การเขียนโปรแกรม ด้วยภาษา C#

Polymorphism

Polymorphism

Polymorphism

- Polymorphism เป็นการประกอบขึ้นจาก 2 คำ
  - Poly แปลว่ามากกว่าหนึ่ง (multiple)
  - Morphs แปลว่ารูปแบบ (forms)
  - แปลรวมกันได้ว่า “หลายรูปแบบ”
- Polymorphism เป็นวิธีการของคลาสที่ช่วยให้สามารถมี method ชื่อเดียวกัน แต่มีการทำงานภายในที่แตกต่างกันได้
  - ถ้าใช้ชื่อ method ซ้ำในคลาสเดียวกัน เรียกว่า method overloading
  - ถ้าใช้ชื่อ method ซ้ำในคลาสที่สืบทอดกันมา เรียกว่า method overriding

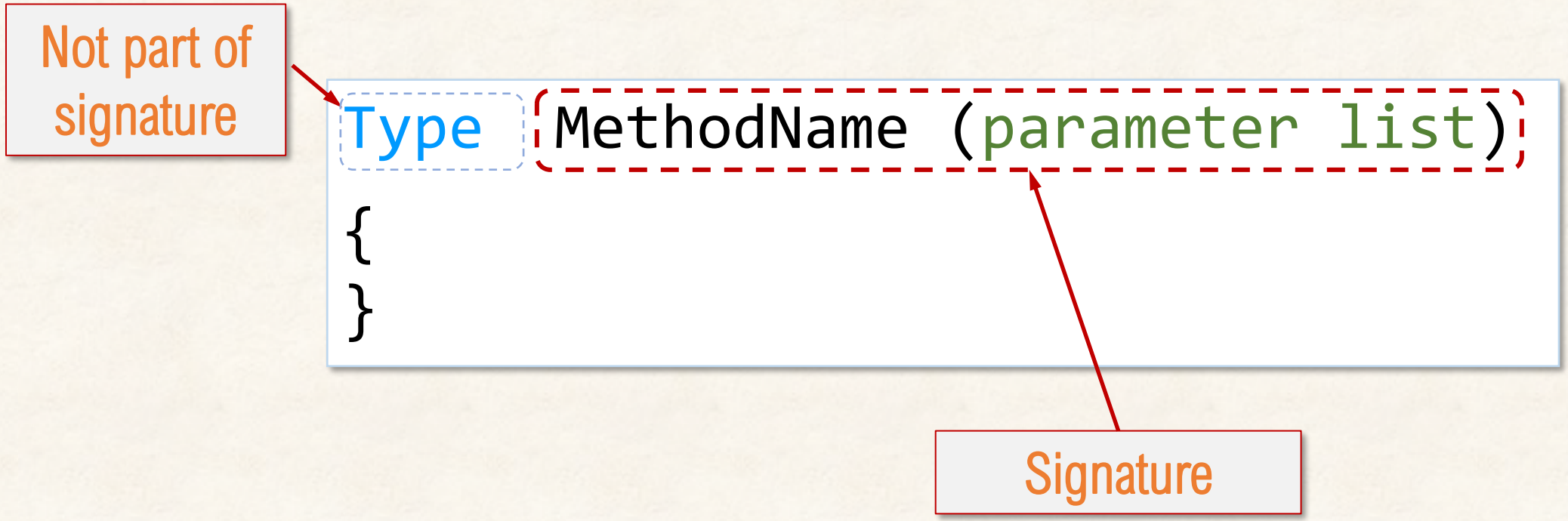
Method Overloading

- คลาสใด ๆ สามารถมี Methods ที่ชื่อซ้ำกันได้
  - แต่ต้องมี signature ต่างกัน
- Signature ของ Methods ประกอบด้วย
  - ชื่อของ method
  - จำนวนของพารามิเตอร์
  - Type และลำดับของพารามิเตอร์
  - Modifier ของพารามิเตอร์

- Return type ของ method ไม่ถือเป็นส่วนหนึ่งของ signature

- ชื่อของ formal parameter ไม่ถือเป็นส่วนหนึ่งของ signature

Signature ของ Methods



Method Overloading

- Method Overloading เรียกได้อีกอย่างว่า “Compile Time Polymorphism”
  - บ้างก็เรียก early binding
  - บ้างก็เรียก static binding

ตัวอย่าง method overloading

```
static void Main(string[] args)
{
    long result = Calculator.AddValues(1, 2);
    result = Calculator.AddValues(1, 2, 3);
    result = Calculator.AddValues(1.5f, 2.5f);
    result = Calculator.AddValues(1L, 2L);
}
```

```
public static class Calculator
{
    public static long AddValues(int a, int b) { return a + b; }
    public static long AddValues(int c, int d, int e) { return c + d + e; }
    public static long AddValues(float f, float g) { return (long)(f + g); }
    public static long AddValues(long h, long i) { return (long)(h + i); }
}
```

ตัวอย่าง signature ที่ใช้ในการทำ overloading ไม่ได้

```
class CalculatorA
{
    long  AddValues(long a, long b) { return a + b; }

    long  AddValues(long c, long d) { return c + d; }

    ulong AddValues(long a, long b) { return a + b; }
}
```

Not part of signature

Signature

ตัวอย่าง method overloading

```
static void Main(string[] args)
{
    Calculate calculate = new Calculate();
    calculate.AddNumbers(2, 3);
    calculate.AddNumbers(4, 5, 6);
}
```

```
public class Calculate

{
    public void AddNumbers(int a, int b)
    {
        Console.WriteLine($"{a} + {b} = {a + b}");
    }
    public void AddNumbers(int a, int b, int c)
    {
        Console.WriteLine($"{a} + {b} + {c} = {a + b + c}");
    }
}
```

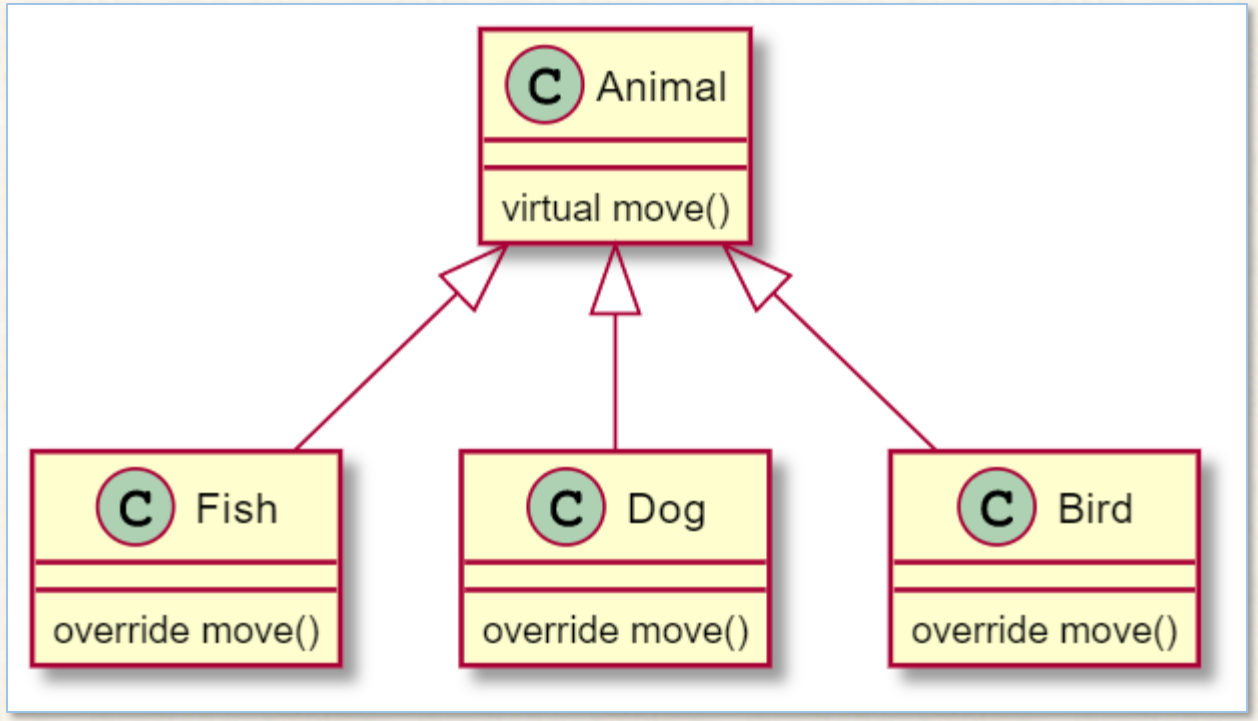
Run Time Polymorphism

- Run time Polymorphism เรียกได้อีกอย่างว่า “Method Overloading”
  - บ้างก็เรียก late binding
  - บ้างก็เรียก dynamic binding
- Method ใน derived class สามารถทับการกระทำของ method ใน base class ได้
  - Method ทั้งสองต้องมี signature ตรงกัน
  - Method ใน base class กำกับด้วย modifier ‘virtual’
  - Method ใน derived class กำกับด้วย modifier ‘override’

Run Time Polymorphism

- ใน Run Time Polymorphism ผู้สร้างคลาสจะต้องสร้างคลาสที่สามารถทำการสืบทอดได้
  - Base class มี method ที่สามารถทำงานได้และกำกับด้วย modifier ‘virtual’
  - Base class มี method ที่สามารถทำงานได้และกำกับด้วย modifier ‘override’
- ในการสร้างวัตถุ เราสามารถสร้างวัตถุของ derived class แต่เชื่อมไปยัง Reference ของ base class
  - การทำงานจะเป็น method ของ derived class
  - Virtual-override จะส่งผ่านการทำงานไปยังคลาสที่เหมาะสม

ตัวอย่าง Run time polymorphism



- Animal มีความสามารถในการเคลื่อนที่
  - Fish เคลื่อนที่โดยการว่ายน้ำ
  - Dog เคลื่อนที่โดยการเดินหรือวิ่ง
  - Bird เคลื่อนที่โดยการบินหรือเดิน

ตัวอย่าง Run time polymorphism

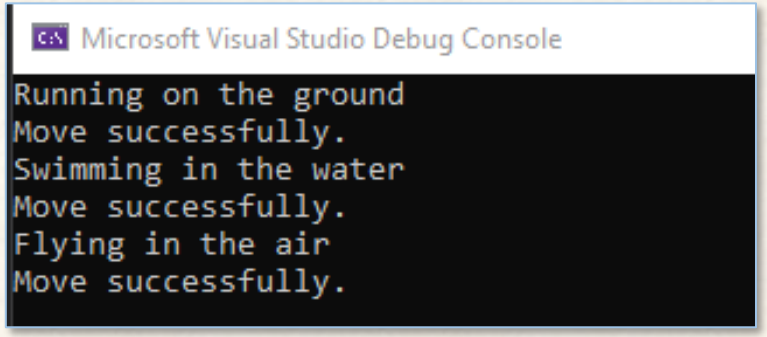
```
class Animals
{
    public virtual void Move()
    {
        Console.WriteLine("Move successfully.");
    }
}

class Dog : Animals
{
    public override void Move()
    {
        Console.WriteLine("Running on the ground");
        base.Move();
    }
}
```

ตัวอย่าง Run time polymorphism

```
class Fish : Animals
{
    public override void Move()
    {
        Console.WriteLine("Swimming in the water");
        base.Move();
    }
}


class Bird : Animals
{
    public override void Move()
    {
        Console.WriteLine("Flying in the air");
        base.Move();
    }
}
```

ตัวอย่าง Run time polymorphism

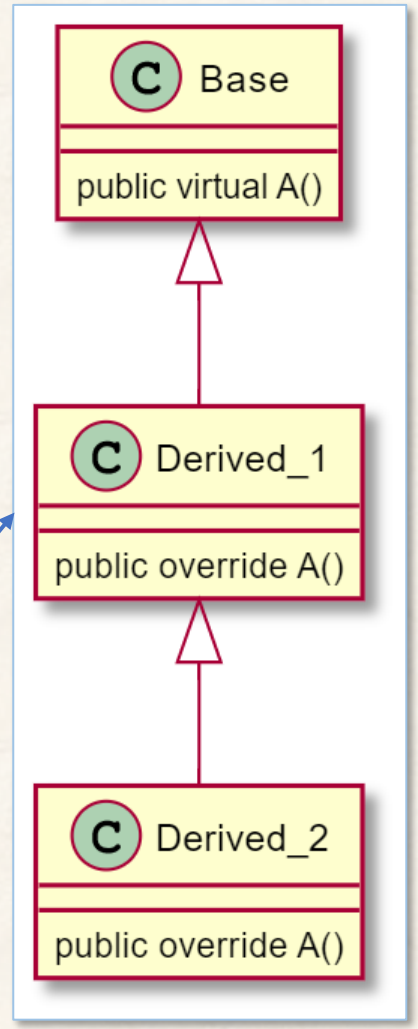
```
static void Main(string[] args)
{
    Animals animalA = new Dog();
    Animals animalB = new Fish();
    Animals animalC = new Bird();
    animalA.Move();
    animalB.Move();
    animalC.Move();
}
```



การสืบทอด virtual member

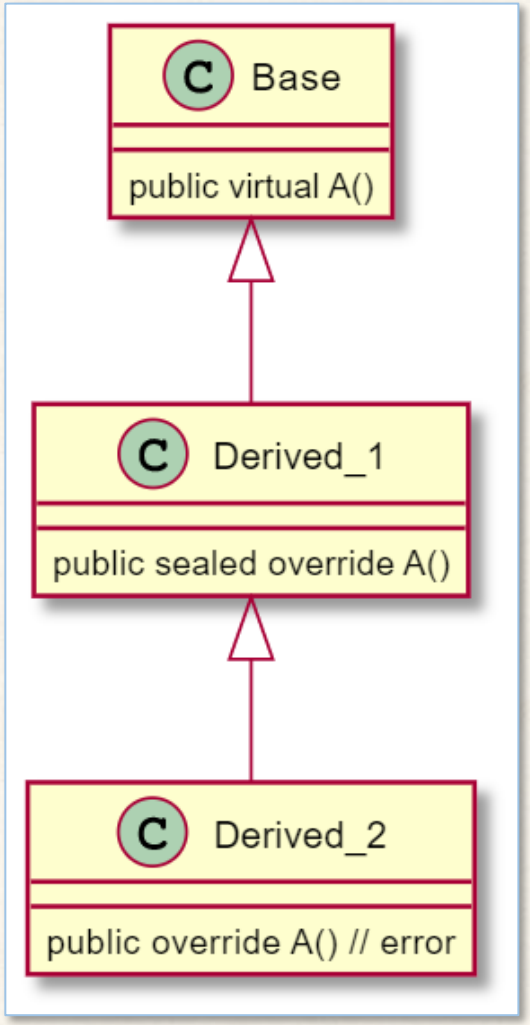
- เมื่อ method ใน base class ถูกประกาศเป็น virtual แล้ว method ที่สืบทอดมาจะมีความเป็น virtual ด้วยเสมอ
- ไม่ว่าละมีลำดับชั้นเท่าใดก็ตาม การสืบทอด virtual method จะยังคงมีความเป็น virtual ใน derive class เสมอ

Derived_2.A() สามารถ override Derived_1.A() ได้เพราะว่า Derived_1.A() สืบทอดความเป็น virtual มาจาก Base.A()



การป้องกันไม่ให้ derived class สืบทอด virtual member

- ใช้ keyword ‘sealed’ กับ method ที่ต้องการไม่ให้มีการสืบทอดความเป็น virtual



การป้องกันไม่ให้ derived class สืบทอด virtual member

```
class Base
{
    public virtual void A() { }
}
class Derived_1: Base
{
    public sealed override void A() { }
}

class Derived_2: Derived_1
{
    public override void A() { }
}
```

Error CS0239 'Derived_2.A()': cannot override inherited member 'Derived_1.A()' because it is sealed

Abstract Members

- Abstract member เป็น member ที่ถูกสร้างมาเพื่อให้ถูก overridden โดย member ใน derived class
- Abstract member จะมีเฉพาะการประกาศ (declare) แต่ไม่มีส่วน implementation
- มี member สี่ประเภทที่สามารถเป็น abstract member ได้
  - Methods
  - Properties
  - Events
  - Indexers

Abstract Members

ใส่ semicolon ในตำแหน่งของ implementation

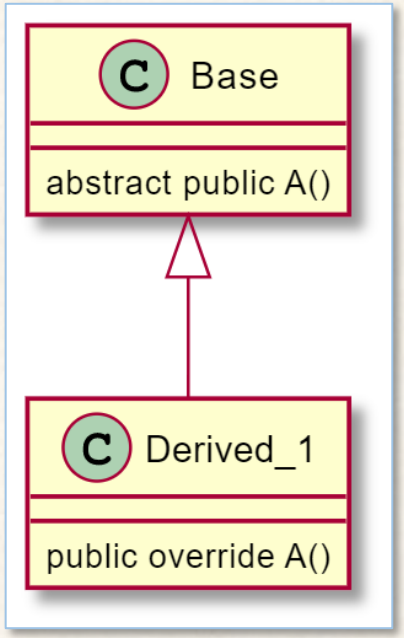
```
abstract public void PrintStuff(string s);


abstract public int MyProperty
{
    get;
    set;
}
```

ใส่ semicolon ในตำแหน่งของ implementation

Abstract member จะโยนภาระให้ derived class ทำส่วน implementation

Abstract Members

- ใน Abstract member นั้นไม่สามารถใช้ modifier ‘virtual’ ได้
- ด้วยความเป็น Abstract member ถึงแม้จะไม่มี modifier ‘virtual’ แต่ส่วน implementation ใน derived class ต้องมี modifier ‘override’ ตามปกติ

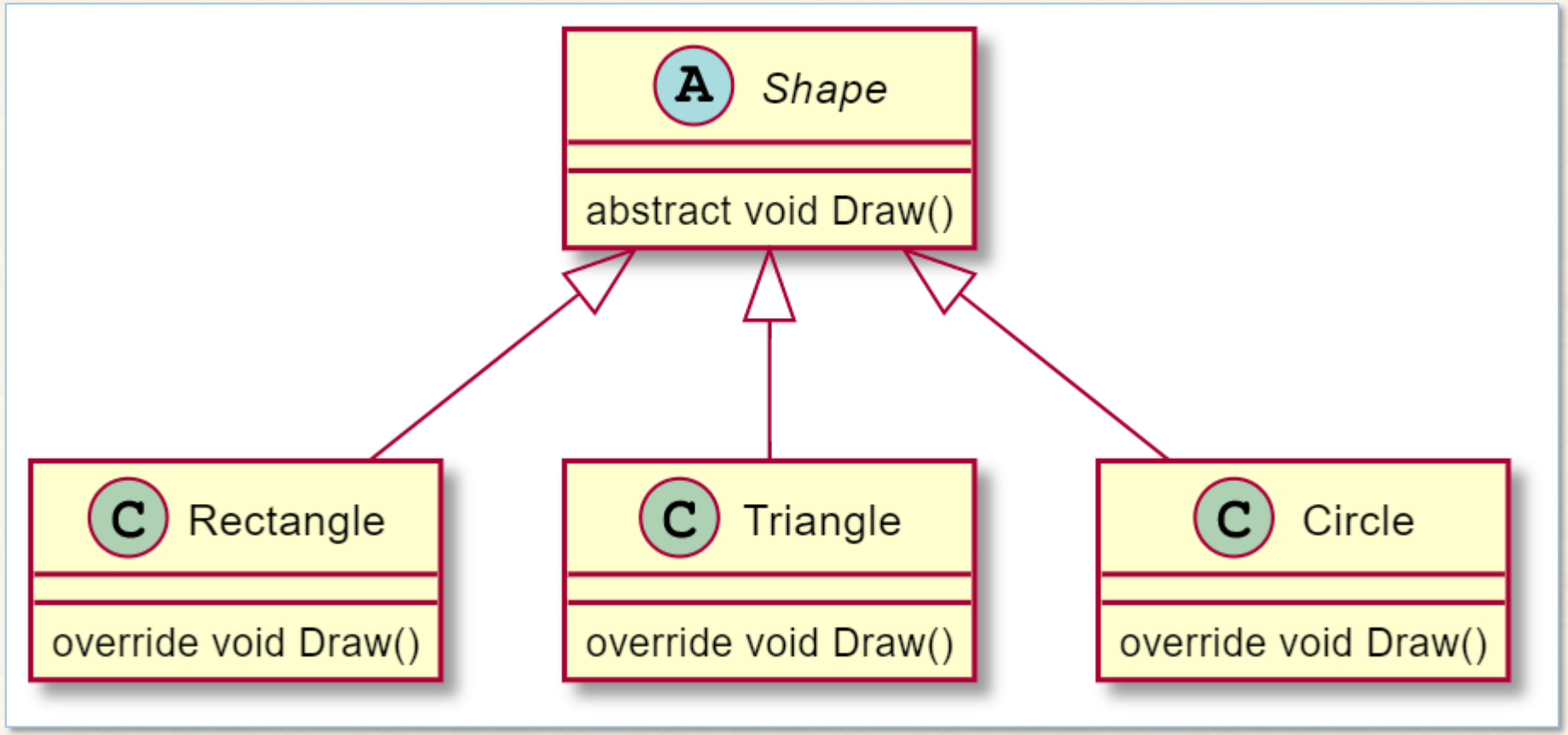


Abstract Class

- Abstract Class ถูกออกแบบมาสำหรับการสืบทอดเพียงอย่างเดียว
- ไม่สามารถนำ abstract class ไปสร้าง instance ได้
- ประกาศ Abstract Class โดยใช้ modifier ‘abstract’

```
abstract class AbsClass
{
      ...
}

abstract class MyAbsClass : AbsClass
{
      ...
}
```

ตัวอย่าง Abstract Class



ตัวอย่าง Abstract Class

```
abstract class Shape
{
    public abstract void Draw();
}

class Rectangle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Draw rectangle");
    }
}
```

ตัวอย่าง Abstract Class

```
class Triangle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Draw Triangle");
    }
}

class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Draw Circle");
    }
}
```

ตัวอย่าง Abstract Class

```
static void Main(string[] args)
{
    Shape[] shapes = new Shape[3];
    shapes[0] = new Rectangle();
    shapes[1] = new Triangle();
    shapes[2] = new Circle();

    foreach (var shape in shapes)
    {
        shape.Draw();
    }
}
```