เขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming)

เขียนโปรแกรมเชิงวัตถุ (OOP)

คือ การเขียนโปรแกรมอีกรูปแบบหนึ่ง โดยมองสิ่งต่างๆเป็นวัตถุ ซึ่งมีมุมมองจากพื้นฐานความจริงในชีวิตประจำวัน โดยในวัตถุนั้นจะประกอบด้วยคุณสมบัติและพฤติกรรม

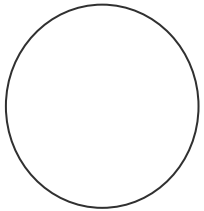
องค์ประกอบพื้นฐาน

คลาส (Class) คือ ต้นแบบของวัตถุ การจะสร้างวัตถุขึ้นมาได้จะต้องสร้างคลาสขึ้นมาเป็นโครงสร้างต้นแบบสำหรับวัตถุก่อนเสมอ

วัตถุหรือออบเจ็ค (Object) คือ สิ่งที่ถูกสร้างจากคลาสประกอบด้วยคุณสมบัติ และ พฤติกรรม

องค์ประกอบพื้นฐาน

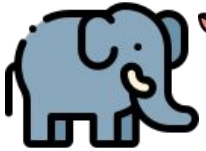
Class



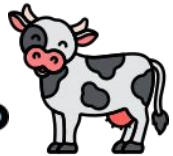
Animal

Object

สิงโต

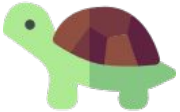


ช้าง



วัว

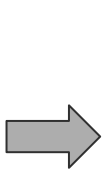
ไก่



เต่า

องค์ประกอบพื้นฐาน

Class



Employee

Accounting

Object

IT

Sale

องค์ประกอบพื้นฐาน

- คุณสมบัติ (Property) สิ่งที่บ่งบอกลักษณะทั่วไปของวัตถุ
- พฤติกรรม (Method) คือ พฤติกรรมทั่วไปของวัตถุที่สามารถกระทำได้

Animal

คุณสมบัติ (Property)

ชื่อ : ช้าง

สี : ฟ้าอ่อน

ประเภท : สัตว์บก

น้ำหนัก : 6 ตัน

จำนวนเท้า : 4 เท้า

พฤติกรรม (Method/Behavior)

- วิ่ง
- นอน
- ส่งเสียงร้อง

Animal

คุณสมบัติ (Property)

ชื่อ : นก

สี : เหลือง

ประเภท : สัตว์ปีก

น้ำหนัก : 0.8 กิโลกรัม

จำนวนเท้า : 2 เท้า

พฤติกรรม (Method/Behavior)

- บิน
- เดิน
- ส่งเสียงร้อง

สรุปการเขียนโปรแกรมเชิงวัตถุ

- Class – ต้นแบบของวัตถุ
- Object – สิ่งที่ถูกสร้างขึ้นมาจาก Class ประกอบด้วย
    - คุณสมบัติ (Property)
    - พฤติกรรม (Method)
- คุณสมบัติของการเขียนโปรแกรมเชิงวัตถุ
    - การห่อหุ้ม (Encapsulation)
    - การสืบทอด (Inheritance)
    - การพ้องรูป (Polymorphism)

การสร้าง Class & Object

การสร้าง Class

```
class class_name{

    Property & Method

}
```

```
class Employee{

    Property & Method

}
```

คุณสมบัติของคลาส

- คลาสเริ่มต้นในภาษา Kotlin จะมีรูปแบบเป็น final class หมายถึง คลาสดังกล่าวไม่สามารถสืบทอดหรือมีคลาสลูกได้

- ถ้าต้องการให้สามารถสืบทอดและสามารถแทนที่คุณสมบัติต่างๆภายในคลาสได้ จะต้องระบุ open กำกับที่คลาสเสมอ

การสร้าง Object (Instance)

โครงสร้างคำสั่ง

```
val obj_name = class_name()
```

ตัวอย่าง

```
val empl = Employee()
```

กฎการตั้งชื่อ

1. ชื่อ Class ควรกำหนดให้ตัวอักษรตัวแรกเป็นตัวพิมพ์ใหญ่ ที่เหลือเป็นพิมพ์เล็ก เช่น MyClass , User เป็นต้น

2. ชื่อ Object กำหนดเป็นตัวพิมพ์เล็กทั้งหมด

3. Property กำหนดเป็นตัวพิมพ์เล็ก เช่น name , age เป็นต้น

การสร้าง Property

Property คือส่วนที่ใช้ในการเก็บข้อมูลภายในคลาส มีลักษณะคล้ายกับตัวแปรคือ สามารถสร้างโดยใช้คำสั่ง var หรือ val โดยมีคุณสมบัติในการใช้งานดังนี้

- สร้างเพื่อกำหนดข้อมูลเริ่มต้นภายในคลาส
- กรณีที่ไม่ได้กำหนดข้อมูลเริ่มต้น สามารถกำหนดผ่าน Constructor ได้

การสร้าง Property

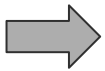
```
class className {
        var/val propertyName : type = ค่าเริ่มต้น;
}
```

- propertyName:type? //nullable (ยังไม่กำหนดค่าเริ่มต้น หรือกำหนดค่าเป็น null ได้)

การสร้าง Property

Class

Employee



ชื่อ : เจน
เงินเดือน : xxxx

Object

ชื่อ : โจโจ้
เงินเดือน : xxxx

ชื่อ : ก้อง
เงินเดือน : xxxx

การเข้าถึงข้อมูล

การเข้าถึงข้อมูลใน Property จะดำเนินการผ่านวัตถุของคลาสโดยใช้งานร่วมกับเครื่องหมายจุด

ขั้นตอนการใช้งาน

- obj.propertyName //ดึงข้อมูลจาก property
- obj.propertyName = value //กำหนดข้อมูลให้ property

การห่อหุ้ม (Encapsulation)

- เป็นกระบวนการซ่อนรายละเอียดการทำงานและข้อมูลไว้ภายในไม่ให้ภายนอกสามารถมองเห็นได้ ส่งผลให้ภายนอกนั้นไม่สามารถทำการเปลี่ยนแปลงแก้ไขข้อมูลภายในได้

- ข้อดีของการห่อหุ้มคือสามารถสร้างความปลอดภัยให้แก่ข้อมูลได้ เนื่องจากข้อมูลจะถูกเข้าถึงจากผู้มีสิทธิ์เท่านั้น

Access Modifier

คือ ระดับในการเข้าถึง Class, Property, Method และอื่นๆ ในภาษาเชิงวัตถุ มีประโยชน์อย่างมากในเรื่องของการกำหนดสิทธิในการเข้าใช้งาน การซ่อนข้อมูล เป็นต้น

Access Modifier

- Public เป็นการประกาศระดับการเข้าถึงในรูปแบบสาธารณะหรือกล่าวได้ว่าใครๆ ก็สามารถเข้าถึงและเรียกใช้งานได้
- Protected เป็นการประกาศระดับการเข้าถึงที่เกี่ยวข้องกับเรื่องการสืบทอด (Inheritance) ทำให้คลาสนั้นๆ สามารถเรียกใช้งานสมาชิกของคลาสที่ถูกกำหนดเป็น Protected ได้
- Internal เป็นการประกาศระดับการเข้าถึงในระดับ Module
- Private เป็นการประกาศระดับการเข้าถึงที่เข้มงวดที่สุด กล่าวคือจะมีเฉพาะคลาสของตัวเองเท่านั้นที่มีสิทธิ์ใช้งานได้

Access Modifier

โครงสร้างคำสั่ง

```
class className {
    modifier var/val propertyName:type
}
```

Backing Field

คำสั่งที่ช่วยให้สามารถจัดการ Property ได้ง่ายมากยิ่งขึ้น สามารถกำหนดได้ว่าต้องการอยากทำงานกับ Property ใด โดยมีองค์ประกอบ 3 ส่วน คือ

- get คือ สำหรับเรียกดูข้อมูลใน Property
- set คือ สำหรับกำหนดหรือเขียนข้อมูลใน Property (var)
- field คือ ตัวแปรพิเศษใช้กำหนดค่าให้กับ Property

โครงสร้างคำสั่ง

```
var/val propertyname:type = default_value

    get() = field //read-only (val หรือ var)

    set (value){ //var

        field=value

    }
```

การสร้าง Method

โครงสร้างคำสั่ง

```
fun method_name (parameter:type ,.....) : type {
    // statement
}
```

การเรียกใช้งาน

```
obj.method_name(arguments)
```

Constructor

เป็นกระบวนการจัดเตรียมข้อมูลให้กับคลาสเมื่อมีการสร้างวัตถุโดยจะทำงานอัตโนมัติเพียงครั้งเดียวในตอนเริ่มต้น

ประเภทของ Constructor

- Primary Constructor (Class Header)
- Secondary Constructor (Constructor Method)

Primary Constructor

- รับข้อมูลเข้ามาทำงานภายในคลาสและเก็บข้อมูลลงใน Property โดยเขียนต่อท้ายชื่อคลาสในพื้นที่วงเล็บ () พร้อมระบุ var/val กำกับที่ชื่อ Property
- ข้อมูลที่รับเข้ามานั้นจะถูกเก็บลงใน Property โดยตรง ไม่จำเป็นต้องสร้าง Property ขึ้นมาเก็บข้อมูลอีก
- กรณีที่ใช้คำสั่ง var จะเปลี่ยนค่าใน Property ได้แต่ถ้าใช้คำสั่ง val จะเปลี่ยนค่าไม่ได้

Primary Constructor

โครงสร้างคำสั่ง

```
class class_name(var/val propertyname:type){



}
```

init Block

เป็นการกำหนดขอบเขตการทำงานในตอนเริ่มต้นของ Primary Constructor

```
init{
// body primary constructor
// กรณีที่มี Secondary Constructor คำสั่งใน init block จะทำงานก่อนเสมอ
}
```

Secondary Constructor

- ไม่ระบุการรับข้อมูลผ่าน Class Header
- สร้างเมธอดพิเศษที่ชื่อว่า constructor() เพื่อรับข้อมูลหรือพารามิเตอร์เข้ามาทำงานในคลาส (ไม่ต้องใส่ var / val กำกับ)
- เมธอด constructor() จะถูกเรียกใช้งานอัตโนมัติและทำงานครั้งเดียวในตอนเริ่มต้นเมื่อมีการสร้างวัตถุจากคลาส
- เมธอด constructor() ไม่สามารถ return ค่าออกมาใช้งานได้
- ใช้งานร่วมกับคำสั่ง this ในการอ้างอิงชื่อ Property

Secondary Constructor

โครงสร้างคำสั่ง

```
class class_name{
    constructor([parameter]){
    }
}
```

Companion Object

คือ การเข้าถึง Propery หรือ Method ภายในคลาสที่สามารถเรียกใช้งานได้โดยตรง โดยไม่ต้องเรียกผ่าน วัตถุ (static ในภาษา Java)

โครงสร้างคำสั่ง

```
companion object{
    // propertyname , methodname
}
```

การเรียกใช้งาน

```
classNameObject.propertyname
classNameObject.method_name(arguments)
```

การสืบทอดคุณสมบัติ (Inheritance)

คือ การสร้างสิ่งใหม่ขึ้นด้วยการสืบทอด หรือ รับเอา (inherit) คุณสมบัติบางอย่างจากสิ่งเดิมที่มีอยู่แล้ว นำกลับมาใช้งานใหม่ (re-use)

ข้อดีของการสืบทอดทำให้ช่วยประหยัดเวลาการทำงานลง เนื่องจากไม่ต้องเสียเวลาพัฒนาระบบขึ้นมาใหม่เองทั้งหมด

การสืบทอดคุณสมบัติ (Inheritance)

- คลาสที่สามารถสืบทอดได้จะระบุคำสั่ง open กำกับไว้
- คลาสต้นแบบ หรือ คลาสแม่เรียกว่า BaseClass หรือ SuperClass
- คลาสลูกคือคลาสที่สืบทอดจากคลาสอื่นเรียกว่า Derived Class หรือ SubClass
- การสืบทอดคุณสมบัติจะใช้เครื่องหมาย : ร่วมกับ ()

คุณสมบัติต่างๆจากคลาสแม่จะถูกถ่ายทอดไปยังคลาสลูก

Class

การสืบทอดคุณสมบัติจะสืบทอดได้เพียงคลาสเดียวเท่านั้น

Employee

Accounting

IT

Sale

Employee

IT

Sale

- ชื่อ
- เงินเดือน

Employee

IT

Sale

- ชื่อ
- เงินเดือน

Employee

- ชื่อ
- เงินเดือน

IT

- ชื่อ
- เงินเดือน

Sale

การสืบทอดคุณสมบัติ (Inheritance)

คลาสแม่

```
open class BaseClass{


// Property & Method
}
```

คลาสลูก

```
class DerivedClass : BaseClass(){


// Property & Method
}
```

การสืบทอดคุณสมบัติ (Inheritance)

คลาสแม่

```
open class Employee{


// Property & Method
}
```

คลาสลูก

```
class Sale : Employee(){


// Property & Method
}
```

Super คืออะไร

เป็นคำสั่งสำหรับเรียกใช้งานเมื่อต้องการคุณสมบัติต่างๆที่ทำงานอยู่ในคลาสแม่ เช่น Property , Method , Constructor

กรณีที่ทำงานกับ Constructor จะระบุในกรณีที่คลาสลูกมีการสร้าง Secondary Constructor เราสามารถสร้าง Constructor เพิ่มเติมให้คลาสลูกได้และสามารถนำ Constructor ของคลาสแม่มาใช้งานได้เช่นเดียวกัน

- ชื่อ
- เงินเดือน

Employee

- ชื่อ
- เงินเดือน
- ประสบการณ์การทำงาน

IT

- ชื่อ
- เงินเดือน
- พื้นที่รับผิดชอบ

Sale

- ชื่อ
- เงินเดือน

Employee

- ชื่อ
- เงินเดือน
- ประสบการณ์การทำงาน

IT

- ชื่อ
- เงินเดือน
- พื้นที่รับผิดชอบ

Sale

การพ้องรูป (Polymorphism)

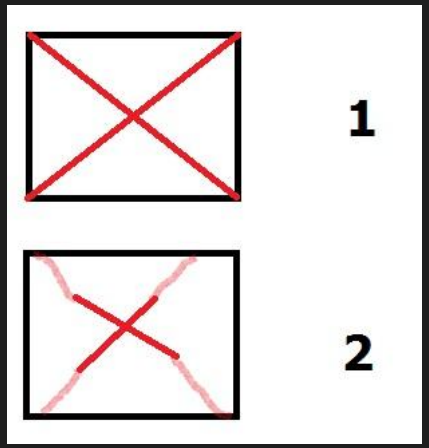
คือ ความสามารถในการตอบสนองต่อสิ่งเดียวกันด้วยวิธีการที่แตกต่างกัน กล่าวคือวัตถุนั้นสามารถกำหนดกระบวนการทำงานได้หลายรูปแบบโดยเพิ่มเติมกระบวนการทำงานจากสิ่งเดิมที่มีอยู่แล้ว

- ข้อดีคือทำให้โปรแกรมสามารถปรับเปลี่ยนหรือเพิ่มเติมการทำงานได้ง่ายขึ้น

การพ้องรูป (Polymorphism)

“ ข้อความเดียวกันแต่กระบวนการทำงานภายในแตกต่างกันนั้น เรียกว่า การพ้องรูป หรือ polymorphism ”

กา



ดำ

Override Property

คือ Property ของคลาสลูกที่มีชื่อเหมือนกับ Property คลาสแม่ (เป็นผลมาจากการสืบทอด) แต่มีกระบวนการทำงานด้านในแตกต่างกัน

Override Property

- คลาสแม่ต้องระบุ open ด้านหน้าชื่อ Property ที่สามารถ override ได้
- คลาสลูกต้องระบุ override พร้อมกำหนดชื่อและชนิดข้อมูล Property เหมือนกับคลาสแม่
- สามารถใช้คำสั่ง super อ้างอิง Property ของคลาสแม่ได้

Override Method

คือ เมธอดของคลาสลูก ที่มีชื่อเหมือนกับเมธอดของคลาสแม่ (เป็นผลมาจากการสืบทอด) แต่มีกระบวนการทำงานด้านในแตกต่างกัน

Override Method

- คลาสแม่ต้องระบุ open ด้านหน้าชื่อ Method ที่สามารถ override ได้

- คลาสลูกต้องระบุ override พร้อมกำหนดชื่อเมธอดและพารามิเตอร์เหมือนกับเมธอดในคลาสแม่

Abstract Class

คือคลาสอีกประเภทหนึ่งที่ไม่มีการกำหนดวิธีการทำงานด้านในหากคลาสใดสืบทอดคุณสมบัติจาก Abstract Class คลาสนั้นจะต้องทำการระบุวิธีการต่างๆภายใน Class นั้นไว้เสมอ (override)

จุดประสงค์เพื่อให้ผู้ใช้งานคลาสไปกำหนดวิธีการทำงานเพิ่มเติมในภายหลัง (ไม่ต้องเติม open นำหน้า Abstract Class)

Abstract Class

```
abstract class class_name{

    //สมาชิกใน Abstract Class

    //Abstract Property

    //Abstract Method

}
```

Abstract Class

```
abstract class class_name{

    abstract  var/val property_name : type

    abstract fun method_name()

    abstract fun method_name():type

}
```

Abstract Class

- คีย์เวิร์ด abstract ใช้งานร่วมกับ class , property , method สำหรับกำหนดโครงสร้างโดยไม่ระบุรายละเอียดการทำงานด้านใน (คลาสแบบ abstract จะไม่สามารถนำไปสร้าง Instance ได้)
- กฎของ Abstract คือ หากคลาสใดสืบทอดมาจาก Abstract Class คลาสนั้นจะต้องทำการระบุสมาชิกที่เป็น Abstract ทั้งหมด (Property , Method) ใน Abstract Class ไว้เสมอ (override)

การสืบทอด Abstract Class

```
class subclass : abstract_class_name(){

    override var/val property_name : type = value

    override fun method_name(){

    }

}
```

ตัวอย่าง

- แต่ละแผนกต้องระบุบทบาทหรือหน้าที่ (Role) ในบริษัท

- แต่ละแผนกต้องรายงานผลการดำเนินงาน เช่น
  - ฝ่ายไอที (รายงานการพัฒนาและดูแลระบบในบริษัท)
  - ฝ่ายขาย (รายงานการยอดขายสินค้า)

อินเทอร์เฟซ (Interface)

มีหลักการคล้ายกับ Abstract Class คือ สร้างขึ้นมาเพื่อกำหนดโครงสร้างการทำงานที่จำเป็นต้องใช้แต่ยังไม่ได้กำหนดรายละเอียดการทำงานใดๆ ลงไป

ข้อจำกัดของคลาสสามารถสืบทอดได้เพียงคลาสเดียวเท่านั้น ถ้าต้องการกำหนดคุณสมบัติเพิ่มเติมต้องอาศัยการ implements สิ่งที่เรียกว่า Interface

อินเทอร์เฟซ (Interface)

```
interface name{
    var/val property_name : type //abstract property
    fun method_name() //abstract method
}
```

- สมาชิกใน interface ทั้ง Property และ Method จะเป็น Abstract ทั้งหมด โดยไม่ต้องระบุ abstract นำหน้า

การใช้งาน Interface

```
class class_name : baseClass , interface_name{
    //สมาชิกใน Interface
    //Abstract Property
    //Abstract Method
}
```

Interface กับ Abstract Class ต่างกันอย่างไร

- คลาสที่จะเรียกใช้งานสมาชิกที่เป็น abstract ใน abstract class จะต้องสืบทอดคุณสมบัติจาก abstract class นั้นแล้วจึงทำการสร้าง Property หรือ Method ของตัวเองขึ้นมาให้มีชื่อเดียวกันกับใน abstract class ที่เป็นคลาสแม่โดยกำหนดรายละเอียดการทำงานให้กับสมาชิกเหล่านั้นตามต้องการ
- คลาสที่จะเรียกใช้งานสมาชิกในอินเทอร์เฟซไม่จำเป็นต้องมีความสัมพันธ์ใดๆ กับอินเทอร์เฟซทั้งสิ้น

Interface กับ Abstract Class ต่างกันอย่างไร

- Property ใน abstract class เป็น Property แบบปกติหรือ abstract Property ก็ได้ แต่ทุก Property ใน interface จะเป็น abstract Property ทั้งหมด
- เมธอดใน abstract class เป็นเมธอดปกติ หรือ abstract method ก็ได้ แต่เมธอดทุกเมธอดใน interface จะเป็น abstract method