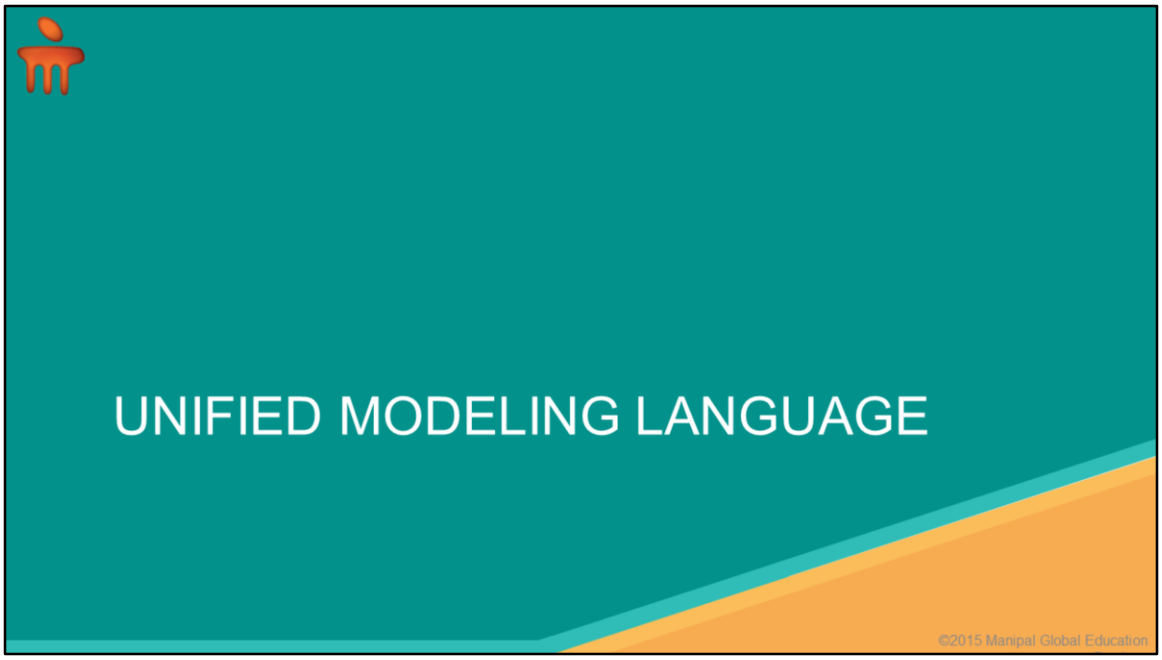
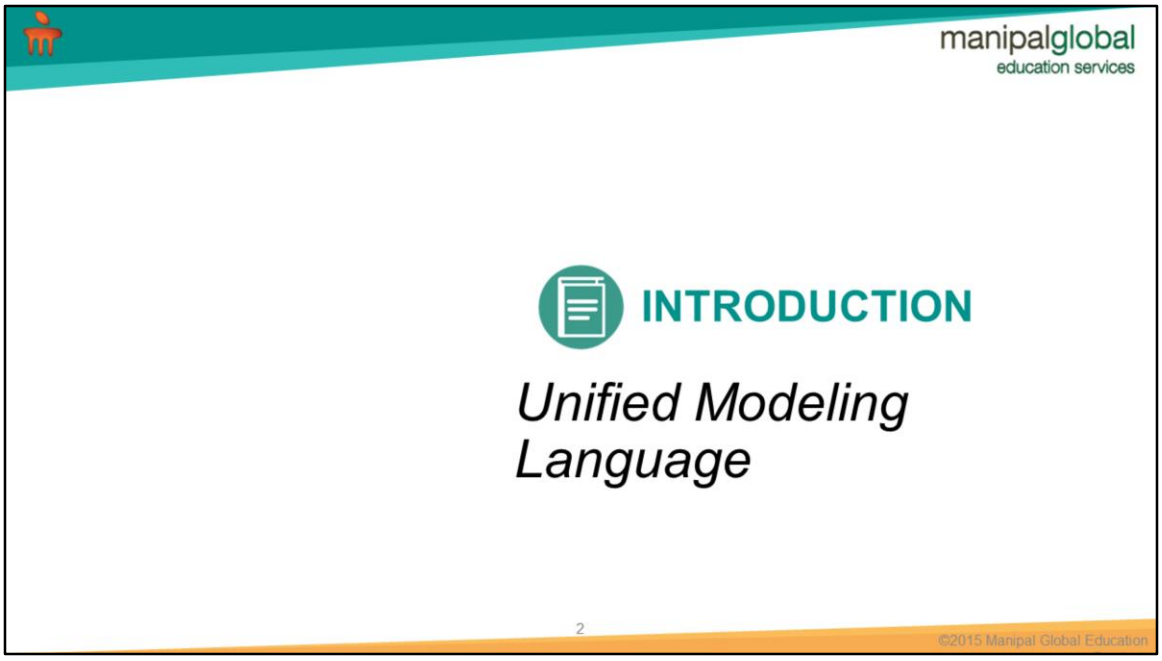

UNIFIED MODELING LANGUAGE
©2015 Manipal Global Education


manipalglobal
education services
INTRODUCTION
Unified Modeling Language
2
©2015 Manipal Global Education


manipalglobal
education services
OBJECTIVES
Unified Modeling Language
3
©2015 Manipal Global Education

LEARNING OBJECTIVES

At the end of this lesson, you will be able to:

- Understand System Modelling using Unified Modelling Language(UML)
- Describe UML and its History
- Distinguish between static and dynamic UML diagrams
- Apply UML diagrammatic techniques
- Create Use Case diagram and Class diagrams
- Understand the representation and use of the UML diagrams

manipalglobal
education services

MODEL

- What is a Model?
  - A model is an abstract representation of a system constructed from a perspective to understand the system prior to building or modifying it
  - A static (or structural) model can be viewed as a "snapshot" of a system's parameters at a specific point in time
  - A dynamic (or behavioral) model is a collection of procedures or behaviors that, taken together, reflect the behavior of a system over time

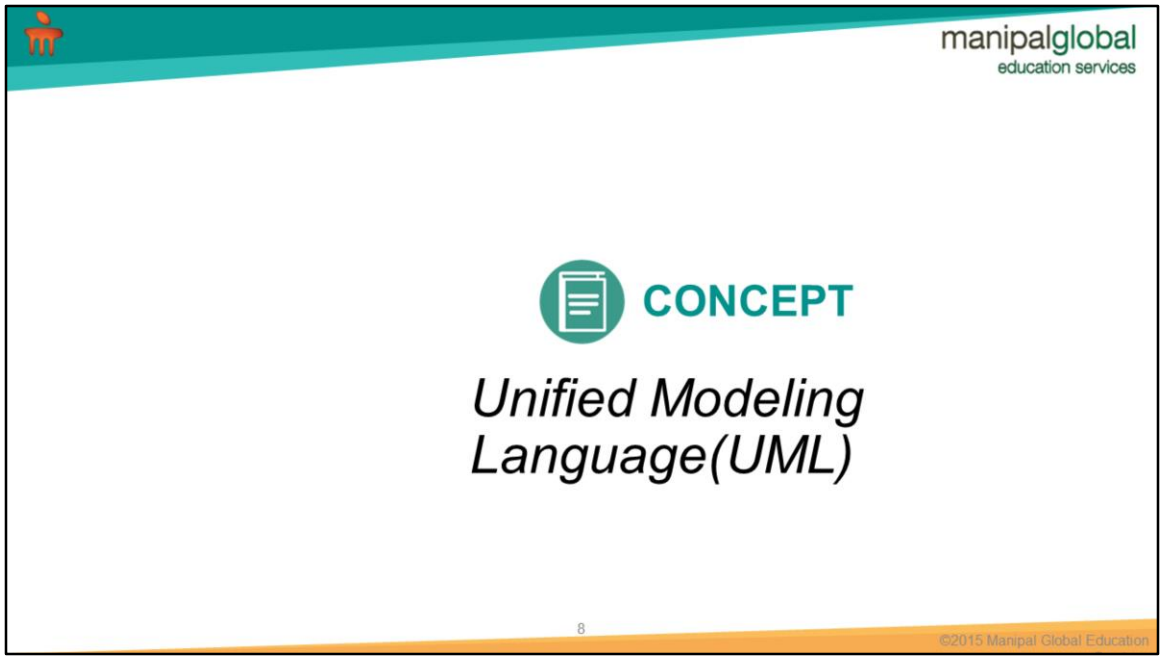
manipalglobal
education services

MODEL

manipalglobal
education services

SYSTEM MODELING

- Models of complex systems are built as it is difficult to comprehend such a system in its entirety
  - "Divide-and-conquer" approach
- Models make it easier to express complex ideas
  - For example, an architect builds a model to communicate ideas more easily to clients
- The cost of the modelling analysis is much lower than the cost of similar experimentation conducted with a real system
- Models enhance learning and better understanding of the real system
- Model has scope for modification


manipalglobal
education services
CONCEPT
Unified Modeling Language(UML)
8
©2015 Manipal Global Education

INTRODUCTION TO UML

Standardized general-purpose modelling language in the oriented software

Modelling language for visualizing, specifying, documenting a software system

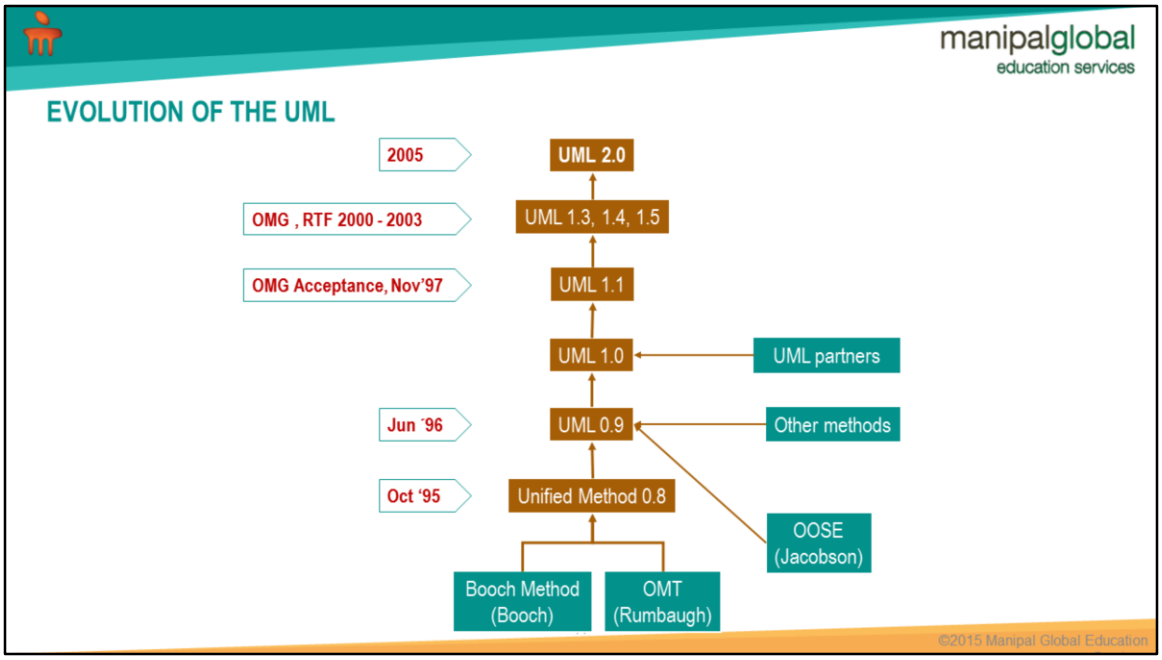
and its components

- UML
  - includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.
  - Based on work from Booch, Rumbaugh, Jacobson at Rational Software in 1990's
  - Independent of Implementation Language and can be directly connected to a

manipalglobal
education services

THE VALUE OF UML

- Is an open standard
- graphical notation for specifying, visualizing, constructing, and documenting software systems
- Supports the entire software development lifecycle
- Supports diverse applications areas
- Increases understanding/communication of system to Clients and Developers
- Is based on experience and needs of the user community
- Supported by many tools like Rational Rose, Rational XDE, Borland Together, Rational Software Architect , Power Designer

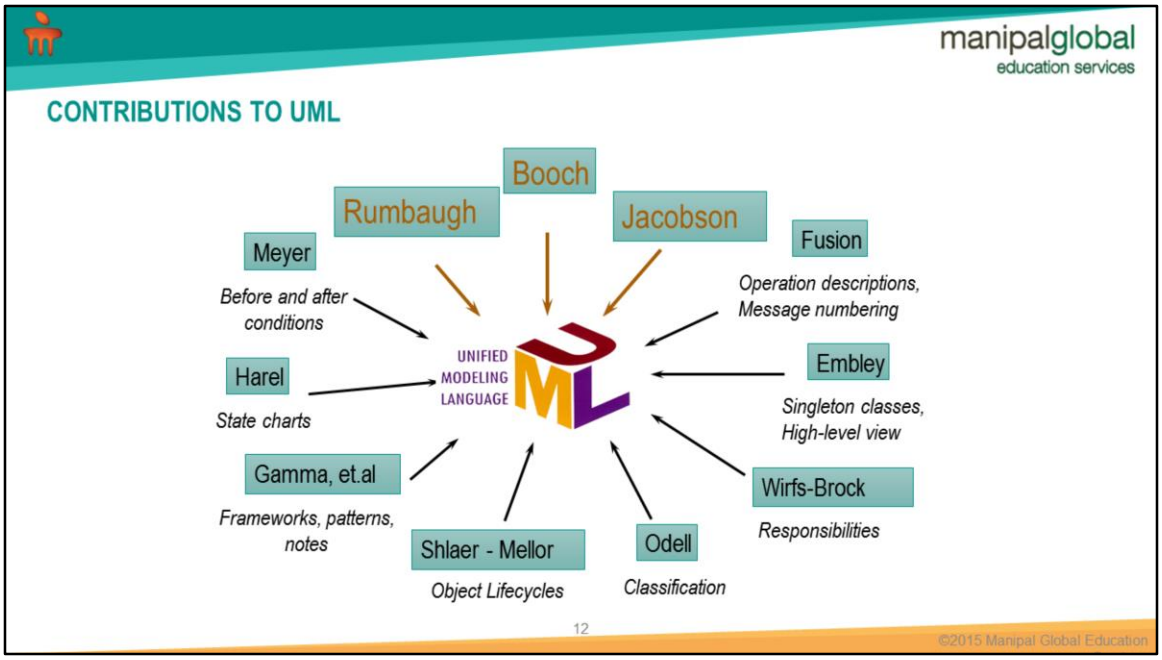


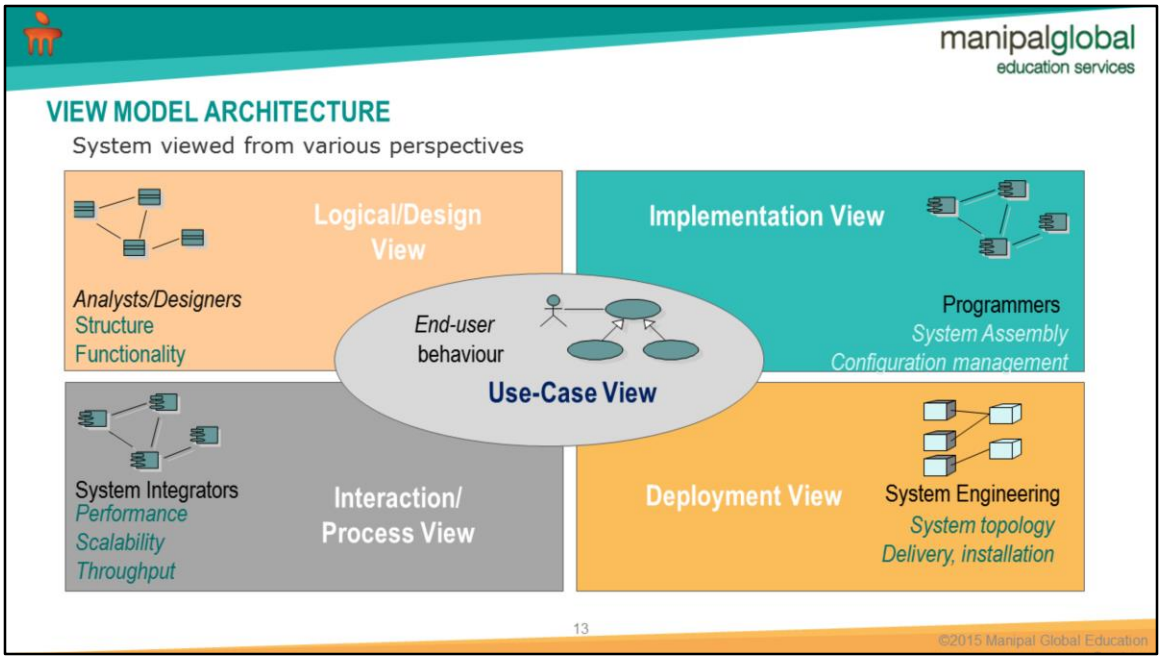
At the beginning of the 1990s, the object-oriented methods of Grady Booch and James Rumbaugh were widely used. In October 1994, the Rational Software Corporation (part of IBM since February 2003) began the creation of a unified modeling language. First, they agreed upon a standardization of notation (language), since this seemed less elaborate than the standardization of methods. In doing so, they integrated the *Booch Method* of Grady Booch, the *Object Modeling Technique* (OMT) by James Rumbaugh, and *Object-Oriented Software Engineering* (OOSE), by Ivar Jacobsen, with elements of other methods and published this new notation under the name UML, version 0.9. The goal was not to formulate a completely new notation, but to adapt, to expand, and to simplify the existing and accepted types of diagrams of several object-oriented methods, such as class diagrams, Jacobson's Use Case Diagrams, or Harel's Statechart Diagrams. The means of representation that were used in structured methods were applied to UML. Thus, UML's activity diagrams are, for example, influenced by the make-up of data flow charts and Petri nets.

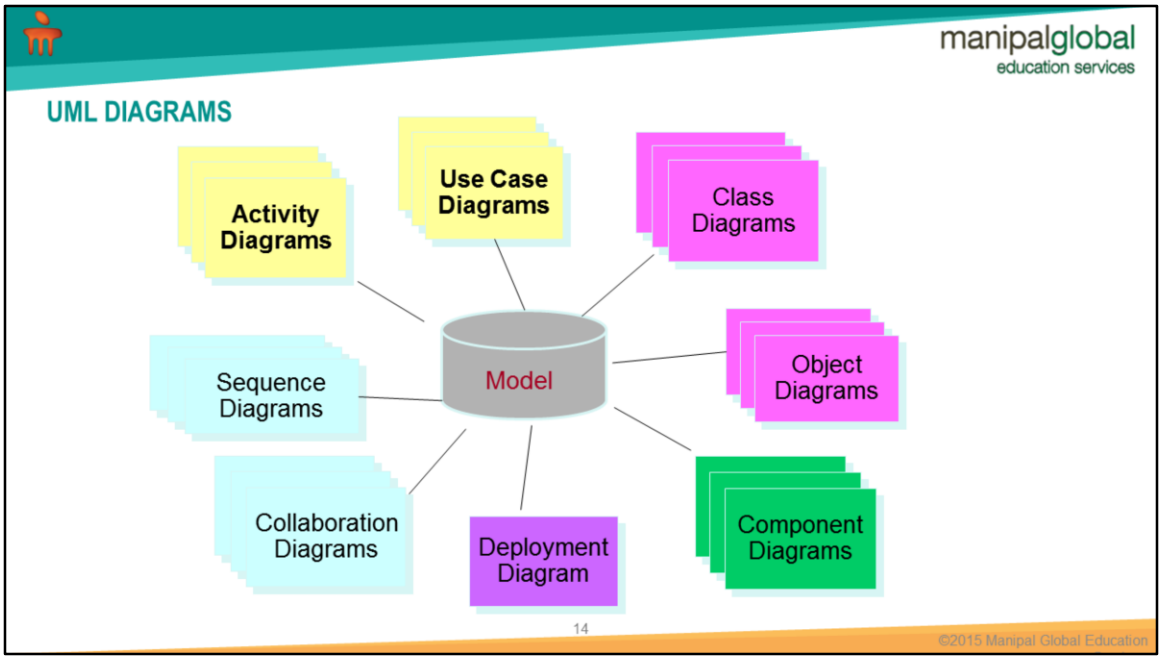
What is outstanding and new in UML is not its content, but its standardization to a single unified language with formally defined meaning.

Well-known companies, such as IBM, Oracle, Microsoft, Digital, Hewlett-Packard, and Unisys were included in the further development of UML. In 1997, UML version 1.1 was submitted to and approved by the OMG. UML version 1.2, with editorial adaptations, was released in 1998, followed by version 1.3 a year later, and UML 1.5 in March, 2003. Developers had already been working on version 2.0 of UMLsince the year 2000, and it was approved as a

Final Adopted Specification by OMG in June, 2003.


manipalglobal
education services
CONTRIBUTIONS TO UML
Booch
Rumbaugh
Jacobson
Meyer
Fusion
Before and after conditions
Operation descriptions, Message numbering
UNIFIED MODELING LANGUAGE
Harel
Embley
State charts
Singleton classes, High-level view
Gamma, et.al
Wirfs-Brock
Frameworks, patterns, notes
Responsibilities
Shlaer - Mellor
Odell
Object Lifecycles
Classification
12
©2015 Manipal Global Education


manipalglobal
education services
VIEW MODEL ARCHITECTURE
System viewed from various perspectives
Logical/Design View
Analysts/Designers
Structure
Functionality
Implementation View
Programmers
System Assembly
Configuration management
End-user behaviour
Use-Case View
System Integrators
Performance
Scalability
Throughput
Interaction/ Process View
Deployment View
System Engineering
System topology
Delivery, installation
13
©2015 Manipal Global Education


manipalglobal
education services
UML DIAGRAMS
Activity Diagrams
Use Case Diagrams
Class Diagrams
Sequence Diagrams
Model
Object Diagrams
Collaboration Diagrams
Deployment Diagram
Component Diagrams
14
©2015 Manipal Global Education

manipalglobal
education services

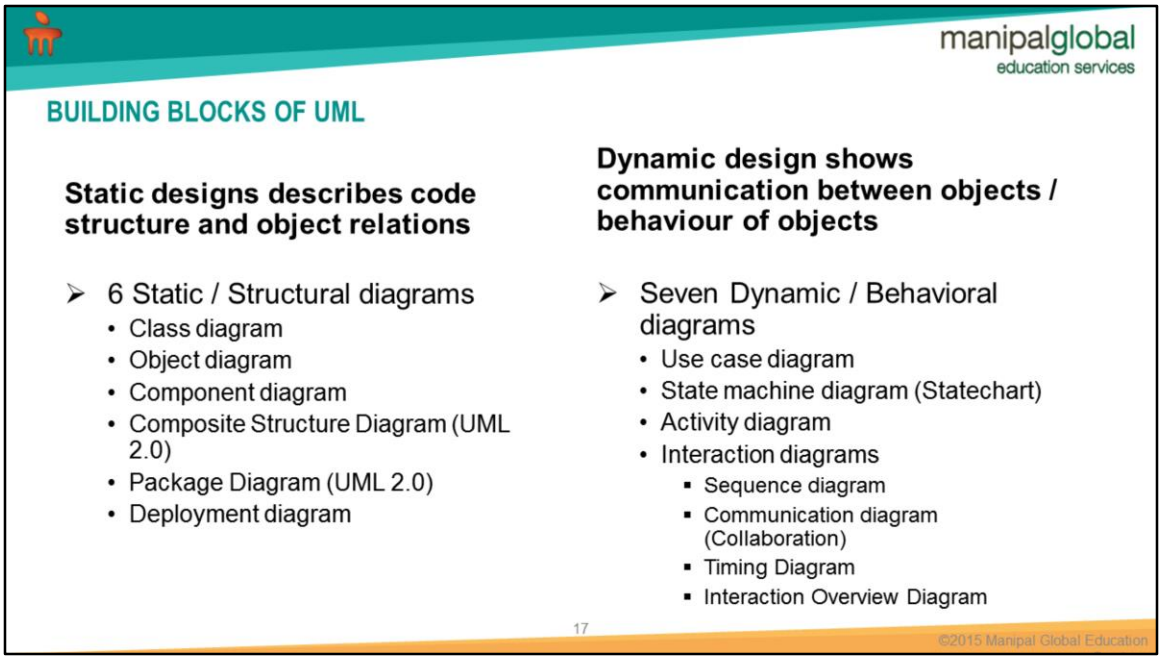
BUILDING BLOCKS OF UML

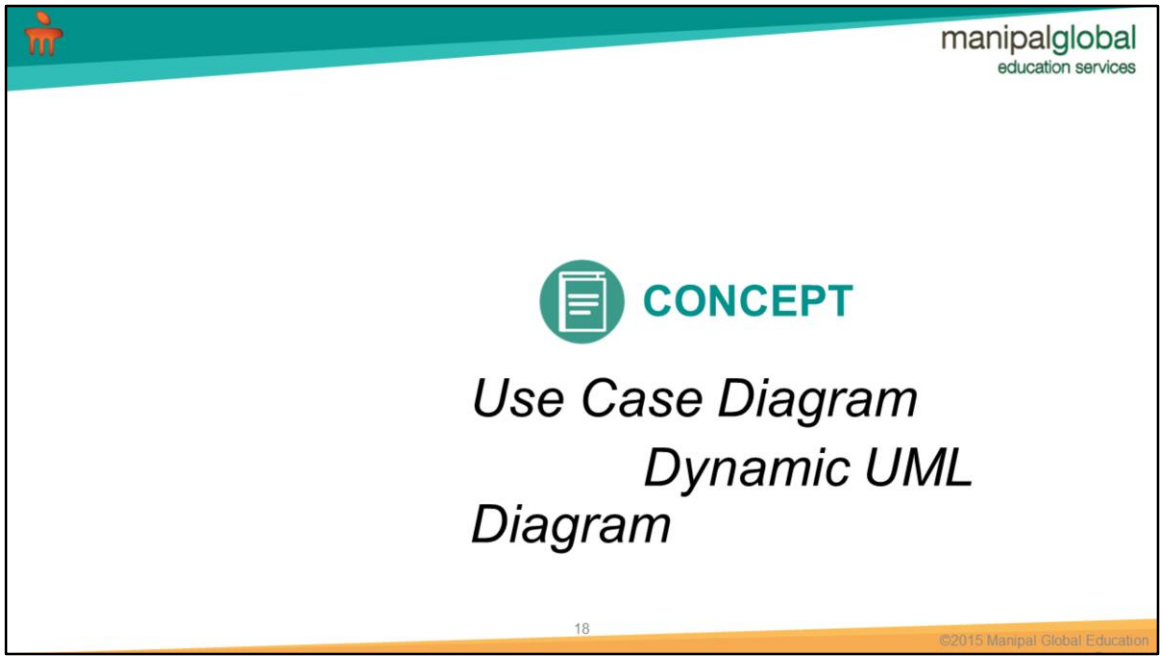
- UML Diagrams
  - Each diagram is a “view” into a model
    - Presented from the aspect of a particular stakeholder
    - Provides a partial representation of the system
    - Not all systems require all views
    - Single process: drop process view
    - Very small program: drop implementation view

manipalglobal
education services

MODELLING STEPS

- Use Cases
  - Requirement Gathering
- Domain Model
  - Process Capturing
  - Identification of Important classes
- Design Model
  - Capture details of use cases and domain objects
  - Add all the classes that do the work
  - Define the architecture


manipalglobal
education services
BUILDING BLOCKS OF UML
Static designs describes code structure and object relations
6 Static / Structural diagrams
Class diagram
Object diagram
Component diagram
Composite Structure Diagram (UML 2.0)
Package Diagram (UML 2.0)
Deployment diagram
Dynamic design shows communication between objects / behaviour of objects
Seven Dynamic / Behavioral diagrams
Use case diagram
State machine diagram (Statechart)
Activity diagram
Interaction diagrams
Sequence diagram
Communication diagram (Collaboration)
Timing Diagram
Interaction Overview Diagram
17
©2015 Manipal Global Education

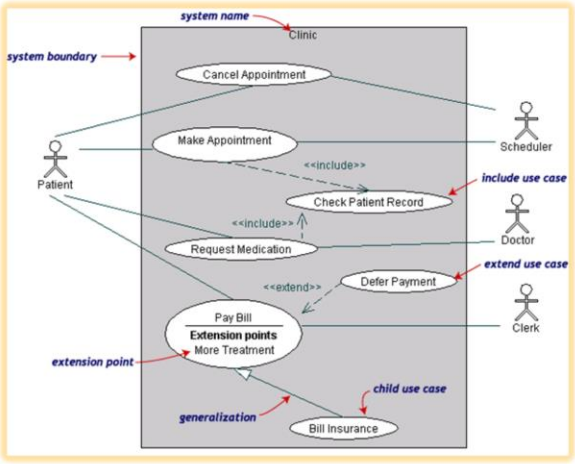

manipalglobal
education services
CONCEPT
Use Case Diagram
Dynamic UML
Diagram
18
©2015 Manipal Global Education

USE CASE DIAGRAM

- Use case diagram captures system functionality as seen by users
- Used during requirements elicitation to represent external behaviour
- Complete description of the functionality of the system and its environment
- Shows set of use cases and actors and their relationship
- Built in early stages of development

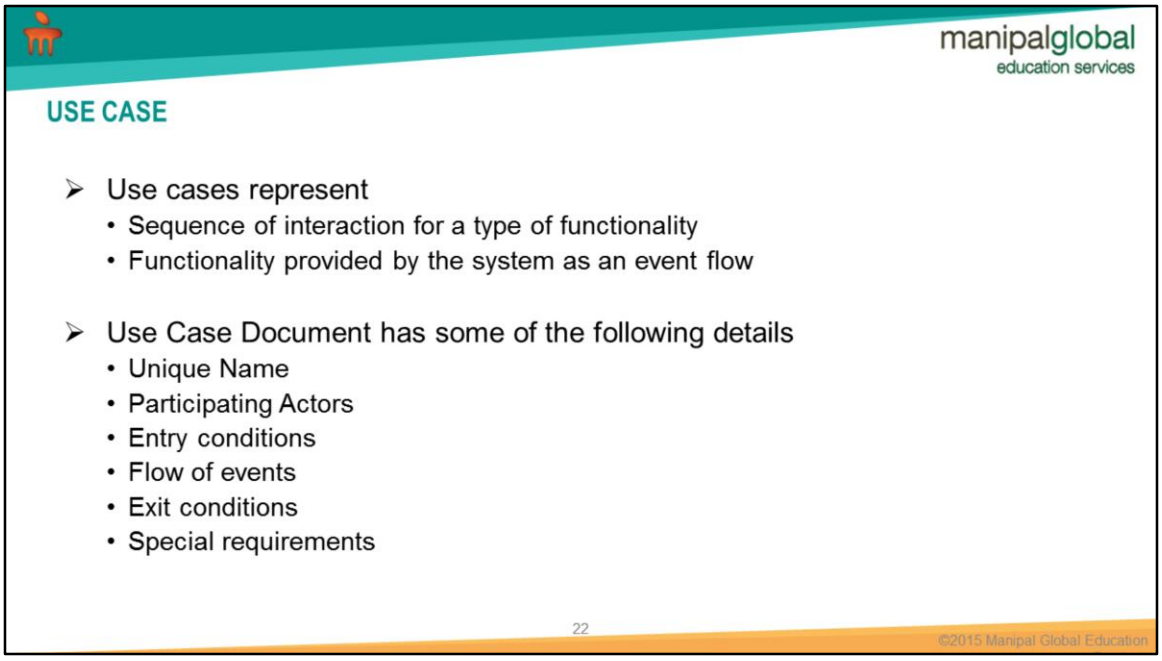
- Purpose
  - Specify the context of a system
  - Capture the requirements of a system
  - Validate a system's architecture
  - Drive implementation and generate test cases
- Developed by analysts and domain experts

UNIFIED MODELING LANGUAGE(UML) > USE CASE DIAGRAM



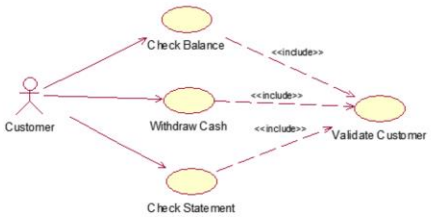
ACTOR

- Actors represent
  - type of user of the system
  - models a type of role played by an entity
  - does not necessarily represent a specific physical entity but merely a role of some entity
  - Could be a user, External System, Physical Environment
    - Ex. Employee, Patient, Payment Gateway


manipalglobal
education services
USE CASE
Use cases represent
Sequence of interaction for a type of functionality
Functionality provided by the system as an event flow
Use Case Document has some of the following details
Unique Name
Participating Actors
Entry conditions
Flow of events
Exit conditions
Special requirements
22
©2015 Manipal Global Education

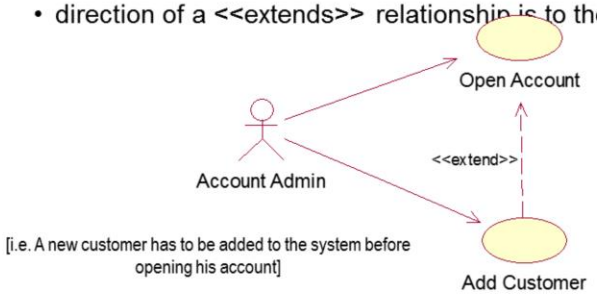
RELATIONSHIP BETWEEN USE CASES

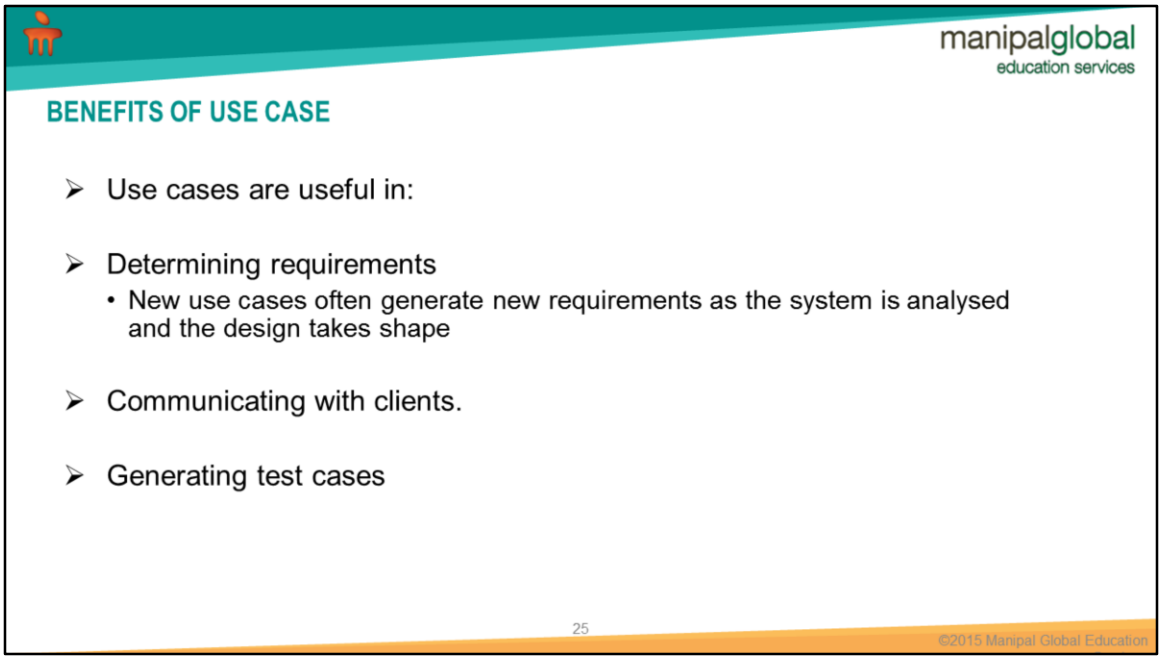
- <<includes>> Relationship
  - Included Use Case is fully executed
  - Not conditional
  - <<includes>> behavior is factored out for reuse, not because it is an exception
  - The direction of a <<includes>> relationship is to the using use case

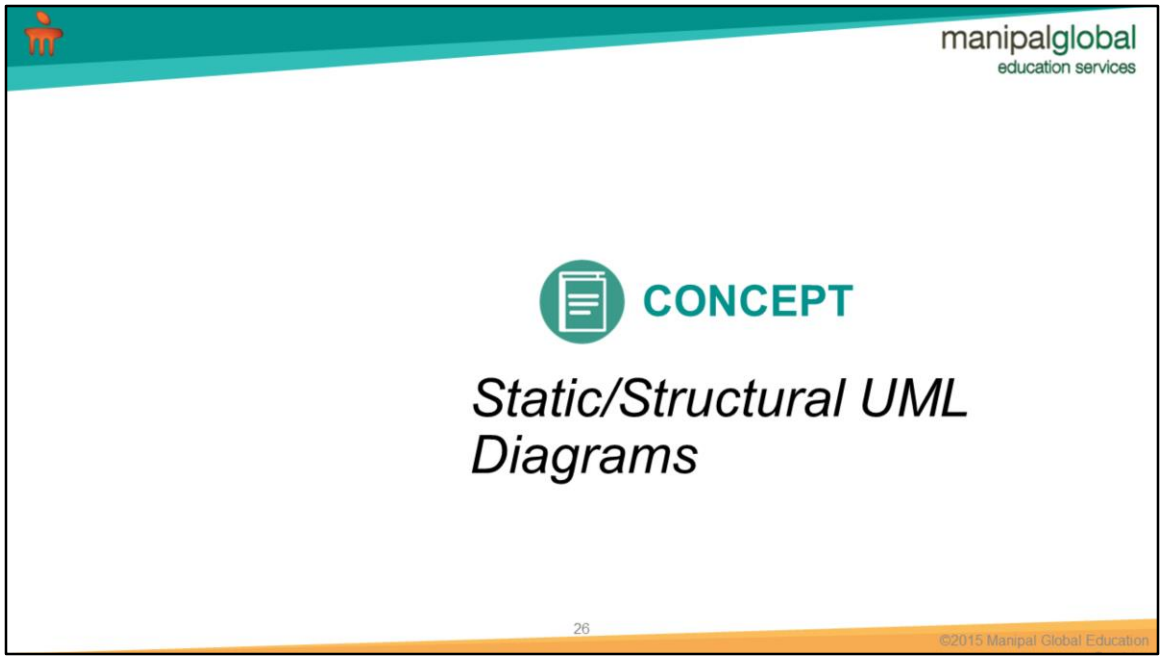


RELATIONSHIP BETWEEN USE CASES

- <<extends>> Relationship
  - represent exceptional or seldom invoked cases
  - Extended Use Case functionality inserted into base Use Case only when extending condition is true
  - exceptional event flows are factored out of the main event flow for clarity
  - direction of a <<extends>> relationship is to the extended use case




manipalglobal
education services
BENEFITS OF USE CASE
Use cases are useful in:
Determining requirements
New use cases often generate new requirements as the system is analysed and the design takes shape
Communicating with clients.
Generating test cases
25
©2015 Manipal Global Education

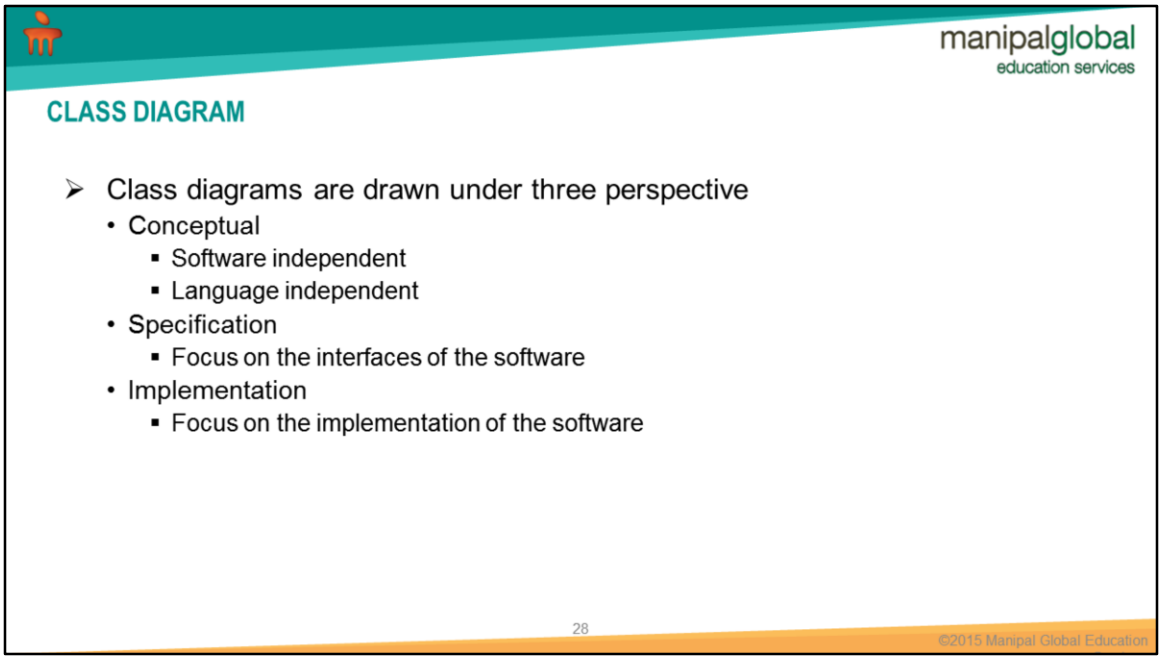

manipalglobal
education services
CONCEPT
Static/Structural UML Diagrams
26
©2015 Manipal Global Education

manipalglobal
education services

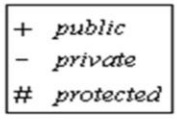
CLASS DIAGRAM

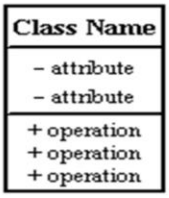
- Class diagram is a static structure diagram that describes the structure of the system showing
  - Classes and attributes, interfaces
  - Operations
  - Relationship among classes

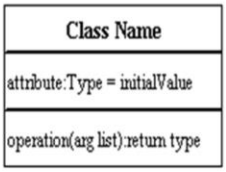
- Features
  - Gives an overview of a system by showing its classes and the relationships among them.
  - They display what interacts but not what happens when they do interact.
  - Also shows attributes and operations of each class.
  - Good way to describe the overall architecture of system components
  - Developed by analysts, designers, and implementers


manipalglobal
education services
CLASS DIAGRAM
Class diagrams are drawn under three perspective
Conceptual
Software independent
Language independent
Specification
Focus on the interfaces of the software
Implementation
Focus on the implementation of the software
28
©2015 Manipal Global Education

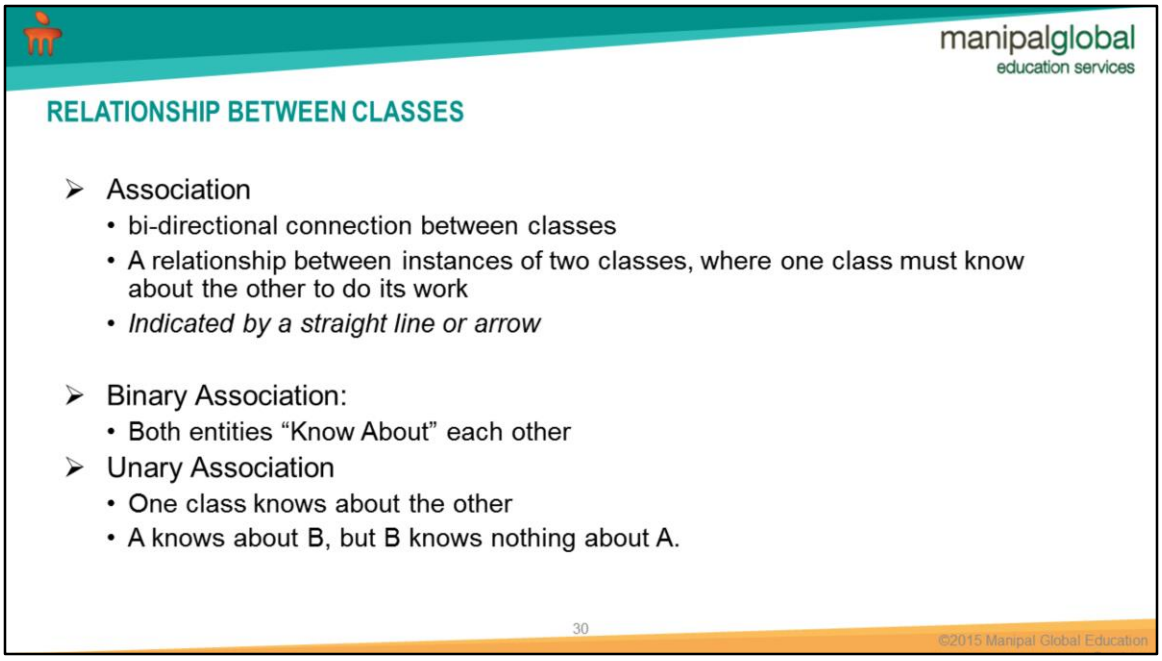
CLASS DIAGRAM







- A class is represented in a rectangle divided into three parts:
  - Class name
  - Class attributes (data members, variables)
  - Class operations (methods)
- Modifiers
  - Private : -
  - Public : +
  - Protected : #
- Static: Underlined (shared among all members of the class)
- Abstract class: Name in italics


manipalglobal
education services
RELATIONSHIP BETWEEN CLASSES
Association
bi-directional connection between classes
A relationship between instances of two classes, where one class must know about the other to do its work
Indicated by a straight line or arrow
Binary Association:
Both entities "Know About" each other
Unary Association
One class knows about the other
A knows about B, but B knows nothing about A.
30
©2015 Manipal Global Education

manipalglobal
education services

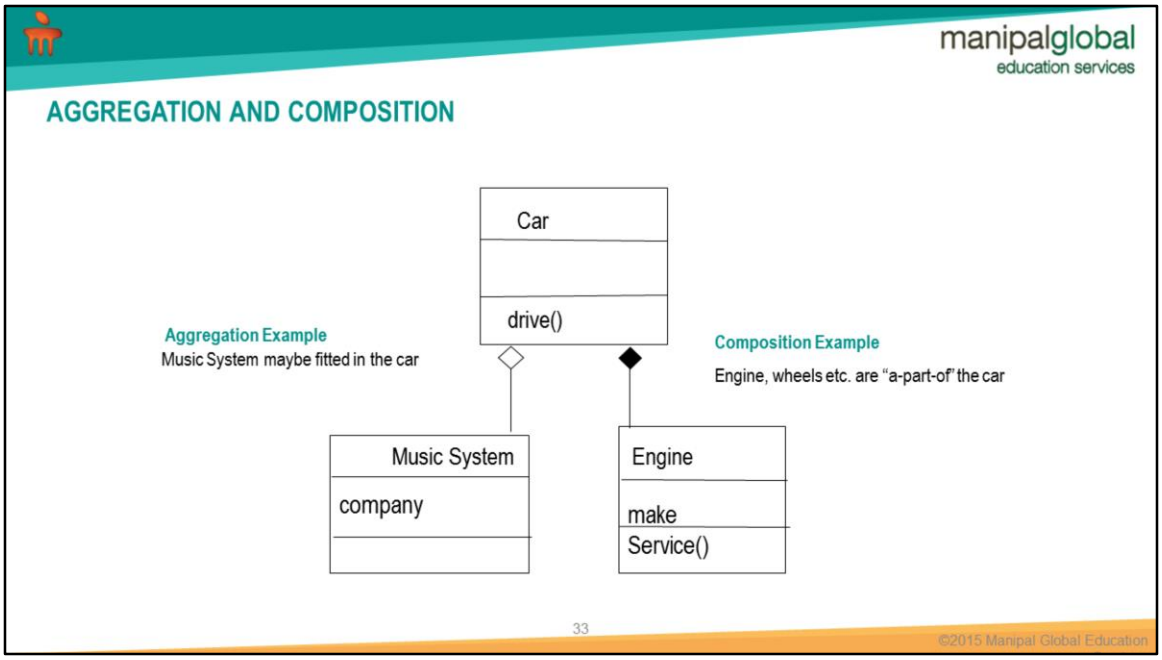
RELATIONSHIP BETWEEN CLASSES

- Aggregation or composition
  - is a relationship among classes by which a class can be made up of any combination of objects of other classes
- Aggregation
  - relationship is between a whole and its parts
  - is an association with a collection-member relationship
  - does not imply ownership
  - (has-a) Not always created and destroyed with the container (e.g. house and furniture)
  - *Indicated by an empty diamond on the side of the collection*

manipalglobal
education services

RELATIONSHIP BETWEEN CLASSES

- Composite Aggregation (or Composition)
  - Strong form of Aggregation
  - Lifetime control; (owner controls construction, destruction)
  - Indicated by a solid diamond on the side of the collection
  - Part object may belong to only one whole object
    (e.g. house & walls)


manipalglobal
education services
AGGREGATION AND COMPOSITION
Car
drive()
Aggregation Example
Music System maybe fitted in the car
Composition Example
Engine, wheels etc. are "a-part-of" the car
Music System
company
Engine
make
Service()
33
©2015 Manipal Global Education

RELATIONSHIP BETWEEN CLASSES

- Multiplicities or Cardinality of Associations
  - Cardinality of a binary association denotes the number of instances participating in an association.
  - There are three types of cardinality ratios, namely:
    - One–to–One: A single object of class A is associated with a single object of class B.
    - One–to–Many: A single object of class A is associated with many objects of class B.
    - Many–to–Many: An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

Notations

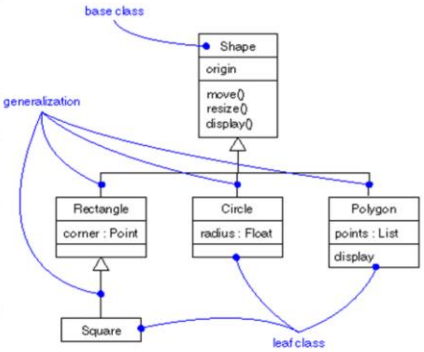
0..1	Zero or one instance. The notation n..m indicates n to m instances.
0..* or *	No limit on the number of instances (including none).
1	Exactly one instance
1..*	At least one instance

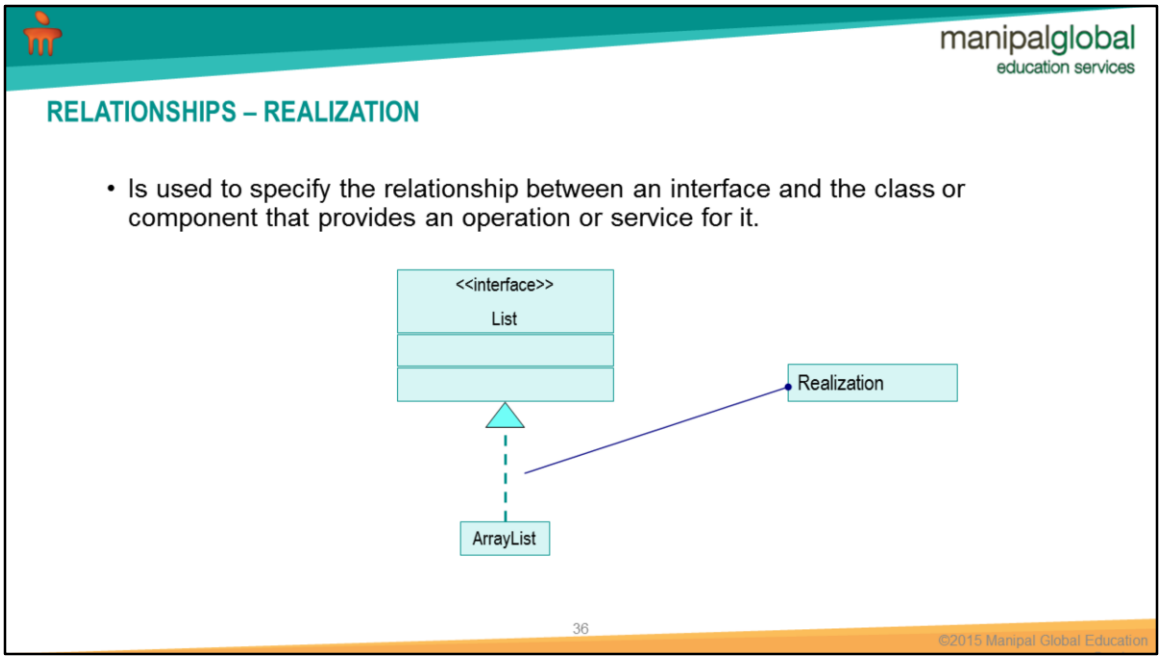
RELATIONSHIPS – GENERALIZATION AND SPECIAIZATION

- Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes

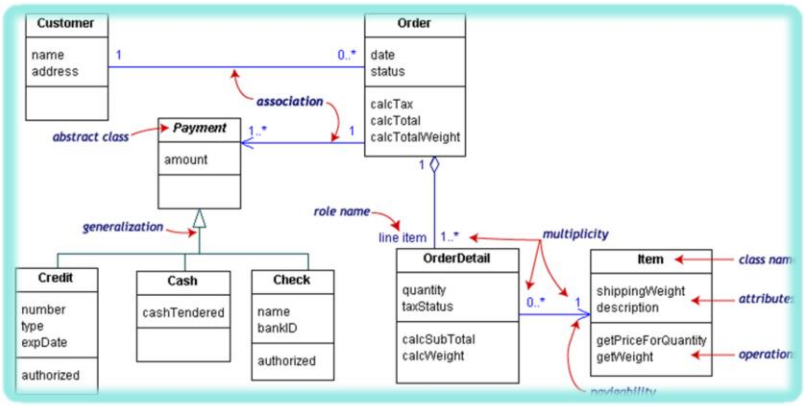
- Generalization
  - In the generalization process, the common chara of classes are combined to form a class in a high hierarchy, i.e., subclasses are combined to form super-class. It represents an "is – a – kind – of"

- Specialization
  - Specialization is the reverse process of generaliz



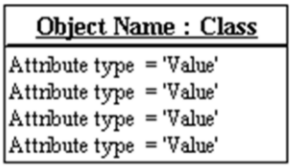

manipalglobal
education services
RELATIONSHIPS – REALIZATION
• Is used to specify the relationship between an interface and the class or component that provides an operation or service for it.
<<interface>>
List
Realization
ArrayList
36
©2015 Manipal Global Education

CLASS DIAGRAM

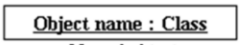


OBJECT DIAGRAM

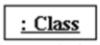
- Object diagram is a set of objects and their relationship at a point in time
- Captures objects and links
- An object diagram is a snapshot of the objects in a system
  - At a point in time
  - With a selected focus



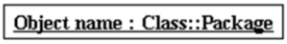
Object with attributes



Named object

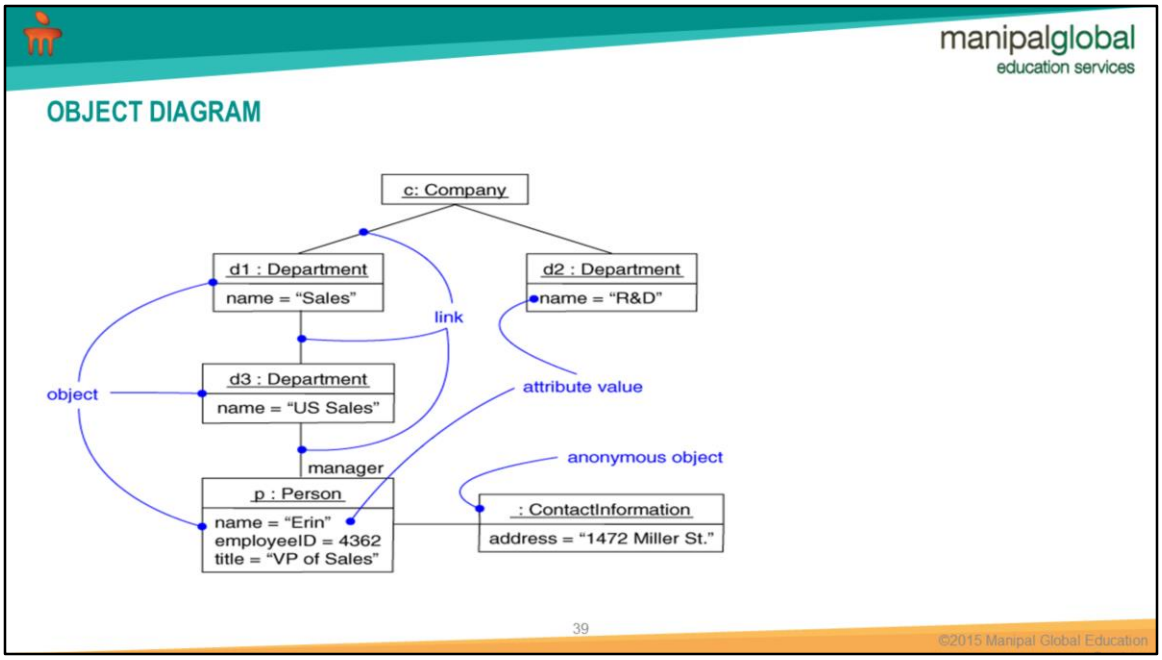


Unnamed object



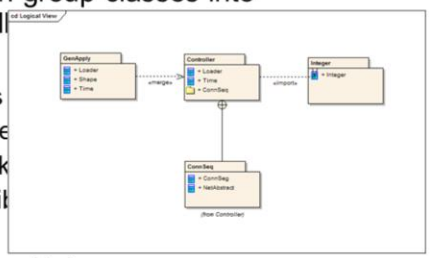
Named object with path name

Note: The object diagram does not show messages across the objects (passing–Collaboration diagram)


manipalglobal
education services
OBJECT DIAGRAM
c: Company
d1 : Department
name = "Sales"
d2 : Department
name = "R&D"
link
d3 : Department
name = "US Sales"
object
attribute value
manager
anonymous object
p : Person
name = "Erin"
employeeID = 4362
title = "VP of Sales"
: ContactInformation
address = "1472 Miller St."
39
©2015 Manipal Global Education

PACKAGE DIAGRAM

- Package Diagrams are used to reflect the organization of packages and their elements.
- To organize complex class diagrams, one can group classes into packages. A package is a collection of logicall
- Notation
  - Packages appear as rectangles with small tabs
  - The package name is on the tab or inside the re
  - The dotted arrows are dependencies. One pack on another if changes in the other could possib changes in the first.
  - Packages are the basic grouping construct with which you may organize UML models to increase their readability



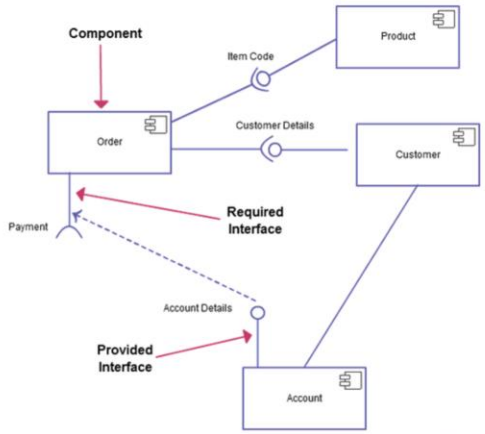
manipalglobal
education services

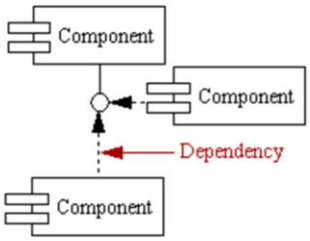
COMPONENT DIAGRAM

- Component diagram shows the organizations and dependencies among a set of components
- Represents the Structure of a system
- Usually a physical collection of classes
  - Similar to a Package diagram in that both are used to group elements into logical structures
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database

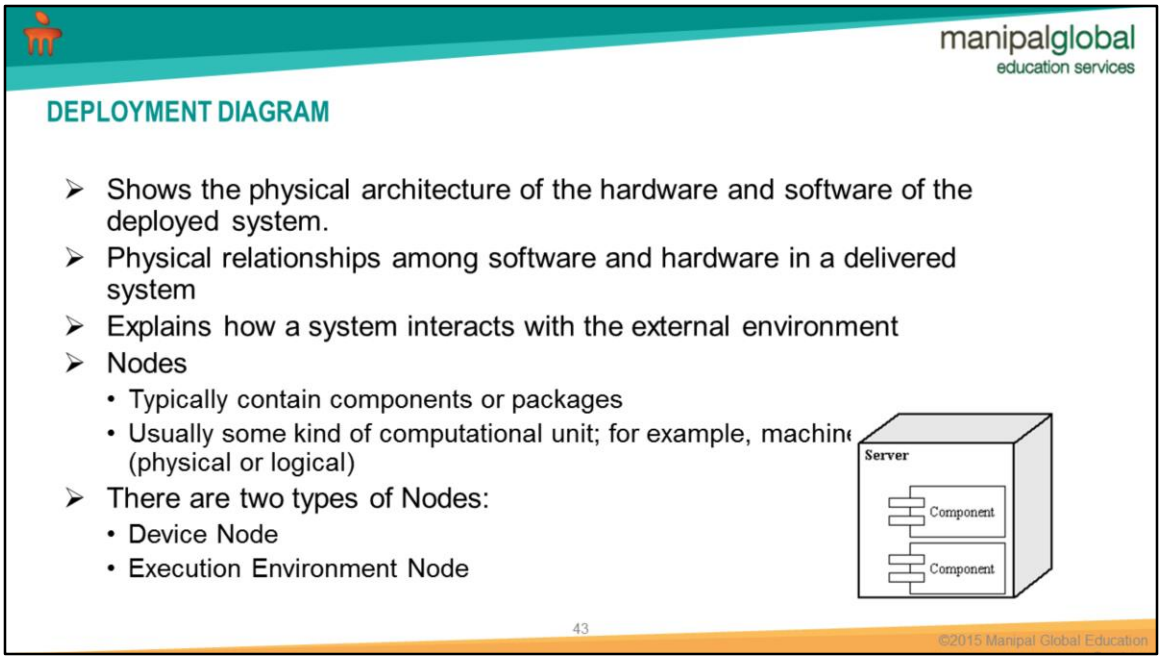
manipalglobal
education services

COMPONENT DIAGRAM

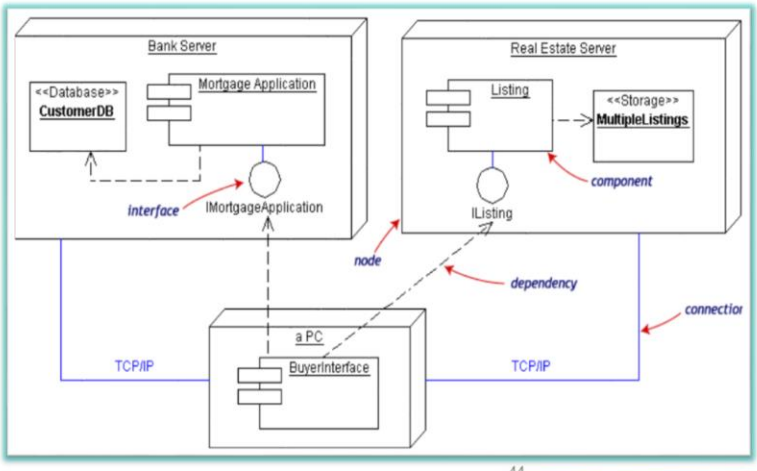




Dashed arrows indicate dependencies
Circle and solid line indicates an interface to the component


manipalglobal
education services
DEPLOYMENT DIAGRAM
Shows the physical architecture of the hardware and software of the deployed system.
Physical relationships among software and hardware in a delivered system
Explains how a system interacts with the external environment
Nodes
Typically contain components or packages
Usually some kind of computational unit; for example, machin
(physical or logical)
There are two types of Nodes:
Device Node
Execution Environment Node
Server
Component
Component
43
©2015 Manipal Global Education

DEPLOYMENT DIAGRAM




manipalglobal
education services
CONCEPT
Dynamic/Behavioural UML Diagrams
45
©2015 Manipal Global Education

manipalglobal
education services

INTERACTION DIAGRAM

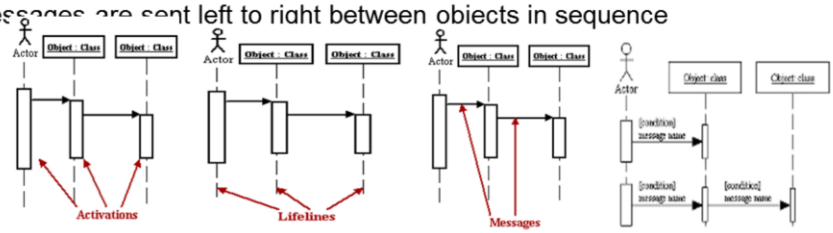
- An interaction diagram is a graphical representation of interactions between objects
- emphasize the flow of control and data among
- There are four kinds of interaction diagrams
  - Sequence diagram
  - Communication diagram
  - Interaction Overview diagram
  - Timing diagram

- Each provides a different view of the same interaction
  - Sequence diagrams represent interaction with respect to time
  - Communication diagrams represent interaction with sequenced messages

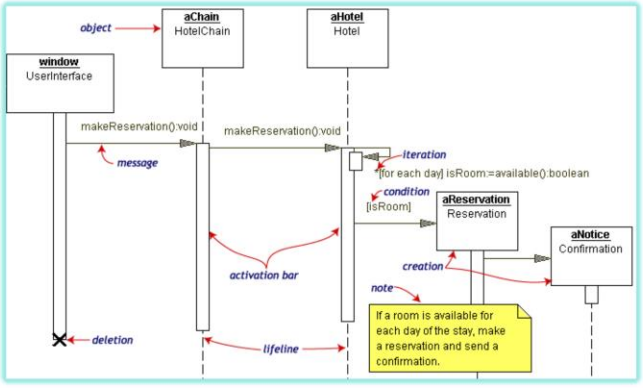
Note: Sequence diagram and communication diagram are semantically equivalent

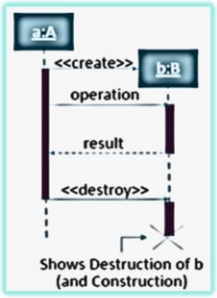
DYNAMIC (BEHAVIORAL) DIAGRAMS

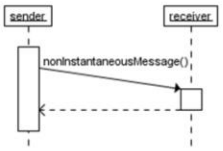
- Sequence Diagram
  - A sequence diagram is an interaction diagram that emphasizes the time ordering of messages
  - Time progresses from top to bottom
  - Objects involved are listed left to right
  - Messages are sent left to right between objects in sequence



SEQUENCE DIAGRAM







Slanted lines show propagation delay of messages
Good for modelling real-time systems

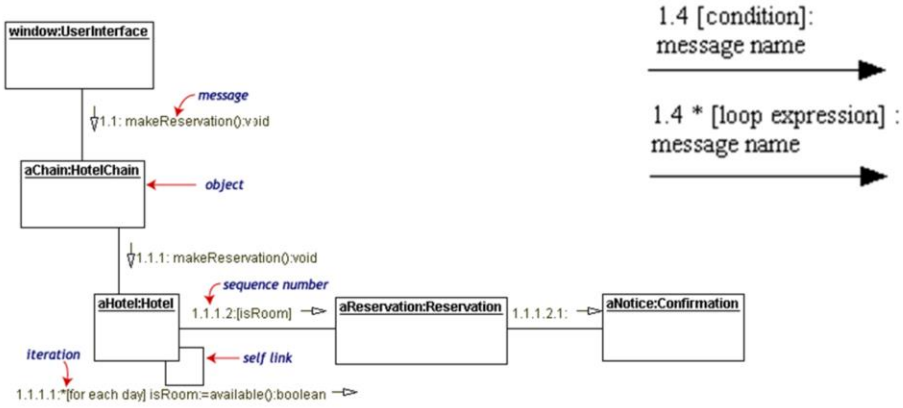
manipalglobal
education services

COLLABORATION DIAGRAM (COMMUNICATION DIAGRAM)

- Represent interactions between objects as a series of sequenced messages
  - Emphasizes the structural organization of the objects sending and receiving the messages
- Captures dynamic behaviour (message-oriented)
- Easier to depict complex procedural logic or multiple parallel transactions

- Purpose
  - Model flow of control over the structural organization of the objects

COLLABORATION DIAGRAM

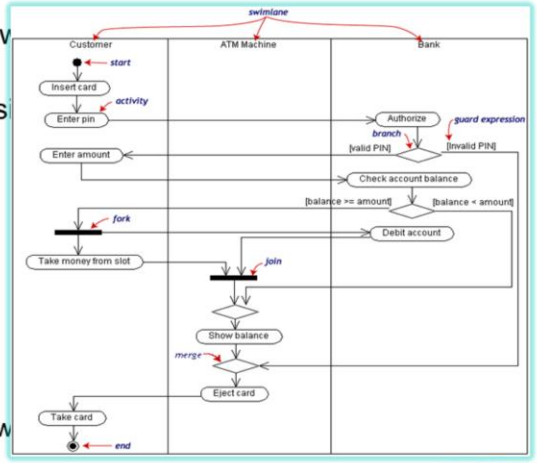


ACTIVITY DIAGRAM

- A fancy flowchart which displays the flov
  activities involved in a single process
- Typically used to model workflow or bus
  operations

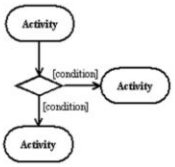
- States
  - Describe what is being processed
  - Represent actions
  - Indicated by boxes with rounded corners

- SwimLanes
  - Indicates which object is responsible for w
  - Groups related activities in on column

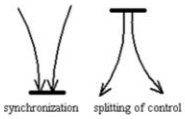


DYNAMIC (BEHAVIORAL) DIAGRAMS

- Branch
  - Decision making point
  - Indicated by a diamond



- Fork/Join
  - transition forking into parallel activitie
  - Indicated by solid bars



Synchronization -- also called a "join"

Splitting of Control -- also called a "fork"

- Black circle represents start of workflow(initial state)
- An encircled black circle represents the end (final state)

STATE TRANSITION /STATE MACHINE DIAGRAM

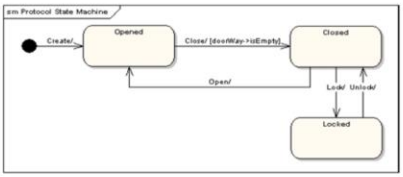
- Represent the possible states of the object and the transitions that cause a change in state
- Models the behavior of a single object over its lifetime, specifying the states an object goes through in response to events
- Especially useful in modelling reactive objects whose states are triggered by specific events
- Actions within each state – syntax: do/ entry/ exit/
- Purpose
  - Shows the possible states of the object and the transitions that cause a change in state (that is, how incoming calls change the state)

manipalglobal
education services

STATE MACHINE DIAGRAM

- E.g. A door can be in one of three states: Opened, Closed or Locked
- It can respond to the events Open, Close, Lock and Unlock.
  - i.e. State transitions can be triggered by above events
- Not all events are valid in all states: for example, if a door is Opened, you cannot lock it until you close it
- A state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition doorWay->isEmpty is true
- State diagrams can also store the sing on them

State Machine diagram for Door



SUMMARY

In this lesson, you've learned to:

- Understand System Modelling using Unified Modelling Language(UML)
- Describe UML and different types of diagrams
- Distinguish between static and dynamic UML diagrams
- Apply UML diagrammatic techniques
- Create Use Case diagram and Class diagrams
- Understand the representation and use of the UML diagrams