Object Oriented Modeling and Design

Time: 3 Hrs.] Prelim Question Paper Solution **[Marks : 100**

Q.1(a) Attempt any THREE of the following **[12]**

Q.1(a) (i) What is modeling? Also state its four features. **[4]**

(A) Importance of Modeling

- Modeling is a proven and well accepted engineering technique. We build architectural model of houses and high rises to help their users visualize the final product. We may even build mathematical models to analyze the effects of wind or earthquakes on our building.
- Modeling is not just the part of the building industry.
- Modeling would be unbelievable to deploy a new aircraft or an automobile without first building models from computer models to physical wind tunnel models full scale prototypes.
- A model is a simplification of reality. A model provides the blueprints of the system. Models may encompass detail plans as well as more general plans.
- A good model includes those elements that have broad effect and omits those minor elements that are not relevant to the given level of abstraction.
- Every system may be described from different aspects using different models and each model is therefore semantically closed abstraction of the system.
- A model may be structural, emphasizing the organization of the system or it may be behavioral, emphasizing the dynamics of the system.
- We built models so that we can better understand the system we are developing through modeling, we achieve four aims :
  1. Models help us to visualize the system as it is or as we want it to be.
  2. Models permit us to specify the structure or behavior of a system.
  3. Model gives us a template that guides us in constructing a system.
  4. Models document the decision we have made.
- We build models of complex system because we cannot understand such a system in its whole.

Q.1(a) (ii) Explain concept of interface in component diagram. **[4]**

(A) Interface :

- An interface is a collection of operations that specifies a service of a class or component.
- An interface therefore describes the externally visible behavior of that element.
- In interface might represent the complete behavior of a class or component or only a part of that behavior.
- An interface defines a set of operation specification (that is, their sing), but never set of operation implementation.
- The declaration of an interface looks like a class with the keyword <<interface>> above the name; attributes are not relevant, except sometimes to show constant.
- An interface provided by a class to the outside world is shown as a small circle attached to the class box by a line.
- An interface required by a class from some other class is shown as a small semicircle attached to the class box by a line, as in Figure.

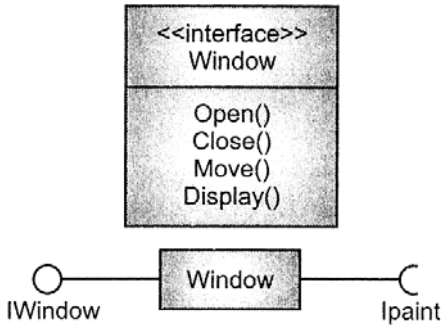


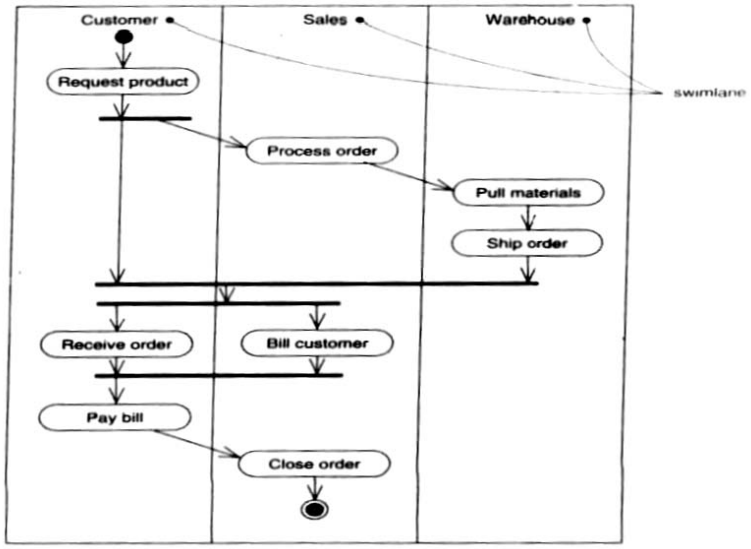
Fig.: Interfaces

Q.1(a) (iii) Explain the swim lane activity diagram with one example. **[4]**

(A) Swim lane:

1. In activity diagram, activity states are partitioned into groups. Each group represents the entity responsible for those activities. Each group is called as swim lane because visually each group is divided by a line from its neighbor.
2. A swim lane specifies a locus of activities.
3. Each swim lane has a unique name within its diagram. It represents some real world entity.
4. Each swim lane represents a high level responsibility for part of the overall activity of an activity diagram and each swim lane may eventually be implemented by one or more classes.
5. In an activity diagram partitioned into swim lane, every activity belongs to exactly on swim lane, but transitions may cross lanes.

Example:



In above diagram we can observe three groups (entities): Customer, sales and Warehouse Customer performs request product, receive order and pay bill activity.

Q.1(a) (iv) Define multiplicity and qualification of class diagram. **[4]**

(A) Multiplicity :

- It is the active logical association when the cardinality of a class in relation to another is being depicted.
- It is also called as cardinality.
- 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", (See Figure 1).

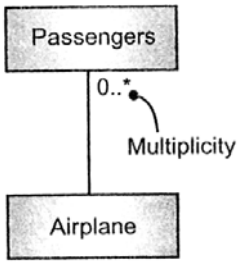
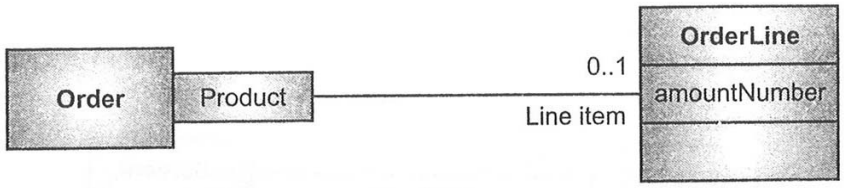


Fig. 1

Qualified Association

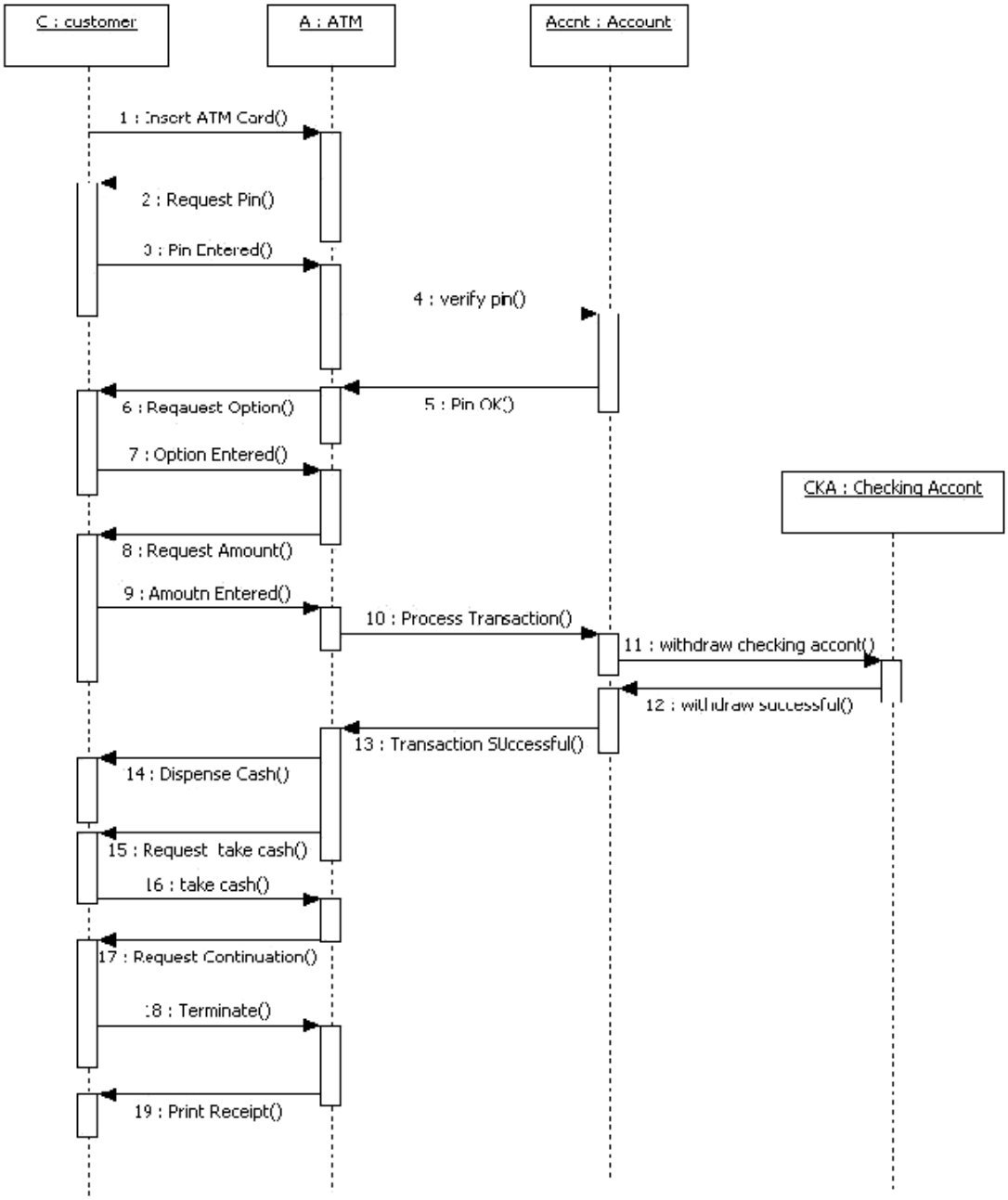
- The qualifier is a special attribute that reduced the effective multiplicity of an association.
- A qualified association is the UML equivalent of a programming concept variously known as associative arrays, maps and dictionaries.
- Figure shows a way of representing the association between the Order and Order Line classes that uses a qualifier.
- The qualifier says that in connection with an Order, there may be one Order Line for each instance of Product.



Q.1(b) Attempt any ONE of the following **[6]**

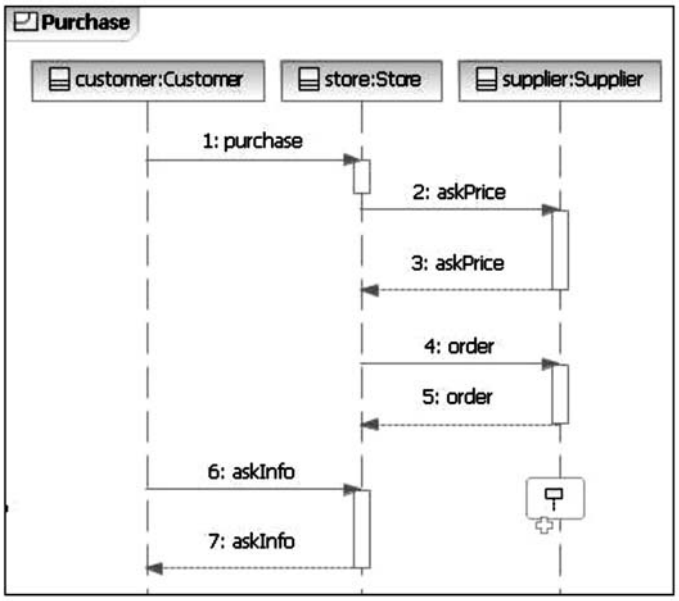
Q.1(b) (i) Draw sequence diagram for ATM session. **[6]**

(A)

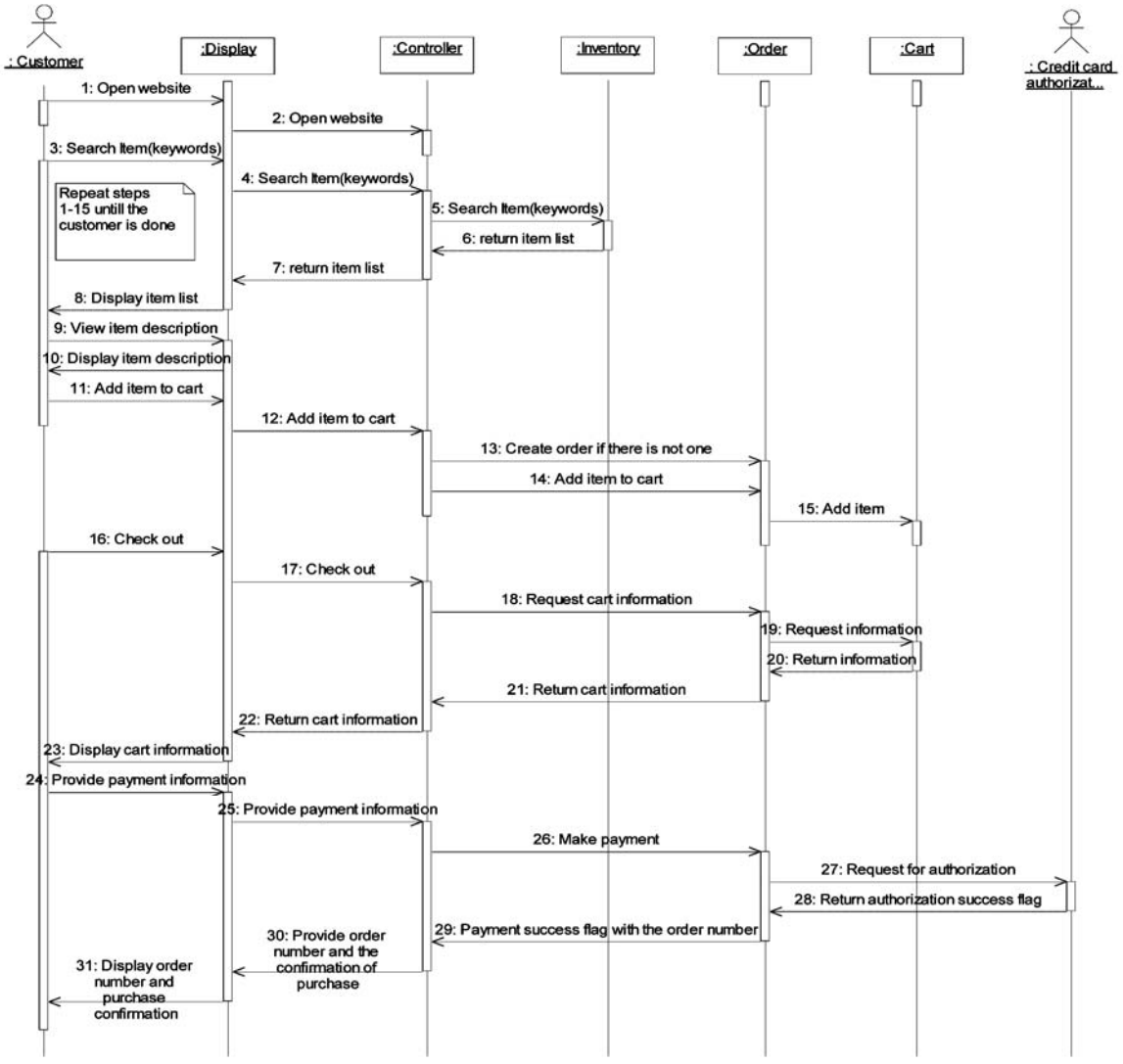


Q.1(b) (ii) Draw sequence diagram for placing purchase order. [6]

(A)



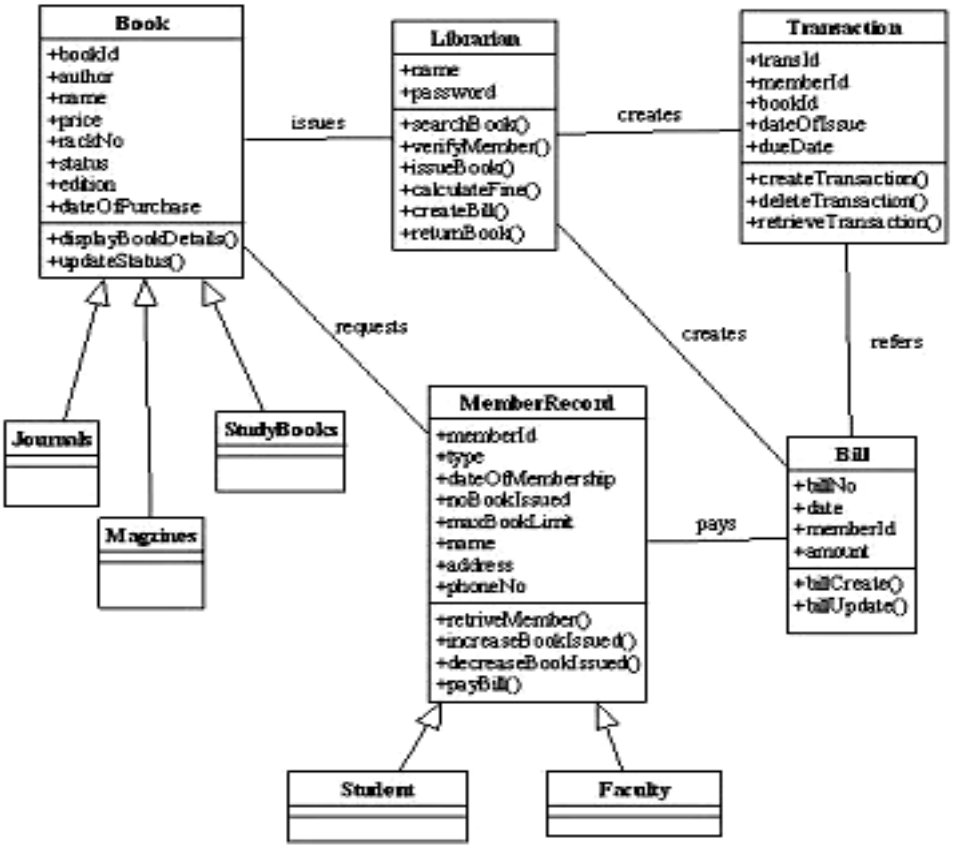
OR



Q.2 Attempt any FOUR of the following : [16]

Q.2(a) Draw class diagram for library management system. [4]

(A)

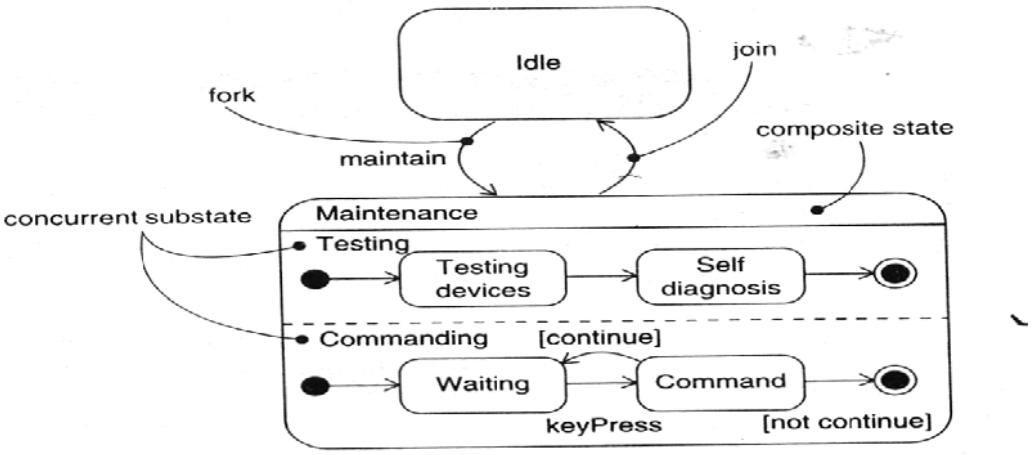


Q.2(b) Describe the concept of concurrent sub-state with respect to state diagram. (4) [4]

(A) Concurrent state diagram shows a set of independent behaviors of an object. Concurrent sub states are independent and can execute in parallel. A state may be divided into regions containing sub-states that exist and execute concurrently.

UML shows concurrency within an object by partitioning the composite / state into regions with dotted lines.

Example:-



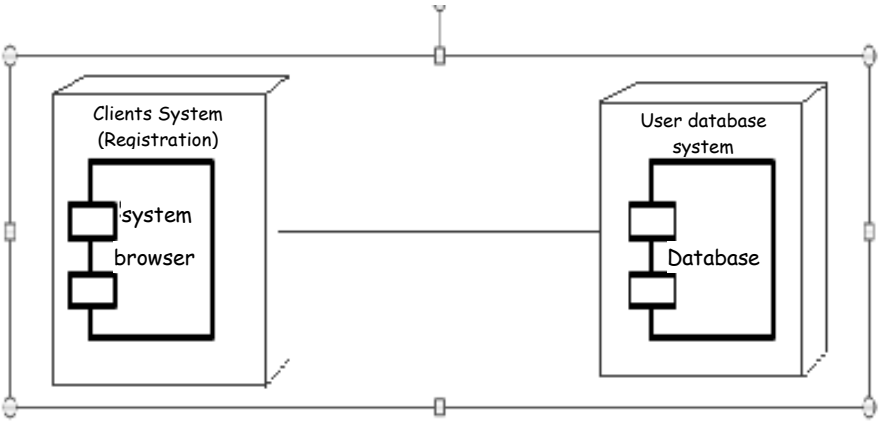
Example:-

In the above example, concurrent sub states are shown. Maintenance is a composite state. It is decomposed into two concurrent sub-states as testing and commanding. Each of these concurrent sub-states is further decomposed into sequential sub-states. When control passes from Idle to Maintenance state, control then forks to two concurrent flows. Execution of these two concurrent sub-states continues parallel in the system. Each nested state machine reaches its final state. If one concurrent sub state reaches its final state

before the other, then control in that sub-state waits at its final state. When both nested state reaches their final state, control from the two concurrent sub-states joins back into one flow.

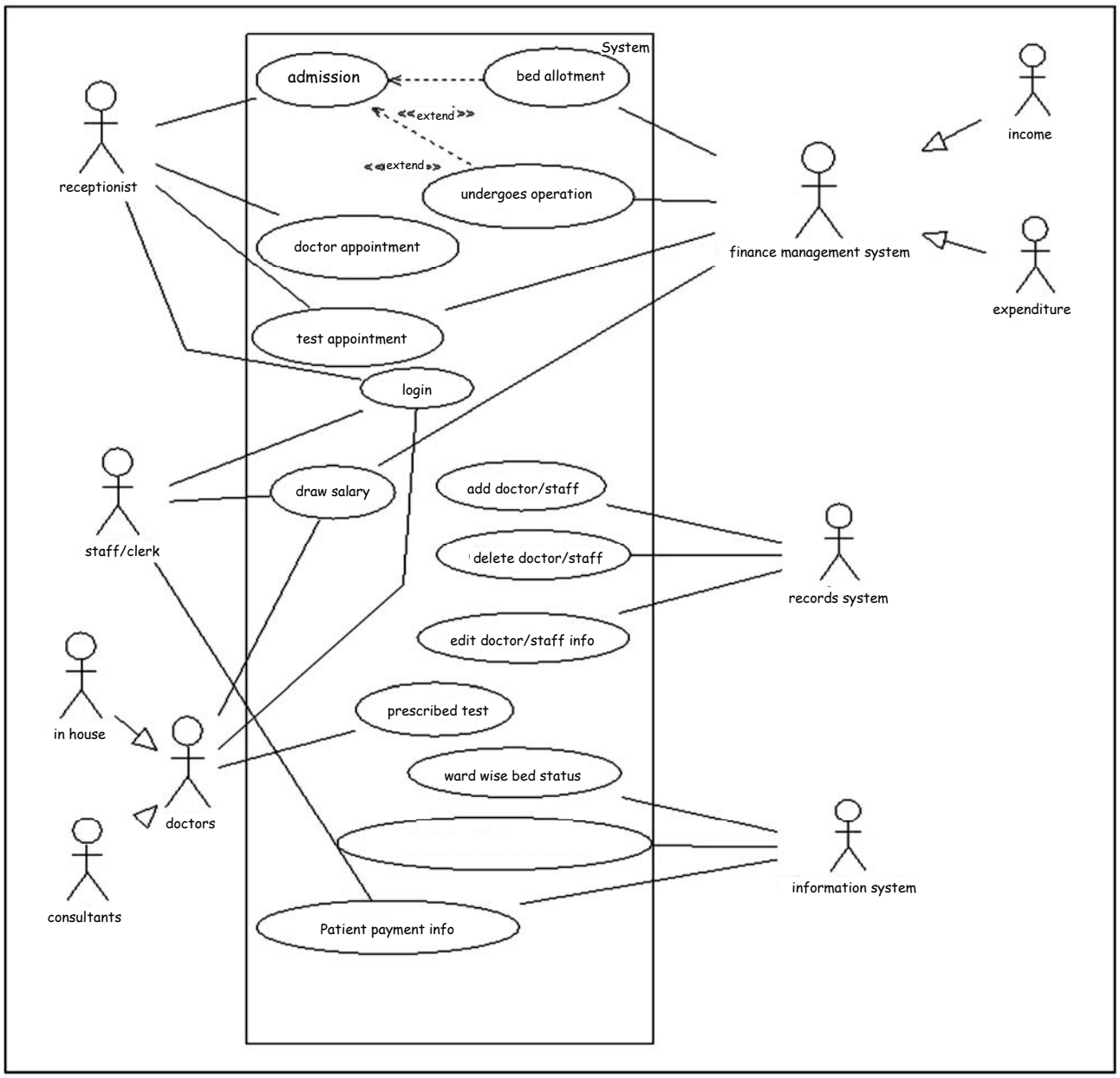
Q.2(c) Draw the deployment diagram for login form. **[4]**

(A)



Q.2(d) Draw use case diagram for Hospital management system. **[4]**

(A)



Q.2(e) Explain following types of relationship with notation: **[4]**

(i) Realization (ii) Generalization (iii) Dependency (iv) Association

(A) (i) Realization

It is a semantic relationship between classifiers, wherein one classifier classifies a contract that another classifier guarantees to carry out. Realization relationships encounters in two places:

- Between interfaces and the classes or components that realize them
- Between use cases and the collaborations that realize them.

Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship, as in figure below:

(ii) Generalization

Generalization: it is a specialization I generalization relationship in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent).

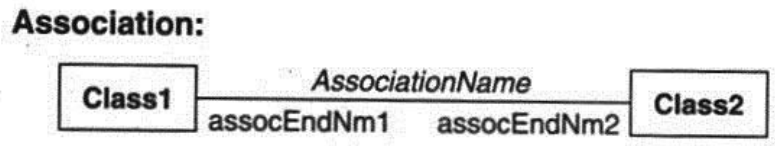
In this way, the child shares the structure and the behavior of the parent. Graphically, a generalization relationship is rendered as a solid line with ahollow arrowhead pointing to the parent, as in figure below:

(iii) Dependency

It is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing. Graphically, a dependency is rendered as a dashed line, possibly directed, and occasionally including a label, as in figure below:

(iv) Association

It is a structural relationship that describes a set of links, a link being a connection among objects. Aggregation is a special kind of association, representing astructural relationship between a whole and its parts. Graphically, an association isrendered as a solid line, possibly directed, occasionally including a label, and often containing other adornments, such as multiplicity and role names, as in figure below:



Q.2(f) Describe notations used for deployment diagram. **[4]**

(A) A deployment is a dependency relationship which describes allocation (deployment) of an artifact to a deployment target. Deployment could be also defined at instance level - as allocation of specific artifact instance to the specific instance of deployment target.

Node is a deployment target which represents computational resource upon which artifacts may be deployed for execution.

Node is shown as a perspective, 3-dimensional view of a cube.

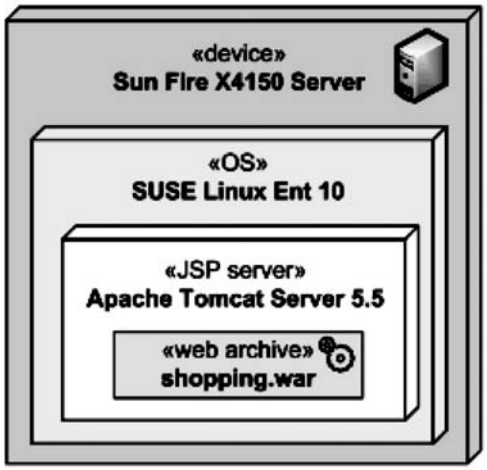
Application Server Node

Nodes can be interconnected with communication paths. Communication paths can be defined between nodes such as application server and database server to define the possible communication paths between the nodes. Specific network topologies can then be defined through links between node instances.

Node is specialized by:

- device
- execution environment

Execution environment is usually part of a general node or «device» which represents the physical hardware environment on which this execution environment resides. Execution environments can be nested (e.g., a database execution environment may be nested in an operating system execution environment).



Several execution environments nested into server device

Device

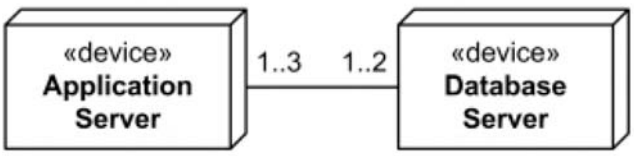
A device is a node which represents a physical computational resource with processing capability upon which artifacts may be deployed for execution.

A device is rendered as a node (perspective, 3-dimensional view of a cube) annotated with keyword «device».

Application Server device

A communication path is association between two deployment targets, through which they are able to exchange signals and messages.

Communication path is notated as association, and it has no additional notation compared to association.

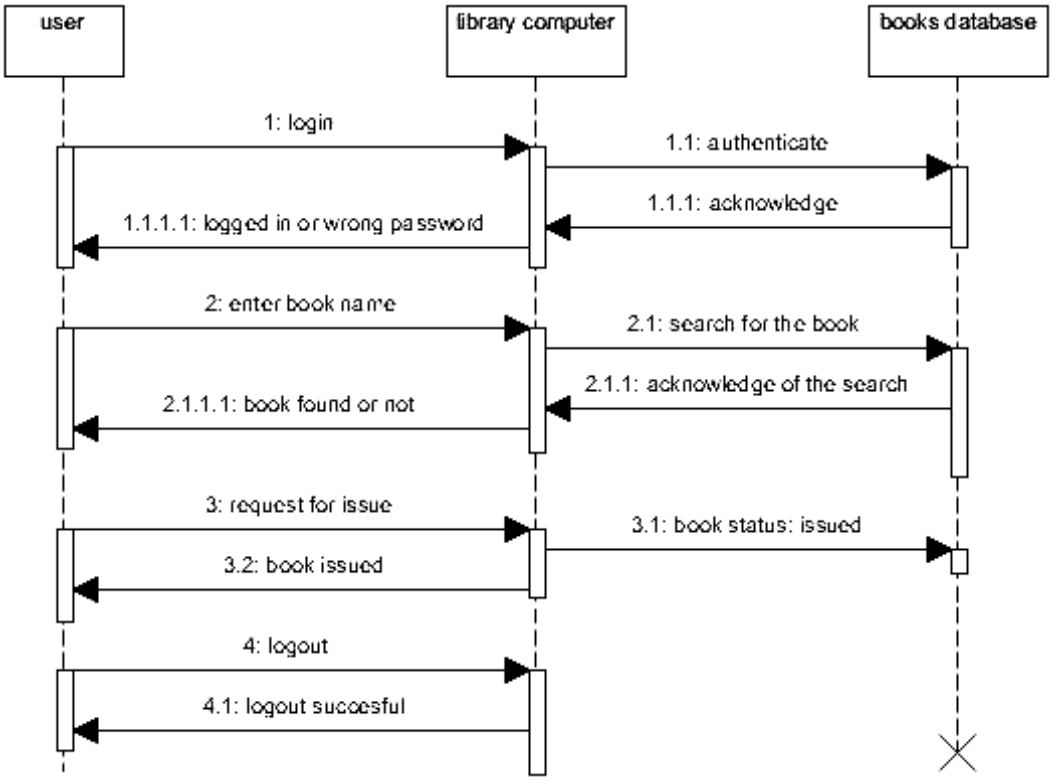


Communication path between several application servers and database servers.

Q.3 Attempt any FOUR of the following : [16]

Q.3(a) Draw sequence diagram for library Management system. [4]

(A)



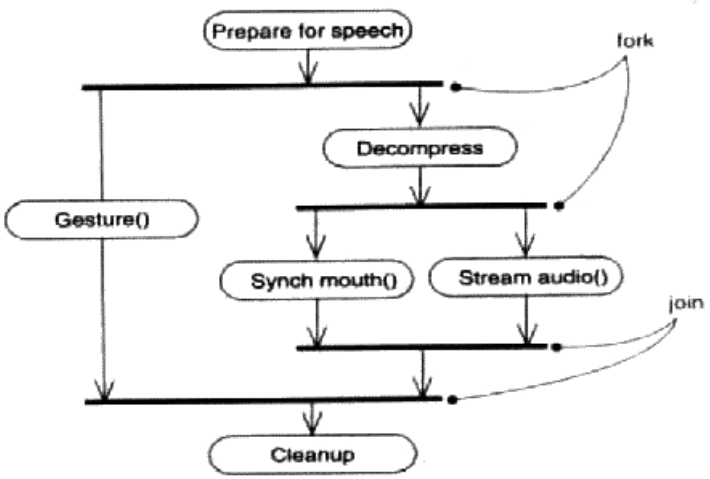
Q.3(b) Explain forking and joining with diagram. [4]

(A) Forking in Activity diagram:

1. A fork in activity diagram represents the splitting of a single flow of control into two or more concurrent flows of control.
2. A fork may have one incoming transition and two or more outgoing transitions, each of which represents an independent flow of control.
3. Below the fork, the activities associated with each of these paths continue in parallel.

Joining in Activity diagram:

1. A join in Activity diagram represents the synchronization of two or more concurrent flows of control.
2. A join may have two or more incoming transitions and one outgoing transition.
3. Above the join, activities associated with each of these paths continue in parallel.
4. At the join, the concurrent flows synchronize, means each waits until all incoming flows have reached at the join, at which point one flow of control continues on below the join.

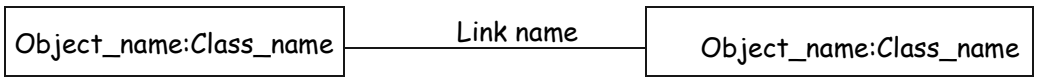


Q.3(c) Draw and explain notations used for object diagram. **[4]**

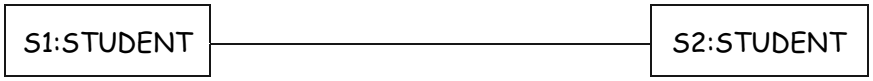
(A) 1. **Object: -** An object is a concept, abstraction or thing that has meaning for an application. Object is basic run time entity. In UML object is represented with a box including its name followed by a colon and class name. Object and class name both are written in bold face with underline.
Object can have attributes. Attributes are specified in the second part of the block. Attribute name is followed by value.

Notation	Example
Oject_name:Class_name attribute_name=value	S1:STUDENT roll no=1

2. **Link-** It is physical or conceptual connection among objects. It is used to show relationship among objects. It is represented with a solid line connecting two objects. Name of the link is written in italic form above line.



Example:-



Q.3(d) Explain SDLC in UML. **[4]**

(A) ***Inception*** is the first phase of the process, when the seed idea for the development is brought up to the point of being at least internally sufficiently well-founded to warrant entering into the elaboration phase.

Elaboration is the second phase of the process, when the product requirements and architecture are defined. In this phase, the requirements are articulated, prioritized, and baselined. A system's requirements may range from general vision statements to precise evaluation criteria, each specifying particular functional or nonfunctional behavior and each providing a basis for testing.

Construction is the third phase of the process, when the software is brought from an executable architectural baseline to being ready to be transitioned to the user community. Here also, the system's requirements and especially its evaluation criteria are constantly reexamined against the business needs of the project, and resources are allocated as appropriate to actively attack risks to the project.

Transition is the fourth phase of the process, when the software is delivered to the user community. Rarely does the software development process end here, for even during this phase, the system is continuously improved, bugs are eradicated, and features that didn't make an earlier release are added.

One element that distinguishes this process and that cuts across all four phases is an iteration.

An ***iteration*** is a distinct set of work tasks, with a baselined plan and evaluation criteria that results in an executable system that can be run, tested, and evaluated. The executable system need not be released externally. Because the iteration yields an executable product, progress can be judged and risks can be reevaluated after each iteration. This means that the software development life cycle can be characterized as involving a continuous stream of executable releases of the system's architecture with a midcourse correction after each iteration to mitigate potential risk. It is this emphasis on architecture as an important artifact that drives the UML to focus on modeling the different views of a system's architecture.

Q.3(e) Describe << include >> and << extend >> relationships in use case diagram. [4]

(A) Include relationship is use case of indicates direct incorporation of one use case to another. Whereas extend relationship is indirect incorporation.

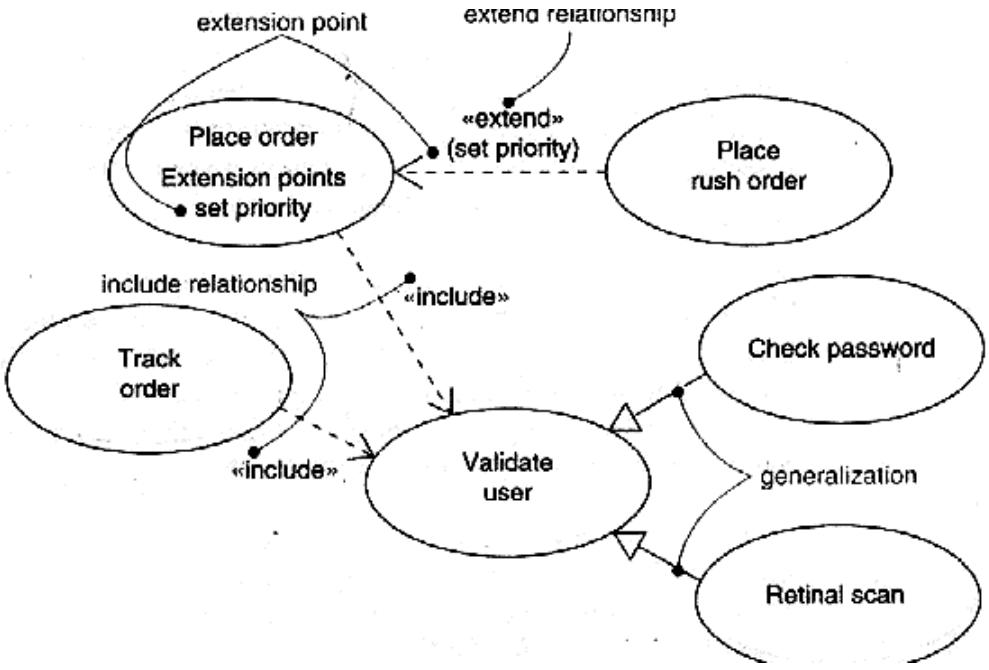
<<include>> relationships:

- An include relationship between use cases means that the base case explicitly incorporates the behavior of another use case at a location specified in the base.
- The include use case never stand alone, but is only instantiated as part of some larger base that include it.
- An include relationship as a dependency can be render with stereotyped as include. To specify the location in a flow of events in which the base use case includes the behavior of another, simply write include followed by the name of the use case.

<<extend>> relationships:

- An extend relationship between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case.
- The base use case may stand alone, but under certain conditions, its behavior may be extended by behavior of another use case.
- An extend relationship as a dependency can be render with stereotyped as extend.

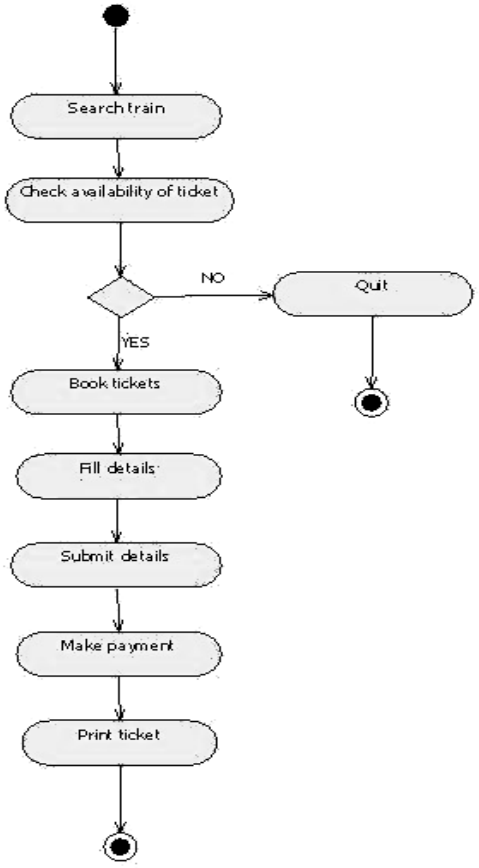
Example:-



Q.4(a) Attempt any THREE of the following : **[12]**

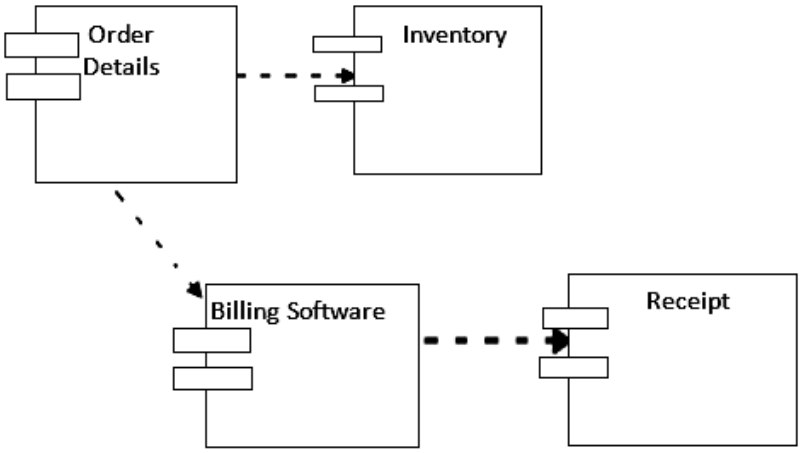
Q.4(a) (i) Draw activity diagram for online railway reservation system. **[4]**

(A)



Q.4(a) (ii) Sketch component diagram for order processing. **[4]**

(A)



Q.4(a) (iii) What is modeling? What are four principles of modeling? **[4]**

(A) Modelling

A model is an abstract representation of a system, constructed to understand system prior to building or modifying it. Building a model for a software system prior to its construction is like preparing a blueprint for building a large building. Model includes fundamental modeling concepts and semantics, notations and guidelines. Models help us to visualize a system as it is or as we want it to be. It permits us to specify the structure or behavior of a system. Models give us a template that guides us in constructing a system. It documents the decisions we have made.

Four principles of modeling:

(1) The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped: It means choose your model well. The right models will brilliantly illuminate the most wicked development problems. The wrong models will mislead you, causing you to focus on irrelevant issues.

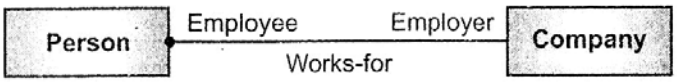
(2) Every model may be expressed at different levels of precision: All the users and developers both may visualize a system at different levels of details at different times.

(3) The best models are connected to reality: In object oriented systems, it is possible to connect all the nearly independent views of a system into one semantic whole.

(4) No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models: To understand the architecture of a system, you need multiple interlocking views such as use case view, design view, process view, implementation view and deployment view. Each of these views may have structural as well as behavioral aspects. Together these views represent a system.

Q.4(a) (iv) Discuss Role name, Constraint, Ordering. **[4]**

(A) Role Names :



Role names for an association

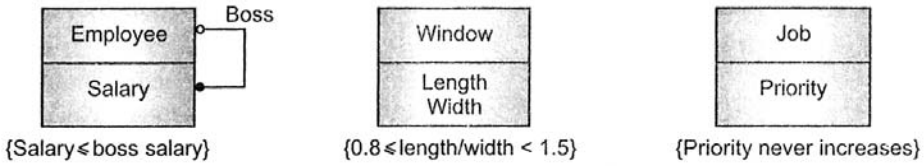
- A role is one end of association.
- A role name is a name written at one end of association.
- Figure shows role names for an association. A person assumes the role of employee with respect to a company and a company assumes the role of employer with respect to a person.
- A role name is written next to the association line near the class that plays the role.
- Some points regarding role names:
  1. Role names are necessary for association between two objects of the same class.
  2. It is useful to distinguish between two associations of same air of classes.
  3. All role names on the far end of associations attached to the class must be unique.
  4. It is really a derived attribute of the source class. So it should be unique.
  5. No role name should be same as an attribute name of the source class.
  6. Role name do not represent derived attributes of the participating classes.

Constraints

- A constraint is a condition that every implementation of the design must satisfy. Constraints can be placed on any kind of model element.
- A UML constraint is a restriction or condition on a UML element.
- They are written in curly braces { }. For example: { size >= 0 }.

Constraints on Objects

- Constraints are functional relationships between entities of an object model.
- The term entity includes objects, classes, attributes, links and associations. A constraint restricts the values that entities can assume.
- A constraint is a rule for a model element.
- For example, no employee's salary can exceed the salary of the employee's boss (a constraints between two things at the same time). No window will have an aspect ratio (length/width) of less than 0.8 or greater than 1.5 (a constraint between properties of a single object). The priority of a job may not increase (constraint on the same object overtime).
- Figure explains these examples. Simple constraints may be placed in object models. Complex constraints should be specified in the functional model.

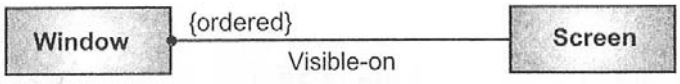


Constraints on Object

- Constraints should be expressed in a declarative manner. Usually, constraints should be converted to procedural form before they can be stated in a programming language.
- Ideally the conversion should be automatic which can be difficult or impossible to achieve. Object model capture some constraints through their very structure. For example, single inheritance implies that subclasses are mutually exclusive.
- Constraints provide one of the criterion for measuring the quality of an object model.
- A good object model captures many constraints through its structure. If often requires several iterations to get the structure of a model right from the prospective of constraints.
- To accomplish more and more structural constraints, object modeling notation must be accompanied with all kinds of special constructs. There will always be constraints that must be expressed in natural languages.
- Constraints are delimited by braes and positioned near the constrained entity. A dotted line connect multiple constrained entities.
- An arrow may be used to connect a constrained entity to the entity it depends on Instantiation is a kind of constraints and therefore uses the same notation.

Ordering :

- Ordering is a special kind of facility available in OMD,
- Usually, the object on the 'many' side of (Side) an association have no external order.
- It can be considered as a set. Sometimes, the objects are externally ordered. Such kind of externally ordered objects are denoted by writing (ordered) on the "many" end of an association i.e. next to the multiplicity dot for the role.



Role names for an association

- Figure shows example of ordering. The windows are externally ordered, so only topmost window is visible at any point on the screen.

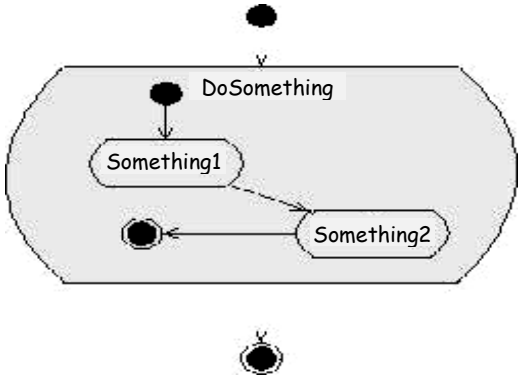
Q.4(b) Attempt any ONE of the following : [6]

Q.4(b) (i) What is importance of nested activity diagram? [6]

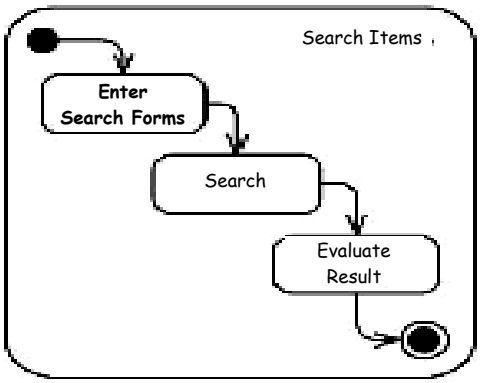
(A) Nested Activity diagram

1. Nested activity diagram contains an activity diagram inside an activity.
2. Nested activity state may reference another activity diagram that shows the internal structure of the activity state.
3. It shows the subgraph(sub activity) inside of the activity state which refers to another diagram. Sub activity can be reused as independent activity.
4. Nested activity diagram gives the result of its activities to the activity in which it resides.

Structure:

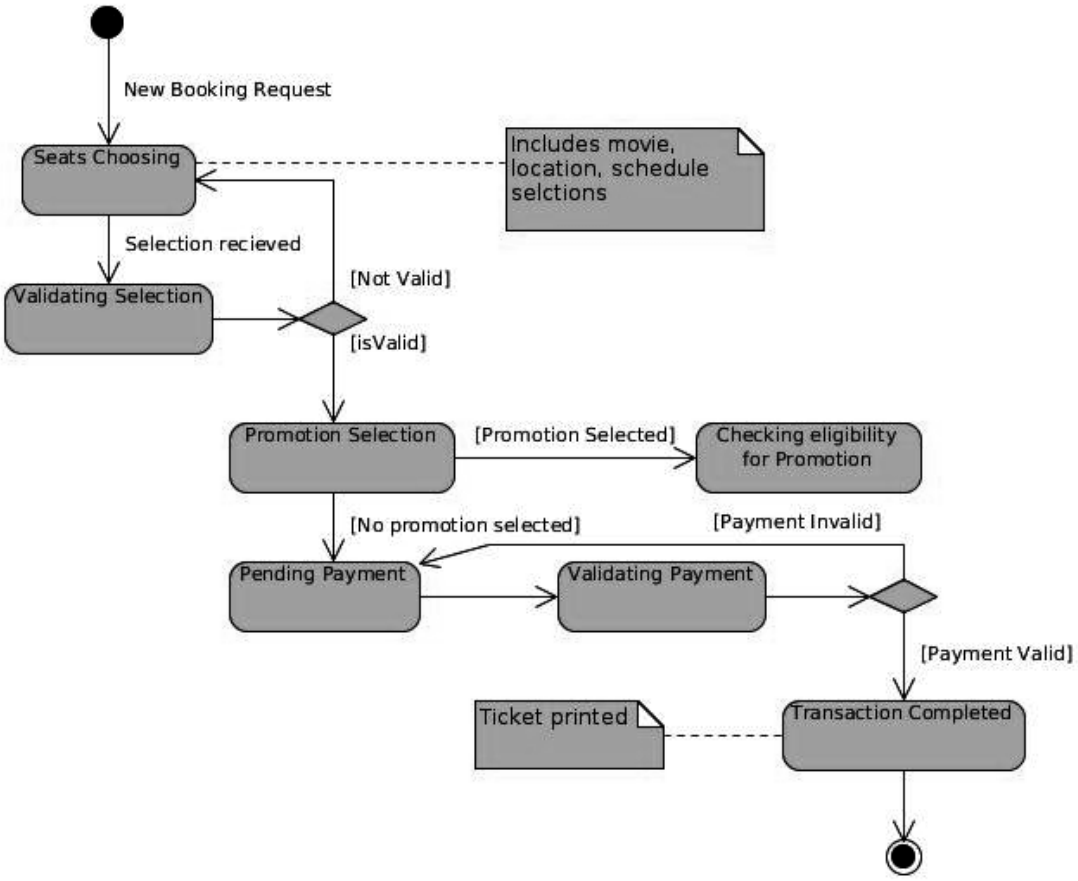


Example:



Q.4(b) (ii) Draw state diagram for online movie reservation system. [6]

(A)



Q.5 Attempt any TWO of the following : [16]

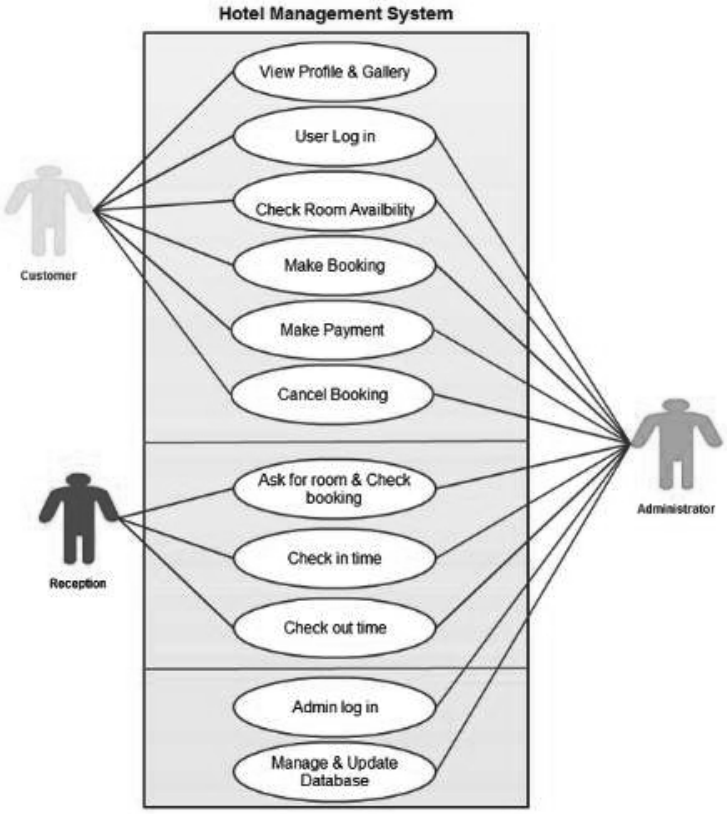
Q.5(a) What is primary and secondary actors in use case? Draw a usecase diagram for hotel management system. [8]

(A) (i) Primary or Principle Actor

The primary actor of a use case is the stakeholder that calls on the system to deliver one of its services. It has a goal with respect to the system-one that can be satisfied by its operation. The primary actor is often, but not always, the actor who triggers the use case.

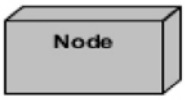
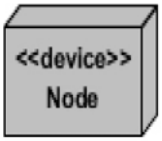
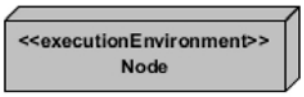
(ii) Supporting or Secondary Actor

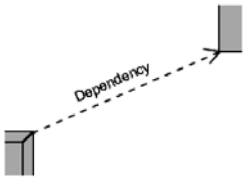
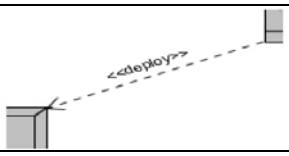
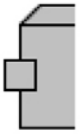
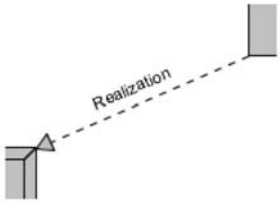
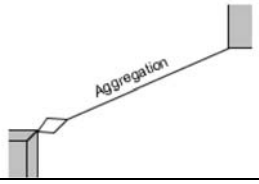
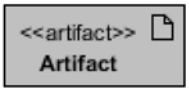
A supporting actor in a use case in an external actor that provides a service to the system under design. It might be a high-speed printer, a web service, or humans that have to do some research and get back to us.

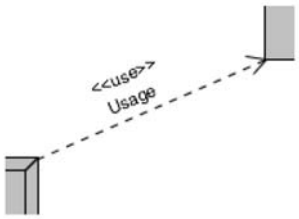


Q.5(b) Draw the symbols in deployment diagram and state use of it.. [8]

(A)

Name	Symbol	Use
Node		A node is computational resource upon which artifacts may be deployed for execution. Eg. Server
Device Node		A Device Node is a physical computational resource with processing capability upon which artifacts may be deployed for execution. APC with specific configuration, mobile phone etc.
Execution Environmental Node		An Execution Environment Node is a node that offers an execution environment for specific types of components that are deployed on it in the form of executable artifacts. Eg. OS, Database etc.
Association		An association specifies a semantic relationship that can occur between typed instances. Associations can be aggregation, dependency, generalization and realization.
Component		A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces.

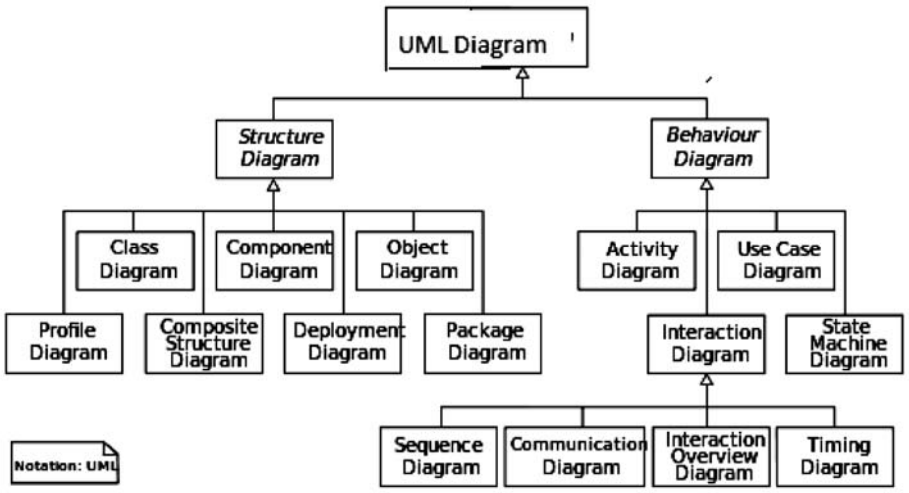
Dependency		A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
Deployment		A deployment is the allocation of an artifact or artifact instance to a deployment target.
Generalization		A generalization is a taxonomic relationship between a more general classifier and a more specific classifier.
Port		A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behavior of the) classifier and its internal parts.
Realization		Realization can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, etc.
Aggregation		A kind of association that has one of its end marked shared as kind of aggregation, meaning that it has a shared aggregation.
Artifact		An artifact is the specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system.
Interface		An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract.
Note		A note (comment) gives the ability to attach various remarks to elements. A comment carries no semantic force, but may contain information that is useful to a modeler.

Usage		A usage is a relationship in which one element requires another element (or set of elements) for its full implementation or operation. In the meta model, a Usage is a Dependency in which the client requires the presence of the supplier.

Q.5(c) Explain conceptual model of UML. **[8]**

(A) To understand the UML, it is needed to form a conceptual model of the language, and this requires learning three major elements: the UML"s basic building blocks, the rules that dictate how those building blocks may be put together, and some common mechanisms that apply throughout the UML.

UML 2.0 has 13 types of diagrams divided into three categories: Six diagram types represent static application structure, three represent general types of behaviors, and four represent different aspects of interactions. These diagrams can be categorized hierarchically as shown in the following block diagram:



Q.6 Attempt any FOUR of the following : **[16]**

Q.6(a) Discuss in brief OMT by Rumbaugh. **[4]**

(A) Brief Overview of OMT by Rumbaugh

- OMT describes a method for the analysis, design and implementation of a system using an object-oriented technique.
- OMT was developed as an approach to software development.
- A fundamental assumption of OMT is that object-oriented thinking represents a more natural and intuitive way for people to reason about reality.
- OMT is also a widely popular and comprehensive approach that exemplifies the vast number of object-oriented approaches to modeling.
- The purposes of modeling according to **Rumbaugh** are :
  1. Testing physical entities before building them (simulation)
  2. Communication with customers,
  3. Visualization, (alternative presentation of information) and
  4. Reduction of complexity.
- As a general modeling approach, OMT may be used to model all types of work.
- OMT proposes three main types of models i.e., Object model, Dynamic model, and functional model, (see figure 1)

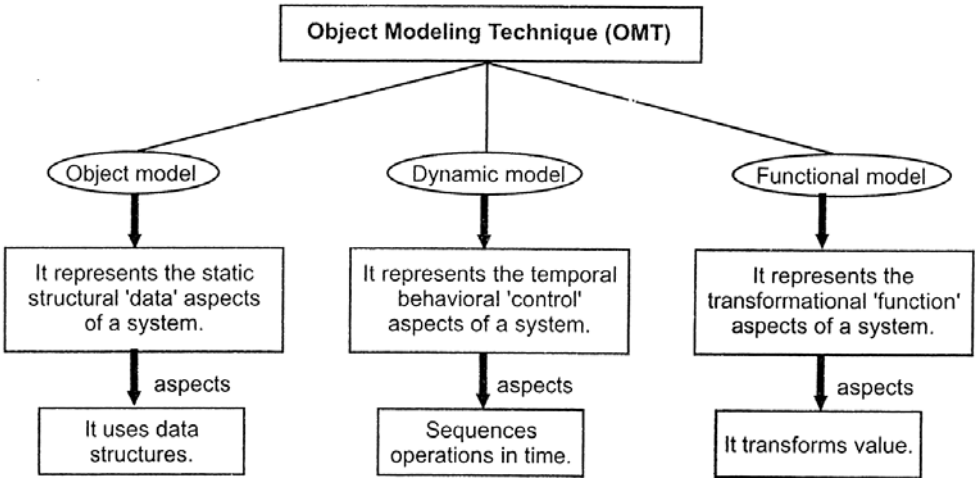


Fig. 1 : OMT Models

Q.6(b) What do you mean by artifacts in deployment diagram? **[4]**

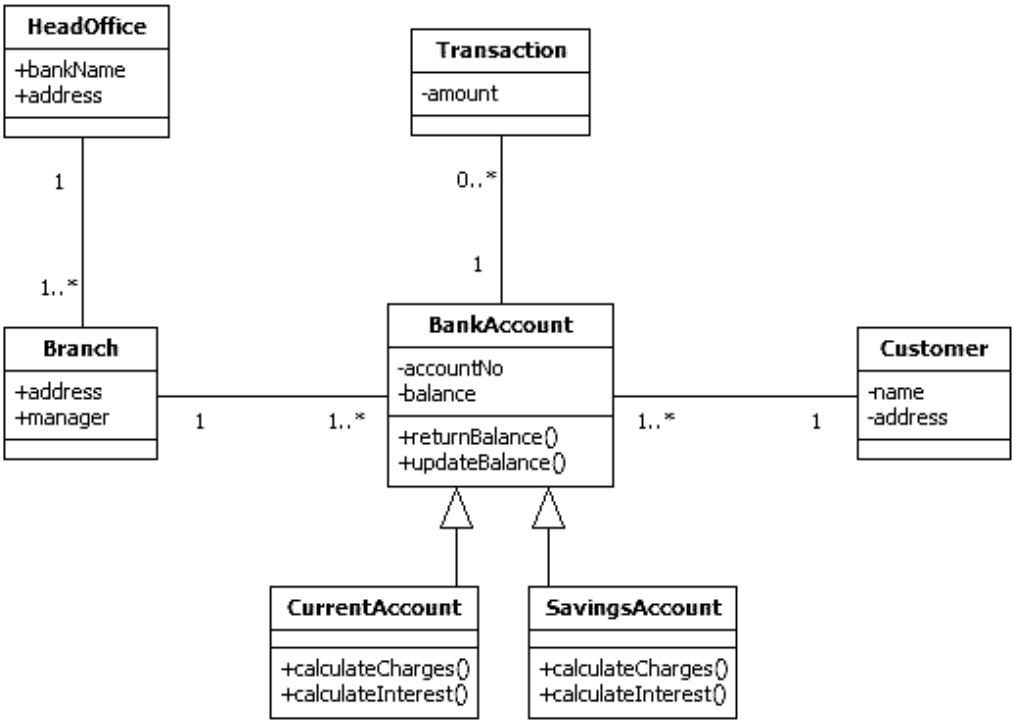
(A) Artifacts :

- The artifacts represent physical things, whereas the previous five things represent conceptual or logical things.
- An artifact is a physical and replaceable part of a system that contains physical information.
- In a system, you will encounter different kinds of deployment artifacts, such as source code files, executables, script etc.
- An artifact typically represents the physical packaging of source or run-time information.
- Graphically, an artifact is as a rectangle with the keywords <<artifact>> above the name is shown in Figure 1.

Fig. 1 : Artifacts

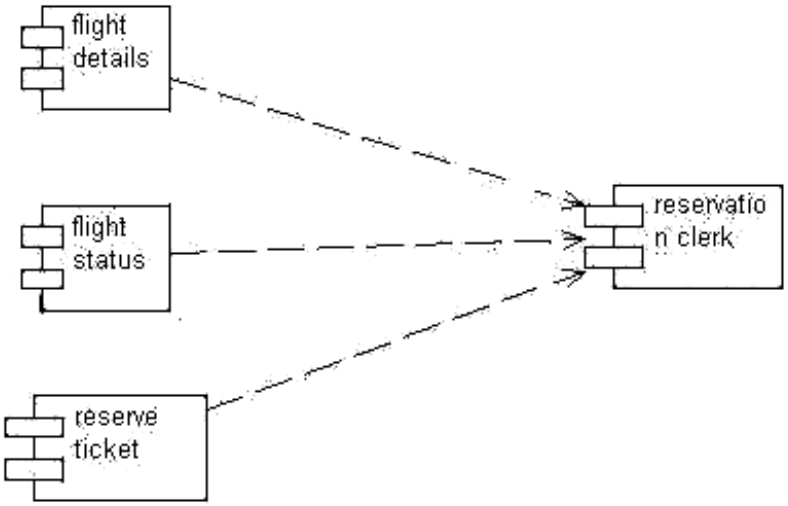
Q.6(c) Draw class diagram for bank system **[4]**

(A)



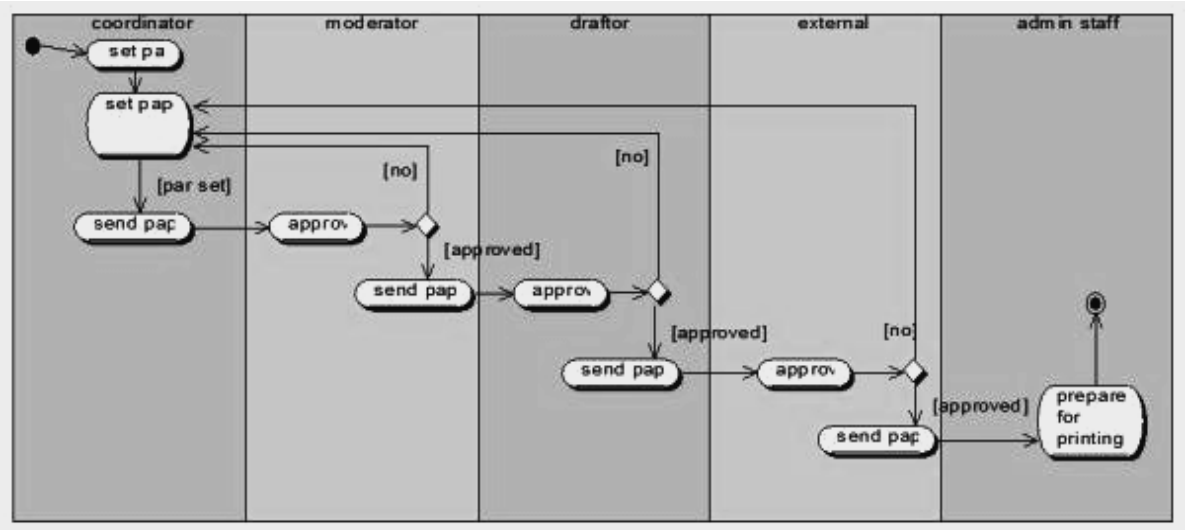
Q.6(d) Draw component diagram for airline reservation system **[4]**

(A)



Q.6(e) Draw swim lane activity diagram for setting exam paper. **[4]**

(A)



❑ ❑ ❑ ❑ ❑