CSY2030
Systems Design & Development

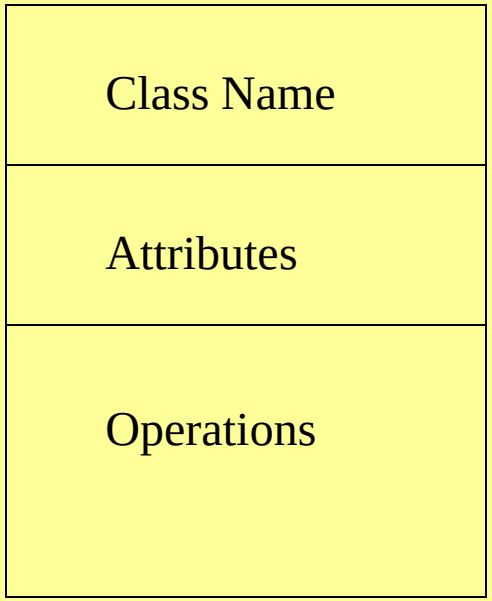
UML Class Diagrams

Class Diagrams

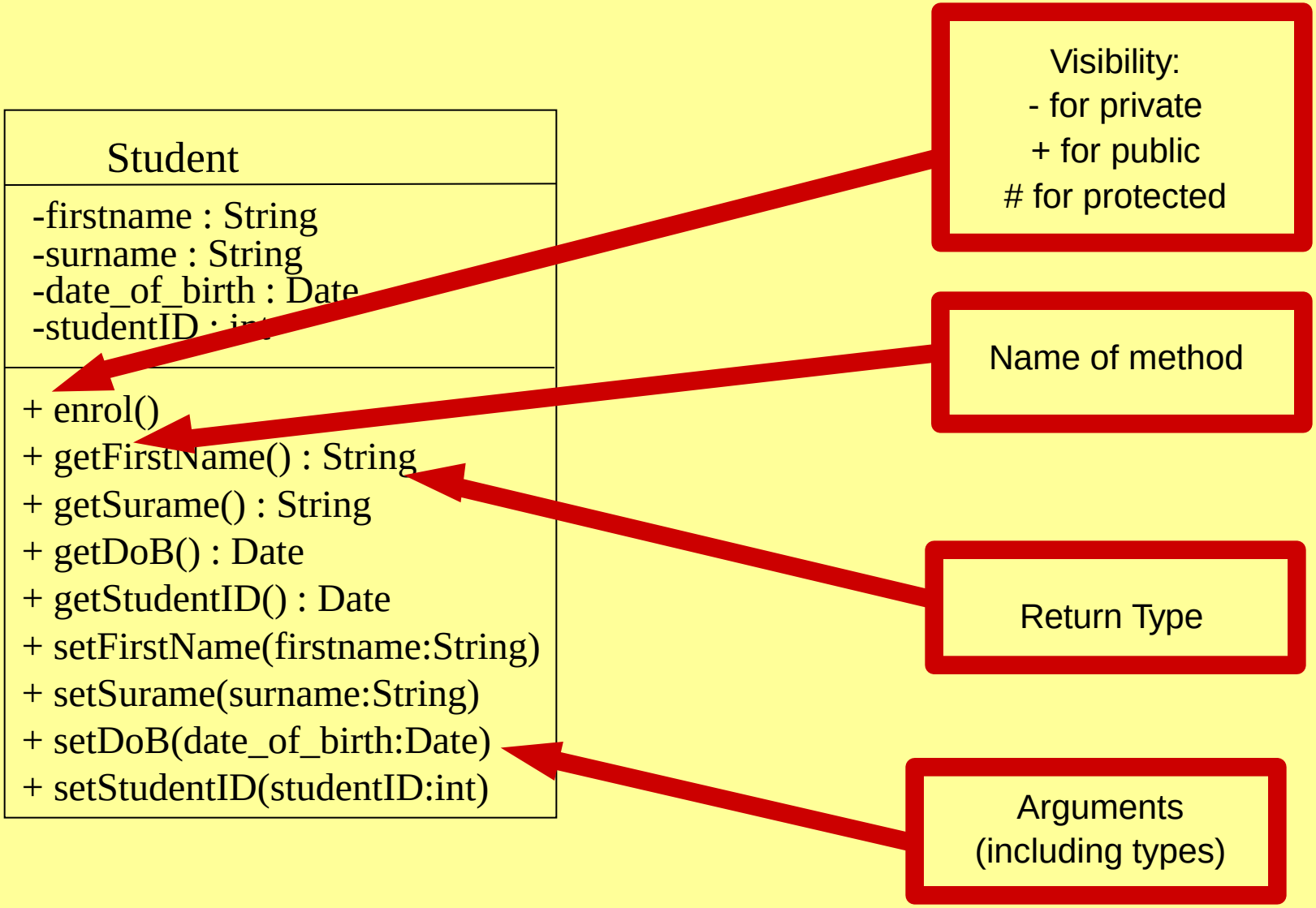
- Class Diagrams presents a view of the static structure of a potential software system
- The Class Diagram models an acceptable solution to the problem posed by the requirements
- It describes the objects and relationships that are needed between them to implement the required functionality
  - You need to find ways to:
    - Describe the required behaviour using use cases and activity diagrams
    - Define the classes composing the software
    - Describe how instances of the classes interact to achieve the required system behaviour

Classes

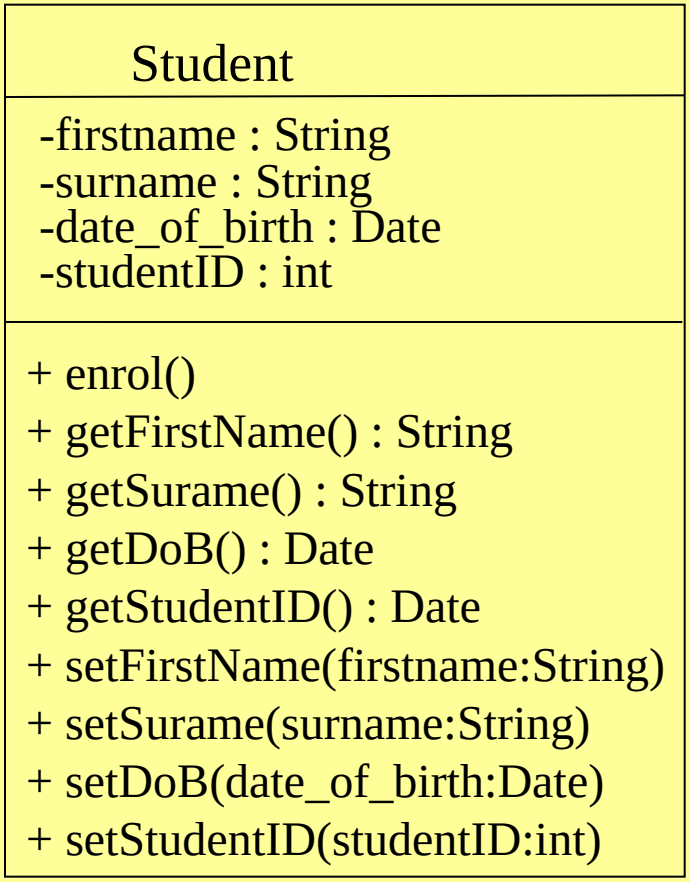
- Class are a collection of objects with common structure, behaviour, relationships and semantics
  - Usually nouns in a specification
  - Obtain from requirements document
- Represented by a box with lines separating
  - Name of class
  - Structure represented by attributes
  - Behaviour represented by operations

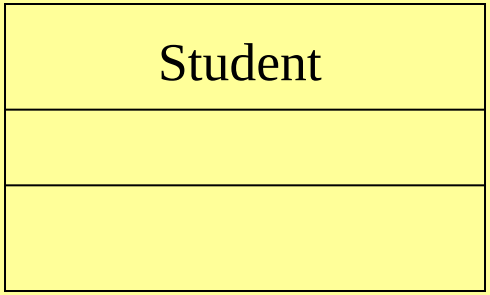


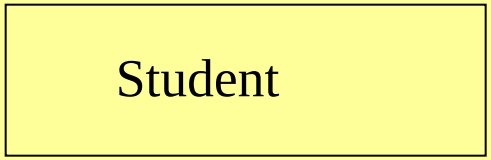
Classes - Example



Classes - examples





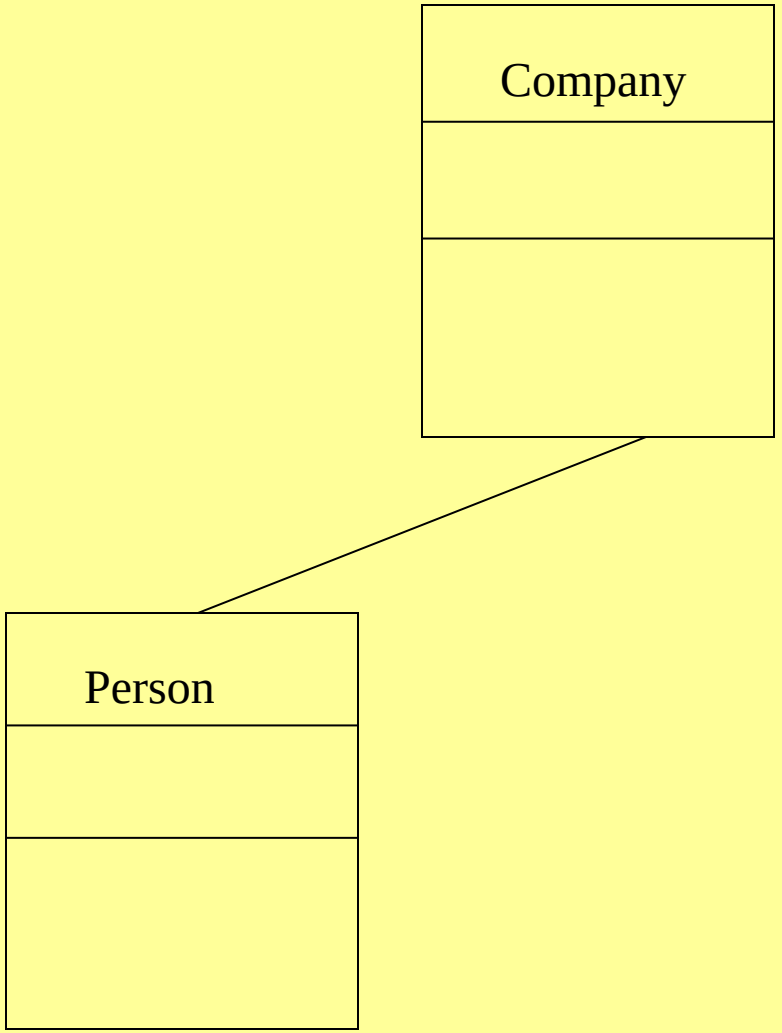


Relationships

- These are the links required by objects in order to accomplish a task
  - Usually a verb in the specification
- If two objects need to talk, there must be a link between them
- There are 3 main types of relationships:
  - Association
  - Aggregation (and composition)
  - Inheritance

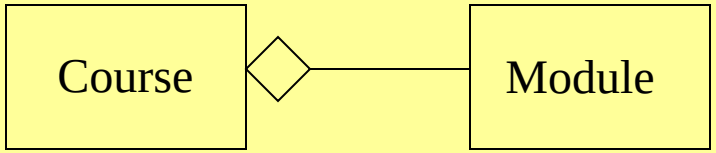
Relationships - Association

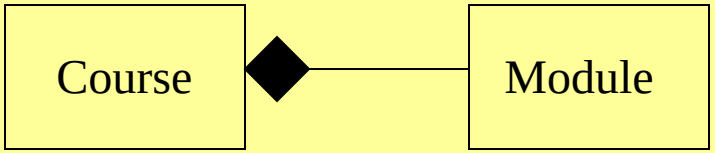
- An association indicates that there will be some communication between the classes
- Relationship is loose
- Relationship is bi-directional (usually)
- Represented as a line between classes



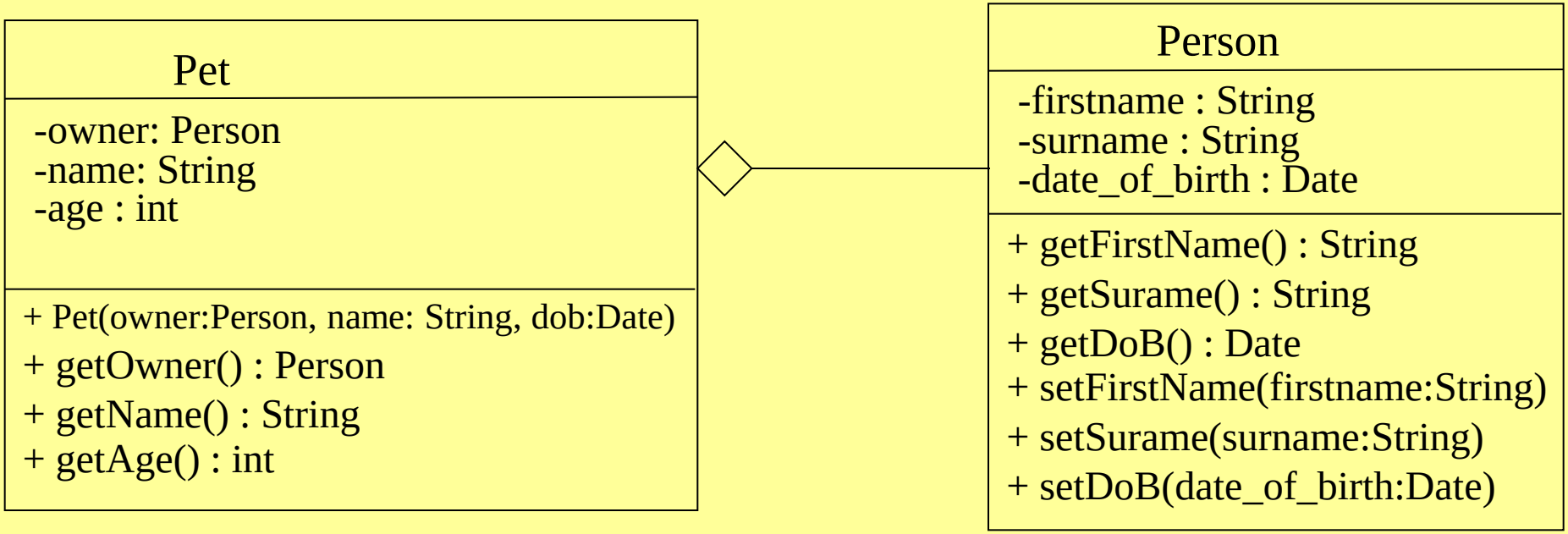
Relationships - Aggregation

- This means one class is ‘part of’ another class
  - A tighter relationship
  - Represented by a diamond
- Can be considered a ‘has-a’ relationship
- **Composition** is a stronger form of aggregation where the lifetime and existence of the part is governed by that of the whole
  - Represented by a filled in diamond





Relationships – Aggregation Example



Note: The Pet class has an attribute of the type Person

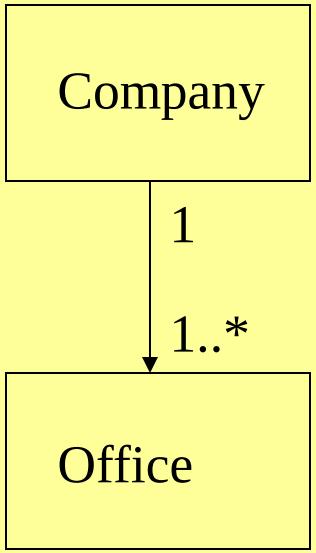
Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationship
  - It is the number of instances of one class related to one instance of another class
  - For each association (and aggregation) a multiplicity has to be defined for each end of the relationship
- Relationships are usually bi-directional, but it is sometimes useful to restrict this to one direction - indicated by an arrowhead
  - this is a.k.a **navigation**

Multiplicity and Navigation

Multiplicities:

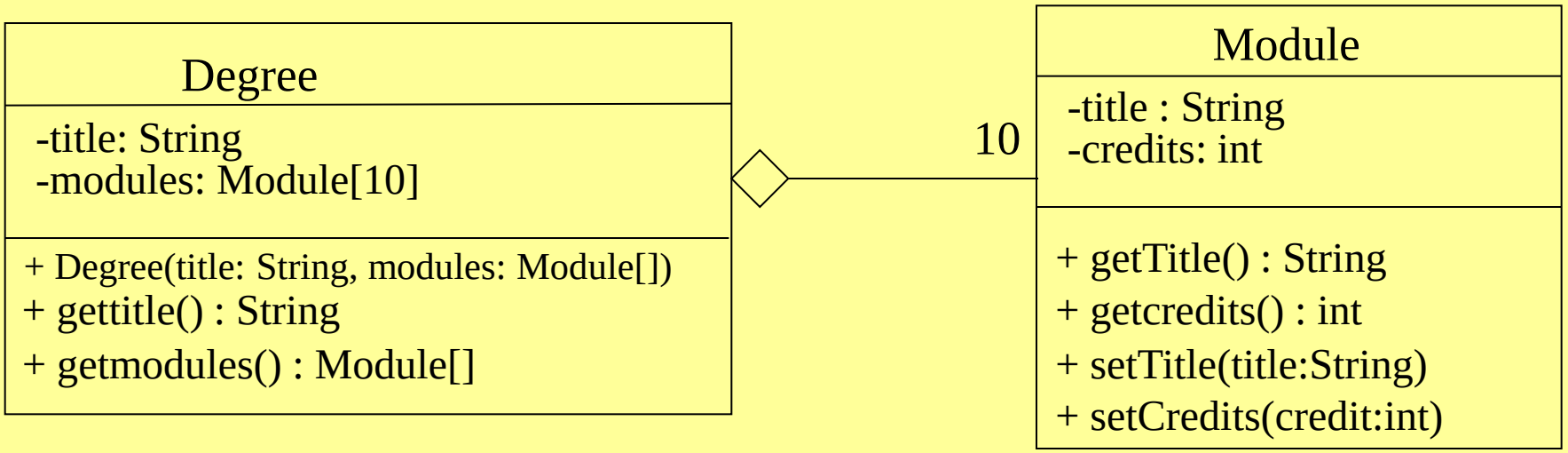
- (blank)
- 0
- 1
- *
- 1..*
- n..m



A company has one or more offices

An office belongs to one company

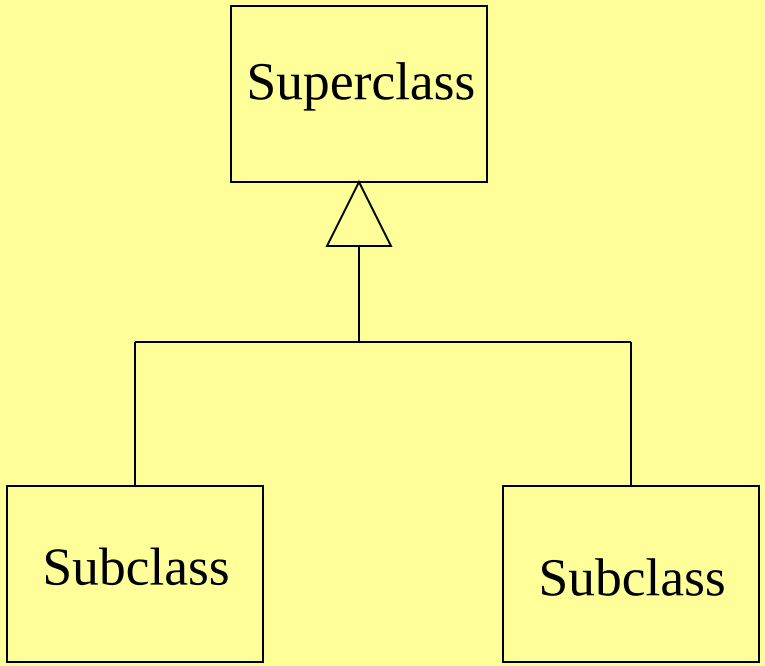
Relationships – Multiplicity Example



Note: The Degree class has an attribute which is an array of type Module

Inheritance

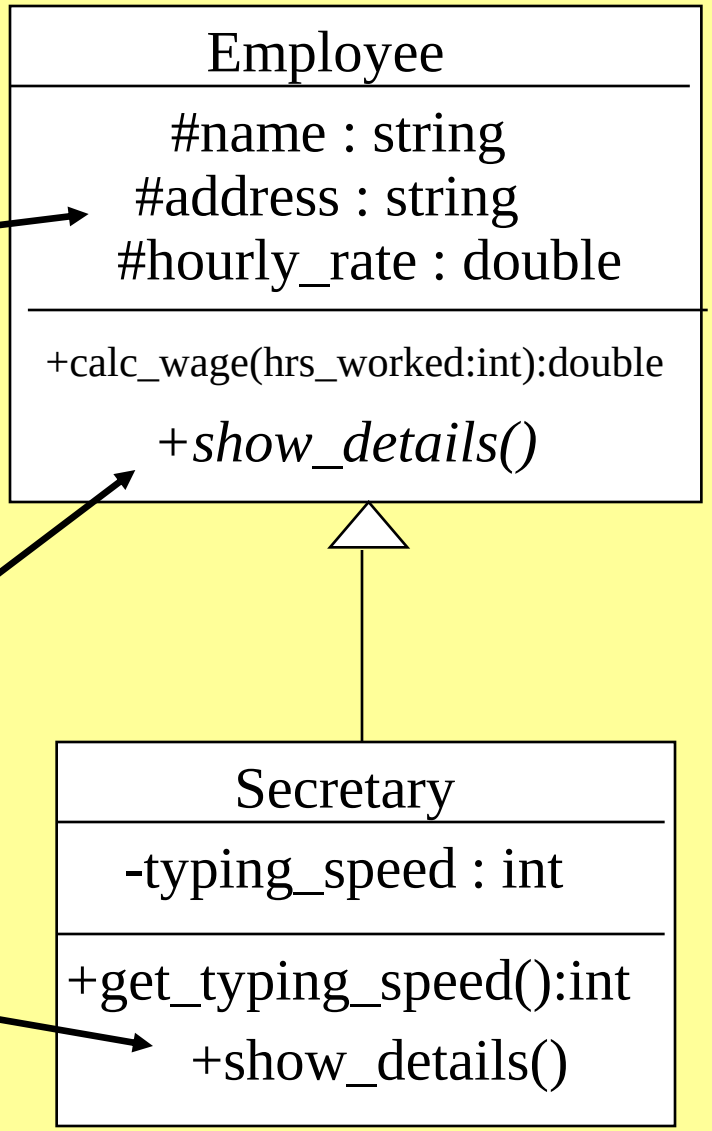
- Inheritance is Generalisation / Specialisation between superclass and subclass
  - Can be considered a ‘is-a’ relationship
- Common attributes, operations / relationships are shown at the highest applicable level in the hierarchy
- Inheritance is represented by an arrow head between superclass and subclasses



Inheritance

We use # to show that an attribute is protected

- This means it is private in superclass and accessible by subclasses
- If it was left private then subclass cannot access superclass's attributes



If method appears in both superclass and subclass then this means the method is redefined (replaced) in the subclass i.e the method in the subclass overrides the method of the superclass

- notation is to put superclass's method in italics

Interface and Abstract Classes

- An *Interface* class is a class with no attributes and a set of operations which have no implementation
  - It is up to the subclasses to provide the implementation
  - Represented by class name preceded by <<**interface**>> and connections are dashed lines
- *Abstract* classes are similar to interface classes but they may have implementations defined for some of their operations and they may also have attributes
  - Represented by class name preceded by <<**abstract**>> and connections are dashed lines

Interface and Abstract Classes

Example