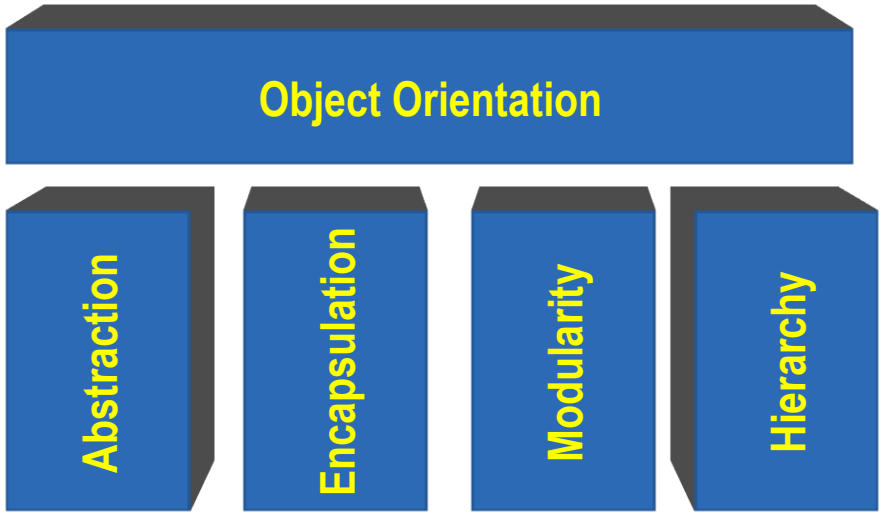
ENSF 593/594

7 – Class Relationships I

Elements of the Object Model

- There are four major elements of the object models:
  - Abstraction
  - Encapsulation
  - Hierarchy
  - Modularity



What is Abstraction?

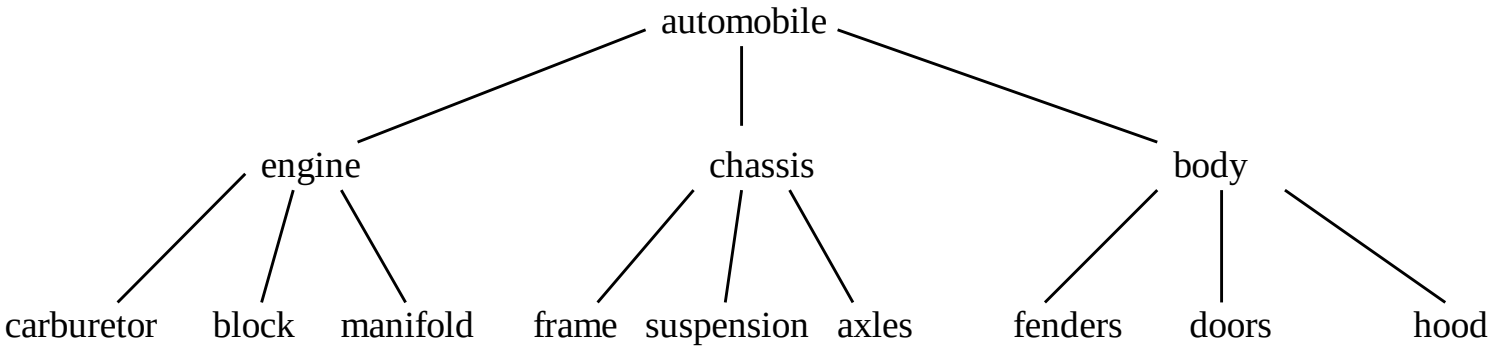
- Abstraction is a technique of dealing with complex system. We make a simplified model of a complex system.
  - By abstraction, we ignore the inessential details.
  - An abstraction focuses on the outside view of an object.
- “Deciding upon the right set of abstractions for a given domain is the central problem in object-oriented analysis and design.”
- In context of Object Oriented Programming, abstraction is represented by a “class” definition .

What is Encapsulation ?

- Encapsulation is achieved through information hiding:
  - The structure of the object is hidden.
  - Although the interface of the methods are public, but the implementation of the object's methods is hidden.
  - We can re-implement anything inside the object's capsule without affecting other objects that interact with it.

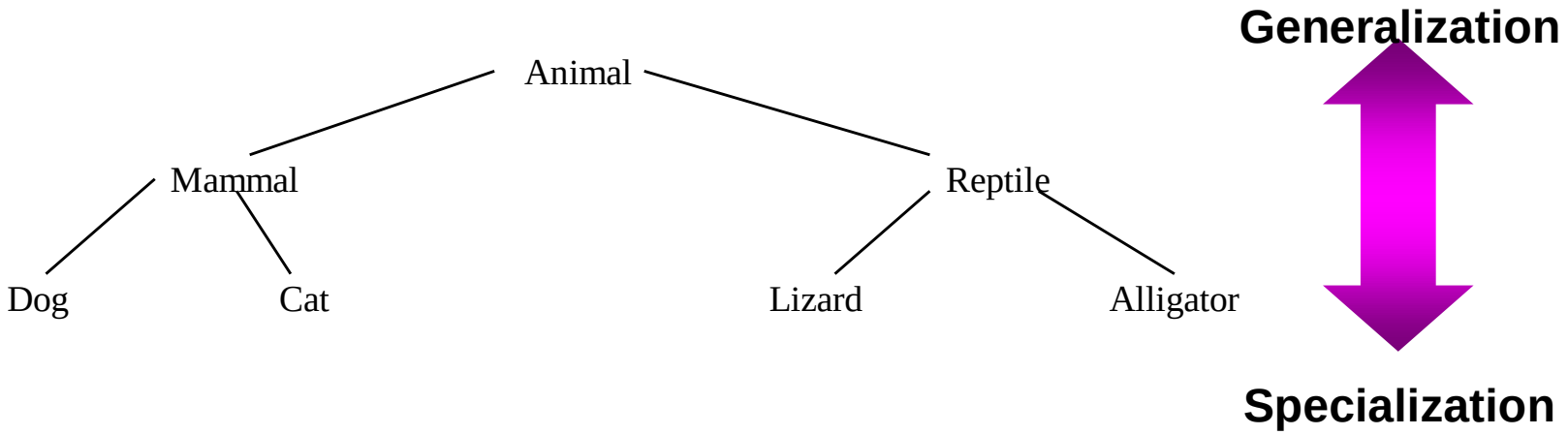
What is Hierarchy?

- Systems generally consist of at least two hierarchies:
  - "has a" hierarchy
    - Often called the *object structure*.



Hierarchy (continued)

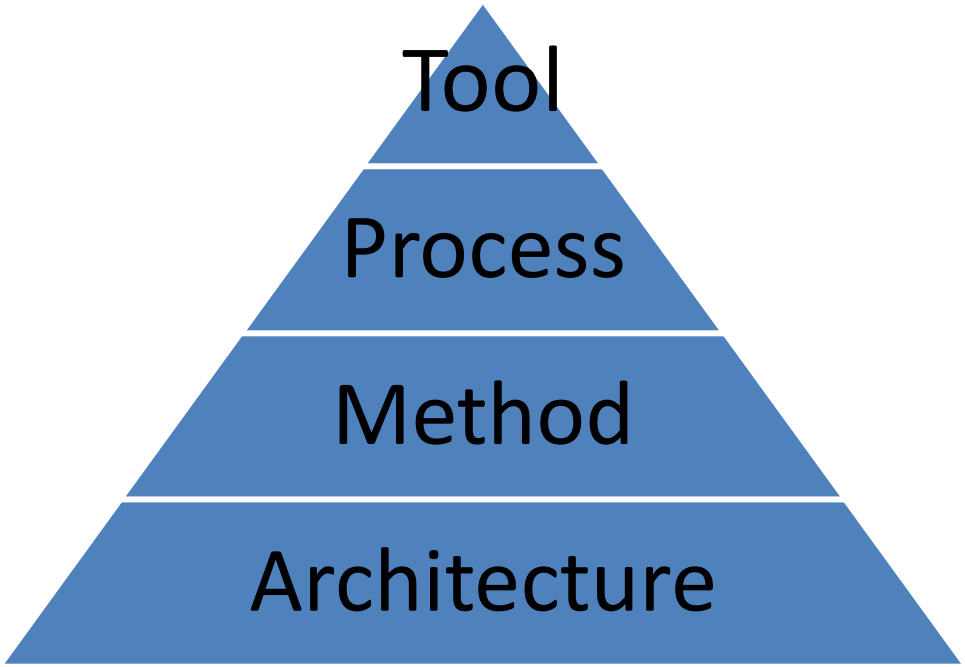
- "Is a" hierarchy
  - Objects are specialization of more general kinds of objects.



Software Design Process, and Modeling Tools

Elements of System Development

- To design the system's *architecture* we need to adopt a *method*, establish a *process* , and use an appropriate *tool*.



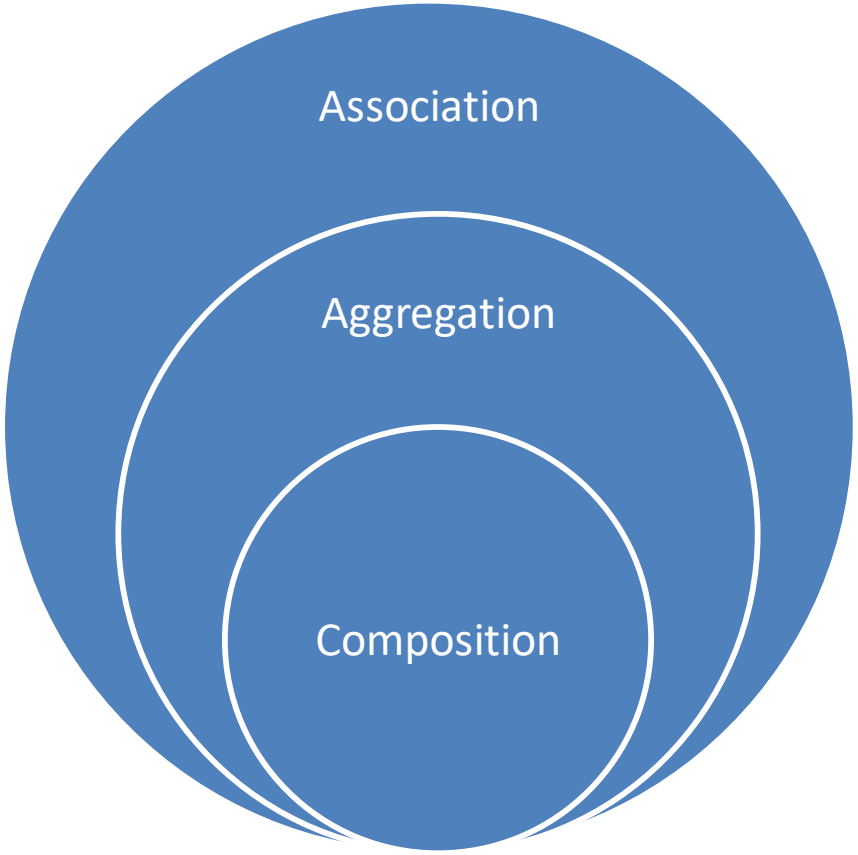
What is a Software Model

Relationship among classes

- There are three major type of relationships among classes that most of O.O. programming languages support them:
  - Association
  - Aggregation
  - Composition
  - Inheritance
  - Dependency

Association Relationship

Association Relationship Types

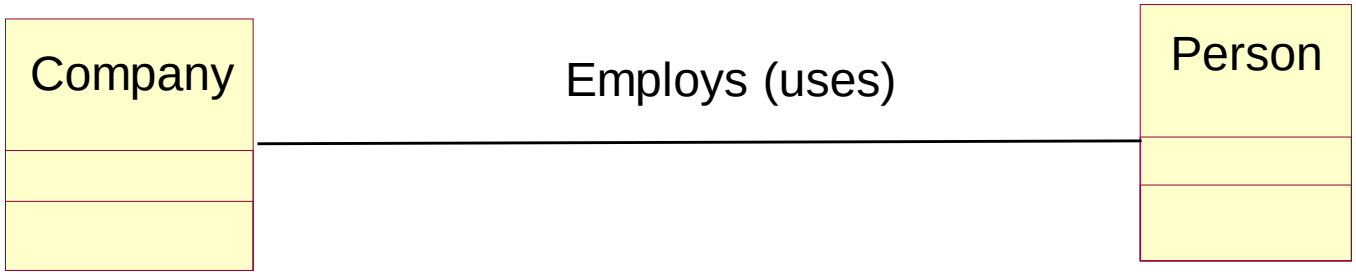


Association

- The association relationship expresses a semantic connection between classes:
  - There is no hierarchy.
- It is a relationship where two classes are weakly connected; i.e. they are merely "associates."
  - All object have their own lifecycle;
  - There is no ownership.
  - There is no whole-part relationship.

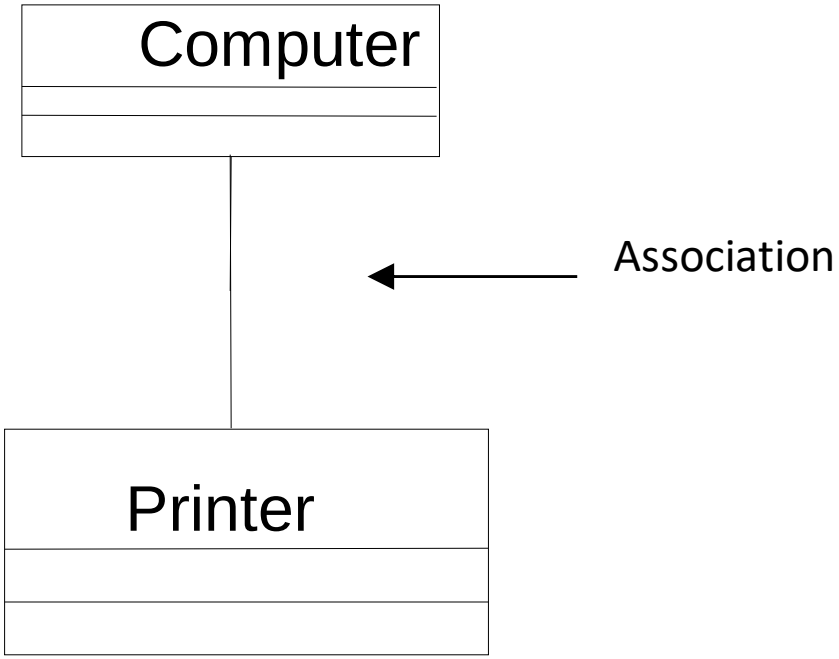
Association Relationship

- Normally an association relationship represents a bi-directional relationship between classes of objects.
- A typical type of association is the “using” relationship



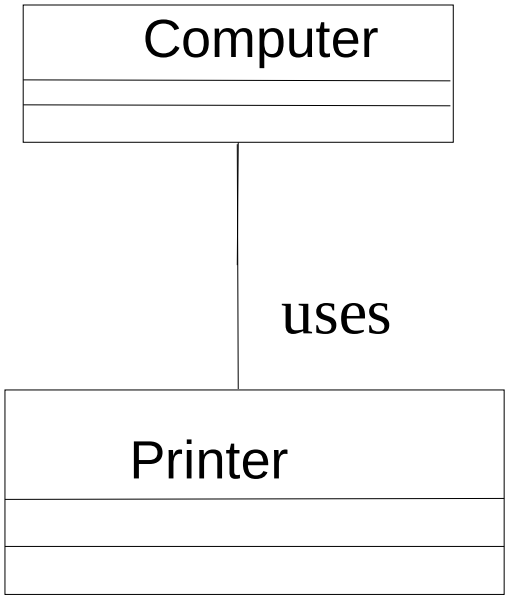
Association

- Another Example:
  - Computer and Printer. Multiple Printers can associate with single Computer and single Printer can associate with multiple Computers.



Association (continued)

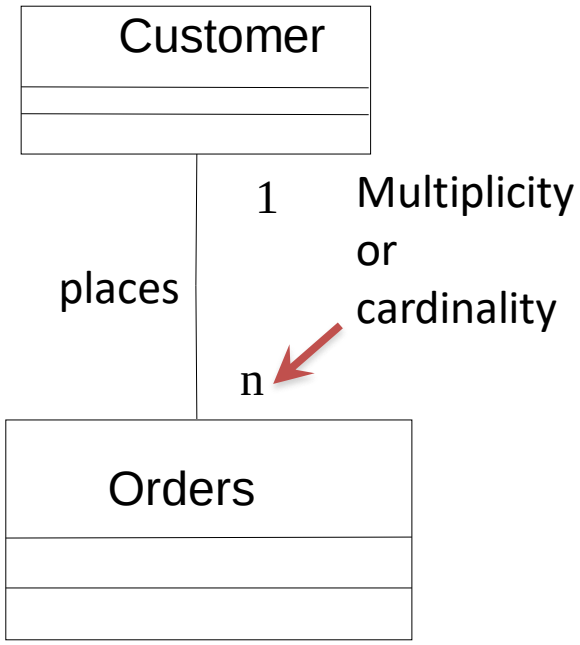
- The association of two classes must be labeled.
- To improve the readability of diagrams, associations may be labeled in active or passive voice.



Multiplicity/Cardinality

- Multiplicity defines the number of objects that are linked to one another.

1	Exactly one
0..*	Zero or more
1..*	One or more
0..1	Zero or one
5..7	Specific range (5, 6, or 7)
4..7, 9	Combination (4, 5, 6, 7 or 9)



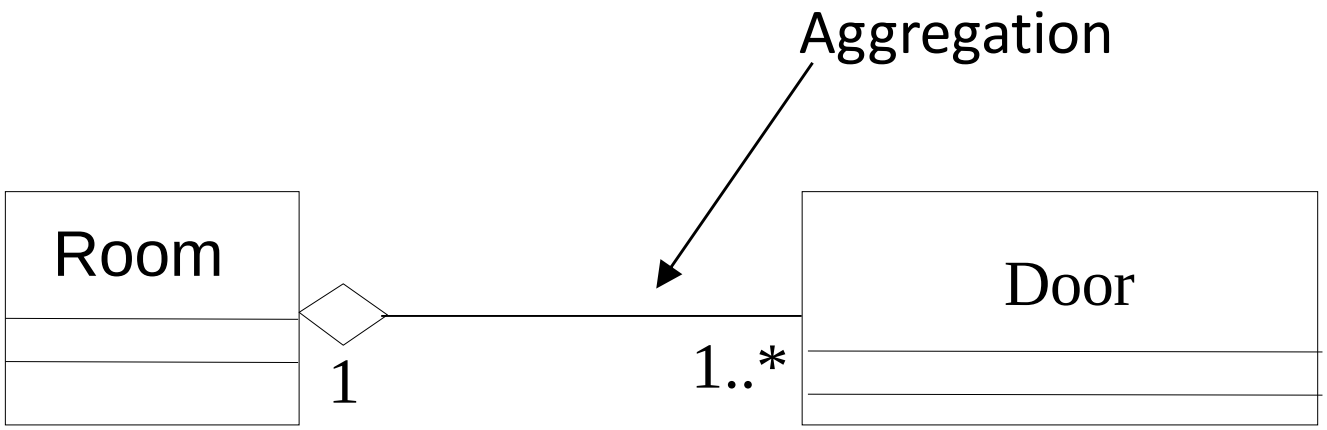
Aggregation

- An aggregation represents an asymmetric association, in which one of the ends plays a more important role than the other one.
- The following criteria imply an aggregation:
  - A class is part of another
  - The objects of one class are subordinates of the objects of another class
- Denotes a whole/part hierarchy with the ability to navigate from whole (aggregate) to its parts (attributes).

Aggregation

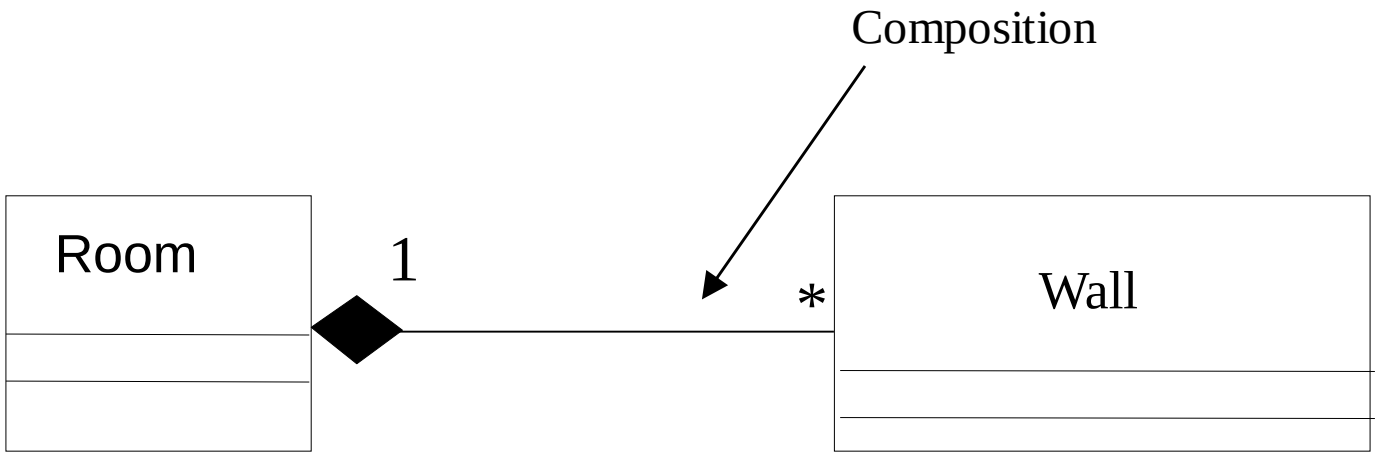
- Aggregation is a specialize form of Association where all object have their own lifecycle but:
  - Either there is ownership, or whole-part relationship.
    - The part object can not belongs to another parent object.
  - Example:
  - Department and teacher.
    - A single teacher can not belongs to multiple departments.
    - If we delete the department teacher object will not destroy. We can think about “has-a” relationship.

Class Relationships - Aggregation



Aggregation (continued)

- A strong type of aggregation, when deletion of the whole causes the deletion of the part, is called *composition.*



Aggregation in Java

- Any class in Java can contain references to other classes.
- The “part” objects are usually instantiated in the constructor of the container class.

Composition

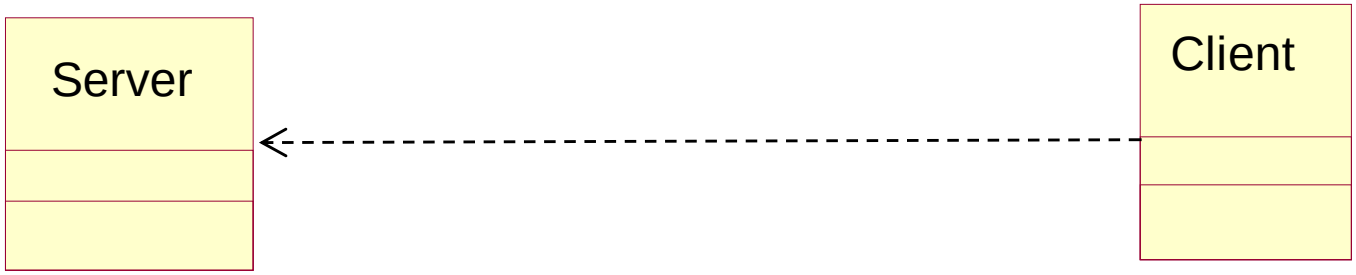
- Composition is again specialization of Aggregation.
  - It is a strong type of Aggregation.
  - The lifecycle of the part depends on part depends on the lifecycle of the owner.
  - Example:
    - Relationship between a House and Room. If one destroys the house the rooms will be destroyed automatically.

More Details on Association

What is Dependency?

- Dependency is again an indicator of some type of links between two software elements.
  - Can be between two class
  - Can be between two packages
  - Can be between two module
  - Etc.
- A typical type of dependency between two class is type of client and service relationship. Where one class is providing services to another class.
  - Most of the time method of one class uses the object (s) of the other class is its input or as its local variable.
  - It is normally not a structural link
  - It is a temporary link when needed

What is Dependency?

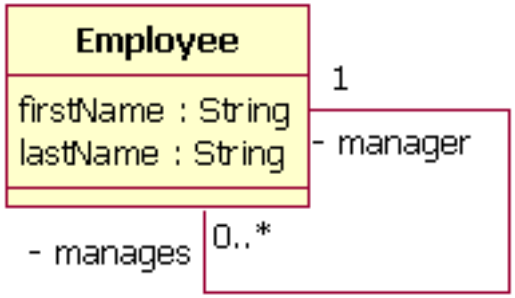


Association Role

- The association end name is commonly referred to as **role** name.
- The role name is optional and suppressible.

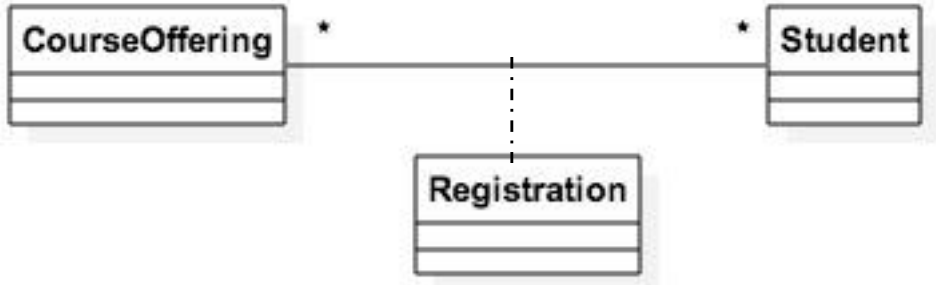
Reflexive Association

- A class can also be associated with itself, using a reflexive association.

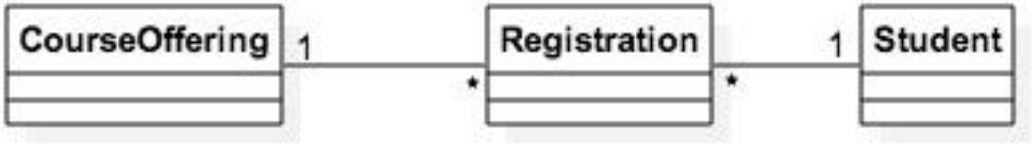


Association Class

- There are times when you need to include another class because it includes valuable information about the relationship.
- During analysis stage of software development, this type of association can be shown like a normal class with a dotted association-line between the primary classes.



- During design stage of software development, this type of association can be shown like a normal class with solid association-line between the primary classes.

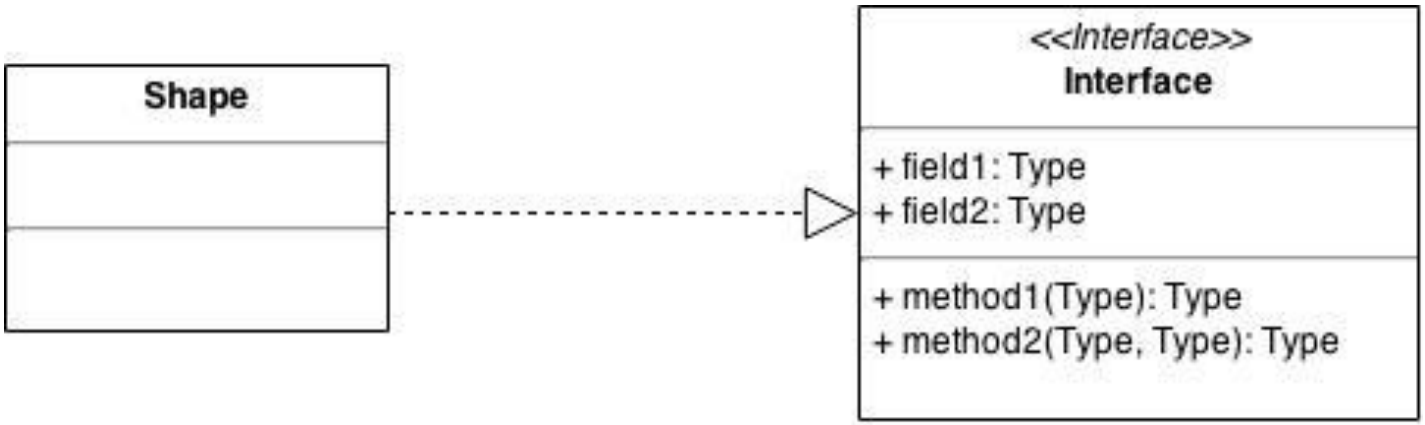


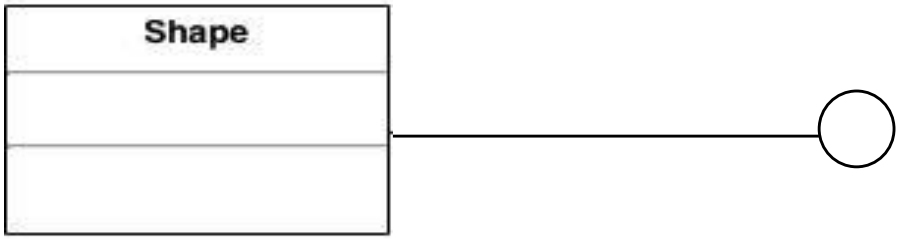
Interface Realization

- An **interface** is a classifier that declares a set of coherent public features and obligations.
  - Specifies a **contract**.
  - Any instance of a class that **realizes** (implements) the interface must fulfill that contract and thus provides **services** described by contract.
  - Will be discussed in more details later.

Interface Realization

- Two options to show Interface Realization





Implementation of Different Type of Associations among Classes in Java

Class Exercises

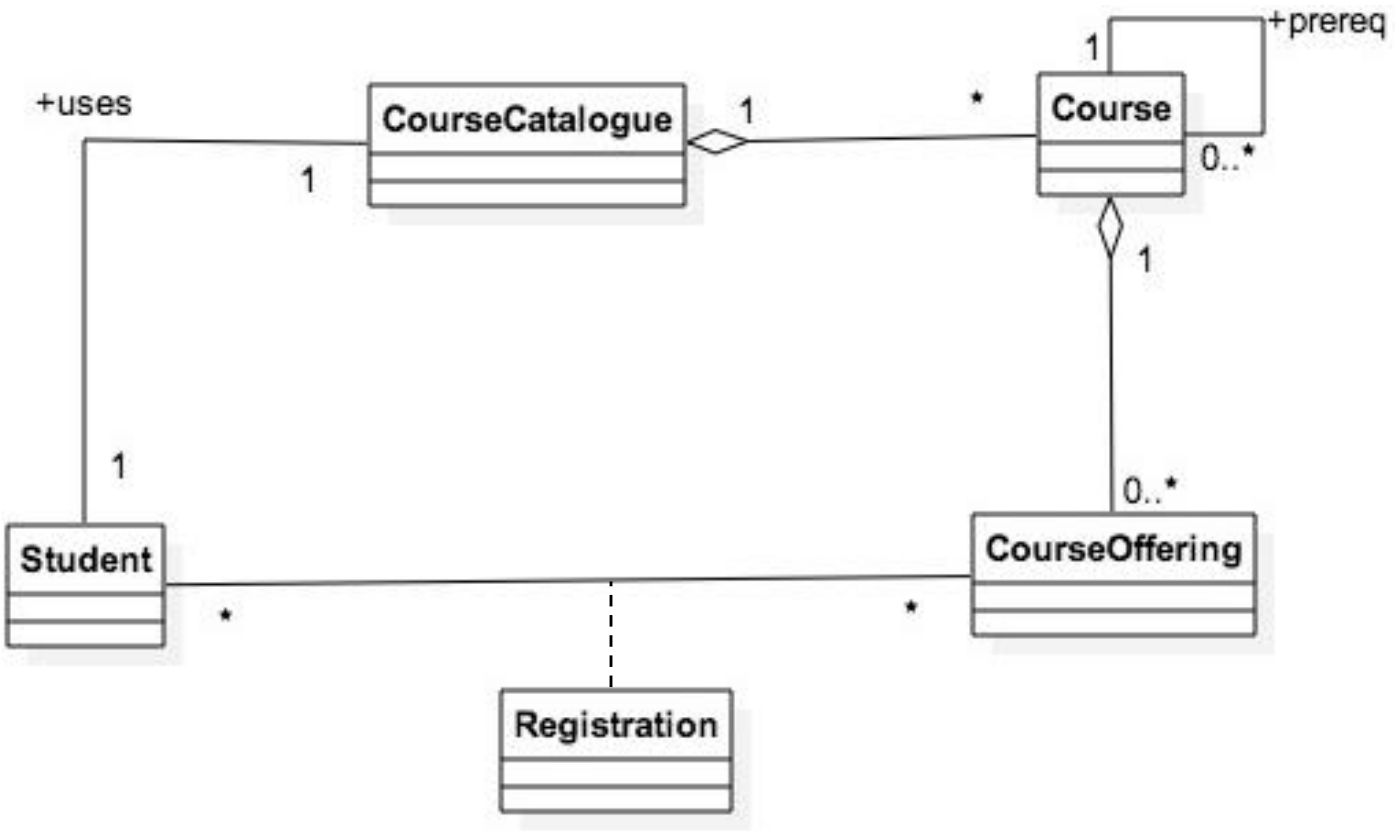
A Simple Example

- As part of a Traffic Application assume:
  - Each vehicle is supposed to be considered to have four wheels, and an engine.
  - A vehicle can to be driven by one person.

A More Complex Exercise

- Course Registration System:
  - Student are supposed to be able to select their courses from course-catalogue and register for maximum 6 courses.
  - Each course, maybe offered in 1 or more sections (offerings), must have a minimum of 8 students to run, and may have one or more prerequisite.

Analysis Class Diagram



Design Class Diagram