# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L-1**

**CSE**

**III / V**

**Course Code & Name** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Name of the Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : I- UML DIAGRAMS **Date of Lecture:**

**Topic of Lecture: Introduction to OOAD**

**Introduction :**

- Object-oriented analysis and design (OOAD) is a technical approach for analyzing and designing an application, system, or business by applying object-oriented programming, as well as using visual modeling throughout the software development process to guide stakeholder communication and product quality.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Class and Object
- Message, operation and method
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

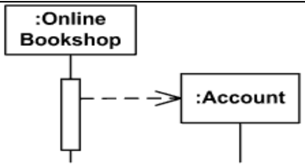
**Detailed content of the Lecture:**

**Class and Object**

- UML class is a classifier which describes a set of objects that share the same features, Constraints, semantics.
- Class may be modeled as being active, meaning that an instance of the class has some autonomous behavior.
- Object is an instance of a class.

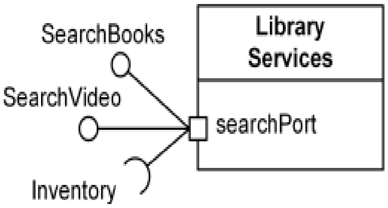
**Message, operation and method**

- Messages are intrinsic elements of UML interaction diagrams. A message defines a specific kind of communication between lifelines of an interaction.
- A communication can be, for example, invoking an operation, replying back, creating or destroying an instance, raising a signal.
- It also specifies the sender and the receiver of the message.
- Create message is shown as a dashed line with open arrowhead, and pointing to the created lifeline's head.



**Encapsulation**

- Encapsulation was describing abstraction mechanisms in programming language CLU in the context of hiding details of implementation**.**
- Encapsulation is a development technique which includes
- creating new data types classes by combining both information structure and behaviors, and
- restricting access to implementation details.
- Encapsulated classifier in UML is a structured classifier isolated from its environment by using ports. Each port specifies a distinct interaction point between classifier and its environment.



**Abstraction**

- Abstraction is a dependency relationship that relates two elements or sets of elements called client and supplier representing the same concept but at different levels of abstraction or from different viewpoints.
- Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification the supplier and the other represents an implementation of the latter the client.

**Inheritance**

- Inheritance as the mechanism by which those more specific elements incorporate structure and behavior of the more general elements. Inheritance supplements generalization relationship.
- Generalization is defined as a taxonomic relationship between a more general element and a more specific element.

**Polymorphism**

Polymorphism is ability to apply different meaning semantics, implementation to the same symbol message, and operation in different contexts.

When context is defined at compile time, it is called static or compile-time polymorphism**.** When context is defined during program execution, it is dynamic or run-time polymorphism.

**Video Content / Details of website for further learning (if any):**
https://www.uml-diagrams.org/uml-object-oriented-concepts.html

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[1-16]

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L -2**

**CSE**

**III / V**

**Course Code & Name** : **16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN**

**Name of the Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **I- UML DIAGRAMS** **Date of Lecture:**

**Topic of Lecture: Unified Process**

**Introduction :**

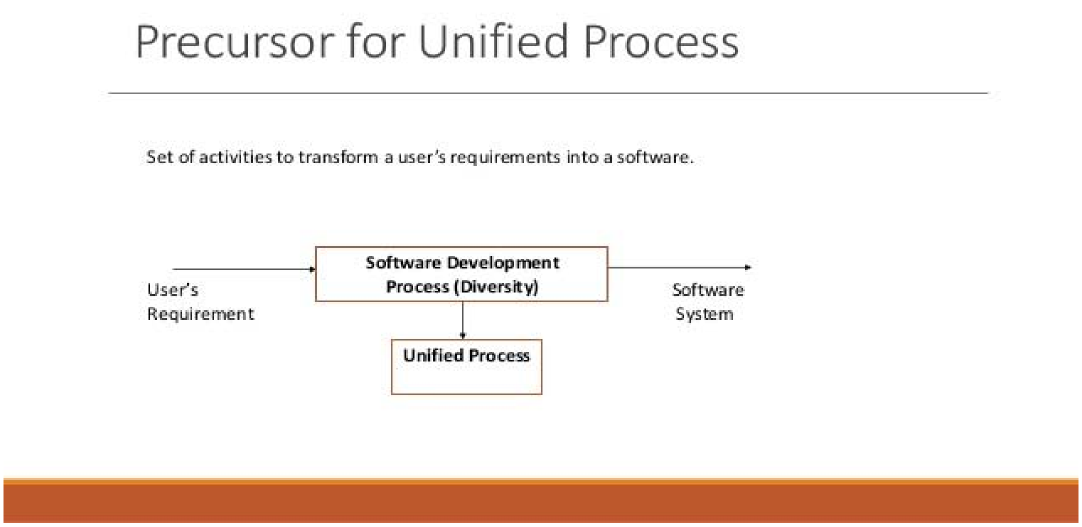
- The Unified Process (UP), or Unified Software Development Process, is a iterative and incremental software development framework from which a customized process can be defined.
- The framework contains many components and has been modified a number of times to create several variations.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Design
- Implementation
- Testing
- deployment

**Detailed content of the Lecture:**

- Unified process (UP) is an architecture-centric, use-case driven, iterative and incremental development process that leverages unified modeling language and is compliant with the system process engineering metamodel .
- Unified process can be applied to different software systems with different levels of technical and managerial complexity across various domains and organizational cultures.



Precursor for Unified Process

Set of activities to transform a user's requirements into a software.

**Analysis and Design**

- Analysis and Design discipline would be better named the Solution Analysis and Design discipline in my opinion.
- This is because the requirements are analyzed from a solution design perspective, rather than a requirements analysis perspective. Specific activities that are part of this discipline include:
  - Understanding and analyzing the requirements for the system
  - Defining a candidate architecture for a system
  - Constructing a proof-of-concept or prototype to validate a candidate architecture
  - Design of components, services, and/or modules
  - Design of interfaces (network, user, and databases)

**Implementation**

The Implementation discipline consists of coding, unit testing, and integration of the software.

**Testing**

The Testing discipline is focused on quality assurance of the software being released in that cycle or iteration. It includes such activities as: [4]

- Planning test efforts
- Creating test cases
- Running tests
- Reporting defects

**Deployment**

The Deployment discipline is focused on planning the deployment of, and actually deploying, the software that is being completed that cycle, phase or iteration. It includes such activities as:[4]

- Planning the deployment
- Developing support and operations materials
- Planning alpha, beta, and pilot testing efforts
- Deploying the software
- Training end users
- Managing acceptance testing efforts

**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=nFoumijTcUg

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[25-25]

**Course Faculty**

**Verified by HOD**

**LECTURE HANDOUTS**

**L-3**

**CSE**

**III / V**

**Course Code & Name** : **16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN**

**Name of the Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **I- UML DIAGRAMS** **Date of Lecture:**

**Topic of Lecture: UML diagram –Use Cases**

**Introduction :**

- The Unified Modeling Language (**UML**) is a graphical language for OOAD that gives a standard way to write a software system's blueprint.
- It helps to visualize, specify, construct, and document the artifacts of an object-oriented system.
- It is used to depict the structures and the relationships in a complex system.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Dynamic behavior.
- UML

**Detailed content of the Lecture:**

- Use case diagrams are considered for high level requirement analysis of a system.
- When the requirements of a system are analyzed, the functionalities are captured in use cases.
- We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors.
- Actors can be defined as something that interacts with the system.

Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.

- Functionalities to be represented as use case
- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram

- The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.
- Do not try to include all types of relationships, as the main purpose of the diagram is to identify the requirements.

- Use notes whenever required to clarify some important points.

Following is a sample use case diagram representing the order management system. Hence, if we look into the diagram then we will find three use cases **(Order, Special Order, and Normal Order)** and one actor which is the customer.

The Special Order and Normal Order use cases are extended from Order use case. Hence, they have extended relationship. Another important point is to identify the system boundary, which is shown in the picture. The actor Customer lies outside the system as it is an external user of the system.

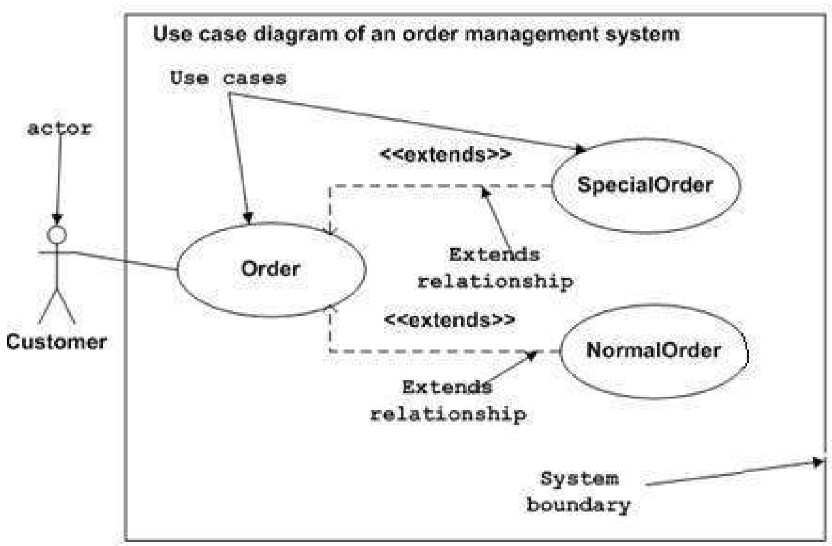


Figure: Sample Use Case diagram

**Video Content / Details of website for further learning (if any):**
https://www.smartdraw.com/uml-diagram/

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[10-19]

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L-4

CSE

III / V

**Course Code & Name** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Name of the Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : I- UML DIAGRAMS **Date of Lecture:**

**Topic of Lecture: Class diagrams**

**Introduction :**

- Class diagrams are the backbone of almost every object-oriented method, including UML.
- They describe the static structure of a system.

**Prerequisite knowledge for Complete understanding and learning of Topic**

- Building blocks of object oriented modeling
- Dynamic structure of the system

**Detailed content of the Lecture:**

**Class diagram**

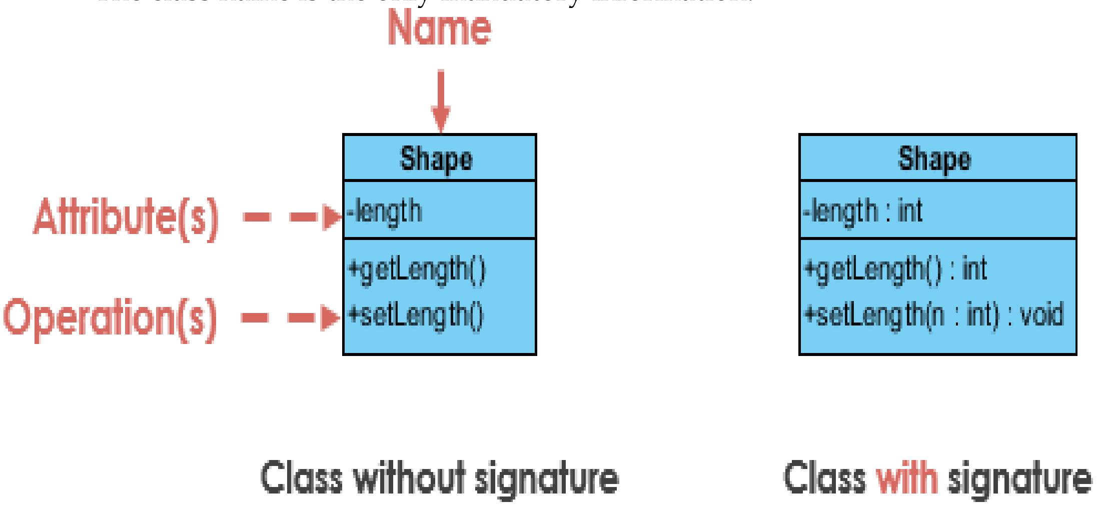
- The UML Class diagram is a graphical notation used to construct and visualize object oriented systems.
- A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:classes,theirattributes,operations (or methods),
and the relationships among objects.
- A Class is a blueprint for an object. Objects and classes go hand in hand. and the entire point of Object-Oriented Design is not about objects, it's about classes, because we use classes to create objects. So a class describes what an object will be, but it isn't the object itself.
- In fact, classes describe the type of objects, while objects are usable instances of classes.
- Each Object was built from the same set of blueprints and therefore contains the same components (properties and methods). The standard meaning is that an object is an instance of a class and object - Objects have states and behaviors.
- A Class is a blueprint for an object. Objects and classes go hand in hand. So a class describes what an object will be, but it isn't the object itself.
- In fact, classes describe the type of objects, while objects are usable instances of classes.
- Each Object was built from the same set of blueprints and therefore contains the same

components (properties and methods).

- The standard meaning is that an object is an instance of a class and object - Objects have states and behaviors.

**UML Class Notation**

- A class represent a concept which encapsulates state attributes and behavior operations.
- Each attribute has a type. Each operation has a signature.
- The class name is the only mandatory information.



**Video Content / Details of website for further learning (if any):**
[https://www.youtube.com/watch?v=UI6lqHOVHic](https://www.youtube.com/watch?v=UI6lqHOVHic)

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[35-50]

**Course Faculty**

**Verified by HOD**

**LECTURE HANDOUTS**

**L-5**

**CSE**

**III / V**

**Course Code & Name** : **16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN**

**Name of the Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **I- UML DIAGRAMS** **Date of Lecture:**

**Topic of Lecture: - Interaction diagrams**

**Introduction :**

- This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram.
- The basic purpose of both the diagrams is similar.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

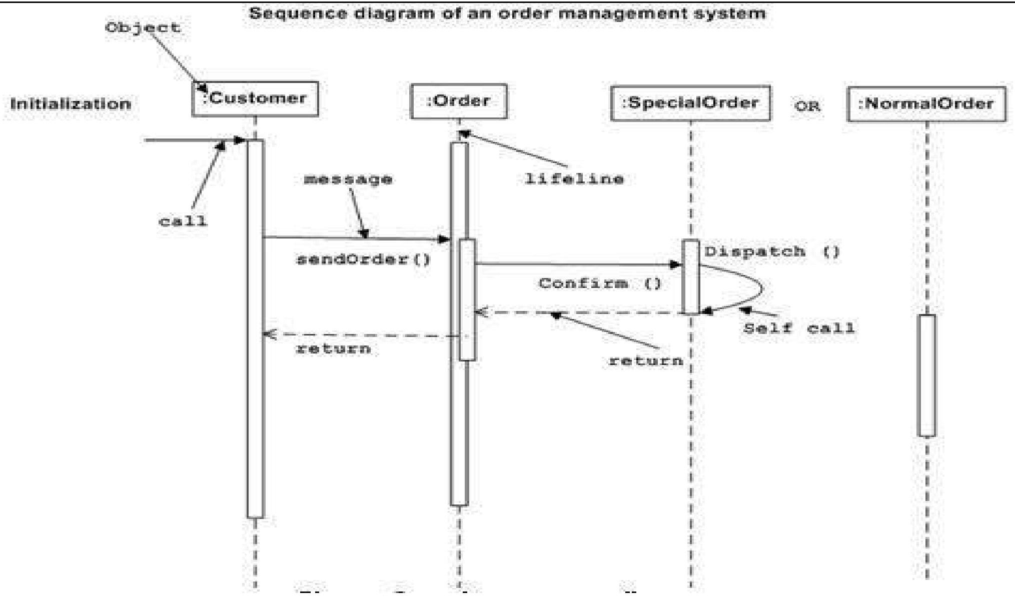
- Dynamic behavior of a system
- Message flow in the system
- Structural organization of the objects
- Interaction among objects

**Detailed content of the Lecture:**

- Two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram.

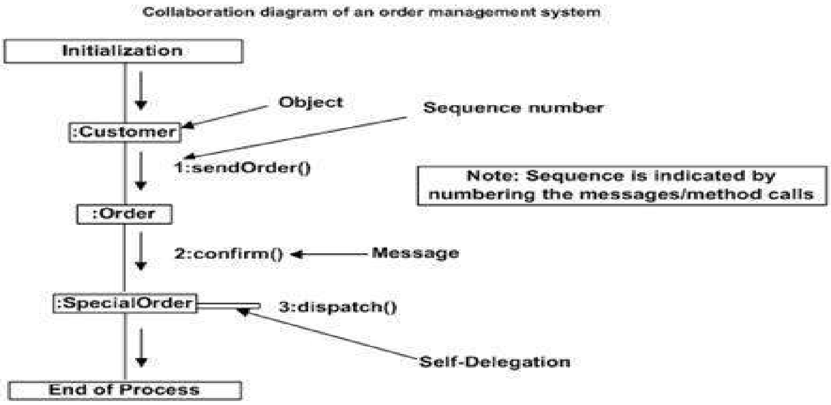
**Sequence Diagram**

- The sequence diagram has four objects (Customer, Order, Special Order and Normal Order).
- The following diagram shows the message sequence for Special Order object and the same can be used in case of Normal Order object.
- It is important to understand the time sequence of message flows.
- The message flow is nothing but a method call of an object.
- The first call is send Order () which is a method of Order object.
- The next call is confirm () which is a method of Special Order object and the last call is Dispatch () which is a method of Special Order object.
- The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.



## The Collaboration Diagram

- The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram.
- In the collaboration diagram, the method call sequence is indicated by some numbering technique.
- The number indicates how the methods are called one after another.
- We have taken the same order management system to describe the collaboration diagram.
- However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.
- To choose between these two diagrams, emphasis is placed on the type of requirement.
- If the time sequence is important, then the sequence diagram is used.
- If organization is required, then collaboration diagram is used.



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=Ba7SyM78cUM

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L-6**

**CSE**

**III / V**

**Course Code & Name** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Name of the Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : I- UML DIAGRAMS **Date of Lecture:**

**Topic of Lecture: - State diagram**

**Introduction :**

- A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems.
- State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

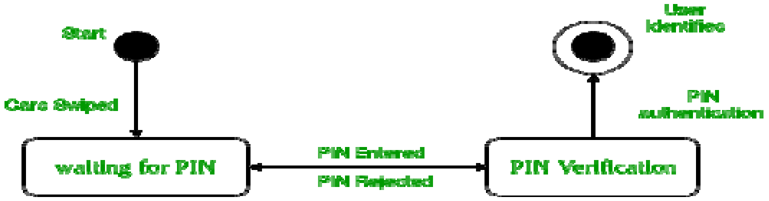
- Behavior of the system
- Reasonable abstraction

**Detailed content of the Lecture:**

- A state diagram is used to represent the condition of the system or part of the system at finite instances of time..

**Difference between state diagram and flowchart –**

The basic purpose of a state diagram is to portray various changes in state of the class and not the processes or commands causing the changes.



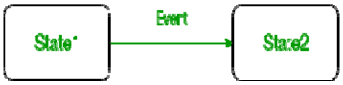
state diagram for user verification

**Basic components of a statechart diagram –**

**Initial state –** We use a black filled circle represent the initial state of a System or a class.

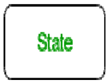
initial state notation

**Transition –** We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.



transition

**State –** We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.



state notation

**Fork –** We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states.

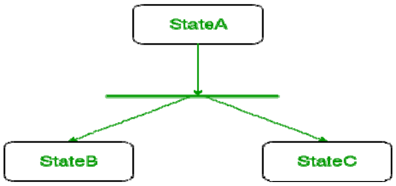


diagram using the fork notation

**Join –** We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state.

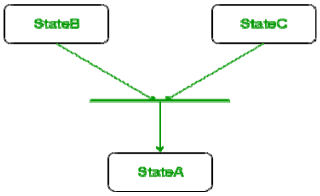
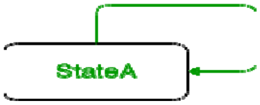


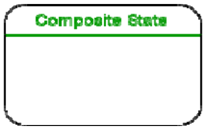
diagram using join notation

**Self transition –** We use a solid arrow pointing back to the state itself to represent a self transition.



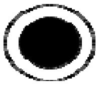
self transition notation

**Composite state –** We use a rounded rectangle to represent a composite state also. We represent a state with internal activities using a composite state.



state with internal activities

**Final state –** We use a filled circle within a circle notation to represent the final state in a state machine diagram.



final state notation

**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=KBNSZp2Ysdg

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[107-115]

**Course Faculty**

**Verified by HOD**

L-7

**LECTURE HANDOUTS**

**CSE**

**III / V**

**Course Code & Name** : **16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN**

**Name of the Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **I- UML DIAGRAMS** **Date of Lecture:**

**Topic of Lecture:** - Activity diagrams

**Introduction :**

- Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

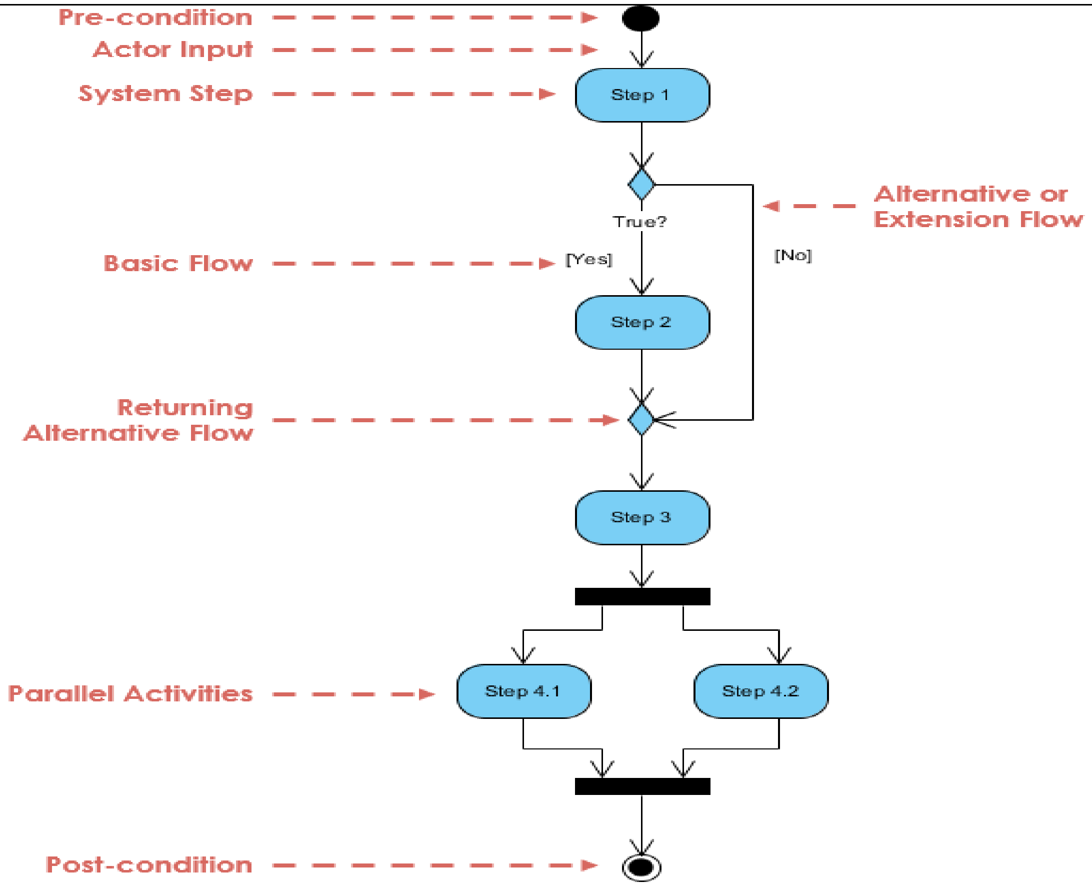
- Behavior of a system
- Interaction and concurrency

**Detailed content of the Lecture:**

- Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction.
- Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination.
- It is also suitable for modeling how a collection of use cases coordinate to represent business workflows
    1. Identify candidate use cases, through the examination of business workflows
    2. Identify pre- and post-conditions (the context) for use cases
    3. Model workflows between/within use cases
    4. Model complex workflows in operations on objects
    5. Model in detail complex activities in a high level activity Diagram

Activity Diagram - Learn by Examples

A basic activity diagram - flowchart like



Activity Diagram - Modeling a Word Processor

The activity diagram example below describes the workflow for a word process to create a document through the following steps:

- Open the word processing package.
- Create a file.
- Save the file under a unique name within its directory.
- Type the document.
- If graphics are necessary, open the graphics package, create the graphics, and paste the graphics into the document.
- If a spreadsheet is necessary, open the spreadsheet package, create the spreadsheet, and paste the spreadsheet into the document.
- Save the file.
- Print a hard copy of the document.
- Exit the word processing package

**Video Content / Details of website for further learning (if any):**
https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/

**Important Books/Journals for further learning including the page nos.:**
Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, Prentice Hall of India, 3rd Edition 2015[117-130]

**Course Faculty**

**Verified by HOD**

L-8

LECTURE HANDOUTS

CSE

III / V

Course Code & Name : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

Name of the Faculty :

Year / Semester/Section : III / V/ A

Unit : I- UML DIAGRAMS Date of Lecture:

**Topic of Lecture: -**Package diagram

**Introduction :**

- A package diagram in the Unified Modeling Language depicts the dependencies between the packages that make up a model.
- A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.
- Deployment diagrams are typically used to visualize the physical hardware and software of a system.

**Prerequisite knowledge for Complete understanding and learning of Topic**

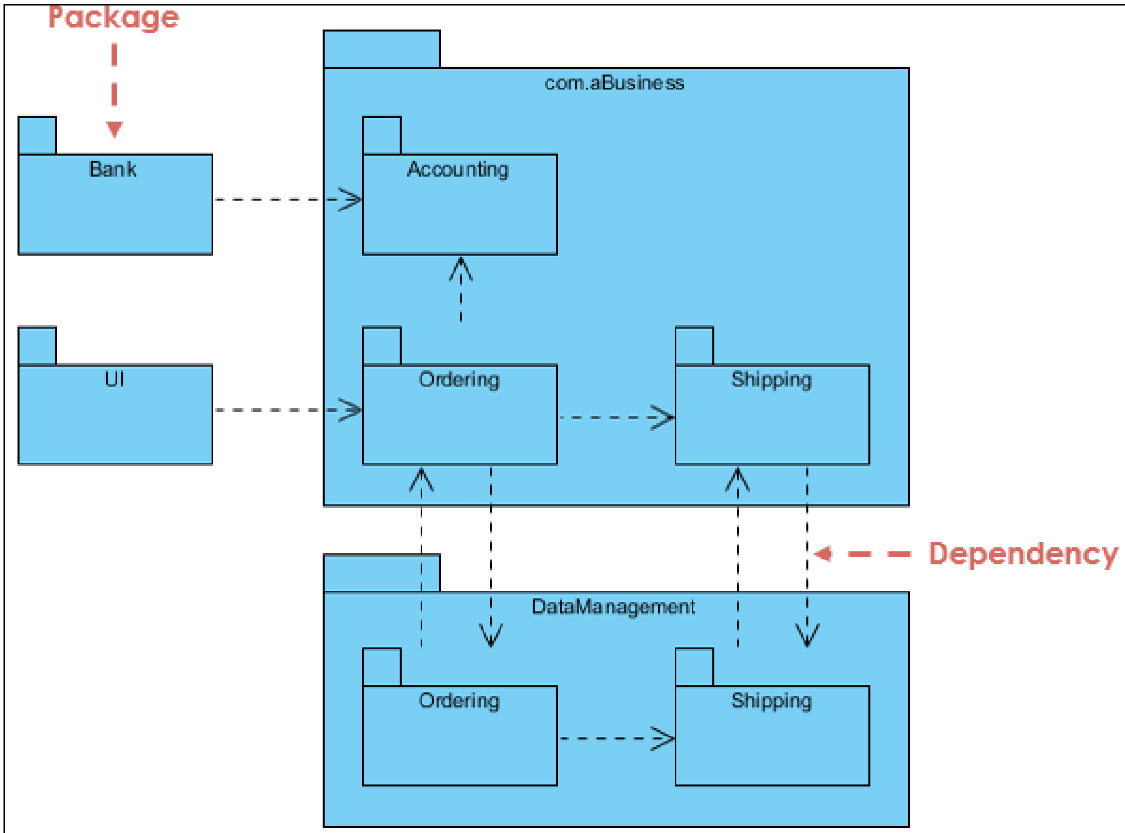
- Dependencies between packages
- Visualize physical hardware and software

**Detailed content of the Lecture:**

**Package Diagram**

Package diagrams are used to structure high level system elements. Packages are used for organizing large system which contains diagrams, documents and other key deliverables.

- Package Diagram can be used to simplify complex class diagrams, it can group classes into packages.
- A package is a collection of logically related UML elements.
- Packages are depicted as file folders and can be used on any of the UML diagrams.
- Packages appear as rectangles with small tabs at the top.
- The package name is on the tab or inside the rectangle.
- The dotted arrows are dependencies.
- One package depends on another if changes in the other could possibly force changes in the first.



**Deployment diagram**

- A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.
- Deployment diagrams are typically used to visualize the physical hardware and software of a system.

**Purpose of Deployment Diagrams**

- They show the structure of the run-time system
- They capture the hardware that will be used to implement the system and the links between different items of hardware.
- They model physical hardware elements and the communication paths between them
- They can be used to plan the architecture of a system.
- They are also useful for Document the deployment of software components or nodes.

**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=3bfJ5AORAjQ

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[89-95]

**Course Faculty**

**Verified by HOD**

L -9

## LECTURE HANDOUTS

CSE

III / V

**Course Code & Name** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Name of the Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : I- UML DIAGRAMS **Date of Lecture:**

**Topic of Lecture:** - Component and Deployment Diagrams

**Introduction :**

- Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.
- Deployment diagrams are used to describe the static deployment view of a system.
- Deployment diagrams consist of nodes and their relationships.

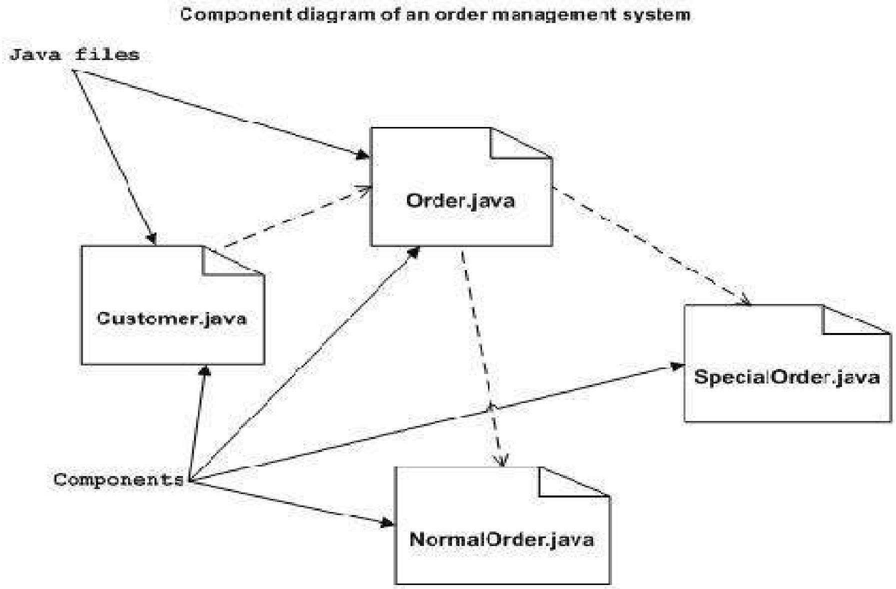
**Prerequisite knowledge for Complete understanding and learning of Topic**

- Dependencies between packages
- Visualize physical hardware and software

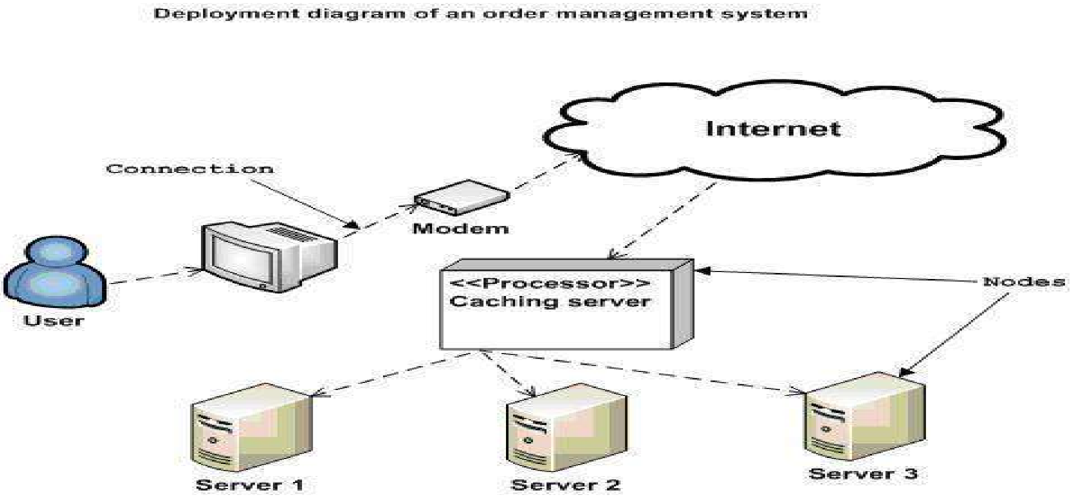
**Detailed content of the Lecture:**

- Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries, etc
- The purpose of this diagram is different. Component diagrams are used during the implementation phase of an application. However, it is prepared well in advance to visualize the implementation details.
- Initially, the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation.
- This diagram is very important as without it the application cannot be implemented efficiently. A well-prepared component diagram is also important for other aspects such as application performance, maintenance, etc.
- Before drawing a component diagram, the following artifacts are to be identified clearly −
- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.
- After identifying the artifacts, the following points need to be kept in mind.
- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing the using tools.
- Use notes for clarifying important points.
- Following is a component diagram for order management system. Here, the artifacts are files. The diagram shows the files in the application and their relationships.
- In actual, the component diagram also contains dlls, libraries, folders, etc.
- In the following diagram, four files are identified and their relationships are produced.

- Component diagram cannot be matched directly with other UML diagrams discussed so far as it is drawn for completely different purpose.
- The following component diagram has been drawn considering all the points mentioned above.



- Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=3bfJ5AORAjQ

**Important Books/Journals for further learning including the page nos.:**
Martin Fowler, UML Distilled, PHI/Pearson Education,2007[97-98]

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

L-10

## LECTURE HANDOUTS

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture:** GRASP: Designing objects with responsibilities

**Introduction :**

- GRASP - General Responsibility Assignment Software Patterns or Principles, abbreviated GRASP, consist of guidelines for assigning responsibility to classes and objects in object-oriented design.
- It is not related to the SOLID design principle.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Assigning responsibilities
- Aspects of object design

**Detailed content of the Lecture:**

Five GRASP patterns:

- Creator
- Information Expert
- Low Coupling
- Controller

**Creator**

- The concept of composition (Composite aggregates Part, Container contains Content, and Recorder records)
- Expert pattern: initializing data is passed in during creation via some kind of initialization method, such as a java constructor that has parameters.
- Assume that a payment instance, when created, needs to be initialized with the sale total. Since sale knows the total, sale is a candidate creator of the payment.

**Information Expert**

- Assign a responsibility to the information expert the class that has the information necessary to fulfill the responsibility.
- To create the interaction diagrams in order to assign responsibilities to objects.
- To fulfill the responsibility of knowing and answering the sale's total.
- Assign three responsibilities to three design classes of objects in the interaction diagram.
- Summarize the methods in the method section of a class diagram

**Low Coupling**

- Assign a responsibility so that coupling remains low. Use this principle to evaluate alternatives

– Coupling

- An element with low coupling is not dependent on too many other classes, subsystems, systems.

High coupling problems:

- Forced local changes because of changes in related classes.
- Harder to understand in isolation.
- Harder to reuse because its use requires the additional presence of the classes on which it is dependent.

**Controller**

- A controller is the first object beyond the UI layer that is responsible for receiving or handling a system operation message.
- System operations were first explored during the analysis of SSD (next page).
- These are the major input events upon our system.

- e.g., – When a cashier using a POS terminal presses the "end sale" button , he is generating a system event indicating "the sale has ended."

- When a writer using a word processor presses the "spell check" button, he is generating a system event indicating "perform a spell check.

**Video Content / Details of website for further learning (if any):**

https://slideplayer.com/slide/9445997/

**Important Books/Journals for further learning including the page nos.:**

James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005

**Course Faculty**

**Verified by HOD**

L -11

LECTURE HANDOUTS

CSE

III / V

**Course Name with Code:** 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture : Creator**

**Introduction :**

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development.
- These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Creational
- Factory
- Behavioral

**Detailed content of the Lecture:**

In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides published a book titled **Design Patterns - Elements of Reusable Object-Oriented Software** which initiated the concept of Design Pattern in Software development.

These authors are collectively known as **Gang of Four (GOF)**. According to these authors design patterns are primarily based on the following principles of object orientated design.

- Program to an interface not an implementation
- Favor object composition over inheritance

**Usage of Design Pattern**

Design Patterns have two main usages in software development.

Common platform for developers

Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

**Best Practices**

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

**Types of Design Pattern**

As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software** , there are 23 design patterns. These patterns can be classified in three categories: Creational, Structural and behavioral patterns. We'll also discuss another category of design patterns: J2EE design patterns.

| S.N. | Pattern & Description |
|---|---|
| 1 | Creational Patterns<br>These design patterns provides way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case. |
| 2 | Structural Patterns<br>These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities. |
| 3 | Behavioral Patterns<br>These design patterns are specifically concerned with communication between objects. |
| 4 | J2EE Patterns<br>These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center. |

**Video Content / Details of website for further learning (if any):**
https://www.journaldev.com/1392/factory-design-pattern-in-java

**Important Books/Journals for further learning including the page nos.:**
James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(Autonomous )

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

L -12

## LECTURE HANDOUTS

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : II -**Design Patterns** **Date of Lecture:**

**Topic of Lecture** : Information expert

**Introduction :**

- Information Expert is a principle used to determine where to delegate responsibilities.
- These responsibilities include methods, computed fields and so on.
- Using the principle of Information Expert a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfil it, and then determine where that information is stored.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Creational
- Factory
- Behavioral

**Detailed content of the Lecture:**

**Problem definition**

- Information Expert is a principle used to determine where to delegate responsibilities.
- These responsibilities include methods, computed fields and so on.
- Using the principle of Information Expert a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfil it, and then determine where that information is stored.
- Information Expert will lead to placing the responsibility on the class with the most information required to fulfil it.

**Analysis**

- Information Expert is a basic principle of delegating responsibilities in object oriented development.
- Main idea is very simple and intuitive – objects do only those operations which are connected with contained by them informations.
- Result of using this principle are solutions in which objects do those operations which in

real world normally do somebody on representing by them real objects.

- This pattern is also analogy to the real world – it's quite natural that responsibilities are delegated to the peoples who have appropriate knowledge.

**Contraindication**

- In some situations the solution resulting from Information Expert principle is not the best one because of problems with coupling and cohesion.
- What is more, we have to be careful about not breaking layer separation principle.
- What object should write class A to the database? The most of information contains class A itself so Information Expert says us that the class A should be responsible for writing itself to database.
- But this new responsibilities would decrease cohesion and break layer separation rule.

**Advantages**

- The rule of encapsulation is fulfilled – the objects do operations on the basis of contained information's.
- This usually will follow to the decreasing the number of connections between objects and decreasing of coupling.
- What is more, different behaviors of the system are assigned to the different classes.

**Video Content / Details of website for further learning (if any):**
https://www.journaldev.com/1392/factory-design-pattern-in-java

**Important Books/Journals for further learning including the page nos.:**
James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

**(An Autonomous Institution)**

**(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)**
**Rasipuram - 637 408, Namakkal Dist., Tamil Nadu**

**L -13**

**LECTURE HANDOUTS**

**CSE**

**III / V**

**Course Name with Code:** 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture :** Information expert

**Introduction :**

- Information Expert is a principle used to determine where to delegate responsibilities.
- These responsibilities include methods, computed fields and so on.
- Using the principle of Information Expert a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfil it, and then determine where that information is stored.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Creational
- Factory
- Behavioral

**Detailed content of the Lecture:**

**Problem definition**

- Information Expert is a principle used to determine where to delegate responsibilities.
- These responsibilities include methods, computed fields and so on.
- Using the principle of Information Expert a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfil it, and then determine where that information is stored.
- Information Expert will lead to placing the responsibility on the class with the most information required to fulfil it.

**Analysis**

- Information Expert is a basic principle of delegating responsibilities in object oriented development.
- Main idea is very simple and intuitive – objects do only those operations which are connected with contained by them informations.
- Result of using this principle are solutions in which objects do those operations which in

real world normally do somebody on representing by them real objects.

- This pattern is also analogy to the real world – it's quite natural that responsibilities are delegated to the peoples who have appropriate knowledge.

**Contraindication**

- In some situations the solution resulting from Information Expert principle is not the best one because of problems with coupling and cohesion.
- What is more, we have to be careful about not breaking layer separation principle.
- What object should write class A to the database? The most of information contains class A itself so Information Expert says us that the class A should be responsible for writing itself to database.
- But this new responsibilities would decrease cohesion and break layer separation rule.

**Advantages**

- The rule of encapsulation is fulfilled – the objects do operations on the basis of contained information's.
- This usually will follow to the decreasing the number of connections between objects and decreasing of coupling.
- What is more, different behaviors of the system are assigned to the different classes.

**Video Content / Details of website for further learning (if any):**
https://www.journaldev.com/1392/factory-design-pattern-in-java

**Important Books/Journals for further learning including the page nos.:**
James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

**(An Autonomous Institution)**

**(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)**
**Rasipuram - 637 408, Namakkal Dist., Tamil Nadu**

**L-14**

## LECTURE HANDOUTS

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture :** Controller-Design Patterns

**Introduction :**

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development.
- These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Creational
- Factory
- Behavioral

**Detailed content of the Lecture:**

- These authors are collectively known as **Gang of Four (GOF)**.
- According to these authors design patterns are primarily based on the following principles of object orientated design.
- Program to an interface not an implementation
- Favor object composition over inheritance
- Design patterns represent the best practices used by experienced object-oriented software developers.

**Usage of Design Pattern**

- Design Patterns have two main usages in software development.
- Common platform for developers
- Design patterns provide a standard terminology and are specific to particular scenario.
- For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

**Best Practices**

- Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development.
- Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

**CONTROLLER**

- The controller pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario.
- The controller is defined as the first object beyond the UI layer that receives and coordinates ("controls") a system operation.
- The controller pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario.
- A controller object is a non-user interface object responsible for receiving or handling a system event.

**Video Content / Details of website for further learning (if any):**
https://www.journaldev.com/1392/factory-design-pattern-in-java

**Important Books/Journals for further learning including the page nos.:**
James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L-15

## LECTURE HANDOUTS

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture : Creational -factory method**

**Introduction :**

- Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
- The basic form of object creation could result in design problems or in added complexity to the design.
- Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Creational
- Factory
- Behavioral

**Detailed content of the Lecture:**

- Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
- The basic form of object creation could result in design problems or in added complexity to the design.

**Creational design patterns**

- Creational design patterns are concerned with **the way of creating objects.**
- These design patterns are used when a decision must be made at the time of instantiation of a class (i.e. creating an object of a class).
- But everyone knows an object is created by using new keyword in java. For example:

```
StudentRecord s1=new StudentRecord();
```

- Hard-Coded code is not the good programming approach.
- Here, we are creating the instance by using the new keyword.
- Sometimes, the nature of the object must be changed according to the nature of the program.
- In such cases, we must get the help of creational design patterns to provide more general an flexible approach
- .

**Types of creational design patterns**

There are following 6 types of creational design patterns.

1. Factory Method Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern
6. Object Pool Pattern

## FACTORY METHOD

- **Factory Method** is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

## FACTORY METHOD PATTERN

- A Factory Pattern or Factory Method Pattern says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.
- In other words, subclasses are responsible to create the instance of the class.

The Factory Method Pattern is also known as Virtual Constructor.

## ADVANTAGE OF FACTORY DESIGN PATTERN

- Factory Method Pattern allows the sub-classes to choose the type of objects to create.
- It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

## USAGE OF FACTORY DESIGN PATTERN

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.

**Video Content / Details of website for further learning (if any):**
https://www.journaldev.com/1392/factory-design-pattern-in-java

**Important Books/Journals for further learning including the page nos.:**
James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L-16**

**CSE**

**II / III**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture:** GRASP-Structural, Bridge

**Introduction :**

- Bridge pattern decouple an abstraction from its implementation so that the two can vary independently.
- Introduce a class to convert the interface of one component into another interface is called adapter.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

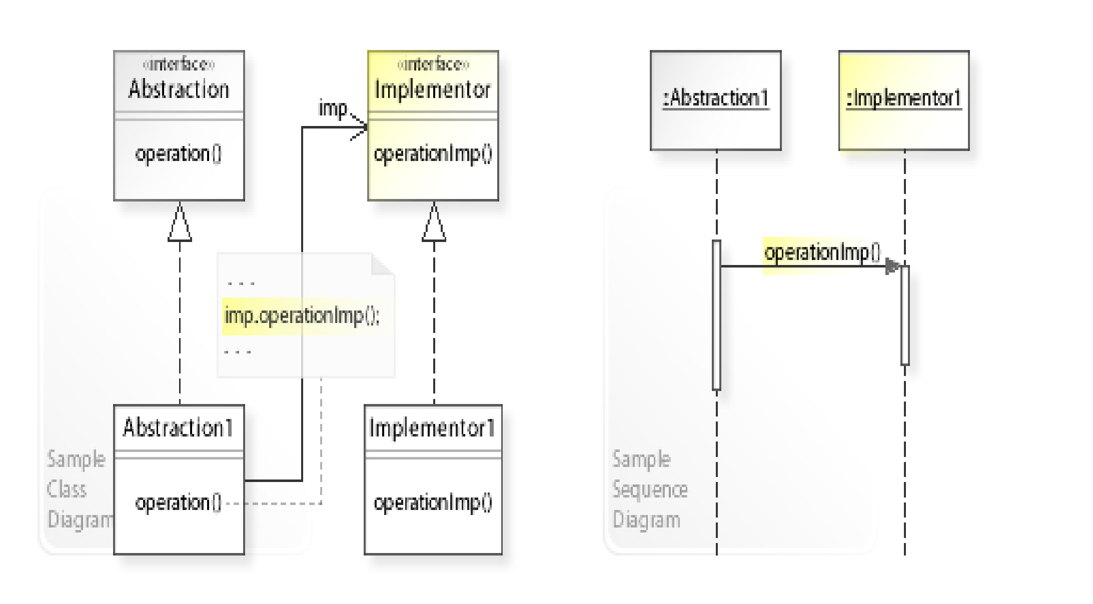
- Software system blueprint
- Depict structures and relationship in complex object

**Detailed content of the Lecture:**

**Bridge pattern**

- The bridge pattern is a design pattern used in software engineering that is meant to decouple an abstraction from its implementation so that the two can vary independently, introduced by the Gang of Four. The bridge uses encapsulation, aggregation, and can use inheritance to separate responsibilities into different classes.
- When a class varies often, the features of object-oriented programming become very useful because changes to a program's code can be made easily with minimal prior knowledge about the program.
- The bridge pattern is useful when both the class and what it does very often.
- The class itself can be thought of as the abstraction and what the class can do as the implementation.
- The bridge pattern can also be thought of as two layers of abstraction.
- When there is only one fixed implementation, this pattern is known as the Pimpl idiom in the C++ world.
- The bridge pattern is often confused with the adapter pattern, and is often implemented using the object adapter pattern, e.g. in the Java code below.
- Variant: The implementation can be decoupled even more by deferring the presence of the implementation to the point where the abstraction is utilized.

**Structure**



Sample UML class and sequence diagram for the Bridge design pattern

- The above Unified Modeling Language class diagram, an abstraction isn't implemented as usual in a single inheritance hierarchy.
- Instead, there is one hierarchy for an abstraction and a separate hierarchy for its implementation , which makes the two independent from each other.
- The operation() interface is implemented in terms of by delegating to the interface (impOperationImp()).

  The UML sequence diagram shows the run-time interactions: The implementation1 object delegates implementation to the implementor1 object by calling operationImp() on Implementor1, which performs the operation and returns to abstraction1.

**Video Content / Details of website for further learning (if any):**
https://practice.geeksforgeeks.org/courses/design-patterns

**Important Books/Journals for further learning including the page nos.:**

James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

**LECTURE HANDOUTS**

**L-17**

**CSE**

**II / III**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **II- Design Patterns** **Date of Lecture:**

**Topic of Lecture:** Adapter, Behavioral

**Introduction :**

- Introduce a class to convert the interface of one component into another interface is called adapter.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Software system blueprint
- Depict structures and relationship in complex object

**Detailed content of the Lecture:**

**ADAPTER**

- In software engineering, the adapter pattern is a software design pattern also known as wrapper, an alternative naming shared with the decorator pattern that allows the interface of an existing class to be used as another interface.
- It is often used to make existing classes work with others without modifying their source code.
- An adapter allows two incompatible interfaces to work together.
- This is the real-world definition for an adapter.
- Interfaces may be incompatible, but the inner functionality should suit the need.
- The adapter design pattern allows otherwise incompatible classes to work together by converting the interface of one class into an interface expected by the clients.
- An adapter can be used when the wrapper must respect a particular interface and must support polymorphic behavior.
- Alternatively, a decorator makes it possible to add or alter behavior of an interface at run-time, and a facade is used when an easier or simpler interface to an underlying object is desired.

**BEHAVIORAL**

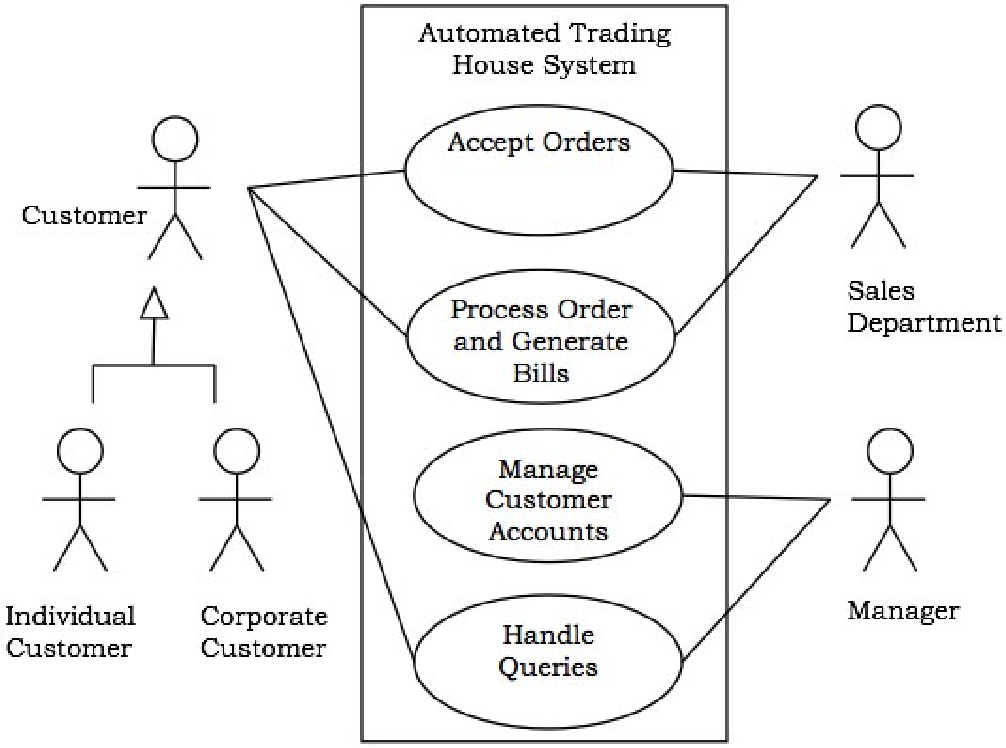
- UML behavioral diagrams visualize, specify, construct, and document the dynamic aspects of a system.

- The behavioral diagrams are categorized as follows:
  - use case diagrams,
  - interaction diagrams
  - state–chart diagrams
  - activity diagrams

1. Interactions Terms and Concepts Modeling Techniques
2. Interaction Diagrams Terms and Concepts Modeling Techniques

**USE CASE DIAGRAMS**

- Use case diagrams present an outside view of the manner the elements in a system behave and how they can be used in the context.
- Use case diagrams comprise of −
- Use cases
- Actors
- Relationships like dependency, generalization, and association



**Video Content / Details of website for further learning (if any):**
[https://practice.geeksforgeeks.org/courses/design-patterns](https://practice.geeksforgeeks.org/courses/design-patterns)

**Important Books/Journals for further learning including the page nos.:**

James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

Estd. 2000

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L-18

## LECTURE HANDOUTS

CSE

**Course Name with Code : 16CSD06** & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty :**

**Year / Semester/Section :** III / V/ A

**Unit : II- Design Patterns** **Date of Lecture:**

**Topic of Lecture: Grasp- Strategy- observer**

**Introduction :**

- Strategy lets the algorithm vary independently from clients that use it.
- Observer defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Observer pattern
- Strategy pattern

**Detailed content of the Lecture:**

**Observer Pattern**

- The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
- It is mainly used to implement distributed event handling systems, in "event driven" software. In those systems, the subject is usually called a "stream of events" or "stream source of events", while the observers are called "sink of events".
- The stream nomenclature simulates or adapts to a physical setup where the observers are physically separated and have no control over the emitted events of the subject/stream-source.
- This pattern then perfectly suits any process where data arrives through I/O, that is, where data is not available to the CPU at startup, but can arrive "randomly" (HTTP requests, GPIO data, user input from keyboard or mouse, distributed databases and blockchains .
- Most modern languages have built-in "event" constructs which implement the observer pattern components.
- While not mandatory most 'observers' implementations will use background threads listening for subject events and other support mechanism from the kernel Linux epoll.
- It addresses following problems:

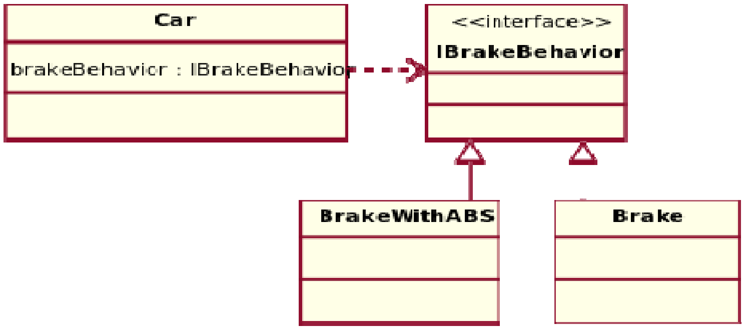
- A one-to-many dependency between objects should be defined without making the objects tightly coupled.
- It should be ensured that when one object changes state an open-ended number of dependent

objects are updated automatically.

- It should be possible that one object can notify an open-ended number of other objects.
- The Observer design pattern is one of the twenty-three well-known "Gang of Four" design patterns that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse.

**Strategy Pattern**

- The strategy pattern (also known as the policy pattern) is a behavioral software design pattern that enables selecting an algorithm at runtime.
- Instead of implementing a single algorithm directly, code receives run-time instructions as to which in a family of algorithms to use.
- For instance, a class that performs validation on incoming data may use the strategy pattern to select a validation algorithm depending on the type of data, the source of the data, user choice, or other discriminating factors.
- These factors are not known until run-time and may require radically different validation to be performed.
- The validation algorithms (strategies), encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.
- This is compatible with the open/closed principle (OCP), which proposes that classes should be open for extension but closed for modification.



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=_BpmfnqjgzQ

**Important Books/Journals for further learning including the page nos.:**
James Rumbaugh Ivar Jacobson Grady Booch, The Unified Modelling Language Reference Manual, Addison Wesley,2005.

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L -19

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture: Case study – the Next Gen POS system**

**Introduction :**

- A POS system is a computerized application used (in part) to record sales and handle payments; it is typically used in a retail store.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Next Gen POS System

**Detailed content of the Lecture:**

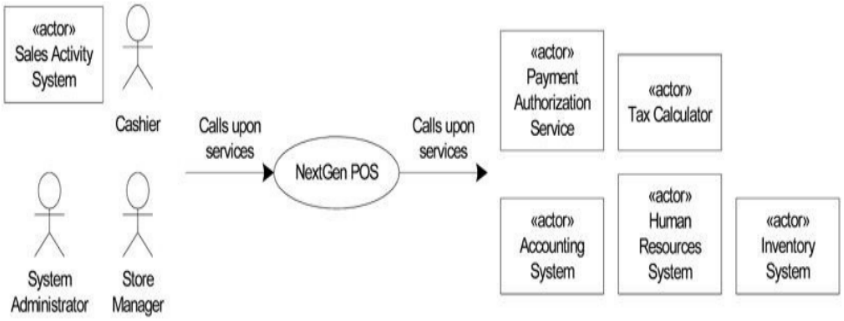
**Case Study: Next POS System**

- In this apparently straightforward problem domain, we shall see that there are very interesting requirement and design problems to solve. In addition, it is a realistic problem; organizations really do write POS systems using object technologies.
- A POS system is a computerized application used (in part) to record sales and handle payments; it is typically used in a retail store.
- It includes hardware components such as a computer and bar code scanner, and software to run the system.
- It interfaces to various service applications, such as a third-party tax calculator and inventory control.

**User-Level Goals**

The users (and external systems) need a system to fulfill these goals:

- Cashier: process sales, handle returns, cash in, cash out
- System administrator: manage users, manage security, manage system tables
- Manager: start up, shut down
- Sales activity system: analyze sales data



**Architectural Layers and Case Study Emphasis**

A typical object-oriented information system is designed in terms of several architectural layers or subsystems.

- **User Interface**—graphical interface; windows.
- **Application Logic and Domain Objects**—software objects representing domain concepts (for example, a software class named Sale) that fulfill application requirements.
- **Technical Services**—general purpose objects and subsystems that provide supporting technical services, such as interfacing with a database or error logging.

  These services are usually application-independent and reusable across several systems
  - OOA/D is generally most relevant for modeling the application logic and technical service layers.
  - The Next Gen case study primarily emphasizes the problem domain objects, allocating responsibilities to them to fulfill the requirements of the application.
  - Object-oriented design is also applied to create a technical service subsystem for interfacing with a database.
  - In this design approach, the UI layer has very little responsibility; it is said to be thin. Windows do not contain code that performs application logic or processing. Rather, task requests are forwarded on to other layers.

**Video Content / Details of website for further learning (if any):**
https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/case-studythe-next-gen-pos-system/

**Important Books/Journals for further learning including the page nos.:**

1. UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
2. Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

**LECTURE HANDOUTS**

**L-20**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Inception -Use case Modeling

**Introduction :**

- Inception is about understanding the project scope and objectives and getting enough information to confirm that the project should proceed - or to convince you that it should not.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Phases of application workflow

**Detailed content of the Lecture:**

**Objectives:**

- **Understand what to build:** Determine an overall vision, including the scope of the system and its boundaries. Identify the stakeholders: who is interested in this system and what their success criteria are.
- **Identify key system functionality:** Decide which requirements are most critical.
- **Determine at least one possible solution:** Assess whether the vision is technically feasible. This may involve identifying a candidate high-level architecture or doing technical prototypes, or both.
- **Understand** the high-level estimate for cost, schedule, and risks associated with the project.

**Key considerations:**

Projects may have one or more iterations in the Inception phase. These are among the reasons for multiple iterations:

- Project is large, and it is hard to define its scope
- Unprecedented system
- Too many stakeholders with competing needs and complex relationships
- Major technical risks demand the creation of a prototype or proof of concept

**Goals:**

- To describe the initial requirements
- To develop and justify the business case for the system
- To determine the scope of your system
- To identify the people, organizations, and external systems that will interact with your

system
- To develop an initial risk assessment, schedule, and estimate for your system
- To develop an initial tailoring of the Unified Process to meet your exact needs

**The essential activities of the inception phase are:**

- Formulating the scope of the project. This involves capturing the context and the most important requirements and constraints to such an extent that you can derive acceptance criteria for the end product.
- Planning and preparing a business case. Evaluating alternatives for risk management, staffing, project plan, and cost/schedule/profitability trade-offs.
- Synthesizing a candidate architecture, evaluating trade-offs in design, and in make/buy/reuse, so that cost, schedule and resources can be estimated.

**The outcome of the inception phase is:**

- A vision document: a general vision of the core project's requirements, key features, main constraints.
- The use-case model survey (identifying all use cases that can be identified at this early stage).
- An initial glossary.
- An initial business case, which includes:.
    - Success criteria (revenue projection, market recognition, and so on).
    - Financial forecast.
    - Business context
- An initial risk assessment.
- A project plan, showing phases, and iterations.

**Video Content / Details of website for further learning (if any):**
https://www.inceptiondesigns.com/CASE_HD_FOAM_p/gcl-005-cf.htm

**Important Books/Journals for further learning including the page nos.:**

- UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
- Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

**(An Autonomous Institution)**

**(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)**
**Rasipuram - 637 408, Namakkal Dist., Tamil Nadu**

Estd. 2000

**LECTURE HANDOUTS**

**L-21**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Relating Use cases

**Introduction :**

- An extend dependency, formerly called an extends relationship in UML v1. 2 and earlier, is a generalization relationship where an extending use case continues the behavior of a base use case.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

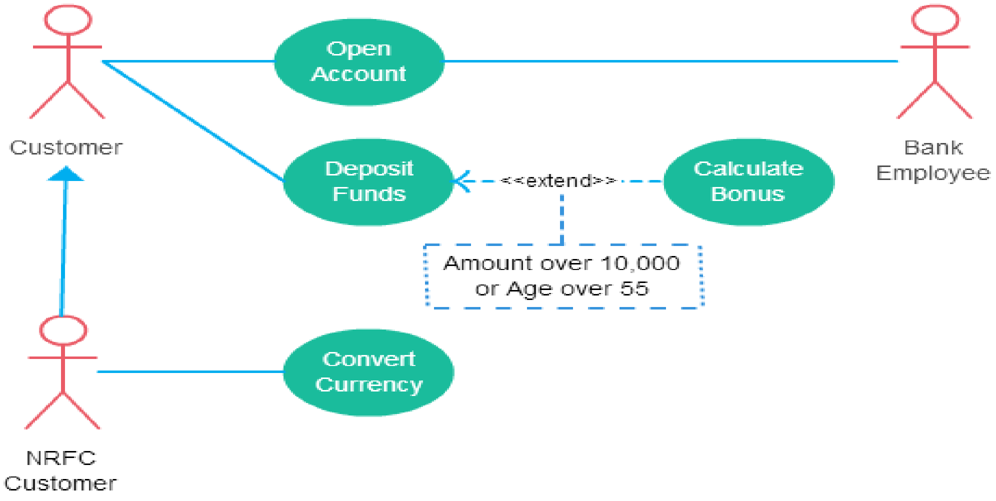
- Phases of application workflow
- Include
- Extend
- Generalization

**Detailed content of the Lecture:**

**Extend Relationship Between Two Use Cases**

Many people confuse the extend relationship in use cases. As the name implies it extends the base use case and adds more functionality to the system. Here are a few things to consider when using the <<extend>> relationship.

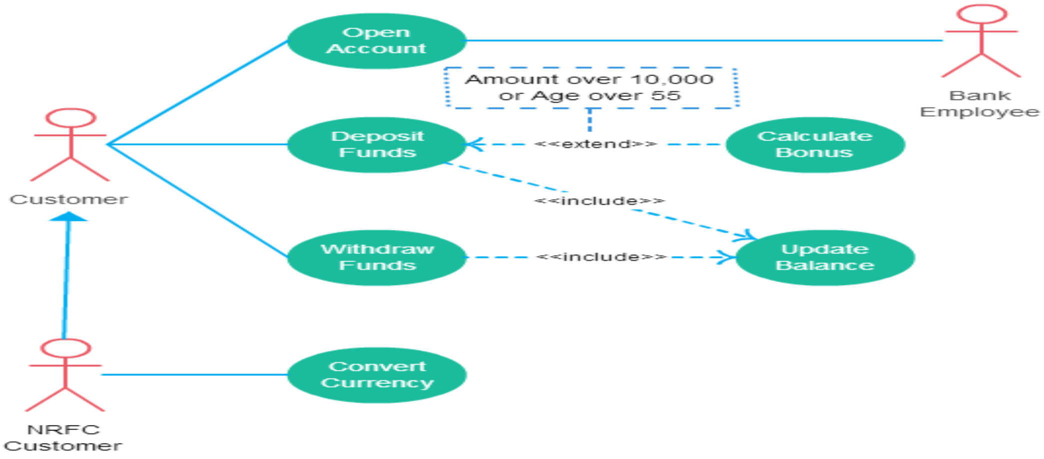
- The extending use case is dependent on the extended (base) use case. In the below diagram the "Calculate Bonus" use case doesn't make much sense without the "Deposit Funds" use case.
- The extending use case is usually optional and can be triggered conditionally. In the diagram, you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55.
- The extended (base) use case must be meaningful on its own. This means it should be independent and must not rely on the behavior of the extending use case.



### Include Relationship Between Two Use Cases

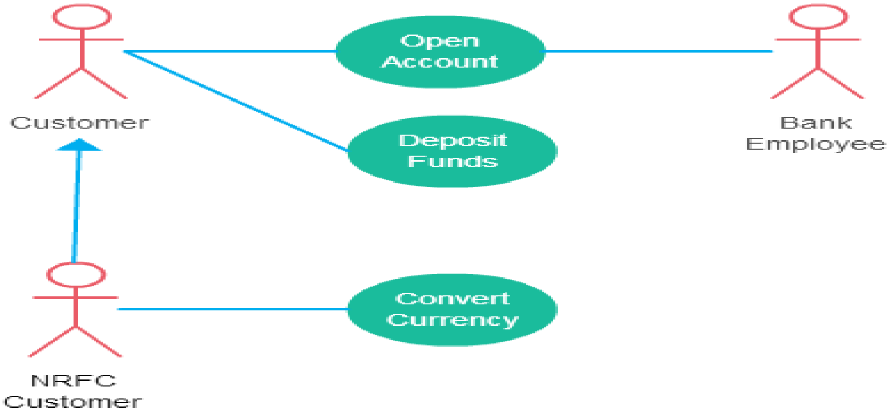
Include relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse common actions across multiple use cases. In some situations, this is done to simplify complex behaviors. Few things to consider when using the <<include>> relationship.

- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.



### Generalization of a Use Case

- This is similar to the generalization of an actor. The behavior of the ancestor is inherited by the descendant.
- This is used when there is common behavior between two use cases and also specialized behavior specific to each use case.
- Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor.



**Video Content / Details of website for further learning (if any):**
https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/relating-use-cases-include-extend-and-generalization/

**Important Books/Journals for further learning including the page nos.:**

- UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
- Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

.

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L -22

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Include, Extend and Generalization

**Introduction :**

- An extend dependency, formerly called an extends relationship in UML v1. 2 and earlier, is a generalization relationship where an extending use case continues the behavior of a base use case.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

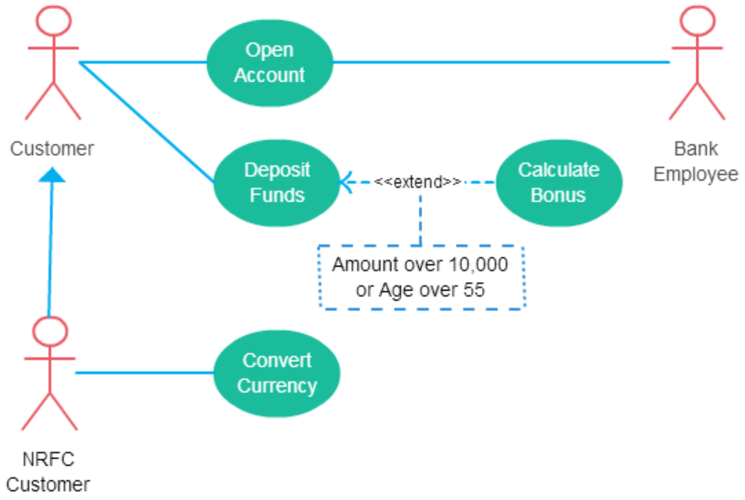
- Relating Use cases
- Include
- Extend
- Generalization

**Detailed content of the Lecture:**

**Extend Relationship Between Two Use Cases**

Many people confuse the extend relationship in use cases. As the name implies it extends the base use case and adds more functionality to the system. Here are a few things to consider when using the <<extend>> relationship.

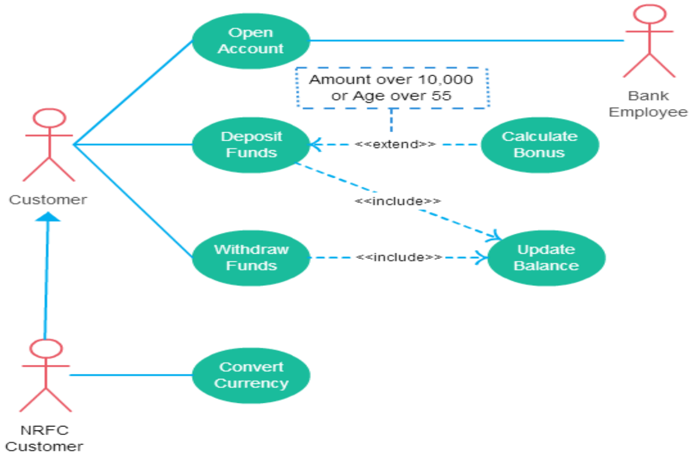
- The extending use case is dependent on the extended (base) use case. In the below diagram the “Calculate Bonus” use case doesn’t make much sense without the “Deposit Funds” use case.
- The extending use case is usually optional and can be triggered conditionally. In the diagram, you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55.
- The extended (base) use case must be meaningful on its own. This means it should be independent and must not rely on the behavior of the extending use case.



### Include Relationship Between Two Use Cases

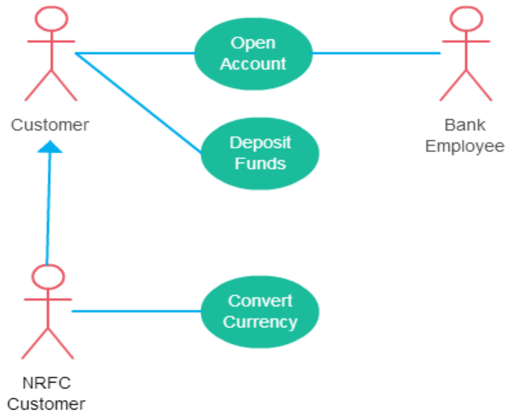
Include relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse common actions across multiple use cases. In some situations, this is done to simplify complex behaviors. Few things to consider when using the <<include>> relationship.

- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.



### Generalization of a Use Case

- This is similar to the generalization of an actor. The behavior of the ancestor is inherited by the descendant.
- This is used when there is common behavior between two use cases and also specialized behavior specific to each use case.
- Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor.
- The descendant has one or more use cases that are specific to that role. Let’s expand the previous use case diagram to show the generalization of an actor.



**Video Content / Details of website for further learning (if any):**
https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/relating-use-cases-include-extend-and-generalization/

**Important Books/Journals for further learning including the page nos.:**

- UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
- Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

L -23

## LECTURE HANDOUTS

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture: Elaboration Domain Models- Finding conceptual classes and description classe**s

**Introduction :**

- Conceptual class hierarchies are often inspiration for software class hierarchies that exploits inheritance and reduce duplication of code.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Domain Model

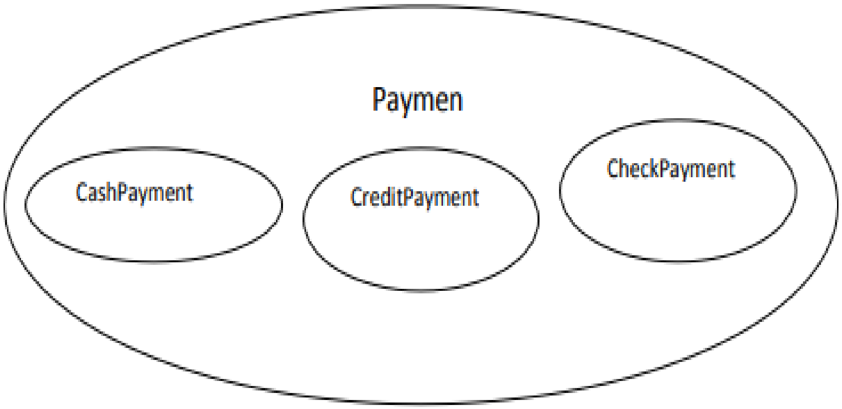
**Detailed content of the Lecture:**

**Uses of defining conceptual super classes and subclasses**

- Defining is valuable to identify conceptual super and subclasses, it is useful to clearly and precisely understand generalization, super classes, and subclasses in terms of class definition and class sets.

**Role of conceptual subclass and super classes in set membership**

- Conceptual subclasses and super classes are related in terms of set membership.
- By definition, all members of a conceptual subclass set are members of their super class set.
- For example, in terms of set membership, all instances of the set Credit Payment are also members of the set Payment.



**100% rule**

100% of the conceptual Super class's definition should be applicable to the subclass. The subclass must conform to 100% of the Super class's attributes and associations.

**Guidelines followed in defining a super class**

Create a super class in a generalization relationship to subclasses when :

- The potential conceptual subclasses represent variations of a similar concept
- The subclasses will confirm to the 100% and Is-A rules
- All subclasses have the same attribute that can be factored out and expressed in the super class
- All subclasses have the same association that can be factored out and related to the super class

**Strong motivations to partition a conceptual class with subclasses**

Create a conceptual subclass of a super class when :

- The subclass has additional attributes of interest
- The subclass has additional associations of interest
- The subclass concept is operated on, handled, reacted-to, or manipulated differently than the super class or other subclasses

**Video Content / Details of website for further learning (if any):**
https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/finding-conceptual-class-hierarchies-in-ooad/

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

**LECTURE HANDOUTS**

**L -24**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Associations – Attributes

**Introduction :**

- An association indicates that the system you are developing stores links of some kind between the instances of the associated types.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Domain Model
- Associations
- Attributes

**Detailed content of the Lecture:**

**Association**

- Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes.
- A link can be defined as an instance of an association.

**Degree of an Association**

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

**Cardinality Ratios of Associations**

- 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.

**Attributes**

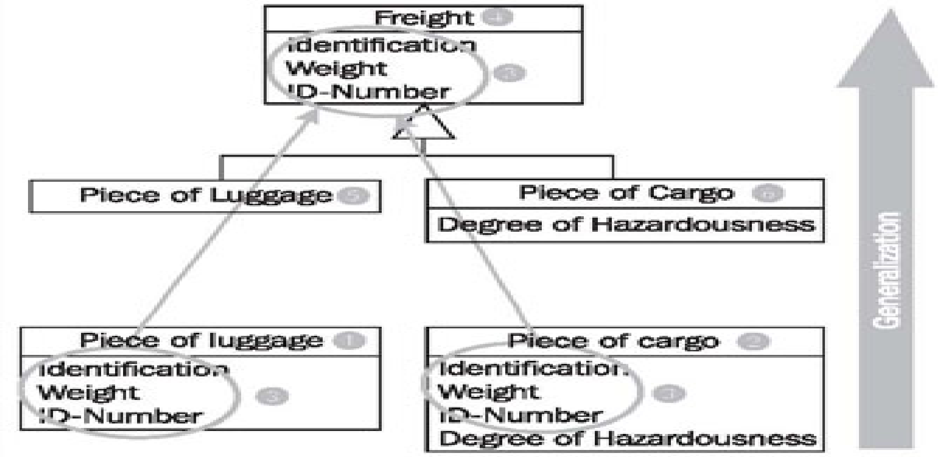
- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

**Domain Model Refinement**

A domain model contains conceptual classes, associations between conceptual classes, and attributes of a conceptual class. "Informally, a conceptual class is an idea, thing, or object".

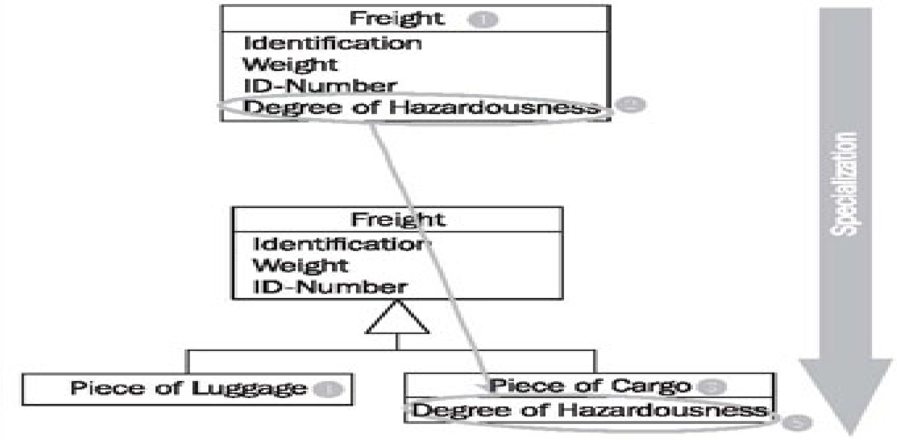
**Generalization**

Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized super class. Shared characteristics can be attributes, associations, or methods.



**Specialization**

Specialization means creating new subclasses from an existing class. If it turns out that certain attributes, associations, or methods only apply to some of the objects of the class, a subclass can be created. The most inclusive class in a generalization/specialization is called the super class and is generally located at the top of the diagram. The more specific classes are called subclasses and are generally placed below the super class.



**Video Content / Details of website for further learning (if any):**

[https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/finding-conceptual-classes-and-description-classes-in-ooad/](https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/finding-conceptual-classes-and-description-classes-in-ooad/)

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L -25**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Domain model refinement

**Introduction :**

- A domain model is illustrated with a set of class diagrams in which no operations (method signatures) are defined

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Associations
- Attributes
- Domain Model

**Detailed content of the Lecture:**

- A domain model is illustrated with a set of class diagrams in which no operations (method signatures) are defined

**Association**

- Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes.
- A link can be defined as an instance of an association.

**Degree of an Association**

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

**Cardinality Ratios of Associations**

- 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.

**Attributes**

- A set of attributes for the objects that are to be instantiated from the class.
- Generally, different objects of a class have some difference in the values of the attributes.
- Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class.
- Operations are also referred as functions or methods.

**Domain Model Refinement**

- A domain model contains conceptual classes, associations between conceptual classes, and attributes of a conceptual class. "Informally, a conceptual class is an idea, thing, or object".

**Generalization**

- Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized super class.
- Shared characteristics can be attributes, associations, or methods.

**Specialization**

- Specialization means creating new subclasses from an existing class.
- If it turns out that certain attributes, associations, or methods only apply to some of the objects of the class, a subclass can be created.
- The most inclusive class in a generalization/specialization is called the super class and is generally located at the top of the diagram.
- The more specific classes are called subclasses and are generally placed below the super class.

**Video Content / Details of website for further learning (if any):**
https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/finding-conceptual-classes-and-description-classes-in-ooad/

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education**,** 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L -26

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Finding conceptual classes and description classes

**Introduction :**

- Conceptual class hierarchies are often inspiration for software class hierarchies that exploits inheritance and reduce duplication of code.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Domain Model

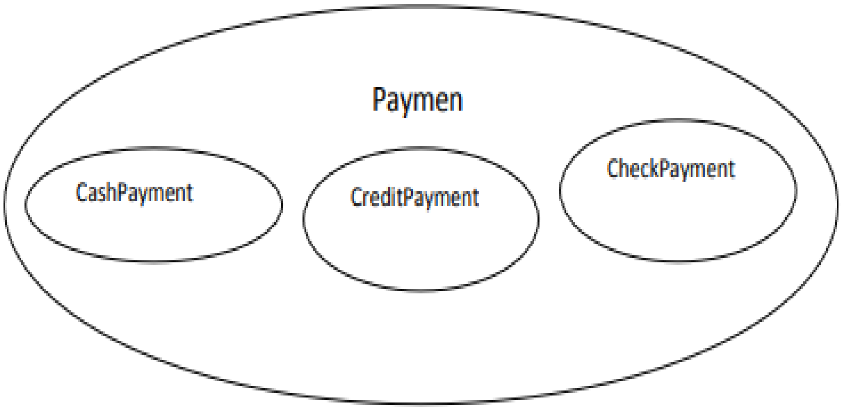
**Detailed content of the Lecture:**

**Uses of defining conceptual super classes and subclasses**

- Defining is valuable to identify conceptual super and subclasses, it is useful to clearly and precisely understand generalization, super classes, and subclasses in terms of class definition and class sets.

**Role of conceptual subclass and super classes in set membership**

- Conceptual subclasses and super classes are related in terms of set membership.
- By definition, all members of a conceptual subclass set are members of their super class set.
- For example, in terms of set membership, all instances of the set Credit Payment are also members of the set Payment.



**100% rule**

100% of the conceptual Super class's definition should be applicable to the subclass. The subclass must conform to 100% of the Super class's attributes and associations.

**Guidelines followed in defining a super class**

Create a super class in a generalization relationship to subclasses when :

- The potential conceptual subclasses represent variations of a similar concept
- The subclasses will confirm to the 100% and Is-A rules
- All subclasses have the same attribute that can be factored out and expressed in the super class
- All subclasses have the same association that can be factored out and related to the super class

**Strong motivations to partition a conceptual class with subclasses**

Create a conceptual subclass of a super class when :

- The subclass has additional attributes of interest
- The subclass has additional associations of interest
- The subclass concept is operated on, handled, reacted-to, or manipulated differently than the super class or other subclasses

**Video Content / Details of website for further learning (if any):**

https://facultytalkies.com/courses/cs8592-object-oriented-analysis-and-design-notes/lessons/finding-conceptual-class-hierarchies-in-ooad/

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

**LECTURE HANDOUTS**

**L -27**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **III - CASE STUDY** **Date of Lecture:**

**Topic of Lecture:** Aggregation and Composition

**Introduction :**

- Aggregation implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.
- Composition implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child).

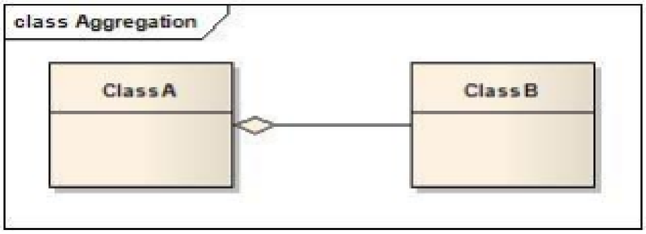
**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Conceptual class hierarchy

**Detailed content of the Lecture:**

**Aggregation (Shared Association)**

- In cases where there's a part-of relationship between Class A (whole) and Class B (part), we can be more specific and use the aggregation link instead of the association link, taking special notice that Class B can also be aggregated by other classes in the application (therefore aggregation is also known as shared association).



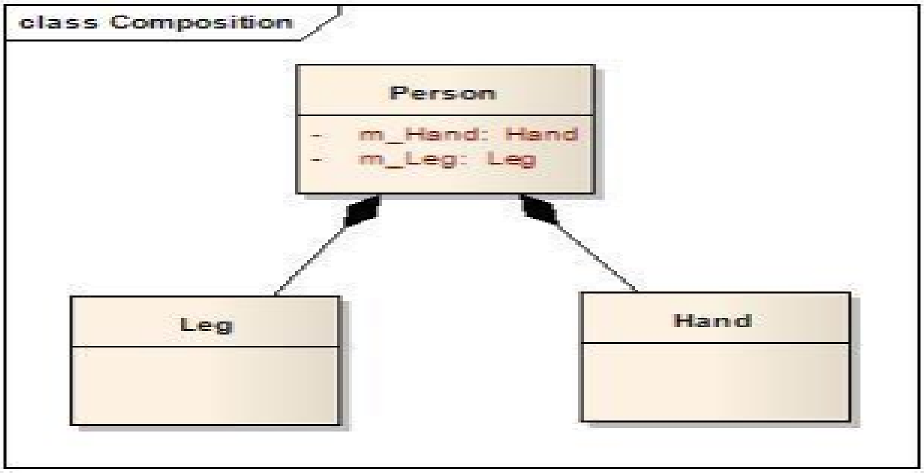
- So basically, the aggregation link doesn't state in any way that Class A owns Class B nor that there is a parent-child relationship (when parent deleted all its child's are being deleted as a result) between the two.
- Actually, quite the opposite! The aggregation link usually used to stress the point that Class A is not the exclusive container of Class B, as in fact Class B has another container.

**Composition (Not-Shared Association)**

- In cases where in addition to the part-of relationship between Class A and Class B - there's a strong life cycle dependency between the two, meaning that when Class A is deleted then Class

B is also deleted as a result, we should be more specific and use the composition link instead of the aggregation link or the association link.

- The composition link shows that a class (container, whole) has exclusive ownership over other class/s (parts), meaning that the containers object and its parts constitute a parent-child/s relationship.
- Unlike association and aggregation, in the composition relationship, the composed class cannot appear as a return type or parameter type of the composite class, thus changes in the composed class cannot be propagated to the rest of the system.
- Consequently, usage of composition limits complexity growth as the system grows.



**Significance:**

- Provides standard for software development.
- Reducing of costs to develop diagrams of UML using supporting tools.
- Development time is reduced.
- The past faced issues by the developers are no longer exists.
- Has large visual elements to construct and easy to follow.

**Video Content / Details of website for further learning (if any):**
https://www.infoworld.com/article/3029325/exploring-association-aggregation-and-composition-in-oop.html

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

**(An Autonomous Institution)**

**(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)**
**Rasipuram - 637 408, Namakkal Dist., Tamil Nadu**

**LECTURE HANDOUTS**

**L -28**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **IV- APPLYING DESIGN PATTERNS Date of Lecture:**

**Topic of Lecture:** System sequence diagrams

**Introduction :**

- System sequence diagram (SSD) is a sequence diagram that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams

**Detailed content of the Lecture:**

**Definition**

- System sequence diagrams, also known as SSD, are actually a sub-type of sequence diagrams, whose style and notation is dictated by the Unified Modeling Language.
- This language provides a toolkit for diagram creators to make and read diagrams that are comprehensible regardless of location or industry.
- Standard sequence diagrams show the progression of events over a certain amount of time, while system sequence diagrams go step further and present sequences for specific use cases.

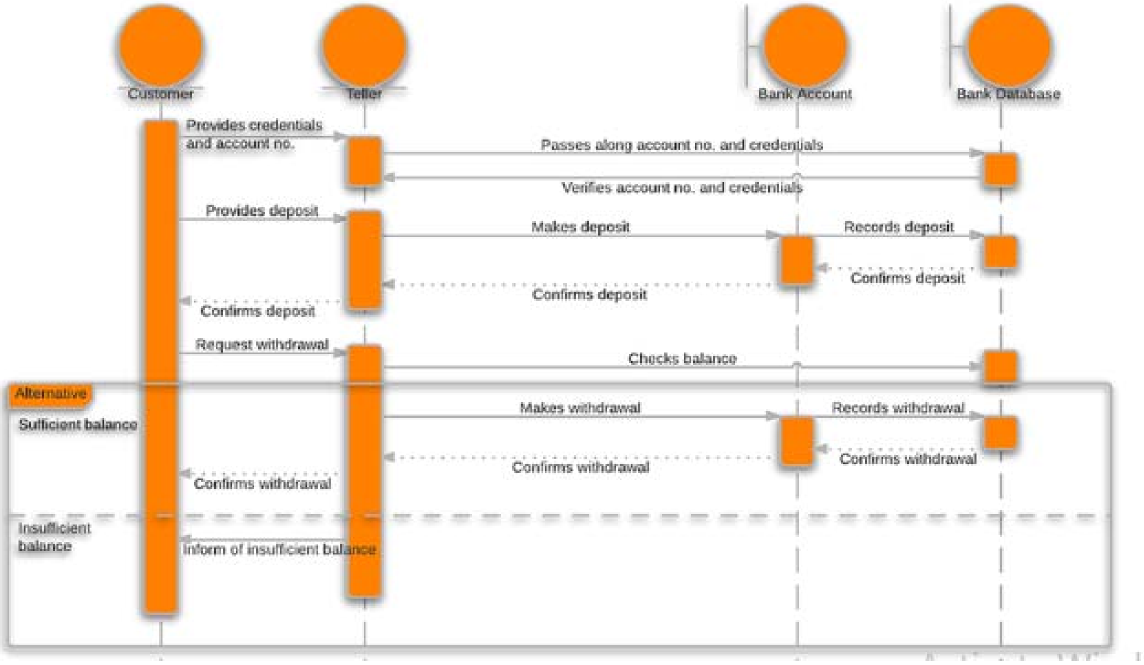
Most elements we cover in use case diagrams remain in use throughout a system sequence diagram, including:

- **Objects** - this box shape with an underlined title represents a class, or object, in UML. Within a SSD, this shape models the system as a black box (a system with inner workings that are not immediately visible).
- **Actors** - shown by stick figures, actors are entities that interact with the system, and yet are external to it.
- **Events** - the system events that the actors generate in the sequence. A dashed line, known as a lifeline, represents events in an SSD. Lifelines may begin with a labeled rectangle shape or an actor symbol.

**Benefits of system sequence diagrams**

SSDs are ideal for demonstrating when and how tasks are completed in a system, especially as those tasks relate to use cases. Here are a few specific examples of when to use SSDs:

- In a step-wise fashion, modeling operations of the system in response to events.
- Building a blueprint for the main success scenario of a given use case, then creating alternative paths.
- Identifying major system events and operations in order to come up with realistic estimates of resources needed.



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=3VX3QpUuvfs

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L -29

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **IV- APPLYING DESIGN PATTERNS Date of Lecture:**

**Topic of Lecture:** Relationship between sequence diagrams

**Introduction :**

- Sequence diagrams only specify the ordering of events and not the exact timings of events.
- An activation box represents the period during which an operation is executed.
- Shows interactions between objects in visual and chronological (time) order.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams

**Detailed content of the Lecture:**

**Use case relate to a sequence diagram**

- A use-case model is built and the actors are connected to use cases.
- Each use case represents a task in which the actor participates.
- For each use case, a sequence diagram is built.
- Each sequence diagram specifies the main interaction steps to be achieved for each task (i.e. use case).

**Describing Use Cases By Means of Sequence Diagrams**

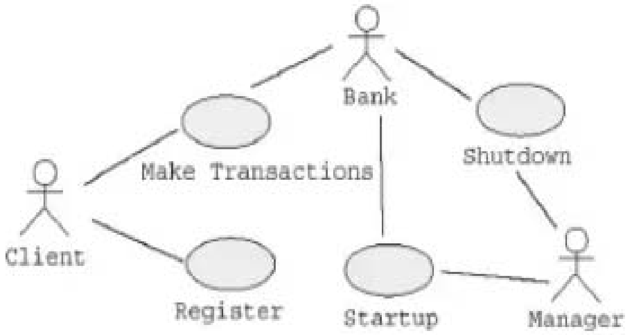
- Regarding the model-driven development, we also show a discussion on how the development process and, in general, the developer decisions affect both use-case modeling and sequence-diagram modeling, in particular, the identification of use-case relationships.

Our technique can be summarized as follows.

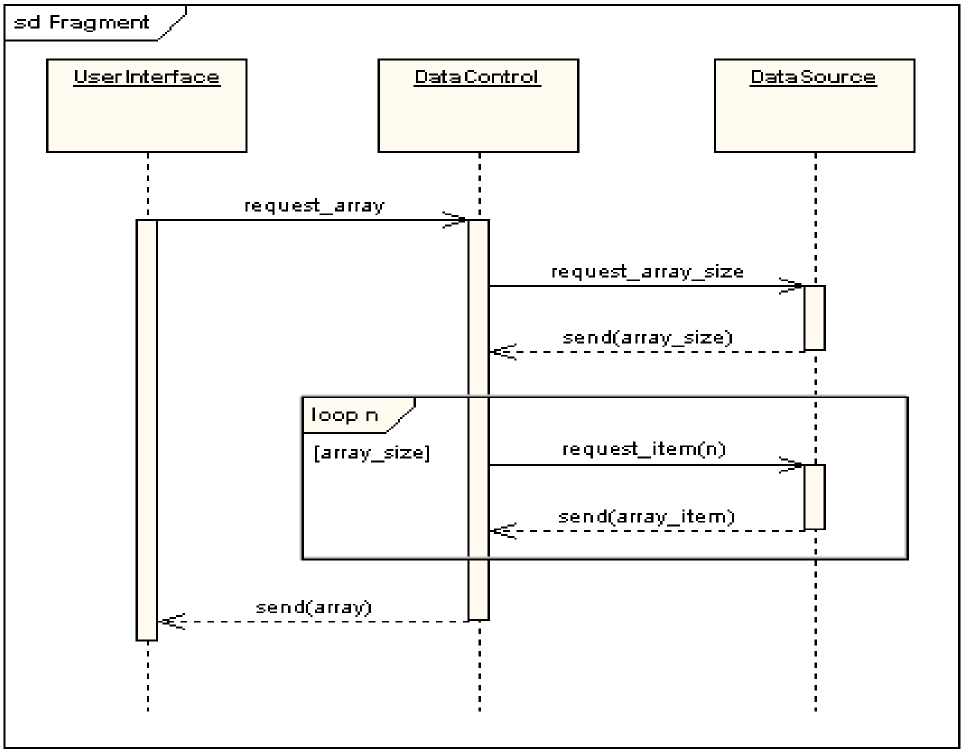
- A use-case model is built and the actors are connected to use cases. Each use case represents a task in which the actor participates.
- For each use case, a sequence diagram is built. Each sequence diagram specifies the main interaction steps to be achieved for each task (i.e. use case).
- From the sequence diagrams, use-case relationships are identified. Sequence sub diagrams are identified with new use cases.
- The sequence diagrams are refined: some interaction steps are added as extensions to the original sequence diagrams. These extensions are represented as new sequence (sub)diagrams.
- These new sub diagrams are identified with new use cases.

- From the refined sequence diagrams, new use-case relationships are discovered: new generalization/ specialization and extension relationships.

- Generalization/ specialization relationships between abstract and particular use cases are identified.
- Some of the previous steps might be applied incrementally in the development process.



**Combined Fragments**



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=3VX3QpUuvfs

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

**LECTURE HANDOUTS**

**L-30**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** :**IV APPLYING DESIGN PATTERNS Date of Lecture:**

**Topic of Lecture:** Use cases

**Introduction :**

- To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating.
- In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them.
- Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams
- Use cases

**Detailed content of the Lecture:**

- These internal and external agents are known as actors. Use case diagrams consists of actors, use cases and their relationships.
- The diagram is used to model the system/subsystem of an application.
- A single use case diagram captures a particular functionality of a system.
- Hence to model the entire system, a number of use case diagrams are used.

PURPOSE OF USE CASE DIAGRAMS

- The purpose of use case diagram is to capture the dynamic aspect of a system.
- However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and Statechart) also have the same purpose.
- We will look into some specific purpose, which will distinguish it from other four diagrams.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows −

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.

- Show the interaction among the requirements are actors.

HOW TO DRAW A USE CASE DIAGRAM:

- Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified.
- Functionalities to be represented as use case
- Actors
- Relationships among the use cases and actors.
- Use case diagrams are drawn to capture the functional requirements of a system. After identifying the above items, we have to use the following guidelines to draw an efficient use case diagram
- The name of a use case is very important. The name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.

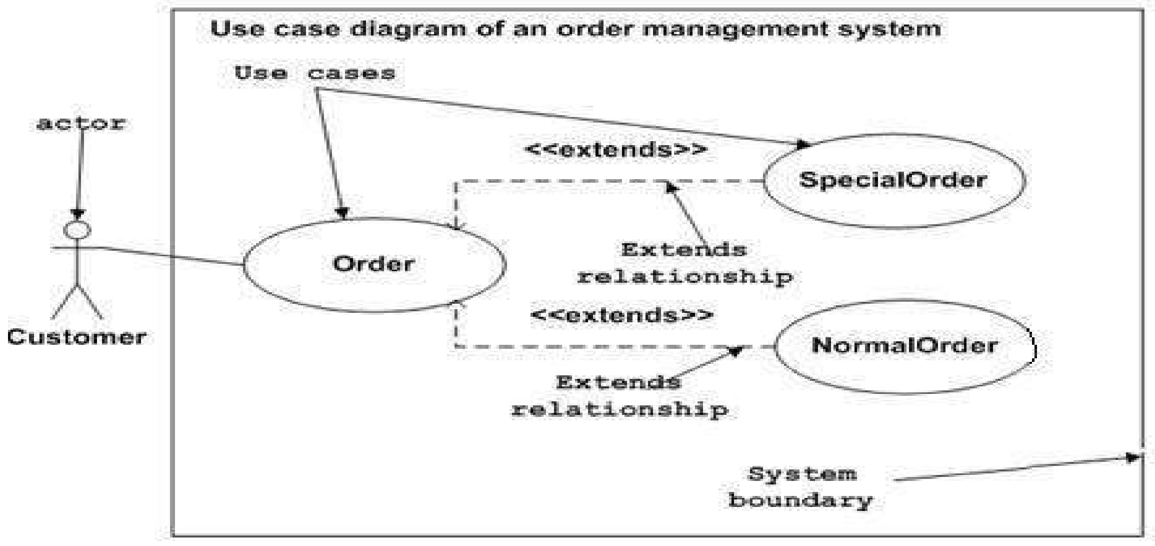


Figure: Sample Use Case diagram

Use case diagrams can be used for −

- Requirement analysis and high level design.
- Model the context of a system.
- Reverse engineering.
- Forward engineering.

**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=3VX3QpUuvfs

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

.

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

## LECTURE HANDOUTS

L-31

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **IV -APPLYING DESIGN PATTERNS** **Date of Lecture:**

**Topic of Lecture:** Logical architecture

**Introduction :**

- Logical architecture is the large-scale organization of the software classes into packages (or namespaces), subsystems, and layers.
- Logical - because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network (deployment architecture).

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams
- Logical architecture

**Detailed content of the Lecture:**

- Any real-world system is used by different users. The users can be developers, testers, business people, analysts, and many more.
- Hence, before designing a system, the architecture is made with different perspectives in mind.
- The most important part is to visualize the system from the perspective of different viewers. The better we understand the better we can build the system.

UML plays an important role in defining different perspectives of a system. These perspectives are −

- Design
- Implementation
- Process
- Deployment

The center is the Use Case view which connects all these four.

A Use Case represents the functionality of the system. Hence, other perspectives are connected with use case.

Design of a system consists of classes, interfaces, and collaboration.

UML provides class diagram, object diagram to support this.

Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

It is very important to distinguish between the UML model. Different diagrams are used for different types of UML modeling. There are three important types of UML modeling.

### STRUCTURAL MODELING

Structural modeling captures the static features of a system. They consist of the following −

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling. They all represent the elements and the mechanism to assemble them.

The structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram.

### BEHAVIORAL MODELING

Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following −

- Activity diagrams
- Interaction diagrams
- Use case diagrams

All the above show the dynamic sequence of flow in a system.

### ARCHITECTURAL MODELING

Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modeling.

**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=3VX3QpUuvfs

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

Verified by HOD

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L-32

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **IV- APPLYING DESIGN PATTERNS Date of Lecture:**

**Topic of Lecture:** UML package diagram

**Introduction :**

- Package diagram is UML structure diagram which shows structure of the designed system at the level of packages. The following elements are typically drawn in a package diagram: package, packageable element, dependency, element import, package import, package merge.

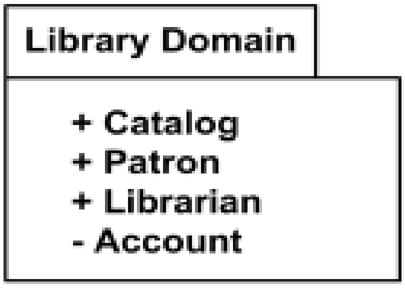
**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams

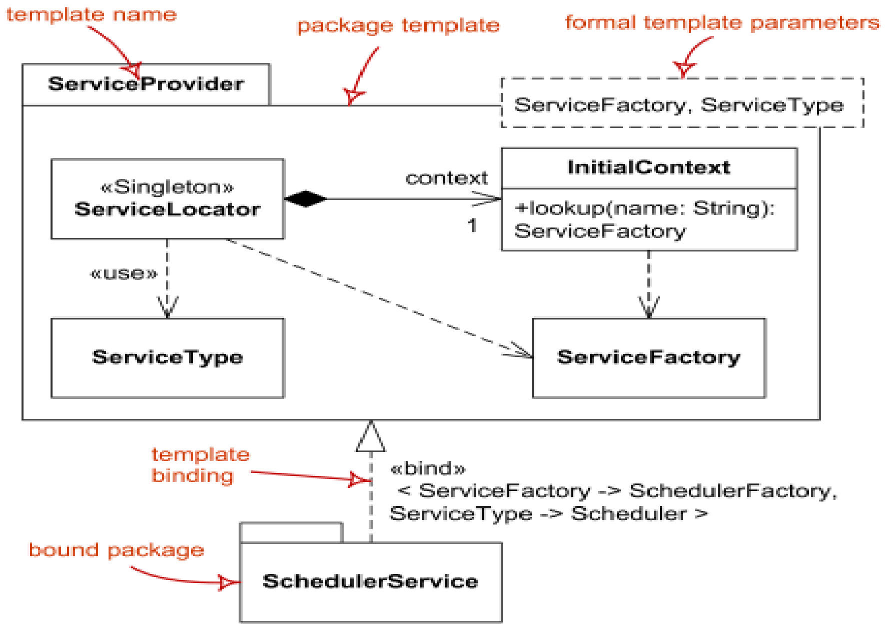
**Detailed content of the Lecture:**

**Package**

- Package is a namespace used to group together elements that are semantically related and might change together.
- It is a general purpose mechanism to organize elements into groups to provide better structure for system model.
- Owned members of a package should all be packageable elements.
- If a package is removed from a model, so are all the elements owned by the package.
- Package by itself is packageable element, so any package could be also a member of other packages.



**Package Template**



**Logical Architecture And Layers**

- Logical architecture is the large-scale organization of the software classes into packages (or namespaces), subsystems, and layers.
- Logical - because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network (deployment architecture).

**Layering Pattern**

- A layer is a very coarse-grained grouping of classes, packages, or subsystems that has cohesive responsibility for a major aspect of the system.
- Also, layers are organized such that "higher" layers (such as the UI layer) call upon services of "lower" layers, but not normally vice versa.
- Strict layered architecture VS Relaxed Layered Architecture.
- A logical architecture doesn't have to be organized in layers.
- But it's very common, and hence, introduced at this time.

**Typical Layers Typically layers in an OO system**

- User Interface.
- Application Logic and Domain Objects software objects representing domain concepts (for example, a software class Sale) that fulfill application requirements, such as calculating a sale total. .

**Video Content / Details of website for further learning (if any):**
https://medium.com/@warren2lynch/uml-what-is-package-diagram-how-to-use-it-dbd317c07d5d

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

Verified by HOD

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L -33

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **IV- APPLYING DESIGN PATTERNS** **Date of Lecture:**

**Topic of Lecture:** Logical architecture refinement

**Introduction :**

- Package diagram is UML structure diagram which shows structure of the designed system at the level of packages.
- The following elements are typically drawn in a package diagram: package, packageable element, dependency, element import, package import, package merge.

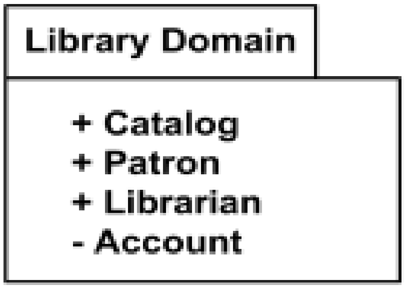
**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams
- UML package diagram

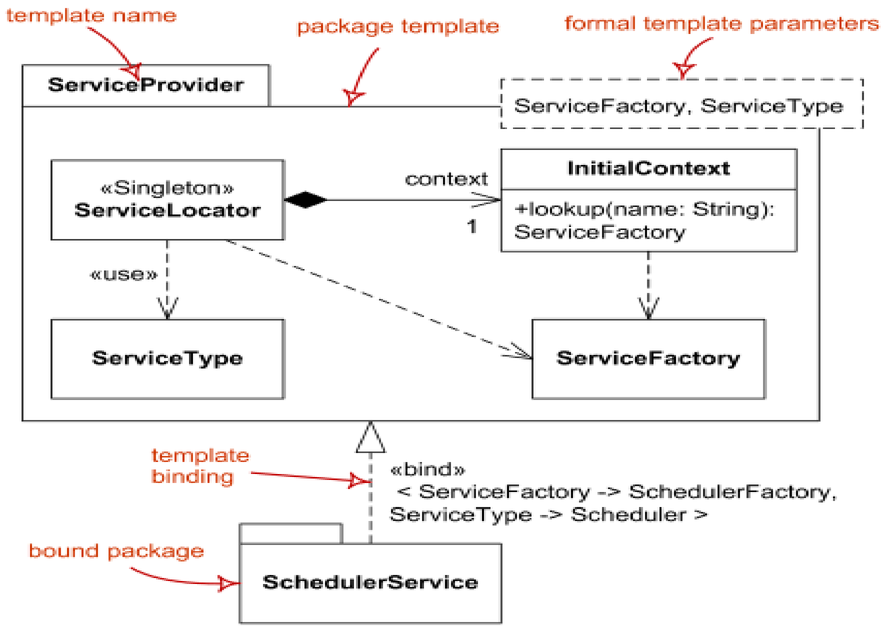
**Detailed content of the Lecture:**

**Package**

- Package is a namespace used to group together elements that are semantically related and might change together.
- It is a general purpose mechanism to organize elements into groups to provide better structure for system model.
- Owned members of a package should all be packageable elements.
- If a package is removed from a model, so are all the elements owned by the package.
- Package by itself is packageable element, so any package could be also a member of other packages.



### Package Template



### Logical Architecture And Layers

- Logical architecture is the large-scale organization of the software classes into packages (or namespaces), subsystems, and layers.
- Logical - because there's no decision about how these elements are deployed across different operating system processes or across physical computers in a network (deployment architecture).

### Layering Pattern

- A layer is a very coarse-grained grouping of classes, packages, or subsystems that has cohesive responsibility for a major aspect of the system.
- Also, layers are organized such that "higher" layers (such as the UI layer) call upon services of "lower" layers, but not normally vice versa.
- Strict layered architecture VS Relaxed Layered Architecture.
- A logical architecture doesn't have to be organized in layers.
- But it's very common, and hence, introduced at this time.

### Typical Layers Typically layers in an OO system

- User Interface.
- Application Logic and Domain Objects software objects representing domain concepts (for example, a software class Sale) that fulfill application requirements, such as calculating a sale total. .

**Video Content / Details of website for further learning (if any):**

https://medium.com/@warren2lynch/uml-what-is-package-diagram-how-to-use-it-dbd317c07d5d

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L -34

CSE

III / V

Course Name with Code : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

Course Faculty :

**Year / Semester/Section** **:** III / V/ A

Unit : IV -**APPLYING DESIGN PATTERNS** **Date of Lecture:**

**Topic of Lecture:** UML class diagrams

**Introduction :**

- Class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams

**Detailed content of the Lecture:**

**Purpose of Class Diagrams**

- The purpose of class diagram is to model the static view of an application.
- Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

**The purpose of the class diagram can be summarized as −**

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

**Where to Use Class Diagrams**

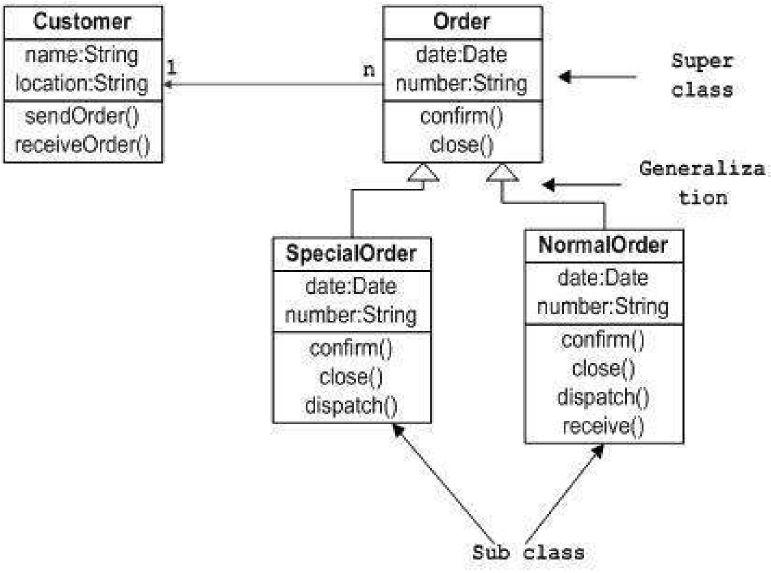
Class diagrams are used for −

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

**Draw a Class Diagram**

The following points should be remembered while drawing a class diagram −

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram.
- At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.



**Video Content / Details of website for further learning (if any):**
https://www.edx.org/course/uml-class-diagrams-for-software-engineering

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L-35**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **IV- APPLYING DESIGN PATTERNS** **Date of Lecture:**

**Topic of Lecture:** UML interaction diagrams

**Introduction :**

- Interaction Overview Diagram is one of the fourteen types of diagrams of the Unified Modeling Language, which can picture a control flow with nodes that can contain interaction diagrams.
- The interaction overview diagram is similar to the activity diagram, in that both visualize a sequence of activities.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams

**Detailed content of the Lecture:**

**Purpose of Interaction Diagrams**

- The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task.
- Hence, the solution is to use different types of models to capture the different aspects of the interaction.

**The purpose of interaction diagram is −**

- To capture the dynamic behavior of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

**Where to Use Interaction Diagrams**

Interaction diagrams can be used −

- To model the flow of control by time sequence.
- To model the flow of control by structural organizations.
- For forward engineering.
- For reverse engineering.

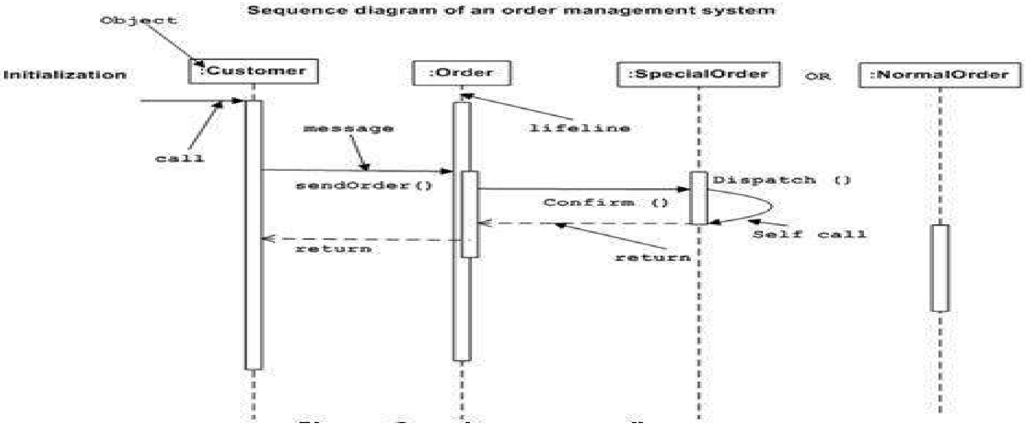
**Draw an Interaction Diagram**

Following things are to be identified clearly before drawing the interaction diagram

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

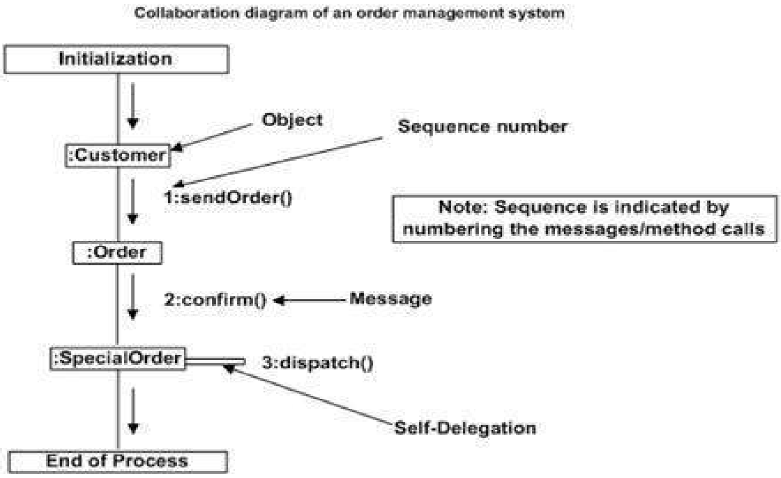
**The Sequence Diagram**

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).



**The Collaboration Diagram**

- In the collaboration diagram, the method call sequence is indicated by some numbering technique.
- The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=Ba7SyM78cUM

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education**,** 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

**(An Autonomous Institution)**

**(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)**
**Rasipuram - 637 408, Namakkal Dist., Tamil Nadu**

Estd. 2000

**LECTURE HANDOUTS**

**L -36**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **IV- APPLYING DESIGN PATTERNS** **Date of Lecture:**

**Topic of Lecture:** UML interaction diagrams

**Introduction :**

- Interaction Overview Diagram is one of the fourteen types of diagrams of the Unified Modeling Language, which can picture a control flow with nodes that can contain interaction diagrams.
- The interaction overview diagram is similar to the activity diagram, in that both visualize a sequence of activities.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- UML Diagrams

**Detailed content of the Lecture:**

**INTERACTION DIAGRAMS**

- The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system.
- However, the specific purpose is more important to clarify and understand.
- Sequence diagrams are used to capture the order of messages flowing from one object to another.
- Collaboration diagrams are used to describe the structural organization of the objects taking part in the interaction.
- A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole.
- Interaction diagrams are used when we want to understand the message flow and the structural organization.
- Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

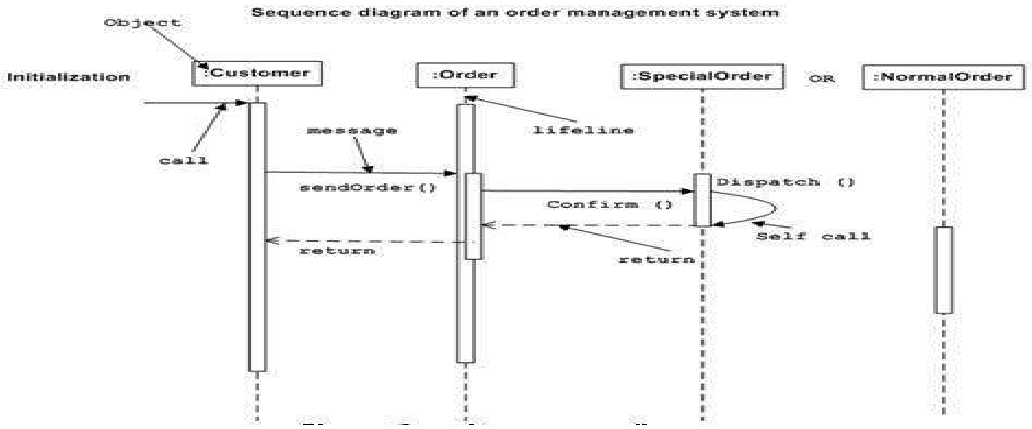
Interaction diagrams can be used −

- To model the flow of control by time sequence.

- To model the flow of control by structural organizations.
- For forward engineering.
- For reverse engineering.

The Sequence Diagram

The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).

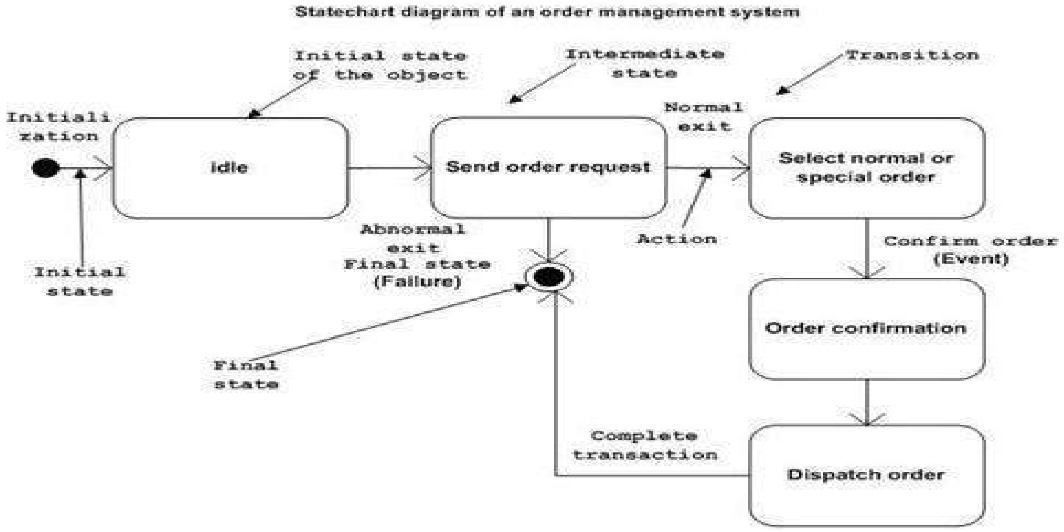


STATECHART DIAGRAM

Statechart diagrams are very important for describing the states.

Before drawing a Statechart diagram we should clarify the following points −

- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=Ba7SyM78cUM

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L-37

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** Applying GoF

**Introduction :**

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Program to an interface not an implementation
- Favor object composition over inheritance

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Design patterns
- UML Diagrams

**Detailed content of the Lecture:**

**Gang of Four (GOF)**

- In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides published a book titled Design Patterns - Elements of Reusable Object-Oriented Software which initiated the concept of Design Pattern in Software development.
- These authors are collectively known as Gang of Four (GOF).
- According to these authors design patterns are primarily based on the following principles of object orientated design.
  - Program to an interface not an implementation
  - Favor object composition over inheritance

**Usage of Design Pattern**

- Design patterns provide a standard terminology and are specific to particular scenario.
- For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.
- Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development.
- Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

**Types of Design Pattern**

**Creational Patterns-**

- These design patterns provides way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.

- This gives program more flexibility in deciding which objects need to be created for a given use case.

**Structural Patterns-**

- These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

**Behavioral Patterns-**

- These design patterns are specifically concerned with communication between objects.

**J2EE Patterns-**

- These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

**Video Content / Details of website for further learning (if any):**

https://www.linkedin.com/learning/java-ee-design-patterns-and-architecture/classic-gof-software-design-patterns

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L-38

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** Design patterns

**Introduction :**

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development.
- These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Design patterns
- UML Diagrams

**Detailed content of the Lecture:**

**Usage of Design Pattern**

- Design patterns provide a standard terminology and are specific to particular scenario.
- Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development.
- Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

**Types of Design Pattern**

**Creational Patterns-**

- These design patterns provides way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
- This gives program more flexibility in deciding which objects need to be created for a given use case.

**Structural Patterns-**

- These design patterns concern class and object composition.
- Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

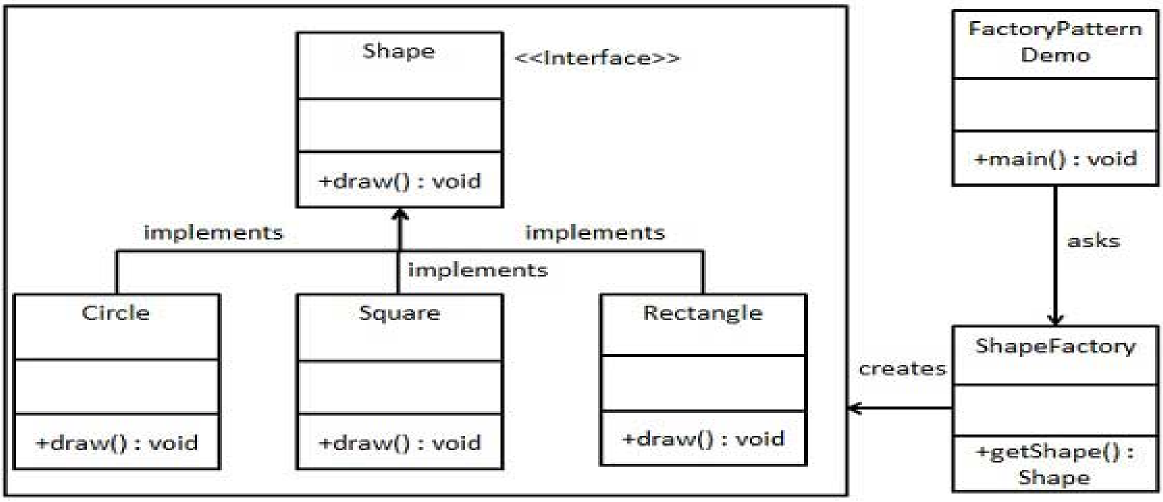
**Behavioral Patterns-**

- These design patterns are specifically concerned with communication between objects.

**J2EE Patterns-**

- These design patterns are specifically concerned with the presentation tier.
- These patterns are identified by Sun Java Center.
- Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

**IMPLEMENTATION**



**Video Content / Details of website for further learning (if any):**
https://www.linkedin.com/learning/java-ee-design-patterns-and-architecture/classic-gof-software-design-patterns

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L-39

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** Mapping design to code

**Introduction :**

- Mapping from Design to Code. Each class in design is implemented by coding it in a programming language or by using a pre-existing component

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- GoF Patterns
- UML Diagrams

**Detailed content of the Lecture:**

**Programming and Iterative, Evolutionary Development**

- The creation of code in an OO programming language is not a part of OOAD, is an end goal
- The artifacts created in the UP Design Model provide some of the information necessary to generate the code
- Roadmap to software development
    - OOAD - logical solution, blueprints
    - OOP - running applications
    - Use case - requirements

**Creativity and Change During Implementation**

- OOAD produces a base for the application
  Scales up with elegances
  Robustness
- Code Changes are inevitable
  CASE Tools, and Reverse-Engineering
  Rational ROSE, or Borland Together

**Mapping Designs to Code**

Implementation in an OO programming language requires writing source code for

- Classes and interface definitions
- methods definition

**Creating Class Definitions from DCDs**

- This is sufficient to create a basic class definition in a OO language.
- If the DCD was drawn in a UML tool, it can generate the basic class definition from the diagrams.

DCDs depict some of the basic elements of a class or interface

- Name
- Superclass
- Operation signatures
- attributes

**Defining a Class with Methods and Attributes**

From the DCD, a mapping to the attributes (Java fields) and method signatures is straightforward.

**Video Content / Details of website for further learning (if any):**

https://study.com/academy/lesson/mapping-code-using-outlines-and-flow-charts.html

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

LECTURE HANDOUTS

L -40

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** Mapping design to code

**Introduction :**

- Mapping from Design to Code. Each class in design is implemented by coding it in a programming language or by using a pre-existing component

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- GoF Patterns
- UML Diagrams

**Detailed content of the Lecture:**

**Programming and Iterative, Evolutionary Development**

- The creation of code in an OO programming language is not a part of OOAD, is an end goal
- The artifacts created in the UP Design Model provide some of the information necessary to generate the code
- Roadmap to software development
    - OOAD - logical solution, blueprints
    - OOP - running applications
    - Use case - requirements

**Creativity and Change During Implementation**

- OOAD produces a base for the application
  Scales up with elegances
  Robustness
- Code Changes are inevitable
  CASE Tools, and Reverse-Engineering
  Rational ROSE, or Borland Together

**Mapping Designs to Code**

Implementation in an OO programming language requires writing source code for

- Classes and interface definitions
- methods definition

**Creating Class Definitions from DCDs**

- This is sufficient to create a basic class definition in a OO language.
- If the DCD was drawn in a UML tool, it can generate the basic class definition from the diagrams.

DCDs depict some of the basic elements of a class or interface

- Name
- Superclass
- Operation signatures
- attributes

**Defining a Class with Methods and Attributes**

From the DCD, a mapping to the attributes (Java fields) and method signatures is straightforward.

**Video Content / Details of website for further learning (if any):**

https://study.com/academy/lesson/mapping-code-using-outlines-and-flow-charts.html

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L-41**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** Testing: Issues in OO Testing

**Introduction :**

- **Issues in Object oriented testing:** Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including encapsulation, inheritance, and polymorphism. These concepts lead to issues, which are yet to be resolved

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Mapping Design to Code

**Detailed content of the Lecture:**

**Definition**

- Software typically undergoes many levels of testing, from unit testing to system or acceptance testing.
- Typically, in-unit testing, small "units", or modules of the software, are tested separately with focus on testing the code of that module.
- In higher, order testing (e.g., acceptance testing), the entire system (or a subsystem) is tested with the focus on testing the functionality or external behavior of the system.

**Issues in Testing Classes:**

- In object-oriented programs, control flow is characterized by message passing among objects, and the control flow switches from one object to another by inter-object communication.
- Consequently, there is no control flow within a class like functions.
- This lack of sequential control flow within a class requires different approaches for testing.

**Techniques of object-oriented testing**

- **Fault Based Testing:**

  This type of checking permits for coming up with test cases supported the consumer specification or the code or both.

  It tries to identify possible faults (areas of design or code that may lead to errors.). For all of these faults, a test case is developed to "flush" the errors out.

  These tests also force each time of code to be executed.

  This method of testing does not find all types of errors.

- **Class Testing Based on Method Testing:**

  This approach is the simplest approach to test classes.

  Each method of the class performs a well defined cohesive function and can, therefore, be related to unit testing of the traditional testing techniques.

  Therefore all the methods of a class can be involved at least once to test the class.
- **Random Testing:**

  It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behavior of the categories
- **Partition Testing:**

  This methodology categorizes the inputs and outputs of a category so as to check them severely.

  This minimizes the number of cases that have to be designed.
- **Scenario-based Testing:**

  It primarily involves capturing the user actions then stimulating them to similar actions throughout the test.

  These tests tend to search out interaction form of error.

**Video Content / Details of website for further learning (if any):**
https://slideplayer.com/slide/7780236/

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

LECTURE HANDOUTS

L-42

CSE

III / V

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** Class Testing

**Introduction :**

- Class testing is the base of object-oriented software testing. It involves three aspects: testing each method, testing the relations among class methods and testing the inheriting relation between class and subclass.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- Issues in OO Testing

**Detailed content of the Lecture:**

**Unit**

- The smallest chunk that can be compiled by itself
- A single method that does not call use other methods
- Something small enough to be developed by one person

**When units are methods**

- Simplistic view reduces to traditional procedural unit testing
  Apply functional and structural test methods
  Require drivers and stubs
- Encapsulation advantage
  Methods are simple
- Encapsulation disadvantage
  Interface complexity is high
  Intense message sending
  Lots of work building stubs and drivers
- Burden of testing moves to integration level
  Intraclass
  Interclass

**When units are classes**

- Solves intraclass integration problem
- Different views
  Static view – class text

Ignores inheritance
Good only for reading
Compile-time view
When inheritance occurs
Execution view
Behavioral view – class are instantiated
This is where testing occurs

- Cannot test abstract classes
- Cannot be instantiated
- Choices

Classes flattened for testing
Need to unflatten when testing is completed
Use inherited classes
Configuration management nightmare

- Make most sense when

Little inheritance occurs
Intraclass control complexity
Interesting statechart

**Video Content / Details of website for further learning (if any):**
https://slideplayer.com/slide/7780236/

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

**LECTURE HANDOUTS**

**L-43**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** OO Integration Testing

**Introduction :**

- Integration Testing: This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead.

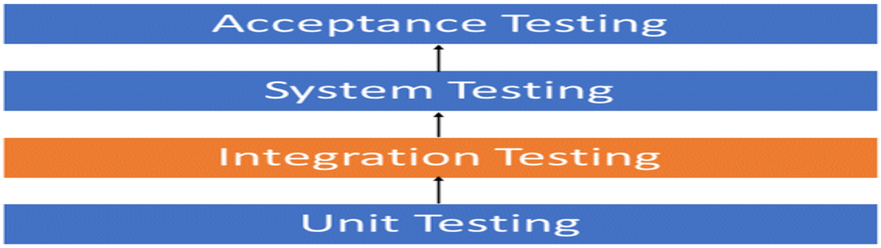
**Prerequisite knowledge for Complete understanding and learning of Topic:**

- OO Testing

**Detailed content of the Lecture:**

**Integration Testing**

- It is defined as a type of testing where software modules are integrated logically and tested as a group.
- A typical software project consists of multiple software modules, coded by different programmers.
- The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated



**Approaches, Strategies, Methodologies of Integration Testing**

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following

  Top Down Approach

  Bottom Up Approach

  Sandwich Approach - Combination of Top Down and Bottom Up

**Big Bang Approach:**

Here all component are integrated together at once and then tested.

**Advantages:**

Convenient for small systems.

**Incremental Approach**

- In this approach, testing is done by joining two or more modules that are logically related.
- Then the other related modules are added and tested for the proper functioning.
- The process continues until all of the modules are joined and tested successfully.

**Stub**: Is called by the Module under Test.

**Driver**: Calls the Module to be tested.

**Bottom-up Integration**

- In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing.

**Top-down Integration:**

- In Top to down approach, testing takes place from top to down following the control flow of the software system.
- Takes help of stubs for testing.

**Hybrid/ Sandwich Integration**

- In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches.

**Guidelines for Integration Testing**

- First, determine the Integration Test Strategy that could be adopted and later prepare the test cases and test data accordingly.
- Study the Architecture design of the Application and identify the Critical Modules.
- These need to be tested on priority.
- Obtain the interface designs from the Architectural team and create test cases to verify all of the interfaces in detail.
- Interface to database/external hardware/software application must be tested in detail.
- After the test cases, it's the test data which plays the critical role.
- Always have the mock data prepared, prior to executing.
- Do not select test data while executing the test cases.

**Video Content / Details of website for further learning (if any):**

https://www.guru99.com/integration-testing.html

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007

Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

**(An Autonomous Institution)**

**(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)**
**Rasipuram - 637 408, Namakkal Dist., Tamil Nadu**

Estd. 2000

**LECTURE HANDOUTS**

**L-44**

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : **III / V/ A**

**Unit** : **V- CODING AND TESTING** **Date of Lecture:**

**Topic of Lecture:** GUI Testing- OO System Testing

**Introduction :**

- GUI testing is defined as the process of testing the system's Graphical User Interface of the Application Under Test.
- GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes, and windows, etc.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- OO Testing
- Testing
- GUI

**Detailed content of the Lecture:**

**Graphical User Interface Testing (GUI) Testing**

- Graphical User Interface Testing (GUI) Testing is the process for ensuring proper functionality of the graphical user interface (GUI) for a specific application.
- GUI testing generally evaluates a design of elements such as layout, colors and also fonts, font sizes, labels, text boxes, text formatting, captions, buttons, lists, icons, links and content.

**Feature of Graphical User Interface Testing (GUI):**

- It is provide customizable test report.
- It is run tests in parallel or distribute on a Selenium Grid with built-in Selenium Web
- driver.
- It allows you to test the functionality from a user's perspective.
- Sometimes the internal functions of the system work correctly but the user interface doesn't then GUI testing is good to have in addition to the other types.
- It provide reliable object identification, even for web elements with dynamic IDs.

**Types of Graphical User Interface Testing (GUI) Testing:**

- Analog Recording
- Object based Recording

**Challenges with Graphical User Interface Testing (GUI) Testing:**

- Technology Support
- Stability of Objects
- Instrumentation

- GUI testing is a testing technique in which the application's user interface is tested whether the application performs as expected with respect to user interface behaviour.
- GUI Testing includes the application behaviour towards keyboard and mouse movements and how different GUI objects such as toolbars, buttons, menubars, dialog boxes, edit fields, lists, behavior to the user input.

**GUI TESTING GUIDELINES**

- Check Screen Validations
- Verify All Navigations
- Check usability Conditions
- Verify Data Integrity
- Verify the object states
- Verify the date Field and Numeric Field Formats

**CHALLENGES WITH GRAPHICAL USER INTERFACE TESTING (GUI) TESTING:**

There is some challenge which is occurring during Graphical user interface testing.

These are given below.

- Technology Support
- Stability of Objects
- Instrumentation

**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=1R3vsu_7n0E

**Important Books/Journals for further learning including the page nos.:**

UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015[24-32]

**Course Faculty**

**Verified by HOD**

# MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

**LECTURE HANDOUTS**

L-45

**CSE**

**III / V**

**Course Name with Code** : 16CSD06 & OBJECT ORIENTED ANALYSIS AND DESIGN

**Course Faculty** :

**Year / Semester/Section** : III / V/ A

**Unit** : V- CODING AND TESTING **Date of Lecture:**

**Topic of Lecture:** GUI Testing- OO System Testing

**Introduction :**

- GUI testing is defined as the process of testing the system's Graphical User Interface of the Application Under Test.
- GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes, and windows, etc.

**Prerequisite knowledge for Complete understanding and learning of Topic:**

- OO Testing
- Testing
- GUI

**Detailed content of the Lecture:**

**System Testing**

- System testing is a level of testing that validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the end-to-end system specifications.
- Usually, the software is only one element of a larger computer-based system.

**System Testing is Blackbox**

- Two Category of Software Testing
  - Black Box Testing
  - White Box Testing
- System test falls under the black box testing category of software testing.
- White box testing is the testing of the internal workings or code of a software application.

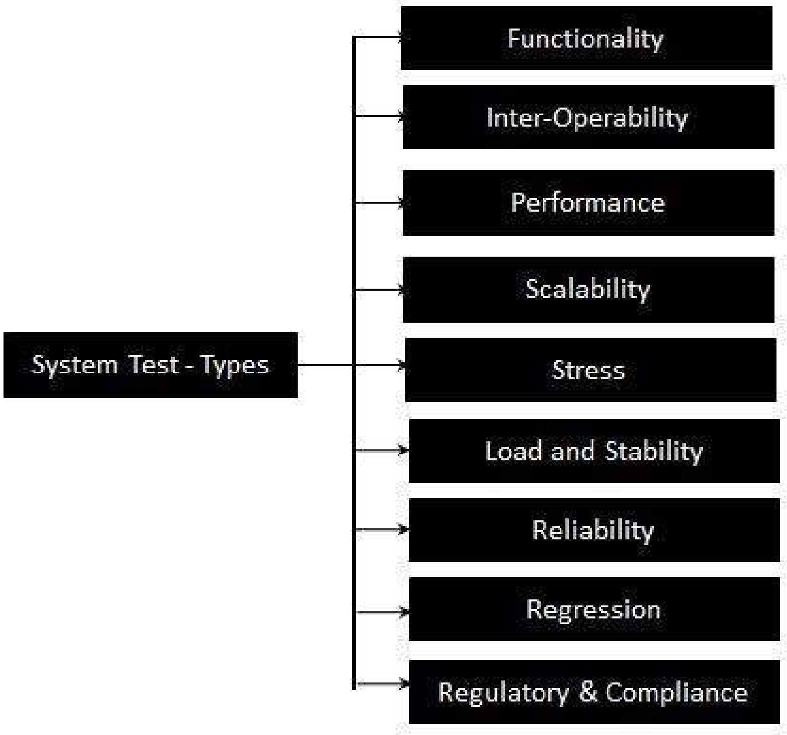
**Different Types of System Testing**

- **Usability Testing**- mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
- **Load Testing**- is necessary to know that a software solution will perform under real-life loads.
- **Regression Testing-** involves testing done to make sure none of the changes made over the course of the development process have caused new bugs.
- It also makes sure no old bugs appear from the addition of new software modules over time.
- **Recovery testing** - is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
- **Migration testing**- is done to ensure that the software can be moved from older system

infrastructures to current system infrastructures without any issues.

- **Functional Testing** - Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions.
- Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
- **Hardware/Software Testing** - IBM refers to Hardware/Software testing as "HW/SW Testing".
- This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.
- System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements.
- In System testing, the functionalities of the system are tested from an end-to-end perspective.
- System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased.
- It includes both functional and Non-Functional testing.

Types of System Tests:



**Video Content / Details of website for further learning (if any):**
https://www.youtube.com/watch?v=1R3vsu_7n0E

**Important Books/Journals for further learning including the page nos.:**
UML Distilled, Abraham Martin Fowler, PHI/Pearson Education, 2007
Object-Oriented Analysis, Design and Implementation, Brahma Dathan, Sarnath Ramnath, Springer, 2015[51-59]

**Course Faculty**

**Verified by HOD**