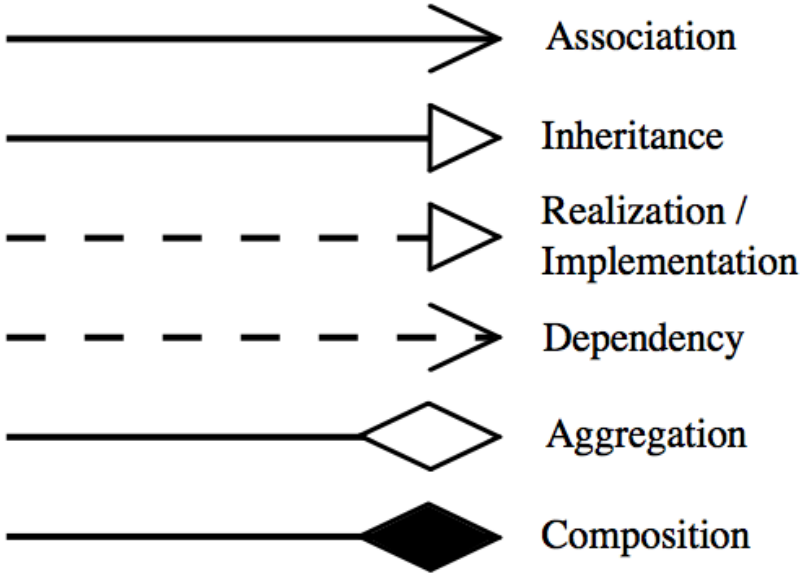
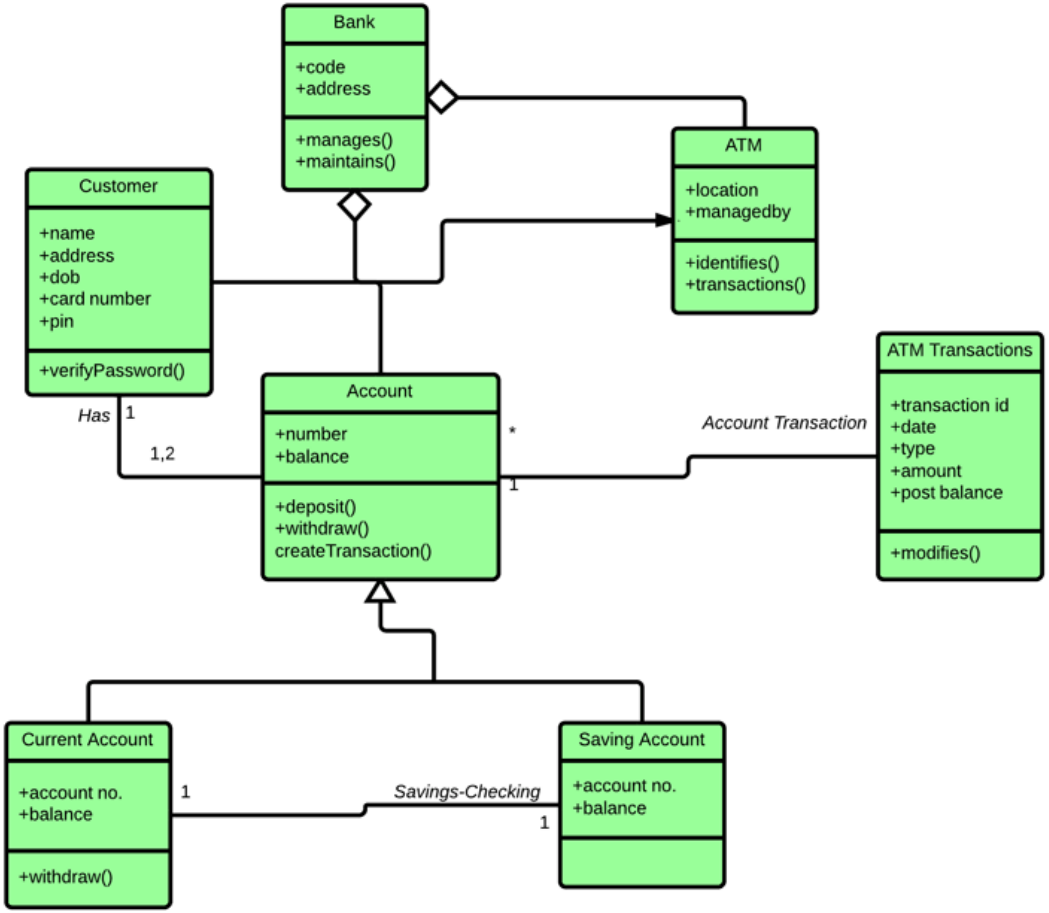
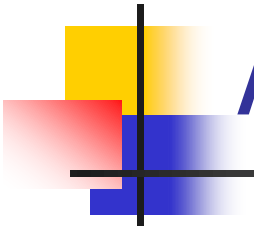
04. OO Relationships



OO Relationship

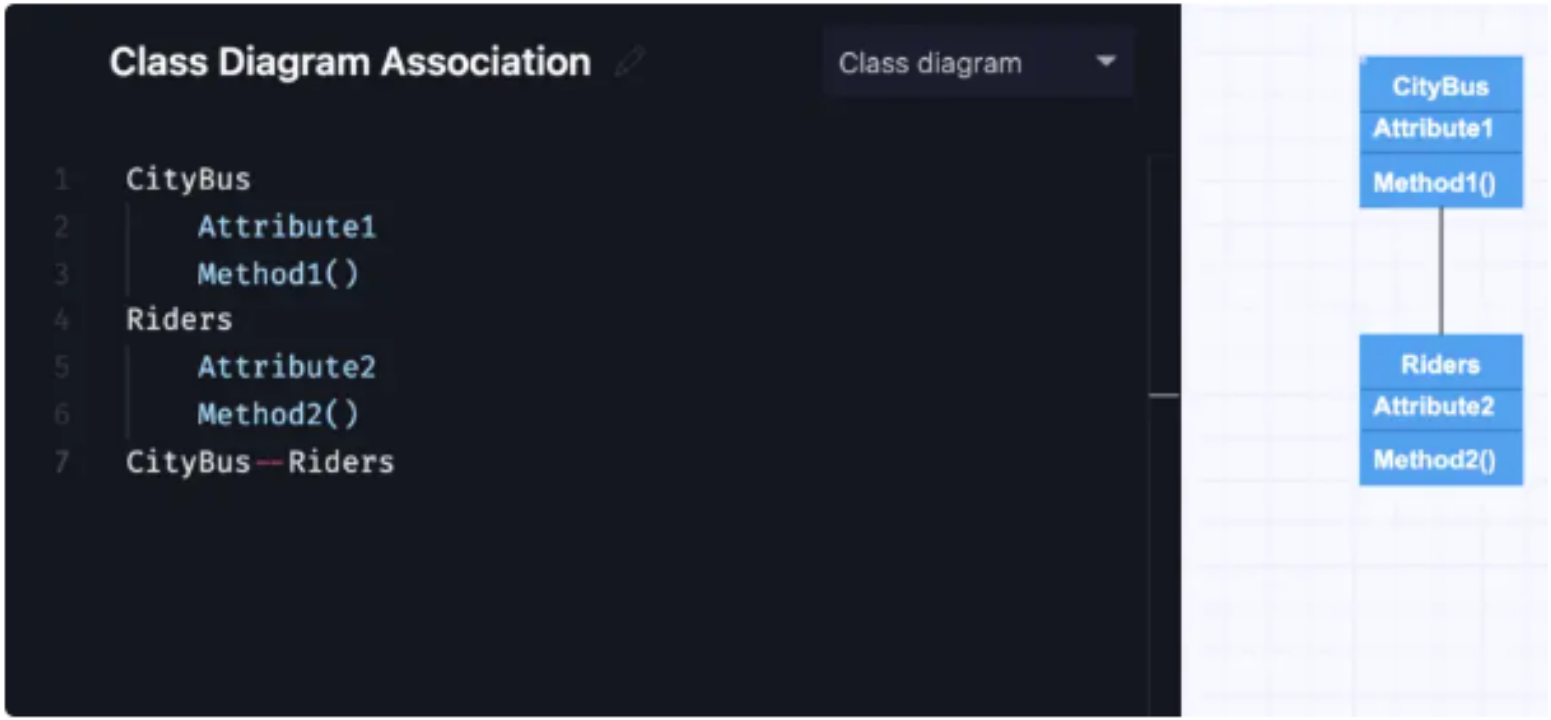




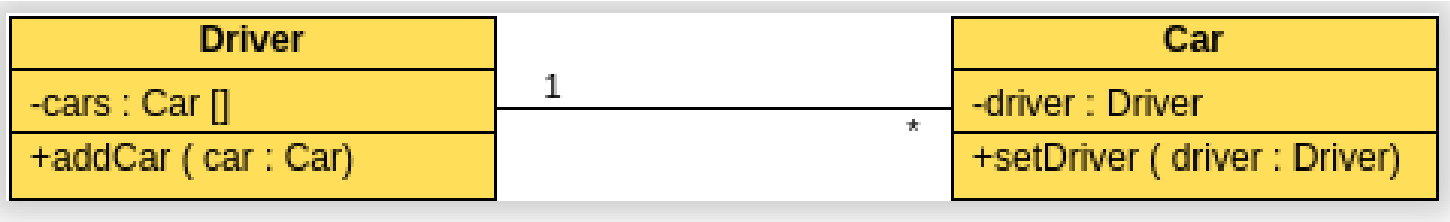
Association

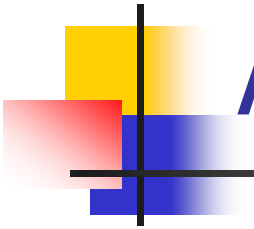
- This simply means that one model element is linked in some way to another model element. The association indicates the nature and rules that govern the relationship. The basic way to represent association is with a line between the elements.

Association



Association





Association

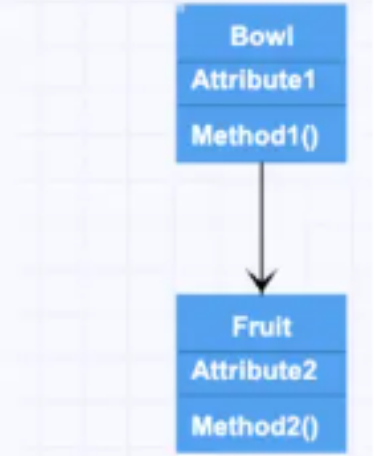
- Association can be more complex, in that it can be directed, which is represented by an arrow showing the flow of control, or even reflexive, in cases where the element has a relationship to itself. In this case, the arrow loops back to the element.

Association

Class Diagram Directed Association

Class diagram

```
Bowl
    Attribute1
    Method1()
Fruit
    Attribute2
    Method2()
Bowl --> Fruit
```



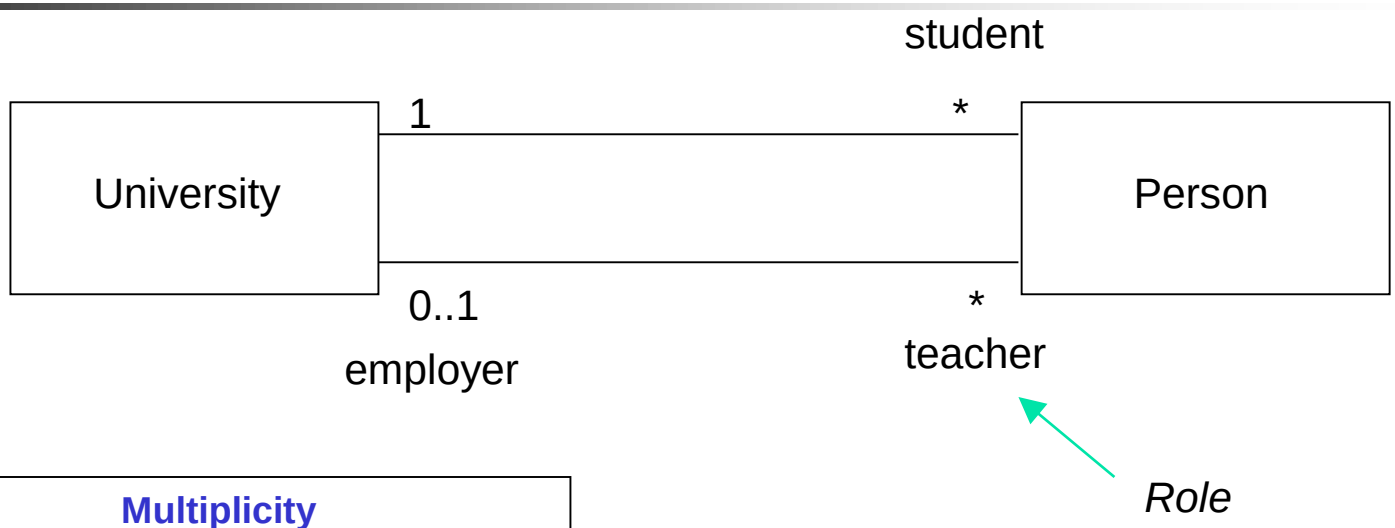
Association (Multiplicity)

- An association relationship between elements can also have cardinality, for instance, one-to-one, one-to-many, many-to-one, or many-to-many, zero-to-many, and so on. This can also be shown in a label on the line..

Association


```
class Child {
    Mother mother;
}

class Mother {
    List<Child> children;
}
```

Association: Multiplicity and Roles



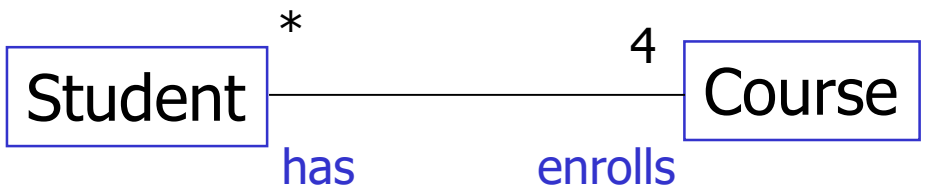
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

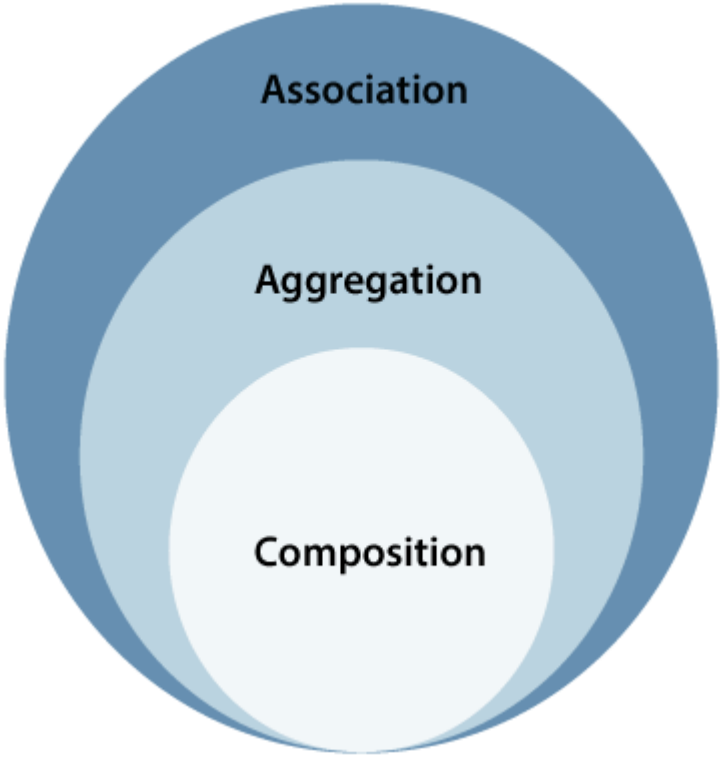
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

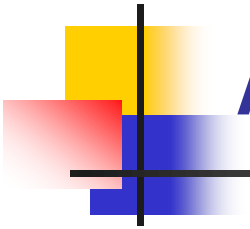
Association: Model to Implementation




```
Class Student {
    Course enrolls[4];
}

Class Course {
    Student have[];
}
```

Association



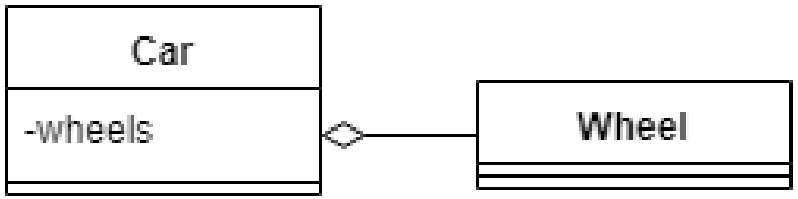


Aggregation

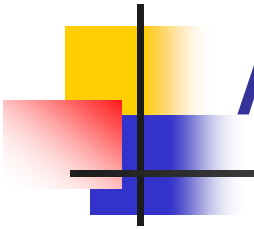
- This type of association relationship indicates an element is formed by a collection of other elements. For instance, a company has departments or a library has books.

- The aggregate element relies on other elements as parts, but those other elements can also exist independently of it.

Aggregation



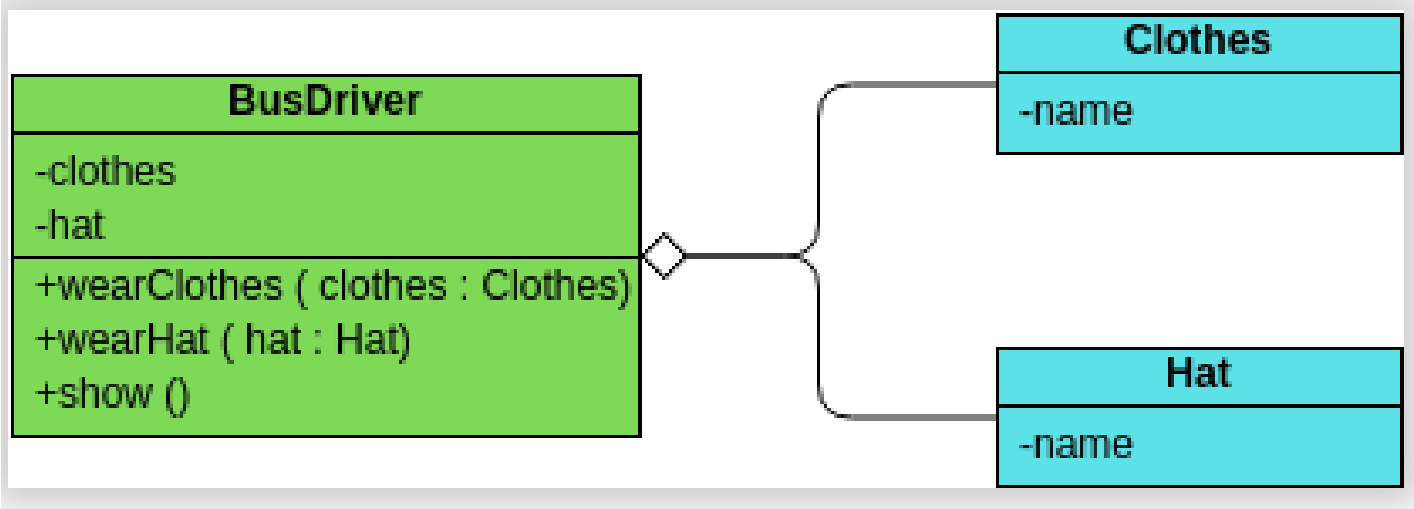

```
class Wheel {
    Car car;
}

class Car {
    List<Wheel> wheels;
}
```



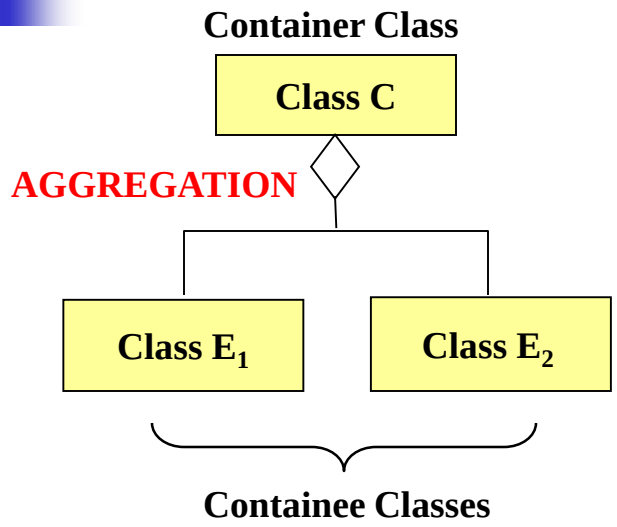
Aggregation

- An aggregation is represented by a line from one class to another, with an unfilled diamond shape near the aggregate, or the element that represents the class that is assembled by combining the part elements.

Aggregation

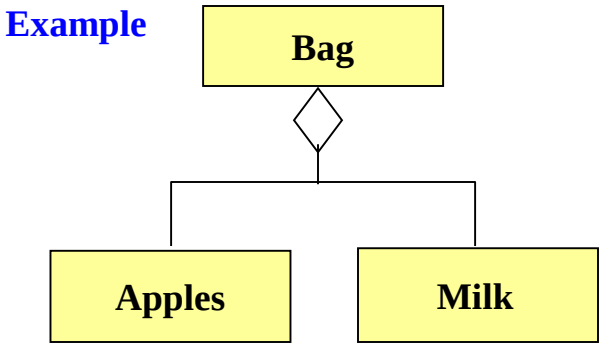


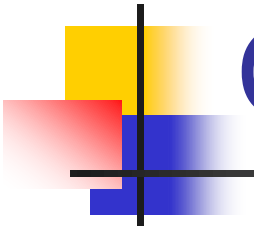
Aggregation



Aggregation:
expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

express a more informal relationship than composition expresses.

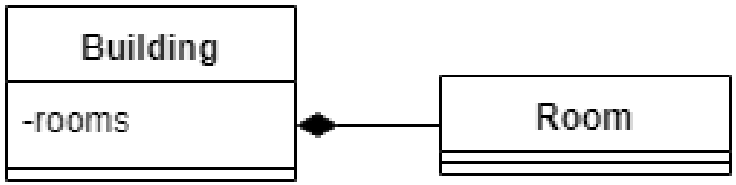




Compostion

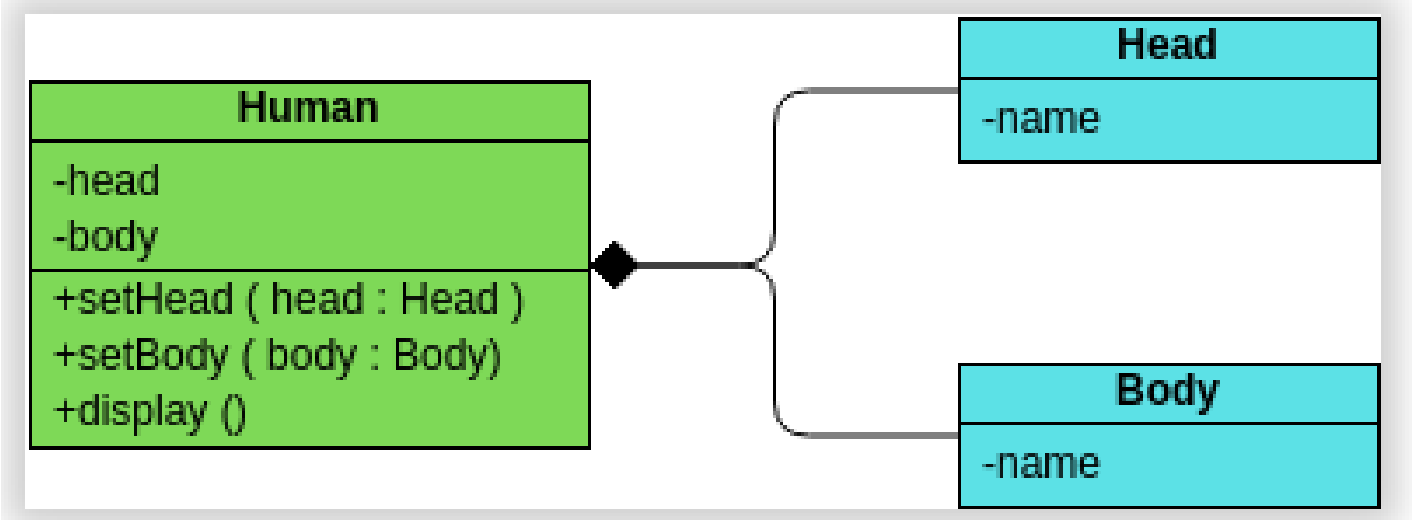
- Another type of aggregation relationship, composition, is one in which the part elements cannot exist without the aggregate. For instance, the rooms in a house cannot continue to exist if the house is destroyed.
- For a composition relationship, a filled diamond is shown on the line near the aggregate.

Composition

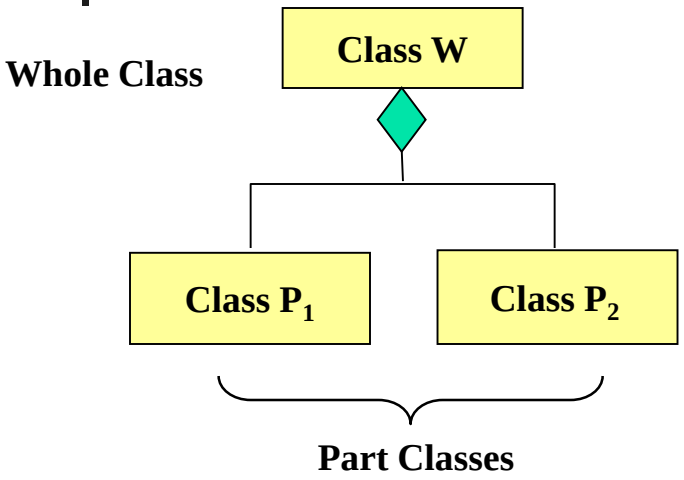



```
class Building {
    String address;

    class Room {
        String getBuildingAddress() {
            return Building.this.address;
        }
    }
}
```

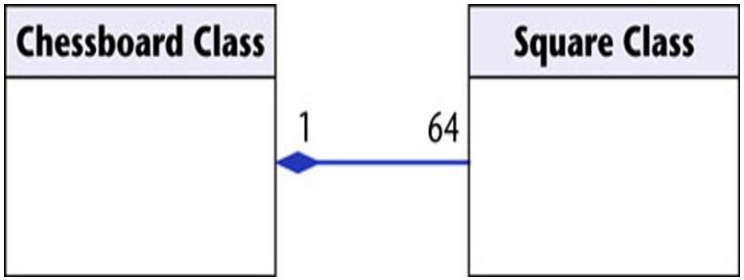
Composition



Composition



Example



Association

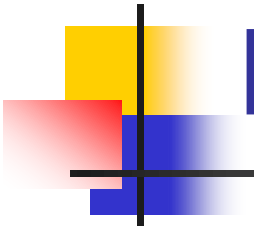
Models the part–whole relationship

Composition

Also models the part–whole relationship but, in addition, Every part may belong to only one whole, and If the whole is deleted, so are the parts

Example:

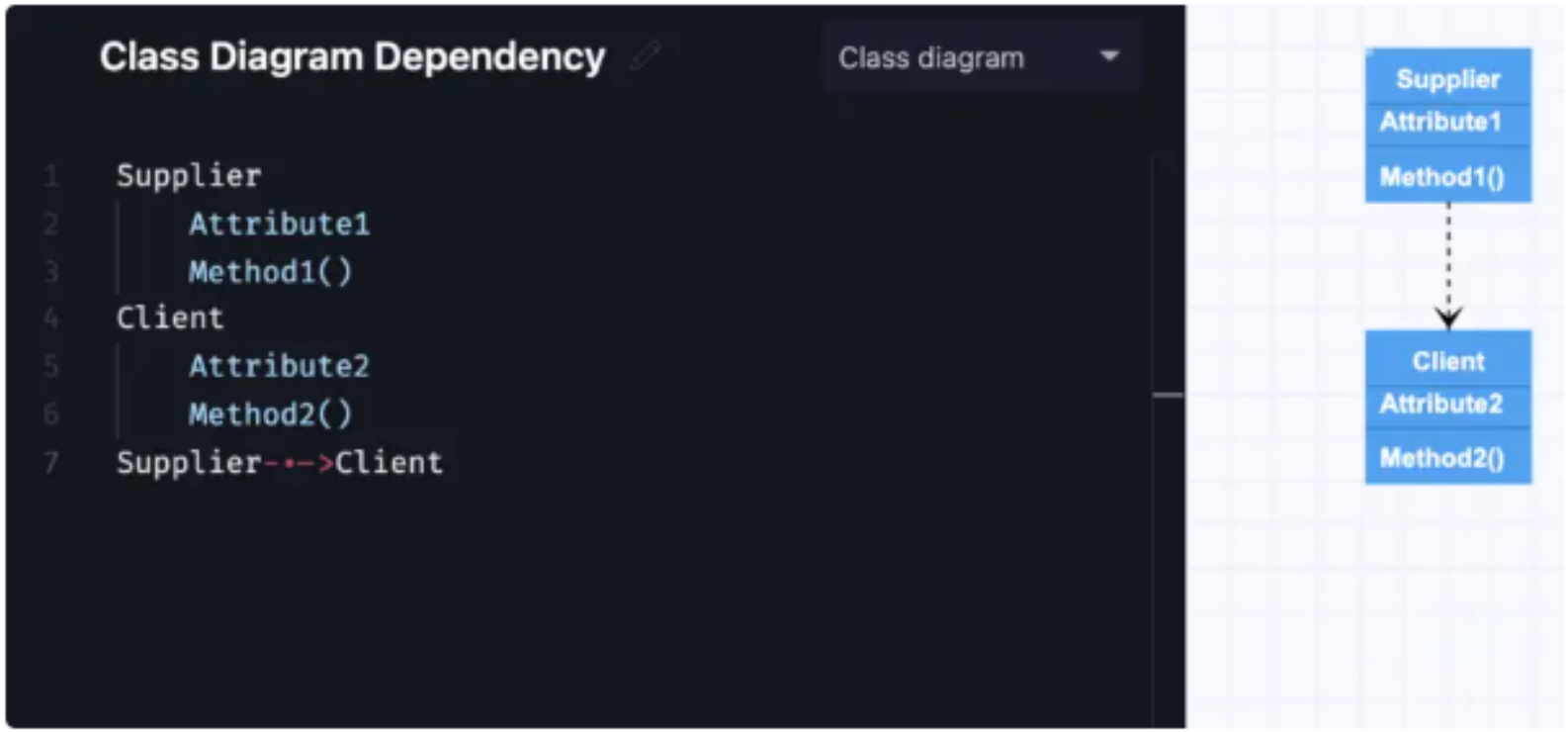
A number of different chess boards: Each square belongs to only one board. If a chess board is thrown away, all 64 squares on that board go as well.



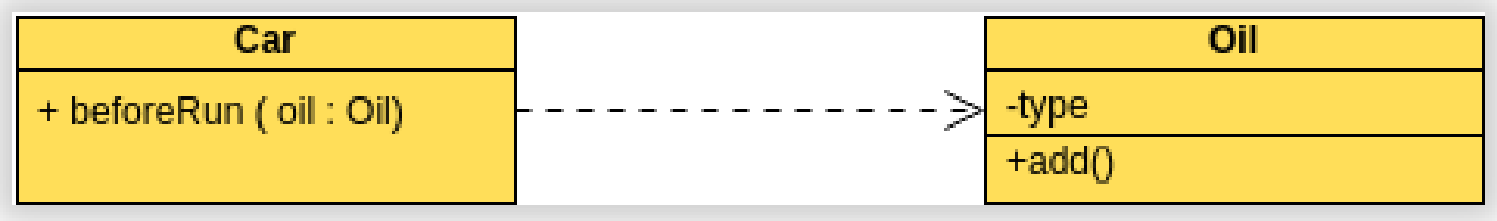
Dependency

- Dependencies in UML indicate that a source element, also called the client, and target element, also called the supplier, are related so that the source element makes use of, or depends upon, the target element.
- Changes in the behavior or structure of the target may mean changes in the source.

Dependency

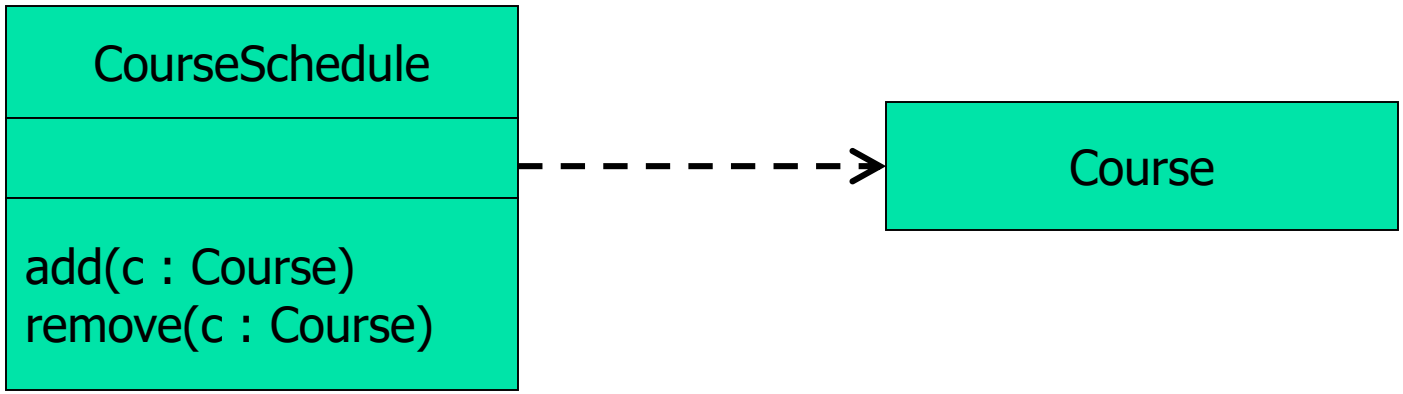

Class Diagram Dependency
Class diagram
1 Supplier
2 Attribute1
3 Method1()
4 Client
5 Attribute2
6 Method2()
7 Supplier--•-->Client
Supplier
Attribute1
Method1()
Client
Attribute2
Method2()

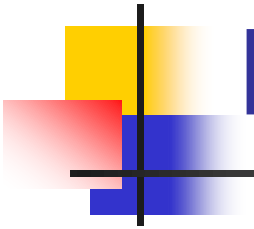
Dependency



Dependency

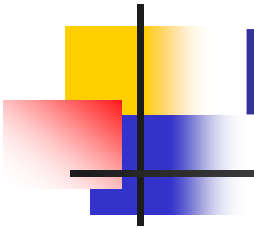
A *dependency* indicates a semantic relationship between two or more elements. The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.





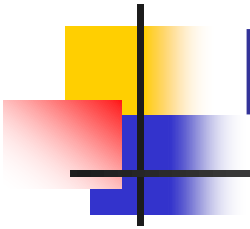
Example

- Let us model a university, which has its departments. Professors work in each department, who also has friends among each other.



Example

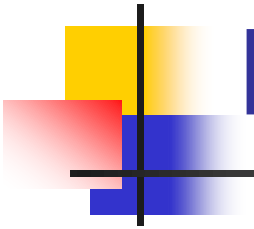
- Will the departments exist after we close the university?
- Of course not, therefore it's a **composition**.



Example

- But the professors will still exist (hopefully). We have to decide which is more logical: if we consider professors as parts of the departments or not. Alternatively: are they members of the departments or not?

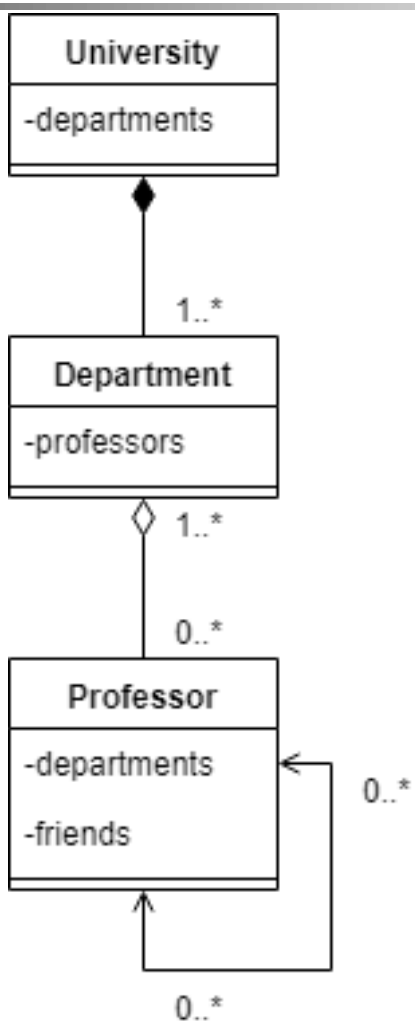
- Yes, they are. Hence it's an aggregation. On top of that, a professor can work in multiple departments.



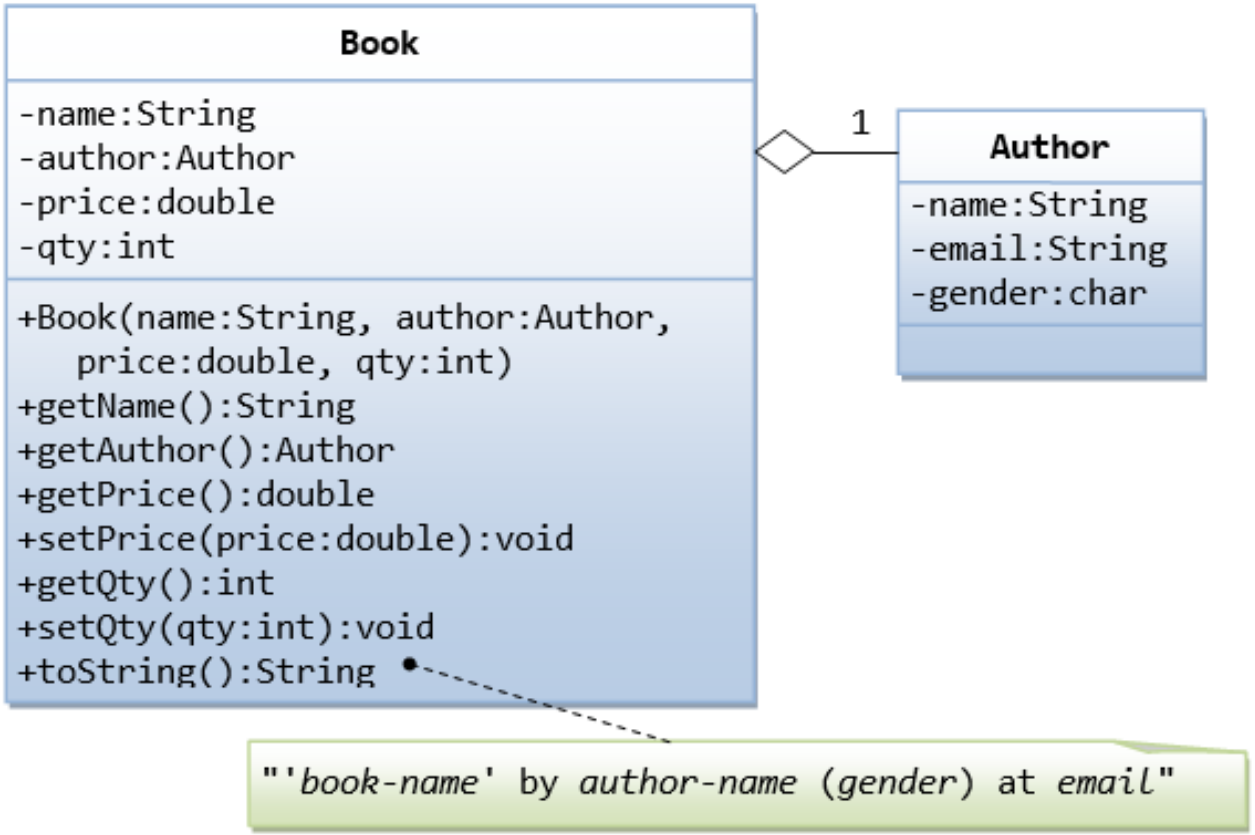
Example

- Finally, the relationship between professors is association because it doesn't make any sense to say that a professor is part of another one.

Example


University
-departments
1..*
Department
-professors
1..*
0..*
Professor
-departments
-friends
0..*
0..*

Exercise


Book
-name:String
-author:Author
-price:double
-qty:int
+Book(name:String, author:Author,
price:double, qty:int)
+getName():String
+getAuthor():Author
+getPrice():double
+setPrice(price:double):void
+getQty():int
+setQty(qty:int):void
+toString():String
1
Author
-name:String
-email:String
-gender:char
"'book-name' by author-name (gender) at email"