UNIT I

UML DIAGRAMS

INTRODUCTION TO OOAD

Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled, and is characterised by its class, its state (data elements), and its behavior. Various models can be created to show the static structure, dynamic behavior, and run-time deployment of these collaborating objects. There are a number of different notations for representing these models, such as the Unified Modeling Language (UML).

Object-oriented analysis (OOA) applies object-modelling techniques to analyze the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. OOA focuses on *what* the system does, OOD on *how* the system does it.

Object-oriented systems

An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves them sending messages to each other. Sending a message differs from calling a function in that when a target object receives a message, it itself decides what function to carry out to service that message. The same message may be implemented by many different functions, the one selected depending on the state of the target object. The implementation of "message sending" varies depending on the architecture of the system being modeled, and the location of the objects being communicated with.

Object-oriented analysis

Object-oriented analysis (OOA) looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed. Analysis models do not consider any implementation constraints that might exist, such as concurrency, distribution, persistence, or how the system is to be built. Implementation constraints are dealt during object-oriented design (OOD). Analysis is done before the Design. The sources for the analysis can be a written requirements statement, a formal vision document, interviews with stakeholders or other interested parties. A system may be divided into multiple domains, representing different business, technological, or other areas of interest, each of which are analyzed separately.

The result of object-oriented analysis is a description of *what* the system is functionally required to do, in the form of a conceptual model. That will typically be presented as a set of use cases, one or more UML class diagrams, and a number of interaction diagrams. It may also include some kind of user interface mock-up. The purpose of object oriented analysis is to develop a model that describes computer software as it works to satisfy a set of customer defined requirements.

Object-oriented design

Object-oriented design (OOD) transforms the conceptual model produced in object-oriented analysis to take account of the constraints imposed by the chosen architecture and any non-functional – technological or environmental – constraints, such as transaction throughput,

response time, run-time platform, development environment, or programming language. The concepts in the analysis model are mapped onto implementation classes and interfaces. The result is a model of the solution domain, a detailed description of *how* the system is to be built.

UNIFIED PROCESS

The **Unified Software Development Process** or *Unified Process* is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP). Profile of a typical project showing the relative sizes of the four phases of the Unified Process.

Overview

The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The *Rational Unified Process* is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

The name *Unified Process* as opposed to *Rational Unified Process* is generally used to describe the generic process, including those elements which are common to most refinements. The *Unified Process* name is also used to avoid potential issues of trademark infringement since *Rational Unified Process* and *RUP* are trademarks of IBM. The first book to describe the process was titled *The Unified Software Development Process* and published in 1999 by Ivar Jacobson, Grady Booch and James Rumbaugh. Since then various authors unaffiliated with Rational Software have published books and articles using the name *Unified Process*, whereas authors affiliated with Rational Software have favored the name *Rational Unified Process*.

Unified Process Characteristics

Iterative and Incremental

The Unified Process is an iterative and incremental development process. The Elaboration, Construction and Transition phases are divided into a series of timeboxed iterations. (The Inception phase may also be divided into iterations for a large project.) Each iteration results in

increment, which is a release of the system that contains added or improved functionality compared with the previous release. Although most iterations will include work in most of the process disciplines (*e.g.* Requirements, Design, Implementation, Testing) the relative effort and emphasis will change over the course of the project.

Use Case Driven

In the Unified Process, use cases are used to capture the functional requirements and to define the contents of the iterations. Each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test and deployment.

Project Lifecycle

The Unified Process divides the project into four phases:

Inception

Elaboration

Construction

Transition

Inception Phase

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process. The following are typical goals for the Inception phase. Establish a justification or business case for the project Establish the project scope and boundary conditions Outline the use cases and key requirements that will drive the design tradeoffs Outline one or more candidate architectures Identify risks Prepare a preliminary project schedule and cost estimate

The Lifecycle Objective Milestone marks the end of the Inception phase.

Elaboration Phase

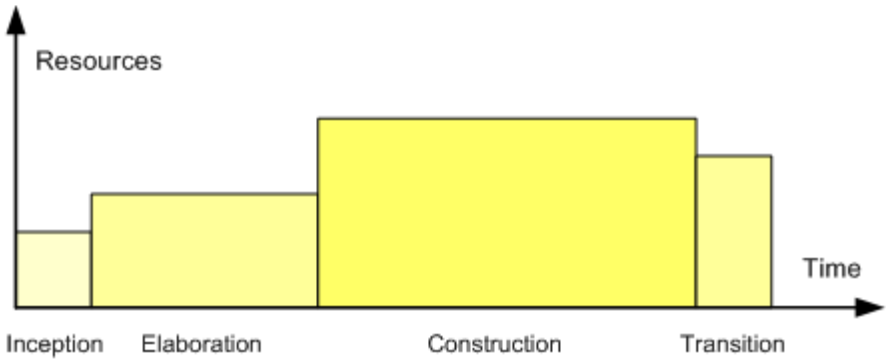
During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams). The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core, most architecturally significant, components. It is built in a series of small, timeboxed iterations. By the end of the Elaboration phase the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability and cost. The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.

Construction Phase

Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, time boxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. Common UML (Unified Modelling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams. The Initial Operational Capability Milestone marks the end of the Construction phase.

Transition Phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training. The Product Release Milestone marks the end of the Transition phase.



UML DIAGRAMS

What is UML?

Unified Modelling Language (UML) is the set of notations,models and diagrams used when developing object-oriented (OO) systems. UML is the industry standard OO visual modelling language. The latest version is UML 1.4 and was formed from the coming together of three leading software methodologists; Booch, Jacobson and Rumbaugh. UML allows the analyst ways of describing structure, behaviour of significant parts of system and their relationships.

Unified Modeling Language (**UML**) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software-intensive systems.

The Unified Modeling Language is commonly used to visualize and construct systems which are software intensive. Because software has become much more complex in recent years, developers are finding it more challenging to build complex applications within short time periods. Even when they do, these software applications are often filled with bugs, and it can take programmers weeks to find and fix them. This is time that has been wasted, since an approach could have been used which would have reduced the number of bugs before the application was completed. However, it should be emphasized that UML is not limited simply modeling software. It can also be used to build models for system engineering, business processes, and organization structures. A special language called Systems Modeling Language was designed to handle systems which were defined within UML 2.0. The Unified Modeling Language is important for a number of reasons. First, it has been used as a catalyst for the advancement of technologies which are model driven, and some of these include Model Driven Development and Model Driven Architecture. Because an emphasis has been placed on the importance of graphics notation, UML is proficient in meeting this demand, and it can be used to represent behaviors, classes, and aggregation. While software developers were forced to deal with more rudimentary issues in the past, languages like UML have now allowed them to focus on the structure and design of their software programs. It should also be noted that UML models can be transformed into various other representations, often without a great deal of effort.

USE CASE – CLASS DIAGRAMS– INTERACTION DIAGRAMS – STATE DIAGRAMS – ACTIVITY DIAGRAMS – PACKAGE, COMPONENT AND DEPLOYMENT DIAGRAMS.

UML DIAGRAMS

UML (Unified Modeling Language) is a std notation for the modeling of real-world objects as a first step in developing an object-oriented design methodology. UML is a visual language for specifying, constructing and documenting the artifacts of a s/m.

Artifacts: An artifact is the specification of a physical piece of information that is used or produced by a s/w development process, or by deployment and operation of a s/m

example of artifacts: model files, source files, scripts and binary executable files, a table in a database s/m, a development deliverable, or a word-processing document, a mail msg.

Various UML diagrams

* Use Case Diagram
* class diagram
* interaction diagram

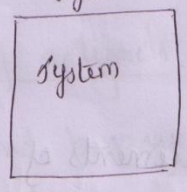
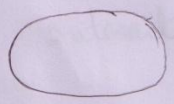
* State diagram
* Activity diagram
* Package diagram

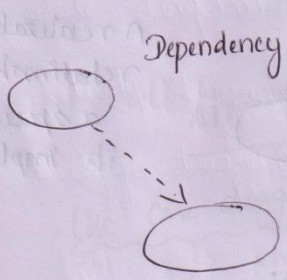
Use Case Diagram

Use case diagrams are used to describe a set of actions that some s/m or s/m's should or can perform in collaboration with one or more external users of the s/m (actors). Each use case should provide some observable and valuable results to the actors or other stakeholders of the s/m

Purpose.

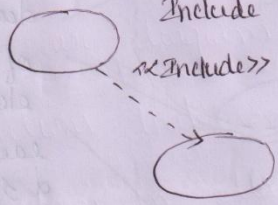
→ Used to gather requirements of a s/m
→ Used to get an outside view of a s/m
→ Identify external and internal factors influencing the s/m
→ Shows the interaction among the requirement through actor

S.No	Notations	Description
1)		Actors are the entries that interact with a system. Actors are used to represent the users of s/m, actors can actually be anything that needs to exchange information with the s/m. An actor may be people, computer hardware, other s/ms etc
2)		The scope of a system (s/m) can be represented by a system (shape) or sometimes known as a system boundary. The use case of the system are placed inside the system shape, while the actor who interact with the system are put outside the s/m.
3		Use case describes the action performed by user



Dependency

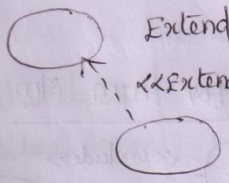
A dependency relationship represents that a model element relies on another model element for specification and/or implementation



Include

<<Include>>

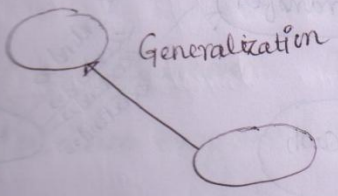
An Include relationship specifies how the behaviour for the inclusion use case is inserted into behaviour defined for base use case



Extend

<<Extend>>

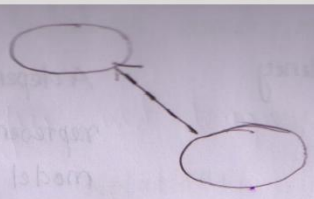
An Extend relationship specifies how the behaviour of extension use case can be inserted into the behaviour defined for the base use case



Generalization

A generalization relationship is used to represent inheritance relationship b/w model elements of same type. The more specific model element share the same specification with the more general the model element but more details

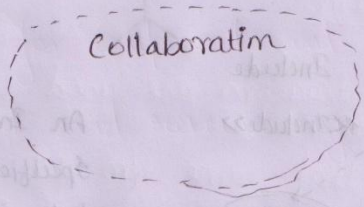
8)



A realizatin is a relatinship b/w a specificatin and its implementatin

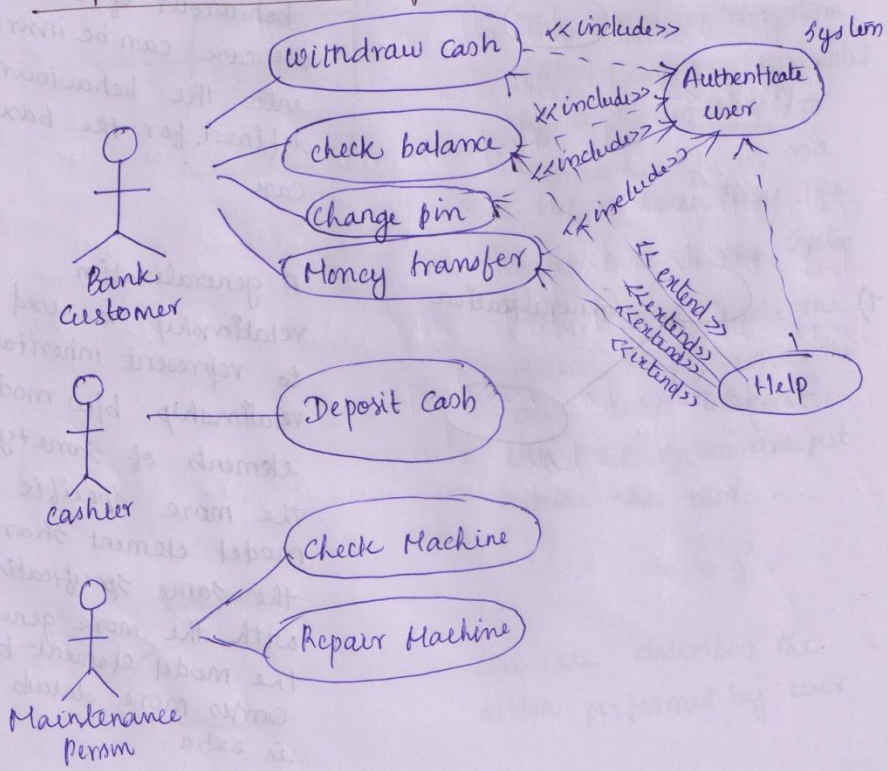
Collaboratin

9)



A collaboratin describes a structur of collaborating elements (roles) each performing a specialized funcetin, which collectively acomplis some desired funcetinality

Example: Use case diagram for ATM sim



Purpose:

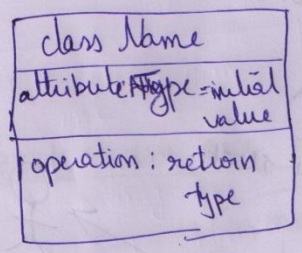
→ Analysis and design of the static view of an applⁿ.

→ Describe responsibilities of a s/m

→ Base for component and deployment diagrams

→ Forward and reverse engineering.

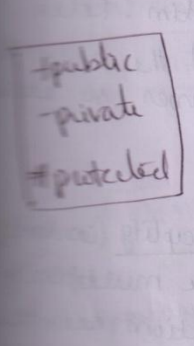
Basic Notation in class diagram

SNo	Notation	Description
1.		* class - classes represent an abstraction of entities with common characteristics * Illustrate classes with rectangles divided into compartments * Place the name of the class in the first partition, list the attributes in the second partition and write operations into third.

2 Active class

+ Active class - Active classes initiate and ctrl the flow of activity, while passive classes store data and serve other classes.

* Illustrate active classes with a thicker border



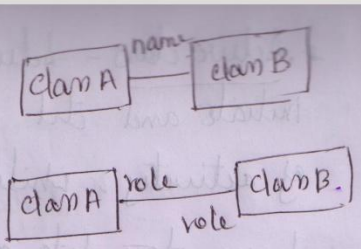
* Visibility - Use visibility markers to signify who can access the information contained within a class

* private: visibility hides information from anything outside the class partition

* Public visibility allows all other classes to view the marked information

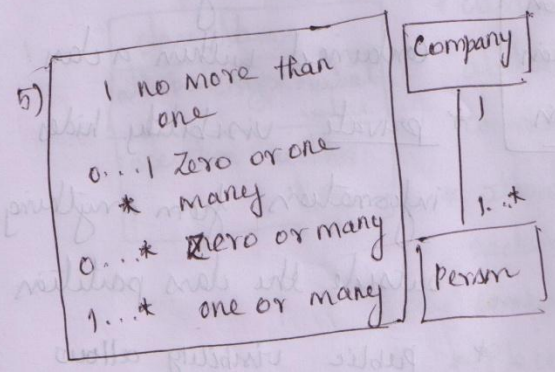
* Protected visibility allows child classes to access information they inherited from a parent class

4)



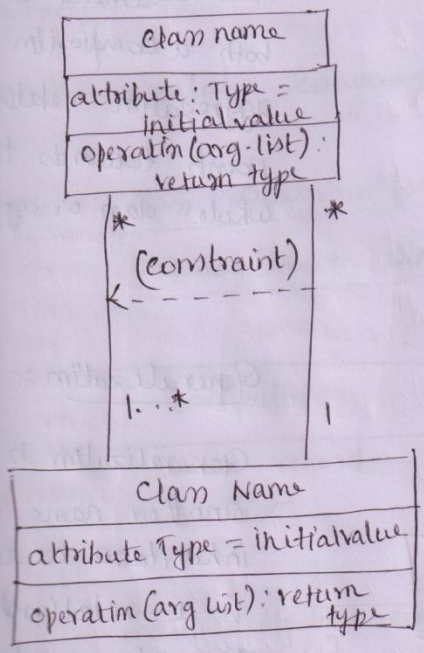
Association - Association represent static relationships b/w classes. Place association names above, on or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other

5)



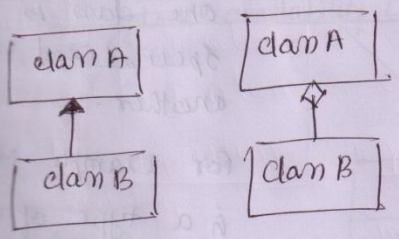
Multiplicity (Cardinality) - Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For Eg one company will have one or more employees, but each employee works for one company only.

Notations



Description

Constraint - place constraints inside curly braces {}

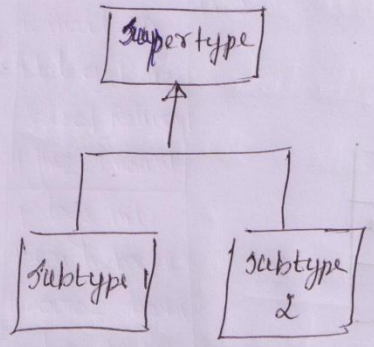


Composition and Aggregation : Composition is a special type of aggregation that denotes a strong ownership b/w class A the whole and class B, its part. Illustrate Composition with a filled diamond

Use a hollow diamond to represent a simple aggregation relationship in which the whole class plays a more important role than the part class, but the two classes are

both a composition & aggregation relationship points towards the whole class or aggregate

8)



Generalization :-

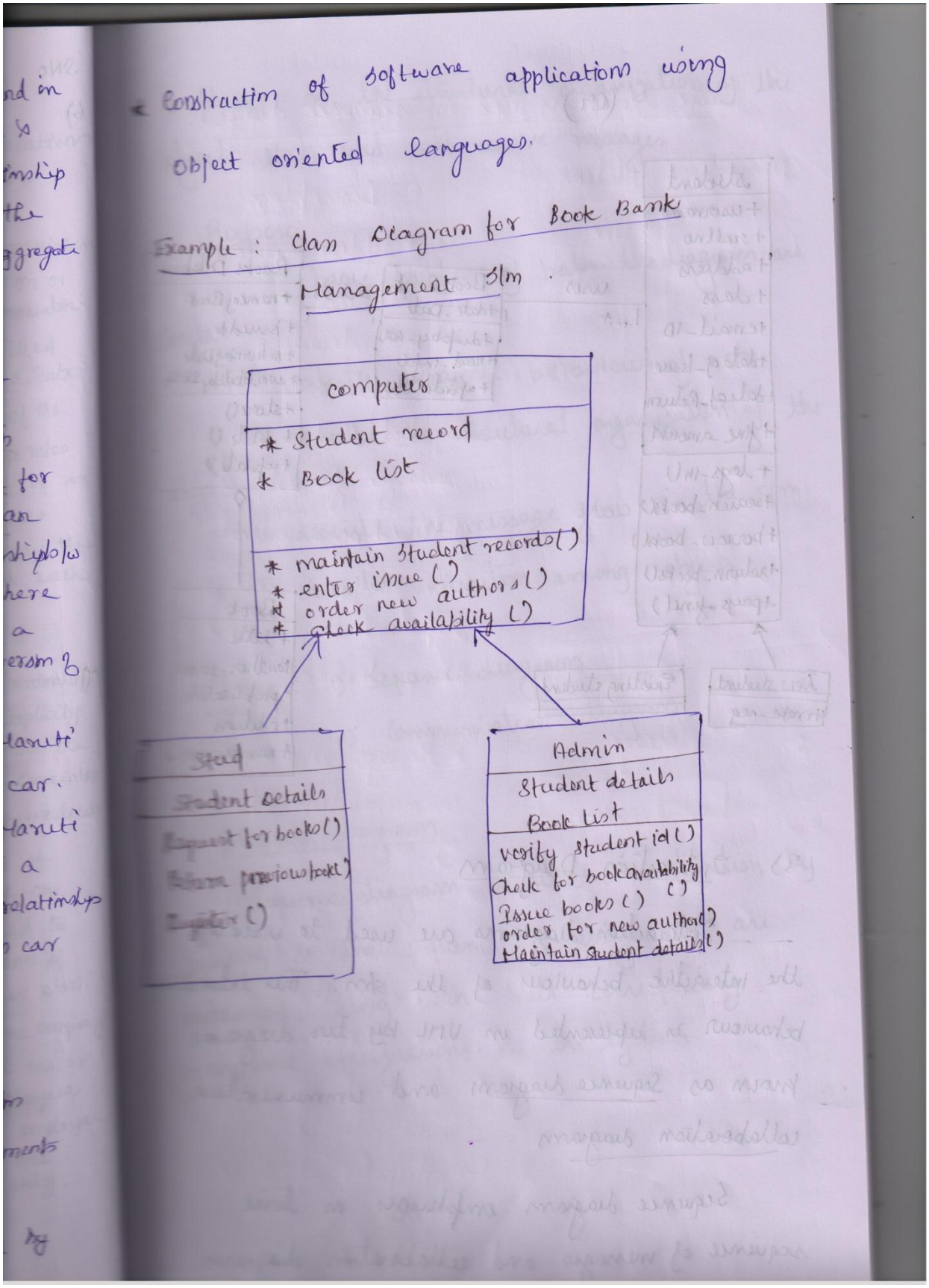
Generalization is another name for inheritance or an is a relationship b/w two classes where one class is a specialized version of another.

For example. Maruti is a type of car. So the class Maruti could have a generalization relationship with the class car

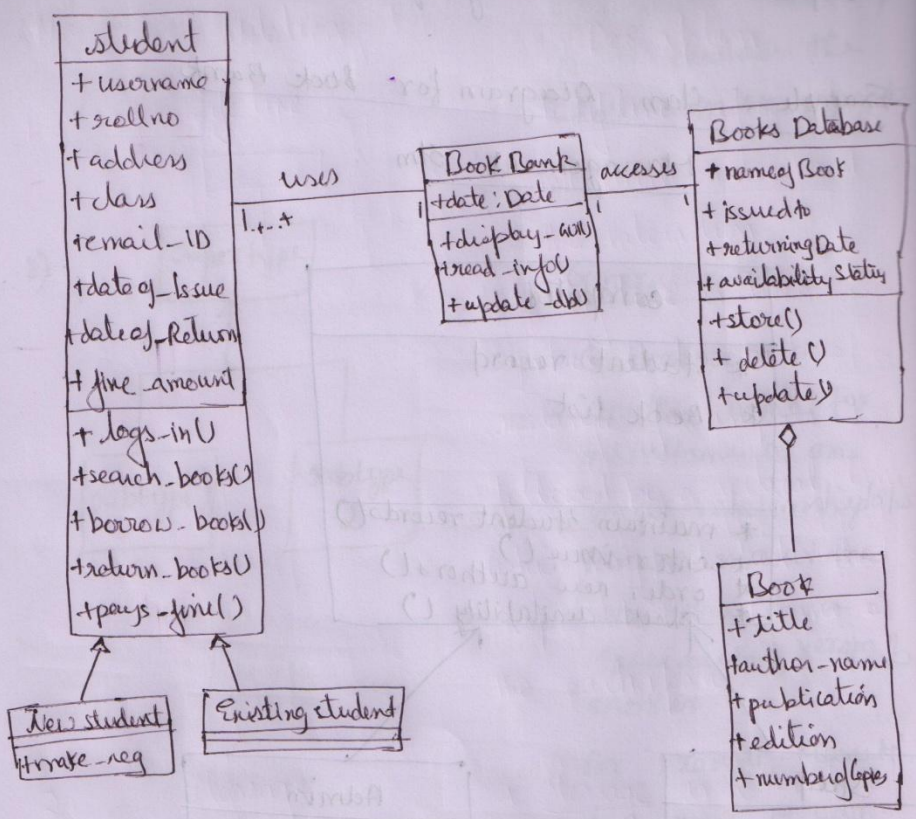
Uses:-

class diagrams are used for

* Describing the static view of the sln
* Showing the collaboration among the elements of the static view.
* Describing the functionalities performed by the sln


Construction of software applications using object oriented languages.
Example : Class Diagram for Book Bank Management Sim.
computer
* Student record
* Book list
* maintain student records()
* enter issue()
* order new authors()
* check availability()
Admin
Student details
Book List
verify student id()
Check for book availability()
Issue books()
order for new author()
Maintain student details()

(or)



3) Interaction Diagram

Interaction diagrams are used to visualize the interactive behaviour of the sm. This interactive behaviour is represented in UML by two diagrams known as Sequence diagram and communication/ collaboration diagram.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram

emphasizes on the structural organization of the objects that send and receive messages.

Purpose:

The basic purpose of both the diagrams are similar.

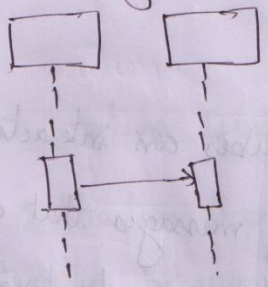
* To capture dynamic behaviour of a s/m
* To describe the structural organization of the objects
* To describe the message flow in the s/m
* To describe interaction among objects

Types: 1) Sequence diagram
2) Communication diagram

Sequence diagram

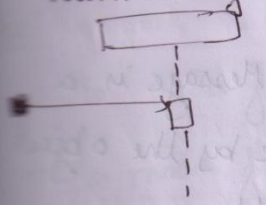
Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines

Basic Notations used in sequence diagram

S.No.	Notation	Description
1.	Life line 	* Lifeline represents an individual participant in the interaction * A lifeline is shown using a symbol that consists of a rectangle forming its "head" followed by a vertical line (which may be dashed) that represents the lifetime of the participant
2.	Message 	* Message is a kind of communication b/w lifelines of an interaction. * The message specifies not only the kind of communicati but also the sender and the receiver. * Arguments of a message could only be: ↳ attributes of the sending lifeline ↳ constants ↳ symbolic values (which

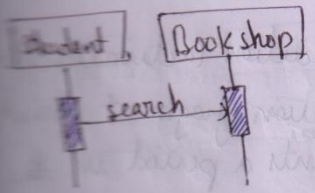
are wildcard values representing any legal value
↳ explicit parameters of the enclosing interaction
↳ attributes of the class owning the interaction

Found Message



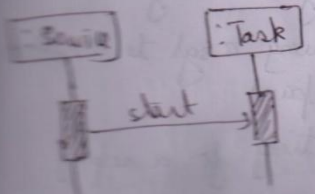
* A found Message is a msg where the receiving event occurrence is known, but there is no sending event occurrence. i.e., starting msg is called found msg

Synchronous Message



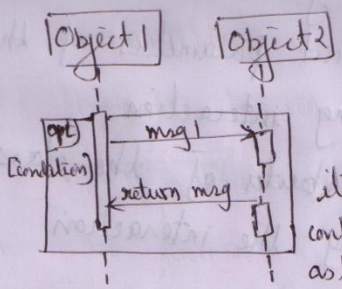
* Synchronous message typically represents operation call - send message and suspend execution while waiting for response
* Synchronous call msgs are shown with filled arrow head.

Asynchronous Message



* Asynchronous message - send msg and proceed immediately without waiting for return value
* Asynchronous messages have

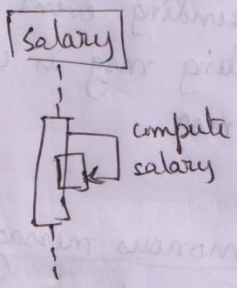
6 Conditional Message



iteration continues as long as the condition is valid

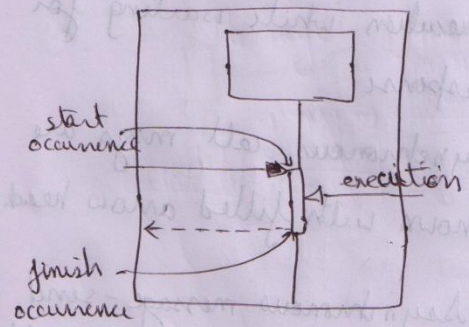
* Conditional msgs are used in sequence diagram whenever particular sequence occurs based on condition

7 Self Message



* Self Message is a message by the object to itself.

8. Execution



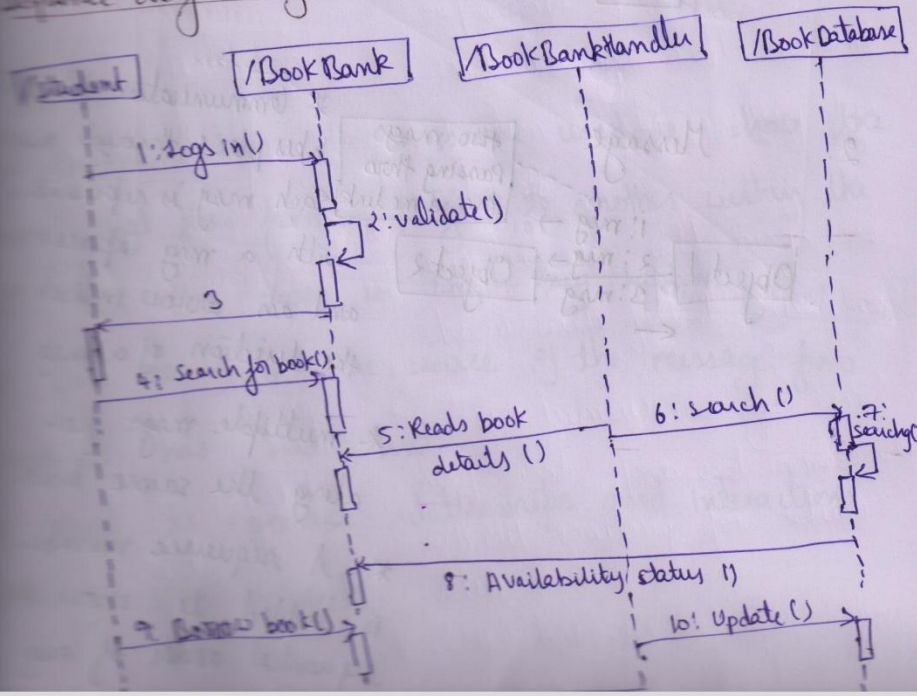
* Execution (activation) is interaction fragment which represents a period in the participant's lifetime when it is
→ executing a unit of behaviour or action within the lifeline
→ sending a sgl to another participant
→ waiting for a reply message from another participant

* The duration of an execution is represented by two execution occurrences - the start occurrences and the finish occurrences
* Execution is represented as a thin grey or white rectangle on the ~~lifetime~~ lifeline

Use:

→ To model flow of control by time sequence.
→ To model flow of control by structural organization
→ For forward engineering
→ For reverse engineering.

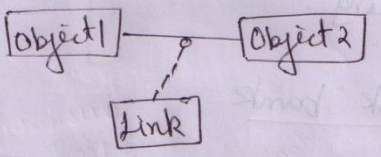
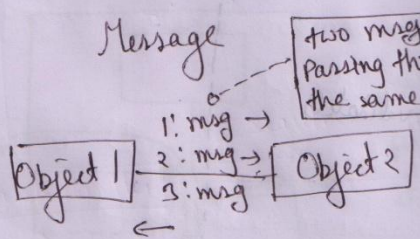
Sequence diagram for book bank



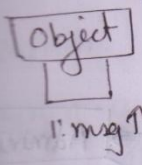
3.2 Communication Diagram / Collaboration Diagram

Communication Diagram is also used to model the dynamic behavior of s/m. When compared to Sequence Diagram, the Communication Diagram is more focused on showing the collaboration of objects rather than the time sequence.

basic notations in communication diagrams

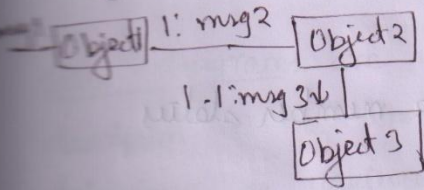
S.No.	Notation	Description
1.	Link solid (line) 	* A link is a connection path b/w two objects. It indicates navigation and visibility b/w the objects
2.	Message 	* Communication b/w objects takes place through msgs. Each msg is represented with a msg expression and an arrow indicating the direction of a msg * multiple msgs flow along the same link. * A sequence number is added to show the sequential order of msg.

Self Message



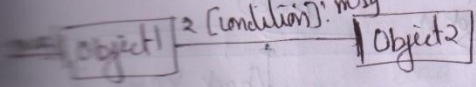
* Message passed from an object to itself.

Message Number Sequencing



* Numbers included along with the messages indicate the order of the message in an interaction

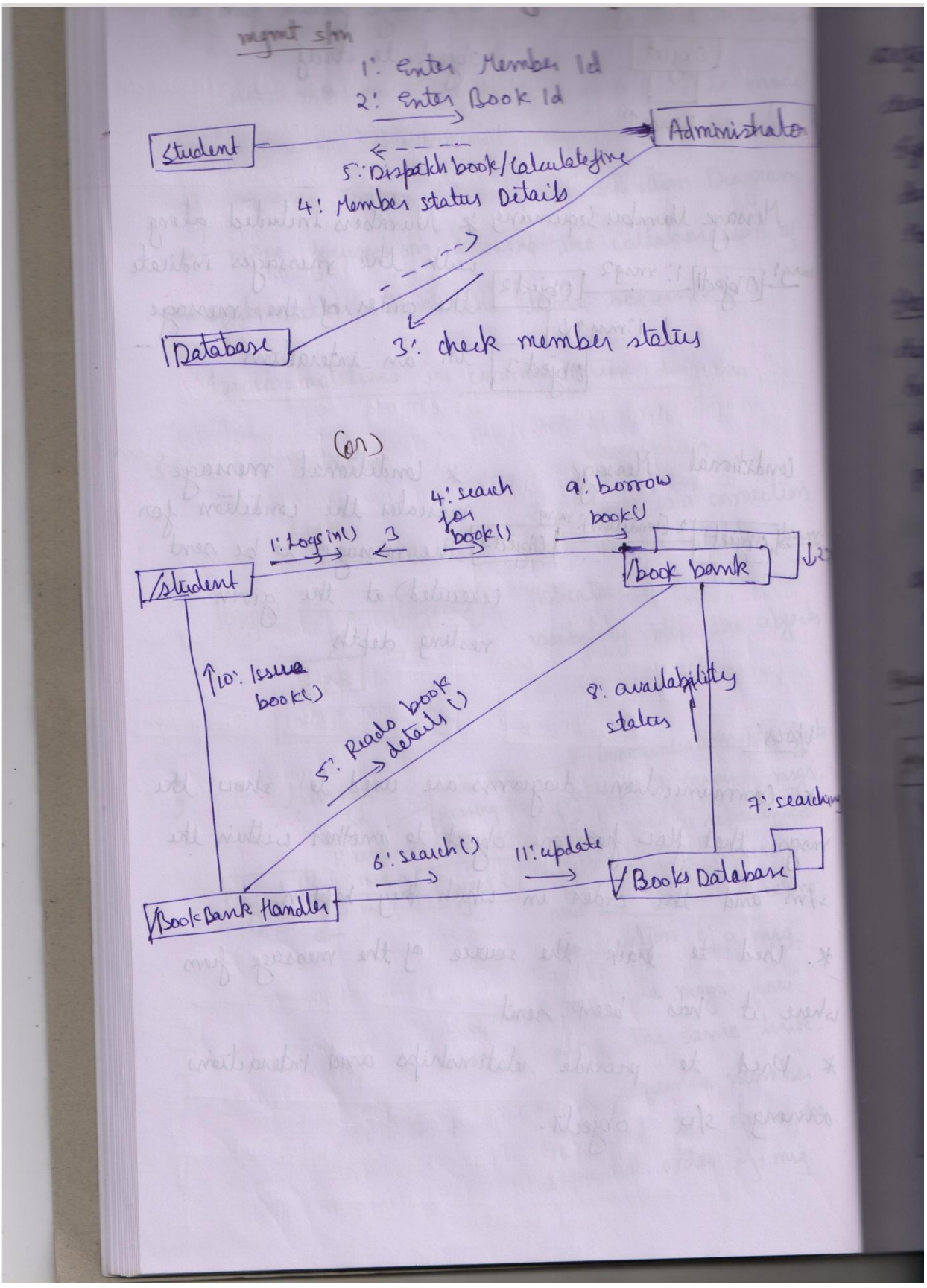
Conditional Message



* Conditional message indicates the condition for the message to be sent (executed) at the given nesting depth.

Uses:

- Communication diagrams are used to show the msgs that flow from one object to another within the sys and the order in which they happen.
- Used to track the source of the message from where it has been sent
- Used to provide relationships and interactions among s/w objects.


1': Enter Member Id
2': Enter Book Id
Student
Administrator
5': Dispatch book/Calculate fine
4': Member status Details
Database
3': check member status
(or)
4': search for book()
9': borrow book()
1': Login()
3
Student
Book bank
10': Issue book()
5': Reads book details()
8': availability status
7': searching
6': search()
11': update
Book bank Handler
Books Database

State Diagram

Definition: A State diagram is a type of diagram used to describe the behaviour of systems. State diagrams require that the system described is composed of a finite number of states.

State diagrams are used to give an abstract description of the behaviour of a system. This behaviour is analyzed and represented in series of events that could occur in one or more possible states

So Each diagram usually represents objects of a single class and tracks different states of its objects through the s/m.

Basic Notation in UML State diagram

Sno	Element	Notation	Symbol	Description
1	Initial State	Solid circle	●	This shows that starting point of the flow. Also called as pseudostate
2	State	Rounded rectangle	▭	Represents the state of object at instant of time
3	Transition	Arrow labelled with their events	Event / Action	An arrow indicating the object to transition from one state to the other

moving state as the player moves the mouse

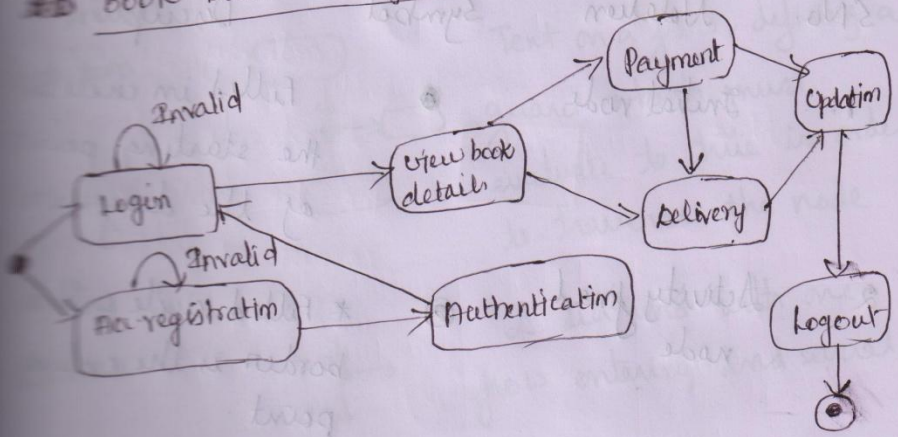
Transition

A transition is a relationship b/w two states that indicates that when an event occurs, the object moves from previous state to the next / subsequent state.

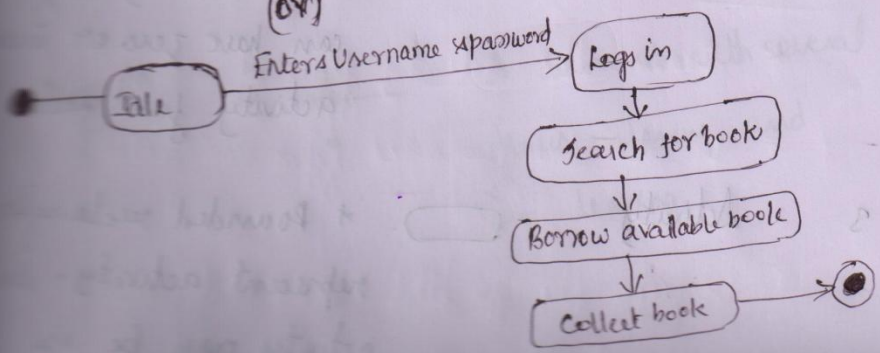
Example :-

When the event move occurs, transition takes place from the initialized to moving state.

Book Bank Management system : Example



(or)



Final State | Bullseye | ⊙ | The end of the diagram

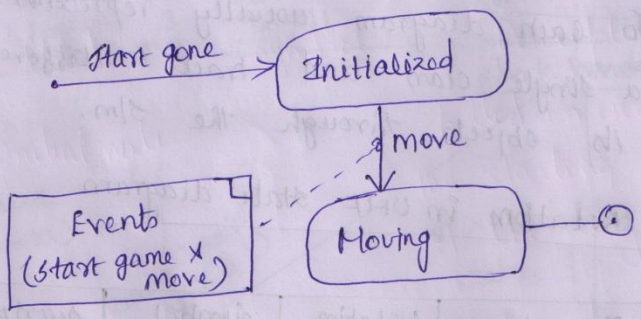
Events, States and Transitions

Events

An event is a notable occurrence at a particular point in time.

Example:

For playing a game, start game is an event to reach initialized state and move event to reach moving state



State :-

A state is the condition of an object at a moment in time (ie) the time b/w events

(or)

A state defines a stage in the evolution or behaviour of an object.

Example:-

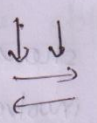
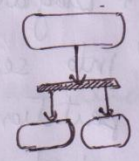
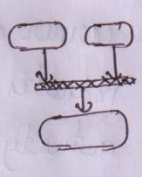
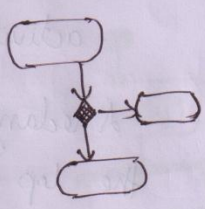
Initialized and moving are the states in the above example (ie), A game is said to be in initialized state first and then to

5. ACTIVITY DIAGRAM

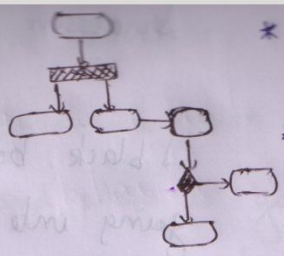
An activity diagram is to provide a view of flows and what is going on inside use case or among several classes. Activity diagrams are typically used for business process modeling for modeling the logic captured by a single use case or usage scenario or for modelling the detailed logic of a business rule.

Notations used in activity diagram

S.No.	Notation	Symbol	Description
1.	Initial node	●	Filled in circle is the starting point of the diagram
2	Activity final node	◉	* Filled circle with a border is the ending point. * An activity diagram can have zero or more activity final nodes
3	Activity	▭	* Rounded rectangles represent activity. An activity may be in physical or electronic form

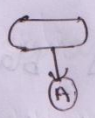
Flow/edge		Arrows on the diagram
Fork		A black bar with one flow going into it and several leaving it. This denoted the beginning of parallel activity.
Join		A black bar with several flows entering into it and one leaving it.
Guard condition		Text on a flow, defining a guard which must evaluate to true in order to traverse the node
Decision		A diamond with one flow entering and several leaving.
Merge		A diamond with several flows entering and one leaving.

10 partition



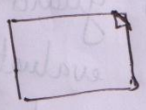
* shows different parti... involved in the proces...
* Diagram can be organize... into several vertical partitions called swimlan... indicating who/what is performing activities

11. Sub-activity indicator



The rank in the bottom corner of an activity. It indicates that the activity is described by a more finely detailed activity diagram

12. Note



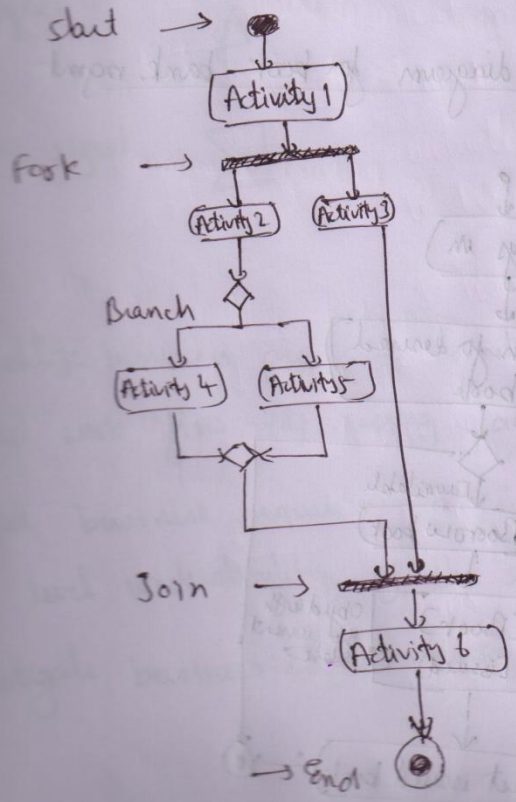
A rectangle with a flip at the top right most ~~corner~~ corner. It helps us to understand the concepts

Purpose:

* Draw the activity flow of a s/m
* Describe the sequence from one activity to another
* Describe the parallel, branched and concurrent flow of the s/m

How to apply activity diagram

- Activity diagrams show the flow of activities in the s/m
- Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities
- A fork is used when multiple activities are occurring at the same time.
- The branch describes what activity will take place based on set of conditions

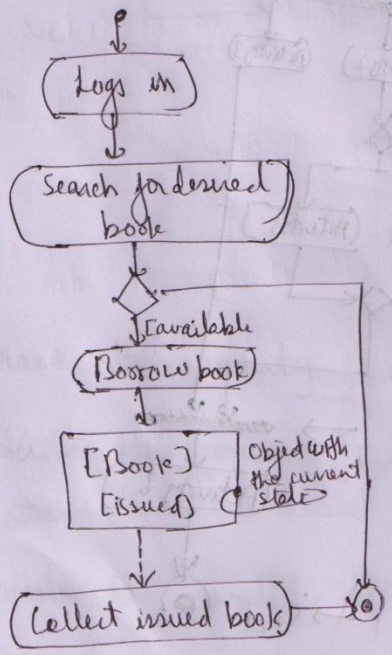


by a merge to indicate the end of the conditional behaviour started by that branch.

(vi) After the merge all of the parallel activities must be combined by a 'join' before transitioning into the final activity state.

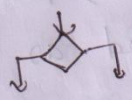
(vii) Activity diagrams are applied to visualize business workflows and processes and use cases.

Activity diagram for book bank mgmt



More UML activity diagram notation

Expansion symbol  shows the activity is expanded in another activity diagram

Decision symbol  shows conditional branches

Merge symbol  shows flows come back together

Time signal 

Sending signal 

Uses:

- Visualize business processes and workflows
- Model work flow by using activities.
- Model business requirements.
- High level understanding of the s/m's functionalities
- Investigate business requirements at a later

6) Package Diagram

Package Diagrams organize the elements of a system into related groups to minimize dependencies among them.

UML package diagrams are used to illustrate the logical architecture of a system the layers, subsystems, packages etc. A layer can be modelled as a package named UI

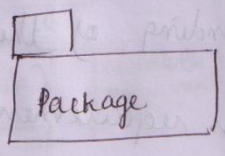
Package :-

Package is a name space used to group together elements that are semantically related and might change together. It is a general purpose mechanism to organize elements into groups to provide better structure for sys model.

Basic Notations in Package Diagram

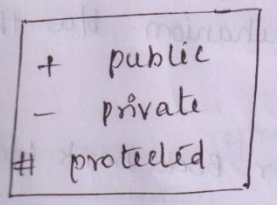
(i) package

Use a tabbed folder to illustrate packages. Write the name of the package on the tab or inside the folder. Similar to classes, list the attributes of a package



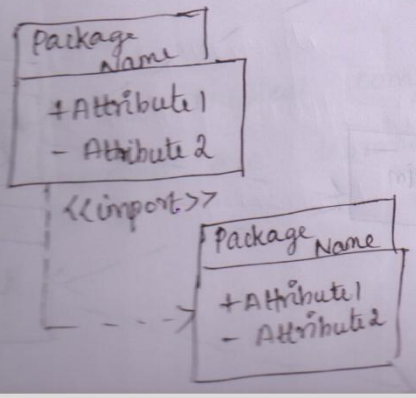
Visibility

Visibility markers signify who can access the information contained within a package. Private visibility means that the attribute or the operation is not accessible to anything outside the package. Public visibility allows an attribute or an operation to be viewed by other packages. protected visibility makes an attribute or operation visible to packages that inherit it only.



Dependency :-

Dependency defines a relationship in which changes to one package will affect another package. Importing is a type of dependency that grants one package access to the contents of another package



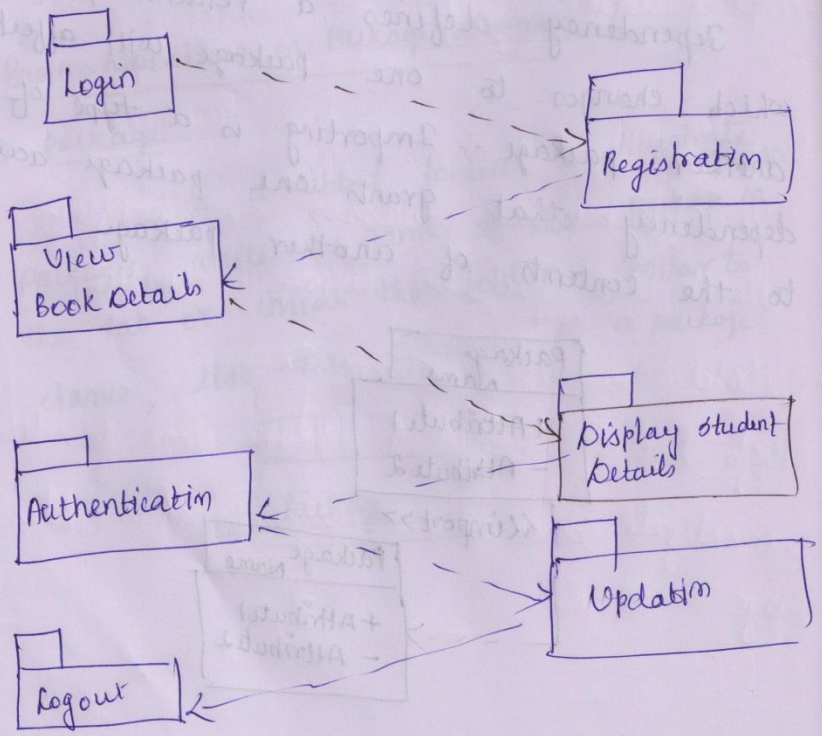
Uses:

1) Package diagrams can use packages containing use cases to illustrate the functionality of a software system

2) Package diagrams can use packages that represent the different layers of a s/w s/m to illustrate the layered architecture of a software s/m. The dependencies b/w these packages can be adorned with labels / stereotype to indicate the communicatin mechanism b/w the layers

Example:-

Package Diagram for Book Bank Management s/m



Component Diagram

→ Component diagrams are used to model physical aspects of a s/m. So Component diagrams are used to visualize the organization and relationship among components in a s/m. These diagrams are also used to make executable systems.

→ physical aspects are the elements like executables, libraries, files, documents which resides in a node.

Component :

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behaviour in terms of provided and required interfaces.

Purpose :

→ To describe the components used to make functionalities

→ To visualize the physical components in a system. These components are libraries, packages, files etc.

→ Act as a static implementation
a System. Static implementation represents the organisation of the components at a particular moment.

A single component diagram cannot represent the entire sln but a collection of diagrams are used to represent the whole

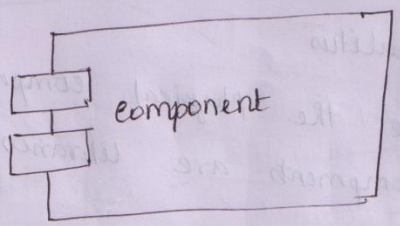
So the purpose of the component diagram can be summarized as follows.

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

Basic Notations in Component Diagram

(i) component

A component is a physical building block of the sln. It is represented as a rectangle with tabs



views of
epresents
at a

ot represent
o diagram

iagram

lm
forward

tionships of

ram

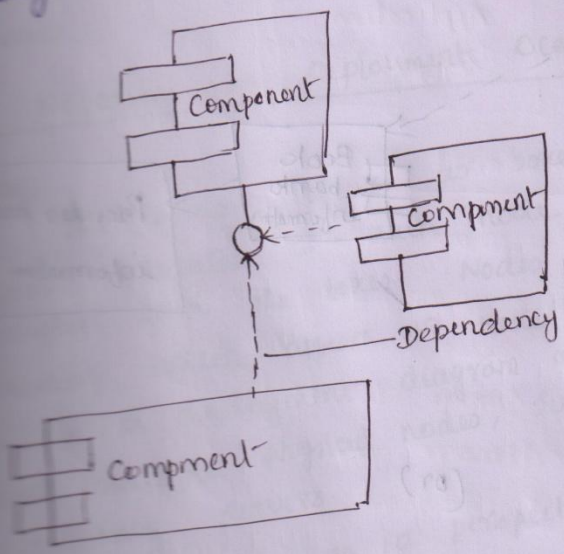
building
ntly as a

Interface :-

An interface describes a group of operations used or created by components

Dependencies

Draw dependencies among components using dashed rows



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

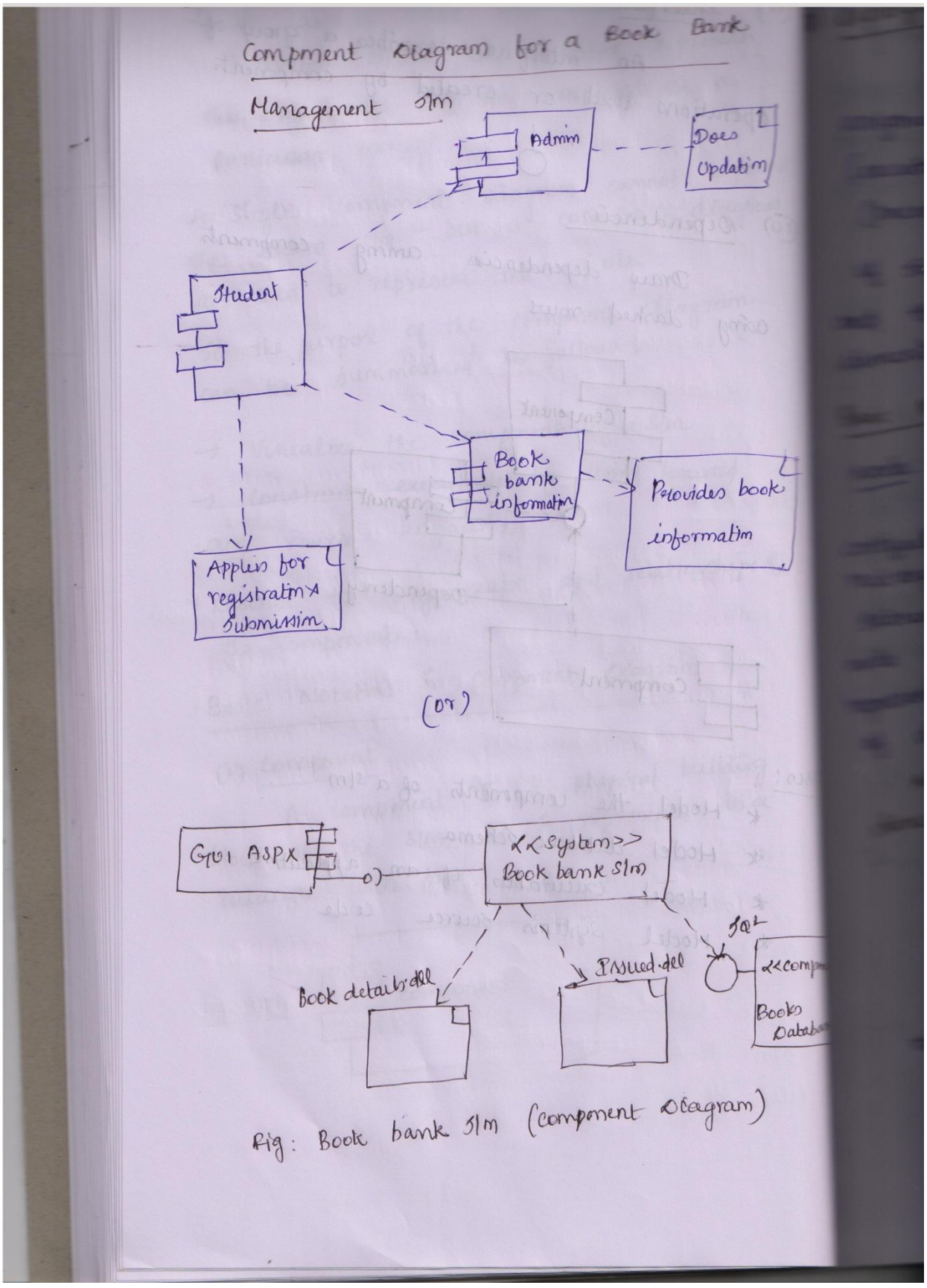


Fig: Book bank Sim (Component Diagram)

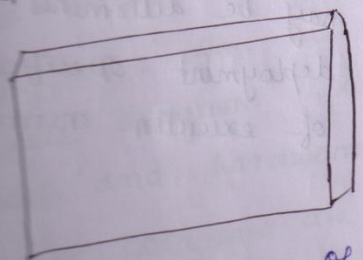
Deployment diagram

Deployment diagram is defined as an assignment of concrete software artifacts (executable files) to computational nodes (running services). In other words deployment of software elements to the physical architecture and the communication (n/w) b/w physical elements.

Notations in Deployment Diagram

The nodes appear as boxes and the artifacts allocated to each node appear as rectangles with the boxes. Nodes may have nested boxes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as cluster of database servers.

A node is shown as a perspective, 3 dimensional view of a cube



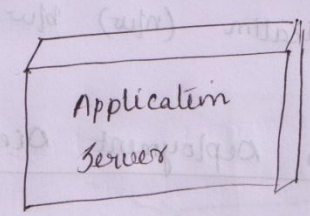
There are two types of Nodes

1. Device Node
2. Execution Environment node

Device Node

Device nodes are physical computing resources with processing memory and services to execute software such as typical computers or mobile phones.

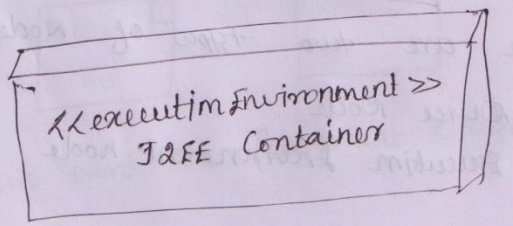
Example



Execution Environment Node

An Execution environment node (EEN) is a software computing resource that runs within an outer node and which itself provides a service to host and execute other executable software elements. Execution Environment implements a standard set of services that compments require at execution time. For each deployment of compment, aspects of these services may be determined by properties in a deployment specification for a particular kind of execution environment.

Example



nodes can be interconnected with communicatim paths

- Communicatim paths can be defined b/w such as applicatim server and database server to define the possible communicatim paths b/w the nodes. Specific topologies can then be defined through links b/w node instances

- The name Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components on which software compments are deployed. Component diagrams and deployment diagrams are closely related.

- Component Diagrams are used to describe the components and deployment diagrams show how they are deployed in hardware

- UML is mainly designed to focus n software artifacts of a sm. But these two diagrams are special diagrams used to focus n software components and hardware components

→ Deployment Diagrams are made to focus n hardware topology of a sm

The purpose of deployment diagrams can be described as

→ Visualize hardware topology of a s/m
→ Describe the hardware compments used to deploy software compments
→ Describe runtime processing nodes.

Uses :-

1) Deployment Diagrams are used to visualize the topology of the physical compments of a s/m where the software compments are deployed
2) Deployment diagrams are used to describe the static deployment view of a s/m
3) Deployment diagrams are mainly used by s/m engineers. These diagrams are used to describe the physical compments (hardware) their distributim and associatim
4) To meet busines requirements, hardware can be designed efficiently in a cost effective way through the deplogment s/m.

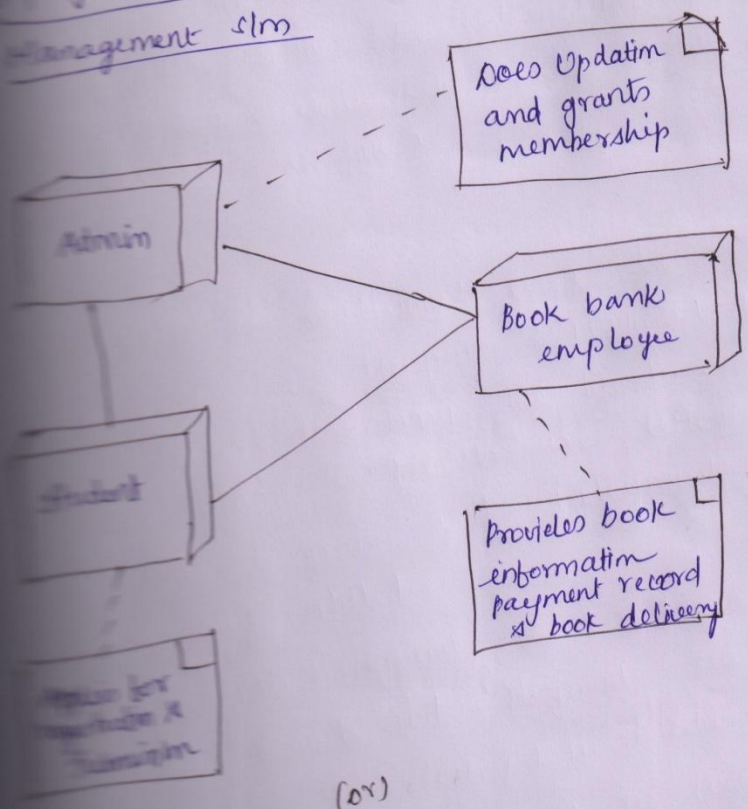
Usage of deployment diagram is as follows

→ To model the hardware topology of the s/m
→ To model Embedded s/m
→ To model Hardware details for a client/ server s/m.

→ To model hardware details of a distributed applicatim

→ Forward and reverse Engineering

Deployment diagram for a Book Bank Management sim



(or)