

Class Modeling

Unit:2

Class Modeling

- ▶ A class model captures the **static structure** of the system by characterizing
 - the **objects** in the system,
 - the **relationships between the objects**, and
 - the **attributes** and
 - **operations** for each class of objects

Object and Class Concepts

Object

- ▶ Concept, abstraction, or a thing **with identity**
- ▶ **Proper nouns** or **specific reference** in problem description.

Class:

A Class is a group of objects with

- Same properties (attributes)
- Behaviour (operations)
- Kinds of relationships

Contd...

Class modelling

- ▶ It describe the static data of objects and their relationships to one another.

Class and Object Diagrams

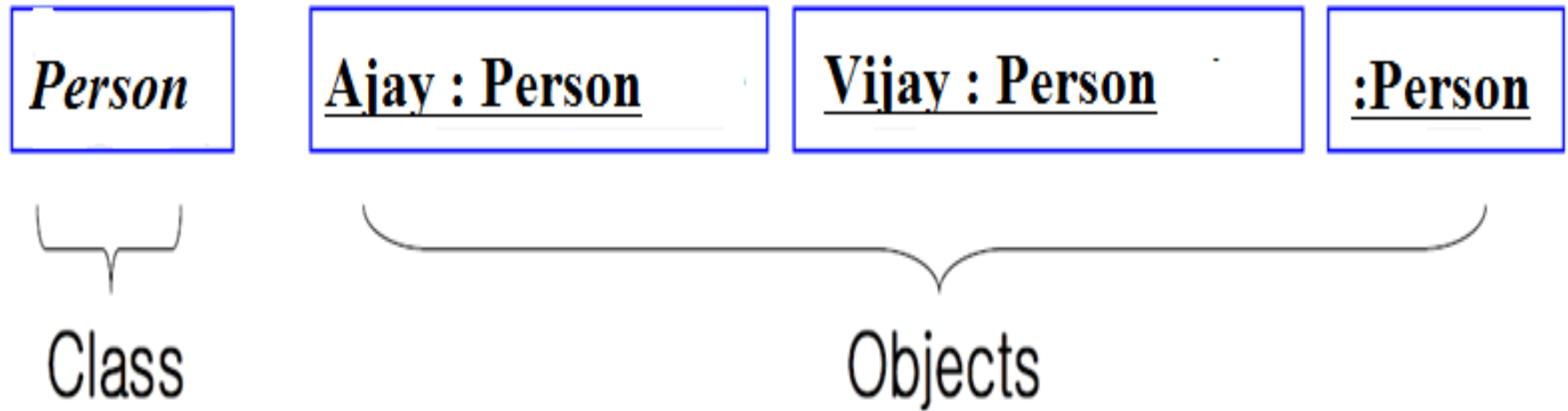
Class diagrams

- ▶ Graphic notation for modelling classes and their relationships.

Object diagrams

- ▶ Graphic notation for individual objects and their relationships

Example

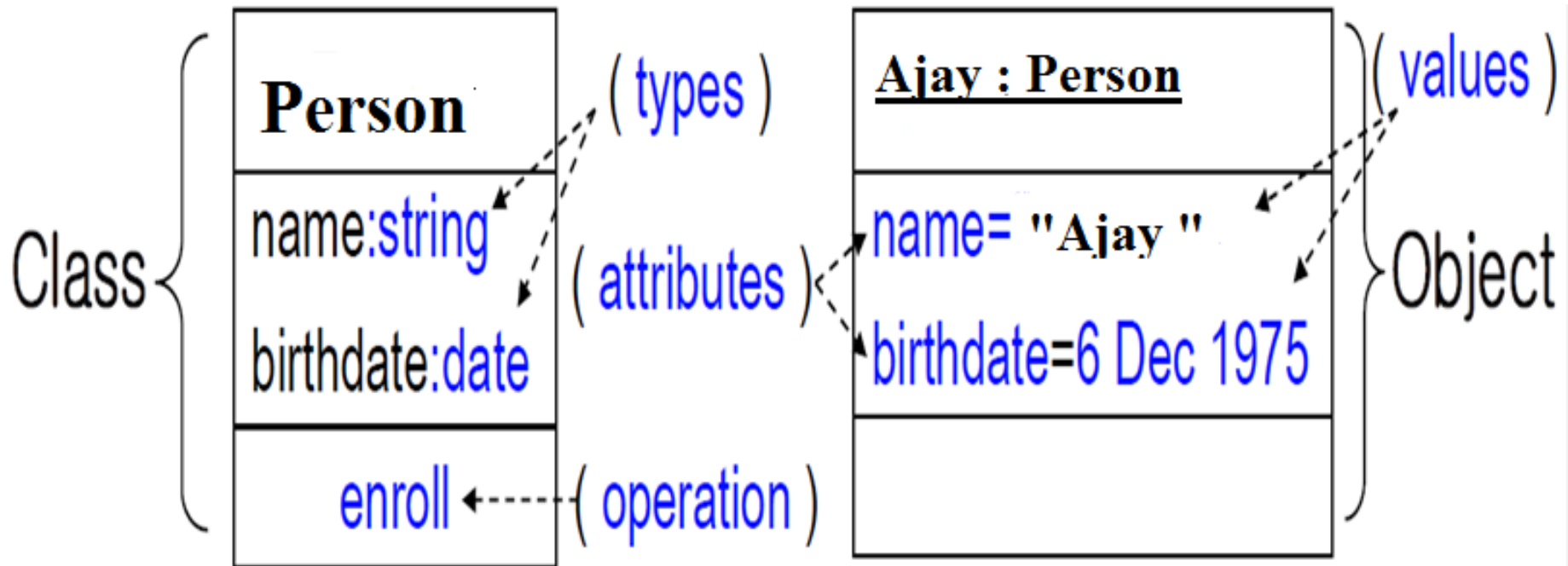


- ▶ *Note: classes & objects are the focus of class modeling.*

Values and Attributes

- ▶ **Value :** is a piece of data
- ▶ **Attribute :** value for each object. It is a named property of class.
- ▶ **Operations(Methods)**
 - ▶ Function or procedure that may be applied to or by objects in a class
 - ▶ All objects in a class share the same operations

Example



Summary of Notation for Classes

- A box represents a class and may have as many as three compartments. It contains from top to bottom: class name, list of attributes, and list of operations.
- The **attribute** and **operation** compartments are **optional** and missing attribute compartment means that are unspecified.

ClassName
attributeName1 : dataType1 = defaultValue1 attributeName2 : dataType2 = defaultValue2 ...
operationName1 (argumentList1) : resultType1 operationName2 (argumentList2) : resultType2 ...

UML conventions used are:

- ▶ UML symbol for both **class** and **objects** is **rectangular box**.
- ▶ Objects are modeled using box with **object name** followed by **colon** followed by **class name** and **underline** .
- ▶ Use **boldface** to list class name, **center** the name in box and **capitalize** first letter of name.

Links and Association concepts

- ▶ Links and Associations are the **means for establishing relationships** among objects and classes.

Then,

- ▶ What is a Link ? and
- ▶ What is an Association?

Links and Association concepts...

Link

- ▶ It is physical or conceptual connection among objects.
- ▶ For Eg: Ajay *WorksFor* IBM Company.
- ▶ Mathematically, we define a link as a **tuple** – i.e., a list of objects.
- ▶ Most links relate two **objects**, but some links relate 3 or **more objects**.

Link...

- ▶ A link is **instance of an association**.
- ▶ UML notation for a link is a **line between objects**. It provides a relationship among objects.
- ▶ The fact is, link is not part of object by itself, but depends on both of them together.
- ▶ Eg: **person is not a part of company and company is not a part of person**

Contd..

Association

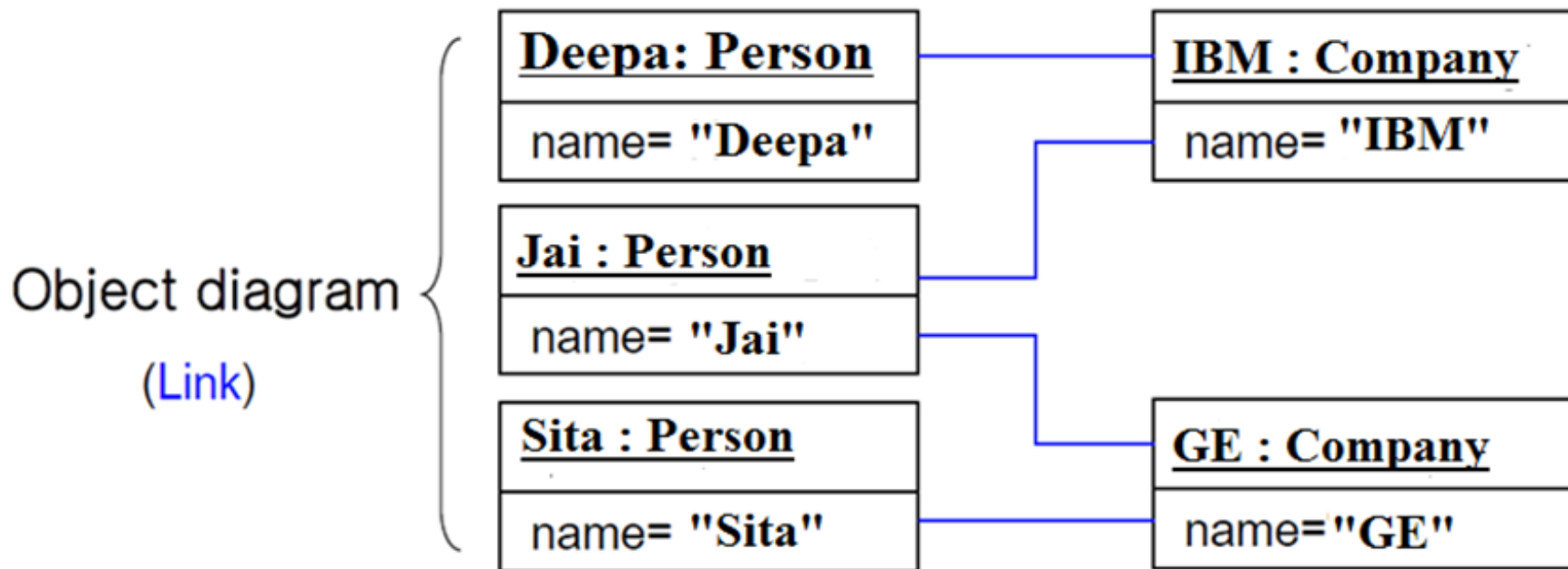
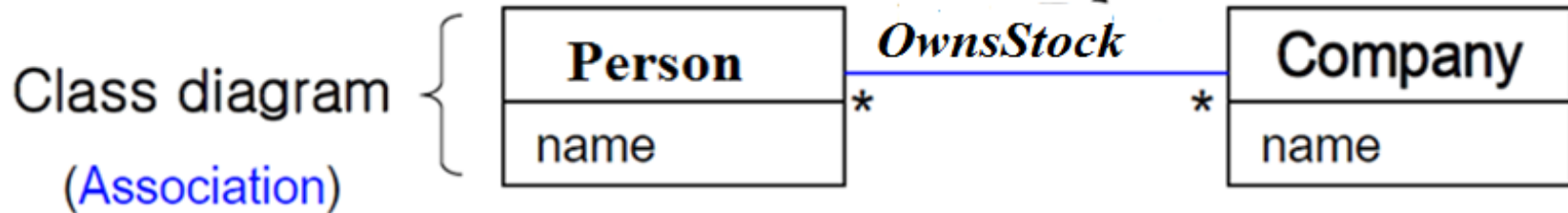
- ▶ Group of links with common structure and common semantics
- ▶ It is set of potential links in the same way that a class describes a set of potential objects.
- ▶ Association is bi-directional
- ▶ Eg: **A Person works for company and inverse company employs person**



if association name is replaced with "owns>",
it would read "Class 1 owns Class 2"

Example

- Links and associations (Cont'd) (Association name)

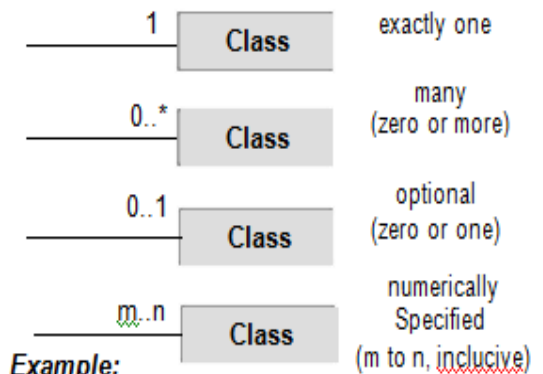


Many_to_many Association

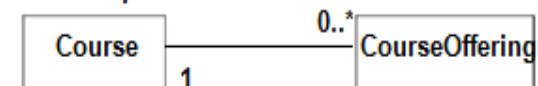
Multiplicity

- ▶ Constrains the number of related objects.
 - ▶ It specifies the number of instances of one class that may relate to a single instance of an associated class.
 - ▶ Assign a multiplicity to an association end
-
- ▶ UML diagrams specifies the 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)

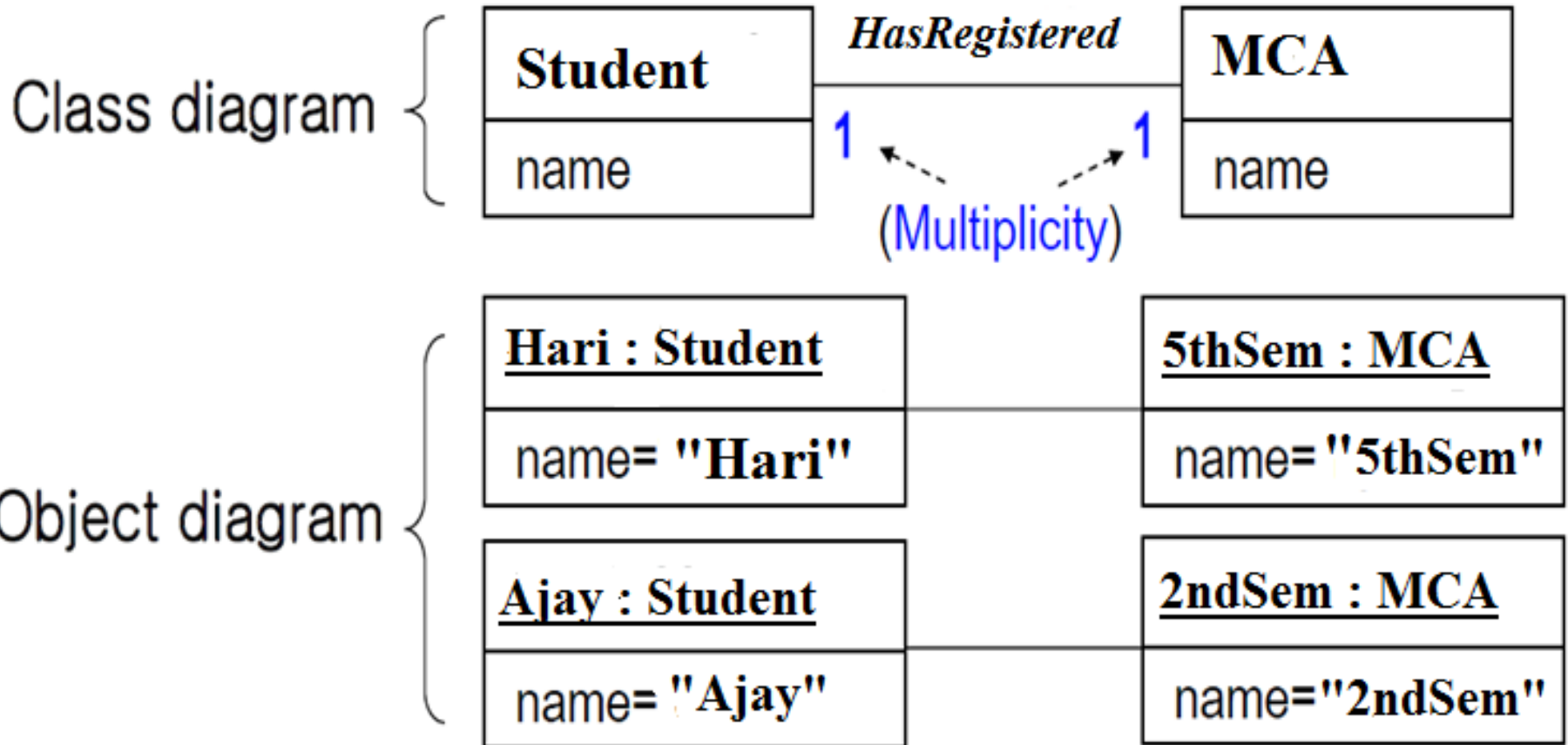
• notations



Example:

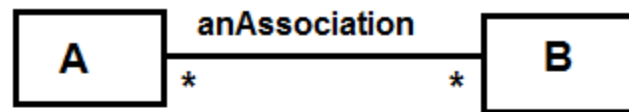


Example

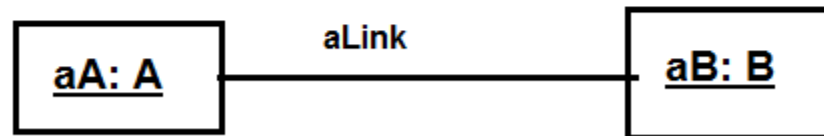


Association Vs Link

- ▶ A pair of objects can be instantiated at most once per association



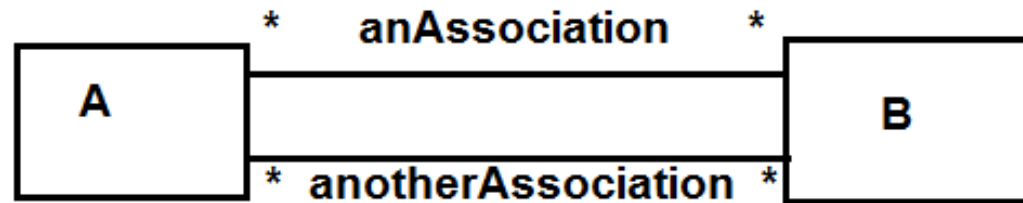
Class Diagram



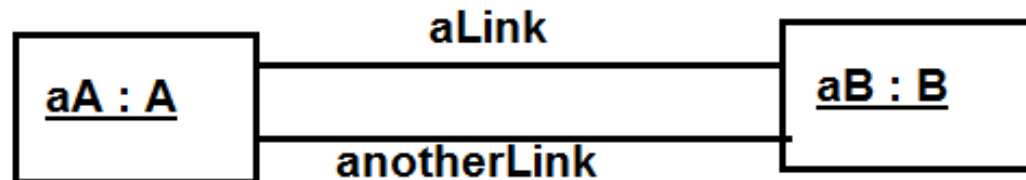
Object Diagram

Association Vs Link...

- ▶ You can use multiple associations to model multiple links between the same objects



Class Diagram



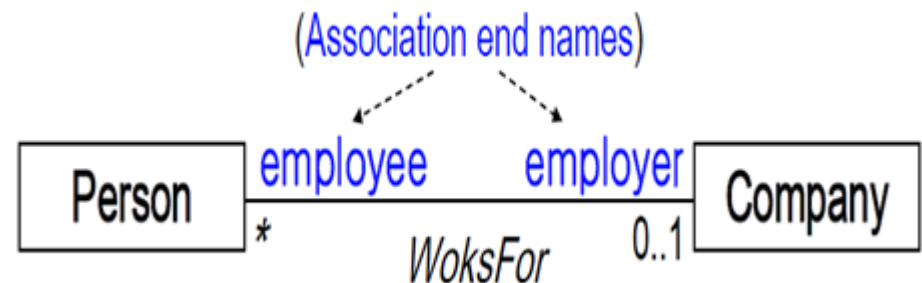
Object Diagram

Multiplicity Vs Cardinality

- ▶ Multiplicity is a **constraint on the size** of a collection.
- ▶ Cardinality is a **count of elements** that are **actually** in a collection.
- ▶ Therefore, **Multiplicity is a constraint on Cardinality.**

Association End names

- ▶ Multiplicity implicitly refers to the end of association.
- ▶ Association ends can not only be assigned multiplicity, they can also be named.
- ▶ Association end names often appeared as **noun** in problem descriptions. **Each end of an association can have a name.**

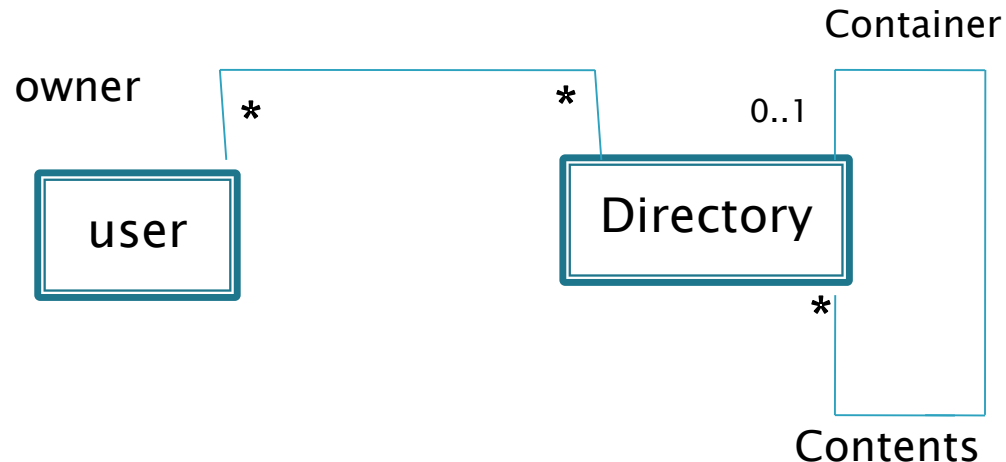


- ▶ These are convenience for traversing associations

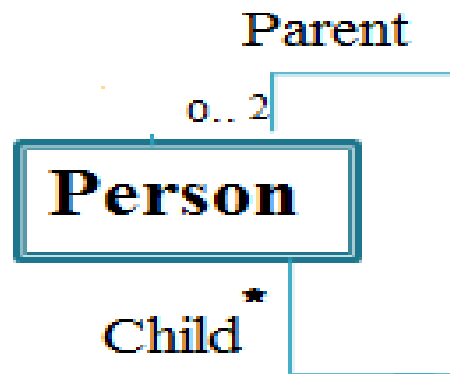
employee	employer
Ajay	IBM
Vijay	GE
Sita	GE

Example

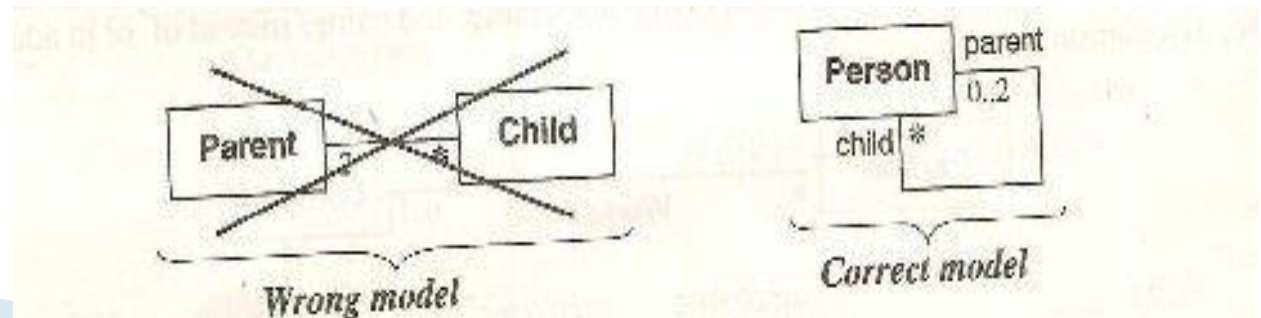
Association end names are necessary for associations between two objects of the same class (self association) they can also distinguish multiple associations between a pair of classes



- Use association end names to model multiple references to the same class.



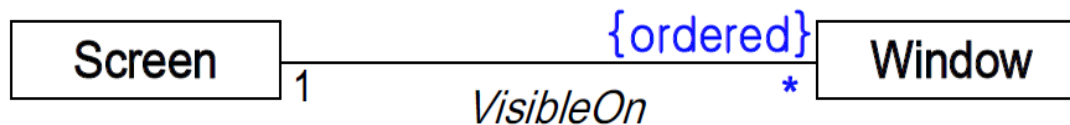
- Note:



Ordering

Ordering

- ▶ Association which follows some **priority**.
- ▶ It indicates an ordered set of objects
 - Provide an explicit order
- ▶ Ordered collection of **elements without duplicates**.
- ▶ For example,



- ▶ Fig shows a **workstation screen containing a number of overlapping windows**. Each window on a screen occurs at most once.
- ▶ The windows have an explicit order, so only the top-most window is visible at most once.
- ▶ The ordering is an inherent part of the association. You can indicate an ordered set of objects by writing “**{ordered}**” next to the appropriate associated end.

Bags and Sequences

Bags and Sequences

- **Permit multiple links for a pair of objects.**

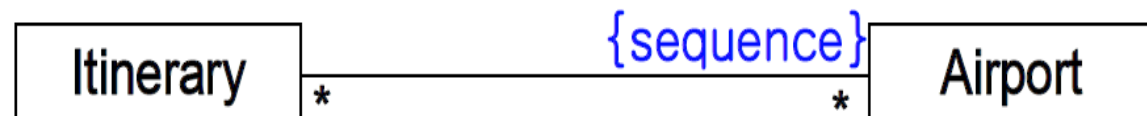
Bag:

It is collection of elements **with duplicates** allowed.

Sequence:

- ▶ It is an **Ordered** collection of elements with duplications allowed.

For example: An itinerary may visit multiple airports so use {sequence} and not {ordered}.

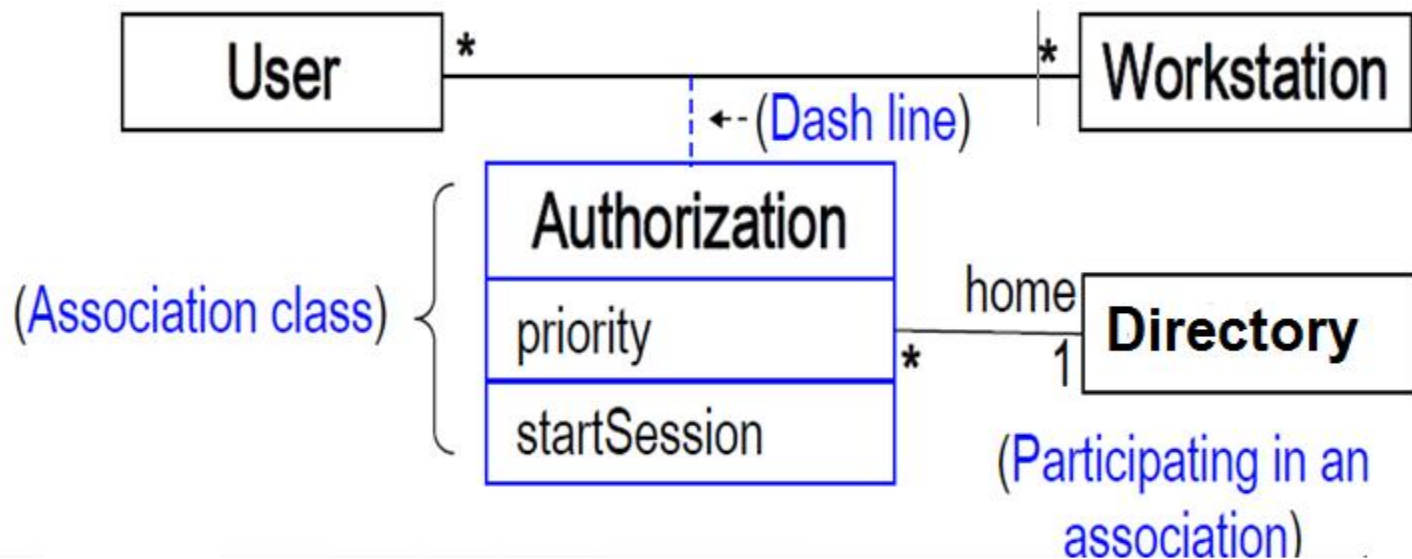


Note:

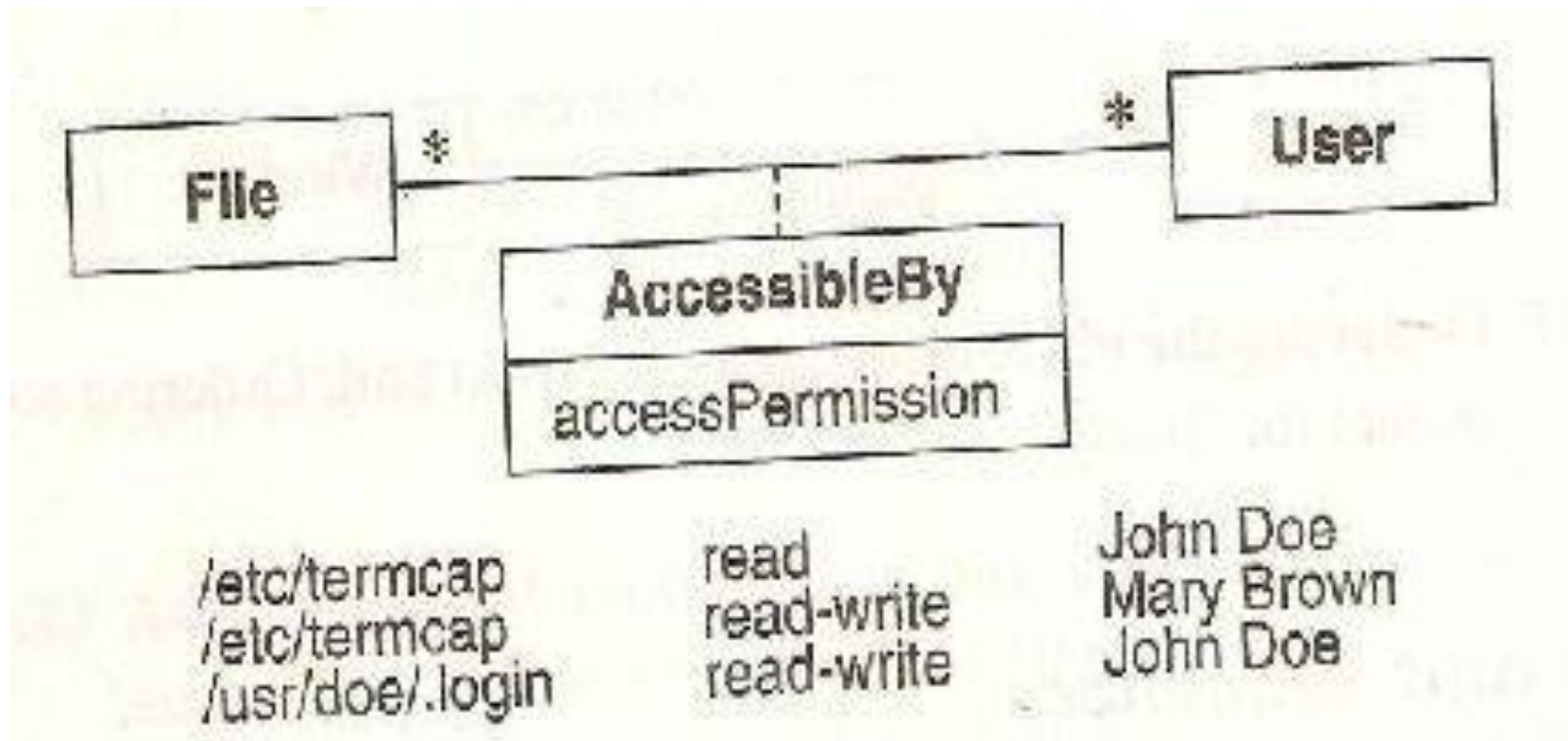
- ▶ UML1: **do not** permit multiple links for a pair of objects. Some modelers misunderstood this restriction with {ordered} association ends and constructed incorrect model, assuming that there could be multiple links.
- ▶ UML2: the modeler's intent is now clear. If you **specify {bag} or {sequence}, then there can be multiple links for a pair of objects**. If you omit these annotations, then the association has at most one link for a pair of objects.
- ▶ {ordered} and {sequence} annotations are the same, except that the first disallows duplication & the other allows them.
 - A sequence association is an ordered bag, while an ordered association is an ordered set.

Association Classes

- ▶ An Association class is an association that is also a class.
- ▶ Derive identity from instances of the constituent classes.
- ▶ Have **attributes and operations** & **participate in associations**. For example

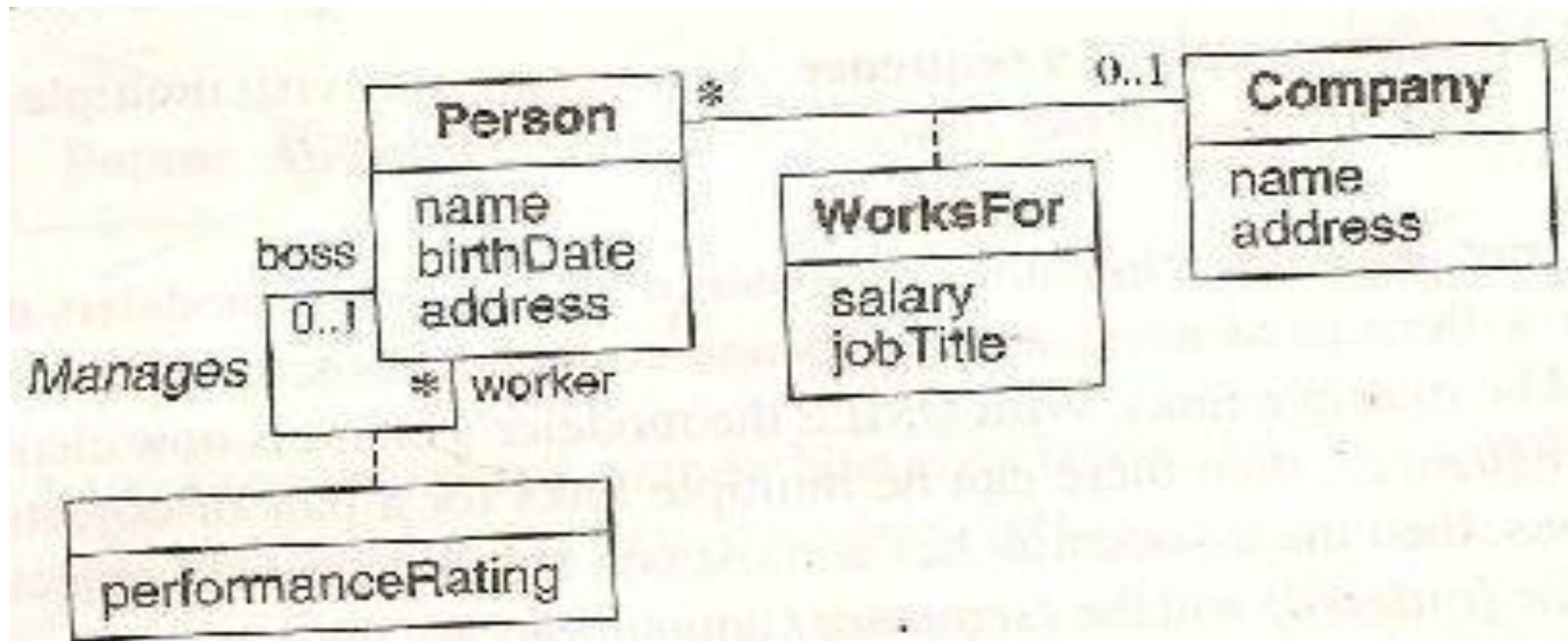


Example1:



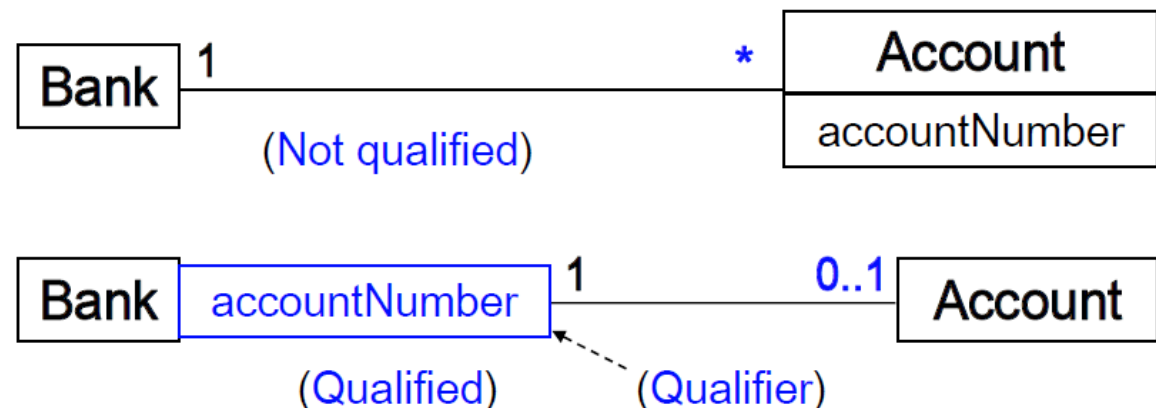
example2:

- ▶ Each person working for a company receives a salary & has a job title.
- ▶ The boss evaluates the performance of each worker.
The attributes may also occur for one-to-one & one-to-many associations

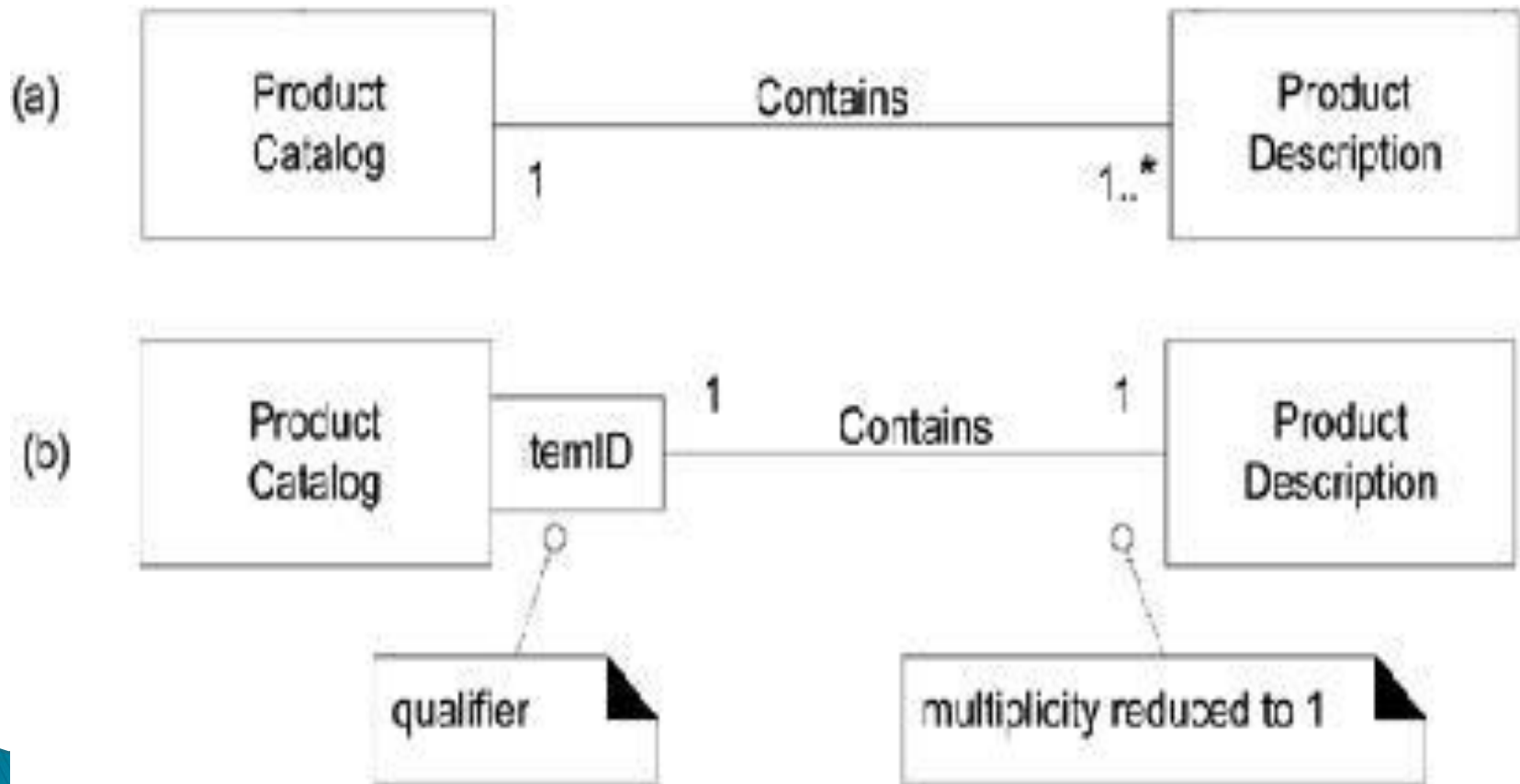


Qualified Associations

- ▶ It is an association in which an attribute called **qualifier** disambiguate the objects for a “many” association end. It is possible to define qualifiers for one-to-many and many-to-many associations.
- ▶ Job of qualifier is to reduce the effective multiplicity, from “many ”to “one”.
- ▶ For example1:



Example2:



Generalization & Inheritance

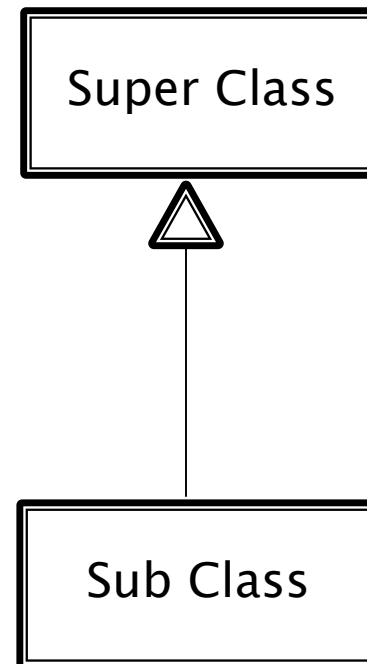
Generalization

- ▶ Relationship between a class (the superclass) and one or more variations of the class (the subclass)
- ▶ Simple generalization organizes classes into a hierarchy; each subclass has a single immediate superclass. There can be multiple levels of generalizations.
- ▶ Generalization, specialization and inheritance all refer to aspects of the same idea.

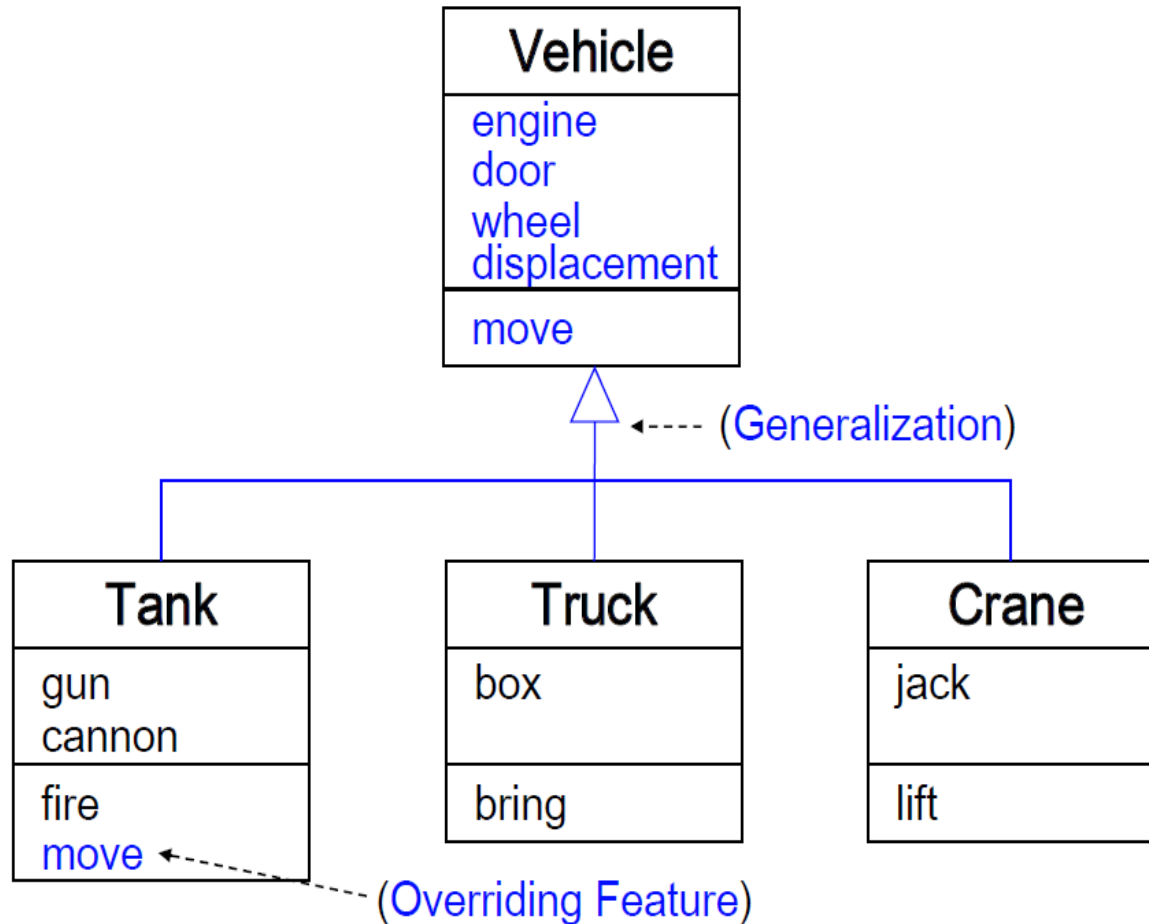
UML notation for generalization

- A large hollow arrowhead denotes generalization.

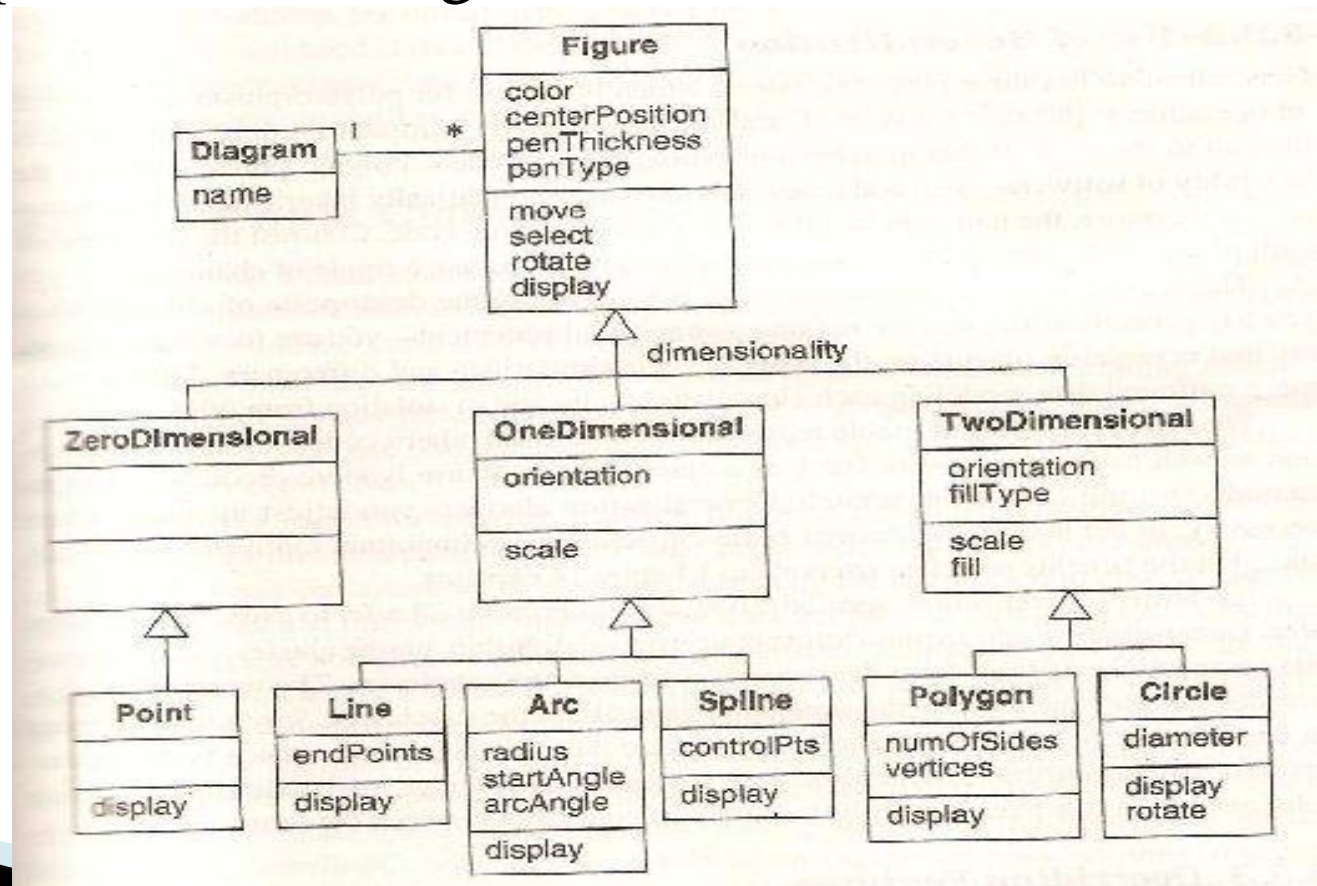
The arrow head points to the superclass from its subclass.



Example



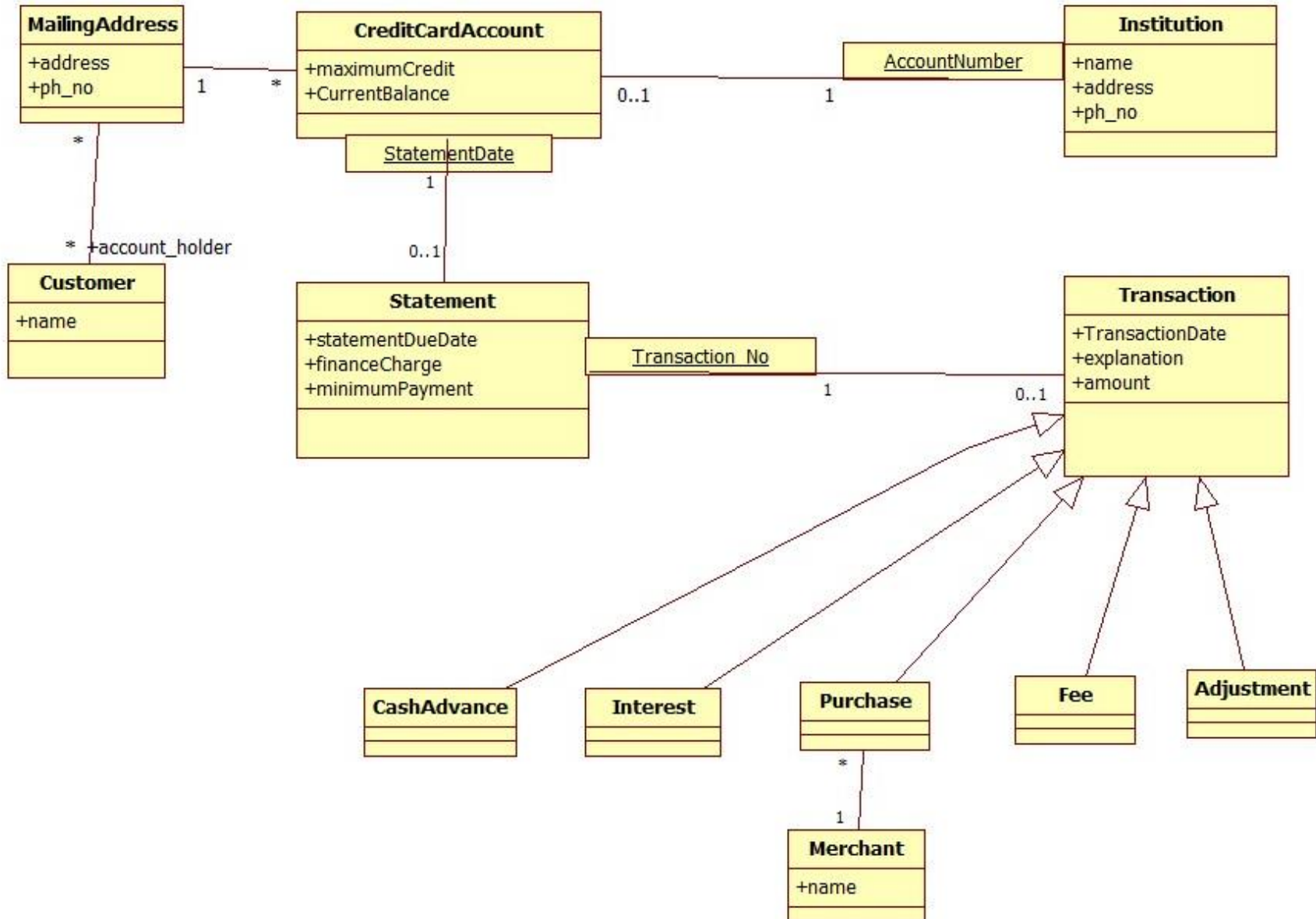
- ▶ In fig. word **dimensionality** is the generalization set name.
- ▶ Generalization set name is an enumerated attribute that indicates **which aspect of an object is being abstracted** by a particular generalization.(and it is optional).
- ▶ Only one aspect need to be generalized at a time.

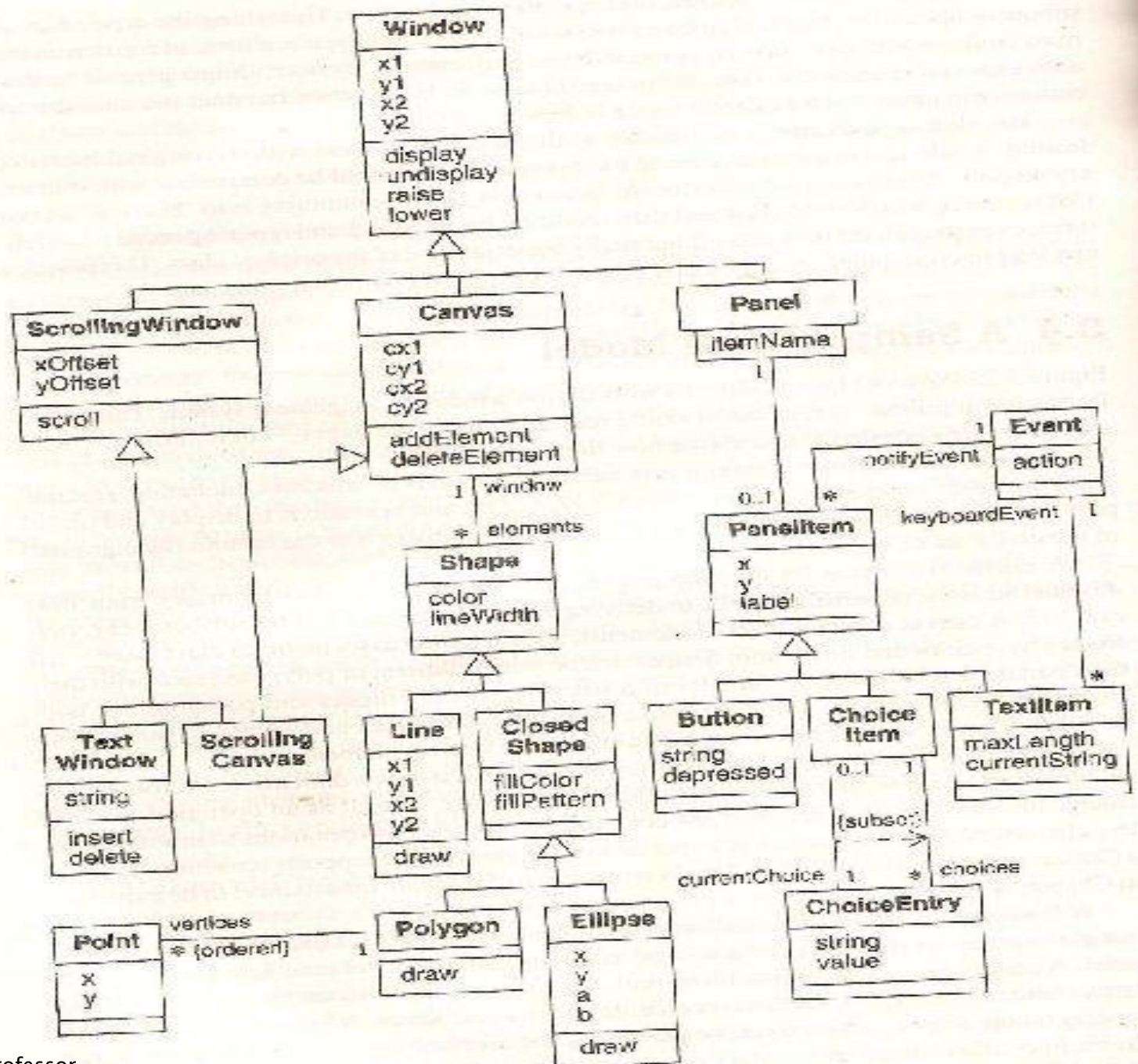


Use of generalization:

- 1. To support polymorphism:** You can call an operation at the superclass level, and the OO language compiler automatically resolves the call to the method that matches the calling object's class.
- 2. To structure the description of objects:** i.e to frame a taxonomy and organizing objects on the basis of their similarities and differences.
- 3. To enable reuse of code:** Reuse is more productive than repeatedly writing code from scratch.

A Sample Class Model





Advanced Class Modeling

»» Chapter 2

Advanced Class Modelling

Advanced Object & Class concepts:

- ▶ It is important for complex applications.

Some of the advanced class modeling concepts are:

1. **Enumerations**
2. **Multiplicity**
3. **Scope**
4. **Visibility**

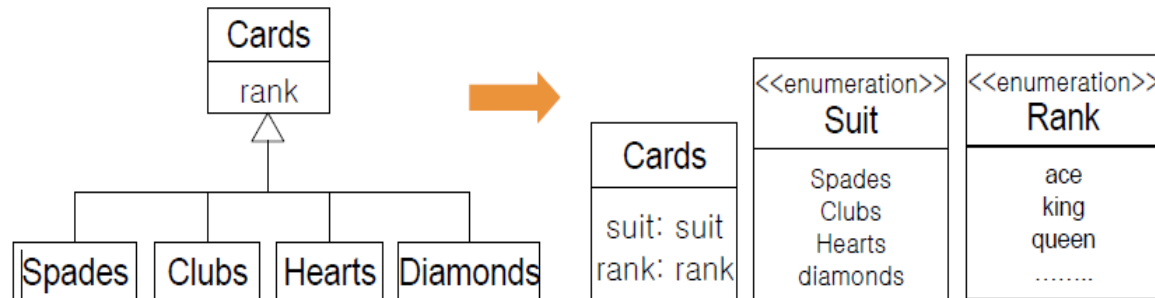
1. Enumerations

- ▶ An enumeration is a data type that has a **finite set of values**.
- ▶ A data type is the description of values. It includes numbers, strings and enumeration.
- ▶ For Eg:
 1. Weekdays = { Monday, Tuesday, ..., Sunday }
 2. Months = { January, February, ..., December }

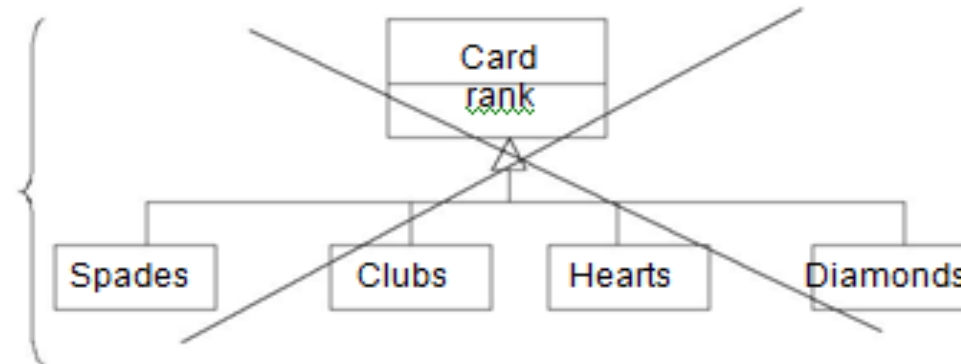
- Pen type is an enumeration that includes solid, dashed and dotted.

Solid _____
 dashed - - - - -
 dotted

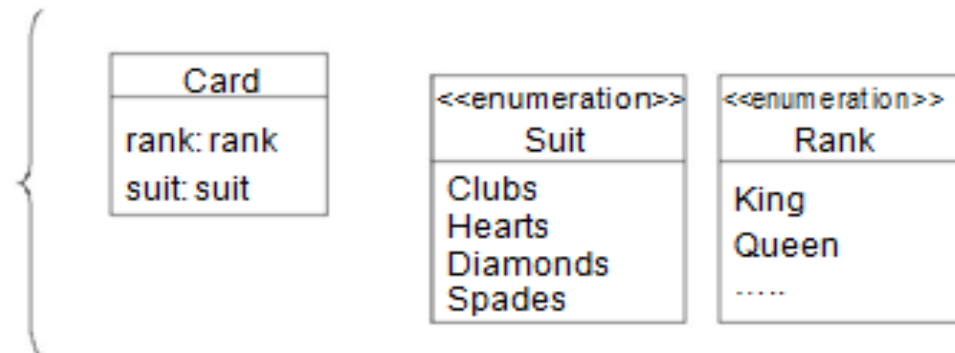
- Two dimensional fill type



Wrong



Correct



Modeling enumerations. Do not use a generalization to capture the values of an enumerated attribute

Advanced objects & class concepts...

2. Multiplicity

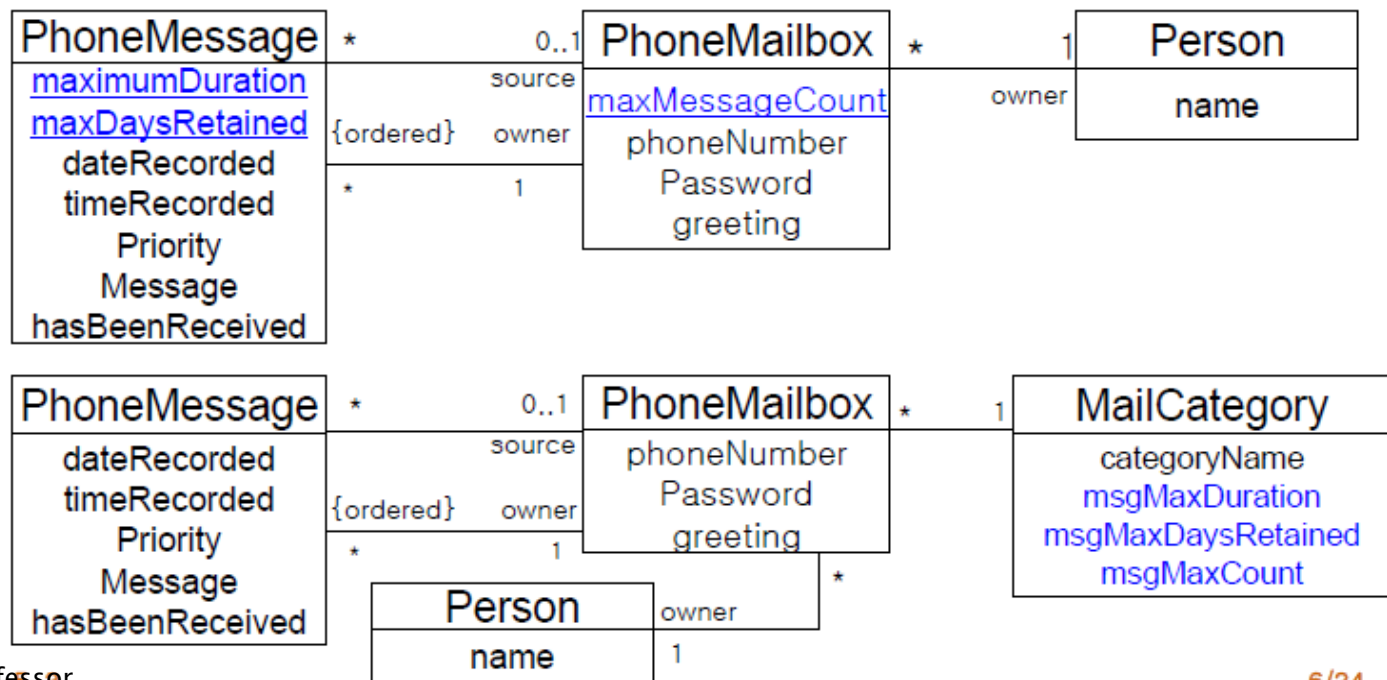
- It is a constraint on the cardinality of a set.
- It can **also applies to attribute**[single or collection]
- it specifies the number of possible values for each instantiation of an attribute.
- For example, the most common specifications are
 - A mandatory single value[1]
 - An optional single value [0..1]
 - Many [*]

Person
name : string[1] address : string[1...*] phoneNumber : string[*] brithDate : date[1]

Advanced objects & class concepts...

3. Scope

- ▶ It indicates if a feature applies to an object or a class.
For example,
- ▶ Underline distinguishes feature with scope from those with object scope



Advanced objects & class concepts...

4. Visibility

Ability of a method to reference a feature from another class

- ▶ Visibility has possible values of:

Public(+), **protected(#)**, **private(-)**, **package(~)**

- ▶ Any method can freely access **public features**.
- ▶ Only methods of containing class & its descendants via **inheritance** can access **protected** features.
- ▶ Only methods of containing class can access **private** features.
- ▶ Methods of classes defined in the same **package** as the target class can access **package** features.

► UML denotes visibility with a prefix

“ + ” precedes public features.

“ # ” precedes protected features.

“ - ” precedes private features.

“ ~ ” precedes package features.

Associations Ends

- ▶ Association end refers to the **end of the association**.
- ▶ A binary associations has two ends, a ternary associations has 3 ends and so forth.
- ▶ **The properties of association end are:**
 1. **Aggregation** → the association end can be an aggregate of constituent parts. Only a binary association can be an aggregation.
 2. **Changeability** → this property specifies the status of an association
 - the possibilities are - changeable (can be updated)
- read only (can only be read)

3. **Navigability** → conceptually, an association may be traversed in either direction.

▶ However, an implementation may support only one direction.

4. **Visibility** → similar to attributes & operations an association ends may be public, protected, private or package.

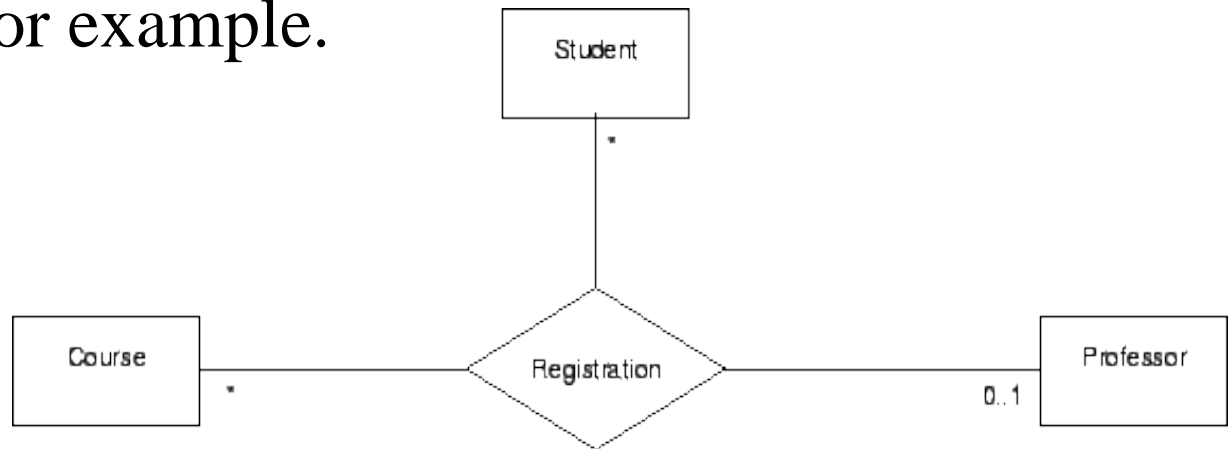
Association...

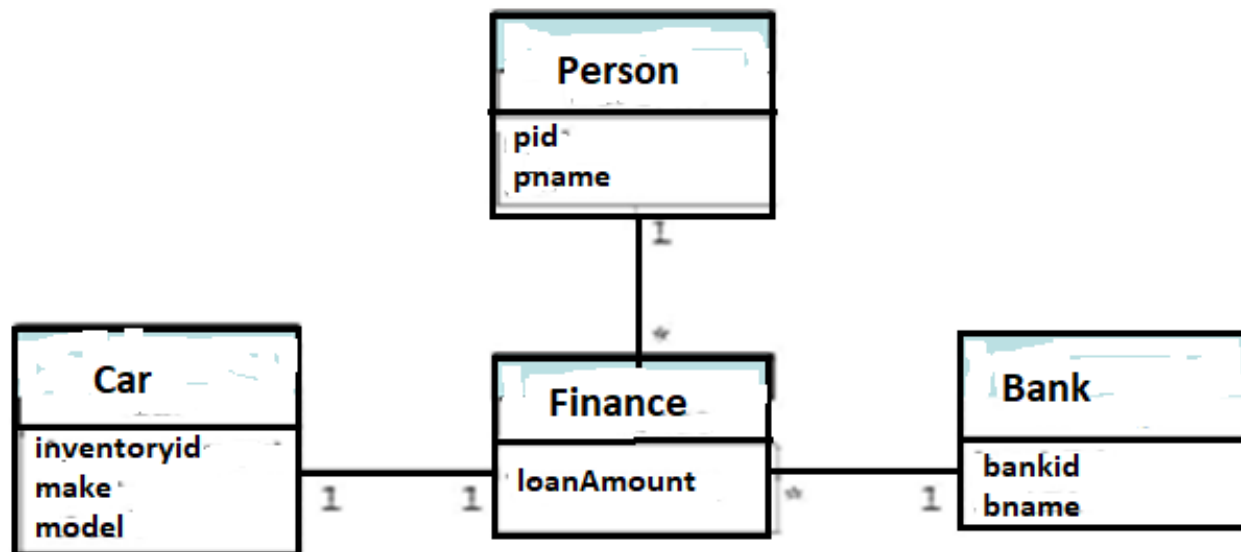
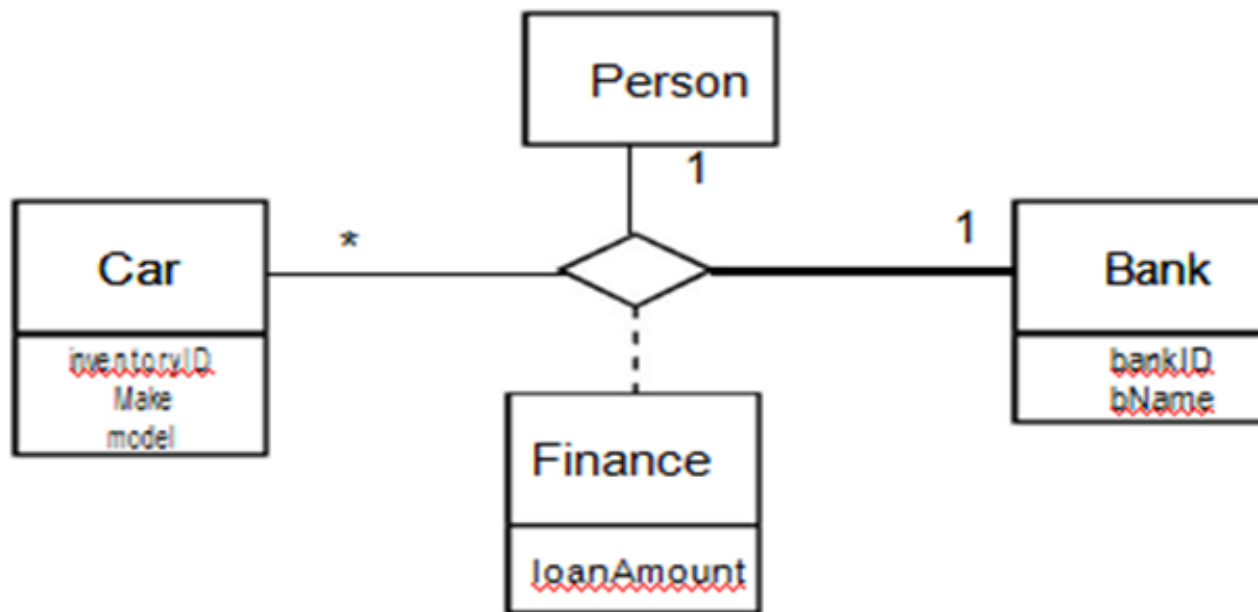
N-ary associations

Association among **three or more classes**

Rules:

- ▶ End names are necessary if a class participates in an n-ary association more than once.
- ▶ Most n-ary associations can be decomposed into binary associations. For example.






- ▶ The UML symbol for n-ary associations is a **diamond with lines connecting to related classes**. If the association has a name, it is written in **italics** next to the diamond.
- ▶ **A typical programming language cannot express n-ary associations. So, promote n-ary associations to classes.**

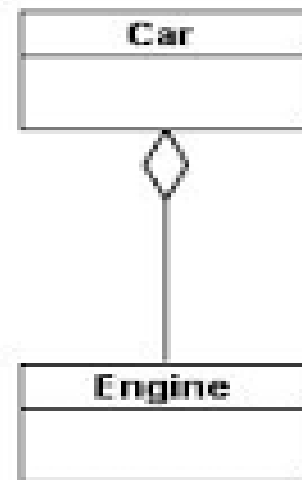
Note: Be aware that do not change the meaning of a model, when you promote n-ary associations to classes.

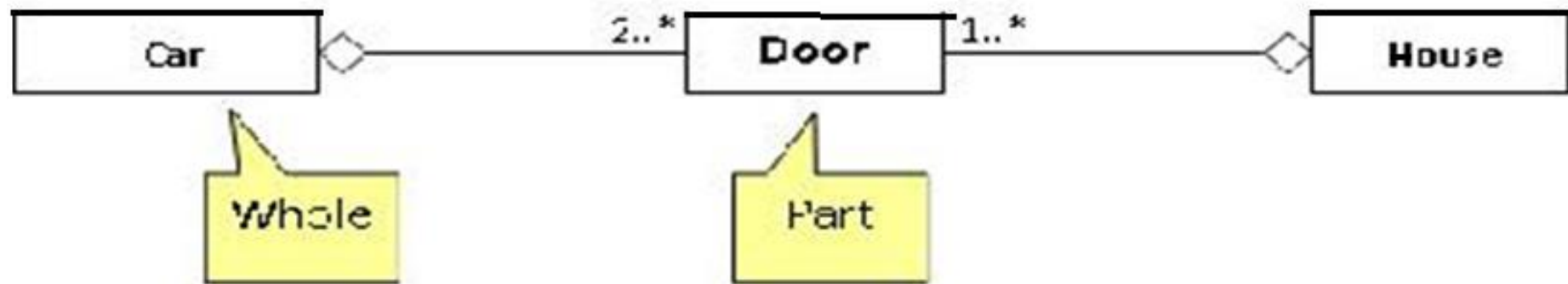
Aggregation

- ▶ Aggregation is a **strong form of association** in which **aggregate object is made of constituent parts**.
- ▶ Aggregation is a special form of association, not an independent concept.
- ▶ If two objects are tightly bound by a **part whole** relationship, it is an aggregation.

Contd...

- ▶ The most significant property of aggregation is **transitivity** and it is also consider as anti symmetric.
- ▶ The aggregation is represented by symbol
- ▶ Example, 





Composition

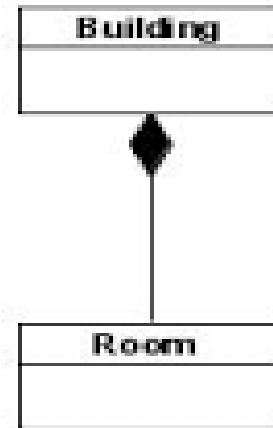
- ▶ Composition is a **form of aggregation with two additional constraints.**
 - A constituent part can belong to at-most one assembly.
 - Composition between two objects, the composed object cannot exist without other object.

Contd..

- ▶ When an object contains other object, if the contained object cannot exist without the existence of container object.
- ▶ The UML notation for composition is



Example



Aggregation versus Association

- ▶ **Aggregation:** if two objects are tightly bound by a part-whole relationship.

Class A **contains** Class B , Or Class A **has** instance of Class B.

Example: Team has Players.

- ▶ Life or existence of the aggregated objects are independent of each other, But one object is playing the role of Owner of the other.

Contd..

- ▶ Association if two objects are usually considered as independent.
- ▶ Class A **uses** Class B.

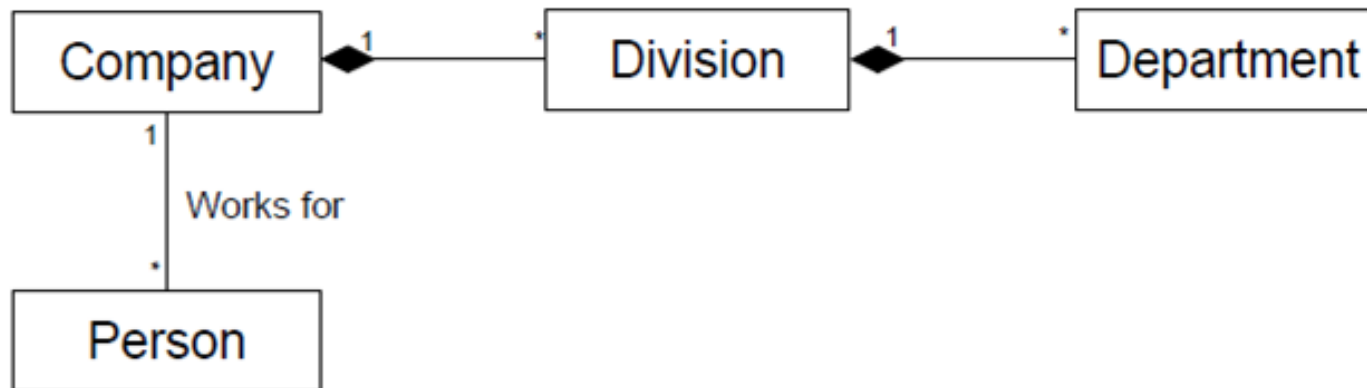
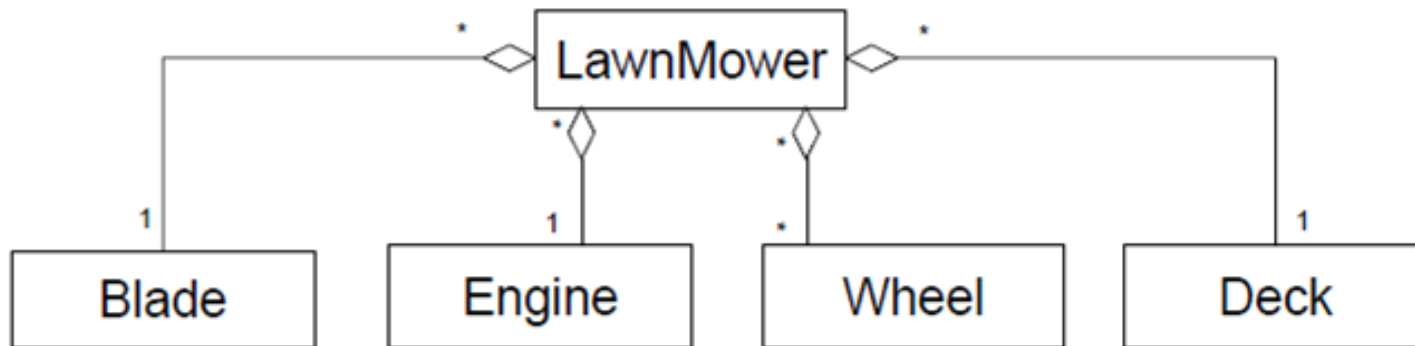
Example: Employee **uses** Bus Service **for** transportation.

Aggregation Versus Composition

- ▶ **Aggregation** - if the part may be simultaneously in many aggregate objects.
- ▶ **Composition** - if the part is not part of any other composite object.
- Composition: Class A **owns** Class B.
- **Example :Body consists of Arm, Head, Legs.**
- Life or existence of the composite object is dependent on the existence of container object, Existence of composite object is not meaningful without its container object.

Contd..

Aggregation



<Composition>

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 objects.
- ▶ Eg: Moving an aggregate moves its parts.
Move operation propagates to the parts.

Propagation of operation to parts is often a good indicator of aggregation.

- ▶ Propagation is possible for the operations including **save/restore**, **destroy**, **print**, **lock**, **display**.

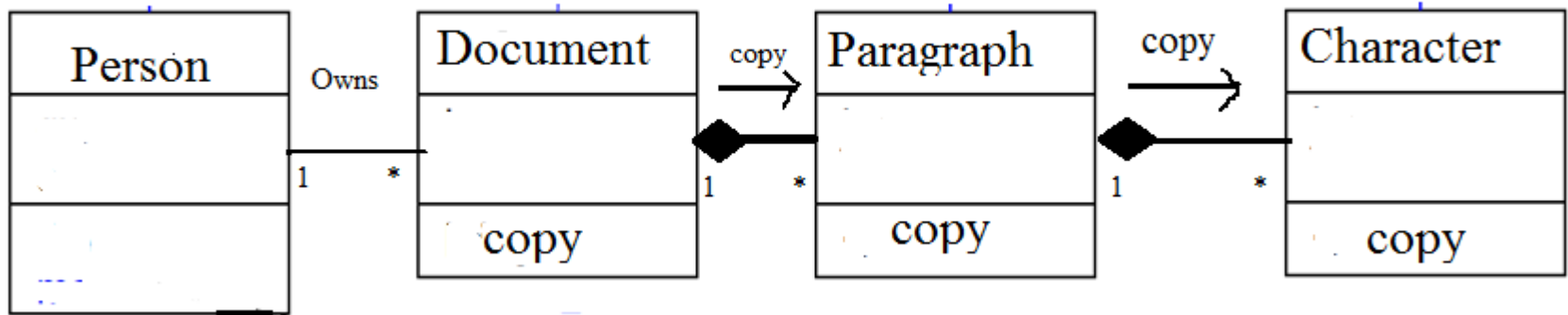
UML notation for representing propagation:

- ▶ You can indicate propagation on class models with a **small arrow** indicating the direction and **operation name**.

Propagation of operation...

For Example:

- ▶ A person owns multiple document,
- ▶ Each document consists of paragraphs that in turn consists of characters.
- ▶ Copying a paragraph copies all the characters in it.



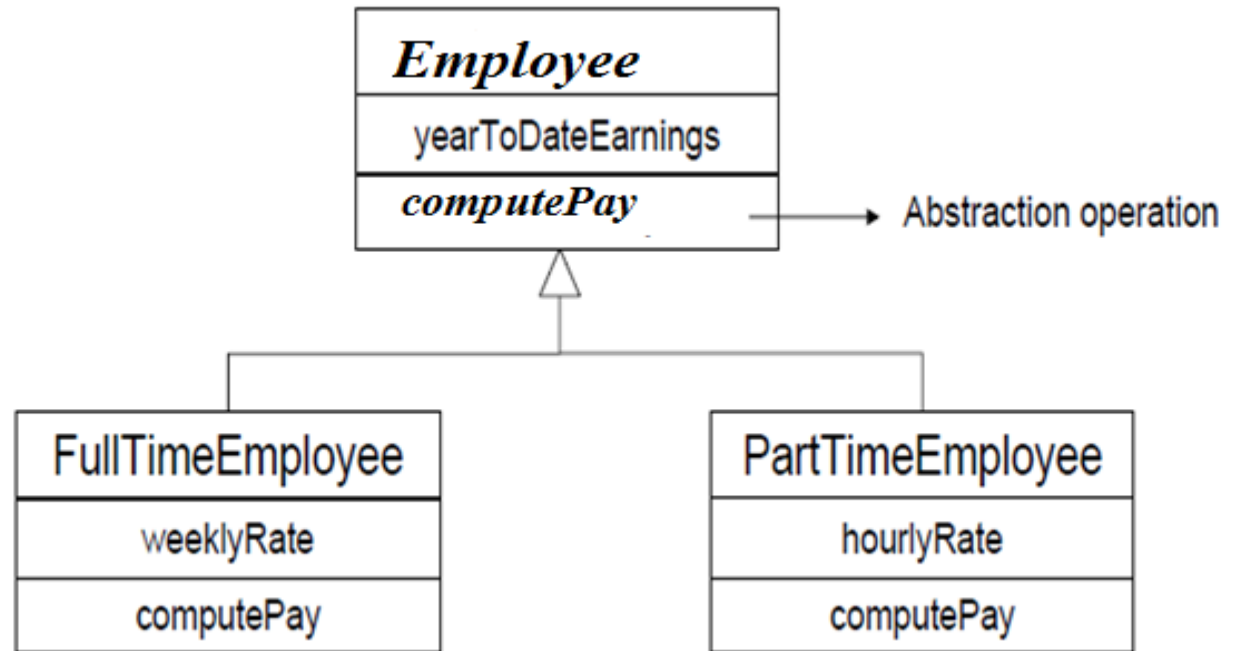
- ▶ Note: the operation **doesnot propagate in reverse direction**

Abstract classes

- ▶ An abstract class is a **class that has no direct instances**.
- ▶ A concrete class is a class that is instanstiable i.e. It can have direct instances.
- ▶ In UML, the name of an abstract class is written in an **italic font**.
- ▶ Signature of an operation for which each concrete subclass must provide its own implementation

Contd..

For Example



In the above diagram there is an abstract class called *Employee*. This class contains one abstract method called *computePay*, it is written in an *italic font*. An abstract method has no implementation.

- ▶ We can use abstract classes to define the methods that can be inherited by subclasses.
- ▶ Abstract operation defines the signature of an operation for which each concrete subclass must provide its own implementation.
- ▶ A concrete class may contain abstract operations, because objects of the concrete class would have undefined operations.

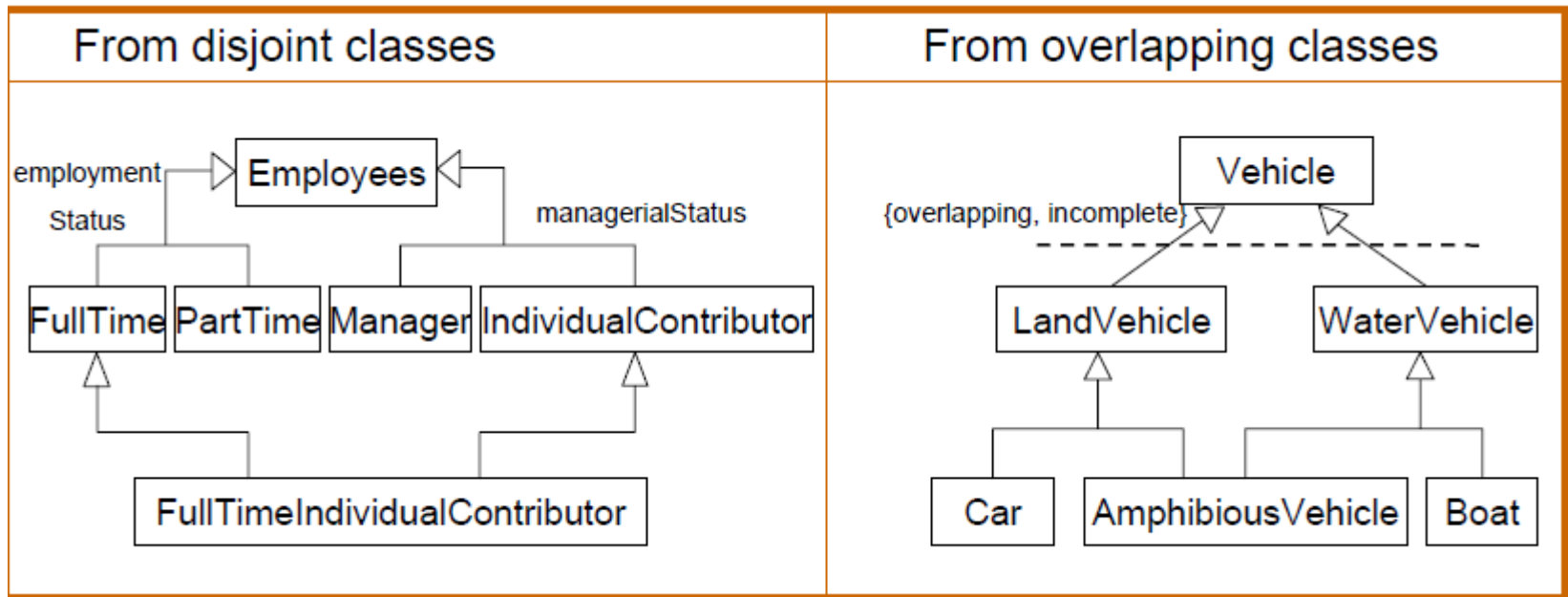
Multiple Inheritance

- ▶ Class has **more than one super class** and to inherit features from all parents.
- ▶ The advantage of multiple inheritance is greater power in specifying classes and an increased opportunity for reuse
- ▶ There are 2 types of multiple inheritance
 - **disjoint classes**
 - **overlapping classes**

Contd..

For Example,

[Go to](#)

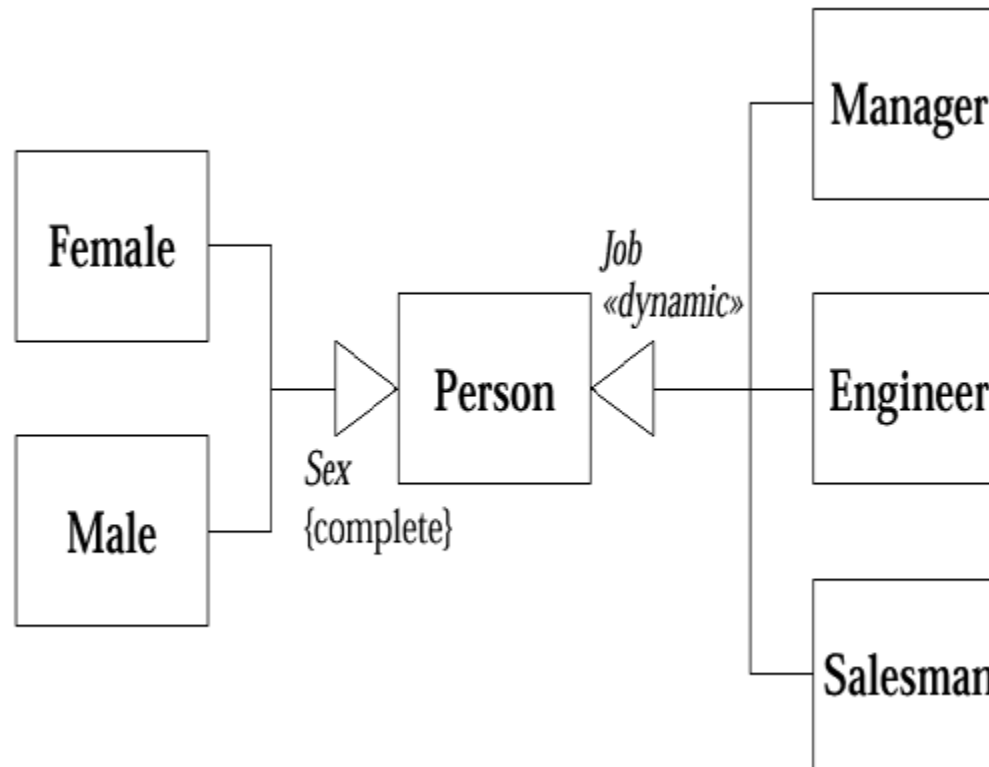


Contd..

- ▶ In the above example FullTimeEmployee and PartTimeEmployee are disjoint. Each employee must belong to exactly one of these.
- ▶ A subclass inherits a feature from the same ancestor class found along more than one path only once.

Multiple classification

- ▶ Consider an example,

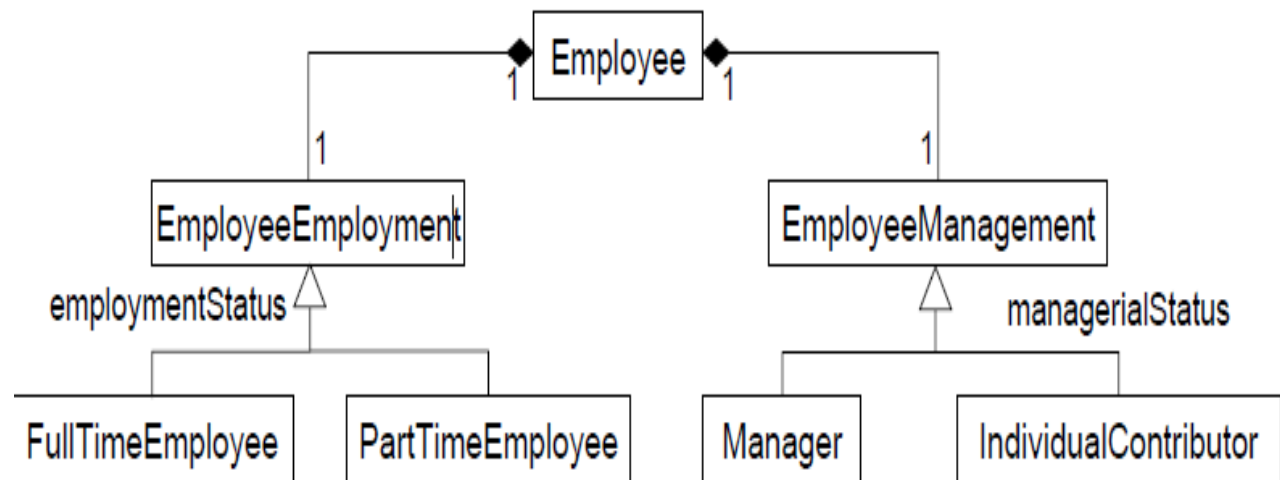


Contd..

- ▶ Object Oriented **does not handle** the multiple classification so, we must use **workarounds**. Here, workaround **replaces the inheritance with delegation**.

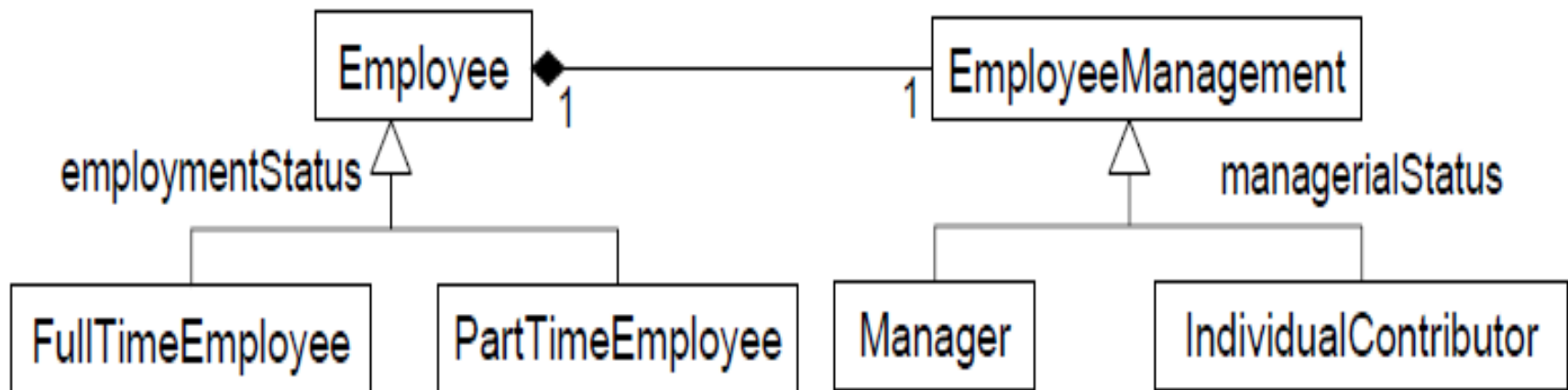
Workarounds

- ▶ Workarounds has some restructuring techniques.
- ▶ Eg: **Delegation using composition** of parts

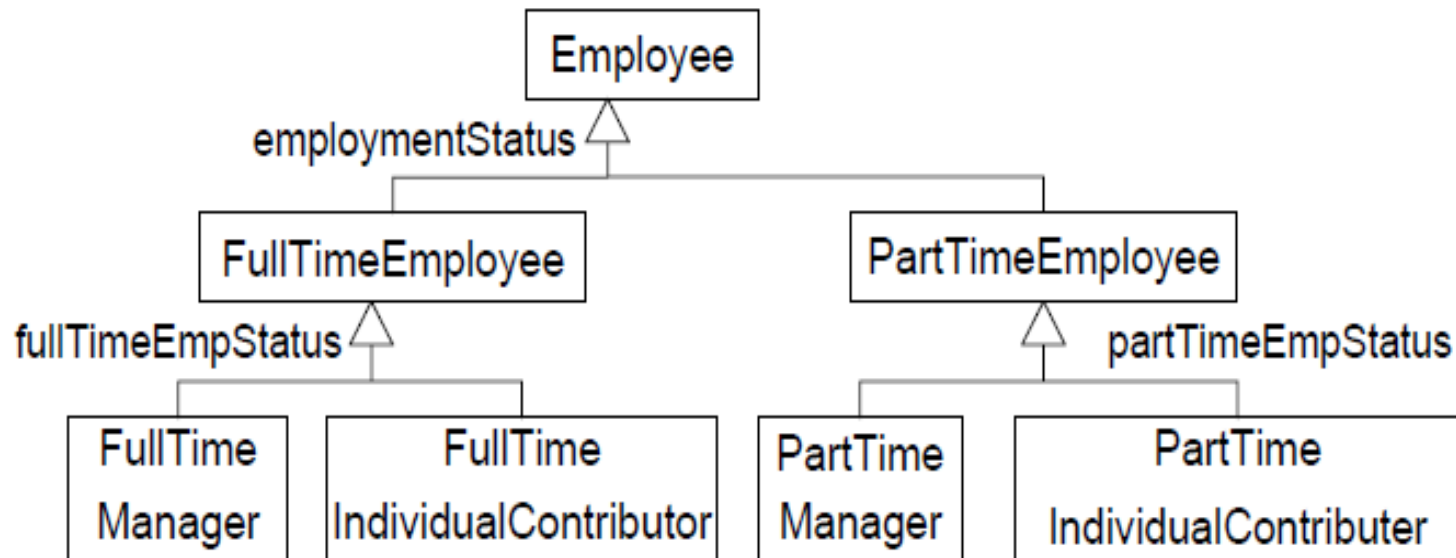


Contd...

➤ **Inheritance and delegation** For example,



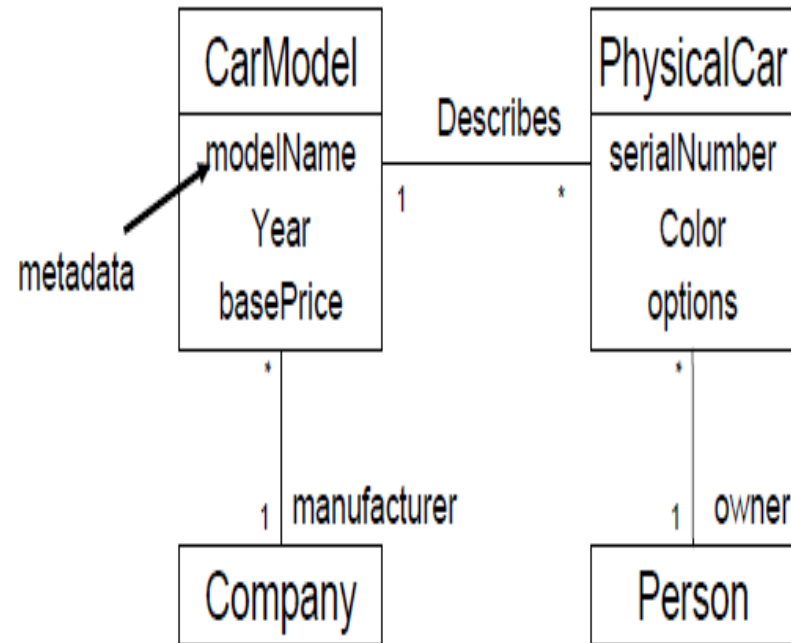
► **Nested generalization** For example,



Metadata

- ▶ It is **data describes the other data**. for example, a class definition is a metadata
- ▶ Many real world applications have metadata, such as a **parts catalogs, blueprints** and **dictionaries**.
- ▶ Class descriptor objects have features, and they in turn have their own classes, which are called metaclasses.

Contd..



➤ In the given example, a car model is a metadata consists of name, year, base price, and a manufacturer etc..

➤ A car model is relative to a physical car, which is data.

Fig 4.7 metadata example

Reification

- ▶ It is a promotion of something that is not an object into an object.
- ▶ It is a helpful technique for meta applications because it lets you shift the level of abstraction.
- ▶ It is also useful to promote attributes, methods, constraints and control information into objects so you can describe and manipulate them as data.

Contd..

- ▶ Consider an example, here it promotes the **substanceName** attribute to a class to capture the many-to-many relationship between **Substance** and **substanceName**.

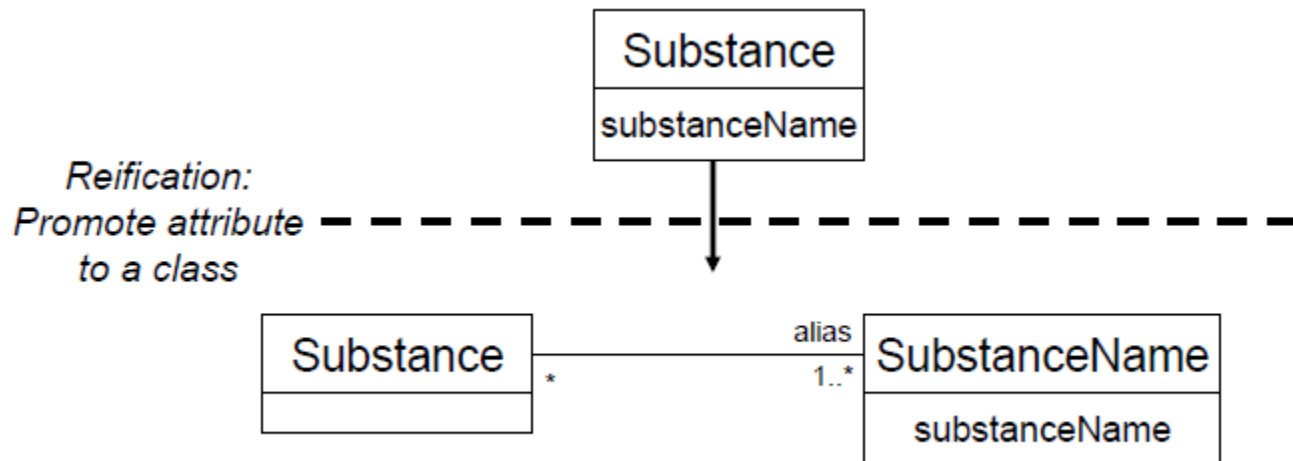


Fig 4.8 Reification

Constraints

- ▶ It is a **Boolean condition involving model elements**, such as objects, classes, attributes, links, associations and generalization.

Constraints on objects

- ▶ **Example** : The following figure shows a portion of DFD for computing the salary of employees of a company that has decided to give incentives to all employees of the sales department and increment the salary of all employees of the HR department.

Contd..

- ▶ It can be seen that the constraint {Dept:Sales} causes **incentive** to be calculated only if the department is sales and the constraint {Dept:HR} causes **increment** to be computed only if the department is HR.

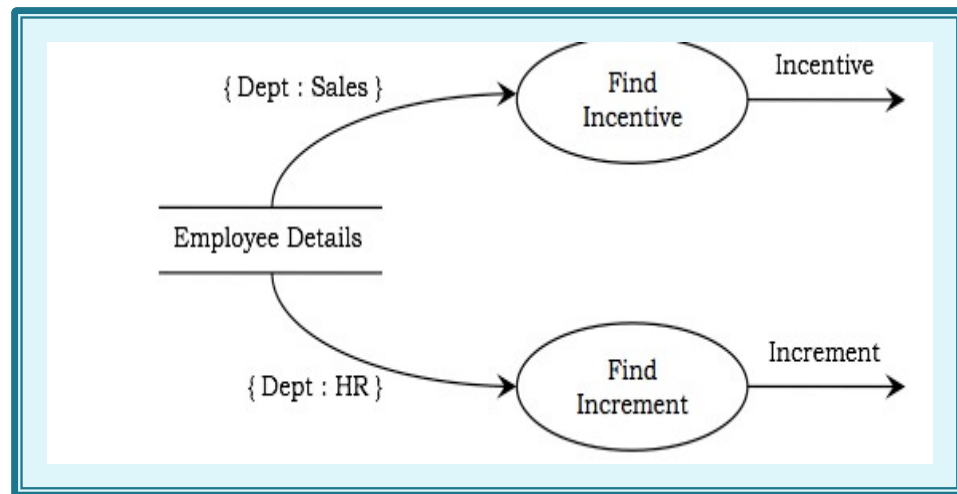


Fig 4.9.1 Constraints on Objects

Constraints on generalization sets

- ▶ **Disjoint:** the subclass are mutually exclusive, each object belongs to exactly one of the subclass.
- ▶ **overlapping:** the subclasses can share some objects. An object may belong to more than one subclass.
- ▶ **Complete:** the generalization lists all the possible subclasses
- ▶ **Incomplete:** the generalization may be missing some subclasses.

For Ex:

Contd..

Constraints on links

Multiplicity, qualification, associations

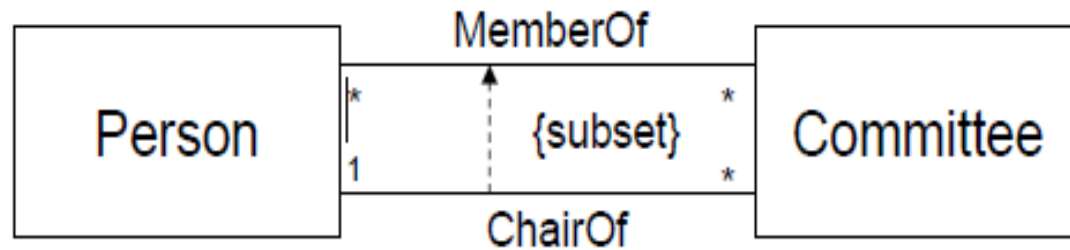
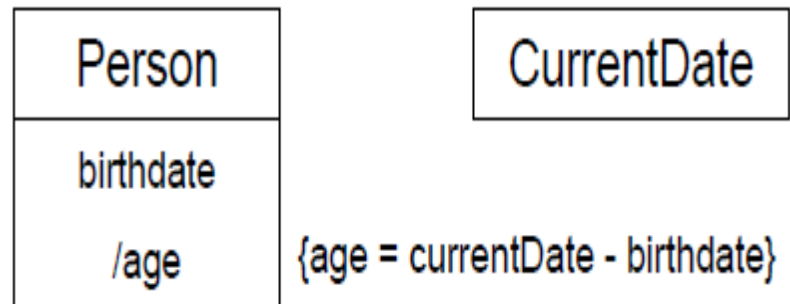


Fig 4.9.2 Constraints between associations

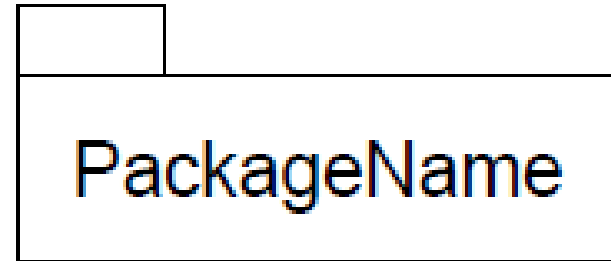
Derived Data

- ▶ It is a **function of one or more elements**, which in turn may be derived.
- ▶ Classes, associations and attributes may be derived.
- ▶ The notation for derived element is a **slash (/)**
- ▶ In the given example age can be derived from birth date and the current date.



Packages

- ▶ It is a group of elements (classes, associations, generalizations, and lesser package)
- ▶ Notation for a package is



Tips for Package

- Carefully delineate each package's scope
- Define each class in a single package
- Make packages cohesive

End of Class Modeling

