Software Engineering

System Modeling

Todsapon Banklongsi
Department of Computer Engineering
Bangkok University

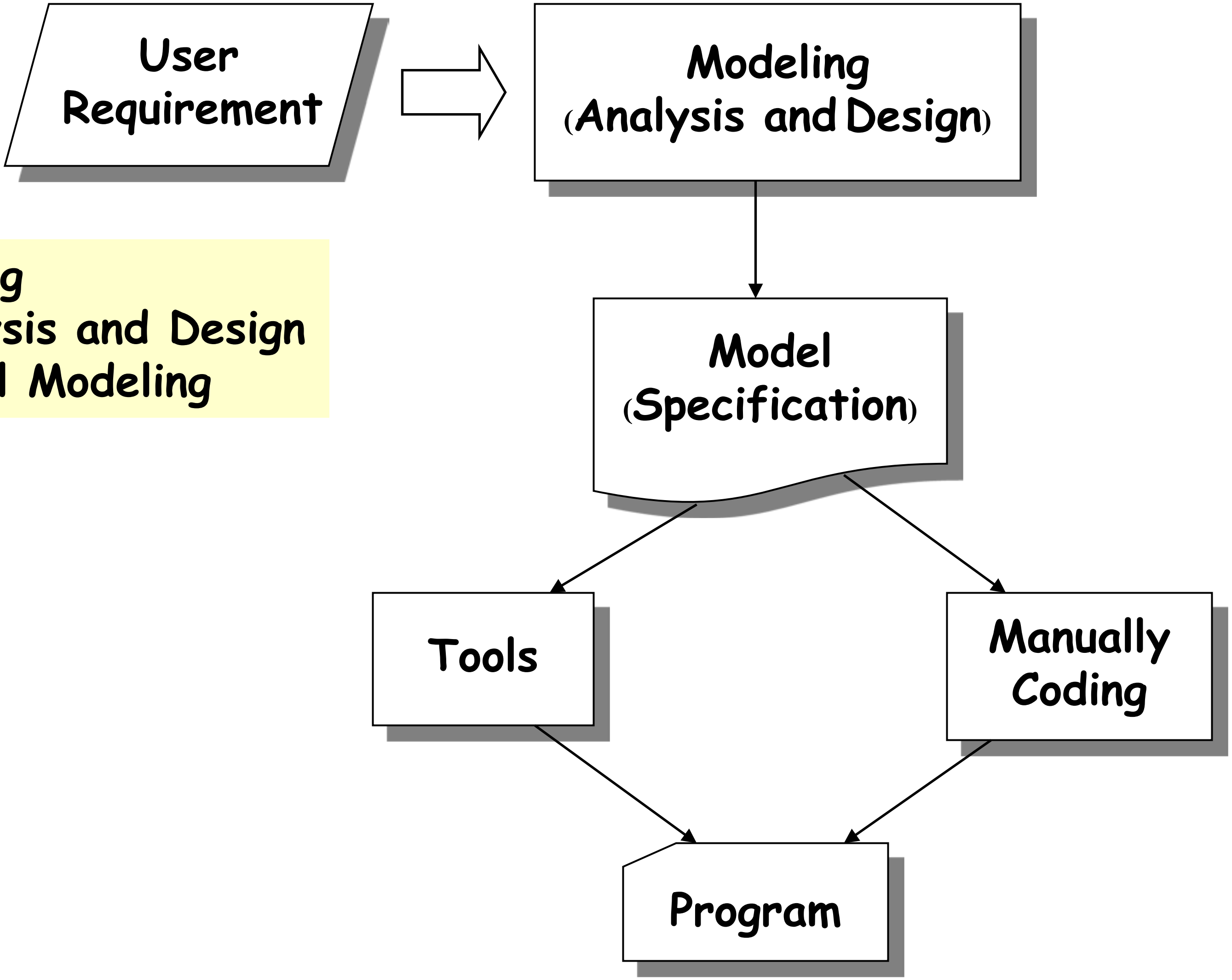
System Modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML)
- System modeling helps the analyst to understand the functionality of the system and models are used to communicate with customers

Existing and planned system models

- Models of the existing system are used during requirements engineering. They help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
- In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

Software Modeling



Modeling
- **Analysis and Design**
- **Visual Modeling**

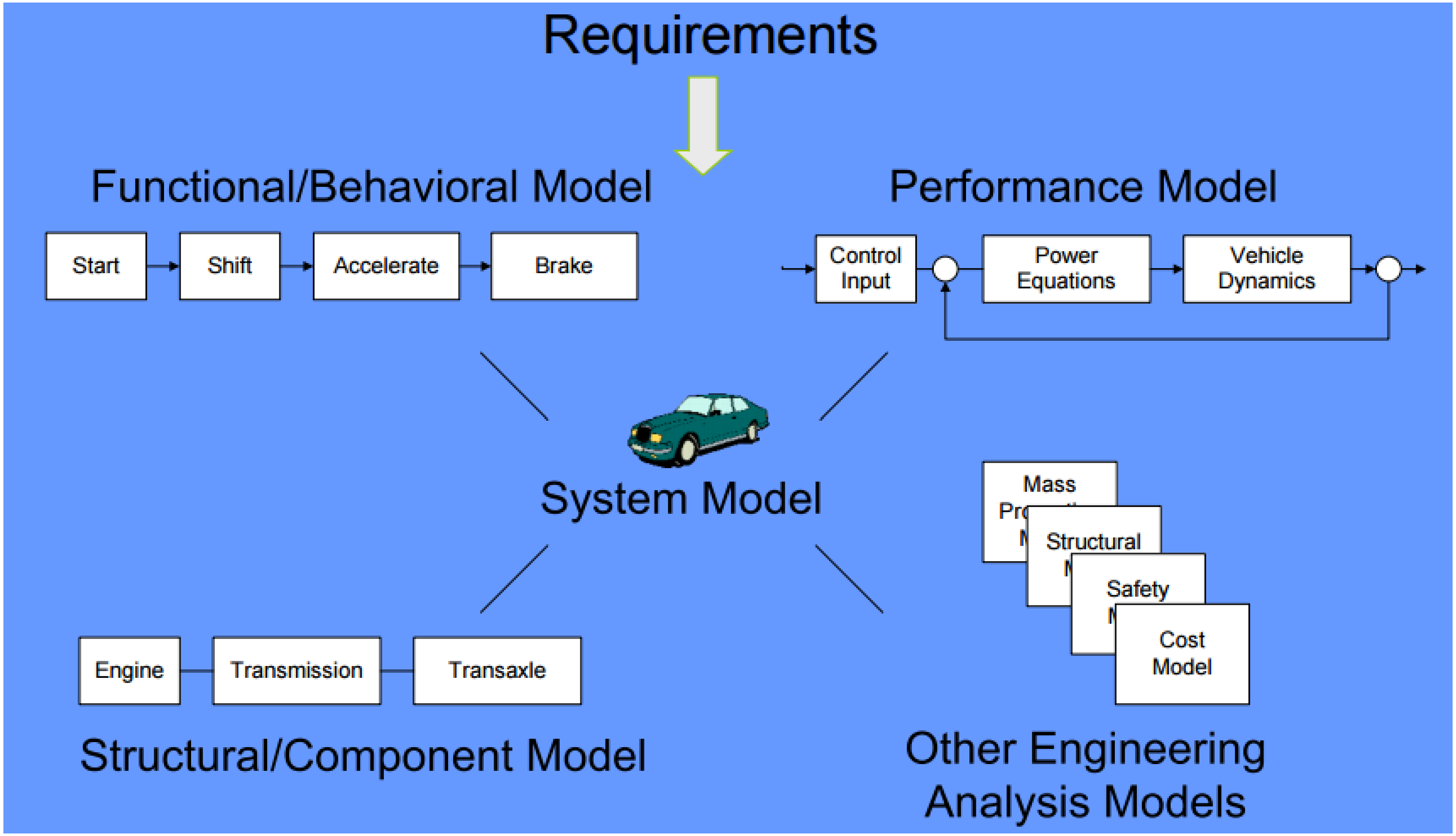
Software Development Process

- Requirement Specification : define problem domain
- **Analysis : what problem to be solved?**
- Design : how to solve the problem?
- Implementation : how to implement the solution?
- Testing : how to ensure that the solution can solve the problem?
- Maintenance : how to adjust the solution to accomodate change?
- Retirement : when does the system to be retired?

Software modeling and models

- Software modeling helps the engineer to understand the functionality of the system
- Models are used for communication among stakeholders
- Different models present the system from different perspectives
  - External perspective showing the system's context or environment
  - Process models showing the system development process as well as activities supported by the system
  - Behavioural perspective showing the behaviour of the system
  - Structural perspective showing the system or data architecture

System Model

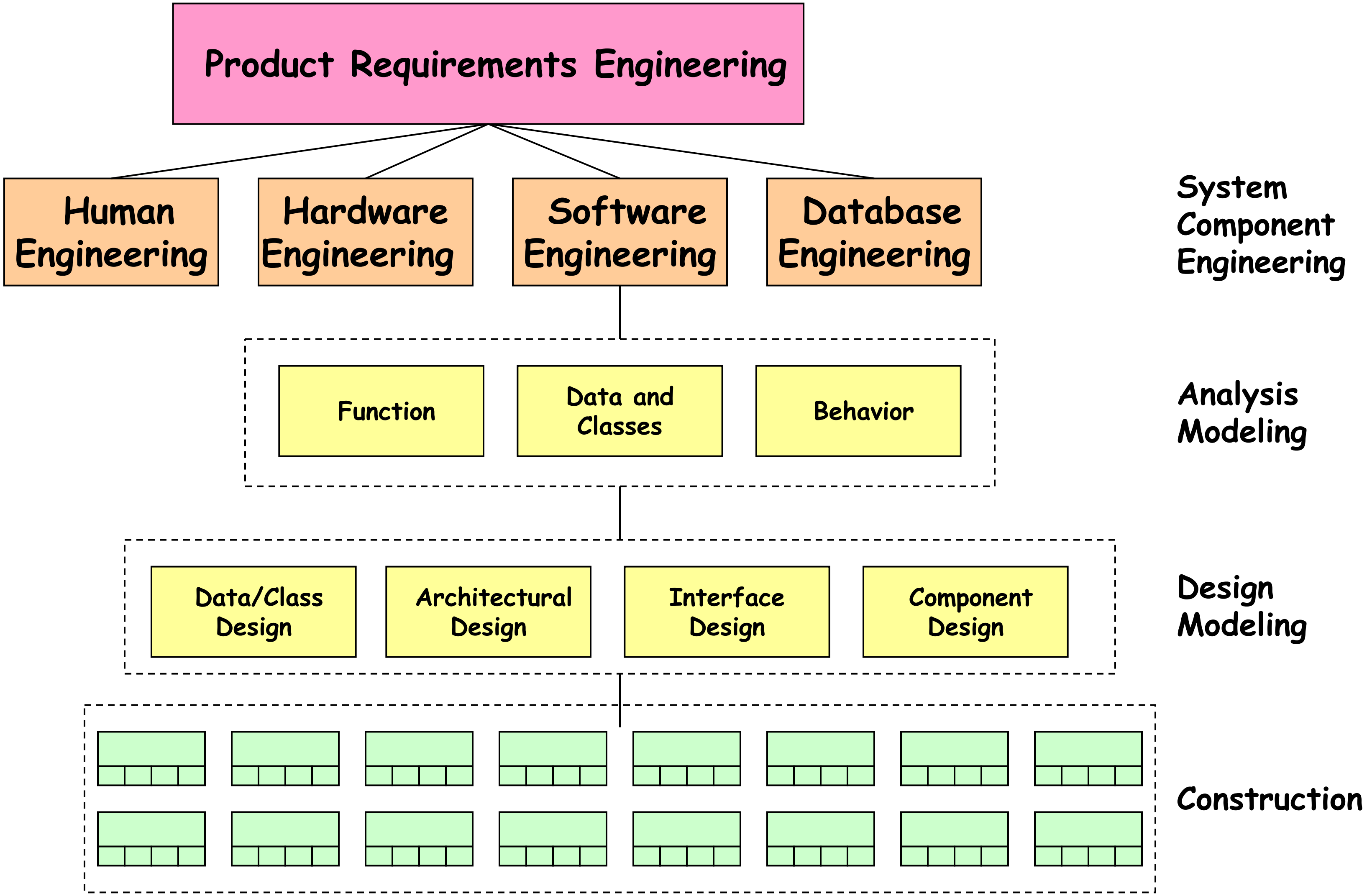


http://www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf

System perspectives

- An external perspective, where you model the context or environment of the system
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system
- A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events

Product Engineering Hierarchy



Product Requirements Engineering

Human Engineering

Hardware Engineering

Software Engineering

Database Engineering

System Component Engineering

Function

Data and Classes

Behavior

Analysis Modeling

Data/Class Design

Architectural Design

Interface Design

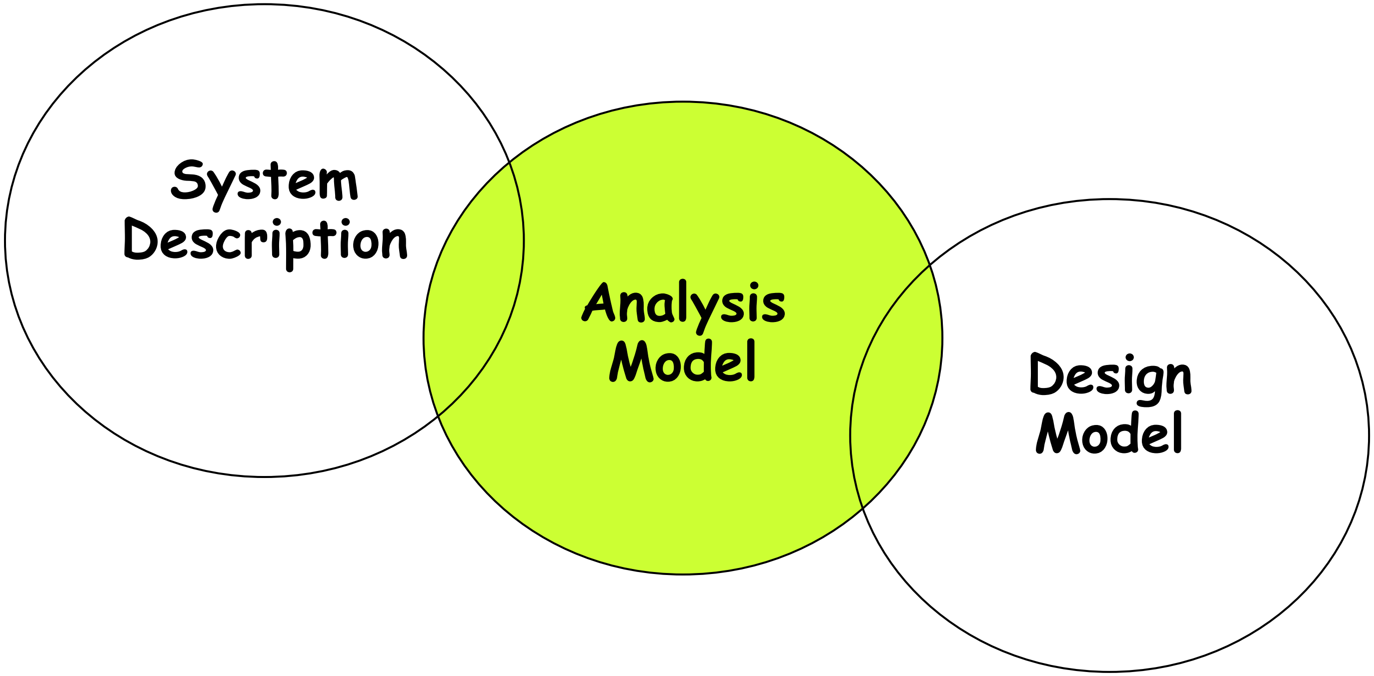
Component Design

Design Modeling

Construction

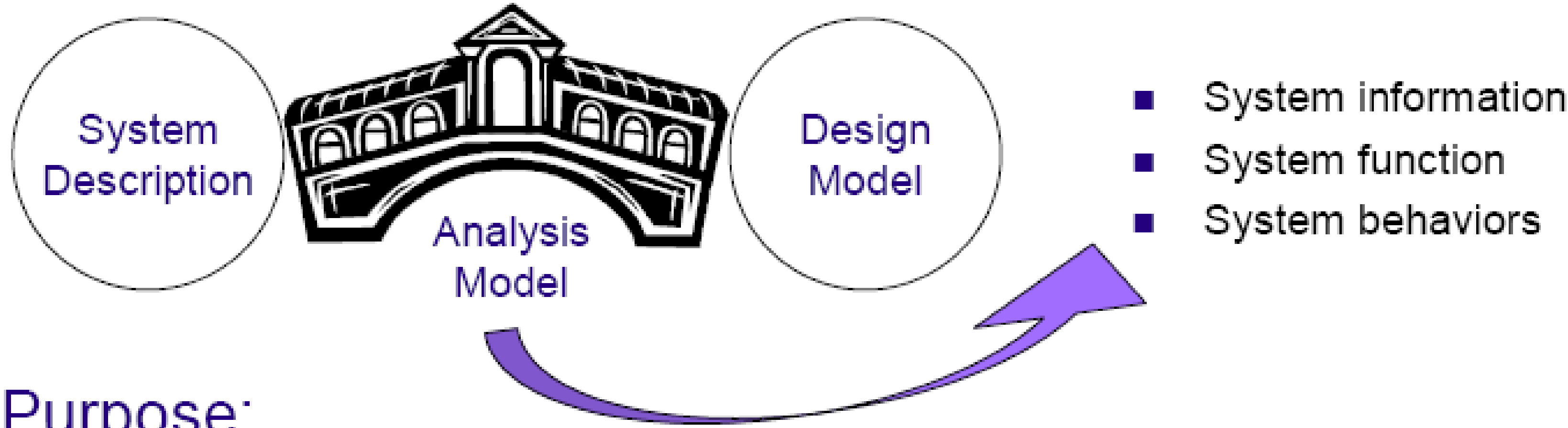
Analysis Modeling

The Analysis Model is the first technical representation of a system. Analysis modeling uses a combination of text and diagrams to represent software requirements (data, function, and behavior) in an understandable way.



The Analysis Model

The analysis model consists of a wide variety of diagrammatic forms used to bridge an important gap.



- System information
- System function
- System behaviors

Purpose:

- Describe what the customer wants built
- Establish the foundation for the software design
- Provide a set of validation requirements

Analysis Modeling Approaches

Structured Analysis:

- Models data elements
  - □ Attributes
  - □ Relationships
- Models processes that transform data

Object-Oriented Analysis

- Models analysis classes
  - □ Data
  - □ Processes
- Models class collaborations

Techniques from both approaches are typically used in practice.

Types of Analysis Model

1. Structural Analysis or
Non-UML System Modeling Methods

- □ **Process Model** **(process-driven systems)**
- - Data Flow Diagram (DFD)
- - Flowcharts
- - Structure Charts
- - Decision Table, Decision Tree

- □ **Data Model** **(data-driven systems)**
- - Entity Relationship Diagram (ER Diagram)
- - Data Dictionary
- - Warneir Diagram

- □ **Control-Oriented Methods** **(real-time systems)**
- - State Transition Diagrams (STD)

2. Object Oriented Analysis or
UML System Modeling Methods

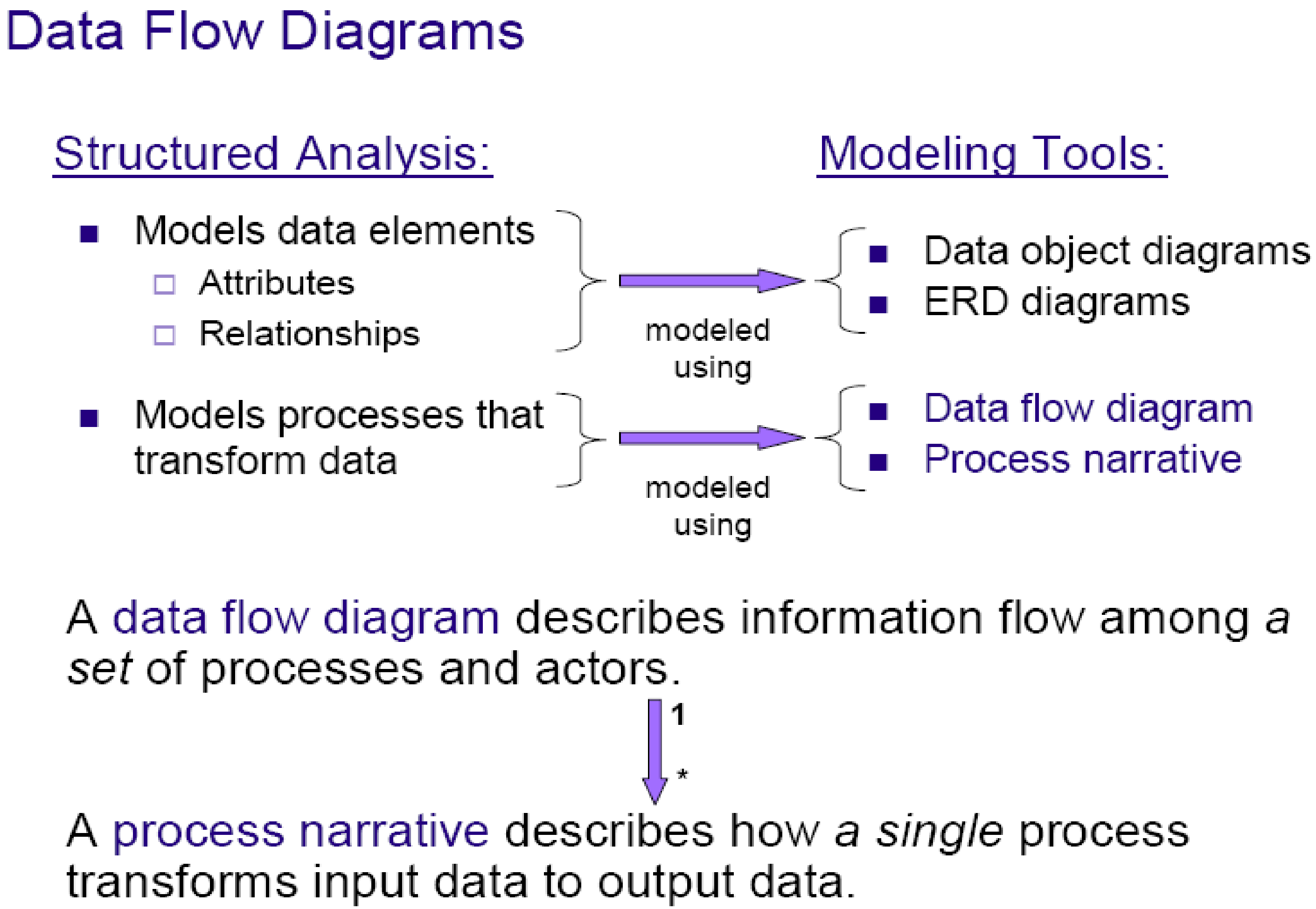
- □ **Structural Diagram**
- - Class Diagram
- - Object Diagram
- - Component Diagram
- - Deployment Diagram

- □ **Behavioral Diagram**
- - Use Case Diagram
- - Sequence Diagram
- - Activity Diagram
- - Collaboration Diagram
- - State Diagram

Structural Analysis

- Structured analysis: the focus is only on process and procedures. Modeling techniques used in it are DFD(Data Flow Diagram), Flowcharts etc.
- Structuring system process requirements
  - Data flow diagrams (DFD) - process modeling
  - Context diagram
  - Process decomposition (DFD levels): 4 types of DFD:
    - Current physical: adequate detail only
    - Current logical: enables analysts to understand current system
    - New logical: technology independent, show data flows, structure, and functional requirements of new system.
    - New physical: technology dependent.
  - Logical modeling: using structured English, decision table/tree
  - Structuring system data requirements: using ER diagram

Data Flow Diagram : DFD



Data Flow Diagrams

Structured Analysis:

- Models data elements
  - □ Attributes
  - □ Relationships
- Models processes that transform data

Modeling Tools:

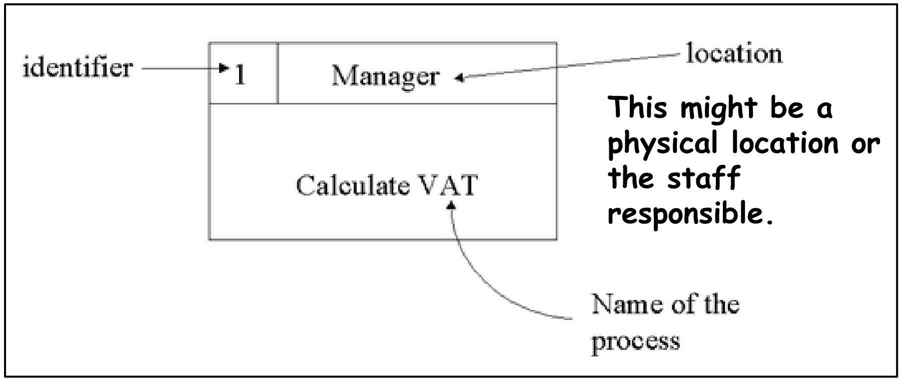
- Data object diagrams
- ERD diagrams
- Data flow diagram
- Process narrative

A data flow diagram describes information flow among *a set* of processes and actors.

A process narrative describes how *a single* process transforms input data to output data.

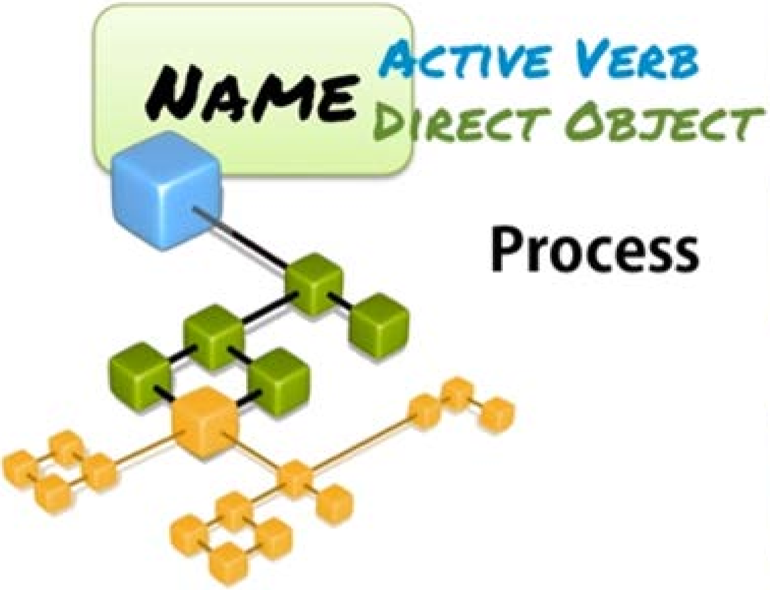
Data Flow Diagram : DFD

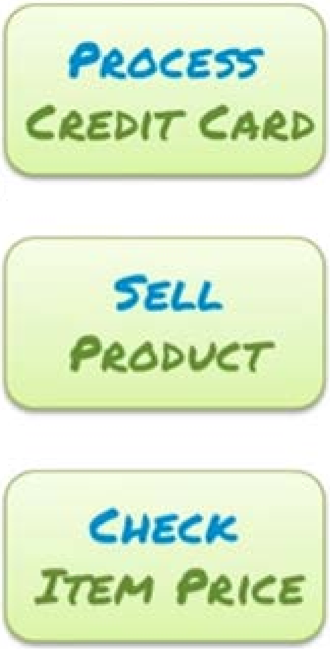
1. Process



This might be a physical location or the staff responsible.

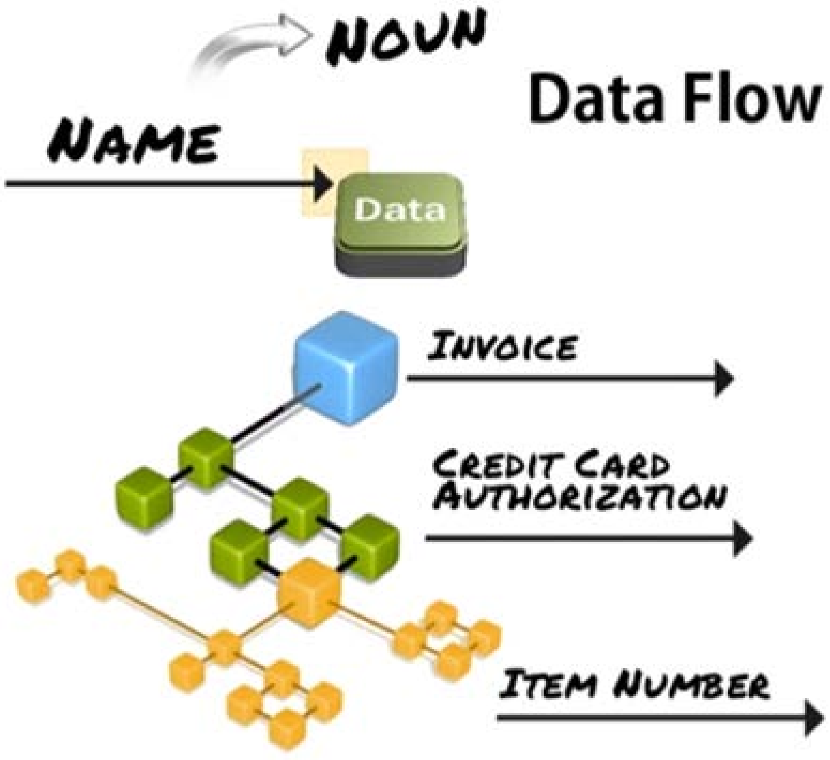
OR SOMETIMES A CIRCLE

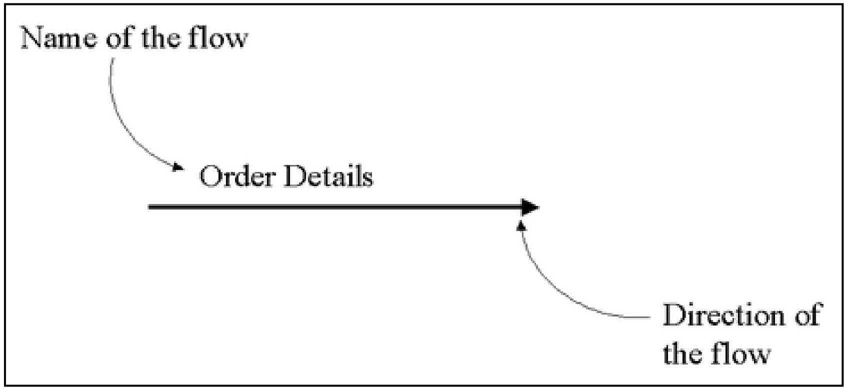




Data Flow Diagram : DFD

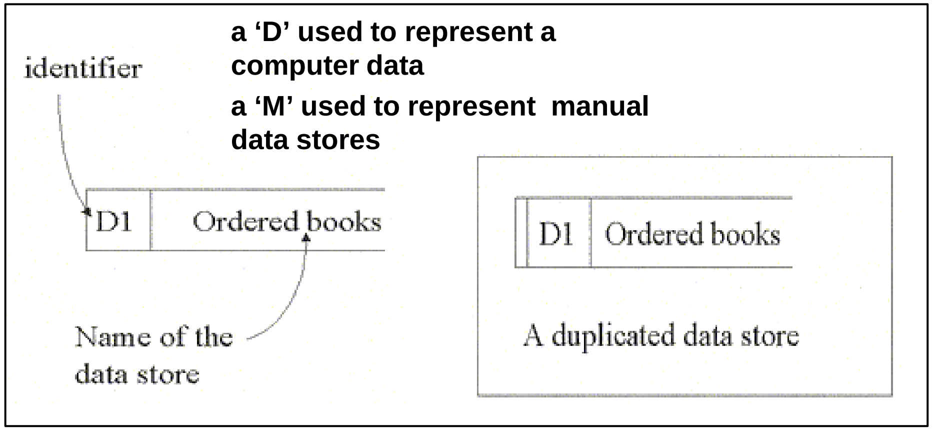
2. Data Flow





Data Flow Diagram : DFD

3. Data Store



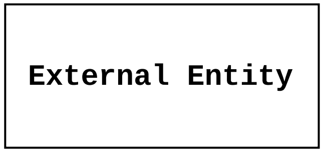
a ‘D’ used to represent a computer data

a ‘M’ used to represent manual data stores

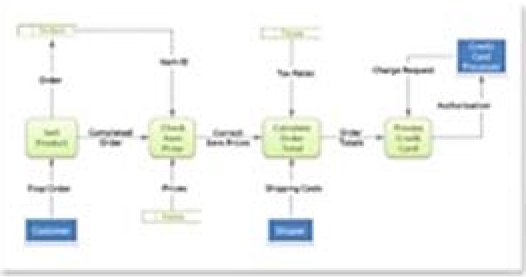
Data Store
OR SOMETIMES 2 LINES
I NEED YOU!
Data at rest
Data at rest

Data Flow Diagram : DFD

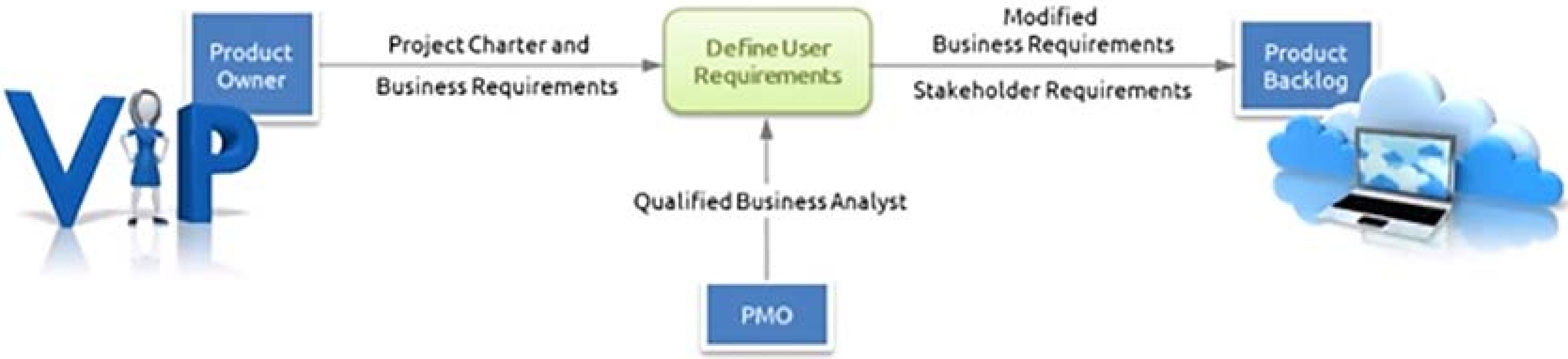
4. External Entity



External Entity

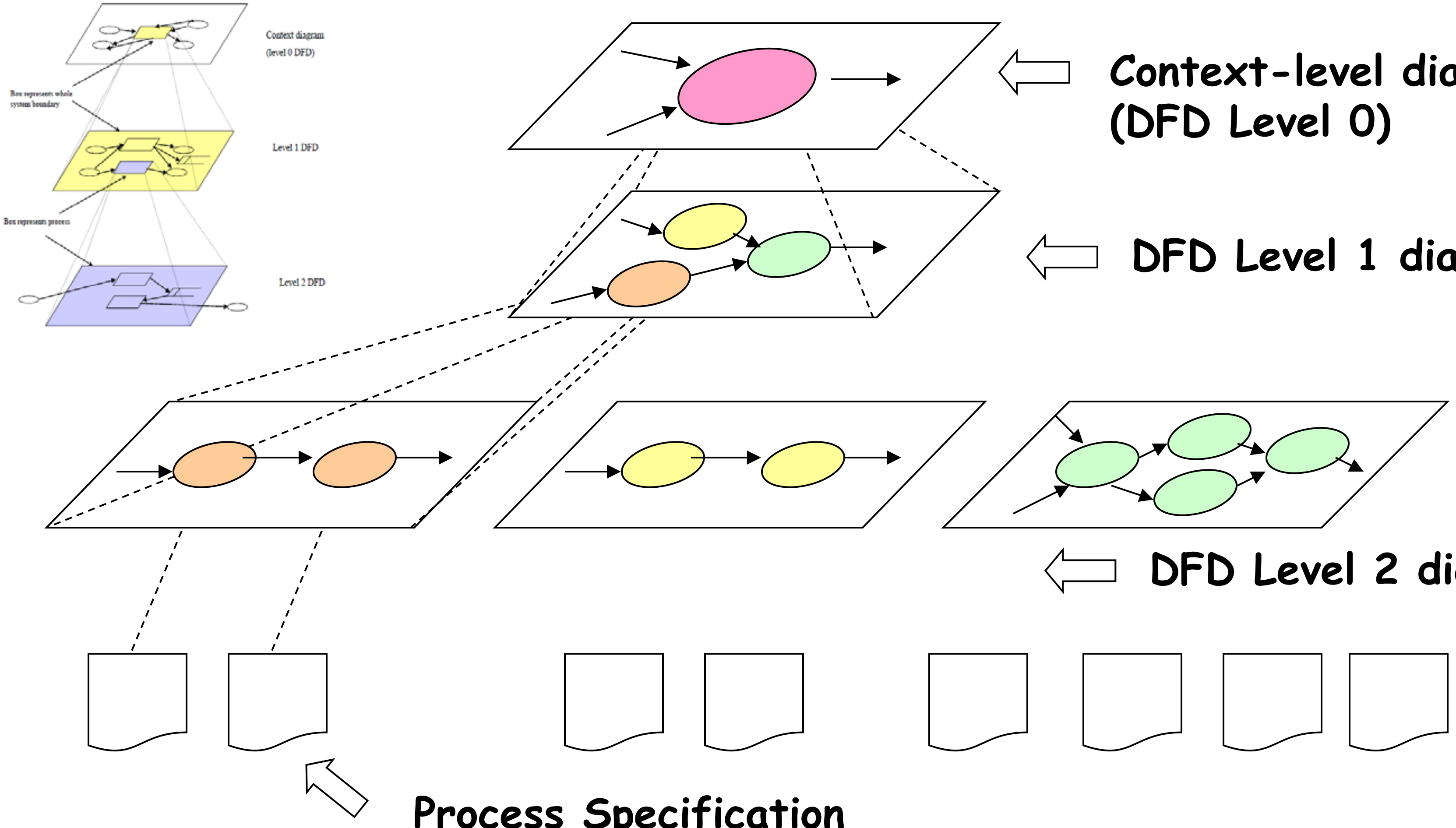


External Entity



Data Flow Diagram : DFD

Diagram Layering and Process Refinement



Context-level diagram (DFD Level 0)

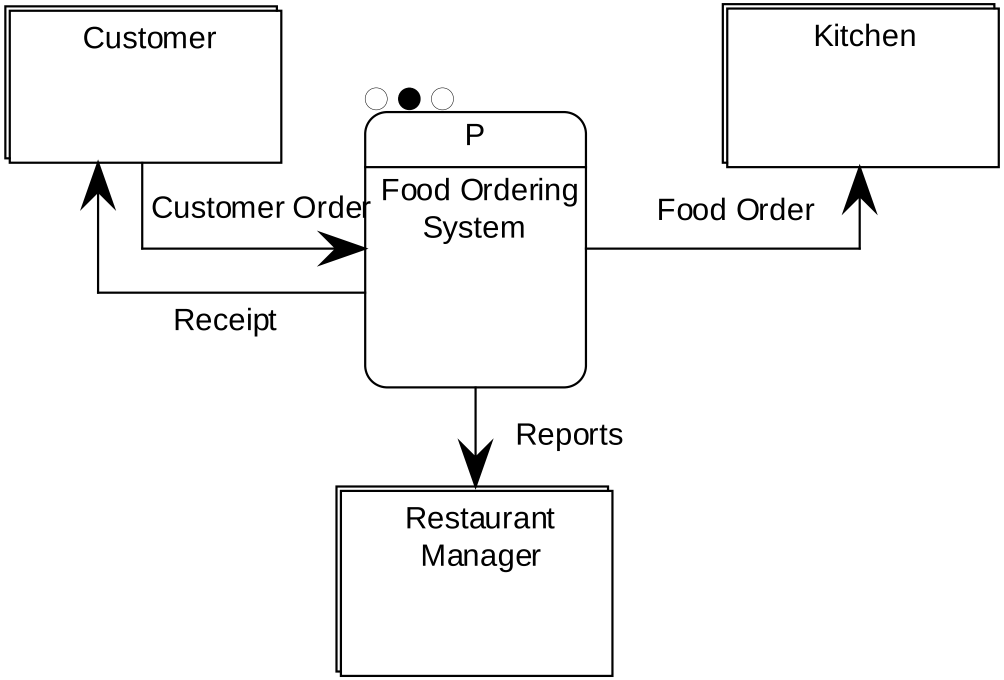
DFD Level 1 diagram

DFD Level 2 diagram

Process Specification

Context Diagram (DFD Level 0)

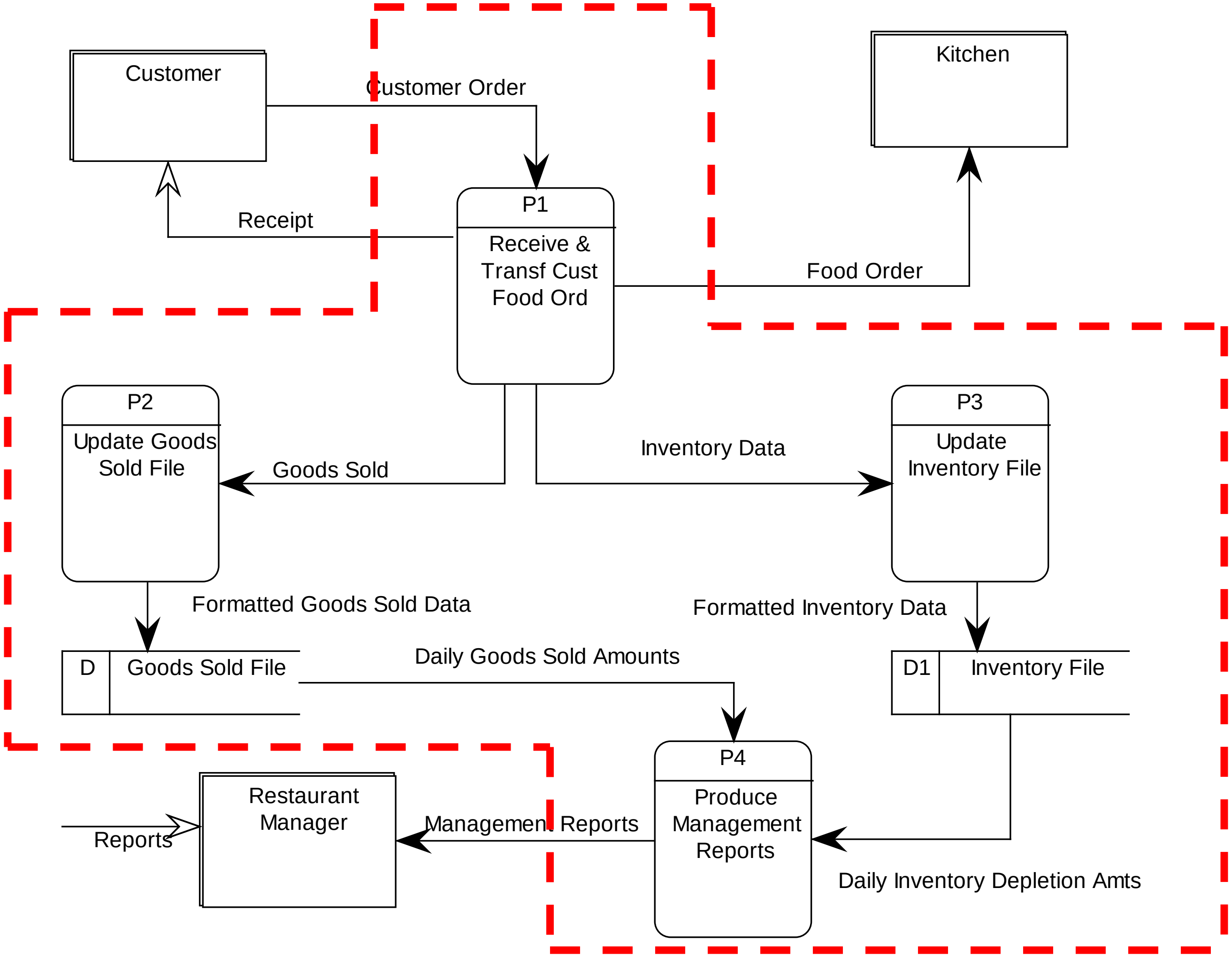
DFD Context Diagram - Example Food Ordering System



- Highest level view of the system
- Contains ONLY one process, i.e., the “system”
- It also shows all external data sources/sinks
- (“electronic” or “manual”)
- And all data flows between data sources/sinks and the process
- It contains NO data stores

DFD Level 1

DFD Level 1 Diagram - Food Ordering System

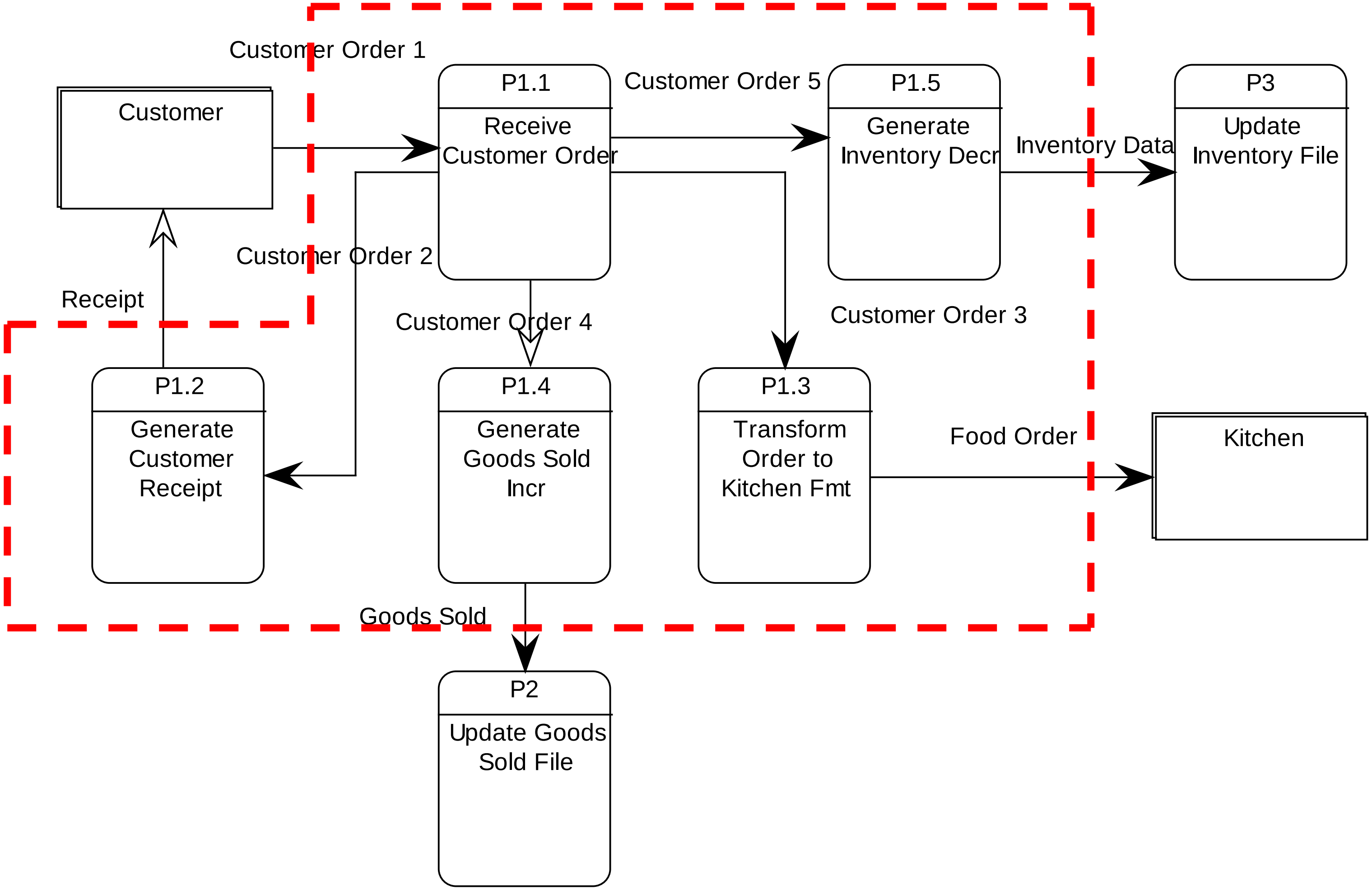


- Expands the main process from context diagram
- Represents the system's major processes
- Which are the primary individual processes at the highest possible level
- This is called "functional decomposition"

DFD Level 2

(DFD Level 1 of Process 1)

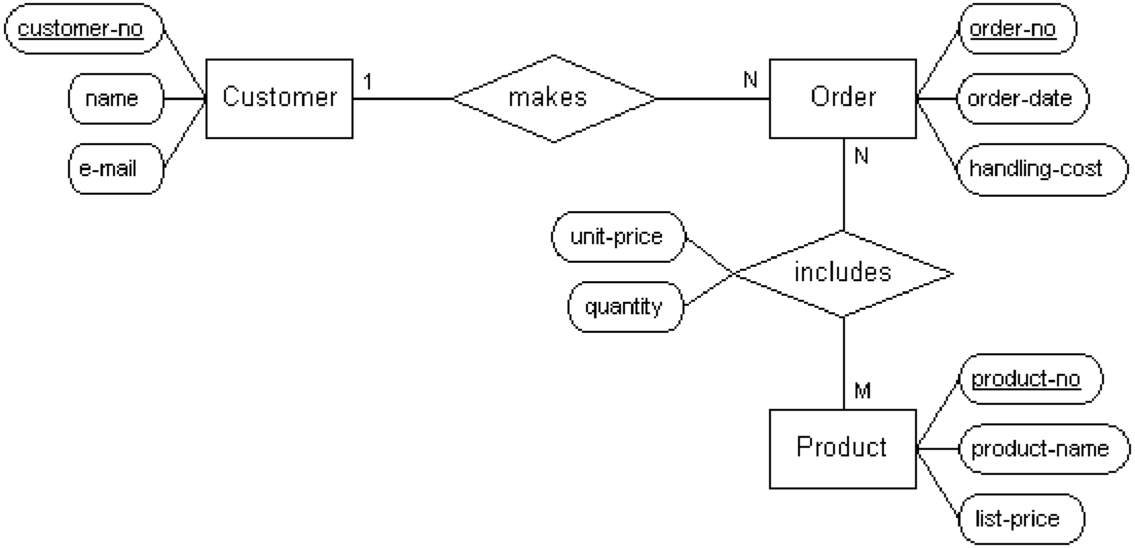
DFD Level 2 Diagram (DFD Level 1 of P1)- Food Ordering System



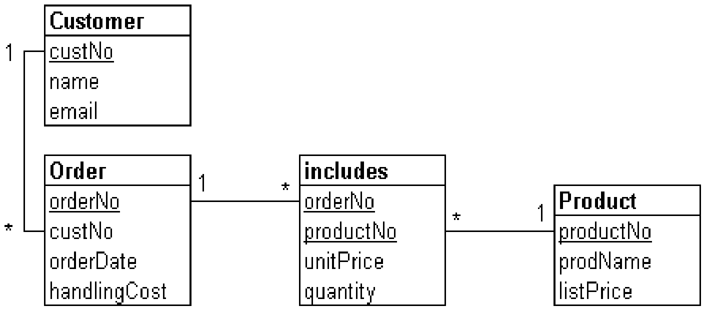
Entity Relationship Diagram (ERD)

- An entity-relationship diagram (ERD) is a graphical representation of an information system that shows the relationship between people, objects, places, concepts or events within that system. An ERD is a data modeling technique that can help define business processes and can be used as the foundation for a relational database.

Entity-Relationship Diagrams

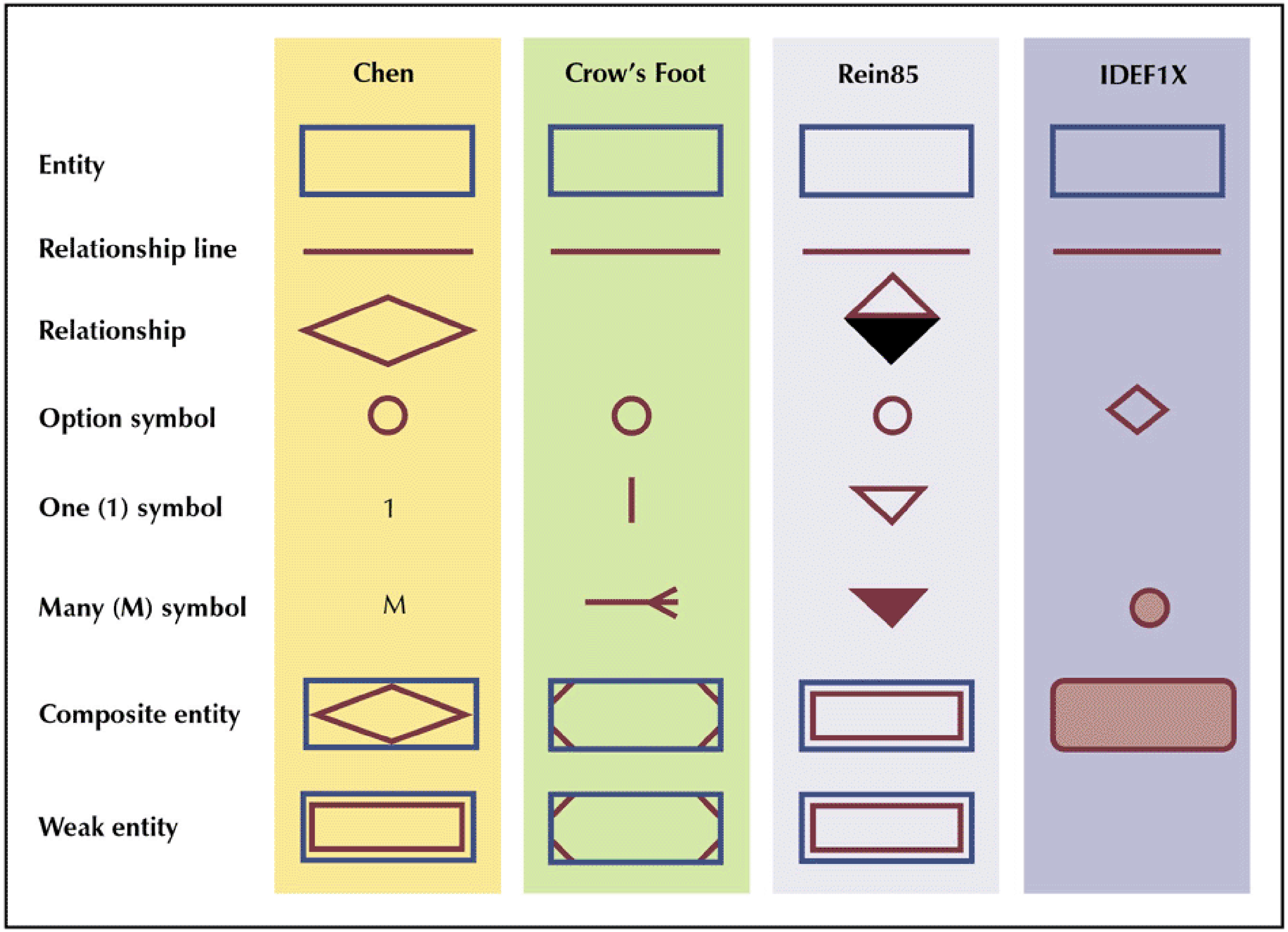


Database Structure Diagrams



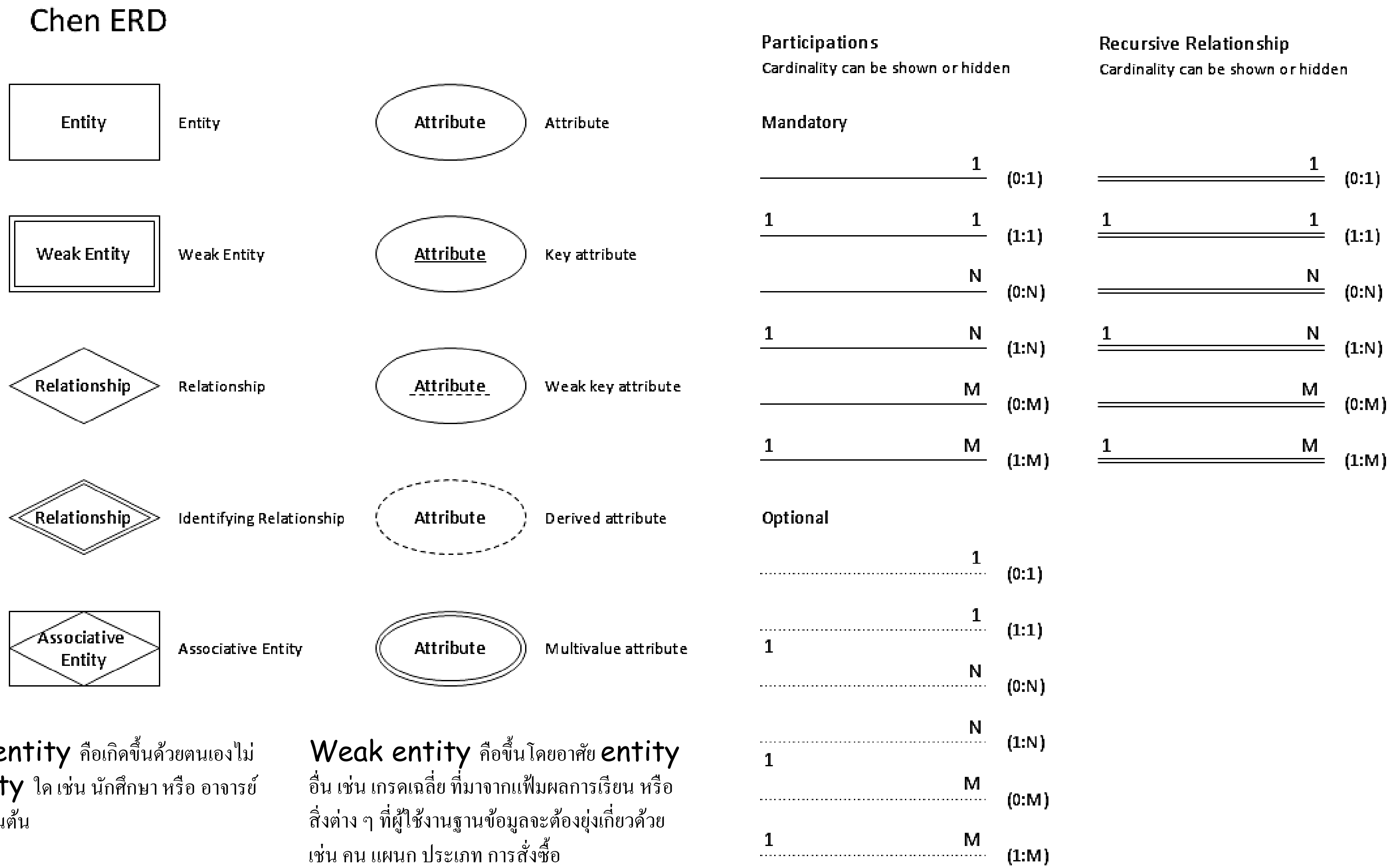
Entity Relationship Diagram (ERD)

FIGURE 4.31 A Comparison of ER Modeling Symbols



	Chen	Crow's Foot	Rein85	IDEF1X
Entity	▭	▭	▭	▭
Relationship line	—	—	—	—
Relationship	◇		⬙	
Option symbol	○	○	○	◇
One (1) symbol	1	|	▽	
Many (M) symbol	M	—<	▼	●
Composite entity	◇	▭	▭	▭
Weak entity	▭	▭	▭	

Entity Relationship Diagram (ERD)



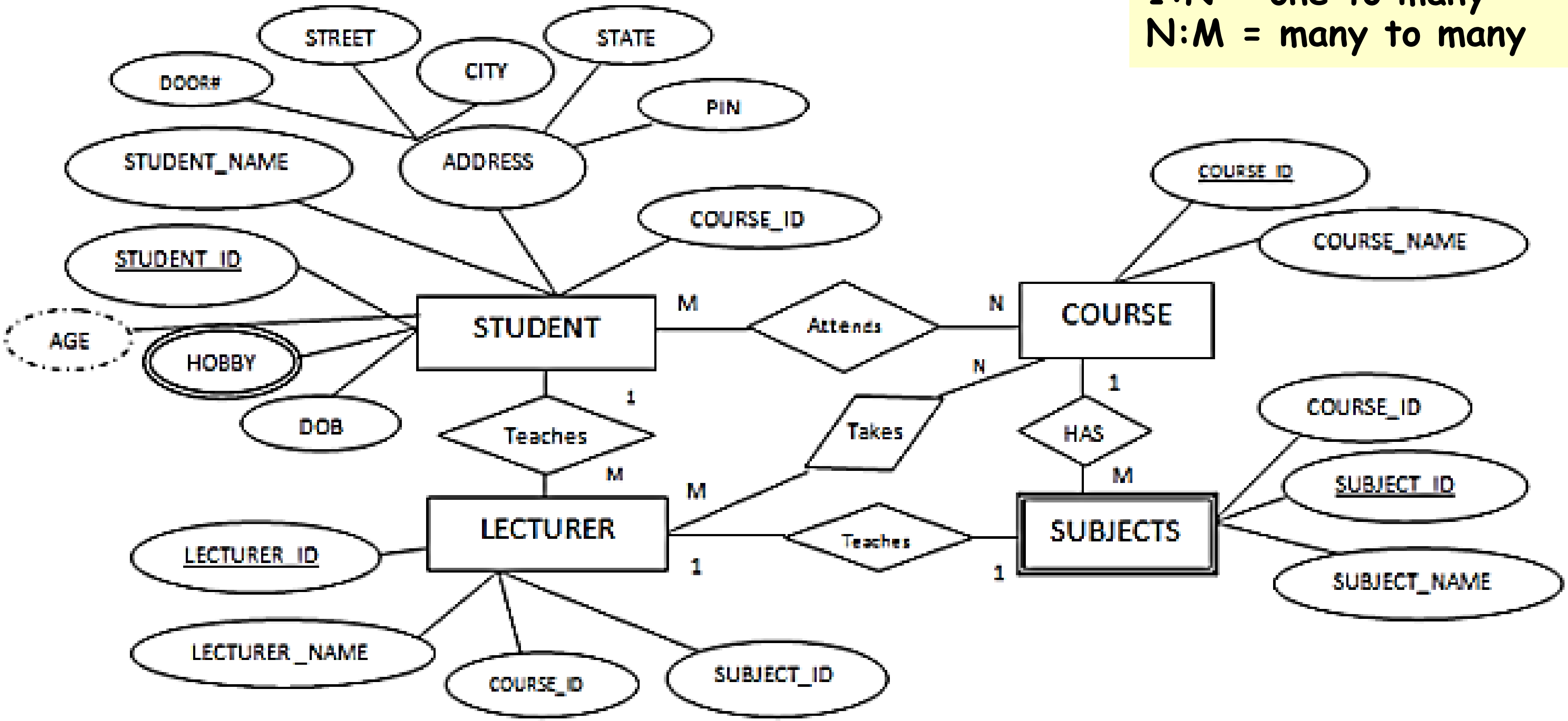
Strong entity คือเกิดขึ้นด้วยตนเองไม่ขึ้นกับ entity ใด เช่น นักศึกษา หรือ อาจารย์ หรือสินค้า เป็นต้น

Weak entity คือขึ้นโดยอาศัย entity อื่น เช่น เกรดเฉลี่ย ที่มาจากแฟ้มผลการเรียน หรือ สิ่งต่าง ๆ ที่ผู้ใช้งานฐานข้อมูลจะต้องยุ่งเกี่ยวด้วย เช่น คน แผนก ประเภท การสั่งซื้อ

http://www.conceptdraw.com/How-To-Guide/picture/Design_Elements(Chen-ERD).png

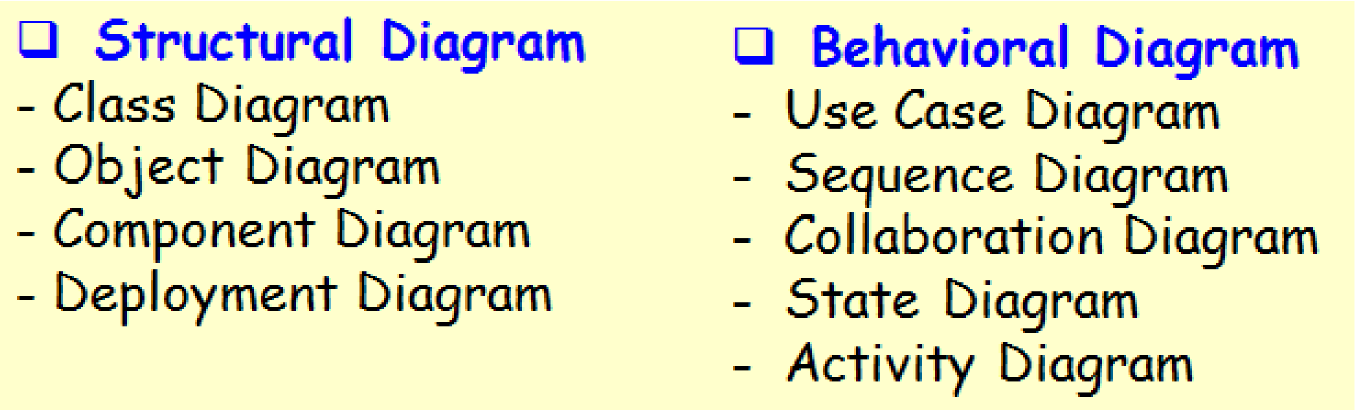
Entity Relationship Diagram (ERD)

1:1= one to one
1:N = one to many
N:M = many to many



Object Oriented Analysis

- In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified
- A semiformal analysis technique for object-oriented paradigm Structuring system process requirements



❑ **Structural Diagram**

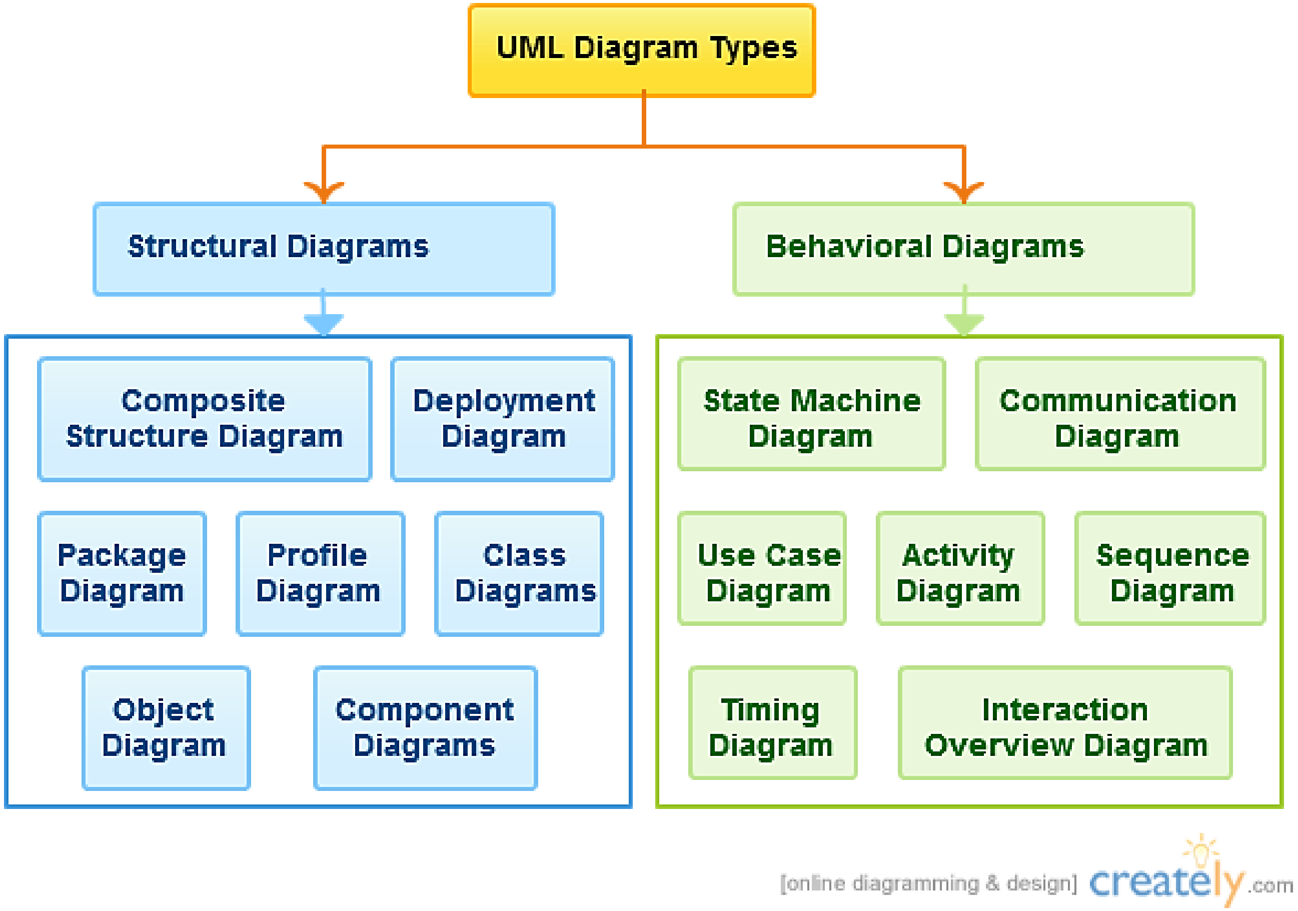
- Class Diagram
- Object Diagram
- Component Diagram
- Deployment Diagram

❑ **Behavioral Diagram**

- Use Case Diagram
- Sequence Diagram
- Collaboration Diagram
- State Diagram
- Activity Diagram

The Unified Modeling Language

- Devised by the developers of object-oriented analysis and design methods
- Has become an effective standard for software modelling



UML Diagram Types

Structural Diagrams

Composite Structure Diagram

Deployment Diagram

Package Diagram

Profile Diagram

Class Diagrams

Object Diagram

Component Diagrams

Behavioral Diagrams

State Machine Diagram

Communication Diagram

Use Case Diagram

Activity Diagram

Sequence Diagram

Timing Diagram

Interaction Overview Diagram

UML-Structural Diagram

- Class diagrams, which show the object classes in the system and the associations between these classes.
- Object diagrams is a graph of instances, including objects and data values. A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. The use of object diagrams is fairly limited, namely to show examples of data structure.
- Component diagrams illustrate the pieces of software, embedded controllers, etc., that will make up a system. A component diagram has a higher level of abstraction than a Class Diagram - usually a component is implemented by one or more classes (or objects) at runtime. They are building blocks so a component can eventually encompass a large portion of a system.
- Deployment diagrams depicts a static view of the run-time configuration of processing nodes and the components that run on those nodes. In other words, deployment diagrams show the hardware for your system, the software that is installed on that hardware, and the middleware used to connect the disparate machines to one another.

Class Diagrams

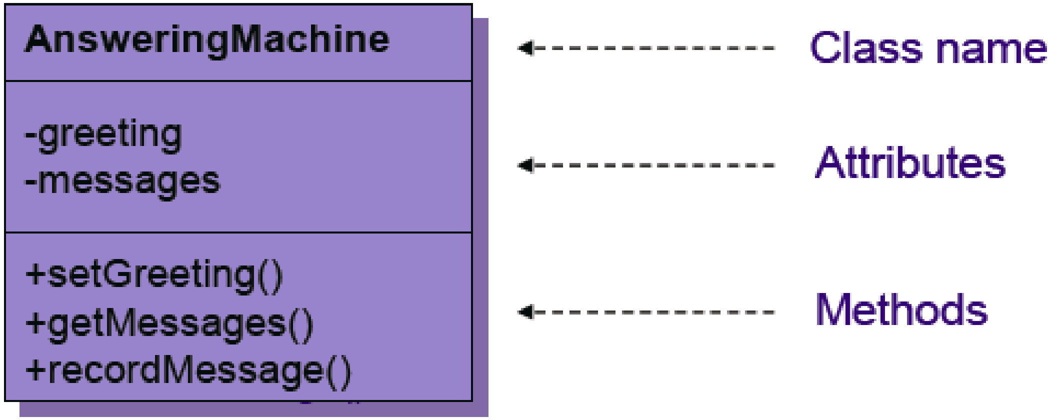
- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes
- An object class can be thought of as a general definition of one kind of system object
- An association is a link between classes that indicates that there is some relationship between these classes.
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

Class Diagram

What is a Class?

แสดงโครงสร้างหยุดนิ่งของระบบในลักษณะความสัมพันธ์ระหว่างคลาส

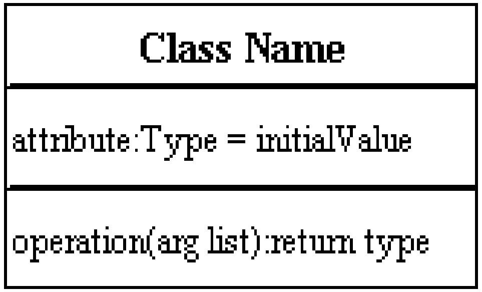
A class consists of a set of ***attributes*** and ***methods***. A ***class diagram*** is used to show the static structure of a class.



AnsweringMachine

-greeting
-messages

+setGreeting()
+getMessages()
+recordMessage()

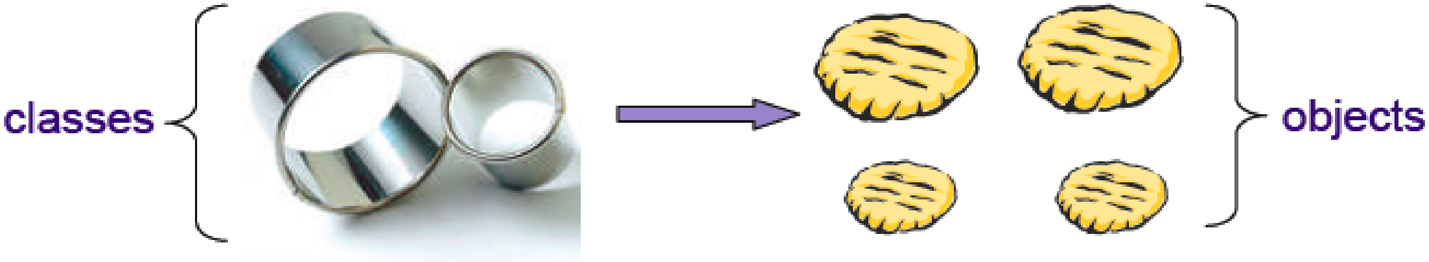


Class Name

attribute:Type = initialValue

operation(arg list):return type

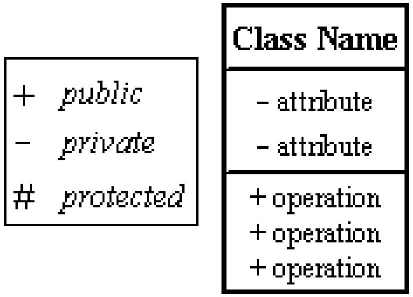
Each class is like a template that defines how ***instances*** of the class – objects – should be created.



Class Diagram

Accessibility

Attributes and methods can be declared at three levels of visibility.



class diagram notation

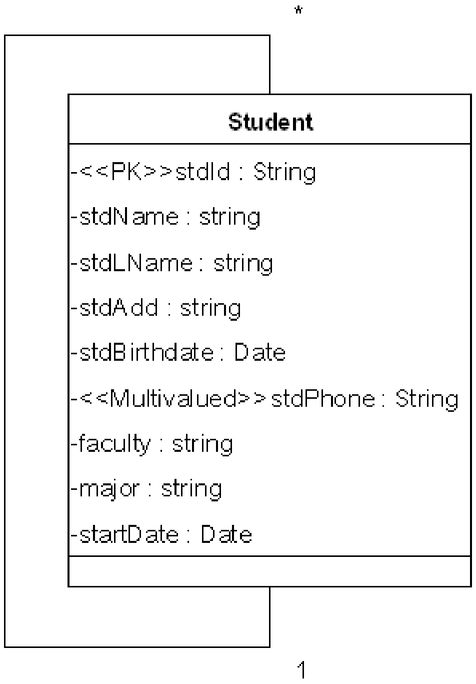
- Public (+)
  - □ Visible everywhere
- Private (-)
  - □ Visible only from within the declaring class
- Protected (#)
  - □ Visible only from within the declaring class and any of its ***subclasses***

We will all now swear **never** to declare public attributes.

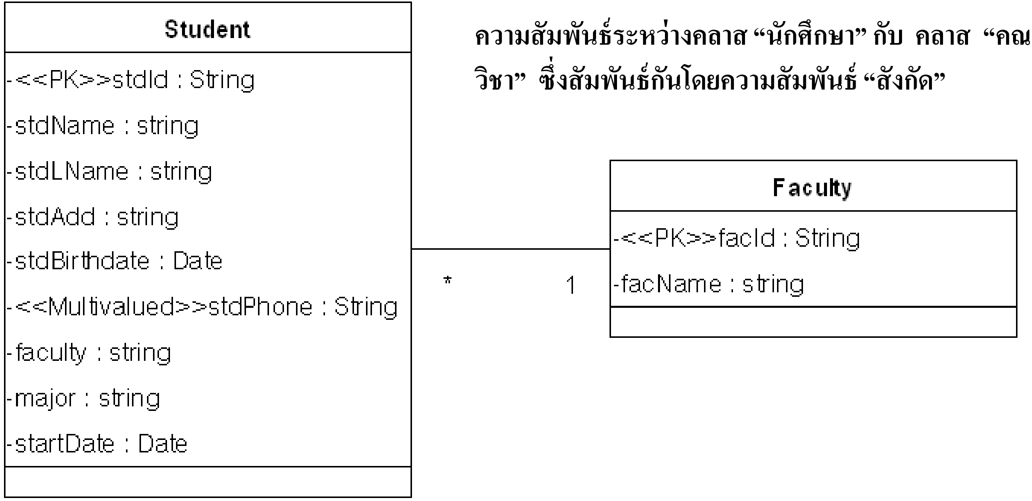
Visibility
Use visibility markers to signify who can access the information contained within a class. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class.

Class Diagram - Relationship

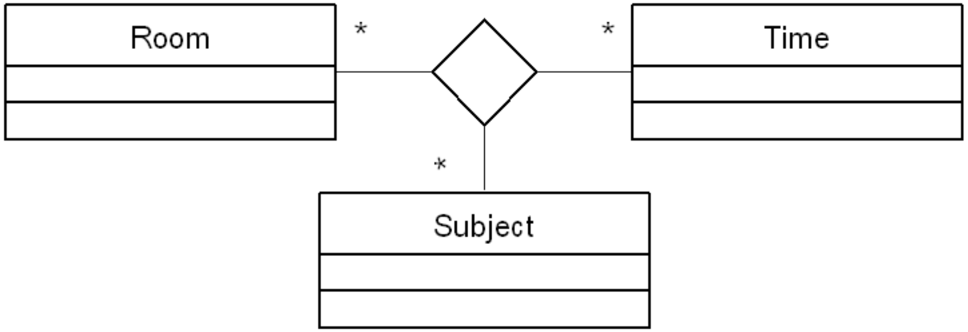
1) Unary Relationship เป็นความสัมพันธ์ที่เกิดกับคลาสเดียว เช่น หัวหน้าห้องของนักศึกษาแต่ละคน



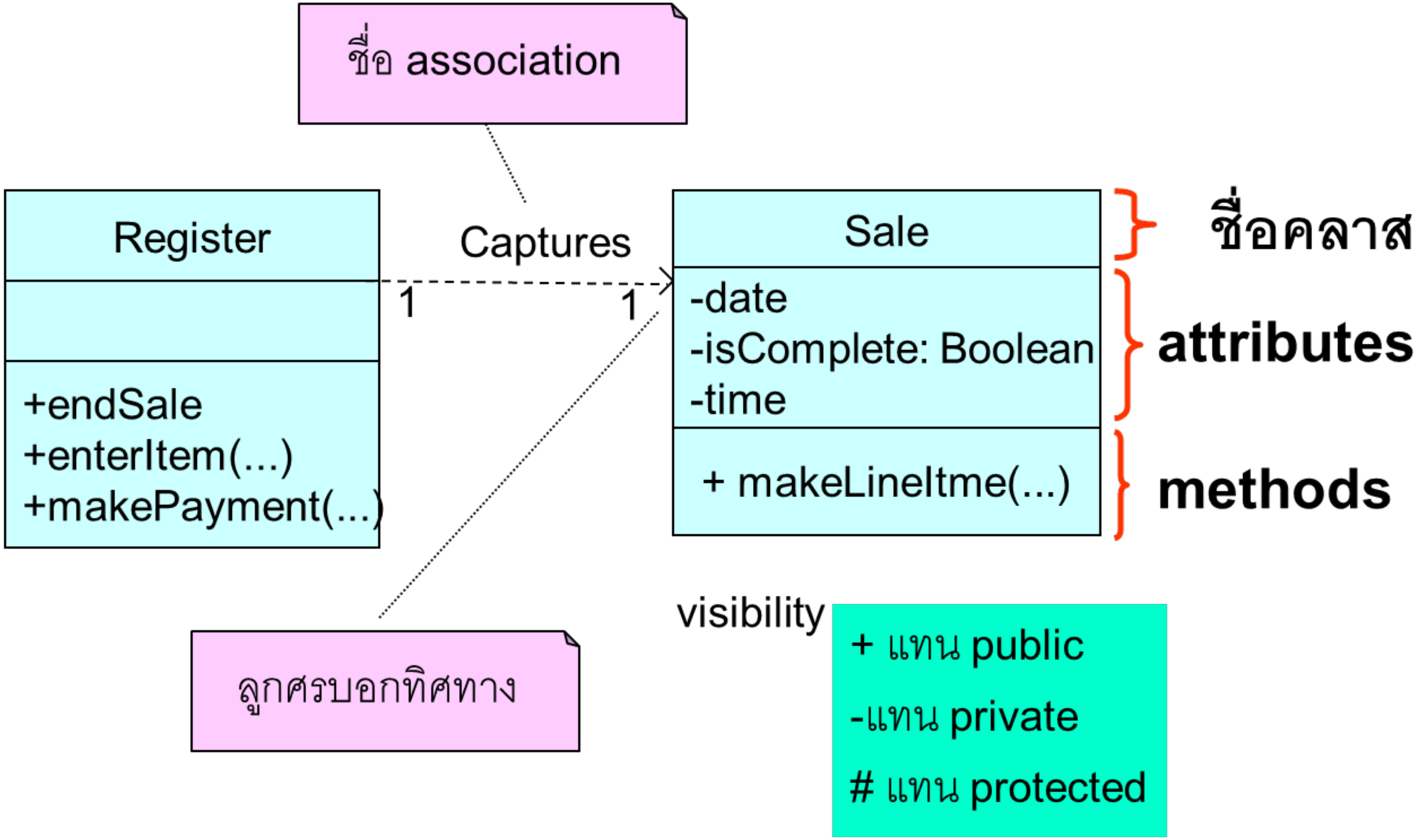
2) Binary Relationship เป็นความสัมพันธ์ที่เกิดขึ้นระหว่างคลาส 2 คลาส เช่น ความสัมพันธ์ระหว่างคลาส “นักศึกษา” กับ คลาส “คณะวิชา” ซึ่งสัมพันธ์กันโดยความสัมพันธ์ “สังกัด”



3) Ternary Relationship เป็นความสัมพันธ์ที่เกิดขึ้นระหว่างคลาสมากกว่า 2 คลาสขึ้นไป



Class Diagram



ชื่อ association

Register

+endSale
+enterItem(...)
+makePayment(...)

Captures

1

1

Sale

-date
-isComplete: Boolean
-time

+ makeLineItme(...)

ชื่อคลาส

attributes

methods

ลูกศรบอกทิศทาง

visibility

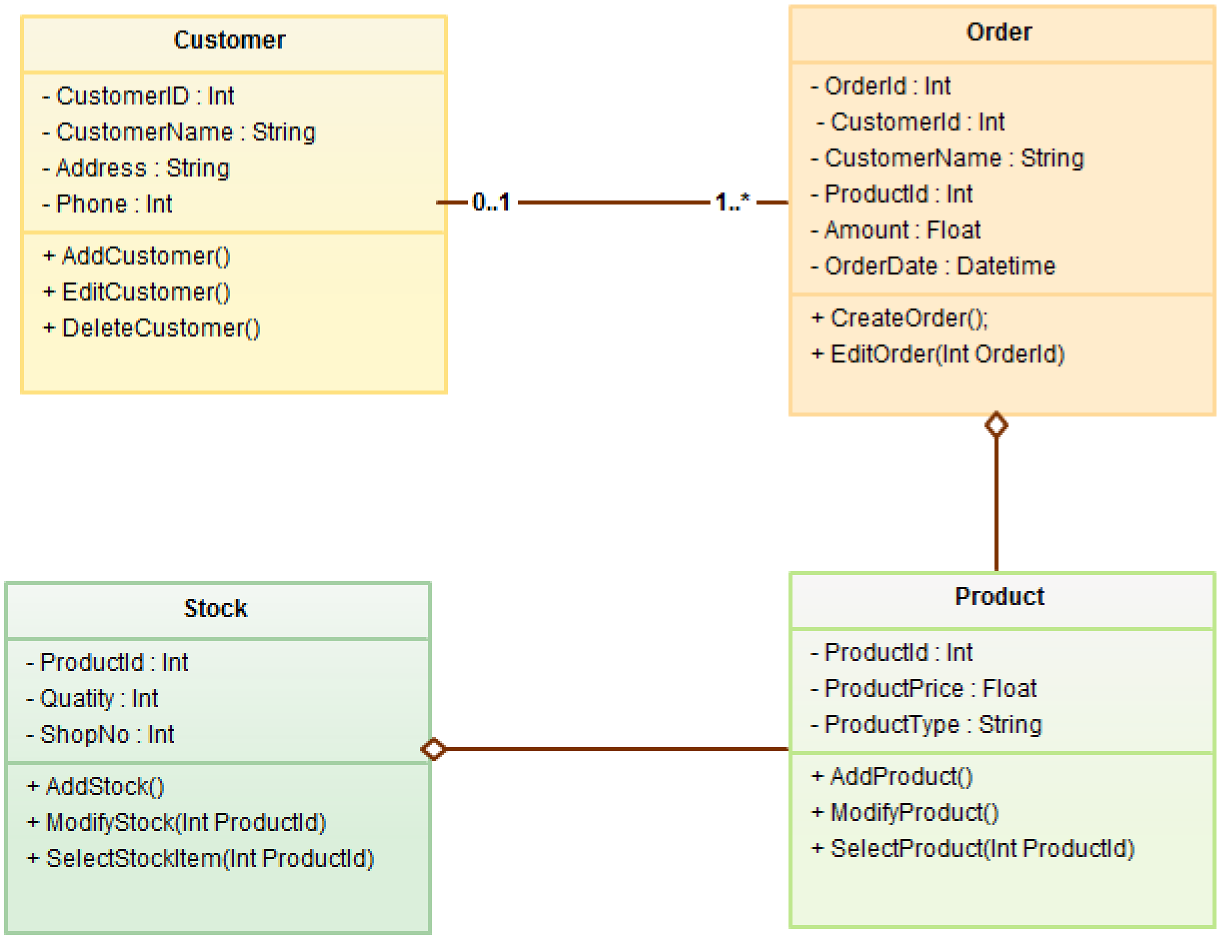
+ แทน public

-แทน private

แทน protected

Class Diagram

Class Diagram for Order Processing System



Customer

- CustomerID : Int
- CustomerName : String
- Address : String
- Phone : Int

+ AddCustomer()
+ EditCustomer()
+ DeleteCustomer()

Order

- OrderId : Int
- CustomerId : Int
- CustomerName : String
- ProductId : Int
- Amount : Float
- OrderDate : Datetime

+ CreateOrder();
+ EditOrder(Int OrderId)

Stock

- ProductId : Int
- Quatity : Int
- ShopNo : Int

+ AddStock()
+ ModifyStock(Int ProductId)
+ SelectStockItem(Int ProductId)

Product

- ProductId : Int
- ProductPrice : Float
- ProductType : String

+ AddProduct()
+ ModifyProduct()
+ SelectProduct(Int ProductId)

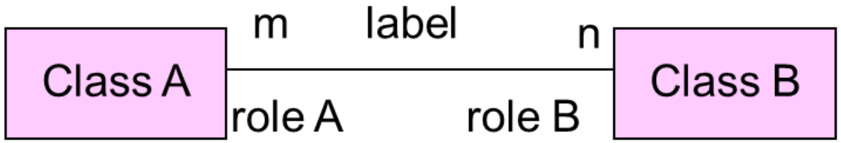
http://creately.com/blog/diagrams/uml-diagram-types-examples/

Class Diagram –
Relations between Classes

- ❑ Association
- ❑ Aggregation
- ❑ Composition
- ❑ Inheritance/Generalization

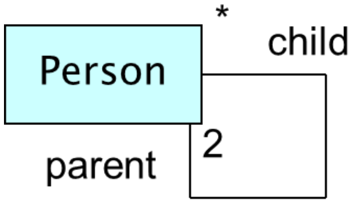
Class Diagram - Association

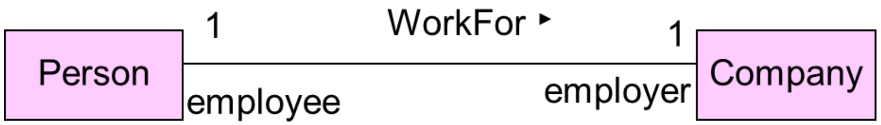
- เป็นความสัมพันธ์ระหว่างคลาส A ไปยังคลาส B และจากคลาส B ไปยังคลาส A

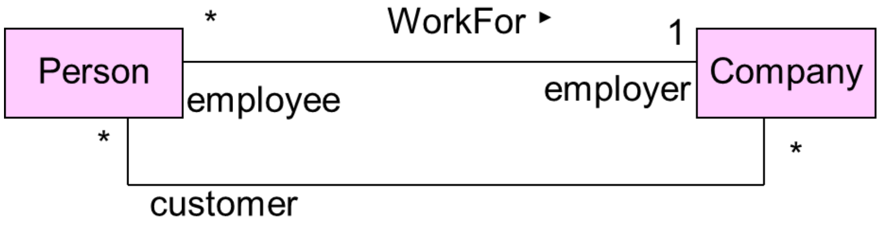


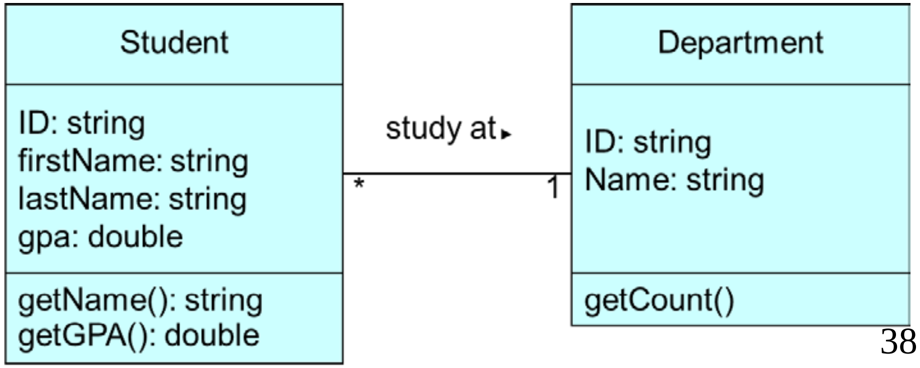
m และ **n** เป็น **multiplicity number**

Association แทนด้วยเส้นเชื่อมระหว่างสองคลาส
label เป็นคำอธิบายเส้นการเชื่อมโยงระหว่างคลาส (เป็นชื่อ association ระหว่างสองคลาส)



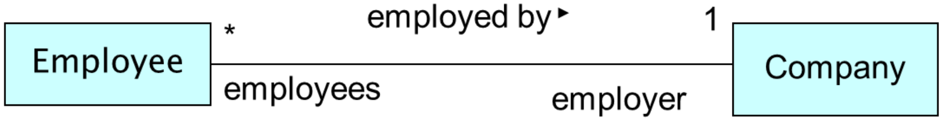






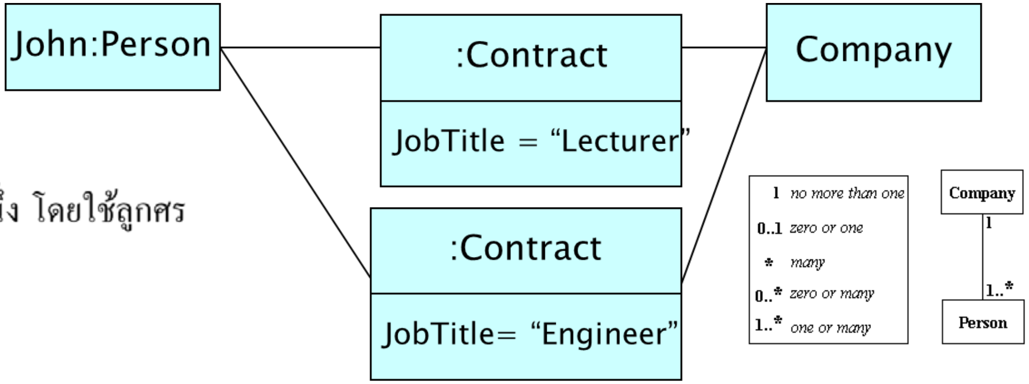
Class Diagram - Multiplicity

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)



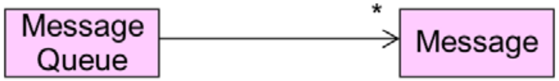
```
public class Employee {
        public Employee() { }

   private string name;
   private Company  employer;
}
```

```
public class Company {
        public Company(){ };
        public Company(stirng s){
name= s;}
   private string name;
   private Employee []  empoyeers;
}
```



(John ทำสัญญาทำงาน มาแล้ว 2 สัญญา สัญญาแรกทำในตำแหน่ง "Lecturer" สัญญาที่สองทำในตำแหน่ง "Engineer" แต่ละสัญญาอาจทำกับบริษัทเดิม หรือบริษัทใหม่ก็ได้

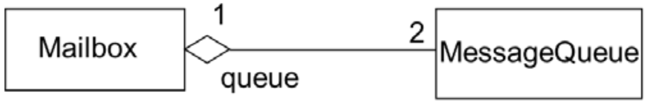
- Directed Association
  - ต้องการระบุความสัมพันธ์ไปทางทิศทางใดทิศทางหนึ่ง โดยใช้ลูกศร



Queue หนึ่งตัว ประกอบด้วย หลาย Message
Queue ต้องการรู้จัก Message จึงจะทำงานได้
แต่ Message ไม่ต้องรู้จัก Queue ก็ทำงานได้

Class Diagram - Aggregation

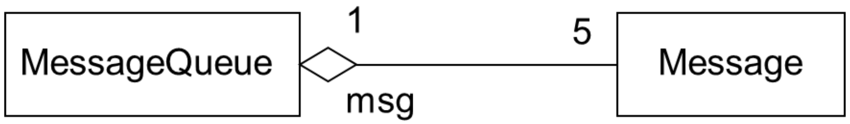
- Aggregation (“has-a” relationship)
  - □ Object ของคลาส A บรรจุ Object ของคลาส B ณ ช่วงเวลาใดเวลาหนึ่ง หรือ an Object of A has a object of B.
  - □ Aggregation ถือว่าเป็นความสัมพันธ์แบบหนึ่งของ dependency



```
public class Mailbox {
 ...
   private MessageQueue[2] queue;
}
```

```
public class MessageQueue {
 ...
   private Message [5 ] msg;
}
```

(Horstmann,2004, p.48)



```
public class MessageQueue {
   ...
   private  Message [5]  msg;

}
```

```
public class Message{
   ...
   private String text;
}
```

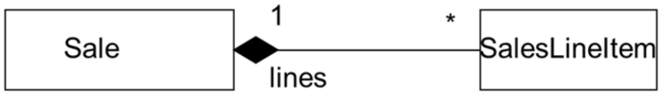
MessageQueue บรรจุ อ็อบเจกต์ของคลาส Message ได้ไม่เกิน 5 ตัว

- An aggregation model shows how classes that are collections are composed of other classes
- Aggregation models are similar to the part-of relationship in semantic data models

Class Diagram - Composition

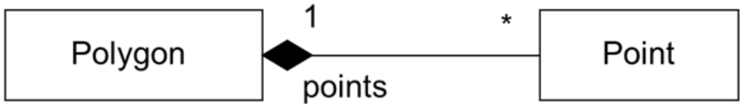
Composition is a strong form of association in which the 'part' objects are dependent on the 'whole' objects.

1. a 'part' object can only belong to one composite at a time,
2. when a composite object is destroyed, all its dependent part must be destroyed at the same time .



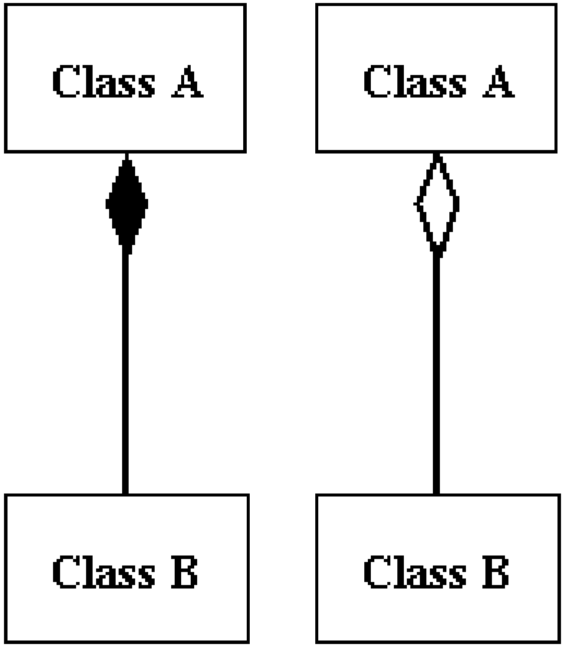
```
public class Sale {
   ...
   private  SalesLineItem [ ]  lines;
}
```

```
public class SalesLineItem{
   ...

}
```

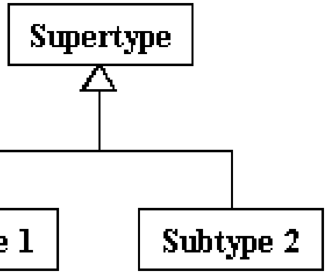


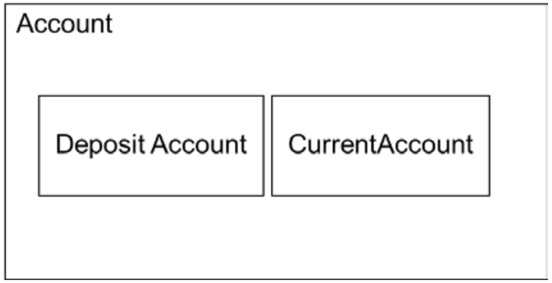
```
public class Polygon {
   ...
   private  Point [ ]  points;
}
```

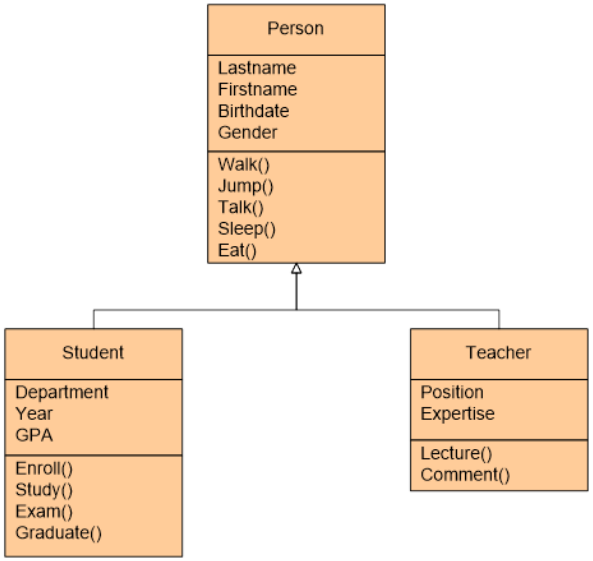
```
public class  Point{

 ...
  private double x;
  private double y;

}
```

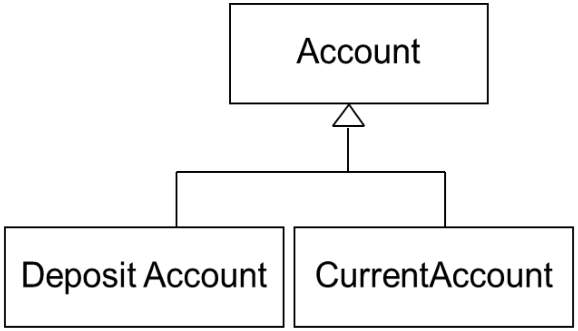


Class Diagram – Inheritance/Generalization









```
public class Account {
            ...
            private String id;
            private double balance;
}
```

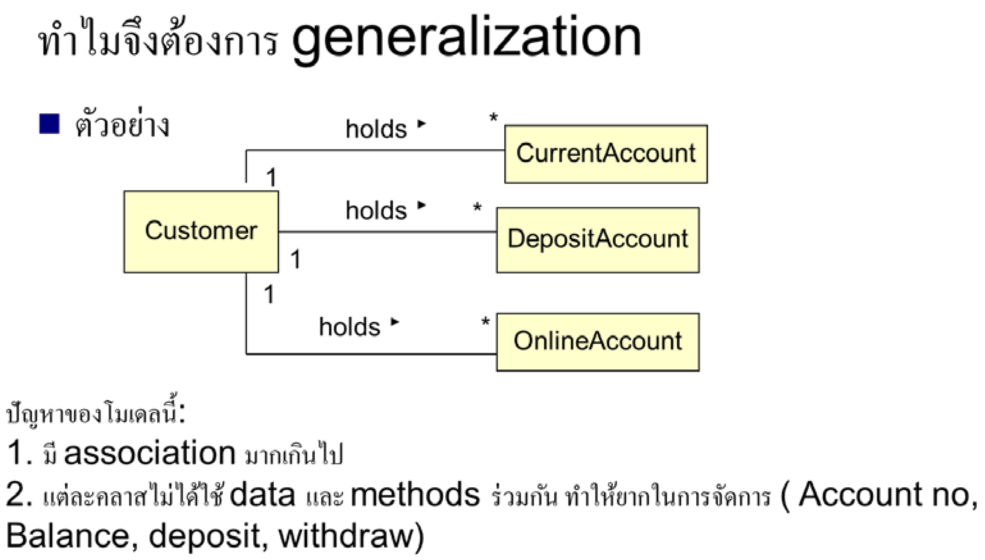
```
public class DepositAccount extends Account{
            ...
}
```

Generalization is an everyday technique that we use to manage complexity

Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes

This allows us to infer that different members of these classes have some common characteristics, e.g. squirrels and rats are rodents

Class Diagram - Generalization



ทำไมจึงต้องการ generalization

- ตัวอย่าง

ปัญหาของโมเดลนี้:

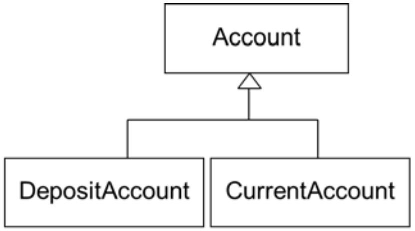
1. มี association มากเกินไป

2. แต่ละคลาสไม่ได้ใช้ data และ methods ร่วมกัน ทำให้ยากในการจัดการ (Account no, Balance, deposit, withdraw)

- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language
- In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes
- The lower-level classes are sub-classes that inherit the attributes and operations from their super-classes. These lower-level classes then add more specific attributes and operations.

Class Diagram – Generalization and Specialization

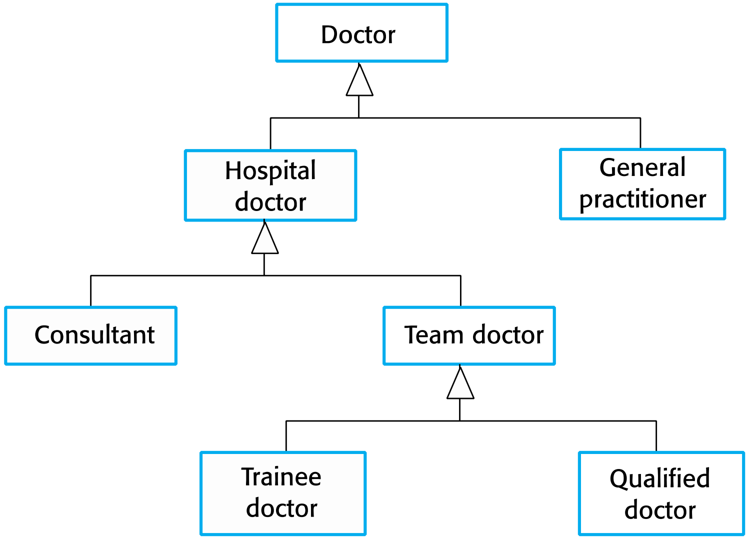
- Superclass
- Subclasses

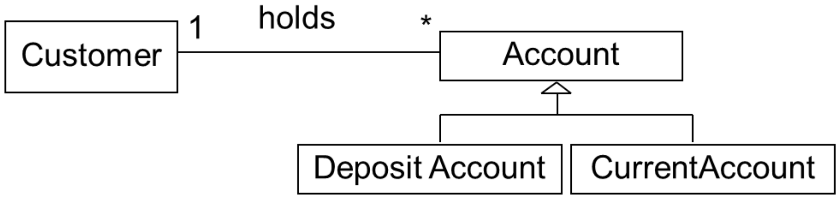


Superclass คือ คลาสที่เป็นบรรพบุรุษของคลาสอื่น ใช้แสดงถึงความเป็นสากลของคลาสอื่น ในรูปได้แก่ Account

Subclass เป็นคลาสที่สืบทอดจาก Superclass ใช้แสดงถึงกรณีเฉพาะของ Superclass ในรูปได้แก่ DepositAccount และ CurrentAccount

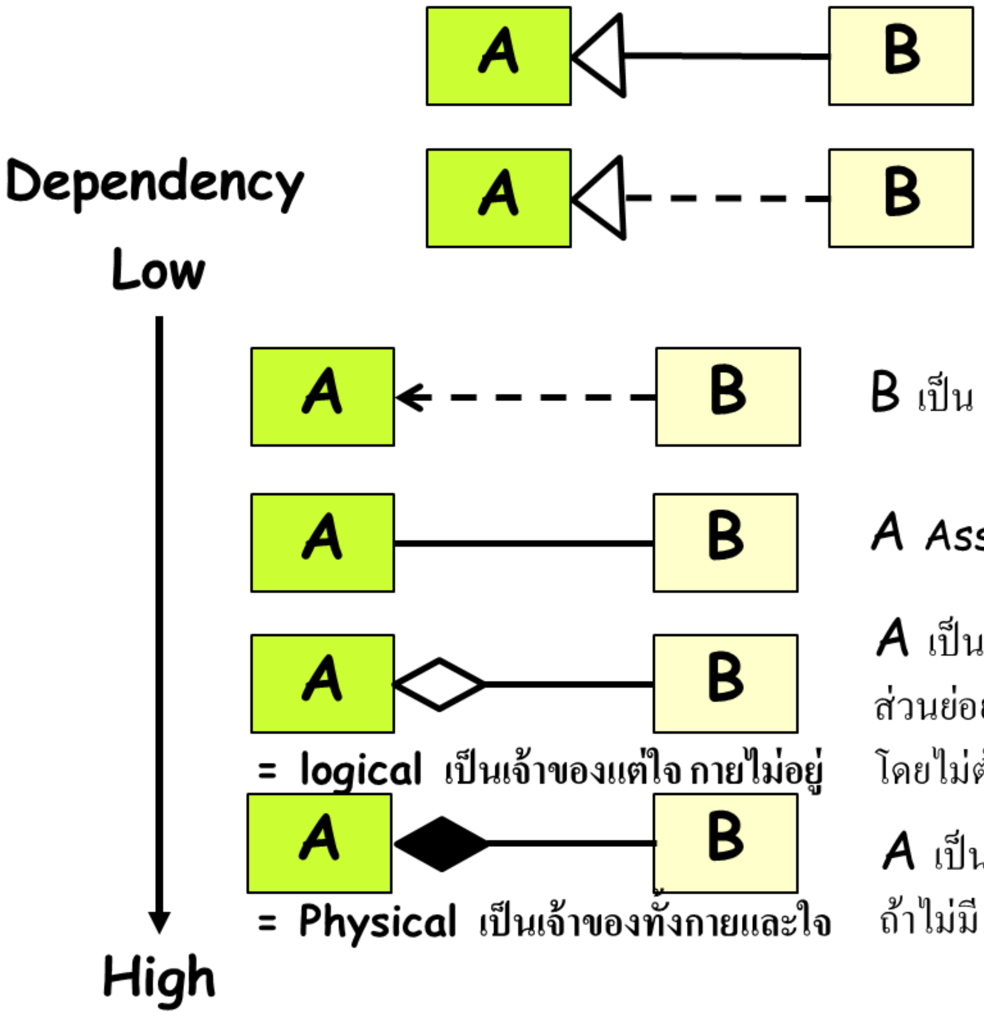
The deposit account is a subclass of Account.
The current account is a subclass of Account.





ลูกค้า 1 คนถือบัญชีได้หลายบัญชี บัญชีเหล่านั้นอาจเป็น บัญชี Deposit หรือ Current ก็ได้

Class Diagram



- A inherit เป็น B
- B Generalize A

- A เป็น Implement B
- B ถูก Implement A

B เป็น on A (ขึ้นต่อกันและกันน้อยที่สุด)

A Associate B หรือ B Associate A

A เป็น Aggregate B (A เป็นส่วนรวม B เป็นส่วนย่อย ถ้า A ไม่อยู่ B ก็สามารถอยู่ได้ หรือ A อยู่ได้โดยไม่ต้องมี B)

= logical เป็นเจ้าของแต่ใจ กายไม่อยู่

A เป็น Composition ของ B (B จะอยู่ไม่ได้ถ้าไม่มี A)

= Physical เป็นเจ้าของทั้งกายและใจ

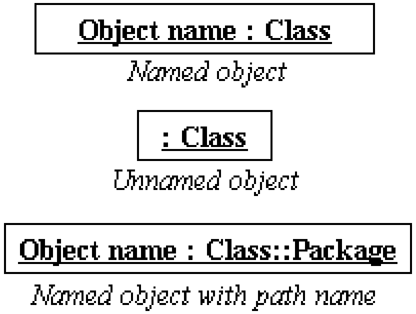
Object Diagram

Object diagrams are also closely linked to class diagrams. Just as an object is an instance of a class, an object diagram could be viewed as an instance of a class diagram. Object diagrams describe the static structure of a system at a particular time and they are used to test the accuracy of class diagrams.

แสดงโครงสร้างหยุดนิ่งของระบบในลักษณะความสัมพันธ์ระหว่างออบเจ็กต์

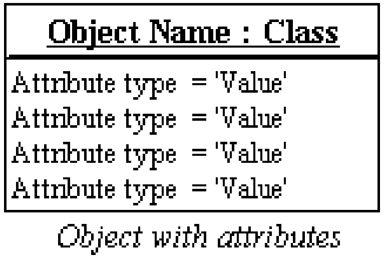
Object names

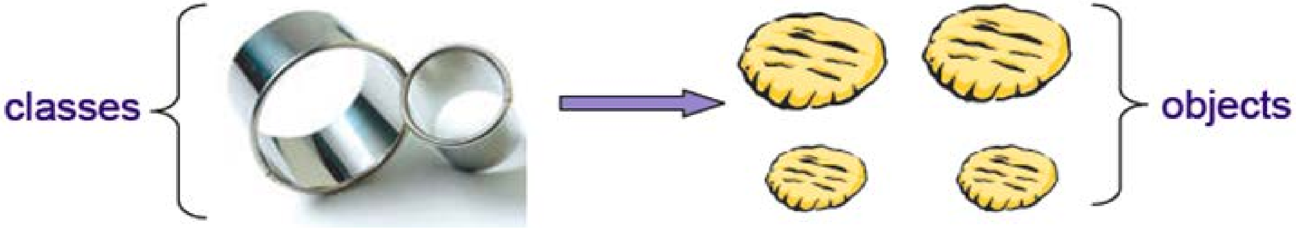
Each object is represented as a rectangle, which contains the name of the object and its class underlined and separated by a colon.



Object attributes

As with classes, you can list object attributes in a separate compartment. However, unlike classes, object attributes must have values assigned to them.

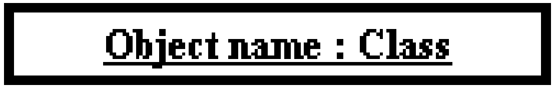




Object Diagram

Active object

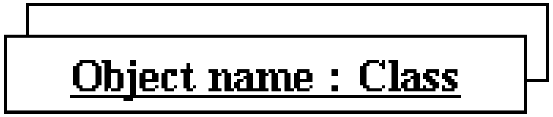
Objects that control action flow are called active objects. Illustrate these objects with a thicker border.



Active object

Multiplicity

You can illustrate multiple objects as one symbol if the attributes of the individual objects are not important.

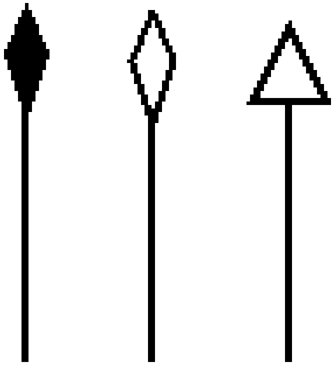


Multiple objects

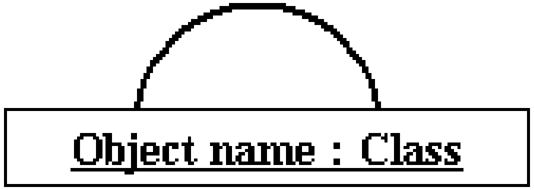
Links

Links are instances of associations. You can draw a link using the lines used in class diagrams.

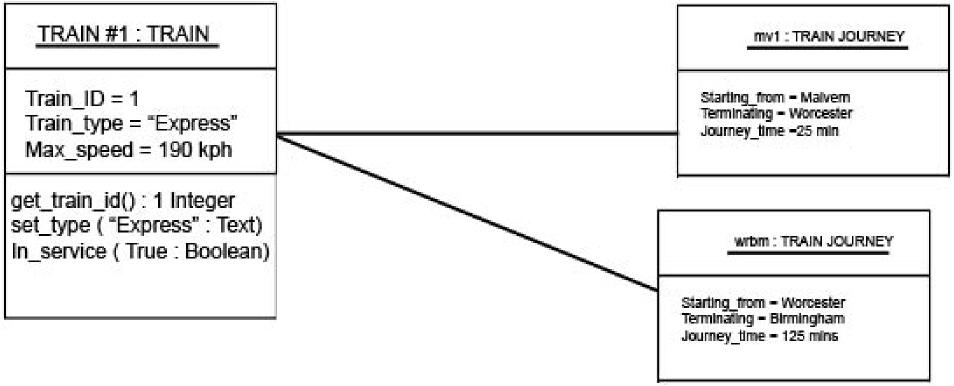
Self-linked
Objects that fulfill more than one role can be self-linked. For example, if Mark, an administrative assistant, also fulfilled the role of a marketing assistant, and the two positions are linked, Mark's instance of the two classes will be self-linked.



Links



Object Diagram



TRAIN #1 : TRAIN

Train_ID = 1
Train_type = "Express"
Max_speed = 190 kph

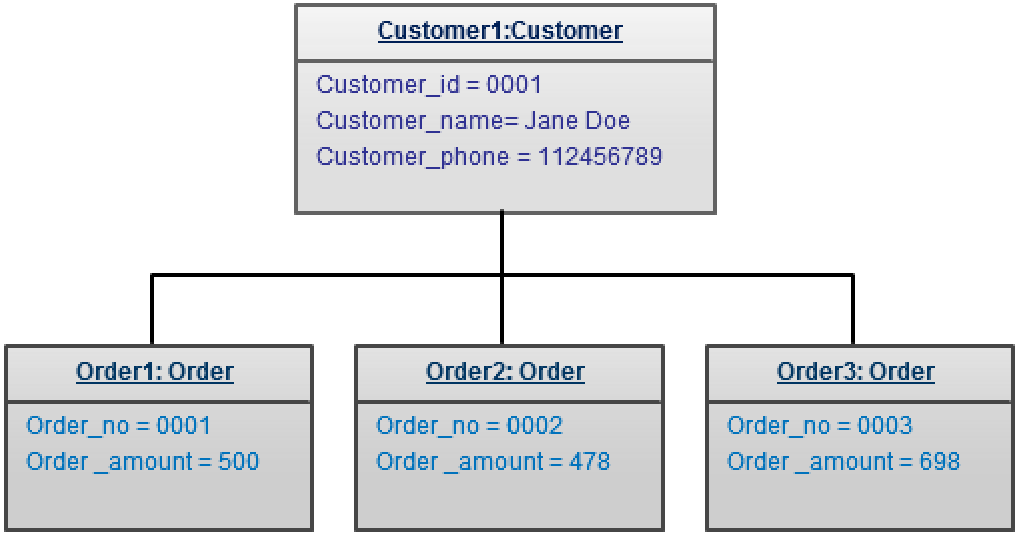
get_train_id() : 1 Integer
set_type ("Express" : Text)
In_service (True : Boolean)

mv1 : TRAIN JOURNEY

Starting_from = Malvern
Terminating = Worcester
Journey_time =25 min

wrbm : TRAIN JOURNEY

Starting_from = Worcester
Terminating = Birmingham
Journey_time = 125 mins



Customer1:Customer

Customer_id = 0001
Customer_name= Jane Doe
Customer_phone = 112456789

Order1: Order

Order_no = 0001
Order _amount = 500

Order2: Order

Order_no = 0002
Order _amount = 478

Order3: Order

Order_no = 0003
Order _amount = 698

[online diagramming & design] creately.com

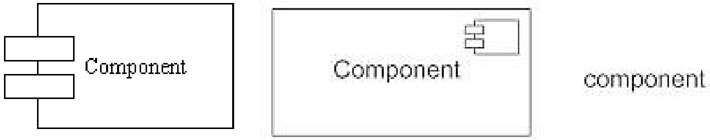
Component Diagram

A component diagram could represent
- **Logical Components** (e.g., business components, process components), and
- **Physical Components** (e.g., CORBA components, EJB components, COM+ and .NET components, WSDL components, etc.),

แสดงองค์ประกอบหรือไฟล์จริงของระบบ (เช่นไฟล์ exe และ dll)ที่ออกแบบและสถานที่เก็บ

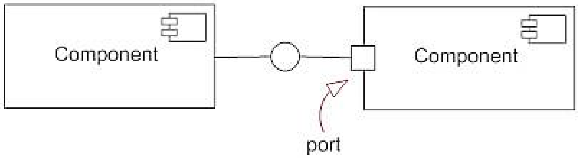
Component

A component is a physical building block of the system. It is represented as a rectangle with tabs.



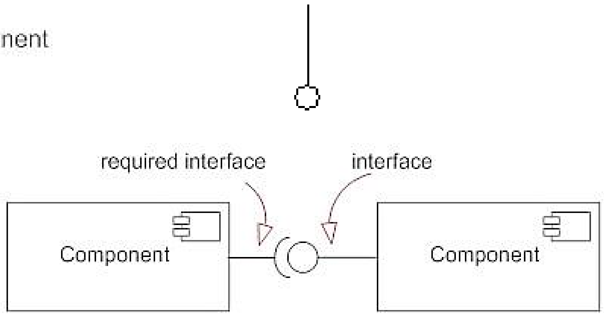
Port

Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.



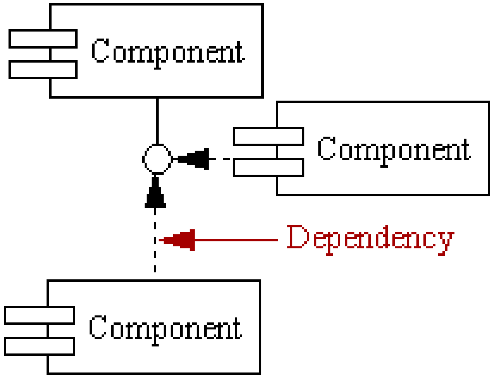
Interface

An interface describes a group of operations used or created by components.

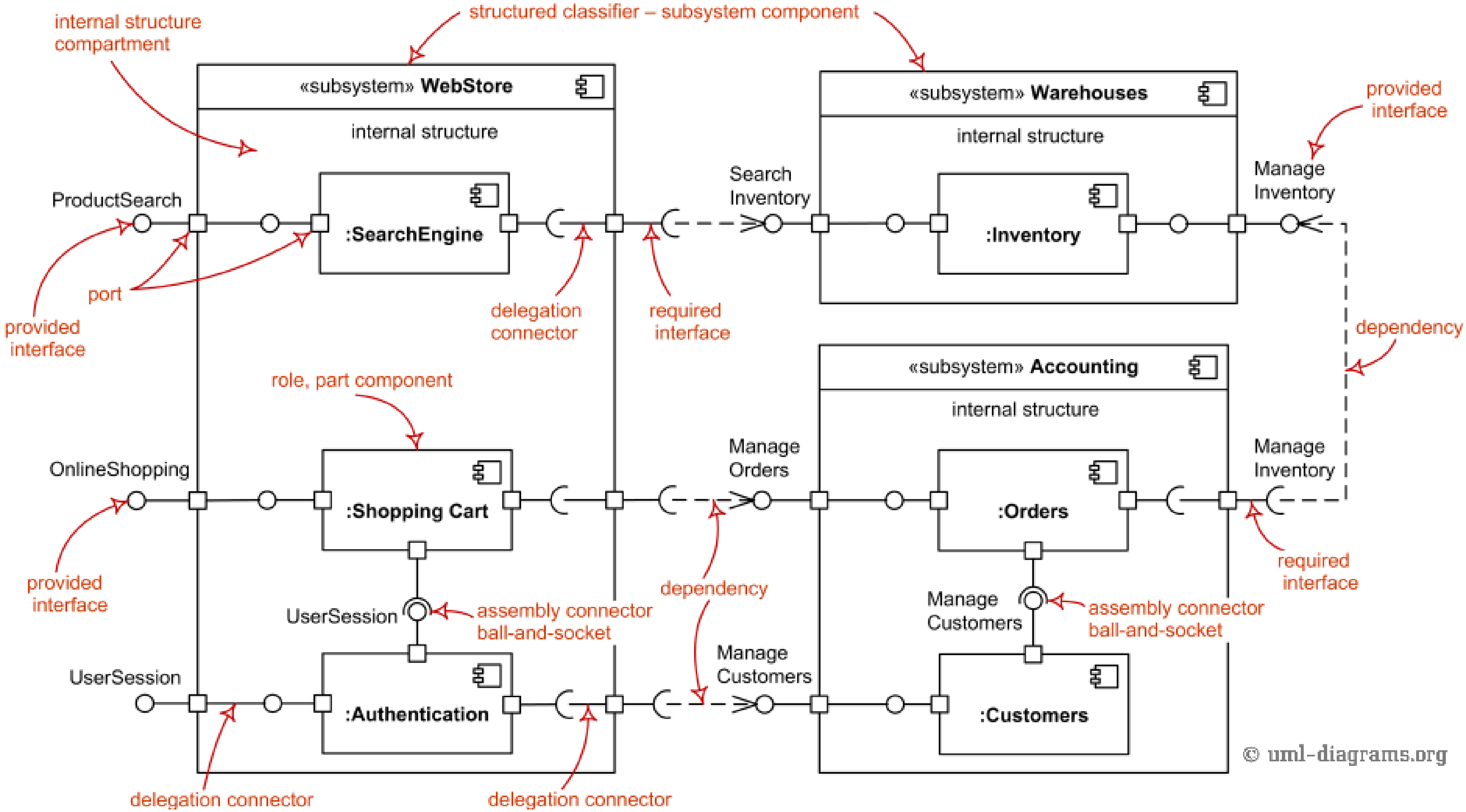


Dependencies

Draw dependencies among components using dashed arrows.



Component Diagram



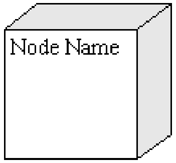
Deployment Diagram

Deployment diagrams depict the physical resources in a system including nodes, components, and connections. (Hardware and Software)

แสดงการนำโปรแกรมที่พัฒนาไปติดตั้งในฮาร์ดแวร์ในระบบ

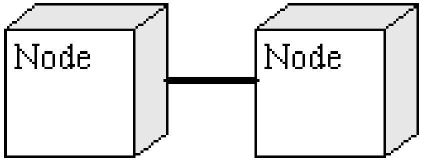
Component

A node is a physical resource that executes code components.



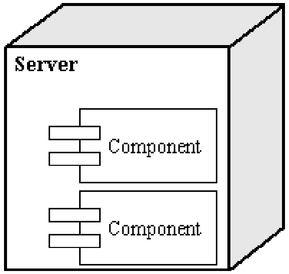
Association

Association refers to a physical connection between nodes, such as Ethernet.

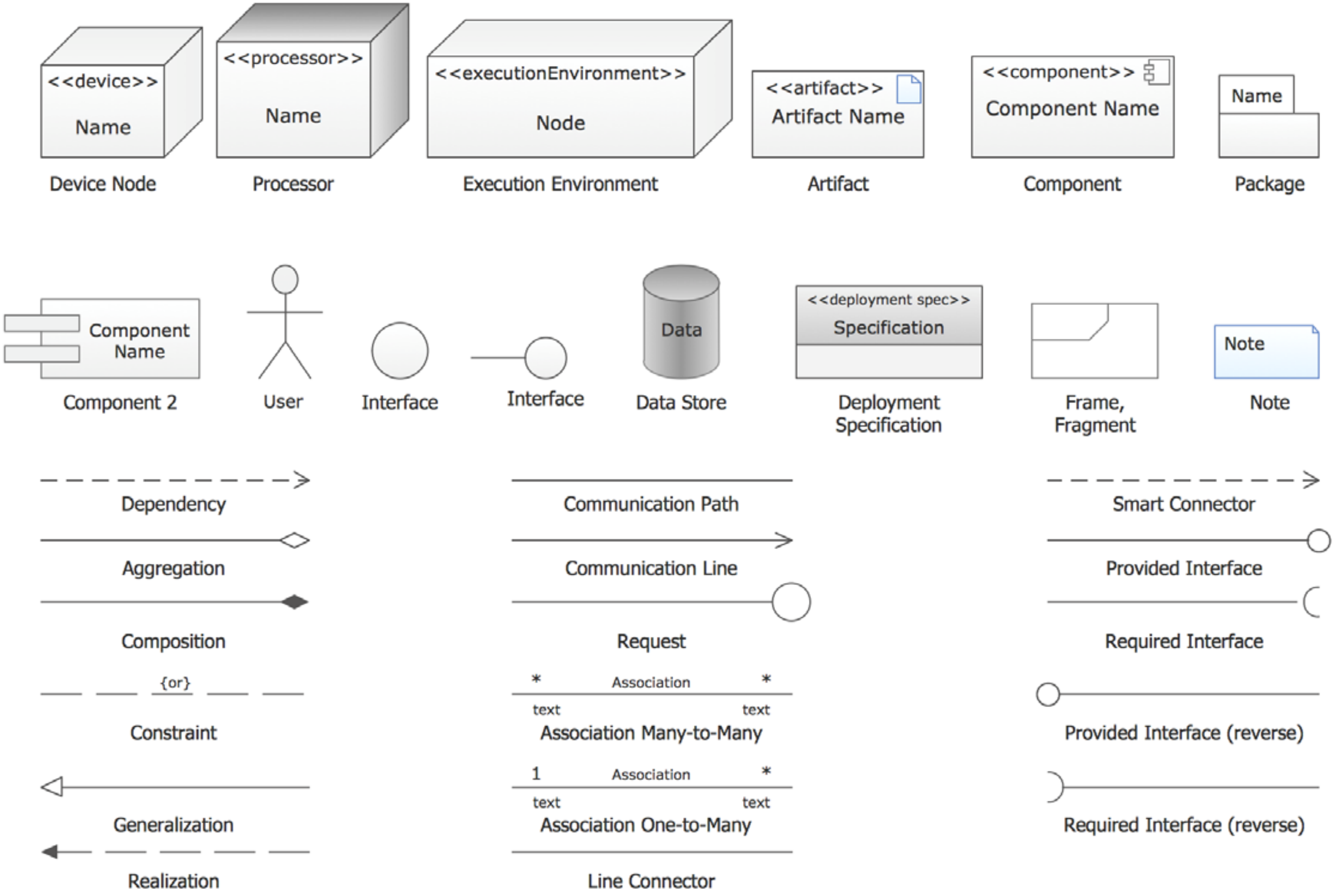


Components and Nodes

Place components inside the node that deploys them.

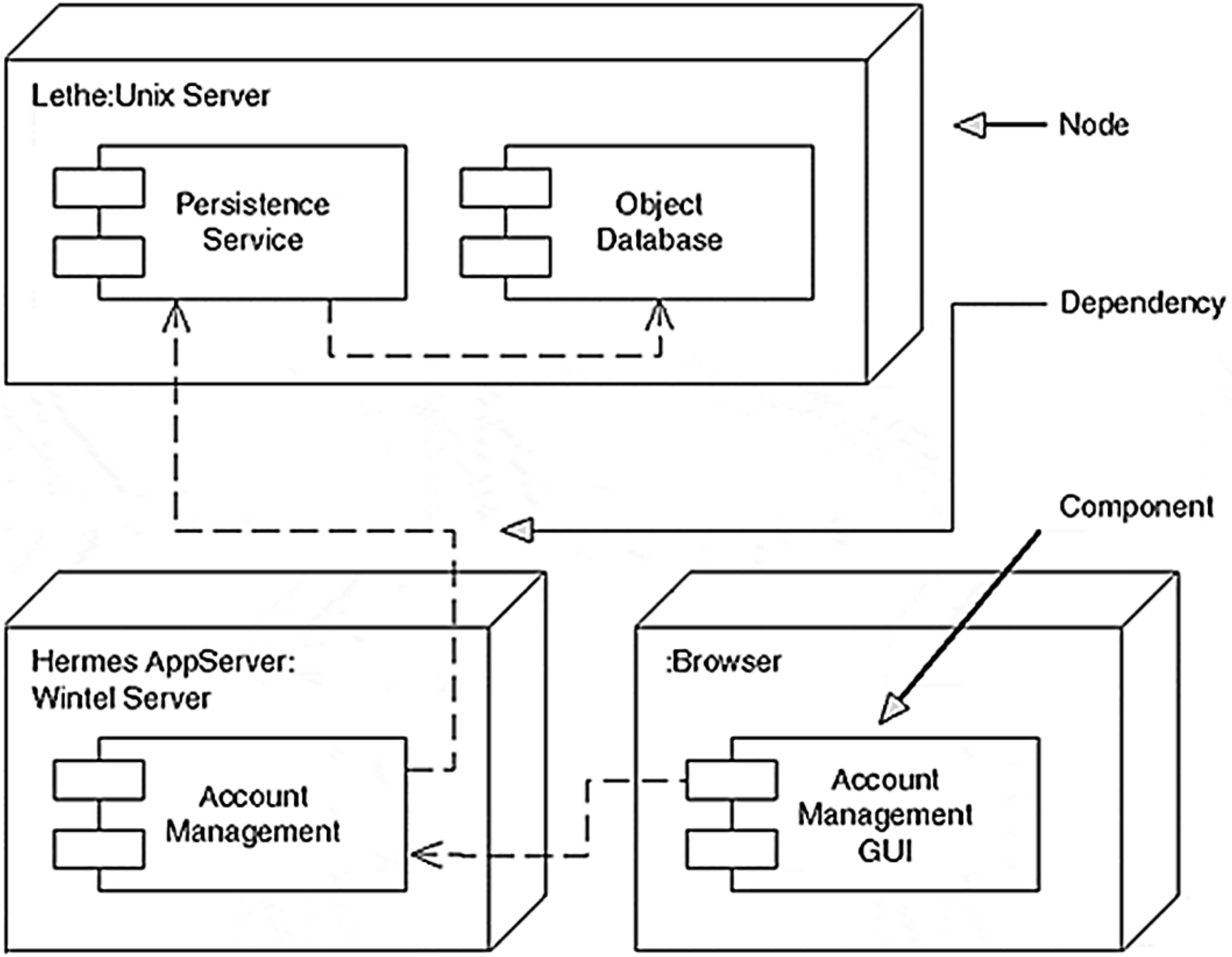


Deployment Diagram

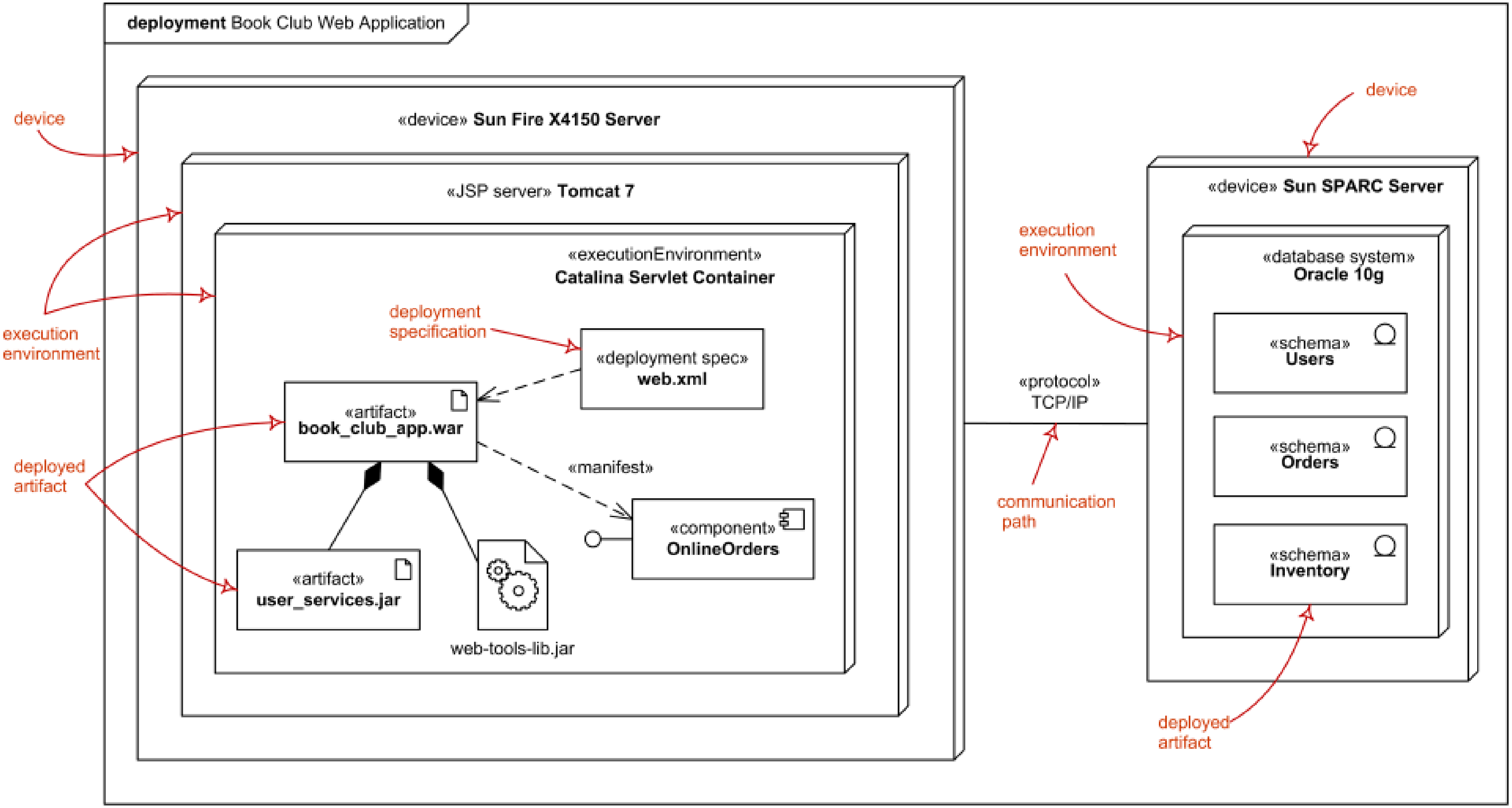


http://www.conceptdraw.com/How-To-Guide/picture/Design-elements-UML-deployment-diagrams.png

Deployment Diagram



Deployment Diagram



http://www.uml-diagrams.org/deployment-diagrams-overview.html

UML-Behavioral Diagram

- Use case diagrams, which show the interactions between a system and its environment
- Sequence diagrams, which show interactions between actors and the system and between system components
- Activity diagrams, which show the activities involved in a process or in data processing
- Collaboration Diagram or Communication Diagrams like UML sequence diagrams, are used to explore the dynamic nature of your software. Collaboration diagrams show the message flow between objects in an OO application, and also imply the basic associations (relationships) between classes.
- State diagrams, which show how the system reacts to internal and external events

Use Case Diagrams

- Use Case Diagrams gives a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions are interacted.
- Use cases were developed originally to support requirements elicitation and now are incorporated into the UML
- Each use case represents a discrete task that involves external interaction with an actor
- Actors in a use case may be people or other systems

ปฏิสัมพันธ์ระหว่างผู้ใช้ภายนอกและฟังก์ชันการทำงานหลักภายในระบบ

- **Use cases** are represented as the horizontally shaped ovals and display the different uses.

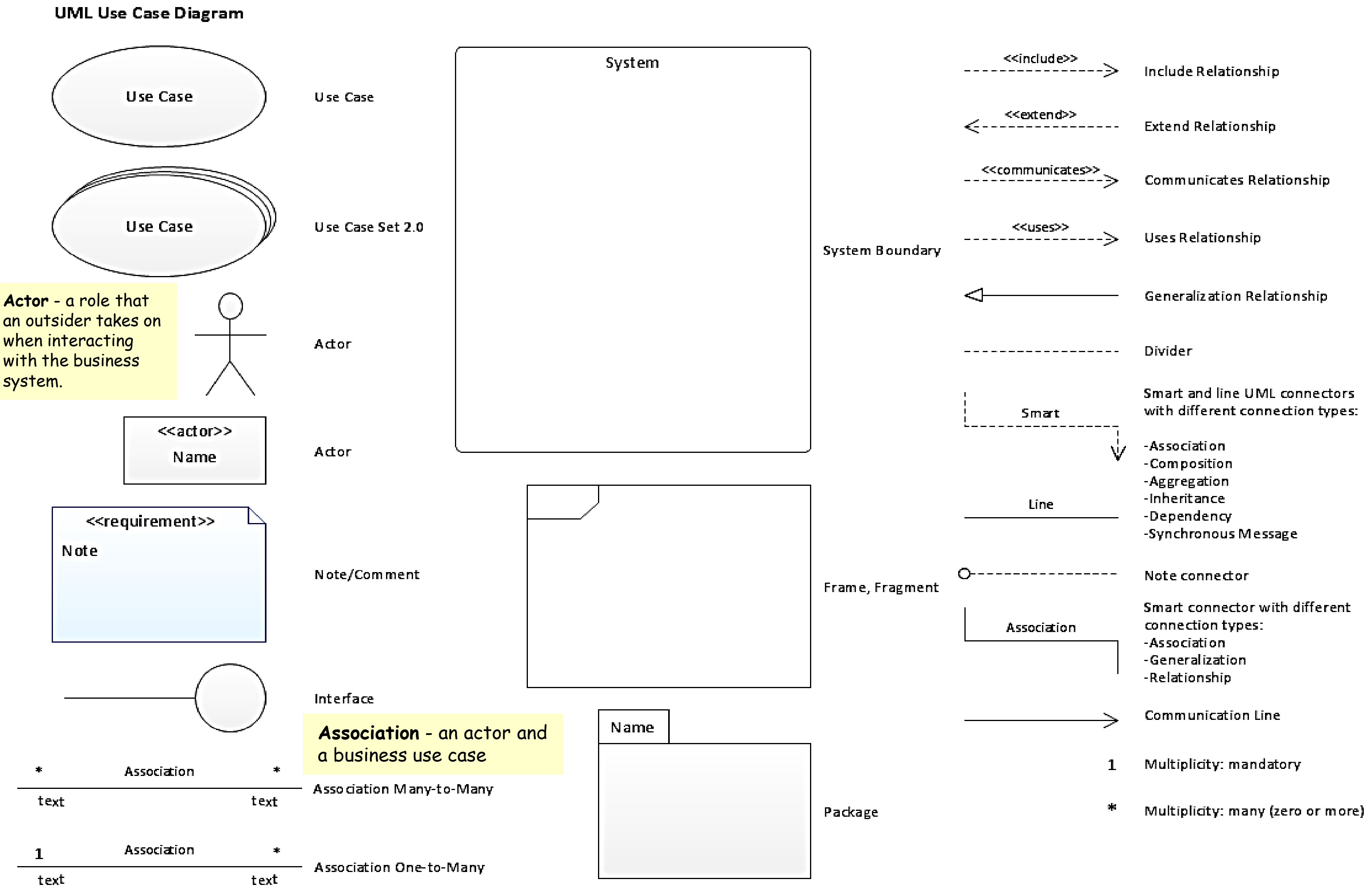
- **Actors** are the people that employ the use cases and are represented on the diagram as figures of persons. Actors cannot be related each to other (except relations of generalization/inheritance).

- **Associations** are shown as lines between actors and use cases.

- **System boundary** – the box with the name and ovals (use cases) inside that sets a system scope to use cases.

- **Packages** that allow you to add the elements in groups.

Use Case Diagrams



Actor - a role that an outsider takes on when interacting with the business system.

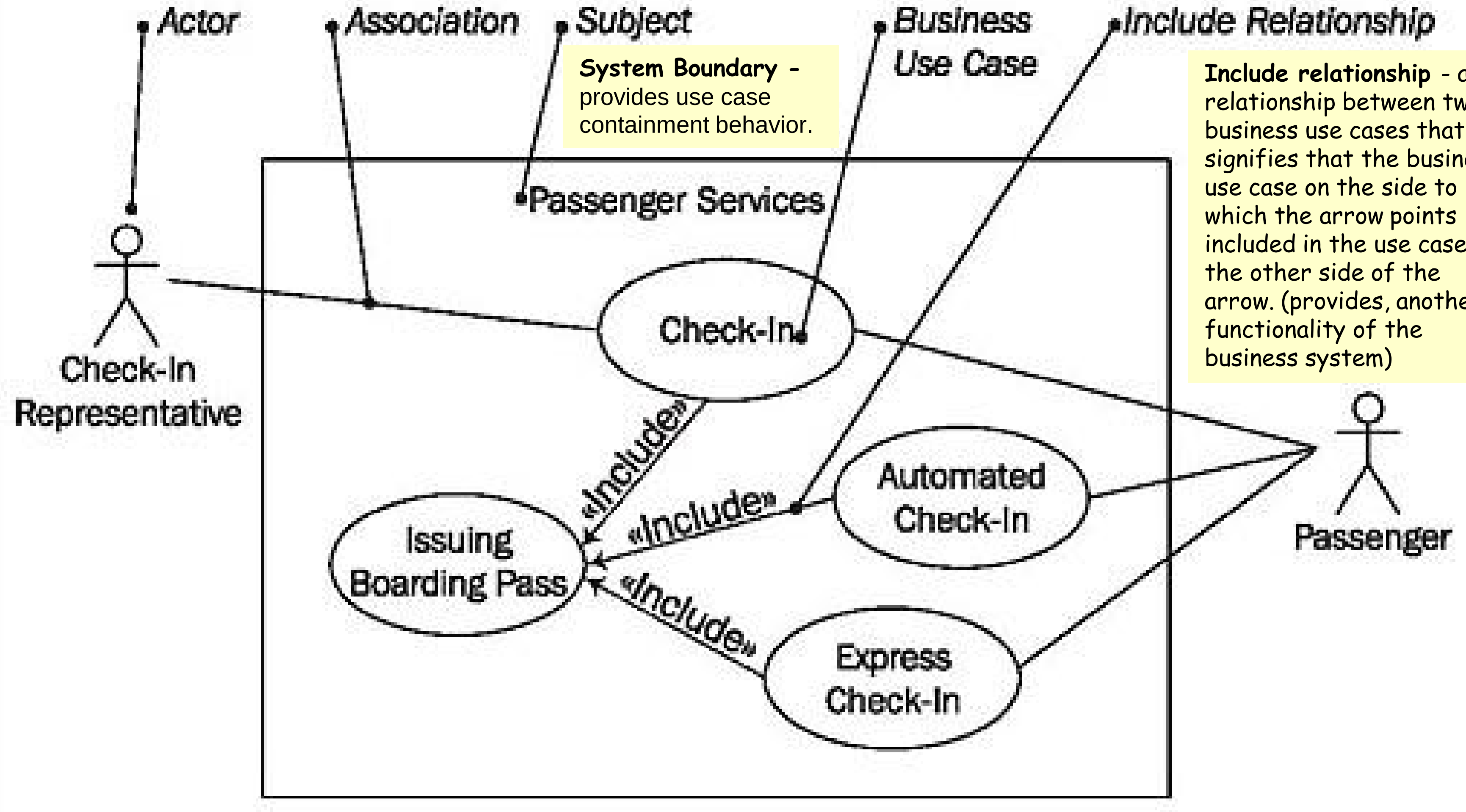
Association - an actor and a business use case

Use Case Diagrams

Actor - a role that an outsider takes on when interacting with the business system.

Association - an actor and a business use case

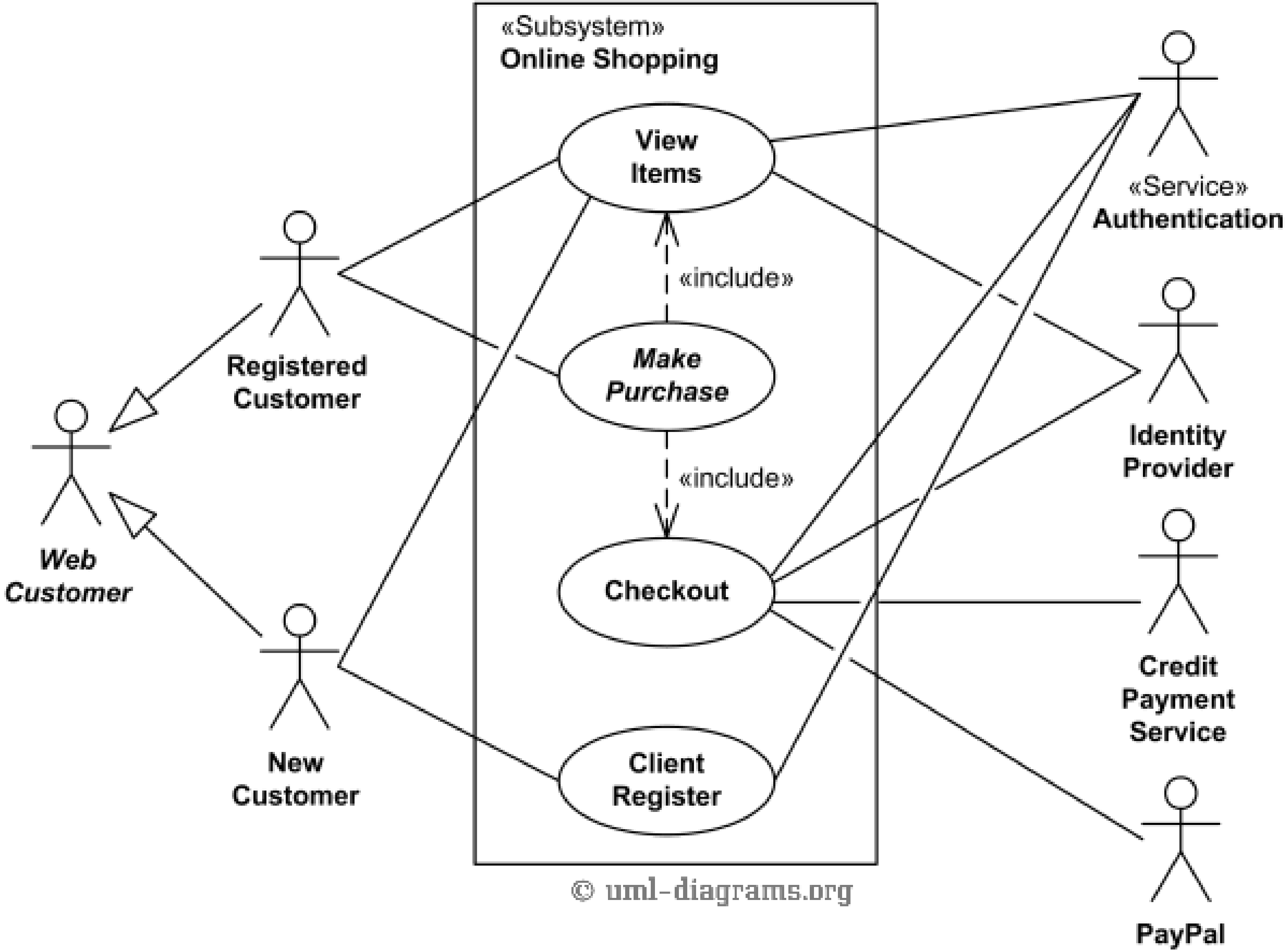
Use Case -the interaction between an actor and a business system, meaning it describes the functionality of the business system



System Boundary - provides use case containment behavior.

Include relationship - a relationship between two business use cases that signifies that the business use case on the side to which the arrow points is included in the use case on the other side of the arrow. (provides, another functionality of the business system)

Use Case Diagrams



© uml-diagrams.org

Sequence Diagrams

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these
- Interactions between objects are indicated by annotated arrows

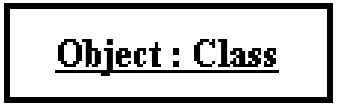
Sequence Diagrams

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

แสดงการปฏิสัมพันธ์ระหว่างออบเจ็กต์สำหรับแต่ละ **Use Case** โดยมีลำดับของเวลาด้วย

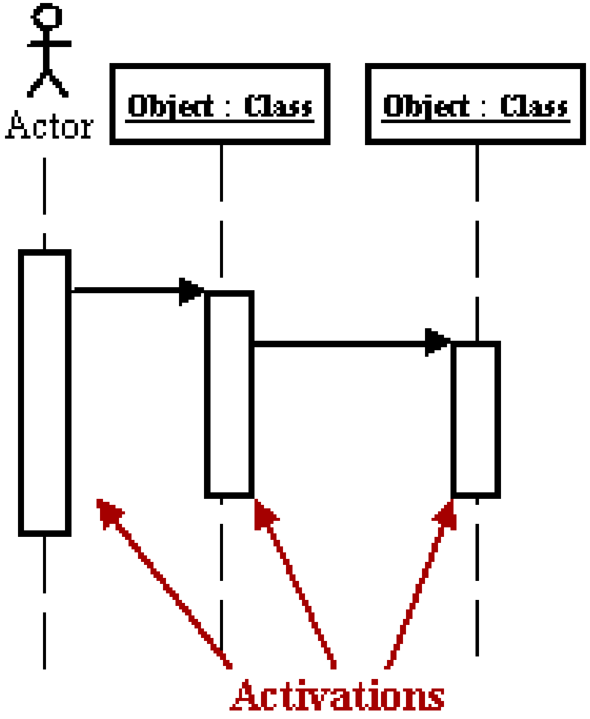
Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



Activation

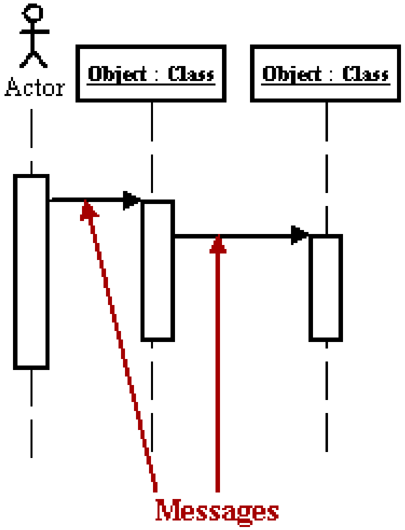
Activation boxes represent the time an object needs to complete a task.

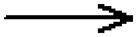
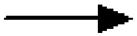
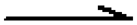
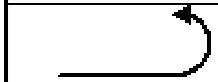
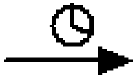


Sequence Diagrams

Messages

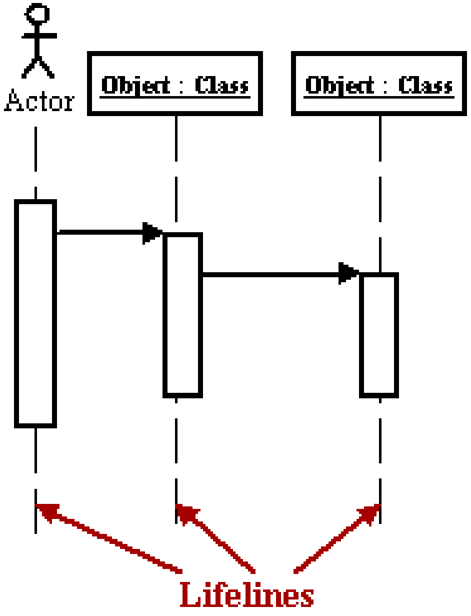
Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.



Arrow	Message type
	Simple
	Synchronous
	Asynchronous
	Balking
	Time out

Lifelines

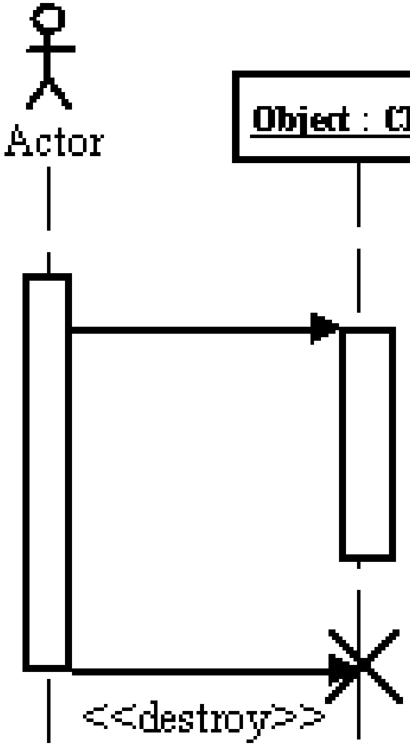
Lifelines are vertical dashed lines that indicate the object's presence over time.



Sequence Diagrams

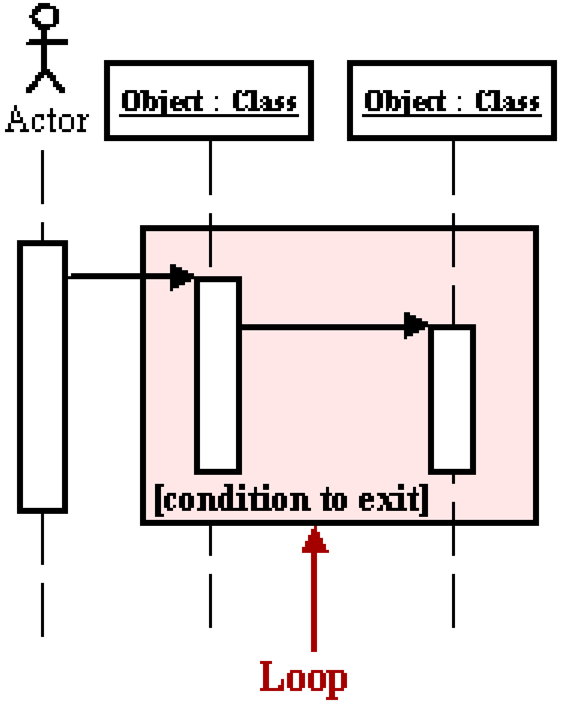
Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X.

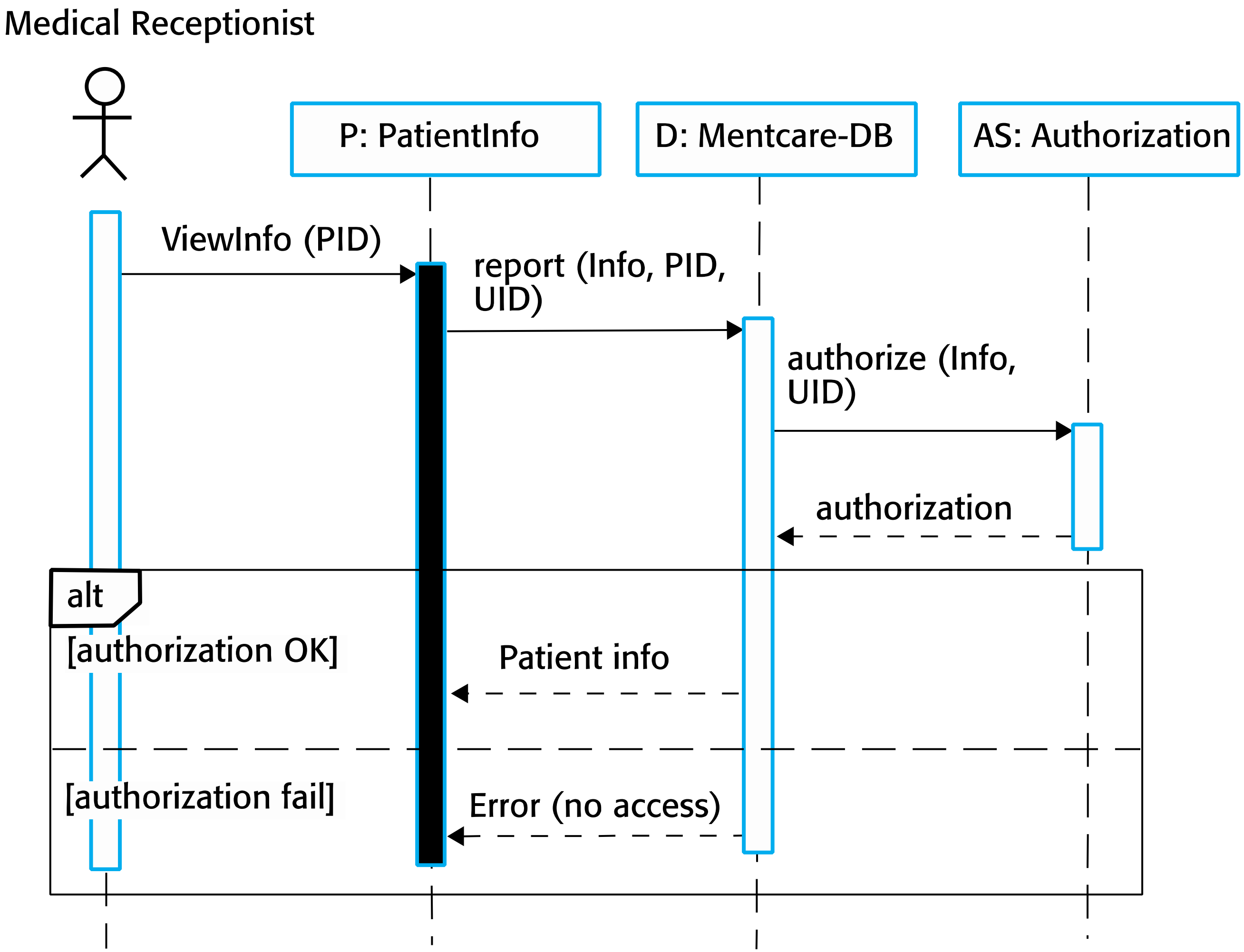


Loops

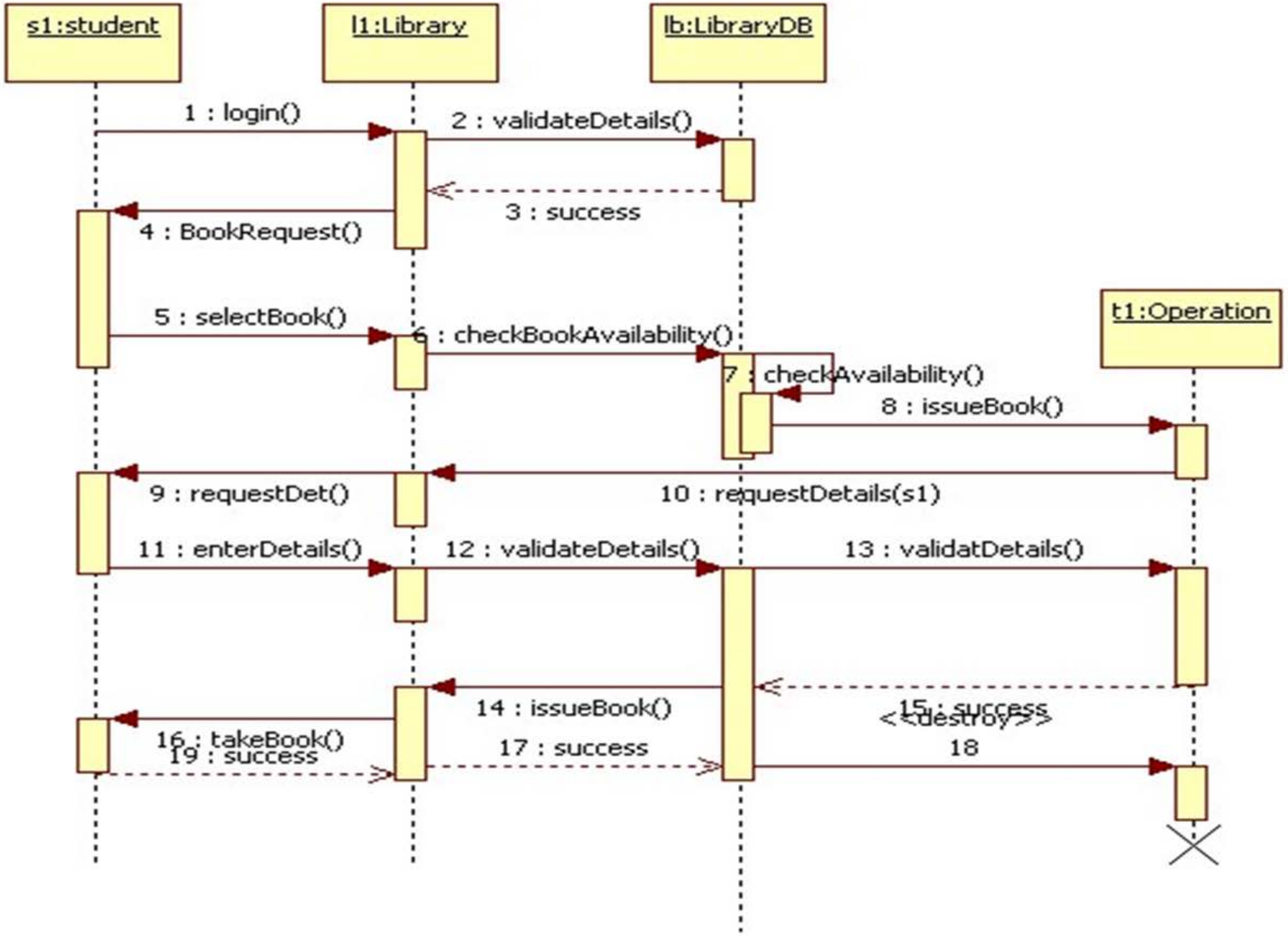
A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



Sequence diagram for View patient information



Sequence Diagrams



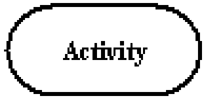
Activity Diagram

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of state chart diagram, it uses some of the same modeling conventions.

แสดงกระบวนการทำงานทางธุรกิจ หรือแสดงปฏิสัมพันธ์ของกลุ่มของออบเจ็กต์ เพื่อแสดงลำดับการไหลและกิจกรรมของแต่ละ Use Case หรือกิจกรรมของหลายคลาส

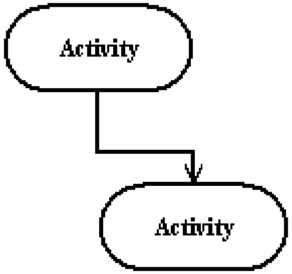
Action states

Action states represent the noninterruptible actions of objects.



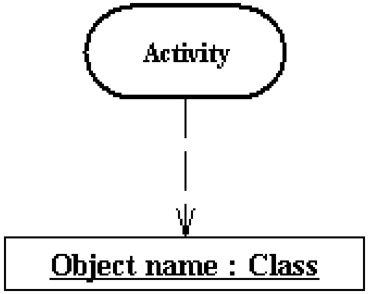
Action Flow

Action flow arrows illustrate the relationships among action states.



Object Flow

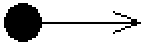
Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



Activity Diagram

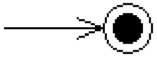
Initial State

A filled circle followed by an arrow represents the initial action state.



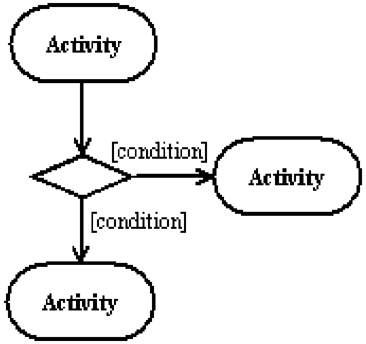
Final State

An arrow pointing to a filled circle nested inside another circle represents the final action state.



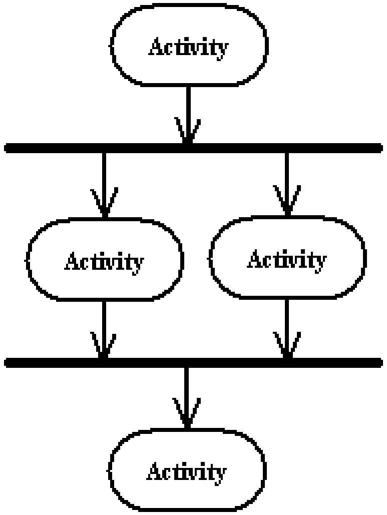
Branching

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



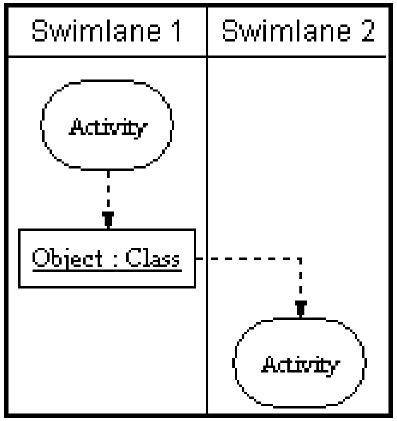
Synchronization

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.

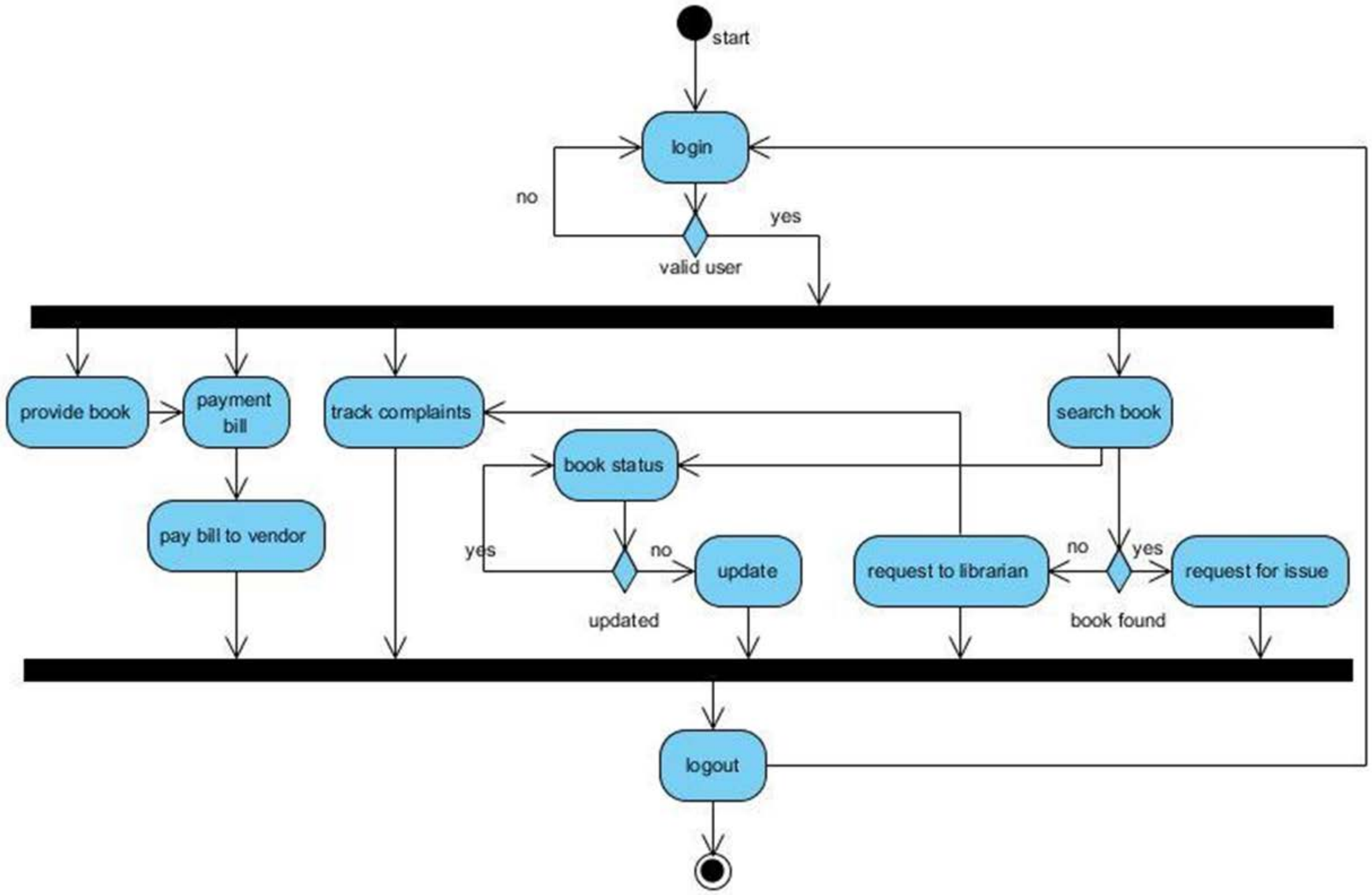


Swimlanes

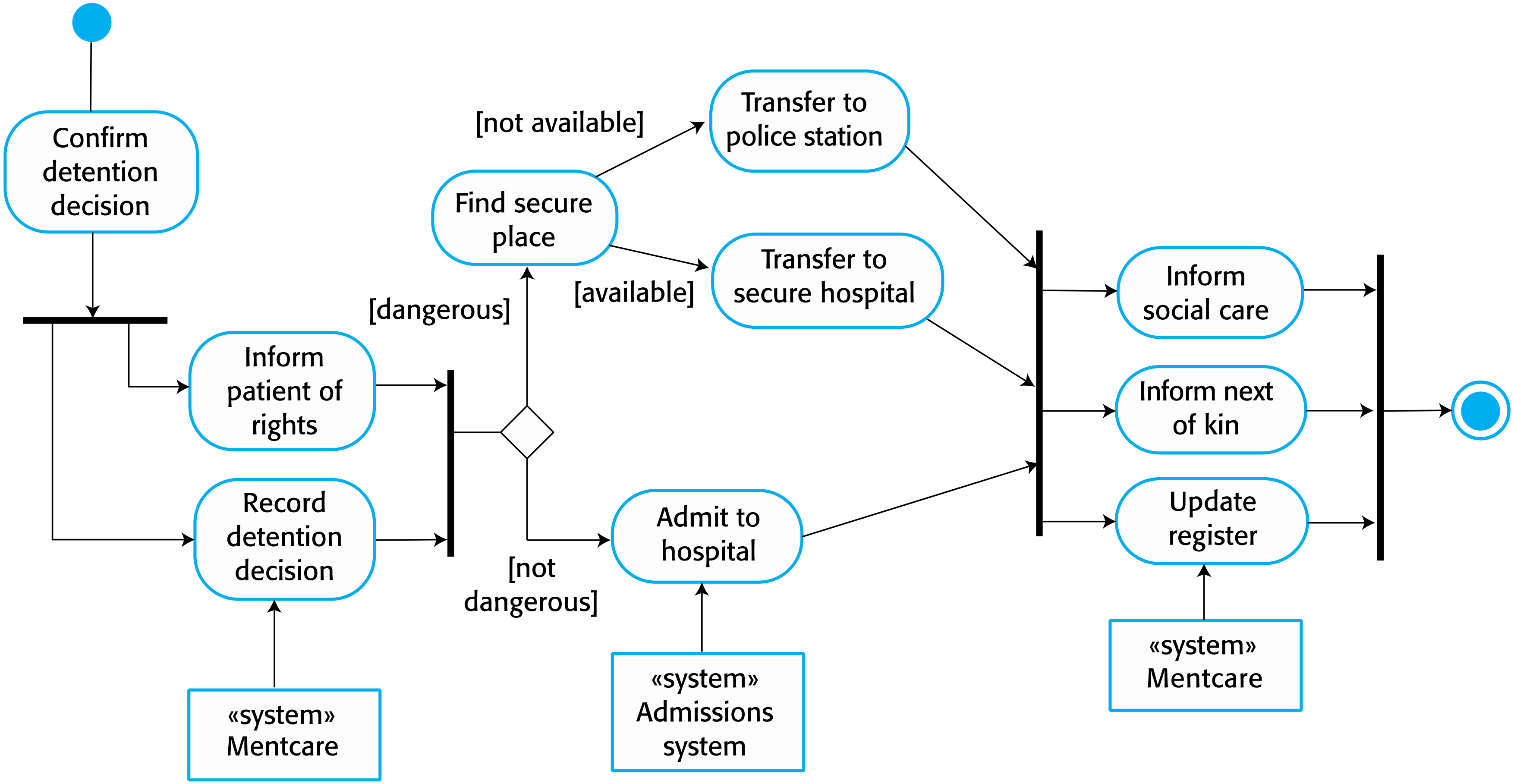
Swimlanes group related activities into one column.



Activity Diagram



Process model of involuntary detention



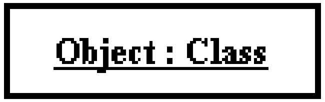
Collaboration Diagram

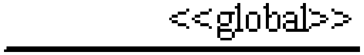
A collaboration diagram or **Communication Diagrams** like UML sequence diagrams, are used to explore the dynamic nature of your software. Collaboration diagrams **show the message flow between objects in an OO application, and also imply the basic associations (relationships) between classes.**

แสดงการปฏิสัมพันธ์ระหว่างออบเจ็กต์โดยไม่มีลำดับของเวลาเข้ามาเกี่ยวข้อง

Class roles

Class roles describe how objects behave. Use the UML object symbol to illustrate class roles, but don't list object attributes.



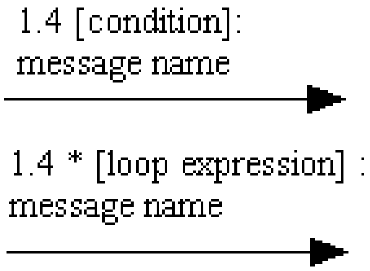


Association roles

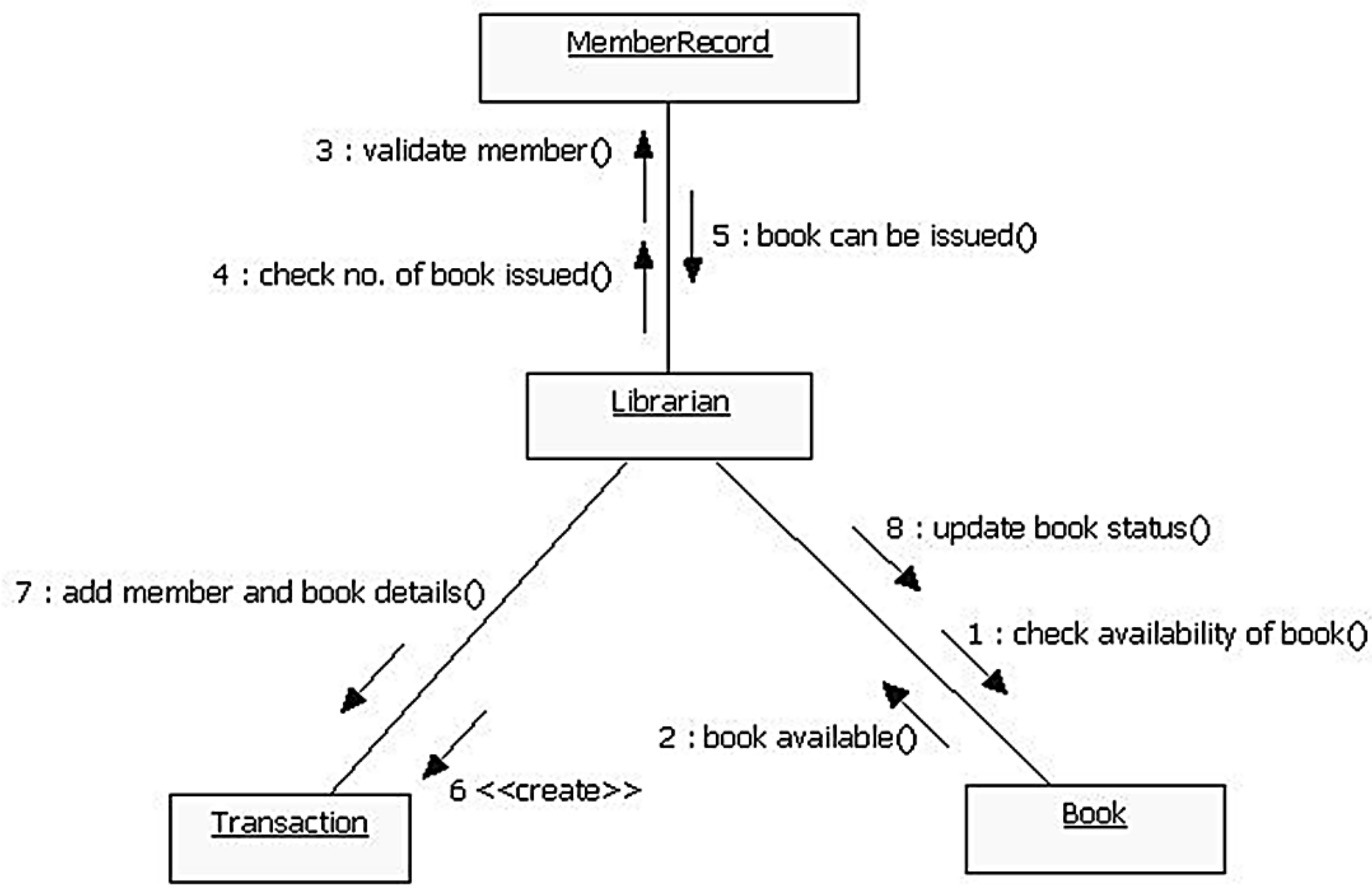
Association roles describe how an association will behave given a particular situation. You can draw association roles using simple lines labeled with stereotypes.

Messages

Unlike sequence diagrams, collaboration diagrams do not have an explicit way to denote time and instead number messages in order of execution. Sequence numbering can become nested using the Dewey decimal system. For example, nested messages under the first message are labeled 1.1, 1.2, 1.3, and so on. The a condition for a message is usually placed in square brackets immediately following the sequence number. Use a * after the sequence number to indicate a loop.



Collaboration Diagram



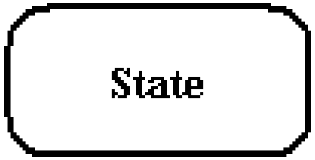
State Diagram

A state diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system.

แสดงสถานะต่าง ๆ ของแต่ละออบเจ็กต์ซึ่งอาจจะเปลี่ยนค่าไปตามกิจกรรมที่เกิดขึ้นในระบบ

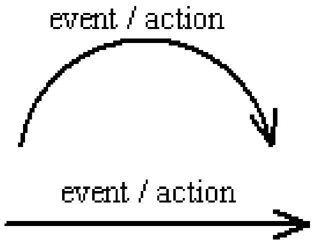
States

States represent situations during the life of an object. You can easily illustrate a state by using a rectangle with rounded corners.



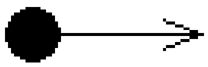
Transition

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it.



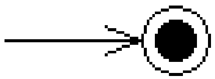
Initial State

A filled circle followed by an arrow represents the object's initial state.



Final State

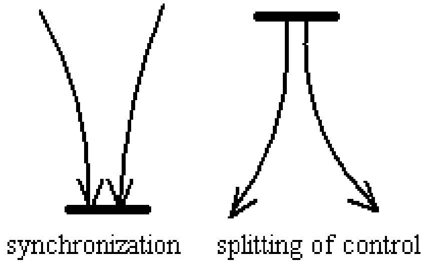
An arrow pointing to a filled circle nested inside another circle represents the object's final state.



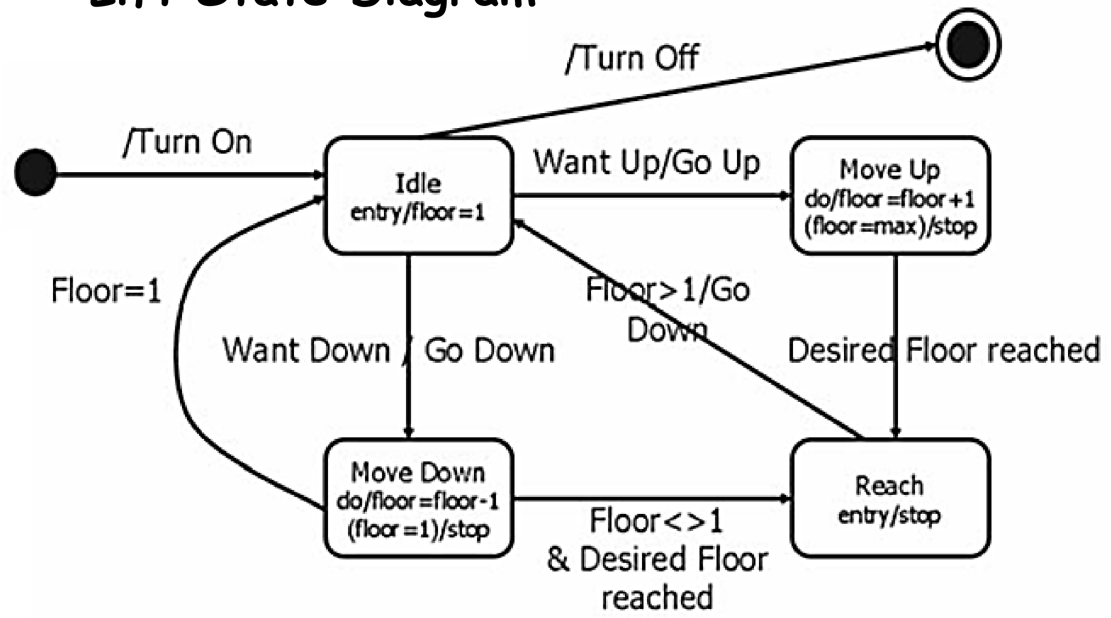
State Diagram

Synchronization and Splitting of Control

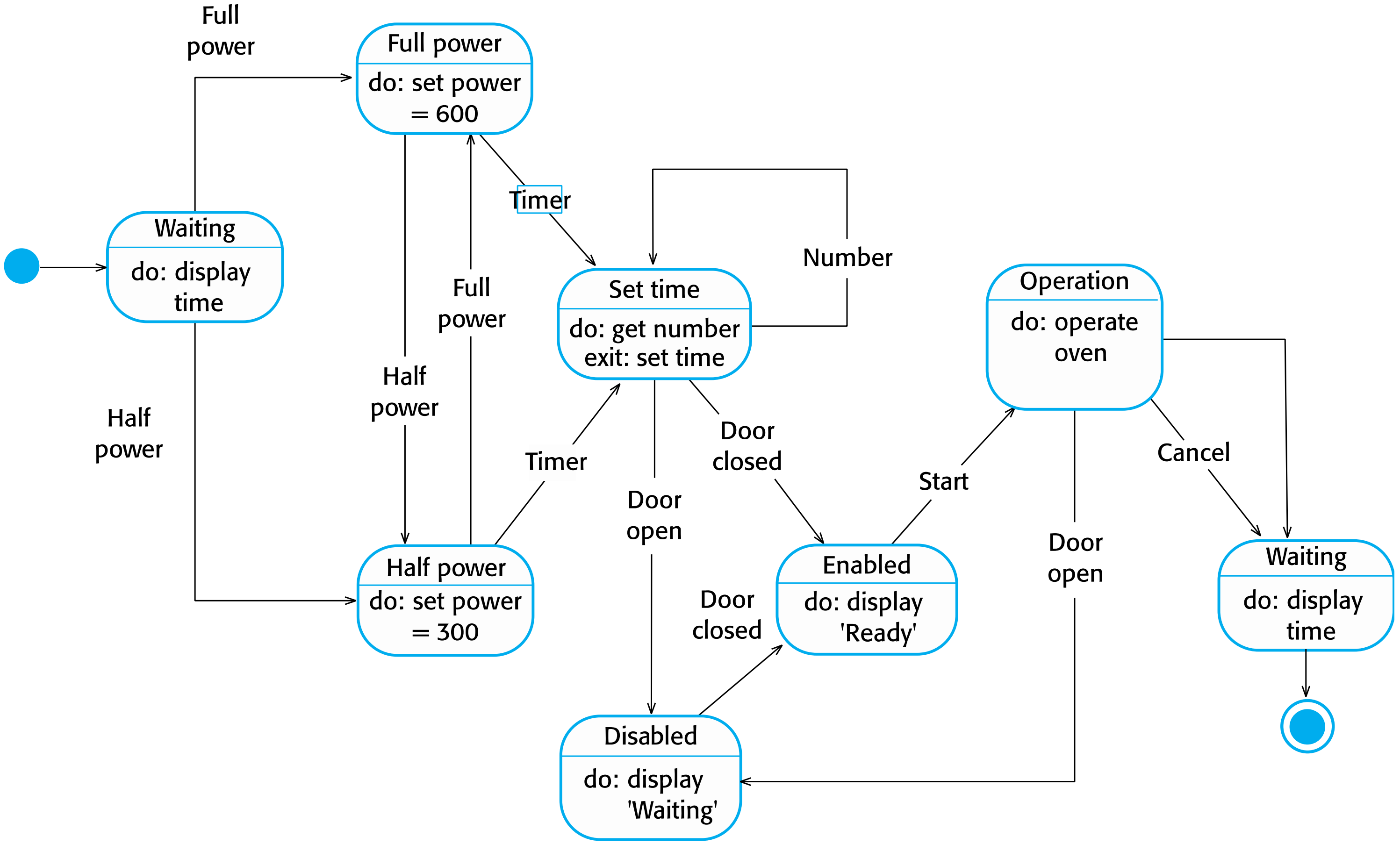
A short heavy bar with two transitions entering it represents a synchronization of control. A short heavy bar with two transitions leaving it represents a splitting of control that creates multiple states.



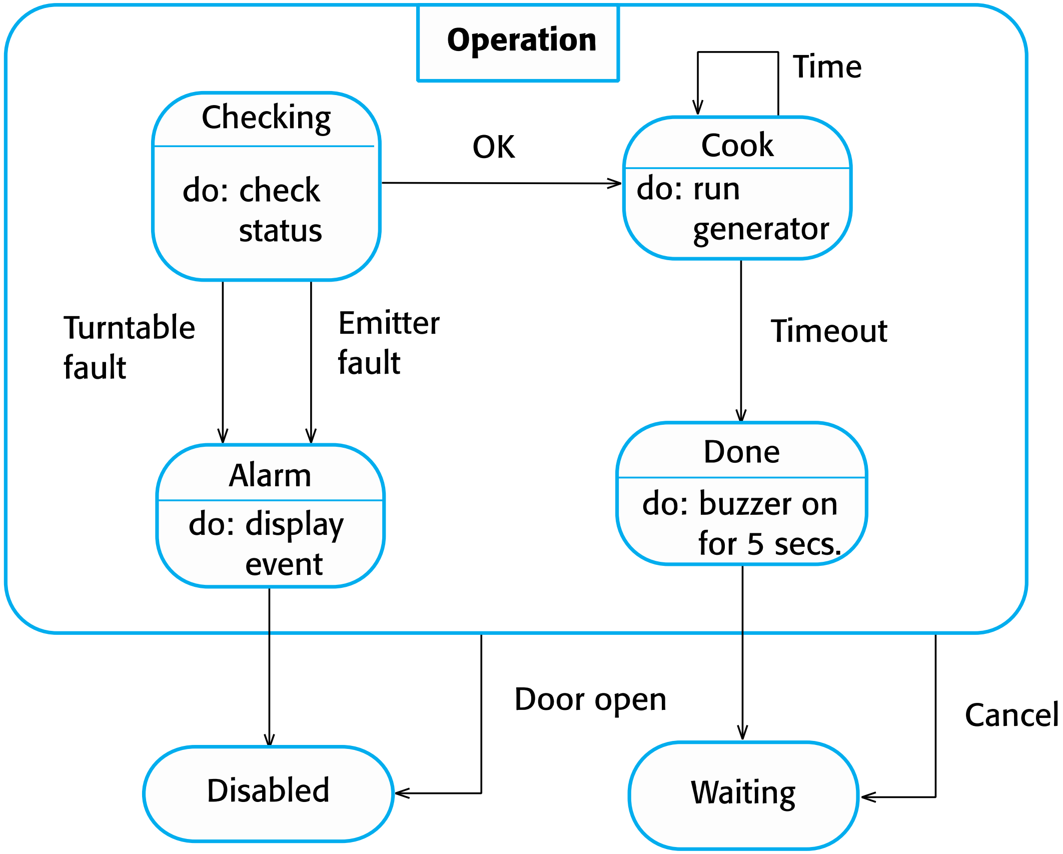
Lift State Diagram



State Diagram of a microwave oven



Microwave oven operation



States and stimuli for the microwave oven (a)

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.

States and stimuli for the microwave oven (b)

Stimulus	Description
Half power	The user has pressed the half-power button.
Full power	The user has pressed the full-power button.
Timer	The user has pressed one of the timer buttons.
Number	The user has pressed a numeric key.
Door open	The oven door switch is not closed.
Door closed	The oven door switch is closed.
Start	The user has pressed the Start button.
Cancel	The user has pressed the Cancel button.

References

- *Ian Sommerville, Software Engineering, 10th Edition Pearson Education, Addison-Wesley, 2015.*
- *Roger S. Pressman and Bruce R. Maxim. Software Engineering a Practitioner's Approach. Eighth Edition. McGraw-Hill, 2014.*
- *Ivan Marsic. Software Engineering. 2012.*
- *John Satzinger, Robert Jackson and Stephen Burd. Systems Analysis and Design in a Changing World. Sixth Edition. Course Technology, 2012.*