KASETSART UNIVERSITY
FACULTY OF SCIENCE AT SRIRACHA

UML (Unified Modeling Language)
Class Diagram

01418321 System Analysis and Design

Chalothon Chootong (Ph.D.)

Department of Computer Science and Information, Faculty of Science at Sriracha, Kasetsart University Sriracha Campus

chootong.c@ku.th

ไดอะแกรมต่างๆ

- ยูสเคสไดอะแกรม (Use Case Diagram)
- คลาสไดอะแกรม (Class Diagram)
- ออบเจ็คไดอะแกรม (Object Diagram)
- แอ็กทิวิตีไดอะแกรม (Activity Diagram)
- สเตทชาร์ตไดอะแกรม (State chart Diagram)
- คอลแลบอเรชันไดอะแกรม (Collaboration Diagram)
- ซีเควนซ์ไดอะแกรม (Sequence Diagram)
- คอมโพเนนต์ไดอะแกรม (Component Diagram)
- ดีพลอยเมนต์ไดอะแกรม (Deployment Diagram)

What is a UML Class Diagram?

- Class diagrams are the backbone of almost every object-oriented method including UML.
- They describe the static structure of a system.
- Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

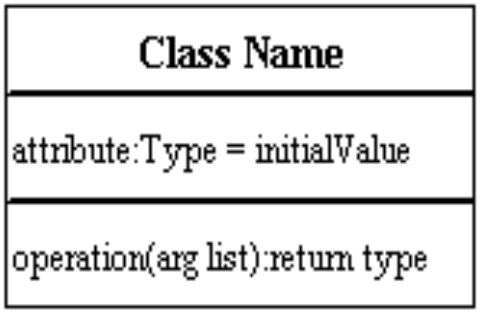
What is a UML Class Diagram?

- A Class Diagram is a diagram describing the structure of a system
- Shows the system's
  - Classes
  - Attributes
  - Operations (or methods)
  - Relationships among the classes

Essential Element of UML Class Diagram

- Class
- Attributes
- Operations
- Relationships
  - Associations
  - Generalization
  - Realization
  - Dependency
- Constraint Rules and Notes

Classes



- Describes a set of objects having similar:

  Attributes (status)

  Operations(behaviour)

  Relationships with other classes

- Attributes and operations may

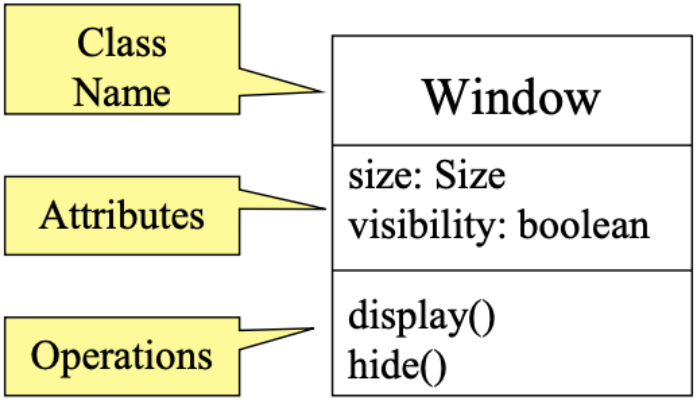
  have their visibility marked:

  "+" for *public*

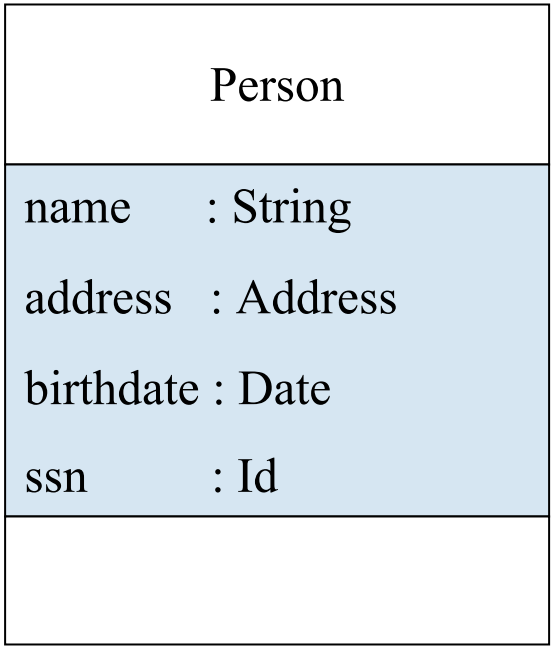
  "#" for *protected*

  "—" for *private*

  "~" for *package*



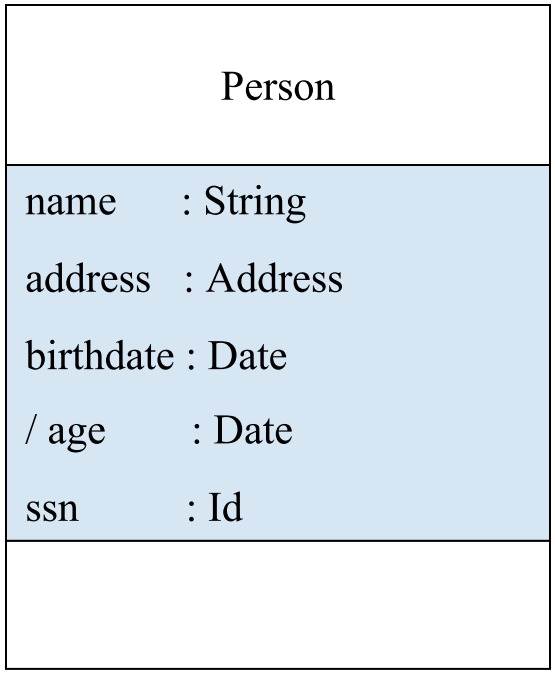
Class Attributes



An *attribute* is a named property of a class that describes the object being modeled.

In the class diagram, attributes appear in the second compartment just below the name-compartment.

Class Attributes



Attributes are usually listed in the form:

attributeName : Type

A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For example,

a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

/ age : Date

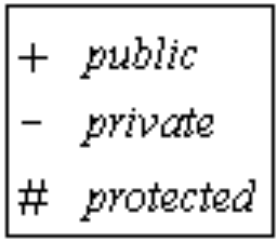
Visibility

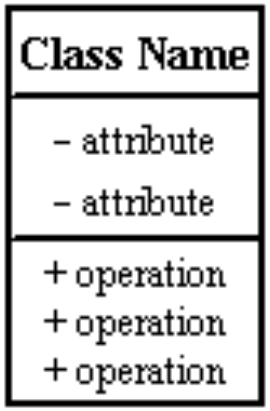
- Use visibility markers to signify who can access the information contained within a class.
- **Private** visibility hides information from anything outside the class partition.
- **Public** visibility allows all other classes to view the marked information.
- **Protected** visibility allows child classes to access information they inherited from a parent class.

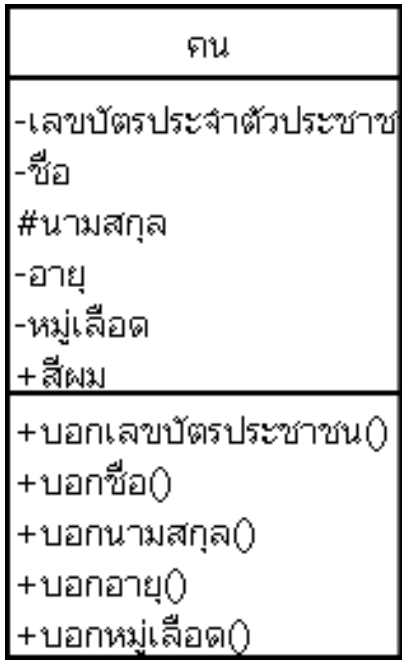
Visibility

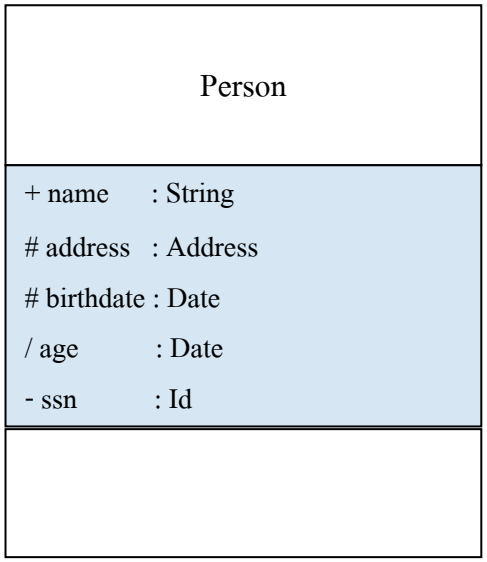
- Public สามารถมองเห็นได้จากภายนอก สามารถเข้าไปเปลี่ยนค่า อ่านค่า หรือเรียกใช้งาน Attributes หรือ Functions นั้นได้ทันทีโดยอิสระจากภายนอก (มักใช้กับ method มากกว่า Attributes)
- Private ไม่สามารถมองเห็นได้จากภายนอกของ Class แต่สามารถ แต่สามารถมองเห็นได้จากภายในตัว Class เองเท่านั้น หากภายนอกต้องการที่จะเข้ามาเพื่อแก้ไขหรืออ่านค่าหรือเรียกใช้ Attributes และ Function ต้องมีการเรียกผ่านทางฟังก์ชัน (มักใช้กับ Attributes มากกว่า Method)
- Protected สงวนไว้สำหรับการทำ Inheritance

Visibility









Attributes can be:

+ public

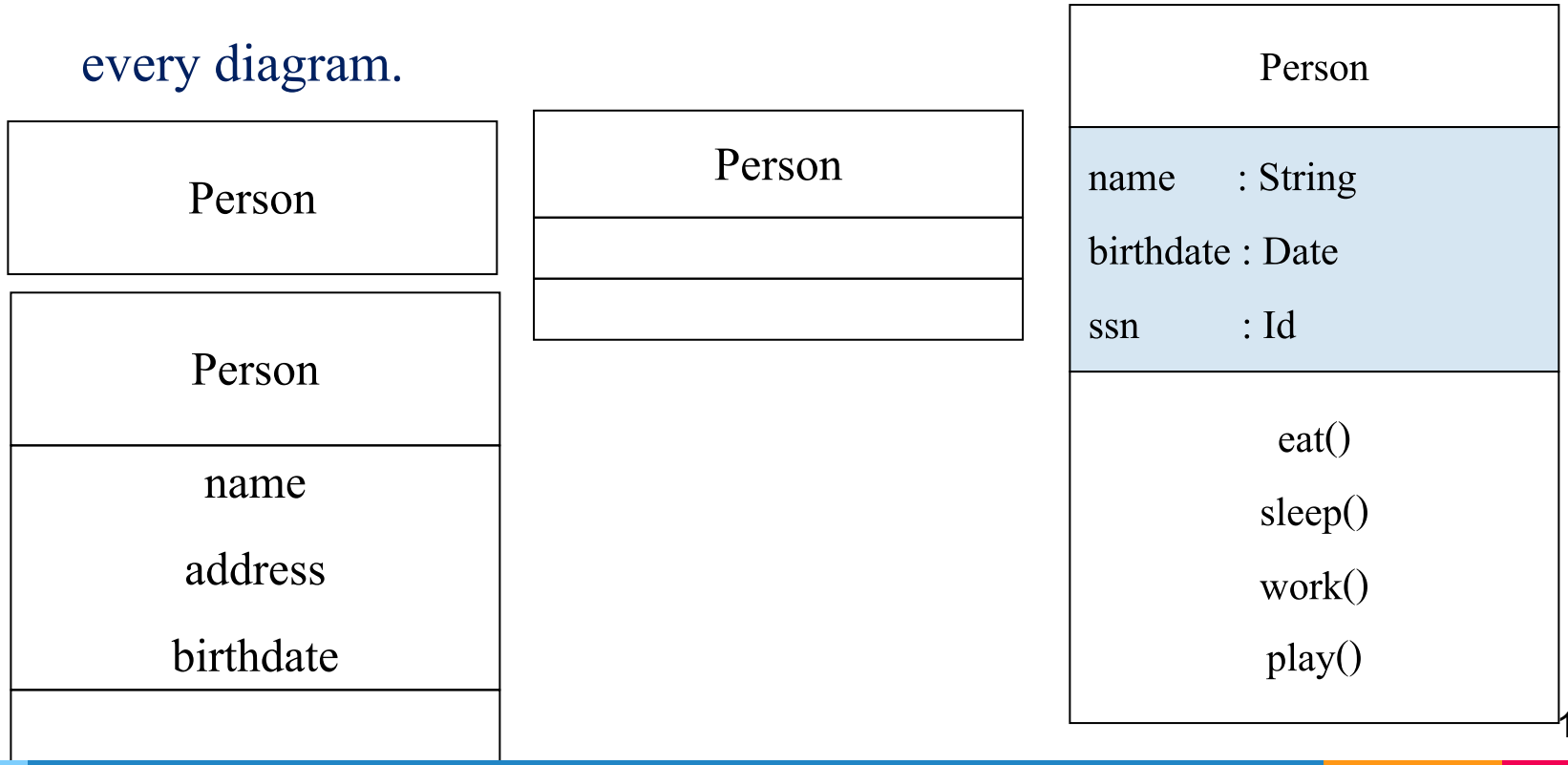
protected

- private

/ derived

Class Operations

- Operations describe the class behavior and appear in the third compartment
- When drawing a class, you needn't show attributes and operation in every diagram.

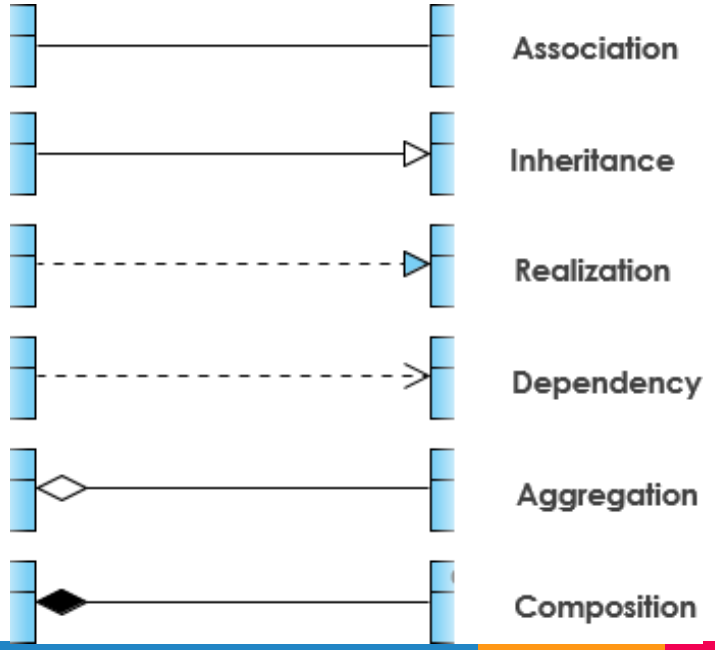


Relationships

In UML, object interconnections (logical or physical), are modeled as relationships.

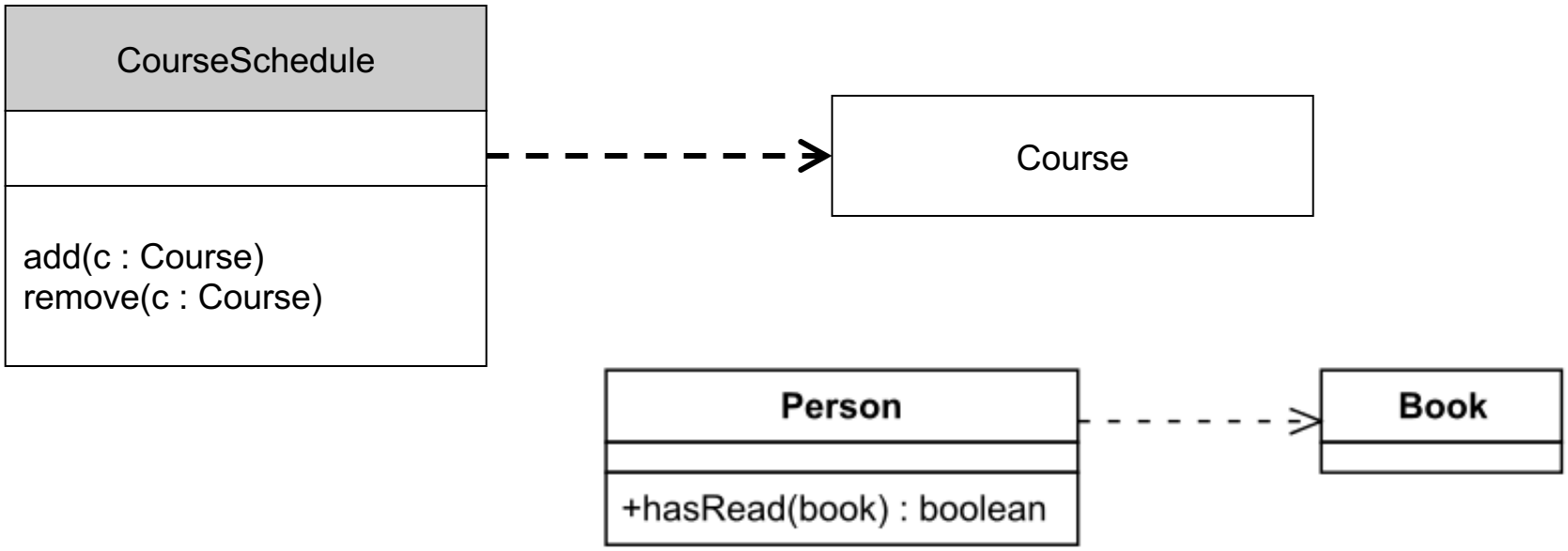
There are three kinds of relationships in UML:

- dependencies
- generalizations
- associations

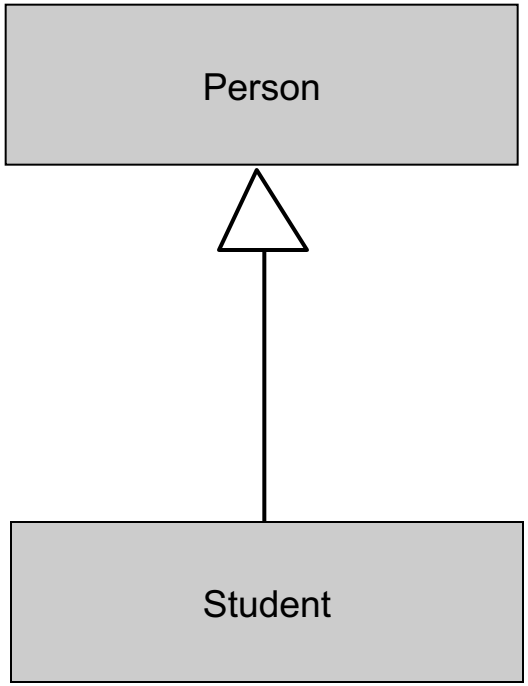


Dependency Relationships

- A *dependency* indicates a semantic relationship between two or more elements.
- The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.



Generalization Relationships

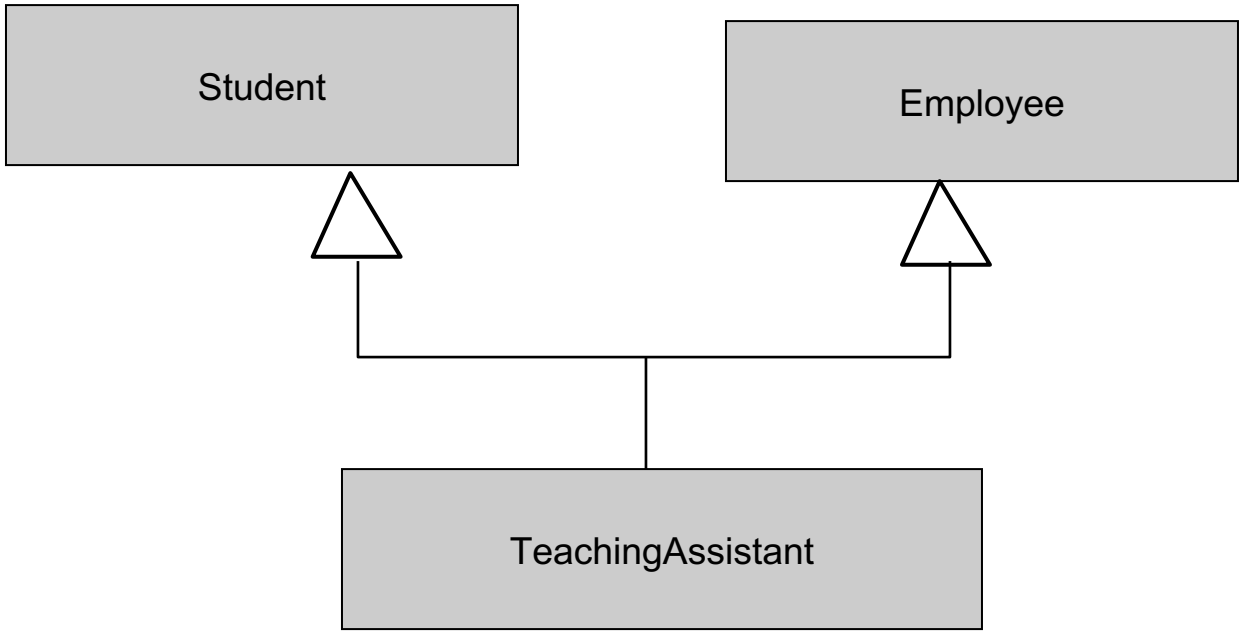


- A *generalization* connects a subclass to its superclass.
- It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

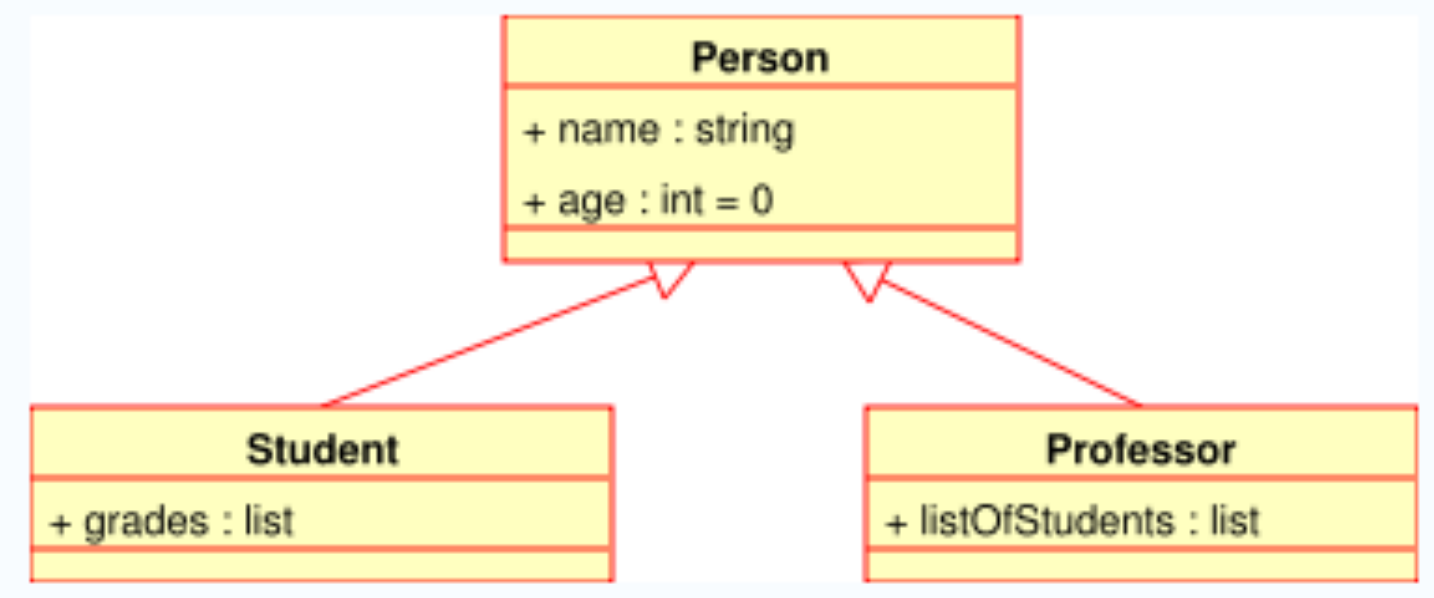
การรับข้อมูลลักษณะบางประการจาก Class แม่

Generalization Relationships

- UML permits a class to inherit from multiple superclasses, although some programming languages (*e.g.,* Java) do not permit multiple inheritance.



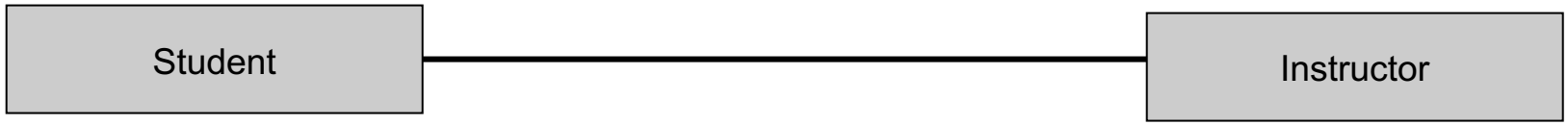
Generalization Relationships



Association Relationships

- If two classes in a model need to communicate with each other, there must be link between them.

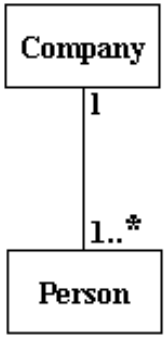
- An *association* denotes that link.



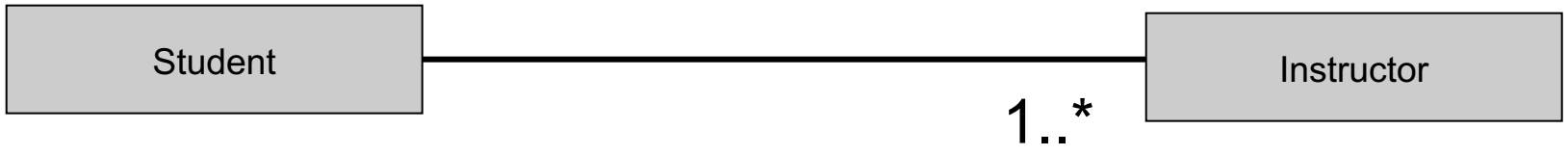
เป็นความสัมพันธ์แบบอ้างอิง โดยไม่มีใครเป็นเจ้าของใคร

Association Relationships

- One to One (1..1) มีได้แค่ 1 ไม่มีไม่ได้
- Zero to One (0..1) มีได้แค่ 1 ไม่มีเลยก็ได้
- One to Many (1..*) มีได้มากกว่า 1 ไม่มีไม่ได้
- Zero to Many (0..*) มีได้มากกว่า 1 ไม่มีเลยก็ได้

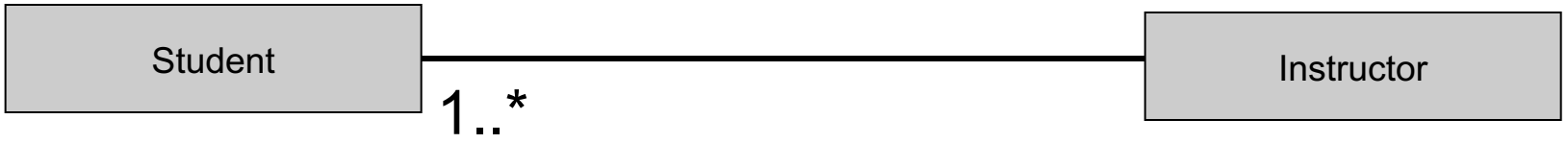


- We can indicate the *multiplicity* (หลากหลาย) of an association by adding *multiplicity adornments* to the line denoting the association.
- The example indicates that a *Student* has one or more *Instructors*:

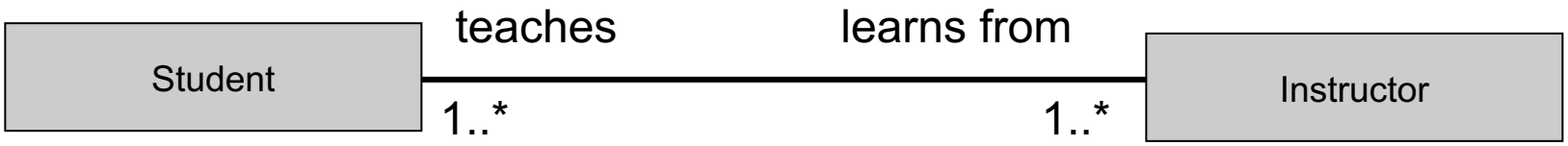


Association Relationships

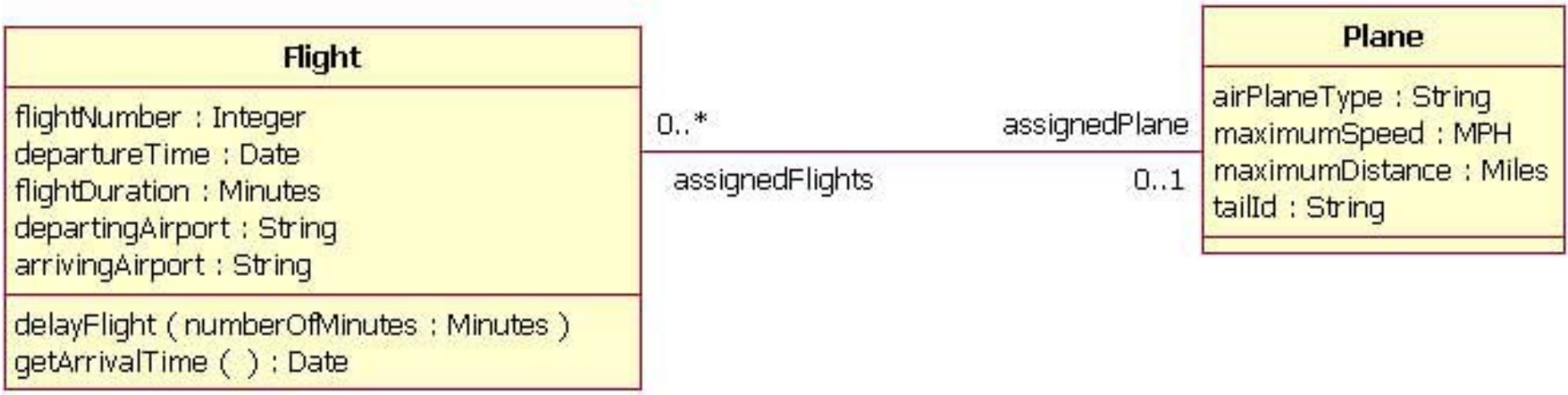
- The example indicates that every *Instructor* has one or more *Students*:



We can also indicate the behavior of an object in an association (*i.e.,* the *role* of an object) using *rolenames*.



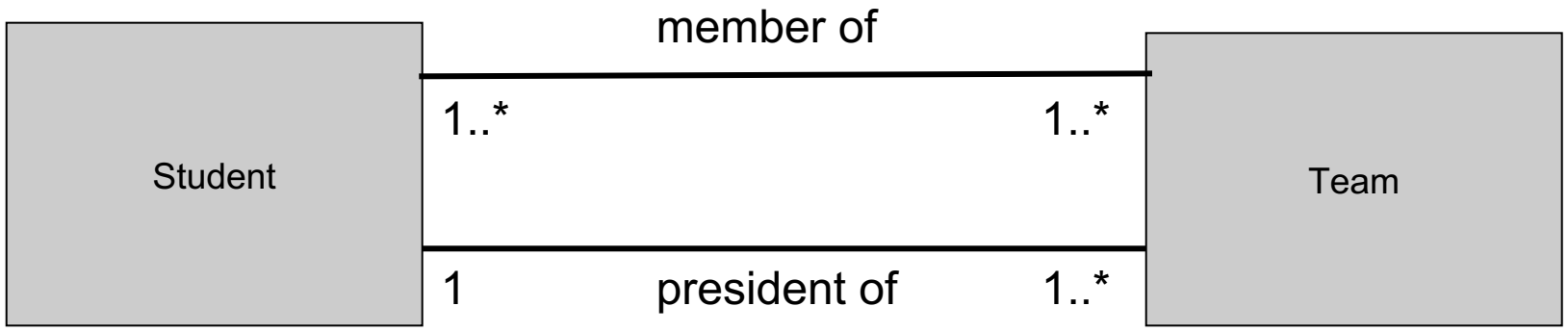
Bi-directional (standard) Association

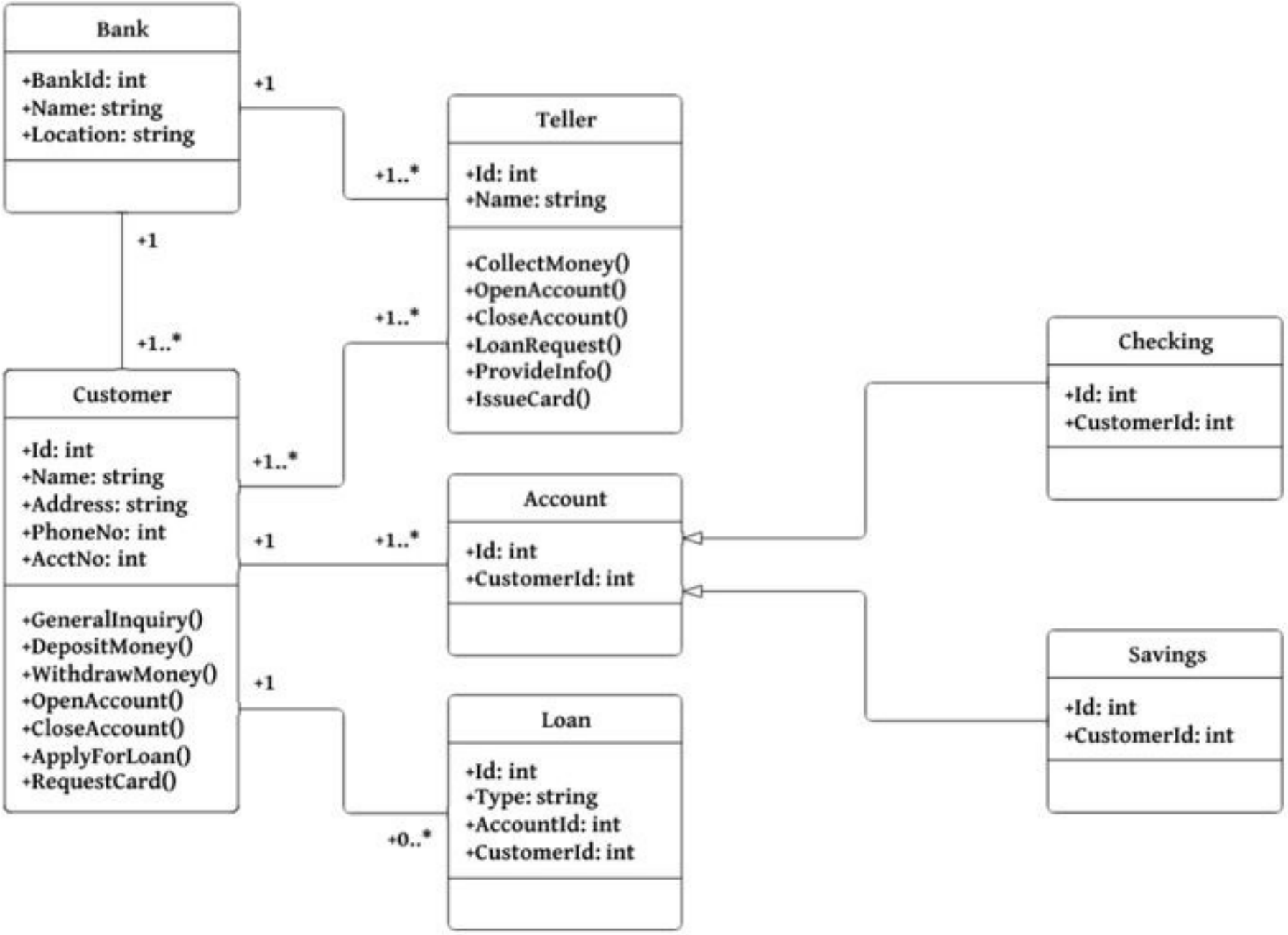


Association Relationships (**Multiplicity**)

We can also name the association.

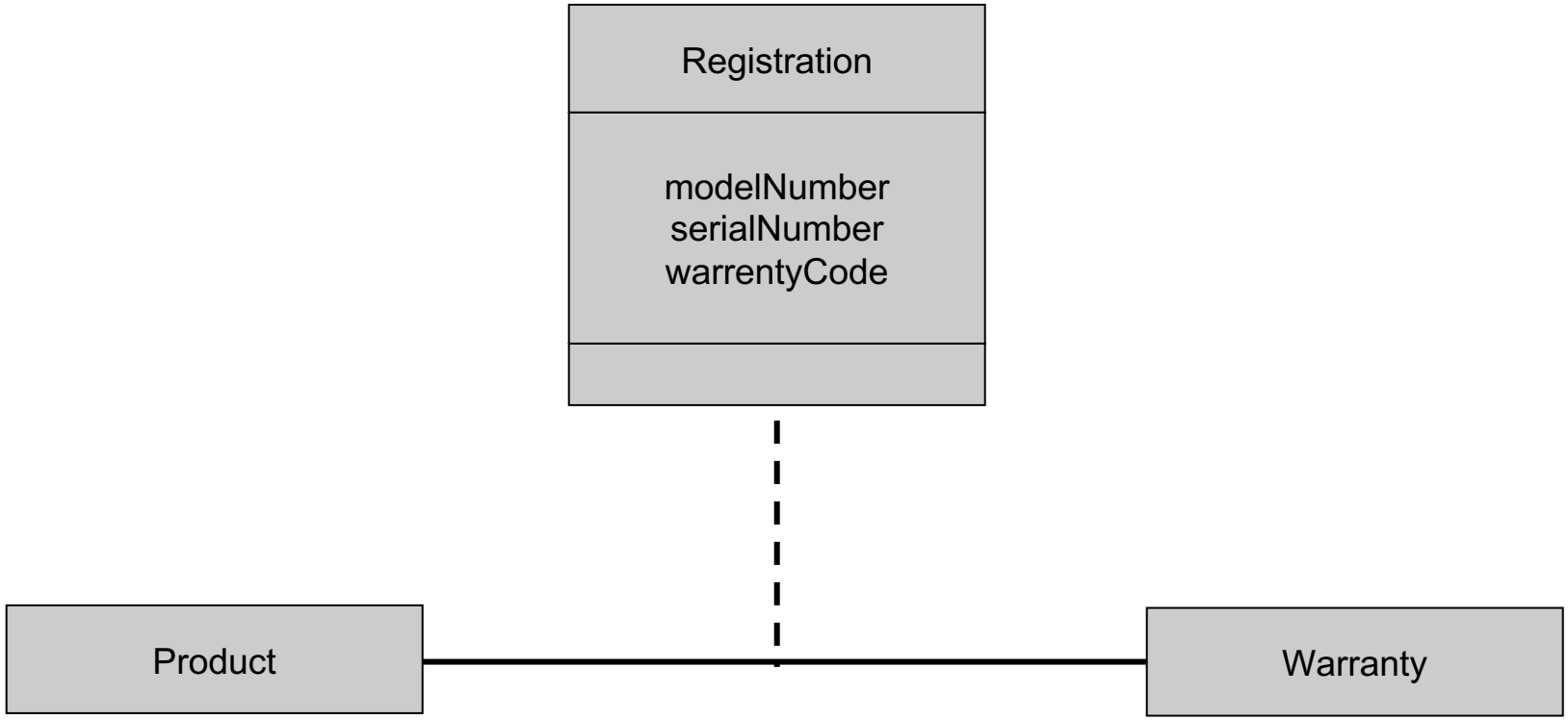
We can specify dual associations.




Bank
+BankId: int
+Name: string
+Location: string
+1
+1..*
Teller
+Id: int
+Name: string
+CollectMoney()
+OpenAccount()
+CloseAccount()
+LoanRequest()
+ProvideInfo()
+IssueCard()
+1
+1..*
+1..*
Customer
+Id: int
+Name: string
+Address: string
+PhoneNo: int
+AcctNo: int
+GeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+ApplyForLoan()
+RequestCard()
+1..*
+1
+1..*
Account
+Id: int
+CustomerId: int
+1
+0..*
Loan
+Id: int
+Type: string
+AccountId: int
+CustomerId: int
Checking
+Id: int
+CustomerId: int
Savings
+Id: int
+CustomerId: int

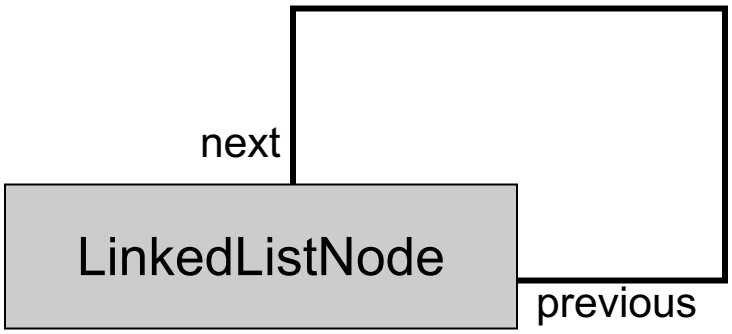
Association Relationships

- Associations can also be objects themselves, called *link classes* or an *association classes*.



Association Relationships

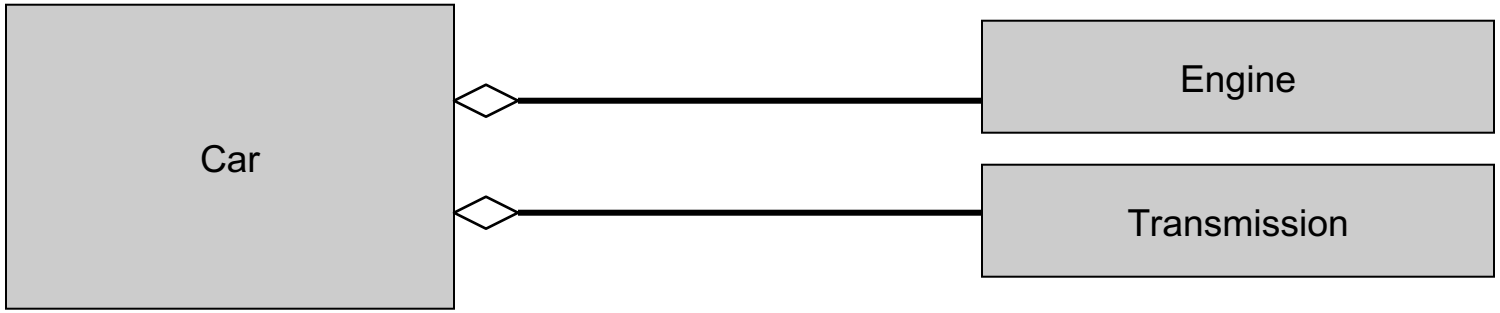
A class can have a *self association*.



Association Relationships

- We can model objects that contain other objects by way of special associations called *aggregations* and *compositions*.

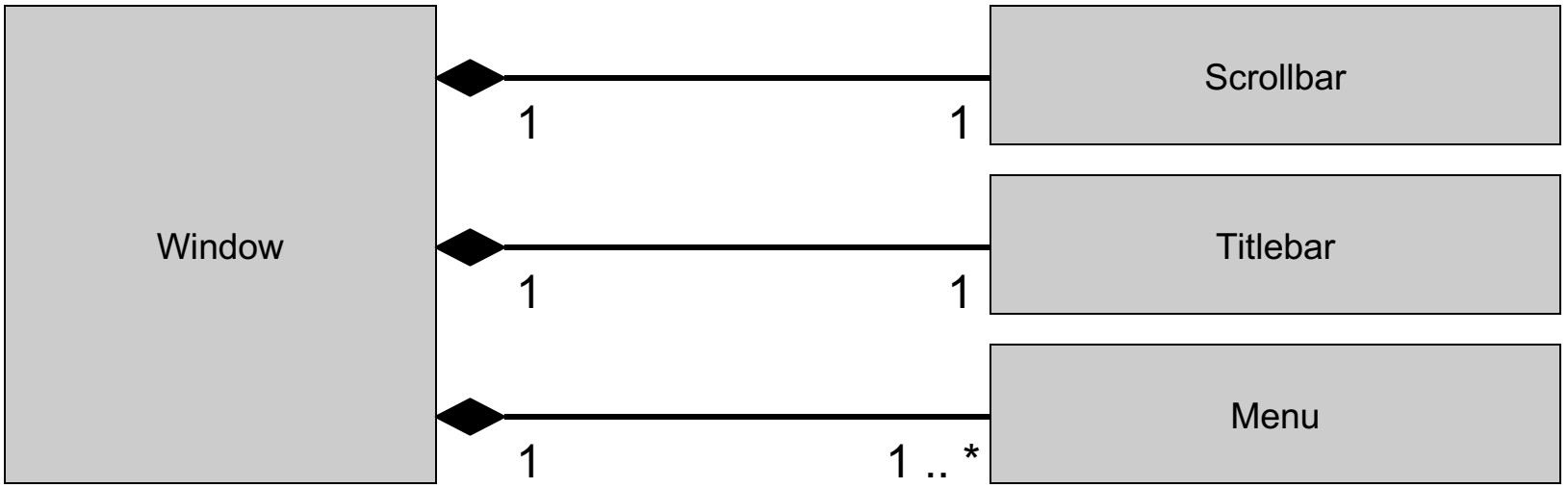
- An *aggregation* specifies a whole-part relationship between an aggregate (a whole) and a constituent part, where the part can exist independently from the aggregate. Aggregations are denoted by a hollow-diamond adornment on the association.



เป็นความสัมพันธ์แบบต้องขึ้นอยู่กับอีกฝ่ายหนึ่ง เช่น สาขาวิชา CS มีอาจารย์หลายท่านเป็นอาจารย์ประจำหลักสูตร

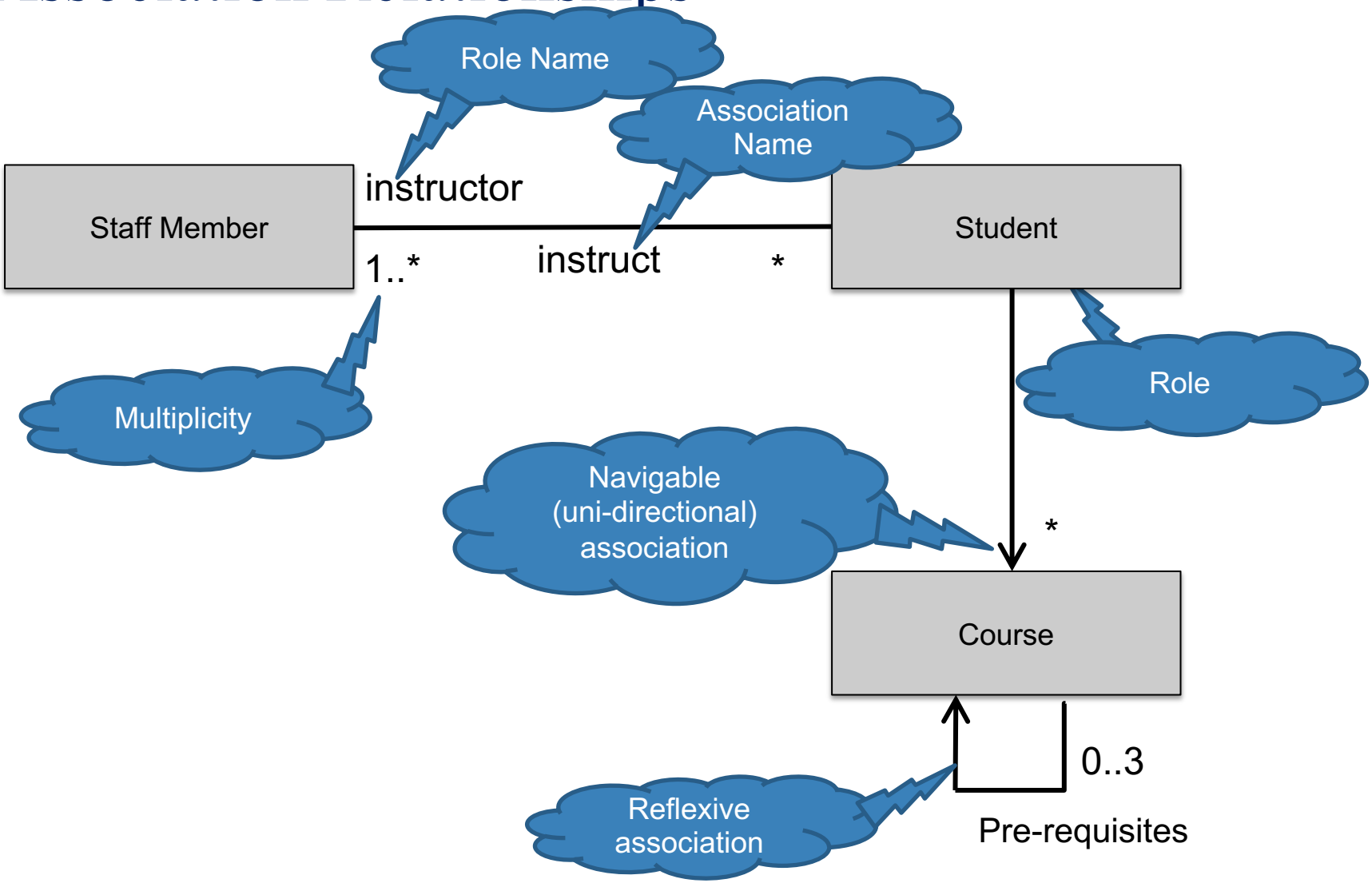
Association Relationships

A *composition* indicates a strong ownership and coincident lifetime of parts by the whole (*i.e.*, they live and die as a whole). Compositions are denoted by a filled-diamond adornment on the association.

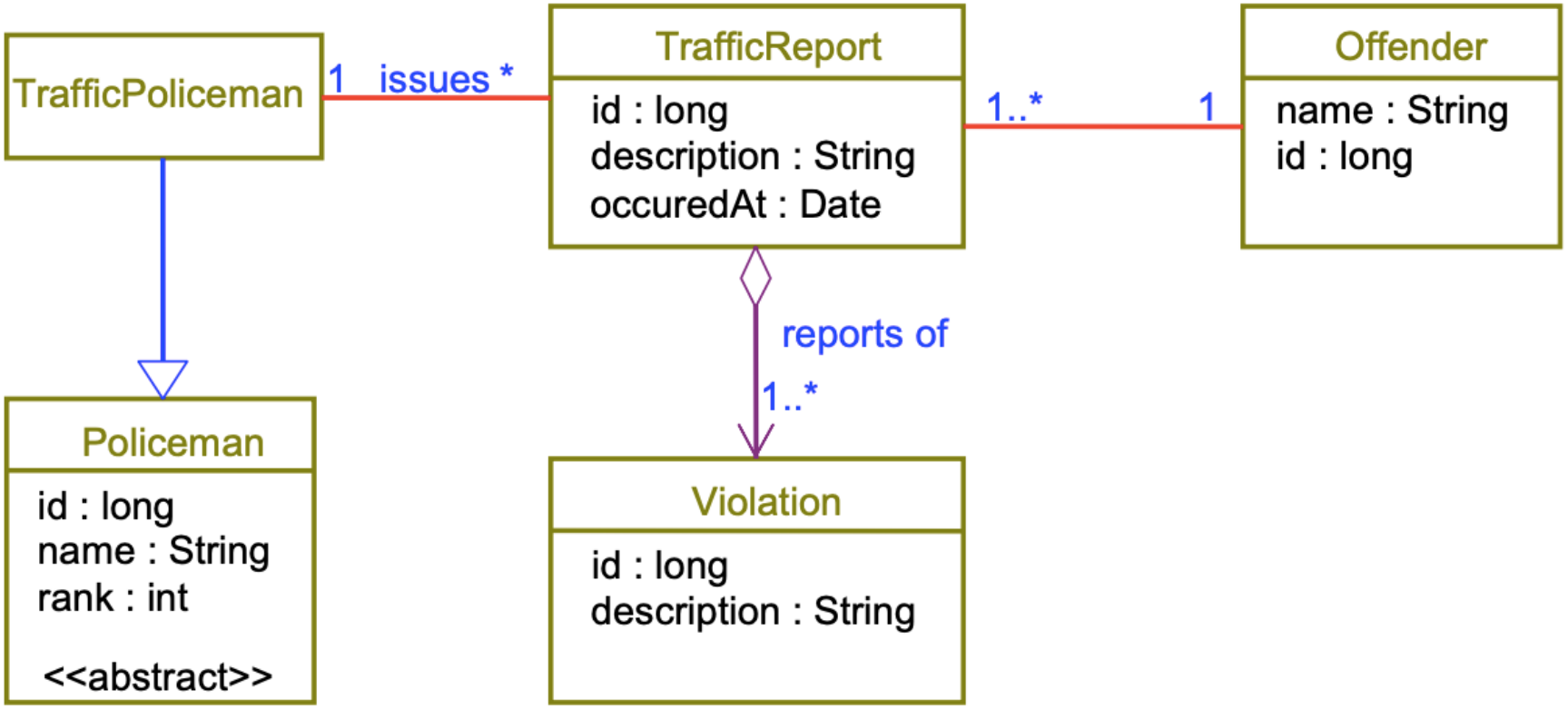


เป็นความสัมพันธ์คล้ายกับ Aggregation แต่จะต่างกันตรง class สมาชิกจำดำรงอยู่ได้เมื่อมี class หลักเท่านั้น

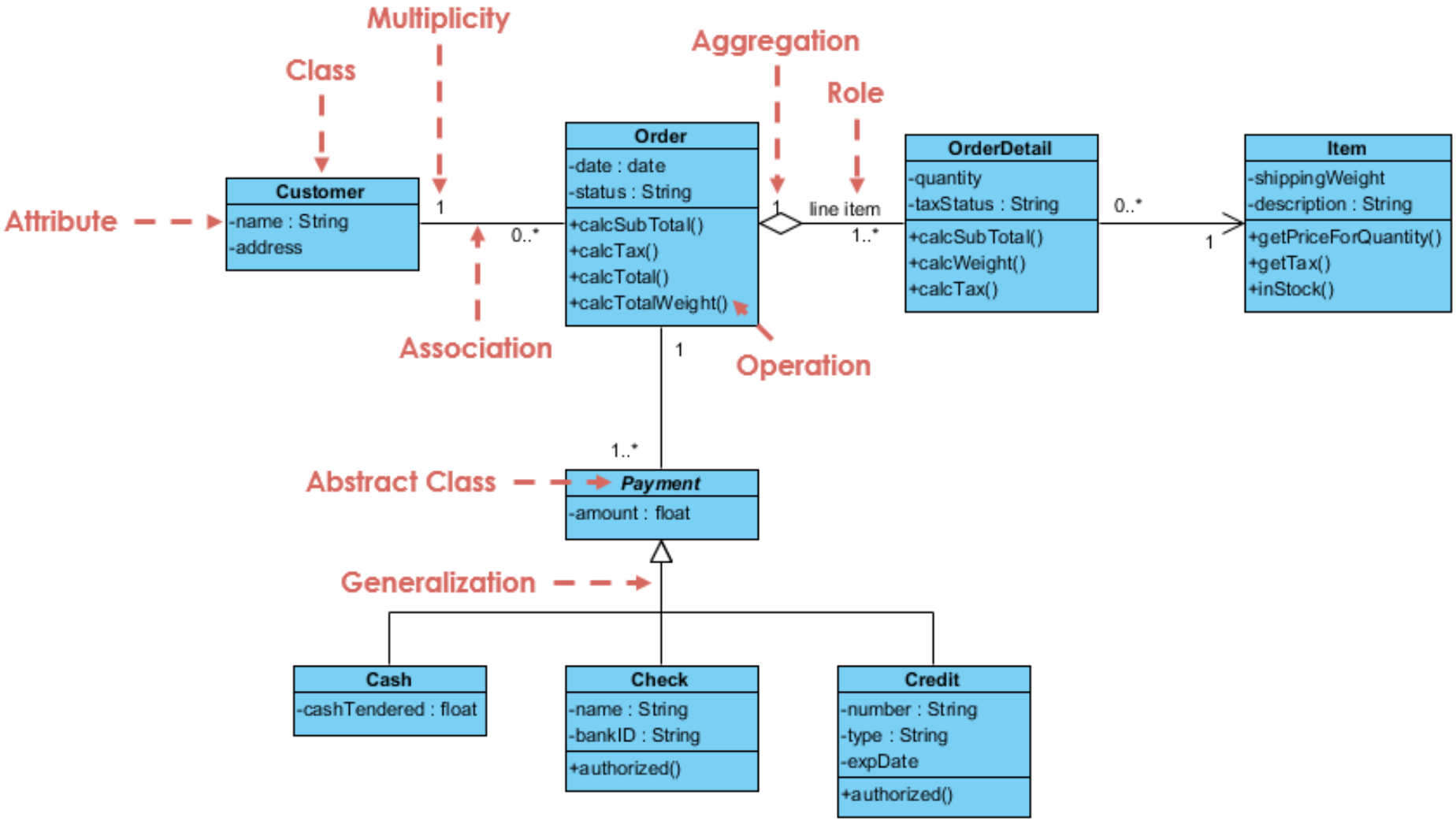
Association Relationships



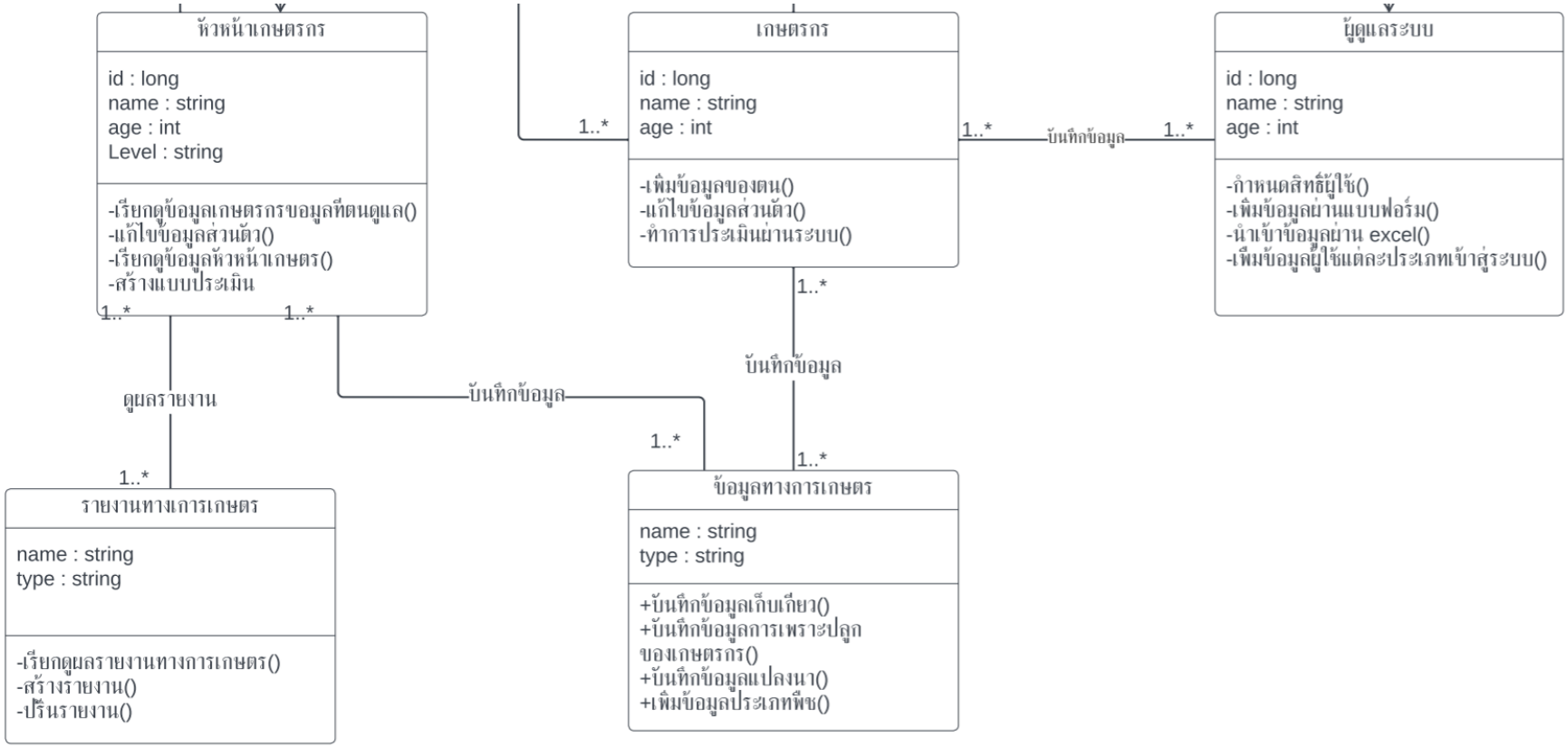
Traffic Violation Report System Example



Class Diagram Example

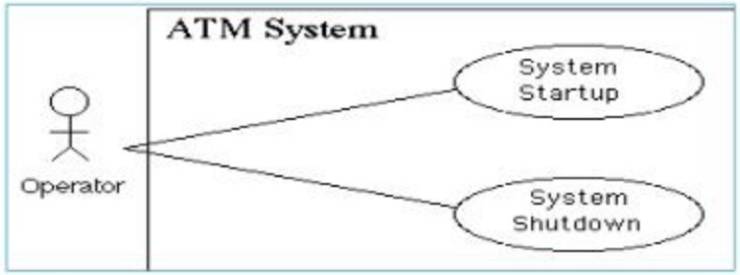


Class Diagram Example



หลักการในการสร้าง Class Diagram

- กำหนดกรอบของโจทย์ให้ชัดเจน หา class ที่มีทั้งหมด
- เขียน use case diagram ของโจทย์ที่กำหนดไว้
- Actor 1 Actor คือ Class 1 Class
- พิจารณาว่าในแต่ละ Use Case มี Class ใดอยู่บ้าง
- ทำให้ครบทุก Use Case



Class ที่ได้ คือ

Actor
- Operator

System Startup
- ATM Machine
- (Printer)

Class ที่ได้ คือ

System Shutdown
- ATM Machine
- (Printer)

หลักการในการสร้าง Class Diagram

- ทำการสร้าง Class โดยใช้หลักการต่อไปนี้

✓ หา Attribute และ Method ที่มีอยู่ใน Class นั้นๆ โดยพิจารณาเฉพาะข้อมูลที่จำเป็น

✓ วาด Class ที่ได้ลงใน Class diagram

✓ หาค่า Visibility ให้กับ Attribute และ Method

หา Attribute และ Method

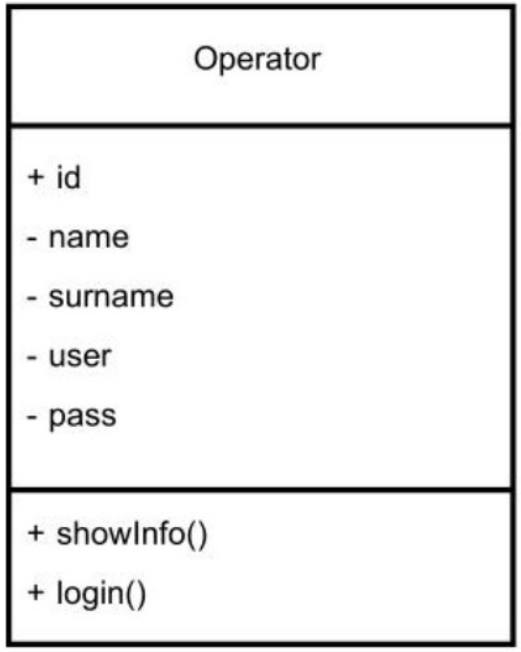
Class Operator

Attribute

- รหัส (id)
- ชื่อ (name)
- นามสกุล (surname)
- Username (user)
- Password (pass)

Method

- แสดงข้อมูล (showInfo)
- เข้าสู่ระบบ (login)



การสร้าง Class Diagram

- Problem Domain ที่กำหนด คือ
  - ในคณะวิชาวิทยาศาสตร์ของสถาบันแห่งหนึ่งมีบุคลากรหลายประเภทด้วยกัน ได้แก่ อาจารย์ นิสิต และเจ้าหน้าที่ โดยที่อาจารย์แต่ละท่านมีหน้าที่ในการสอนวิชาใดวิชาหนึ่งหรือมากกว่า 1 วิชาก็ได้ และนักศึกษาก็มีหน้าที่ในการศึกษาวิชาหนึ่งหรือมากกว่า 1 วิชาก็ได้ ในเวลาเดียวกันเจ้าหน้าที่ของคณะ คือเจ้าหน้าที่ประจำห้องทดลองต่างๆ โดยกำหนดว่าใน 1 ห้องทดลองจะต้องมีเจ้าหน้าที่ 1 คนเสมอ

แนวทางการวิเคราะห์

- พิจารณาผู้ที่เกี่ยวข้องกับระบบก่อนว่ามีใครบ้าง
- พิจารณากระบวนการต่างๆที่เกิดขึ้นในระบบ
- วาด use case diagram เพื่อพิจารณาภาพรวมของระบบ
- พิจารณา Object หรือ Class ต่างๆ ที่ใช้ โดยพิจารณาจาก

    Actor ใน Use Case Diagram

    Object และ Class จาก Use Case ต่างๆ
- กำหนด Object หรือ Class ใน Class Diagram
- สร้างความสัมพันธ์ของ Class
- กำหนด Attribute และ Method ที่เกี่ยวข้องกับระบบ

Thanks!

Any questions?