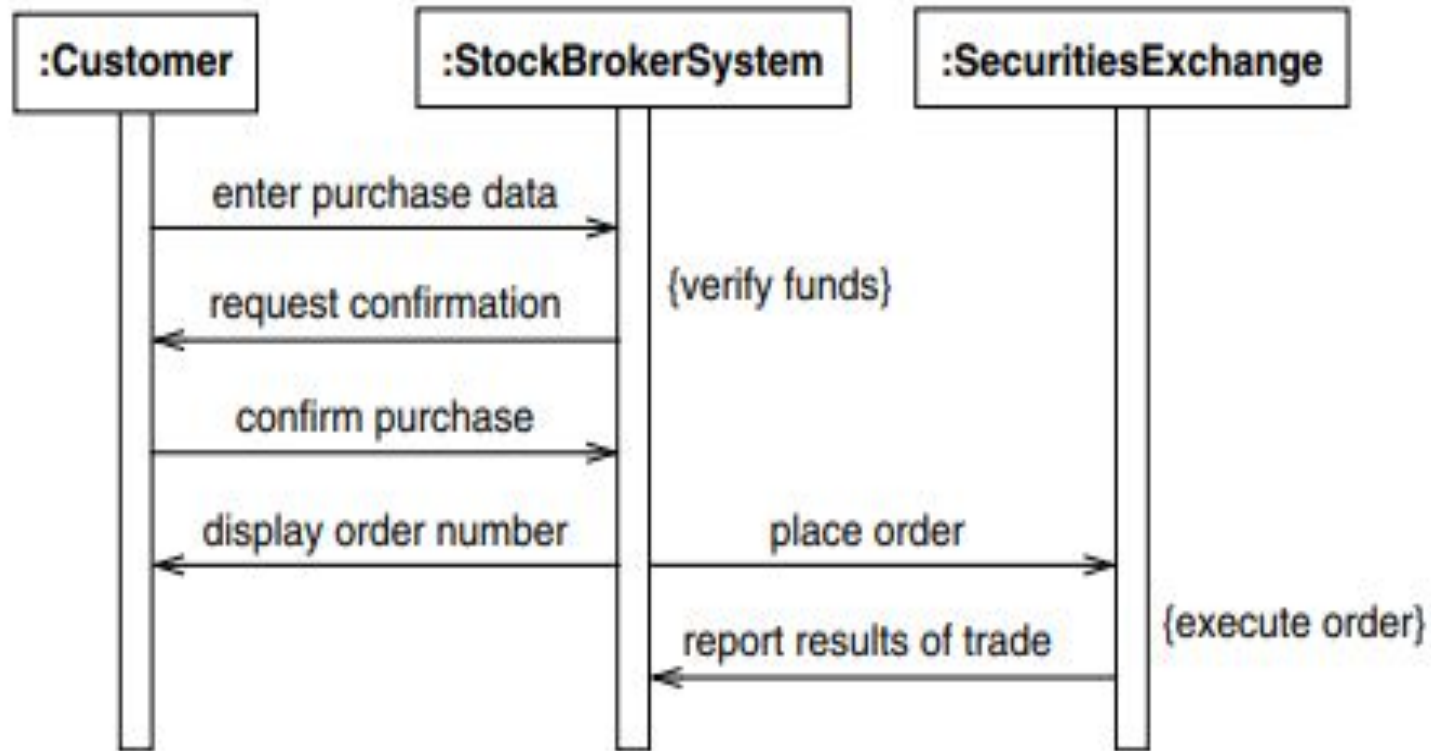
- Figure 7.5 shows a sequence diagram corresponding to the stock broker scenario.
- Each **actor** as well as the **system** is represented by a **vertical line** called a **lifeline** and each **message** by a **horizontal arrow** from the sender to the receiver.
- Time proceeds from top to bottom, but the spacing is irrelevant; the diagram shows only the sequence of messages, not their exact timing. (Real-time systems impose time constraints on event sequences, but that requires extra notation.)
- Each use case requires one or more sequence diagrams to describe its behavior. Each sequence diagram shows a particular behavior sequence of the use case.

# SEQUENCE DIAGRAM for a session with online stock broker



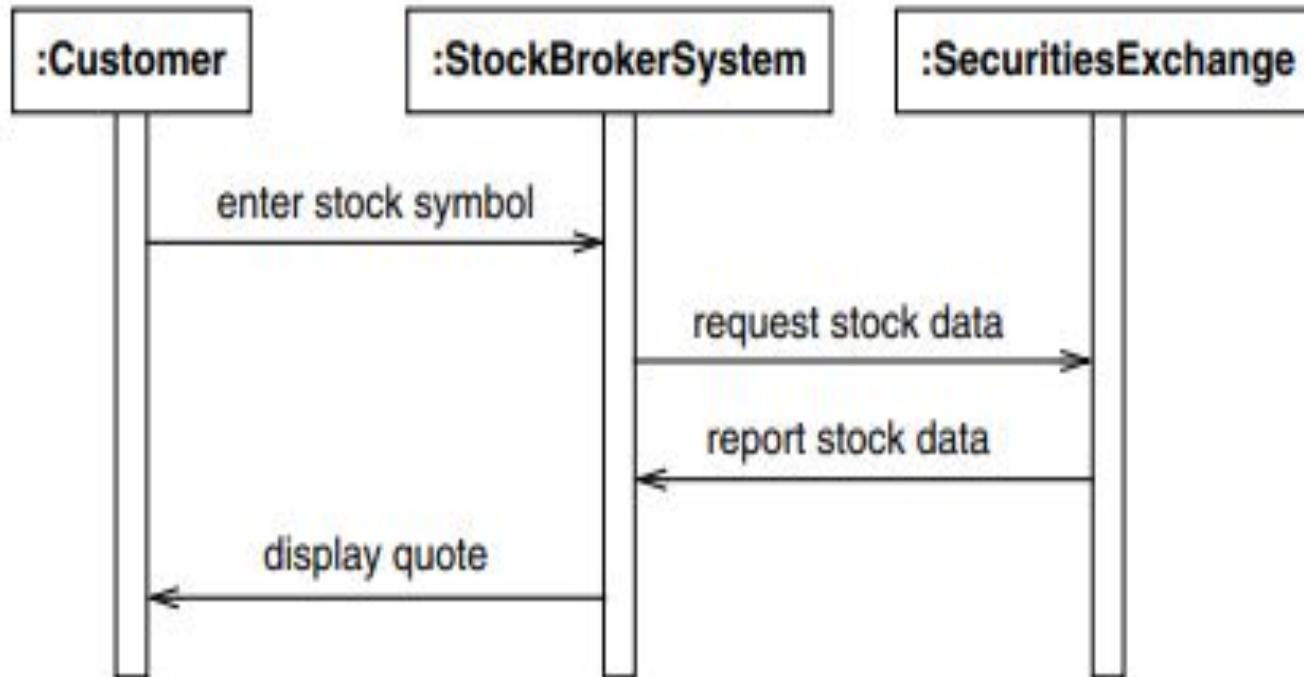
**Figure 7.5** Sequence diagram for a session with an online stock broker. A sequence diagram shows the participants in an interaction and the sequence of messages among them.

# Sequence diagram for a stock purchase



**Figure 7.6** Sequence diagram for a stock purchase. Sequence diagrams can show large-scale interactions as well as smaller, constituent tasks.

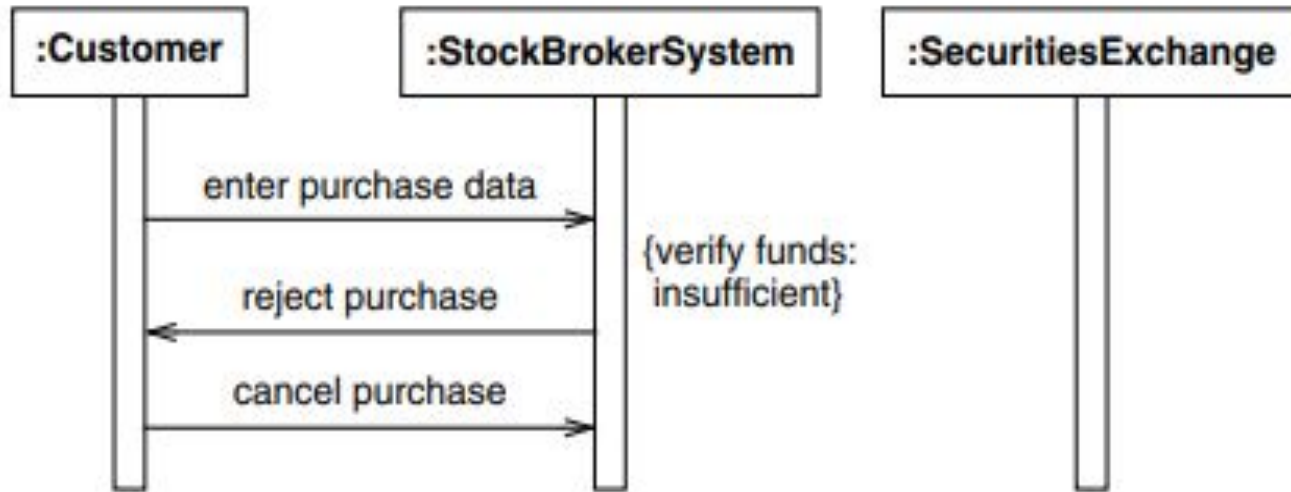
# Sequence diagram for stock quote



**Figure 7.7** Sequence diagram for a stock quote

# Sequence diagram for a stock purchase that fails.

An exception condition within the use case that fails .

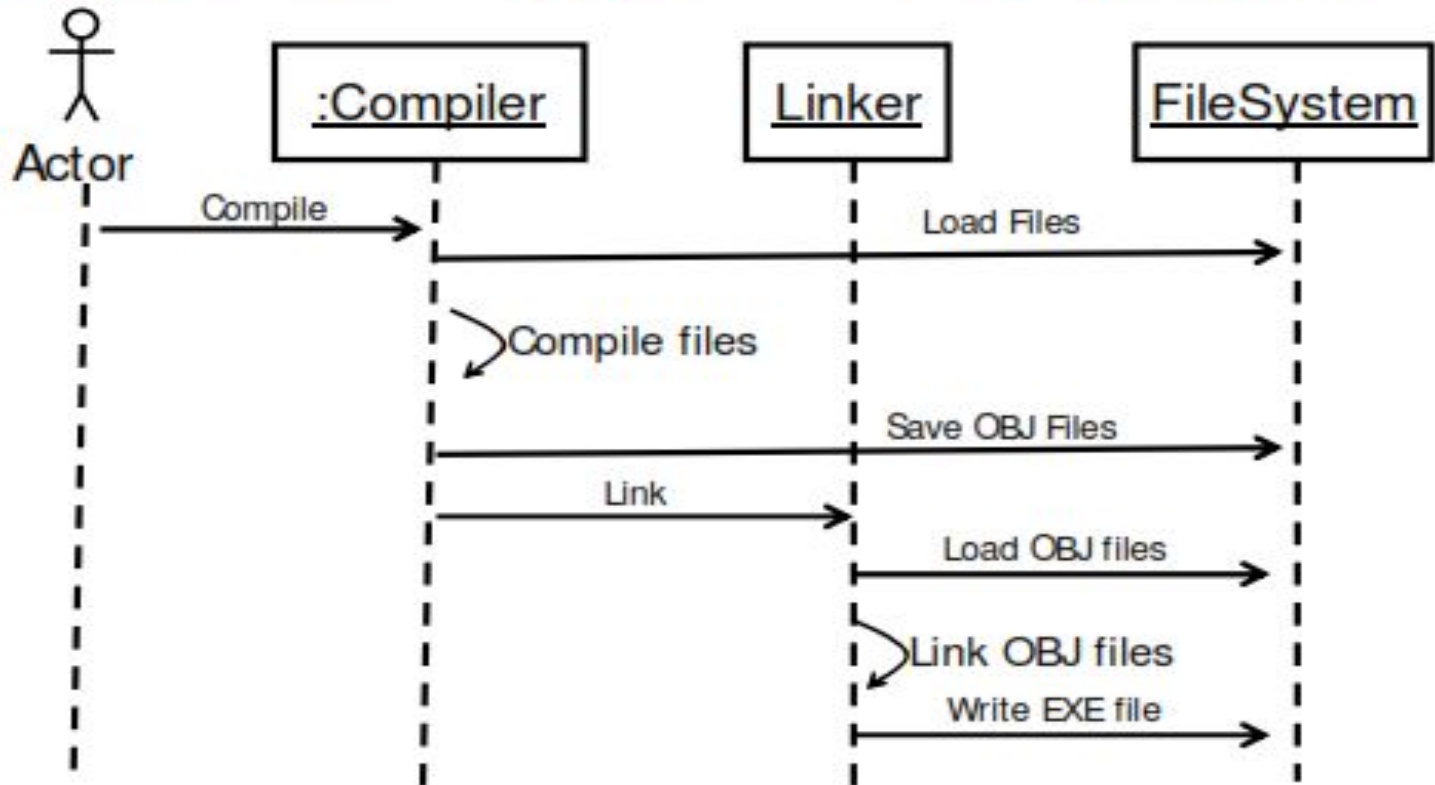


**Figure 7.8** Sequence diagram for a stock purchase that fails

For example, Figure 7.8 shows a variation in which the customer does not have sufficient funds to place the order. In this example, the customer cancels the order.

# Example Compilation

## Sequence Diagram – Compilation



## 7.2 Sequence Models

### 7.2.3 Guidelines for Sequence Diagrams:

- 1. Prepare at least one scenario per use case.**
  - Steps in Scenario should be logical commands, not industrial button clicks.
- 2. Abstract the scenario into sequence diagrams.**
  - Shows the contribution of each actors.
- 3. Divide complex interactions**
  - Break large interaction into smaller tasks.
- 4. Prepare a sequence diagram for each error condition**
  - Shows system response to error condition



## 7.3 Activity Models

Def: “An **Activity diagram** shows the sequence of steps that make up a complex process, such as an algorithm or workflow”.

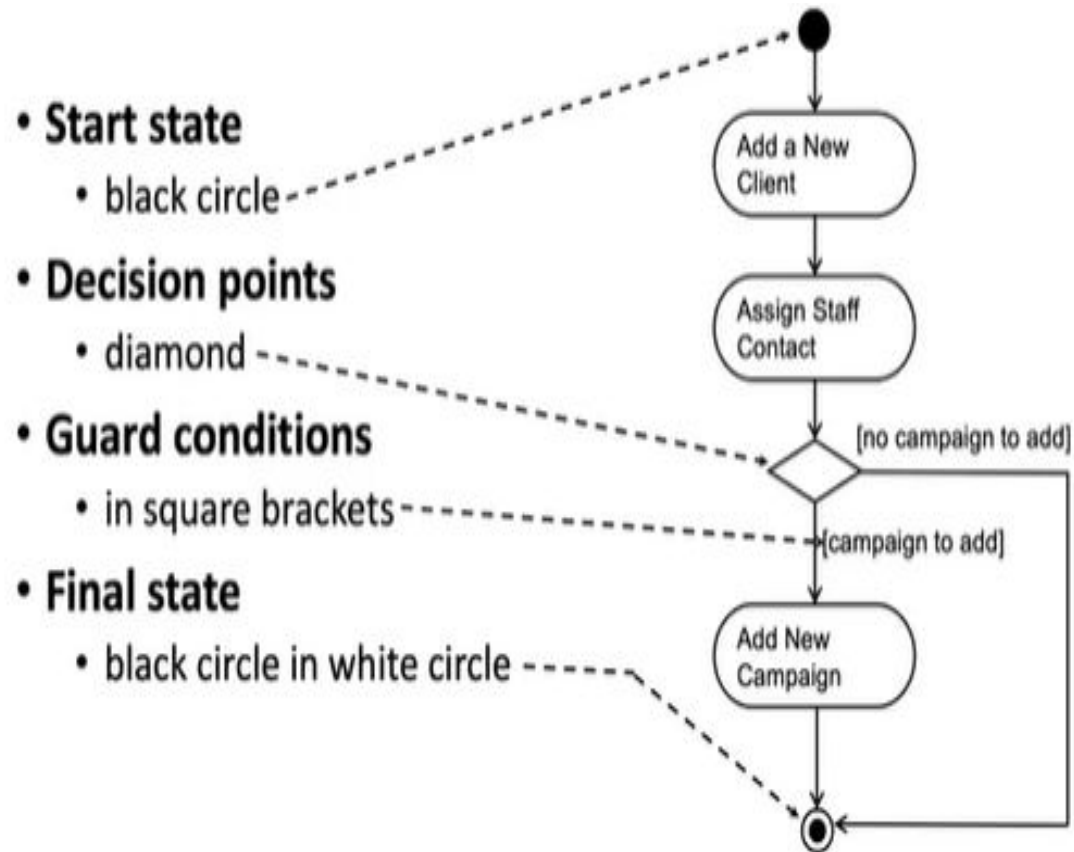
An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system.

- Activity diagram shows flow of control, similar to a sequence diagram, but focuses on operations rather than on objects.
- Activity diagrams are most useful during the early stages of designing algorithms and workflows.
- Activity diagram is like a traditional flowchart in that it shows the flow of control from step to step

## 7.3 Activity Models

1. Activities
2. Branches
3. Initiation and termination
4. Concurrent Activities
5. Executable Activities

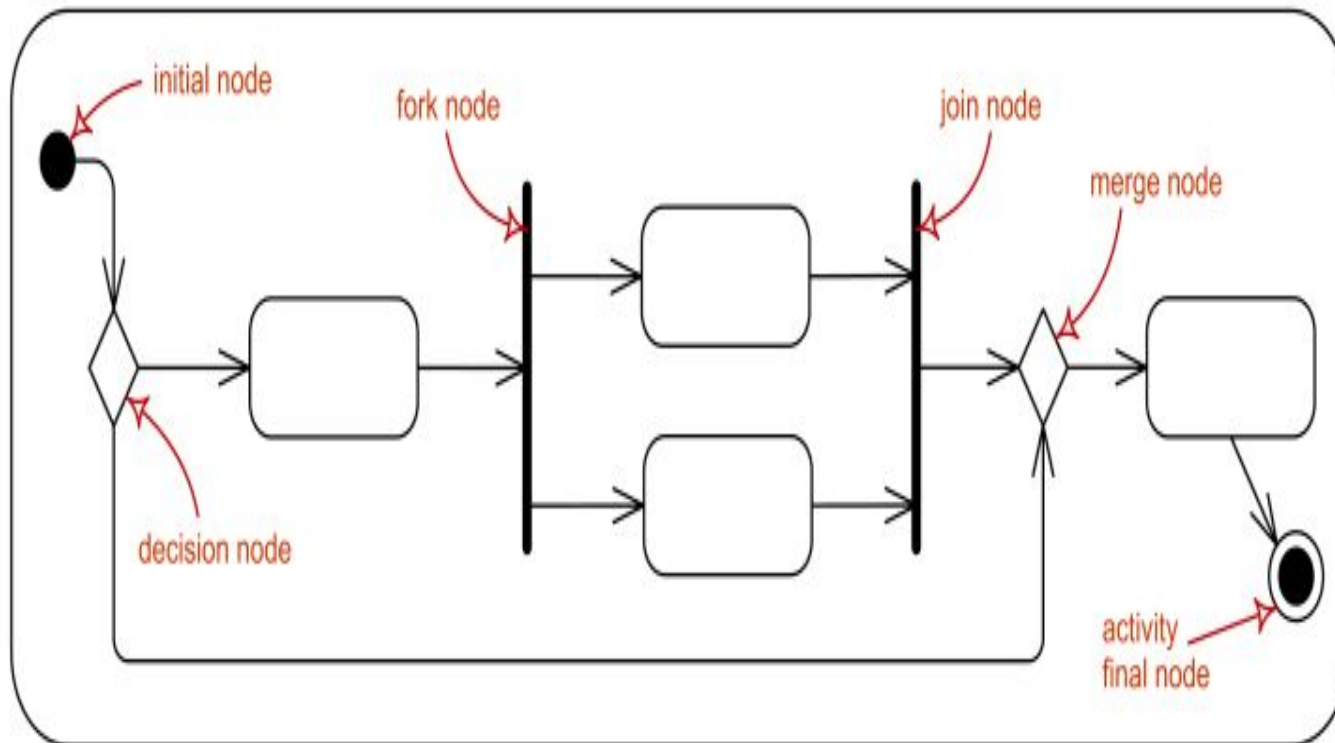
# Notation Initiation and Termination






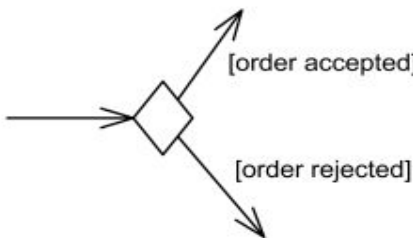
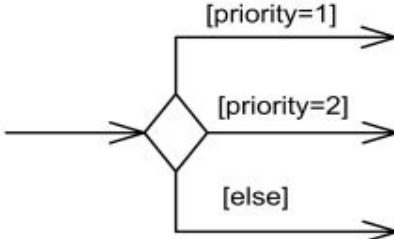
# Activity Models

- Activity Diagram Notation Summary

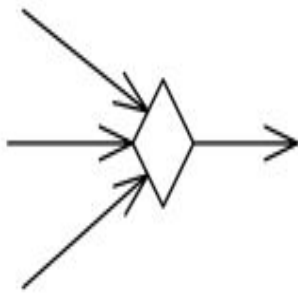
## Control Nodes



*Activity control nodes overview.*

<b>Initial Node</b>	
 <p><i>Activity initial node.</i></p>	<p><b>Initial node</b> is a control node at which flow starts when the activity is invoked. Activity may have more than one initial node. Initial nodes are shown as a small solid circle.</p>
<b>Flow Final Node</b>	
 <p><i>Flow final node.</i></p>	<p><b>Flow final node</b> is a control final node that terminates a flow. The notation for flow final node is small circle with X inside.</p>
<b>Activity Final Node</b>	
 <p><i>Activity final node.</i></p>	<p><b>Activity final node</b> is a control final node that stops all flows in an <b>activity</b>. Activity final is new in UML 2.0. Activity final nodes are shown as a solid circle with a hollow circle inside. It can be thought of as a goal notated as "bull's eye," or target.</p>
<b>Decision</b>	
 <p><i>Decision node with two outgoing edges with guards.</i></p>	<p><b>Decision node</b> is a <b>control node</b> that accepts tokens on one or two <b>incoming edges</b> and selects one <b>outgoing edge</b> from one or more outgoing flows.</p> <p>The notation for a decision node is a diamond-shaped symbol.</p>
 <p><i>Decision node with three outgoing edges and [else] guard.</i></p>	<p>For <b>decision points</b>, a predefined guard "<b>else</b>" may be defined for at most one outgoing edge.</p>

## Merge

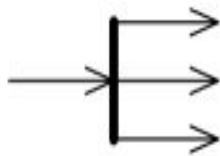


*Merge node with three incoming edges and a single outgoing edge.*

**Merge node** is a control node that brings together multiple incoming **alternate flows** to accept single outgoing flow. There is no joining of tokens. Merge **should not** be used to synchronize **concurrent flows**.

The notation for a merge node is a diamond-shaped symbol with two or more edges entering it and a single activity edge leaving it.

## Fork

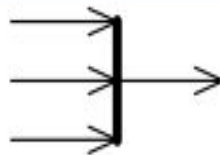


*Fork node with a single activity edge entering it, and three edges leaving it.*

**Fork node** is a control node that has one incoming edge and multiple outgoing edges and is used to split incoming flow into multiple **concurrent** flows.

The notation for a **fork node** is a line segment with a single activity edge entering it, and two or more edges leaving it.

## Join Node

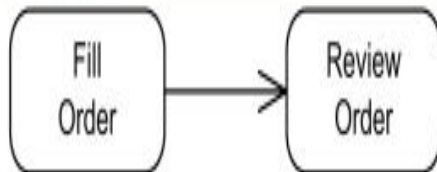


*Join node with three activity edges entering it, and a single edge leaving it.*

**Join node** is a control node that has multiple incoming edges and one outgoing edge and is used to synchronize incoming concurrent flows.

The notation for a join node is a line segment with several activity edges entering it, and only one edge leaving it.

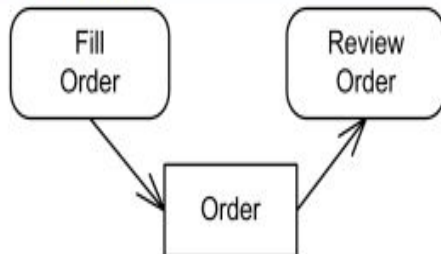
### Activity Edge



*Activity edge connects Fill Order and Review Order.*

**Activity edge** could be **control edge** or **data flow edge** (aka **object flow edge**). Both are notated by an open arrowhead line connecting activity nodes.

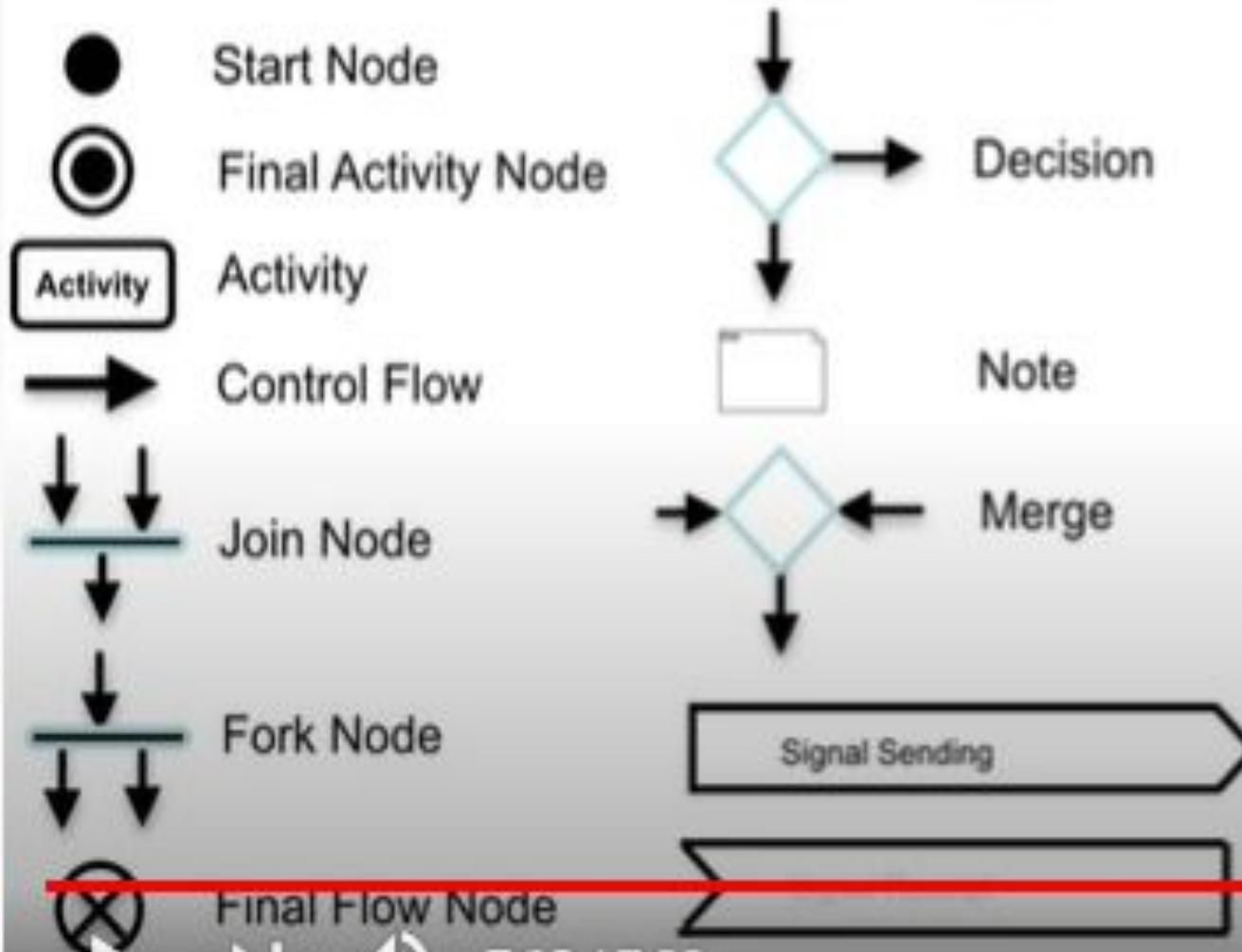
### Object Flow Edge



*Data flow of Orders between Fill Order and Review Order actions*

**Object flow edges** are activity edges used to show data flow between action nodes. Object flow edges have no specific notation.

# Symbols





# Activity diagram for an order management system

- Example of an activity diagram for order management system. In the diagram, four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and is mainly used by the business users

Following diagram is drawn with the four main activities –

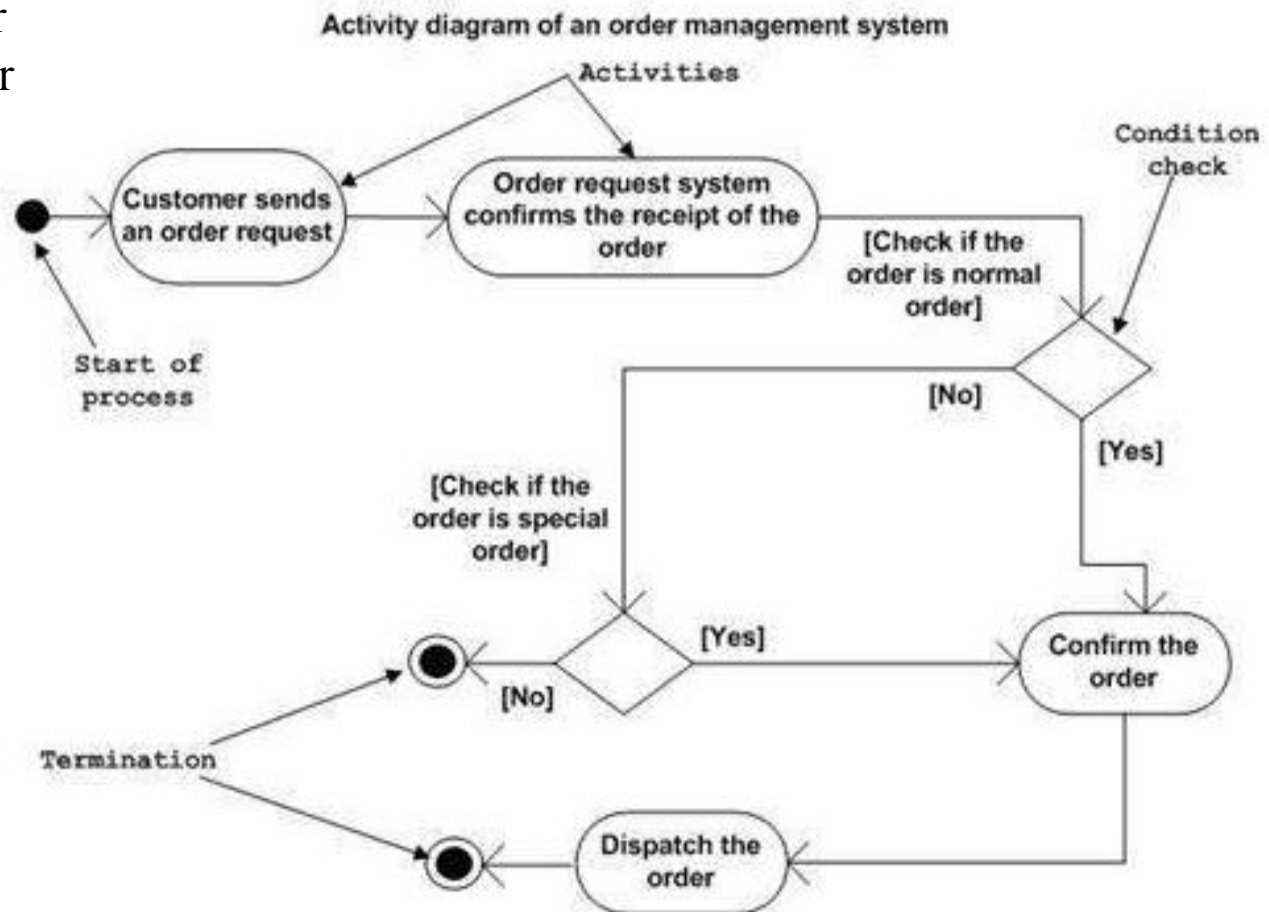
- Send order by the customer
- Receipt of the order
- Confirm the order
- Dispatch the order

After receiving the order request, condition checks are performed to check if it is normal or special order. After the type of order is identified, dispatch activity is performed and that is marked as the termination of the process.

# Activity diagram for an order management system

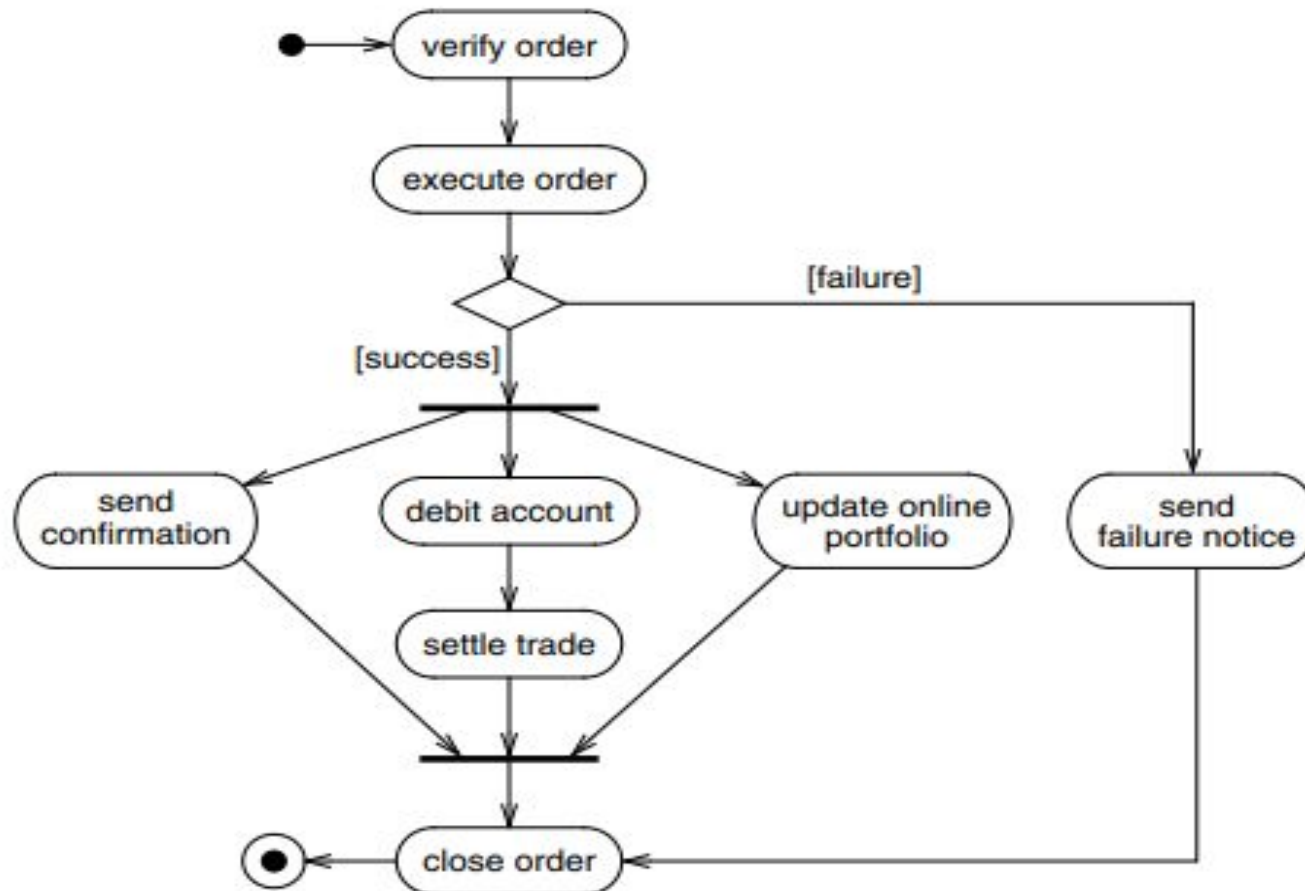
Following diagram is drawn with the four main activities –

- Send order by the customer
- Receipt of the order
- Confirm the order
- Dispatch the order



## 7.3 Activity Models

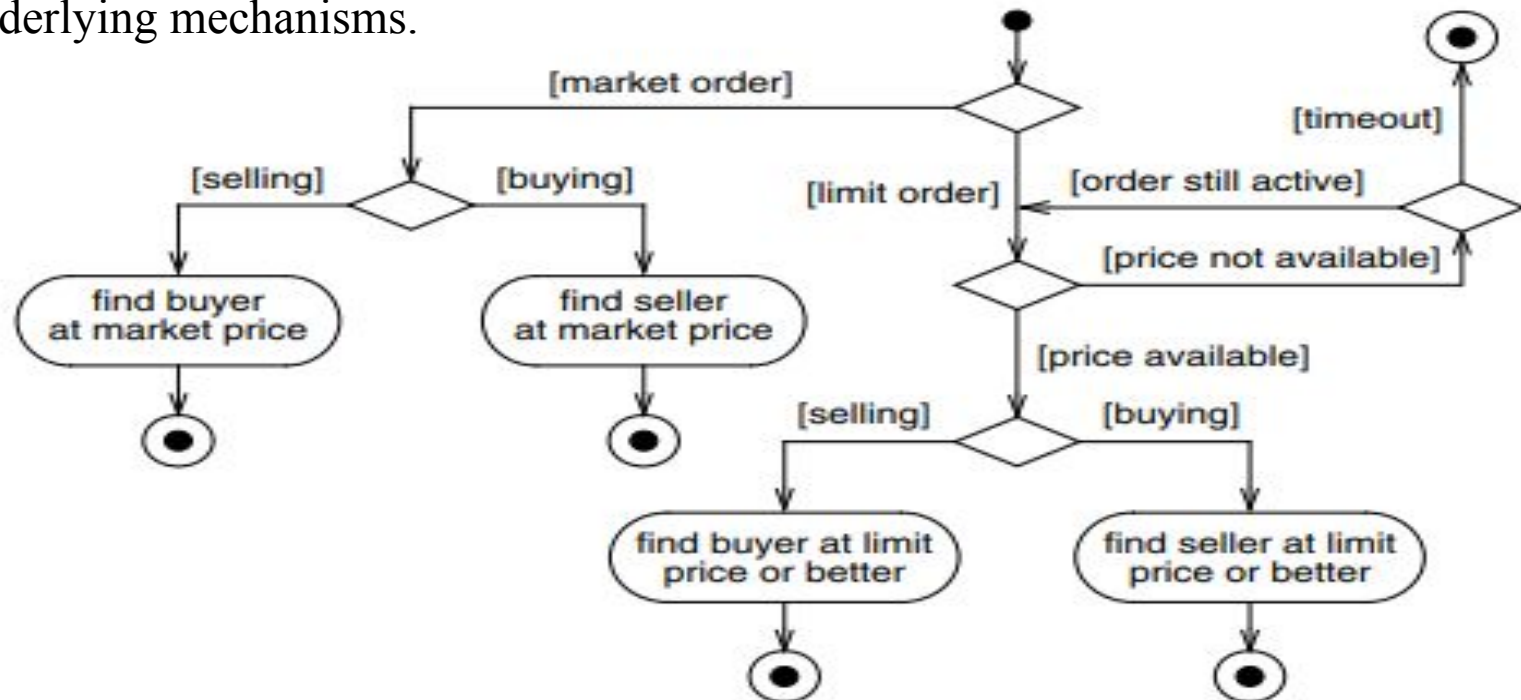
Figure 7.9 shows an activity diagram for the processing of a stock trade order that has been received by an online stock broker. The elongated ovals show activities and the arrows show their sequencing. The diamond shows a decision point and the heavy bar shows splitting or merging of concurrent threads



**Figure 7.9 Activity diagram for stock trade processing.** An activity diagram shows the sequence of steps that make up a complex process.

## 7.3 Activity Models

An activity may be decomposed into finer activities. For example, Figure 7.10 expands the execute order activity of Figure 7.9. It is important that the activities on a diagram be at the same level of detail. For example, in Figure 7.9 execute order and settle trade are similar in detail; they both express a high-level operation without showing the underlying mechanisms.



**Figure 7.10** Activity diagram for *execute order*. An activity may be decomposed into finer activities.

## 7.2.6 Guidelines for Activity Models:

1. **Don't misuse activity diagrams** :Activity diagrams supplement the object-oriented focus of UML models and should not be used as an excuse to develop software via flowcharts.
2. **Level Diagrams:** Activities on a diagram should be at a consistent level of detail. Place additional detail for an activity in a separate diagram.
3. **Be careful with branches and conditions:** If there are conditions, at least one must be satisfied when an activity completes—consider using an else condition. In undeterministic models, it is possible for multiple conditions to be satisfied—otherwise this is an error condition
4. **Be careful with concurrent activities:** Concurrency means that the activities can complete in any order and still yield an acceptable result. Before a merge can happen, all inputs must first complete
5. **Consider executable activity diagrams:** Executable activity diagrams can help developers understand their systems better.

# **Unit-III**

## **Advanced Interaction Modeling**

# Topics Covered

## □ **Use case Relationships**

- 1) Include Relationship
- 2) Extend Relationship
- 3) Generalization
- 4) Combinations of Use Case Relationships
- 5) Guidelines for Use Case Relationships

## □ **Procedural Sequence Models**

- 1) Sequence Diagrams with Passive Objects
- 2) Sequence Diagrams with Transient Objects
- 3) Guidelines for Procedural Sequence Models

## □ **Special Constructs for Activity Models**

- 1) Sending and Receiving Signals
- 2) Swimlanes
- 3) Object Flows

# Use case Relationships

Complex use cases can be built from smaller pieces of use case relationship.

There are three kinds of relationships between use cases :

- Include Relationship
- Extend Relationship
- Generalization Relationship

**Include Relationship** :The include relationship incorporates one use case within the behavior sequence of another use case

- An included use case is like a subroutine—it represents behavior that would otherwise have to be described repeatedly.
- The UML notation for an include relationship is a dashed arrow from the source (including) use case to the target (included) use case. The keyword «include» annotates the arrow

The **include** relationship could be used:

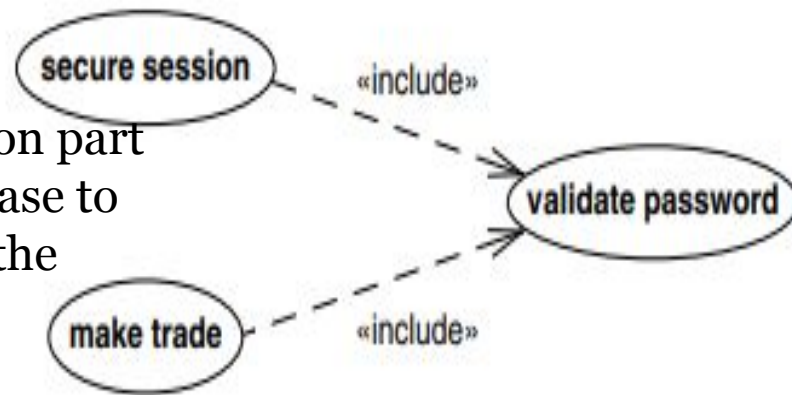
- to simplify large use case by splitting it into several use cases,
- to extract **common parts** of the behaviors of two or more use cases.



# Include Relationship

**Figure 8.1** shows an example from an online stock brokerage system. Part of establishing a secure session is validating the user password. In addition, the stock brokerage system validates the password for each stock trade. Use cases **secure session** and **make trade** both include use case **validate password**.

When two or more use cases have some **common behavior**, this common part could be extracted into a separate use case to be included back by the use cases with the UML **include** relationship.



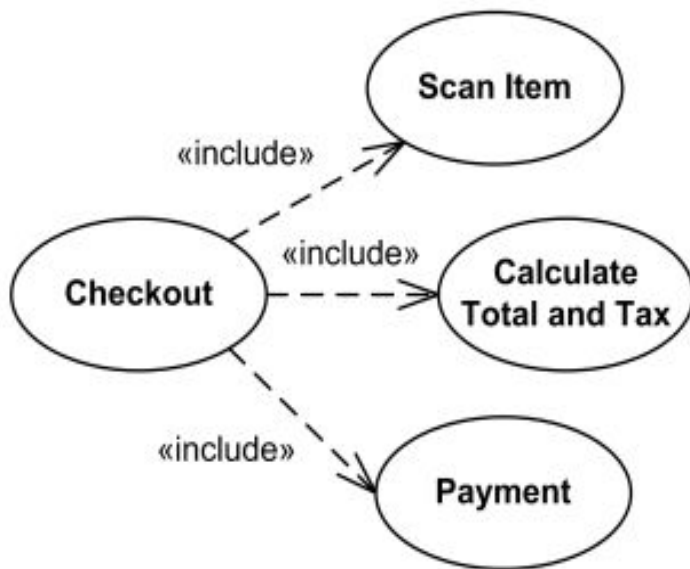
**Figure 8.1** Use case inclusion. The *include* relationship lets a base use case incorporate behavior from another use case.

# Example

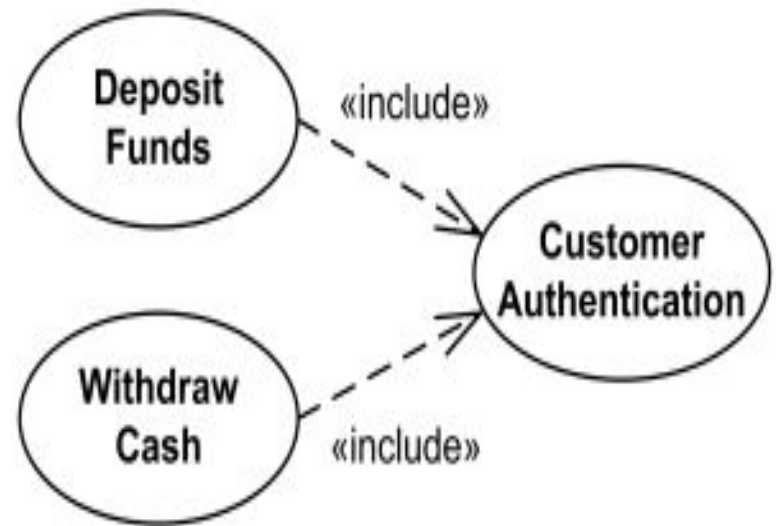
**Include** relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case.

The arrow is labeled with the keyword «include».

Note, that including Checkout use case becomes incomplete by itself and requires included use cases to be complete.



*Checkout use case includes several use cases - Scan Item, Calculate Total and Tax, and Payment*



*Deposit Funds and Withdraw Cash use cases include Customer Authentication use case.*

# Extend Relationship

- The extend relationship adds incremental behavior to a use case. It is like an include relationship looked at from the opposite direction, in which the extension adds itself to the base, rather than the base explicitly incorporating the extension.
- The include and extend relationships both add behavior to a base use case.
- The UML notation for an extend relationship is a dashed arrow from the extension use case to the base use case.
- The keyword «extend» annotates the arrow.

# Extend Relationship

Figure 8.2 shows the base use case **trade stocks** for a stock brokerage system. The base use case permits simple purchases and sales of a stock at the market price. The brokerage system adds three capabilities: buying a stock on margin, selling a stock short, and placing a limit on the transaction price. The use case trade options also has an extension for placing a limit on the transaction price



**Figure 8.2** Use case extension. The *extend* relationship is like an *include* relationship looked at from the opposite direction. The extension adds itself to the base.

# Generalization

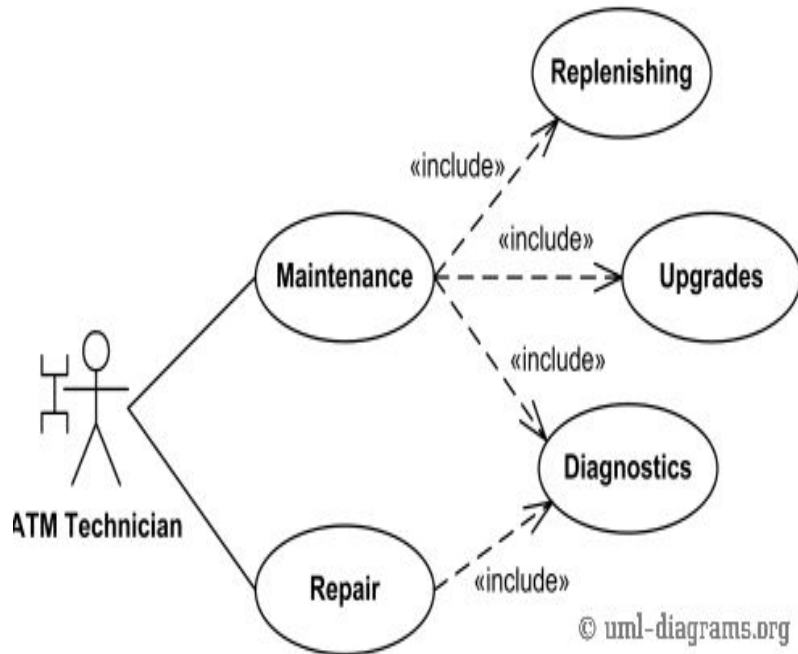
- It is a parent-child relationship between two or more use case.
- It is a kind of relationship in which the properties of the parent use case are inherited by the child use case.
- Generalization can show specific variations on a general use case, analogous to generalization among classes. **A parent use case represents a general behavior sequence. Child use cases specialize the parent by inserting additional steps or by refining steps**
- The UML indicates generalization by an arrow with its tail on the child use case and a triangular arrowhead on the parent use case, the same notation that is used for classes

- For example, an online stock brokerage system (Figure 8.3) might specialize the general use case make trade into the child use cases trade bonds, trade stocks, and trade option
- The parent use case contains steps that are performed for any kind of trade, such as entering the trading password.
- Each child use case contains the additional steps particular to a specific kind of trade, such as entering the expiration date of an option.

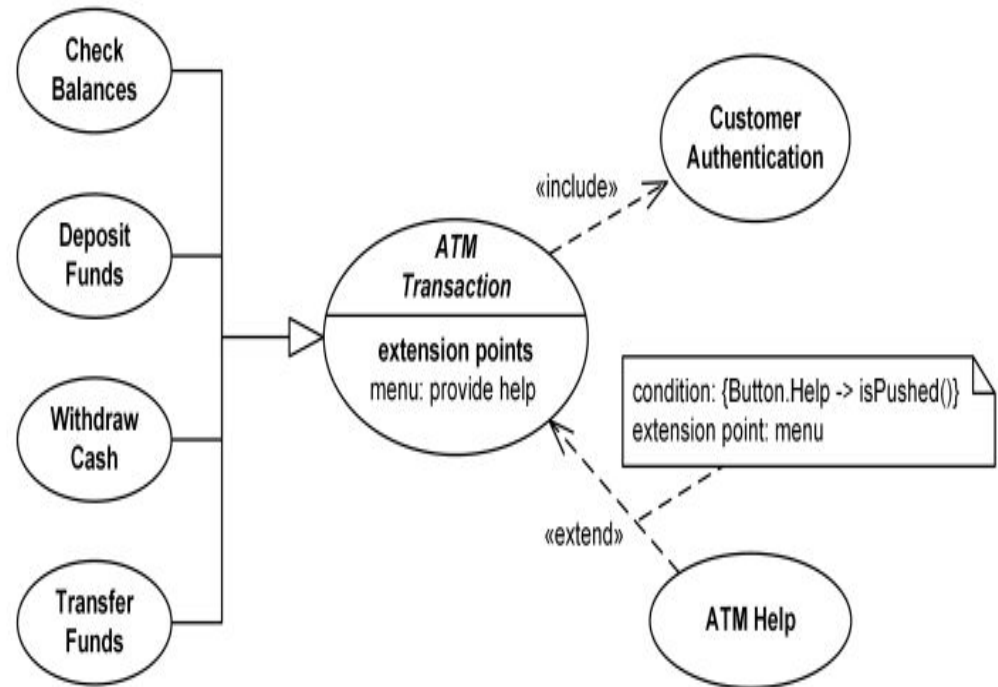


**Figure 8.3 Use case generalization.** A parent use case has common behavior and child use cases add variations, analogous to generalization among classes.

# Include ,Extend, Generalization for ATM example



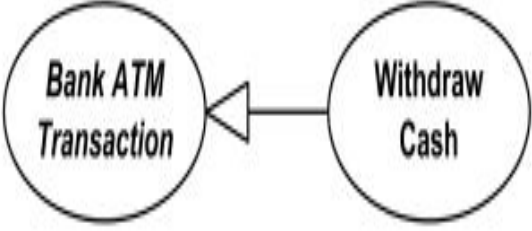
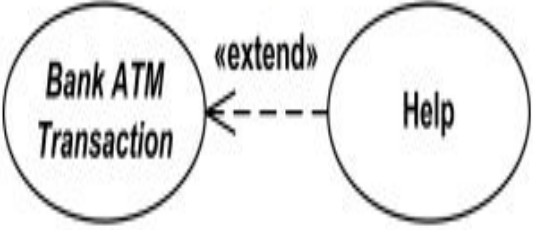
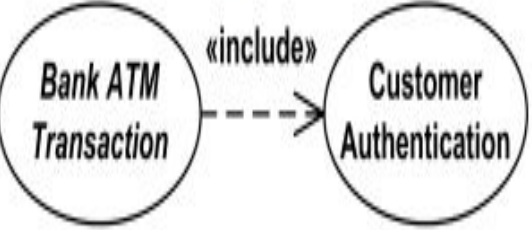
*Bank ATM Maintenance, Repair, Diagnostics Use Cases Example.*



*Bank ATM Transactions and Customer Authentication Use Cases Example.*



# Use case Relationships compared

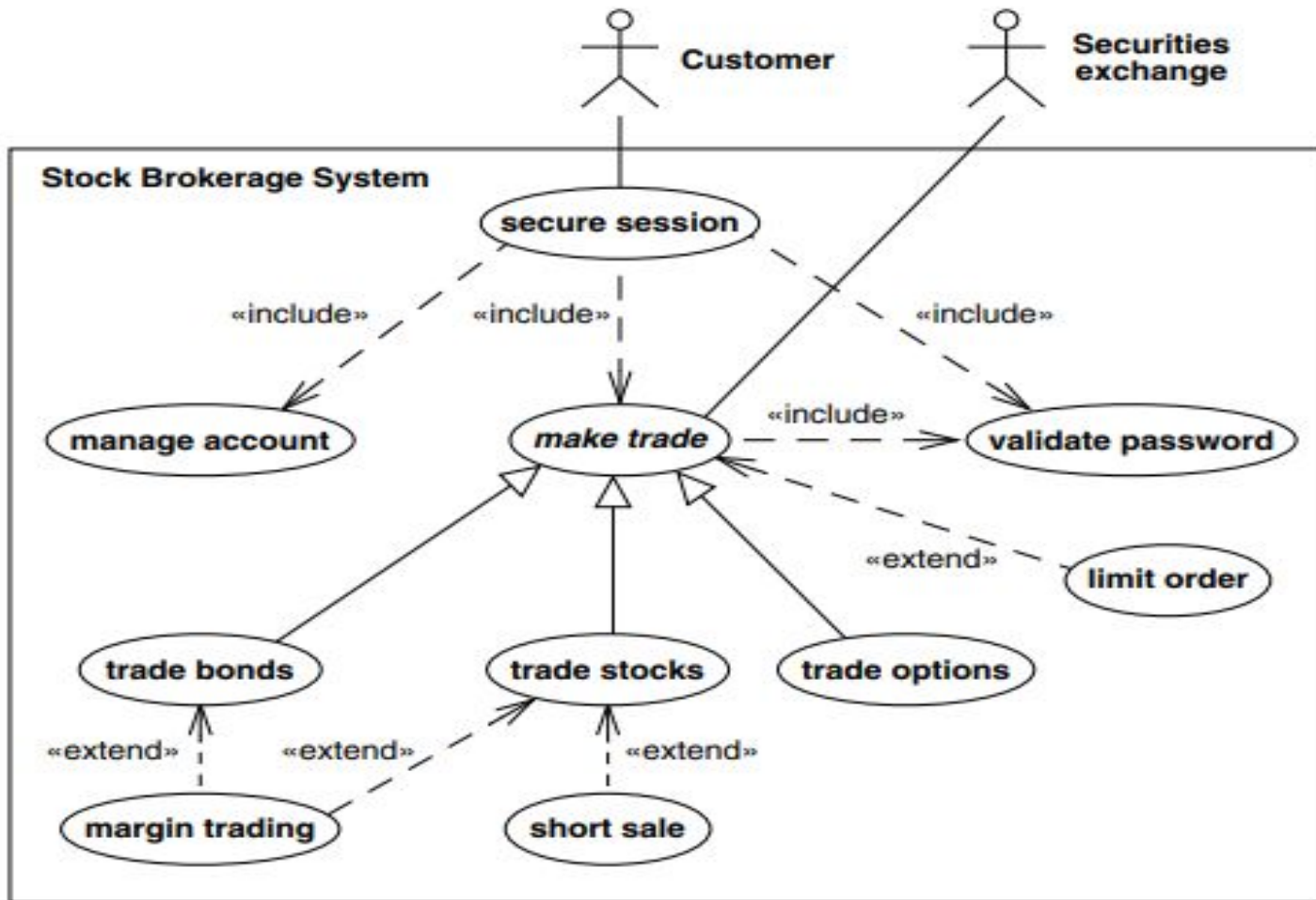
Generalization	Extend	Include
		
Base use case could be <b>abstract use case</b> (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete ( <b>abstract use case</b> ).
Specialized use case is required, not optional, if base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



# Combinations of Use Case Relationships

- A single diagram may combine several kinds of use case relationships.
- Figure 8.4 shows a use case diagram from a stock brokerage system.
- The secure session use case includes the behavior of the validate password, make trade, and manage account use cases.
- Make trade is an abstract parent with the children—trade bonds, trade stocks, and trade options.
- Use case make trade also includes the behavior of validate password. The brokerage system validates the password once per session and additionally for every trade
- The use case margin trading extends both trade bonds and trade stocks—a customer may purchase stocks and bonds on margin, but not options.
- Use case limit order extends abstract use case make trade—limit orders apply to trading bonds, stocks, and options.
- We assume that a short sale is only permitted for stocks and not for bonds or options.

# Combinations of Use Case Relationships



**Figure 8.4 Use case relationships.** A single use case diagram may combine several kinds of relationships.

# Guidelines for Use Case Relationships

1 . **Use case generalization:** If a use case comes in several variations, model the common behavior with an abstract use case and then specialize each of the variations.

2. **Use case inclusion:** If a use case includes a well-defined behavior fragment that is likely to be useful in other situations, define a use case for the behavior fragment and include it in the original use case. In most cases, you should think of the included use case as a meaningful activity but not as an end in itself. For example, validating a password is meaningful to users but has a purpose only within a broader context.

3. **Use case extension:** We can model the use case with optional features, model the base behavior as a use case and add features with the extend relationship. This permits the system to be tested and debugged without the extensions, which can be added later. Use the extend relationship when a system might be deployed in different configurations, some with the additional features and some without them.

# Guidelines for Use Case Relationships

4. **Include relationship vs. extend relationship** :The include relationship, however, implies that the included behavior is a necessary part of a configured system (even if the behavior is not executed every time), whereas the extend relationship implies that a system without the added behavior would be meaningful (even if there is no intention to configure it that way).

OR

"The **include relationship** is intended for reusing behavior modeled by another use case, whereas the **extend relationship** is intended for adding parts to existing use cases as well as for modeling optional system services“




# Procedural Sequence Models

- Sequence diagrams containing independent objects, all of which are active concurrently.
- An object remains active after sending a message and can respond to other messages without waiting for a response.
- This is appropriate for high-level models.
- However, most implementations are procedural and limit the number of objects that can execute at a time.
- The UML has elaborations for sequence diagrams to show procedural calls.
- With procedural code all objects are not constantly active. Most objects are passive and do not have their own threads of control.
- A passive object is not activated until it has been called. Once the execution of an operation completes and control returns to the caller, the passive object becomes inactive

## 7.2 Sequence Models

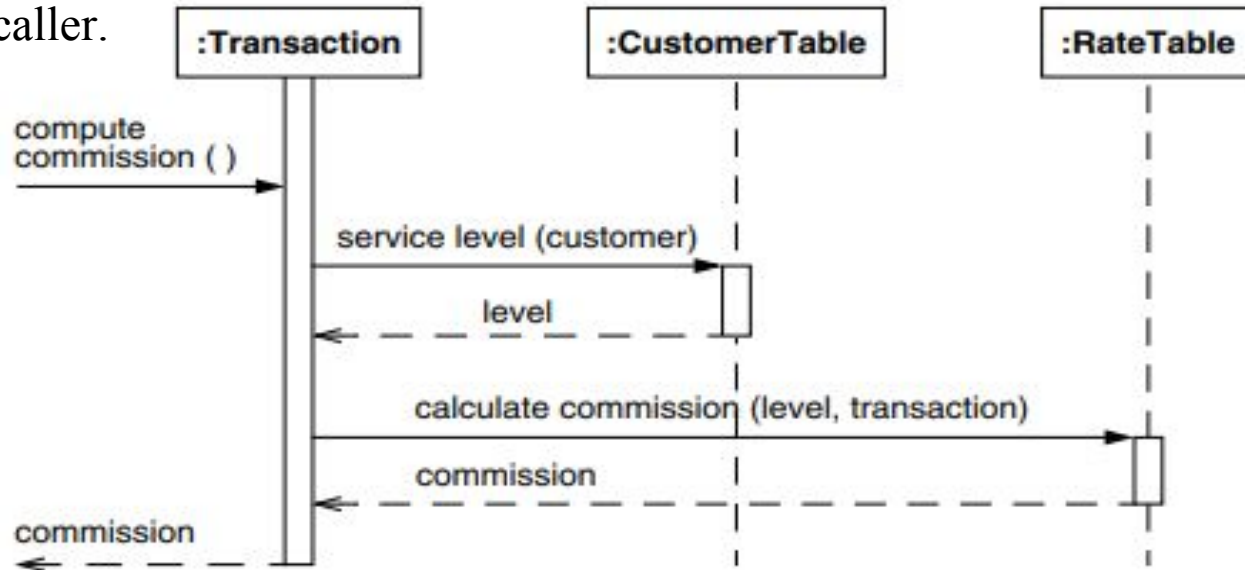
### 7.2.2 Sequence Diagrams:

#### Message Type Notations

Message	Description
	<b>Synchronous:</b> A synchronous message between active objects indicates wait semantics; the sender waits for the message to be handled before it continues. This typically shows a method call.
	<b>Asynchronous:</b> With an asynchronous flow of control, there is no explicit return message to the caller. An asynchronous message between objects indicates no-wait semantics; the sender does not wait for the message before it continues. This allows objects to execute concurrently.
	<b>Reply:</b> This shows the return message from another message.

# Sequence Diagrams with Passive Objects

**The transaction object** receives a request to compute its commission. It obtains the customer's service level from the customer table, then asks the rate table to compute the commission based on the service level, after which it returns the commission value to the caller.

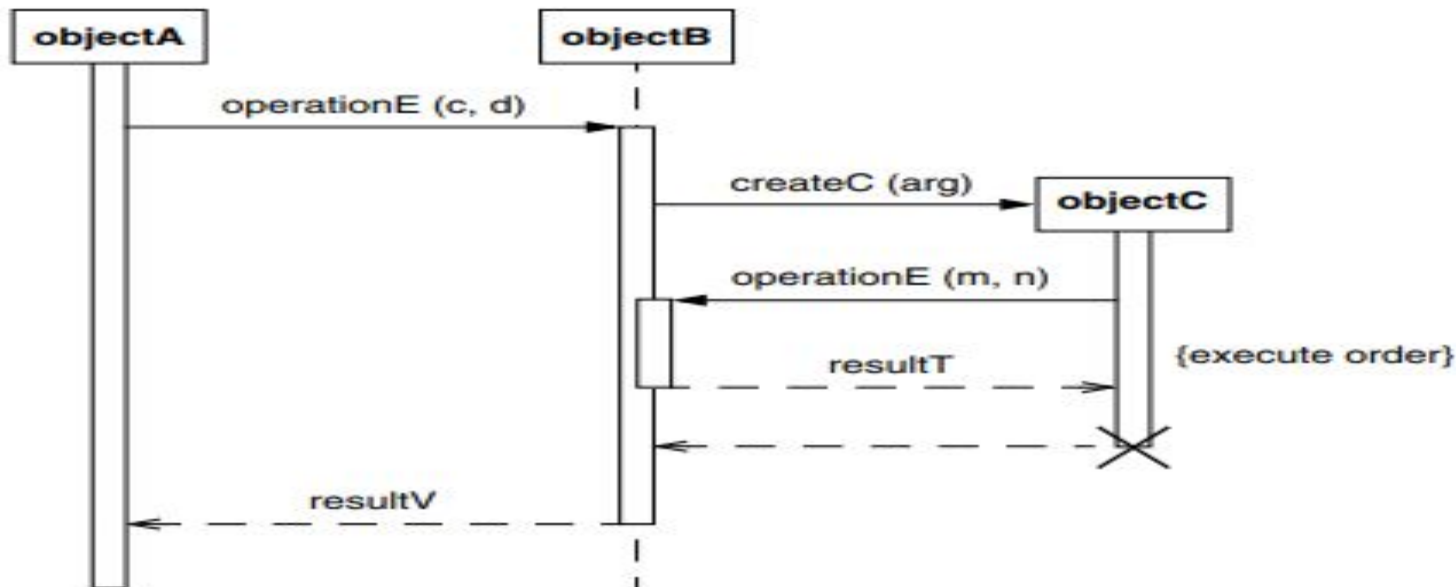


**Figure 8.5** Sequence diagram with passive objects. Sequence diagrams can show the implementation of operations.

- The UML shows the period of time for an object's execution as a thin rectangle. This is called the **activation or focus of control**.
- The period of time when an object exists but is not active is shown as a dashed line.
- The entire period during which the object exists is called the **lifeline**, as it shows the lifetime of the object.

# Sequence Diagrams with Transient Objects

- ObjectA is an active object that initiates an operation. Because it is active, its activation rectangle spans the entire time shown in the diagram.
- ObjectB is a passive object that exists during the entire time shown in the diagram, but it is not active for the whole time.
- The UML shows its existence by the dashed line (the lifeline) that covers the entire time period..



**Figure 8.6** Sequence diagram with a transient object. Many applications have a mix of active and passive objects. They create and destroy objects.



# Sequence Diagrams with Transient Objects

- Fig 8.6 ObjectB's lifeline broadens into an activation rectangle when it is processing a call
- During part of the time, it performs a **recursive operation**, as shown by the doubled activation rectangle between the call by objectC on operationE and the return of the result value. ObjectC is created and destroyed during the time shown on the diagram, so its lifeline does not span the whole diagram.
- The notation for a call is an arrow from the calling activation to the activation created by the call. The tail of the arrow is somewhere along the rectangle of the calling activation. The arrowhead aligns with the top of the rectangle of the newly created activation, because the call creates the activation. The filled arrowhead indicates a call
- The UML shows a return by a dashed arrow from the bottom of the called activation to the calling activation. Not all return arrows have result values—for example, the return from objectC to objectB.

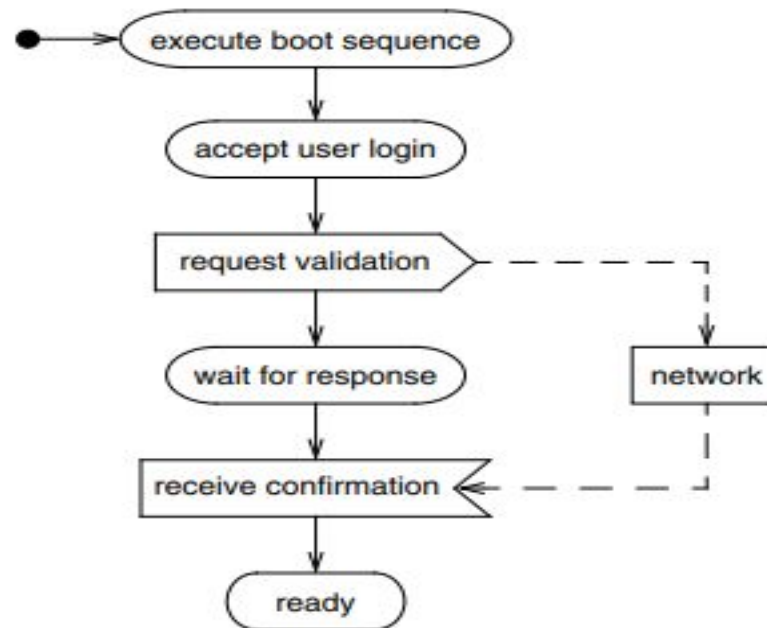
# Guidelines for Procedural Sequence Models

- **Active vs. passive objects:** Differentiate between active and passive objects. Most objects are passive and lack their own thread of control. By definition, active objects are always activated and have their own focus of control.
- **Advanced features:** Advanced features can show the implementation of sequence diagrams. Be selective in using these advanced features. Only show implementation details for difficult or especially important sequence diagrams.

# Special Constructs for Activity Models

Activity diagrams have additional notation that is useful for large and complex applications.

**Sending and Receiving Signals:** Consider a workstation that is turned on. It goes through a boot sequence and then requests that the user log in. After entry of a name and password, the workstation queries the network to validate the user. Upon validation, the workstation then finishes its startup process. Figure 8.7 shows the corresponding activity diagram.



**Figure 8.7 Activity diagram with signals.** Activity diagrams can show fine control via sending and receiving events.

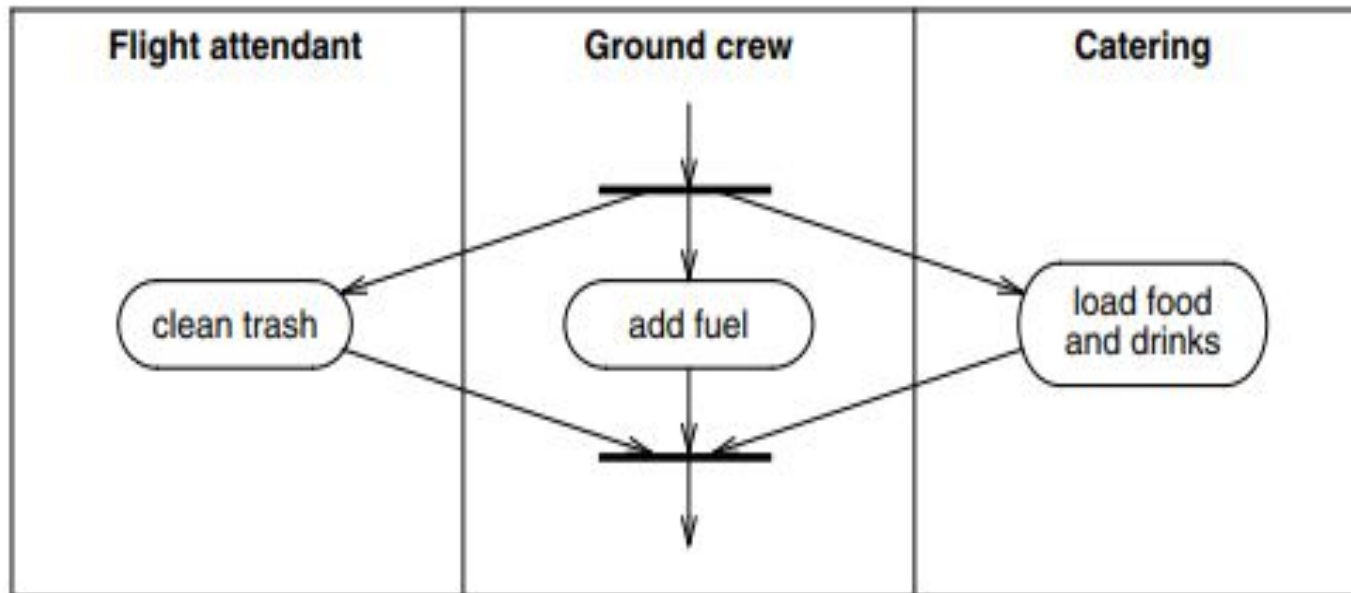
# Special Constructs for Activity Models

- The UML shows the sending of a signal as a **convex pentagon**. When the preceding activity completes, the signal is sent, then the next activity is started.
- The UML shows the receiving of a signal as a **concave pentagon**. When the preceding activity completes, the receipt construct waits until the signal is received, then the next activity starts.

# Swimlanes

- Swimlanes are columns with solid vertical lines on each side. Each swimlane represents the responsible class, person, or organizational unit.
- Partitioning an activity diagram by dividing it into columns and lines. Each column is called a swimlane by analogy to a swimming pool. Placing an activity within a particular swimlane indicates that it is performed by a person or persons within the organization.
- Lines across swimlane boundaries indicate interactions among different organizations, which must usually be treated with more care than interactions within an organization.
- The horizontal arrangement of swimlanes has no inherent meaning, although there may be situations in which the order has meaning.

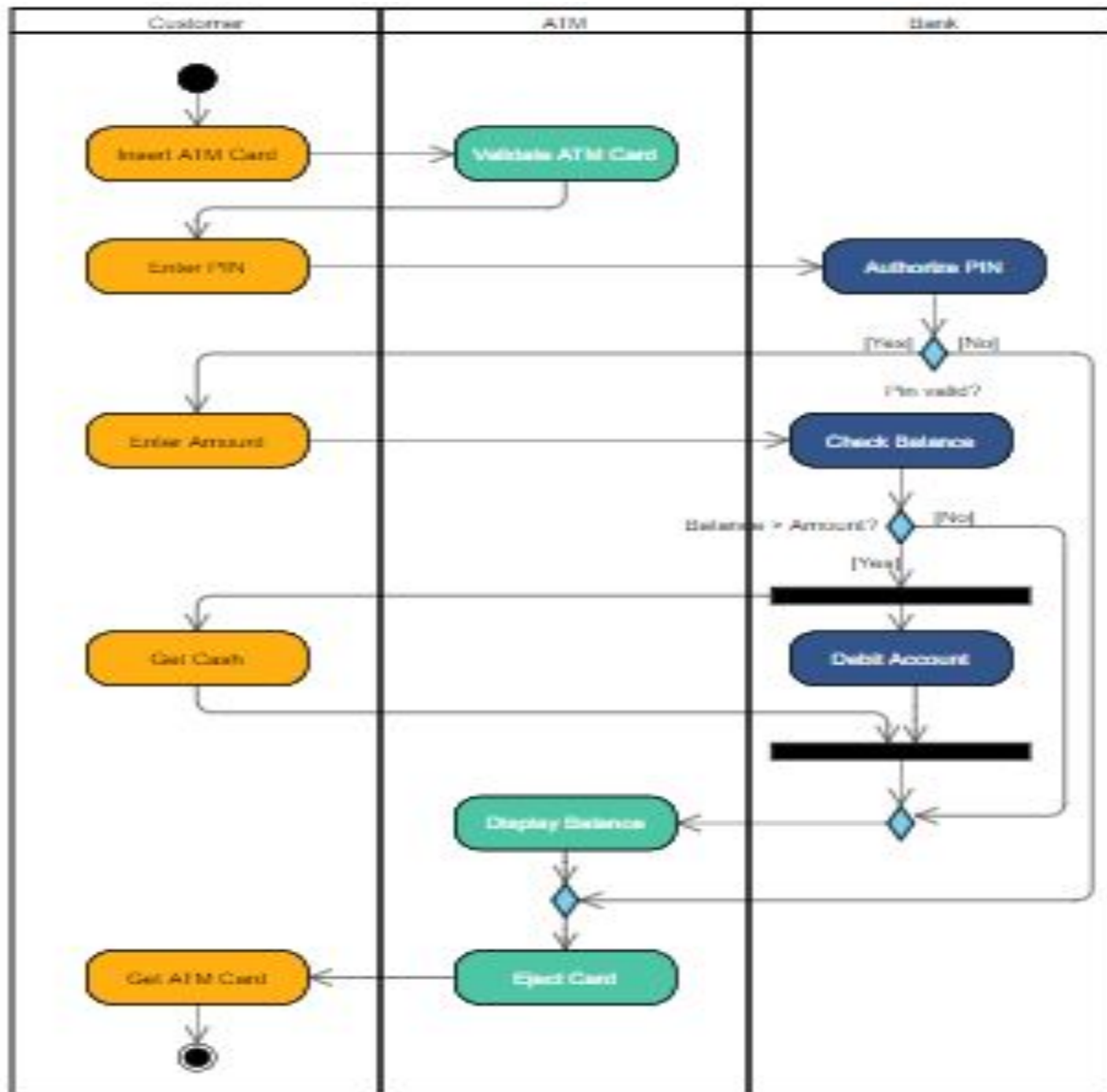
# Swimlanes :Example



**Figure 8.8** Activity diagram with swimlanes. Swimlanes can show organizational responsibility for activities.

Figure 8.8 shows a simple example for servicing an airplane. The flight attendants must clean the trash, the ground crew must add fuel, and catering must load food and drink before a plane is serviced and ready for its next flight.

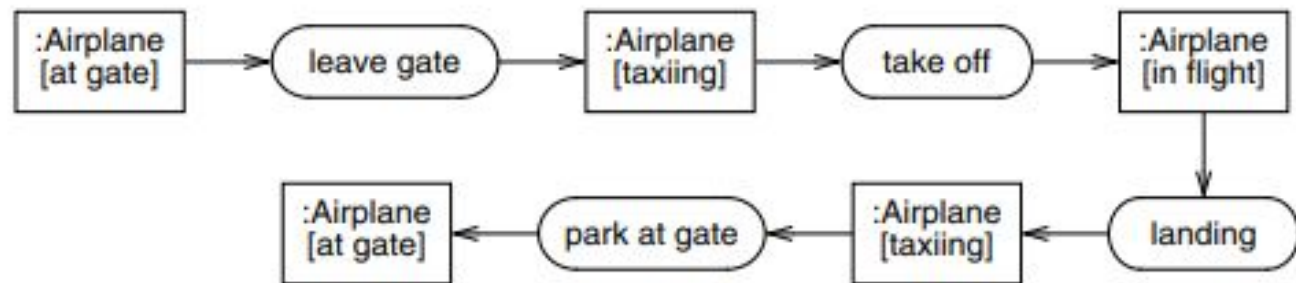
# Swimlanes :ATM Example



# Object Flows

- An activity diagram can show objects that are inputs to or outputs from the activities
- UML shows an object value in a particular state by placing the state name in square brackets following the object name.
- In Figure 8.9 an airplane goes through several states as it leaves the gate, flies, and then lands again.

In below figure the same object goes through several states during the execution of an activity diagram. The same object may be an input to or an output from several activities, but on closer examination an activity usually produces or uses an object in a particular state



**Figure 8.9 Activity diagram with object flows.** An activity diagram can show the objects that are inputs or outputs of activities.

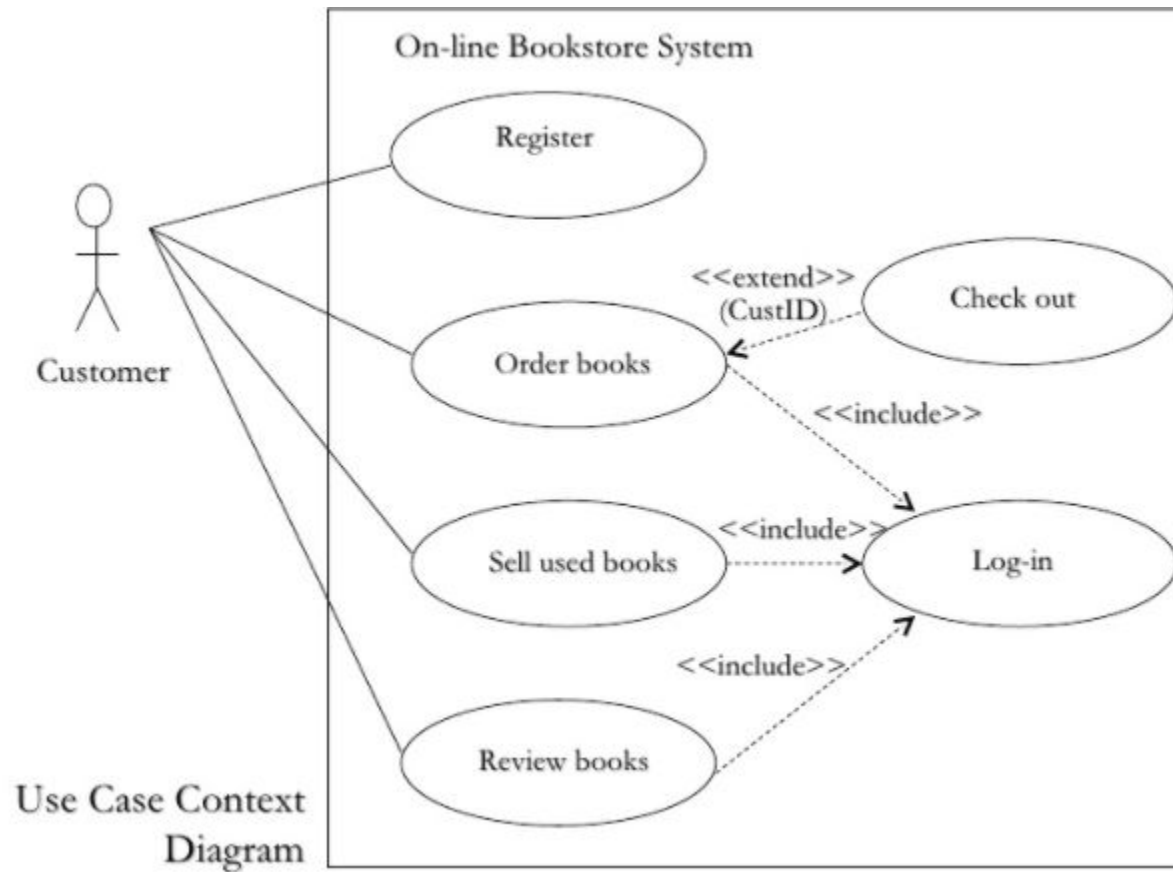


# Questions.

- Consider a physical bookstore, such as in a shopping mall. a. (2) List three actors that are involved in the design of a checkout system. Explain the relevance of each actor.
- b. (2) One use case is the purchase of items. Take the perspective of a customer and list another use case at a comparable level of abstraction. Summarize the purpose of each use case with a sentence.
- c. (4) Prepare a use case diagram for a physical bookstore checkout system. d. (3) Prepare a normal scenario for each use case. Remember that a scenario is an example, and need not exercise all functionality of the use case. e. (3) Prepare an exception scenario for each use case. f. (5) Prepare a sequence diagram corresponding to each scenario in (d)

# ONLINE BOOK STORE SYSTEM

- The steps involved: -
  - Identify the actor : CUSTOMER
  - Identify the use case for the actor:
    - CUSTOMER
      - REGISTER
      - LOG-IN
      - ORDER BOOKS
      - CHECK OUT
      - REVIEW BOOKS
      - SELL USED BOOKS
  - For each use case, determine include and extend relationships, if any
    - A Customer **must** log-in first before he/she can order books, check out, review books or sell used books: include relationship
    - A Customer **can** proceed to check out after he/she has ordered books: extend relationship



# Use case description

Use Case	Description
Register	<p>A new CUSTOMER needs to first register into the system before performing any transaction.</p> <p>Actor/s: CUSTOMER</p> <p>Pre-condition: An unregistered CUSTOMER.</p> <p>Main flow of events:</p> <ol style="list-style-type: none"> <li>1. The CUSTOMER clicks the REGISTER button on the Home Page.</li> <li>2. The system displays the Register Page.</li> <li>3. The CUSTOMER enters all of the required information.</li> <li>4. The CUSTOMER clicks the SEND button.</li> <li>5. The system checks that all of the required information were entered. If                     <ul style="list-style-type: none"> <li>yes, the system update the CUSTOMER's record in the CUSTOMER and ACCOUNT tables in the database. System displays OK message.</li> </ul> </li> </ol>
Log-in	<p>A CUSTOMER needs to log-in before performing any transaction</p> <p>Post-condition: The new CUSTOMER has registered. The ACCOUNT and CUSTOMER tables are updated.</p> <p>Actor/s: CUSTOMER</p> <p>Pre-condition: A registered user.</p> <p>Main flow of events:</p> <ol style="list-style-type: none"> <li>1. The CUSTOMER clicks the Log-in button on the Home Page.</li> </ol>

# Use case description

Use Case	Description
Log-in (continue ...)	<p>2. The system displays the Log-in Page.</p> <p>3. The CUSTOMER enters his/her user ID and password.</p> <p>4. The CUSTOMER clicks the OK button.</p> <p>5. The system validates the log-in information against the ACCOUNT table in the database.</p> <p>6. CUSTOMER is an authorised user; the system displays the Personal Home Page to the CUSTOMER</p> <p>Post-condition: The CUSTOMER has been authorised to perform transactions.</p> <p>Alternate flow:</p> <p>1. The CUSTOMER clicks the Log-in button on the Home Page.</p> <p>2. The system displays the Log-in Page.</p> <p>3. The CUSTOMER enters his/her user ID and password.</p> <p>4. The CUSTOMER clicks the OK button.</p> <p>5. The system validates the log-in information against the ACCOUNT table in the database.</p> <p>6. CUSTOMER is not an authorised user; the system displays a pop-up message to inform the CUSTOMER.</p> <p>Post-condition:</p>

# Use case description

Use Case	Description
Order Books	<p>A CUSTOMER can order books to purchase.</p> <p>Actor/s: CUSTOMER</p> <p>Pre-condition: User have logged-in.</p> <p>Main flow of events:</p> <ol style="list-style-type: none"> <li>1. The CUSTOMER enters the keyword for a book and clicks the SEARCH button on the Personal Home Page.</li> <li>2. The system displays the matching books on the web Page.</li> <li>3. The CUSTOMER chooses the desired book and clicks the ADD TO SHOPPING CART button on the web page.</li> <li>4. The system adds the book into the CUSTOMER's Order table in the database.</li> </ol>
Check Out	<p>A CUSTOMER can purchase the books in his/her Shopping Cart.</p> <p>Post-condition: The ORDER table has been updated.</p> <p>Actor/s: CUSTOMER</p> <p>Pre-condition: The user have logged in and has at least one book in the Shopping Cart. Main flow of events:</p> <ol style="list-style-type: none"> <li>1. The CUSTOMER clicks the Check out button on the Web Page.</li> </ol>

Clip slide

# Use case description

Use Case	Description
Check Out (continue ...)	<p>2. The system displays the books in the ORDER table of the CUSTOMER on the web Page.</p> <p>3. The CUSTOMER checks the order list for any inconsistency. If nothing found, CUSTOMER clicks the PROCEED button.</p> <p>4. The system displays the Invoice page.</p> <p>5. The Customer enters the relevant credit card information and clicks the OK button.</p> <p>6. The system checks that the credit card is valid. Then, the system displays the Delivery Details page.</p> <p>7. The CUSTOMER chooses destination for delivery, along with delivery options. Then, he/she clicks the PROCEED button.</p> <p>8. The system will display the check-out information for confirmation.</p> <p>7. The CUSTOMER checks that all information is correct and then</p>
Sell used books	<p>A CUSTOMER can sell his/her used books.</p> <p>Actor/s: CUSTOMER</p> <p>6. The system sends a confirmation via CUSTOMER's e-mail.</p> <p>Pre-condition: The user have logged-in..</p> <p>Post condition: The ORDER table has been updated.</p>



# Use case description

Use Case	Description
Sell Used Books (continue ...)	<p>Main flow of events:</p> <ol style="list-style-type: none"> <li>1. The CUSTOMER clicks the Sell Used Books button on the Home Page.</li> <li>2. The system displays the Sell used books web page.</li> <li>3. The CUSTOMER enters the required information on the used books that he/she wants to sell.</li> <li>4. The CUSTOMER clicks the SEND button on the webpage.</li> <li>5. The system displays a confirmation page listing the information that the CUSTOMER has entered.</li> <li>6. The CUSTOMER checks that the information displayed are accurate.</li> </ol> <p>If yes, the CUSTOMER clicks the OK button on the web page.</p>
Review Books	<p><del>A CUSTOMER can review books.</del></p> <p>7. The system updates the USED BOOKS table in the database.</p> <p>Actor/s: CUSTOMER</p> <p>Pre-condition: User have logged-in..</p> <p>Post condition: The Used Books table has been updated.</p> <p>Main flow of events:</p> <ol style="list-style-type: none"> <li>1. The CUSTOMER enters the keyword to search for a book and then</li> </ol> <p>clicks the SEARCH button on the Personal Web Page.</p>



# Use case description

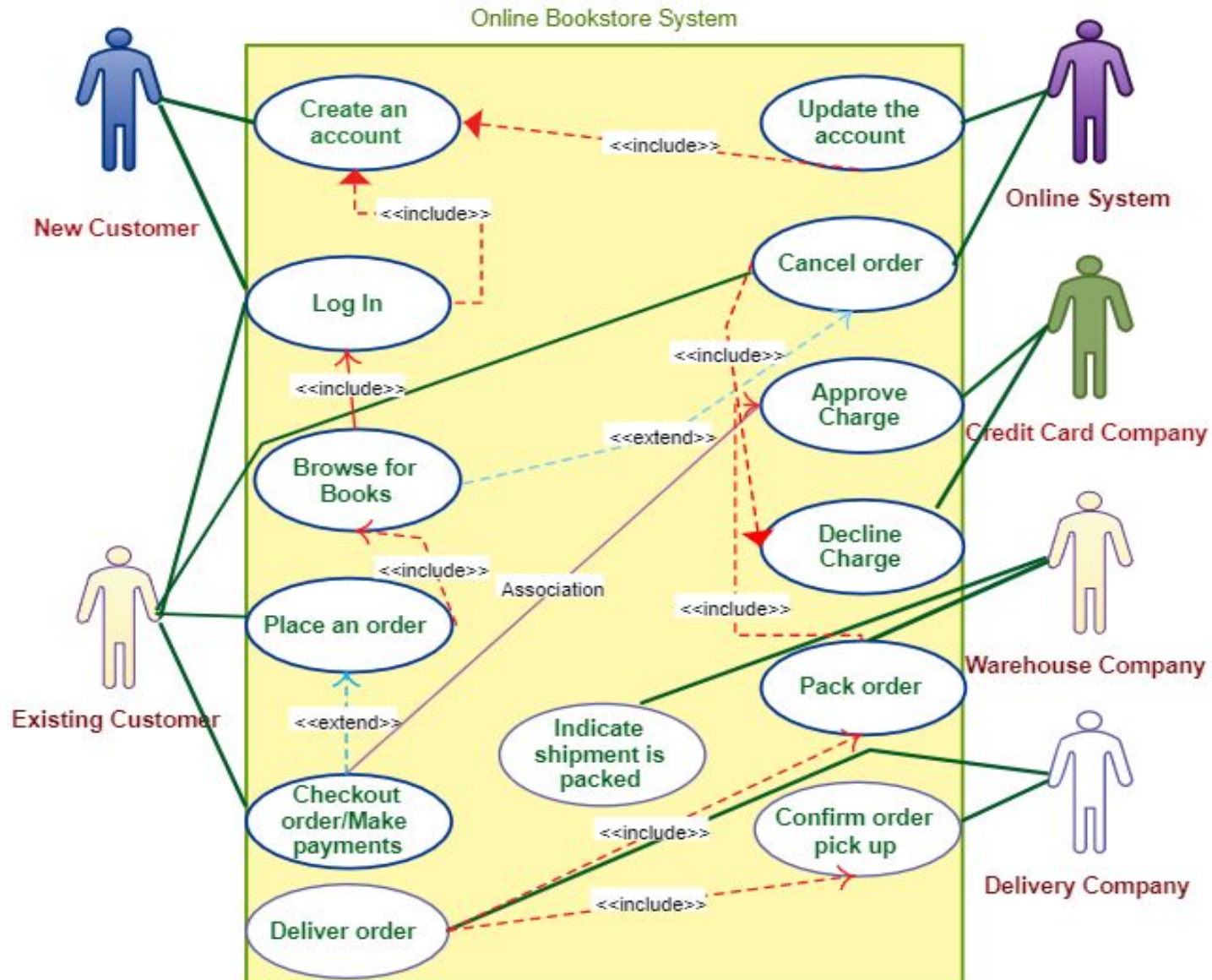
Use Case	Description
Review Books (continue ...)	<p>2. The system displays the matching books on the web Page.</p> <p>3. The CUSTOMER checks for the desired book and clicks on the chosen book icon.</p> <p>4. The system displays the book's detail in the Book Detail web page.</p> <p>5. The CUSTOMER clicks the REVIEW button on the web page.</p> <p>6. The system displays the Review Book web page.</p> <p>7. The CUSTOMER clicks on the desired star button and the click the</p> <p>OK button on the web page.</p> <p>8. The system calculates the overall rating of the book and updates the</p> <p>Book table in the database.</p> <p>9. The system displays the Book Detail web pages that have been</p>

updated.

Post-condition:

The BOOK and REVIEW tables are updated.

# Use case diagram



THANK YOU