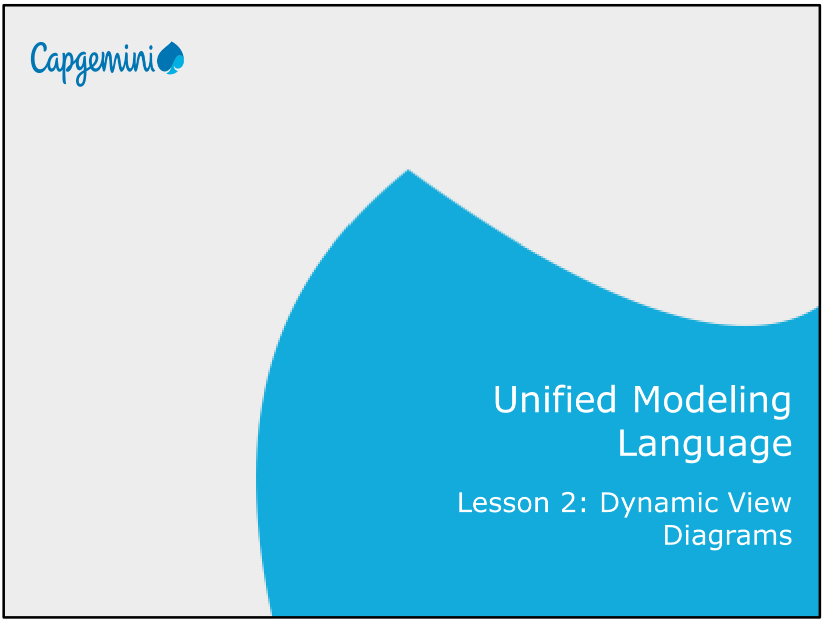
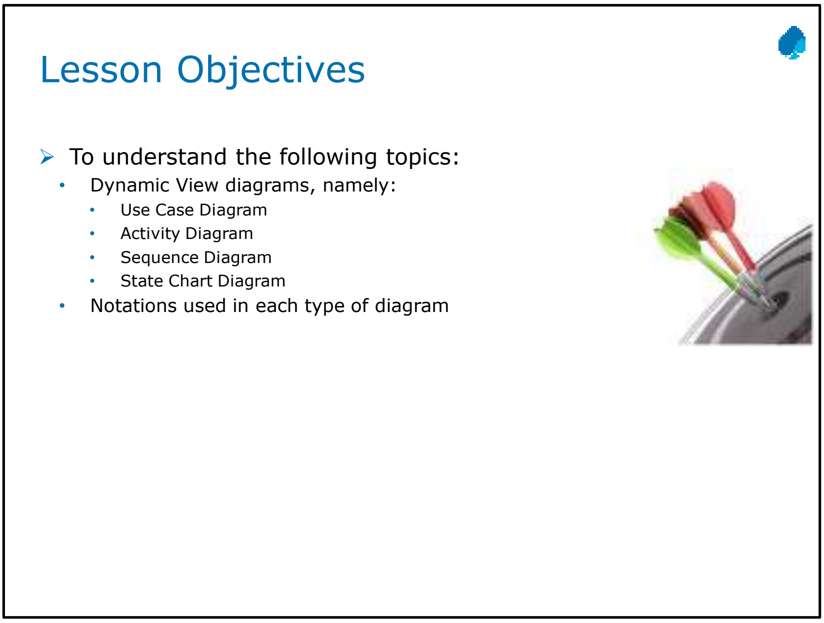


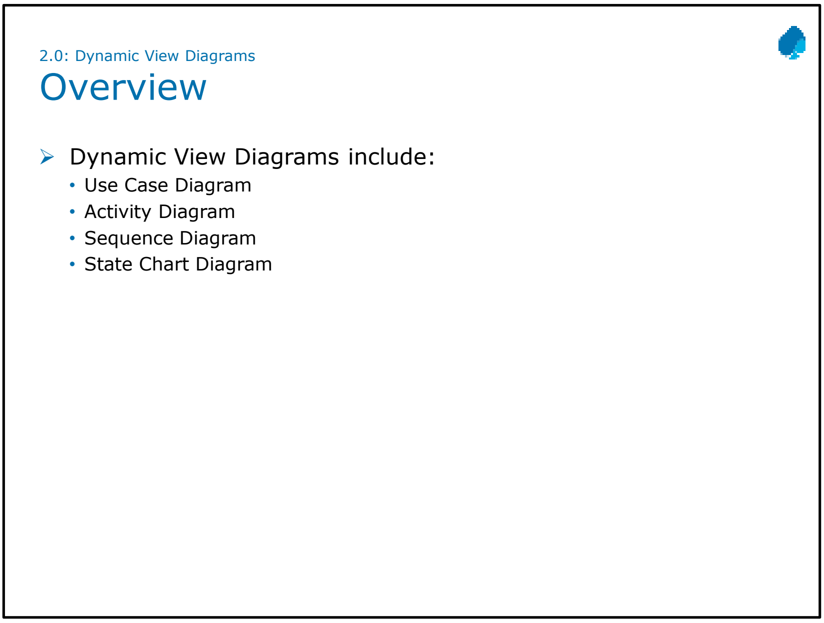
Unified Modeling Language

Lesson 2: Dynamic View Diagrams



Lesson Objectives

- To understand the following topics:
  - Dynamic View diagrams, namely:
    - Use Case Diagram
    - Activity Diagram
    - Sequence Diagram
    - State Chart Diagram
  - Notations used in each type of diagram



2.0: Dynamic View Diagrams

Overview

- Dynamic View Diagrams include:
  - Use Case Diagram
  - Activity Diagram
  - Sequence Diagram
  - State Chart Diagram

Dynamic View Diagrams:

As seen in the earlier lesson, UML diagrams are classified as:

Dynamic View Diagrams

Static View Diagrams

Physical View Diagrams

Let us begin with Dynamic View Diagrams.

Dynamic View Diagrams include:

Use Case Diagram

Activity Diagram

Sequence Diagram

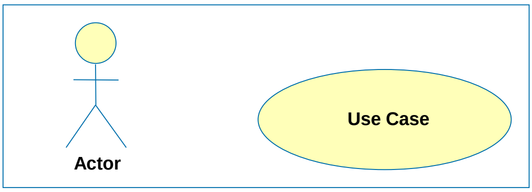
State Chart Diagram

In this lesson we will discuss Use Case Diagrams, Activity Diagrams, Sequence Diagrams and State Chart Diagrams.

2.1: Use Case Diagrams

Use Case Diagrams - Features

- Use Case Diagrams model the functionality of a system by
- using Actors and Use Cases:
  - Actor is a user of the system.
  - Use cases are services or functions provided by the system to its users.



Use Case Diagrams:

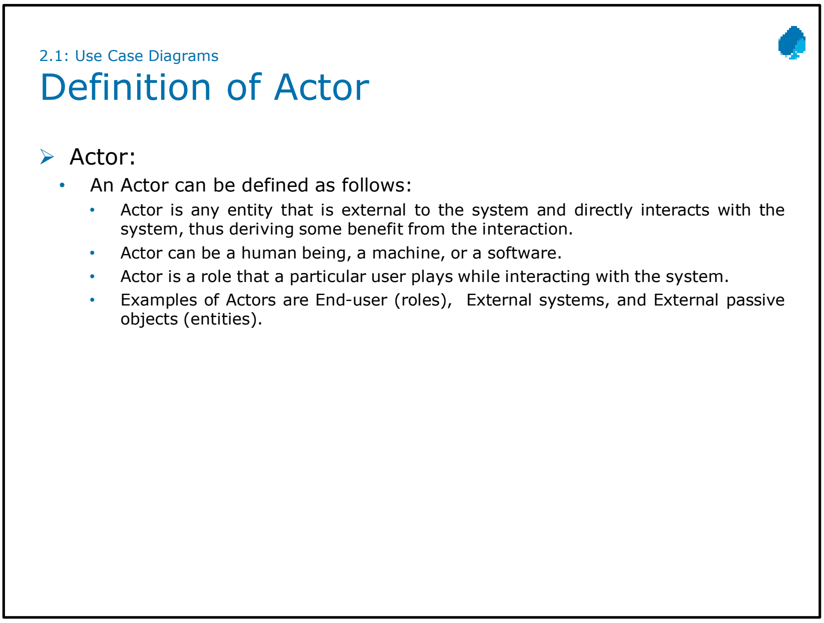
Use Case is a description of a system's behavior from a user's point of view. It is a set of scenarios that describe an interaction between a user and a system. It also displays the relationship among Actors and Use Cases.

Two main components of Use Case diagram are Use Cases and Actors.

Use case diagrams, which render the User View of the system, describe the functionality (Use Cases) provided by the system to its users (Actors).

An Actor represents a user or another system that will interact with the system you are modeling.

An Use Case is an external view of a system that represents some action that the user might perform in order to complete a task.



2.1: Use Case Diagrams

Definition of Actor

- Actor:
  - An Actor can be defined as follows:
    - Actor is any entity that is external to the system and directly interacts with the system, thus deriving some benefit from the interaction.
    - Actor can be a human being, a machine, or a software.
    - Actor is a role that a particular user plays while interacting with the system.
    - Examples of Actors are End-user (roles), External systems, and External passive objects (entities).

Actor:

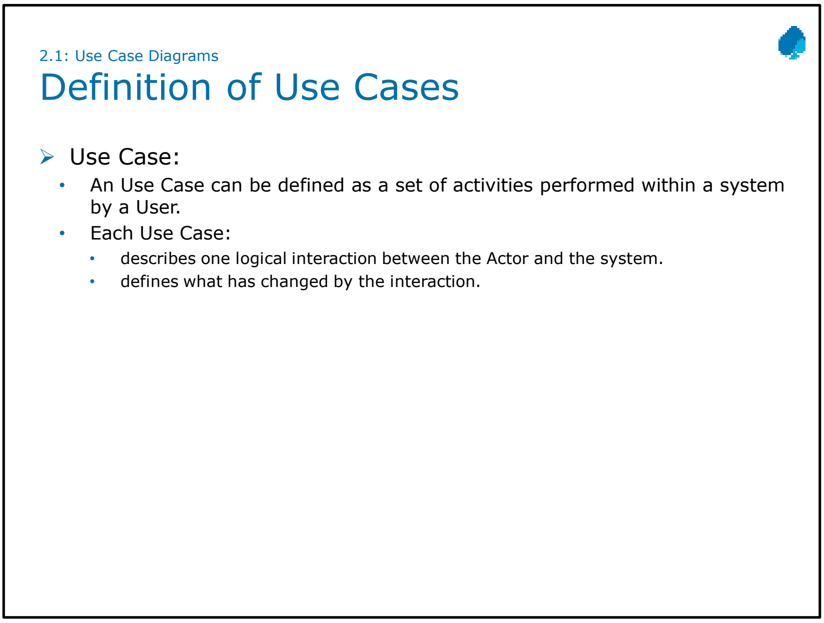
Actors are people, organizations, systems, or devices which use or interact with our system. The system exists to support that interaction. Therefore, the important part of the project is to identify the Actors and find out what they want from the system.

Actors are characterized by their external view rather than their internal structures. It is a role that the user plays to get something from the system.

Role and organization Actors only require logical interactions with the system. Ask who wants what from our system, rather than who operates the system.

For example: ABC and XYZ are users who wish to buy from an online store. For the online stores system, they play the role of a customer, and hence customer is the Actor for the system. The database for this system may already be existing, and hence this may be another Actor (note that user in this case is not a human).

The Actors will finally be used to describe classes, which will interact with other classes of the system.



2.1: Use Case Diagrams

Definition of Use Cases

- Use Case:
  - An Use Case can be defined as a set of activities performed within a system by a User.
  - Each Use Case:
    - describes one logical interaction between the Actor and the system.
    - defines what has changed by the interaction.

Use Cases:

The Use Cases define “units of functionality” provided by system. They model “work units” that the system provides to its outside world.

A Use Case is one usage of the system. It is a generic description of a use of the system. It allows interactions in a specific sequence.

At the lowest level, they are nothing but methods which need to be implemented by various classes in the system.

Use Cases determines everything that the Actor wants to do with the system.

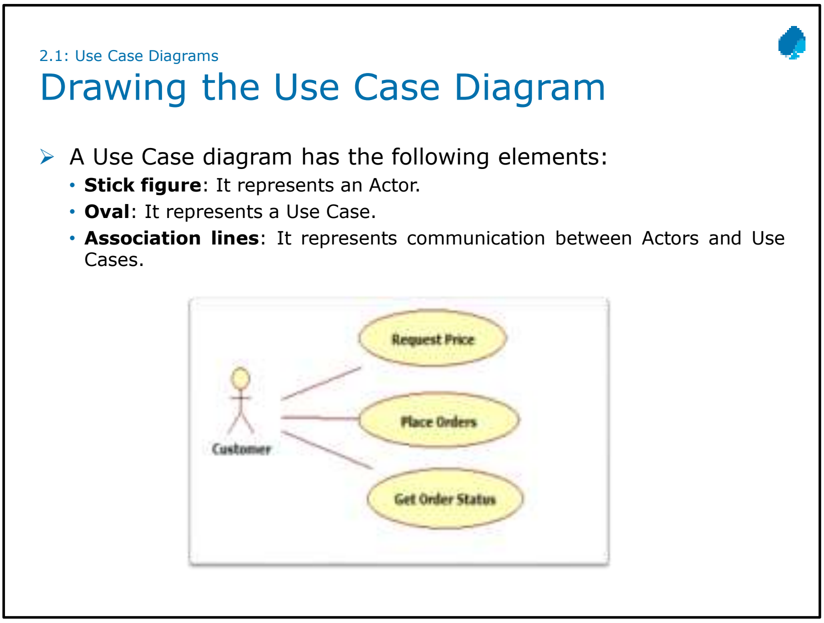
A Use Case performs the following functions:

Defines main tasks of the system

Reads, writes, and changes system information

Informs the system of real world changes

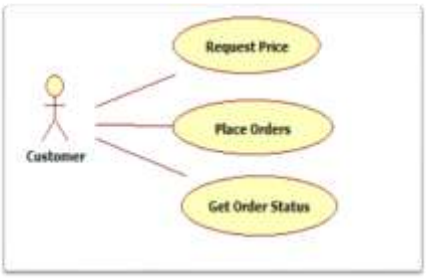
A Use Case needs to be updated / informed about system changes.



2.1: Use Case Diagrams

Drawing the Use Case Diagram

- A Use Case diagram has the following elements:
  - **Stick figure**: It represents an Actor.
  - **Oval**: It represents a Use Case.
  - **Association lines**: It represents communication between Actors and Use Cases.



Drawing the Use Case Diagram:

The Use Case Diagram has the following elements:

A stick figure, which represents Actors (sometimes stereotyped classes, as explained later, are also used to represent Actors). They differ from tool to tool.

Ovals or ellipses, which represent Use Cases

Association lines, which indicate interactions between Actors and Use Cases.

Use Cases will have description of what the Use Case is supposed to do when it is used.

An example of use case description is given.

2.1: Use Case Diagrams

Use Case Specification

- ➢ Each Use Case would have a Use Case Specification associate with it
- ➢ No standard template used by UML for this, but typical formats would include
  - Name and Brief Description
  - Invoking Actor
  - Flow of Events including Basic and Alternate Flows
  - Special Requirements
  - Pre-Conditions, Post Conditions & Extension Points

See the notes pages for an example of Use Case Specification (or Description Document)

Example:

Use Case Specification: To make a Reservation

This use case describes how a reservation is made in the Airlines Reservation System.

Invoking Actor:

Customer (passenger)

Flow of Events:

Basic Flow:

The use case begins when a customer wishes to make a reservation.

The system displays a reservation form.

The customer enters the reservation details. Reservation details include:

Round Trip or One Way, Origin, Destination, Departure and Arrival Dates, Number of Passengers (Adults and Children).

The system retrieves Flight Details based on reservation request.

The customer selects the desired flight.

The system calculates and displays the total fares applicable inclusive of taxes.

The system requests passenger details (First and Last Name) and Contact Information (Phone No., Email Id and Mobile No.)

Example (contd.):

- **Flow of Events (contd.)**

7. The customer specifies the requested details.
8. The system requests for Credit Card Details: Credit Card Type, Credit Card Holder Name, Credit Card Number, Expiry Date, CVV Number and Payment Amount.
9. The customer specifies the details.
10. The system requests the Credit Card Agency to validate the credit card details and charge payment.
11. On successful payment, system generates the ticket.
12. The customer can print the ticket.

➢ Alternative Flows:

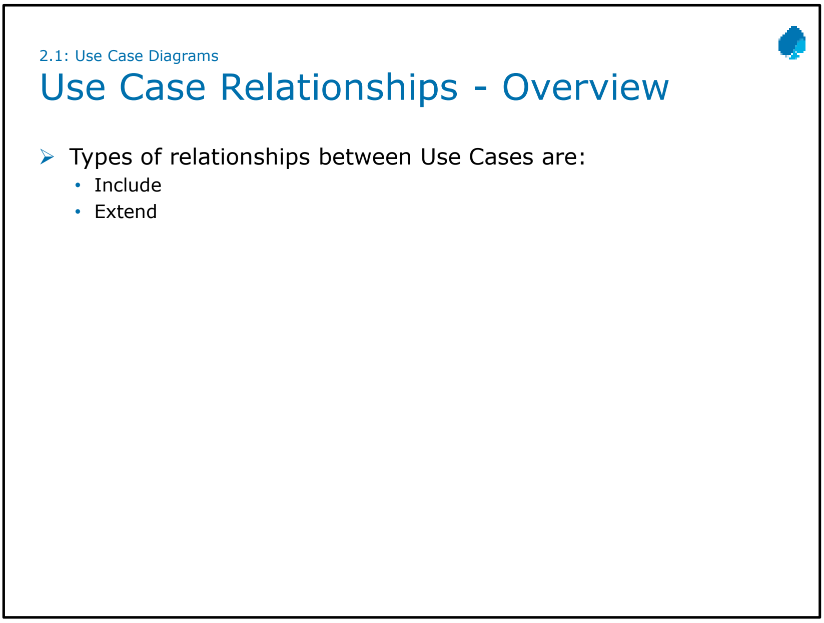
- Flight not available:
  If no flight is available for specified reservation details, appropriate message is displayed. The home page is then displayed.
- Credit Card Authorization not available:
  If connection to the credit card system is not available, appropriate message is displayed. The customer can retry payment.
- Credit Card Authorization Fails:
  If the credit card authorization fails, appropriate message is displayed and use case is terminated.

- **Special Requirements**

➢ All customer entries to be validated on forms.
➢ Total fare includes taxes. Present Taxes per ticket are: Fuel Surcharge of Rs.1000/-, Airport Taxes of Rs.750/-.

- **Pre Conditions**
  The customer is on the home page of the system. The reservation form is on the home page.
- **Post Conditions**
  The customer successfully reserves the tickets.

- **Extension Points**
  None.



2.1: Use Case Diagrams

Use Case Relationships - Overview

➢ Types of relationships between Use Cases are:
- Include
- Extend

Use Case Relationships:

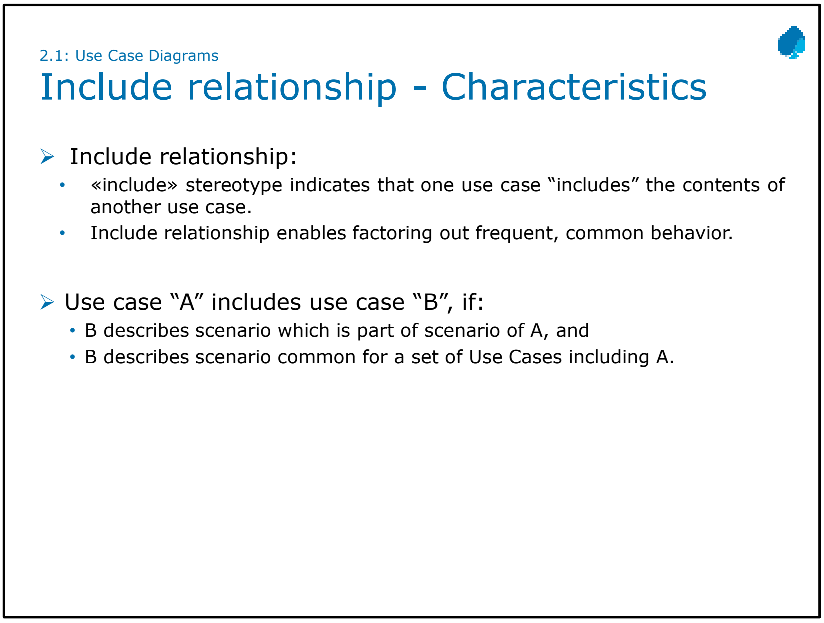
Relationships help us connect the model elements.

After finding out the primary Use Cases, one can start looking “into” the system to see if there are any relationships between the Use Cases.

The following types of relationships can exist between the Use Cases:

include

extend



2.1: Use Case Diagrams

Include relationship - Characteristics

- Include relationship:
  - «include» stereotype indicates that one use case "includes" the contents of another use case.
  - Include relationship enables factoring out frequent, common behavior.

- Use case "A" includes use case "B", if:
  - B describes scenario which is part of scenario of A, and
  - B describes scenario common for a set of Use Cases including A.

Use Case Relationship – Include:

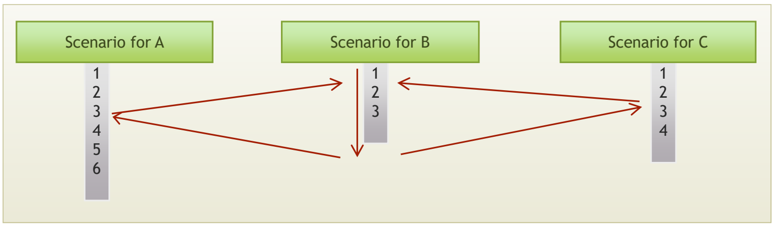
In an Include relationship, one Use Case includes behavior specified by another Use Case. If there are common steps in the scenarios of many Use Cases, they can be factored out into a separate Use Case. This Use Case can then be included as part of the "Primary Use Case".

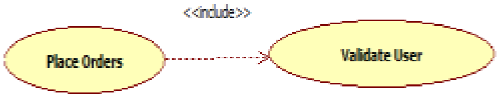
The above arrangement helps us segregate and organize common sub-tasks. An "Included Use case" is not a complete process. Extra behavior is added to the base Use Case.

This may also be used in case of complex Use Cases (where there is too much functionality in one Use Case). In such cases, the primary functionality can be distributed across Use Cases, and a primary Use Case can then include the remaining secondary Use Cases.

2.1: Use Case Diagrams

Include relationship - Example

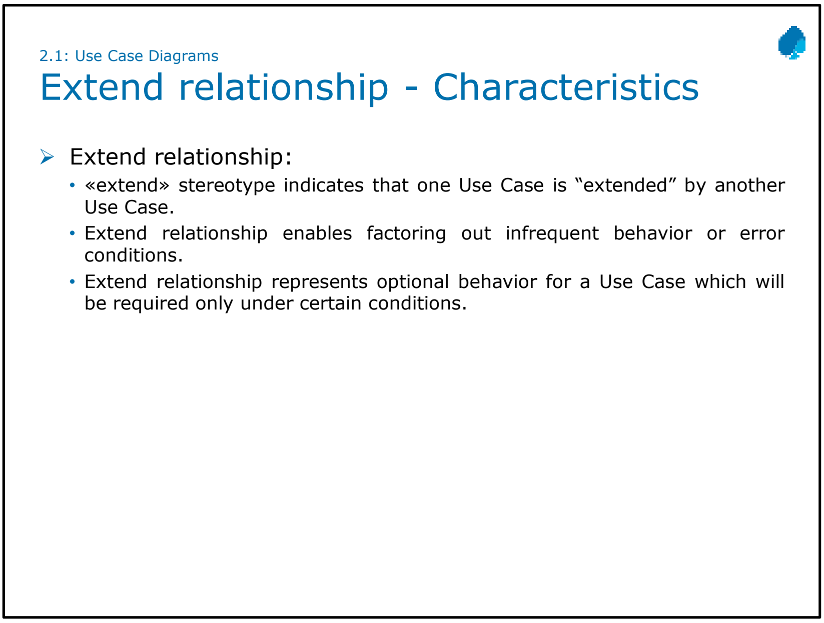




Use Case Relationships – Include (contd.):

As illustrated in the figure shown in the slide, scenario of Use Case B is required by Use Case A and Use Case C. Hence both Use Cases A and C can include Use Case B.

After completing the scenario of Included Use Case B, the Use Cases A and C will continue with their respective scenarios.



2.1: Use Case Diagrams

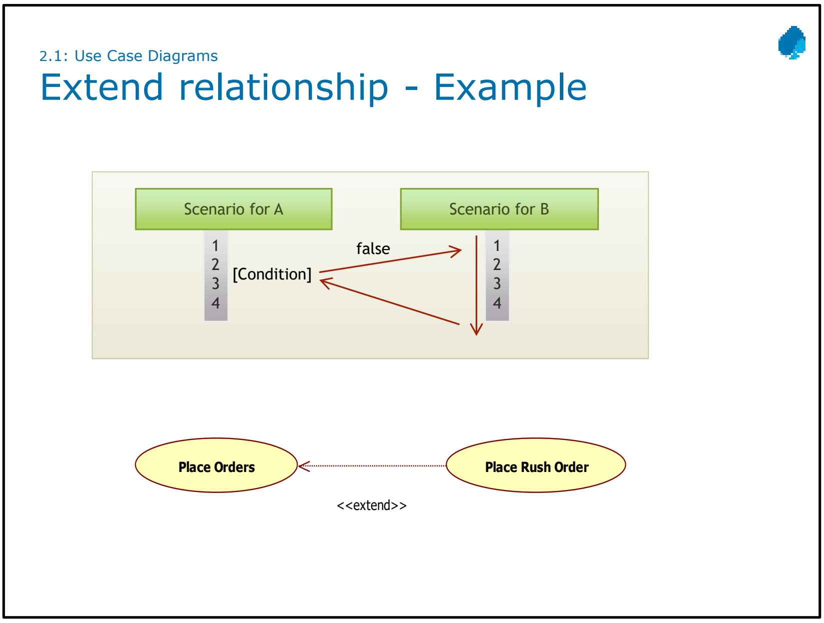
Extend relationship - Characteristics

- Extend relationship:
  - «extend» stereotype indicates that one Use Case is "extended" by another Use Case.
  - Extend relationship enables factoring out infrequent behavior or error conditions.
  - Extend relationship represents optional behavior for a Use Case which will be required only under certain conditions.

Use Case Relationships - Extend :

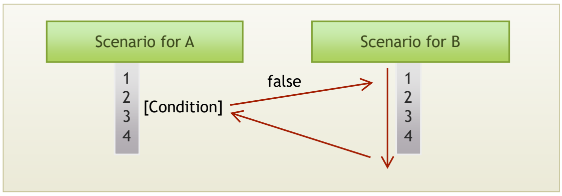
In an Extend relationship, an Use Case may be required by another use case based on "some condition", or due to "some exceptional situation".

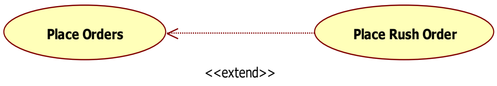
Again, upon completion of extension activity sequence, the original Use Case will continue.



2.1: Use Case Diagrams

Extend relationship - Example





Use Case Relationships – Exclude (contd.):

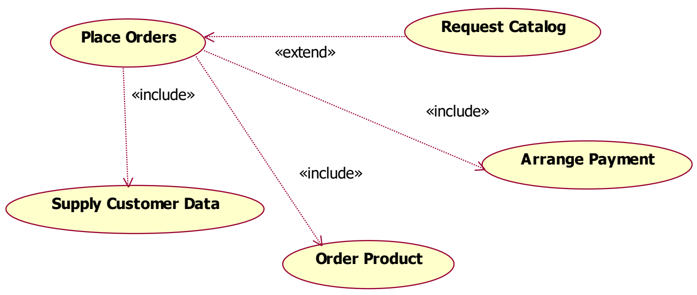
As illustrated in the figure shown in the slide, in Use Case A when the condition becomes false, the scenario of Use Case B is invoked.

Note that Use Case B is said to extend Use Case A. The stereotyped generalization arrow with keyword “extend” depicts the extend relationship between the Use Cases.

2.1: Use Case Diagrams

Examples of Use Case Relationships

- Example 1:



Example of Use Case Diagram:

The slide shows an example where we are looking at the Use Case relationships (though the Actors and system boundary has not been shown here, let us assume that they exist. They have been left out so as to focus on the relationships).

The Request for Catalog may not always happen when an Order needs to be placed. Hence, Extend is the relationship used between the two Use Cases of “Place Order” and “Request Catalog”.

“Place Order” being a complex Use Case, it is broken down into secondary use cases of:

Supply Customer data

Order Product

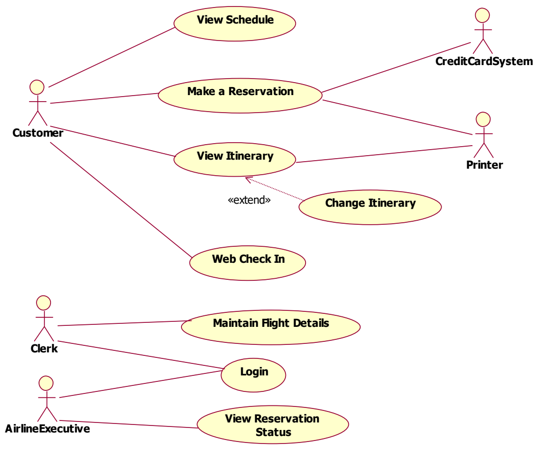
Arrange Payment

It is important to note that the diagram by itself does not indicate any kind of ordering (i.e. first invoke order product, and then arrange payment, etc). The ordering has to be taken care of as part of the Use Case scenario.

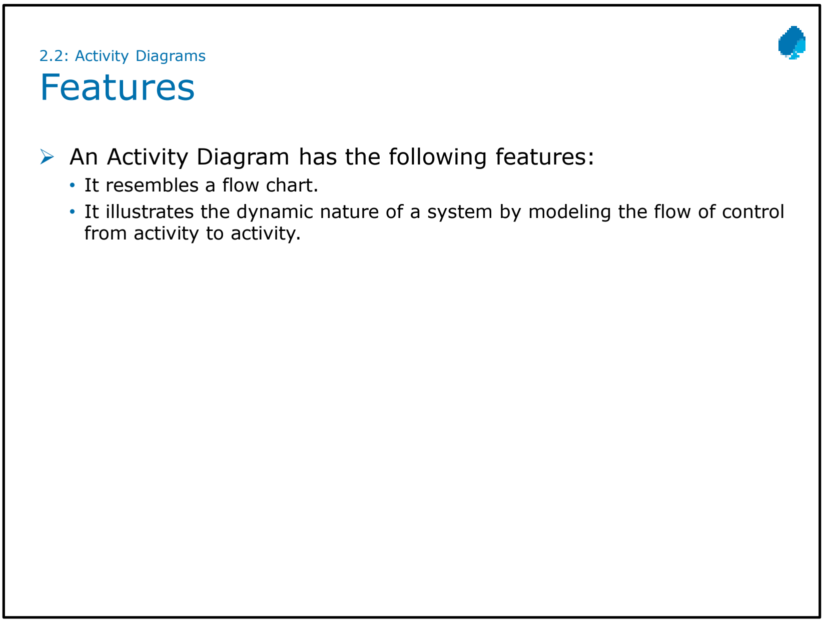
2.1: Use Case Diagrams

Examples of Use Case Relationships (Contd…)

Example 2:



How do you interpret this diagram?



2.2: Activity Diagrams

Features

- An Activity Diagram has the following features:
  - It resembles a flow chart.
  - It illustrates the dynamic nature of a system by modeling the flow of control from activity to activity.

Activity Diagrams:

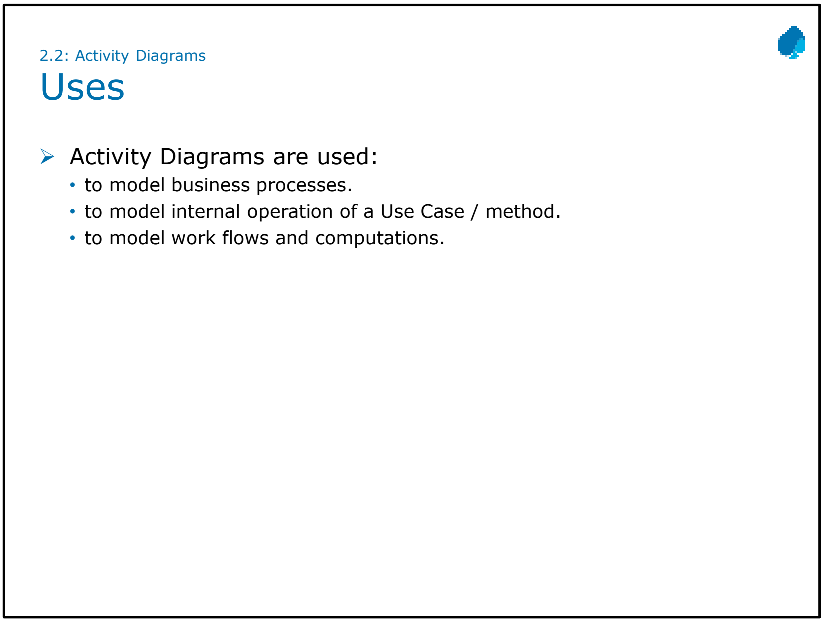
Activity Diagrams are typically used:

for business process modeling,

for modeling the logic captured by a single use case or usage scenario, or

for modeling the detailed logic of a business rule

Activity Diagrams look similar to flow charts.



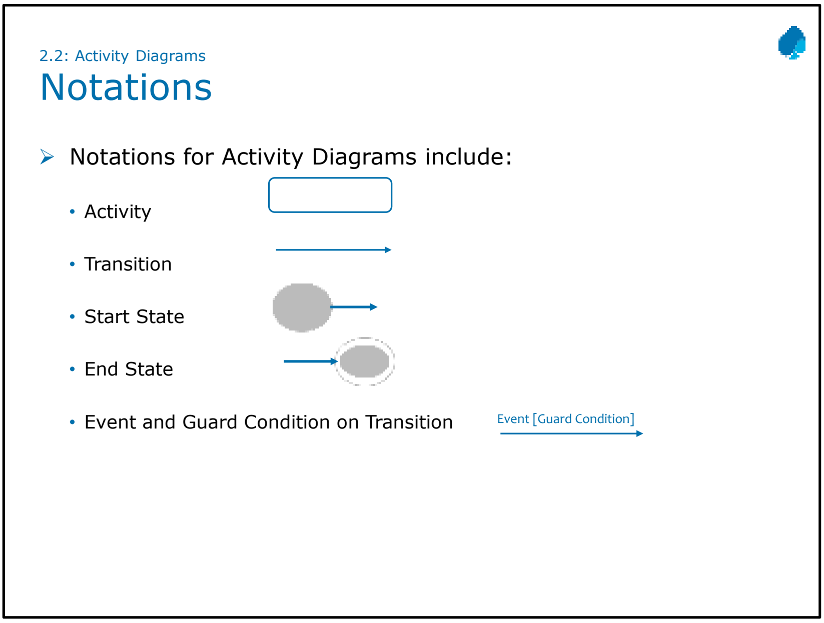
2.2: Activity Diagrams

Uses

- Activity Diagrams are used:
  - to model business processes.
  - to model internal operation of a Use Case / method.
  - to model work flows and computations.

Uses of Activity Diagrams:

The Activity Diagrams describe flow of activities. Hence, they can be used to better understand the flow of activities of the business model, a requirement (Use Case), or even a complicated work flow or computation.



2.2: Activity Diagrams

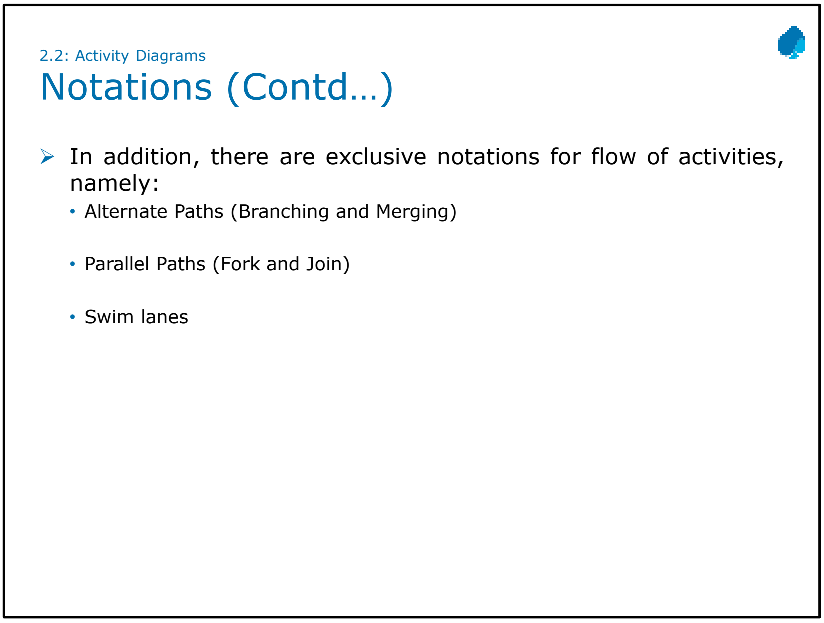
Notations

- Notations for Activity Diagrams include:
  - Activity
  - Transition
  - Start State
  - End State
  - Event and Guard Condition on Transition

Event [Guard Condition]

Activity Diagrams – Notations:

Note that an Activity Diagram is a special case of State Chart Diagram, and many notations are same. The common notations are listed here. State Chart Diagrams are discussed later on in the flow of the topic.



2.2: Activity Diagrams

Notations (Contd…)

- In addition, there are exclusive notations for flow of activities, namely:
  - Alternate Paths (Branching and Merging)
  - Parallel Paths (Fork and Join)
  - Swim lanes

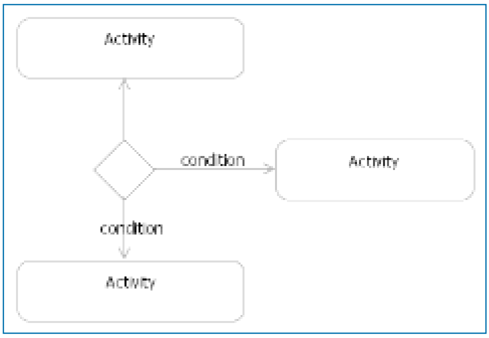
Activity Diagrams – Notations (contd.):

Apart from the notations as required for State Diagrams, the other notations include those required to model the flow of activities like alternate and parallel paths, and swim lanes, as well.

2.2: Activity Diagrams

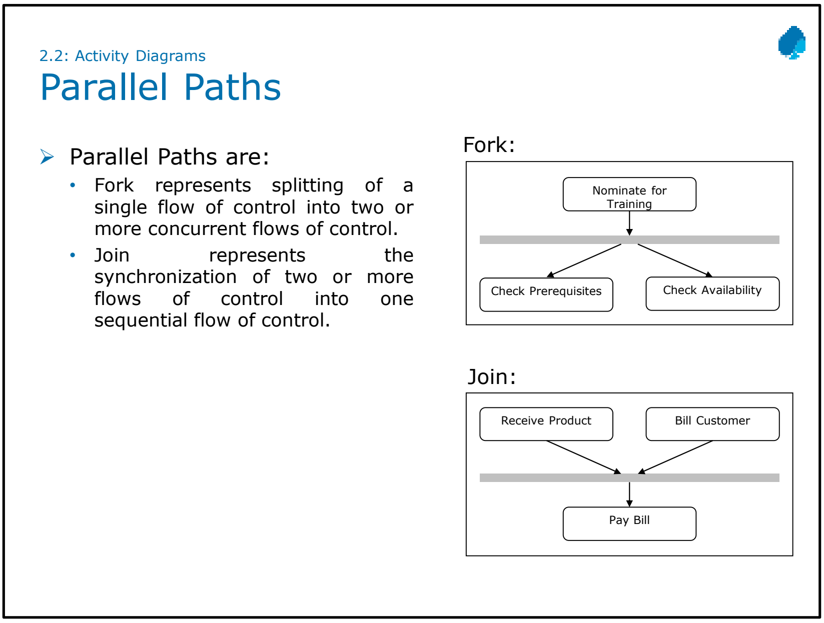
Alternate Paths

- Alternate Paths:
  - Alternate Paths are shown with the following notations:
    - Diamond representing a decision with alternate paths
    - Guard conditions used to label the alternates



Activity Diagram – Notations for Activity Flow: Alternate Paths:

As the name suggests, alternate paths indicate the different flows based on evaluation of “guard conditions”. Like in a flow chart, the diamond specifies the point where alternate paths are available.

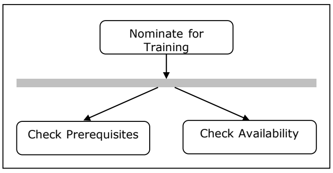


2.2: Activity Diagrams

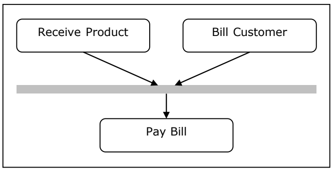
Parallel Paths

- Parallel Paths are:
  - Fork represents splitting of a single flow of control into two or more concurrent flows of control.
  - Join represents the synchronization of two or more flows of control into one sequential flow of control.

Fork:



Join:



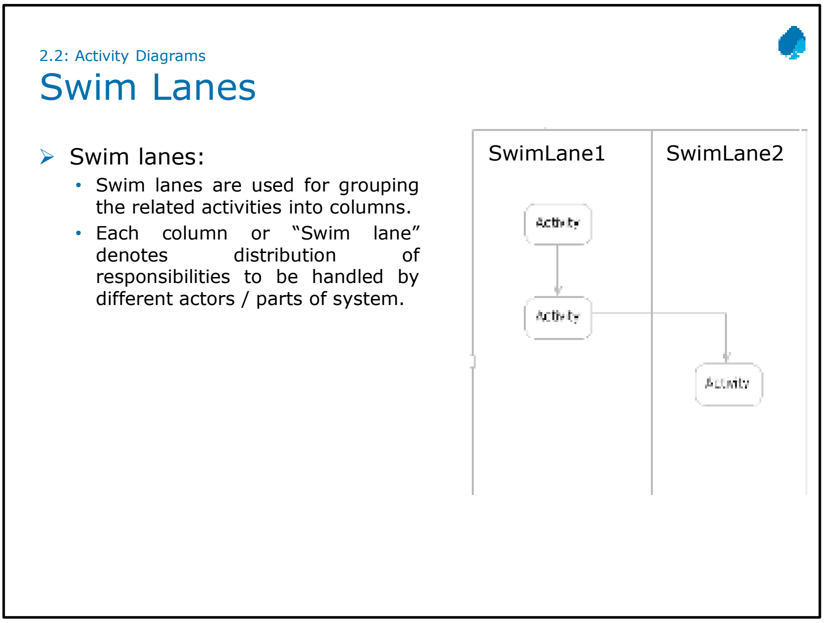
Activity Diagrams – Notations for Activity Flow: Parallel Paths:

The fork and join respectively represent splitting and synchronization of activities. Both are required for modeling parallel flow of activities.

For example:

“Check Prerequisites” and “Check Availability” can happen in parallel after the activity of “Nominate for Training”. This is depicted in the example for Fork.

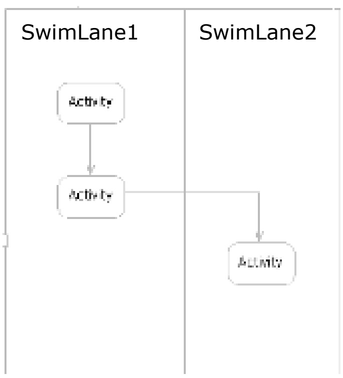
The “Pay Bill” activity can occur only after the completion of activities of “Receive Product” and “Bill Customer”. This is depicted in the example for Join.



2.2: Activity Diagrams

Swim Lanes

- Swim lanes:
  - Swim lanes are used for grouping the related activities into columns.
  - Each column or "Swim lane" denotes distribution of responsibilities to be handled by different actors / parts of system.

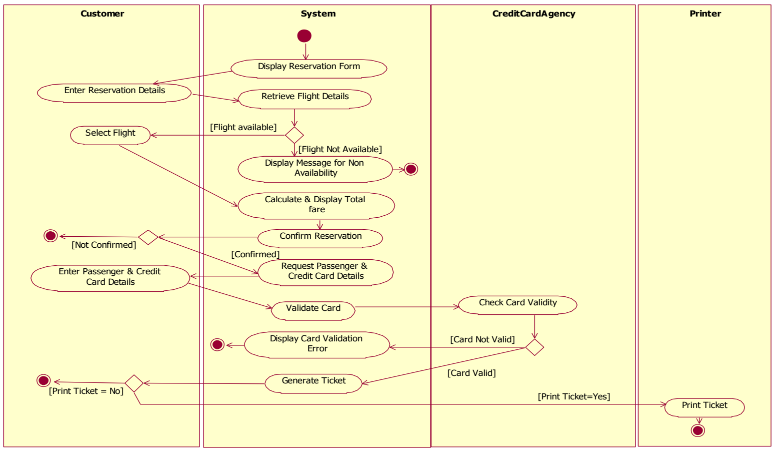


Activity Diagrams – Notations for Activity Flow : Swim lanes:

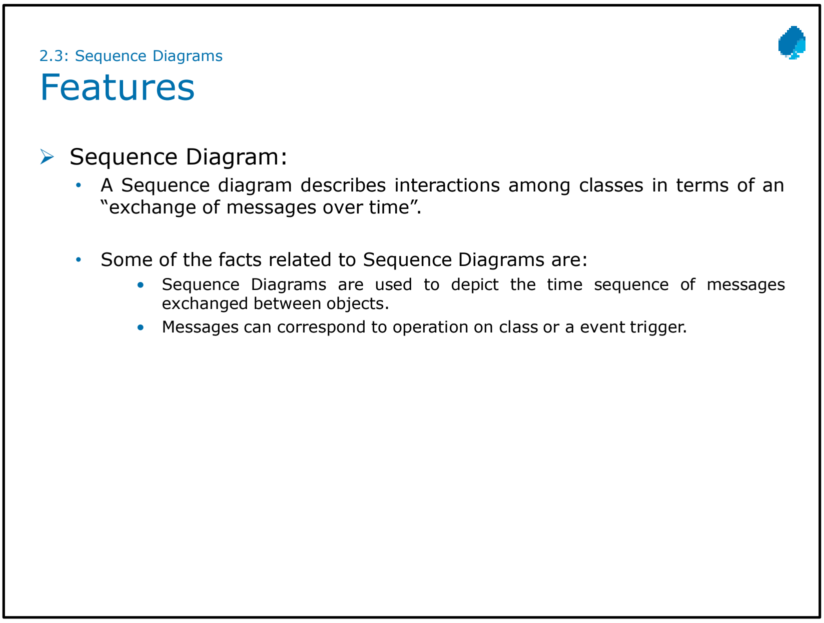
Swim lanes are used to partition set of actions, thereby dividing activities into groups. Responsibility for each group of actions can be assigned to actors or objects within the system.

2.2: Activity Diagrams

Example of Activity Diagram



The slide shows an example of an Activity Diagram. How do you interpret this?



2.3: Sequence Diagrams

Features

- Sequence Diagram:
  - A Sequence diagram describes interactions among classes in terms of an “exchange of messages over time”.
  - Some of the facts related to Sequence Diagrams are:
    - Sequence Diagrams are used to depict the time sequence of messages exchanged between objects.
    - Messages can correspond to operation on class or a event trigger.

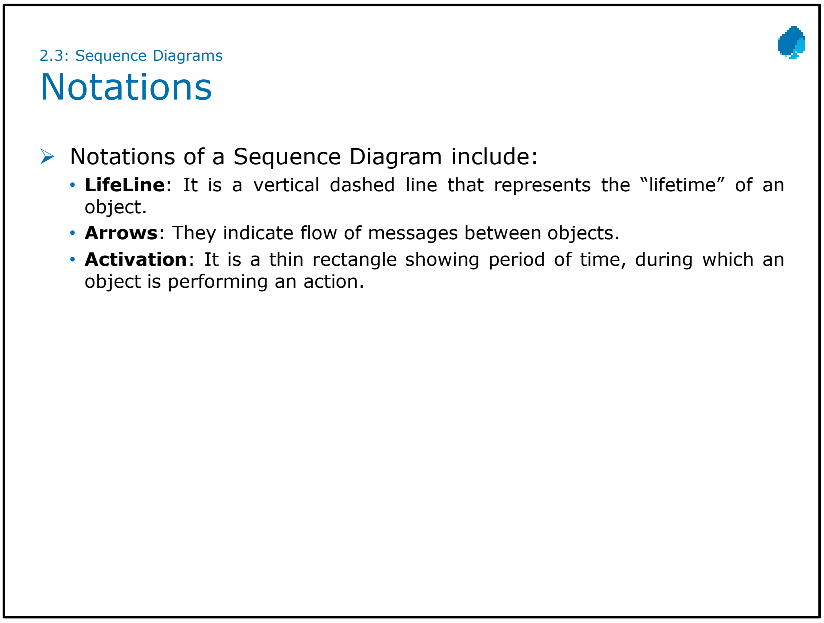
Sequence Diagrams: Features:

Starting from the scenarios of the Use Case, interactions between objects in the scenario can be depicted in the Sequence Diagram. These interactions are in the “sequence of time”, and finally correspond to “operations” or “event triggers”.

Only the sequences of messages, with respect to time, are important. There is no particular span of time that is considered.

Sequence Diagrams show objects, and not classes. This is different from Class Diagrams, which contains classes that signify the existence of objects.

Sequence Diagrams provide a better correlation between the “dynamic view” of the system and the “static view” held in the Class Diagram.



2.3: Sequence Diagrams

Notations

- Notations of a Sequence Diagram include:
  - **LifeLine**: It is a vertical dashed line that represents the "lifetime" of an object.
  - **Arrows**: They indicate flow of messages between objects.
  - **Activation**: It is a thin rectangle showing period of time, during which an object is performing an action.

Sequence Diagrams: Notations:

In the Sequence Diagram, different objects / class roles are laid out horizontally at the top. There is no significance to the ordering of these objects.

The Lifeline, denoted as dashed line beneath the class role, is used to model "existence of entities over time".

Activation, shown as thin rectangles along the lifeline, model the time during which entities are active. Activation may not always be depicted.

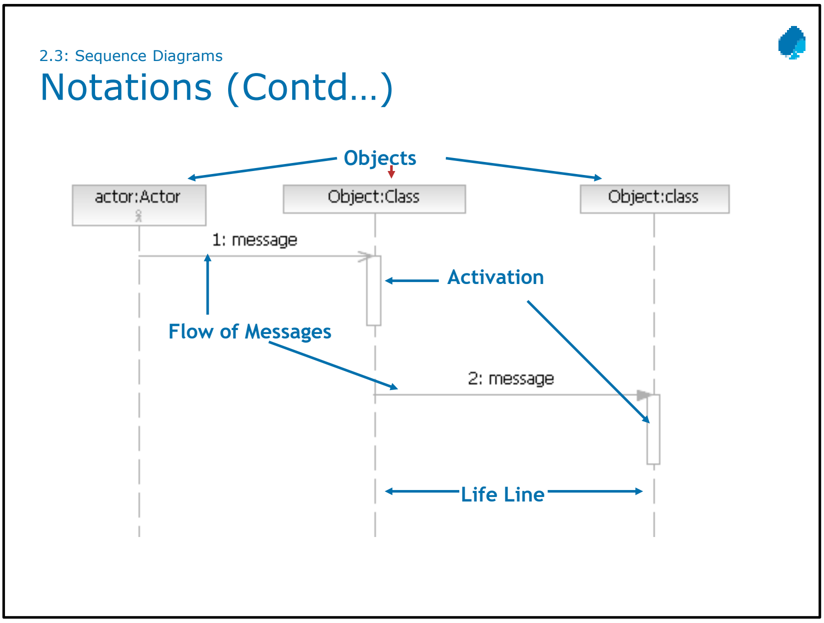
Arrows indicate flow of messages between objects.

Messages are denoted as horizontal arrows between lifelines.

Messages may be:

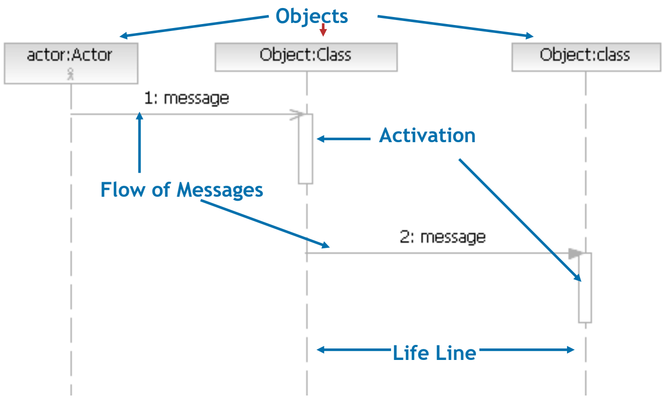
as simple as a "string" conveying an operation, or

as detailed as a "method", which includes parameters passed and values returned.



2.3: Sequence Diagrams

Notations (Contd…)

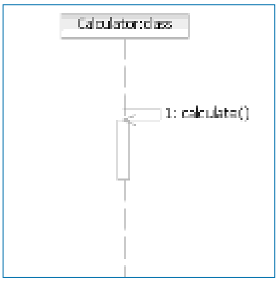


The slide shows the notations used in a Sequence Diagram.

2.3: Sequence Diagrams

Direction of Arrows

- Direction of Arrows:
  - Direction indicates which object's method is being called by whom.
  - A circulating arrow on the Object Lifeline is for a self method - called within the object by itself.



Sequence Diagram: Notations (Directions and Branches):
Direction of arrows:

Every message has a "sender" and "receiver" and this is depicted by the direction of the arrow head. Control gets passed from the sender to receiver.

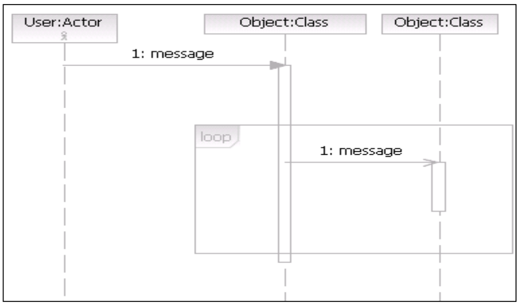
"Implicit returns" are assumed in case there are no return messages to the sender.

It is possible that there is a "call to object's own method". A "circulating arrow" is used to depict such a case. In the example shown on the slide, Calculator object calls its own method calculate().

2.3: Sequence Diagrams

Branch Conditions

- Branch Conditions:
  - Branch Conditions are depicted as “Guard Conditions” within Square Brackets.
  - Repetition or Looping depicted as a rectangle, with condition for exiting the loop placed at the bottom left corner.



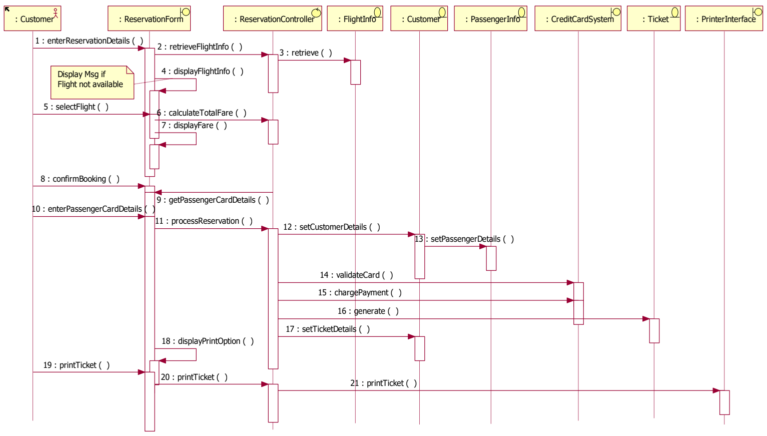
Sequence Diagram: Notations (Directions and Branches):
Branch Conditions:

Branching involves multiple messages originating at same time to different objects, based on some condition called as the “Guard Conditions”. These Guard Conditions are given in square brackets.

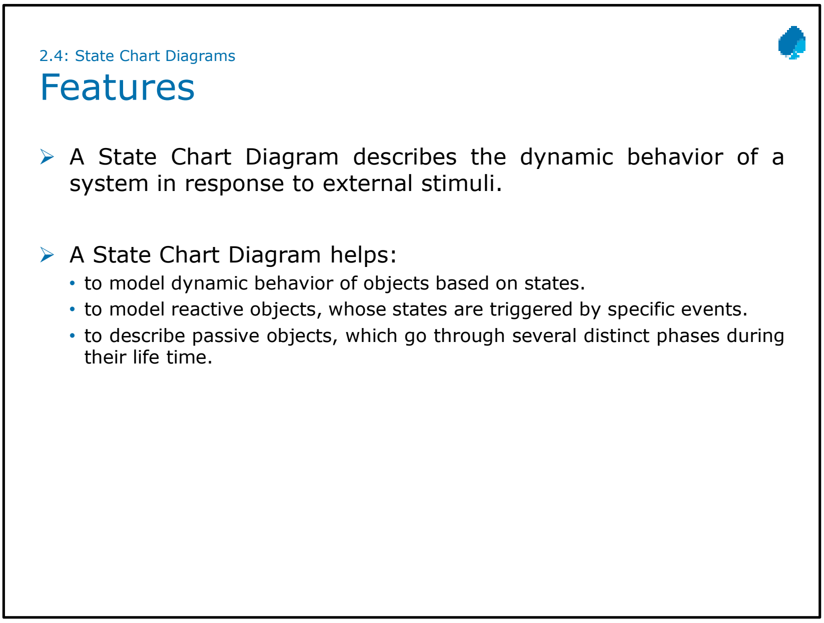
Iteration involves sets of messages sent multiple times. A rectangle enclosing the set of messages indicates looping.

2.3: Sequence Diagrams

Example of Sequence Diagrams


: Customer
: ReservationForm
: ReservationController
: FlightInfo
: Customer
: PassengerInfo
: CreditCardSystem
: Ticket
: PrinterInterface
1 : enterReservationDetails ( )
2 : retrieveFlightInfo ( )
3 : retrieve ( )
4 : displayFlightInfo ( )
Display Msg if Flight not available
5 : selectFlight ( )
6 : calculateTotalFare ( )
7 : displayFare ( )
8 : confirmBooking ( )
9 : getPassengerCardDetails ( )
10 : enterPassengerCardDetails ( )
11 : processReservation ( )
12 : setCustomerDetails ( )
13 : setPassengerDetails ( )
14 : validateCard ( )
15 : chargePayment ( )
16 : generate ( )
17 : setTicketDetails ( )
18 : displayPrintOption ( )
19 : printTicket ( )
20 : printTicket ( )
21 : printTicket ( )

How do you interpret this diagram?



2.4: State Chart Diagrams

Features

- A State Chart Diagram describes the dynamic behavior of a system in response to external stimuli.
- A State Chart Diagram helps:
  - to model dynamic behavior of objects based on states.
  - to model reactive objects, whose states are triggered by specific events.
  - to describe passive objects, which go through several distinct phases during their life time.

State Chart Diagrams:

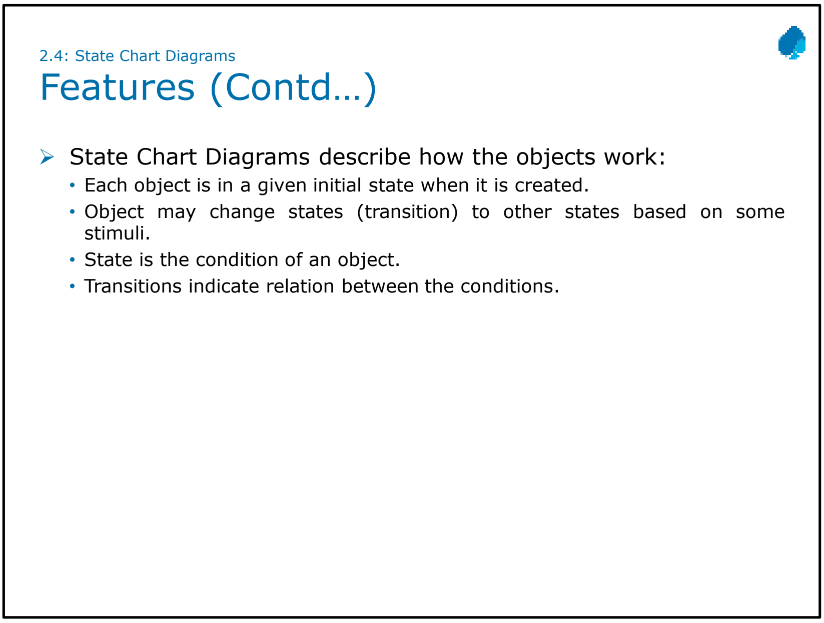
Understanding the state of an object is important as the state of the object determines the behavior of the object. States of an object can be modeled by using the State Chart Diagram.

State Chart Diagrams, also known as State Diagrams or State Machines, are useful for describing the lifecycle of an object.

> The lifecycle is the set of all the states that an object can be in, and the events based on which an object will transition from one state to another.

The State Chart Diagrams are “graphs” of states and their transitions.

contd.



2.4: State Chart Diagrams

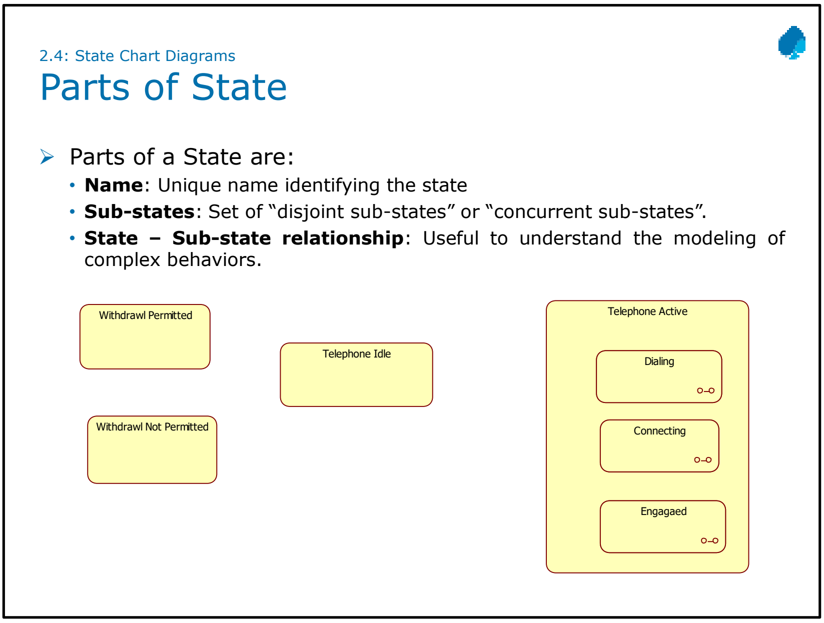
Features (Contd…)

- State Chart Diagrams describe how the objects work:
  - Each object is in a given initial state when it is created.
  - Object may change states (transition) to other states based on some stimuli.
  - State is the condition of an object.
  - Transitions indicate relation between the conditions.

State Chart Diagrams (contd.):

As discussed earlier, a state is the set of current values of attributes. Each object is in a given initial state when it is created. The object may change states, i.e. transition to other states based on some stimuli.

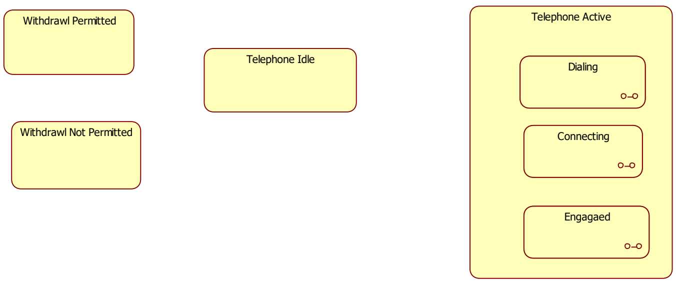
So the “states” represent the “conditions” of an object. Further, the “transitions” specify how these conditions are related.



2.4: State Chart Diagrams

Parts of State

- Parts of a State are:
  - **Name**: Unique name identifying the state
  - **Sub-states**: Set of "disjoint sub-states" or "concurrent sub-states".
  - **State – Sub-state relationship**: Useful to understand the modeling of complex behaviors.



State Chart Diagrams – Notations for States:

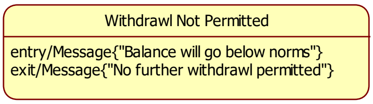
Every state has a unique name which identifies the state. For a state which may be complex, it may be modeled with nested sub-states. These sub-states could be "disjoint" or "concurrent".

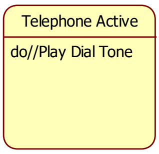
Other parts of the state include the internal transitions to specify actions that happen as a result of coming into the state.

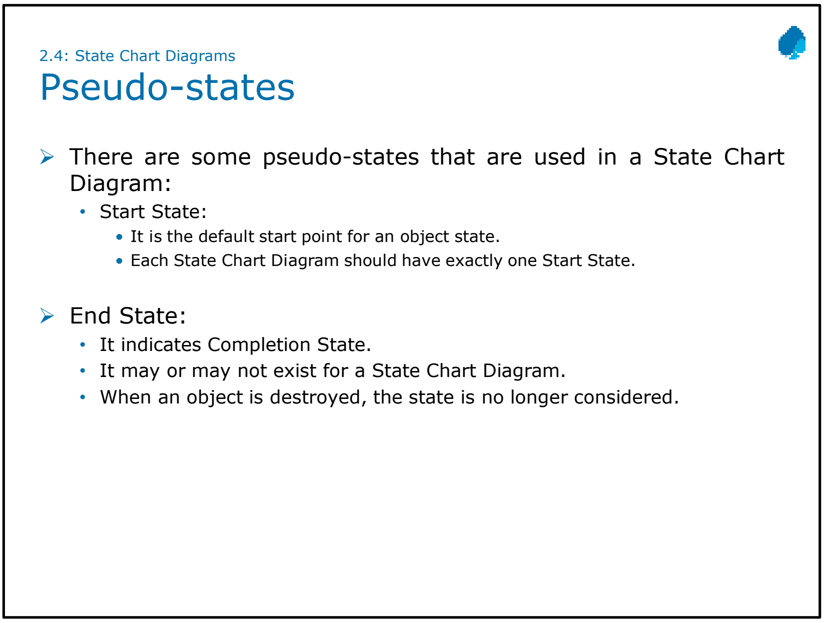
2.4: State Chart Diagrams

Parts of State

- Entry action: An action that happens as a result of transition into a state.
- Exit action: An action that has to happen immediately before a state change.
- Internal activity: Activities that occur within an object while it is in a particular state.


Withdrawl Not Permitted
entry/Message{"Balance will go below norms"}
exit/Message{"No further withdrawl permitted"}





2.4: State Chart Diagrams

Pseudo-states

- There are some pseudo-states that are used in a State Chart Diagram:
  - Start State:
    - It is the default start point for an object state.
    - Each State Chart Diagram should have exactly one Start State.

- End State:
  - It indicates Completion State.
  - It may or may not exist for a State Chart Diagram.
  - When an object is destroyed, the state is no longer considered.

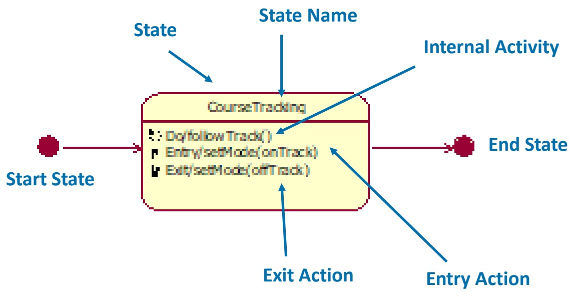
State Chart Diagram: Notations for States (contd.):

In a State Chart Diagram, there has to be a Start State for an object. A filled circle as shown above points to the Start State. There are no incoming transitions to the filled circle, and there is exactly one outgoing transition from the same.

End State represents the completion of an activity in the enclosing state. It is depicted by using a circle enclosing filled circle as shown in the diagram in the slide. It is not mandatory to have an End State for each object.

2.4: State Chart Diagrams

Example



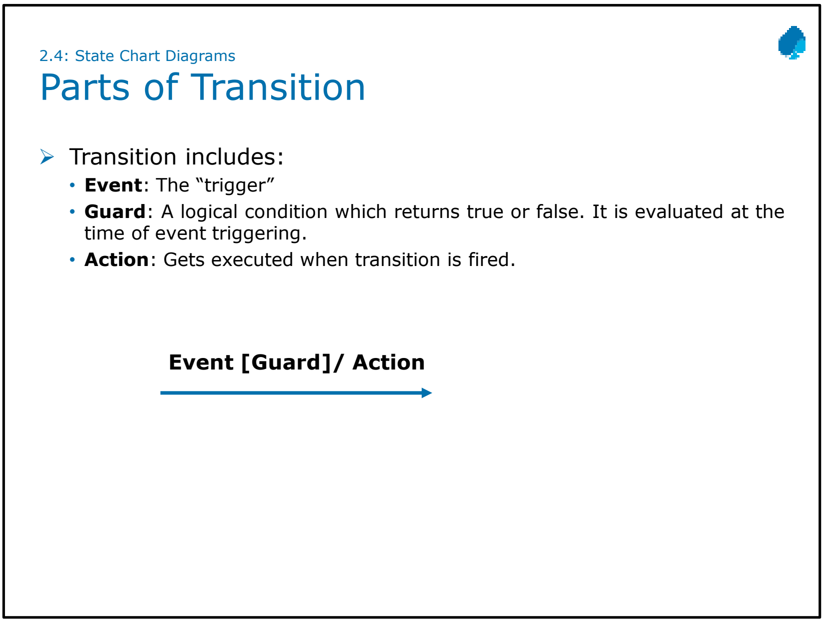
State Chart Diagram: Notations for States (contd.):

The slide shows an example where the state Tracking has:

- an entry action called setMode with parameter onTrack
- an exit action called setMode with parameter offtrack
- an internal activity called followTarget

Note that Entry, Exit, and Internal activity are represented respectively as follows: (Notations may differ in various tools)

- entry/Name of Entry Activity
- exit/Name of Exit Activity
- do/Name of Internal Activity



2.4: State Chart Diagrams

Parts of Transition

- Transition includes:
  - **Event**: The "trigger"
  - **Guard**: A logical condition which returns true or false. It is evaluated at the time of event triggering.
  - **Action**: Gets executed when transition is fired.

$\xrightarrow{\textbf{Event [Guard]/ Action}}$

State Chart Diagrams: Notations for Transitions:

Transitions are associations between "states" and "model relationships" between states.

Transitions are shown by using arrows between the two states.

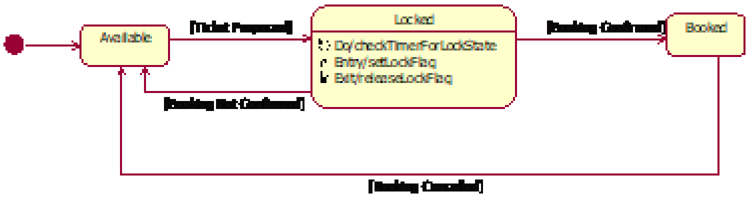
Transition occurs as a result of some event.

The event is indicated along the arrow.

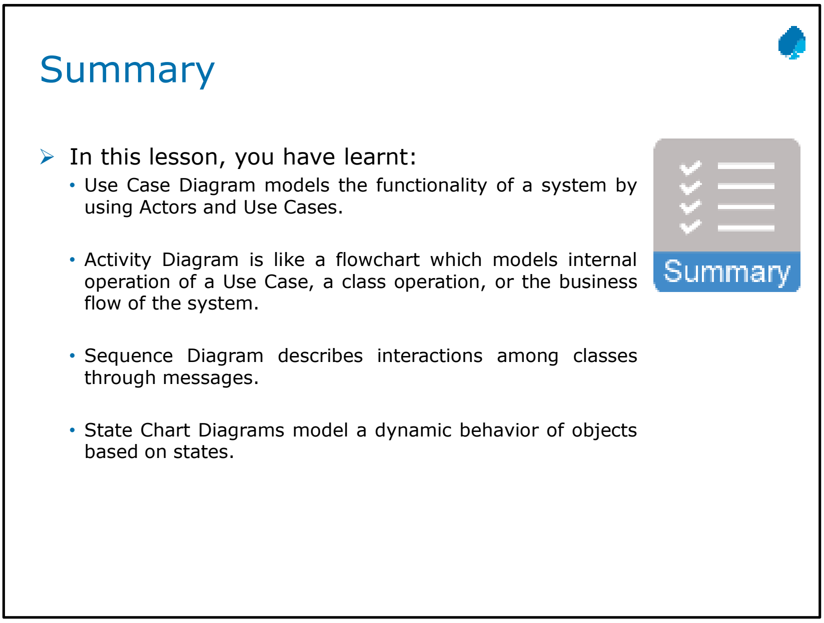
The action that occurs as a result of the trigger is also shown along the arrow.

2.4: State Chart Diagrams

Example



How do you interpret this diagram?



Summary

- In this lesson, you have learnt:
  - Use Case Diagram models the functionality of a system by using Actors and Use Cases.
  - Activity Diagram is like a flowchart which models internal operation of a Use Case, a class operation, or the business flow of the system.
  - Sequence Diagram describes interactions among classes through messages.
  - State Chart Diagrams model a dynamic behavior of objects based on states.

Summary

Review Question

- Question 1: ___ and ___ are called pseudo-states.
- Question 2: A state can have sub states.
  - True / False
- Question 3: A State Transition Diagram must have ___ start state and ___ end states.

Review Question

- Question 4: Start and End States can be found in which diagrams?
  - Use Case Diagram
  - Activity Diagram
  - Sequence Diagram
- Question 5: Use Case Diagrams represent the functionality needed in a system.
  - True / False
- Question 6: A message in a Sequence Diagram will help identifying ___.

Review Question: Match the Following

1. Activity Diagram	**A. Lifelines, Messages, Activation**
2. Sequence Diagram	**B. Actors, Use Cases**
3. Use Case Diagram	**C. Swimlanes, Parallel Paths, Start and End States**