CHAPTER2: CLASS MODELLING

Note: A class model captures the static structure of a 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 CONCEPT

Objects

Purpose of class modeling is todescribe objects.

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

Ex: Joe Smith, Infosys Company, process number 7648 and top window are objects.

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 and semantics.

Ex: Person, company, process and window are classes.

Note: All objects have identity and are distinguishable. Two apples with same color, shape and texture are still individual apples: a person can eat one and then the other. The term identity means that the objects are distinguished by their inherent existence and not by descriptive properties that they may have.

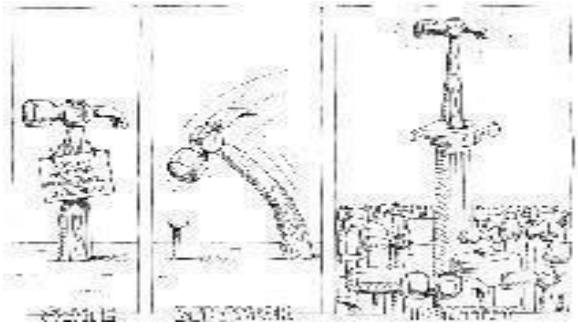
CLASS MODELLING

OBJECT AND CLASS CONCEPT

An object has three characteristics: state, behavior and a unique identification. or An object is a concept, abstraction or thing with identity that has meaning foran application.

Eg:

Note: The term identity means that the objects are distinguished by their inherent existence and not by descriptive properties that they may have.



Class diagrams

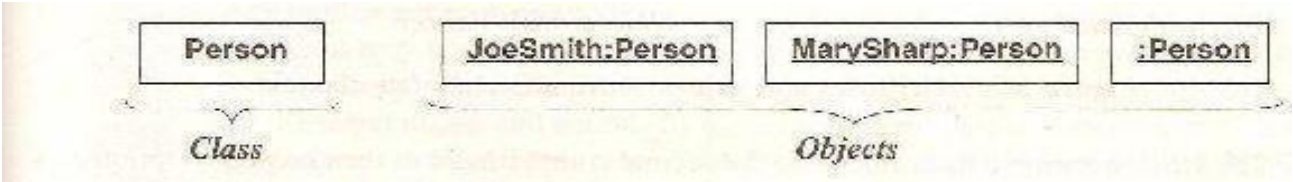
Class diagrams provide a graphic notation for modeling classes and their relationships, thereby describing possible objects.

Note: An object diagram shows individual objects and their relationships. Useful for documenting test cases and discussing examples.

Class diagrams are useful both for abstract modeling and for designing actual programs.

Note: A class diagram corresponds to infinite set of object diagrams.

Figure below shows a class (left) and instances (right) described by it.



Conventions used (UML):

UML symbol for both classes and objects is box.

Objects are modeled using box with object name followed by colon followedby class name.

Use boldface to list class name, center the name in the box and capitalize thefirst letter. Use singular nouns for names of classes.

To run together multiword names (such as JoeSmith), separate the words with intervening capital letter.

Values and Attributes: Value is a piece of data.

Attribute is a named property of a class that describes a value held by each object ofthe class.

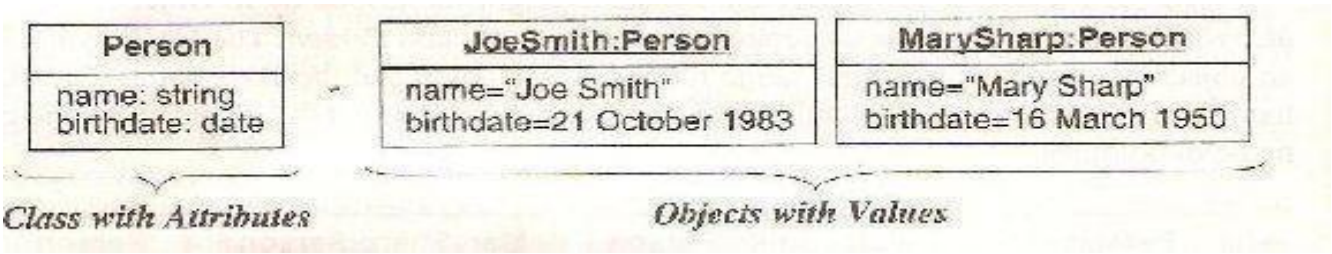
Following analogy holds:

Object is to class as value is to attribute.

E.g. Attributes: Name, bdate, weight.

Values: JoeSmith, 21 October 1983, 64. (Of person object).

Fig shows modeling notation



Conventions used (UML):

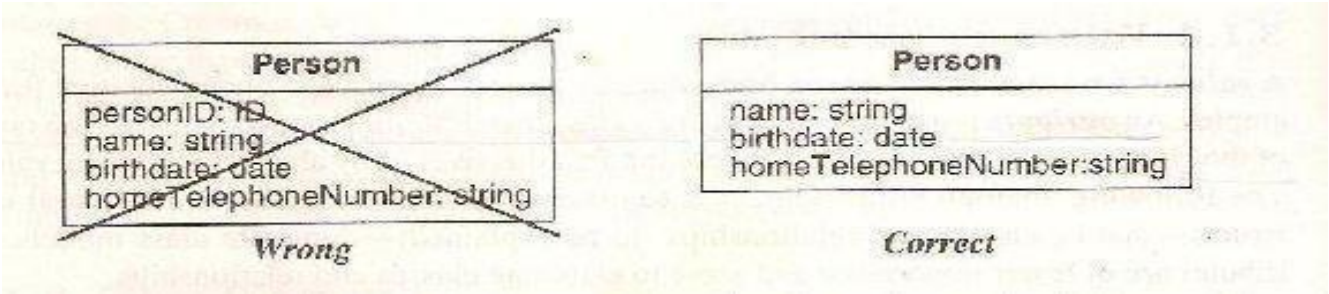
List attributes in the 2nd compartment of the class box. Optional details (like default value) may follow each attribute.

A colon precedes the type, an equal sign precedes default value.

Show attribute name in regular face, left align the name in the box and use small case for the first letter.

Similarly we may also include attribute values in the 2nd compartment of object boxes with same conventions.

Note: Do not list object identifiers; they are implicit in models.E.g.



An **operation** is a function or procedure that maybe applied to or by objects in aclass.

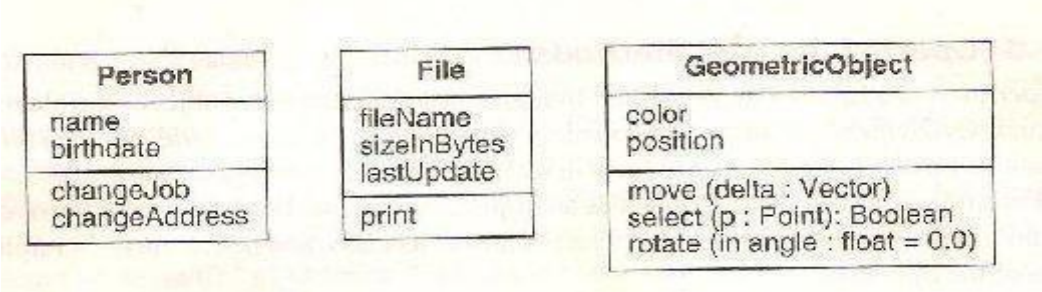
E.g. Hire, fire and pay dividend are operations on Class Company. Open, close, hideand redisplay are operations on class window.

A **method** is the implementation of an operation for a class.

E.g. In class file, print is an operation you could implement different methods toprint files.

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

Fig shows modeling notation.



UML conventions used –

List operations in 3rd compartment of class box.

List operation name in regular face, left align and use lower case for first letter.

Optional details like argument list and return type may follow each operation name.

Paranthesis 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 amongobjects of same class.

Summary of Notation for classes

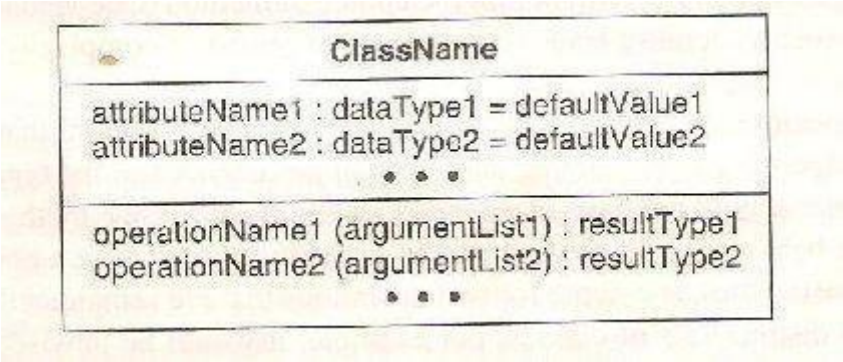


Fig: Notation for an argument of an operation

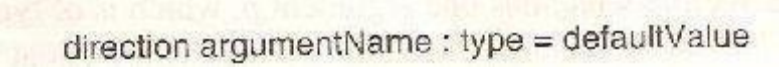


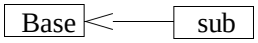
Fig: Notation for an argument of an operation

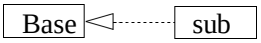
Class Digarms: Relationships

Classes can related to each otherthrough different relationships:

– Dependency 

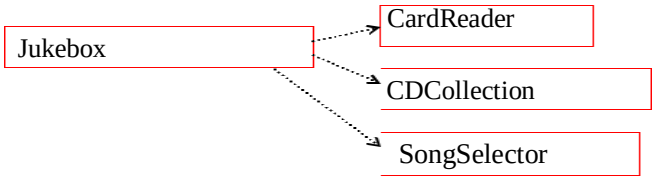
– Association (delegation) Class1 

– Generalization (inheritance) 

– Realization (interfaces) 

Dependency: A Uses Relationship Dependencies

- occurs when one object depends on another
- if you change one object's interface, youneed to change the dependent object
- arrow points from dependent to neededobjects



2)Association: Structural Relationship

- **Association**
  - a relationship between classes indicates some meaningful and interesting connection
  - Can label associations with a hyphen connected verb phrase which reads well between concepts

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

LINK AND ASSOCIATION CONCEPTS

Note: Links and associations are the means for establishing relationships among objects and classes.

Links and associations

A **link** is a physical or conceptual connection among objects.

E.g. JoeSmith *WorksFor* Simplex Company.

Mathematically, we define a link as a tuple– that is, a list of objects.

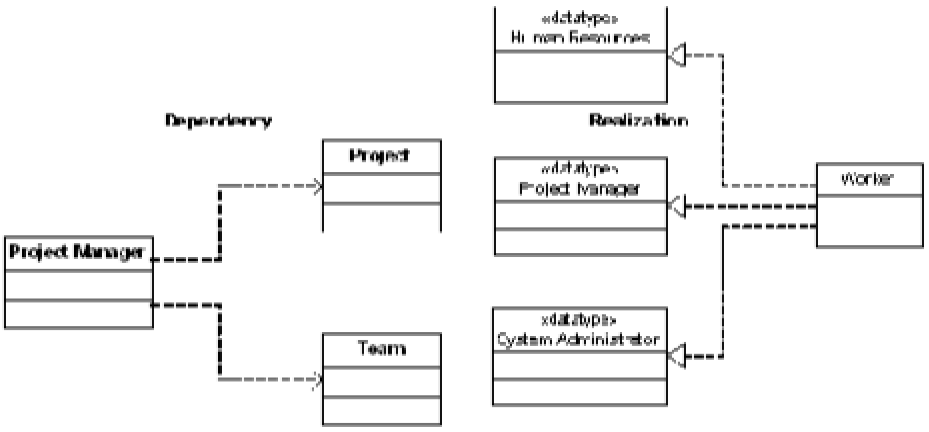
A link is an instance of an**association**.

An **association** is a description of a group of links with common structure and common semantics.

E.g. a person *WorksFor* a company.

An association describes a set of potential links in the same way that a class describes a set of potential objects.

Class Diagrams (cont)



Co The source class depends on (uses) the target class

Class supports all operations of target class but not all attributes or associations.

Lin k has the name, it is
und

Association connects related classes and is also denoted by a line.Show link and association names in italics.

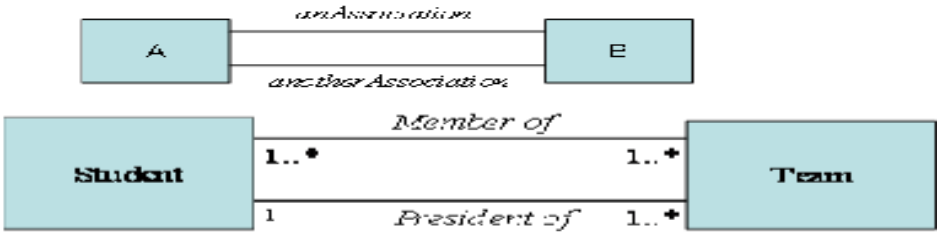
Note:

Association name is optional, if the model is unambiguous. Ambiguity arises when a model has multiple associations among same classes.

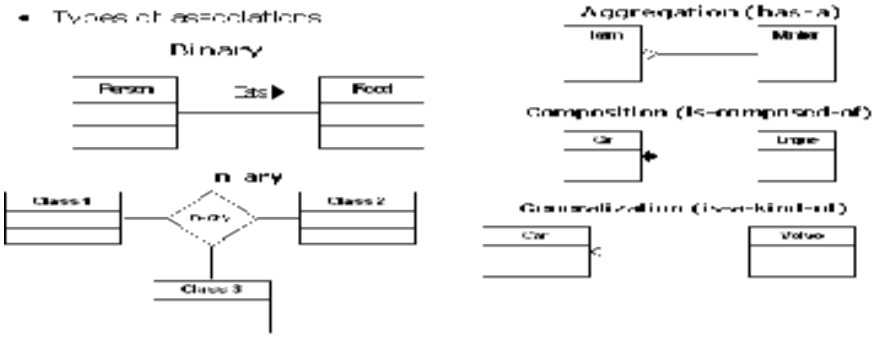
Developers often implement associations in programming languages as references from one object to another. A reference is an attribute in one object that refers to another object.

Association Relationships

We can specify dual associations.



Class Diagrams (cont)



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:

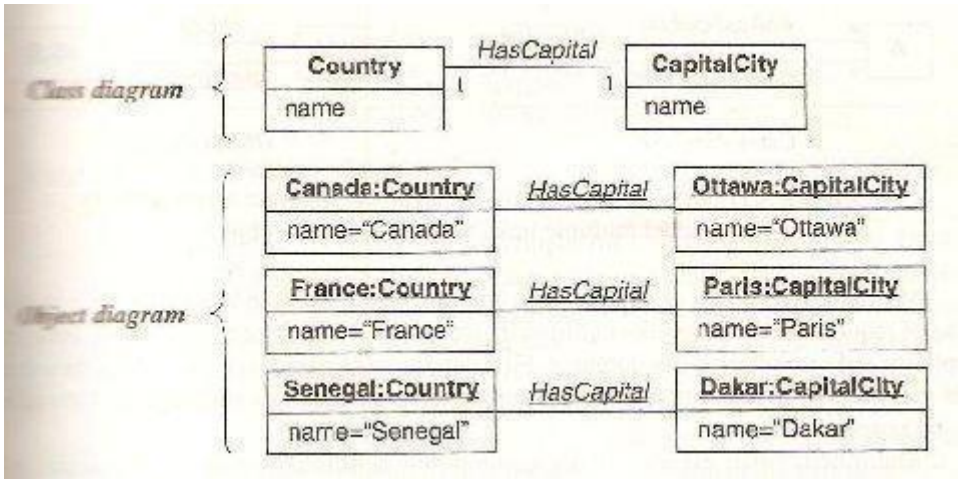
UML diagrams explicitly lists multiplicity at the ends of association lines.

UML specifies multiplicity with an interval, such as"1" (exactly one).

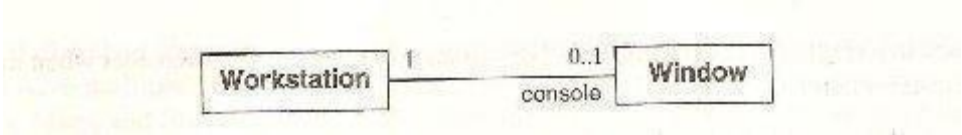
"1.."(one or more).

"3..5"(three to five, inclusive)." * " (many, i.e zero or more).

Previous figure illustrates many-to-many multiplicity. Below figure illustrates one-to-one multiplicity



Below figure illustrates zero-or-one multiplicity.



Note 1: Association vs Link

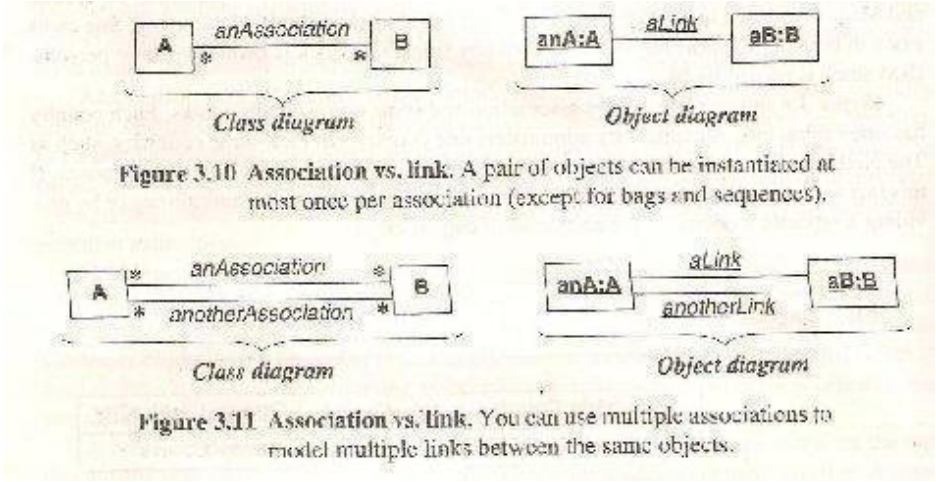


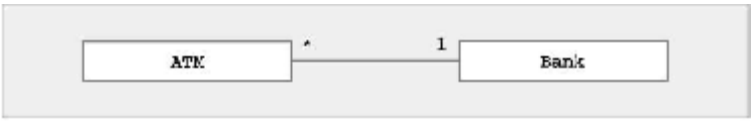
Figure 3.10 Association vs. link. A pair of objects can be instantiated at most once per association (except for bags and sequences).

Figure 3.11 Association vs. link. You can use multiple associations to model multiple links between the same objects.

Multiplicity of Associations

Many-to-one

– Bank has many ATMs, ATM knows only 1 bank

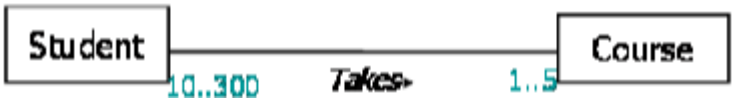


One-to-many

– Inventory has many items, items know 1 inventory

Association - Multiplicity

- A Student can take up to five Courses.
- Student has to be enrolled in at least one course.
- Up to 300 students can enroll in a course.
- A class should have at least 10 students.



–

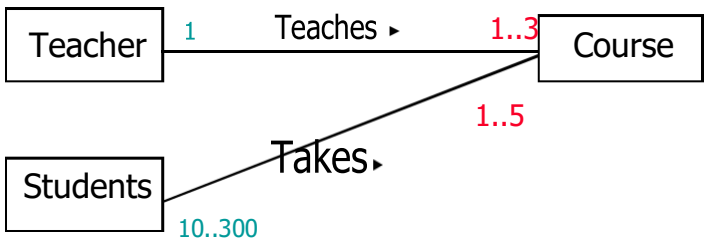
Association - Multiplicity

A teacher teaches 1 to 3 courses (subjects)

Each course is taught by only one teacher.

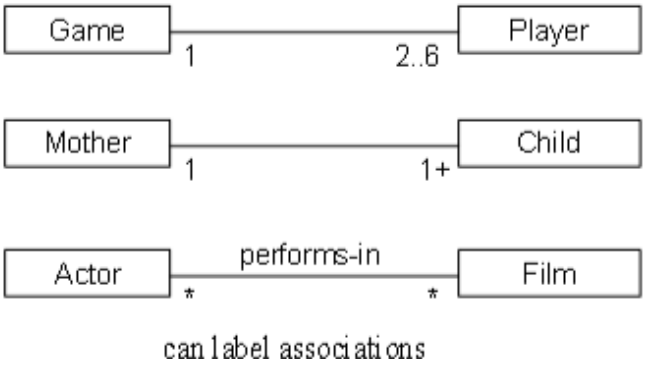
A student can take between 1 to 5 courses.

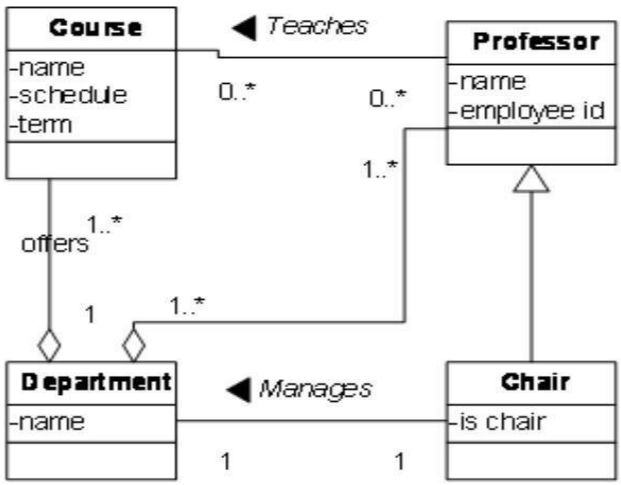
A course can have 10 to 300 students.



Multiplicity

- Multiplicity defines how many instances of type A can be associated with one instance of type B at some point





MULTIPLICITIES IN ASSOCIATIONS

min..max notation (related to at least min objects and at most max objects)	0..*	related to zero or more objects
	0..1	related to no object or at most one object
	1..*	related to at least one object
	1..1	related to exactly one object.
	3..5	related to at least three objects and at most five objects
short hand notation	1	same as 1..1
	*	same as 0..*

Note 2: 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.

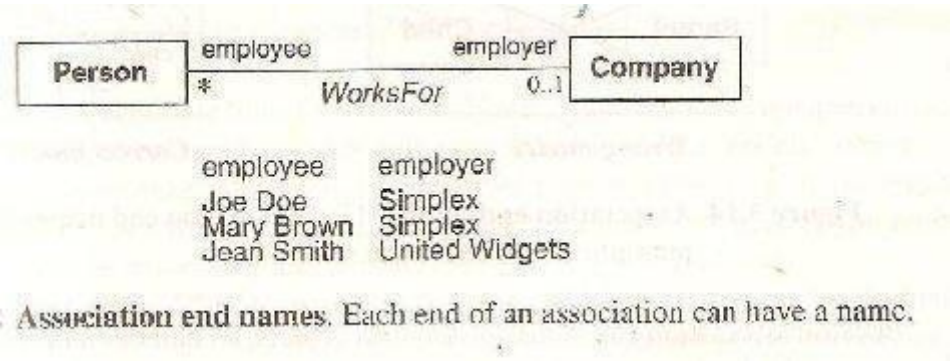
Note 3: The literature often describes multiplicity as being "one" or "many", but more generally it is a subset of the non negative numbers.

Association end names

Multiplicity implicitly refers to the ends of associations. For E.g. A one-to-many association has two ends –

an end with a multiplicity of "one" an end with a multiplicity of "many"

You can not only assign a multiplicity to an association end, but you can give it aname as well.



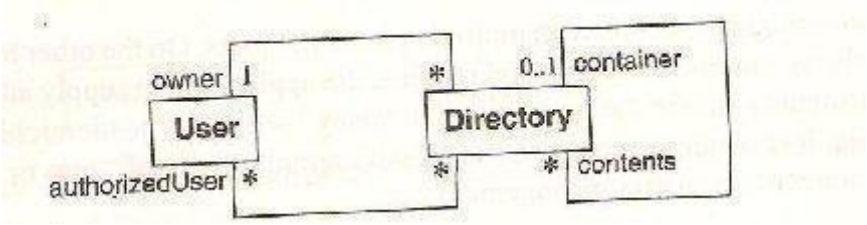
Association end names. Each end of an association can have a name.

A person is an employee with respect to company. A company is an employer with respect to a person.

Note 1: Association end names are optional.

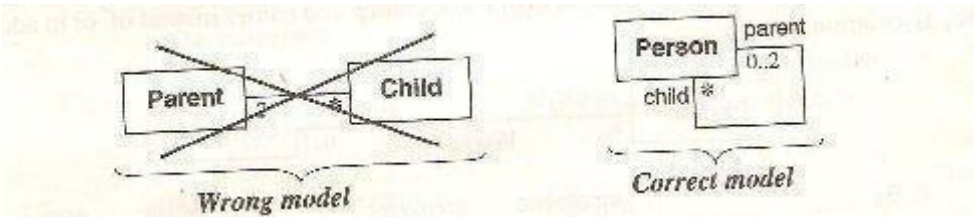
Note 2: Association end names are necessary for associations between two objects of the same class. They can also distinguish multiple associations between a pair of classes.

E.g. each directory has exactly one user who is an owner and many users who are authorized to use the directory. When there is only a single association between apair of distinct classes, the names of the classes often suffice, and you may omit association end names.



Note 3: Association end names let you unify multiple references to the same class.

When constructing class diagrams you should properly use association end names and not introduce a separate class for each reference as below fig shows.



Sometimes, the objects on a "many" association end have an explicit order.

E.g. Workstation screen containing a number of overlapping windows. Each window on a screen occurs at most once. The windows have explicit order so only the top most windows are visible at any point on the screen.

Ordering is an inherent part of association. You can indicate an ordered set of objects by writing "{ordered}" next to the appropriate association end.

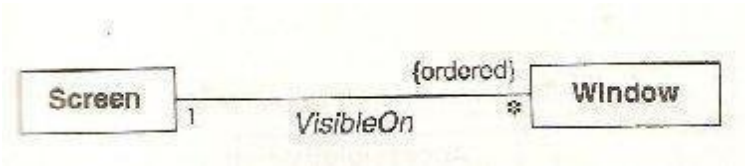


Fig: ordering sometimes occurs for "many" multiplicity

Bags and Sequences

Normally, a binary association has **at most one link** for a pair of objects. However, you 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.

Example:

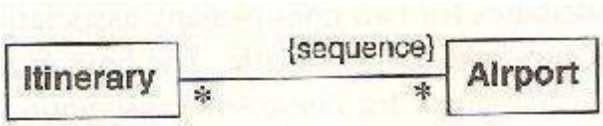


fig: an itinerary may visit multiple airports, so you should use {sequence} and not {ordered}

Note: {ordered} and {sequence} annotations are same, except that the first disallows duplicates and the other allows them.

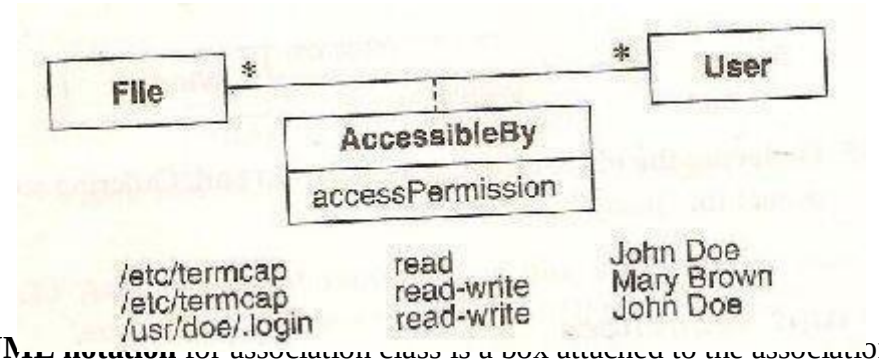
Association classes

An **association class** is an association that is also a class.

Like the links of an association, the instances of an association class derive identity from instances of the constituent classes.

Like a class, an association class can have attributes and operations and participate inassociations.

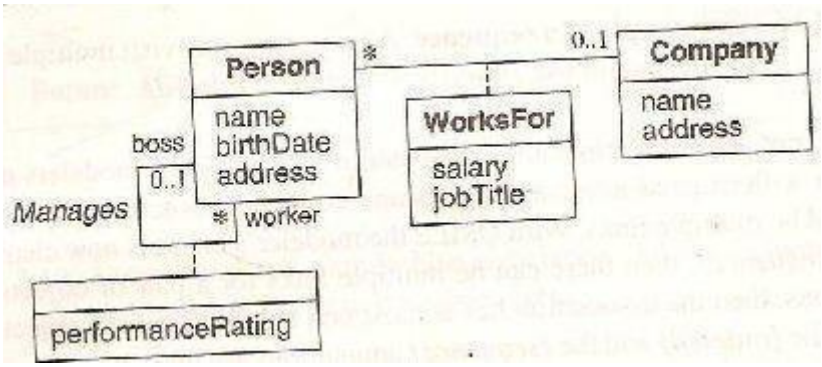
Ex:



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

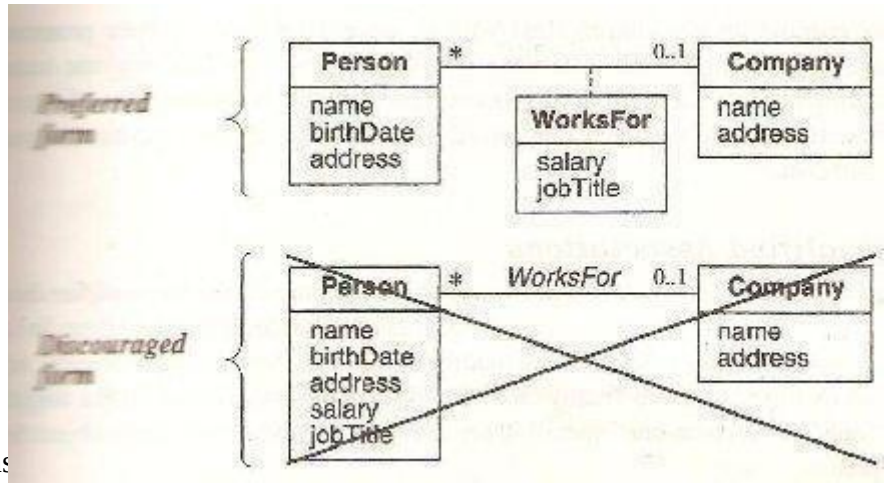
Note: Attributes for association class unmistakably belong to the link and cannot be ascribed to either object. In the above figure, access Permission is a joint property of File and user cannot be attached to either file or user alone without losing information.

Below figure presents attributes for two one-to-many relationships. Each person working for a company receives a salary and has job title. The boss evaluates the performance of each worker. Attributes may also occur for one-to-one associations.

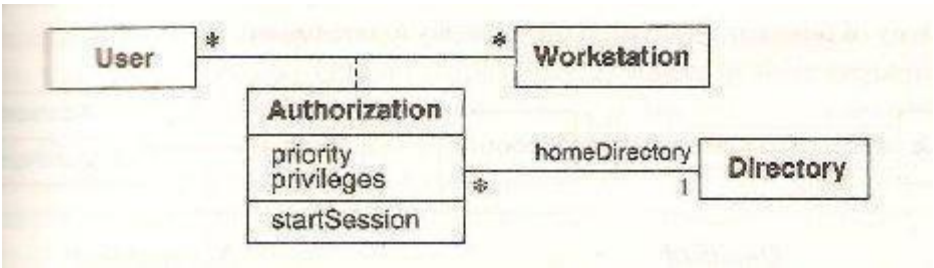


Note 1: Figure shows how it's possible to fold attributes for one-to-one and one- to-many associations into the class opposite a "one" end. This is not possible for many-to-many associations.

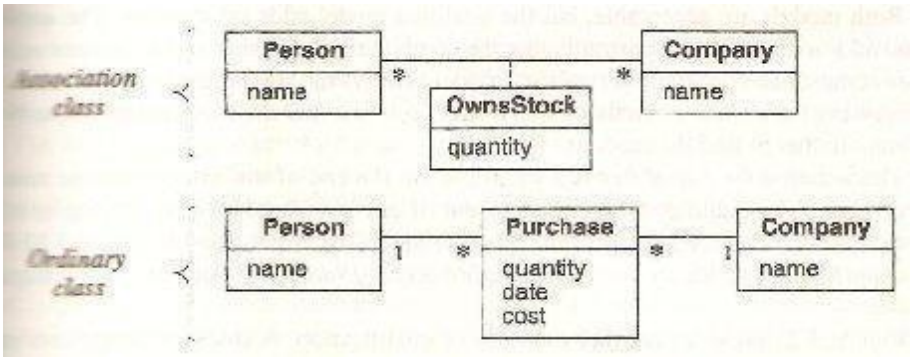
As a rule, you should not fold such attributes into a class because the multiplicity of the association may change.



Note 2: An as



Note 3: Association class vs ordinary class.



eg:

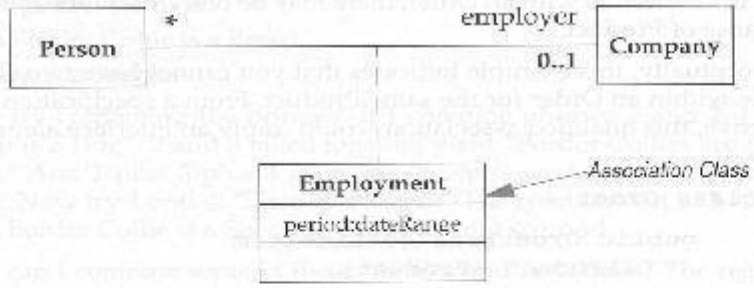
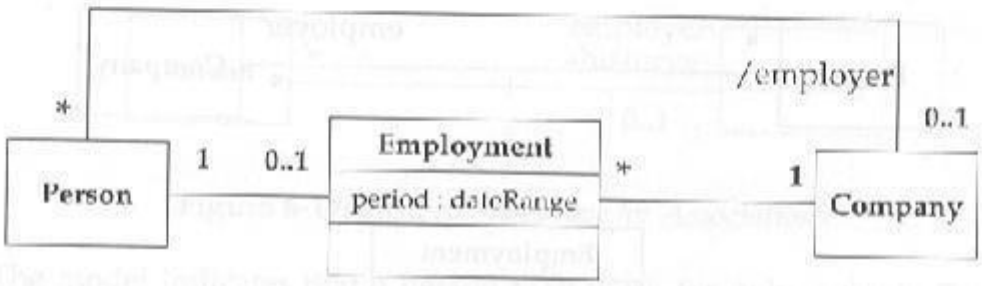


Figure 6-14: *Association Class*



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 and many-to-many associations.

A qualifier selects among the target objects, reducing the effective multiplicity from "many" to "one".

Ex 1: qualifier for associations with one to many multiplicity. A bank services multiple accounts. An account belongs to single bank. Within the context of a bank, the Account Number specifies a unique account. Bank and account are classes, and Account Number is a qualifier. Qualification reduces effective multiplicity of this association from one-to-many to one-to-one.

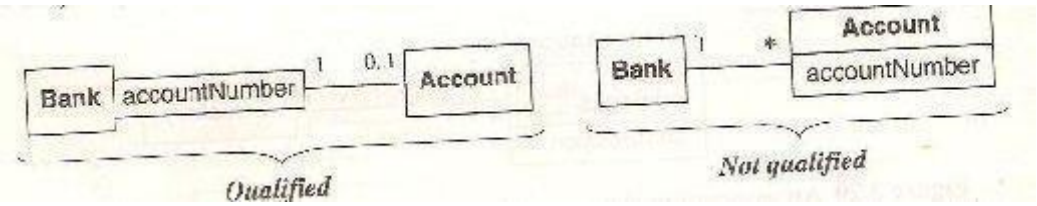
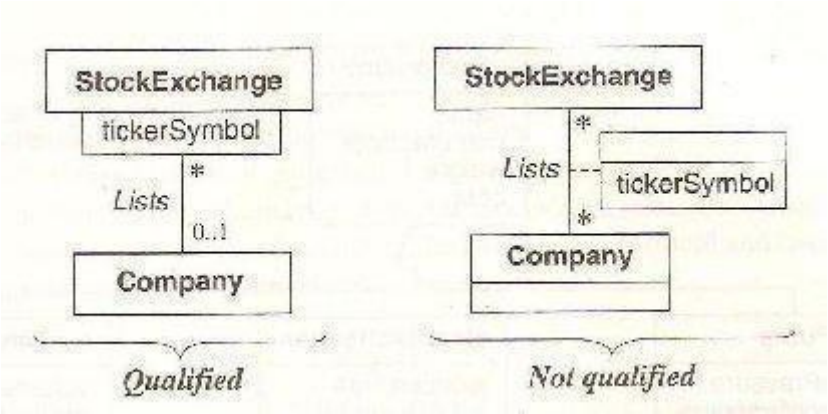
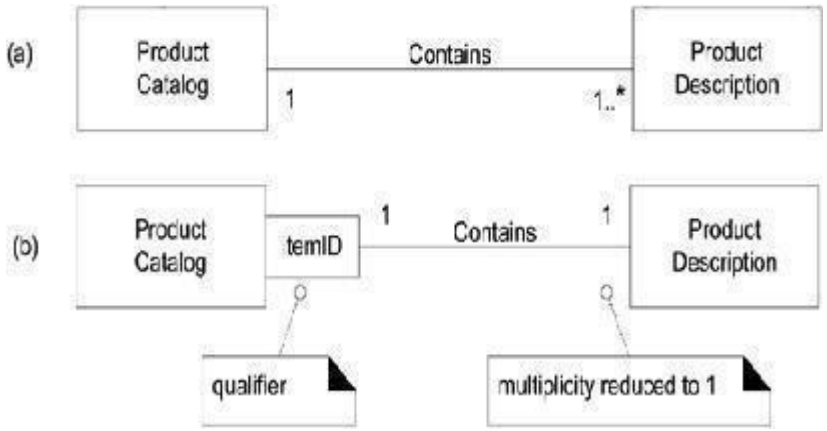


Fig: qualification increases the precision of a model. (note: however, both are acceptable)

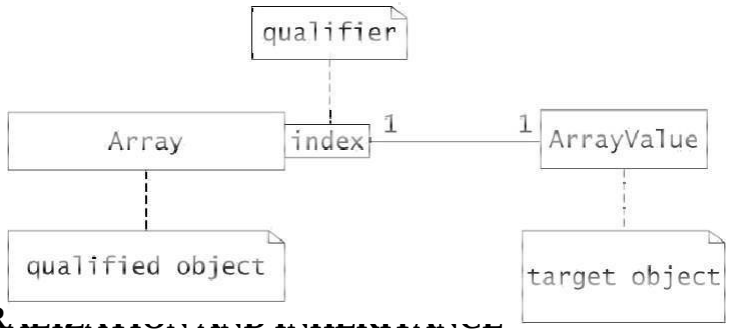
Ex 2: a stock exchange lists many companies. However, it lists only one company with a given ticker symbol. A company maybe listed on many stock exchanges, possibly under different symbols.



Eg 3: Qualified Association



eg 4:



GENER

Generalization is the relationship between a class (the superclass) and one or more variations of the class (the subclasses). Generalization organizes classes by their similarities and differences, structuring the description of objects.

The superclass holds common attributes, operations and associations; the subclasses add specific attributes, operations and associations. Each subclass is said to **inherit** the features of its superclass.

There can be **multiple levels** of generalization.

Fig(a) and Fig(b) (given in the following page) shows examples of generalization.

Fig(a) – Example of generalization for equipment.

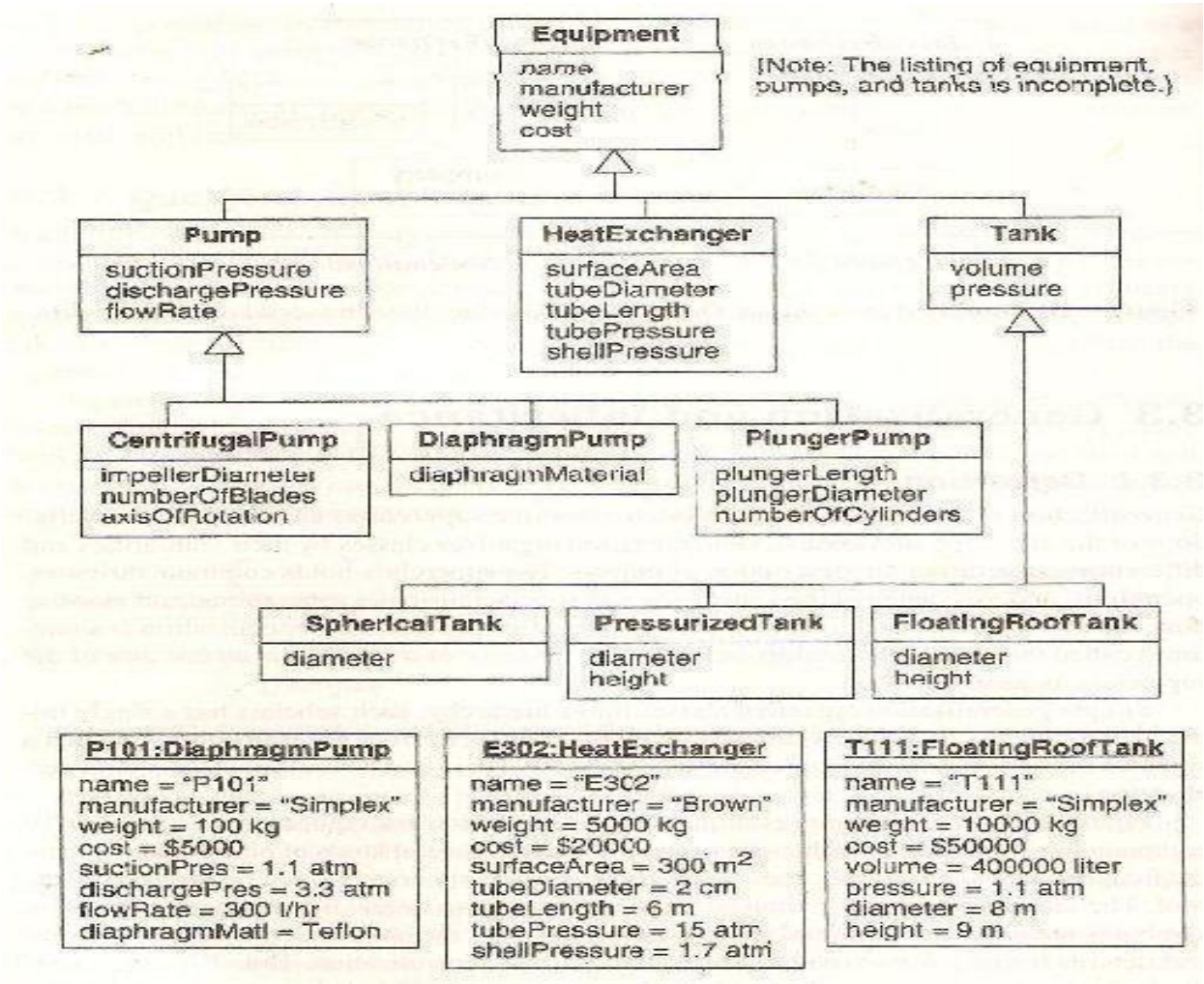
Each object inherits features from one class at each level of generalization.

UML convention used:

Use large hollow arrowhead to denote generalization. The arrowhead points tosuperclass.

Fig(b) – inheritance for graphic figures.

The word written next to the generalization line in the diagram (i.e dimensionality) is a generalization set name. A generalization set name is an enumerated attribute that indicates which aspect of an object is being abstracted by a particular generalization. It is optional.


Equipment
name
manufacturer
weight
cost
[Note: The listing of equipment, pumps, and tanks is incomplete.)
Pump
suctionPressure
dischargePressure
flowRate
HeatExchanger
surfaceArea
tubeDiameter
tubeLength
tubePressure
shellPressure
Tank
volume
pressure
CentrifugalPump
impellerDiameter
numberOfBlades
axisOfRotation
DiaphragmPump
diaphragmMaterial
PlungerPump
plungerLength
plungerDiameter
numberOfCylinders
SphericalTank
diameter
PressurizedTank
diameter
height
FloatingRoofTank
diameter
height
P101:DiaphragmPump
name = "P101"
manufacturer = "Simplex"
weight = 100 kg
cost = $5000
suctionPres = 1.1 atm
dischargePres = 3.3 atm
flowRate = 300 l/hr
diaphragmMatl = Teflon
E302:HeatExchanger
name = "E3C2"
manufacturer = "Brown"
weight = 5000 kg
cost = $20000
surfaceArea = 300 m2
tubeDiameter = 2 cm
tubeLength = 6 m
tubePressure = 15 atm
shellPressure = 1.7 atm
T111:FloatingRoofTank
name = "T111"
manufacturer = "Simplex"
weight = 10000 kg
cost = $50000
volume = 400000 liter
pressure = 1.1 atm
diameter = 8 m
height = 9 m

Fig(a)

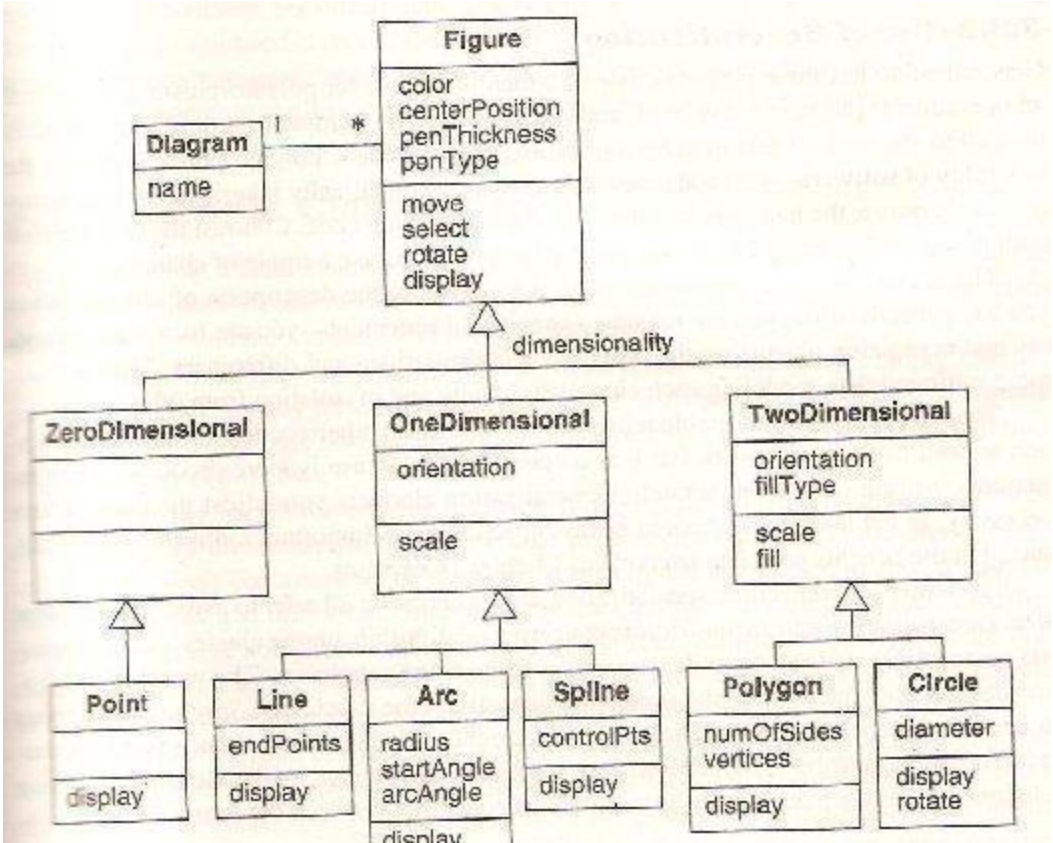


Figure
color
centerPosition
penThickness
penType
move
select
rotate
display
Diagram
name
1
*
dimensionality
ZeroDimensional
OneDimensional
orientation
scale
TwoDimensional
orientation
fillType
scale
fill
Point
display
Line
endPoints
display
Arc
radius
startAngle
arcAngle
display
Spline
controlPts
display
Polygon
numOfSides
vertices
display
Circle
diameter
display
rotate

Fig (b)

'move**',** **'**select**',** 'rotate', and 'display' are operations that all subclasses inherit.
'scale**'** applies to one-dimensional and two-dimensional figures. 'fill' applies only to two-dimensional figures.

Use of generalization: Generalization has three purposes –

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

Note: The terms generalization, specialization and inheritance all refer to aspectsof the same idea.

Overriding features

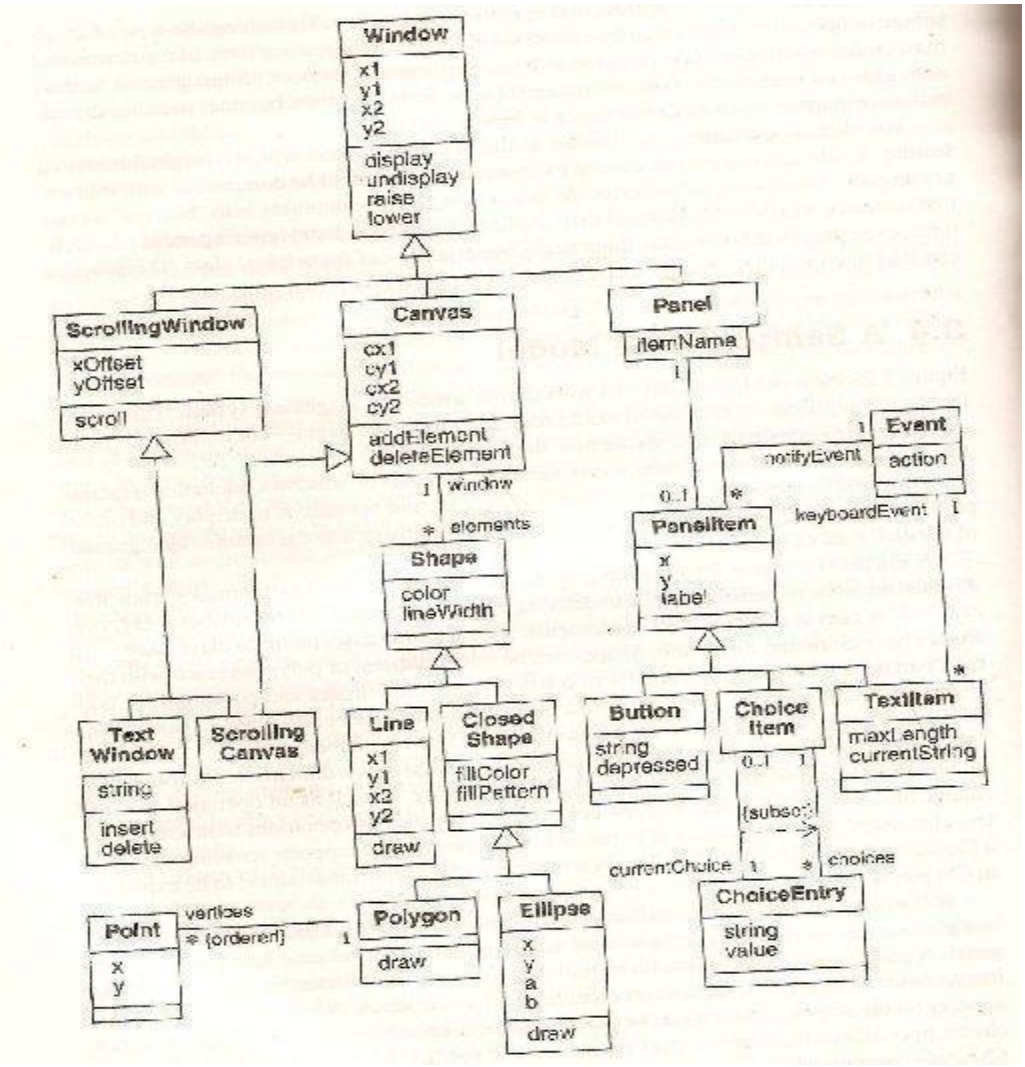
A subclass may override a superclass feature by defining a feature with the same name.
The overriding feature (subclass feature) refines and replaces the overridden feature (superclass feature).

Why override feature?

- To specify behavior that depends on subclass.
- To tighten the specification of a feature.
- To improve performance.
- In fig(b) (previous page) each leaf subclasses had overridden 'display' feature.

Note: You may override methods and default values of attributes. You should never override the signature, or form of a feature.

A SAMPLE CLASS MODEL



NAVIGATION OF CLASS MODELS

Class models are useful for more than just data structure. In particular, **navigation of class model** lets you express certain behavior. Furthermore, navigation exercises a class model and uncovers hidden flaws and omission, which you can then repair.

UML incorporates a language that can be used for navigation, the **object constraint language(OCL).**

OCL constructs for traversing class models

OCL can traverse the constructs in classmodels.

Attributes: You can traverse from an object to an attribute value.

Syntax: source object followed by dot and then attribute name.

Ex: aCreditCardAccount.maximumcredit

Operations: You can also invoke an operation for an object or collection of objects. Syntax:

source object or object collection, followed by dot and then the operation followed by parenthesis even if it has no arguments. OCL has special operations that operate on entire collections (as opposed to operating on each object in a collection). Syntax for collection operation is: source object collection followed by "->", followed by the operation.

Simple associations: Dot notation is also used to traverse an association to a target end. Target end maybe indicated by an association end name, or class name (if there is no ambiguity).

Ex: refer fig in next page.

- aCustomer.MailingAddress yields a set of addresses for a customer (the target end has "many" multiplicity).
- aCreditCardAccount.MailingAddress yields a single address(the target end has multiplicity of "one").

Qualified associations: The expression aCreditCardAccount.Statement [30 November 1999] finds the statement for a credit card account with the statement dateof November 1999. The syntax is to enclose the qualifier value in brackets.

Associations classes: Given a link of an association class, you can find the constituent objects and vice versa.

Generalization: Traversal of a generalization hierarchy is implicit for theOCL notation.

Filters: Most common filter is 'select' operation.

Ex:aStatement.Transaction->select(amount>$100).