UNIT-1

INTRODUCTION, MODELLING CONCEPTS, CLASS MODELLING

Object-oriented modeling & design is a way of thinking about problems using models organized around real world concepts. The fundamental construct is the object, which combines both data structure & behavior.

Defn: Object orientation means that we organize software as a collection of discrete objects (that incorporate both data structure & behavior)

There are 4 aspects required by an OO approach

1) Identity: identity means that data is quantized into discrete distinguishable entities called objects.
Ex: Can, PC, bicycle
* objects can be concrete (such as a file in a file system) or conceptual (such as scheduling policy in a multiprocessing OS)

2) Classification: classification means that objects with the same data structure (attribute) & behavior (operations) are grouped into a class
Ex: paragraph, chess piece
* Each object is said to be an instance of its class

3) Inheritance: It is the sharing of attributes & operations among classes based on a hierarchical relationship. A super class has general information that sub classes refine & elaborate
Ex: Scrolling window & fixed window are sub-classes of window.

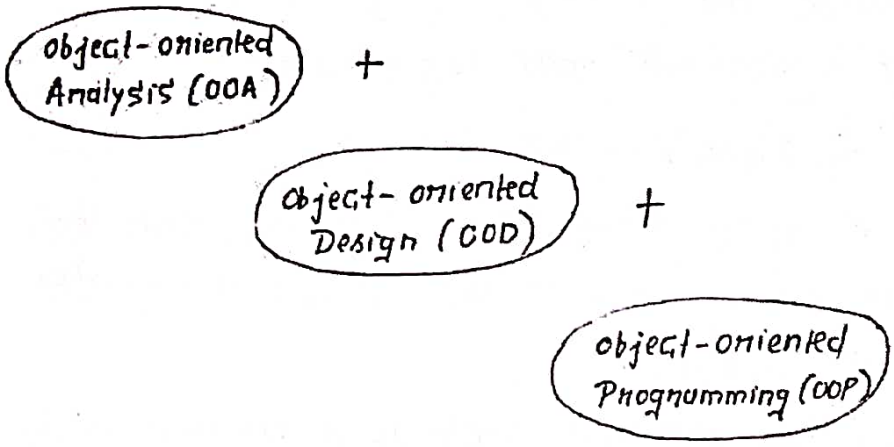
4) Polymorphism : polymorphism means that the same operation may behave differently for different classes

Ex: move operation behaves differently for a pawn than for the queen in a chess game

~~Encapsulation~~

Encapsulation

Object - Oriented Development



* Development refers to the software life cycle: Analysis, Design & implementation. The essence of OO Development is the identification & organization of application concepts, rather than their final representation in a Programming language.

Object oriented Methodology

* The Process of building a model of an application & then adding details to it during design.
* It is transformation of one stage of development to another.
* Here we present a process for OO development & a graphical notation for representing Object Oriented Concepts.

The methodology has the following stages

* System Conception - Software development begins with business analysis on users conceiving an application & formulating tentative

* Analysis - The analyst scrutinizes & rigorously restates the requirements from the system conception by constructing models.

The Analysis model is a concise, precise abstraction of what the desired system must do, not how it will be done.

The Analysis model has 2 parts

1) Domain Model : A description of real world objects reflected within the system.

2) Application Model : A description of parts of the application system itself that are visible to the user.

Ex: In case of stock broker application.

Domain objects may include - stock, bond, trade & commission

Application objects might control the execution of trades & present the results.

* System Design - The development team device a high-level strategy. The System Architecture - for solving the application problem. The system designer should decide what performance characteristics to optimize, choose a strategy of attacking the problem & make tentative resource allocations

* Class Design : The class designer adds details to the analysis model in accordance with the system design strategy. His focus is the data structure & algorithms needed to implement each class

* Implementation : Implementers translate the class & relationships developed during class design into a particular programming language, database or hardware. During implementation, it is important to follow good software engineering practice

The 3 models

We use 3 kinds of models to describe a system from different view points.

1) **Class Model** : represents the static, structural, "data" aspects of a system.

* It describes the structure of objects in a system their identity, their relationships to other objects, their attributes & their operations
* Goal in constructing class model is to capture those concepts from the real world that are important to an application.
* Class diagrams express the class model

2) **State Model** : represents the temporal, behavioral, "control" aspects of a system.

* It describes those aspects of objects concerned with time & the sequencing of operations - events that mark changes, states that define the context for events & the organization of events & states.
* State diagram express the state model
* Actions & events in a state diagram become operations on objects in the class model. References betn state diagrams become interactions in the interaction model.

3) **Interaction model** : represents the collaboration of individual objects, the "interaction" aspects of a system.

* Interaction model describes interactions between objects how individual objects collaborate to achieve the behavior of the system as a whole.
* The state & interaction models describe different aspects of behavior & you need both to describe behavior fully.
* use cases, sequence diagrams & activity diagrams document the interaction model

Several Themes pervade OO technology. Few are

1) Abstraction

* Abstraction lets you focus on essential aspects of an application while ignoring details i.e focusing on what an object is & does before deciding how to implement it.
* It's the most important skill required for object oriented development

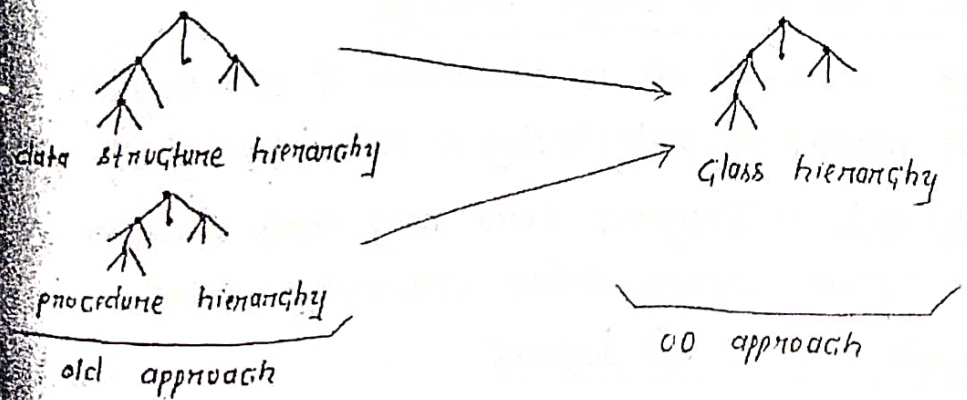
2) Encapsulation (information hiding)

* It separates the external aspects of an object (that are accessible to other objects) from the internal implementation details (that are hidden from other objects)
* Encapsulation prevents portions of a program from becoming so interdependent that a small change has massive ripple effect.

3) Combining data & behavior

Callers of an operation need not consider how many implementations exist

In OO system the data structure hierarchy matches the operation inheritance



4) Sharing

OO techniques provide sharing at different levels

Inheritance of both data structure & behavior lets sub classes share common code

OO development not only lets you share information within an application, but also offers the prospect of reusing designs & code on future projects.

5> Emphasis on the essence of an object

* OO development places a greater emphasis on data structure & a lesser emphasis on procedure structure than functional-decomposition methodologies

6> Synergy

* Identity, classification, polymorphism & inheritance characterise OO languages.

Each of these concepts can be used in isolation, but together they complement each other synergistically.

Object Oriented Modeling History

* 1991 GE R&D led to object Modeling Technique (OMT)
* 1994 Jim Rambaugh joined IBM Rational & He started working with Grady Booch. Ivan Jacobson also joined with both.
* 1996 Object Management Group (OMG) issued request for proposals
* 1997 OMG Accepted the Unified Modeling Language (UML)

Modelling as a Design Technique

Note: A model is an abstraction of something for the purpose of understanding it before building it.

MODELING - Designers build many kinds of models for various purposes before constructing things

Models serve several purposes

* Testing a physical entity before building it:

Medieval built scale models of Gothic cathedrals to test the forces on the structures. Engineers test scale models of airplanes, cars & boats to improve their dynamics.

- Communication with Customers

Architects & product designers build models to show their customers.

- Visualization : Storyboards of movies, TV shows & advertisements let writers see how their ideas flow

- Reduction of Complexity

Models reduce complexity to understand directly by separating out a small no. of important things to do with at a time.

- Abstraction is the selective examination of certain aspects of a problem.

The goal of abstraction is to isolate those aspects that are important for some purpose & suppress those aspects that are unimportant.

CLASS MODELING

Note: A class model captures the static structure of a system by characterizing the objects in the system, the relationships betn the objects & the attributes & operations for each class of objects.

Object & Class Concept

Objects: purpose of class modeling is to describe objects

* An object is a concept, abstraction or thing with identity that has meaning for an application.

Ex: Joe Smith, Infosys Company

* An object has 3 characteristics: state, behavior & a unique identification.

Classes

* An object is an instance or occurrence of a class
* A class describes a group of objects with the same properties (attributes), behavior (operations), kinds of relationships & semantics.

Ex: Person, Company

Class diagrams

* Class diagrams provide a graphic notation for modeling classes & their relationships, thereby describing possible objects

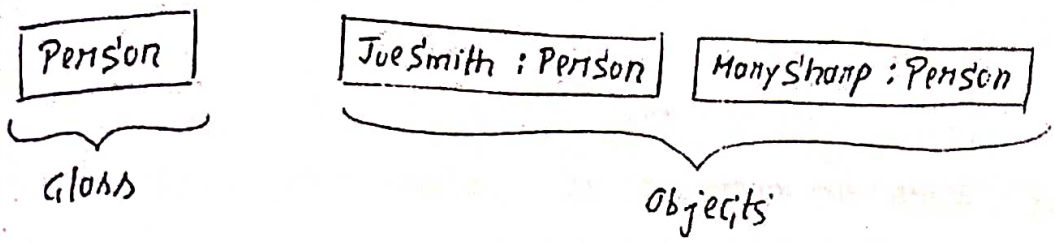
Object diagrams

* An object diagram shows individual objects & their relationships - useful for documenting test cases & discussing examples

* Class diagrams are useful both for abstract modeling & for designing actual programs

Note: A Class diagram Corresponds to infinite set of object diagrams

Figure below shows a Class & instances described by it



Conventions used (UML)

- UML Symbol for both Classes & Objects is box
- Objects are modeled using box with object name followed by Colon followed by Class name.
- Use boldface to list Class name. Center the name in the box & Capitalize the first letter. Use singular nouns for names of classes
- To run together multiword names (such as JoeSmith) separate the words with intervening Capital letter

Values, Attributes & Operation

- Value is a piece of data
- Attribute is a named property of a Class that describes a value held by each object of the Class

Conventions used (UML)

- List attributes in the 2nd Compartment of the Class box. Optional details (like default value) may follow each attribute.
- A Colon precedes the type, an equal sign precedes default value.
- Show attribute name in regular face, left align the name in the box & use small case for the first letter.

Note: Do not list object identifiers (Ex: personID : ID) They are implicit in models.

* An operation is a function or procedure that maybe applied to or by objects in a class.
* A method is the implementation of an operation for a class

Note: Some operation may apply to many different classes. Such an operation is polymorphic

Conventions used (UML)

* List Operations in 3rd compartment of class box.
* List Operation name in regular face, left align & use lower case for first letter.
* Optional details like argument list & return type may follow each operation name.
* Parenthesis enclose an argument list, commas separate the arguments. A colon precedes the result type

Note: We do not list operations for objects, because they do not vary among objects of same class

Class Example

Customer
name : String Addrss : String type : String Cust_ph : int b_date : Date
register() Update() Cancel()

Object Example

Ramesh : Customer
name = "Ramesh" Addrss : "Plot" Type : "regular" Cust_ph : 9901375126 b_date : 12/05/1978
register() Update() Cancel()

Summary of Notation for Classes

ClassName
attributeName1 : Datatype1 = value1 attributeName2 : Datatype2 = value2 attributeName3 : Datatype3 = value3
operationName1 (argumentList1) : resultType1 operationName2 (argumentList2) : resultType2 operationName3 (argumentList3) : resultType3

Link & Association Concepts

- A Link is a physical or conceptual connection among objects.
- Mathematically, we define a link as a tuple - that is, a list of objects.
- A link is an instance of an association.

Ex: Joe Smith WorksFor Simplex Company

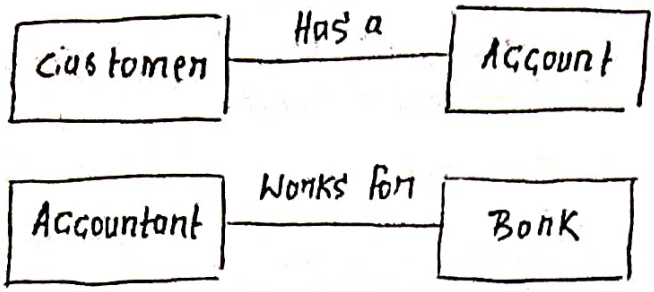
- An association is a description of a group of links with common structure & common semantics
- An association describes a set of potential links in the same way that a class describes a set of potential objects

Ex: a person WorksFor a Company

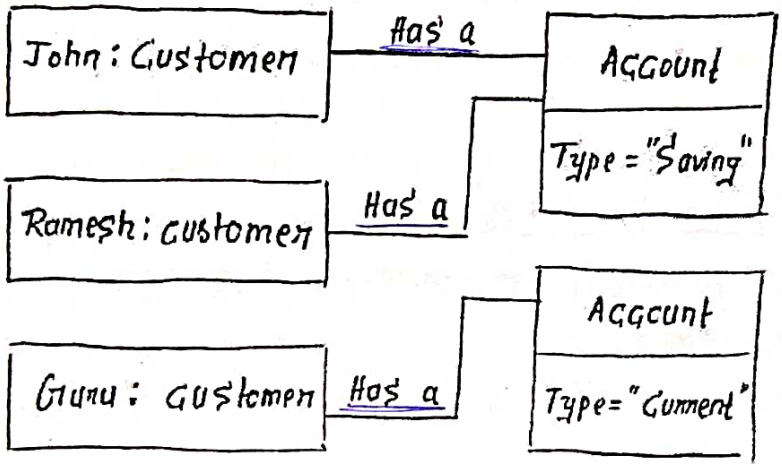
Conventions used (UML)

- link is a line between objects; a line may consist of several line segments
- If the link has the name, it is underlined
- Association connects related classes & it also denoted by a line
- Show link & association names in italics

Association Example



Link Example



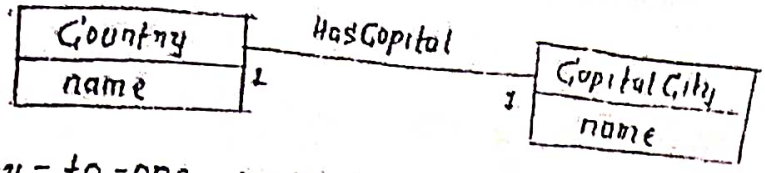
Multiplicity

* Multiplicity specifies the number of instances of one class that may relate to a single instance of an associated class.
* Multiplicity constrains the number of related objects

UML Conventions

* UML diagrams explicitly lists multiplicity at the ends of association lines.
* UML specifies multiplicity with an interval such as
  "1" (exactly one)
  "1.." (one or more)
  "3..5" (three to five, inclusive)
  "*" (many ; i.e zero or more)

one-to-one multiplicity

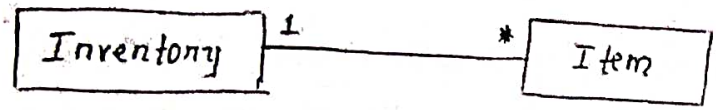


many-to-one multiplicity

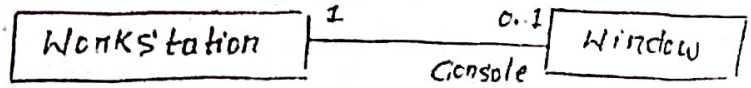
Bank has many ATMs, ATM knows only 1 bank

one-to-many multiplicity

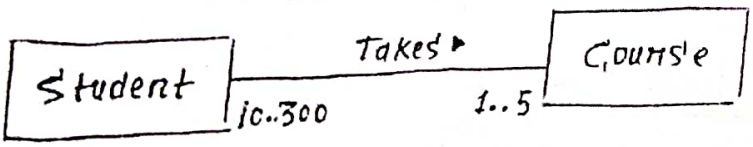
Inventory has many items, items know 1 inventory



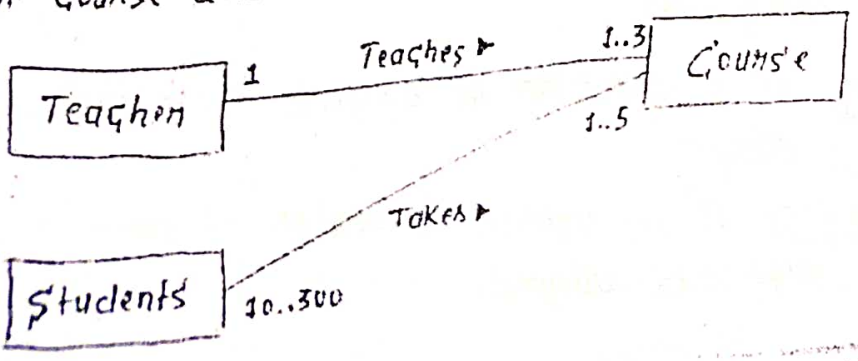
zero-on-one multiplicity



Ex1: * A student can take up to 5 courses
* Student has to be enrolled in atleast one course
* upto 300 students can enroll in a course
* A class should have at least 10 students



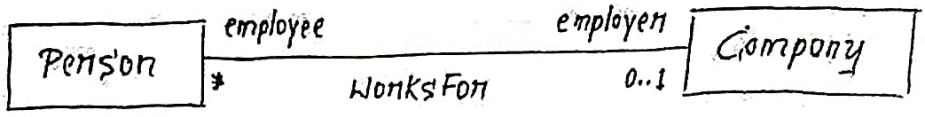
Ex2: * A teacher teaches 1 to 3 courses
* Each course is taught by only one teacher
* A student can take between 1 to 5 courses
* A course can have 10 to 300 students



Association end names

Multiplicity implicitly refers to the ends of Associations.
For Ex: A one-to-many association has 2 ends
→ an end with a multiplicity of "one"
→ an end with a multiplicity of "many"

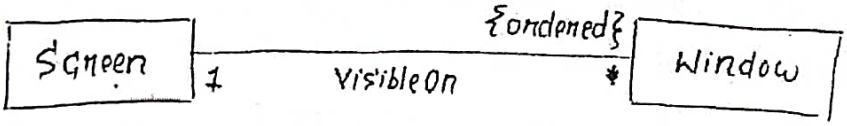
We cannot only assign a multiplicity to an association end, but we can give it a name as well



→ A person is an employee with respect to Company
→ A company is an employer with respect to a person

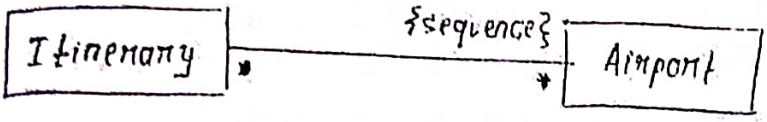
Ordering

* Ordering is an inherent part of association. we can indicate an ordered set of objects by writing "{ordered}" next to the appropriate association end.
* Ordering is permitted only for binary associations



Bags & Sequences

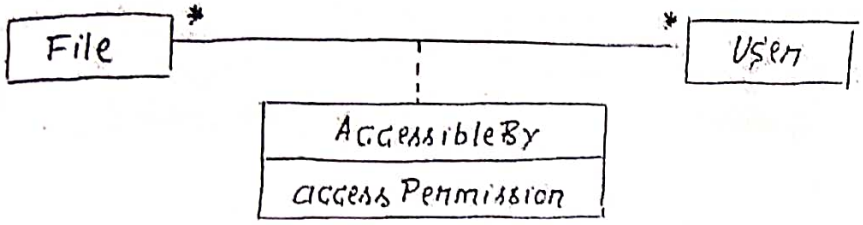
* Normally, a binary association has at most one link for a pair of objects
* However, we can permit multiple links for a pair of objects by annotating an association end with {bag} or {sequence}
* A bag is a collection of elements with duplicates allowed
* A sequence is an ordered collection of elements with duplicates allowed.



An itinenary may visit multiple airports, so we should use {sequence} & not {ordered}

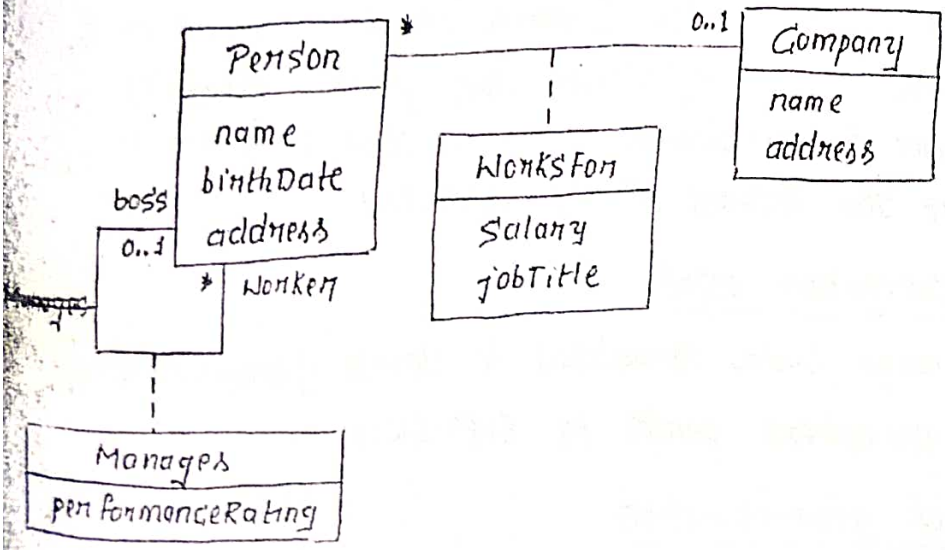
Association Class

- An association class is an association that is also a class
- Like a class, an association class can have attributes & operations & participate in associations



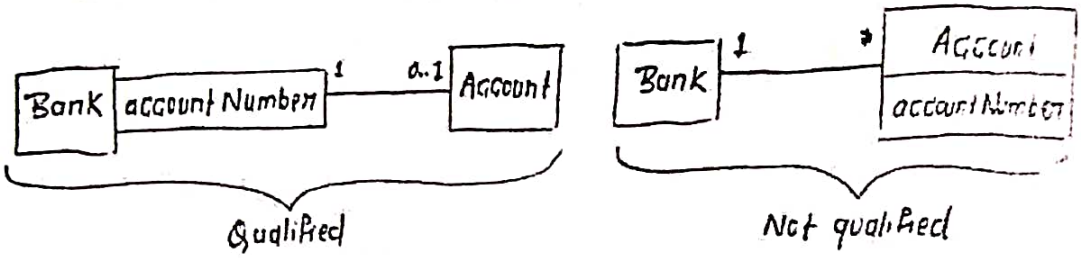
- UML notation for association class is a box attached to the associated by a dashed line.

Ex: Each person working for a company receives a salary & has job title. The boss evaluates the performance of each worker.



Qualified Associations

* A Qualified Association is an association in which an attribute called the qualifier disambiguates the objects for a 'many' association ends.
* It is possible to define qualifiers for one-to-many & many-to-many associations
* A qualifier selects among the target objects, reducing the effective multiplicity from "many" to "one"



Generalization & Inheritance

* Generalization is the relationship between a class (the superclass) & one or more variations of the class (the subclasses).
* Generalization organizes classes by their similarities & differences, structuring the description of objects
* The superclass holds common attributes, operations & associations; the subclasses add specific attributes, operations & associations. Each subclass is said to inherit the features of its superclass

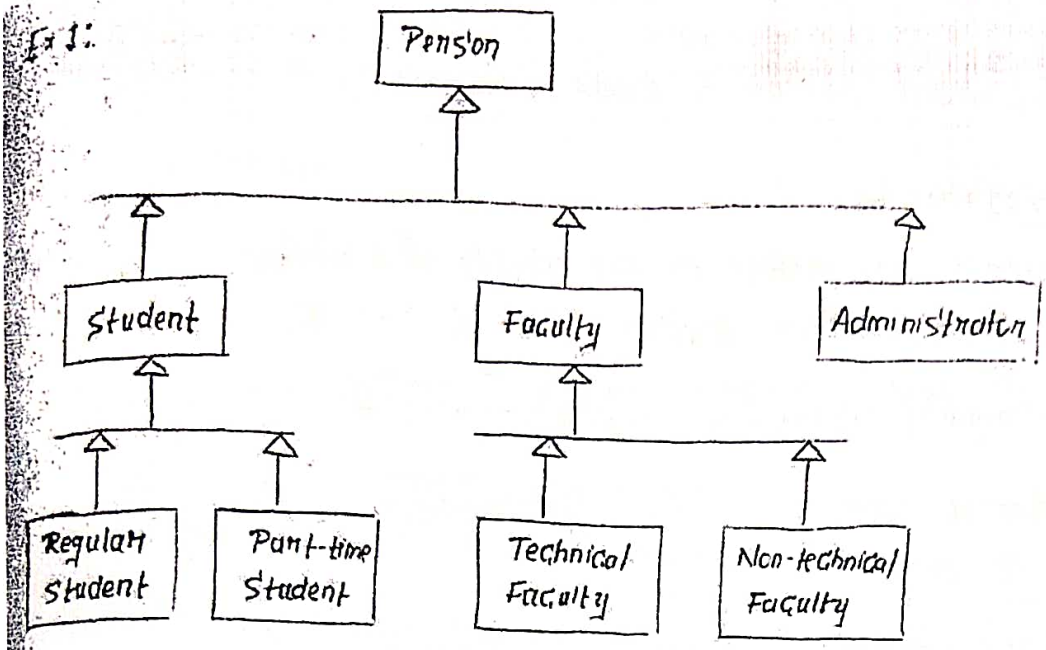
UML Convention used

* Use large hollow arrowhead to denote generalization. The arrowhead points to superclass

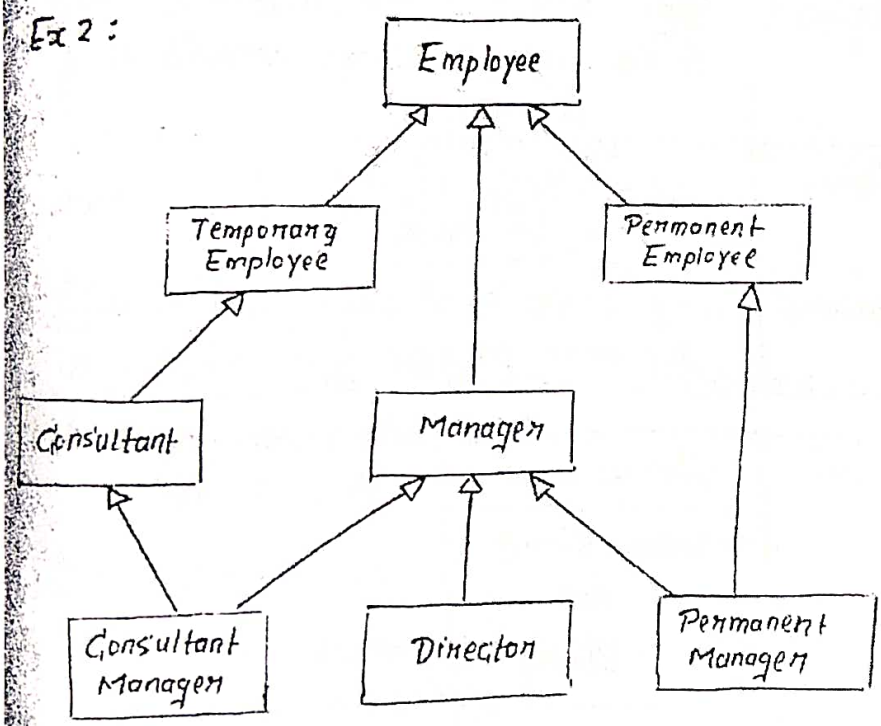
Uses of generalization

1) To support polymorphism
2) To structure the description of objects
3) To enable reuse of code

Ex 1:



Ex 2:

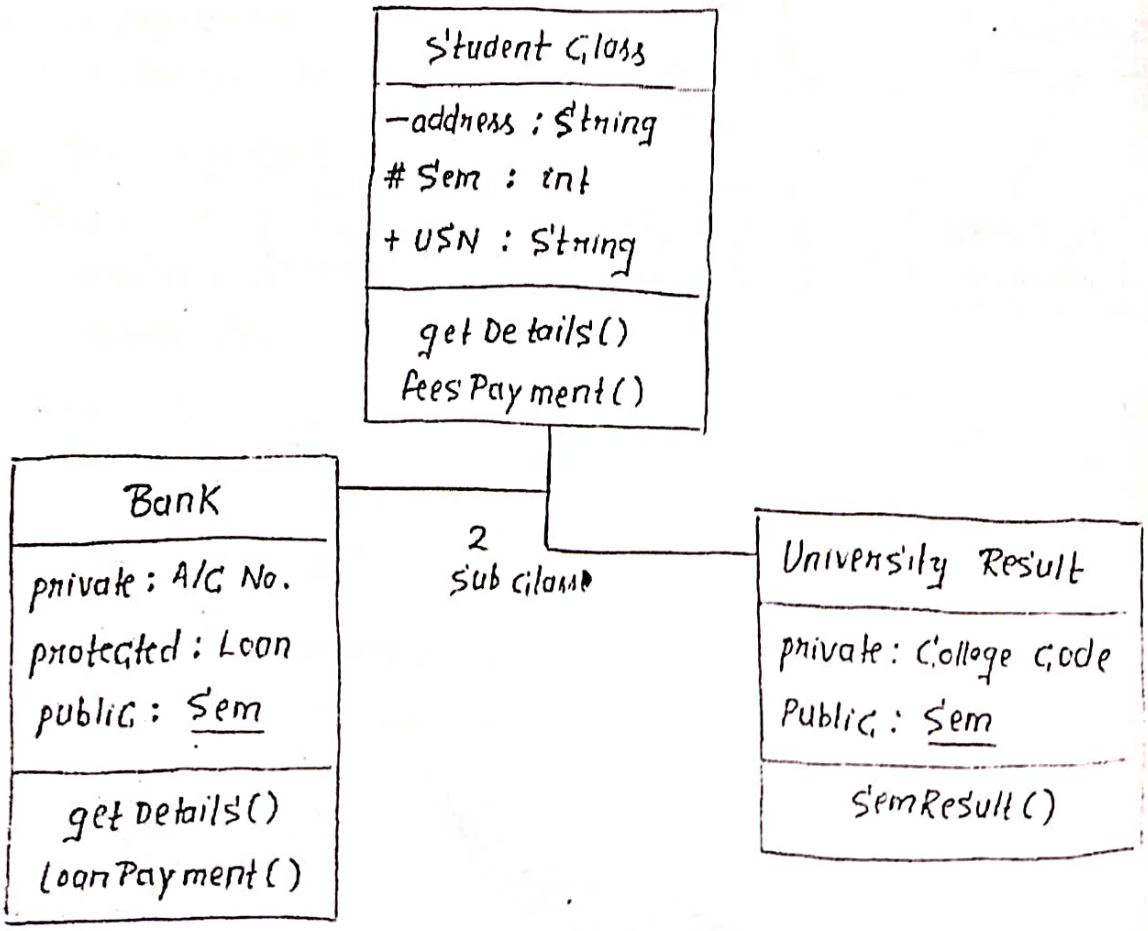


Advanced Object & Class Concepts

Visibility

* Visibility refers to the ability of a method to reference a feature from another class & has the possible values of public, protected, private & package.

Access specifiers features	Descriptions
Public features	Any method can access
Protected features	Only methods of the containing class & its descendants via inheritance can access
Private features	Only methods of the containing class can access
Package features	Only methods of classes defined in the same package can access



UML Convention used

"+" → public

"-" → private

"#" → protected

"~" → package

Several issues to consider when choosing visibility are

- Comprehension : understand all public features to understand the capabilities of a class
- Extensibility : many classes can depend on public methods, so it can be highly disruptive to change them. Therefore changes can only be made in only private, protected & package methods
- Context : private methods may calculate incorrect results or cause the object to fail.

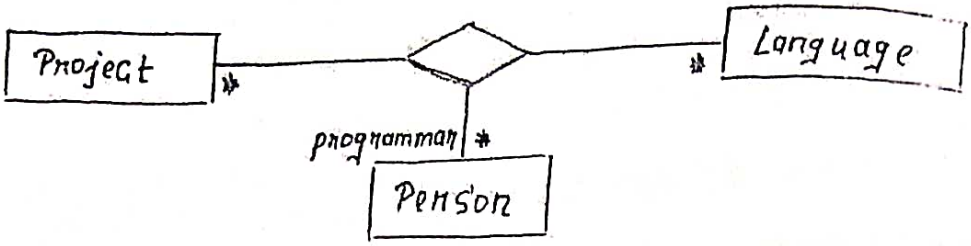
Association Ends

- Association End is an end of association
- A binary association has 2 ends ; a ternary association has 3 ends

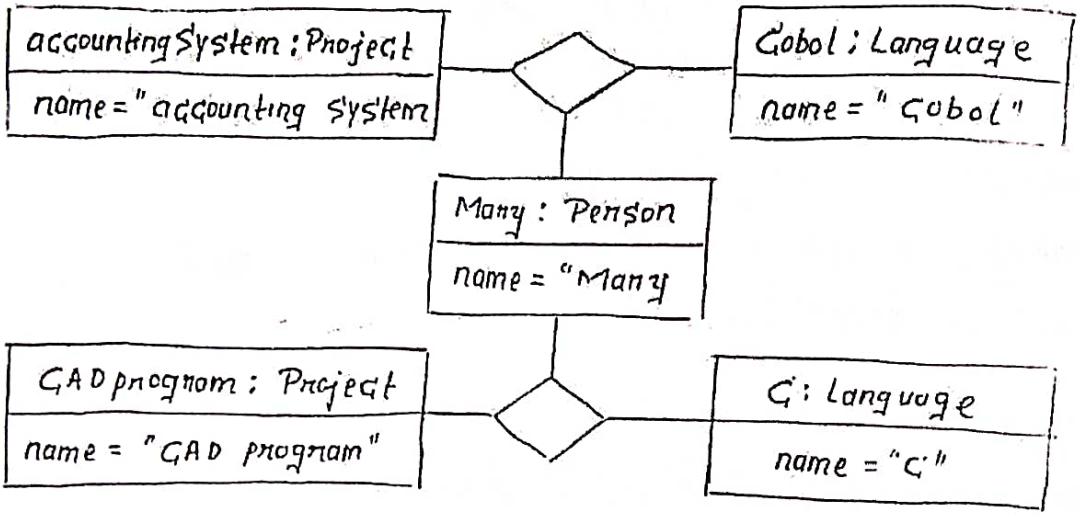
N-ary Association

- Association between 3 or more classes are known as N-ary association
- We should try to avoid n-ary associations - most of them can be decomposed into binary associations, with possible qualifiers & attributes.
- The UML symbol for n-ary associations is a diamond with lines connecting to related classes. If the association has a names, it is written in italics next to the diamond.
- An n-ary association enforces that there is atmost one link for each combination

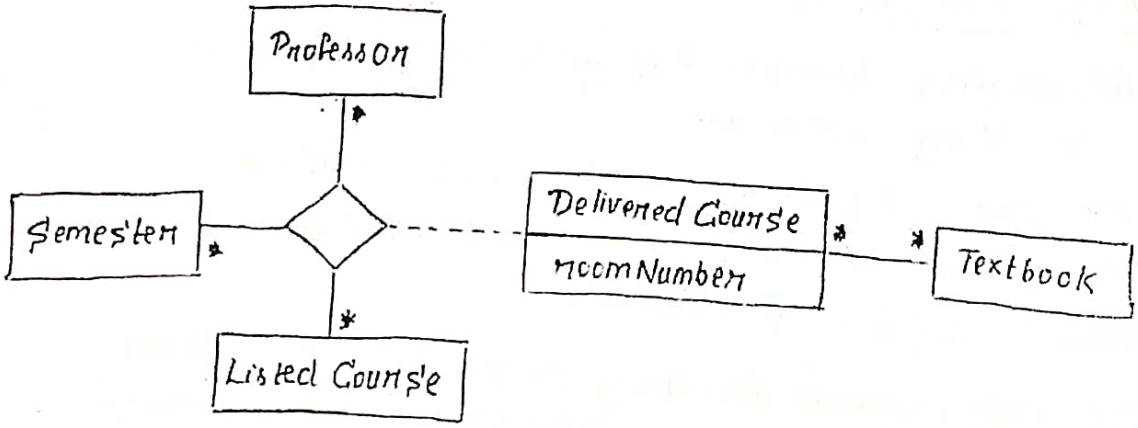
Class diagram



Instance diagram



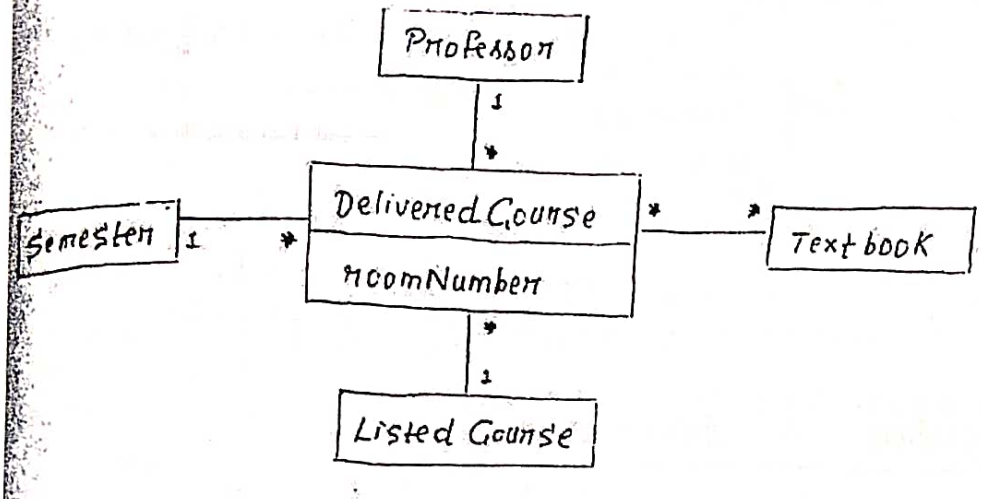
Ex: Figure below shows another ternary association
A professor teaches a listed course during a semester



* N-ary associations are fully-fledged associations & can have association classes
* The typical programming language cannot express n-ary associations

* Thus during programming, n-ary associations must be promoted to classes as the figure below does for DeliveredCourse

Promoting an n-ary association



Aggregation

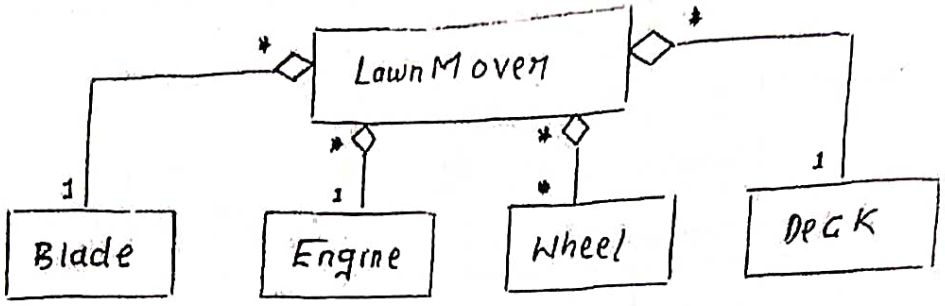
* Aggregation is a strong form of association in which an aggregate object is made of constituent parts.
* Constituents are the parts of aggregate.
* The aggregate is semantically an extended object that is treated as a unit in many operations, although physically it is made of several lesser objects
* We define an aggregation as relating an assembly class to one constituent part class.
* We define each individual pairing as an aggregation so that we can specify the multiplicity of each constituent part within the assembly. This definition emphasizes that aggregation is a special form of binary association.

Properties of aggregation

1) Transitivity – if A is a part of B & B is part of C, then A is part of C.
2) Antisymmetric – if A is part of B then B is not part of A

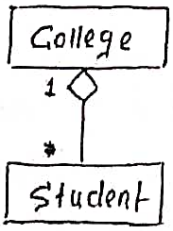
Ex: LawnMoven consists of a Blade, an Engine, many wheels & a Deck

LawnMoven is the assembly & the other parts are constituents

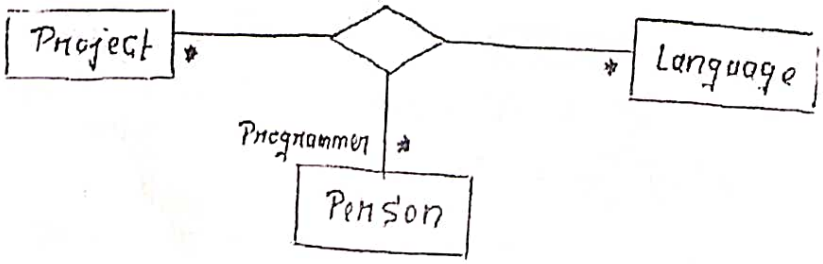


Aggregation V/S Association

* Aggregation is a special form of association, not an independent concept
* Aggregation adds semantic connotations
* If 2 objects are tightly bound by a part-whole relationship, it is an aggregation.

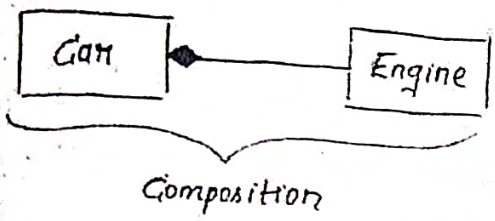


* If the 2 objects are usually considered as independent even though they may often be linked, it is an association
* Aggregation is drawn like association, except a small (hollow) diamond indicates the assembly end



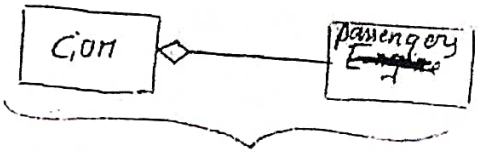
Aggregation v/s Composition

- The UML has 2 forms of part-whole relationships
  - A general form called aggregation
  - A more restrictive form called composition
- Composition is a form of aggregation with 2 additional constraints
  1) A constitute part can belong to atmost one assembly
  2) Once a constitute part has been assigned an assembly, it has a coincident lifetime with the assembly. Thus composition implies ownership of the parts by the whole
- Notation for composition is a small solid diamond next to the assembly class



Composition

Every car has an ~~engine~~ engine

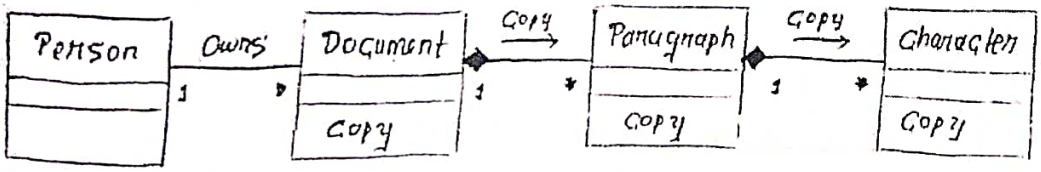


Aggregation

Cars may have passengers, they come & go

Propagation of Operations

- Propagation (also called triggering) is the automatic application of an operation to a network of objects when the operation is applied to some starting object



- Operations can be propagated across aggregations & compositions
- propagation can be shown with a small arrow indicating the direction & operation name next to the affected association
- The notation binds propagation behavior to an association (or aggregation), direction & operation