



# ***CPE3243 วิศวกรรมซอฟต์แวร์ (Software Engineering )***

**Piyavit Laung-Aram**  
**Major of Computer Engineering**  
**Faculty of Engineering**  
**Ramkhamhaeng University, Thailand**

# **เรียนรู้ UML 2**

## **(UML Tutorial 2)**

# **UML Diagram ส่วนที่ 2**

## **(UML Diagram Part 2)**



*Unified Modeling Language*



## UML (Unified Modeling Language)

---

UML เป็นทุล(tool) หรือเครื่องมือช่วยตัวหนึ่งที่ใช้ในการพัฒนาซอฟต์แวร์ ในยุคปัจจุบัน และเป็นภาษากลางที่ใช้สื่อสารระหว่างลูกค้า นักวิเคราะห์ระบบ โปรแกรมเมอร์ ลักษณะของ UML จะคล้ายกับพิมพ์เขียว (blueprint) หรือแบบจำลองที่ถูกสร้างขึ้นเพื่อให้ลูกค้า นักวิเคราะห์ระบบ และโปรแกรมเมอร์ ได้ใช้ตกลงกันในรายละเอียดของซอฟต์แวร์ที่จะพัฒนาขึ้น เพื่อให้ซอฟต์แวร์ที่พัฒนาขึ้นมาเป็นไปตามความต้องการของลูกค้าได้อย่างครบถ้วน และมีประสิทธิภาพสูงที่สุด

ลูกค้า  
Customer



นักวิเคราะห์ระบบ  
System Analyst



นักพัฒนา  
Developer



# UML Use Case Diagram



# UML Use Case Diagram

---

Use Case Diagram ใช้เพื่อแสดงพฤติกรรมแบบไดนามิกของระบบ มันสรุปการทำงานของระบบโดยรวมกรณีการใช้งาน ผู้ดำเนินการ และความสัมพันธ์ของพวกเขา โดยจำลองงาน บริการ และฟังก์ชันที่ระบบ/ระบบย่อยของแอปพลิเคชันต้องการ มันแสดงให้เห็นการทำงานระดับสูงของระบบและยังบอกวิธีที่ผู้ใช้จัดการกับระบบ





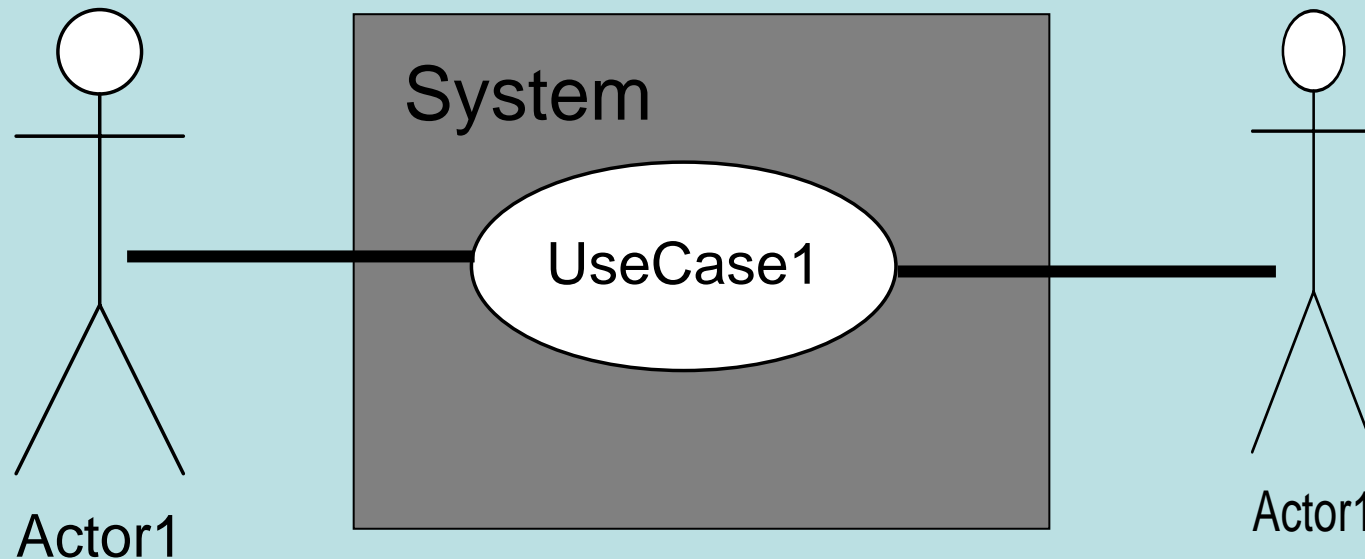
## วัตถุประสงค์ของ Use Case Diagram

วัตถุประสงค์หลักของ Use Case Diagrams คือการแสดงภาพไดนามิกของระบบ มันรวบรวมความต้องการของระบบซึ่งรวมถึงอิทธิพลทั้งภายในและภายนอก โดยจะเรียกบุคคล กรณีใช้งาน และหลายสิ่งหลายอย่างที่เราเรียกใช้ตัวแสดงและองค์ประกอบที่รับผิดชอบในการดำเนินการตามแผนภาพกรณีการใช้งาน มันแสดงให้เห็นว่าเอนทิตีจากสภาพแวดล้อมภายนอกสามารถโต้ตอบกับส่วนหนึ่งของระบบได้อย่างไร

วัตถุประสงค์ของแผนภาพกรณีการใช้งานที่ระบุด้านล่าง:


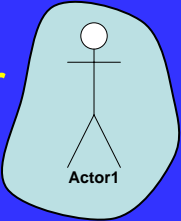
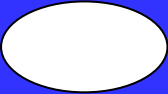



- รวบรวมความต้องการของระบบ
- แสดงให้เห็นมุมมองภายนอกของระบบ
- ตระหนักถึงปัจจัยภายในและภายนอกที่มีอิทธิพลต่อระบบ
- แสดงถึงปฏิสัมพันธ์ระหว่างผู้เกี่ยวข้องกับระบบ

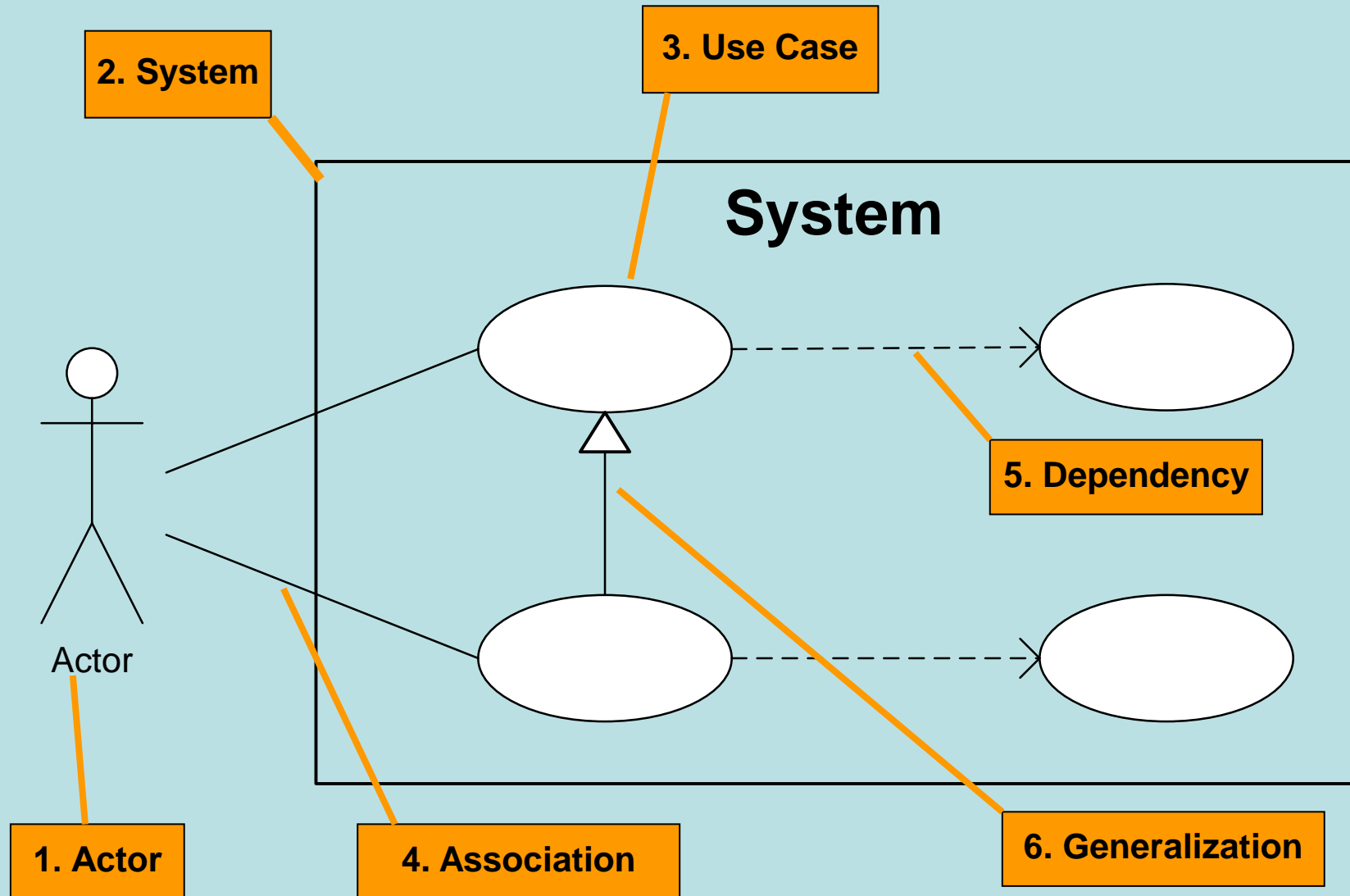
Use case diagram เป็น diagram ที่มีความสำคัญมาก เนื่องจาก use case diagram จะช่วยนักวิเคราะห์ระบบในการทำความเข้าใจ และมองภาพรวมของระบบงานที่กำลังดำเนินการศึกษาอยู่ หน้าที่หลักของ use case diagram คือการแสดงความติดต่อกันระหว่างผู้ใช้ กับระบบที่สร้างขึ้น



# Use Case Diagram

ประกอบด้วย 6 สัญลักษณ์ที่ใช้แสดงโครงสร้างของระบบงานที่เราจะสร้างขึ้น

1. **System**  แสดงขอบเขตของระบบและความสัมพันธ์ในการทำงานที่มีต่อผู้ใช้
2. **Actor**  แสดงผู้ใช้งาน ระบบ อุปกรณ์ต่างๆ ที่มาเกี่ยวข้องกับระบบของท่าน
3. **Use Case**  แสดงคุณลักษณะหรือกระบวนการในระบบของท่าน ซึ่งถ้าขาดส่วนส่วนนี้แล้วจะทำให้ระบบของท่านไม่อาจทำงานได้หรือไม่สมบูรณ์
4. **Association**  แสดงการโต้ตอบระหว่าง actor และ use case ต่างๆ
5. **Dependency**  แสดงความสัมพันธ์ระหว่าง use case 2 ตัว
6. **Generalization**  แสดงความสัมพันธ์ระหว่าง use case 2 ตัว หรือ actor 2 ตัว  
ที่ use case หรือ actor ตัวนั้นสืบทอดคุณลักษณะมาจาก use case หรือ actor อีกตัวหนึ่ง



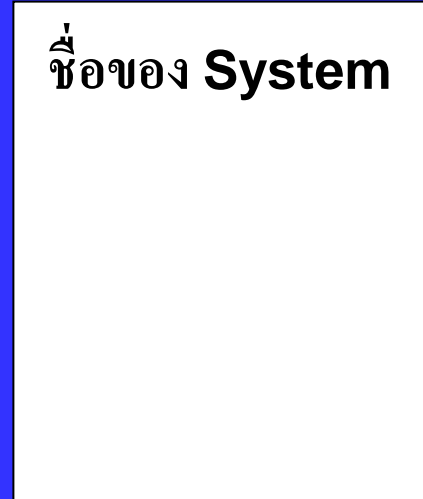
# 1. System

ชื่อของ System



หรือ

ชื่อของ System



## 2. Actor

**Actor** หรือผู้ที่เกี่ยวข้องกับระบบไม่ได้จำกัดอยู่ที่คนที่ใช้เกี่ยวข้องกับระบบเท่านั้น แต่ยังรวมถึงระบบอื่น ๆ และอุปกรณ์ต่าง ๆ ที่เกี่ยวข้องกับระบบด้วย



Actor

**<<actor>>**

**HR System**

**<<actor>>**

**Satellite Feed**

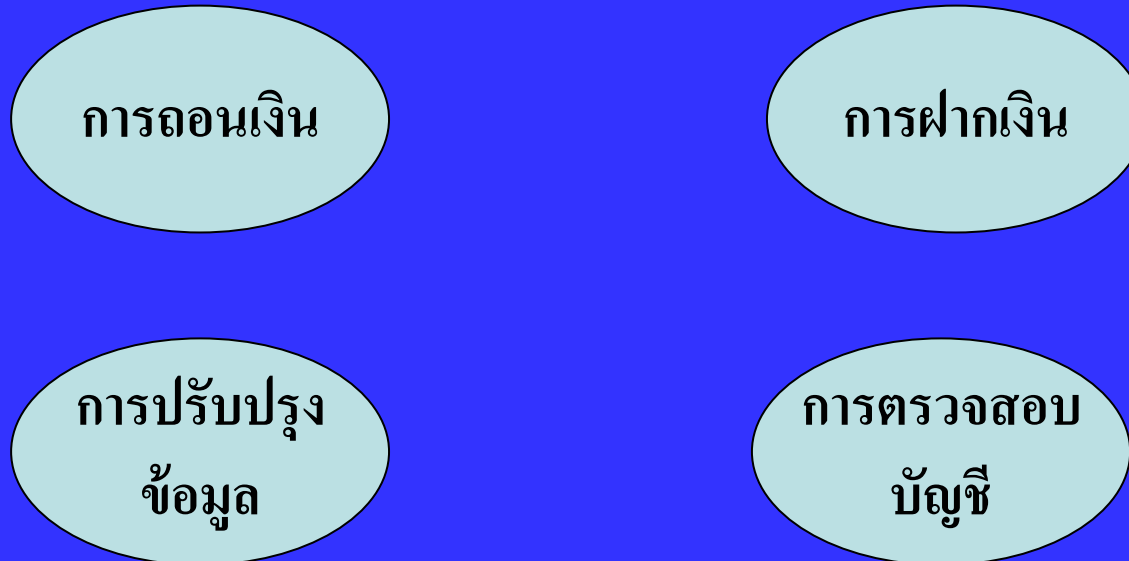
ตัวอย่าง **actor**  
ที่เป็นคน

ตัวอย่าง **actor**  
ที่เป็นระบบอื่น

ตัวอย่าง **actor**  
ที่เป็นอุปกรณ์

# 3. Use Case

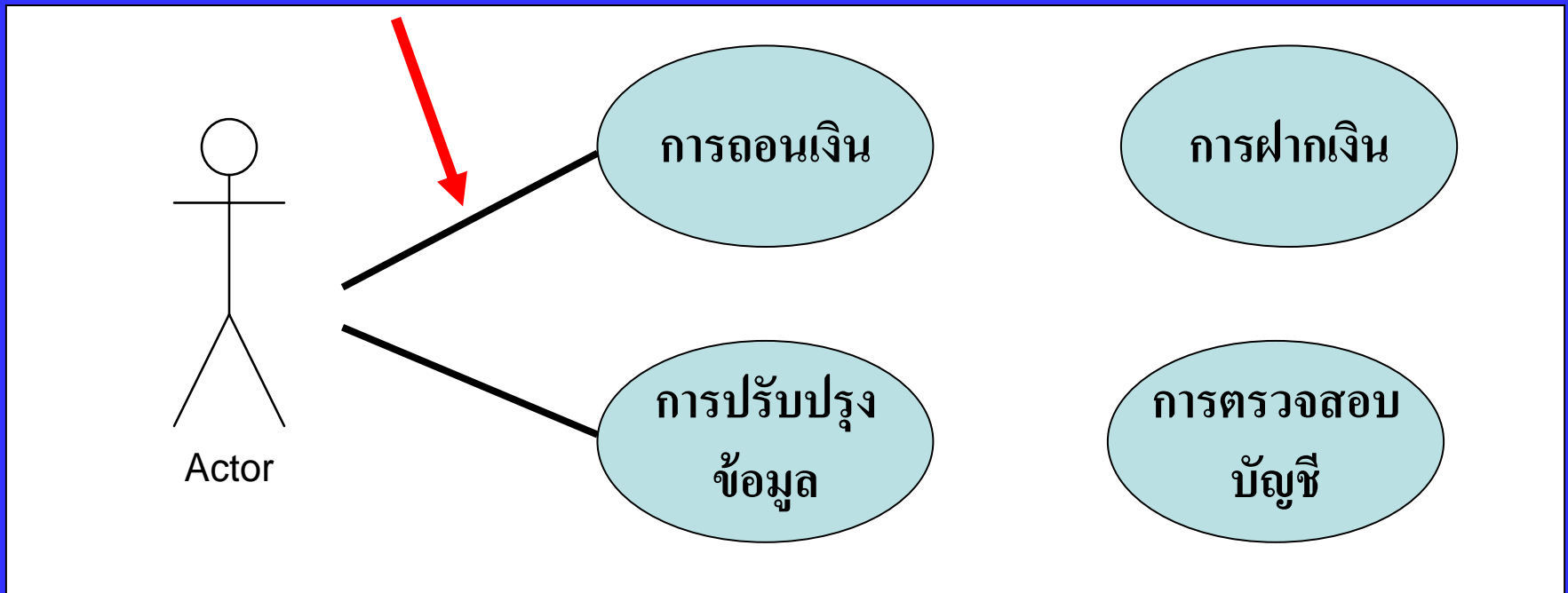
ใช้ระบุการทำงานต่าง ๆ ของระบบชื่อของ Use Case จะเป็นคำกริยาแสดงถึงสิ่งที่เกิดขึ้นกับ Use Case นั้น ๆ



## 4. relationship

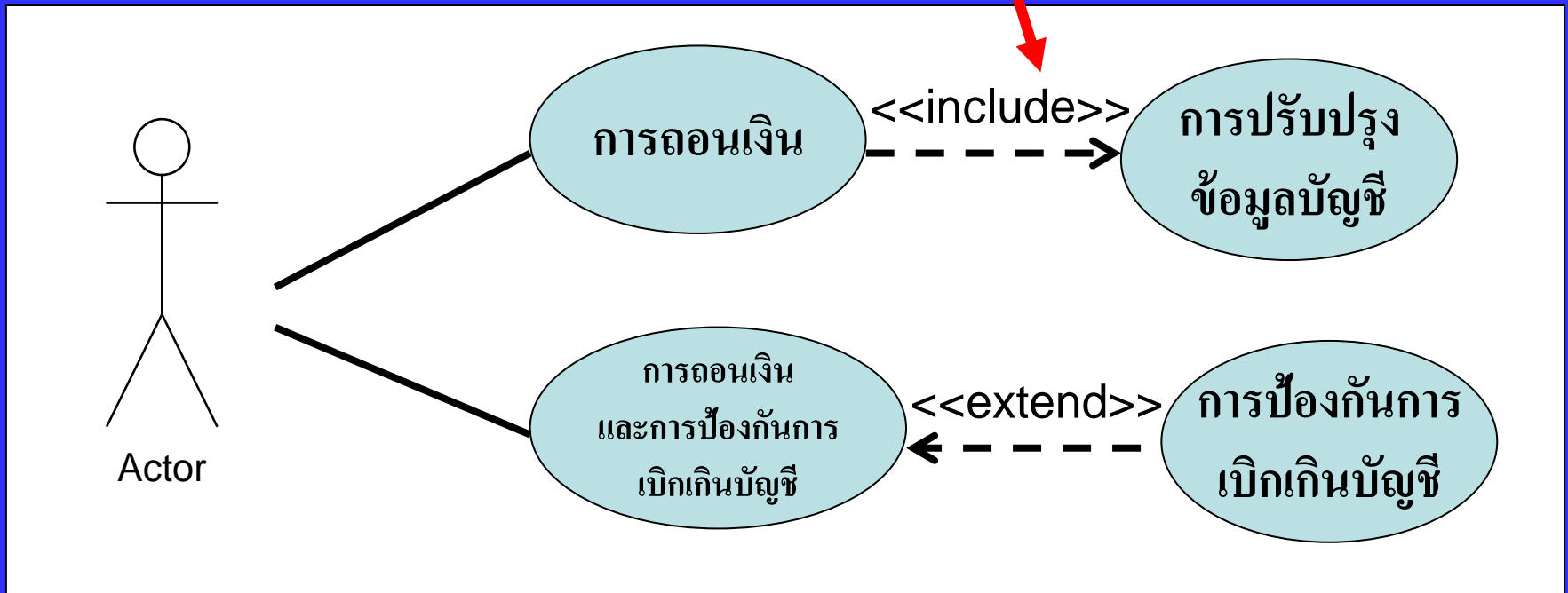
ใช้ระบุความสัมพันธ์ขององค์ประกอบต่าง ๆ ภายใน Use Case Diagram ประกอบด้วย

- Association notation ใช้แสดงความสัมพันธ์ระหว่าง actor กับ Use Case

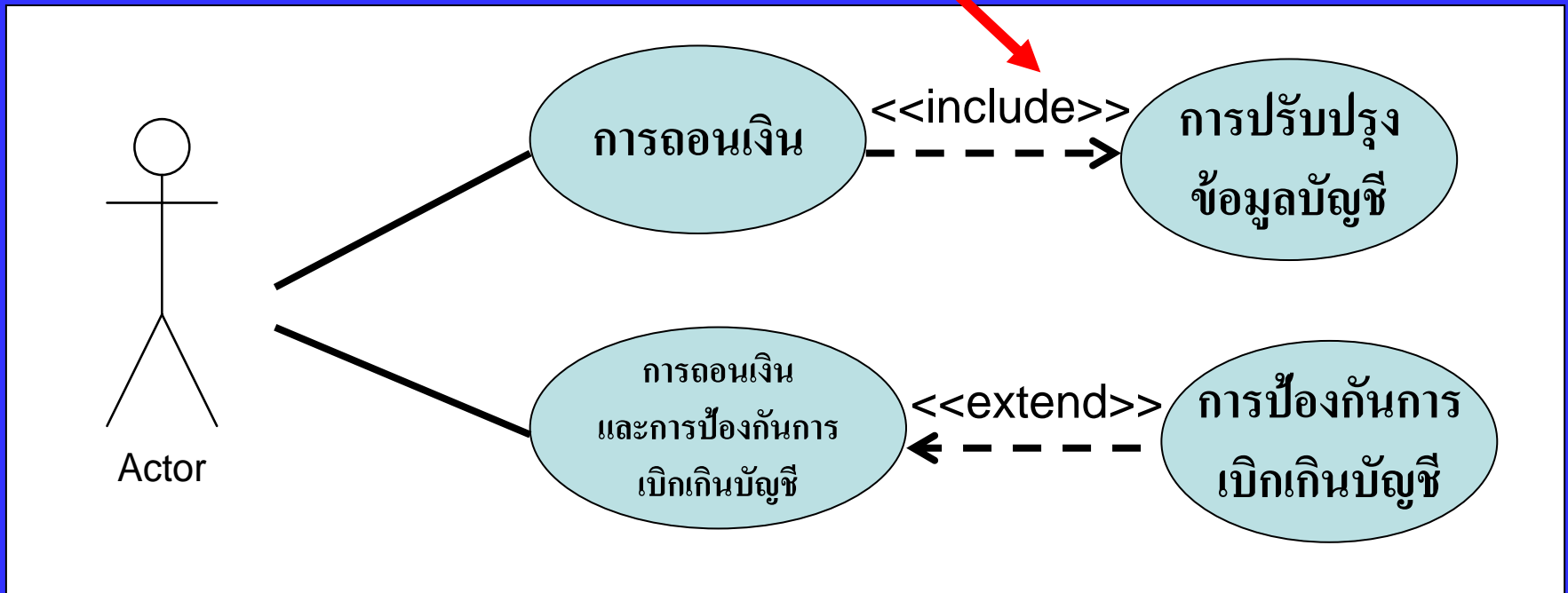




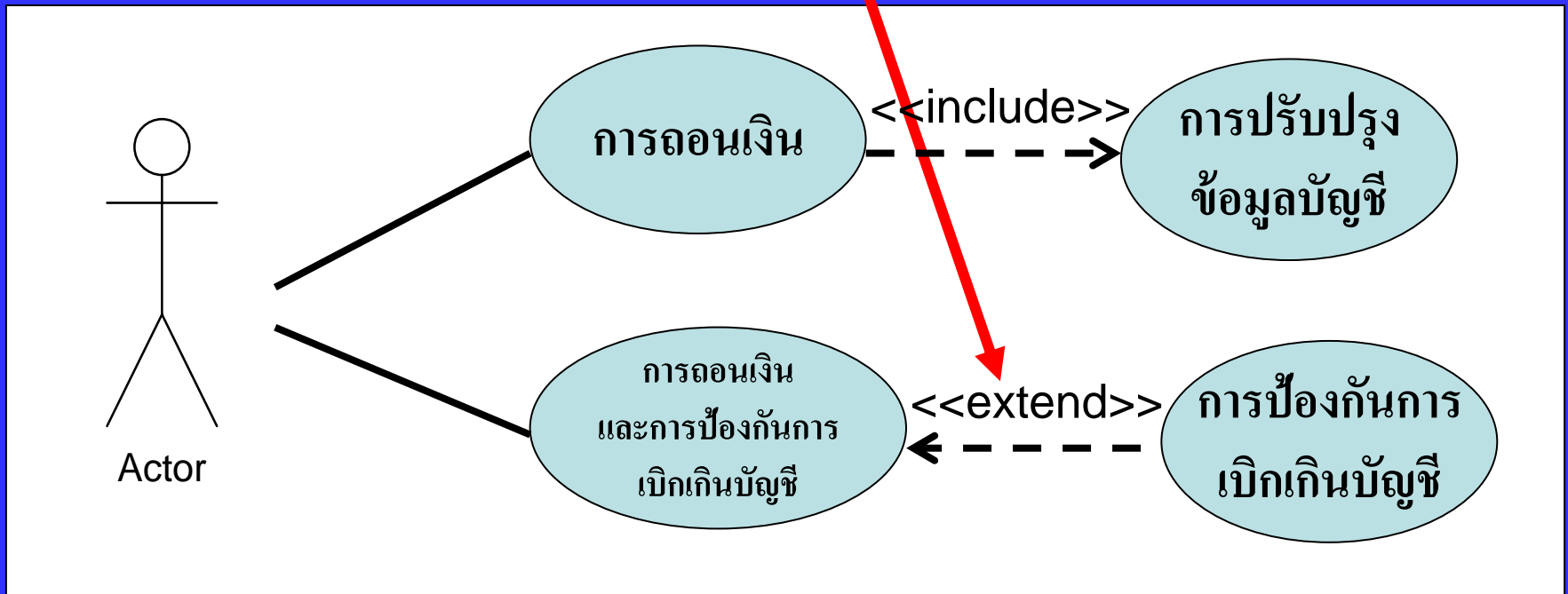
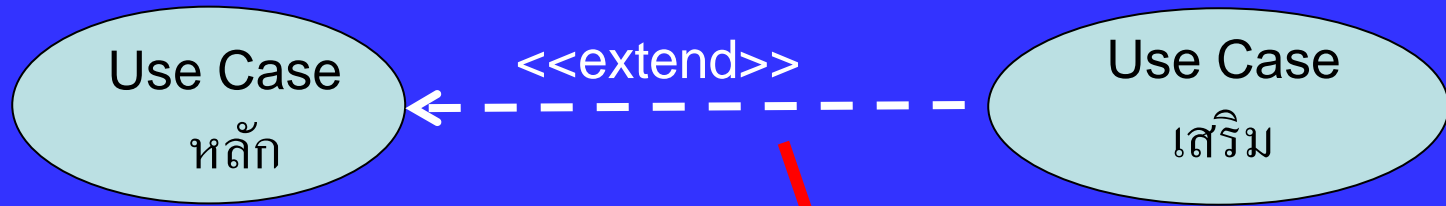
-Stereotype notation ใช้แสดงความสัมพันธ์ระหว่าง Use Case ซึ่งจะใช้  
สัญลักษณ์ <<.....>> และมีคำอธิบายใน <<...>>  
----->  
(French Quote)



-Stereotype notation <<include>> ใช้กรณีที่ use case หนึ่งต้องการเรียกใช้  
อีก use case หนึ่งให้มาช่วยทำงาน

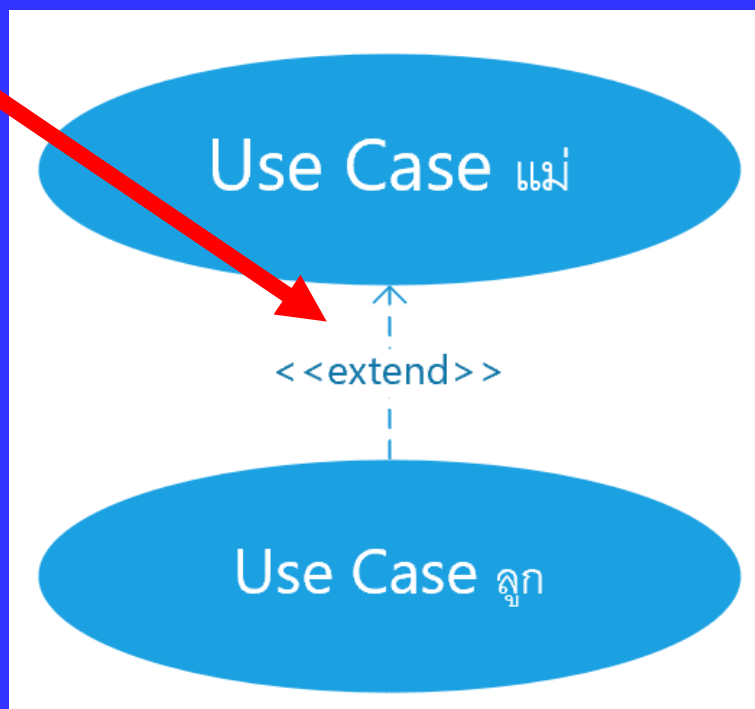


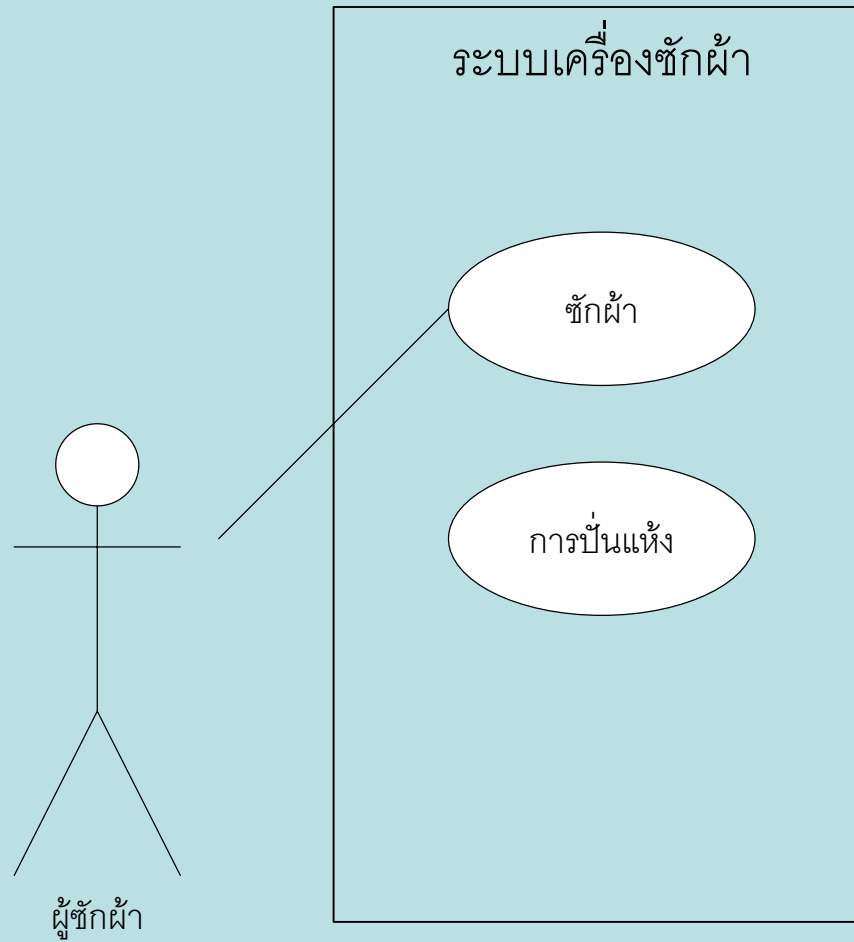
-Stereotype notation `<< extend >>` ใช้กรณีที่ use case หนึ่งต้องการขยายขีดความสามารถในการทำงานโดยการสร้างอีก use case เสริมขึ้นมาช่วยในการทำงาน

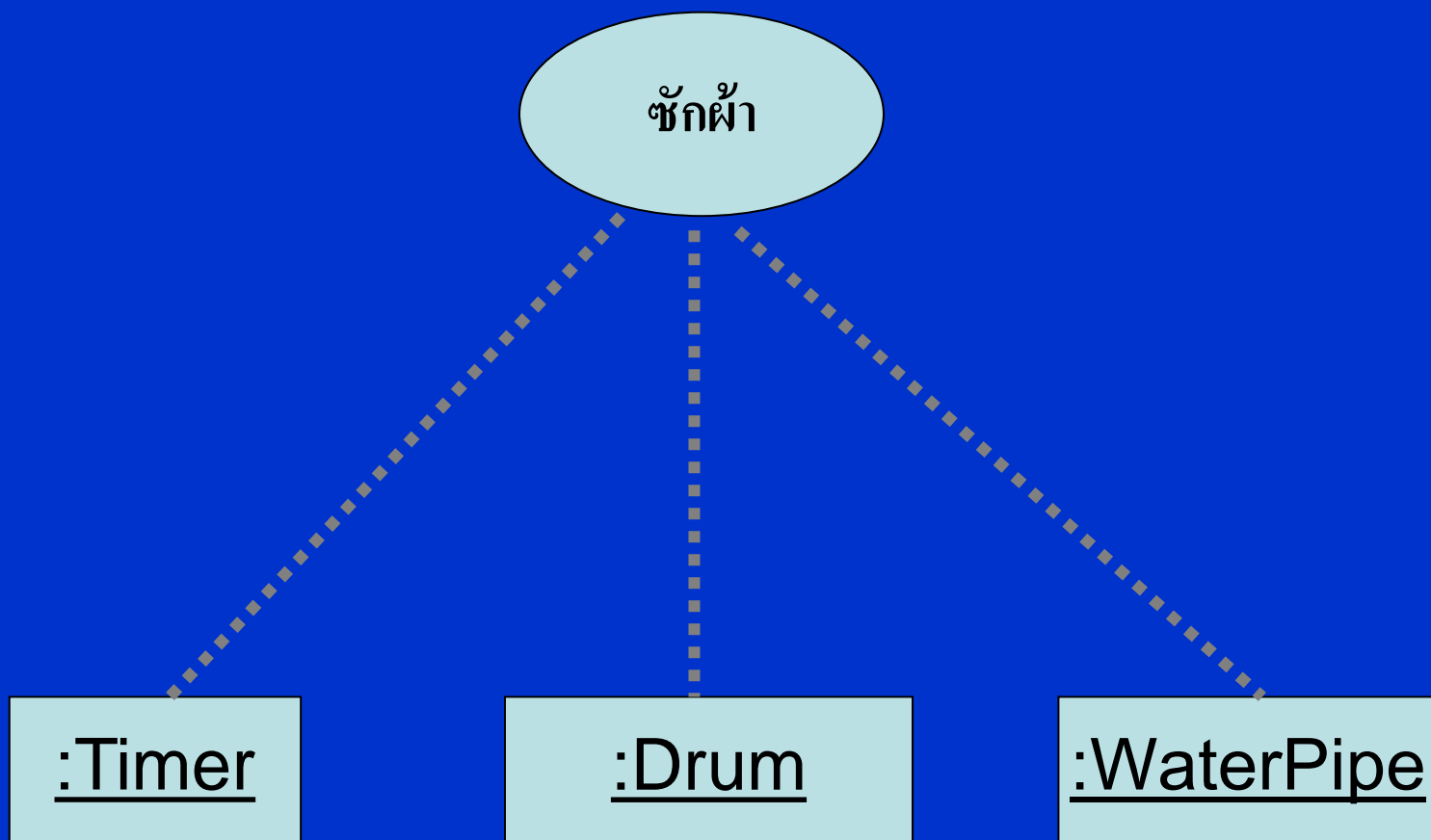


-Stereotype notation << extend >> ใช้กรณีที่ use case หนึ่งสืบทอดมาจาก  
อีก use case หนึ่ง







<<extend>>  
----->



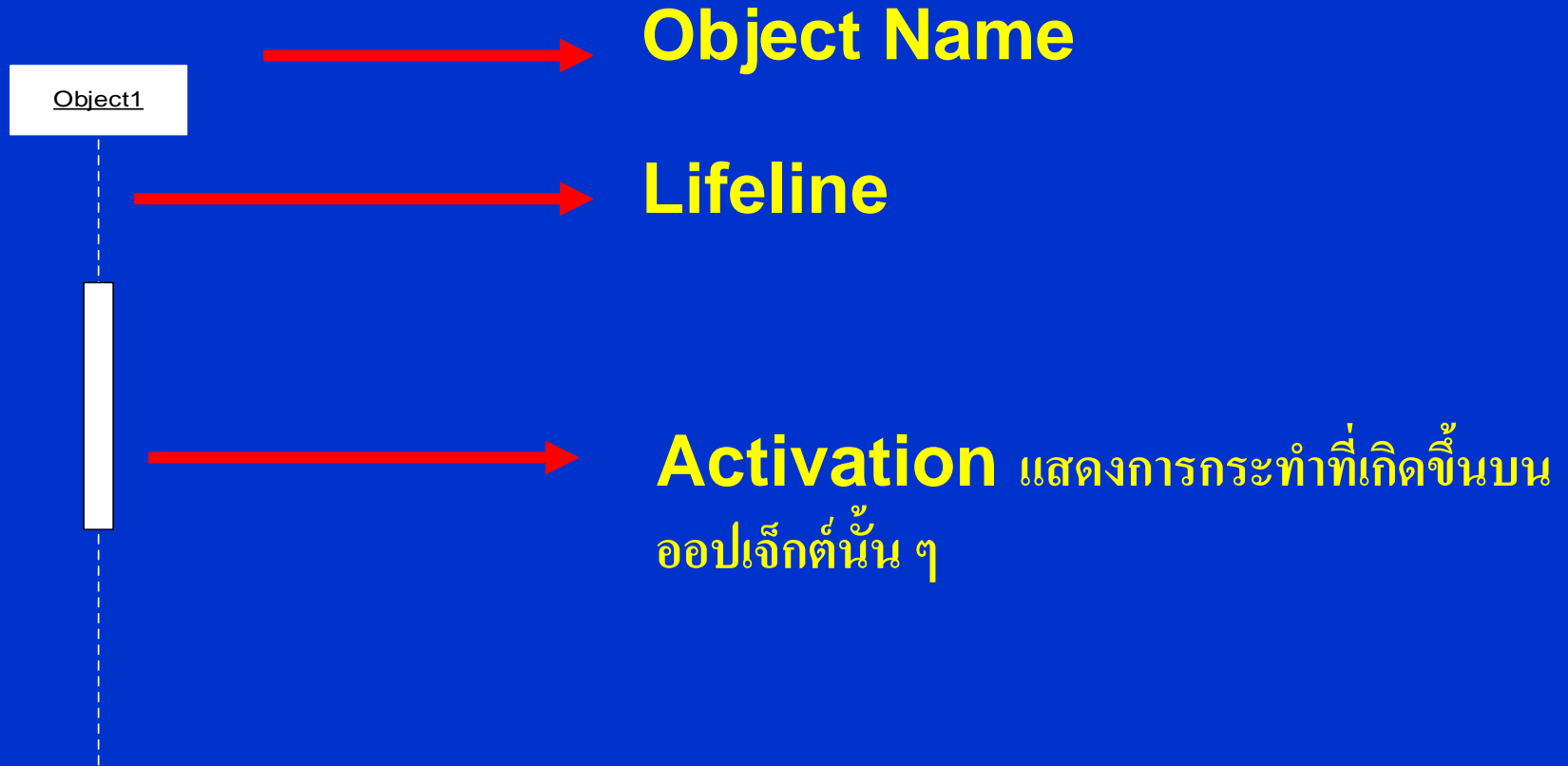




# Sequence Diagram

สัญลักษณ์	ชื่อ	ความหมาย
	Actor	ผู้ที่เกี่ยวข้องกับระบบ
	Object	ออบเจกต์ที่ทำงานอยู่ในระบบ
	Lifeline	เส้นชีวิตของออบเจกต์นั้น ๆ ที่คงอยู่ในระบบ
	Focus of Control / Activation	Activation แสดงการกระทำที่เกิดขึ้นบนออบเจกต์นั้น ๆ
	Message	ข้อความหรือการติดต่อที่เกิดขึ้นระหว่างออบเจกต์ต่าง ๆ ที่อยู่ในระบบ
	Callback / Self Delegation	การประมวลผลและคืนค่าที่ได้ภายในออบเจกต์เดียวกัน





Message1  
→



Asynchronous Message = การ  
ติดต่อแบบไม่ต้องรอคอยคำตอบ

Message2  
→



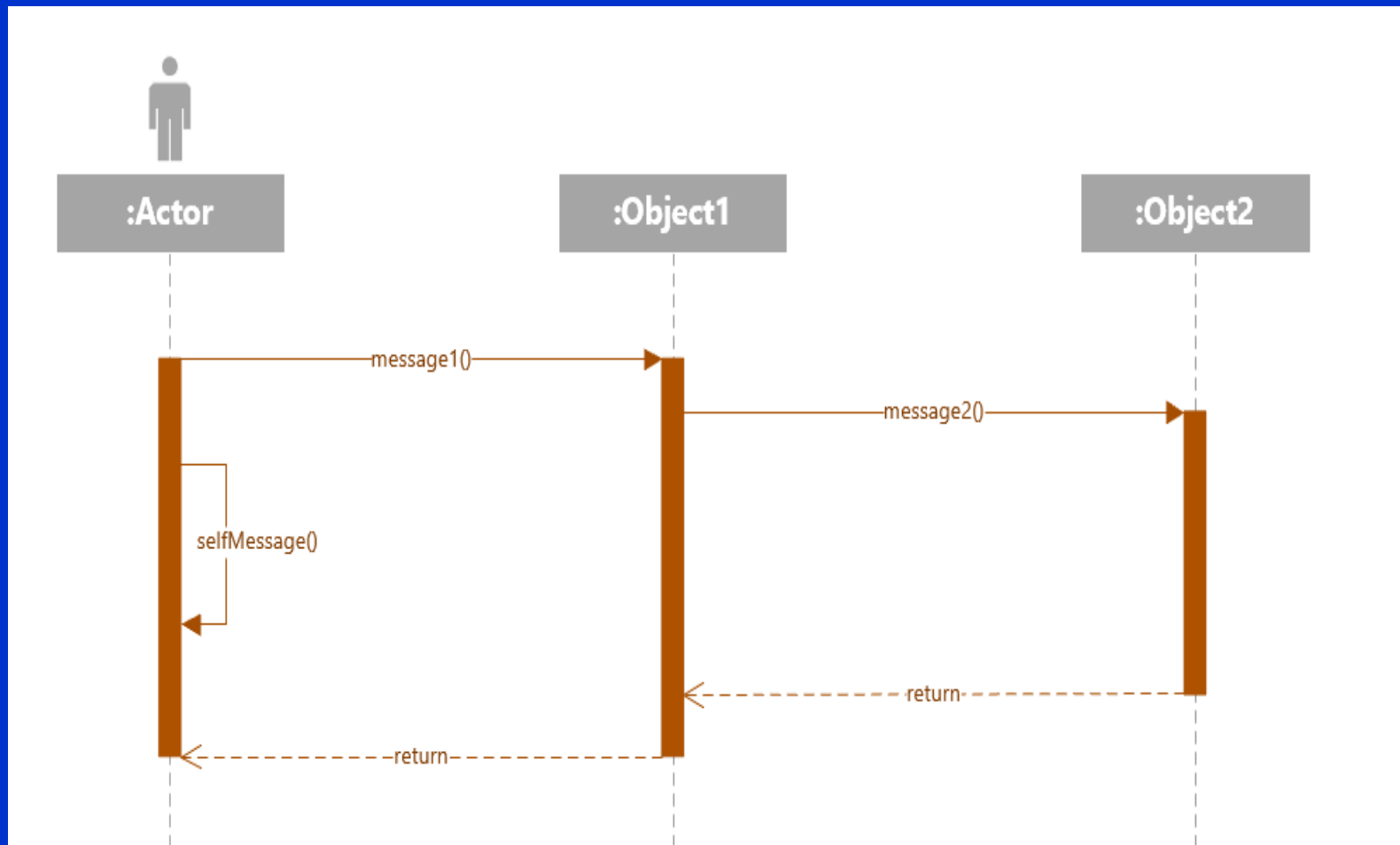
Synchronous Message = การ  
ติดต่อแบบรอคอยคำตอบ

Return Message  
←

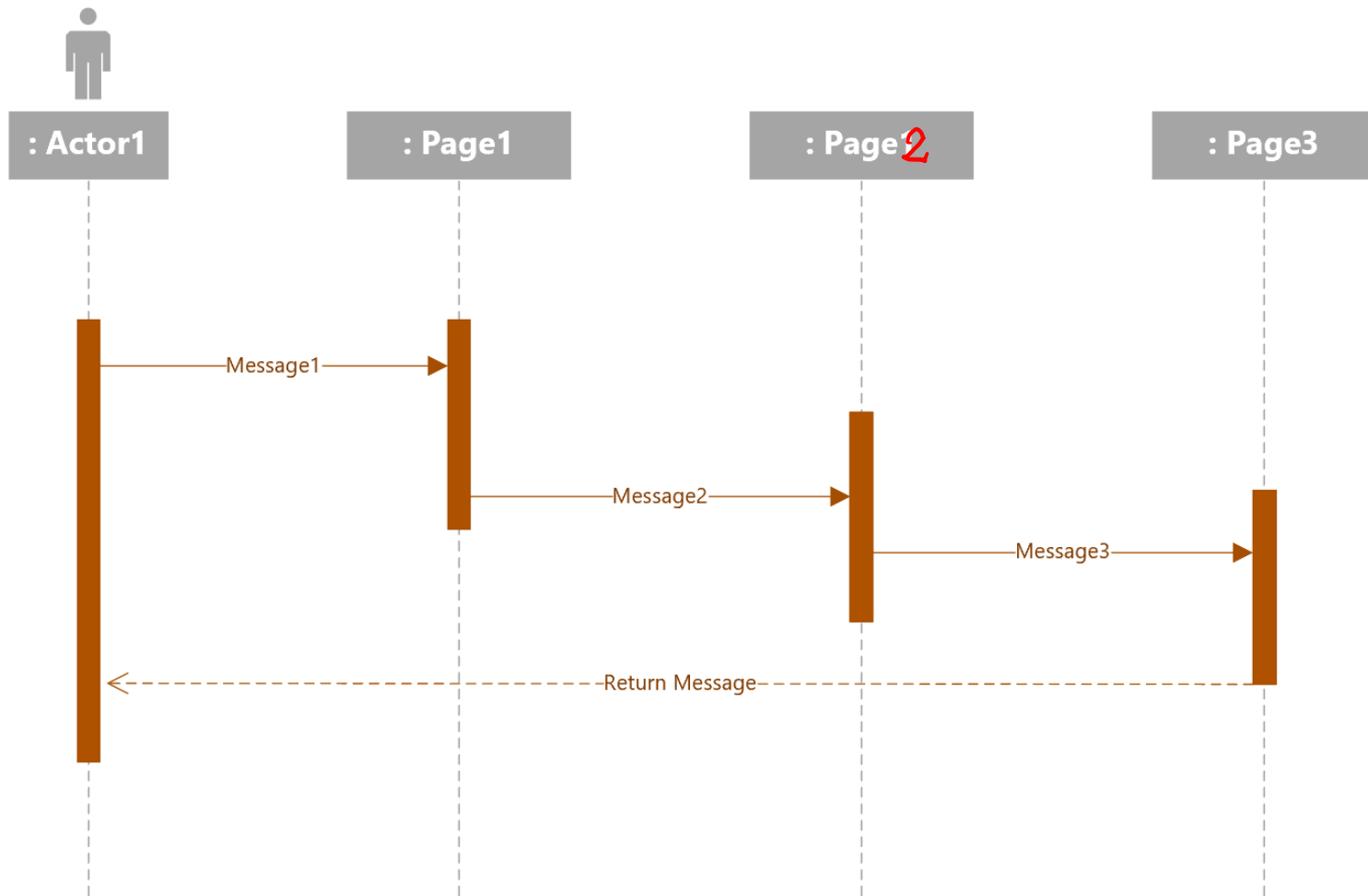


Return Message = การติดต่อ  
กลับ

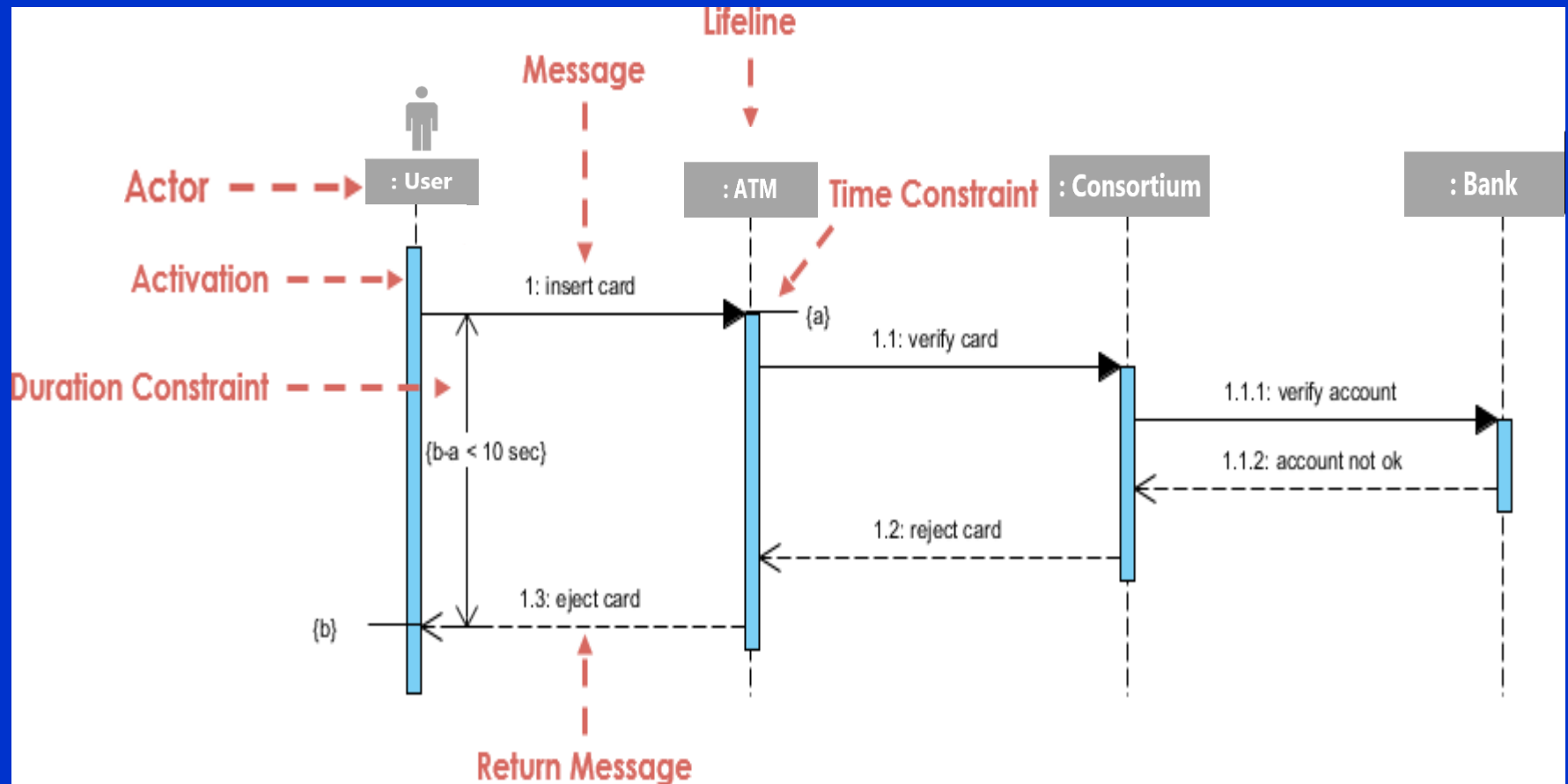
# ตัวอย่าง Sequence Diagram



# ตัวอย่าง Sequence Diagram



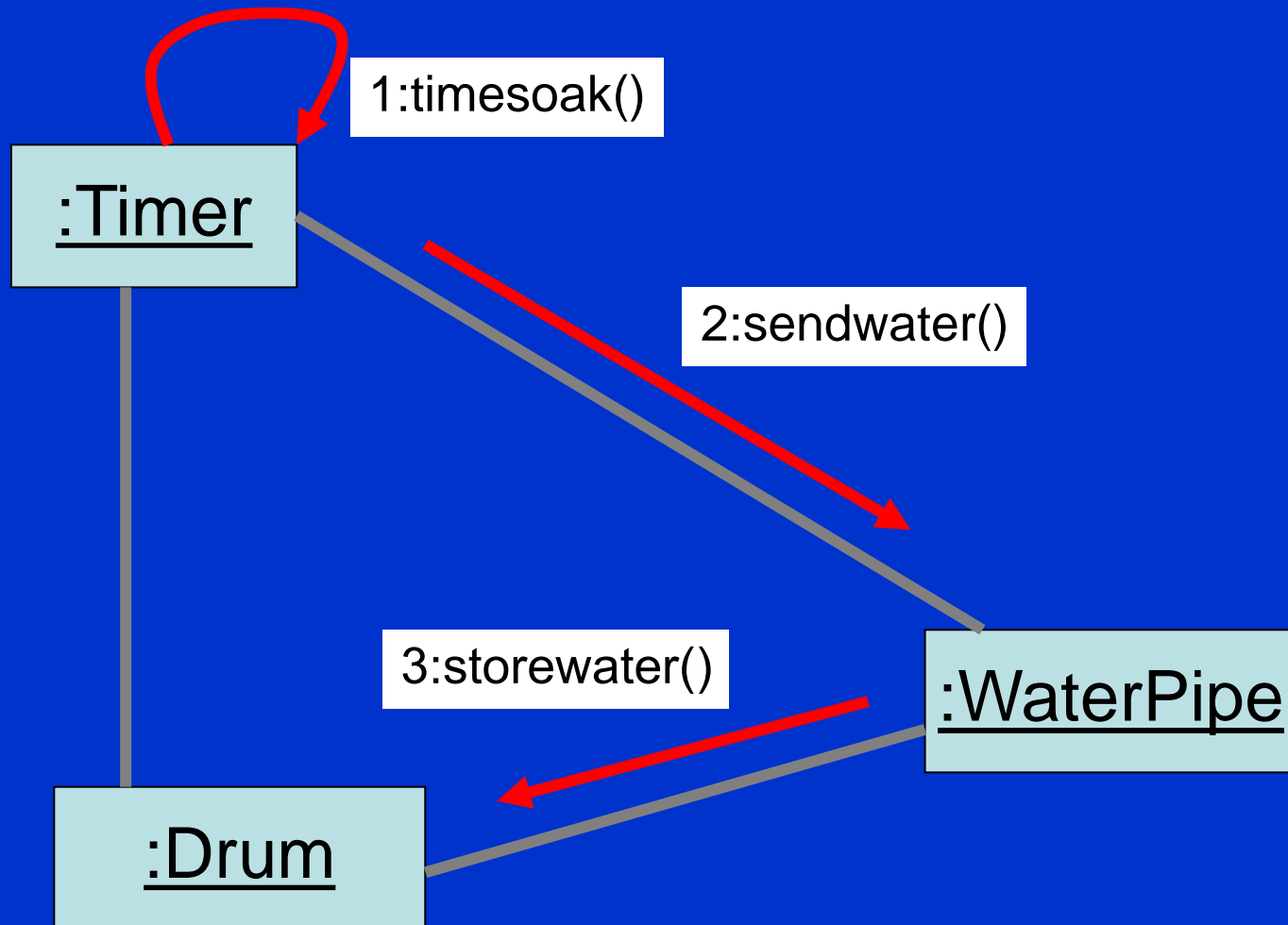
# ตัวอย่าง Sequence Diagram ระบบ ATM



# **Collaboration Diagram**

หรือ

# **Communication Diagram**



# Class Diagram





# *Class Diagram*

---

- Class diagram จัดเป็น Static Diagram คือ เป็น Diagram ที่เน้นโครงสร้างของอ็อบเจกต์ของคลาส (Class) ความสัมพันธ์ระหว่างอ็อบเจกต์ของคลาส (Relationship) แอททริบิวต์ (Attribute) และโอเปอเรชัน (Operation)
- ความสัมพันธ์ที่แสดงใน Class Diagram เป็นความสัมพันธ์เชิงแบบ Static ไม่ใช่ความสัมพันธ์แบบ Dynamic Class Diagram จึงไม่แสดงมองเห็นการเปลี่ยนแปลงต่างๆที่เกิดขึ้น แม้จะเกิดเหตุการณ์ใดๆ
- Class diagram เป็นมุมมองแบบ Logical View



- **Static relationship**

- นาย ก เป็นเจ้าของ บ้าน

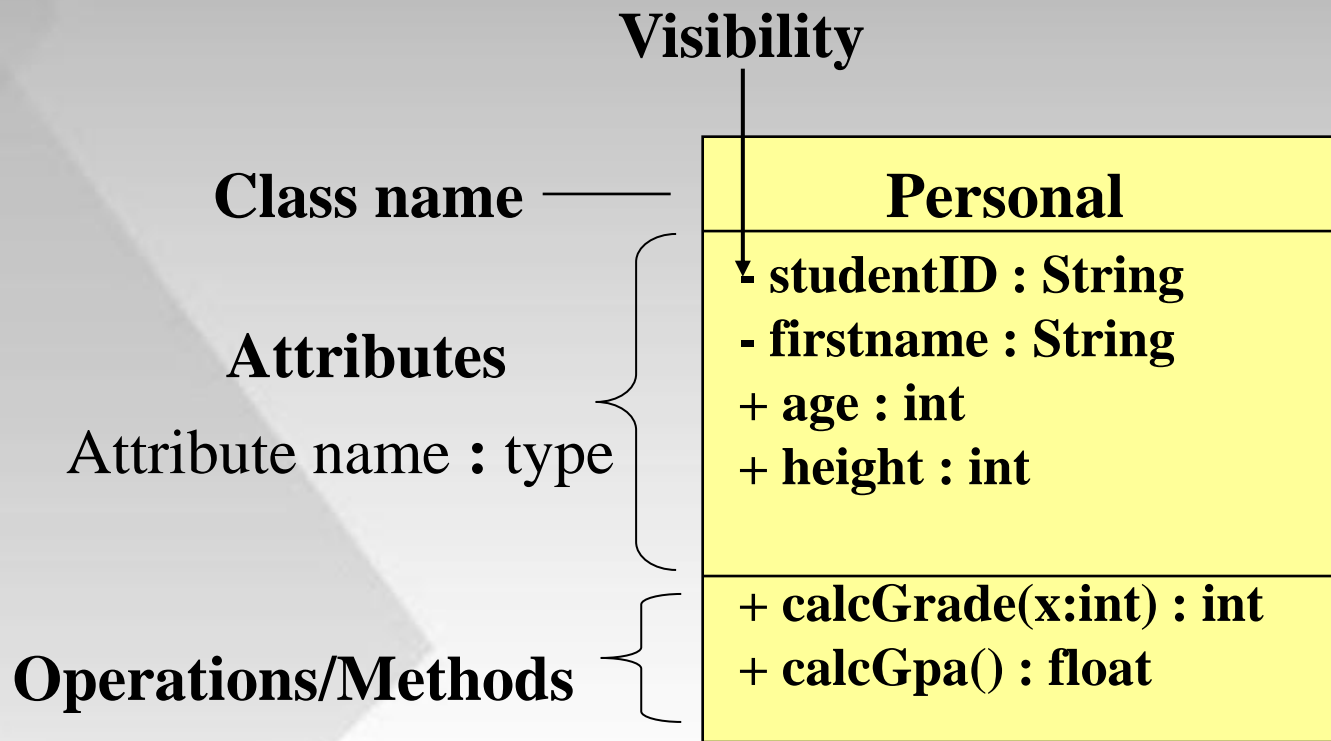
- **Dynamic relationship**

- นาย ก ซ่อมแซม บ้าน
- นาย ก ต่อเติม บ้าน
- นาย ก ทาสี บ้าน

- **Class Diagram เป็น Static relationship**



## ข้อมูลแต่ละ Class ใน Class Diagram



Operation name(Parameter List : type) : Return type



## *Visibility*

- private ( - )** ใช้ได้กับ Attributes หรือ ตัวแปร และ เมธอด(method) หรือ Operations  
เห็นได้ภายในเฉพาะตัว class ไม่สามารถเห็นได้จากภายนอกคลาส  
การเข้าถึงจากภายนอกคลาสนั้นจะกระทำผ่าน Operation ที่เป็น public
- public ( + )** ใช้ได้กับ Attributes หรือ ตัวแปร และ เมธอด(method) หรือ Operations  
มองเห็นและเข้าถึงได้โดยตรงจากภายนอกคลาสนั้น
- protected (#)** ใช้ได้กับ Attributes หรือ ตัวแปร และ เมธอด(method) หรือ Operations  
การเข้าถึงจากภายนอกคลาสนั้นจะกระทำได้อีกถ้าคลาสนั้นอยู่ใน package เดียวกัน  
สงวนไว้สำหรับการทำ Inheritance โดยเฉพาะ  
โดยปกติจะเป็นของ Superclass

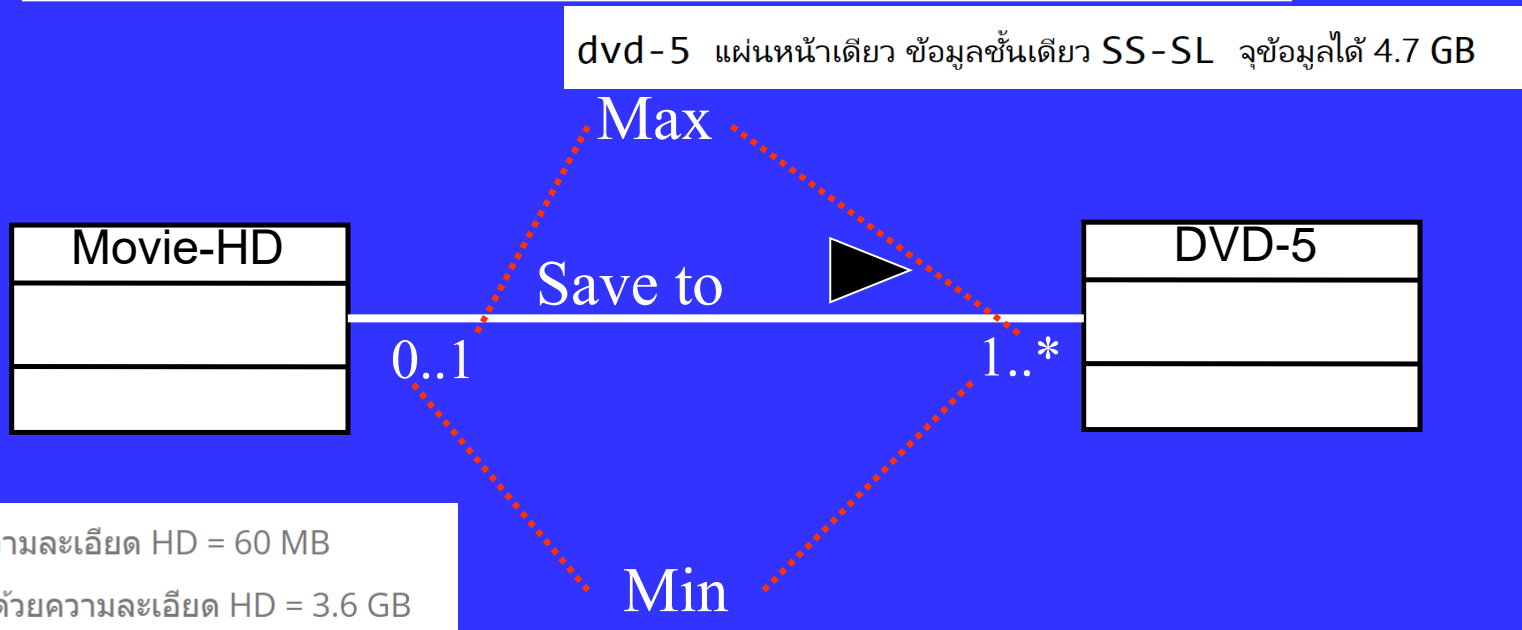
# ความสัมพันธ์ระหว่าง Class

จำนวนความสัมพันธ์ (Relationship) หรือ Multiplicity

เป็นการแสดงจำนวนความสัมพันธ์ Object ของ Class หนึ่ง กับ Objects ของอีก Class หนึ่ง บนเส้นความสัมพันธ์ระหว่างคลาสโดยใช้รูปแบบ

Minumum Cardinality .. Maximum Cardinality

ตัวอย่าง



# ความสัมพันธ์ระหว่าง Class

## วิธีการอ่านค่าความสัมพันธ์

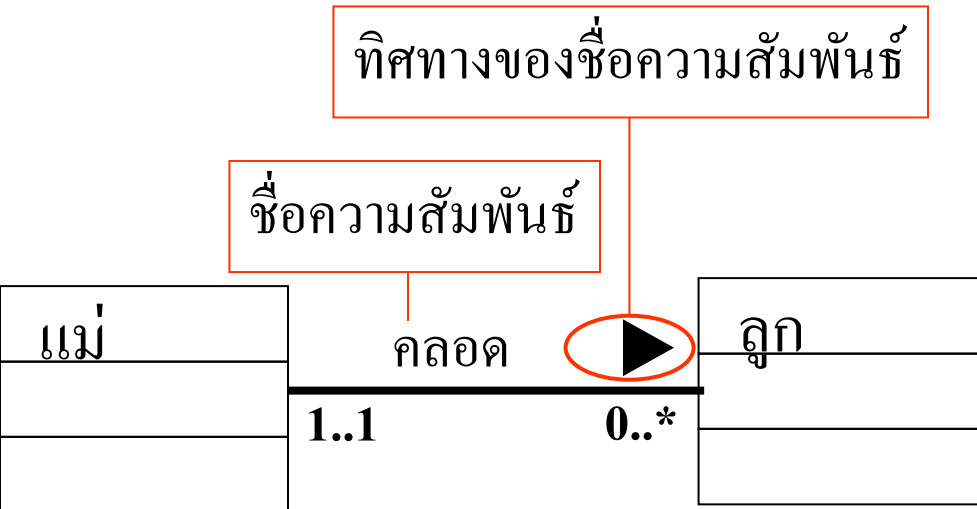


1. อ่านด้านที่มี Min เป็น 1 เพื่อกำหนด Min..Max ของฝั่งตรงข้าม  
เช่น DVD-5 1 แผ่น อาจไม่บันทึกหนังเลย (0) หรือ บันทึกได้ 1 เรื่อง
2. เปลี่ยนไปอ่านฝั่งตรงกันข้าม โดยกำหนดให้สมาชิกเริ่มต้นเป็น 1  
เช่น หนัง 1 เรื่องบันทึกโดยใช้ DVD-5 ต่ำสุด 1 แผ่น และสูงสุดหลายแผ่น(n)

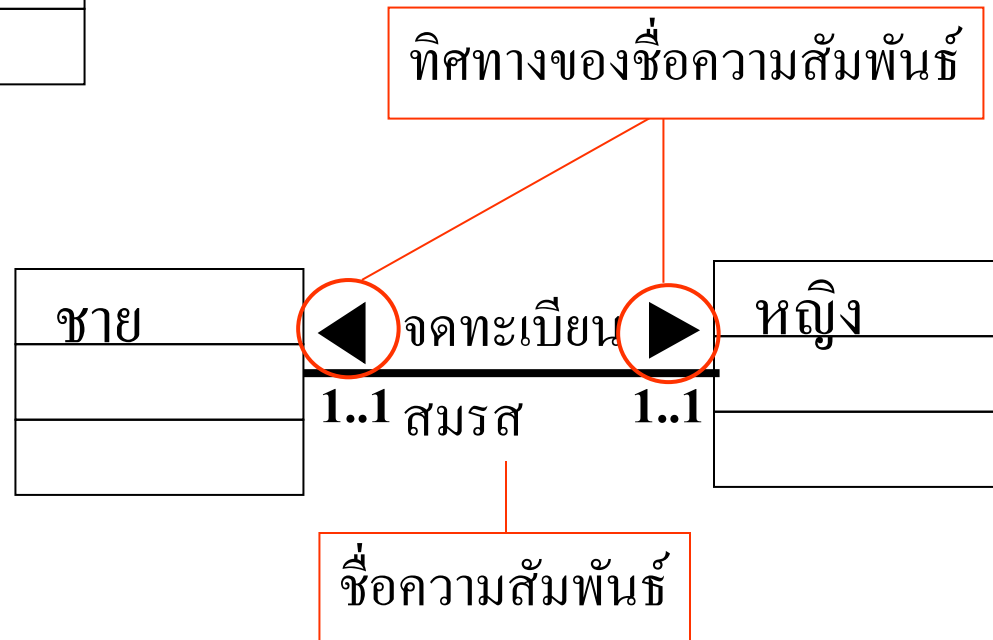
หมายเหตุ ไม่พิจารณาทิศทางของชื่อความสัมพันธ์ เพราะต้องอ่านทั้ง 2 ด้าน  
อยู่แล้ว

# ความสัมพันธ์ระหว่าง Class

## ตัวอย่าง ความสัมพันธ์ Association



ความสัมพันธ์ทิศทางเดียว  
แม่สามารถคลอดลูกได้  
แต่ลูกไม่สามารถ คลอดแม่ได้

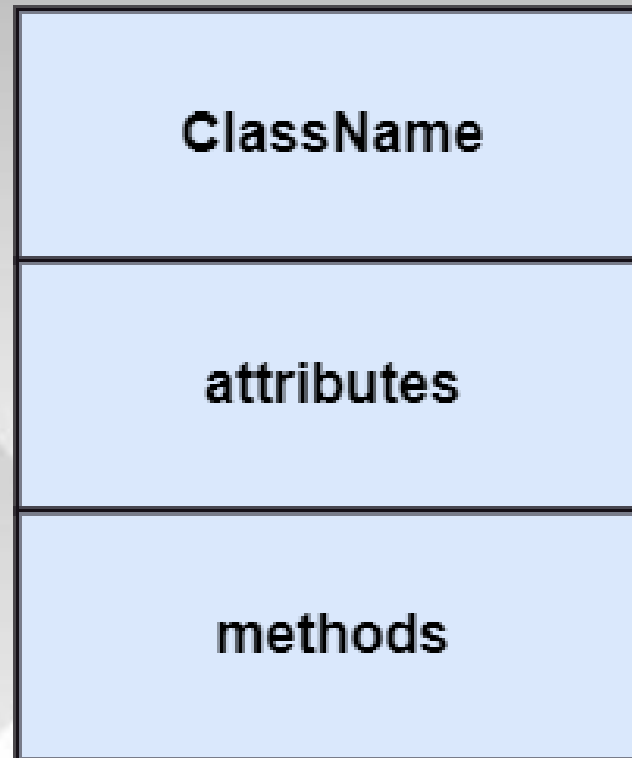


ความสัมพันธ์แบบสองทิศทาง  
ชาย 1 คน จดทะเบียนสมรสกับ  
หญิงได้ 1 คนเท่านั้น และ  
หญิง 1 คน จดทะเบียนสมรสกับ  
ชายได้ 1 คนเท่านั้น



# *Class Diagram*

---

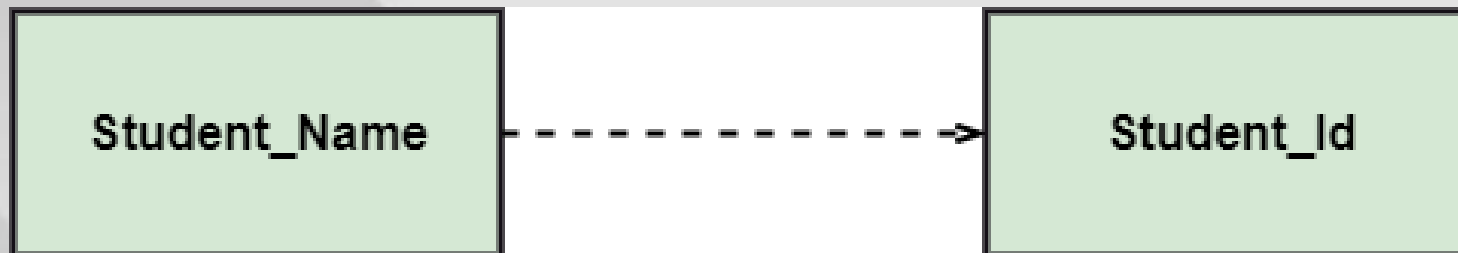






# Relationships

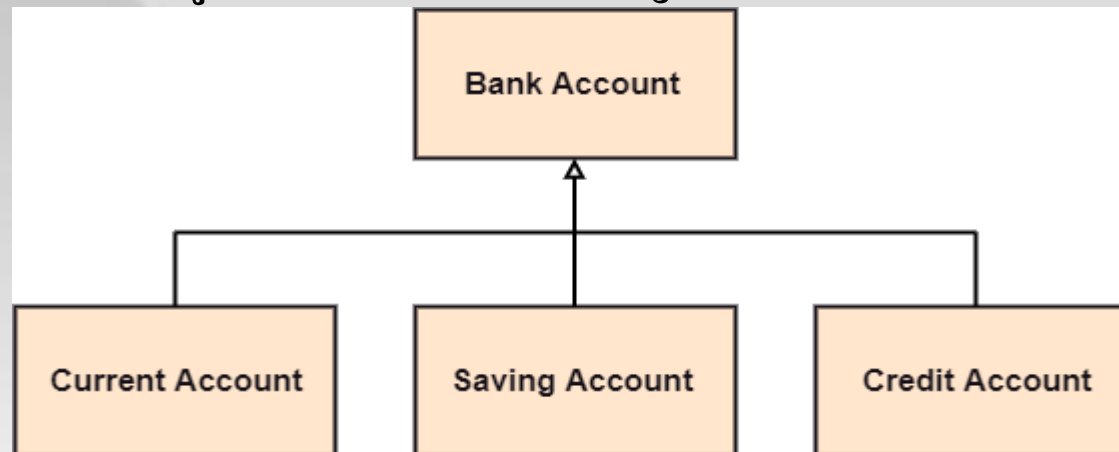
- Dependency: การพึ่งพาอาศัยกันเป็นความสัมพันธ์เชิงความหมายระหว่างสองคลาสขึ้นไป โดยที่การเปลี่ยนแปลงในคลาสหนึ่งทำให้เกิดการเปลี่ยนแปลงในอีกคลาสหนึ่ง มันสร้างความสัมพันธ์ที่อ่อนแอกว่า ในตัวอย่างต่อไปนี้ Student\_Name ขึ้นอยู่กับ Student\_Id





# Relationships

- **Generalization:** ลักษณะทั่วไปคือความสัมพันธ์ระหว่างคลาสแม่ (ซูเปอร์คลาส) และคลาสลูก (คลาสย่อย) ในกรณีนี้ คลาสย่อยจะสืบทอดมาจากคลาสแม่ตัวอย่างเช่น บัญชีกระแสรายวัน บัญชีออมทรัพย์ และบัญชีเครดิต เป็นรูปแบบทั่วไปของบัญชีธนาคาร



บัญชีกระแสรายวัน

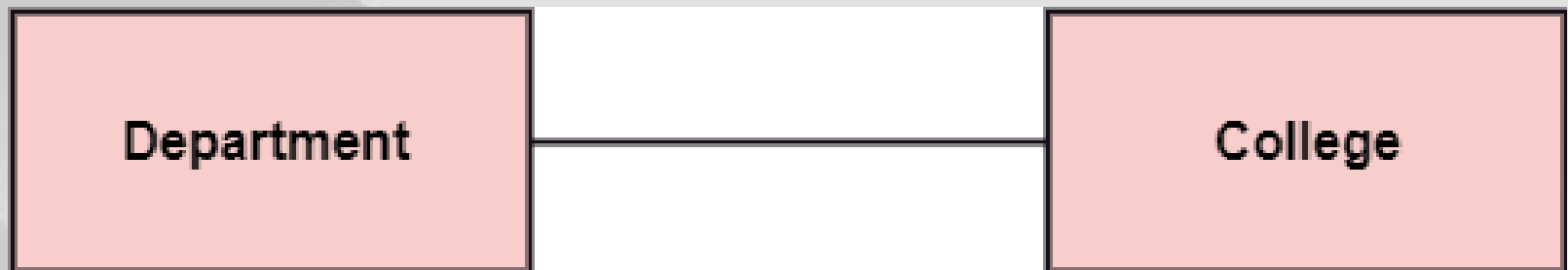
บัญชีออมทรัพย์

บัญชีเครดิต



# Relationships

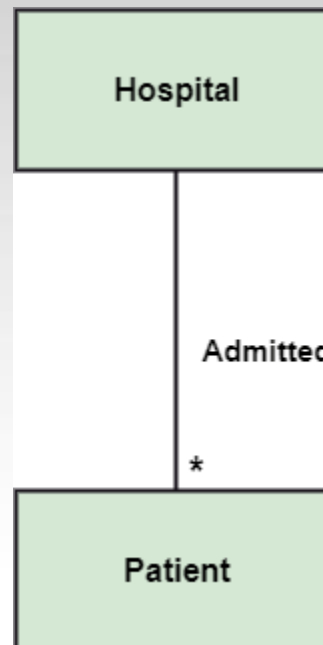
- **Association** : การเชื่อมโยงมันอธิบายการเชื่อมต่อแบบคงที่หรือทางกายภาพระหว่างวัตถุสองชิ้นขึ้นไป มันแสดงให้เห็นว่ามีวัตถุที่ขึ้นในความสัมพันธ์ตัวอย่างเช่น แผนกที่เกี่ยวข้องกับวิทยาลัย





# Relationships

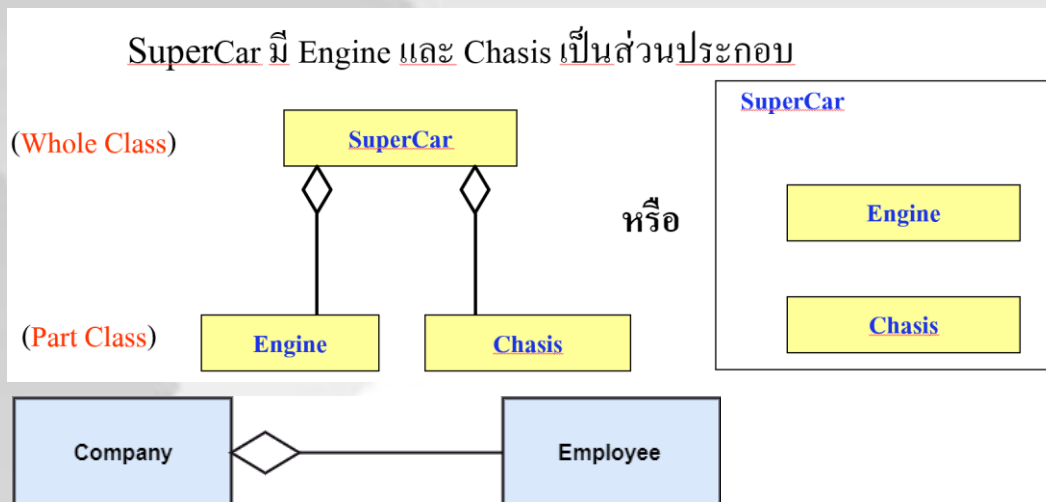
- **Multiplicity:** กำหนดช่วงเฉพาะของอินสแตนซ์แอตทริบิวต์ที่อนุญาต  
ในกรณีที่ไม่ได้ระบุช่วง ระบบจะถือว่าช่วงหนึ่งเป็นค่าดีฟอลต์  
หลากหลายตัวอย่างเช่น ผู้ป่วยหลายรายเข้ารับการรักษาในโรงพยาบาล  
เดียว





# Relationships

- **Aggregation** : การรวมเป็นกลุ่มย่อยของการเชื่อมโยง ซึ่งแสดงถึงความสัมพันธ์ มีความเฉพาะเจาะจงมากกว่าสมาคม มันกำหนดความสัมพันธ์บางส่วนหรือบางส่วน ในความสัมพันธ์ประเภทนี้ คลาสถูกสามารถมีอยู่โดยไม่ขึ้นกับคลาสพาเรนต์บริษัทประกอบด้วยพนักงานจำนวนหนึ่ง และถึงแม้พนักงานคนหนึ่งจะลาออก บริษัทก็ยังมีอยู่





# Relationships

- **Composition:** องค์ประกอบเป็นส่วนย่อยของการรวมกลุ่ม มันแสดงให้เห็นถึงการพึ่งพาระหว่างผู้ปกครองและเด็ก ซึ่งหมายความว่าหากส่วนหนึ่งถูกลบ อีกส่วนหนึ่งก็จะถูกละทิ้งด้วย มันแสดงถึงความสัมพันธ์ที่ทั้งส่วนสมุดติดต่อประกอบด้วยผู้ติดต่อหลายราย และหากคุณลบสมุดติดต่อ ผู้ติดต่อทั้งหมดจะสูญหาย





# Example ข้อมูลแต่ละ Class

```
1 package ejb;
2
3 import java.math.BigDecimal;
4 import javax.ejb.Stateless;
5
6 @Stateless
7 public class CurrencyConverterBean {
8     private BigDecimal USD = new BigDecimal("0.0324803");
9     private BigDecimal THB = new BigDecimal("30.787800");
10
11     public BigDecimal convert(String from, String to, BigDecimal amount)
12     {
13         if(from.equals(to))
14         {
15             return amount;
16         }
17         else
18         {
19             BigDecimal toRate = findRate(to);
20             BigDecimal result = toRate.multiply(amount);
21             return result.setScale(2, BigDecimal.ROUND_UP);
22         }
23     }
24
25     public BigDecimal findRate(String to)
26     {
27         BigDecimal returnValue = null;
28         if(to.equals("THB"))
29         {
30             returnValue = THB;
31         }
32         if(to.equals("USD"))
33         {
34             returnValue = USD;
35         }
36         return returnValue;
37     }
38 }
```

## CurrencyConverterBean

- USD : BigDecimal

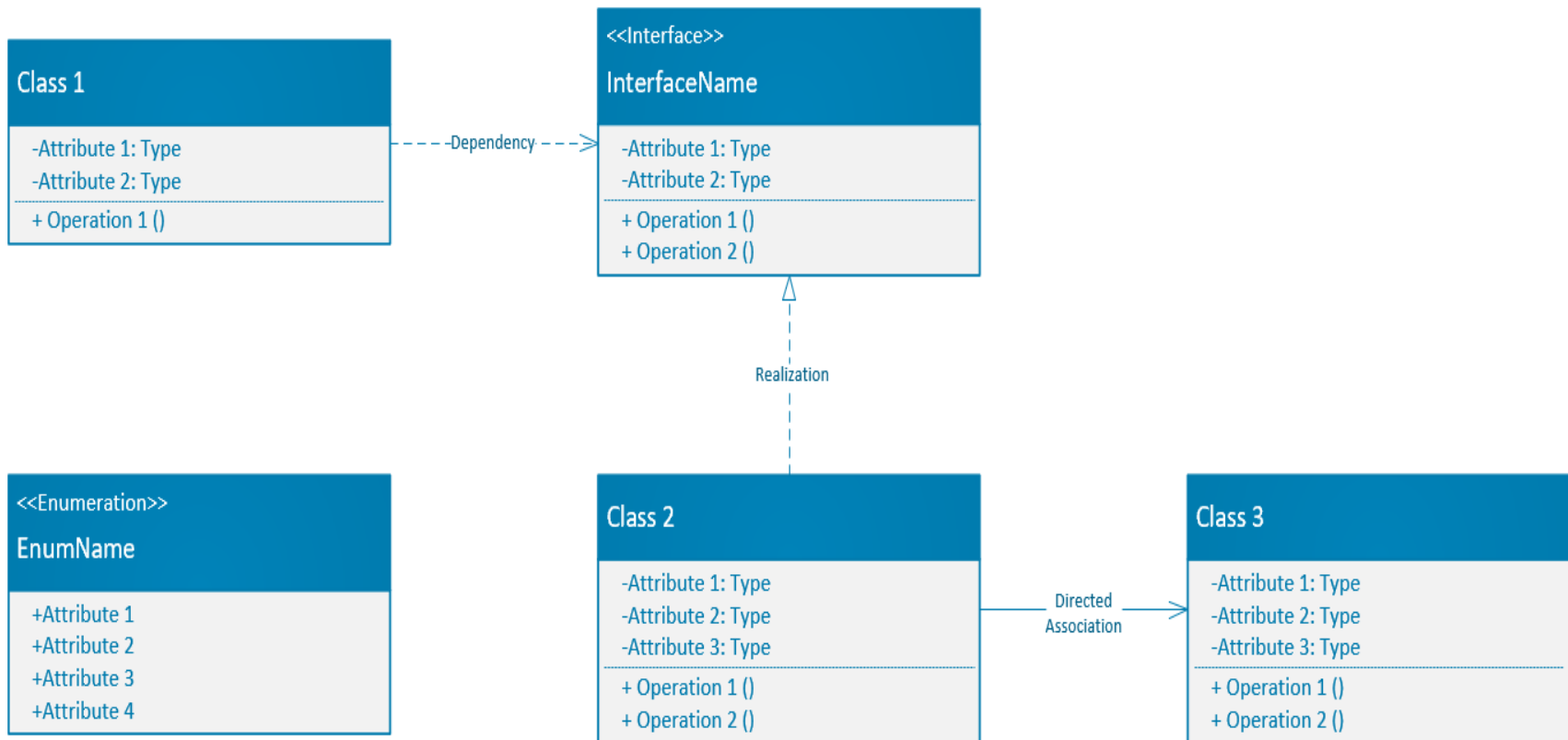
- THB : BigDecimal

+ convert(from:String ,to:String , amount:BigDecimal) : BigDecimal

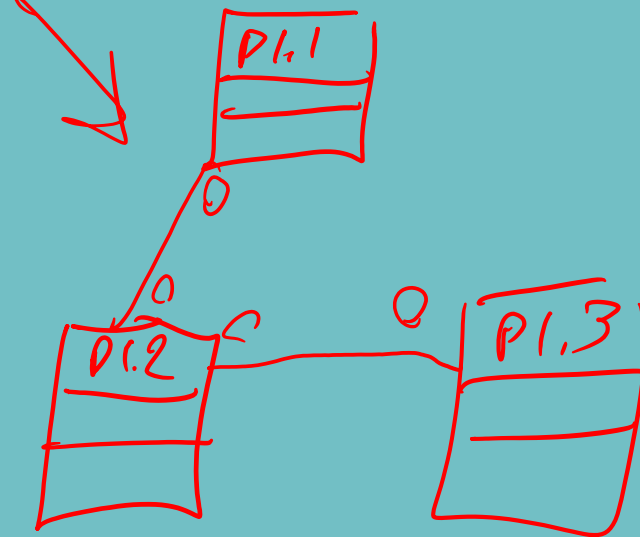
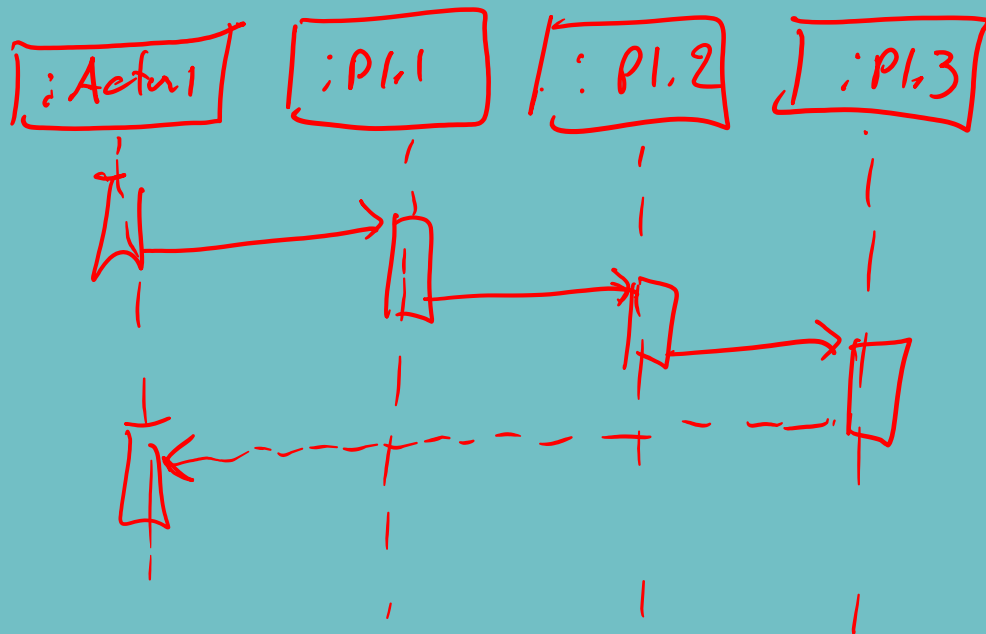
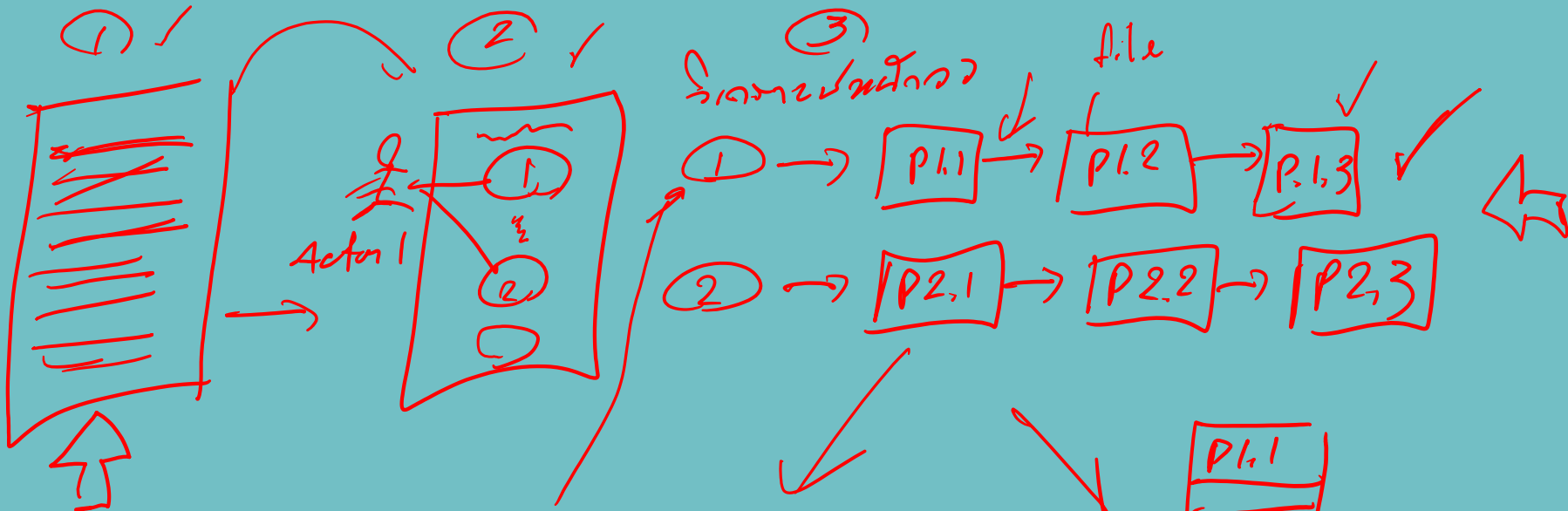
+ findRate(to:String) : BigDecimal



# Example ข้อมูลแต่ละ Class







ตัวอย่าง

ตัวอย่าง