

UML Graphical Notation to describe classes and objects

Class

Customer
- name: String - address : String - id : Integer - city : String
+ getCustomId() : Integer + getCustomerName() : String + getAddress : String

```
public class Customer
{
    private String name;
    private String address;
    private Integer id;
    private String city;

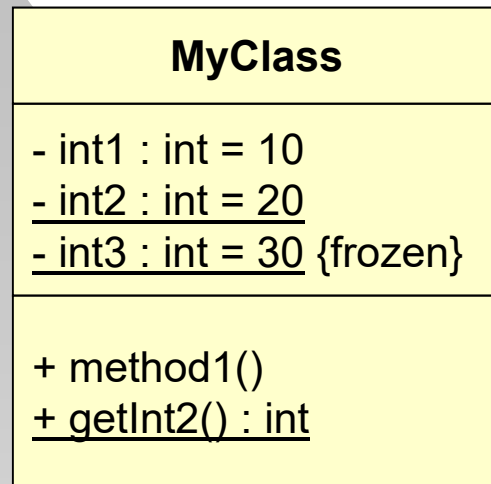
    public Integer getCustomerId()
    {
        .....
    }

    public String getCustomerName()
    {
        .....
    }

    public String getAddress()
    {
        .....
    }

}
```

UML instance-scope / class-scope



Attribute **int2** and **int3** are marked as class-scoped, class-scope is indicated by underlining.

Read-only

int3 is class-scoped, often “frozen” attributes are made class-scoped.

Method **getInt2** is class-scoped; it can only access class-scoped attributes and methods.

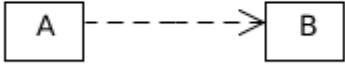
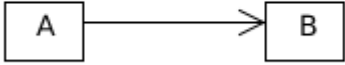
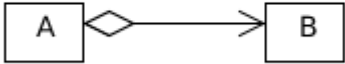
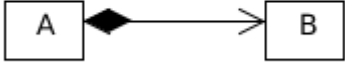
```
public class MyClass {  
    private int int1 = 10;  
    private static int int2 = 20;  
    private static final int int3 = 30;  
    public void method1() { int1 = 20;};  
    public static int getInt2() { return int2;};  
}
```

The Need for Relationships

- ระบบเชิงวัตถุทุก ๆ ระบบประกอบไปด้วยออบเจกต์และคลาสต่าง ๆ
- การทำงานของระบบสามารถทำได้โดยการติดต่อกันระหว่างออบเจกต์ภายในระบบ ผ่านความสัมพันธ์ระหว่างกัน
- เพื่อให้คลาสหนึ่งสามารถส่ง 메시지를ไปยังคลาสอื่น ๆ จำเป็นจะต้องมีความสัมพันธ์ระหว่างคลาสทั้งสอง
- ความสัมพันธ์มีลักษณะเป็น semantic ระหว่างคลาส โดยคลาสหนึ่งสามารถทราบรายละเอียดของแอตทริบิวต์และเมธอดของคลาสอื่น
- ชนิดความสัมพันธ์ระหว่างคลาสมีดังนี้

Dependency, Association, Aggregation และ Composition

Relationships

Relationship	Depiction	Interpretation
Dependency		<p>A depends on B</p> <p>This is a very loose relationship and so I rarely use it, but it's good to recognize and be able to read it.</p>
Association		<p>An A sends messages to a B</p> <p>Associations imply a direct communication path. In programming terms, it means instances of A can call methods of instances of B, for example, if a B is passed to a method of an A.</p>
Aggregation		<p>An A is made up of B</p> <p>This is a part-to-whole relationship, where A is the whole and B is the part. In code, this essentially implies A has fields of type B.</p>
Composition		<p>An A is made up of B with lifetime dependency</p> <p>That is, A aggregates B, and if the A is destroyed, its B are destroyed as well.</p>

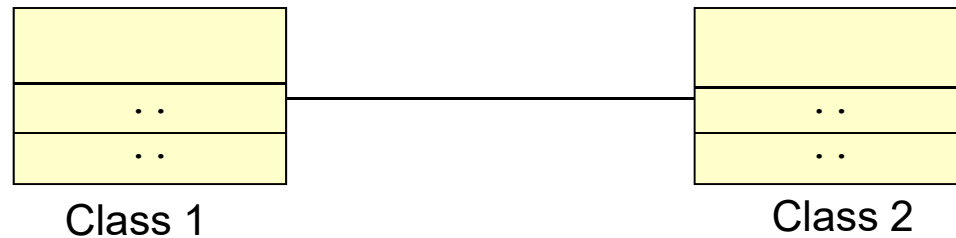
Association Relationship

- ความสัมพันธ์แบบนี้จะคล้ายกับความสัมพันธ์แบบ Dependency แต่จะแตกต่างกันตรงที่ช่วงเวลาที่เกิดความสัมพันธ์กันจะมีระยะเวลานาน และมีลักษณะเป็นความสัมพันธ์แบบโครงสร้าง
- ความสัมพันธ์แบบนี้อาจสร้างขึ้นได้โดยการกำหนดคลาสอื่นในรูปของแอตทริบิวต์ภายในคลาส และสามารถเรียกใช้เมธอดจากคลาสนั้นได้ในกรณีที่ต้องการ
- ความสัมพันธ์แบบนี้จะยอมให้มีการกำหนดทิศทางความสัมพันธ์ระหว่างคลาสได้ทั้งแบบ bi-directional หรือ unidirectional
- ความสัมพันธ์แบบนี้อาจถูกมองในรูปของการมีอยู่ (**has-a**) ได้ เช่น ออกไปจากคลาสหนึ่งมีค่าอ้างอิงไปยังออกไปจากคลาสอื่นได้

Association Direction

- นำเสนอผ่านสัญลักษณ์ลูกศรสองทิศทาง หรือไม่มีหัวลูกศรทั้งสองด้าน

Bi-directional



- ส่วนการกำหนดหัวลูกศรทางเดียวไว้จะเป็นความสัมพันธ์แบบ Uni-direction

Uni-directional



Association End

- หมายถึง end ของการ association เมื่อมีความสัมพันธ์ระหว่างคลาสเกิดขึ้น
- ถือเป็นส่วนหนึ่งของ association แต่ไม่ใช่ส่วนหนึ่งของคลาส
- สัญลักษณ์ต่าง ๆ ที่สามารถนำเสนอประกอบใน association end ได้ เช่น:
 - Multiplicity
 - Navigability
 - Aggregation indicator
 - Role name

Naming Relationships

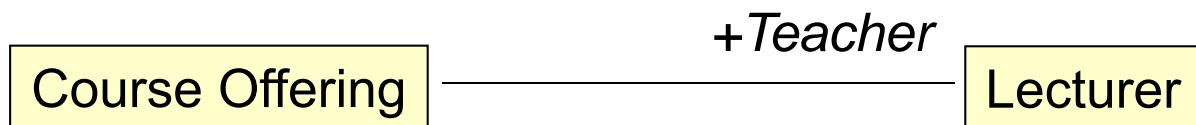
- ความสัมพันธ์สามารถกำหนดชื่อได้
- โดยปกติแล้วชื่อจะถูกกำหนดด้วย **active verb** หรือ **verb phrase** ที่ใช้สื่อความหมายในการติดต่อกันของความสัมพันธ์ที่ปรากฏ เช่น

a Lecturer **teaches** a Course

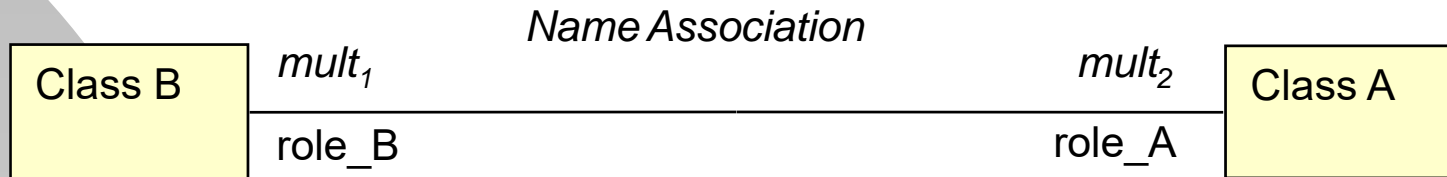
- ชื่อความสัมพันธ์ถือเป็น optional
- ชื่อจะถูกเพิ่มเข้าไปเพื่อช่วยเพิ่มความชัดเจนของความสัมพันธ์
- สำหรับความสัมพันธ์แบบ Aggregation โดยปกติจะไม่มีกำหนดชื่อไว้ แต่จะเป็นการระบุโดยใช้คำประเภท “has”, “part of” หรือ “contains” เป็นต้น

Role Names

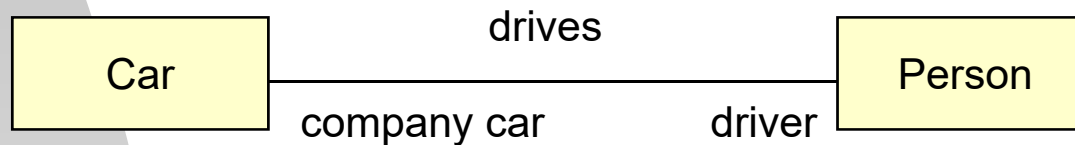
- บริเวณปลายเส้นความสัมพันธ์ที่ติดกับคลาสจะถูกเรียกว่า **association role**
- **Role name** สามารถใช้แทนชื่อความสัมพันธ์ได้
- โดยอยู่ในรูปของคำนามที่แสดงถึงเหตุผลของความสัมพันธ์ที่เกิดขึ้น
- **Role name** จะถูกกำหนดตำแหน่งบนความสัมพันธ์ใกล้ ๆ กับคลาสที่มีการ modifies
- **Role name** อาจถูกกำหนดไว้ที่ด้านใดด้านหนึ่งหรือทั้งสองด้านของเส้นแสดงความสัมพันธ์กัน
- ทั้ง role name และ association name ถือเป็น Optional



Basic notation for associations



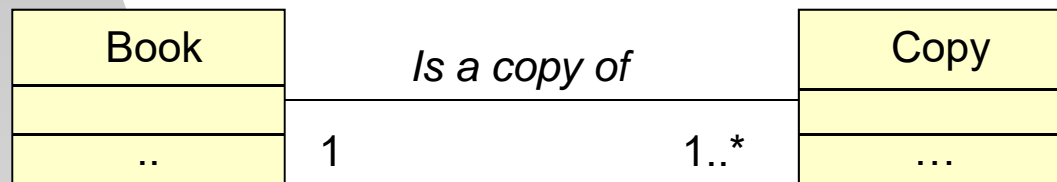
Example:



- person จะเป็นบทบาทของ driver และ car จะเป็นบทบาทของ company car ซึ่งจะเป็นข้อมูลของออบเจกต์ที่ต้องกระทำ

Multiplicity

- เป็นการระบุจำนวนของ instances จากคลาส ถัดจากสัญลักษณ์แสดงจำนวน multiplicity ปรากฏ จะเป็นการอ้างอิงโดยใช้ single instance ของคลาสซึ่งอยู่ด้านท้ายสุดของ association path อีกด้านหนึ่ง
- ระบุ lower และ upper bound ของจำนวน instances



ทุก ๆ copy ออปเจก (จากคลาส copy) จะถูก associated โดย “is a copy of” จาก Book เพียงหนึ่งจากคลาส Book

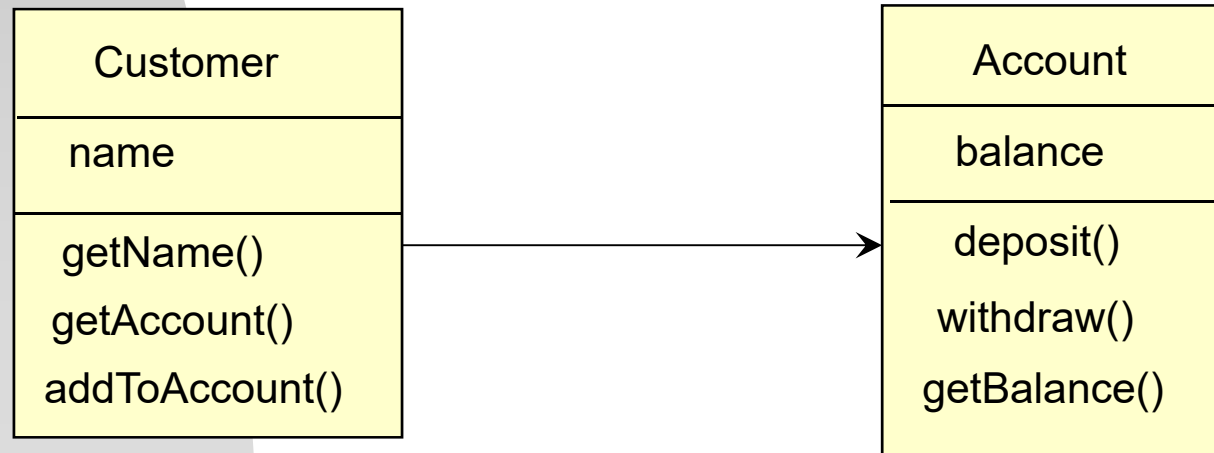
Multiplicity

- สัญลักษณ์แสดงการระบุจำนวนความสัมพันธ์

Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

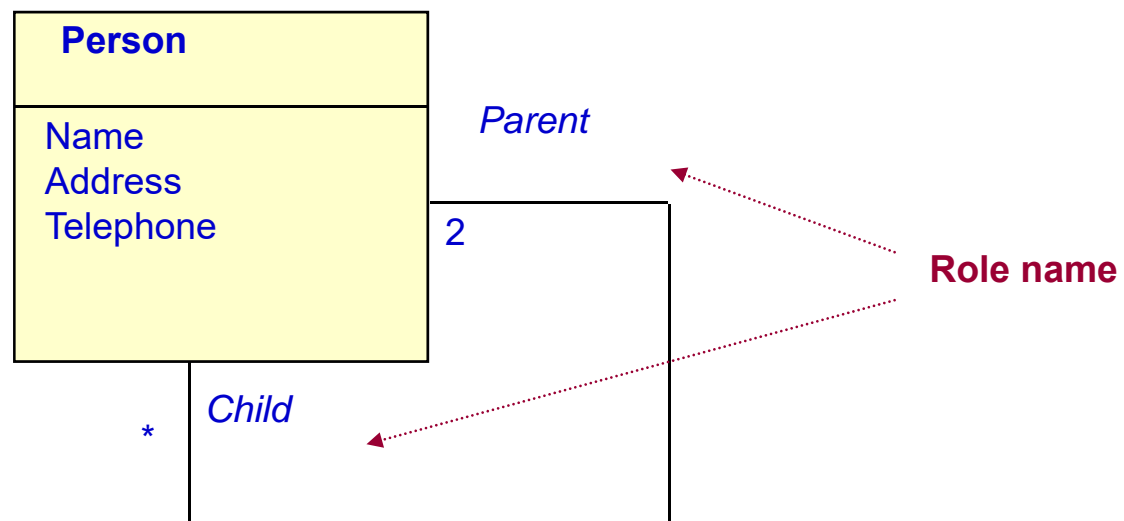
Navigability

- โดยปกติแล้วความสัมพันธ์ระหว่างออปเจกจะเป็นไปในทิศทางเดียวเป็นส่วนใหญ่
- ตัวอย่างเช่น Customer ออปเจกต้องการใช้บริการจาก Account ออปเจก สัญลักษณ์ลูกศรจะเป็นการระบุว่า Customer ออปเจกส่งเมสเสจไปยัง Account ออปเจกเท่านั้น

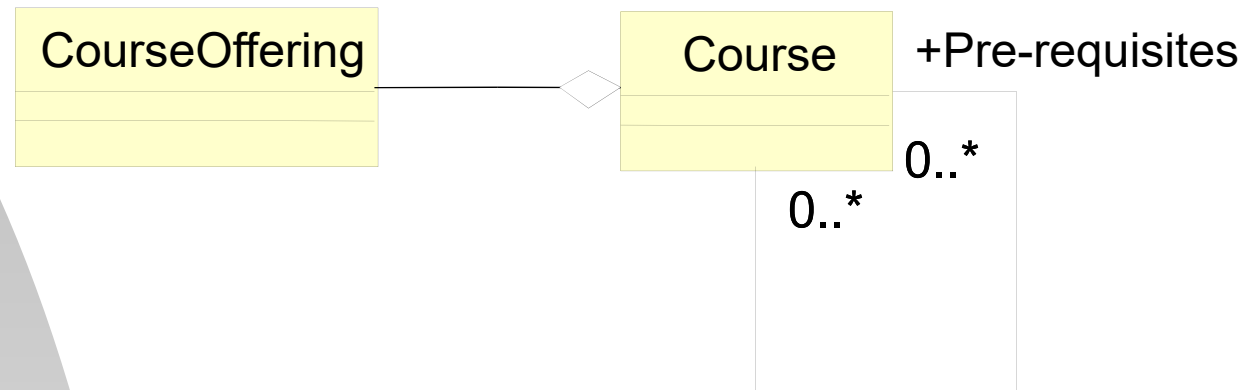


Reflexive relationships

- ❑ คลาสอาจมีความสัมพันธ์กับตัวเองได้ เช่น ในกรณีที่คลาสมีออบเจกต์ที่สามารถเป็นได้หลาย roles
- ❑ ความสัมพันธ์แบบนี้ถือเป็น Reflexive Association ที่สร้างจากคลาสเดียวกัน
- ❑ ชื่อของ Role โดยปกติจะถูกใช้สำหรับ reflexive relationships

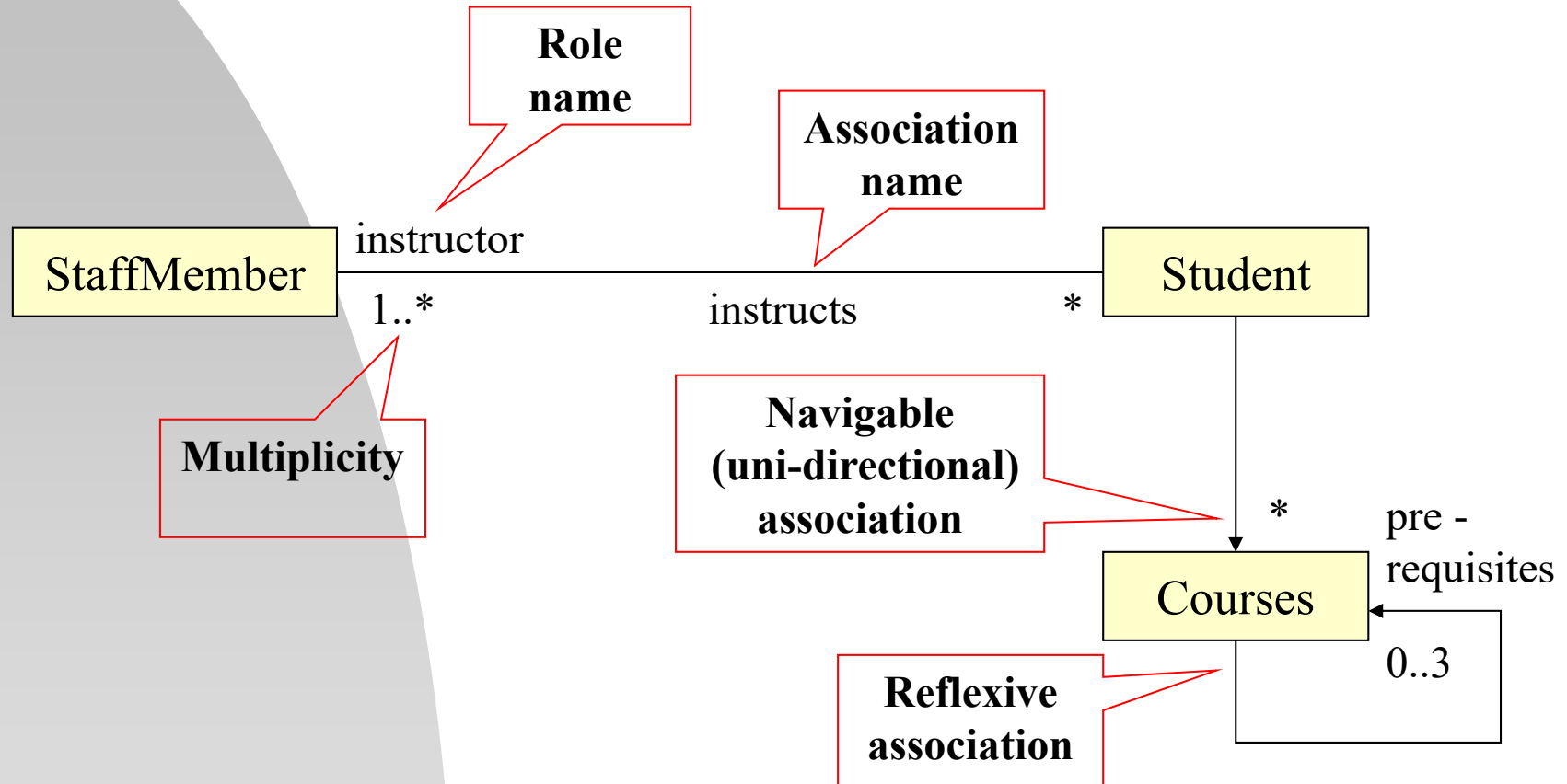


Reflexive relationships cont..



- หนึ่งใน Course ออบเจกต์สามารถเป็น role ของ Prerequisite ที่เกี่ยวข้องกับ Course ออบเจกต์ตั้งแต่ 0 หรือมากกว่า
- หนึ่งใน Course ออบเจกต์จะเกี่ยวข้องกับ Course ออบเจกต์ตั้งแต่ 0 หรือมากกว่าที่มี role เป็น Prerequisite

Associations (cont.)

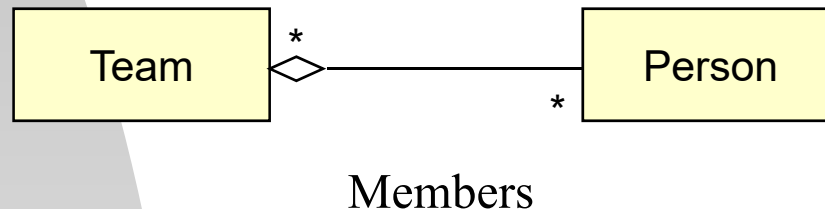


Aggregation

- ความสัมพันธ์แบบ Association สามารถมีได้ 2 รูปแบบ ได้แก่ Aggregation และ Composition
- Aggregation
 - ออบเจกต์หนึ่งมีออบเจกต์อื่นๆ เป็นส่วนหนึ่งของมัน
 - เช่น Car มี engine และ wheels เป็นส่วนหนึ่งของ Car
- Composition
 - ออบเจกต์หนึ่งมีออบเจกต์อื่นๆ เป็นองค์ประกอบของมัน
 - ถ้า delete วัตถุที่เป็นองค์ประกอบรวม ส่วนย่อยจะต้องถูก delete ไปด้วย

What is Aggregation?

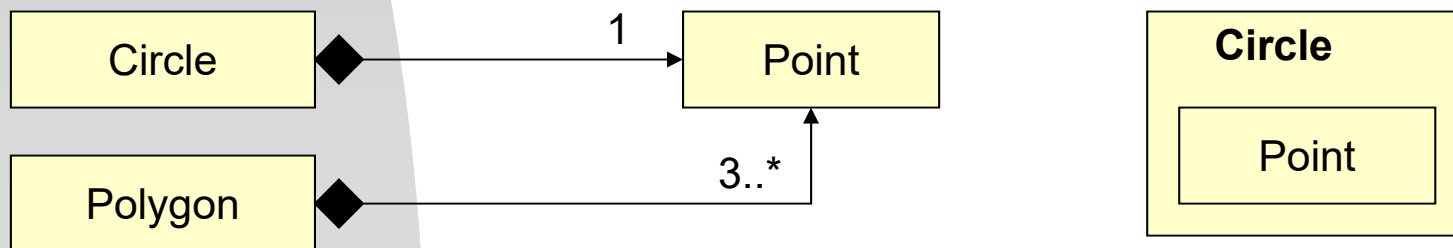
- ความสัมพันธ์แบบ Aggregation บางครั้งจะถูกเรียกว่า Shared Aggregation
- เป็นความสัมพันธ์ในลักษณะที่เรียกว่า “whole-part” ซึ่งประกอบไปด้วยคำประเภทต่อไปนี้ “consists of”, “contains”, “is part of”
- ส่วนที่เป็น parts อาจเป็นส่วนประกอบของส่วนที่เป็น wholes อื่น ๆ ได้



- team ประกอบไปด้วย member ภายในทีม
- person หนึ่งสามารถเป็น member ของทีมอื่น ๆ ได้
- person จะถือเป็นส่วนของการ shared parts

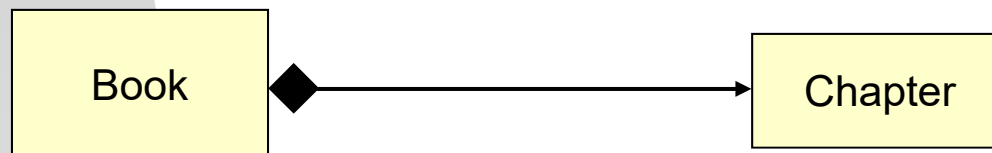
Composition

- เป็นความสัมพันธ์แบบ aggregation ที่มีความแน่นหนามากขึ้น
 - โดยส่วนที่เป็น whole จะทำหน้าที่เป็นเจ้าของส่วนที่เป็น part
 - ส่วนที่เป็น part อาจเป็นส่วนที่เป็น whole ได้เพียงหนึ่งเดียว
 - Multiplicity ของส่วนที่เป็น whole จะต้องมีค่าเป็น 0 หรือ 1 เท่านั้น
 - ช่วงชีวิตของส่วนที่เป็น part จะขึ้นอยู่กับส่วนที่เป็น whole เสมอ นั่นคือจะเกี่ยวข้องการสร้างหรือลบส่วนที่เป็น part เสมอ



Composite Relationship

- สัญลักษณ์ diamond สีดำจะใช้ในการนำเสนอความสัมพันธ์แบบ **composition** ซึ่งจะถูกกำหนดไว้ทางด้าน Book เนื่องจาก Book จะประกอบไปด้วย Chapter
- สัญลักษณ์ลูกศรแสดงถึงความสัมพันธ์เป็นแบบทางเดียว นั่นคือ
 - Chapter ไม่รู้จักส่วนที่เป็น Book
- ใน UML ความสัมพันธ์จะถูกตั้งสมมุติฐานไว้เป็นแบบ **bidirectional** เสมอ เว้นแต่จะมีการกำหนดสัญลักษณ์ลูกศรเพื่อจำกัดไว้



Aggregation & Composition

- เป็นความสัมพันธ์แบบ transitive
 - ถ้าวัตถุ A เป็นส่วนหนึ่งของ B และ B เป็นส่วนหนึ่งของ C แล้ว A เป็นส่วนหนึ่งของ C ด้วย
 - เช่น ถ้าที่จับประตูเป็นส่วนหนึ่งของประตู ประตูเป็นส่วนหนึ่งของรถยนต์ แล้ว ที่จับประตูเป็นส่วนหนึ่งของรถยนต์
- เป็นความสัมพันธ์แบบ Anti-symmetric
 - วัตถุใดๆ อาจไม่จำเป็นต้องเป็นส่วนประกอบของตัวเองทั้งทางตรงและทางอ้อม
 - เช่น ถ้าประตูเป็นส่วนหนึ่งของรถยนต์ แต่รถยนต์ไม่จัดเป็นส่วนหนึ่งของประตู