



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

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

# **UML Diagram ส่วนที่ 1**

## **(UML Diagram Part 1)**



*Unified Modeling Language*



## UML (Unified Modeling Language)

---

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

ลูกค้า  
Customer



นักวิเคราะห์ระบบ  
System Analyst



นักพัฒนา  
Developer





## UML (Unified Modeling Language) (ต่อ)

---

UML (Unified Modeling Language) เป็นภาษาการสร้างแบบจำลองกราฟิกสำหรับวัตถุประสงค์ทั่วไปในด้านวิศวกรรมซอฟต์แวร์ UML ใช้เพื่อระบุการดำเนินการด้านวิศวกรรมซอฟต์แวร์ด้วยภาพ

เริ่มแรกได้รับการพัฒนาโดย Grady Booch, Ivar Jacobson และ James Rumbaugh ในปี 1994-95 โดยใช้ซอฟต์แวร์ Rational และการพัฒนาเพิ่มเติมได้ดำเนินการจนถึงปี 1996 ในปี 1997 มันถูกนำไปใช้เป็นมาตรฐานโดย Object Management Group



## UML (Unified Modeling Language) (ต่อ)

UML ได้รับการพัฒนาในปี 1994-95 โดย Grady Booch, Ivar Jacobson และ James Rumbaugh ที่ Rational Software ในปี 1997 Object Management Group (OMG) ได้นำมาใช้เป็นมาตรฐาน

Object Management Group (OMG) เป็นการเชื่อมโยงของบริษัทต่างๆ ที่ใช้งาน UML และร่วมกันพัฒนา UML เป็นมาตรฐานแบบเปิด

OMG ก่อตั้งขึ้นเพื่อสร้างมาตรฐานแบบเปิดซึ่งสนับสนุนการทำงานของระบบเชิงอ็อบเจกต์เป็นหลัก แต่ UML ไม่ได้ถูกจำกัดเฉพาะงานด้านซอฟต์แวร์แต่ยังสามารถใช้สำหรับการสร้างแบบจำลองระบบที่ไม่ใช่ซอฟต์แวร์ได้ด้วย OMG ได้รับการยอมรับอย่างมากสำหรับมาตรฐาน Common Object Request Broker Architecture (CORBA)

# Grady Booch



# James Rumbaugh

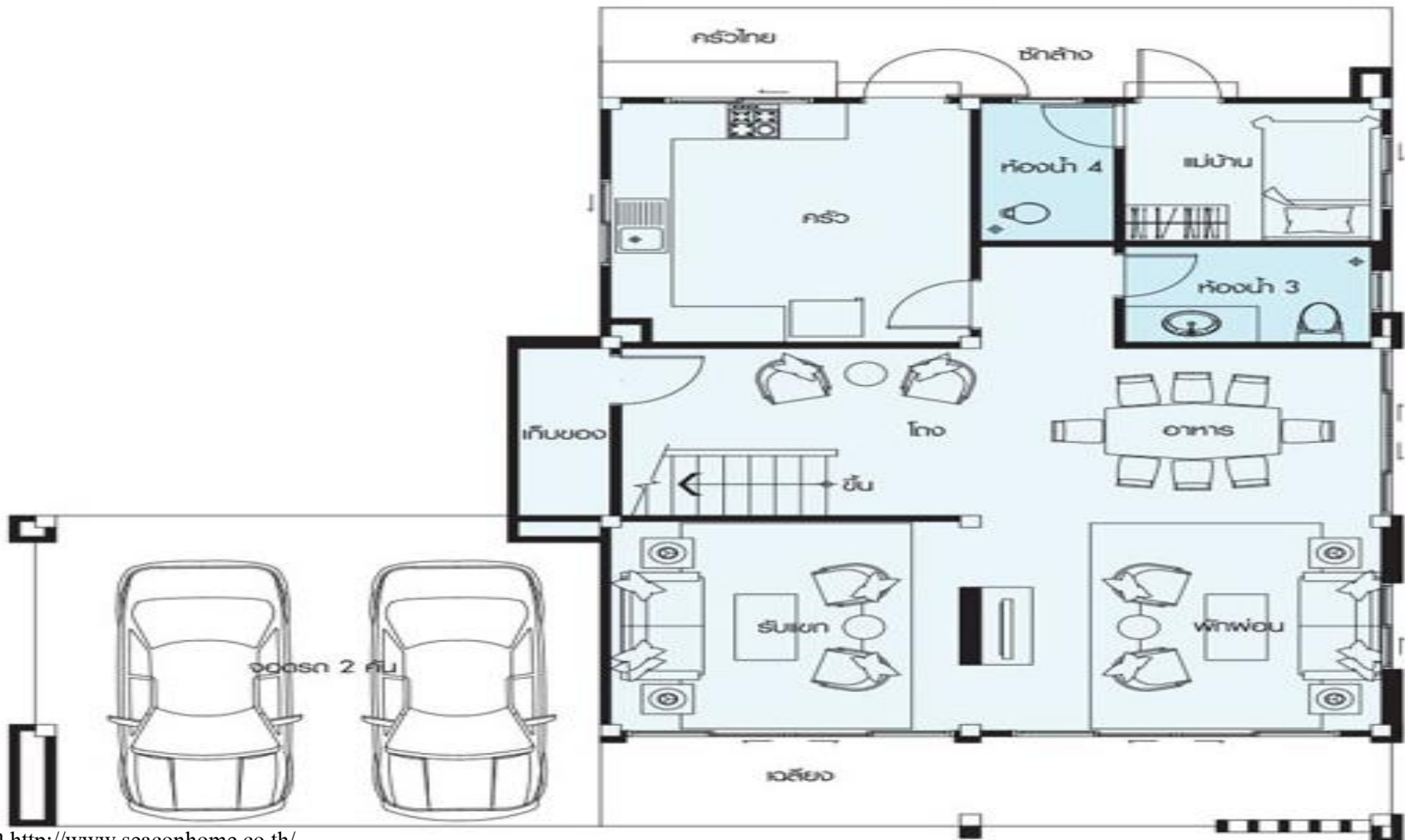


# Ivar Jacobson





## แบบแปลน



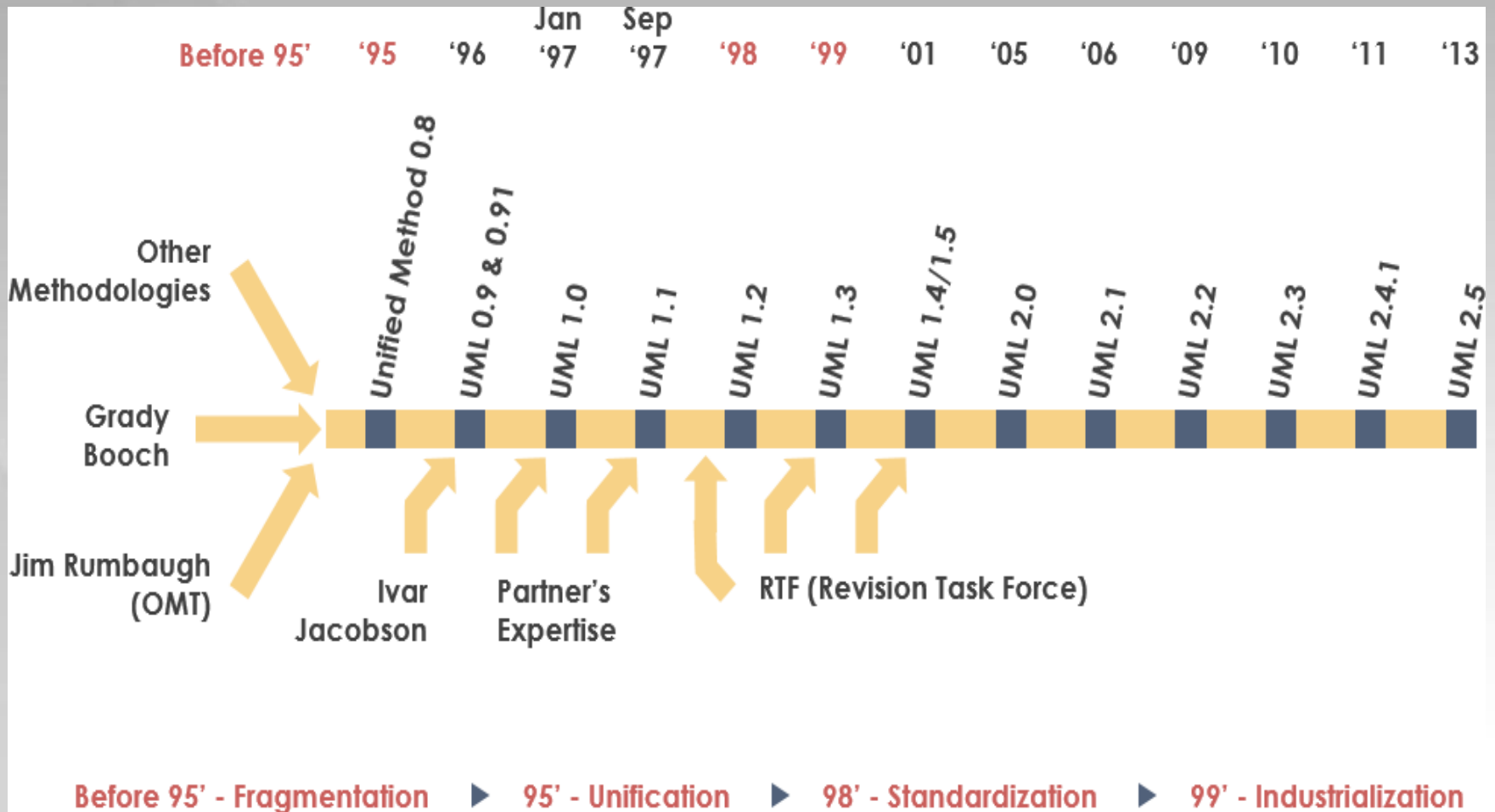


## ที่มาและประวัติของ UML

UML นั้นเก่าแก่พอๆ กับประวัติศาสตร์ของภาษาเชิงอ็อบเจกต์ ซึ่งเริ่มดำเนินการในปลายทศวรรษ 1980 เดิมทีมีพื้นฐานมาจากวิธี Booch ซึ่ง Grady Booch นำมาใช้ ในปี 1994 James Rumbaugh แห่ง General Electric ได้รวมเข้ากับเทคนิคการสร้างแบบจำลองอ็อบเจกต์ ที่เป็นที่ยอมรับ นอกจากนี้ยังมีวิธีการและหลักการอื่น ๆ อีกสองสามข้อ ( บางส่วนแนะนำโดย Ivar Jacobson) ด้วยเช่นกัน ภายใต้การนำของ Rumbaugh, Jacobson และ Booch - วิธีการ UML 1 อย่างเป็นทางการได้รับการเผยแพร่ในปี 1997 มีมาตรฐานมากมายเพื่อให้แน่ใจว่าจะรักษาความรู้สึกลึกซึ้งคล่องแคล่วในทุกไคอะแกรมต่อมาได้มีการรวมกฎคาร์ดินาลิตี้(cardinality rule)ไว้ในภาษาและมีการจัดตั้งคณะกรรมการแก้ไขเพื่อปรับปรุงหลักการเพิ่มเติม ในที่สุด UML 2.0 ได้เปิดตัวในปี 2548 ซึ่งใช้วิธีการเชิงอ็อบเจกต์เป็นหลัก เพื่อให้ทันกับแนวโน้มอย่างต่อเนื่อง เวอร์ชันที่ปรับปรุงแล้วจึงได้รับการเผยแพร่โดยอิงจาก UML 2.x เป็นระยะๆ



## ที่มาและประวัติของ UML (ต่อ)





## ทำไมเราถึงต้องการ UML?

- UML ให้ภาพรวมหรือมุมมองของระบบด้วยวิธีนี้ แม้แต่ผู้ที่ไม่ใช่ นักพัฒนาซอฟต์แวร์ก็สามารถเข้าใจวิธีการทำงานของระบบได้
- ไลอะแกรม UML ยังมีบทบาทสำคัญในการแก้ปัญหา เราสามารถพรรณนาถึงปัญหาในโลกแห่งความเป็นจริงและค่อยๆ หาวิธีแก้ไขได้
- เนื่องจากไลอะแกรมเหล่านี้เข้าใจง่าย จึงไม่จำเป็นต้องมีความรู้ล่วงหน้าเพื่อทราบ ว่าเอนทิตีทำงานอย่างไรหรือโฟลว์กระบวนการทำงานอย่างไร
- ด้วยการใช้อิอะแกรม UML แบบองค์รวม ทีมพัฒนาซอฟต์แวร์ทั้งหมดสามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ
- UML มีแอปพลิเคชันที่ใช้งานมากมายในการเขียนโปรแกรมเชิงวัตถุทั้งการพัฒนาเว็บ การพัฒนาต้นแบบ การวิเคราะห์ธุรกิจ และอื่นๆ



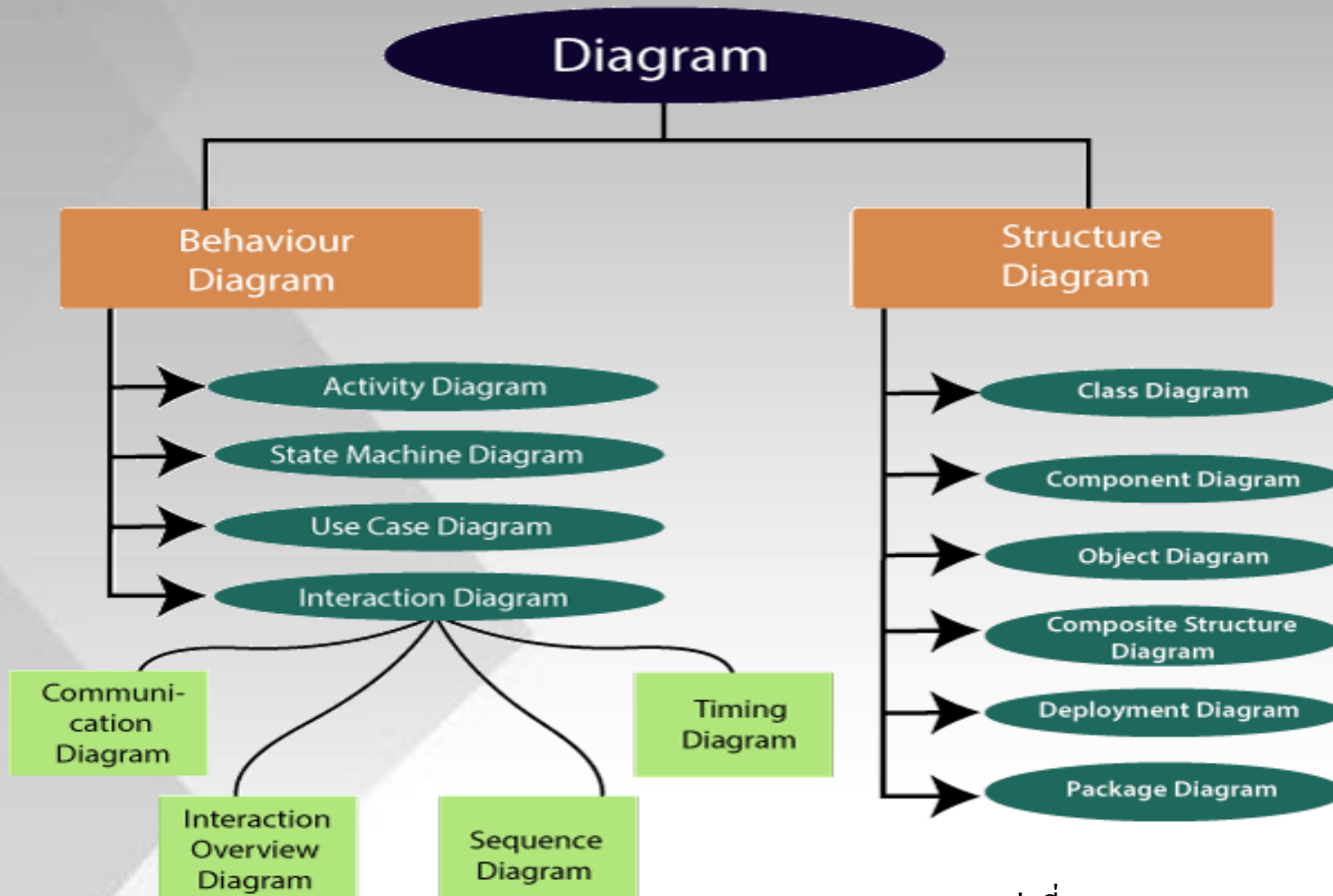
## ไคอะแกรม UML ประเภทต่างๆ

---

ดังที่ทราบ UML ไม่ได้กำหนดประเภทไคอะแกรมใดไคอะแกรมใดหนึ่งเป็นไคอะแกรมหลักเพียงอย่างเดียว ในทางตรงกันข้าม UML มีไคอะแกรมหลากหลายรูปหรือหลากหลายประเภทเพื่อใช้ในงานแต่ละด้านตามความต้องการของผู้ใช้ ประเภทไคอะแกรม UML สามารถจำแนกได้เป็น โครงสร้างหรือพฤติกรรม การจำแนกประเภทเหล่านี้แต่ละประเภทสามารถมีประเภทได้



## ไดอะแกรม UML ประเภทต่างๆ (ต่อ)

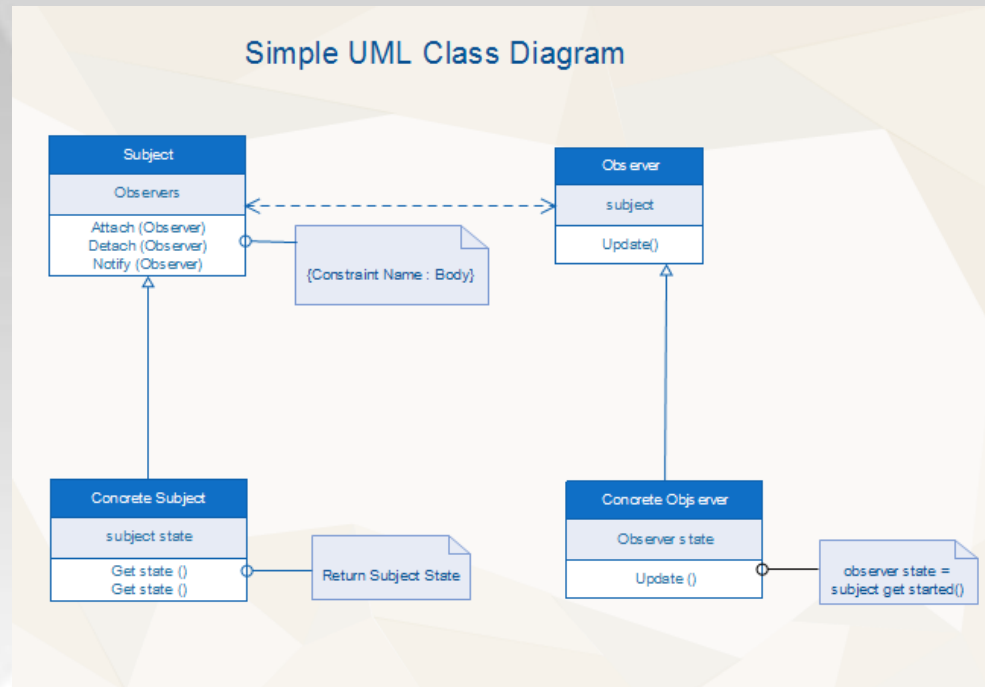






## Class Diagram

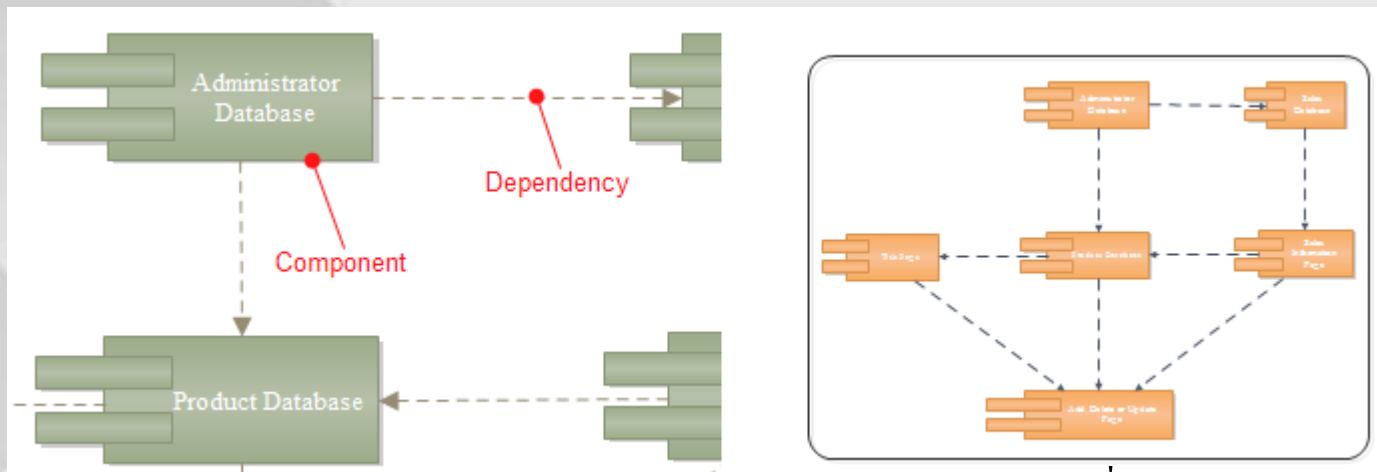
ไดอะแกรม UML พื้นฐานนี้แสดงถึงการแยกส่วนของระบบออกเป็นแต่ละคลาส เราใช้ประเภทไดอะแกรม UML เหล่านี้เพื่อจัดเตรียมการแสดงโปรแกรมแบบคงที่ คลาสมีองค์ประกอบหลักสามอย่าง – ชื่อ คุณลักษณะ และพฤติกรรม





## Component Diagram

**Component Diagram** ส่วนใหญ่ถูกสร้างขึ้นเมื่อระบบมีความซับซ้อนและประกอบด้วยคลาสมากเกินไป ดังนั้นเราจึงแบ่งระบบทั้งหมดออกเป็น ส่วนประกอบต่างๆ และแสดงให้เห็นว่าส่วนประกอบต่างๆ พึ่งพากันและกันอย่างไร ไคอะแกรม UML ประเภทนี้ประกอบด้วยสองสิ่งเท่านั้นคือ ส่วนประกอบและการพึ่งพา



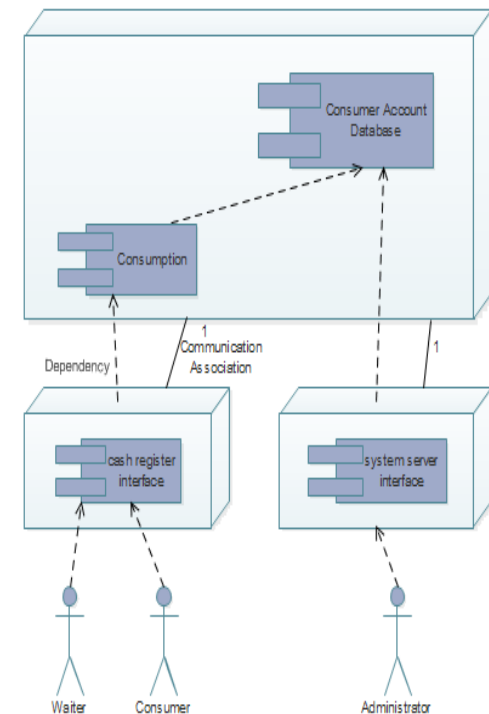




## Deployment Diagram

**Deployment Diagram** จะแสดงการวางองค์ประกอบต่าง ๆ ในระบบจริง ประกอบด้วยสองส่วนหลัก ๆ คือ โหนด และ Artifact โหนดเป็นตัวแทนของหน่วยทางฮาร์ดแวร์ในขณะที่ Artifact เป็นตัวแทนของไคลเอนต์หรือสคริปต์

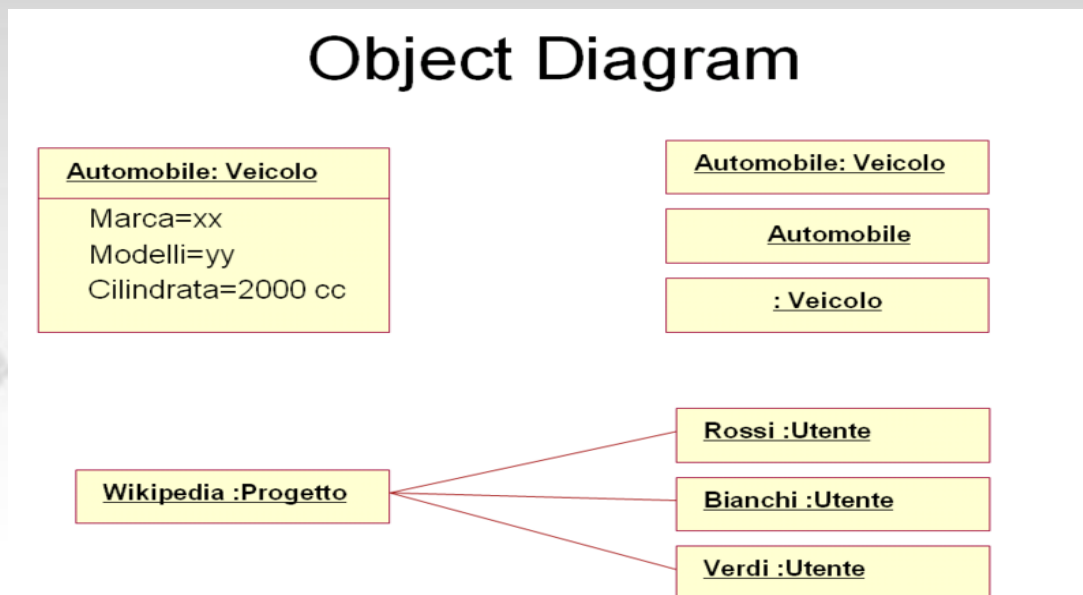
Cafeteria UML Deployment Diagram





## Object Diagram

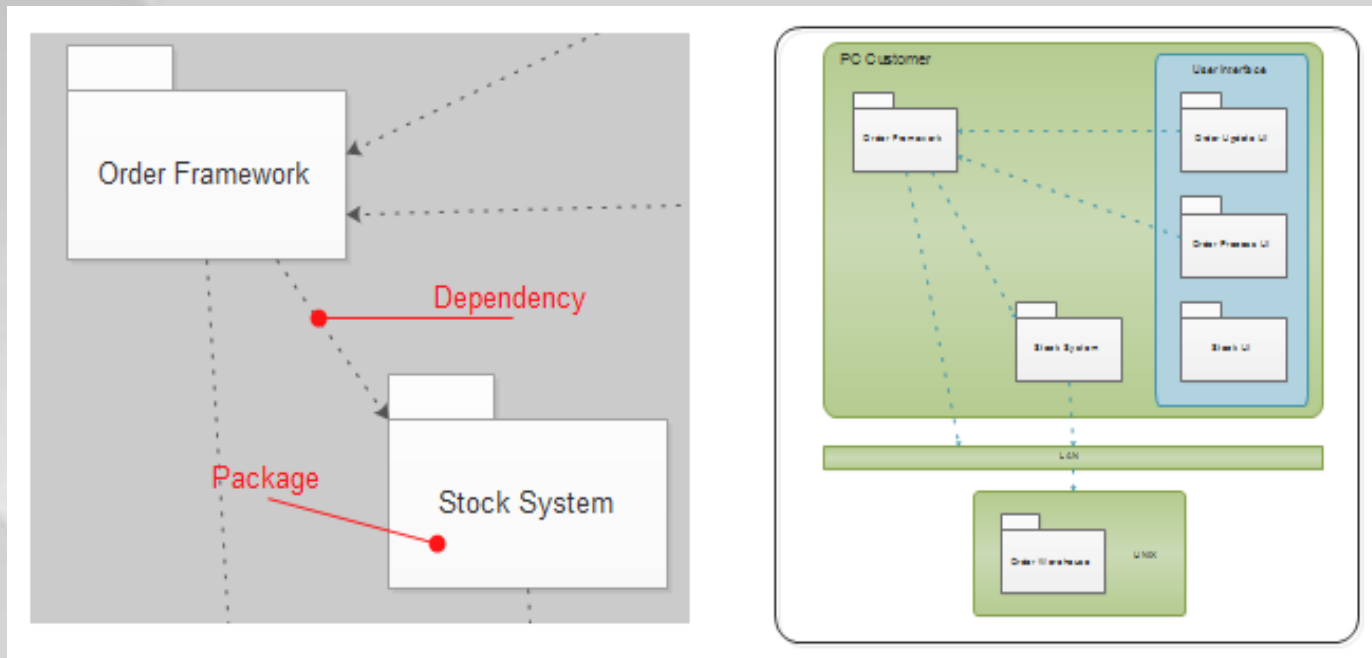
**Object Diagram** เป็นการแจกแจงรายละเอียดของคลาสไดอะแกรมออกมา (เนื่องจากคลาสคือคอลเล็กชันของอ็อบเจกต์ต่างๆ หรือเป็นแหล่งรวมของอ็อบเจกต์ต่างๆ ) แม้ว่าไดอะแกรม UML ต่าง ๆ จะอิงตามองค์ประกอบในโลกแห่งความเป็นจริง แต่มันมีคุณลักษณะที่แตกต่างออกไปตามอ็อบเจกต์และการเชื่อมโยงต่าง ๆ ของอ็อบเจกต์





## Package Diagram

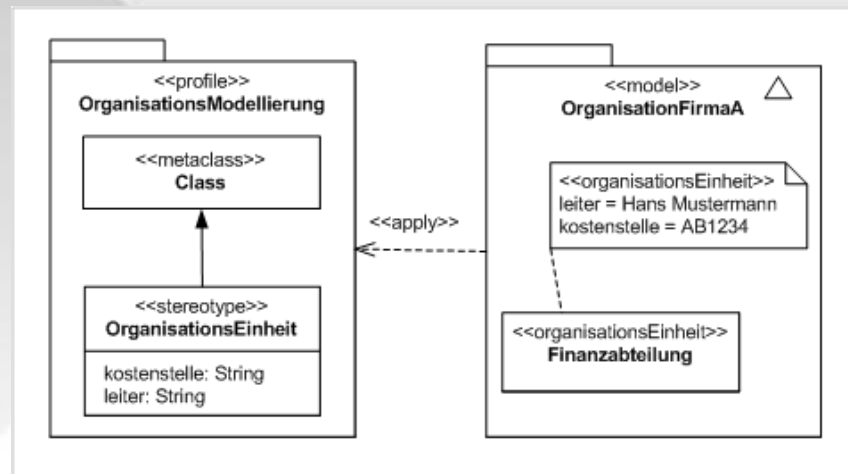
**Package Diagram** เป็นการนำเสนอที่สูงขึ้นของระบบ ประกอบด้วยโมดูลและส่วนย่อยต่าง ๆ ของระบบ ทำให้เราสามารถแตกระบบทั้งหมดและเชื่อมโยงไปยังส่วนประกอบต่างๆ ได้อย่างง่ายดาย แพคเกจจะแสดงด้วยไอคอนโฟลเดอร์ที่มีชื่อ





## Profile Diagram

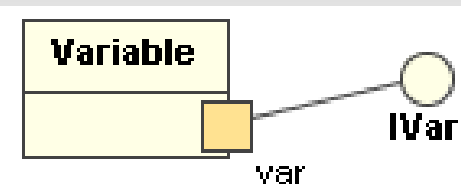
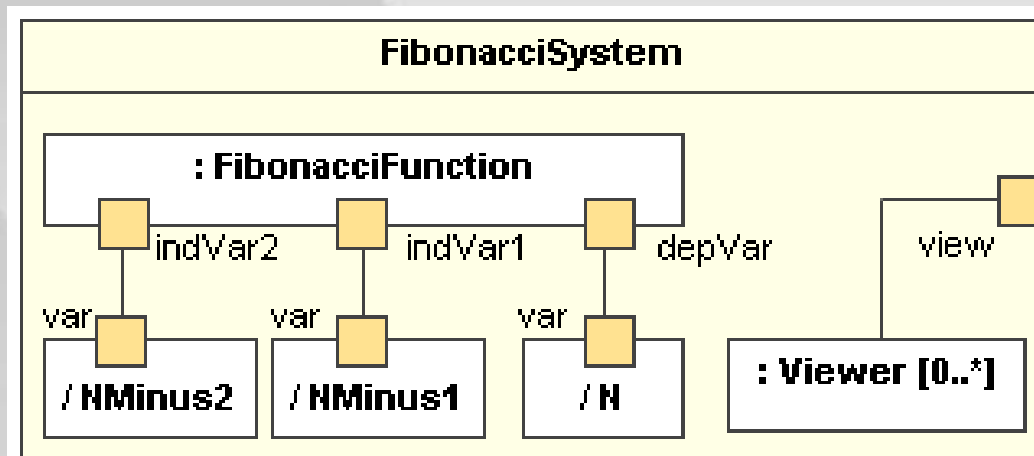
**Profile Diagram** เป็นไดอะแกรม UML แบบใหม่ที่ได้รับการแนะนำใน UML 2 แม้ว่าไดอะแกรม UML ประเภทนี้จะไม่ได้รับความนิยมมากนัก แต่ก็สามารถใช้เพื่อแสดง meta-structure ของระบบได้ โดยจะแสดงว่าอะไรเป็นโปรไฟล์หลัก อะไรเป็น meta-class ๑ โดยอาจมีการเชื่อมต่อภายในองค์ประกอบของโปรไฟล์และระหว่างโปรไฟล์ต่างๆได้





## Composite Structure Diagram

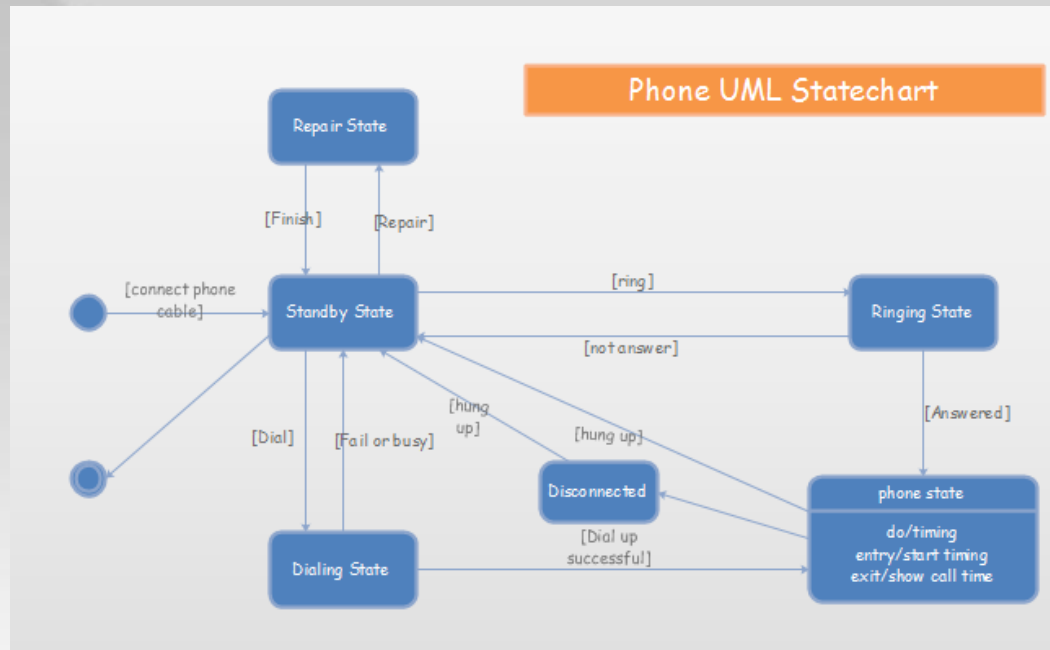
**Composite Structure Diagram** เป็นไดอะแกรม UML ที่ใช้เพื่อแสดงโครงสร้างภายในของคลาส โดยจะแสดงให้เห็นว่าองค์ประกอบต่างๆ ภายในคลาสมีความสัมพันธ์กันอย่างไร และคลาสนั้นเชื่อมโยงกับองค์ประกอบภายนอกอย่างไร ทุกองค์ประกอบจะมี "บทบาท" กำหนดไว้





## State Machine Diagram

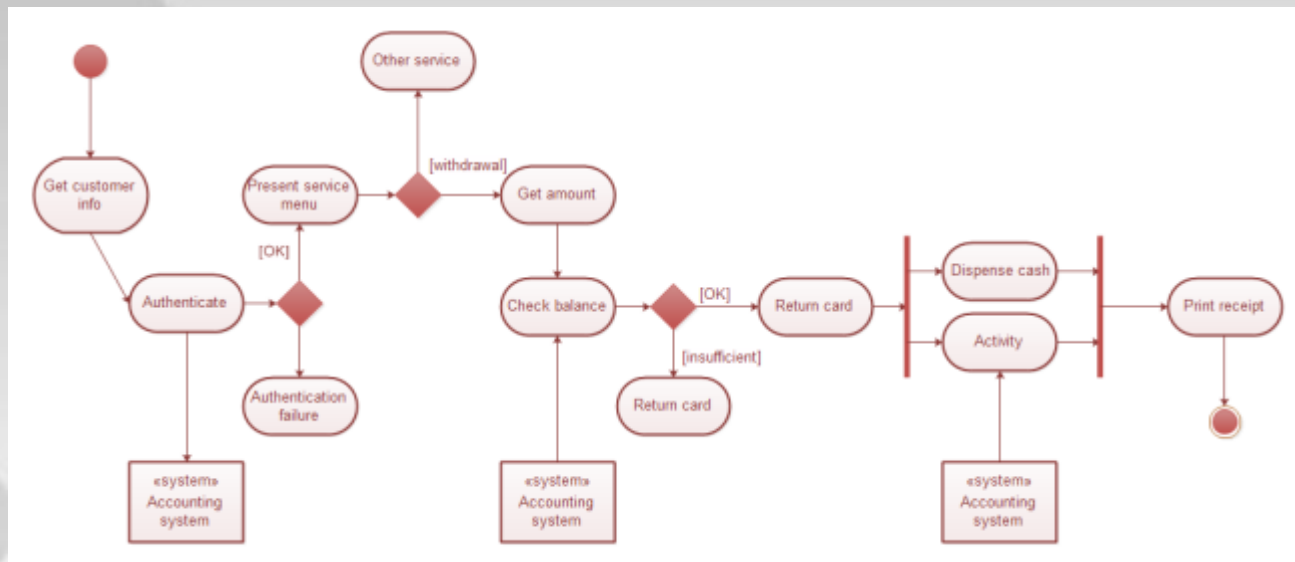
**State Machine Diagram** จะอธิบายสถานะต่างๆ ในการทำงานของระบบ โดยจะมีการเปลี่ยนแปลงสถานะต่าง ๆ อันเกิดขึ้นมาจากการดำเนินการในแต่ละขั้นตอนของระบบ





## Activity Diagram

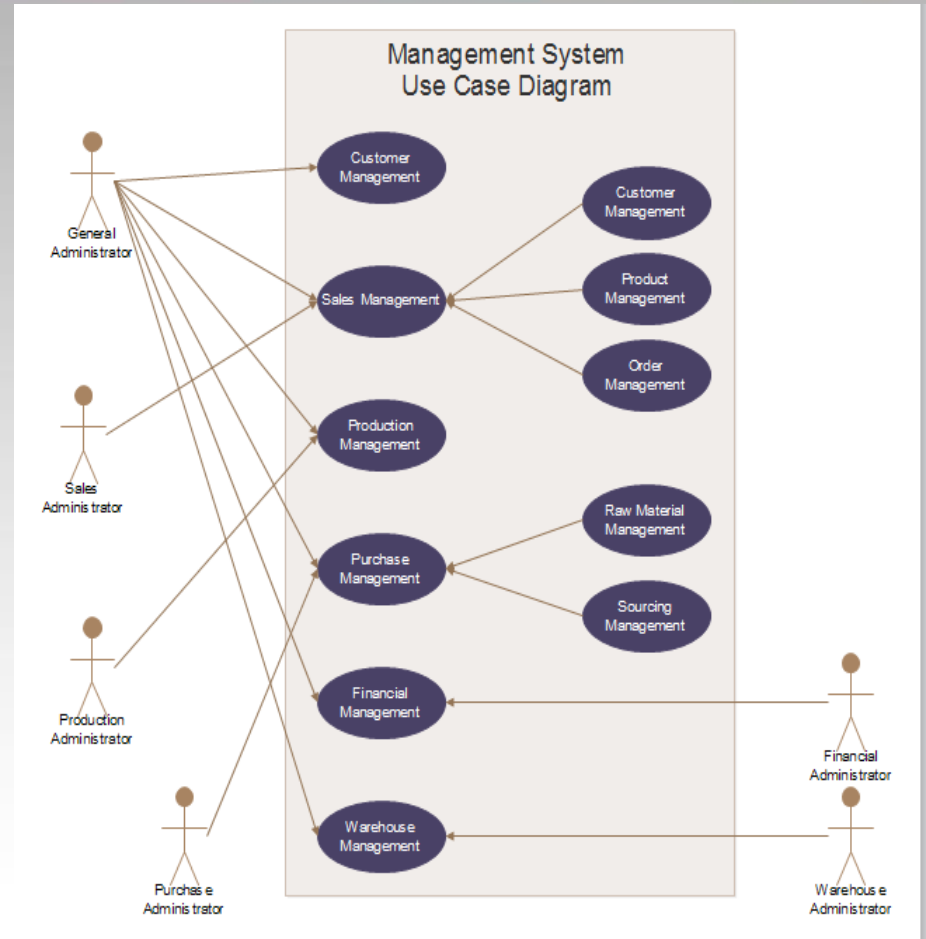
**Activity Diagram** นี้มีการใช้อย่างกว้างขวาง เนื่องจากแสดงถึงกระบวนการต่าง ๆ ของระบบว่าประกอบด้วยกิจกรรมอะไรบ้าง และมีขั้นตอนการดำเนินการของกิจกรรมต่าง ๆ อย่างไร





## Use Case Diagram

Use Case Diagram เป็นหนึ่งในไคอะแกรม UML ที่ได้รับความนิยมมากที่สุดซึ่งแสดงให้เห็นว่าผู้ใช้จะโต้ตอบกับระบบอย่างไร มีใครผู้เกี่ยวข้องกับระบบบ้าง รวมทั้งมีกระบวนการภายในระบบอะไรบ้าง และผู้เกี่ยวข้องไปเกี่ยวข้องกับกระบวนการภายในอะไรบ้าง

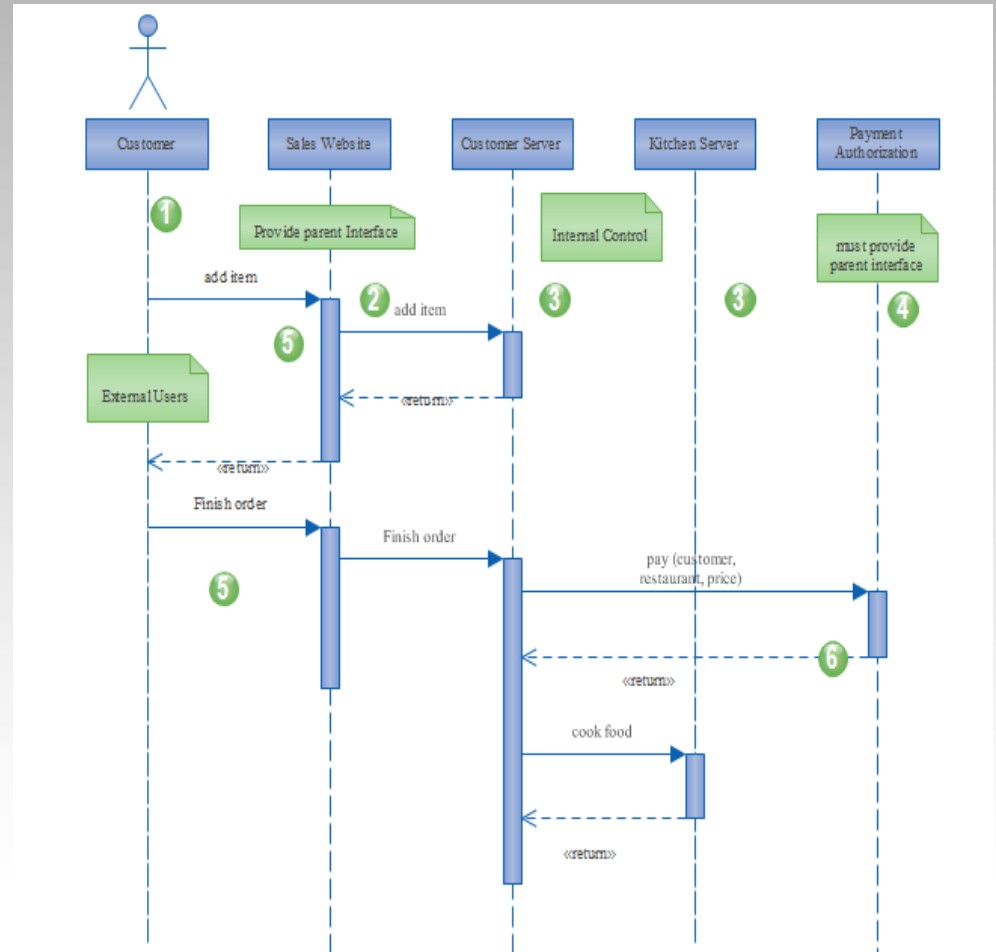






## Sequential Diagram

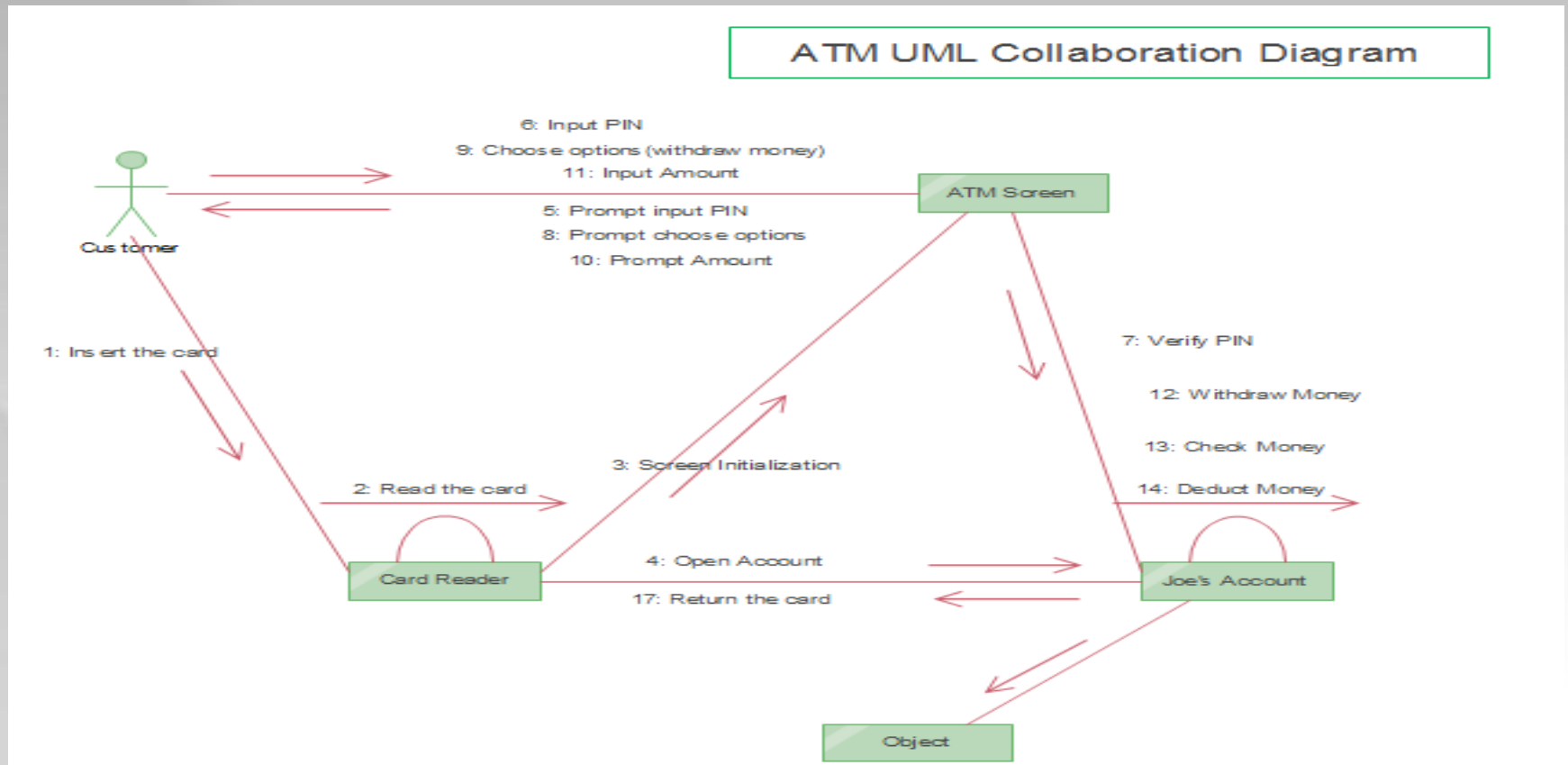
**Sequential Diagram** จะอธิบายการดำเนินการของอ็อบเจกต์ต่าง ๆ ในระบบ ณ ช่วงเวลาต่าง ๆ ทำให้เราสามารถมองเห็นลำดับการทำงานก่อนหลัง ลำดับการปฏิสัมพันธ์ของอ็อบเจกต์ต่าง ๆ ในแต่ละช่วงเวลา





## Communication Diagram หรือ collaborative diagram

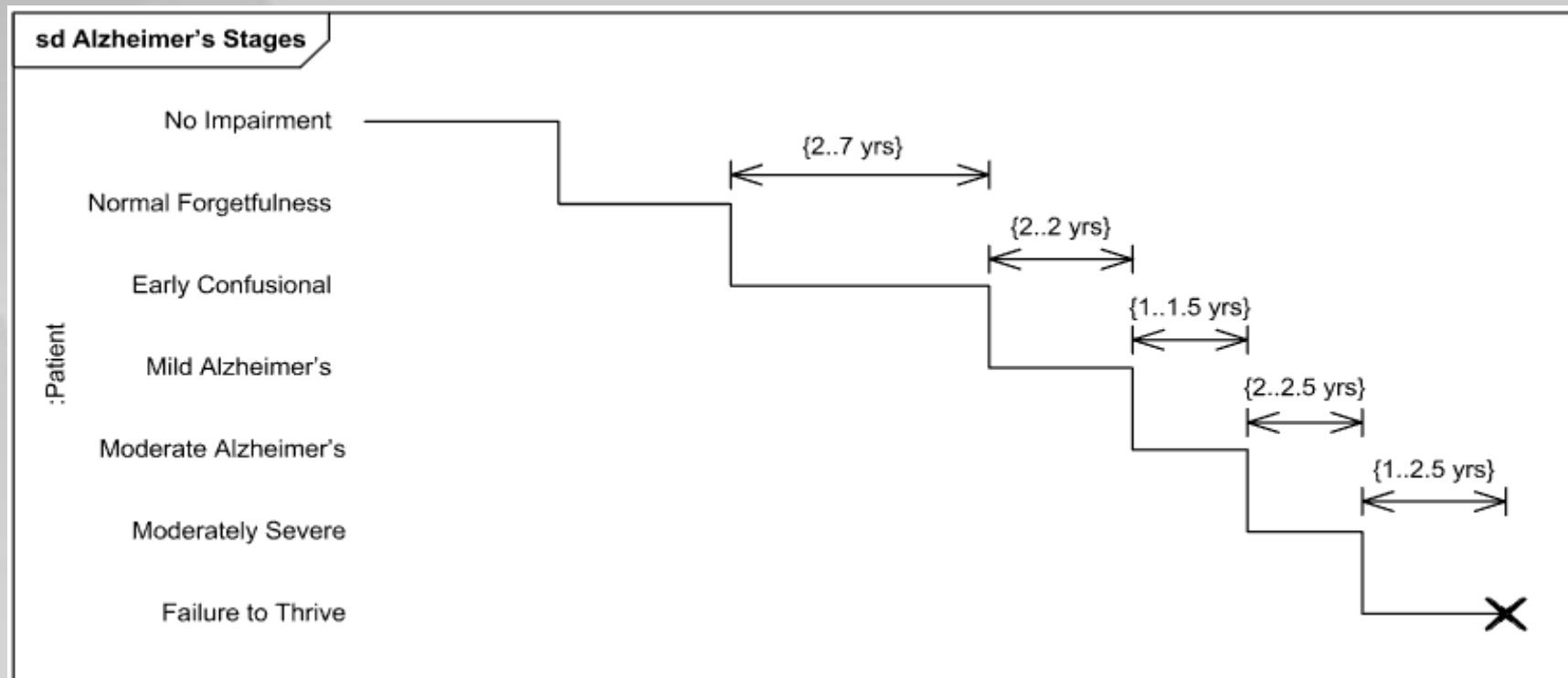
จะแสดงให้เห็นว่าอ็อบเจกต์ต่าง ๆ สื่อสารอะไรกันบ้าง สื่อสารกันอย่างไร





## Timing Diagram

**Timing Diagram** แม้ว่าจะคล้ายกับ **Sequential Diagram** แต่ก็แสดงถึงพฤติกรรมของอ็อบเจกต์ใดๆ ในช่วงระยะเวลาที่กำหนด





## Interactive Overview Diagram

### Interactive Overview Diagram

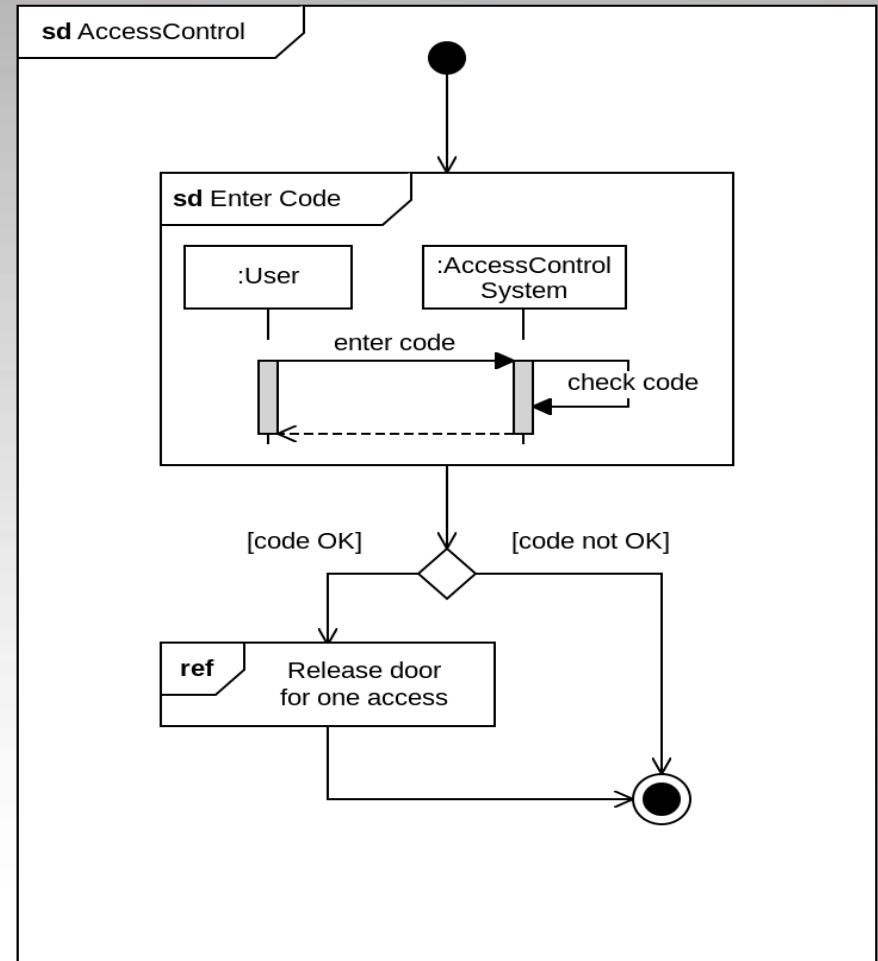
อาจดูคล้ายกับ Activity Diagram

แม้ว่าแทนที่จะแสดงกิจกรรม แต่

### Interactive Overview Diagram

แสดงลำดับการโต้ตอบต่างๆ

กิจกรรมแทน





## ลักษณะของUML

---

- UML ใช้สร้างแบบจำลองของระบบซอฟต์แวร์
- UML แตกต่างจากภาษาโปรแกรมอื่น ๆ เช่น C ++, Python เป็นต้น เพราะ UML เป็นภาษาเชิงสัญลักษณ์โคบใช้รูปภาพสื่อความหมาย
- UML สัมพันธ์กับการวิเคราะห์และการออกแบบเชิงอ็อบเจกต์
- UML ถูกใช้เพื่อเรามองเห็นภาพการทำงานของระบบ



## การสร้างแบบจำลองแนวคิด

ก่อนดำเนินการกับแนวคิดของ UML เราควรเข้าใจพื้นฐานของโมเดลแนวคิดก่อน  
แบบจำลองแนวคิดประกอบด้วยแนวคิดที่สัมพันธ์กันหลายอย่าง ทำให้ง่ายต่อการเข้าใจอ็อบ  
เจ็กต์และวิธีที่อ็อบเจ็กต์โต้ตอบกัน นี่เป็นขั้นตอนแรกก่อนที่จะวาดไดอะแกรม UML

แนวคิดเชิงอ็อบเจ็กต์บางส่วนที่จำเป็นในการเริ่มต้นด้วย UML:

- อ็อบเจ็กต์: อ็อบเจ็กต์เป็นองค์ประกอบต่าง ๆ ของระบบ อ็อบเจ็กต์เกิดจากการสร้างของ  
คลาสหรือแหล่งกำเนิดอ็อบเจ็กต์
- คลาส: คลาสคือพิมพ์เขียวสำหรับอ็อบเจ็กต์ ซึ่งหมายความว่าคลาสนั้นกำหนดตัวแปรและ  
เมธอดทั่วไปสำหรับอ็อบเจ็กต์ หรือกล่าวได้ว่าคลาสเป็นต้นกำเนิดของอ็อบเจ็กต์



## การสร้างแบบจำลองแนวคิด

- Abstraction: คือกระบวนการแสดงลักษณะสำคัญของ อ็อบเจกต์ต่อผู้ใช้ในขณะที่ยกเว้นข้อมูลที่ไม่เกี่ยวข้อง
- Inheritance: การสืบทอดเป็นกระบวนการของการสืบทอดคลาสใหม่จากคลาสที่มีอยู่
- Polymorphism: มันเป็นกลไกของการแสดงอ็อบเจกต์ที่มีหลายรูปแบบที่ใช้เพื่อวัตถุประสงค์ที่แตกต่างกัน
- Encapsulation: การห่อหุ้มข้อมูลและฟังก์ชันเข้าไว้ด้วยกันเป็นอันหนึ่งอันเดียวกัน ทำให้สามารถเชื่อมต่อระหว่างกันได้อย่างมีประสิทธิภาพ



## การวิเคราะห์และการออกแบบ OO (OO Analysis and Design)

OO คือ การวิเคราะห์อ็อบเจกต์และการออกแบบ คือการระบุอ็อบเจกต์ที่มีในระบบสำหรับการออกแบบ และการวิเคราะห์การทำงานของอ็อบเจกต์เหล่านั้นในระบบ ซึ่งการวิเคราะห์จะมีประสิทธิภาพมากขึ้นได้หากเราสามารถระบุอ็อบเจกต์ต่าง ๆ ในระบบได้ เมื่อเราระบุอ็อบเจกต์แล้ว ความสัมพันธ์ของอ็อบเจกต์จะถูกระบุ และการออกแบบการทำงานก็สามารถถูกระบุขึ้นได้ด้วย

วัตถุประสงค์ของ OO คือ:

- เพื่อระบุอ็อบเจกต์ของระบบ
- เพื่อระบุความสัมพันธ์ของอ็อบเจกต์
- เพื่อออกแบบการทำงานที่เกิดขึ้นในระบบ





## แนวคิด OO

ต่อไปนี้เป็นขั้นตอนที่ใช้และนำแนวคิด OO ไปใช้:

### ขั้นตอนที่ 1: การวิเคราะห์ OO

วัตถุประสงค์หลักของการวิเคราะห์ OO คือการระบุอ็อบเจกต์และอธิบายอย่างถูกต้อง หลังจากระบุอ็อบเจกต์แล้ว ขั้นตอนการออกแบบก็สามารถทำได้ง่าย เราจำเป็นต้องระบุอ็อบเจกต์ที่มีในระบบ เมื่อระบุอ็อบเจกต์ได้ก็สามารถระบุหน้าที่การทำงานของอ็อบเจกต์ได้ แต่ละอ็อบเจกต์จะมีหน้าที่ของตัวเองและจะต้องดำเนินการร่วมกับอ็อบเจกต์อื่นตามความต้องการของระบบ



## แนวคิด OO (ต่อ)

### ขั้นตอนที่ 2: การออกแบบ OO

ขั้นตอนนี้จะเน้นที่ความต้องการของระบบเป็นหลัก โดยในขั้นตอนนี้จะนำอ็อบเจกต์ต่าง ๆ มาเชื่อมต่อเข้าการทำงานเข้าด้วยกันตามความสัมพันธ์ที่กำหนดไว้ในระบบ หลังจากการเชื่อมต่อเสร็จสิ้น ขั้นตอนการออกแบบก็จะเสร็จสมบูรณ์เช่นกัน

### ขั้นตอนที่ 3: การใช้งาน OO

นี่เป็นขั้นตอนสุดท้ายที่เกิดขึ้นหลังจากการออกแบบเสร็จสิ้น การใช้การระบบที่ออกแบบไว้นำไปสู่การเขียนโปรแกรมภาษาต่าง เช่น C++ หรือ Java ๗



## บทบาทของ UML ในการออกแบบ OO

เนื่องจาก UML เป็นภาษาการสร้างแบบจำลองที่ใช้ในการสร้างแบบจำลองซอฟต์แวร์รวมถึงระบบที่ไม่ใช่ซอฟต์แวร์ แต่ที่นี้จะเน้นที่การสร้างแบบจำลองซอฟต์แวร์เป็นหลัก เราจึงจำเป็นต้องเข้าใจความสัมพันธ์ระหว่างการออกแบบ OO กับ UML การออกแบบ OO สามารถแปลงเป็น UML ได้ตามความต้องการภาษาที่เป็น OO มีอิทธิพลต่อโลกแห่งการเขียนโปรแกรม

UML นั้นเป็นการผสมผสานระหว่างสัญลักษณ์เชิงอ็อบเจกต์ เช่น Object-Oriented Design (OOD), Object Modeling Technique (OMT) และ Object-Oriented Software Engineering (OOSE) UML การใช้จุดแข็งของแนวทางทั้งสามนี้จะทำให้ความสอดคล้องขององค์ประกอบต่าง ๆ ในระบบเพิ่มมากขึ้น



## *UML-Building Blocks*

UML ประกอบด้วยองค์ประกอบหลักสามส่วน ได้แก่ สิ่งต่าง ๆ ที่เกี่ยวข้อง ความสัมพันธ์ และไคอะแกรม การสร้างบล็อกจะสร้างไคอะแกรมแบบจำลอง UML ที่สมบูรณ์หนึ่งรายการ เกิดจากการดูหรือพิจารณาบล็อกไคอะแกรมต่าง ๆ และพิจารณาถึงการทำงานที่ครบถ้วน ซึ่งจะมีบทบาทสำคัญต่อการพัฒนาไคอะแกรม UML ให้สมบูรณ์

การสร้าง UML มีองค์ประกอบพื้นฐานดังนี้ :

- สิ่งต่าง ๆ ที่เกี่ยวข้อง(Things)
- ความสัมพันธ์(Relationships)
- ไคอะแกรม(Diagrams)

สิ่งต่าง ๆ ที่เกี่ยวข้อง(*Things*)



## สิ่งต่าง ๆ ที่เกี่ยวข้อง(Things)

---

สิ่งใดก็ตามที่เป็นตัวตนหรืออ็อบเจกต์ในโลกแห่งความเป็นจริง เรียกว่า สิ่งต่าง ๆ ที่เกี่ยวข้อง สามารถแบ่งออกเป็นหลายประเภท:

- สิ่งที่เป็นโครงสร้าง-Structural things
- สิ่งที่เป็นพฤติกรรมต่างๆ-Behavioral things
- สิ่งที่ใช้ในการจัดกลุ่ม-Grouping things
- สิ่งที่ใช้คำอธิบายประกอบ-Annotational things

สิ่งต่าง ๆ ที่เกี่ยวข้อง(*Things*)

สิ่งที่เป็นโครงสร้าง-*Structural things*



## สิ่งที่เป็น โครงสร้าง-*Structural things*

---

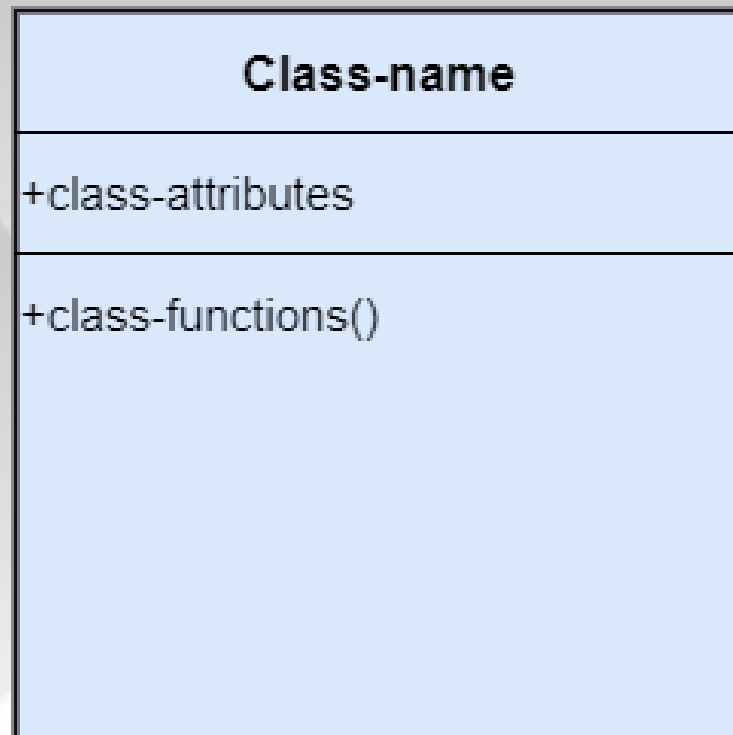
สิ่งที่แสดงถึงพฤติกรรมต่าง ๆ ของแบบจำลอง เรียกว่า สิ่งที่เป็น โครงสร้าง ทำหน้าที่แสดงองค์ประกอบทางกายภาพและแนวคิดของระบบ ซึ่งรวมถึงคลาส อ็อบเจกต์ อินเทอร์เฟซ โหนด การทำงานร่วมกัน ส่วนประกอบ และกรณีการใช้งานต่าง ๆ





## คลาส-Class

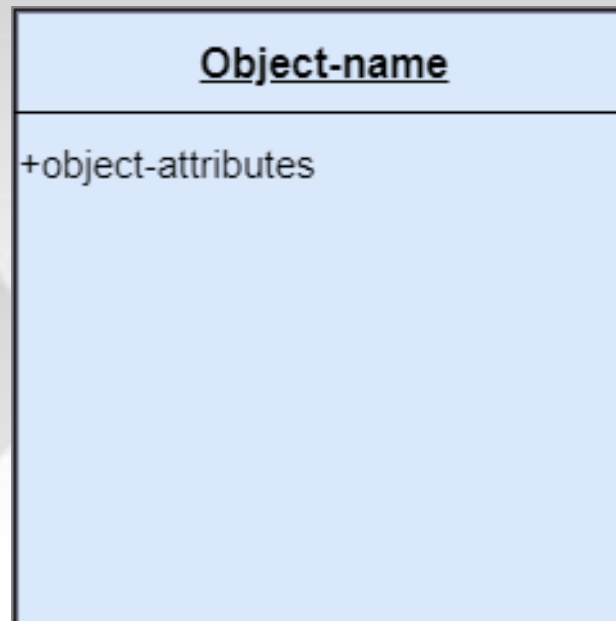
คลาส-class : คลาสคือชุดของสิ่งที่เหมือนกัน ซึ่งประกอบด้วยข้อมูลหรือตัวแปร และฟังก์ชันหรือเมธอดหรือ มีสัญลักษณ์ดังนี้





## อ็อบเจกต์-Object

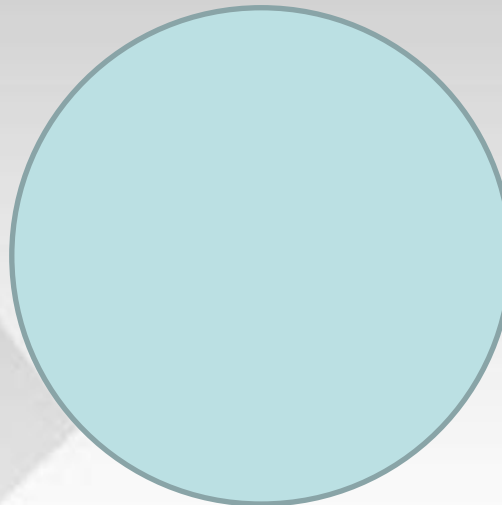
**อ็อบเจกต์-Object:** สิ่งที่อยู่ภายใต้การควบคุมและหน้าที่ของระบบ สัญลักษณ์ของอ็อบเจกต์นั้นคล้ายกับของคลาส ข้อแตกต่างเพียงอย่างเดียวคือชื่ออ็อบเจกต์จะถูกขีดเส้นใต้เสมอและระบุสัญลักษณ์ดังนี้





## อินเทอร์เฟซ-interface

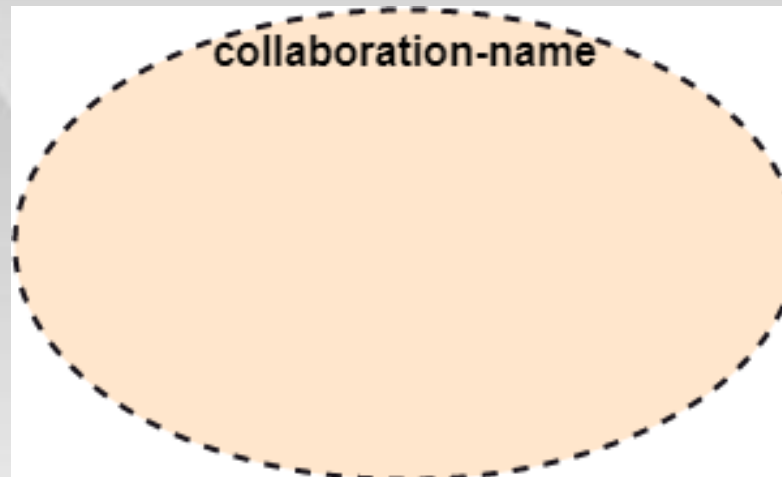
อินเทอร์เฟซ-interface : เซตของการดำเนินการหรือ operation ที่อธิบายฟังก์ชันการทำงานของคลาส ซึ่ง Operation เหล่านี้จะถูกเรียกใช้ทุกครั้งที่มีการใช้งานอินเทอร์เฟซ





## การทำงานร่วมกัน- *Collaboration*

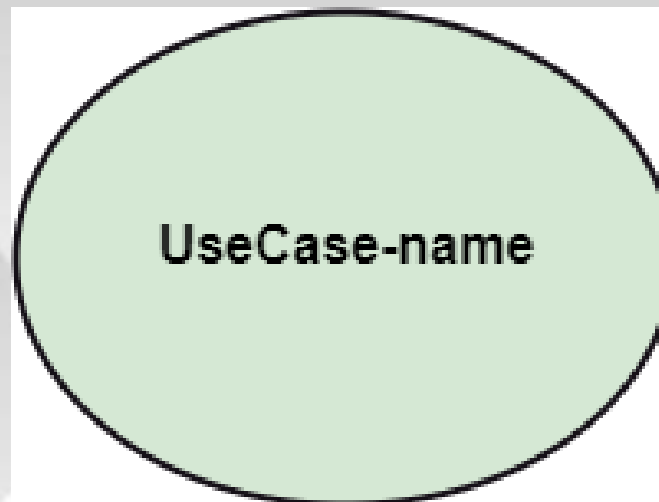
การทำงานร่วมกัน- Collaboration : แสดงถึงการปฏิสัมพันธ์ระหว่างกัน เพื่อให้บรรลุเป้าหมายในการทำงาน มันเป็นสัญลักษณ์ของวงรีเส้นประที่มีชื่อเขียนอยู่ข้างใน





## Use case

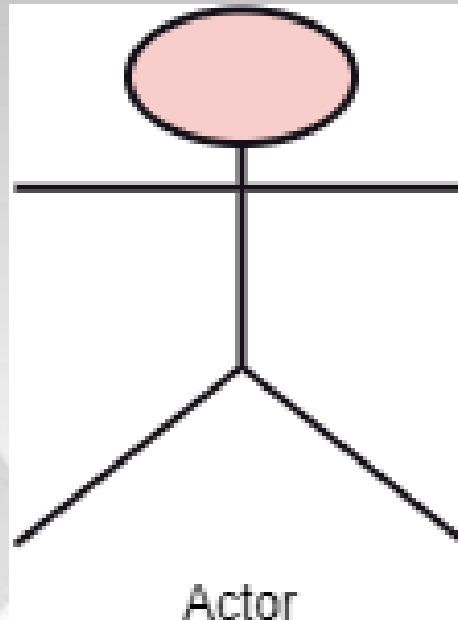
Use case: Use case เป็นแนวคิดหลักของการสร้างแบบจำลองเชิงอ็อบเจกต์ Use Case จะแสดงกระบวนการต่าง ๆ ในระบบ





## ผู้เกี่ยวข้อง-Actor

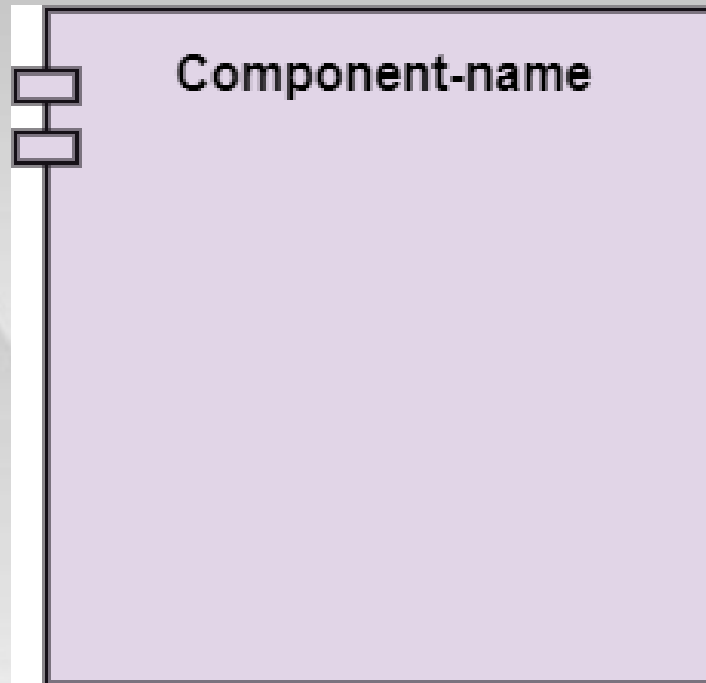
ผู้เกี่ยวข้อง-Actor: อยู่ภายใต้ Use Case Diagram เป็นอ็อบเจกต์ที่โต้ตอบกับระบบ เช่น ผู้ใช้





## คอมโพเนนต์-Component

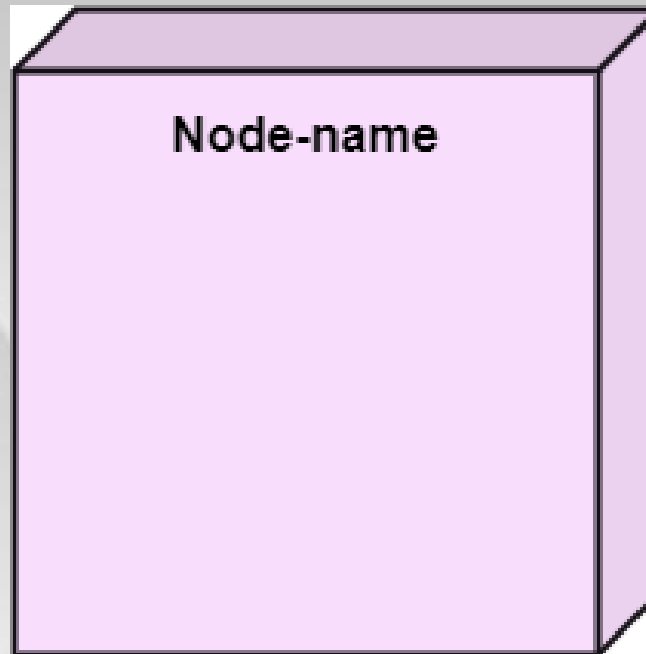
คอมโพเนนต์-Component: แสดงถึงส่วนทางกายภาพหรือส่วนประกอบต่าง ๆ  
ของระบบ





## โหนด-Node

โหนด-Node : องค์ประกอบทางกายภาพที่มีอยู่ ณ รั้นไทม์ หรือจุดที่สามารถรันระบบได้





สิ่งต่าง ๆ ที่เกี่ยวข้อง(*Things*)

-สิ่งที่เป็นพฤติกรรมต่างๆ-*Behavioral things*



## สิ่งที่เป็นพฤติกรรมต่างๆ-*Behavioral things*

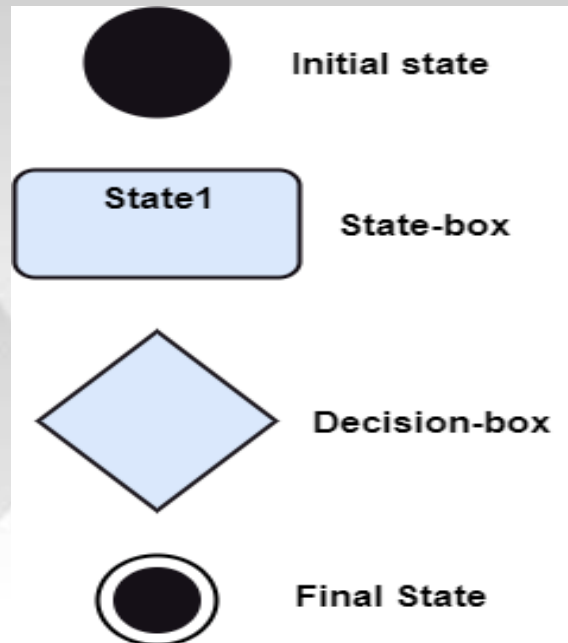
---

สิ่งที่เป็นพฤติกรรมต่างๆ-*Behavioral things* : เป็นคำกริยาที่ครอบคลุมส่วนใดนามิกของแบบจำลอง มันแสดงให้เห็นพฤติกรรมของระบบ เกี่ยวข้องกับ state machine แผนภาพกิจกรรม แผนภาพการโต้ตอบ สิ่งที่ใช้ในการจัดกลุ่ม-*Grouping things* สิ่งที่ใช้คำอธิบายประกอบ-*Annotational things*



## *State Machine*

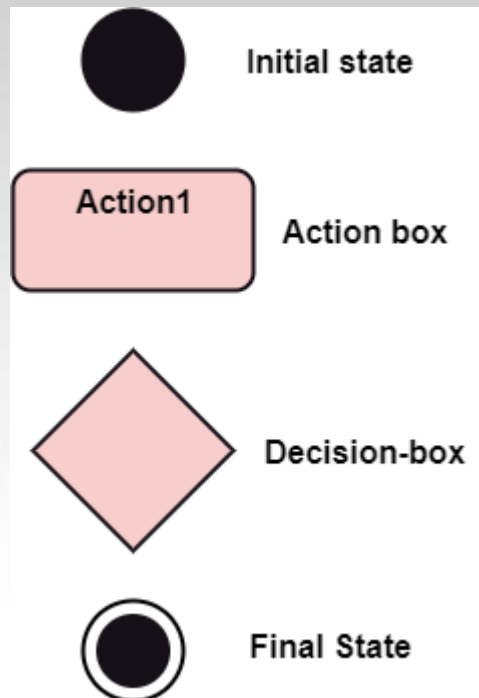
State Machine: กำหนดลำดับของสถานะต่าง ๆ ที่องค์ประกอบต่าง ๆ ของระบบต้องผ่านในวงจรการพัฒนาซอฟต์แวร์ โดย State Machine จะเก็บบันทึกสถานะที่แตกต่างกันหลายประการขององค์ประกอบของระบบ





## แผนภาพกิจกรรม- *Activity Diagram*

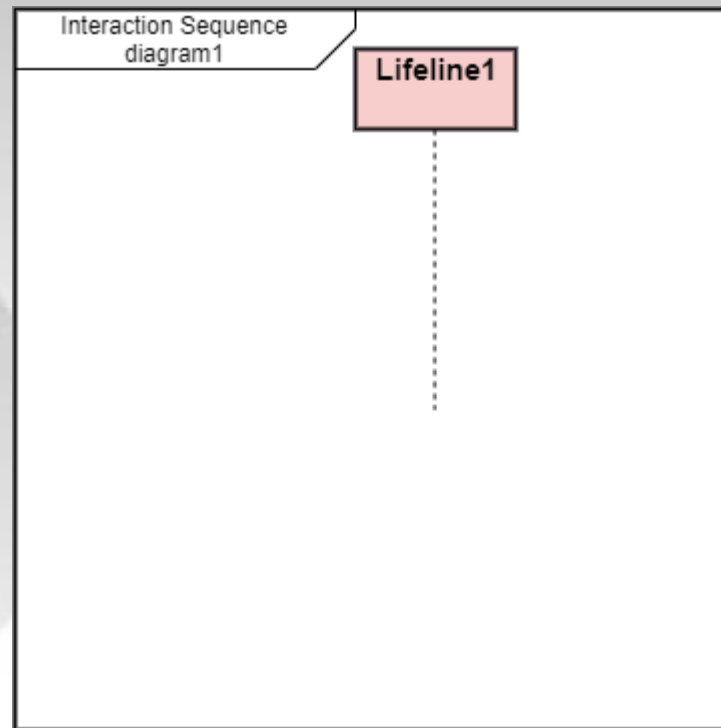
แผนภาพกิจกรรม- Activity Diagram : แสดงภาพกิจกรรมทั้งหมดที่ทำโดยส่วนต่างๆ ของระบบ Activity Diagram ประกอบด้วยสถานะเริ่มต้น สถานะสุดท้าย การตัดสินใจ การดำเนินการกิจกรรม และหมายเหตุการดำเนินการ





## *Interaction Diagram*

Interaction Diagram: ใช้เพื่อจินตนาการถึงการปฏิสัมพันธ์ระหว่างส่วนประกอบต่างๆ ในระบบ



สิ่งต่าง ๆ ที่เกี่ยวข้อง(*Things*)

-สิ่งที่ใช้ในการจัดกลุ่ม-*Grouping things*



## สิ่งที่ใช้ในการจัดกลุ่ม-*Grouping things*

---

เป็นวิธีการที่เชื่อมโยงองค์ประกอบของแบบจำลอง UML เข้าด้วยกัน ใน UML  
แพ็คเกจเป็นสิ่งเดียวที่ใช้สำหรับการจัดกลุ่ม



## *Package*

Package: Package เป็นสิ่งเดียวที่มีให้สำหรับการจัดกลุ่มพฤติกรรมและโครงสร้าง





สิ่งต่าง ๆ ที่เกี่ยวข้อง(*Things*)

- สิ่งที่ใช้คำอธิบายประกอบ-*Annotational things*



## คำอธิบายประกอบสิ่งต่าง ๆ - *Annotational things*

---

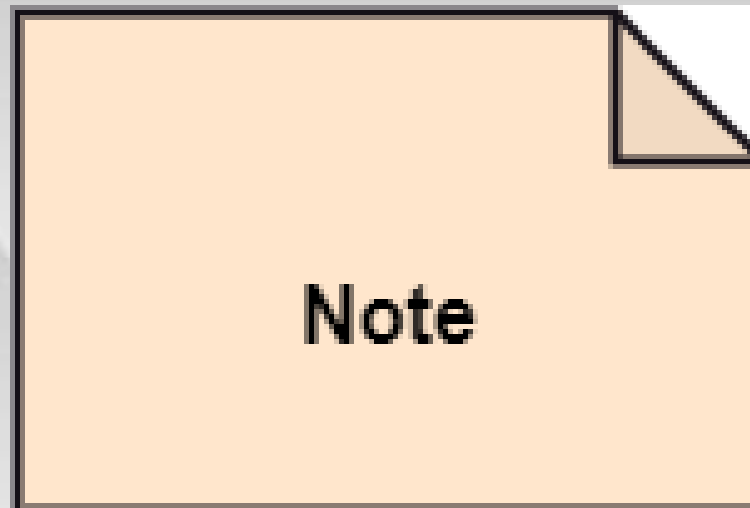
คำอธิบายประกอบสิ่งต่าง ๆ - *Annotational things*

เป็นกลไกที่รวบรวมข้อสังเกต คำอธิบาย และความคิดเห็นขององค์ประกอบ



## หมายเหตุ-note

หมายเหตุ-note : ใช้เพื่อแนบข้อจำกัด ความคิดเห็น และกฎเกณฑ์ ที่เกี่ยวกับองค์ประกอบต่าง ๆ ในแบบจำลอง เป็นกระดาษโน้ตชนิดหนึ่งสีเหลือง



ความสัมพันธ์ (*Relationships*)



## ความสัมพันธ์ (Relationships)

ความสัมพันธ์ (Relationships) จะแสดงให้เห็นความเชื่อมโยงที่มีความหมายระหว่างสิ่งต่าง ๆ ในระบบ โดยจะแสดงความสัมพันธ์ระหว่างองค์ประกอบต่าง ๆ ที่เกี่ยวข้องกับระบบและกำหนดกระบวนการทำงานที่เกี่ยวข้อง ความสัมพันธ์มีทั้งหมดสี่ประเภทดังนี้:

- การพึ่งพาอาศัยกัน-Dependency
- การเชื่อมโยง- Association
- ลักษณะทั่วไป-Generalization
- การตระหนักรู้- Realization



## การพึ่งพาอาศัยกัน-Dependency

การพึ่งพาอาศัยกัน-Dependency: การพึ่งพาอาศัยกันเป็นความสัมพันธ์ประเภทหนึ่ง ที่การเปลี่ยนแปลงในองค์ประกอบเป้าหมายหรือองค์ประกอบปลายทาง ส่งผลต่อองค์ประกอบต้นทาง หรือพูดง่ายๆ ว่าองค์ประกอบต้นทางขึ้นอยู่กับองค์ประกอบเป้าหมายหรือองค์ประกอบปลายทาง เป็นหนึ่งในสัญลักษณ์ที่สำคัญที่สุดใน UML มันแสดงให้เห็นการพึ่งพาจากเอนทิตีหนึ่งไปยังอีกเอนทิตีหนึ่งแสดงด้วยเส้นประตามด้วยลูกศรด้านหนึ่งดังแสดงด้านล่าง

**- - - - Dependency - - ->**



## การเชื่อมโยง- Association

การเชื่อมโยง- Association : ชุดของลิงก์ที่เชื่อมโยงองค์ประกอบต่าง ๆ ที่เกี่ยวข้องกับระบบ มันบอกว่ามีองค์ประกอบกี่องค์ประกอบจริง ๆ ที่มีส่วนร่วมในการสร้างความสัมพันธ์นั้น โดยจะแสดงด้วยเส้นประที่มีหัวลูกศรทั้งสองด้านเพื่ออธิบายความสัมพันธ์กับองค์ประกอบทั้งสองด้าน

◀ - - Association - - - ▶



## ลักษณะทั่วไป-Generalization

ลักษณะทั่วไป-Generalization : มันแสดงให้เห็นความสัมพันธ์ระหว่างคลาสพารেন্টหรือซูเปอร์คลาส และคลาสลูกหรือคลาสรายย่อย ใช้เพื่ออธิบายแนวคิดเรื่องสิ่งที่ถูกถ่ายโอนไปสู่ลูกหลาน เขียนแทนด้วยเส้นตรงแล้วตามด้วยหัวลูกศรเปล่าที่ด้านหนึ่ง

—Generalization—▶





## การตระหนักรู้- Realization

การตระหนักรู้- Realization : เป็นความสัมพันธ์เชิงความหมายระหว่างสองสิ่ง โดยที่สิ่งหนึ่งกำหนดพฤติกรรมที่จะดำเนินการ และอีกสิ่งหนึ่งนำพฤติกรรมดังกล่าวไปใช้ เขียนแทนด้วยเส้นประที่มีหัวลูกศรว่างอยู่ด้านหนึ่ง

- - - - - Realization - - - 

# ไคอะแกรม(*Diagrams*)

# UML Use Case Diagram



# UML Use Case Diagram

---

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



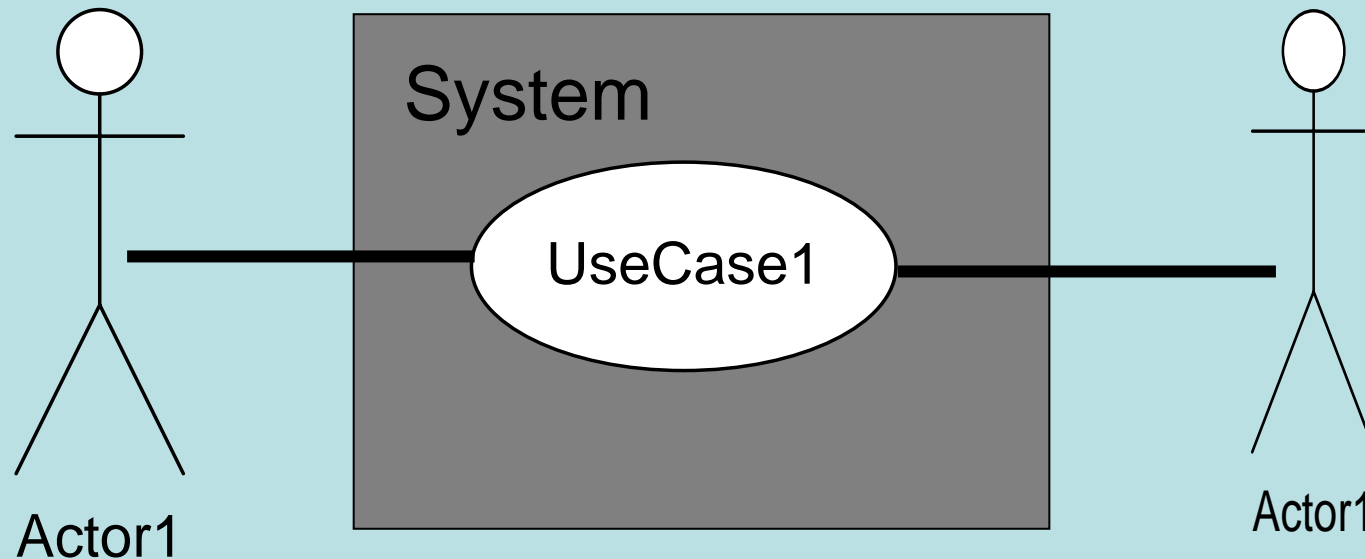
## วัตถุประสงค์ของ Use Case Diagram

วัตถุประสงค์หลักของ Use Case Diagrams คือการแสดงภาพไดนามิกของระบบ มันรวบรวมความต้องการของระบบซึ่งรวมถึงอิทธิพลทั้งภายในและภายนอก โดยจะเรียกบุคคล กรณีใช้งาน และหลายสิ่งหลายอย่างที่เราเรียกใช้ตัวแสดงและองค์ประกอบที่รับผิดชอบในการดำเนินการตามแผนภาพกรณีการใช้งาน มันแสดงให้เห็นว่าเอนทิตีจากสภาพแวดล้อมภายนอกสามารถโต้ตอบกับส่วนหนึ่งของระบบได้อย่างไร

วัตถุประสงค์ของแผนภาพกรณีการใช้งานที่ระบุด้านล่าง:


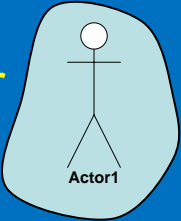




- รวบรวมความต้องการของระบบ
- แสดงให้เห็นมุมมองภายนอกของระบบ
- ตระหนักถึงปัจจัยภายในและภายนอกที่มีอิทธิพลต่อระบบ
- แสดงถึงปฏิสัมพันธ์ระหว่างผู้เกี่ยวข้องกับระบบ

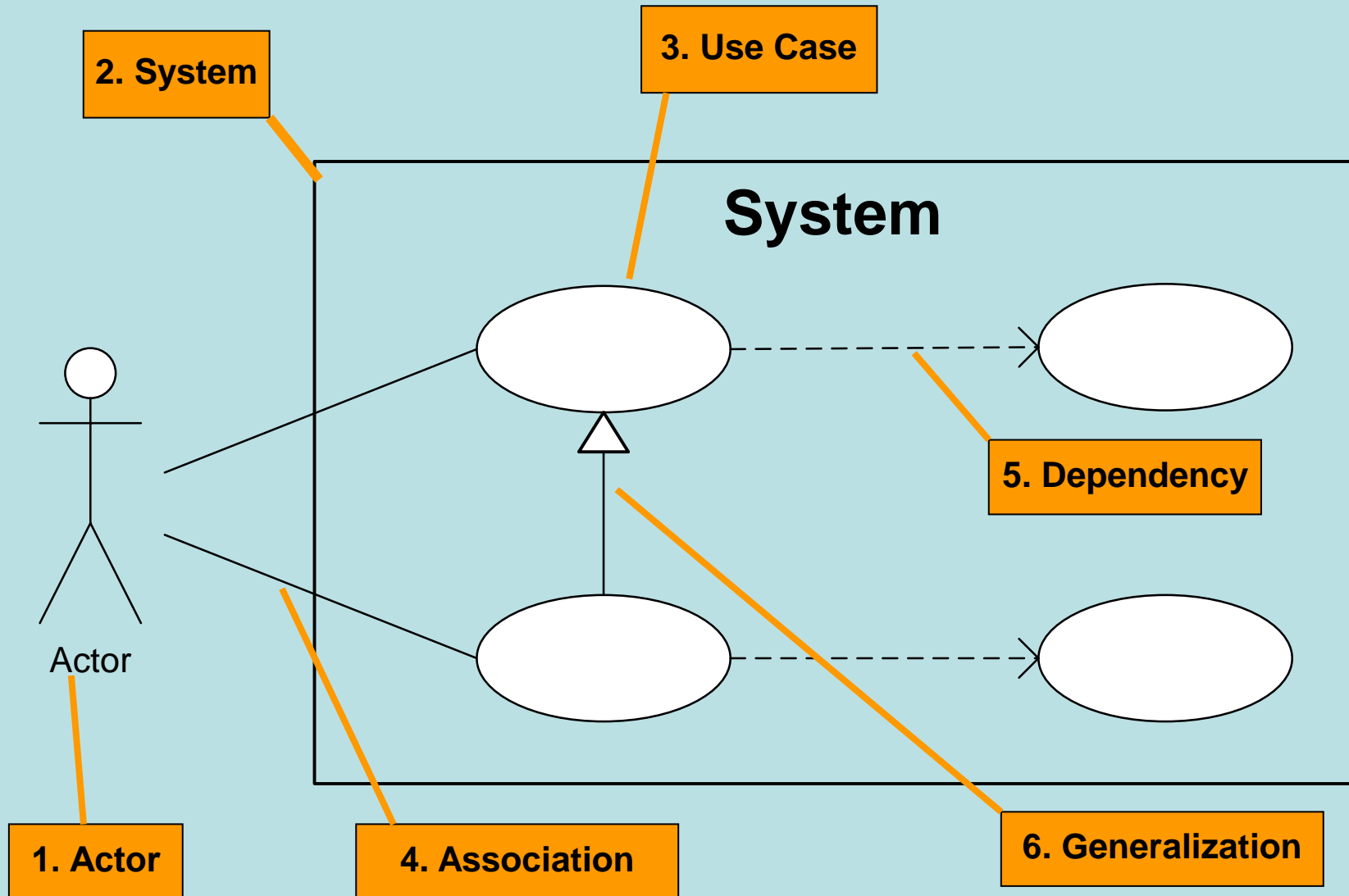
Use case diagram เป็น diagram ที่มีความสำคัญมาก เนื่องจาก use case diagram จะช่วยนักวิเคราะห์ระบบในการทำความเข้าใจ และมองภาพรวมของระบบงานที่กำลังดำเนินการศึกษาอยู่ หน้าที่หลักของ use case diagram คือการแสดงความติดต่อระหว่างผู้ใช้ กับระบบที่สร้างขึ้น



# Use Case Diagram

ประกอบด้วย 6 สัญลักษณ์ที่ใช้แสดงโครงสร้างของระบบงานที่เราจะสร้างขึ้น

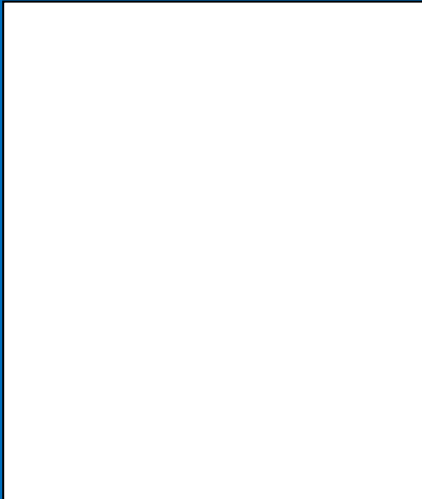
1. **System**  แสดงขอบเขตของระบบและความสัมพันธ์ในการทำงานที่มีต่อผู้ใช้
2. **Actor**  แสดงผู้ใช้งาน ระบบ อุปกรณ์ต่างๆ ที่มาเกี่ยวข้องกับระบบของท่าน
3. **Use Case**  แสดงคุณลักษณะหรือกระบวนการในระบบของท่าน ซึ่งถ้าขาดส่วนส่วนนี้แล้วจะทำให้ระบบของท่านไม่อาจทำงานได้หรือไม่สมบูรณ์
4. **Association**  แสดงการโต้ตอบระหว่าง 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





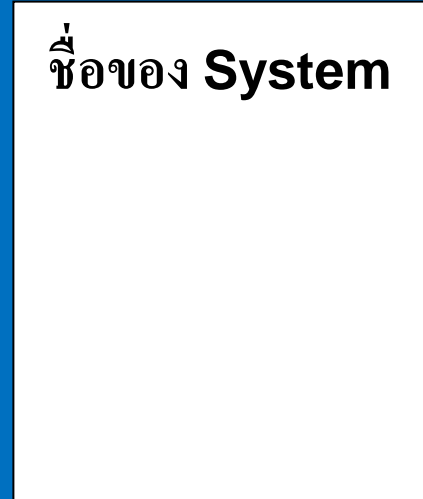
# 1. System

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

การถอนเงิน

การฝากเงิน

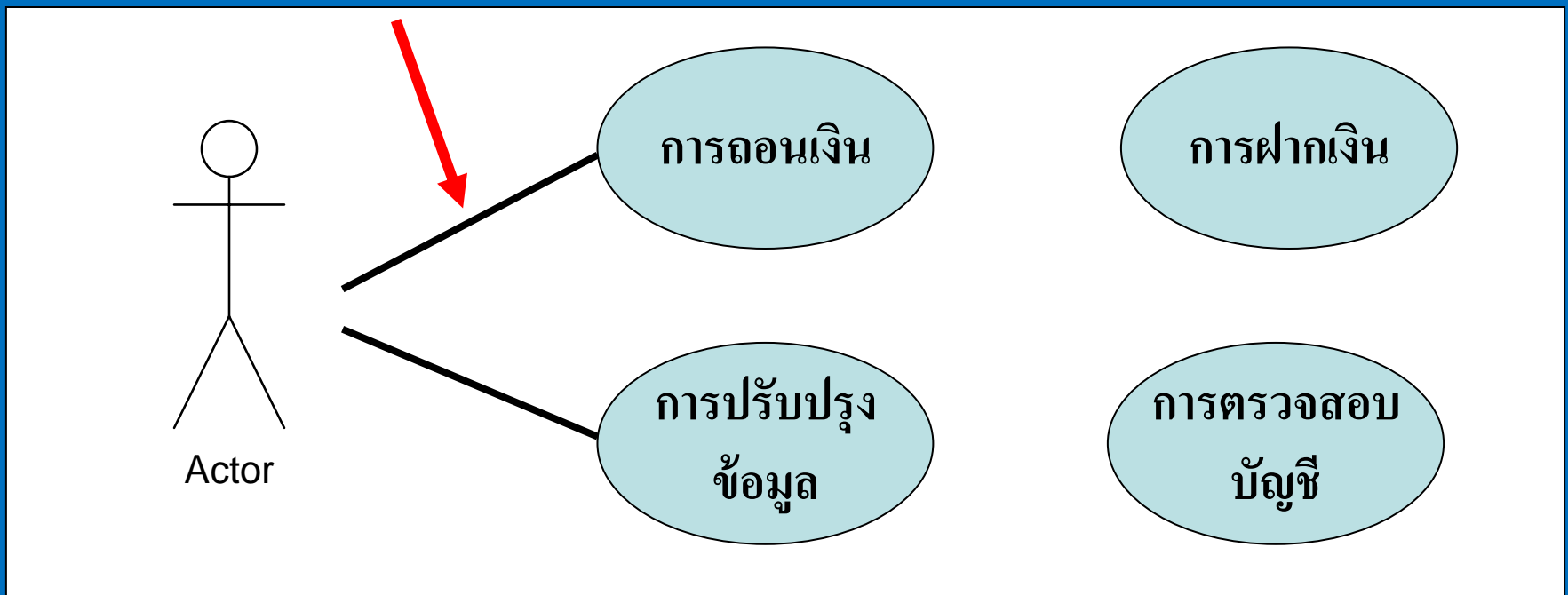
การปรับปรุง  
ข้อมูล

การตรวจสอบ  
บัญชี

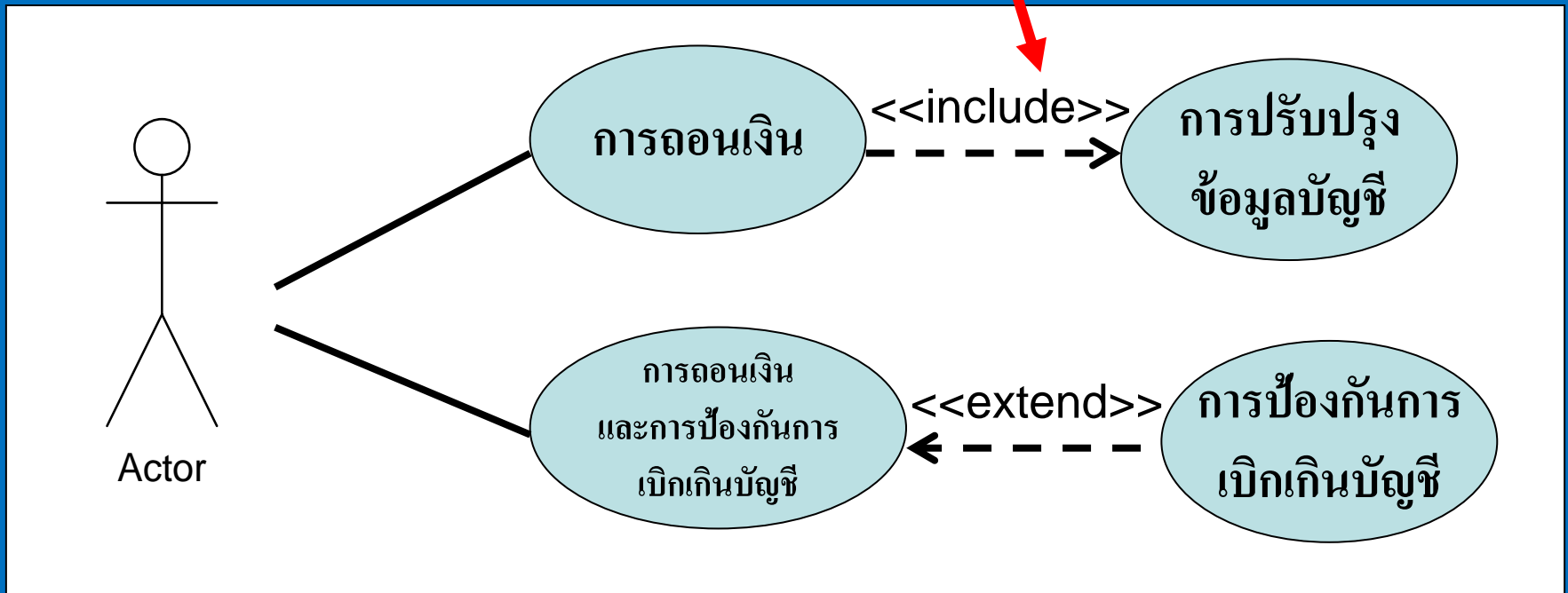
## 4. relationship

ใช้ระบุความสัมพันธ์ขององค์ประกอบต่าง ๆ ภายใน Use Case Diagram ประกอบด้วย

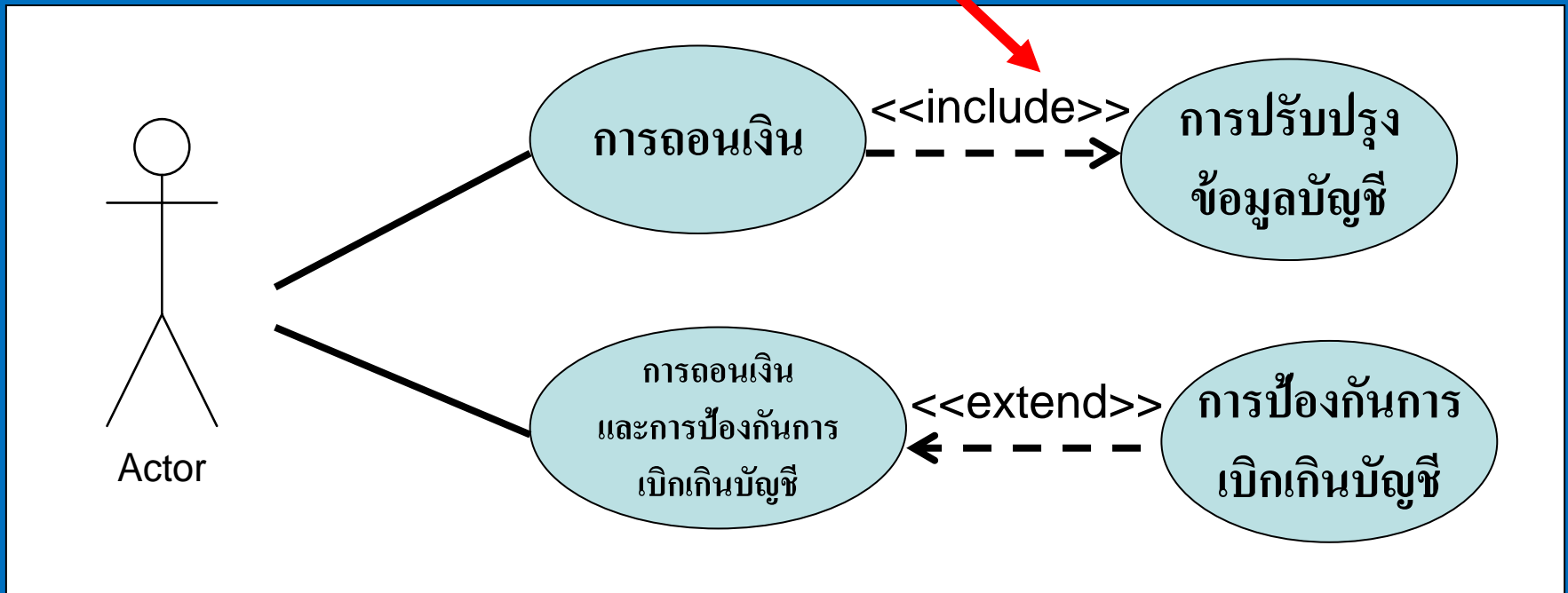
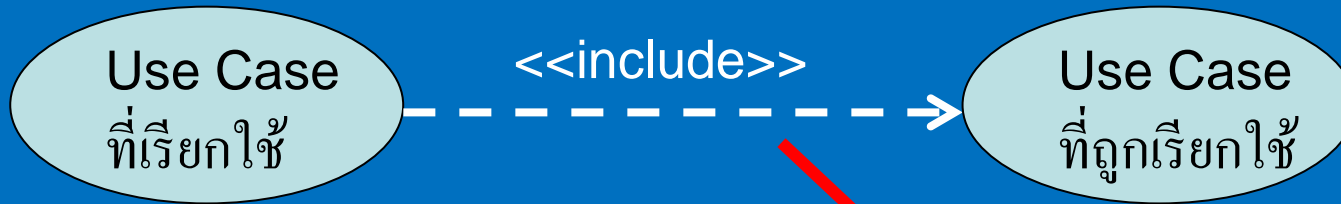
- Association notation ใช้แสดงความสัมพันธ์ระหว่าง actor กับ Use Case



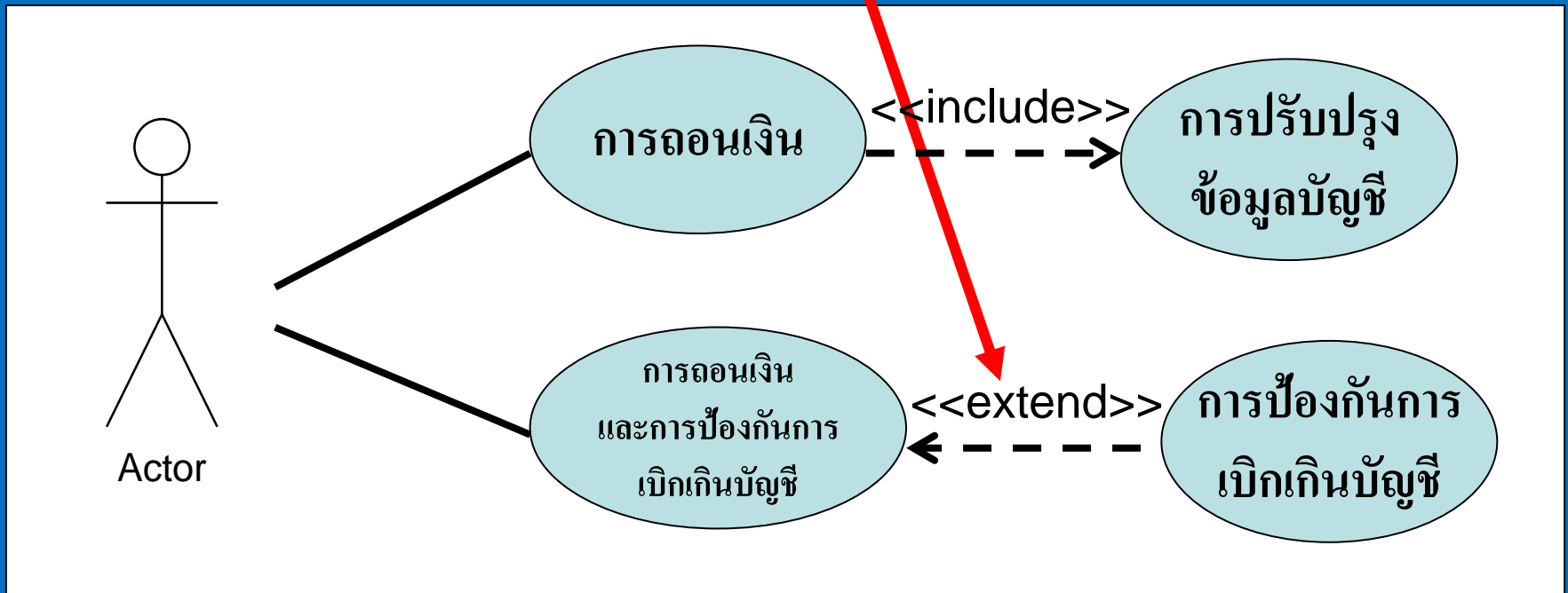
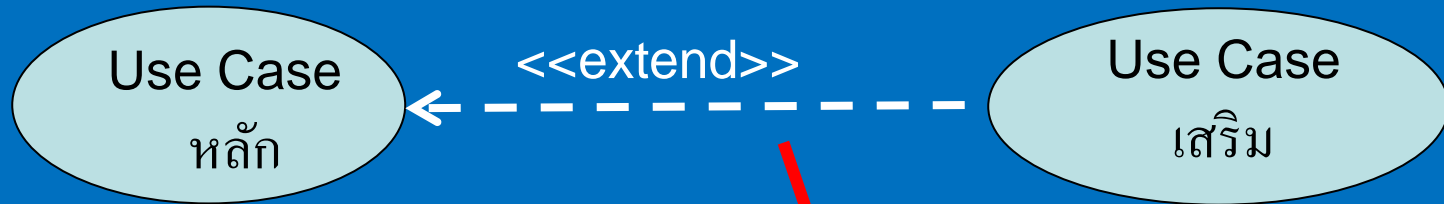
-Stereotype notation ใช้แสดงความสัมพันธ์ระหว่าง Use Case ซึ่งจะใช้  
สัญลักษณ์ <<.....>> และมีคำอธิบายใน <<...>>  
----->  
(French Quote)



-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>>  
----->

