

# การเขียนโปรแกรม ด้วยภาษา C#

## Class

# User-Defined Types

- Class types
- Struct types
- Array types
- Enum types
- Delegate types
- Interface types

# Overview of Classes

# Overview of Classes

- ในภาษา C# นั้น Class คือ user-defined type (หนึ่งในจำนวนทั้งหมด 6 ชนิด)
  - ก่อนยุคของ class โปรแกรมคอมพิวเตอร์ถูกเขียนเป็นเชิงโครงสร้าง
  - งานส่วนใหญ่ของการเขียนโปรแกรมเชิงโครงสร้างคือ structuring และ optimizing
  - เมื่อถึงยุคของ Object-Oriented Programming งานหลักจึงเปลี่ยนเป็น organizing เพื่อรวมเอา program และ data ให้เป็นก้อนเดียวกัน (encapsulate)

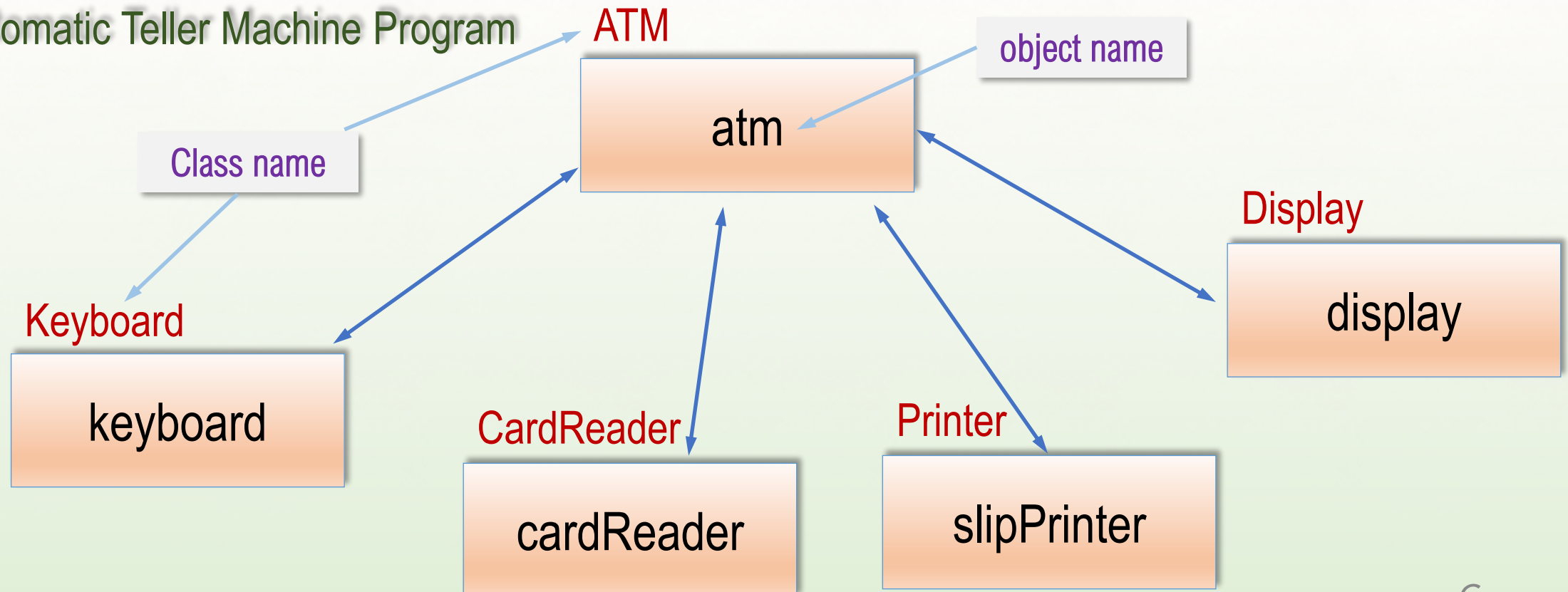
# Class คือ Active Data Structure

- Class คือ type ที่มีข้อมูล (Data members) และฟังก์ชัน (Function members) สำหรับการจัดการกับข้อมูลเหล่านั้น
  - Class ได้มาจากการนิยามข้อมูลของวัตถุในโลกจริง (real-world object)
    - Data members ได้มาจาก attributes ของ real world object
    - Function members ได้มาจากพฤติกรรมหรือความสามารถของ real world object
  - Data members และ Function members ของ Class สามารถมีได้ไม่จำกัดจำนวน
  - ในภาษา C# เรียก data members ว่า fields และเรียก function members ว่า methods

# Classes และ Programs

- โปรแกรมภาษา C# ถูกสร้างขึ้นจาก Classes จำนวนมาก

Automatic Teller Machine Program

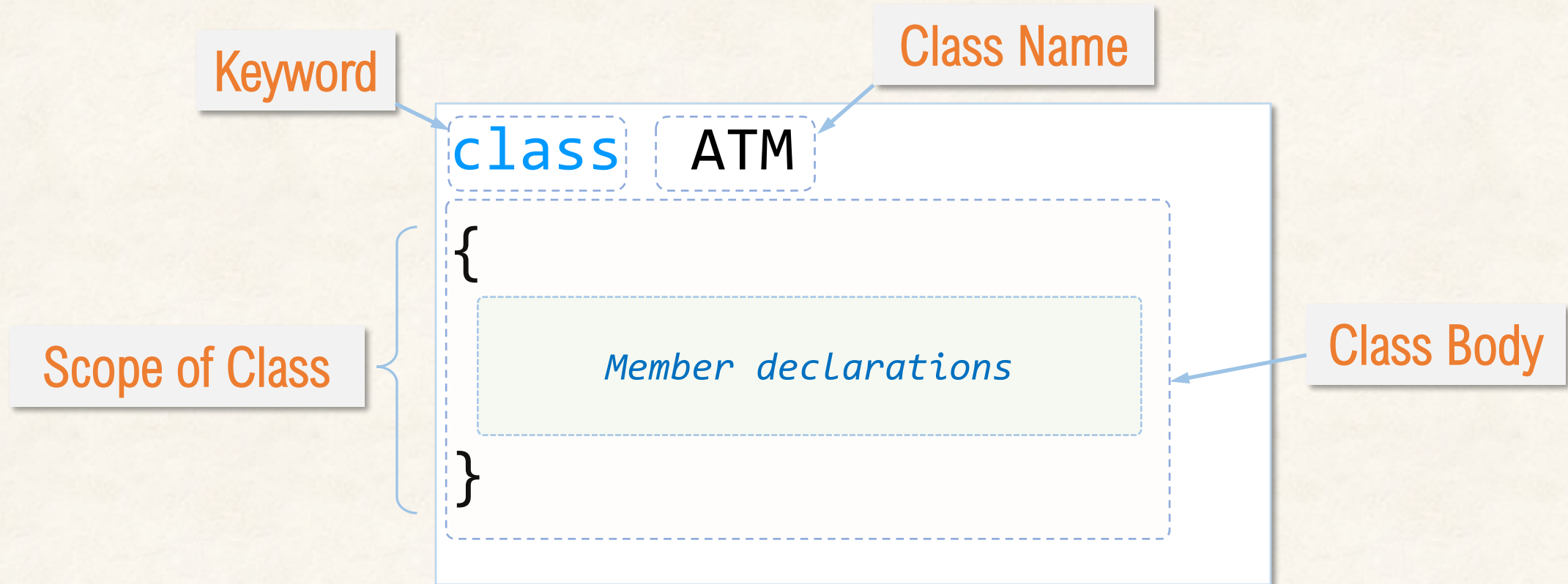




# Declaring a Class

- ในภาษา C# มี predefined type ให้ใช้งานอยู่แล้ว 18 ชนิด เช่น int, float, double, char, ...
  - ถ้าต้องการ type ที่แตกต่างไปจากนั้น เราต้องสร้างขึ้นมา ด้วยกระบวนการ class declaration
- การ declaration ของ class จะต้องประกอบด้วย
  - Keyword “class”
  - ชื่อคลาส (class name)
  - ตัวคลาส (class body) อยู่ภายในขอบเขตของเครื่องหมาย { และ }

# รูปแบบการประกาศ Class





# ส่วนประกอบของ Class

## ENCAPSULATION

### Data members

- Fields
- Constants

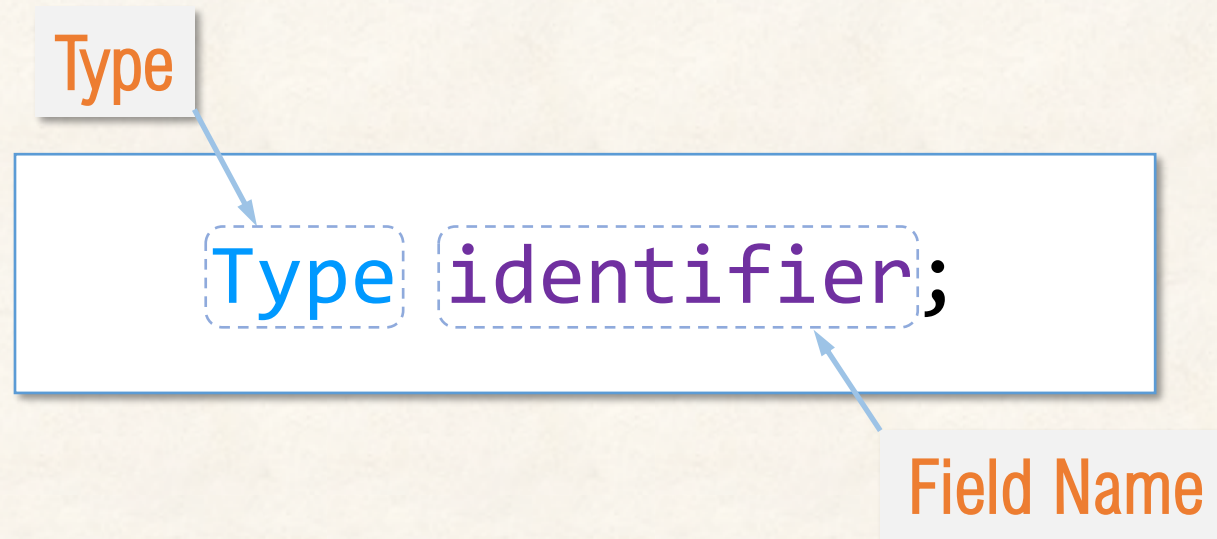
### Function Members

- Methods
- Properties
- Constructors
- Destructors
- Operators
- Indexers
- Events

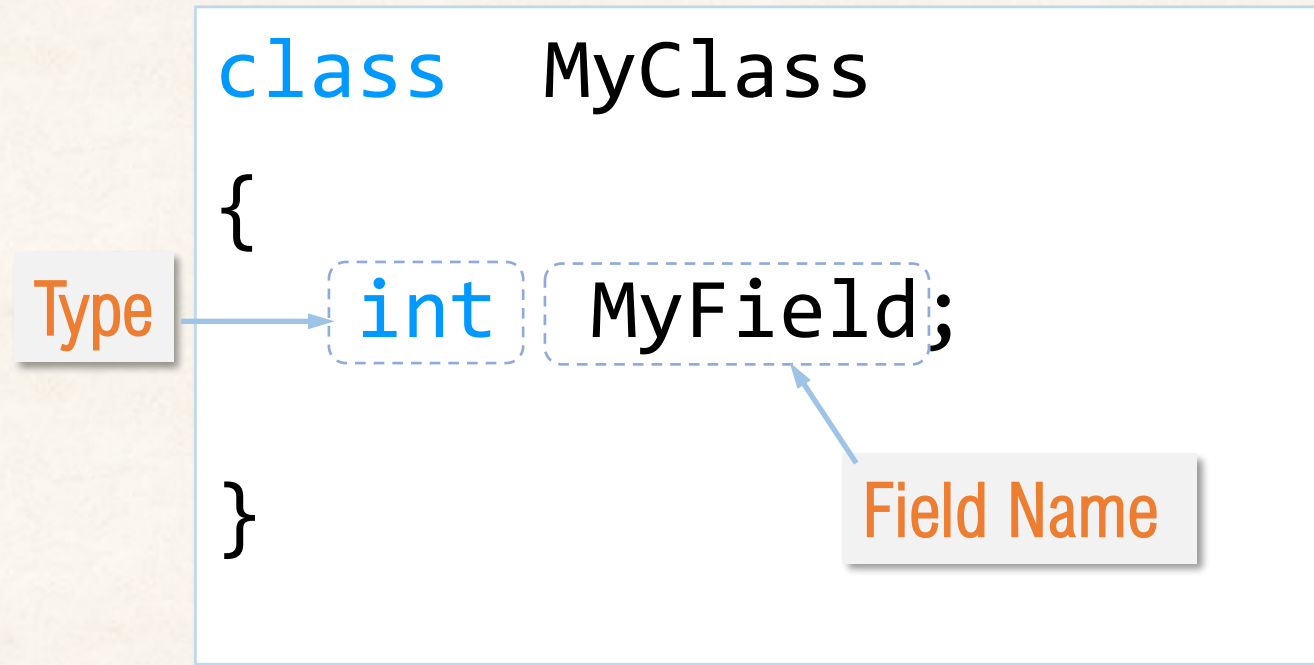
# ส่วนประกอบพื้นฐานของ Class : Fields

- Fields คือส่วนประกอบที่เป็น data member (ตัวแปร) ประจำคลาส
  - ถ้าไม่ใช่ data members ก็จะไม่เรียก fields
  - ยังมีอีกตัวที่เป็น data member คือ ค่าคงที่ (constant)
- Fields สามารถมีชนิดเป็นอะไรก็ได้ (ทั้ง predefined และ user-defined)
- เราสามารถเขียน (written to) และอ่าน (read from) ข้อมูลจาก Fields ได้

# รูปแบบ Fields declaration โดยทั่วไป



# ตัวอย่าง Fields declaration (Data member)



- ในภาษา C# นั้น ไม่สามารถประกาศ fields ไว้ภายนอก type ได้ (ในตัวอย่างนี้ Type ก็คือ Class)
- Fields จะต้องเป็นสมาชิกและมีขอบเขตภายใต้ Type เสมอ

# การกำหนดค่าเริ่มต้นสำหรับ Fields

- Fields คือ Variable ชนิดหนึ่ง
  - กฎเกณฑ์ในการกำหนดค่าเริ่มต้นให้ fields จึงเหมือนกับของ variables
  - การกำหนดค่าให้ field เองนั้นจะเรียกว่า **Explicit field initialization**
    - คอมไพเลอร์จะนำไปใส่ไว้ในหน่วยความจำสำหรับ fields (ในช่วง compile time)
  - หากไม่กำหนดค่าให้ field, คอมไพเลอร์จะเป็นผู้กำหนดค่าเริ่มต้นให้ เรียกว่า **Implicit field initialization**

# ตัวอย่าง Fields declaration (Data member)

Implicit fields initialization

```
class MyClass
```

```
{
```

```
int F1;
```

```
string F2;
```

F1 = 0

F2 = null

Explicit field initialization

```
int F3 = 100;
```

```
string F4 = "ASDF";
```

```
}
```



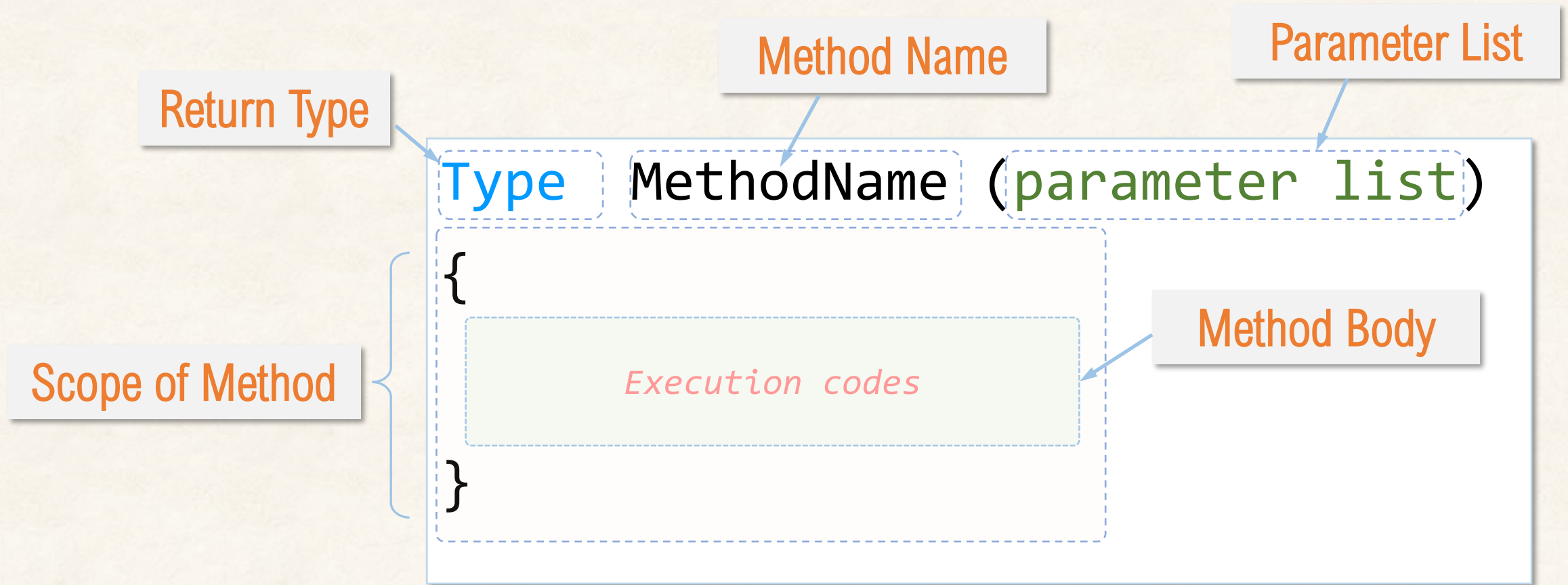
# ส่วนประกอบพื้นฐานของ Class : Methods

- Methods เป็น block ของการประมวลผล ซึ่งเป็นสมาชิกของ Type
  - สามารถเรียกให้ทำงานได้จากส่วนอื่นของ Type, ส่วนอื่นของ program หรือแม้กระทั่งจาก program อื่นๆ
- ชื่อของ method มักจะได้อามาจากพฤติกรรมของวัตถุในโลกจริง (real world object)
  - ในบางครั้ง method ที่ถูกใช้เพียงครั้งเดียว จะถูกสร้างขึ้นเป็น block ณ ที่จุดเรียกใช้ และไม่ได้ตั้งชื่อให้ method นั้น เรียกว่า anonymous method
    - รายละเอียดเรื่อง anonymous method จะอยู่ในหัวข้อแยกไปจากเรื่องนี้

# Methods

- การเรียกให้ method ทำงาน เรียกว่า calling หรือ invocation
  - เมื่อถูกเรียก method จะทำงานตั้งแต่เริ่ม block จนจบ block
  - Block ของ method ถูกกำหนดด้วยเครื่องหมาย { และ }
- Method บางตัว จะส่งค่าที่ได้จากการทำงานกลับไปยังจุดที่ถูกเรียก (return)
- ในภาษา C# เราสามารถส่งค่ากลับผ่านทาง input parameter ได้เช่นกัน

# รูปแบบการประกาศ Methods



# ตัวอย่าง Method

```
class ClassHello
```

Parameter list

```
{
```

Return Type

```
void
```

```
MethodHello ( )
```

```
{
```

```
    Console.WriteLine("1");
```

```
    Console.WriteLine("2");
```

```
    Console.WriteLine("3");
```

```
}
```

```
}
```

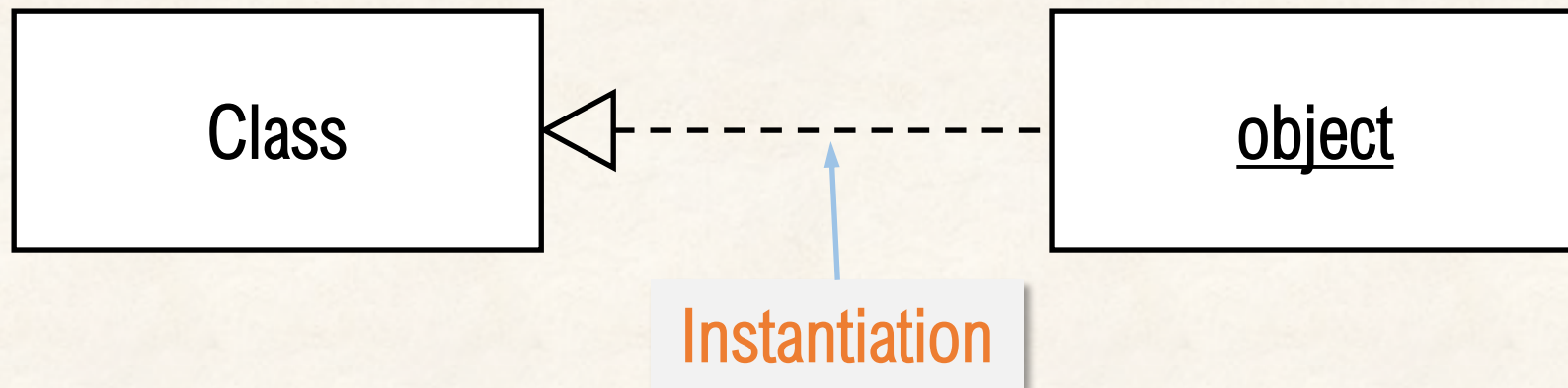
Executable codes

- ในภาษา C# นั้น ไม่สามารถประกาศ methods ไว้ภายนอก type ได้ (ในตัวอย่างนี้ Type ก็คือ Class)
- Methods จะต้องเป็นสมาชิกและมีขอบเขตภายใต้ Type เสมอ

# การใช้งานคลาส

# การสร้าง Objects จาก Class

- Class เป็นเพียงแค่แม่แบบหรือพิมพ์เขียว (blueprint) ของวัตถุ
- เมื่อประกาศ class เสร็จแล้ว เราสามารถนำ blueprint นั้นมาสร้างวัตถุได้
  - เรียกวัตถุที่สร้างจากคลาสนั้นว่า instances ของ class
  - การสร้างวัตถุเรียกว่า instantiation

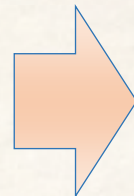




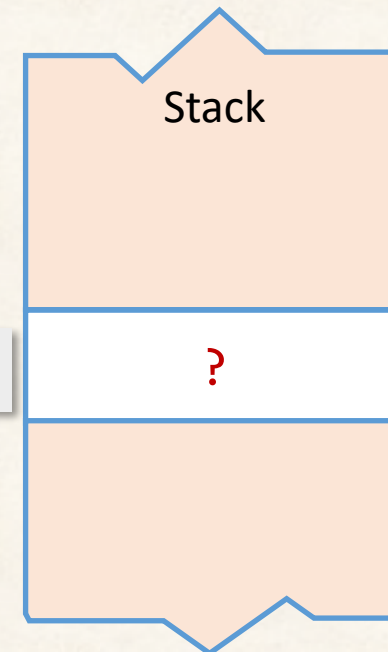
# การสร้าง Variables จาก Class

- Class เป็น reference type แต่เราก็สามารถสร้าง variable จากคลาสได้
  - Variable ที่สร้างขึ้นจากคลาสจะมีค่าเป็น undefined เนื่องจาก reference type จะไม่มี implicit variable initialization

```
class Display { ... }  
  
class ATM  
{  
    static void Main()  
    {  
        Display theDisplay;  
    }  
}
```



theDisplay

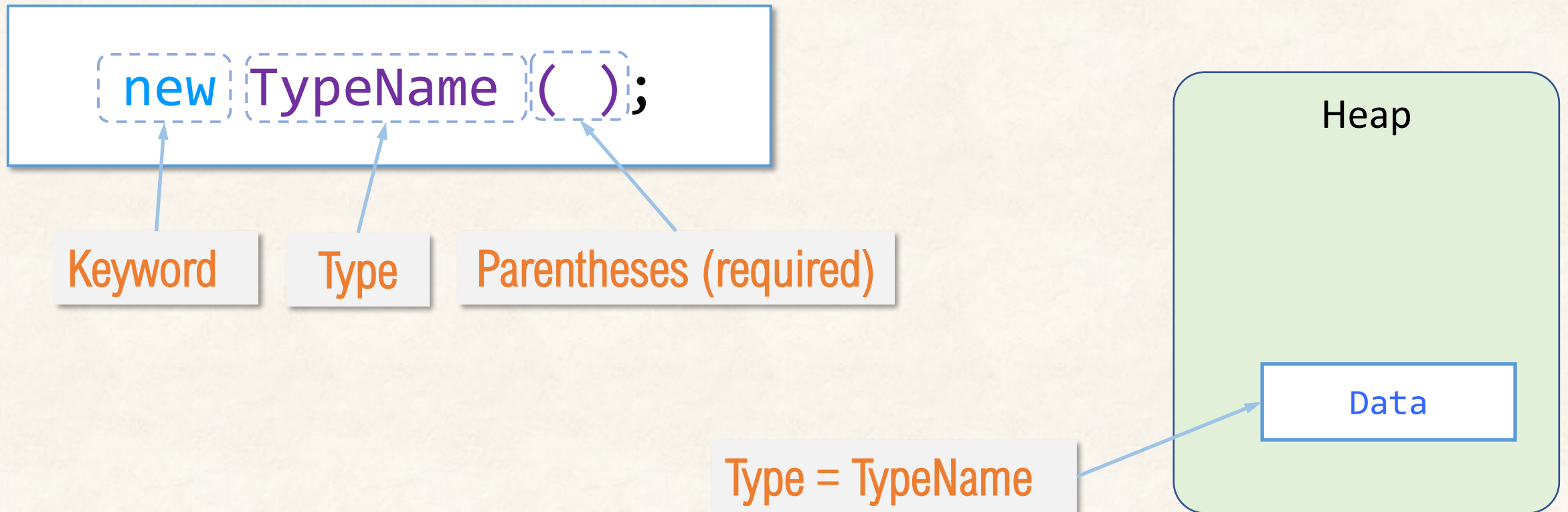


Uninitialized

# การสร้าง Instances ของ Class

- Class เป็น reference type เมื่อสร้าง variable จาก class ก็จะได้เพียง reference ที่ถูกสร้างขึ้นบน stack
  - stack จะมีเนื้อที่จำกัด ไม่สามารถสร้าง instance ซึ่งอาจจะมีขนาดใหญ่ และ/หรือ จำนวนมาก ๆ ได้เพียงพอ
- ถ้าต้องการพื้นที่สำหรับเก็บ data ตาม type ต้องสร้างขึ้นใน heap เรียกว่า memory allocation (ขอใช้พื้นที่ในหน่วยความจำ heap)
- Memory allocation ทำได้โดยการใช้ operator ที่ชื่อว่า “new”

# การสร้าง Instances บนหน่วยความจำ Heap

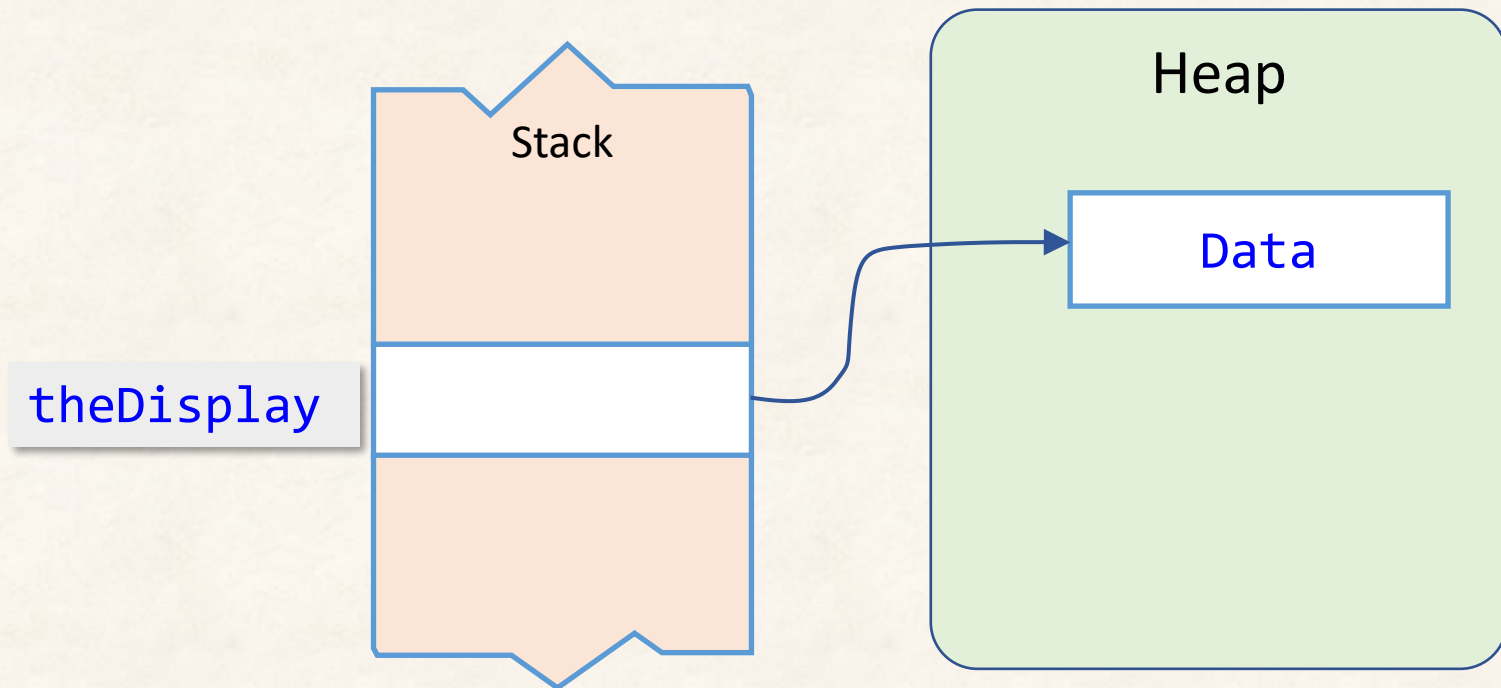


Data ที่สร้างขึ้นนี้จะไม่สามารถใช้งานได้ เนื่องจากยังไม่มี reference

# การเชื่อม Variable และ Instances

- เพื่อให้ใช้งานวัตถุที่สร้างจาก Class เราต้องสร้าง reference บน stack และ instance บน heap แล้วนำมาเชื่อมเข้าด้วยกัน

```
class Display { ... }  
  
class ATM  
{  
    static void Main()  
    {  
        Display theDisplay;  
        theDisplay = new Display();  
    }  
}
```



# การเชื่อม Variable และ Instances (บรรทัดเดียว)

```
class Display { ... }
```

```
class ATM
```

```
{
```

```
    static void Main()
```

```
{
```

```
    Display theDisplay = new Display();
```

```
}
```

```
}
```

Assignment

Reference

Instance

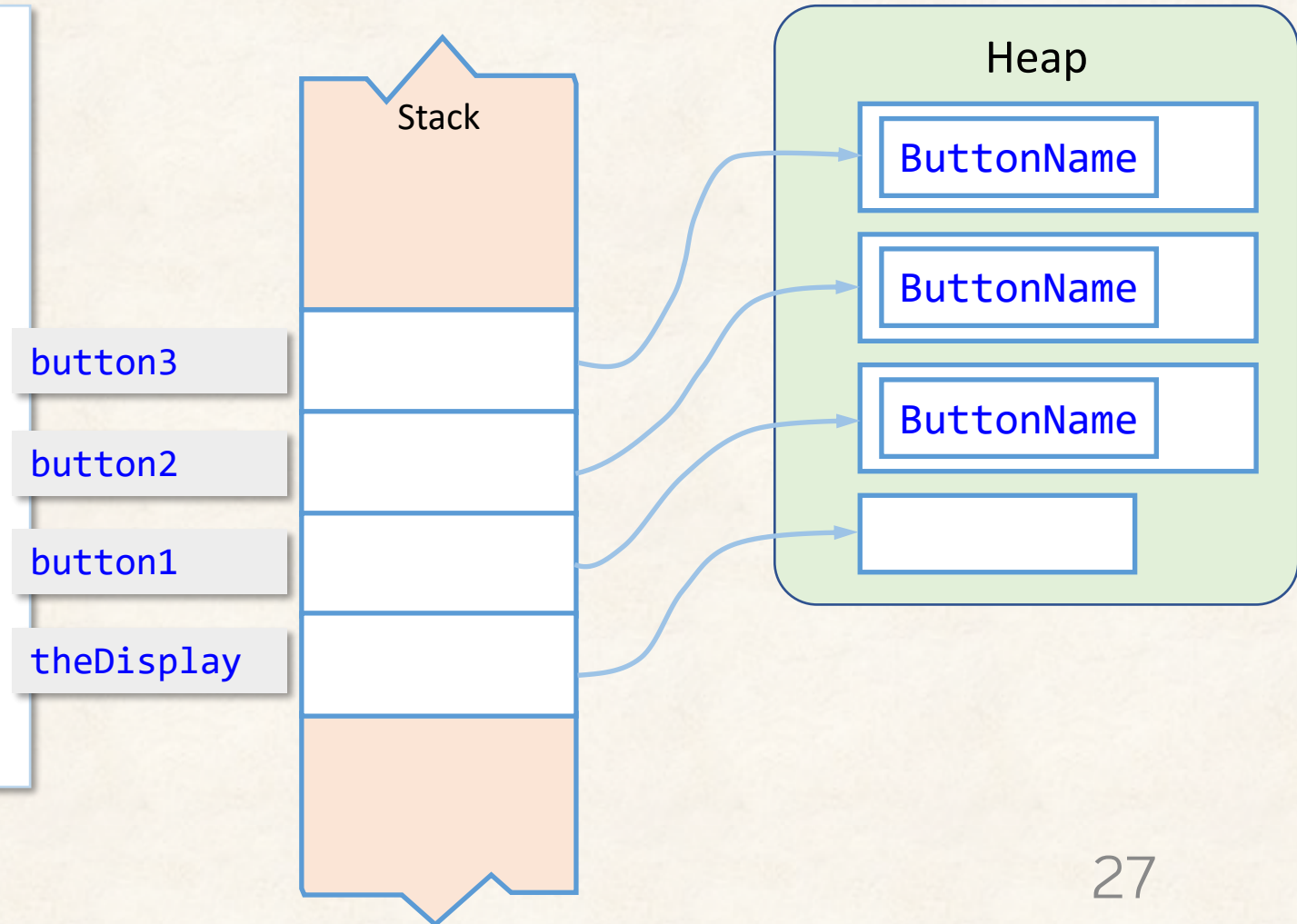
# Instances member และ Static member

- การสร้างคลาส คือการสร้างพิมพ์เขียวของวัตถุ
- ทุกครั้งที่สร้างวัตถุ (instance) จากคลาส member ทุกตัวที่ประกาศไว้ในคลาส จะถูกสร้างขึ้นในขอบเขตของหน่วยความจำ (heap) ที่เราขอมาได้
- แต่จะมีสมาชิกประเภท static ที่จะถูกสร้างเพียงตัวเดียว ไม่ว่าเราจะสร้าง instance กี่ตัวก็ตาม
  - สมาชิกประเภท static สามารถใช้งานได้เลย ไม่ต้องจองพื้นที่ใน heap ด้วย operator 'new'



# Instances member และ Static member

```
class Display { ... }  
class Button  
{  
    string ButtonName;  
}  
  
class ATM  
{  
    static void Main()  
    {  
        Display theDisplay = new Display();  
        Button button1 = new Button();  
        Button button2 = new Button();  
        Button button3 = new Button();  
    }  
}
```



# Access modifiers ของ class members

## Fields

```
AccessModifier Type identifier;
```

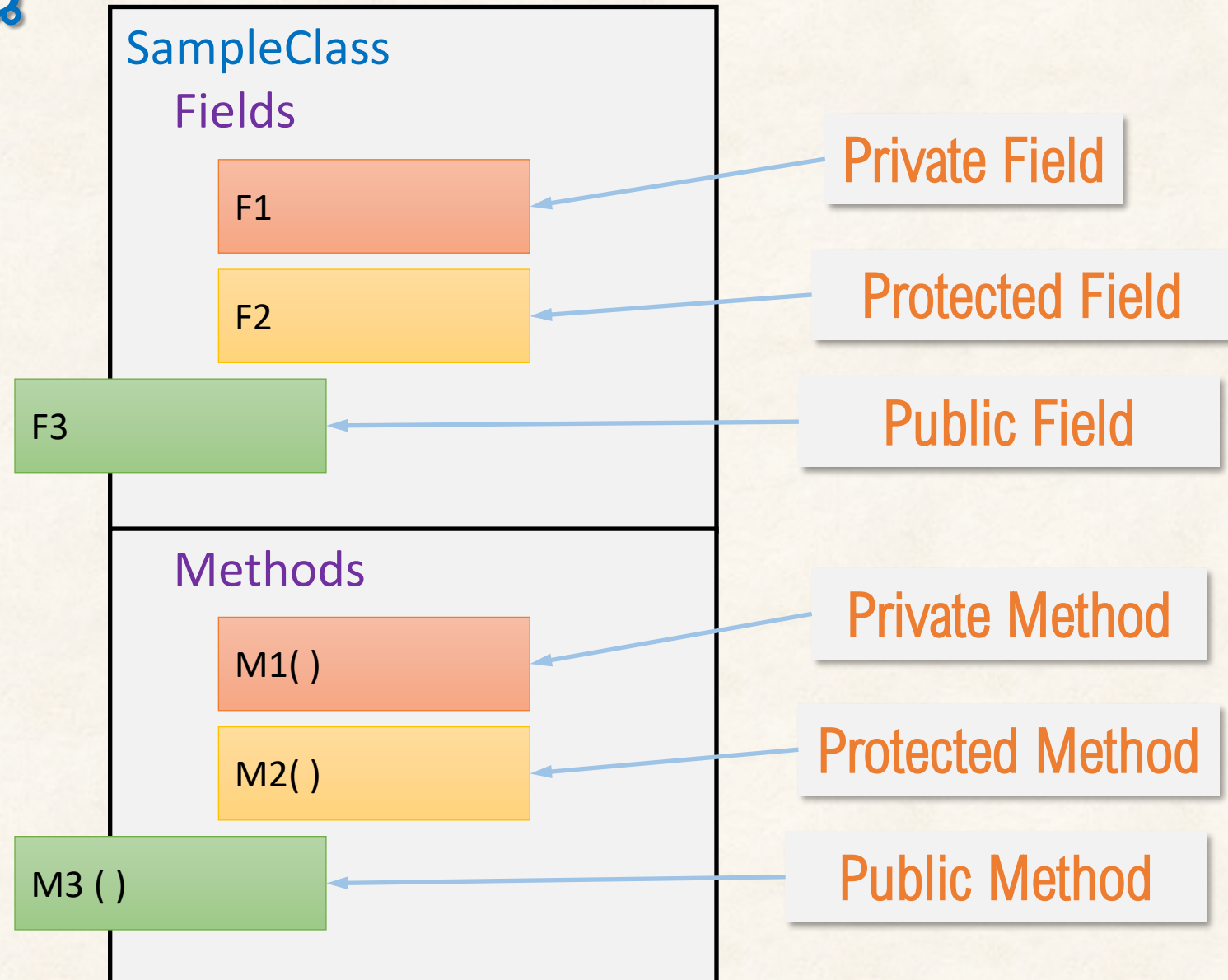
## Methods

```
AccessModifier Type MethodName()  
{  
  
};
```

# Access modifiers มี 5 ประเภท

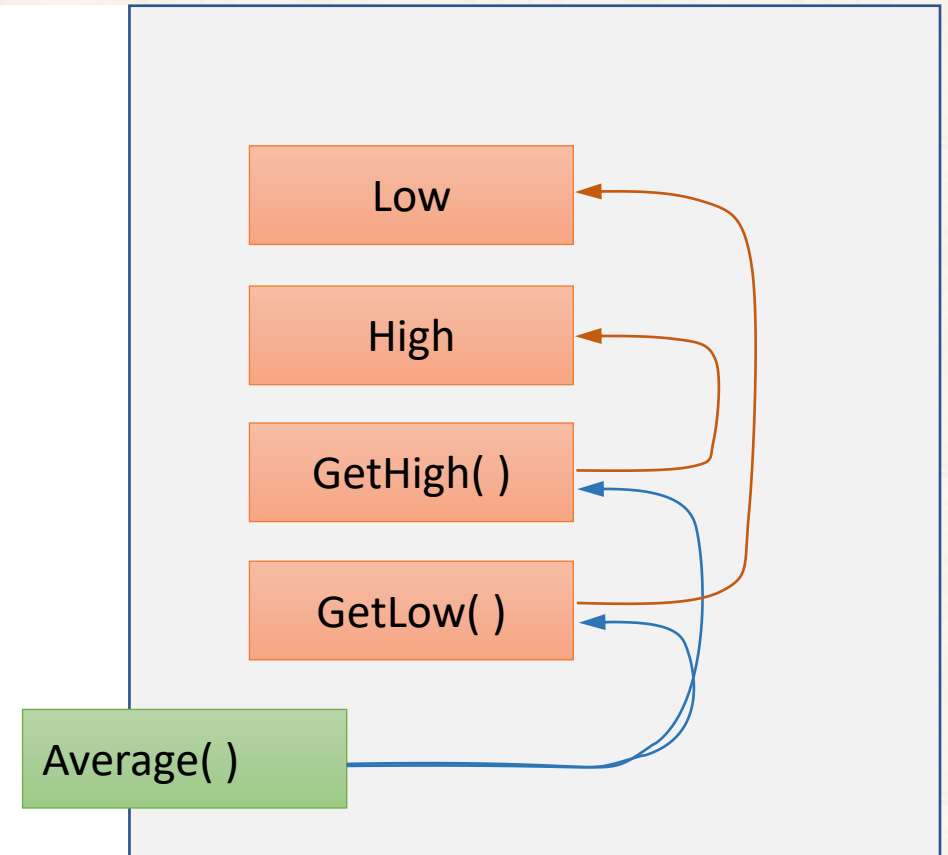
- private
- public
- protected
- internal
- protected internal

# ตกลงกันก่อน

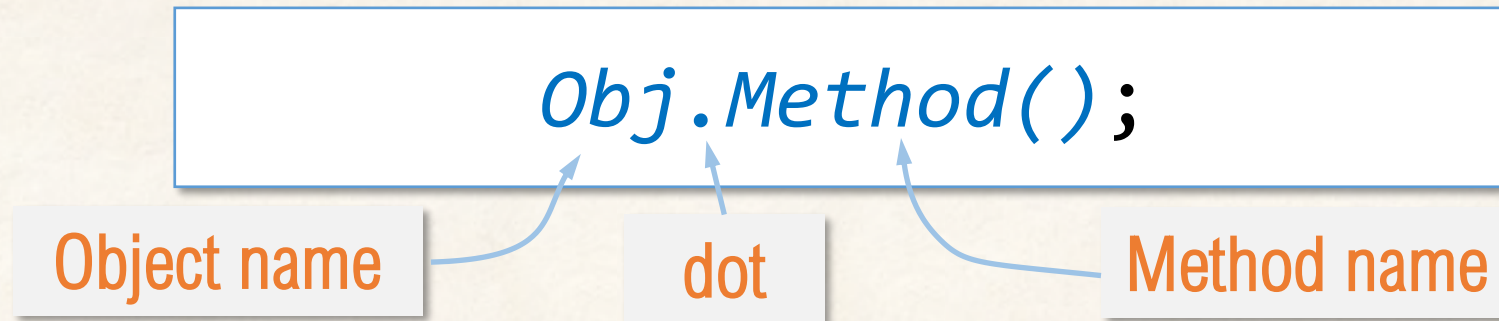


# การเข้าถึง members จากภายในคลาส

```
class DaysTemp
{
    // Fields
    private int High = 75;
    private int Low = 45;
    // Methods
    private int GetHigh()
    {
        return High; // Access private field
    }
    private int GetLow()
    {
        return Low; // Access private field
    }
    public float Average ()
    {
        return (GetHigh() + GetLow()) / 2; // Access private methods
    }
}
```



# การเข้าถึง members จากภายนอกคลาส



```
class Display
{
    public void Render() { ... }
}
class ATM
{
    static void Main()
    {
        Display theDisplay = new Display();
        theDisplay.Render();
    }
}
```



# แบบฝึกหัด

ใน Github