The three models

→ From different view point to model a system, we have 3 - models

→ These models are related to one another

→ These models capture important aspects of the system

→ All three models are required combinedly for the complete description of a system (or to model a system)

The three models are :

(i) Class Model

(ii) State model

(iii) Interaction model.

The class Model

The class model represents the static,

→ structural (i.e. data) aspect of a system

→ The class model describes the structure of objects in a system i.e. their identity, their relationship to other objects, their attributes and their operations.

→ class model contains computer constructs i.e. implementation details.

→ class model provides context for the state and interaction models.

→ class model is represented by class diagrams.

class and object

→ A class describes group of objects with the same properties (attributes) and behaviour (operation) and semantics.

→ An object is an instance (or occurance of a class)

→ Thus, the common structure of a set of objects which represent the same kind of entity is described by a class and each object of that kind is said to be an instance of the class.

→ Class provides idea of Abstraction.

Object

→ Object is an instance of a class

→ Once a class is defined, many number of objects are created from that class

→ Each object knows its class i.e. an object's class is implicit property of an object.

→ The objects belonging to a class share the common attributes and behaviour that defined in that class.

→ Object stores data (as value of attributes) and have certain operations to manipulate that stored data.

→ Object is something which has

- State
- Behaviour

and - Identity

State

→ Objects are containers of data

→ In object oriented system all the data manipulated by the system is stored in objects.

→ The data value contained in an object's attribute is known as the object's state.

→ object's state is not static. The state will change when object's data changes.

Behaviour

→ Each object provides a no. of operations as an interface.

→ These operations are known as behaviour of the object.

→ Generally, some of these operations will provide access and update functions for the data stored in the object and other operations will implement system's functionality

Identity

→ Every object is distinguishable from every other object

→ objects belonging to a class are similar in nature, but they are not same

→ So, every object is provided with a unique identity which is used to distinguish the objects from other objects.

Class Diagram

→ UML notation (graphical notation) for modeling classes and their relationship is known as class diagram.

→ The class diagrams are useful for both abstract modeling and for designing actual programs.

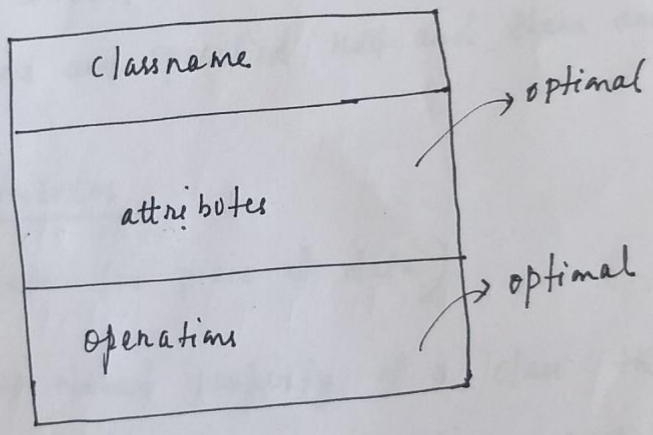
UML Notation

→ UML notation for a class is a box

→ Class names are singular nouns

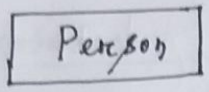
→ class name should be in bold face, mention at the centre of the box and the first letter should be capitalized.

→ Generally, a class diagram is represented by a box with three compartments

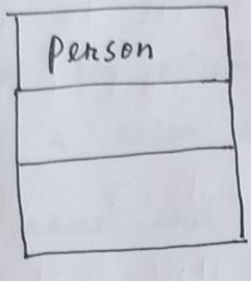


attributes and operations compartment is optional i.e. you may or may not specify it.

A missing attribute compartment means attributes are unspecified and a missing operations compartment means operations are unspecified.



attributes and operations are not specified

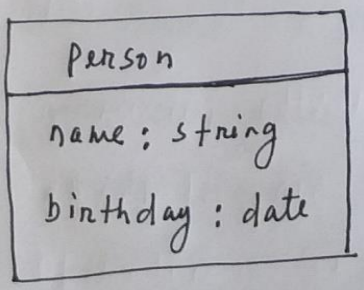


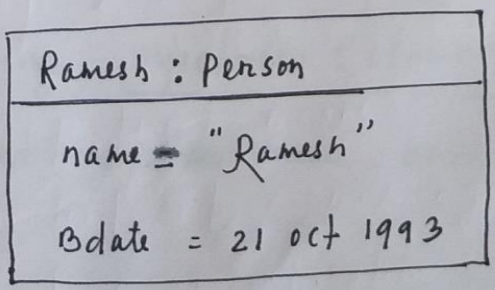
Empty compartment means attributes and operations are specified and there are none.

Values and attributes

Value → data (or piece of data)

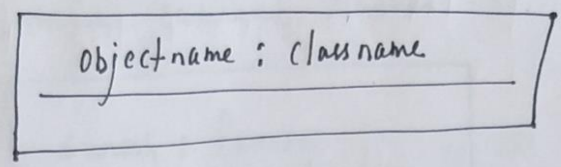
Attribute → Named property of a class that describes values held by each object of the class.





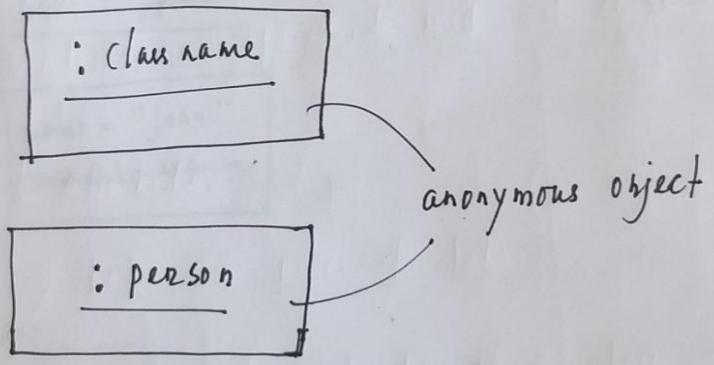
Object diagram

UML notation



UML symbol for object is a box with an object name followed by a colon and a class name.

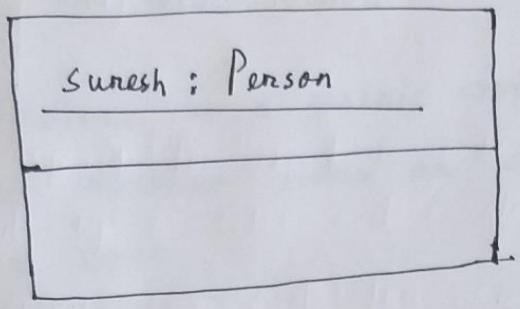
The object name and class name both should be underlined

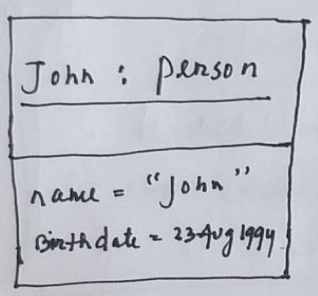


An object diagram is represented by a rectangle with two compartments. The upper compartment is containing objectname : classname

(objectname followed by class name, both are underlined)

The lower compartment of the object diagram contains the attribute names of an object and their current values.





Operations

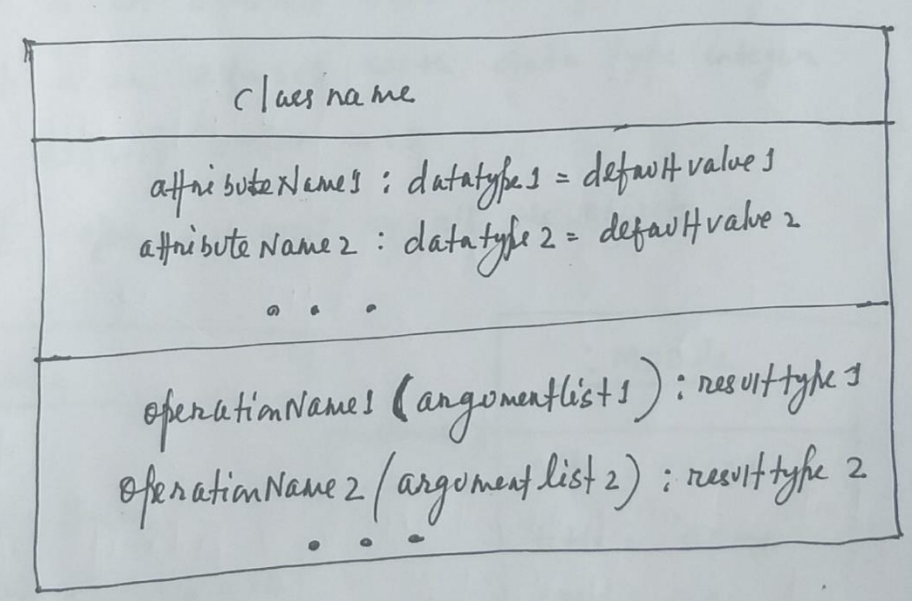
→ An operation is a function or procedure that may be applied to or by objects in a class

→ A method is the implementation of an operation in a class.

e.g. class File may have an operation Print. We implement different methods to Print ASCII files, binary files, Picture files.

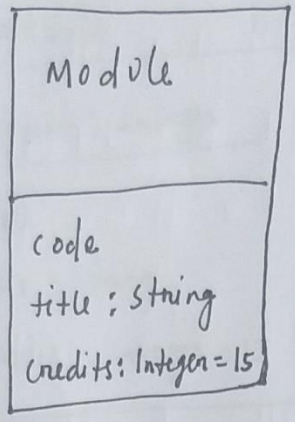
→ When an operation has methods on several classes, the methods have the same signature. The number and type of arguments and the type of result value.

→ feature is a generic word for classes to represent either an attribute or operation or both



Visibility level

+	public	for both attributes & operations
−	private	
#	Protected	
~	package	



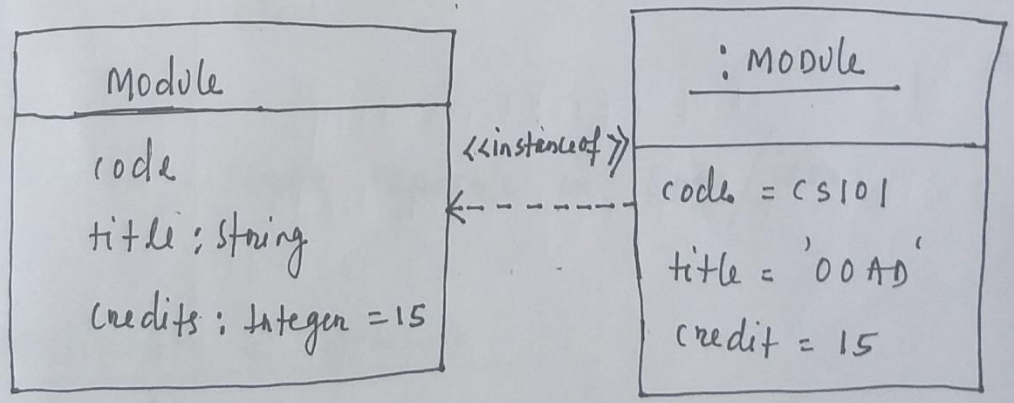
No decission has been made yet for the data type of code.

code is the name of an attribute with no type specified

title is an attribute with datatype string

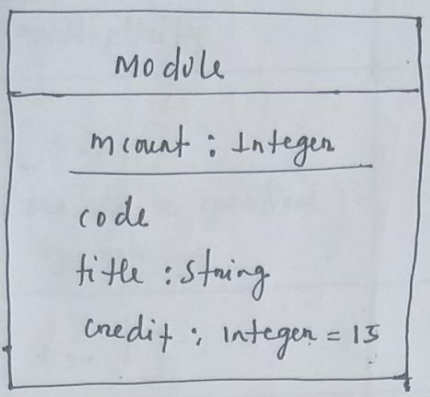
Credit is an attribute with data type integer and default value = 15.

default value is shared by all the objects.



(Class diagram)

(Object diagram)



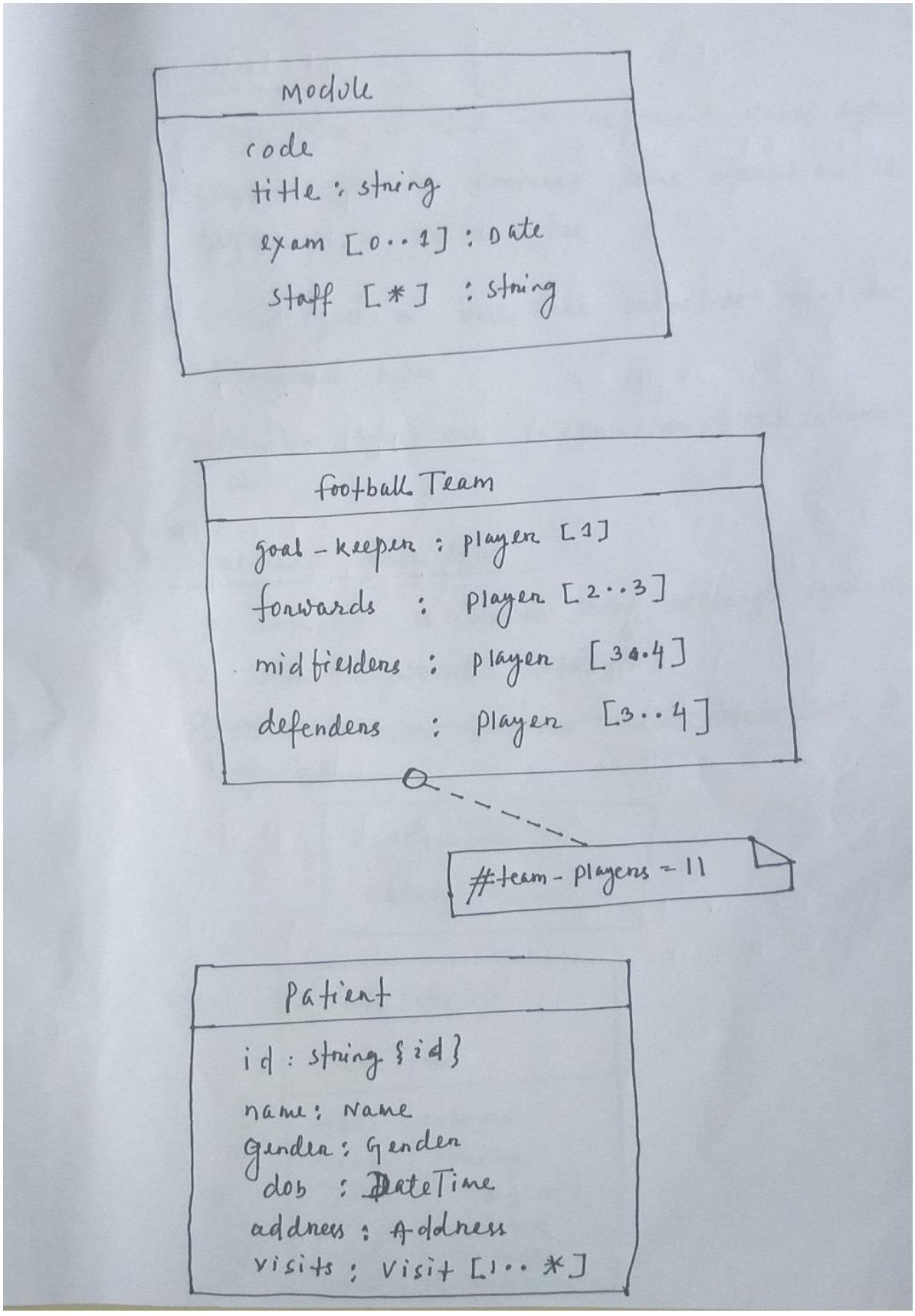
mcount : integer i.e. the underlined attribute integer has the class scope. That means there is a single copy of the attribute which is accessible to all the objects of the class.

e.g. static variable in C++./java.

Multiplicity of attribute

- Multiplicity is the cardinality i.e. the no. of elements
- Here, attribute multiplicity means, no. of admissible value
- The default multiplicity of an attribute is "exactly one".

multiplicity	cardinality
[0..0] (can also be specified by only 0)	collection must be empty
[0..1]	No instance or one instance
[1..1] (also specified by 1)	Exactly one instance
[0..*] (also specified by *)	zero or more instances
[1..*]	atleast one instance (or one or more instances)
[5..5] (also specified by 5)	Exactly 5 instances
[m..n]	Atleast m but not more than n instances


Module
code
title : string
exam [0..1] : Date
staff [*] : string
football Team
goal-keeper : player [1]
forwards : player [2..3]
midfielders : player [3..4]
defenders : player [3..4]
#team-players = 11
Patient
id : string {id}
name : Name
gender : Gender
dob : DateTime
address : Address
visits : Visit [1..*]

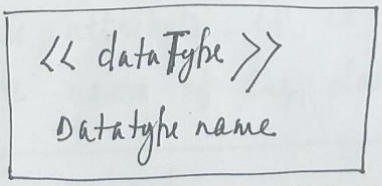
Data types

- Data types is used to represent value types
- Data type also contains some operations to support the value types.
- Data types in UML are primitive type or structured types
- Primitive types are integer, Char, float, double, String etc.

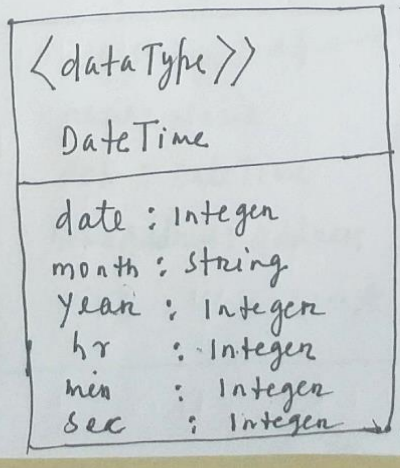
Structured data type

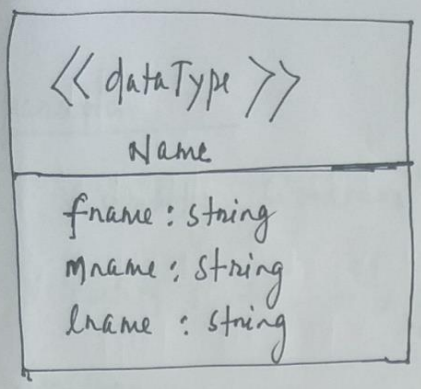
- Data type is shown using rectangle symbol with keyword datatype

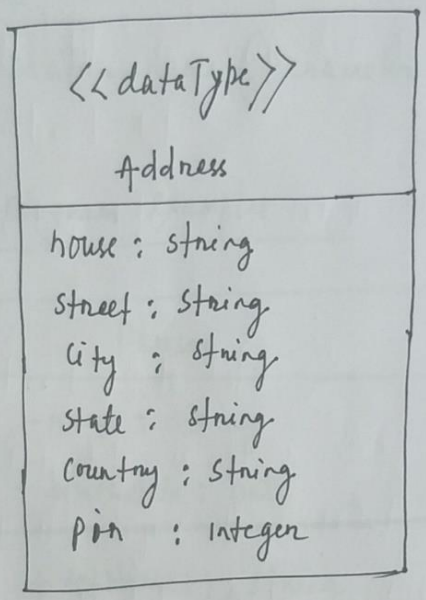
~~Data type contains attributes and operations~~



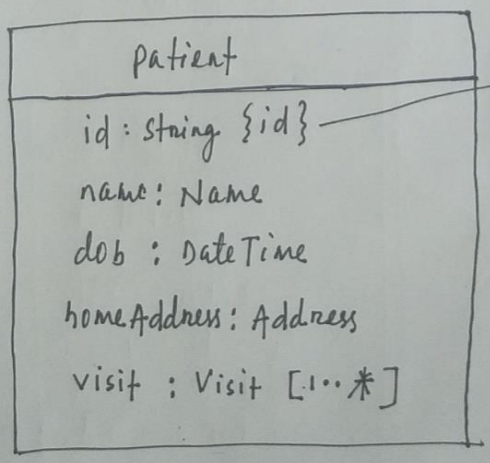
e.g.







Note: When a data type is referenced by a class attribute, it is shown simply as the name of the data type



{ } → modifier
i.e. readonly
Ordered
Unique
Non-Unique
Union
Subset
Constraint etc.

Operation

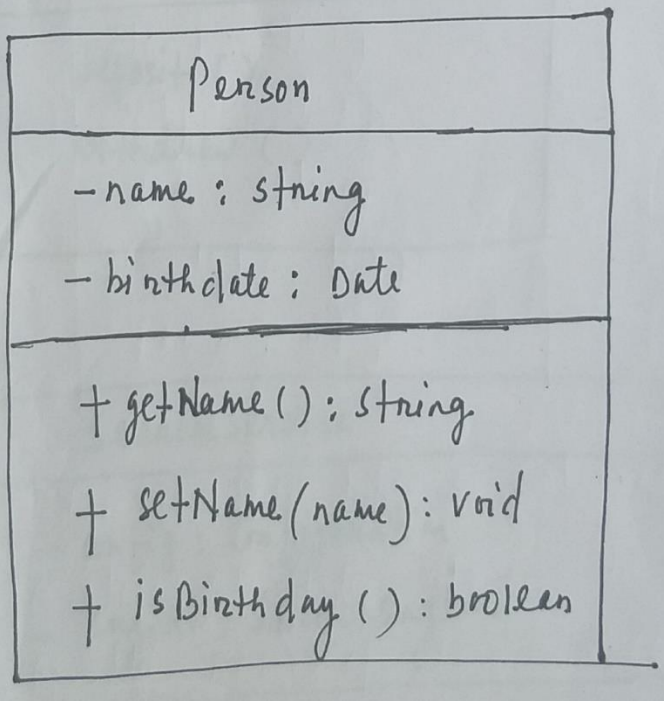
Visibility Signature

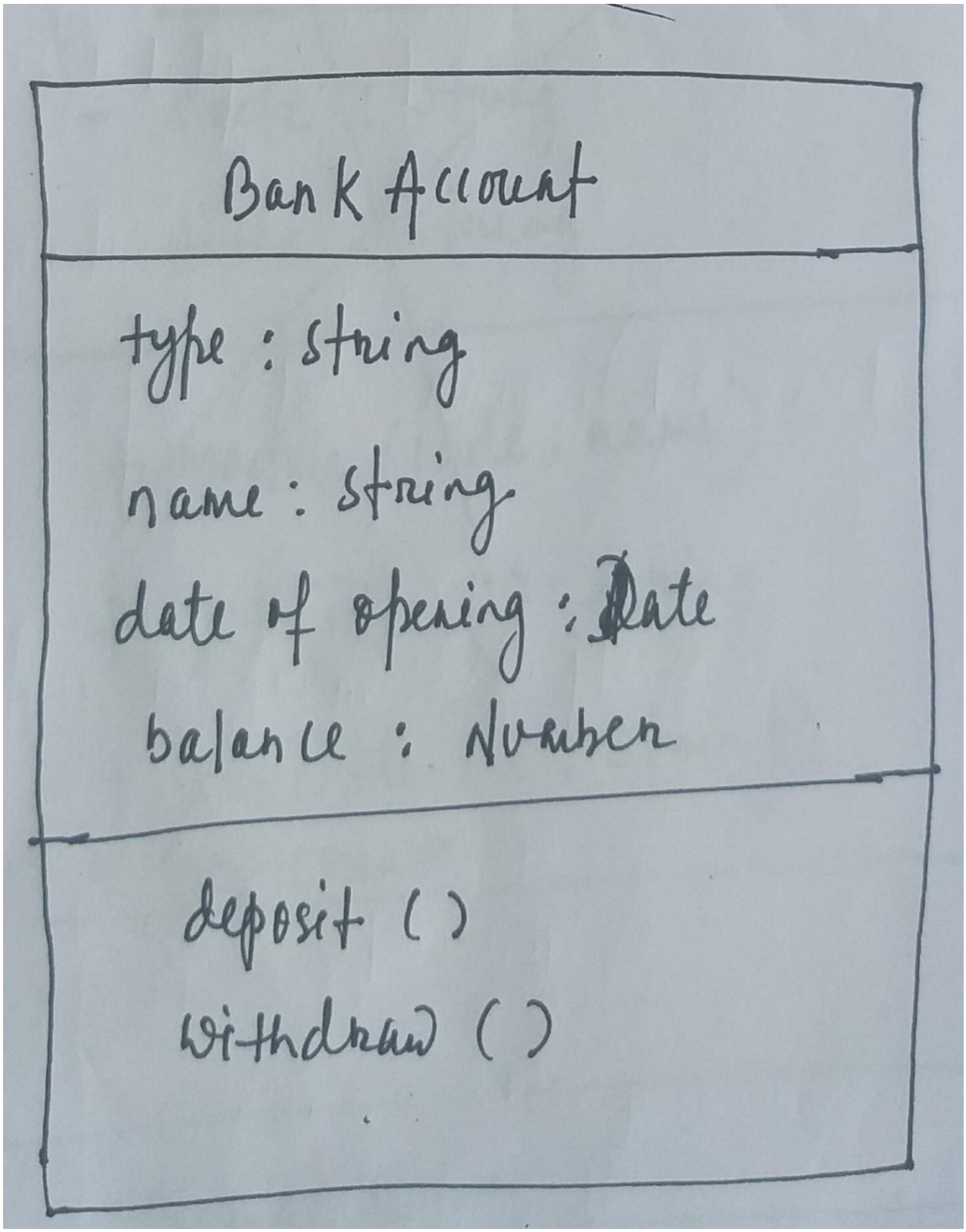
Visibility : +, −, #, ~, / (derived), underline (static)

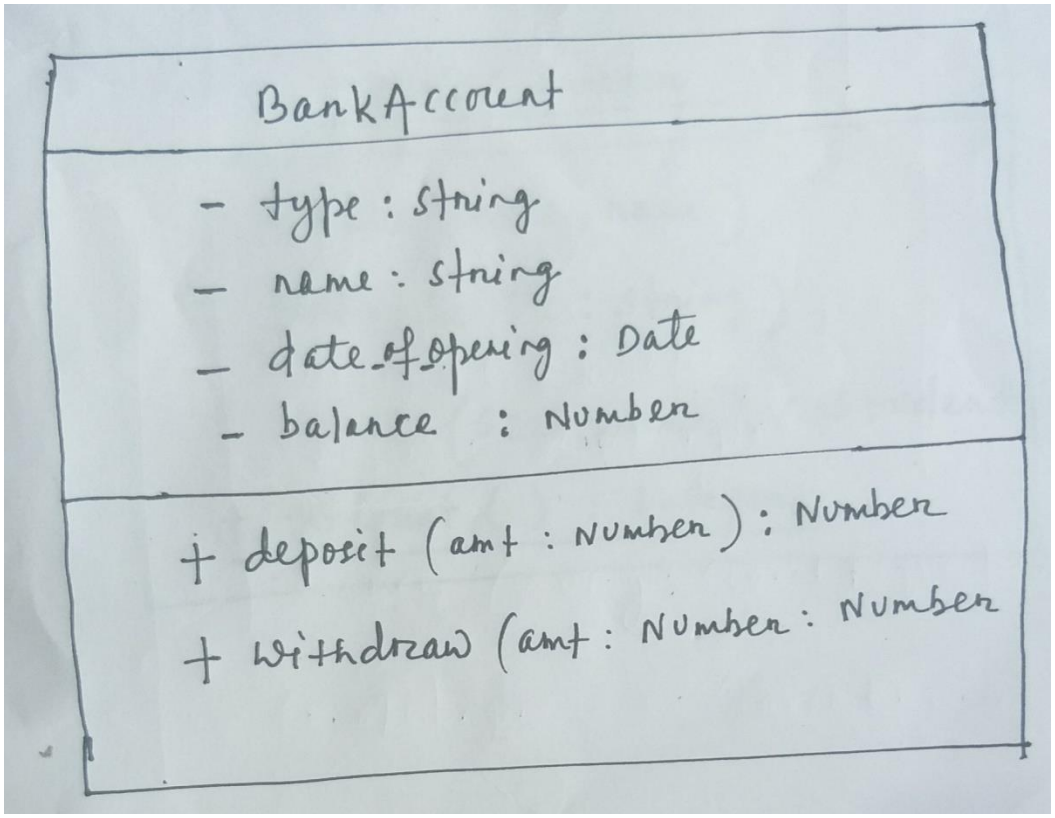
Signature

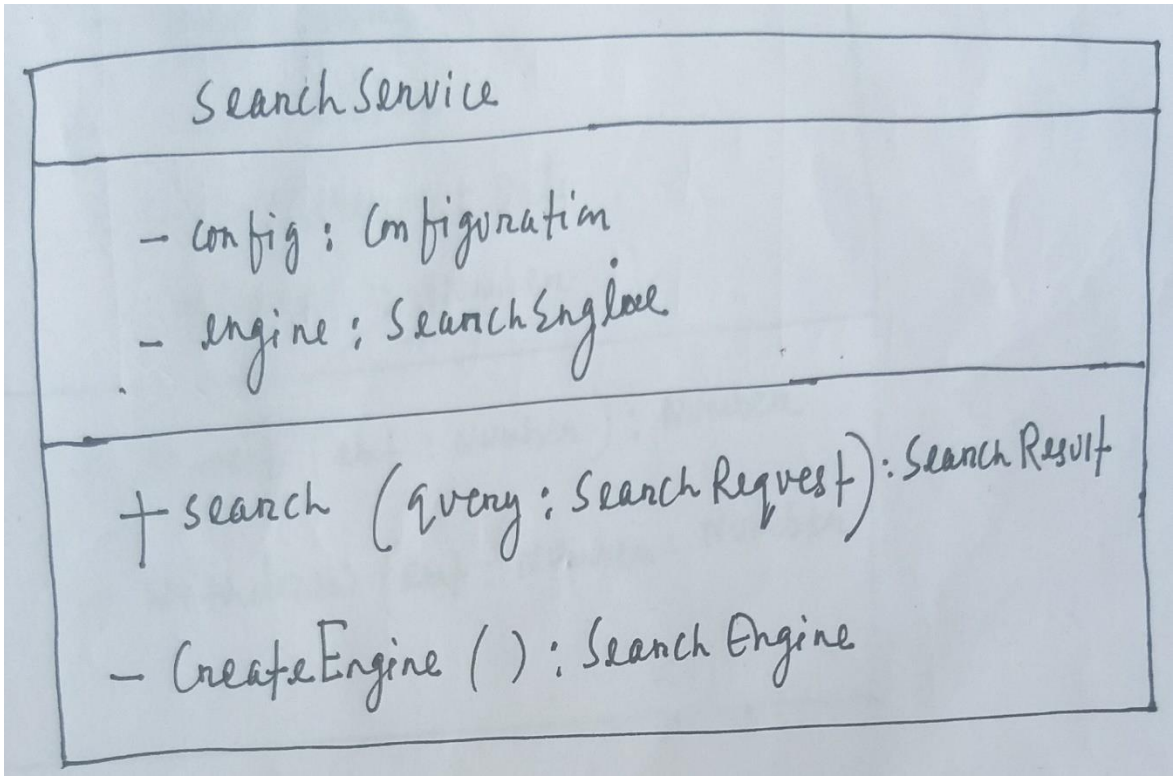
Operation-name (Parameter list) : Return Type

Class Diagram Examples




Bank Account
type : string
name : string
date of opening : Date
balance : Number
deposit ( )
withdraw ( )


BankAccount
- type : string
- name : string
- date_of_opening : Date
- balance : Number
+ deposit (amt : Number) : Number
+ withdraw (amt : Number : Number


SearchService
- config : Configuration
- engine : SearchEngine
+ search (query : SearchRequest) : SearchResult
- CreateEngine ( ) : SearchEngine

Relationships

- Link
- Association
- Generalization and Inheritance
- Aggregation
- Composition

Link

- Link is a physical or conceptual connection among objects
- A link is a relationship among objects
- Most links relates two objects but some links relate three or more objects

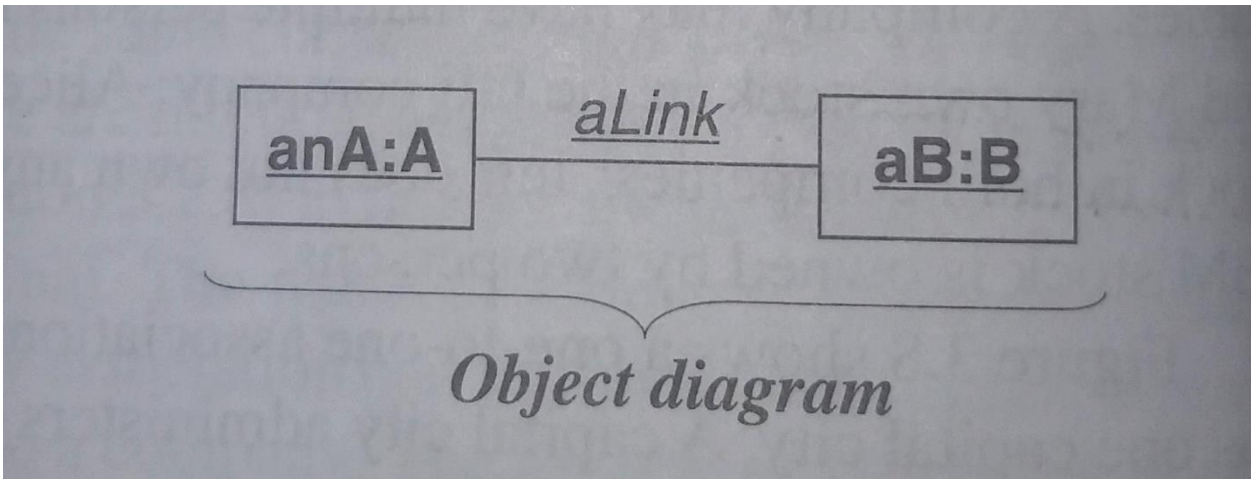
UML Notations

The UML notation for a link is a line between the objects

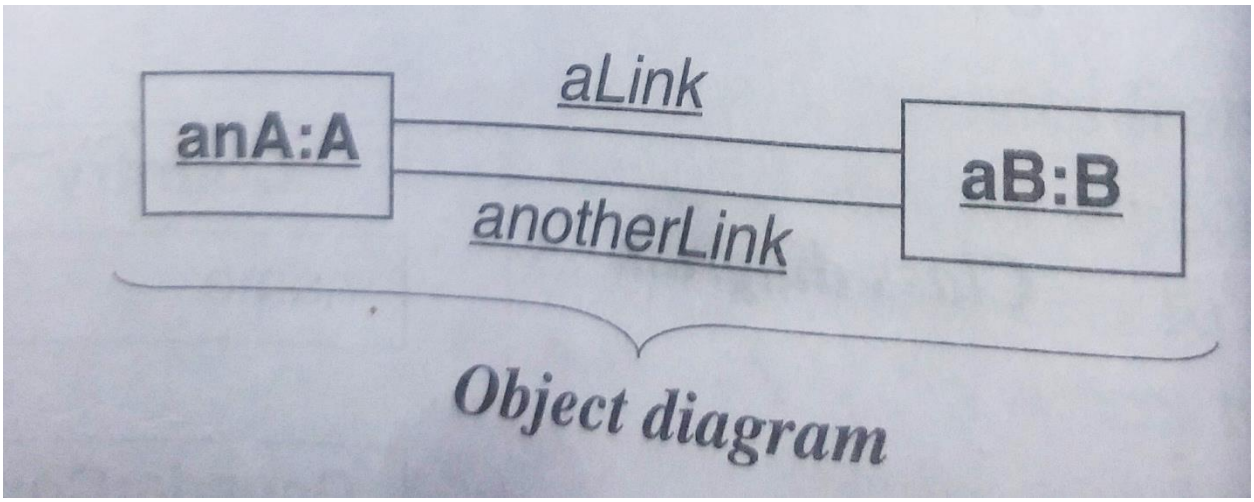
The link name (if specified) must be underlined.

The link name is specified as a verb.

The link name is specified in italics letters.

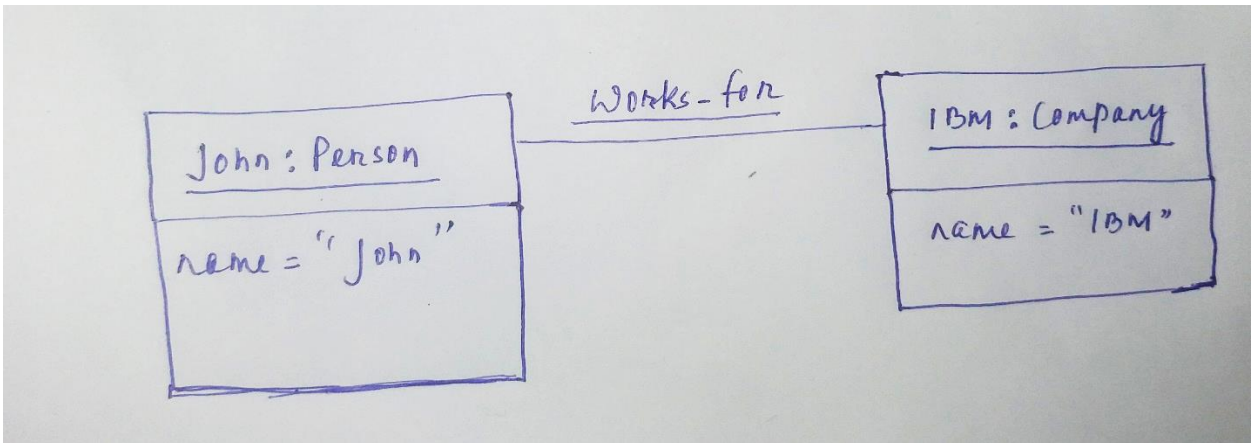


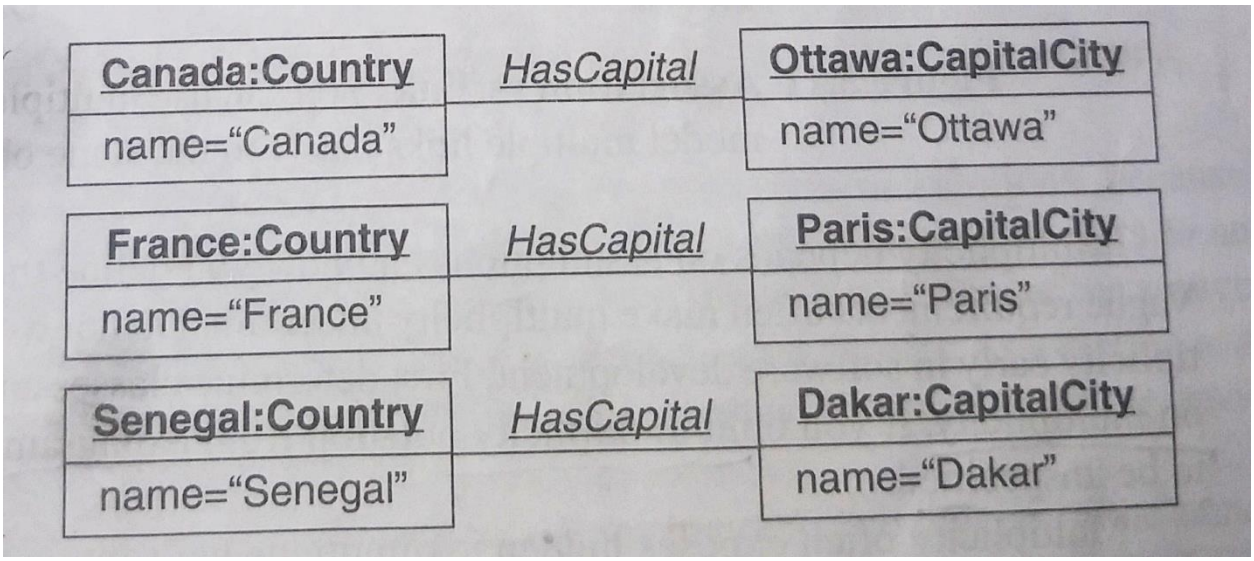
Object diagram



Object diagram

The followings are the example of **Object Diagrams with links**





Association

- Association is the structural relationship among classes
- In OO designs, all connections among classes are modeled as associations.
- Association describes a set of links with common structure and semantics.

for example, person owns-stock in a company
person works-for a company.

Here, owns-stock and works-for are associations which relates a person class to a company class.

UML Notations

- An association connects related classes and denoted by a line
- Association name is a verb and specified in italics
- Association name is optional if a model is unambiguous.

→ When the model is ambigous (in case of multiple association among classes), association-names or association-end-names must be specified

→ By default associations are bidirectional i.e. they can be traversed in both directions with different conditions.

Multiplicity

- Multiplicity specifies the no. of instances of one class that may relate to an instance of an associated class.
- Multiplicity is a constraint on the no. of related objects.

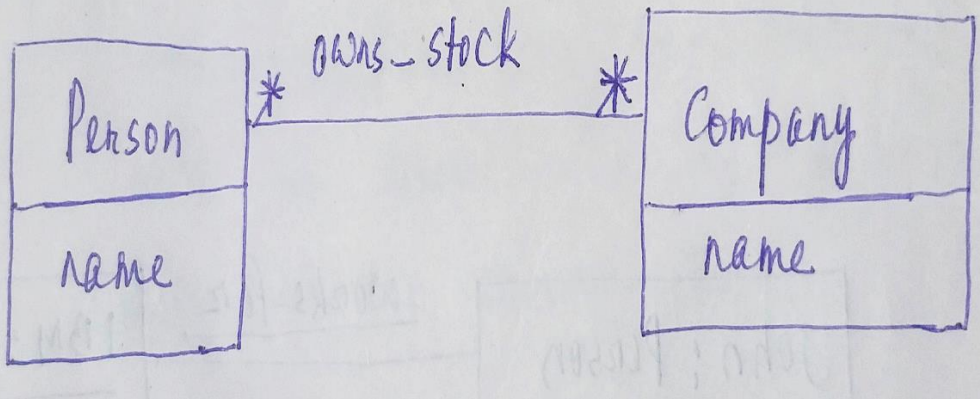
- UML specification for multiplicity

1	exactly one
0..1	zero or one
1..*	one or more
M..N e.g. 3:6	M to N (both are enclusive) 3 to 6 (both 3 and 6 are enclusive)
*	many
0..*	zero or more

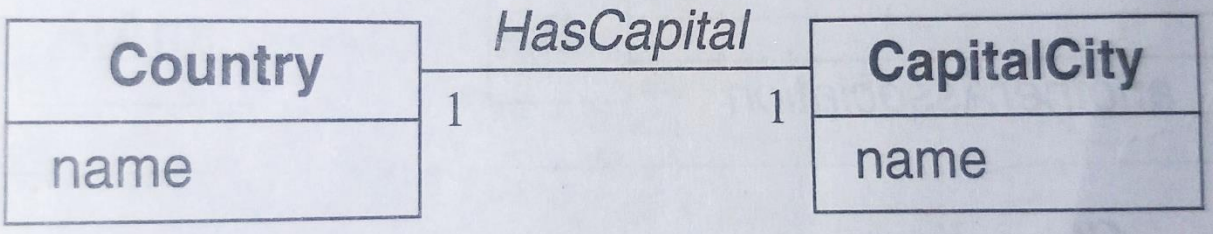
UML Notations.

Multiplicity is specified at the association ends. for example, a one-to-many association has two ends - one end with multiplicity of one and other end with multiplicity of many.

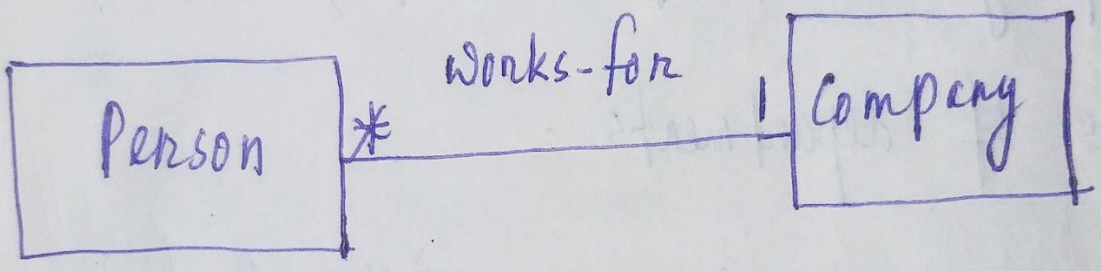
The following figure specifies many-to-many multiplicity. A person may holds stock in many companies. A company may have multiple persons holding its stock.



The following figure shows one-to-one association i.e. Each country has one capital city and a capital city administers one country.



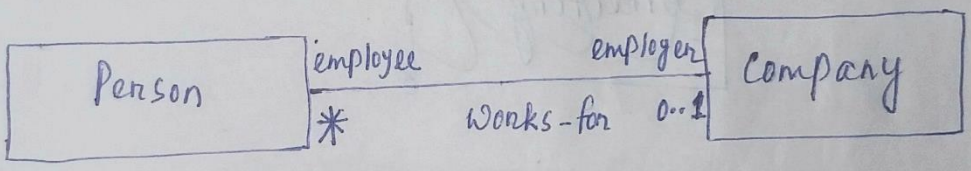
The following figure specifies many-to-one multiplicity as many employees works for a company



Association End Names

Association end name is a name (noun) appears next to the association end.

for example, in the following figure, person and Company participate in association Works-for.
A person is an employee with respect to a Company.
A company is an employer with respect to a person.

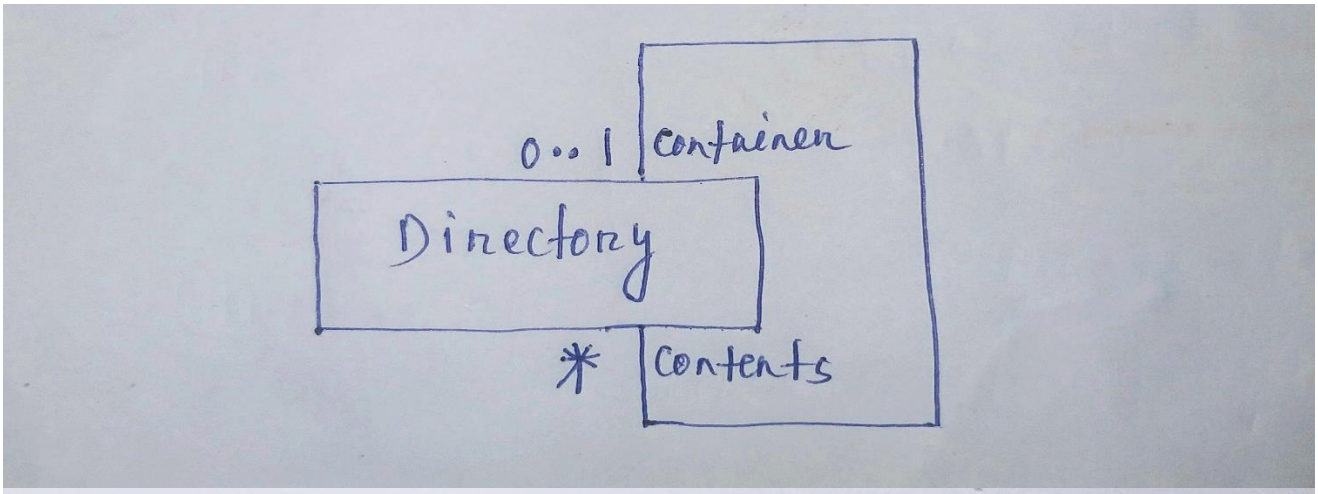


- Association end names is optional in binary associations
- Association end names especially convinient for traversing associations.
- Association end names are mandatory for recursive or reflexive associations or self association

Self Association
or Recursive association
or Reflexive association

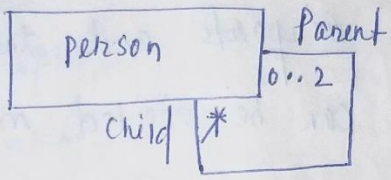
These are the association between two objects of the same class.

Some Examples of Self Association are as follows



In the above figure each directory has exactly one user who is an owner and many users who are authorized to use the directory.

for example, a person class, one person instance participates in two or more link, twice as a parent, zero or more times as a child



Association class

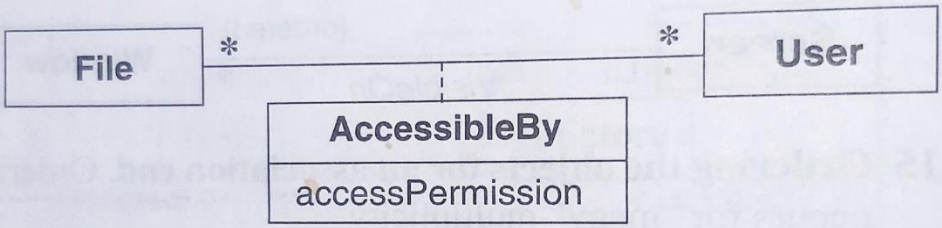
- When an association is described by attributes, it is known as association class
- An association class is an association that is also a class.
- Association class can have attributes and operations of their own.
- Association class can also participate in other associations with different classes.
- Instance of an association class derives identity from the instances of constituent classes.

UML Notations

- The name of an association class is generally an adverb.
- The notation of association class is a rectangular box (with 3-compartments like a class).
- The association class attached to the association by a dashed line.

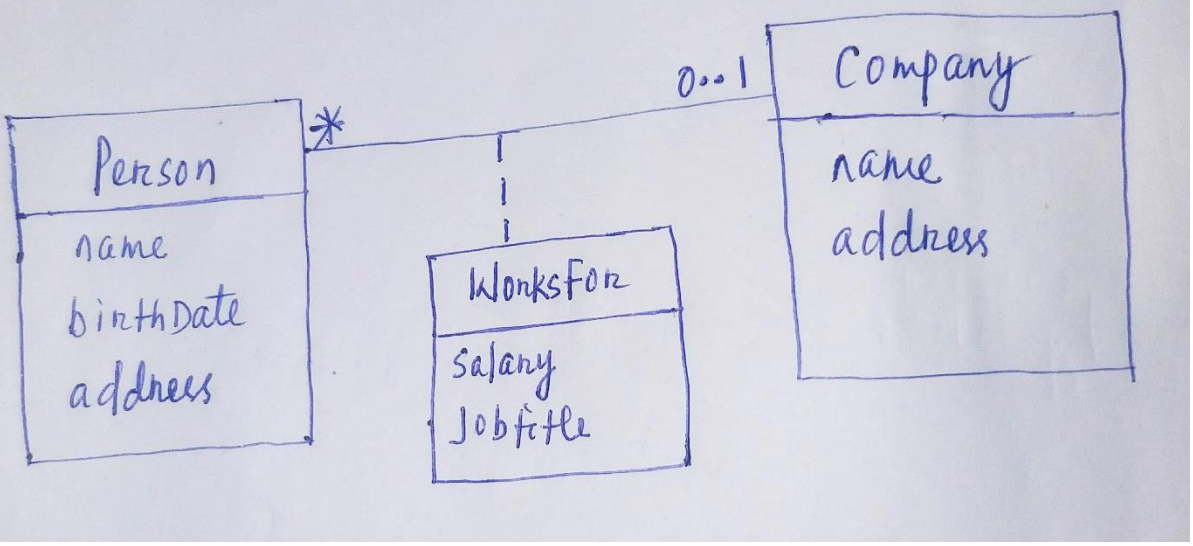
In the following diagram, AccessibleBy is an association between file class and user class which is also an association class.

The attribute accessPermission of the association class AccessibleBy is a joint property of File and User and can't be attached to either File and User alone.



(d)

In the following diagram, works-for Association is represented as an association class with attributes salary and JobTitle.



The following diagram shows an association class also participating in an association. For example, users may be authorized on many workstations. Each authorization carries priority and access privileges.

A user has a home directory for each authorized workstation, but several workstations and users can share the same home directory.

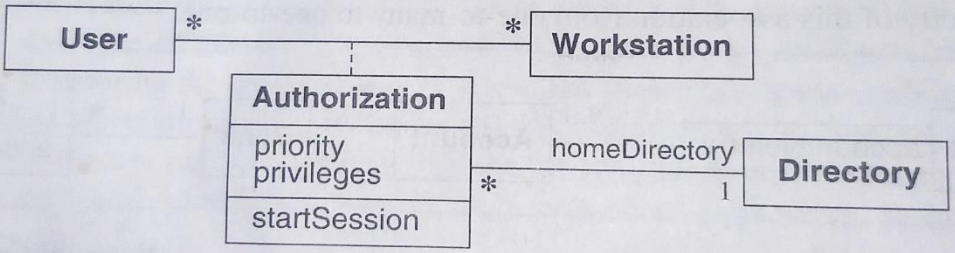


Figure 3.20 An association class participating in an association. Association classes let you specify identity and navigation paths precisely.

The following figure shows the difference between association class and ordinary class.

The association class has only one occurance for each pairing of Person and Company.

But, there can be any number of a Purchase for each Person and Company. Each purchase instance is distinct here.

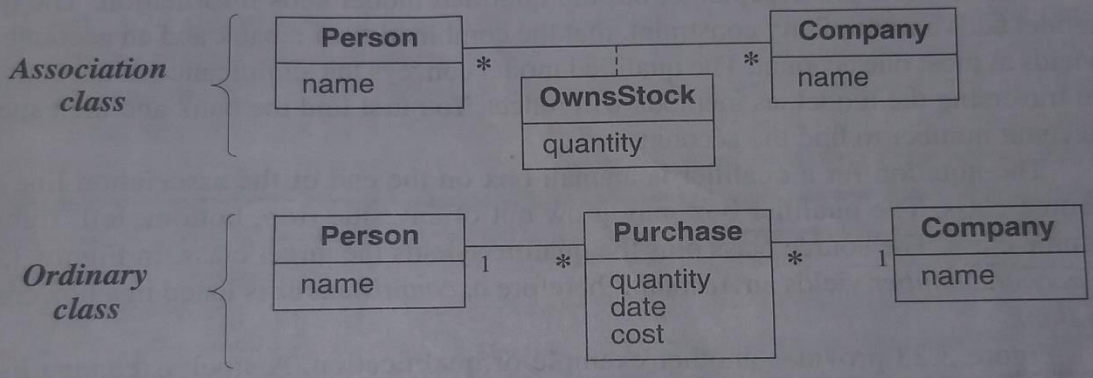


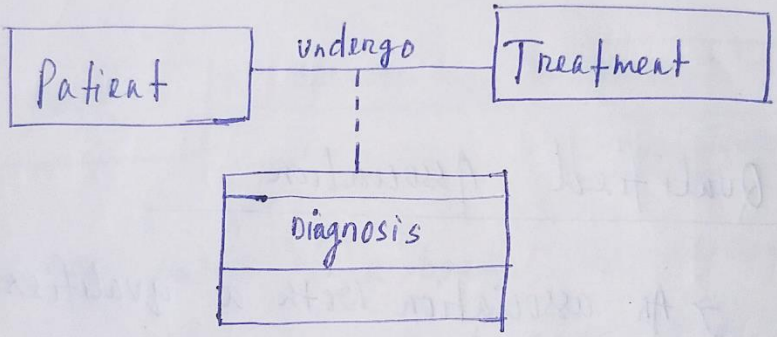
Figure 3.21 Association class vs. ordinary class. An association class is much different than an ordinary class.

Attribute association

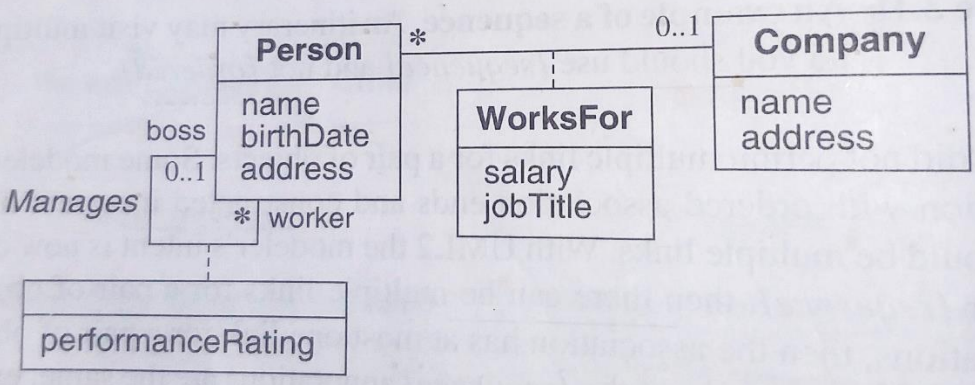
→ Attribute association is an association that contains attributes but does not directly participate in relationships with other classes.

→ In this case the class does not carry a specific name.

e.g.



Here, patient undergo treatment based on the diagnosis made. Diagnosis is known as attribute Association.



Qualified Association

→ An association with a qualifier is known as qualified association

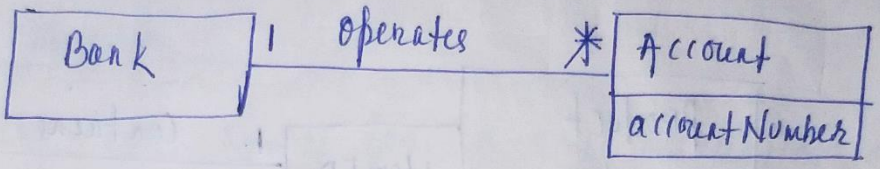
→ A qualifier distinguishes the set of objects at the many end of the association based on the qualifier value.

i.e. A qualifier is used to select an object from a large set of related objects based on the qualifier value.

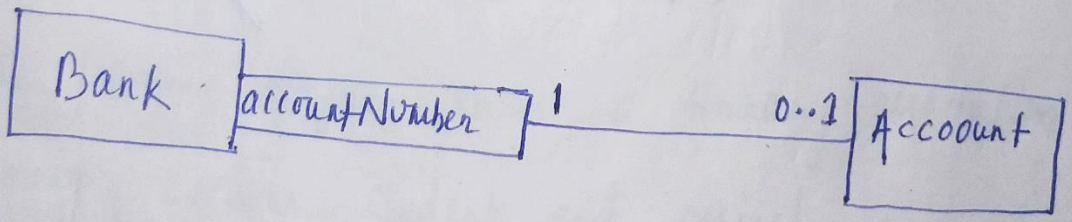
→ A qualified association reduces the effective multiplicity from "many" to "one" at the target end.

→ It is possible to define qualifiers for association with multiplicity one-to-many or many-to-many.

e.g. the following diagram shows a bank operates multiple accounts. Here, multiplicity is one to many

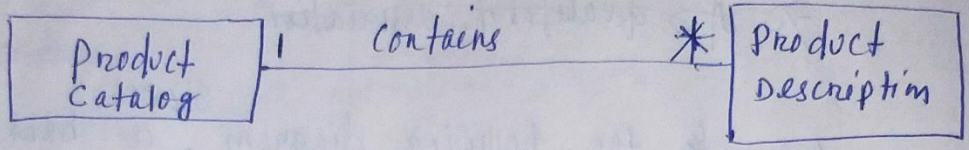


Within the context of a bank, the account number specifies a unique account. If account number is chosen as qualifier, it reduces the effective multiplicity from one-to-many to one-to-one

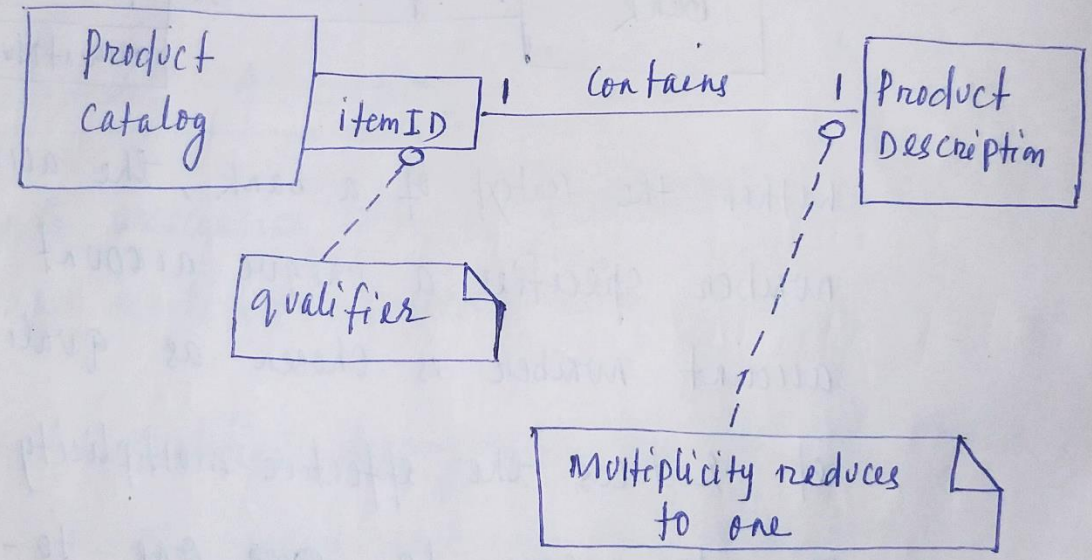


eg. of

another example of qualified association is as follows



(Not qualified)



(qualified association)

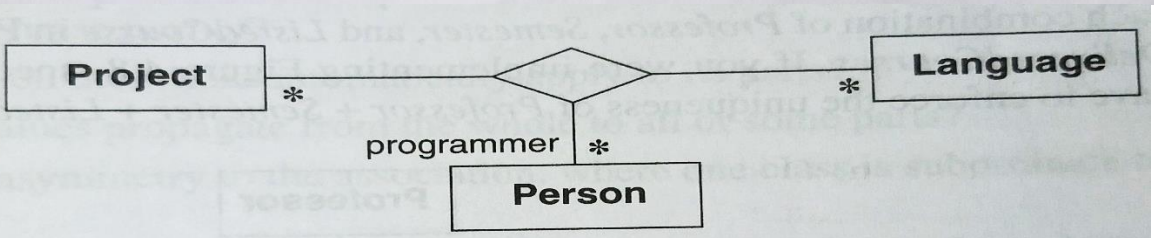
N-ary Association

- Associations among three or more classes is known as N-ary association.
- As N-Ary associations are complex to program, most N-ary associations can be decomposed into binary associations.

UML Notation

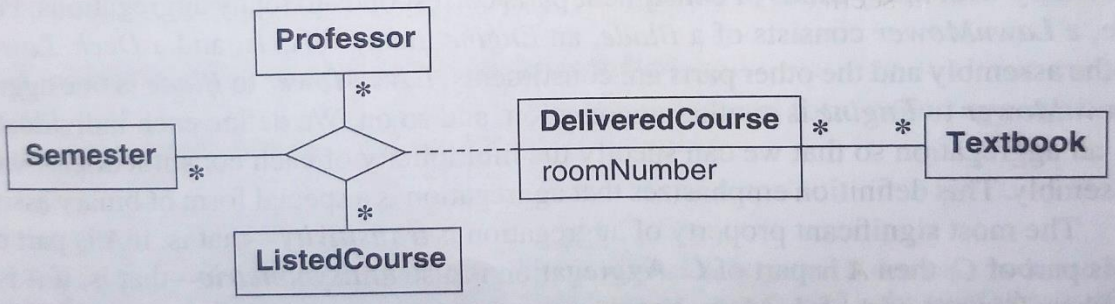
The UML symbol for n-ary association is a dimond with line connecting to related classes.

The following diagram shows a n-ary (ternary) association. A programmer use computer languages to work on projects.



Consider the following diagram representing ternary association.

A professor teaches a listed course during a Semester. The resulting delivered course is an association class has a roomNumber and any number of text books.



The typical programming language can't express N-ary association. So, N-ary association can be promoted into classes (of binary associations).

The following diagram shows the conversion of ~~the~~ above ternary association into classes with possible binary associations.

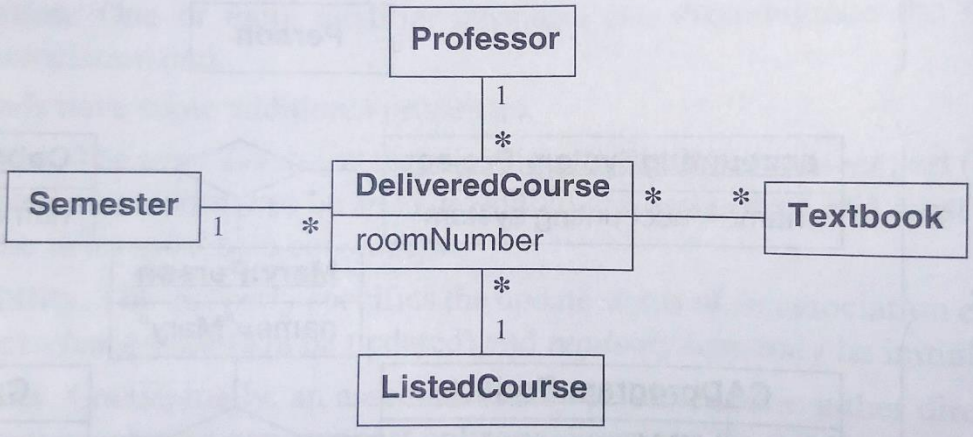


Figure 4.8 Promoting an n-ary association. Programming languages cannot express n-ary associations, so you must promote them to classes.

Ordering ,Bags and Sequences:

Ordering

→ Ordering specifies, ordered collection of objects in an association.

→ Ordering is the ordered collection of objects with no duplicates (i.e. duplicates are not allowed). It can be considered as an ordered set.

e.g. A workstation screen containing a no. 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 any point on the screen.

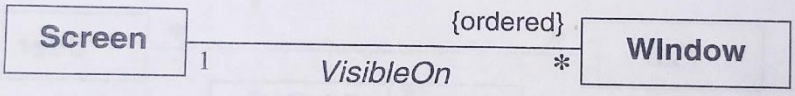


Figure 3.15 Ordering the objects for an association end. Ordering sometimes occurs for "many" multiplicity.

Bag: A Bag is a collection of elements with duplicate allowed

Sequence: A sequence is an ordered collection of elements (i.e. objects) with duplicate allowed.

for example, An itinerary is a sequence of airports and the same airport can be visited more than once.

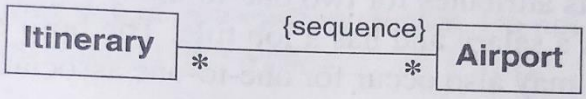


Figure 3.16 An example of a sequence. An itinerary may visit multiple airports, so you should use *{sequence}* and not *{ordered}*.

Note:

→ {ordered}, {bag} and {sequence} are permitted only for binary associations

→ A sequence association is an ordered bag

→ {ordered} and {sequence} are same except {ordered} does not allows duplicate where as {sequence} allows.

Enumerations

→ An enumeration is a datatype that has a finite set of values.

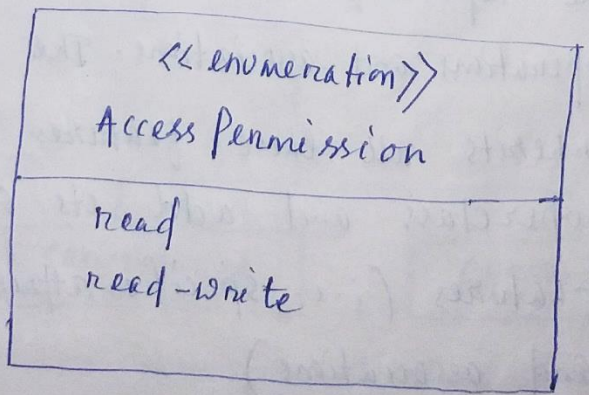
→ Enumeration specifies possible values and we must restrict data to these legitmate values.

UML Representation

→ In UML, an enumeration is a data type

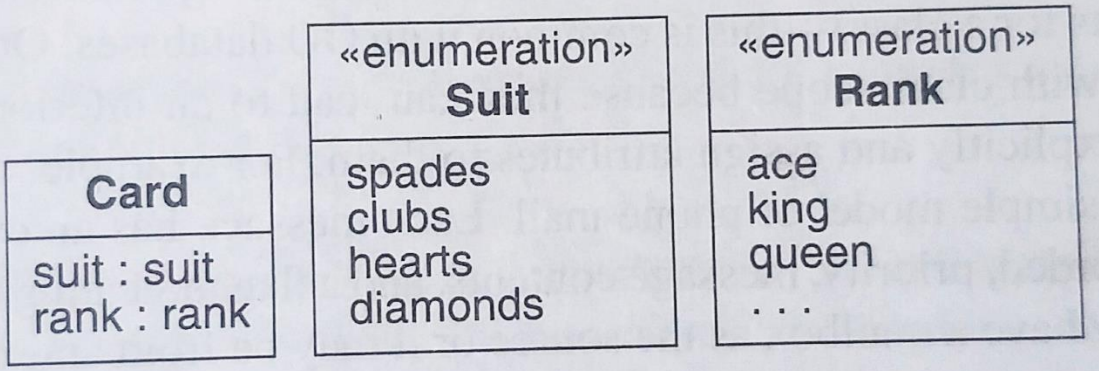
→ Enumeration can be specified by listing the keyword enumeration in guillemets (<< >>) above the enumeration name in the top section of a box. The second section in the box specifies enumeration values.

for example file access permission is an enumeration with possible values read and read-write



for example, the playing card suit is an enumeration with values spades, clubs, hearts and diamonds. No other values can be taken for suits except these values.

Similarly, the rank of cards are in the order ace, king, queen and so on which is also an enumeration.



Generalization, specialization and Inheritance

Generalization:

→ Generalization is the ~~relation between~~ relationship between the superclass and the subclasses

→ The superclass holds common attributes, operations and associations. The subclass inherits all these features from the superclass and add its specific features (i.e. specific attributes, operations and associations)

→ Generalization is also called as "is-a" relationship

(Generalization is known as "is-a" relationship because each instance of a subclass is also an instance of the super class)

→ A super class is also known as Parent class or Ancestor class or Base class

A subclass is also called as child class or descendant class or derived class

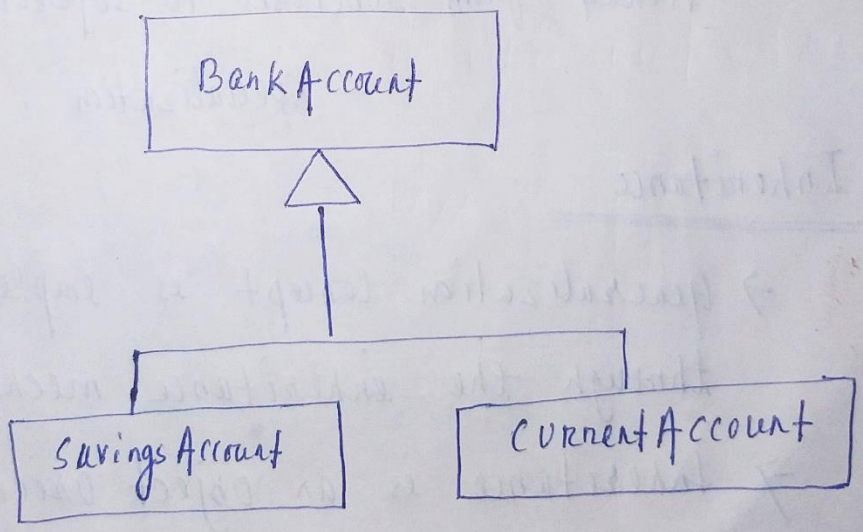
→ An instance (object) of a subclass is symultaneously an instance (object) of all its ancestor classes.

An instance includes a value for every attribute of every ancestor class. An instance can invoke (call) any operation on any ancestor class.

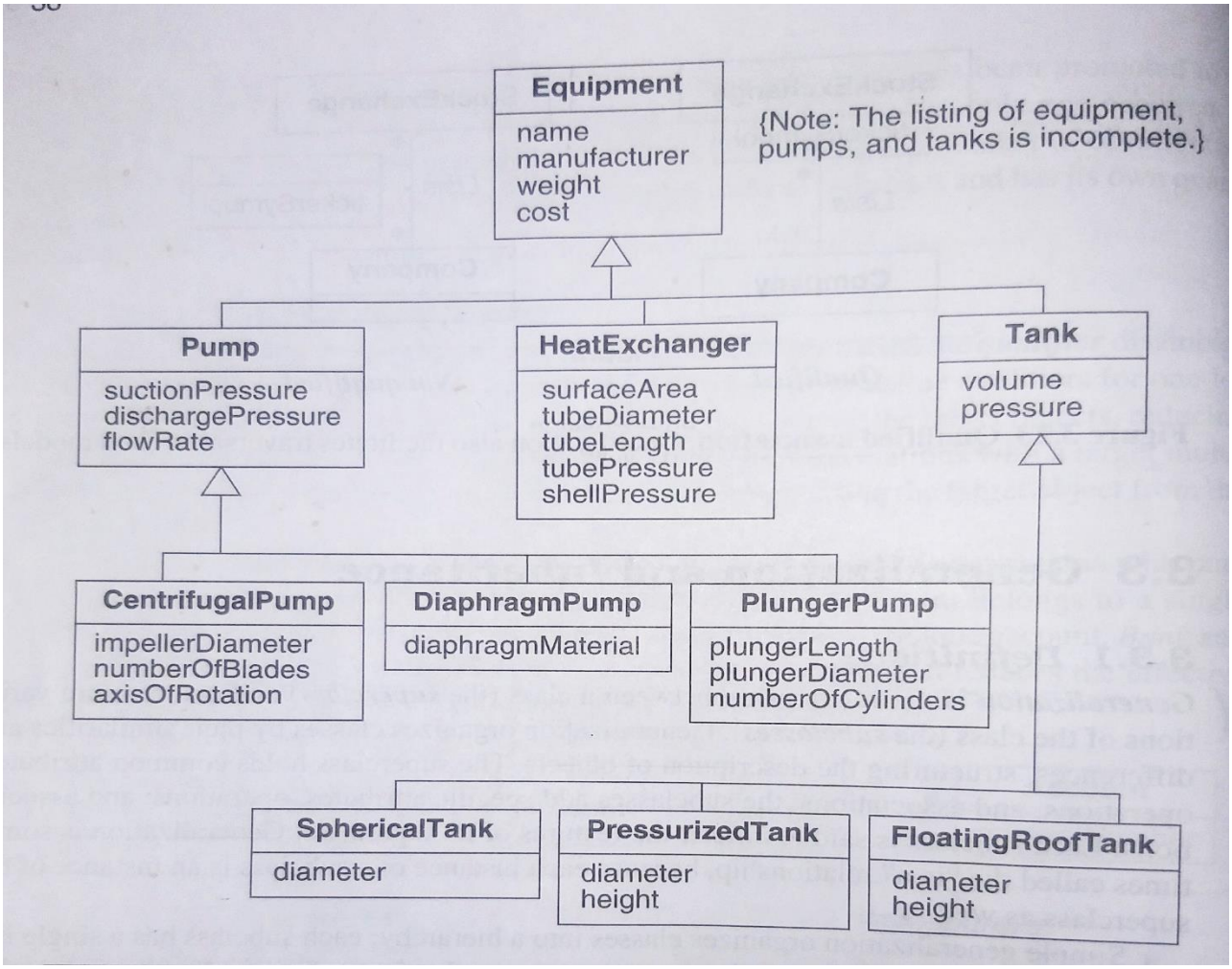
Each subclass inherits all the features of its ancestors and add its own specific features.

UML Notation

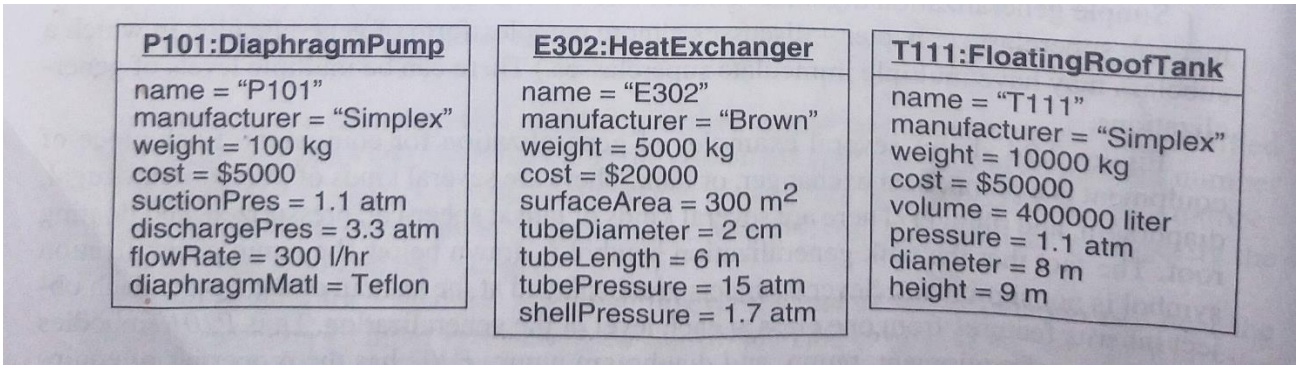
A large hollow arrow head (△) denotes generalization. The arrow head points to the superclass.



Example of generalization:



Object diagrams of DiaphragmPump, HeatExchanger, FloatingRoofTank are as follows which contain features of ancestor classes plus its own features



Example

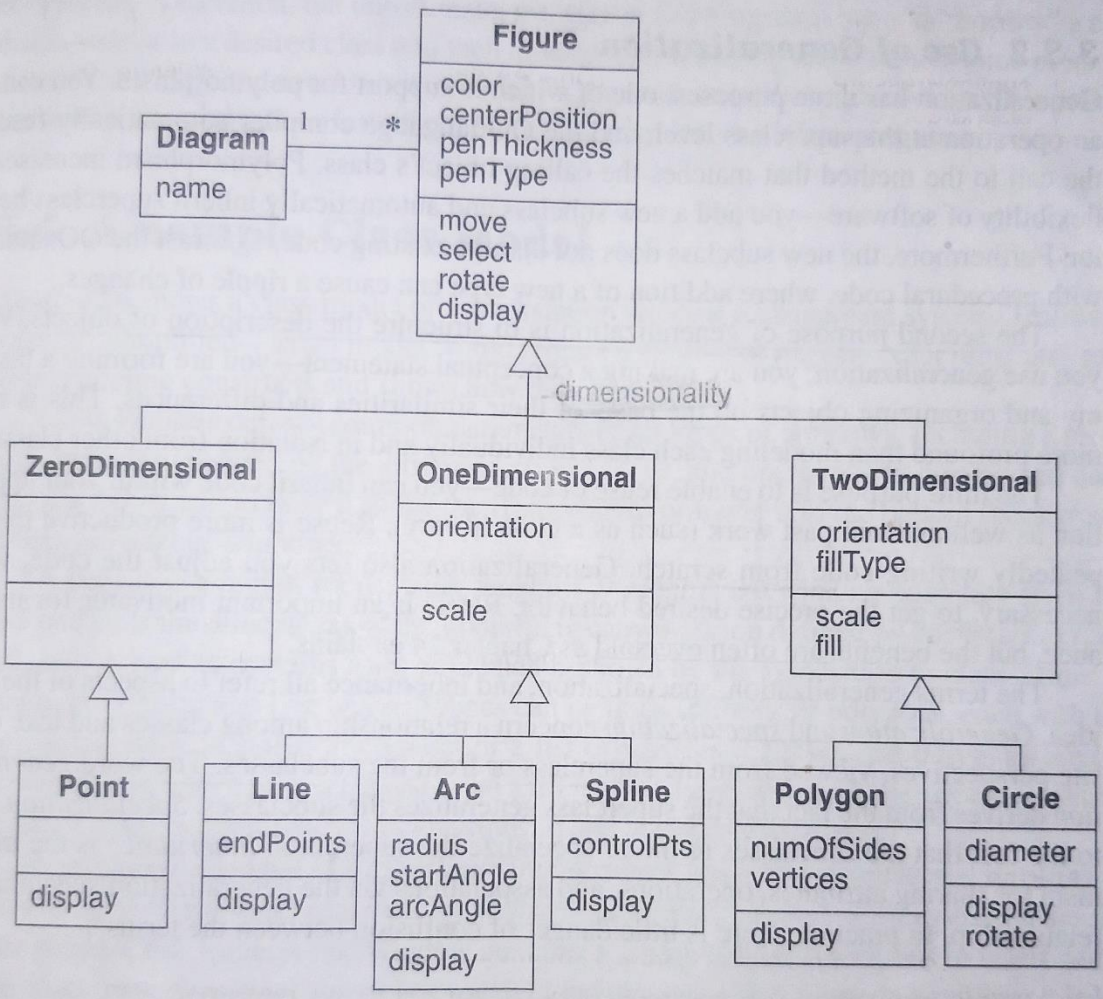


Figure 3.25 Inheritance for graphic figures. Each subclass inherits the attributes, operations, and associations of its superclasses.

Generalization set

→ A generalization set is a packagable element whose instances define collection of subsets of generalization relationships

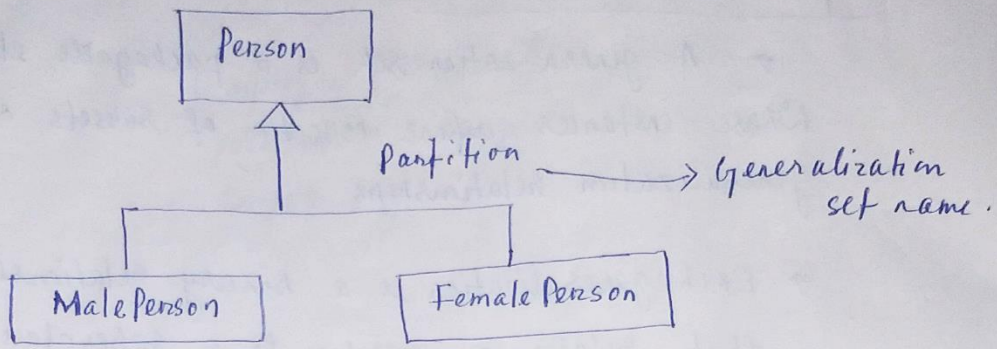
→ Each generalization is a binary relationship that relates a subclass to a superclass.

Each generalization set defines a particular set of generalization relationships that defines the way in which a superclass is devided into specific subclasses.

for example, a generalization set could define a partioning of the class person into two subclasses male person and female person

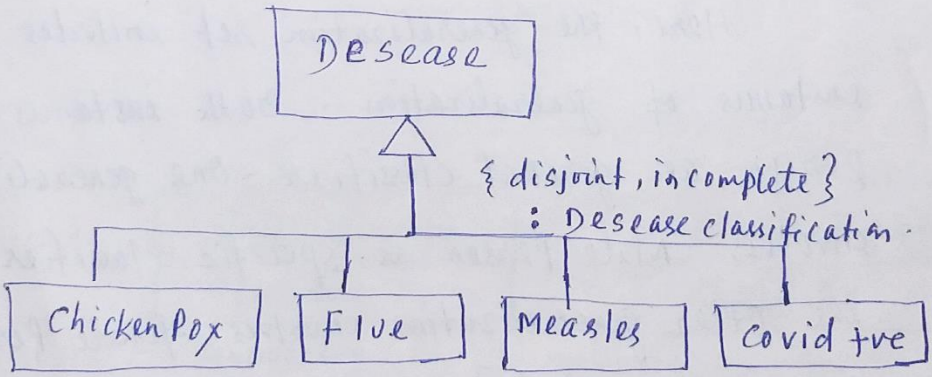
Here, the generalization set includes two instances of generalization. Both instances have person as general classifier. One generalization involves Male Person as specific classifier and the other generalization involves female Person as specific classifier.

→ Generalization set names are optional in OML notations.



Each generalization set may also be associated with a classifier known as power type

Power type specification is shown as colon followed by the name of the powertype classifier near the corresponding generalization set.



Here, {disjoint, incomplete} is Generalization set constraints and : Desease Classification is the power type classifier for generalization set.

Constraints on generalization set

The UML defines the following four types of constraints for generalization sets.

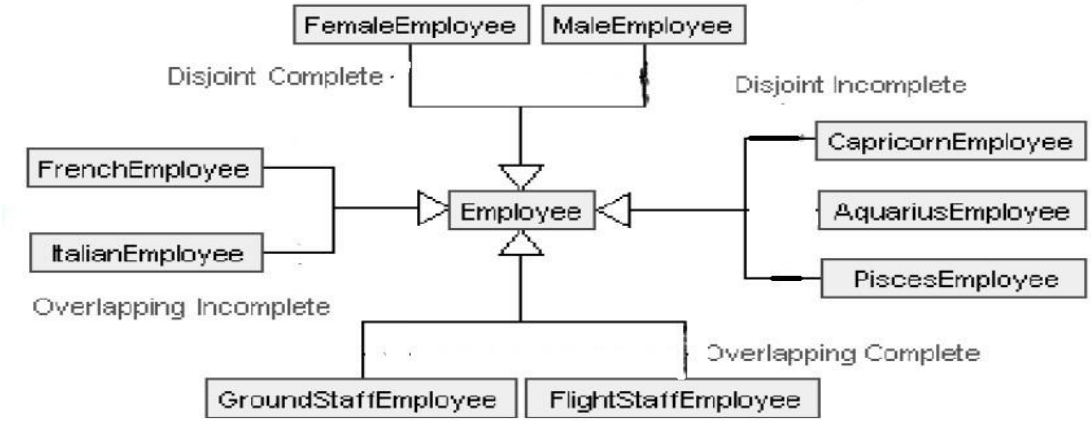
→ disjoint : the subclasses are mutually exclusive. Each object belongs to exactly one of the subclasses.

→ Overlapping : The subclasses can share some objects. An object may belong to more than one subclass.

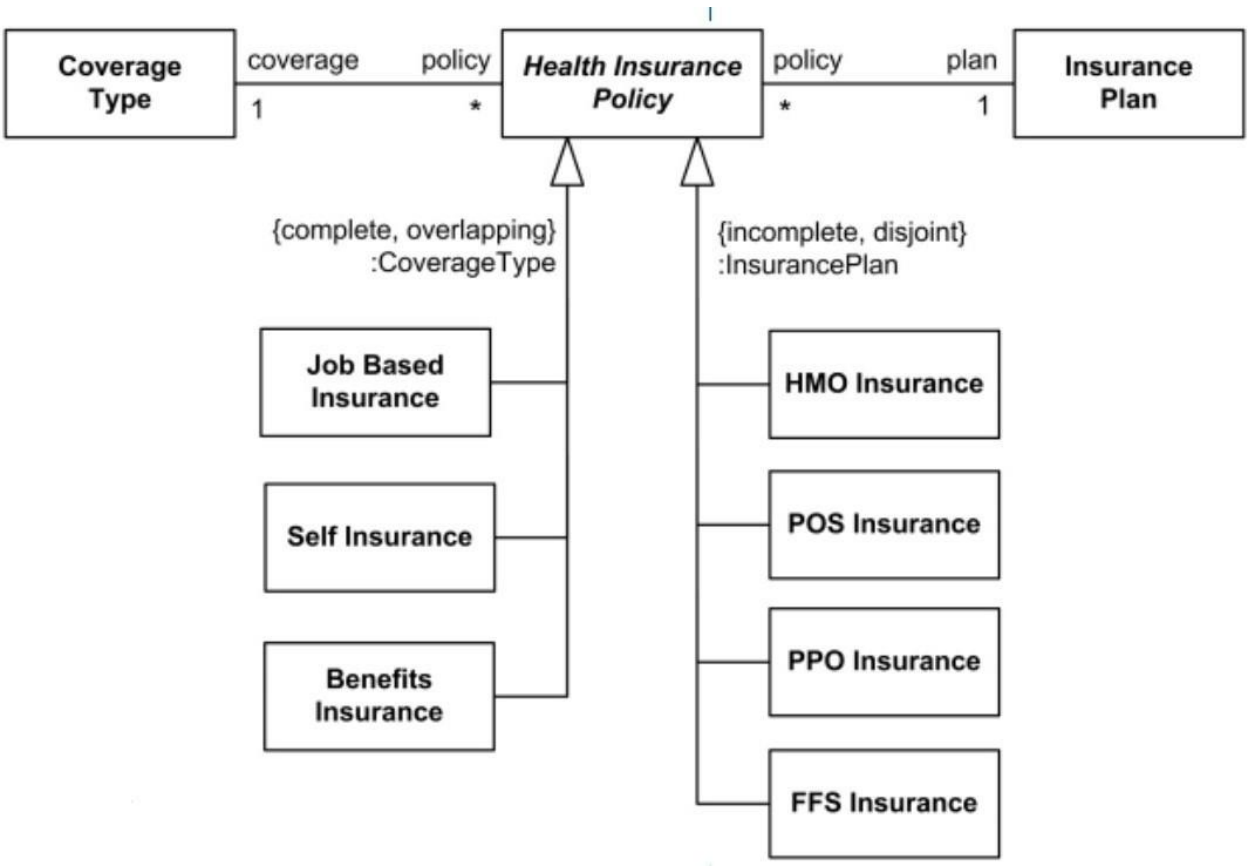
→ Complete : The generalization lists all the possible subclasses. (i.e. no more subclasses can be inherited from a particular base class)

→ Incomplete : The generalization may be missing some subclasses. (i.e. further classification of a superclass can be possible)

Examples of Generalization with generalization set constraints



Example:



Specialization

→ It is the same idea as generalization

→ The word generalization means a superclass generalizes the subclasses

e.g. Bank Account class generalizes Savings Accounts class and Current Accounts class.

→ specialization is the fact that the subclasses refine or specialize the superclass.

e.g. SavingsAccount class and CurrentAccount class specializes the BankAccount class.

→ Viewed from superclass to subclass
= Generalization

Viewed from subclasses to superclass
= specialization.

Inheritance

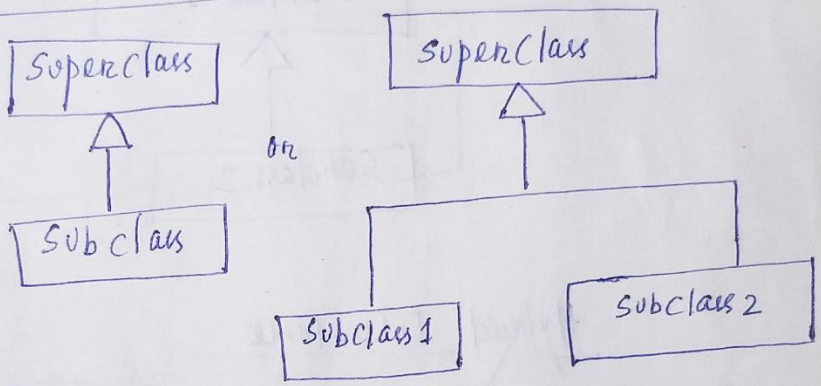
→ Generalization concept is implemented through the inheritance mechanism.

→ Inheritance is an object oriented programming constructs where subclasses acquire the features of superclass and add its own features.

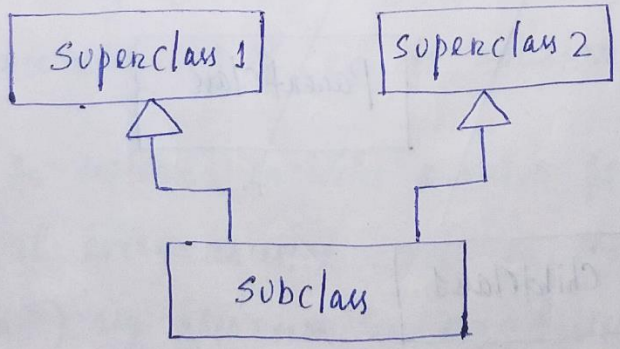
or, a super class shares its features among subclasses and subclasses includes their own specific features in addition to generalized super class features.

Types of inheritance

i) Single inheritance



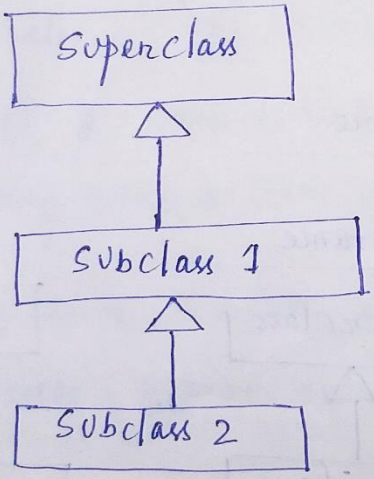
ii) Multiple inheritance



(more than one superclass)

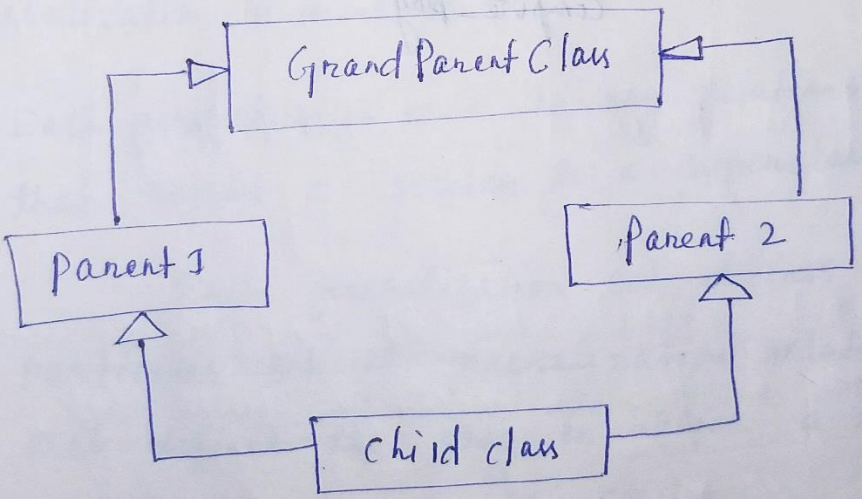
Multilevel Inheritance

A subclass is derived from another subclass



Hybrid Inheritance

Combinations of more than one form of inheritance.



Benefits / Advantages of Generalization

→ Code reusability

Code reusability facilitates to reuse the features (attributes and operations) of an existing class in a newly created class.

Or, in other words, once a class has been designed, it can be adopted by other applications to suit their requirements. This is basically done by creating new classes by reusing the properties of existing classes through the generalization principle.

→ Polymorphism

Generalization support polymorphism.
An operation with same name is present in both the superclass and subclass level. The mechanism of polymorphism automatically resolves the call to the method that matches the calling objects class.

So, it is a process in which a function call to the overridden method is resolved at run time. (e.g. Dynamic Method Dispatch in Java)

Overriding features

→ A subclass may overrides superclass methods by defining its own specific features with the same name.

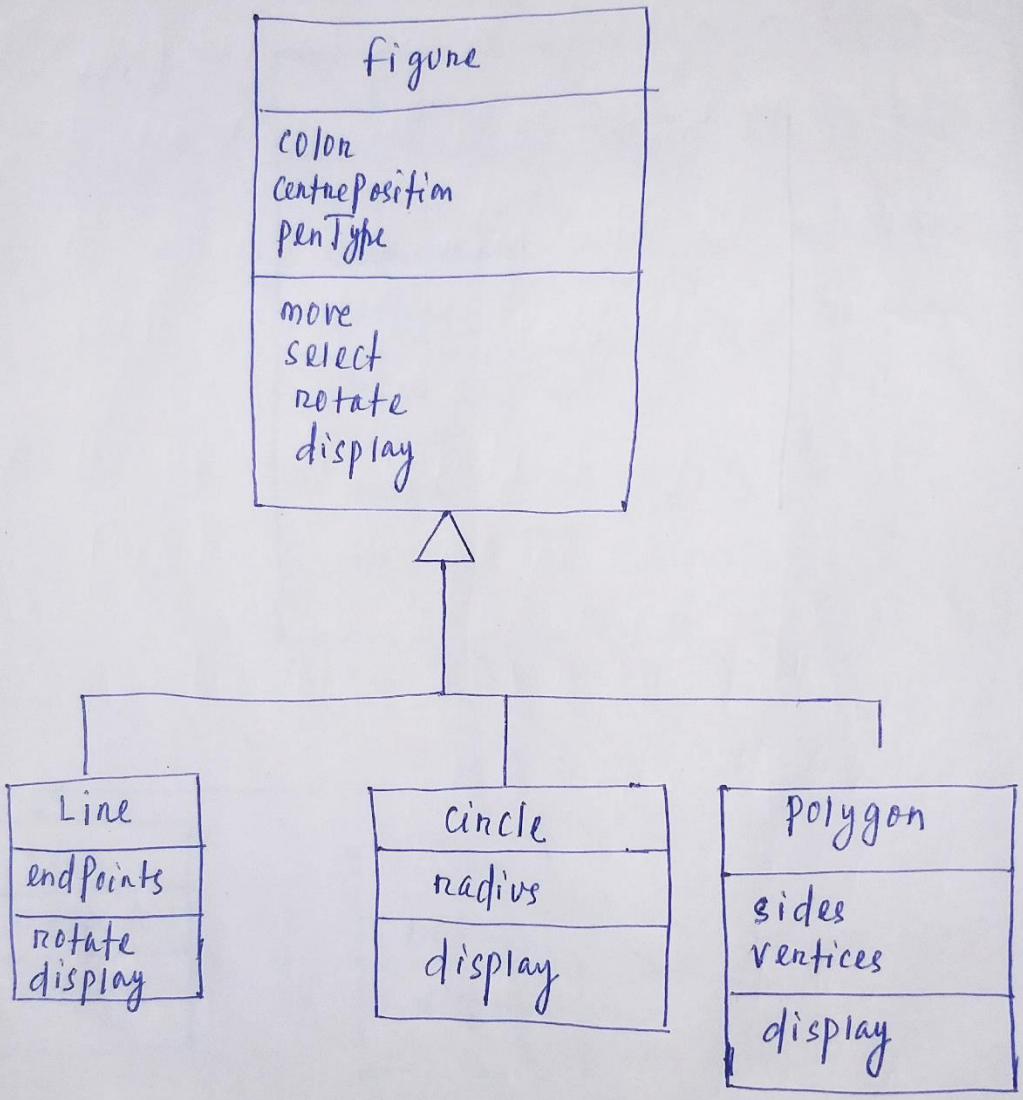
→ Here subclass and super class have a method with the same name and

Overriding features

→ Here, the superclass and subclasses have a method with same name and same signature. The subclass overrides the super class method by defining its own specific implementation

→ In overriding, a subclass replaces method implementation of super class by its own method implementation.

→ In overriding, method signature (i.e. name of the method, no. and type of arguments, return type) should be same in both the superclass and subclass.



In the above diagram, the display method is overridden by the subclasses Line, Circle and Polygon.

Aggregation

→ Aggregation is a strong form of association with part-whole relationship

→ In aggregation, an aggregate object is made up of constituent parts. constituents are part of the aggregate.

→ Aggregation can be defined as a relation of one constituent part class to an assembly class

→ An assembly with many types of constituent parts corresponds to many aggregation.

→ The significant property of aggregation is Transitivity and Antisymmetric

Transitive: If A is part of B and B is part of C, then A is part of C

Antisymmetric: If A is part of B, then B is not part of A.

→ In aggregation, an aggregate object is made up of constituent objects but existance of constituent objects are not depending upon the aggregate object.

UML Notation

UML notation of aggregation is a small hollow diamond at the assembly class end.

e.g. A LawnMower consists of a Blade, an Engine, many Wheels and a deck.

LawnMower is the assembly class and other parts are constituent classes.

LawnMower to Blade is one aggregation,
LawnMower to Engine is another aggregation
LawnMower to Wheel is also an aggregation
and so on.

Multiplicity is specified for each individual pair of execution.

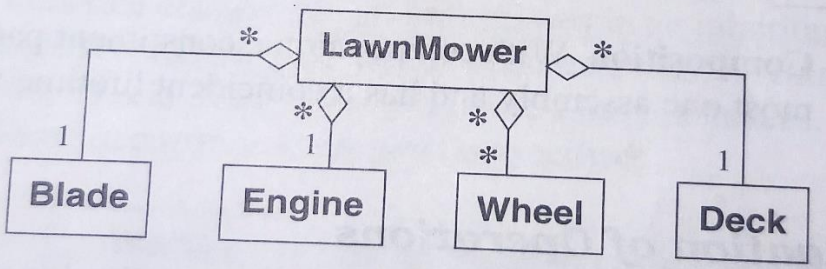


Figure 4.9 **Aggregation**. Aggregation is a kind of association in which an aggregate object is made of constituent parts.

Composition :

→ Composition is a part-whole relationship like aggregation but in a more restrictive form.

→ It is a form of aggregation with two additional constraints

→ A constituent part can belong to atmost one assembly.

→ Once a constituent part has been assigned to an assembly class, the constituent part has coincident lifetime with the assembly i.e. it lives and dies with the assembly class.

In other words the existance of the constituent part is dependent on the existance of the assembly class.

→ Composition implies ownership of the parts by the whole. Deletion of assembly objects triggers deletion of all constituent objects via composition.

UML notation

Small solid diamond next to the assembly class.

Consider the following example.

A company consists of divisions, which in turn consists of departments.

i.e. A company is a composition of divisions and a Division is a composition of departments.

But a company is not a composition of its employees.

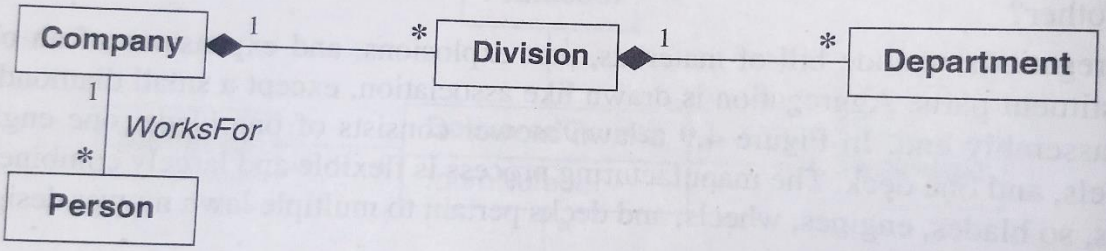


Figure 4.10 Composition. With composition a constituent part belongs to at most one assembly and has a coincident lifetime with the assembly.

Abstract class

→ An abstract class is a class that has no direct instances i.e. an abstract class can not be instantiated. (An abstract class can not be used to create any object.

→ An Abstract class has no objects but its sub classes have direct objects.

→ A concrete class is a class that has its own direct objects.

→ Abstract classes can define abstract methods.

→ Abstract method defines only the signature but no implementation with respect to the abstract class. The concrete subclasses inherit the signature of the abstract method and provide their own implementation details.

Note: → Generally concrete superclasses are avoided in inheritance mechanism.

→ Only concrete classes are the leaf classes in an inheritance hierarchy.

UML Notation :

- An abstract class in UML is specified in italic front or the keyword {abstract} is specified below or after the abstract class name.

The same UML notation is used for abstract methods.

In the following figure Employee class is an abstract class. The subclasses fullTimeEmployee and PartTimeEmployee are concrete subclasses.

Also, the Employee class has an abstract method ComputePay which has no implementation with respect to Employee class. The subclasses fullTimeEmployee and PartTimeEmployee provide their own implementation details for the abstract method ComputePay.

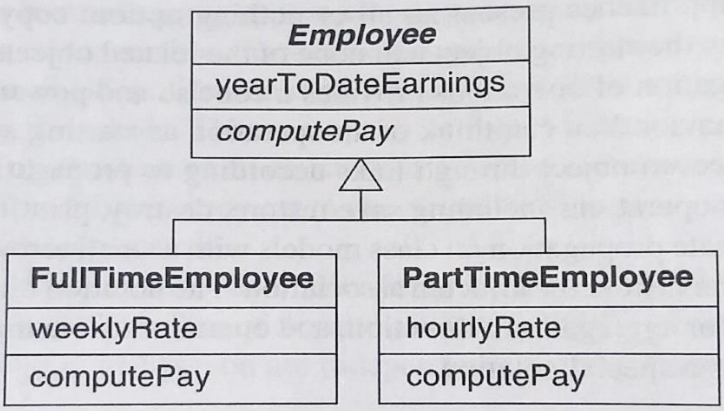


Figure 4.13 Abstract class and abstract operation. An abstract class is a class that has no direct instances

Or, it may be represented as follows