

18. 資訊隱藏與封裝

- Information Hiding
- Encapsulation
- Access modifiers
- Setters and Getters

在物件導向程式設計裡面有兩個很重要的概念，就是**資訊隱藏(information hiding)**和**封裝(encapsulation)**。資訊隱藏是指當一個物件被外界存取的時候，外界無法得知物件內部的運作方式，或者是物件內部使用了哪些私有的資料成員和函式成員。而封裝指的是因為外界無法得知物件內部情形，只能使用物件提供給外界存取的函式，就好像一台有外殼包裝的機器，您只能看到按鈕，看不到裡面的電路板。

要做到資訊隱藏(information hiding)可以使用存取修飾字，讓物件的使用者無法存取內部的資料項目與方法。至於封裝(encapsulation)則可以透過設計供外界使用的界面來完成。

18.1 存取修飾字

要達到資訊隱藏和封裝，就一定要了解C++如何控制類別成員(包含資料成員與成員函式)的存取權限。public、protected和private這三個存取修飾字(access modifier)是C++用來控制存取權限的識別字。

就如同字面上的意思，這三個access modifier的意思分別是公開的、被保護的和私有的，分別有著不同程度的保護層級。

- public是指對存取權限完全的公開，任何物件都可以存取。
- protected的存取是受限的，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(關成員函式的朋友及朋友類別後續會再行說明)。
- private是限制最嚴格的access modifier，只有在類別本身內部可以存取。

我們將C++語言提供的三個存取修飾字public、protected與private再加上不使用修飾字的話共有四種情形，彙整於table 1

位置	private	protected	public	不使用修飾字
同一個類別中	✓	✓	✓	✓
朋友類別	✓	✓	✓	✓
子類別		✓	✓	
其它類別			✓	

Tab. 1: Access Modifiers and Accessibility

18.2 類別定義與存取控制

一般來說，我們都會將類別的資料成員和某些只供內部使用的成員函式設為private，然後開放一些成員函式做為和外界溝通的介面，依開放權限的程度可設定這些成員函式為public或者是protected，這樣做的好

處就是可以讓類別內部的程式修改不會影響到其它的類別，而其它外界類別因為無法存取類別內部的資料成員，也可以減少不可預知的程式錯誤，進而控制程式除錯的範圍。

我們把定義類別的語法，增加存取修飾字(access modifier)，來限制資料成員與成員函式的使用，其語法如下：

```
class className
{
private:
    資料成員或成員函式定義;

protected:
    資料成員或成員函式定義;

public:
    資料成員或成員函式定義;
}
```

注意，若呼略存取修飾字，則所定義的資料成員或成員函式之權限為private

18.3 供外部使用的界面: Setters and Getters

適當地定義存取權限，讓其它程式只能透過我們所開放的method來存取資料成員，可以確保程式碼的正確性與安全性。如果想開放資料成員的存取，我們通常會設計成public的setXXX()與getXXX()成員函式來讓他人使用。

- setters又稱為Mutators，其成員函式命名通常為setXXX()或是set_XXX()
- getters又稱為Accessors，其成員函式命名通常為getXXX()或是get_XXX()

現在，我們再將Person類別修改如下：

```
#ifndef _PERSON_
#define PERSON

class Person
{
private:
    string firstname;
    string lastname;

public:
    Person();
    Person(string, string);
    void showInfo();

    void set_firstname(string fn);
    string get_firstname();
}
```

```
    void set_lastname(string ln);  
    string get_lastname();  
};  
#endif
```

```
#include <iostream>  
#include "person.h"  
using namespace std;  
  
Person::Person()  
{  
}  
  
Person::Person(string fn, string ln)  
{  
    firstname=fn;  
    lastname=ln;  
}  
  
void Person::showInfo()  
{  
    cout << "Name: " << firstname << " " << lastname << endl;  
}  
  
void Person::set_firstname(string fn)  
{  
    firstname=fn;  
}  
  
void Person::set_lastname(string ln)  
{  
    lastname=ln;  
}  
  
string Person::get_firstname()  
{  
    return firstname;  
}  
  
string Person::get_lastname()  
{  
    return lastname;  
}
```

```
#include <iostream>
#include "person.h"
using namespace std;

int main()
{
    Person *amy = new Person;

    amy->set_firstname("Amy");
    amy->set_lastname("Chang");

    cout << amy->get_firstname() << " " << amy->get_lastname() << endl;
    return 0;
}
```

18.4 this指標

讓我們設想一個情形，假設每個 `Person` 類別的物件有除了名字之外，還有一個 `int age` 的資料成員，並且我們想要寫一個成員函式 `compare()` 用以比較兩個 `Person` 類別的物件誰的年齡比較大：

```
#ifndef _PERSON_
#define PERSON

class Person
{
private:
    string firstname;
    string lastname;
    int age;

public:
    Person();
    Person(string, string);
    void showInfo();

    void set_firstname(string fn);
    string get_firstname();

    void set_lastname(string ln);
    string get_lastname();

    void set_age(int a);
    int get_age();

    Person *compareAge(Person *p2);
};
```

```
    Person &compareAge(Person &p2);  
};  
#endif
```

```
#include <iostream>  
#include "person.h"  
using namespace std;  
  
Person::Person()  
{  
}  
  
Person::Person(string fn, string ln)  
{  
    firstname=fn;  
    lastname=ln;  
}  
  
void Person::showInfo()  
{  
    cout << "Name: " << firstname << " " << lastname << endl;  
}  
  
void Person::set_firstname(string fn)  
{  
    firstname=fn;  
}  
  
void Person::set_lastname(string ln)  
{  
    lastname=ln;  
}  
  
string Person::get_firstname()  
{  
    return firstname;  
}  
  
string Person::get_lastname()  
{  
    return lastname;  
}  
  
void Person::set_age(int a)  
{  
    age =a;  
}
```

```
int Person::get_age()
{
    return age;
}

Person * Person::compareAge(Person *p2)
{
    if(age>(p2->age))
        return this;
    else
        return p2;
}

Person & Person::compareAge(Person &p2)
{
    if(age>(p2.age))
        return *this;
    else
        return p2;
}
```

```
#include <iostream>
#include "person.h"
using namespace std;

int main()
{
    Person *amy = new Person;
    Person *tony = new Person;

    amy->set_firstname("Amy");
    amy->set_lastname("Chang");
    amy->set_age(20);

    tony->set_firstname("Tony");
    tony->set_lastname("Wang");
    tony->set_age(10);

    Person *older;

    older = amy->compareAge(tony);
    cout << older->get_firstname() << " "
         << older->get_lastname() << " is older." << endl;

    delete amy;
    delete tony;

    Person p1, p2;
```

```
p1.set_firstname("p");
p1.set_lastname("1");
p1.set_age(30);
p2.set_firstname("p");
p2.set_lastname("2");
p2.set_age(45);

Person &p3=p2;
p3=p1.compareAge(p2);
cout << p3.get_firstname() << " "
    << p3.get_lastname() << " is older." << endl;
return 0;
}
```

From:

<https://junwu.nptu.edu.tw/dokuwiki/> - Jun Wu的教學網頁

國立屏東大學資訊工程學系

CSIE, NPTU

Total: 281797

Permanent link:

<https://junwu.nptu.edu.tw/dokuwiki/doku.php?id=c++:encapsulation>



Last update: **2022/05/05 15:26**