



# Object-Oriented Design

**Lecturer: Raman Ramsin**

## Lecture 8: Relationships



# Analysis Workflow: *Analyze a Use Case*

- The *analysis workflow* consists of the following activities:
  - Architectural analysis
  - **Analyze a use case**
    - **Outputs:**
      - **analysis classes**
      - **use case realizations**
  - Analyze a class
  - Analyze a package



# Relationships: Links and Associations

- Relationships are semantic connections between things.
  - *Links* are connections between objects.
  - *Associations* are connections between classes.
  - *Links* are instances of *associations* or *dependencies*.



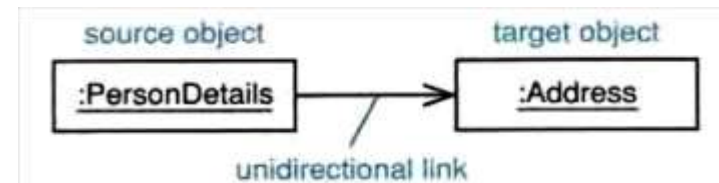
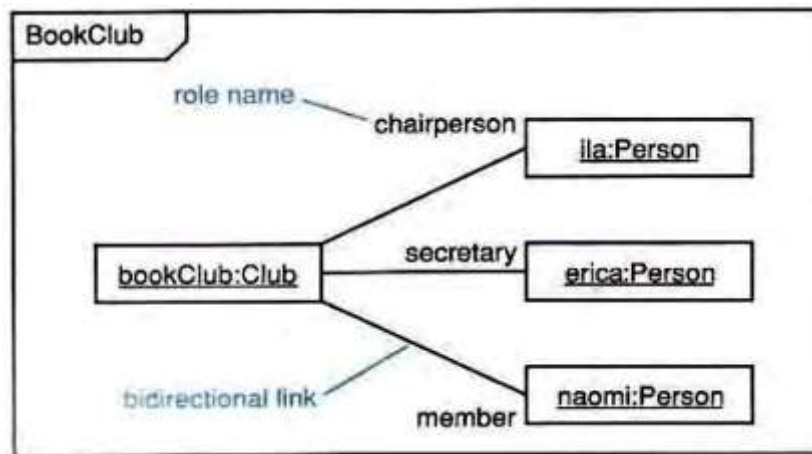
# Links

- A link occurs when one object holds an object reference to another object.
- Objects realize system behavior by collaborating:
  - collaboration occurs when objects send each other messages across links;
  - when a message is received by an object, it executes the appropriate operation.



# Links and Object Diagrams

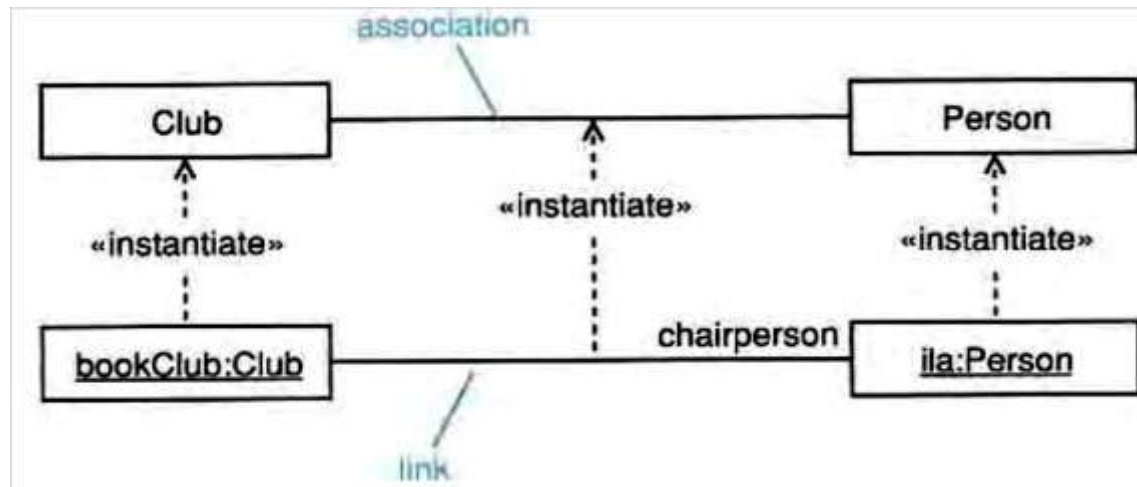
- Object diagrams show objects and their links at a particular point in time.
  - They are snapshots of an executing OO system at a particular time.
  - Objects may adopt roles with respect to each other - the role played by an object in a link defines the semantics of its part in the collaboration.
  - N-ary links may connect more than two objects - they are drawn as a diamond with a path to each object but are not widely used.





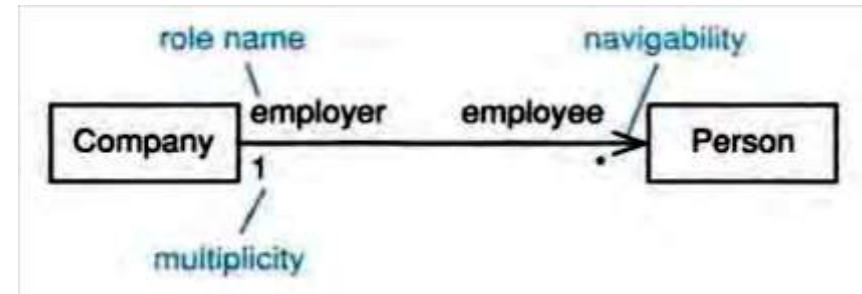
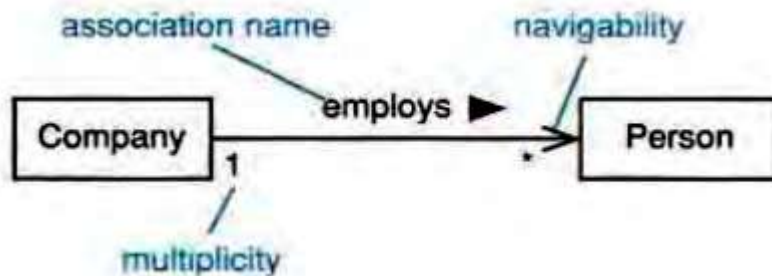
# Associations

- Associations are semantic connections between classes.
  - If there is a link between two objects, there *must* be an association or dependency between the classes of those objects.
  - Links are instances of associations just as objects are instances of classes.



# Associations: Details

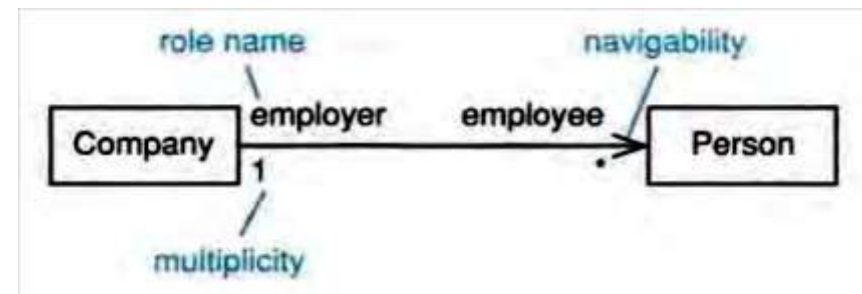
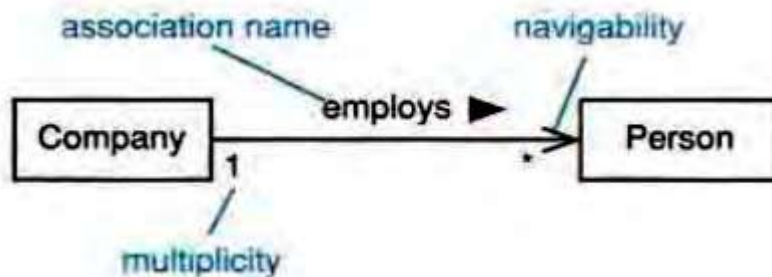
- Associations may optionally have the following:
  - Association name
  - Role names
  - Multiplicity
  - Navigability





# Associations: Names and Roles

- Association name:
  - may be prefixed or postfixed with a small black arrowhead to indicate the direction in which the name should be read;
  - should be a verb or verb phrase;
  - in lowerCamelCase.
- Role names:
  - on one or both association ends;
  - should be a noun or noun phrase describing the semantics of the role;
  - in lowerCamelCase.
- Use either an association name or role names but *not* both.

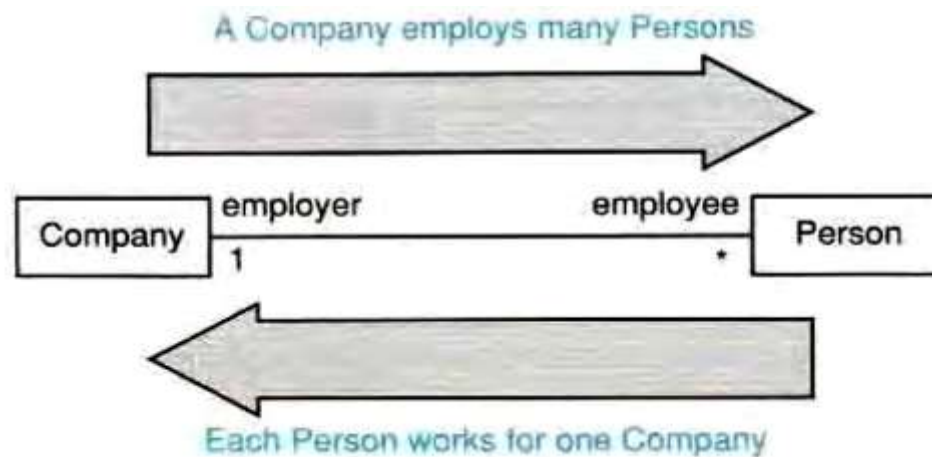






# Associations: Multiplicity

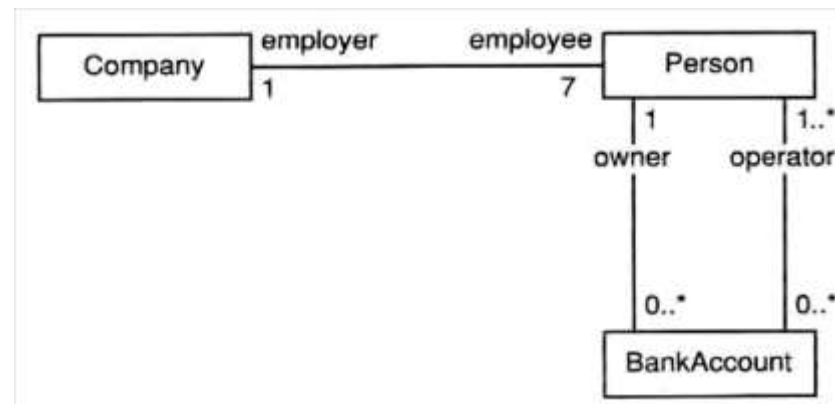
- Indicates the number of objects that can be involved in the relationship at any point in time.
- Objects may come and go, but multiplicity constrains the number of objects in the relationship at any point in time.
- Multiplicity is specified by a comma-separated list of intervals, for example, 0..1, 3..5.
- There is no default multiplicity - if multiplicity is not explicitly shown, then it is undecided.





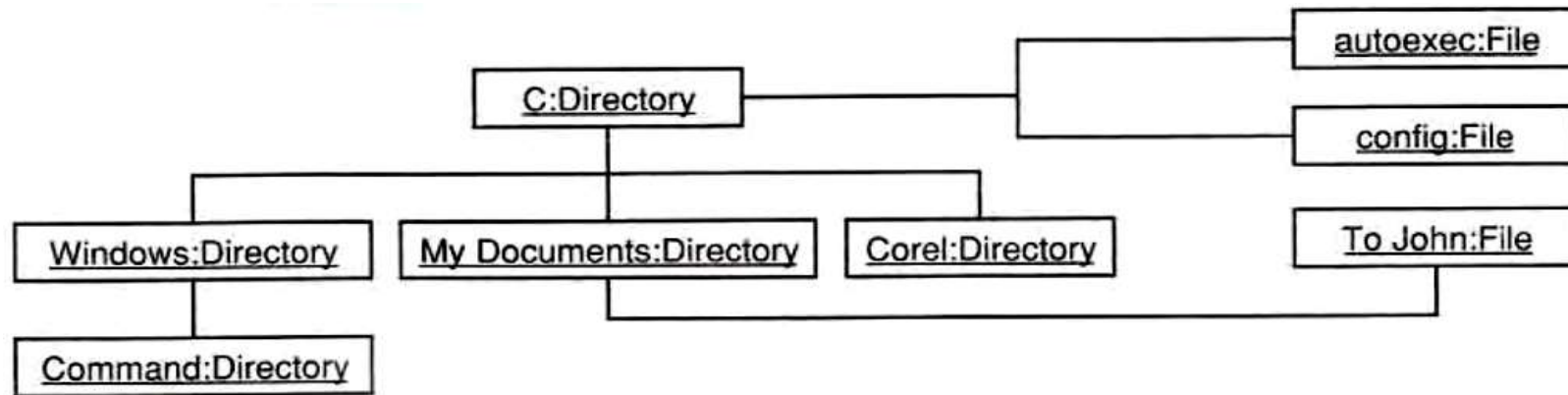
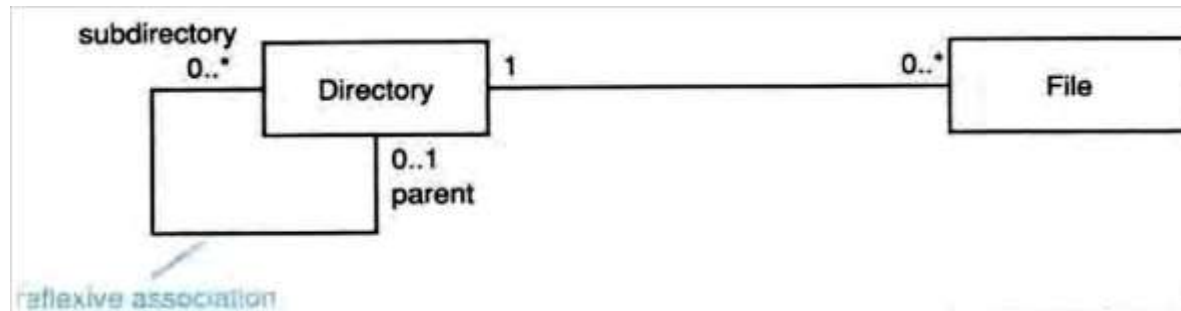
# Associations: Multiplicity Syntax

Adornment	Semantics
0..1	Zero or 1
1	Exactly 1
0..*	Zero or more
*	Zero or more
1..*	1 or more
1..6	1 to 6
1..3, 7..10, 15, 19..*	1 to 3 <i>or</i> 7 to 10 <i>or</i> 15 exactly <i>or</i> 19 to many

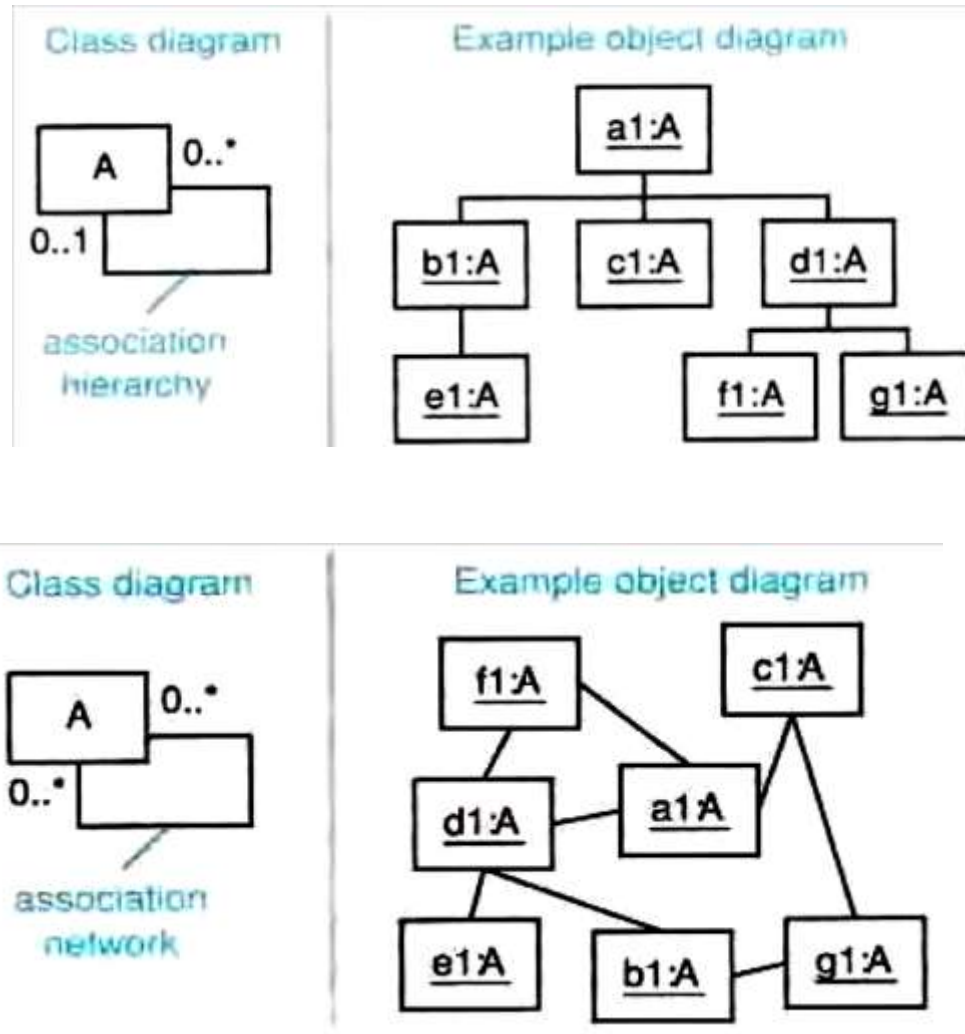




# Reflexive Associations



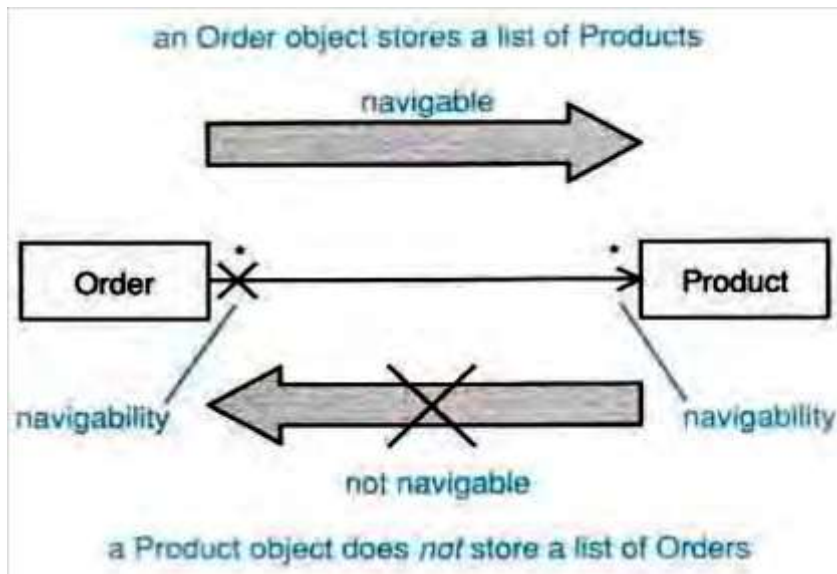
# Reflexive Associations: Hierarchies and Networks



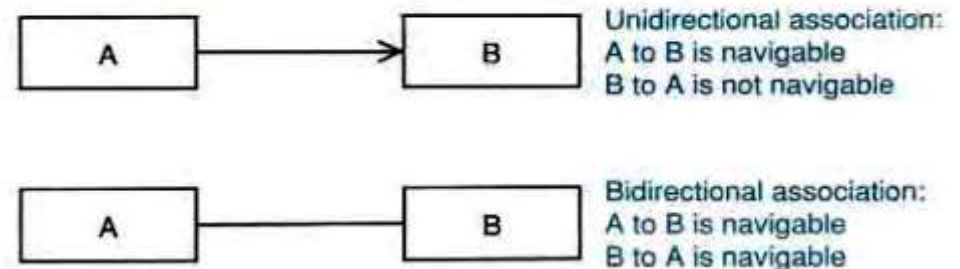


# Associations: Navigability

- Shown by an arrowhead on one end of the relationship - if a relationship has no arrowheads, then it is bidirectional.
- Navigability indicates that you can traverse the relationship in the direction of the arrow.
- You may also be able to traverse back the other way, but it will be computationally expensive to do so.



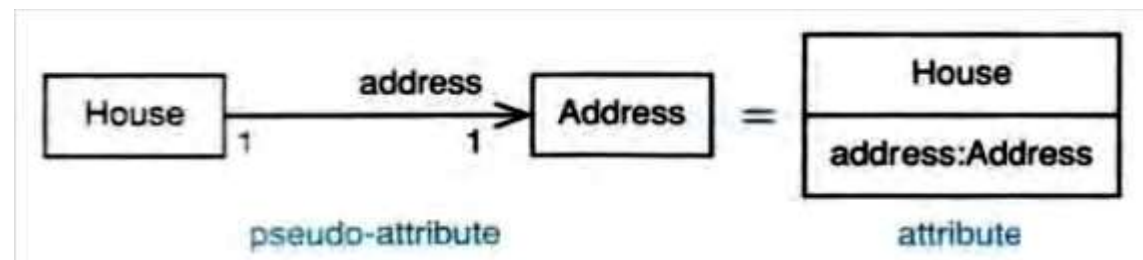
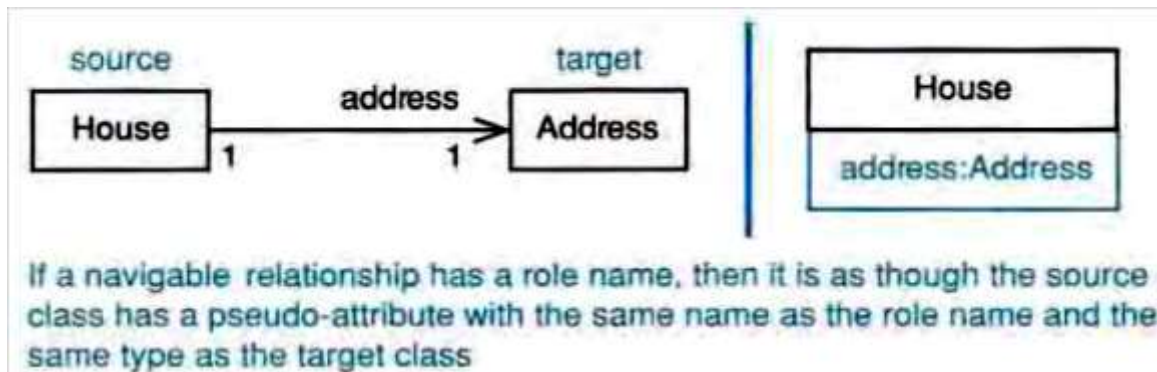
Visibility Idiom 3 is used almost exclusively in practice





# Associations and Attributes

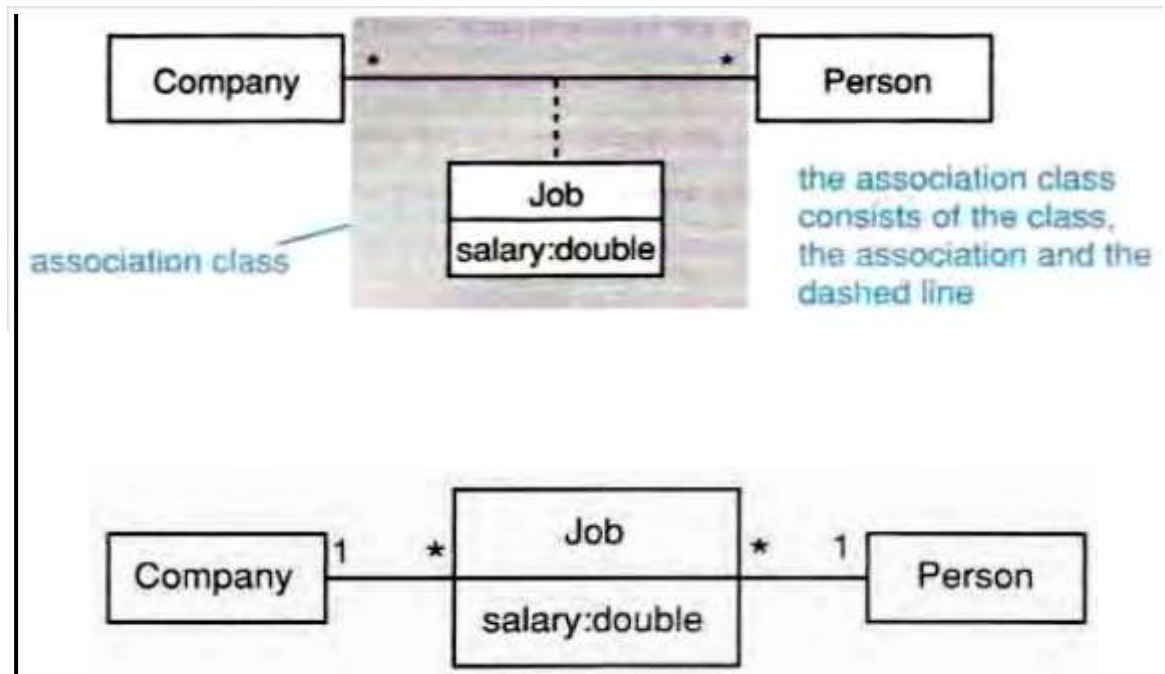
- An association between two classes is equivalent to one class having a pseudo-attribute that can hold a reference to an object of the other class:
  - you can often use associations and attributes interchangeably;
  - use association when you have an important class on the end of the association that you wish to emphasize;
  - use attributes when the class on the end of the relationship is unimportant (e.g., a library class such as String or Date).





# Association Classes

- An association class is an association that is also a class:
  - it may have attributes, operations, and relationships;
  - you can use an association class when there is exactly one unique link between any pair of objects at any point in time;
  - if a pair of objects may have many links to each other at a given point in time, then you reify the relationship by replacing it with a normal class.

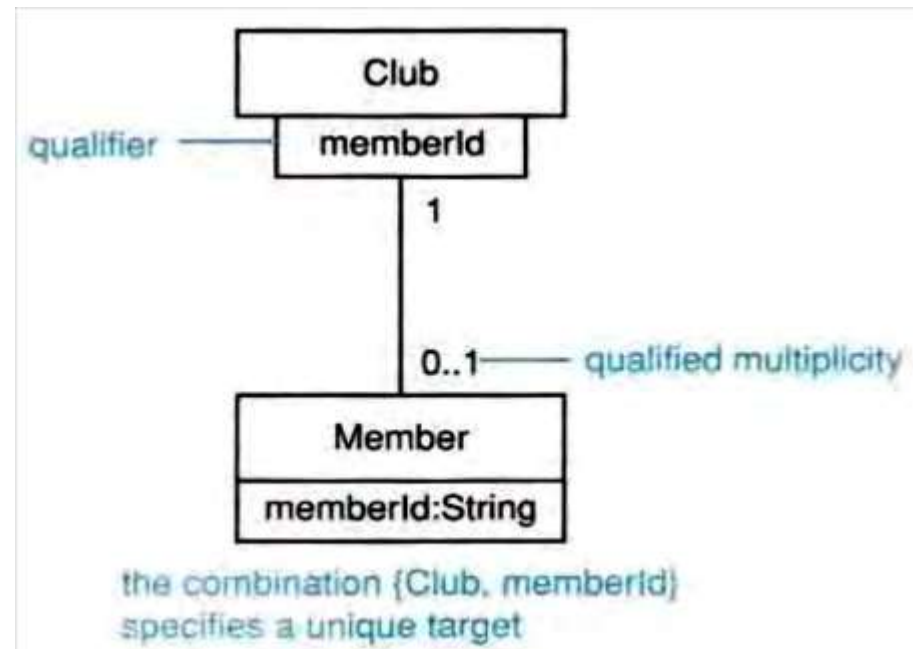
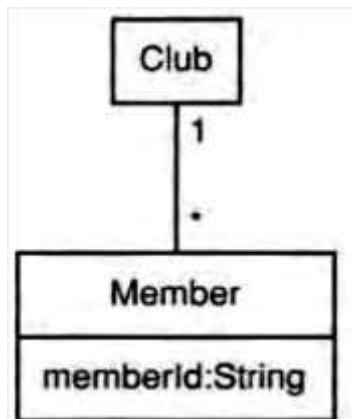






# Qualified Associations

- Qualified associations use a qualifier to select a unique object from the target set:
  - the qualifier must be a unique key into the target set;
  - qualified associations reduce the multiplicity of n-to-many relationships, to n-to-one;
  - they are a useful way of drawing attention to unique identifiers.







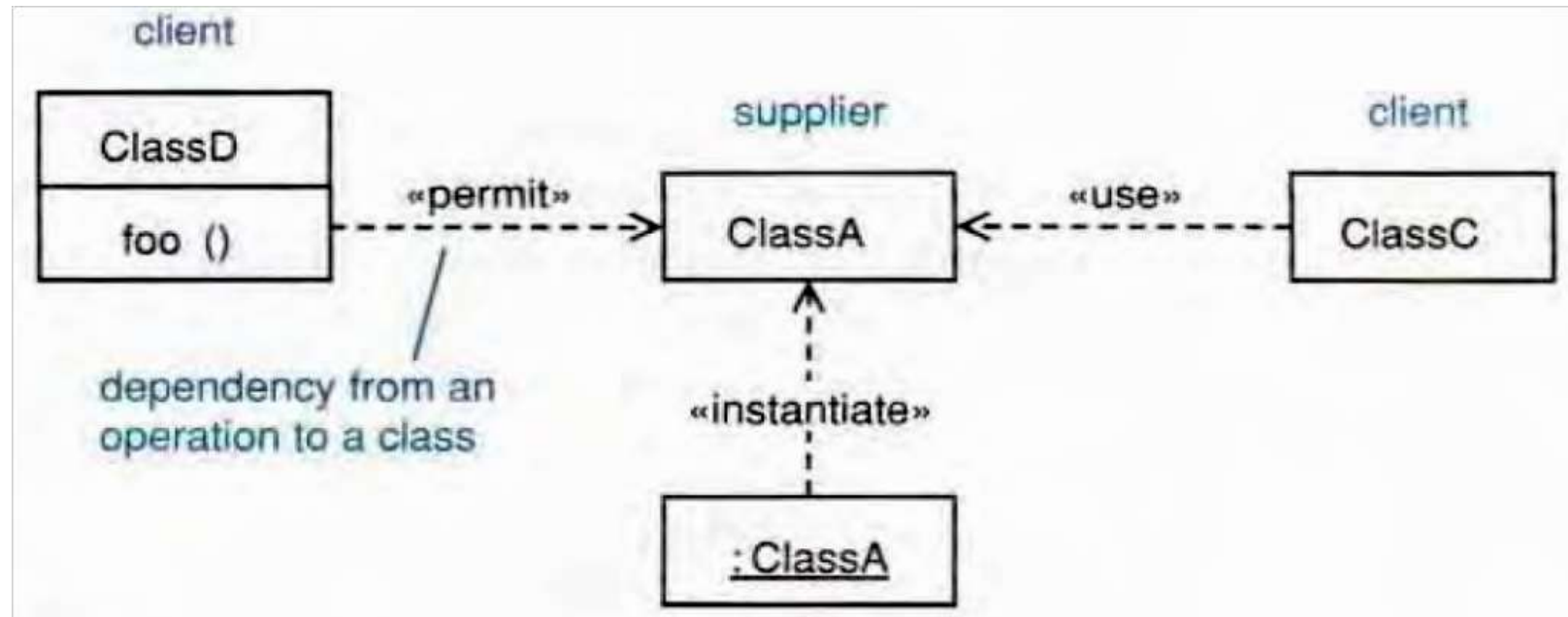
# Dependencies

- Dependencies are relationships in which a change to the supplier affects or supplies information to the client.
- The client depends on the supplier in some way.
- Dependencies are drawn as a dashed arrow from client to supplier.

Type	Semantics
Usage	The client uses some of the services made available by the supplier to implement its own behavior – this is the most commonly used type of dependency
Abstraction	<p>This indicates a relationship between client and supplier, where the supplier is more abstract than the client.</p> <p>What do we mean by “more abstract”? This could mean that the supplier is at a different point in development than the client (e.g., in the analysis model rather than the design model)</p>
Permission	The supplier grants some sort of permission for the client to access its contents – this is a way for the supplier to control and limit access to its contents



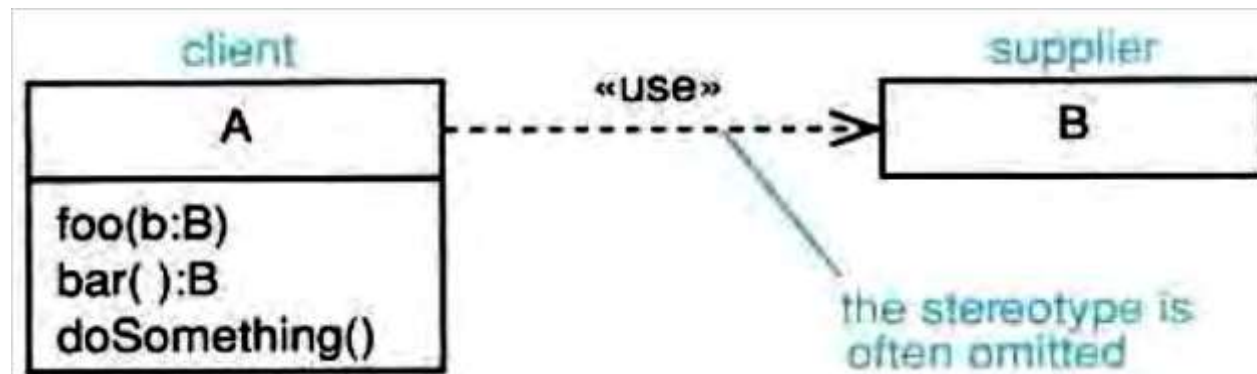
# Dependencies: Example





# Usage Dependencies

- ❑ «use»- the client makes use of the supplier in some way - this is the catch-all.
- ❑ «call»- the client operation invokes the supplier operation.
- ❑ «parameter»- the supplier is a parameter or return value from one of the client's operations.
- ❑ «send»- the client sends the supplier (which must be a signal) to the specified target.
- ❑ «instantiate»- the client is an instance of the supplier.





# Abstraction Dependencies

- «trace»- the client is a historical development of the supplier.
- «substitute»- the client can be substituted for the supplier at runtime.
- «refine»- the client is a version of the supplier.
- «derive»- the client can be derived in some way from the supplier:
  - you may show derived relationships explicitly by using a «derive» dependency;
  - you may show derived relationships by prefixing the role or relationship name with a slash;
  - you may show derived attributes by prefixing the attribute name with a slash.



# Abstraction Dependencies: «derive»

Model	Description
<pre> classDiagram     class BankAccount     class Transaction     class Quantity     BankAccount "1" -- "0..*" Transaction     Transaction "1" -- "1" Quantity     BankAccount "1" -- "1" Quantity : balance     Transaction ..&gt; Quantity : «derive»         </pre>	<p>The BankAccount class has a derived association to Quantity where Quantity plays the role of the balance of the BankAccount</p> <p>This model emphasizes that the balance is derived from the BankAccount's collection of Transactions</p>
<pre> classDiagram     class BankAccount     class Transaction     class Quantity     BankAccount "1" -- "0..*" Transaction     Transaction "1" -- "1" Quantity     BankAccount "1" -- "1" Quantity : /balance         </pre>	<p>In this case a slash is used on the role name to indicate that the relationship between BankAccount and Quantity is derived</p> <p>This is less explicit as it does not show what the balance is derived from</p>
<pre> classDiagram     class BankAccount     class Transaction     class Quantity     BankAccount "1" -- "0..*" Transaction     Transaction "1" -- "1" Quantity     class BankAccount {         /balance:Quantity     }         </pre>	<p>Here the balance is shown as a derived attribute – this is indicated by the slash that prefixes the attribute name</p> <p>This is the most concise expression of the dependency</p>



# Permission Dependencies

- «access»- a dependency between packages where the client package can access all of the public contents of the supplier package – the name spaces of the packages remain separate.
- «import»- a dependency between packages where the client package can access all of the public contents of the supplier package – the namespaces of the packages are merged.
- «permit»- a controlled violation of encapsulation where the client may access the private members of the supplier - this is not widely supported and should be avoided if possible.



## *Reference*

- Arlow, J., Neustadt, I., *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, 2<sup>nd</sup> Ed. Addison-Wesley, 2005.