Introduction

UNIT-3

Unified Modeling language (UML)

→ UML is a general purpose modelling language.
→ The main aim of UML is to define a standard way to visualize the way a s/y has been designed.
→ It is used for specifying, visualizing, constructing and documenting the primary artifacts (by product of s/w development that helps describe the architecture, design & function of s/w) of the s/w s/y.
→ It is quite similar to blueprints used in other fields of engineering.
→ It is not a programming language, it is rather a visual language.
→ We use UML diagrams to portray the behavior and structure of a system.
→ UML helps s/w engineers, businessmen and s/y architects with modelling, design and analysis.
→ It helps in designing and characterizing, especially those s/w systems that incorporate the concept of object orientation.
→ UML makes the use of elements and forms associations b/w them to form diagrams.
→ UML diagrams is classified as

1) Structural Diagrams	2) Behavior Diagrams.
→ Static.	→ Dynamic.
→ capture static aspects (or) structure of a system.	→ Capture dynamic aspects (or) behavior of the system.
→ It includes, Class diagrams, Component diagrams, Object diagrams etc.	→ It includes, use case diagrams, Interaction diagrams, Activity, Sequence etc.

* Principles of Modeling.

→ Four basic principles

1) Choose your model Well

→ The choice of model profoundly impacts the analysis of the problem and the design of the solution.
(oo)
→ The right models will highlight the most nasty development problems. wrong models will mislead you, causing you to focus on irrelevant issues.

2) Every model may be expressed at different levels of precision

→ The same model can be scaled up (or down) to different granularities.

3) The best models are connected to reality

→ Simplify the model, but dont hide important details.
→ In s/w, the gap b/w the analysis model and the sys design model must be less.

4) No Single model is Sufficient

→ a set of models is needed to solve any non trivial s/y.

Conceptual model of UML

- Building Blocks
  - Things
    - Structural things
      - Classes
      - Interfaces
      - Collaborations
      - Use case
      - Components
      - Node.
    - Behavioral Things
      - Interaction
      - State machines
    - Grouping Things
      - package
    - Annotational things
      - ~~Node~~
      - Notes
  - Relationships
    - Association
    - Dependency
    - Generalization
    - Realization
  - Diagrams
    - class
    - objects
    - Use case
    - sequence
    - Collaboration
    - State chart
    - Activity
    - Deployment
    - Component.
- Rules
  - Names
  - scope
  - visibility
  - Integrity
  - Execution
- Common mechanisms
  - Specifications
  - Adornments
  - Common Divisions.
  - Extensibility mechanism
    - Stereotypes
    - ~~tagged~~ tagged values
    - Constraints

* Conceptual model of the UML

↓

To understand UML, you need to form a conceptual model of language, 4 this requires 3 major elements.

1) UML's Basic Building blocks
2) The rules
3) mechanisms.

I Building Blocks of UML

3 types

1) Things 2) Relationships 3) Diagrams

1) Things in the UML :-

→ 4 kinds

a) Structural things
b) Behavioral "
c) Grouping "
d) Annotational"

a) Structural things :- Static part of a model, representing elements that are either conceptual or physical

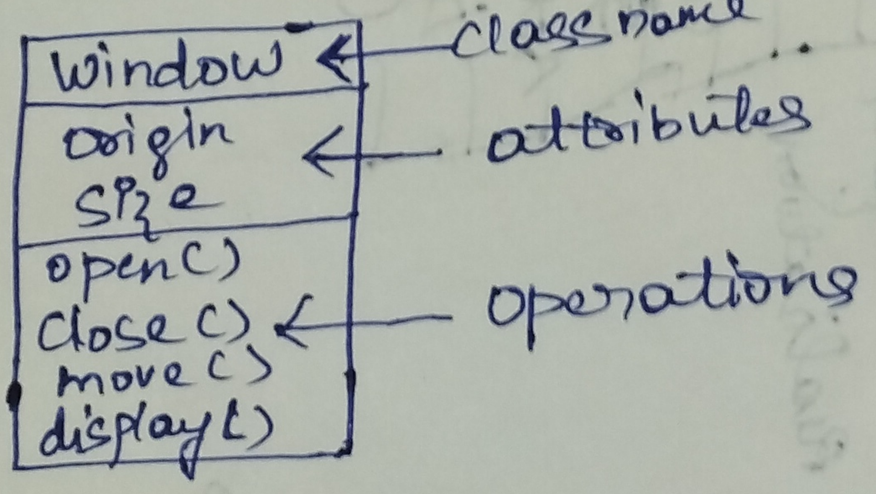
- classes, - interfaces, - Collaborations, - use cases, - Active classes, - Components, - Nodes.

- Class

→ Class is a description of a set of objects that share the same attributes, operations, relationships and semantics.

→ Class implements one (or) more interfaces.

Diagram

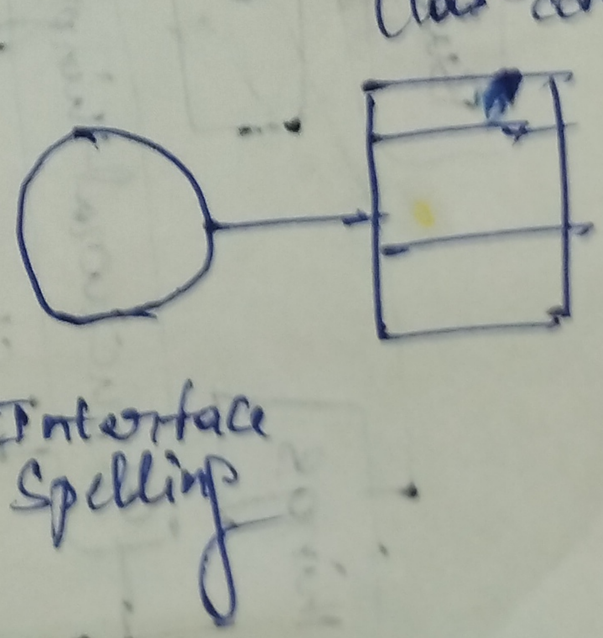


- Interface

→ a collection of operation that specify a service of a class (or) component.

→ Describes externally visible behavior of that elements.

→ Attached to class (or) component that realizes the Interface.



- Collaboration

→ Defines an interaction, which have structural, as well as behavioral dimensions.

→ A given class may participate in several collaborations.

(chain of responsibility)

- Usecase :

→ A description of set of sequence of action that a s/y performs that yields to observable result of value to a particular actor.

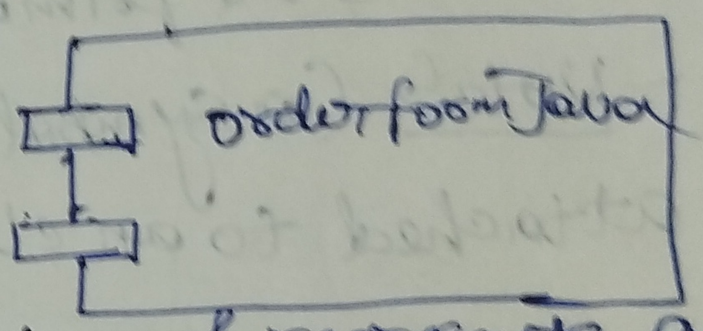
(place order)

→ It is realized by collaboration.

- Component

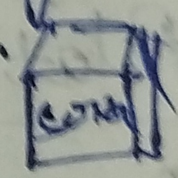
→ a physical and replaceable part of a s/y that conforms to and provides the realization of a set of interfaces.

→ represents physical packaging of classes, interfaces and collaboration.



- Node

→ a physical element that exists at runtime & represents a computational resource, having at least some memory and often processing capability.



→ A set of components may reside on a node and may also migrate from node to node.

b) Behavioral things

→ Behavioral things are dynamic part of UML models.

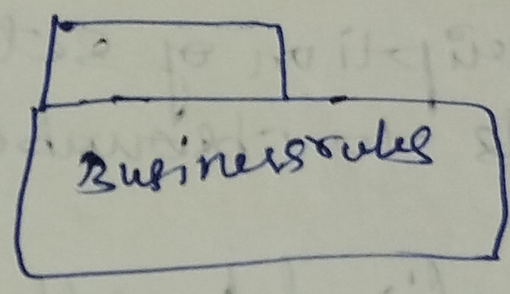
→ These are the verbs of a model representing behaviour overtime and space.

→ Two primary kinds of behavioral things.

- Interaction (display messages)
- State machine (waiting states)

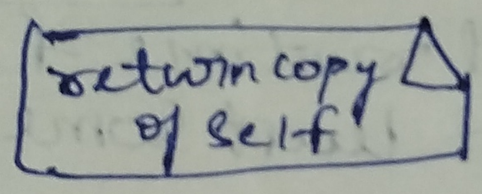
c) Grouping Things

→ Grouping things are the organizational parts of UML models.
→ These are the boxed into which a model can be decomposed.
→ There is one primary kind of grouping.
- Packages (mechanism for organizing elements into groups)
- a package is purely conceptual meaning that it exists only at development time.



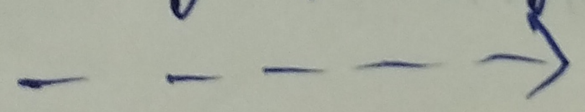
d) Annotional things.

→ Annotional things are the explanatory parts of UML models. These are the comments. you may apply to describe, & remark about any element in a model.
→ There is one primary kind of annotional things.
- Notes (a symbol for rending constraints and comments attached to an element or a collection of elements)

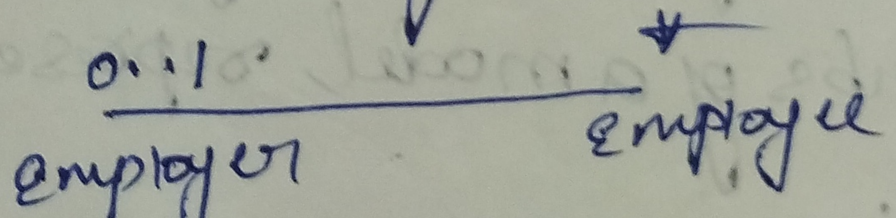


2) Relationships in the UML

a) Dependency :- change of thing may affect the other



b) Association :- a structural relationships that describes a set of links, a link being a connection among objects.



→ It contains a multiplicity and role names.
⇒ (The description of a set of links blw objects)

c) Generalization :-

→ It is also called a parent-child relationship.
→ In generalization, one element is a specialization of another general component. → It may be substituted for e.g.
→ It is mostly used to represent inheritance.

⟶▷

d) Relization :-

→ one entity denotes some responsibility which is not implemented by itself and the other entity that implements them. → This relationship is mostly found in the case of interfaces.

- - - ▷

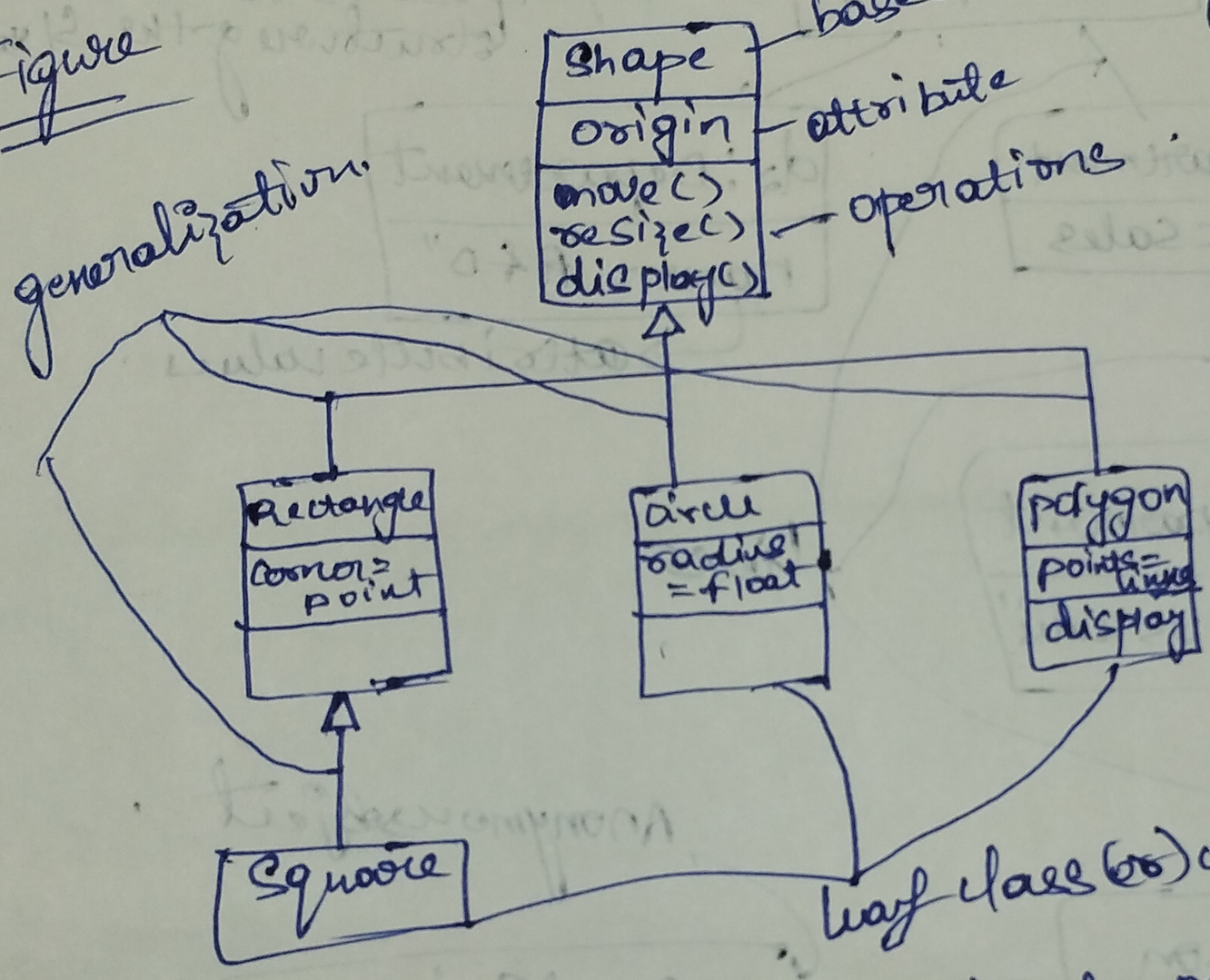
* Class Diagram

→ Class diagram shows a set of classes, interfaces, and collaborations and their relationships.

→ Most common digram found in modeling object orient systems.

→ Address the static design view of a s/y

Figure



→ The purpose of class diagram is to model the structure/ skeleton of the s/w s/y

→ Class diagram as a static diagram.

→ It represents the static view of application.

→ It describes the attributes & operations of a class & also the constraints imposed on the s/y.

Purpose of Class Diagram

→ Analysis & design of the static view of an application.
→ Describe responsibilities of a system.
→ Base for component & deployment diagrams.
→ Forward & reverse engineering
→ A collection of class diagram represent the whole system.

Following points should be remembered while drawing a class diagram :-

① The name of class diagram should be meaning-full to describe the aspect of the system.
② Each element & their relationships should be identified in advance.
③ Responsibility (attributes & 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 Finally, before making the final version, the diagrams should be drawn on plain paper & reworked as many times as possible to make it correct.

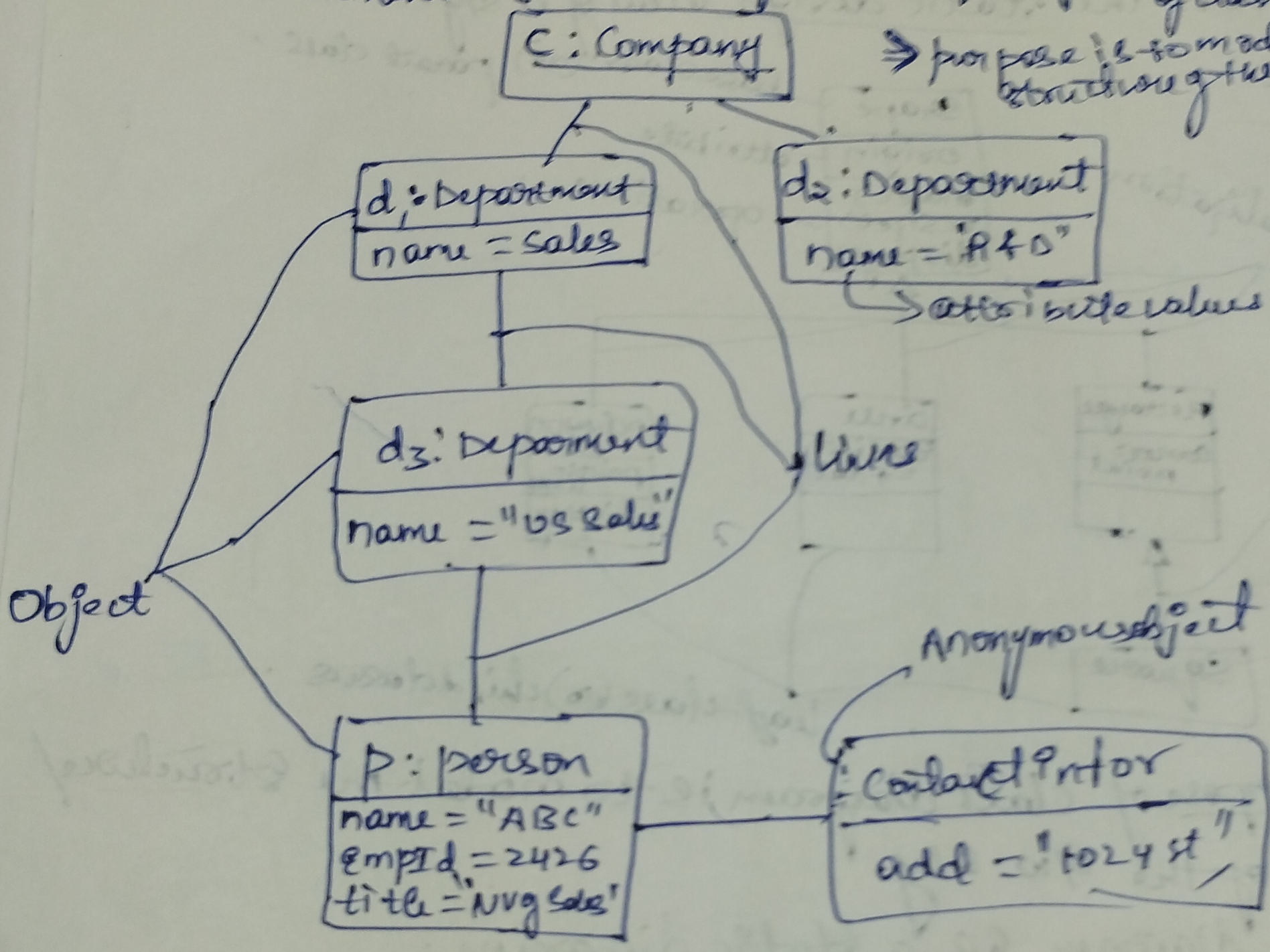
* Object Diagram

→ Object diagram shows a set of objects and their relationships.

→ Object diagram represent static snapshots of instance of the things found in class diagram.

→ The fundamental of object diagram completely stands on basis of class diagram.

⇒ purpose is to model the structures of the flow sty.



→ attribute values

→ It may include data values of entities(∞) attributes in the sty.

⇒ object diagram shows how these objects behave at run time.

→ object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

→ It represent an instance of a class diagram.

Purpose of Object Diagrams

⇒ The purpose is to capture the static view of a system at a particular moment.

① Forward & reverse engineering
② Object relationships of a system.
③ Static view of an interaction.
④ Understand object behaviour & their relationship from pratical perspective

Following things are to be decide before starting the construction of the diagram:

① Object diagram should have a meaning-ful name to indicate its purpose.
② The most important elements are to be identified
③ The association among objects should be clarified.
④ Values of different elements need to be captured to include in the object diagram.
⑤ Add proper notes at points where more clarity is required.

Common Modeling Techniques

of
Class Diagram

Modeling Simple Collaborations

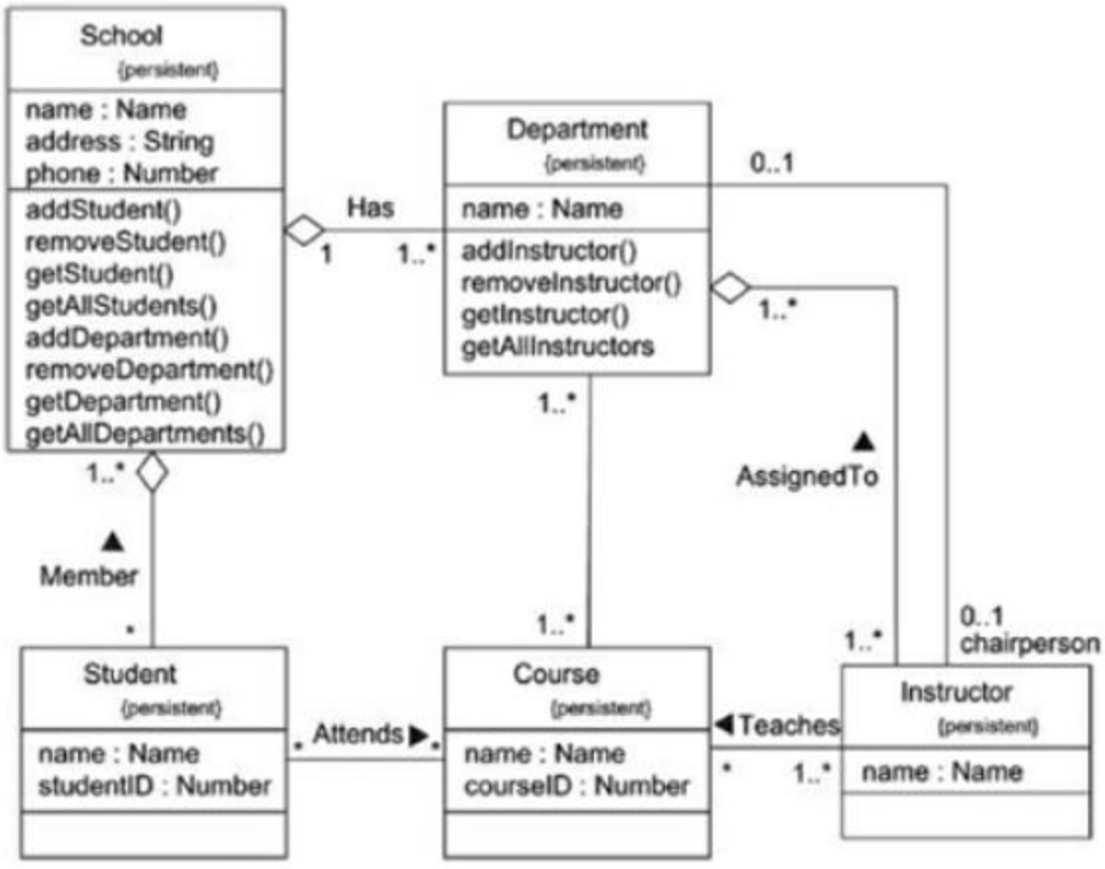
To model simple collaborations,

1. Identify the function or behavior of the part of a system you would like to model.
2. For each function or mechanism identify the classes, interfaces, collaborations and relationships between them.
3. Use scenarios (sequence of activities) to walk through these things. You may find new things or find that existing things are semantically wrong.
4. Populate the things found in the above steps. For example, take a class and fill its responsibilities. Now, convert these re-sponsibilities into attributes and operations.

Modeling a Logical Database Schema

To model a schema,

1. Identify the classes whose state must be saved over the lifetime of the application.
2. Create a class diagram and mark these classes as persistent by using tagged values.
3. Provide the structural details for these classes like the attributes, associations with other classes and the multiplicity.
4. Minimize the common patterns which complicate the physical database design like cyclic associations, one-to-one associations and n-ary associations.
5. Provide the behavior for these classes by listing out the operations that are important for data access and integrity.
6. Wherever possible, use tools to convert the logical design to physical design.



Forward and Reverse Engineering

To forward engineer a class diagram,

1. Identify the rules for mapping the models to the implementation language of your choice.
2. Depending on the semantics of the language, you may want to restrict the infor-

3. Use tagged values to specify the target language.
4. Use tools to convert your models to code.

Common Modeling Techniques

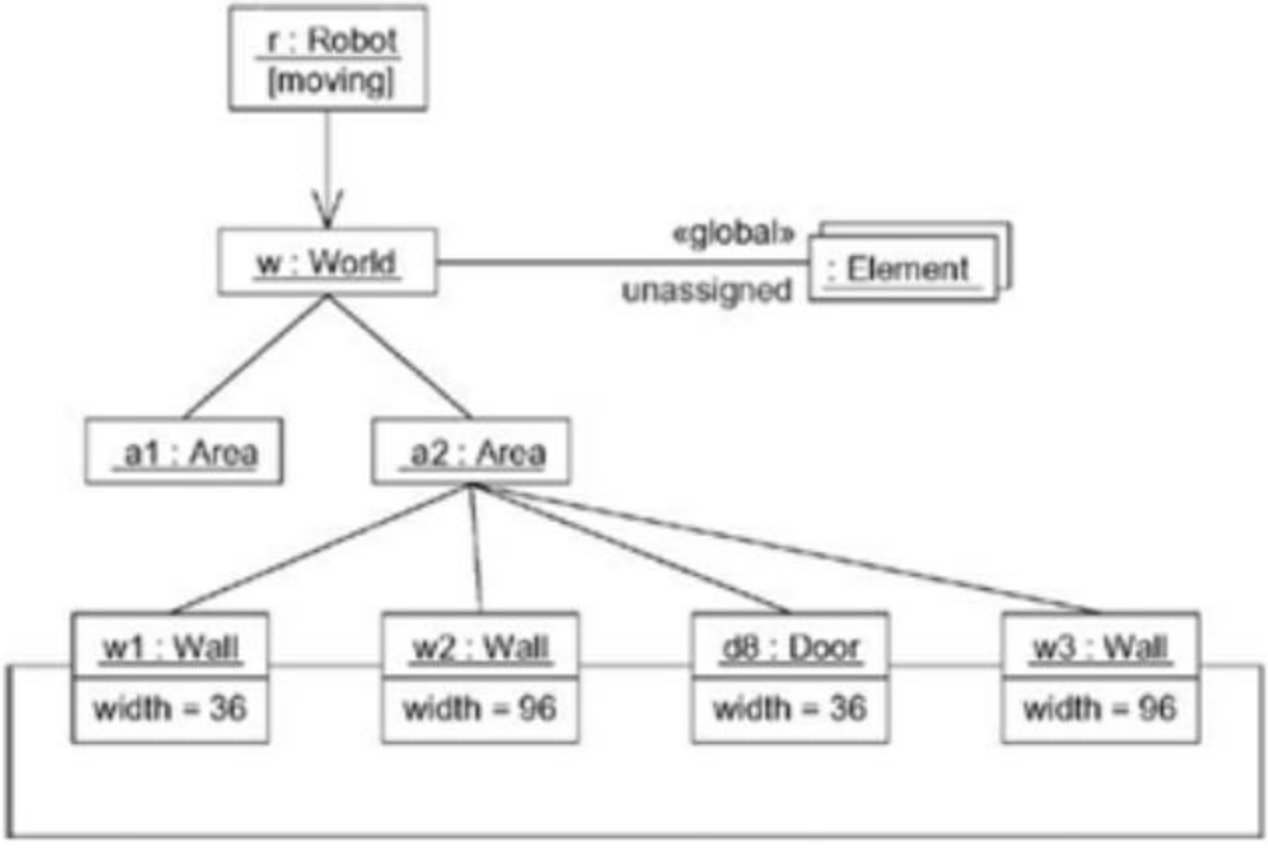
of
Object Diagram

Modeling Object Structures

To model an object structure,

- First, identify the function or behavior or part of a system you want to model as collection of classes, interfaces and other things.
- For each function or mechanism identify the classes and interfaces that collaborate and also identify the relationships between them.

- Consider a scenario (context) that walks through this mechanism and freeze at a moment in time and identify the participating objects.
- Represent the state of objects by listing out the attributes and their values.
- Represent the links between objects which are instances of associations.



Forward and Reverse Engineering

Forward engineering a object diagram is theoretically possible but practically of limited value as the objects are created and destroyed dynamically at runtime, we cannot represent them statically.

To reverse engineer a object diagram,

1. Choose the target (context) you want to reverse engineer.
2. Use a tool to stop execution at a certain moment in time.
3. Identify the objects that collaborate with each other and represent them in an object diagram.
4. To understand their semantics, expose these object's states.