CH. 11

# 類別與物件 (CLASS & OBJECT)

HORAZON

C#程式設計

# 什麼是物件導向 (OBJECT-ORIENTED)

在我們之前的課程中，程式碼多半是「一行一行」或「一個函式」的概念。
但在大型專案 (例如遊戲開發) 中，我們會用**物件導向程式設計 (OOP)** 來思考。

我們將程式中的功能看作是一個個的**物件 (Object)**，它們各自負責不同的工作，並且互相溝通。

# 核心概念：類別 (CLASS) VS 物件 (OBJECT)

這兩個名詞是 OOP 的基礎。

- **類別 (Class)**：藍圖、設計圖、模具。
  - 定義了某種東西「長什麼樣子」、「有什麼功能」。
  - 例如：「設計圖：汽車」。
- **物件 (Object)**：根據藍圖做出來的實體 (Instance)。
  - 真正存在記憶體中，可以使用的東西。
  - 例如：「這台紅色的 Ferrari」、「那台藍色的 Toyota」。

# 範例：定義一個類別

在 C# 中，使用 `class` 關鍵字來定義類別。

```
// 定義一個 "玩家" 類別
class Player
{
    // 1. 成員變數 (Fields)：描述屬性/狀態
    public string name;
    public int hp;

    // 2. 方法 (Methods)：描述行為/功能
    public void Attack()
    {
        Console.WriteLine($"{name} 發動了攻擊！");
    }
}
```

- `public` 代表這個成員可以被外部存取 (稍後會詳談)。

# 範例：建立物件 (NEW)

有了設計圖（`class Player`），我們需要用 `new` 關鍵字將它「做出來」。

```
void Main()
{
    // 建立一個 Player 物件 (p1)
    Player p1 = new Player();
    p1.name = "勇者";
    p1.hp = 100;

    // 建立另一個 Player 物件 (p2)
    Player p2 = new Player();
    p2.name = "魔王";
    p2.hp = 999;

    // 讓物件執行動作
    p1.Attack(); // 輸出：勇者 發動了攻擊！
    p2.Attack(); // 輸出：魔王 發動了攻擊！
}
```

# 記憶體中的運作 (STACK VS HEAP)

這是非常重要的觀念！

- **實值型別 (Value Type)**：如 `int`, `float`, `bool`。
  - 資料直接存放在變數中 (Stack)。
- **參考型別 (Reference Type)**：如 `string`, `Array`, `class`。
  - 變數 ( `p1` ) 只存了一個**地址 (Reference)**。
  - 真正的資料 ( `name`, `hp` ) 存放在記憶體堆積 (Heap) 中。

當你寫 `Player p3 = p1;` 時，其實只是複製了「地址」。
修改 `p3.hp`，`p1.hp` 也會跟著變！(因為它們指向同一個實體)

# 為什麼需要類別？

1. **資料封裝**：將相關的資料（`name`、`hp`、`exp`）綁定在一起，而不是散落在各處的變數。
2. **邏輯統一**：將操作這些資料的函式（`Attack`、`Heal`）也放在一起，更好維護。
3. **擴充性**：可以輕易產生多個獨立的物件 (多個敵人、多個 NPC)。

Unity 中的 `GameObject` 或是 `MonoBehaviour`，本質上全都是類別！

# 總結

- Class (**類別**) 是設計圖，定義屬性與行為。
- Object (**物件**) 是實體，使用 `new` 關鍵字建立。
- 類別包含 Fields (**變數**) 與 Methods (**函式**)。
- 類別是 **參考型別**，變數存的是記憶體地址。

# 下章預告：封裝與建構子

現在我們的變數都是 `public` (公開的)，這其實有點危險！
任何人都可以隨意修改 `hp` 成 `-1000` 或是奇怪的數值。

下一章我們將學習如何保護資料：

- **封裝** (Encapsulation)
- **屬性** (Property)
- **建構子** (Constructor)