CS204362 – Object-Oriented Design

UML Class Diagrams
ภาคต่อ

Kamonphop Srisopha

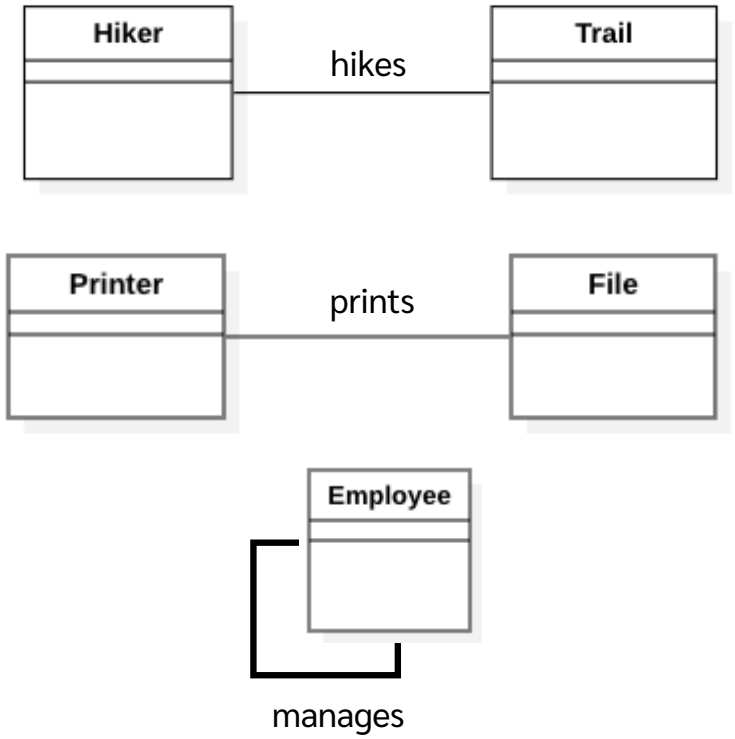
COMPUTER SCIENCE
Chiang Mai University

Faculty of Science, Chiang Mai University
คณะวิทยาศาสตร์ มหาวิทยาลัยเชียงใหม่

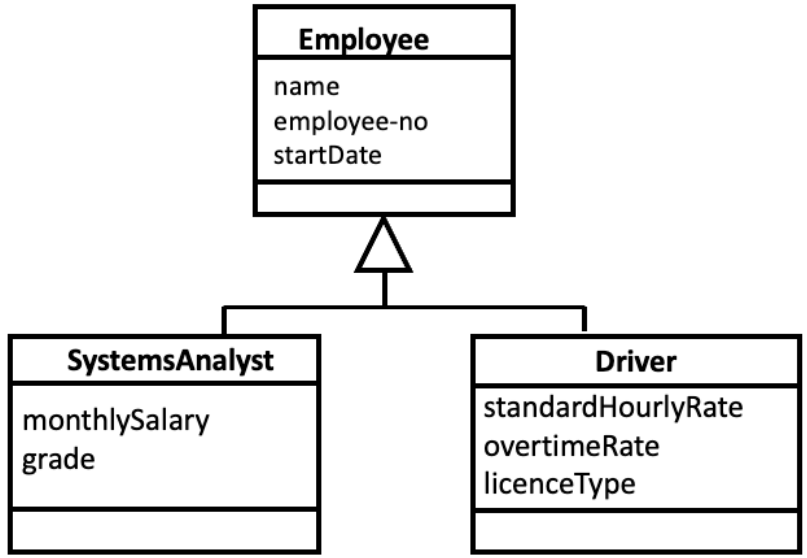
Relationships between Classes

ความสัมพันธ์ระหว่างคลาสแบ่งเป็น 2 ลักษณะใหญ่ๆ ได้ดังนี้

Association

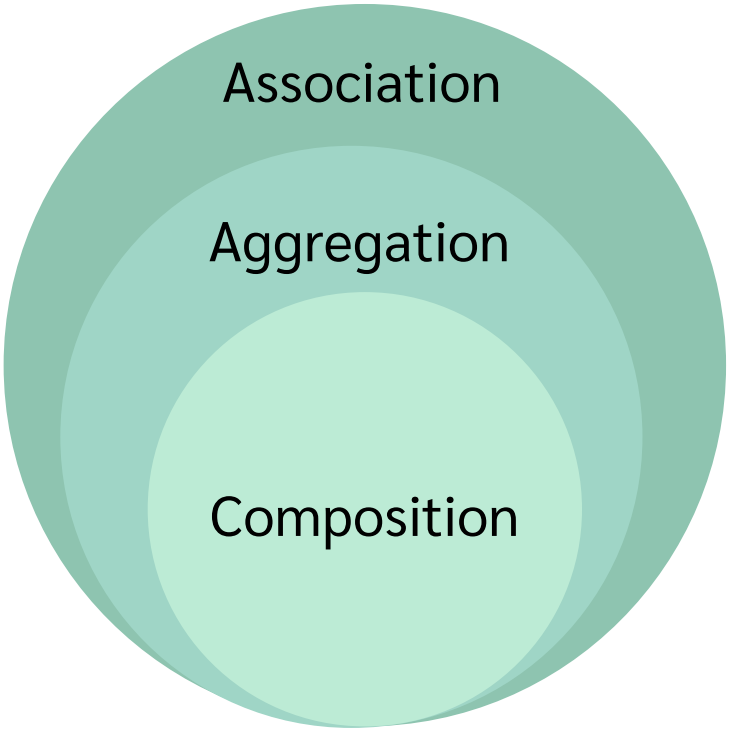


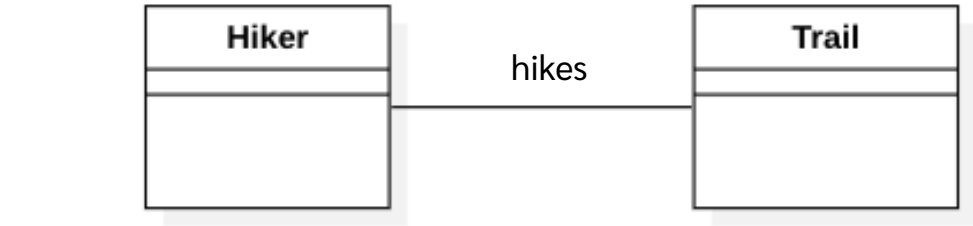
Generalization

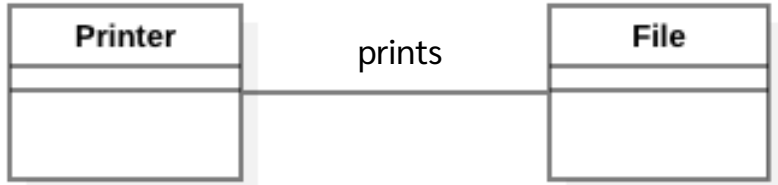


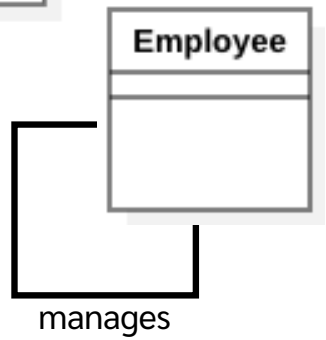
Association Relationships

Association Relationships









- ความสัมพันธ์ระหว่างคลาสอย่างง่ายและพื้นฐานที่สุด
- ความสัมพันธ์ peer-to-peer
- เพิ่มความจำเพาะของความสัมพันธ์ได้เป็น Aggregation และ Composition
- ความสัมพันธ์แบบ Aggregation กับ Composition ก็เรียกว่าเป็น Association เช่นกัน

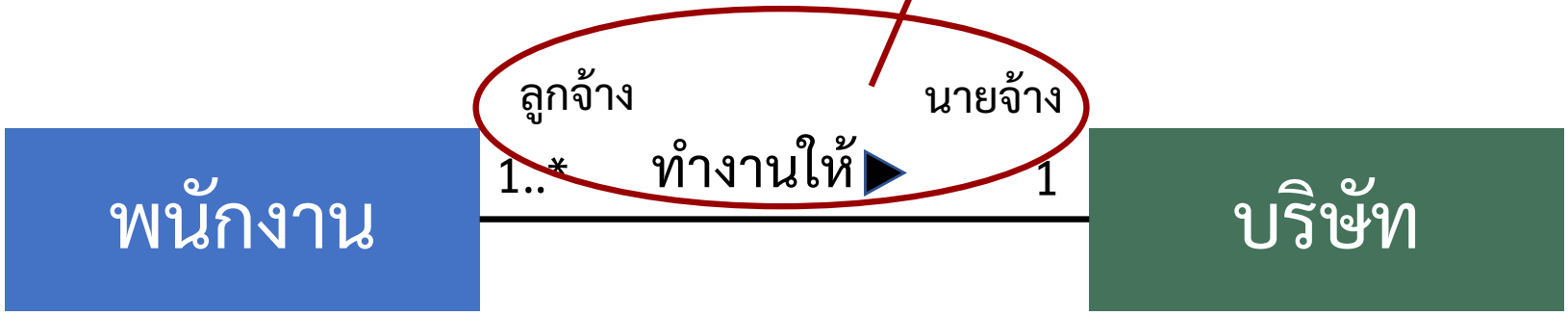
Multiplicity of Associations

ปริมาณความสัมพันธ์ บ่งบอกว่า หนึ่ง instance ของคลาสด้านหนึ่งของความสัมพันธ์ สามารถมีความสัมพันธ์กับกี่ instance ของคลาสอีกด้านหนึ่งของความสัมพันธ์ได้

ตัวอย่าง

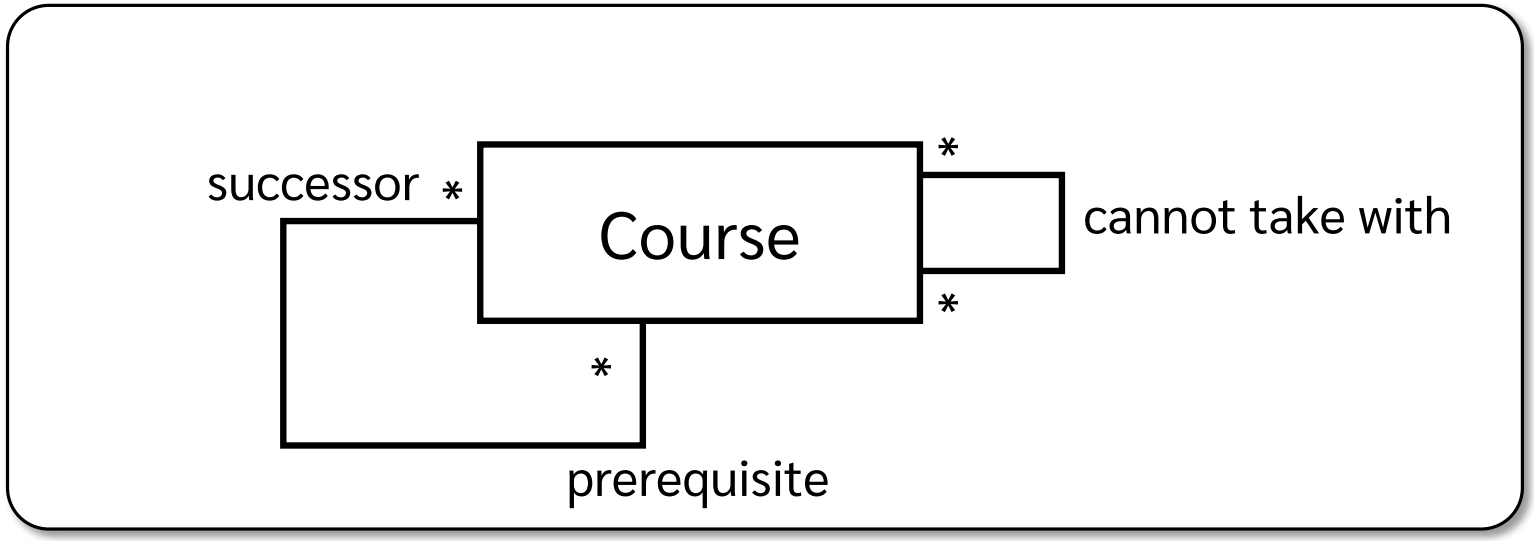
- * หมายถึง 0 หรือมากกว่า
- 1 หมายถึง 1 เท่านั้น
- 2..4 หมายถึง ระหว่าง [2,4]
- 3..* หมายถึง 3 ขึ้นไป

สามารถใส่ชื่อเพื่อบ่งบอกความเกี่ยวข้อง (association name) บทบาทความเกี่ยวข้อง (role) รวมถึงทิศทางของการเกี่ยวข้องได้ด้วย (ให้ง่ายต่อการอ่านและเข้าใจ)



Reflexive Associations

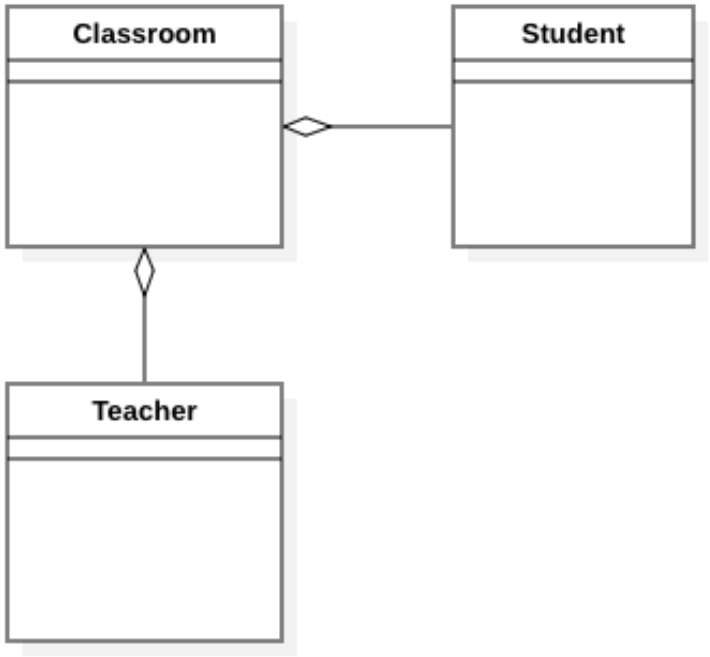
คลาสหนึ่งๆสามารถมีปฏิสัมพันธ์กับตัวมันเองได้



เช่น จะเรียน CS362 ต้องเรียน CS361 มาก่อน

บางวิชาอาจจะมีกฎว่าห้ามเรียนพร้อมกัน

Association Type: Aggregration



A Classroom has a Student(s)
A Classroom has a Teacher(s)

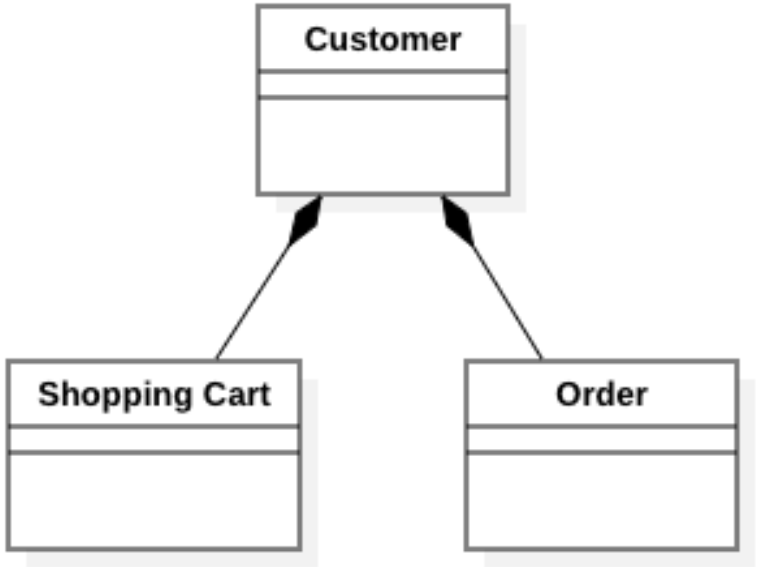
A Student and a Teacher can exist independently from a classroom

ความสัมพันธ์ระหว่างกันที่บ่งบอกความ**มีหรือความเป็นส่วนประกอบ (แต่แยกจากกันได้)**

Notation: ใช้สัญลักษณ์ลูกศรหัวข้าวหลามตัดโปร่งลากจากคลาสย่อยไปยังคลาสหลัก

ลักษณะความสัมพันธ์ คือ ถ้าไม่มีคลาสย่อย คลาสหลักก็ไม่สูญหายหรืออยู่ไม่ได้ ในทางกลับกัน ถึงแม้ไม่มีคลาสหลัก คลาสย่อยก็ยังอยู่ได้

Association Type: Composition



A customer “**owns**” a shopping cart and an order(s)

The shopping cart and order have no meaning in the model without a customer

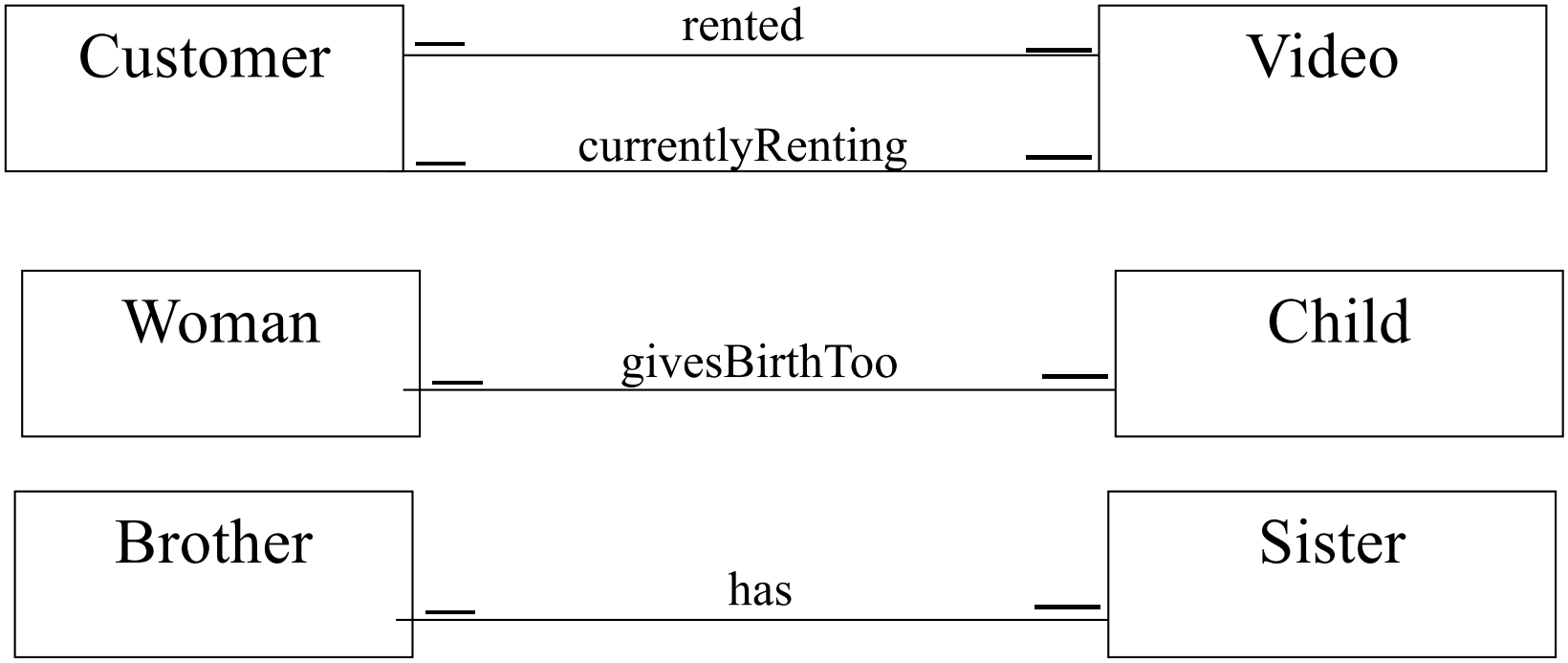
ความสัมพันธ์คล้ายกับ Aggregation แต่เป็นแบบที่จำเพาะกว่า เพราะมีการแสดง**ความเป็นเจ้าของ**ระหว่างความสัมพันธ์ (แยกจากกันไม่ได้)

Notation: ใช้สัญลักษณ์ลูกศรหัวข้าวหลามตัดทึบลากจากคลาสย่อยไปยังคลาสหลัก

ลักษณะของความสัมพันธ์ คือ เมื่อคลาสหลักสูญหายไป คลาสย่อยก็อยู่ไม่ได้ กล่าวอีกนัยคือคลาสย่อยจะต้องทำงานร่วมกับคลาสหลักไปจนตลอดอายุของคลาสหลัก

Check Your Understanding

เราจะใส่ค่าปริมาณความสัมพันธ์ของ class เหล่านี้อย่างไรดี?



Check Your Understanding

ให้นักศึกษาออกแบบ Class Diagram โดยใช้ 2 หรือ 3 class ที่มีปฏิสัมพันธ์กันจาก scenario ต่อไปนี้:

1. A landlord renting apartments to tenant
2. An author writing books distributed by publishers
3. People registering for fitness activities at a gym.

- ให้ระบุ ปริมาณความสัมพันธ์ (และต้องอธิบายได้ด้วยว่าเลือกปริมาณนี้เพราะอะไร?)
- ในแต่ละ class ให้ระบุ 1 attribute ด้วย

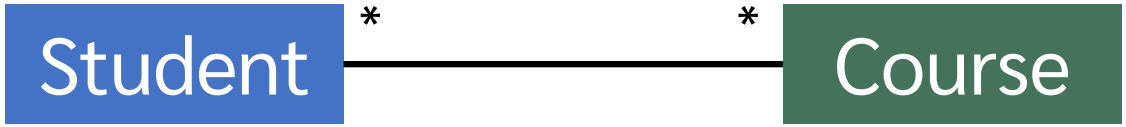
Association Classes

ในบางสถานการณ์ เราไม่สามารถวาง attribute ที่เกี่ยวข้องกับสองคลาสไว้ในคลาสใดคลาสหนึ่งได้

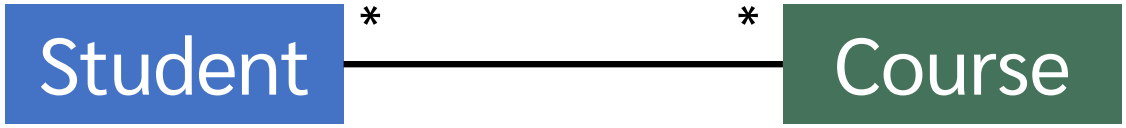
ตัวอย่างเช่น จากภาพด่านล่าง นักเรียนสามารถลงเรียนหลายกระบวนวิชา และแต่ละกระบวนวิชาสามารถมีนักเรียนจำนวนเท่าใดก็ได้

ปัญหาคือ เราควรจะนำ grade (attribute) ไปวางไว้ในคลาสไหน?

ถ้าเอาไปไว้ใน *Student* – แต่ละ student ก็จะมีแค่ grade เดียว ไม่ใช่แต่ละ course แต่ถ้าเอาไปไว้ใน *Course* แต่ละ course ก็จะมีแค่ grade เดียว ไม่ใช่ แต่ละ student.



Association Classes

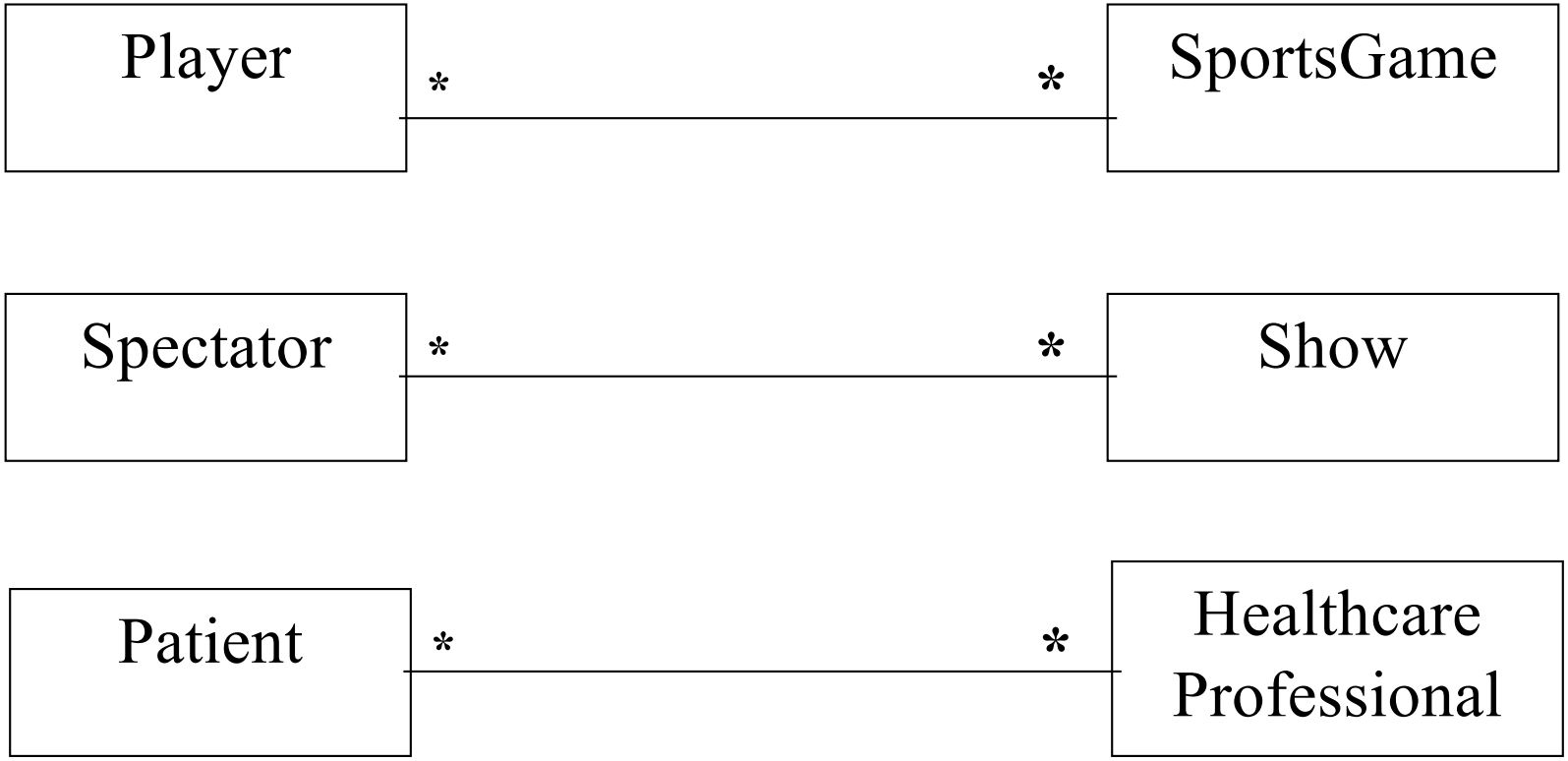


แก้ปัญหานี้โดยการสร้าง **Association Class**

ฉะนั้นเมื่อไหร่ก็ตามที่เราต้องเจอความสัมพันธ์แบบ many-to-many เราควรต้องคิดด้วยว่าจำเป็นต้องมีการสร้าง association class หรือไม่

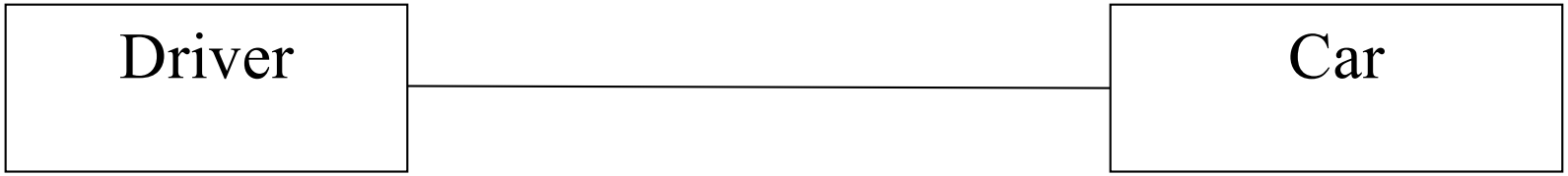
Check Your Understanding

ให้นักศึกษาหา attribute ที่ไม่สามารถใส่ลงไปได้ในคลาสใด คลาสหนึ่งในสองคลาสที่มีความสัมพันธ์แบบ many-to-many ด้านล่างนี้ และให้ระบุ associate class ที่เหมาะสมมาด้วย



Directionality in associations

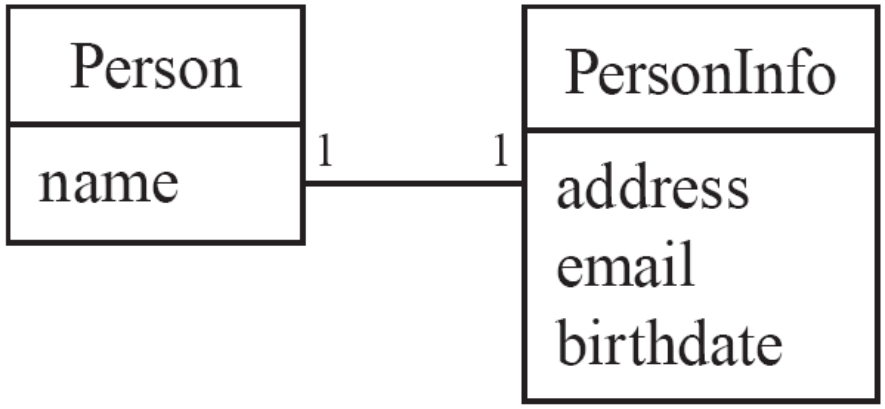
ความสัมพันธ์แบบ association ปกติแล้วจะเป็นแบบ **bi-directional** เช่น ในรูปแผนภาพด่านล่าง ถ้าเรารู้ว่าใครเป็นคนขับ เราก็จะรู้ว่ารถของเขาคือคันไหน และถ้าเรารู้รถสักคันนึงเราก็จะรู้ว่าใครเป็นคนขับด้วย



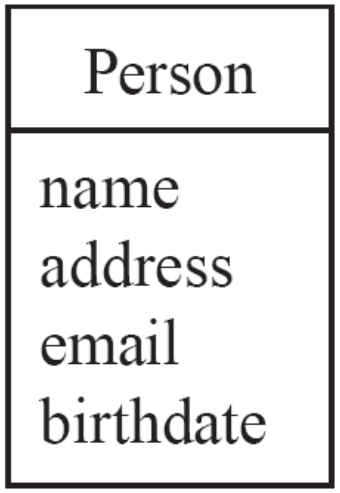
แต่เราก็สามารถบอกทิศทางของความสัมพันธ์ได้ โดยการใส่ลูกศรบอกทิศทาง เช่น ตัวอย่างด้านล่างบ่งบอง 2 class ที่อาจจะมีอยู่ในระบบ Calendar (ปฏิทิน) โดยที่ผู้ใช้สามารถใส่ note ไว้ในวันไหนก็ได้ในปฏิทิน ในกรณีนี้ **Day** จะรู้การมีอยู่ของปฏิสัมพันธ์กับ **Note** แต่ **Note** ไม่รู้ว่าตนเองมีปฏิสัมพันธ์กับ **Day**

Avoid unnecessary one-to-one

หลีกเลี่ยงการทำแบบนี้



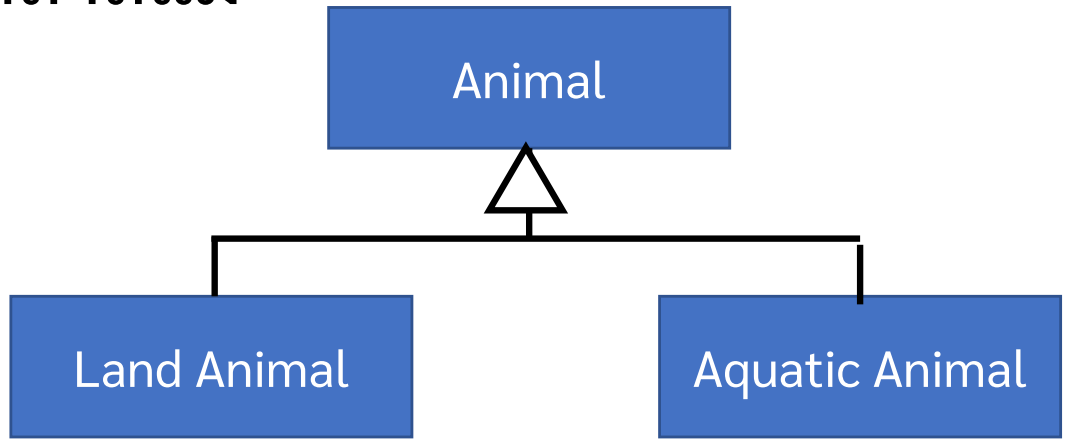
ให้ทำแบบนี้แทน



Generalization Relationships

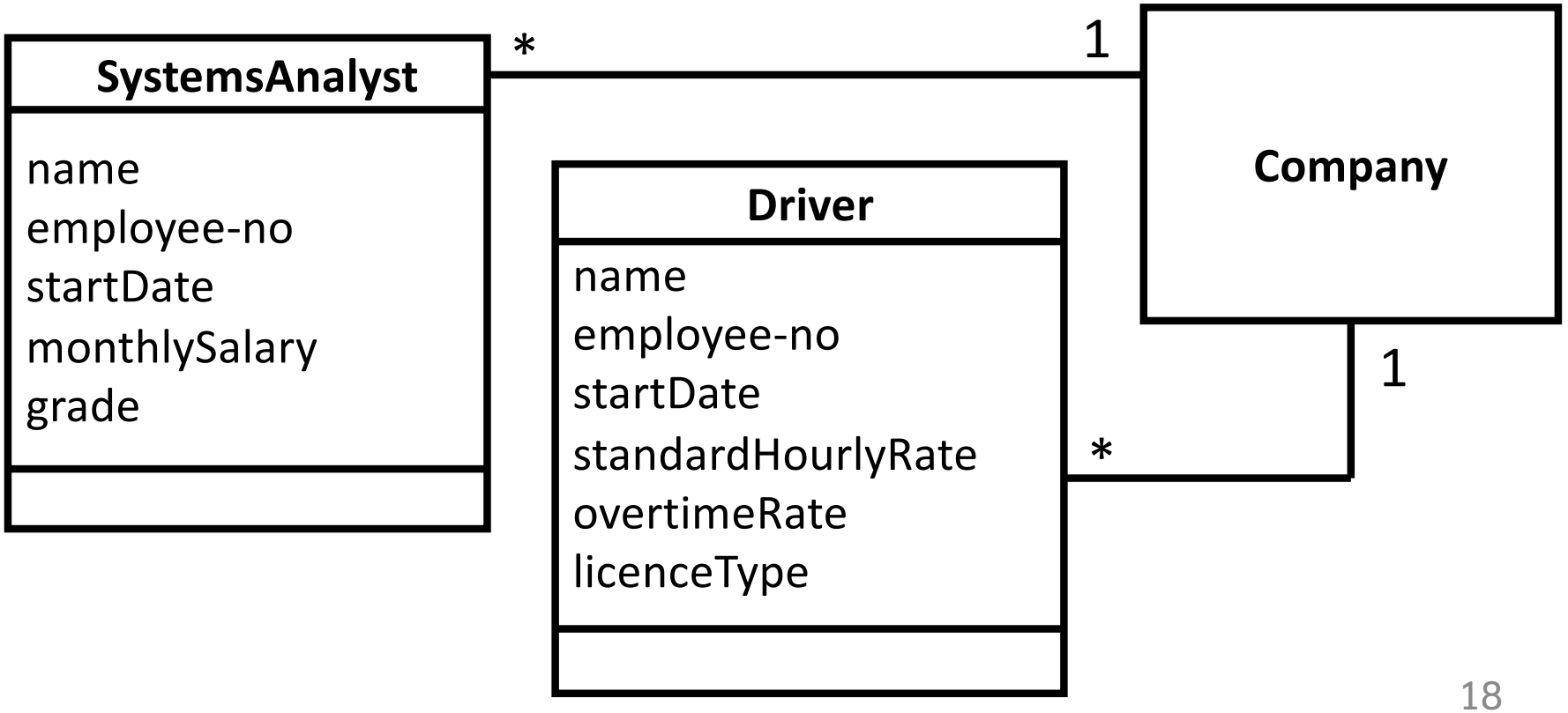
Generalization Relationships

- ระบุว่าคลาสหนึ่ง**เป็นอีกประเภท**ของอีกคลาสหนึ่ง
- สัญลักษณ์เป็นเส้นตรงหัวลูกศรปลายปิดแบบเปิดออกจากคลาสลูก ไปยังคลาสแม่
- สามารถอ่านได้ว่า คลาสลูก เป็นอีกประเภทหนึ่งของคลาสแม่



Generalization Relationships

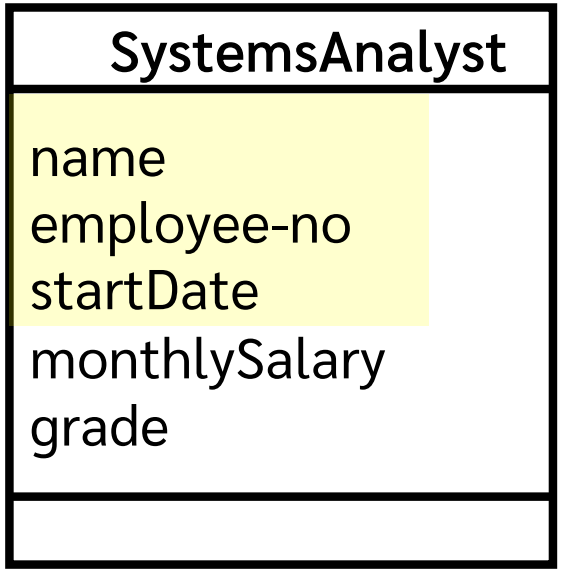
ประเด็นหลักของ Generalization คือ การนำเอาคุณสมบัติที่เหมือนหรือคล้ายกันของหลายๆคลาส มาสร้างเป็นคลาสใหม่ เพื่อที่จะลดการใช้และประกาศซ้ำของข้อมูล และ ลดความซับซ้อนของแผนภาพ

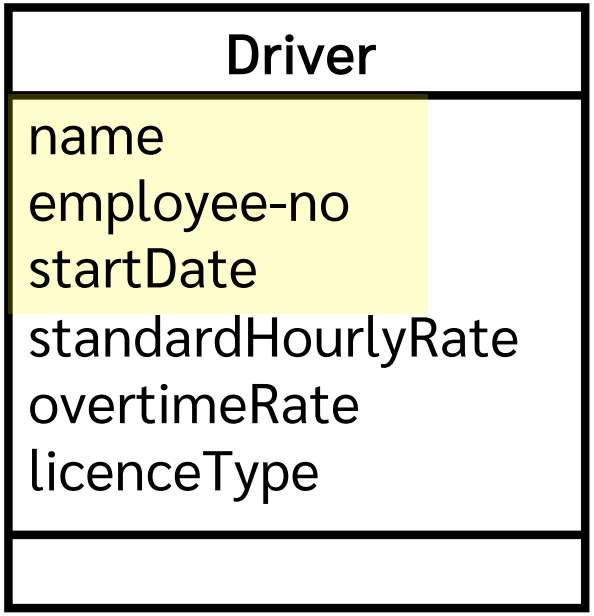


Generalization Relationships

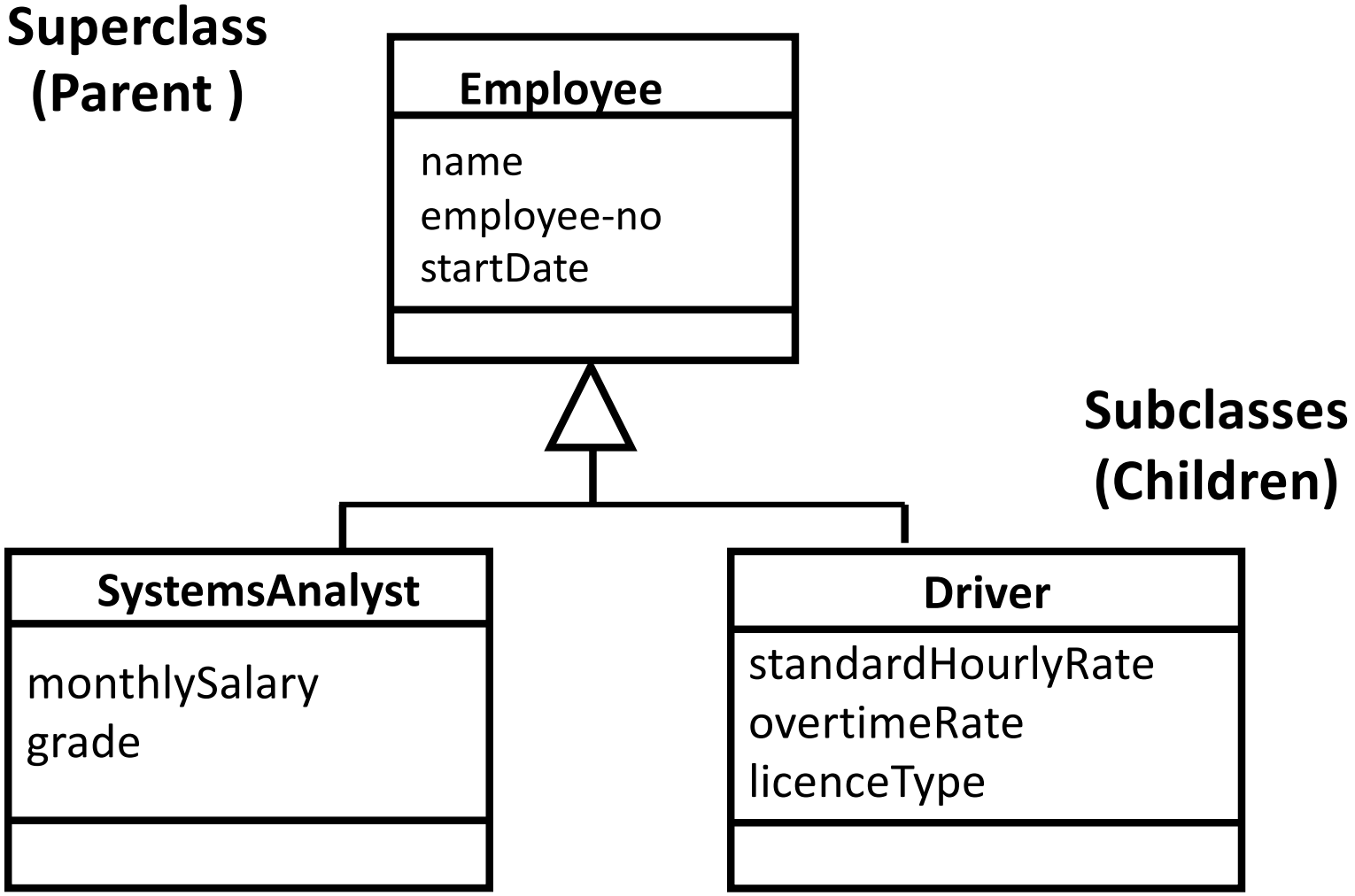
ประเด็นหลักของ Generalization คือ การนำเอาคุณสมบัติที่เหมือนหรือคล้ายกันของหลายๆคลาส มาสร้างเป็นคลาสใหม่ เพื่อที่จะลดการใช้และประกาศซ้ำของข้อมูล

(More general bits of description are *abstracted out* from specialized classes)

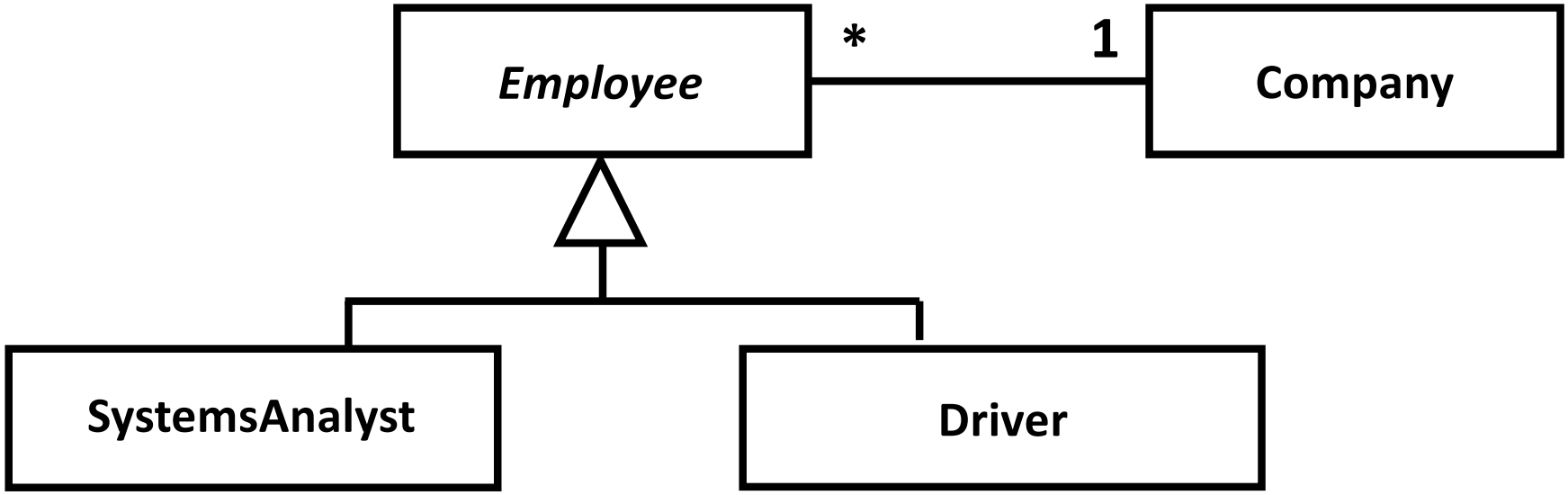




Generalization Relationships



Putting it all together

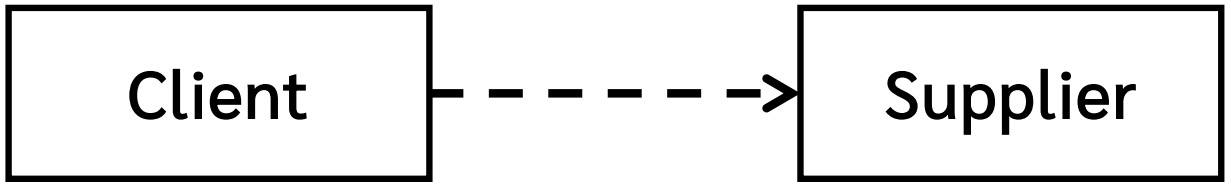


Note: Concept เรื่อง Abstract class, virtual method, abstract method ก็อยู่ในส่วนของ Generalization Relationship

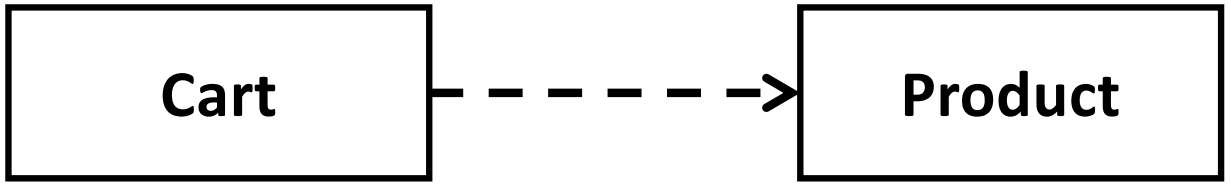
Dependency

อีกความสัมพันธ์ประเภทหนึ่งระหว่างคลาส นอกเหนือจาก association และ generalization ที่เรียกว่าความสัมพันธ์**แบบพึ่งพา**

- เกิดขึ้นได้เมื่อคลาสหนึ่งใช้อีกคลาสหนึงเป็น method parameters
- เกิดขึ้นได้เมื่อคลาสหนึ่งใช้อีกคลาสหนึ่งภายใน method
- ใช้สัญลักษณ์เป็นลูกศรเส้นประหัวเปิดลากจากคลาสที่ใช้งานอีกคลาสไปถึงคลาสที่ถูกใช้งาน
- **หาก มีการเปลี่ยนแปลงใน Supplier, Client ต้องมีการเปลี่ยนแปลงด้วย, แต่ไม่ใช่ในทางกลับกัน**



Example: Dependency Relationships




```
class Cart{
      public void add(Product p){…}
      …
}
```

- คลาส Cart พึ่งพาคลาส Product เนื่องจากคลาส Cart ใช้คลาส Product เป็น parameter สำหรับ method add()
- หาก คลาส Product มีการเปลี่ยนแปลงภายใน คลาส Cart อาจจะต้องมีการเปลี่ยนแปลงภายในด้วย แต่ไม่ใช่ในทางกลับกัน

Dependency Relationship vs Unidirectional Association

	Unidirectional Association	Dependency Relationship
Direction	คลาสหนึ่งรู้เกี่ยวกับอีกคลาสหนึ่ง แต่ไม่ใช่ในทางกลับกัน	คลาสหนึ่งใช้งานคลาสหนึ่ง แต่แค่สั้น ๆ
Reference Storage	คลาสหนึ่งมี reference ของอีกคลาสหนึ่งอยู่	ไม่เก็บข้อมูลของอีกคลาสหนึ่งไว้ (เพียงแค่ใช้งาน ณ ขณะนั้น)
Lifetime	เกิดความสัมพันธ์แบบยาวนาน ส่วนใหญ่จะจนกว่า object หนึ่งจะสูญหายไป	แค่สั้น ๆ หรือ ตอนรัน method

Dependency Relationship vs Unidirectional Association

```java
class Book {
    private String title;

    public Book(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }
}

class Library {
    private List<Book> books;

    public Library() {
        books = new ArrayList<>();
    }

    public void addBook(Book book) {
        books.add(book);
    }

    public List<Book> getBooks() {
        return books;
    }
}
```

Unidirectional Association

```java
class PaymentProcessor {
    public void processPayment(Order order, double amount) {
        // Logic to process the payment
        System.out.println("Processing payment for order: " + order.getId() + " Amo
unt: $" + amount);
    }
}

class Order {
    private String id;

    public Order(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public void pay(PaymentProcessor processor, double amount) {
        processor.processPayment(this, amount);
    }
}
```

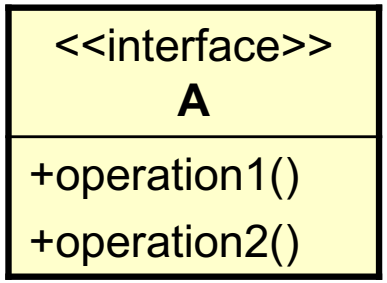
Dependency Relationship

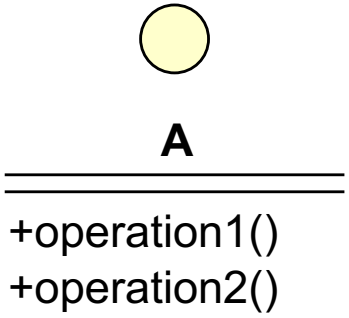
Realization in UML Class Diagram

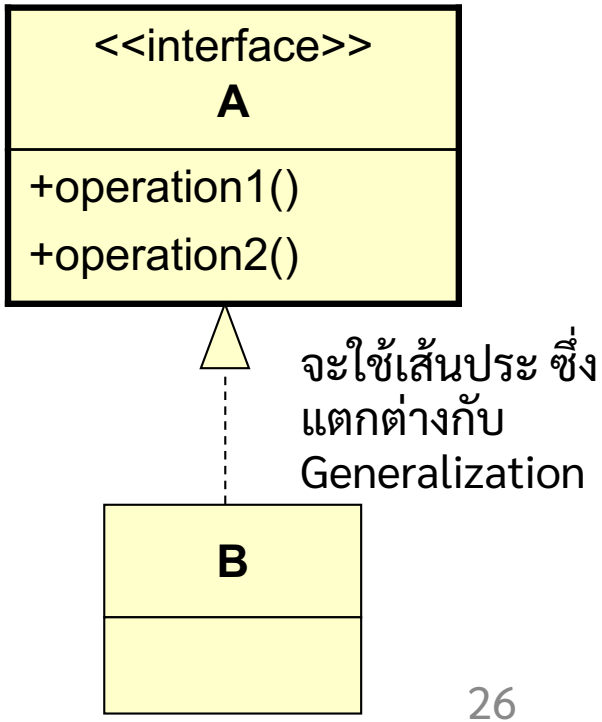
บ่งบอกว่า คลาสหนึ่งถูกสืบทอดพฤติกรรมการทำงานมาจากคลาสอื่นในรูปของ interface.

UML ใช้สัญลักษณ์ในการนำเสนอ interface ได้เป็น 2 รูปแบบคือ

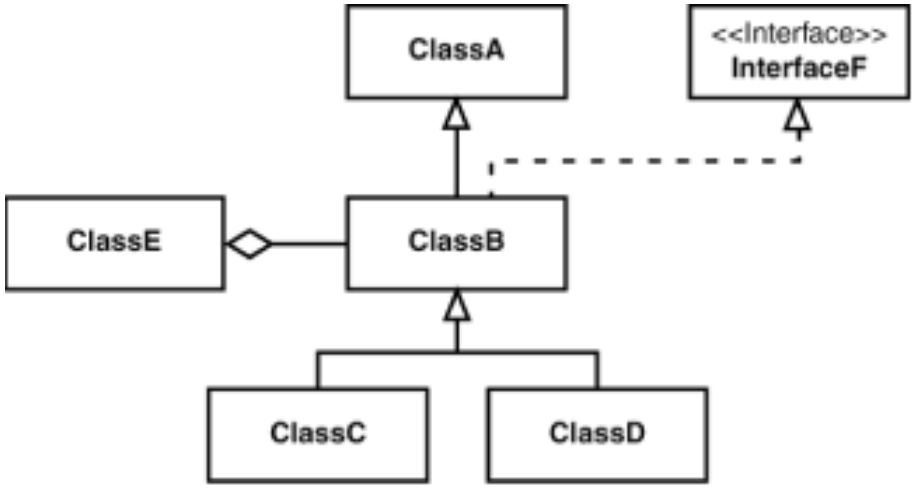
- สัญลักษณ์ "lollipop" ที่ใช้ร่วมกับคลาส หรือ subsystems อื่น ๆ
- สัญลักษณ์ของคลาสที่ใช้ stereotype โดยระบุ <<interface>>





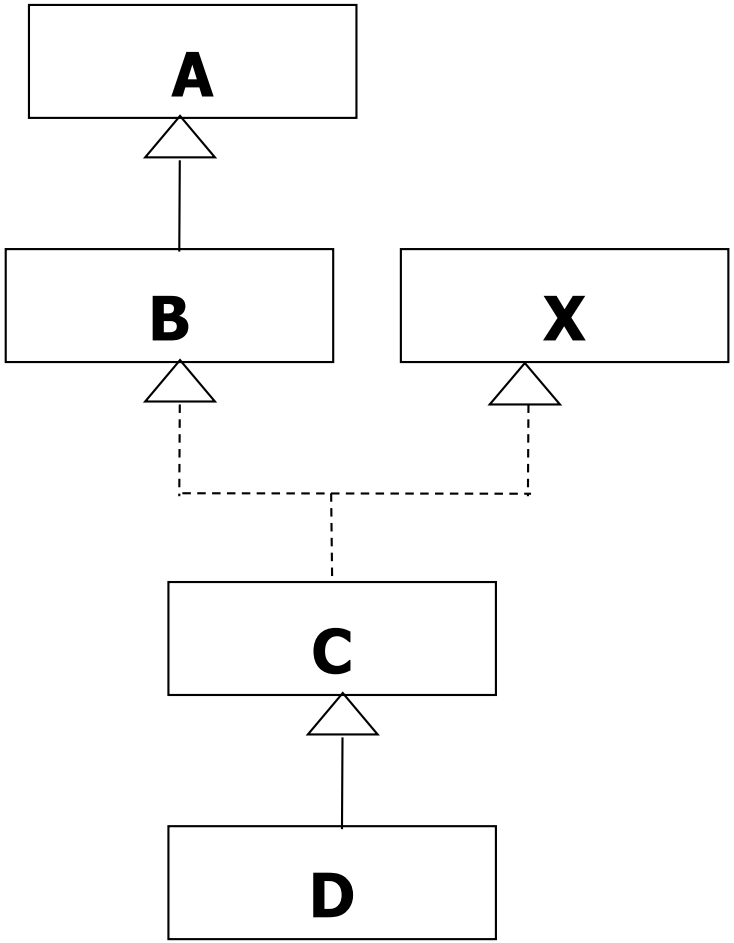


Class Diagram Review



- คลาส A เป็น **Superclass** ของคลาส B
- คลาส B เป็น superclass ของคลาส C และคลาส D
- คลาส B เป็น **subclass** ของคลาส A
- B **implements** (realizes) มาจาก InterfaceF
- จากหลักการสืบทอดทั้งคลาส C และ D เป็น subclass ของคลาส A
- B **implements** (realizes) มาจาก InterfaceF
- คลาสB เป็น (**IS-A**) InterfaceF
- คลาส C เป็น (**IS-A**) คลาส B คลาส A และ InterfaceF
- คลาส E ประกอบไปด้วยอย่างน้อยหนึ่ง reference ไปยังคลาส B (aggregation)

Check Your Understanding



แบบนี้เป็นไปได้หรือไม่?

จงเขียน Class Diagram ตามโจทย์ดังต่อไปนี้

- A zoo consists of a set of cages.
- Every cage is the home of at least 2 animals.
- Cages are located besides each other.
- Every cage has at most one left neighbor and at most one right neighbor.
- Animals can be reptiles, insects, and mammals.
- Mammals are elephants, monkeys, and tigers.
- Monkeys eat bananas.
- Tigers prefer meat.

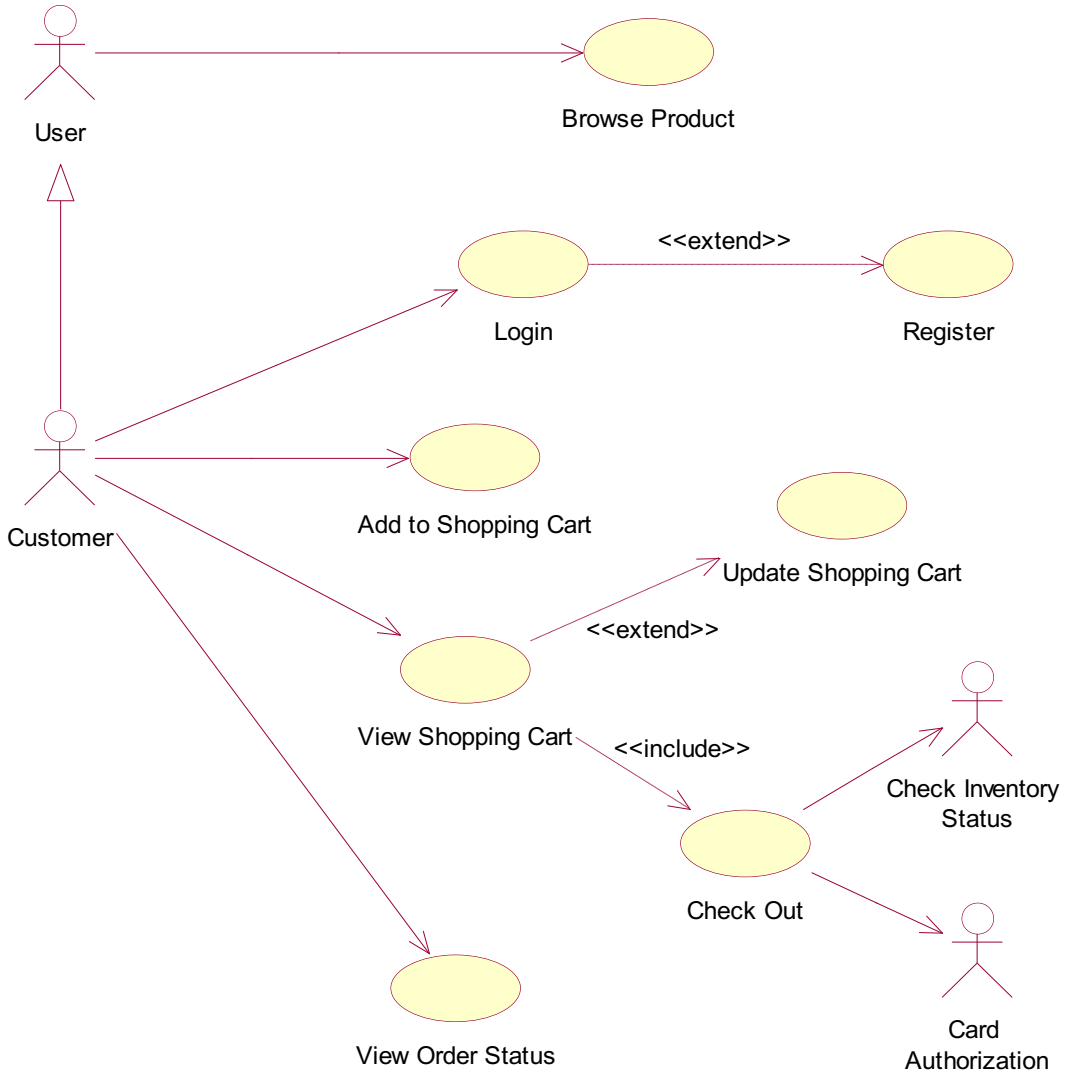
Classes from Sequence Diagrams

- ให้ดำเนินการ แยก Class ออกจากทุก Sequence Diagram
- หากมี Class ปรากฏซ้ำอยู่ในหลาย Sequence Diagram ให้ตรวจสอบ Message ที่ชี้มายังแต่ละ Class ที่มีชื่อซ้ำกัน ดังนี้
  - หากเป็น Message เดียวกัน ไม่ต้องเพิ่ม Method ดังกล่าวใน Class Diagram
  - หากเป็น Message ที่ต่างกัน ให้เพิ่ม Method ดังกล่าวใน Class Diagram
  - (optional) หลังจากดำเนินการครบทุก Sequence Diagram ให้แยกประเภทของ Class ออกเป็น 3 กลุ่ม คือ
    - Model Class
    - View Class
    - Controller Class
- ให้นำ Model Class ไปออกแบบ Database Schema ต่อไป (หากระบบจัดเก็บข้อมูลในฐานข้อมูล)

Steps to Create a UML Class Diagram

1. ระบุว่าระบบควรจะมีคลาสอะไรบ้าง (จาก Sequence Diagrams)
2. ระบุคุณลักษณะของแต่ละคลาส
3. ระบุ method ของแต่ละคลาส (จาก message ใน SD)
4. ค้นหาคลาสที่มีคุณลักษณะหรือ method ที่คล้ายๆกัน
5. ระบุความสัมพันธ์ระหว่างคลาส (association vs generalization) และ ปริมาณความสัมพันธ์
6. จากความสัมพันธ์ที่หาได้ คลาสบางคลาส***อาจจะ***ไม่จำเป็นต้องมี ให้ลบออก โดยเฉพาะพวกที่ความสัมพันธ์เป็นแบบ one-to-one
7. ทำซ้ำจากขั้นตอนที่ 1 จนได้ class diagram มีรายละเอียดที่แสดงถึงระบบที่จะพัฒนาได้ดี

Use Case Diagram



Use Case: Login

- ระบบแสดงหน้าจอลอกอินโดยอัตโนมัติเมื่อผู้ใช้เข้าสู่ระบบสั่งซื้อออนไลน์ โดยหน้าจอดังกล่าวจะประกอบด้วยฟิลด์**ชื่อผู้ใช้(Username)** และ**รหัสผ่าน (Password)** ระบบจะรับค่าชื่อผู้ใช้และรหัสผ่านเพื่อตรวจสอบกับข้อมูลผู้ใช้ที่อยู่ภายในฐานข้อมูลของระบบ ในกรณีที่ผู้ใช้ใส่ค่าชื่อผู้ใช้และรหัสผ่านที่ตรวจสอบแล้วไม่มีอยู่ในระบบ**ฐานข้อมูล(Database)** ระบบจะแสดงข้อความชื่อผู้ใช้และรหัสผ่านไม่ถูกต้อง (Invalid Password and/or User ID)
- ในกรณีที่ผู้ใช้ยังไม่มีชื่อผู้ใช้ รหัสผ่าน และข้อมูลที่จำเป็นในการสั่งซื้อ ระบบจะยอมให้ผู้ใช้สามารถลงทะเบียนโดยผ่านฟังก์ชัน**ลงทะเบียน (Register)** ที่อยู่ด้านล่างของหน้าจอลอกอิน หลังจากขั้นตอนการลอกอินแล้วระบบจะยอมให้ผู้ใช้เข้าถึงการทำงานของ**การสั่งซื้อ** (Place Order) **การยกเลิกการสั่งซื้อ** (Cancel Order) และ**การแก้ไขรายการสั่งซื้อ** (Update Order)

Use Case: Browse Product

- Browse Product ใช้สำหรับแสดงรายละเอียดสินค้าตามประเภทที่ผู้ใช้เลือก โดยเริ่มต้นจากระบบแสดง**ประเภท (Category)** ของสินค้าทั้งหมด เมื่อ**ผู้ใช้ (User)** หรือ**ลูกค้า (Customer)** เลือกประเภทสินค้าที่ต้องการ รายละเอียดของรายการ**สินค้า (Product)** ทั้งหมดจะถูกนำเสนอบนหน้าจอ โดยประกอบไปด้วย **รหัสสินค้า (productId)** **ชื่อสินค้า (productName)** **ราคาต่อหน่วย (pricePerUnit)** เป็นต้น

Use Case: Add Shopping Cart

- ขั้นตอนนี้เริ่มต้นเมื่อลูกค้าเลือกการสั่งซื้อโดยเลือก**รายการสินค้า** (Order Item) ที่ต้องการแล้ว ระบบจะแสดงรายละเอียด**สินค้า** (Product) ที่ประกอบด้วย **รหัสสินค้า** (ProductID) **ชื่อสินค้า** (ProductName) **ราคาต่อหน่วย** (Unit Price) โดยผู้ใช้กรอก**จำนวนหน่วย** (Unit) ที่ต้องการ จากนั้นระบบทำการคำนวณ**ราคารวมทั้งหมด** (Total Price) ทีละรายการ จนกว่าผู้ใช้จะเลือก**การจ่ายเงิน** (Payment) โดยการกรอกข้อมูล**บัตรเครดิต** (Credit Card) ที่ประกอบไปด้วย **หมายเลขบัตร** (Credit card Number) และ**วันหมดอายุ** (Expire Date) จากนั้นลูกค้าสามารถส่ง**รายการสั่งซื้อ** (Order) เข้าสู่ระบบ เพื่อให้ระบบทำการตรวจสอบบันทึก และส่งข้อมูลต่อไปยัง**ระบบบัญชี** (accounting system) เพื่อยืนยันการสั่งซื้อ ระบบจะคืนค่า**รหัสรายการสั่งซื้อ** (OrderID) กลับไปยังลูกค้า และสิ้นสุดการทำงานของยูสเคส

Noun Analysis

ลอกอิน (Login)	**ชื่อผู้ใช้(Username)**	**รหัสผ่าน(Password)**
ลูกค้า (Customer)	**ชื่อ** (Name)	**นามสกุล** (Lastname)
วันเดือนปีเกิด (DateOfBirth)	**อาชีพ** (Occupation)	**ที่อยู่** (Address)
บ้านเลขที่ (Number)	**ถนน** (Street)	**ตำบล** (Sub-District)
อำเภอ (District)	**จังหวัด** (Province)	**รหัสไปรษณีย์** (ZipCode)
สถานที่จัดส่งสินค้า (Delivery Address)	**ยืนยันการกรอกรหัสผ่าน** (Confirm Password)	**รหัสลูกค้า** (CustomerID)
รายการสั่งซื้อ (Order)	**วันที่สั่งซื้อ** (OrderDate)	**รายการสินค้า** (Order Item)
จำนวนหน่วย (Unit)	**สินค้า** (Product)	**รหัสสินค้า** (ProductID)
ชื่อสินค้า (ProductName)	**ราคาต่อหน่วย** (Unit Price)	**ราคารวมทั้งหมด** (Total Price)
การจ่ายเงิน (Payment)	**บัตรเครดิต** (Credit Card)	**หมายเลขบัตร** (Credit card Number)
วันหมดอายุ (Expire Date)	**สถานะบัตรเครดิต** (Status)	**ระบบบัญชี** (accounting system)
รหัสรายการสั่งซื้อ (OrderID)	**สถานะการสั่งซื้อ** (OrderStatus)	**จัดส่ง** (Deliverly)

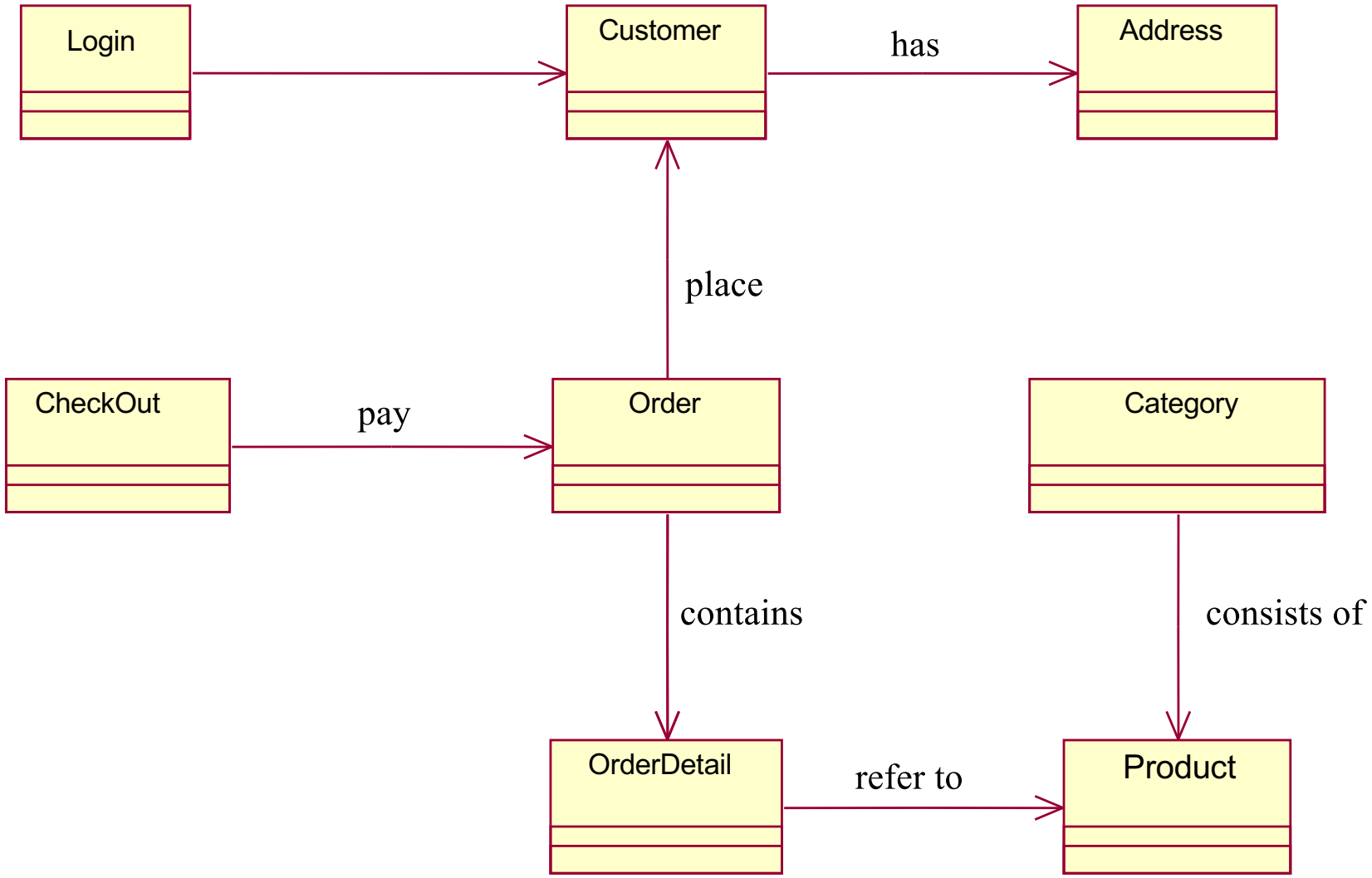
Class Analysis

- คลาสที่มีความหมายซ้ำกัน (Redundant Classes) เช่น คลาสผู้ใช้ (User) หรือ คลาสลูกค้า (Customer)
- คลาสที่ไม่ตรงประเด็น (Irrelevant Classes) ซึ่งจะหมายถึงคลาสที่ไม่มีอะไรเกี่ยวข้องโดยตรงกับระบบ เช่น ลูกค้าร้องเรียน (Customer complaint)
- คลาสที่มีความหมายคลุมเครือ (Vague Classes) เป็นคลาสที่มีความหมายไม่ชัดเจนต่อการพัฒนาระบบ เช่น system และ software เป็นต้น
- คำนามที่อยู่ในรูปของแอททริบิวต์ เป็นคำนามที่ไม่สามารถเป็นคลาสได้โดยตรง เนื่องจากมีลักษณะเป็นรายละเอียดประกอบในคลาสอื่น ๆ เป็นหลัก เช่น ชื่อลูกค้า (Customer’s name)
- คำนามที่มีลักษณะเป็นการทำงานไม่สามารถกำหนดเป็นคลาสได้โดยตรงเช่น การค้นหาสินค้า (Product Search)
- คลาสที่เป็นบทบาทหรือหน้าที่ (Roles) เป็นคลาสคู่แข่งที่มีลักษณะเป็นความสัมพันธ์มากกว่าที่จะเป็นคลาส

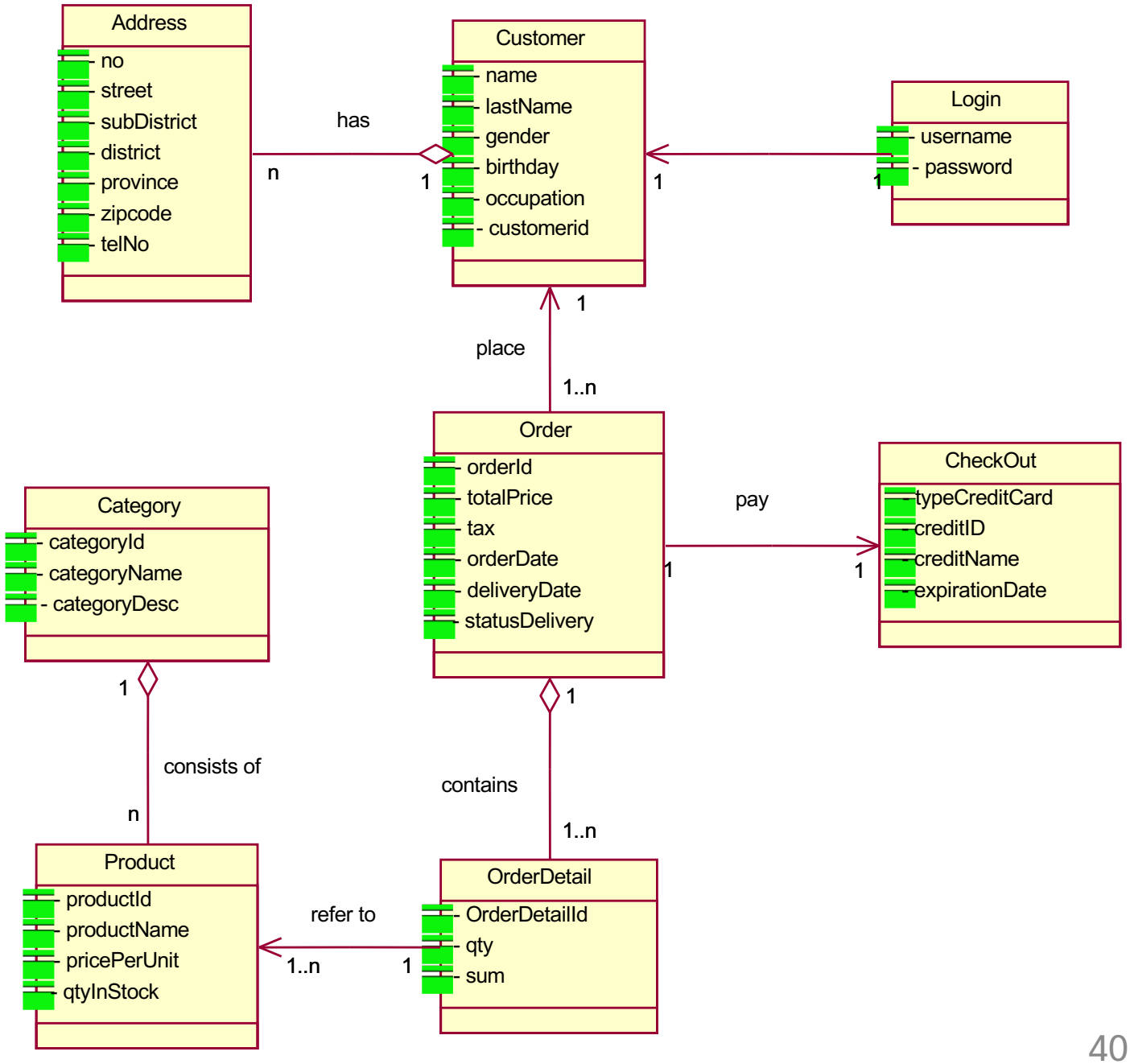
Class (with Attribute) Extraction

รายการคลาสคู่แข่ง	**คลาส**	**เหตุผล**
ลอกอิน (Login)	/	- กำหนดเป็นคลาสสำหรับการลอกอิน
ชื่อผู้ใช้(Username)	-	- กำหนดเป็นแอททริบิวต์ของคลาสลอกอิน
รหัสผ่าน(Password)	-	- กำหนดเป็นแอททริบิวต์ของคลาสลอกอิน
ลูกค้า (Customer)	/	-กำหนดเป็นคลาสลูกค้า
ชื่อ (Name)	-	- กำหนดเป็นแอททริบิวต์ของคลาสลูกค้า
นามสกุล (Lastname)	-	- กำหนดเป็นแอททริบิวต์ของคลาสลูกค้า
วันเดือนปีเกิด (Date of Birth)	-	- กำหนดเป็นแอททริบิวต์ของคลาสลูกค้า
อาชีพ (Occupation)	-	- กำหนดเป็นแอททริบิวต์ของคลาสลูกค้า

Conceptual Level



With Details



Summary

- Learned what a UML Class Diagram is.
- Went over different types of relationship between classes.
  - Association
    - Aggregation
    - Composition
  - Generalization
  - Dependency
- Steps to create a UML Class Diagram.

References:

- Lethbridge, Timothy Christian, and Robert Laganiere. *Object-oriented software engineering*. Vol. 11. New York: McGraw-Hill, 2005. (Chapter 5)

- Fowler, Martin. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004. (Chapter 3)

- รศ. รังสิต ศิริรังษี, อ. สายัณห์ อุ่นนันกาศ, *Object Oriented Development with UML: Class Diagram*, 2006

- Icons from https://www.flaticon.com