บทที่ 2
หลักการเขียนโปรแกรมเชิงอ็อบเจกต์

การเขียนโปรแกรมเชิงอ็อบเจกต์จะมีลักษณะเฉพาะที่แตกต่างจากการเขียนโปรแกรมเชิงโครงสร้างทั้งทางด้านแนวคิด การออกแบบ รวมถึงโครงสร้างสำหรับการเขียนโปรแกรม การศึกษาทฤษฎีเชิงอ็อบเจกต์ และหลักการในเขียนโปรแกรมเชิงอ็อบเจกต์ให้เข้าใจอย่างถ่องแท้ จะทำให้สามารถเขียนโปรแกรมเชิงอ็อบเจกต์ได้อย่างมีประสิทธิภาพมากขึ้น

การเขียนโปรแกรมเชิงอ็อบเจกต์ (Object Oriented Programming: OOP)

การเขียนโปรแกรมเชิงอ็อบเจกต์ (Object Oriented Programming: OOP) จะมีการแบ่งขอบเขตของงานออกเป็นส่วนย่อย ๆ ที่เป็นอิสระต่อกันโดยมองส่วนต่าง ๆ เป็นวัตถุหรืออ็อบเจกต์ที่ไม่ขึ้นต่อกัน แต่มีการทำงานร่วมกัน ซึ่งวัตถุในมุมมองของการเขียนโปรแกรมเชิงอ็อบเจกต์อาจเป็นวัตถุที่เป็นรูปธรรม เช่น นาฬิกา รถยนต์ หนังสือ สุนัข เป็นต้น หรือวัตถุที่เป็นนามธรรม เช่น เวลา รายวิชา ตัวละคร เป็นต้น ข้อดีของการพัฒนาโปรแกรมโดยใช้หลักการเชิงอ็อบเจกต์ มีดังนี้

1. พัฒนาโปรแกรมได้อย่างรวดเร็ว (Rapid development) สนับสนุนการพัฒนาโปรแกรมเป็นทีม โดยการแบ่งแต่ละโมดูลในการพัฒนาโปรแกรมมีความเป็นอิสระต่อกัน และสามารถนำมารวมเข้าด้วยกันได้ง่าย
2. การนำกลับมาใช้ใหม่ (Re-usability) บนสภาพแวดล้อมของระบบเดิมหรือระบบใหม่สามารถทำได้ง่าย และประหยัดเวลาในการปรับปรุง แก้ไข
3. ดูแลรักษาได้ง่าย (Maintainability) การปรับปรุง แก้ไขโปรแกรมสามารถทำได้ง่าย
4. ประหยัดงบประมาณ (Low-cost development)

ความแตกต่างระหว่างการเขียนโปรแกรมเชิงโครงสร้างกับการเขียนโปรแกรมเชิงอ็อบเจกต์

การเขียนโปรแกรมเชิงโครงสร้าง (Structure Programming) หรือการเขียนโปรแกรมแบบกระบวนการ (Procedural Programming) เป็นการเขียนโปรแกรมสำหรับการพัฒนาระบบงานสารสนเทศซึ่งเน้นในเรื่องหน้าที่การทำงาน (Function) ให้ความสำคัญกับขั้นตอนหรือกระบวนการที่ทำ มีการแยกส่วนของข้อมูลจากฟังก์ชันอย่างชัดเจน จึงส่งผลให้เกิดความสับสนและความยุ่งยากในการเปลี่ยนแปลงแก้ไขโดยเฉพาะโปรแกรมขนาดใหญ่

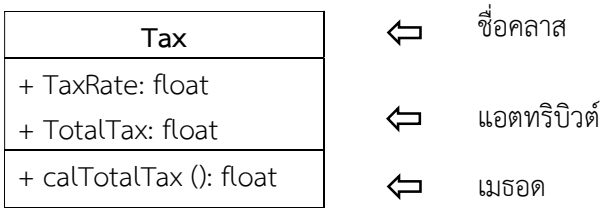
ในขณะที่การเขียนโปรแกรมเชิงอ็อบเจกต์มีแนวคิดที่ต้องการลดความซับซ้อนของโปรแกรม จึงแบ่งระบบงานให้เป็นส่วนย่อย ๆ ที่มีอิสระต่อกันโดยเก็บข้อมูลต่าง ๆ รวมกันเพื่อให้ง่ายต่อการเรียกใช้งาน ง่ายต่อการแก้ไข (Modify) และง่ายต่อการนำกลับมาใช้ใหม่ (Reuse) ซึ่งสามารถแสดงความแตกต่างระหว่างการเขียนโปรแกรมภาษาเชิงโครงสร้างและการเขียนโปรแกรมภาษาเชิงอ็อบเจกต์ ได้ดังนี้

ภาษาเชิงโครงสร้าง	ภาษาเชิงอ็อบเจกต์
1. การออกแบบแบบ Top-down	1. การออกแบบแบบ Bottom-up
2. เน้นขั้นตอนกระบวนการ	2. เน้นการมองเป็นอ็อบเจกต์
3. ใช้ข้อมูลร่วมกัน	3. ซ่อนรายละเอียดข้อมูล (Information hiding)
4. แยกข้อมูลออกจากฟังก์ชัน	4. ห่อหุ้มข้อมูล (Encapsulation)

คลาส (Class) และอ็อบเจกต์ (Object)

การทำความเข้าใจกับการเขียนโปรแกรมเชิงอ็อบเจกต์แบบง่าย ๆ ให้คิดถึงการสร้างบ้านที่ต้องเขียนแบบพิมพ์เขียวก่อนสร้างเสมอ ซึ่งแบบพิมพ์เขียว 1 แบบ สามารถนำไปสร้างบ้านเป็น 10 เป็น 100 หลัง หรือจะกล่าวว่าสร้างเท่าไรก็ได้ไม่จำกัด จากแนวคิดการสร้างบ้านเมื่อเปรียบกับแนวคิดของการเขียนโปรแกรมเชิงอ็อบเจกต์ ก็จะได้ว่าแบบพิมพ์เขียวก็คือ คลาส บ้านแต่ละหลังที่สร้างจากแบบพิมพ์เขียวก็คือ อ็อบเจกต์ หมายความว่าเราออกแบบสร้างคลาสเพียงครั้งเดียวก็สามารถนำคลาสไปสร้างอ็อบเจกต์เพื่อใช้งานได้ไม่จำกัด และหากเปรียบบ้าน 1 หลังเป็นโปรแกรม 1 โปรแกรม ก็จะได้ว่าบ้านหนึ่งหลังประกอบด้วยประตู หน้าต่าง ห้องนอน ห้องน้ำ ฯลฯ เช่นเดียวกับโปรแกรมหนึ่งโปรแกรมที่ประกอบด้วยอ็อบเจกต์หลาย ๆ อ็อบเจกต์นั่นเอง โดยแต่ละอ็อบเจกต์จะมีหน้าที่แตกต่างกันออกไป (อ็อบเจกต์แต่ละอ็อบเจกต์ก็เปรียบได้กับประตู หน้าต่าง ห้องนอน ห้องน้ำ นั่นเอง)

คลาส คือสิ่งที่ใช้อธิบายลักษณะและความสามารถของอ็อบเจกต์ เปรียบได้กับแม่แบบของอ็อบเจกต์ โดยสามารถสร้างอ็อบเจกต์ให้มีคุณลักษณะที่เหมือนกับแม่แบบกี่ครั้งก็ได้ อ็อบเจกต์ คือสิ่งต่าง ๆ ที่ปรากฏอยู่รอบตัวซึ่งมีคุณลักษณะและความสามารถในการทำงาน เช่น คน, รถยนต์, เครื่องคอมพิวเตอร์ เป็นต้น โดยสิ่งที่ใช้อธิบายลักษณะของอ็อบเจกต์จะถูกเรียกว่า แอตทริบิวต์ (Attribute) และสิ่งที่ใช้อธิบายการทำงานของอ็อบเจกต์ในอ็อบเจกต์จะถูกเรียกว่า เมธอด (Method) เช่น ให้ภาษีเป็นคลาสแล้วคุณลักษณะของภาษีคือ อัตราภาษีและจำนวนเงินภาษีที่ต้องจ่าย ส่วนการทำงานก็คือการคำนวณจำนวนเงินภาษีที่ต้องจ่าย จึงสามารถแสดงส่วนประกอบของคลาสได้ดังรูปที่ 1



รูปที่ 1 ตัวอย่างแสดงส่วนประกอบของคลาส

จากรูปที่ 1 มีคลาสชื่อ Tax มีแอตทริบิวต์ชื่อ TaxRate และ TotalTax และมีเมธอดชื่อ calTotalTax ซึ่งสามารถอธิบายในรูปแบบภาษาจาวาได้ดังนี้

```
class Tax {
    public float TaxRate;
    public float TotalTax;
    public float calTotalTax () {
        float test = 0;
        return test;
    }
}
```

คุณสมบัติของภาษาเชิงอ็อบเจกต์

หากจะบอกว่าภาษาเขียนโปรแกรมภาษานั้นเป็นภาษาเชิงอ็อบเจกต์หรือไม่ เราสามารถตรวจสอบได้จากคุณสมบัติของภาษาซึ่งภาษาที่เป็นภาษาเชิงอ็อบเจกต์จะต้องมีคุณสมบัติต่อไปนี้

1. Abstraction เป็นคุณลักษณะสร้างความคิดรวบยอดกับวัตถุใด ๆ โดยการสร้างคลาสและรายละเอียดการทำงาน ผู้ใช้ไม่ต้องไปกังวลใจว่าคลาสนั้นมีรายละเอียดการทำงานเป็นอย่างไร ขอให้เรียกใช้งานให้ถูกต้องผ่านแอตทริบิวต์และเมธอดได้ก็พอ เช่น คลาส Tax มีรายละเอียดอย่างไรไม่ต้องสนใจขอให้เรียกใช้งานอ็อบเจกต์ที่สร้างจากคลาส Tax ได้ก็พอ
2. Encapsulation เป็นกลไกที่ช่วยซ่อนสิ่งที่ไม่ต้องการให้เข้าถึงได้จากภายนอกของคลาส นั่นคือใครก็ตามจะเรียกใช้งานคลาสก็ต้องสร้างเป็นอ็อบเจกต์ แล้วจึงใช้งานแอตทริบิวต์และเมธอดผ่านอ็อบเจกต์นั้น ไม่มีสิทธิ์ดัดแปลงแก้ไขรายละเอียดภายในคลาส กระบวนการซ่อนข้อมูลหรือที่เรียกว่า Information hiding นี้สามารถกำหนดให้มีระดับการเข้าใช้งานที่แตกต่างกันได้โดยใช้ Access Modifier ซึ่งจะได้กล่าวถึงอีกครั้งต่อไป
3. Inheritance เป็นคุณสมบัติที่ว่าคลาสสามารถสืบทอดแอตทริบิวต์และเมธอดจากคลาสแม่ไปสู่คลาสลูกได้ คลาสที่ถูกสืบทอดมานั้นจะมีแอตทริบิวต์และเมธอดเหมือนกับคลาสแม่ อีกทั้งยังสามารถพัฒนาต่อโดยเพิ่มเติมและแก้ไขแอตทริบิวต์และเมธอดได้ตามความเหมาะสมอีกด้วย
4. Polymorphism เป็นคุณสมบัติที่ว่าความสามารถของคลาสมีได้หลายรูปแบบ การระบุชนิดและจำนวนพารามิเตอร์ที่ใช้ในการทำงานของเมธอดแตกต่างกัน ทำให้อ็อบเจกต์มีวิธีการทำงานแตกต่างกันด้วย เช่น การระบุจำนวนพารามิเตอร์ให้ MessageBox ต่างกัน ทำให้การแสดงผลของ MessageBox แตกต่างกันด้วย

ข้อดีของการเขียนโปรแกรมเชิงอ็อบเจกต์

1. สามารถปรับปรุงแก้ไขได้ง่าย: เราจะมองสิ่งต่าง ๆ เป็นอ็อบเจกต์โดยที่แต่ละอ็อบเจกต์มีความเป็นอิสระจากกันทำให้เราสามารถปรับปรุงหรือแก้ไขส่วนใดส่วนหนึ่งได้โดยไม่มีผลกระทบกับส่วนอื่น ๆ ของโปรแกรม
2. ใช้งานง่าย: การใช้งานอ็อบเจกต์แต่ละครั้งไม่ต้องเสียเวลาสร้างใหม่ทุกครั้ง เราสามารถใช้งานอ็อบเจกต์ต่าง ๆ ที่มีอยู่ได้ทันที
3. สามารถพัฒนาเพิ่มเติมได้ง่าย: เราสามารถนำคลาสที่มีอยู่มาพัฒนาเพิ่มเติมความสามารถได้โดยไม่จำเป็นต้องสร้างใหม่ทั้งหมด

4. ซ่อนรายละเอียดของโปรแกรม: เราสามารถซ่อนรายละเอียดที่ต้องการได้โดยที่ผู้เรียกใช้งานคลาสนั้นไม่จำเป็นต้องทราบรายละเอียดเหล่านั้น

การสร้างคลาส

ในการเขียนโปรแกรม การสร้างคลาสขึ้นมาก็เหมือนกับการเขียนพิมพ์เขียวสร้างบ้านซึ่งก็คือคลาสเป็นเพียงรูปแบบที่ได้ออกแบบไว้เท่านั้น จะมีประโยชน์เมื่อมีการนำคลาสนั้นไปสร้างอ็อบเจกต์ (พิมพ์เขียวเป็นเพียงแปลนบ้านที่ออกแบบไว้จะมีประโยชน์เมื่อมีการนำแปลนนั้นไปสร้างเป็นบ้าน)

ดังนั้น เพื่อความเข้าใจเราจะทดลองสร้างอ็อบเจกต์จากคลาสที่สร้างขึ้นเองในที่นี้จะทดลองสร้างคลาสที่ชื่อ Student ซึ่งเป็นคลาสนักเรียนโดยมีขั้นตอนดังนี้

1. การประกาศคลาส
2. การประกาศแอตทริบิวต์
3. การประกาศเมธอด

1. การประกาศคลาส

การประกาศคลาสเป็นขั้นตอนแรกของการสร้างคลาสเพื่อให้โปรแกรมรู้จักกับคลาสนั่นเอง ซึ่งมีรูปแบบการประกาศดังนี้

```
[modifier] class ClassName {
    [attributeName]
    [methodName]
}
```

โดยที่	modifier	เป็นคีย์เวิร์ดที่กำหนดคุณสมบัติการเข้าถึงคลาส
	ClassName	เป็นชื่อคลาส
	attributeName	เป็นส่วนของการประกาศแอตทริบิวต์
	methodName	เป็นส่วนของการประกาศเมธอด

เราสามารถสร้างคลาสด้วยโปรแกรม NetBeans โดยการสร้างคลาสจากการสร้างโปรเจ็กต์ใหม่ หรือสร้างคลาสจากโปรเจ็กต์ที่มีอยู่เดิมก็ได้

2. การประกาศแอตทริบิวต์

เมื่อประกาศคลาสเรียบร้อยแล้วต่อไปเป็นการประกาศแอตทริบิวต์ซึ่งเป็นตัวแปรหรือค่าคงที่ที่ประกาศไว้ภายในคลาส จึงเป็นสมาชิกของคลาสนั้น ๆ โดยมีรูปแบบการประกาศดังนี้

```
[modifier] datatype attributeName;
```

โดยที่	modifier	เป็นคีย์เวิร์ดที่กำหนดคุณสมบัติการเข้าถึงแอตทริบิวต์
	datatype	เป็นชนิดข้อมูลของแอตทริบิวต์ซึ่งอาจเป็นชนิดข้อมูลพื้นฐานหรือเป็นคลาสก็ได้ในชื่อแอตทริบิวต์จากคลาสก็ได้
	attributeName	เป็นชื่อแอตทริบิวต์

จากคลาส Student ที่สร้างขึ้นนี้เป็นคลาสของนักเรียน กำหนดให้นักเรียนมีข้อมูลชื่อนักเรียนเป็นชนิดข้อความ และอายุเป็นชนิดเลขจำนวนเต็ม ซึ่งเราสามารถประกาศแอตทริบิวต์เพิ่มเข้าไปในคลาส Student ได้ดังนี้

```
public class Student {
   String name; // ข้อมูลชื่อนักเรียนมีชนิดเป็น string
   int age; // ของมูลอายุมีชนิดเป็น int
}
```

3. การประกาศเมธอด

เมื่อประกาศแอตทริบิวต์เรียบร้อยแล้วต่อไปเป็นการประกาศเมธอดซึ่งเป็นการทำงานต่าง ๆ ของคลาสที่ถูกสร้างขึ้นเพื่อทำงานอย่างใดอย่างหนึ่งภายในคลาสจึงเป็นสมาชิกของคลาสนั้น ๆ โดยมีรูปแบบการประกาศดังนี้

```
[modifier] return_type methodName ([parameter]) {
[method_body}
return Value:
 }
```

โดยที่	modifier	เป็นคีย์เวิร์ดที่กำหนดคุณสมบัติการเข้าถึงเมธอด
	return_type	เป็นชนิดของข้อมูลที่เมธอดจะส่งค่ากลับในกรณีที่ไม่มีการส่งค่ากลับให้กำหนดเป็น void
	methodName	เป็นชื่อเมธอด
	parameter	เป็นตัวแปรที่ใช้ในการรับข้อมูล
	method_body	ข้อมูลเป็นชุดคำสั่งการทำงานของเมธอด
	Value	เป็นค่าที่ต้องการส่งค่ากลับในกรณีที่กำหนดให้ return_type เป็น void จะไม่มีคำสั่ง return

จากแอตทริบิวต์ที่ประกาศให้เป็นสมาชิกของคลาส Student เราจะลองสร้างเมธอดที่ทำหน้าที่กำหนดชื่อและอายุของนักเรียน (setData) และสร้างเมธอดที่ทำหน้าที่แสดงชื่อและอายุของนักเรียน (getData) โดยเมธอดทั้งสองนี้จะไม่มีการคืนค่าคือไม่มีในส่วนของ return_type จึงกำหนดเป็น void ซึ่งมีโค้ดดังนี้

```
public class Student {
    String name; // ข้อมูลชื่อนักเรียน มีชนิดเป็น string
    int age;       // ข้อมูลอายุ 4 มีชนิดเป็น int
    public void setData () {
       name = "JAVA";
       age = 20;
    }
    public void getData () {
        System.out.println ("Name :" + name);
        System.out.println ("Age :"+ age);}
    }
}
```

การใช้งานคลาส

เมื่อสร้างคลาสพร้อมกับการกำหนดสมาชิกของคลาสเป็นที่เรียบร้อยแล้วเราสามารถนำคลาสไปใช้งานได้โดยการประกาศอ็อบเจกต์และใช้งานผ่านสมาชิกของคลาสได้

การประกาศอ็อบเจกต์

การประกาศอ็อบเจกต์ก็คือการสร้างอ็อบเจกต์จากคลาสที่สร้างไว้นั่นเองซึ่งในภาษา Java มีรูปแบบการประกาศอ็อบเจกต์ดังนี้

```
ClassName objectName;
```

โดยที่ ClassName เป็นชื่อของคลาสที่ใช้สร้างอ็อบเจกต์นั้น
objectName เป็นชื่ออ็อบเจกต์ที่ประกาศใช้งานและสามารถสร้างอ็อบเจกต์ได้ตามรูปแบบดังนี้

```
objectName = new ClassName ();
```

โดยที่ ClassName เป็นชื่อของคลาสที่ใช้สร้างอ็อบเจกต์นั้น
objectName เป็นชื่ออ็อบเจกต์ที่ประกาศใช้งาน

หรือในบางครั้งเราอาจจะประกาศและสร้างอ็อบเจกต์พร้อมกันเป็นคำสั่งเดียวกันก็ได้โดยมีรูปแบบดังนี้

```
ClassName objectName = new ClassName ():
```

โดยที่ ClassName เป็นชื่อของคลาสที่ใช้สร้างอ็อบเจกต์นั้น
objectName เป็นชื่ออ็อบเจกต์ที่ประกาศใช้งาน

การเข้าถึงสมาชิกของคลาส

เมื่อประกาศและสร้างอ็อบเจกต์จากคลาสใด ๆ แล้วอ็อบเจกต์นั้นจะสามารถเรียกใช้หรือเข้าถึงสมาชิกของคลาสได้โดยมีรูปแบบดังนี้

objectName.AttributeName;

โดยที่ objectName เป็นชื่ออ็อบเจกต์

attributeName เป็นชื่อแอตทริบิวต์ที่ต้องการใช้งานกรณีเข้าถึงเพื่อเรียกใช้เมธอด สามารถทำได้ 2 รูปแบบคือ

1. กรณีที่เมธอดไม่มีการคืนค่า มีรูปแบบการเรียกใช้ดังนี้

   objectName.methodName ([argument]);

โดยที่ objectName เป็นชื่ออ็อบเจกต์

methodName เป็นชื่อเมธอดที่ต้องการใช้งาน

argument เป็นค่าที่ต้องการส่งผ่านไปให้เมธอดที่ต้องการใช้งาน

2. กรณีที่เมธอดมีการคืนค่า มีรูปแบบการเรียกใช้ดังนี้

   datatype methodValue = objectName.methodName ([argument]);

โดยที่ objectName เป็นชื่ออ็อบเจกต์เป็นชื่อเมธอด

methodName เป็นชื่อเมธอดที่ต้องการใช้งานที่ต้องการส่งผ่าน

argument เป็นค่าที่ต้องการส่งผ่านไปให้เมธอดที่ต้องการใช้งาน

datatype เป็นชนิดข้อมูลของเมธอดที่มีการคืนค่า

methodValue เป็นตัวแปรที่ใช้เก็บค่าที่ได้จากการคืนค่าของเมธอด

จากคลาส Student ที่มีการกำหนดแอตทริบิวต์ชื่อนักเรียนและอายุอีกทั้งมีการสร้างเมธอด setData() และ getData() ไว้ ต่อไปนี้จะเป็นการใช้งานคลาสโดยมีการประกาศอ็อบเจกต์และใช้งานผ่านสมาชิกของคลาส ซึ่งมีขั้นตอนดังนี้

1. สร้างคลาส StudentTest เป็น Java Main Class เพื่อทดสอบการเรียกใช้คลาส Student โดยคลิกขวาที่ Introduction แล้วเลือกคำสั่ง New => Java Main Class.
2. ในส่วนของ Class Name: กรอกชื่อคลาสว่า StudenTest
3. คลิกปุ่ม Finish
4. จะได้คลาส StudentTest ดังนี้

```
public class StudentTest {
    public static void main (String [] args) {
    }
}
```

5. เพิ่มส่วนของการประกาศอ็อบเจกต์และการเข้าถึงสมาชิกของคลาสดังนี้

```
public class StudentTest {
    public static void main(String [] args) {
        Student std = new Student();
        std.setData();
        std.getData();
        std.age = 15;
        std.getData();
    }
}
```

จากตัวอย่างโปรแกรม อธิบายได้ดังนี้

บรรทัดที่ 3 ประกาศ std เป็นอ็อบเจกต์ที่สร้างขึ้นจากคลาส Student

บรรทัดที่ 4 เรียกใช้เมธอด setDate() ซึ่งเป็นเมธอดกำหนดค่าข้อมูลชื่อและอายุของนักเรียนให้เท่ากับ JAVA และ 20 ปี ตามลำดับเนื่องจากกำหนดไว้ในการประกาศเมธอด

บรรทัดที่ 5 เรียกใช้เมธอด getData() ซึ่งเป็นเมธอดแสดงผลข้อมูลชื่อและอายุของนักเรียน

บรรทัดที่ 6 กำหนดค่าแอตทริบิวต์ age เท่ากับ 15

บรรทัดที่ 7 เรียกใช้เมธอด getData() เพื่อแสดงผลอีกครั้ง

เมื่อรันโปรแกรม จะได้ผลการทำงานของโปรแกรมดังนี้

```
run:
    Name: JAVA
    Age: 20
    Name: JAVA
    Age: 15

BUILD SUCCESSFUL (total time: 0 seconds)
```

Modifier ในภาษา Java

จากตัวอย่างโปรแกรมการสร้างคลาสเมื่อพิจารณาการใช้งาน modifier พบว่าเป็นการใช้ modifier แบบ public ทั้งหมด ความหมายของ modifier ก็คือคีย์เวิร์ดของภาษา Java ที่ใช้กำหนดคุณสมบัติของคลาสซึ่งภาษา Java มีรูปแบบ modifier สำหรับใช้งานแตกต่างกันมากมายโดยสามารถแบ่งออกเป็นประเภทหลัก ๆ ได้ 2 ประเภทคือ Non access modifier และ Access modifier

Non access modifier

Non access modifier คือคีย์เวิร์ดที่ใช้กำหนดคุณสมบัติอื่น ๆ ที่ไม่ใช่ระดับการเข้าถึงข้อมูลของคลาสหนึ่ง ๆ เช่น

static ใช้เป็นคีย์เวิร์ดสำหรับกำหนดให้แอตทริบิวต์หรือเมธอดนั้น ๆ เป็นแบบ static

แอตทริบิวต์ที่เป็น static ควรมีคุณสมบัติดังนี้

1. แอตทริบิวต์นั้นถูกสร้างขึ้นครั้งแรกเพียงครั้งเดียว
2. แอตทริบิวต์นั้นถูกเรียกใช้บ่อยและถูกเรียกใช้จากหลายออบเจ็บต์

เมธอดที่เป็นแบบ static จะมีคุณสมบัติดังนี้

1. เมธอดนั้นถูกเรียกใช้งานได้ตลอดเวลาที่อยู่ภายในขอบเขตของคลาส
2. เมธอดนั้นถูกเรียกใช้จากคลาสอื่นผ่านชื่อคลาสได้โดยไม่ต้องใช้อ็อบเจกต์

final ใช้เป็นคีย์เวิร์ดสำหรับกำหนดให้แอตทริบิวต์หรือคลาสนั้น ๆ เป็นแบบ final modifier ซึ่งมีคุณสมบัติดังนี้

1. แอตทริบิวต์นั้นจะไม่สามารถเปลี่ยนแปลงค่าได้ตลอดการทำงานของโปรแกรม
2. คลาสนั้นจะไม่อนุญาตให้คลาสอื่นมาสืบทอดคุณสมบัติได้

Access modifier

Access Modifier คือ คีย์เวิร์ดที่ใช้กำหนดระดับการเข้าถึงข้อมูล เป็นกลไกของการกำหนดระดับการเข้าใช้งานสมาชิกของคลาสเพื่อรักษาความปลอดภัยและป้องกันการเปลี่ยนแปลงข้อมูลภายในคลาสซึ่งอาจเกิดขึ้นได้ในขณะที่มีการนำคลาสดังกล่าวไปใช้งานโดยภาษา Java มีการแบ่งระดับของ Access modifier เป็น 4 ระดับคือ

1. public มีขอบเขตการเข้าใช้งานได้จากทุกคลาส
2. private มีขอบเขตการเข้าใช้งานข้อมูลภายในคลาสเดียวกันเท่านั้น
3. protected มีขอบเขตการเข้าใช้งานข้อมูลภายในคลาสเดียวกันและคลาสที่สืบทอดกัน
4. package มีขอบเขตการเข้าใช้งานข้อมูลภายในคลาสเดียวกันและคลาสอื่นที่อยู่ภายในแพ็กเกจเดียวกัน

ในการกำหนดระดับของ Access modifier ให้กับข้อมูลใด ๆ มีแนวคิดในการพิจารณาดังนี้

1. กรณีที่ข้อมูลทำหน้าที่เป็นตัวแปรสาธารณะอนุญาตให้คลาสใด ๆ เรียกใช้ได้โดยตรงให้กำหนดเป็นแบบ public
2. กรณีที่ข้อมูลมีค่าเป็นส่วนตัวไม่อนุญาตให้คลาสอื่นมาเปลี่ยนแปลงค่าได้ให้กำหนดเป็นแบบ private
3. กรณีที่ข้อมูลมีความสัมพันธ์ในลักษณะที่มีการสืบทอดจากคลาสแม่สู่คลาสลูกให้กำหนดเป็นแบบ protected
4. กรณีที่เป็นข้อมูลทั่วไปการไม่กำหนดระดับของ Access modifier ภาษา Java จะถือว่ามีระดับการใช้งานเป็นแบบ package ซึ่งมีการป้องกันระดับการเข้าใช้งาน ดังนี้
    1. ข้อมูลสามารถถูกเรียกใช้จากคลาสที่อยู่ภายในแพ็กเกจเดียวกัน
    2. ถ้าต้องการใช้ข้อมูลจากคลาสต่างแพ็กเกจกันจะต้องกำหนดเป็นแบบ public
    3. ใช้ในกรณีที่ไม่ต้องการให้ตัวแปรเป็นสาธารณะและไม่ต้องการให้เป็นตัวแปรที่มีค่าส่วนตัวสามารถจำกัดสิทธิ์การใช้งานภายในแพ็กเกจเดียวกัน

ระดับการใช้งานแบบ public

public เป็นการกำหนดระดับ Access modifier ให้กับแอตทริบิวต์หรือเมธอดที่ต้องการให้คลาสใด ๆ เรียกใช้งานก็ได้ สามารถศึกษาการกำหนดระดับ Access modifier เป็น public ได้จากตัวอย่างต่อไปนี้

```
public class PublicEmp {
    public float salary = 15000.0f;
}
```

คลาส PublicEmp มีแอตทริบิวต์ salary เป็นแบบ public เมื่อคอมไพล์โปรแกรมแล้วจะเกิดคลาส PublicEmp ซึ่งคลาสอื่นสามารถเข้าถึงข้อมูล salary ในคลาส PublicEmp ได้ดังตัวอย่างต่อไปนี้

```
public class PublicAcMod {
    public static void main (String [] args) {
        PublicEmp emp = new PublicEmp();
        System.out.println ("Current Salary is " + emp.salary);
        emp. salary = 24000.0f;
        System.out.println (" New Salary is " + emp. salary);
    }
}
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานได้ดังนี้

บรรทัดที่ 3 ประกาศอ็อบเจกต์ emp จากคลาส public_emp
บรรทัดที่ 4-5 แสดงค่าข้อมูล salary ซึ่งสามารถเข้าถึงโดยผ่านอ็อบเจกต์ emp
บรรทัดที่ 6 เปลี่ยนแปลงค่าข้อมูล salary ผ่านอ็อบเจกต์ emp
บรรทัดที่ 7-8 แสดงค่าข้อมูล salary ผ่านอ็อบเจกต์ emp อีกครั้ง

เมื่อรันโปรแกรมจะได้ผลการทำงานดังนี้

```
run:
    Current Salary is 15000.0
    New Salary Is 24000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

จะพบว่าในบรรทัดที่ 5 ข้อมูล salary ถูกเปลี่ยนแปลงค่าได้ทำให้เกิดความไม่ปลอดภัยกับข้อมูล salary ดังนั้นการกำหนดระดับ Access modifier ให้ข้อมูล salary จึงควรเป็นแบบ private เพื่อเป็นการซ่อนข้อมูลไว้ไม่ให้คลาสอื่นสามารถเปลี่ยนแปลงแก้ไขค่าได้โดยตรง

ระดับการใช้งานแบบ private

private เป็นการกำหนดระดับ Access modifier ให้กับแอตทริบิวต์หรือเมธอดที่ต้องการใช้งานภายในคลาสเท่านั้น ไม่อนุญาตให้คลาสอื่นเรียกใช้งาน สามารถศึกษาการกำหนดระดับ Access modifier เป็น private ได้จากตัวอย่างต่อไปนี้

```
public class PrivateEmp {
    private float salary = 15000.0f;
    public float getSalary() {
        return salary;
    }
    public float setSalary(float s) {
        salary = s;
        return salary;
    }
}
```

คลาส PrivateEmp มีแอตทริบิวต์ salary เป็นแบบ private และมีเมธอด getSalary() และ setSalary() เป็นแบบ public เมื่อคอมไพล์โปรแกรมจะเกิดคลาส PrivateEmp ซึ่งคลาสอื่นไม่สามารถเข้าถึงข้อมูล salary ได้เนื่องจากมีระดับ Access modifier เป็นแบบ private ทำให้ข้อมูล salary ถูกซ่อนไว้เพื่อป้องกันไม่ให้ถูกเปลี่ยนแปลงค่าได้โดยตรง

ถ้าต้องการเข้าถึงข้อมูล salary เราสามารถทำผ่านเมธอด getSalary() และ setSalary() ซึ่งมีระดับ Access modifier เป็นแบบ public ได้ดังตัวอย่างต่อไปนี้

```
public class PrivateAcMod {
    public static void main(String [] args) {
        PrivateEmp emp = new PrivateEmp ();
        System.out.println ("Current Salary is " + emp.getSalary());
        System.out.println ("New Salary is " + emp.setSalary (23000));
    }
}
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานได้ดังนี้

บรรทัดที่ 3 ประกาศอ็อบเจกต์ emp จากคลาส PrivateEmp

บรรทัดที่ 4-5 แสดงค่าข้อมูล salary ซึ่งสามารถเข้าถึงได้โดยเรียกใช้เมธอด getSalary ผ่านอ็อบเจกต์ emp เนื่องจากไม่สามารถเข้าถึงข้อมูล salary ได้โดยตรง

บรรทัดที่ 6-7 ที่เปลี่ยนแปลงค่าข้อมูล salary โดยเรียกใช้เมธอด setSalary ผ่านอ็อบเจกต์ emp

เมื่อรันโปรแกรมจะได้ผลการทำงานดังนี้

```
run:
    Current Salary is 15000.0
    New Salary is 23000.0

BUILD SUCCESSFUL (total time: 0 seconds)
```

จะพบว่าในข้อมูล salary ถูกปกป้องและซ่อนข้อมูลไว้ด้วยระดับ Access modifier แบบ private คลาสอื่นที่ต้องการดำเนินการใด ๆ กับข้อมูล salary จะต้องทำผ่านเมธอดที่มีระดับ Access modifier แบบ public เท่านั้น

ระดับการใช้งานแบบ protected

protected เป็นการกำหนดระดับ Access modifier ให้แอตทริบิวต์หรือเมธอดที่อนุญาตให้คลาสที่มีความสัมพันธ์จากคลาสแม่สู่คลาสลูกใช้งานเท่านั้นไม่อนุญาตให้คลาสอื่นเรียกใช้ได้ สามารถศึกษาการกำหนดระดับ Access modifier เป็น protected ได้จากตัวอย่างต่อไปนี้

```
public class ProtectedEmp {
    protected float salary = 15000.0f;
}
```

คลาส ProtectedEmp มีแอตทริบิวต์ salary แบบ protected เมื่อคอมไพล์โปรแกรมแล้วจะเกิดคลาส ProtectedEmp ซึ่งคลาสที่สามารถเข้าถึงข้อมูล salary จะต้องมีความสัมพันธ์เป็นคลาสลูกของคลาส ProtectedEmp เท่านั้น ดังตัวอย่างต่อไปนี้

```
class AcMod extends ProtectedEmp {
    void setSalary () {
        System.out.println ("Current Salary is " + salary);
        salary = 22000.0f;
        System.out.println ("New Salary is "  + salary);
    }
}
public class protected AcMod {
    public static void main (String [] args) {
        AcMod emp = new AcMod ();
        emp.setSalary();
        }
}
```

จากตัวอย่างโปรแกรม สามารถอธิบายการทำงานได้ดังนี้

บรรทัดที่ 1 สร้างคลาส AcMod ให้มีความสัมพันธ์เป็นคลาสลูกของคลาส ProtectedEmp

บรรทัดที่ 2-9 สร้างเมธอด setSalary() ภายในคลาส AcMod ซึ่งสามารถเข้าถึงเปลี่ยนแปลง และแสดงผลค่าข้อมูล salary ในคลาสแม่ได้ โดยไม่ต้องผ่านอ็อบเจกต์

บรรทัดที่ 12 ประกาศอ็อบเจกต์ emp จากคลาส AcMod

บรรทัดที่ 13 เรียกใช้เมธอด setSalary ผ่านอ็อบเจกต์ emp

เมื่อรันโปรแกรมจะได้ผลการทำงานดังนี้

```
run:
   Current Salary is 15000.0
   New Salary is 22000.0

BUILD SUCCESSFUL (total time: 0 seconds)
```

ระดับการใช้งานแบบ package

package เป็นการกำหนดระดับ Access modifier ให้กับแอตทริบิวต์หรือเมธอดที่ต้องการใช้งานภายในคลาสเดียวกันหรือคลาสที่อยู่ในแพ็กเกจเดียวกันและไม่สามารถเรียกใช้จากคลาสที่อยู่ต่างแพ็กเกจกันได้ โดยคลาสที่อยู่ในแพ็กเกจเดียวกันหมายถึงคลาสที่จัดเก็บอยู่ในโฟลเดอร์เดียวกันผู้สามารถศึกษาการกำหนดระดับ Access modifier เป็น package ได้จากตัวอย่างต่อไปนี้

```
public class PackageEmp {
   double salary = 15000.0f:
}
```

คลาส PackageEmp มีแอตทริบิวต์ salary แบบไม่กำหนดระดับ Access modifier ซึ่งลักษณะเช่นนี้จะถือว่าเป็นการกำหนดระดับ Access modifier เป็น package เมื่อคอมไพล์โปรแกรมแล้วจะเกิดคลาส PackageEmp ซึ่งคลาสอื่น ๆ ที่สามารถเข้าถึงข้อมูล salary ได้จะต้องเป็นคลาสที่อยู่ในโฟลเดอร์เดียวกันกับคลาส PackageEmp เท่านั้น ดังตัวอย่างต่อไปนี้

```
public class PackageAcMod {
   public static void main(String [] args) {
      PackageEmp emp = new PackageEmp();
      System.out.println ("Current Salary is " + emp.salary);
      emp.salary = 21000.0f;
      System.out.println ("New Salary is " + emp.salary);
   }
}
```

จากตัวอย่างโปรแกรมสามารถอธิบายการทำงานได้ดังนี้

บรรทัดที่ 3 ประกาศอ็อบเจกต์ emp จากคลาส PackageEmp

บรรทัดที่ 4-5 แสดงค่าข้อมูล salary ซึ่งสามารถเข้าถึงได้โดยผ่านอ็อบเจกต์ emp

บรรทัดที่ 6 เปลี่ยนแปลงค่าข้อมูล salary ผ่านอ็อบเจกต์ emp

บรรทัดที่ 7-8 แสดงค่าข้อมูล salary ผ่านอ็อบเจกต์ emp อีกครั้ง

เมื่อรันโปรแกรมจะได้ผลการทำงานดังนี้

```
run:
  Current Salary is 15000.0
  New Salary is 21000.0

BUILD SUCCESSFUL (total time: 0 seconds)
```

จะพบว่าข้อมูล salary ถูกเปลี่ยนแปลงค่าได้อาจจะเกิดความไม่ปลอดภัยกับข้อมูล salary แต่ในความเป็นจริงแล้วการเปลี่ยนแปลงค่านี้จะถูกจำกัดไว้ให้เฉพาะคลาสที่อยู่ในแพ็กเกจเดียวกัน คลาสอื่นที่อยู่ต่างแพ็กเกจกันจะไม่สามารถเข้าถึงข้อมูล salary ได้

แพ็กเกจและการใช้งาน

เมื่อมีการสร้างคลาสจำนวนมากไว้ใช้งานเราจึงต้องจัดหมวดหมู่ของคลาสเพื่อให้เป็นระเบียบและง่ายต่อการเรียกใช้งาน แพ็กเกจจึงถูกนำมาใช้ในการจัดหมวดหมู่ของคลาส โดยแพ็กเกจเปรียบได้กับโฟลเดอร์ที่มีไว้สำหรับรวบรวมคลาสให้อยู่ด้วยกันโดยอาศัยหลักการตั้งชื่อแบบโครงสร้างต้นไม้ซึ่งมีเครื่องหมาย "." คั่น

การประกาศแพ็กเกจ

การประกาศแพ็กเกจเป็นขั้นตอนแรกของการสร้างคลาสเพื่อให้โปรแกรมรู้จักกับคลาสนั่นเอง ซึ่งมีรูปแบบการประกาศดังนี้

```
package packageName;
```

โดยที่ packageName เป็นชื่อแพ็กเกจ

ถ้าต้องการสร้างคลาส Tax ในแพ็กเกจ TaxCalculation สามารถทำได้ดังนี้

1. สร้างโปรเจ็กต์ใหม่เป็น New Java Application
2. ในส่วนของ Project Name: กรอกชื่อโปรเจกต์ว่า TaxCalculation
3. ในส่วนของ Use Dedicated Folder for Storing Libraries ไม่ต้องคลิกเครื่องหมายถูก
4. ในส่วนของ Create Main Class ไม่ต้องคลิกเครื่องหมายถูกเนื่องจากเราไม่ต้องการสร้าง Main Class
5. คลิกปุ่ม Finish
6. จะปรากฏโฟลเดอร์ชื่อ TaxCalculation
7. สร้าง New Java Class เพื่อสร้างคลาส Tax ไว้ในแพ็กเกจ TaxCalculation
8. ในส่วนของ Class Name: กรอกชื่อคลาสว่า Tax
9. ในส่วนของ Package: คลิกเลือกแพ็กเกจ TaxCalculation
10. คลิกปุ่ม Finish

11. เขียนส่วนของโปรแกรมเพิ่มเติม
12. ผลที่ได้คือจะมีคลาส Tax ปรากฏอยู่ในโฟลเดอร์ TaxCalculation ดังนี้

```
package TaxCalculation;
public class Tax {
    public float taxrate = 0.10f;
}
```

การอ้างถึงคลาสที่อยู่ในแพ็กเกจเดียวกัน

ในการอ้างถึงคลาสที่อยู่ในแพ็กเกจเดียวกันเราสามารถอ้างถึงหรือเรียกใช้ผ่านชื่อคลาสได้ สามารถศึกษาการใช้งานได้จากตัวอย่างต่อไปนี้

```
package TaxCalculation;
public class Vat {
    public static void main(String [] args) {
        Tax x = new Tax ();
        System.out.println ("tax_rate = " + x.taxrate);
    }
}
```

จากตัวอย่างโปรแกรมข้างต้นมีการสร้างคลาส Tax ไว้ในแพ็กเกจ TaxCalculation หากเราต้องการใช้งานคลาส Tax เราสามารถสร้างคลาส Vat ไว้ในแพ็กเกจเดียวกับคลาส Tax คือแพ็กเกจ TaxCalculation เพื่อใช้งานคลาส Tax ได้ดังนี้

1. สร้าง New Java Main Class เพื่อสร้างโปรแกรมหลัก Vat ไว้ในแพ็กเกจ TaxCalculation
2. ในส่วนของ Class Name: กรอกชื่อคลาสว่า Vat
3. ในส่วนของ Package: ให้คลิกเลือกแพ็กเกจ TaxCalculation
4. คลิกปุ่ม Finish
5. เขียนส่วนของโปรแกรมเพิ่มเติม
6. ผลที่ได้คือจะมีคลาส Vat ปรากฏอยู่ในโฟลเดอร์ TaxCalculation

เมื่อรันโปรแกรมจะได้ผลการทำงานของ vat1.java ดังรูป

```
run:
  tax rate = 0.1

BUILD SUCCESSFUL (total time: 0 seconds)
```

การอ้างถึงคลาสจากแพ็กเกจอื่น

การอ้างถึงคลาสจากแพ็กเกจอื่นในภาษา Java จะต้อง import คลาสในแพ็กเกจที่ต้องการก่อนจึงจะสามารถอ้างถึงหรือเรียกใช้ผ่านชื่อคลาสได้

หากเราต้องการนำคลาส Tax ซึ่งอยู่ในแพ็กเกจ TaxCalculation มาใช้งานในคลาสที่อยู่ต่างแพ็กเกจกันเราสามารถสร้างคลาส Vat2 ไว้ในแพ็กเกจ VatCalculation และเรียกใช้คลาส Tax ได้ดังนี้

1. สร้าง New Java Class เพื่อสร้างโปรแกรมหลัก Vat2 ไว้ในแพ็กเกจ VatCalculation
2. ในส่วนของ Class Name: กรอกชื่อคลาสว่า Vat2
3. ในส่วนของ Package: ให้คลิกเลือกแพ็กเกจ VatCalculation
4. คลิกปุ่ม Finish
5. เขียนส่วนของโปรแกรมเพิ่มเติม

```
package VatCalculation;
import TaxCalculation.Tax;
public class Vat2 {
    public static void main(String [] args) {
        Tax x = new Tax ();
        System.out.println ("tax rate =" + x.taxrate);
    }
}
```

เมื่อรันโปรแกรมจะได้ผลการทำงานของ vat2.java ดังรูป

```
run:
  tax rate = 0.1

BUILD SUCCESSFUL (total time: 0 seconds)
```

บทสรุป

ในบทนี้ทำให้ได้รู้จักกับโปรแกรมเชิงอ็อบเจกต์ที่จะต้องประกอบด้วยคลาส ซึ่งมีสมาชิกคือแอตทริบิวต์และเมธอด ได้รู้ถึงวิธีการประกาศอ็อบเจกต์ การเข้าถึงและใช้งานสมาชิกของคลาส การเรียกใช้สมาชิกที่เป็นเมธอดซึ่งถือว่าเป็นบทที่มีความสำคัญเพราะเป็นหัวใจของการเขียนโปรแกรมเชิงอ็อบเจกต์

แบบฝึกหัดท้ายบท

นักศึกษาจงตอบคำถามต่อไปนี้

1. จงเขียนอธิบายถึงความแตกต่างระหว่างแอตทริบิวต์และเมธอด
2. จงเขียนอธิบายถึงความสำคัญและขอบเขตการใช้งานของ Access modifier แต่ละระดับ
3. จงสร้างคลาสนักเรียน (Student) ซึ่งประกอบด้วยข้อมูลชื่อ (name), คะแนนสอบ score) โดยมีเมธอดรับข้อมูลชื่อและคะแนนสอบของนักเรียน (getStdData) และเมธอดแสดงผลชื่อและคะแนนสอบของนักเรียน (showStdData)
4. จงสร้างคลาสพนักงาน (Employee) เพื่อเรียกใช้เมธอดคำนวณค่าแรงชื่อ calWage) ที่สร้างขึ้นในคลาสค่าแรง (Wage) โดยการคำนวณค่าแรงคิดจากจำนวนชั่วโมงทำงาน x ค่าแรงต่อชั่วโมง และกำหนดให้ระดับการใช้งานเมธอดเป็นแบบ package
5. จงสร้างคลาสพนักงาน (Employee) ในข้อ 4 ให้อยู่ในแพ็กเกจชื่อ myclass และสร้างคลาสค่าแรง (Wage) ให้อยู่ในแพ็กเกจชื่อ mainclass โดยมีวิธีการคำนวณเหมือนข้อ 4 แต่ให้ระดับการใช้งานเมธอดเป็นแบบ public

เอกสารอ้างอิง

กิตติ ภักดีวัฒนะกุล, และศิริวรรณ อัมพรดนัย. (2544). Object-Oriented ฉบับพื้นฐาน. กรุงเทพฯ : เคทีพี คอมพ์ แอนด์ คอนซัลท์.

รุ่งโรจน์ โพนคา, และปราลี มณีรัตน์. (2545). Java Programming. กรุงเทพฯ: ซัคเซส มีเดีย บจก.

วรรณิกา เนตรงาม. (2545). คู่มือการเขียนโปรแกรมภาษาจาวา ฉบับผู้เริ่มต้น. นนทบุรี: อินโฟเพรส บจก.

วีระศักดิ์ ซึงถาวร. (2543). Java Programming Volume I. กรุงเทพฯ: ซีเอ็ดยูเคชั่น บจก.

วีระศักดิ์ ซึงถาวร. (2545). Java Programming Volume II. กรุงเทพฯ: ซีเอ็ดยูเคชั่น บจก.

สุดา เธียรมนตรี. (2556). คู่มือเรียนเขียนโปรแกรมภาษา Java ฉบับสมบูรณ์ (2nd Edition). นนทบุรี : ไอดีซี พรีเมียร์ บจก.

สุรางคนา ระวังยศ. (2555). การเขียนโปรแกรมเชิงวัตถุ. ค้นเมื่อ 23 สิงหาคม 2560, จาก http://www.kmitl.ac.th/~s3010819/MyFile/My%20Ebook/JAVA/%BA%B7%B7%D5%E8%201%20%20%BA%B7%B9%D3.pdf.

About the java technology. (2012). Retrieved August 28, 2016, from: http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Anban Pillay. (2007). Object Oriented Programming using Java. University of KwaZuluNatal.

Armstrong, E. (2002). The J2EE™ 1.4 Tutorial. Network Circle, CA: Sun Microsystem, Inc.

Java download (2016). Retrieved August 28, 2015, from: http://java.sun.com/javase/downloads/index.jsp