CS4125
SYSTEMS ANALYSIS
SPRING SEMESTER 2010-2011

J.J. Collins
Dept of CSIS
University of Limerick

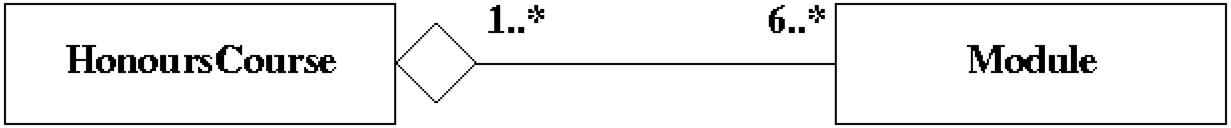
Lecture H (W5L1)

Analysis – Drawing the Class Diagram, Part 2

Associations: Aggregation and Composition

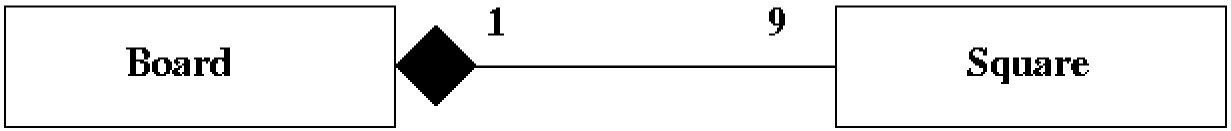
- Association: important real world relationships between classes.
- Just as an object is an instance of a class, a link is an instance of an association.

- Aggregation and composition are both associations that record that an object of one class is part of an object of another class.
- *Module* is part of an *Honours Course*.
- Open diamond denotes aggregation and records a part-whole relationship.



Associations: Aggregation and Composition

- With aggregation, an object can take part in other associations including aggregation but not composition.
- Convention: no need to name an aggregation.
- In a composition association, the whole strongly owns its part.
- A part cannot have an association with more than one object.

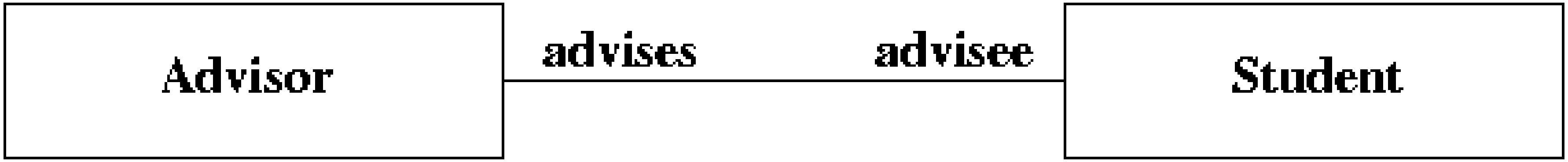


Associations: Aggregation and Composition

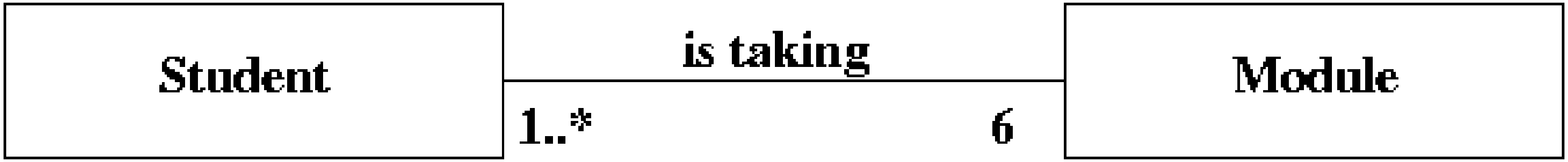
- If the whole object is copied or deleted, its part(s) are also copied and deleted.
- Multiplicity at the whole end must be 1 or 0..1.
- e.g. Each *Square* is part of exactly one *Board*.
- C++: you have aggregation if the whole contains a reference or pointer to the part.
- C++: you have composition if the whole contains the part by value. Why?
- What is the difference between aggregation and association?

Associations: Roles

- Sometimes, more readable to show each role that both objects play in an association.

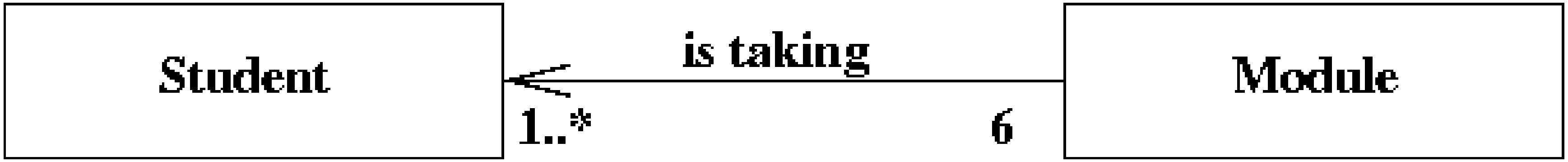


Associations: Multiplicity and Navigability



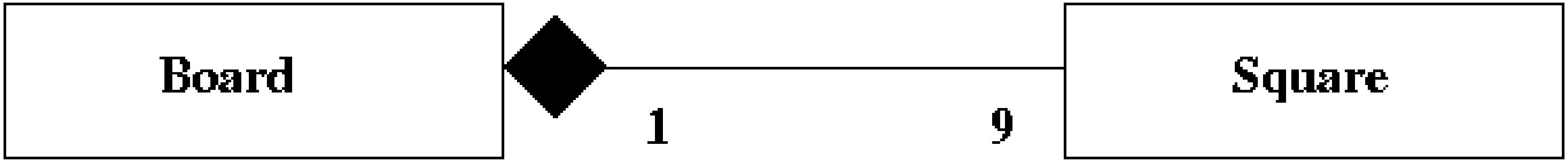
- Top Figure demonstrates that:
  - Each object of class *Student* is associated with 6 objects of class *Module*.
  - For each object of class *Module*, there are some *Student* objects associated with it.
- Does not show navigability i.e. should *Student* object be able to send messages to its associated *Module* object, or vice versa, or both.
- An arrow on one end of the association represents the direction of navigability.

Associations: Multiplicity and Navigability

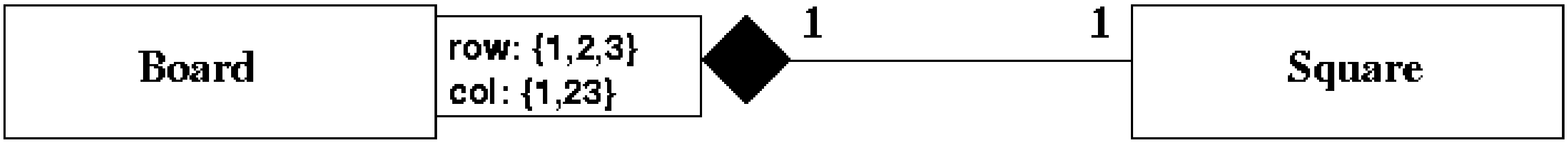


- Figure shows that *Module* knows about *Student*, but not vice versa.
- How. e.g. letting *Module* have an attribute that is a collection of student objects - *students: StudentCollection*.
- Why not use inheritance - taxonomic hierarchy?
- Downside: if class A knows about class B, then it is impossible to REUSE class A without class B.
- Should not introduce navigability unless absolutely necessary.

Associations: Qualified Associations



- Consider the game noughts and crosses. Plain composition (association) shown in figure between *Square* and *Board*.
- Does not convey concept that each square's identity is determined by 9 possible pairs of values to the attributes *row* and *col*.



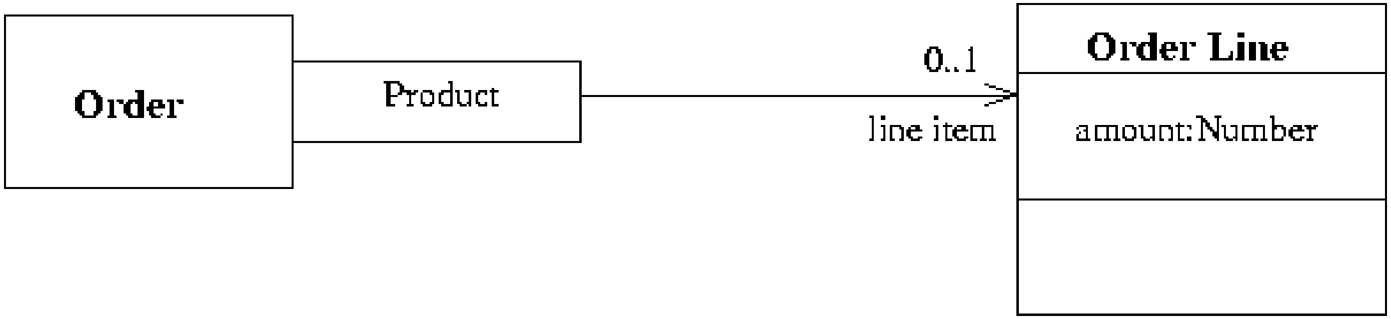
Associations: Qualified Associations

- UML equivalent of a programming construct known as associative array, hash table, dictionary, etc.

Class order …

Public OrderLine getLineItem(Product aProduct);

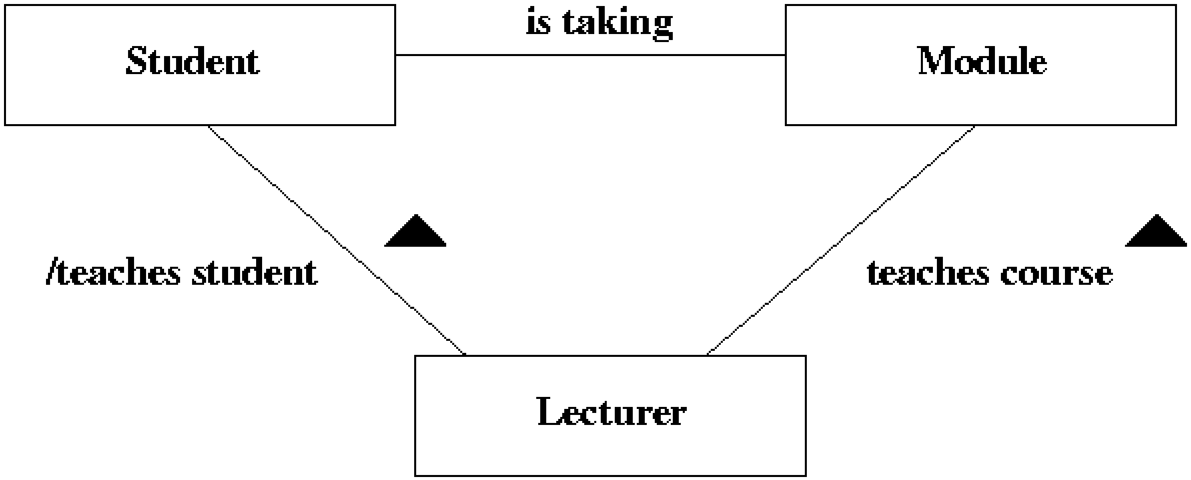
Public void addLineItem(Number amount, Product forProduct);



A Qualified Association

Associations: Derived Associations

- If *Student* is associated with *Module* and *Module* is associated with *Lecturer*, do we need to show an association "*teaches student*" between *Lecturer* and *Student*.
- UML option: show the association as a derived association. Exists automatically once the main associations have been implemented.
- Shown using / in front of its name, as in fig. 8.
- Black triangles can be used on any association name, and show which direction of the association the name describes.

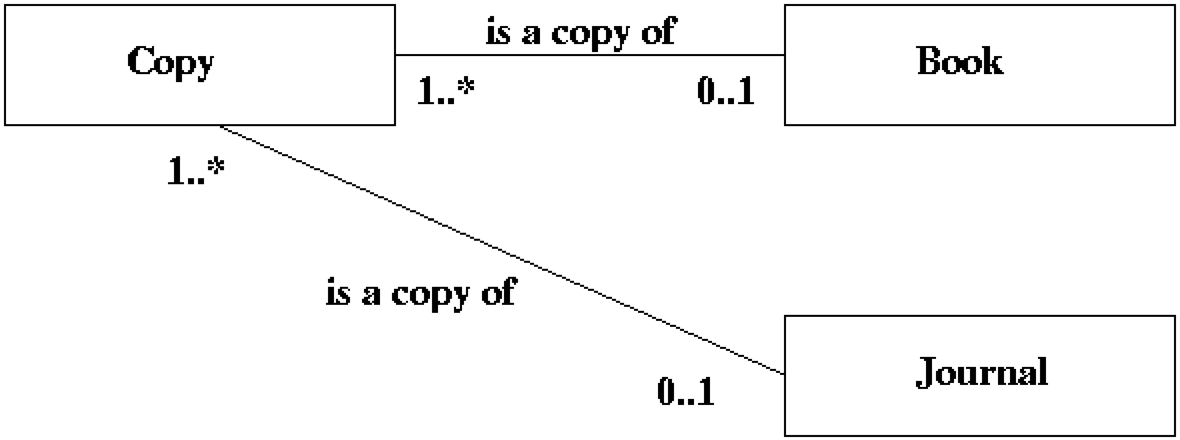


Associations: Constraints

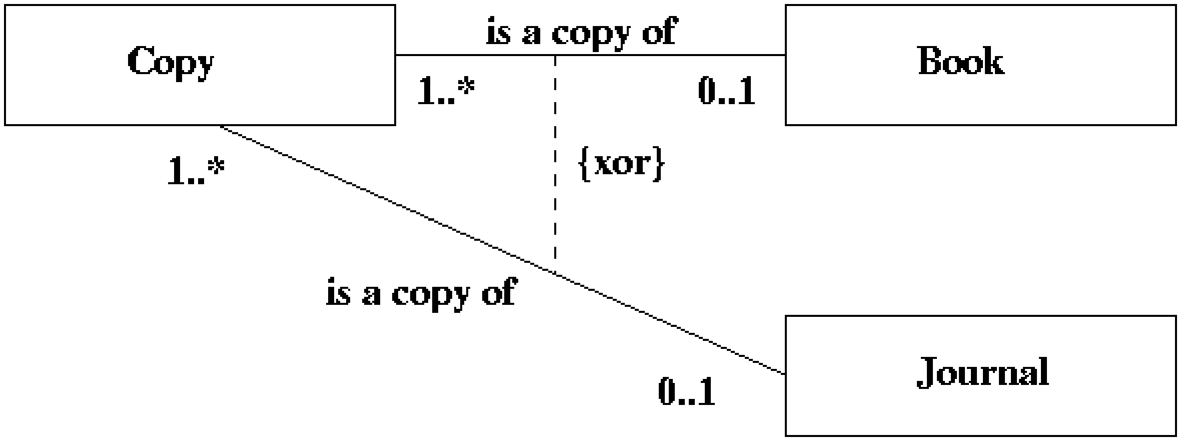
- A constraint is any condition that has to be satisfied by any correct implementation of the design.
- Express constraints as class invariants, written formally in Object Constraint Language (OCL), which UML has adopted.
- e.g. an invariant of class module
- {self.noOfStudents > 50 implies (not (self.room = S205))}
- Constraints may be useful when there is an exclusive or between two associations - an object takes part in exactly one of the associations.
- The XOR constraint not formalised in OCL, special predefined constraint in UML.

Associations: Constraints

- An under-constrained diagram



- Using the Xor constraint

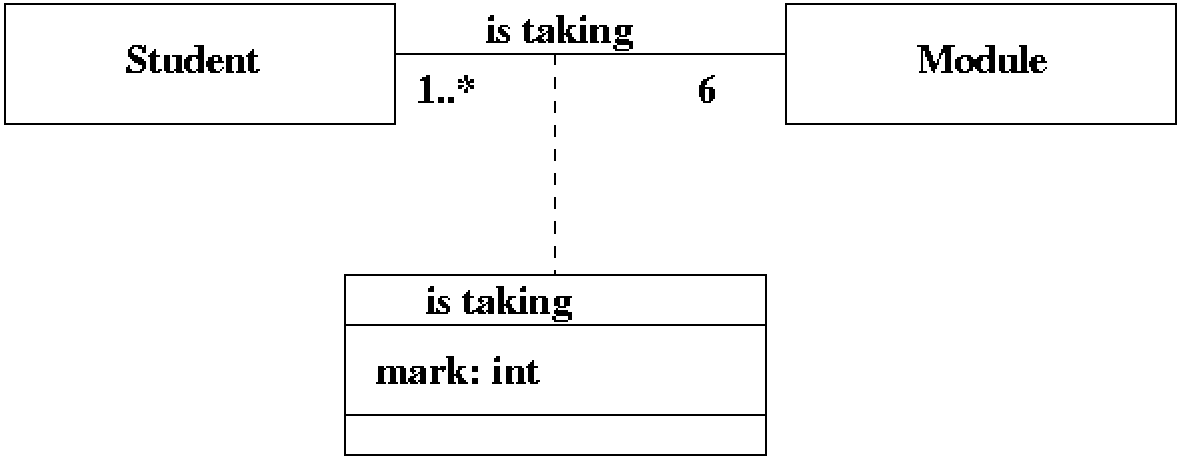


Associations: Constraints

- Constraints which constrain several model elements not contained within one class signal dependencies, which may hamper both maintenance and reuse - Ian Graham.
- OCL originated in Syntropy method developed by Cooks and Daniels. Further development by IBM as a business modelling language

Associations: Association Classes

- Consider *Student* and *Module*. The *marks* are connected with both objects.
- Treat the association between *Student* and *Module* as a class, with attributes and methods.
- The class icon and association line must have the same name, because they represent the same concept.
- Poses a problem, since associations normally have verb phrases as names, and classes have noun phrases as names.
- Could have associated a new class *Mark* with both objects



Defining Attributes and Operations

- Attributes
  - Visibility name: type multiplicity = default {property-string}
  - {property-string} – {ReadOnly}
- Operations
  - Visibility name (parameter-list): return-type {property-string}

10. Reading

- Stevens and Pooley: Chapters 5 and 6.