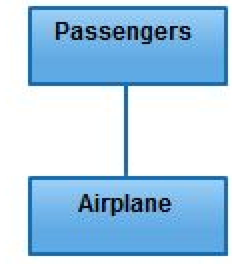
Object Oriented System Design (KCS-056)

Question Bank based on Unit-1 and Class Diagram of Unit-2

Section-A (Short answer based questions)

a) Is association class same as ordinary class? Explain with an example.

For most purposes, an **association class** behaves in the **same** manner as though it was an **ordinary class** with two * -- 1 **associations** to the participating **classes**.



Association is a broad term that encompasses just about any logical connection or relationship between classes. For example, passengers and airplane may be linked as above.

b) Difference between private & protected visibility mode.

Private visibility mode: The **private** derivation means, the derived class can access the public and **private** members **of the** base class privately.
Protected visibility mode: The **protected** derivation means that the derived class can access the public and **private** members **of the** base class protected.

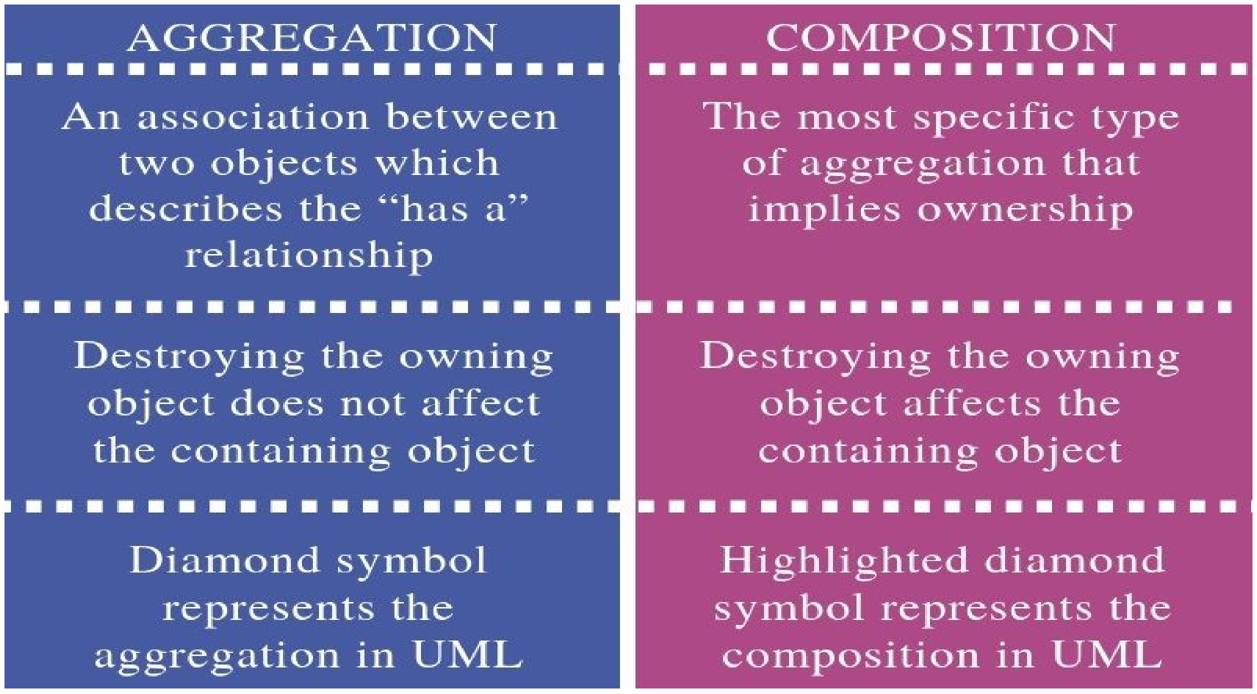
c) Difference between ordered and sequences in class diagram.

- **ordered** means that the objects are ordered in the collection of associated elements. This means that there is a mapping between a positive integer to the elements in the collection. It doesn't say more, and in particular, it doesn't say if the elements in the associated collection are unique or not.
- **sequence** or **seq** means that the association implements an *ordered bag*, that is an ordered collection of elements where the same elements may occur several times (example: a trip made of a sequence of towns : *New-York -> Chicago -> New-York -> Los-Angeles -> New-York*).

And longer

In an UML model, association-ends have two possible properties: isOrdered and isUnique. The ordered implies that isOrdered = true (explicitly specified). For sequence, you can deduce from the text that it's isOrdered = true and isUnique = false.

d) Difference between Composition & Aggregation.

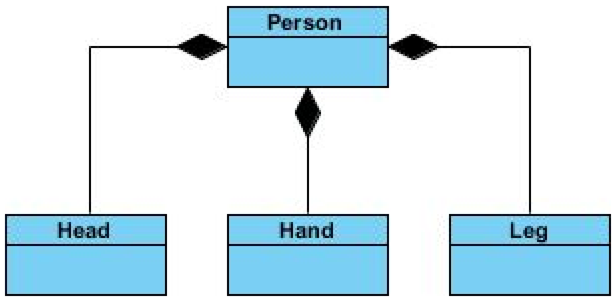


AGGREGATION	COMPOSITION
An association between two objects which describes the “has a” relationship	The most specific type of aggregation that implies ownership
Destroying the owning object does not affect the containing object	Destroying the owning object affects the containing object
Diamond symbol represents the aggregation in UML	Highlighted diamond symbol represents the composition in UML

- **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). Rooms don't exist separate to a House.

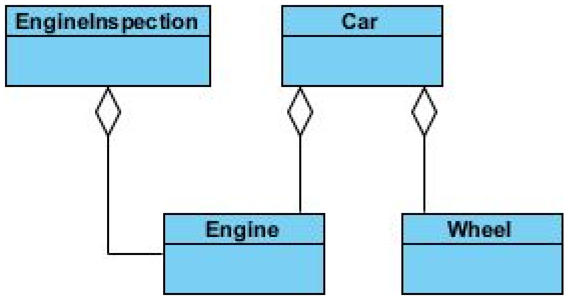
Composition Example:

We should be more specific and use the composition link in cases where in addition to the part-of relationship between Class A and Class B - there's a strong lifecycle dependency between the two, meaning that when Class A is deleted then Class B is also deleted as a result



Aggregation Example:

It's important to note that the **aggregation link doesn't state in any way that Class A owns Class B nor that there's 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 is usually used to stress the point that a Class A instance is not the exclusive container of a Class B instance, as in fact the same Class B instance has another container/s.



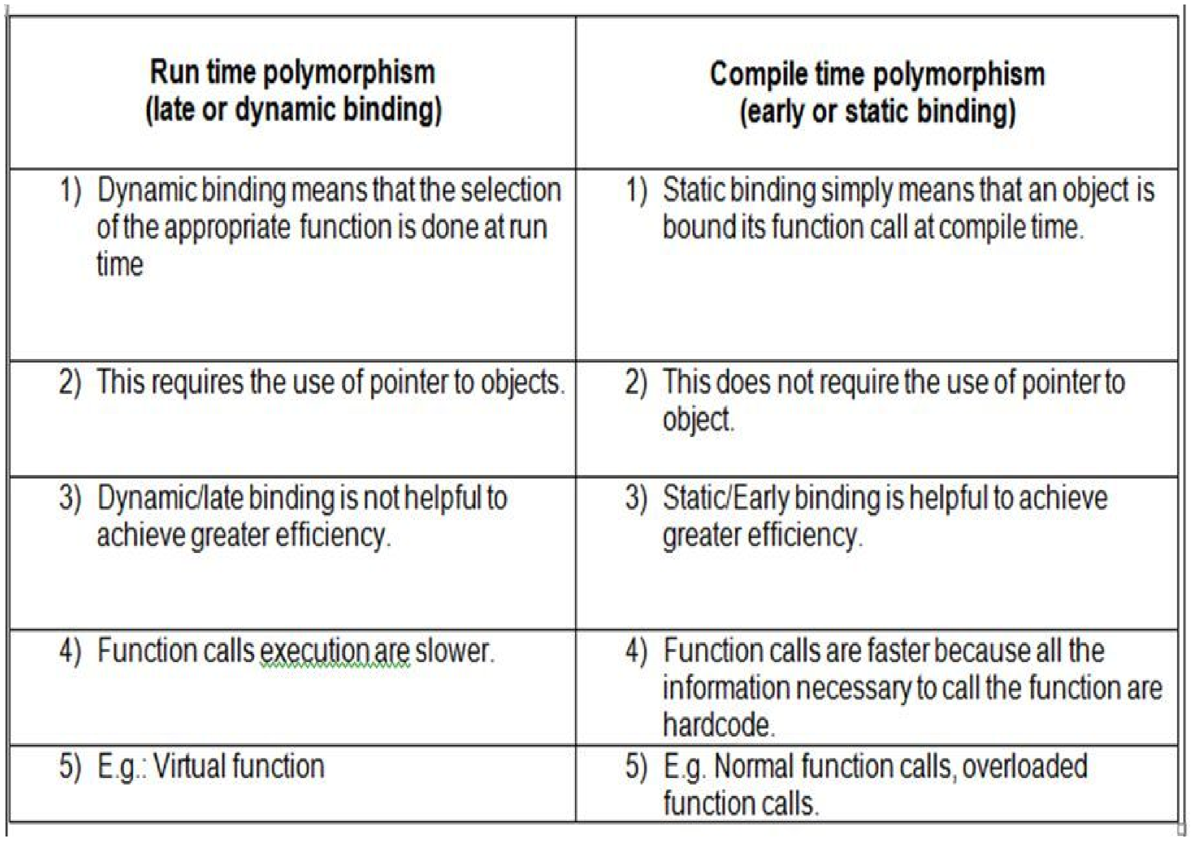
Summing it up -

To sum it up association is a very generic term used to represent when one class used the functionalities provided by another class. We say it's a composition if one parent class object owns another child class object and that child class object cannot meaningfully exist without the parent class object. If it can then it is called Aggregation.

e) Give UML notation for visibility.

In object-oriented design, there is a **notation** of **visibility** for attributes and operations. **UML** identifies four types of **visibility**: public, protected, private, and package. The +, -, # and ~ **symbols** before an attribute and operation name in a class denote the **visibility** of the attribute and operation.

f) Differentiate compile & run time polymorphism.



Run time polymorphism (late or dynamic binding)	Compile time polymorphism (early or static binding)
1) Dynamic binding means that the selection of the appropriate function is done at run time	1) Static binding simply means that an object is bound its function call at compile time.
2) This requires the use of pointer to objects.	2) This does not require the use of pointer to object.
3) Dynamic/late binding is not helpful to achieve greater efficiency.	3) Static/Early binding is helpful to achieve greater efficiency.
4) Function calls execution are slower.	4) Function calls are faster because all the information necessary to call the function are hardcode.
5) E.g.: Virtual function	5) E.g. Normal function calls, overloaded function calls.

g) State any two principles of modeling.

Principles of Modeling

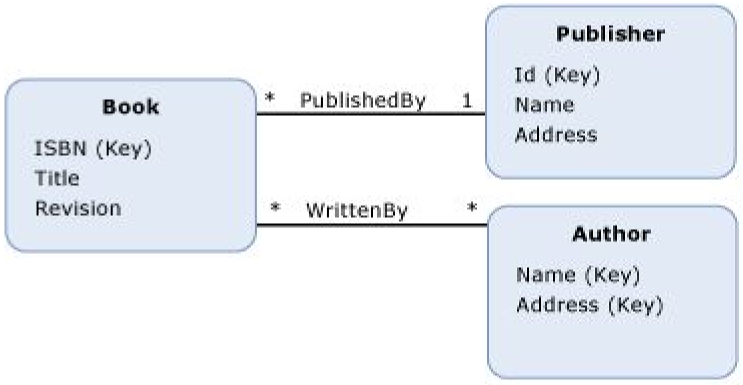
- **First Principle of modeling**
    - The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.
    - The right models will highlight the most nasty development problems.
    - Wrong models will mislead you, causing you to focus on irrelevant issues.
- **Second Principle of modeling**
    - Every model may be expressed at different levels of precision.
    - Degree of detail, depending on who is viewing it.
        - An analyst or end user - **issues of what**
        - A developer - **issues of how.**

h) Give examples of association end names.

(i) **Association end names**:
An association end name is a **name** that uniquely identifies **one end** of **an association**. It specifies a role of **an** object of a class which it plays in the **association**. **An association end name** is written next to the **association** line near the class that plays the role.

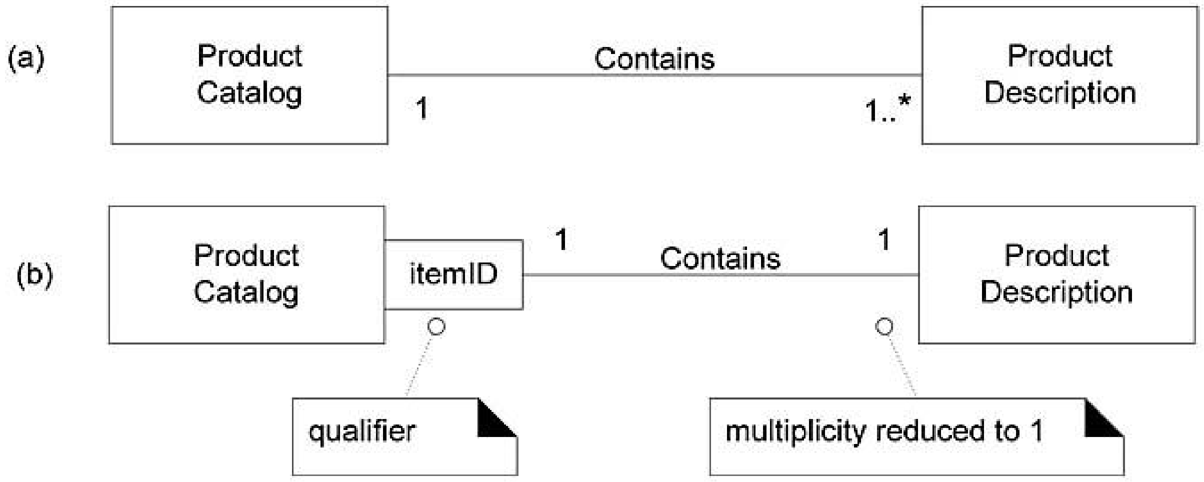
Example

The diagram below shows a conceptual model with two associations: PublishedBy and WrittenBy. The association ends for the PublishedBy association are the Book and Publisher entity types. The multiplicity of the Publisher end is one (1) and the multiplicity of the Book end is many (*), indicating that a publisher publishes many books and a book is published by one publisher.

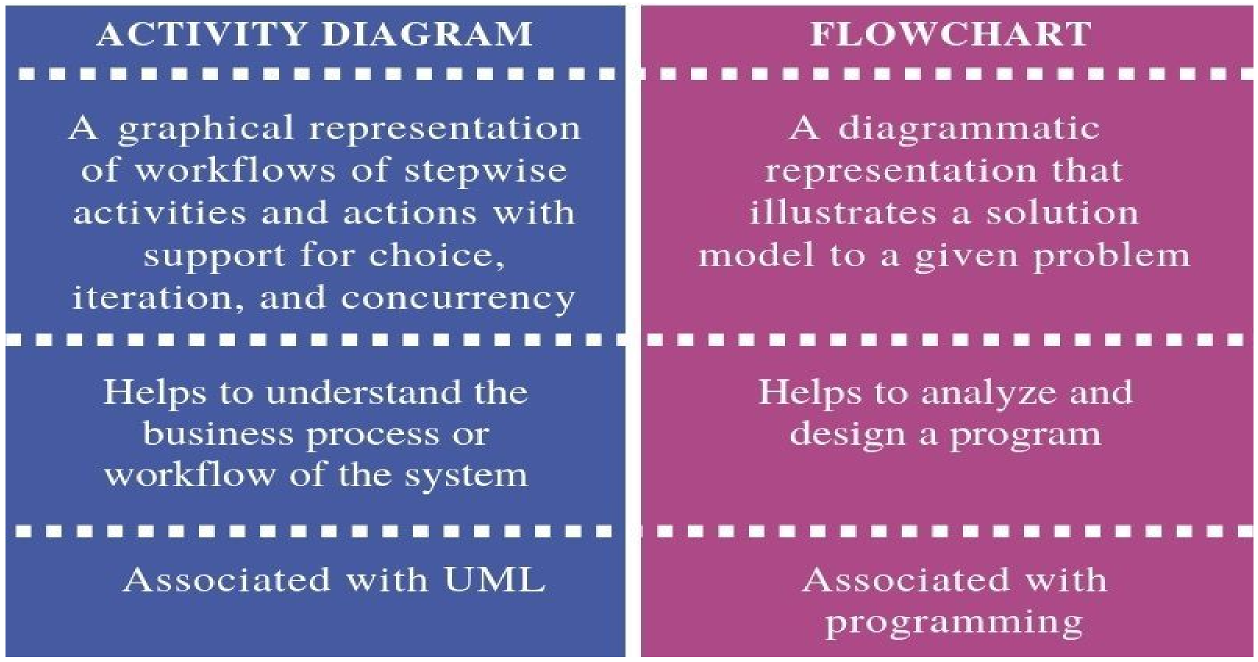


i) What is a qualified association?

A **qualified association** has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key. Informally, from a software perspective, it suggests looking things up by a key, such as objects in a HashMap.



j) Difference between Activity diagram & Flow chart



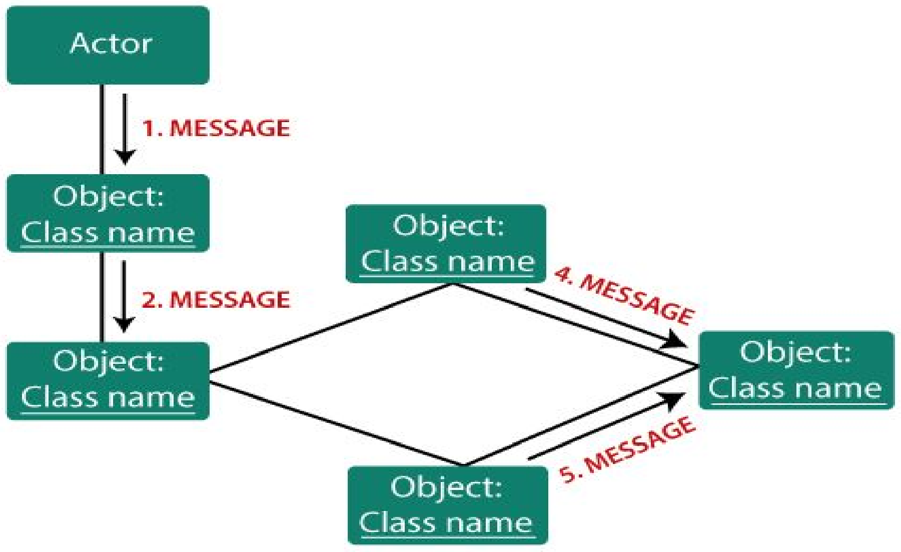
ACTIVITY DIAGRAM	FLOWCHART
A graphical representation of workflows of stepwise activities and actions with support for choice, iteration, and concurrency	A diagrammatic representation that illustrates a solution model to a given problem
Helps to understand the business process or workflow of the system	Helps to analyze and design a program
Associated with UML	Associated with programming

11.Define the following terms

a. collaboration diagram

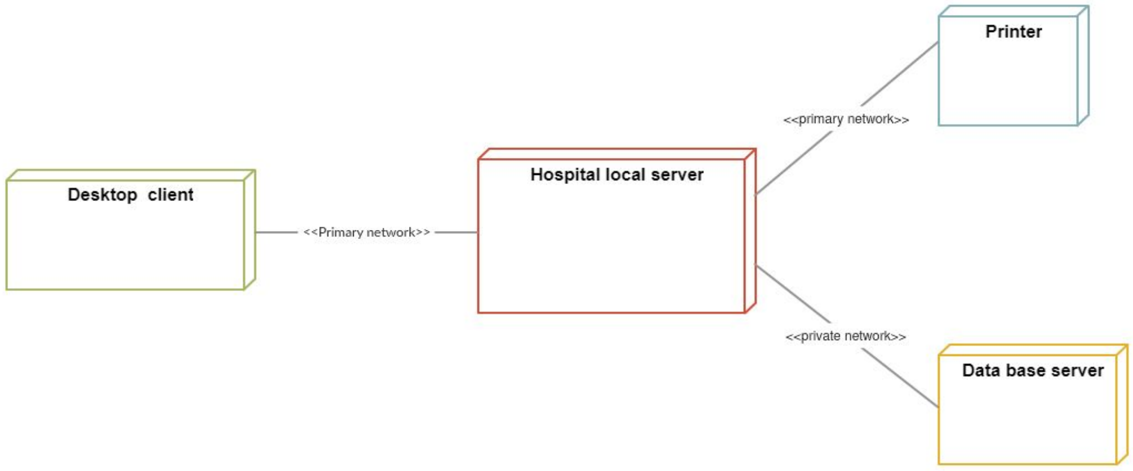
The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Components of a collaboration diagram

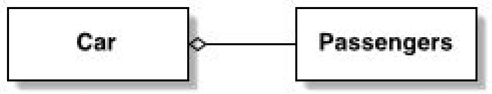


b. **Deployment diagrams** are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

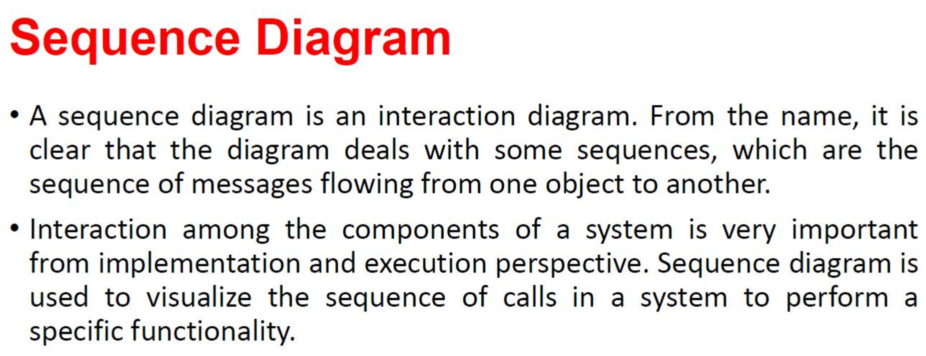


c. **Aggregation** is a specialized form of association between two or more objects in which each object has its own life cycle but there exists an ownership as well. **Aggregation** is a typical whole/part or parent/child relationship but it may or may not denote physical containment.



Aggregation: cars may have passengers, they come and go

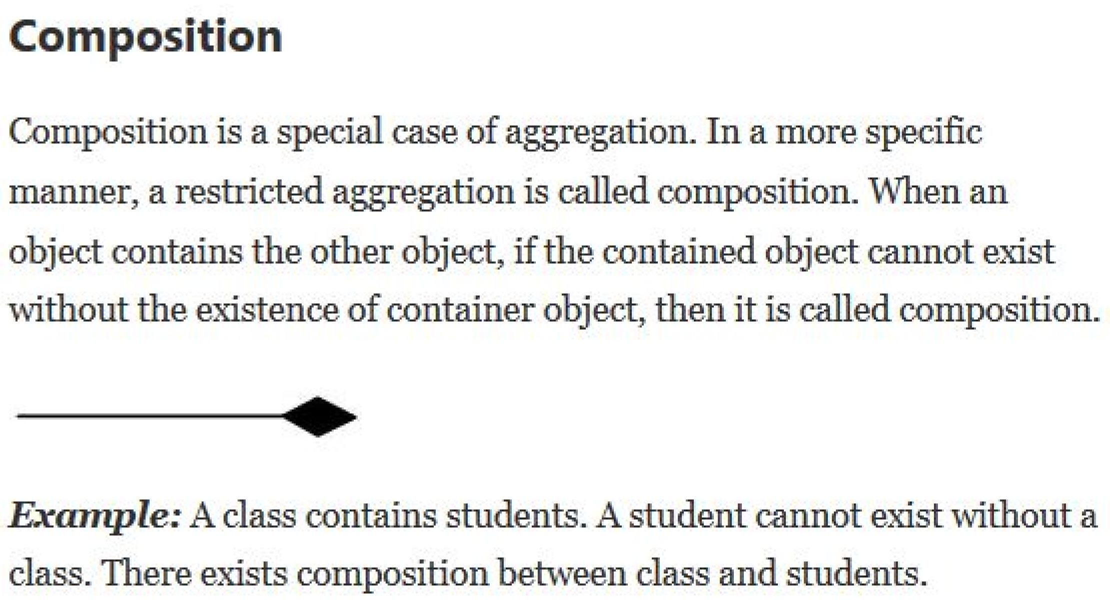
d. **Sequence Diagram**



Sequence Diagram

- A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.
- Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

e.



Composition

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

f. **Annotational things** can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. **Note -** It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.

g. **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**.

h. **Multiplicity** (which is optional) indicates how many objects of a class may relate to the other classes in an association. **Multiplicity** is shown as a comma-separated sequence of the following: Integer intervals. Literal integer values.

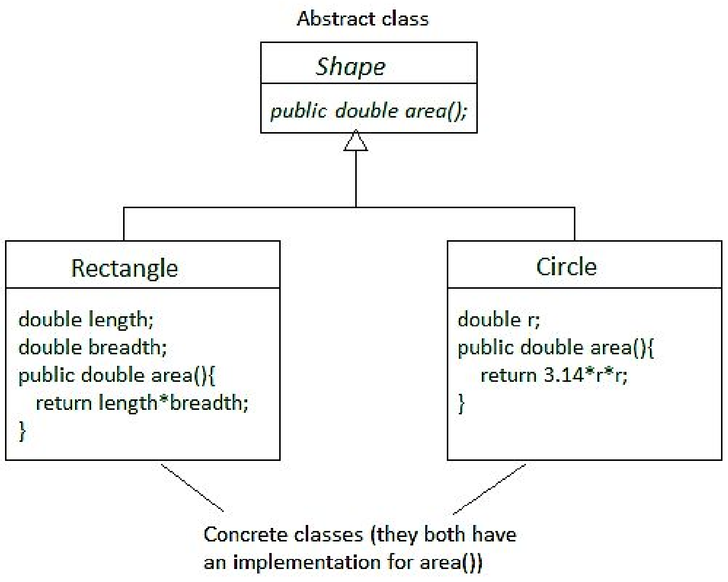
i. **Inheritance** is the sharing of attributes and operations (features) among classes based on a hierarchical relationship. Each subclass incorporates, or inherits, all the features of its superclass and adds its own unique features. Subclasses need not repeat the features of the superclass.

j. A **concrete class** is a class that has an implementation for all of its methods. They cannot have any unimplemented methods. It can also extend an abstract class or implement an interface as long as it implements all their methods. It is a complete class and can be instantiated.

   In other words, we can say that any class which is not abstract is a concrete class.

Necessary condition for a concrete class: There must be an implementation for each and every method.

Example: The image below shows three classes Shape, Rectangle and Circle. Shape is abstract whereas Rectangle and Circle are concrete and inherit Shape. This is because Rectangle and Circle implement area() method.



k. An **abstract class** is a **class** that is declared **abstract** —it may or may not include **abstract** methods. **Abstract classes** cannot be instantiated, but they can be subclassed.

l. **Encapsulation**
Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system.

m. **Polymorphism** is the ability to take on many different forms. It applies to both objects and operations. A polymorphic object is one whose true type hides within a super or parent class.

In polymorphic operation, the operation may be carried out differently by different classes of objects. It allows us to manipulate objects of different classes by knowing only their common properties.

Section-B (Medium answer based questions)

a) Briefly explain the following characteristics of object oriented systems: Identity, inheritance, encapsulation, polymorphism.

Identity
Identity means that data is quantized into discrete, distinguishable entities called objects. The white queen in a chess game are examples of objects. Objects can be concrete, such as a file in a file system, or conceptual, such as a scheduling policy in a multiprocessing operating system. Each object has its own inherent identity. In other words, two objects are distinct even if all their attribute values (such as name and size) are identical.

Inheritance is the sharing of attributes and operations (features) among classes based on a hierarchical relationship. Each subclass incorporates, or inherits, all the features of its superclass and adds its own unique features. Subclasses need not repeat the features of the superclass.
For example, ScrollingWindow and FixedWindow are subclasses of Window. Both subclasses inherit the features of Window, such as a visible region on the screen. ScrollingWindow adds a scroll bar and an offset.
The ability to factor out common features of several classes into a superclass can greatly reduce repetition within designs and programs and is one of the main advantages of OO technology.

Encapsulation
Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system.

Polymorphism means that the same operation may behave differently for different classes. The move operation, for example, behaves differently for a pawn than for the queen in a chess game. An operation is a procedure or transformation that an object performs or is subject to. An implementation of an operation by a specific class is called a method. Because an OO operator is polymorphic, it may have more than one method implementing it, each for a different class of object.

b) Why is the model required in analysis and design? What is the role of UML in preparing the model?

Models are used during the analysis process to help to elicit the requirements, during the design process to describe the system to engineers, and after implementation to document the system structure and operation.

Role of UML in OO Design

UML is a modeling language used to model software and non-software systems. Although UML is used for non-software systems, the emphasis is on modeling OO software applications. Most of the UML diagrams discussed so far are used to model different aspects such as static, dynamic, etc. Now whatever be the aspect, the artifacts are nothing but objects.

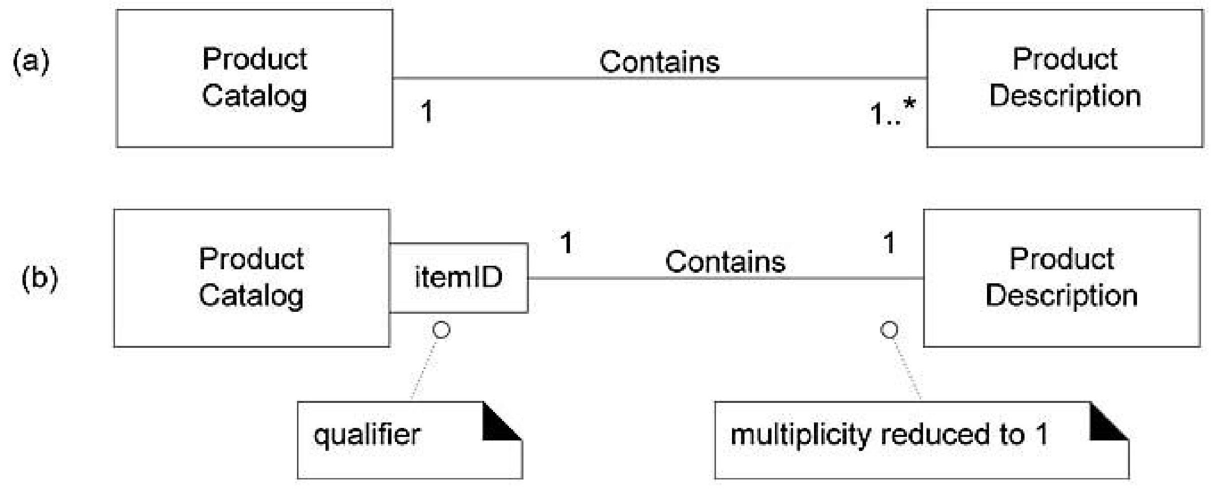
If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

Hence, the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement. Before understanding the UML in detail, the OO concept should be learned properly. Once the OO analysis and design is done, the next step is very easy. The input from OO analysis and design is the input to UML diagrams.

c) Explain following terms with suitable UML notations with respect to class model.
Qualified association, Association class, Aggregation, Multiplicity

A **qualified association** has a **qualifier** that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key. Informally, from a software perspective, it suggests looking things up by a key, such as objects in a *HashMap*. For example, if a *ProductCatalog* contains many *ProductDescriptions*, and each one can be selected by an *itemID*, then the UML notation in Figure can be used to depict this.

There's one subtle point about qualified associations: the change in multiplicity. For example, as contrasted in Figure (a) vs. (b), qualification reduces the multiplicity at the target end of the association, usually down from many to one, because it implies the selection of usually one instance from a larger set.

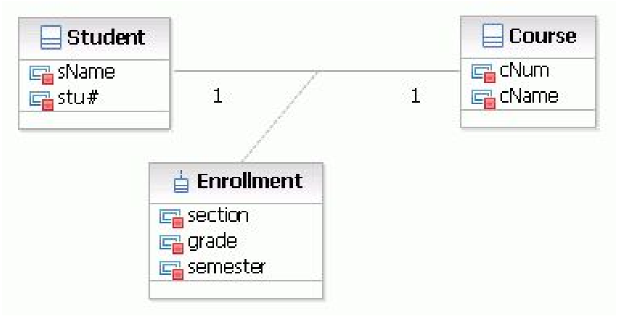


In UML diagrams, an **association class** is a class that is part of an association relationship between two other classes.

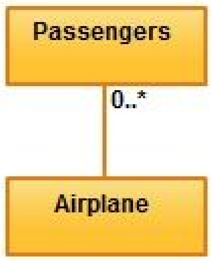
You can attach an association class to an association relationship to provide additional information about the relationship. An association class is identical to other classes and can contain operations, attributes, as well as other associations.

For example, a class called Student represents a student and has an association with a class called Course, which represents an educational course. The Student class can enroll in a course. An association class called Enrollment further defines the relationship between the Student and Course classes by providing section, grade, and semester information related to the association relationship.

As the following figure illustrates, an association class is connected to an association by a dotted line.

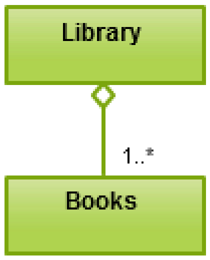


Multiplicity is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means “zero to many”.



Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

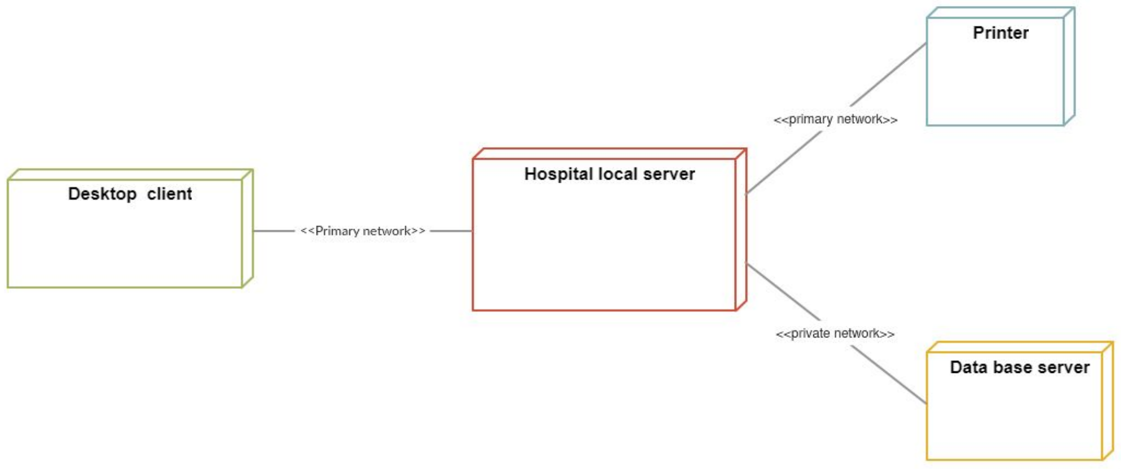


d) Write short notes on 1) Deployment diagram. 2) Component diagram.

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed. Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams are important for visualizing, specifying, and documenting embedded, client/server, and distributed systems and also for managing executable systems through forward and reverse engineering.

- 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.

A deployment diagram is just a special kind of class diagram, which focuses on a system's nodes. Graphically, a deployment diagram is a collection of vertices and arcs.

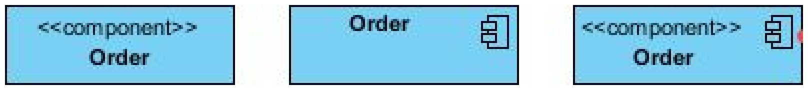


Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

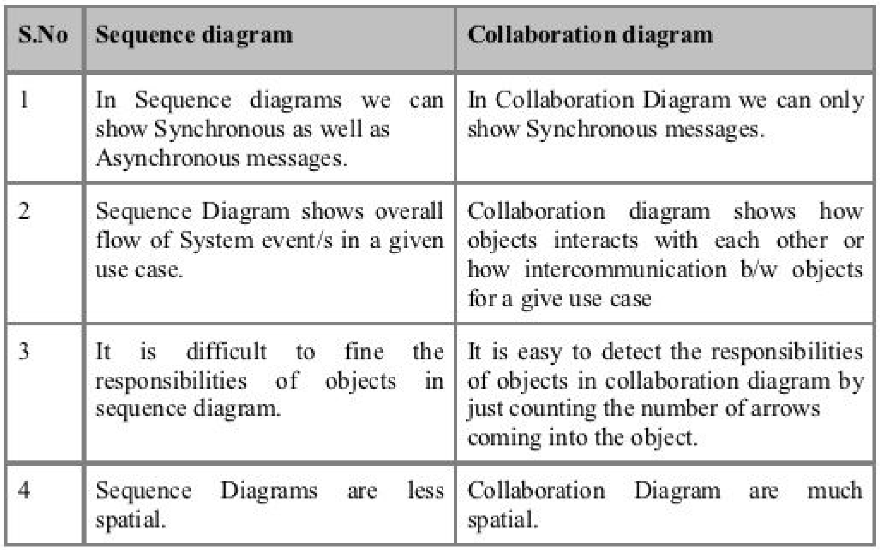
Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. In UML 2, a component is drawn as a rectangle with optional compartments stacked vertically. A high-level, abstracted view of a component in UML 2 can be modeled as:

1. A rectangle with the component's name
2. A rectangle with the component icon
3. A rectangle with the stereotype text and/or icon



e) What are the dissimilarities between a sequence diagram and collaboration diagram?



S.No	Sequence diagram	Collaboration diagram
1	In Sequence diagrams we can show Synchronous as well as Asynchronous messages.	In Collaboration Diagram we can only show Synchronous messages.
2	Sequence Diagram shows overall flow of System event/s in a given use case.	Collaboration diagram shows how objects interacts with each other or how intercommunication b/w objects for a give use case
3	It is difficult to fine the responsibilities of objects in sequence diagram.	It is easy to detect the responsibilities of objects in collaboration diagram by just counting the number of arrows coming into the object.
4	Sequence Diagrams are less spatial.	Collaboration Diagram are much spatial.

f) What does the term object oriented mean? Explain the four aspects included in the object oriented approach?

Object-oriented programming combines a group of variables (properties) and functions (methods) into a unit called an object. These objects are organized into classes where individual objects can be grouped together. OOP can help you consider objects in a program's code and the different actions that could happen in relation to the objects.

This programming style widely exists in commonly used programming languages like Java, C++ and PHP. These languages help simplify the structure and organization of software programs. Programmers often use OOP when they need to create complex programs.

Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance and polymorphism. Even if these concepts seem incredibly complex, understanding the general framework of how they work will help you understand the basics of a computer program. Here are the four basic theories and what they entail:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation

The different objects inside of one program will try to communicate with each other automatically. If a programmer wants to stop objects from interacting with each other, they need to be encapsulated in individual classes. Through the process of encapsulation, classes cannot change or interact with the specific variables and functions of an object.

Just like a pill "encapsulates" or contains the medication inside of its coating, the principle of encapsulation works in a digital way to form a protective barrier around the information that separates it from the rest of the code. Programmers can replicate this object throughout different parts of the program or other programs.

Abstraction

Abstraction is like an extension of encapsulation because it hides certain properties and methods from the outside code to make the interface of the objects simpler. Programmers use abstraction for several beneficial reasons. Overall, abstraction helps isolate the impact of changes made to the code so that if something goes wrong, the change will only affect the variables shown and not the outside code.

Inheritance

Using this concept, programmers can extend the functionality of the code's existing classes to eliminate repetitive code. For instance, elements of HTML code that include a text box, select field and checkbox have certain properties in common with specific methods.

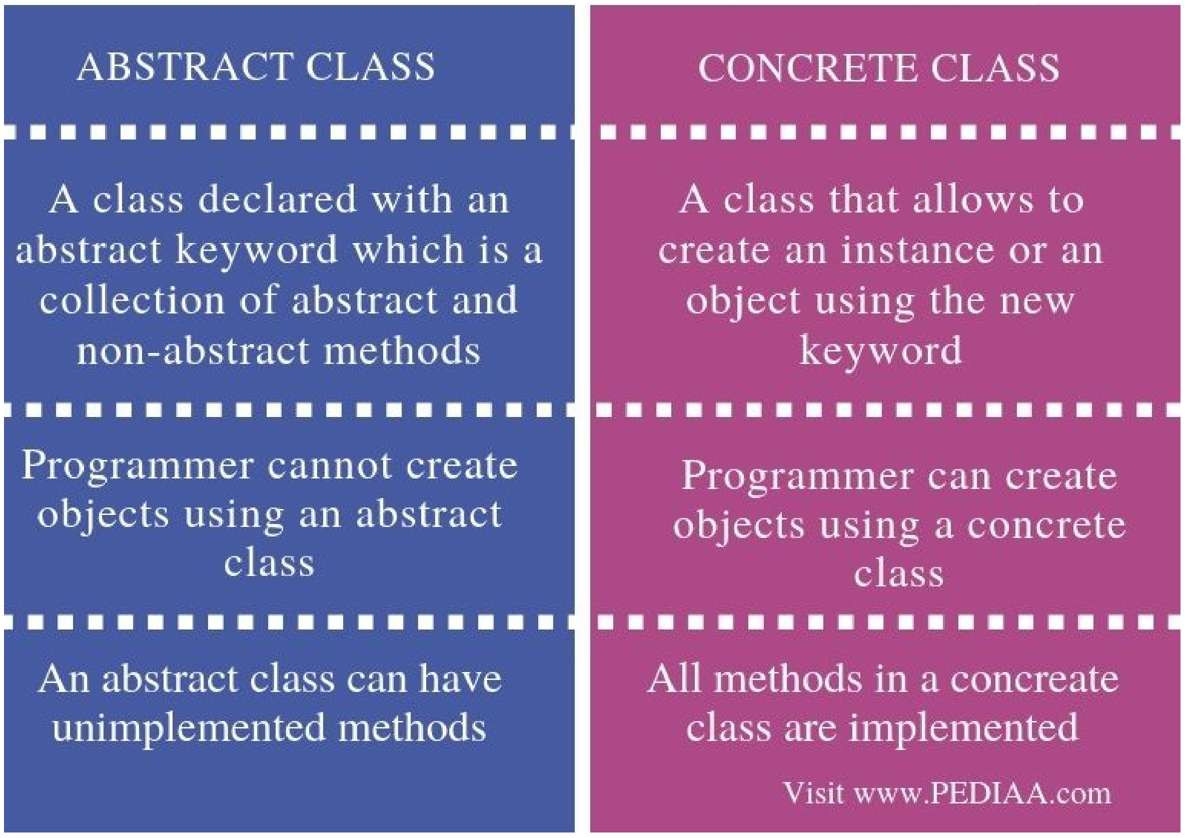
Instead of redefining the properties and methods for every type of HTML element, you can define them once in a generic object. Naming that object something like "HTMLElement" will cause other objects to inherit its properties and methods so you can reduce unnecessary code.

The main object is the superclass and all objects that follow it are subclasses. Subclasses can have separate elements while adding what they need from the superclass.

Polymorphism

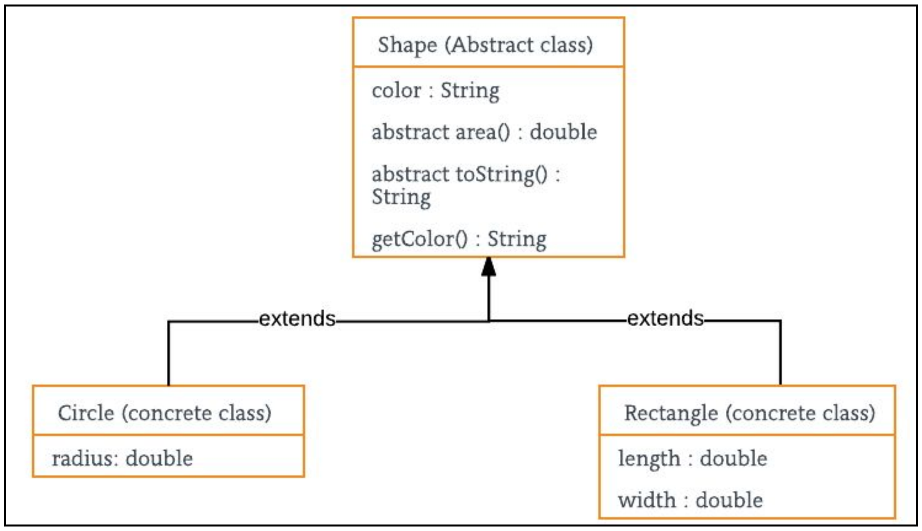
This technique meaning "many forms or shapes" allows programmers to render multiple HTML elements depending on the type of object. This concept allows programmers to redefine the way something works by changing how it is done or by changing the parts in which it is done. Terms of polymorphism are called overriding and overloading.

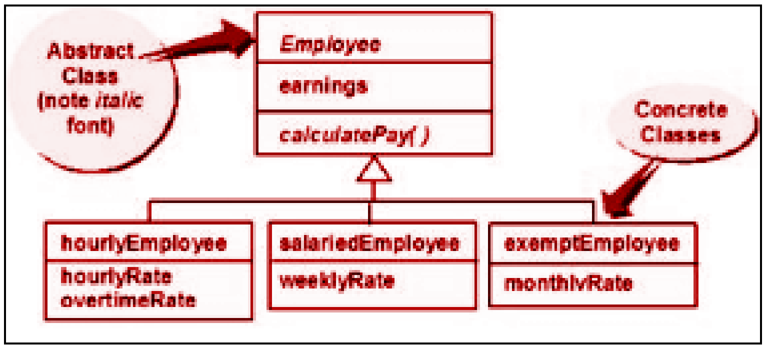
g) What is an abstract & concrete class? Explain with an example.

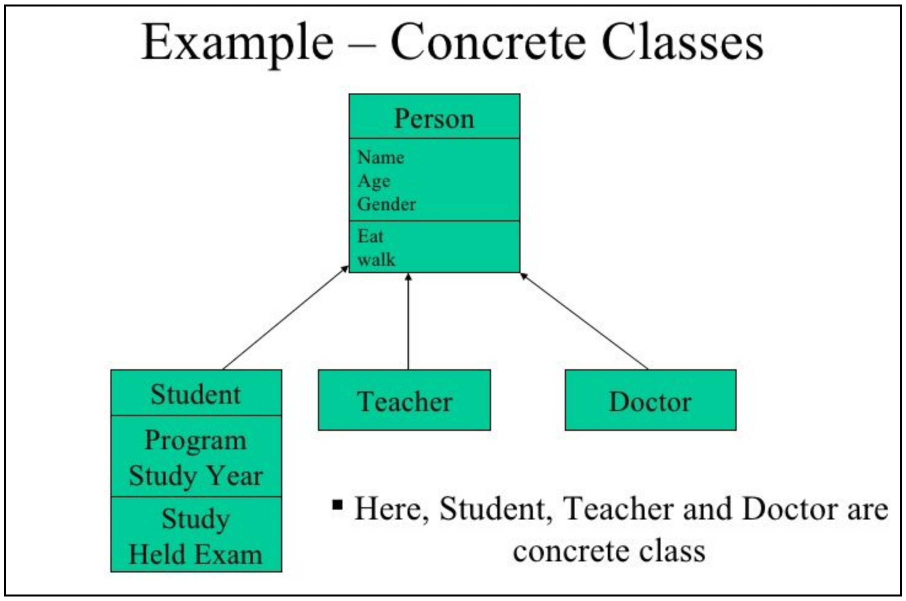


ABSTRACT CLASS	CONCRETE CLASS
A class declared with an abstract keyword which is a collection of abstract and non-abstract methods	A class that allows to create an instance or an object using the new keyword
Programmer cannot create objects using an abstract class	Programmer can create objects using a concrete class
An abstract class can have unimplemented methods	All methods in a concreate class are implemented

Visit www.PEDIAA.com



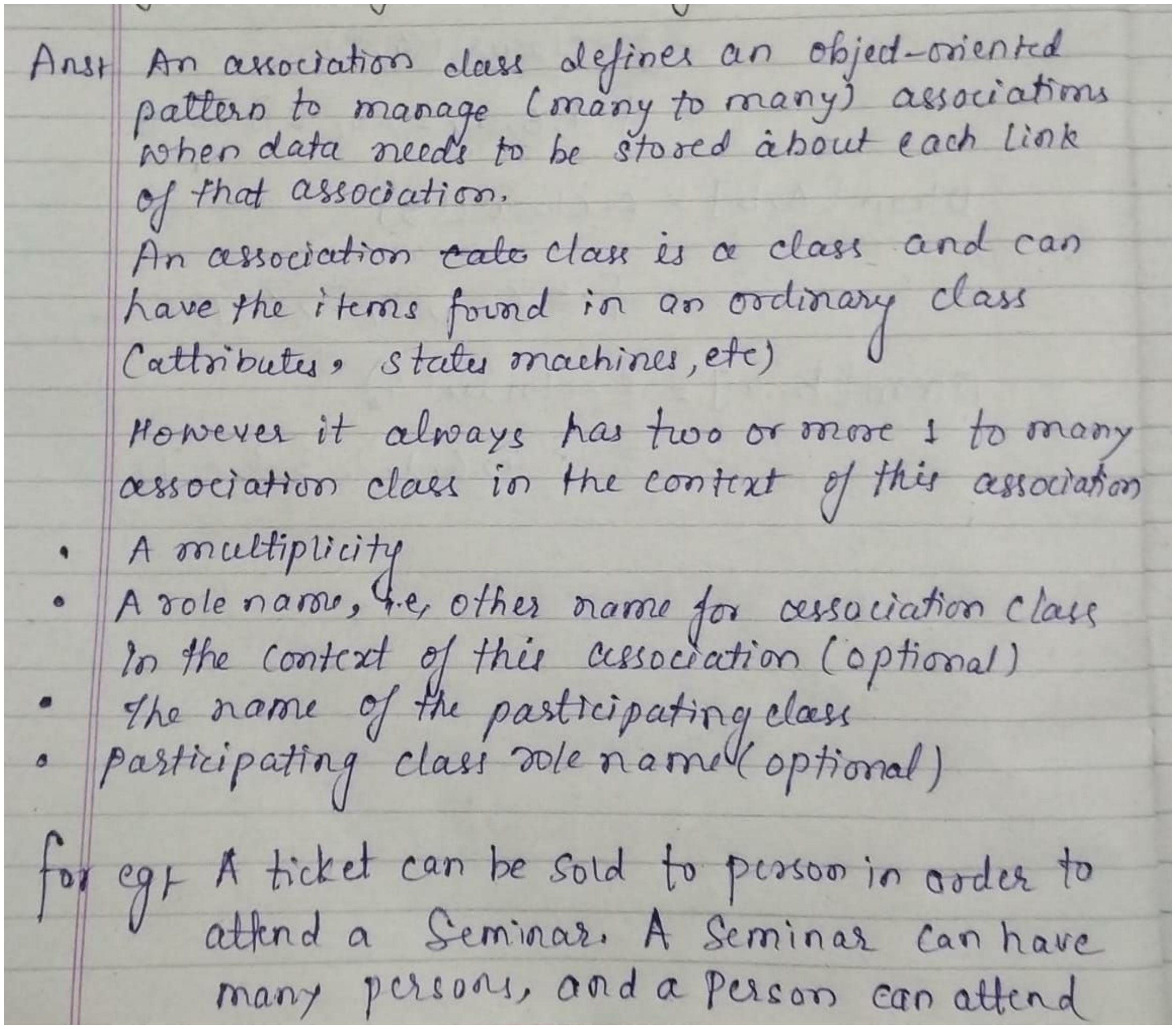




Example – Concrete Classes

- Here, Student, Teacher and Doctor are concrete class

h) Which type of associations provides compelling rationale for association classes? Give examples of attributes for one to many associations.



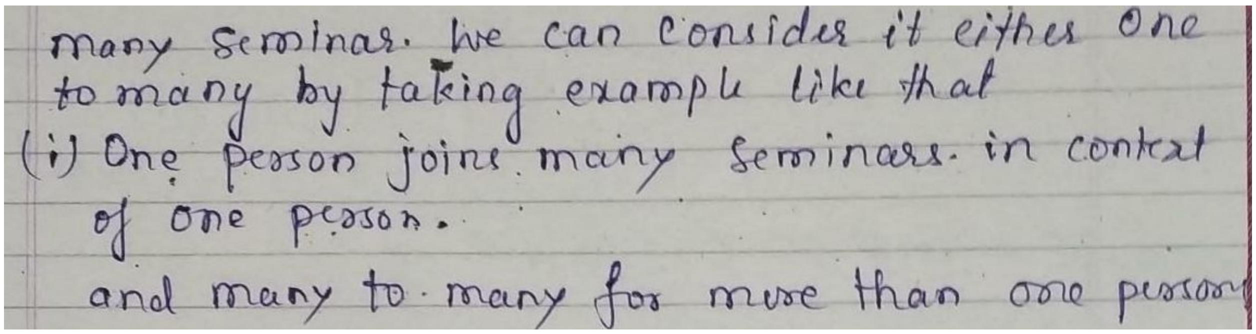
Ans: An association class defines an object-oriented pattern to manage (many to many) associations when data needs to be stored about each link of that association.

An association ~~calo~~ class is a class and can have the items found in an ordinary class (attributes, states machines, etc)

However it always has two or more 1 to many association class in the context of this association

- A multiplicity
- A role name, i.e, other name for association class in the context of this association (optional)
- The name of the participating class
- participating class role name (optional)

for eg: A ticket can be sold to person in order to attend a Seminar. A Seminar can have many persons, and a Person can attend

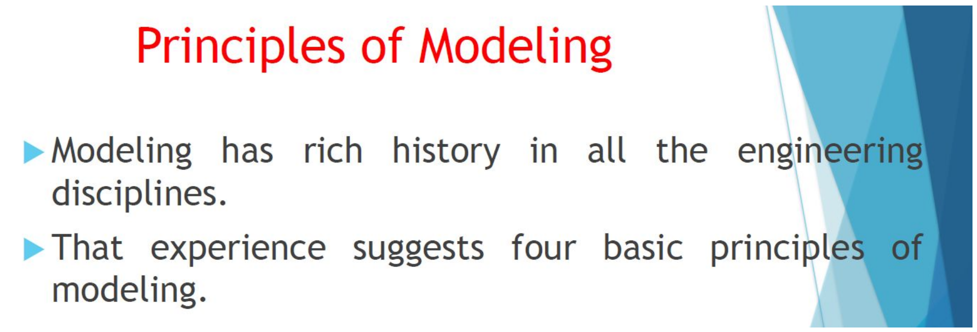


many Seminar. We can consider it either one to many by taking example like that

(i) One person joins many Seminars. in context of one person.

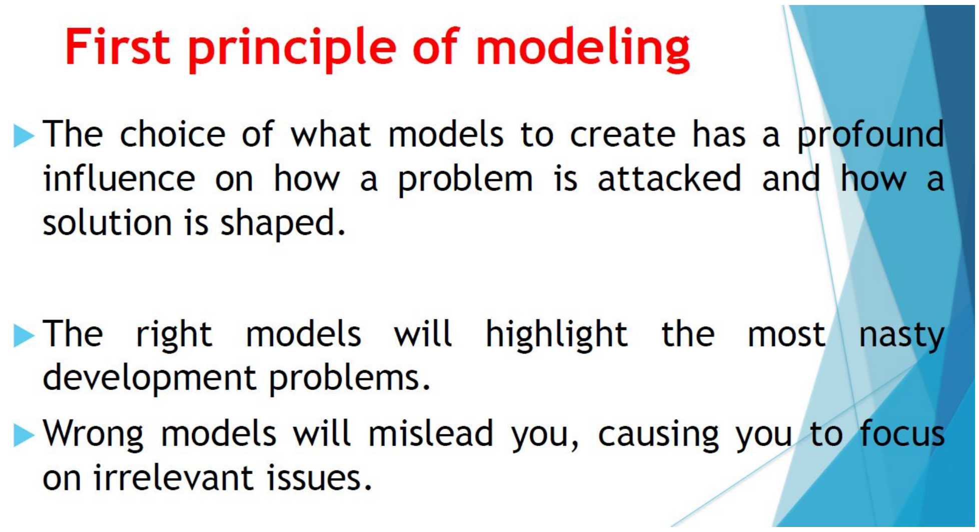
and many to many for more than one person

i) Write four principles of modeling & justify their relevance with suitable examples.



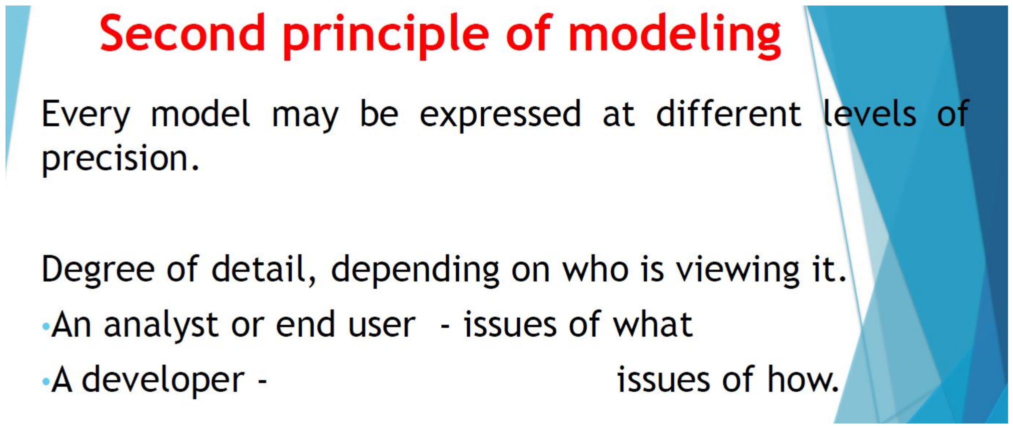
Principles of Modeling

- Modeling has rich history in all the engineering disciplines.
- That experience suggests four basic principles of modeling.



First principle of modeling

- The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.
- The right models will highlight the most nasty development problems.
- Wrong models will mislead you, causing you to focus on irrelevant issues.

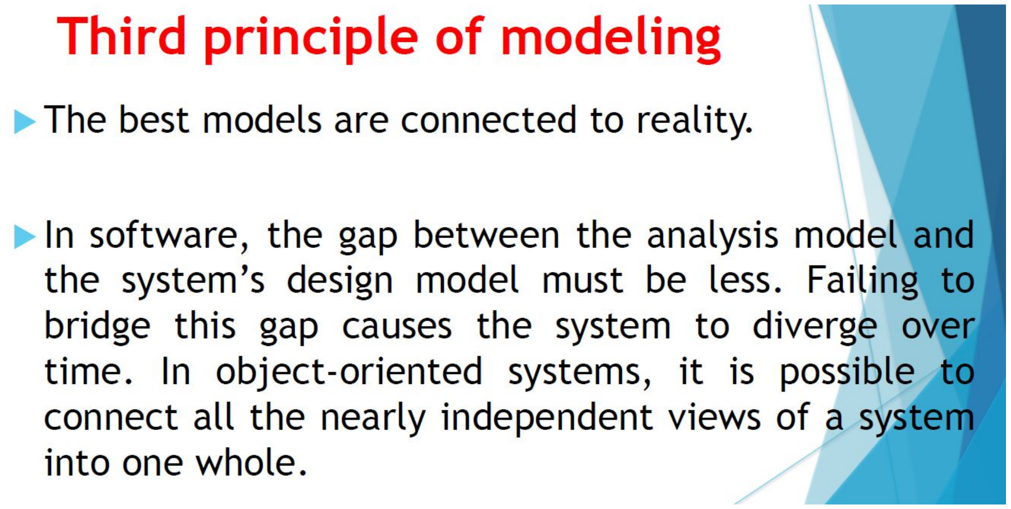


Second principle of modeling

Every model may be expressed at different levels of precision.

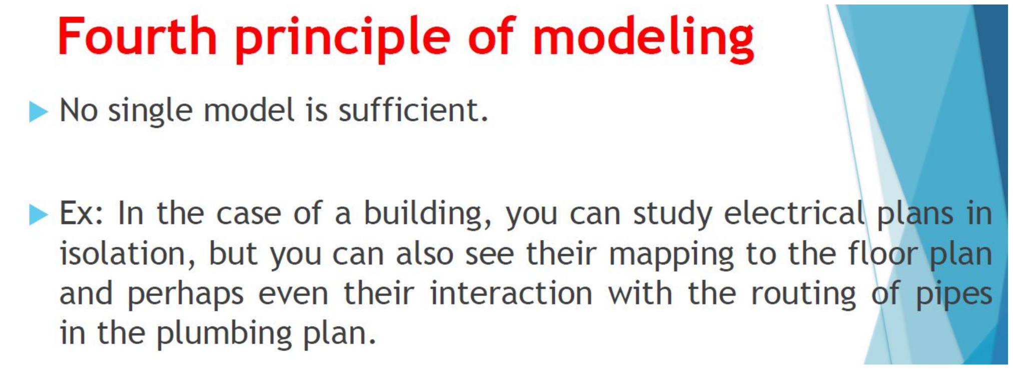
Degree of detail, depending on who is viewing it.

- An analyst or end user - issues of what
- A developer - issues of how.



Third principle of modeling

- The best models are connected to reality.
- In software, the gap between the analysis model and the system's design model must be less. Failing to bridge this gap causes the system to diverge over time. In object-oriented systems, it is possible to connect all the nearly independent views of a system into one whole.



Fourth principle of modeling

- No single model is sufficient.
- Ex: In the case of a building, you can study electrical plans in isolation, but you can also see their mapping to the floor plan and perhaps even their interaction with the routing of pipes in the plumbing plan.

Example nahi mil raha

j) What is object-orientation? Why is a model required in analysis and design ?

Superficially the term object-oriented (OO) means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This contrasts with previous programming approaches in which data structure and behavior are only loosely connected.

They generally include four aspects: identity, classification, inheritance, and polymorphism.

- **Identity** means that data is quantized into discrete, distinguishable entities called objects. The white queen in a chess game are examples of objects. Objects can be concrete, such as a file in a file system, or conceptual, such as a scheduling policy in a multiprocessing operating system.
- **Classification** means that objects with the same data structure (attributes) and behavior (operations) are grouped into a class. Paragraph, Monitor, and ChessPiece are examples of classes.
- **Inheritance** is the sharing of attributes and operations (features) among classes based on a hierarchical relationship. Each subclass incorporates, or inherits, all the features of its superclass and adds its own unique features. Subclasses need not repeat the features of the superclass.

❑ **Polymorphism** means that the same operation may behave differently for different classes. The move operation, for example, behaves differently for a pawn than for the queen in a chess game.

A **model** is a simplification of reality, providing blueprints of a system. UML(Unified Modeling Language), in specific:

- Permits you to specify the structure or behavior of a system.
- Helps you visualize a system.
- Provides a template that guides you in constructing a system.
- Helps to understand complex systems part by part.
- Document the decisions that you have made.

We build models so that we can better understand the system we are developing. A model may encompass an overview of the system under consideration, as well as a detailed planning for system design, implementation and testing.

k) What is inheritance? List the different types of inheritance and explain how it encourages reusability and sharing.

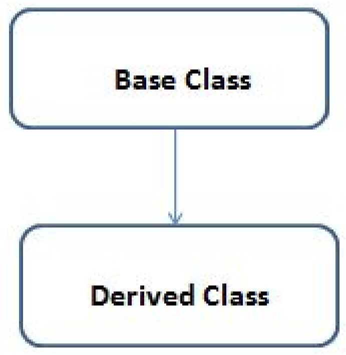
Different Types of Inheritance

Inheritance is the process of creating a new Class, called the **Derived Class** , from the existing class, called the **Base Class** . The Inheritance has many advantages, the most important of them being the reusability of code. Rather than developing **new Objects** from scratch, new code can be based on the work of other developers, adding only the new features that are needed. The reuse of **existing classes** saves time and effort.

However, inheritance may be implemented in different combinations in **Object-Oriented Programming languages** as illustrated in figure and they include:

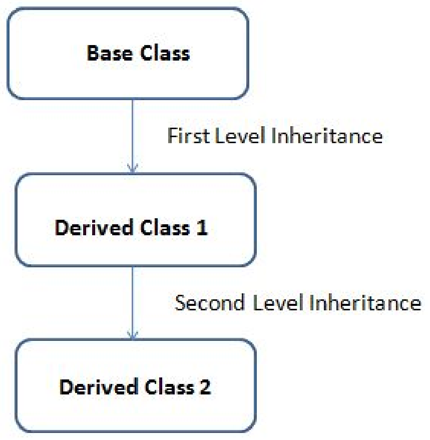
- ■ Single Inheritance
- ■ Multi Level Inheritance
- ■ Hierarchical Inheritance
- ■ Hybrid Inheritance
- ■ Multiple Inheritance

Single Inheritance



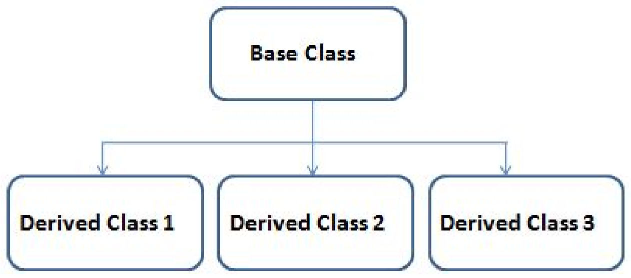
When a **Derived Class** inherits properties and behavior from a single **Base Class** , it is called a single inheritance.

Multi Level Inheritance



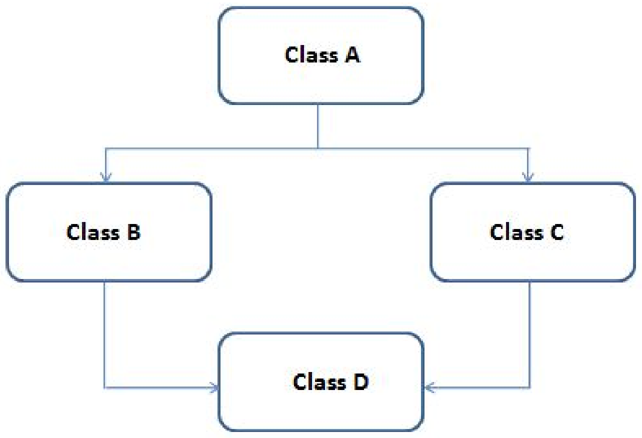
A **derived class** is created from another derived class is called **Multi Level Inheritance**.

Hierarchical Inheritance



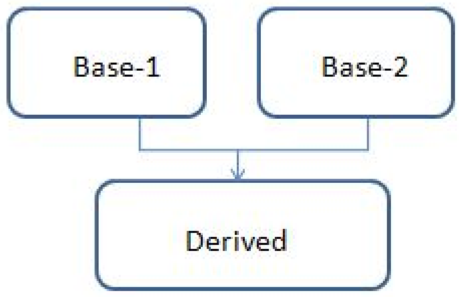
More than one **derived class** is created from a single base class, is called **Hierarchical Inheritance** .

Hybrid Inheritance



Any combination of the above **three inheritance** (single, hierarchical and multi level) is called hybrid **inheritance** .

Multiple Inheritance



Multiple inheritances allows programmers to create classes that combine aspects of multiple classes and their corresponding hierarchies. In .Net Framework, the classes are only allowed to inherit from a single parent class, which is called single inheritance.

I) What is the purpose of class modeling? Explain the following concept with an example.

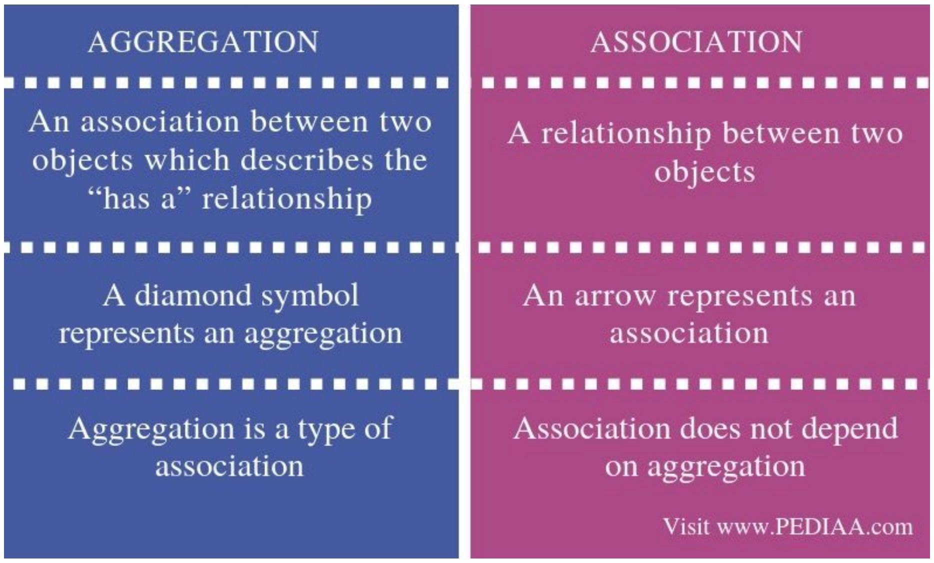
Class Model:

The class model shows all the classes present in the system. The class model shows the attributes and the behavior associated with the objects.

The class diagram is used to show the class model.The class diagram shows the class name followed by the attributes followed by the functions or the methods that are associated with the object of the class.Goal in constructing class model is to capture those concepts from the real world that are important to an application.

In the **example**, a **class** called “loan account” is depicted.

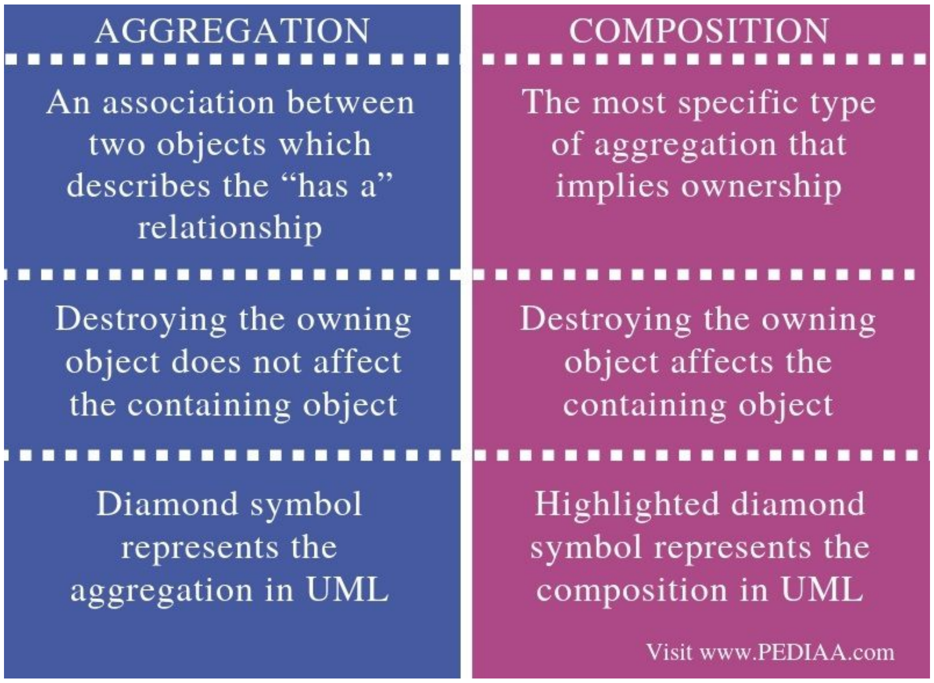
i) Aggregation versus association



AGGREGATION	ASSOCIATION
An association between two objects which describes the “has a” relationship	A relationship between two objects
A diamond symbol represents an aggregation	An arrow represents an association
Aggregation is a type of association	Association does not depend on aggregation

Visit www.PEDIAA.com

ii) Aggregation versus composition



AGGREGATION	COMPOSITION
An association between two objects which describes the “has a” relationship	The most specific type of aggregation that implies ownership
Destroying the owning object does not affect the containing object	Destroying the owning object affects the containing object
Diamond symbol represents the aggregation in UML	Highlighted diamond symbol represents the composition in UML

Visit www.PEDIAA.com

Section-C (Long answer based questions)

a) Why is a model required in analysis and design? What is the role of UML in preparing the model? Explain the types of model with their purpose in brief. Which of these models belong to the structural group and which of them fall under the behavioral group?

Models are used during the analysis process to help to elicit the requirements, during the design process to describe the system to engineers, and after implementation to document the system structure and operation.

Role of UML in OO Design

UML is a modeling language used to model software and non-software systems. Although UML is used for non-software systems, the emphasis is on modeling OO software applications. Most of the UML diagrams discussed so far are used to model different aspects such as static, dynamic, etc. Now whatever be the aspect, the artifacts are nothing but objects.

If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects.

Hence, the relation between OO design and UML is very important to understand. The OO design is transformed into UML diagrams according to the requirement. Before understanding the UML in detail, the OO concept should be learned properly. Once the OO analysis and design is done, the next step is very easy. The input from OO analysis and design is the input to UML diagrams.

Types of Models:

There are 3 types of models in object oriented modeling and design are: Class Model, State Model, and Interaction Model. These are explained as follows below.

1. **Class Model:**

The class model shows all the classes present in the system. The class model shows the attributes and the behavior associated with the objects.

The class diagram is used to show the class model.The class diagram shows the class name followed by the attributes followed by the functions or the methods that are associated with the object of the class.Goal in constructing class model is to capture those concepts from the real world that are important to an application.

2. **State Model:**
   State model describes those aspects of objects concerned with time and the sequencing of operations – events that mark changes, states that define the context for events, and the organization of events and states.Actions and events in a state diagram become operations on objects in the class model. State diagram describes the state model.

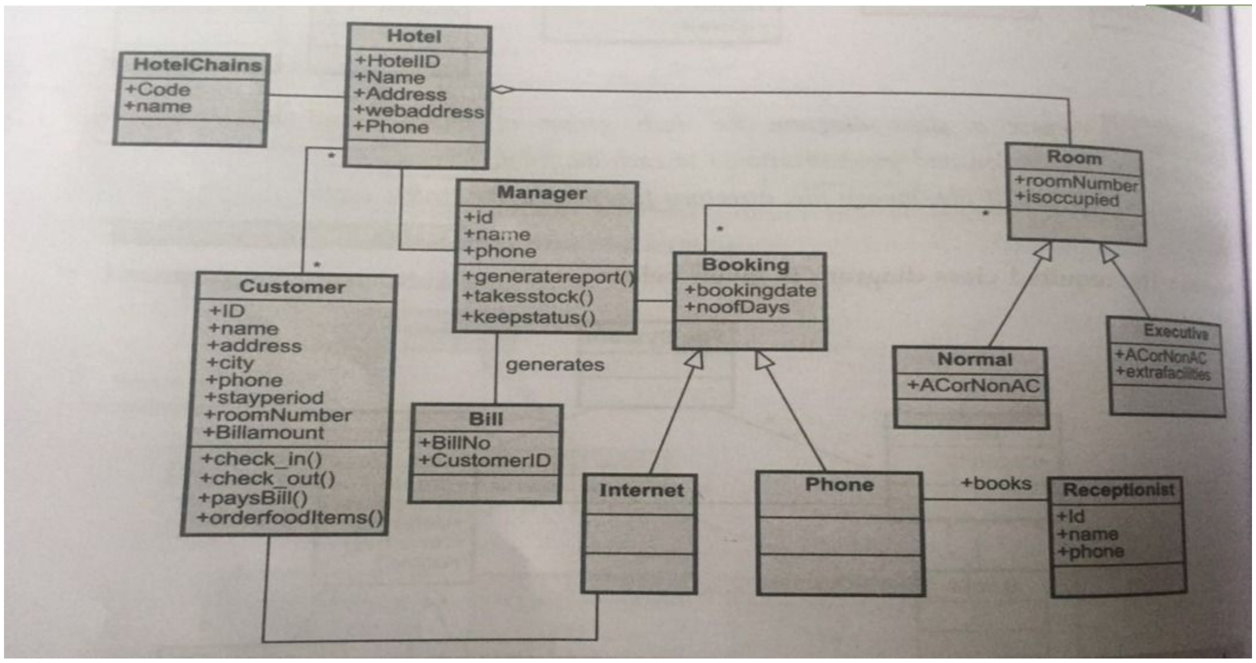
3. **Interaction Model:**
   Interaction model is used to show the various interactions between objects, how the objects collaborate to achieve the behavior of the system as a whole.
   The following diagrams are used to show the interaction model:
   - Use Case Diagram
   - Sequence Diagram
   - Activity Diagram

Class model falls under structural group and state model falls under behavioural group.

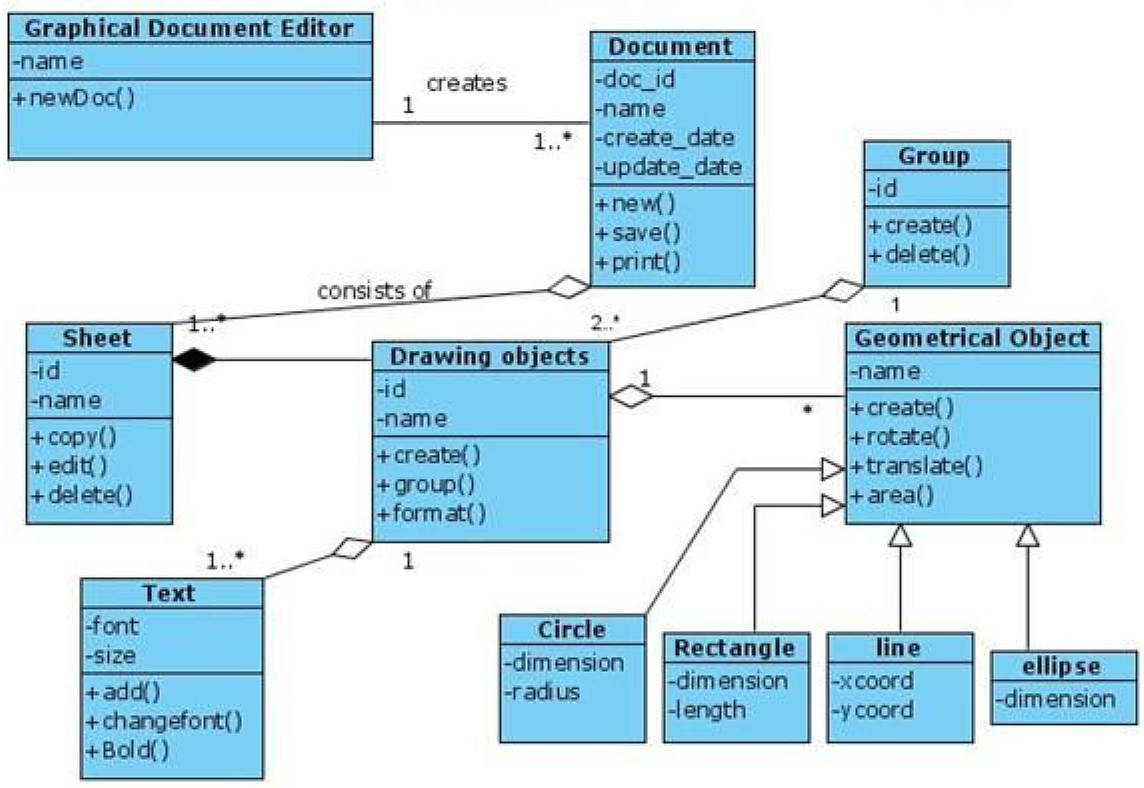
b) Define the purpose of following terms with suitable examples and UML notations with respect to class models.

(i) Qualified association (ii) Association class (iii) Aggregation (iv) Multiplicity

c) Prepare a class model for the hotel management system.
The system should support a chain of hotels. A hotel contains two categories of rooms: executive and normal, both AC and non-AC. The customers of executive rooms can avail extra facilities like games, swimming, food service in rooms, etc. The booking is possible by internet or by phone. If the booking is through the phone, the process is done by the receptionist, and if booking is done through the internet the process is carried out by the customer through the hotel website. Depending on the number of days the customer stays, an appropriate bill is generated. The bill also contains an amount for transport, food and other facilities enjoyed by the customer along with necessary taxes. The manager should be able to generate reports like list of customers staying in the hotel, list of rooms empty, monthly/yearly income, etc



d) Prepare a class diagram for a graphical document editor that supports grouping. Assume that a document consists of several sheets. Each sheet contains drawing objects, including text, geometrical objects and groups. A group is simply a set of drawing objects, possibly including other groups. A group must contain at least two drawing objects. A drawing object can be a direct member of at most one group. Geometrical objects include circles, ellipses, rectangles, lines and squares.



e) What is the purpose of the interaction model? Identify the UML diagram used to represent this model and explain the objectives of each.

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.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is −

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

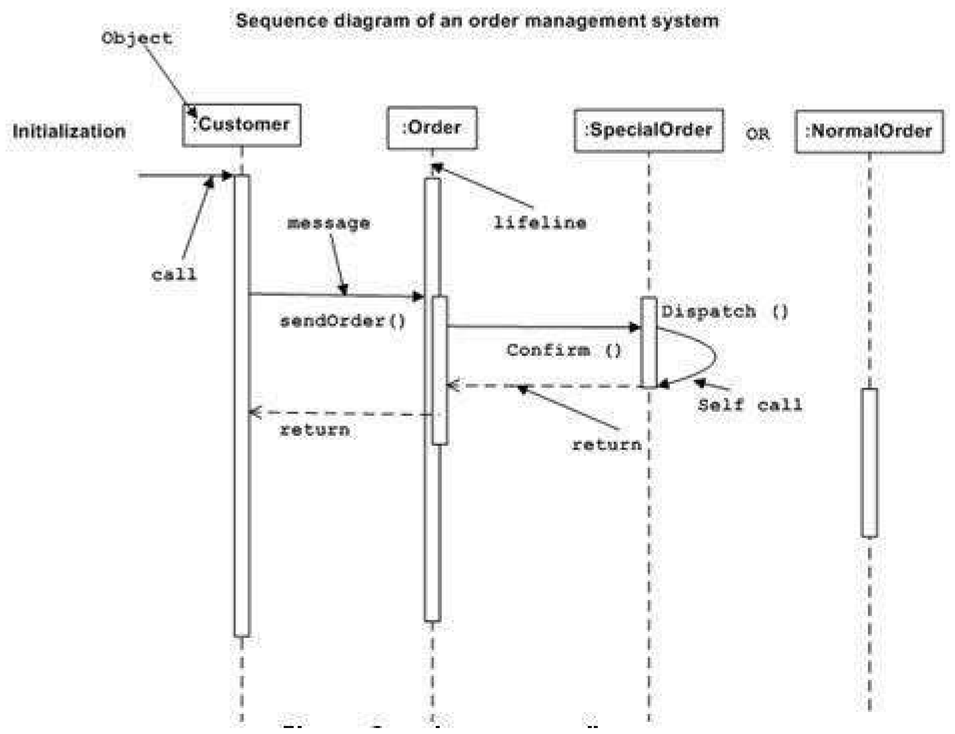
To describe the interaction among objects.

The Sequence Diagram

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

The following diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* 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 *sendOrder ()* which is a method of *Order object*. The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* 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.

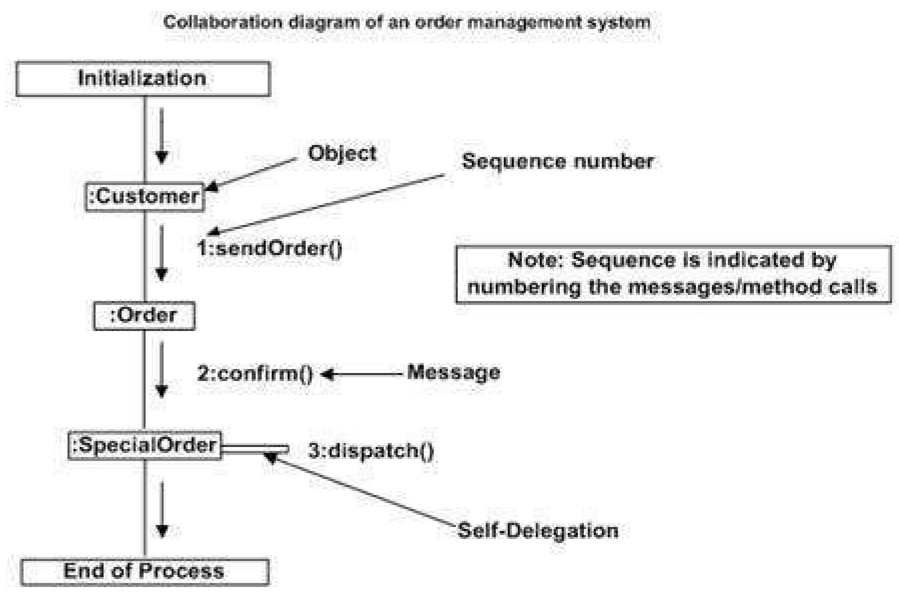


The Collaboration Diagram

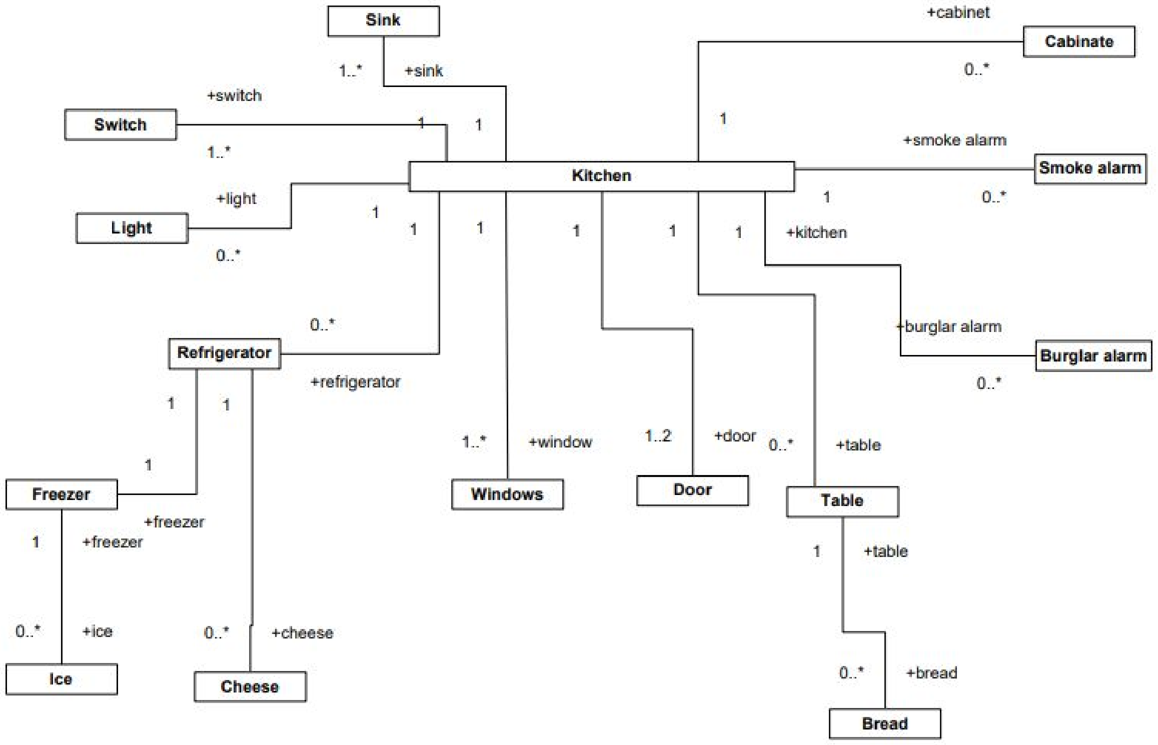
The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

Method calls are similar to that of a sequence diagram. However, the 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 a collaboration diagram is used.



f) Prepare a class diagram for a group of classes.
Sink, freezer, refrigerator, table, light, switch, window, smoke alarm, burglar alarm, cabinet, bread, cheese, ice, door, kitchen.


Sink
1..*
+sink
+switch
Switch
1
1
1..*
+cabinet
Cabinate
0..*
1
+smoke alarm
Smoke alarm
Kitchen
0..*
1
+light
Light
1
1
1
1
1
1
0..*
+kitchen
+burglar alarm
Burglar alarm
0..*
0..*
Refrigerator
+refrigerator
1
1
1
1..*
+window
1..2
+door
0..*
+table
Freezer
+freezer
1
+freezer
Windows
Door
Table
1
+table
0..*
+ice
0..*
+cheese
Ice
Cheese
0..*
+bread
Bread