COMP201 Software Engineering I
Lecture 22 – Class Diagrams

Lecturer: T. Carroll

Email: Thomas.Carroll2@Liverpool.ac.uk

Office: G.14

See Vital for all notes

Recap

Lecture 21 Recap

- Class diagrams represent the static (as opposed to the dynamic) nature of the system to be built.
- We discussed class attributes and operations
- We discussed associations and multiplicity.
- We looked at representations of generalization.
- We learned about CRC cards

Today

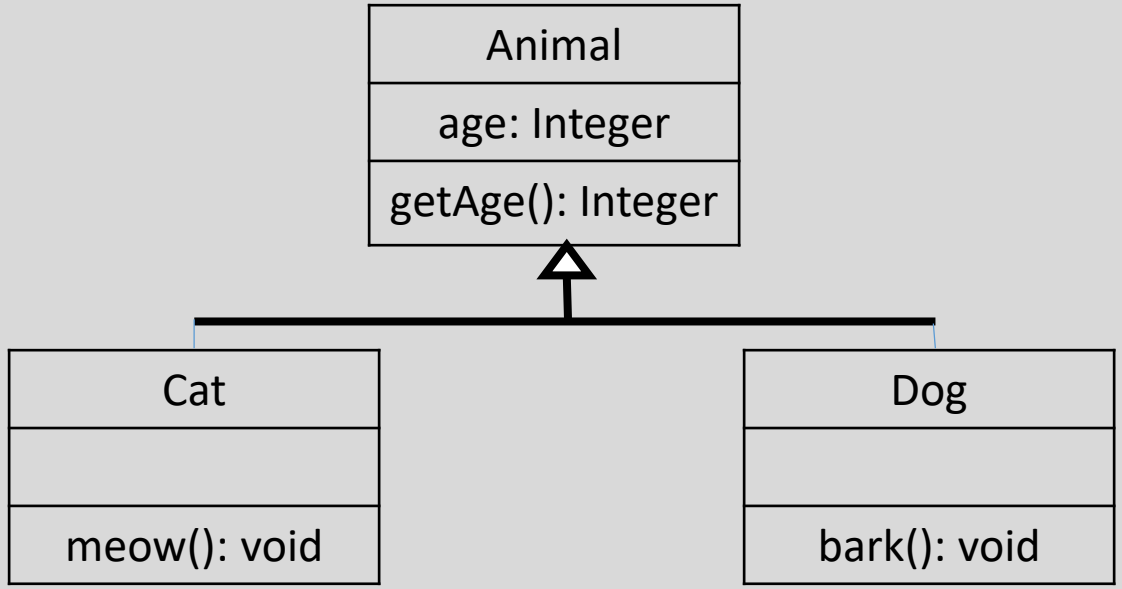
Lecture Outline

- Aggregation and composition
- Roles
- Navigability
- Qualified association
- Derived association
- Constraints
- Association classes
- Interfaces and abstract classes

Aggregation and Compostion

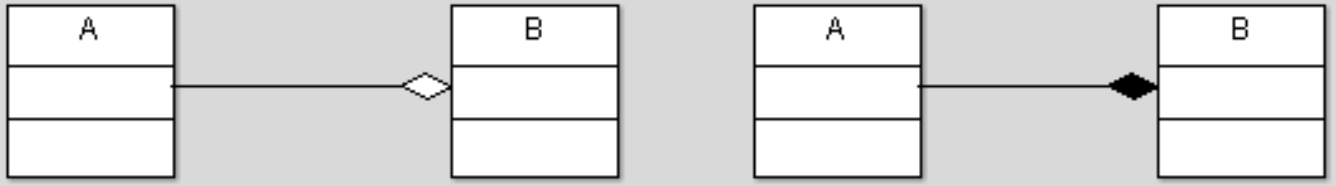
We already know some types of Association…

- Association
  - Shows a relationship exists
- Generalisation
  - Shows inheritance



Aggregation and Composition

- **Aggregation** and **composition** are kinds of association:
- They show an object of class A *is part of* an object of class B.

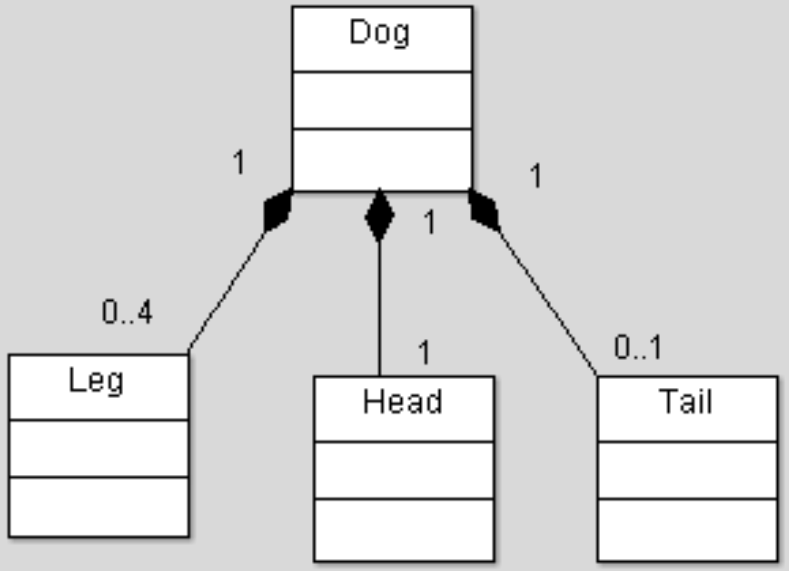


Aggregation

Composition

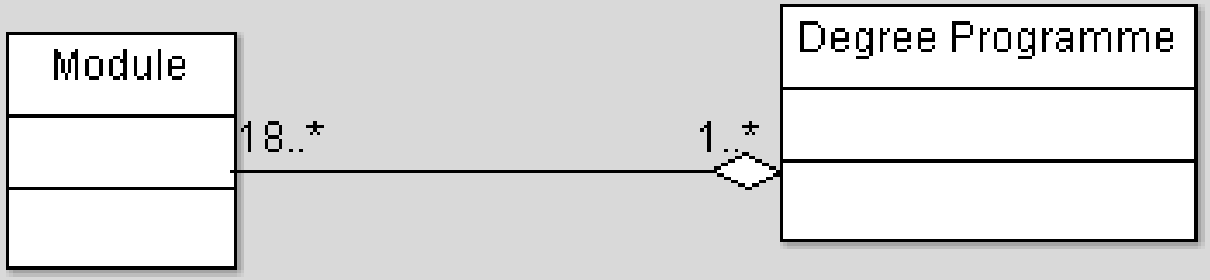
Composition – "…*can't exist without* …"

- In composition association, the parts ***can't exist without*** their associated owner
- If you delete/copy the owning object- the parts are copied/deleted too!
- The multiplicity at the whole end of a composition association must be 1 or 0..1
  - A part cannot be part of more than one whole by composition



Aggregation – *"…is part of…"*

- Denotes a *part-whole* relationship in UML
- The *part* is independent of the *whole*
- Delete the whole, and the part still exists!



Examples

- Should we use *composition* or *aggregation?*
  - The relationship between an **Employee** and a **Team**?
  - The relationship between a **Wheel** and a **Car**?
  - The relationship between an **Account** and a **Customer**?

Roles

- Associations are sometimes read in both directions
- Sometimes it is more readable to have separate names for the roles in either direction.

Association with no Navigability

- The diagram records that:
  - For each object of class *Passenger* there is one objects of class *Train* associated with the *Passenger*;
  - For each object of class *Train* there are some *Passenger* objects (number unspecified) associated with the *Train*.

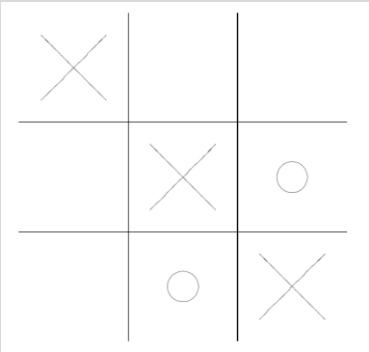
Navigability

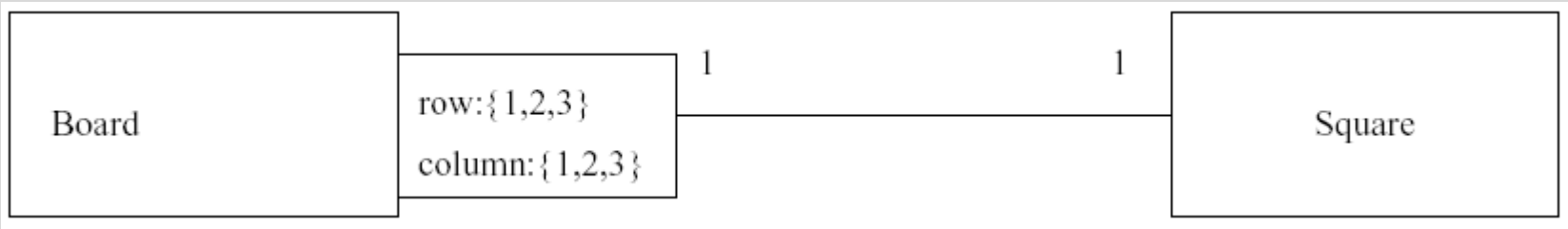
- An arrow restricts messages to be sent only in the direction of the arrow

- We say that *Passenger* knows about *Train*, but not vice versa.

Qualified Associations

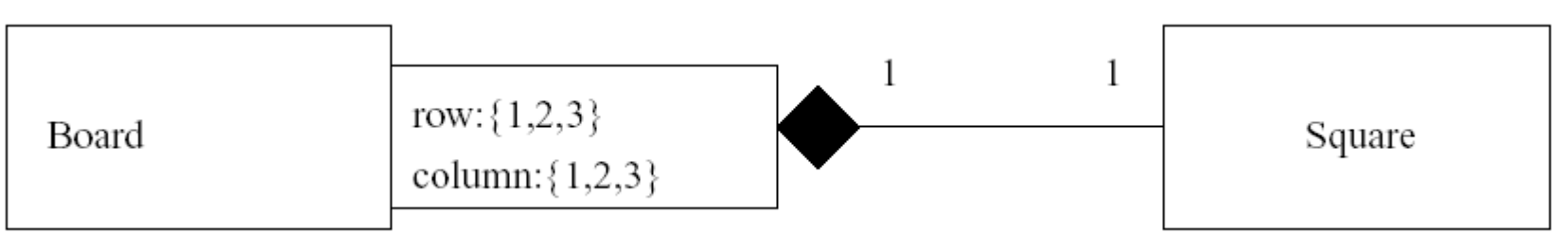
- A **qualified association** can give *finer detail*
- Consider a tic-tac-toe game board:
  - 1 board is made of 9 squares
  - Each square is identified by attributes row and column, each taking a value between 1 and 3





Qualified Composition

- We can combine qualified association with the other association notations
- Eg: combining with composition



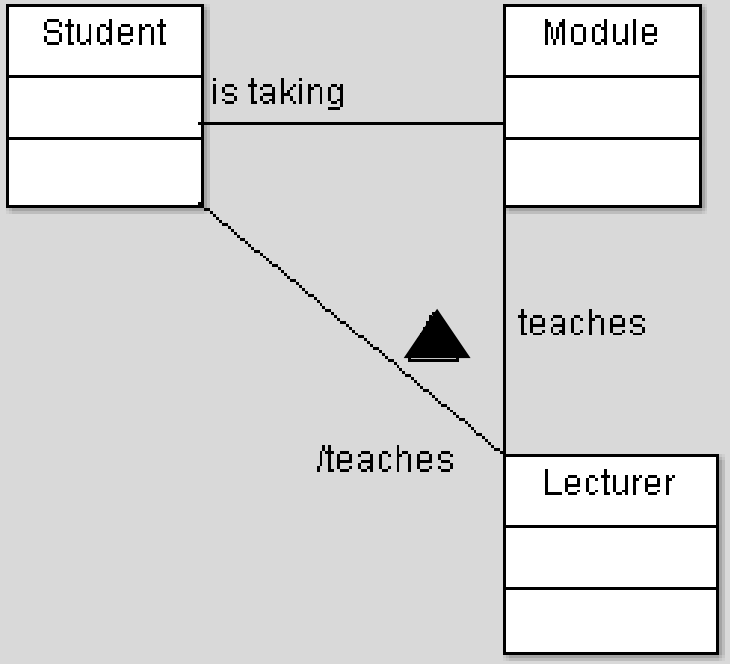
Derived Associations

- Imagine that a **student** *takes* a **module** and a **lecturer** *teaches* a **module**.
- Do we also have to record that a **lecturer** *teaches* **students**?
- Is it necessary?
- Is it already implied by the other two associations?
- UML has the concept of derived associations
- Derived associations emphasise to the designer that there is no need to implement this behaviour directly.

Derived Associations

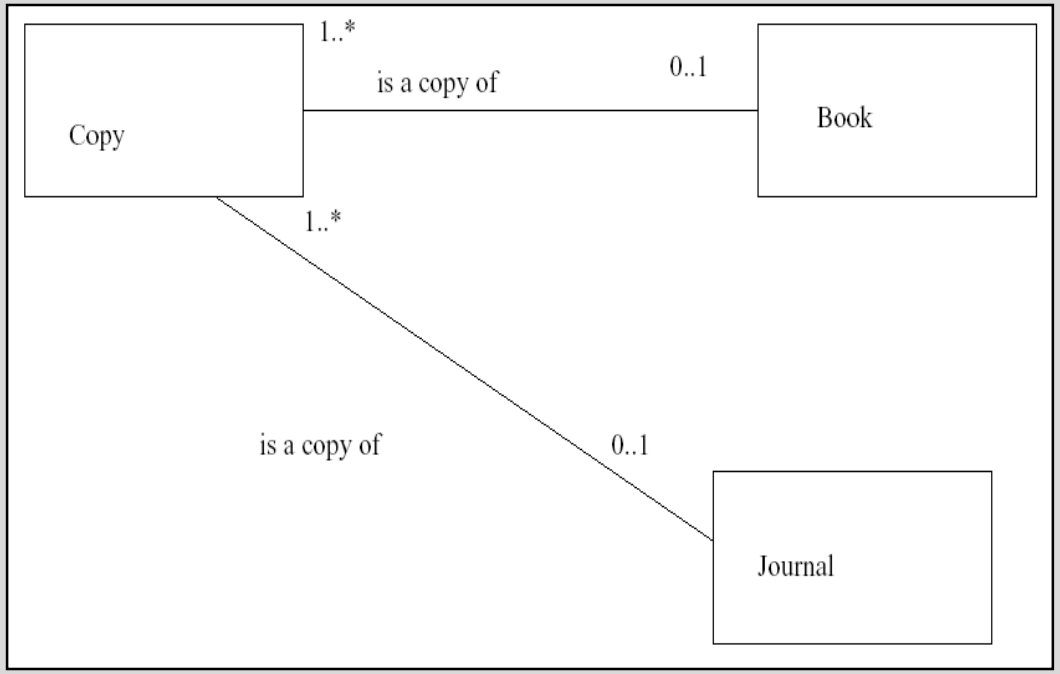
/name

- A derived association exists automatically once we have implemented the main association
- A derived association as shown using a slash in front of its name
- Filled triangles indicate which direction of the association the name describes.



XOR Constraints – Library example

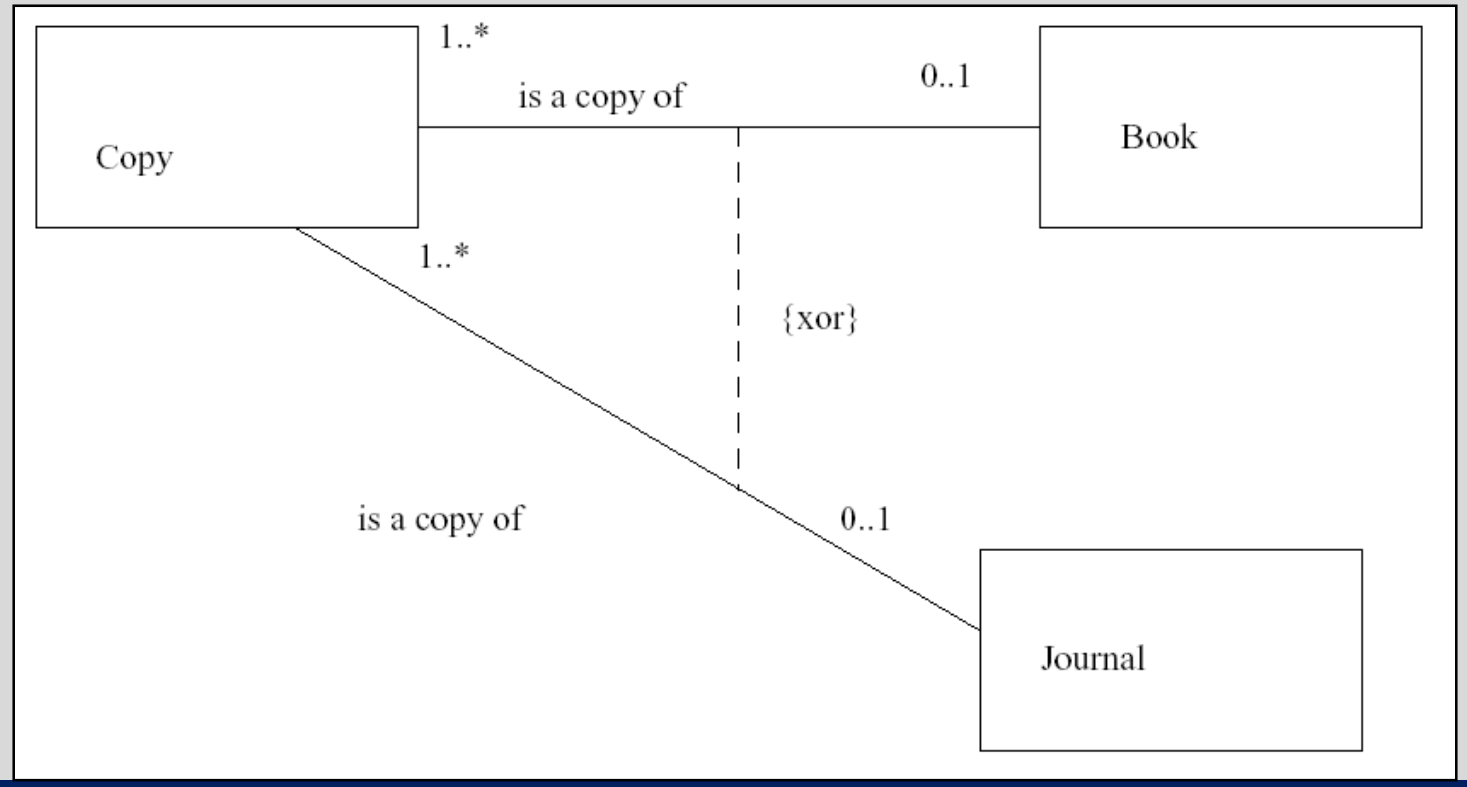
- Imagine that we know that a **Copy** is either a **Book** or a **Journal**.
- Having two associations: **Copy-Book** and **Copy-Journal** allows a **Copy** which is *both* a **Book** and a **Journal**



XOR Constraints

- To get round this problem, we use an **xor constraint**
- Xor =“exclusive or”.

Either A or B $A \oplus B$

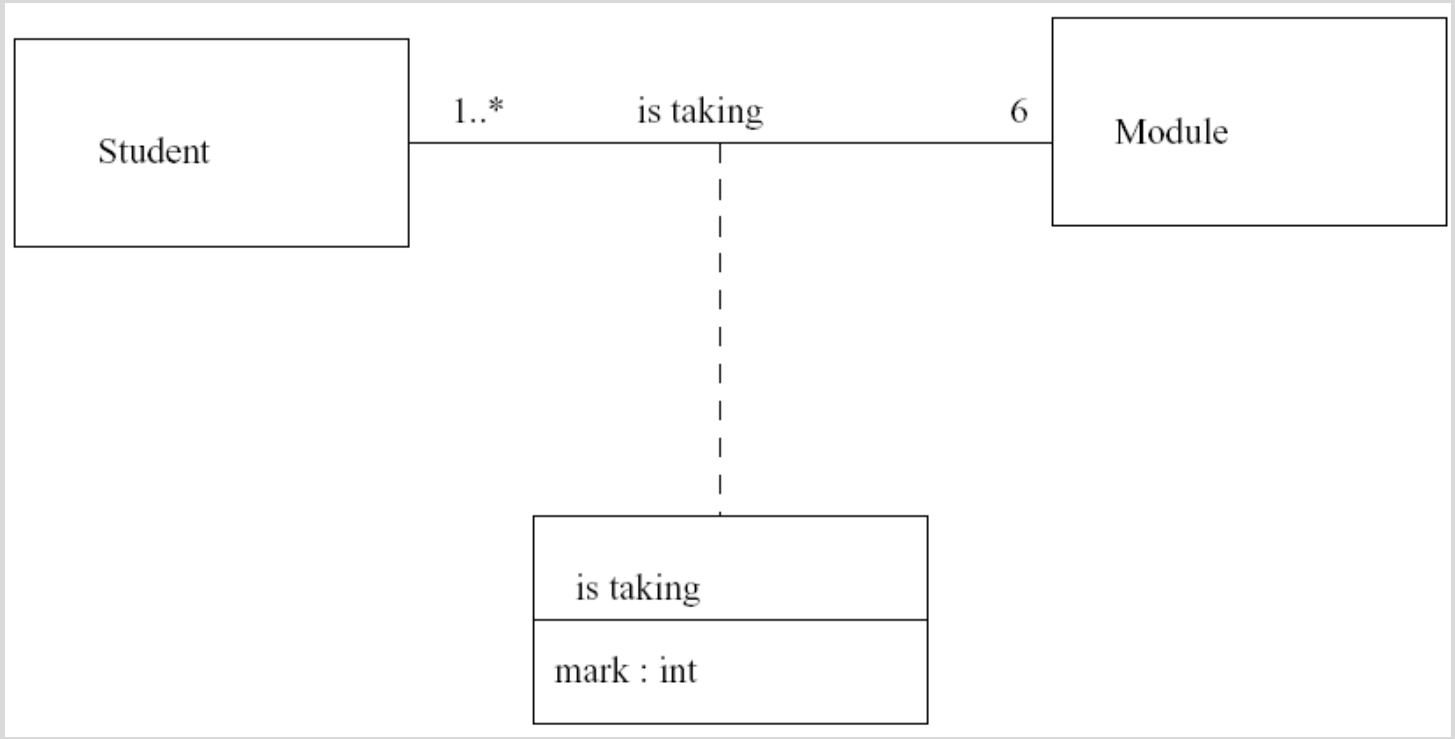


Association Classes

- Sometimes the **association** between classes may need attributes and operations.
- Consider the situation that a **Student** class is associated with a **Module** class.
  - Where should the students grade for that module be stored?
  - Is it a part of the **Student** class?
  - Is it part of the **Module** Class?
- The grade belongs to the **association** of these two classes..

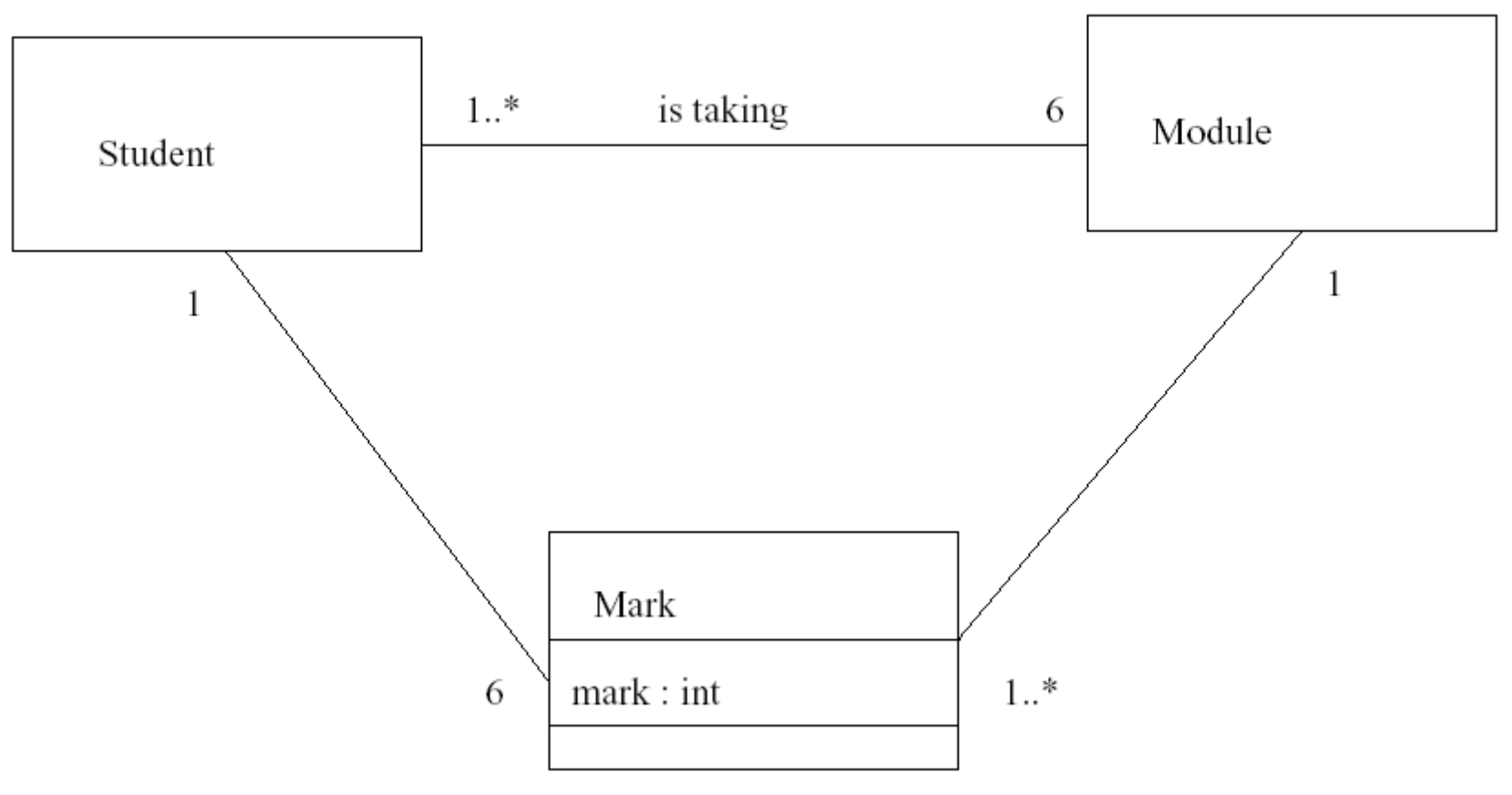
Association Classes

- An association class is both an **association** and a **class**.



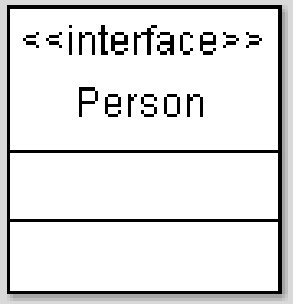
Avoiding an Association Class

- We can avoid the association class by making a third class….



Interfaces

- An interface specifies operations of some model element visible outside of the class.
- In UML2, an interface may specify some attributes and associations.
- All the elements of such an interface in a class diagram are public.
- The notation is to use a rectangle just like a class with “<<interface>>” string.



Abstract Classes

- An interface is similar to the idea of an abstract class
- An **abstract class** is one in which, for at least one operation, the implementation of that method is not defined.
- Thus the class **cannot be instantiated**.
- A class where *no method has an implementation* is essentially an **interface**
- Abstract class can be modelled in UML using *italic font* for the class name