

หลักการออกแบบและใช้งานคลาส

Actor

- ให้เราสร้างออบเจกต์ได้
- แล้วทำงานอย่างไรอย่างหนึ่งให้กับโปรแกรมอื่นๆได้
- เช่น **Scanner, Random**

Utility

- มีแต่ค่าคงที่กับ `static method` ให้เรียกใช้
- เช่น `Math`

หลักการออกแบบ

- ชื่อคลาสต้องเป็นคำนาม
- ชื่อคลาสต้องสื่อให้เรารู้ว่าจะทำอะไรกับออบเจกต์ของคลาสนั้นได้บ้าง
 - ตัวอย่าง ถ้าทำโปรแกรมเกี่ยวกับการจ่ายเงินเดือน เราไม่ควรสร้างคลาสคลาสเดียวที่ทำทุกอย่าง อย่าง **PaycheckProgram**
 - แต่เราควรเขียน **Paycheck** ขึ้นมา แล้วนิยามเมธอดที่เปลี่ยนค่าหรือจัดการ **Paycheck** ได้ แล้วจึงนำ **Paycheck** ออบเจกต์ไปใช้ในคลาสที่เรารัน **main**

- ถ้าเกิดคลาสมีเมธอดที่ทำหน้าที่ไม่ค่อยเกี่ยวข้องกัน เราควรแยกคลาสนั้นออกเป็นสองคลาส ให้แต่ละคลาสเรียกเมธอดที่เกี่ยวข้องกันภายในคลาสนั้น
 - ทำให้โปรแกรมดูง่ายขึ้น แก้ไขง่ายขึ้น
 - แต่ว่าคลาสหนึ่งจะต้องใช้อีกคลาสหนึ่ง ถ้ามีมากๆ อาจเป็นปัญหา
 - ดังนั้นให้แยกคลาสเท่าที่จำเป็นเท่านั้น

- เวลาเขียนเมธอดต่างๆ ถ้าเป็นเมธอดที่มีหลักการทำงานเหมือนกันอยู่
แล้ว หัวเมธอดก็ต้องเขียนให้เป็นแนวเดียวกัน

Immutable class

- เป็นคลาสที่มีแต่ **accessor** เมธอดเท่านั้น
 - ตัวอย่างคือ **String** เมื่อออบเจกต์ที่เป็น **String** ถูกสร้างขึ้นมาแล้ว จะไม่มีวันเปลี่ยนแปลง เพราะไม่มีเมธอดในคลาส **String** ที่จะเปลี่ยนค่าอะไรของ **String** ออบเจกต์ได้เลย

`String name = "John";`

`String uppercase = name.toUpperCase(); // name ไม่เปลี่ยนนะ`

- ข้อดีของคลาสประเภทนี้คือ เค้า **ref** ของออบเจกต์ให้คนอื่นใช้งานได้โดยไม่ต้องกังวลว่าค่าจะเปลี่ยน

Side effect

- เมธอดที่มี **side effect** คือ เมธอดที่โค้ดของมันเปลี่ยนค่าที่มองเห็นได้ แม้ว่าเมธอดจะรันเสร็จไปแล้ว เช่น ค่าของ **instance variable**

ตัวอย่างเช่น

```
public class BankAccount{
```

```
....
```

```
public void transfer(double amount, BankAccount other){
```

```
    balance = balance - amount;
```

```
    other.deposit(amount);
```

```
}
```

```
}
```

เปลี่ยนค่าตัวแปรใน **this**

เปลี่ยนค่าตัวแปรในออบเจกต์อื่นด้วย

ข้อควรระวัง

- ถ้าใช้ออบเจกต์เป็นพารามิเตอร์ของเมธอด เราเปลี่ยนค่าภายในออบเจกต์นั้นได้ การเปลี่ยนแปลงจะยังอยู่แม้ว่าเมธอดจะรันเสร็จแล้ว
- แต่ว่า ถ้าพารามิเตอร์ของเมธอดเป็น **primitive type** เราจะเปลี่ยนค่ายังไง ก็จะมีผลอยู่แค่ในเมธอดเท่านั้น
- ดูตัวอย่าง

- สมมุติเราเปลี่ยนเมธอด **transfer** เป็นแบบนี้

```
public void transfer(double amount, double otherBalance){  
    balance = balance - amount;  
    otherBalance = otherBalance + amount;  
}
```

แล้วไปเรียกใช้งานแบบนี้

```
double savingBalance = 1000;  
myAccount.transfer(500,savingBalance);  
System.out.println(savingBalance);
```

ตอนเข้าไปรันเมธอด มัน
จะถือว่ารับค่าเข้าไป
เท่านั้น ตัวต้นฉบับไม่
เปลี่ยน

อย่าใช้พารามิเตอร์เป็นที่ทด

- ถึงเราจะรู้ว่า พารามิเตอร์ในเมธอด ที่เป็น **primitive type** นั้น จะไม่มีผลกับภายนอก แต่ก็ไม่ควรใช้มันเป็นที่ทด

```
public void deposit(double amount){
```

```
    amount = balance + amount;
```

```
    ...
```

```
}
```

อย่างนี้จะทำให้พารามิเตอร์ที่เรา
เอาเข้ามา เปลี่ยนค่า ทำให้ใช้ค่า
ผิดจากที่ต้องการได้ภายในเมธอด
นี้ ทำให้เพื่อนที่มาใช้โปรแกรมเรา
ต่อสับสนด้วย

ถ้าต้องการทด สร้าง **local
variable** ขึ้นมาใหม่
ภายในเมธอดเลยดีกว่า

ข้อควรระวังอีกอย่าง

- ระวังจะลบหรือเปลี่ยนข้อมูลที่ไม่สมควรเปลี่ยน เช่น

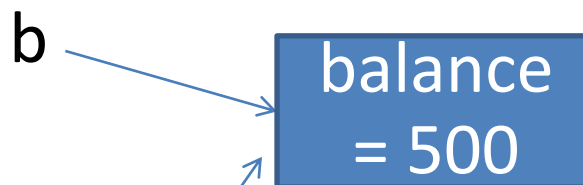
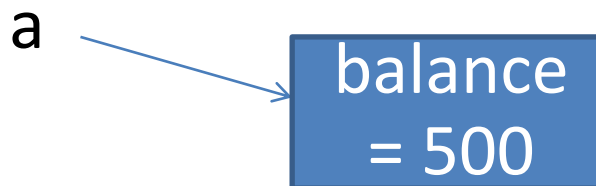
```
public class GradeBook{  
    public void addStudents(ArrayList<String> names){  
        while(names.size()>0){  
            String a_name = names.remove(0);  
            add(a_name);  
        }  
    }  
}
```

ตัวอาร์เรย์ลิสต์ต้นฉบับจะถูกลบด้วย ดังนั้นจะเอาไปใช้
งานที่อื่นในโปรแกรมอีกไม่ได้แล้ว

ข้างล่างนี้ผิด เพราะอะไร?

```
public class BankAccount{  
....  
    public void transfer(double amount, BankAccount  
    other){  
        balance = balance -amount;  
        double newBalance = other.balance +amount;  
        other = new BankAccount(newBalance);  
    }  
}
```

a.transfer(100,b);



ตอนเมธอดรัน จะรับ ref ของ b
เข้าไป ทำให้เกิดอีก ref ขึ้นมา

other

a



this

b



สรุปว่า **b** ไม่เปลี่ยนแปลง
เลยถ้าเขียนแบบนี้

other



precondition

- คือสิ่งที่ต้องเป็นจริงก่อนที่เมธอดจะถูกเรียกใช้ ถ้าไม่เป็นจริง เจ๊งแน่
- เช่น เมธอด **deposit** ต้องรับเลขที่ไม่เป็นลบเท่านั้น
- ตามปกติ ควรจะเขียนเป็นส่วนหนึ่งของคอมเมนต์ไว้ จะได้เอาไว้เช็ค

/**

Deposit money to the bank account.

@param amount the amount to deposit

(Precondition: amount ≥ 0)

*/

public void deposit(double amount){...}

- ถ้าเมธอดถูกเรียกโดยที่ **precondition** ของมันยังไม่เป็นจริง
 - เราอาจตรวจแล้ว **throw exception** ได้ (देखได้เรียนเอง)
 - หรือเราอาจปล่อยเลยตามเลย เพราะถือว่าคนเรียกเมธอดผิดเองที่ไม่ได้เรียกตาม **precondition**
 - देखก่อน ยังมีอีกวิธีสำหรับ **java**

assertion

```
Public double deposit(double amount){  
    assert amount >= 0;  
    balance = balance + amount;  
}
```

สามารถสั่ง

enable/disable ได้

ทำให้แยกส่วนนี้ออกจาก

โปรแกรมได้

ถ้า **enable** อยู่ แล้วได้ **false**

โปรแกรมจะหยุดทำงาน แล้ว **throw**

AssertionError

Java -ea MainClass

- ทำให้ **error** ชัดเจนแบบนี้ จะได้รู้ตัวว่าเขียนผิด และรีบแก้
- ดีกว่ามา **if** แล้ว **return** กลับ โดยไม่ทำอะไร เพราะอันนี้คนอ่านจะไม่เห็นว่าโปรแกรมผิดเลย

postcondition

- ได้ค่า **return value** ที่ถูกต้อง
- ออบเจกต์เปลี่ยนไปตามที่ต้องการ

```
/**
```

```
    Deposit money to the bank account.
```

```
    (Postcondition: getBalance() >= 0)
```

```
    @param amount the amount to deposit
```

```
    (Precondition: amount >= 0)
```

```
*/
```

```
public void deposit(double amount){...}
```

Class invariant

- คือสิ่งที่เป็จริงเสมอหลังจากสร้างออบเจกต์แล้ว ไม่ว่าจะเมธอดอะไรจะโดนเรียกก็เป็นจริงอยู่
- ถ้าเรารู้ว่า **class invariant** คืออะไร เราก็ตรวจสอบโปรแกรมเราได้ว่าขณะนั้นเขียนถูกอยู่หรือไม่

```
public class BankAccount{
```

```
    double balance;
```

```
    /**
```

```
        Construct a bank account with a given balance.
```

```
        @param initialBalance the money in the account at the time it is open.
```

```
        (Precondition: initialBalance >=0)
```

```
    */
```

```
    public BankAccount(double initialBalance){
```

```
        balance = initialBalance;
```

```
    }
```

```
    /**
```

```
        Deposit money to the bank account.
```

```
        (Postcondition: getBalance() >= 0)
```

```
        @param amount the amount to deposit
```

```
        (Precondition: amount >=0)
```

```
    */
```

```
    public void deposit(double amount){...}
```

```
/**  
    Withdraws money from the bank account.  
    @param amount the amount to withdraw  
*/  
public void withdraw(double amount){...}
```

```
/**  
    Gets the current balance of the bank account.  
    @return the current balance  
*/  
public double getBalance(){...}
```

- เราสังเกตได้ว่า ตั้งแต่สร้างออบเจกต์ ค่า `getBalance() >= 0` เสมอแน่นอน ดูจาก **precondition** ของทุกเมธอด แล้วก็สังเกตดูด้วยว่า หลังจากแต่ละเมธอดถูกเรียกแล้ว ยังเป็นจริงอยู่หรือเปล่า
- เอาค่า **invariant** นี้ไปเขียนบรรยายเป็นคอมเมนต์ซะ

```
/**
```

```
...
```

```
(Invariant: getBalance() >= 0)
```

```
*/
```

```
public class BankAccount{...}
```

Static method

- ไม่ต้องสร้างออบเจกต์ ก็เรียกใช้ได้เลย เช่นเมธอดใน **utility class** อย่าง **Math**
- สร้างขึ้นเพื่อลดการเขียนโค้ด อย่างที่เรียนตอนปีหนึ่งนั่นแหละ แต่เมธอดแบบนี้จะไม่ต้องใช้เปลี่ยนหรืออ่านค่าของออบเจกต์ไง
- หลายๆเมธอดเป็น **static** เพราะว่ามันเล่นกับ **primitive type** เช่นตัวเลข ซึ่งเราสร้างออบเจกต์ไม่ได้
- เรียกเมธอดประเภทนี้ ต้องเรียกจากการใช้ชื่อคลาส

การใช้งานอีกอย่างของ **static method**

- ใช้เมื่อเราไม่ต้องการเปลี่ยนคลาสที่มีอยู่แล้ว หรือเปลี่ยนคลาสที่มีอยู่แล้วไม่ได้

```
public class Geometry{
```

```
    public static double area(Rectangle rect){
```

```
        return rect.getWidth()*rect.getHeight();
```

```
    }
```

```
}
```

เป็นของจาวา เราเปลี่ยนไม่ได้ เลยรับเป็นพารามิเตอร์ของเมธอดของคลาสใหม่แทน ให้เมธอดเป็น **static** เพราะเรามีออบเจกต์อยู่แล้ว คือตัว

Rectangle

หรือใช้เพื่อ **access/mutate** ตัวแปรที่เป็น **static**

- ใช้สำหรับจัดการกับตัวแปรของคลาส ที่แชร์กันระหว่างทุกออบเจกต์ในคลาสนั้น

วิธี initialize static variable

- ไม่ทำอะไร มันก็มีค่า **default** เอง
- **Assign** ค่า ตอนที่เรา **declare** มัน
`private static int x = 1000;`
- อีกวิธีคือใช้ **static initialization block**

Static import

```
import static java.lang.System.*;  
import static java.lang.Math.*;
```

```
double r = sqrt(PI);  
out.println(r);
```

ทำให้ไม่ต้องเขียนยาว
และได้ค่านาง่ายขึ้น

วิธี initialize instance variable

- ใช้ค่า default
- ใช้ constructor
- เขียนค่า default เอง ตอน declare ตัวแปรนั้น

```
public class Coin{  
    private double value =1;  
  
    ...  
}
```

ค่านี้จะเป็น default ตอนสร้างทุกออบเจกต์ของคลาสนี้

- 初始化块

```
public class Coin{  
    private double value ;  
    static int shared;  
    {  
        value =1;  
    }  
  
    static  
    {  
        shared = 1;  
    }  
  
    ...  
}
```