**การทดลองที่ 7 เรื่องคลาสและออปเจ็กต์**

**Class and Object**

|  |
| --- |
| **วัตถุประสงค์เชิงพฤติกรรม** |

1. เพื่อให้นักศึกษาสามารถอธิบายแนวคิดในการใช้งานคลาสและออปเจ็กต์ได้
2. เพื่อให้นักศึกษาใช้งานคลาสสำเร็จรูปได้
3. เพื่อให้นักศึกษาสร้างคลาสเพื่อใช้งานเองได้

|  |
| --- |
| **ความรู้เบื้องต้น : แนวคิดในการเขียนโปรแกรมเชิงวัตถุ** |

แนวคิดในการเขียนโปรแกรมเชิงวัตถุ มี 3 กระบวนการหลักๆ คือ การปกปิดข้อมูลหรือการห่อหุ้ม (Encapsulation), การสืบทอดคุณสมบัติ (Inheritance) และการพ้องรูป (Polymorphism) การสร้างคลาสที่มีความสามารถรองรับกระบวนการต่างๆ ได้นั้น จำเป็นต้องมีองค์ประกอบในคลาสตามความเหมาะสม ในการทดลองนี้ เราจะมาศึกษาและฝึกใช้งานองค์ประกอบต่างๆ ของคลาสและการนำคลาสไปสร้างวัตถุ เพื่อให้เข้าใจแนวคิดและสามารถใช้งานได้อย่างถูกต้อง

ในปี 1968 องค์การนาโต้ (NATO องค์การสนธิสัญญาแอตแลนติกเหนือ) ได้จัดสัมมนาเรื่อง Mass Produced Software Components[[1]](#footnote-0) [[2]](#footnote-1) (การผลิตชิ้นส่วนซอฟต์แวร์เป็นจำนวนมาก) ที่ประเทศเยอรมัน เพื่อต่อสู้กับวิกฤตการณ์ซอฟท์แวร์ การสัมมนานี้วางรากฐานแนวคิดการทำซอฟท์แวร์ให้เป็นชิ้นส่วนย่อย (component) ผู้นำการสัมมนาเรียกแนวคิดนี้ว่า Component Oriented Programming (ย่อว่า COP) หรือการเขียนโปรแกรมโดยเน้นการสร้างชิ้นส่วน โดย component หมายถึง Object ที่ถูกสร้างขึ้นตามข้อกำหนดจำเพาะเพื่อตอบสนองเป้าหมาย 5 ประการคือ

1. นำกลับมาใช้ใหม่ได้
2. เป็นอิสระต่อสภาพแวดล้อม
3. ทำงานร่วมกับ component อื่นได้
4. ใช้วิธี Encapsulation
5. เป็นอิสระต่อการนำไปใช้และการปรับเปลี่ยนเวอร์ชัน

ภาษา C# เป็นภาษา OOP ที่สนับสนุนหลักการ COP อย่างจริงจัง ดังนั้นนอกจากการเรียนรู้เรื่องตัวภาษาแล้ว เราควรยึดถือตามหลักการของ COP และ OOP อย่างเคร่งครัด เนื่องจากมันถูกใช้เป็นกรอบความคิดในการออกแบบภาษา C#

|  |
| --- |
| **การปกปิดข้อมูลหรือการห่อหุ้ม (Encapsulation)** |

การปกปิดข้อมูลหรือการห่อหุ้ม (Encapsulation) มีไว้เพื่อซ่อนรายละเอียดในการปฏิบัติงานของคลาส โดยคลาสที่ดีจะเปิดเผยเฉพาะส่วนที่ต้องการให้ผู้ใช้งานเห็นหรือเรียกใช้งานได้เท่านั้น แนวคิดในการห่อหุ้ม เกิดจากความต้องการนำคลาสมาสร้างวัตถุที่มีความยืดหยุ่นในการใช้งานมากที่สุด สามารถใช้งานทดแทนกันได้ ผู้ใช้ไม่จำเป็นต้องรู้กลไกการทำงานข้างใน เราเรียกกระบวนการคิดในลักษณะนี้ว่า Abstraction ซึ่งเกิดขึ้นขณะออกแบบคลาส โดยเราจะตัดส่วนที่ไม่จำเป็นทิ้งไป ให้พิจารณาตัวอย่างต่อไปนี้

|  |  |
| --- | --- |
|  | คลาส Person มีคุณสมบัติ (Properties) รวมทั้งความสามารถ (Operations หรือ methods) ต่างๆ ดังที่ปรากฏในแผนภาพ  เราจะเห็นว่าถ้า Person นี้ มาสมัครเป็นนักศึกษา จะมีคุณสมบัติหรือ ความสามารถบางอย่าง ที่ “เกิน” ความต้องการของสถาบันการศึกษาซึ่งเราสามารถ ตัดส่วนเกินเหล่านั้นออกไปได้ เรียกกระบวนการตัดส่วนเกินออกไปว่าเป็นการทำ abstraction ให้กับคลาส ซึ่งในแต่ละระบบจะมีคุณสมบัติและความสามารถของ Person ที่แตกต่างกันไปตามความต้องการหรือความเหมาะสม  **คำถาม แล้วเราจะตัด (**คุณสมบัติและความสามารถ) **อะไรทิ้งไป และคงเหลืออะไรไว้** |

|  |
| --- |
| **การสืบทอดคุณสมบัติ (Inheritance)** |

การสืบทอดคุณสมบัติ เป็นคุณลักษณะเด่นหนึ่งในสามประการของการคิดแบบ Object Oriented Programming ใช้สำหรับการเพิ่มความสามารถของคลาสที่สืบทอดคุณสมบัติ (Derived class) มาจากคลาสต้นแบบ (Base class) ตัวอย่างเช่น ยานพาหนะ (Vehicle) หมายถึงสิ่งประดิษฐ์ที่สามารถเคลื่อนที่ได้ สามารถบรรทุกผู้คน สัมภาระสิ่งของ จากต้นทางไปยังปลายทางได้ สมมติให้คุณสมบัติที่มีในยานพาหนะคือ การเคลื่อนที่และการบรรทุกสิ่งของสัมภาระหรือผู้โดยสาร เมื่อวิวัฒนาการของยานพาหนะในโลกมีมากขึ้นตามเวลาที่ผ่านไป เราจะได้เห็นยานพาหนะที่สามารถเคลื่อนที่ไปบนบก ในน้ำ ในอากาศ หรือแม้แต่ในอวกาศ ซึ่งจะมีคุณสมบัติจำนวนหนึ่งติดตัวเสมอคือ การเคลื่อนที่ และการบรรทุกสิ่งของสัมภาระหรือผู้โดยสาร สิ่งประดิษฐ์ใดที่มีคุณสมบัติสอดคล้องกับยานพาหนะ เราก็จะเรียกรวมๆ ว่า ยานพาหนะ แต่พอพิจารณารายละเอียด จะพบว่า หากเรานำยานพาหนะมาใส่ล้อ ก็จะเป็นยานพาหนะทางบก ใส่ปีกก็จะเป็นยานพาหนะทางอากาศ ใส่จรวดขับเคลื่อนก็จะเป็นยานอวกาศ เป็นต้น

|  |
| --- |
| **การพ้องรูป (Polymorphism)** |

หลักการพ้องรูป (Polymorphism) ในทาง Object Oriented Programming นั้น จะทำให้ระบบการสืบทอดคุณสมบัติมีความยืดหยุ่นในการใช้งาน ใกล้เคียงความเป็นจริงในชีวิตประจำวันมากขึ้น ซึ่งในเรื่อง Polymorphism นี้จะมี 2 ชนิด ได้แก่

1. Static polymorphism หรือ compile time polymorphism หรือ early binding เป็นการทำ overloading ของเมธอดหรือตัวดำเนินการ (Method overloading หรือ Operator overloading)
2. Dynamic polymorphism หรือ run time polymorphism หรือ late binding จะมีประเด็นหลักอยู่ 2 เรื่องคือ การกำหนดทับการกระทำ (Method overriding) และการปิดบังความสามารถ (Method hiding) ของคลาสต้นแบบ

**Method overloading** เป็นการพ้องรูปของเมธอดที่อยู่ในคลาสเดียวกันแต่มี signature ต่างกัน (signature ที่ใช้แยกแยะความแตกต่างในการทำ Method overloading ได้แก่ จำนวนของพารามิเตอร์และชนิดของพารามิเตอร์ ส่วนชนิดของการส่งค่ากลับ ไม่มีผลในการทำ Method overloading)

**Operator Overloading** เป็นการนำตัวดำเนินการมาใช้กับชนิดข้อมูลที่เรานิยามขึ้นใหม่ โดยใช้ตัวดำเนินการซ้ำกับชนิดข้อมูลอื่นๆ เช่นการนำตัวดำเนินการ + มาใช้กับข้อความ จะเป็นการเชื่อมข้อความเข้าด้วยกัน หรือใช้กับเวคเตอร์ใน 2 มิติ จะเป็นการหาผลบวกเวตเตอร์แบบหัวต่อหาง เป็นต้น

**Method Overriding** เป็นกรณีที่นักพัฒนารู้ว่าคลาสต้นแบบ (น่า) จะต้องมีการสืบทอดเพื่อนำไปใช้งานจริง จึงออกแบบให้คลาสที่สืบทอด สามารถมีวิธีการทำงานของตนเองได้ (โดยใช้เมธอดชื่อเดียวกันกับคลาสต้นแบบ) เช่น ยานพาหนะ มีการเคลื่อนที่ Move() แต่รายละเอียดของการเคลื่อนที่ทางบกคือ แล่นไปบนพื้น ทางน้ำคือแล่นไปบนผิวน้ำ ในอากาศคือการบิน เป็นต้น เราจะเห็นได้ว่า ไม่ว่าจะแล่นหรือบิน ก็คือการเคลื่อนที่ ดังนั้น ในการพัฒนาคลาสในระดับที่มีการสืบทอด จึงอนุญาตให้มีการใช้ชื่อเดียวแต่หลายวิธีการได้ เรียกว่า polymophism แบบ overriding

**Method Hiding** เป็นกรณีที่นักพัฒนาต้องการลบความสามารถบางอย่างในคลาสที่รับสืบทอด แต่โดยความเป็นจริง ถ้าสืบทอดคลาสมาแล้ว เราจะได้คุณสมบัติและความสามารติดมาทุกอย่างและไม่สามารถตัดออกไปได้ (นอกจากจะไปลบ source code ของคลาสต้นแบบ ซึ่งก็ไม่สามารถทำได้) อย่างไรก็ตาม เรามีวิธีการปิดบังความสามารถของคลาสต้นแบบ เช่น ยานพาหนะทางบก มีความสามารถในการเลี้ยว แต่รถไฟก็เป็นยานพาหนะทางบก ที่ไม่สามารถบังคับเลี้ยวได้ตามใจชอบ รถไฟจะต้องวิ่งไปตามราง ดังนั้นหากมีการใส่ความสามารถด้วยเมธอด Turn() ไว้ในคลาสต้นแบบ เราสามารถสืบทอดมาเป็นคลาสรถไฟ โดยที่เมธอด Turn() ของคลาสรถไฟจะไม่มีการกระทำใดๆ ภายในเมธอดนั้นเลย พร้อมทั้งใส่คีย์เวิร์ด new ไว้เป็น modifier เมื่อมีการใช้งานเมธอด Turn() ในคลาสรถไฟก็จะไม่มีการกระทำใดๆ เกิดขึ้น ซึ่งสอดคล้องกับความเป็นจริง

|  |
| --- |
| **ส่วนประกอบของคลาส** |

ในการสร้างคลาส จะพบว่ามีส่วนประกอบหลักๆ ดังต่อไปนี้

1. การประกาศชื่อคลาสและ body ของคลาส
2. Fields
3. Properties
4. Methods
5. Constructor (and Destructor)
6. ส่วนประกอบอื่นๆ

|  |
| --- |
| 1. **การประกาศชื่อคลาสและ body ของคลาส** |

การประกาศชื่อของคลาส มักจะประกาศให้เป็นคำนาม โดยมีรูปแบบการประกาศดังต่อไปนี้

|  |
| --- |
| [<attributes>] [<access-modifiers>] <class identifier> [:<base-class> [<,interface(s)>]]  {  class-body  } |

เมื่อ

|  |  |
| --- | --- |
| Attributes | ใช้เพื่อเพิ่ม metadata ให้กับคลาส  ไม่ต้องมีก็ได้ |
| Access modifiers | เป็นส่วนกำหนดการเข้าถึงจากภายนอกคลาส อาจเป็นค่าใดค่าหนึ่งจากต่อไปนี้  public, private, protected, internal, protected internal |
| Base-class | ใช้เพื่อระบุคลาสต้นแบบเพื่อการสืบทอดคุณสมบัติ  ถ้าไม่ระบุไว้ C# จะกำหนดให้เป็น System.object โดยอัตโนมัติ |
| Interfaces | ใช้เพื่อระบุ interfaces เพื่อการสืบทอดกฏเกณฑ์มาจาก interface จะระบุหรือไม่ก็ได้ |
| Class-body | เป็นพื้นที่เก็บ data และ method ของคลาส |

**ตัวอย่างการประกาศคลาส**

|  |  |
| --- | --- |
| 1  2  3  4 | public class Student  {  // class body  } |

|  |
| --- |
| 1. **Fields** |

Fields ใช้สำหรับเก็บข้อมูลภายในคลาส ข้อมูลใน fields สามารถประกาศเป็นชนิดข้อมูลใดก็ได้ ซึ่งถ้ายึดตามหลัก OOP แล้ว Fields ทั้งหมดจะต้องมี access-modifier เป็น private เนื่องจากถ้าเราประกาศเป็น public แล้วจะขัดกับกฏการทำ encapsulation เป็นผลให้ไม่สามารถควบคุมการเข้าถึงข้อมูลจากภายนอก เราไม่ควรประกาศ fields เป็น public ยกเว้นในกรณีที่ไม่มีเวลาหรือในความเร่งรีบ ถ้าต้องการเข้าถึง fields จากภายนอก ให้ใช้ Properties เพื่อเข้าถึงและควบคุมข้อมูลใน fields

**รูปแบบการประกาศ Field**

|  |
| --- |
| Type Identifier [= value]; // Type = data type, Identifier = field name.  // fields can be initialized by [= value]. |

**ตัวอย่างการประกาศ Fields**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public class Student  {  // Fields declaration  private string name;  private string ID;  private float GPA;  } |

|  |
| --- |
| 1. **Properties** |

Properties เป็นคุณสมบัติของคลาส ที่เปิดเผยต่อโลกภายนอก ให้เห็นถึงช่องทางที่จะเข้าถึง fields ที่อยู่ภายใน แต่ไม่จำเป็นที่ทุกๆ field จะมี property ไว้คอยรองรับ เนื่องจากเราไม่จำเป็นต้องเปิดเผยข้อมูลทุกอย่างต่อโลกภายนอก เช่นในคลาสเครื่องคำนวณ อาจจะมี properties สำหรับตัวกระทำ ตัวถูกกระทำ ตัวดำเนินการ และ ผลลัพธ์ เป็น public ส่วนตัวทดรวมทั้งการกระทำอื่นๆ ที่ไม่ใช่คำตอบสุดท้ายของการคำนวณ ก็กำหนดให้เป็น private

เมธอดสำคัญที่ใช้งานคู่กับ properties คือเมธอดสำหรับการเข้าถึงข้อมูล (accessors) ประกอบด้วยเมธอดสำหรับการดึงค่าจากวัตถุ (getter) และเมธอดสำหรับการกำหนดค่าให้วัตถุ (setter) ในภาษา C# คือ get { } และ set { } ตามลำดับ ซึ่ง accessor นี้สามารถใช้กำหนดลักษณะการเข้าถึงข้อมูลภายในวัตถุได้หลายรูปแบบ เช่น ถ้ามี getter เพียงอย่างเดียว เราจะได้ property แบบ read-only ถ้ามีทั้ง getter และ setter จะเป็น property แบบ read/write เป็นต้น

**รูปแบบการประกาศ properties**

|  |
| --- |
| <property type> <property name>  {  set  {  <statements>;  }  get  {  <statements>;  }  } |

**ตัวอย่างการประกาศ Properties**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | public class Student  {  // Fields declaration  private string name;  private string id;  private float gpa;  // read-only property  public string Name  {  get { return name; }  }  // read-only property  public string ID  {  get { return id; }  }  // read/write property  public float GPA  {  get { return gpa; }  set { gpa = value; }  }  } |

|  |
| --- |
| 1. **Methods** |

เมธอดภายในคลาส มีหลักการเขียนและใช้งานเหมือนเมธอดทั่วไป ยกเว้นเมธอดที่มีชื่อเดียวกับชื่อคลาส จะเรียกว่า constructor (ดูรายละเอียดจากหัวข้อ constructor)

ถึงแม้ว่า modifier ของเมธอดจะสามารถกำหนดการเข้าถึงจากภายนอกเป็น public, private, protected, internal ได้ก็ตาม แต่เนื่องจากเมธอดมีขอบเขตอยู่ภายในคลาส หากมีการกำหนดระดับการเข้าถึงของคลาสไว้ต่ำกว่า ของเมธอด ก็จะมีการกำหนดระดับโดยใช้ modifier ของคลาสแทนที่ เช่น ประกาศเมธอดเป็น public แต่ประกาศคลาสเป็น private ก็จะทำให้เมธอดและที่ประกาศเป็น public ทั้งหมด มีขอบเขตการมองเห็นจากภายนอกสูงสุดที่ระดับ private เท่านั้น

**รูปแบบการประกาศ method**

|  |
| --- |
| <Return type> Identifier ([<Parameter list>])  {  [<statements>;]  } |

|  |
| --- |
| 1. **Constructor (and Destructor)** |

Constructor เป็นกลไกที่สำคัญอย่างหนึ่ง ที่ช่วยให้ Object Oriented Programming ทำงานได้อย่างมีประสิทธิภาพ นั่นคือ constructor จะเป็นตัวช่วยกำหนดค่าเริ่มต้นให้กับตัวแปร (fields) ต่างๆ ภายในคลาส ช่วยให้เราทำงานได้อย่างราบรื่น นอกจากนั้น ในแต่ละคลาส สามารถมี constructor ได้เป็นจำนวนไม่จำกัด ดังนั้น หน้าที่สำคัญอีกประการของ constructor ก็คือการรับพารามิเตอร์สำหรับคลาส ในการใช้งานคลาส

จากข้อกำหนดที่ว่า คลาสต้องปกปิดข้อมูล จะทำให้ไม่มีหนทางใดที่จะกำหนดค่าตัวแปรในคลาส แต่เรายังมี constructor ที่สามารถรับพารามิเตอร์มาใส่ให้กับตัวแปรชนิด private ได้ในขณะที่เริ่มสร้างวัตถุ ดังตัวอย่างต่อไปนี้

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | class Program  {  static void Main(string[] args)  {  Student s1 = new Student("Prayuth", "60039999");  Console.WriteLine("Student name = " + s1.Name +", ID = " + s1.ID);  }  }  public class Student  {  // Fields declaration  private string name;  private string id;  private float gpa;  public Student(string value1, string value2)  {  this.name = value1;  this.id = value2;  }  // read-only property  public string Name  {  get { return name; }  }  // read-only property  public string ID  {  get { return id; }  }  // read/write property  public float GPA  {  get { return gpa; }  set { gpa = value; }  }  } |

จากโปรแกรมด้านบน properties ที่ชื่อ Name และ ID ของคลาส Student เป็นแบบ read only เราไม่สามารถกำหนดค่าให้กับ properties ทั้งสองตัว และไม่สามารถกำหนดค่าให้กับตัวแปร name และ id ได้ แต่เราสามารถให้ constructor ช่วยรับเป็นพารามิเตอร์ไปกำหนดค่าให้กับตัวแปรทั้งคู่ได้ นอกจากนี้เรายังสามารถใช้ properties ทำการตรวจสอบความถูกต้องของข้อมูลได้ ซึ่งจะเรียนรู้กันในการทดลอง

|  |
| --- |
| 1. **ส่วนประกอบอื่นๆ** |

ส่วนประกอบอื่นๆ ภายในคลาสเช่น event, delegate, indexer เป็นเนื้อหาที่ยังไม่นำมาบรรจุไว้ในการทดลองนี้

|  |
| --- |
| **การทดลอง : การใช้งานคลาสในลักษณะต่างๆ** |

คลาส มีลักษณะเป็น reference type ซึ่งจะไม่เก็บค่าตัวแปรต่างๆ ไว้ในตัวเอง วิธีการสร้างวัตถุจากคลาส จะทำได้โดยการใช้คีย์เวิร์ด new และทำการเชื่อมต่อวัตถุเข้ากับตัวแปร โดยใช้ตัวดำเนินการ = ดังตัวอย่าง

|  |  |
| --- | --- |
| 1  2 | Cat myCat = new Cat(); // create new object and then assign reference to variable  Cat myCat2 = myCat; // copy object's reference to another one. |

โดยปกติ ตัวภาษา C# เองจะไม่ได้จัดเตรียมคลาสใดๆ ไว้ให้เลย แต่เพื่ิอความสะดวกแก่ผู้ใช้ .NET Framework จึงรับหน้าที่จัดเตรียมไว้ให้ อย่างน้อยก็เพื่อความเข้ากันได้กับภาษาอื่นๆ ที่จัดเตรียมคลาสสำเร็จรูปไว้ให้ ผู้ที่สนใจสามารถศึกษา class library สำหรับภาษา C# ได้จาก หน้าเพจ Class Library Reference ของ Microsoft Developer Network [<https://msdn.microsoft.com/en-us/library/ms306608.aspx>] ซึ่งจะพบว่ามีอยู่เป็นจำนวนมาก

|  |
| --- |
| **การทดลองเรื่องการใช้งานคลาสและวัตถุ (object)** |

ทดลองสร้างคลาสโดยเขียนโปรแกรมตามตัวอย่างต่อไปนี้

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | using System;  namespace ConsoleApp5  {  class Program  {  static void Main(string[] args)  {  Point myPoint = new Point(); // create new object  myPoint.X = 10;  myPoint.Y = 10;  Console.WriteLine("myPoint.x = {0}", myPoint.X);  Console.WriteLine("myPoint.y = {0}", myPoint.Y);  Console.ReadLine();  }  }  class Point // class name  {  private int x; // fields  private int y;  public int X // properties  {  get { return x; }  set { x = value; }  }  public int Y  {  get { return y; }  set { y = value; }  }  }  } |

* บรรทัดที่ 9 เป็นการสร้างวัตถุใหม่โดยมีชนิด (class) เป็น Point
  + ทางด้านขวาของเครื่องหมาย = เป็นการสร้างวัตถุใหม่
* ด้านซ้ายของเครื่องหมาย = เป็นตัวแปรชื่อ myPoint สำหรับเก็บค่าการอ้างอิงไปยังวัตถุนั้น
* บรรทัดที่ 10 และ 11 เป็นการกำหนดค่าให้กับ properties X และ Y ของวัตถุ

|  |
| --- |
| **การทดลองเรื่องการปกปิดข้อมูลหรือการห่อหุ้ม (Encapsulation)** |

1. **การเข้าถึง fields แบบ public**

ให้เขียนโปรแกรมตาม code snippet ต่อไปนี้ แล้วรันโปรแกรม

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Program  {  static void Main(string[] args)  {  Student su = new Student();  su.Name = "Student Name";  su.ID = "12345678";  su.GPA = 3.5f;  Console.WriteLine("Student name : " + su.Name);  Console.WriteLine("Student ID : " + su.ID);  Console.WriteLine("Student GPA : " + su.GPA);  Console.ReadLine();  }  }  class Student  {  public string Name;  public string ID;  public float GPA;  } |

* จากโปรแกรมด้านบน ให้เปลี่ยนบรรทัดที่ 8 เป็น su.GPA = 7.5f;
* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร

จาก code ด้านบน จะเห็นว่าเราสามารถเข้าถึง fields ต่างๆ ของคลาส Student จากภายนอกได้อย่างอิสระ ซึ่งขัดกับหลักการปกปิดข้อมูลของ OOP เราจำเป็นต้องปกปิดข้อมูลดังกล่าว โดยการกำหนดให้ field ในบรรทัดที่ 17 - 19 มี modifier เป็น private ให้แก้ไขโปรแกรมเป็นดังนี้

|  |  |
| --- | --- |
| ...  17  18  19  ... | ...  private string Name;  private string ID;  private float GPA;  ... |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร

1. **การเข้าถึง fields ผ่านทาง properties**

เนื่องจากเราต้องการให้ fields ถูกปิดบังจากภายนอกจึงต้องกำหนด access modifier เป็น private ทำให้ผลที่ตามมาคือ ไม่สามารถกำหนดค่าให้กับ fields ต่างๆ ได้ เราจึงต้องพึ่งองค์ประกอบอีกอย่างหนึ่ง ของคลาสซึ่งสามารถมี modifier เป็น public โดยไม่ขัดกับกฏของ OOP นั่นก็คือ properties

ให้แก้ไขคลาส Student โดยการเพิ่ม properties และเปลี่ยนชื่อตัวแปรสำหรับ fields เป็นตัวพิมพ์เล็กดังต่อไปนี้

|  |  |
| --- | --- |
|  | class Student  {  private string name;  private string id;  private float gpa;  public string Name  {  get { return name; }  set { name = value; }  }  public string ID  {  get { return id; }  set { id = value; }  }  public float GPA  {  get { return gpa; }  set { gpa = value; }  }  } |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร

1. **การตรวจสอบความถูกต้องของข้อมูลโดย properties**

การใช้งาน properties มีประโยชน์ในเรื่องการควบคุมการเข้าถึงข้อมูล (ผ่านเมธอด getter และ setter) จากตัวอย่างด้านบนจะพบว่า GPA ยังคงมีค่าเป็น 7.5 ซึ่งมีค่าเกินปกติ 4.0 ซึ่งเราสามารถตรวจสอบความถูกต้องของข้อมูล ได้ในเมธอด setter หรือ getter ให้แก้โปรแกรมเป็นดังต่อไปนี้

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | class Program  {  static void Main(string[] args)  {  Student su = new Student();  try  {  su.Name = "Student Name";  su.ID = "12345678";  su.GPA = 7.5f;  Console.WriteLine("Student name : " + su.Name);  Console.WriteLine("Student ID : " + su.ID);  Console.WriteLine("Student GPA : " + su.GPA);  }  catch (Exception e)  {  Console.WriteLine(e.Message);  }  Console.ReadLine();  }  }  class Student  {  private string name;  private string id;  private float gpa;  public string Name  {  get { return name; }  set { name = value; }  }  public string ID  {  get { return id; }  set { id = value; }  }  public float GPA  {  get  {  return gpa;  }  set  {  if (value > 0.0 && value <= 4.0 )  gpa = value;  else  throw (new Exception("Error!!!! invalid GPA"));  }  }  } |

|  |
| --- |
| **อธิบายการทำงานของโปรแกรม**  โปรแกรมด้านบนนี้ จะมีการตรวจสอบความถูกต้องของข้อมูลโดยเมธอด setter ของคลาส Student (ในบรรทัดที่ 43 - 49) ถ้าหากข้อมูลที่ป้อนไม่ถูกต้องจะมีการ throw exception และจะไปรับการ exception ในประโยค try-catch ของฝั่งผู้เรียก (ในบรรทัดที่ 6 - 18) แต่ถ้าข้อมูลที่ป้อนอยู่ในขอบเขตที่ถูกต้อง จะรายงานผลออกมาตามปกติ (ในบรรทัดที่ 11 - 13). |

* ทดลองเปลี่ยนค่าให้อยู่ในขอบเขตที่ถูกต้อง แล้วรันและบันทึกผลการทดลอง

1. **การกำหนดวิธีการเข้าถึงข้อมูลภายในคลาส**

โปรแกรมด้านบน ยังมีจุดอ่อนอีกประการหนึ่งคือ ผู้ใช้สามารถแก้ไข properties ที่ชื่อ Name และ ID ของวัตถุที่สร้างจากคลาส Student ได้ (ในบรรทัดที่ 8 - 9) การป้องกันการแก้ไขข้อมูลจากภายนอก สามารถทำได้โดยการลบเมธอด setter ออกไปจาก property ของคลาส แต่ผลที่ตามมาก็คือ เราจะไม่สามารถกำหนดค่าเริ่มต้นให้กับวัตถุได้ แต่ยังโชคดีที่คลาสได้ให้ทางออกอย่างหนึ่งแก่เรา นั่นคือการใช้ constructor

จากโปรแกรมด้านบนเราจะพบว่า Name และ ID มี setter อยู่ ซึ่งเป็นหนทางเดียวที่จะนำข้อมูลเข้าไปยังคลาส เราต้องลบออกไป และเพิ่ม constructor ที่รับพารามิเตอร์จำนวน 2 ตัว (ได้แก่ Name และ ID) มาทำหน้าที่ดังกล่าวแทน เมื่อทำดังนั้นแล้ว เราจะได้คลาส Student ที่สามารถกำหนดชื่อและ ID ได้ในขณะสร้างวัตถุและไม่สามารถแก้ไขในภายหลังได้

ให้แก้โปรแกรมเป็นดังต่อไปนี้

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | class Program  {  static void Main(string[] args)  {  Student su = new Student("Student Name", "12345678");  try  {  // su.Name = "My name";  su.GPA = 3.5f;  Console.WriteLine("Student name : " + su.Name);  Console.WriteLine("Student ID : " + su.ID);  Console.WriteLine("Student GPA : " + su.GPA);  }  catch (Exception e)  {  Console.WriteLine(e.Message);  }  Console.ReadLine();  }  }  class Student  {  private string name;  private string id;  private float gpa;  public Student(string Name, string ID)  {  this.name = Name; // assign auto variable to the field  this.id = ID;  }  public string Name  {  get { return name; }  }  public string ID  {  get { return id; }  }  public float GPA  {  get  {  return gpa;  }  set  {  if (value > 0.0 && value <= 4.0 )  gpa = value;  else  throw (new Exception("Error!!!! invalid GPA"));  }  }  } |

* จงอธิบายการทำงานของโปรแกรมด้านบน ว่าเกิดอะไรขึ้น เพราะอะไร
* ให้ทำการ uncomment บรรทัดที่ 8 (ลบ // ด้านหน้า su.Name = "My name"; ออกไป) แล้วรันโปรแกรม พร้อมทั้งบันทึกผล และอธิบายให้เหตุผลประกอบ
* ทดลองเปลี่ยน GPA เป็นค่าที่มากกว่า 4.0 ผลที่ได้เป็นเช่นไร

|  |
| --- |
| **สรุปผลการทดลอง** |
|  |

|  |
| --- |
| **คำถาม** |
| นักศึกษาที่เรียนในชั้นปีที่ 3 ของภาควิชาครุศาสตร์วิศวกรรม จะต้องออกฝึกงานในสถานประกอบการด้านอุตสาหกรรม นักศึกษาจะได้รับเงินเดือนในการทำงาน ตามค่าแรงขั้นต่ำ (ปัจจุบัน คือ 300 บาท) แต่ไม่เกิน 450 บาทต่อวัน  ให้เขียนคลาส TraineeStudent ที่มี field ชื่อ StudentID และ salary โดยทั้งคู่จะถูกกำหนดค่าผ่าน constructor แต่จะมีเมธอดที่ชื่อ getSalary() และ getStudentID() ไว้อ่านค่าตัวแปร field ทั้งสอง |

|  |
| --- |
| **การทดลองเรื่องการสืบทอดคุณสมบัติ (Inheritance)** |

ในโลกแห่งความเป็นจริง วัตถุทุกชนิด จะแบ่งปันหรือมีคุณสมบัติและความสามารถร่วมกันอยู่เสมอ ยกตัวอย่างเช่น อาชีพครู ที่มีหลากหลายระดับ แต่มีคุณสมบัติบางอย่างร่วมกันเช่น บทบาทหน้าที่ ความรับผิดชอบ อัตราเงินเดือน เงินประจำตำแหน่ง เป็นต้น ในการทดลองเรื่องการสืบทอดคุณสมบัตินี้ เราจะใช้ลำดับการสืบทอดคุณสมบัติของอาชีพครู เป็นวัตถุในการทดลอง ให้พิจารณารูปต่อไปนี้



**การทดลอง**

1. **สร้างคลาสชื่อ Teacher ไว้ด้านล่างของคลาส Program**

**หมายเหตุ** หมายเลขบรรทัด มีไว้เพื่ออธิบายการทำงานของโปรแกรม (ถ้ามี) อาจจะไม่ตรงกับหมายเลขบรรทัดในโปรแกรม

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | class Teacher  {  // constructor (for initial private/protected variables)  public Teacher(string name, float billingRate)  {  this.name = name;  this.billingRate = billingRate;  }  // figure out the charge based on teacher's rate  public float CalculateCharge(float hours)  {  return (hours \* billingRate);  }  // return the name of this type  public string TypeName()  {  return ("Teacher");  }  private string name;  protected float billingRate;  } |

1. **แก้ไขเมธอด Main()**

**หมายเหตุ** หมายเลขบรรทัด มีไว้เพื่ออธิบายการทำงานของโปรแกรม (ถ้ามี) อาจจะไม่ตรงกับหมายเลขบรรทัดในโปรแกรม

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | static void Main(string[] args)  {  Teacher teacher = new Teacher("Tom", 350f);  // teacher work for 20Hr/month  Console.WriteLine("{0} charge = {1}", teacher.TypeName(),  teacher.CalculateCharge(20f));  Console.ReadLine();  } |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร
* อธิบายการทำงาน ตามที่เข้าใจ

**3. ให้สร้างคลาสชื่อ Professor ซึ่งสืบทอดคุณสมบัติจากคลาส Teacher ดังต่อไปนี้**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | class Professor : Teacher  {  private float emolument; // เงินประจำตำแหน่ง    public Professor(string name, float billingRate): base(name, billingRate)  {  }    public Professor(string name, float billingRate, float emolument)  : this(name, billingRate)  {  this.emolument = emolument;  }    // new function, because it's different than the base version  public new float CalculateCharge(float hours)  {  if (hours < 1.0F)  hours = 1.0F; // minimum charge.  return (hours \* billingRate) + emolument;  }  // new function, because it's different than the base version  public new string TypeName()  {  return ("Professor");  }  } |

ในคลาส Professor นี้จะมี constructor จำนวน 2 ตัว คือ ตัวที่ 1 จะพ้องรูปกับคลาส Teacher ส่วนตัวที่ 2 จะมีการเพิ่มพารามิเตอร์มา 1 ตัว สำหรับรองรับการใส่ค่าเงินประจำตำแหน่ง

เนื่องจาก constructor เป็นเมธอดที่ไม่สามารถสืบทอดผ่านการทำ inheritance ได้ ดังนั้นใน derived class ทุกตัว เราต้องเขียน constructor ไว้เสมอ ซึ่งในคลาส professor นี้มี constructor จำนวน 2 ตัวโดยตัวแรก มีรูปแบบเป็น

|  |
| --- |
| public Professor(string name, float billingRate): base(name, billingRate) |

หมายความว่า constructor ตัวนี้รับพารามิเตอร์ในขณะสร้างวัตถุเป็นจำนวน 2 พารามิเตอร์ แต่เนื่องจากตัวแปรทั้งสองที่จะนำค่าไปเก็บ มี accessor เป็น private และ protected จึงต้องส่งต่อให้ constructor ของ base class ช่วยดำเนินการเก็บลงใน fields ซึ่งวิธีการดังกล่าวทำได้โดยใช้คำสั่ง : base(name, billingRate) ถ้าหากเราไม่เขียน constructor ตัวนี้ ก็จะไม่สามารถนำค่าไปเก็บใน fields ทั้งสองได้ และค่าใน fields ทั้งสองจะมีค่าเป็น default value ส่วน constructor ตัวที่สองจะรับพารามิเตอร์ในขณะสร้างวัตถุเป็นจำนวน 3 พารามิเตอร์ โดยพารามิเตอร์ที่เพิ่มเข้ามาคือเงินประจำตำแหน่ง (float emolument)

|  |
| --- |
| public Professor(string name, float billingRate, float emolument)  : this(name, billingRate) |

วิธีการที่ง่ายและตรงไปตรงมาสำหรับ constructor ตัวนี้ก็คือ เราจะนำพารามิเตอร์ตัวที่ 3 ไปเก็บในฟิลด์ private float emolument ส่วนพารามิเตอร์ 2 ตัวแรกจะส่งให้ constructor ตัวที่หนึ่งรับไปดำเนินการ ด้วยคำสั่ง : this(name, billingRate) ซึ่งก็จะทำหน้าที่เรียก constructor ใน base class มาทำงาน

**3. แก้ไขเมธอด Main()**

**หมายเหตุ** หมายเลขบรรทัด มีไว้เพื่ออธิบายการทำงานของโปรแกรม (ถ้ามี) อาจจะไม่ตรงกับหมายเลขบรรทัดในโปรแกรม

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | static void Main(string[] args)  {  Teacher teacher = new Teacher("Tom", 350f);  Professor prof = new Professor("Jerry", 500f, 5000f);  // teacher work 20Hr/month  Console.WriteLine("{0} charge = {1}", teacher.TypeName(),  teacher.CalculateCharge(20f));  // professor work 12Hr/month  Console.WriteLine("{0} charge = {1}", prof.TypeName(),  prof.CalculateCharge(12F));  Console.ReadLine();  } |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร
* อธิบายการทำงาน ตามที่เข้าใจ

|  |
| --- |
| **หมายเหตุ** การใช้งานคลาสเพื่อการสืบทอดคุณสมบัติยังมีรายละเอียดปลีกย่อยอีกหลายประการ ซึ่งจะได้เรียนในวิชา Object Oriented Programming |

|  |
| --- |
| **การทดลองเรื่องการพ้องรูป (Polymorphism)** |

**1. การพ้องรูปแบบ static polymorphism**

**1.1 Method overloading**

การพ้องรูปแบบ static polymorphism ในกรณี method overloading จะเกิดขึ้นกับเมธอดในคลาสเดียวกัน ให้เขียนโปรแกรมต่อไปนี้ แล้วรันและสังเกตผลการทำงานของโปรแกรม

(โปรแกรมนี้ยังรันไม่ได้ ต้องเพิ่มส่วน using ก่อน)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | class Program  {  void print(int i)  {  Console.WriteLine("Printing int: {0}", i);  }    void print(double f)  {  Console.WriteLine("Printing float: {0}", f);  }  void print(string s)  {  Console.WriteLine("Printing string: {0}", s);  }    static void Main(string[] args)  {  Program p = new Program();  p.print(5);  p.print((5.26f).ToString());  p.print(264.2871);  p.print((int)1354.23);  p.print("Hello World of C#");  Console.ReadKey();  }  } |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร
* อธิบายการทำงาน ตามที่เข้าใจ

**1.2 Operator overloading**

ในกรณี operator overloading นี้จะช่วยให้เราสามารถใช้งานวัตถุได้อย่างเป็นธรรมชาติมากขึ้น เช่นการบวกเวคเตอร์ 2 ตัวเข้าด้วยกันโดยตัวดำเนินการบวก หรือการกำหนดค่าลบให้กับเวคเตอร์ด้วยตัวกระทำลบแบบ unary ให้เขียนโปรแกรมต่อไปนี้ พร้อมทั้งรันและบันทึกผลการทำงานของโปรแกรม

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | class Program  {  static void Main(string[] args)  {  Vector v1 = new Vector(1, 4);  Vector v2 = new Vector(8, 5);  Vector v3 = v1 + v2;  Console.WriteLine("vector v1 = " + v1);  Console.WriteLine("vector v2 = " + v2);  Console.WriteLine("vector v3 = " + v3);  Console.WriteLine("-v3 = " + (-v3));  Console.ReadKey();  }  }  class Vector  {  public Vector(int x, int y)  {  this.X = x;  this.Y = y;  }  public int X { get; set; } // X - accessor (property)  public int Y { get; set; } // Y - accessor (property)  public override string ToString()  {  return string.Format("({0},{1})", X, Y);  }  // All operator overloads must be public and static.  public static Vector operator +(Vector v1, Vector v2) //binary operator +  {  return new Vector(v1.X + v2.X, v1.Y + v2.Y);  }  public static Vector operator -(Vector v1) //unary operator -  {  return new Vector(-v1.X, -v1.Y );  }  } |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร
* อธิบายการทำงาน ตามที่เข้าใจ

**2. การพ้องรูปแบบ dynamic polymorphism**

การพ้องรูปแบบ dynamic polymorphism จะเกิดขึ้นเมื่อใช้งานการพ้องรูปร่วมกับการสืบทอดคุณสมบัติ สมมติมีระบบคลาสดังรูปด้านล่างนี้ ซึ่งคลาส Student และ Teacher ต่างก็สืบทอดมาจากคลาส Person แต่จะมี modifier ที่ต่างกัน นั่นคือเมธอด SayHi() ของ Student มี modifier เป็น override แต่ของ Teacher มี modifier เป็น new

ในการสร้างวัตถุ เราสามารถสร้างวัตถุเป็นชนิดคลาสลูกแล้วนำไปกำหนดให้กับตัวแปรที่สร้างจากคลาสแม่ได้ เมื่อเรียกเมธอดในคลาสลูก ระบบจะวิ่งหาเมธอดที่ทำ polymorphism จนพบลำดับชั้นสุดท้ายแล้วเรียกขึ้นมาทำงาน

เมธอดที่ไม่ถูกเรียกขึ้นมาทำงานในลำดับขั้นการสืบทอด จะเรียกว่ามันถูกซ่อนจากระบบ (ด้วยกระบวนการ method hiding)



ให้สร้าง project เป็น Console Application (.NET Framework) แล้วเพิ่มคลาสต่อไปนี้ลงใต้คลาส Program

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | class Person  {  public virtual void SayHi()  {  Console.WriteLine("Hi I'm a person");  }  }  class Student : Person  {  public override void SayHi()  {  Console.WriteLine("Hi I'm a student");  }  }  class Teacher : Person  {  public new void SayHi()  {  Console.WriteLine("Hi I'm a teacher");  }  } |

ในเมธอด Main ให้แก้โค้ดเป็นดังต่อไปนี้

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | static void Main(string[] args)  {  Person p = new Person();  p.SayHi();  Student s = new Student();  s.SayHi();  Person sp = new Student();  sp.SayHi();  Teacher t = new Teacher();  t.SayHi();  Person tp = new Teacher();  tp.SayHi();  Console.ReadKey();  } |

* รันและสังเกตผลการทำงานของโปรแกรม มีสิ่งใดผิดปกติหรือไม่ อย่างไร
* อธิบายการทำงาน ตามที่เข้าใจ
* การประกาศแบบใด ที่ทำให้เกิด method overriding และ method hiding.

**แบบฝึกหัด**



ให้ดัดแปลงโปรแกรมด้านบน เพื่อเพิ่มคลาส Professor ที่สืบทอดจากคลาส Teacher ดังแผนภาพด้านบน จากนั้นให้รันโปรแกรมและบันทึกผล โดยกำหนดให้คำสั่งในการเรียกใช้ดังต่อไปนี้ พร้อมทั้งอธิบายว่าเหตุใดจึงเป็นเช่นนั้น

1.

|  |
| --- |
| static void Main(string[] args)  {  Person p = new Professor();  p.SayHi();  Console.ReadKey();  } |

2.

|  |
| --- |
| static void Main(string[] args)  {  Student p = new Professor();  p.SayHi();  Console.ReadKey();  } |

3.

|  |
| --- |
| static void Main(string[] args)  {  Teacher t = new Person();  t.SayHi();  Console.ReadKey();  } |

1. "NATO Software Engineering Conference 1968." <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>. Accessed 20 Sep. 2017. [↑](#footnote-ref-0)
2. "NATO Software Engineering Conferences." 13 Aug. 2001, <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/>. Accessed 20 Sep. 2017. [↑](#footnote-ref-1)