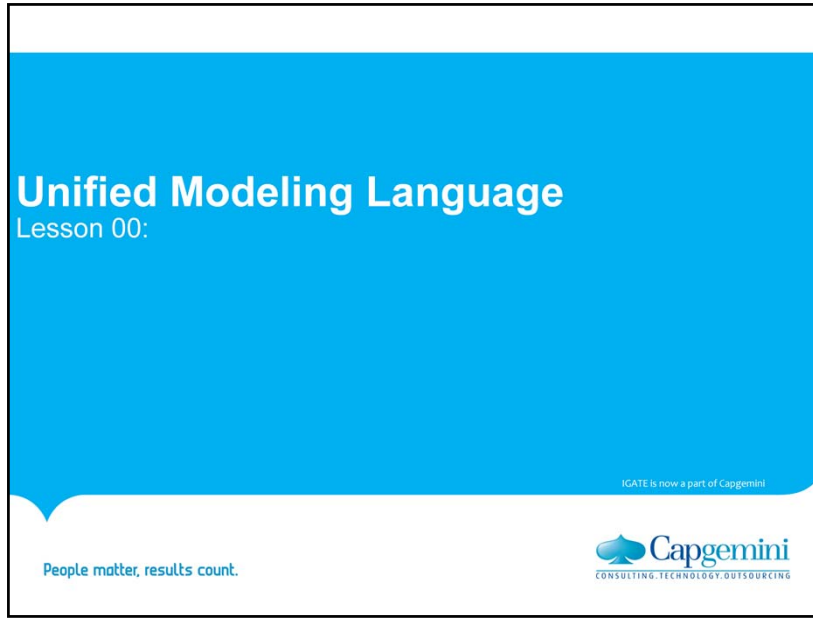


Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
06-Oct-2008	0.1D	NA	Vaishali Kunchur	Content Creation
09-Dec-2008		NA	CLS team	Review
Jan-2009	1.0	NA	Nilendra Nagwekar	Review
08-May-2009	1.2	NA	Veena Deshpande	Updates based on Repository Review Comments
05-May-2011	1.3	NA	Veena Deshpande	Updates as part of Integration Exercise

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 2

Course Goals and Non Goals

- Course Goals
  - At the end of this program, participants gain an understanding of the need of UML and different diagrams in UML.
- Course Non Goals
  - Detailed design and integration is not the part of this course.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Pre-requisites

- Fair Knowledge of OOP

Intended Audience

- Programmers and Designers in Object-Oriented
- Technology

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 5

Day Wise Schedule

- Day 1
  - Lesson 1: Introducing UML
  - Lesson 2: Dynamic View Diagrams(contd on Day 2 also)
- Day 2
  - Lesson 3: Static View Diagrams
  - Lesson 4: General and Extension Mechanisms in UML

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Table of Contents

- Lesson 1: Introducing UML
  - 1.1. Principles of Modeling
  - 1.2. What is UML? What UML is NOT?
  - 1.3. UML Building Blocks
  - 1.4. UML Diagrams
- Lesson 2: Dynamic View Diagrams
  - 2.1. Use Case Diagrams
  - 2.2. Activity Diagrams
  - 2.3. Sequence Diagrams
  - 2.4. State Chart Diagrams

Table of Contents

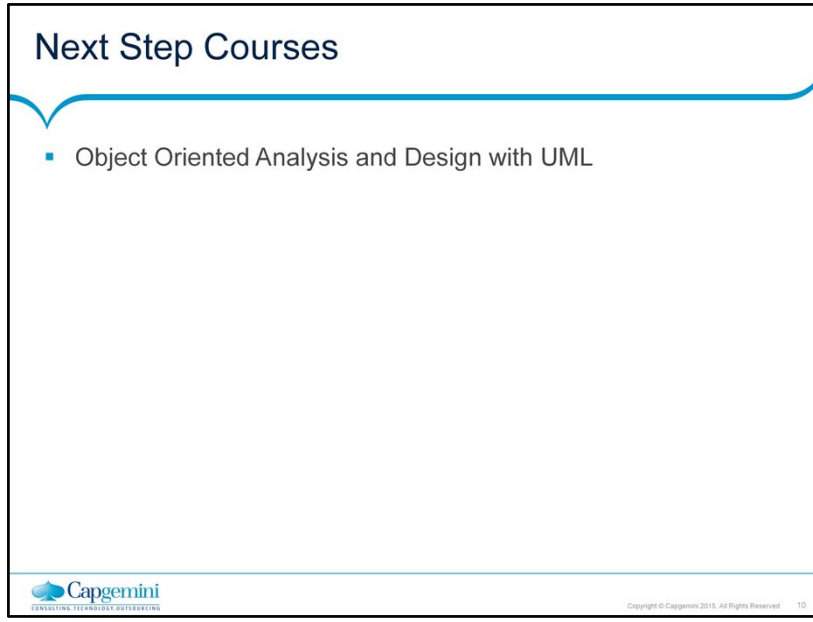
- Lesson 3: Static View Diagrams
  - 3.1. Class Diagrams
  - 3.2. Object Diagrams
- Lesson 4: General and Extension Mechanisms
  - 4.1. UML General Mechanisms
  - 4.2. UML Extension Mechanisms

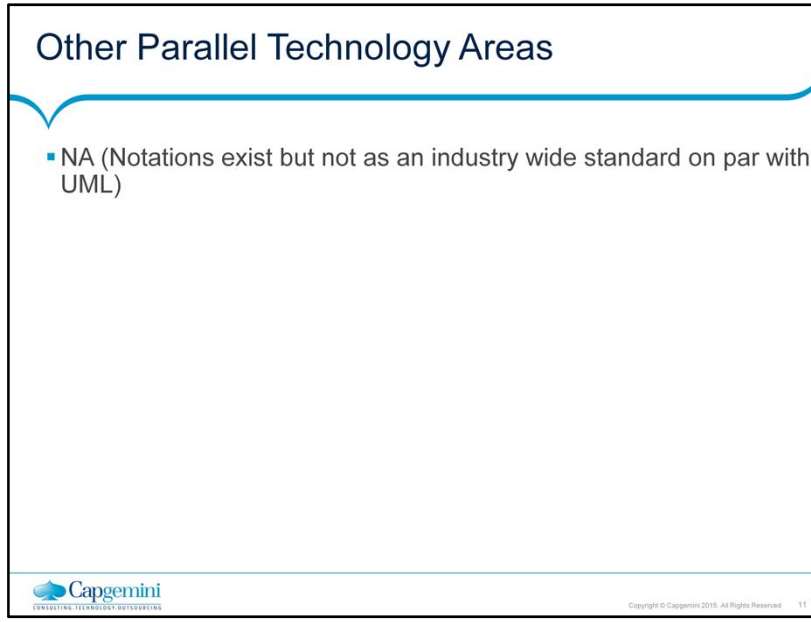
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 8

References

- Student material:
  - Class Book (presentation slides with notes)
- Book:
  - UML User's Guide; by Grady Booch, Ivar Jacobson, and James Rambaugh
- Web-site:
  - http://www.uml.org/


Next Step Courses
Object Oriented Analysis and Design with UML
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
10


Other Parallel Technology Areas
NA (Notations exist but not as an industry wide standard on par with UML)
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
11

Unified Modeling Language

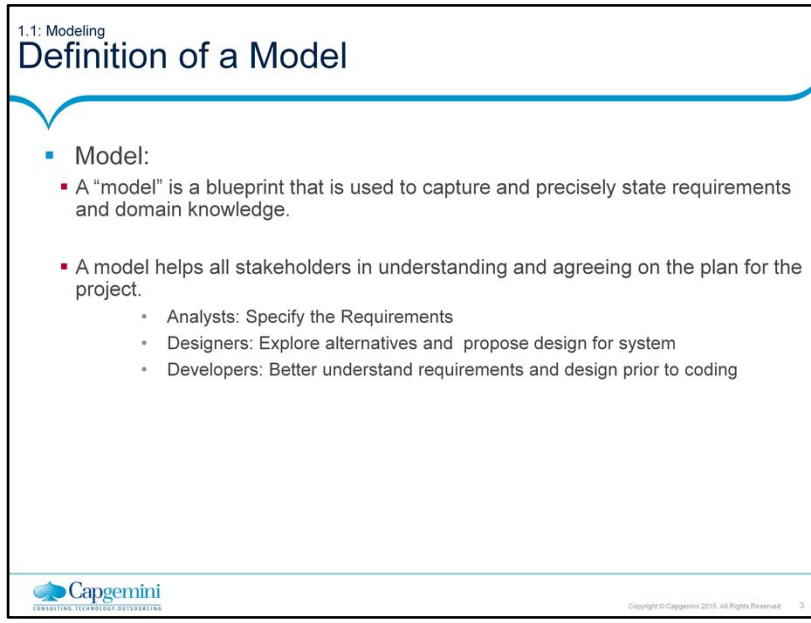
Lesson 1: Introducing UML

Lesson Objectives

- To understand the following topics:
  - Principles of Modeling
  - Basics of UML – What is UML? What UML is NOT?
  - UML building blocks
  - List of UML diagrams

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Principles of Modeling: What is Modeling?

Modeling is an essential activity in many domains, including the fields of construction and engineering.

Actual building of a house is almost always preceded by a blueprint, a model, which describes the architectural layout and other details. Such a blueprint is essential for understanding the system and to convey the same to concerned parties.

The same concept applies to software systems as well. Models can be used to capture the knowledge about the system. A model helps to capture and precisely state the requirements and domain knowledge so that all stakeholders may understand and agree on the plan for the project.

Different models of a software system may capture the following:

requirements about its application domain,

the ways in which users will use the application,

its breakdown into application modules,

common patterns used in its construction, etc.

Thus modeling helps the developers easily explore several architectures and design solutions before writing code.

Models have two major aspects:

Semantic information (semantics)

Visual presentation (notation)

A model can tell what a function does (specification), and also how the function is accomplished (implementation).

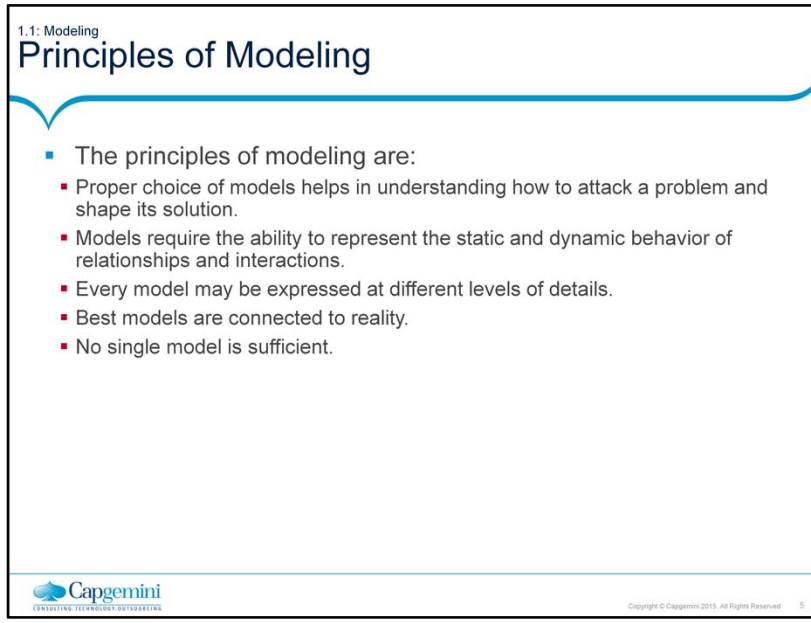
1.1: Modeling

Features of Modeling

- Modeling can be used:
  - to simplify complexity and understand working of system before it is actually built
  - to communicate the desired structure and behavior of the system
  - to visualize and control the system's architecture
  - to allow evaluation of all situations and expose opportunities for simplification and reuse
  - to manage risk
  - to document the decisions that are made

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4



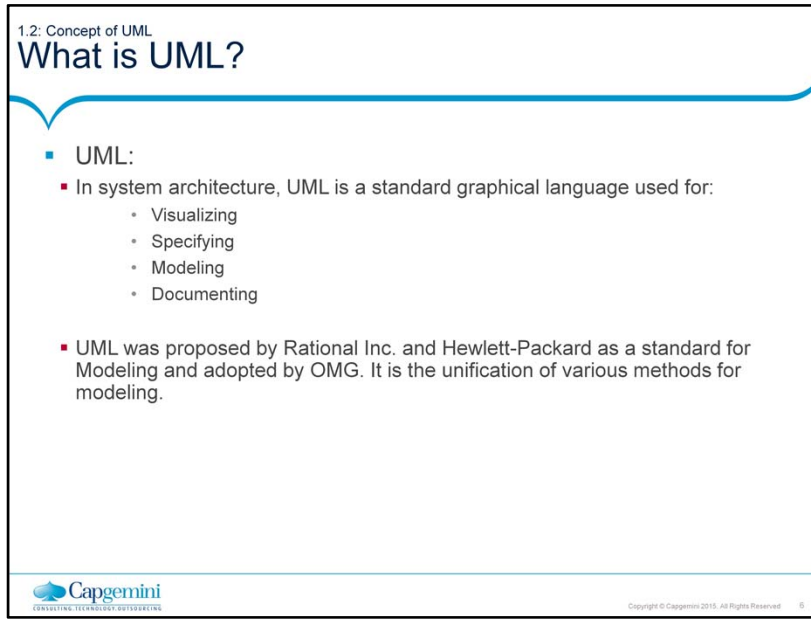
Principles of Modeling (contd.):

While modeling, the problem must always be kept in mind. Only the models that will add value to a “view” of the system must be considered. For example: If a system is supposed to be a stand alone system, having a model to depict the deployment details may not be required. However, such a model would be required for a system which is supposed to be deployed across a network.

Besides, every system has static as well as dynamic aspects. So the models must be capable of depicting the same.

Further, based on the view and the people it is intended for, models may be at different granular levels. Each user may require different degrees of details.

It is unlikely that one model gives the complete system description. This is where multiple models, each giving a different (but relevant) view of the system, becomes important. So proper choice of models is important, which will best help in understanding how to attack the problem and shape its solution.



What is UML?

OMG specification states: “The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system”. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions, as well as concrete things such as programming language statements, database schemas, and reusable software components.

UML is used for the following tasks:

Visualizing - Visual Model helps better communication and goes beyond what can otherwise be textually described.

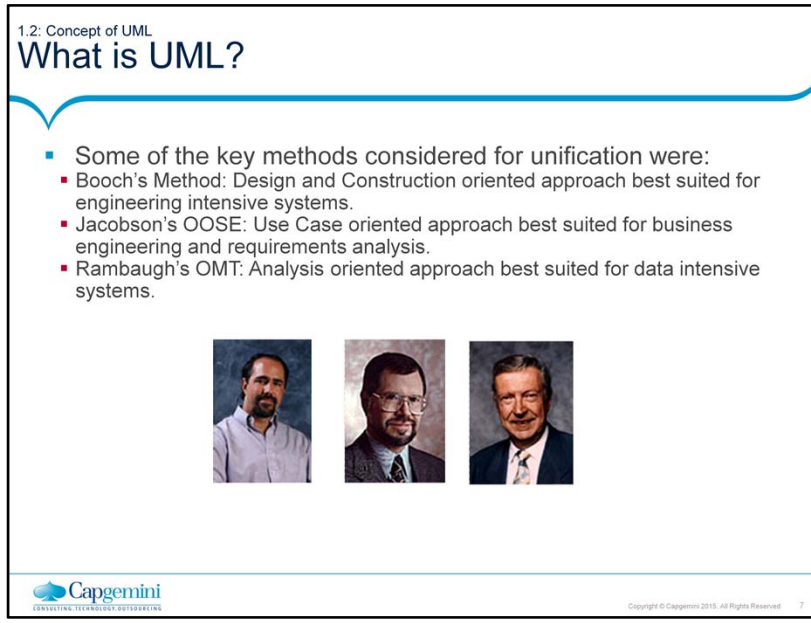
Specifying - UML can help in specifying all important analysis, design, and implementation decisions.

Modeling - Allows for construction of the system from the various models.

Documenting - Models can help in documenting all decisions taken during the entire system development lifecycle.

Prior to UML, there were many methods with similar modeling languages having minor differences in overall expressive power. However, there was no single “leading” modeling language. Lack of disagreement on a general-purpose modeling language discouraged new users from adopting the OO approach.

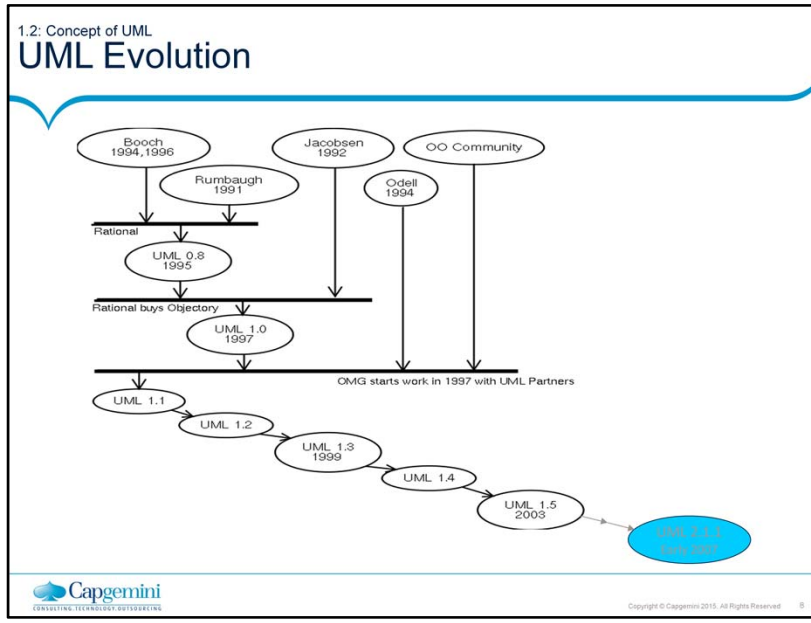
contd.



What is UML? (contd.)

In the period around mid-90s, there were efforts made in the direction of unifying the prominent methods available at that time. After a couple of drafts, UML was adopted by OMG in 1997. Some of the key methods considered for unification were:

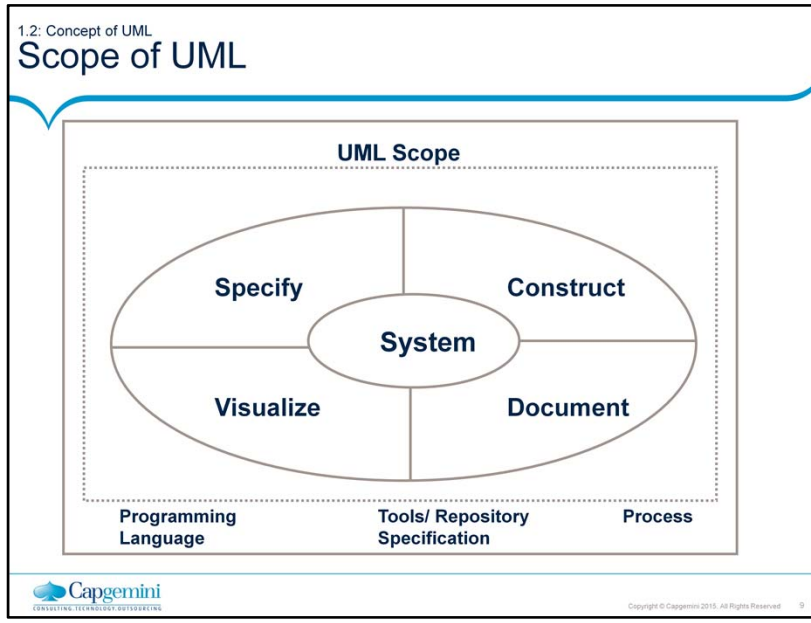
- Booch's Method: Design and Construction oriented approach best suited for engineering intensive systems.
- Jacobson's OOSE: Use Case oriented approach best suited for business engineering and requirements analysis.
- Rambaugh's OMT: Analysis oriented approach best suited for data intensive systems.



Here is a figure illustrating how UML has evolved over a period of time. Ver 2.x

is the current industry standard. We are dealing with Ver 1.4 in our current

course.



Scope of UML:

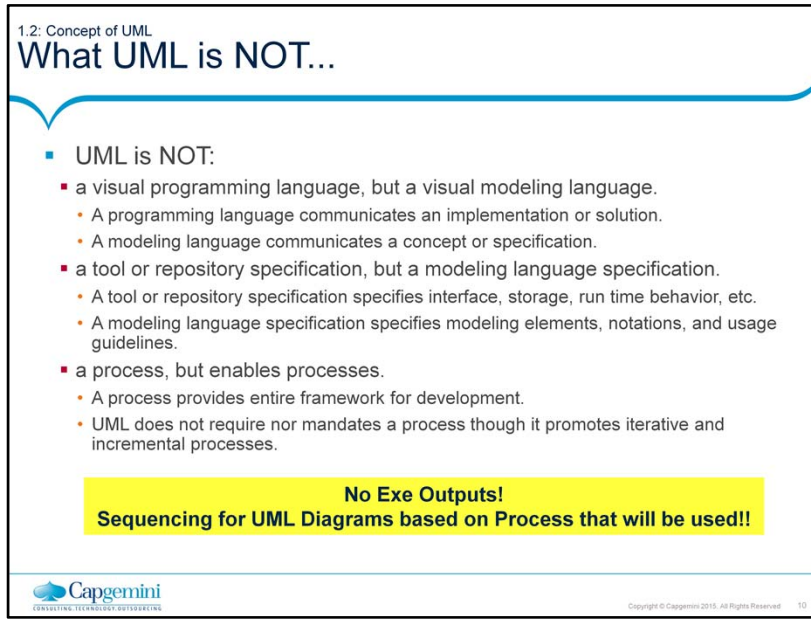
The figure shown in the slide illustrates what is within the scope of UML, and what is outside of the UML scope. While it provides adequate notation and semantics to address contemporary modeling issues, there are some items outside the scope of UML as explained in subsequent slides.

Some of the things that are outside the scope of UML are:

Programming Language: UML is a “visual modeling language”. It is not intended to be a visual programming language. It is a language for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system. It does have a close mapping with OO languages.

Tools: Language standards form the foundation for tools and process. UML defines a semantic meta-model, and not a tool interface, storage, or run-time model.

Process: Software development process will use UML as a common language for its project artifacts, but will use the same type of UML diagram in the context of different processes. UML is process independent.



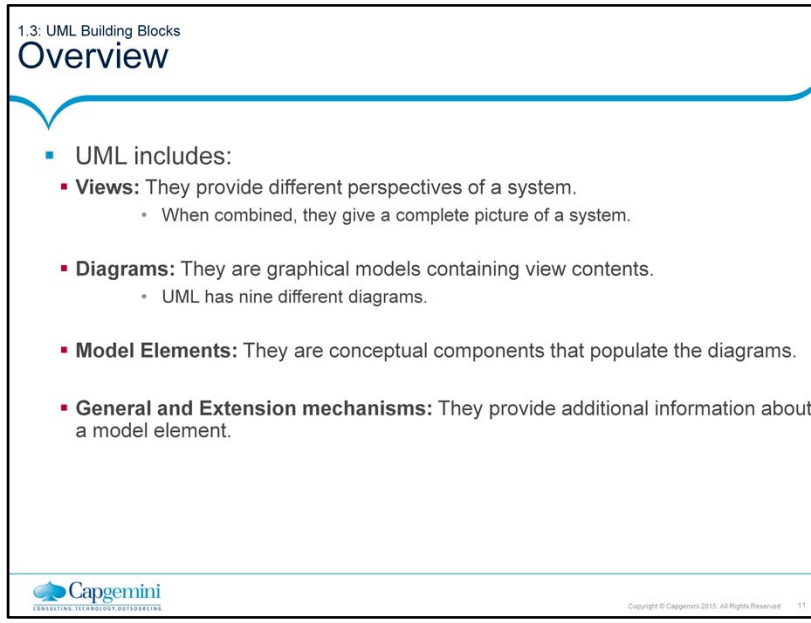
And What UML is NOT …

UML is not meant to be a programming language, rather it is a language meant for modeling. By using UML, one can convey a concept or a specification but not a solution (which a program does).

UML comprises model elements, each with its own associated notation and semantics. UML is not meant to specify a tool or repository in terms of interfaces, storage, or run time behavior.

Similarly, UML is not a process. A process will provide guidance regarding order of activities, and spell out the work products that have to be developed. They are usually domain specific.

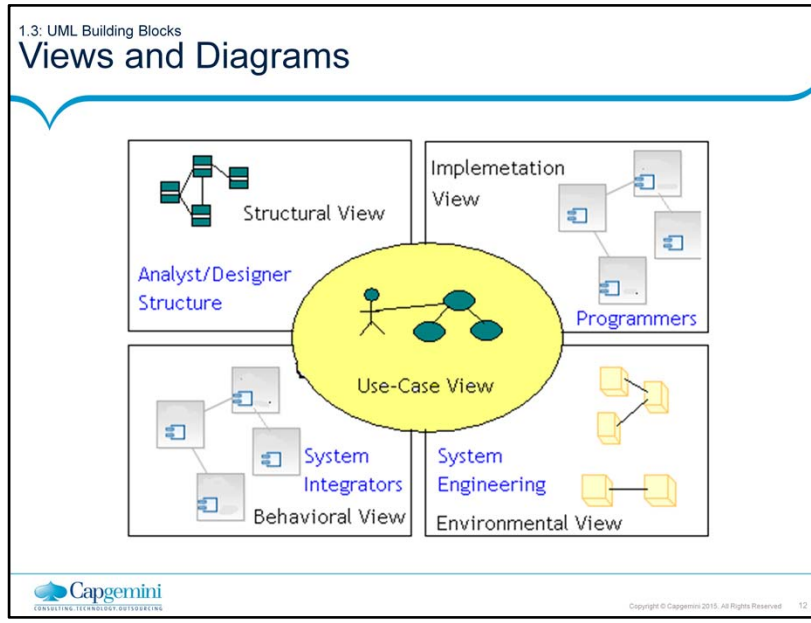
UML does not require a process. However, it enables and promotes Object-Oriented and component-based processes.



UML Building Blocks:

Let us look at the building blocks of UML. UML offers different views of the system, each view containing different diagrams. The diagrams are made up of specific modeling elements.

In addition to existing modeling elements, UML allows extending available notation and semantics by the use of extension mechanisms.



UML Building Blocks – Views and Diagrams:

For the end user, the User view is useful to understand the functionality that will be provided by the system. Various views are:

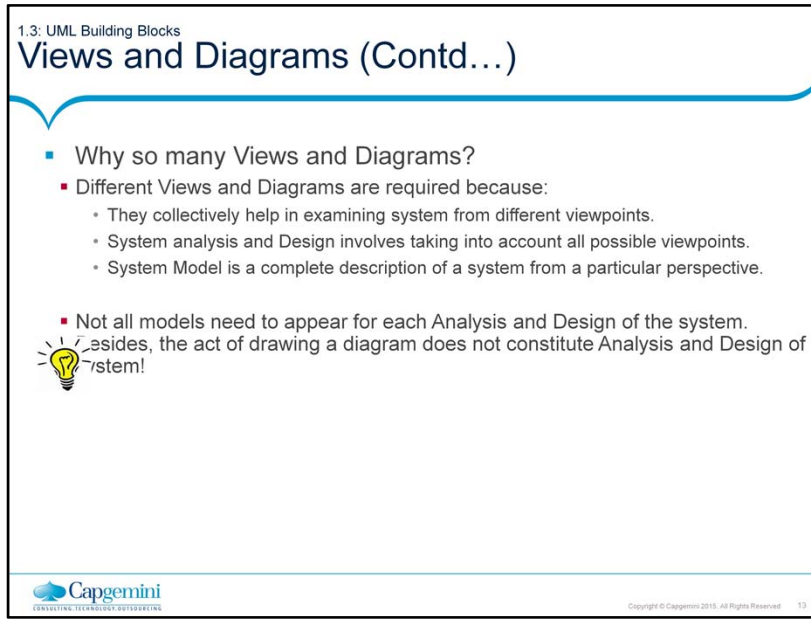
Structural view - Structure diagrams define the static architecture of a model. They are used to model the "things" that make up a model. They are used to model the relationships and dependencies. Analysts and Designers can get the view of the structural aspects of the system through the Structural view.

Behavioral view - It can give important inputs in terms of performance, scalability, and throughput, which can be used by the system integrator.

Implementation view - This view is helpful for programmers.

Environment view – This view can convey decisions relating to system topology, delivery mode, installation, and communication.

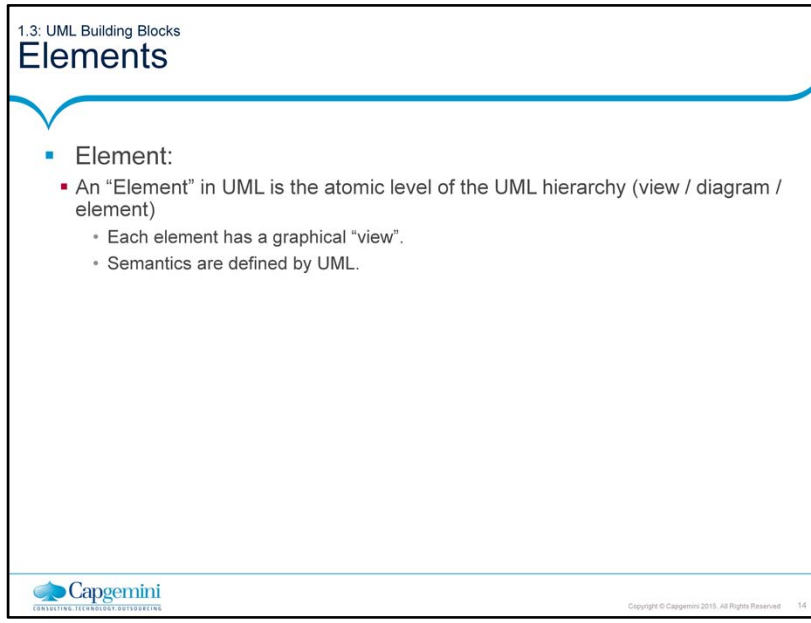
contd.



UML Building Blocks – Views and Diagrams (contd.):

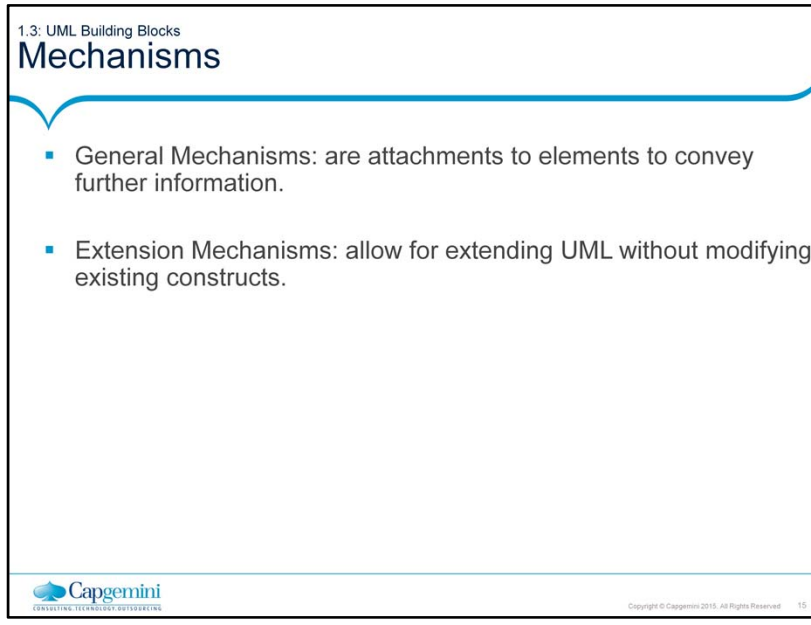
Often one has to decide which views / diagrams are required for the system under consideration. While deciding on this, consider the “reason for communication” of models. Depending on what aspects of the system need to be emphasized on, the views / diagrams can be chosen. (To that extent, each view / diagram is an independent entity in itself). Hence, it may not be required to have all models for each Analysis and Design.

It is important to note that the activity of drawing diagram by itself is not Analysis and Design. Rather, the diagrams are a means of representing and conveying the “Analysis and Design decisions”.



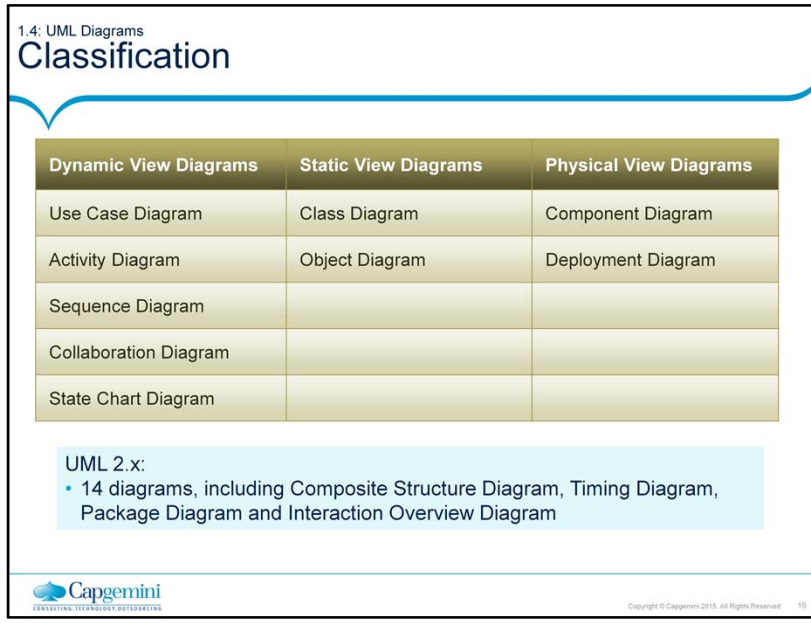
UML Building Blocks – Elements:

Elements form the atomic level of the UML hierarchy. Each element has a predefined “meaning” and a “graphical notation” associated with it.



UML Building Blocks – Mechanisms:

There are some mechanisms available to add on to the expressive power of UML. They are broadly categorized as General mechanisms and Extension mechanisms.



UML Diagrams:

The slides shows a list of the nine diagrams in UML 1.4. Also mentions the additional diagrams of UML 2.x.

In the subsequent sections we will be looking at each UML 1.4 diagram in detail. The sections provide the notations and associated semantics for the constituents of each diagram.

Summary

- In this lesson, you have learnt:
  - Modeling helps simplify complexity and understand the working of a system before it is actually built.
  - UML is a standard graphical language used for:
    - Visualizing
    - Specifying
    - Modeling
    - Documenting
  - UML includes views, diagrams, model elements, general and extension mechanisms.
  - UML diagrams are categorized as:
    - Dynamic View Diagrams
    - Static View Diagrams
    - Physical View Diagrams

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review Question

- Question 1: UML Stands for ___.
- Question 2: UML offers an approach to capture different views of the system.
  - Option: True / False
- Question 3: UML describes a sequence in which diagrams must be drawn.
  - Option: True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review Question: Match the Following

1. Dynamic View Diagrams	A. Use Case Diagram
2. Static View Diagrams	B. Deployment Diagram
3. Physical View Diagrams	C. Class Diagram
	D. Activity Diagram
	E. Sequence Diagram
	F. Component Diagram

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Unified Modeling Language

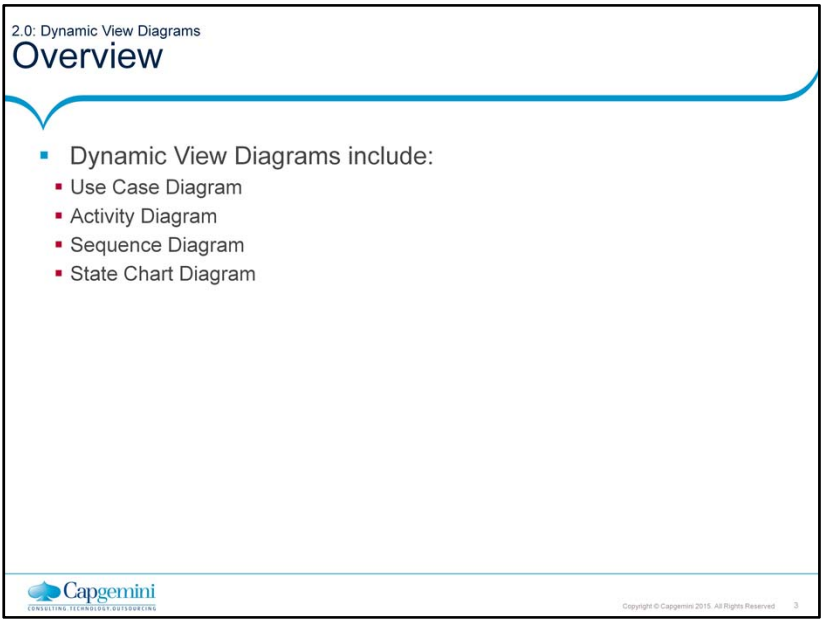
Lesson 2: Dynamic View Diagrams

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Dynamic View Diagrams:

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

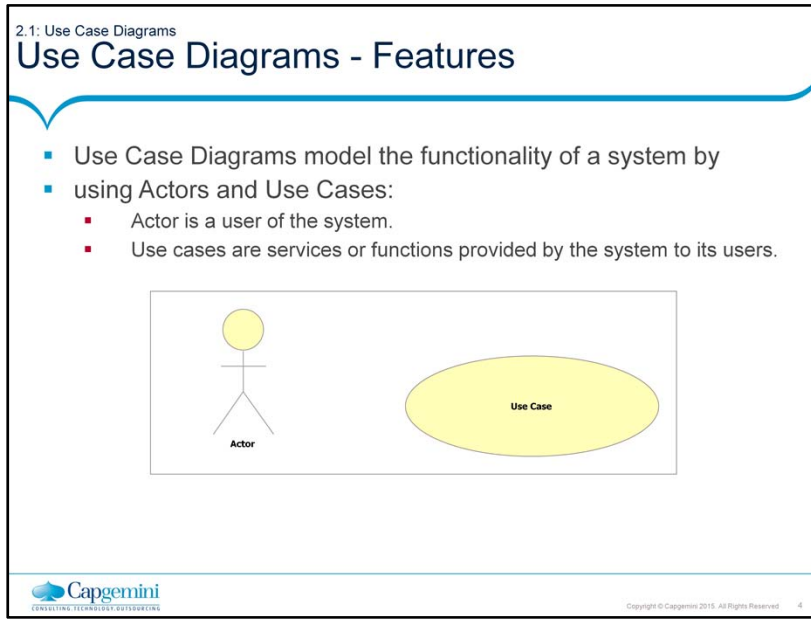
- Dynamic View Diagrams
- Static View Diagrams
- Physical View Diagrams

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.



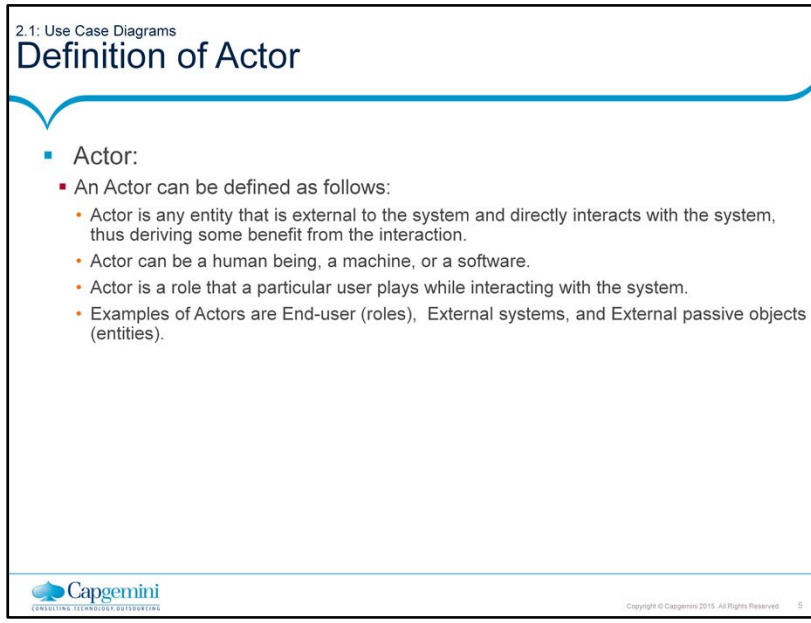
Use Case Diagrams:

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.



Actor:

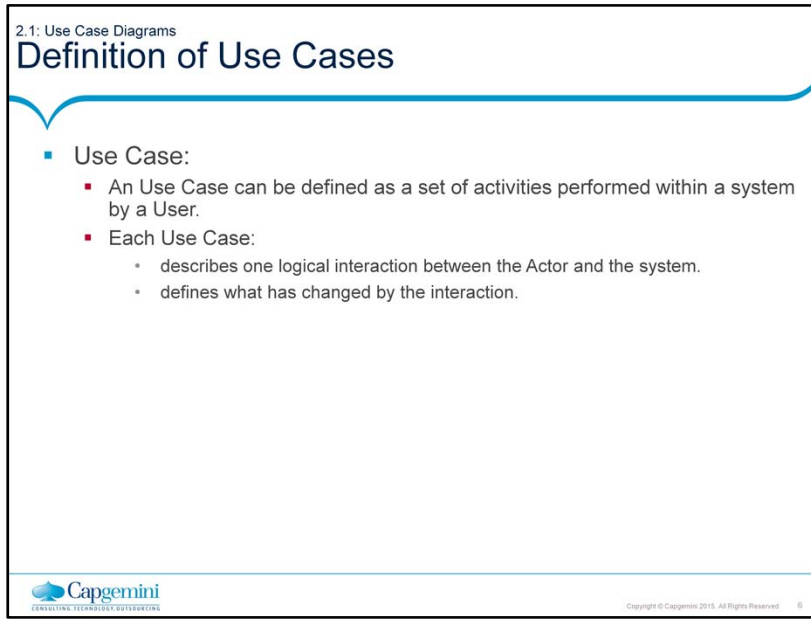
Actors are people, organizations, systems, or devices which use or interact with our system. The system exists to support that interaction. Therefore, the important part of the project is to identify the Actors and find out what they want from the system.

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.



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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

Drawing the Use Case Diagram:

The Use Case Diagram has the following elements:

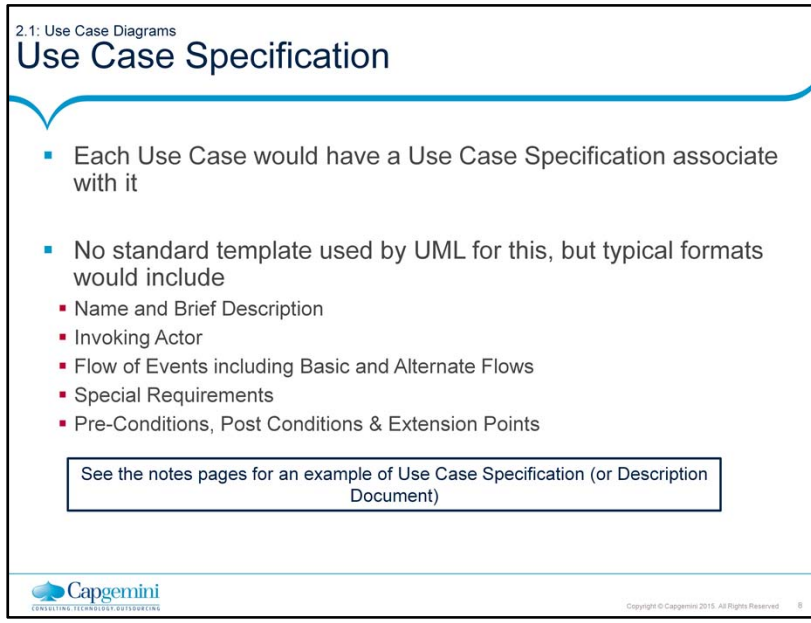
A stick figure, which represents Actors (sometimes stereotyped classes, as explained later, are also used to represent Actors). They differ from tool to tool.

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.



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.)**

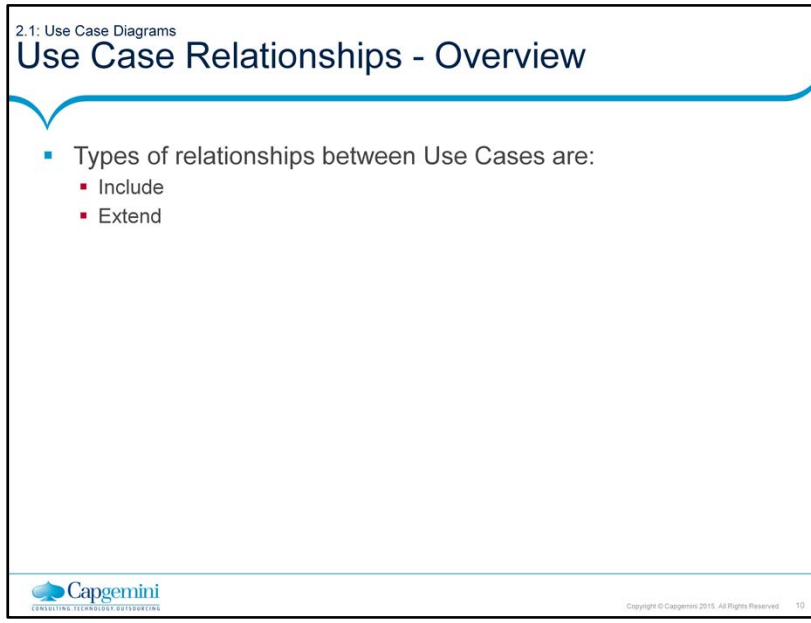
7. The customer specifies the requested details.
8. The system requests for Credit Card Details: Credit Card Type, Credit Card Holder Name, Credit Card Number, Expiry Date, CVV Number and Payment Amount.
9. The customer specifies the details.
10. The system requests the Credit Card Agency to validate 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.



Use Case Relationships:

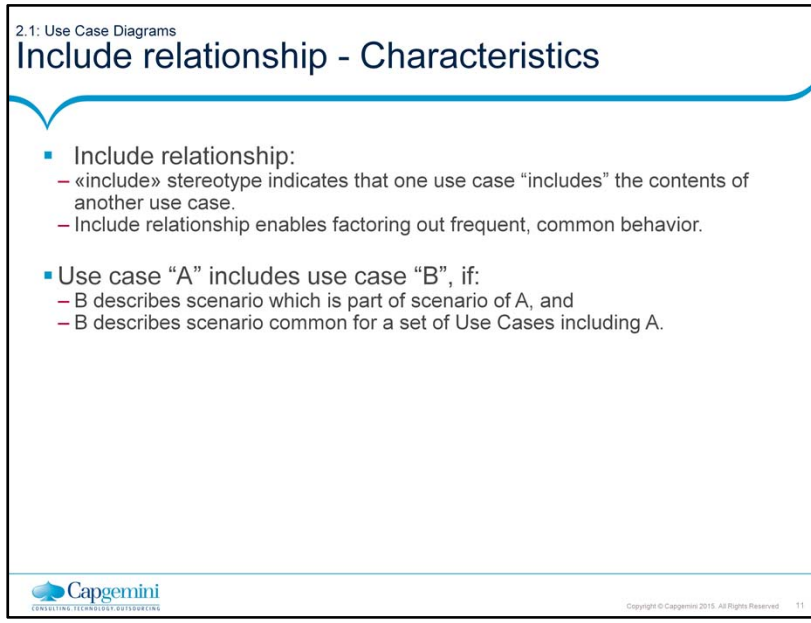
Relationships help us connect the model elements.

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

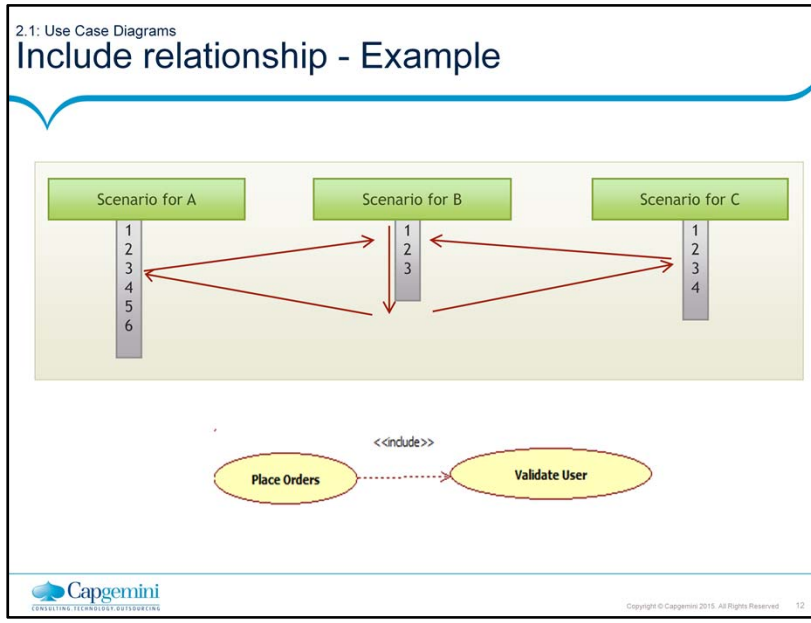


Use Case Relationship – Include:

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.



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.

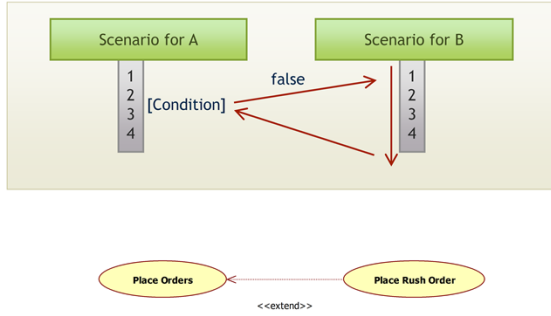
Use Case Relationships - Extend :

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

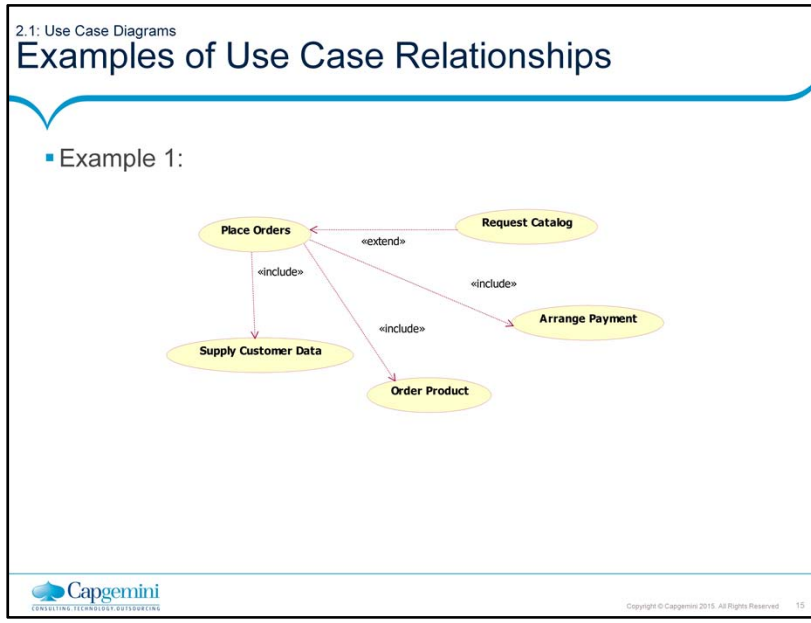
Extend relationship - Example



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.



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

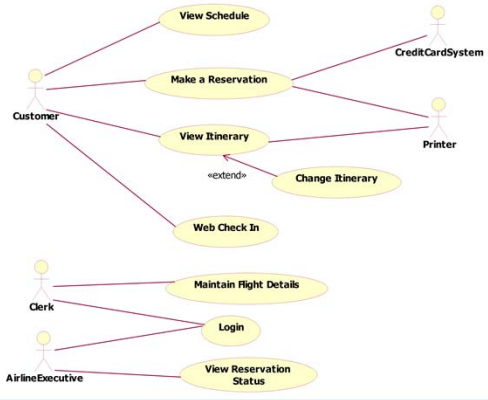
Arrange Payment

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…)

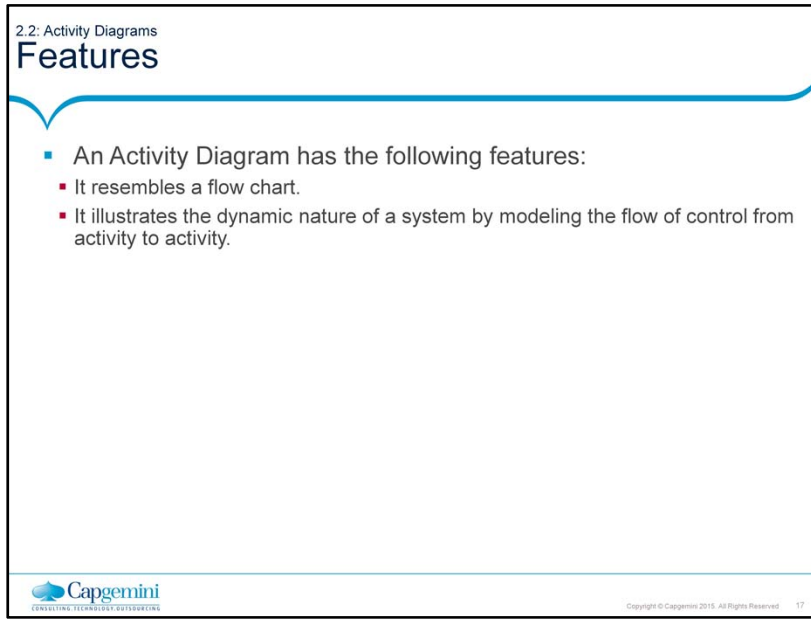
Example 2:



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

How do you interpret this diagram?



Activity Diagrams:

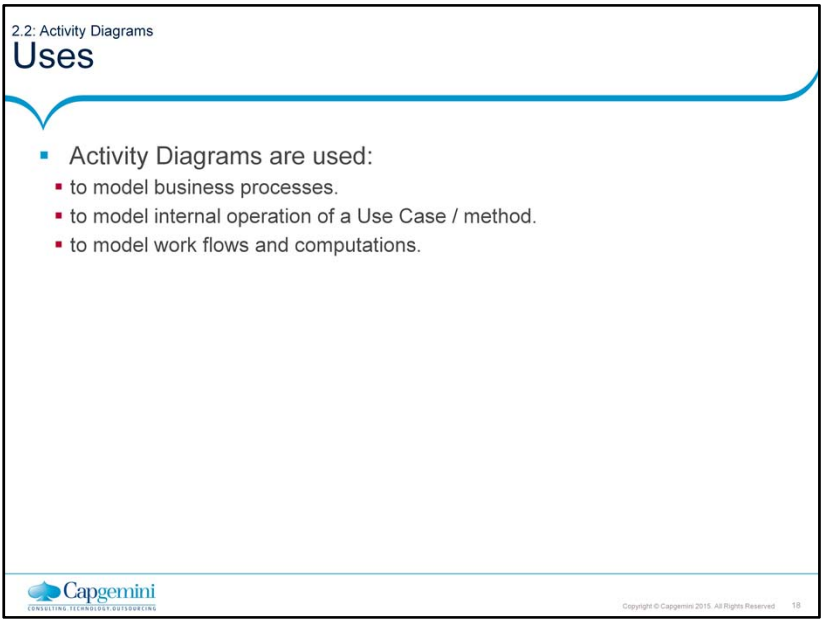
Activity Diagrams are typically used:

for business process modeling,

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

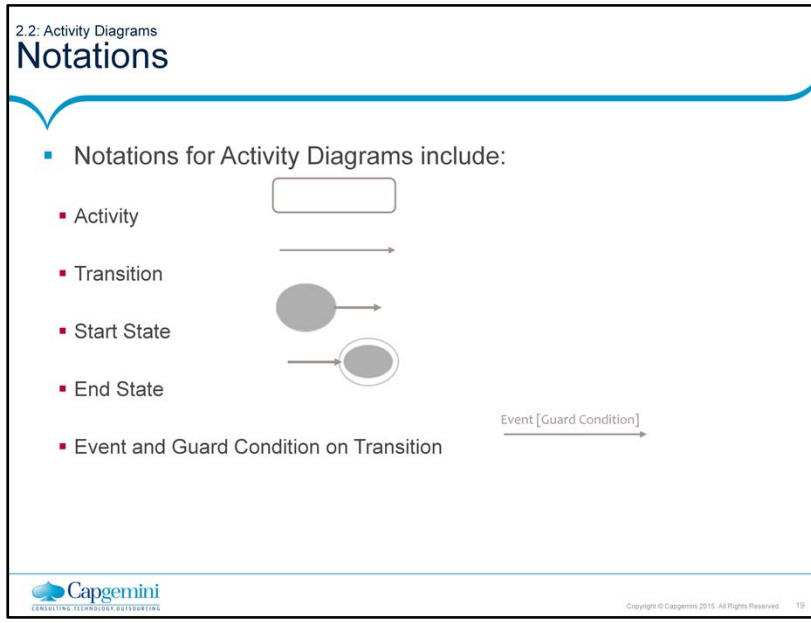
for modeling the detailed logic of a business rule

Activity Diagrams look similar to flow charts.



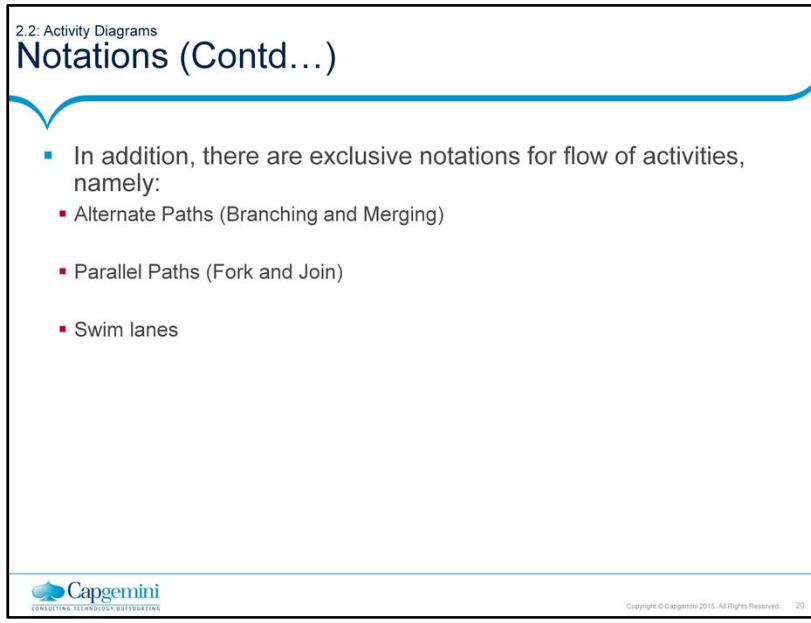
Uses of Activity Diagrams:

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.



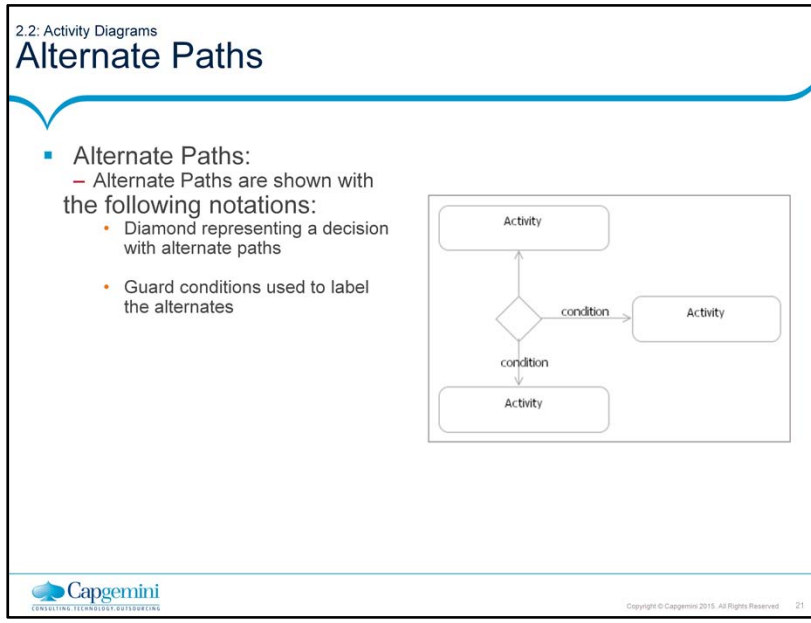
Activity Diagrams – Notations:

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.



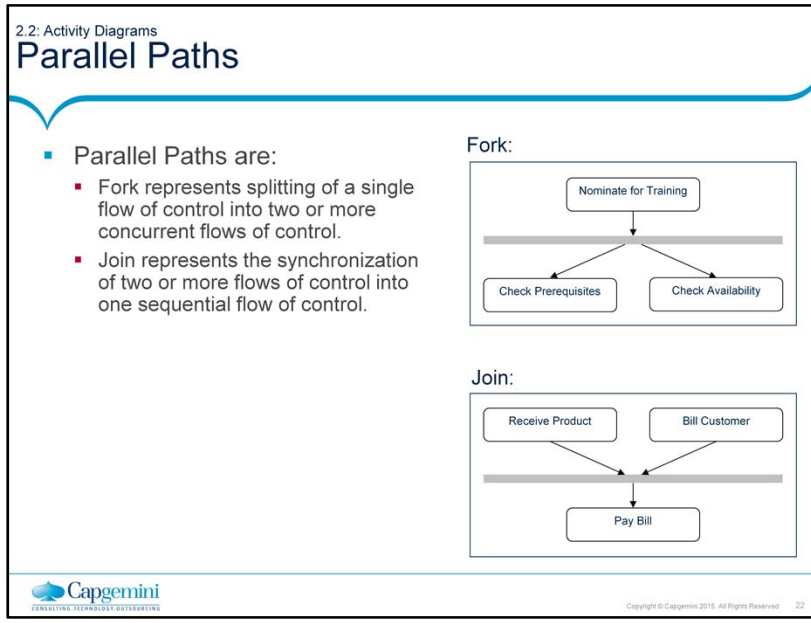
Activity Diagrams – Notations (contd.):

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.



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.

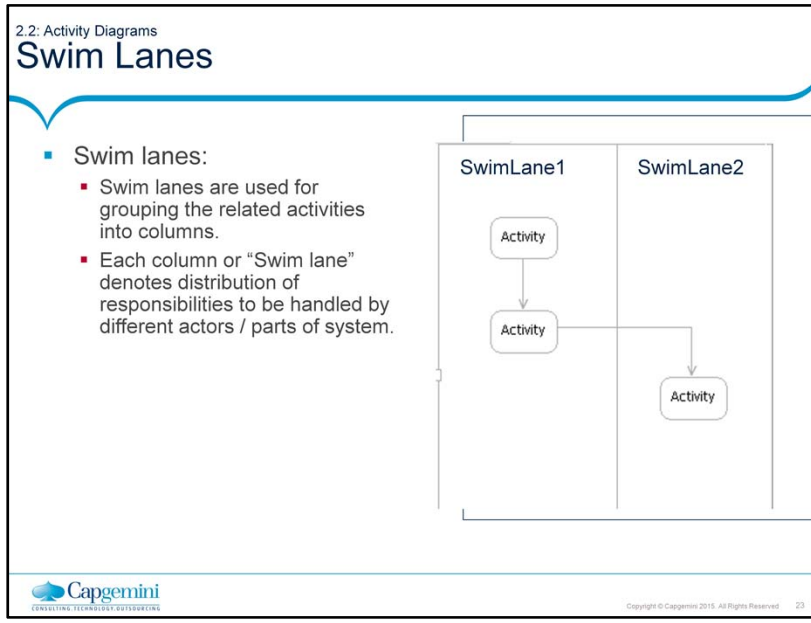


Activity Diagrams – Notations for Activity Flow: Parallel Paths:

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

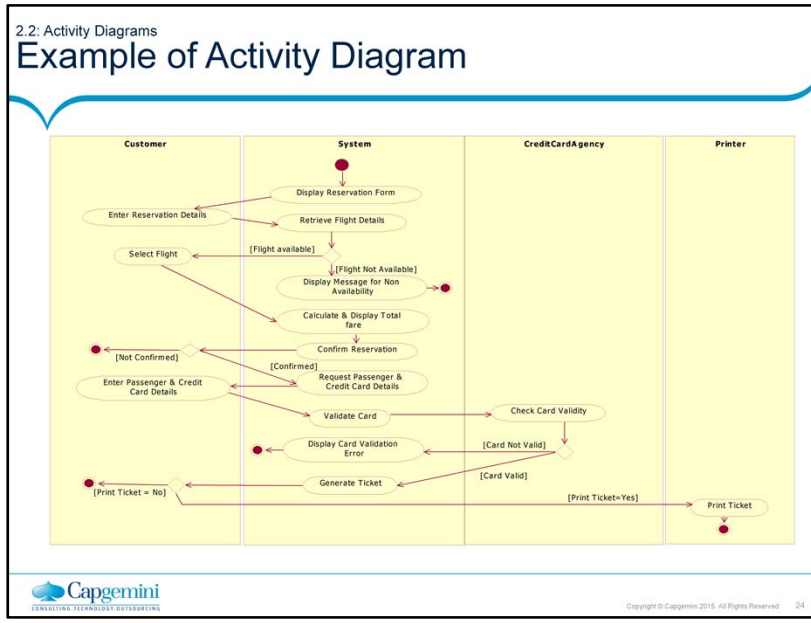
For example:

> “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.

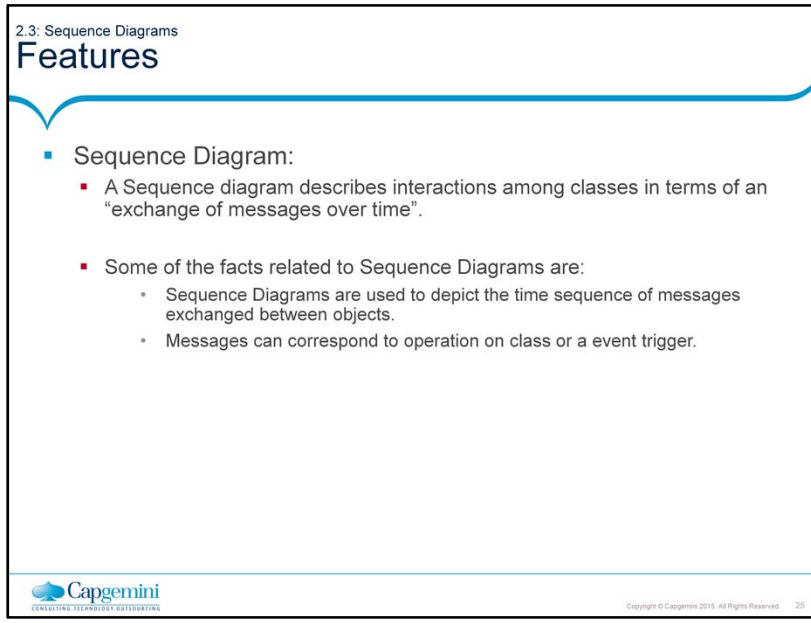


Activity Diagrams – Notations for Activity Flow : Swim lanes:

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.



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



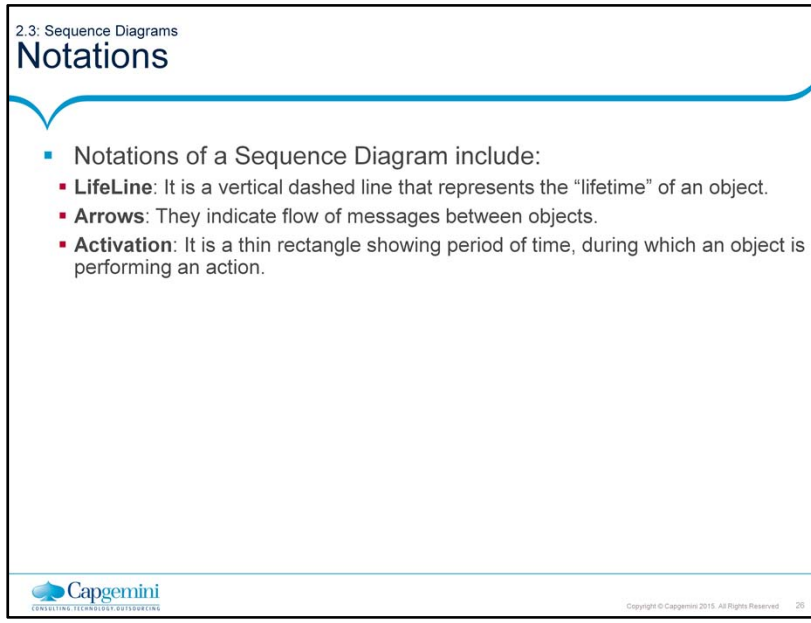
Sequence Diagrams: Features:

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.



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.

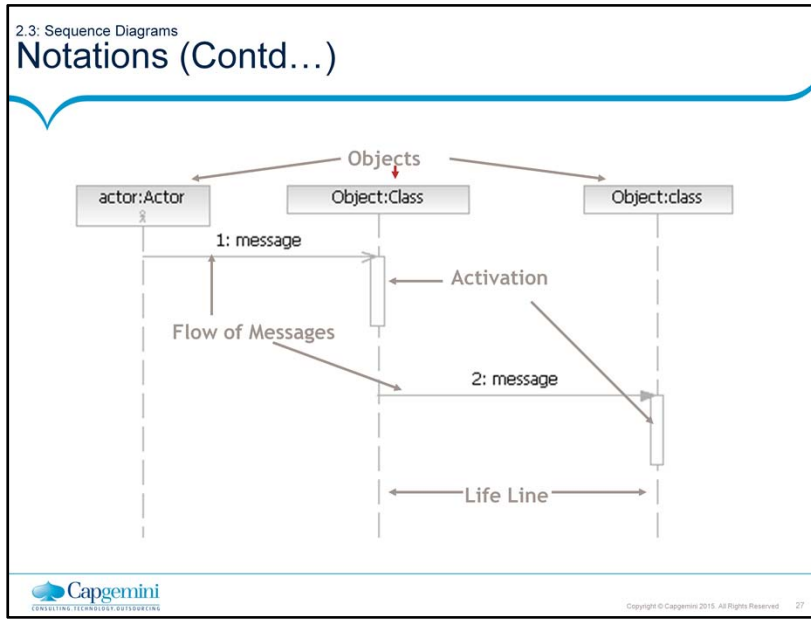
Arrows indicate flow of messages between objects.

Messages are denoted as horizontal arrows between lifelines.

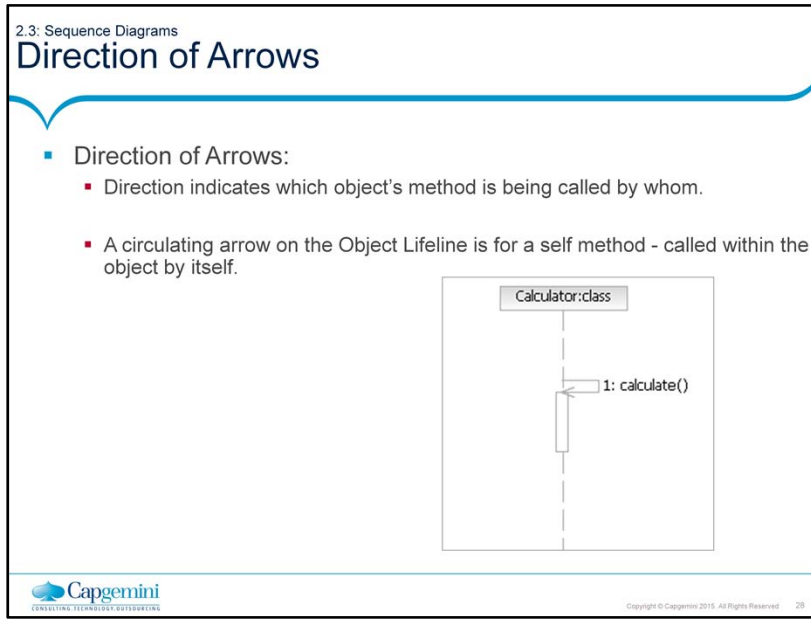
Messages may be:

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

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



The slide shows the notations used in a Sequence Diagram.



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.

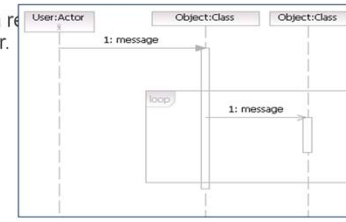
“Implicit returns” are assumed in case there are no return messages to the sender.

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 re
    loop placed at the bottom left corner.



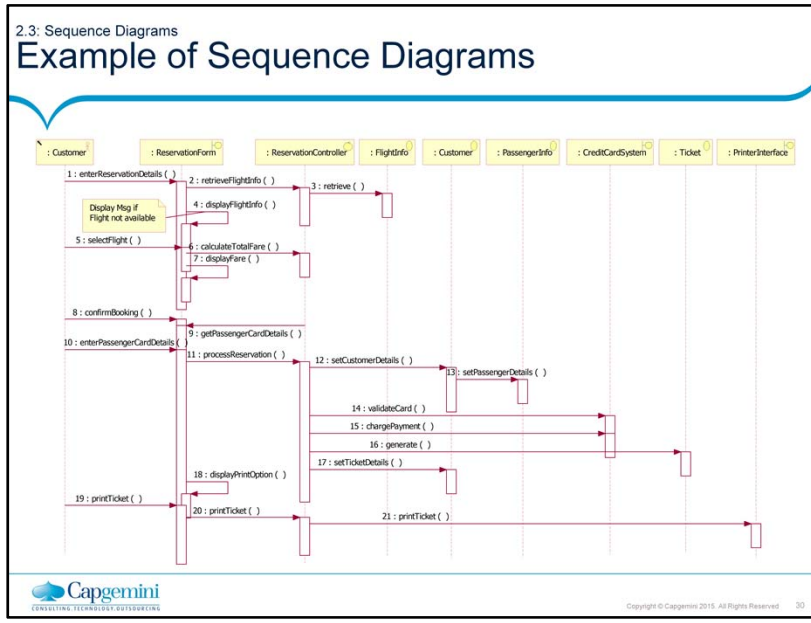
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

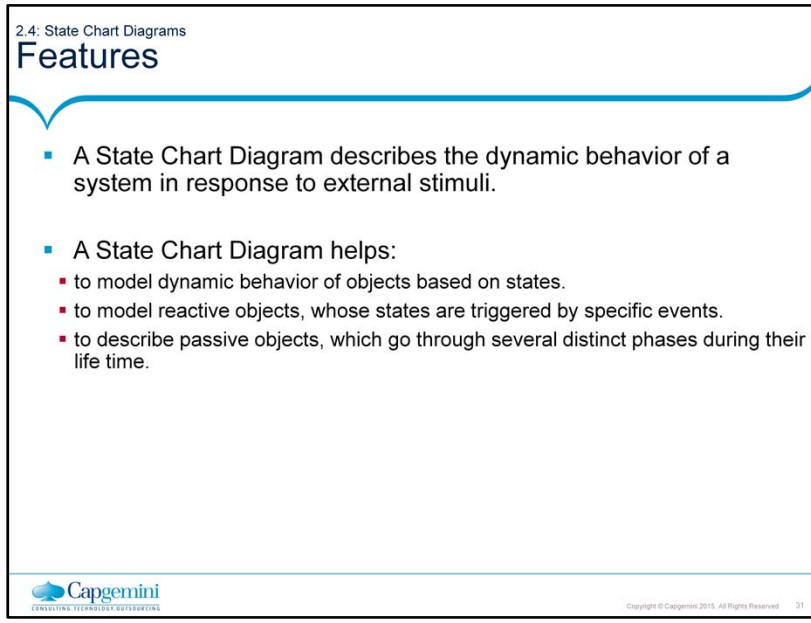
Sequence Diagram: Notations (Directions and Branches):
Branch Conditions:

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.



How do you interpret this diagram?



State Chart Diagrams:

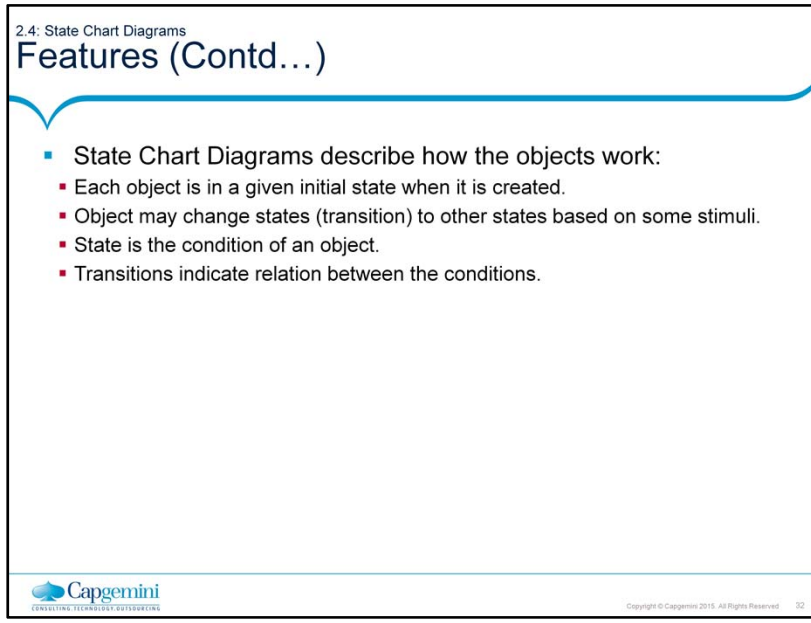
Understanding the state of an object is important as the state of the object determines the behavior of the object. States of an object can be modeled by using the State Chart Diagram.

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.

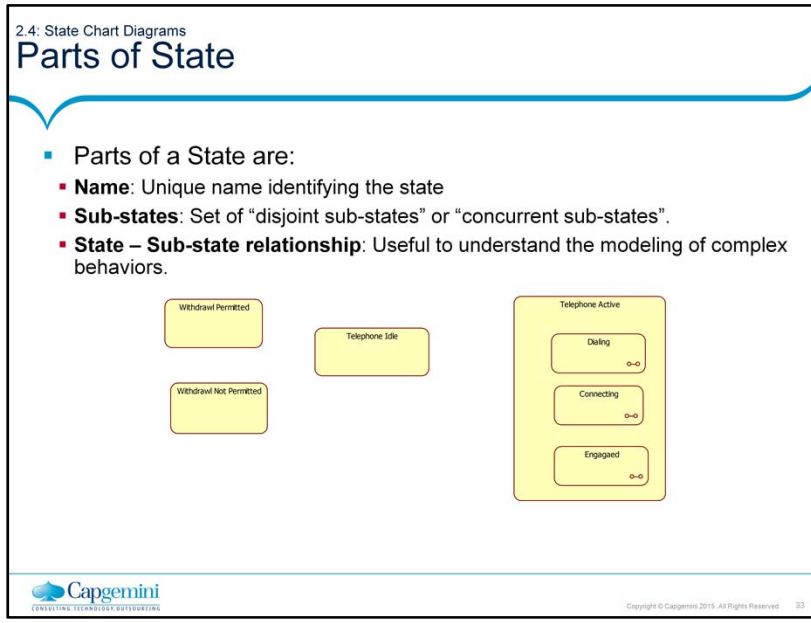
contd.



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.



State Chart Diagrams – Notations for States:

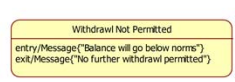
Every state has a unique name which identifies the state. For a state which may be complex, it may be modeled with nested sub-states. These sub-states could be “disjoint” or “concurrent”.

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

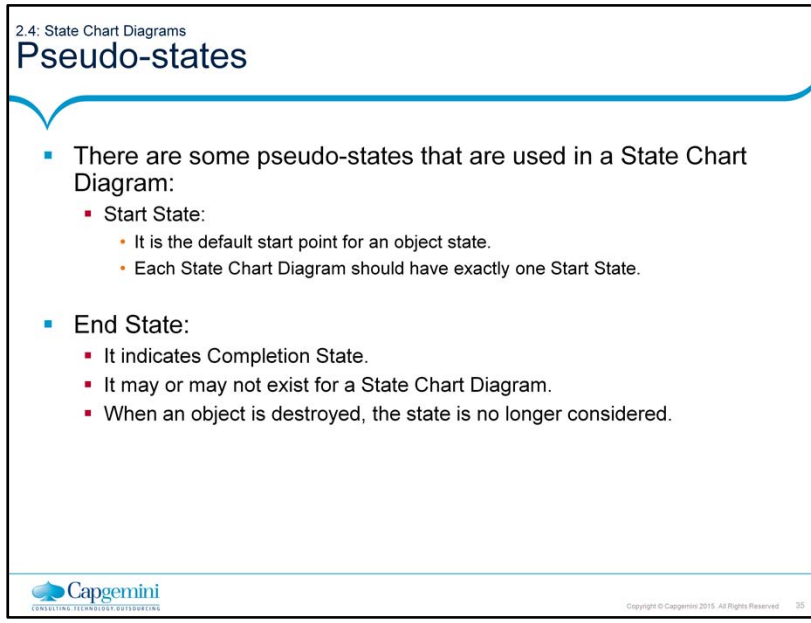
Parts of State

- 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.



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

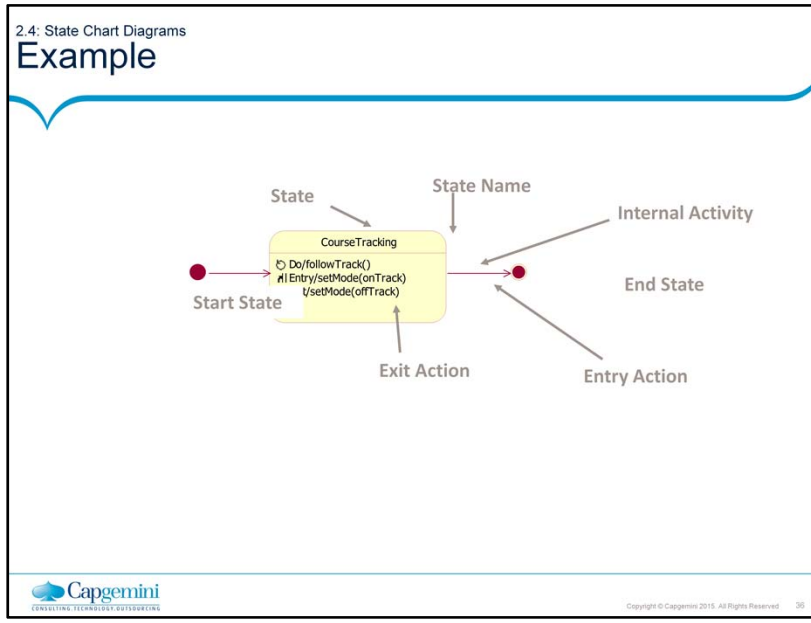
 34



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.



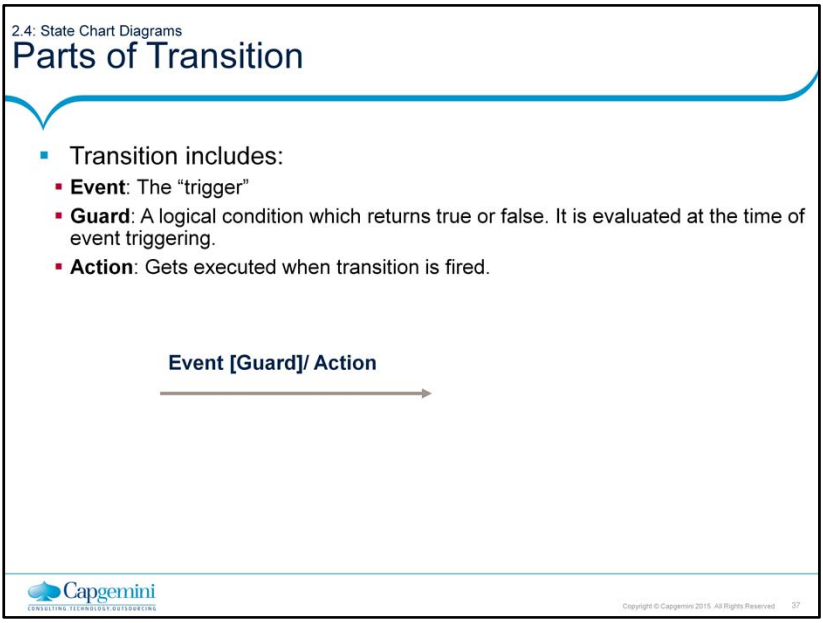
State Chart Diagram: Notations for States (contd.):

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



State Chart Diagrams: Notations for Transitions:

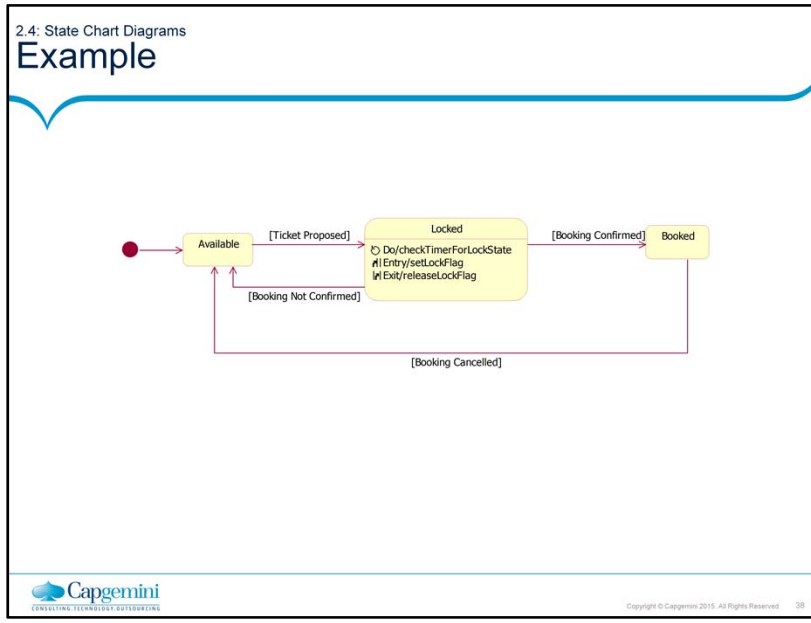
Transitions are associations between “states” and “model relationships” between states.

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.



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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Unified Modeling Language

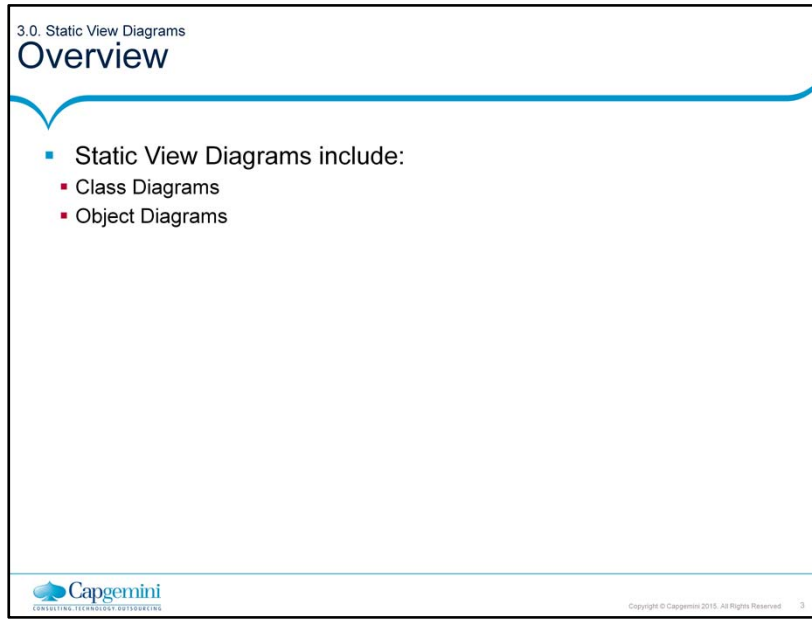
Lesson 3: Static View Diagrams

Lesson Objectives

- To understand the following topics:
  - Static View Diagrams
    - Class Diagrams
    - Object Diagrams

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 2



Static View Diagrams:

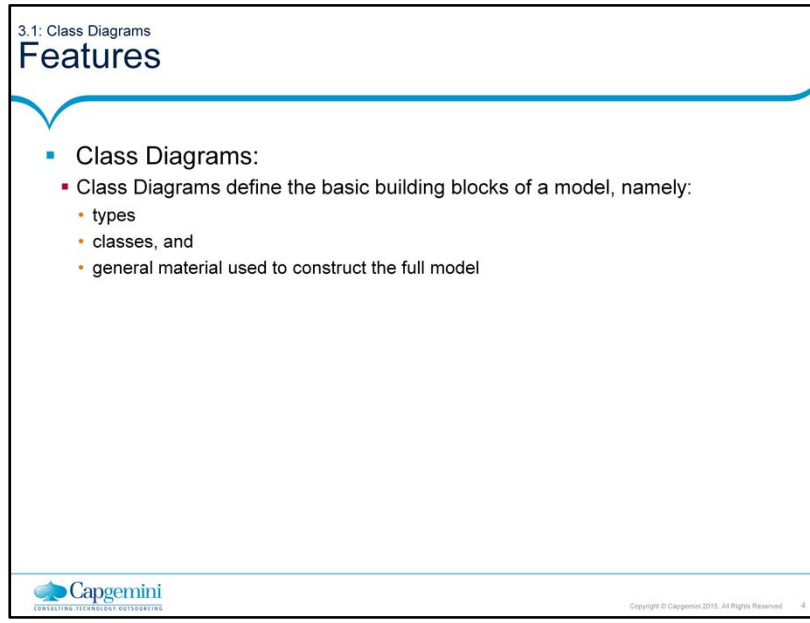
As seen in the earlier lesson, UML diagrams are classified as:

- Dynamic View Diagrams
- Static View Diagrams
- Physical View Diagrams

Let us begin with the Static View Diagrams.

Static View Diagrams include:

- Class Diagrams
- Object Diagrams



Class Diagrams: Features:

Class Diagrams can be used to model classes, and the relationships between classes.

When drawn during the analysis stage, only the names of the classes maybe represented.

During further refinements in the detailed analysis or design stage, details like “attributes” and “services” get added to each class, and are depicted in the Class Diagram.

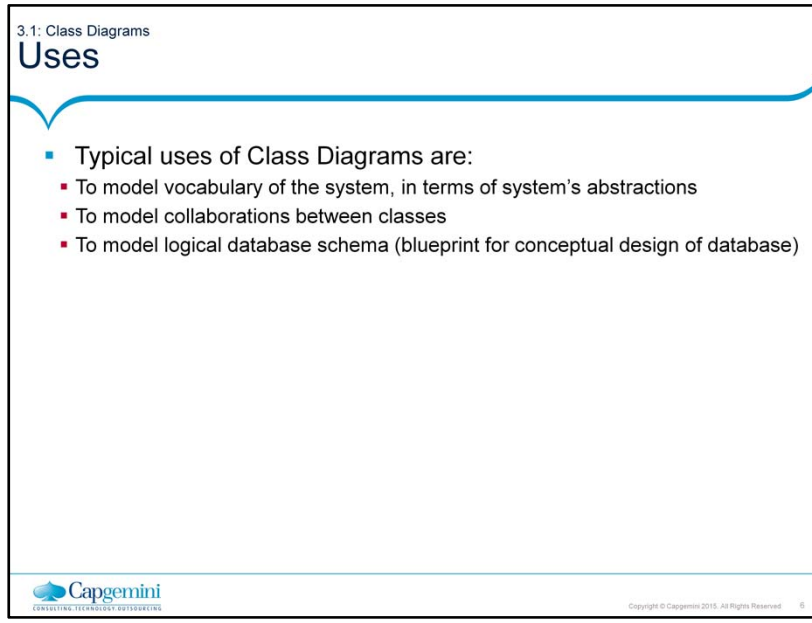
3.1: Class Diagrams

Functions

- Class Diagrams have the following functions:
  - They describe the static structure of a system.
  - They show the existence of classes and their relationships.
    - Classes represent an abstraction of entities with common characteristics.
    - Relationships may be:
      - Generalization
      - Association
      - Aggregation
      - Composition, or
      - Dependency

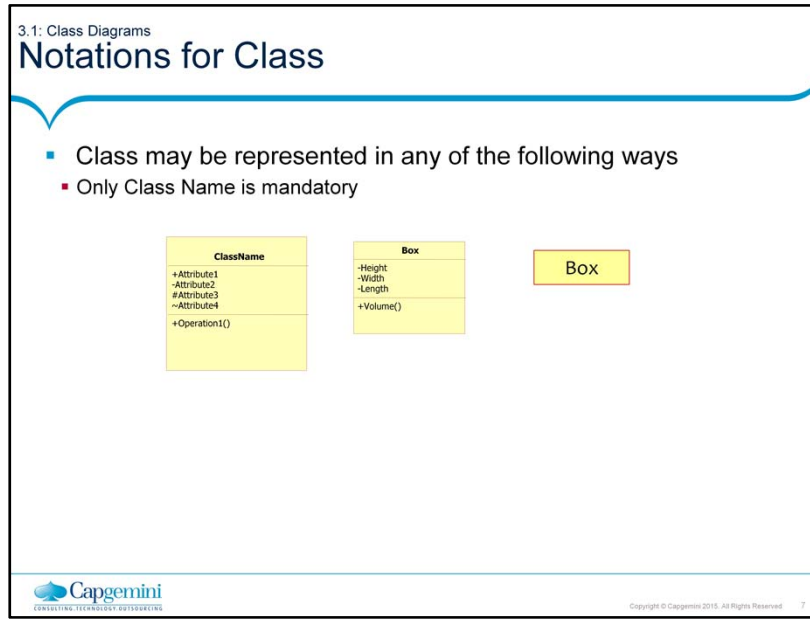
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 5



Uses of Class Diagram:

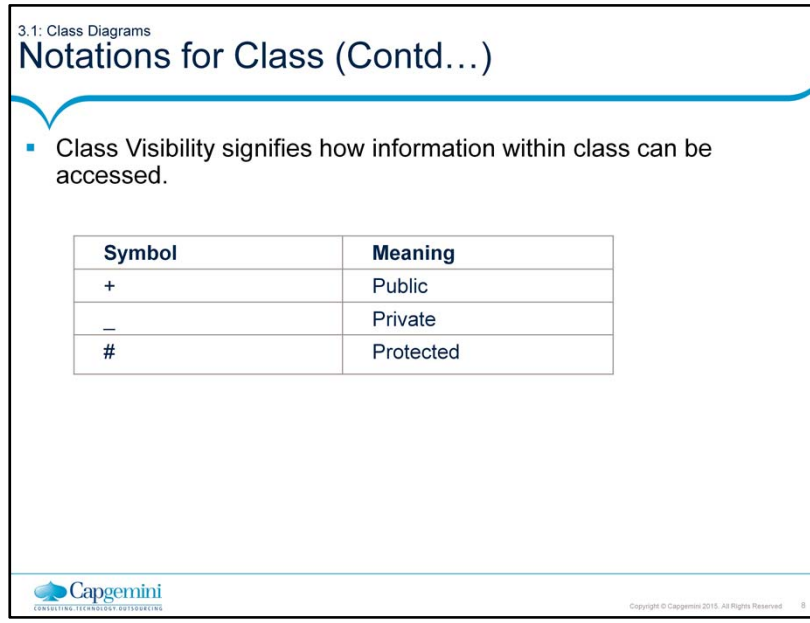
The importance of the Class diagram is that it gives a view of all the classes that are required to make up the system. It also conveys the collaborations that exist between classes to give the system behavior.



Notations for Class:

Classes are denoted as rectangles, with compartments for name, attributes, and operations. There is optionally a last compartment that can be used for specifying responsibilities, variations, business rules, etc.

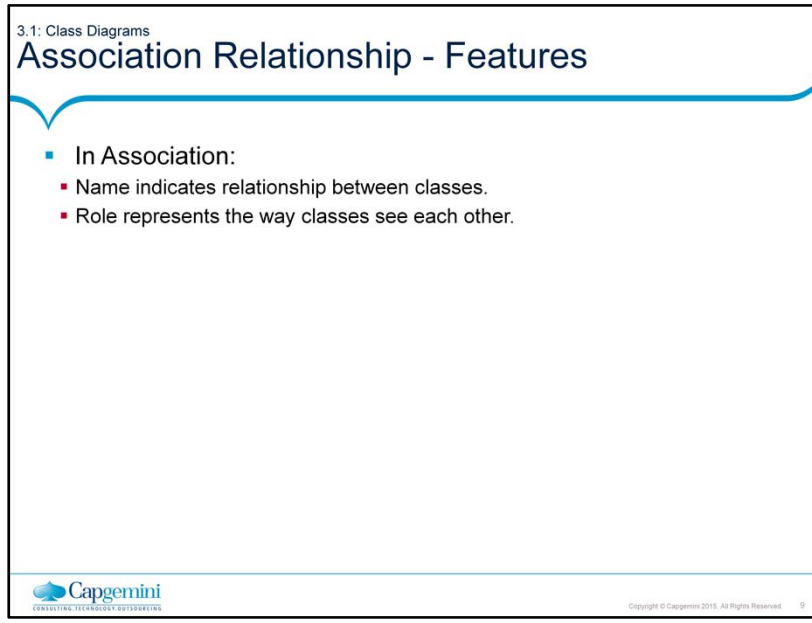
The name is the mandatory part. Other compartments may be included based on the amount of details required to be communicated. The representations of classes that do not have all compartments are known as “elided notations for class”.



Notations for Class: Class Visibility:

Information about visibility of attributes and operations can sometimes be represented by using symbols like + for public, - for private, or # for protected.

These symbols may vary from tool to tool.



Relationships: Association:

Associations may be characterized by the following:

Name: The name signifies purpose of association, and is written along with the line indicating association, role and direction of association.

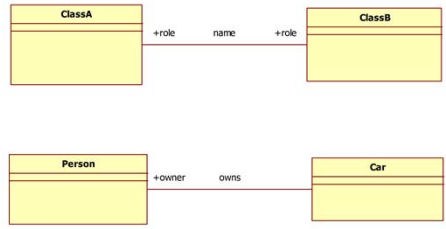
Role: In case there are specific roles played by classes in the association, then it is indicated by the role name, which is written near the class.

Arrow: Arrows may be used to indicate whether the association is uni-directional or bi-directional. Absence of arrows implies that no inferences can be drawn about the navigability.

The example in the slide shows an association relationship between a class Person and a class Car. The class Person plays the role of an owner.

3.1: Class Diagrams

Association Relationship - Example



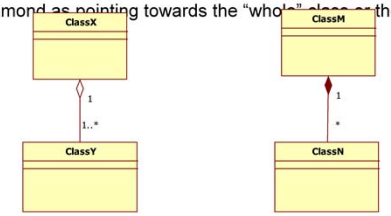
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

3.1: Class Diagrams

Relationships - Features

- Aggregation and Composition:
  - The following Class Diagram, possessing Composition and Aggregation, displays:
    - Aggregation as indicated by a hollow diamond.
    - Composition as indicated by a filled diamond.
    - Diamond as pointing towards the "whole" class or the aggregate.



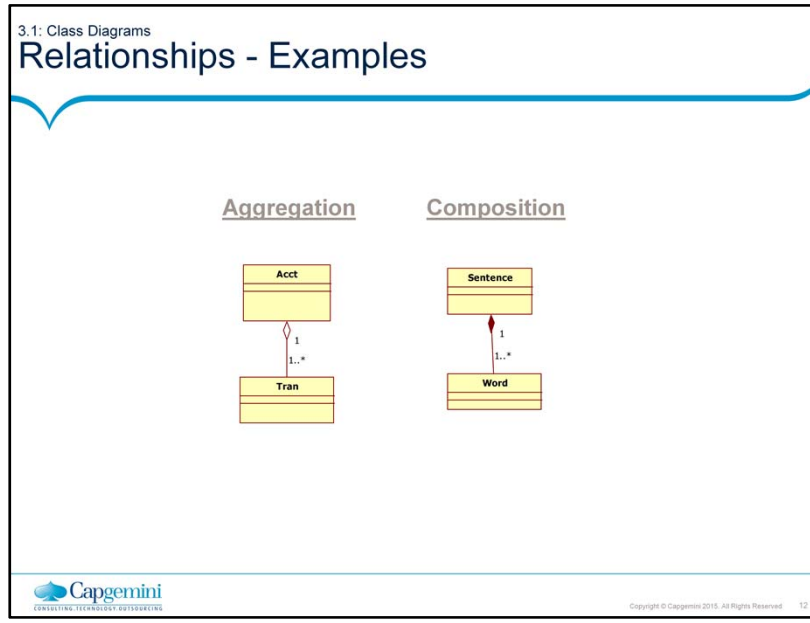
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Relationships: Aggregation and Composition:

Composition and Aggregation are modeled with filled diamond and hollow diamond, respectively, on the "Whole" part.

Roles and multiplicity, if required, can be mentioned here, as well. Typically they are done for the "Part" part of the "Whole" part.



Examples of Aggregation and Composition:

In the examples shown in the slide,

the relationship between a Sentence and a Word is represented as a “Composition” (Word is a part of a sentence).

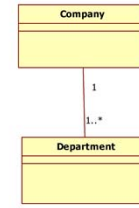
the relationship between Account and Transaction is represented as an “Aggregation”.

3.1: Class Diagrams

Definition of Multiplicity

- Multiplicity:
  - Multiplicity indicates the “number of instances” of one class linked to “one instance” of another class.

Symbol	Meaning
1	Exactly one
0..1	Zero or one
*	Many
0..*	Zero to many
1..*	One to many



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

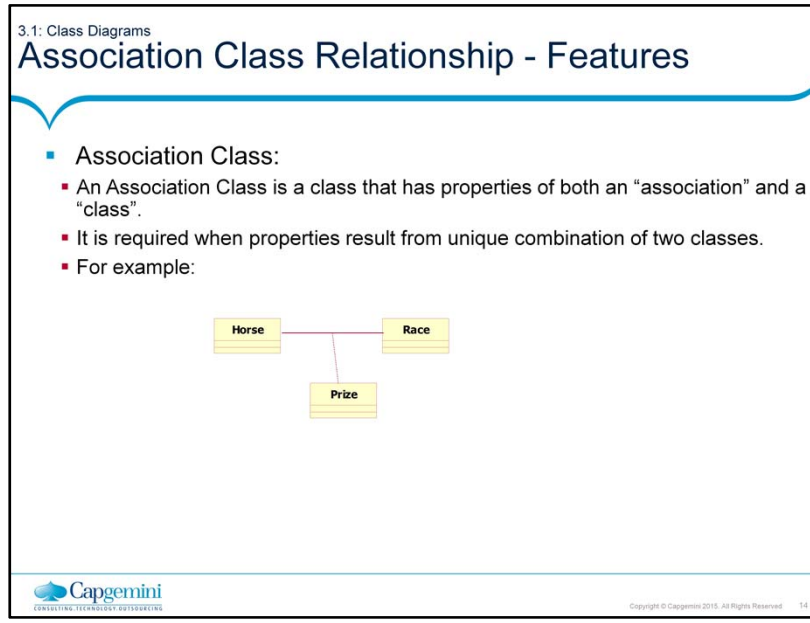
 13

Multiplicity:

Multiplicity attached to a class denotes the possible cardinalities of objects of the association.

For example: The above figure depicts that “One company has one or more departments, and a department is associated with one company”.

Multiplicity values can be indicated in association, aggregation, and composition relationships.



Association class:

An Association class is a class required as the result of association between two classes.

For example:

The Prize class is a result of association between the Horse class and the Race class.

For each Horse placed in a Race there is a prize.

The amount of prize depends on the race.

The Prize class could not be associated with the Horse class alone because a Horse might have many Prizes, and the relationship between the Prize and Race would be lost.

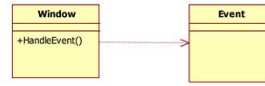
Similarly, Prize class cannot be associated with Race class alone because a Race has many Prizes, and the relationship between the Prize and the Horse would be lost.

Similarly, result of a student (in terms of marks in assignments, test, and grade) in a course is a unique combination of an individual student, and a particular course. So we can have an association between Student and Course Classes, with Result being an Association Class.

3.1: Class Diagrams

Dependency - Features

- Dependency:
  - Dependency is a “using” relationship within which the change in the specification of one class may affect another class that uses it.
  - For example:



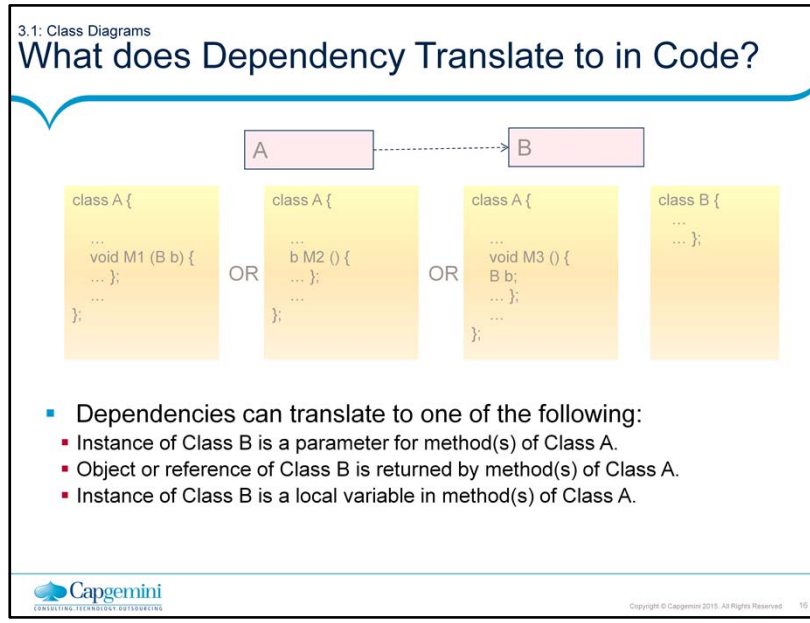
Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 15

Dependency:

The dependencies are denoted as dashed arrows with arrow head pointing to the independent element.

In the example shown in the slide, the structure and behavior of the Window Class is dependent on the structure and behavior of the Event Class.



UML Relationships: What does Dependency translate to in code?

In a dependency relationship, an instance of the independent class (B) will be used in the dependent class (A) in one of the following manners:

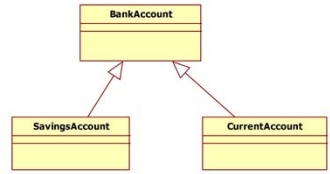
- Instance of B can a parameter to one or more methods of Class A.
- Instance of B can be returned by one or more methods of Class A.
- Instance of B can be a local variable in one or more methods of Class A.

Note that dependency is non-structural, that is, instance of Class B does not come as an attribute of Class A and hence not part of the structure of Class A. Class A is aware of existence of Class B only when the concerned method is called. The relationship is temporary in nature too…once the method invocation is completed, Class A need not maintain information about Class B.

3.1: Class Diagrams

Generalization - Features

- Generalization:
  - Generalization indicates relationships between super-class and sub-class.
  - For example:



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

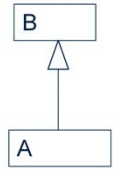
 17

Relationships: Generalization:

Generalizations are denoted as “paths” from specific elements to generic elements, with a hollow triangle pointing to the more general elements.

3.1: Class Diagrams

What does Generalization Translate to in Code?




```
class B {
  ...
  ...
... };
```

```
class A extends B {
  ...
  ...
... };
```

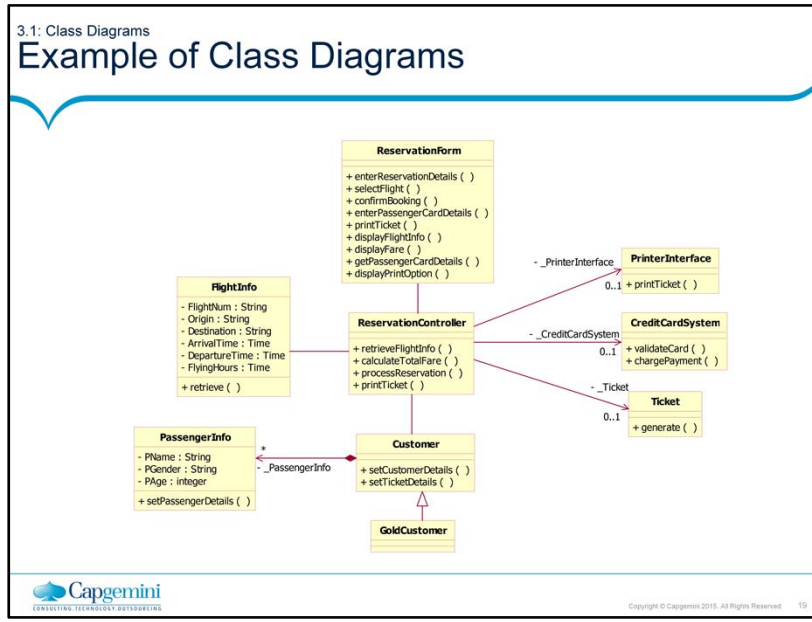
- Generalization entails using the language constructs to implement inheritance relationship.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

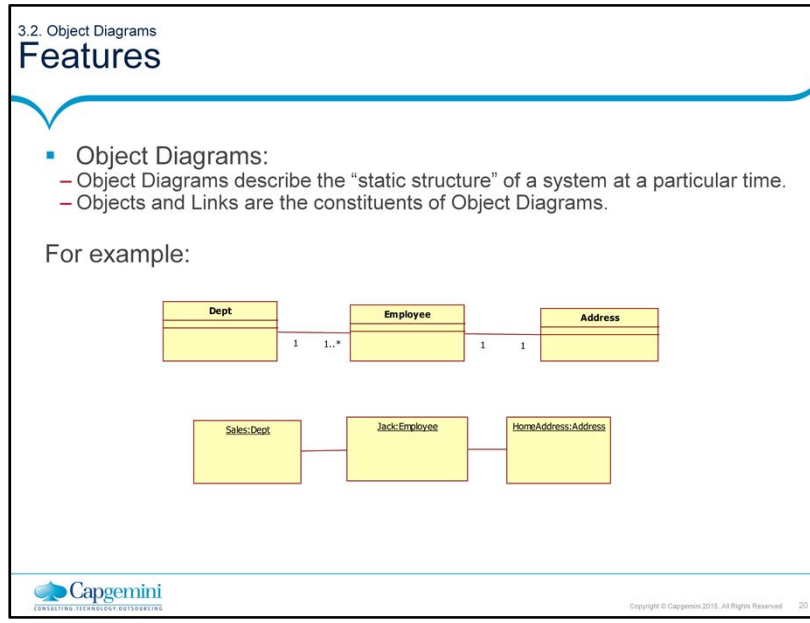
 18

UML Relationships: What does Generalization translate to in code?

Object oriented languages provide language constructs for implementing inheritance relationship. This will be used for coding generalization.



How do you interpret this diagram?



Object Diagrams:

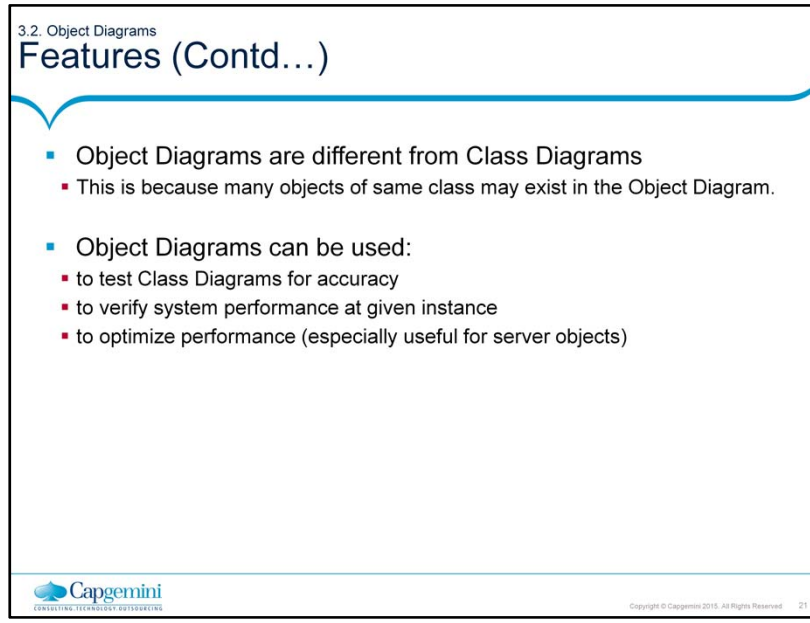
The Object Diagrams describe an instantiation of Class Diagram at a particular instance of time. They may be used to explore different configurations of objects. These configurations when combined can be generalized into relevant Class Diagrams.

Object Diagrams provide a snapshot of the instances in a system, and the relationships between the instances. It is a structural diagram that shows the instances of the classifiers in models.

In the example shown in the slide – Sales, Jack, HomeAddr are instances of the classes like department, person, and address respectively. Object Name may be omitted. That is to say, if there is only a colon followed by Class Name (with underlining) such an object is called an “anonymous object”.

Objects too may have other compartments as defined by their classes. However, usually attributes (along with values) are mentioned and operations are not mentioned.

contd.



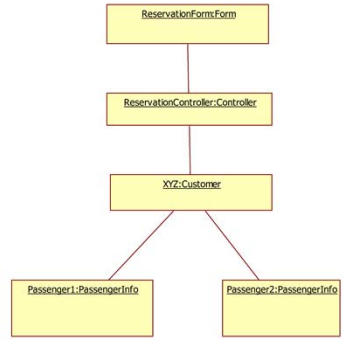
Object Diagrams (contd.):

Depending on the multiplicity values associated with class relationships, there may be different number of objects of the classes that exist in the Class Diagram. “Links” relate the objects. They are instances of relationships of Class Diagrams.

Since Object Diagrams indicate objects of system at particular instance of time, they can be used to verify system performance in terms of memory utilization of objects, and communication links between them. Thought can be given to minimize use of resources by techniques like pooling or queuing, batch processing, etc.

3.2: Object Diagrams

Example of Object Diagrams



Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 22

How would you interpret this?

Summary

- In this lesson, you have learnt:
  - Class Diagrams describe the static structure of a system
    - Class Diagrams show the existence of classes and relationship between them like Generalization, Association, Aggregation, Composition, or Dependency
  - Object Diagrams can be used to test the accuracy of Class Diagrams, and to optimize their performance, as well.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Review Question

- Question 1: A Class Diagram gives information about:
  - A. Attributed defined for a class
  - B. Operations defined for a class
  - C. Logic to be used for an operation of a class
- Question 2: An Object Diagram is unique for an application.
  - True / False
- Question 3: Relationships that you may find on a Class Diagram are ___, ___, ___, ___ and ___.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

Review Question: Match the Following

1. Object Diagram	A. Object and Links
2. Class Diagram	B. Classes and Relationships between then.

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 25

Unified Modeling Language

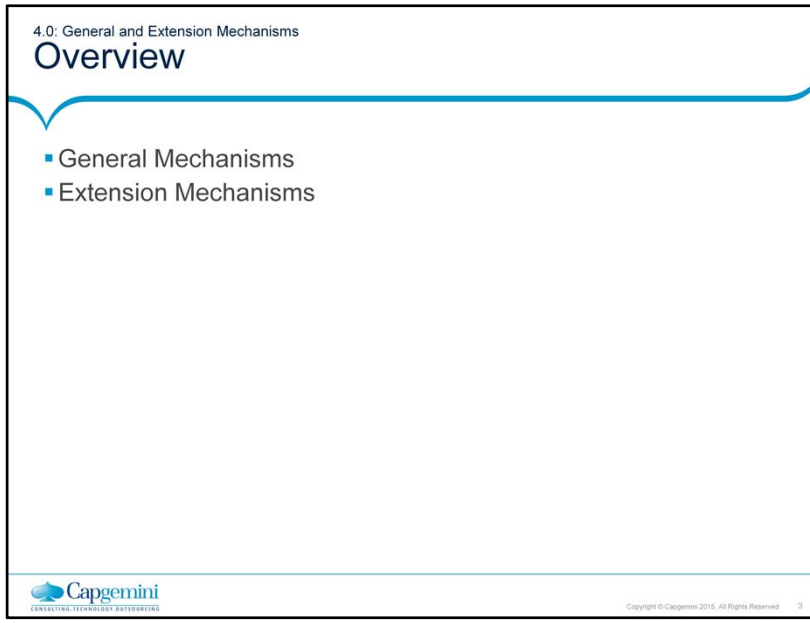
Lesson 04: General and Extension Mechanisms

Lesson Objectives

- To understand the following topics:
  - General mechanism
    - Adornment
    - Notes
  - Extension mechanism
    - Constraints
    - Tagged Values
    - Stereotypes

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



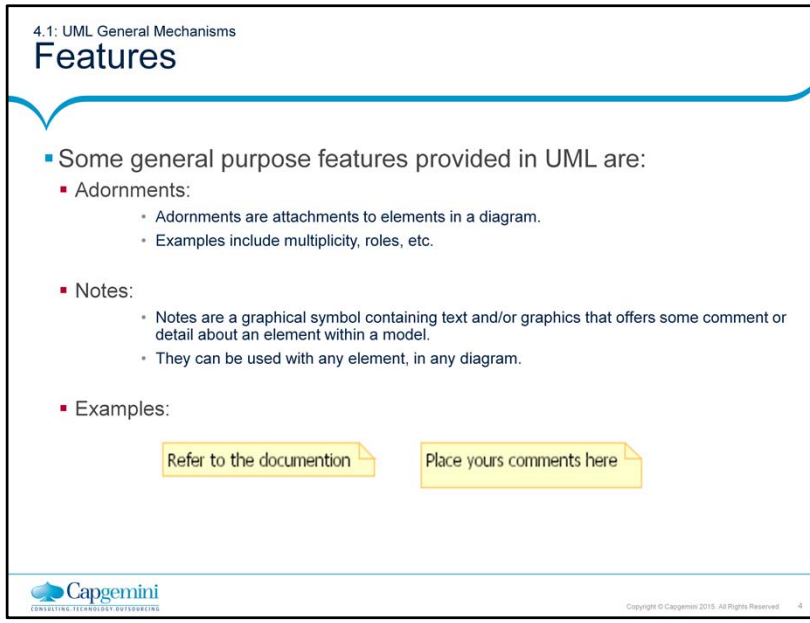
General and Extension Mechanisms:

We have completed our discussions on the various UML views, and their associated diagrams and modeling elements.

We will now focus on the General and Extension mechanisms available in UML.

The General Mechanisms help in giving further details in terms of comments (notes), etc.

The Extension mechanisms help in taking care of situations where existing UML notations may not suffice.

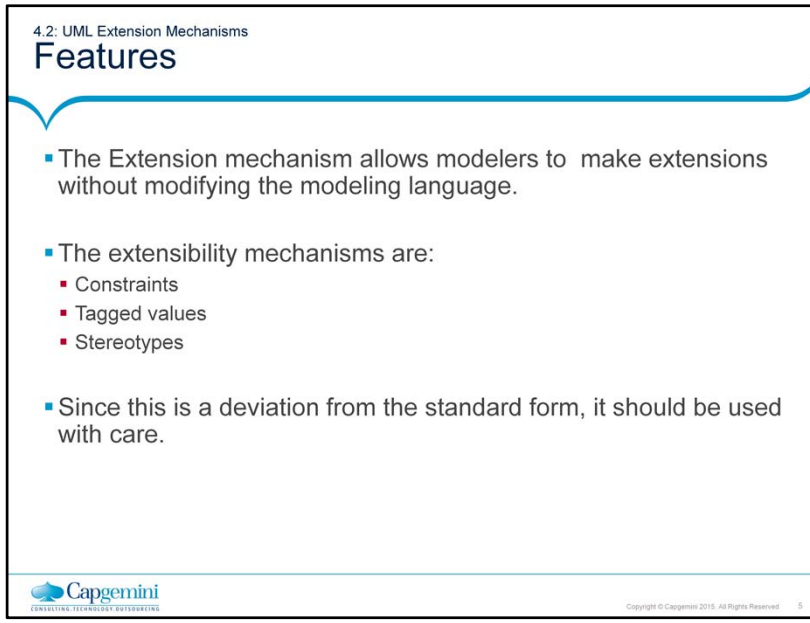


UML General Mechanisms:

General mechanisms include Adornments and Notes.

Adornments are attachments to elements in a diagram.

Notes are used to provide comments or additional documentation about a model element.

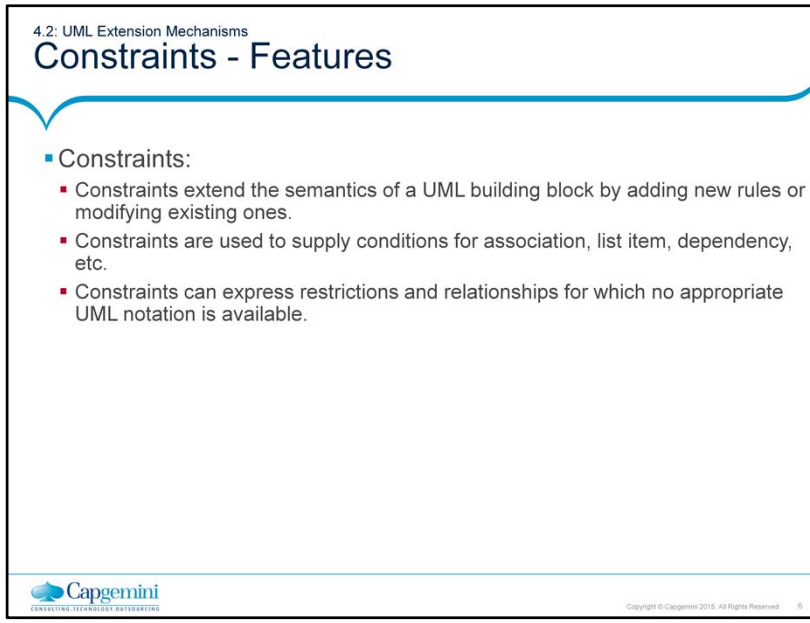


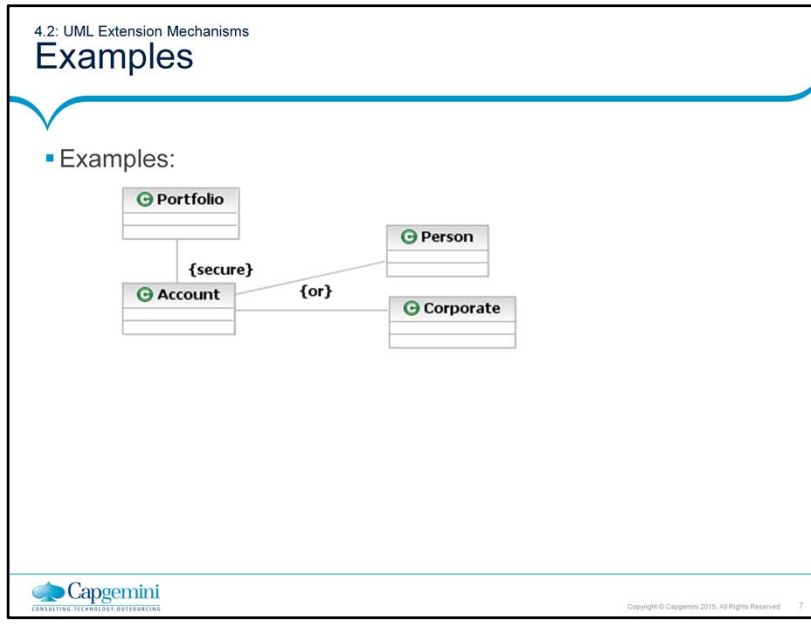
UML Extension Mechanisms:

UML, being a language, has a fixed set of notations with associated notation and semantics. However, we may need constructs that are not provided for in UML.

Instead of users of UML adding their own notations, UML provides for ways by which extensions can be made to the existing modeling language.

These Extension Mechanisms of constraints, tagged values, and stereotypes are meant for “customizing” and “extending” UML. However, they need to be used with care and should be well documented. After all they still are deviations from the standards.


4.2: UML Extension Mechanisms
Constraints - Features
Constraints:
Constraints extend the semantics of a UML building block by adding new rules or modifying existing ones.
Constraints are used to supply conditions for association, list item, dependency, etc.
Constraints can express restrictions and relationships for which no appropriate UML notation is available.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 6



UML Extension Mechanisms – Constraints:

Constraints are strings, which are enclosed in curly parenthesis, that are used to add on to the meaning of an existing notation. They are typically used for indicating conditions or restrictions.

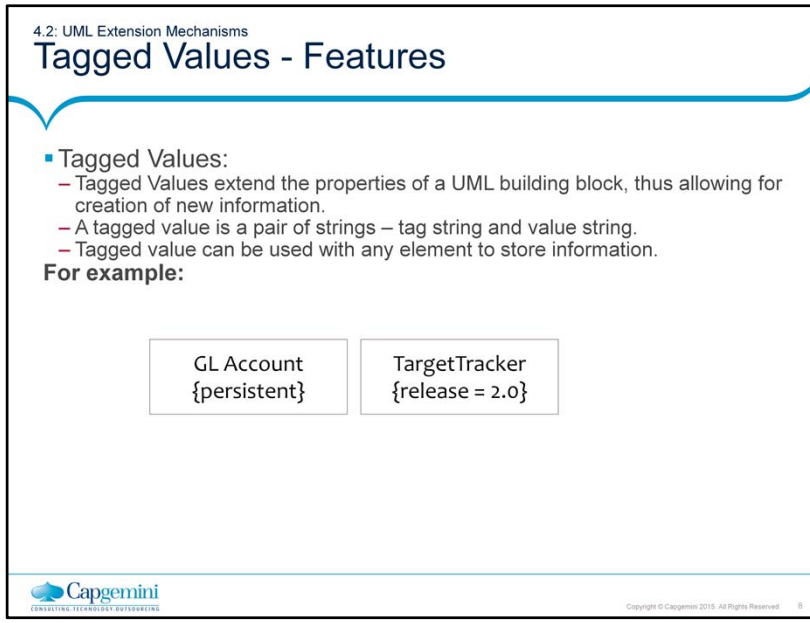
For example:

The relationship between classes Bank Account and Portfolio is shown as a “secure relationship”. The constraint {secure} is added on to the association relationship thereby extending the meaning.

Similarly, the constraint {or} in the example denotes that the bank account class may be related to Person, or the Corporation class.

There are some standard constraints, as well.

For example: abstract, global, concurrent, etc.



UML Extension Mechanisms – Tagged Values:

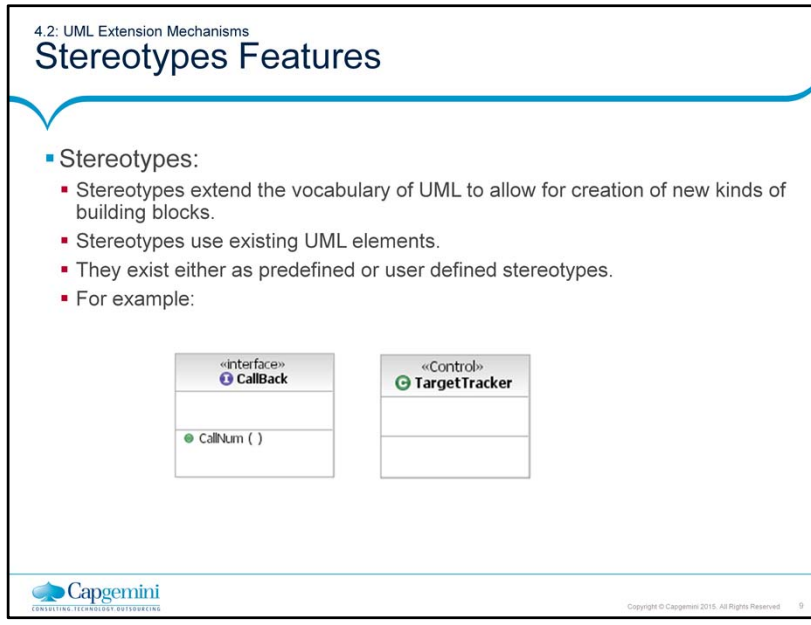
Properties of model elements can be extended by using tagged values. A “keyword” indicates the “tag”, and “values” represent “value of tag”.

In case the “value” is omitted, the property is assumed to be Boolean with value as TRUE.

For example:

The persistent tag, which is a pre-defined tag, indicates nature of persistence of the class GL account.

The release tag of the TargetTracker class indicates the version of release of this class.



UML Extension Mechanisms – Stereotypes:

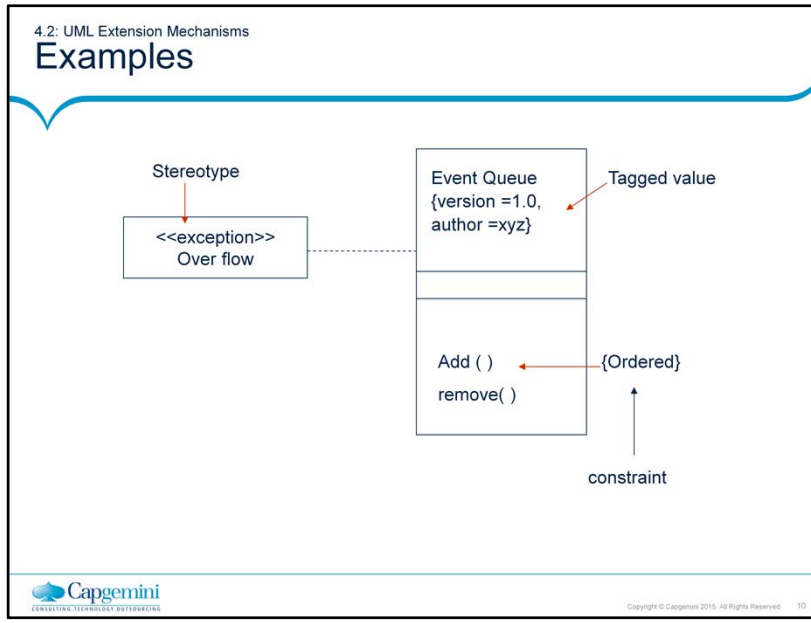
Stereotypes are strings enclosed in guillemots (<< >>). They are mechanisms by which model elements can be “marked” or “classified” to introduce new modeling element.

For example: The interface stereotype may be used to indicate that class may contain only externally accessible methods.

The stereotype control may indicate nature of Target Tracker class as the Control Class in the system.

There are many standard stereotypes.

For example: ‘becomes’ indicates that source and target represent same instances at different points in times, may be with different roles or states.



Examples of UML Extension Mechanisms:

In the Class Diagram shown in the slide, the stereotype “exception” indicates that “Overflow” is a class meant for handling exceptions that occur in the “Event Queue” class.

The tagged values for Event Queue give information about the version and author of the Event Queue.

The constraint “Ordered” associated with the method “Add()” of “Event Queue” denotes that elements are added to the “Event Queue” based on some ordering.

Summary

- In this lesson, you have learnt:
  - General mechanisms include:
    - Adornments: to indicate roles and multiplicity.
    - Notes: that will have some comments.
  - Extensibility mechanisms include:
    - Constraints: Extend the semantics of a UML building block by adding new rules.
    - Tagged values: Allow creation of new information.
    - Stereotypes: Provide mechanisms by which model elements can be marked or classified to introduce new modeling element.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Review Question

- Question 1: ___ is a UML general mechanism used to give additional information about a model element.
- Question 2: The UML extension mechanism ___ can be used to extend the language by adding new notations.

UML

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
18-Dec-2008	0.1D	Veena Deshpande	Creation
22-Dec-2008		CLS team	Review
Jan-2009	1.0	Veena Deshpande	Baselined
8 May 2009	1.1	Veena Deshpande	Updated for including comments related to tool usage for Labs.
20-May-11	1.2	Latha S	Added some more UML exercises , Added STARUML points needed to work on the UML exercises
04-Apr-15	1.3	Kavita Arora	Made changes as per revised TOC

Table of Contents

Document Revision History 2
Table of Contents 3
Getting Started 4
Overview 4
Setup Checklist for UML 4
Instructions 4
Learning More (Bibliography) 4
UML Problem Statement/ Case Study 5
Overview – Banking System 5
Lab 1. Dynamic View Diagrams – Use Case Diagrams 6
1.1: Interpret Use Case Diagram 6
1.2: Create Use Case Diagram 7
Lab 2. Dynamic View Diagrams – Activity Diagrams 8
2.1: Interpret Activity Diagram 8
2.2: Create Activity Diagram 9
Lab 3. Dynamic View Diagrams – Sequence Diagrams 10
3.1: Interpret Sequence Diagram 10
3.2: Create Sequence Diagram 11
Lab 4. Some More Dynamic View Diagrams – State Chart Diagrams 12
4.1: Interpret State Chart Diagram 12
4.2: Create State Chart Diagram 13
Lab 5. Static View Diagrams – Class Diagrams 14
5.1: Interpret Class Diagram 14
5.2: Create Class Diagram 15
Lab 6. General and Extension Mechanisms 16
6.1: Identify approaches to extend UML 16
Appendices 17
Appendix A: Table of Figures 17

Getting Started

Overview

This lab book comprises of assignments to be done for Unified Modeling Language (UML). The course is focused towards understanding Object Oriented Concepts and understanding UML diagrams with some exposure to creating diagrams which are closely related to developer roles – class diagrams, sequence diagrams, activity diagrams.

It is recommended to get familiar with STARUML – an open source modeling tool. Instructor would give a demo of the tool. Participants would be expected to explore the tool and model the lab assignment diagrams using the tool.

Setup Checklist for UML

- STARUML to be available on all machines.

Instructions

- Specified for each of the individual assignments.

Learning More (Bibliography)

- Applying UML – Advanced Applications by Rob Pooley , Pauline Witcox
- UML User's Guide by Grady Booch, Ivar Jacobson and James Rambaugh

UML Problem Statement/ Case Study

Overview – Banking System

A well-known bank desires to build a process of account maintenance of savings and current accounts. The savings and the current account are however maintained in different sources (databases). The system should have the online capabilities to cater to requirements viz.

- View Account Balance & details (Both Current & Savings.
- View Account details (should have only last 10 transactions)
- View Statements for the account (month-wise)
- Request for Address Changes
- Transfer requests from Savings to current account (User should be able to transfer funds amongst the current and savings account

The default view for all users is View Account Balance.

Lab 1. Dynamic View Diagrams – Use Case Diagrams

Goals	• Interpret Use Case Diagram • Create Use Case Diagram
Time	30 minutes

1.1: Interpret Use Case Diagram

Study the Use Case Diagram and answer the questions given below.

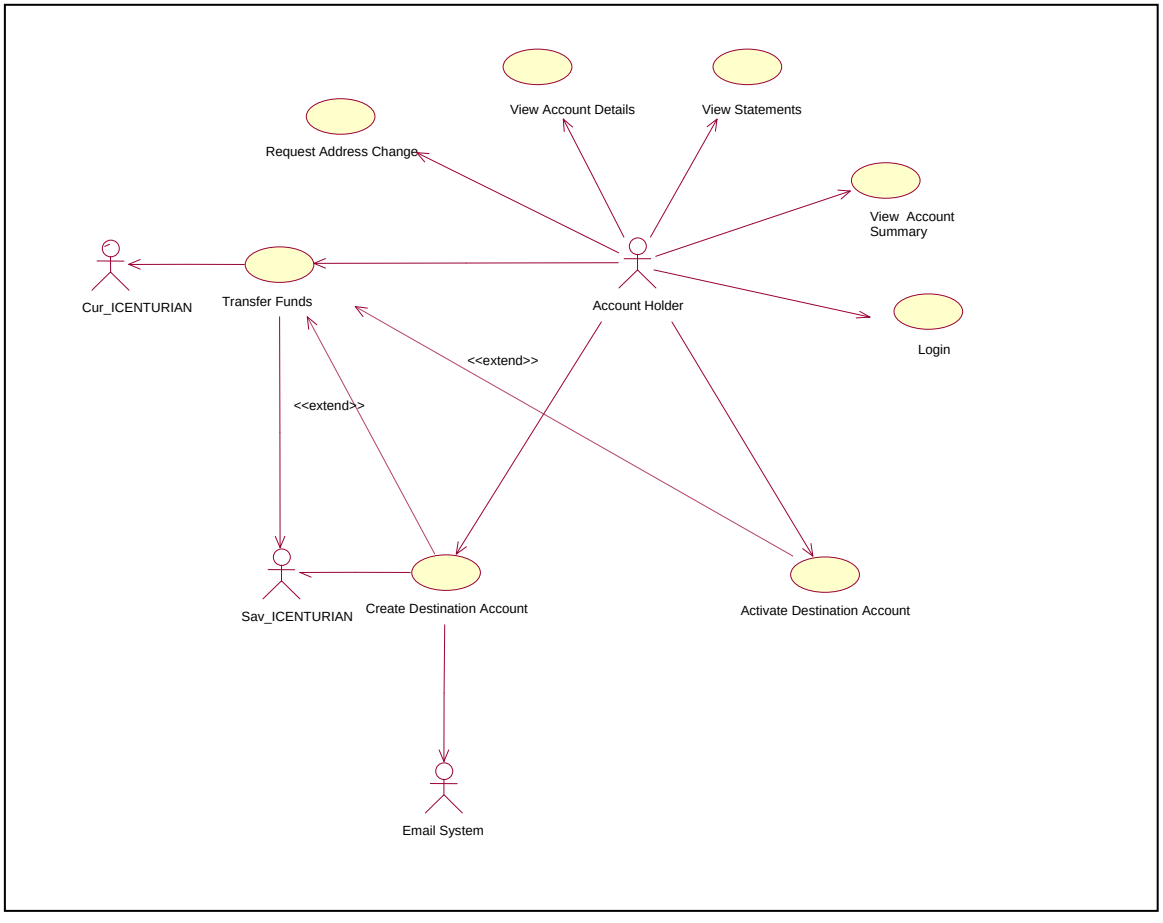


Figure 1: Use Case Diagram

Questions:

1. Name the actors.
2. Why is the Email System considered as an actor for this system?
3. How would the above diagram be different if the data sources were designed as part of the application?
4. Name the use cases that can be invoked by Account Holder.
5. Which use case has extension points? What are the extended use cases?
6. Does this online system allow the use to pay insurance premiums? If yes, which is the use case? If no, how can the diagram be changed to include that?

1.2: Create Use Case Diagram

[This assignment is to be done in groups of 2 to 4 participants.]

A. Draw the Use Case diagram for the following scenario. Discuss your diagram with other groups.

Visitors can browse the catalog of an online book store and check for availability for a book.
Registered customers can browse, search and place orders. S/He can also cancel the placed order. Placed orders are fulfilled only after the credit card details entered by the customer are validated by a third party payment gateway. The shipping details of the fulfilled orders are sent to Shipping Agent. The Sales report is sent to the CEO every month.

B. Consider any other online application of your choice. For this application, identify actors and use cases. Draw the use case diagram. Discuss your diagram with other groups.

Hint: Consider online applications such as Hotel Reservations, CD Rental Library and so on.

Lab 2. Dynamic View Diagrams – Activity Diagrams

Goals	• Interpret Activity Diagram • Create Activity Diagram
Time	30 minutes

2.1: Interpret Activity Diagram

Study the activity diagram and answer the questions given below.

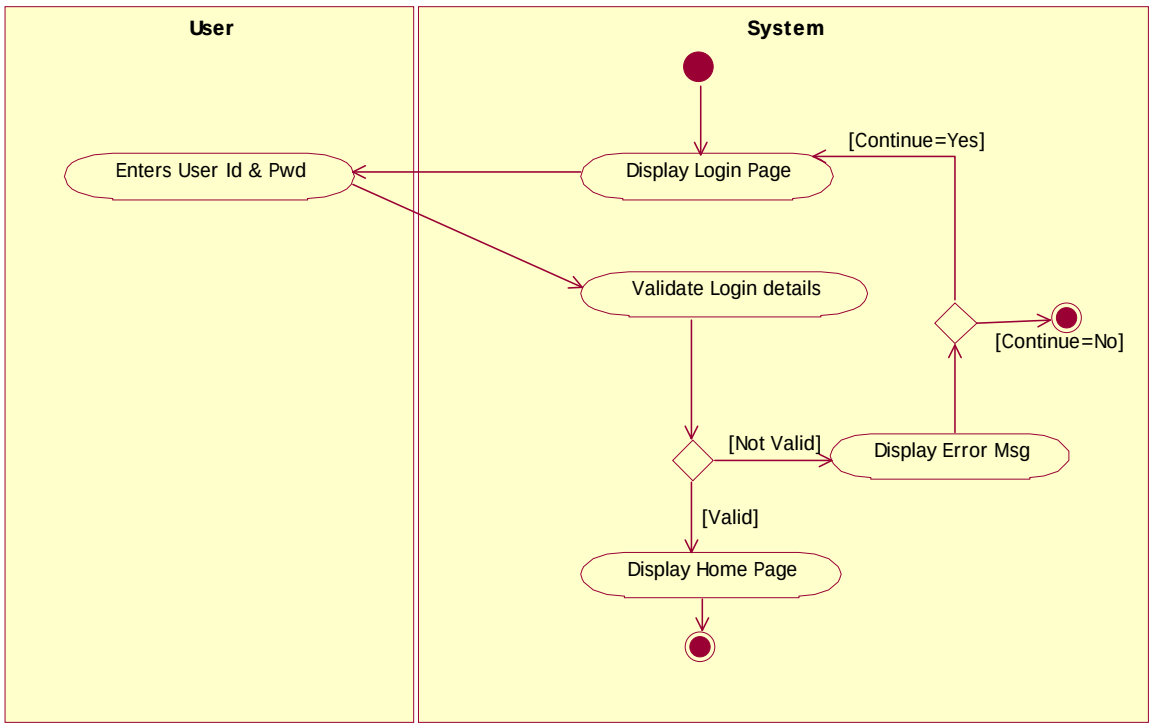


Figure 2: Activity Diagram

Questions:

7. Refer 1.1. Which use case do you think this activity diagram is describing?
8. Name the swimlanes that you find in this diagram.
9. Consider the following change to the use case: Only 3 attempts are permitted to enter valid user id and password. Make appropriate changes to this diagram.

2.2: Create Activity Diagram

[This assignment is to be done in groups of 2 to 4 participants.]

A. For the system considered in 1.2 A, draw an activity diagram for the purchase use case. Discuss your diagram with other groups. You can use the following for guidelines for the workflow associated with this use case.

   Registered customer can browse and select items, add it to shopping cart, and when he has finished shopping, proceed to billing.

B. For the system considered in 1.2 B, draw an activity diagram for a use case of your choice. Discuss your diagram with other groups.

Lab 3. Dynamic View Diagrams – Sequence Diagrams

Goals	Interpret Sequence Diagram
Time	10 minutes

3.1: Interpret Sequence Diagram

A. Study the sequence diagram. This illustrates retrieval of account details of a customer from an external data source. Answer the questions given below.

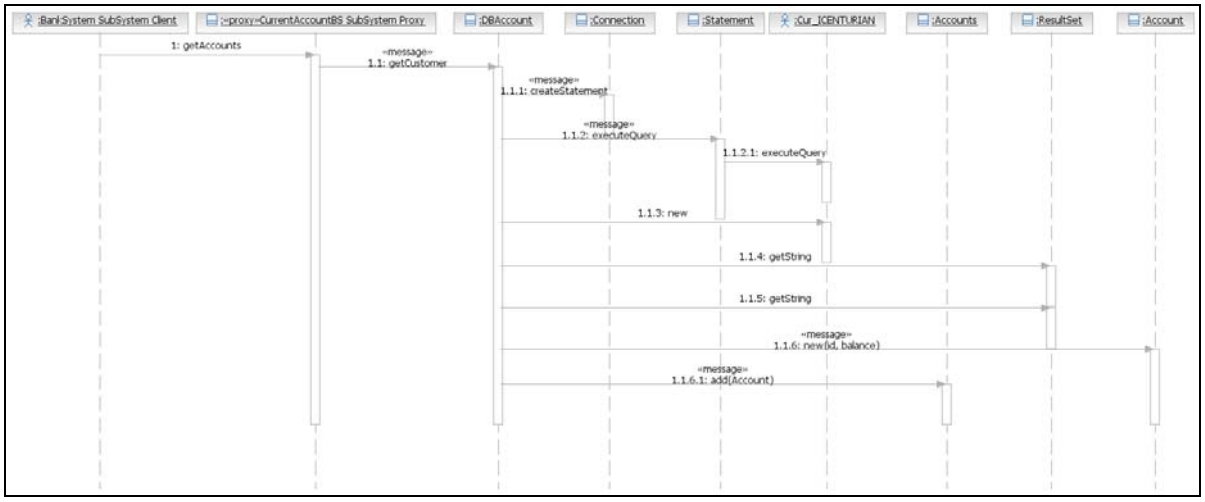


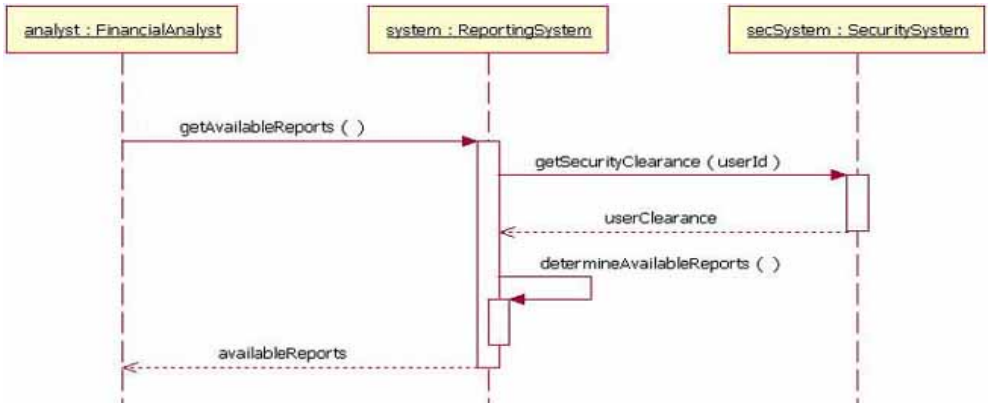
Figure 3: Sequence Diagram

Questions:

10. Name any 3 objects that are part of this diagram.
11. Which is the external data source from where the data is being retrieved?
12. If a class called “Statement” is coded considering this sequence diagram, what operation must be part of the “Statement” class?
13. Does this sequence diagram model error conditions? If yes, what are they? If no, what could be the error conditions and how would you model them?

B. Study the sequence diagram given below and answer the following questions.

1. The operation getAvailableReports() belongs to the class FinancialAnalyst. TRUE /FALSE
2. What methods would be implemented by the class ReportingSystem ?
3. What is the return value of the method getAvailableReports ?
4. Who is responsible for calling the operation determineAvailableReports() ?



3.2: Create Sequence Diagram

Draw the sequence diagram for login module for Registered customers:

a. User enters id and password on the Form and clicks the submit button.
b. The OnSubmit() method is invoked.
c. The OnSubmit() method instantiates the LoginBO object. It calls the IsValidLogin Method and passes the id and password.
d. Two methods, IsValidID() and IsValidPWD() of LoginBO object, are executed within the LoginBO object.
e. If one of the methods returns false, then false is returned by IsValidLogin()
f. If both methods are returning true, then LoginBO instantiates the LoginDAO object. It calls the ValidateLogin() method of the DAO and passes the id and password.
g. The LoginDAO’s checks with the database and returns true /false accordingly.
h. The return value of DAO is returned by the BO object to the presentation layer i.e. to the Form.

Additional Exercise: Add the feature that the user is allowed to try the above functionality 5 times.

Lab 4. Some More Dynamic View Diagrams – State Chart Diagrams

Goals	• Interpret State Chart Diagram • Create State Chart Diagram
Time	20 minutes

4.1: Interpret State Chart Diagram

Study the state chart diagram. This illustrates the various states in which an "Account" object can be in. Answer the questions given below.

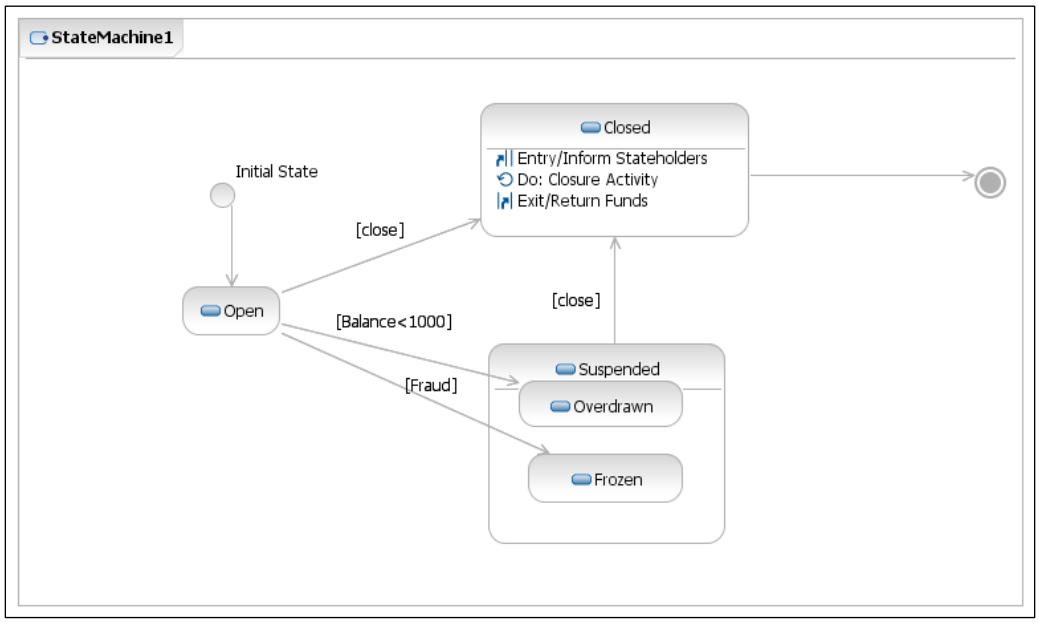


Figure 4: State Chart Diagram

Questions:

14. What are the start and end states indicated on this diagram.
15. Are there any states depicting “State-Sub state” relationships? Which are these states?
16. Which states depicts entry action, exit action and internal activity? What are these actions/activities?
17. Under what conditions does the Account go into Suspended state?

4.2: Create State Chart Diagram

Draw a state chart diagram which illustrates the states in which text style can be in a MS-word document. Indicate appropriate transitions.

Hint: Consider that the font style and size of the text remains same. But the text can be in normal style, bold, italics and so on. Are combinations also possible? For example: Bold and Underlined?

Lab 5. Static View Diagrams – Class Diagrams

Goals	• Interpret Class Diagram • Create Class Diagram
Time	20 minutes

5.1: Interpret Class Diagram

Study the class diagram and answer the questions given below.

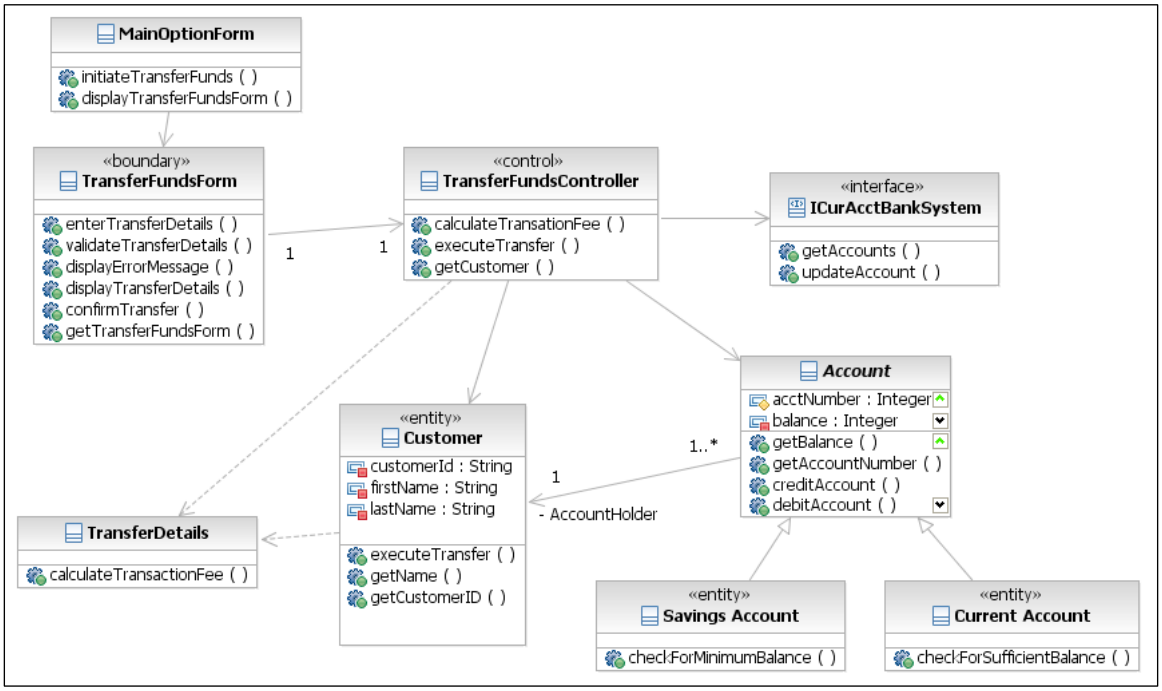


Figure 5: Class Diagram

Questions:

18. Name all the classes that are part of this diagram.
19. Name the attributes defined for the “Customer” class.
20. Name the operation defined for the “TransferFundsController” class.
21. The following relationships are depicted amongst which classes?
    i. Generalization
    ii. Dependency
    iii. Association
22. Can a customer hold multiple accounts in the bank? Are joint accounts permitted as per this diagram?

23. What needs to change in the above diagram if we wanted to depict the relationship that "Customer *has* Accounts"?

5.2: Create Class Diagram

1. Use the following information to create a class diagram.
    a. A customer class is related to an order class with an association relationship. One customer can be associated with multiple orders, but an order belongs to exactly one customer.
    b. Each order is composed of multiple products. Depict a composition relationship between Order class and Product class.
    c. A product could be assembled using several product parts. Depict a composition relationship between the Product class and Product Part class.
    d. There are "Gold Customers" i.e. Special customers who get better deals on orders. Illustrate class called "Gold Customer" as a special type of customer class.

2. Draw the class diagram for the following using STARUML, referring to the Question 3.2. The following guidelines are given

- LoginBO

	Attributes	**Methods**
Public		Default constructor
		IsValidLogin takes 2 string as arguments and returns boolean
Private		GetUserName takes 2 string and returns String
		IsValidUser takes string as argument and returns boolean
		IsValidPWD takes string as argument and returns boolean

- LoginDAO

	Attributes	**Methods**
Public		Constructor which takes string as an argument
		IsValidLogin takes 2 string as arguments and returns boolean
Private	Connection object (user defined)	GetConnection takes no arguments and returns a connection reference

- LoginBO has a dependency relation with LoginDAO.

Lab 6. General and Extension Mechanisms

Goals	Identify Extension Mechanisms
Time	5 minutes

6.1: Identify approaches to extend UML

Revisit the diagrams given in the previous labs to answer the questions given below.

24. Name the extension mechanisms you find in the Use Case Diagram and Deployment Diagram.
25. Name some stereotypes that you find on the Class Diagram.
26. Name some adornments that you find on the Class Diagram.
27. Do you find any Notes being depicted on any of the diagrams? If yes, in which diagrams?

Appendices

Appendix A: Table of Figures

Figure 1: Use Case Diagram 6
Figure 2: Activity Diagram 8
Figure 3: Sequence Diagram 10
Figure 4: State Chart Diagram 12
Figure 5: Class Diagram 14

