Information System Design
IT60105

Lecture 6

Object-Oriented Design Paradigms

Lecture #5

- **Concepts of objects**
- **Object-Oriented Paradigms**
  - Class
    - Encapsulation
  - Relation between classes
    - Association
    - Aggregation
  - Classifications
    - Inheritance
    - Delegation
  - Polymorphism
    - Dynamic Binding

Concepts of Objects

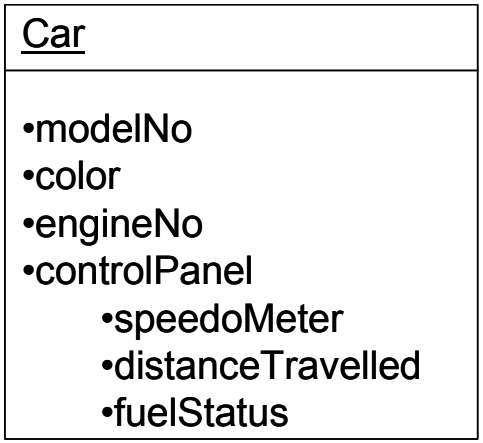
Fundamental Characteristics of Objects

- Object = **State** + Behavior + Identity

- **State**

  The group of values of all attributes at a given point of time

  e.g. Car



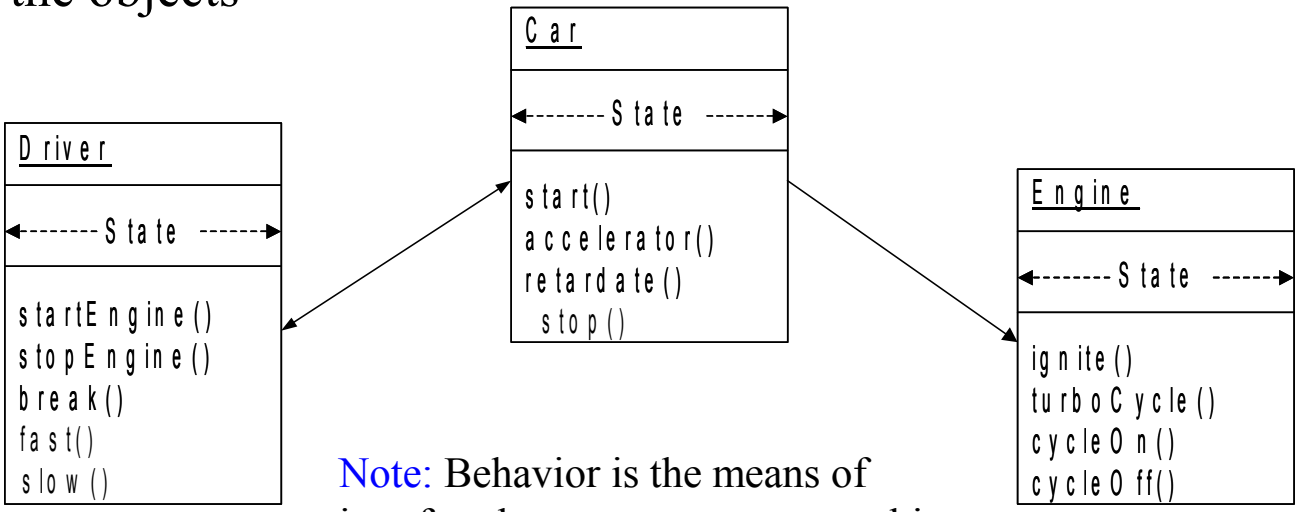
Note:
Some state attributes are constant Where as some are dynamically change
e.g .distanceTravelled, fuelStatus

Fundamental Characteristics of Objects

- Object = State + **Behavior** + Identity

- **Behavior**

  The group of all abilities of an object and describes the action and reaction among the objects



Note: Behavior is the means of interface between two or more objects.

Fundamental Characteristics of Objects

- Object = State + Behavior + **Identity**

- **Identity**

  The characterization of its existence. The identity makes it possible to distinguish an object in an unambiguous way and independently form its group

  e.g.

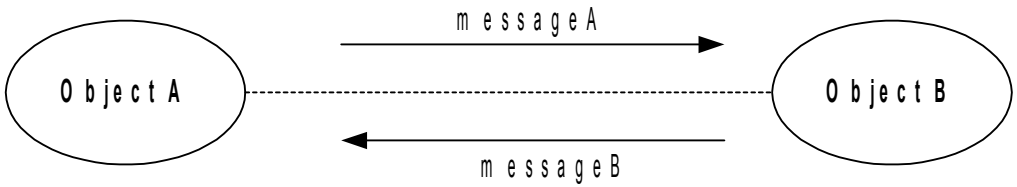
  Car – carNo

  Borrower – borrowerNo

  Person – empId / voterId / rollNo

Fundamental Characteristics of Objects

- Communication between two objects



- Type of objects
  - Active objects
  - Passive objects
  - Transient/ephemeral objects
  - Persistent objects

- Other objects categorization

Type of Objects

- Active objects
  - An object is an active object if it is capable to send a message to another object

    Example:
    - All clients are like active objects
- Passive objects
  - An object that is not capable of sending a message to any other objects

    Example:
    - All servers are like passive objects (however, they can reply to any message)
- Transient objects
  - When an object constantly changes its state

    Example:
    - Car is in motion
- Persistent objects
  - Storing the state of objects to a permanent storage
  - Stores attributes of an object into a permanent storage before leaving sessions

Other Type of Objects

- Entity objects
  - Objects which are related to some entities

    Example:
    - Customer, order, book, transaction etc.
- Controller Objects
  - Objects which control the communication of several objects

    Example:
    - Registration Controller, Scheduler, ATM System etc.
- Boundary objects
  - These are the objects interfaced with the system or sub-system

    Example:
    - Database wrapper, external system etc.
- Interface objects
  - Act as an interface between a customer and system

    Example:
    - Registration Screen, login screen etc.

Object-Oriented Paradigms

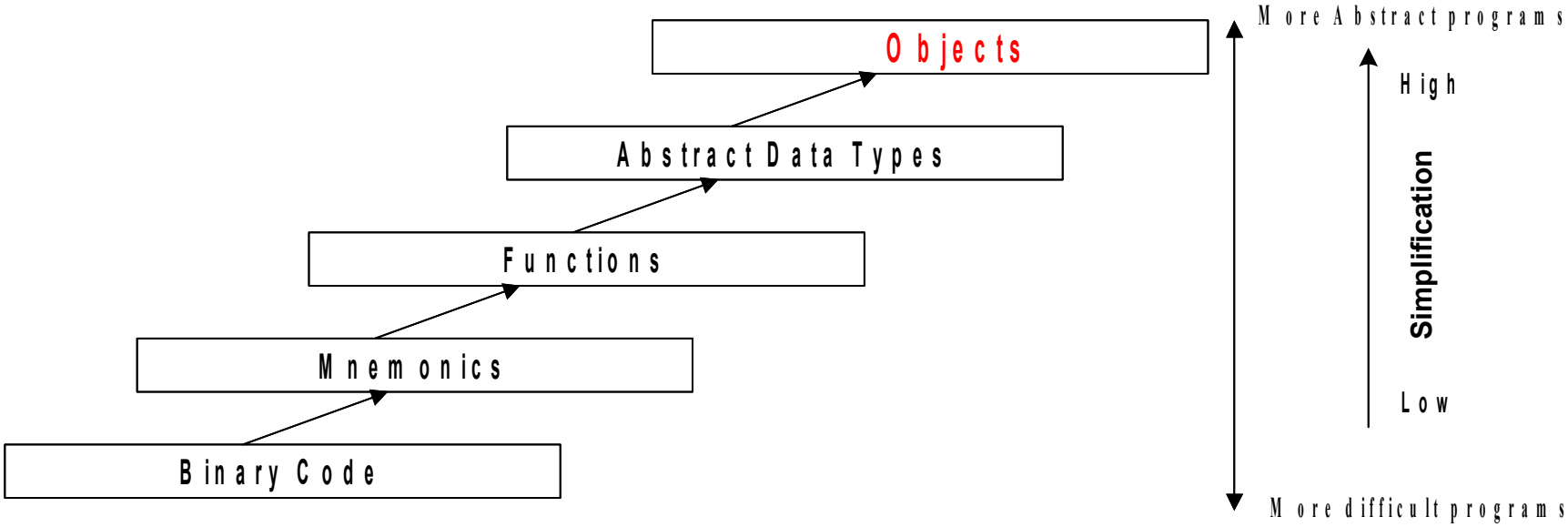
Aim of Object-Oriented Paradigms

- Real world is composed of very large number of interacting objects

- Objects are abstract things than the way computer process them
  - Easy to understand
  - Easy to manipulate
  - Processed at higher level

- Paradigm based on the concept of object reduces the gap between our way of reasoning (by abstraction) and the concept understood by computers

Aim of Object-Oriented Paradigms



Aim of Object-Oriented Paradigms

- **Challenges**
  - Real world objects are too complicated to manipulate
- **Solution**
  - To reduce inherent complexity object oriented paradigm have been formulated

Object-Oriented Paradigms

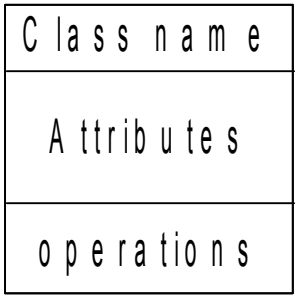
- Encapsulation
- Relation
- Classification
- Polymorphism

O-O Paradigms: Encapsulation

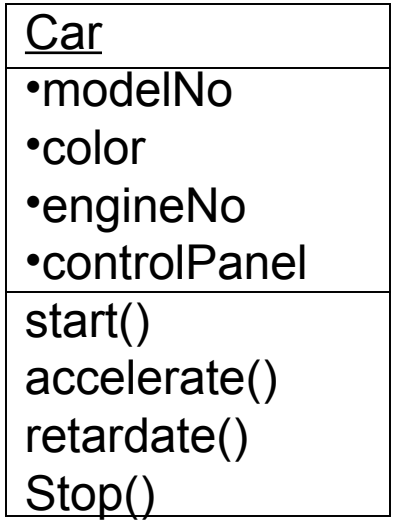
- **Encapsulations**
  - Combined data and methods to manipulate data into one entity
- **Class is the concept of encapsulation**
  - A varieties of objects are grouped together such as book, car, ..etc
  - A set of similar objects grouped together with some distinguishing structure (at a high level of abstraction)
  - Class is a process of abstraction

O-O Paradigms: Encapsulation

- Graphical representation of a class



Example: **Car**



O-O Paradigms: Encapsulation

Exercise: Give possible class structure to the following

- Set
- Complex number
- Rational number
- TV set
- Customer (of a Bank)
- Account (of a customer in a Bank)
- Stack/ Queue/Tree/Graph/Vector

Note: Visibility of attributes and operations

Public	+
Private	-
Protect	#

O-O Paradigms: Relations

- Several classes may be interrelated with each other
- A relationship expresses a kind of interrelation between two classes
- Two types of relationships between any two classes
  - Association
  - Aggregation

Relation: Association

- The association relationship expresses a bi-directional, semantic connection between classes

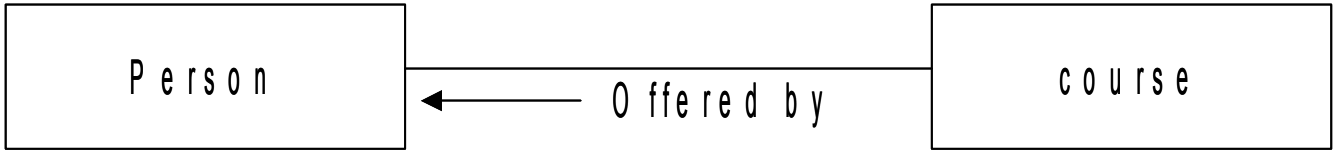
- Bi-directional
  - Data may flow across the association (This is unlike data flow in DFD)

- Semantic connection
  - An association between classes means that there is a link between objects in the associated class

- If an association between two objects are there then one can navigate from an object to another object in the association

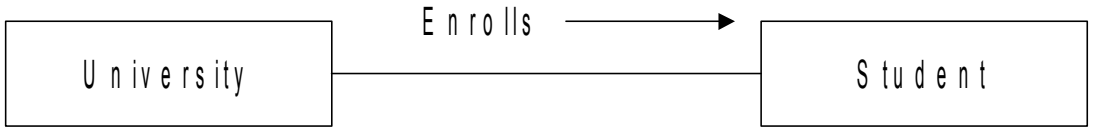
Relation: Association

- Example



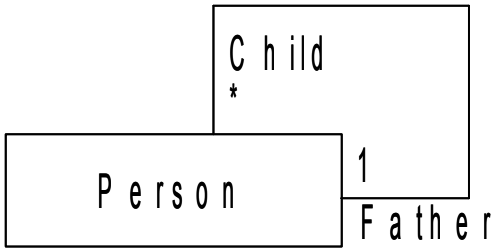
Association: Notation

Example:



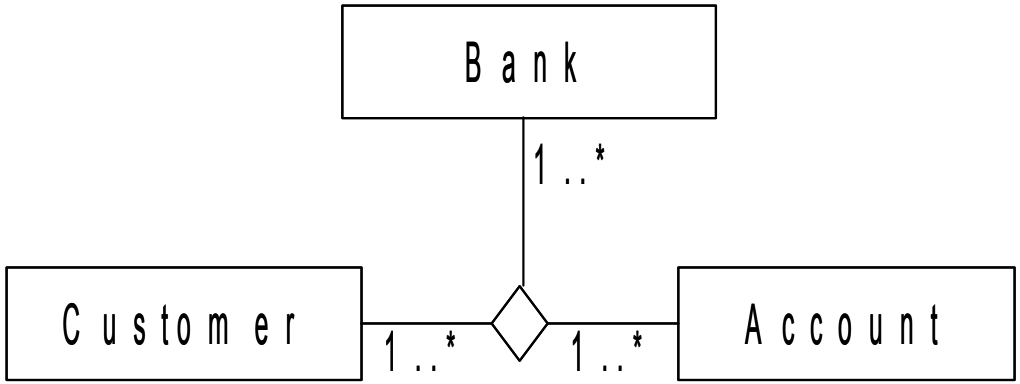
Association: Notation

- Association between two classes, in general, is called binary association
  - It is also legal to have both ends of an association circle to back to the same class



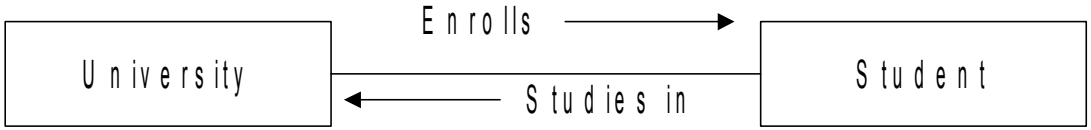
Association: Notation

- *n-ary* association is also possible
  - An association having more than two classes also possible

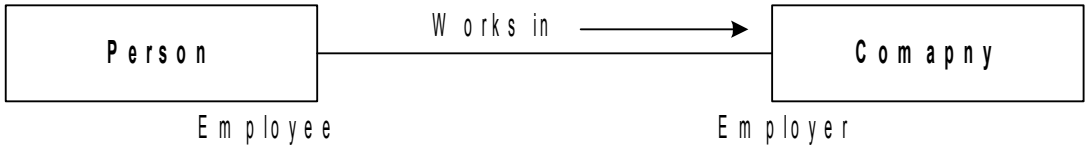


Association: Notation

- More than one association may be mentioned between two classes

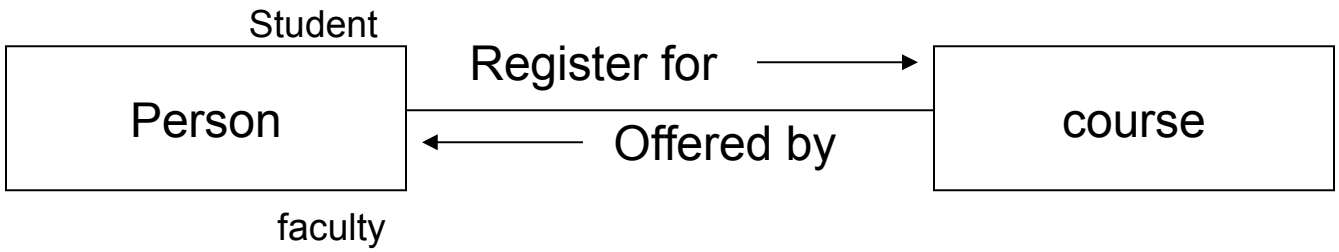


- It is possible to specify the role of a class within an association

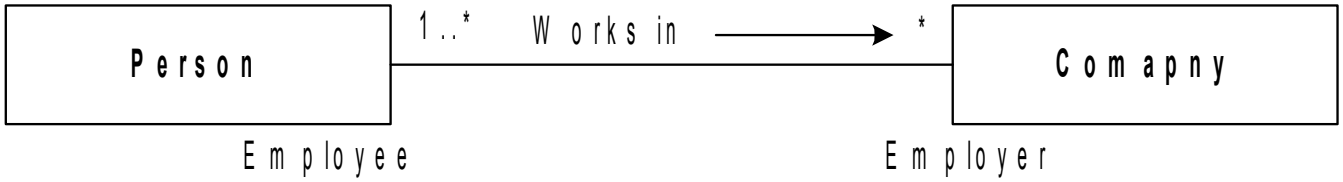


Association: Notation

- A class can play more than one role at a time



- Role also carry multiplicity information that specifies the number of instances that participate in a relationship



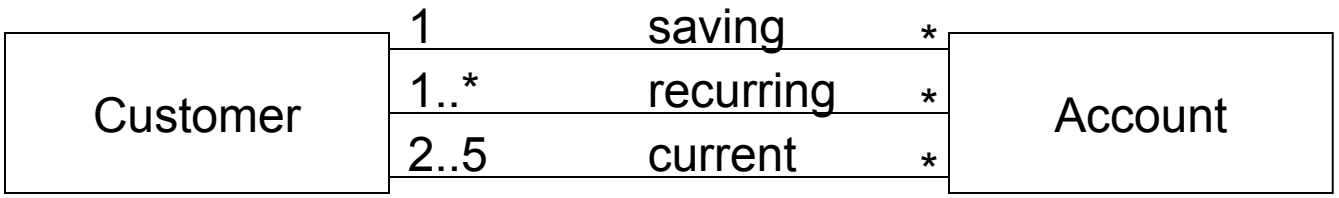
Association: Notation

Multiplicity Rules

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural integers)
*	From zero to any positive integers
0..*	From zero to any positive integers
1..*	From one to any positive integers

Example: Multiplicity Rule

- Customer may have more than one account
- An account may be jointly hold by more than one customer
- There are may be three types of accounts
  - Saving (single customer only)
  - Recurring (single or joint)
  - Current (joint only, but allow at most 5 customers)



Relation: Aggregation

- Aggregation allows the representation of "whole/part" relationship
  - Whole-part relationship: one represents a large thing (the "whole"), which consists of smaller things (the "parts")

Example

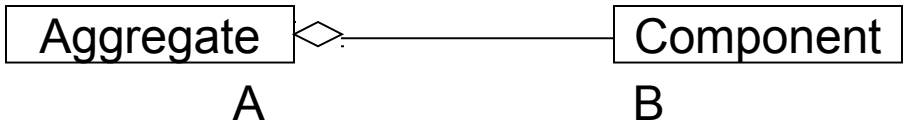
Engine is a part of Car

- Also, it expresses "has a" relationship

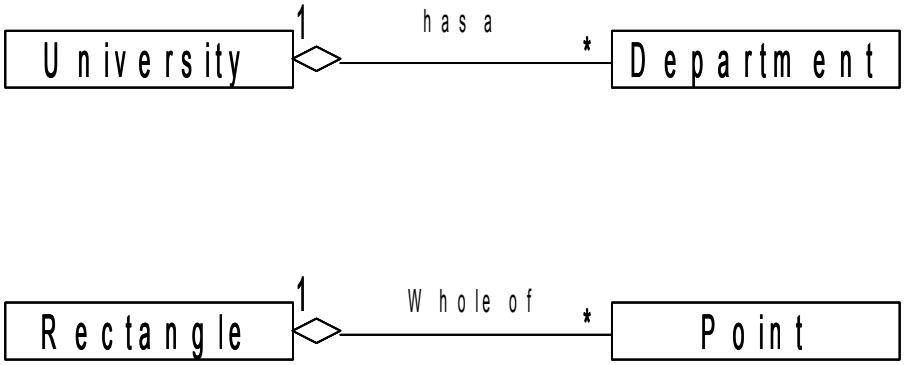
Example

University has a number of department

Aggregation: Notation



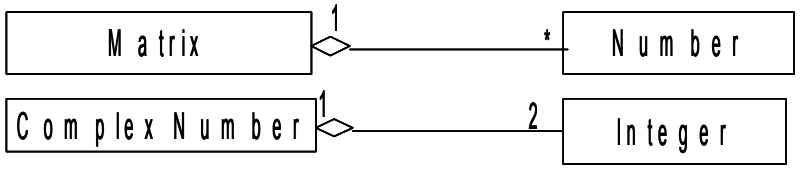
- Meaning: B is being aggregated into A



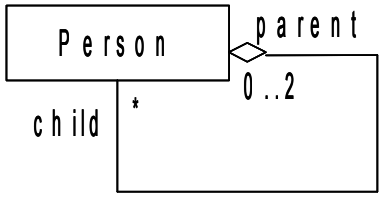
Aggregation: Notation

- Aggregation represents one-to-many as well as many-to-many relations

Examples of One to Many:

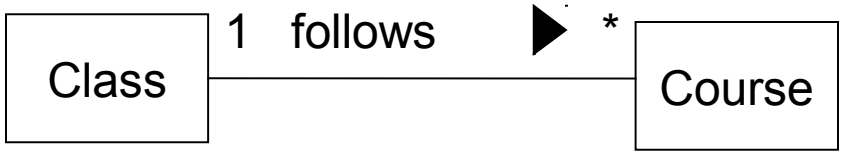


Example of Many to Many:

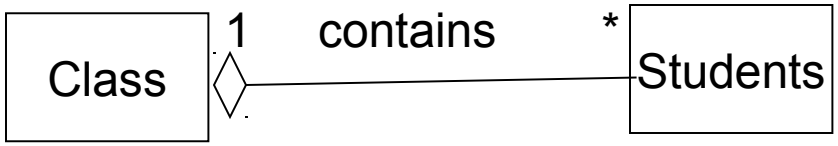


Association vs. Aggregation

- Association represents structural relationships between peers, meaning that both classes are at same level

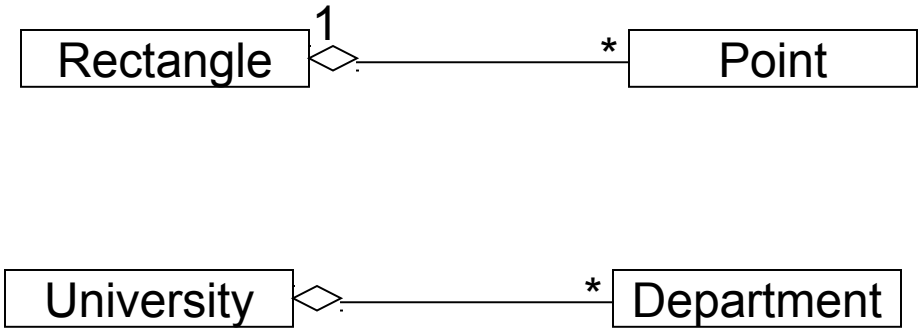


- Aggregation represents a “master and slave” relationship between two classes



Association vs. Aggregation

- The following tests may be used to determine if a relation is an association or an aggregation
  - Is the phase "part" describe the relationship?



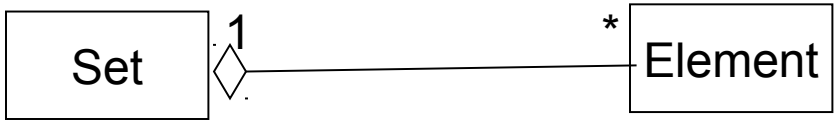
Association vs. Aggregation

- The following tests may be used to determine if a relation is an association or an aggregation
  - Are some operations on the whole automatically applied to its parts?

If delete a class then all of its student also deleted

Association vs. Aggregation

- The following tests may be used to determine if a relation is an association or an aggregation
  - Is there any intrinsic asymmetry to the relationship where one classis subordinate to the other?



Association vs. Aggregation

- Truly, Aggregation is a special kind of Association
  - Association: is-a relationship with weak coupling

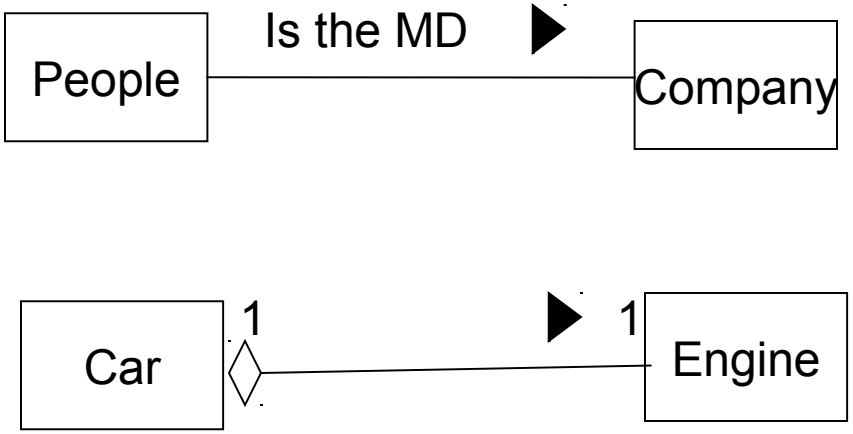
    Aggregation: is-a relationship with strong coupling
  - Association: bi-directional and symmetric connection between classes

    Aggregation: bi-directional and asymmetric connection between classes

Association vs. Aggregation

- A good heuristic test for whether a relationship is an aggregation is to ask

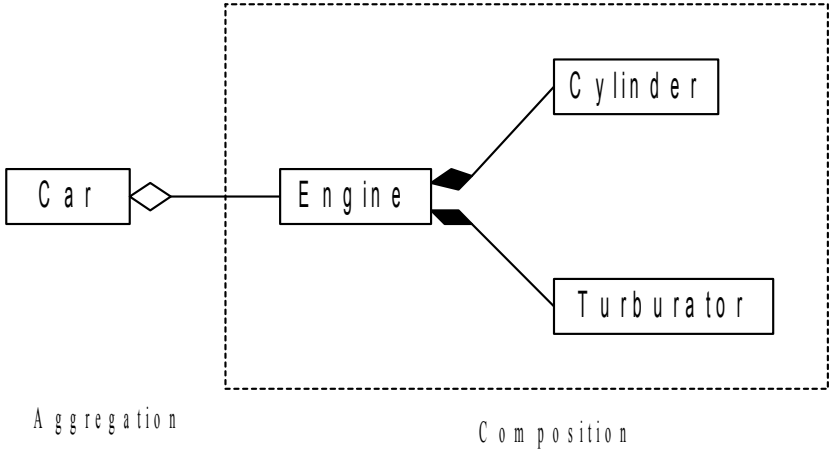
If the part moves, can one deduce that the whole moves with it?



Composition

- Composition is a special case of aggregation
  - Attributes are particular case of aggregation
  - Attributes are physically contained in the aggregate

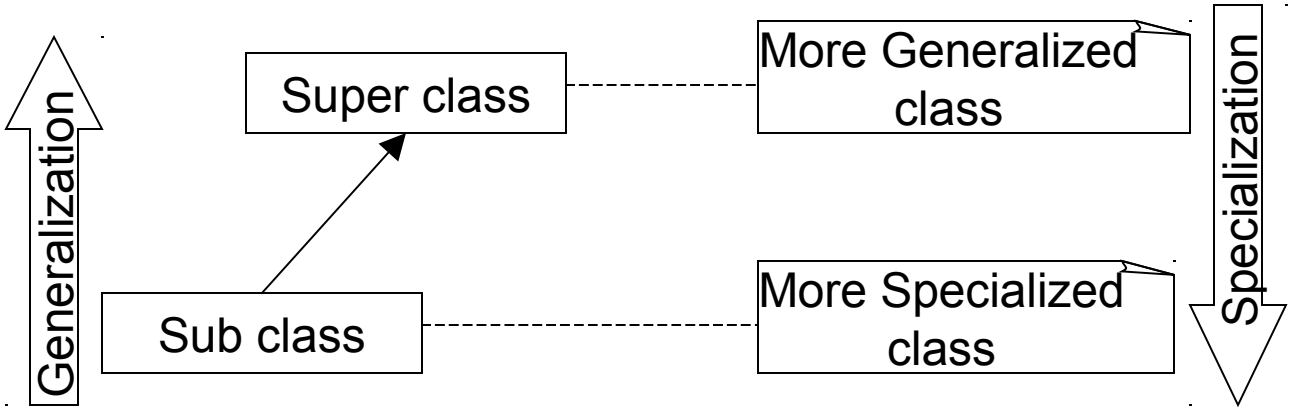
Example:



O-O Paradigms: Classification

- Class hierarchies (or classification) makes it possible to manage complexity by ordering the objects within trees or classes, with increase level of abstraction

- Generalization and specialization are the two point of views that are based on class hierarchy



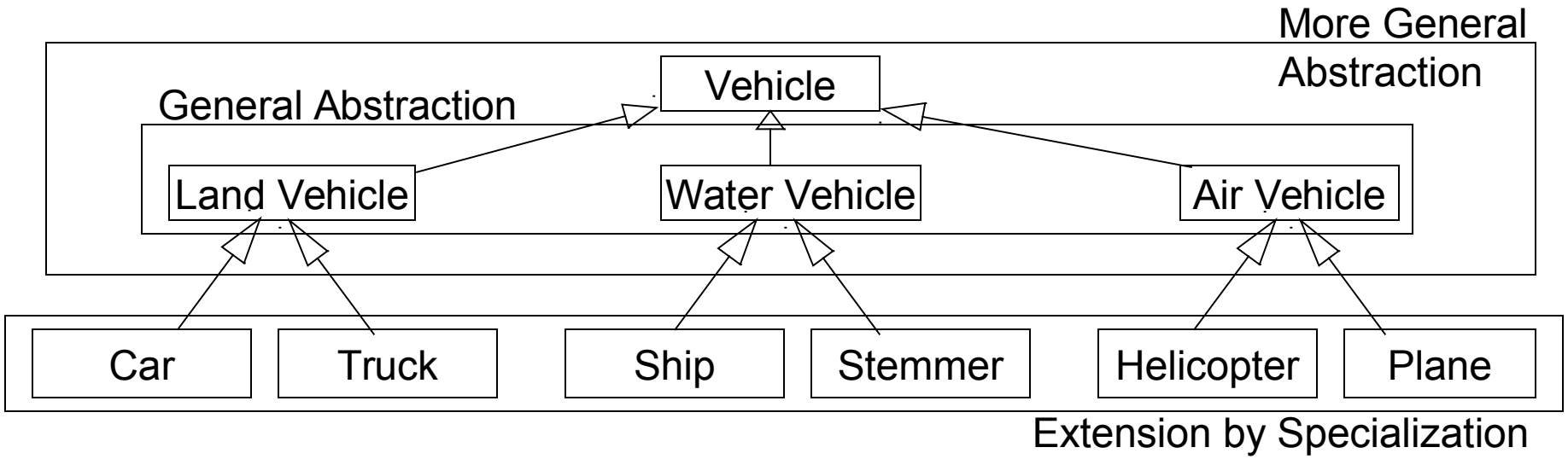
Classification: Generalization

- Generalization consists of factoring common elements (attributes, operations) within the set of classes into more general class called super class

- A new class (sub class) can be derived from the super class with some additional features in addition to the features in the super class

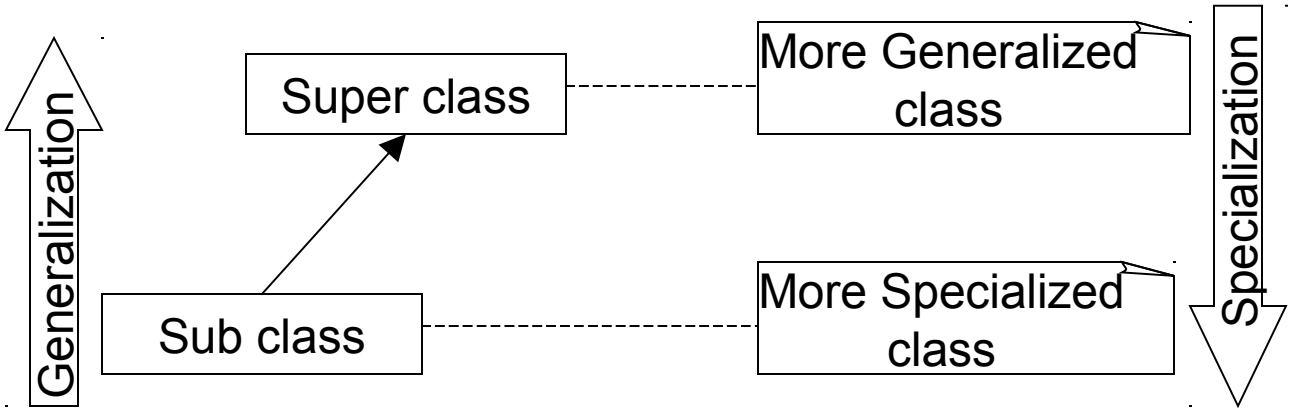
- Super class is an abstraction of its sub classes. Alternately, sub class is a detailed version than that of super class

Generalization: An Example



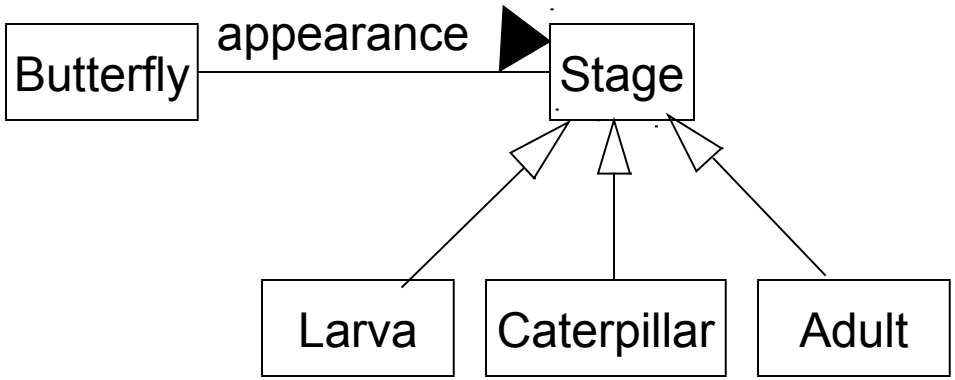
Classification: Specialization

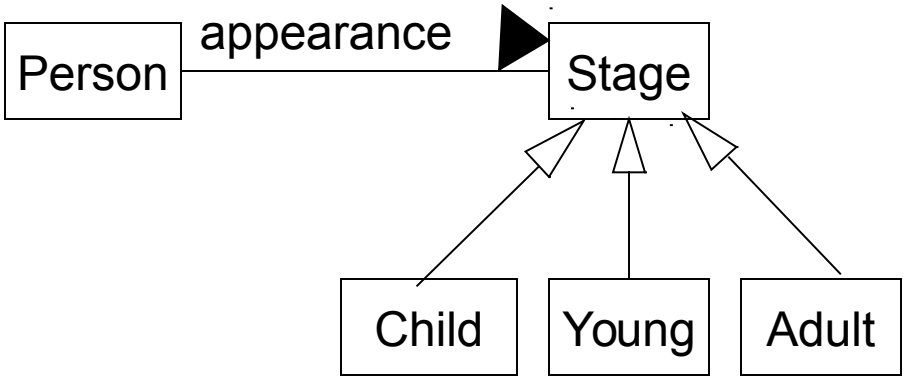
- Specialization allows the capture of the specific features of a set of objects that have not been distinguished by the classes already identified
- The characteristics are represented by a new class , which is a subclass of the one of the existing classes



Classification with Relations

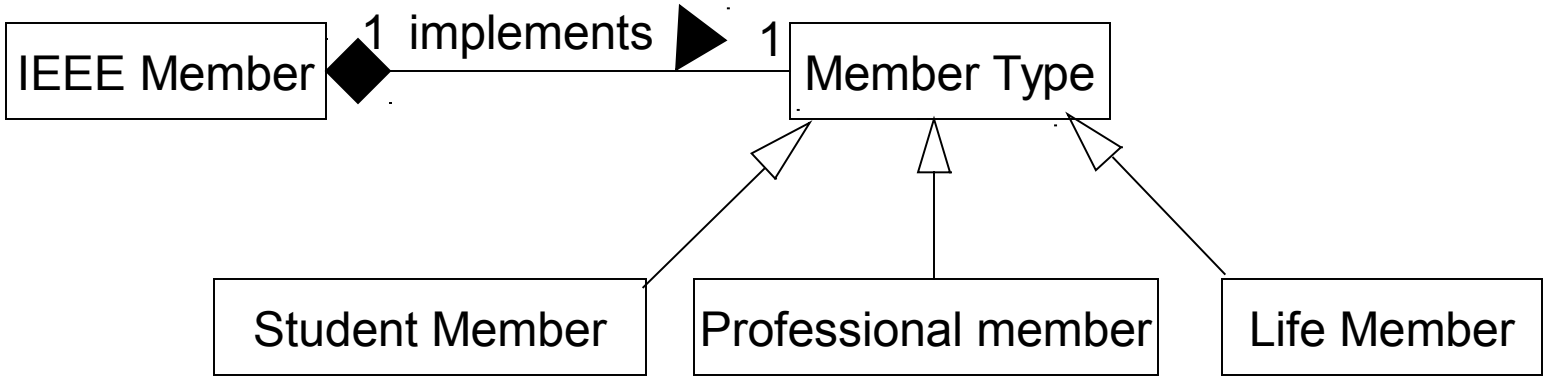
Examples: Classification with Association





Classification with Relations

Example: Classification with Aggregation

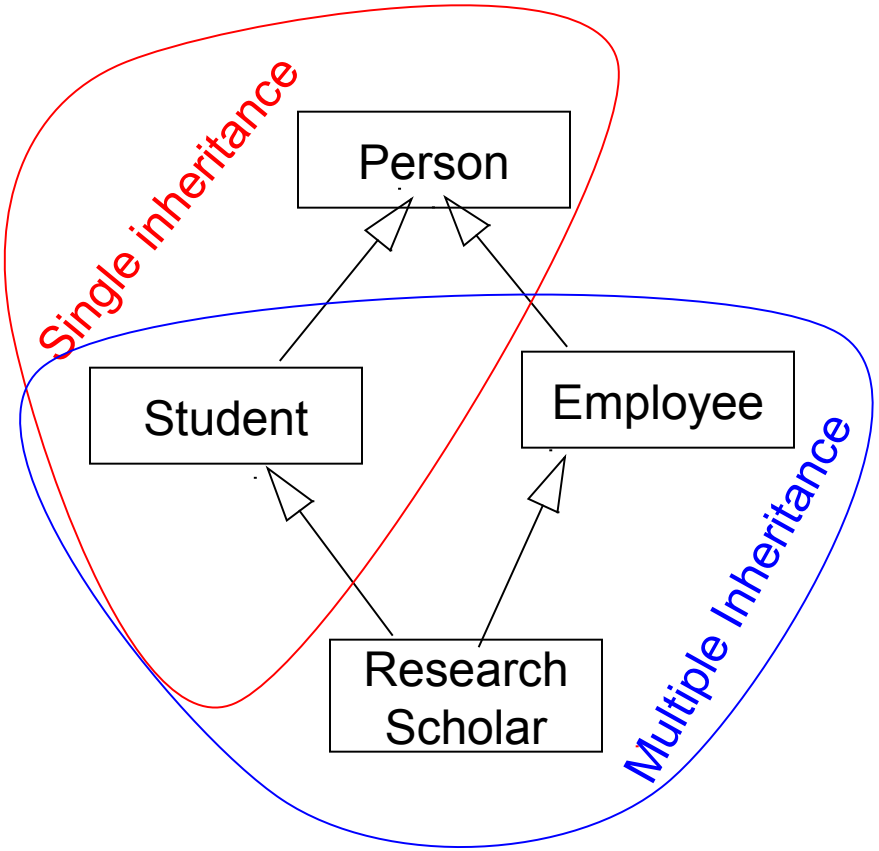


Inheritance & Delegation

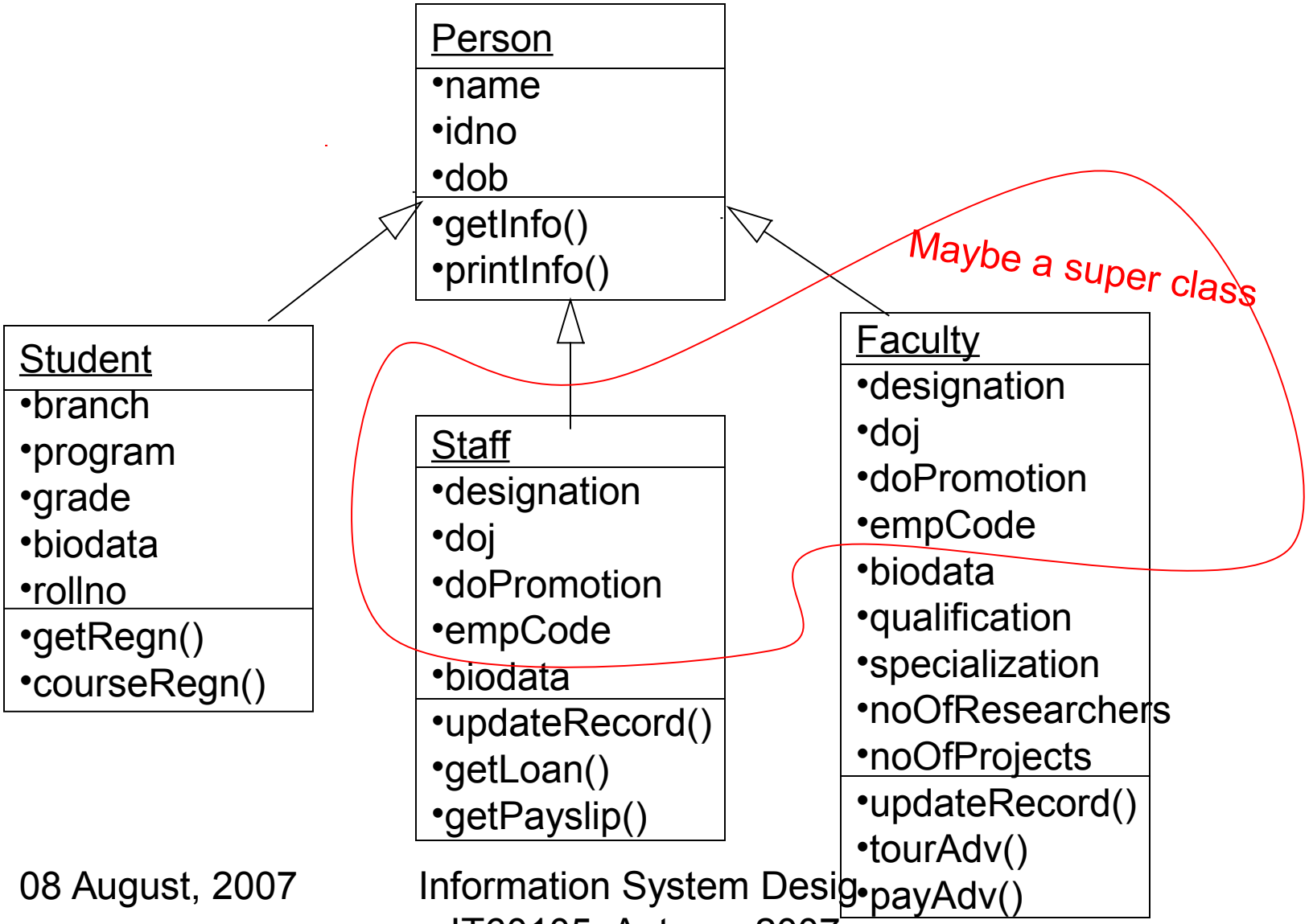
- Inheritance is the way to implement classification
  - Single inheritance
  - Multiple inheritance

- Delegation is the ability of an object to issue a message to another objects in response to a message

- Delegation is an alternative to inheritance



Inheritance: An Example



Polymorphism

Poly = many

Morphism = form

- An object (as well as attributes and operations) may be viewed with many forms

Example :

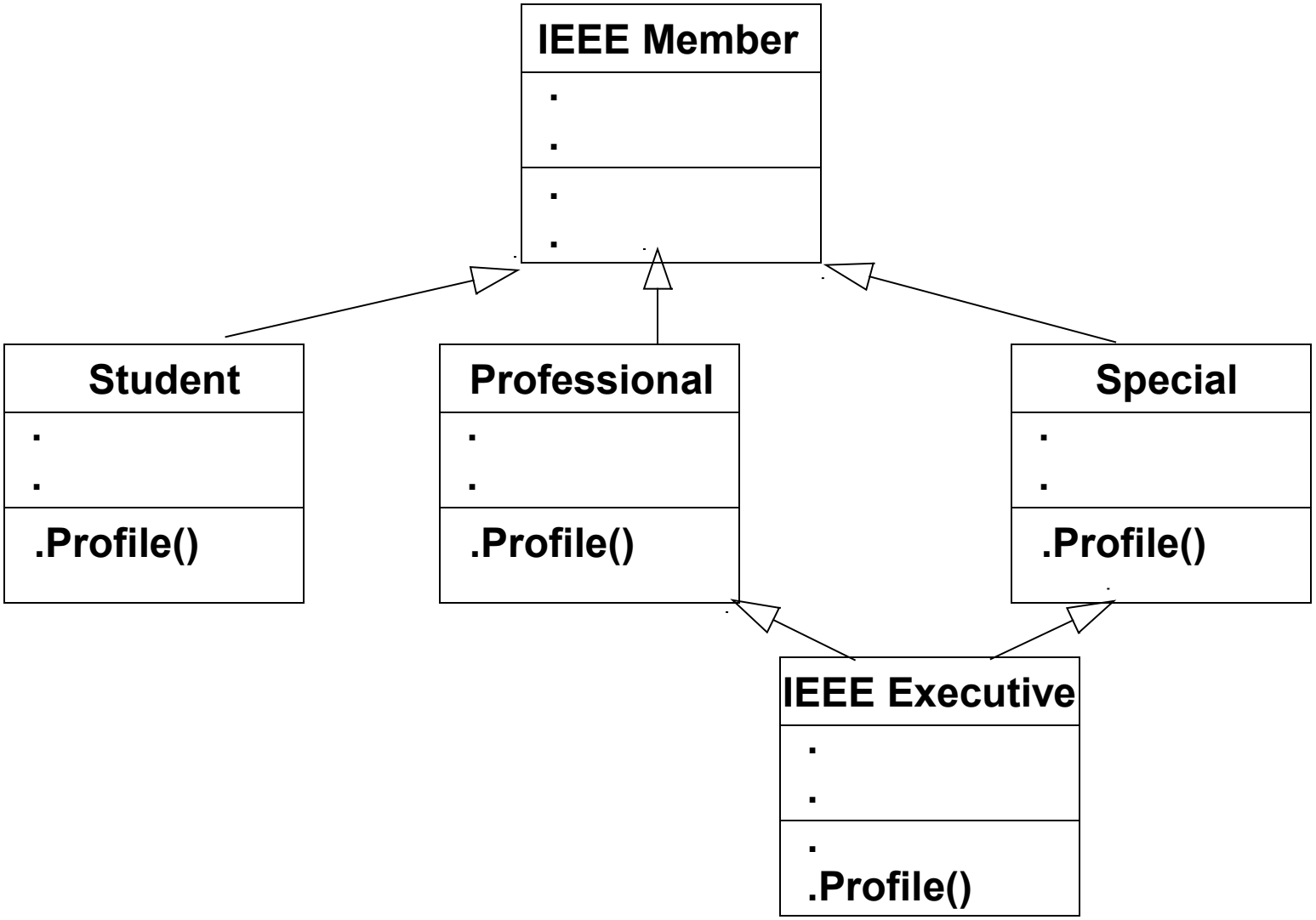
- Real world

  Water has two forms: solid (ice) and liquid (water)

- Algebra

$$z = z_1 + z_2 * z_3 + a * 5 - z_4$$

Polymorphism: An Example



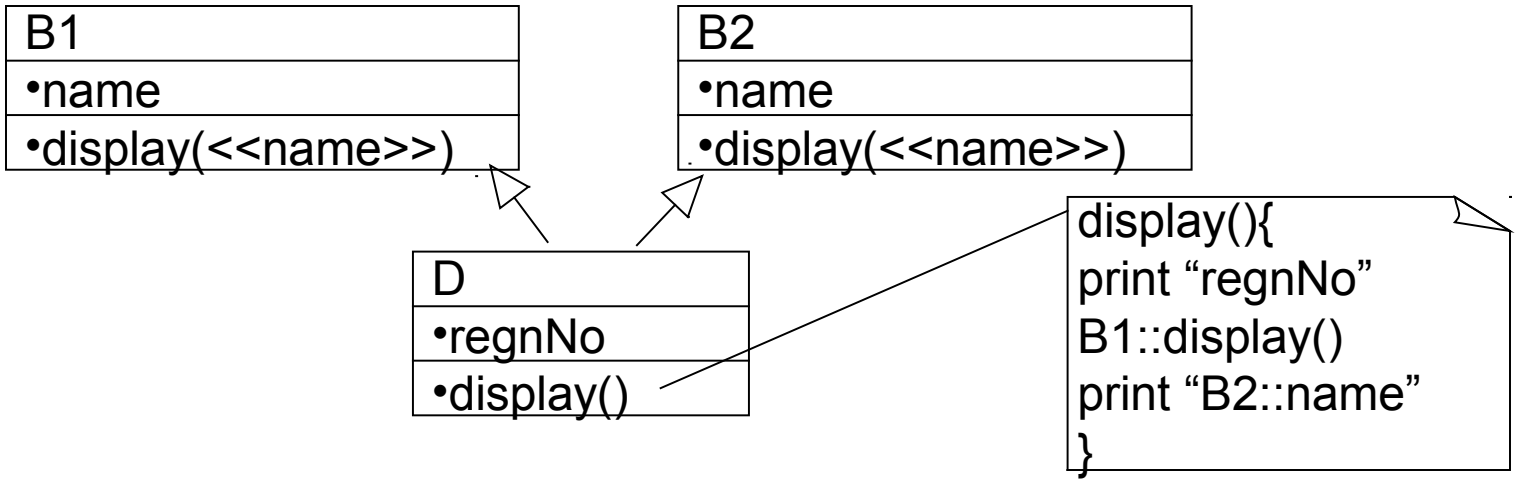
Binding

- Binding closely related to Polymorphism
  - Static binding (early binding)
    - Resolve during compile time
  - Dynamic binding (late binding)
    - Resolve during run time

Static Binding

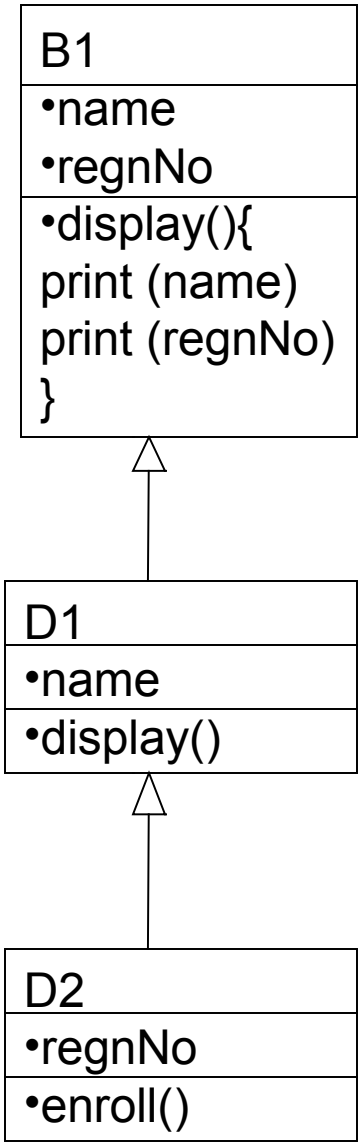
- In the case of static binding, the scope is required to be specified explicitly
- In C++, Java scope resolution operator(::) is used

Example:

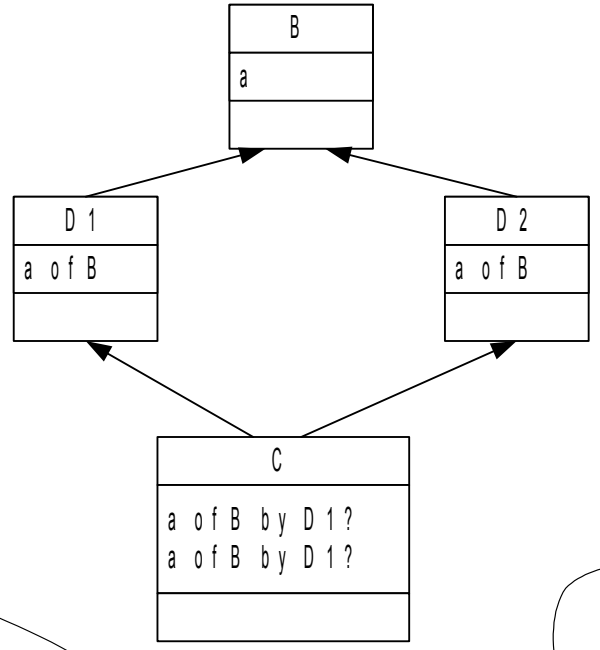


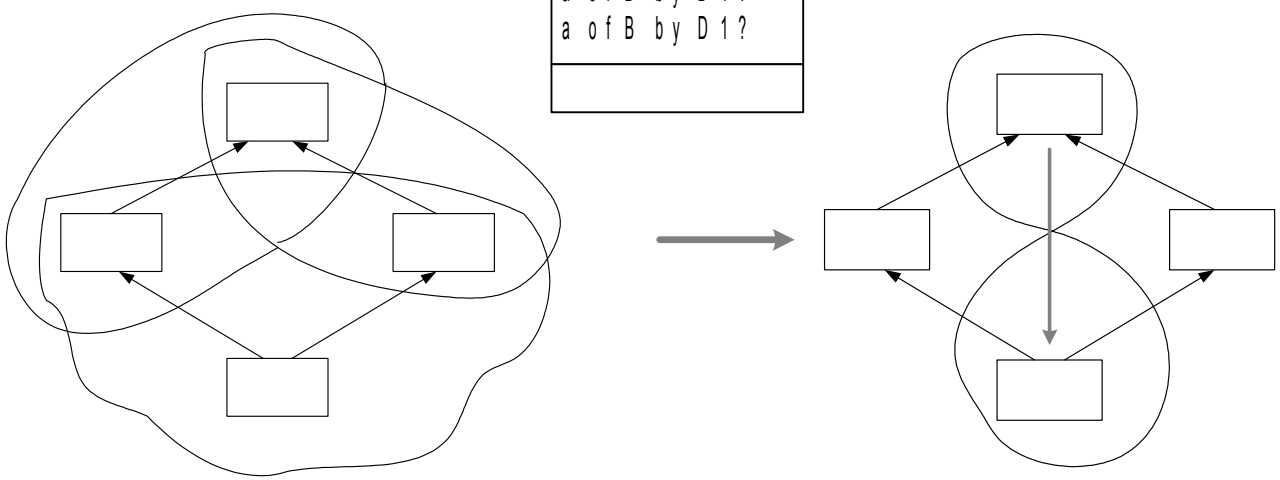
Dynamic Binding

- It resolve during run time
- Example
  - In d1.display ( ): name from d1, regnNo from B
  - In d2.display ( ): name from D1, regnNo from D2
  - Note: here d1, d2 are object of D1, D2 classes respectively



Problem to Ponder





Ambiguous

Unambiguous