Lecture 6

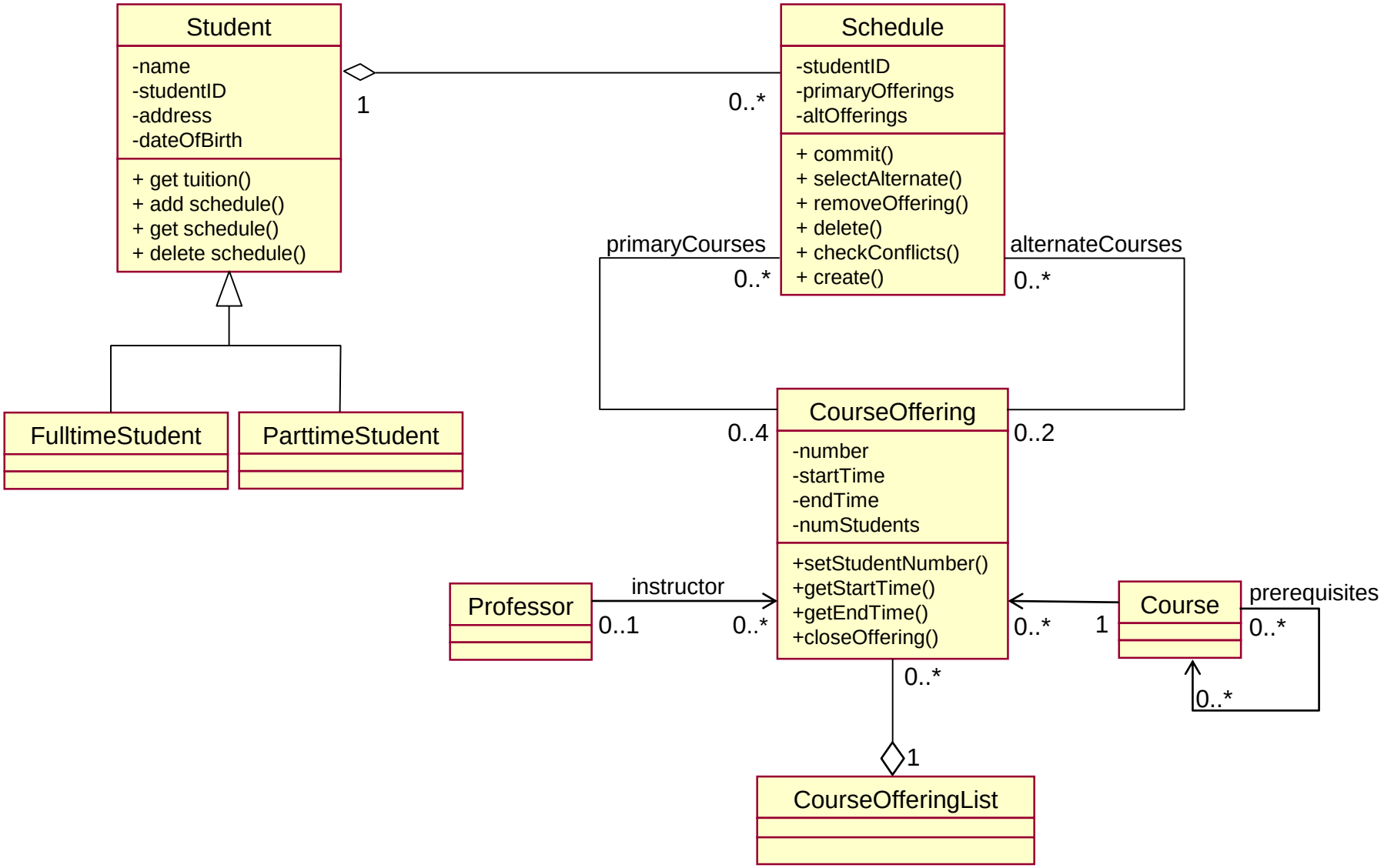
Modelling Relationships in Class Diagrams

Philipp Reinecke
reineckep@cardiff.ac.uk

Class Diagrams

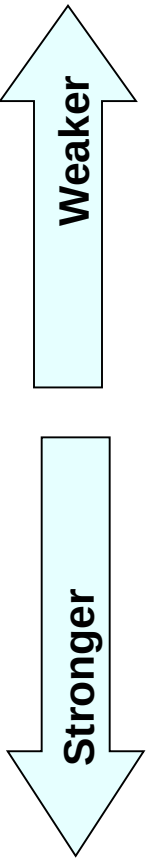
- A class
  - Specifies common data elements (attributes)
  - Specifies behaviour (operations)
- A class diagram
  - Models interaction between system elements
  - Specifies links & relationships between classes
  - Gives visual representation without code syntax
  - Also called domain model

Class Diagram



Class Relationships

Class diagrams may contain the following relationships:



- Association 

- Aggregation 

- Composition 

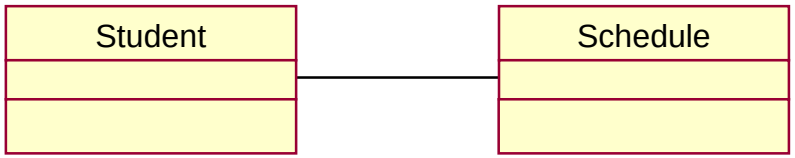
- Generalization 

Association

Association

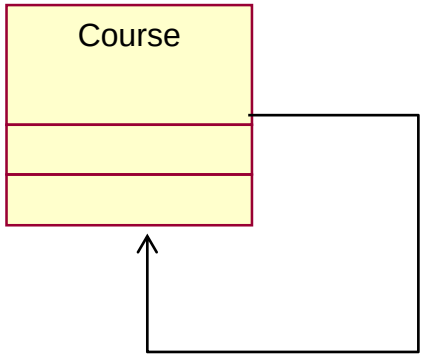
OR →

- The semantic relationship between two or more classes that specifies connections among their instances
  - A structural relationship, specifying that objects of one type are connected to objects of another
- Example: Students and Schedules are connected



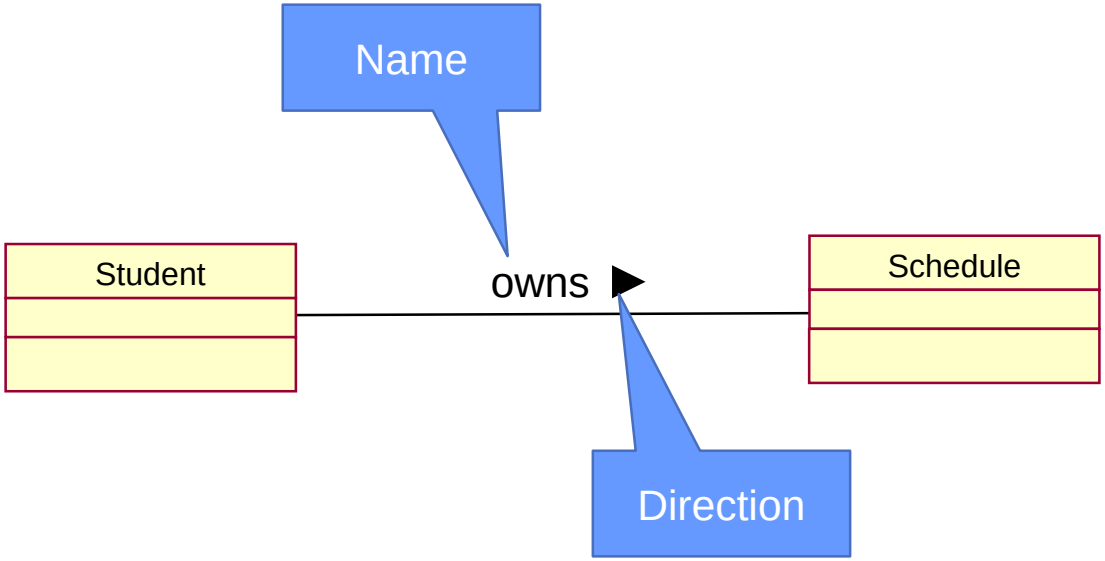
Association to Self

- A class can have an association to itself.
- This means: One instance of the class has associations to other instances of the same class.
- Example: A course may have other courses as prerequisites



Association Names

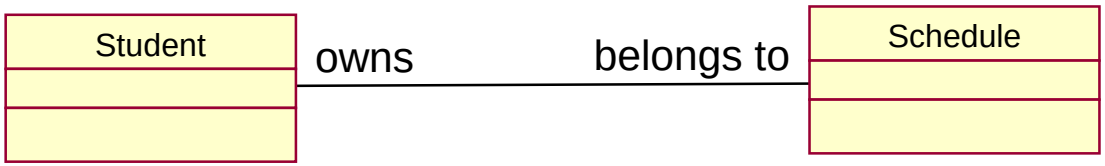
- An association can have a name
  - Describes the nature of the relationship



- Direction triangle indicates reading direction
  - Avoids ambiguity

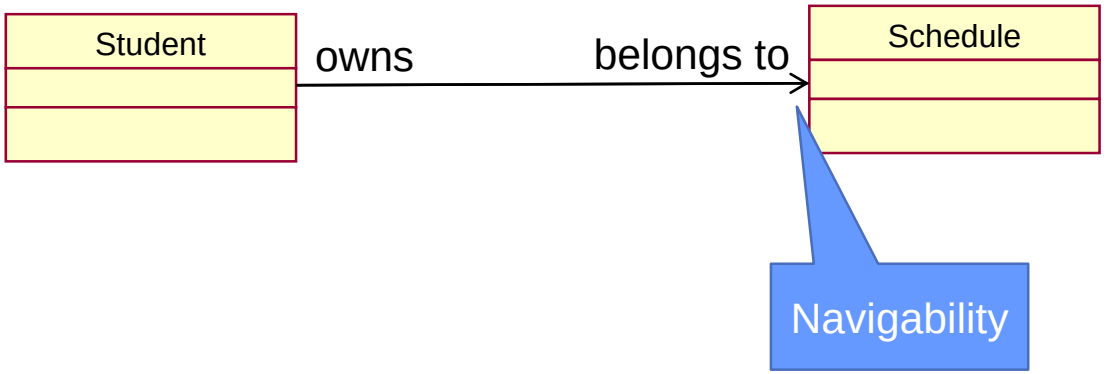
Association Roles

- A class plays a specific role in each relationship.
- End name: Role played by the class at the end of the association
- Each association may have 2 roles
  - Useful in describing or reading a diagram



Association Navigability

- Navigability:
  - Describes which class contains the attribute that supports the relationship
  - Indicated by navigability arrow

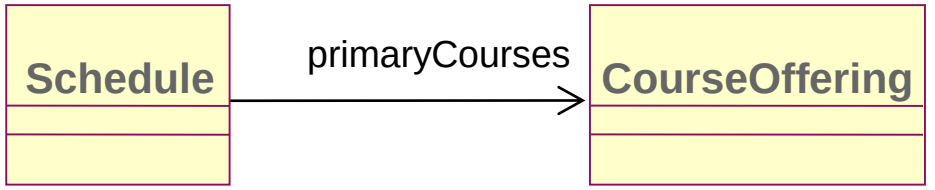


A Student must know the Schedule assigned to it, but the Schedules have no knowledge of the Students

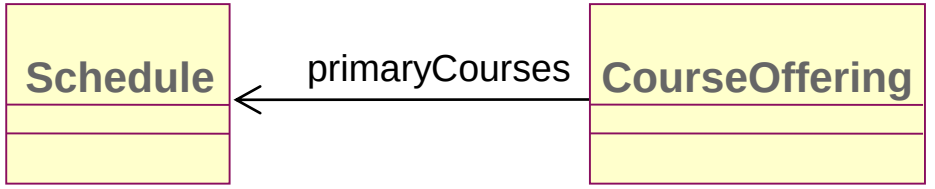
- Can navigate from Student to Schedule

Example: Navigability Refinement

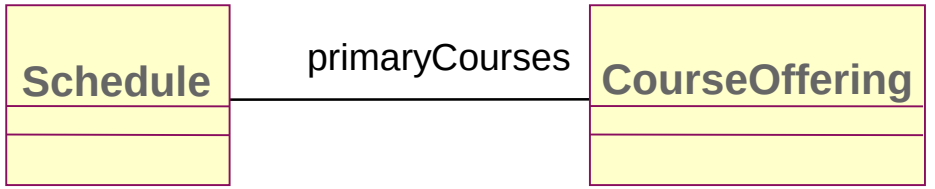
- Each Schedule knows its CourseOfferings
- Use if
  - Total number of Schedules is small, or
  - No list of Schedules on which a CourseOffering appears is needed



- Each CourseOffering knows the Schedules it belongs to
- Use if
  - Total number of CourseOfferings is small, or
  - No list of CourseOfferings on a Schedule is needed

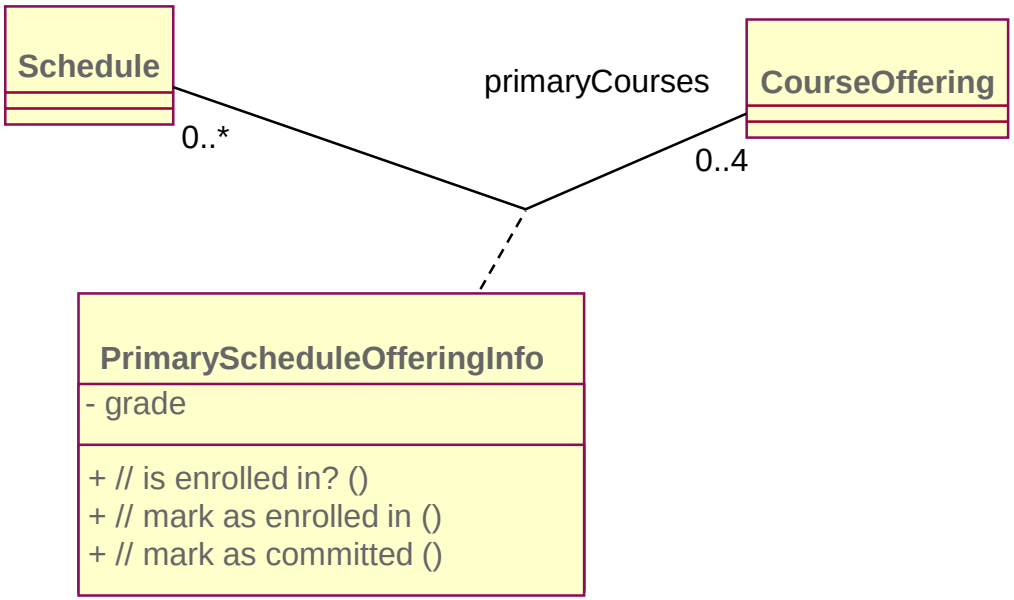


- Schedules and CourseOfferings know each other
- Use if:
  - Total number of CourseOfferings and Schedules are not small
  - Navigation in both directions required



Association Class

- A class that is "attached" to an association
- Contains properties of a relationship
- Has one instance per link

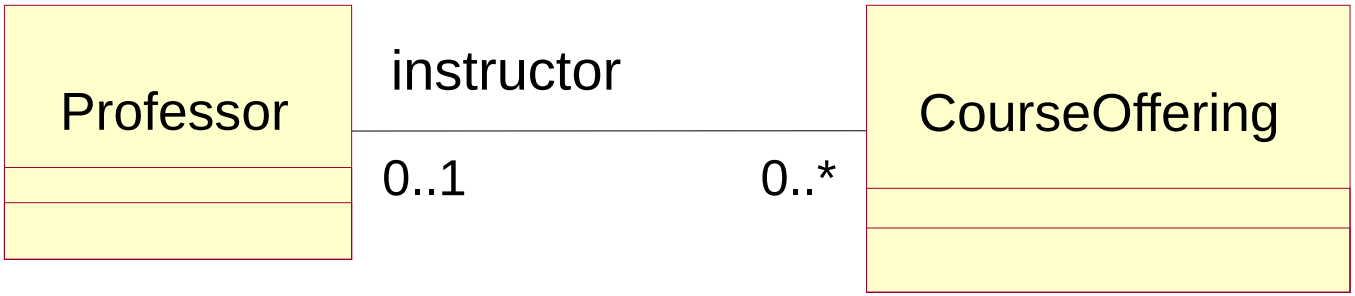


Association Multiplicity

- Multiplicity is the number of instances in one class that relates to ONE instance of another class.
- For each association, there are two multiplicity decisions to make, one for each end of the association.

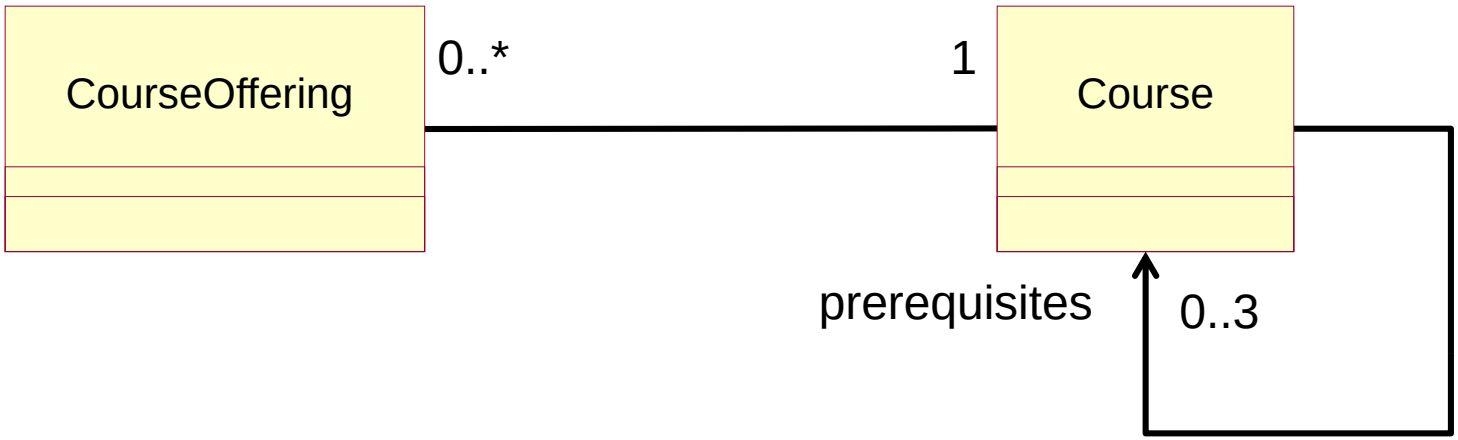
In this association:

- One CourseOffering may be taught by either 0 or exactly 1 Professor.
- One Professor can teach 0 or many CourseOffering objects.



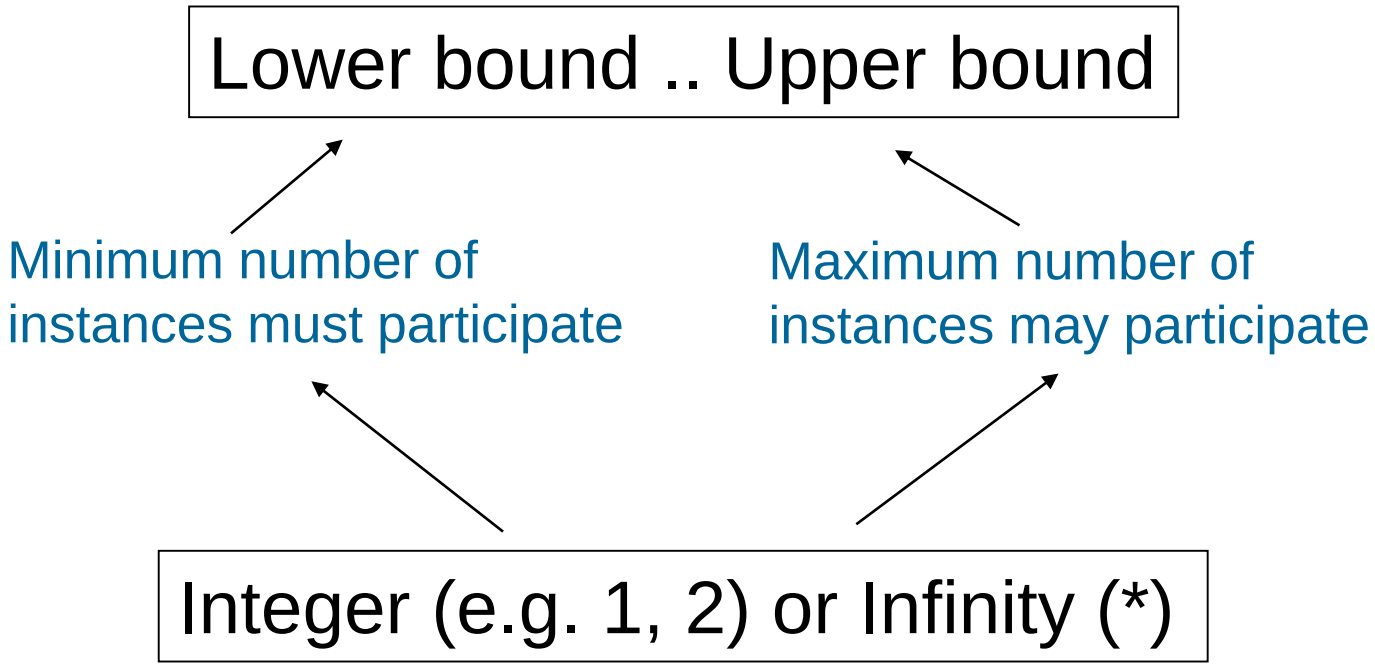
What Does Multiplicity Mean?

- Multiplicity answers two questions:
  - Is the association mandatory or optional?
  - What is the minimum and maximum number of instances that can be linked to one instance?



Specifying Multiplicity

- For each role of an association, we specify multiplicity as a range



Multiplicity Indicators

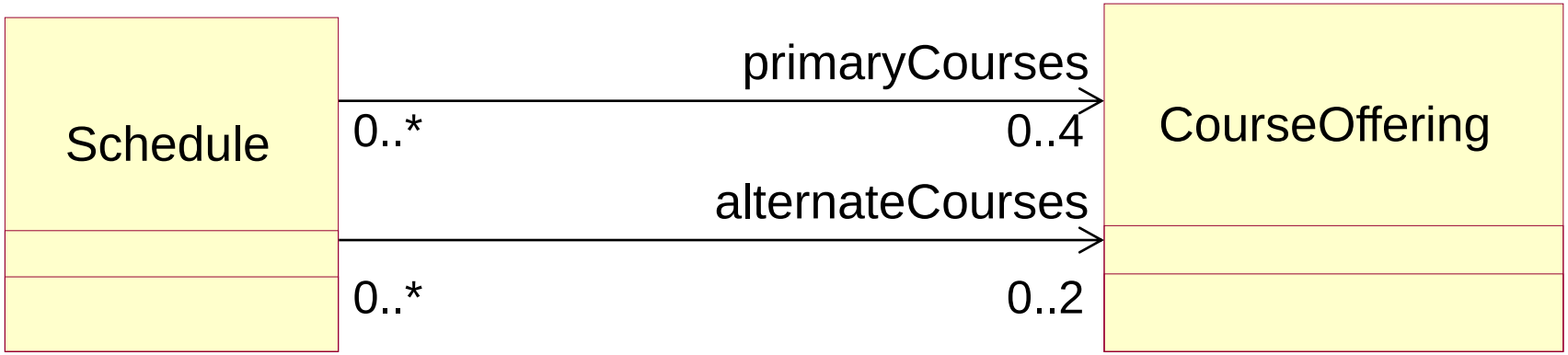
Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One	0..1
Specified Range	2..4

Example

Register for Courses Use Case:

The student selects four primary course offerings and two alternate course offerings from the list of available course offerings.

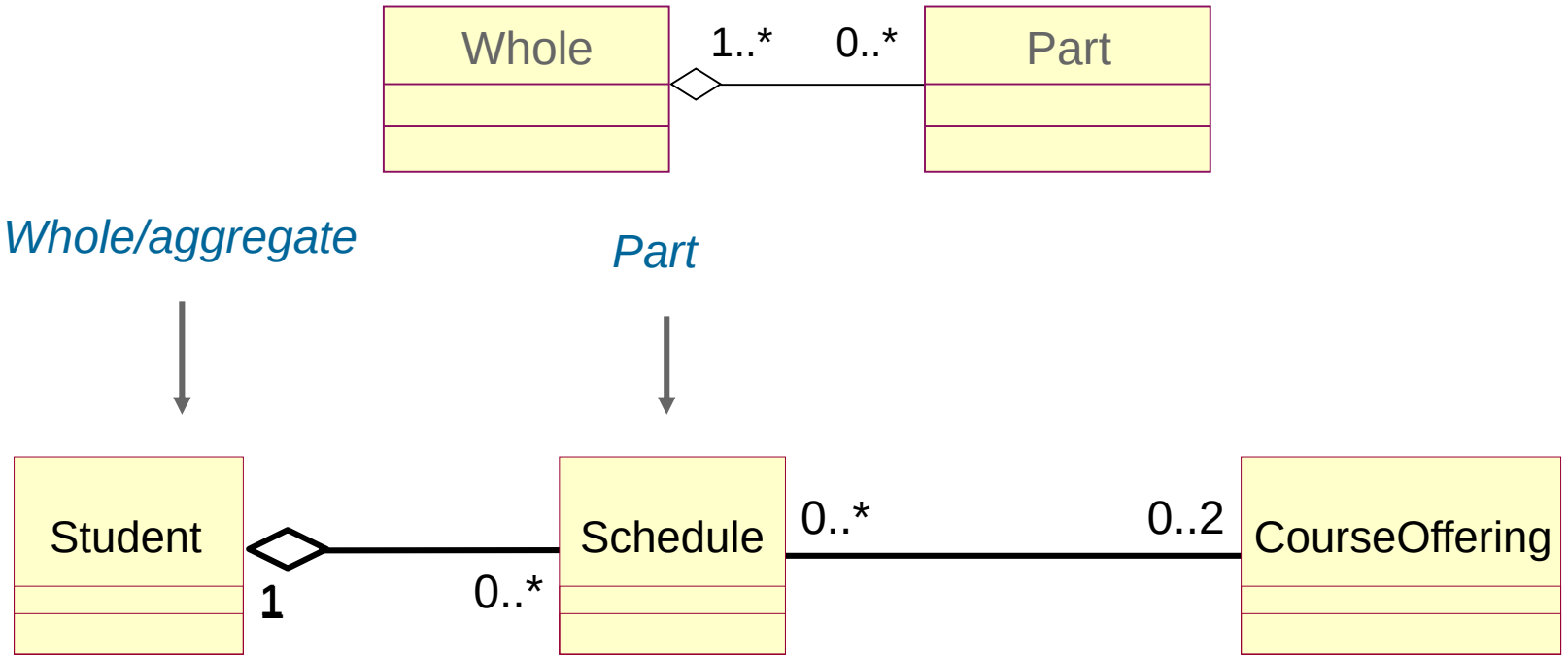
- One Schedule may have 4 primary CourseOfferings
- One Schedule may have 2 alternate CourseOfferings
- One CourseOffering may be associated with 0 or more Schedules



Aggregation

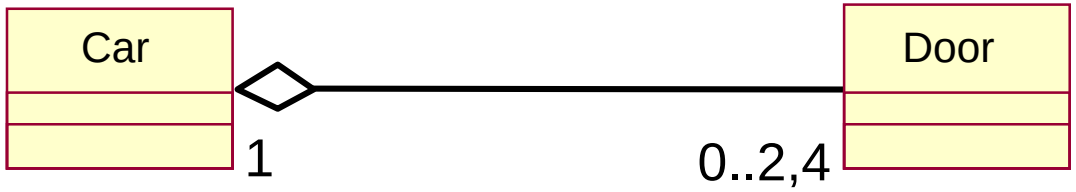
Aggregation

- Aggregation:
  - Class owns objects of another class
  - Class may share them
- One class represents a larger thing (the whole) which consists of smaller things (the parts)

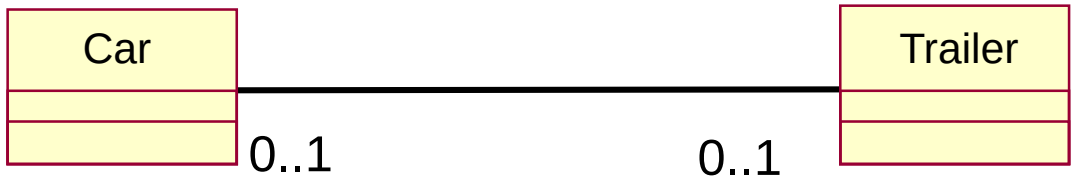


Association or Aggregation?

- If two objects are tightly bound by a whole-part relationship
  - The relationship is an aggregation.



- If two objects are usually considered as independent, although they are often linked
  - The relationship is an association.

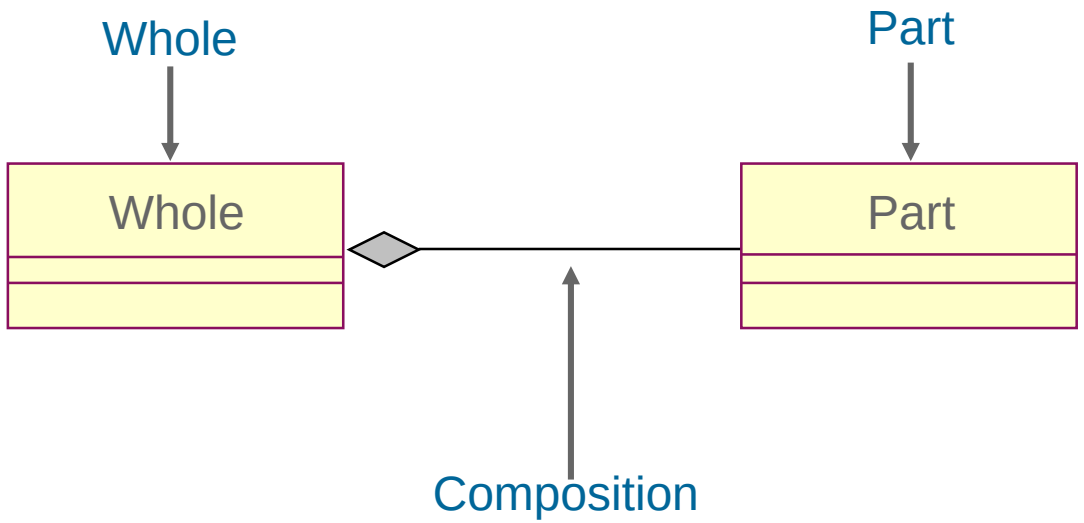


When in doubt: Use association

Composition

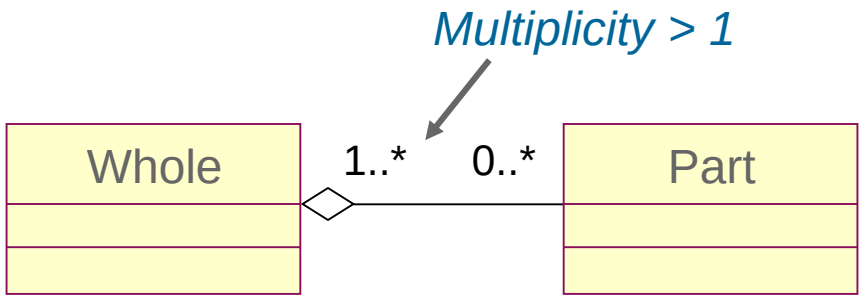
3. Composition

- Special form of aggregation
  - Strong ownership and coincident lifetimes
  - The parts cannot survive the whole/aggregate
- Instances of the parts are not shared with anything else
  - They are removed when the “whole” object is removed

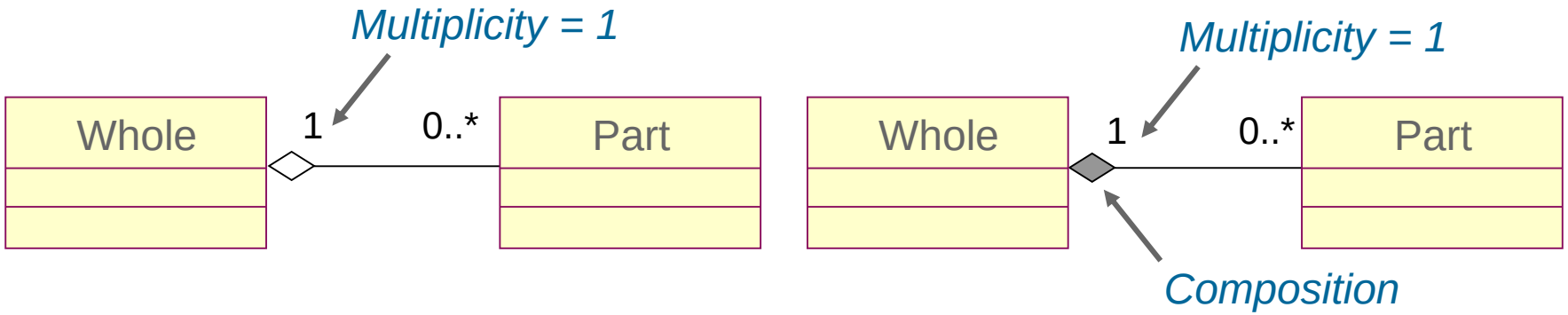


Aggregation: Shared vs. Non-shared

- Shared Aggregation



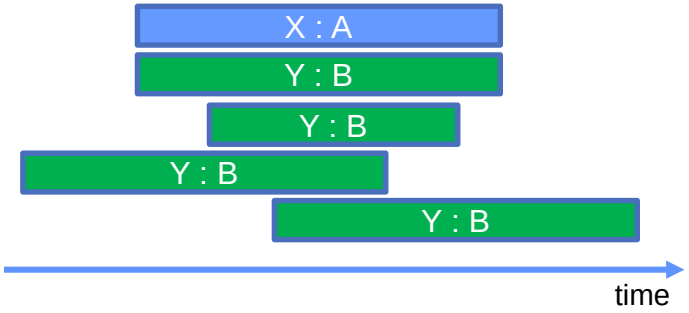
- Non-shared Aggregation

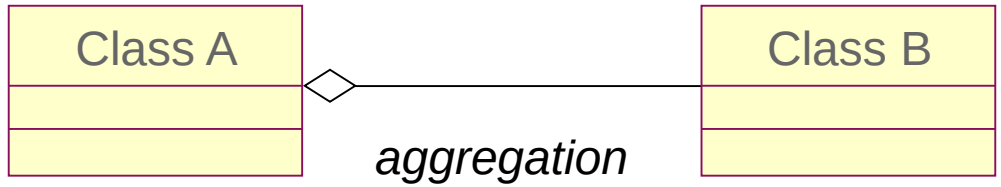


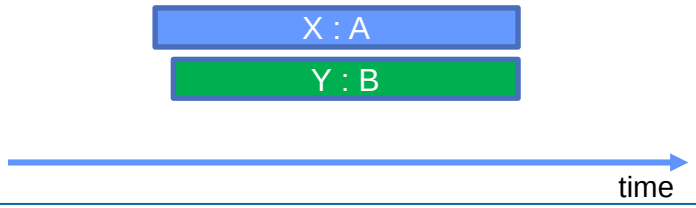
By definition, composition is non-shared aggregation

Aggregation or Composition?

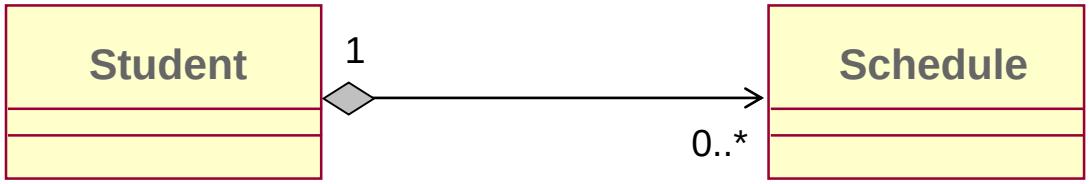
- Consideration
  - Lifetime of Class A = Lifetime of Class B?
  - Use composition if the whole and part must have coincident lifetimes.







Example: Composition

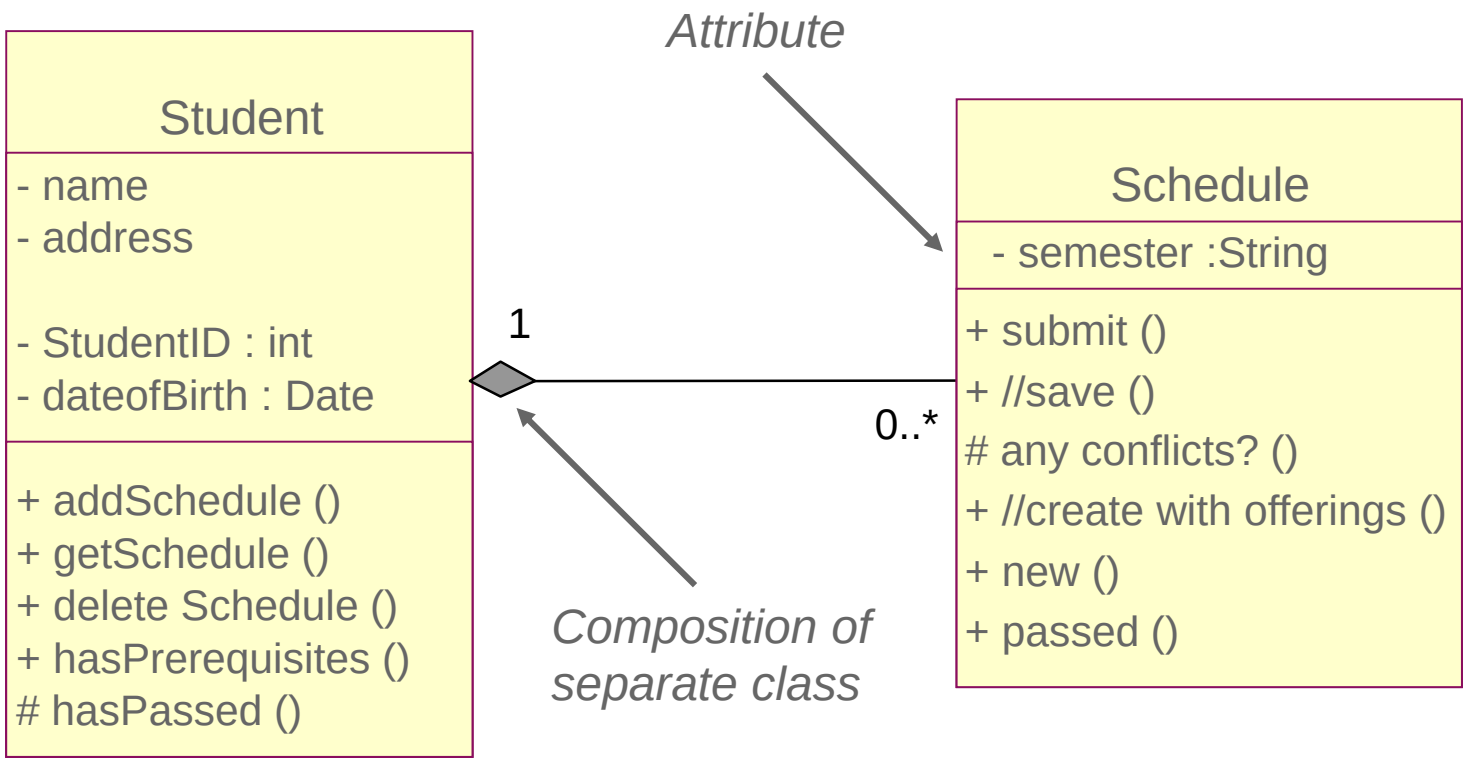


If Student is removed, then his Schedule is also removed.

Attributes Vs Composition

- Structure of an object can be modelled by attributes or composition
- When to use which?
- Use composition when
  - Properties need independent identities, referenced from a number of objects
  - Properties have a complex structure and properties of their own
  - Properties have complex behavior of their own
  - Properties have relationships of their own
- Otherwise use attributes

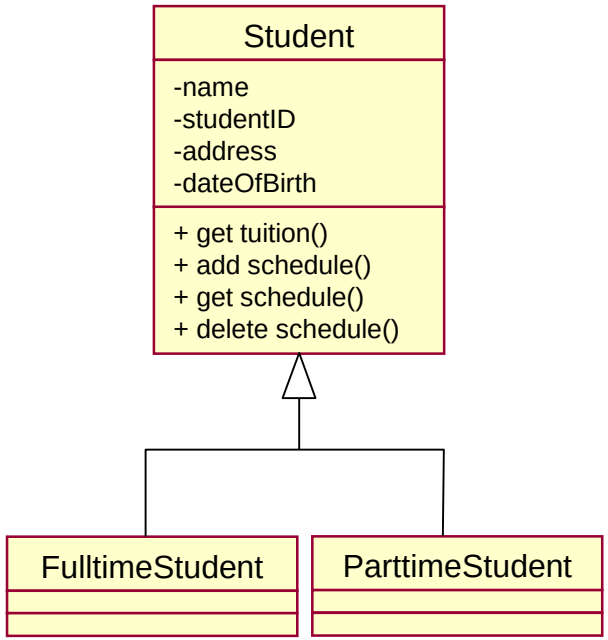
Example: Attributes vs. Composition



Generalisation

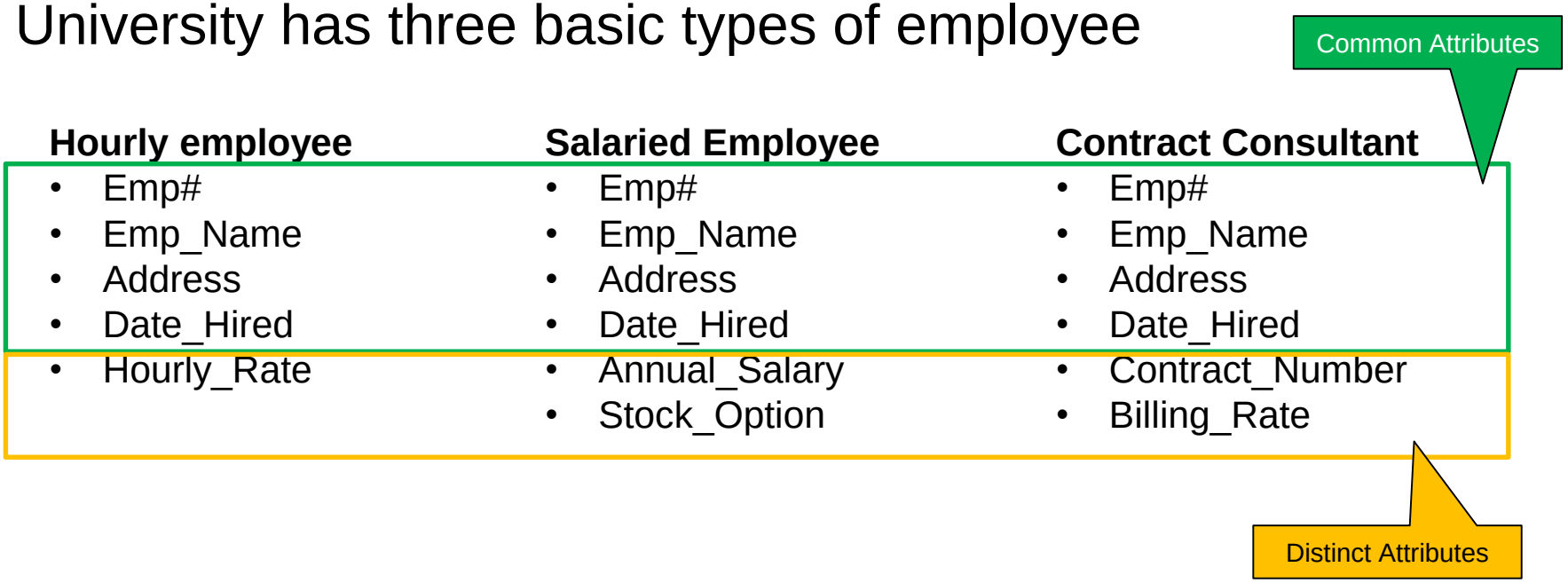
4. Generalization

- A relationship among classes where one class shares the structure, behavior, or both of one or more classes
- Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses
  - Single inheritance
  - Multiple inheritance
- Is an “is a kind of” relationship



Why Generalisation?

- University has three basic types of employee



All three types have some attributes in common and each has some distinct attribute(s).

Solution 1: Define a single class

Hourly employees with attributes:

Emp#, Emp_Name, Address,
Date_Hired, Hourly_Rate

Salaried employees with attributes:

Emp#, Emp_Name, Address,
Date_Hired, Annual_Salary,
Stock_Option

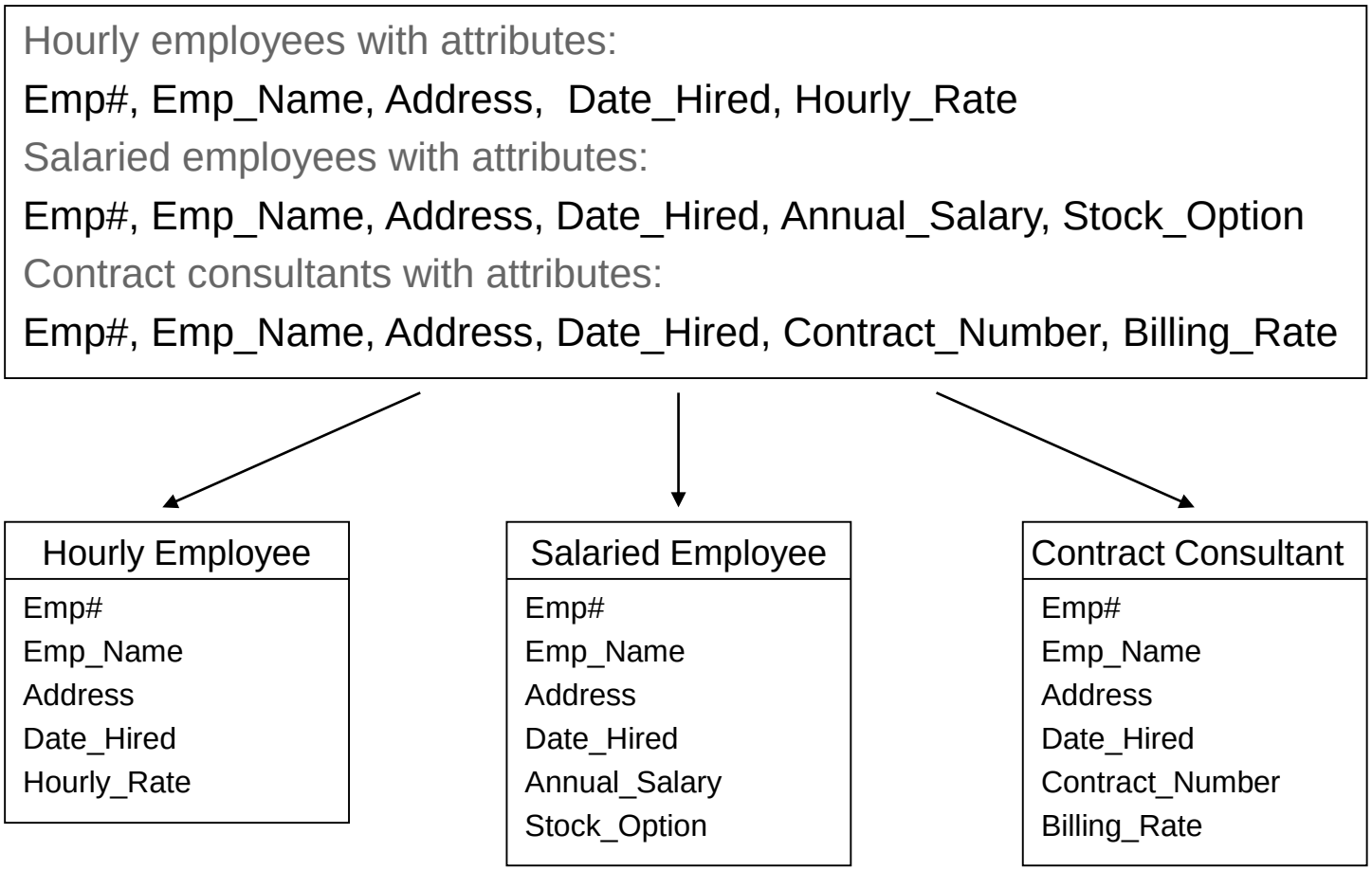
Contract consultants with attributes:

Emp#, Emp_Name, Address,
Date_Hired, Contract_Number,
Billing_Rate

Employee
Emp#
Emp_Name
Address
Date_Hired
Hourly_Rate
Annual_Salary
Stock_Option
Contract_No
Billing_Rate

This model does not show "three" types explicitly!

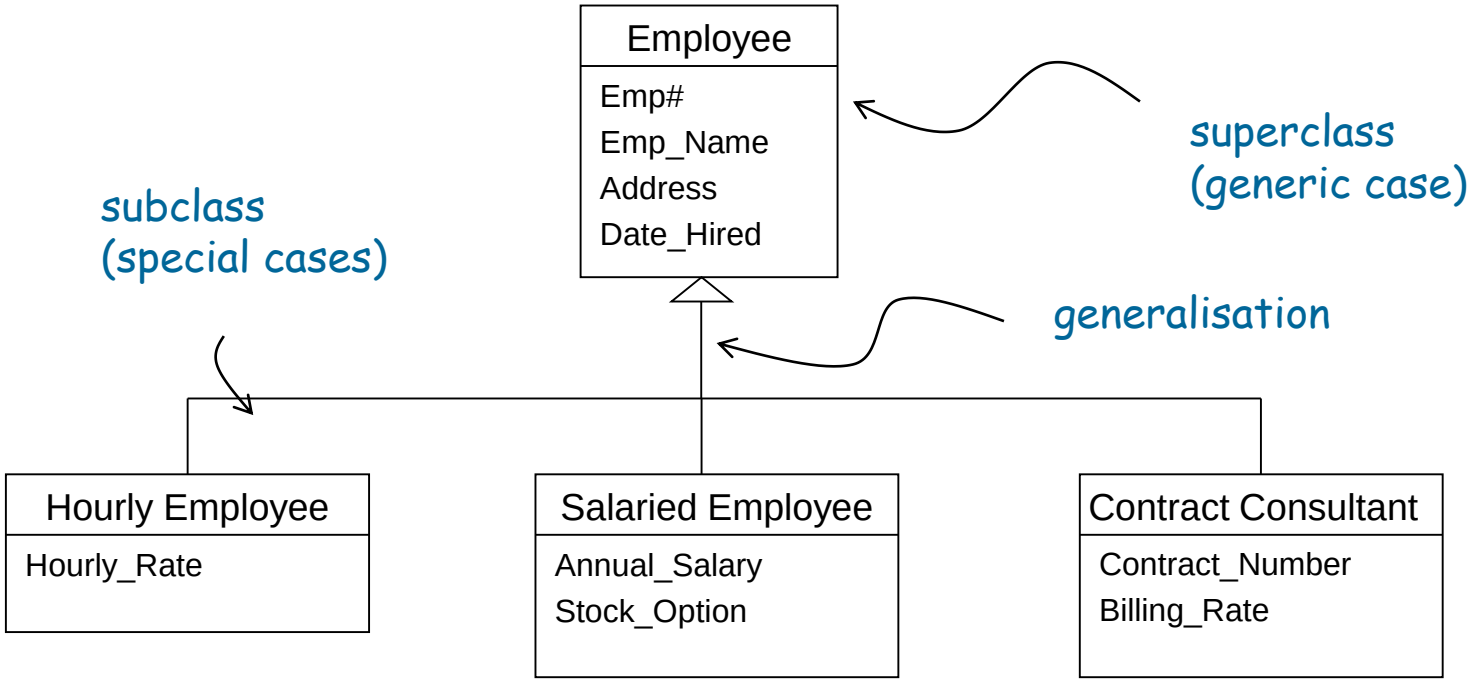
Solution 2: Define three classes



This model contains a lot of redundancy!

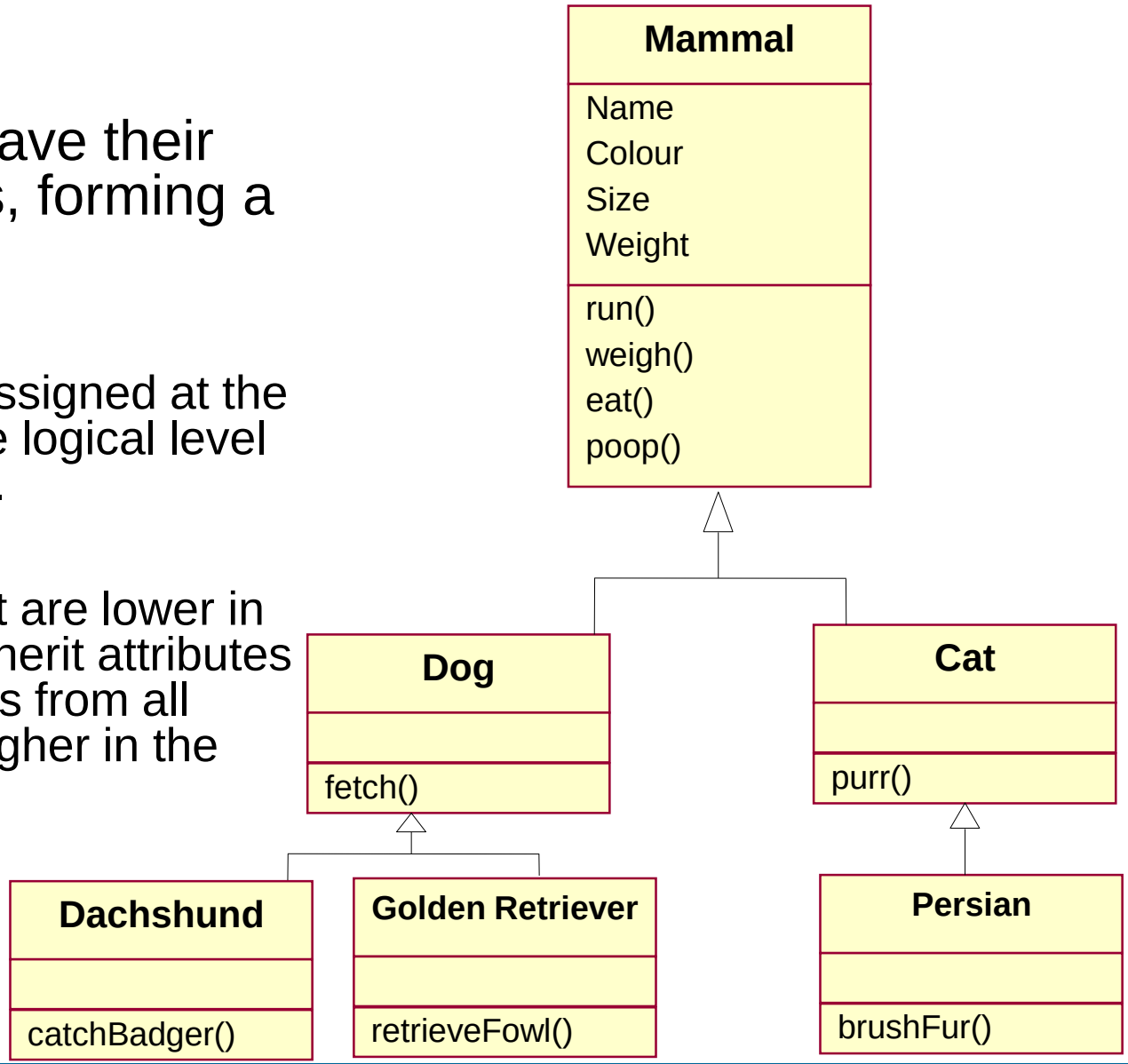
Generalisation

- Ideally, we would like to define a class to capture the common aspects of **Employees** and then make the three types of employee its special cases.



Class Hierarchy

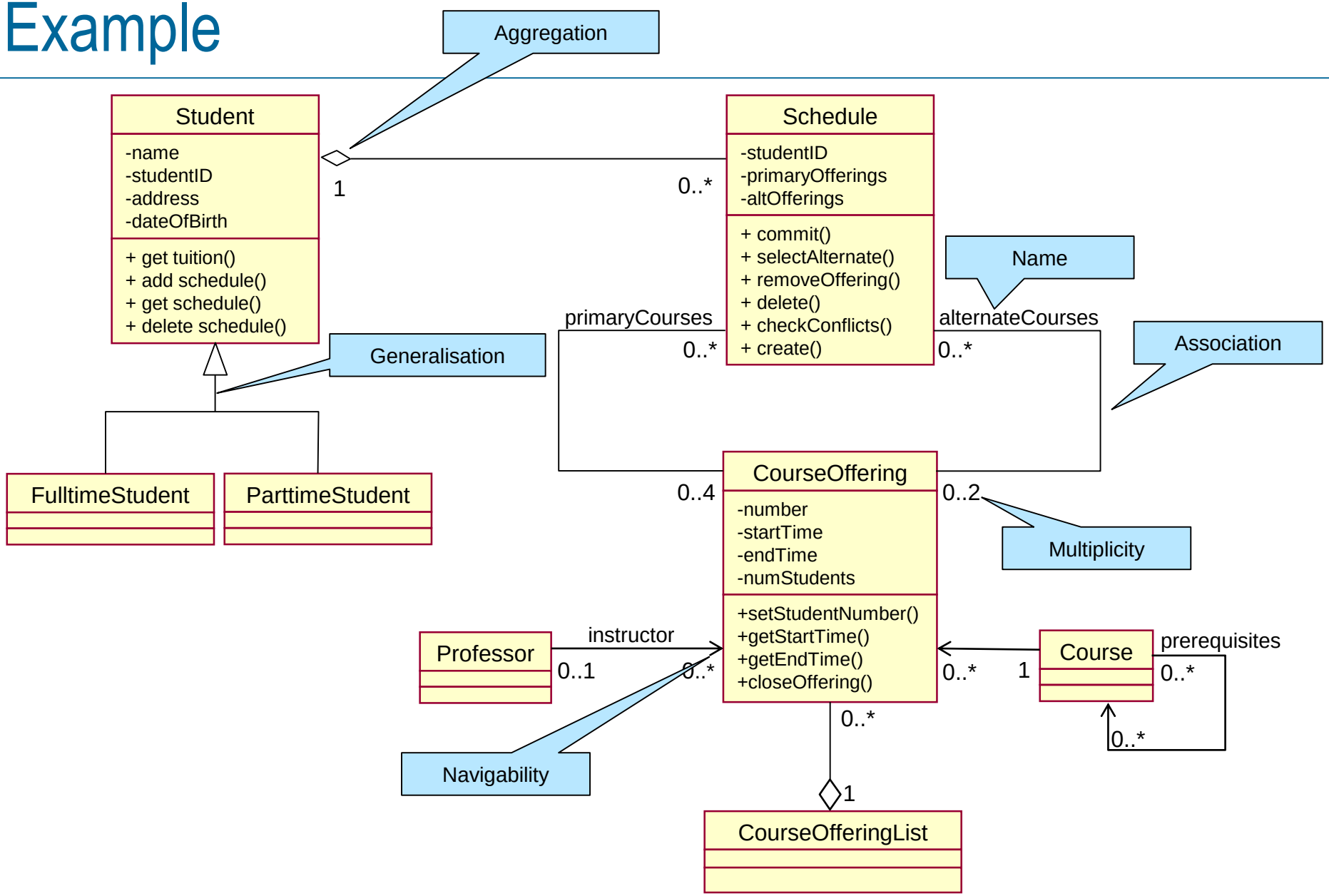
- It is possible for subclasses to have their own subclasses, forming a class hierarchy
  - Attributes are assigned at the highest possible logical level in the hierarchy.
  - Subclasses that are lower in the hierarchy inherit attributes and associations from all superclasses higher in the hierarchy.



Summary

- A class diagram specifies links and relationships between classes to build a model allowing us to see how different objects of the system interact
- Relationships:
  - Association
  - Aggregation
  - Composition
  - Generalisation

Example


Aggregation
Student
-name
-studentID
-address
-dateOfBirth
+ get tuition()
+ add schedule()
+ get schedule()
+ delete schedule()
1
0..*
Schedule
-studentID
-primaryOfferings
-altOfferings
+ commit()
+ selectAlternate()
+ removeOffering()
+ delete()
+ checkConflicts()
+ create()
Name
primaryCourses
0..*
alternateCourses
0..*
Association
Generalisation
FulltimeStudent
ParttimeStudent
0..4
CourseOffering
-number
-startTime
-endTime
-numStudents
+setStudentNumber()
+getStartTime()
+getEndTime()
+closeOffering()
0..2
Multiplicity
Professor
instructor
0..1
0..*
Navigability
0..*
1
CourseOfferingList
0..*
Course
prerequisites
0..*
0..*