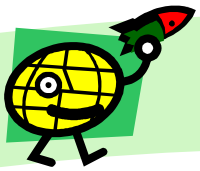


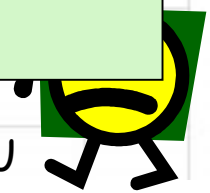
Software Design

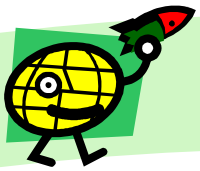
การออกแบบซอฟต์แวร์

คือกระบวนการกำหนดสถาปัตยกรรม ส่วนประกอบ ส่วนประสานและลักษณะด้านอื่นๆ ของระบบหรือส่วนประกอบของระบบ โดยการออกแบบซอฟต์แวร์ยังมีความหมายรวมถึงสิ่งที่ได้จากการออกแบบ คือ “แบบจำลองของการออกแบบ” (Design Model)

การออกแบบซอฟต์แวร์ เป็นการนำข้อกำหนดความต้องการของผู้ใช้ (ลูกค้า) มากำหนดรายละเอียดโครงสร้างภายในซอฟต์แวร์ เพื่อนำมาใช้ในการเขียนและทดสอบโปรแกรมในระยะการสร้างซอฟต์แวร์ โดยนำศาสตร์ด้าน “วิศวกรรมการออกแบบ (Design Engineering)” เพื่อให้แบบร่างมีคุณสมบัติ

- Firmness : ต้องไม่มีข้อผิดพลาด
- Commodity : ต้องตรงกับวัตถุประสงค์การใช้งาน
- Delight : ต้องทำให้ผู้ใช้รู้สึกพึงพอใจ

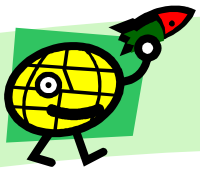




Software Design

หลักการออกแบบซอฟต์แวร์

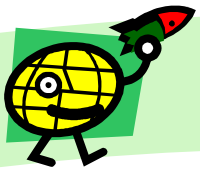
1. การออกแบบเลือกรูปแบบสถาปัตยกรรมที่ชัดเจนและมีแบบแผน และนำไปพัฒนาแบบ Evolutionary ได้ ดังนั้นต้องออกแบบให้พัฒนาและทดสอบง่าย
2. การออกแบบควรมีลักษณะเป็น Module
3. การออกแบบควรนำเสนอด้านข้อมูล สถาปัตยกรรม ส่วนประสาน และคอมโพเนนท์ที่ชัดเจน
4. ควรออกแบบคอมโพเนนท์ให้มีอิสระต่อกัน
5. ควรออกแบบให้ส่วนประสานระหว่างคอมโพเนนท์กับสภาพแวดล้อมภายนอก มีความซับซ้อนน้อยที่สุด และขึ้นต่อกันน้อยที่สุด
6. การออกแบบควรนำข้อมูลที่ได้จากการวิเคราะห์ระบบ และใช้ระเบียบวิธีปฏิบัติเดียวกัน
7. สัญลักษณ์ที่ใช้ในการออกแบบควรสื่อความหมายได้ชัดเจน และเป็นมาตรฐาน
8. งานออกแบบควรมีโครงสร้างที่ดี เพื่อให้แก้ไขที่ง่ายและใช้ต้นทุนน้อย



Software Design

แนวคิดในการออกแบบซอฟต์แวร์

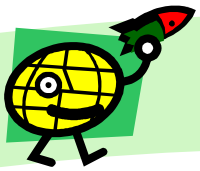
1. **คิดแบบนามธรรม (Abstraction)** : เป็นพื้นฐานการคิดอย่างหนึ่งที่ใช้แก้ปัญหาที่ช่วยลดความซับซ้อนตามระดับ โดยพิจารณาปัญหาและสร้าง Abstraction 2 ชนิด คือ Procedural Ab. (เพื่อสร้างลำดับขั้นตอน) และ Data Ab. (Object ข้อมูลที่อยู่ใน Procedural Ab.)
2. **สถาปัตยกรรม (Architecture)** : เป็นโครงสร้างทั้งหมดของซอฟต์แวร์ที่แสดงให้เห็นโครงสร้างของโปรแกรมย่อย อาทิ Structure Chart Model เป็นต้น
3. **แบบแผน (Pattern)** : แบบแผนการออกแบบ คือหลักและวิธีการแก้ไขปัญหาชนิดใดชนิดหนึ่ง ที่สามารถนำมาใช้กับปัญหาชนิดเดียวกันที่เกิดซ้ำได้ จะช่วยให้ความผลิตซอฟต์แวร์ดำเนินไปได้อย่างรวดเร็ว



Software Design

แนวคิดในการออกแบบซอฟต์แวร์

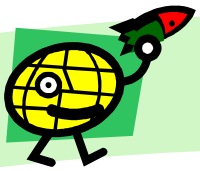
5. **การแบ่งระบบ (Modularity)** : การแบ่งซอฟต์แวร์ออกแบบส่วน ๆ ซึ่งจะถูกประสานให้ทำงานร่วมกันเพื่อแก้ปัญหาตามที่ผู้ใช้ต้องการ
6. **การซ่อนรายละเอียด (Information Hiding)** : การแบ่งระบบออกเป็นโมดูล ปัญหาที่เกิดคือความยุ่งยากในการใช้ข้อมูลร่วมกัน ดังนั้นเพื่อป้องกันการเข้าถึงข้อมูลโดยไม่จำเป็น แต่ละโมดูลจะซ่อนรายละเอียดการทำงานไว้
7. **ความเป็นอิสระต่อกันในการทำงาน (Functional Independence)** : การออกแบบให้โมดูลมีความเป็นอิสระต่อกัน จะทำให้ซ่อมบำรุงและทดสอบง่ายขึ้น และยังมีโอกาสสร้างเป็น Reusable Module ได้ง่ายขึ้น



Software Design

แนวคิดในการออกแบบซอฟต์แวร์

9. **การกลั่นกรอง (Refinement)** : เป็นกลยุทธ์ด้านการออกแบบที่ชื่อ Top-Down Design โดยบรรยายรายละเอียดของแต่ละฟังก์ชั่นเป็นลำดับขั้น
10. **การปรับโครงสร้างการออกแบบ (Refactoring)** : เป็นเทคนิคในการปรับโครงสร้างการออกแบบภายในของคอมโพเนนท์ (หรือซอฟต์แวร์) เริ่มต้นจากการนำงานออกแบบเดิมมาพิจารณาถึงความซ้ำซ้อน ส่วนประกอบที่ไม่ได้ถูกใช้งาน อัลกอริทึมไม่มีประสิทธิภาพหรือไม่จำเป็น ตลอดจนโครงสร้างข้อมูลไม่เหมาะสม หรือผิดพลาด โดยนำมาแก้ไขให้มีประสิทธิภาพและถูกต้องมากขึ้น

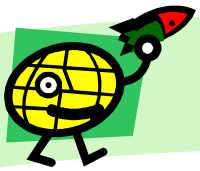


Analysis and Design Phase

ในการวิเคราะห์ความต้องการ (ในบทที่ผ่านมา) ทีมงานจำเป็นต้องสร้างแบบจำลองความต้องการขึ้นมา เพื่อใช้สื่อสารกับบุคคลอื่นไม่ว่าจะเป็นผู้ใช้ระบบ ลูกค้า หรือระหว่างทีมงานด้วยกันเองให้เกิดความเข้าที่ถูกต้องตรงกัน แทนการสื่อสารด้วยข้อความหรือคำพูด โดยแบบจำลองที่ใช้ระบบมีหลายชนิดแตกต่างกัน แต่ละชนิดใช้จำลองระบบในด้านต่าง ๆ จึงได้แบ่งเป็น 2 แนวทางคือ เชิงโครงสร้าง และเชิงวัตถุ

ความสำคัญของแบบจำลอง

แบบจำลอง (Model) คือ สัญลักษณ์ที่ใช้จำลองข้อเท็จจริงต่าง ๆ ที่เกิดขึ้นในระบบ แบบจำลองจึงประกอบไปด้วยแผนภาพชนิดต่าง ๆ เพื่อแสดงให้เห็นแต่ละมุมมองของระบบ



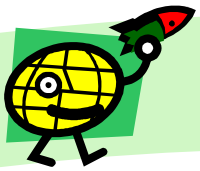
Analysis and Design Phase

แบบจำลองตามแนวทางเชิงโครงสร้าง

พิจารณากระบวนการทำงานกับข้อมูลของระบบแยกออกจากกัน ดังนั้นจึงแบ่งออกเป็น 2 ชนิด ได้แก่

1. **แบบจำลองกระบวนการ (Process Model)**
   ใช้จำลองขั้นตอนการทำงานของระบบ คือ แผนภาพกระแสข้อมูล (Data Flow Diagram : DFD)

2. **แบบจำลองข้อมูล (Data Model)**
   ใช้จำลองโครงสร้างข้อมูลทั้งหมดในระบบ คือ แผนภาพแสดงความสัมพันธ์ระหว่างข้อมูล (Entity Relationship Diagram : ERD)



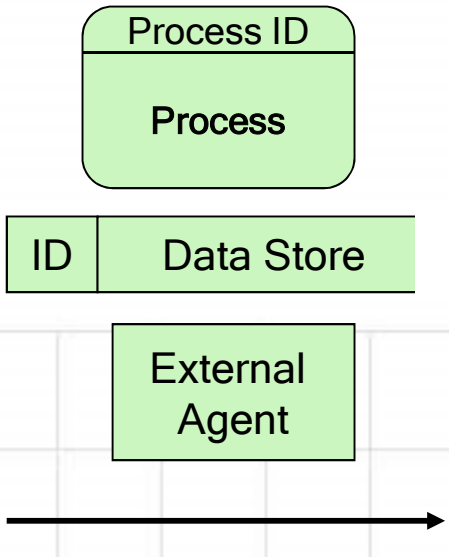
Analysis and Design Phase

แบบจำลองตามแนวทางเชิงโครงสร้าง

- **แผนภาพกระแสข้อมูล (Data Flow Diagram : DFD)**

  หมายถึง แผนภาพที่แสดงให้เห็นถึงทิศทางการไหลของข้อมูลที่มีอยู่ในระบบ จากกระบวนการหนึ่ง -> กระบวนการหนึ่ง หรือไปยังแหล่งจัดเก็บข้อมูล หรือผู้มีส่วนเกี่ยวข้องนอกระบบ

  สัญลักษณ์ของ DFD



กระบวนการ (Process)

แหล่งจัดเก็บข้อมูล (Data Store)

ผู้มีส่วนเกี่ยวข้องนอกระบบ (External Agent)

ทิศทางการไหล (Flow Direction)

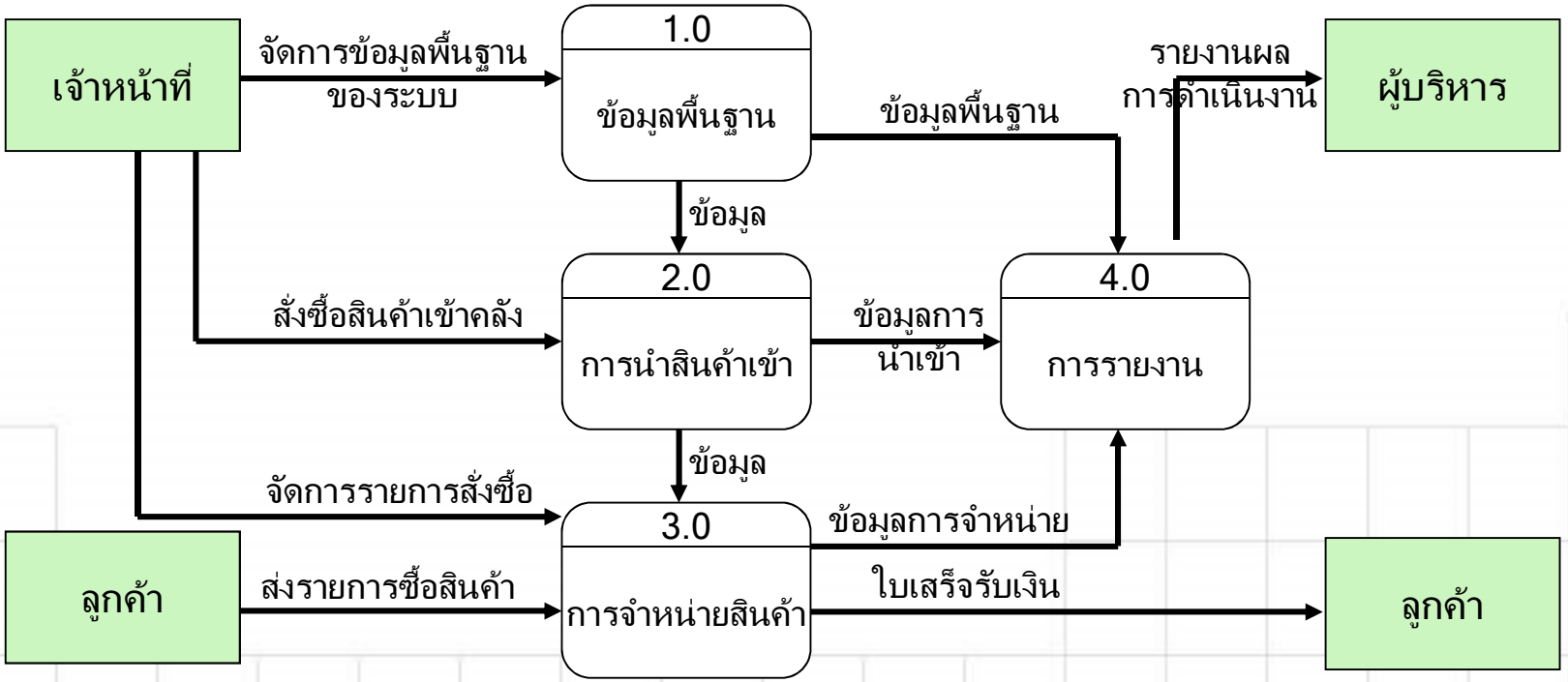
แบบจำลองตามแนวทางเชิงโครงสร้าง

แผนภาพกระแสข้อมูล (Data Flow Diagram : DFD)

DFD Context Level : ระบบสหกรณ์ร้านค้า มหาวิทยาลัยราชภัฏลำปาง

แบบจำลองตามแนวทางเชิงโครงสร้าง

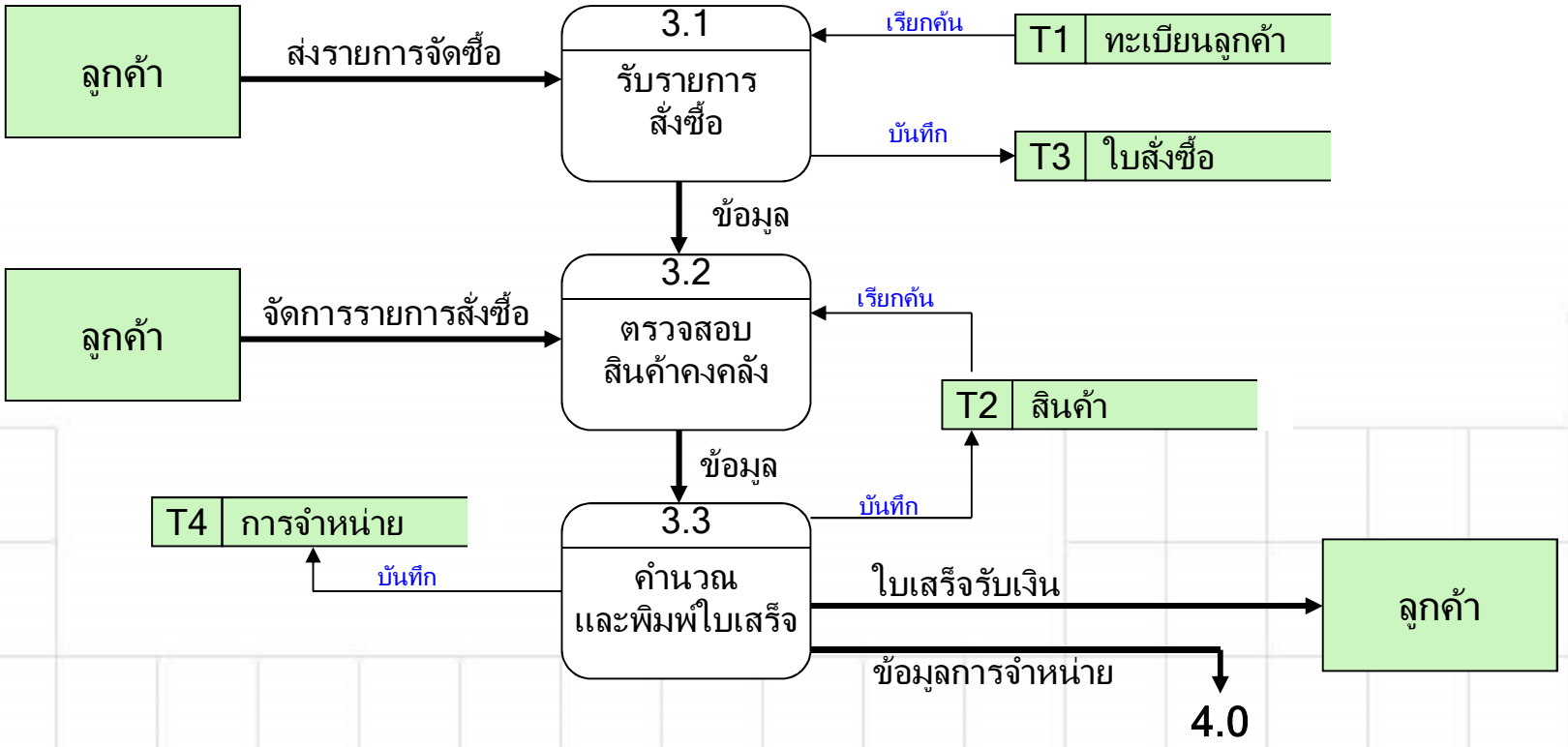
แผนภาพกระแสข้อมูล (Data Flow Diagram : DFD)



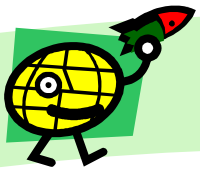
DFD Level 0 : ระบบสหกรณ์ร้านค้า มหาวิทยาลัยราชภัฏลำปาง

แบบจำลองตามแนวทางเชิงโครงสร้าง

แผนภาพกระแสข้อมูล (Data Flow Diagram : DFD)



DFD Level 1 ของ Process 1.0 ข้อมูลพื้นฐาน ระบบสหกรณ์ร้านค้าฯ



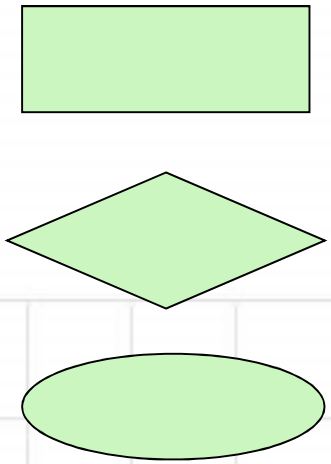
Analysis and Design Phase

แบบจำลองตามแนวทางเชิงโครงสร้าง

- **แผนภาพแสดงความสัมพันธ์ระหว่างข้อมูล (Entity Relationship Diagram : ERD)**

  หมายถึง แผนภาพที่ใช้เป็นเครื่องมือสำหรับจำลองข้อมูล ซึ่งจะประกอบไปด้วย Entity , Relationship และ Attribute

  สัญลักษณ์ของ ERD



กลุ่มข้อมูล (Entity)

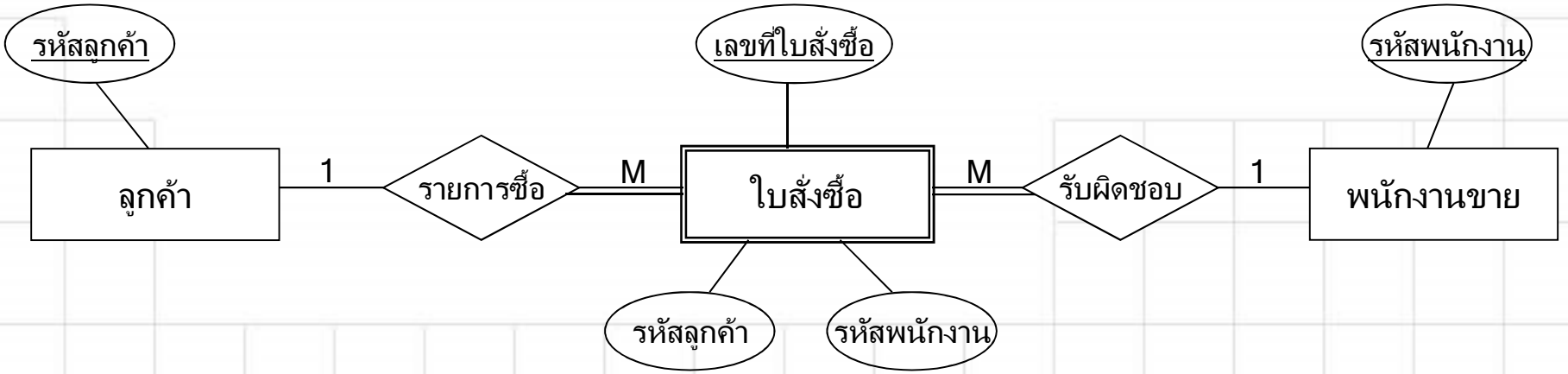
ความสัมพันธ์ระหว่างข้อมูล (Relationship)

คุณสมบัติ/รายละเอียดของกลุ่มข้อมูลหรือความสัมพันธ์ (Attribute)

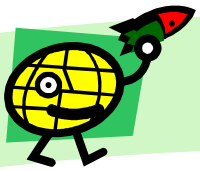
แบบจำลองตามแนวทางเชิงโครงสร้าง

แผนภาพกระแสข้อมูล (Data Flow Diagram : DFD)

ลูกค้า (รหัสลูกค้า, ชื่อ-สกุลลูกค้า, ที่อยู่ลูกค้า, จำนวนเงินเครดิต)
พนักงานขาย (รหัสพนักงาน, ชื่อ-สกุลพนักงาน, แผนกที่ทำงาน, เงินเดือน, ที่อยู่)
ใบสั่งซื้อ (เลขที่ใบสั่งซื้อ,วันที่สั่งซื้อ, รหัสลูกค้า, รายการที่สั่งซื้อ, จำนวนเงินทั้งสิ้น, รหัสพนักงาน)



ERD ของ Entity ลูกค้า ใบสั่งซื้อ และพนักงานขาย



Analysis and Design Phase

แบบจำลองตามแนวทางเชิงวัตถุ

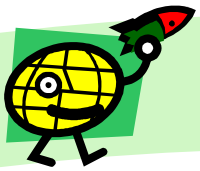
เนื่องจากแนวทางเชิงโครงสร้าง แยกกระบวนการทำงานและข้อมูลออกจากกัน ทำให้อาจเกิดความสับสนว่า**ควรพิจารณาสิ่งใดก่อน!!** แต่แนวทางเชิงวัตถุ จะพิจารณาทุก ๆ สิ่งในระบบที่สนใจเป็นวัตถุ (Object) เข้าด้วยกัน และพิจารณาไปพร้อม ๆ กัน$_1$

ในระบบจึงประกอบด้วย **อ็อบเจ็ค (Object)** จำนวนมากที่สัมพันธ์กันทำงานร่วมกัน ทำให้เกิดเป็นการทำงานของระบบ หากอ็อบเจ็คใดที่มี**คุณลักษณะ (Attribute)** และ **พฤติกรรม (Method/Behavior/Operation)** เหมือนกัน จะถูกจัดให้ใน **คลาส (Class)** เดียวกัน$_2$ เช่น

- อ็อบเจ็ค “ผู้ชาย (Man)” และ “ผู้หญิง (Women)”
- มีลักษณะ หู ตา จมูก ปาก แขน ขา เหมือนกัน
- จึงจัดให้อยู่ในคลาสเดียวกันคือ คน (Human) เป็นต้น

ปัจจุบันแบบจำลองของระบบที่สร้างขึ้นตามแนวทางเชิงวัตถุ มีหลากหลายชนิด จึงต้องสร้างมาตรฐานเพื่อเป็นแนวทางการวิเคราะห์และออกแบบเชิงวัตถุและให้เป็นภาษารูปภาพที่ใช้สร้างแบบจำลองเชิงวัตถุมาตรฐาน เรียกว่า

“UML (Unified Modeling Language)”



Analysis and Design Phase

แบบจำลองตามแนวทางเชิงวัตถุ

สำหรับแผนภาพที่จะใช้จำลองระบบเชิงวัตถุของภาษา UML ถูกแบ่งออกเป็น 2 กลุ่ม รวม 9 แผนภาพ ดังนี้

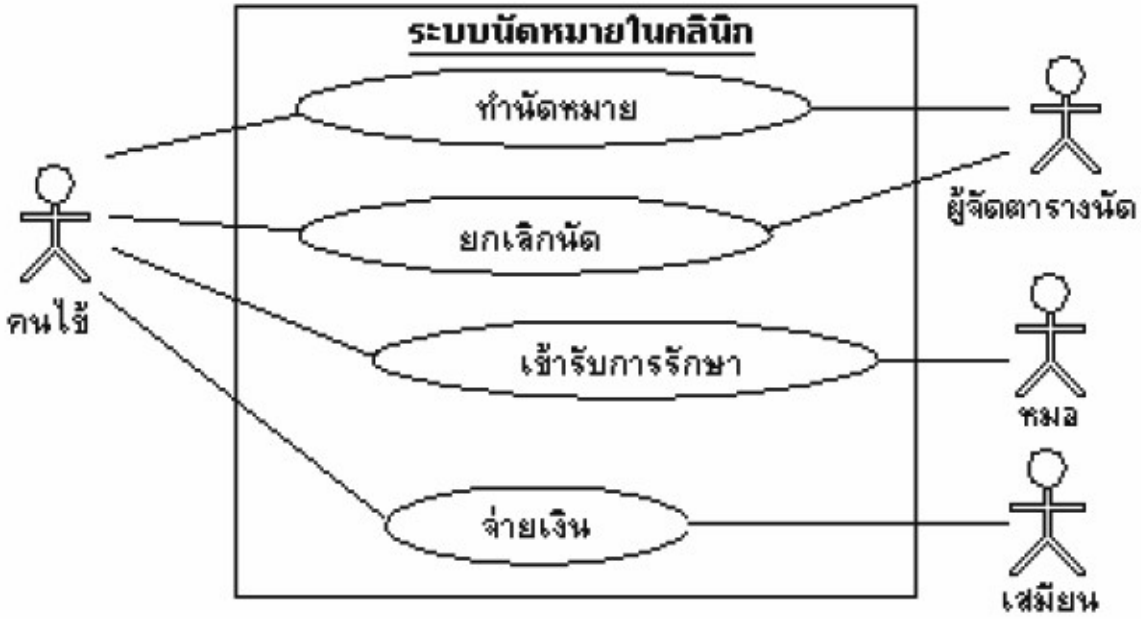
1. **Structural Diagram** : กลุ่มแผนภาพที่แสดงให้เห็นโครงสร้างเชิงสถิต (Static) ของระบบ คือโครงสร้างที่ไม่มีการเคลื่อนไหวของข้อมูลหรือเหตุการณ์
   - Class Diagram
   - Component Diagram
   - Object Diagram
   - Deployment Diagram

2. **Behavioral Diagram** : กลุ่มแผนภาพที่แสดงให้เห็นกิจกรรมของระบบ (Dynamic) คือ แสดงให้เห็นการเคลื่อนไหวของข้อมูลหรือพฤติกรรมเพื่อแสดงให้เห็นความสามารถของระบบในการดำเนินในหน้าที่บางอย่างใด
   - Use Case Diagram
   - Collaboration Diagram
   - Activity Diagram
   - Sequence Diagram
   - State Diagram

แบบจำลองตามแนวทางเชิงวัตถุ

Use Case Diagram

ในขั้นตอนของการวิเคราะห์ความต้องการของระบบ มักใช้ Use Case Diagram เป็นเครื่องมือจำลองหน้าที่ของระบบที่ผู้ใช้ต้องการ โดยแผนภาพจะแสดงว่าระบบทำงานหรือหน้าที่ใดบ้าง ทราบถึงผู้ใช้งานในแต่ละส่วนของระบบ รวมถึงแสดงปฏิสัมพันธ์ระหว่างระบบและสิ่งที่อยู่นอกระบบงาน

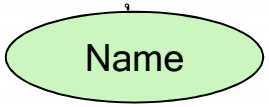


แบบจำลองตามแนวทางเชิงวัตถุ 2

Use Case Diagram

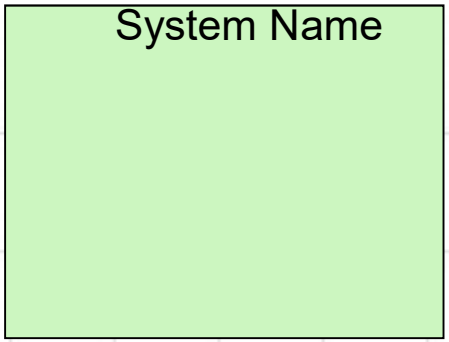
การสร้าง Use Case Diagram มุ่งเน้นไปที่ Functional Requirement ซึ่งเป็นการวิเคราะห์ความต้องการของระบบในด้านความสามารถหรือหน้าที่ที่ระบบต้องกระทำ หลังจากนั้น จึงนำ Use Case Diagram ไปเป็นพื้นฐานการสร้างแผนภาพ (Diagram) ชนิดอื่นๆ (ซึ่งเน้น Nonfunctional Requirement) จนแสดงให้เห็นโครงสร้างของระบบงานใหม่จนครบทุกส่วน ประกอบด้วยสัญลักษณ์

- Use Case



- Actor

- System Boundary

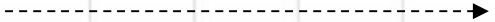


- Relationship (เส้น + Stereotype <<..>>)

1. Association :Use Case - Actor

2. Generalization :Use Case-Use Case

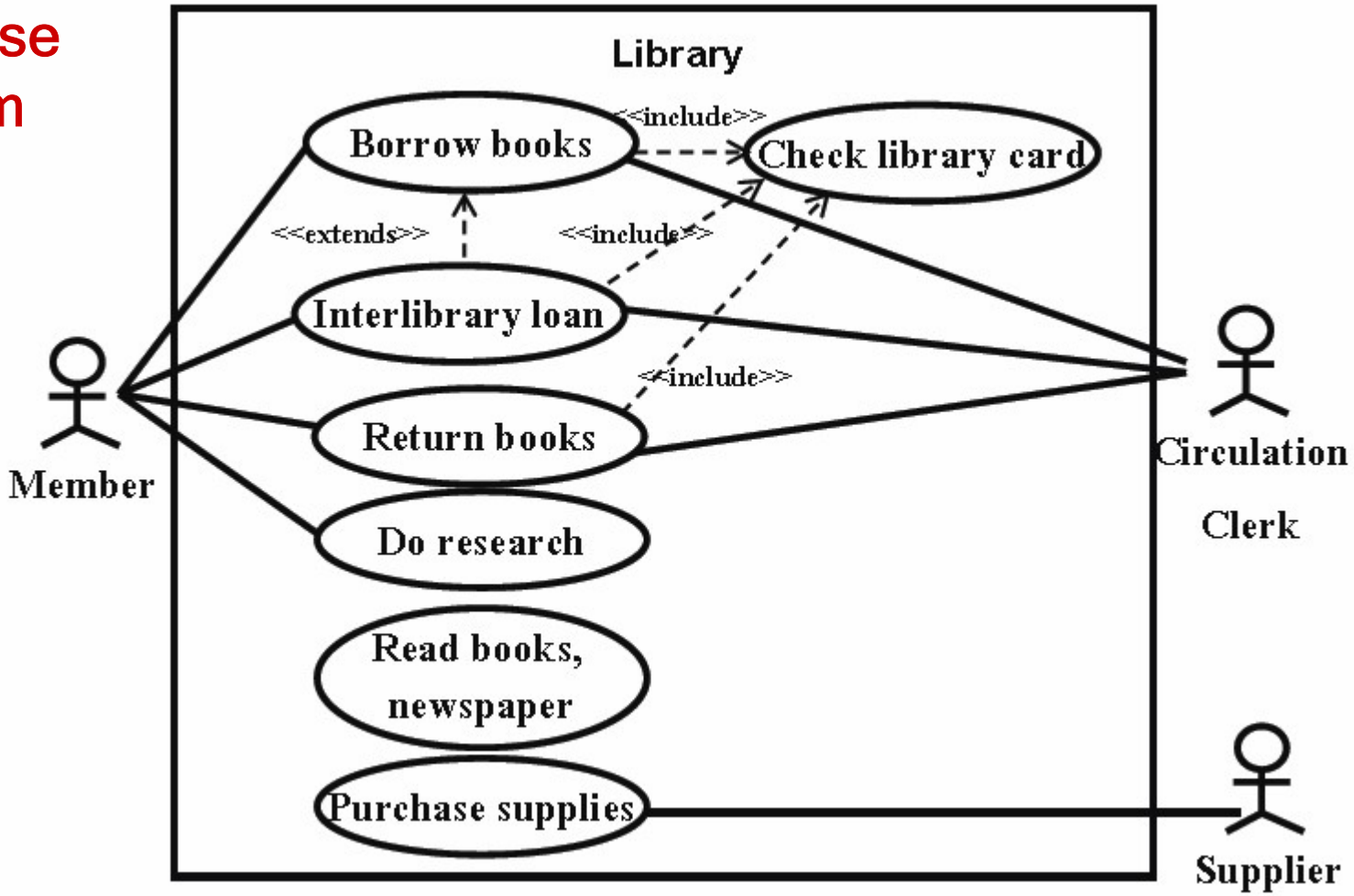
3. Dependency



Analysis and Design Phase

แบบจำลองตามแนวทางเชิงวัตถุ

Use Case Diagram

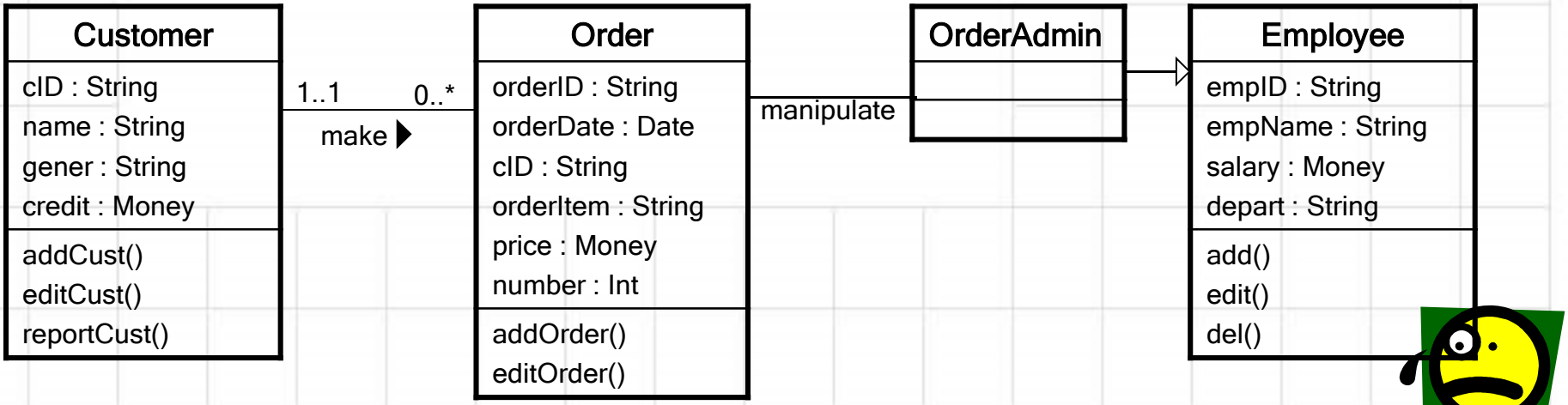


แบบจำลองตามแนวทางเชิงวัตถุ 1

Class Diagram

เป็นแผนภาพแสดงกลุ่มของคลาส โครงสร้างของคลาส และ Interface ตลอดจนแสดงความสัมพันธ์ (Relationship) ระหว่างคลาส การเริ่มต้นสร้างต้องหาอ็อบเจคใน Use Case ก่อน โดยมีเทคนิคดังนี้

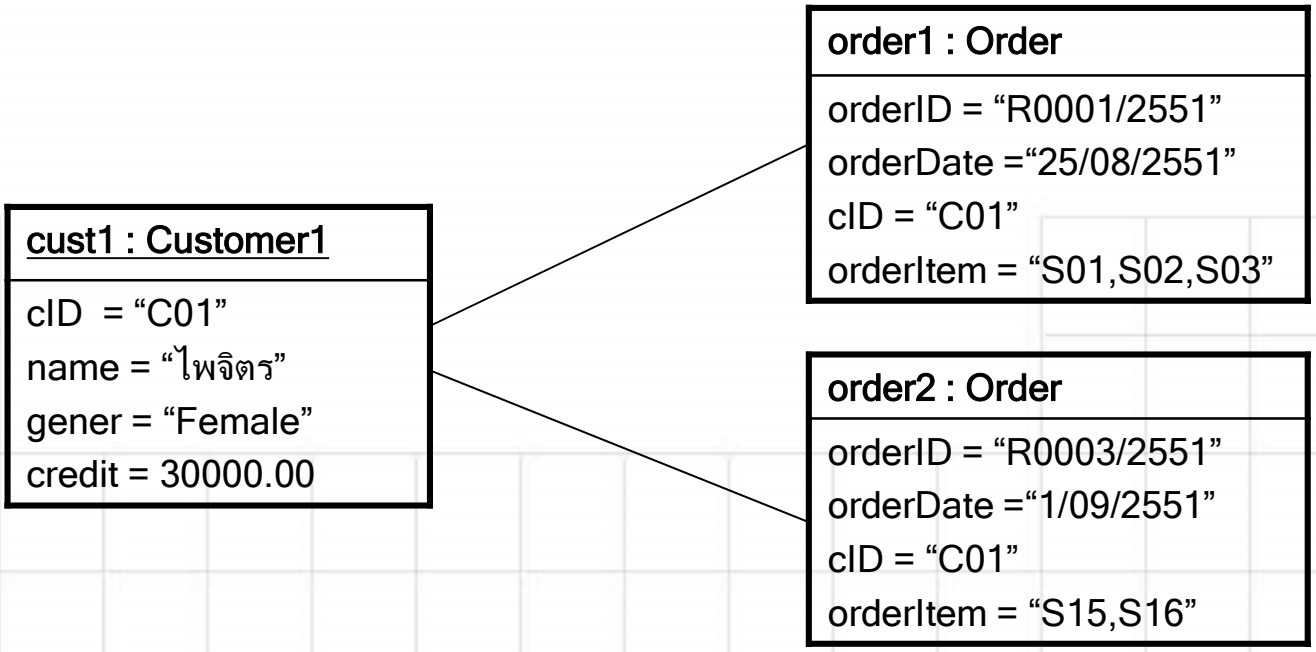
- ชื่อของคลาส หาได้จาก “คำนาม” (Noun)
- Attribute ของคลาส หาได้จาก “คำคุณศัพท์” (Adjective)
- Method ของคลาส หาได้จาก “คำกริยา” (Verb)



แบบจำลองตามแนวทางเชิงวัตถุ 1

Object Diagram

เป็นแผนภาพแสดงกลุ่มของอ็อบเจ็คและความสัมพันธ์ระหว่างอ็อบเจ็คที่เกิดขึ้นในคลาสต่าง ๆ ของ Class Diagram เนื่องจากเราไม่สามารถนำคลาสไปใช้งานได้โดยตรง แต่จะต้องสร้าง Instance ของคลาสขึ้นมา เรียกว่า Object



แบบจำลองตามแนวทางเชิงวัตถุ 2

Sequence Diagram

เป็นแผนภาพแสดงให้เห็นถึงปฏิสัมพันธ์ ระหว่างอ็อบเจ็ค โดยเฉพาะการส่ง Message ระหว่างอ็อบเจ็คตามลำดับของเวลา โดยแสดงในรูปแบบ 2 มิติ ด้วยเส้นประแนวตรั้ง (Lifeline) และเส้นแนวนอน (Message) นำเสนอเกี่ยวกับกับโต้ตอบระหว่างอ็อบเจ็คและคลาสต่าง ๆ ประกอบด้วยสัญลักษณ์ดังนี้

Actor

Object

Lifeline

Focus on control
Or Activation

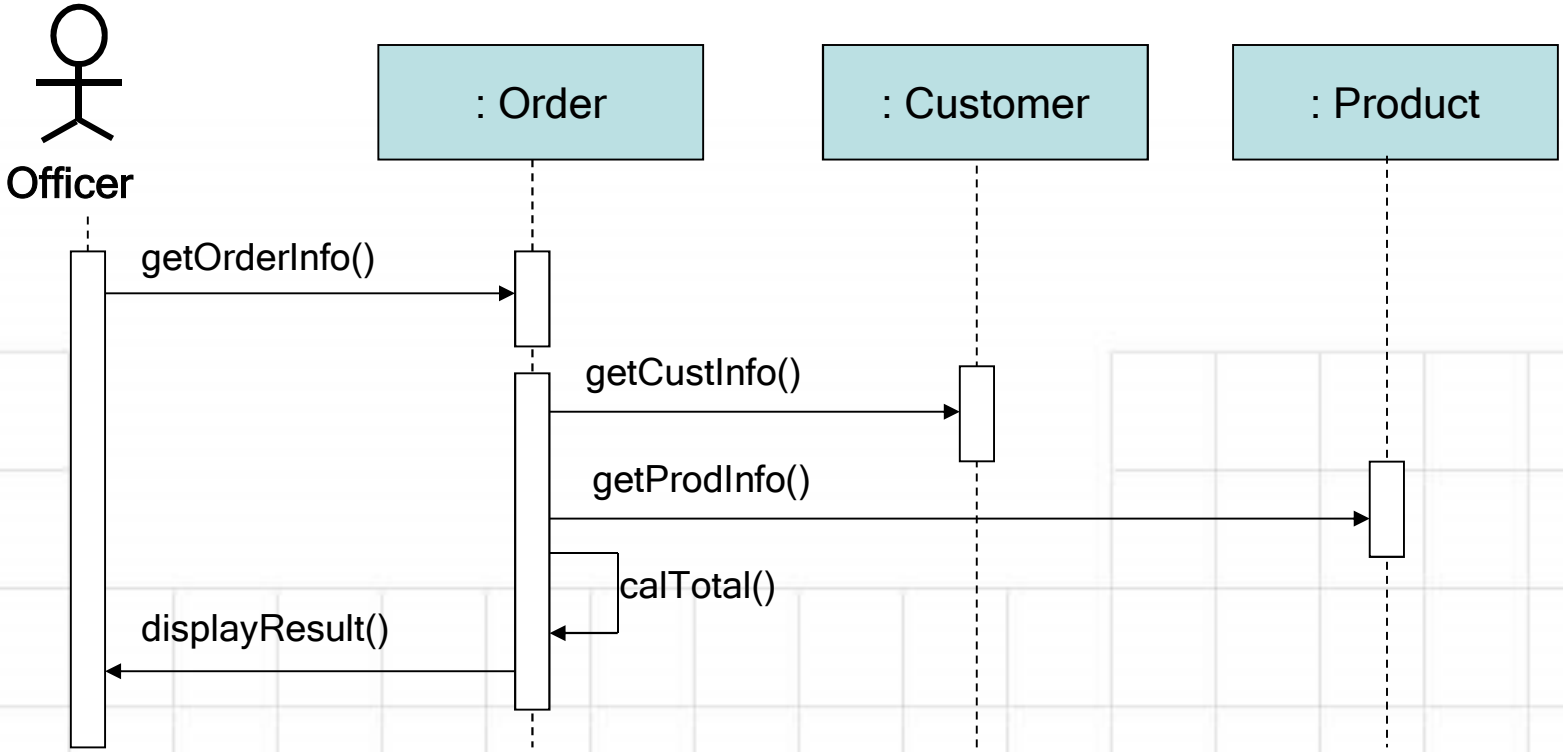
A Message

Callback Or
Self Delegation

แบบจำลองตามแนวทางเชิงวัตถุ 2

Sequence Diagram : พนักงานดึงข้อมูลแสดงใบสั่งซื้อ

(ในอ็อบเจ็ค Order Admin)



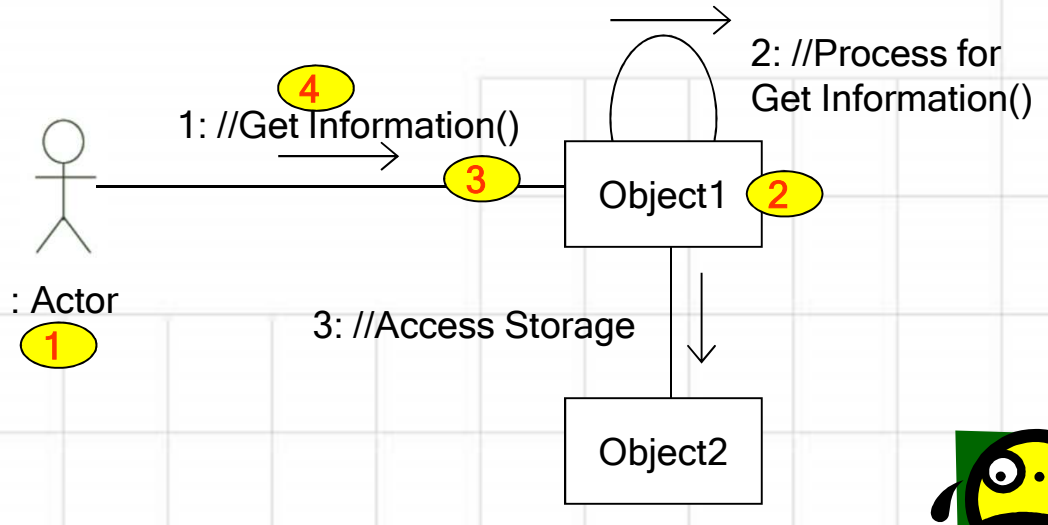
แบบจำลองตามแนวทางเชิงวัตถุ 2

Collaboration Diagram

เป็นแผนภาพแสดงให้เห็นถึงปฏิสัมพันธ์ ระหว่างอ็อบเจ็ค เช่นเดียวกับ Sequence Diagram แต่ต่างตรงที่ไม่แสดงลำดับการส่ง Message อย่างชัดเจน แต่จะเน้นความสัมพันธ์ระหว่างอ็อบเจ็คตามลักษณะการทำงาน (Control by Organization) ด้วยความที่แสดงใกล้เคียงกันนี้เอง ทีมผู้ออกแบบควรเลือกว่าจะใช้แผนภาพใดแผนภาพหนึ่งตามความเหมาะสม เช่น เน้นเรื่องลำดับเวลา หรือ เน้นความสัมพันธ์ระหว่างอ็อบเจ็ค เป็นต้น

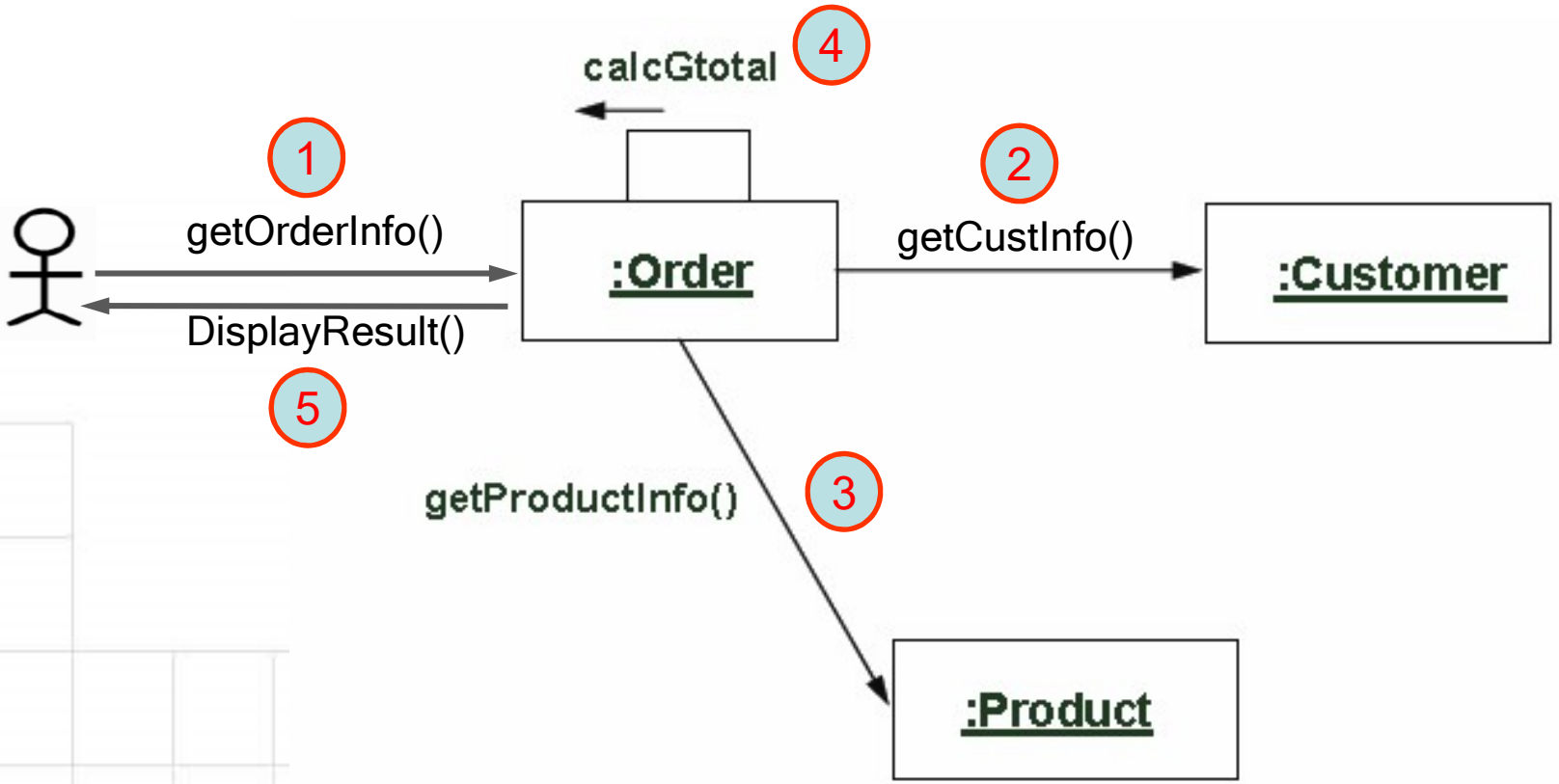
ประกอบด้วยสัญลักษณ์ ดังนี้

1. Actor
2. Object
3. Link
4. Message



แบบจำลองตามแนวทางเชิงวัตถุ 2

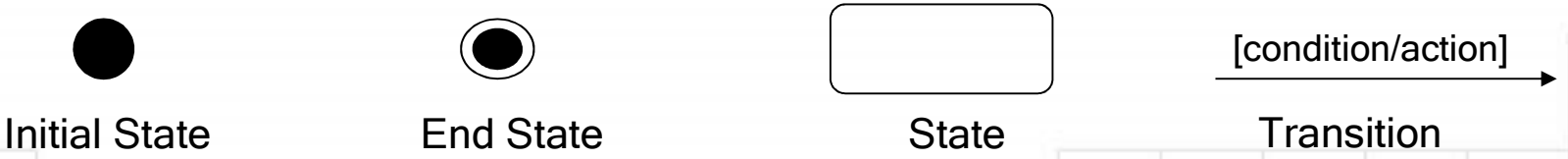
Collaboration Diagram



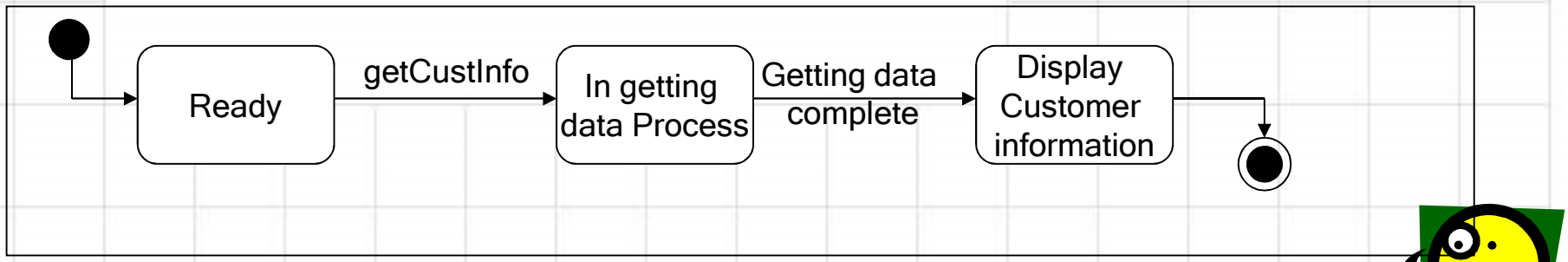
แบบจำลองตามแนวทางเชิงวัตถุ 2

State Diagram

เป็นแผนภาพแสดงให้เห็นพฤติกรรมของอ็อบเจ็คเช่นเดียวกับแผนภาพในกลุ่ม Behavioral Diagram แต่ State Diagram จะเน้นการแสดงให้เห็นถึงสถานะ (State) การเปลี่ยนสถานะ (Transition) ที่มีต่อเหตุการณ์ (Event) ที่เกิดขึ้นในช่วงชีวิตหนึ่ง ๆ ของอ็อบเจ็ค ประกอบไปด้วยสัญลักษณ์ดังนี้

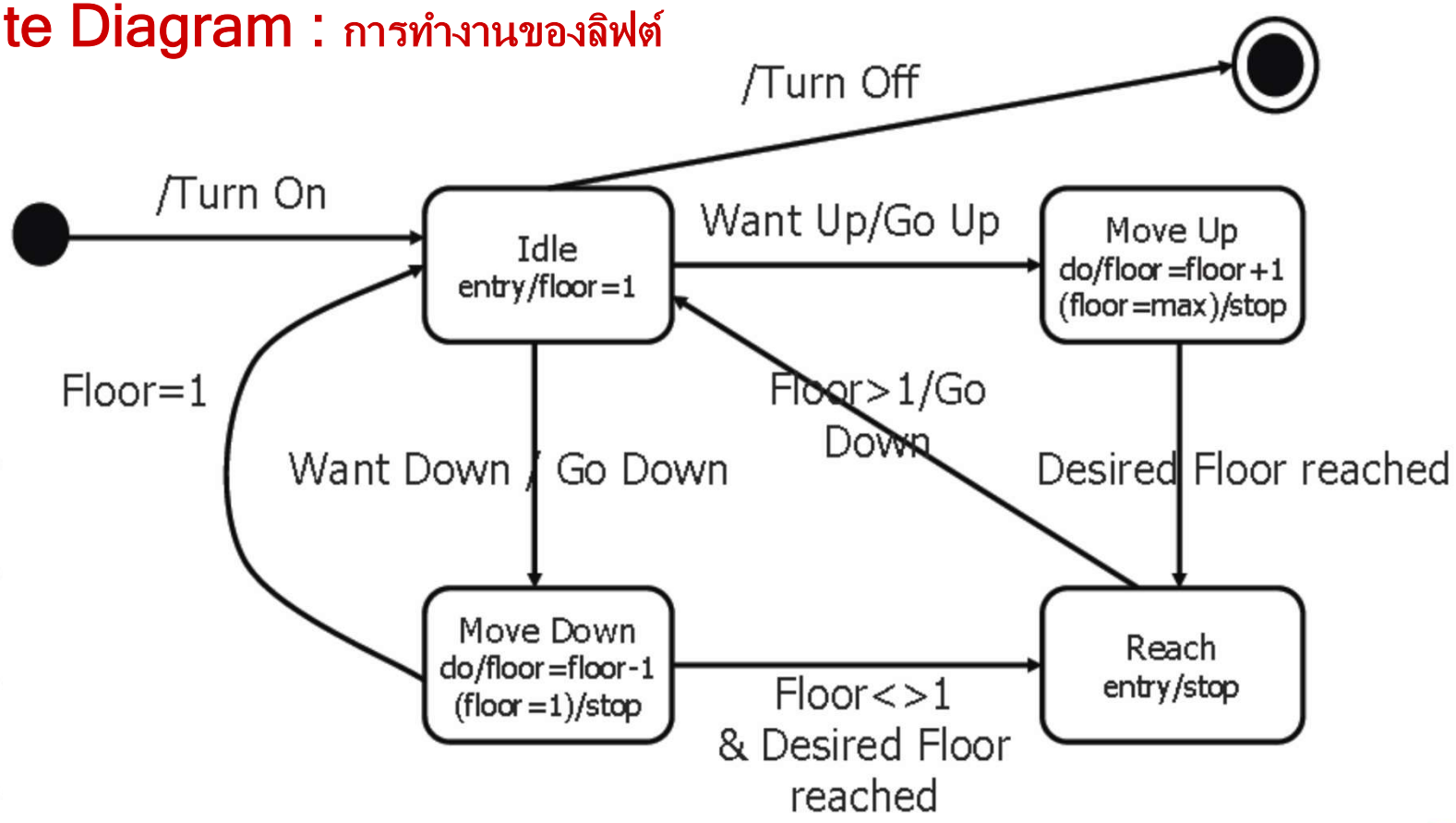


ตัวอย่างการเปลี่ยนแปลงใน อ็อบเจ็ค Customer



แบบจำลองตามแนวทางเชิงวัตถุ 2

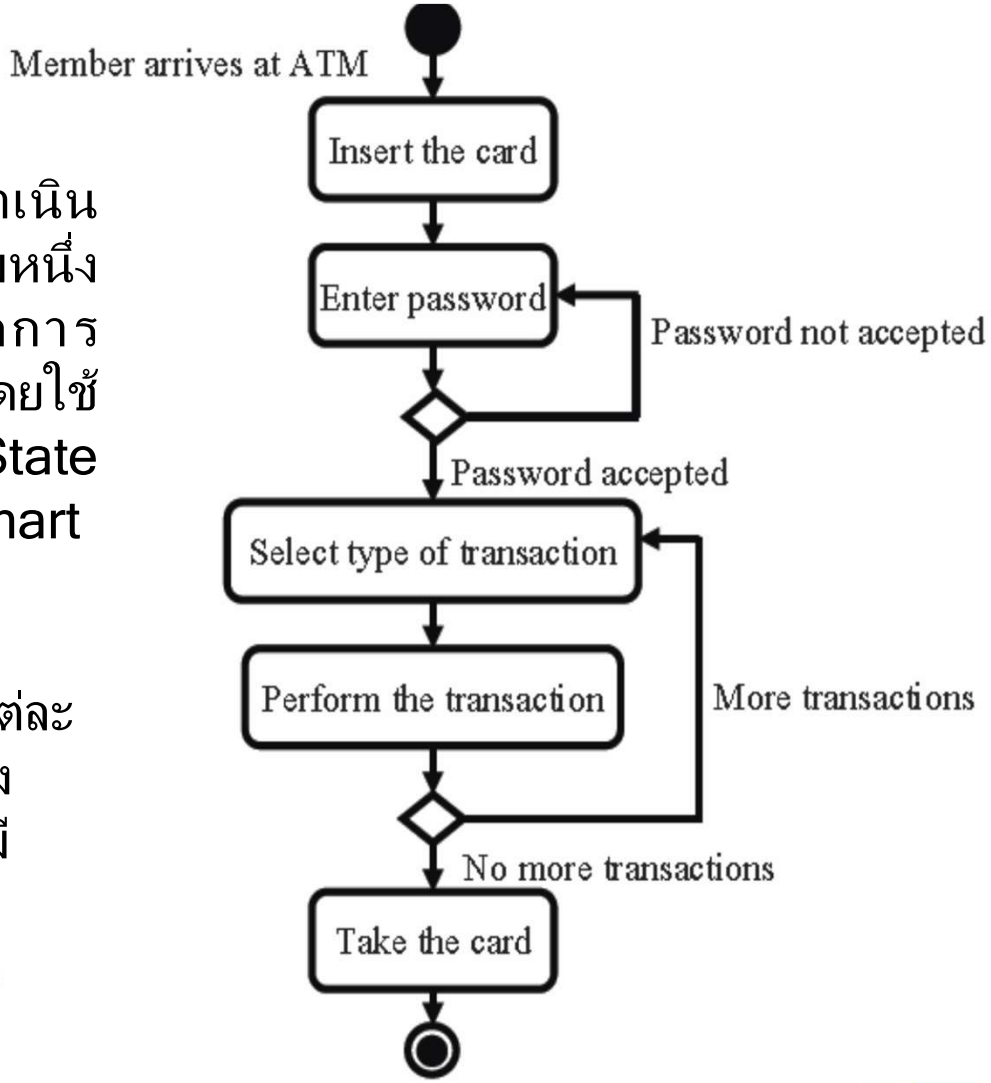
State Diagram : การทำงานของลิฟต์



แบบจำลองตามแนวทางเชิงวัตถุ

Activity Diagram 2

เป็นแผนภาพแสดงให้เห็นลำดับการดำเนินกิจกรรม (Activity) จากกิจกรรมหนึ่งไปยังกิจกรรมหนึ่ง ซึ่งเกิดจากการทำงานของอ็อบเจ็คภายในระบบ โดยใช้สัญลักษณ์เช่นเดียวกับ State Diagram มีลักษณะคล้าย Flowchart

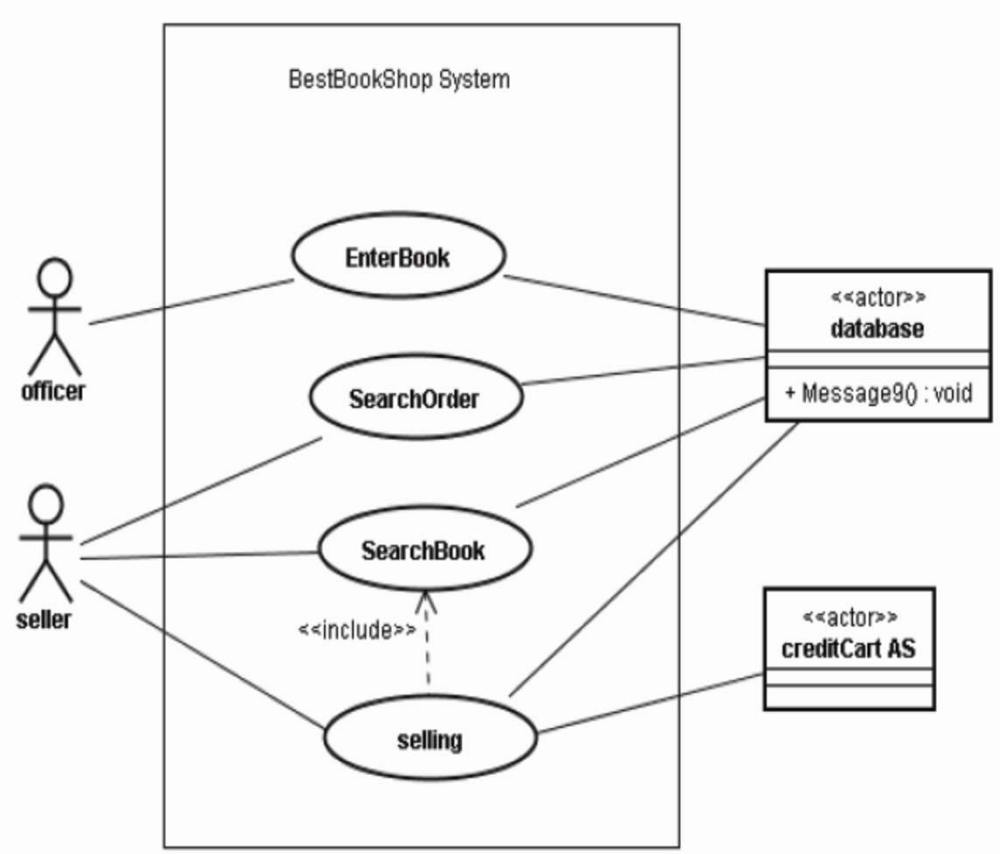


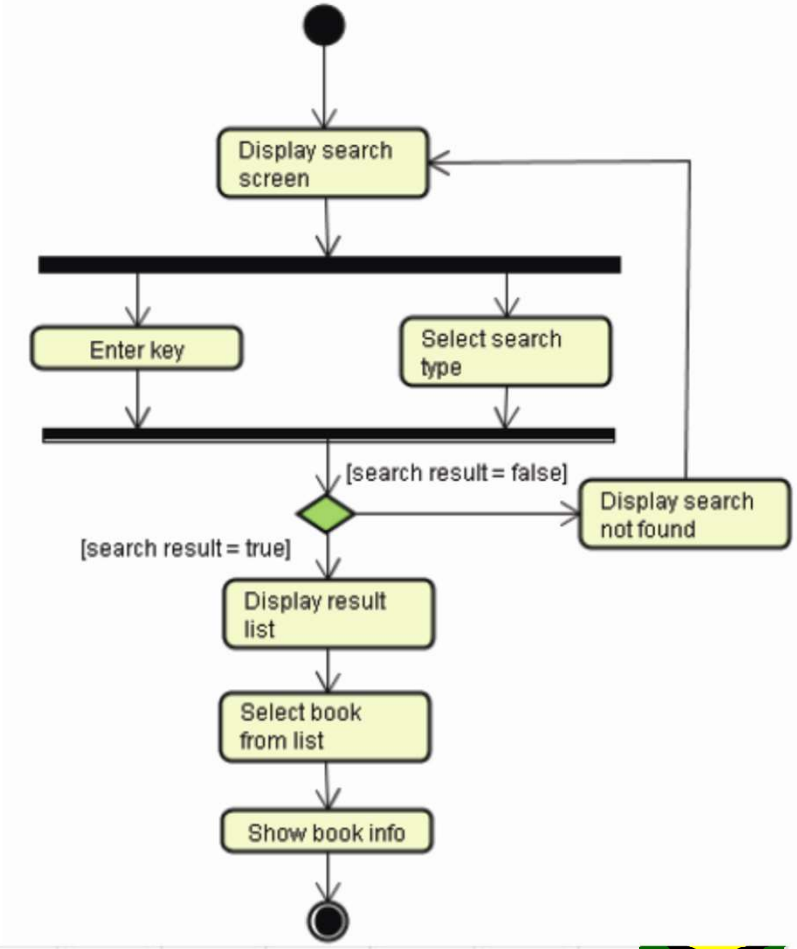
ข้อสังเกต

สามารถสร้าง Activity Diagram จากแต่ละ Use case หรือแต่ละ Class ได้ เพื่อบ่งบอกว่าในแต่ Use Case หรือ Class มีกิจกรรม หรือขั้นตอนการทำงานอย่างไร

แบบจำลองตามแนวทางเชิงวัตถุ

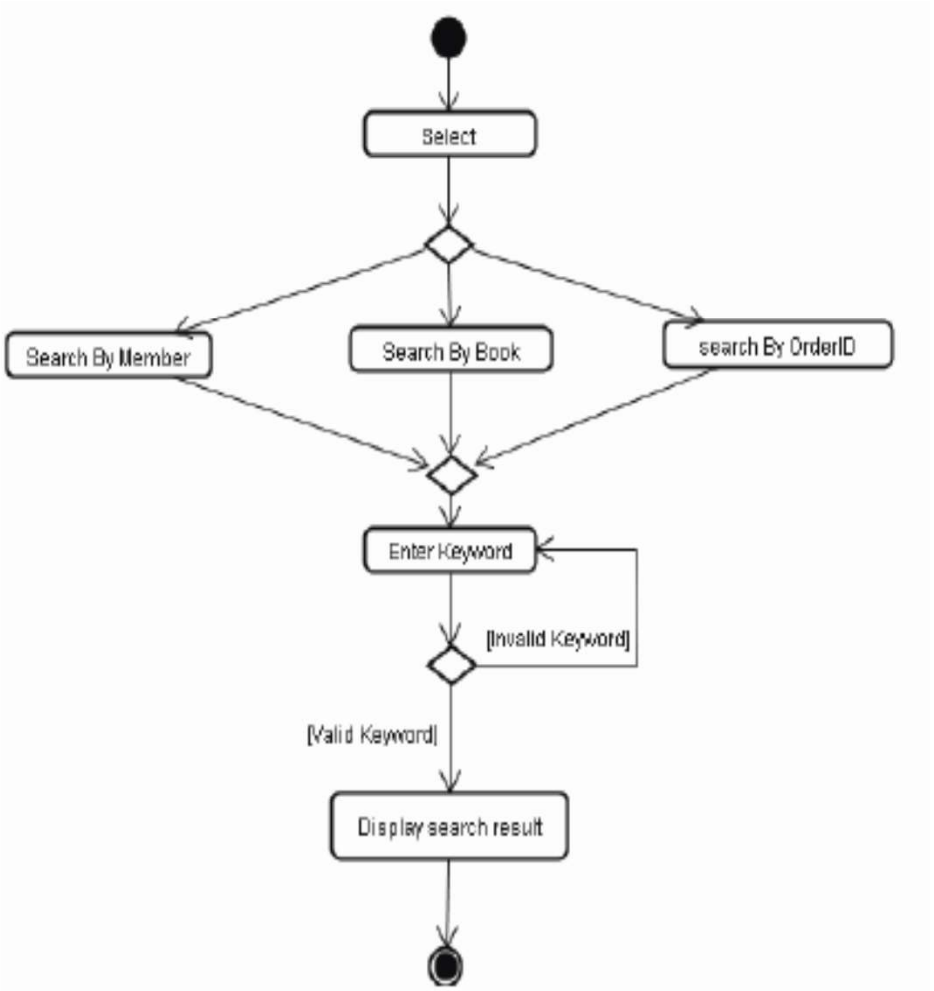
ข้อสังเกต สามารถสร้าง Activity Diagram จากแต่ละ Use case หรือแต่ละ Class ได้ เพื่อบ่งบอกว่าในแต่ Use Case หรือ Class มีกิจกรรม หรือขั้นตอนการทำงานอย่างไร

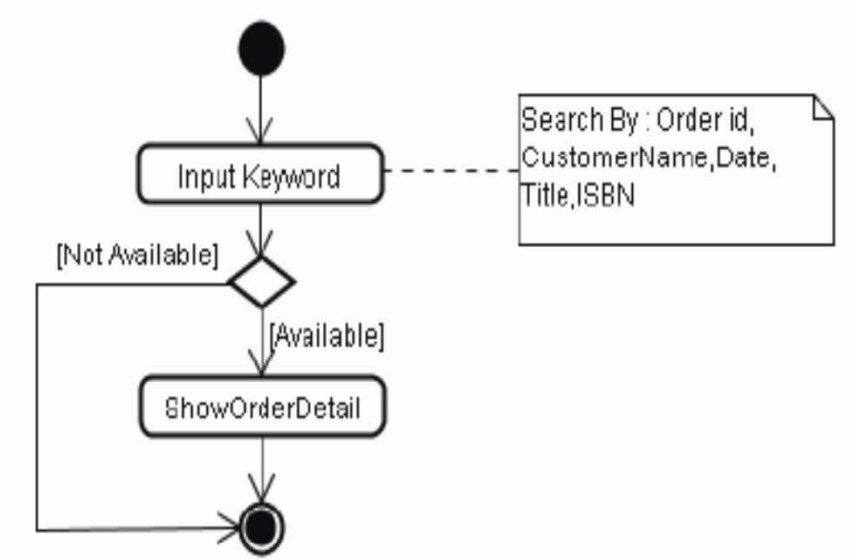




Analysis and Design Phase

แบบจำลองตามแนวทางเชิงวัตถุ



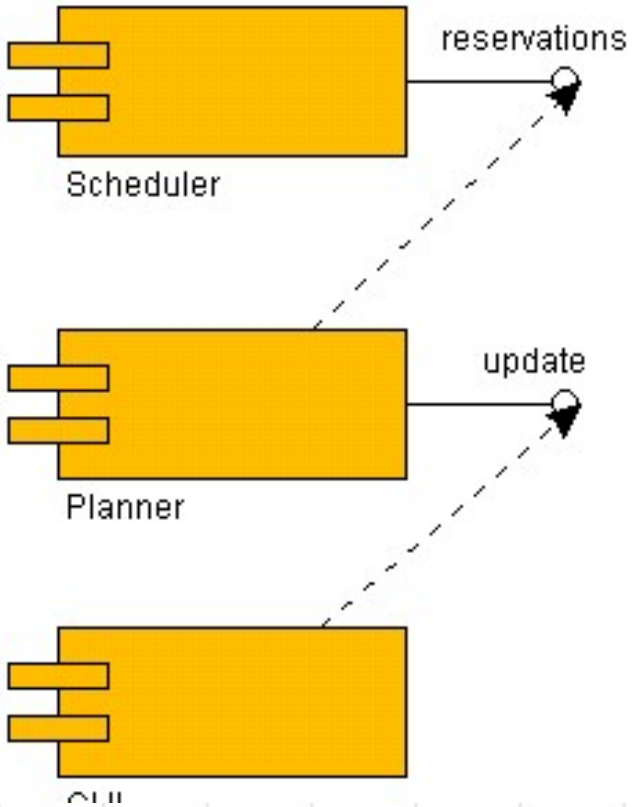


แบบจำลองตามแนวทางเชิงวัตถุ 1

Component Diagram

เป็นแผนภาพแสดงถึงโครงสร้างทางกายภาพของโปรแกรม ประกอบไปด้วยส่วนต่าง ๆ ที่เรียกว่า Module หรือ Component ซึ่งหมายถึงส่วนต่างๆ ของซอฟต์แวร์ของระบบทั้งหมด

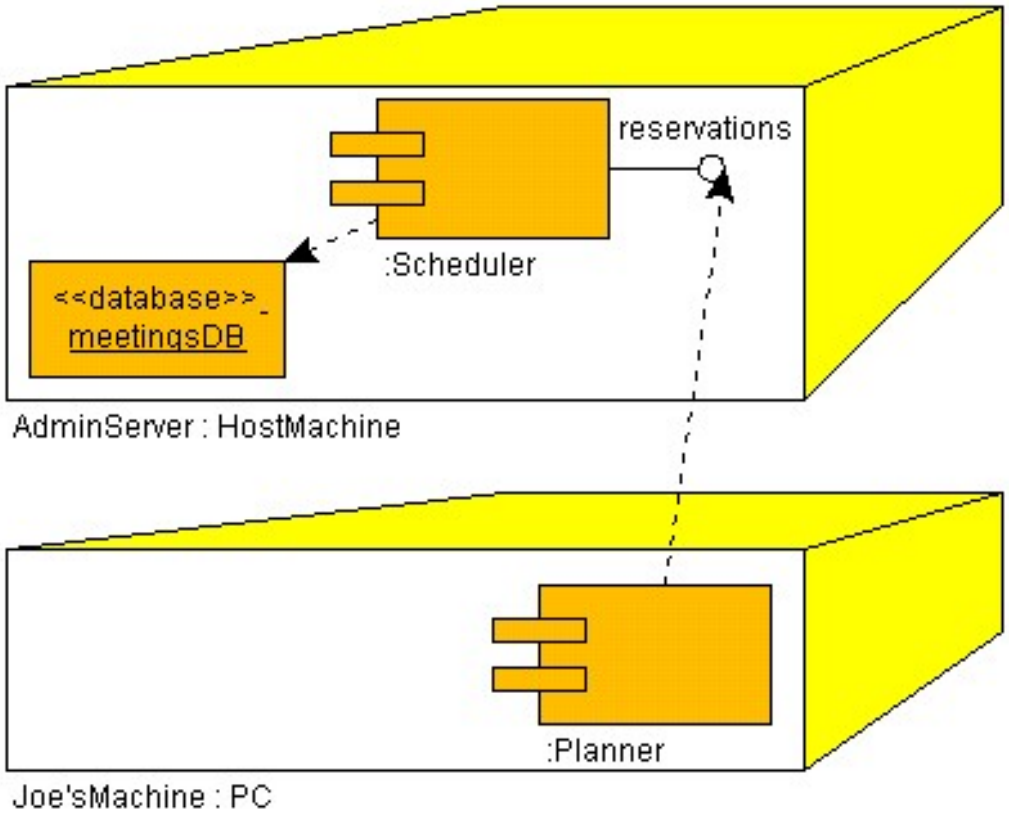
โดยแสดงการประสานกัน (Interface) หรือการขึ้นต่อกัน (dependencies) ของซอฟต์แวร์คอมโพเนนท์ที่ประกอบด้วย Source Code, Object Code และ Executables

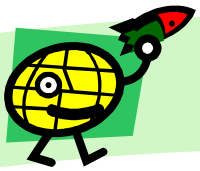


แบบจำลองตามแนวทางเชิงวัตถุ 1

Deployment Diagram

เป็นแผนภาพแสดงถึงโครงสร้างทางฮาร์ดแวร์และซอฟต์แวร์ทั้งหมดของระบบงาน ส่วนใหญ่จะใช้ร่วมกับ Component Diagram โดยการมองอุปกรณ์ฮาร์ดแวร์ทั้งหมดเป็นอ็อบเจ็คหรือคลาสได้เช่นเดียวกัน โดยแสดงการติดต่อสื่อสาร (Communication) ระหว่างคอมโพเนนท์ที่กำลังดำเนินการบนระบบ





Software Design Document

Software Design Document (SDD)

เป็นเอกสารอธิบายการผลิตซอฟต์แวร์ ชี้ให้เห็นแนวทางในการพัฒนาซอฟต์แวร์ของนักออกแบบ ซึ่งเอกสารนี้ต้องใช้การประสานร่วมกันภายในทีม เนื่องจากเป็นเสมือนการอธิบายว่าพวกเขาทำงานกันอย่างไร สื่อสารกันในเหตุการณ์ไหน ซึ่งต้องอธิบายได้อย่างสมบูรณ์ เพราะมีผลกระทบต่อการบำรุงรักษา และการวิวัฒน์ของซอฟต์แวร์ในรุ่นถัดไป

SDD ควรประกอบไปด้วย

1. Data Design : อธิบายถึงโครงภายในซอฟต์แวร์ ประกอบด้วย Attributes และ Relationship ของ Data Object (E-R Diagram/Class Diagram, Data Structure)
2. Architecture Design : อธิบายถึงการไหลของข้อมูลและงานภายในโครงสร้างโปรแกรม เป็นเสมือนแผนผังที่แสดงการเข้าออกของข้อมูลจาก method หนึ่งสู่ method หรือ (Data Flow Diagram/Use Case Diagram)
3. Interface Design : อธิบายถึงรูปแบบติดต่อใช้งานโปรแกรมทั้งภายในภายนอก ซึ่งแสดงถึงปฏิสัมพันธ์ของโปรแกรม (Component/Deployment Diagram, Users Interface)
4. Procedural Design : อธิบายถึงหลักการของโปรแกรม โดยใช้ทั้ง Graphical, Tabular และ Textual ที่สามารถนำเสนอถึงรายละเอียดในกระบวนการทำงาน เปรียบเสมือน blueprint ของการพัฒนา (Data Module Description)