UNIT-II

Methodologies and UML

Rambaugh, Booch, Jacobson → Origins of UML.
Rambaugh → Suited for describing the object model or static structure of the system.

Jacobson → Good for producing user-driven analysis model.

Booch → Produces detailed object-oriented design model.

Rambaugh - object Modeling Techniques:-

(static, dynamic & functional)

→ Describes method for analysis, design and implementation using object oriented technique.
→ Fast approach for identifying & modeling all the object that makes up a system.
↳ static / dynamic / functional / object
→ Details of class, attributes, methods, inheritance can be expressed easily.
① → Dynamic behaviour of objects can be described using OMT dynamic model.
→ Allows to specify detailed state transitions and their descriptions within a system.
② → Process description & consumer-producer relationship expressed by functional model.
Static model → trained offline.

4 Phases of OMT:-
1) Analysis → Result → (object, dynamic & functional model)

2.) System design → Basic Architecture of system with high level Strategy decisions.
3.) object design → Design document, detailed objects Static, dynamic & functional models.
4.) Implementation → Produces reusable, extendable & robust code.

Static model → represents class, object, Interfaces & their relationship.
→ trained offline (i.e) we train the model exactly once & then use that trained model for a while.

Dynamic model → Diagrams: Interaction, Activity, object diag.
→ Trained online (i.e) data is continuously entering the system & we are using that data into the model through continious updates.

Functional model → It gives an overview of what the system is supposed to do. It defines the internal processes in the system with DFD.

object model → It visualizes the element in a s/w appl. in terms of object.

3 Parts of modeling:-
1.) object model → object model & data dictionary
2.) dynamic model → State diagrams & event flow diag.
3.) Functional model → data flow & constraints.

1.) object model:

→ Describes the structure of an object, their identity, relationship to other objects, attributes & operations.

→ Represented by object diagram.

→ classes are interconnected using Association lines.

→ one-to-many → ●

→ ▲ → Specialization.

2.) Functional model:

→ DFD.

→ Circles ⇒ process, Boxes ⇒ External entities.

a.) process → ⬭ → any function being performed.

b.) Data flow → ⟶ → Direction of data element movement.

c.) Data store → ═ → where data are stored (location)

d.) External entity → ▭ → source/Destination of data element ↓

Information entering or leaving the system.

3.) OMT dynamic model:

→ depict status, transitions, events & actions.

□ →

Booch Methodology:-

typical example.

→ Design system using object Paradigm.
(provides a systematic framework for designing & developing s/w systems).
→ Covers Analysis & Design Phase.

Booch prescribes { → 1.) Macro dvl. process
2.) Micro dvl. process

1.) Macro development process:-

↳ Primary concern → Technical mngtl. of the system.

Steps:

↳ Conceptualization: Produce the core requirement of the system and estl. goals & develop prototype to prove the concept.

↳ Analysis & development of the model: using class diagram to describe roles & responsibilities of object.
object diagram and Interaction diagram.

→ Design or create the system Architecture.
1.) class diagram → what class exist & how they relate.
2.) object diagram → decide what mechanisms are used to regulate how objects collaborate.
3.) Module diagram → decide where each class & objects should be declared.
4.) Process diagram → determine to which processor to allocate process.

→ Evolution & Implementation
→ Maintanence.

2.) Micro Development Process:

1.) Identify class & object
2.) Identify class & object Semantics
3.) Identify class & object relationship.
4.) Identify class & object Interfaces & Implementation.

Jacobson Methodology:-

→ Covers entire lifecycle and stress traceability b/w different Phases
→ Heart of the methodology is use-case concept.

Use-Case:

→ Scenarios for understanding system requirement. It's an interaction b/w user & a system.
→ Use-cases are:
1.) Nonformal text (no clear flow)
2.) Formal text (clear flow)
3.) Formal Style (Pseudo code)
→ Every single usecase should describe one main flow of events.
→ use common behaviours in diff. use cases
→ use-case model employs extends and uses relationship:

↓

used when one use case is similar to another use-case.

Abstract use-case:

It's not complete and has no actors that initiate it but is used by another use-case.

object oriented s/w Engineering: objectory:-

→ Also called as use-case driven development.
→ aim to fit the development of large, real-time systems.

→ Use-Cases are used in several phases like analysis, design, validation & testing.
→ Jacobson's method is used in CASE tool.

objectory is built around several models:

1.) ↳ use-case model:- defines outside (actors) & inside (use-case) of system behaviour.
2.) ↳ Domain-object model: Real world objects are mapped into domain model.
3.) ↳ Analysis: How source code is (Implementation) is carried out and written.
4.) Implementation model
5.) Test model

object oriented Business Engineering:-
↳ object modeling at enterprise level.
↳ use-cases are used.

1.) Analysis Phase:
→ Defines the system to be built in terms of problem-domain object model, requirement model & analysis model.
→ Does not care about actual implementation.
→ Reduces complexity and promotes maintainability.
→ dveloped just to form a base of understanding for the requirement model.
→ Jacobson suggest prototyping with a tool is useful during this Phase.

2.) Design and Implementation Phase:
→ Implementation environment must be identified for design model.

→ Implementation environment like DBMS, distribution of process etc.,.

3.) Testing Phase:
→ Jacobson describes several testing levels
a.) Unit testing
b.) Integration testing
c.) System testing.

PATTERNS:-
→ Reusable fixes for typical s/w design issues that occur during the development process.
(Main Idea: To provide documentation to help categorize & communicate about solutions to recurring prblm)
↓
Severe prblm / Problems
→ First use of design Pattern is by Christopher Alexander in late 1970's.
↓
2 books → 1.) A Pattern language
2.) A Timeless way of building.
→ Main Idea of using patterns is to provide documentation to help categorize and communicate about solutions to problems.
Proto-Pattern:
"Pattern in waiting" which is not yet to recur.
Coplien methods of defining a good Pattern:
1.) It solves a problem
2.) It is a proven concept
3.) The solution is not obvious.

4.) It describes a relationship.
5.) The pattern has a significant human component.

Recurring Prblm:
The problem has likely happened multiple times already & continues to happen or seems like to continue happening.

Generative (Good) and Non-Generative (Bad) Pattern:
Generative:- 1.) Describe recuring Problem
2.) Describe how to generate something new.
3.) Result can be observed in resulting system.
Non-Generative: 1.) Static and Passive.
2.) Just say about recuring Phenomena without saying how to reproduce them.

Pattern template:
The Essential components are
1.) Name: Meaningful name (to identify it and also the knowledge & structure it describes)
2.) Problem: 1.) Statement of the Problem.
2.) Goals & objectives it wants to reach.
3.) Context: 1.) Pre conditions under which Problems & solutions seems to recur.
4.) Forces 5.) solution .... etc.,

Anti Pattern:
Bad/worst practice (or) Lesson Learned.
1.) Bad situation resulted in Bad solution & Viceversa
2.) How to get out of bad situation.

Capturing Pattern:
The process of looking for patterns to document is called Pattern Mining.

FRAMEWORKS:-

→ It delivers application development Patterns to support best practice during application development.

→ It's a way of Presenting a generic solution to a problem that can be applied to all levels in a development.

→ (i) Design framework
(ii) S/w framework.

→ Single framework contains several design Patterns.

Diff. b/w Framework and Pattern:-

Framework	Pattern
1) Executable S/w.	1) Represent knowledge & experience about s/w.
2) Physical nature (Physical realization)	2) Logical nature. (Instructions to Implement the solutions).
3) can be applied on coding	3) only examples can be applied on coding.
4) written on programming language, so can be executed & reused directly.	

By Gamma:-

1) Design Patterns are more abstract than framework.
2) Design Patterns are smaller architectural elements than frameworks.
3) Design patterns are less specialized than framework.

Unified approach:-

Unified approach → Estbl. unifying & unitary framework
→ uses UML (Unified Modeling Language) ⇒
↓
to describe s/w devl. process.

→ Main idea ⇒ not to introduce another methodology, but to combine best practices, procus, methods, guidelines.

Procus and concepts of unified approach:

1.) usecase driven development
2.) O-O Analysis
3.) O-O Design
4.) Incremental development & Prototyping
5.) Continious testing

Methods & Technology:

→ 1.) UML (Modeling)
2.) approach (Layered approach)
3.) System Development (Patterns & framework)
4.) Component based development

1.) O-O Analysis:-

→ why ⇒ Extracting the needs of the system and what the system must to do, to satisfy user's requirements.
→ Goal ⇒ to understand the problem & system
→ Responsibilities ⇒ can be done by constructing models. model focus on what the system does rather than how it does it.

Steps in OOA:

1.) Identify Actors
2.) Devl. business model using UML diagrams
3.) Develop use case
4.) Devl. Interaction diagram
5.) Identify classes.

2.) O-O Design:-

UA combines all

Jacobson → analysis and interaction diagram, lifecycle model.

Produce designs that ↓ are tracable on requirem analysis, design, coding & testing.

Booch → object diagram

Rumbaugh → domain model.

process includes:

1.) Design class, attributes, methods, associations, structures, protocols, apply design axioms.

2.) Design Access layer.

3.) Design and prototype user interface.

4.) User satisfaction and usability test based on use-case

5.) Iterate and refine the design.

3.) Iterative Devl. & Testing:-

On this iterative process prototypes will be incrementally transformed in to actual application.

4.) Modeling Based on UML:-

→ Combines Rumbaugh, Jacobson, Booch.

→ UML becomes universal language for modeling system

→ UML is the standard notation for O-O modeling system

5.) UA Proposed repository:-

→ Idea to create a repository that allows max. reuse of previous experience and previously defined objects, Patterns, frameworks & user interfaces in an easily accusible manner and easy utilized format.

→ This Idea is a concept of Pattern.

Ex: Microsoft repository, Visual Age, power Builder, Visual C++.

Reuse Inside repository:-
→ objects, components, lesson from past devel. appl., mistakes, s/w components & how it should be used.
→ should be easily accessable to many people.

6.) Layered Approach to S/w Development:-
→ System developed using CASE Tools or client-server applications lean towards two-layered architecture (P.e) Interface and data.
→ Interface tied to data through routines. (routines execute when you click on a button).
→ Routines are required to acces the data, so it must exist in every screen.
→ with every Interface we create, we must re-create the business logic needed to run the screen.
→ This two layered approach makes object very Peculiar & cannot be reused on other projects.

Three layered approach:-
1.) view (or) user-Interface layer
2.) Business layer
3.) Acces layer.

1.) Business layer:-
→ Contains all objects that represents the business.
→ It model the objects of the business & how they Interact to accomplish the business.
→ On Business layer, the objects should not be responsible for the following
a.) Display Details:- objects should not have special knowledge of how they are being displayed and by whom.

b.) Data access detail:- Business objects have no special knowledge of "where they come from" & where the data is stored & retrieved.

(#) Business objects are modeled during O-O Analysis.
→ Business model captures both static & dynamic relationship among business objects.
Static => Includes object association & aggregation
Dynamic => Shows how Busi. object Interacts to perform task.

2.) View layer:
→ Contains objects with which user interacts.
→ also contains objects need to control the Interface.
Responsible for:
(1.) Responding to user interaction:
(User Interface layer is designed in such way that objects must be designed to translate action by the user in to response).
(#) Business logic doesnot exist here.
→ only contains which message to send to which object.
(2.) Display Business objects:
→ Best possible picture of B.O for the user.
→ Entry fields, list boxes, graph etc.,
(#) Interface layer objects are identified during O-O Design Phase.

3.) Access layer:-
This layer contains objects, which knows how to communicate with the place where the data actually reside (like DB, N/w etc.,)

Responsibilities:-

① Translate request:
It translates any data-related request from Business layer to appropriate protocol for data access.

② Translate result:
Translate the data requested back in to appropriate business objects.

③ Access objects are identified during O.O design.

Unified Modeling language:-

→ A model is an abstract representation of a system constructed to understand the system prior to building or modifying it.

→ Modeling enables us to cope with the complexity of the system.

→ Modeling techniques used for Analysis & design involves graphic languages (set of symbols).

→ The symbols are used according to certain rules for communicating the complex relationships of information more clearly than descriptive text.

→ Modeling is used on Analysis, Design & Implementation Phases of Sw lifecycle.

object is built around several models:-

1.) Use-case model:-
Defines outside (actors) & inside (use case) of system behaviour.

2.) Domain object model:-
objects of real world are mapped to domain object.

3.) Analysis object model:-
presents how the source code should be carried out & written.

4.) Implementation model:
Represents Implementation of the System.

5.) Test model:
Constitutes test plan, Specifications & reports.

Static and Dynamic models:-

Static => Represents System parameter at rest or at specific point in time.
→ Needed to represent static or structural aspec of the System.
→ Stable & changes on data won't occur frequen
Ex: UML Class diagram.

Dynamic => Collection of procedures or behaviour that reflect the behaviour of a system over time.
=> It shows how the business object Interact.

Static	Dynamic
1.) A System is developed first using static model which describes the structure of the class objects and its relationship.	1.) More useful of design and Implementation Phas Ex: Interaction diagram and activity models.
2.) It's like a base line.	

Why modeling?

Model language must Include
1.) Model Elements => fundamental modeling concepts & Semantics.
2.) Notation => Visual rendering of model elements.
3.) Guidelines => Expression of usage with in the model.

Modeling a problm can provide us several benefits:
1.) Clarity: Visual examination of model is possible so picking out errors & omissions becomes better rather than listings of code.
2.) Familiarity: Representation form of a model is similar to the way in which information actually is represented. More comfortable to work with this type of representation.
3.) Maintanence: Visual notation improves maintainability of a system. So we can make changes faster.
4.) Simplification:

Advantages of modeling by Turban:-
1.) Easy to express complex ideas.
2.) Seperates important from unimportant which results in reduction of complexity ⇒ to strengthen more
3.) Enhance of & reinforce learning & Training.
4.) Cost of modeling is lower than cost of experiment conducted.
5.) Making changes in model is easier than making changes on real system.

Key Ideas:
1.) A model is rarely correct on first try.
2.) Always ready to hear advice & criticism of others.
3.) Avoid discuss model revisions.

Introduction to Unified Modeling language:-

UML is a language for specifying, Constructing, Visualizing & documenting s/w systems & its components.

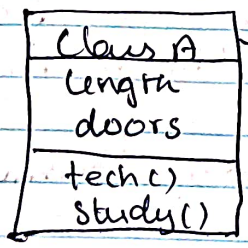
UML is a graphical language with set of rules & semantics.

rules and Semantics
↳ expressed in English form known as Object Constraint language (OCL).
↳ It is a specification language that uses simple logic for specifying the properties of a system.
(DFD not included in UML)

Primary goal in the design of UML:-
1.) Provide a basis for understanding the modeling langua
2.) Supports high level devel concepts.
3.) Independent of programming language.
4.) Provide users a ready-to-use expressive visual model, so they can develop & exchange meaningful mode
5.) Encourage the growth of OO tools.
6.) Support high level development.
7.) Integrate best practice & methods.

UML Diagrams:-
1.) UML class Diagram:-
→ Also called as object Modeling.
→ It's a main static analysis diagram.
→ These diagrams shows the static structure of the model.
→ UML C.D are a collection of static modeling elements, such as classes or their relationships connected as a graph.

class notation: static structure

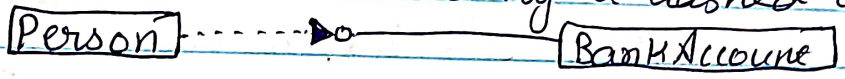


→ classname
→ attributes
→ list of operations

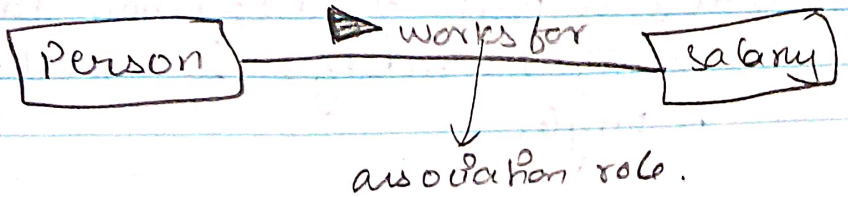
→ In this either (or) both attribute & operations can be suppressed.
→ Seperated by horizontal lines.

Object diagram:-
→ A static object diagram is an instance of a class diagram.
→ class diagram can contain objects.
→ So a class diagram with objects & no class is an object diagram.

class Interface notation:-
→ Used to describe external visible behavior of a class (Public visibility).
→ Notation :- Small circle with the name of the interface connected to the class.
→ The class that requires the operation in the interface is attached to the circle by a dashed arrow.

Ex: 

Binary Association Notation:-
→ Solid path connecting two classes (or) same class.
→ Association may have association name.
→ Association name have optional black Δ^{le}, where the point of Δ^{le} indicating in which direction to read the name.
→ End of an association, where it connects to a class is called association role.



Association Role:-

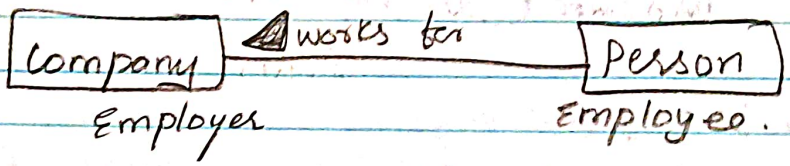
→ Simple Association ⇒ termed as Binary Association.

⇒ Solid line connecting 2 class symbols.

⇒ End of association connecting to a class is called Association role.

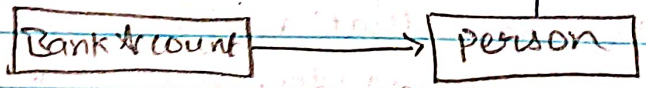
⇒ It's a part of association, not a part of class.

⇒ Each association has two or more roles.



In UML: → UML uses the term Association Navigation or Navigability to specify a role of association.

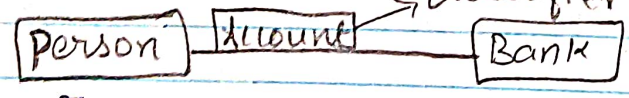
→ In UML association is represented by an open arro



(→ In this Person class is frozen & cannot be extended to know about the BankAccount class but the BankAccount class can know about Person class).

Qualifier:

→ Qualifier is an association object.



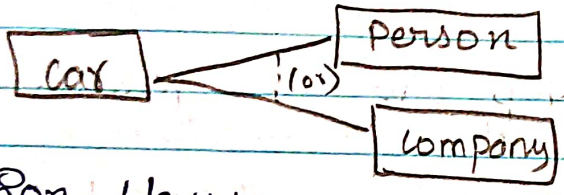
→ Qualifier represented in small rectangle attached to end of association.

→ It's a part of association Path, not a part of cla

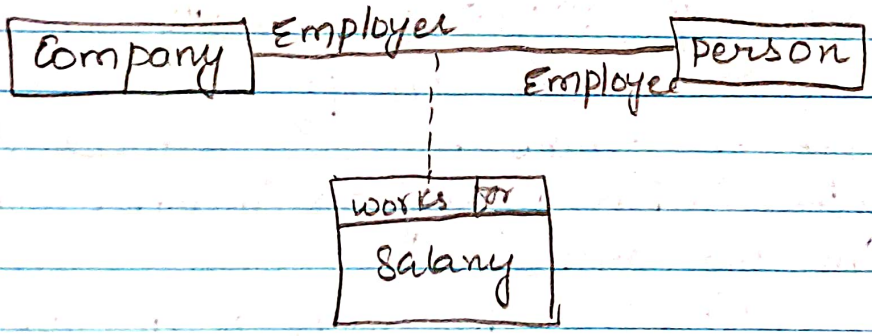
→ Qualifier rectangle is smaller than class rectangle.

Multiplicity:-
→ Range of allowable associated classes.
→ lower bound & upper bound
...1, 1-1, 1- etc.,.

OR Association:-
→ Only one of the several associations can be instantiated at one time for any single object.
Should have a common class.



Association Class:-
→ It is an association that also has class properties.
→ Shown as a class symbol attached by a dashed line to an association path.
→ Name in the class symbol and name string attached to the association path are same.



N-Array Association:-
→ Association among more than 2 classes.
→ It's difficult to understand, so it is converted to binary association.

Aggregation and composition:-

→ Aggregation is a form of Association.

→ ◇ hollow diamond is attached to the end of the Path.

→ Composition also called as (a-part-of) or (Part-whole relationship) → (strong ownership to represent the component of complex object).

→ ◆ Solid diamond.

Generalization:-

→ Relationship blw more general class and more specific class.

→ Direct line with a closed hollow arrowhead at the Superclass.

USE-CASE DIAGRAM:-

→ Introduced by Ivar Jacobson.

→ Functionality of a system is described in a number of diff. use cases, each represent a specific

→ Use-case corresponds to sequence of transaction where each transaction is invoked from outside of the system (actors) & engages internal objects to interact with one another.

→ Initiated by actors & describe the flow of events that these actors set off.

→ Ellipse containing the name of usecase.

Relationship shown on Use-case diagram:-

1.) Communication → relationship of an actor is shown by connecting actor symbol to use-case symbol.

2.) Uses

3.) Extends → used when we have one use-case similar to another use case but does a bit more.

UML Dynamic modeling :- (Behaviour diagrams)

→ So far the diagrams created & seen are static.
→ The state of an object is the result of its behaviour.

Behaviour diagrams:-

- Interaction diagram
  - Sequence diagram
  - Collaboration diagram
- Statechart diagram
- Activity diagram.

UML Interaction diagram :-

→ Describes how group of objects collaborate to get a job done.
→ Two kinds — 1.) Sequence
2.) Collaboration

Sequence Diagram :-

→ Easy way of describing the behavior of a system by viewing the interaction b/w the system.
→ shows an interaction arranged in timesequence.
→ Shows object participating in Interaction by their lifeline and message they exchange.
→ 2 dimensions
- 1.) Vertical → represents time
- 2.) Horizontal → represents object called object's lifeline

This method was popularised by Jacobson

lifeline represents object's existence during the Interaction

→ object is shown at top.
→ Role ⇒ slot for an object, that shows the type of object that may play the role & its relationship to other roles.
→ Sequence diagram dosnot show the relationship among the roles or association among objects.
→ Each message is represented by an arrow blw the lifelines of 2 objects.
→ Each message is labelled with message name
→ A sequence diagram is an alternative way to understand the overall flow of the control of a program.

Collaboration Diagram:-

Collaboration ⇒ set of objects related in particular context.

Interaction ⇒ Set of messages exchanged among the objects with in the collaboration.

In collaboration diagram, the sequence (order) of messages for communication is indicated by numbering the messages.

UML Statechart diagram:

→ Shows the sequence of states that an object goes through during its life in response to outside stimuli
→ A state is a set of values that describes object at specific point.
→ It represents the state of the method execution (i.e the state of the object executing the method)
→ Activities in the diagram represent the activities of the object that perform the method.
→ Purpose of state diagram is to understand the algorithm involved in performing a method.
→ Similar to Petri-net diagram.

● → activity symbol.

▭ → Statechart.

→ → Transition triggered by the completion of the activity.

→ An event occurs at the instant in time when the value is changed.
→ A message is data passed from one object to another.
→ State (▭) contains one or more compartments.
→ 2 types 1.) Name Compartment
2.) Internal transition compartment.

Name Compartment :→ Holds the optional name of the state. same
→ Do not show the named state twice in the same diagram.

Internal transition compartment → holds list of internal actions or activities performed in response to events received while the object is in the state, without changing states.

→ Two special events 1) Entry 2) Exit (Reserved words and cannot be used for event names)

→ entry / action expression / exit / action expression.

→ Statechart supports nested state machines.

→ to activate a substate machine use the keyword do: do / machine-name (argument-list).

(→ when an activity symbol appears within a state symbol, it indicates the execution of an operation.

→ The same operation name may appear more than once in a state diagram, indicating the invocation of same operation in different phase.)

→ A transaction can be simple or complex.

→ Relationship b/w 2 states

→ object in first state enter to second step and perform certain actions when a specific event occurs

Complex:

→ Have multiple source & target states.

→ Splitting of control into concurrent threads.

Internal transition compartment → holds list of internal actions or activities performed in response to events received while the object is in the state, without changing states.

→ Two special events 1) Entry 2) Exit
(Reserved words and cannot be used for event names)

→ entry / action expression / exit / action expression.

→ Statechart supports nested state machines.

→ to activate a substate machine use the keyword do: do / machine-name (argument-list).

(→ when an activity symbol appears within a state symbol, it indicates the execution of an operation.

→ The same operation name may appear more than once in a state diagram, indicating the invocation of same operation in different phase.)

→ A transaction can be simple or complex.

→ Relationship b/w 2 states

→ object in first state enter to second step and perform certain actions when a specific event occurs

Complex:

→ Have multiple source & target states.

→ Splitting of control into concurrent threads.

UML Activity Diagram:-

→ States are activities representing the performance of operations.

→ Transitions are triggered by completion of the operations.

→ A.D can be used to design/model an entire bussiness process.

(Represents the flow of activities or tasks in a process or system)

It is used to model the sequence of actions, decisions & interactions involved in a particular workflow.

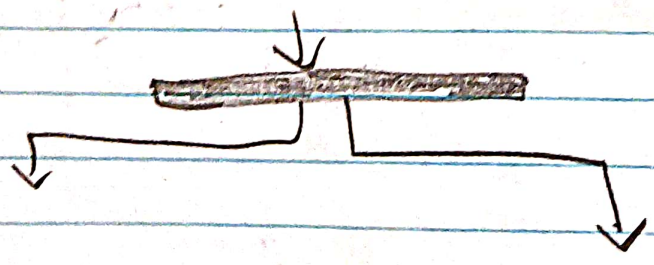
● → represents start of operation.

▭ → contains Activity (contains name of the operation)

(+) → The concurrent control is represented by multiple arrows using a synchronization bar, which is represented by short thick bar with incomming and outgoing arrows.

→ Joining concurrent control is expressed by multiple arrows entering the synchronization bar.

→ A/D



→ A.D is used to show internal state of an object but external events may appear.

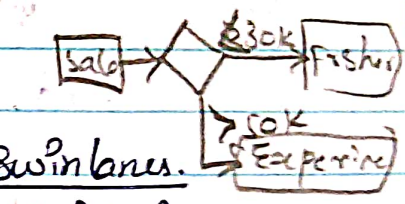
→ An external event occurs when the object is in "wait state" (a state during which there is no internal activity by the object & object is waiting for some external event to occur as a result of an activity by another object).

→ More than one possible event might take the object out of wait state

→ A Diagram expres a decision when conditions are used to indicate different possible transitions that depend on Boolean condition.

Eg:

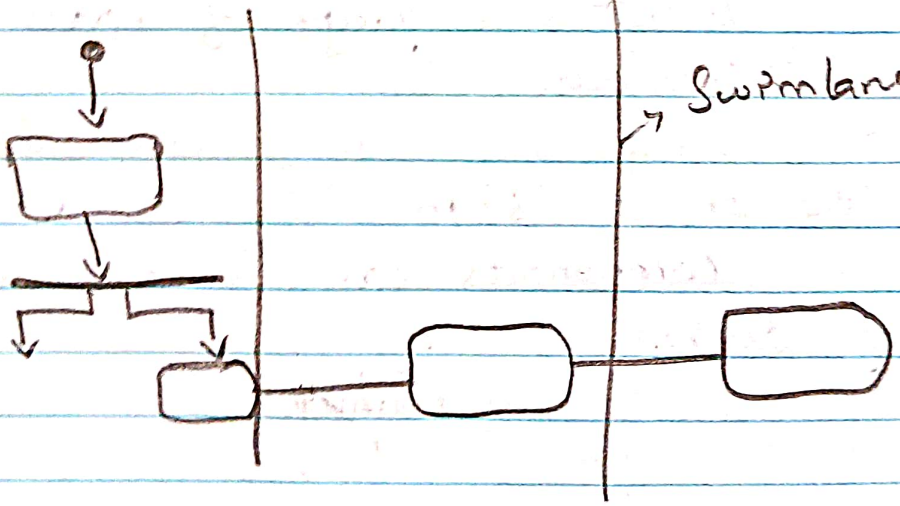
◇ - Diamond Shape.



→ Actions are organised in to swimlanes. (Vertical solid line on both sides).

→ Each swimlane represent the responsibility for part of the overall activity.

→ Each action is assigned to one swimlane.



Implementation Diagram: (shows implementation phase of system Development)

2 types

1) Component diagram: Shows the structure of the code itself.

2.) Deployment diagram: Shows the structure of the run time system.

→ can be s/w classes, collection of classes or subsystems.

Component Diagram:

→ It model the physical component (source code, executable prgm, user Interface) in a design.

→ Another way of looking at components is the concept of Packages.

→ Package is used to show how you can group together classes, which is a smaller scale component.

→ Packages used to group logical components and not physical components.

→ Component represented by boxed figure. Dependency shown as dashed arrow.

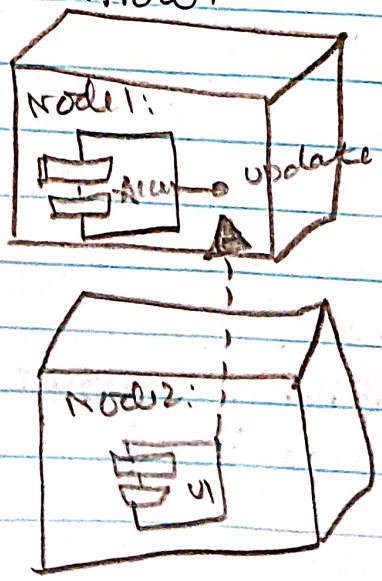
Deployment Diagram:

→ shows the configuration of run-time processing elements & s/w components, processes & objects that live in them.

→ D.D is a graph of nodes connected by communication association.

→ Components may contain objects, this indicates that the object is part of the component.

→ components are connected to other components using dashed-arrow.



Model Management : Packages & Model Organization :-

⇒ Package is a group of model elements.

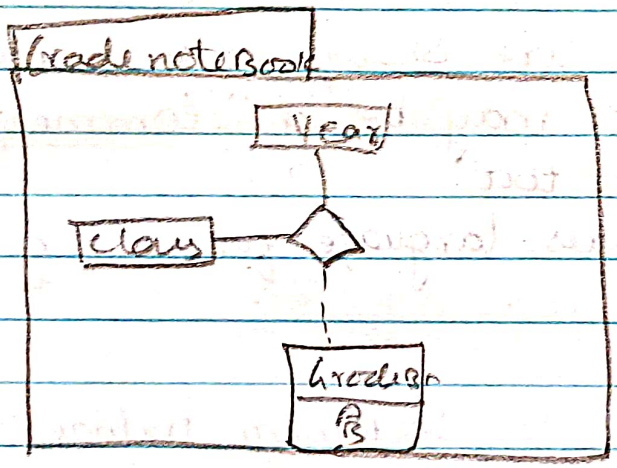
⇒ Package themselves may contain other Packages.

⇒ Entire system can be thought of as a single high-level Package.

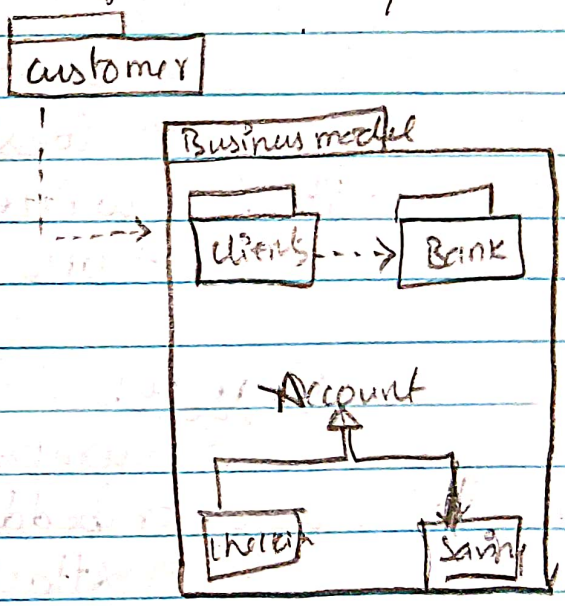
⇒ All UML model elements and diagrams can be organized in to packages.

⇒ Package is represented as a folder, shown as a large rectangle with a tab attached to its upper left corner.

Ex:: Package & its contents



Package & its dependencies



Model dependency :

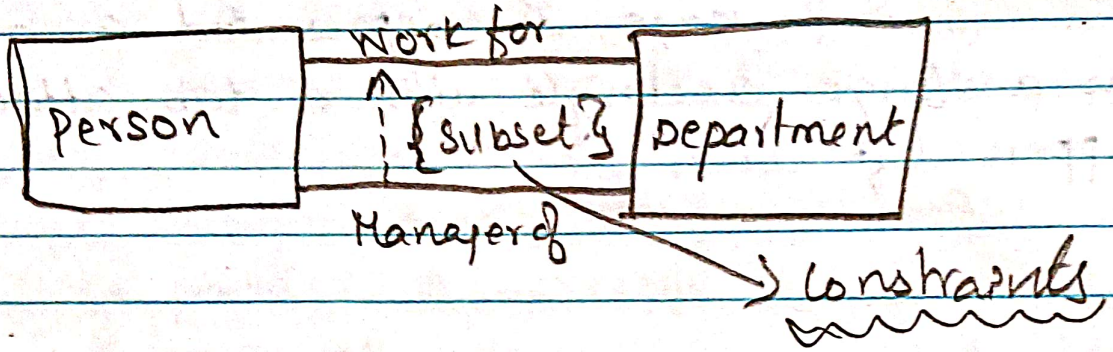
Represents a situation in which a change to the target element may require a change to the source element in the dependency, thus indicating the relationship b/w 2 or more model elements. Customer package contains (i.e) linked to Busines model through client, Bank package.

UML Extensibility:-

Mechanism applied to modeling element.

1.) Model Constraints & Comments:-

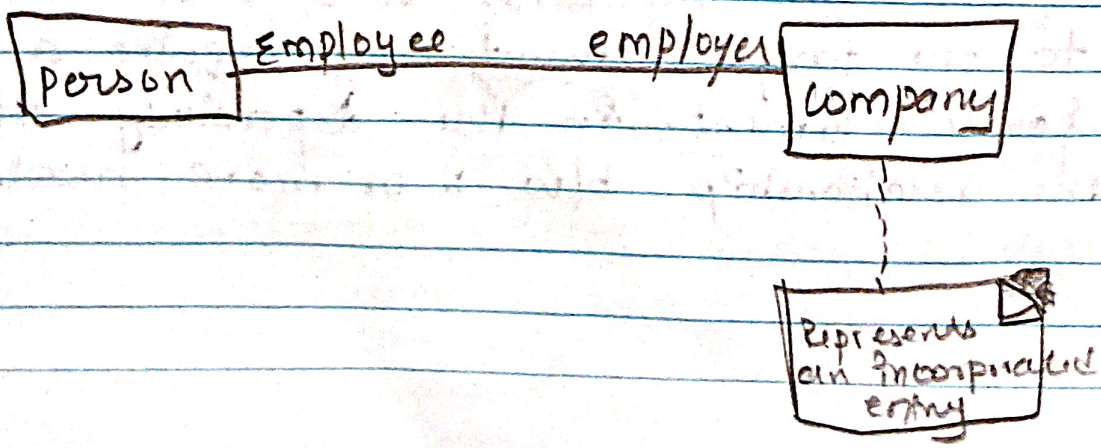
Constraints are relationship among model elements specifying conditions that must be maintained as true.



→ Constraints are shown as text in braces { }.
→ Constraint may be a "comment" in which it is written in text.
→ UML provides language for writing constraint.

2.) Note:

→ Graphic symbol containing textual information & also embedded images.
→ Attached to a diagram rather than model element
→ Represented with rectangle with a "bent corner" in the upper right corner.



3.) Stereotype:

UML Element => fundamental building block of UML model.
Ex: Class, Usecase, Activity, Association etc.,
=> A stereotype is one of 3 types of extensibility mechanism in UML (the other 2 are tags & constraints).
=> They allow designers to extend the vocabulary of UML inorder to create new model elements, derived from existing ones.
=> Graphically a stereotype is rendered as a name enclosed by guillemets (<< >>) and placed above the name of another element (or) above the name of the UML element they extend.
=> Alternatively it can be indicated by a specific icon.
=> The icon image may even replace the entire UML symbol.
=> so a stereotype is a model element that identifies the purpose of other model element.
=> Stereotype can include additional properties & constraints that further refine the behaviour & attributes of UML element they stereotype.