Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch :- Information Technology **Class :- IV Year**
Subject :- CLAB-IV **Sem :- VIII**

Teacher Manual

PRACTICAL NO. 1

Aim: Introduction to UML.

Theory: The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML has emerged as the software blueprint language for analysts, designers, and programmers alike. It is now part of the software trade. The UML gives everyone from business analyst to designer to programmer a common vocabulary to talk about software design. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams.

A *modeling* language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. UML has a direct relation with object oriented analysis and design. UML is powerful enough to represent all the concepts exists in object oriented analysis and design.

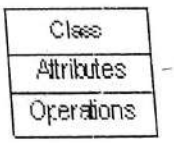
Building Blocks of the UML: Vocabulary consist of three kinds of building blocks:

1. Things
2. Relationships
3. Diagrams

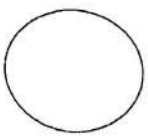
Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

Things: There are four kinds of things in the UML:

1. **Structural things:** It define the static part of the model. They represent physical and conceptual elements. Following are the brief descriptions of the structural things. There are seven kinds of structural things:
2. **Class:** *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations,



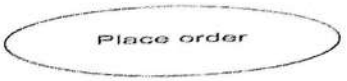
3. **Interface:** an *interface* is a collection of operations that specify a service of a class or component. Graphically, an interface is rendered as a circle together with its name.



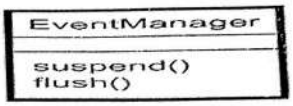
IName

4. **Collaboration:** *Collaboration* defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than the sum of all the elements. Graphically, a collaboration is rendered as an ellipse with dashed lines, usually including only its name,

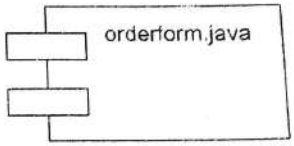
5. **Use Case:** a *use case* is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular actor. Graphically, a use case is rendered as an ellipse with solid lines.



6. **Active Class:** an *active class* is a class whose objects own one or more processes or threads and therefore can initiate control activity. Graphically, an active class is rendered just like a class, but with heavy lines.



7. **Component:** a *component* is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces. Graphically, a component is rendered as a rectangle with tabs, usually including only its name, as



8. **Node:** a *node* is a physical element that exists at run time and represents a computational resource, generally having at least some memory and, often, processing capability. Graphically, a node is rendered as a cube.

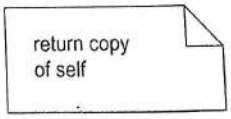
Behavioral Things: *Behavioral things* are the dynamic parts of UML models. These are the verbs of a model, representing behavior over time and space. In all, there are two primary kinds of behavioral things.

1. Interaction: an *interaction* is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. Graphically, a message is rendered as a directed line.

2. State Machine: *state machine* is a behavior that specifies the sequences of states an object or an interaction goes through during its lifetime in response to events, together with its responses to those events. Graphically, a state is rendered as a rounded rectangle

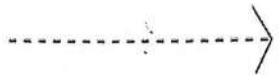
Grouping Things: *Grouping things* are the organizational parts of UML models. These are the boxes into which a model can be decomposed. In all, there is one primary kind of grouping thing, namely, packages.

Annotational Things: *Annotational things* are the explanatory parts of UML models. These are the comments you may apply to describe, illuminate, and remark about any element in a model.

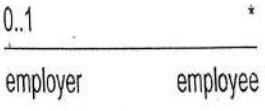


Relationships :There are four kinds of relationships:

- **Dependency:** a *dependency* is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing).



- **Association:** an *association* is a structural relationship that describes a set of links, a link being a connection among objects. Association uses Dependency, Aggregation, Composition.

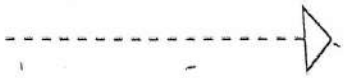


- **Aggregation:** They are used to depict elements which are made up of smaller components. It is also called as 'is a part of' relationship.

- **Composition:** It is used where components can be included in a maximum of one composition at a time

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

- **Realization:** *realization* is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out. Graphically, a realization relationship is rendered as a cross between a generalization and a dependency relationship,



Result:

main building block in object oriented modeling. It is used both for

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch :- Information Technology **Class :- IV Year**
Subject :- Clab-IV **Sem :- VIII**

Teacher Manual

PRACTICAL NO. 2

Aim: Study of UML Diagrams.

Theory: A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). Diagrams in UML are broadly classified into two categories as Static Modeling and Dynamic Modeling. UML includes nine diagrams as

1. Class diagram
2. Object diagram
3. Component diagram
4. Deployment diagram

Static Modeling

5. Use case diagram
6. Sequence diagram
7. Collaboration diagram
8. State chart diagram
9. Activity diagram

Dynamic Modeling

1. **Class Diagram**: Class Diagrams describe the static structure of a system, or how it is structured rather than how it behaves. A class diagram shows the existence of classes and their relationships in the logical view of a system. These diagrams contain the following elements.
   - Classes and their structure and behavior
   - Association, aggregation, dependency, and inheritance relationships
   - Multiplicity and navigation indicators
   - Role names

   These diagrams are the most common diagrams found in O-O modeling systems.
2. **Object Diagrams:** An object diagram shows a set of objects and their relationships. Object diagrams represent static snapshots of instances of the things found in class diagrams. Object Diagrams describe the static structure of a system at a particular time. Whereas a class model describes all possible situations, an object model describes a particular situation. Object diagrams contain the following elements:
   - **Objects:** which represent particular entities. These are instances of classes.

- **Links:** which represent particular relationships between objects. These are instances of associations.

3. **Use case diagrams**: Use Case Diagrams describe the functionality of a system and users of the system. These diagrams contain the following elements:
   - **Actors**: which represent users of a system, including human users and other systems.
   - **Use Cases**: which represent functionality or services provided by a system to users.
4. **Sequence Diagrams:** Sequence Diagrams describe interactions among classes. These interactions are modeled as exchanges of messages. These diagrams focus on classes and the messages they exchange to accomplish some desired behavior. Sequence diagrams are a type of interaction diagrams. Sequence diagrams contain the following elements:
   - **Class roles:** which represent roles that objects may play within the interaction.
   - **Lifelines**: which represent the existence of an object over a period of time.
   - **Activations**: which represent the time during which an object is performing an operation.
   - **Messages**: which represent communication between objects.
5. **Collaboration Diagram:** Collaboration Diagrams describe **interactions among classes and associations**. These interactions are modeled as exchanges of messages between classes Through their associations. Collaboration diagrams are a type of interaction Diagram. Collaboration diagrams contain the following elements.
   - **Cass roles:** which represent roles that objects may play within the interaction.
   - **Association roles**: which represent roles that links may play within the interaction.
   - **Message flows**: which represent messages sent between objects via links. Links transport or implement the delivery of the message.
6. **Statechart Diagrams**: State chart (or state) diagrams describe the states and responses of a class. Statechart Diagrams describe the behavior of a class in response to external stimuli. These diagrams contain the following elements:
   - **States**: which represent the situations during the life of an object in which it satisfies some condition, performs some activity, or waits for some occurrence.
   - **Transitions**: which represent relationships between the different states of an object.
7. **Activity Diagrams:** Activity diagrams describe the activities of a class. These diagrams are similar to Statechart diagrams and use similar conventions, but activity diagrams describe the behavior of a class in response to internal processing rather than external events as in Statechart diagram.

ML Modeller is a UML diagram tool that can support you in the software development

- **Swimlanes:** which represent responsibilities of one or more objects for actions within an overall activity; that is, they divide the activity states into groups and assign these groups to object that must perform the activities.
- **Action States:** which represent atomic, or noninterruptible, actions of entities or steps in the execution of an algorithm.
- **Action flows:** which represent relationships between the different action states of an entity.
- **Object flows:** which represent the utilization of objects by action states and the influence of action states on objects.

8. **Component Diagram:** Component diagrams describe the organizations and dependencies among software Implementation components. These diagrams contain components, which represent Distributable physical units, including source code, object code, and executable code. These are static implementation view of a system.
9. **Deployment Diagrams:** Deployment diagrams describe the configuration of run-time processing resource elements and the mapping of software implementation components onto them. These Diagrams contain components and nodes, which represent processing or computational resources, including computers, printers, etc.

Result: Thus we studied UML Diagrams.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: -Information Technology **Class: - IV Year**
Subject: -CLab IV **SEM: - VIII**

Teacher Manual

PRACTICAL NO. 3

Aim: Design a class diagram for company.

S/W Required: Staruml or Umbrello

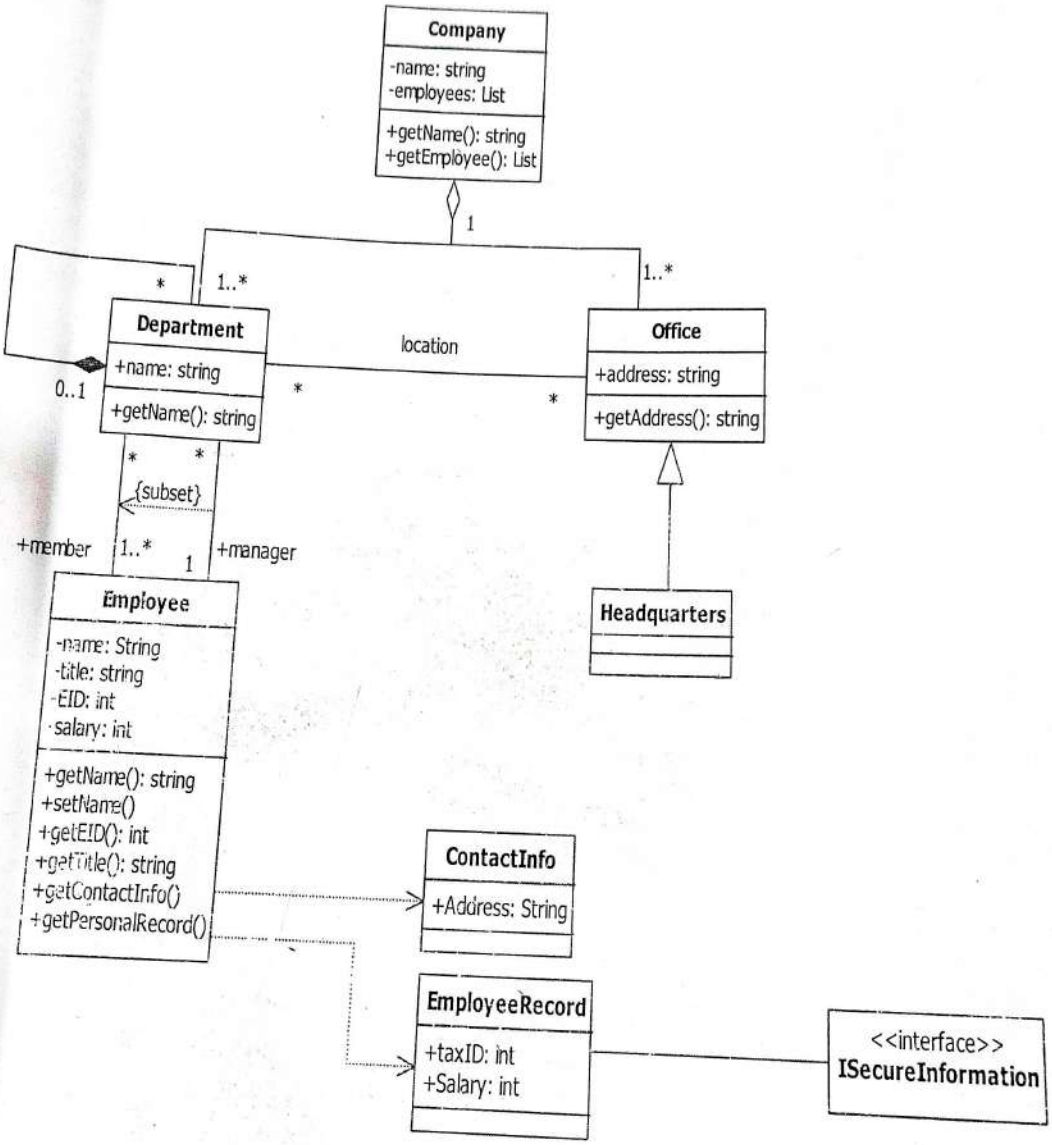
Theory: Umbrello UML Modeller is a UML diagram tool that can support you in the software development Process. Especially during the analysis and design phases of this process, Umbrello UML Modeler will help you to get a high quality product. UML can also be used to document your software designs to help you and your fellow developers.

The class diagram is the main building block in object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed. A class diagram shows a set of classes, interfaces, collaborations and their relationships and also contain packages or subsystems used to group the things. notes and constraints for comments,.

Here we are constructing a class diagram for company which consists of departments. Departments are located in one or more offices. One office acts as a headquarter. Each department has a manager who is recruited from the set of employees.

Construction of a Class Diagram

1. Identify the classes that are modeled
2. Relate the classes, giving them association names.
3. Consolidate similar classes.
4. Identify any appropriate role names.
5. Add classes for any independent functionality required being encapsulated in another class.
6. Add attributes and operations to provide the functionality required in the class diagram.
7. Detail all the operations and attributes giving them data types and parameters.
8. Identify the multiplicity used throughout.



Result: Thus we have design a class diagram for company.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: -Information Technology **Class: - IV Year**
Subject: -CLAB IV **SEM: - VIII**

Teacher Manual

PRACTICAL NO. 4

Aim: Design an object diagram for company as a class & various department & employee as an object.

S/W Required: Staruml or Umbrello

Theory: Object diagrams model the instances of things contained in class diagrams. An object diagram is a group of instances. Graphically, an object diagram is a collection of vertices and arcs. An object diagram is a diagram that shows a set of objects and their relationships at a point in time. An object diagram has:

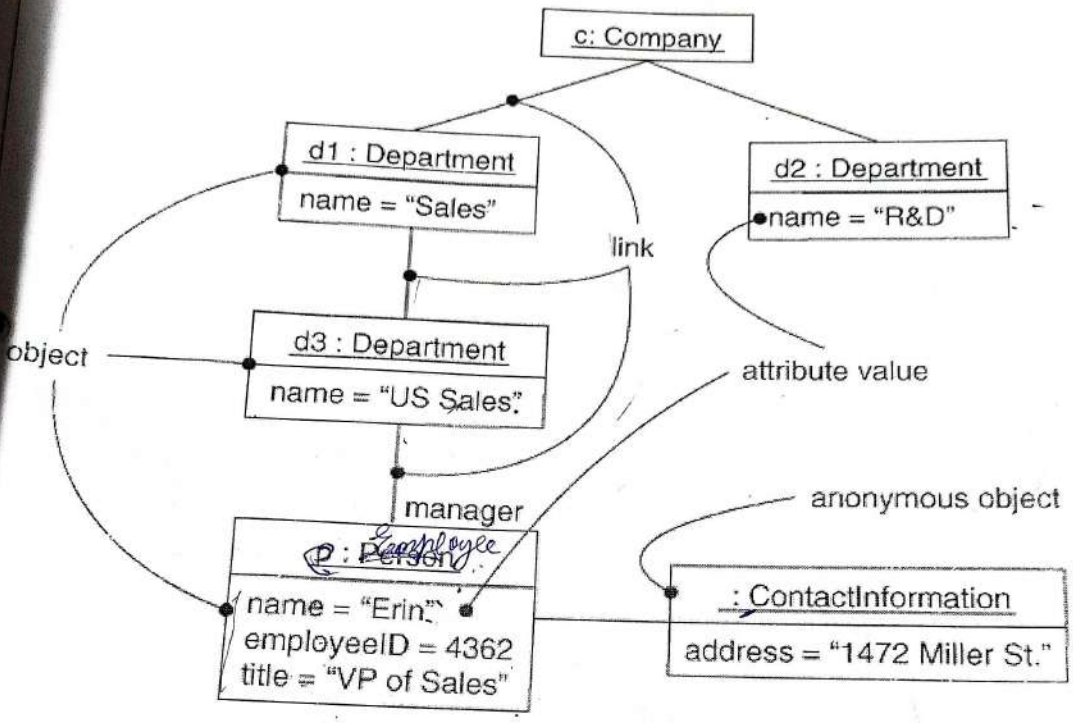
- Objects
- Links

Like all other diagrams, object diagrams may contain notes, constraints, packages and subsystems.

Construction of Object Diagram: When use objects diagrams, we have to expose the important sets of concrete or prototypical objects.

To model an object diagram,

- Identify the required mechanism you'd like to model.
- For each mechanism, identify the classes, interfaces and other elements.
- Consider one scenario that walks through these mechanisms.
- Expose the state and attribute values of each object.
- Expose the links among the objects, representing instances of associations among them.



Result: Thus we have design Object diagram for company.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: -Information Technology **Class: - IV Year**
Subject: - CLab IV **Sem: - VIII**

Teacher Manual

PRACTICAL NO. 5

Aim: Design the use-case diagram for bank management system, having deposit & withdraw is use case & clerk & customer is actor (assume other parameter).

S/W Required: Staruml or Umbrello

Theory: A use case diagram in the UML is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted. Interaction among actors is not shown on the use case diagram. Interaction among actors can be part of the assumptions used in the use case.
The following elements are available in a use-case diagram.

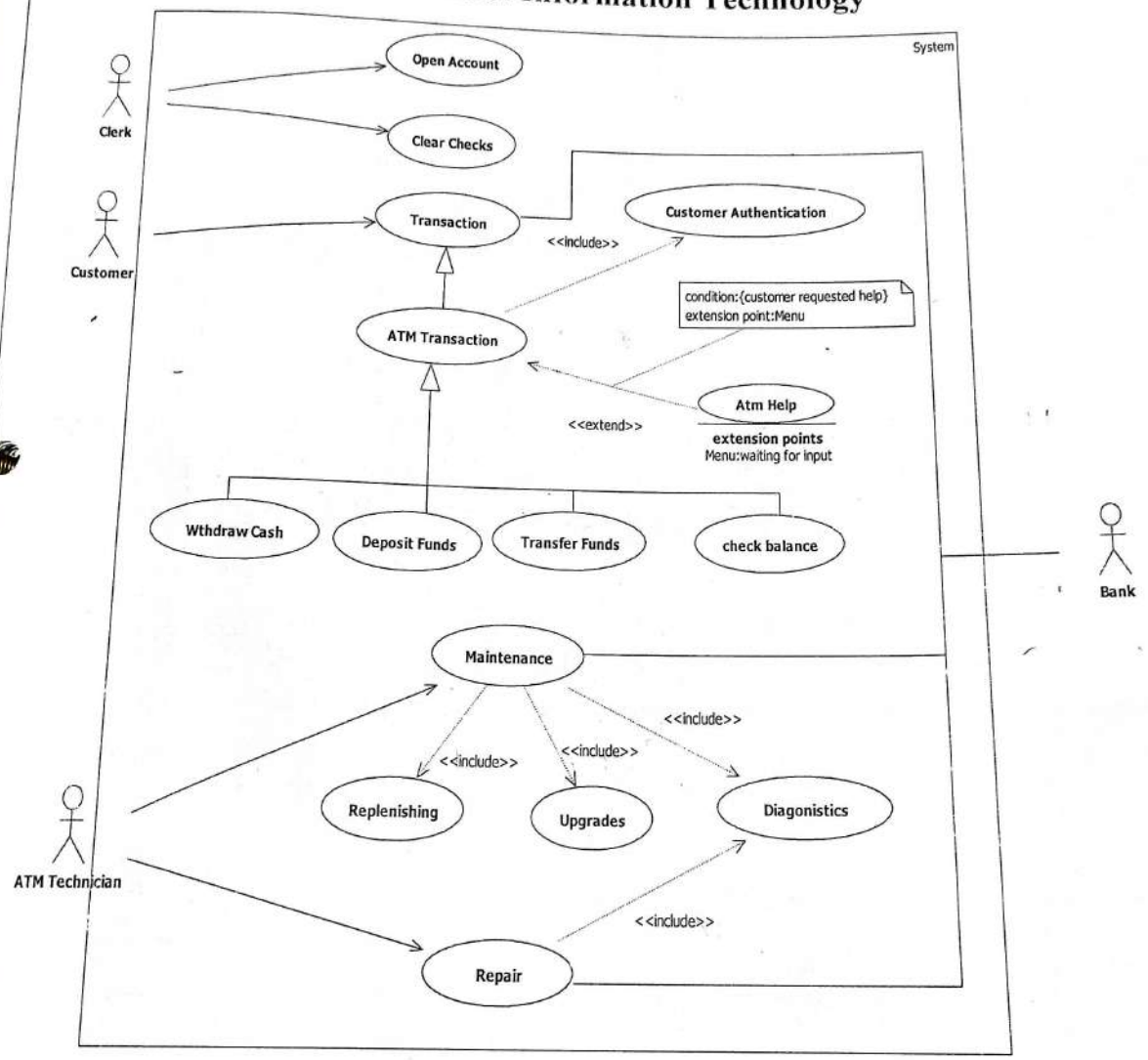
1. **Actor:** An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. Actors can be anything - humans, devices, other systems Actors can have attributes and operations.
2. **Use Case:** It describes what a system does, not how it does it. Each use case specifies a sequence of actions, including variants that the entity can perform, interacting with actors of the entity.
3. **Actor Generalization:** One popular relationship between actors is Generalization/ Specialization. This is useful in defining overlapping roles between actors. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general actor.
4. **System Boundary:** A System Boundary is a type of partition that represents the boundary between the thing you are representing with the use cases (inside the boundary) and the actors (outside the boundary). Its most typical usage is the boundary of an entire system. Use cases can be used to represent subsystems and classes and so the boundary may be more specific than an entire system.
5. **Use Case Relationships:** Four relationships among use cases are used often in practice.
   - **Include:** In one form of interaction, a given use case may include another. "Include is a Directed Relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case". The first use case often depends on the outcome of the included use case. This is useful for extracting truly common behaviors from multiple use cases into a single description. The notation is a

- Dashed arrow, with the label "«include»". The tip of the arrowhead points to the child use case and the parent use case is connected at the base of the arrow.
- **Extend:** An extend relationship defines that instances of a use case may be augmented with some additional behavior defined in an extending use case. In an *extend* relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case. Extend relationship is used to design handling exceptions. An extend relationship is depicted with a directed arrow having a dotted shaft, with the label "«extend»". The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow.
- **Generalization:** In the third form of relationship among use cases, a generalization/specialization relationship exists. A given use case may have common behaviors, requirements, constraints, and assumptions with a more general use case. The notation is a solid line ending in a hollow triangle drawn from the specialized to the more general use case.
- **Associations:** Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicate the direction of the initial invocation of the relationship or to indicate the primary actor within the use case. The arrowheads imply control flow and should not be confused with data flow.

Use case modeling provides a black box approach to functional modeling. This is especially true in early phases of modeling where systems are defined by exploring the functions that will be necessary. In the early phases the emphasis is on compiling a (as near as possible to) complete definition of functionality. How these functions will work and how it is implemented is not considered until much later.

Constructing of Use Cases diagram:

- Identify what tasks the actor must perform
- Identify what resources the actor requires in terms of data read/written
- Keep the use cases simple



Result: Thus we have design use case diagram for bank management System.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: - Information Technology **Class: - IV Year**
Subject: -CLab IV **Sem: - VIII**

Teacher Manual

PRACTICAL NO. 6

Aim- Design Sequence diagram for Online Shopping & explain in detail.
S/W Required: Staruml or Umbrello

Theory- A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construction of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner. A sequence diagram shows a set of objects and the messages sent and received by those objects accomplish some desired behavior. One use of a sequence diagram is to show the behavior sequence of a use case. When the behavior is implemented, each message on a sequence diagram corresponds to an operation on a class or an event trigger on a transition in a state machine. A sequence diagram emphasizes the time ordering of messages.

Sequence diagrams contain the following elements:

- **Object:** Object as well as classes can be targets on sequence diagram which means that messages can be sent to object is denoted by rectangle.
- **Class roles:** which represent roles that objects may play within the interaction.
- **Lifelines**: which represent the existence of an object over a period of time.
- **Activations**: which represent the time during which an object is performing an operation.
- ***Stimulus*** A Stimulus is a communication between two Instances that conveys information with the expectation that action will perform. A Stimulus will cause an Operation to be invoked, raise a Signal, or cause an Instance to be created or destroyed.
- **Self-stimulus:** A message that the object sends to itself.

Procedure for changing Action Kind of stimulus

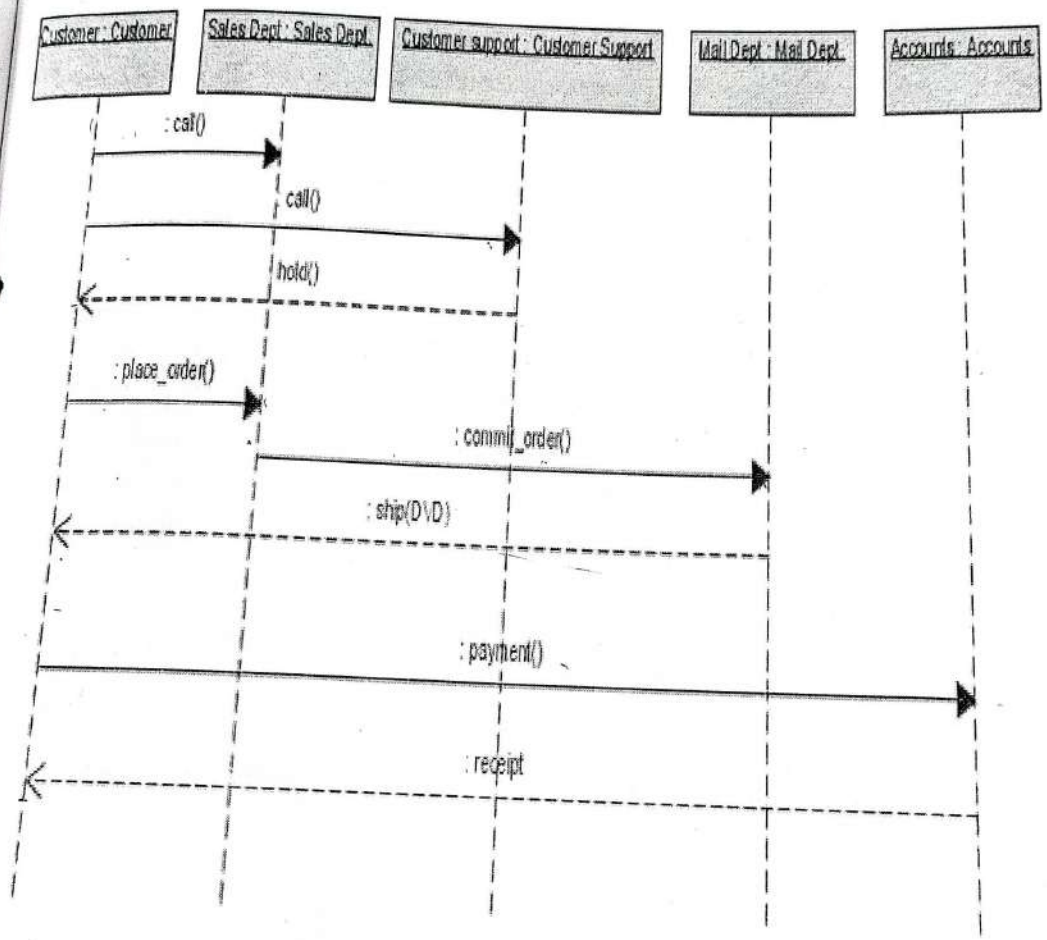
The ActionKind property of stimulus should be assigned to one of five sorts as following. To change ActionKind property, select stimulus and select the ActionKind property on the properties window.

ActionKind	Shape
Call	——→
Send	——— . >
Return	..... . >
Create	<<create>>→
Destroy	<<destroy>>→

Here we are constructing a sequence diagram for online shopping. A customer decides to upgrade her PC and purchase a DVD player online. She begins by calling the sales department of PC vendor and they tell her to contact customer support. She than calls customer support and they put her on hold while talking to engineering. Finally, customer support tells the customer about several supported DVD options. The customer chooses a DVD and it is shipped by the mail department. The customer receives the DVD, installs it satisfactorily and then mails her payment to accounts department.

So the following things are to be identified clearly before drawing the sequence diagram:

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



Result: Thus we have designed sequence diagram for online shopping.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: - Information Technology **Class: - IV Year**
Subject: -CLab IV **Sem: - VIII**

Teacher Manual

PRACTICAL NO. 7

Aim: Design a collaboration diagram for Hospital Management System.
S/W Required: Staruml or Umbrello

Theory: There are three kinds of diagrams in UML that depict dynamic models. State diagrams describe how a system responds to events in a manner that is dependent upon its state. The other two kinds of dynamic diagram fall into a category called Interaction diagrams. They both describe the flow of messages between objects

A *collaboration diagram* is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. A collaboration diagram shows a set of objects, links among those objects, and messages sent and received by those objects. You use collaboration diagrams to illustrate the dynamic view of a system. Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other. The choice between the two depends upon what the designer wants to make visually apparent.

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. The difference is that the sequence diagram does not describe the object organization where as the collaboration diagram shows the object organization. If the time sequence is important then sequence diagram is used and if organization is required then collaboration diagram is used.

Collaboration diagrams contain the following elements.

Cass roles: which represent roles that objects may play within the interaction.

Association roles: which represent roles that links may play within the interaction.

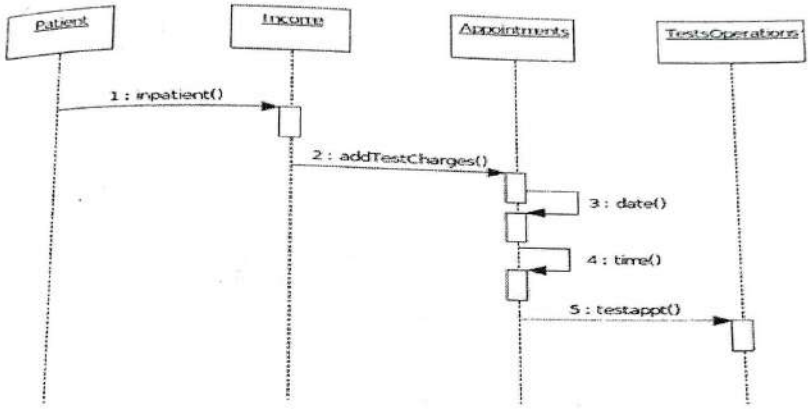
Message flows: which represent messages sent between objects via links. Links transport or implement the delivery of the message.

Hospital management system helps in registering information about patients and handles patient's query. A unique ID is generated for each patient after registration. This system deals with testing appointments as and when ID is generated the patient receives the appointment time and number and accordingly undergoes the test.

It also deals with bed allotments to various patients by checking their ID. As per doctor diagnoses the patient, gives treatment and gives suggestions to patients and prescribe laboratory tests and medicines. This system also takes care of medical equipment, doctor visit, vitals recording, patient

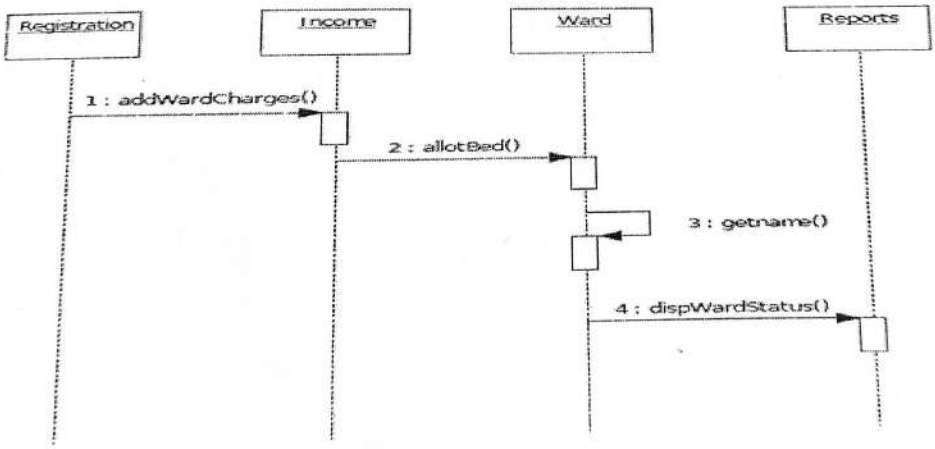
case sheet, diet ordering, blood requisition, transfer information and discharge information, maintenance of wards, inter and intrawards transfers also it generates patient's discharge summary which includes patients health at the time of discharge, medical history, various diagnosis and drug prescriptions, history of patients illness and course in hospital. Patient can pay bill through credit card, cash or cheque whose information is maintained by this system

TESTS APPOINTMENTS:



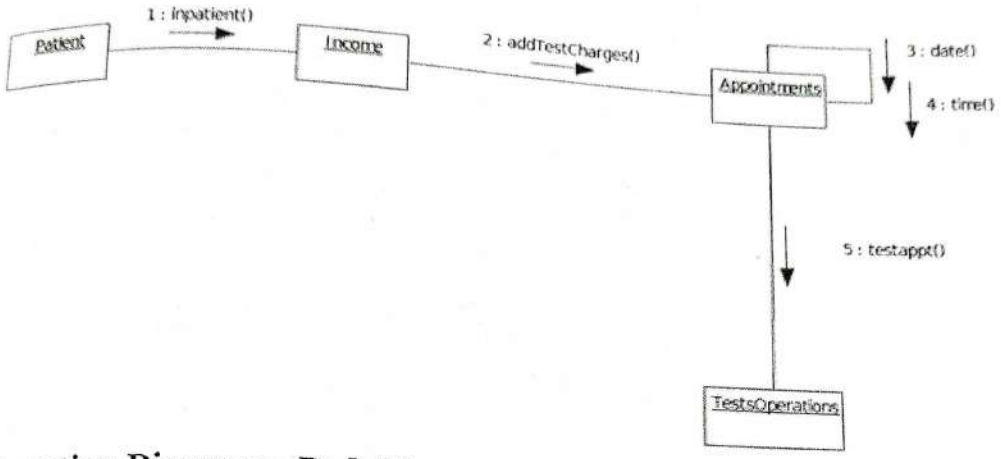
Sequence Diagram:- Bed Allotment

BED ALLOTMENT:



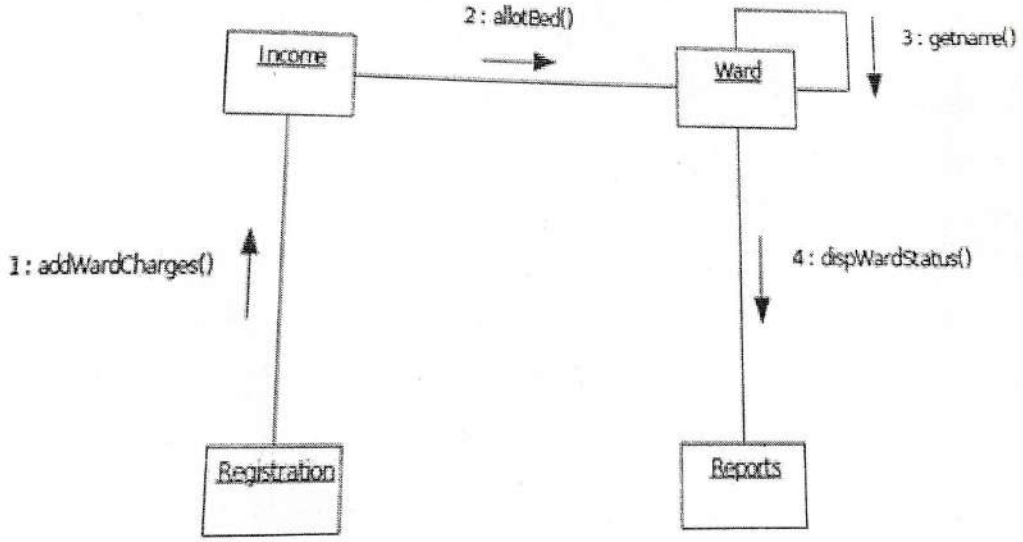
Collaboration Diagram: Test Appointments

TESTS APPOINTMENTS:



Collaboration Diagram:- Bed Allotments

BED ALLOTMENT:



Result: Thus we have design a collaboration diagram for Hospital Management System.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: - Information Technology
Subject: -CLab IV

Class: - IV Year
Sem: - VIII

Teacher Manual

PRACTICAL NO. 8

Aim: Design State Diagram for telephone system.

S/W Required: Staruml or Umbrello

Theory: State diagram is one of the five UML diagrams used to model dynamic nature of a system. State diagram describes a state machine. State machine can be defined as a machine which defines different states of an object during its lifetime and these states are changed by events. So state diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of State diagram is to model life time of an object from creation to termination.

Activity diagram is a special kind of a State diagram. As State diagram defines states it is used to model lifetime of an object.

The following elements are available in state diagram.

State: A state is a condition during the life of an object or an interaction it satisfies some condition, performs some action, or waits for some event.

Submachine State: A submachine state is a syntactical convenience that facilitates reuse and modularity. It is shorthand that implies a macro-like expansion by another state machine and is semantically equivalent to a composite state.

Initial State: An initial is a kind of pseudo state that represents the starting point in a region of a state machine. It has a single outgoing transition to the default state of the enclosing region, and has no incoming transitions. There can be one (and only one) initial state in any given region of a state machine. It is not itself a state but acts as a marker.

Final State: A final state represents the last or "final" state of the enclosing composite state. There may be more than one final state at any level signifying that the composite state can end in different ways or conditions. When a final state is reached and there are no other enclosing states it means that the entire state machine has completed its transitions and no more transitions can occur.

Junction Point: Junction Point chains together transitions into a single run-to-completion path. May have multiple input and/or output transitions. Each complete path involving a junction is logically independent and only one such path fires at one time.

Choice Point: Choice Point splits an incoming transition into several disjoint outgoing transitions. Each outgoing transition has a guard condition that is evaluated after prior actions on the incoming path have been completed. At least one outgoing transition must be enabled or the model is ill formed.

Shallow History: When reached as the target of a transition, shallow history restores the state within the enclosing composite state that was active just before the enclosing state was last exited. Does not restore any sub states of the last active state.

Deep History: When reached as the target of a transition, deep history restores the full state configuration that was active just before the enclosing composite state was last exited.

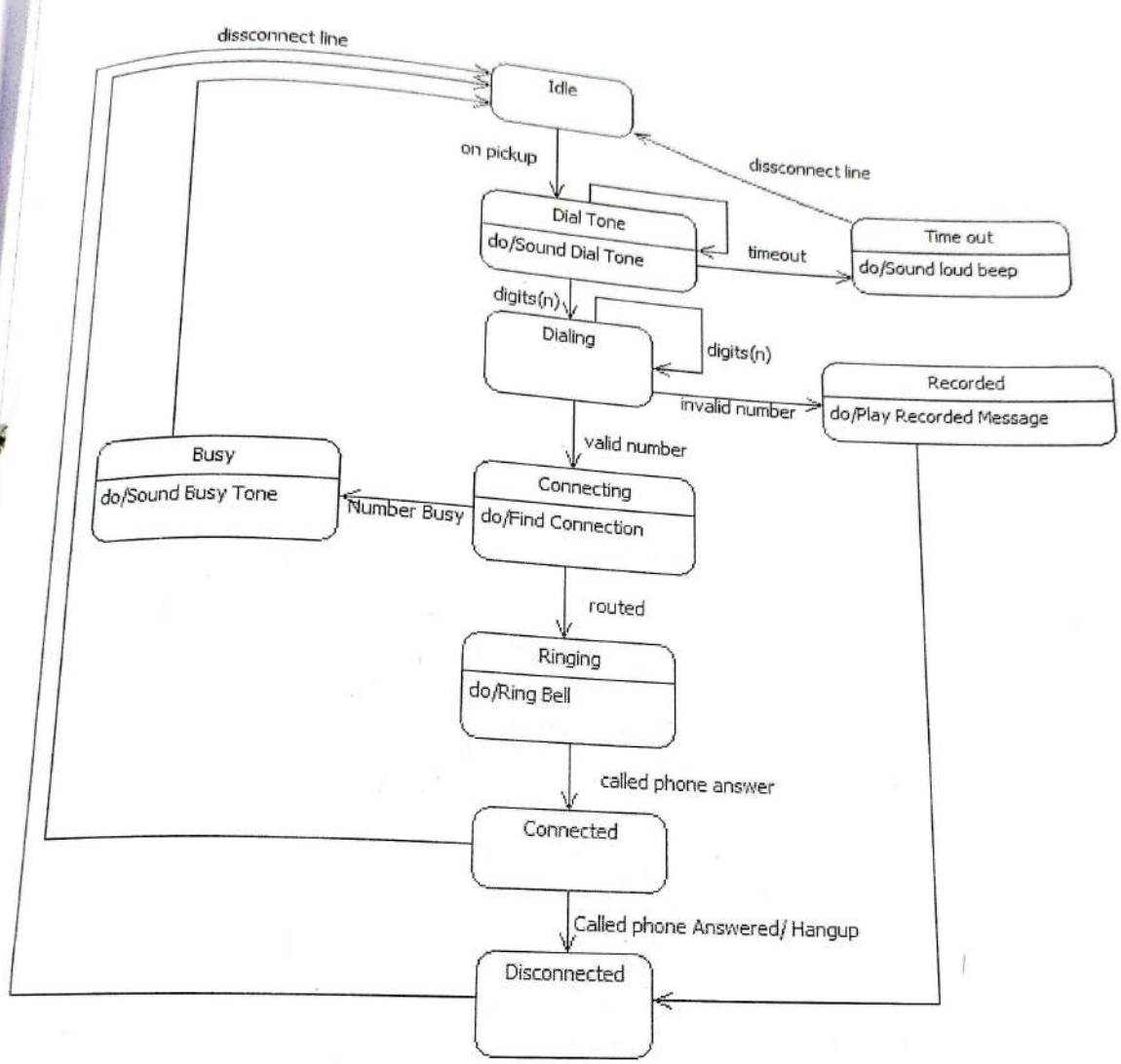
Transition: A transition is a directed relationship between a source state vertex and a target state vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to a particular event instance.

Self Transition: a transition drawn to itself.

Before drawing a State diagram we must have clarified the following points:

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

Consider the class for telephone line with following activities and states: As a start of a call, the telephone line is idle. When the phone receiver is picked from hook, it gives a dial tone and can accept the dialing of digits. If after getting dial tone, if the user doesn't dial number within time interval then time out occurs and phone line gets idle. After dialing a number, if the number is invalid then some recorded message is played. Upon entry of a valid number, the phone system tries to connect a call & routes it to proper destination. If the called person answers the phone, the conversation can occur. When called person hangs up, the phone disconnects and goes to idle state.



Result: Thus we have design state diagram for telephone system.

Sipna College of Engineering & Technology, Amravati.
Department of Information Technology

Branch: - Information Technology
Subject: -CLab IV

Class: - IV Year
Sem: - VIII

Teacher Manual

PRACTICAL NO 9

Aim: Design the activity diagram for industry management system and prepare plan

S/W Required: Staruml or Umbrello

Theory: Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc.

Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

The following elements are available in activity diagram.

Action State: An action state represents the execution of an atomic action, typically the invocation of an operation.

Sub-activity State: A sub-activity state represents the execution of a non-atomic sequence of steps that has some duration; i.e., internally it consists of a set of actions and possibly waiting for events .i.e., a sub-activity state is a hierarchical action, where an associated sub-activity graph is executed.

Decision: A state diagram expresses a decision when guard conditions are used to indicate different possible transitions that depend on Boolean conditions of the owning object.

Object Flow: An object flow is one of two types of activity edges, which are directed connection (flows) between activity nodes, the other being a control flow. As soon as the activity node at the source (tail) end of the flow is finished it presents tokens to the object flow at the target (arrowhead) end of the flow. An object flow can only carry object (data) tokens; it cannot carry control tokens.

Signal Accept State: The signal accepts may be shown as a concave pentagon that looks like a rectangle with a triangular notch in its side (either side). The signature of the signal is shown inside the symbol.

Signal Send State: The sending of a signal may be shown as a convex pentagon that looks like a rectangle with a triangular point on one side (either side). The signature of the signal is shown inside the symbol.

Swimlane: Actions and sub-activities may be organized into Swimlane. Swimlane are used to organize responsibility for actions and sub-activities. They often correspond to organizational units in a business model.

So before drawing an activity diagram we should identify the following elements:

- Activities
- Association
- Conditions
- Constraints

Once the above mentioned parameters are identified we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

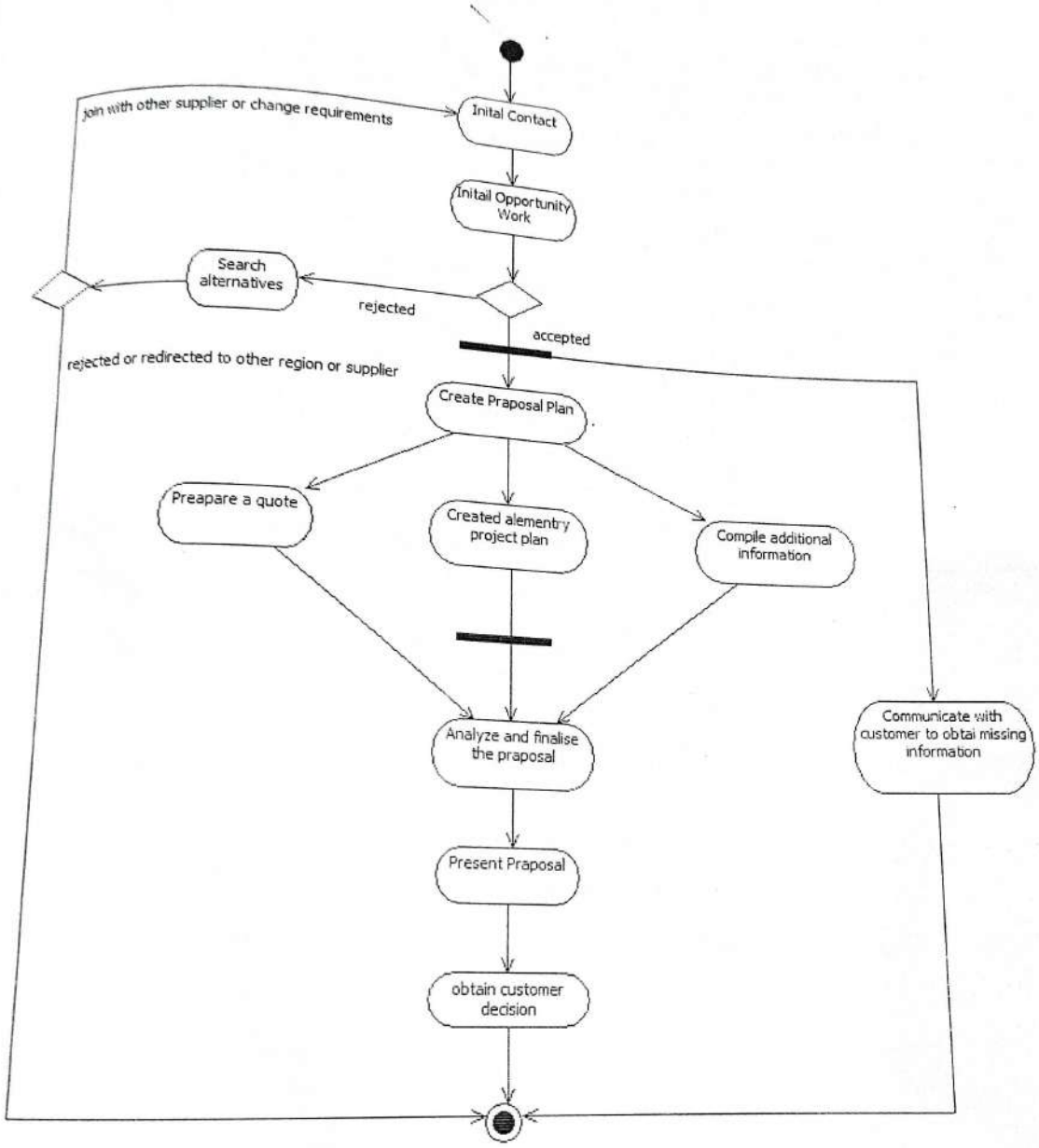
The purpose of Industry Management System is for deciding and compiling the proposals in a industry. This case study on the industry management system gives us the complete information about the project and the decisions has to be taken in a industry. We need to create new proposals according to customer requirements which mainly focuses on basic operations in a project like adding new project plan , new quotes , and updating additional information. It features a familiar and well thought-out, an attractive user interface, combined with strong searching, insertion and reporting capabilities. The report generation facility of this system helps to get a good idea of which are the perfect plans selected by the customers, makes users possible to generate hardcopy.

End-Users:

- Industry: To propose new proposals and also to cater the needs of the users.
- Customer: Need a proposal for his current problem definitions according to need
- Vendor/Supplier: To provide and meet the requirement of the prescribed.

Activities:

- Initial Contact
- Initial opportunities to wok
- If Accepted then create new proposal project plan including quote, delivery plan and some additional information if needed.
- Analyze and finalize the proposal.
- Obtain customers review by communication.
- If rejected search for new alternative.
- Consider requirements and according to that change supplier.



Result: Thus we have design activity diagram for industry management system.

Teacher Manual

PRACTICAL NO. 10

Aim: Design the component diagram for software project building.

SW Required: Staruml or Umbrello

Theory: Component diagrams are different in terms of nature and behavior. They are used to model physical aspects of a system. Physical aspects are the elements like executables, libraries, files, documents etc which resides in a node. So component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Component diagrams are used during the implementation phase of an application. But it is prepared well in advance to visualize the implementation details. This diagram is very important because without it the application cannot be implemented efficiently. A well prepared component diagram is also important for other aspects like application performance, maintenance etc.

The following elements are available in component diagram.

Package: A package is a grouping of model elements. Packages themselves may be nested within other packages. A package may contain subordinate packages as well as other kinds of model elements. All kinds of UML model elements can be organized into packages.

Component: A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.

Component Instance: A component instance is an instance of a component that resides on a node instance. A component instance may have a state.

Artifact: An Artifact represents a physical piece of information that is used or produced by a software development process. Examples of Artifacts include models, source files, scripts, and binary executable files. An Artifact may constitute the implementation of a deployable component.

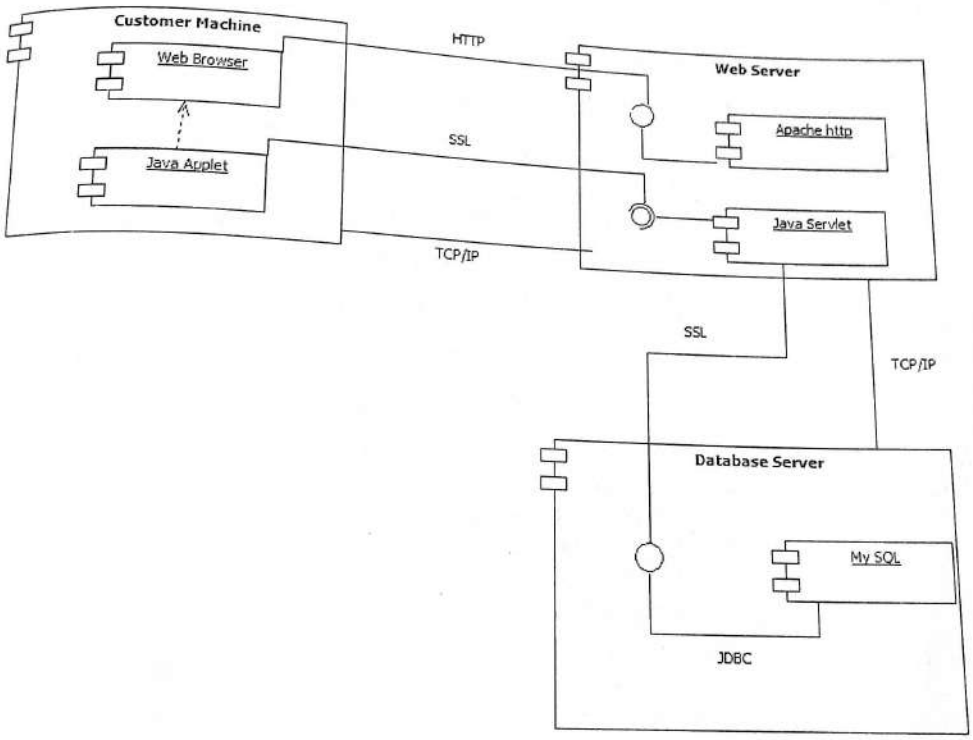
So before drawing a component diagram the following artifacts are to be identified clearly:

- Files used in the system.
- Libraries and other artifacts relevant to the application.
- Relationships among the artifacts.

Now after identifying the artifacts the following points needs to be followed:

- Use a meaningful name to identify the component for which the diagram is to be drawn.
- Prepare a mental layout before producing using tools.
- Use notes for clarifying important points.

Using the software building component diagram is for making a website using java applet, which provides the interface between customer machine and web server and requires the interface of HTTP protocol. The website component is made up of three components: Customer Machine, Web Server, and Database server components. In a simplistic sense, ports provide a way to model how a component's *provided/required* interfaces relate to its internal parts. [Note: In actuality, ports are applicable to any type of classifier (i.e., to a class or some other classifier your model might have).



Result: Thus we have design the component diagram for software project building.