Class Diagram

Objective

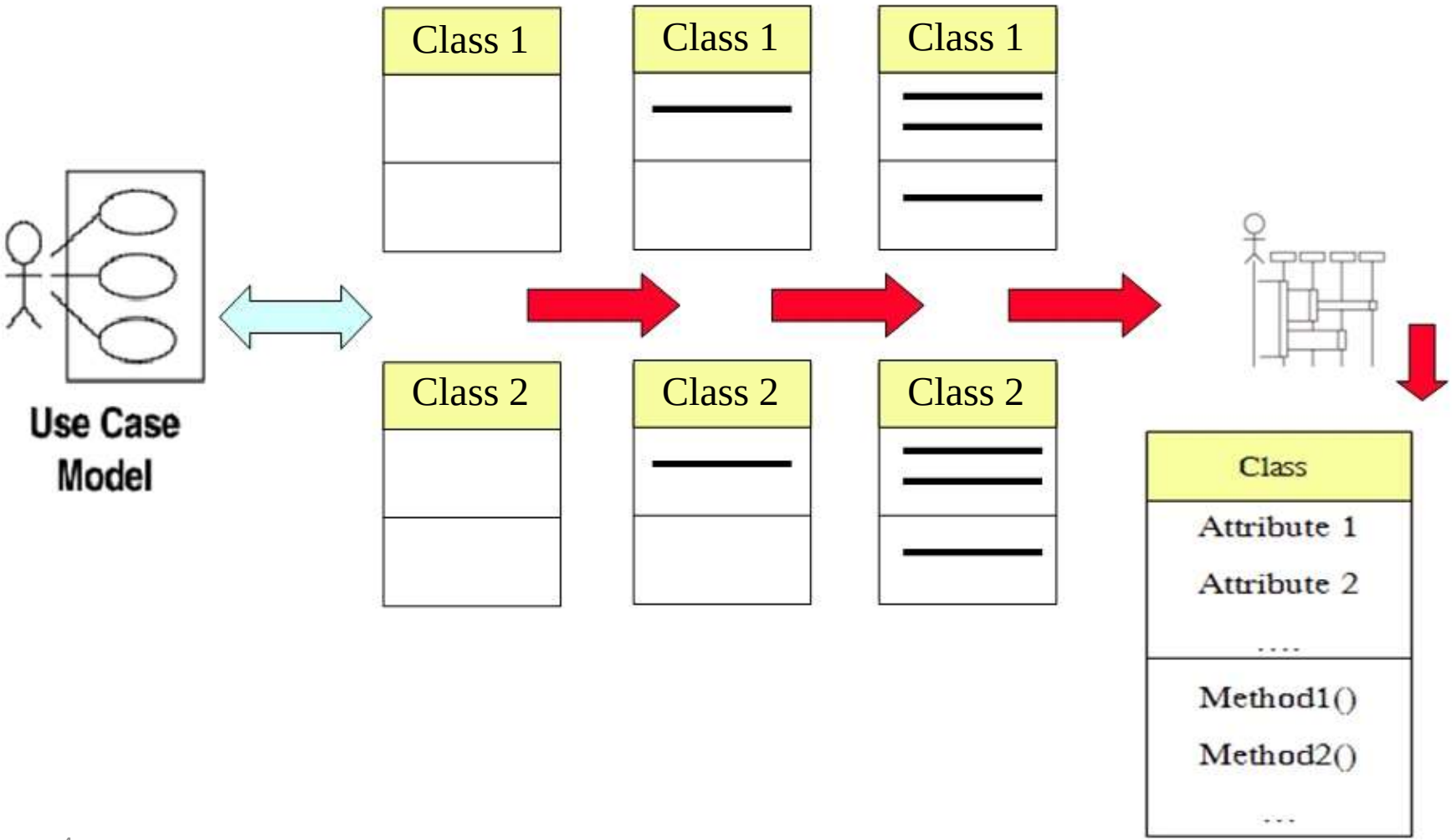
- Introduces the evolutionary approach for building classes
- Explain how to identify objects and attributes of classes
- Describe the technique of CRC ‘Class Responsibility and Collaborator’
- Explain how classes are related in a class diagram
- Explain generalization, association, aggregation and composition
-

OO Structural Modelling

The Static View of a system may be described using UML diagrams:

- **UML Class Diagrams**

From Use Cases to: Objects, Attributes, Operations (methods) - **"evolutionary "**



Identifying objects

- Look for **nouns** in the SRS (System Requirements Specifications) document
- Look for **NOUNS** in use cases descriptions
- A **NOUN** may be
  - Object
  - Attribute of an object

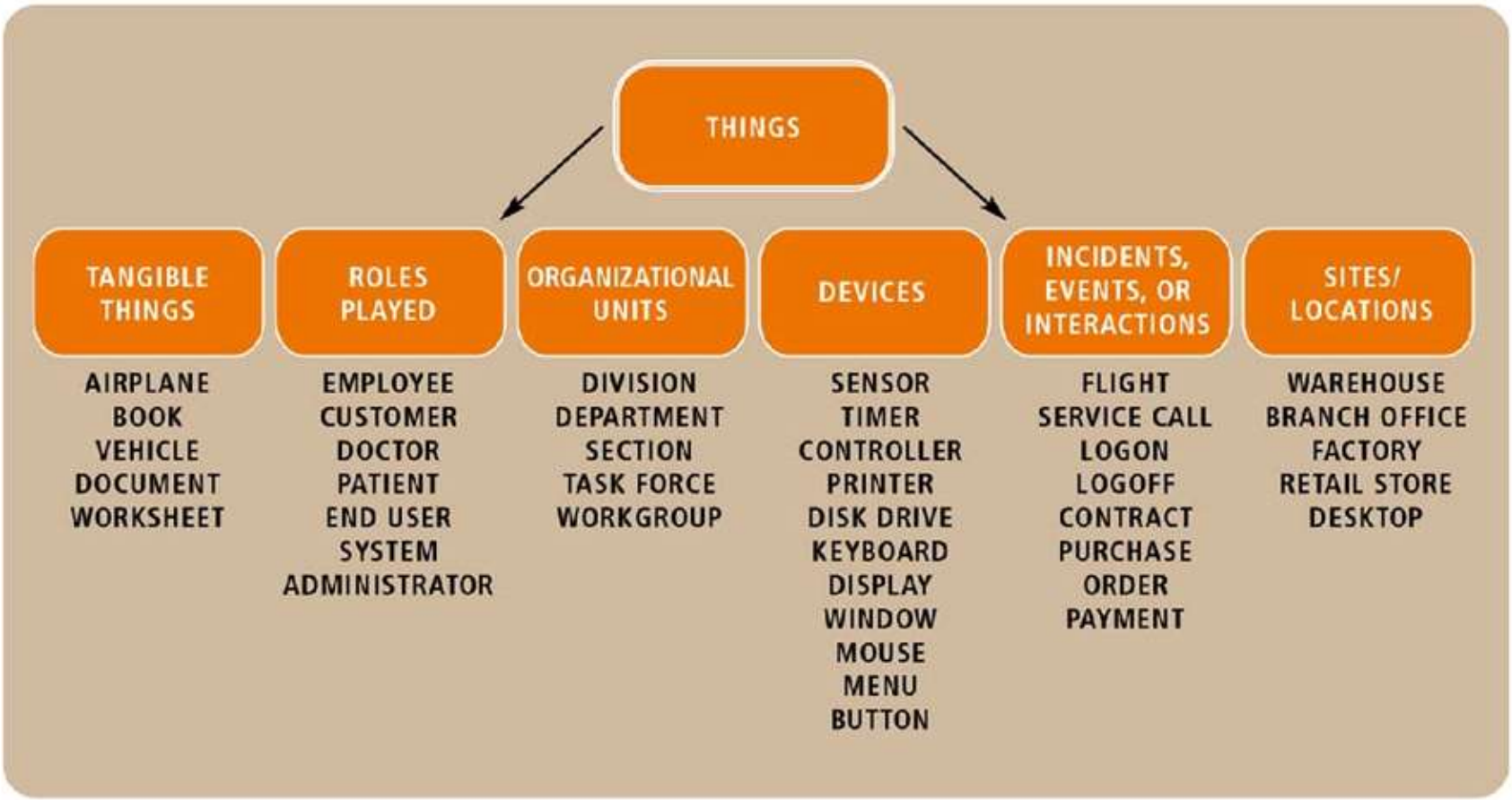
Identifying Operations ‘methods’

- Look for verbs in the SRS (System Requirements Specifications) document

- Look for **VERBS** in use cases descriptions

- A **VERB** may be
  - translated to an **operation** or set of operations
  - A method is the code implementation of an operation.

Objects


THINGS
TANGIBLE THINGS
ROLES PLAYED
ORGANIZATIONAL UNITS
DEVICES
INCIDENTS, EVENTS, OR INTERACTIONS
SITES/ LOCATIONS
AIRPLANE
BOOK
VEHICLE
DOCUMENT
WORKSHEET
EMPLOYEE
CUSTOMER
DOCTOR
PATIENT
END USER
SYSTEM
ADMINISTRATOR
DIVISION
DEPARTMENT
SECTION
TASK FORCE
WORKGROUP
SENSOR
TIMER
CONTROLLER
PRINTER
DISK DRIVE
KEYBOARD
DISPLAY
WINDOW
MOUSE
MENU
BUTTON
FLIGHT
SERVICE CALL
LOGON
LOGOFF
CONTRACT
PURCHASE
ORDER
PAYMENT
WAREHOUSE
BRANCH OFFICE
FACTORY
RETAIL STORE
DESKTOP

Objects

An object is a thing:
- student;
- transaction;
- car;
- customer account;
- employee;
- complex number;
- spreadsheet table;
- spreadsheet cell;
- document;
- paragraph;
- GUI Combo box
- GUI button. . . and so on.

Class and Class diagram

- Class naming: Use **singular** names
  - because each class represents a generalized version of a singular object.

- **Class diagrams are at the core of OO Eng.**

Class and Class diagram

- Things naturally fall into categories (computers, automobiles, trees...).
- We refer to these categories as classes.

- An object class is an abstraction over a set of objects with common:
  - **attributes (states)**
  - **and the services (operations) (methods)**

  provided by each object

- Class diagrams provide the representations used by the developers.

CRC ‘Class Responsibility and Collaborator’

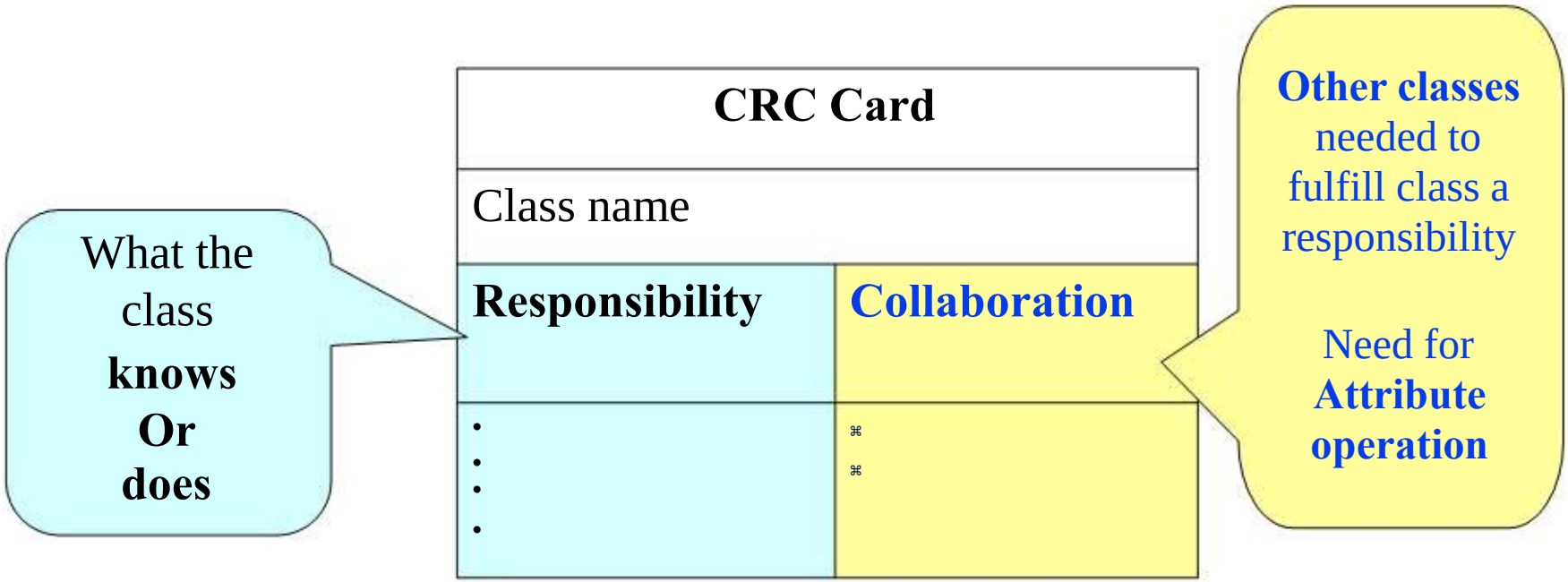
- CRC card

- Class Responsibility:
  - What the class **knows: attributes**
  - What the class **does: services (operations / methods)**

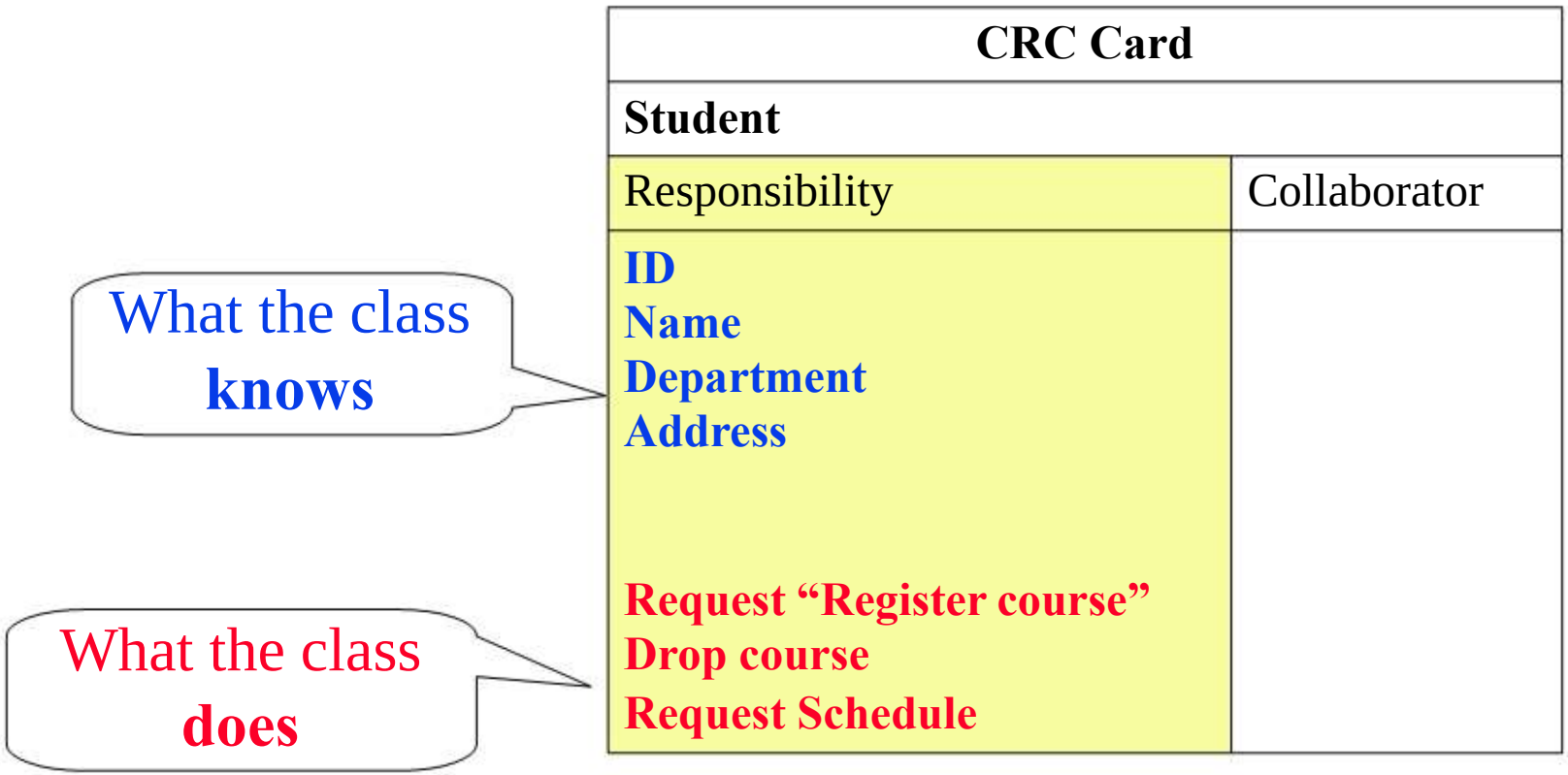
CRC ‘Class Responsibility and Collaborator’

- Class Collaboration:
  - Request for **information** from another class (what the other class knows as **attributes**)
  - Request another class to **do some thing** (what the other class does as **operation**)

CRC Card


What the class **knows Or does**
CRC Card
Class name
Responsibility
Collaboration
Other classes needed to fulfill class a responsibility
Need for **Attribute operation**

CRC - Class Responsibility



CRC Card	
Student	
Responsibility	Collaborator
ID **Name** **Department** **Address** **Request "Register course"** **Drop course** **Request Schedule**	

CRC - Class Collaborator

- Sometimes a **class A** has a responsibility to fulfill, but **not have enough** information to do it.

- So class A needs help from another class
- See next example

CRC - Class Collaborator

- For example, as you see in students register in courses.
  - To do this, a student needs to know if a spot is available in the course and, if so, he then needs to be added to the course. However, students only have information about themselves (their names and so forth), and not about courses.
  - What the student needs to do is collaborate/interact with the card labeled ***Course*** to sign up for a course.
  - Therefore, ***Course*** is included in the list of collaborators of *Student*.

CRC - Class Collaborator

Student	
Responsibility	**Collaborator**
ID Name Department Address	
Check course availability →	**Course** (Attribute: availability)
Request "Register course" →	**Course (**Operation: increment number of registered student)
Drop course →	**Course (Operation:** decrement number of registered student)
Request Schedule	-----

Collaborator **Course**:

Class **Course** is needed to fulfill class **Student** responsibilities

Need for **Attribute** and/or **operation**

CRC - Class Collaborator

- Collaboration takes one of two forms:
  - A request for information
  - or a request to do something.

- **Example Alternative 1:**
- The card ***Student*** requests an indication from the card **C*ourse*** whether a space is available, a request for **information**.
- ***Student*** then requests to be added to the **C*ourse*** , a request to do something.

- **Alternative 2**: Another way to perform this logic, however, would have been to have ***Student*** simply request **C*ourse*** to enroll himself (***Student***) into itself (**C*ourse)***. Then have **C*ourse*** do the work of determining if a seat is available and, if so, then enrolling the student and, if not, then informing the student that he was not enrolled.

Types of Operations

• Operations can be classified into four types, depending on the kind of service requested by clients:

1. **constructor:**
   creates a new instance of a class
2. **query**:
   is an operation without any side effects; it ccesses the state of an object but does not alter the state
3. **update:**
   An operation that alters the state of an object.

Class diagrams

- Shows relationship between classes
- A class diagram may show:

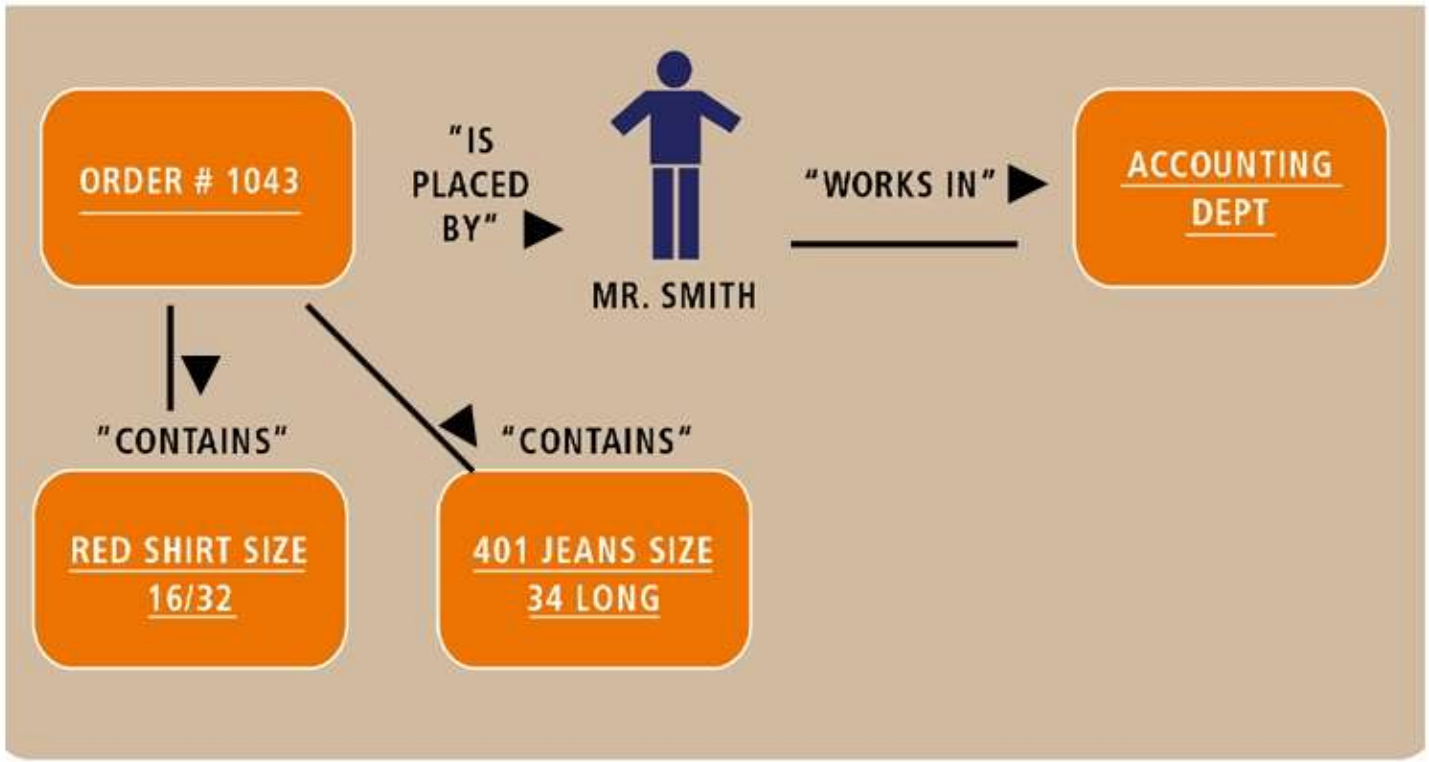
Relationship	
Generalization (inheritance)	 **"is a"** **"is a kind of"**
Association (dependency)	**does** ▶ **"Who does What"** **"uses"**
Aggregation	 **"has"** **"composed of"**
Composition: Strong aggregation	

Association, aggregation and composition

- When considering the 3 relationships, association, aggregation and composition,
  - the most **general relationship is association**,
  - followed by aggregation
  - and, finally, composition.

Association between classes

Who does What



Multiplicity of Relationships

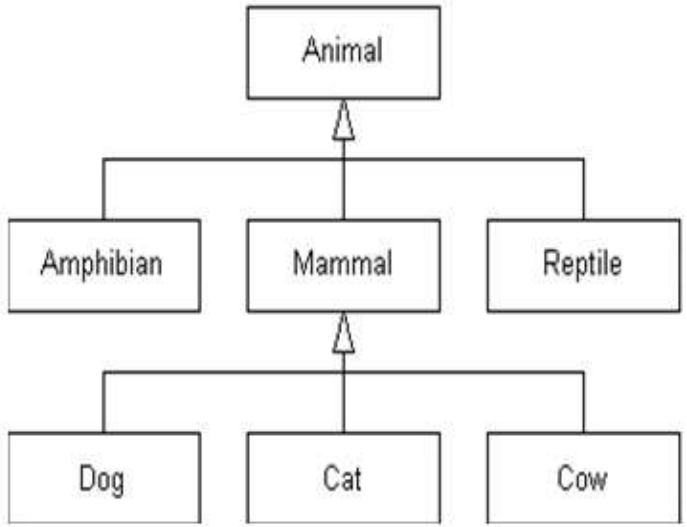
Mr. Jones has placed no order yet, but there might be many placed over time. → multiplicity is zero or more—optional association

A particular order is placed by Mr. Smith. There can't be an order without stating who the customer is. → multiplicity is one and only one—mandatory association

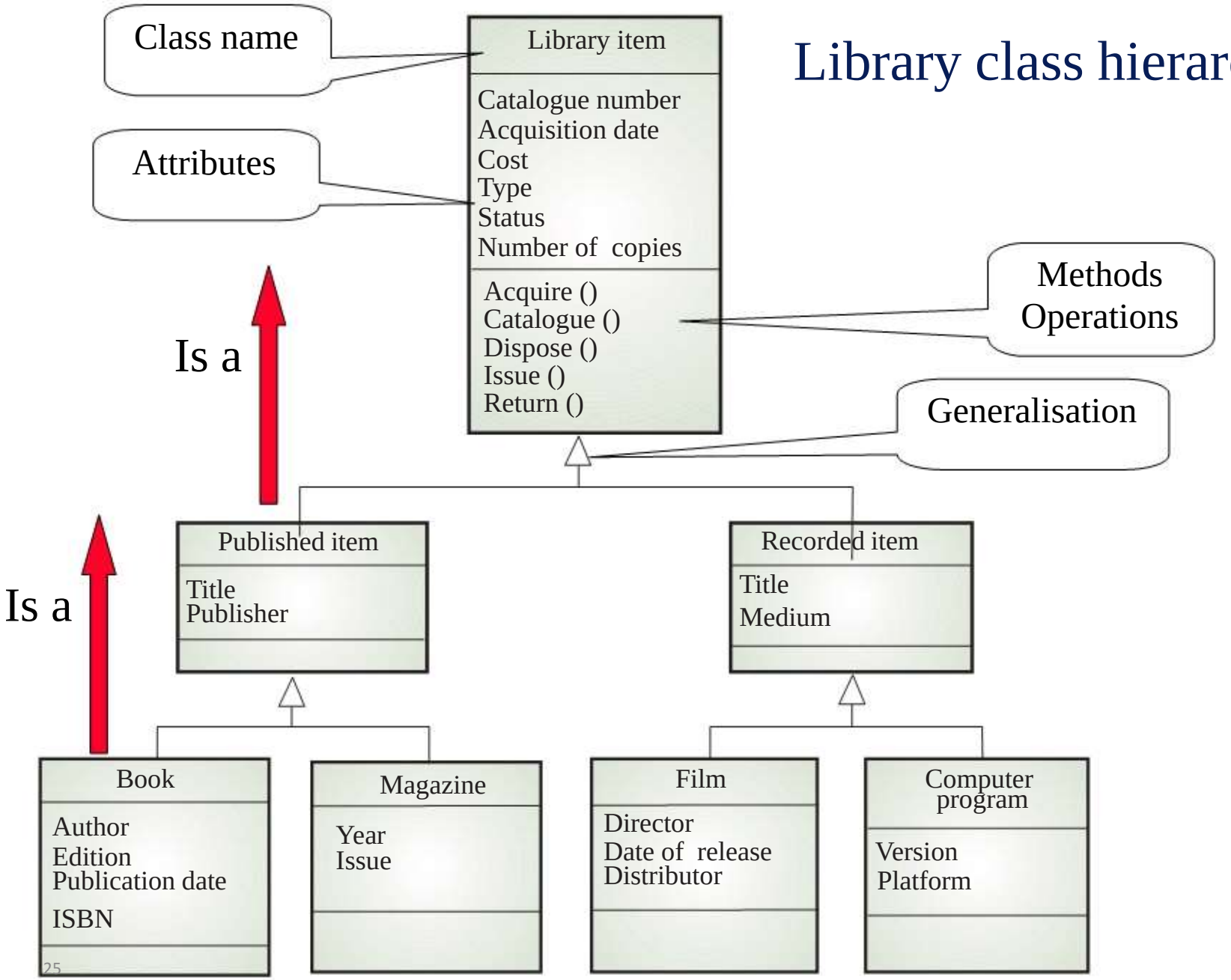
An order contains at least one item, but it could contain many items. → multiplicity is one or more—mandatory association

Inheritance: ***is a*** *"is a kind of"*

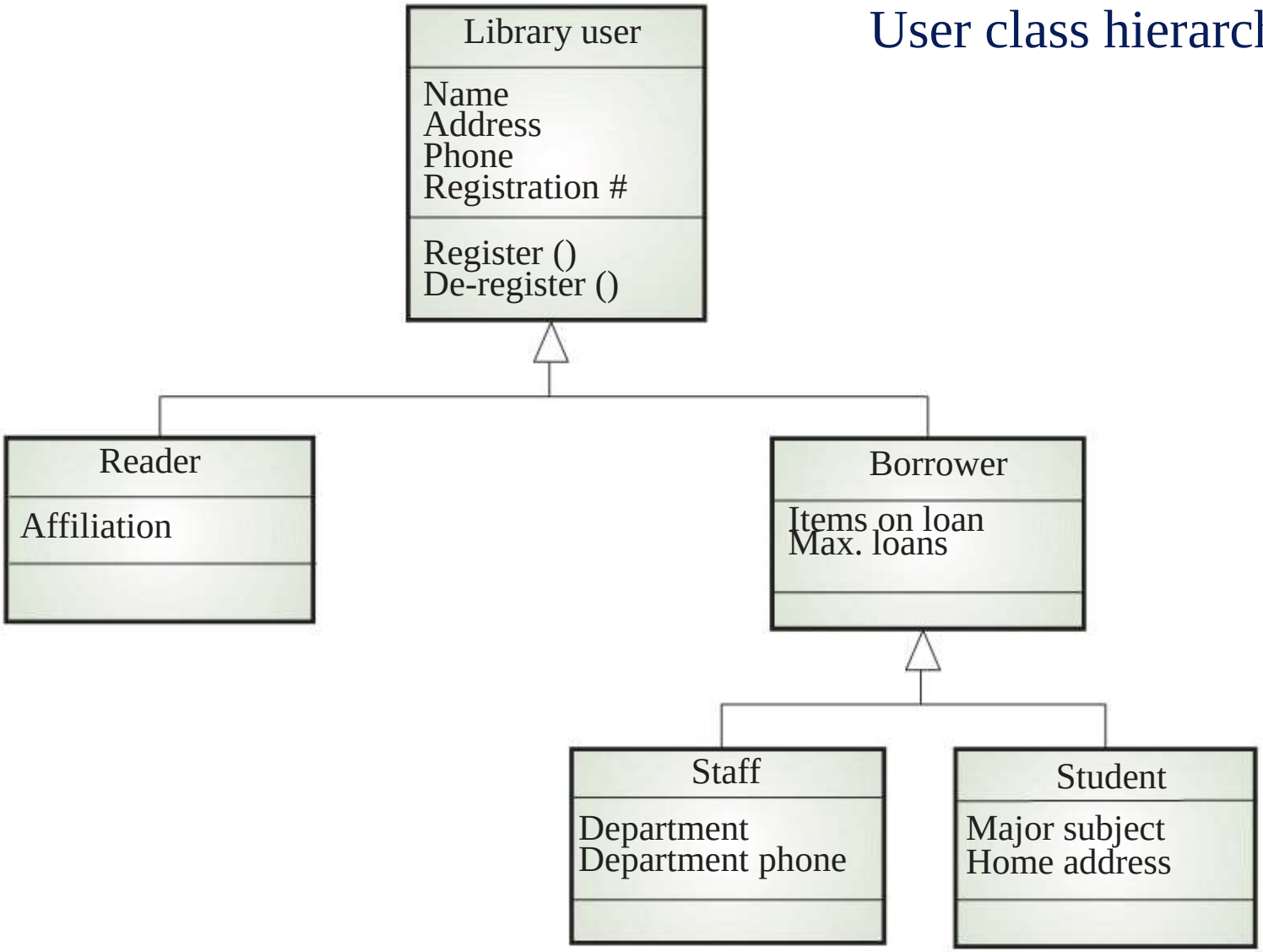
- ***is a*** association.
- Child class 'subclass' can inherit attributes and operations from parent class'superclass'.
- *Example: An inheritance hierarchy in the animal kingdom*



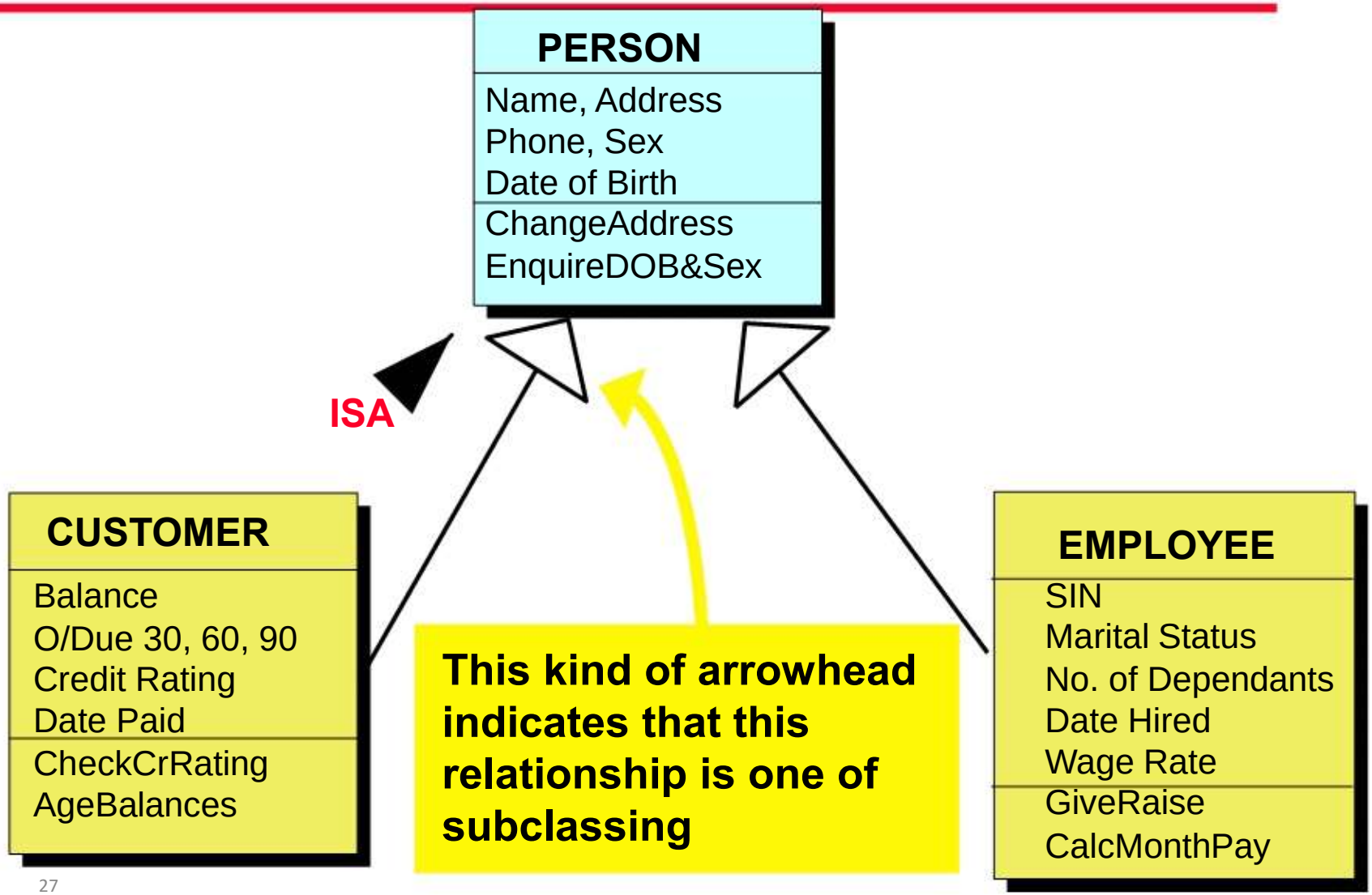
Library class hierarchy



User class hierarchy



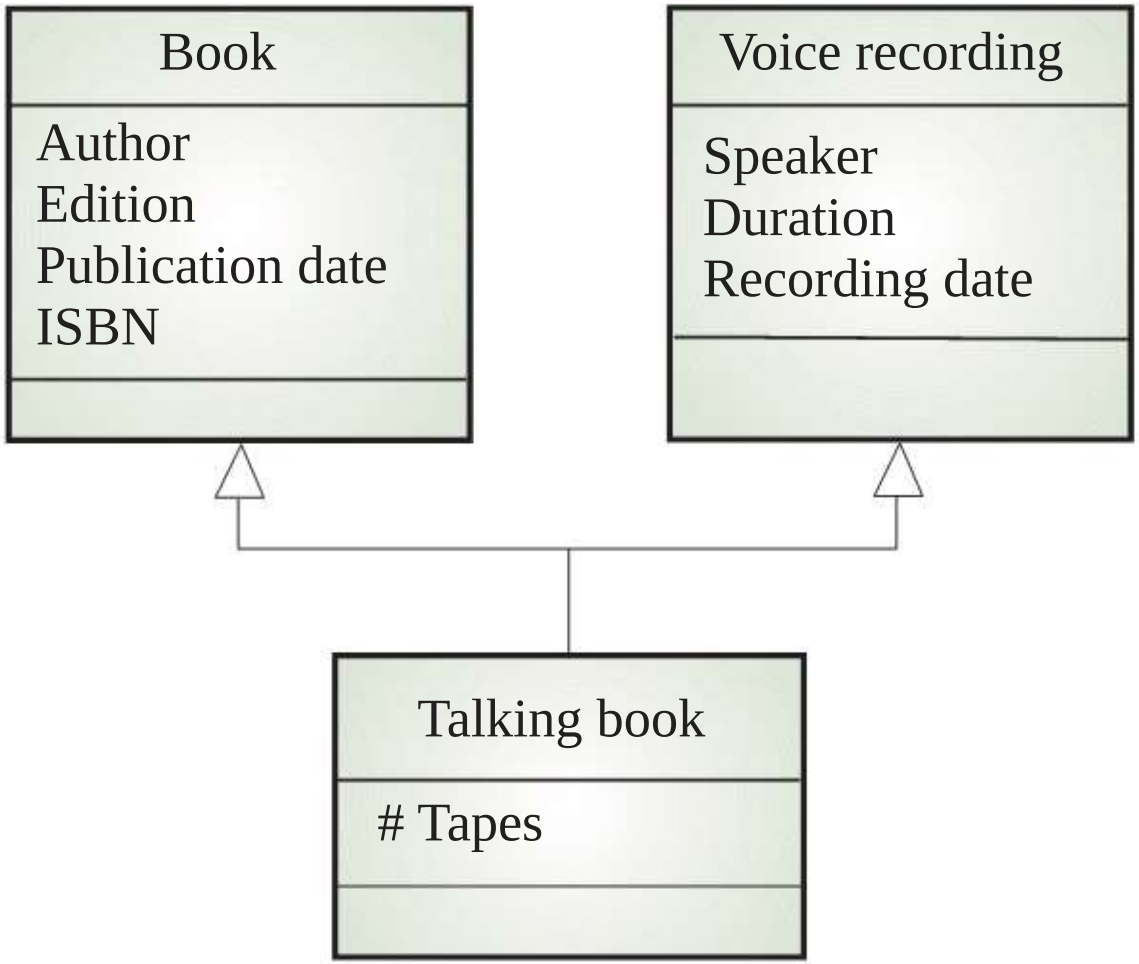
Hierarchy Diagram (*UML notation*)



Multiple inheritance

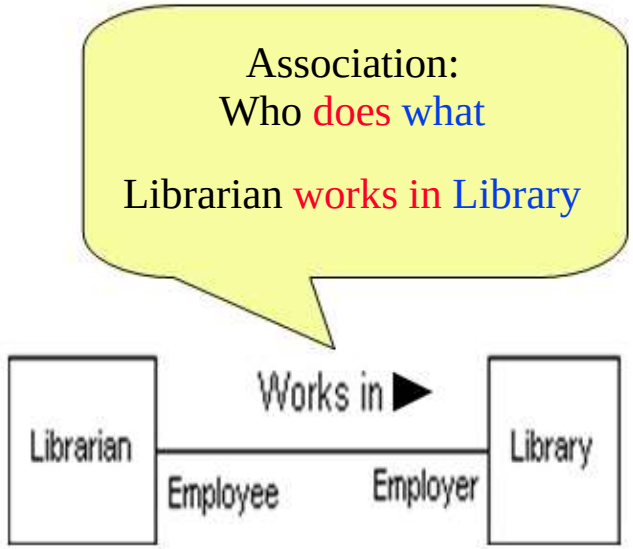
- Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes
- Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics
- Makes class hierarchy reorganisation more complex
- Java does not support multiple inheritance

Example: Multiple inheritance
The talking book



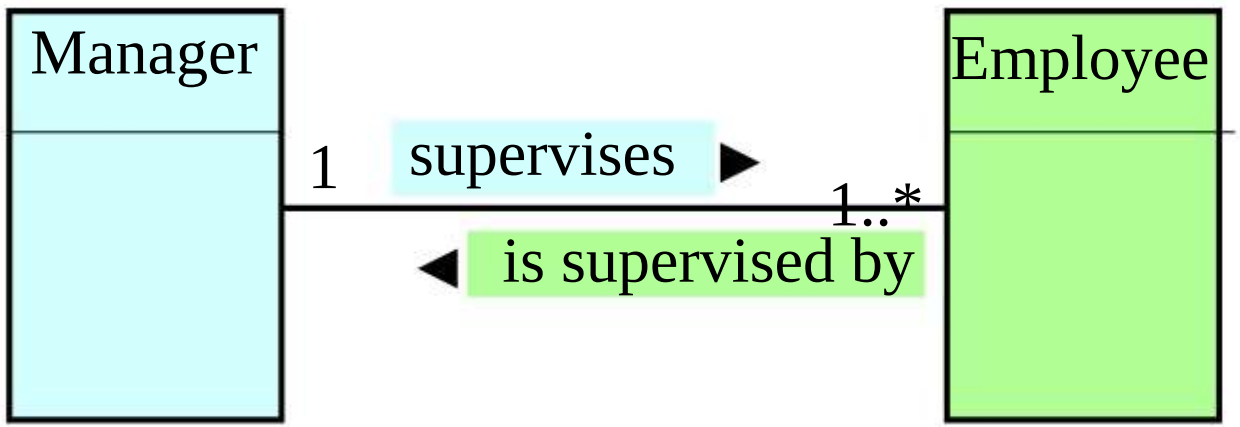
UML: **Associations** of regular classes

- **Who does what relationship**
- When classes are connected together conceptually, that connection is called an association



Associations of regular classes - **Who does what**

- A manager supervises 1..* employees
- An employee is supervised by 1 manager



Multiplicity of an Association

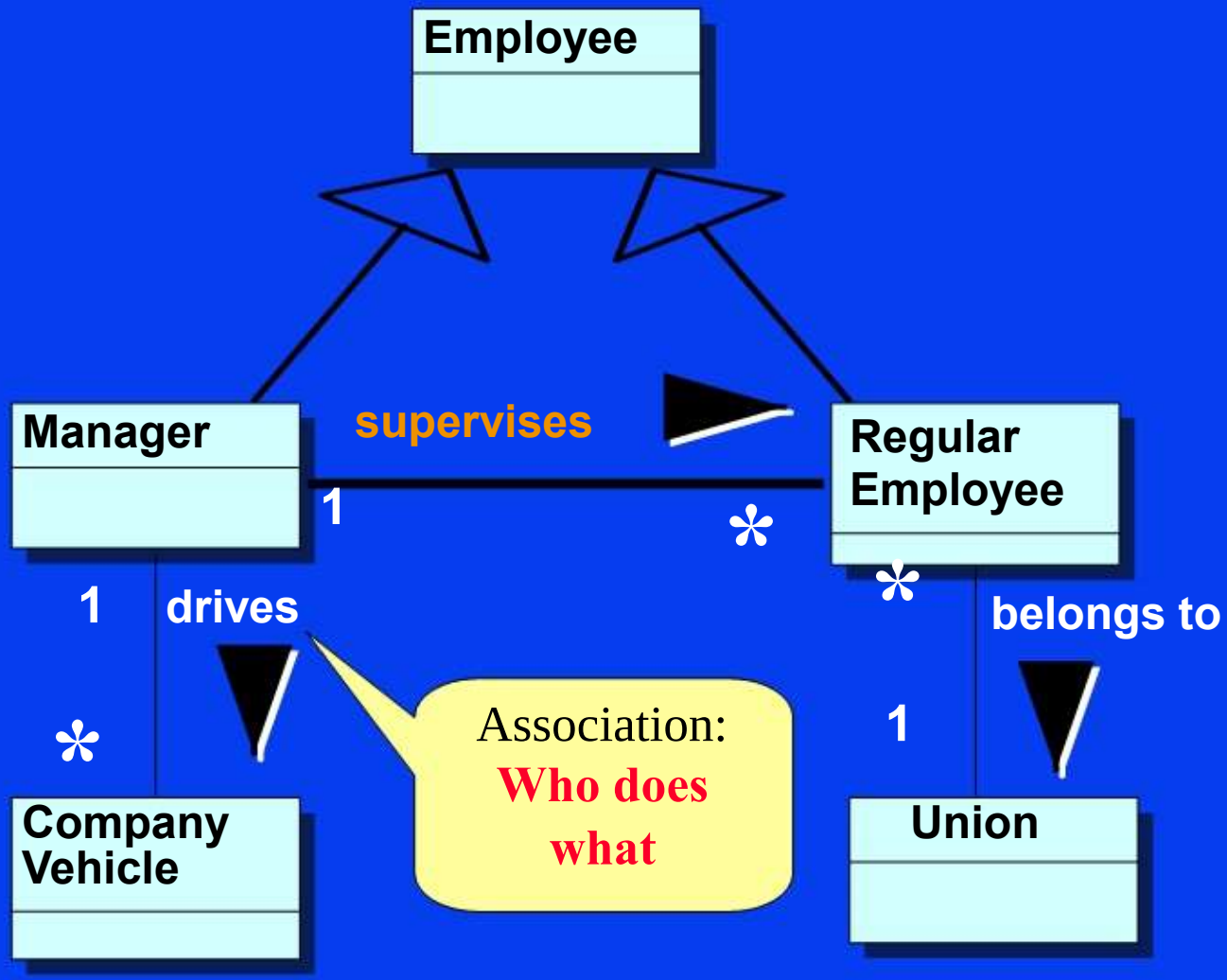
- Shows the number of objects from one class that relate with a number of objects in an associated class.

UML: Multiplicity

One class can be relate to another in a:

- one-to-one
- one-to-many
- one-to-one or more
- one-to-zero or one
- one-to-a bounded interval (one-to-two through twenty)
- one-to-exactly n
- one-to-a set of choices (one-to-five or eight)
- The UML uses an asterisk (*) to represent ***more*** and to represent ***many*** .

Association and Inheritance.



OO: Visibility of attributes or operations

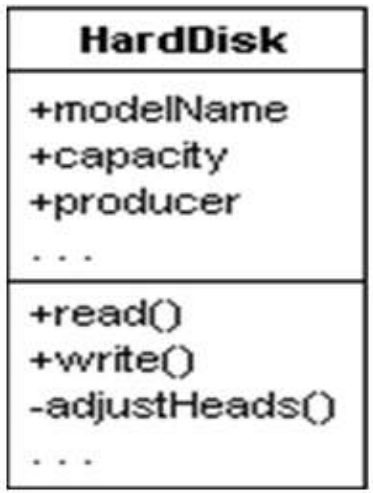
- Visibility: specifies the extent to which other classes can use a given class's attributes or operations.

Three levels of visibility:

- \+ **: public level (**usability extends to other classes)
- \# **: protected level (**usability is open only to classes that inherit from original class)
- \- **: private level (**only the original class can use the attribute or operation)

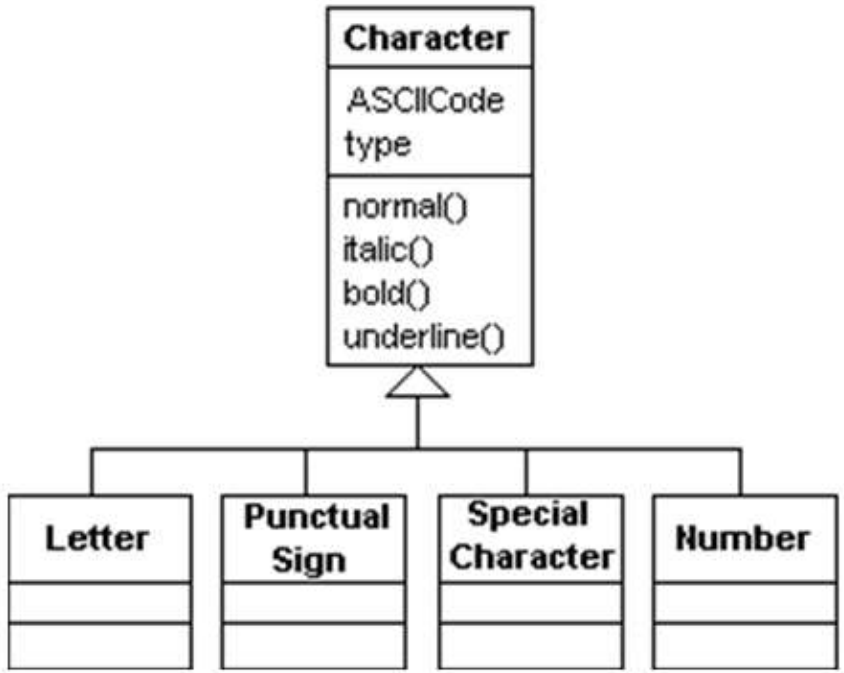
OO: Visibility

Ex: *Public and private operations in a Hard Disk*

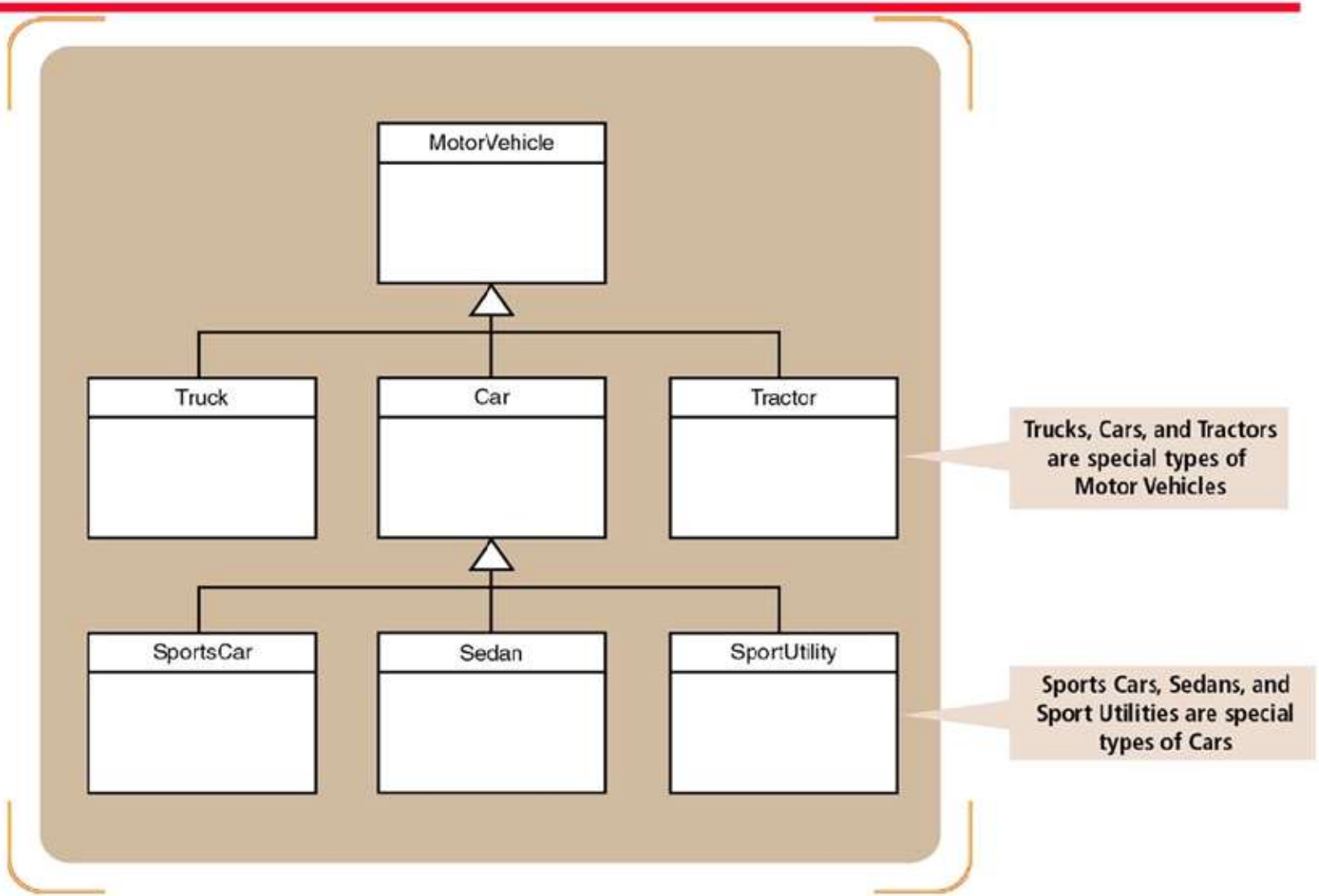


Ex: *The character hierarchy*

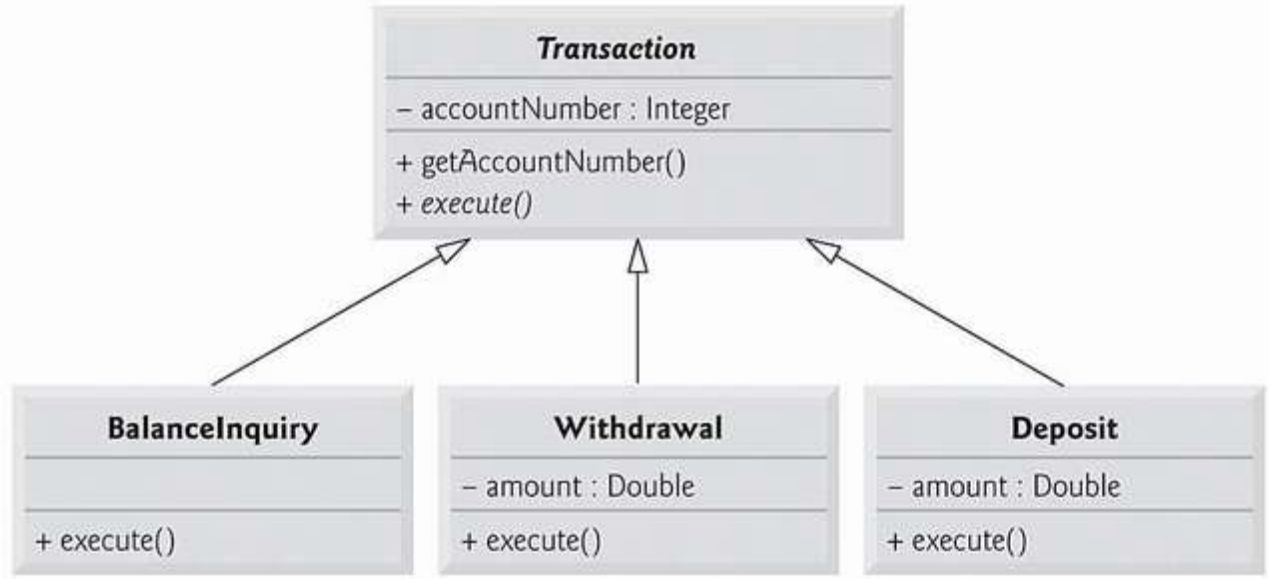
- The **Character class** will have ***ASCIIcode*** and ***type*** as attributes (type tells the type of the character - normal, italic, bold or underline), and ***normal()***, ***bold()***, ***italic()*** and ***underline()*** as operations. The Character class children will be: **Letter**, **PunctualSign**, **SpecialCharacter** and **Number**.

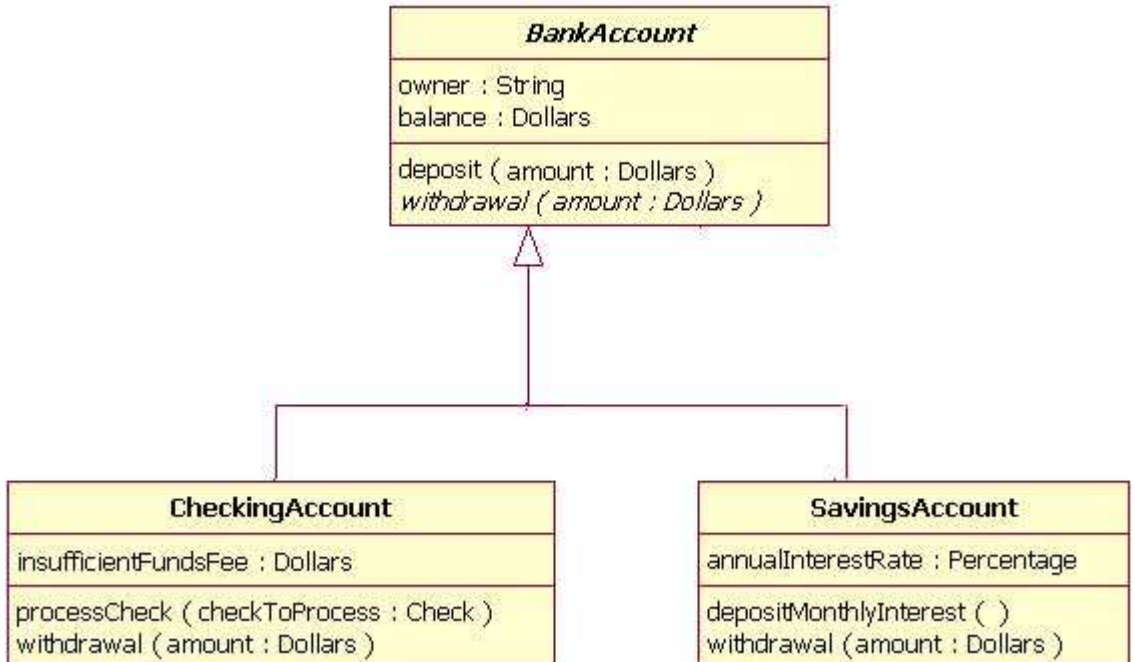


Ex: Generalization/Specialization Hierarchy Notation for Motor Vehicles



Ex: Generalization/Specialization Hierarchy





Object Aggregation ◊

- **Has-a** relationship
- Structural: **whole/part**
- **Peer** relationship
  - Whole & parts objects **can exist independently**
- A special form of association

Object Aggregation: **Peer** relationship ◊

- Whole & parts objects **can exist independently**
- Example: a bank (whole) has customers (as parts)
- Deleting a bank does not cascade deleting customers
- Customers can move to another bank

- Programming: whole contains an array of parts

Object Aggregation ◊

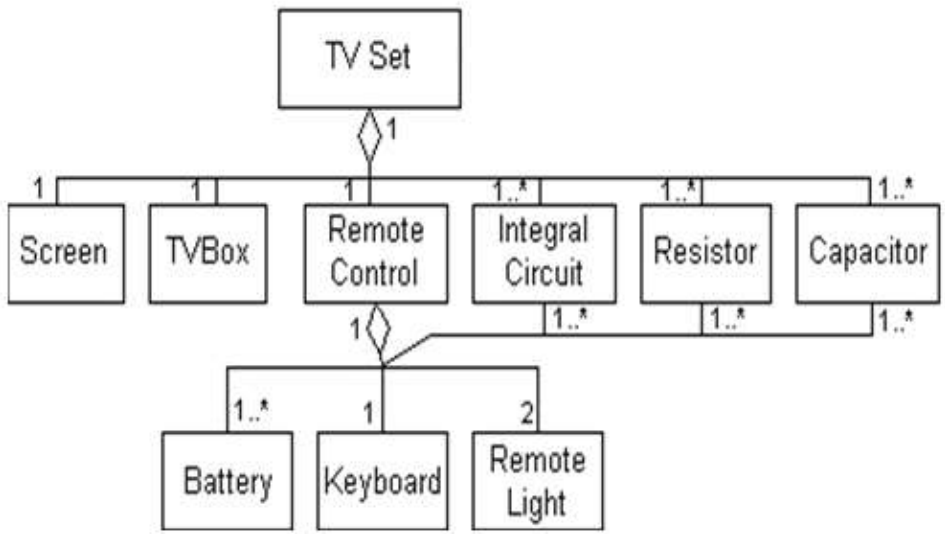
- Aggregation model shows how classes (which are collections) are composed of other classes.
- Similar to the part-of relationship in semantic data models.

- A line joins a whole to a part (component) with an open diamond on the line near the whole.

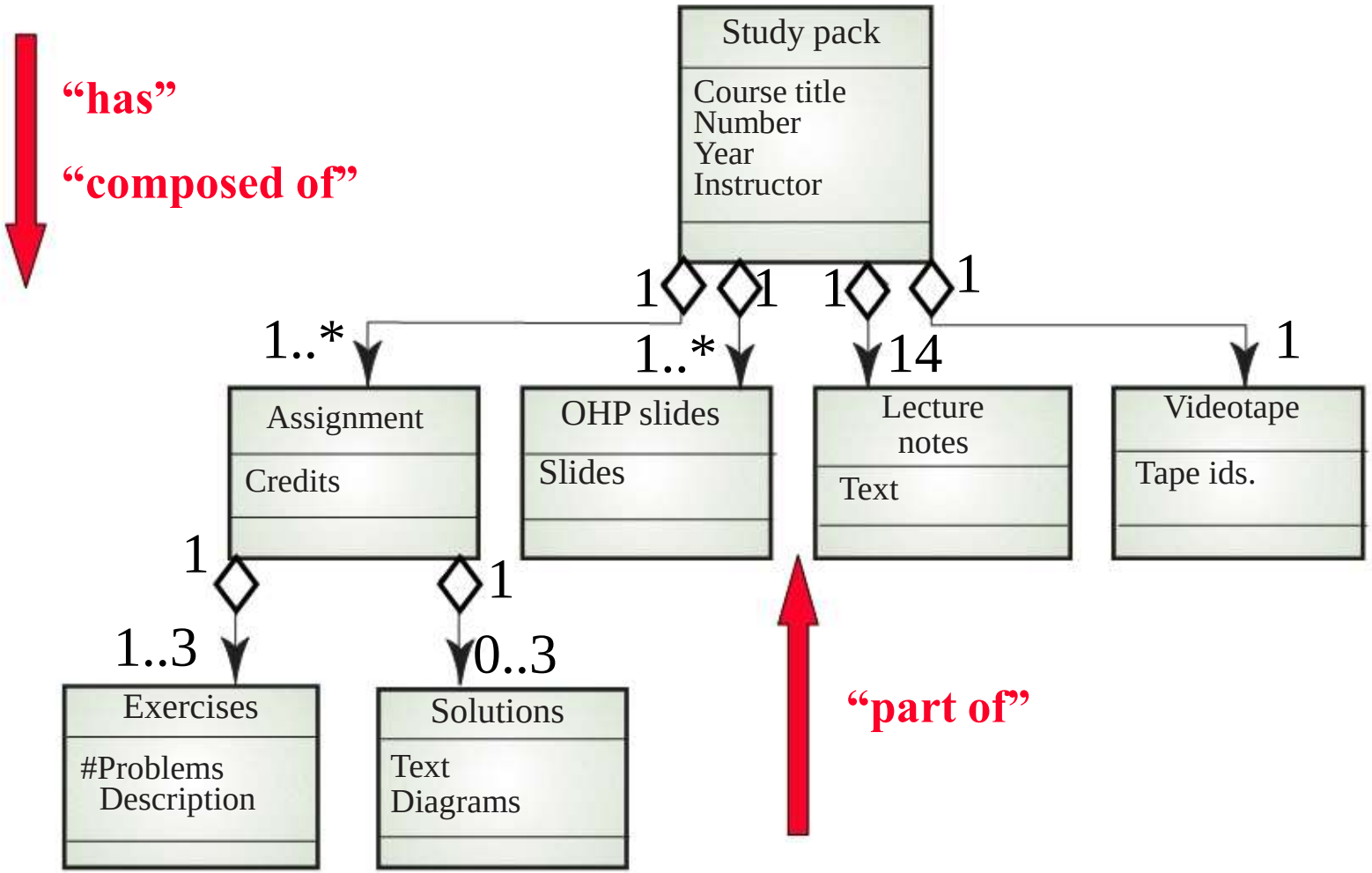
Object Aggregation ◊

Example: An aggregation association in the TV Set system

- Every TV has a TV box, screen, speaker(s), resistors, capatitors, transistors, ICs... and possibly a remote control.
- **Remote control** can have these parts: resistors, capatitors, transistors, ICs, battery, keyboard and remote lights.



Object aggregation ◊



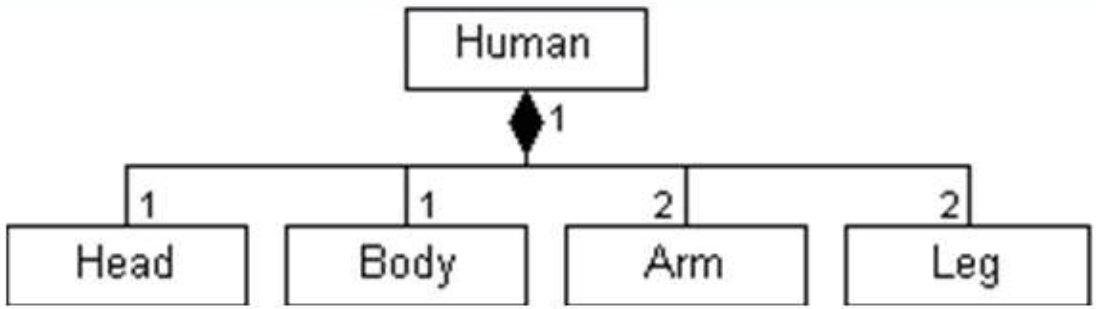
Composition ♦

- A composite is a strong type of aggregation.
- Each component in a composite can belong to just **one whole** .
- The symbol for a composite is the same as the symbol for an aggregation except the diamond is **filled**.

Composition ♦ - Example 1

- Human's outside:

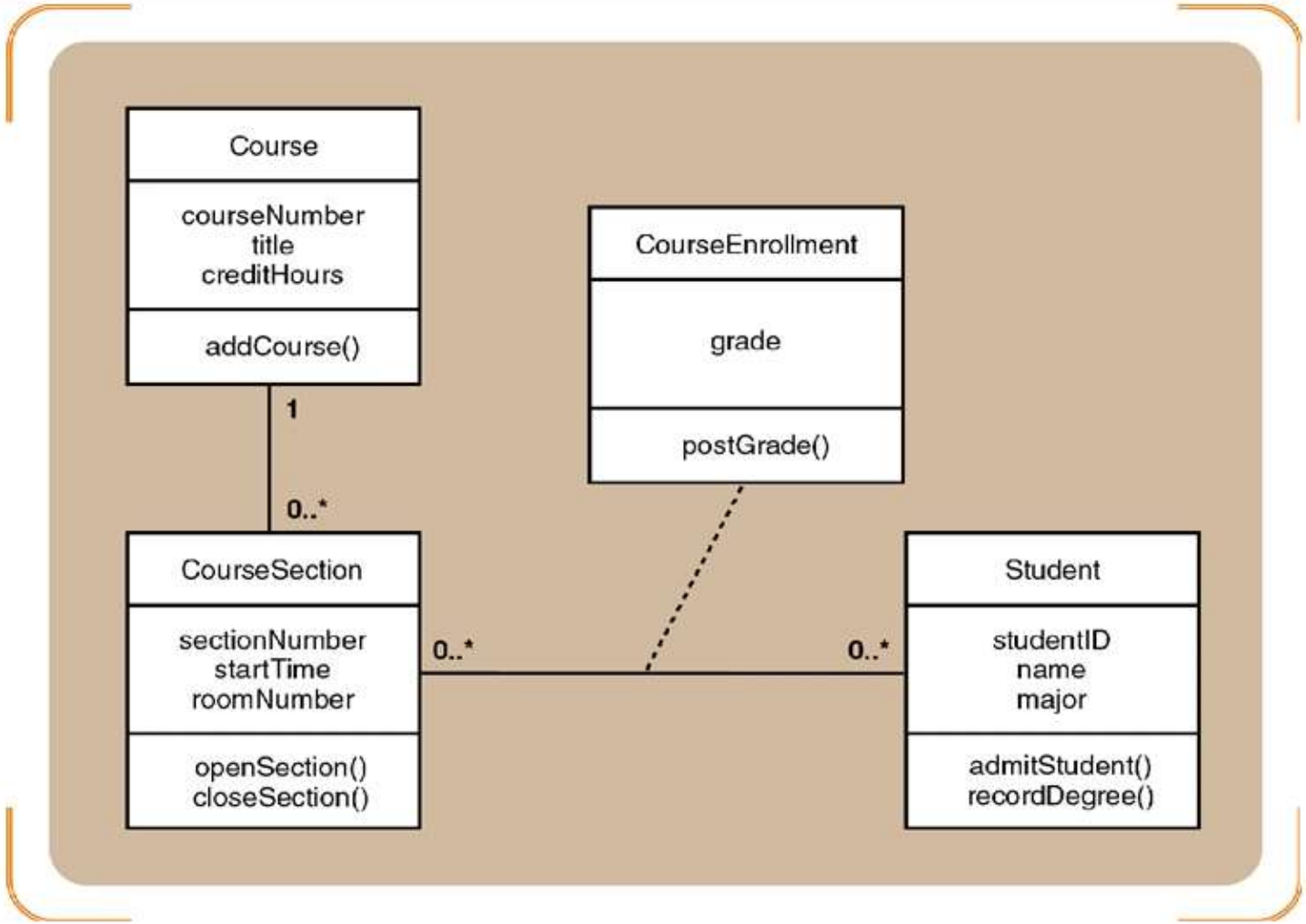
  Every person has: head, body, arms and legs.
- *A composite association. In this association each component belongs to exactly one whole .*
- Whole & parts objects **can NOT exist independently**



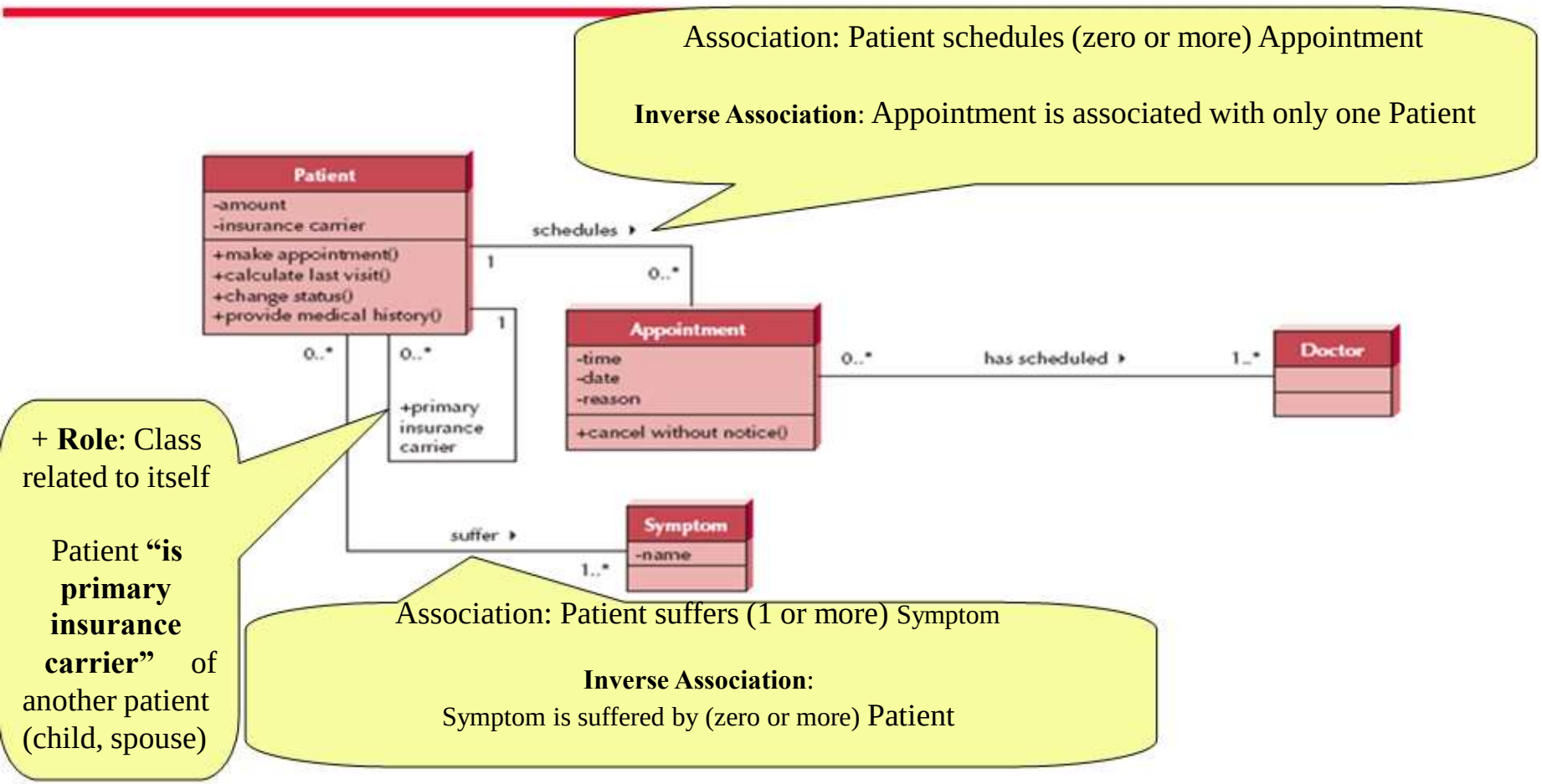
Composition ♦ - Example 2

- A bank (whole) has many branches (parts)
- Branches can not exist independently of the whole (parts objects **can NOT exist independently)**
- Deleting a bank (whole) cascades deleting branches (parts)
- But, if a branch (part) is deleted, the bank (whole) may remain

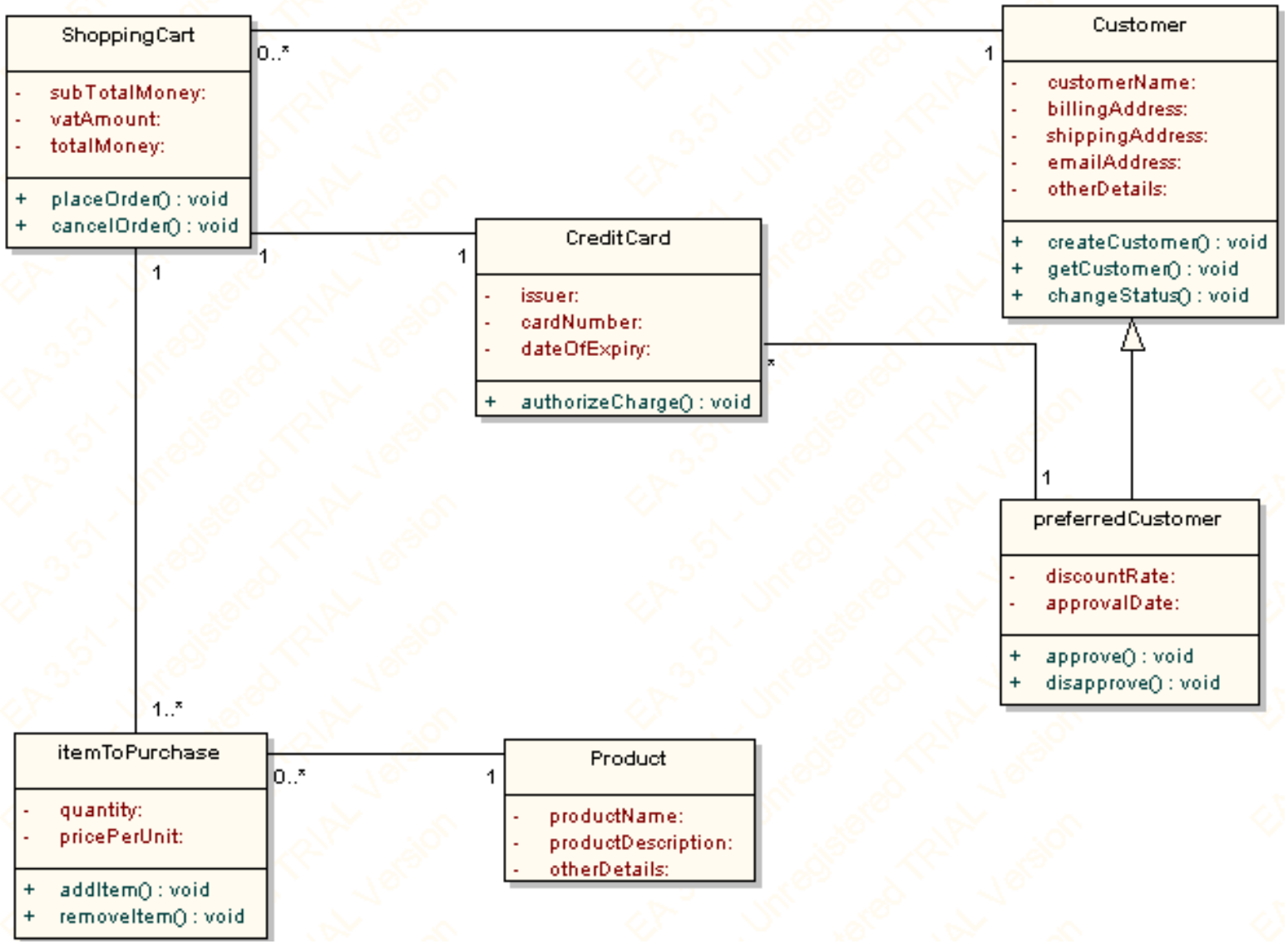
University Course Enrollment Design Class Diagram (With Methods)



Class diagram - Example
Reflexive association



Example:


ShoppingCart
- subTotalMoney:
- vatAmount:
- totalMoney:
+ placeOrder() : void
+ cancelOrder() : void
0..*
1
Customer
- customerName:
- billingAddress:
- shippingAddress:
- emailAddress:
- otherDetails:
+ createCustomer() : void
+ getCustomer() : void
+ changeStatus() : void
1
1
CreditCard
- issuer:
- cardNumber:
- dateOfExpiry:
+ authorizeCharge() : void
*
1
1
preferredCustomer
- discountRate:
- approvalDate:
+ approve() : void
+ disapprove() : void
1..*
itemToPurchase
- quantity:
- pricePerUnit:
+ addItem() : void
+ removeItem() : void
0..*
1
Product
- productName:
- productDescription:
- otherDetails: