Prog Meth 101.zip

Who am I ?

Java

- OOP (Object-oriented programming)
- Cross-platform (Write once run everywhere)
- Fast

Chapter 1

Basic Programming

Variable

- int (... -2 -1 0 1 2 ...)
- long (64 bit int)
- float (1.01 1.414 1.721 ...)
- double (64 bit float)
- char (A B C D ...)
- String* (“Cat“, “Dog“, “Bird“, ...)
- boolean (True or False)

เป็นตัวแปรประเภท dynamic type !!!

Variable Declaration

Python

```
a = 5
b = 1.312
c = “cat“
```

Java

```
int a = 5;
float b = 1.312f;
String c = “cat“;
```

Type Casting

ปัญหาเรื่องตัวแปรประเภททศนิยม

float a = 5 / 2; // a = 2.0 WHY ?

5 เป็น integer

2 เป็น integer

ดังนั้น 5 / 2 = 2 (เอาทศนิยมออก)

วิธีแก้ปัญหา

double a = (double) 5 / 2;

Type Casting แบบอื่นๆ

- เอาทศนิยมออก

int a = (int) 3.5; // a = 3
int b = 3.5; // b = 3

Condition Statement : If-else

Python

```
if expression :
    // do something
elif expression2 :
    // do something
else :
    // do something
```

Java

```
if(expression) {
    // do something
} else if(expression2) {
    // do something
} else {
    // do something
}
```

Condition Statement : Switch-case

Python

ไม่มี Swith Case หรือใช้วิธีอื่นในการจัดการ Switch Case เอา

Java

```
switch (option) {
    case value1 :
        // statement 1
    case value2 :
        // statement 2
    case value3 :
        // statement 3
    case ...
    default ...
}
```

Condition Statement : While

Python

```
while condition :
    // statement
```

Java

```
while (condition) {
    // statement
}
```

Condition Statement : For

Python

for *item* in *list* :
    // do something

Java

for(*initialization*; *condition*; *increment*) {
    // do something
}

for(*item* : *list*) {
    // do something
}

Condition Statement : Break Continue

Python

```
while condition :
    ...
    break
    ...
```

```
while condition :
    ...
    continue
    ...
```

Java

```
while (condition) {
    ...
    break;
    ...
}
```

```
while (condition) {
    ...
    continue;
    ...
}
```

Function

Python

```
def functionName(parameter, ...) :
    // statement
    return variable

def area(w, h) :
    return w * h

def displayText(str) :
    print(str)
```

Java

```
returnType functionName(parameter, ...)
{
    // statement
    return variable;
}

int area(int w, int h) {
    return w * h;
}

void displayText(String str) {
    System.out.println(str);
}
```

Container

Array

```
int[] studentid = new int[1000];
double[] score = {3.5, 1.2, 5.7, 5};

int size = score.length;    // size = 4

score[0];    // 3.5
score[1];    // 1.2
score[2];    // 5.7
score[3];    // 5
```

- ใช้เก็บข้อมูลประเภทเดียวกันเยอะๆ
- มีความคล้าย List บน Python มาก แต่ Syntax ค่อนข้างแตกต่างกันเยอะ
- เราจำเป็นต้อง new หรือใช้ { } ในการประกาศ Array เสมอ
- ใช้ xxx.length ในการดึงจำนวนข้อมูลใน Array
- การดึงข้อมูลจาก Array ใช้ [] เหมือน Python ได้เลย แต่ไม่สามารถดึง sublist โดยใช้ [:] ได้

ArrayList

```
import java.util.ArrayList;

ArrayList<Integer> stock = new ArrayList<Integer>();

stock.add(123);
stock.add(456);
stock.add(789);
stock.add(999);

stock.size();    // 4

stock.get(0);   // 123
stock.get(1);   // 456

stock.remove(0);    // remove 123

stock.get(0);   // 456
stock.size();   // 3

stock.toString();   // [456, 789, 999]
```

- ArrayList เป็น Array ที่ขนาดไม่คงที่ เราสามารถใช้ ArrayList ในการเก็บข้อมูลจำนวนเยอะๆที่ไม่ทราบจำนวนที่แน่นอนได้
- แต่การประกาศเราจะไม่ใช้ตัวแปรประเภท Primative แต่ใช้ Class แทน
- มี method ให้เล่นสำหรับ ArrayList เยอะมาก เช่น xxx.add(); xxx.size(); xxx.remove(); xxx.get(); ในขณะที่ Array แบบธรรมดานั้นมีจำกัด

Hashtable

```
import java.util.Hashtable;

Hashtable<Integer, String> student = new Hashtable<Integer,
String>();

student.put(54321, “Cat”);
student.put(65432, “Dog”);
student.put(76543, “Bird”);
student.put(87654, “Dolphin”);

student.get(54321);    // Cat

student.containsKey(87654);    // true
student.containsKey(12345);    // false
```

- Hashtable คล้ายๆ Dictionary ใน Python แต่ค่อนข้างใช้ยากกว่า Python มาก
- ลักษณะการเก็บข้อมูลจะเป็น key-value เป็นคู่ๆกัน
- การใส่ข้อมูล (put) หรือการดึงข้อมูล (get) ต้องเรียกผ่าน function ของตัวแปร
- การประกาศและการเรียกใช้จะมีพื้นฐานคล้ายๆกับ ArrayList

DEMO

Chapter 2

Class & Access Modifier

Class ?????

Object ?????

Single Class

1. Property (All variable)
2. Method (All function)
3. Constructor

Example 1

```
class Human {

    int height;
    float weight;
    String nationality;
    ....

    void speak() { ... }
    void walk() { ... }
    void learn() { ... }
    ...
}
```

- ยกตัวอย่างคลาสมนุษย์
- มนุษย์ก็จะมี Property หลายอย่าง เช่น ส่วนสูง น้ำหนัก สัญชาติ
- กลไกของมนุษย์ สามารถพูดได้ วิ่ง หรือเรียนรู้ได้

Example 2

```
class Car {

      int capacity;
      float maxSpeed;
      String type;
      ....

      void run() { ... }
      void playRadio() { ... }
      ...
}
```

- Class รถ ก็มีจำนวนคนที่จุได้ ความเร็วสูงสุด ชนิดของรถ กลไกที่ทำงานได้ก็มีสามารถวิ่งได้ เล่นวิทยุได้ เป็นต้น

Example 3

Dice.java

```
class Dice {

	int faceNumber;
	int[] faceAvailable = {1,2,3,4,5,6};

	Dice() {
		faceNumber = 1;
	}

	void roll() {
		int randomIndex = (int) (Math.random() * faceAvail-
able.length);
		faceNumber = faceAvailable[randomIndex];
	}

	void show() {
		System.out.println(faceNumber);
	}
}
```

- เราจะสร้าง class ลูกเต๋า
- ลูกเต๋าจะมี เลขของลูกเต๋าที่โชว์ขึ้น (faceNumber) และหน้าทั้งหมดที่ปรากฏบนลูกเต๋า (faceAvailable)
- ลูกเต๋าสามารถทอยได้ (roll) และแสดงเลขของลูกเต๋าได้ (show)
- Constructor คือกำหนดค่าเริ่มต้นให้แก่ลูกเต๋า ซึ่งค่าที่เราจะกำหนดในที่นี่คือให้ลูกเต๋ามีหน้าเป็นเลข 1

Constructor

- Default instantiate
- Overloading

Create Object

Main.java

```
public class Main {

    public static void main(String[] args) {
        Dice myDice = new Dice();

        myDice.show();

        for(int i=0; i<6; i++) {
            myDice.roll();
            myDice.show();
        }
    }
}
```

- เราจะสร้าง Object จาก Class ที่เราเขียนขึ้น
- Object กับ Class คล้ายๆกันแต่ไม่เหมือนกันเลย คนละความหมาย
- เราจะ “สร้าง“ Object ใหม่จากการ new เสมอ ขาด new ไม่ได้
- สามารถเรียก method ผ่าน object โดยใช้ xxx.methodName(); หาก Method นั้นมีพารามิเตอร์ก็ใส่พารามิเตอร์ด้วย

Welcome to world of

Reference

Reference

Main.java

```
public class Main {

    public static void main(String[] args) {

        Dice diceA = new Dice();
        Dice diceB = diceA;

        for(int i=0; i<6; i++) {
            diceA.roll();

            diceA.show();
            diceB.show();
            System.out.print('\n');
        }
    }
}
```

- จะเห็นว่า faceNumber ของ diceB เปลี่ยนไปตาม diceA คำถามคือเพราะอะไร ?
- เพราะว่า diceB จะเก็บ reference ที่โยงไปหา diceA เอาไว้ ทำให้ di-ceA และ diceB นั้นถือ object ตัวเดียวกัน

Reference

Main.java

```
public class Main {

    public static void main(String[] args) {

        Dice diceA = new Dice();
        Dice diceB = new Dice();

        for(int i=0; i<6; i++) {
            diceA.roll();
            diceB.roll();

            diceA.show();
            diceB.show();
            System.out.print('\n');
        }
    }
}
```

- ถ้า new แยกกัน จะหมายถึง diceA และ diceB ถือ object คนละตัวกัน ดังนั้นอย่าลืมว่า ถ้าเราจะสร้าง object ใหม่ต้อง new ใหม่เสมอ “ทุกครั้ง“
- กรณีที่ต้องการ copy ข้อมูลทั้งหมดใน diceA ไปยัง diceB นั้น จะสามารถทำด้วยโดยการ clone object *(เนื้อหานี้จะไม่สอน แต่สามารถ Search ได้โดยใช้ keyword ว่า Clone Object Java)*

Access Modifier

- สามารถแสดง ป้องกัน ควบคุมการเข้าถึงข้อมูลต่างๆบน Object
- เหมาะสำหรับงานเล็กๆไปจนถึงงานใหญ่ สามารถสเกลได้
- เหมาะแก่การให้คนอื่นไปพัฒนาต่อ หรือพัฒนาที่มีคนเยอะๆ คนที่พัฒนาไม่จำเป็นต้องรู้รายละเอียดอะไรมากก็สามารถพัฒนาต่อจากโค้ดเราได้
- โค้ดสวย
- จำเป็นต้องใช้ในเรื่องของการ extends class (อยู่ในบทถัดไป)

Access Modifier

- Public
- Private
- Protected
- No modifier

Modifier	Class	Package	Subclass	World
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
No modifier	Yes	Yes	No	No
Private	Yes	No	No	No

งงจัง งั้น..

DEMO

Miscellaneous

This

- ใช้สำหรับอ้างอิง object ปัจจุบัน
- นิยมใช้เมื่อ method รับ parameter ที่ตั้งชื่อเหมือนกับ property ใน class

Final

- ใช้สำหรับตัวแปรที่ไม่ต้องการแก้ไข

Static

- เป็นตัวแปรที่เราต้องการควบคุมให้ทุก object นั้นใช้ร่วมกัน (ปกติแล้วแต่ละ object ตัวแปรแต่ละตัวนั้นจะเป็นอิสระต่อกัน)
- เพิ่ม Performance

Final Static

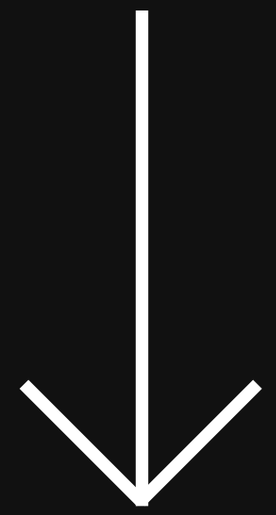
- เป็นทั้งค่าคงที่ (ไม่ต้องการแก้ไขอะไร) และเป็นค่าที่ใช้ร่วมกัน
- ส่วนมากนิยมใช้ในเวลาส่งพารามิเตอร์เป็น int แต่อยากส่งเป็นชื่อภาษาอังกฤษ
- นิยมตั้งชื่อแบบนี้ THIS_IS_VAR_NAME

เข้าใจรึยัง ?

public static void main (String[] args)

Chapter 3.1
Inheritance

Base Class (Super Class)



Extends Class (Sub Class)

เพื่ออะไร ?

- เราต้องการสร้าง class บางอย่าง ที่มีคุณสมบัติพื้นฐานมาจาก class อีก class
- การ extends เปรียบเสมือนกับการปรับปรุงแต่งของบางอย่างจาก base class จึงทำให้เราไม่จำเป็นต้องเขียน class ใหม่ทั้งหมด แต่เขียนส่วนที่ “เพิ่มเติม/แก้ไข“ เท่านั้น
- เหมาะสำหรับ Copy & Paste style
- ก่อนเรียน WTF is this ------> หลังเรียน Mind blow
- ถ้าพูดในแง่ประสิทธิภาพ และความสะดวกของการเขียนโค้ด ถือว่าสะดวกมากๆๆๆๆๆ

Animal.java

```
public class Animal {
	public Animal() {}
	public void eat() { ..... }
	public void breath() { ..... }
	public void sound() { System.out.println("..."); }
}
```

Cat.java

```
public class Cat extends Animal {
	public Cat() {}
	public void sound() { System.out.println("Meow"); }
}
```

Dog.java

```
public class Dog extends Animal {
	public Cat() {}
	public void sound() { System.out.println("Hong"); }
}
```

Seal.java

```
public class Seal extends Animal {
	public Cat() {}
	public void sound() { System.out.println("Aung"); }
	public void swim() { ..... }
}
```

- ยกตัวอย่างเราต้องการจะสร้าง Class ของสัตว์ 3 ตัว ได้แก่ หมา แมว และแมวน้ำ
- แต่ละตัวนั้นมีสิ่งหนึ่งที่เหมือนกันคือมันสามารถกินอาหารได้ หายใจได้ และส่งเสียงได้ แต่ว่าการส่งเสียงของแต่ละตัวไม่เหมือนกัน
- เราจึง extends จาก class Animal แล้วมา “แก้ไข“ เสียงของสัตว์แต่ละตัวให้ตัวตามลักษณะของสัตว์แต่ละตัว
- และแมวน้ำยังมีข้อได้เปรียบเพื่อนอยู่อย่างหนึ่งคือมันว่ายน้ำได้ จึงสามารถเพิ่ม method การว่ายน้ำให้แมวน้ำได้

หลักการ

- โดยหลักๆแล้ว คลาสที่ Extends นั้น ส่วนมากแล้วจะ “เพิ่มเติม“ หรือ “แก้ไข“ จาก Base Class แต่นั่นไม่ใช่ประเด็นหลัก
- ส่วนที่สำคัญจริงจะเป็นการที่มี Class จำนวนมาก สามารถ Extends ได้จาก Base class จึงใช้คุณสมบัติของการ Extends มาช่วยในการเขียนโค้ด
- การ Extends สามารถ Extends ต่อทอดเป็นทอดๆหลายๆชั้นได้ เหมือน เรามีลูก หลาน เหลน เป็นต้น
- ถ้าใช้ร่วมกับ Access Modifier (เช่น Private Protected) จะทำให้การ Extends มีความยืดหยุ่นมากขึ้น ลดความซับซ้อนของการ Extends ได้อย่างมาก (เพราะไม่จำเป็นต้องเห็นทุกอย่างใน Base class)

ข้อควรระวัง

- ถ้าคลาส 1 คลาส จะสามารถ extends คลาสได้เพียงแค่คลาสเดียวเท่านั้น
- ไม่สามารถ extends เป็นวงวนได้ เช่น C -> B และ B -> A ไม่สามารถ A -> C ได้
- แน่นอนว่า C -> B แล้ว C -> A ไม่ได้ เพราะ extends ได้แค่คลาสเดียว

- การ Override (แก้ไข) method ไม่สามารถ override method ที่เป็น static หรือ final ได้

Super

Sub.java

```
-------------------------------------------
public class Sub extends Base {

        int b;

        public Sub() {
                super();
                b = 5;
                System.out.println(a + " " + b);
        }

        @Override
        public change() {
                super.change();
                b = 10;
                System.out.println(a + " " + b);
        }
}
```

Base.java

```
-------------------------------------------
public class Base {
        int a;
        public Base() { a = 3; }
        public change() { a = 100; }
}
```

- Super จะเหมือนเป็นตัว Reference ที่อ้างอิงไปยัง Super Class
- หาก super() เฉยๆจะหมายถึง เรียก Constructor ของ Super Class
- ถ้าเป็น super.xxx() จะหมายถึงเรียก Method xxx บน super class
- Super เรียกใช้บน Base Class ได้ก็จริง คำถามคือทำไมถึงเรียกได้ๆ ทั้งๆ ที่ไม่ได้ Extends มาจาก Class ใด
- เพราะว่า Class ทุกคลาสนั้นสืบทอดมาจาก java.lang.Object โดยอัตโนมัติ การเรียก super() จะหมายถึงเรียน Constructor ของ Object

Chapter 3.2

Polymorphism

Polymophism (Class Casting)

อันนี้ต้องจด

ทำไมต้องจด เพราะคนสอนขี้เกียจทำสไลด์

Polymorphism

Cat.java

```
-----------------------------------------
public class Cat extends Animal {
        ......
}
```

Dog.java

```
-----------------------------------------
public class Dog extends Animal {
        .......
}
```

Main.java

```
-----------------------------------------
public class Main {
        public static void main(String[] args) {
                Animal A = new Cat();
                Animal B = new Dog();

                Animal[] animals = {new Cat(), new Dog()};
        }
}
```

- อธิบาย Polymorphism ใน 1 บรรทัด คือ object เป็นอะไรได้หลายแบบ (หลากหลาย)
- เช่น object cat เป็นได้ทั้ง Cat เป็นได้ทั้ง Animal
- หลายคนจะงง มัน make sense อยู่แล้วนี่ แต่หลายคนอาจจะยังไม่รู้ว่าเราสามารถประกาศ A ให้เป็น Animal ได้ ดูจากตัวอย่างใน Main
- จะเห็นว่ารูปแบบนี้คล้ายๆกับ Class casting แต่ไม่ใช่ทีเดียวเสมอไป
- เพราะอะไร ? คงต้องถามกลับว่า ถ้าให้ A เป็นแมว แต่ถูก cast เป็น Animal คำถามคือ หลังจาก A เป็น Animal จะกลับเป็นแมวได้อีกหรือไม่
- คำตอบคือ “ได้“
- นี่คือ concept ของ Polymorphism

Polymorphism (instanceof)

Main.java

```
public class Main {
	public static void main(String[] args) {
		Animal A = new Cat();
		Animal B = new Dog();
		Animal C = new Bird();

		Animal[] animals = {A, B, C};

		for(int i=0; i<animals.length; i++) {
			if(animals[i] instanceof Cat) {
				System.out.println(
					animals[i].getClass().getName() + " is a cat"
				);
			} else {
				System.out.println(
					animals[i].getClass().getName() + " is not a cat"
				);
			}
		}
	}
}
```

- เราสามารถใช้ instance of ในการระบุได้ว่า object นั้นมีคุณสมบัติตรงกับ Class ได้บ้าง
- P.S. instanceof ต้องเขียนติดกันนาจาสำหรับ coding
- instanceof จะให้ผลลัพธ์ออกมาในรูปของ true/false เพื่อบ่งบอกว่า object นั้นเป็นตัวแทนหรือพูดง่ายๆ คือมีคุณสมบัติเป็น Class นั้นอยู่หรือไม่
- จากตัวอย่าง หมา ไม่เป็น instance ของแมว แต่ แมวเป็น instance ของแมว
- แต่ทั้ง แมว และ หมา ต่างเป็น instance ของ Animal ทั้งคู่
- instanceof จะเข้าใจเข้าใจความหมายว่าเป็นคุณสมบัติหรือเป็นตัวแทนของคลาสนั้นๆหรือไม่ก็ได้

จบแล้วเย้....

ซะเมื่อไหร่ละ...

Chapter 3.3

Encapsulation

Access Modifier +
Hiding Information

Encapsulation

StudentData.java

```
-----------------------------------------
public class StudentData {

	private studentID;

	public getStudentID() {
		return studentID;
	}
}
```

Car.java

```
-----------------------------------------
public class Car {

	private currentSpeed;

	public void increseSpeed( int speed ) {
		currentSpeed += speed;
	}
}
```

- คอนเซ็ปหลักๆของ Encapsulation หลักๆเลยคือ การซ่อนข้อมูลบางอย่างที่เราไม่ต้องการให้ข้างนอกมากยุ่งเกี่ยว ซึ่งข้อมูลในที่นี้มักจะเป็น variable
- ยกตัวอย่าง studentID เราต้องการให้อ่านได้ แต่ไม่ต้องการให้แก้ไขได้ เราก็เลยตั้ง studentID เป็น private แล้วสร้าง method สำหรับอ่านค่า studentID นั้นอย่างเดียว
- ยกตัวอย่างความเร็วรถ ที่ไม่ต้องการรู้ความเร็วปัจจุบัน แต่บังคับให้มันวิ่งตามคำเร็วที่กำหนดให้ก็พอ เราก็ private currentSpeed ไว้ แล้วให้ method increaseSpeed จัดการเอา

Encapsulation

- คอนเซ็ปของ Encapsulation มักมากับ ตัวแปรที่เป็น private protected
- เราสามารถจำกัดการ “อ่าน“ หรือ “แก้ไข“ variable ได้ โดยใช้หลักการนี้
- หากต้องการให้ “อ่าน“ ข้อมูลได้ เราก็จะสร้าง method สำหรับอ่านข้อมูลนั้น ปกตินิยมเรียกว่า getter
- หากต้องการ “แก้ไข“ ข้อมูลนั้น เราก็จะสร้าง method (รับ paramater เพื่อจะแก้ไขตัวแปรนั้นๆ) สำหรับการแก้ไขข้อมูลนั้น นิยมเรียกว่า setter
- ผสมกัน เป็นที่มาของ getter/setter *และอาจารย์ชอบใช้บ่อยด้วย*

- ปกติตัว setter นั้นเรามักนิยมใช้ this ในการ set variable ต่างๆควบคู่ด้วย (ตรงนี้คนสอนจะอธิบายเพิ่มเติม)

ยังไม่จบอีกหรอ T_T...

Chapter 3.4

Interface

Class ไม่สมประกอบ

Interface

Animal.java

```
public interface Animal {

    // some constant variable
    int a = 5;
    static bool b = true;
    final String c = “eiei“;

    // Some empty method
    public void eat();
    protected void swim();
    public void jump();
    public void walk();
    protected void speak();
}
```

- Interface นั้นจะทำหน้าที่คล้ายๆกับ Base Class คือรอให้คนอื่นมาสืบทอดต่อ แต่ความแตกต่างของ Class ปกติกับ Interface คือ Interface จะ “ไม่“ ระบุการทำงานของ method ไว้แม้แต่นิดเดียวเลย
- หน้าที่หลักๆคือให้คลาสอื่นมา “implements“ ต่อ
- จะเห็นว่าไม่ได้ใช้คำว่า “extends“
- เพราะ Java มันบังคับให้ใช้คำนี้
- หน้าที่หลักๆของ interface คือบรรจุกลไกบางอย่างที่เราต้องการ แต่เราไม่ได้ต้องการเขียนข้างใน แค่โยนให้คนอื่นไปเขียนต่อ
- เหมือนเราออกแบบการทำงานของไฟ คือถ้ากดสวิตช์ ไฟต้องเปิด แต่เราไม่ใช่คนออกแบบวงจรออก
- นี่จึงเป็น idea ของ interface

Interface (implements)

Box.java

```
-------------------------------------------
public interface Box {
	public void move();
	public void slide();
}
```

LargeBox.java

```
-------------------------------------------
public class LargeBox implements Box {
	@Override
	public void move() { System.out.println(“Move very slow“); }
	@Override
	public void slide() { System.out.println(“Can’t slide“); }
}
```

SmallBox.java

```
-------------------------------------------
public class LargeBox implements Box {
	@Override
	public void move() { System.out.println(“Move so fast“); }
	@Override
	public void slide() { System.out.println(“Easy to slide“); }
}
```

- เราจะไม่ใช้ extends กับ interface แต่จะใช้ Implements แทน
- การ Implements จะต้องใส่การกระทำให้แก่ Method นั้นเสมอ
- เหมือนกับว่าเราเป็นคนวางวงจรไฟฟ้าให้สวิตช์เปิดไฟได้นั่นแหละ
- ควรเขียน Override notation ด้านบน Method ด้วย

Interface & Implements

- Interface สามารถ extends Interface อื่นๆได้ แต่ Extends class ธรรมดาไม่ได้นะ
- แต่ความประหลาดอยู่ที่ interface 1 อัน สามารถ Extends interface ได้หลายอันๆ ทั้งๆที่ Class ปกติจะ extends ได้แค่อันเดียว
- Interface เหมือนอยู่คนละโลกกับ Class รูปแบบการเขียน และตัว Syntax เอง คนข้างพิลึกกึกกือกว่า Class อย่างมาก
- อย่าลืม @Override
- Class 1 Class สามารถมี Implements interface ได้ “หลาย“ อันได้ แต่ extends ได้อันเดียว อย่าลืมนะ
- ถ้าเอาการเขียน Interface มาประยุกต์กับ Polymophism ได้ เปรียบเสมือนเรามีรถถังอยู่ในมือ
- แต่เชื่อว่าตอนนี้หลายคนงง
- Class ปกติที่ Implements interface นั้นจะต้องมี Body ให้แก่ Method ทุกตัวของ Interface ที่ Implements มา
- ยกเว้น Abstract class, Abstract method ที่ไม่ต้องใส่ Body

เดี๋ยวนะ....

แล้ว Abstract Class มันคือ

อะร๊ายยยยย *เสียงสูงมาก*

ผ่างงงงง....

Chapter 3.5

Abstract Class

Abstract Class

Dice.java

```
-----------------------------------------------
public abstract class Dice {

	int diceFace;
	int[] diceAvailable = {1,2,3,4,5,6};

	public abstract void roll();
	public int getDiceFace() {
		return diceAvailable[diceFace];
	}
}
```

CompleteDice.java

```
-----------------------------------------------
public class CompleteDice extends Dice {
	@Override
	public void roll() {
		diceFace = (int) (Math.random() * this.diceAvailable.length);
	}
}
```

- Abstract Class ทำหน้าที่เป็นกึ่งกลางระหว่างโลกของ Class ธรรมดา กับโลกของ Interface
- จะมีบาง Method ที่ไม่ต้องเขียน Body ข้างใน โดยจะต้องกำหนดคำว่า abstract ข้างหน้า
- ซึ่งการที่ method ไม่มี Body เนี่ยมันหมายถึงว่า Class นั้นยังไม่สมประกอบ ต้องให้ Class อื่นมา extends เพื่อทำให้สมประกอบอีกที
- ดังนั้นคลาส Abstract Class ไม่สามารถ new ได้เหมือน Class ทั่วๆ ไป
- Abstract จะมองได้ว่าเป็น Class พิการก็ได้ เป็นการบีบบังคับว่าตัวเองไม่ใช่ Class สมบูรณ์ จึงต้องทำการ Extends อีกที

Abstract Class

- Abstract Class นั้นจะประกอบไปด้วย ส่วนที่สมบูรณ์แล้ว (Method ที่มีการทำงานครบถ้วน) และ ส่วนที่ไม่สมบูรณ์ (Method ที่ไม่มี Body และตัว Method เองก็เป็น Abstract)
- ยำ้อีกครั้ง Abstract Class ไม่สามารถสร้างเป็น object ได้ (เพราะ Class มันไม่สมบูรณ์) ดังนั้นต้อง Extends ก่อนแล้ว Override method ให้ครบ
- เมื่อครบแล้วเราก็สร้างเป็น Object ได้ แต่ไม่ใช่จาก Class ที่เป็น Abstract แน่นอน แต่มาจาก Class ที่ Extends abstract class แล้วต่างหาก
- Abstract Class เปรียบเสมือน blueprint ที่ยังไม่สมบูรณ์ดี แต่เป็น blueprint ที่มีข้อมูลอยู่บางส่วน คนที่มาสืบทอดก็สามารถแก้ไขหรือเพิ่มเติม blueprint ได้
- มีความสำคัญอย่างมากเมื่อต้องการสร้าง Class หลายๆตัวที่ อ้างอิงจะมาจาก Class พื้นฐาน
- Abstract Class ทำหน้าที่แทบใกล้เคียง Base Class เลย แต่จุดที่แตกต่างคือ Abstract Class นั้นมีการทำงานบางส่วนที่ต้องยกให้ Subclass ในการจัดการ
- เหมือน Blueprint ของรถที่ส่วนที่เป็นเครื่องยนต์หายไป แต่ว่าเราต้องการเครื่องยนต์นะ ใส่ตรงนี้นะ แล้วบริษัทต่างๆก็เอา Blueprint โครงรถนี้ไปสร้างเครื่องยนต์ข้างใน กระจายเป็นยี่ห้อต่างๆ

จบละนะ...

จบจริงๆจ้า 555

Next Chapter

Basic Game Programming