

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	

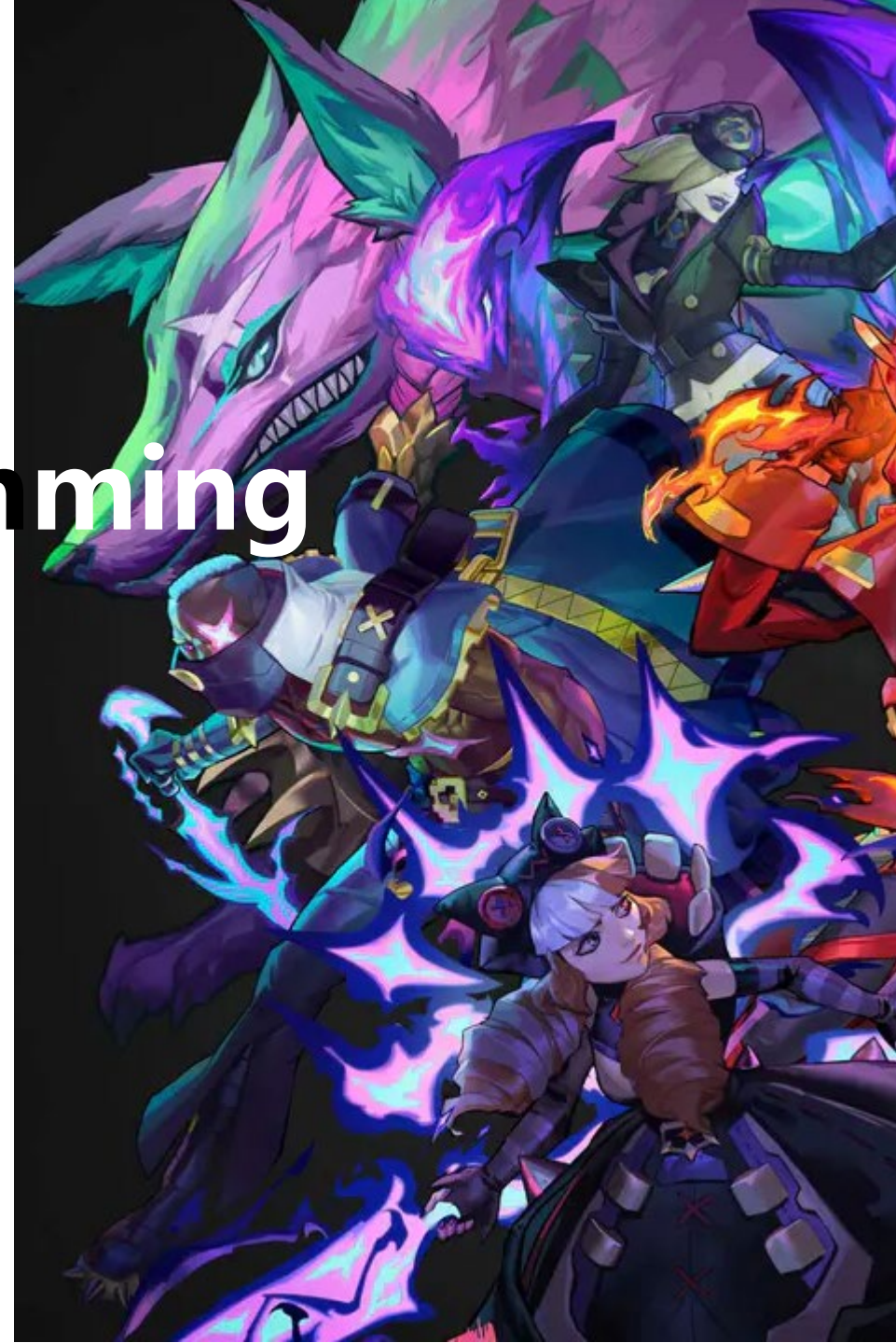
Object Oriented Programming

Lec04: Inheritance

Sun Chin-Yu (孫勤昱)

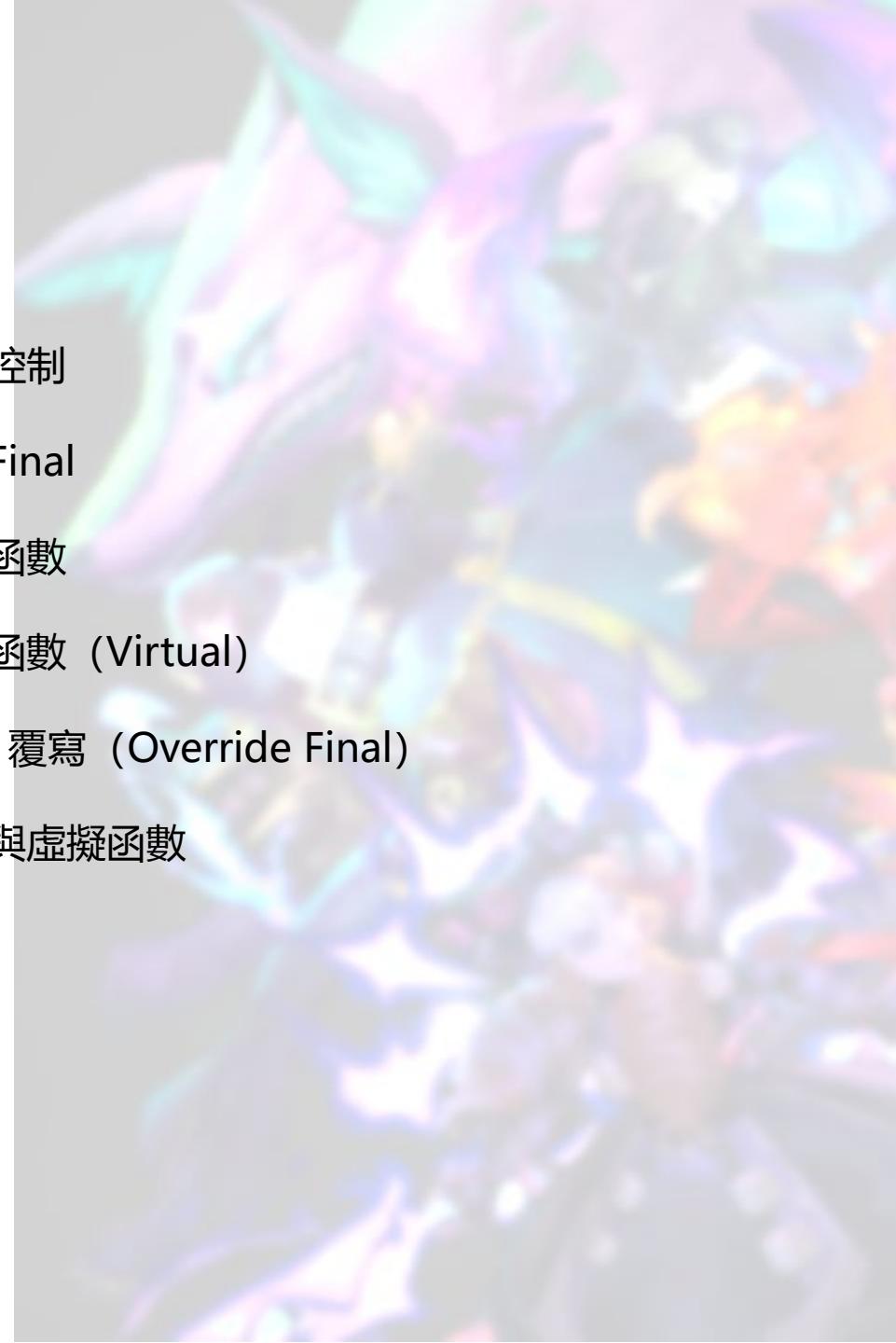
cysun@ntut.edu.tw

2023/10/20



大綱

- 回顧：繼承大概念
- 嘗試解決問題的方向
- 概念：父類別（Parent）與子類別（Children）
- 初始化列表（Initialize List）與為何需要初始化列表
- 權限控制：protect
- 繼承與權限控制
- 最終繼承：Final
- 子類：覆寫函數
- 父類：虛擬函數（Virtual）
- 父類：Final 覆寫（Override Final）
- 純虛擬函數與虛擬函數



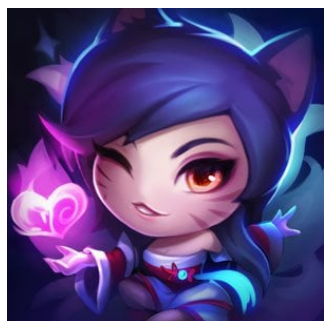
回顧：繼承大概念

- 以 League of Legends 這款遊戲當作例子，我們將再次解釋繼承相關的概念

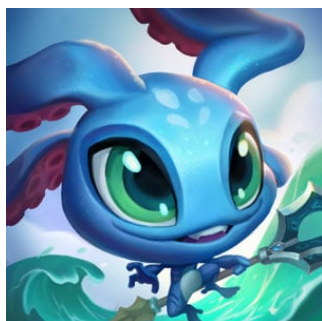


回顧：繼承大概念

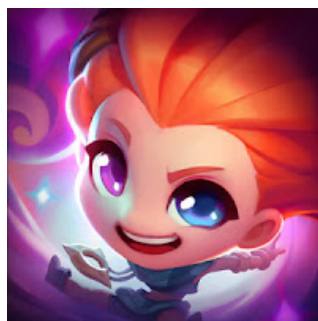
- 作為一個想要寫一個跟 LoL 一樣的遊戲的程式設計師
- 你今天想要「先」撰寫這些角色。



阿璃 Ahri
法師



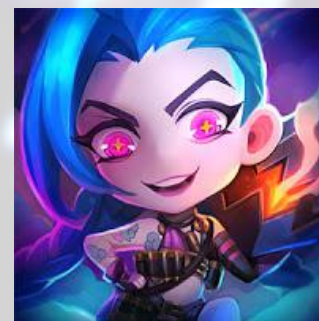
飛斯 Fizz
法師



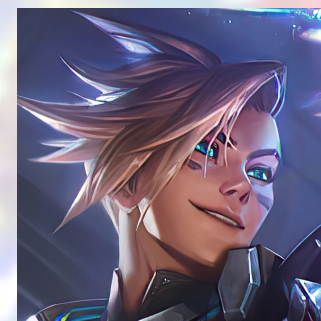
柔伊 Zoe
法師



燼 Jhin
射手



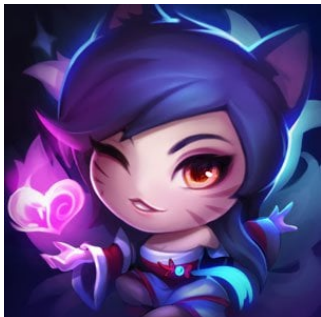
吉因克斯 Jinx
射手



伊澤瑞爾 Ezreal
射手

回顧：繼承大概念

- 針對於這些角色，你先撰寫了阿璃的類別



阿璃 Ahri
法師

Q鍵：幻玉 (OrbOfDescription)

- 阿璃擲出她的幻玉後再召回，飛出時造成魔法傷害。折返時造成真實傷害

W鍵：魅火 (FoxFire)

- 阿璃獲得短暫的爆發性跑速加成，並釋放出三團狐火，鎖定附近的敵人進行攻擊

E鍵：傾城 (Charm)

- 阿璃送出飛吻，傷害並魅惑一名命中的敵方英雄，立刻打斷移動技能，並使對方毫無抵抗的走向她

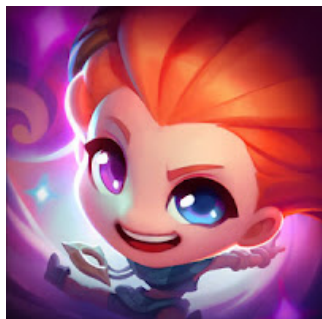
R鍵：飛仙 (SpiritRush)

- 阿璃向前衝刺並發射魔法彈，傷害附近敵人。飛仙進入冷卻前最多可以施放 3 次，參與敵方英雄的擊殺時，可獲得額外的重新施放次數

```
1  #include <string>
2
3  class Ahri {
4  private:
5      std::string name;
6      int abilityPower;
7      int health;
8  public:
9      /* Getter */
10     std::string GetName();
11     int GetAbilityPower();
12     int GetHealth();
13
14     /* Setter */
15     void SetName();
16     void SetAbilityPower();
17     void SetHealth();
18
19     /* Wizard Only*/
20     void WizardLevelUp();
21
22     /* Ahri-Only Skill */
23     void OrbOfDescription();
24     void FoxFire();
25     void Charm();
26     void SpiritRush();
27 };
```

回顧：繼承大概念

- 然後撰寫了柔伊的類別



柔伊 Zoe
法師

Q鍵：星迴百轉 (MoreSparkles)

- 柔依發射一顆星星，飛行期間她可以重啟技能，將星星導向至新的位置。星星在直線上飛行的距離越長，造成的傷害越高。

W鍵：法術竊盜 (PaddleStar)

- 柔依可拾起敵方召喚師技能及主動道具效果的碎片，並施放一次該召喚師技能或主動道具效果。每次施放一個召喚師技能，她會獲得 3 顆星彈，其會朝最接近的敵方目標發射。

E鍵：沉睡夢域 (SpellThief)

- 使目標疲倦，接著陷入沉睡。沉睡期間，目標的魔法防禦將會降低。首次對該目標的攻擊會喚醒目標並造成雙倍傷害，此傷害不會超過系統設定的上限

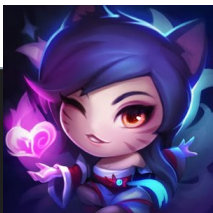
R鍵：星界躍動 (SpeelyTroubleBubble)

- 暫時傳送至附近新的位置，1 秒後返回原本的位置

```
1  #include <string>
2
3  class Zoe {
4  private:
5      std::string name;
6      int abilityPower;
7      int health;
8  public:
9      /* Getter */
10     std::string GetName();
11     int GetAbilityPower();
12     int GetHealth();
13
14     /* Setter */
15     void SetName();
16     void SetAbilityPower();
17     void SetHealth();
18
19     /* Wizard Only*/
20     void WizardLevelUp();
21
22     /* Zoe-Only Skill */
23     void MoreSparkles();
24     void PaddleStar();
25     void SpellThief();
26     void SpeelyTroubleBubble();
27 };
```

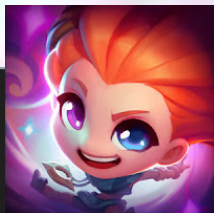
回顧：繼承大概念

- 到這邊會開始覺得不對勁
- 出現了许多重複的 Code



阿璃 法師

```
1 #include <string>
2
3 class Ahri {
4 private:
5     std::string name;
6     int abilityPower;
7     int health;
8 public:
9     /* Getter */
10    std::string GetName();
11    int GetAbilityPower();
12    int GetHealth();
13
14    /* Setter */
15    void SetName();
16    void SetAbilityPower();
17    void SetHealth();
18
19    /* Wizard Only*/
20    void WizardLevelUp();
21
22    /* Ahri-Only Skill */
23    void OrbOfDescription();
24    void FoxFire();
25    void Charm();
26    void SpiritRush();
27 };
```



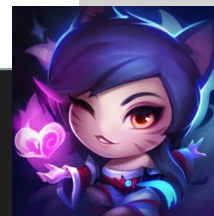
柔伊 法師

```
1 #include <string>
2
3 class Zoe {
4 private:
5     std::string name;
6     int abilityPower;
7     int health;
8 public:
9     /* Getter */
10    std::string GetName();
11    int GetAbilityPower();
12    int GetHealth();
13
14    /* Setter */
15    void SetName();
16    void SetAbilityPower();
17    void SetHealth();
18
19    /* Wizard Only*/
20    void WizardLevelUp();
21
22    /* Zoe-Only Skill */
23    void MoreSparkles();
24    void PaddleStar();
25    void SpellThief();
26    void SpeelyTroubleBubble();
27 };
```

...

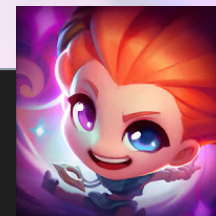
回顧：繼承大概念

- 假設2043年這遊戲還在...法師數量已經高達100隻
 - 重複撰寫許多次 WizardLevelUp() 的實作是可預期的情況
 - 若我們修改 WizardLevelUp() 的實作, 要改 100 個類別的實作
- 有人想過怎麼解決這個問題嗎?



阿璃 法師

```
1  #include <string>
2
3  class Ahri {
4  private:
5      std::string name;
6      int abilityPower;
7      int health;
8  public:
9      /* Getter */
10     std::string GetName();
11     int GetAbilityPower();
12     int GetHealth();
13
14     /* Setter */
15     void SetName();
16     void SetAbilityPower();
17     void SetHealth();
18
19     /* Wizard Only*/
20     void WizardLevelUp();
21
22     /* Ahri-Only Skill */
23     void OrbOfDescription();
24     void FoxFire();
25     void Charm();
26     void SpiritRush();
27 };
```



柔伊 法師

```
1  #include <string>
2
3  class Zoe {
4  private:
5      std::string name;
6      int abilityPower;
7      int health;
8  public:
9      /* Getter */
10     std::string GetName();
11     int GetAbilityPower();
12     int GetHealth();
13
14     /* Setter */
15     void SetName();
16     void SetAbilityPower();
17     void SetHealth();
18
19     /* Wizard Only*/
20     void WizardLevelUp();
21
22     /* Zoe-Only Skill */
23     void MoreSparkles();
24     void PaddleStar();
25     void SpellThief();
26     void SpeelyTroubleBubble();
27 };
```


嘗試解決問題的方向

- 想一下，有沒有什麼辦法可以解決這個問題？



嘗試解決問題的方向

- 我們可以歸納出，這些角色其實都是法師。



嘗試解決問題的方向

- 如果我們有辦法可以實踐這個階層圖，也許就會便利許多！



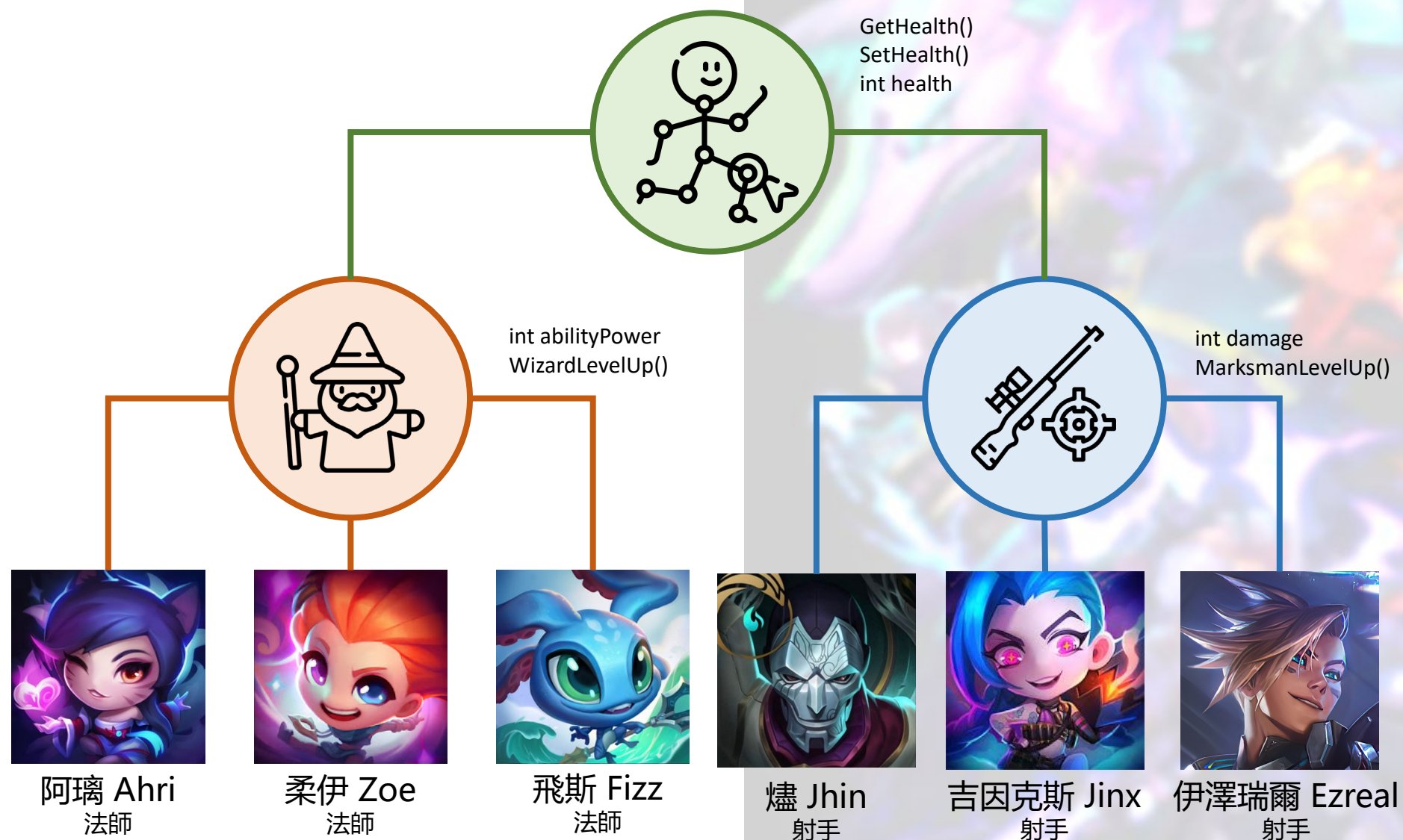
嘗試解決問題的方向

- 用這樣的方式，我們可以統一每個法師 WizardLevelUp() 的實作。
- 這樣即使有 100 個法師種類的角色，我們只需要實作角色內專屬的函式即可。
- 這樣的方式可以大幅減少重複的程式碼。



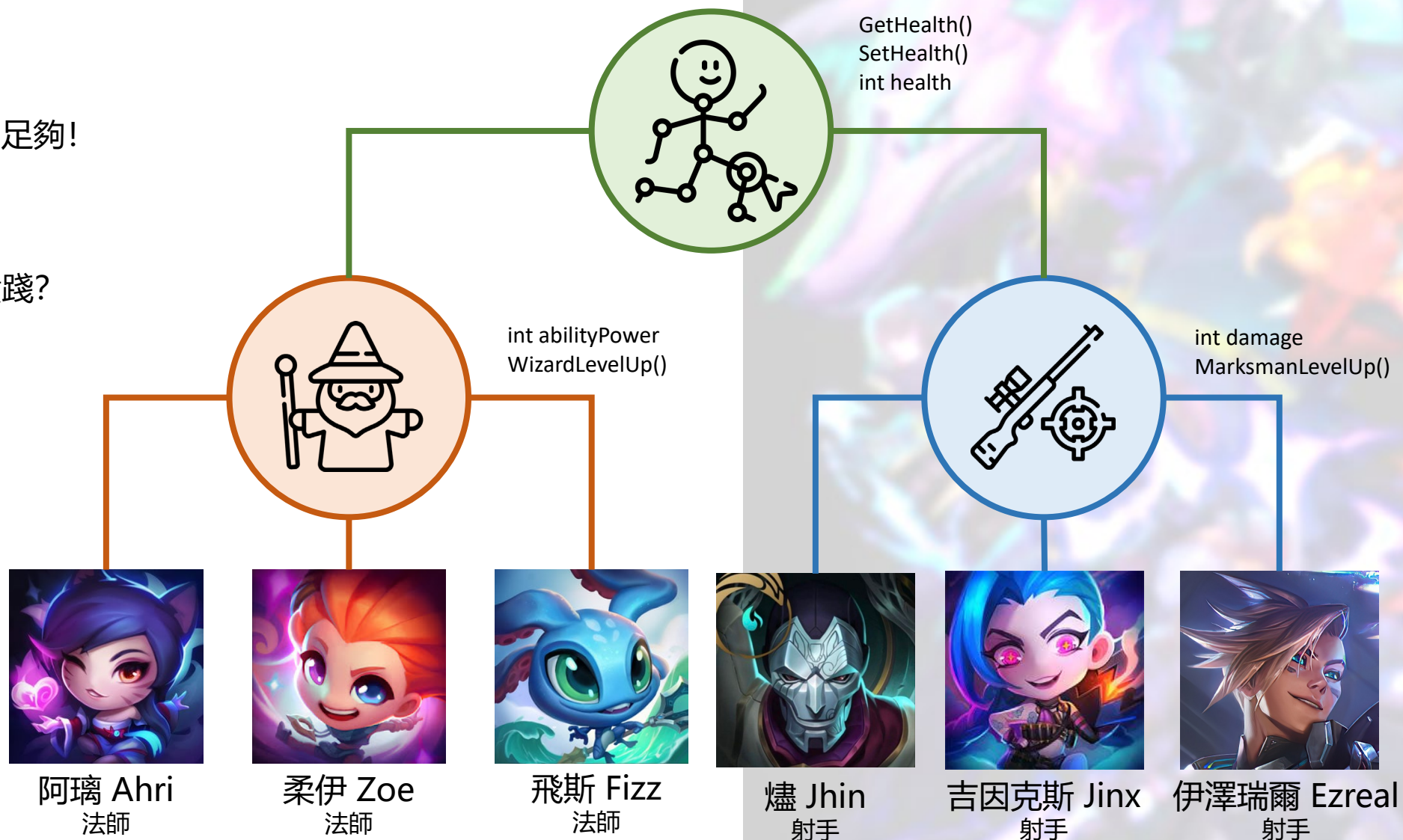
嘗試解決問題的方向

- 我們甚至還可以...



嘗試解決問題的方向

- 看起來更加簡潔且表達力足夠!
- 想法 Approved, 如何實踐?



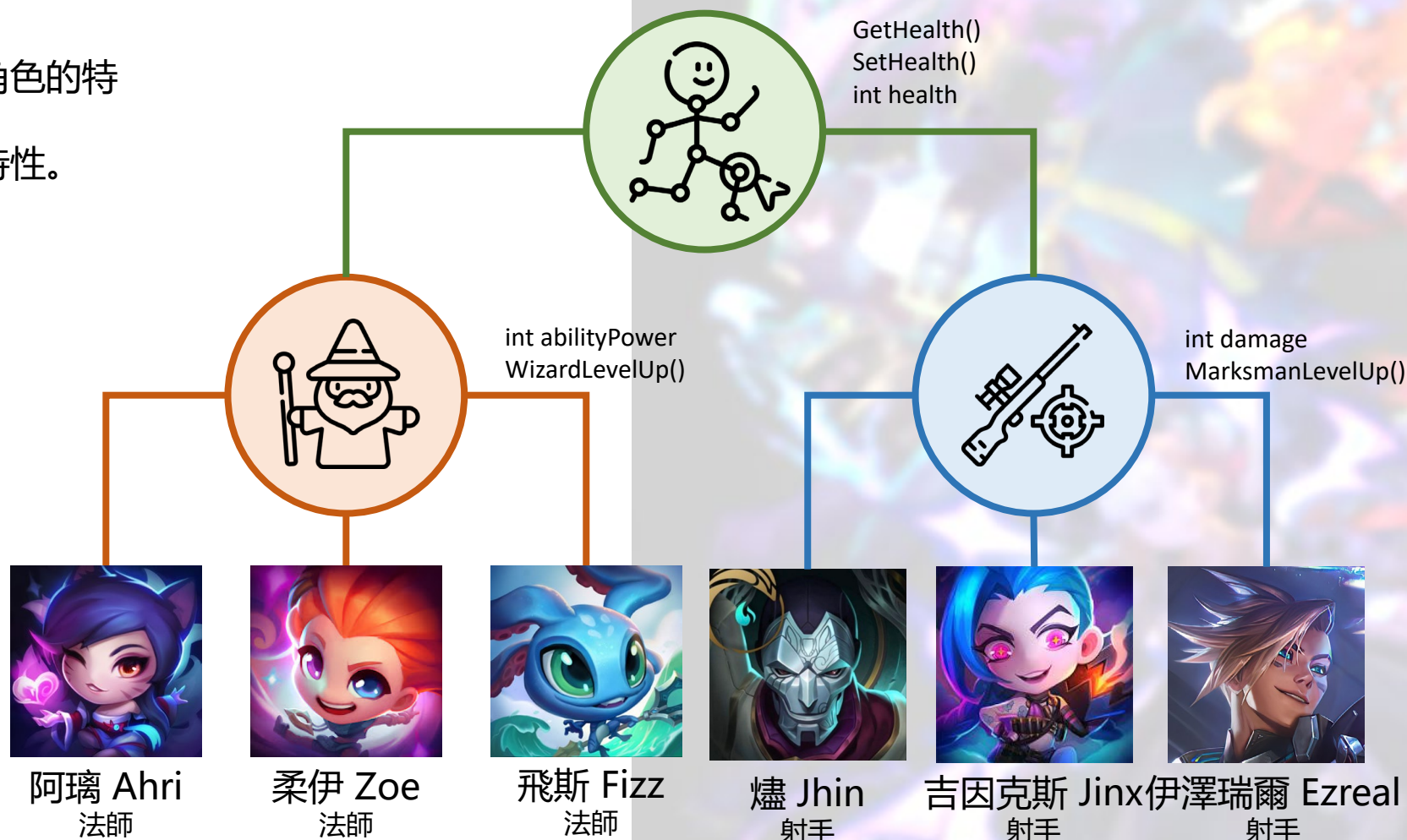
概念：父類別 (Parent) 與子類別 (Children)

- 在這個章節，我們將延續「問題解決的方向」，繼續推演父類別與子類別的關係。



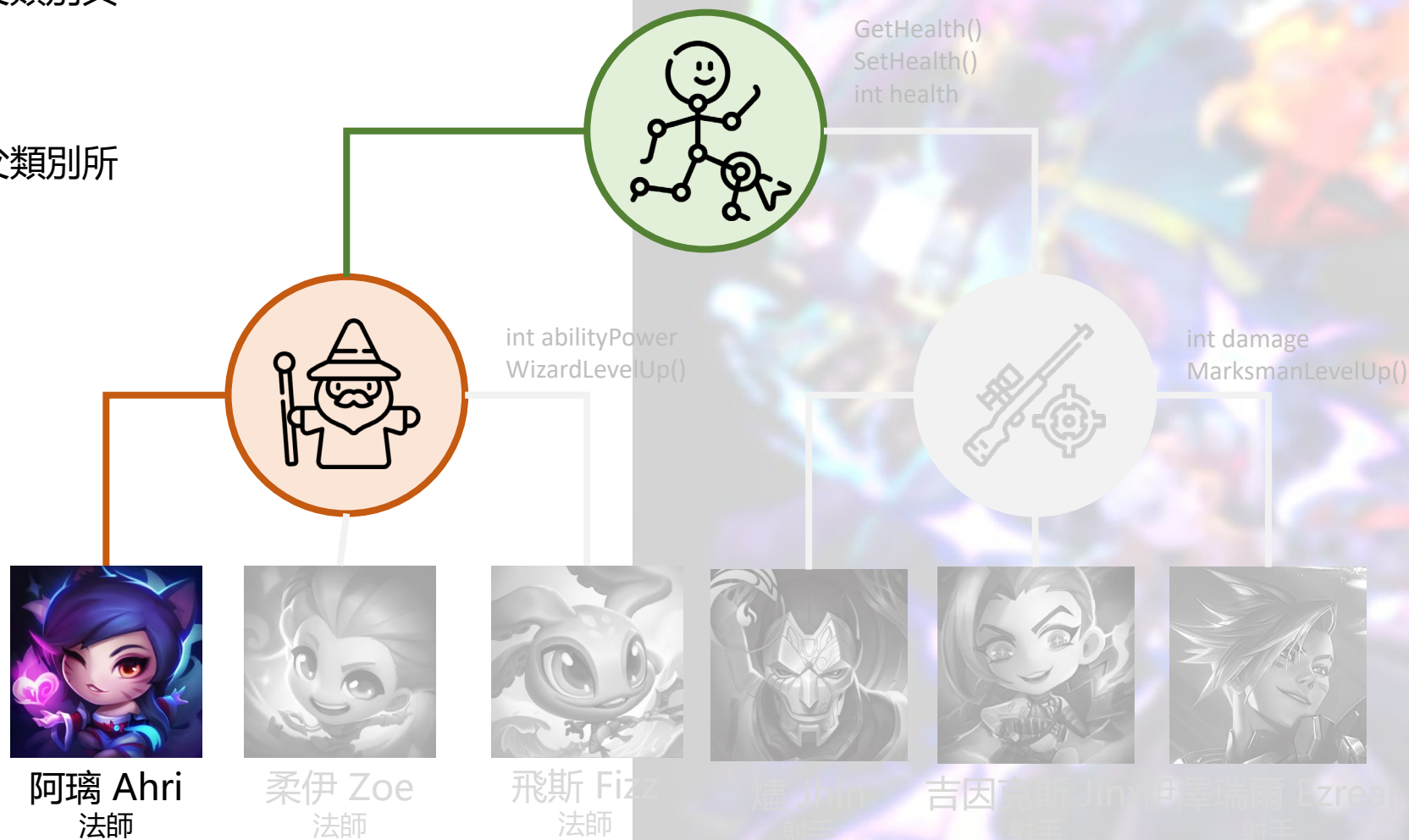
概念：父類別（Parent）與子類別（Children）

- 透過這樣的方式，我們稱為繼承。
- 阿璃繼承了法師的特性、法師繼承的角色的特性，等同於阿璃繼承了法師與角色的特性。
- 阿璃是角色而且是法師，合理。



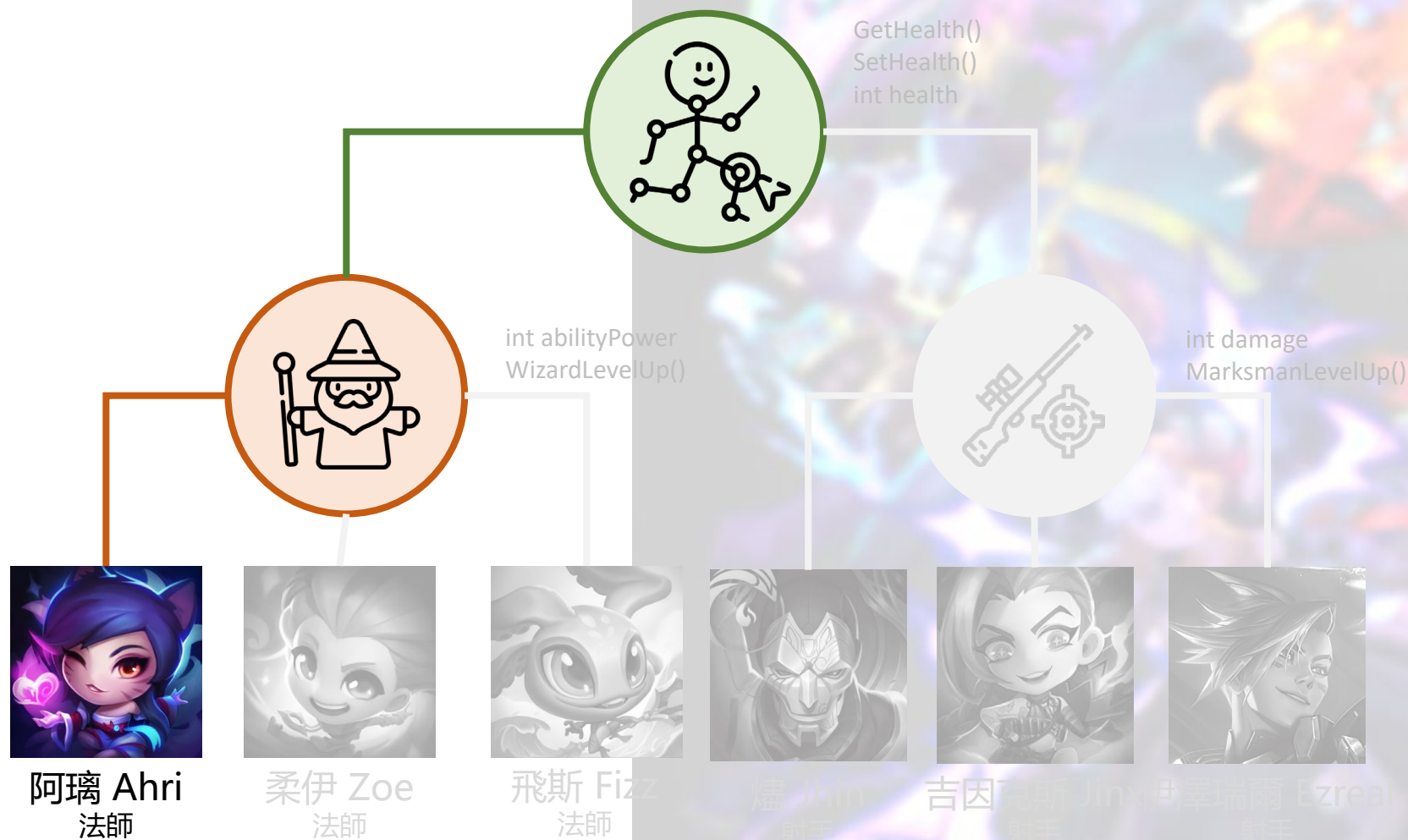
概念：父類別（Parent）與子類別（Children）

- 在這邊，我們可以稍微提一下所謂的父類別與子類別。
- 父類別主要為繼承上游，子類別繼承父類別所有的成員與函式。



概念：父類別（Parent）與子類別（Children）

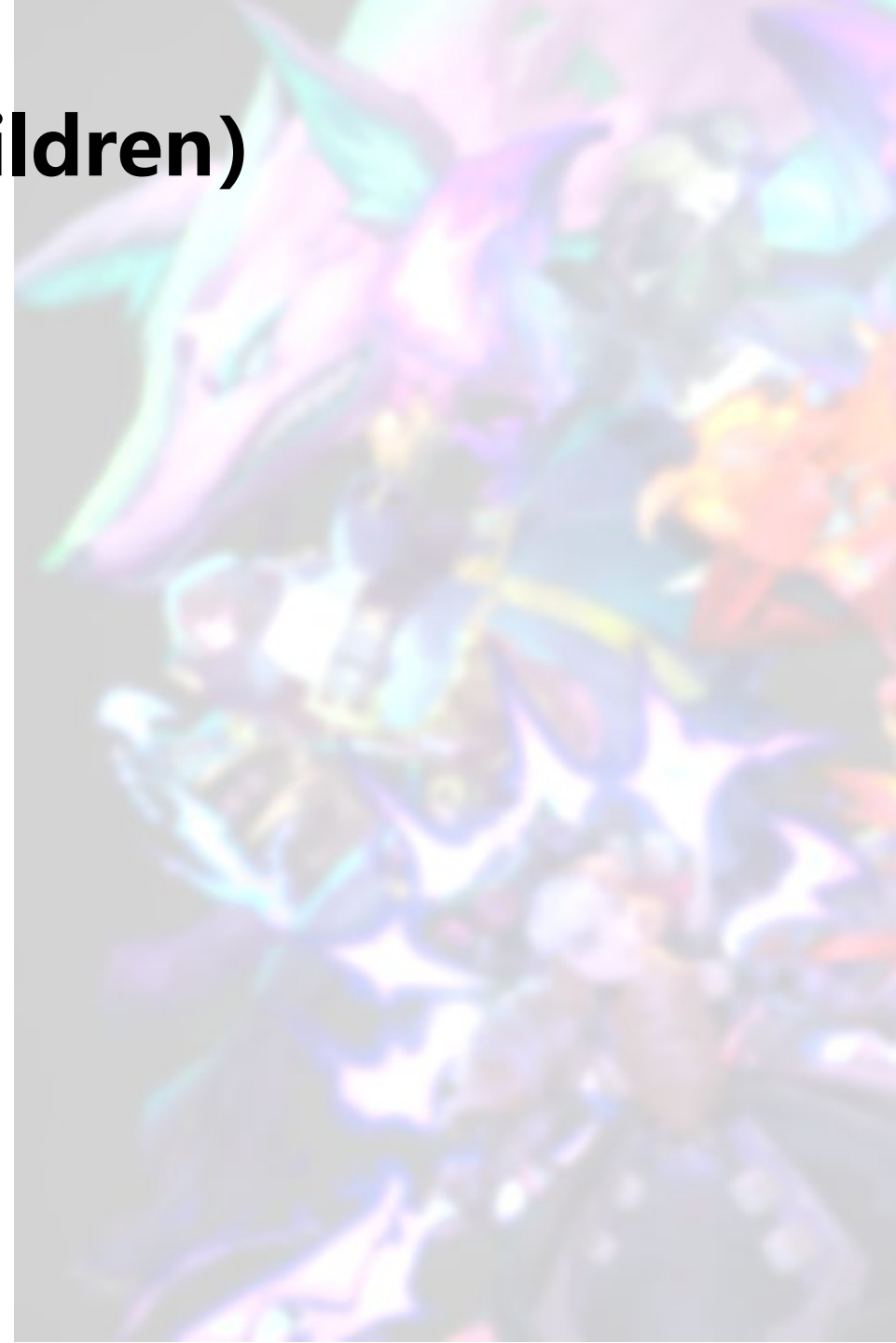
- 在這個例子中：
 - 法師是角色的子類別。
 - 阿璃是法師的子類別。
- 法師具有角色**所有的**成員與函式。
- 阿璃具有法師**所有的**成員與函式。



概念：父類別（Parent）與子類別（Children）

- 回來看這個程式...
- 實踐一下之前的想法？

```
1  #include <string>
2
3  class Ahri {
4  private:
5      std::string name;
6      int abilityPower;
7      int health;
8  public:
9      /* Getter */
10     std::string GetName();
11     int GetAbilityPower();
12     int GetHealth();
13
14     /* Setter */
15     void SetName();
16     void SetAbilityPower();
17     void SetHealth();
18
19     /* Wizard Only */
20     void WizardLevelUp();
21
22     /* Ahri-Only Skill */
23     void OrbOfDescription();
24     void FoxFire();
25     void Charm();
26     void SpiritRush();
27 };
```



概念：父類別（Parent）與子類別（Children）

- 以阿璃來實踐一下想法。

```
1 #include <string>
2
3 class Ahri {
4 private:
5     std::string name;
6     int abilityPower;
7     int health;
8 public:
9     /* Getter */
10    std::string GetName();
11    int GetAbilityPower();
12    int GetHealth();
13
14    /* Setter */
15    void SetName();
16    void SetAbilityPower();
17    void SetHealth();
18
19    /* Wizard Only*/
20    void WizardLevelUp();
21
22    /* Ahri-Only Skill */
23    void OrbOfDescription();
24    void FoxFire();
25    void Charm();
26    void SpiritRush();
27 };
```

=

```
1 #include <string>
2
3 #include "mage.h"
4
5 class Ahri : Mage {
6 public:
7     /* Ahri-Only Skill */
8     void OrbOfDescription();
9     void FoxFire();
10    void Charm();
11    void SpiritRush();
12 };
```

阿璃特有的技能。

```
1 #include "character.h"
2
3 class Mage : Character {
4 private:
5     int abilityPower;
6 public:
7     /* Getter */
8     int GetAbilityPower();
9
10    /* Setter */
11    void SetAbilityPower();
12
13    /* Wizard-only */
14    void WizardLevelUp();
15 };
```

我們假設法師都會有魔法攻擊，升級時針對魔法攻擊作調整。

```
1 #include <string>
2
3 class Character {
4 private:
5     std::string name;
6     int health;
7 public:
8     /* Getter */
9     std::string GetName();
10    int GetHealth();
11
12    /* Setter */
13    void SetName();
14    void SetHealth();
15 };
```

我們假設角色都會有名字與血量。

概念：父類別（Parent）與子類別（Children）

- 加上柔依呢？

```
1  #include <string>
2
3  class Zoe {
4  private:
5      std::string name;
6      int abilityPower;
7      int health;
8  public:
9      /* Getter */
10     std::string GetName();
11     int GetAbilityPower();
12     int GetHealth();
13
14     /* Setter */
15     void SetName();
16     void SetAbilityPower();
17     void SetHealth();
18
19     /* Wizard Only*/
20     void WizardLevelUp();
21
22     /* Zoe-Only Skill */
23     void MoreSparkles();
24     void PaddleStar();
25     void SpellThief();
26     void SpeelyTroubleBubble();
27 };
```

=

```
1  #include <string>
2
3  #include "mage.h"
4
5  class Ahri : Mage {
6  public:
7      /* Ahri-Only Skill */
8      void OrbOfDescription();
9      void FoxFire();
10     void Charm();
11     void SpiritRush();
12 };
```

```
1  #include <string>
2
3  #include "mage.h"
4
5  class Zoe : Mage {
6  public:
7      /* Zoe-Only Skill */
8      void MoreSparkles();
9      void PaddleStar();
10     void SpellThief();
11     void SpeelyTroubleBubble();
12 };
```

```
1  #include "character.h"
2
3  class Mage : Character {
4  private:
5      int abilityPower;
6  public:
7      /* Getter */
8      int GetAbilityPower();
9
10     /* Setter */
11     void SetAbilityPower();
12
13     /* Wizard-only */
14     void WizardLevelUp();
15 };
```

```
1  #include <string>
2
3  class Character {
4  private:
5      std::string name;
6      int health;
7  public:
8      /* Getter */
9      std::string GetName();
10     int GetHealth();
11
12     /* Setter */
13     void SetName();
14     void SetHealth();
15 };
```

程式碼重複率近乎 0!

概念：父類別（Parent）與子類別（Children）

- 看完前面的精采操作之後



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？

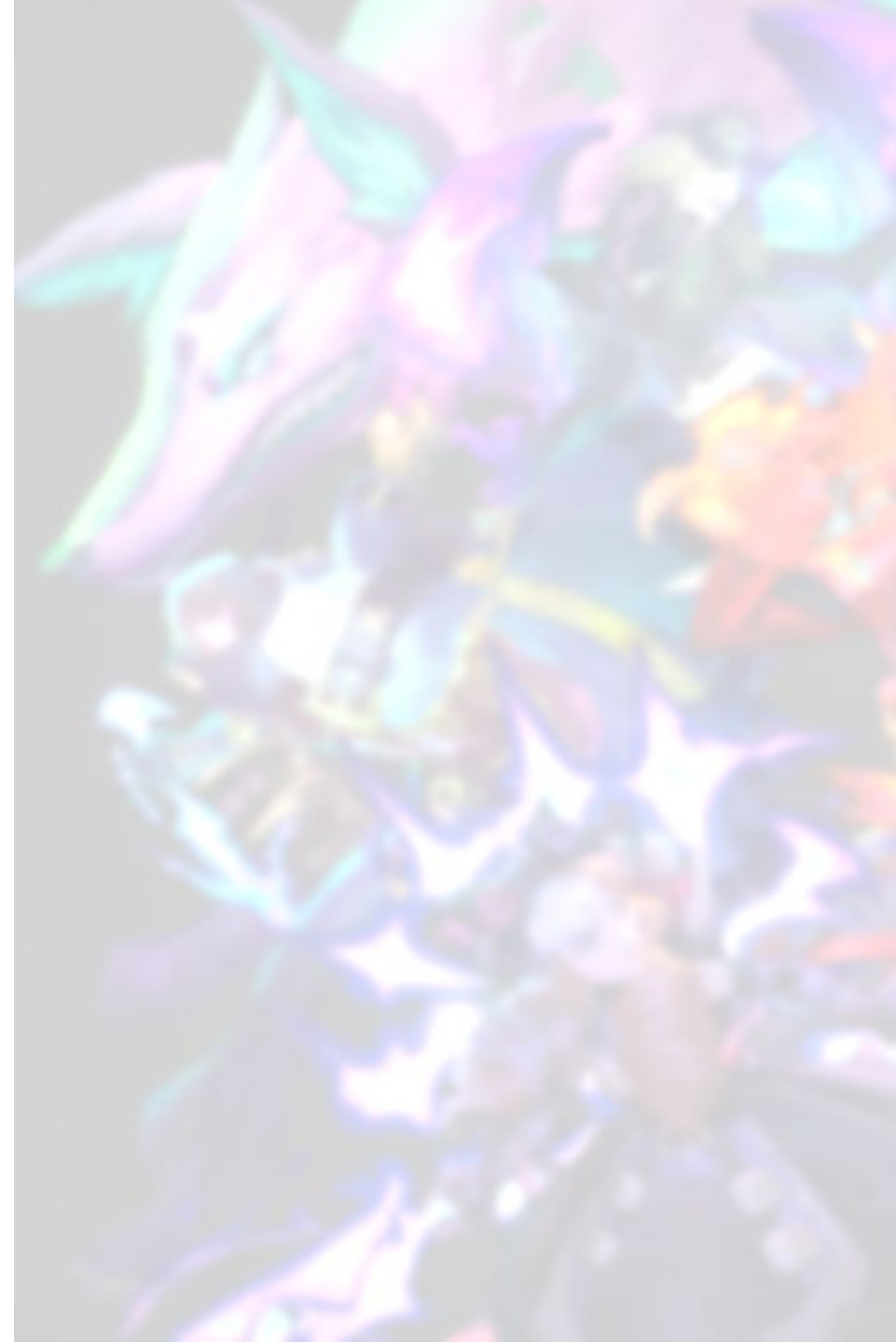
```
1  #include <string>
2
3  #include "mage.h"
4
5  class Zoe : Mage {
6  public:
7      /* Zoe-Only Skill */
8      void MoreSparkles();
9      void PaddleStar();
10     void SpellThief();
11     void SpeelyTroubleBubble();
12 };
```

```
1  #include "character.h"
2
3  class Mage : Character {
4  private:
5      int abilityPower;
6  public:
7      /* Getter */
8      int GetAbilityPower();
9
10     /* Setter */
11     void SetAbilityPower();
12
13     /* Wizard-only */
14     void WizardLevelUp();
15 };
```

```
1  #include <string>
2
3  class Character {
4  private:
5      std::string name;
6      int health;
7  public:
8      /* Getter */
9      std::string GetName();
10     int GetHealth();
11
12     /* Setter */
13     void SetName();
14     void SetHealth();
15 };
```

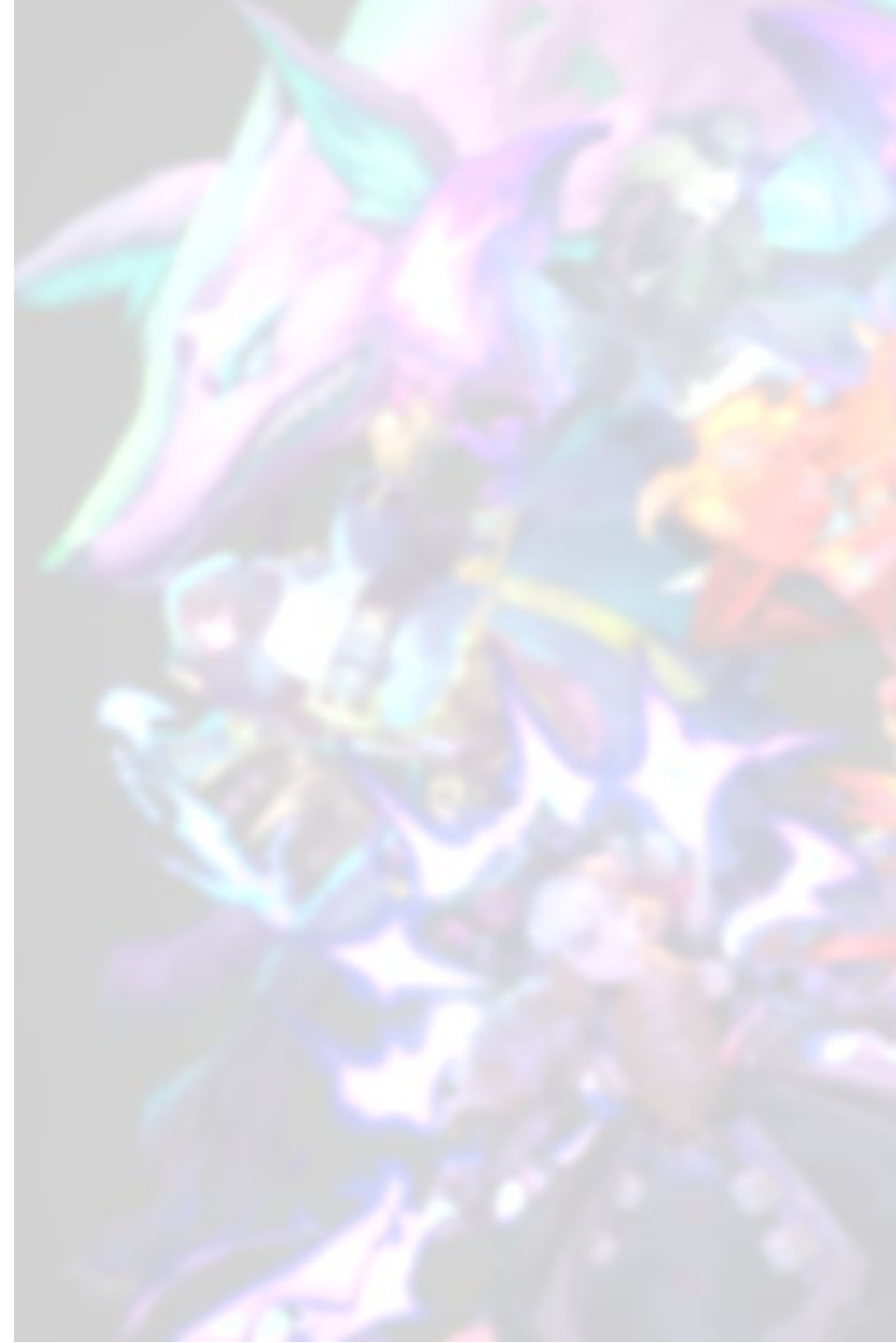

繼承關係

- 不同類別之間的關係
 - 合成關係 vs. **繼承關係**
- 合成關係 (has-a)
 - 物件內含一或多個其他類別的物件
 - Ex. 車子有多個輪子（輪子屬於車子的物件）
- **繼承關係 (is-a)**
 - **衍生類別 “is-a” 基礎類別**
 - **Ex. 貓咪是一種動物**
 - 把貓咪看成一種動物
 - 貓咪具有動物的屬性與行為



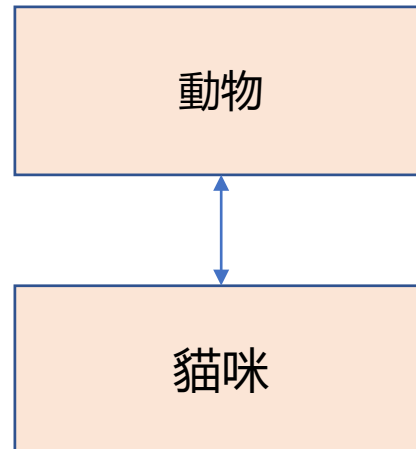
繼承的好處

- 軟體（或程式碼）重複使用
- 使用舊的類別建構新類別
 - 站在現有的基礎上
 - 可擁有現有類別的成員資料與行為
 - 可改寫現有類別的成員資料與行為
 - 可外加新的成員資料與行為



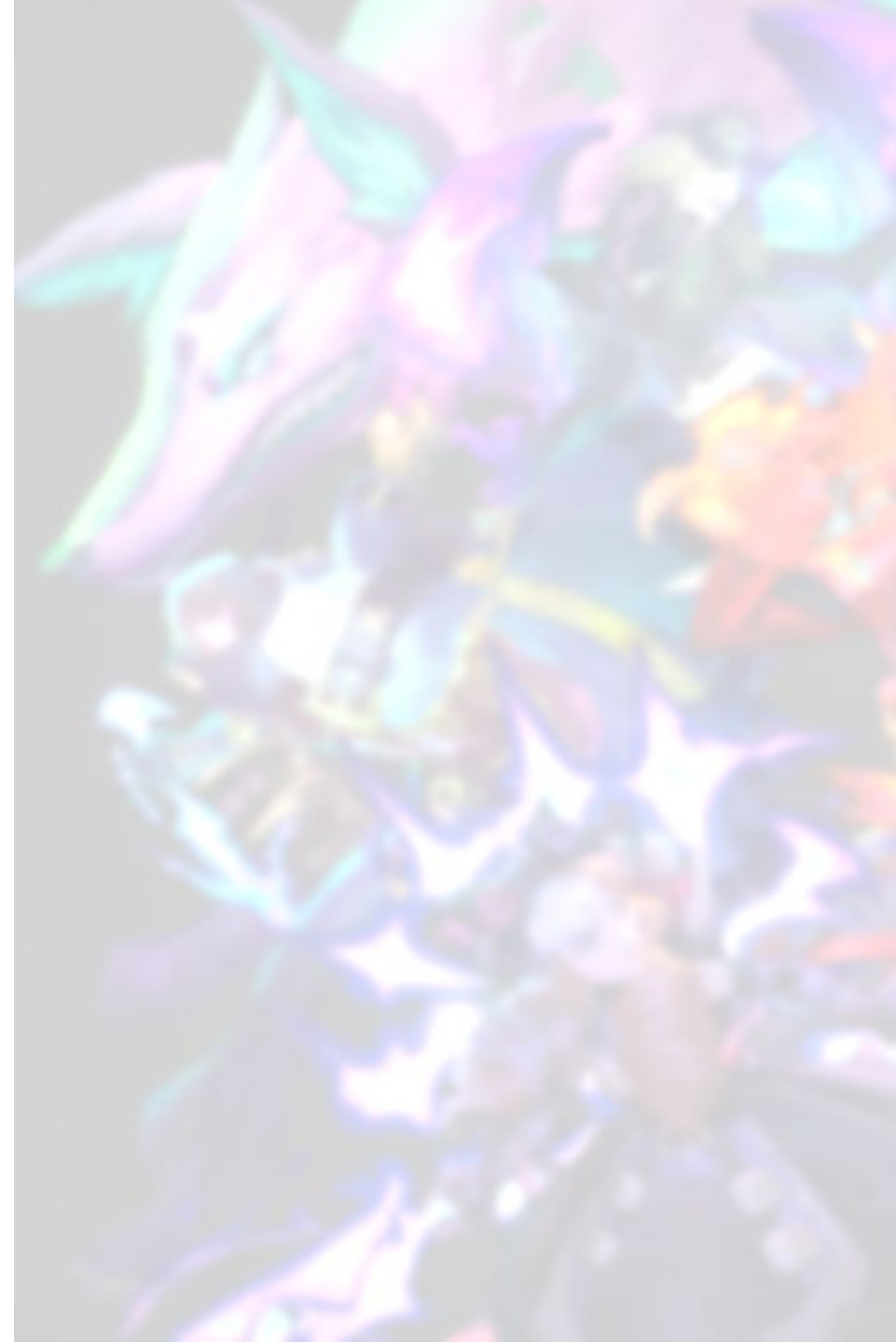
基礎類別與衍生類別

- 被繼承的類別稱為基礎類別 (Base Class)
- 繼承別人的類別稱為衍生類別 (Derived Class)
- 衍生類別的物件也是基礎類別的物件
 - Ex. 貓咪類別繼承自動物
 - 貓咪的物件也是動物的物件
 - 動物是基礎類別
 - 貓咪是衍生類別



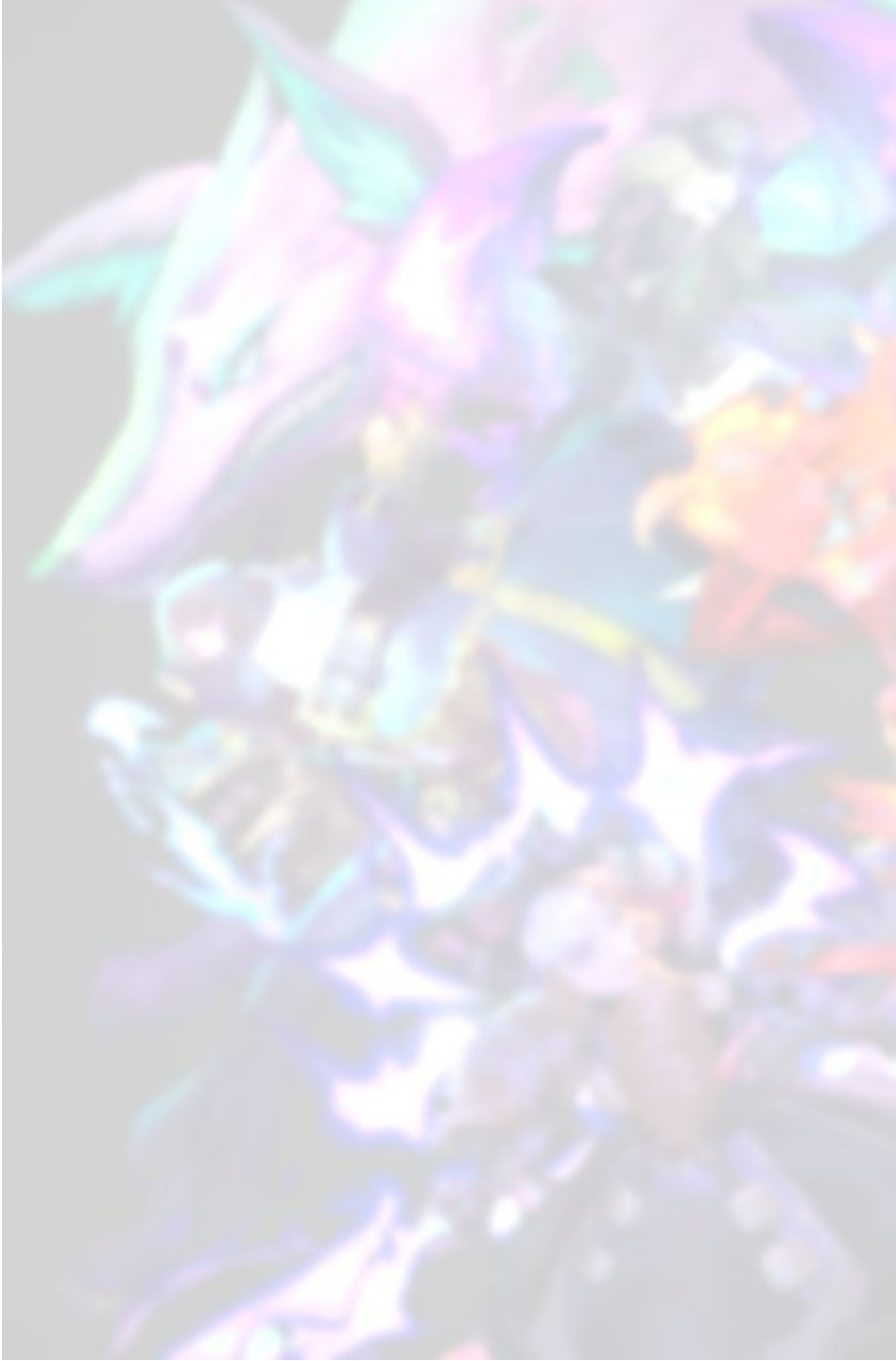
範圍比較

- 基礎類別包含的範圍比衍生類別廣
- Ex.
 - 基礎類別：載具 (Vehicle)
 - 汽車、卡車、船、腳踏車、...
 - 衍生類別：汽車 (Car)
 - 雙門轎車、四門轎車、超跑、電動車、...



繼承的例子

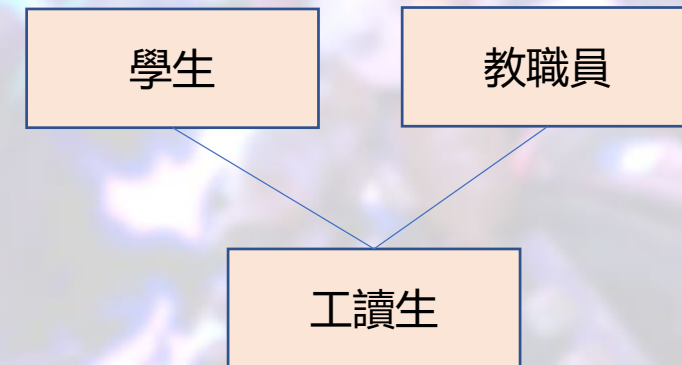
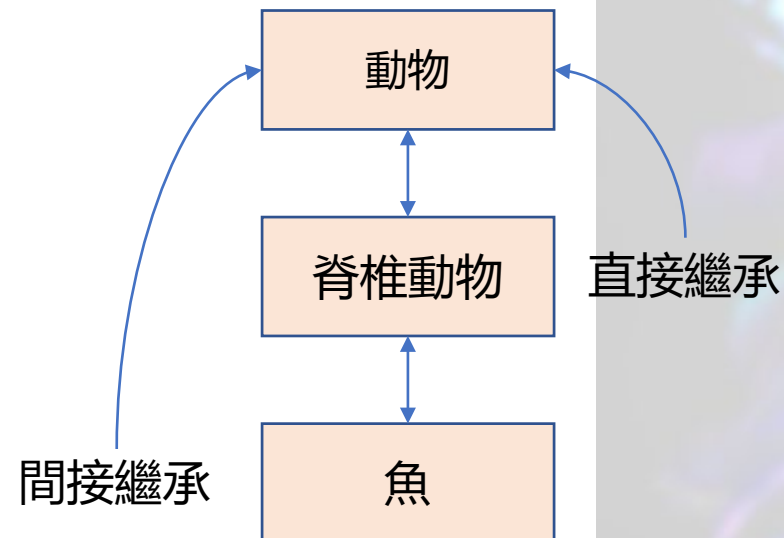
基礎類別	衍生類別
學生	國小生 高中生 大學生 研究生 博士生
形狀	圓形 三角形 四邊形
貸款	學生貸款 汽車貸款 房屋貸款
員工	教職員 行政人員 警衛 清潔人員



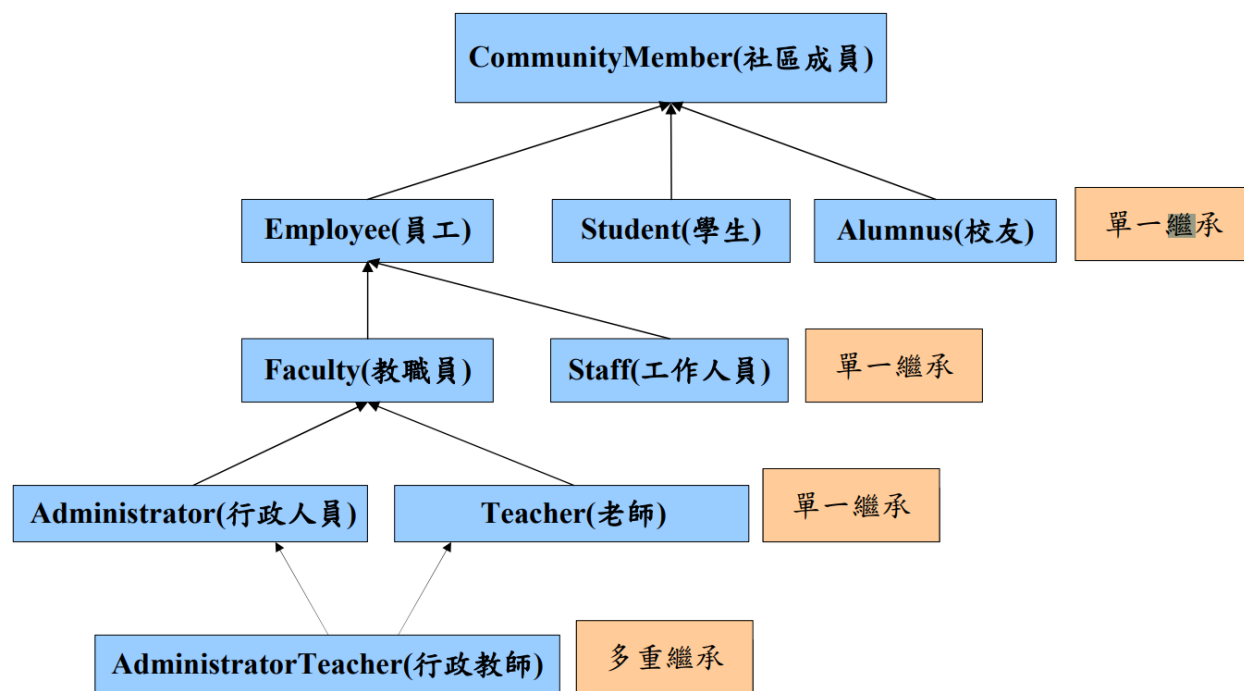
繼承階層圖 (Inheritance hierarchy)

- 繼承關係：樹狀階層圖
- 每個類別可以是
 - 基礎類別
 - 提供其他類別繼承
 - 提供成員與行為給其衍生類別
 - 衍生類別
 - 繼承其他類別
 - 可以使用基礎類別的成員與行為

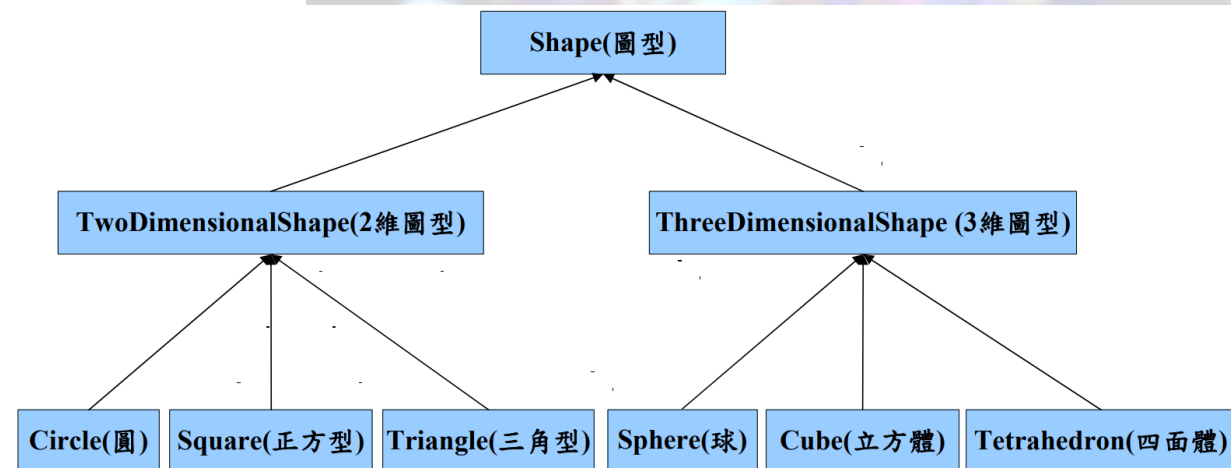
- 繼承關係分為：
 - 直接 (Direct) 繼承
 - 間接 (Indirect) 繼承
 - 單一 (Single) 繼承
 - 繼承自單一個基礎類別
 - 多重 (Multiple) 繼承
 - 繼承自多個基礎類別



繼承階層圖 (Inheritance hierarchy)



大學社群繼承階層圖



形狀繼承階層圖