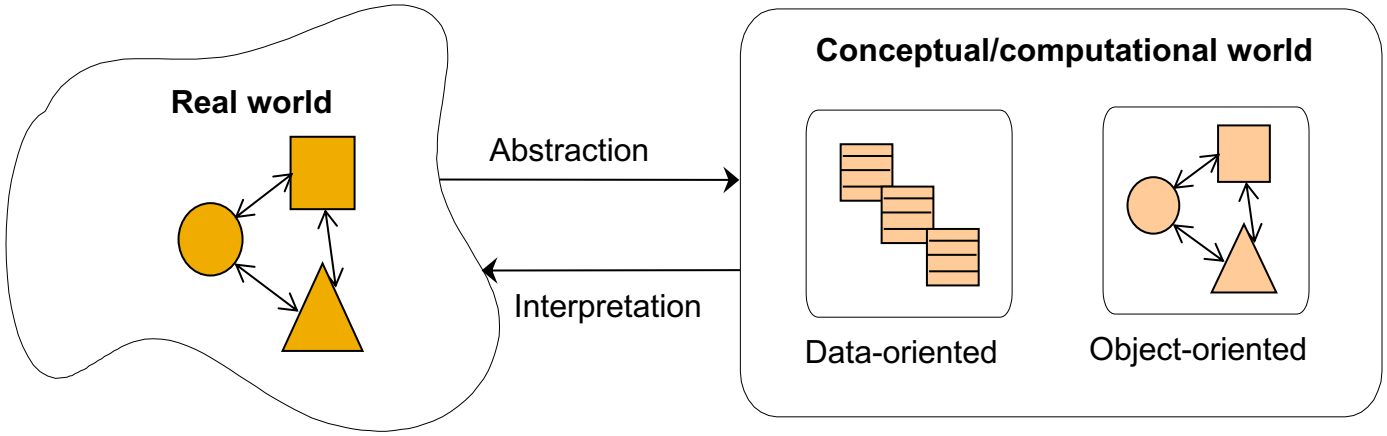
Java

Class Relationship

By Assoc. Prof. Rangsit Sirirangsi

The Need for Relationships

- ระบบเชิงวัตถุทุก ๆ ระบบประกอบไปด้วยออปเจคและคลาสต่าง ๆ
- การทำงานของระบบสามารถทำได้โดยการติดต่อกันระหว่างออปเจคภายในระบบผ่านความสัมพันธ์ระหว่างกัน
- เพื่อให้คลาสหนึ่งสามารถส่งแมสเสจไปยังคลาสอื่น ๆ จำเป็นจะต้องมีความสัมพันธ์ระหว่างคลาสทั้งสอง

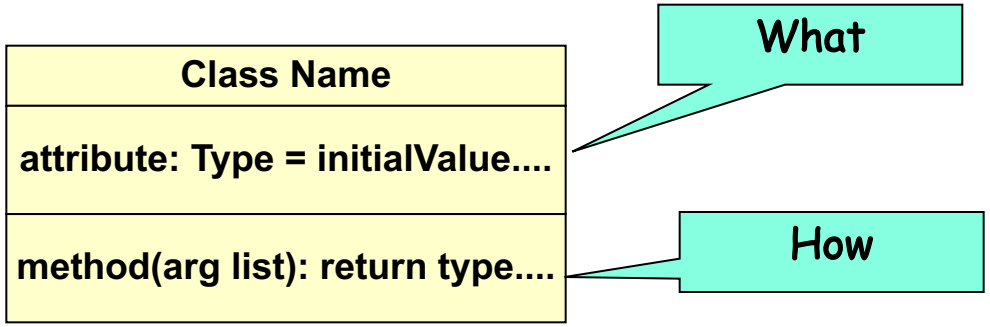


What is a Class Diagram?

- ใช้สำหรับแสดงรายละเอียดชนิดของออปเจคต่าง ๆ ที่อยู่ภายในระบบ ตลอดจนแสดงถึงความสัมพันธ์ชนิดต่าง ๆ ที่เกิดขึ้นระหว่างออปเจคเหล่านั้น
  - สัญลักษณ์รูปกราฟิกที่ใช้สำหรับการนำเสนอมุมมองแบบ static ของส่วนประกอบต่าง ๆ ที่ถูกประกาศไว้เป็นแบบ static
  - ถือเป็นหัวใจหลักของการวิเคราะห์และออกแบบระบบเชิงวัตถุ
  - ขั้นตอนของการวิเคราะห์ระบบหรือขั้นตอนแรก ๆ ของการออกแบบคลาสไดอาแกรม จะถูกนำเสนอโดยไม่มีการระบุรายละเอียดเชิงลึกแต่อย่างใด ทั้งนี้เนื่องจากเป็นการนำเสนอแอททริบิวต์และเมธอดหลัก ๆ เท่านั้น
  - รายละเอียดต่าง ๆ ของคลาสไดอาแกรมจะถูกเพิ่มเติมขึ้นในภายหลัง

Class

- คุณสมบัติของออปเจค 3 ประการ ได้แก่ : **state**, **behavior** และ **identity**
- คลาสมีลักษณะเป็นพิมพ์เขียวของออปเจคที่ประกอบไปด้วยส่วนที่เป็นแอททริบิวต์ (state) และเมธอด (behavior)

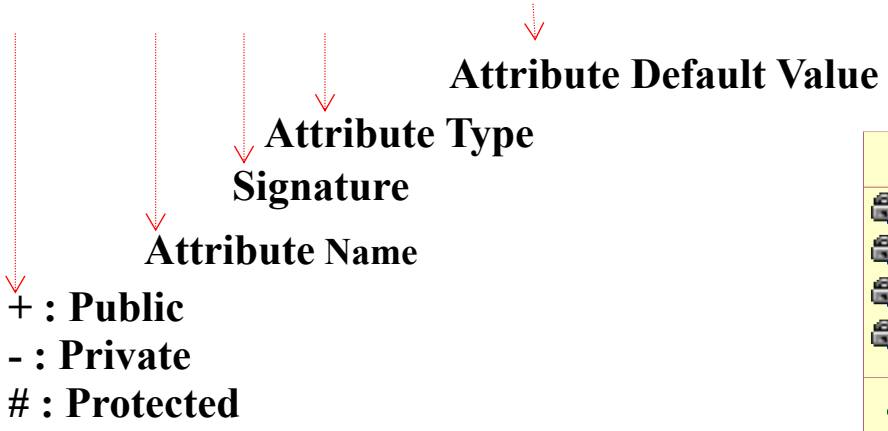


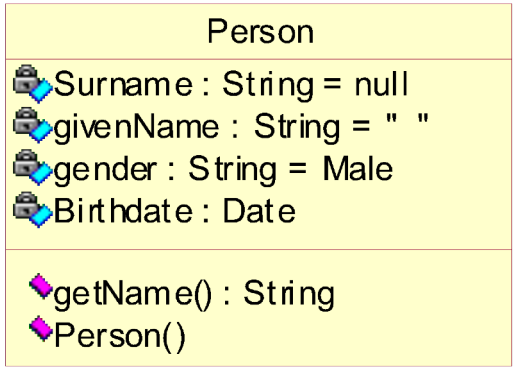
- รายละเอียดที่ปรากฏอยู่ภายในคลาสไดอาแกรมจะขึ้นอยู่กับระดับของการพัฒนาระบบ เช่น หากไม่มีการระบุส่วนของเมธอดไว้จะถือเป็นขั้นตอนของการวิเคราะห์ระบบ

Attributes

- เป็นข้อมูลที่ถูกจัดเก็บไว้โดยออปเจคที่ถูกสร้างขึ้นจากคลาส และถือเป็นคุณสมบัติของแต่ละออปเจค
- ชนิดของตัวแปรที่กำหนดให้กับแอททริบิวต์จะขึ้นอยู่กับภาษาโปรแกรมที่ใช้ โดยปกติจะเป็นชนิดข้อมูลแบบพื้นฐาน (Built-in type) และชนิดของตัวแปรที่ผู้ใช้กำหนดขึ้น (User-defined type)

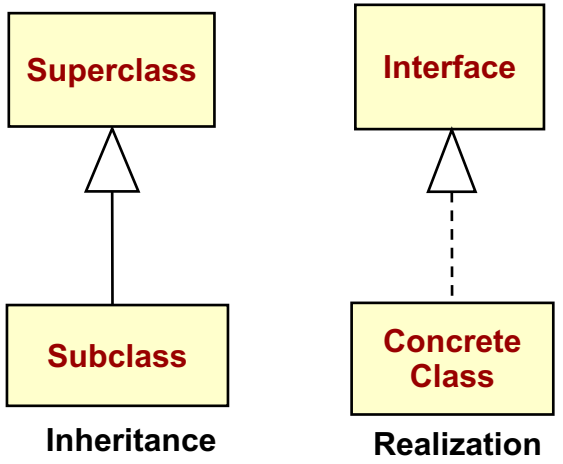
Visibility Name : Type = Default Value

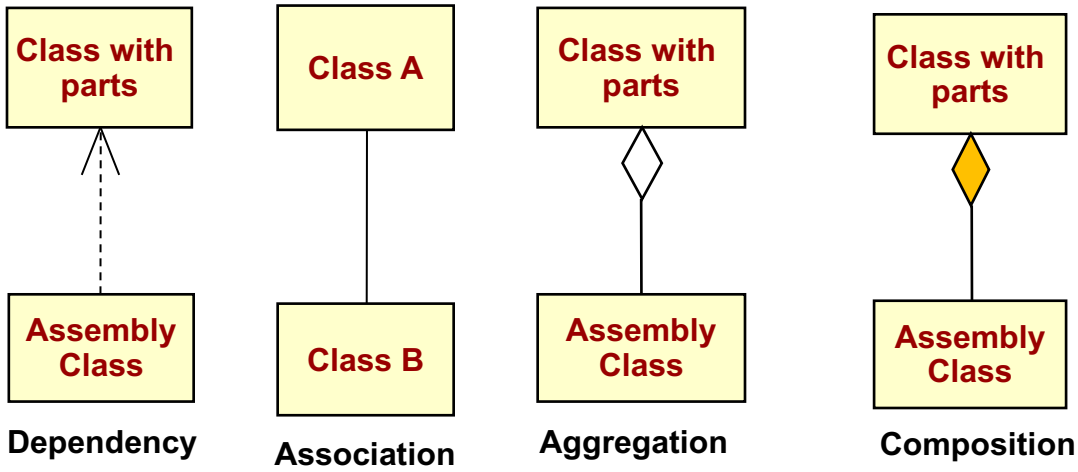




Relationship Types

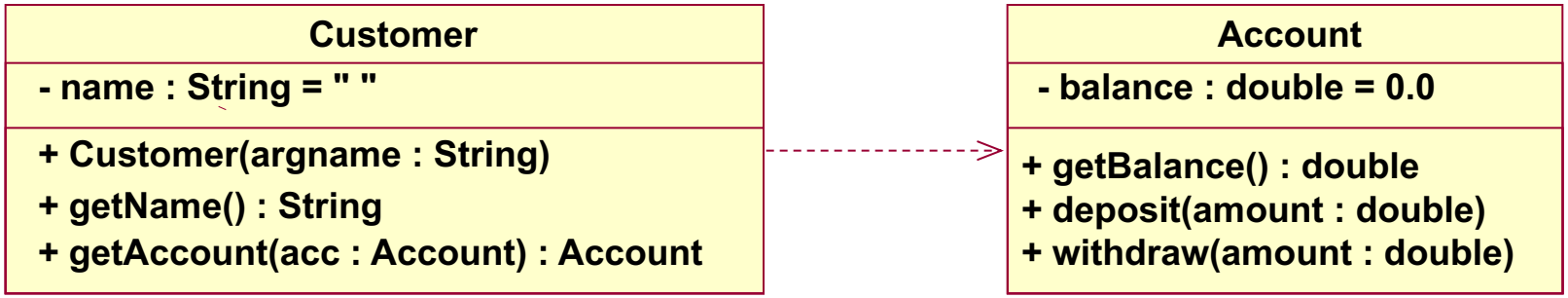
- ชนิดความสัมพันธ์ระหว่างคลาสใน UML มีดังนี้
  - Generalization และ Realization
  - Dependency
  - Association
    - Aggregation
    - Composition





Dependency

- ความสัมพันธ์ระหว่างคลาสที่คลาสหนึ่งมีการทำงานขึ้นอยู่กับอีกคลาสหนึ่ง
- นั่นคือหากมีการเปลี่ยนแปลงเกิดขึ้นภายในคลาสหนึ่งอาจจำเป็นต้องเปลี่ยนแปลงคลาสที่ dependency กันด้วย
- ความสัมพันธ์แบบ dependency จะถูกนำเสนอโดยใช้ลูกศรเส้นประระหว่างคลาสนั้น ๆ



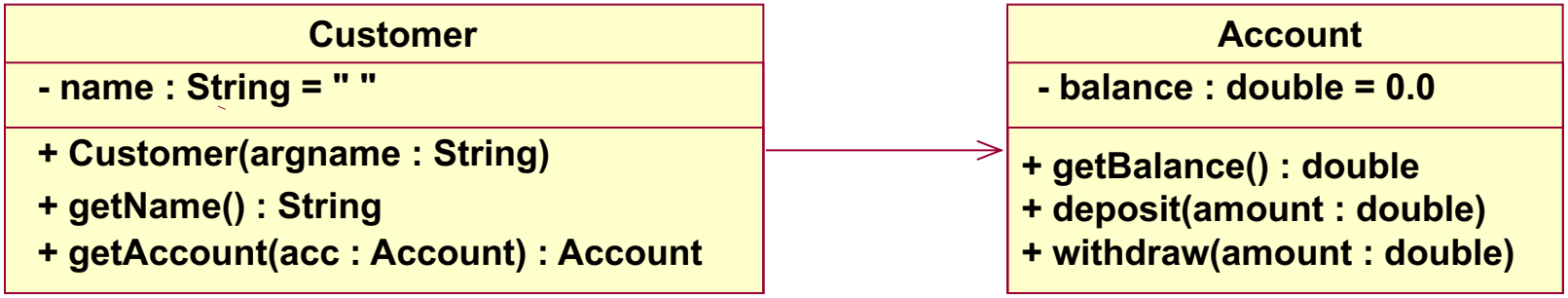
Account & Customer

```
class Account       {
   private double balance;
   public Account() {          balance = 0;   }
   public void deposit(double amount)  {   balance = balance + amount;     }
   public void withdraw(double amount) {   balance = balance - amount;     }
   public double getBalance() {   return balance;   }
}
                    class Customer{
                         private String fname;
                         public Customer(String name)     {          this.fname=name;       }
                         public String getName()          {          return fname;          }
                         public Account getAccount( Account acc)    {          return acc;     }
                    }

                    class CustAcc          {
                         public static void main(String[] args)     {
                          Account a1 = new Account();
                          a1.deposit(500);
                          Customer c1 = new Customer("Peak");
                          System.out.println("\nCustomer \"" + c1.getName() +
                           "\"" + " has :" + c1.getAccount(a1).getBalance());
                        }
                    }
```

Association Relationship

- เป็นความสัมพันธ์ที่มีลักษณะคล้ายกับความสัมพันธ์แบบ Dependency แต่จะแตกต่างกันตรงที่ช่วงเวลาที่เกิดความสัมพันธ์กันจะมีระยะเวลานาน และมีลักษณะเป็นความสัมพันธ์แบบโครงสร้าง
- ความสัมพันธ์แบบนี้อาจถูกมองในรูปของการมีอยู่ (*has-a)* ได้ เช่น ออปเจคจากคลาสหนึ่งมีค่าอ้างอิงไปยังออปเจคจากคลาสอื่นได้
- ใช้เส้นทึบในการนำเสนอความสัมพันธ์แบบนี้



การออกแบบ Account


```
class Account  {
 private double balance;
 public Account() {    balance = 0;   }
 public Account(double initialBalance)  {
       balance = initialBalance;
 }
 public void deposit(double amount)  {
     balance += amount;
 }
 public void withdraw(double amount )  {
    balance = balance – amount;
 }
 public double getBalance()  {    return balance;  }
 public String toString()  {
       return Double.toString(balance);
 }
}
```

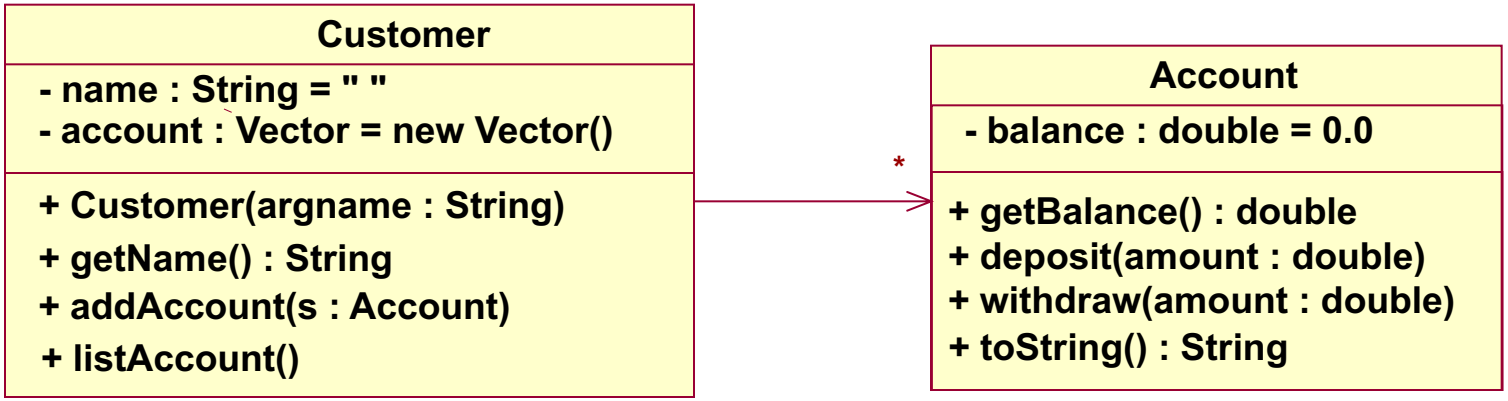
การออกแบบ Customer

```
class Customer {
 private Account acc;
 private String name;
 public Customer(String aName)  {
     name = aName;
    acc = new Account(0);
 }
 public String getName() {    return name;    }
 public Account getAccount()  {   return acc;   }
  public void addToAccount(double amt)  {
   acc.deposit(amt);
 }
 public String toString() {
   return name + acc;
 }
}
```

```
public class Run {
  public static void main(String args[]) {
      Customer big = new Customer("Tom");
       big.addToAccount(1000);
       big.getAccount().withdraw(100);
        System.out.println(big);
   }
}
```

Association Customer & Account

- ความแตกต่างระหว่างคลาส Account และ Customer ในที่นี้จะได้แก่ ความสัมพันธ์ที่มีลักษณะเป็นแบบ Relationship Cardinality จากตัวอย่างจะเป็นความสัมพันธ์แบบ One to Many



Customer : Account (1 : Many)


```
class Customer {
      private String fname;
      private List<Account> account = new Vector<>();
      public Customer(String name)     {  fname=name; }
      public String getName()              { return fname;  }
      public void addAccount(Account s)      {   account.addElement(s); }
      public void listAccount()      {
        for (int i = 0; i < account.size(); i++)  {
              System.out.println(" "+ account.elementAt(i));
        }
      }
}
```

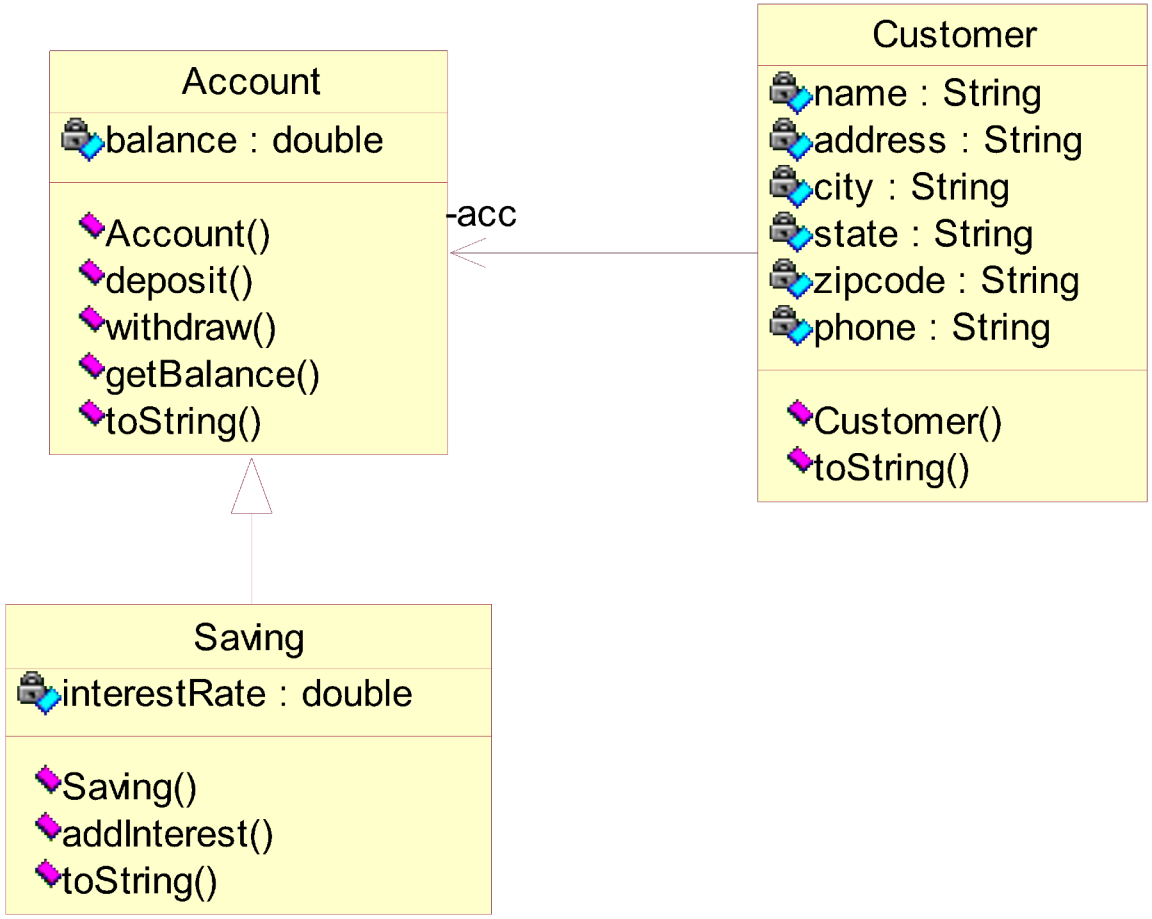
```
class Account  {
  private double balance;
  public Account(double initialBalance)  {    balance = initialBalance;  }
  public void deposit(double amount) {        balance += amount; }
  public void withdraw(double amount ) {  balance = balance - amount; }
  public double getBalance() {     return balance;    }
}
```

Customer : Account (1 : Many)


```
class CustAcc    {
      public static void main(String[] args)       {
       Account a1 = new Account(101);
        a1.deposit(500);
        Account a2 = new Account(102);
        a2.deposit(1000);
        Customer c1 = new Customer("Dang“);
        c1.addAccount(a1);
        c1.addAccount(a2);
        System.out.println("\nCustomer \"" + c1.getName() + "\"" + " has accounts :“);
        c1.listAccount();

      }
}
```

Association Customer & Account



คลาส Account


```
class Account  {
  private double balance;
  public Account() {     balance = 0;   }
  public Account(double initialBalance) {
        balance = initialBalance;
  }
  public void deposit(double amount) {         balance += amount; }
  public void withdraw(double amount ) {
        balance = balance - amount;
  }
  public double getBalance() {      return balance;    }

   public String toString(){
        return "from " +this.getClass()+" with balance = "+ this.getBalance();
   }
}
```

คลาส Saving


```
public class Saving extends Account
{
      private double interestRate;
      public Saving ( double rate)  {
          interestRate = rate;
      }
     public void addInterest()        {
        double interest = getBalance()* interestRate / 100;
         deposit( interest);
      }
     public String toString(){
        return super.toString()+ " and Interest Rate = " + this.interestRate;
    }
}
```

คลาส Customer


```
class Customer {
      private String name;
      private String address;
      private String city;
      private String state;
      private String zipcode;
      private String phone;
      private Account acc;
      public Customer( String name, String address, String city, String zipcode, Account
      other) {
        this.name = name;
        this.address = address;
        this.city = city;
        this.zipcode = zipcode;
        acc = other;
      }
      public String toString(){
        return " " + name+ " "+address+ " " +city+" "+zipcode+" has account "+this.acc;
      }
}
```

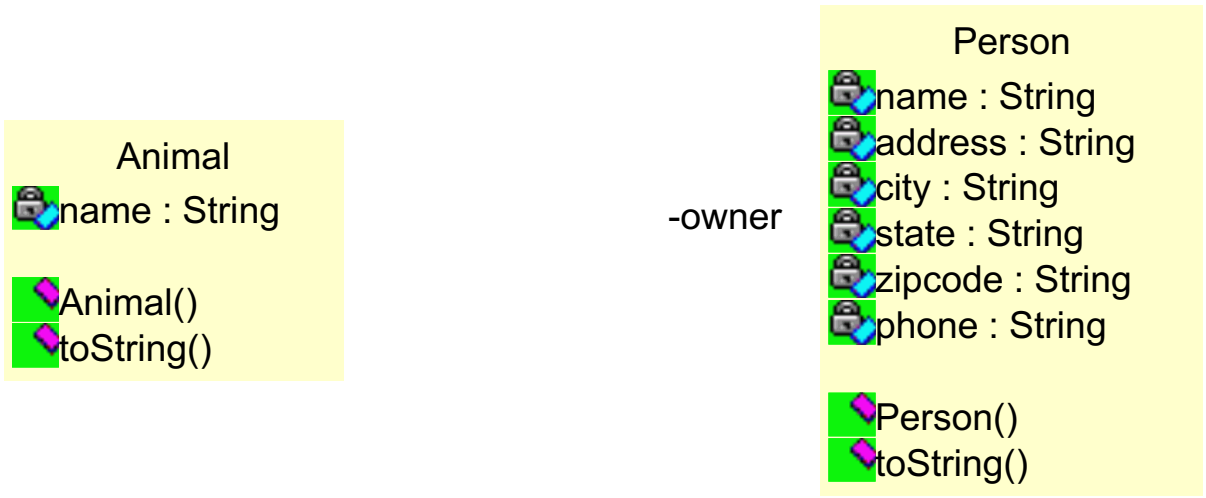
คลาส Run

```
class Run
{
      public static void main(String[] args)
       Account ac = new Account();
       Saving sc = new Saving(1.5);
       Customer p = new Customer("Fatty","100 Praw St.","Maejo","50290",ac);
       Customer q = new Customer("Peak","500 Chotana st.", "Maerim","50180",sc);
       System.out.println(p);
       System.out.println(q);
      }
}
```

Fatty 100 Praw St. Maejo 50290 has account from class Account with balance = 500.0

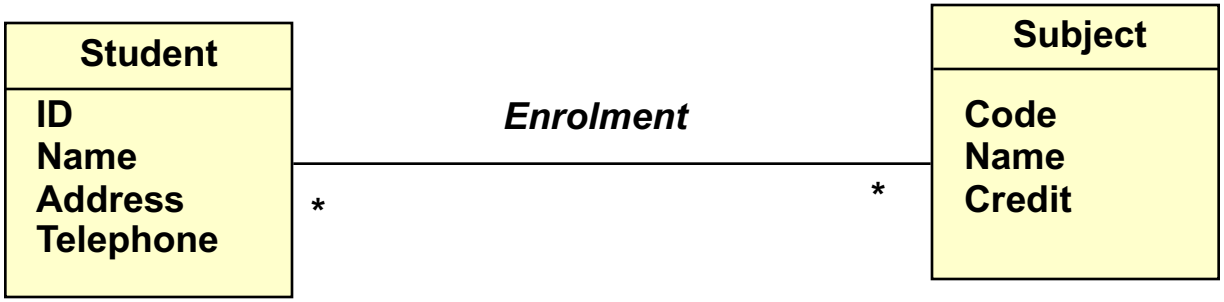
Peak 500 Chotana st. Maerim 50180 has account from class Saving with balance = 500.0 and Interest Rate = 1.5

Person with Super Class Animal



Association Class

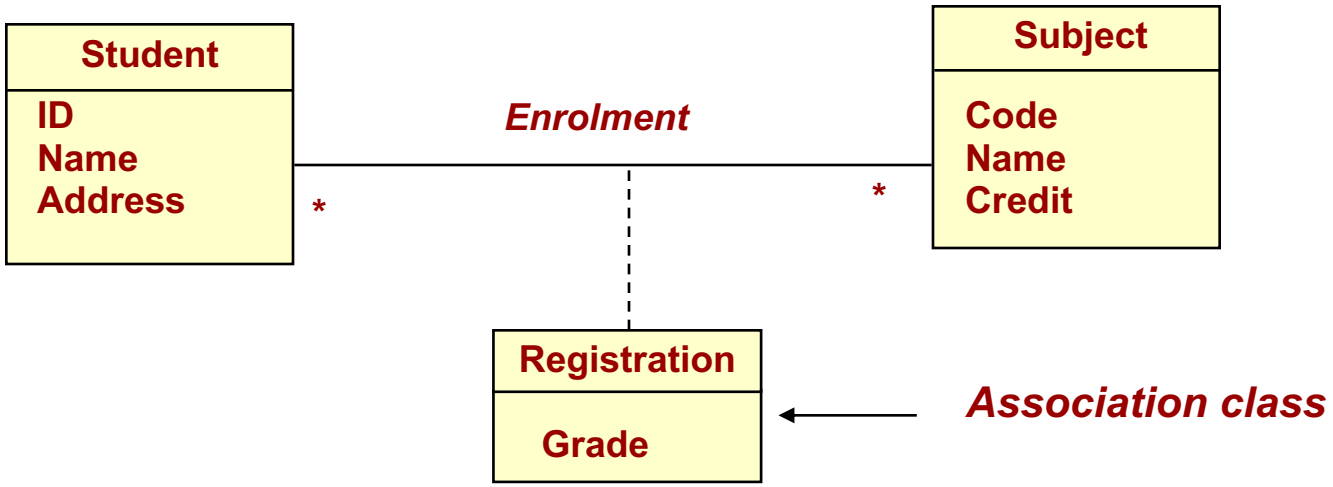
- บ่อยครั้งที่การออกแบบคลาสอาจมีจำนวนความสัมพันธ์เป็นแบบ many-to-many ดังนั้นการกำหนดความสัมพันธ์ระหว่างคลาสอาจมีความจำเป็นในการแก้ไขให้อยู่ในรูปของความสัมพันธ์แบบ one-to-many
- ตัวอย่างเช่น: Student แต่ละคนสามารถลงทะเบียนได้หลาย Subject และแต่ละ Subject ประกอบไปด้วย Student หลายคน



- จะเกิดอะไรขึ้นหากมีการกำหนดผลการเรียน (grade) ให้กับ Student ที่ลงทะเบียนเรียนใน Subject ที่กำหนดไว้

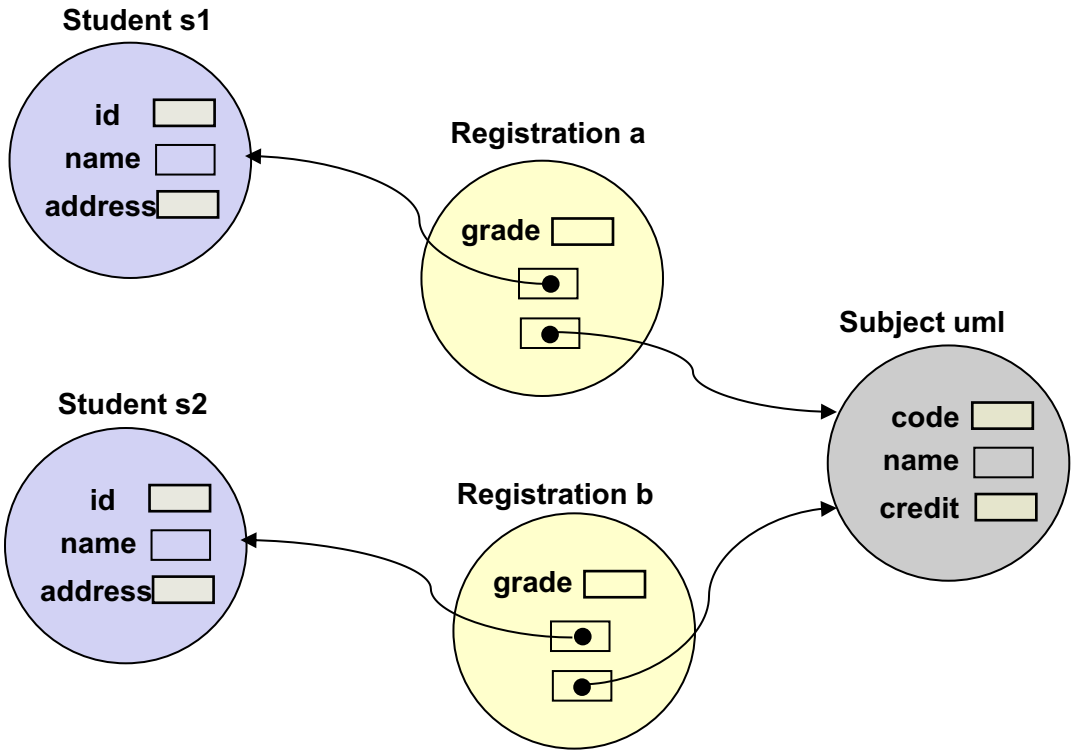
Association Classes

- การทำงานของ Association คลาส สามารถทำได้โดยการกำหนดคลาสที่เรียกว่าคลาส *Registration* ซึ่งประกอบไปด้วยแอททริบิวต์ในรูปของออปเจคจากคลาส Student และ Subject



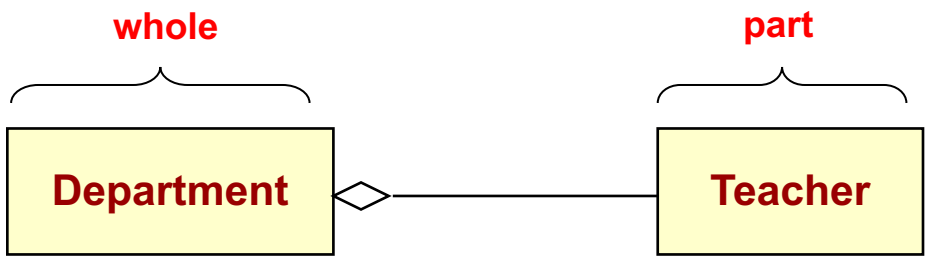
Association Classes : Object View

- มุมมองในการจำลองภาพการทำงานของ Association คลาส

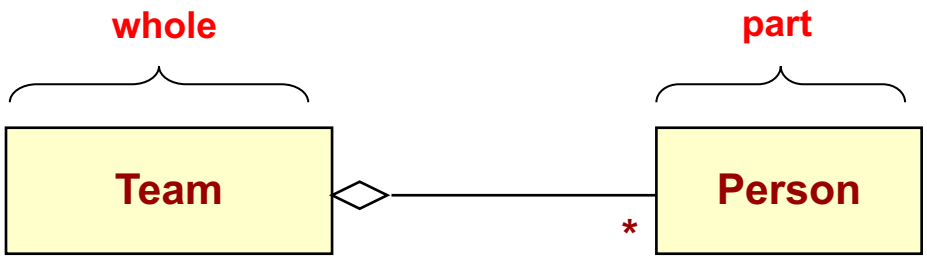


What is Aggregation?

- ความสัมพันธ์แบบ Aggregation บางครั้งจะถูกเรียกว่า Shared Aggregation
- เป็นความสัมพันธ์ในลักษณะที่เรียกว่า “whole-part”
- ส่วนที่เป็น part อาจเป็นส่วนประกอบของส่วนที่เป็น whole อื่น ๆ ได้
- สัญลักษณ์ที่ใช้ใน UML จะมีลักษณะคล้ายกับความสัมพันธ์แบบ association แต่จะเพิ่มสัญลักษณ์ diamond ไว้ในส่วนที่เป็น whole เพื่อระบุว่าเป็นความสัมพันธ์แบบ Aggregation



What is Aggregation?

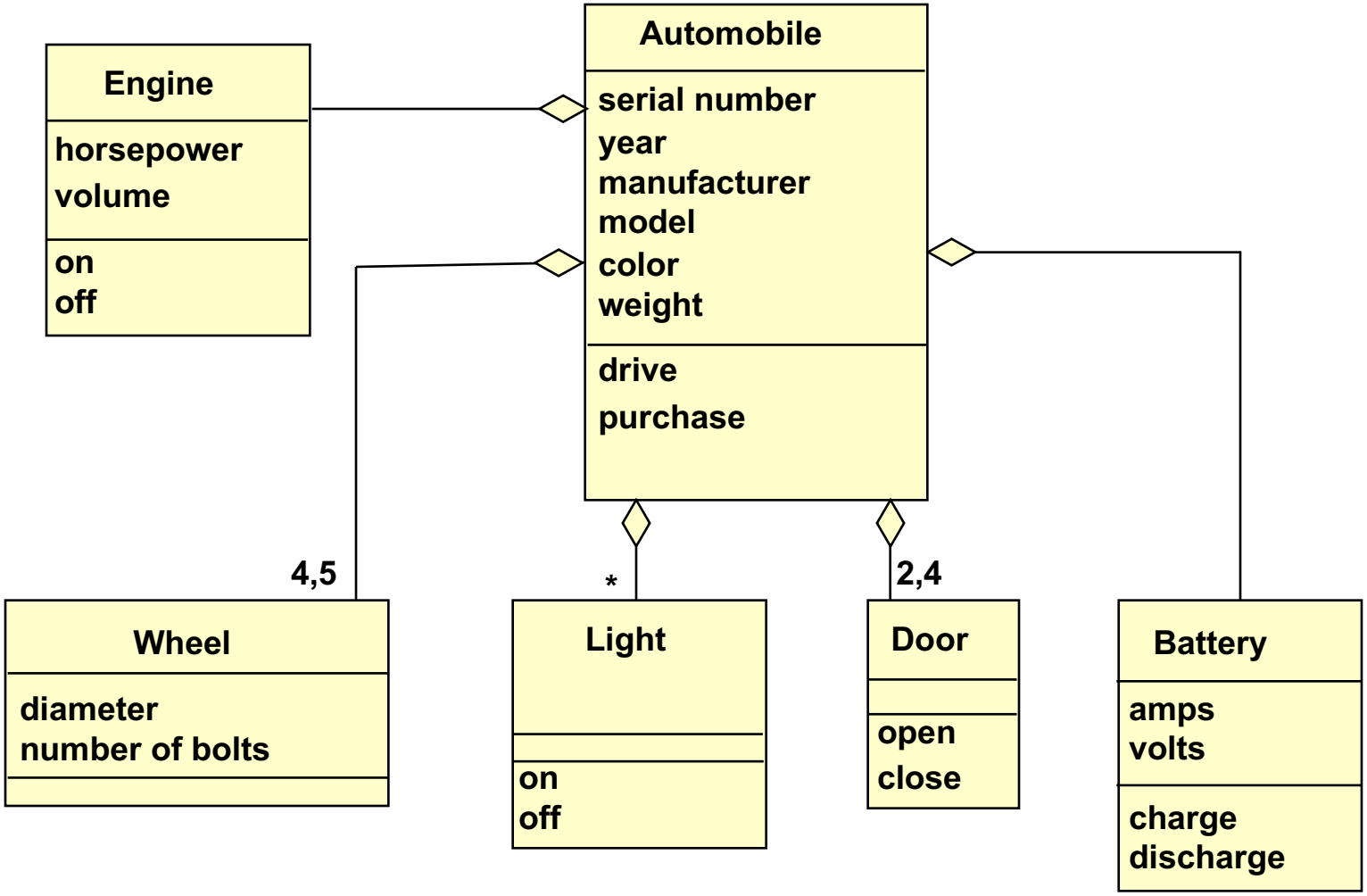


- Team ทำหน้าที่เป็น whole ส่วน Person ทำหน้าที่เป็น part
- ความสัมพันธ์แบบนี้ช่วงชีวิตของ part จะไม่ขึ้นอยู่กับส่วนที่เป็น whole
- ในกรณีที่ Team ถูกลบออกจากระบบ Person ออปเจคจะไม่ถูกลบตามไปด้วย
  - Team ประกอบไปด้วย Person ที่เป็นสมาชิกภายในทีม
  - Person หนึ่งสามารถเป็นสมาชิกของ Team อื่น ๆ ได้
  - Person ถือเป็นส่วนของ part ที่มีการใช้งานร่วมกัน

What is Whole/Parts

- การจำแนกความสัมพันธ์แบบ whole/part
- เป็นความสัมพันธ์แบบ transitive
  - ถ้าวัตถุ A เป็นส่วนหนึ่งของ B และ B เป็นส่วนหนึ่งของ C แล้ว A เป็นส่วนหนึ่งของ C ด้วย
  - เช่น ถ้าที่เปิดประตูเป็นส่วนหนึ่งของประตู ประตูเป็นส่วนหนึ่งของรถยนต์แล้ว ที่เปิดประตูเป็นส่วนหนึ่งของรถยนต์
- เป็นความสัมพันธ์แบบ Anti-symmetric
  - วัตถุใดๆ อาจไม่จำเป็นต้องเป็นส่วนประกอบของตัวเองทั้งทางตรงและทางอ้อม
  - เช่น ถ้าประตูเป็นส่วนหนึ่งของรถยนต์ แต่รถยนต์ไม่จัดเป็นส่วนหนึ่งของประตู

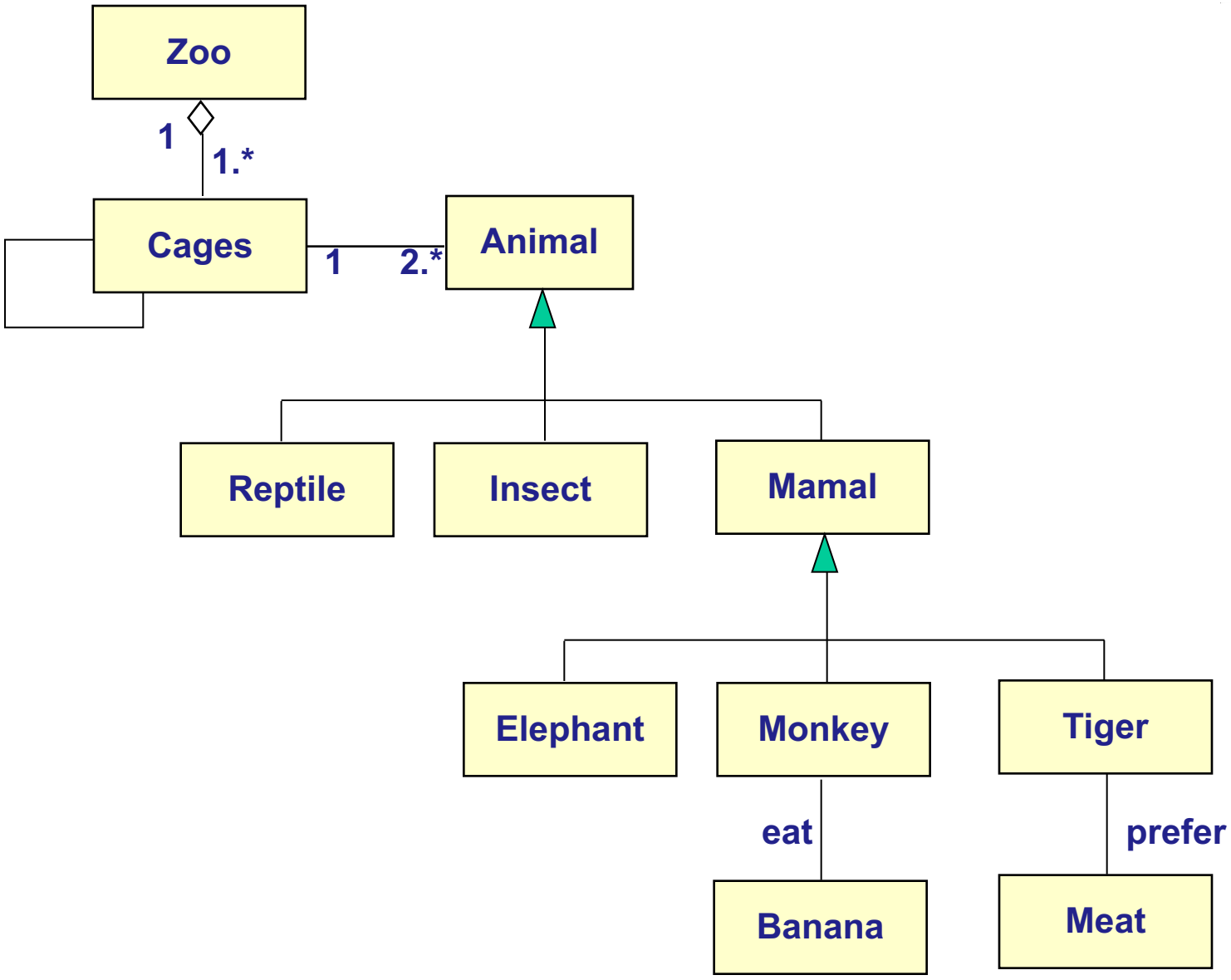
Aggregation



Class Diagram - Example

- A zoo consists of a set of cages.
- Every cage is the home of at least 2 animals.
- Cages are located besides each other.
- Every cage has at most one left neighbor and at most one right neighbor.
- Animals can be reptiles, insects, and mammals.
- Mammals are elephants, monkeys, and tigers.
- Monkeys eat bananas.
- Tigers prefer meat.

Zoo Class Diagram



How to find Class:

- เรียนรู้เกี่ยวกับ problem domain โดยอาศัยการสังเกตจาก client
- ประยุกต์ใช้กับสภาพความเป็นจริงในปัจจุบัน
- ใช้ลำดับเหตุการณ์ที่ปรากฏอยู่ในรายละเอียดของยูสเคส เพื่อช่วยในการค้นหาออปเจคที่เกี่ยวข้อง
- ใช้วิธีการวิเคราะห์คำนามจาก scenario หรือลำดับของเหตุการณ์ (Abbott Textual Analysis, 1983)
  - แสดงรายการของคำนามทั้งหมดในรูปของคลาสคู่แข่ง
  - ตัดคำนามที่มีความหมายซ้ำกันออกไป
  - กำหนดคำนามที่เป็นแอททริบิวต์หรือคำนามที่สามารถเป็นคลาส

Library System : Requirement

- **Books and Journals**
  - The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.
- **Borrowing**
  - The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

The Candidate Classes

- ขั้นตอนแรกของการสร้างคลาสไดอาแกรมจะได้แก่ การกำหนดคลาสคู่แข่ง (Candidate Classes) โดยปกติจะประกอบไปด้วยคำนามทุก ๆ คำที่ปรากฏในเอกสารประกอบการกำหนดความต้องการของระบบ
- การกำหนดคลาสคู่แข่งทำได้โดยการขีดเส้นใต้ noun และ noun phrases ทั้งหมดที่มีอยู่ภายในเอกสารประกอบความต้องการของระบบดังนี้:

  - **Library , books , journals , copies of a given book , short term loans , library member , weeks , Members of the library , items , time , members of staff , System and Rules**

Discard the Candidate Class

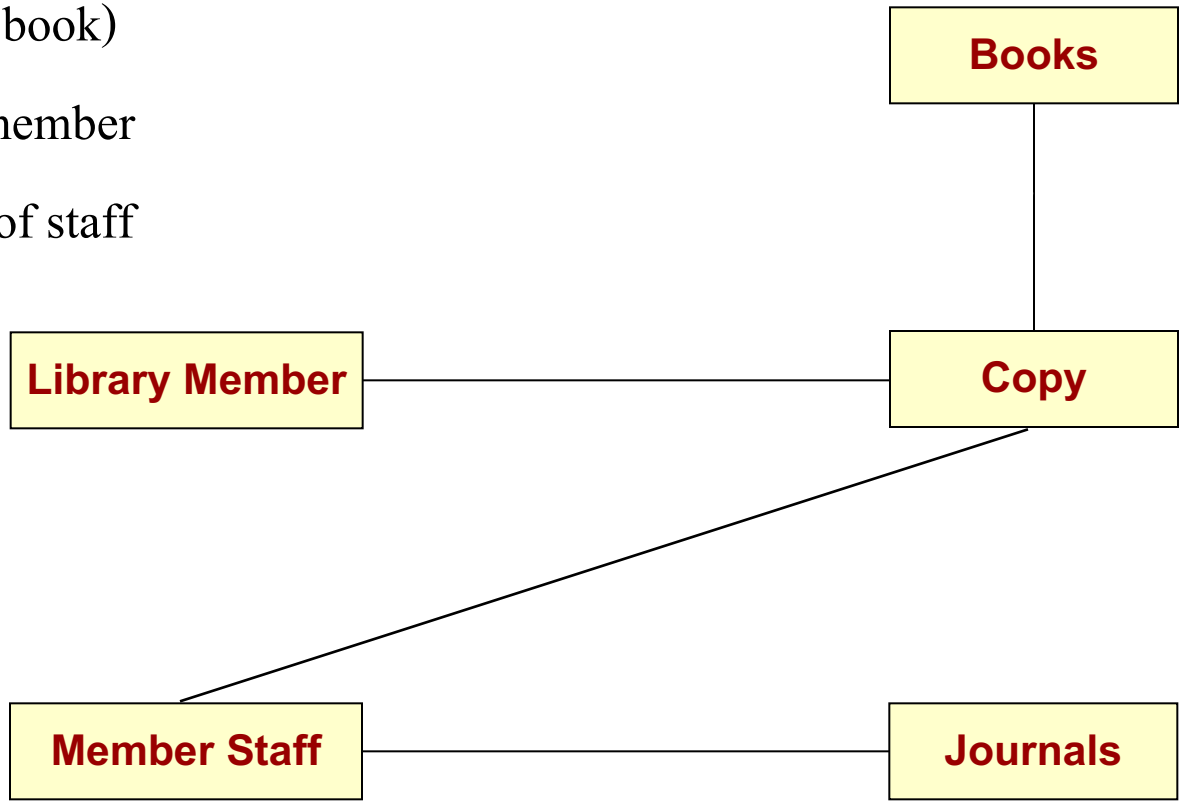
- ขั้นตอนต่อไปเป็นการตัดคำนามจากคลาสคู่แข่งที่มีความหมายซ้ำกันหรือไม่เกี่ยวข้องออกไป เช่น
  - **Library** เป็นสิ่งที่อยู่นอกเหนือขอบเขตของระบบ
  - **Short term loan** เนื่องจากการ loan ในความเป็นจริงถือเป็นเหตุการณ์ที่เกิดขึ้น ซึ่งยังไม่สามารถนำมากำหนดเป็นออปเจคภายในระบบได้ขณะนี้
  - **Member of the library** มีความหมายซ้ำซ้อน
  - **Week** เป็นหน่วยของการวัดไม่สามารถกำหนดเป็นออปเจคได้
  - **Item** มีความหมายคลุมเครือ ไม่ชัดเจน
  - **Time** เป็นสิ่งที่อยู่นอกเหนือขอบเขตของระบบ
  - **System** เป็นนิยามที่ใช้ระบุในความต้องการของระบบ ไม่ใช่ปัญหาที่แท้จริง
  - **Rule** เช่นเดียวกับ System

The remaining Classes

- คลาสคู่แข่งที่ถูกตัดออกไป ได้แก่ :
  - Library , **books , journals , copies of a given book** , short term loans , **library member** , weeks , Members of the library , items , time , **members of staff** , System and rules
- คลาสที่เหลืออยู่ ได้แก่
  - **Book**
  - **Journal**
  - **Copy (of book)**
  - **Library member**
  - **Member of staff**

Relations between Classes

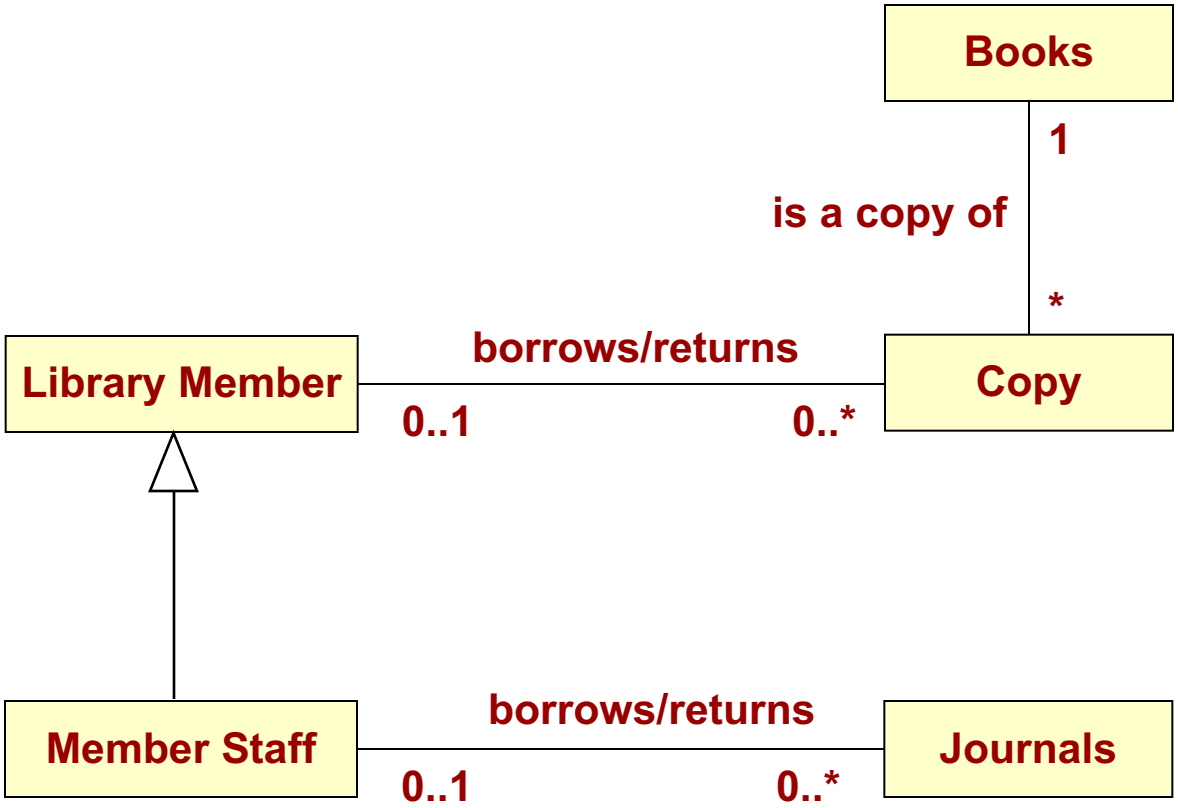
- Book
- Journal
- Copy (of book)
- Library member
- Member of staff



Let's revise our class model

- ในขั้นตอนสุดท้ายอาจทำการแก้ไขความถูกต้องของคลาสดังต่อไปนี้
  - **MemberOfStaff** ใช้ความสัมพันธ์ที่มีลักษณะเดียวกับ LibraryMember
  - ดังนั้นจึงสามารถระบุได้ว่า member of staff ถือเป็น library member ตามไปด้วย
- ความสัมพันธ์ที่ปรากฏอยู่ในคลาสไดอาแกรมจึงเป็นแบบ generalization ที่เกิดขึ้นระหว่าง LibraryMember และ MemberOfStaff
- ในระดับของภาษาโปรแกรมจึงได้แก่คุณสมบัติในการสืบทอดนั่นเอง
  - โดย MemberOfStaff เป็นคลาสที่สืบทอดมาจาก LibraryMember

Revised library class model



Exercise : More

- ระบบคลังข้อสอบจะถูกนำไปใช้ในการสอบของนักศึกษาผ่านเครือข่าย Internet
- ระบบคลังข้อสอบหนึ่ง ๆ สามารถจัดเก็บข้อสอบได้หลายวิชา แต่ละวิชาจะประกอบไปด้วยข้อสอบปรนัยที่ประกอบด้วยหนึ่งคำถามและตัวเลือกสี่คำตอบ
- จำนวนข้อสอบในแต่ละวิชาอาจมีไม่เท่ากัน แต่จะต้องไม่น้อยกว่า 25 ข้อและไม่เกิน-60 ข้อ
- นักศึกษาที่เข้าสอบจะต้องกรอกรายละเอียดข้อมูลที่จำเป็นในการเข้าสอบแต่ละครั้ง
- เมื่อนักศึกษาเริ่มเข้าสอบระบบจะแสดงข้อสอบทีละข้อ เพื่อให้นักศึกษาเลือกคำตอบที่ถูกที่สุดเพียงข้อเดียว
- เมื่อเสร็จสิ้นการสอบนักศึกษาระบบจะตรวจสอบและแจ้งผลคะแนนสอบให้ผู้เข้าสอบได้ทราบในทันที