From UML to Relations

Revised by Gonçalo Gonçalves for ESIN

Original slides by Carla Teixeira Lopes for Bases de Dados (Mestrado Integrado em Engenharia Informática e Computação, FEUP)

Based on Jennifer Widom slides

UML key concepts

Classes

Associations

Association Classes

Generalizations

Composition & Aggregation

Constraints

Derived Elements

Classes

Every class becomes a relation

Student
sid
sname
grade

College
cname
state
enrollment

Classes

Every class becomes a relation

Student (sid, sname, grade)

College (cname, state, enrollment)

UML key concepts

~~Classes~~

Associations

Association Classes

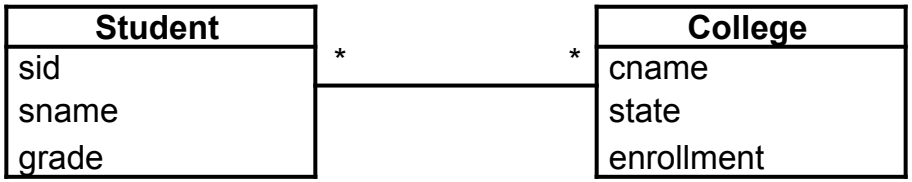
Generalizations

Composition & Aggregation

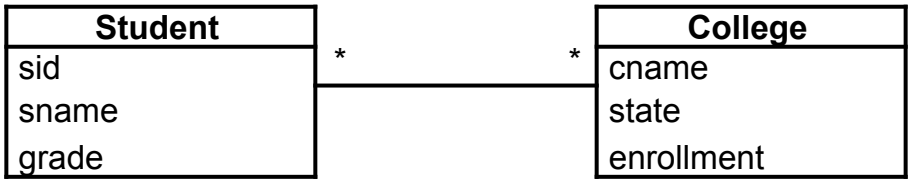
Constraints

Derived Elements

Many-to-many associations



Many-to-many associations



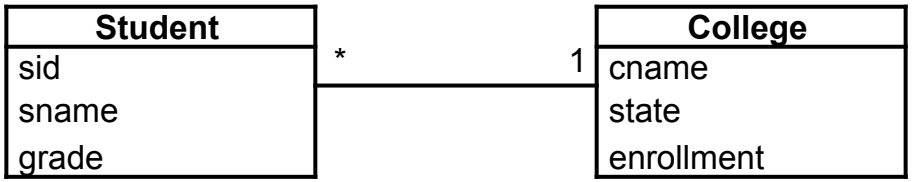
Add a new relation with foreign keys to each side:

Student (sid, sname, grade)

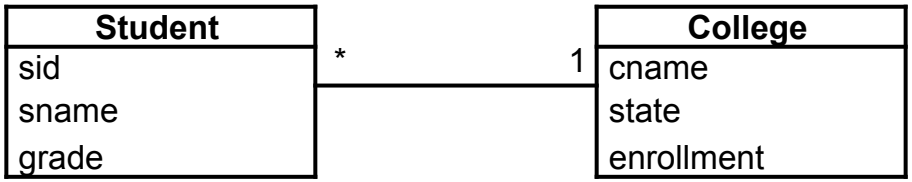
College (cname, state, enrollment)

Application (sid->Student, cname->College)

Many-to-one associations



Many-to-one associations

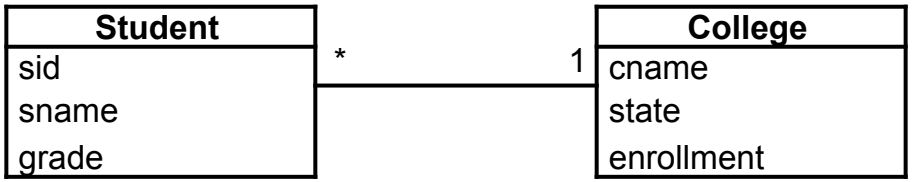


OPTION 1: Add a foreign key in the many (*) side of the relationship pointing to the relation on the one (1) side:

Student (sid, sname, grade, cname->College)

College (cname, state, enrollment)

Many-to-one associations



OPTION 2: Add a new relation with foreign keys to both sides: (the foreign key to the many side is also the primary key)

Student (sid, sname, grade)

College (cname, state, enrollment)

Application (sid->Student, cname->College)

Many-to-one associations

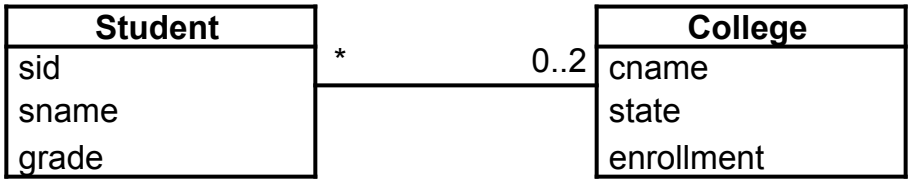
OPTION 1: Add a foreign key in the many (*) side of the relationship pointing to the relation on the one (1) side

- Most common
- Less relations in the schema
- Increased performance due to a smaller number of relations

OPTION 2: Add a new relation with foreign keys to both sides (the foreign key to the many side is also the primary key)

- Increased rigour of the schema
- Increased extensibility

Question



Suppose we had 0..2 on the right-hand side, so students can apply to up to 2 colleges.

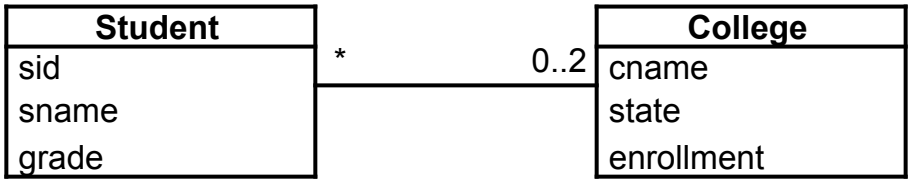
This means that the association is now a many-to-many association and so we can convert it as we saw before:

Student (sid, sname, grade)

College (cname, state, enrollment)

Application (sid->Student, cname->College)

Question



Suppose we had 0..2 on the right-hand side, so students can apply to up to 2 colleges.

This means that the association is now a many-to-many association and so we can convert it as we saw before:

Student (sid, sname, grade)

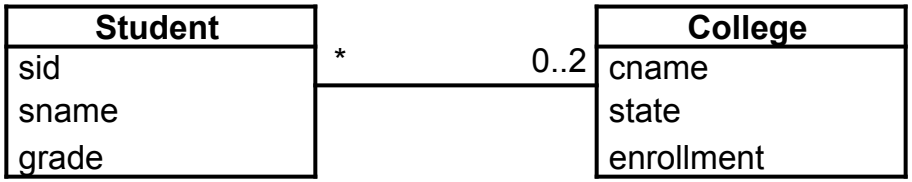
College (cname, state, enrollment)

Application (sid->Student, cname->College)

Is there a way to avoid having to create a separate Application relation?

- Yes, there is a way.
- No, if it's not 0..1 or 1..1 then Application is required.

Question

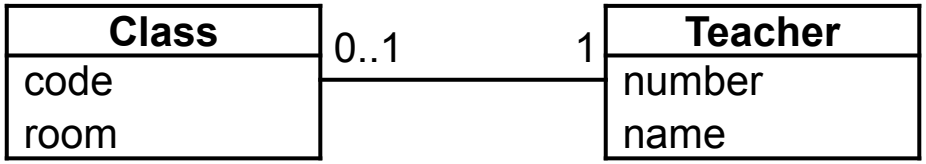


Yes, there is a way:

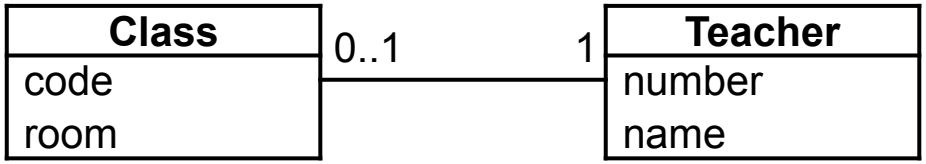
Student (sid, sname, grade, cname1->College, cname2->College)

College (cname, state, enrollment)

One-to-one associations



One-to-one associations



Add a foreign key in one of the relations to the other:

Class (code, room, teacher->Teacher)
Teacher (number, name)

For efficiency, the foreign key should be in the relation that is expected to have less tuples

Add a unique key constraint for the foreign key

UML key concepts

~~Classes~~

~~Associations~~

Association Classes

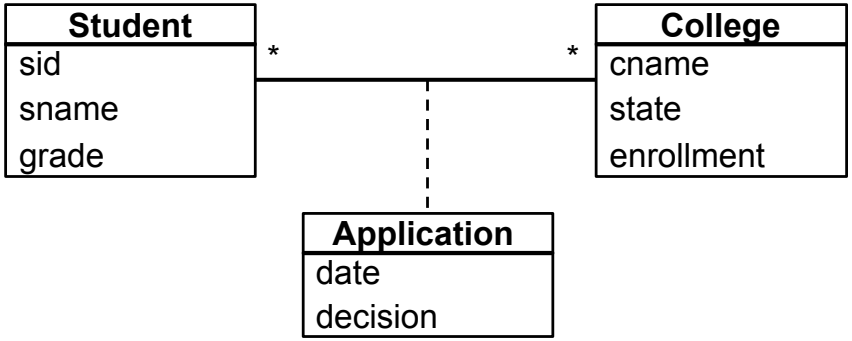
Generalizations

Composition & Aggregation

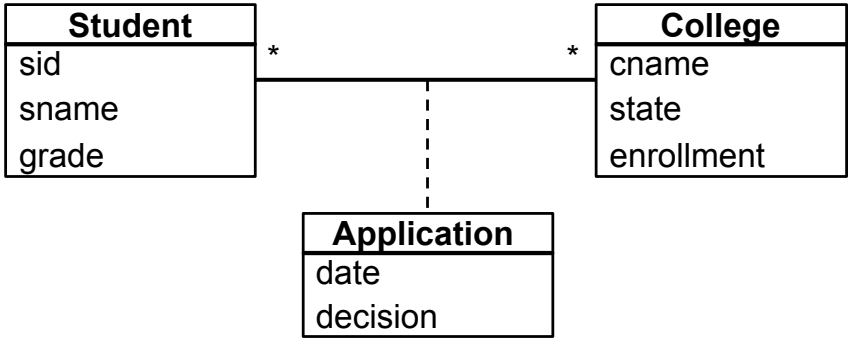
Constraints

Derived Elements

Association classes



Association classes



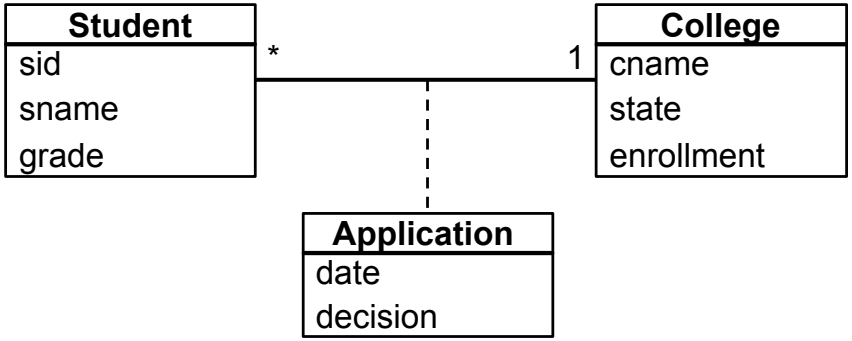
Same as a regular many-to-many association, but also add the specific attributes of the association class:

Student (sid, sname, grade)

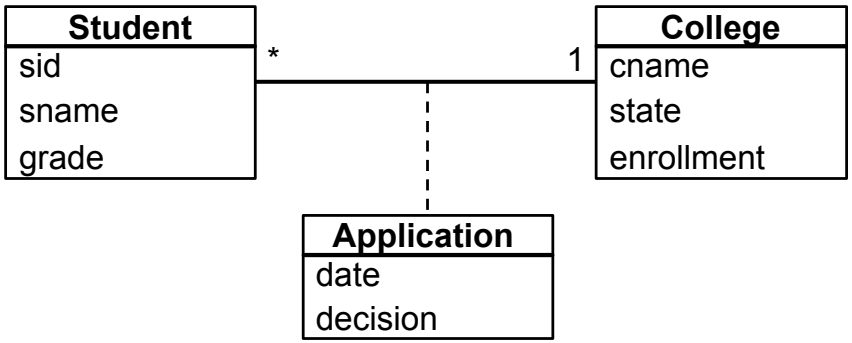
College (cname, state, enrollment)

Application (sid->Student, cname->College, date, decision)

Association classes



Association classes



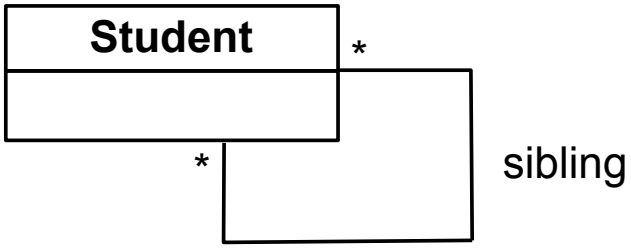
Same as OPTION 2 of the many-to-one association, but also add the specific attributes of the association class:

Student (sid, sname, grade)

College (cname, state, enrollment)

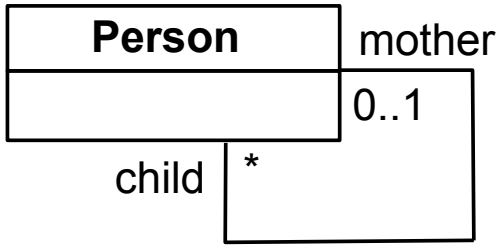
Application (sid->Student, cname->College, date, decision)

Self associations



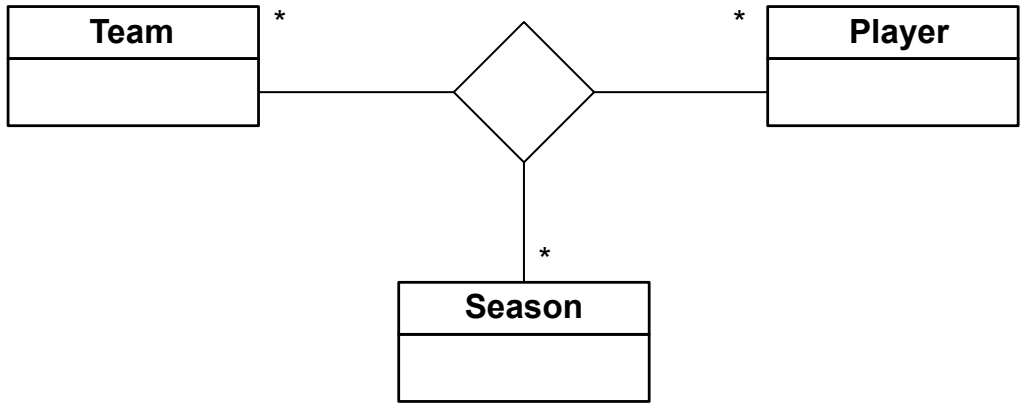
Student (sid, ...)
Sibling (sid1->Student, sid2->Student)

Self associations

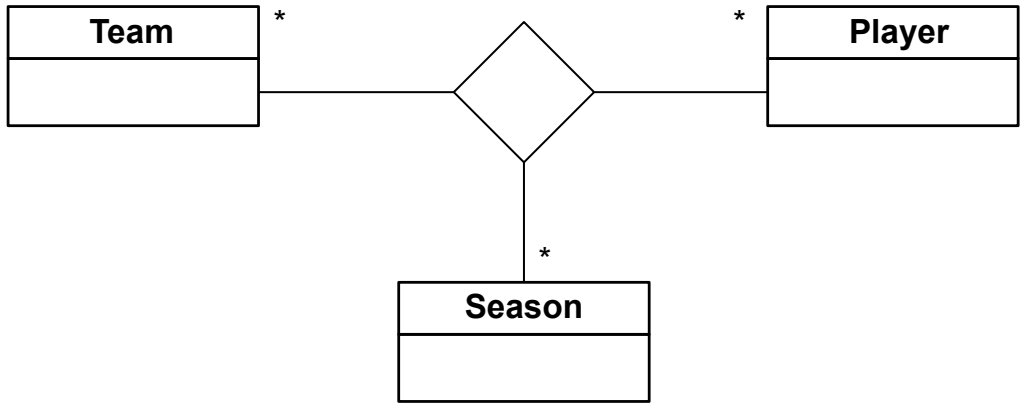


Person (id, …)
Relationship (mother->Person, child->Person)

Associations n-ary



Associations n-ary



Relation with key to each side:

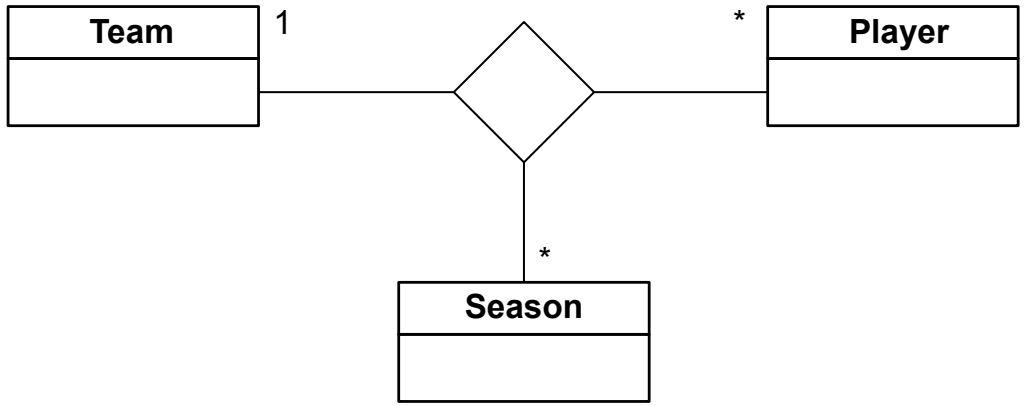
Team (id, …)

Player (id, …)

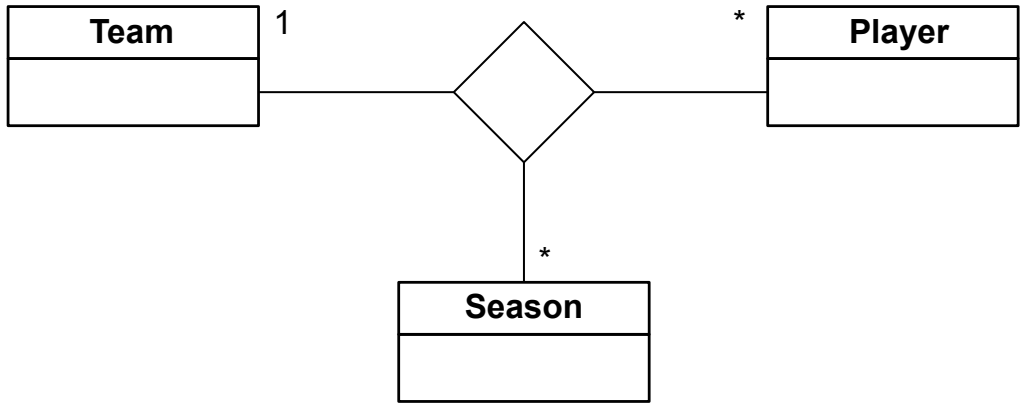
Season (id, …)

PlayerSeasonTeam (playerID->Player, seasonID->Season, teamID->Team)

Associations n-ary



Associations n-ary



Relation with key to each side:

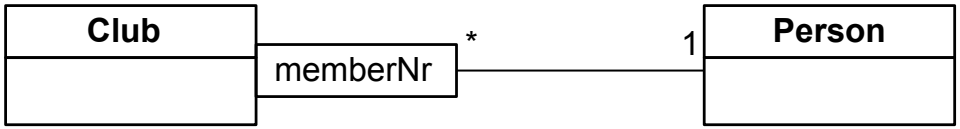
Team (id, …)

Player (id, …)

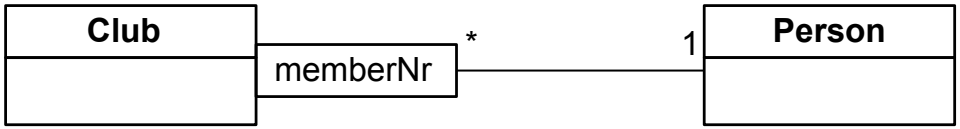
Season (id, …)

PlayerSeasonTeam (playerID->Player, seasonID->Season, teamID->Team)

Qualified associations



Qualified associations



Same as an association class, but with an extra unique constraint:

Club (id, …)

Person (id, …)

Membership (clubID->Club, personID->Person, memberNr)

UNIQUE(clubID, memberNr)

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

Generalizations

Composition & Aggregation

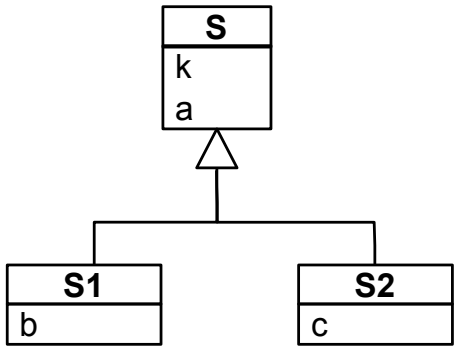
Constraints

Derived Elements

Generalizations

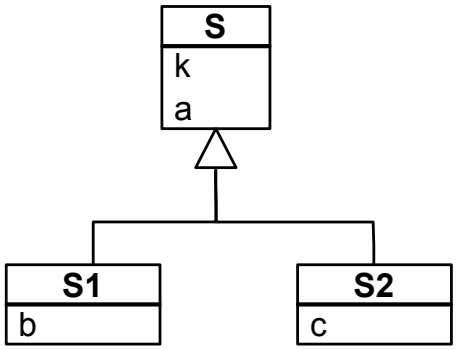
3 conversion strategies

- E/R style
- Object-oriented
- Use nulls

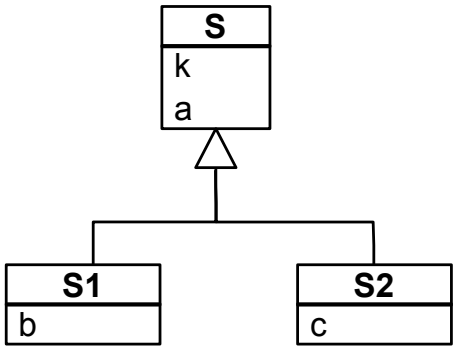


Best conversion may depend on the properties of the generalization

Generalizations – E/R style



Generalizations – E/R style



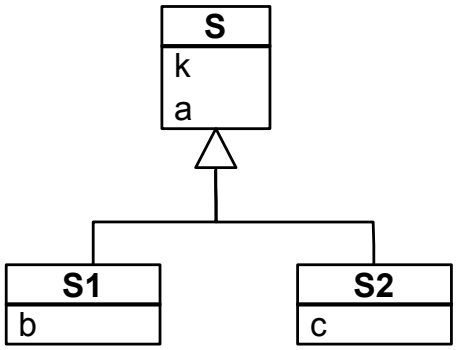
S (k, a)
S1 (k->S, b)
S2 (k->S, c)

A relation per each class

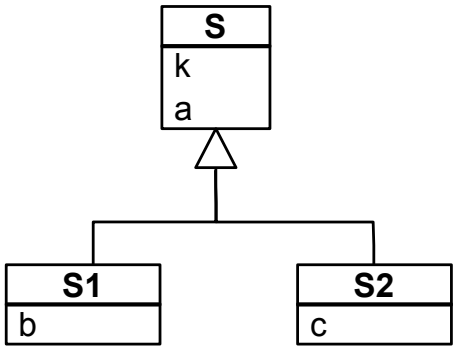
Subclass relations contain key to superclass + specialized attributes

Good for overlapping generalizations with a large number of subclasses

Generalizations – Object-oriented



Generalizations – Object-oriented



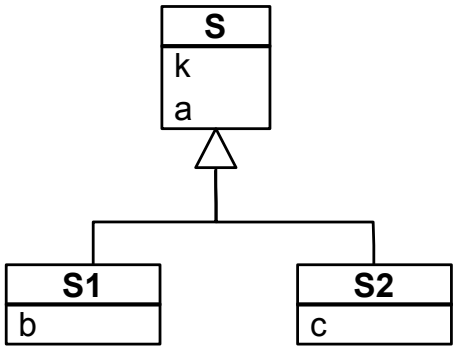
S (k, a)
S1 (k->S, a, b)
S2 (k->S, a, c)

Subclass relations contain all attributes

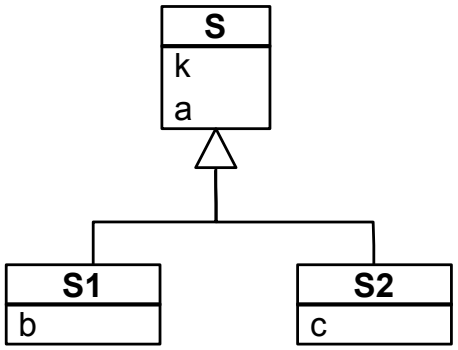
In complete generalizations, the relation for the superclass may be eliminated

Good for disjoint generalizations and when the superclass has few attributes and subclasses many attributes

Generalizations – Use nulls



Generalizations – Use nulls



S (<u>k</u>, a, b, c)

One relation with all the attributes of all the classes

NULL values on non-existing attributes for a specific object

Good for heavily overlapping generalizations with a small number of subclasses

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

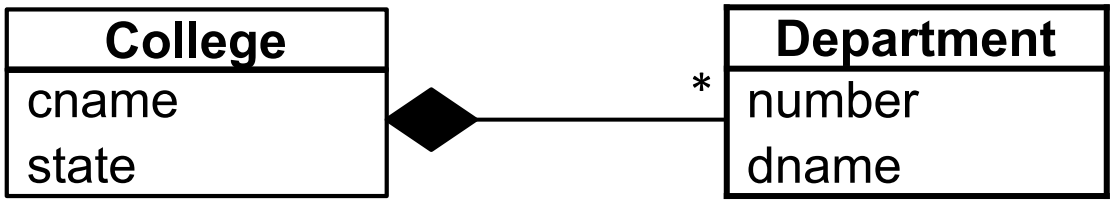
~~Generalizations~~

Composition & Aggregation

Constraints

Derived Elements

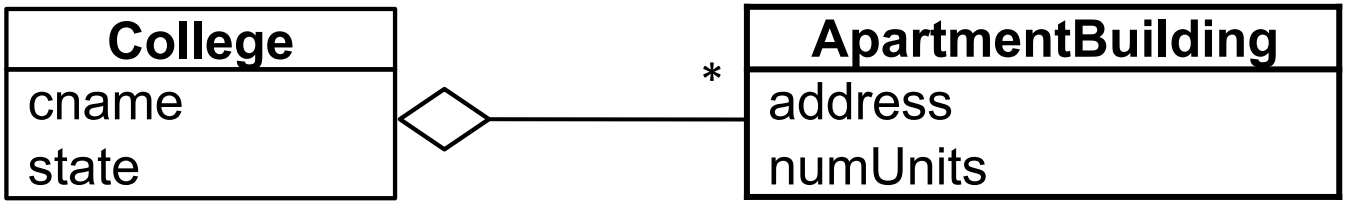
Composition



Treat it as a regular many-to-one association:

College (cname, state)
Department (dnumber, dname, cname->College)

Aggregation



Treat it as a regular many-to-one association:

College (cname, state)

ApartmentBuilding (address, numUnits, cname->College)

can be NULL

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

~~Generalizations~~

~~Composition & Aggregation~~

Constraints

Derived Elements

Constraints and Derived Elements

- Constraints

  Apart from primary keys (underline) and foreign keys (arrow), we can use the following keywords to set additional constraints:

  NOT NULL
  : Ensures that the value for an attribute can never be null

  UNIQUE
  : Ensures that the value for an attribute is always unique

  CHECK
  : Ensures that the value for an attribute meets a specific condition

  DEFAULT
  : Specifies a default value for an attribute

- Derived Elements

  Treat them as regular elements

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

~~Generalizations~~

~~Composition & Aggregation~~

~~Constraints~~

~~Derived Elements~~

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

- Section 2.1 – Basics of the Relational Model
- Section 4.8 – From UML Diagrams to Relations
- Section 4.6 – Converting Subclass Structures to Relations