ครั้งที่ 3

Unified Modeling Language

UML

UML (Unified Modeling Language)

หมายถึง ภาษารูปภาพที่ทำการกำหนดลักษณะของ class และการสร้าง class รวมถึงเป็นเอกสารที่บอกถึงรายละเอียดของระบบโครงสร้างโปรแกรม

UML

ข้อดีของ UML

- เป็นภาษารูปภาพมาตราฐานหรือภาษาสากลที่ใช้ในการพัฒนาซอฟต์แวร์เชิงวัตถุ
- ไม่ผูกติดกับภาษาโปรแกรมภาษาใดภาษาหนึ่ง
- เป็นภาษาที่ง่ายต่อการทำความเข้าใจ
- แปลงเป็นภาษาที่ใช้สร้างระบบงานได้อย่างอัตโนมัติ
- เป็นการเข้าใจปัญหาและค้นพบทางแก้ปัญหาได้อย่างรวดเร็วและง่ายขึ้น
- เป็นเอกสารที่พร้อมนำไปปรับปรุงแก้ไขได้อย่างรวดเร็ว

UML

องค์ประกอบของ UML

1. **สัญลักษณ์ทั่วไป(Things)**

   - หมวดโครงสร้าง (Structural) ได้แก่ ยูสเคส คลาส อินเทอร์เฟซ คอมโพเนนต์ คอลแลบอเรชั่น และโหนด
   - หมวดพฤติกรรม (Behavioral)ได้แก่ อินเตอร์แอ็กชั่น สเตตแมชชีน
   - หมวดจัดกลุ่ม (Grouping)ได้แก่ แพ็กเกจ
   - หมวดคำอธิบายประกอบ(Annotational) ได้แก่ โน้ต (Note)

UML

องค์ประกอบของ UML

2. ความสัมพันธ์ (Relationships)

- ความสัมพันธ์แบบพึ่งพา (Dependency Relationship)
- ความสัมพันธ์แบบเกี่ยวพันธ์ (Association Relationship)
- ความสัมพันธ์แบบเจเนอรัลไลเซชั่น (Generalization Relationship)

หรือความสัมพันธ์แบบไม่เจาะจง

ได้แก่ ความสัมพันธ์แบบสืบทอดคุณสมบัติ (Inheritance)

UML

องค์ประกอบของ UML

3. ไดอะแกรม (Diagrams)

- Use Case Diagram เป็นโมเดลฟังก์ชั่นการทำงานของระบบ
- Class Diagram เป็นโมเดลคลาสต่าง ๆ ที่จำเป็นต่อระบบ
- Activity Diagram มีหลักการเดียวกับ Flowchart
- Statechart Diagram ใช้สำหรับแสดงสถานะของ Object ในระหว่างทำงาน
- Collaboration Diagram ใช้แสดงการทำงานร่วมกันของ Object ในระบบ

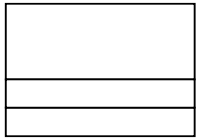
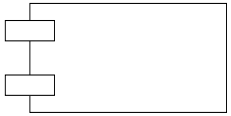
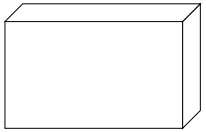
UML

องค์ประกอบของ UML

3. ไดอะแกรม (Diagrams)ต่อ

- Sequence Diagram ใช้เป็นโมเดลกิจกรรมต่าง ๆ ที่เกิดขึ้นกับ Object ในระบบ
- Component Diagram ใช้สำหรับสร้างโมเดล Component ในระบบ
- Deployment Diagram ใช้แสดงการติดตั้งใช้งานส่วนประกอบของระบบ

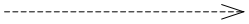
Structural Modeling: Core Elements

Construct	Description	Syntax
class	a description of a set of objects that share the same attributes, operations, methods, relationships and semantics.	
interface	a named set of operations that characterize the behavior of an element.	
component	a physical, replaceable part of a system that packages implementation and provides the realization of a set of interfaces.	
node	a run-time physical object that represents a computational resource.	

Structural Modeling: Core Elements (cont'd)

Construct	Description	Syntax
constraint[1]	a semantic condition or restriction.	

Structural Modeling: Core Relationships

Construct	Description	Syntax
association	a relationship between two or more classifiers that involves connections among their instances.	
aggregation	A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.	
generalization	a taxonomic relationship between a more general and a more specific element.	
dependency	a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).	

Use Case Diagram

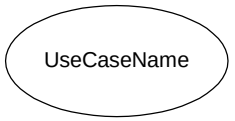
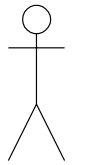
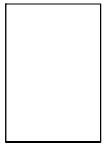
ค้นหาว่าระบบทำอะไร โดยไม่สนใจกลไกการทำงานหรือเทคนิค

เปรียบเสมือน BLACK BOX กล่องดำ

ประโยชน์

- ทราบความสามารถของระบบ
- ทราบผู้ใช้งานในแต่ละส่วนของระบบ
- ง่ายต่อการสื่อสารระหว่างลูกค้าและผู้พัฒนาระบบ
- ใช้ทดสอบระบบว่าตรงตามความต้องการของระบบหรือไม่

Use Case Modeling: Core Elements

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

Use Case Modeling: Core Relationships

Construct	Description	Syntax
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
generalization	A taxonomic relationship between a more general use case and a more specific use case.	
extend	A relationship from an *extension* use case to a *base* use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	

Use Case Modeling: Core Relationships (cont'd)

Construct	**Description**	**Syntax**
include	An relationship from a *base* use case to an *inclusion* use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.	<<include>>

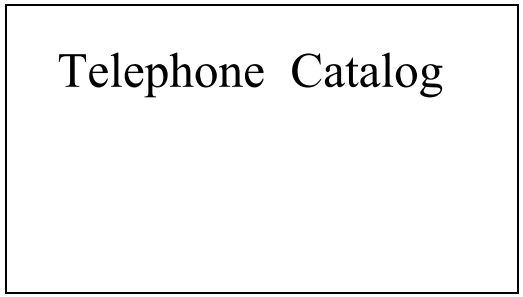
องค์ประกอบUse Case Diagram

1. **Use Case**

- ระบบใน Use Case Diagram แสดงด้วยรูปกล่องสี่เหลี่ยม

  ซึ่งบรรจุสัญลักษณ์ Use Case อยู่ภายใน

  และมีชื่อของระบบเขียนอยู่ข้างบนหรือข้างในกล่อง



องค์ประกอบ ของ USE CASE DIAGRAM

2. Actor

- ผู้ที่กระทำกับระบบ โดยอาจเป็นคนหรือไม่ก็ได้
- เช่น นักศึกษาพิมพ์ชื่อหนังสือเข้าไปในระบบเพื่อค้นหนังสือในห้องสมุด

และรอรับผลการค้นหา

ดังนั้น นักศึกษาจัดเป็น Actor

องค์ประกอบ ของ USE CASE DIAGRAM

2. Actor

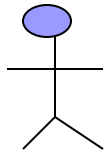
แบ่งเป็น 2 ประเภท

- แอ็กเตอร์หลัก หมายถึง แอ็กเตอร์ที่มีความสำคัญโดยตรงต่อความสามารถหลักของระบบ ซึ่งแสดงด้วย Use Case ถือว่าสำคัญสุด
- แอ็กเตอร์รอง หมายถึง แอ็กเตอร์ที่มีหน้าที่ความสำคัญรองลงไปจากแอ็กเตอร์หลัก

องค์ประกอบ ของ USE CASE DIAGRAM

2. Actor

ใช้แทนด้วยรูปคน Stick Man



Customer

องค์ประกอบ ของ USE CASE DIAGRAM

2. Actor

ความสัมพันธ์ระหว่างแอ็กเตอร์ (Relationship)

หากแอ็กเตอร์หลายแอ็กเตอร์ในระบบซึ่งมีคุณสมบัติคล้าย ๆ กัน สามารถแยกออกมาเป็นอีกแอ็กเตอร์หนึ่งที่รวมบทบาทที่เหมือนกันของแต่ละแอ็กเตอร์ ซึ่งเป็นคุณสมบัติพื้นฐาน (General) เสมือนเป็น ซุปเปอร์คลาส (Superclass)

แอ็กเตอร์พื้นฐานจะถูกสืบทอด (Inherit) จากแอ็กเตอร์อื่น เรียกว่า Generalization

องค์ประกอบ ของ USE CASE DIAGRAM

2. Actor

ความสัมพันธ์ระหว่างแอ็กเตอร์ (Relationship)

ตัวอย่างเช่น ในระบบหนึ่งมี 3 แอ็กเตอร์คือ

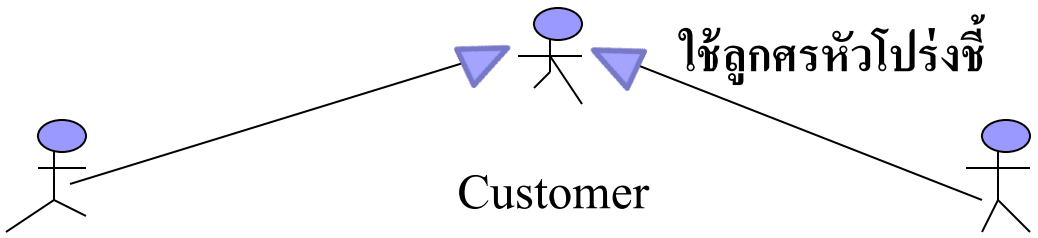
* ลูกค้า(Customer)
* ลูกค้าที่สั่งซื้อทางโทรศัพท์ (Telephone Customer)
* ลูกค้าที่เดินทางมาด้วยตนเอง (Personal Visit Customer)

องค์ประกอบ ของ USE CASE DIAGRAM

2. Actor

ความสัมพันธ์ระหว่างแอ็กเตอร์ (Relationship)

ในระบบนี้ลูกค้าที่สั่งซื้อทางโทรศัพท์และลูกค้าที่เดินทางมาด้วยตนเอง ต่างก็มีบทบาทพื้นฐานเดียวกันก็คือเป็นแอ็กเตอร์ลูกค้าเหมือนกัน

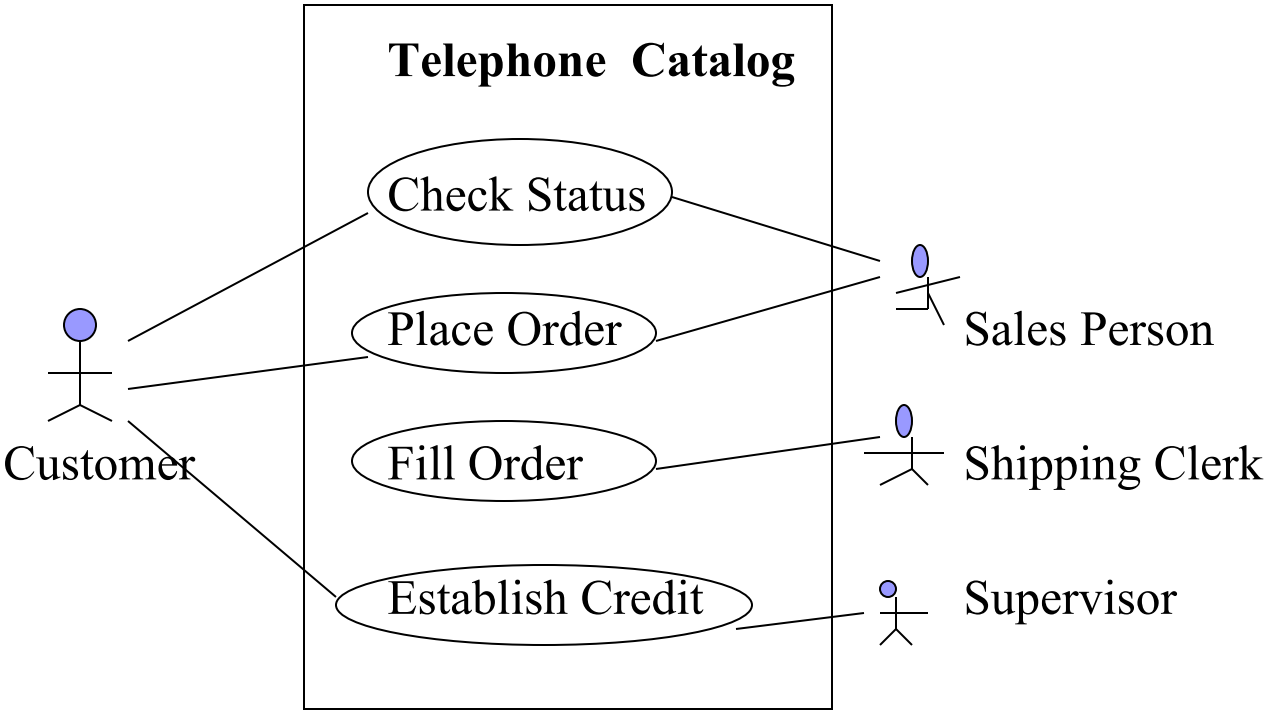


Telephone Customer

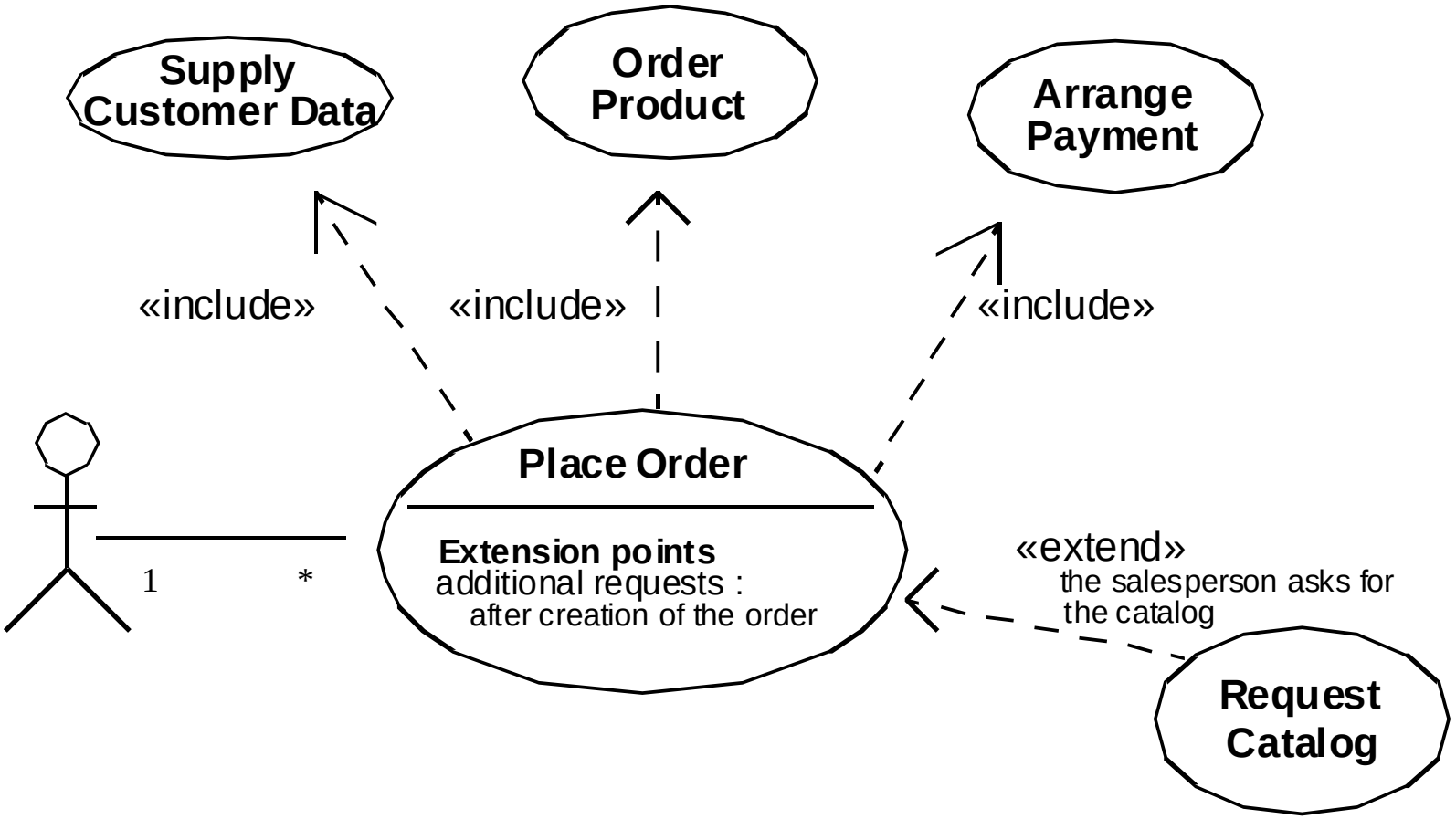
Personnal Visit Customer

Use Case Diagram

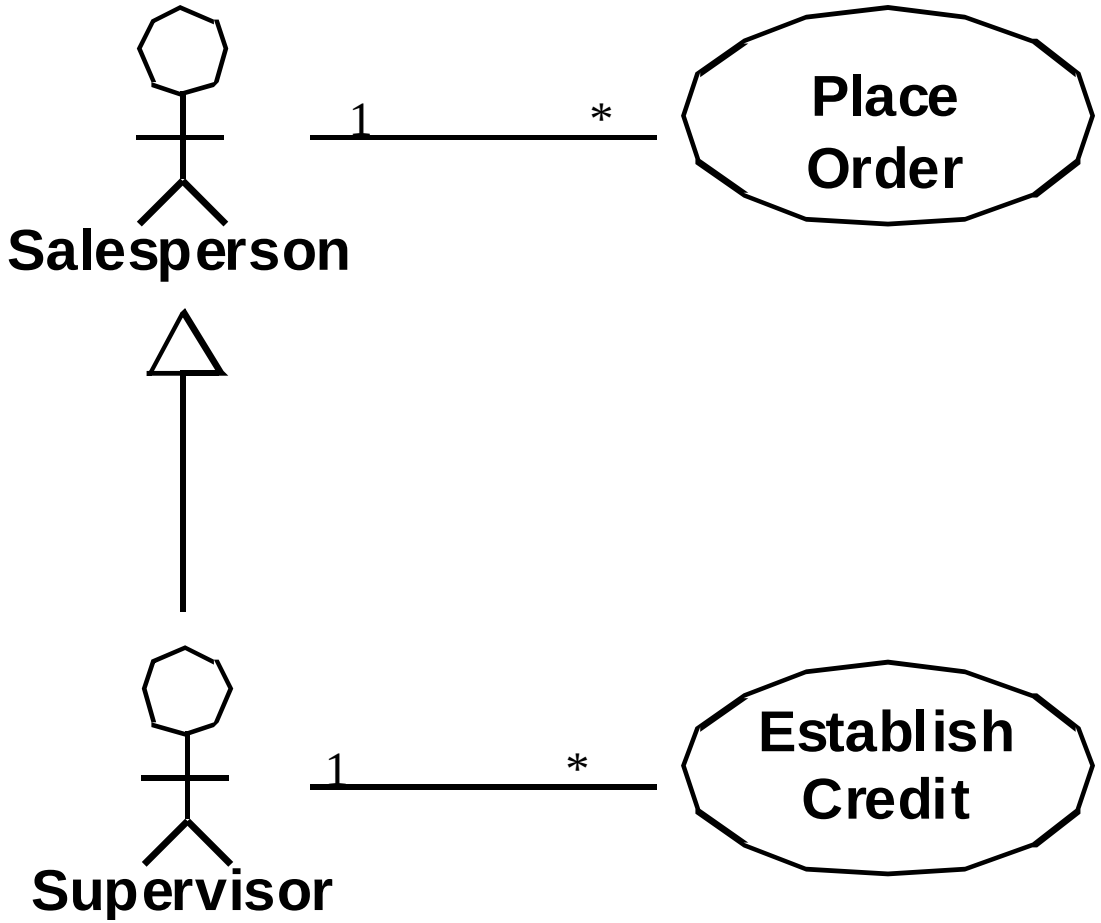
ตัวอย่าง



Use Case Relationships



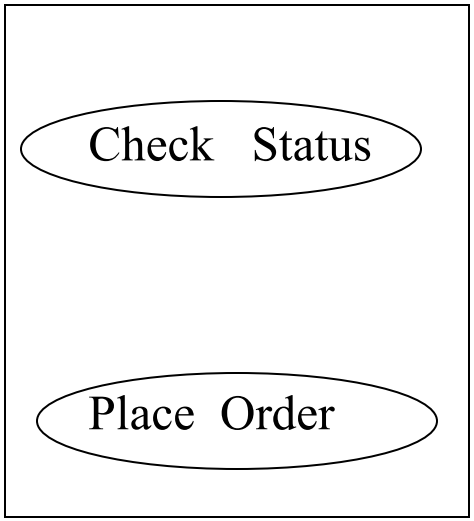
Actor Relationships



องค์ประกอบ ของ USE CASE DIAGRAM

1. Use Case ยูสเคส

ใช้สัญญลักษณ์รูปวงรี มีชื่อยูสเคสอยู่ข้างใน และทุกยูสเคสอยู่ภายใต้กรอบสี่เหลี่ยม ดังนี้



องค์ประกอบ ของ USE CASE DIAGRAM

3. ความสัมพันธ์ระหว่างยูสเคส

1. ความสัมพันธ์แบบขยาย (Extend Relationship)

ยูสเคสหนึ่งอาจถูกช่วยเหลือโดยการทำงานยูสเคสอื่น

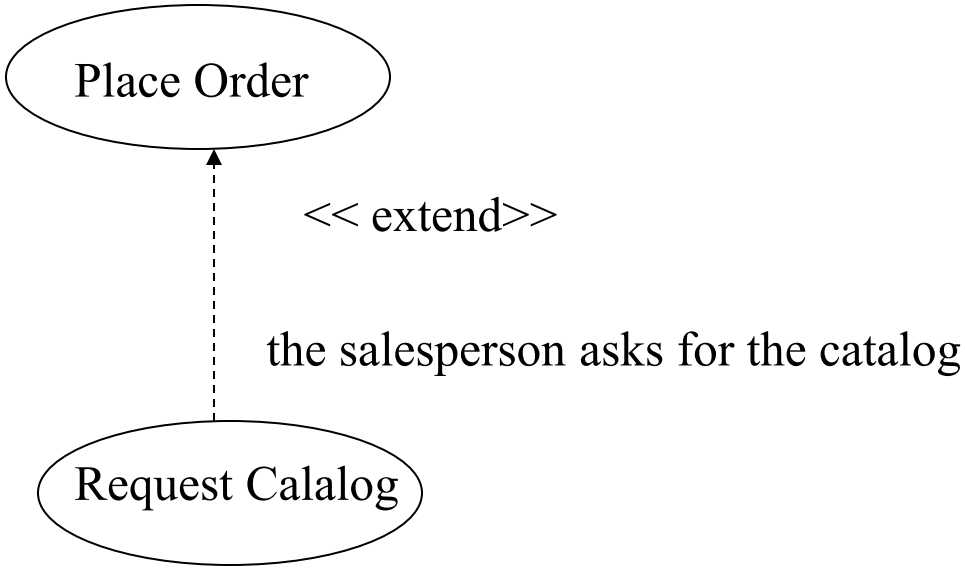
เช่น ยูสเคสการใส่รหัสอาจถูกช่วยเหลือโดยยูสเคสคำอธิบายการใส่รหัส

สัญลักษณ์ใน UML คือลูกศรเส้นประที่ชี้จากยูสเคสแรกไปยังยูสเคสที่ถูกช่วยเหลือหรือถูกขยาย โดยมีคำว่า extend อยู่ในเครื่องหมาย << >>

องค์ประกอบ ของ USE CASE DIAGRAM

3. ความสัมพันธ์ระหว่างยูสเคส

1. ความสัมพันธ์แบบขยาย (Extend Relationship)



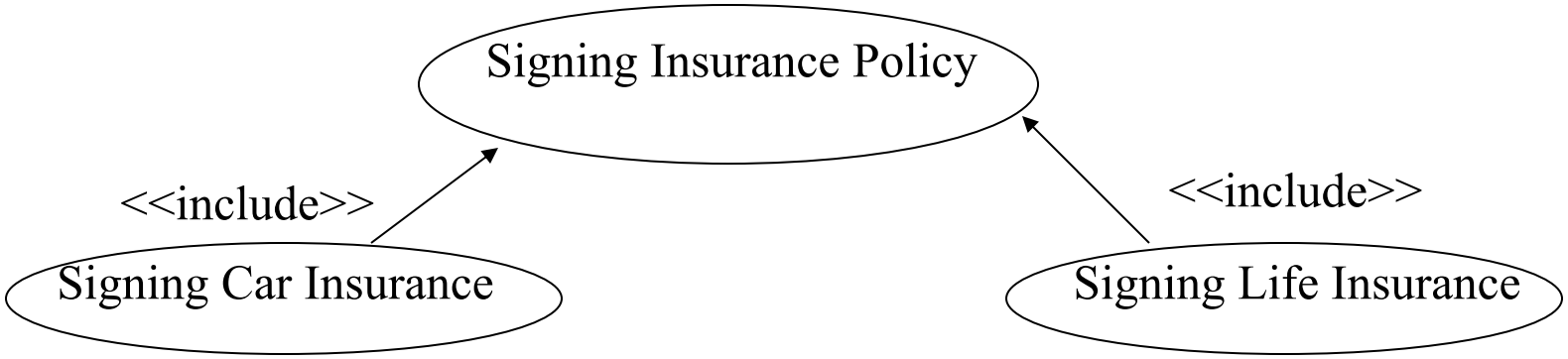
องค์ประกอบ ของ USE CASE DIAGRAM

3. ความสัมพันธ์ระหว่างยูสเคส

2. ความสัมพันธ์แบบรวม (Include Relationship)

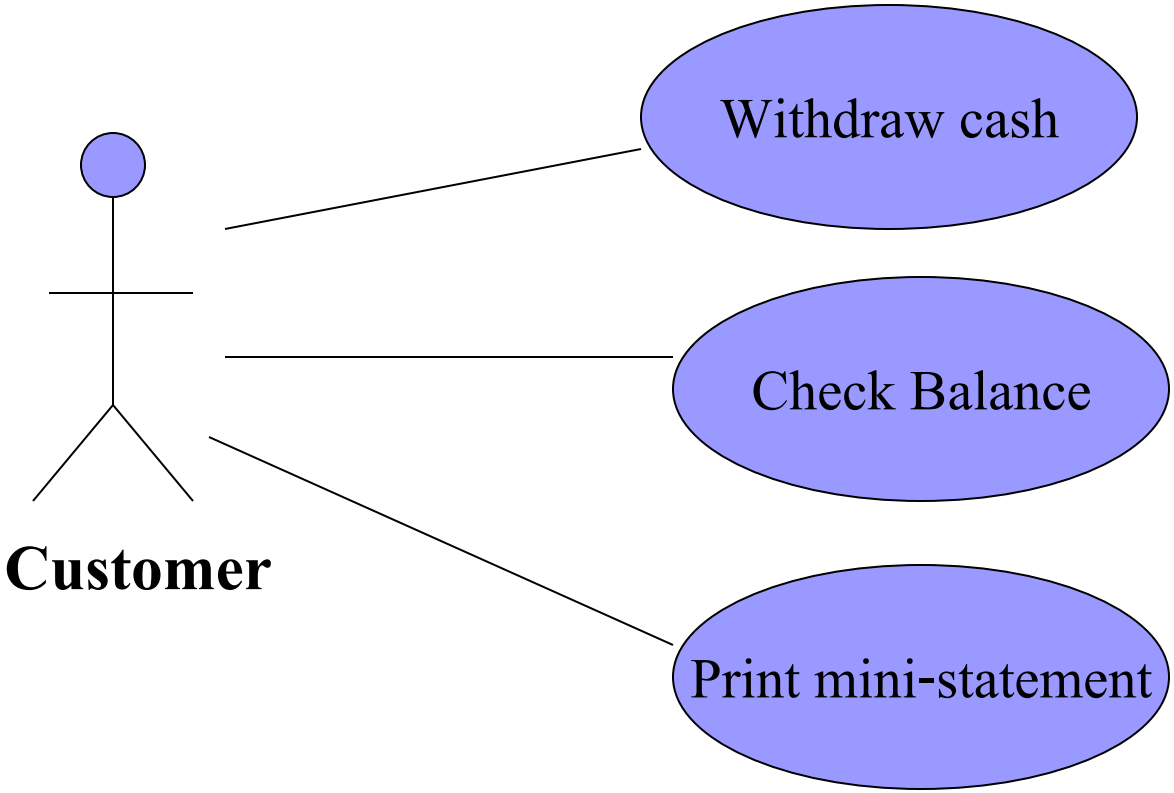
ยูสเคสหนึ่งอาจจำเป็นต้องอาศัยการทำงานของยูสเคสอื่น

สำหรับยูสเคสที่ถูกเรียกใช้โดยยูสเคสอื่น แทนด้วย << include>> ที่กึ่งกลางลูกศร



ตัวอย่าง

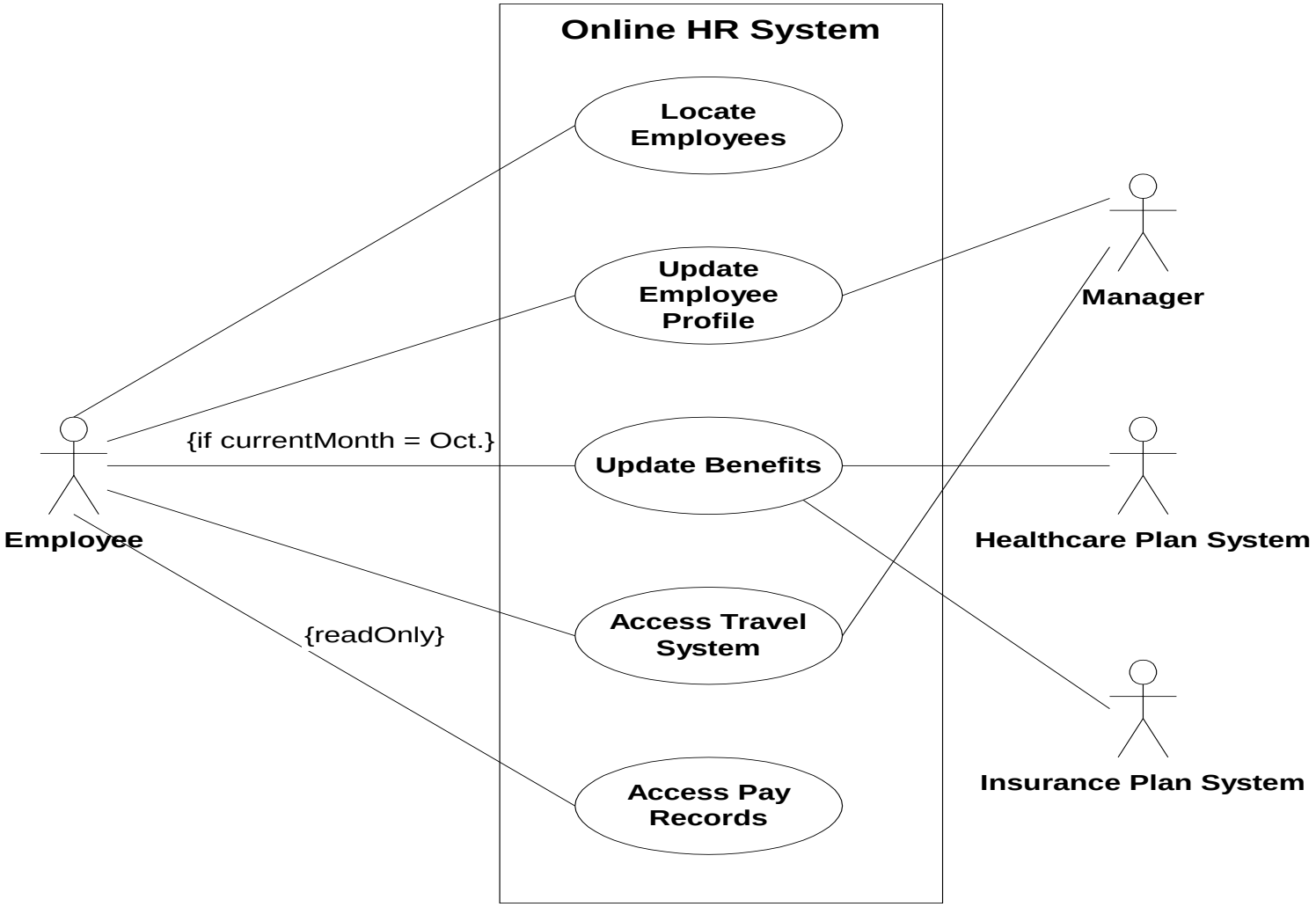
Use Case Diagram for ATM subsystem



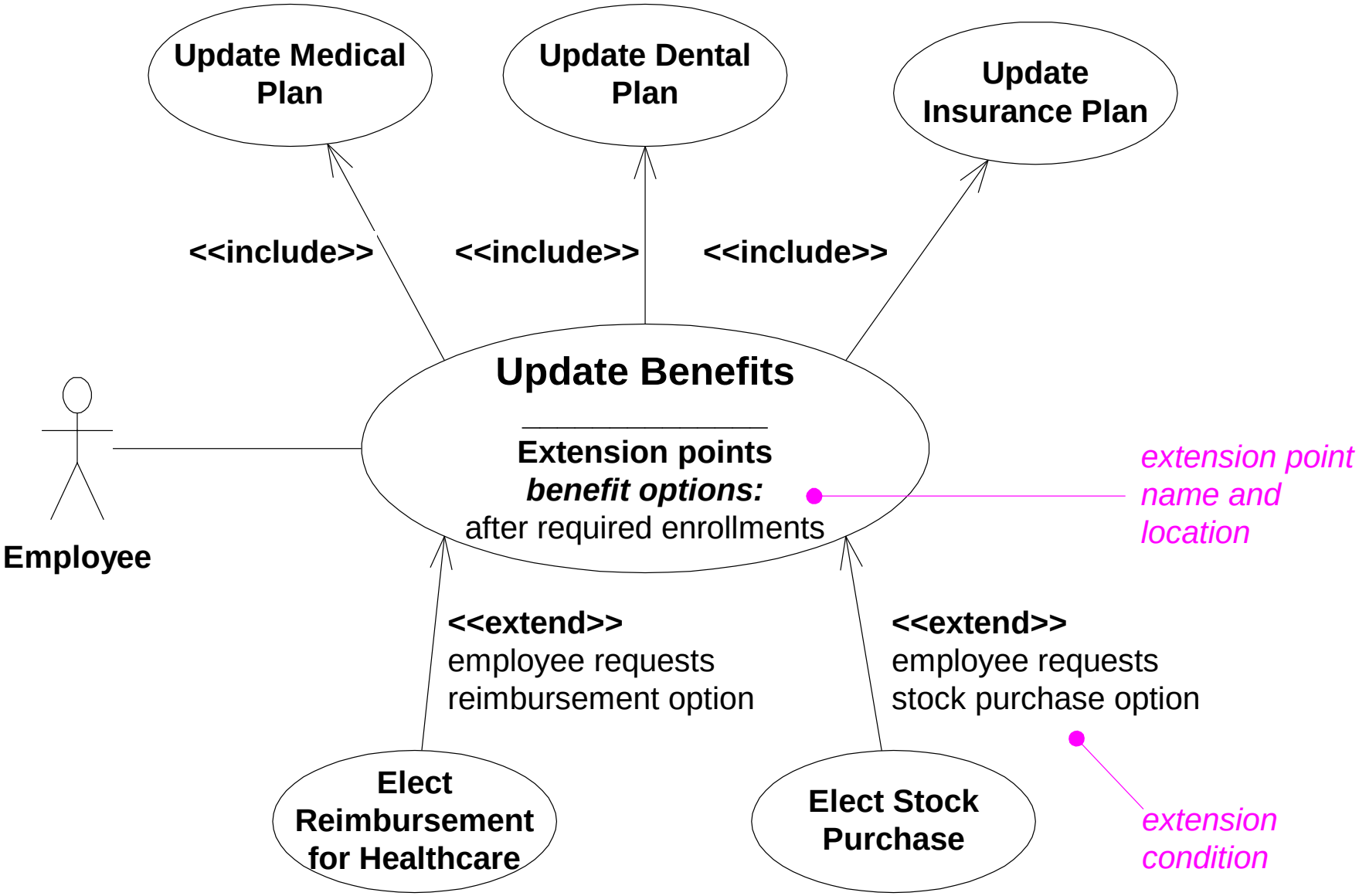
Online HR System: Update Benefits Use Case

- **Actors:** employee, employee account db, healthcare plan system, insurance plan system
- **Preconditions:**
  - Employee has logged on to the system and selected ‘update benefits’ option
- **Basic course**
  - System retrieves employee account from employee account db
  - System asks employee to select medical plan type; **include** Update Medical Plan.
  - System asks employee to select dental plan type; **include** Update Dental Plan.
  - …
- **Alternative courses**
  - If health plan is not available in the employee’s area the employee is informed and asked to select another plan...

Example: Online HR System



Online HR System: Use Case Relationships



UML : Class Diagram

Class Diagram

- เป็น Diagram ที่แสดงองค์ประกอบทั้งหมดของระบบ

Object

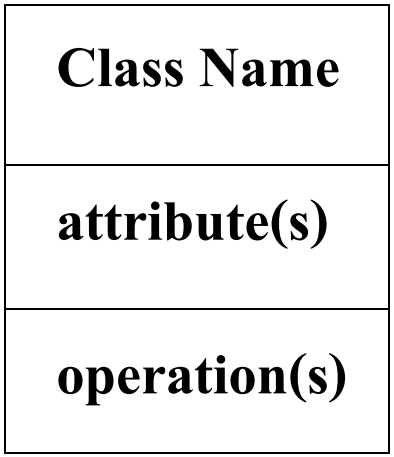
- คือทุกสิ่งที่เรามองเห็น เช่น คน สิ่งของ เครื่องจักร ไฟล์

Class

- ชนิดของกลุ่ม Object
- ตัวอย่าง ของคลาส: ระบบซอฟต์แวร์ มีคลาส ไฟล์ ไอคอน วินโดว์ เมาส์

UML : Class Diagram

สัญลักษณ์ UML แสดงคลาส



1. **Class Name** จะขึ้นต้นด้วยอักษรตัวใหญ่แบบหนา และจะเอียงหากเป็น

   Abstract Class

UML : Class Diagram

สัญลักษณ์ UML แสดง Class (ต่อ)

2. ส่วนสำหรับ Attribute

- ชนิดของการเข้าถึง (Visibility) ของ Attribute ได้แก่
  - ชนิด Public แทน +
  - ชนิด Private แทน -
  - ชนิด Protect แทน #

UML : Class Diagram

สัญลักษณ์ UML แสดง Class (ต่อ)

2. ส่วนสำหรับ Attribute

- ชื่อของ Attribute
- ประเภทของ Attribute จะอยู่ต่อจากเครื่องหมาย : เช่น Integer Real
- ค่าเริ่มต้น Attribute จะอยู่ต่อจากเครื่องหมาย =

UML : Class Diagram

สัญลักษณ์ UML แสดง Class (ต่อ)

3. ส่วนสำหรับ Operation

- ชนิดของ Operation เหมือนกับ Attribute
- ชื่อของ Operation
- พารามิเตอร์ ประกอบด้วย ชื่อพารามิเตอร์ : ประเภทพารามิเตอร์
- ประเภทของค่าที่ส่งคืน (Return Type) ซึ่งจะเขียนตามหลัง :

  อาจเป็น Primitive Data Type หรืออาจเป็นคลาสอื่น ๆ ก็ได้

UML : Class Diagram

สัญลักษณ์ UML แสดง Class (ต่อ)

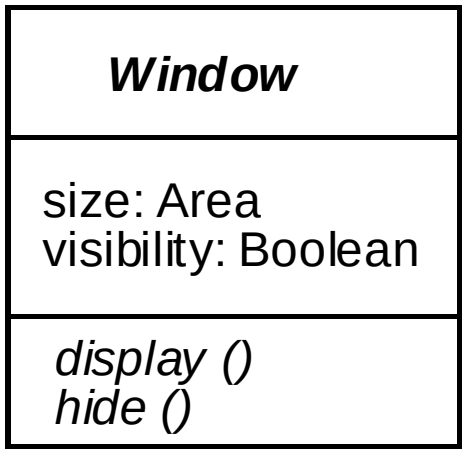
3. **ส่วนสำหรับ Operation**

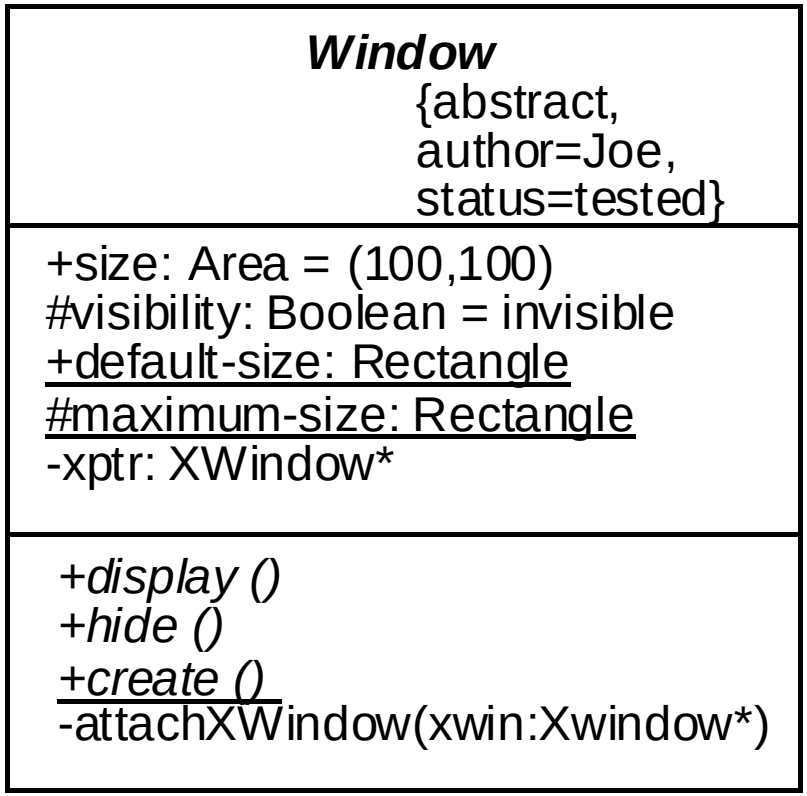
ตัวอย่าง

Car
- number of cars : integer - data:CarData = “Unspecified” - speed : Integer = 0 - direction : Direction
+ drive (speed: Integer, direction:Direction) + getData(): CarData

Classes


Window


Window
size: Area
visibility: Boolean
display ()
hide ()

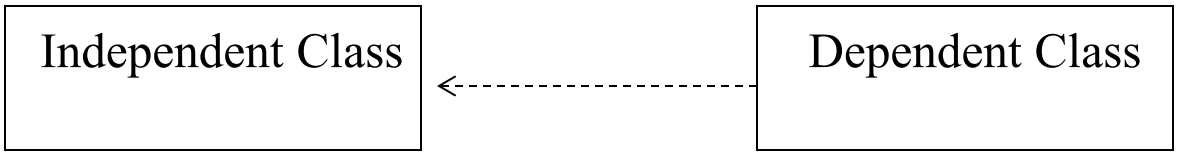

Window
{abstract,
author=Joe,
status=tested}
+size: Area = (100,100)
#visibility: Boolean = invisible
+default-size: Rectangle
#maximum-size: Rectangle
-xptr: XWindow*
+display ()
+hide ()
+create ()
-attachXWindow(xwin:Xwindow*)

UML : Class Diagram

ความสัมพันธ์ระหว่างคลาส (Relationship)

มี 3 รูปแบบ

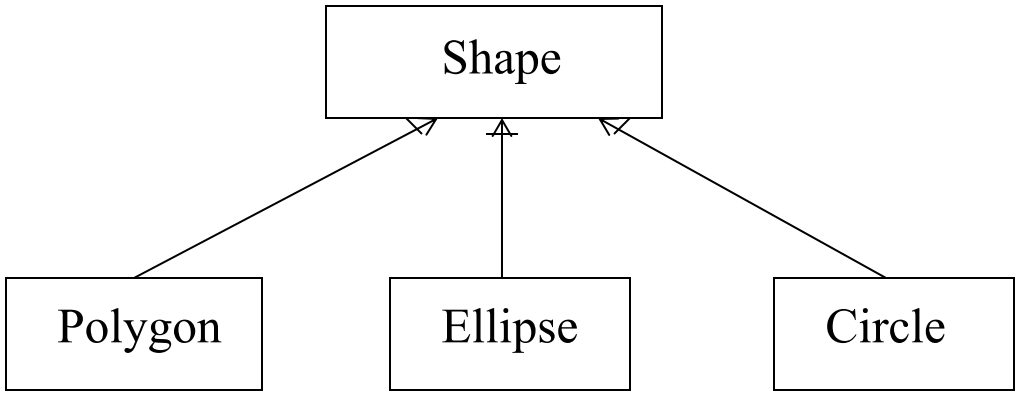
1. Dependency คือความสัมพันธ์แบบพึ่งพิง ความสัมพันธ์แบบนี้เกิดขึ้นเมื่อการเปลี่ยนแปลงที่เกิดขึ้นกับคลาสที่ถูกพึ่งพิง (Independent Class) จะส่งผลต่อคลาสที่พึ่งพิง(Dependent Class) การโมเดลความสัมพันธ์โดยวาดเส้นตรงแบบประที่มีหัวลูกศรเป็นเส้นโปร่งชี้จากซับคลาสที่พึ่งพิงไปยังคลาสที่ถูกพึ่งพิง ดูตัวอย่างจากรูป Dependent Class อาจเป็นค่าใช้จ่าย ส่วน Independent Class อาจเป็นคลาสงบประมาณ



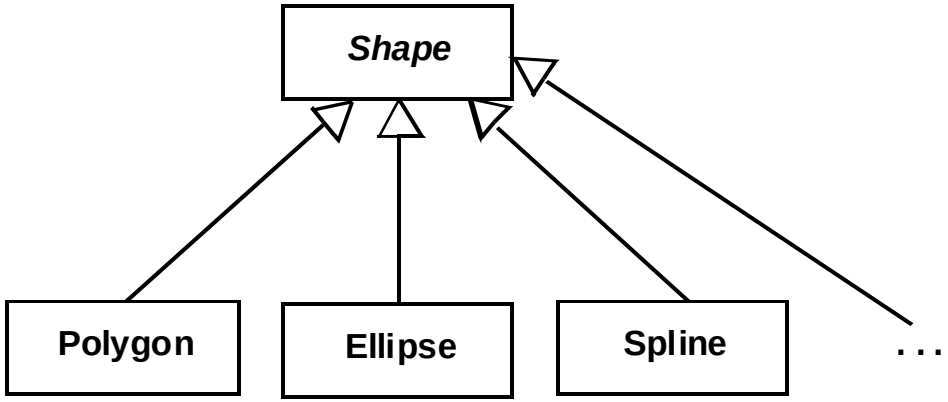
UML : Class Diagram

ความสัมพันธ์ระหว่างคลาส (Relationship)

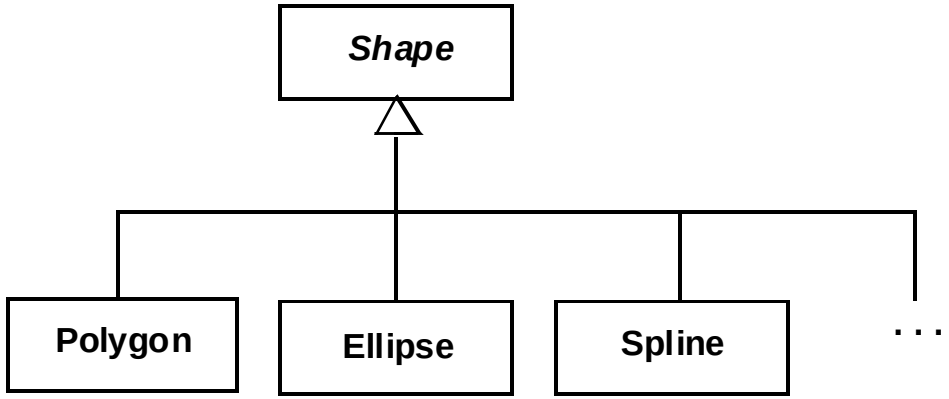
2. Generalization คือความสัมพันธ์ระหว่างซูปเปอร์คลาสและซับคลาสนั่นเอง การโมเดลความสัมพันธ์โดยการวาดเส้นตรงทึบที่มีหัวลูกศรเป็นรูปสามเหลี่ยมโปร่งชี้จากซับคลาสไปยังซูเปอร์คลาส ดังรูป



Generalization



Separate Target Style



Shared Target Style

UML : Class Diagram

ความสัมพันธ์ระหว่างคลาส (Relationship)

3. Association เป็นความสัมพันธ์อีกชนิดหนึ่งระหว่างคลาสซึ่งแบ่งได้ดังนี้

3.1 Normal Association เป็นโมเดลที่ซับซ้อน 2 ทิศทาง

3.2 Aggregation เน้นความสัมพันธ์ในแง่การรวมกันหรือประกอบกัน

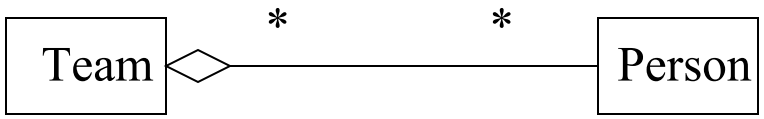
3.1 Normal Association

โมเดลเชิงซับซ้อน มีความสัมพันธ์แบบ 2 ทิศทาง และกำหนดปริมาณของคลาสหรือ Object เรียกว่า Multiplicity ดังรูป

UML : Class Diagram

3.2 Aggregation

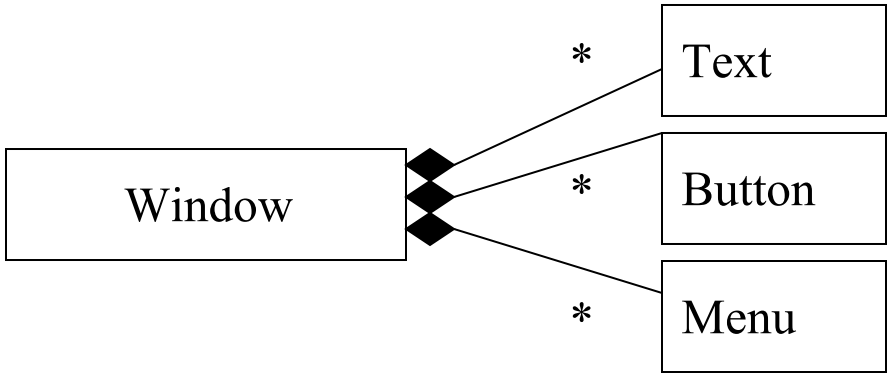
3.2.1 Normal Aggregation แสดงด้วยเส้นตรงทึบโยงระหว่างคลาสโดยมีสัญลักษณ์ข้าวหลามตัดติดอยู่ระหว่างปลายเส้นความสัมพันธ์กับคลาสที่ใหญ่กว่า



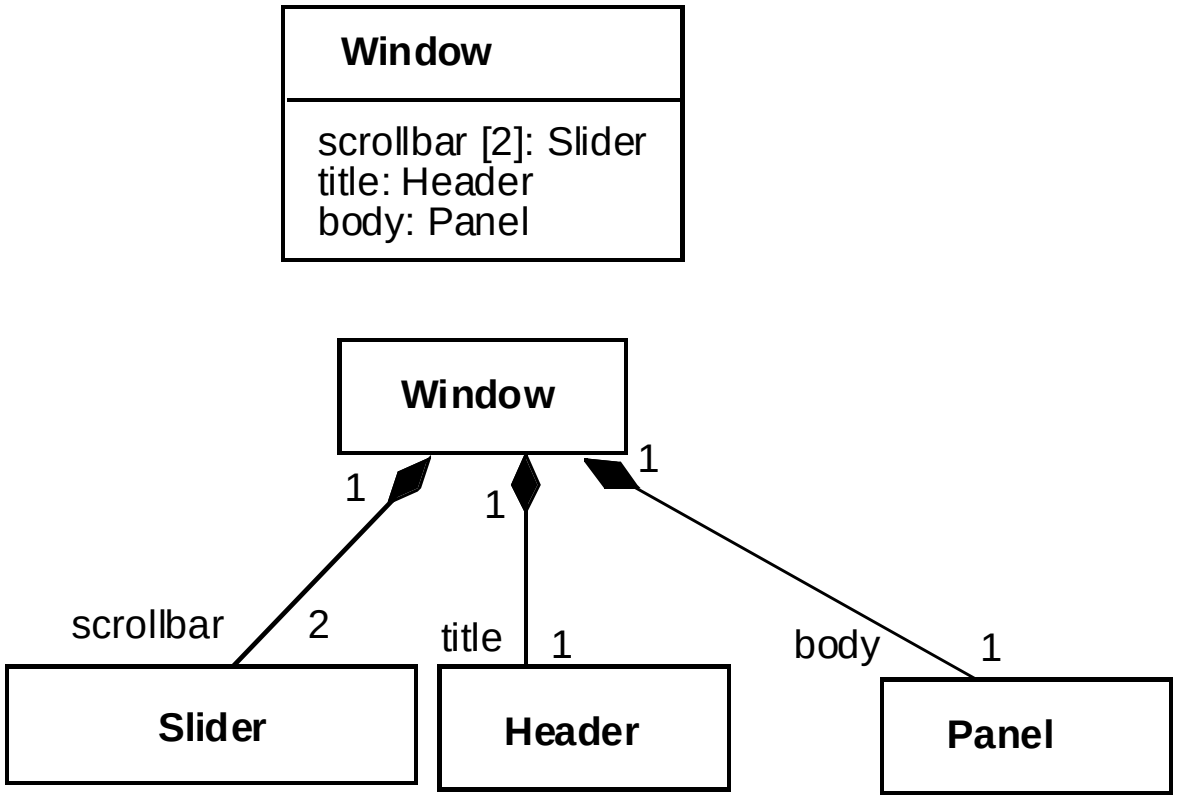
แต่ละทีมประกอบด้วยสมาชิกหลายคนในทางกลับกัน แต่ละคนอาจสังกัดอยู่มากกว่าหนึ่งทีม

UML : Class Diagram

3.2.2 Composition เน้นที่คลาสที่เป็นองค์ประกอบจะเป็นส่วนหนึ่งของคลาสที่ใหญ่กว่า และเมื่อคลาสใหญ่กว่าถูกทำลาย คลาสที่เป็นองค์ประกอบก็จะถูกทำลายไปด้วย เช่น คลาสวิโดวส์ จะประกอบไปด้วยคลาสปุ่ม คลาสเมนู เป็นต้น ซึ่งคลาสเหล่านั้น จะอาศัยอยู่ในวินโดวส์ทั้งหมด และเมื่อวินโดวส์ถูกปิดลงปุ่ม เมนูก็จะหายไปพร้อม ๆ กันด้วย



Composition



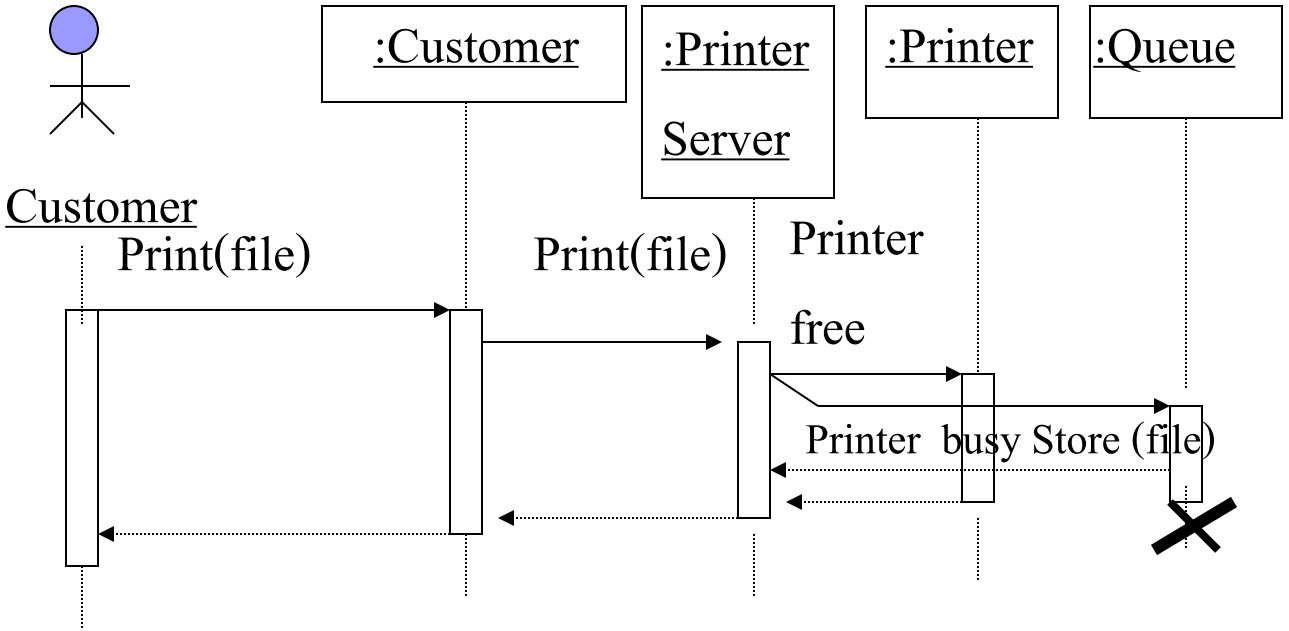
UML : Class Diagram

Package เป็นระบบย่อย (Subsystem) เป็นการจัดกลุ่มของคลาสหรือComponent

สัญลักษณ์ เป็น สี่เหลี่ยมมุมฉากที่ไม่กำหนดขนาดและรูปสี่เหลี่ยมเล็ก(แท็บ, Tab) ซ้อนอยู่ที่มุมบนด้านซ้ายสุด พร้อมกับชื่อPackage

Sequence Diagram

บอกว่าใน Use Case นั้น วัตถุแต่ละตัวจะติดต่อสื่อสารกันอย่างไร มีขั้นตอนทำงานอย่างไร โดยจะเน้นไปที่แกนเวลาเป็นสำคัญ ถ้าเวลาเปลี่ยน ขั้นตอนการทำงานจะเปลี่ยน โดยมี Actor เป็นผู้กระทำเริ่มต้น

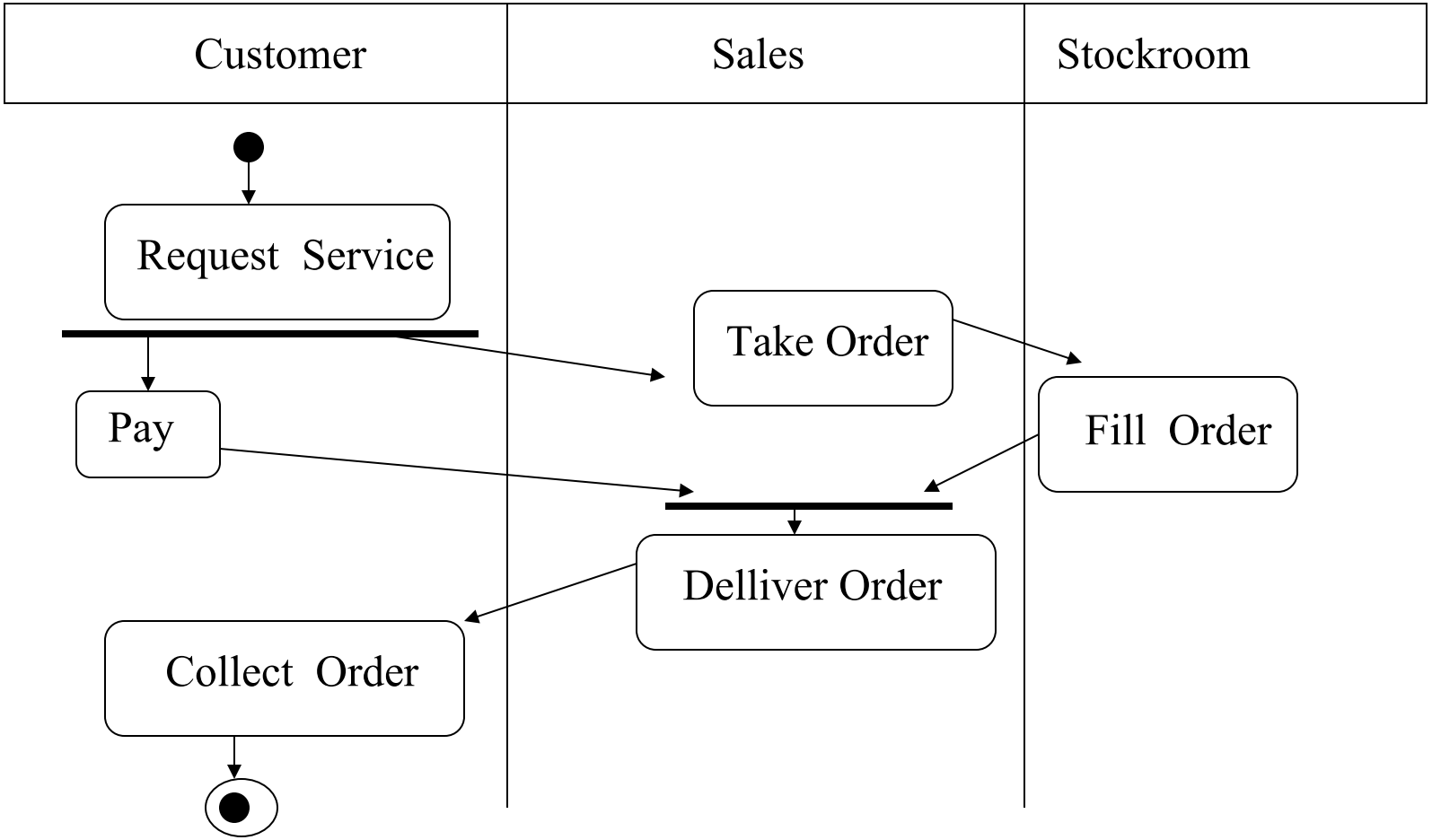


Activity Diagram

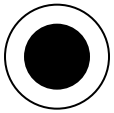
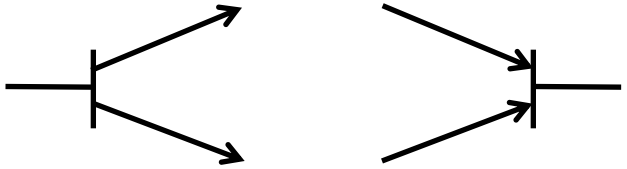
แสดงขั้นตอนการทำงานของ Use Case แต่มันจะเปลี่ยนสถานะตามกระบวนการทำงานคล้ายกับ Flowchart

ข้อดี คือ สามารถแสดงถึงการทำงานในวัตถุนั้น ๆ อย่างละเอียดคล้าย Flowchart และมีการแบ่งแยกหมวดหมู่งานตาม Object

Activity Diagram

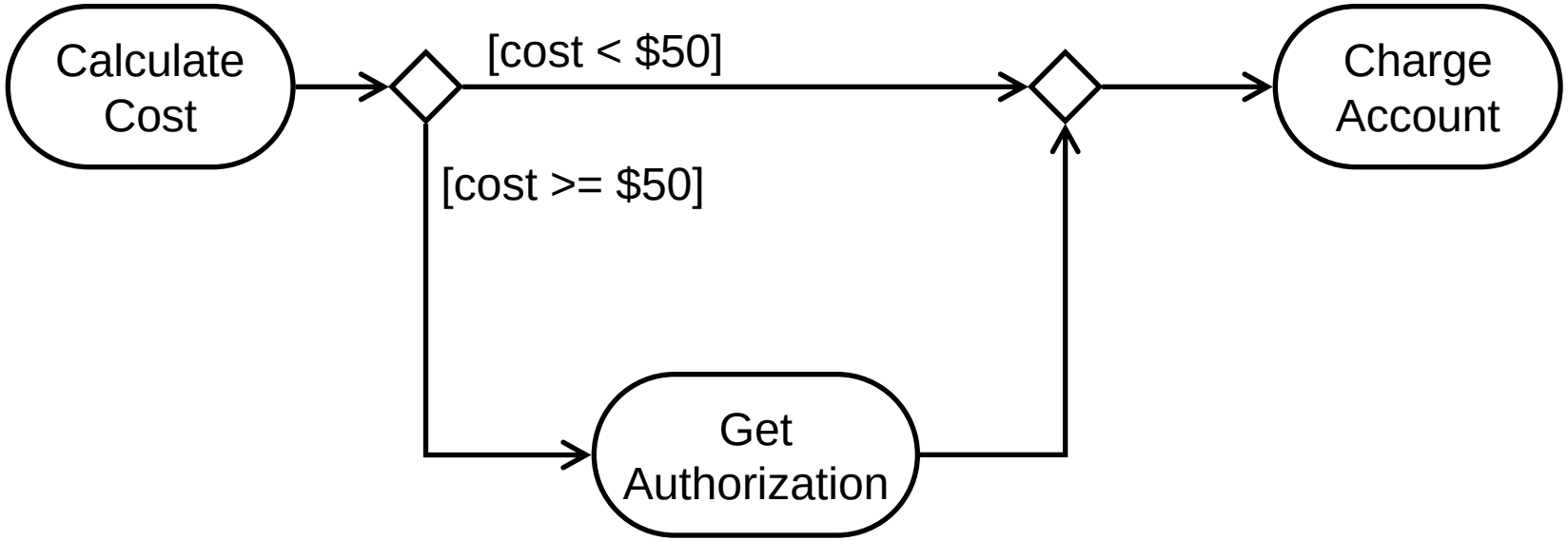


Coordinating Steps

- Inherited from state machines
- Initial state 
- Final state 
- Fork and join 

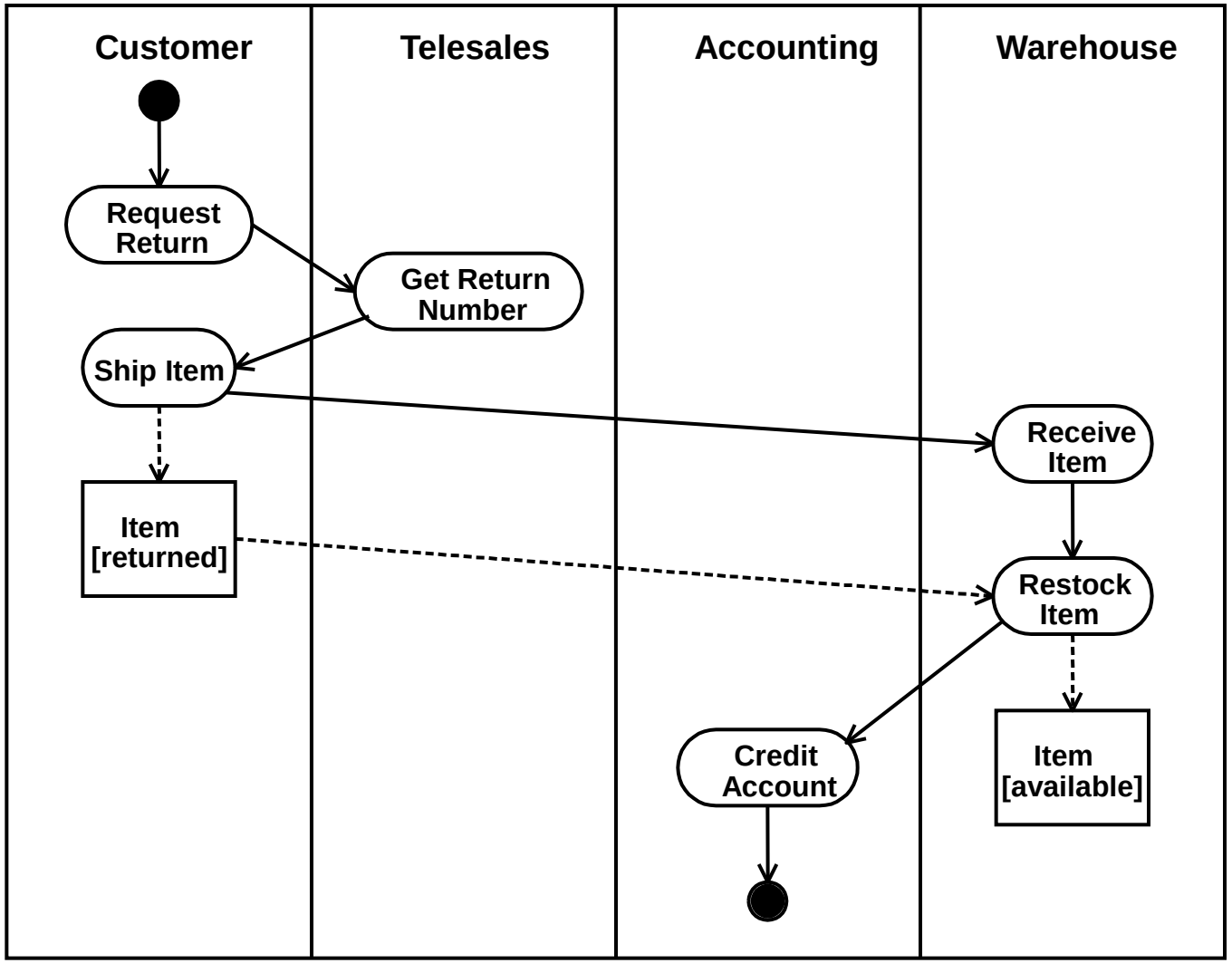
Coordinating Steps

- Decision point and merge (◇) are inherited from state machines.
- For modeling conventional flow chart decisions.



Activity Diagram Modeling Tips

From UML User Guide:



Activity Diagram ของการตรวจสอบผู้ใช้บริการ