# แนวคิดเชิงวัตถุ : Object – Oriented Concepts

สุดา เธียรมนตรี
อาจารย์ประจำภาควิชาคณิตศาสตร์
คณะวิทยาศาสตร์ มหาวิทยาลัยทักษิณ

## บทนำ

แนวคิดเชิงวัตถุมาจากคำภาษาอังกฤษว่า Object – Oriented Concepts โดย Object เป็นคำนาม แปลว่าวัตถุทั้งที่สัมผัสได้และสัมผัสไม่ได้ ส่วนคำว่า Oriented เป็นคำกริยาที่แปลว่า นำทาง หรือนำไป เมื่อรวมความหมายกันแล้ว Object – Oriented Concepts หมายถึงแนวคิดที่มีการใช้วัตถุเพื่อการนำทางในการกำหนดหรืออธิบายสิ่งต่าง ๆ ที่เกิดขึ้น

แนวคิดหรือเทคโนโลยีเชิงวัตถุเกิดขึ้นเมื่อประมาณช่วงปี 1970 และถูกนำมาใช้เพื่อการพัฒนาซอฟต์แวร์กันอย่างแพร่หลายในปัจจุบัน ด้วยหลักการของวัตถุที่ประกอบด้วยกลุ่มของข้อมูลที่เรียกว่า property และ กลุ่มของ method เช่น รถเป็นวัตถุมี property คือ ข้อมูลชื่อรถ รุ่นรถ จำนวนล้อ หรือ จำนวนประตู ในขณะที่ method ได้แก่ การกำหนดความเร็วรถ การคำนวณระยะทางที่รถวิ่งได้ เป็นต้น หรือตัวอย่างของคน จะมี property คือ ข้อมูลชื่อคน อายุ ชื่อตำแหน่งหรือ ประวัติการขาดงาน ใน ขณะที่ method ได้แก่ การกำหนดอัตราเงินเดือน การคำนวณเงินเดือนที่ได้รับ เป็นต้น จากตัวอย่างจะพบว่าแต่ละวัตถุมีกลุ่มข้อมูลและหน้าที่การทำงานเฉพาะอย่างและแตกต่างกันไป ทำให้แนวคิดเชิงวัตถุเป็นที่นิยมกันอย่างกว้างขวางในวงการอุตสาหกรรมซอฟต์แวร์และเริ่มเข้ามามีบทบาทต่อการพัฒนาระบบที่ทำงานผ่านเว็บ (Web Application) มากขึ้น ดังนั้นจึงหลีกเลี่ยงไม่ได้ที่จะต้องศึกษาทำความเข้าใจในหลักการของแนวคิดเชิงวัตถุ เพื่อสามารถพัฒนางานด้วยหลักการเชิงวัตถุได้อย่างถูกต้อง และไม่เป็นปัญหาสำหรับคนที่คุ้นเคยกับการเขียนโปรแกรมแบบเดิม (Procedural Programming) ซึ่งมีแนวคิดที่แตกต่างจากไปจากแนวคิดเชิงวัตถุ คือมีการมองที่กระบวนการเป็นหลักและแบ่งกระบวนการแยกย่อยไปเรื่อยๆ เพื่อให้แต่ละกระบวนการทำงานหลักเพียงอย่างเดียว มีการแบ่งการทำงานอย่างเป็นขั้นตอนและมีลำดับที่ชัดเจน แต่อาจจะเกิดความสลับซับซ้อนขึ้นในกรณีที่โปรแกรมมีขนาดใหญ่และมีกระบวนการทำงานค่อนข้างมาก ทำให้เป็นปัญหาสำหรับการนำไปใช้เพื่อการพัฒนาต่อไป หากผู้พัฒนาไม่เข้าใจในการทำงานของโปรแกรมนั้นอย่างแท้จริง ในขณะที่แนวคิดเชิงวัตถุมุ่งเน้นการพัฒนาระบบหรือโปรแกรมให้เป็นส่วนๆ ที่เป็นอิสระแยกจากกัน ทำให้ลดปัญหาความซับซ้อนของโปรแกรม และส่วนของโปรแกรม

ที่พัฒนาแล้วสามารถนำกลับมาใช้เพื่อการพัฒนาระบบงานอื่นได้อีก โดยแนวคิดที่ควรศึกษาทำความเข้าใจ คือ 1) Encapsulation 2) Inheritance และ 3) Polymorphism

**Encapsulation**

Encapsulation เป็นการห่อหุ้มหรือปกปิดข้อมูลเพื่อป้องกันการแก้ไขข้อมูลโดยตรง ซึ่งการแก้ไขข้อมูลโดยตรง หมายถึงว่า ถ้าต้องการแก้ไขเมื่อไร ก็สามารถกำหนดค่าใหม่ให้กับข้อมูลที่บรรทัดใด ๆ ในตัวโปรแกรมได้เลย

Encapsulation เกิดขึ้นได้จากการสร้างกลุ่มของ property และ method รวมเข้าด้วยกันไว้ในคลาส (Class) ซึ่งเป็นกรอบกำหนดคุณสมบัติของวัตถุ โดยวัตถุจะเกิดหรือถูกสร้างขึ้นภายใต้คลาสที่มีการกำหนดกลุ่มของ property และ method แล้ว ข้อมูลที่ได้รับการป้องกันการแก้ไขโดยตรงจะ ต้องมีสถานะข้อมูลเป็น private และการแก้ไขจะทำได้โดยกระทำผ่าน method เท่านั้น

```
class car                                                    ----- (1)
{                                                            ----- (2)
private :                                                    ----- (3)
  char *name;                                                ----- (4)
  char *version;                                             ----- (5)
public:                                                      ----- (6)
  void set_cardata();                                        ----- (7)
};                                                           ----- (8)
void main()                                                  ----- (9)
{                                                            ----- (10)
   car x;                                                    ----- (11)
   x. set_cardata();                                         ----- (12)
}                                                            ----- (13)
void car::set_cardata()                                      ----- (14)
{                                                            ----- (15)
   name   = "VOLVO";                                         ----- (16)
   version  = "v380";                                        ----- (17)
   cout << "Car's name    = "    << name << "\n";           ----- (18)
   cout << " Car's version = "   << version << "\n";        ----- (19)
}                                                            ----- (20)
```

โปรแกรมที่ 1 แสดงการกำหนดข้อมูลของ name และ version ผ่าน set_cardata()

จากตัวอย่างโปรแกรมที่ 1 ซึ่งเป็นการเขียนโปรแกรมด้วยภาษา C++ ในบรรทัดที่ 1 – 8 มีการกำหนด property คือ name และ version ให้มีสถานะเป็น private การกำหนดสถานะเป็น private ทำให้เกิดคุณสมบัติที่เรียกว่า Information Hiding ซึ่งเป็นการจำกัดการใช้งานของกลุ่มข้อมูลให้อยู่ภายในวัตถุของคลาส car เท่านั้น ส่วน method คือ set_cardata() ได้ถูกกำหนดสถานะให้เป็น public เพื่อให้สามารถติดต่อและรับส่งข้อมูลกับวัตถุอื่นได้ การรวมกลุ่มของ property และ method เข้าด้วยกันนั้นเป็นคุณสมบัติที่เรียกว่า Encapsulation ในบรรทัดที่ 11 เป็นการสร้างวัตถุ x ให้อยู่ภายใต้คลาส car ทำให้ x มีคุณสมบัติที่ได้รับมาจาก car มี property คือ name และ version ส่วน method คือ set_cardata() ในบรรทัดที่ 12 เป็นการเรียกใช้ method คือ x.set_cardata() เพื่อกำหนดข้อมูลของ name และ version ให้กับวัตถุ x การเรียกใช้นี้อยู่ภายใต้ฟังก์ชันหลักของโปรแกรมคือ ฟังก์ชัน main() และในบรรทัด 14 – 20 เป็นการทำงานของ method คือ set_cardata() ซึ่งเป็น method ของคลาส car โดยในบรรทัดที่ 16 - 17 เป็นการกำหนดค่าข้อมูลของ name และ version โดยกระทำผ่าน method คือ set_cardata() ซึ่งไม่สามารถนำ 2 บรรทัดนี้ไปไว้ที่ตำแหน่งอื่นได้

ประโยชน์ของ Encapsulation อีกอย่างหนึ่งคือ ถ้ามีการแก้ไขเปลี่ยนแปลงกลุ่มข้อมูลภายในวัตถุจะไม่ส่งผลกระทบต่อวัตถุอื่น ทำให้การเปลี่ยนแปลงข้อมูลสามารถทำได้โดยอิสระ ไม่เกิดปัญหากับกลุ่มข้อมูลของวัตถุอื่น

**Inheritance**

โดยหลักการของการเขียนโปรแกรมแบบ Procedural Programming นั้น จะมีขั้นตอนของการรวบรวมส่วนของโปรแกรมที่มีการทำงานซ้ำ ๆ กัน และถูกเรียกใช้งานบ่อย ๆ ไว้ในรูปแบบของ Procedure หรือ Function ซึ่งทำให้สามารถนำมาใช้ซ้ำได้ (reuse) ในแนวคิดเชิงวัตถุก็มีคุณสมบัติหนึ่งที่มีลักษณะของการนำมาใช้ซ้ำได้ ซึ่งก็คือ Inheritance โดยหมายถึงการสืบทอดคุณสมบัติที่มีการสร้างขึ้นในคลาสหนึ่งซึ่งเรียกว่าคลาสแม่ (Parent class) ไปให้กับอีกคลาสหนึ่งซึ่งเรียกว่าคลาสลูก (Subclass) ทำให้คลาสลูกไม่จำเป็นต้องสร้างกลุ่มของ property และ method เอง แต่สามารถเรียกใช้ได้จากคลาสแม่ และยังสามารถสร้างกลุ่มของ property และ method เพิ่ม เพื่อใช้งานในคลาสเองได้

```
class car                                   ----- (1)
{                                           ----- (2)
protected :                                 ----- (3)
```

```
  char *name;                                                    ----- (4)
  char *version;                                                 ----- (5)
public:                                                          ----- (6)
  void set_cardata();                                            ----- (7)
};                                                               ----- (8)
class mycar : car                                                ----- (9)
{                                                                ----- (10)
private :                                                        ----- (11)
  char *color;                                                   ----- (12)
public:                                                          ----- (13)
  void set_mycardata();                                          ----- (14)
};                                                               ----- (15)
void main()                                                      ----- (16)
{                                                                ----- (17)
   car  x;                                                       ----- (18)
   mycar y;                                                      ----- (19)
  x.set_cardata();                                               ----- (20)
  y.set_mycardata();                                             ----- (21)
}                                                                ----- (22)
void car::set_cardata()                                          ----- (23)
{                                                                ----- (24)
   name   = "VOLVO";                                             ----- (25)
   version  = "v380";                                            ----- (26)
   cout << "The first car's name    = "  << name << "\n";        ----- (27)
   cout << "The first Car's version = "  << version << "\n";     ----- (28)
}                                                                ----- (29)
void mycar::set_mycardata()                                      ----- (30)
{                                                                ----- (31)
    name    = "BM";                                              ----- (32)
    version  = "vi250";                                          ----- (33)
    cout << "The second car's name   = "   << name << "\n";      ----- (34)
    cout << "The second car's version = "   << version << "\n";  ----- (35)
    color  = "SILVER";                                           ----- (36)
    cout << " The second car's color  = "   << color << "\n";    ----- (37)
}                                                                ----- (38)
```

โปรแกรมที่ 2 แสดงการสืบทอดคุณสมบัติ (inheritance) ของวัตถุ

จากตัวอย่างโปรแกรมที่ 2 ในบรรทัดที่ 1 – 8 มีการกำหนด property คือ name และ version และกำหนด method คือ set_cardata() รวมอยู่ในคลาสของ car ในบรรทัดที่ 9 – 15 มีการสร้างคลาส mycar ซึ่งเป็นคลาสลูกของคลาส car และมีการกำหนด property และ method เพิ่ม คือ color และ set_mycardata() ซึ่งหมายความว่า วัตถุของคลาส mycar สามารถเรียกใช้ name version และset_cardata() ซึ่งเป็น property และ method ของคลาส car โดย name และ version จะต้องมีสถานะเป็น protected แทนที่จะเป็น private เพื่อที่จะเข้าถึง name และ version ได้ เพราะการกำหนดสถานะเป็น protected เป็นการกำหนดการใช้งานของกลุ่มข้อมูลให้กับวัตถุของคลาสลูก โดยที่วัตถุของคลาสอื่น ไม่สามารถนำกลุ่มข้อมูลนี้ไปใช้งานได้

ในบรรทัดที่ 18 - 19 เป็นการสร้างวัตถุ x ให้อยู่ภายใต้คลาส car และสร้างวัตถุ y ภายใต้คลาส mycar ในบรรทัดที่ 21 – 22 เป็นการเรียกใช้ method คือ x.set_cardata() และ y.set_mycardata() เพื่อกำหนดข้อมูลของ name และ version ให้กับวัตถุ x และกำหนดข้อมูลของ name version และ colorให้กับวัตถุ y โดยการเรียกใช้ทั้งหมดนี้อยู่ภายใต้ฟังก์ชันหลักของโปรแกรมคือฟังก์ชัน main()ในบรรทัด 23 – 29 เป็นการทำงานของ method คือ set_cardata() คือการกำหนดค่าข้อมูลของ name และ version และในบรรทัด 30 – 38 เป็นการทำงานของ method คือ set_mycardata() คือการกำหนดค่าข้อมูลของ name version และ color โดยที่ color เป็น property ที่กำหนดเพิ่มในคลาส mycar เอง ส่วน name และ version เป็นคุณสมบัติที่สืบทอดมาจากคลาส car เรียกว่า inheritance

**Polymorphism**

Polymorphism หมายถึงการพ้องรูปหรือการมีได้หลายรูปแบบ ในที่นี้รูปแบบก็คือ method ซึ่งอนุญาตให้มี method ที่มีชื่อเหมือนกันมากกว่า 1 method ทำงานอยู่ในโปรแกรมเดียวกันได้ โดยการทำงานมี 2 ลักษณะคือ Overloading Method และ Overriding Method

Overloading Method หมายถึงการใช้ชื่อ method เหมือนกันมากกว่า 1 method แต่จะแยกความต่างได้ด้วยชนิดของข้อมูลหรือพารามิเตอร์ ด้วยหลักการที่ว่าแต่ละวัตถุมี property และ method เป็นของตนเอง การกำหนดค่าให้กับ property จะต้องกระทำผ่าน method ซึ่งในความเป็นจริงแล้วข้อมูลของวัตถุหนึ่งมีมากมายและแตกต่างกันไปตามชนิดของข้อมูล จึงเป็นสิ่งที่ถูกต้องที่ method หนึ่ง สามารถใช้ได้หลายครั้งกับชนิดข้อมูลที่ต่างกัน

```
class car                                                    ----- (1)
{                                                            ----- (2)
public:                                                      ----- (3)
  void set_cardata(char *name);                              ----- (4)
  void set_cardata(int wheel);                               ----- (5)
  void set_cardata(double mile);                             ----- (6)
};                                                           ----- (7)
void main()                                                  ----- (8)
{                                                            ----- (9)
  car  x;                                                    ----- (10)
  x.set_cardata("volvo");                                    ----- (11)
  x.set_cardata(4);                                          ----- (12)
  x.set_cardata(501.45));                                    ----- (13)
}                                                            ----- (14)
void car::set_cardata(char *n)                               ----- (15)
{                                                            ----- (16)
  cout <<  "Your car is " << n << "\n";                      ----- (17)
}                                                            ----- (18)
void car::set_cardata(int n)                                 ----- (19)
{                                                            ----- (20)
  cout <<  "Your car have " << n << " wheels \n";            ----- (21)
}                                                            ----- (22)
void car::set_cardata(double n)                              ----- (23)
{                                                            ----- (24)
  cout <<  "The mile of your car is "  << n << " miles\n";   ----- (25)
}                                                            ----- (26)
```

โปรแกรมที่ 3 แสดงการใช้คุณสมบัติของ Overloading Method

จากตัวอย่างโปรแกรมที่ 3 ในบรรทัดที่ 1 – 7 มีการกำหนด method โดยใช้ชื่อ set_cardata() เหมือนกัน 3 method ให้กับคลาส car ในบรรทัดที่ 8 – 14 เป็นการสร้างวัตถุ x ให้

อยู่ภายใต้คลาส car และมีการเรียกใช้ method คือ x.set_cardata() โดยส่งค่าข้อมูล 3 ค่า ที่มีชนิดต่างกันและในบรรทัดที่ 15 – 18 19 – 22 และ 23 - 26 เป็นการแสดงค่าข้อมูลซึ่งเป็นค่าที่รับมาจากทั้ง 3 method ที่มีชื่อเหมือน กันแต่ต่างกันตรงชนิดของข้อมูลที่รับมา method ในลักษณะเช่นนี้เรียกว่า Overloading Method

Overriding Method หมายถึงการใช้ชื่อ method ที่เหมือนกันในคลาสลูก ๆ ที่มีการสืบทอด คุณสมบัติมาจากคลาสแม่เดียวกัน ทำให้คลาสลูก ๆ มีกลุ่มของ property และ method เหมือนกัน จึงเป็นสิ่งถูกต้องที่จะใช้ชื่อ method ที่เหมือนกันในคลาสลูก ๆ ที่มีการสืบทอดคุณสมบัติมาจากคลาสแม่เดียวกัน

```
class car                                          ----- (1)
{                                                  ----- (2)
protected :                                        ----- (3)
  char *name;                                      ----- (4)
  char *version;                                   ----- (5)
public:                                            ----- (6)
  void set_cardata();                              ----- (7)
};                                                 ----- (8)
class mycar1 : car                                 ----- (9)
{                                                  ----- (10)
public:                                            ----- (11)
  void set_cardata();                              ----- (12)
};                                                 ----- (13)
class mycar2 : car                                 ----- (14)
{                                                  ----- (15)
public:                                            ----- (16)
  void set_cardata();                              ----- (17)
};                                                 ----- (18)
void main()                                        ----- (19)
{                                                  ----- (20)
  mycar1  x;                                       ----- (21)
```

```
   mycar2  y;                                                    ----- (22)
   x.set_cardata();                                             ----- (23)
   y.set_cardata();                                             ----- (24)
}                                                               ----- (25)
void mycar1::set_cardata()                                      ----- (26)
{                                                               ----- (27)
   name   = "VOLVO";                                            ----- (28)
   version  = "v380";                                           ----- (29)
   cout <<  "The first car's name is    " << name << "\n";      ----- (30)
   cout <<  "The first car's version is " << version << "\n";   ----- (31)
}                                                               ----- (32)
void mycar2::set_cardata()                                      ----- (33)
{                                                               ----- (34)
   name    = "HONDA";                                           ----- (35)
   version  = "DREAM";                                          ----- (36)
   cout <<  "The second car's name is    " << name << "\n";     ----- (37)
   cout <<  "The second car's version is " << version << "\n";  ----- (38)
}                                                               ----- (39)
```

โปรแกรมที่ 4 แสดงการใช้คุณสมบัติของ Overriding Method

จากตัวอย่างโปรแกรมที่ 4 ในบรรทัดที่ 1 – 8 มีการกำหนด property คือ name และ version และกำหนด method คือ set_cardata() รวมอยู่ในคลาสของ car ในบรรทัดที่ 9 – 13 และ 14 – 18 มีการสร้างคลาส mycar1 และ mycar2 ซึ่งเป็นคลาสลูก โดยมีคลาส car เป็นคลาสแม่ และกำหนด method ชื่อ set_cardata() ไว้ในคลาสทั้งสองซึ่งเหมือนกับชื่อ method ในคลาสแม่ และในบรรทัดที่ 26 – 32 และ 33 - 39 เป็นการทำงานแบบเดียวกันของ method ที่มีชื่อเหมือนกัน คือเป็นการกำหนดค่าให้กับ name และ version ของคลาสที่ต่างกัน method ในลักษณะเช่นนี้เรียกว่า Overriding Method

## บทสรุป

แนวคิดเชิงวัตถุเป็นสิ่งที่ได้ยินกันมานาน มีภาษาโปรแกรมมากมายที่สนับสนุนการทำงานของแนวคิดดังกล่าว แต่การพัฒนาระบบงานที่มีการใช้แนวคิดเชิงวัตถุอย่างสมบูรณ์นั้นยังมีไม่มากนัก ซึ่งสาเหตุหนึ่ง อาจจะเกิดจากการเปลี่ยนแปลงแนวคิดที่ผิดไปจากเดิมอย่างสิ้นเชิง ทำให้คนที่เคยชินกับการเขียนโปรแกรมแบบเดิมไม่อยากเสียเวลาที่จะทำความเข้าใจกับสิ่งใหม่ หรืออาจได้ลองทำความเข้าใจและมีการนำมาใช้แล้วแต่ยังไม่ประสบความสำเร็จเท่าที่ควร ไม่ว่าจะเป็นเหตุผลใดก็ตาม ด้วยประโยชน์ของแนวคิดเชิงวัตถุที่มีมากมาย และความนิยมในแนวคิดดังกล่าวที่มีมากขึ้น จึงเป็นสิ่งยืนยันว่า ควรจะศึกษาทำความเข้าใจในเรื่องของแนวคิดนี้อย่างจริงจัง และต้องให้ความสำคัญกับแนวคิด 3 ประการ คือ 1) Encapsulation 2) Inheritance และ 3) Polymorphism เพื่อนำไปสู่การพัฒนาระบบงานที่มีการใช้แนวคิดเชิงวัตถุอย่างถูกต้องสมบูรณ์

## เอกสารอ้างอิง(References)

(1) กิตติ ภักดีวัฒนะกุล และ ศิริวรรณ อัมพรดนัย. 2544. *Object – Oriented ฉบับพื้นฐาน*. บริษัท เคพีที คอมพ์ แอนด์ คอนซัลท์ จำกัด. กรุงเทพฯ.

(2) เกษมสันต์ พานิชการ. 2537. *C++ และหลักการของ OOP ฉบับเริ่มต้น*. บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน). กรุงเทพฯ.

(3) ธวัชชัย งามสันติวงศ์. 2545. *การเขียนโปรแกรมเชิงวัตถุ แนวคิดเชิงวัตถุโดยจาวาและ UML*. โรงพิมพ์ 21 เซ็นจูรี่ จำกัด. ศูนย์หนังสือ จุฬาลงกรณ์มหาวิทยาลัย จัดจำหน่าย. กรุงเทพฯ.

(4) วรเศรษฐ สุวรรณิก และ ทศพล ธนะทิพานนท์. 2549. *เขียนโปรแกรม JAVA เบื้องต้น*. บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน). กรุงเทพฯ.

(5) วิทยา สุคตบวร. 2548. *เรียนลัดการเขียนโปรแกรมเชิงวัตถุแบบ OOP (Object – Oriented Programming)*. บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน). กรุงเทพฯ.

(6) PHP5. *คู่มือเรียนภาษาซี*. บริษัท โปรวิชัน จำกัด. กรุงเทพฯ.

(7) John Lewis & William Loftus. *Java Software Solutions, foundation of programming design*. Fourth Edition (International Edition). PEARSON Addison Wesley.

(8) Robert W. Sebesta. 2002. *Concepts of Programming Languages*. Fifth Edition. Addison Wesley Publishing Company.