Slide 6

UML CLASS DIAGRAMS

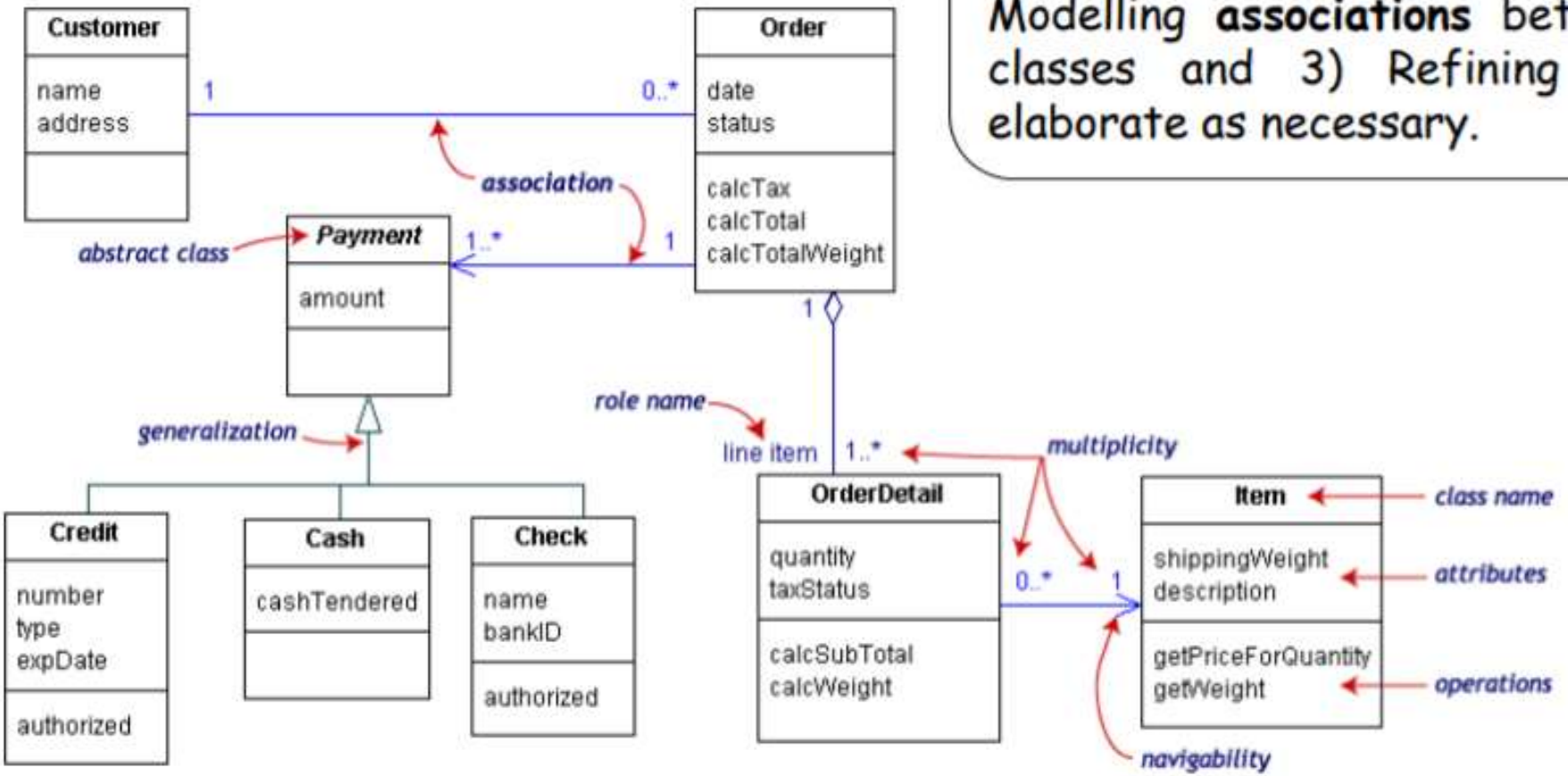
CLASSES

- Classes are the building blocks of OO Modelling
- A class diagram provide a conceptual model of the system in terms of Entities and their relationship

CLASS DIAGRAM

- Class diagrams provide a structural view of systems.
- Class diagrams capture the static structure of Object-Oriented systems, or how they are structured rather than how they behave.
- Class diagrams support architectural design.
- Class diagrams represents the basics of Object Oriented systems. They identify what classes there are, how they interrelate and how they interact.

These diagrams contain classes and associations. Construction involves: 1) Modelling **Classes**, 2) Modelling **associations** between classes and 3) Refining and elaborate as necessary.



Class Diagrams in the Life Cycle

- They can be used throughout the development life cycle
- Class diagram carry different information depending on the phase of the development process and the level of detail being considered.
    - Initially, class diagrams reflect the problem domain, which is familiar to end-users
    - As development progresses, class diagrams move towards the implementation domain, which is familiar to software engineers
- The contents of a class diagram will reflect this change in emphasis during the development process.

Class Diagrams - Basics

Classes

- Basic Class Components
- Attributes and Operations

Class Relationships

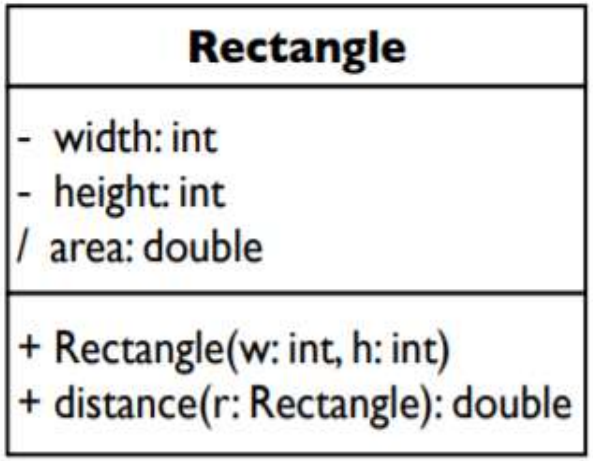
- Associations
- Generalizations
- Aggregations and Compositions
- Dependency

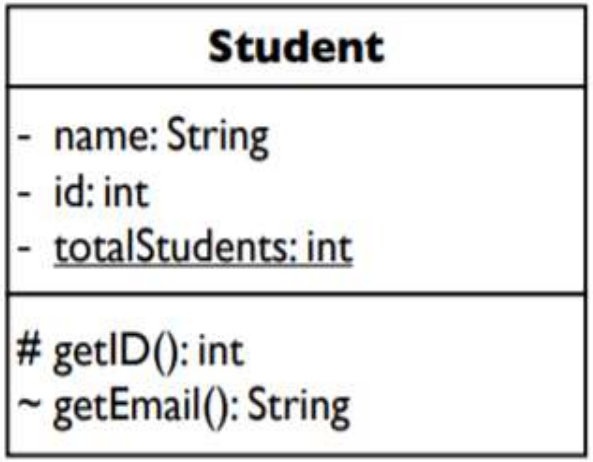
CLASS DIAGRAM

Each class is represented as a rectangle subdivided into three compartment

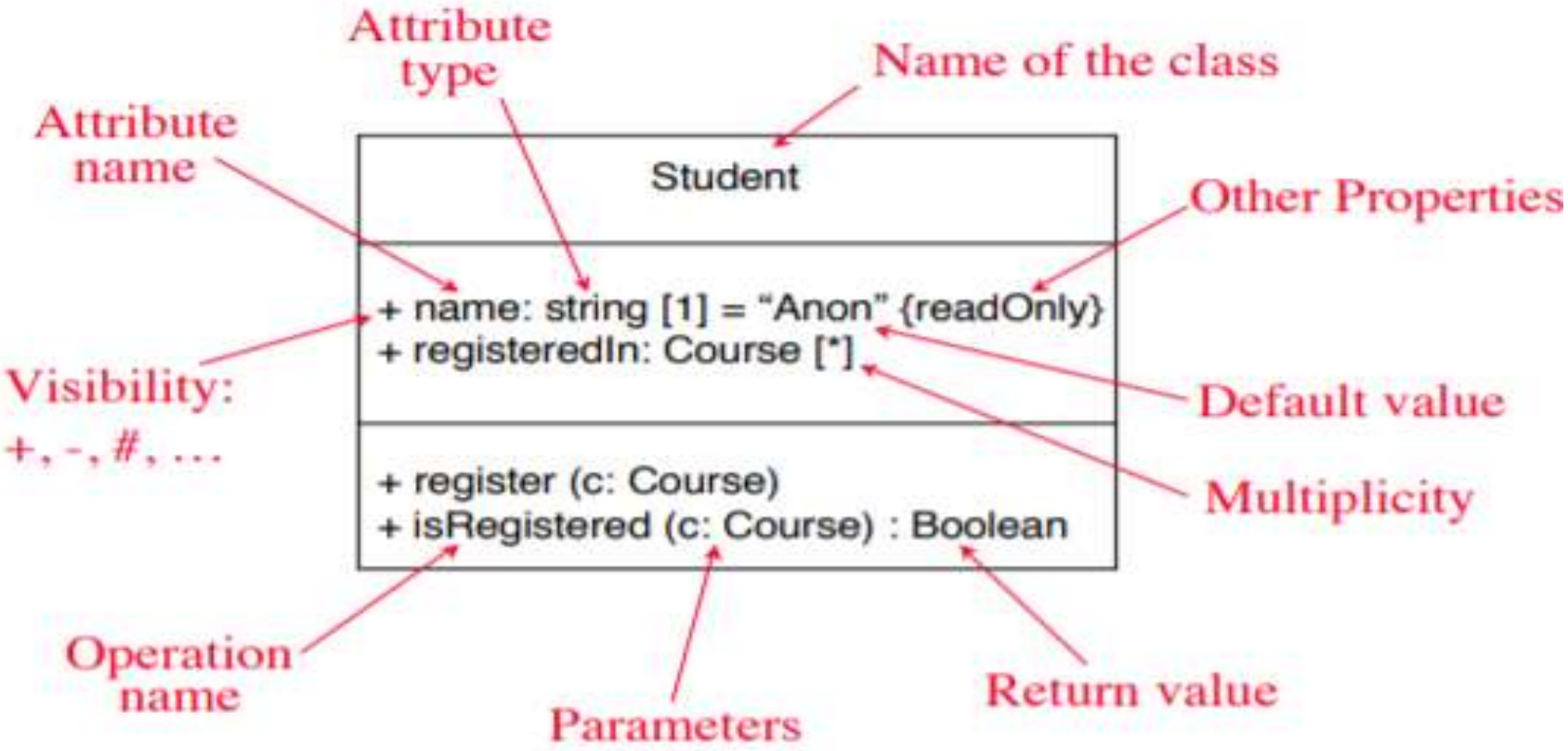
- Class name – identifies the class
- Attributes - fields of the class
- Operations / methods/functions

- Class name
  - write «interface» on top of interfaces' names
  - use *italics* for an abstract class name
- Attributes (optional)
  - fields of the class
- Operations / methods (optional)
  - may omit trivial (get/set) methods
  - but don't omit any methods from an interface!
  - should not include inherited methods





ELEMENTS OF A CLASS DIAGRAM



Attributes and Operations

- < featureName >:<type>
- **Type** is the data type of the attribute or the data returned by the operation
- **Visibility**: private (**-**), public (**+**) or protected (**#**)
- **Attributes**
  - **Initial value**, **Derived Attribute**, Multiplicity [m..n]
  - Examples of **Multiplicity**: n..m - n to m instances; **0..1** - zero or one instance; **0..*** or ***** - no limit on the number of instances (including none). **1** - exactly one instance; **1..*** at least one instance
- **Operations**
  - Parameters (passed by **value** or by **reference**), Method Note, Grouping by **Stereotype**
  - A **Method Note** captures the actual implementation of operations

VISIBILITY OF CLASS MEMBERS

Modifiers are used to indication the visibilty of attributes and operations

+ Denote Public visibility ; Accessible to other classes

- Denote Private visibility; Accessible to members of that class only

Denote Protected Visibility; Accessible to members of that class and classes directly derived from that class

Modelling by Class Diagrams

Class Diagrams (models)

- From a conceptual viewpoint, reflect the requirements of a problem domain
- From a specification (or implementation) viewpoint, reflect the intended design or implementation, respectively, of a software system

Producing class diagrams involve the following iterative activities:

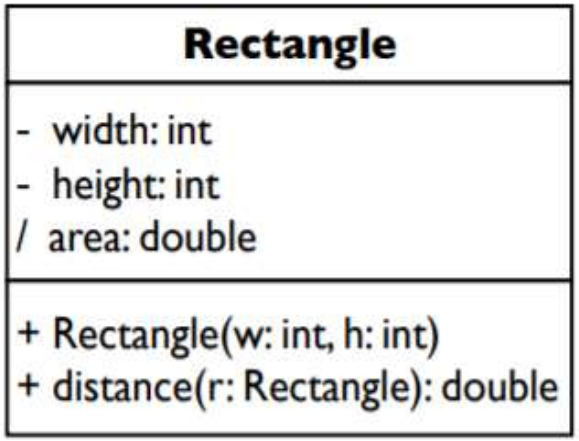
- Find classes and associations (directly from the use cases)
- Identify attributes and operations and allocate to classes
- Identify generalization structures

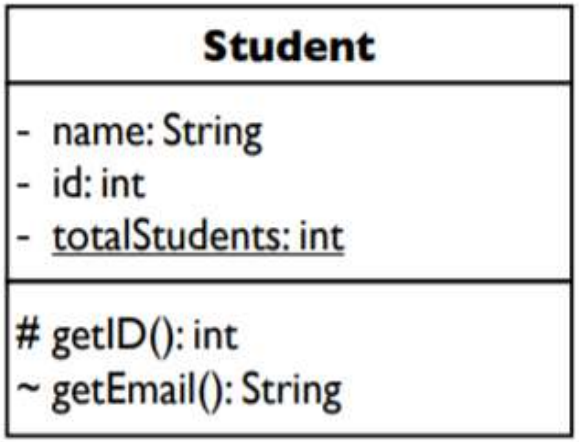
How to design classes

- Identify classes and interactions from project requirements:
- Nouns are potential classes, objects, and fields
- Verbs are potential methods or responsibilities of a class
- Relationships between nouns are potential interactions (containment, generalization, dependence, etc.)

visibility name : type [count] = default_value

- visibility
  - + public
  - # protected
  - - private
  - ~ package (default)
  - / derived
- **derived attribute**: not stored, but can be computed from other attribute values
  - "specification fields" from CSE 331
- underline static attributes





What UML class diagrams can show

Division of Responsibility

- Operations that objects are responsible for providing

Subclassing

- Inheritance, generalization

Navigability / Visibility

- When objects need to know about other objects to call their operations

Aggregation / Composition

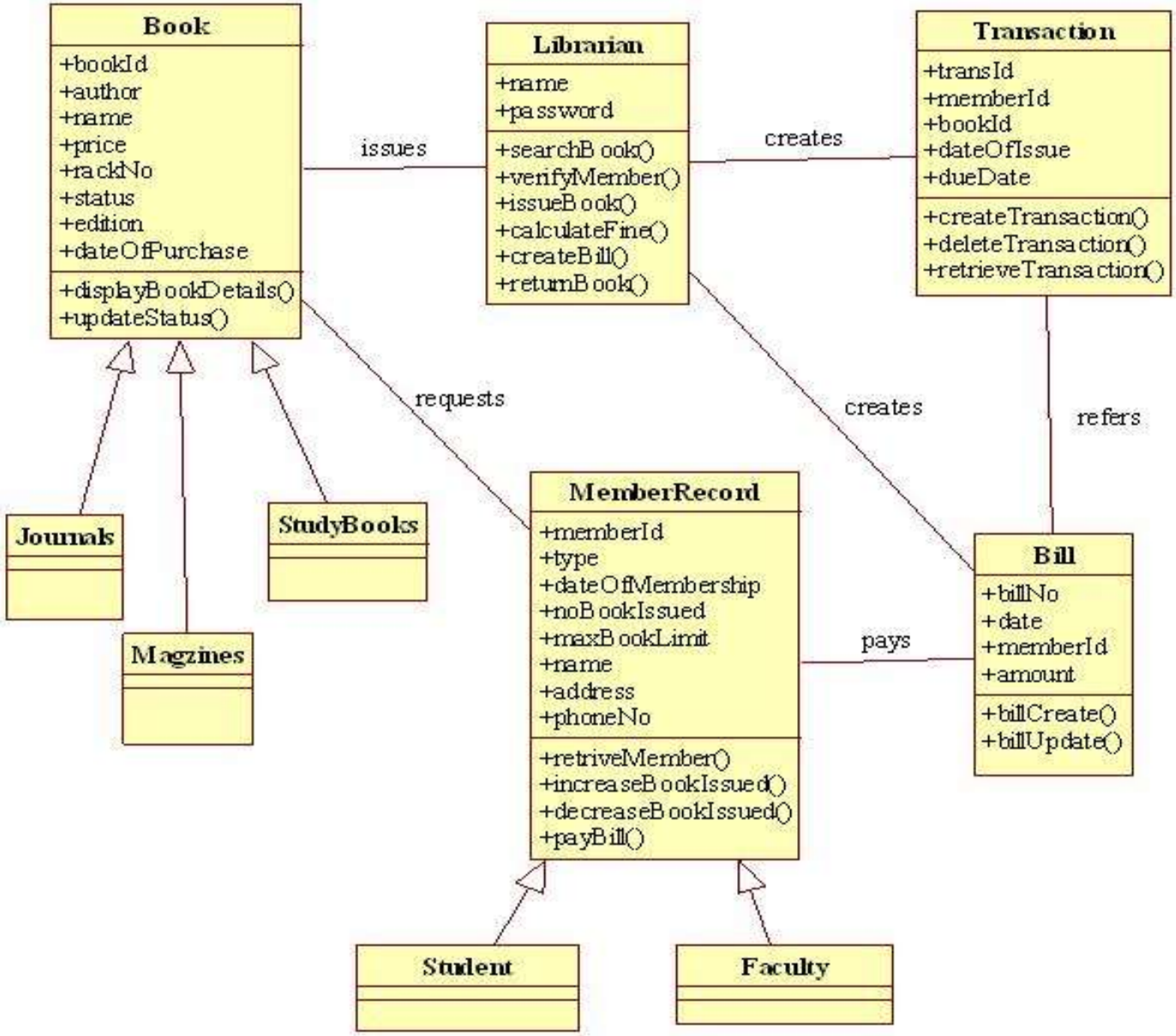
- When objects are part of other objects

Dependencies

- When changing the design of a class will affect other classes

Interfaces

- Used to reduce coupling between objects


Book
+bookId
+author
+name
+price
+rackNo
+status
+edition
+dateOfPurchase
+displayBookDetails()
+updateStatus()
issues
Librarian
+name
+password
+searchBook()
+verifyMember()
+issueBook()
+calculateFine()
+createBill()
+returnBook()
creates
Transaction
+transId
+memberId
+bookId
+dateOfIssue
+dueDate
+createTransaction()
+deleteTransaction()
+retrieveTransaction()
requests
creates
refers
Journals
StudyBooks
Magzines
MemberRecord
+memberId
+type
+dateOfMembership
+noBookIssued
+maxBookLimit
+name
+address
+phoneNo
+retriveMember()
+increaseBookIssued()
+decreaseBookIssued()
+payBill()
pays
Bill
+billNo
+date
+memberId
+amount
+billCreate()
+billUpdate()
Student
Faculty

TYPES OF CLASSES

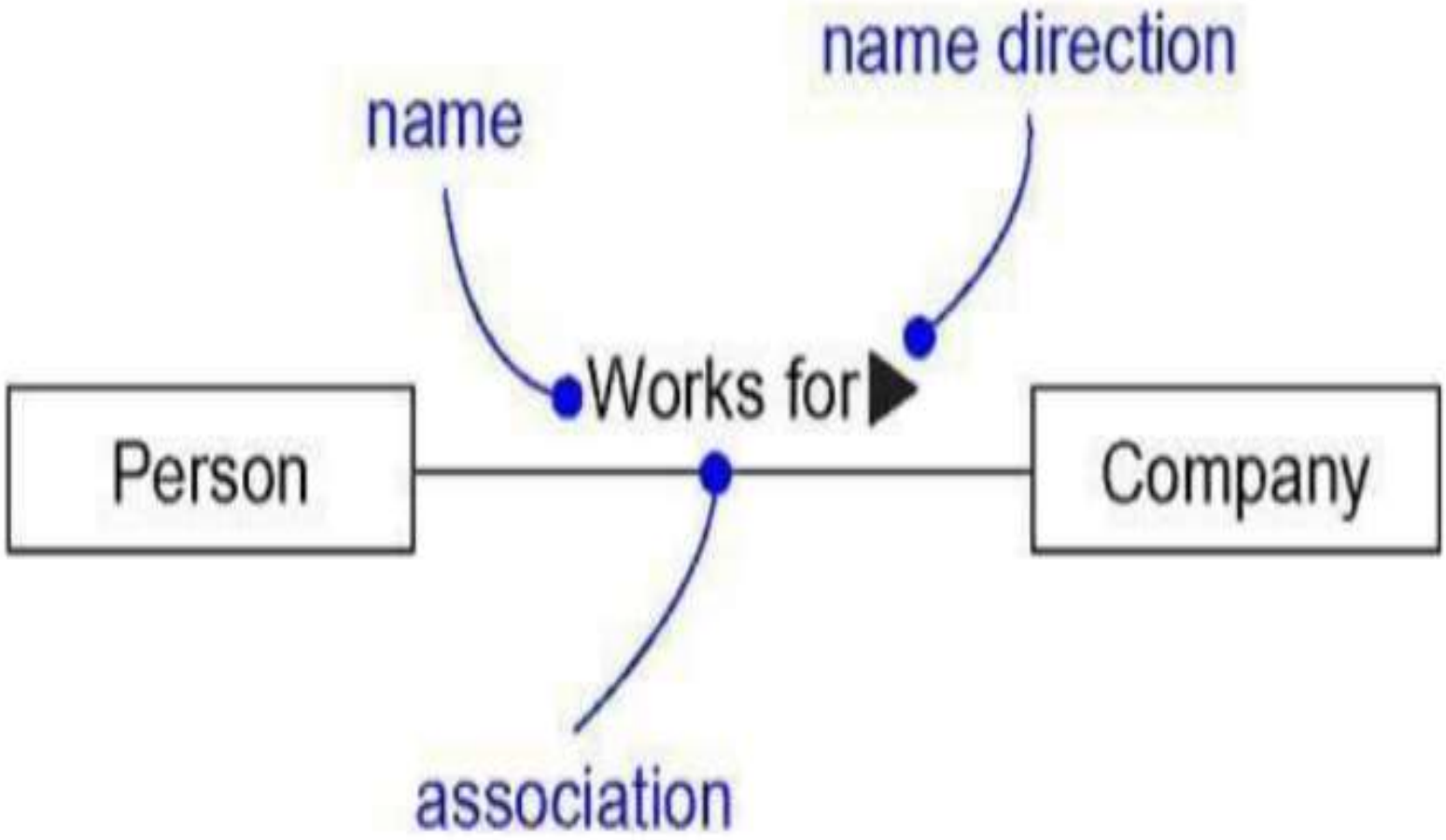
- ENTITY CLASSES –Represent the information that the system uses. Examples Customer, Book, Supplier, Student
- BOUNDARY CLASSES – Represent the interaction between the system and its actors. Examples GUI, Msg box, Dialog box
- CONTROL CLASSES – Represent the control logic of the system. They implement the flow of events as given in a use case
- DATA STORE CLASSES – Encapsulates the design decisions about data storage and retrieval strategies. This provides the flexibility to move an application from one database platform to another

Association

- An association is a structural relationship that specifies that objects of one thing are connected to objects of another.
- Given an association connecting two classes, you can navigate from an object of one class to an object of the other class, and vice versa.

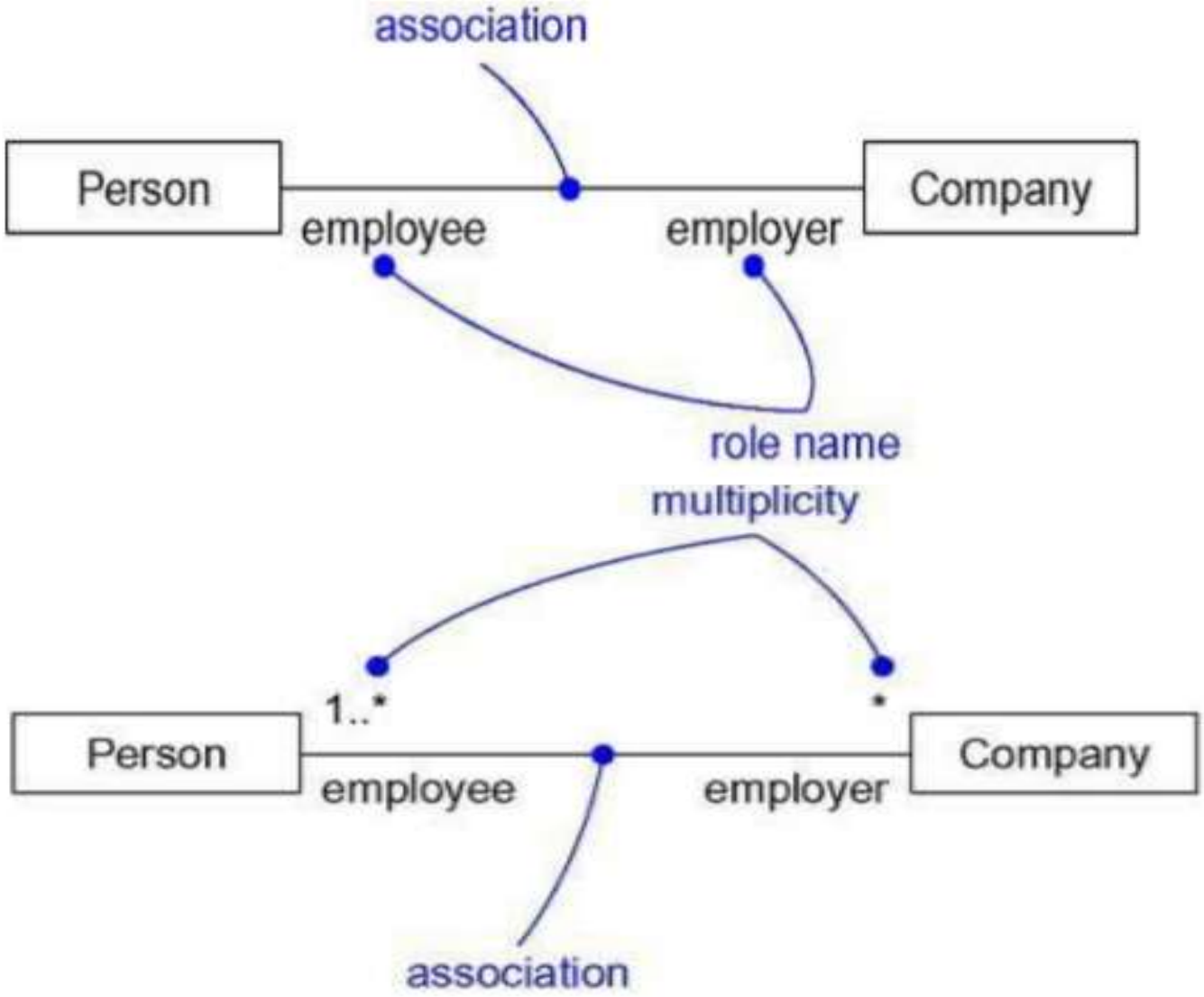
Components of an Association

- Name An association can have a name, and you use that name to describe the nature of the relationship. So that there is no ambiguity about its meaning, you can give a direction to the name by providing a direction triangle that points in the direction you intend to read the name, as shown in Figure


name direction
name
Works for
Person
Company
association

Role • When a class participates in an association, it has a specific role that it plays in that relationship;
A role is just the face the class at the near end of the association presents to the class at the other end of the association.

Multiplicity • An association represents a structural relationship among objects. In many modeling situations, it's important for you to state how many objects may be connected across an instance of an association. • This "how many" is called the multiplicity of an association's role, and is written as an expression that evaluates to a range of values or an explicit value


association
Person
employee
employer
Company
role name
multiplicity
1..*
*
Person
employee
employer
Company
association

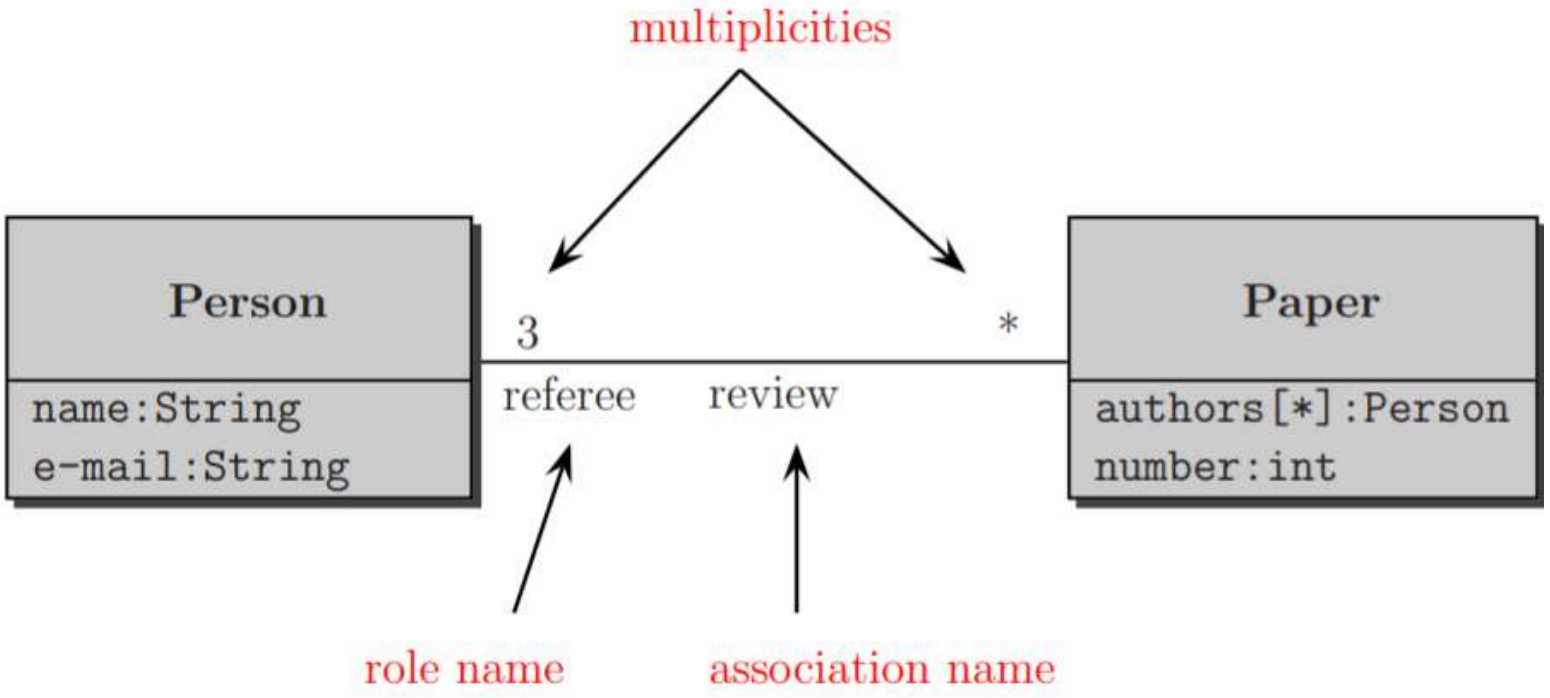
ASSOCIATION MULTIPLICITY

- Multiplicity depicts the cardinality of a class in relation to another
- Indicates the number of instances of one classes linked to another instance of the other class

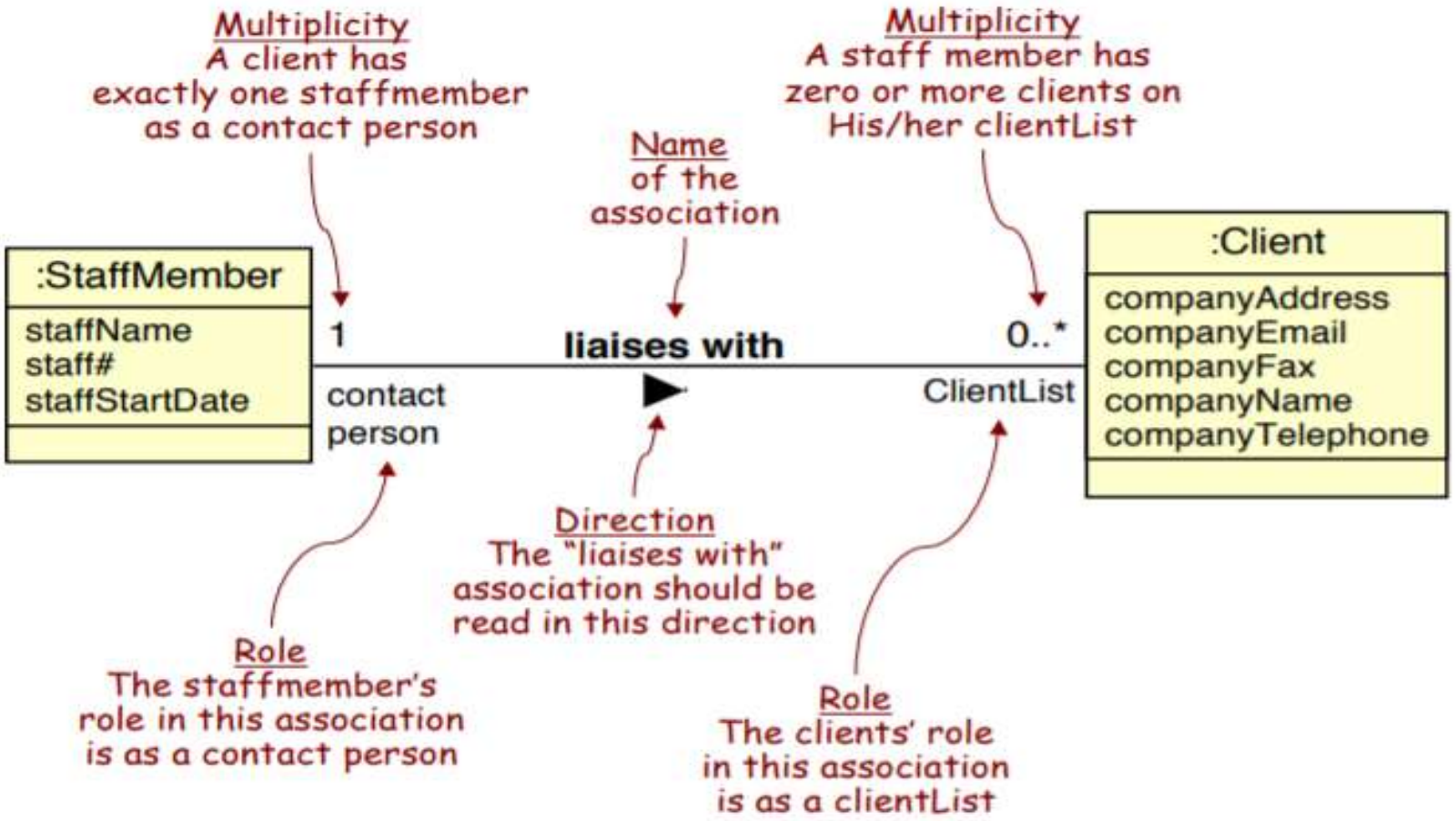
Some examples of specifying multiplicity:

- Optional (0 or 1) 0..1
- Exactly one 1 = 1..1
- Zero or more 0..* = *
- One or more 1..*
- A range of values 2..6

Associations

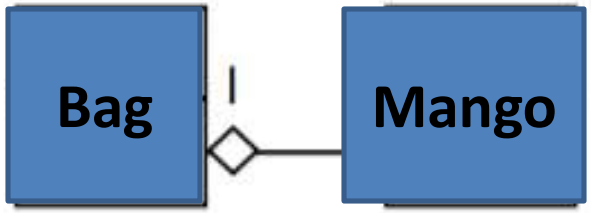


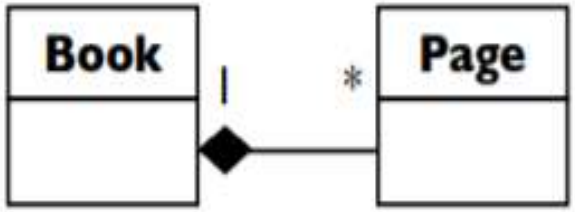
Class associations



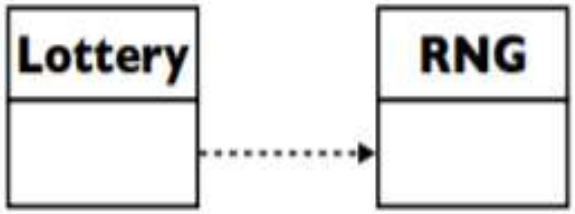
ASSOCIATION TYPES

- **Aggregation**: "is part of"
  - symbolized by a clear white diamond
- **Composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond
- **Dependency**: "uses temporarily"
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of the object's state





Mango



ASSOCIATION TYPES

- Aggregation – Container/Containee Relationship. May exist independent of each other
- Composition – Whole/Part relationship Components can not exist independently

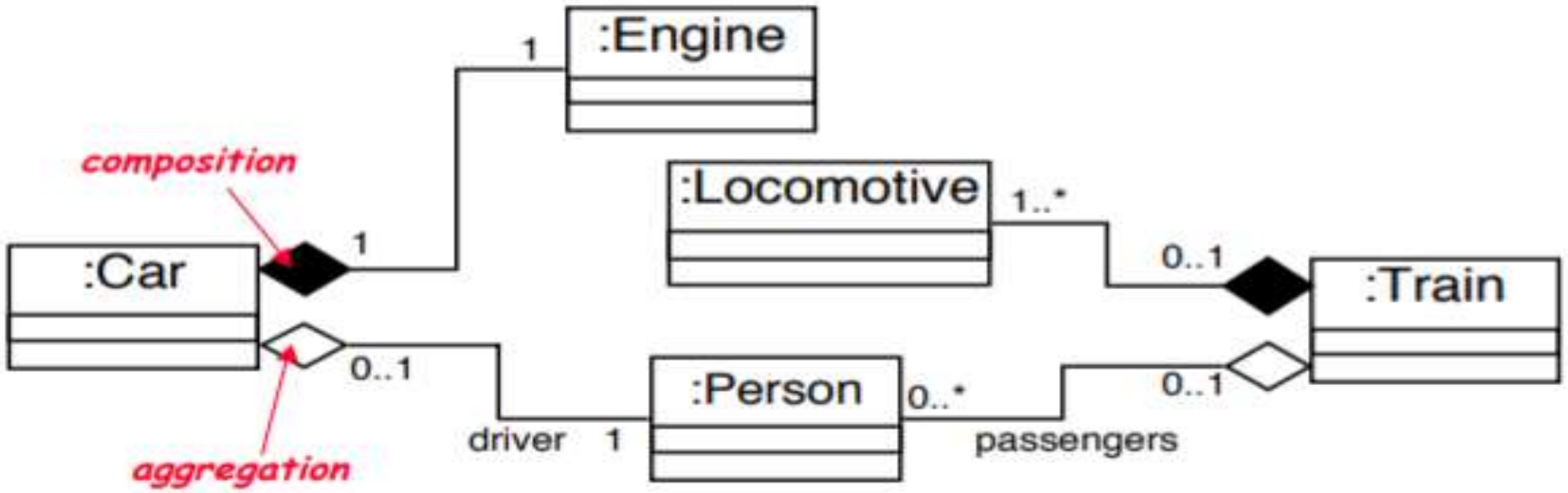
Aggregation and Composition

Aggregation

- This is the "Has-a" or "Whole/part" relationship

Composition

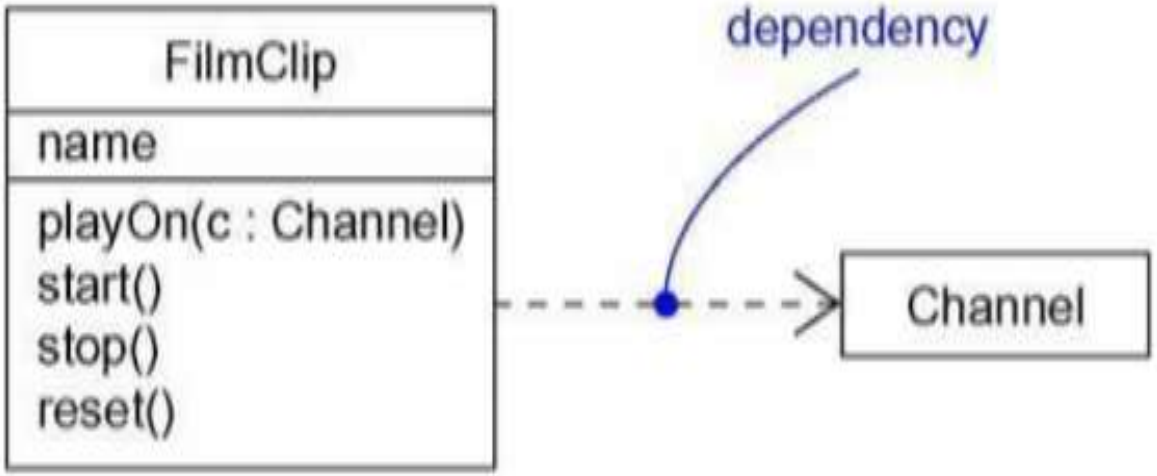
- Strong form of aggregation that implies ownership:
  - if the whole is removed from the model, so is the part.
  - the whole is responsible for the disposition of its parts



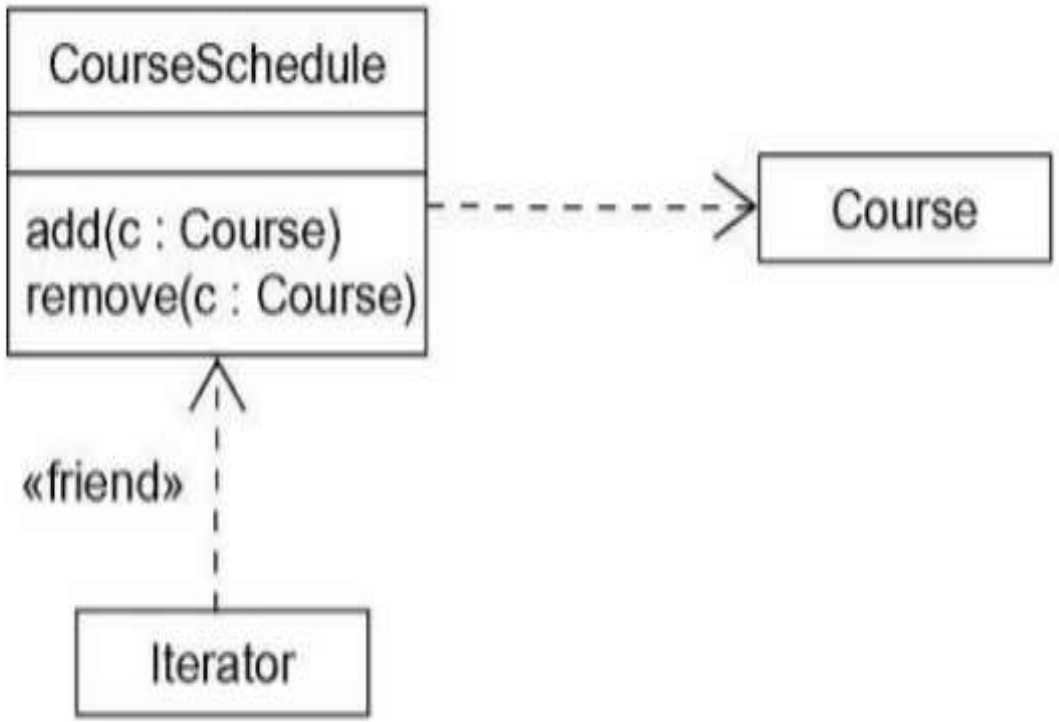
Dependency

A *dependency* is a using relationship that states that a change in specification of one thing (for example, class **Event**) may affect another thing that uses it (for example, class **Window**), but not necessarily the reverse.

- Graphically, a dependency is rendered as a dashed directed line.



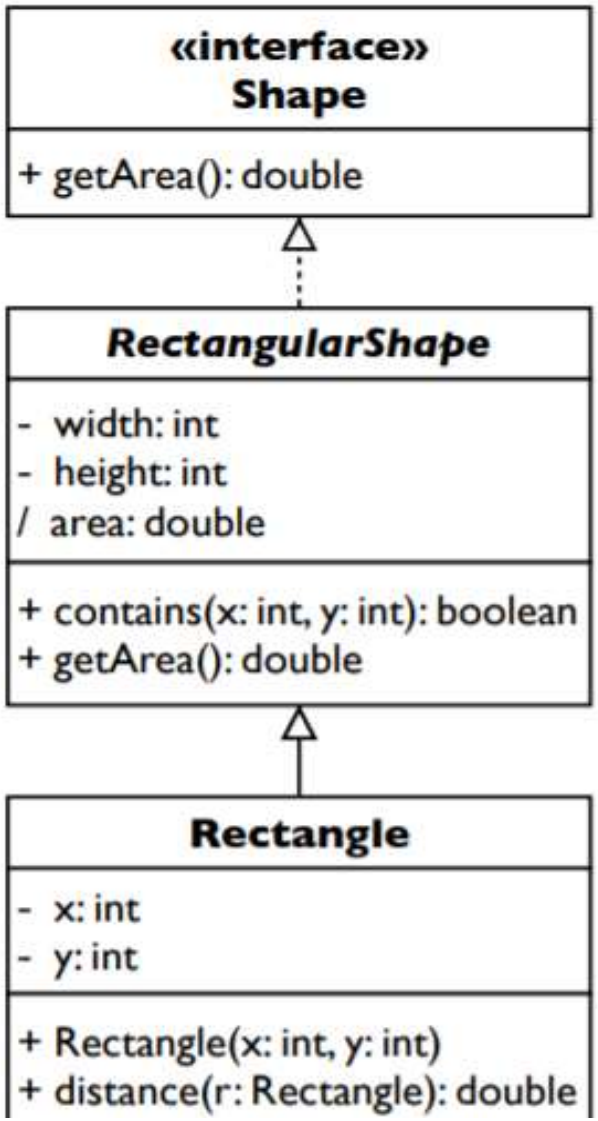
Create a dependency pointing from the class with the operation to the class used as a parameter in the operation.

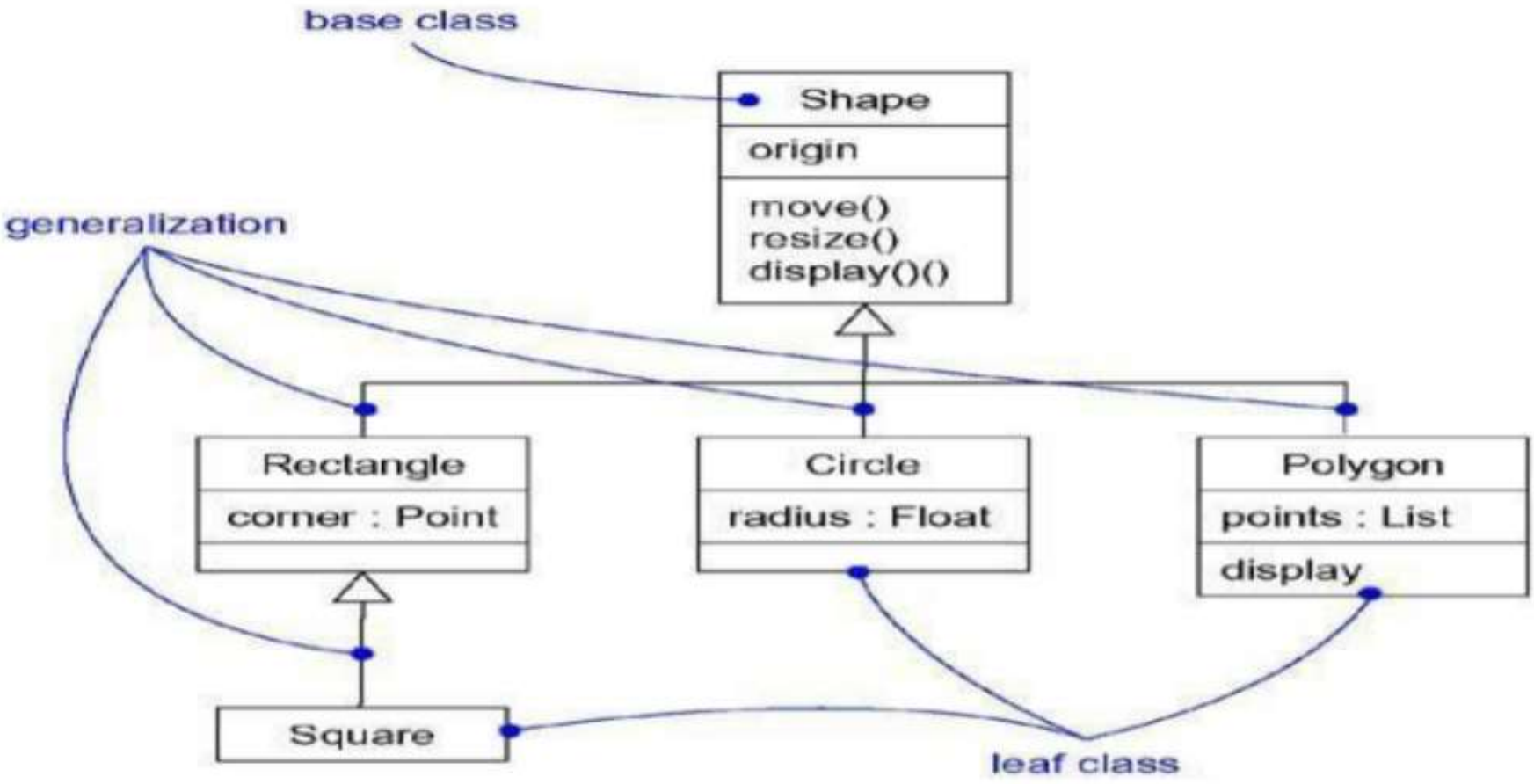


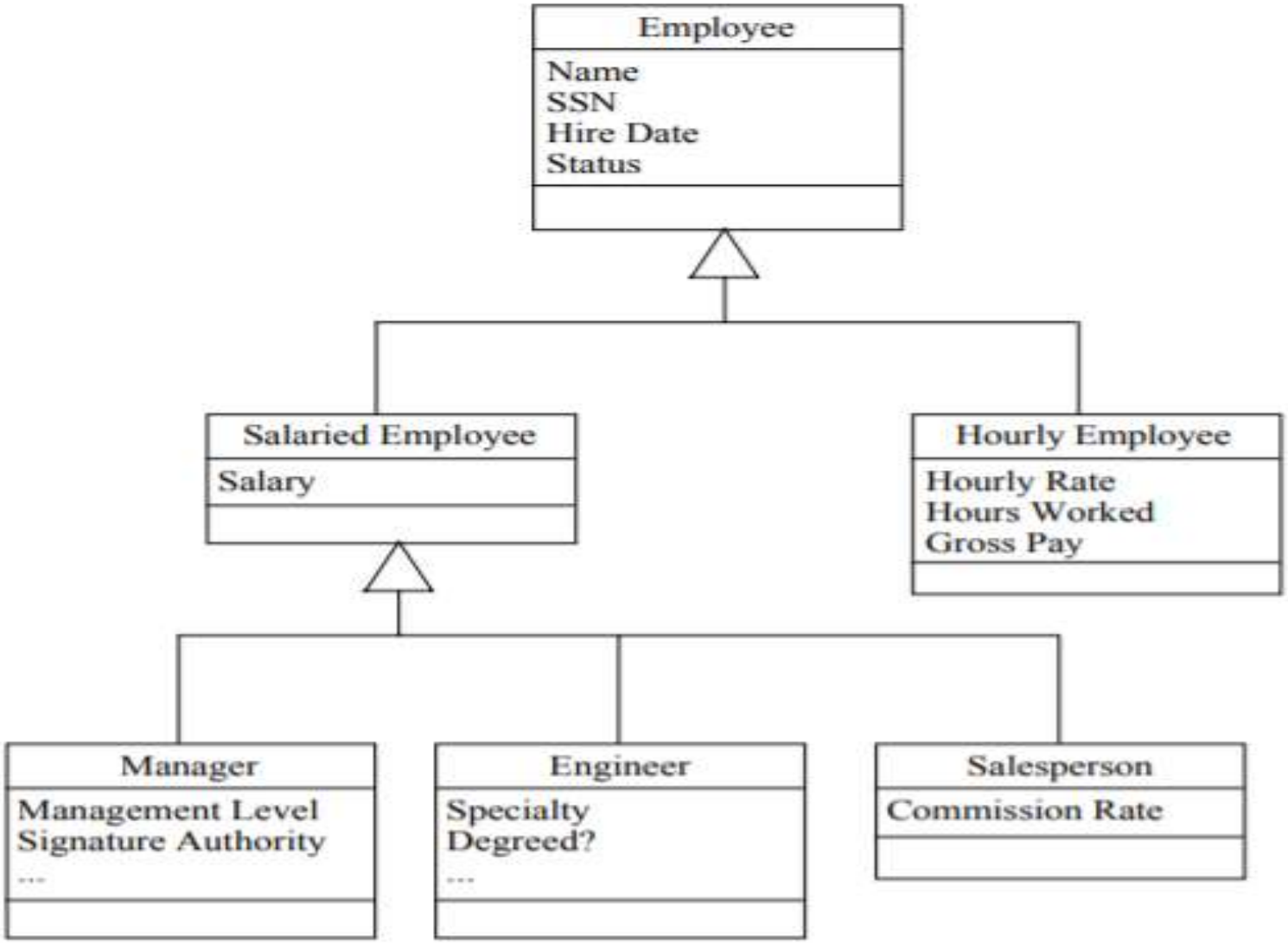
Generalization relationships

Generalization relationship is used to model inheritance

- Hierarchies drawn top-down
- Arrows point upward to parent
- Line/arrow styles indicate if parent is a(n):
  - **class**: solid line, black arrow
  - **abstract class**: solid line, white arrow
  - **interface**: dashed line, white arrow
- Often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



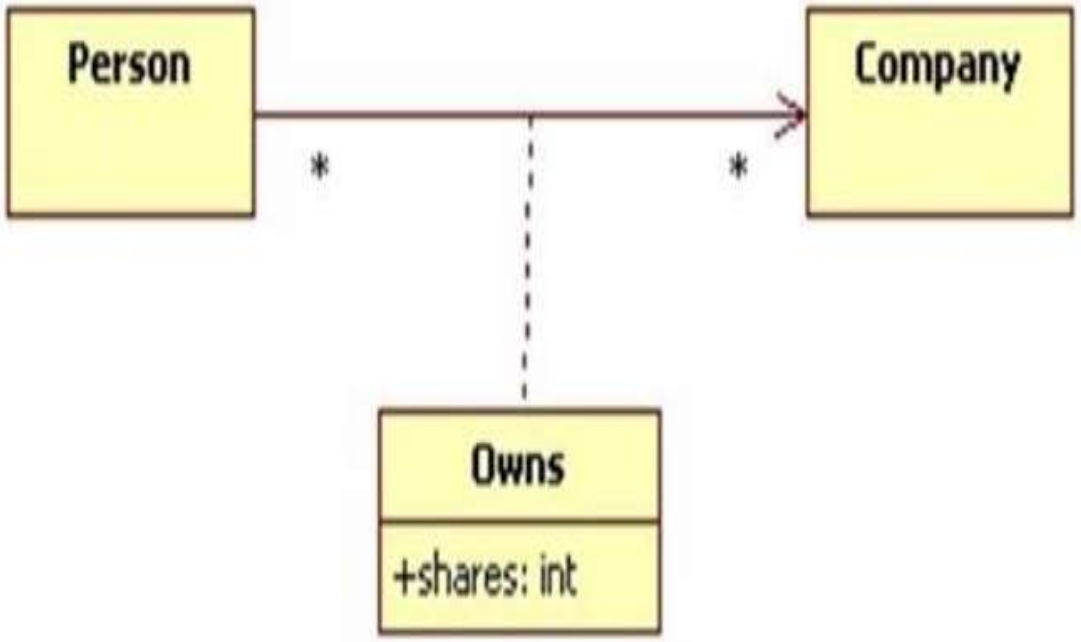

base class
Shape
origin
move()
resize()
display()()
generalization
Rectangle
corner : Point
Circle
radius : Float
Polygon
points : List
display
Square
leaf class

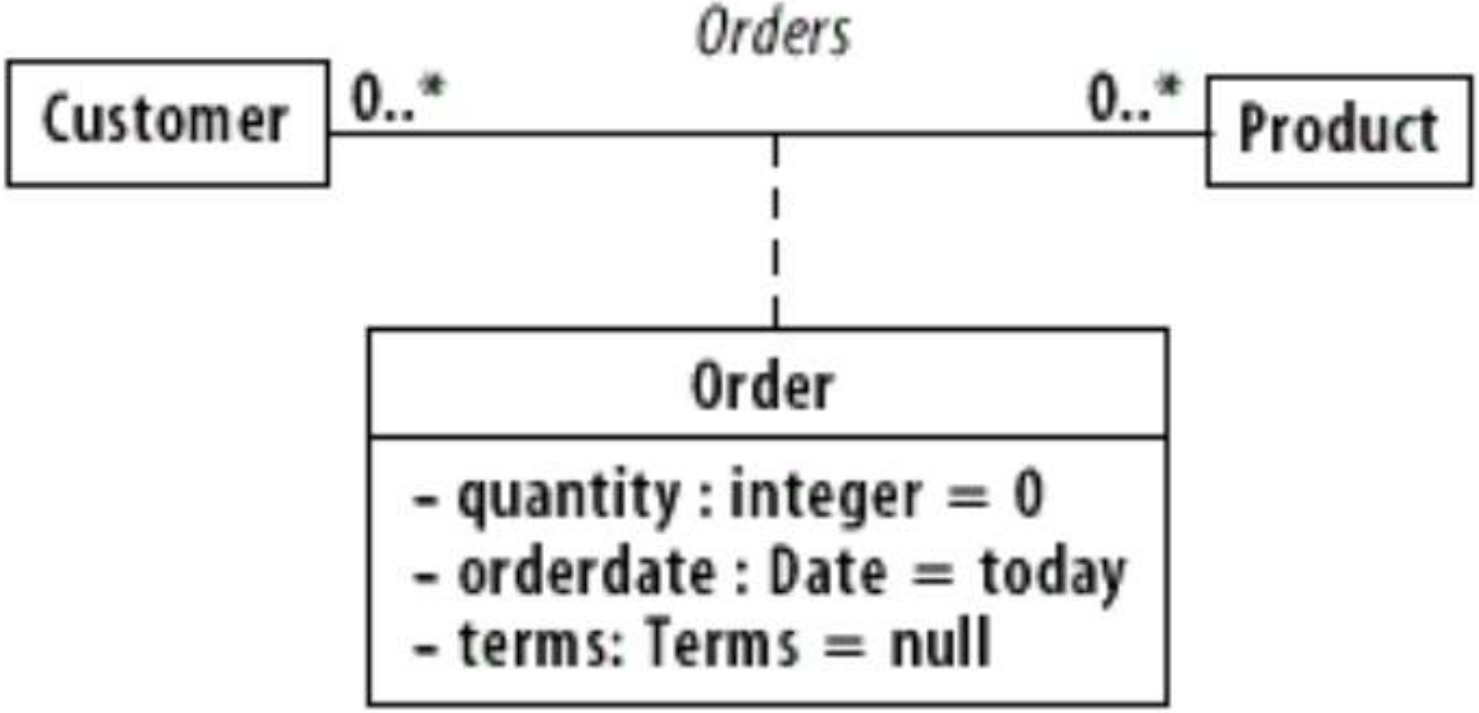

Employee
Name
SSN
Hire Date
Status
Salaried Employee
Salary
Hourly Employee
Hourly Rate
Hours Worked
Gross Pay
Manager
Management Level
Signature Authority
...
Engineer
Specialty
Degreed?
...
Salesperson
Commission Rate

Association Class

Consider the relationship "Person X owns N shares of Company Y". Where will N be stored? N is neither an attribute of Company nor Person. In cases like this we can represent links as objects. These link objects are instances of association classes:

An association class encapsulates information about an association.



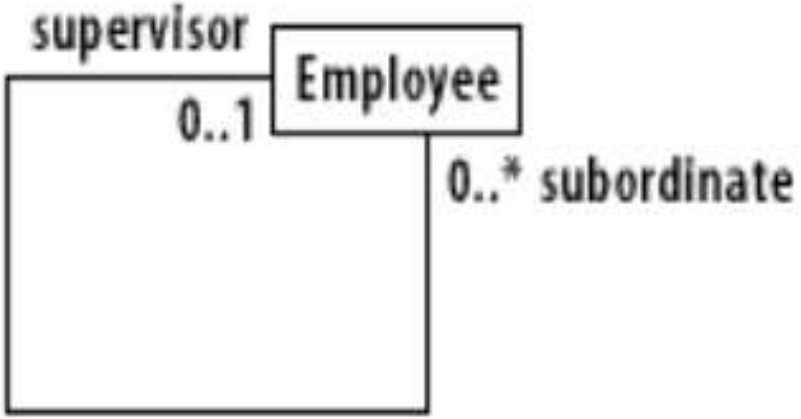


Association class

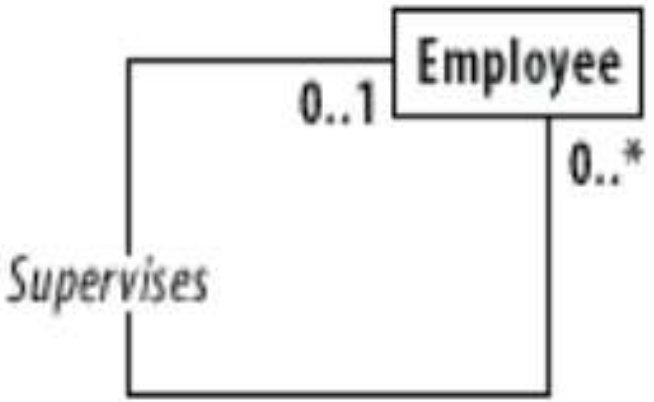
Reflexive Association

- This is a fancy expression that says objects in the same class can be related to one another

Using roles



Using association name



Reflexive association

END.

Wairagu G.R