Schedule Review

W	Date	Lecture	Homework
1	09/11, 09/15	Lec01: Course Information, Environment Introduction, and OOP Concept	Homework 00 (Fri.)
2	09/18, 09/22	Lec01: Course Information, Environment Introduction, and OOP Concept / Lec02: Class Introduction, and Essential STL Introduction	
3	09/25, **09/29**	Lec02: Class Introduction, and Essential STL Introduction / No class due to Moon Festival	Homework 01 (Mon.)
4	10/02, 10/06	Lec02: Class Introduction, and Essential STL Introduction	
5	**10/09**, 10/13	No class due to the bridge holiday of Nation day / Lec03: Encapsulation	Homework 02 (Mon.)
6	10/16, 10/20	Lec03: Encapsulation / Lec04: Inheritance	
7	10/23, 10/27	Lec04: Inheritance	Homework 03 (Mon.)
8	10/30, 11/03	Lec04: Inheritance	
9	**11/06**, **11/10**	**Physical Hand-Written Midterm** / **Physical Computer-based Midterm**	
10	11/13, 11/17	Lec05: Polymorphism	
11	11/20, 11/24	Lec05: Polymorphism	Homework 05 (Mon.)
12	11/27, 12/01	Lec05: Polymorphism	
13	12/04, 12/08	Lec06: Composition & Interface	Homework 06 (Mon.)
14	12/11, 12/15	Lec06: Composition & Interface	
15	12/18, 12/22	Lec07: Efficiency + Dependency Injection	Homework 07 (Mon.)
16	12/25, 12/29	**Physical Hand-Written Final**, Flexible time	
17	**01/01**, 01/05	No class / **Physical Computer-based Final**	
18	01/08, 01/12	**No class here**	

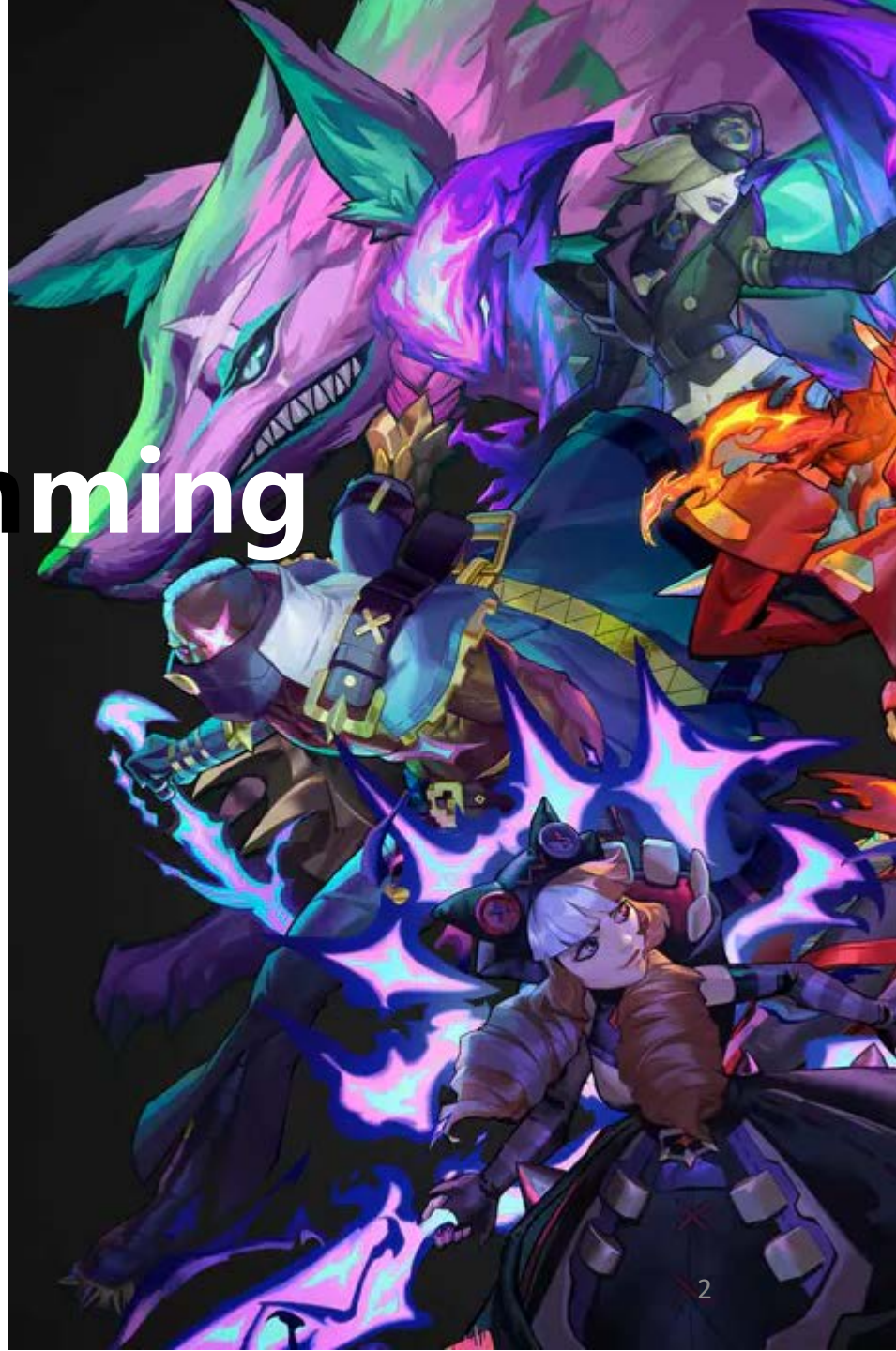
Object Oriented Programming

Lec04: Inheritance

Sun Chin-Yu（孫勤昱）

cysun@ntut.edu.tw

2023/10/20



概念：父類別（Parent）與子類別（Children）

- 看起來很酷，不過... 我要怎麼初始化這些成員？

```cpp
#include <string>

#include "mage.h"

class Zoe : Mage {
public:
    /* Zoe-Only Skill */
    void MoreSparkles();
    void PaddleStar();
    void SpellThief();
    void SpeelyTroubleBubble();
};
```

```cpp
#include "character.h"

class Mage : Character {
private:
    int abilityPower;
public:
    /* Getter */
    int GetAbilityPower();

    /* Setter */
    void SetAbilityPower();

    /* Wizard-only */
    void WizardLevelUp();
};
```

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;
public:
    /* Getter */
    std::string GetName();
    int GetHealth();

    /* Setter */
    void SetName();
    void SetHealth();
};
```

初始化列表（Initialize List）與為何需要初始化列表

- 在這個章節，我們將嘗試解決繼承在初始化成員的行為。

初始化列表（Initialize List）與為何需要初始化列表

- 如果我們幫阿璃加上建構子，你會發現...

```cpp
#include <string>

#include "mage.h"

class Ahri : Mage {
public:
    Ahri(int abilityPower, std::string name, int health);
    /* Ahri-Only Skill */
    void OrbOfDescription();
    void FoxFire();
    void Charm();
    void SpiritRush();
};
```

```cpp
Ahri::Ahri(int abilityPower, std::string name, int health) {
    this->abilityPower = abilityPower
}

void Ahri:
```

'abilityPower' is a private member of 'Mage' clang(access)

mage.h(5, 9): Declared private here

Class Character

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;
public:
    Character(std::string name, int healt

    /* Getter */
    std::string GetName();
    int GetHealth();

    /* Setter */
    void SetName(std::string name);
    void SetHealth(int health);
};
```

Class Mage

```cpp
#include "character.h"

class Mage : Character {
private:
    int abilityPower;
public:
    Mage(int abilityPower, std::string name, int health);

    /* Getter */
    int GetAbilityPower();

    /* Setter */
    void SetAbilityPower(int abilityPower);

    /* Wizard-only */
    void WizardLevelUp();
};
```

- `abilityPower` 是 `Mage` 的私有成員，所以 `Ahri` 無法存取

初始化列表（Initialize List）與為何需要初始化列表

- 想一下解決方法?
- Character 與 Mage 通常都會有建構子
  - Character 一定會需要血量與名稱，所以建構子一定要有血量與名稱
  - Mage 一定會需要魔法攻擊、血量與名稱，所以建構子一定要有魔法攻擊、血量與名稱

Class Character

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;
public:
    Character(std::string name, int health);

    /* Getter */
    std::string GetName();
    int GetHealth();

    /* Setter */
    void SetName(std::string name);
    void SetHealth(int health);
};
```

Class Mage

```cpp
#include "character.h"

class Mage : Character {
private:
    int abilityPower;
public:
    Mage(int abilityPower, std::string name, int health);

    /* Getter */
    int GetAbilityPower();

    /* Setter */
    void SetAbilityPower(int abilityPower);

    /* Wizard-only */
    void WizardLevelUp();
};
```

初始化列表（Initialize List）與為何需要初始化列表

- 如果我們可以透過建構子來進行初始化呢?
  - Mage 自己私有的成員在自己的建構子程式區塊中初始化
  - Mage 使用 Character 的建構子來初始化 Character 的血量與名稱

Class Character

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;
public:
    Character(std::string name, int health);

    /* Getter */
    std::string GetName();
    int GetHealth();

    /* Setter */
    void SetName(std::string name);
    void SetHealth(int health);
};
```

Class Mage

```cpp
#include "character.h"

class Mage : Character {
private:
    int abilityPower;
public:
    Mage(int abilityPower, std::string name, int health);

    /* Getter */
    int GetAbilityPower();

    /* Setter */
    void SetAbilityPower(int abilityPower);

    /* Wizard-only */
    void WizardLevelUp();
};
```

初始化列表（Initialize List）與為何需要初始化列表

原本要初始化的參數

```
1  Mage::Mage(int abilityPower, std::string name, int health) : Character(name, health){
2      this->abilityPower = abilityPower;
3  }
```

- 上圖的 ": Character(name, health)"，我們稱為 Initialize List
  - 冒號後面代表指定哪個父類別的建構子
- 使用 Character 的建構子來初始化 Character 的成員，讓 Character 的成員保持私有又能夠被子類給初始化

初始化列表（Initialize List）與為何需要初始化列表

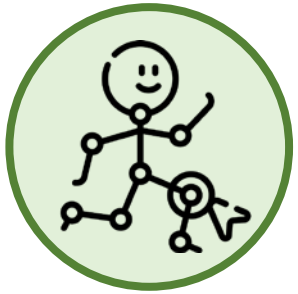
- 但不能是這樣!

```
1  Mage::Mage(int abilityPower, std::string name, int health){
2      Character(name, health);
3      this->abilityPower = abilityPower;
4  }
```

- 這個動作等同於建構一個 Character 物件，與我們預期要初始化父類成員是完全不同的結果

初始化列表（Initialize List）與為何需要初始化列表

- 回到阿璃的例子，我們要怎麼在建構阿璃時初始化阿璃的父類 Mage 與 Character 呢?



```cpp
Character::Character(std::string name, int health) {
    this->name = name;
    this->health = health;
}
```

```cpp
Mage::Mage(int abilityPower, std::string name, int health) : Character(name, health){
    this->abilityPower = abilityPower;
}
```

阿璃 Ahri

```cpp
Ahri::Ahri(int abilityPower, std::string name, int health) : Mage(abilityPower, name, health) {

}
```

順序問題

- 初始化衍生類別物件
  - 導致一連串的建構子呼叫與處理
  - 衍生類別建構子會呼叫基礎類別建構子

```
Character::Character(std::string name, int health) {
    this->name = name;
    this->health = health;
}
```

```
Mage::Mage(int abilityPower, std::string name, int health) : Character(name, health){
    this->abilityPower = abilityPower;
}
```

阿璃 Ahri

```
Ahri::Ahri(int abilityPower, std::string name, int health) : Mage(abilityPower, name, health) {

}
```

	呼叫順序	完成順序
Character	③	①
Mage	②	②
Ahri	①	③

IRVINE, C++ AND OBJECT-ORIENTED PROGRAMMING, PRENTICE-HALL,INC.

權限控制：protected

- 在這個章節，我們將嘗試解決無法存取父類成員的問題。

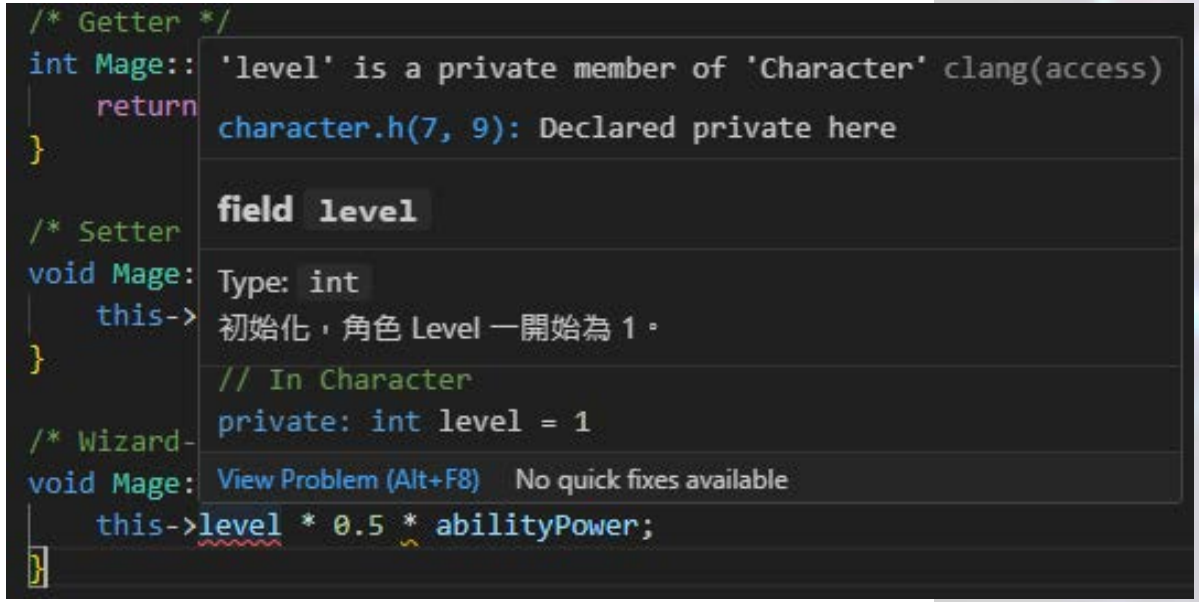
權限控制：protected

- 還記得 WizardLevelUp() 這個函數嗎，我們要開始設計這個函數
- 如果要設計WizardLevelUp() 這個函數，原先的code少了什麼?
  - 但由於角色還沒有 Level 這個成員，所以我們把它加上去

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;

public:
    Character(std::string name, int health);

    /* Getter */
    std::string GetName();
    int GetHealth();

    /* Setter */
    void SetName(std::string name);
    void SetHealth(int health);
};
```

權限控制：protected

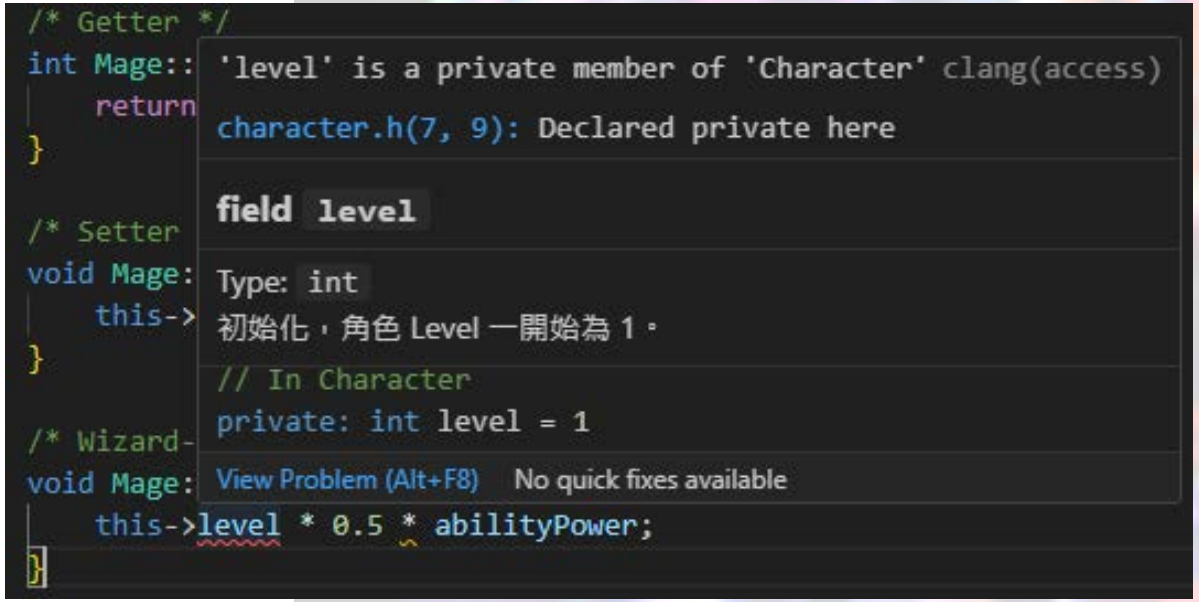
- 接下來我們開始設計 WizardLevelUp() 這個函數
- 請問WizardLevelUp()應該實作在哪一份code中?
  - 在Mage.hpp/Mage.cpp
  - 我們期望每升級一等，角色會得到 $Level \times 0.5 \times AbilityPower$ 的魔力加成



- 我們沒有辦法直接拿到 level，因為 level 是 Character 的 private member
- **怎麼辦呢?**

權限控制：protected

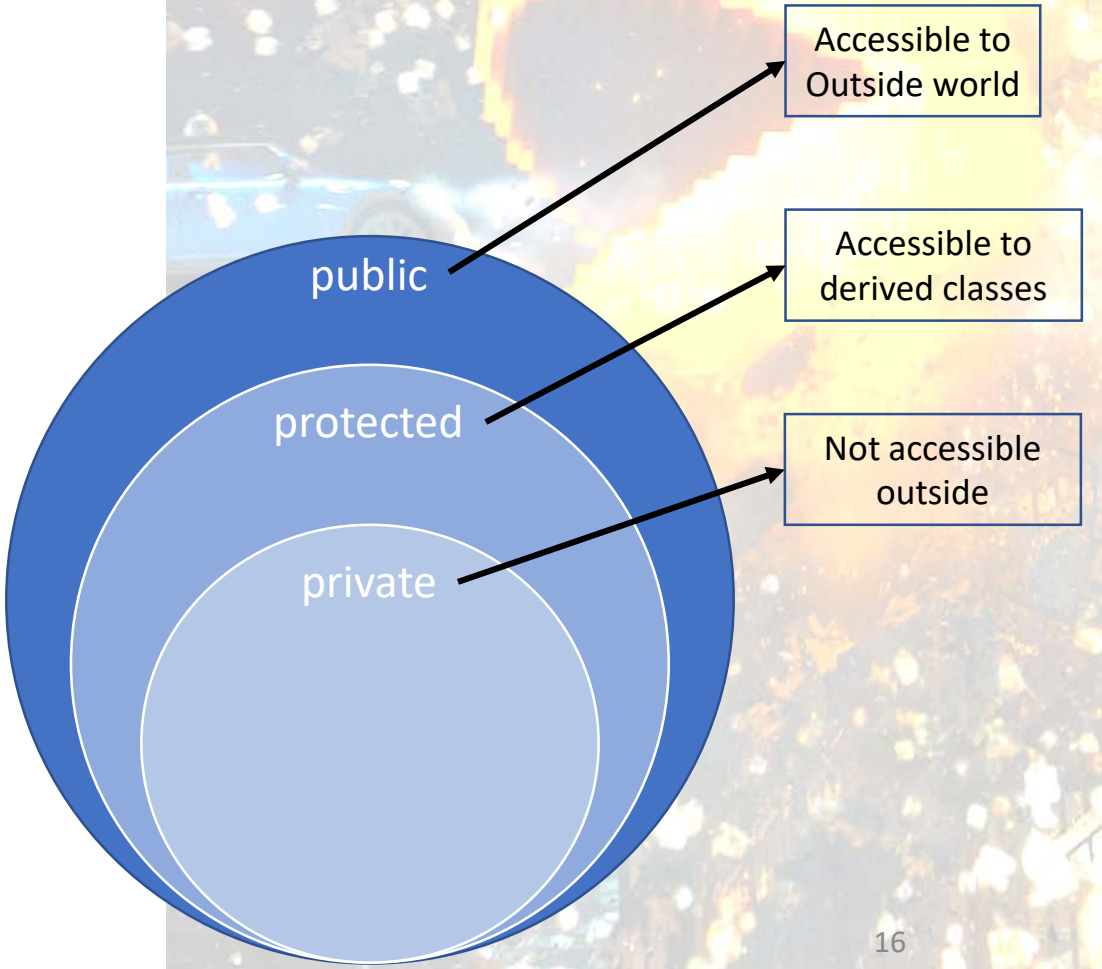
- 我們把 Character 中的 level 設定成 public，完成!



- 但我們不可能讓 Character 的 level 直接 public，這樣違反了封裝性
- 有沒有什麼辦法可以「只讓子類別存取、不讓外部存取」呢?

封裝：存取修飾字

- 要做到資訊隱藏（Information hiding）可以使用存取修飾字
- 分為：
  - Public：是指對存取權限完全的公開，任何物件都可以存取
  - Private：是限制最嚴格的access modifier，只有在類別本身內部可以存取
  - **Protected：存取是受限的，除了類別自己可以使用外，只有其成員函式與類別的朋友（friend）以及子類別可以存取**
    - **「繼承」的時候介紹**



權限控制：protected

- 對於這個例子，我們可以使用 protected 來使父類的成員能夠被子類所存取
- 透過這樣的方式，Mage 就能夠取得 Character 的 Level，進行 WizardLevelUp() 的運算

```cpp
/* Wizard-only */
void Mage::WizardLevelUp(){
    this->level * 0.5 * abilityPower;
}
```

- 但...這樣就結束了嗎?

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;
protected:
    int level = 1; // 初始化，角色 Level 一開始為 1。
public:
    Character(std::string name, int health);

    /* Getter */
    std::string GetName();
    int GetHealth();
    int GetLevel(); // 取得等級

    /* Setter */
    void SetName(std::string name);
    void SetHealth(int health);
    void LevelUp(); // 升級
};
```

權限控制：protected

- 記得封裝性嗎？如果我們讓 level 放入 protected 其實依然是稍微違反封裝性的
  - 子類別可以直接更改父類別的內部狀態，這可能會導致數據不一致或其他不可預見的錯誤
  - 因為子類能夠任意改動 level 的值，對於「我們希望 Mage 僅讀 level 的值」意願是違反的

- 所以我們應該要繼續遵循使用函數來取得 level 值，也就是從 GetLevel() 下手

```cpp
#include <string>

class Character {
private:
    std::string name;
    int health;
    int level = 1; // 初始化，角色 Level 一開始為 1。
public:
    Character(std::string name, int health);

    /* Getter */
    std::string GetName();
    int GetHealth();
    int GetLevel(); // 取得等級

    /* Setter */
    void SetName(std::string name);
    void SetHealth(int health);
    void LevelUp(); // 升級
};
```

權限控制：protected

- 記得封裝性嗎？如果我們讓 level 放入 protected 其實依然是*稍微*違反封裝性的。
  - 子類別可以直接更改父類別的內部狀態，這可能會導致數據不一致或其他不可預見的錯誤
  - 因為子類能夠任意改動 level 的值，對於「我們希望 Mage 僅讀 level 的值」意願是違反的

- 所以我們應該要繼續遵循使用函數來取得 level 值，也就是從 GetLevel() 下手

- 那為什麼我們需要 protected?
  - 若我們希望**某函式**不應該被外部存取，但希望能夠在子類中被存取，我們應該幫該函式設為 protected
  - 這樣不稍微違反封裝性，但又能體現 protected 的用意

```
1  #include <string>
2
3  class Character {
4  private:
5      std::string name;
6      int health;
7      int level = 1; // 初始化，角色 Level 一開始為 1。
8  public:
9      Character(std::string name, int health);
10
11     /* Getter */
12     std::string GetName();
13     int GetHealth();
14     int GetLevel(); // 取得等級
15
16     /* Setter */
17     void SetName(std::string name);
18     void SetHealth(int health);
19     void LevelUp(); // 升級
20 };
```

權限控制：protected（Bonus）

Manager

新遊戲角色目標：
「放越多次技能，法術傷害越高」
or
「每放100次Q技能，法術傷害+1」

- 考慮到我們希望在遊戲上，**統計使用技能的數量**，並在未來供 WizardLevelUp() 當作提升數值使用
- 所以 Character 紀錄了使用技能的數量（SkillUsageCount）

```cpp
//
// Created by 黃漢軒 on 2023/10/9.
//

#ifndef OOP_CHARACTER_HPP
#define OOP_CHARACTER_HPP

#include <string>

class Character {
private:
    std::string name;
    int health;
public:
    Character(std::string name, int health);

    std::string GetName();
    int GetHealth();

    void SetName(std::string name);
    void SetHealth(int health);

    void SkillUsageCount();
};

#endif // OOP_CHARACTER_HPP
```

(更正：回傳不會是void)

權限控制：protected（Bonus）

- 但是 SkillUsageCount 是一個必須要公開的東西嗎?
  - 你玩LOL會去看QWER放過幾次嗎?

- 一般來說，玩家不需要知道技能被用了幾次
  - SkillUsageCount 的用意單純只是想要在 Mage 中用來當作升級時增加魔力值的依據

- 因此，這個 Function 不應該被公開，但應該要能夠被
  - Mage類別存取（用於WizardLevelUp()）
  - 特定子類別所存取（用於特殊角色）

```
1  //
2  // Created by 黃漢軒 on 2023/10/9.
3  //
4
5  #ifndef OOP_CHARACTER_HPP
6  #define OOP_CHARACTER_HPP
7
8  #include <string>
9
10 class Character {
11 private:
12     std::string name;
13     int health;
14 public:
15     Character(std::string name, int health);
16
17     std::string GetName();
18     int GetHealth();
19
20     void SetName(std::string name);
21     void SetHealth(int health);
22
23     void SkillUsageCount();
24 };
25
26 #endif // OOP_CHARACTER_HPP
27
```

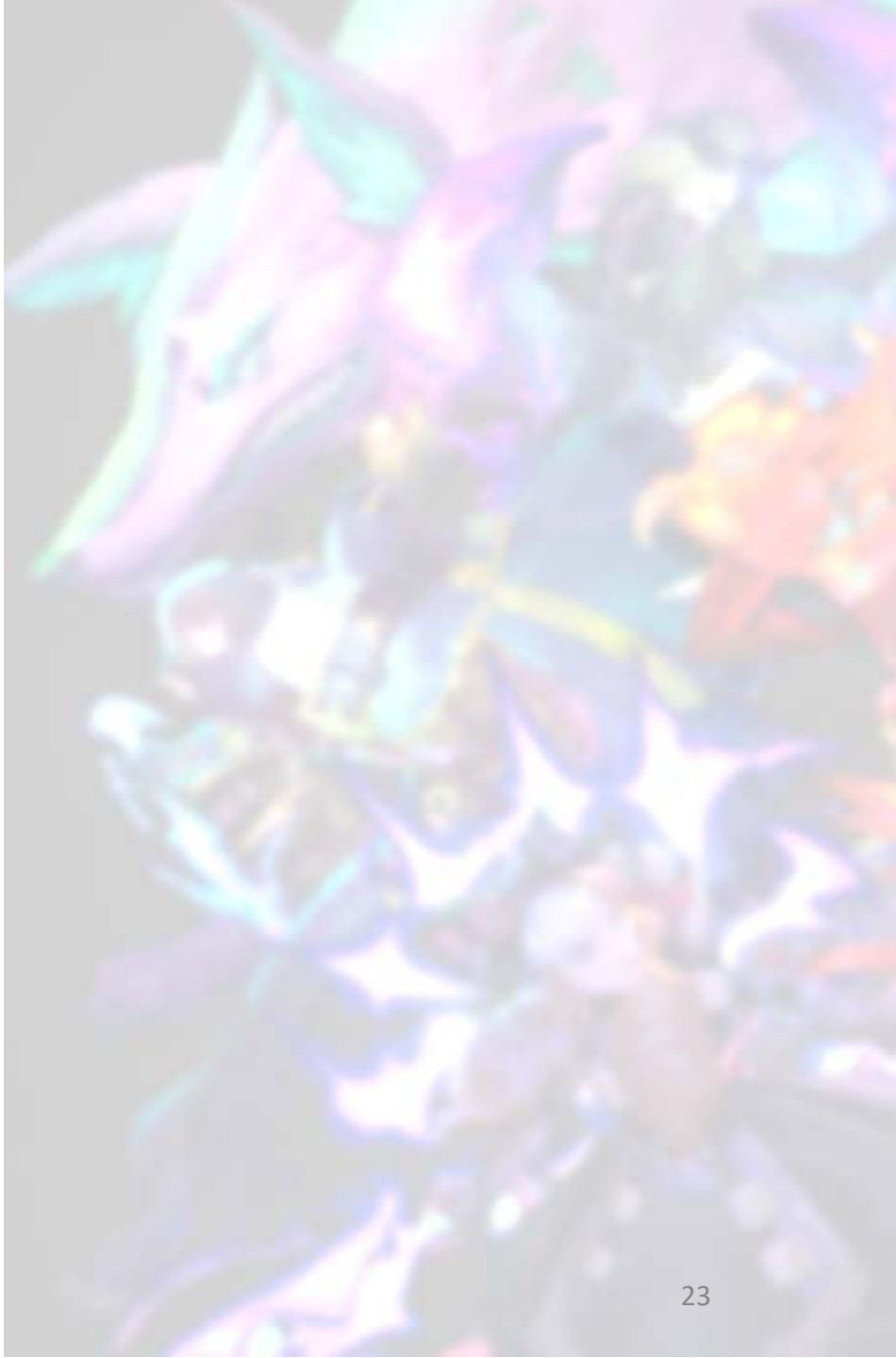
權限控制：protected（Bonus）

- 在這個例子下，SkillUsageCount() 就應該被放到 protected
  - 就能夠在不公開這個 function 的情況下
  - 依然被子類別所存取
- 符合「最小公開原則」（Principle of Least Privilege, POLP）
  - 該原則指出，安全架構的設計應確保每個實體僅獲得執行其功能所需的最少系統資源和授權
  - 重要的安全程式設計原則之一

```
1  //
2  // Created by 黃漢軒 on 2023/10/9.
3  //
4
5  #ifndef OOP_CHARACTER_HPP
6  #define OOP_CHARACTER_HPP
7
8  #include <string>
9
10 class Character {
11 private:
12     std::string name;
13     int health;
14 public:
15     Character(std::string name, int health);
16
17     std::string GetName();
18     int GetHealth();
19
20     void SetName(std::string name);
21     void SetHealth(int health);
22 protected:
23     void SkillUsageCount();
24 };
25
26 #endif // OOP_CHARACTER_HPP
27
```

Protected資料成員的優缺點

- 優點:
  - 衍生類別可直接存取
  - 程式執行效率較高
    - 因為沒有使用SETTER/GETTER
- 缺點:
  - 值沒有經過set/get函式驗證
    - 衍生類別直接存取可能指派不合法的值
  - 實作相關性高
    - 衍生類別的成員函式與基礎類別的實作相關性較高
    - 基礎類別的實作方式改變,衍生類別必須跟著變



IRVINE, C++ AND OBJECT-ORIENTED PROGRAMMING, PRENTICE-HALL,INC.