

Object and classes

บทนี้เรียนอะไรบ้าง

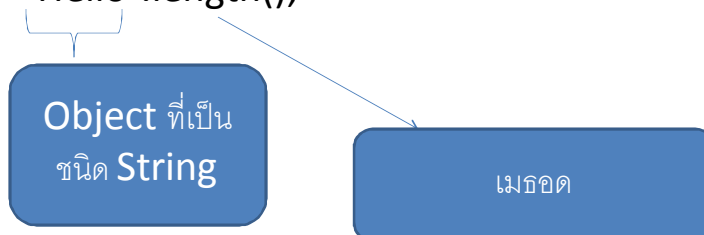
- แนะนำ **Object-oriented programming**
- บอกวิธีการสร้าง **object** จากคลาสที่จำมาให้มา
- สอนวิธีสร้างคลาสของเราเอง

แนะนำ Object-Oriented Programming

- โปรแกรมประกอบขึ้นจากออบเจกต์
- แต่ละออบเจกต์จะมีฟังก์ชันให้เรียกใช้
 - คนเรียกใช้ไม่จำเป็นต้องรู้ว่าข้างในออบเจกต์ทำงานยังไง
 - ขอให้ใช้ได้ตามที่ต้องการก็พอ
- วิธีคิดในการโปรแกรม
 - เมื่อต้องการแก้ปัญหา ให้ออกแบบตัวข้อมูลที่ต้องใช้ก่อน
 - แล้ววิธีแก้ปัญหาค่อยตามมา

ตัวอย่าง

`“Hello”.length();`



เมื่อรันแล้วจะได้ 5 เป็นคำตอบออกมา สามารถ print ออกมาได้ เช่น

`System.out.println(“Hello”.length());`

System.out.println("Hello".length());

นี่เป็น **implicit** พารามิเตอร์
ของเมธอด **println** ครั้งนี้

จริงๆมันก็คือ ออบเจ็กต์ที่เรียก
เมธอดนั่นเอง

นี่เป็น **explicit** พารามิเตอร์
ของเมธอด **println** ครั้งนี้

การสร้างออบเจ็กต์ พารามิเตอร์ ของคอน
สตรัคเตอร์

- ตัวอย่าง คลาส **Rectangle**

Rectangle box = new Rectangle(5,10,20,30);

ควรเก็บไว้ในตัวแปรไม่จำเป็น
เอาไปใช้ต่ออยาก

สร้างออบเจ็กต์

คอนสตรัคเตอร์ ไม่มี
พารามิเตอร์ก็ได้

เอาออบเจ็กต์ ที่สร้างไปเป็นพารามิเตอร์เมธอดอื่นก็ได้

API documentation

- คือ **help file** ที่บอกวิธีใช้งานทุกคลาสและเมธอดที่คนเขียนจาวาทำมาแล้ว
- เราเขียนของคลาสเราเองได้ด้วยนะ
- ลองทำความเข้าใจ จากหน้า
java.sun.com/javase/7/docs/api/index.html
 — ลองหาของ **Rectangle** ดู

เราสามารถเรียกเมธอดของ **Rectangle** กับออบเจกต์ชนิดนี้ที่เราเพิ่งสร้างขึ้นมา เช่น

 เคลื่อนที่เหลี่ยมไป 15 จุดตามแนวแกน x และ 25 จุดตามแนวแกน y

อย่าลืม ว่าถ้าจะเอา **Rectangle** มาใช้ได้ เราต้อง **import package** ที่มันอยู่ เข้ามาในโปรแกรมที่เราเขียนก่อน



Package คือกลุ่มของคลาสที่จัดไว้ชุดเดียวกันไว้ใช้งานร่วมกัน

exercise

- จงใช้คลาส **Day** ใน API เขียนโปรแกรมที่รับวันเดือนปีเกิด แล้วพิมพ์คำตอบว่า คนที่เกิดวันเดือนปีนั้น มีชีวิตมาแล้วทั้งหมดกี่วัน
- จงใช้คลาส **Picture** มาเขียนโปรแกรมในการเปลี่ยนรูปภาพ ทดลองเมธอดต่างๆตามที่ต้องการ

คลาส

- คลาสคือ **template** ที่เราจะใช้สร้างออบเจกต์
 - พุดง่าย ๆ มันเป็นตัวกำหนดว่า ออบเจกต์ที่สร้างจากมัน จะมีข้อมูลอะไรอยู่ภายในบ้าง
 - คลาสเปรียบเสมือนแม่พิมพ์ ส่วนออบเจกต์ก็เป็นสิ่งที่แม่พิมพ์พิมพ์ออกมา
- เมื่อเราสร้างออบเจกต์ขึ้นมาจากคลาส
 - เราเรียกเหตุการณ์นี้ว่า การสร้าง **instance** ของคลาส

- ภายในแต่ละออบเจกต์
 - ข้อมูล หรือค่าที่เราเรียกว่า **instance fields** หรือ **instance variables**
 - แต่ละออบเจกต์ จะมีค่าของแต่ละ **instance field** เป็นของตนเอง ไม่ใช่ค่าร่วมกับออบเจกต์อื่น
 - สถานะของค่าตัวแปรเหล่านี้ ทั้งหมด ถือเป็น **state** ของออบเจกต์
 - ซึ่งเมื่อเรียกเมธอด **state** ของออบเจกต์อาจเปลี่ยนได้
 - ส่วนฟังก์ชันที่ออบเจกต์นั้น เรียกเพื่อจัดการข้อมูลข้างต้น เรียกว่า **methods**

- ห้ามเข้าถึงค่าของ **instance field** โดยตรง
- ต้องอ่าน และเขียนค่าตัวแปร จากเมธอดเท่านั้น
- ทั้งนี้ เพื่อ
 - ป้องกันการยัดค่าผิดๆ ใส่องไปใน **instance field**
 - เช่น **a.x = 100;** จริงๆ x อาจจะมีค่าเกิน 99 ไม่ได้ ดังนั้นเขียนเมธอดคุมดีกว่า
 - เวลาจะเปลี่ยนพฤติกรรมทำอะไร จะได้แก่ในเมธอด ที่เดียว ไม่ต้องมาแก้หลายๆที่

ตัวอย่าง

```

public class Counter{
  private int value;
  public void count(){ value = value+1;}
  public int getValue(){return value;}
}

public class Auser{
  ...
  Counter tally = new Counter();
  Counter tally02 = new Counter();
  tally.count();
  tally.count();
  int result = tally.getValue(); //result is set to 2
  int result02 = tally02.getValue(); //result 02is 0, a default value
}

```

อ่านได้เฉพาะเมธอดของคลาส Counter เท่านั้น เพราะเป็น **private** คือ ต้องเรียกใช้ตัวแปรนี้ ผ่านเมธอดที่คลาส Counter มี เท่านั้น

↓
เรียกว่า
encapsulation

Instance var มีค่า default แต่ local var ต้อง initialize ทุกครั้งนะ ไม่เหมือนกัน

อีกตัวอย่าง

```

public class BankAccount{
  private double balance;
  public BankAccount(){...}
  public BankAccount(double initialBalance){...}
  public void deposit(double amount){...}
  public void withdraw(double amount){...}
  public double getBalance(){...}
}

```

Constructor ห้าม return นะ

อย่าลืม initialize ในคอนสตรัคเตอร์ เพราะ null เรียกเมธอดไม่ได้นะ

mutator

accessor

จากหน้าที่แล้ว ถ้าเราต้องการทำให้คลาสของเรามี **help file** แบบที่ **API** ตามปกติมี

เติมคอมเมนต์ไว้ด้านบนของคลาสหรือเมธอดที่ต้องการสร้าง **API** ดังตัวอย่างนี้

```
/**
 * 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(){...}
```

ใช้ **javadoc**
สร้างไฟล์ **API**
จากคอมเมนต์
อัตโนมัติ ทุก **tool**
มีให้ทดลอง

สรุป วิธีเขียนคลาสของเราขึ้นมาใช้งานเอง

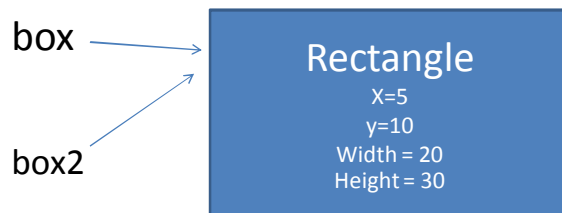
- คิดก่อน ว่าคลาสของเราต้องมีเมธอดอะไรที่เรียกใช้ได้บ้าง
- เขียนหัวเมธอดเหล่านี้ น้ออกมาก่อน
- เขียนคอมเมนต์ของเมธอดนี้ ันให้เรียบร้อย
- หาว่า ต้องใช้ **instance variable** อะไรบ้าง
- เขียนโค้ดของคอนสตรัคเตอร์และเมธอด
- **Test** ตัวโค้ดที่ละเมธอด ด้วยเครื่องมือ อย่างเช่น **Junit**
(คลาสไม่จำเป็นต้องมี **main** นะ ขอให้เมธอดให้เรียกใช้ได้ก็โอเคแล้ว)

คราวนี้ มาดูเรื่อง object ref

- ตัวแปรที่มีไทป์เป็นคลาส ไม่ได้เก็บออบเจ็กต์
 - แต่เก็บตำแหน่งในเมมโมรีของออบเจ็กต์นั้น
- **Object reference** หมายถึงตำแหน่งในเมมโมรีของออบเจ็กต์
- ตัวแปรที่เก็บตำแหน่งในเมมโมรีของออบเจ็กต์ เราเรียกว่าตัวแปรนั้น **refer to** ออบเจ็กต์นั้น

```
Rectangle box = new Rectangle(5,10,20,30);
```

```
Rectangle box2 = box;
```



เปรียบเทียบ int กับ อ็อบเจกต์

```
int n =13;
int n2 = n;
int n2 = 12;
```

เมื่อรันเสร็จ **n** จะเท่ากับ **13** ส่วน
n2 จะเท่ากับ **12**

```
Rectangle box = new
Rectangle(5,10,20,30);
Rectangle box2 = box;
box2.translate(15,25);
```

เมื่อรันเสร็จจะได้ **box** กับ **box2**
ที่ข้างในเหมือนกันทั้งคู่

this

ถ้าใช้ในเมธอด หรือคอนสตรัคเตอร์ จะหมายถึงอ็อบเจกต์ที่เมธอดนี้ ulla ถูกเรียก
เช่น

```
public void deposit(double amount){
    balance = balance + amount;
} ซึ่งมีความหมายเหมือนกับ
```

```
public void deposit(double amount){
    this.balance = this.balance + amount;
}
```

- หรือจะใช้เพื่อแยก `instance var` กับ `local var` ก็ได้ เช่น

```
public BankAccount(double balance){
    this.balance = balance;
}
```

ใช้คอนสตรัคเตอร์ตัวหนึ่งเป็นฐาน เรียกตัวอื่น

```
public BankAccount(double initialBalance){
    balance = initialBalance;
}

public BankAccount(){
    this(0); // การเรียกคอนสตรัคเตอร์อื่น ทำได้แค่บรรทัดแรกของ
           // คอนสตรัคเตอร์เท่านั้น
}
```