# การเขียนโปรแกรม ด้วยภาษา C#

## Members ชนิดอื่น ๆ ของ Class

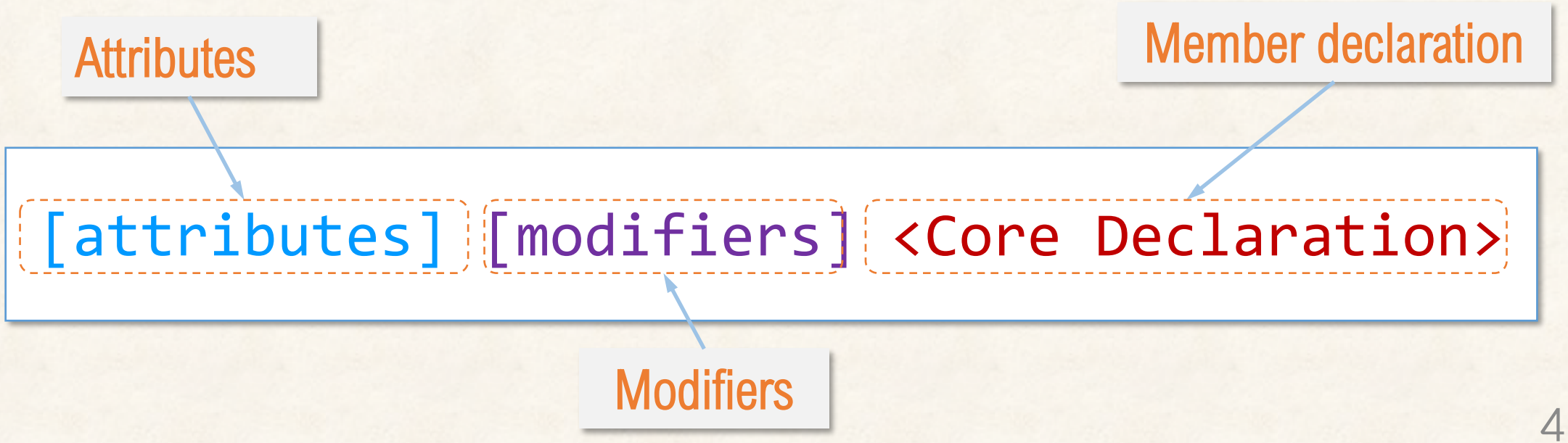
# ส่วนประกอบของ Class

| Data members | Function Members |
| --- | --- |
| ✓ Fields | ✓ Methods |
| ➢ Constants | ➢ Properties |
| | ➢ Constructors |
| | ➢ Destructors |
| | ➢ Operators |
| | ➢ Indexers |
| | ➢ Events |

# Attribute and Modifier

# Attribute and Modifier

- เพื่อให้การเขียนโปรแกรมรองรับ OOP ได้อย่างสมบูรณ์ ในการประกาศ member ต่าง ๆ ของคลาส จะสามารถกำหนด attribute และ modifier ได้
  - attribute และ modifier เป็น option จะมีหรือไม่ก็ได้



# Attribute

- ถ้าในการประกาศ member นั้นมี attributes
  - ให้วาง attribute ไว้ก่อน modifier และการประกาศ member ตามปกติ
  - ถ้ามี attribute หลายตัว สามารถวางสลับตำแหน่งกันได้
  - attribute เขียนไว้ในเครื่องหมาย [ ]

```
[SomeAttribute] int MyInt = 0;

[SomeAttribute]
int Method()  { statements; }
```

# Modifier

- ถ้าในการประกาศ member นั้นมี modifiers
  - ให้วาง modifier ไว้ก่อนการประกาศ member ตามปกติ
  - ถ้ามี modifier หลายตัว สามารถวางสลับตำแหน่งกันได้

```
public static  int MyInt1 = 0;
static public  int MyInt2 = 0;

private int Method1()  { statements; }
private static int Method2()  { statements; }
```

# Modifier

```
  public
  private
  protected   <field declaration> | <method declaration>
  static
  const
  ...
```

Modifiers

# Instance vs static members

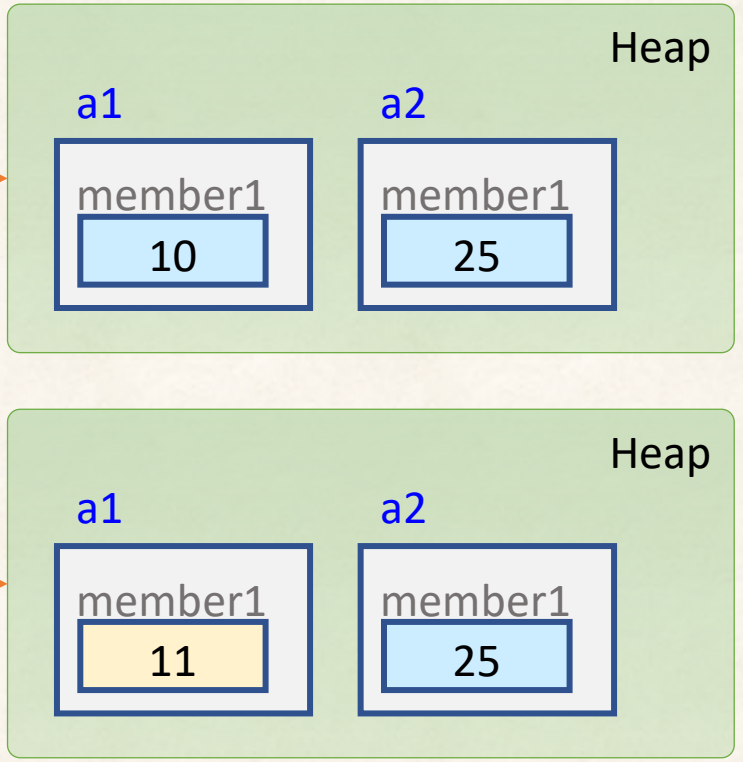
# Instance vs static members

- Member ของ class สามารถเป็นได้ทั้งแบบ instance member และ static member
- instance member จะถูกสร้างขึ้นใหม่ทุกครั้งเมื่อสร้าง instance ด้วยคำสั่ง new
  - การเปลี่ยนแปลงค่าใด ๆ ใน instance member จะไม่กระทบต่อ member ใน instance อื่น ๆ
- static member จะถูกสร้างเพียงครั้งเดียวและไม่ใช้คำสั่ง new
  - ทุก instance จะใช้ static member ร่วมกัน การเปลี่ยนค่าจะส่งผลต่อทุก instance

# Instance members

```
class A
{
    public int member1;
}

class Program
{
    static void Main()
    {
        A a1 = new A();
        A a2 = new A();
        a1.member1 = 10;
        a2.member1 = 25;
        Console.WriteLine($"a1 = {a1.member1}, a2 = {a2.member1}");
        a1.member1++;
        Console.WriteLine($"a1 = {a1.member1}, a2 = {a2.member1}");
    }
}
```

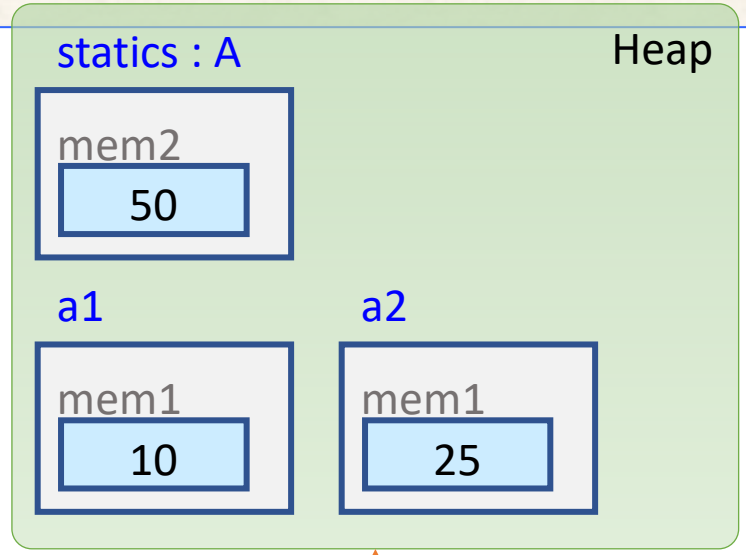


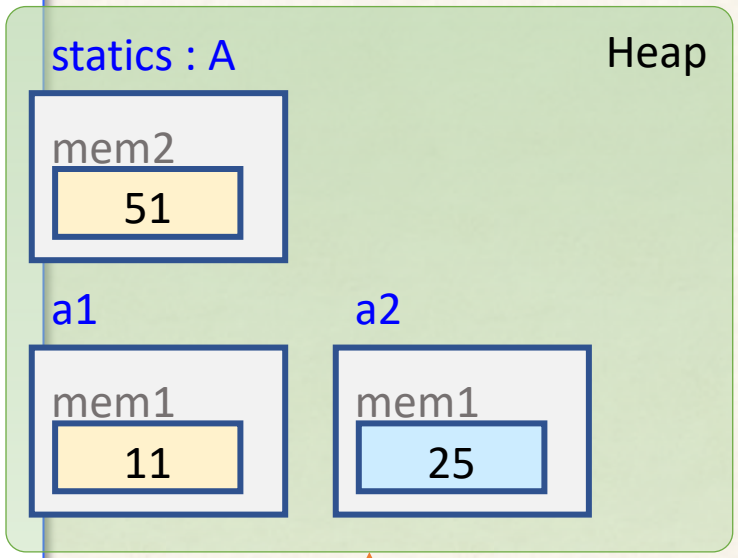
# Instance vs static members

```
class A
{
    public int mem1;
    public static int mem2;
}

class Program
{
    static void Main()
    {
        A a1 = new A();
        A a2 = new A();
        a1.mem1 = 10;
        A.mem2 = 50;
        a2.mem1 = 25;

        Console.WriteLine($"a1 = {a1.mem1}, a2 = {a2.mem1}, A.mem2 = {A.mem2}");
        a1.mem1++;
        A.mem2++;
        Console.WriteLine($"a1 = {a1.mem1}, a2 = {a2.mem1}, A.mem2 = {A.mem2}");
    }
}
```



# Accessing instance vs static members

```
<instance name>.<instance member>;
```

```
<class name>.<static member>;
```

# Lifetimes of Instance vs Static Members

- Lifetime ของ instance member จะเท่ากับ instance
  - เริ่มเมื่อสร้าง instance นั้นด้วยคำสั่ง new
  - สิ้นสุดเมื่อลบ instance นั้น
- Lifetime ของ static member จะไม่ขึ้นกับ instance ใด
  - สามารถเข้าถึง (ตามสิทธิ์) และเรียกใช้ได้ทันที ถึงแม้ว่าไม่มีการสร้าง instance ใดๆ จากคลาสนั้นเลยก็ตาม

# Lifetimes of Static Members

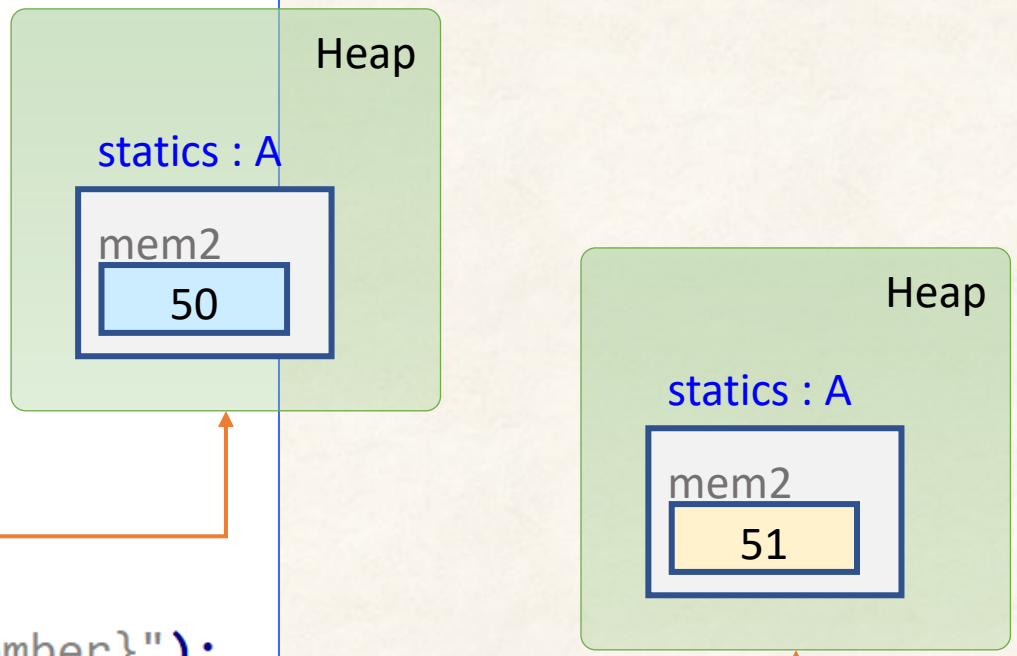
```
class A
{
    public static int member;
}

class Program
{
    static void Main()
    {
        A.member = 50;

        Console.WriteLine($"A.member = {A.member}");

        A.member++;

        Console.WriteLine($"A.member = {A.member}");
    }
}
```



# Static Function Members

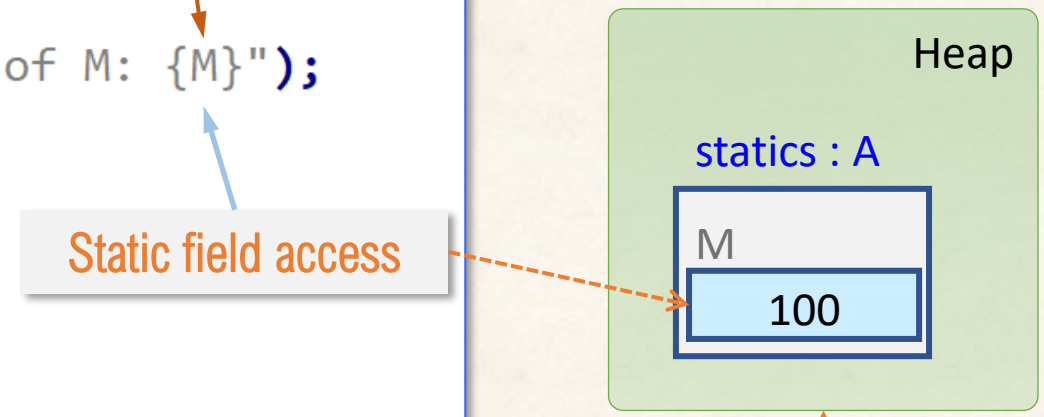
- Static function member มีลักษณะเหมือน Static member อื่น ๆ
  - Static function member สามารถใช้งาน instance member อื่น ๆ
  - Static function member ไม่สามารถใช้งาน static member ใด ๆ ได้

# Static Function Members

```
class A
{
    public static int M;
    public static void PrintValueM()
    {
        Console.WriteLine($"Value of M: {M}");
    }
}

class Program
{
    static void Main()
    {
        A.M = 100;

        A.PrintValueM();
    }
}
```



# Static Function Members

```
class Program
{
    static void Main()
    {
        A.M = 100;
        A.PrintValueM();

        A a1 = new A();

        a1.PrintValueM();
    }
}
```

```
class A
{
    public static int M;
    public static void PrintValueM()
    {
        Console.WriteLine($"Value of M: {M}");
    }
}
```

```
error CS0176: Member 'A.PrintValueM()' cannot be accessed with an instance reference; qualify it with a type name instead
```

# Static Members เป็นอะไรได้บ้าง

- Data Members
  - ✓ Fields
  - ➢ Types
- Function Members
  - ✓ Methods
  - ➢ Properties
  - ➢ Constructors
  - ➢ Operators
  - ➢ Events

# ส่วนประกอบของ Class

| Data members | Function Members |
| --- | --- |
| ✓ Fields | ✓ Methods |
| ➢ Constants | ➢ Properties |
| | ➢ Constructors |
| | ➢ Destructors |
| | ➢ Operators |
| | ➢ Indexers |
| | ➢ Events |

# Members Constants

- Member constants มีลักษณะเหมือน local constants ในหัวข้อที่เรียนมาแล้ว
  - Member constants  เป็นสมาชิกระดับคลาส
  - ต้องกำหนดค่าเริ่มต้นทันทีที่ประกาศ
  - ไม่สามารถเปลี่ยนแปลงค่าได้
  - สามารถใช้ได้จากทุก method ในคลาส
  - Member constants ต้องระบุในขอบเขตของคลาส
    - ในภาษา C# จะไม่สามารถประกาศตัวแปรไว้ภายนอกคลาส
    - ในภาษา C# ไม่สามารถสร้าง global constants ได้

# Members Constants

```
class MyMath
{
    const double PI = 3.14159;
    public static void PrintCircleArea(double radius)
    {
        Console.WriteLine($"R={radius}, Area={radius * radius * PI}");
    }
}

class Program
{
    static void Main()
    {
        MyMath.PrintCircleArea(100);
    }
}
```

# Members Constants

```
class MyMath
{
    const double PI = 3.14159;
    const double PI2 = PI * 2.0;
    public static void PrintCircleArea(double radius)
    {
        Console.WriteLine($"R={radius}, Area={radius * radius * PI}");
    }
    public static void PrintCircleCircumference(double radius)
    {
        Console.WriteLine($"R={radius}, Area={PI2 * radius}");
    }
}

class Program
{
    static void Main()
    {
        MyMath.PrintCircleArea(100);
        MyMath.PrintCircleCircumference(100);
    }
}
```

สามารถใช้ const ใน expression ได้

# Members Constants


```
class MyMath
{
    const double PI = 3.14159;
    public static void PrintCircleArea(double radius)
    {
        Console.WriteLine($"R={radius}, Area={radius * radius * PI}");
    }
    PI = 3.14159265359;

}
```

error CS1519: Invalid token '=' in class, record, struct, or interface member declaration

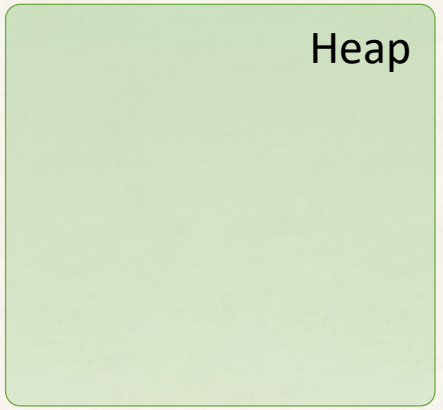
# Constants vs statics

- Member constants มีลักษณะการทำงานคล้าย statics
  - สามารถใช้งานได้กับทุก instance ของคลาส
  - สามารถใช้งานได้โดยไม่ต้องมี instance ใด ๆ ของคลาส
  - Constants ไม่ต้องการพื้นที่เก็บข้อมูล
    - คอมไพเลอร์จะแทนค่าตัวแปรด้วยค่าคงที่ในขณะคอมไพล์

# Constants vs statics

```
class MyMath
{
    public const double PI = 3.14159265359;
}

class Program
{
    static void Main()
    {
        Console.WriteLine($"Value of PI = {MyMath.PI}");
    }
}
```



# Constants vs statics

- ถึงแม้ว่า Member constants จะมีลักษณะการทำงานคล้าย statics แต่ไม่สามารถประกาศ constants ด้วย modifier ‘static’

```
class MyMath
{
    public static const double PI = 3.14159265359;
}

class Program
{
    static void Main()
    {
        Console.WriteLine($"Value of PI = {MyMath.PI}");
    }
}
```

```
error CS0504: The constant 'PI'
cannot be marked static
```

# ส่วนประกอบของ Class

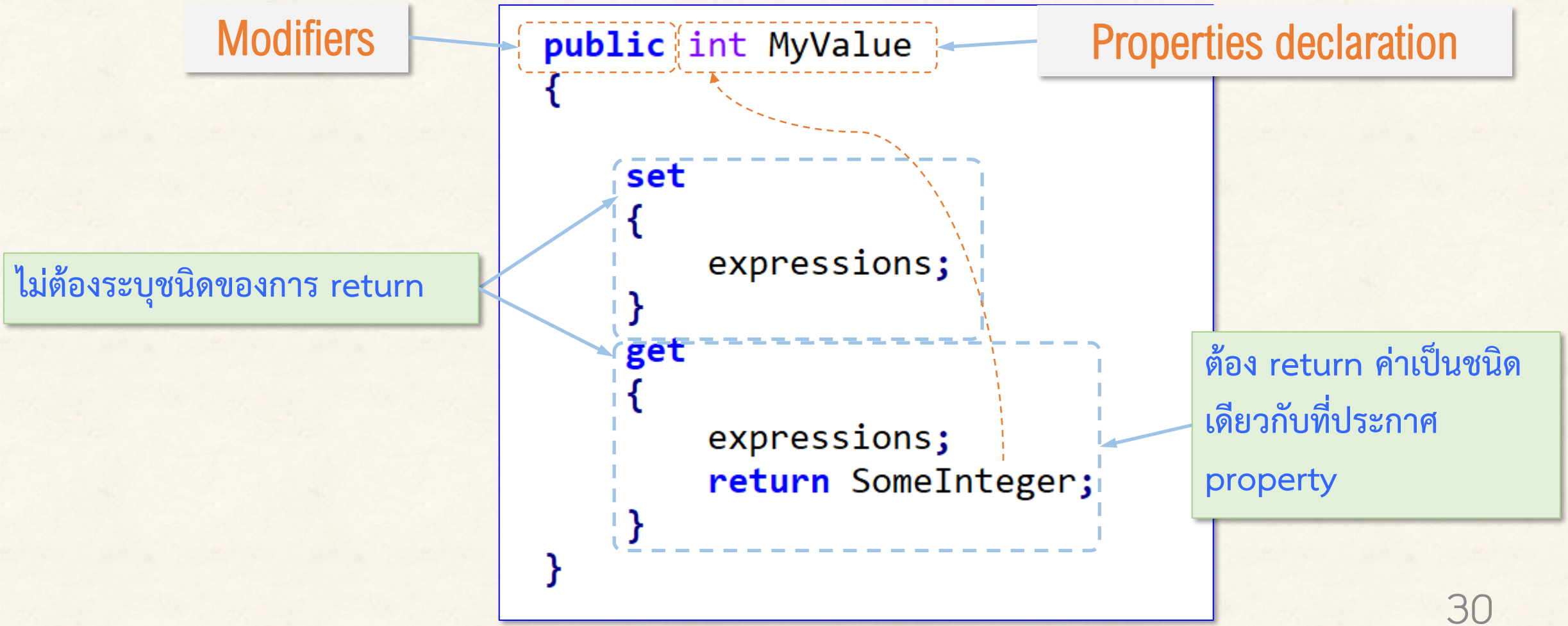
| Data members | Function Members |
| --- | --- |
| ✓ Fields | ✓ Methods |
| ✓ Constants | ➢ Properties |
| | ➢ Constructors |
| | ➢ Destructors |
| | ➢ Operators |
| | ➢ Indexers |
| | ➢ Events |

# Properties

# Properties

- Properties เป็นสมาชิกของคลาส ที่ใช้เก็บข้อมูลในลักษณะเดียวกับ fields
  - ต้องมี type เสมอ
  - ต้องมี identifier ตามกฎการตั้งชื่อ
  - ต้องเป็นสมาชิกของคลาส ไม่สามารถประกาศไว้นอกคลาส
- Properties ต่างจาก fields
  - มี accessor ที่ชื่อ get และ set
    - สามารถกำหนดได้ว่า อ่านได้อย่างเดียว เขียนได้อย่างเดียว หรืออ่านได้-เขียนได้
  - สามารถรันคำสั่ง (statements) ภายใน properties ได้

# Properties declaration



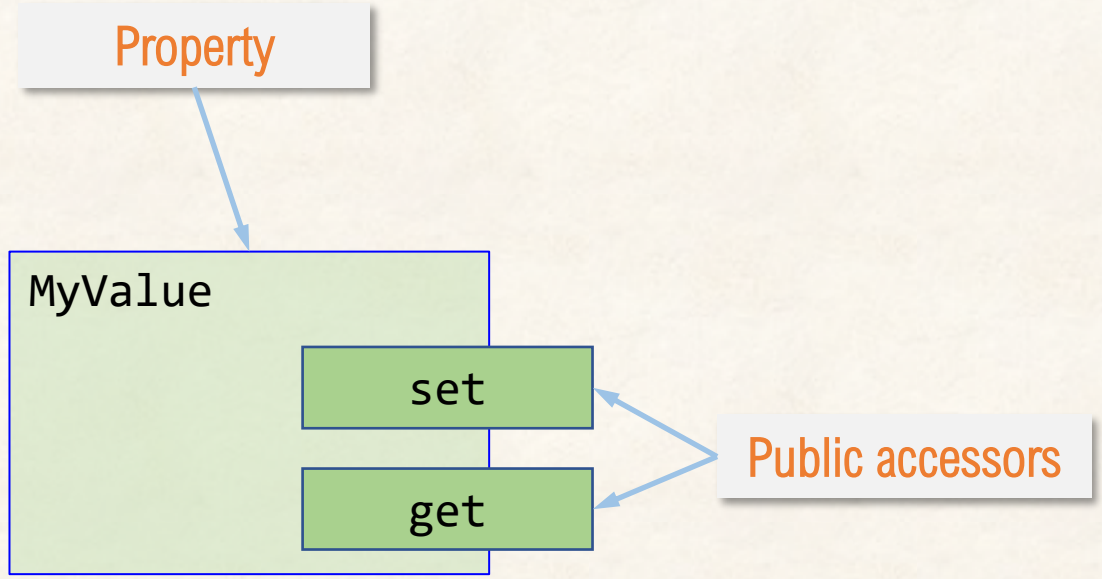
# Properties

- Accessor { get และ set } เป็น method เพียงสองตัวที่สามารถอยู่ใน properties
  - ไม่สามารถมี method ชื่ออื่น ๆ อยู่ใน properties
  - { get และ set } ไม่บังคับการเรียงลำดับก่อน-หลัง
- **ข้อควรระวัง**
  - เมธอด get จะต้องมีคำสั่ง return ในทุกๆ เส้นทาง
    - เช่น  if, else, ฯลฯ ที่เป็นเหตุให้การดำเนินการไม่เป็นไปตามปกติ

# Properties

```
public int MyValue
{

    set
    {
        expressions;
    }
    get
    {
        expressions;
        return SomeInteger;
    }
}
```



# A Properties example

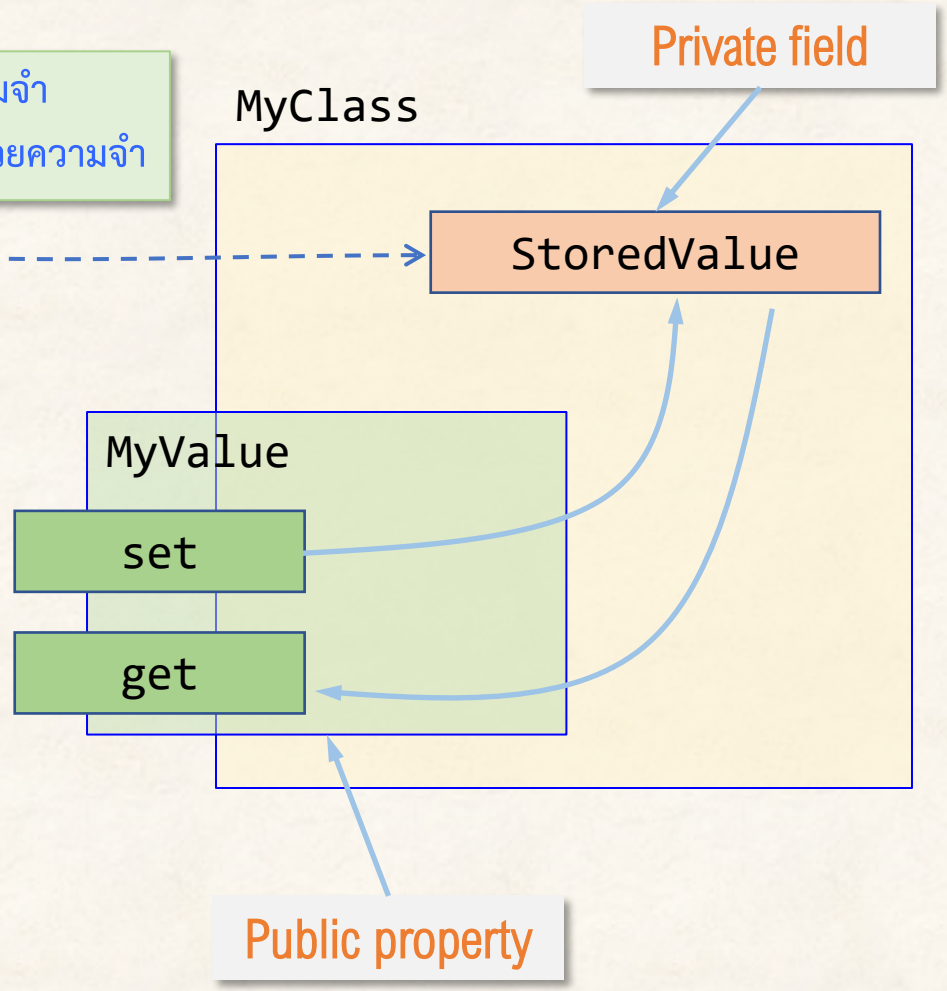
```
class MyClass
{
    private int StoredValue;

    public int MyValue
    {
        set
        {
            StoredValue = value;
        }

        get
        {
            return StoredValue;
        }
    }
}
```

- Field มีการจองหน่วยความจำ
- Property ไม่มีการจองหน่วยความจำ

Input parameter



# Using a property

```
int MyValue
{
    get {...}
    set {...}
}
```

```
int day = MyValue.get();

MyValue.set(30);
```

error

ไม่สามารถใช้ get และ set ในฐานะ method ได้

# Properties calculation

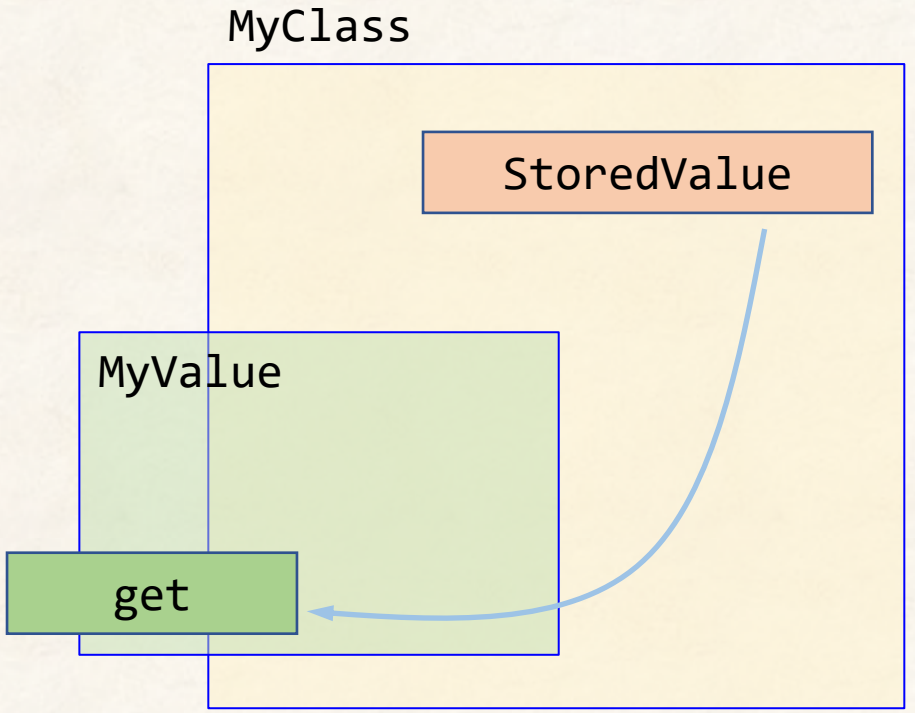
```
class MyClass
{
    private int StoredValue;
    public int MyValue
    {
        set
        {
            if(value > 100 )
                StoredValue = 100;
            else
                StoredValue = value;
        }
        get
        {
            return StoredValue;
        }
    }
}
```

```
class Program
{
    static void Main()
    {
        MyClass C = new MyClass();
        C.MyValue = 120;
        Console.WriteLine($"{C.MyValue}");
    }
}
```

สามารถใส่ expression ต่าง ๆ รวมทั้งการคำนวณใน property ได้

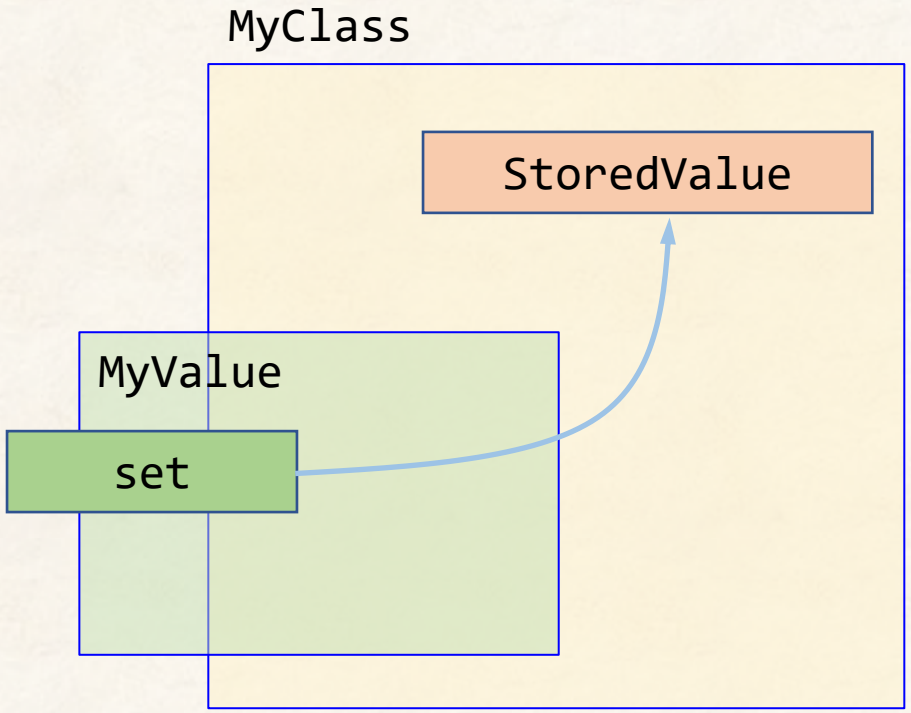
# Read-only Properties

```
class MyClass
{
    private int StoredValue = 10;
    public int MyValue
    {
        get
        {
            return StoredValue;
        }
    }
}
```



# Write-only Properties

```
class MyClass
{
    private int StoredValue;
    public int MyValue
    {
        set
        {
            StoredValue = value;
        }
    }
}
```



# ส่วนประกอบของ Class

| Data members | Function Members |
| --- | --- |
| ✓ Fields | ✓ Methods |
| ✓ Constants | ✓ Properties |
| | ➢ Constructors |
| | ➢ Destructors |
| | ➢ Operators |
| | ➢ Indexers |
| | ➢ Events |

# คำถาม?