Software Engineering (CSE 327)

Lecture 8

(Class Diagram)

Class Diagram

- A diagram that shows the **building blocks** of a system
  - Can be at different perspectives
  - Already encountered the Domain model made up of conceptual classes (with attributes but no responsibilities)
- It shows the "classes" that make up the system
  - Can be seen as Entities or Things in the system
- A class diagram captures the classes that make up the system along with potential collaboration among these classes

Class Diagram

- **A class diagram captures,**
  - *Types* of objects in the system ← "*Class*"
  - Attributes and behaviours of classes
- **Relationships between Classes**
  - Generalisation (Subtypes)
    - E.g. A programmer *is a* <u>*kind of*</u> human
  - Associations
    - E.g. A customer may <u>*rent*</u> a number of videos
  - Other
    - E.g. dependency

What is a Class Diagram?

- A class diagram depicts classes and their interrelationships
- Used for describing structure and behavior in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

Why do we need class diagrams?

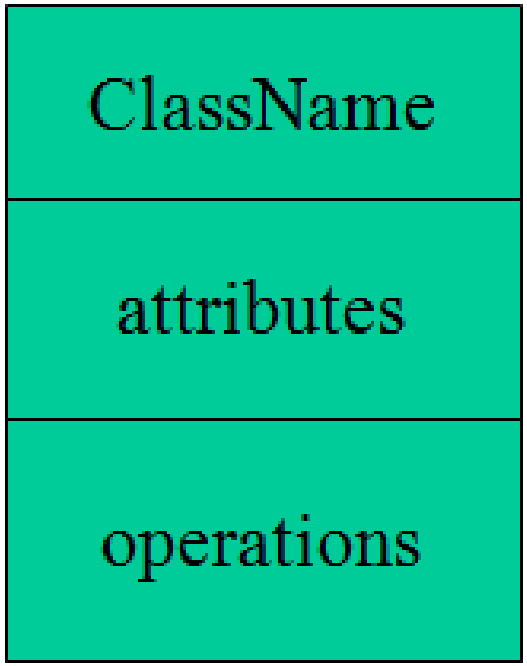
- Planning and modeling ahead of time make programming much easier.
- Besides that, making changes to class diagrams is easy, whereas coding different functionality after the fact is kind of annoying.
- When someone wants to build a house, they don't just grab a hammer and get to work. They need to have a blueprint — a design plan — so they can ANALYZE & modify their system.
- You don't need much technical/language-specific knowledge to understand it.

Essential Elements of a UML Class Diagram

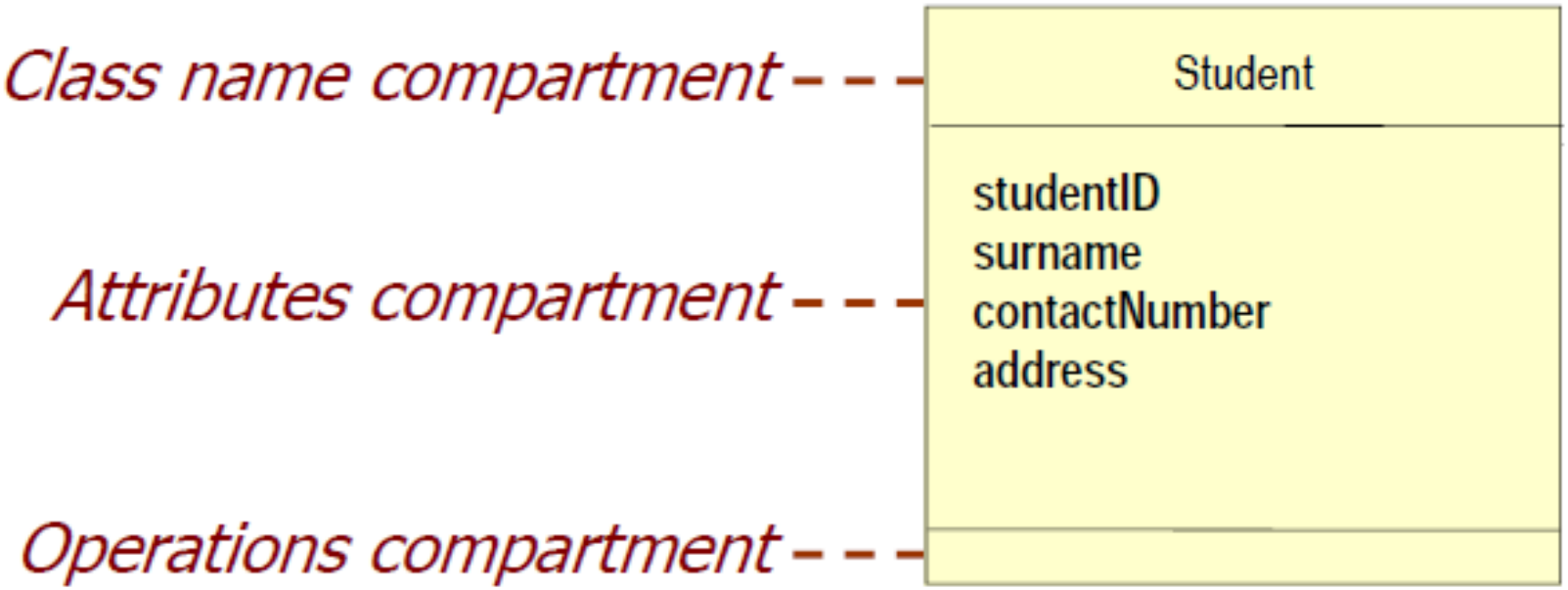
- Class
- Attributes
- Operations
- Relationships
  - Associations
  - Generalization
  - Realization
  - Dependency
- Constraint Rules and Notes

Class

- Describes a set of objects having similar:
  - Attributes (status)
  - Operations (behavior)
  - Relationships with other classes
- Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

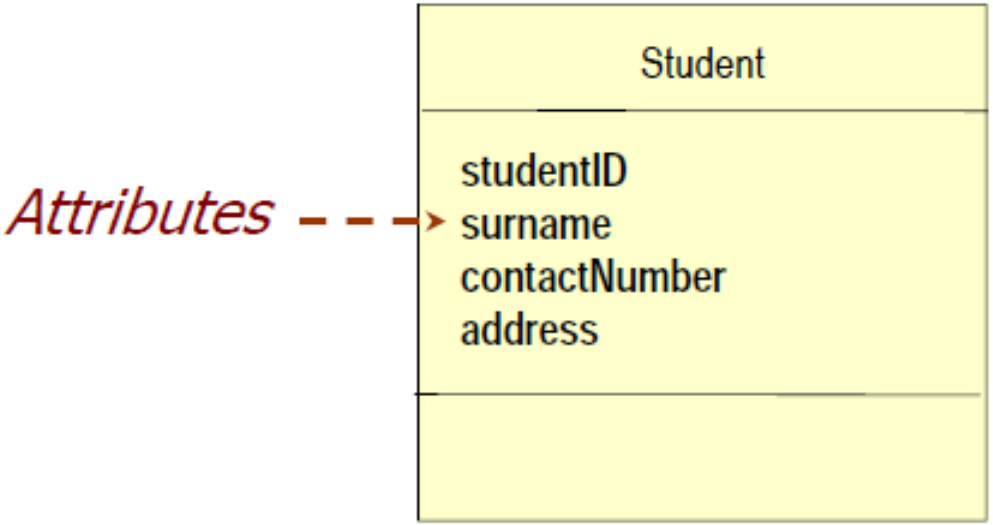


Class diagram : Class Symbol



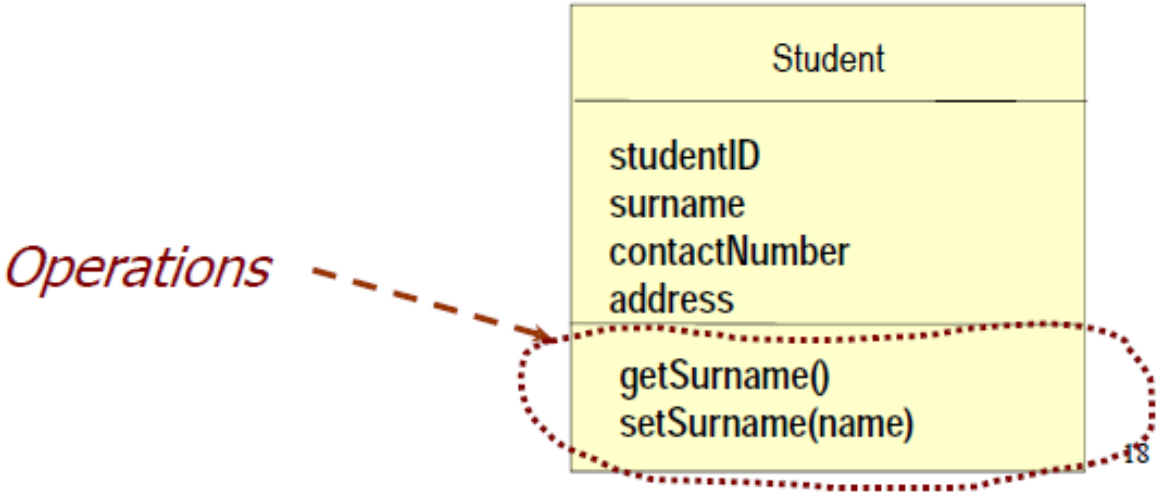
Class: Attributes and State

- **Attributes are**:
  - Essential to the business description of a class
  - Each object has its own *value* for each attribute in its class
  - The attribute values determine the *state* of the object



Class: Operations

- **Operations are**
  - Essential to the design description of a class
  - The common behaviour shared by all objects of the class
  - Services that objects of a class can provide to other objects
  - Invoked by messages
  - Two types
    - Command
    - Query



Class Attributes and Operations

An attribute is a named property of a class that describes the object being modeled.

In the class diagram, attributes appear in the second compartment just below the name-compartment.

Attributes can be:

+ public
# protected
- private

Attributes are usually listed in the form:

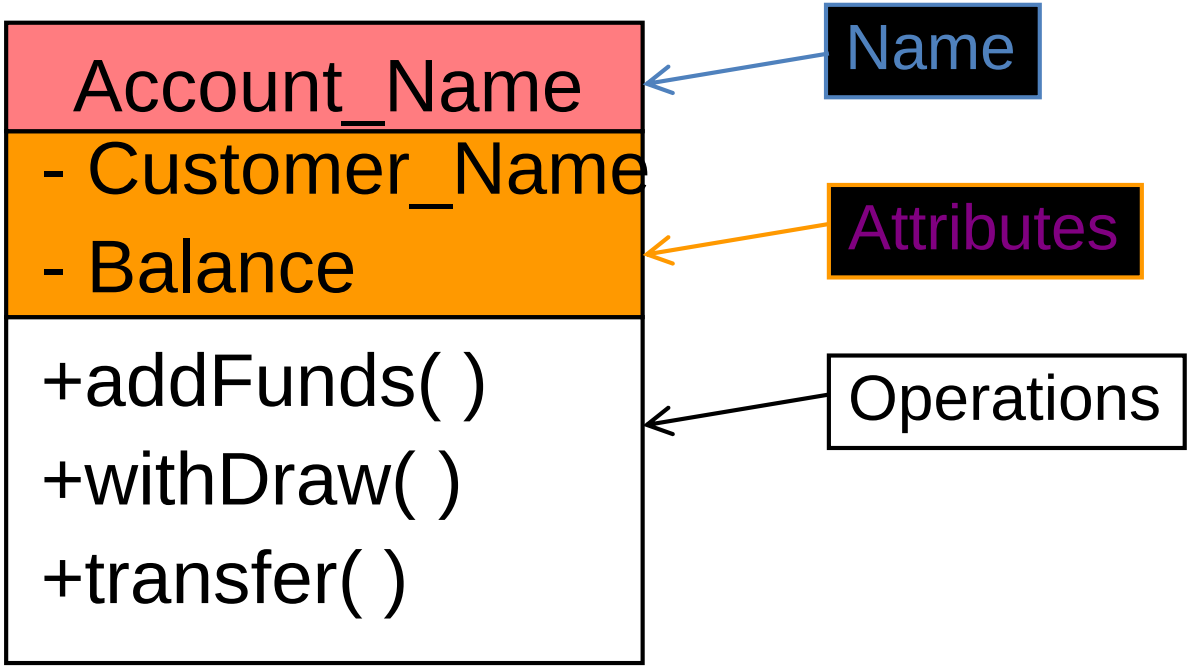
attributeName : Type **- number : int**

Operations describe the class behavior and appear in the third compartment.

Visibility and Access for attributes and operations of a class

Access	public (+)	private (-)	protected (#)
Members of the same class	yes	yes	yes
Members of derived classes	yes	no	yes
Members of any other class	yes	no	no

Class


Account_Name
- Customer_Name
- Balance
+addFunds( )
+withDraw( )
+transfer( )
Name
Attributes
Operations

Class Attributes and Operations

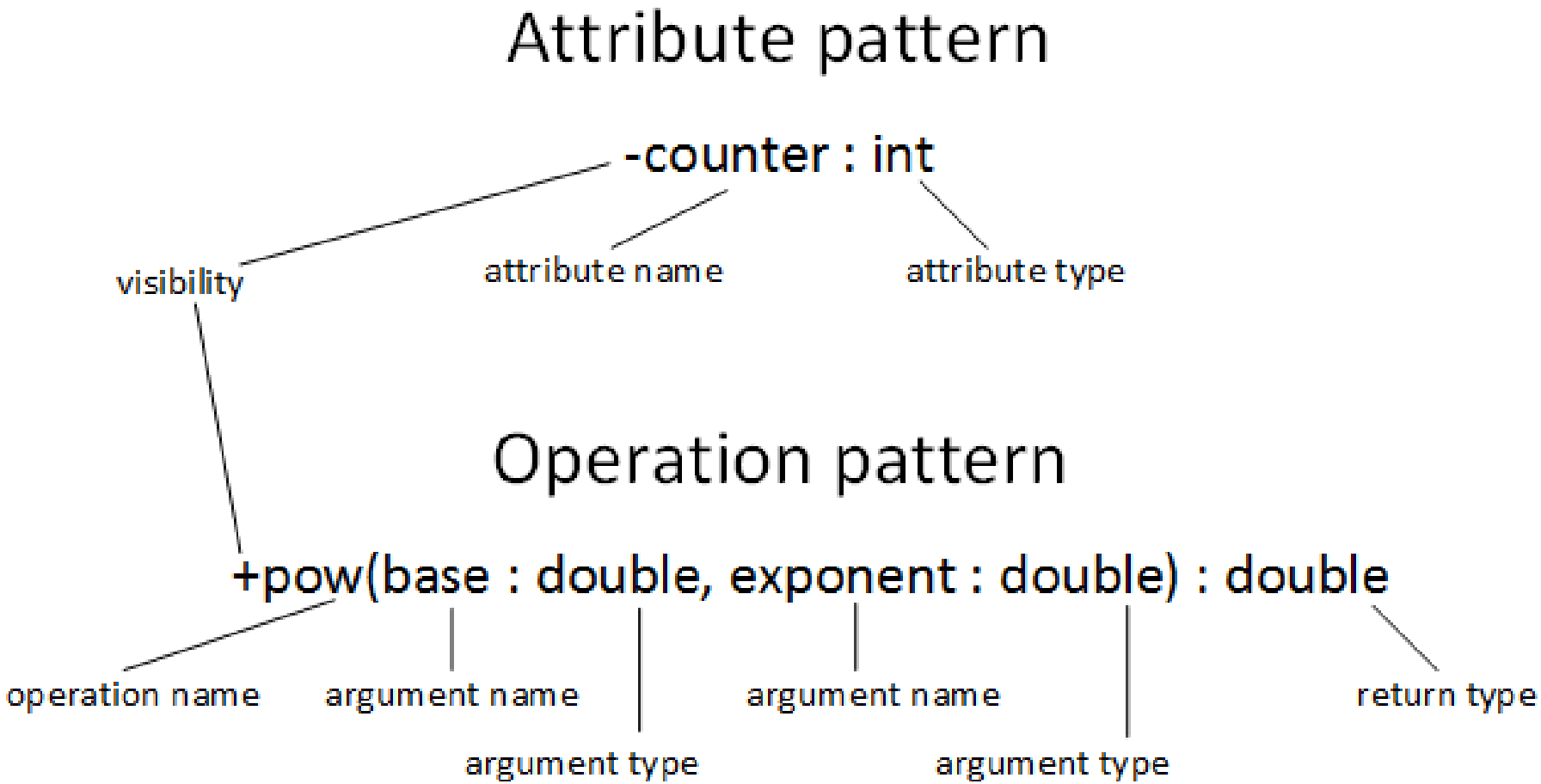
```
Person
-name : string
-height : double
-weight : int
-instances : int
<<constructor>>
+Person(a_name : string, a_height : double, a_weight :int)
<<process>>
+pay_taxes() : bool
+catch_bus(direction : int) : void
+get_instances() : int
<<helper>>
-get_address() : Address
```

```
public void catch_bus(int direction)
{

}
```

+ sum(num1 : int, num2: int) :int

Class Attributes and Operations

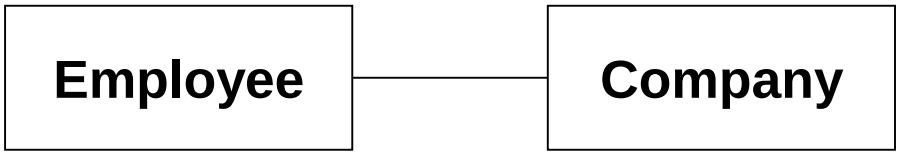


Relationships between classes

- Kinds of static relationships
  - Generalisation (Subtypes)
    - E.g. A programmer *is a kind of* human
  - Associations
    - E.g. A customer may *rent* a number of videos
  - Dependency
    - e.g. dependency

Association

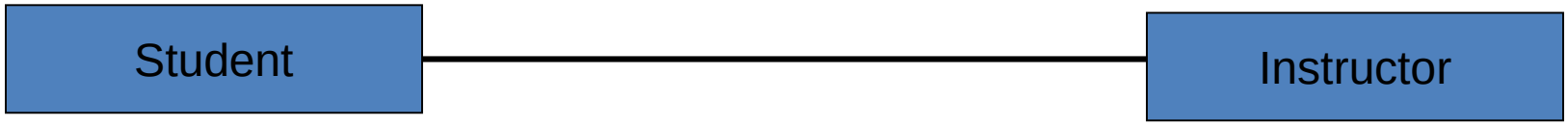
- A relationship between two separate classes. It joins two entirely separate entities.
- In UML, object interconnections (logical or physical), are modeled as **relationships**.
- An **association** between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.
- Example: "An Employee works for a Company"



Association Relationships

If two classes in a model need to communicate with each other, there must be link between them.

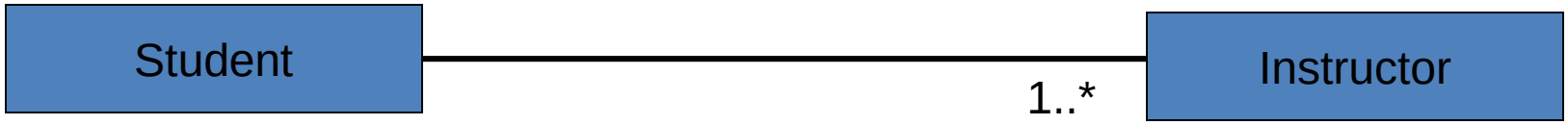
An *association* denotes that link.



Association Relationships

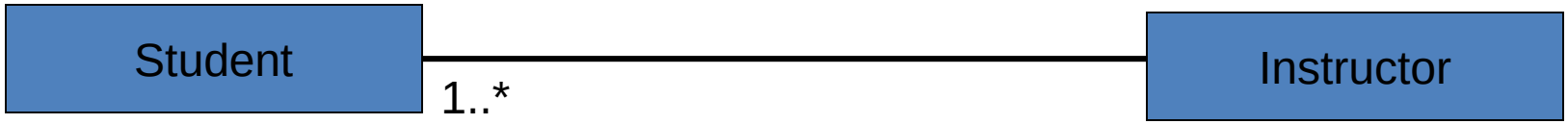
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:



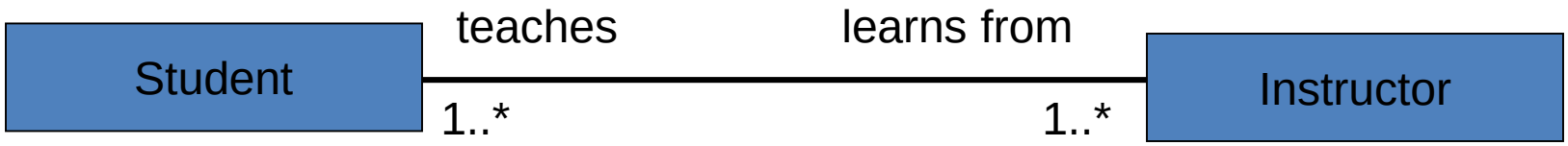
Association Relationships

The example indicates that every *Instructor* has one or more *Students*:



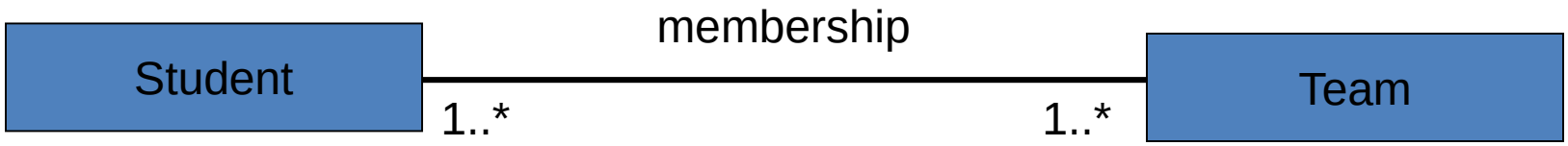
Association Relationships

We can also indicate the behavior of an object in an association (*i.e.,* the ***role*** of an object) using *rolenames.*



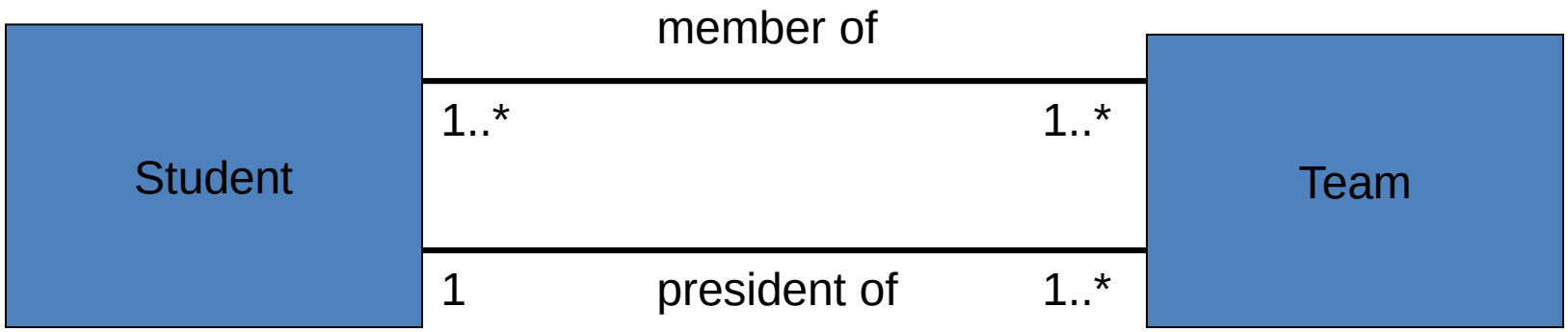
Association Relationships

We can also name the association.



Association Relationships

We can specify dual associations.



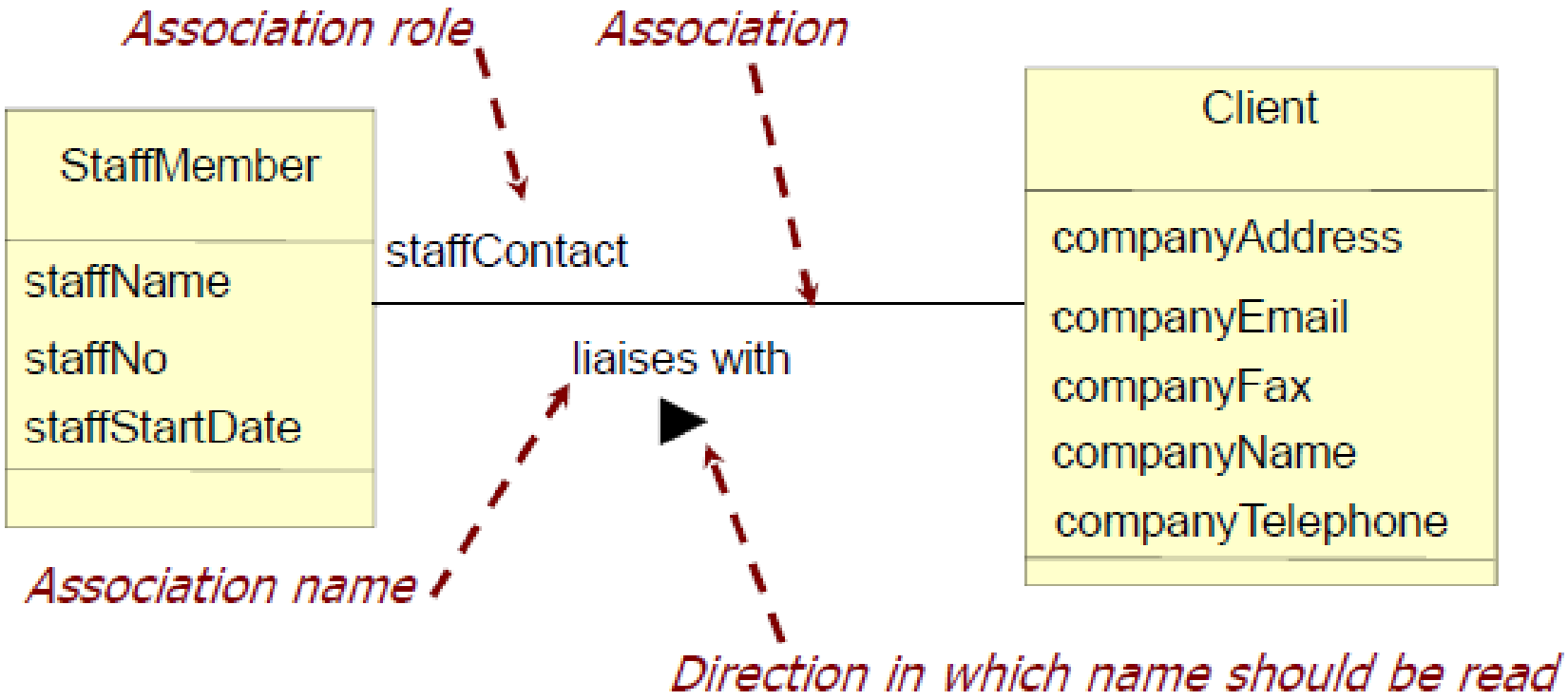
Associations

- Multiplicity
  - the number of objects that participate in the association.
  - Indicates whether or not an association is mandatory.

Multiplicity Indicators

Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

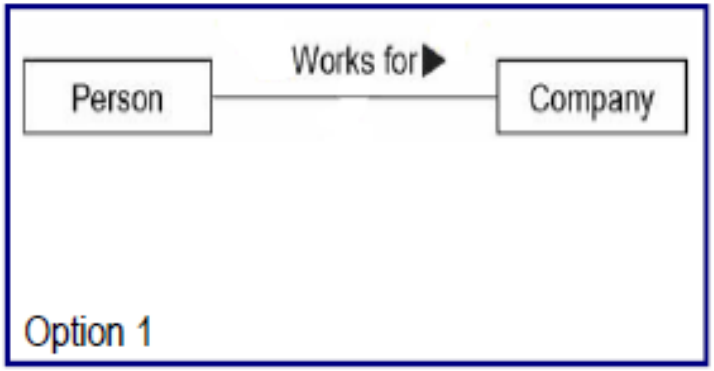
Class diagram: Associations

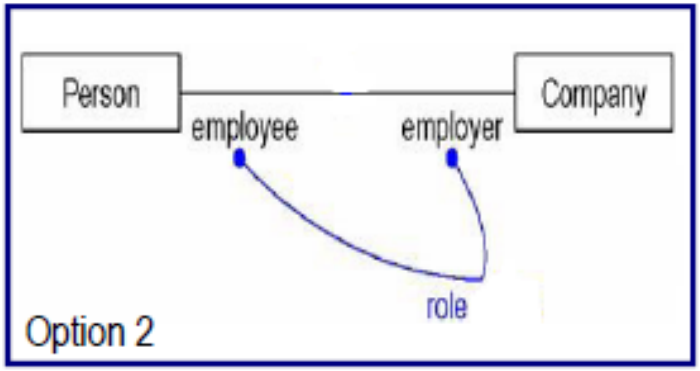


- **Associations represent:**
  - The possibility of a logical relationship or connection between objects of one class and objects of another.
  - Information stored about the associated objects.

Class relationship: Associations (cont.)

- An association can have a ***label***, which is used to describe the nature of the relationship
- You can exclude the association label, if you explicitly provide ***roles*** of classes for the association

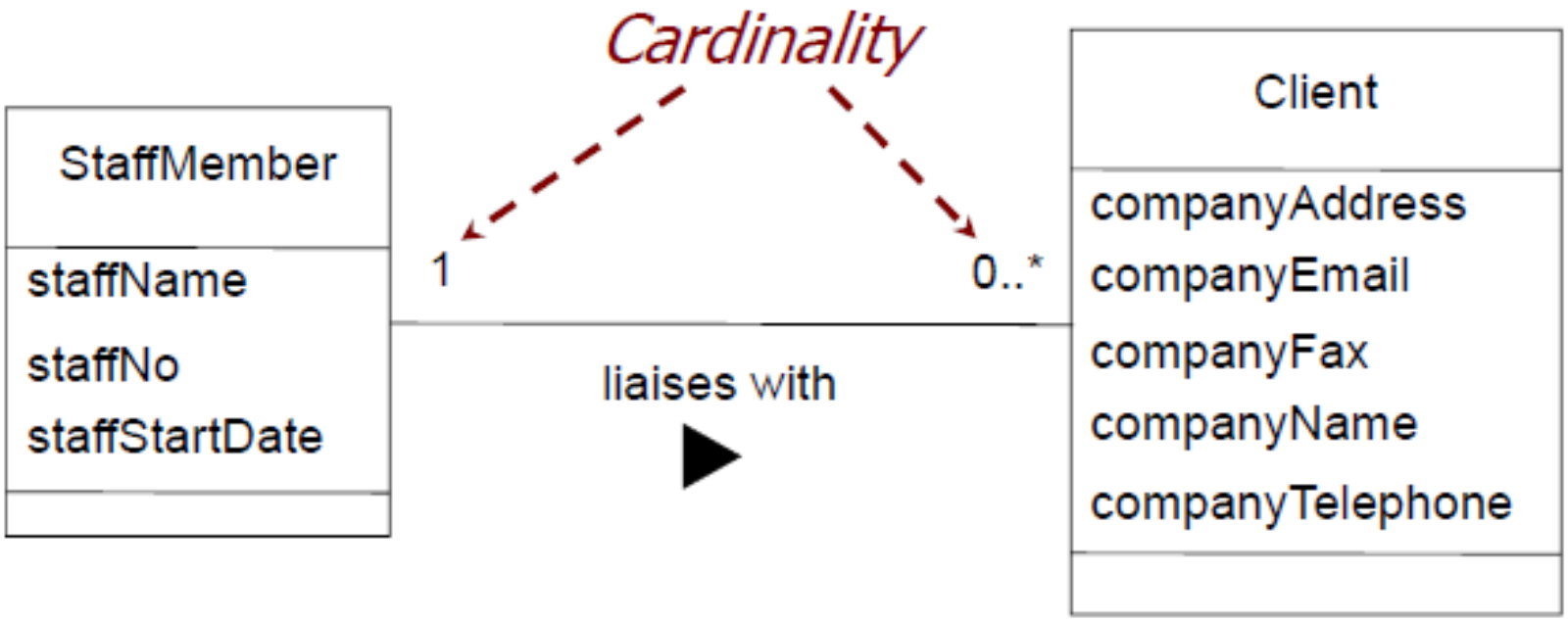




Class Diagram: Cardinality

- Associations have Cardinality
- Cardinality is the range of permitted objects linked in an association to a given object.
- Represent *business rules*.
- For example:
  - *A staff member may liaise with one or more clients.*
  - *Each client must liaise with just one, and only one, staff member.*

Class Diagram: Association cardinality



- Exactly one staff member liaises with each client
- A staff member may liaise with zero, one or more clients

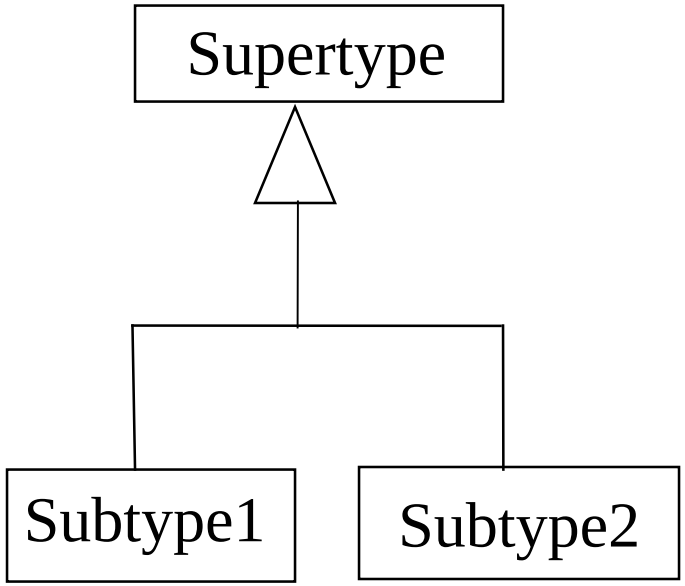
Class Diagrams: Constraints

- Not all domain properties can be shown graphically:
  - e.g., it should be impossible to double-book a table.
- *Constraints* add information to models:
  - written in a *note* connected to the model element being constrained.

Constraints

Generalization Relationship

- A *generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.



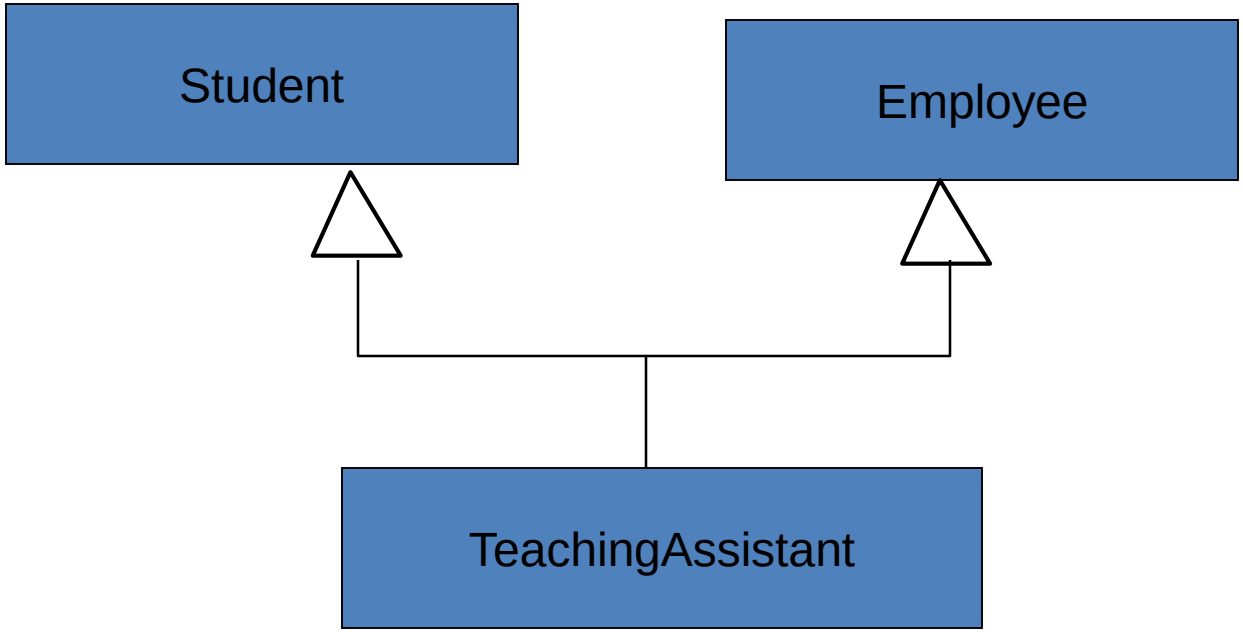
- "is kind of" relationship.

Generalization

- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship **may not** be used to model interface implementation.

Generalization Relationships

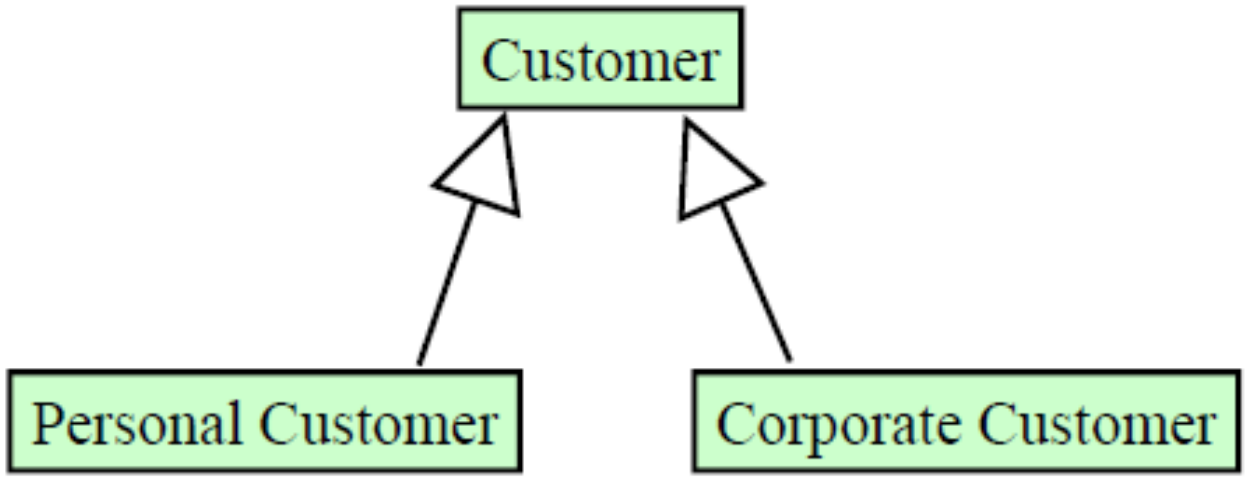
UML permits a class to inherit from multiple super classes, although some programming languages (*e.g.,* Java) do not permit multiple inheritance.



Class relationships: Generalization

- Consider the following example:
  - A company has *personal customers* and *corporate customers*
  - There are differences between these two types of *customers*
  - There are also many similarities between these *customers*
- UML recommends the use of "*generalization*" to model situations like in the example above

Class diagram: Generalization



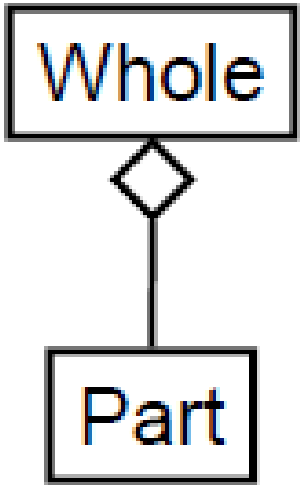
- A personal customer *is a kind of* customer
- A corporate customer *is a kind of* customer

Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
- A directional association between objects.
- When an object ‘has-a’ another object, then we have got an aggregation between them.
- Direction between them specified which object contains the other object.
- It is also called ‘Has-a’ relationship.

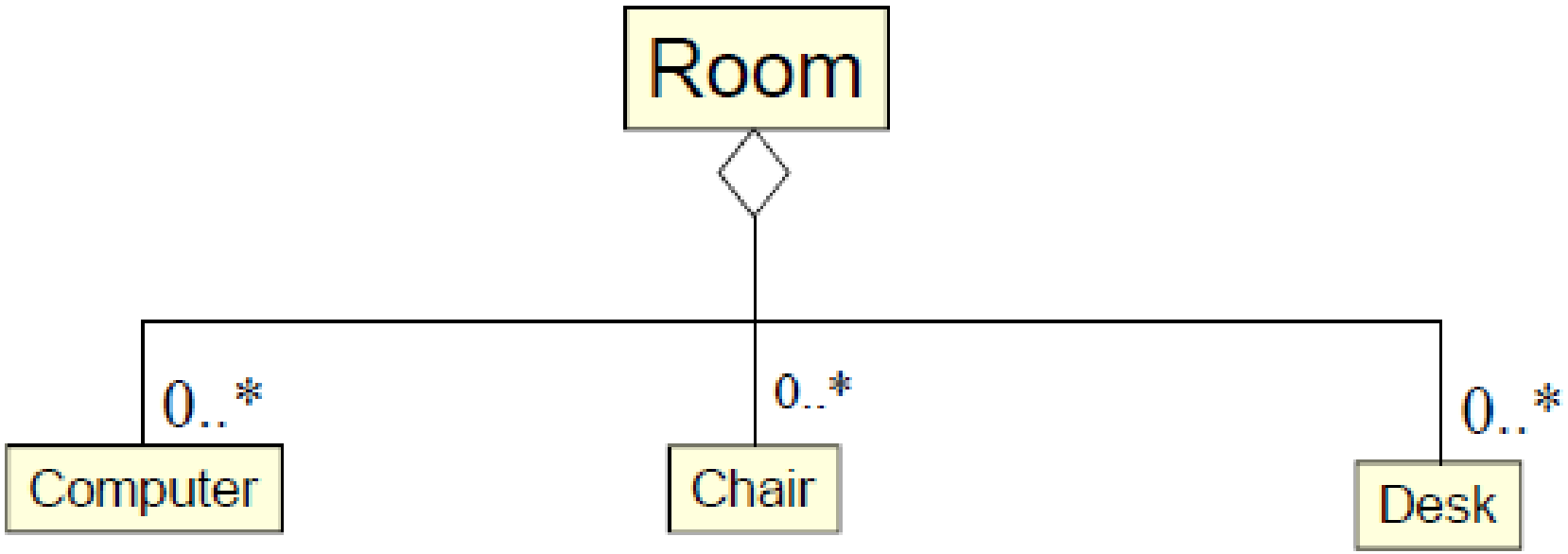
Class diagram : Aggregation

- When modelling we often find the need to capture "whole-part" relationships
  - An office room *has* furniture
- A type of association called **aggregation** can be used to capture whole-part relationships
- Aggregation is modelled as a *hollow diamond* in UML

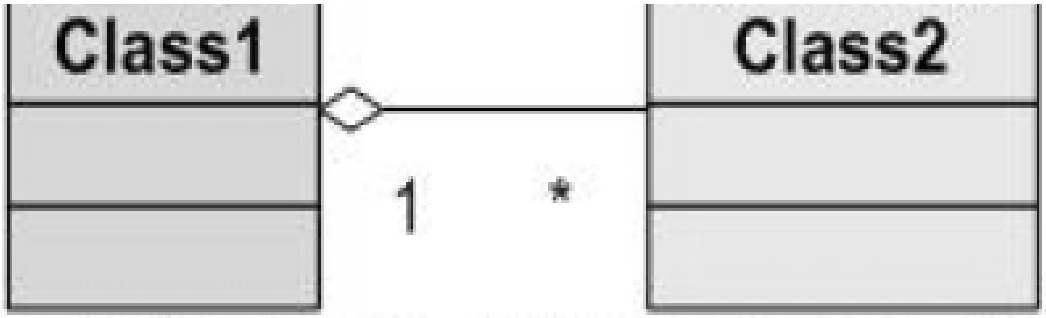


Aggregation example

- An office room *has* furniture



Aggregation



- *Class2* is part of *Class1.*
- Many instances (denoted by the *) of *Class2* can be associated with *Class1.*
- Objects of *Class1 and Class2* have separate lifetimes.

Aggregation

- **<u>Note:</u>**

- If we delete the parent object, even then the child object may exist.
- One object can contain the other, but there is no restriction that the composed object has to exist in order to have existence of child object.

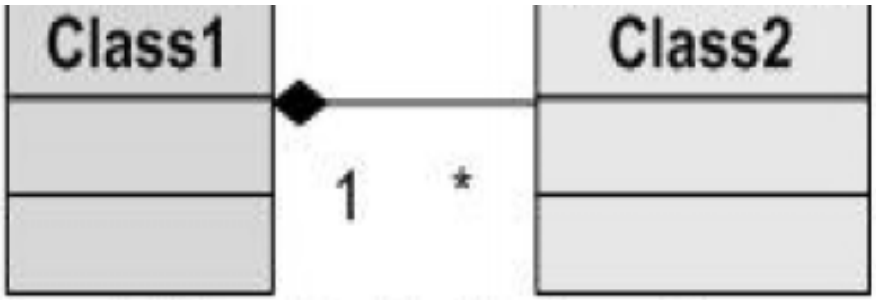
Composition

- A variation on the simple aggregation – that adds some extra semantics (meaning/rules)
- Strong ownership
- Coincident lifetime as part of the whole
  - Parts may be created after the composite itself
    - But once born, they live & die with the whole
    - These parts may be removed before the death of the whole
  - An object may be a part of only one composite at a time

Composition

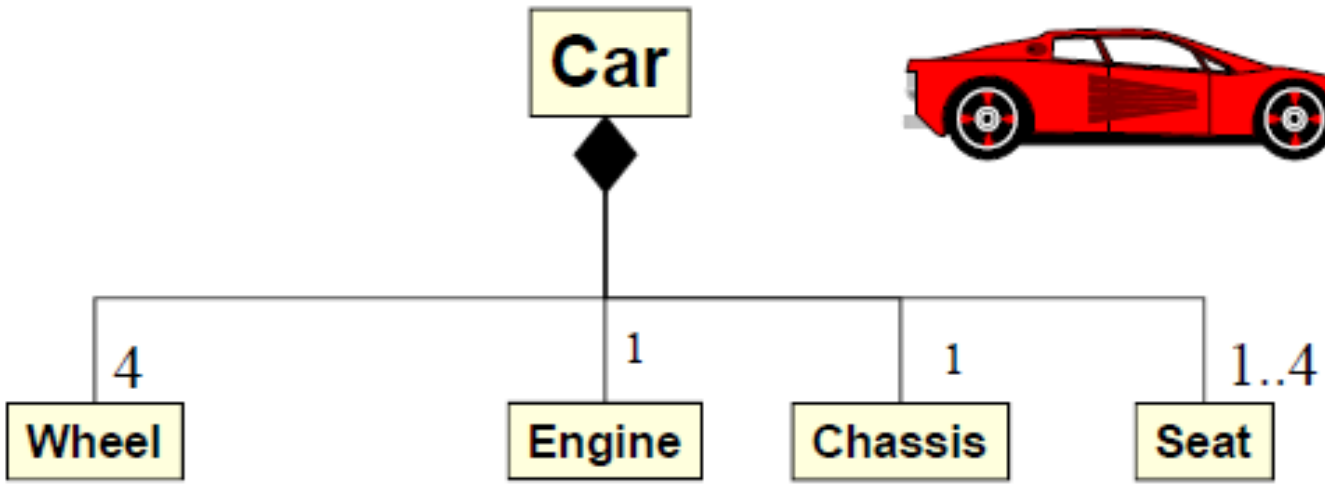
- A strong form of aggregation
- In a more specific manner, a restricted aggregation is called composition.
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.

Composition



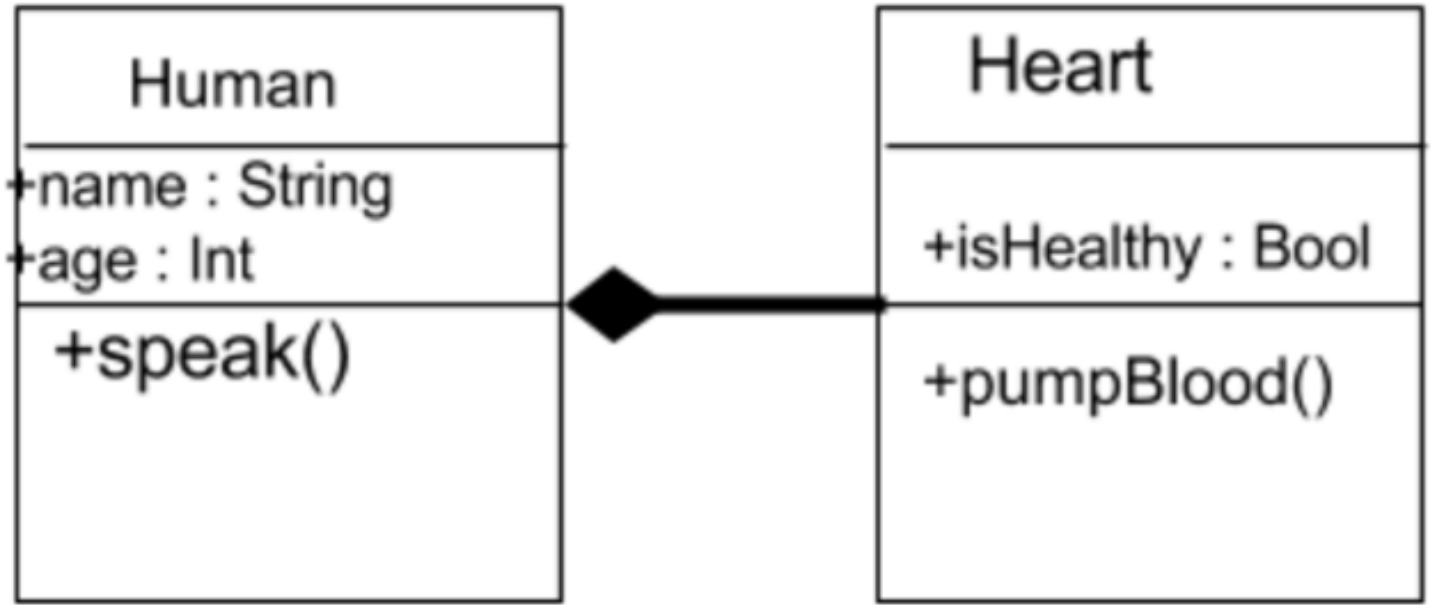
- Objects of *Class2* *live and die* with *Class1.*
- *Class2* *cannot stand by itself.*

Composition Example

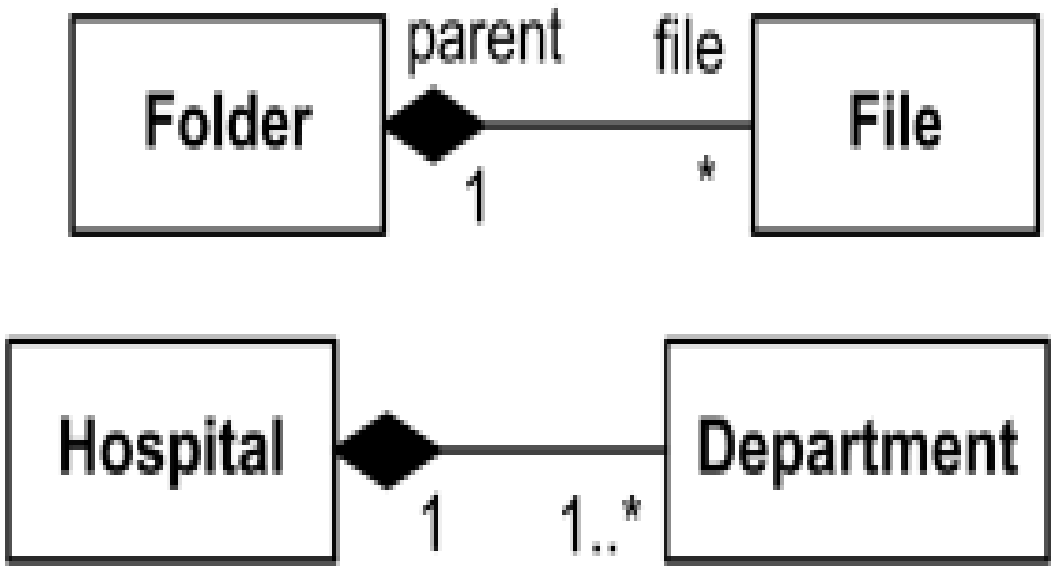


- A wheel cannot be shared between cars
- Car is created implies engine, chassis & 4 wheels are created
- Seats may be added/removed after Car is created
- When a car is destroyed all parts are also destroyed

Composition Example



Composition



- A class contains students. A student cannot exist without a class. There exists composition between class and students.

Difference between aggregation and composition

- Composition is more **restrictive**. When there is a composition between two objects, the composed object cannot exist without the other object.

- This restriction is not there in aggregation. Though one object can contain the other object, there is no condition that the composed object must exist. The existence of the composed object is entirely optional.

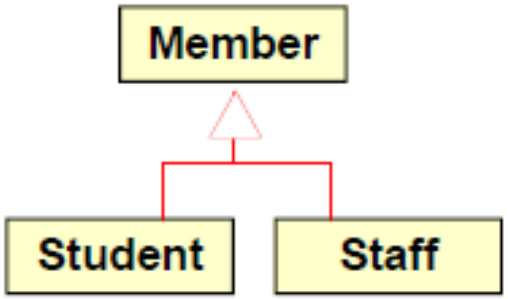
Building a class diagram

- The Noun/Verb approach is the most widely used technique to help identify candidate classes and potential operations -- Textual analysis
- The idea is to look for nouns and noun phrases in the narrative or use cases

Noun / Verb approach

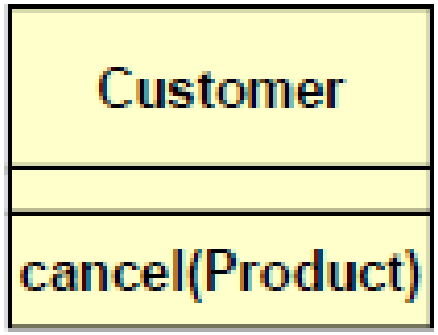
- Verbs
  - ✓ Facts about objects
    - ☐ eg. *Customer orders product*
    - ☐ Modelled by associations in the Entity Class Diagram.

  - ✓ Classification Declarations
    - ☐ eg *All students and staff are members.*
    - ☐ The verb in this context is modelled as a *generalization* in the Entity Class Diagram



Noun / Verb approach

- Verbs
  - Actions involving objects *may* become operations on objects.
    - These are best modelled during design.
    - e.g. *Customer cancels product*

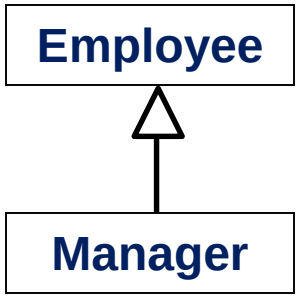


Example of Relations

1. A manager is a type of employee.
2. Manager uses a swipe card to enter company premises.
3. He has many workers under him.
4. Manager uses a car which contains an engine and four wheels.

Example of Relations

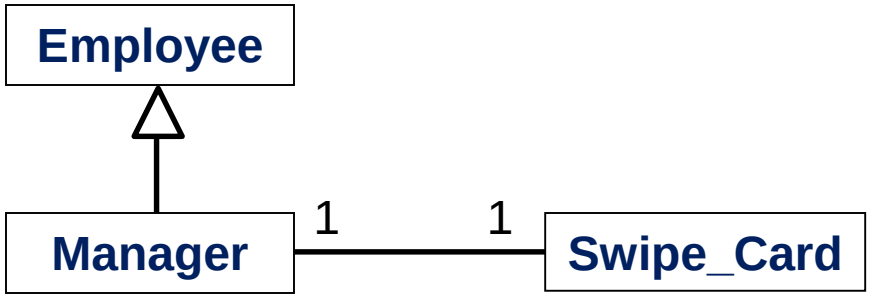
1. **A manager is a type of employee.** ← **Inheritance**
2. Manager uses a swipe card to enter company premises.
3. He has many workers under him.
4. Manager uses a car which contains an engine and four wheels.



Example of Relations

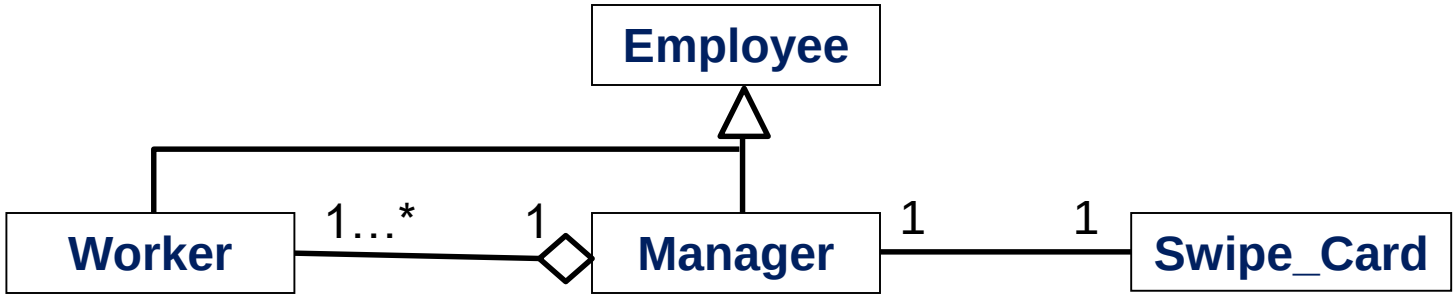
1. A manager is a type of employee.
2. **Manager uses a swipe card to enter company premises.**
3. He has many workers under him.
4. Manager uses a car which contains an engine and four wheels.

Association



Example of Relations

1. A manager is a type of employee.
2. Manager uses a swipe card to enter company premises.
3. **He has many workers under him.** ← **Aggregation**
4. Manager uses a car which contains an engine and four wheels.

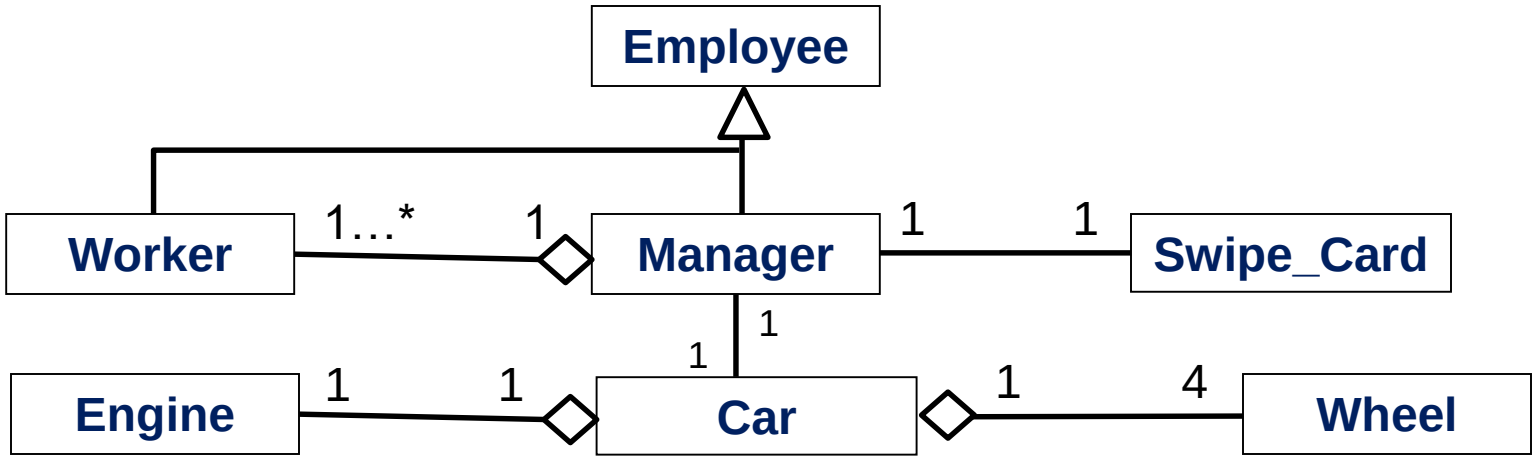


Example of Relations

1. A manager is a type of employee.
2. Manager uses a swipe card to enter company premises.
3. He has many workers under him.
4. **Manager uses a car which contains an engine and four wheels.**

Association

Composition



Example of Relations

1. A manager is a type of employee.
2. Manager uses a swipe card to enter company premises.
3. He has many workers under him.
4. Manager uses a car which contains an engine and four wheels.