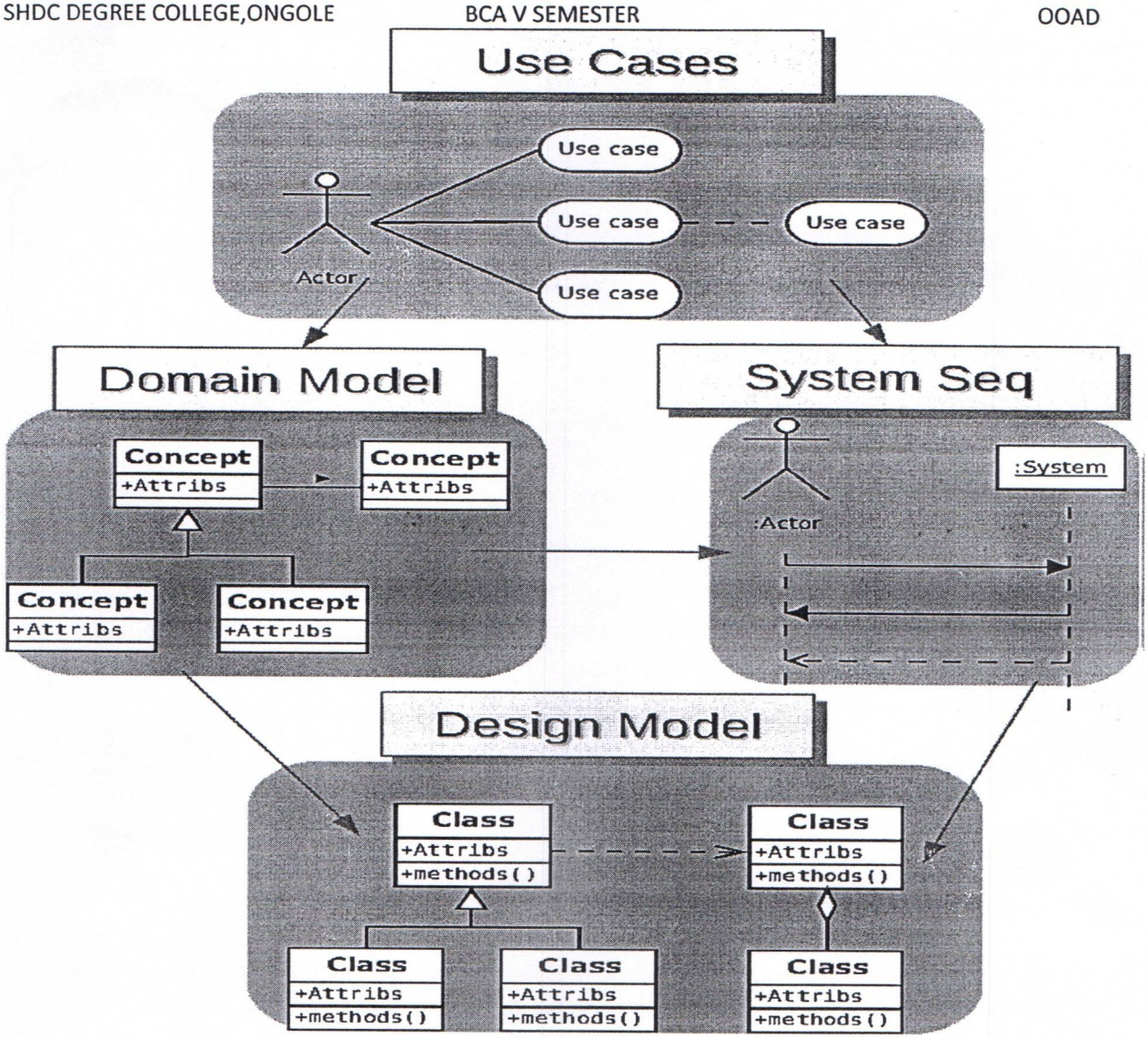
Contract CO2: enterItem

Operation:	enterItem(itemID: ItemID, quantity: integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	– A SalesLineItem instance sli was created *(instance creation).* – sli was associated with the current Sale *(association formed).* – sli.quantity became quantity *(attribute modification).* – sli was associated with a ProductDescription, based on itemID match *(association formed).*

Operation Contracts are defined in terms ofsystem operations
- Operations (say, *methods*) that the system offers as a *whole*
I☐ The system is still a black box at this stage
I☐ The System Sequence Diagrams show system events
- I.e., the SSD's messages
I☐ System operations handle system eventsWriting Operation Contracts



Operation Contracts

- Use Cases often fully describe the behavior of a system
- But they may not be enough
- Operation Contracts describe how the internal state of the concepts in the Domain Model may change
- Operation Contracts are described in terms of preconditions and postconditions

Operation Contract Syntax

Name: appropriateName

Responsibilities: Perform a function

Cross References: System functions and Use Cases

Exceptions: none

Preconditions: Something or some relationship exists

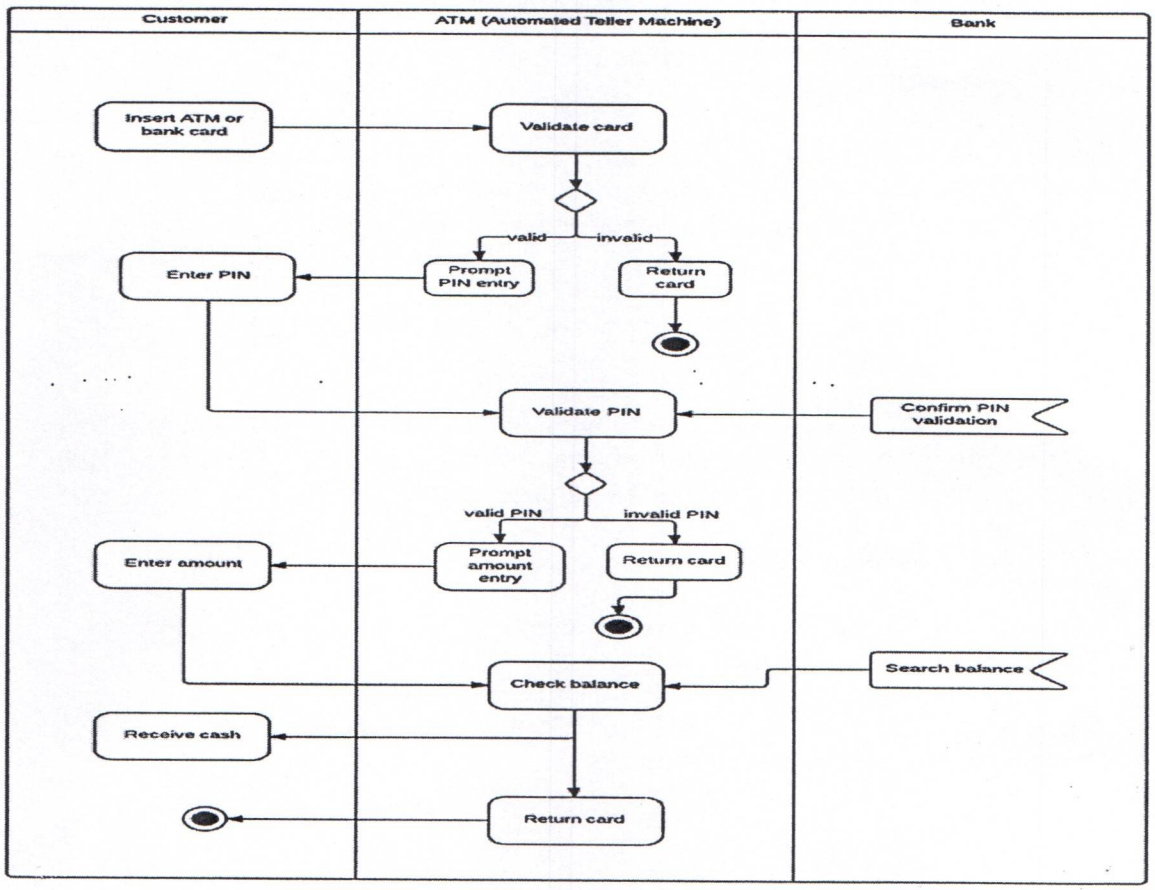
Postconditions: An association was formed

When making an operation contract, think of the state of the system before the action (snapshot) and the state of the system after the action (a second snapshot). The conditions both before and after the action should be described in the operation contract. Do not describe how the action or state changes were done. The pre and post conditions describe state, not actions.

Typical postcondion changes:

- Object attributes were changed.
- An instance of an object was created.
- An association was formed or broken.

Postconditions are described in the past tense. They declare state changes to the system. Fill in the name, then responsibilities, then postconditions.



UML Operation Contract

A UML Operation contract identifies system state changes when an operation happens. Effectively, it will define what each system operation does. An operation is taken from a system sequence diagram. It is a single event from that diagram. A domain model can be used to help generate an operation contract. The domain model can be marked as follows to help with the operation contract:

- Green - Pre existing concepts and associations.
- Blue - Created associations and concepts.
- Red - Destroyed concepts and associations.

Where to Use Deployment Diagrams?

Deployment diagrams are mainly used by system engineers. These diagrams are used to describe the physical components (hardware), their distribution, and association.

Deployment diagrams can be visualized as the hardware components/nodes on which the software components reside.

Software applications are developed to model complex business processes. Efficient software applications are not sufficient to meet the business requirements. Business requirements can be described as the need to support the increasing number of users, quick response time, etc.

To meet these types of requirements, hardware components should be designed efficiently and in a cost-effective way.

Now-a-days software applications are very complex in nature. Software applications can be standalone, web-based, distributed, mainframe-based and many more. Hence, it is very important to design the hardware components efficiently.

Deployment diagrams can be used –

- To model the hardware topology of a system.
- To model the embedded system.
- To model the hardware details for a client/server system.
- To model the hardware details of a distributed application.
- For Forward and Reverse engineering.

- Maintainability
- Portability

Before drawing a deployment diagram, the following artifacts should be identified –

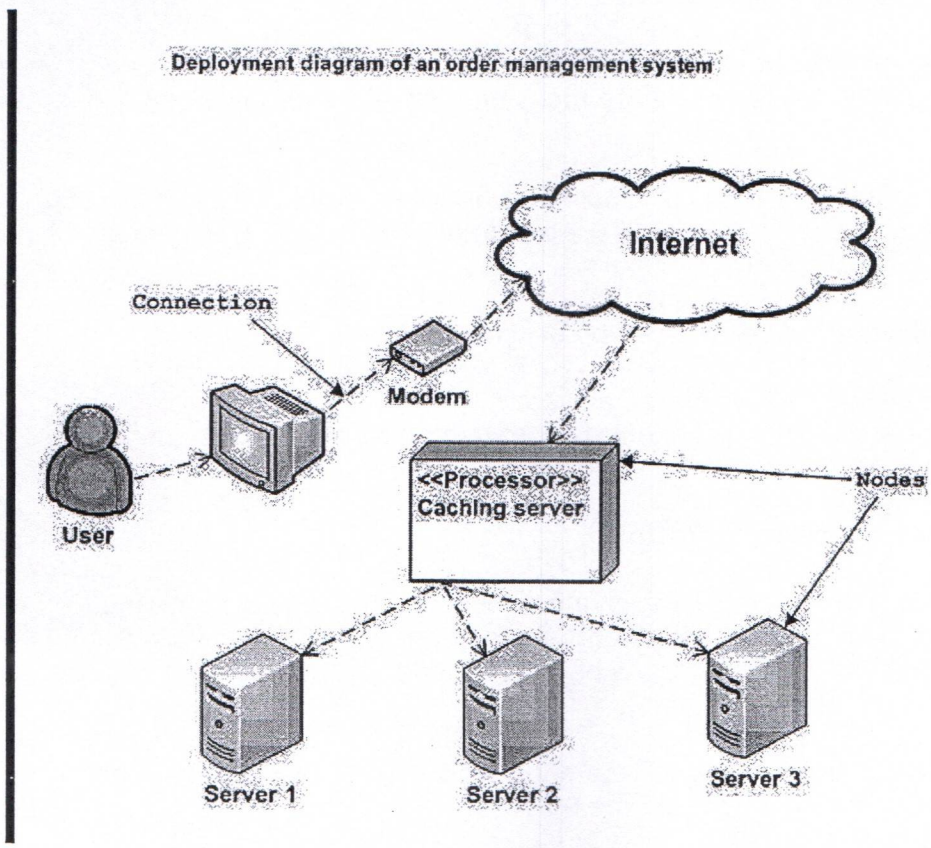
- Nodes
- Relationships among nodes

Following is a sample deployment diagram to provide an idea of the deployment view of order management system. Here, we have shown nodes as –

- Monitor
- Modem
- Caching server
- Server

The application is assumed to be a web-based application, which is deployed in a clustered environment using server 1, server 2, and server 3. The user connects to the application using the Internet. The control flows from the caching server to the clustered environment.

The following deployment diagram has been drawn considering all the points mentioned above.



- Model the components of a system.
- Model the database schema.
- Model the executables of an application.
- Model the system's source code.

UML - Deployment Diagrams

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.

Purpose of Deployment Diagrams

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.

Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as –

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

How to Draw a Deployment Diagram?

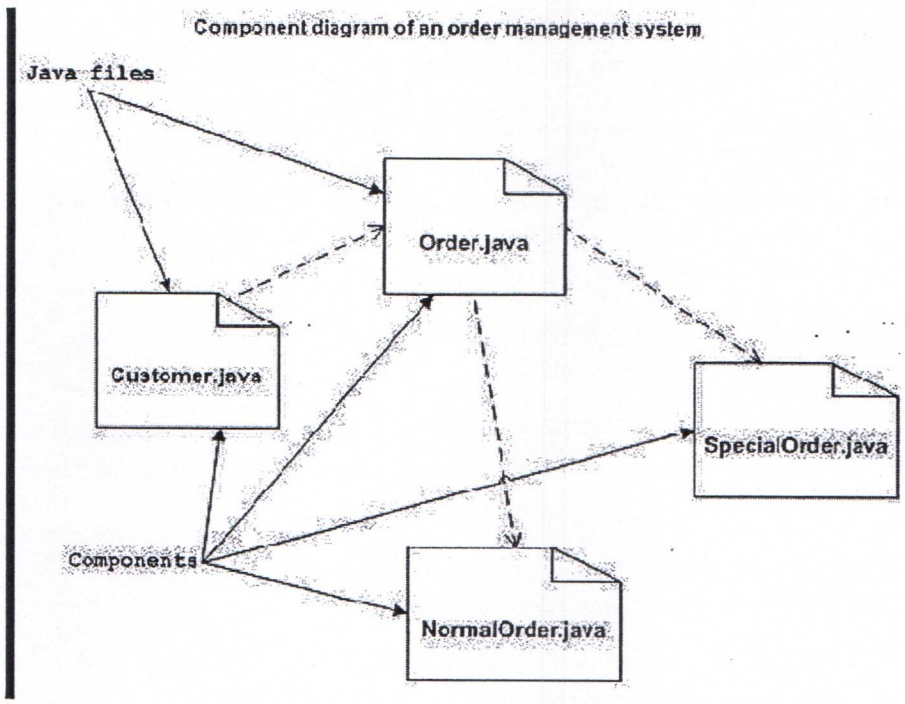
Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters –

- Performance
- Scalability

In the following diagram, four files are identified and their relationships are produced. Component diagram cannot be matched directly with other UML diagrams discussed so far as it is drawn for completely different purpose.

The following component diagram has been drawn considering all the points mentioned above.



Where to Use Component Diagrams?

We have already described that component diagrams are used to visualize the static implementation view of a system. Component diagrams are special type of UML diagrams used for different purposes.

These diagrams show the physical components of a system. To clarify it, we can say that component diagrams describe the organization of the components in a system.

Organization can be further described as the location of the components in a system. These components are organized in a special way to meet the system requirements.

As we have already discussed, those components are libraries, files, executables, etc. Before implementing the application, these components are to be organized. This component organization is also designed separately as a part of project execution.

Component diagrams are very important from implementation perspective. Thus, the implementation team of an application should have a proper knowledge of the component details

Component diagrams can be used to –

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as –

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

How to Draw a Component Diagram?

Component diagrams are used to describe the physical artifacts of a system. This artifact includes files, executables, libraries, etc

The purpose of this diagram is different. Component diagrams are used during the implementation phase of an application. However, it is prepared well in advance to visualize the implementation details.

Initially, the system is designed using different UML diagrams and then when the artifacts are ready, component diagrams are used to get an idea of the implementation.

This diagram is very important as without it the application cannot be implemented efficiently. A well-prepared component diagram is also important for other aspects such as application performance, maintenance, etc.

Before drawing a component diagram, the following artifacts are to be identified clearly –

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

After identifying the artifacts, the following points need to be kept in mind.

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

Following is a component diagram for order management system. Here, the artifacts are files. The diagram shows the files in the application and their relationships. In actual, the component diagram also contains dlls, libraries, folders, etc.

Where to Use Statechart Diagrams?

From the above discussion, we can define the practical applications of a Statechart diagram. Statechart diagrams are used to model the dynamic aspect of a system like other four diagrams discussed in this tutorial. However, it has some distinguishing characteristics for modeling the dynamic nature.

Statechart diagram defines the states of a component and these state changes are dynamic in nature. Its specific purpose is to define the state changes triggered by events. Events are internal or external factors influencing the system.

Statechart diagrams are used to model the states and also the events operating on the system. When implementing a system, it is very important to clarify different states of an object during its life time and Statechart diagrams are used for this purpose. When these states and events are identified, they are used to model it and these models are used during the implementation of the system.

If we look into the practical implementation of Statechart diagram, then it is mainly used to analyze the object states influenced by events. This analysis is helpful to understand the system behavior during its execution.

The main usage can be described as –

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes.
- Forward and reverse engineering.

(1)

(2) **UML - Component Diagrams**

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? 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.

Purpose of Component Diagrams

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

How to Draw a Statechart Diagram?

Statechart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.

Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

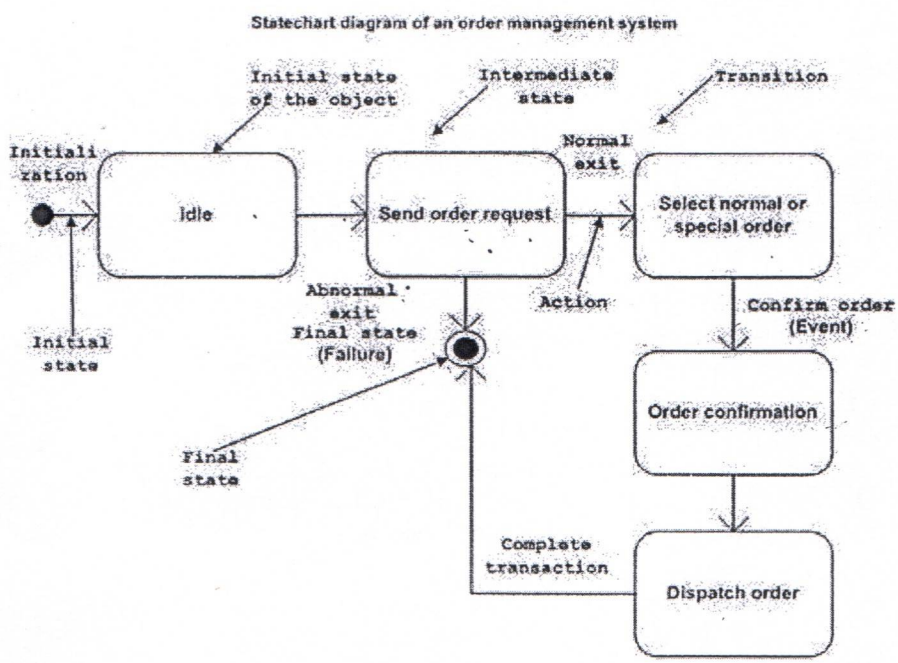
Before drawing a Statechart diagram we should clarify the following points –

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

Following is an example of a Statechart diagram where the state of Order object is analyzed

The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for the state changes of order object.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exits. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete, it is considered as a complete transaction as shown in the following figure. The initial and final state of an object is also shown in the following figure.



One way to achieve global visibility is to assign an instance to a global variable, which is possible in some languages, such as C++, but not others, such as Java.

UNIT-V

UML STATE DIAGRAMS AND MODELING

UML - Statechart Diagrams

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Activity diagram explained in the next chapter, is a special kind of a Statechart diagram. As Statechart diagram defines the states, it is used to model the lifetime of an object.

Purpose of Statechart Diagrams

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart 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. The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

Following are the main purposes of using Statechart diagrams –

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

public class Register {privateProductCatalog catalog;}"

This visibility is required because in the enterItem diagram shown in Figure, a Register needs to send the getProductDescription message to a Product - Catalog:

Parameter Visibility

Parameter visibilityfrom A to B exists when B is passed as a parameter to a method of A. It is a relatively temporary visibility because it persists only within the scope of the method. After attribute visibility, it is the second most common form of visibility in object - oriented systems.

To illustrate, when the makeLineItem message is sent to a Sale instance, a ProductDescription instance is passed as a parameter. Within the scope of the makeLineItem method, the Sale has parameter visibility to a ProductDescription.

It is common to transform parameter visibility into attribute visibility. When the Sale creates a new SalesLineItem, it passes the ProductDescription in to its initializing method (in C++ or Java, this would be its constructor).Within the initializing method, the parameter is assigned to an attribute, thus establishing attribute visibility.

Local Visibility

Local visibility from A to B exists when B is declared as a local object within a method of A. It is a relatively temporary visibility because it persists only within the scope of the method. After parameter visibility, it is the third most common form of visibility in object - oriented systems.

Two common means by which local visibility is achieved are:

Create a new local instance and assign it to a local variable. Assign the returning object from a method invocation to a local variable.

As with parameter visibility, it is common to transform locally declared visibility into attribute visibility.

An example of the second variation (assigning the returning object to a local variable) can be found in the enterItem method of class Register .

Global Visibility

Global visibilityfrom A to B exists when B is global to A. It is a relatively permanent visibility because it persists as long as A and B exist. It is the least common form of visibility in object - oriented systems.

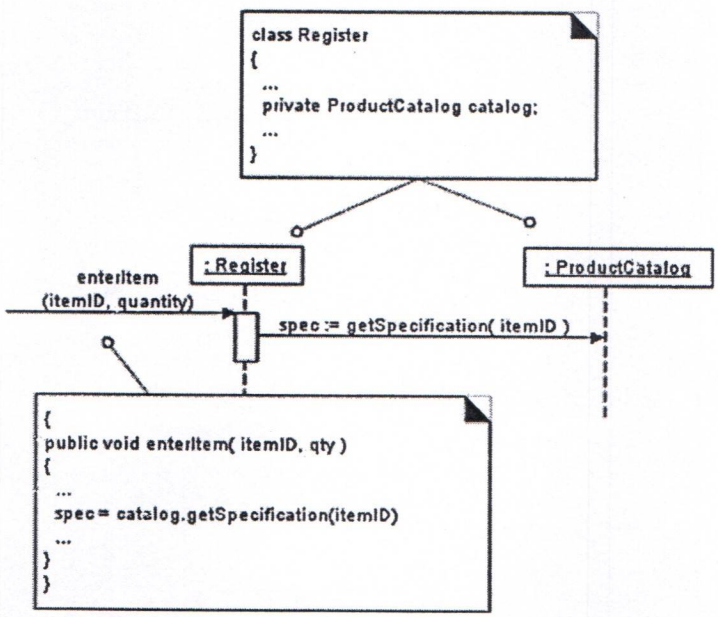
What is Visibility?

In common usage, visibilityis the ability of an object to "see" or have a reference to another object. More generally, it is related to the issue of scope: Is one resource (such as an instance) within the scope of another? There are four common ways that visibility can be achieved from object A to object B:

Attribute visibility— B is an attribute of A.
Parameter visibility— B is a parameter of a method of A.
Local visibility— B is a (non - parameter) local object in a method of A.
Global visibility— B is in some way globally visible.

The motivation to consider visibility is this:

For example, to create an interaction diagram in which a message is sent from a ... Register instance to a ProductCatalog instance, the Register must have visibility to the ProductCatalog. A typical visibility solution is that a reference to the ProductCatalog instance is maintained as an attribute of the Register.

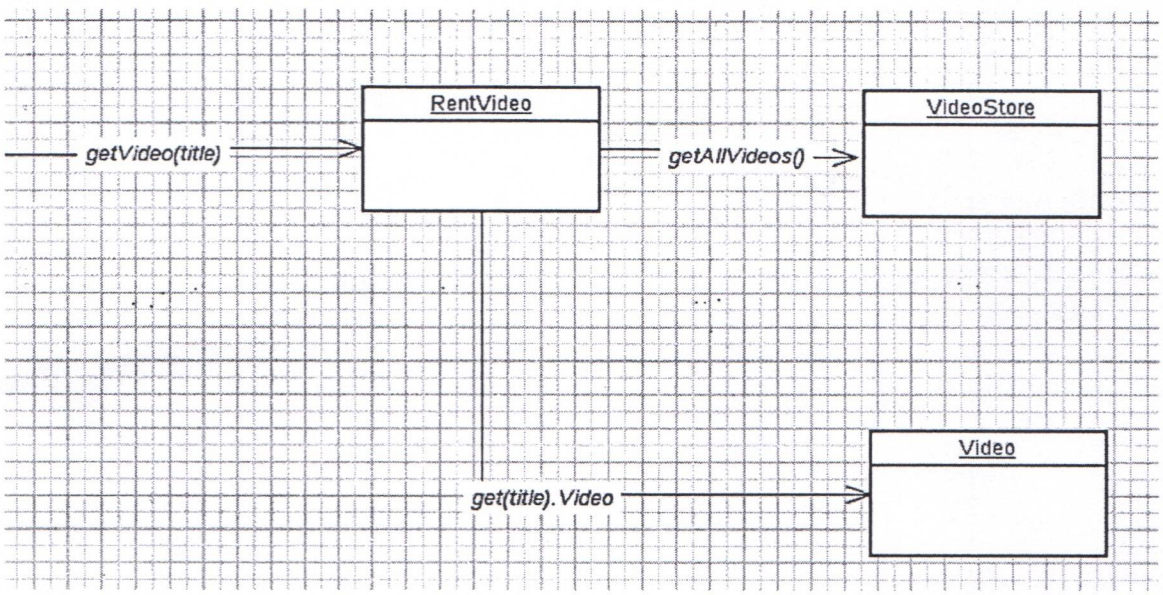


Attribute Visibility

Attribute visibility from A to B exists when B is an attribute of A. It is a relatively permanent visibility because it persists as long as A and B exists. This is a very common form of visibility in object - oriented systems.

To illustrate, in a Java class definition for Register, a Register instance may have attribute visibility to a ProductCatalog, since it is an attribute (Java instance variable) of the Register.

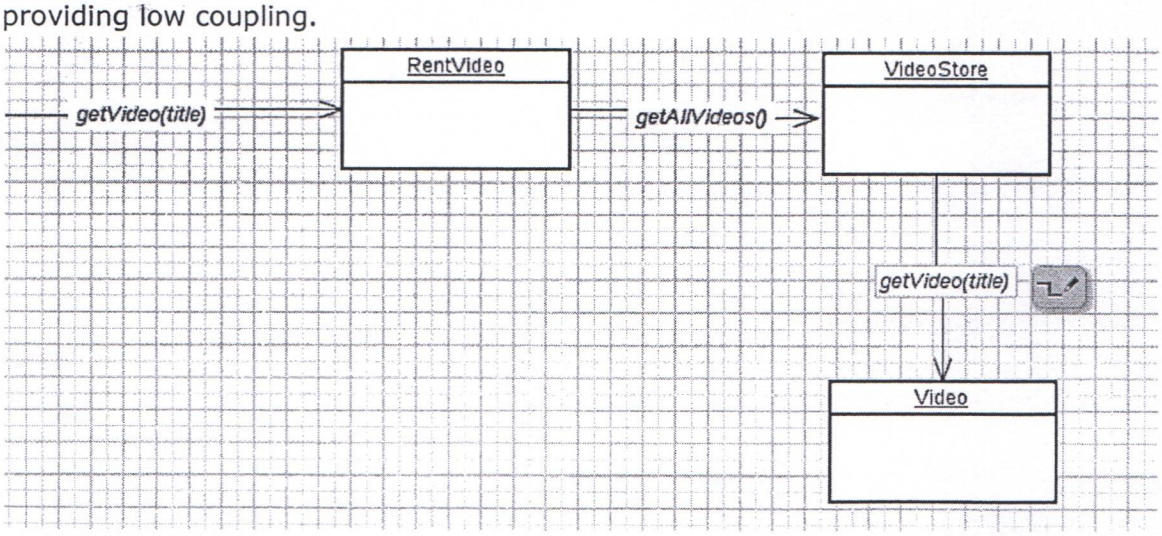
Example for poor coupling:



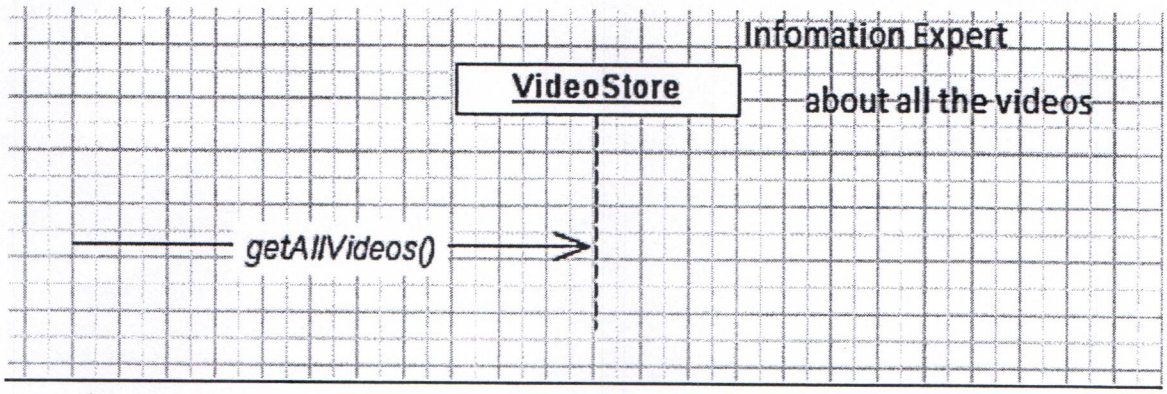
here class Rent knows about both VideoStore and Video objects. Rent is depending on both the classes.

Example for low coupling

VideoStore and Video class are coupled, and Rent is coupled with VideoStore. Thus providing low coupling.



Example for Expert:2



Low coupling

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. **Low coupling** is an evaluative pattern that dictates how to assign responsibilities to support:

- lower dependency between the classes,
- change in one class having lower impact on other classes,
- higher reuse potential.

Low coupling
- Two elements are coupled, if
- One element has aggregation/composition association with another element.
- One element implements/extends other element.

Information expert (also **expert** or the **expert principle**) is a principle used to determine where to delegate responsibilities. These responsibilities include methods, computed fields, and so on.

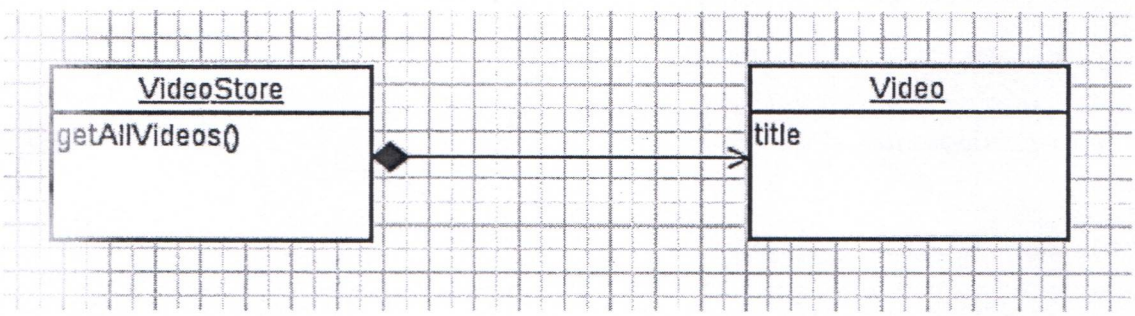
Using the principle of information expert, a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfill it, and then determine where that information is stored.

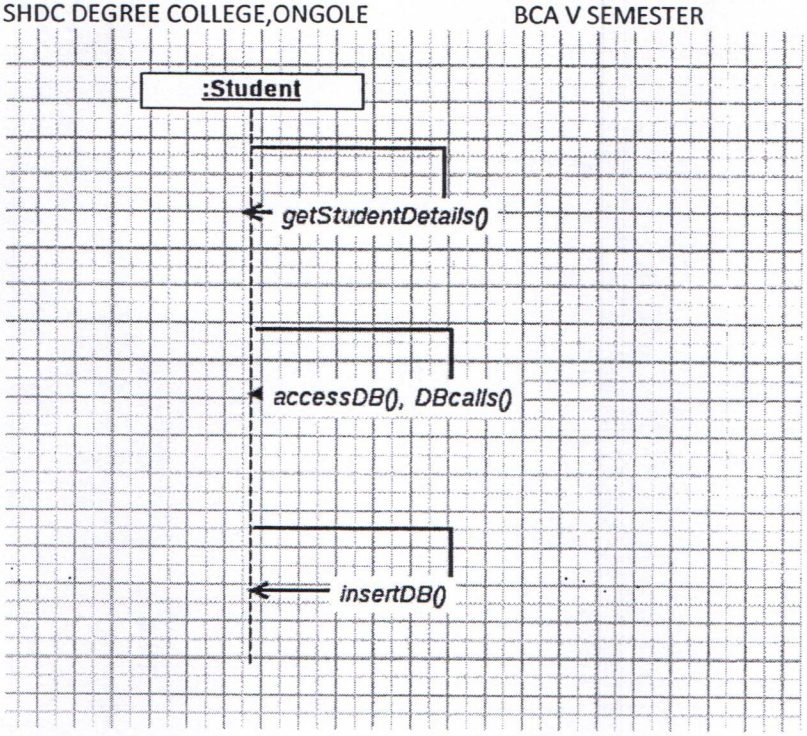
Information expert will lead to placing the responsibility on the class with the most information required to fulfill it.

Example for Expert

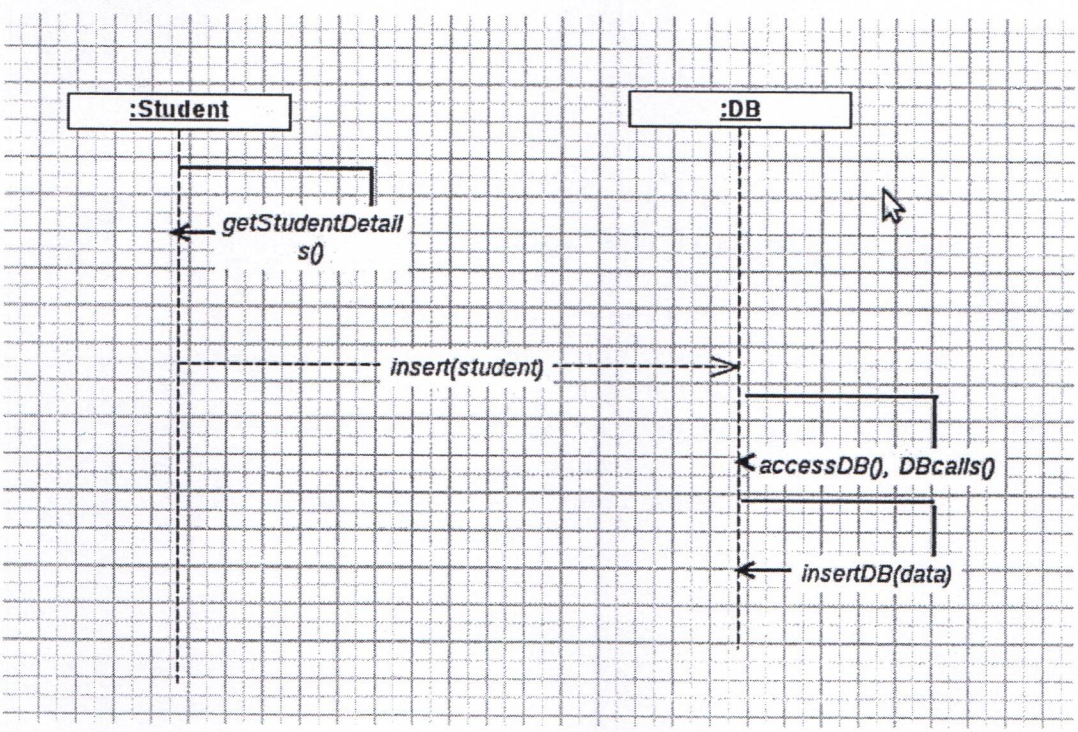
- Assume we need to get all the videos of a VideoStore.
- Since VideoStore knows about all the videos, we can assign this responsibility of giving all the videos can be assigned to VideoStore class.

- VideoStore is the information expert





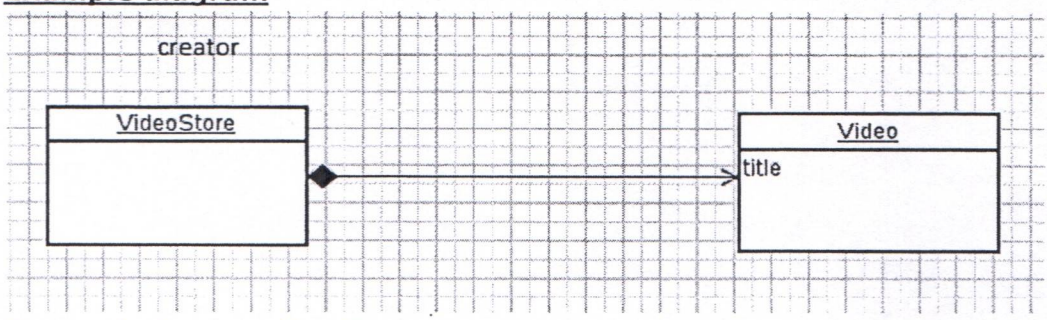
Example for High Cohesion



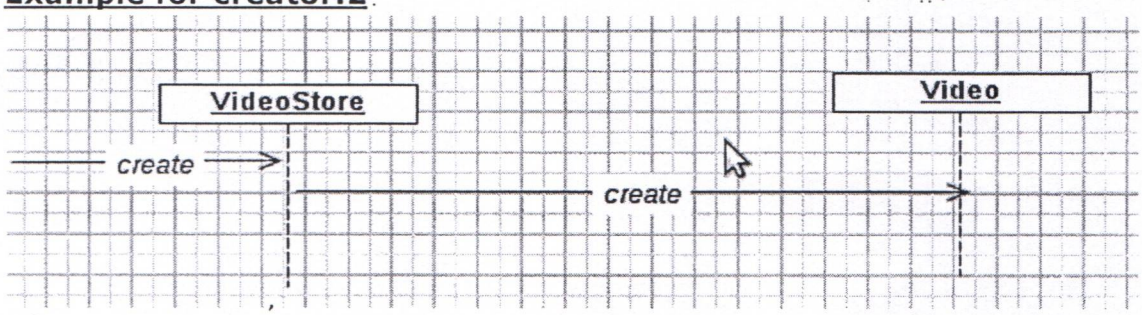
~~Information expert~~

- VideoStore has an aggregation associationwith Video. I.e, VideoStore is the container andthe Video is the contained object.
- So, we can instantiate video object inVideoStore class

Example diagram



Example for creator:2



High cohesion

High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of low coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system. Alternatively, low cohesion is a situation in which a given element has too many unrelated responsibilities. Elements with low cohesion often suffer from being hard to comprehend, hard to reuse, hard to maintain and averse to change.

Benefits

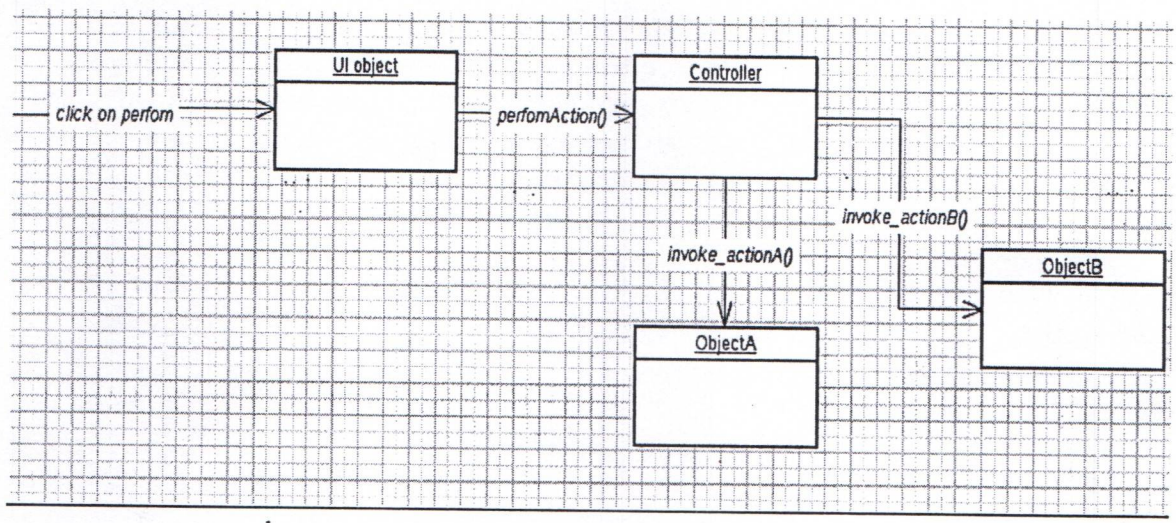
- Easily understandable and maintainable.
- Code reuse

- Low coupling

Example for low cohesion

It is defined as the first object beyond the UI layer that receives and coordinates ("controls") a system operation. The controller should delegate the work that needs to be done to other objects; it coordinates or controls the activity. It should not do much work itself. The GRASP Controller can be thought of as being a part of the application/service layer.(assuming that the application has made an explicit distinction between the application/service layer and the domain layer) in an object-oriented system with common layers in an information system logical architecture.

Example for Controller



Creator

Creation of objects is one of the most common activities in an object-oriented system. Which class is responsible for creating objects is a fundamental property of the relationship between objects of particular classes.

In general, a class B should be responsible for creating instances of class A if one, or preferably more, of the following apply:

- Instances of B contain or compositely aggregate instances of A
- Instances of B record instances of A
- Instances of B closely use instances of A
- Instances of B have the initializing information for instances of A and pass it on creation.

Example for Creator

- Consider VideoStore and Video in that store.

UNIT-IV GRASP

General responsibility assignment software patterns (or **principles**), abbreviated **GRASP**, consist of guidelines for assigning responsibility to classes and objects in object-oriented design.

The different patterns and principles used in GRASP are: controller, creator, indirection, information expert, high cohesion, low coupling, polymorphism, protected variations, and pure fabrication. All these patterns answer some software problem, and these problems are common to almost every software development project. These techniques have not been invented to create new ways of working, but to better document and standardize old, tried-and-tested programming principles in object-oriented design.

Computer scientist Craig Larman states that "the critical design tool for software development is a mind well educated in design principles. It is not UML or any other technology. Thus, GRASP are really a mental toolset, a learning aid to help in the design of object-oriented software.

Responsibility:

- Responsibility can be:
- accomplished by a single object.
- or a group of object collaboratively accomplish a responsibility.
- GRASP helps us in deciding which responsibility should be assigned to which object/class.
- Identify the objects and responsibilities from the problem domain,and also identify how objects interact with each other.
- Define blue print for those objects – i.e. class with methods implementing those responsibilities.

Patterns

Controller

The **controller** pattern assigns the responsibility of dealing with system events to a non-UI class that represents the overall system or a use case scenario. A controller object is a non-user interface object responsible for receiving or handling a system event.

A use case controller should be used to deal with *all* system events of a use case, and may be used for more than one use case (for instance, for use cases *Create User* and *Delete User*, one can have a single *UserController*, instead of two separate use case controllers).

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

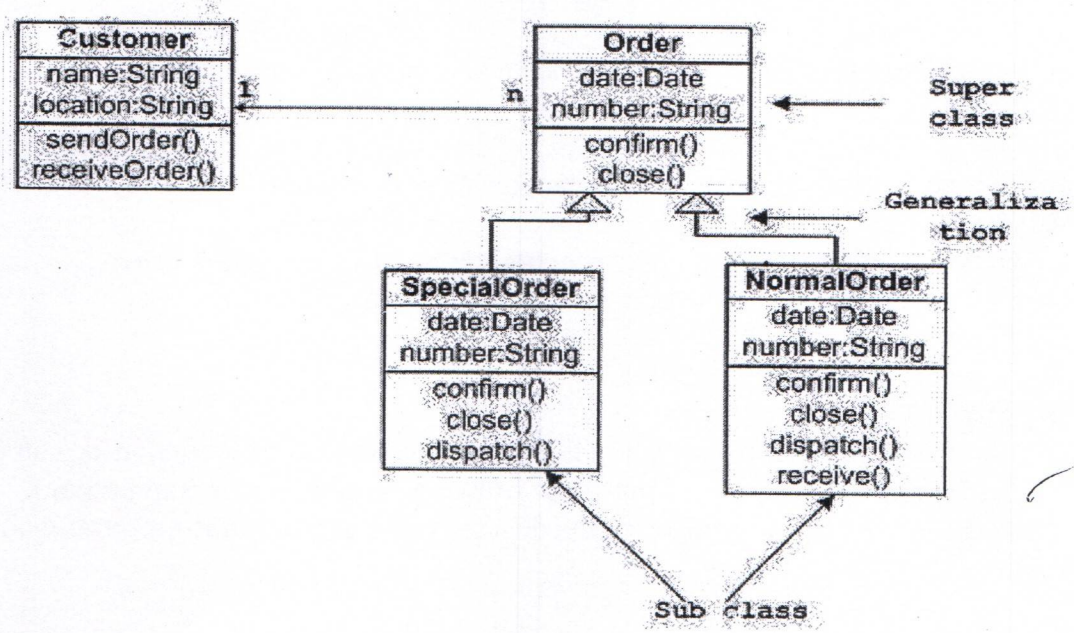
The following points should be remembered while drawing a class diagram –

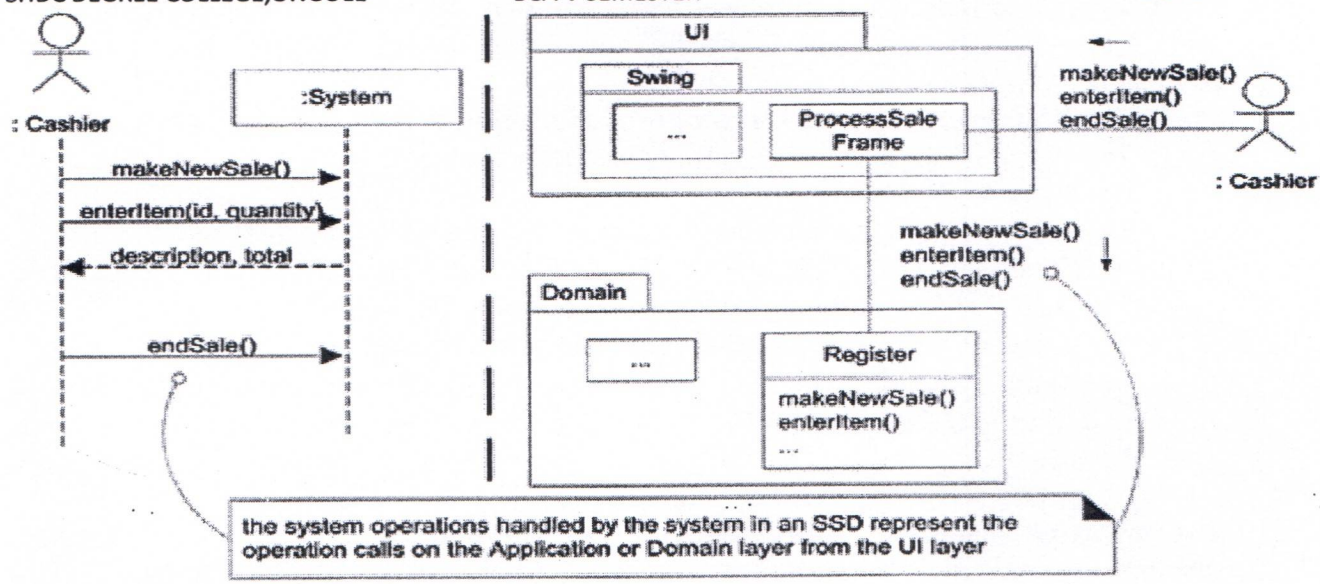
1. The name of the class diagram should be meaningful to describe the aspect of the system.
2. Each element and their relationships should be identified in advance.
3. Responsibility (attributes and methods) of each class should be clearly identified
4. For each class, minimum number of properties should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

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

1. First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
2. Order class is an abstract class and it has two concrete classes (inheritance relationship) Special Order and Normal Order.
3. The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

Example:-





UML - Class Diagram

The class diagram class allows you to add, retrieve and delete classes and categories to and from a class diagram. The class diagram class has a set of properties and methods that apply specifically to class diagrams. In addition, it inherits all diagram class properties and methods. Class diagram is a static diagram that's why it represents the static view of an application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

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

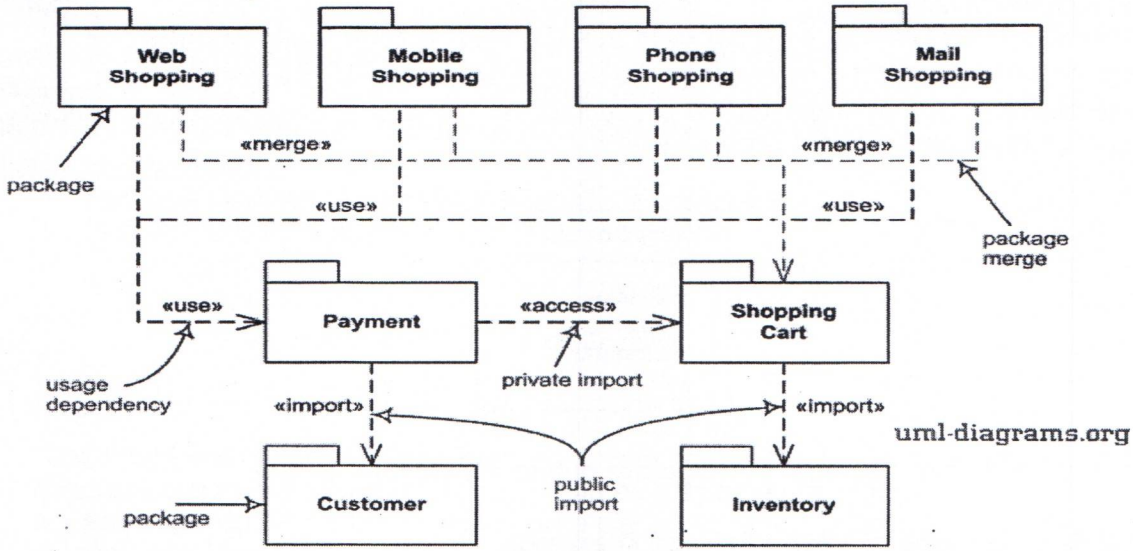
The purpose of the class diagram can be summarized as –

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

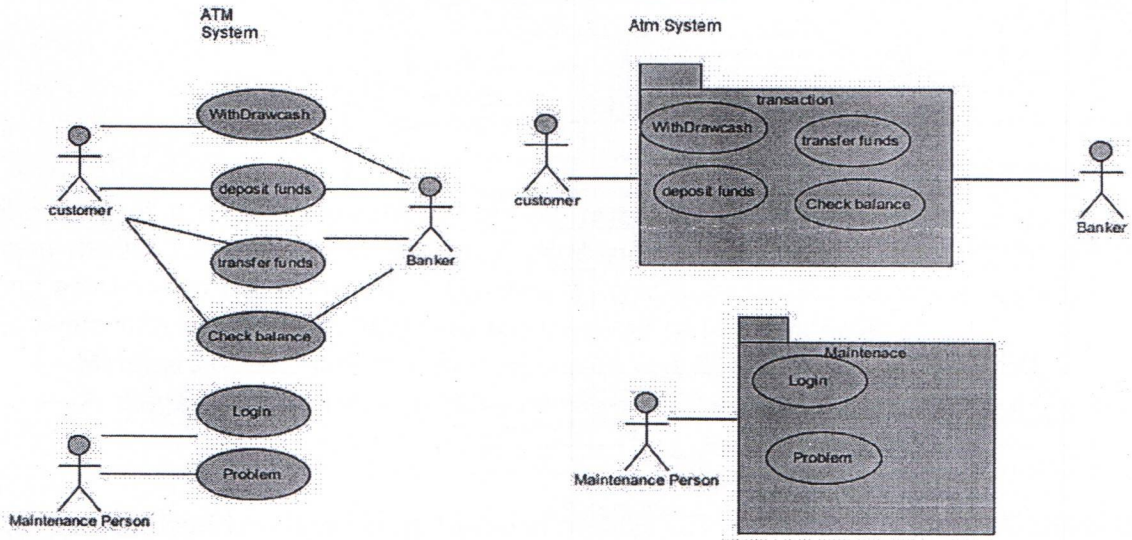
How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.



Relating between Use case and Package model:-



Use case Diagram ➔➔ Use case Package Diagram

Relating between Sequence and Package model:-

UNIT-III

SYSTEM SEQUENCE DIAGRAMS

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence and what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. This type of diagram is best used during early analysis phases in design because they are simple and easy to understand. Sequence diagrams are normally associated with use cases.

They're also called event diagrams. A sequence diagram is a good way to visualize (picture) and validate (certify) various runtime scenarios.

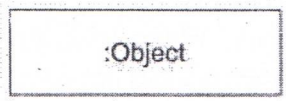
Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction. There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other.

A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

Basic Sequence Diagram Notations

Class Roles or Participants

Class roles describe the way an object will perform in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



component

Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

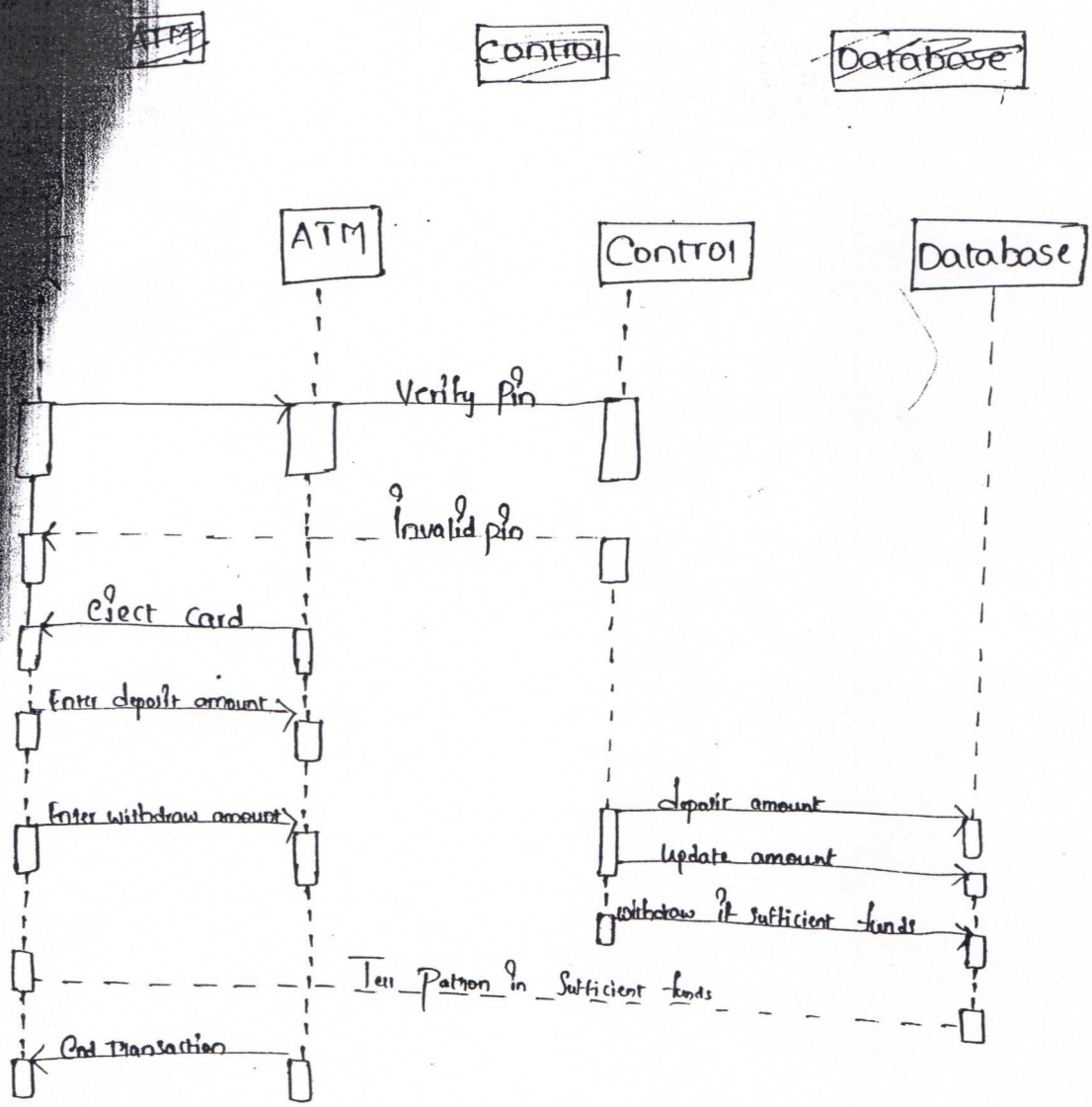
Activation or Execution Occurrence

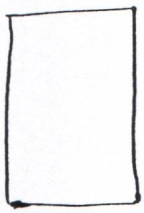
Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.

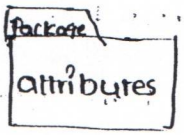
→(If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response.)

allows patrons or users or customers to access bank acounts through a Completely automated. The following diagram shows the Sequence for order of interactions in the ATM System's-



2)  Activation Box — It represents the time needed for an object to complete a task.

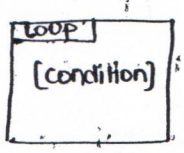
3)  Actor — It shows entities that interact with (or) external to the system.

4) 

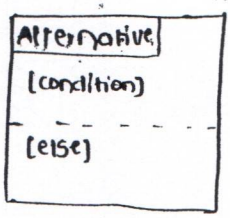
Package — It contains all interactive elements and attributes of the diagram.

5) 

lifeline — It represents passage of time as it extends down-wards

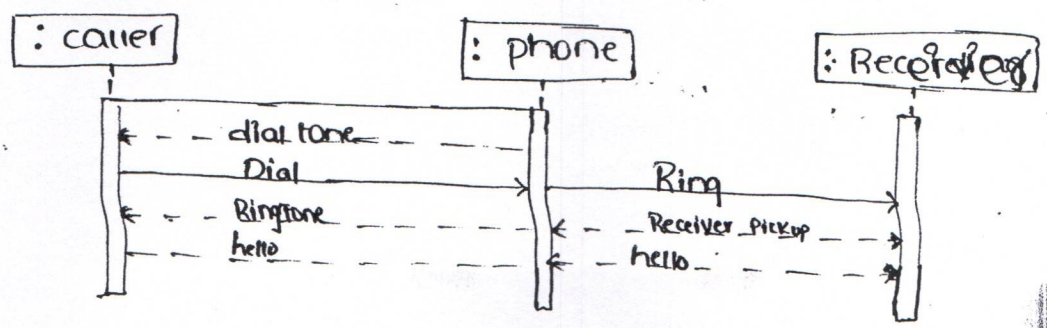
6) 

option, Loop — used to model if / then Scenarios i.e; A circumstance that will only occur under certain condition

7) 

Alternative — A choice b/w two or more message Sequences.

Sequence diagram for a phone call

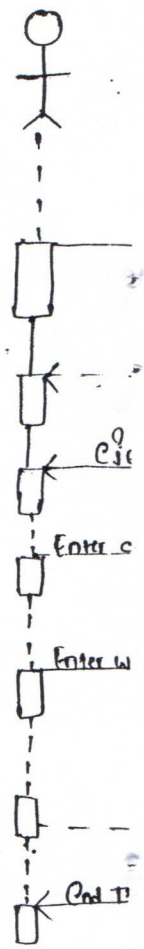


AT

their

Process

diagrar



Unit-3

ce Diagrams

nteraction Diagrams –

* Interaction diagrams are used to describe dynamic behaviour of the system.
* It is also used to describe interaction among the different objects or elements in the model.
* The interactive behaviour is represented in UML by two diagrams. They are
  i) Sequence Diagrams (Event Diagram (or) Event scenario)
  ii) Collaboration Diagrams

* The purpose of these two diagrams are similar.
* Sequence diagram emphasizes on time sequence of messages and colaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

→ Basic Symbols and components used in sequence Diagrams

~~i) Symbol~~
~~ii) Name~~ ~~object~~
~~iii) Description~~

Symbol	Name	Discription
	object	

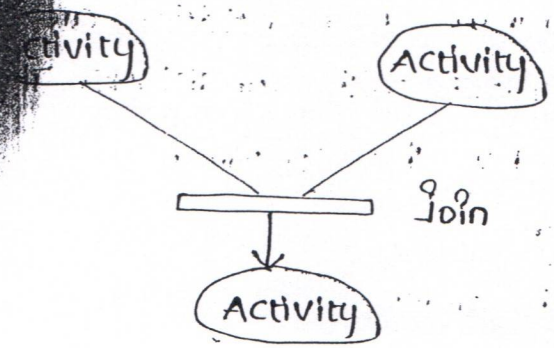
In the above diagram four activities are identified which are associated with conditions. They are

i) Send order by the customer
ii) Receipt order
iii) conform the order
iv) Dispatch the order

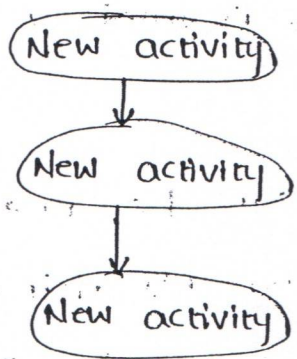
After receiving the order request, a condition is checked if it is normal (or) Special Order. After the type of order is identified, dispatch activity is performed and i.e, marked as the termination of the process.

→ Activity diagrams can be used for

* Modelling workflow by using activities
* Modelling Business requirements
* High level understanding of the System's functionalities
* Investing business requirements at a later states.



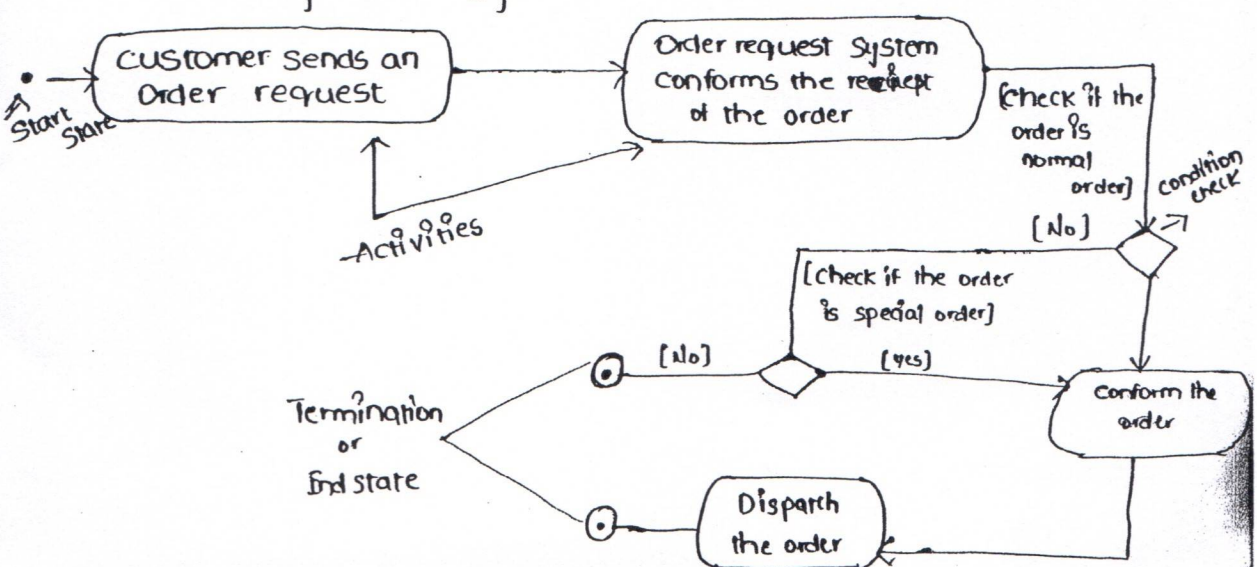
→ Transission – A transission connects multiple activitie represents the flow of control from one activity another activity.



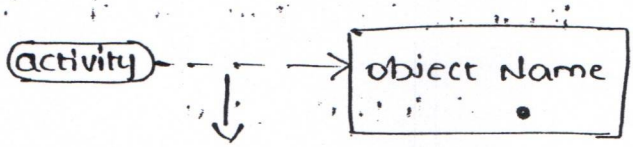
→ End state – It is used to show the end of the process. The representation as follows.

Ex: Activity Diagram

The following is an example of activity diagram for order management System.



f) Object Flow — It specifies the connection among activity and object. An object flow is a kind of dependency relationship.



object flow

g) Swim lane — A swim lane is a business workflow that separates the multiple activities in the activity diagram. An activity diagram may have any number of swim lanes.

h) Synchronization — A synchronization is a flow of control among the activities. Each Synchroniza may have two properties.

i) fork()
ii) join()

i) fork() — It is one of the horizontal Synchroni It means flow of control can be divided in to multiple work flows. Each fork is a division of single flow of control in to two or more work flows.

ii) join() - Join is one of the horizontal Synchr It is used to combine multiple activities in to single activity

nsitions

State / Final State :

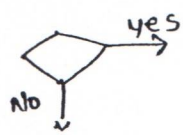
State – It is defined to show the start activity diagram (or) start of the process.

• → Intial State

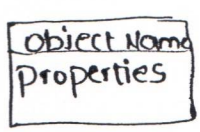
vity – An activity represents operation of a (or) Execution of a task (or) Execution of a ate.

Activity Name

Decisions – A decision represents work flow. Work flow divided in to several posibilities based on condition. A decision may have inflow and out flow of events.



d) object – An object is a instance (or) Blue print of the class. An object contains properties and operations which are associated with activity diagram.



e) State :

— Activity Diagrams – Activity diagram is another important diagram in UML to describe dynamic aspect of the System.

Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the System.

→ Purpose – Activity is a particular operation of the System. Activity diagrams are not only used for visualizing dynamic nature of the System but they are also used to construct the executable System.

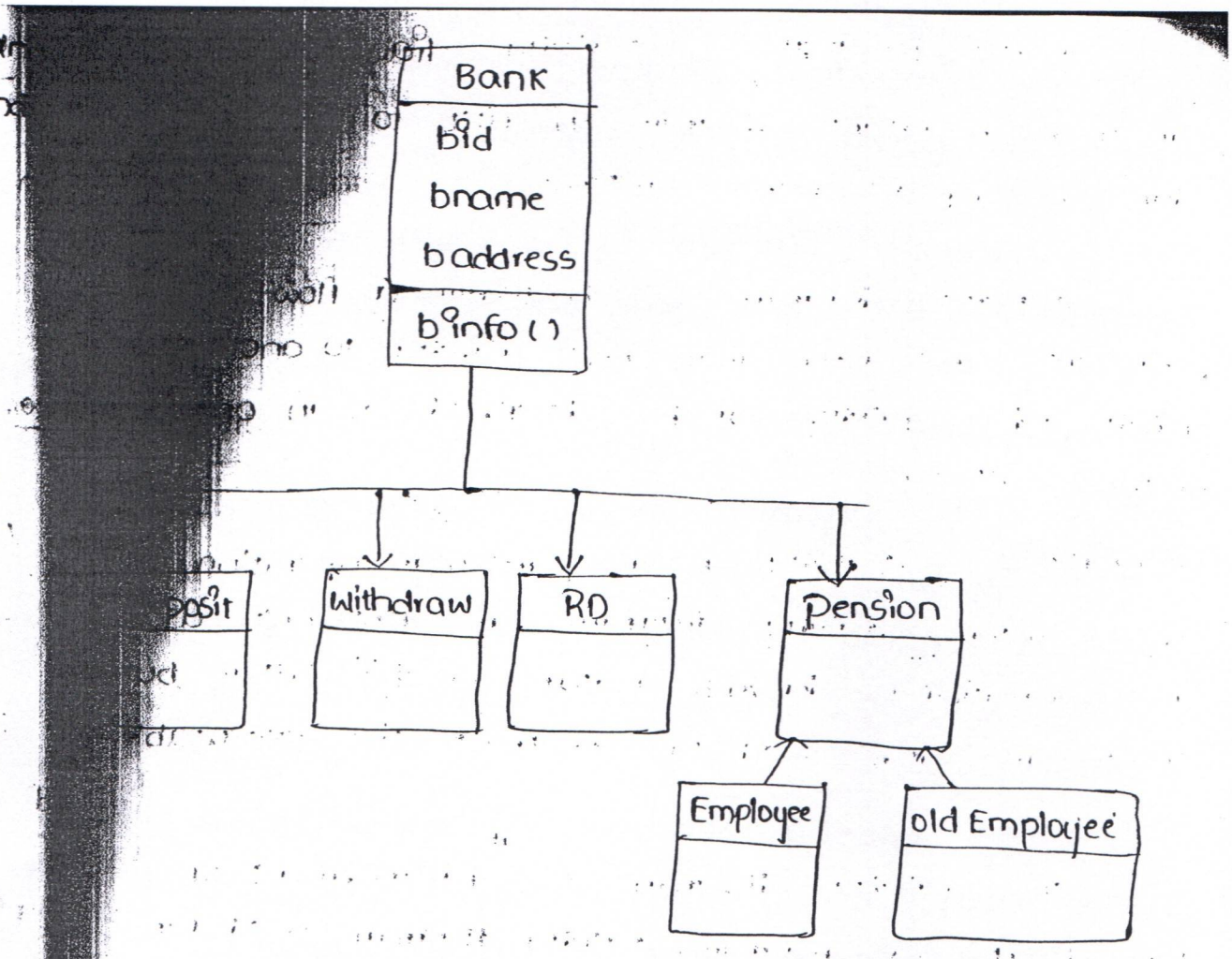
An activity diagram is an enhancement of flowchart / er diagram. Activity diagram used for special case of state machine diagram.

The main difference b/w activity diagram and State machine diagram are

* Activity diagrams are activity centry while state machine diagrams are state centry.
* Activity diagram is used for modelling the sequence of activities of the given process. where as state machine diagram is used to stages of object life line.

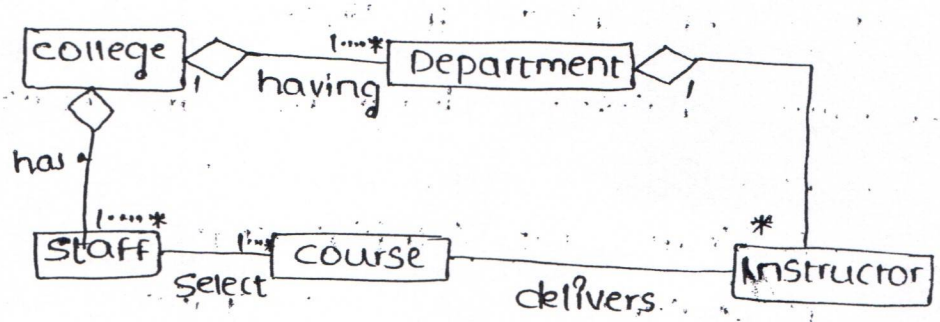
Generally activity diagrams are having following components.

a) Start State / Initial State
b) Activity
c) Decision
d) object
e) States
f) object flow
g) Swim line
h) Syncranization

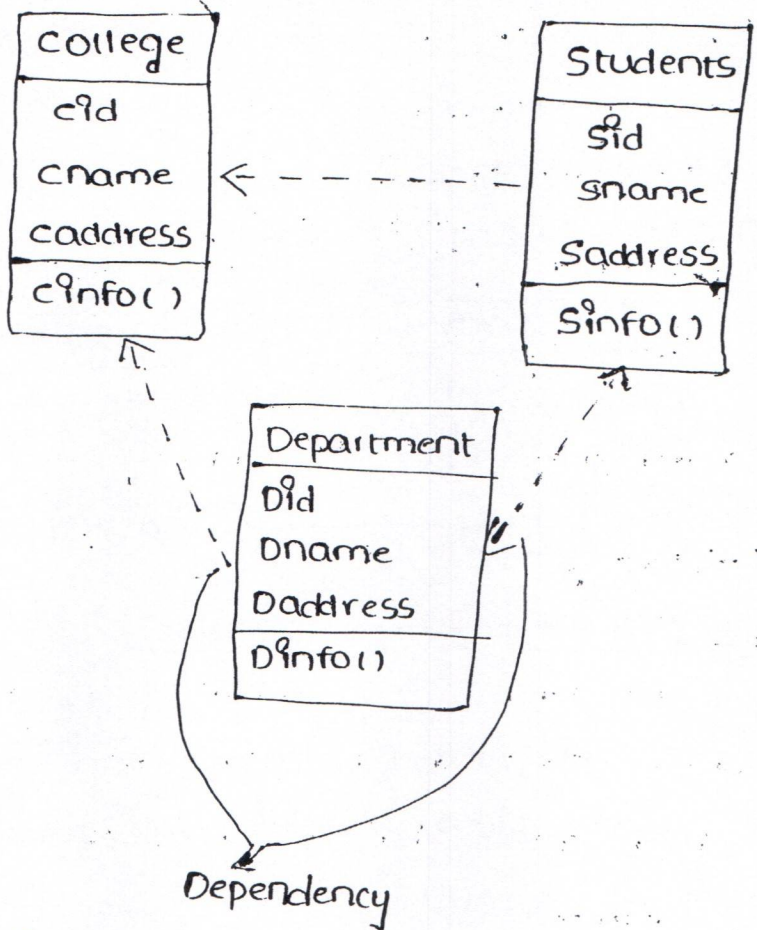


c) Modelling Structural relationships - To model Structural relationships follow the steps

* Apply associations among the set of classes in the system (/) model.
* Specify association among the classes like association, name, association role, multiplicity, Aggregation



with its properties like attributes, operations and responsibility and, the correct set of class using dependen relation.



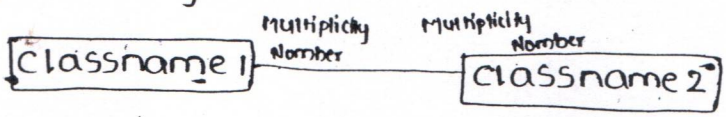
b) Modelling Single Inheritance — To model the Single Inheritance,

* Identify number of classes and specify Super class and its sub classes
* If any common properties are shared by the classes then Seperate them in to one group.
* Apply generalization relationship among the classes

about
class A
f parts.
ation then
me.
ation name
2
lame
then

plicity allows how many objects Participat
onship. There are 3 types of multiplicitie
ne to one (1....1)
one-to many (1.... *)
many to many (*.... *)

Eax:



Ex!

b) Ex!

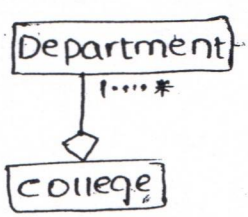
c) Ex!

Agreegation –

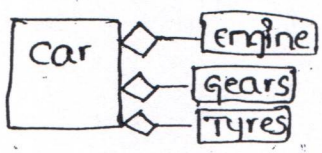
* It is one kind of association relationship in blw the whole class and its sub parts.

* It is represented by a solid line along with hallow diamond.

Ex! ①



②



→ Common modelling techniques for class diagrams

Relationship supports 3 types of common modelling techniques.

a) Modelling simple dependency
b) Modelling single Inheritance
c) Modelling structural relationships.

a) Modelling simple dependencies – To model simple dependency among classes, identify number of classes within the system (/) model along

Syntax: 

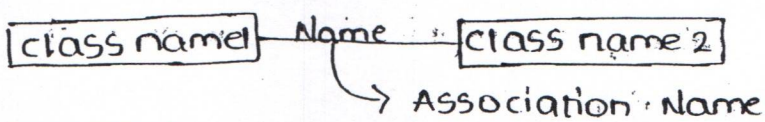
In the above Syntax, class A knows about class B and class B knows about class A

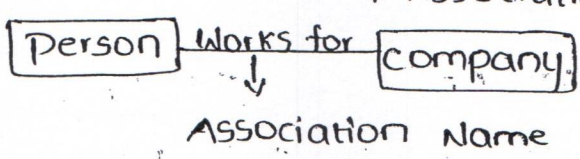
Ex! 

* Association can have four types of parts.
  a) Association Name
  b) Association Role
  c) Multiplicity
  d) Agreegation.

a) Association Name -

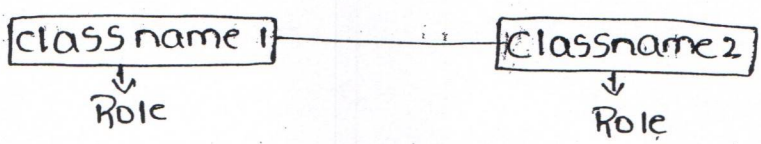
* A class may participate in association then represent each class with specific name.
* Every association can have association name

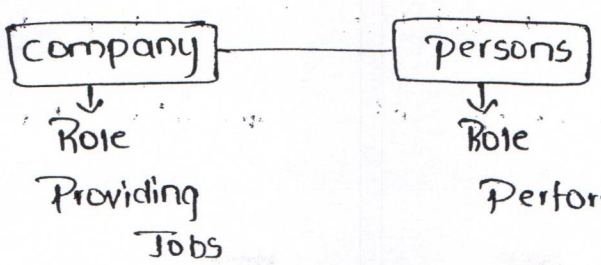
Syntax! 

Ex! 

b) Association Role -

* A class may Participate in association then each class can have specific role.

Syntax: 

Ex! 

delling – It is the activity of translating

of relat... ...nal description of a domain into a ...odel.

on domain

ASSOCIATION

conceptua... ...sociation shows relationship b/w classes.

(or)

...ntation of ...sociation is a structural kind of relationship problem classes.

* It is represented by a solid line. '____'
* In general there are two types of assiosciations
  a) uni directional association
  b) Bi directional association

a) uni directional association – In this type of association, communication among the classes in one direction.

Syntax: 

* class A knows about class B
* class B knows nothing about class A

Ex: 

In the above example person knows its address Address does not know who is living there

b) Bi directional association – In this type of association, communication among classes in two directions

DOMAIN MODEL

→ Domain - A domain is a collection of related concepts, relationships & work flows.

Ex: Govt domain, Business domain, Education domain

→ Domain model - It is also known as conceptual model (or) domain object model.

* Definition - A domain model is a representation of the concepts or objects appearing in the problem domain. It also captures the relationship among those objects.

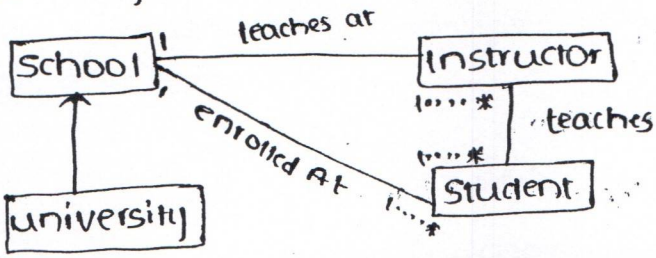
Ex: Book, - - - -

(Or)

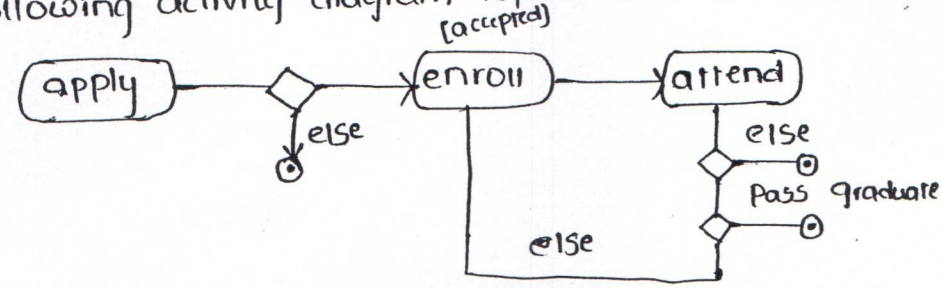
A domain model is a package containing a class diagrams and activity diagrams.

Ex: Education domain can be modeled by a package. This package contains class diagrams which models the concepts of school, university, instructor and student as well as relationships.

The following class diagrams represents education mod



The following activity diagram represents a student work flow



Unit-2

The primary purpose of this phase complete the most essential part of oject that are high risks and plan the ruction phase.

is the part of the project where technical s are fully evaluated and los eliminated by ilding the highest risk of the Project.

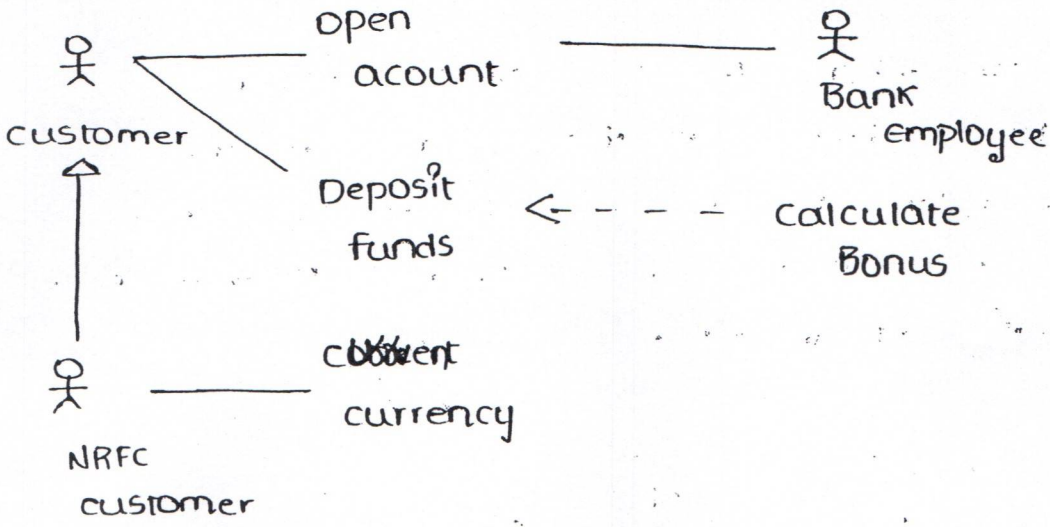
During this phase estimate man powers to complete this project.

The complete cost and time of the project is determined.

* Sum of the usecases will help identify the risk
* complete project plan with construction iteration
* The Project domain model is defined.

→ Requirement to be completed for this phase

* Description of the S/w architecture, we can use activity diagrams, Sequence diagrams and domain models should be used.
* Complete project plan
* complete development plan.



→ Include relationship b/w 2 usecases –

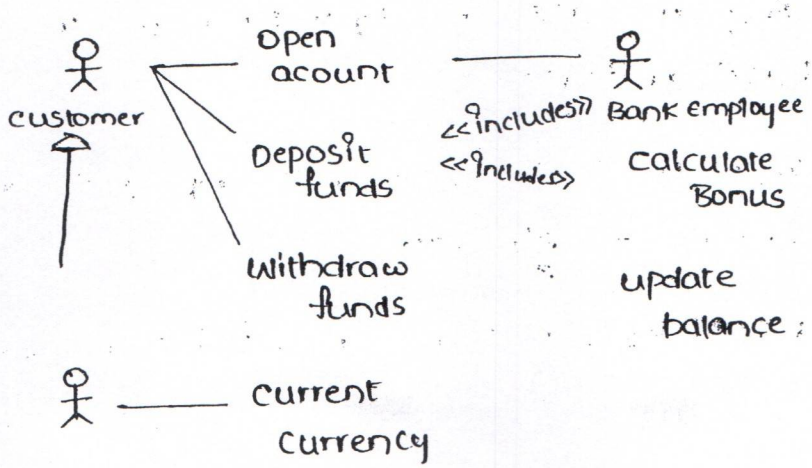
* Include relationship show that the behaviour of the included usecase is part of the including usecase.

* The main reason for this is to reuse the common actions across multiple usecases. Few things to remember when using the include relationship.

<<include>>

* The base usecase is in-complete without the included usecase.

* The included usecase is mandatory and not optional.



→Elabor

is
the
cons

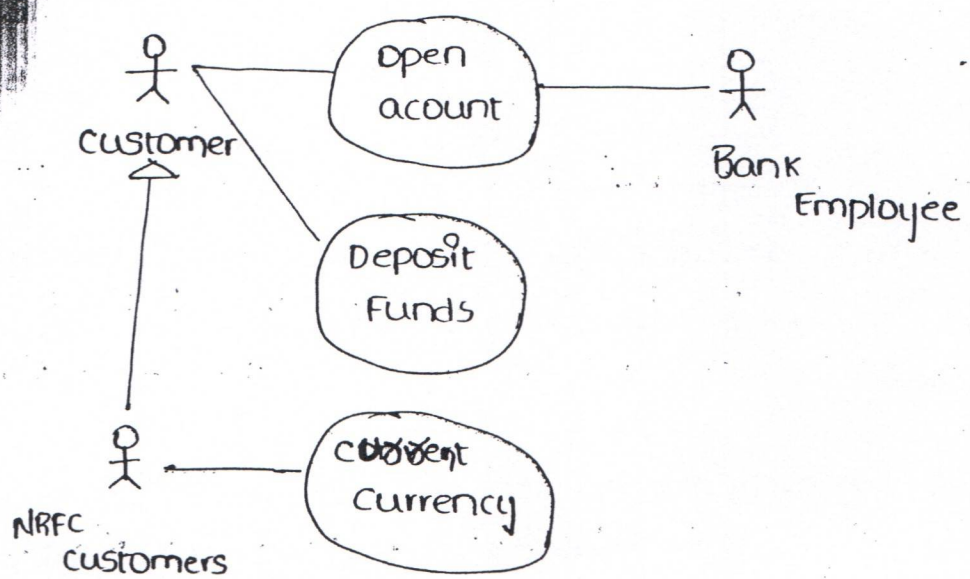
* Th
risks
build

* Dur
compl

* The
deter
Sum
com
re
Req

of an actor - It means that one

inherit the role of the other actor. The
inherits all the usecases of the
ndent has one or more usecases that are
to that role. The following diagram represents
ation of an actor.



→ Extending relationship b|w 2 usecases -

* As the name implies it extends the base usecase and adds more functionality to the system.

* Few things to consider when using extend relation-ship <<Extend>>

* The Extending usecase is dependend on the extended (base) usecase.

* The Extending usecase is usually optional and can be triggend conditionally.

* The Extended (base) usecase must be meaningful on its own

1) Include

2) Expand

3) Generalization

* By using above 3 relationships, a user creates 4 types of relationships.

1) Assioscation b|w actor and usercase

2) Generalization of an actor

3) Extend relationship b|w two usecases.

4) Include relationship b|w two usecases.

→ Assioscation b|w actor and usercase -

* This is one straight forward relationship used in usecase diagrams.

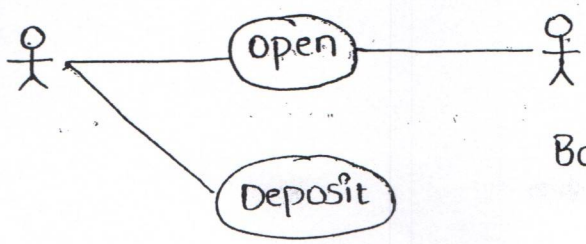
* We can follow the following rules assioscation ship b|w actor & usecase.

* • An actor must be assioscated with atleast one usecase.

• An actor can be assioscated with multiple usecases.

• Multiple actors can be assioscated with a single usecase.

The following diagram represents different ways assioscation relationship in the usecase diagrams.



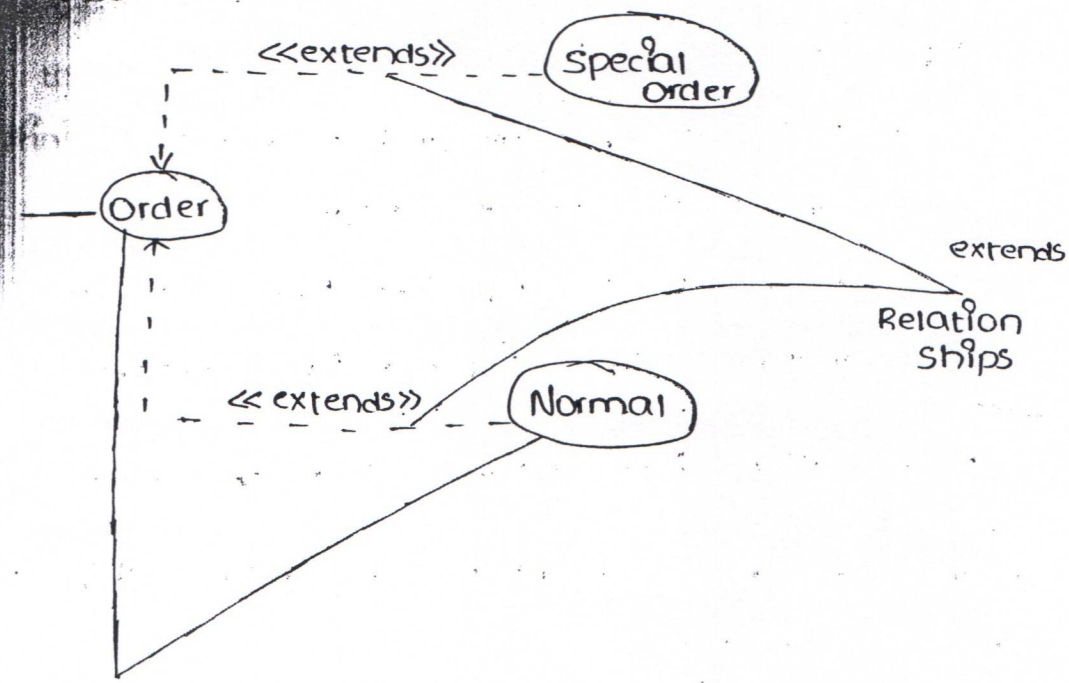
→ Ger

actor
desenc
The c
Specif
generali

→ Exte

* As

diagram of an order management System



In the above diagram we have 3 usecases med order, Special order and normal order.

In the above diagram we have only actor called Customer.

The normal order and Special order usecases are tended from order usecase.

The actor customer lies outside the System as it is external user of the System.

Usecase diagrams used for

Requirement analyasis and high level design.

Model the context of a System.

To draw usecase diagram, we use 3 types of relationships. They are

functionality of a system. Hence to model of a system we can use number of diagrams

— Purpose of usecase diagrams — The following are purpose

* used to gather dynamic requirements of a system.
* used to get and outside view of the system.

— Drawing usecase diagrams — When we are planning to draw a usecase diagram we should have the following the rules.

* Functionalities to be represented as usecases.
* Actors.
* Relationships among the usecases & Actors.

After identify the above items, Follow the steps to draw an efficient usecase diagrams.

* The name of the usecase is very important. The name should be choosen in such a way so that it can identify the functionalities performed
* Give a Suitable name for users.
* Show relationships, clearly in the diagram.

The following diagram is a usecase diagram representing the order management diagram

Case Modelling —

* To model a system the most important aspect is to capture the dynamic behaviour. Dynamic behaviour means the behaviour of the system when it is running.

* There are two types of behaviours.
  - Static behaviour
  - Dynamic behaviour

— Static behaviour is system behaviour in stable state.

— Dynamic behaviour is system behaviour in varying situations (or) running.

* only static behaviour is not sufficient to model a system rather than dynamic behaviour is more important than static behaviour.

* In UML, they are 5 diagrams available to model the dynamic behaviour of a system. usecase diagram is one of them.

* usecase diagram shows a set of usecases and actors and there relationships.

* usecase diagram consists of actors, usecase and there relationships.

* A usecase diagram is used to model the sub-system of an application.

* A single usecase diagram captures a particular

Use

*

aspe

Dyna

Syst

*

—

s

—

Si

* C

a

Impl

* In

mod

diagr

* US

act

* US

ther

Us

A

* The project deem involves building the System iteratively and incrementally

- Transition —

* In this phase the SIW (or) project is released to the public.
* If there is any final adjustments (or) updates are made based on feedback from the end users (or) clients

→ Case Study : The Next Gen pos System — A pos System is used to record sales and handed payments. pos stands for point of Sale.

* It is typically used in retail Stores
* It includes HIW components such as computer and bar code reader, and SIW to run the System.
* These Systems must be relatively fault tolerant that is even if remote services are temporary unavilable they must still capable of capturing Sales and handling atleast cash payments.

→ Inception —

- Inception —

* The primary goal of the Inception phase is to develop idea for the project how to start.

* Define scope of the System
* Identifying critical risks and determining when and how the project will adress.
* Estimation of cost, Effer and product quality
* It determines what resources will be needed.
* It uses class diagrams and package diagrams to gather requirements..

- Elaboration —

* The primary goal of the elaboration phase to build the ~~dete~~ new System based on requirements.
* The project architecture and required resources are further evaluated.
* Developers consider possible applications of the S|W and cost assioscated with the development.
* The UML diagrams used during this phase are activity diagram, Sequence diagram, Colaboration, State transition and Interaction diagrams.

- Construction —

* In this phase the project is developed and completed.
* The S|W is designed and tested.

Process phases — unified process is nothing "SLW development process." A SLW development process is the set of activities needed to transform a users requirement into a SLW System".

The unified process is a SLW development process i.e., usecase driven, architecture centry and iterative and incremental.

→ usecase driven —

* usecases captured requirement of the user
* usecases are used to verify the correctness of the implemented SLW.
* usecases divides the development project into Smaller Subprojects.

→ Architecture centry —

* Find the Structures which are suitable to achieve the functions specified in the usecases.
* Find the Structure which are understandable
* Find the Structure which are reusable for later expensions.

→ Iterative and Incremental — Develop the project/ SLW using iterative and incremental model.

unified process divides the developed process into four phases. They are — Inception
— Elaboration
— construction
— Transition

— Generalization — It can be defined as a relationship which connect a specalized element with a generalised element. It basically describes the inheritance relationship.

Symbol — 

— Realization — It can be defined as a relationship in which two elements are connected. one element describes some responsbility, which is not implemented and the other one is implements them. This relationship is exists in case of interfaces.

Symbol — 

—UML Diagrams — UML Diagrams are output of the given System are application/ SW. All the elements (Things), R.ships are used to complete UML diagrams and these diagrams represents a System. UML includes 9 diagrams. They are

- Class Diagram
- object "
- usecase "
- Sequence "
- collaboration "
- Activity "
- State chart "
- Diployment "
- Component "

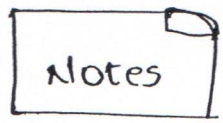
group
to
tion as

ate of
defines
hrough
machine

There is
ckage
ural
Package

as
l
only
odes

- Notes – It is the only one A.T. used to write comments of UML element. The Symbol of Notes as follows



UML Building Blocks

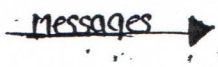
— Relationships: – It is another important building blocks of UML it shows how the elements are associated with each. There are 4 kinds of relationships available there. They are

- Dependency
- Assiosleation
- Generalization
- Realization

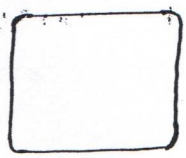
– Dependency – Dependency is a relationship b|w two things in which change in one element also effects the other. The Symbol of dependency as shown below

— Assiosleation — Assiosiation is basically a set of things that connects a elements of UML model. It also describes how many objects are taking part in the relationship. Symbol of assiosciation as follows

- Interaction – Interaction is defined as group of messages exchanged among elements to accomplish a talk. The Symbol of Interaction as shown below.



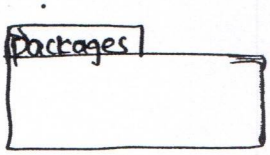
- State Machine – It is useful when the state of object in its life cycle is important. It defines the sequence of states an object goes through in response to even. The Symbol of state machine as follows



~~Diagram~~

→ Grouping Things – It can be defined as mechanism to group elements of UML model. There is only one grouping thing available i.e, Package

- Package – It is used to gathering Structural and behavioural things. The Symbol of packa as follows

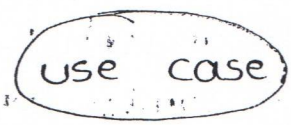


→ Annotational Things – It can be defined as a mechanism to write descriptions and comments of UML model elements there is only one annotational model is available i.e, Nodes

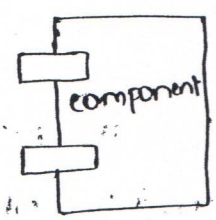
collaboration - It defines an interaction blw elements or components or elements. The symbol of collaboration is called collaboration. Symbol as follows:

Symbol of collaboration — ⊏

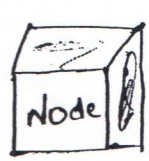
d) use case - use case represents set of actions performed by a system for a specific goal. The representation of use case as follows



e) Component - It describes the physcial part of a system. The representation of Component symbol as follows



f) Node - A node can be define as a physcial element that exists at run time. Node represented as follows



→ Behavioural Things - It consists of dynamic part of the UML model. There are two types of behavioural things they are

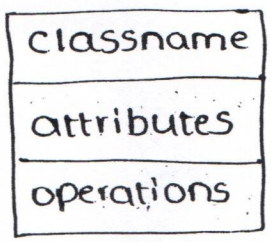
- Interaction
- State machine

→ Structural thing
→ Behaviour thing
→ Grouping thing
→ Annotational thing

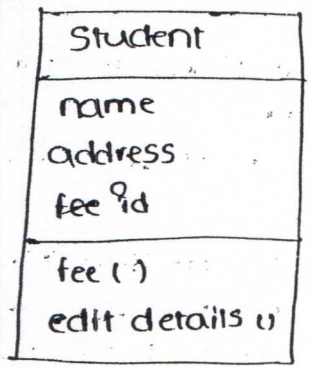
→ Structural Things — It defines the static part of the model. The following are structural things

a) class b) Interface c) collaboration d) use case
e) component f) Node

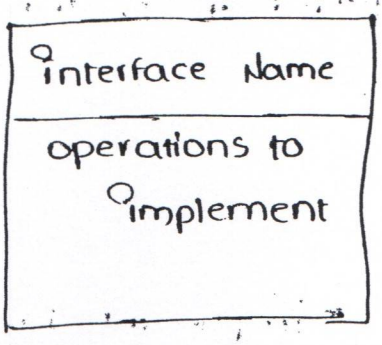
a) class - class is a set of objects having common responsibilities. A class represents as follows.



Ex: Representation of student class



b) Interface — It defines set of operations, which specify the responsibility of a class



c) col
Sy
col
Sym

d) us
Perf
repr

e) Con
a
Syn

f) Node
eler
as

→ B
Part
beha

→ Constructing — It means construct a model with identified requirements.

→ Documenting — Prepare a template which consist of results of the SIW System.

→ UML is a pictoral language used to may SIW blue print.

• Applications of UML (or) uses of UML —

* Development of SIW Application
* preparation of blue prints for Application
* Banking and Retail Sector Application
* Web based Application
* Health care Application
* construction of business applications
* Enterprize application of organisation

• conceptual model for UML

* The conceptual model for uml is construction of the system in effective manner.
* The conceptual model of uml can be divided into three major elements
  - UML Building Blocks
  - Rules to connect the building blocks
  - common mechanism of UML

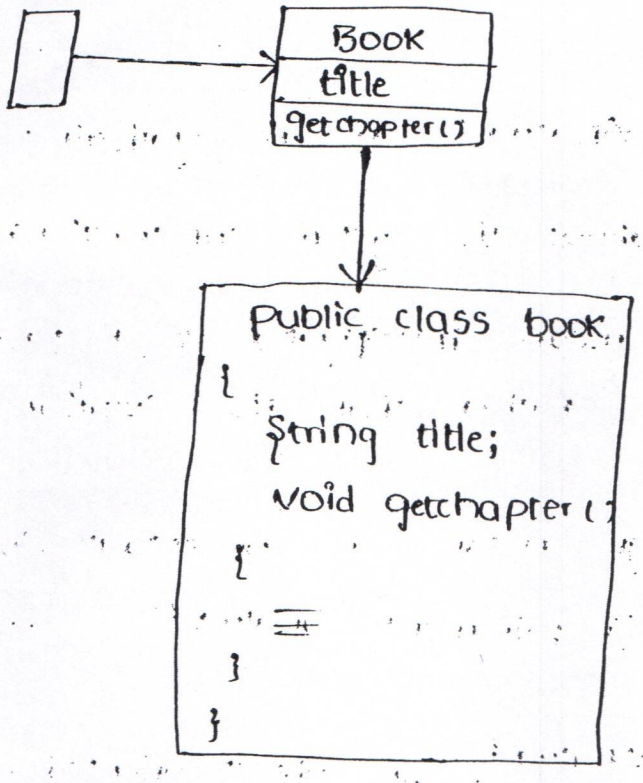
UML Building Blocks - It gives the information to how to construct the system (or)

UML Building Blocks can be divided in to three blocks

→ Things
→ Relationships
→ Diagrams

→ Things — Things are the most important building blocks of the UML diagrams. It gives basic components of the given model. It classified in to 4 types.

Domain object



Representation of book class in object oriented language.

→ UML - Stands for unifying modelling language.

- UML is Standard language for specifiying, visualizing, constructing and documenting the artificats of SIW System.
- UML is not a programming language but used to generate code in various languages using UML Diagrams.

→ Visualising - It means process of identifiying System requirements.

→ Specifiying - It means design the identifiying fine System requirements.

→ Cor
Ider
→ Doc
of
→ UML
Print
• Applica
* Devel
* Prepa
* Banki
* Web
* Health
* Constr
* Enterp
• concept
* The c
effective
* The
major
UML
how
UML
blocks
→
→
→
→ Thin
block
comp

Unit-1

Introduction

Analyasis - Analyasis means gathering the requirements about the System construction(or)

Analyasis means investigation of a problem and requirements rather than its solution

Design - Design means a conceptual solution that fullfill the requirements analyasis

object oriented Analyasis -- In object oriented analyasis, there is an emphasis on finding and describing the objects in the problem domain.

Ex: In case of library information System, Sum of the objects include book and patron

object oriented design - In object oriented design, there is an emphasis on defining software objects and how they are colebarate to fullfill the requirements.

Ex: In library system, a book, SW object may have title attribute and getchapter method.

During the implementation of book object by using object oriented programming, design objects are implemented as shown below.