

Abstract Class & Interface

Pisit Nakjai

Abstract class

- ▶ Abstract class เป็นคลาสที่มีอย่างน้อย 1 Method เป็นประเภท Abstract Method
- ▶ Abstract Method คือ Method ที่มีแต่ชื่อไม่มีเนื้อหาหรือการทำงานของ Method
- ▶ Abstract class ไม่สามารถสร้างเป็น Object ได้โดยตรงต้องถ่ายทอดไปยัง Class ลูกแล้วให้ Class ลูกกำหนดการทำงานของ Abstract Method ก่อนจึงสามารถสร้างเป็น Object ได้ การกระทำดังกล่าวทำได้โดย Overriding Method

Abstract Class

- ▶ Class ลูกที่สืบทอด Abstract class จะต้องกำหนดการทำงานของ Abstract Method ทุก Method ก่อน
- ▶ Abstract Class อาจจะมี Method ที่กำหนดหน้าที่การทำงานแล้วด้วยได้ ซึ่งจะเรียก Method ในส่วนที่กำหนดการทำงานแล้วว่า Concrete Method

การสร้าง Abstract Class

- ▶ การสร้าง Abstract Class เหมือนกับการสร้าง Class ทั่วไป ต่างกันเพียงมี Abstract อยู่หน้า Class โดยมีรูปแบบดังนี้

```
abstract class AbstractClassName{  
    [AbstractMethodName]  
    [ConcreteMethodName]  
}
```

- ▶ โดยที่ AbstractClassName เป็นชื่อ Abstract Class
- ▶ AbstractMethod เป็นส่วนของการประกาศ Abstract Method ที่มีแต่ชื่อ ไม่มีส่วนของการทำงาน
- ▶ Concrete MethodName เป็นส่วนของการประกาศ Concrete Method จะมีหรือไม่มีก็ได้

ตัวอย่างการสร้าง Abstract Class

```
abstract class employee{  
    public abstract void setOT(double a);  
    public double calOT(int h,double s){  
        return h*s  
    }  
}
```

Abstract Method

ConcreteMethod

การถ่ายทอด Abstract Class ไปยัง Class ลูก

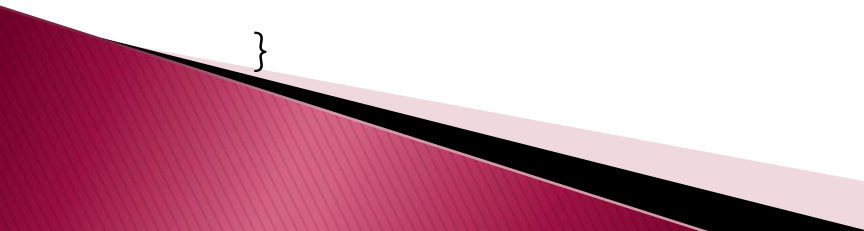
จากที่กล่าวไว้ว่าต้องนำ Abstract Class ไปถ่ายทอดให้กับ Class ลูกก่อน และกำหนดการทำงานของ Abstract Method จึงสามารถนำไปสร้าง Object ได้ ดังนั้นก่อนการใช้งานจะต้องมีการทำ Overriding Method ซึ่งมีรูปแบบดังนี้

```
class ClassName extends AbstractClassName{  
    AbstractMethodName(){  
        [Statements];  
    }  
}
```

ตัวอย่างการใช้งานการสืบทอด Abstract Class

```
public class AbstractTest extends employee{  
    public void setOT(double a){  
        System.out.println("OT Rate = "+s+ "Baht/Hr");  
    }  
}
```

```
public class Story{  
    public static void main(String[] args){  
        AbstractTest emp = new AbstractTest();  
        Scanner scan = new Scanner(System.in);  
        System.out.println("Enter OT Rate (Baht/Hr)");  
        double salary = scan.nextFloat();  
        System.out.println("Enter OT Hour >>>>");  
        int hr = scan.nextInt();  
        System.out.println("OT HOUR = " + hr + "HR.");  
        emp.setOT(salary);  
        System.out.println("Total OT=" + emp.calOT(hr,salary) + "BAHT");  
    }  
}
```



ตัวอย่างที่ 2

```
public abstract class Shape{  
    public abstract double calArea();  
    public abstract double calPerimeter();  
    public String getcolor(){  
        return "this shape is red color";  
    }  
}
```

```
public class Circle extends Shape{  
    protected double radius;  
    public Circle(double r){  
        radius = r;  
    }  
    public double calArea(){  
        return Math.PI * radius*radius;  
    }  
    public double calPerimeter(){  
        return 2* Math.PI * radius;  
    }  
}
```

```
public class Story{  
    public static void main(String args[]){  
        Shape shape;  
        shape = new Circle(20.2);  
        System.out.println(shape.calArea());  
        System.out.println(shape.calPerimeter());  
        System.out.println(shape.getcolor());  
    }  
}
```

Interface

Interface เป็น Class ที่มีทุก Method เป็น Abstract Method ซึ่งกำหนดเพียงแค่ว่ามี Method อะไรบ้างและมีการรับและส่งข้อมูลเป็นชนิดใดเปรียบเสมือนเป็นการตกลงที่มีไว้ให้ออบเจกต์ติดต่อสื่อสารถึงกันได้

การสร้าง Class Interface มีรูปแบบคล้ายกับ Abstract Class ต่างกันที่ Interface จะมีคำว่า Interface อยู่หน้า Class ดังนี้

```
[modifier] interface interfaceName{  
    AbstractMethod();  
}
```

Interface

```
[modifier] interface InterfaceName{  
    Method();  
}
```

modifier เป็นคีย์เวิร์ดที่กำหนดการเข้าถึง Class

interfaceName เป็นชื่อ Interface ที่ต้องการใช้งาน

เนื่องจาก Interface ประกอบด้วย Abstract Method เหมือนกับ AbstractClass เราจึงต้อง Implements อินเตอร์เฟสที่เราต้องการด้วย เพื่อกำหนดหน้าที่การทำงานให้ Abstract Method ก่อนที่จะนำสร้าง Object ได้

Interface

เนื่องจาก Interface ต้องทำการ Implement Method ก่อนที่จะนำไปใช้
ดังนั้น Class ที่จะเอา Interface class ไปใช้จะต้องทำการ Overring
Method ก่อนเสมอ เพราะ ทุก Method ของ Interface class ไม่ได้
ระบุการทำงานของ Method ไว้ รูปแบบการนำ Interface class ไปใช้
ดังนี้

```
[modifier] class ClassName implements InterfaceName{  
    AbstractMethod(){  
        [Statements];  
    }  
}
```

ตัวอย่างการใช้งาน Interface

```
public interface employee{  
    public void setOT(double s);  
    public double calOT(int h , double s);  
}
```

```
public class InterfaceTest implements employee{  
    public void setOT(double s){  
        System.out.println("OT Rate = " + s + " Bath/HR.");  
    }  
    public double calOT(int h,double s){  
        return h*s;  
    }  
}
```

ตัวอย่างการใช้งาน Interface

```
public class Story{  
    public static void main(String args[]){  
        InterfaceTest emp = new InterfaceTest();  
        Scanner scan = new Scanner(System.in);  
        System.out.print("Enter OT Rate >>> ");  
        double salary = scan.nextDouble();  
        System.out.print("Enter OT HOUR >>> ");  
        int hr = scan.nextInt();  
        System.out.print("OT HOUR = " + hr + "HR.");  
        emp.setOT(salary);  
        System.out.println("TOTAL OT = " + emp.calOT(hr,salary) + "BAHT");  
    }  
}
```


ข้อแตกต่างระหว่าง Abstract Class และ Interface

- ▶ Abstract Class เป็นคลาสที่ประกอบ Method 2 แบบ คือ แบบที่กำหนดการทำงาน (Concrete Method) และแบบไม่กำหนดการทำงาน (Abstract Method)
- ▶ Interface เป็นคลาส ที่ยังไม่มีกำหนดการทำงาน เมื่อเรานำ Interface มาใช้งานใน Class เราจะต้องกำหนดการทำงานให้กับเมธอดใน Interface นั้นก่อนเสมอ เรียกว่าเป็นการทำ Implements Interface
- ▶ คลาสใดๆ จะสามารถสืบทอด Abstract Class ได้เพียง Class เดียวเท่านั้น
- ▶ คลาสใดๆ จะใช้งาน Interface ได้มากกว่าหนึ่ง Interface เราจึงใช้งาน Interface สร้าง Multiheritance ได้

Multiple Inheritance

Multiple Inheritance คือการสืบทอดพฤติกรรมจาก Super Class

หลายๆ Class และถ่ายทอดสู่ Supclass เพียง Class เดียว

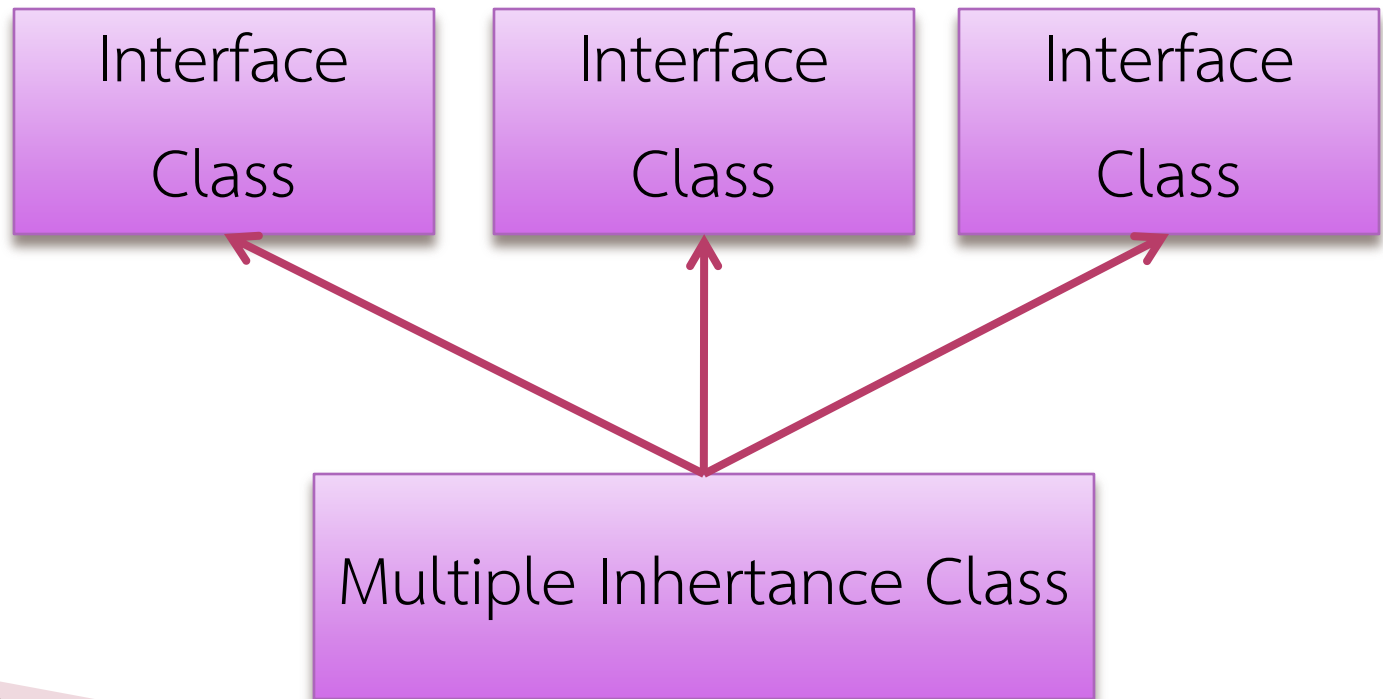
เช่น Class Amphibian สืบทอดมาจาก Class Car กับ Class Boat ดังนั้น

Class Amphibian



การสร้างและใช้งาน Multiple Inheritance

สำหรับการใช้งาน Multiple Inheritance นั้น Class ที่จะเป็น Super Class จะต้องมียลักษณะที่เป็น Interface เท่านั้นจึงจะสามารถใช้งานแบบ Multiple Inheritance ได้ ดังภาพ



ตัวอย่างการใช้งาน

Employee1.java

```
public interface Employee1 {  
    public float setOT(float s);  
}
```

Employee2.java

```
public interface Employee2 {  
    public float calOT(float s,int h);  
}
```

ตัวอย่างการใช้งาน

MultInterface.java

```
public class MultInterfaceTest implements employee1,employee2 {  
    public float setOT(float s) {  
        return s/30;  
    }  
  
    public float calOT(float s,int h){  
        return setOT(s)*h;  
    }  
}
```

```
public class Story{  
    public static void main(String[] args) {  
        MultilInterfaceTest emp = new MultilInterfaceTest();  
        Scanner scan = new Scanner(System.in);  
        System.out.print("Enter OT RATE >>> ");  
        float salary = scan.nextFloat();  
        System.out.print("Enter OT HOUR >>> ");  
        int hr = scan.nextInt();  
        System.out.println("OT HOUR  = " + hr + " HR.");  
        System.out.println("OT RATE  = " + emp.setOT(salary) + " BAHT/HR.");  
        System.out.println("TOTAL OT = " + emp.calOT(salary,hr) + "  
        BAHT");  
    }  
}
```

ตัวอย่างที่ 2

Dog.java

```
public interface Dog {  
    public void Bark();  
    public void Run();  
}
```

Robot.java

```
public interface Robot {  
    public void Calculator();  
    public void StatusOnOff();  
}
```

```
public class RoboDog implements Dog,Robot{
    boolean onoff,cal;
    private String name;

    public RoboDog(String n,boolean oo,boolean ca){
        name = n;
        onoff = oo;
        cal = ca;
        System.out.println("RoboDog's name is "+ name);
    }

    public void Bark(){
        System.out.println("RoboDog can bark");
    }
}
```


RoboDog.java (ต่อ)

```
public void Run(){  
    System.out.println("RoboDog can run about 150 Km/hr.");  
}  
  
public void Calculator(){  
    if(cal == true) System.out.println("RoboDog can Calculate.");  
    else System.out.println("RoboDog cannot calculate.");  
}  
  
public void StatusOnOff(){  
    if(onoff == true) System.out.println("RoboDog is available.");  
    else System.out.println("RoboDog is unavailable.");  
}  
}
```

StoryMultiple.java

```
public class StoryMultiple{  
    public static void main(String[] args){  
        RoboDog d1 = new RoboDog("D1",true,true);  
        d1.Bark();  
        d1.Run();  
        d1.Calculator();  
        d1.onOff=false;  
        d1.StatusOnOff();  
    }  
}
```

สรุป

| Interface | Abstract Class |
|--|--|
| ไม่ใช่ Class | เป็น Class |
| เมธอดใน Interface ไม่ต้องประกาศ abstract | บางเมธอดของ Abstract Class ต้องประกาศ Abstract |
| การเรียกใช้งานใช้คำว่า Implement | การเรียกใช้งานใช้คำว่า extends |
| คลาสที่ทำการ Implement จะต้องระบุเมธอดที่มีอยู่ใน Interface ให้ครบทุกเมธอด | คลาสที่ทำการ extends ไม่จำเป็นต้องระบุเมธอดที่มีอยู่ใน Abstract Class ครบทุก เมธอด |