COMP 3111
SOFTWARE ENGINEERING

LECTURE 3
MODELING SOFTWARE SYSTEMS USING UML

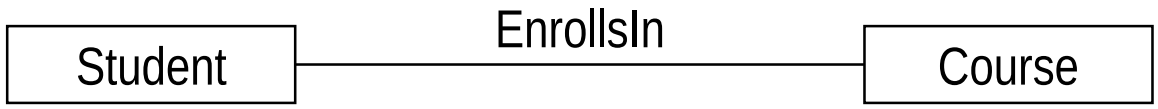
MODELING SOFTWARE SYSTEMS USING UML: OUTLINE

- ✓ UML and Object-oriented Modeling
  - Overview of the UML
  - Object-oriented Modeling
- ✓ Class
  - Attribute
  - Operation
- ✓ Association
  - Multiplicity
  - Aggregation and Composition
- ➔ **Association Class**

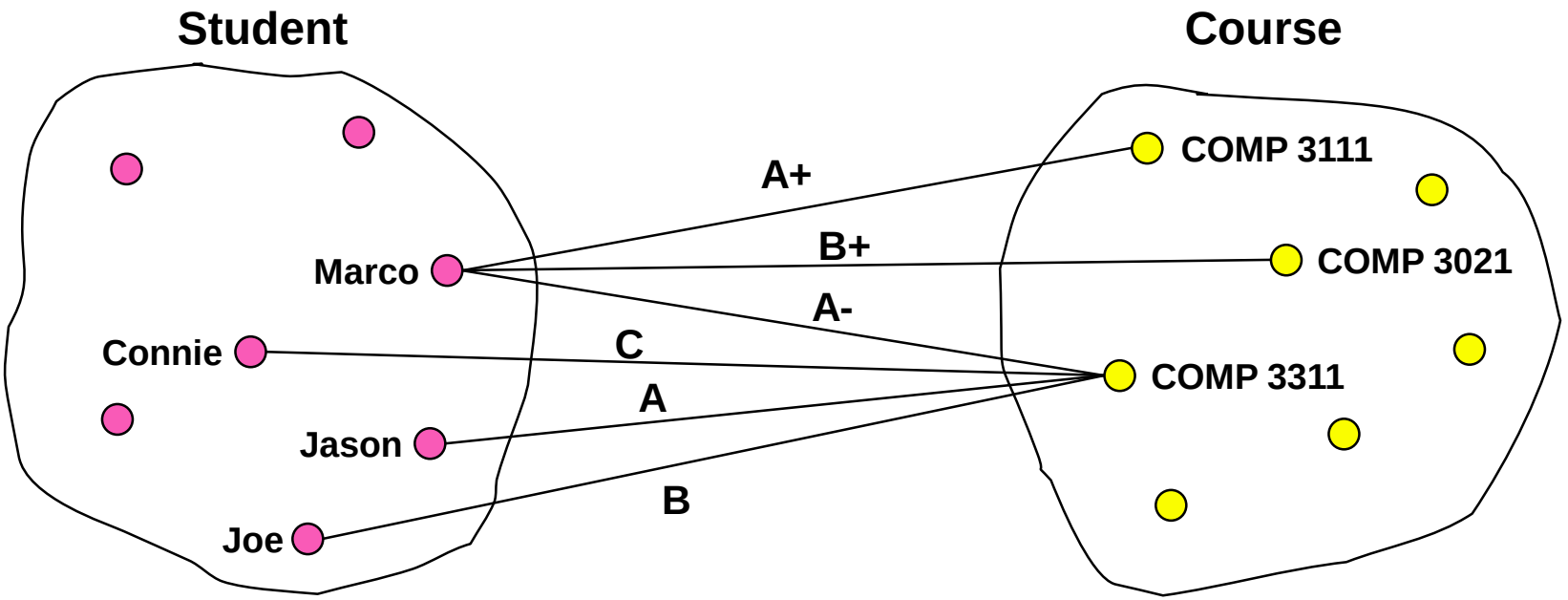
Generalization
- Inheritance
- Coverage

Constraints

ASSOCIATION: ASSOCIATION CLASS



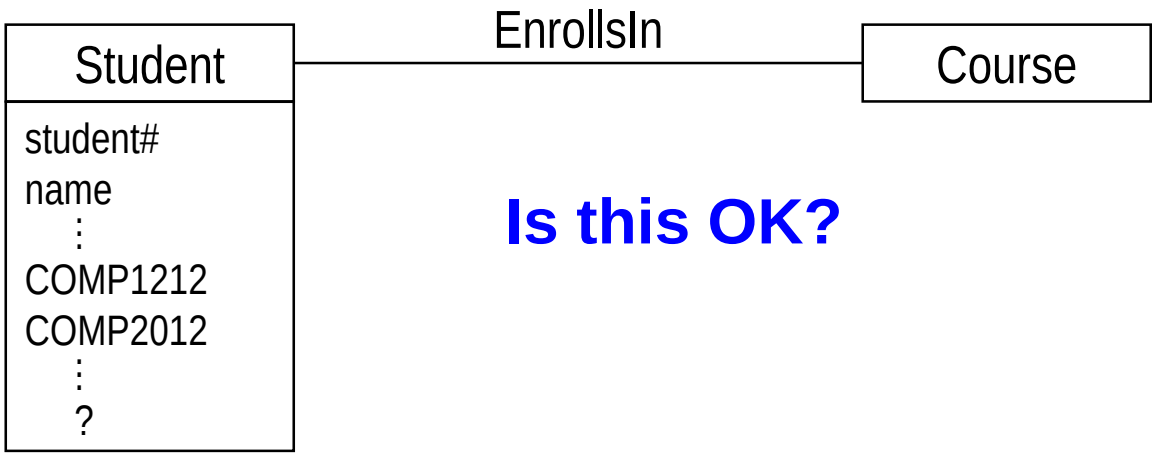
Where should we put an attribute like grade?



ASSOCIATION: ASSOCIATION CLASS (cont'd)

Option 1: Use many attributes for grade.

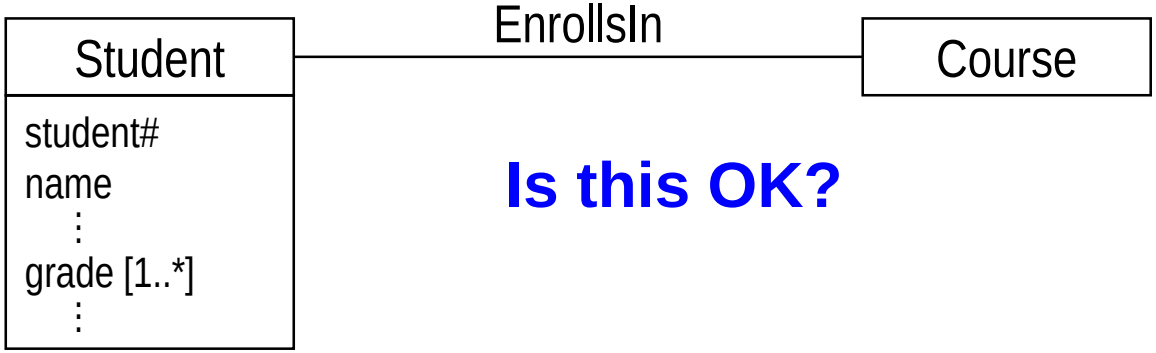
E.g., in Student



Is this OK?

Option 2: Use a multi-valued attribute for grade.

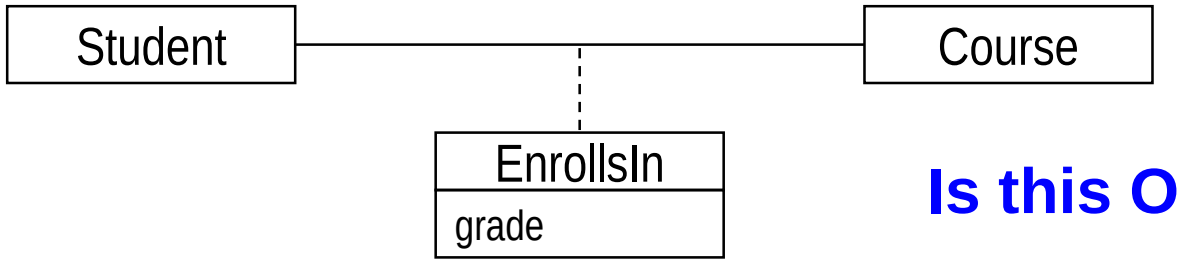
E.g., in Student



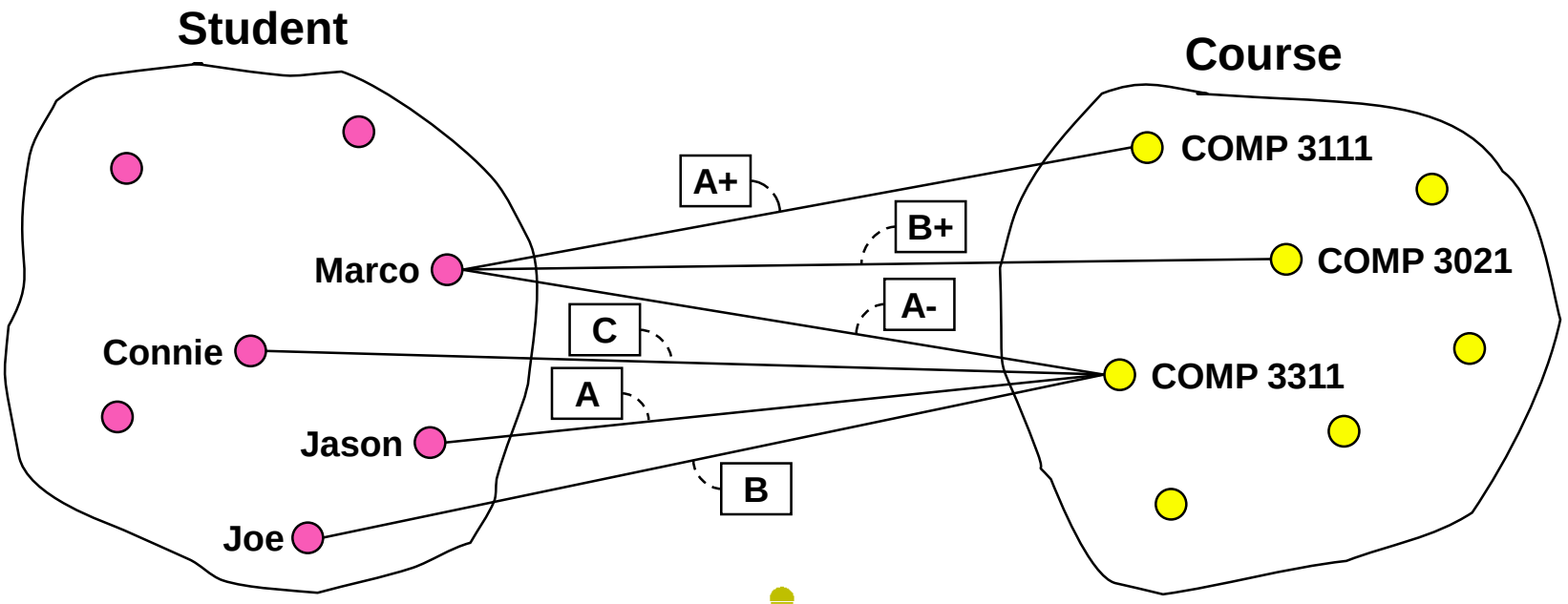
Is this OK?

ASSOCIATION: ASSOCIATION CLASS (cont'd)

Option 3: Use an association class.

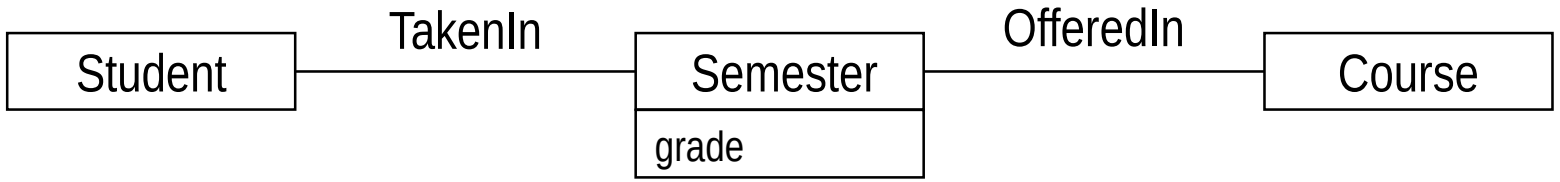


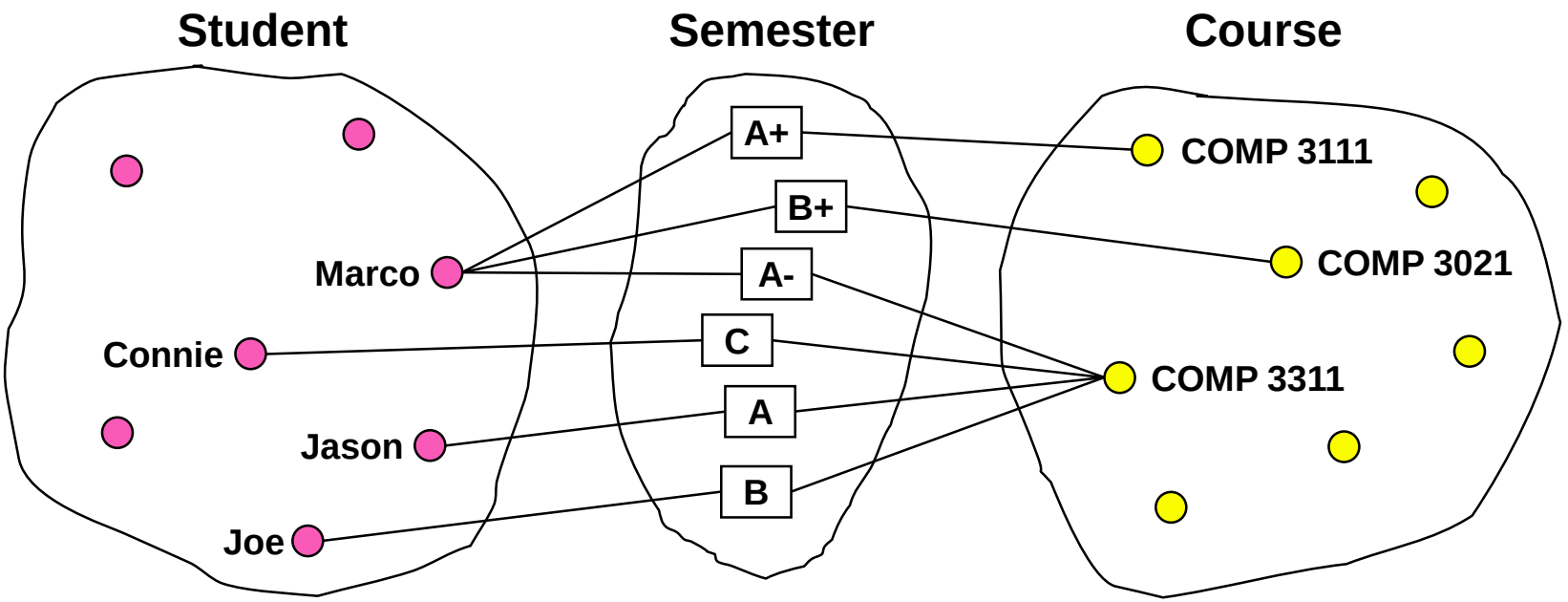
Is this OK?



ASSOCIATION: ASSOCIATION CLASS (cont'd)

Option 4: Use a separate class.



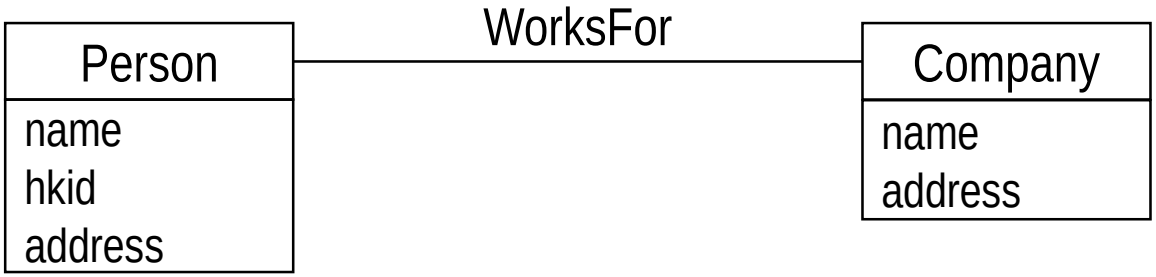


Note that each Semester object *must be related* to a Student and Course.

ASSOCIATION: ASSOCIATION CLASS (cont'd)

When to associate an attribute with an association?

What about an attribute like salary **?**



An association class is *most often* needed for many to many associations!

MODELING SOFTWARE SYSTEMS USING UML: OUTLINE

- ✓ UML and Object-oriented Modeling
  - Overview of the UML
  - Object-oriented Modeling
- ✓ Class
  - Attribute
  - Operation
- ✓ Association
  - Multiplicity
  - Aggregation and Composition
- ✓ Association Class
- ➔ **Generalization**
  - **Inheritance**
  - **Coverage**

Constraints

GENERALIZATION

A *generalization* is a relationship between classes of the same kind.

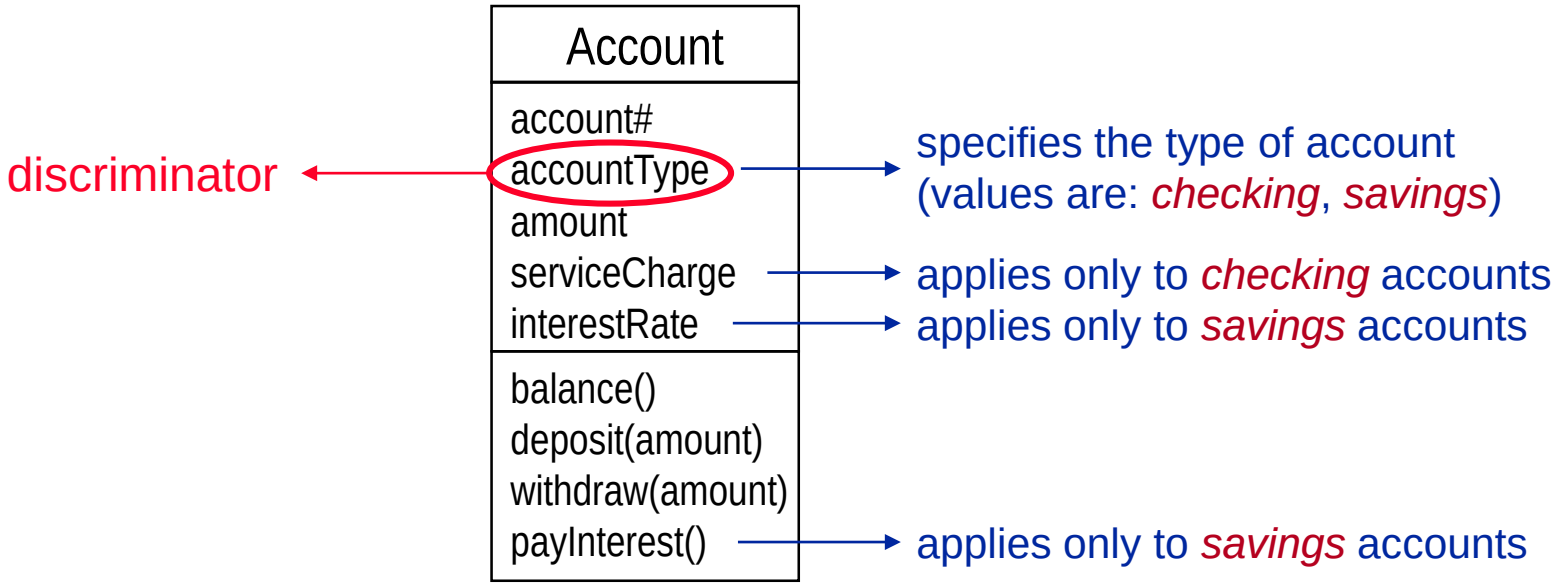
- *If it is meaningful to do so*, we classify classes according to the similarity of their attributes, operations and associations.

☞ **Look for "kind-of" statements that are true in an application domain.**

Goal: Simplicity of representation and modeling clarity.

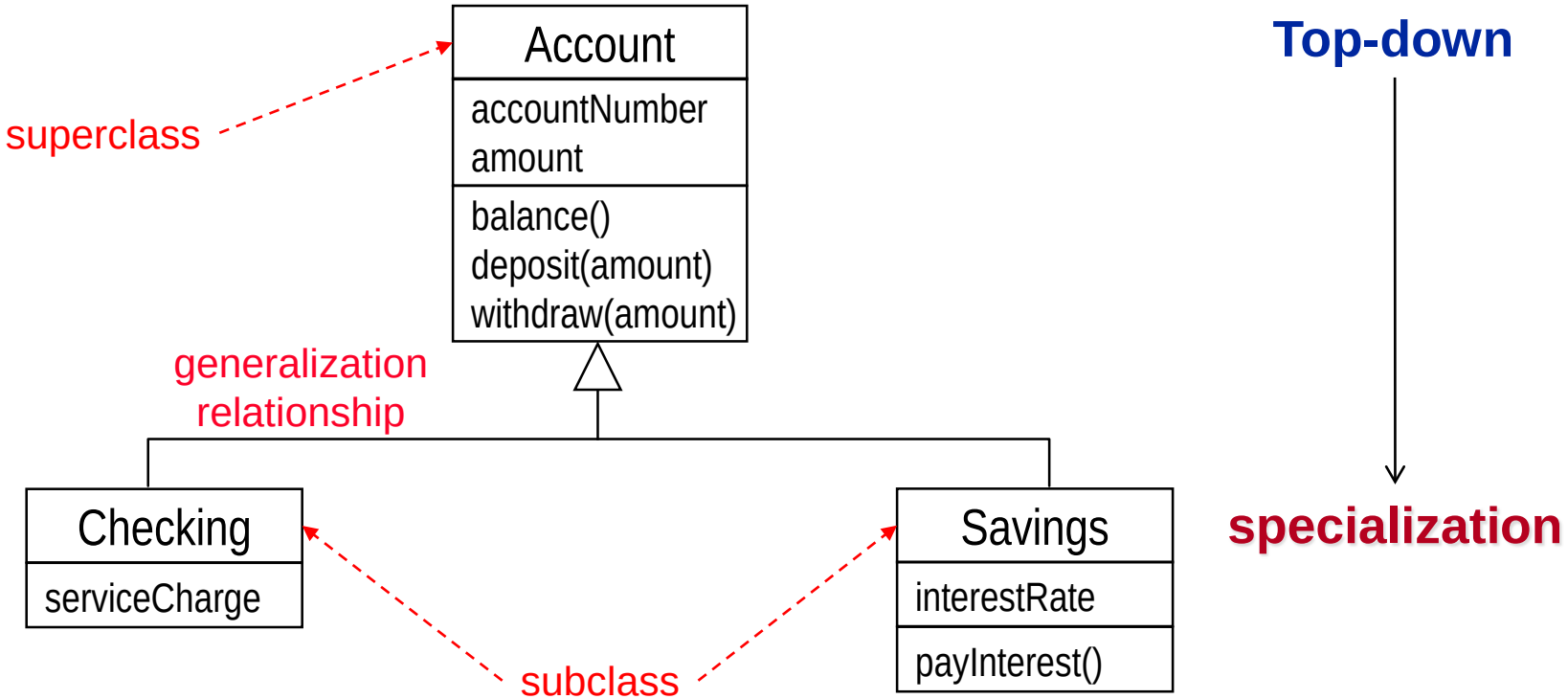
GENERALIZATION (cont'd)

discriminator: An attribute of enumeration type that indicates which property of a class is used to create a generalization relationship.



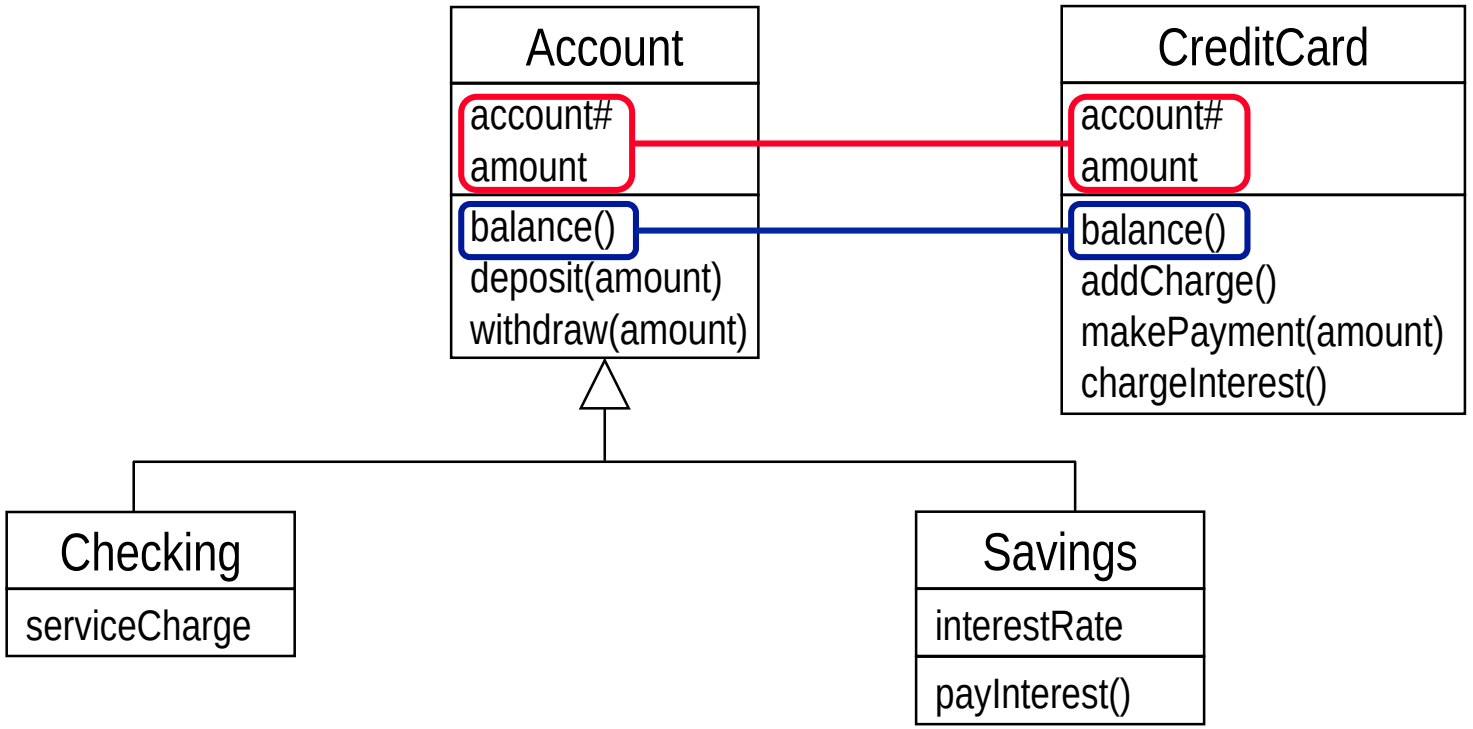
GENERALIZATION (cont'd)

discriminator: An attribute of enumeration type that indicates which property of a class is used to create a generalization relationship.



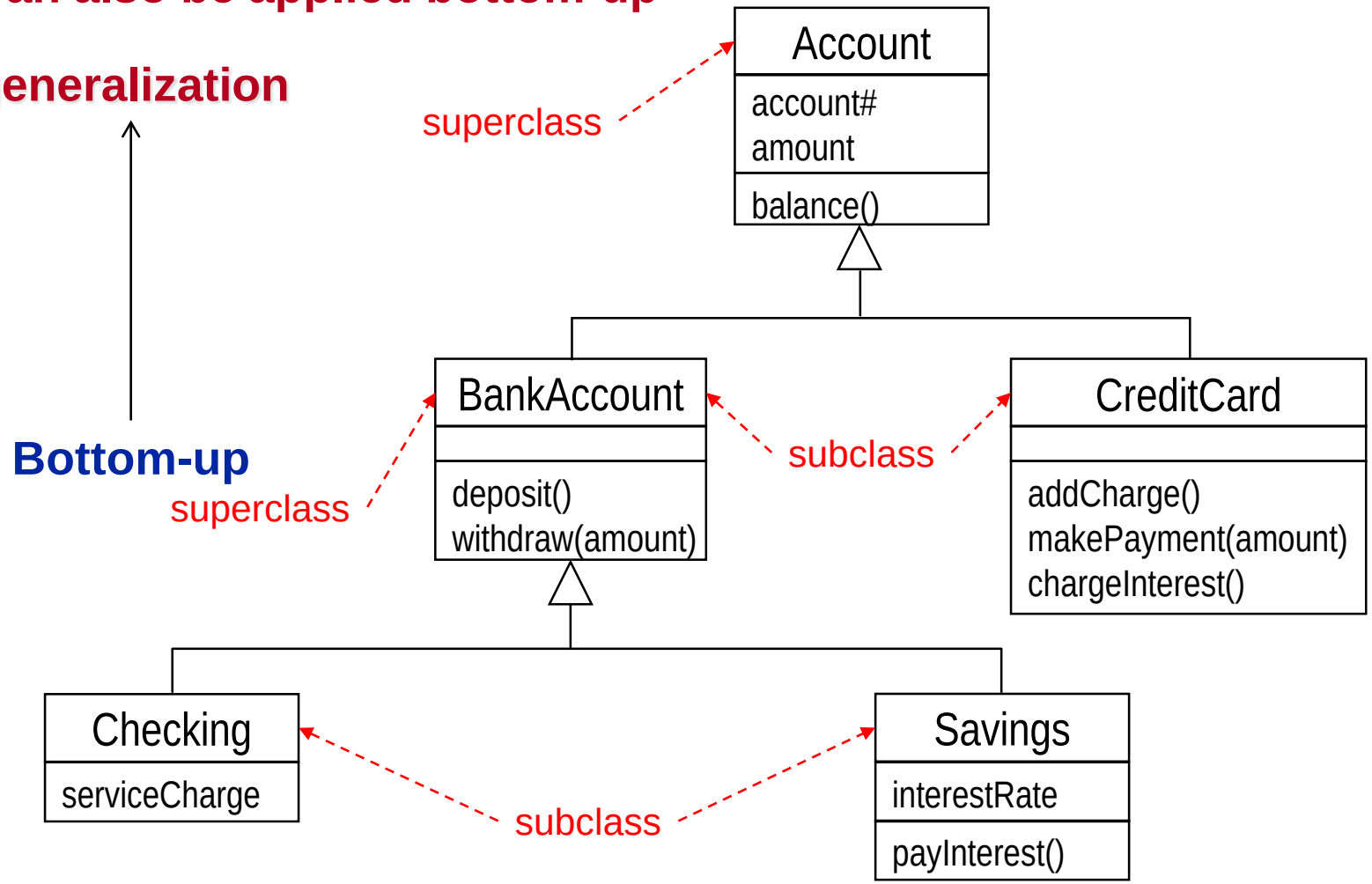
GENERALIZATION (cont'd)

Can also be applied bottom-up



GENERALIZATION (cont'd)

Can also be applied bottom-up generalization



GENERALIZATION: INHERITANCE

***Inheritance* is the taking up of properties by a subclass from its superclasses.**

- We extract the *common* attributes, relationships and operations, associate them with the superclass and inherit them to the subclass(es).

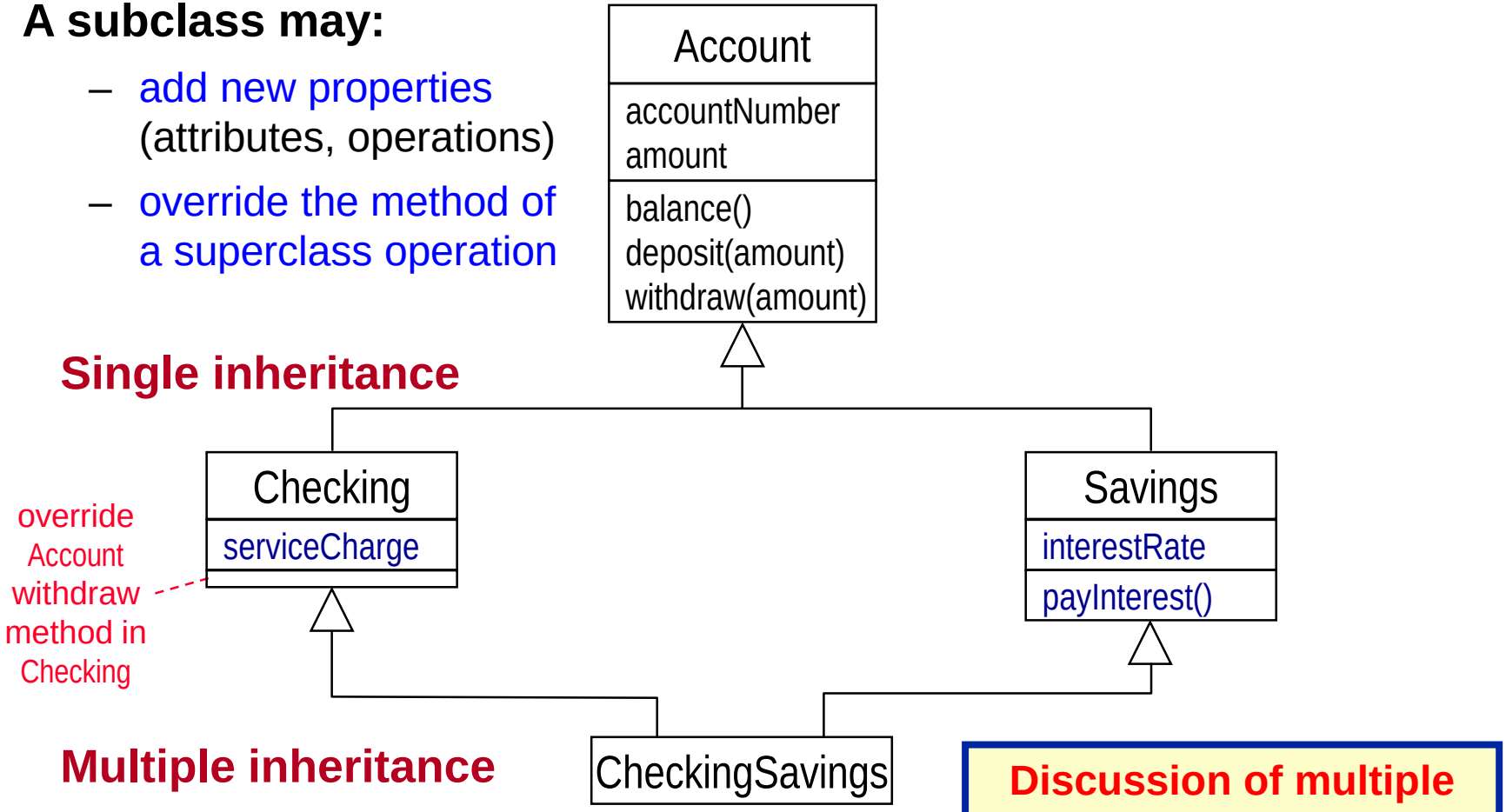
☞ **Attributes and operations are only defined in one place:**

- ✓ reduces redundancy of descriptions.
- ✓ promotes reusability of descriptions.
- ✓ simplifies modification of descriptions.

GENERALIZATION: INHERITANCE (cont'd)

A subclass may:

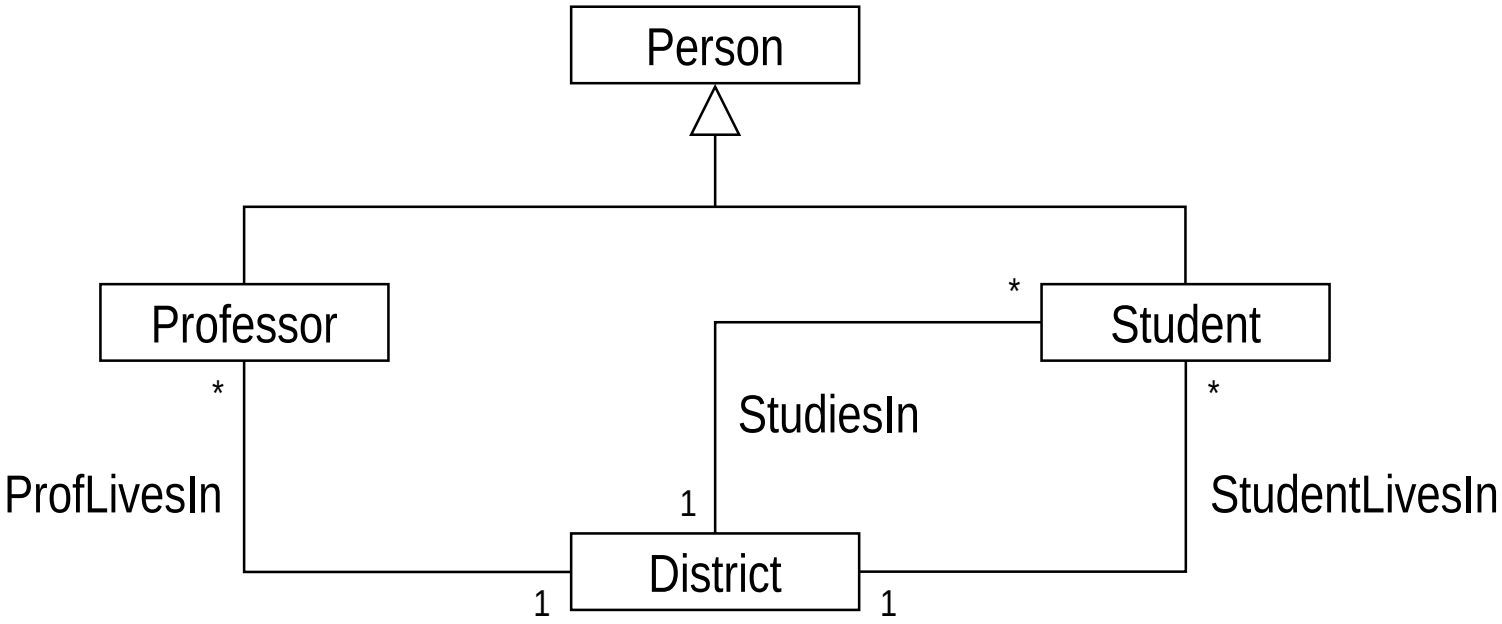
- add new properties (attributes, operations)
- override the method of a superclass operation



Discussion of multiple inheritance is beyond the scope of this course.

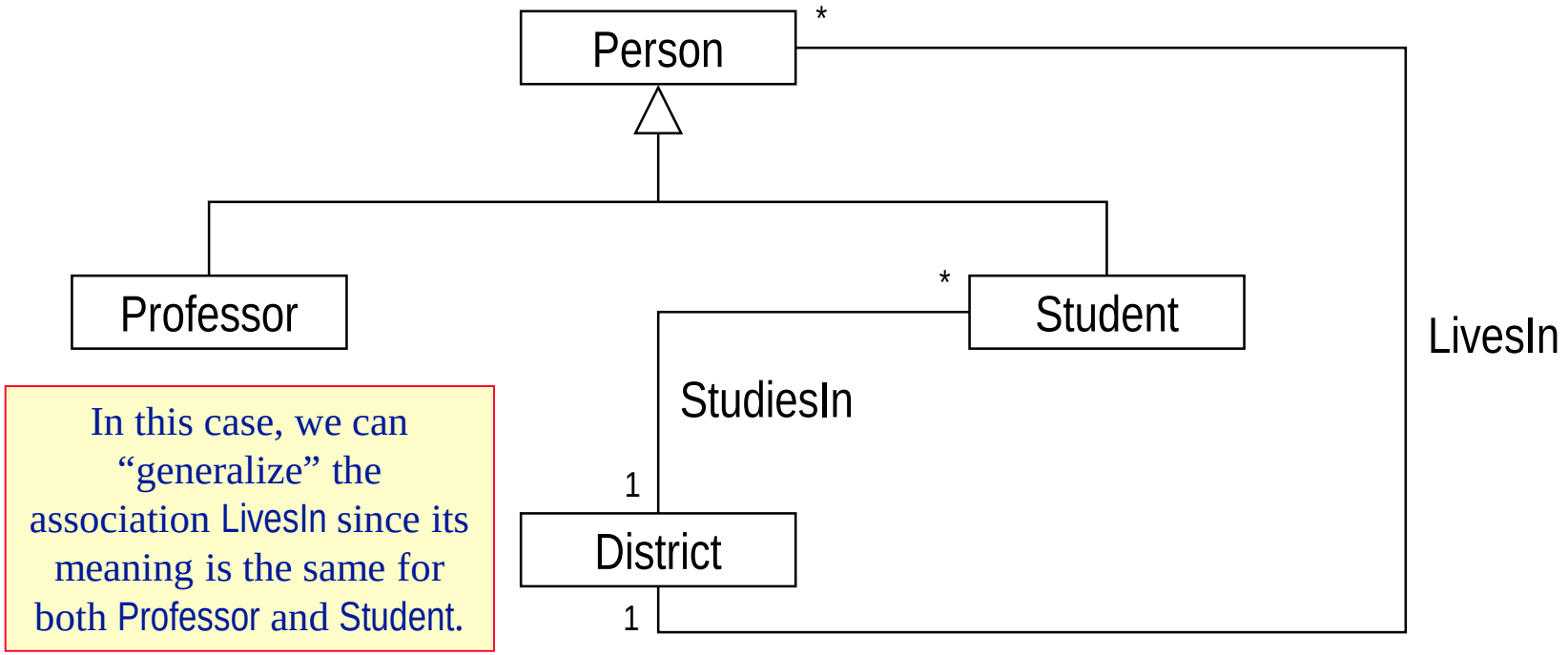
GENERALIZATION: INHERITANCE (cont'd)

☞ **How to handle associations in which subclasses participate?**



GENERALIZATION: INHERITANCE (cont'd)

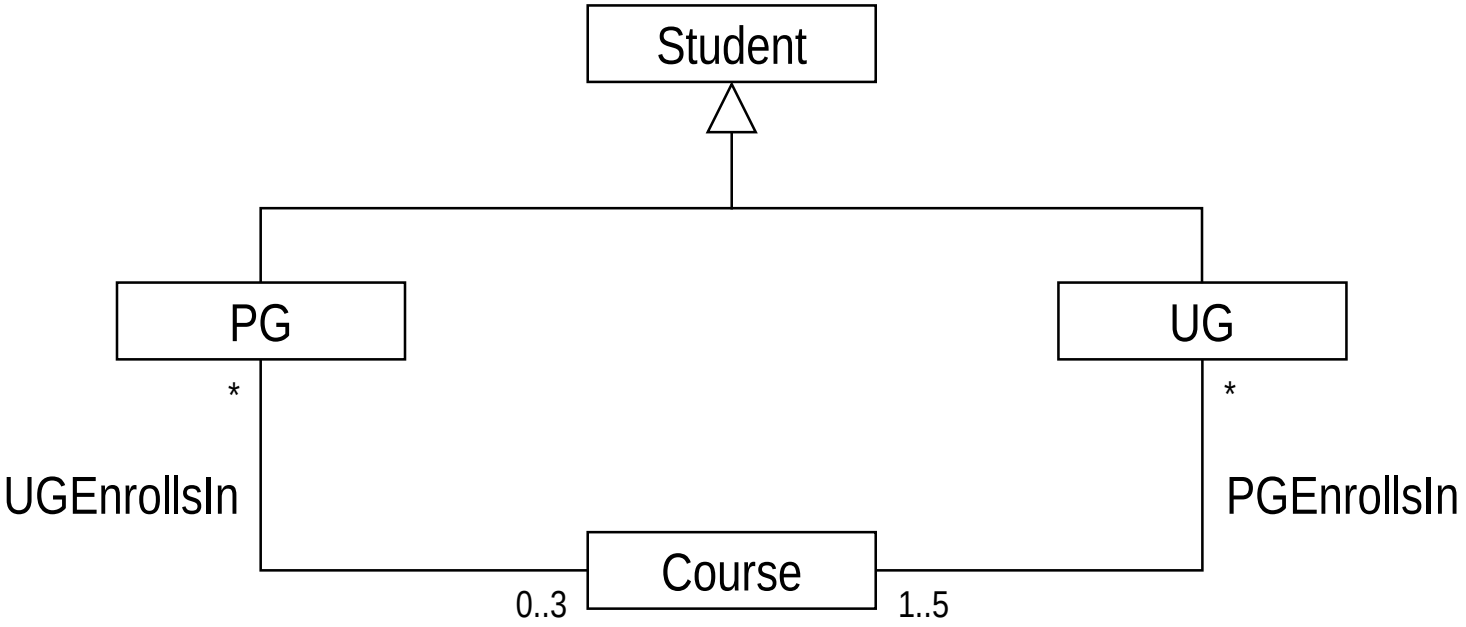
☞ **How to handle associations in which subclasses participate?**



In this case, we can "generalize" the association LivesIn since its meaning is the same for both Professor and Student.

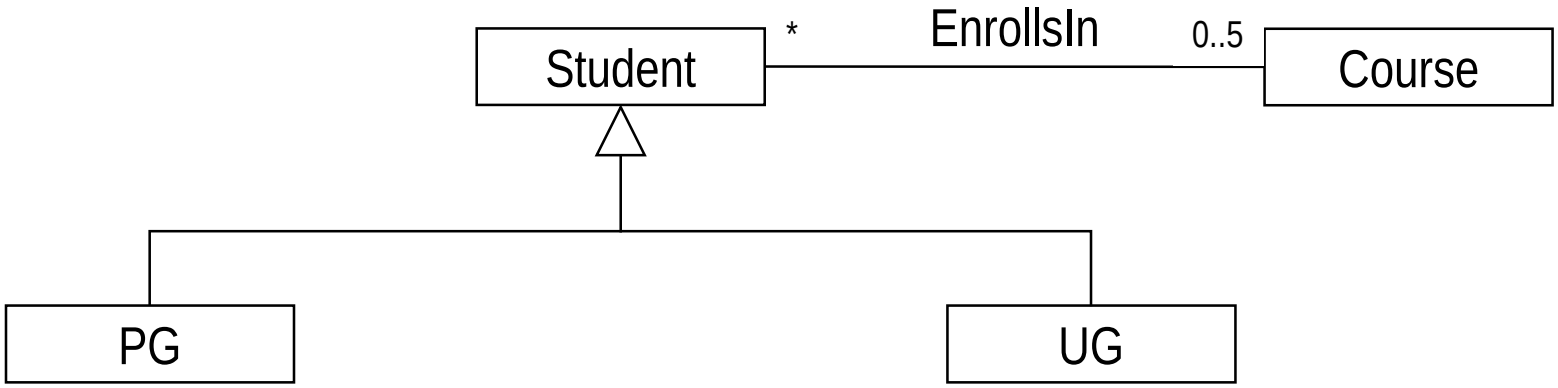
GENERALIZATION: INHERITANCE (cont'd)

☞ **Sometimes it is necessary to "adjust" the multiplicity of an association.**



GENERALIZATION: INHERITANCE (cont'd)

☞ **Sometimes it is necessary to "adjust" the multiplicity of an association.**

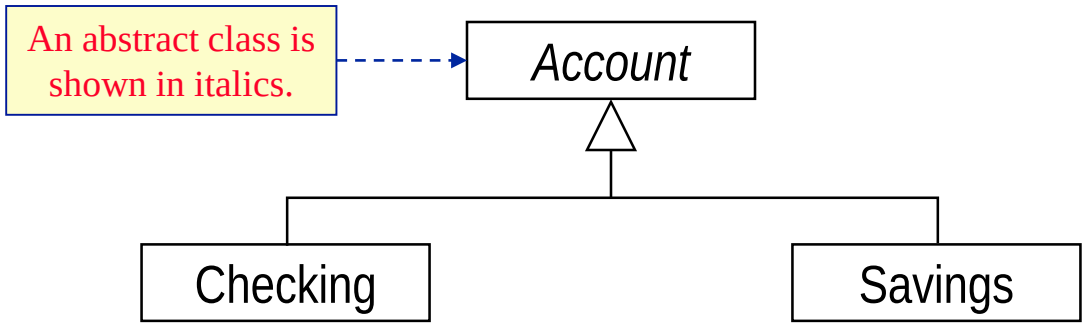


In this case, although we can "generalize" the EnrollsIn association, we need to use the *minimum* of the min-card and the *maximum* of max-card of both original EnrollsIn associations in order to preserve the semantics of the original class diagram.

GENERALIZATION: ABSTRACT CLASS

An *abstract class* is a class that has no direct instances.

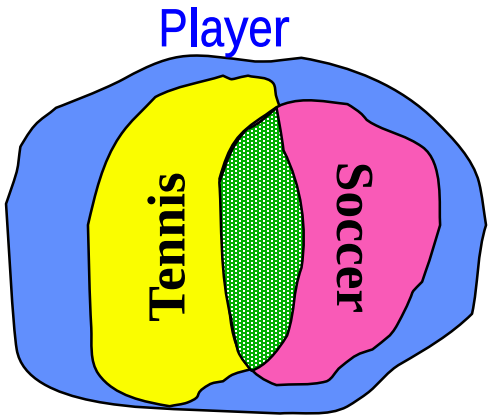
- An abstract class is used, for modeling purposes, as a *container for definitions*, but no instances of the class are of interest.

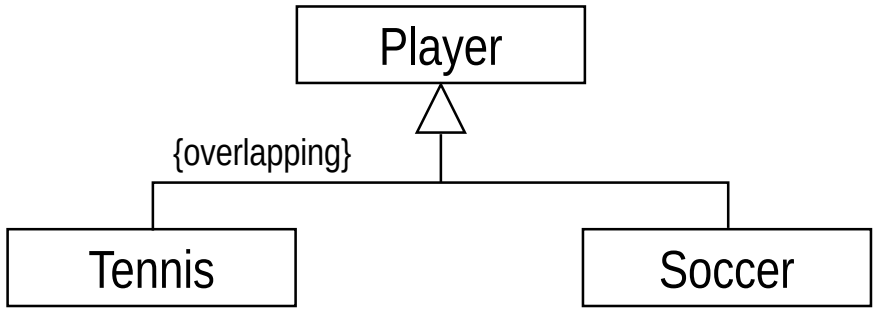


Note: Operations may also be abstract → no method specified.

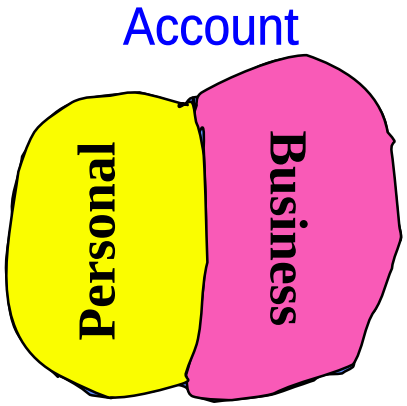
GENERALIZATION: DISJOINTNESS COVERAGE

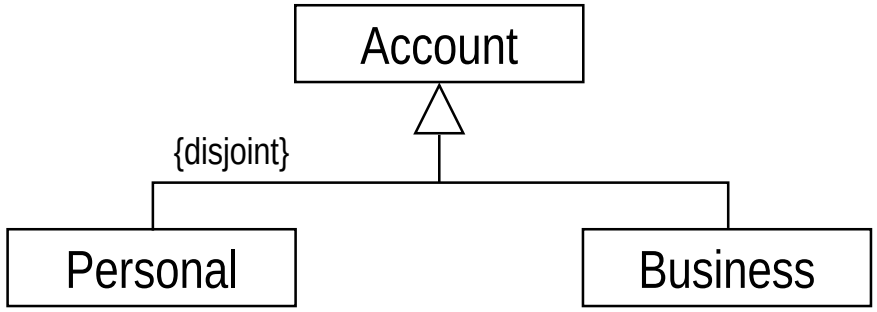
overlapping - A superclass object can be a member of more than one subclass.





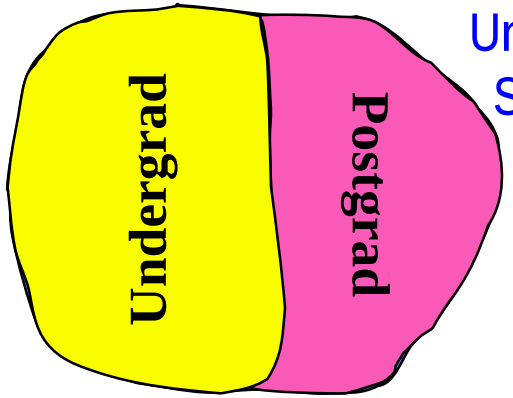
disjoint - A superclass object is a member of at most one subclass.

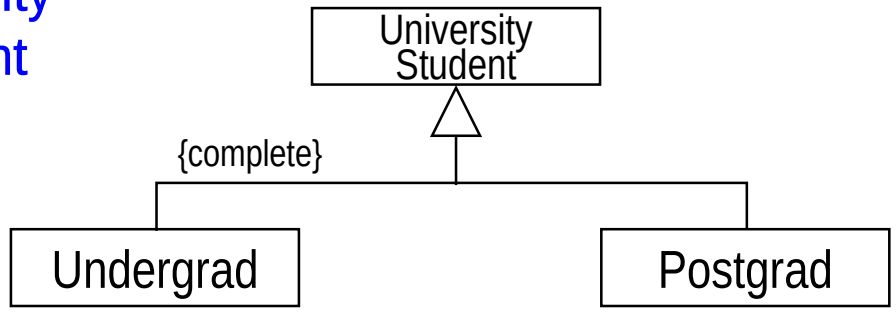




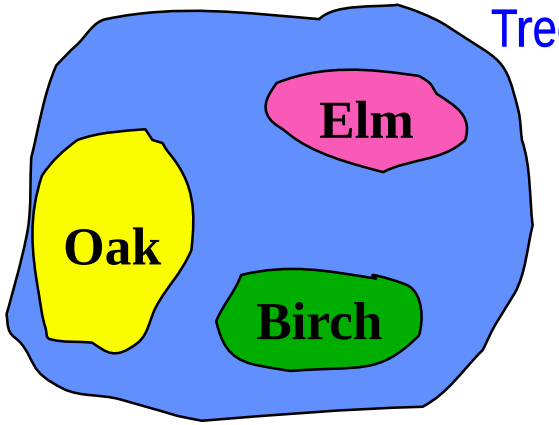
GENERALIZATION: COMPLETENESS COVERAGE

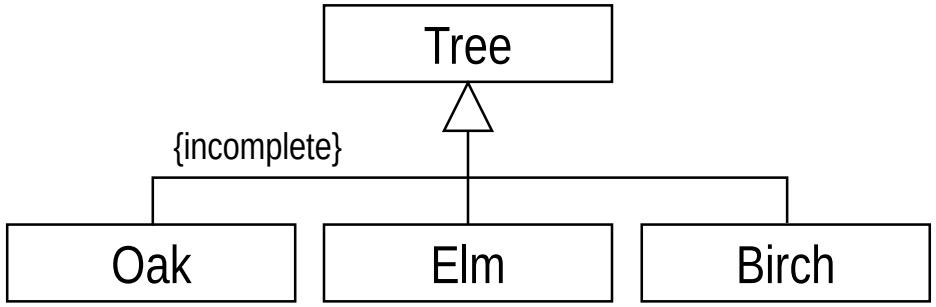
complete - All superclass objects must be members of some subclass.





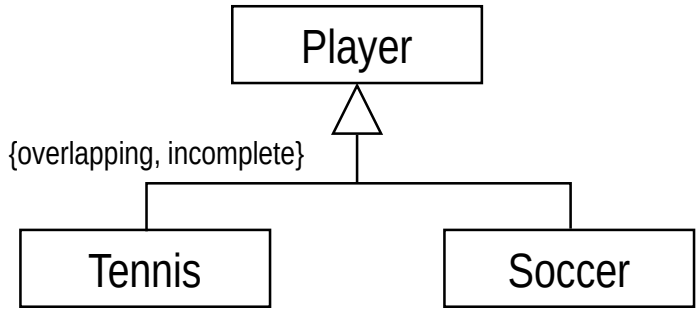
incomplete - Some superclass object is not a member of any subclass.



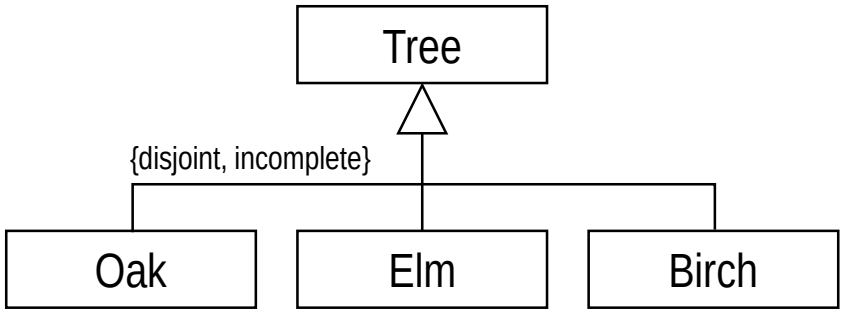


GENERALIZATION: COVERAGE TYPES

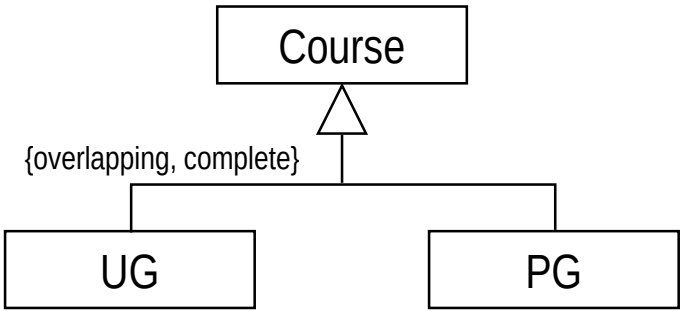
overlapping, incomplete



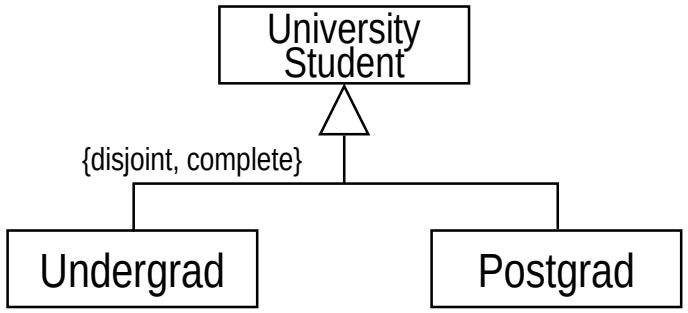
disjoint, incomplete



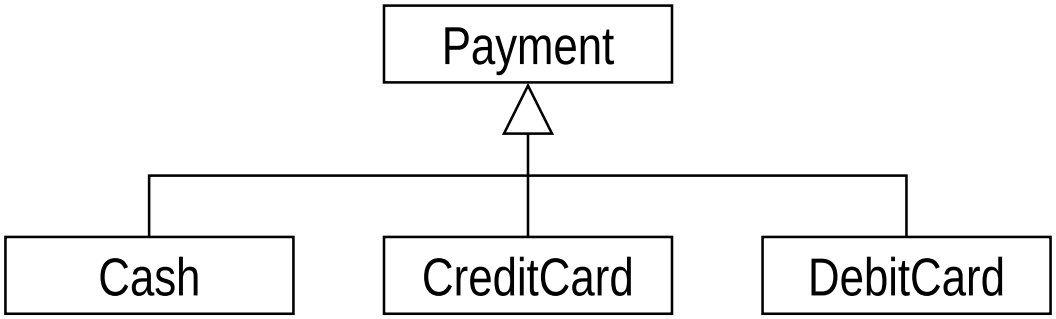
overlapping, complete



disjoint, complete



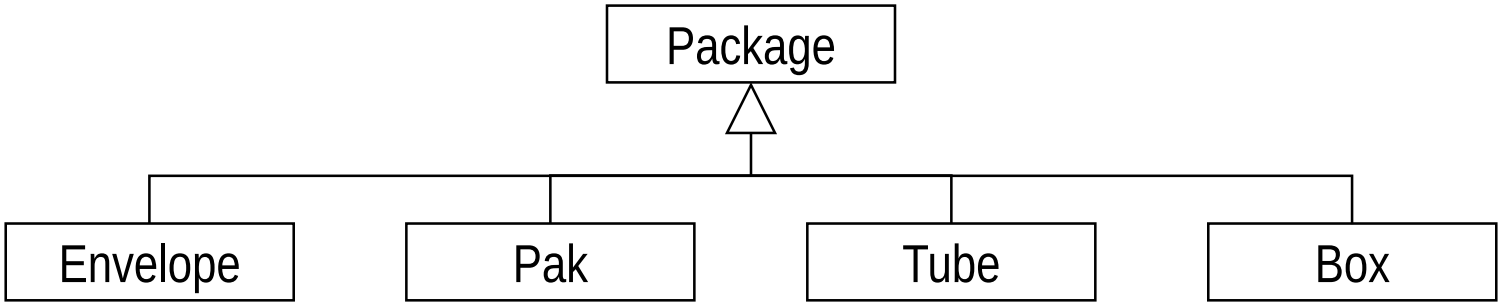
GENERALIZATION EXAMPLE



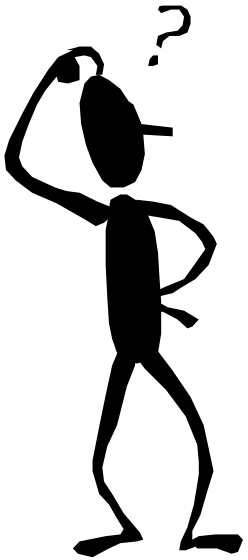
The coverage of the generalization shown above is:

- overlapping, complete
- disjoint, complete
- overlapping, incomplete
- disjoint, incomplete

SINEX — COURSE PROJECT QUESTION?

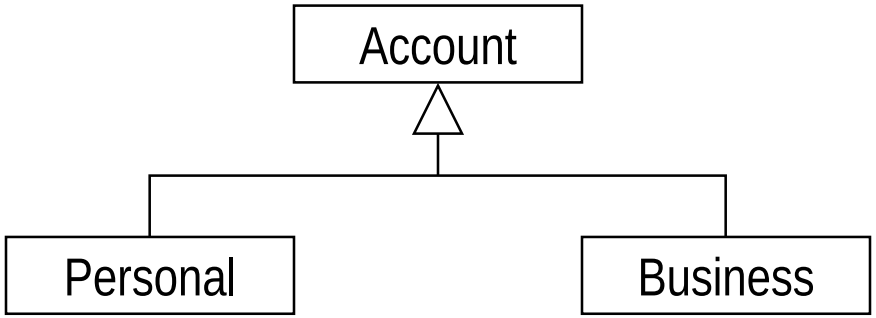


The coverage of the generalization shown above is:

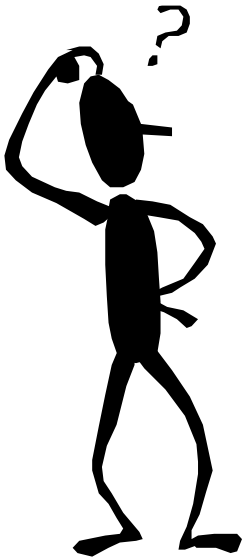


- overlapping, complete
- disjoint, complete
- overlapping, incomplete
- disjoint, incomplete

SINEX — COURSE PROJECT QUESTION?



The coverage of the generalization shown above is:



- overlapping, complete
- disjoint, complete
- overlapping, incomplete
- disjoint, incomplete

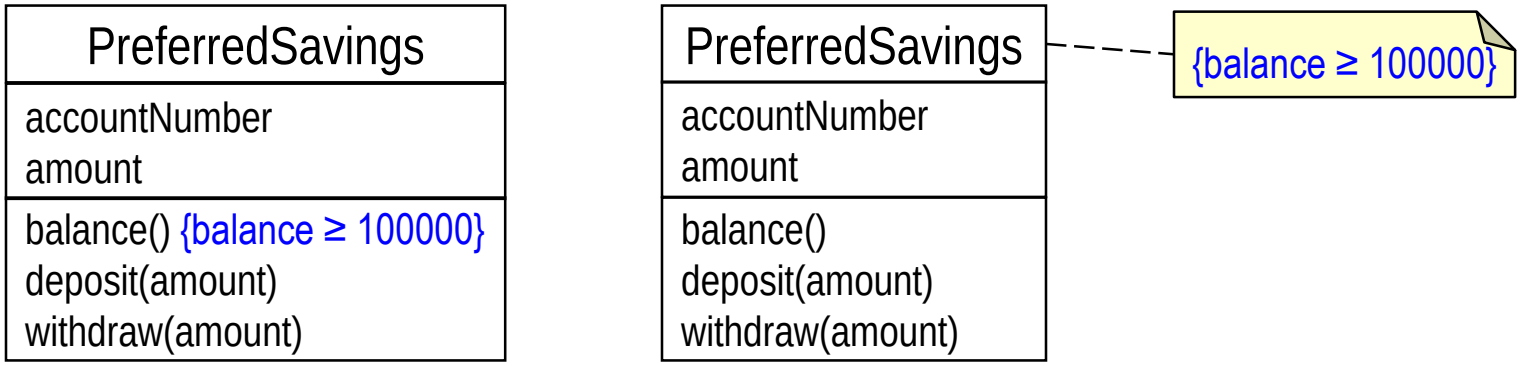
MODELING SOFTWARE SYSTEMS USING UML: OUTLINE

- ✓ UML and Object-oriented Modeling
  - Overview of the UML
  - Object-oriented Modeling
- ✓ Class
  - Attribute
  - Operation
- ✓ Association
  - Multiplicity
  - Aggregation and Composition
- ✓ Generalization
  - Inheritance
  - Coverage
- ✓ Association Class
- ➔ **Constraints**

CONSTRAINTS

A *constraint* is an assertion about properties of model elements that must be satisfied by the system.

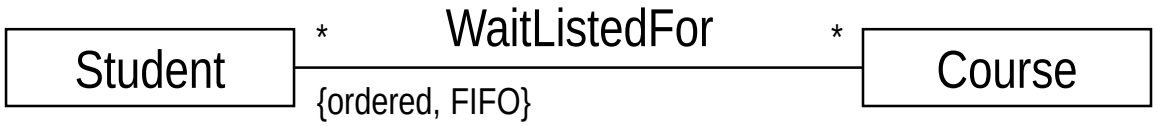
Example A preferential savings account whose balance always must be greater than $100,000.



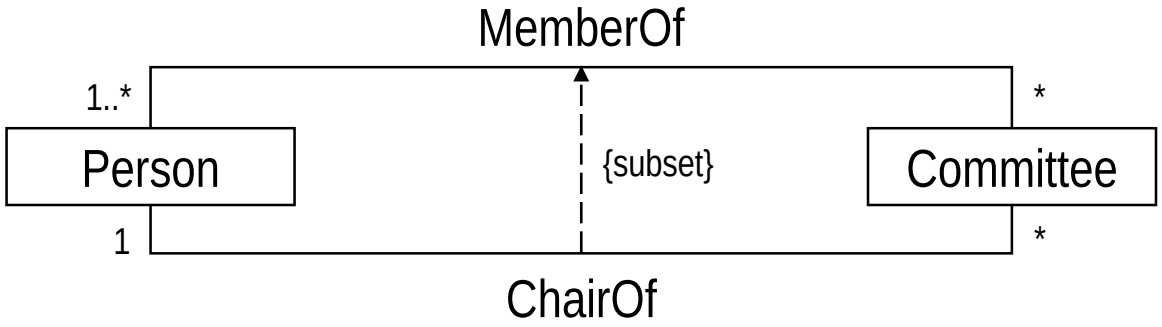
A constraint is a statement that can be tested (true/false) and should be enforced by the system implementation.

COMMON ASSOCIATION CONSTRAINTS

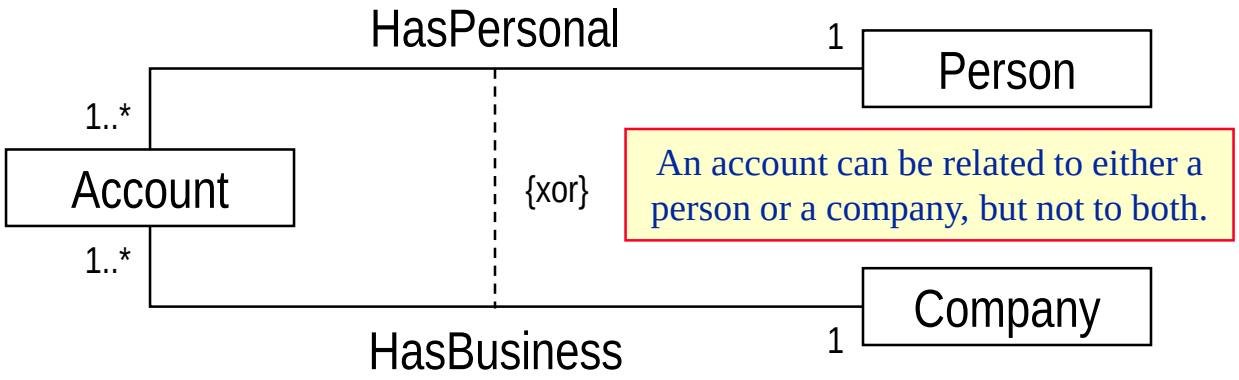
ordering



subset



xor



OBJECT CONSTRAINT LANGUAGE (OCL)

- UML provides a text-based formal constraint specification language called Object Constraint Language (OCL) (similar in form to C++ or Java).

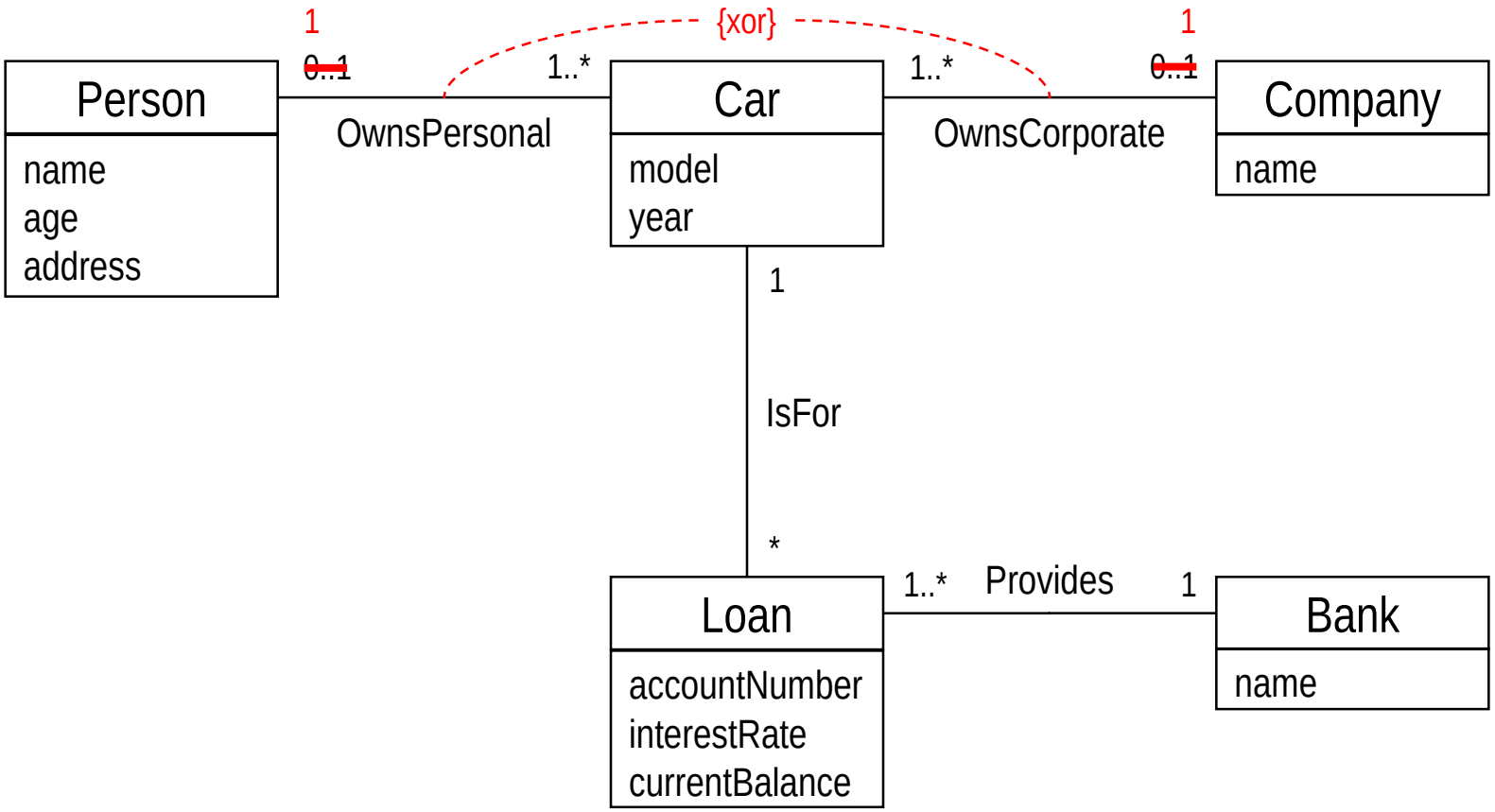
Discussion of OCL is beyond the scope of this course.

- For more information on OCL, please see:

  Object Constraint Language Specification, Version 2.0. available at http://www.omg.org/technology/documents/formal/ocl.htm.

  Warmer, J. and Kleppe, A. *The Object Constraint Language Second Edition: Getting Your Models Ready for MDA.* Addison-Wesley, 2003.

EXERCISE: CAR OWNERSHIP AND LOANS — SOLUTION REVISITED

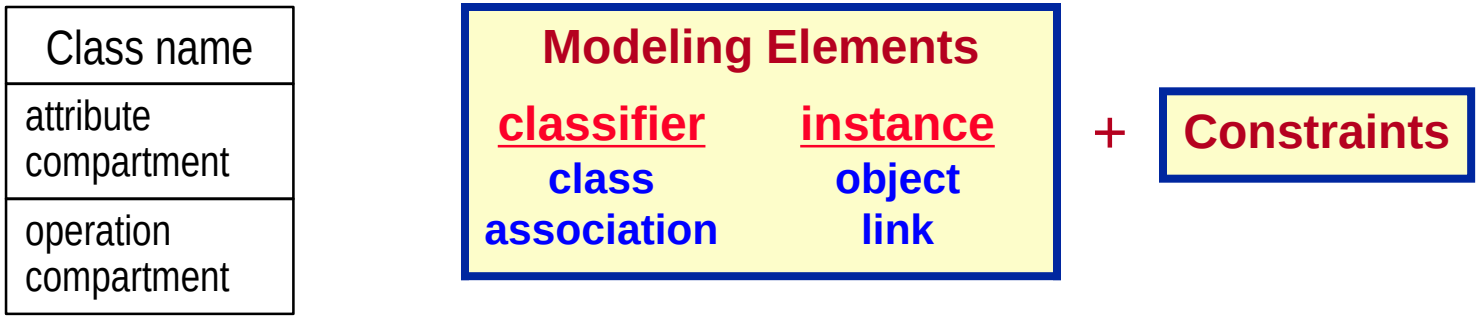


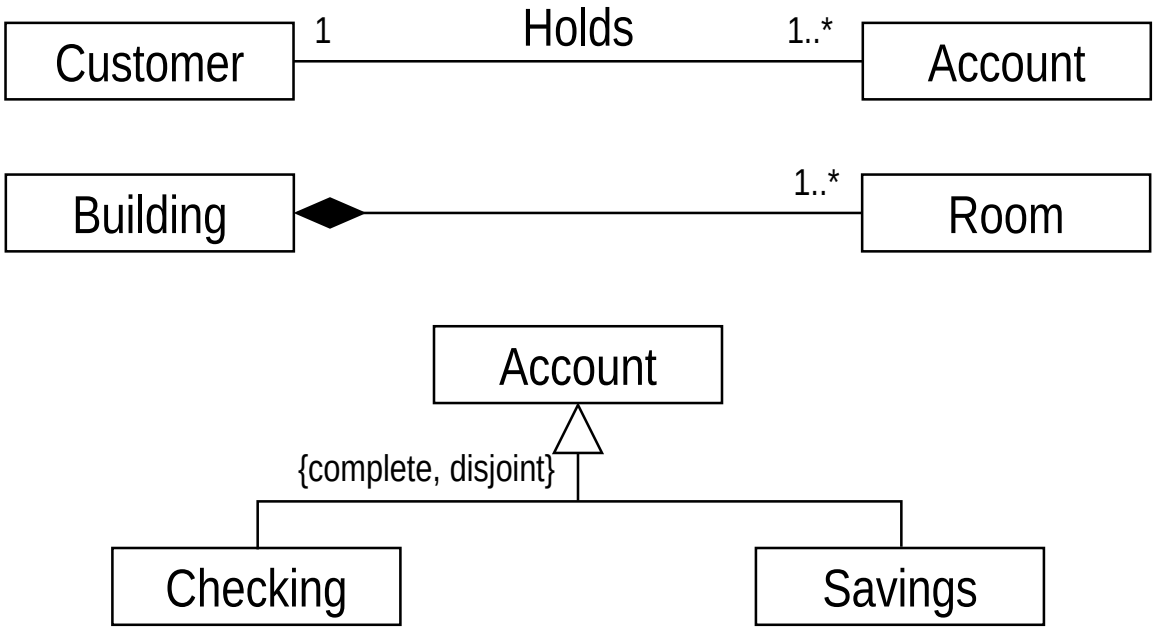
Problem 3

It is still possible for a car to be owned by both a person and a company or to not be owned at all!

➡ Requires an exclusive or constraint (see later).

MODELING SOFTWARE SYSTEMS USING UML: SUMMARY



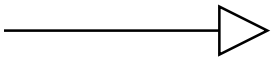


MODELING SOFTWARE SYSTEMS USING UML: SUMMARY

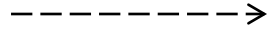
Relationships

association — describes links among object instances (only relationship that relates object instances)

generalization — relates a more general class (superclass) to a more specific kind of the general class (subclass)



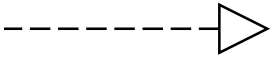
dependency — relates classes whose behaviour or implementation affect other classes



flow — relates two versions of an object at successive times

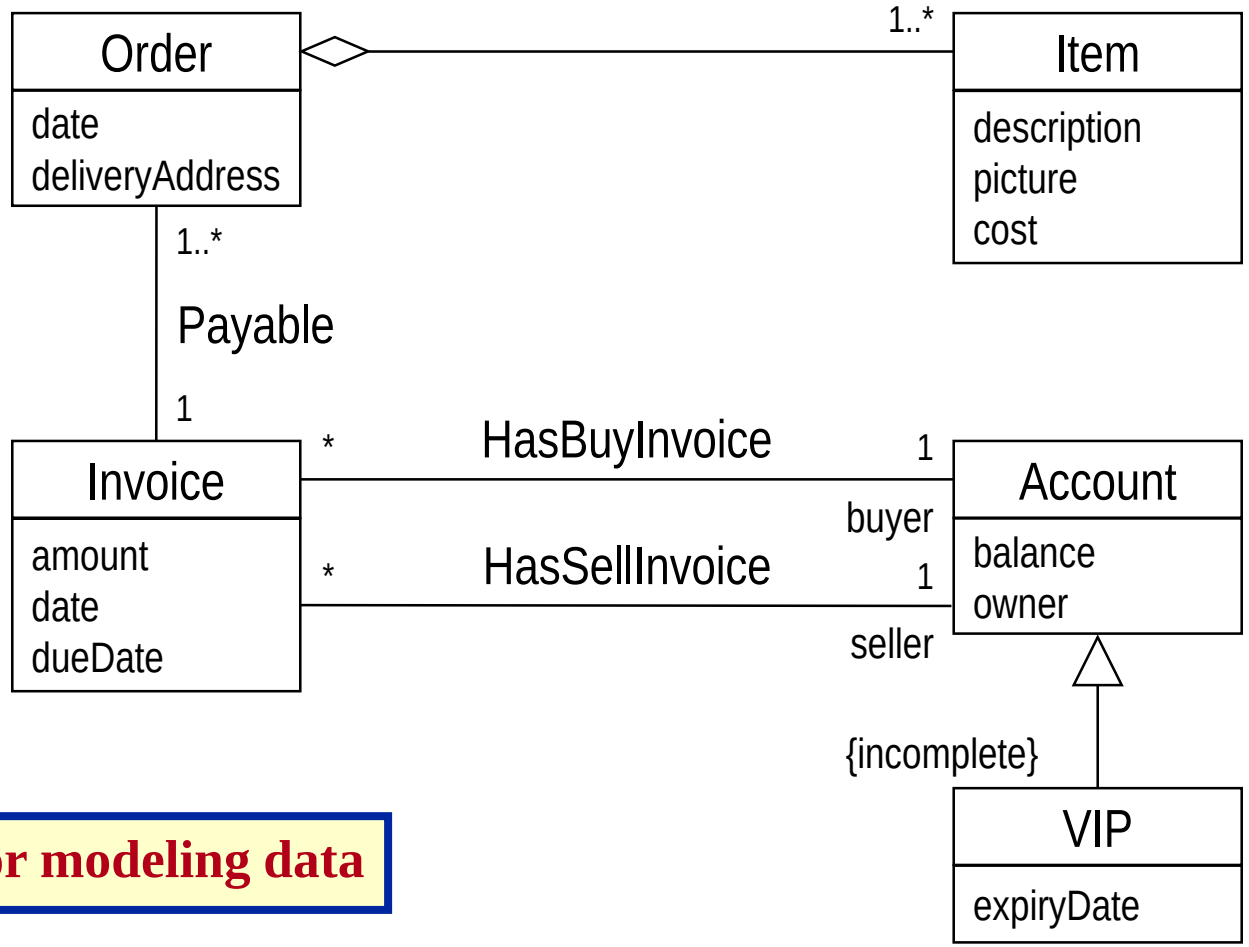
usage — shows that one class requires another class for its correct functioning

realization — relates a specification to its implementation (e.g., an interface to the classes that implement it)



MODELING SOFTWARE SYSTEMS USING UML: SUMMARY

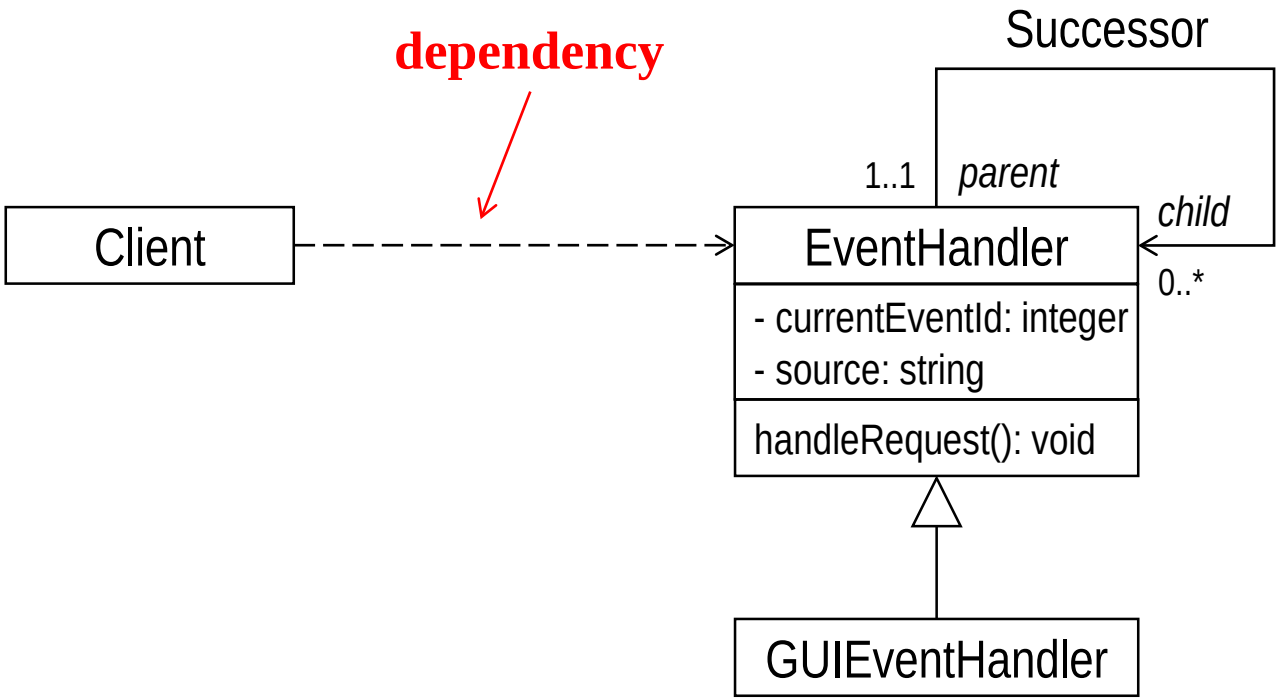
Static Modeling Example



For modeling data

MODELING SOFTWARE SYSTEMS USING UML: SUMMARY

Dynamic Modeling Example



For modeling programs

COMP 3111 SYLLABUS

1. ✓ **Introduction**
2. ✓ **Modeling Software Systems using UML**
3. Software Development
4. System Requirements Capture
5. Implementation
6. Testing
7. System Analysis and Design
8. Software Quality Assurance
9. Managing Software Development

MODELING SOFTWARE SYSTEMS USING UML

EXERCISE