OOT - . .

Static, Final, Package and Interface

คลาส Center

```
1 package oot.lab6;

2 import oot.lab6.interfaces.Drivable;
3 import oot.lab6.car.model.City;
4 import oot.lab6.car.model.Vios;
5 import oot.lab6.car.Car;

6 public class Center {
7    public static void showBrand(Car c) {
8        System.out.println(c.getModel() + " is from brand:" + c.getBrand());
9        c.fix();
10       Drivable d = (Drivable) c;
11       d.drive();
12   }

13   public static void main(String[] args) {
14       final Vios viosCar = new Vios();
15       City cityCar = new City();
16       showBrand(viosCar);
17       showBrand(cityCar);
18   }
19}
```

บรรทัดที่ 1 ประกาศแพ็คเกจ (package) ของคลาส Centerในที่นี้คือ oot.lab6 โดยแพ็กเกจคือโครงสร้างเป็นระดับชั้นคล้ายโฟลเดอร์ (หรือไดเร็กทอรี) ที่ใช้จัดเก็บไฟล์ในภาษาจาวา แพ็กเกจใช้จัดเก็บหมวดหมู่ของคลาส (class)

บรรทัดที่ 2-5 เป็นการประกาศ import statement เพื่อบอกว่าเราจะนำคลาสไหนเข้ามาใช้บ้าง เหตุผลที่ต้อง import ก็เพราะว่าคลาสแต่ละคลาสอยู่ต่างแพ็คเกจกัน

บรรทัดที่ 13 เป็นการประกาศเมธอดชื่อ main เป็นชนิด void และเป็นประเภท static โดยรับพารามิเตอร์ชื่อ args เป็นชนิด อะเรย์ของ String ซึ่งเป็นข้อกำหนด ในภาษา Java ว่าการประกาศเมธอดแบบนี้เท่านั้นจึงจะใช้เป็นจุดเริ่มต้นของโปรแกรมได้

บรรทัดที่ 14 เป็นการสร้างวัตถุที่มีชนิด (type) เป็นคลาส Vios โดยใช้คอนสตรักเตอร์ Vios() แล้วเก็บค่าวัตถุนี้ไว้ในตัวแปรชื่อ viosCar จุดที่น่าสนใจคือ ในบรรทัดนี้มีการใช้คำว่า final หน้าการประกาศตัวแปร viosCar การประกาศ final ทำให้ตัวแปรนั้นสามารถตั้งค่าได้แค่ครั้งเดียว เช่นในตัวอย่าง viosCar = new Vios() โดยไม่สามารถเปลี่ยนค่าตัวแปร viosCar ได้อีกแล้ว

บรรทัดที่ 15 เป็นการสร้างวัตถุที่มีชนิด (type) เป็นคลาส City โดยใช้คอนสตรักเตอร์ City() แล้วเก็บค่าวัตถุนี้ไว้ในตัวแปรชื่อ cityCar สังเกตว่าในบรรทัดนี้ไม่มีการใช้ final ดังนั้นตัวแปร cityCar นี้จะยังสามารถตั้งค่าได้ใหม่ตามปกติ

บรรทัดที่ 16 เป็นการส่งตัวแปร viosCar (มีค่า: new Vios()) ให้เมธอดชื่อ showBrand(Car) เมธอด showBrand() ที่บรรทัดที่ 7 รับพารามิเตอร์เป็นตัวแปรของคลาส Car ดังนั้นต้องตรวจสอบก่อนว่า Vios เป็นคลาสลูกของ Car หรือไม่ (ดูที่คลาส Vios บรรทัดที่ 3) ปรากฏว่า คลาส Vios extends Car ดังนั้นจึงสามารถเรียกใช้ได้

บรรทัดที่ 8 มีการเรียกใช้เมธอด getModel() บนตัวแปร c และเมธอด getBrand() บนตัวแปร c เช่นกัน เมื่อวัตถุต่างชนิดกันถูกส่งเข้ามา โดยการอ้างอิงเป็นวัตถุของคลาสแม่ ก็จะเกิด polymorphism การเรียกจากบรรทัดที่ 16 มายังบรรทัดที่ 7, 8 นั้น ค่าวัตถุ c จะเป็น new Vios() ซึ่งในขณะนี้ต้องแทน new Vios() เป็น this นั่นเอง การค้นหาเมธอด getModel ก็จะเริ่มหาจากคลาสของวัตถุจริงปรากฏว่าไม่มีการประกาศเมธอด getModel ในคลาส Vios ดังนั้นต้องหาย้อนขึ้นไปที่คลาสแม่ซึ่งก็คือ คลาส Car (* ตามหลักการสืบทอด - inheritance) แต่ต้องไม่ลืมว่าการแทนค่า this จะต้องแทนด้วยวัตถุจริง

ในคลาส Car บรรทัดที่ 11 พบว่า มีเมธอด getModel() ซึ่งจะคืนค่าฟิลด์ model ของ this (this.model) ดังนั้นจึงต้องวิเคราะห์ this ต่อ (ในขณะนี้ this เป็น new Vios()) * หลักการเหมือนเดิมคือ ดูวัตถุแบบติด new + คอนสตรักเตอร์ไว้ จึงต้องไปดูต่อที่คอนสตรักเตอร์ของคลาส Vios พบว่า ในบรรทัดที่ 6 ของคอนสตรักเตอร์มีการตั้งค่า this.model ด้วยค่าจาก VIOS_MODEL ต้องค้นต่อว่า VIOS_MODEL อยู่ที่ไหน โดยเทคนิคการค้นคือ เริ่มหาจากคลาสที่เรียกใช้ค่านั้นในที่นี้ คือ คลาส Vios

ซึ่งจะไม่เจอการประกาศ จึงค้นต่อโดยค้นจากคลาสแม่จะพบว่า VIOS_MODEL ถูกประกาศเป็นค่าประเภท String มีการเข้าถึงแบบ public เป็นค่าระดับคลาส (กำหนดไว้ด้วย static) และเป็นค่าที่แก้ไขไม่ได้อีกแล้ว (กำหนดไว้ด้วย final) ค่านี้ถูกประกาศไว้ในคลาส Car

ค่าคงที่ประเภทระดับคลาสนี้สามารถเรียกใช้ได้ในหลายลักษณะ

- ถ้าเป็นคลาสที่สืบทอดกันมา เช่น Vios สืบทอดจาก Car จะสามารถอ้างถึงได้เลย
- ถ้าไม่ได้เป็นคลาสที่สืบทอดกัน เช่น Center กับ Car เวลาอ้างถึงของใน Car มาใช้ต้อง import แล้วระบุชื่อคลาสก่อน

  String x = Car.VIOS_MODEL;

  เป็นต้น

ค่าที่ได้จาก c.getModel() และ c.getBrand() ในบรรทัดที่ 8 จึงเป็นค่า “VIOS” และ”Toyota” ตามลำดับ

บรรทัดที่ 9 เป็นการเรียกใช้เมธอด fix บน c ในจุดนี้การเรียกใช้ยังมีผลจาก polymorphism เช่นเดิม โดยวัตถุจริงขณะนี้คือ new Vios() จึงต้องไปดูที่คลาส Vios ว่ามีเมธอด fix() หรือไม่ จะพบที่บรรทัดที่ 11

สำหรับบรรทัดที่ 10, 11 เป็นแนวคิดการใช้ interface โดย interface จะเป็นโครงสร้างคล้ายคลาส แต่จะไม่มีการ implement เมธอดใดๆเลย คือมีแต่โครงอย่างเดียว สิ่งที่คล้ายกับ interface ที่สุดคือ abstract class ที่ทุกๆ เมธอดประกาศเป็น abstract แต่จุดต่างก็คือ ในภาษา Java จะอนุญาตให้สืบทอดคลาสได้แบบ 1-1

A extends B

B extends C

เป็นต้น

แต่จะอนุญาตให้คลาสหนึ่งๆมีได้หลาย interface เช่น

A implements IB, IC, ID

เป็นต้น

ซึ่งในการใช้งานจริงแล้ว abstract class และ interface จะมีประโยชน์ต่างกัน

บรรทัดที่ 10 จะเป็นการ cast ให้วัตถุในตัวแปร c กลายเป็นตัวแปรของ interface Drivable เมื่อดูใน interface Drivable จะพบว่า มีการประกาศโครงสำหรับเมธอด drive() เพียงอย่างเดียว จึงทำให้ตัวแปร d เรียกใช้ได้เฉพาะเมธอดนี้เท่านั้น *

บรรทัดที่ 11 จะเป็นการเรียกใช้เมธอด drive() เราจะพิจารณาตัวแปร d เป็นวัตถุจริง (คือ new Vios()) แล้วนำไปค้นหาจากคลาส Vios ก่อน หากไม่มีจึงดูที่คลาสแม่ โดยดูที่เมธอด drive() ของบรรทัดที่ 21 ของคลาส Car จะพบว่า this ที่อ้างอิงอยู่นั้นจะแทนค่าด้วยวัตถุจริง และพิมพ์ค่าจากฟิลด์ในวัตถุจริงออกมา

บรรทัดที่ 17 จะทำงานในลักษณะเดียวกันกับบรรทัดที่ 16 แต่เป็นคลาส City

คลาส Car

```
1 package oot.lab6.car;

2 import oot.lab6.interfaces.Drivable;

3 public abstract class Car implements Drivable {
4    public static final String TOYOTA_BRAND = "Toyota";
5    public static final String VIOS_MODEL = "Vios";

6    protected String model;
7    protected String brand;

8    public void setModel(String model) {
9        this.model = model;
10   }

11   public String getModel() {
12       return this.model;
13   }

14   public void setBrand(String brand) {
15       this.brand = brand;
16   }

17   public String getBrand() {
18       return this.brand;
19   }

20   public abstract void fix();
```

```
21  public void drive() {
22      System.out.println("Driving " + this.brand + " " + this.model + " ...");
23  }
24}
```

คลาส City

```
1 package oot.lab6.car.model;

2 import oot.lab6.car.Car;

3 public class City extends Car {
4    public City() {
5        super();
6        this.model = "City";
7        this.brand = "Honda";
8    }

9    @Override
10   public void fix() {
11       System.out.println("Fixing at Honda center.");
12   }
13}
```

คลาส Vios

```
1 package oot.lab6.car.model;

2 import oot.lab6.car.Car;

3 public class Vios extends Car {
4    public Vios() {
5        super();
6        this.model = VIOS_MODEL;
7        this.brand = TOYOTA_BRAND;
8    }

9    @Override
10   public void fix() {
11       System.out.println("Welcome to Toyota maintenance.");
12   }
13}
```

interface Drivable

```
1 package oot.lab6.interfaces;

2 public interface Drivable {

3    void drive();

4 }
```