OOMD VIVA Questions

What are traditional development methodologies?

Most traditional development methodologies are either algorithm centric or data centric. In an algorithmic centric methodology, you think of an algorithm that can accomplish the task, and then build data structures for that algorithm to use.
In data centric methodology, you think how to structure the data, and then build algorithms around that structure.

What is Object Oriented development methodology?

In object oriented environment, software is a collection of discrete objects that encapsulate their data and the functionality to model the real world "objects". The object oriented life cycle encourages a view of the world as a system of cooperative and collaborating agents.

Define an object.

An object is an instance of a class. An object is the combination of data and logic that represents some real world entity. An object has identity, state, and behavior.

What is a class?

A group of objects having common structure and behavior is called a class. A class is also called an object template from which objects can be created.
Classes are important mechanism for classifying objects.

What is a method?

In the object model, object behavior is described in methods or procedures. Basically a method is a function or procedure that is defined for a class. Methods encapsulate the behavior of the object, provide interfaces to the object, and hide any of the internal structures and states maintained by the object.
Methods are similar to functions, procedures, or subroutines in more traditional programming languages, such as COBOL, Basic or C.

How messages are different from methods?

Messages essentially are non-specific function calls. A message is different from a subroutine call, since different objects can responds to same message in different ways. A message differs from function in that a function says how to do something and message says what to do.

What is the distinction between traditional development methodologies and object oriented development methodologies?

Traditional approach focuses on functions of the system where as object oriented approach centers on object which is the combination of data and functionality.

What are the advantages of object oriented approach?

a)Higher levels of abstraction
b)Seamless transition among different phases of software development

c)Encouragement of good programming techniques
d)Promotion of reusability

What is Unified Approach(UA)?
UA is based on methodologies by Booch, Rumbaugh,and Jacobson which combines best practices, processes, and guidelines along with OMG's Unified Modeling Language(UML)

What is the heart of UA?
The heart of UA is Jacobson's Use case. The use case represents a typical interaction between a user and a computer system to capture the user's goals and needs.

What is class hierarchy?
An OO system organizes classes into a super class-subclass hierarchy. At the top of the class hierarch are the most general classes and at the bottom are the most specific.
A subclass inherits all of the properties and methods (procedures) defined in its super class. Subclasses usually add new methods and properties specific to that class.
Subclasses may refine or constrain the state and behavior inherited from its super class.

What is inheritance?
Inheritance is the property of OO systems that allows objects to be built from other objects. Inheritance is a relationship between classes where one class is the parent class of another (derived) class. The parent class is also known as the base class or super class.

What is Polymorphism?
Poly means "many" and morph means "form". Polymorphism means the same operation may behave differently on different classes.

What is association?
Association represents the relationship between objects and classes.

What is multiple inheritances?
When one class inherits its state (attributes) and behavior from more than one super class, it is referred to as multiple inheritances.

What is dynamic binding?
The process of determining (dynamically) at run time which functions to invoke is termed dynamic binding.

What is static binding?
The process of determining at compile time which functions to invoke is termed static binding.

Write the four quality measures for software development?
Correspondence, correctness, verification, and validation.

What is object persistence?
Objects have life time. They are created and can exist for a period of time.
A file or a database can provide support for objects having a longer life time longer than the duration of the process for which they were created. This characteristic is called object persistence.

What is cardinality?
Cardinality specifies how many instances of one class may relate to a single instance of an associated class.

What is a formal class or abstract class?
Formal or abstract classes have no instances but define the common behaviors that can be inherited by more specific classes.

What is a meta-class?
A meta-class is a class about a class. They are normally used to provide instance variables and operations.

Define Encapsulation?
Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior.

What is the need of an Object diagram?
An object diagram is used to show the existence of objects and their relationships in the logical design of a system.

Explain object relationship and associations.

a. Association represents the relationships between objects and classes.
b. Associations are bidirectional.
c. Cardinality specifies how many instances of one class may relate to a single instance of an associated class.
d. Cardinality constraints the number of related objects and often is described as being “one” or “many”

Define collaboration.
The object-oriented programming community has adopted use-cases to a remarkable degree. Scenarios are a great way of examining who does what in the interactions among objects and what role they play; that is their interrelationships. This intersection among objects’ roles to achieve a given goal is called collaboration.

Why do we go for object oriented systems development?
The motivation factor behind object-oriented system development is the desire to make software development easier and more natural by raising the level of abstraction to the point where the level of abstraction to the point where applications can be implemented in the same terms n which they are described by users.

Write about the four phases in OMT?

OMT consists of four phases. They are

• Analysis-The results are objects and dynamic & functional models.

• System design-The results are a structure of the basic architecture of the system along with high-level strategy decisions.

• Object Design-Produces a design document, consisting of detailed objects static, dynamic and functional models

• Implementation-This activity produces reusable, extendible, robust code.

What do you mean by object diagram?

The object model of OMT is represented graphically with an object diagram. The object diagram contains classes interconnected by association lines. Each class represents a set of individual objects. The association lines establish relationships among the classes. Each association line represents a set of links from the objects o f one class to the objects of another class

What are the diagrams used in Booch methodology?

The Booch methodology consists of the following diagrams:

Class diagrams.
Object diagrams.
State transition diagrams.
Module diagrams.
Process diagrams.
Interaction diagrams

Give the steps involved in Macro development process in Booch methodology.

The macro development process consists of the following steps:

o Conceptualization
Establish the core requirements and develop a prototype.

o Analysis and development of the model
Use the class diagram to describe the roles and responsibilities of objects. Use the object diagram to describe the desired behavior of the system.

o Design or create the system architecture.
Use the class diagram to decide what classes exist and how they relate to each other, the object diagram to decide what mechanisms are used, the module diagram to map out where each class and object should be declared, and the process diagram to determine to which processor to allocate a process.

o Evolution or implementation-
Refine the system through much iteration.

o Maintenance-Make localized changes to the system to add new requirements and eliminate bugs.

Give the steps involved in Micro development process in Booch methodology.

The micro process is a description of the day-to-day activities by a single or small group of software developers. It consists of the following steps.

o Identify classes and objects.
o Identify class and object semantics.
o Identify class and object relationships.

Write briefly about Use Cases.
Use cases are scenarios for understanding system requirements. A use case is an interaction between users and a system. The use-case model captures the goal of the user and the responsibility of the system to its users. The use-case model employs extends and uses relationships. The use cases are described as one of the following:

Nonformal text with no clear flow of events
Text with a clear flow of events
Formal style using pseudo code.

Define patterns.
Design pattern identifies the key aspects of a common design structure that makes it useful for creating a reusable object-oriented design. Furthermore, it identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. It describes when it applies, whether it can be applied in view of other design constraints and the consequences and trade-offs of its use.
A pattern is an instructive information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces.

Define frame work. Give the differences between design patterns and frameworks.
A frame work is a way of presenting a generic solution to a problem that can be applied to all levels in a development. Frameworks are a way of delivering application development patterns. A framework provides architectural guidance, captures the design decisions that are common to its application domain and thus emphasize design reuse over code reuse. The major differences between design patterns and frameworks are as follows:

A framework is executable software whereas design patterns represent knowledge and experience about the software.
Design patterns are more abstract than frameworks.
Design patterns are smaller architectural elements than frameworks.
Design patterns are less specialized than frameworks.

Define model. Explain about the types of model.
A model is an abstract representation of a system, constructed to understand the system prior to building or modifying it. It is a model of a simplified representation of reality. Models can represent static or dynamic situations.

• **Static model**
o It can be viewed as a snapshot of a system's parameters at rest or a specific point in time. They are needed to represent the structural or static aspect of a system. The UML class diagram is an example of static model.
• **Dynamic model**

o It can be viewed as a collection of procedures or behaviors that taken together reflect the behavior of a system over time. Dynamic modeling is the most useful during the design and implementation phases of the system development. The UML interaction diagrams and activity models are examples of dynamic models.

What are the advantages of Modeling?
Good models are essential for communication among project teams. As the complexity of systems increases, so does the importance of good modeling techniques. Some of the advantages are as follows:

Models make it easier to express complex ideas.
The main reason for modeling is to reduction of complexity.
Models enhance and reinforce learning and training.
The cost of modeling analysis is much lower than the cost of similar perimentation conducted in real system.
Manipulation of the model is much easier.

Give the nine UML graphical diagrams.

a. Class diagram(static)
b. Use-case diagram
c. Behavior diagram(dynamic)
i. Interaction diagram
1. Sequence diagram
2. Collaboration diagram
ii. State chart diagram
iii. Activity diagram
d. Implementation diagram.
i. Component diagram
ii. Deployment diagram.

What is a Package?
A package groups and manages the modeling elements, such as classes, their associations, and their structures. Packages themselves may be nested within other packages.

Define use-case.
Use cases are scenarios that describe how actors use the system. A use case is an interaction between users and a system. It captures the goal of the users and the responsibility of the system to its users. Jacobsons' definition of use case is, "A use case is a sequence of transactions in a system whose task is to yield results of measurable value to an individual actor of the system."

When 'extends' association is used?
The 'extends' association is used when you have one use case that is similar to another use case but does a bit more or is more specialized. It is like a subclass. Extends association is utilized to expand the common behavior to fit the special circumstances.

Define 'uses' association.
The uses association occurs when some of the use cases have subflows in common. Here to avoid describing a subflow more than once in several use cases, the common subflow can be extracted and made into a new use case of its own.
The relationship among the other use cases and this new extracted use case is called a uses association. When you want to share common sequences in several use cases, utilize the uses association by extracting common sequences into a new, shared use case. The uses association helps us to avoid redundancy by allowing a use case to be shared.

What is CRC?
Classes, responsibilities, and collaborators is a technique used for identifying classes' responsibilities, and collaborators and therefore their attributes and methods. Furthermore, CRC can help us identify classes. CRC is based on the idea that an object either can accomplish a certain responsibility itself or it may require the assistance of other objects.

Why do we need to identify the system's responsibilities?
We need to identify the system's responsibilities because responsibilities identify problems that are to be solved. A responsibility serves as a handle for discussing potential solutions. Once the system's responsibilities are understood we can start identifying the attributes of the system's classes.

What do you mean by coupling?
Coupling is a measure of the strength of association established by a connection from one object or software component to another. Coupling is a binary relationship. For example A is coupled with B. Coupling is important when evaluating a design because it helps us focus on an important issue in design.

What do you mean by degree of coupling?
The degree of coupling is a function of How complicated the connection is. Whether the connection refers to the object itself or something inside it. What is being sent or received. The degree or strength of coupling between two components is measured by the amount and complexity of information transmitted between them. Coupling increases with increasing complexity and decreases when the connection is to the component interface rather than to an internal component. Coupling is also lower for data connections than for control connections.

What are the two types of coupling?
Object oriented design has two types of coupling. They are,

Interaction coupling

Interaction coupling involves the amount and complexity of messages between components. It is desirable to have little interaction.

_ Inheritance coupling

Inheritance is a form of coupling between super and subclasses. A subclass is coupled to its super class in terms of attributes and methods.
Unlike interaction coupling, high inheritance coupling is desirable.

What do you mean by cohesion? Give the types of cohesion.
Cohesion can be defined as the interactions within a single object or software component. Cohesion reflects the "single-purpose ness" of an object. Cohesion helps in designing classes that have very specific goals and clearly defined purposes.

Method cohesion
A method should carry only one function.
Class cohesion
All the class's methods and attributes must be used by internal methods or derived classes' methods.
Inheritance cohesion
Concerned with how classes are interrelated.

Differentiate coupling and cohesion?
Coupling deals with interactions between objects or software components while cohesion deals with the interactions within a single object or software component. Highly cohesive components can lower coupling because only a minimum of essential information need to b passed between components.

What are some characteristics of a bad design?
The five rules are,
If it looks messy, then it's probably a bad design.
If it is too complex, then it's probably a bad design.
If it is too big, then it's probably a bad design.
If people don't like it, then it's probably a bad design.
If it doesn't work, then it's probably a bad design.

Describe Rumbaugh's Object Modeling Technique?
• Describes the dynamic behavior of objects in a system.
• Four phases.
_ Analysis – results are objects, dynamic and functional models.
System design – gives a structure of the basic architecture.
Object design – produces a design document.
Implementation – produces reusable code.
• Three different parts
Object Model – presented by object model and the data dictionary.
Dynamic model - presented by the state diagrams and event flow diagrams.
_ Functional Model – presented by data flow and constraints.

Give detailed notes about the Booch Methodology?
• Helps to design your system using the object paradigm.
• Is criticized for his large set of symbols.
• Consists of the following diagrams:
Class diagrams.
Object diagrams.

State transition diagrams.
Module diagrams.
Process diagrams.
Interaction diagrams.
• Two processes:
Macro development process
Micro development process.
• Macro development process.
Primary concern – technical management of the system.
Steps involved:
o Conceptualization.
o Analysis and development of the model.
o Design or create the system architecture.
o Evolution or implementation.
o Maintenance.
• Micro development process.
Describes the day-to-day activities.
Steps involved:
o Identify classes and objects.
o Identify classes and object semantics.
o Identify classes and object relationships.
o Identify classes and object interfaces and implementation.

Give a detailed account of Jacobson methodology?
• Covers the entire life cycle and stress traceability between the various phases.
• Use cases
_ Scenarios for understanding system requirements.
_ Non formal text with no clear flow of events.
_ Text easy to read.
_ Formal style using pseudo code.
Can be viewed as concrete or abstract (not initiated by actors).
• Object oriented software Engineering: Objectory
Use case model.
Domain Object Model.
Analysis Object Model.
Implementation model.
Test model.
• Object oriented business Engineering
Analysis phase.
Design and Implementation phase.
_ Testing phase.

What is UML? UML is Unified Modeling Language. It is a graphical language for visualizing specifying constructing and documenting the artifacts of the system. It allows you to create a blue print of all the aspects of the system, before actually physically implementing the system.

What is modeling? What are the advantages of creating a model? Modeling is a proven and well-accepted engineering technique which helps build a model. Model is a simplification of reality; it is a blueprint of the actual system that needs to be built. Model helps to visualize the system. Model helps to specify the structural and behavior of the system. Model helps make templates for constructing the system. Model helps document the system.

What are the different views that are considered when building an object-oriented software system? Normally there are 5 views. Use Case view - This view exposes the requirements of a system. Design View - Capturing the vocabulary. Process View - modeling the distribution of the systems processes and threads. Implementation view - addressing the physical implementation of the system. Deployment view - focus on the modeling the components required for deploying the system.

What are diagrams? Diagrams are graphical representation of a set of elements most often shown made of things and associations.

What are the major three types of modeling used? Major three types of modeling are structural, behavioral, and architectural.

Mention the different kinds of modeling diagrams used? Modeling diagrams that are commonly used are, there are 9 of them. Use case diagram, Class Diagram, Object Diagram, Sequence Diagram, statechart Diagram, Collaboration Diagram, Activity Diagram, Component diagram, Deployment Diagram.

What is Architecture? Architecture is not only taking care of the structural and behavioral aspect of a software system but also taking into account the software usage, functionality, performance, reuse, economic and technology constraints.

What is SDLC? SDLC is Software Development Life Cycle. SDLC of a system included processes that are Use case driven, Architecture centric and Iterative and Incremental. This Life cycle is divided into phases. Phase is a time span between two milestones. The milestones are Inception, Elaboration, Construction, and Transition. Process Workflows that evolve through these phase are Business Modeling, Requirement gathering, Analysis and Design, Implementation, Testing, Deployment. Supporting Workflows are Configuration and change management, Project management.

What are Relationships? There are different kinds of relationships: Dependencies, Generalization, and Association. Dependencies are relations ships between two entities that that a change in specification of one thing may affect another thing. Most commonly it is used to show that one class uses another class as an argument in the signature of the operation. Generalization is relationships specified in the class subclass scenario, it is shown when one entity inherits from other. Associations are structural relationships that are: a room has walls, Person works for a company. Aggregation is a type of association where there is a has a relation ship, That is a room has walls, Ã±o if there are two classes room and walls then the relation ship is called a association and further defined as an aggregation.

How are the diagrams divided? The nine diagrams are divided into static diagrams and dynamic diagrams.

1. **Static Diagrams** (Also called Structural Diagram): Class diagram, Object diagram, Component Diagram, Deployment diagram.
2. **Dynamic Diagrams** (Also called Behavioral Diagrams): Use Case Diagram, Sequence Diagram, Collaboration Diagram, Activity diagram, Statechart diagram.
3. **What are Messages?** A message is the specification of a communication, when a message is passed that results in action that is in turn an executable statement.

4. **What is an Use Case?** A use case specifies the behavior of a system or a part of a system, Ã³se cases are used to capture the behavior that need to be developed. It involves the interaction of actors and the system.
5. **What are Relationships in UML?**
6. **Dependency**
   Dependency is the relationship between two things in which a change in one thing may affect the other thing, but not necessarily the reverse. Graphically, it can be represented as a dashed directed line, directed to the thing being depended on.
7. **Association**
   An association is a relationship that connects classes. Example of association is relationship between person and company. An association is rendered as solid line.

   Adornments of association

   Name: You can use Name to specify the nature of association
   Role: Each class has specific role that participates in the relationship. We can specify role to the classes involve in the relationship.
   Multiplicity: It represents how many objects may be connected across an instance of an association.
   Aggregation: It implies whole/part relationship where one class represents larger thing which consists of smaller things.

   Generalization
   In this type of relationship, the child shares the structure and behavior of the parent. It is represented as a solid line with a hollow arrowhead pointing to the parent.

Deployment diagram

Deployment Diagram shows the configuration of run time processing nodes and the components that live on them. It is used for modeling topology of the hardware on which your system executes.

Contents of the diagram are
Nodes,
Dependency.

Define UML?

Unified Modeling Language, a standard language for designing and documenting a system in an object-oriented manner. It has nine diagrams which can be used in design document to express design of software architecture.

(I) Can you explain use case diagrams?

Use case diagram answers what system does from the user point of view. Use case answer 'What will the system do?'. Use cases are mainly used in requirement document to depict clarity regarding a system. There are three important parts in a use case scenario, actor and use case.

Scenario: A scenario is a sequence of events which happen when a user interacts with the system.

Actor: Actor is the who of the system, in other words the end user.

Use Case: Use case is task or the goal performed by the end user. Below figure 'Use Case' shows a simple scenario with 'Actor' and a 'Use Case'. Scenario represents an accountant entering accounts data in the system. As use case's represent action performed they are normally represented by strong verbs.

Actor's are represented by simple stick man and use case by oval shape as shown in figure 'Use Case' below.

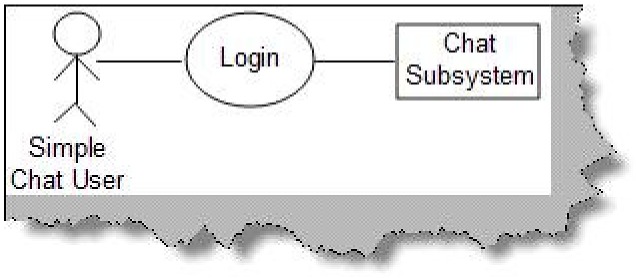


Figure: Use Case

(I) Can you explain primary and secondary actors?

Actors are further classified in to two types primary and secondary actors. Primary actors are the users who are the active participants and they initiate the user case, while secondary actors are those who only passively participate in the use case.

(I) How does a simple use case look like?

Use case's have two views of representation in any requirement document. One is the use case diagrams and the other is a detail step table about how the use case works. So it's like a pair first an over view is shown using a use case diagram and then a table explaining the same in detail. Below is a simple 'login' use case shown diagrammatically and then a detail table with steps about how the use case is executed.

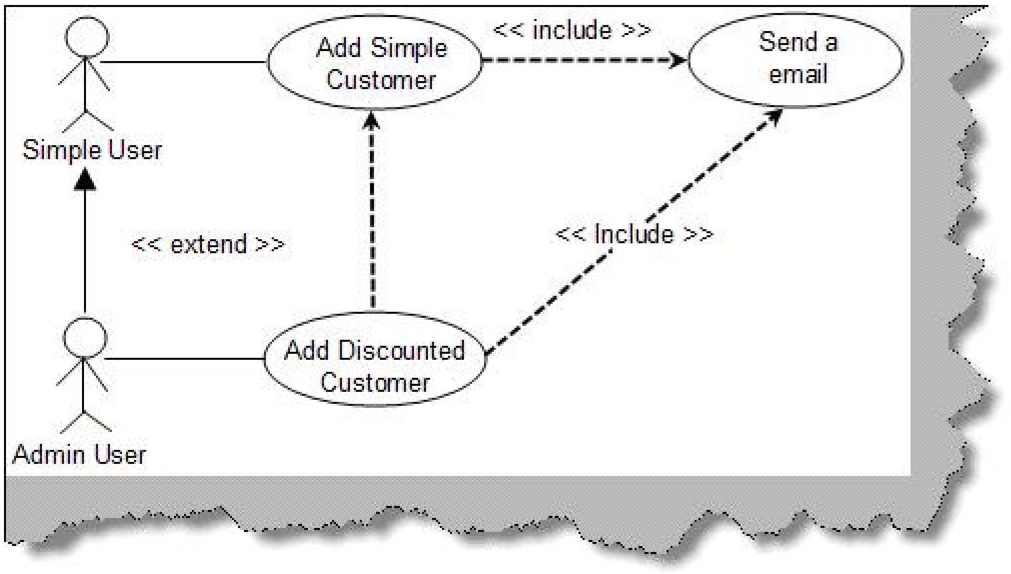


Figure: Login Use Case

Use Case	Rel001
Use Case Name	Login
Description	This uses depicts the flow of how user will log-in into the chat application.
Primary Actor	Simple chat user.
Trigger	User types chat application on URL of the browser.
Pre-condition	NA
Assumption	No password is currently present for the system Rooms will remain constant as explained in the assumption section of this document
Failed End conditions	Duplicate user name is not allowed in the chat application.
Action	User clicks on the log-in button.
Main Scenario	• User types chat application on URL of the browser which in turn opens the main page. • In the main page of application user is popped up with 'Enter user name' option and various 'rooms' option drop down menu. • User then types the name and selects one of the room from drop down menu and then clicks on the 'Log-in' button. • Application then checks whether the user name is unique in the system if not then user is popped up with error message that "user already exist".

	• After entering the unique name the user is finally logged in the application.
Action	NA
Alternate Scenario	NA
Success Scenarios	1. Opens page of a selected room in that other user names and their messages can be seen.
Note and Open Issues	NA

Table: Login use case table

Note: You must be wondering why we have this pair why not just a use case table only. Use case diagrams are good to show relationship between use case and they also provide high over view. The table explanation of a use case talks details about the use case. So when a developer or a user is reading a requirement document, he can get an overview by looking at the diagram if he is interested he can read the use case tables for more details.

(I) Can you explain 'Extend' and 'Include' in use cases?

'Extend' and 'Include' define relationships between use cases. Below figure 'Extend and Include' shows how these two fundamentals are implemented in a project. The below use case represents a system which is used to maintain customer. When a customer is added successfully it should send an email to the admin saying that a new customer is added. Only admin have rights to modify the customer. First lets define extend and include and then see how the same fits in this use case scenario.

Include: Include relationship represents an invocation of one use case by the other. If you think from the coding perspective its like one function been called by the other function.

Extend: This relationship signifies that the extending use case will work exactly like the base use case only that some new steps will inserted in the extended use case.

Below figure 'Extend and Include' shows that 'add customer' is same as the 'add discounted customer'. The 'Add discounted customer' has an extra process, to define discount for the discounted customer which is not available for the simple customer. One of the requirements of the project was that when we add a customer, the system should send an email. So after the customer is added either through 'Add simple customer' use case or 'Add discounted customer' use case it should invoke ‘send a email’ use case. So we have defined the same with a simple dotted line with <<include>> as the relationship.

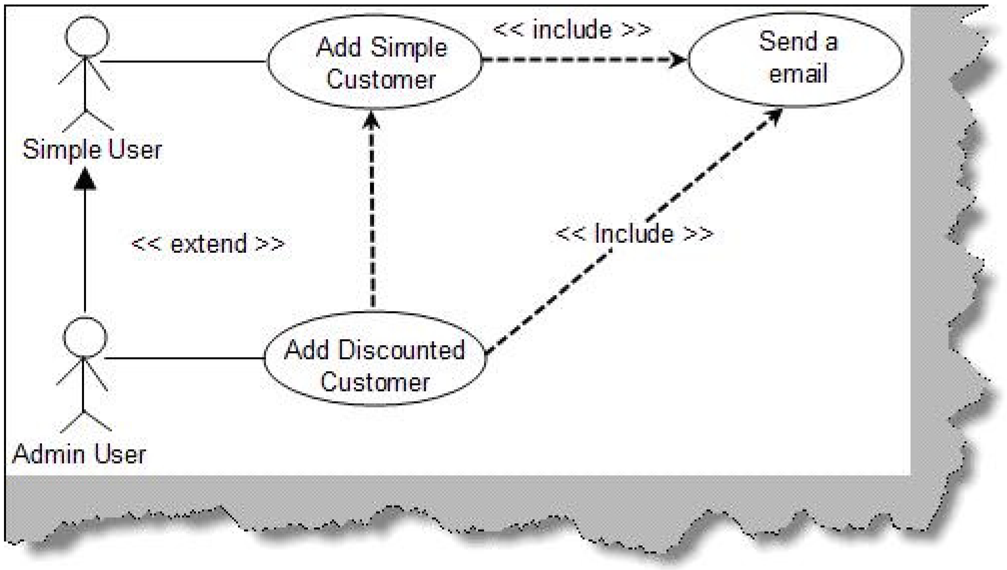


Figure: Extend and Include

Note: One of the points to be noted in the diagram ‘Extend and Include’ is we have defined inheritance relationship between simple and admin user. This also helps us defining a technical road map regarding relationships between simple and admin user.

(I) Can you explain class diagrams?

Class diagram

Class is basically a prototype which helps us create objects. Class defines the static structure of the project. A class represents family of an object. By using Class we can create uniform objects.

In the below figure you can see how the class diagram looks. Basically there are three important sections which are numbered as shown in the below. Let’s try to understand according to the numbering:

- Class name: This is the first section or top most section of the Class which represents the name of the Class (clsCustomer).
- Attributes: This is the second section or the middle section of the class which represents the properties of the system.
- Methods: This section carries operation or method to act on the attributes.

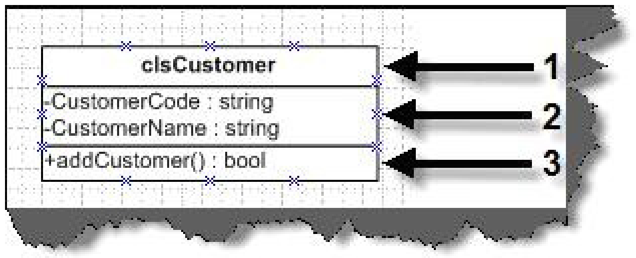


Figure: Three sections of the class

Now in the next section we will have a look on Association relationship between these classes.

(B) How do we represent private, public and protected in class diagrams?

In order to represent visibility for properties and methods in class diagram we need to place symbols next to each property and method as shown in figure 'Private, Public and Protected'. '+' indicates that it's public properties/methods. '-'indicates private properties which means it can not be accessed outside the class. '#' indicate protected/friend properties. Protected properties can only be seen within the component and not outside the component.

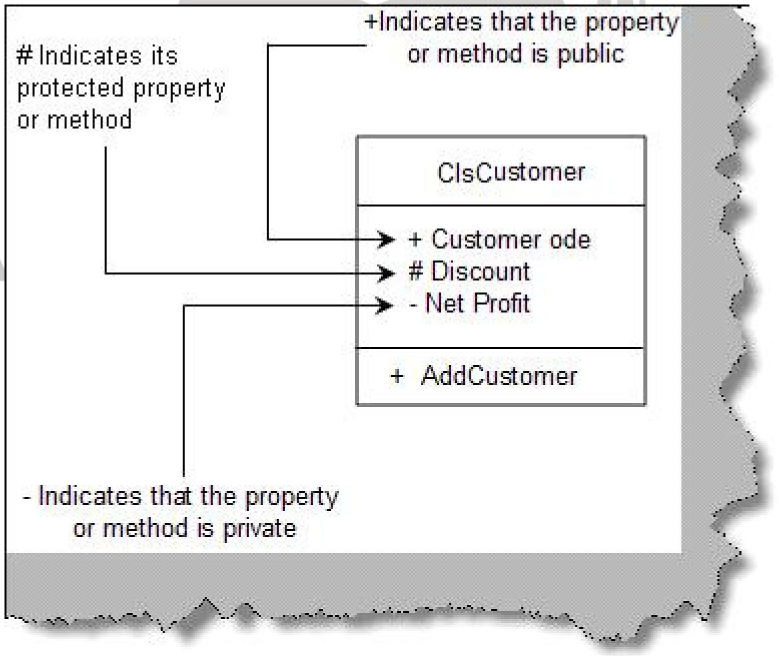

+Indicates that the property or method is public
# Indicates its protected property or method
ClsCustomer
+ Customer ode
# Discount
- Net Profit
+ AddCustomer
- Indicates that the property or method is private

Figure: Private, public and protected

(I) what does associations in a class diagram mean?

Associations in Class diagrams

A single Class cannot represent the whole module in a project so we need one or more classes to represent a module. For instance, a module named 'customer detail' cannot be completed by the customer class alone , to complete the whole module we need customer class, address class, phone class in short there is relationship between the classes. So by grouping and relating between the classes we create module and these are termed as Association. In order to associate them we need to draw the arrowed lines between the classes as shown in the below figure.

In the figure 'Order is paid by payments class', we can see Order class and the Payment class and arrowed line showing relationship that the order class is paid using payment class in other words order class is going to be used by payment class to pay the order. The left to right marked arrow basically shows the flow that order class uses the payment class.
In case payment class using the order class then the marked arrow should be right to left showing the direction of the flow.

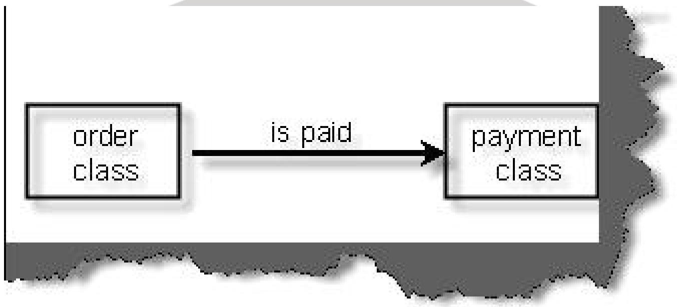
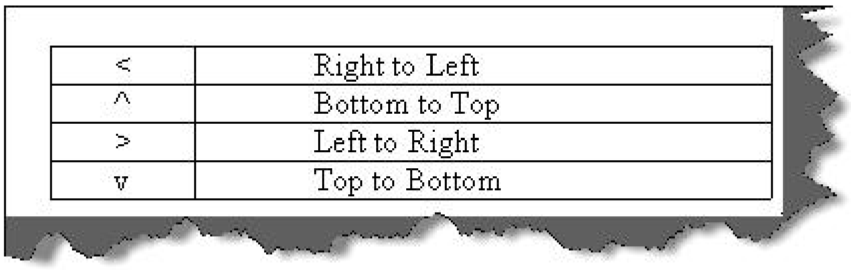


Figure:- Order is paid by Payments class

There are four signs showing the flow:-



<	Right to Left
^	Bottom to Top
>	Left to Right
v	Top to Bottom

Figure: Direction signs in UML

Multiplicity

Multiplicity can be termed as classes having multiple associations or one class can be linked to instances of many other classes. If you look at the below figure the

customer class is basically associated with the address class and also observes the notations (*, 0 and 1).If you look at the right hand side the (1….*) notation indicates that at least one or many instance of the address class can be present in the customer class. Now towards left hand side we have (0….*) notation indicating that address class can exist without or many customer class can link him.
In order to represent multiplicity of classes we have to show notations like (1….*), (0….*) as shown in below figure.

Note: '*' means "many" where as '(0, 1)' means "(zero or at least one)" respectively.

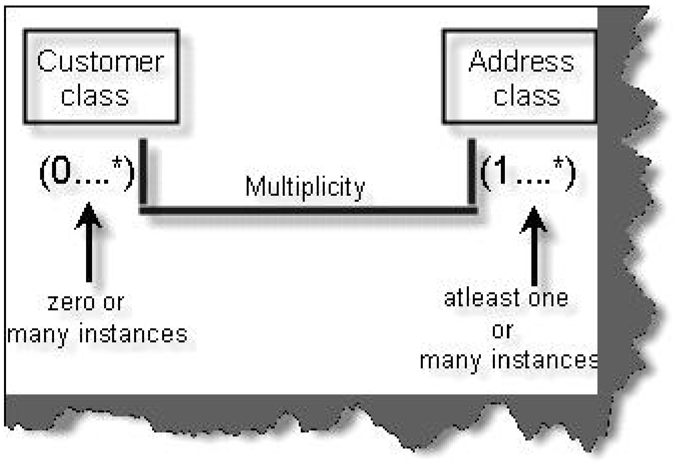


Figure: Multiplicity in Classes

(I) Can you explain aggregation and composition in class diagrams?

In this Association there are two types mainly Aggregation Association and Composition Association.

Aggregation Association signifies that the whole object can exist without the Aggregated Object. For example in the below figure we have three classes university class, department class and the Professor Class. The university cannot exist without department which means that university will be closed as the department is closed. In other words lifetime of the university depend on the lifetime of department.

In the same figure we have defined second Association between the department and the Professor. In this case, if the professor leaves the department still the department continues in other words department is not dependent on the professor this is called as Composition Association.

Note: The filled diamond represents the aggregation and the empty diamond represents the composition. You can see the figure below for more details.

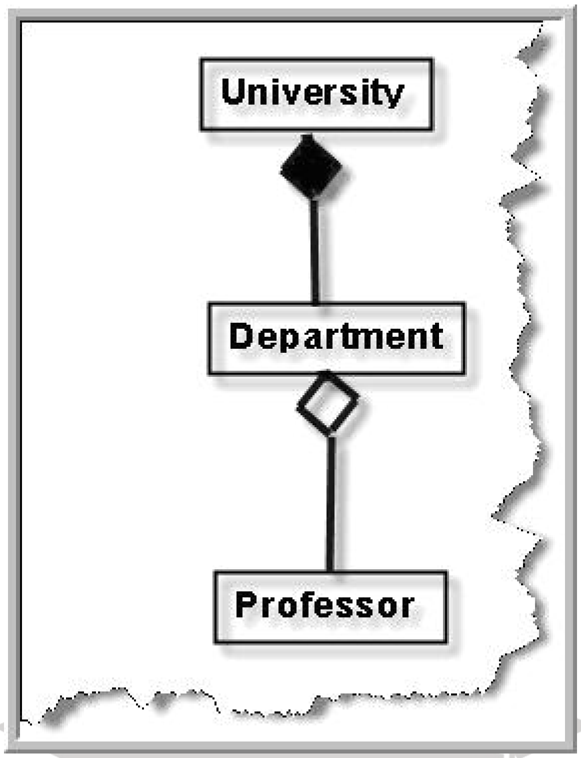


Figure: Aggregation and composition in action

(A) What are composite structure diagram and reflexive association in class diagrams?

Composite structure diagram

When we try to show Aggregation and Composition in a complete project the diagram becomes very complicated so in order to keep it simple we can use Composite structure diagram. In the below figure we have shown two diagrams one is normal diagram other is Composite structure diagram and the simplicity can easily be identified. In the composite diagram the aggregated classes are self contained in the main class which makes it simpler to read.

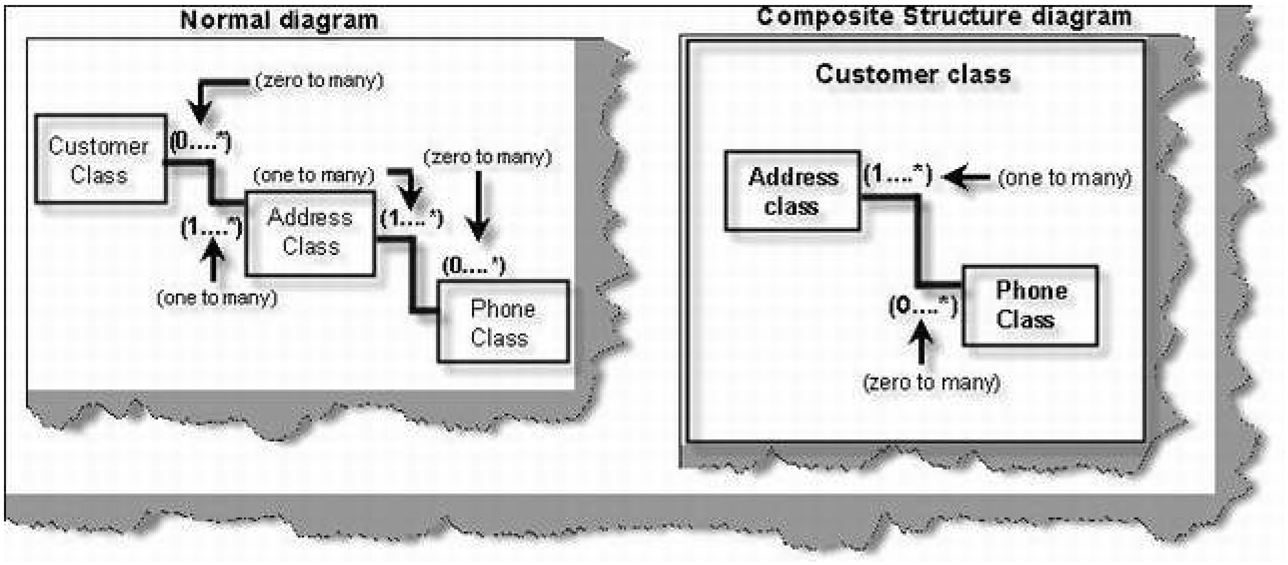


Figure: Composite Structure diagram

Reflexive associations

In many scenarios you need to show that two instances of the same class are associated with each other and this scenario is termed as Reflexive Association. For instance in the below figure shows Reflexive Association in the real project. Here you can see customer class has multiple address class and addresses can be a Head office, corporate office or Regional office. One of the address objects is Head office and we have linked the address object to show Reflexive Association relationship. This is the way we can read the diagram Regional address object is blocked by zero or one instance of Head office object.

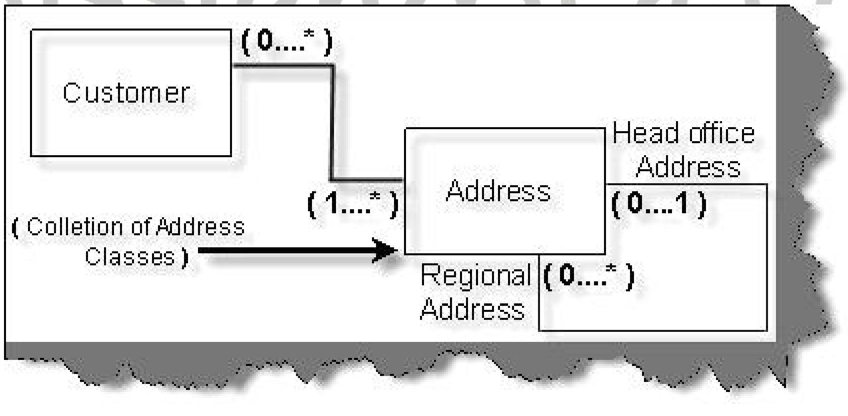


Figure: Reflexive association

(I) Can you explain business entity and service class?

Business entity objects represent persistent information like tables of a database. Just making my point clearer they just represent data and do not have business validations as such. For instance below figure 'Business entity and service' shows a simple customer table which with three fields 'Customer Code',' Customer Address' and 'Phone Number'. All these fields are properties in 'ClsCustomer' class. So 'ClsCustomer' class becomes the business entity class. The business entity class by itself can not do anything it's just a place holder for data. In the same figure we have one more class 'ClsServiceCustomer'. This class aggregates the business entity class and performs operations like 'Add',' Next' (Move to next record), 'Prev' (Move to previous record) and 'GetItem' (get a customer entity depending on condition).

With this approach we have separated the data from the behavior. The service represents the behavior while the business entity represents the persistent data.

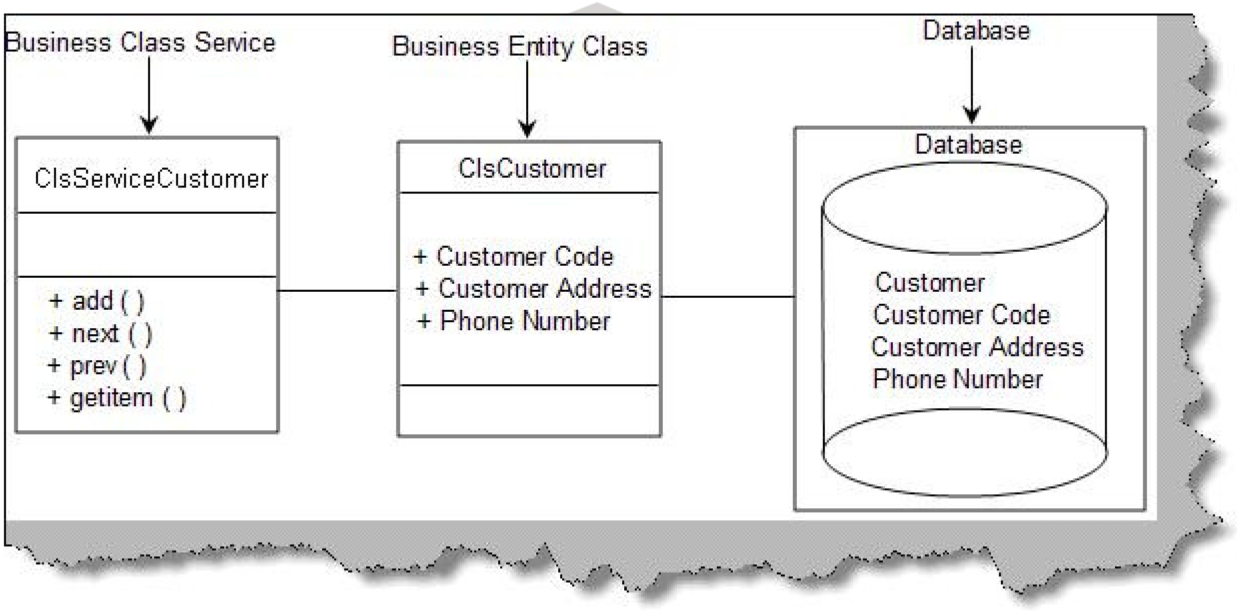


Figure:-Business entity and service

(I) Can you explain System entity and service class?

System entity class represents persistent information which is related to the system. For instance in the below figure 'System entity and service class' we have a system entity class which represents information about 'loggedindate' and 'loggedintime' of the system registry. System service class come in two flavors one is it acts like a wrapper in the system entity class to represent behavior for the persistent system entity data. In the figure you can see how the 'ClsAudit' system entity is wrapped by the 'ClsAuditSytem' class which is the system service class. 'ClsAuditSystem' adds 'Audit' and 'GetAudit' behavior to the 'ClsAudit' system entity class.

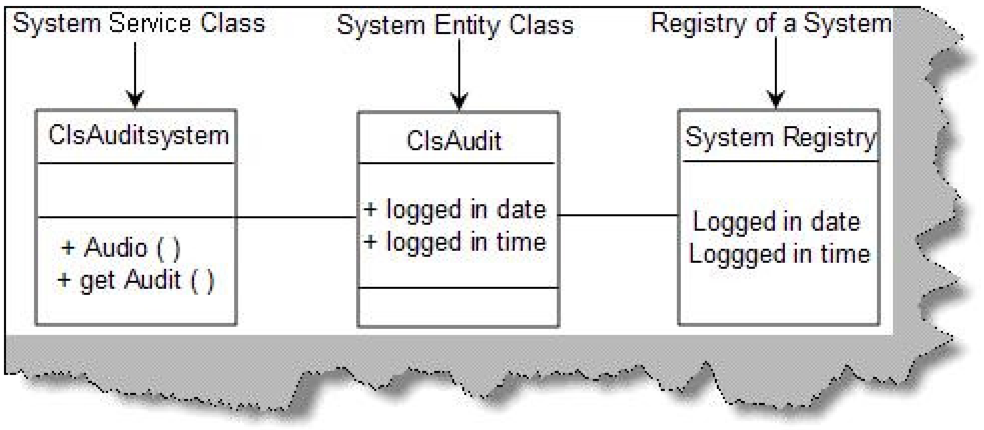


Figure: System entity and service class

The other flavor of the system service class is to operate on non-persistent information. The first flavor operated on persistent information. For instance the below figure 'Non-persistent information' shows how the class 'ClsPaymentService' class operates on the payment gateway to Check is the card exists , Is the card valid and how much is the amount in the card ?. All these information are non-persistent. By separating the logic of non-persistent data in to a system service class we bring high reusability in the project.

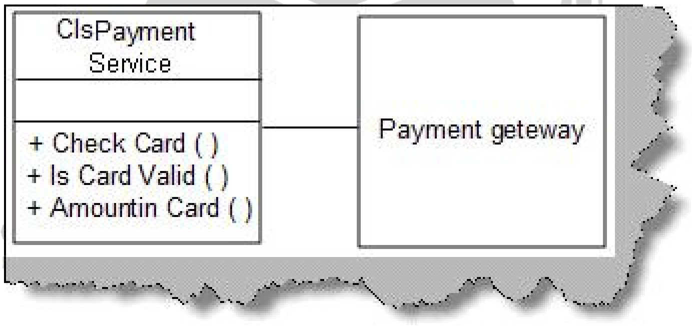


Figure: Non-persistent information

Note: The above question can be asked in interview from the perspective of how you have separated the behavior from the data. The question will normally come twisted like 'How did you separate the behavior from the data?'.

(B) Can you explain generalization and specialization?

Generalization and Specialization

In Generalization and Specialization we define the parent-child relationship between the classes. In many instance you will see some of the classes have same properties and operation these classes are called super class and later you can inherit from super class and make sub classes which have their own custom properties. In the below figure there are three classes to show Generalization and Specialization relationship. All phone types have phone number as a generalized property but depending upon landline or mobile you can have wired or simcard connectivity as specialized property. In this diagram the clsphone represent Generalization whereas clslandline and clsmobile represents specialization.

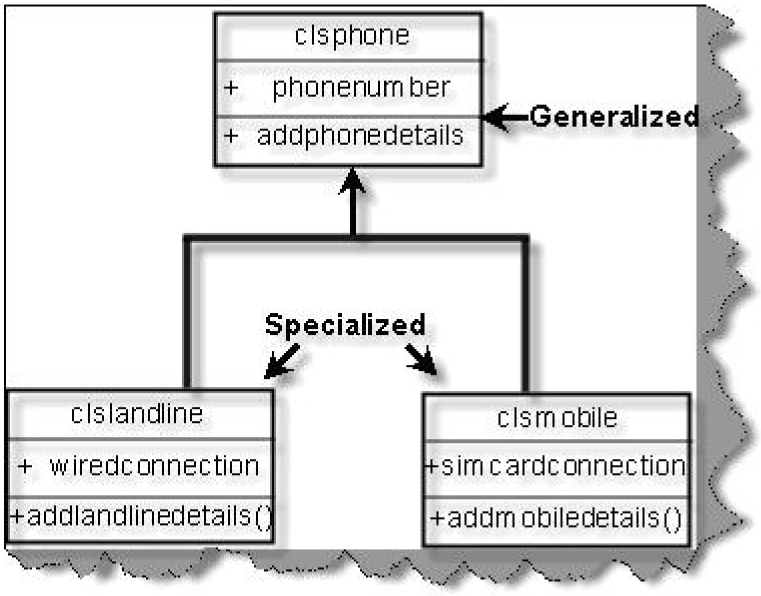


Figure: Generalization and Specialization

(B) How do we represent an abstract class and interface UML?

Interface is represented by <<type>> in the class diagram. Below figure 'Interface in action' shows we have defined an interface 'IContext'. Note the '<<type>>' represents an interface. If we want to show that the interface is used in a class we show the same with a line and a simple circle as shown in figure 'Interface in Action' below.

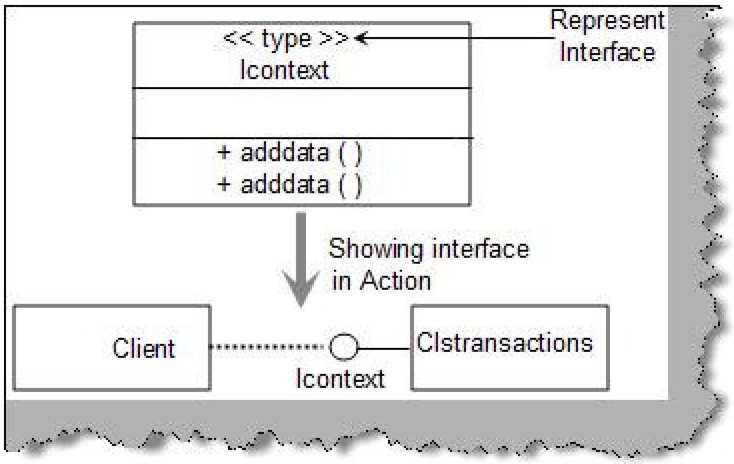


Figure: Interface in action

Abstract classes are represented by '{abstract}' as shown in figure 'Abstract classes in action'.

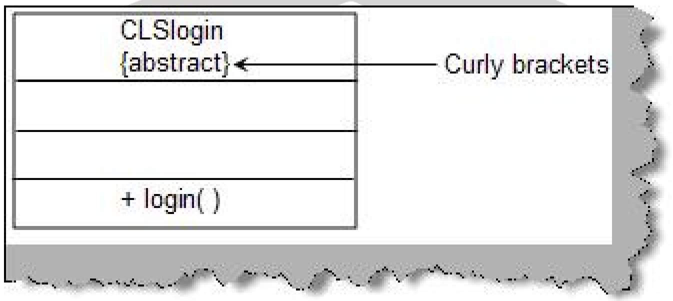


Figure: Abstract classes in action.

(B) How do we achieve generalization and specialization?

By using inheritance.

(I) Can you explain object diagrams in UML?

Class represents shows the static nature of the system. From the previous question you can easily judge that class diagrams shows the types and how they are linked. Classes come to live only when objects are created from them. Object diagram gives a pictorial representation of class diagram at any point of time. Below figure 'Object diagram' shows how a class looks in when actual objects are created. We have shown a simple student and course relationship in the object diagram. So a student can take multiple courses. The class diagram shows the same with the multiplicity relationship. We have also shown how the class diagram then looks when the objects are created using the object diagram. We represent object with Object Name: Class Name. For instance in the below figure we have shown 'Shiv : ClsStudent' i.e 'Shiv' is the object and 'ClsStudent' the class. As the objects are created we also need to show data of the properties, the same is represented by 'PropertyName=Value' i.e. 'StudentName=Shiv'.

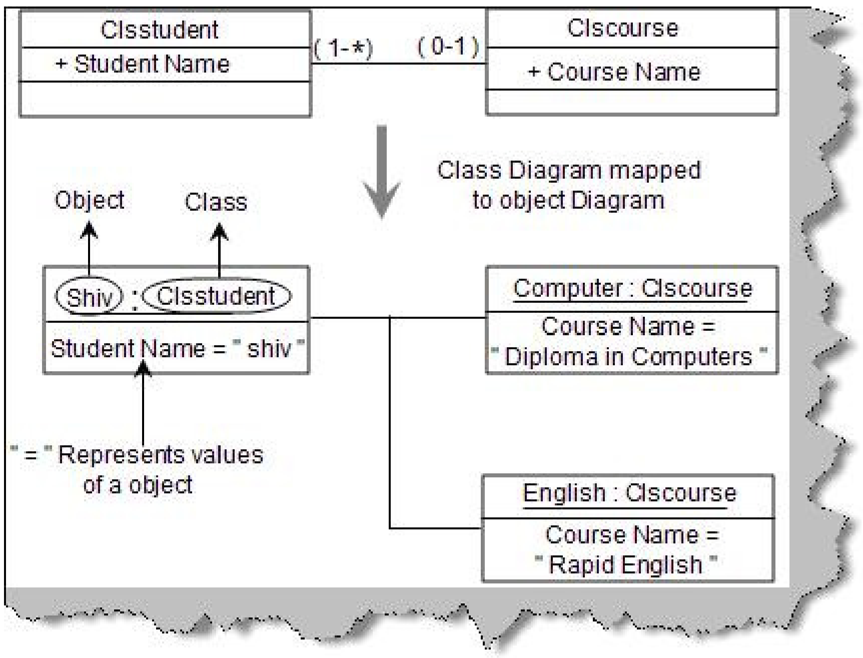


Figure: Object diagrams

The diagram also states that 'ClsStudent' can apply for many courses. The same is represented in object diagram by showing two objects one of the 'Computer' and the other of 'English'.

Note: Object diagrams should only be drawn to represent complicated relationship between objects. It's possible that it can also complicate your technical document as lot. So use it sparingly.

(I) Can you explain sequence diagrams?

Sequence diagrams

Sequence diagram shows interaction between objects over a specific period time. Below figure 'Sequence diagram' shows how a sequence diagram looks like. In this sequence diagram we have four objects 'Customer','Product','Stock' and 'Payment'. The message flow is shown vertically in waterfall manner i.e. it starts from the top and flows to the bottom. Dashed lines represent the duration for which the object will be live. Horizontal rectangles on the dashed lines represent activation of the object. Messages sent from a object is represented by dark arrow and dark arrow head. Return message are represented by dotted arrow. So the figure shows the following sequence of interaction between the four objects:

- Customer object sends message to the product object to request if the product is available or not.
- Product object sends message to the stock object to see if the product exists in the stock.
- Stock object answers saying yes or No.
- Product object sends the message to the customer object.
- Customer object then sends a message to the payment object to pay money.
- Payment object then answers with a receipt to the customer object.

One of the points to be noted is product and stock object is not active when the payment activity occurs.

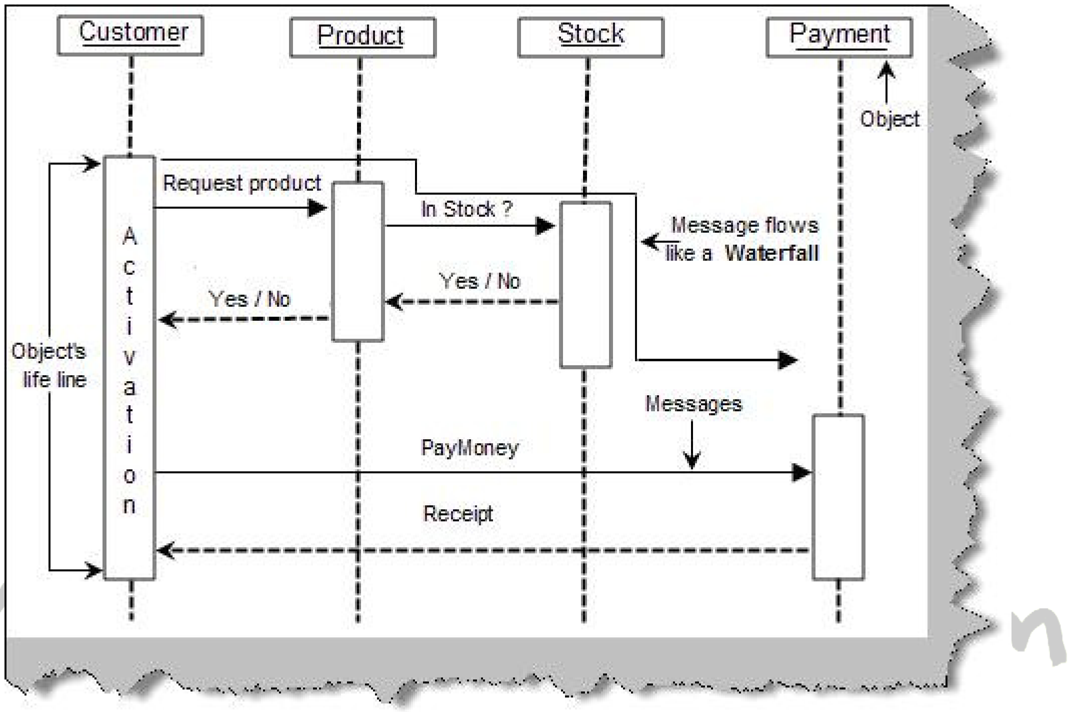


Figure: Sequence diagram

Messages in sequence diagrams

There are five different kinds of messages which can be represented by sequence.

Synchronous and asynchronous messages

Synchronous messages are represented by a dark arrow head while asynchronous messages are shown by a thin arrow head as shown in figure 'Synchronous and Asynchronous'.

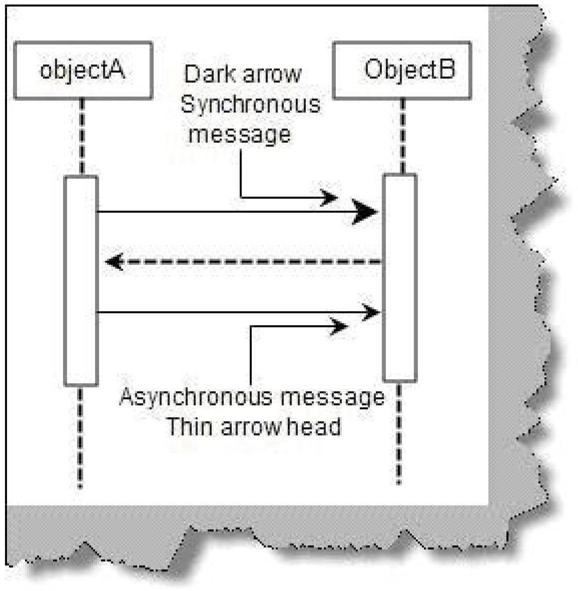


Figure: Synchronous and Asynchronous

Recursive message

We have scenarios where we need to represent function and subroutines which are called recursively. Recursive means the method calling himself. Recursive messages are represented by small rectangle inside a big rectangle with an arrow going from the big rectangle to the small rectangle as shown in figure 'Recursive message'.

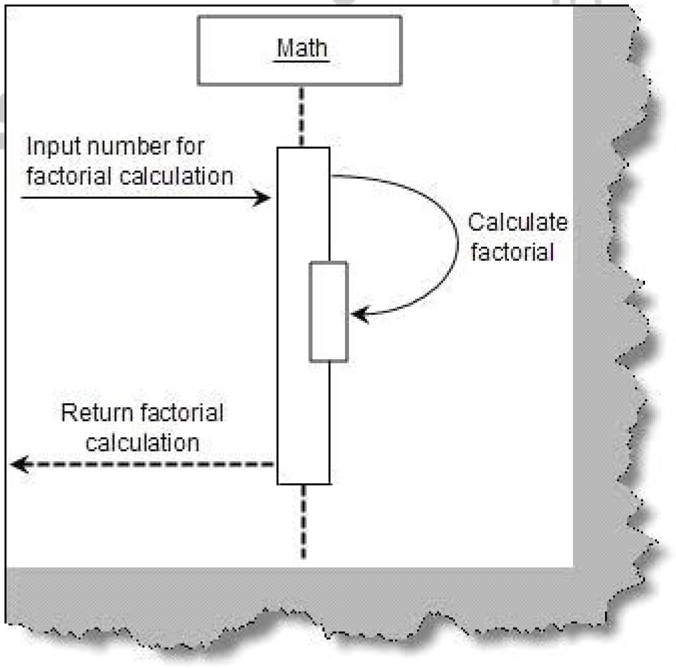


Figure: Recursive message

Message iteration

Message iteration represents loops during sequences of activity. Below figure 'message iteration' shows how 'order' calls the 'orderitem' objects in a loop to get cost. To represent loop we need to write 'For each <<object name>>'. In the below figure the object is the 'orderitem'. Also note the for each is put in a box to emphasize that it's a loop.

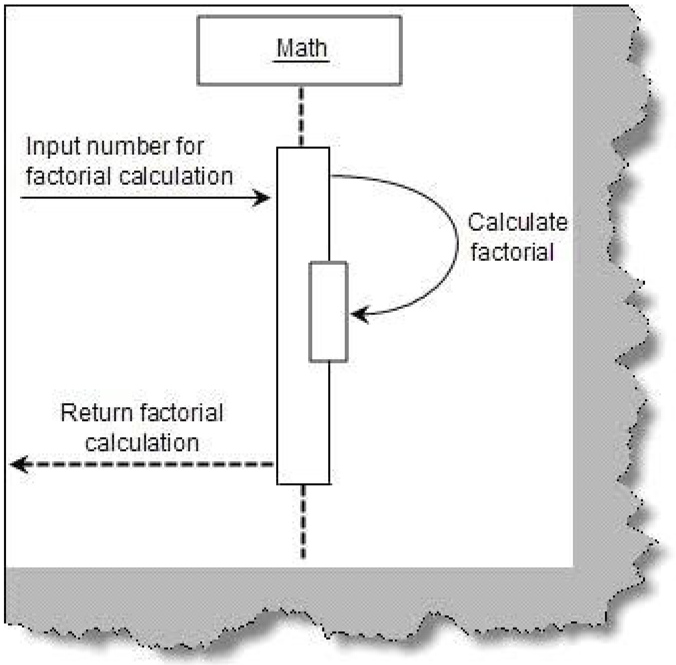

Math
Input number for factorial calculation
Calculate factorial
Return factorial calculation

Figure: Message iteration

Message constraint

If we want to represent constraints it is put in a rectangle bracket as shown in figure 'message constraint'. In the below figure 'message constraint' the 'customer' object can call 'book tickets' only if the age of the customer is greater than 10.

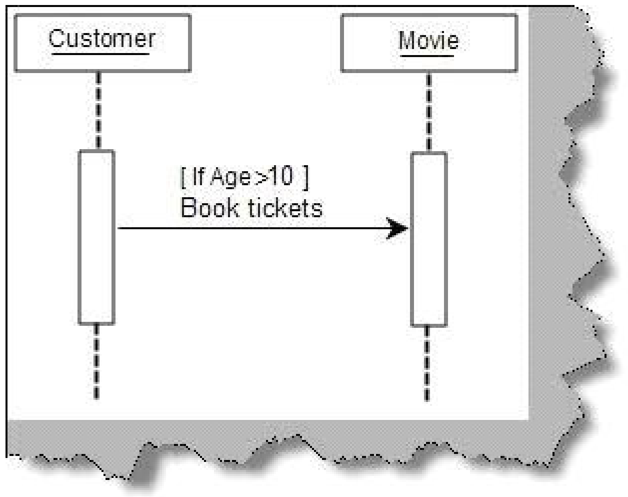


Figure: Message constraint

Message branching

Below figure 'message branching' shows how 'customer' object have two branches one is when the customer calls save data and one when he cancels the data.

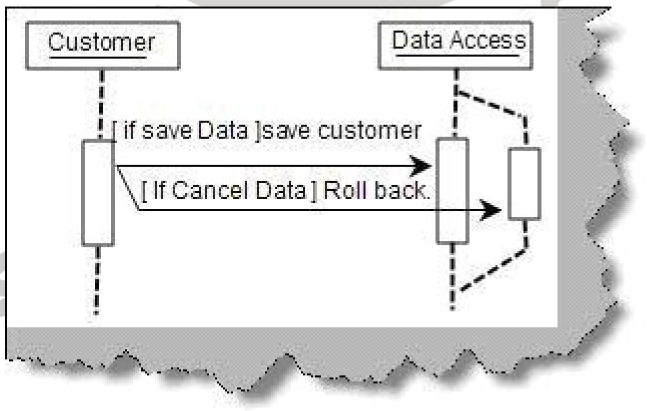


Figure: Message branching

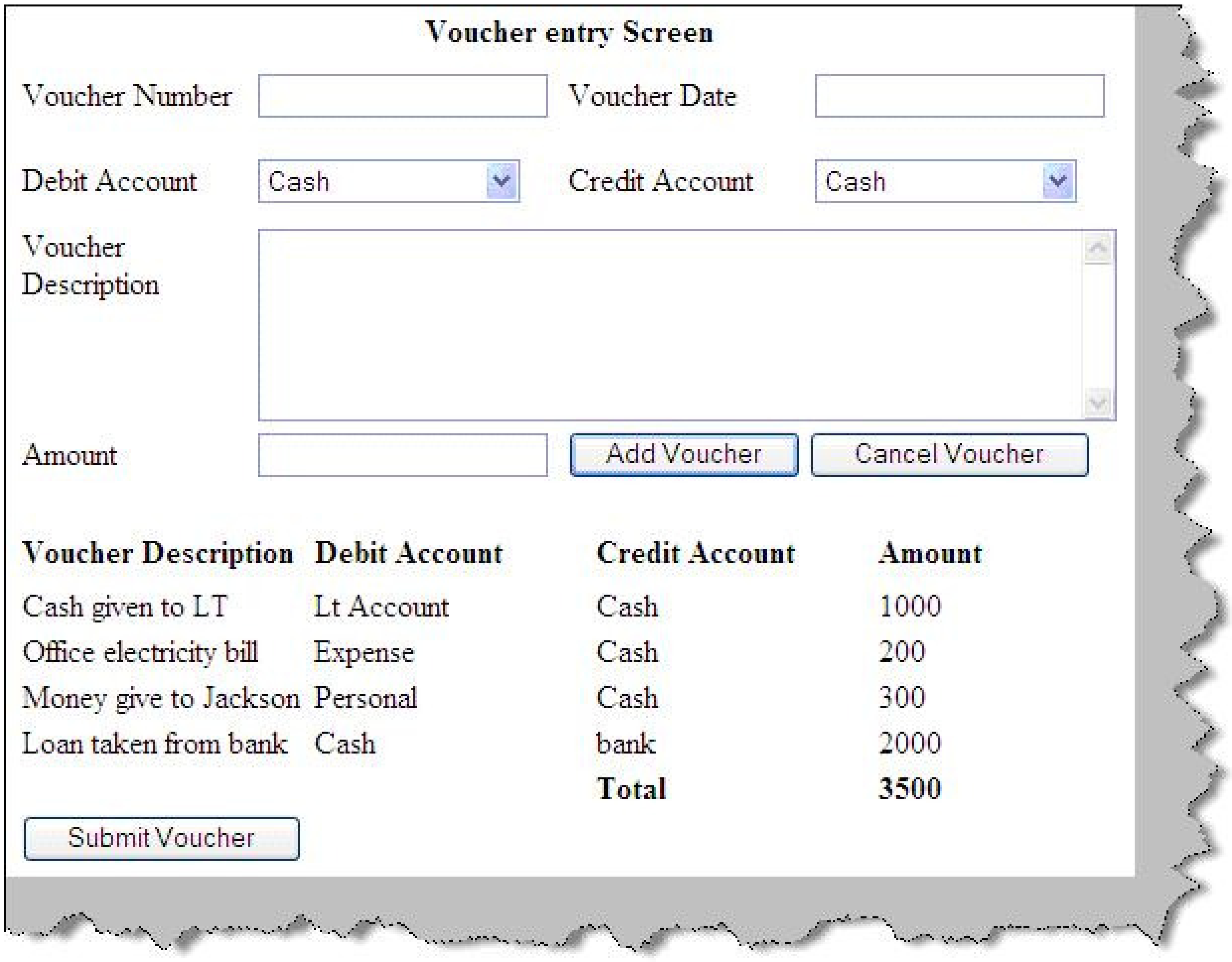
Doing Sequence diagram practically

Let's take a small example to understand sequence diagram practically. Below is a simple voucher entry screen for accounts data entry. Following are the steps how the accountant will do data entry for the voucher:-

- Accountant loads the voucher data entry screen. Voucher screen loads with debit account codes and credit account codes in the respective combo boxes.

- Accountant will then fill in all details of the voucher like voucher description, date, debit account code, credit account code, description, and amount and then click 'add voucher' button.
- Once 'add voucher' is clicked it will appear in the voucher screen below in a grid and the voucher entry screen will be cleared and waiting for new voucher to be added. During this step voucher is not added to database it's only in the collection.
- If there are more vouchers to be added the user again fills voucher and clicks 'add voucher'.
- Once all the vouchers are added he clicks 'submit voucher' which finally adds the group of vouchers to the database.

Below figure 'Voucher data entry screen' shows pictorially how the screen looks like.



Voucher entry Screen

Voucher Number		Voucher Date	
Debit Account	Cash	Credit Account	Cash
Voucher Description			
Amount		Add Voucher	Cancel Voucher

Voucher Description	Debit Account	Credit Account	Amount
Cash given to LT	Lt Account	Cash	1000
Office electricity bill	Expense	Cash	200
Money give to Jackson	Personal	Cash	300
Loan taken from bank	Cash	bank	2000
		Total	**3500**

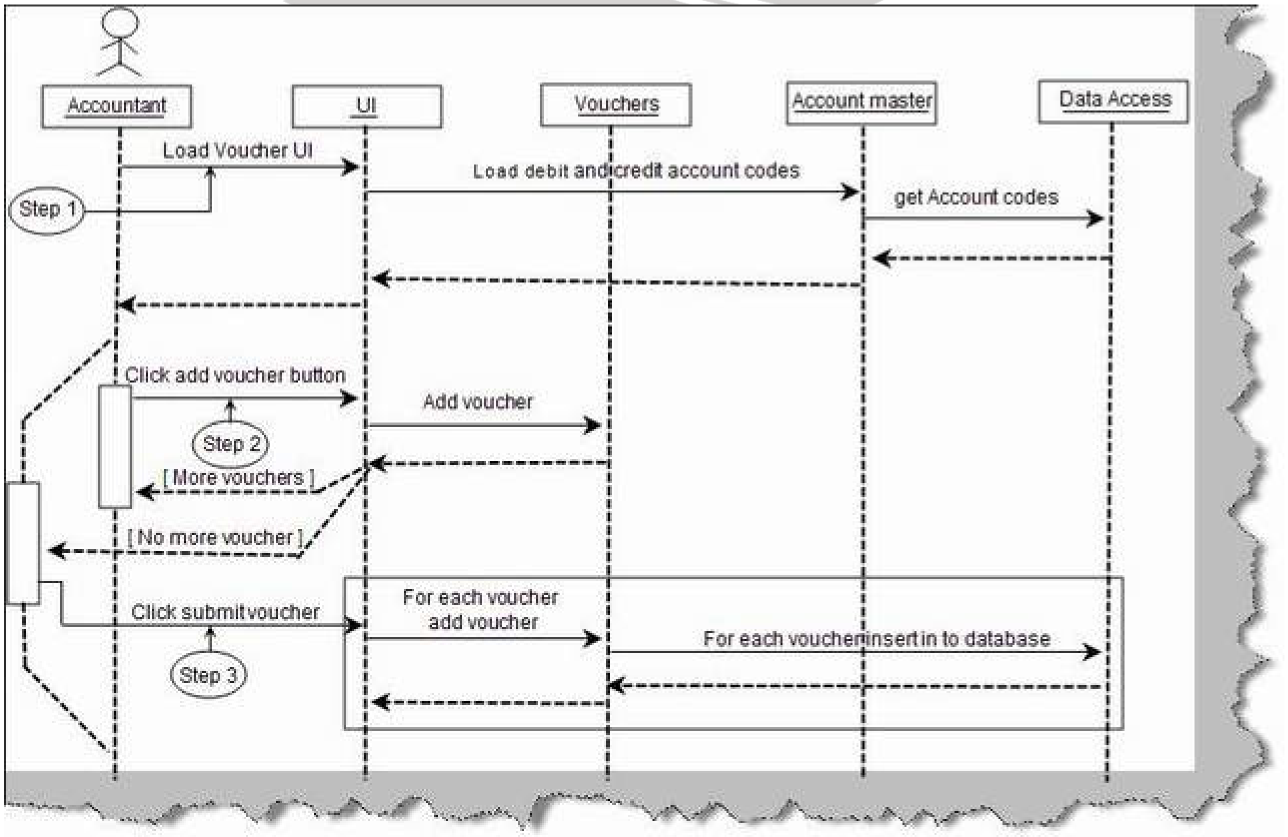
Figure: Voucher data entry screen

Figure 'Voucher data entry sequence diagram' shows how the sequence diagram looks like. Below diagram shows a full sequence diagram view of how the flow of the above screen will flow from the user interface to the data access layer. There are three main steps in the sequence diagram, let’s understand the same step by step.

Step 1:- The accountant loads the voucher data entry screen. You can see from the voucher data entry screen image we have two combo boxes debit and credit account codes which are loaded by the UI. So the UI calls the ‘Account Master’ to load the account code which in turn calls the data access layer to load the accounting codes.

Step 2:- In this step the accountant starts filling the voucher information. The important point to be noted in this step is that after a voucher is added there is a conditional statement which says do we want to add a new voucher. If the accountant wants to add new voucher he again repeats step 2 sequence in the sequence diagram. One point to be noted is the vouchers are not added to database they are added in to the voucher collection.

Step 3:- If there are no more vouchers the accountant clicks submit and finally adds the entire voucher in the database. We have used the loop of the sequence diagram to show how the whole voucher collection is added to the database.

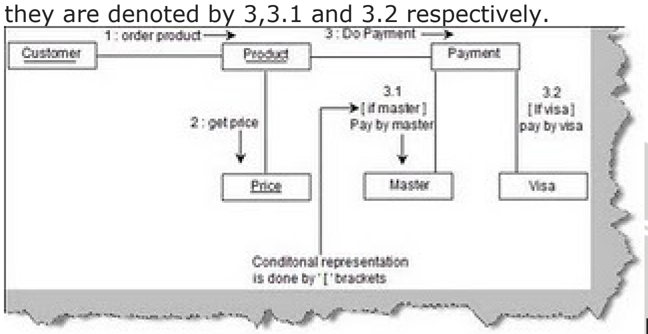


(I) Can you explain collaboration diagrams ?

Collaboration diagrams

Collaboration diagrams provide the same information as shown by sequence diagram but they show it in a different way. In sequence diagram we pay more attention to the time and sequence order, but in collaboration diagram we pay more emphasis on the interaction messages between the objects.

Figure 'Collaboration diagrams' shows how a collaboration diagram looks like. Below are some points you can easily pick up looking at the figure:-
• Objects are represented in rectangle.
• Messages are represented by an arrow and sequence number. For instance in figure 'collaboration diagrams' we can see the 'order product' has a arrow which shows that the message is sent from the customer object to the product and '1' represents that it's the first messages.
• Conditional statements are denoted by square brackets '['.
• We can also group sequence numbers by grouping them using decimals. For instance 'Pay by master' and 'Pay by Visa' are all grouped in to message sequence number '3' ('Do payment'). So they are denoted by 3,3.1 and 3.2 respectively.


1 : order product
3 : Do Payment
Customer
Product
Payment
2 : get price
3.1 [if master] Pay by master
3.2 [If visa] pay by visa
Price
Master
Visa
Conditonal representation is done by '[' brackets
Figure: - Collaboration diagrams

You can represent the for each loop using the 'for each' keyword as shown in below figure 'for each in collaboration'.

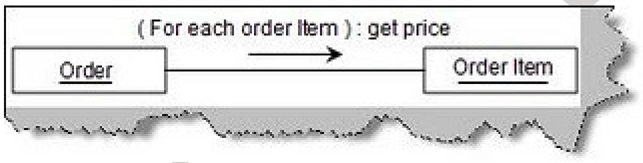


Figure: - For each in collaboration

Note: - Try drawing a collaboration diagram for the same voucher data entry screen. Sequence and collaboration diagrams are also called as iteraction diagrams.

(I) Can you explain activity diagrams?

Activity diagrams
Activity diagram are used to capture complicated process flows in a system. Below figure 'Activity' shows a simple activity diagram. Some of the points which you can easily note from the activity diagram figure are:-

• Start of an activity is denoted by a dark circle.
• End of an activity is denoted by a dark circle inside a white circle.
• Activities are denoted by simple oval rectangles.

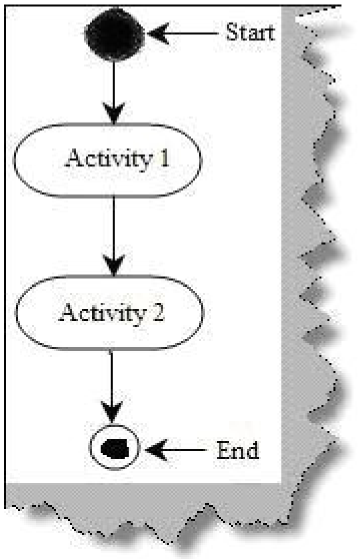


Figure: - Activity

A decision in activity diagram is as shown figure 'Decision in Activity Diagram'. Figure shows the condition in a '[' (Square bracket). So the first activity is 'Check Age', if the age is greater than 16 then we can go for 'Adult Movie' activity or else we need to execute the 'Kids Movie' activity.

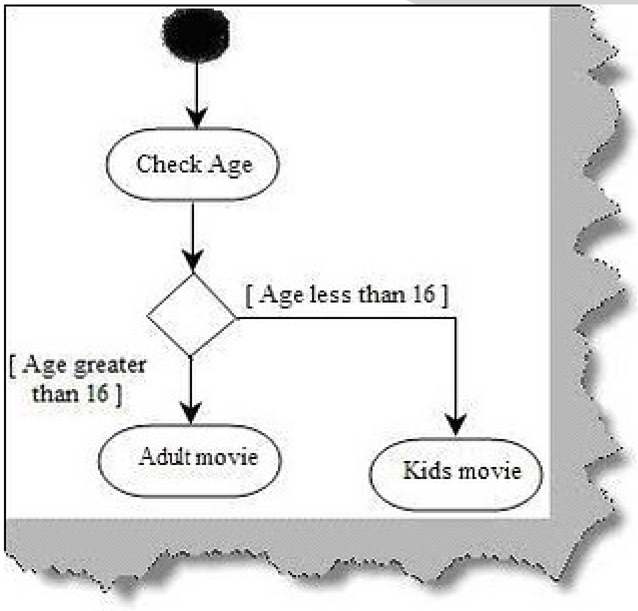


Figure: - Decision in Activity Diagram

There are situations in project where we need to execute two parallel activities in a project. A solid bold line represents from where the activities will split and execute in parallel and then again a solid line is where the parallel activities will meet. For instance in the below figure 'Parallel Processing' we can see how 'Make lemon juice' and 'Make Ginger Juice' activities are executed in parallel.

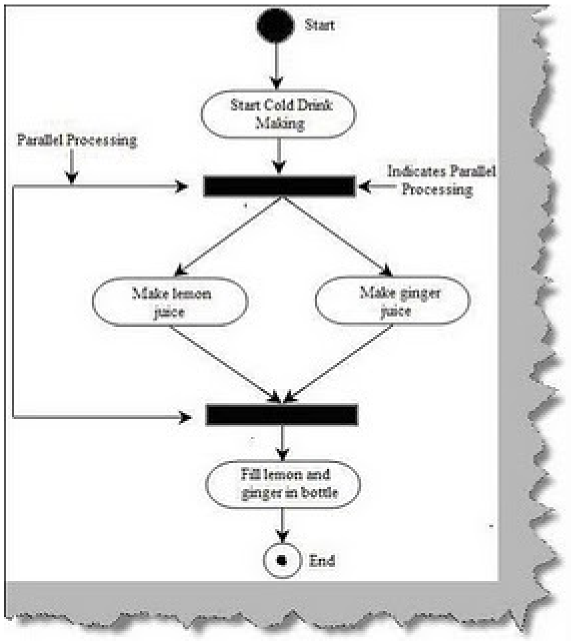


Figure: - Parallel Processing

In big and complex activity diagrams it's very difficult to figure out which object is responsible for which activities. This problem is solved by 'Swimlanes'. Consider the below figure 'Without Swimlanes'. The whole activity diagram looks very complex and it's very difficult to figure out which object is responsible for which activity.

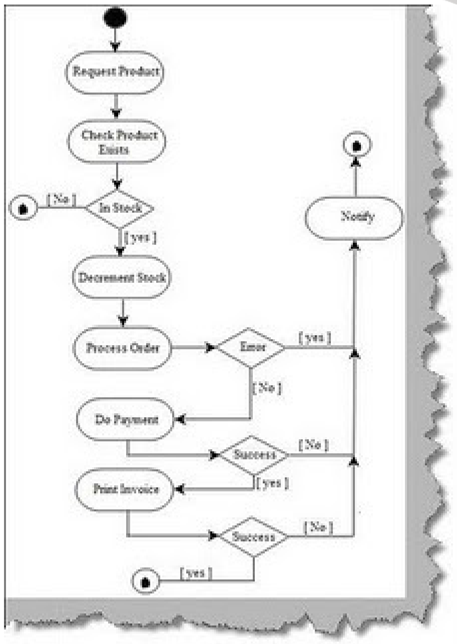


Figure: - Without Swimlanes

Now see the below figure 'With Swimlanes' we have partitioned the activity to the respective objects like Customer, Stock, Order processing, Payment and Invoice. These partitions are termed as 'Swimlanes' , so if you feel that the activity diagram is complex think about using 'Swimlanes' it can really make your activity diagram readable.

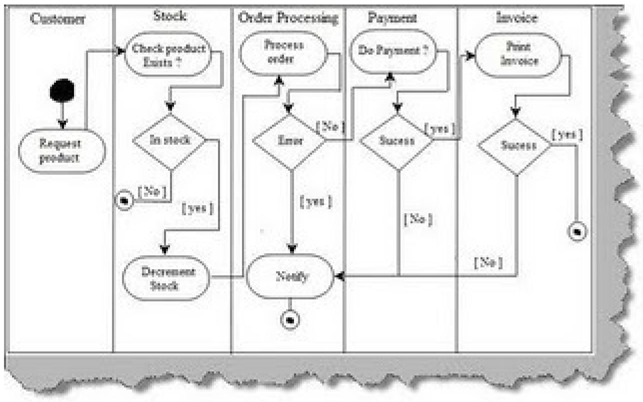


Figure: - With Swimlanes

(I) What is state chart diagram?

State Chart Diagram

State diagram depicts different states that an object goes through during their life cycle. State diagram depicts how an object responds to events. We think state diagrams as optional and should be used if your project has scenarios where the object goes through lot of complicated states and transitions. If your project does not have such kind of scenarios then sequence, collaboration or activity would be sufficient to meet your needs. So all objects have states and an object moves from one state to other state when there is some event or transition.

There are three important things when we consider state of an object event, guard and action. Let's first define these three things: - .

Action: - Action triggers an object state from one state to another.

Event: - Event triggers the action.

Guard: - Guard is condition on which it evaluates which action to be triggered.

These three things are principle component of state chart diagrams. Below figure 'Types of event and action' shows how they are represented in state chart diagrams.

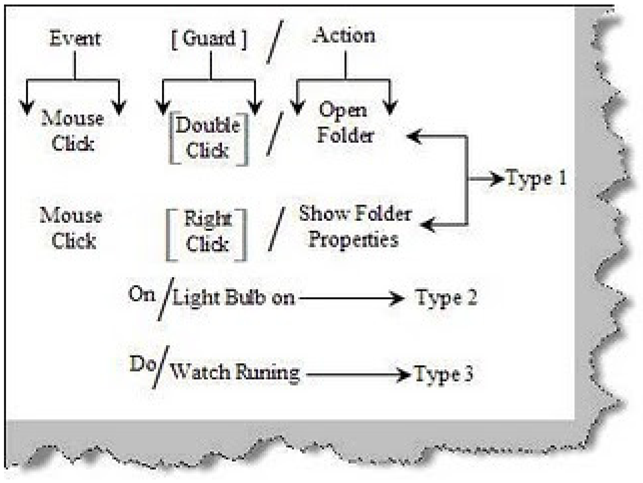


Figure: - Types of Event and Action

There are three ways by which we can represent the same.

Type 1:- This methodology of writing is used when we need to map an event with action using a guard. For instance in the above figure 'Types of Event and Action' shows the event à mouse click, guard à double click and the resulting action à open folder.

Type 2:- The guard is optional. For instance sometimes we only have events and actions, i.e. with out the guard. For instance when the even 'On' happens the action is that 'Light Bulb is on'.

Type 3:- This type of representation shows an infinite loop for an action. For instance the 'Watch will be running' infinitely during a state, as shown in figure 'Type of Event and Action'.

Now that we know how to write event, actions and guard, let's see how state diagram looks like. Below figure 'State example' shows how a state looks like. It's an oval rectangle as shown below. In order to show a transition we need to show an arrow from one state to other state as shown in figure 'State example'.

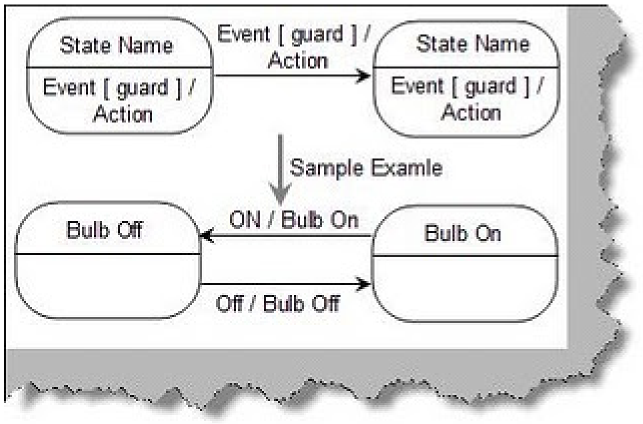


Figure: - State example

Below figure 'Sample state chart' shows a simple state chart diagram. Some points which are immediately visible from the diagrams are as follows:-

• A dark black arrow indicates start of a state chart diagram.
• A dark circle with a white circle outside indicates end of a state chart diagram.
• Circular rectangle indicates state while arrows indicate events / transitions.

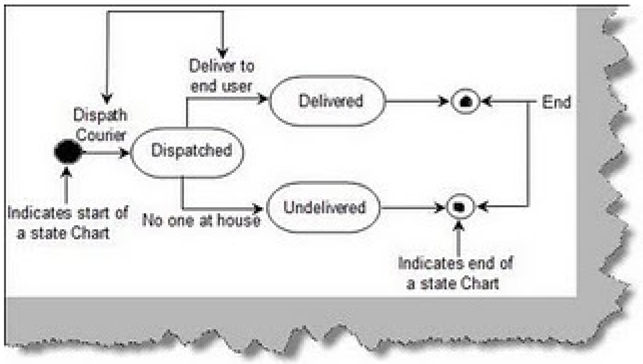


Figure: - Sample state chart

State is represented as shown in figure 'Basic element of state diagram'. It's a simple rectangle which is rounded. In the top section we give the state name. The below section is optional which has 'do/action'. It represents a long running activity when the object goes through this state.

(I) Can you explain stereotypes in UML?

Stereotypes are a way to define variations on existing UML model. This variation is brought in to place to extend UML in a consistent manner. They are displayed in double less than and double greater than sign with a simple text as shown below. The below figure shows at the left hand side a class diagram with out stereo types while the right hand side shows with stereo types. You can easily make out how the class diagram is readable with stereo types. For instance the 'Customer()' can be mistaken for a method but we have clarified that it's a constructor by using stereo types.

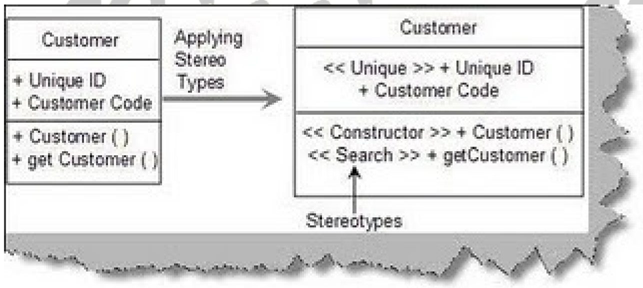


Figure: - Stereotypes

Below are some of the commonly used stereo types while writing UML.

<<Application>>:- Used to represent a UI system in a application.
<<Database>> :- represents a database in a application.
<<Table>> :- A table with in database.
<<Library>> :- A reusable library or function.
<<File>> :- Physical file on a folder.
<<Executable>> :- A software component which can be executed.
<<Web services>> :- Represents a web service.

<<JDBC>> :- Java database connectivity , a JAVA API to connect to database.
<<ODBC>> :- Open database connectivity , a Microsoft API to connect to database.

(I) Can you explain package diagrams?

Packages are like folders in a system which allows you to logically group UML diagrams. They make complex UML diagram readable. In actual projects they are used to logically group use cases and classes. So we can say there are two types of package diagrams one is class package diagram and other is use case package diagram. Package diagram depict a high level of overview for class and use cases.

Class package diagram: - Class package diagram are used to logical group classes. You can think that package technically map to 'Package' in JAVA and 'Namespaces' in C# and VB.NET. Packages are denoted by small rectangle on a big rectangle as shown in figure 'Package diagram'. One of the points to be noted is the stereotypes. We have numbered the figure so that we can understand it better.

1 – We are using the MVC (Model View controller) framework. So we have denoted this package using the << Framework >> stereo type. Refer the commonly used stereo type table discussed in the previous sections.

2 and 3 – 'Book tickets' is the second package which inherits from the MVC model. The stereo type is '<<application>>' which means it's a user interface.

4 – A simple line shows a link between two classes stating that one class package uses the other class package.

5 – This package is collection of the booking engine library.

6 – The final package is the database.

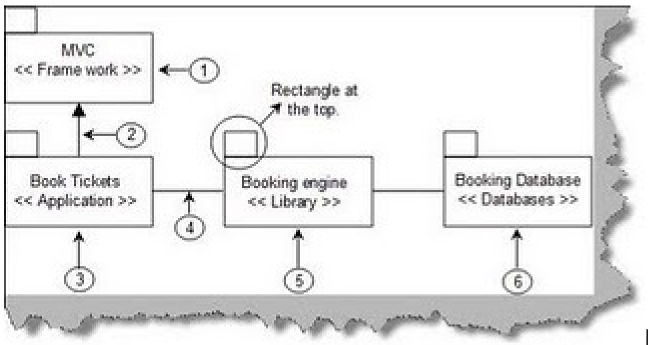


Figure: - Package diagram

As said before packages are nothing but collection of logical classes or any UML entity. We have shown the detail of the above package diagram. We should restrict from using package diagram for showing the in depth architecture as it can become very complicated. For instance the below diagram 'Detail Package diagram' shows how complicated it can look if use the package diagram to show in depth architecture. To avoid complication its good to only draw an over all diagram as shown in ‘Package diagram’.

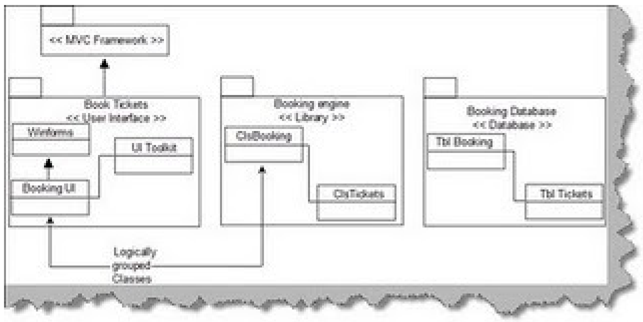


Figure: - Detail Package diagram

Use case package diagram: - The way we have logically grouped classes we can also use the package diagram to logically group use cases. Below figure shows how a use case package diagram looks like.

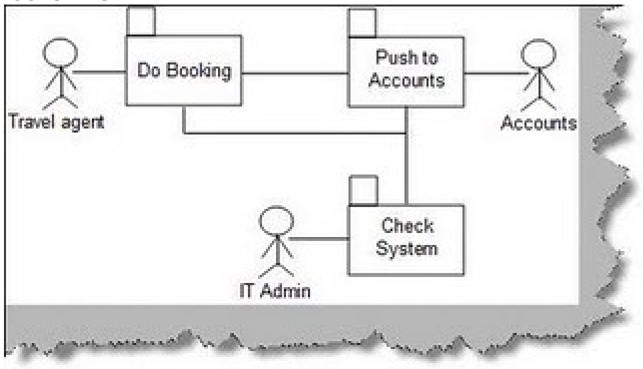


Figure: - Use Case Package

(B) Can you explain component diagrams?

Component diagrams achieve the same objective like package diagrams. They show the dependencies among software components. Below figure ‘Component diagram’ shows a sample component diagram for simple data entry application which uses a web service to interact with the database. We have numbered the steps to understand the details of the component diagram.

1 – Two rectangles are shown to represent a component of a system.
2 – Stereo types are used to denote what kind of system it represents.
3 – A line with a circle denotes an interface by which the external world can interact with the component. For instance in the figure we have represented a ‘Customer Web service’ which can is interacted by using XML.

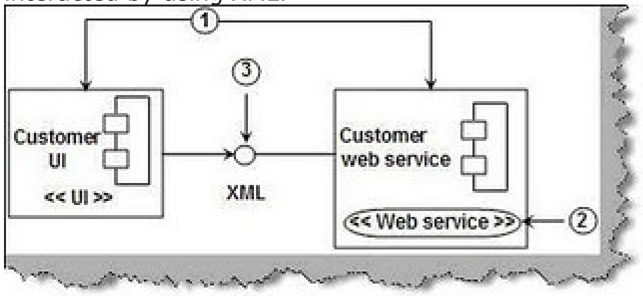


Figure: - Component Diagram

(B) Can you explain deployment diagrams?

Deployment diagrams represents an overall static view of how software and hardware nodes in the application. They show what the hardware is and which components are installed on which hardware. In deployment diagram we represent the hardware with a solid box and simple underlined text description showing which hardware is it. You can have more than one component on a single hardware. So the browser is an application UI which resides on the workstation computer and the database and web server resides on the web server hardware. Deployment diagram can be more complex with firewalls, payment gateways, PDA devices, VPN etc.

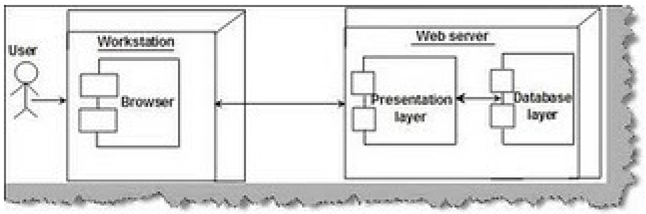


Figure: - Deployment diagram

(I) Can you explain how UML flows in actual project?

In actual projects we do not draw all the diagrams. Every UML diagram is made for a purpose. It completely depends on what's the nature of the project. In short you should ask yourself questions like, is this diagram important, what's my need etc. So below is a flow which you can follow in your project, again as we said it depends on what kind of scenario you want to depict.

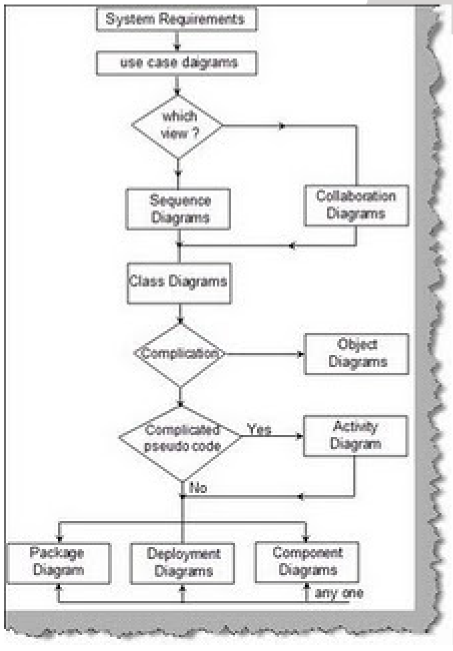


Figure: - UML flow in actual projects

• The first step is to derive use cases from the requirement documents.
• Once use cases are derived we need to decide the messages which will flow in the system. This can be done using interaction diagrams. If you need to know the object creation life times we use the sequence diagram and if we want to concentrate on the messages we use the collaboration

diagrams. So depending on scenario we need to make a choice which diagram we need to draw.
• Now that we are clear about messages we can draw class diagrams to depict the static part of the project i.e. classes.
• If we find any complicated class relationships we draw object diagrams.
• If we need to depict any complicated code we need to represent same with a activity diagram.
• Finally to give an overview of the project we can use package diagram, component or deployment diagram. As said before we can use combination of component and deployment diagram to give a overview of the architecture.

And all the questions from previous university question papers.

All the Best!!!