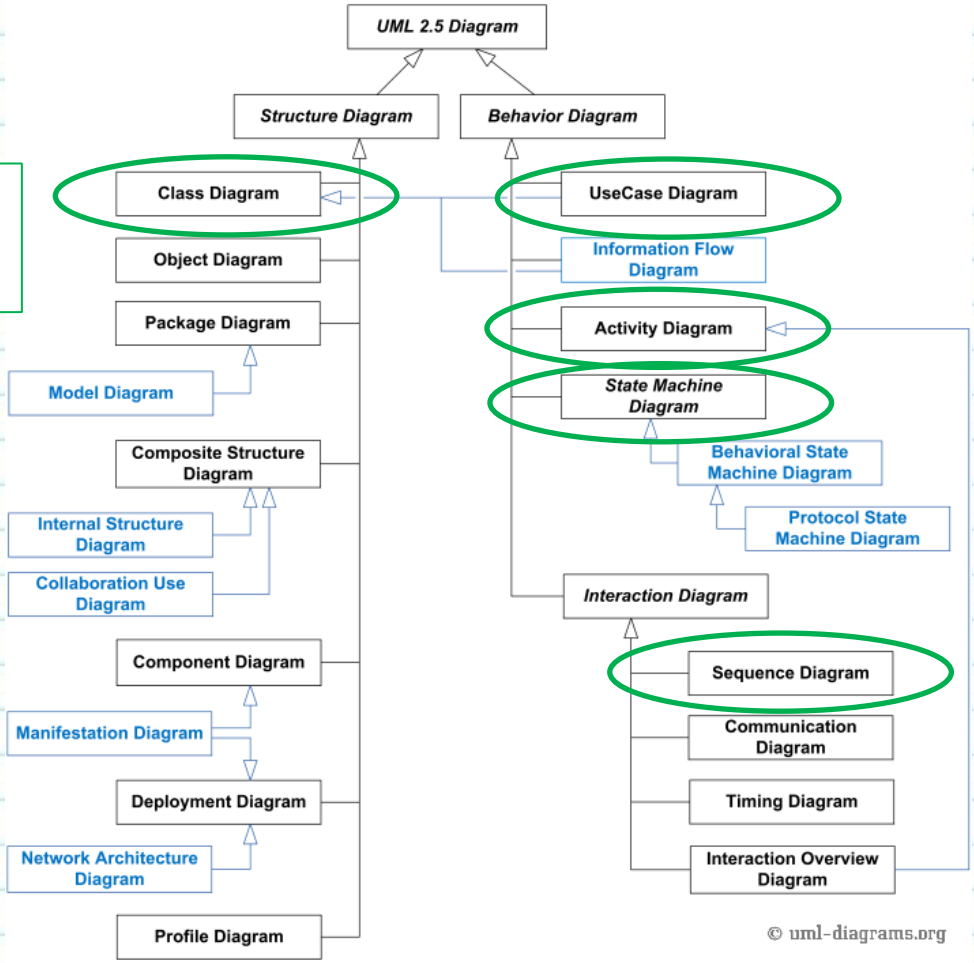
Chapter 5 – Domain Modeling

Dr John H Robb, PMP
UT Arlington
Computer Science and Engineering

Domain Modeling and UML

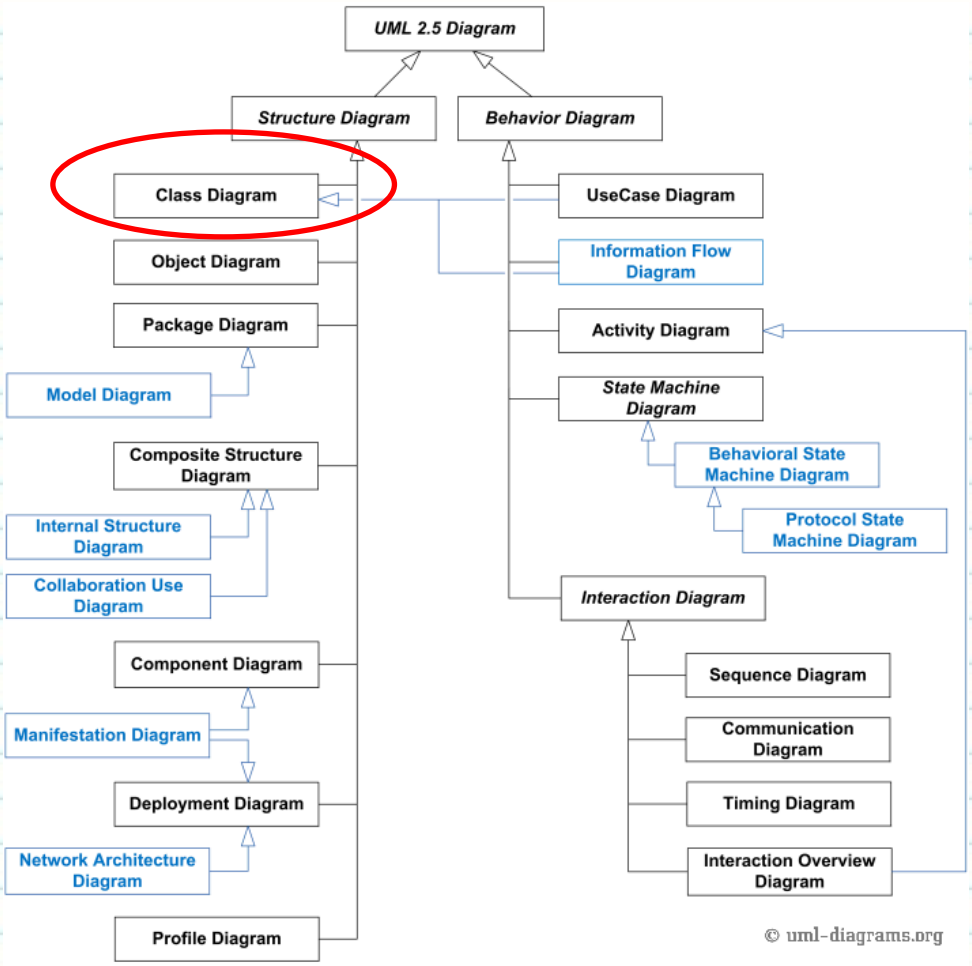
- The Domain model represents our first use of a UML structure in the class so it's important to look at the overall UML strucuture
- UML has two sides - behavior (dynamic) and structure (static)

We will learn these UML diagrams this semester



Domain Modeling and UML (cont.)

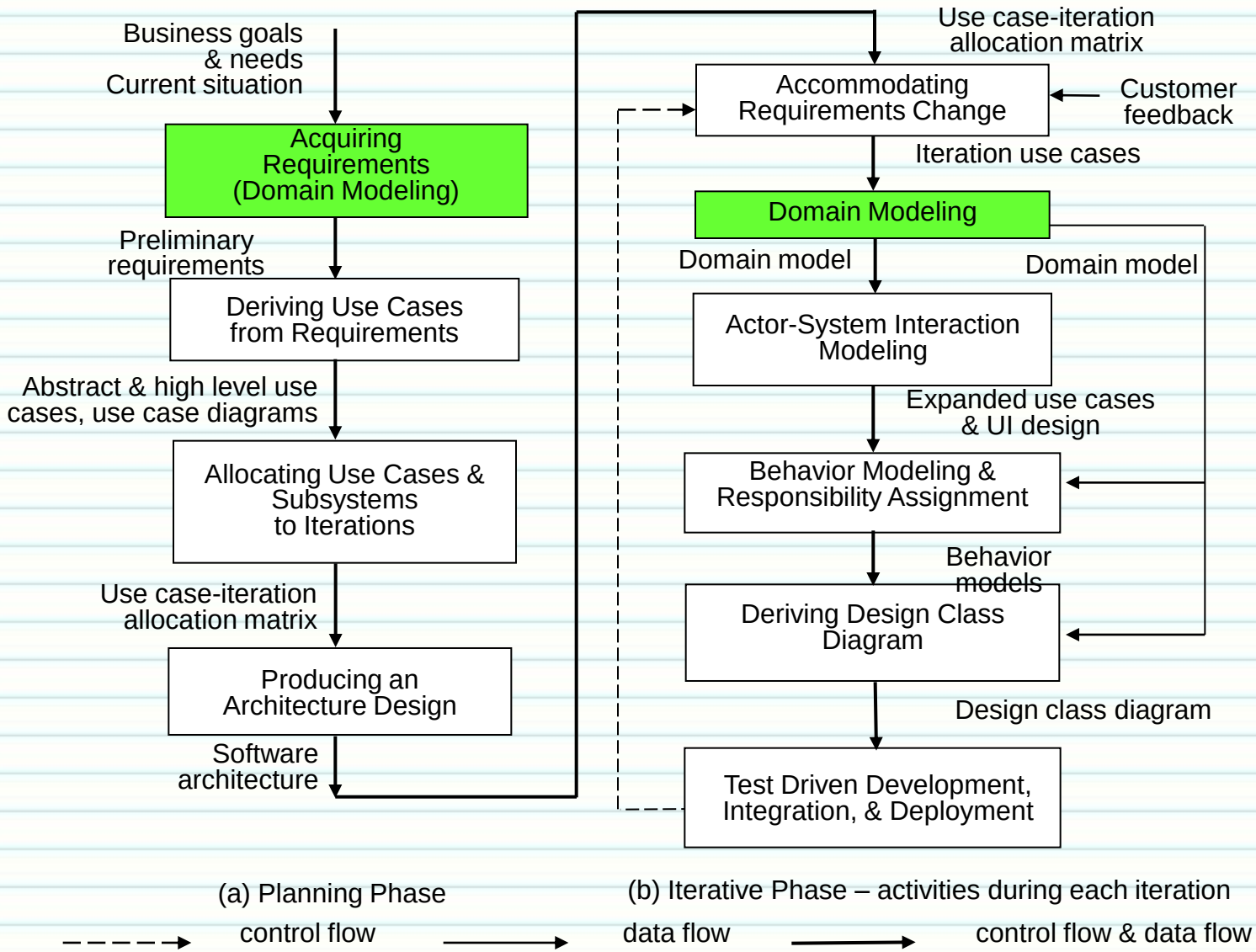
- The Domain model and Domain diagram are not officially part of UML 2.5 - the domain model is a top-level Class diagram which is in UML 2.5



Key Takeaway Points

- Domain modeling is a conceptualization process to help the development team understand the application domain.
- Five easy steps: collecting information about the application domain; brainstorming; classifying brainstorming results; visualizing the domain model using a UML class diagram; and performing inspection and review.

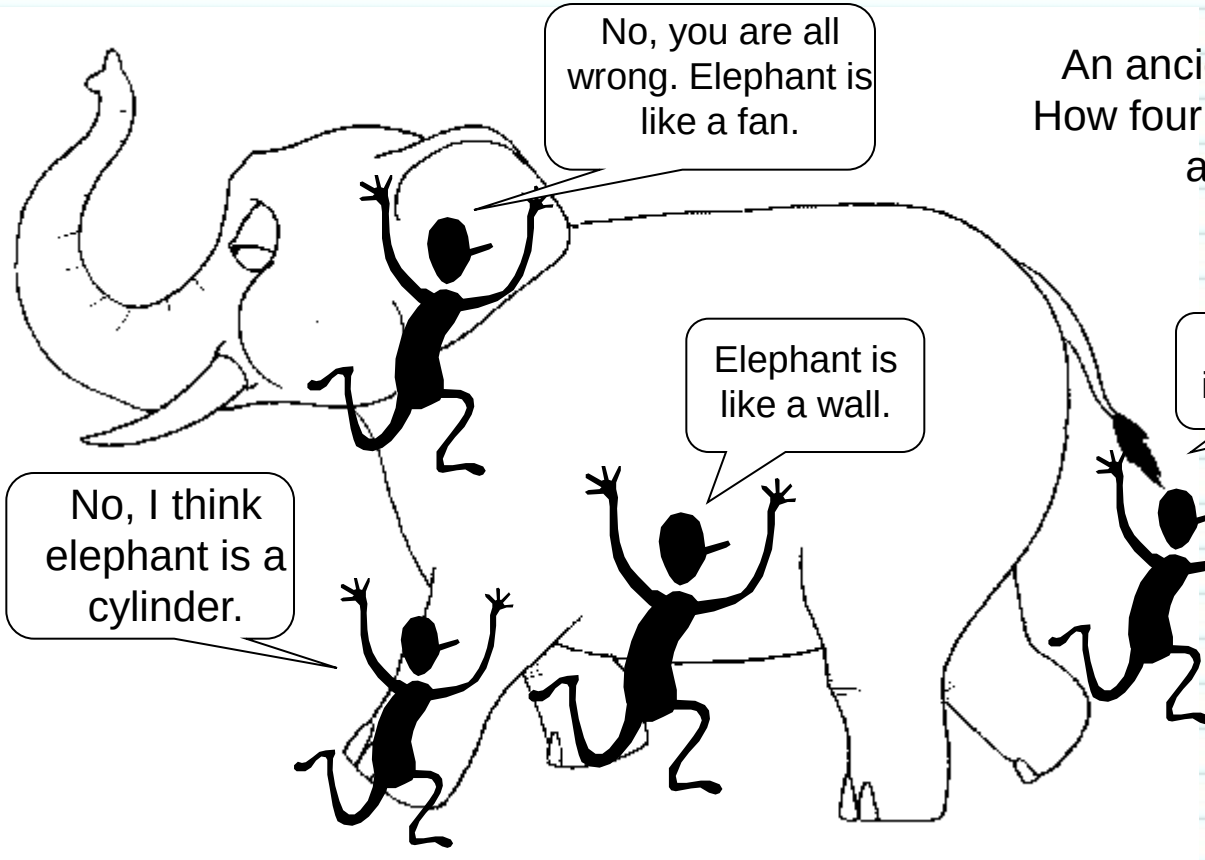
Domain Modeling in the Methodology Context



What Is a Model?

- A conceptual representation of something.
- A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics. (Dictionary Definition)

Why Do We Need Model?



An ancient Chinese story. How four blind men perceive an elephant.

We perceive the world differently due to differences in backgrounds and viewpoints. Modeling facilitate collective understand of the application.

Why Do We Need Model?

Because the team members and the users need to communicate their perception about a piece of reality. A model facilitates the team members and users to communicate their perception and design ideas.

Why Do We Need Models?

Because we need models during the maintenance phase to perform enhancement maintenance.

Domain Modeling

- What:
  - A process that helps the team understand the application or application domain.
  - It enables the team to establish a common understanding.
- Why:
  - Software engineers need to work in different domains or different projects. They need domain knowledge to develop the system.
  - Software engineers come from different backgrounds, which affect their perception of the application domain.
- How:
  - Collect domain information, perform brainstorming and classification, and visualize the domain knowledge using a UML class diagram

Domain Modeling

- A domain model defines application domain concepts in terms of classes, attributes and relationships.
- The construction of the domain model
  - helps the development team or the analyst understand the application and the application domain
  - lets the team members communicate effectively their understanding of the application and the application domain
  - improves the communication between the development team and the customer/user in some cases
  - provides a basis for the design, implementation and maintenance
- Domain model is represented by UML class diagrams (without showing the operations).

Domain Modeling in the OO Paradigm

- The OO paradigm views the real world as consisting of
  - objects
  - that relate to each other, and
  - interact with each other

- The basic build blocks and starting point are objects.

Important Object-Oriented Concepts

Class --- a class is a type

- an abstraction of objects with similar properties and behavior
- an intentional definition of a collection of objects

Attribute --- define properties of class of objects

Operation --- define behaviors of class of objects

Object --- an instance of a class

Encapsulation --- defining/storing together properties and behavior of a class/object

Information hiding --- shielding implementation detail to reduce change impact to other part of a program

Polymorphism --- one thing can assume more than one form

Represent Domain Model as UML Class Diagram

- UML class diagram is a structural diagram.
- It shows the classes, their attributes and operations, and relationships between the classes.
- Domain model is represented by a class diagram without showing the operations.

UML Class Diagram: Notion and Notation

Class: a type (in OO)

Class Name

Attributes of class

Operations of class

Class Name
Attribute compartment
Operation compartment

Compact view

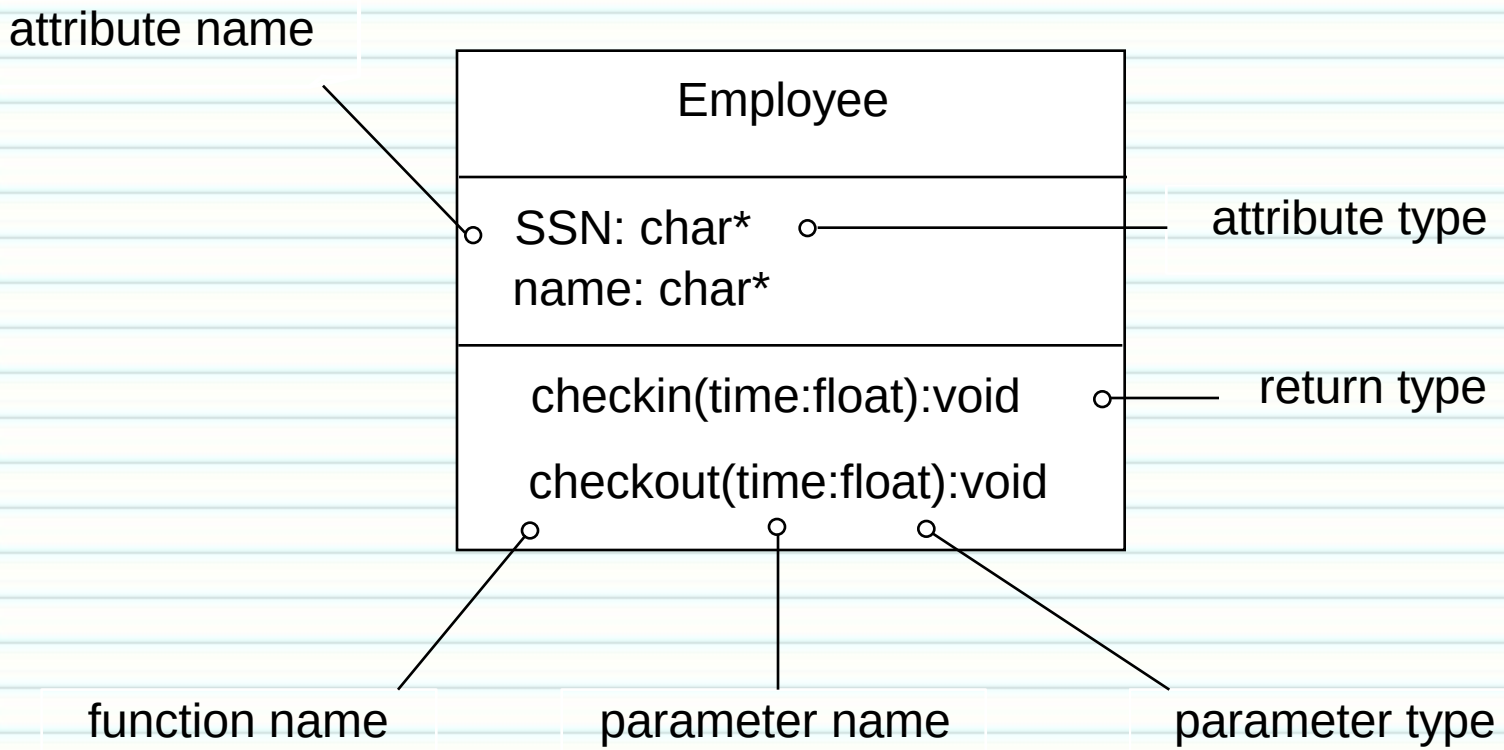
Expanded view

Example:

Employee

Employee
SSN name
checkIn(time) checkOut(time)

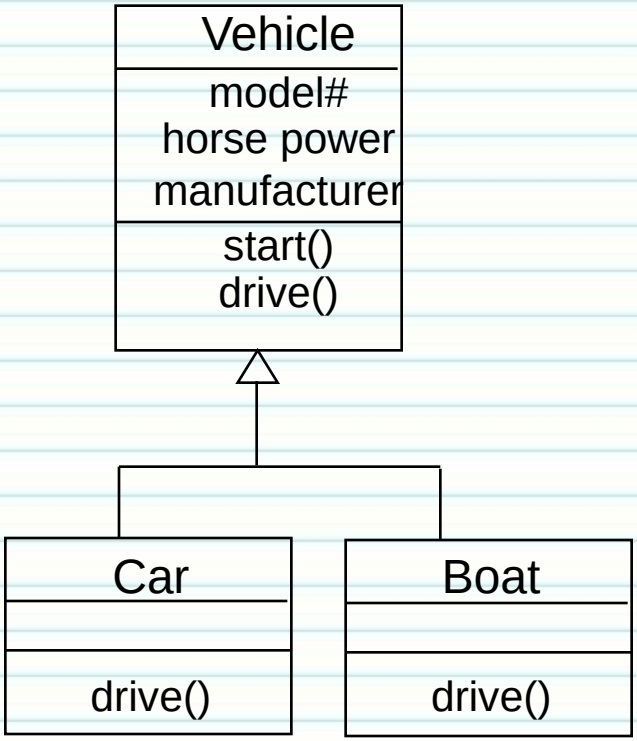
Representing Type in UML



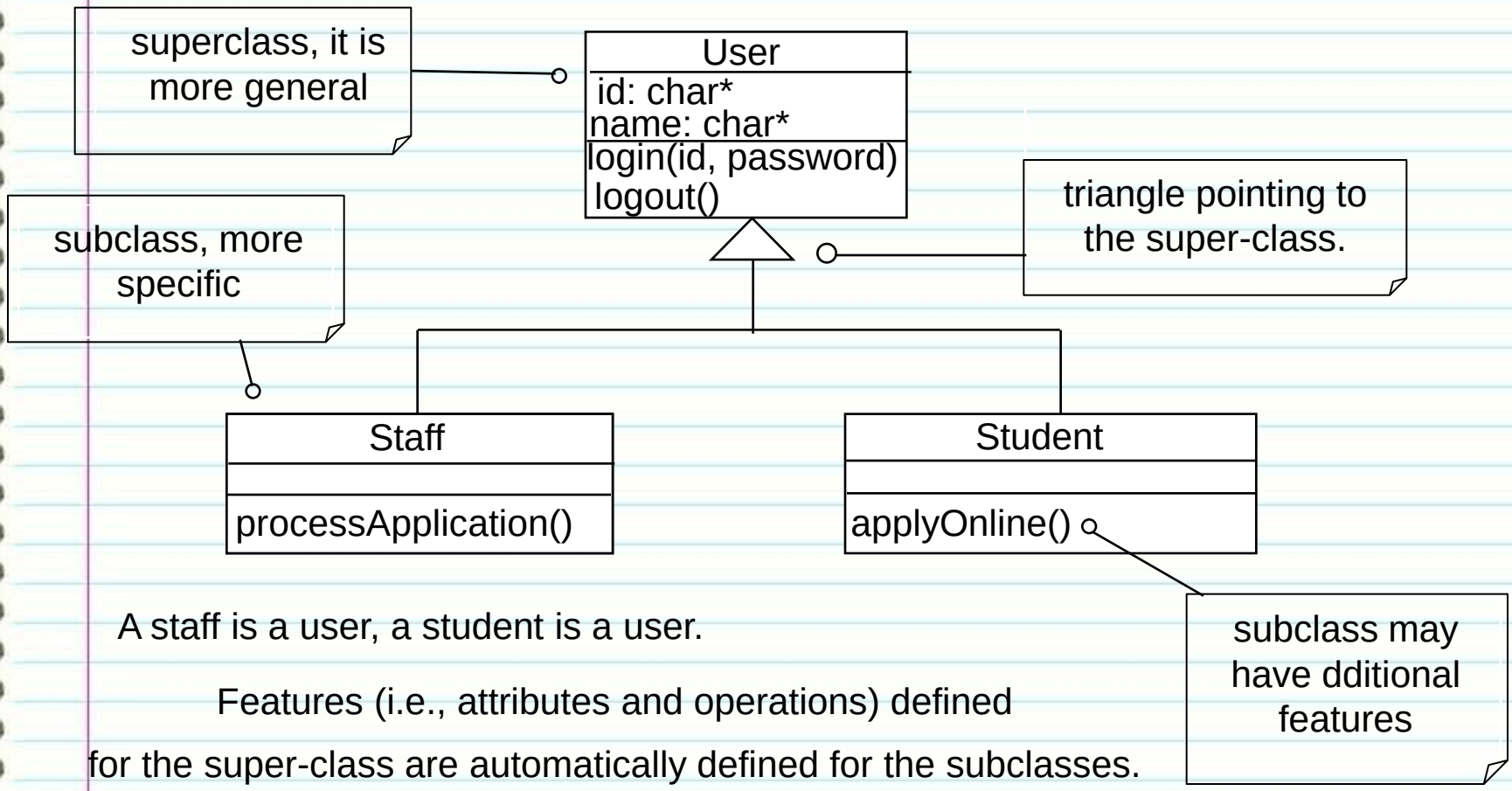
general syntax name **:** type

Inheritance Relationship

- It expresses the generalization / specialization relations between concepts.
- One concept is more general/specialized than the other.
- Example: vehicle is a generalization of car, car is a specialization of vehicle.
- It is also called IS-A relation.



Example: Inheritance



A staff is a user, a student is a user.

Features (i.e., attributes and operations) defined for the super-class are automatically defined for the subclasses.

Object and Attribute

- A noun/noun phrase can be a class or an attribute, how do we distinguish?
- This is often a challenge for beginners.
- Rules to apply:

An object has an *"independent existence*" in the application/application domain, an attribute does not (have).

Example: "Number of seats", class or attribute?

Attribute, because "number of seats" cannot exist without referring to a car, airplane, or classroom as in

"number of seats of a car"

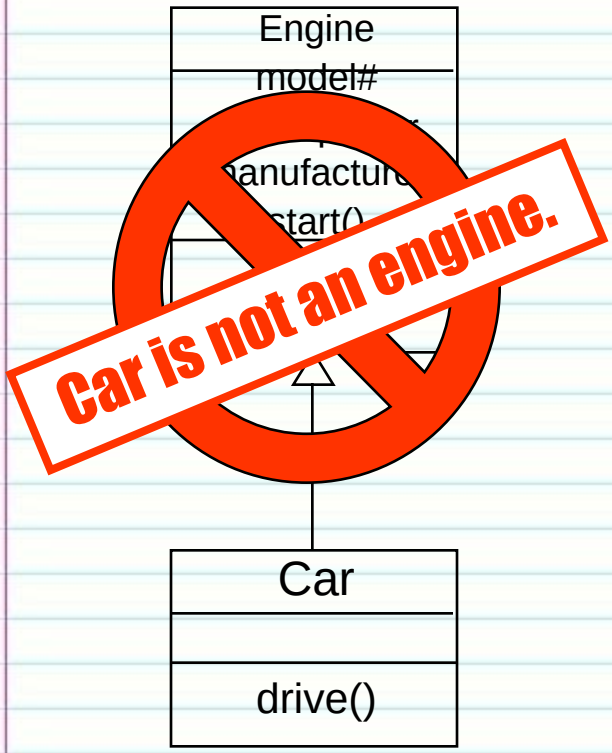
"number of seats of a classroom"

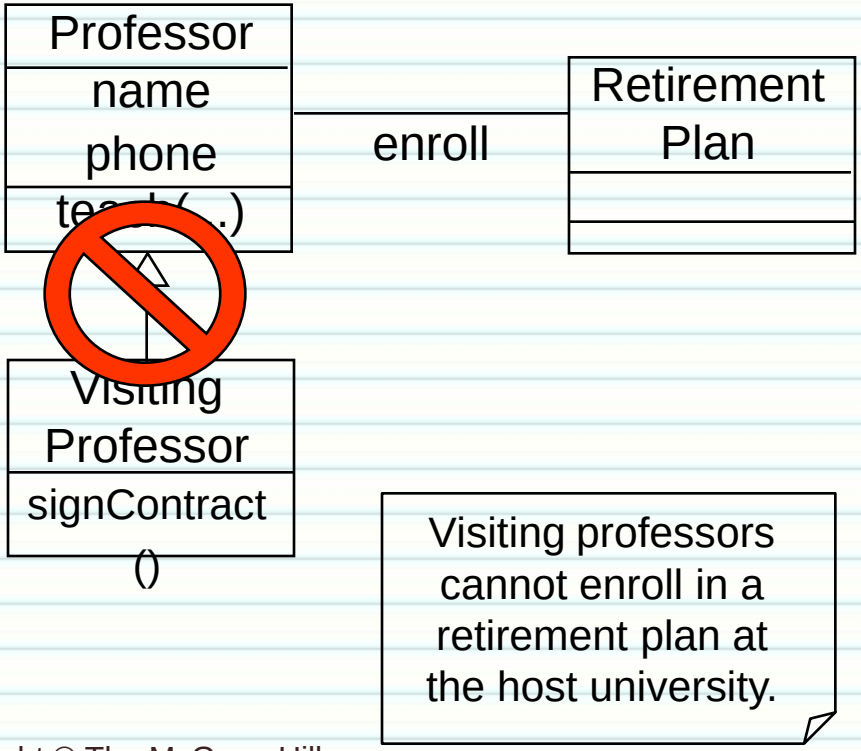
Object and Attribute

- Rules to apply:
  - Attributes describe objects or store state information of objects.
  - You can enter an attribute (value) from the keyboard, but you cannot enter an object.
  - Objects must be created by invoking a constructor (explicitly or implicitly).

Two Tests for Inheritance

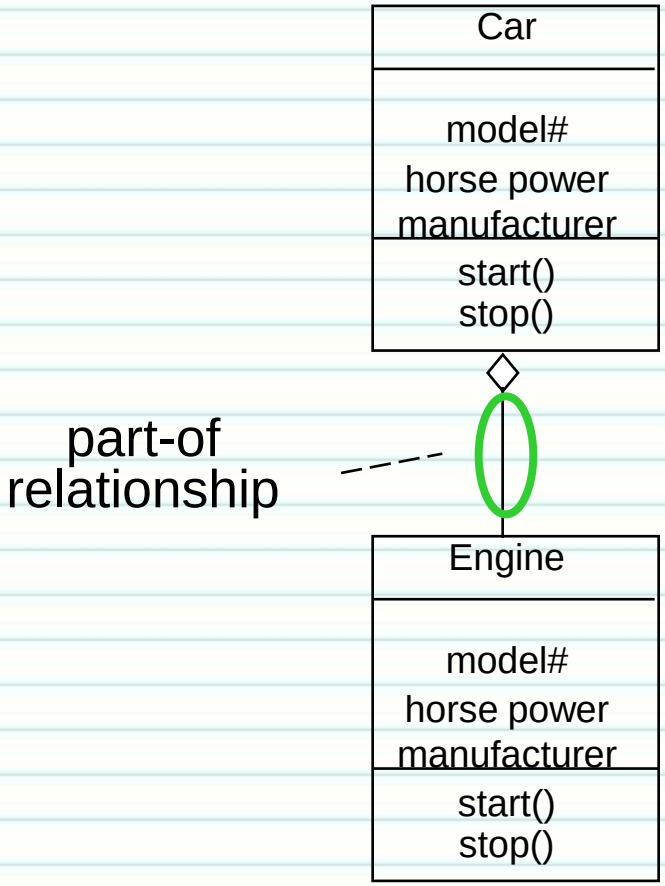
- IS-A test: every instance of a subclass is also an instance of the superclass.
- Conformance test: relationships of a superclass are also relationships of subclasses.





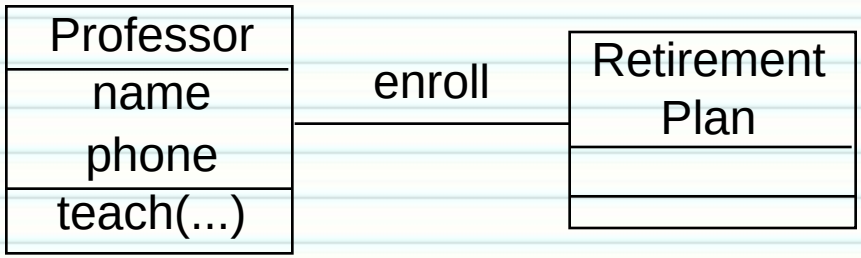
Aggregation Relationship

- It expresses the fact that one object is part of another object.
- Example: engine is part of a car.
- It is also called part-of relationship.



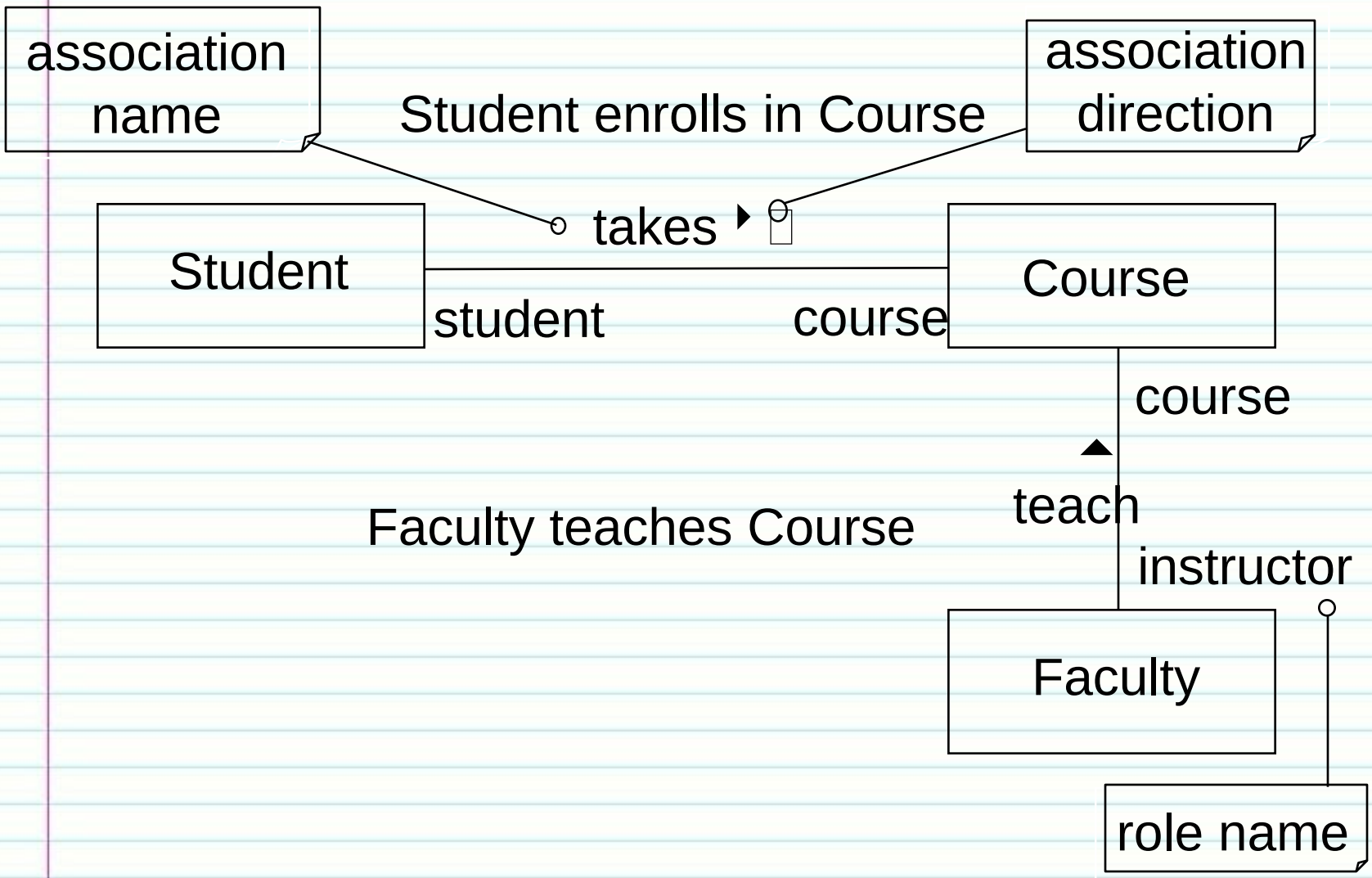
Association Relationship

- It expresses a general relationship other than inheritance and aggregation.
- These can be application specific relationships between two concepts.
- Example: "instructor teach course," "user has account."



Enroll is not an inheritance or aggregation relationship.

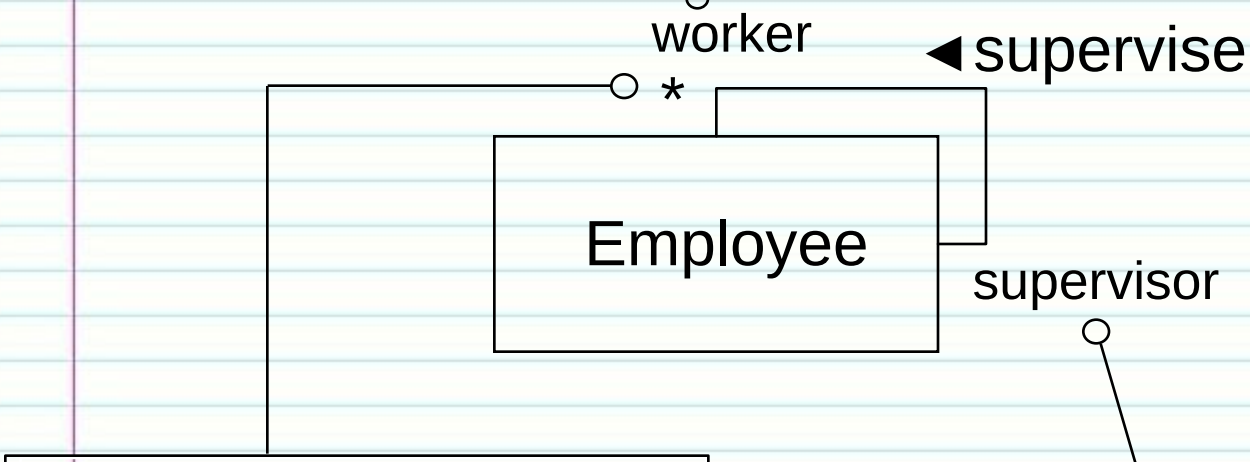
Role and Association Direction



Role and Multiplicity

Another employee is the worker.

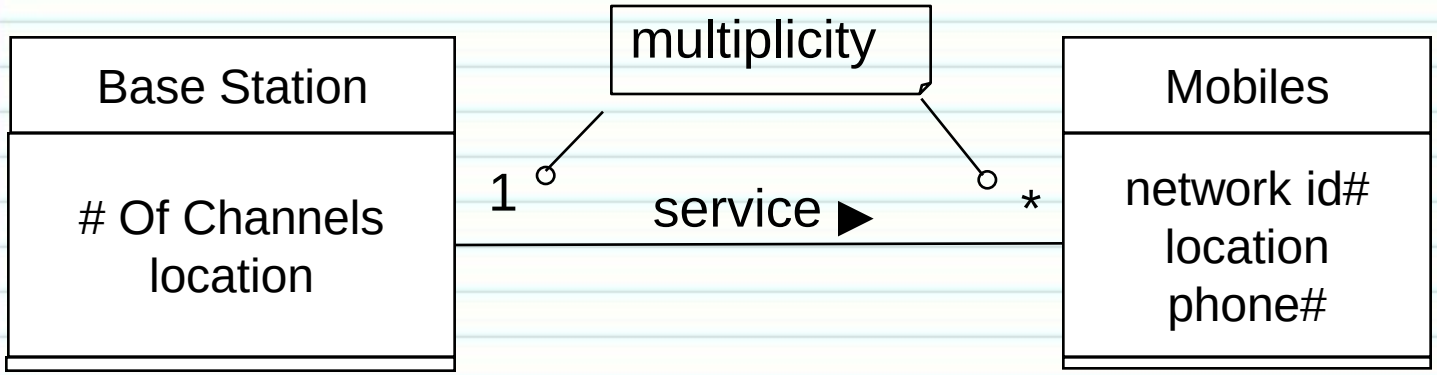
Employee supervises other employees.



A supervisor supervises zero or more employees.

An employee is the supervisor.

Multiplicity Assertion/Constraint



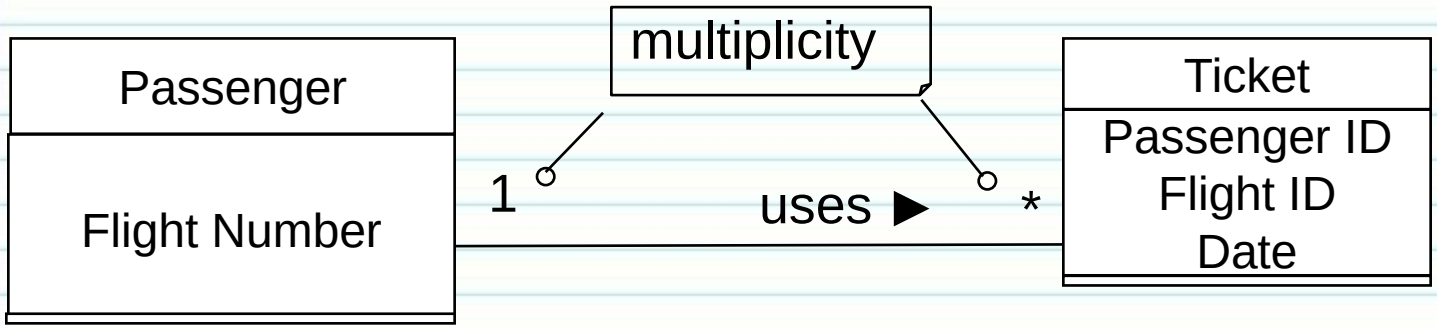
One Base station services zero or more Mobiles and all Mobiles are serviced by exactly one base station.

Other multiplicity constraints:

1	exactly one (default)	1..*	one or more
0..1	zero or one	m..n	m to n
, 0..	zero or more	n	exactly n

All relationships except inheritance must have a multiplicity
- all Aggregation and Association relationships

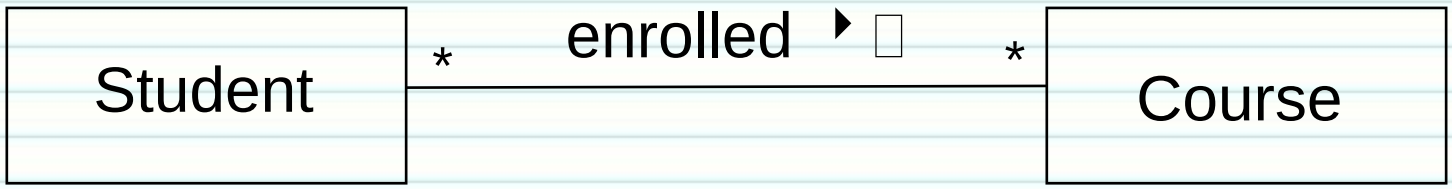
Multiplicity Assertion/Constraint (cont.)



A Passengers uses zero or more Tickets and all Tickets are used by exactly one Passengers . We can't read this as:

1. *:* - All Tickets are used by all Passengers - this means that a ticket may be used by more than one passenger
2. 1:1 - One Ticket is used by exactly one Passenger - a Passenger may use only one Ticket.
3. *:1 - One Ticket is used by all Passengers

Multiplicity Assertion/Constraint (cont.)



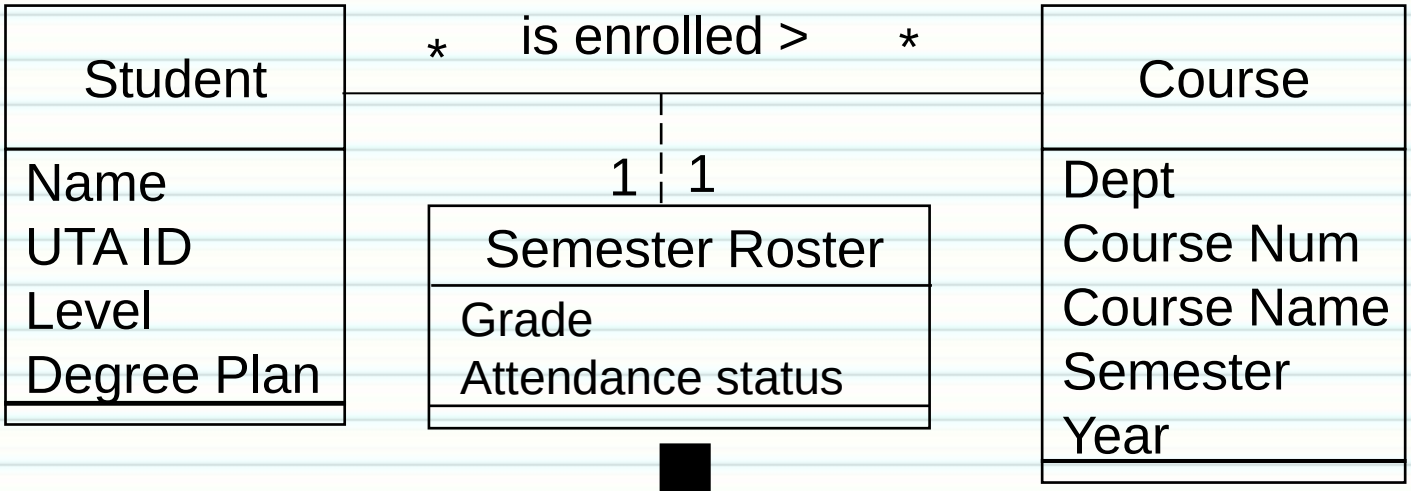
Each Student is enrolled in many Courses and each Course may be enrolled by many Students. We can't read this as:

1. 1:* - Each Student is enrolled in 0 or more Courses, each courses is enrolled by exactly 1 Student
2. 1:1 - Each Student is enrolled in exactly one Course, each Course has exactly one Student
3. *:1 - Each Course is enrolled by 0 or more Students, each Student is enrolled in exactly 1 Course

A *:* relationship MUST have an Association Class to break this up.

Association Class - Where Does Grade Go?

- Putting the grade in the Student class gives a student the same grade for all courses.
- Putting the grade in the Course class gives all students taking the same course the same grade.



Java List

Name	UTA ID	Level	Degree Plan	Dept	Course Num	Course Name	Semester	Year	Grade	Attendance status

Summary: Relationships in UML Class Diagram

Notion	Notation		Example
Inheritance (between classes) IS-A relationship		pointing from subclass to super-class	Employee Manager
Aggregation (between classes) part-of relationship	Uses	File still exists even when the Text Editor is terminated	Text Editor File
Composition (between classes) aggregate exclusively owns the part	Owns	Buffer has no meaning without the Text Editor	Text Editor Buffer
Association (between classes)	service		Student * enroll * Course

Aggregation Vs Composition

- Aggregation is used to represent ownership or a whole/part relationship, and composition is used to represent an even stronger form of ownership
- The composite object has sole responsibility for the disposition of its parts in terms of creation and destruction.
- A composite object can be designed with the knowledge that no other object will destroy its parts.
- There are many, many opinions on the internet about the difference - typically they tend to lean toward physical objects having aggregation relationships and their software counterparts having a composition relationship - primarily because of object creation and destruction.
- The best tip is not to spend too much time trying to distinguish - there is no practical reason for the distinction

Applying Agile Principles

1. *Work closely with the customer and users to understand their application and application domain.*
2. *Perform domain modeling only if it is needed. Keep it simple and expand it incrementally.*
3. *Domain modeling may be performed simultaneously with actor-system interaction modeling, object interaction modeling, object state modeling, and activity modeling.*

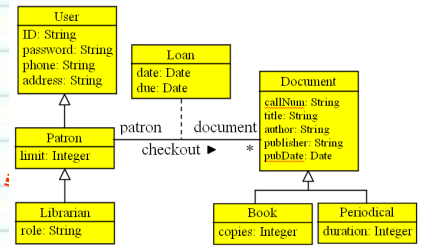
Domain Modeling Steps

1.Collecting application domain information

2. Brainstorming

3. Classifying brainstorming result

4. Visualizing the domain model



5. Reviewing the domain model.

Steps for Domain Modeling

- 1) Collecting application domain information
  - focus on the functional requirements
  - also consider other requirements and documents
  - also consider business descriptions
- 2) Brainstorming
  - list important application domain concepts
  - list their properties/attributes
  - list their relationships

3) Classifying the domain concepts into:

- classes
- attributes / attribute values
- relationships
  - association, inheritance, aggregation

4) Visualizing the result using a UML class diagram

Brainstorming: Rules to Apply

- The team members get together to identify & list domain specific
  1. nouns / noun phrases
  2. "X of Y" expressions (e.g., color of car)
  3. transitive verbs
  4. adjectives
  5. numeric
  6. possession expressions (has/have, possess, etc.)
  7. "constituents / part of" expressions
  8. containment / containing expressions
  9. "X is a Y" expressions

Classifying Brainstorming Result

1. nouns/noun phrases ⇒ class or attributes
2. "X of Y" expressions ⇒ X is an attribute of Y
   ⇒ X is part of Y
   ⇒ X is a role in an association
3. transitive verbs ⇒ association relationships
4. adjectives ⇒ attribute values
5. numeric ⇒ attribute / multiplicity values
6. possession expressions ⇒ aggregation or attribute
   (has/have, possess, etc.)
7. "consist of/part of" expression ⇒ aggregation relationships
8. containment / containing expressions ⇒ association or aggregation
9. "X is a Y" expressions ⇒ inheritance

Objects have independent existence, attributes do not.

Example

noun/noun phrases

numeric

A car has make, model, horse power, number of seats ...

possession expression

(c) car
(a) make
(a) model
(a) horse power
(a) number of seats

Car has independent existence. Make, model, horse power, and number of seats do not.

A customer can rent one or more cars ...

(c) customer
(c) car

rent

1+

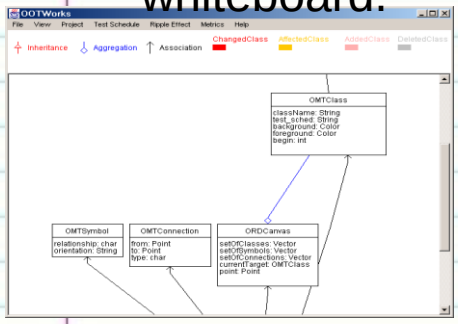
Tip for Domain Modeling

Do not do brainstorming and drawing at the same time. The result could be very poor.

1) Team brainstorming: List the concepts, and then classify them on a whiteboard.

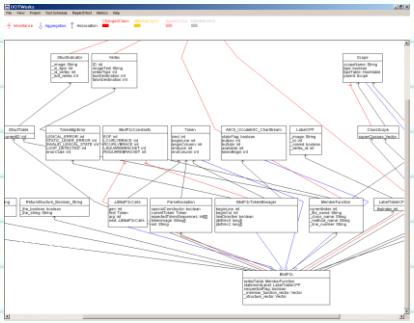
2) Take a picture(s) of the whiteboard using a digital camera.

3) Email the digital images to team members.



4) Have a member or two to convert the result to a UML class diagram.

5) Email the UML class diagram to all members to review.



6) Modify the diagram to reflect corrections and comments.

Class Exercise

- Do the following for your team project or the vending machine (next slide).
- Identify the concepts that exist in the application domain.
- Classify the concepts in terms of
  - classes
  - attributes of classes
  - relationships between the classes
    - inheritance
    - aggregation and
    - association

Class Exercise: The Vending Machine

The Vending Machine has a display, an alphanumeric keypad, a coin insertion slot, and an item dispenser.

The display shows the vending items like chocolates, candies, potato chips, Coke, sprite, etc. Each type of item has a price and a label consisting of a letter A, B, C, ... and a digit 1, 2, ... A customer inserts coins through the coin slot.

Each time a coin is inserted an LCD displays the total amount.

The customer can press a letter and a digit to enter his selection after enough coins have been inserted. If the total amount is greater than or equals to the item selected, the vending machine dispenses the item and returns the change to the customer.

A customer can change his mind and request that the coins be returned by pressing the return button.

Applying Agile Principles

1. *Work closely with the customer and users to understand their application and application domain.*
2. *Perform domain modeling only if it is needed. Keep it simple and expand it incrementally.*
3. *Domain modeling may be performed simultaneously with actor-system interaction modeling, object interaction modeling, object state modeling, and activity modeling.*

Domain Diagrams Related to the Class Project

Domain Modeling - Putting it all together

- Start with the UCID each unique input and/or output table should be attributes within a class
- Make sure to capture each system user from the UCD in the DD
- Use inheritance where needed to identify common and unique attributes across different system users

Domain Diagram Common Problems

- Not showing all nouns and attributes from the requirements or descriptions
- Wrong associations - aggregation instead of inheritance or vice-versa
- Classes -
  1. start with the domain as a class - e.g., UTA housing system has-a ... use aggregation to capture this
  2. Not showing actor hierarchies as inheritance hierarchies
- Multiplicities
  1. Not showing any or all
  2. Not eliminating many-to-many relationships using an associative class
- Association relationships
  1. Not showing any or all
  2. Not labeling them with a verb

Domain Diagram Common Problems (cont.)

- Make sure to update requirements with any discoveries about
  1. Nouns, verbs, attributes
  2. Multiplicities (esp. when not 0 or * - such as restrictions on the number)
- Use forms or lists in your data especially where you have many-to-many relationships
  - Capture these as association classes