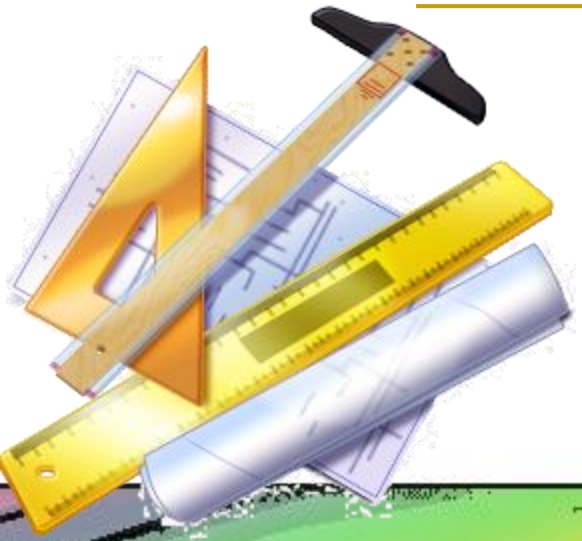
แนะนำให้รู้จัก Class และวัตถุ



Lecture 2

รศ.ดร.เยาวดี เต็มธนาภัทร์

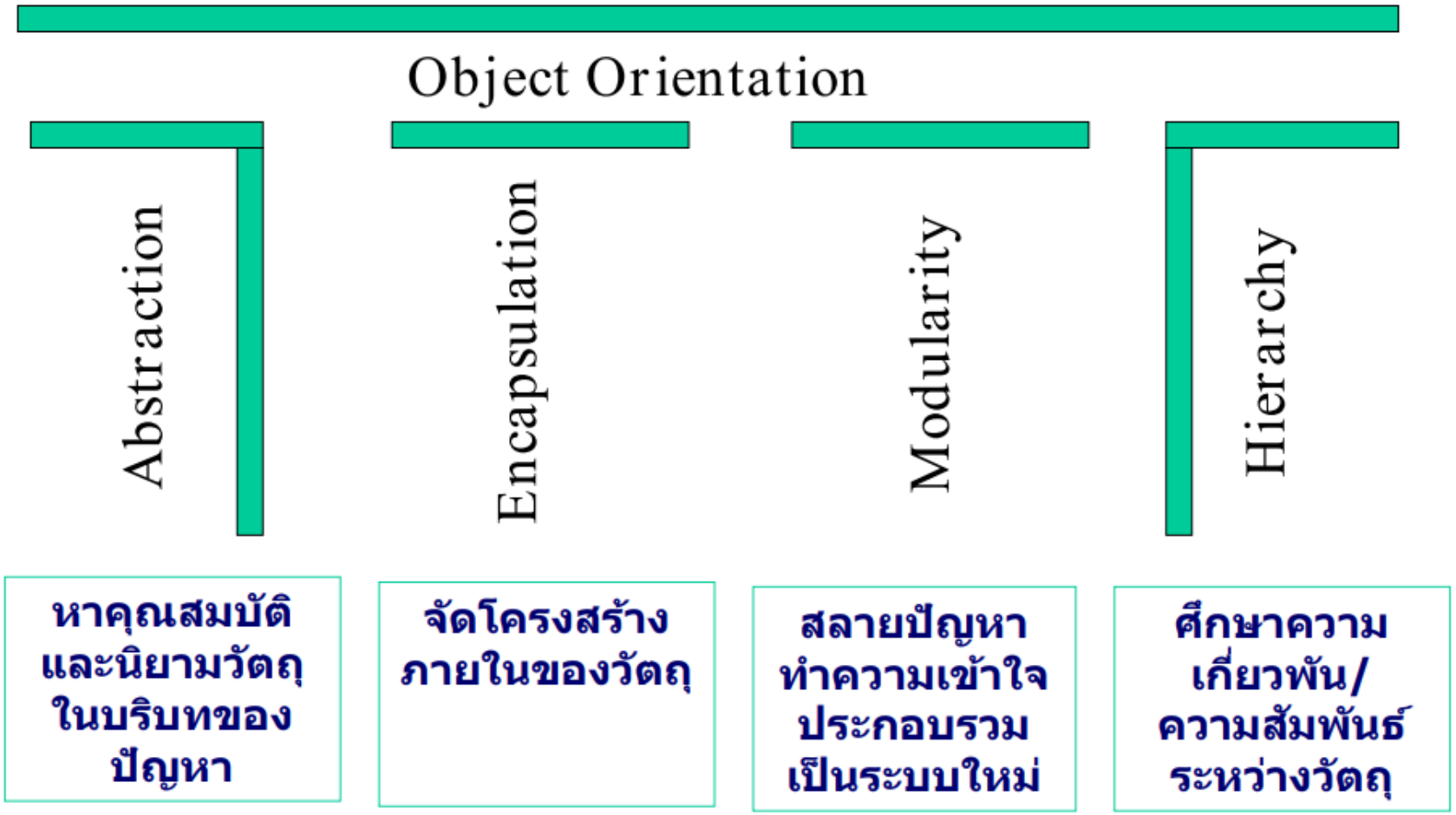
รศ.ปกรณ์ เสริมสุข

ปรับปรุงโดย อ.ดร.กฤตคม ศรีจิรานนท์

จุดมุ่งหมายของบทนี้

- เรียนรู้แนวคิดเกี่ยวกับการทำ Abstraction, ADT, Encapsulation
- เห็นประโยชน์ของการทำ encapsulation ใน program
- เข้าใจแนวคิดเกี่ยวกับ Class และ วัตถุ
- เรียนรู้การระบุหาวัตถุจากปัญหา
- เข้าใจข้อแตกต่างระหว่างวัตถุและตัวอ้างถึงวัตถุ (Object Reference)
- เรียนรู้การออกแบบอย่างง่าย ๆ โดยใช้ Class Diagram

หลักคิดของ Object Orientation



Abstraction

- Abstraction (การกำหนดสาระสำคัญ):
  - กำหนด**คุณลักษณะ**ที่สำคัญของสิ่งที่สนใจในปัญหา เพื่ออธิบายแนวคิดที่เป็นขอบเขตที่สนใจโดยละเว้นการมองรายละเอียดไว้
  - มนุษย์ใช้ Abstraction ในชีวิตประจำวันโดยระบุวัตถุและสิ่งที่วัตถุทำได้ตามระดับของสิ่งที่มนุษย์ต้องปฏิสัมพันธ์ด้วย
    - ในการขับรถ วัตถุที่สนใจคือรถยนต์ ส่วนประกอบของรถยนต์คือ เกียร์ clutch คันเร่ง พวงมาลัย เบรก สิ่งที่รถยนต์ทำได้คือการเคลื่อนไปข้างหน้า ซ้าย ขวา ถอยหลัง หรือหยุด
    - ในการซ่อมรถ วัตถุที่สนใจคือเครื่องยนต์ ส่วนประกอบที่สนใจคือ pump, carburetor, etc. สิ่งที่รถยนต์ทำได้คือ กลไกการทำงานของ pump
  - Abstraction มี 2 ประเภทคือ Function และ Data Abstraction

ตัวอย่าง : Abstraction ในบริบทต่างๆ

Functional Abstraction

- กำหนดหน้าที่ที่สนใจ
  - เพื่ออธิบายขอบเขตของความสามารถที่จะทำได้ โดยละเว้นการพิจารณารายละเอียดของสิ่งที่ต้องทำ
- ตัวอย่างเช่น
  - int find(int[] list, int searchedData): หาตำแหน่งของตัวเลขที่กำหนดในอาร์เรย์
  - Implementation ของ find()

ละเว้น

```
public static int find (int[] list, int searchedData) {
    for (int i = 0; i < list.length; i++){
        if (list[i] == searchedData){
            return i;
        }
    }
    return -1;
}
```

Data Abstraction

- กำหนดวัตถุที่สนใจ เพื่ออธิบายขอบเขตของข้อมูลของวัตถุ โดยละเว้นการพิจารณาโครงสร้างของการ implement ภายในวัตถุ
  - วันที่ (Date)
  - Implementation ของ Date อาจเป็น

```
class Date {
    int day;
    int month;
    int year;
}
```

ละเว้น

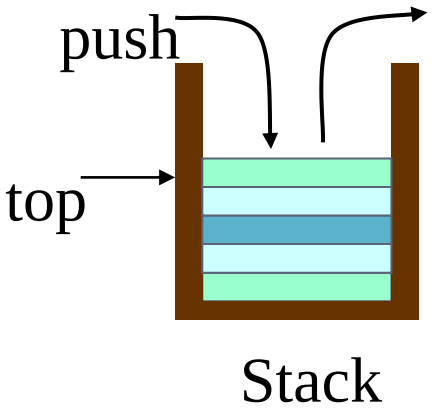
```
class Date {
    String date;
}
```

```
class Date {
    long timePassed;
}
```

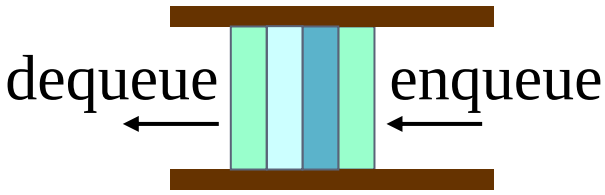
  - ทั่วไป เกี่ยวข้องกับ Functional Abstraction (เช่น วันที่ของวันก่อนหน้า n วัน)

Abstract Data Type

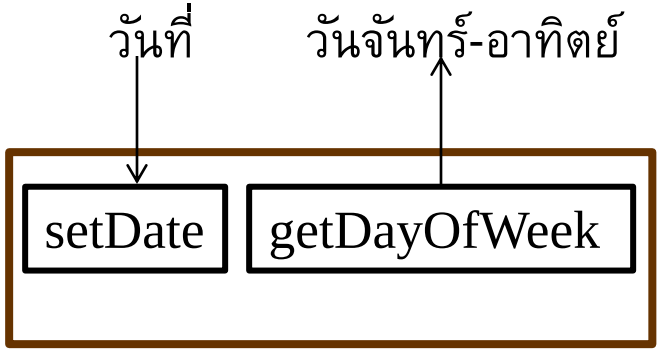
- Abstract Data Type (ADT):
  - อธิบายถึงโครงสร้างของข้อมูลโดย**ไม่**อธิบายรายละเอียดของ implementation แต่กำหนดบริการที่มีให้ใช้สำหรับโครงสร้างนั้น พร้อมความหมายของบริการ
  - ADT: ข้อกำหนดพฤติกรรม (behavioral specification) ของวัตถุ



Stack



Queue

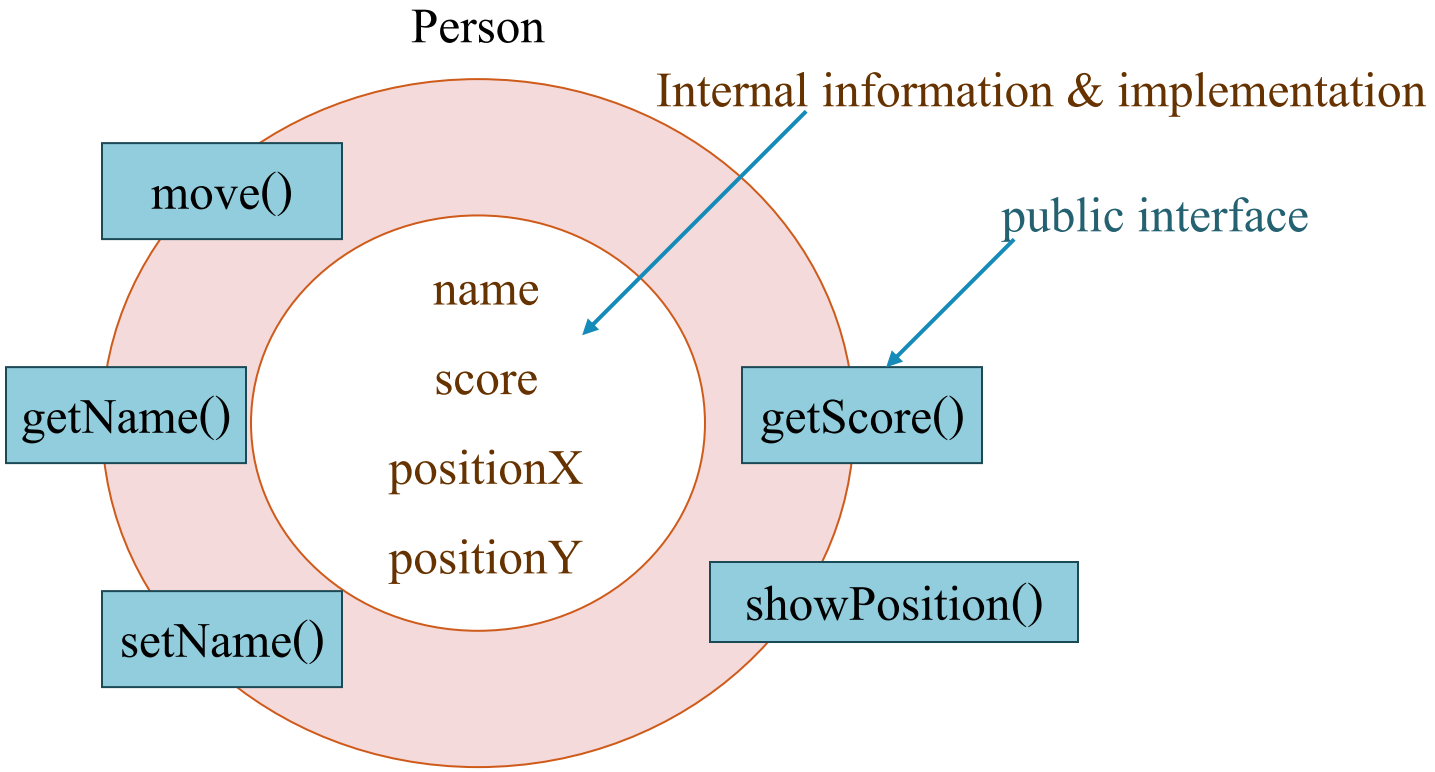


Date

Encapsulation

- Encapsulation เป็นการส่งเสริม abstraction โดยทำให้การเรียกใช้อยู่ในรูปของบริการ (WHAT) ไม่ใช่วิธีการทำ implementation (HOW)
  - การเปลี่ยนแปลงภายใน ไม่มีผลกระทบกับมุมมองที่มีให้สำหรับภายนอก
  - ป้องกันการเปลี่ยนสถานะในวัตถุในลักษณะที่ไม่พึงประสงค์จากผู้ใช้ภายนอก
- เป็นการบังคับการใช้นโยบายแบบ “เท่าที่จำเป็นต้องรู้ (Need to Know)”
- วัตถุนั้นถูก encapsulated ถ้าการ implementation และ interface ถูกแยกออกจากกัน โดยที่ผู้เรียกใช้รับรู้เพียงแต่ interface เท่านั้น
- Encapsulation อาจมองว่าเป็น ADT + information hiding

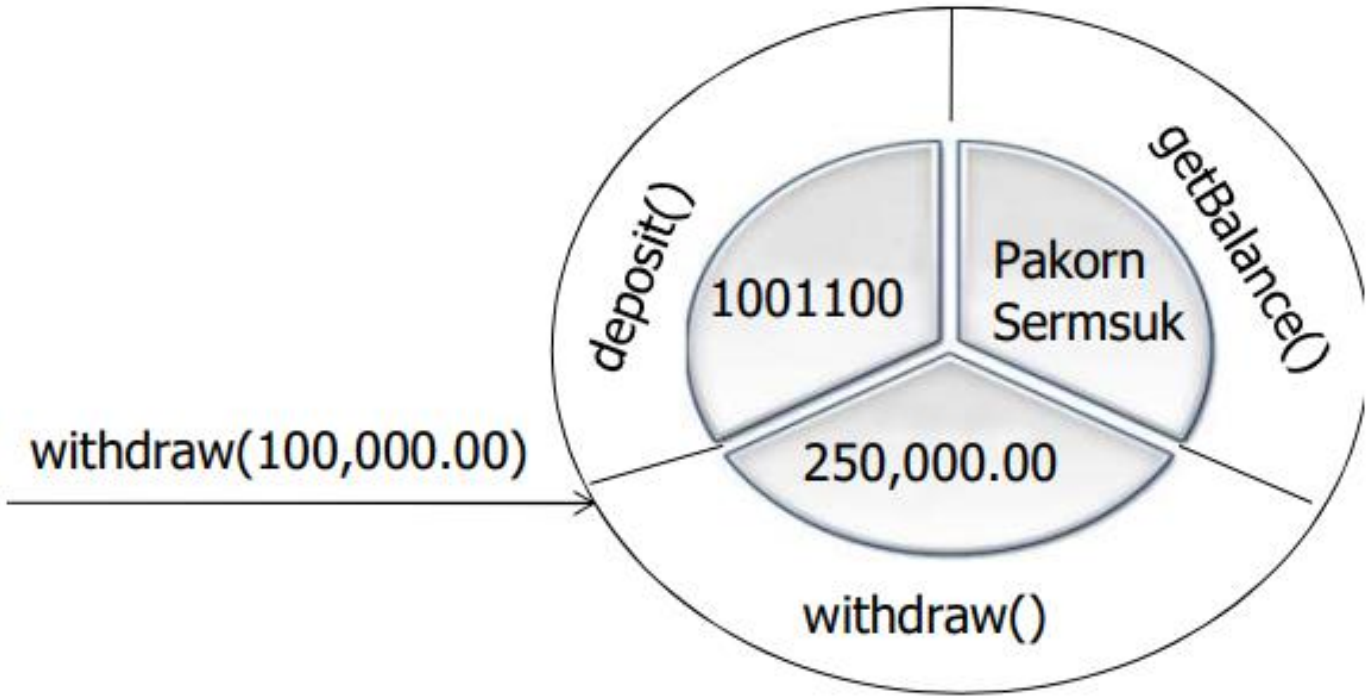
Encapsulation



Encapsulation ทำให้เกิดความปลอดภัย 2 ประเภท:

- ป้องกันการเปลี่ยนสถานะของวัตถุจากผู้ใช้ภายนอก
- การเปลี่ยนแปลงพฤติกรรมการทำงานไม่ส่งผลกระทบกับวัตถุอื่น ๆ

Encapsulation



วัตถุ

- การโปรแกรมเชิงวัตถุ คือการสร้างโมเดลจากวัตถุ
- วัตถุ
  - อาจเป็นสิ่งจับต้องได้ เช่น สินค้า
  - อาจเป็นนามธรรม เช่น การนัดหมาย
  - อาจเป็นกระบวนการทำงาน เช่น การประมวลผลเกรด
- องค์ประกอบของวัตถุ
  - พฤติกรรม หรือ Method: กิจกรรมที่วัตถุนั้นรู้หรือว่าสามารถทำได้
  - ลักษณะ หรือ Attribute: ส่วนที่ประกอบเป็นวัตถุ ค่าอาจเปลี่ยนไปตามเวลาและวัตถุ

แนวคิดการออกแบบ: การระบุหาวัตถุ

- ตัวอย่างเช่น โปรแกรมเกมที่มีคนเดินสู้กับสัตว์ประหลาดในห้อง
  - Person, Game, Room, Monster
- วัตถุ Person ประกอบด้วย
  - ลักษณะ: ชื่อ ตำแหน่ง คะแนน และ
  - พฤติกรรม เช่น move getScore, etc.

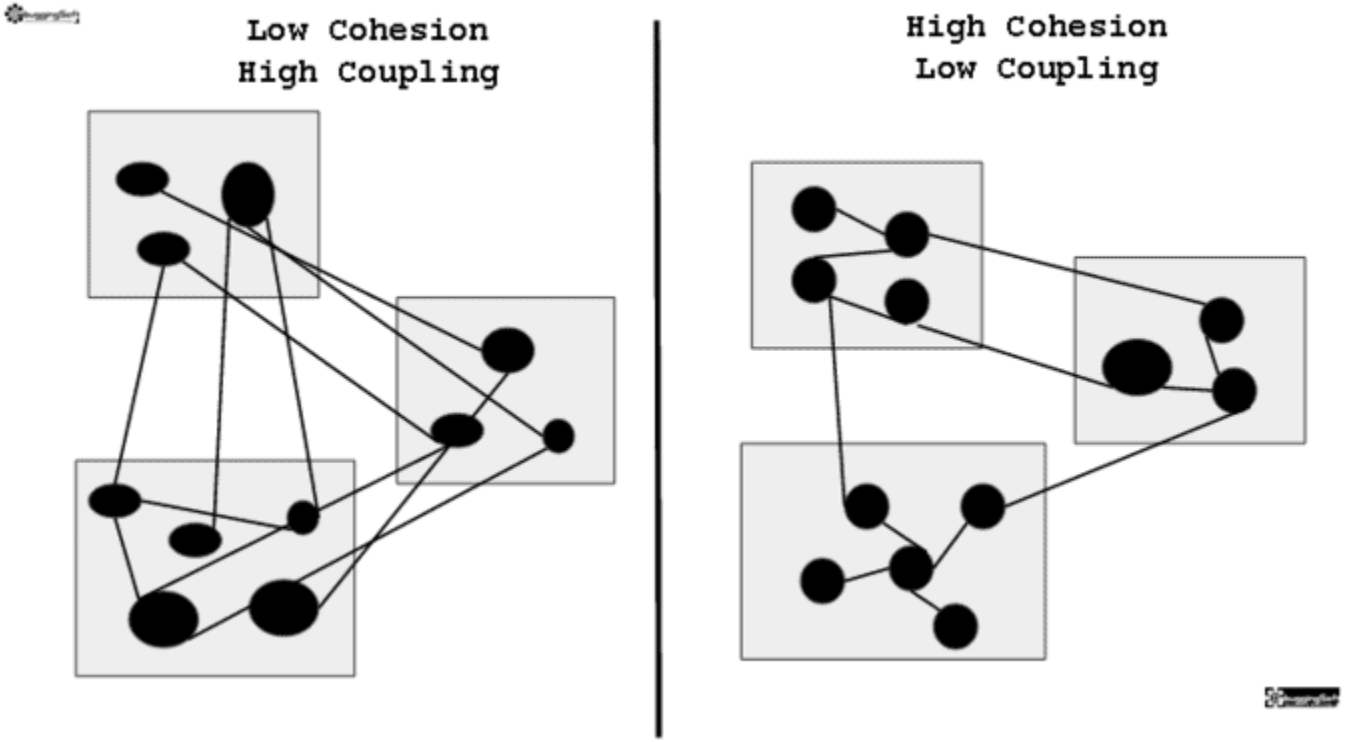
การทดสอบความเป็นวัตถุ

- ความเกี่ยวข้องกับปัญหา
  - อยู่ในขอบเขตของปัญหา
  - รับผิดชอบในการทำหน้าที่อย่างหนึ่งอย่างใดในปัญหา
  - อาจเป็นส่วนหนึ่งของวัตถุอื่นที่จำเป็นในปัญหา
- มีความเป็นเอกเทศจากวัตถุอื่น
  - ปรากฏอยู่ได้ด้วยตัวเอง และมีความเป็น modularity
- มี attributes และ methods ของตัวเอง

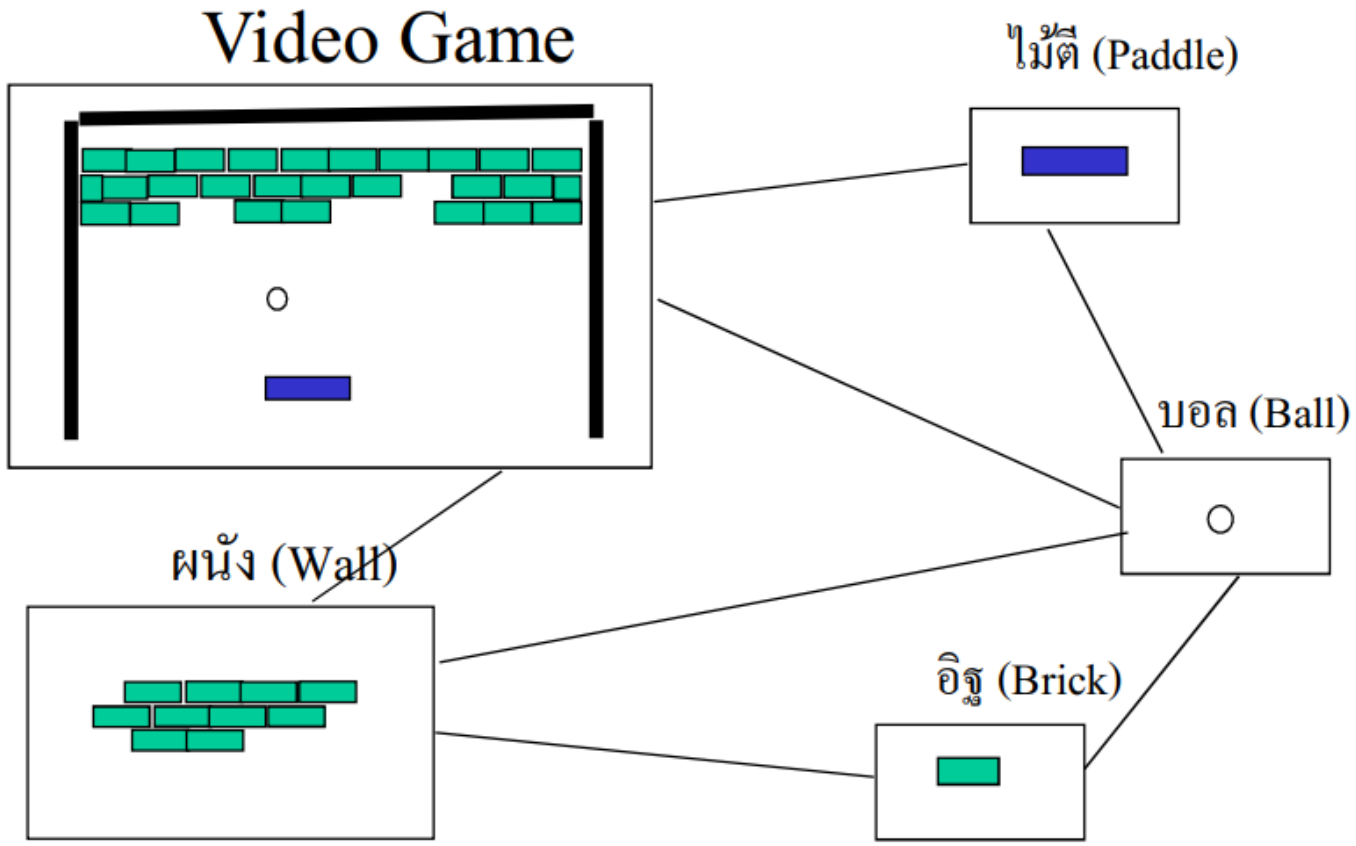
วัตถุควรมีความเป็น Modularity

- ลักษณะของการแยกวัตถุภายในระบบที่ดี
  - วัตถุควรมีลักษณะที่เบ็ดเสร็จภายในตัวเองและง่ายในการบริหารจัดการ
  - cohesiveness สูงและมี coupling แบบหลวม ๆ
- *Cohesiveness*: วัดความสัมพันธ์ภายในวัตถุเดียวกัน (intra-object relatedness)
  - ทั่วไปแสดงในรูป Interface จึงควรเป็น concept เดียวที่เกี่ยวกับวัตถุนั้นเท่านั้น
- *Coupling*: วัดความเป็นอิสระระหว่างวัตถุที่ต่างกัน
  - ทั่วไปแสดงถึงการขึ้นต่อกันของวัตถุกับวัตถุอื่น

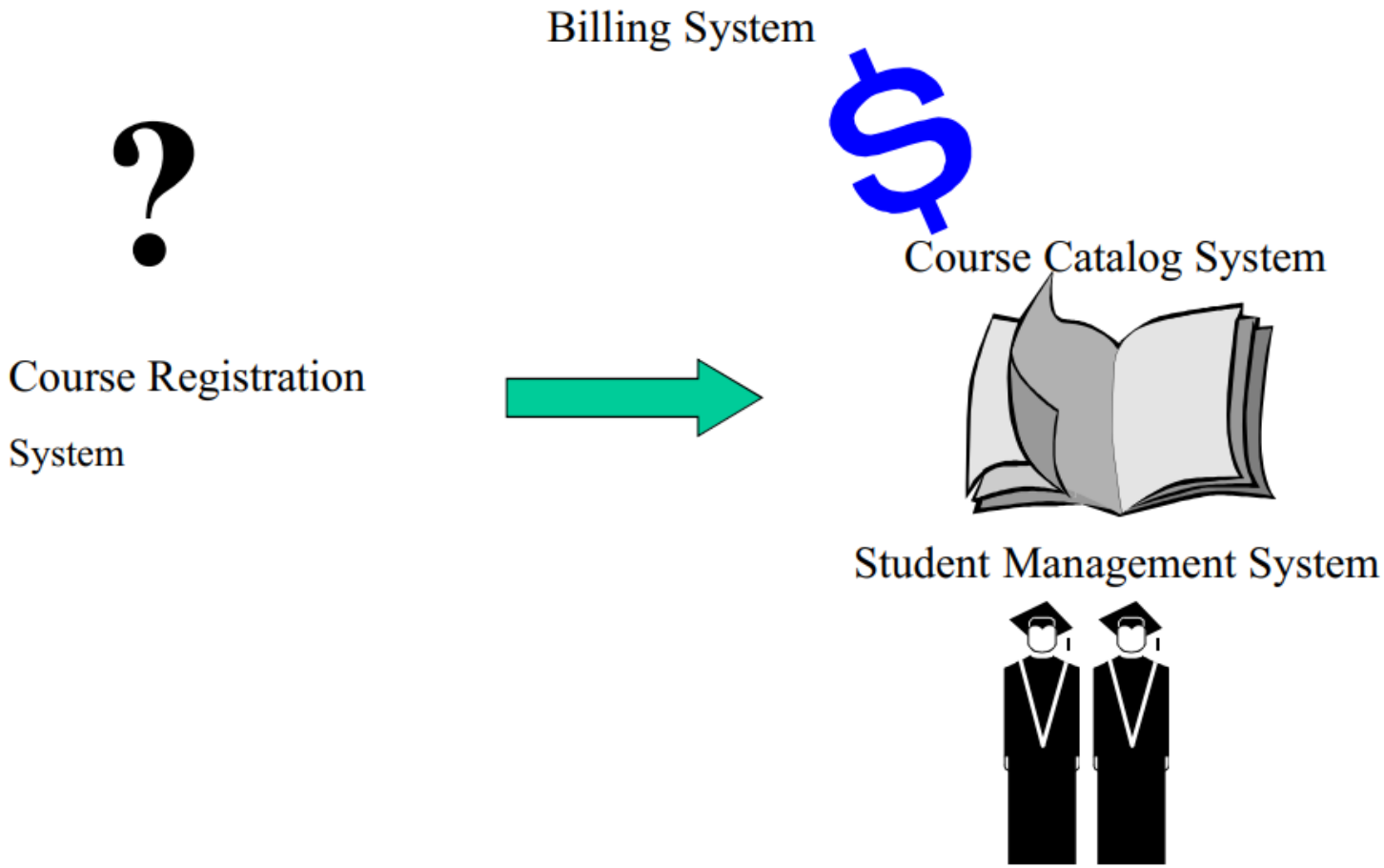
Cohesion VS Coupling



ตัวอย่าง : Modularity

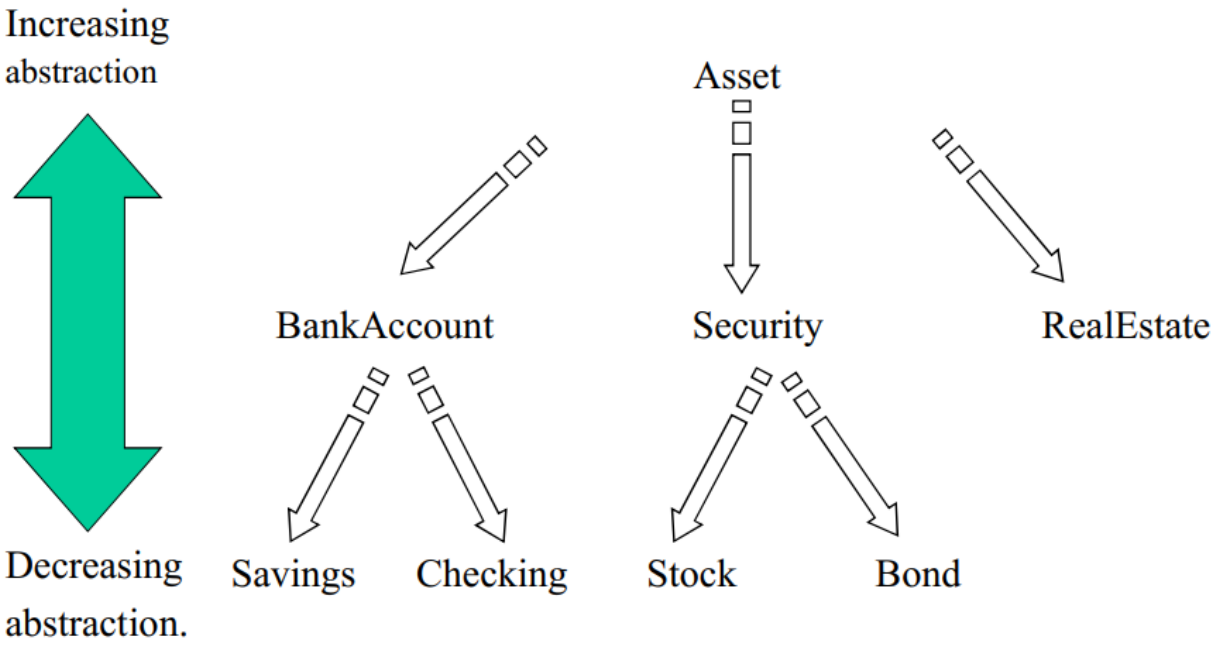


ตัวอย่าง : Modularity

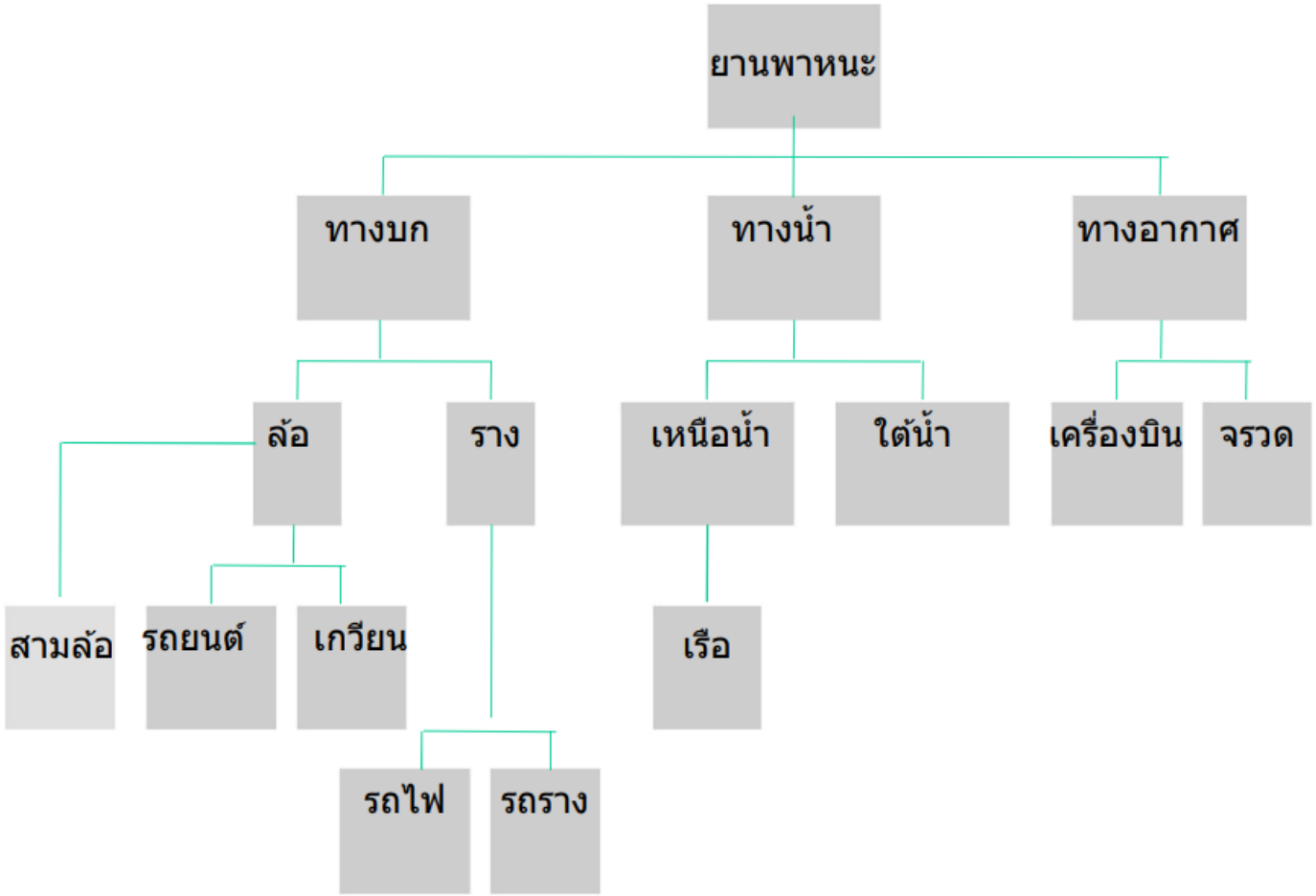


ลำดับชั้น (Hierarchy)

- จัดกลุ่มความสัมพันธ์ของสิ่งต่าง ๆ ในลักษณะโครงสร้าง โดย Elements ในระดับเดียวกันของต้นไม้เป็นตัวแทน (Abstraction) ของสิ่งที่อยู่ในระดับเดียวกัน



ตัวอย่าง : Hierarchy



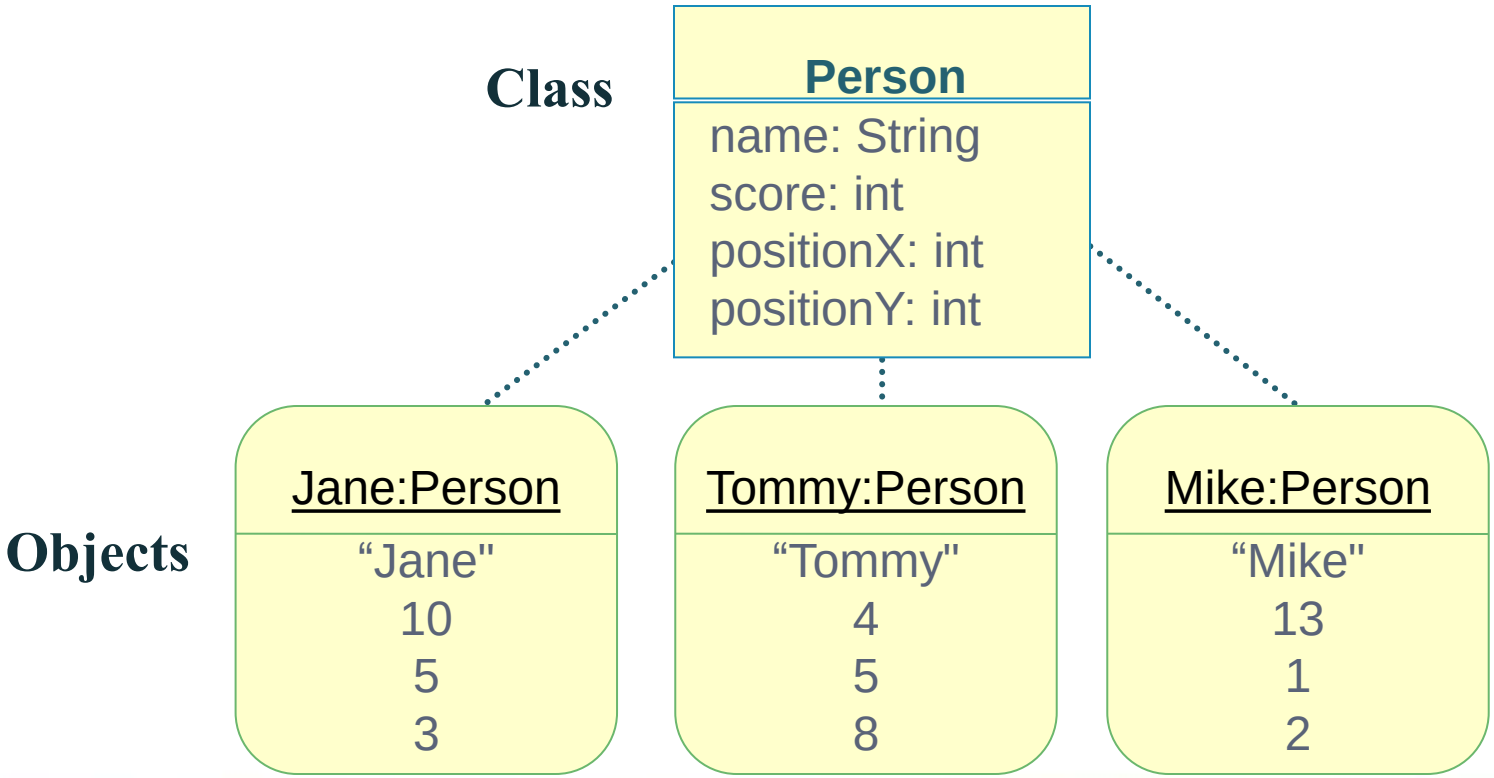
การสร้างแม่แบบ หรือคลาส (Class)

- เพื่ออธิบายโครงสร้าง (ลักษณะและพฤติกรรม) ร่วมของทุกวัตถุ (instance) ที่เป็นประเภทเดียวกัน
- ในการโปรแกรมเชิงวัตถุ
  - คลาส (class) ทำหน้าที่เป็นพิมพ์เขียวหรือ Template ที่ใช้สร้างวัตถุ
    - อธิบายลักษณะโดย ตัวแปร
    - อธิบายพฤติกรรมโดย method
  - จากนั้นใช้คลาสไปสร้างวัตถุเพื่อให้ทำงานตามที่ต้องการ
    - มีค่าลักษณะหรือสถานะ ซึ่งขอดูได้จากตัวแปรที่กำหนดในคลาส
    - สามารถเรียกให้ทำงานตามพฤติกรรมหรือ method ที่ถูกกำหนดไว้ในคลาสได้

ความสัมพันธ์ระหว่าง Class กับ Object

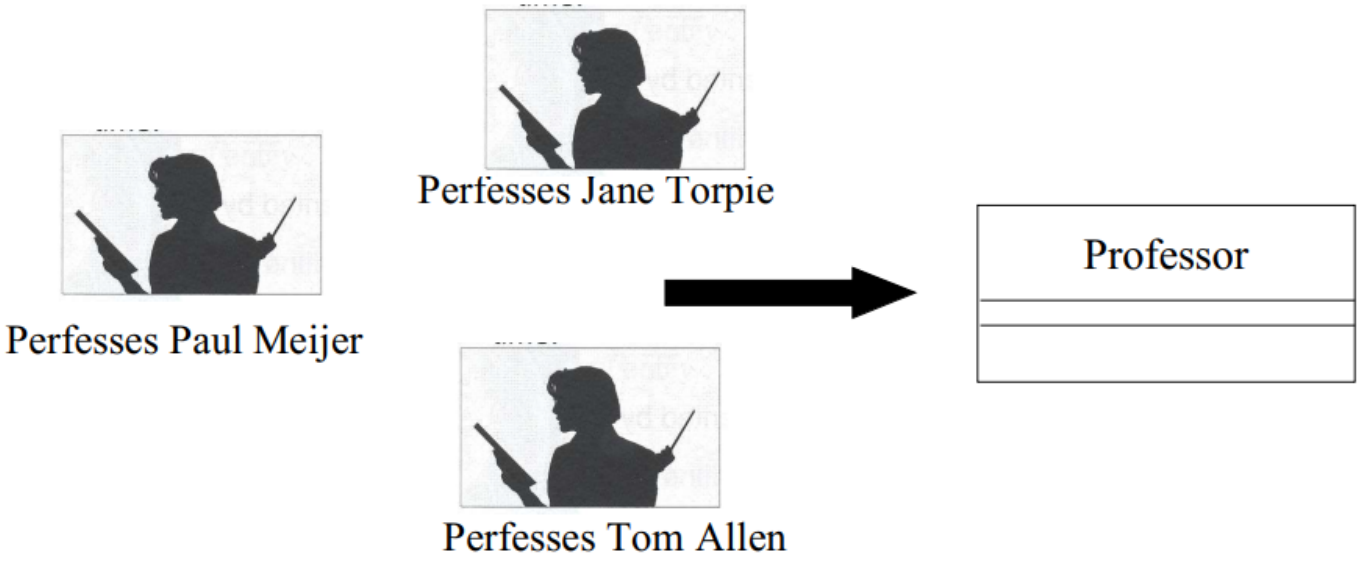
Object คือ Instance หรือผลผลิตของ class

1 class สามารถใช้สร้างวัตถุได้หลายอัน



ความสัมพันธ์ระหว่าง Class กับ Object

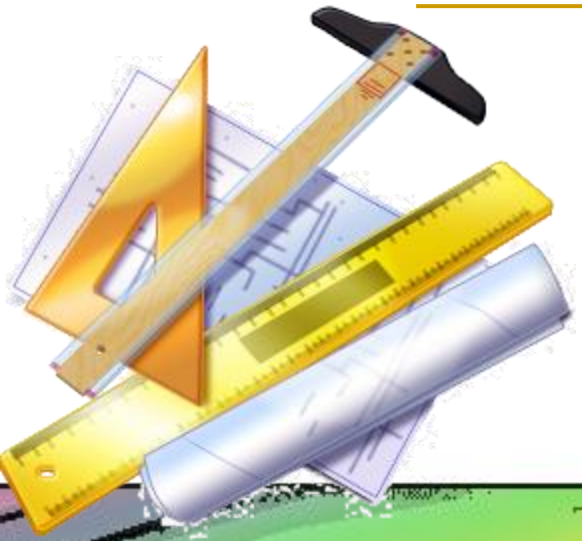
- Class เป็นนามธรรมที่นิยามถึงวัตถุใด ๆ
  - นิยามถึงคุณสมบัติโครงสร้าง และพฤติกรรมที่เป็นความสามารถของวัตถุในคลาสนั้น
  - เป็นเสมือนแบบพิมพ์ (Template) สำหรับการสร้างวัตถุ
- Objects ถูกจัดกลุ่มเป็นคลาส



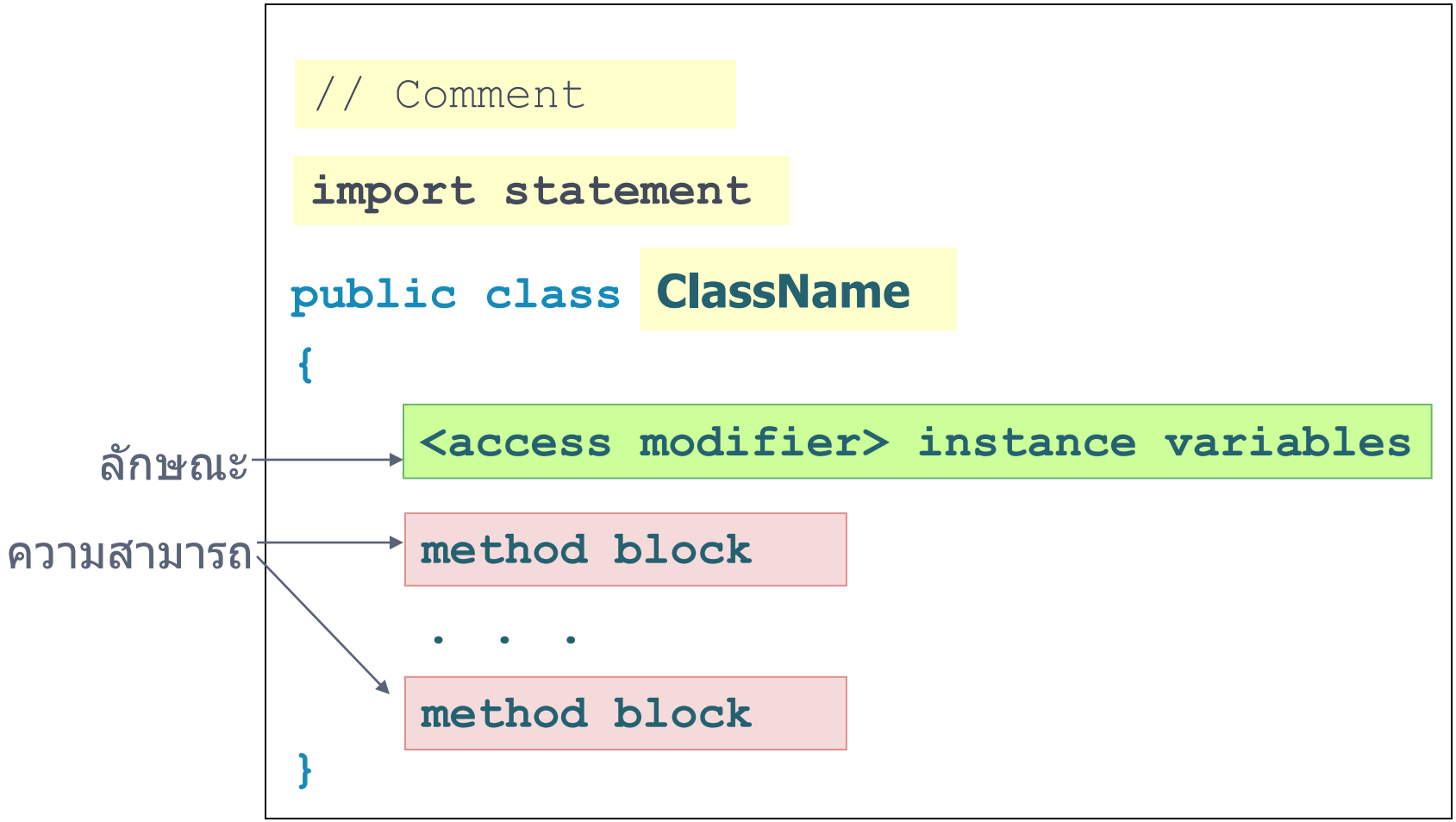
ตัวอย่าง : หา Classes และ Objects

- สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยธรรมศาสตร์ ได้เปิดให้บริการห้องคอมพิวเตอร์สำหรับนักศึกษาทั้งนักศึกษาปริญญาตรี ปริญญาโท โดยนักศึกษาที่ต้องการใช้ห้องบริการต้องมาลงชื่อ โดยนักศึกษาต้องแจ้งชื่อ นามสกุล รหัสนักศึกษา ชั้นปี พร้อมสแกนลายนิ้วมือ สำหรับนักศึกษาปริญญาโทต้องแจ้งโปรแกรมการเรียนด้วย การเข้าใช้บริการทุกครั้งนักศึกษาต้องสแกนนิ้วมือที่เครื่องสแกนหน้า ห้องถึงจะเข้าใช้บริการได้
- ปัจจุบันสาขาวิชาฯ มีห้องบริการคอมพิวเตอร์ 5 ห้องคือหองหมายเลข101 102 103 104 และ 105 แต่ละห้องรับนักศึกษาได้ 50 คน นักศึกษาสามารถใช้บริการได้ทุกห้อง ยกเว้นห้อง 105 ที่ให้บริการสำหรับนักศึกษาปริญญาโทเท่านั้น จากการสำรวจ ปรากฏว่า มีนักศึกษาปริญญาตรี 4 คน ได้แก่ สมชาย สมหญิง สมบัติ สมลักษณ์ และนักศึกษาปริญญาโทโปรแกรมทำวิทยานิพนธ์เพียง 1 คนได้แก่ สมยศ ที่มาลงรายชื่อ

Java กับการ implement คลาส การสร้างวัตถุและแนวคิดเกี่ยวกับ encapsulation



การ implement Class อย่างง่าย



จาวา

- คลาส implement *ลักษณะ*โดยใช้ *instance variables*
  - หน่วยความจำที่จองไว้สำหรับตัวแปรวัตถุ (instance variables)
- คลาส implement *พฤติกรรม*ของวัตถุโดยใช้ *methods*
  - ภายในเมท็อด เป็นลำดับคำสั่งการทำงาน
  - วัตถุทำงานร่วมกันโดยการส่ง messages ไปยังอีกวัตถุ
  - แต่ละ message เป็นการ “invokes” (เรียก) เมท็อด
  - ตัวอย่างพฤติกรรมของวัตถุ เช่นเมท็อดเพื่อสร้างตัววัตถุเอง (constructor), เข้าถึง (getter), เปลี่ยนค่า (setter)

การประกาศตัวแปรวัตถุ (Instance Variables)

- การประกาศตัวแปรเพื่อจองพื้นที่ให้ตัวแปรที่เป็นชนิดหรือคลาสนั้น
- Syntax:

```
<type name> <variable name>;
```

  - `<type name>` = ชื่อของชนิดหรือคลาส
  - `<variable name>` = กลุ่มลำดับของตัวแปรแยกจากกันด้วย “,”
- การประกาศตัวแปรวัตถุ เป็นการกำหนดตัวแปรภายในคลาส
  - **ตัวอย่าง**: คนในเกม (Person) มีคะแนน ชื่อ และตำแหน่งในแนวแกน X และ Y

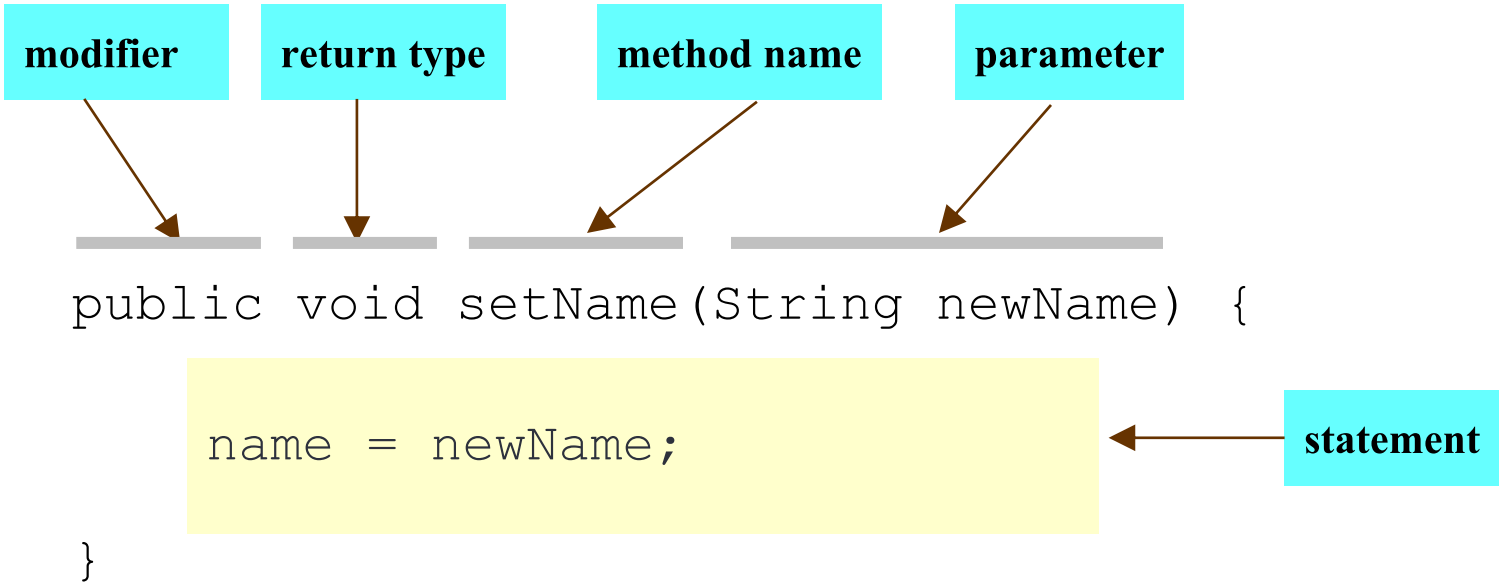
```
String name;
int score, positionX, positionY;
```

ตัวอย่างพฤติกรรม

- วัตถุคน มีหน้าที่พื้นฐานที่ควรจะทำได้ เช่น
  - เคลื่อนที่ (move) → `move`
  - ตั้งค่าชื่อ (set name) → `setName`
  - บอกค่าชื่อ (get name) → `getName`

การกำหนด method

```
<modifiers> <return type> <method name> (<parameter list>)
{
   <statement list>
}
```



ตัวอย่างการ implement คลาส Person

```
public class Person {
    private String name;
    private int score = 0, positionX = 0, positionY = 0;

    public void move(int horizantalDistance, int verticalDistance) {
        positionX = positionX + horizantalDistance;
        positionY = positionY + verticalDistance;
        score = score + 5;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void showInfo() {
        System.out.println("My postion is at x: " + positionX +
                " and y: " + positionY + " score: " + score);
    }
}
```

การสร้างวัตถุจากคลาส

- ขั้นตอนการสร้างวัตถุ หรือที่เรียกว่า Instantiate ทำได้โดย
  - ใช้ **new** operator
  - ตามด้วยชื่อของคลาส
  - ตามด้วย parameters ที่ใช้สำหรับการสร้าง
- เพื่อให้วัตถุที่สร้างขึ้นถูกอ้างอิงได้ในภายหลัง โดยมีตัวแปรอ้างอิง
  - เราให้ค่า (assign) กับตัวแปร ผ่านเครื่องหมาย =
- ตัวอย่าง:

```
mike = new Person();
```

การสร้าง object

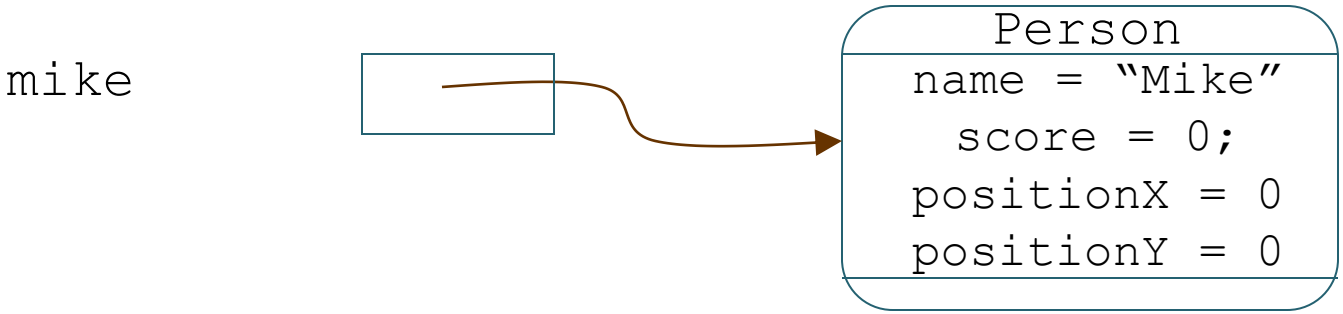
```
Person mike;
```

- เป็นเพียงการสร้างตัวแปรที่ใช้อ้างถึงวัตถุคลาส Person

```
new Person();
```

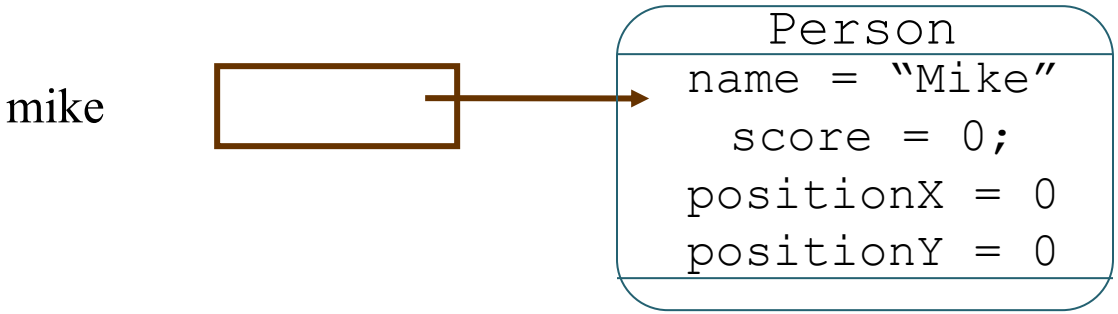
- เป็นการสร้าง object Account โดยไม่มีตัวแปรอ้างถึง
- เมื่อรวม 2 ขั้นตอนข้างต้น

```
Person mike = new Person();
```



ตัวแปรที่อ้างถึงวัตถุ (Object Reference)

```
Person mike;
mike =  new Person();
```



Messages และ Methods

- ในการขอให้วัตถุของคลาสใดทำงาน ทำได้โดยการส่ง message ไปยังวัตถุนั้น
  - การส่ง message หรือการเรียกให้ทำงาน ทำได้โดยใช้ "dot" notation
  - MyHelloWorld ส่ง message ไปยัง Object greeting เพื่อให้พิมพ์ Hello World

```
greeting.say("Hello World!");
```

- ตัวอย่าง เราสามารถเลื่อนตำแหน่งของคนจากตำแหน่ง x =0, y = 0 ไปที่ x=3, y =2 โดย

```
mike.move(3,2);
```

การเขียนคลาสทดสอบ (อีกครั้ง)

- ประกาศคลาส ที่ภายในมีเมท็อด main
- สร้างวัตถุที่ต้องการทดสอบ (new)
- ส่ง message ไปยังวัตถุเพื่อให้ทำงาน (dot)

```
public class PersonTest {
    public static void main(String[] args) {
        Person person1 = new Person();
        person1.setName("Jane");
        person1.move(5, 0);
        Person person2 = new Person();
        person2.setName("Tommy");
        person2.move(4, 0);
        person2.move(0, 3);
        person1.showInfo();
        person2.showInfo();
    }
}
```

Public VS Private

- public กับ private เป็นการกำหนดความสามารถในการเข้าถึง (accessibility) ของ attributes และ methods
- ไม่แนะนำให้ยกเว้นการกำหนด public หรือ private ของ attribute หรือ methods ถึงแม้ว่าการไม่ใส่ก็สามารถทำงานได้ก็ตาม
- private เป็นการประกาศว่า attribute หรือ method นั้นเป็นส่วนตัวใช้ได้เพียงภายใน class นั้นเอง
- public อนุญาตให้ method ภายนอกเรียกใช้ method หรือ attribute นั้นได้

Encapsulation

- การประกาศให้ attribute เป็น private เพื่อประกันความถูกต้องในการใช้ข้อมูลของ class นั้น
- ตัวอย่าง

```
public void move(int horizantalDistance, int verticalDistance) {
    positionX = positionX + horizantalDistance;
    positionY = positionY + verticalDistance;
    score = score + 5;
}                   15
```

ภาษาเชิงวัตถุ ทำ Encapsulation (ประกันความถูกต้องของการใช้ข้อมูล) ผ่านตัวขยาย public vs private

ตัวอย่าง (ต่อ)

หากสามารถเข้าถึง attribute ได้โดยตรง อาจกำหนดค่า score ของผู้เล่นเป็นอะไรก็ได้

```
...
mike.score = 2000;
...
```

Attribute และ Method สำหรับการเข้าถึง

- ประกาศ attribute เป็น private และสร้าง method สำหรับใช้ในการเข้าถึงข้อมูลนั้น -- set และ get methods
  - ตัวอย่าง

```
public void setScore(int newScore) {
     score = newScore;
}
public int getScore( ) {
     return score;
}
```

Note ทั่วไปจะกำหนดค่าข้อมูลผ่านวิธี set แต่ใน Person ไม่กำหนดคะแนนเอง (เพิ่มคะแนนเมื่อตอนผู้เล่นเคลื่อนที่เท่านั้น)

- ไม่ต้องกำหนด set ในกรณีของ attribute ที่เป็นค่าคงที่ (constant)

การวิเคราะห์และการออกแบบ (Analysis & Design)

- กฎพื้นฐาน: พยายามค้นหา
  - **ค้นหา class:** ว่าอะไรคือ objects, ต้องแบ่ง project ของเราอย่างไรให้เป็นส่วนย่อย (component parts)
  - **กำหนดพฤติกรรมของแต่ละ class:** ต้องมี method อะไรบ้างใน class
  - **กำหนดความสัมพันธ์ระหว่าง class:** อะไรบ้างที่เป็นส่วน interfaces ของ object เหล่านี้, messages อะไรบ้างที่ต้องการเพื่อทำให้สามารถติดต่อกันได้ระหว่าง objects

ความสัมพันธ์ระหว่าง class

- เกี่ยวพันกับ (Association)
- เป็นส่วนประกอบของ (Aggregation)
- พึ่งพิง (Dependency)
- เป็นชนิดหนึ่งของ (Generalization)

เกี่ยวพันกับ (Association)

- ความสัมพันธ์ระหว่างสองสิ่งที่บ่งชี้ว่าสองสิ่งนั้นมีความเกี่ยวพันธ์กันอย่างเป็นโครงสร้าง
- Association เป็นความสัมพันธ์แบบโครงสร้าง ระบุว่าสิ่งหนึ่งเชื่อมโยงกับอีกสิ่งหนึ่ง

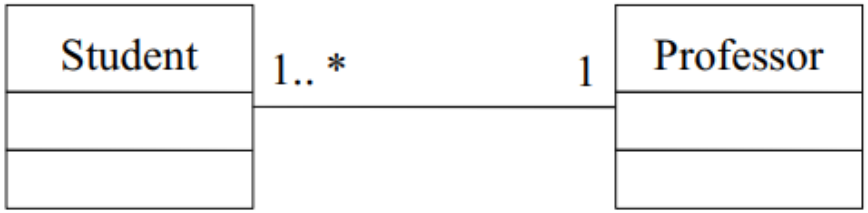
ตัวอย่าง นักศึกษาอยู่ในความดูแลของอาจารย์ที่ปรึกษา

ตัวอย่าง อาจารย์มีวิชาสอน

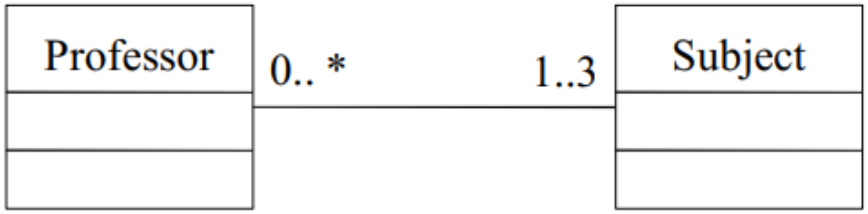
Multiplicity

- บ่งชี้จำนวนของ Object ที่มีความเกี่ยวพันกัน
- ในแต่ละสายความสัมพันธ์ Association พิจารณา 1 Object ของ Class หนึ่งสามารถมีความสัมพันธ์กับกี่ Object ของอีก Class หนึ่งที่อยู่ตรงข้าม

ตัวอย่าง นักศึกษามีอาจารย์ที่ปรึกษา 1 ท่านดูแล อาจารย์ 1 ท่านดูแลนักศึกษา 1 คนหรือหลายคน



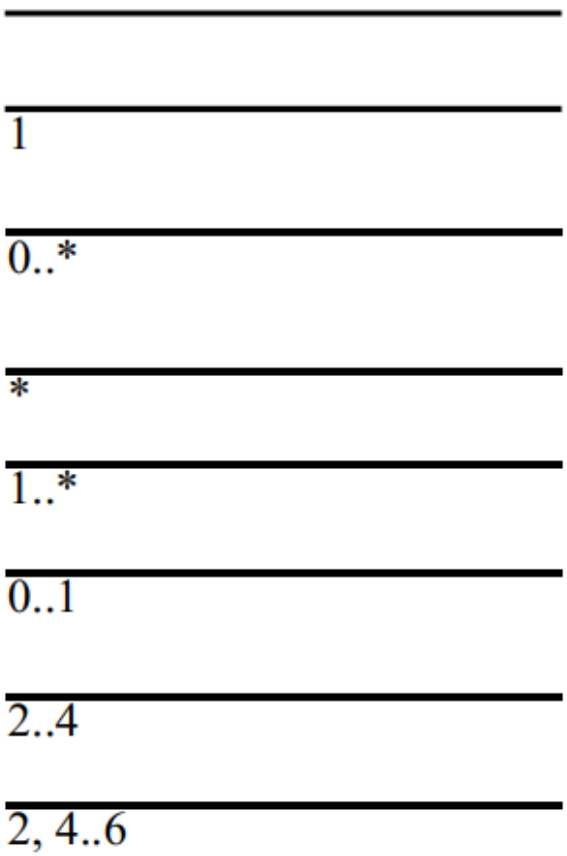
ตัวอย่าง อาจารย์ 1 ท่านมีวิชาสอน 1 ถึง 3 วิชาในแต่ละเทอม วิชาหนึ่งๆอาจไม่มีอาจารย์สอนหรืออาจมีอาจารย์สอนได้หลายท่าน



Multiplicity Indicators

- Unspecified
- Exactly one
- Zero or more (many,unlimited)

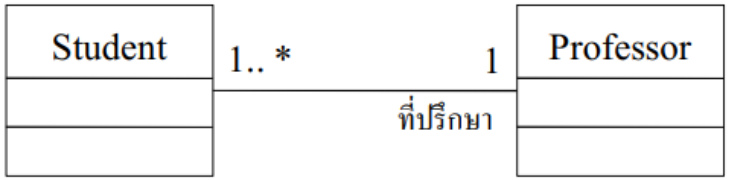
- One or more
- Zero or one (optional scalar role)
- Specified range
- Multiple, disjoint ranges



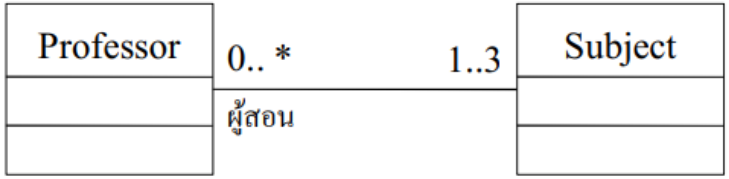
Role และ Association Name

- Role – อธิบายบทบาทของ Class ที่มีในสายสัมพันธ์ Association
- Association Name – ชื่อของ Association
- ทั้ง Role และ Association name จะช่วยให้เกิดความเข้าใจในความสัมพันธ์ได้ชัดเจนขึ้น

ตัวอย่าง อาจารย์ที่ปรึกษาให้คำปรึกษากับนักนักศึกษา



ตัวอย่าง อาจารย์รับผิดชอบสอนในวิชาต่างๆ

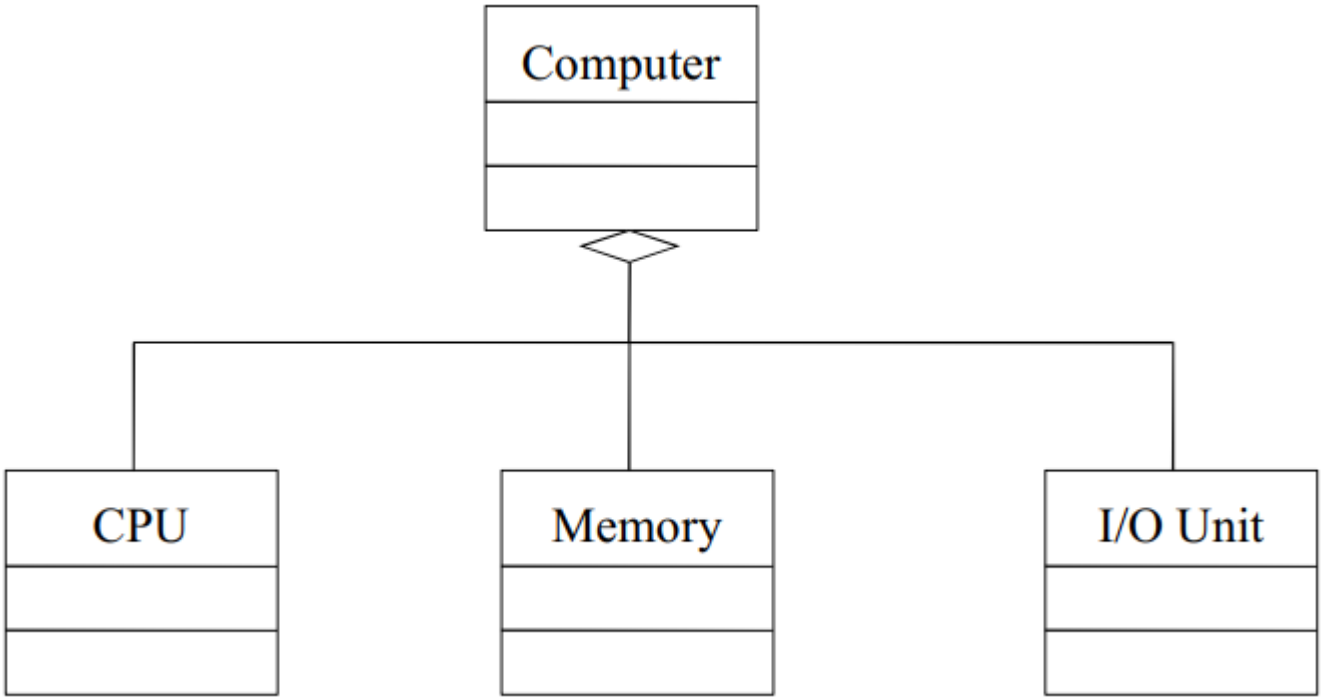


เป็นส่วนประกอบของ (Aggregation)

- เป็นชนิดหนึ่งของ association บ่งชี้ความเกี่ยวพันในลักษณะสิ่งหนึ่งเป็นส่วนประกอบของอีกสิ่งหนึ่ง (ชิ้นวัตถุใหญ่ (whole) กับส่วนประกอบ (part))
  - An aggregation “Is a part-of” relationship.
  - Multiplicity is represented like other associations.

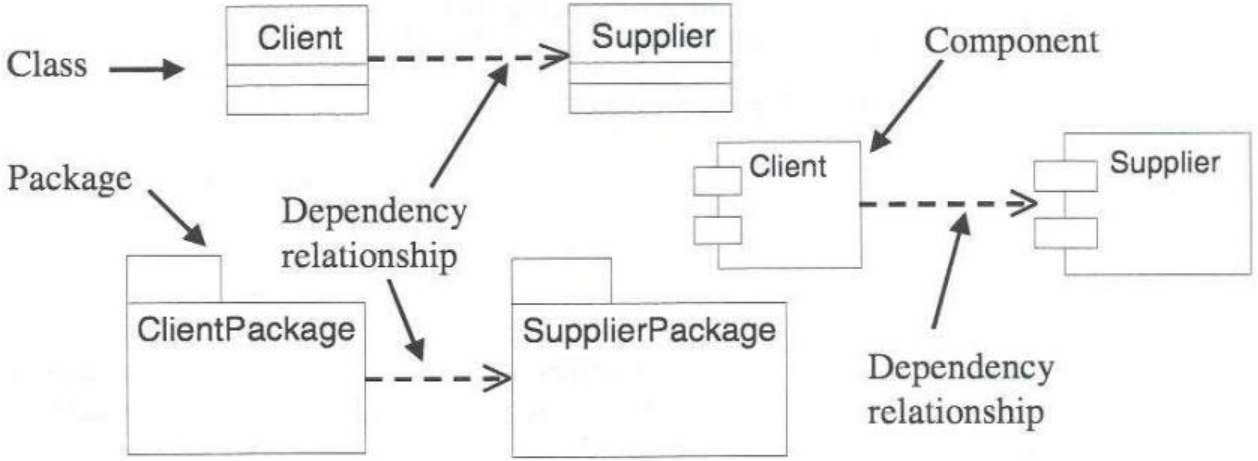
ตัวอย่าง Aggregation

- คอมพิวเตอร์ประกอบด้วยหน่วยประมวลผล (CPU) หน่วยความจำ (Memorv) และส่วนข้อมลเข้าออก (I/O Unit)



พึ่งพิง (Dependency)

- Object ของ Class หนึ่งอาศัยหรือใช้ทรัพยากรของ Object ในอีก Class หนึ่ง (การทำพฤติกรรมของ Object หนึ่งต้องใช้ Object ของอีก Class หนึ่ง)
- Non-structural, “using” relationship



Association and Dependency

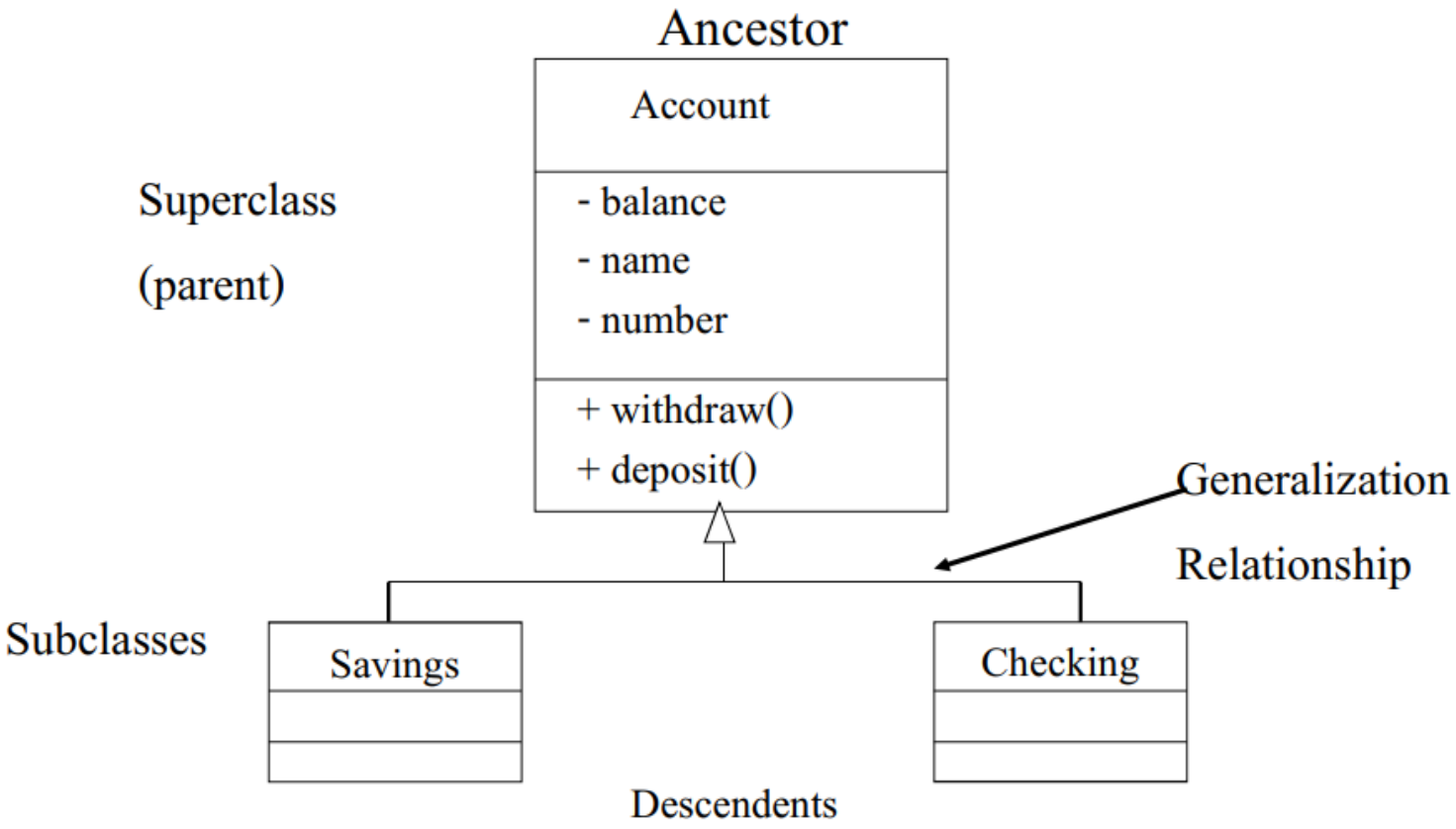
- Association – โครงสร้าง
  - เช่น คนมีแขน ขา
- Dependency – ชั่วคราว มีความเกี่ยวพันเมื่อมีการใช้งาน ใช้บริการ
  - เช่น คนใส่เสื้อผ้า เสื้อผ้าถูกใช้โดยคน
  - Dependency มักตัดสินใจในช่วงการออกแบบ

เป็นชนิดหนึ่งของ (Generalization)

- การสืบทอดคุณสมบัติและพฤติกรรมของบรรพบุรุษ (Super Class) มายังผู้สืบทอด (Subclass)
- ผู้สืบทอดเป็นชนิดหนึ่งของบรรพบุรุษ
- Defines a hierarchy of abstractions in which a subclass inherits form one or more superclasses
  - Single inheritance
  - Multiple inheritance
- • Is an “is a kind of” relationship

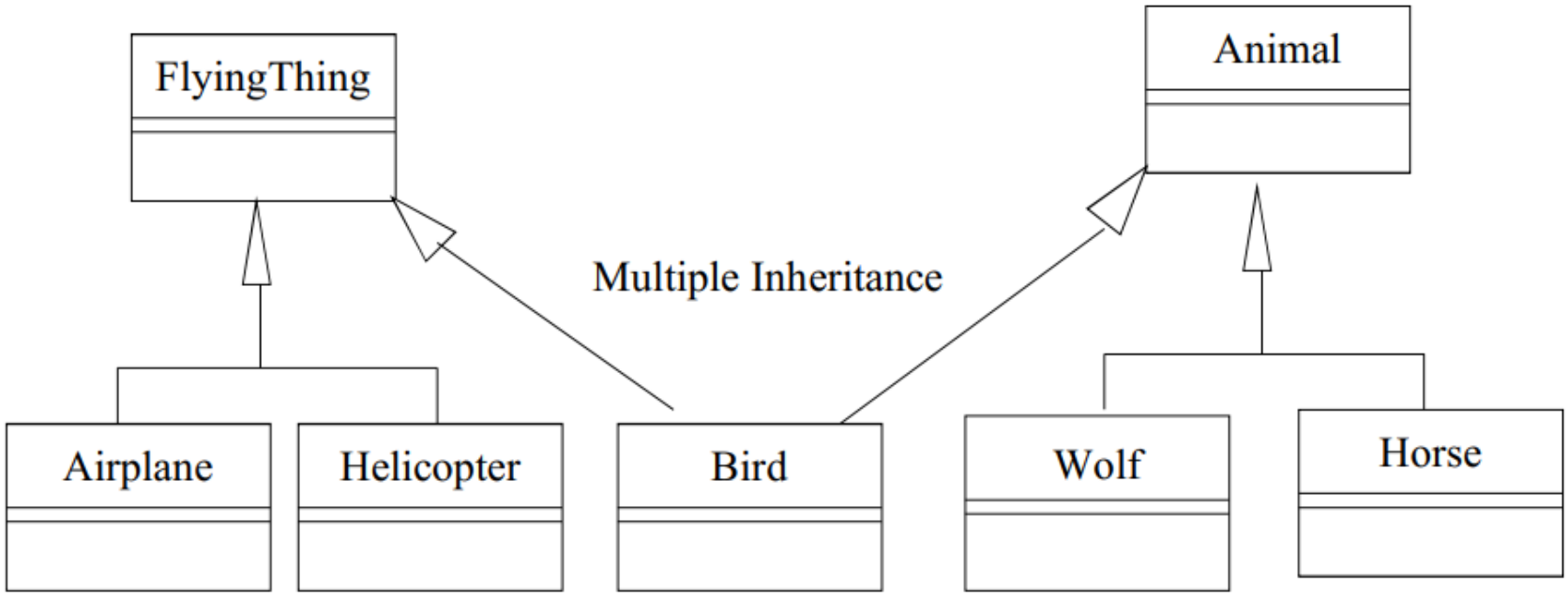
ตัวอย่าง : Single Inheritance

- One class inherits from another



ตัวอย่าง : Multiple Inheritance

- A class can inherit from several other classec.

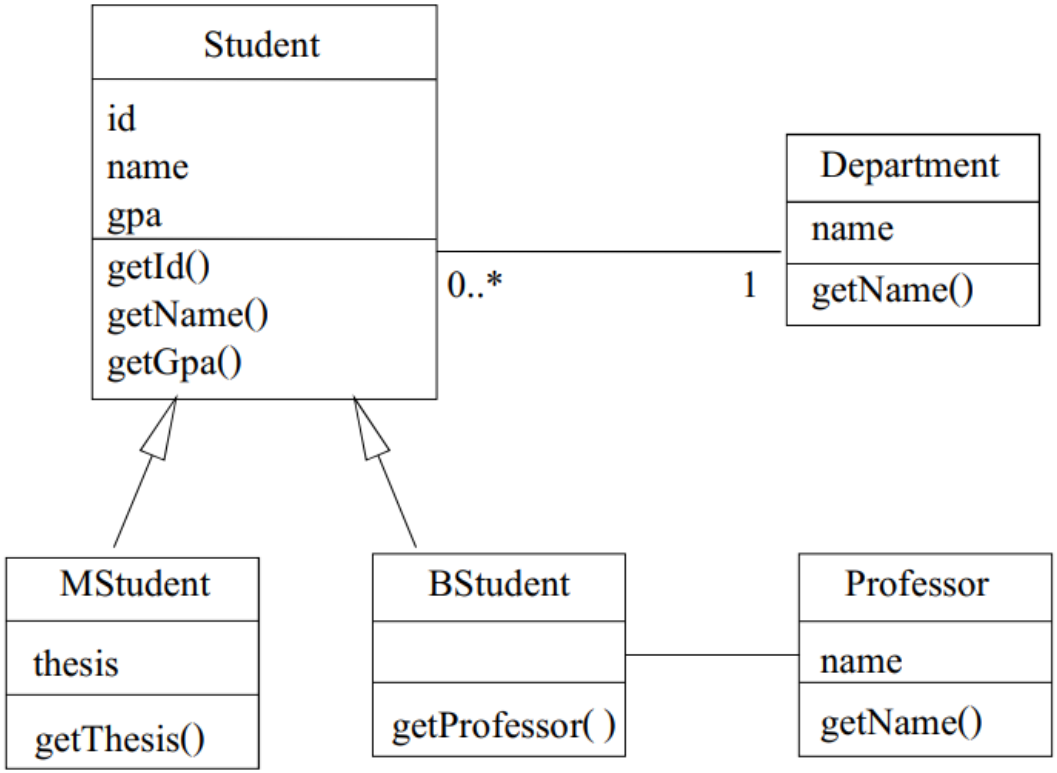


Use Multiple inheritance only when needed and always with caution!

สิ่งที่ถ่ายทอด

- A subclass inherits its parent's attributes, operations and relationships
- A subclass may:
  - Add additional attributes, operations, relationships
  - Redefine inherited operations (Use caution!)
  - Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy
- **Inheritance leverages the similarities among classes**

ตัวอย่าง : การถ่ายทอด



ถ้าให้สมชายเป็นนศ.ปริญญาตรี
ให้สมหญิงเป็นนศ.ปริญญาโท

วิธีอย่างง่ายในการออกแบบ

- เป็นวิธีการอย่างง่ายที่ช่วยในการออกแบบ class **วิธีการ:**
  - ค้นหา class โดยมองหาคำนามใน problem description คำนามที่มีลักษณะที่อธิบายได้จะเป็น candidate ของ class
  - หาคำกริยาที่ช่วยอธิบายหน้าที่ที่ต้องมีใน project ที่กำลังออกแบบ จากนั้นให้พิจารณาว่า class ใดควรจะต้องรับผิดชอบในการทำงานนั้น บันทึก คำกริยานั้นเพื่อกำหนดเป็น method หนึ่งของ class นั้น
  - สำหรับในแต่ละหน้าที่ที่ถูกบันทึกไว้ ให้พิจารณาว่ามี class ใดที่เกี่ยวข้องกับหน้าที่นั้น

ตัวอย่างปัญหา

- Problem Statement: ต้องการคำนวณและพิมพ์ใบเรียกเก็บเงิน

INVOICE

นายเล็ก ช่างเถอะ

100 ถนนพระราม 7

บางซื่อ กรุงเทพฯ 10800

Description	Price	Qty	Total
Toaster	29.95	3	89.85
Hair Dryer	24.95	1	24.95
Car Vacuum	19.99	2	39.98

Amount Due: 154.78

ขั้นตอน

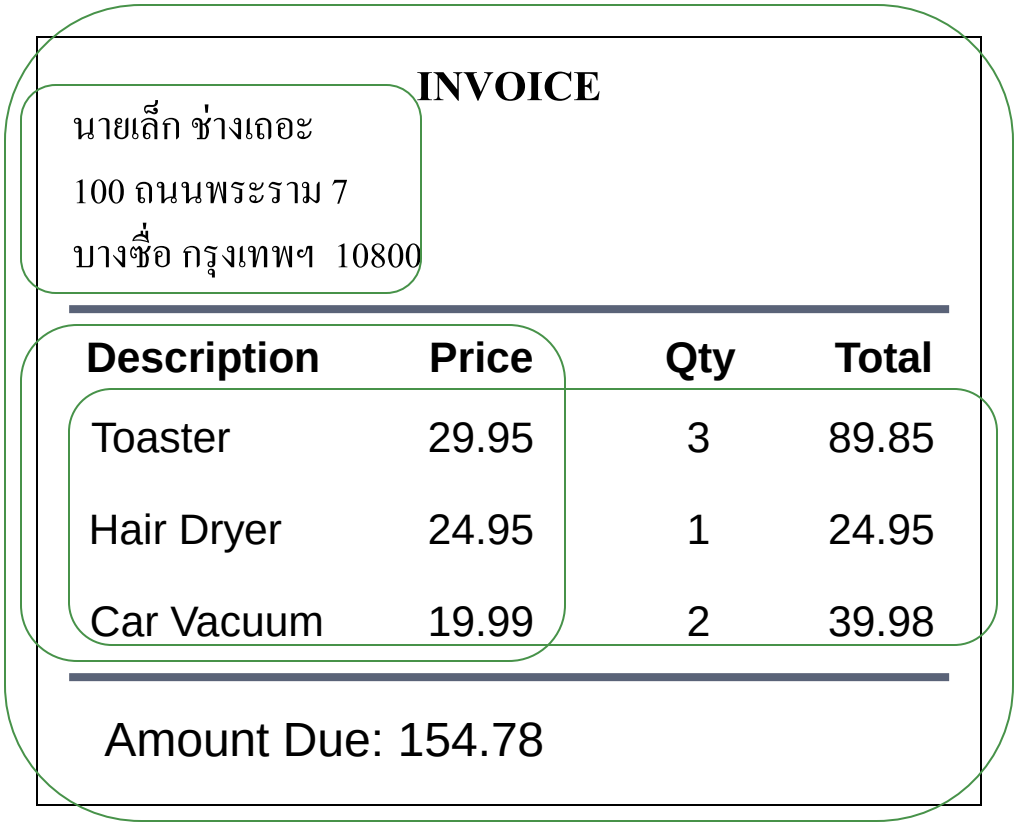
- **หาคำนาม**
  - Invoice
  - Address
  - LineItem
  - Product
  - Description
  - Quantity
  - Price
  - Amount Due
  - Total

ตัวอย่างปัญหา (อีกครั้ง)

Candidate:

- Invoice,
- Address,
- Product,
- LineItem

- Problem Statement: ต้องการคำนวณและพิมพ์ใบเรียกเก็บเงิน



INVOICE

นายเล็ก ช่างเถอะ
100 ถนนพระราม 7
บางซื่อ กรุงเทพฯ 10800

Description	Price	Qty	Total
Toaster	29.95	3	89.85
Hair Dryer	24.95	1	24.95
Car Vacuum	19.99	2	39.98

Amount Due: 154.78

ที่อยู่ผู้ซื้อ

สินค้า

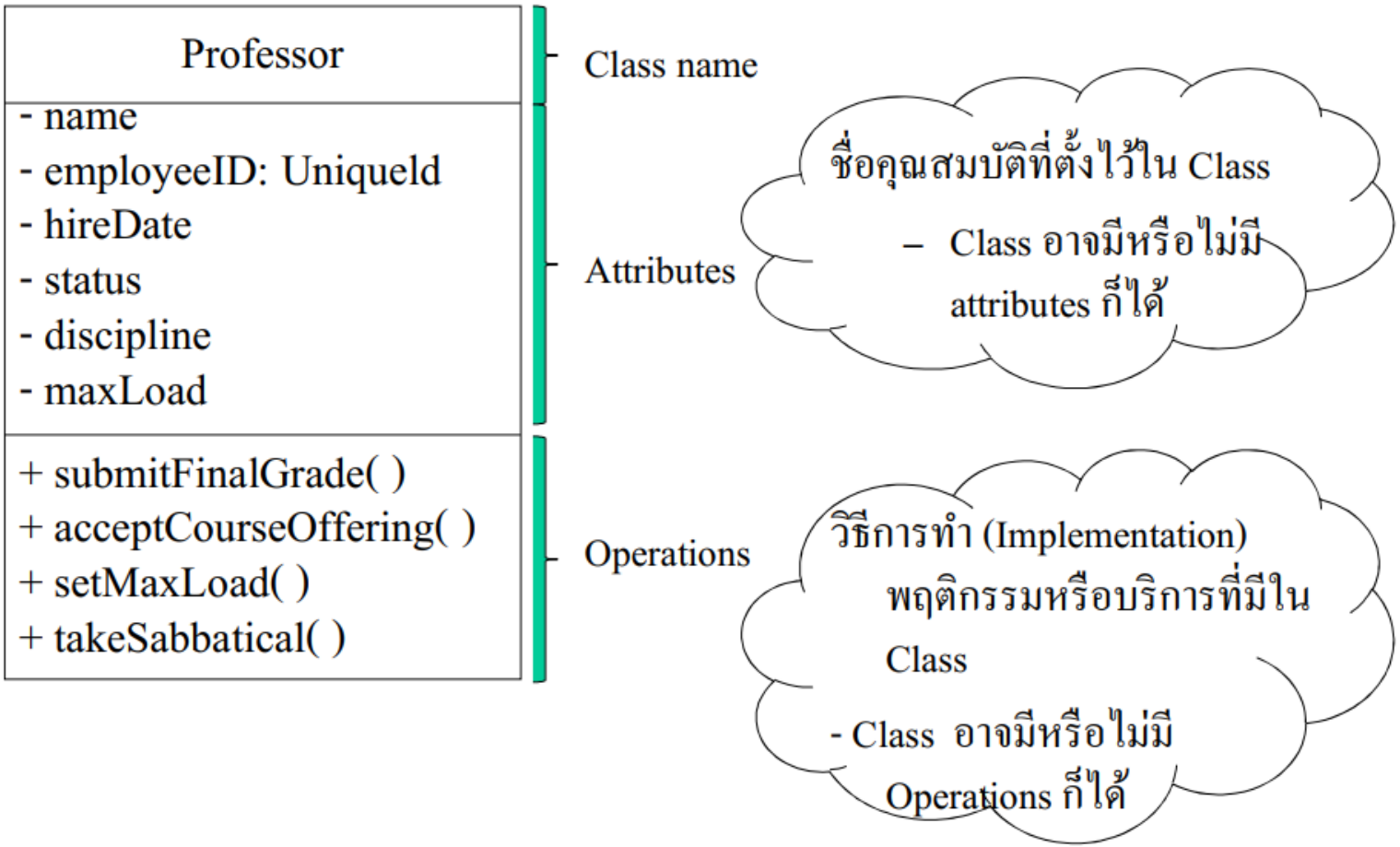
ใบเรียกเก็บเงิน

บรรทัดรายการ

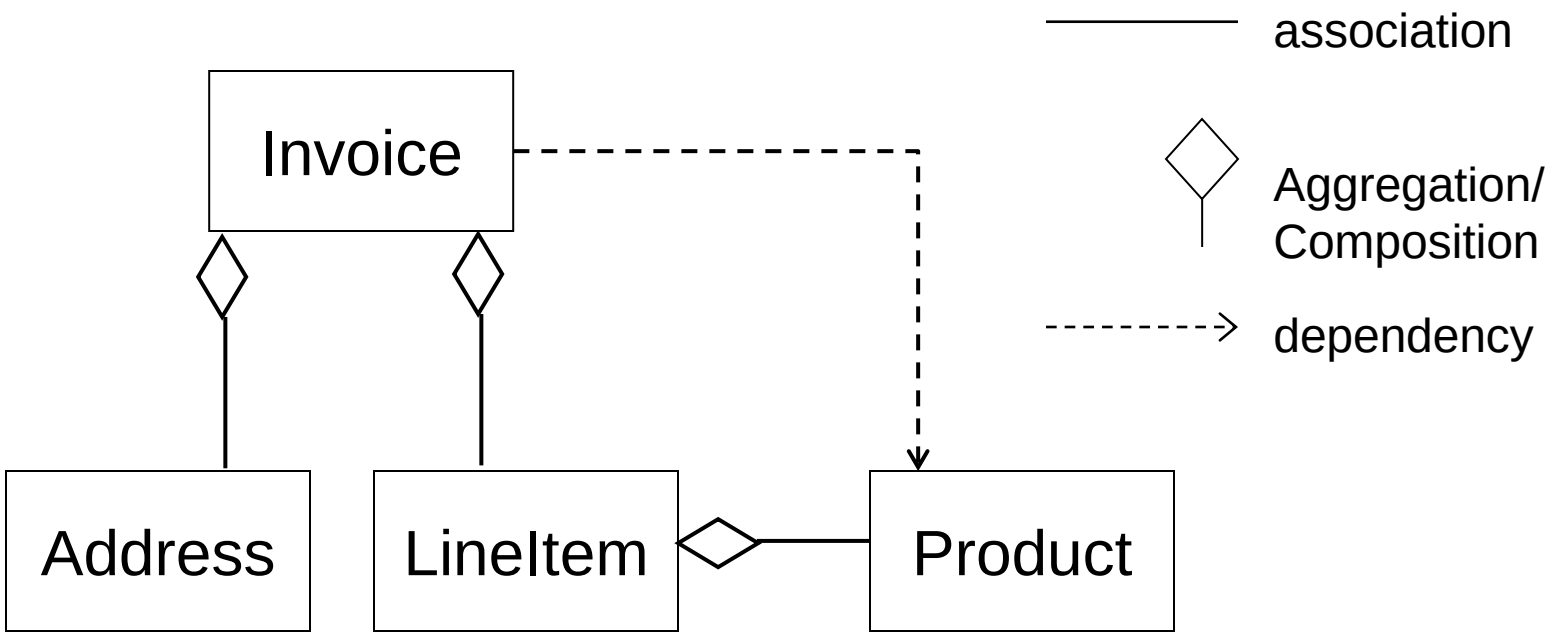
คลาส Invoice: หาหน้าที่และความสัมพันธ์

- **หาคำกริยา:**
  - พิมพ์รายงาน
  - เพิ่มบรรทัดรายการ (สินค้า และปริมาณ)
- **หาความสัมพันธ์กับคลาสอื่น:**
  - บรรทัดรายการ
  - ที่อยู่ผู้ซื้อ
  - สินค้า

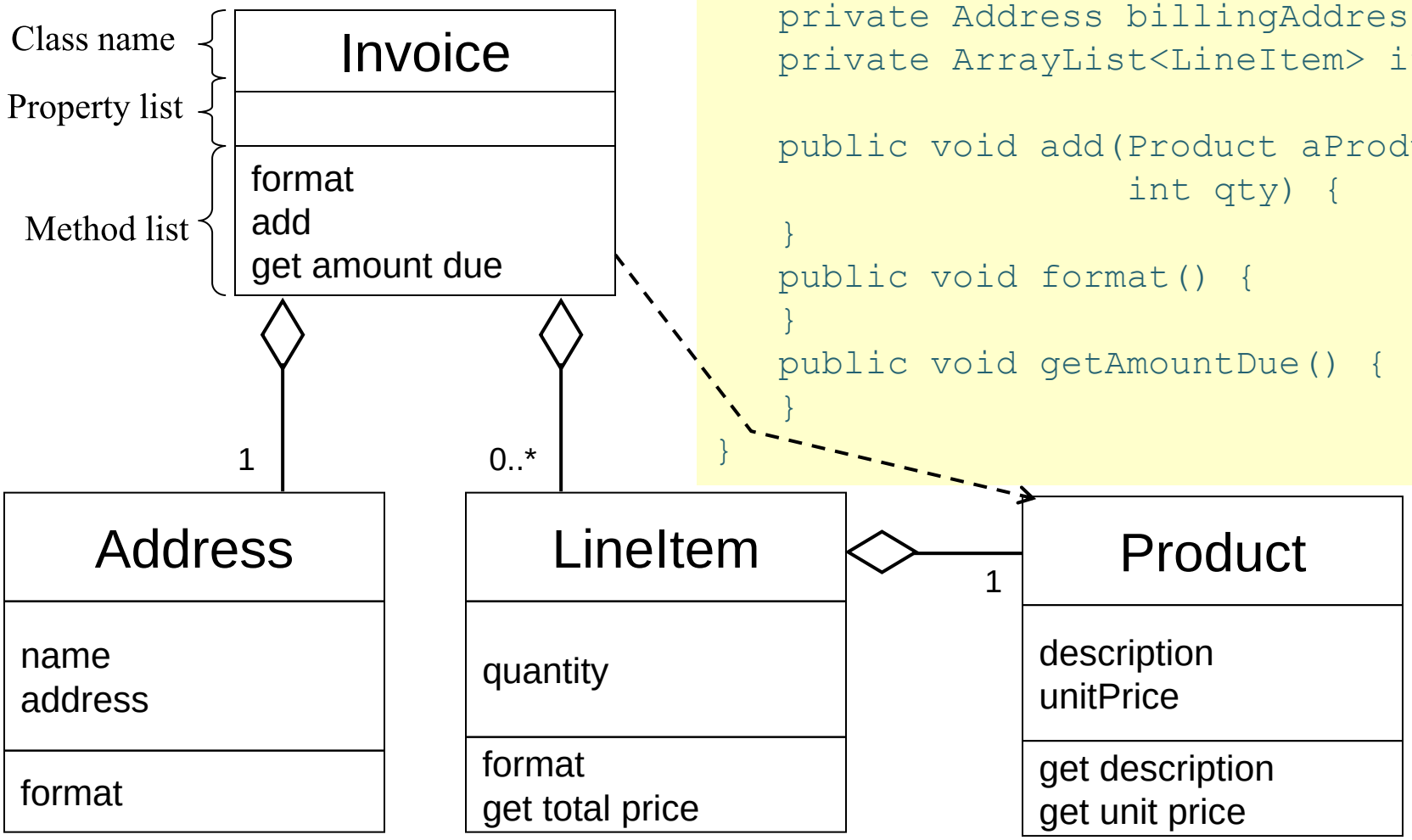
การแทน Classes ใน UML



Relationships



UML: Class Diagram (class, attributes, methods)




```
public class Invoice {
   private Address billingAddress;
   private ArrayList<LineItem> items;

   public void add(Product aProduct,
                   int qty) {
   }
   public void format() {
   }
   public void getAmountDue() {
   }
}
```

Program Coding

```
public class Invoice {
  private Address billingAddress;
  private ArrayList<LineItem> items;

  public void add(Product p,
                  int qty) { }
  public void format() { }
  public double getAmountDue() { }
}

public class Address {
  private String name;
  private String address;

  public void format() { }
}
```

```
public class LineItem {
  private Product product;
  private int quantity;

  public double getTotalPrice() {}
  public void format() { }
}

public class Product {
  private String description;
  private double price;

  public String getDescription() {
  }
  public double getprice() { }

}
```

Class ที่มีลักษณะที่ไม่ดี

- Class ที่ใช้ชื่อไม่สื่อความหมายหรือสื่อผิด
- Class ที่ทำหน้าที่หลายอย่างเกินไป ให้สร้าง Class อื่นมาช่วย
- Class ที่เปลี่ยนแปลง class อื่น
- Class ที่ไม่ทำหน้าที่อะไร แต่มีอยู่เพียงเพื่อให้มี
- Class ที่มี features ที่ไม่ได้ถูกใช้งาน
- Class ที่มีความรับผิดชอบที่ไม่เกี่ยวข้องกัน
- Class ที่มีการสืบทอดแบบผิด ๆ เช่น ไม่ได้มีความสัมพันธ์ในเชิง isA จริง หรือเป็น class ที่สืบทอดแบบไม่มีประโยชน์
- Class ที่มีการทำหน้าที่ซ้ำไปซ้ำมา

สรุปในบทนี้

- เข้าใจแนวคิดเกี่ยวกับการทำ Abstraction, ADT, Encapsulation
- เข้าใจแนวคิดเกี่ยวกับ Class และ วัตถุ
- เข้าใจการระบุหาวัตถุจากปัญหา และการออกแบบอย่างง่าย โดยใช้ Class Diagram
- เข้าใจขั้นตอนในการ implement classes และ methods อย่างง่าย ๆ ได้
- เข้าใจการใช้และเข้าถึง instance variables
- เข้าใจข้อแตกต่างระหว่างวัตถุและตัวอ้างถึงวัตถุ (Object Reference)