國立屏東大學 資訊工程學系 C++程式設計入門教材

17. 封裝

封裝(Encapsulation)是物件導向的四個主要特性之一[1)]，其意是將物件相關的資料以及操作資料的方法綁在一起，並限制外界對物件內部的存取。此特性可以達成「資訊隱藏(Information Hiding)」的目的 — 讓外界無法得知物件內部的資料數值、狀態以及運作方式的目的。例如一台神奇美味咖啡機，消費者完全不需要具備、也不需要理解在咖啡機外殼內的機械構造與運作原理等知識，只要按下一個「神奇的按鈕(Magic Button)」就可以得到一杯香醇濃郁、神奇好喝的咖啡。對於咖啡機製造商與消費者來說，能夠在滿足消費者(能喝到神奇好喝的咖啡)需求的前題之下，將製造商如何開發出產品或產品如何運作的細節隱藏起來，這是一種雙贏的結果。類別就是C++語言用以實現封裝的機制，物件相關的資料成員與成員函式可以透過類別的定義包裹在一起，然後再透過存取修飾字(Access Modifier)控管，將物件內部使用了什麼變數、資料結構、運算邏輯、處理流程等細節隱藏起來，這就是本節所要介紹的「以封裝達成資訊隱藏」的做法。當然，在應用系統裡，一個物件的存在自有其特定目的，若是真的將其所有的資料成員與成員函式都隱藏起來，讓外界(此處指相對於類別的外界，也就是在程式裡面除了類別定義的地方)完全無法使用它們，這樣物件就變得一點用處都沒有了。就像是咖啡機將所有細節都隱藏在機器外殼裡一樣，若是沒有「能製作一杯神奇好喝的咖啡的按鈕」、放置沖泡咖啡用水的水箱、放置咖啡豆的儲存槽都沒有的話，那就變成一台「明明有能力製作神奇好喝的咖啡，但卻不對外開放使用的咖啡機了」！所以除了將物件封裝起來以外，C++讓我們可以在類別內部自定不同的存取限制，將不想公開或是僅限內部使用的資料成員或成員函式完全隱藏起來，但也可以選擇將部份成員視為是物件的介面，讓外界可以透過開放的介面來對物件進行操作 — 就好比那顆「能製作一杯神奇好喝的咖啡的按鈕」一樣。

17.1 存取修飾字

要以類別做到封裝與資訊隱藏和，就一定要了解C++用以控制類別成員(包含資料成員與成員函式)存取權限的public、protected和private這三個存取修飾字(Access Modifier)，就如同字面上的意思，這三個存取修飾字的意思分別是公開的、被保護的和私有的，分別有著以下不同程度的保護層級：

- public是指對存取權限完全的公開，任何函式或物件裡都可以存取。
- 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(關於成員函式與類別的朋友，以及子類別後續會再行說明)。
- private是限制最嚴格的存取修飾字，只有在類別本身內部可以存取。

我們將<nowiki>C++</nowiki>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	v	v	v	v
朋友類別	v	v	v	v
子類別		v	v	
其它類別			v	

Tab. 1: Access Modifiers and Accessibility

在繼續後續的說明前，我們要先為讀者揭露結構體2.0(struct)與類別(class)的一項差異，在預設的情況下結構體的成員是public的，而類別的成員預設是private的，因此以下兩個結構體的定義是完全相同的：

```
struct Student
{
   string name;
   string SID;
   int    score;
};
```

```
struct Student
{
public:
   string name;
   string SID;
   int    score;
};
```

同理，以下的兩個類別定義也是相同的：

```
class Student
{
   string name;
   string SID;
   int    score;
};
```

```
class Student
{
private:
   string name;
   string SID;
   int    score;
};
```

17.2 類別定義與存取控制

一般來說，為了達成資訊隱藏的目的，我們通常都會將類別裡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：

包含存取修飾字的類別定義語法

```
class 類別名稱
{
[存取修飾字:
   // 資料成員宣告
   [資料型態 變數名稱[, 變數名稱]*];*

   // 成員函式宣告(與實作)
   [回傳型態 函式名稱([參數型態 參數名稱 [, 參數型態 參數名稱]*]?)
    [{
      // 函式實作
      程式敘述;*
    }]?]*
};
```

注意，若呼略存取修飾字，則類別所定義的資料成員與成員函式之權限預設值為private(若是結構體的話，其預設值為public)。

依據上述包含存取修飾字的類別定義語法，一個類別裡面可以視需求使用一個或一個以上的存取修飾字，來將特定部份的成員限制為不同的存取權線。下面的程式範例將所有的資料成員宣告為private並將所有的成員函式宣告為public意即資料成員不開放外界使用，但所有的成員函式都公開給外界使用：

```
class Student
{
private:
   string name;
   string SID;
   int    score;
public:
   bool isPass()
   {
      return score>=60;
   }
   void showInfo()
   {
      cout << name << "(" << SID << ") " << score << endl;
   }
};
```

17.3 供外部使用的界面: Setters and Getters

本章開頭處已經簡單說明過，資訊隱藏的作用。通常為了滿足物件的資訊隱藏需求，我們會把類別裡面所有的資料成員都宣告為私有的(private) — 意即完全不允許外界使用，但會另外提供一些公開的(public)成員函式，讓外界能夠用來存取資料成員。透過這種使用特定的成員函式來存取資料成員的做法，可確保物件的資料成員數值的正確性與安全性 — 因為外界無法任意存取它們的數值，只能透過這些特定的函式為之，而這些函式可以用來檢查對資料成員的存取是否會造成問題？是否在正確的數值範圍內？

這些特定供外界用以存取私有資料成員的函式，通常會被命名為公開的setXXX()與getXXX()成員函式，我們分別將它們稱做setter()函式與getter()函式：

- setter()又稱為Mutators其成員函式命名通常為setXXX()或是set_XXX()
- getter()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Student類別修改如下：

```
#ifndef _STUDENT_
```

```
#define _STUDENT_

class Student
{
private:
  string name;
  string SID;
  int    score;
public:
  Student (string n, string i, int s);
  bool   isPass();
  void   showInfo();
  void   setName(string n);
  string getName();
  void   setSID(string sid);
  string getSID();
  void   setScore(int s);
  int    getScore();
};
#endif
```

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

Student::Student(string n, string i, int s)
{
  name=n;
  SID=i;
  score=s;
}

bool Student::isPass()
{
   return score>=60;
}
void Student::showInfo()
{
   cout << name << "(" << SID << ") " << score << endl;
}

void Student::setName(string n)
{
  name=n;
}

void Student::setSID(string sid)
{
```

```
  SID=sid;
}

string Student::getName()
{
  return name;
}

string Student::getSID()
{
  return SID;
}

void Student::setScore(int s)
{
  if(s>100)
     score=100;
  else if(score<0)
     score=0;
  else
     score=s;
}

int Student::getScore()
{
  return score;
}
```

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

int main()
{
  Student *bob = new Student;

  bob->setName("Bob");
  bob->setSID("CBB01");
  bob->setScore(102);

  cout << bob->getName() << " " << bob->getSID() << endl;
  cout << "Score=" <<  bob->getScore() << endl;
  return 0;
}
```

在上面的程式當中，我們為所有私有的資料成員都設計了對應的setter與getter□其中針對setScore()函式，我們還特別針對成績超出合理範圍的情況做了處置 — 若是將score改為公開的資料成員，那麼就有可能發

生下面這種情況：

```
bob->score=101;
```

然而，現在透過封裝所實現的資訊隱藏，透過setter函式的幫助，我們就可以針對這種情況做處理，不用再擔心會被外界其它的程式碼將score設為不合理的數值。

17.4 this指標

C++語言在類別定義中為每個物件實體，預設了一個隱含的指標this□此指標會指向物件實體本身。請參考下的例子，我們為Student類別增加一個公開的成員函式，可用以比較自己與其它同樣是Student類別的物件實體，兩者間誰的成績比較高分，並傳回高分者的物件實體傳回：

```
#ifndef _STUDENT_
#define _STUDENT_

class Student
{
private:
  string name;
  string SID;
  int    score;
public:
  Student (string n, string i, int s);
  bool   isPass();
  void   showInfo();
  void   setName(string n);
  string getName();
  void   setSID(string sid);
  string getSID();
  void   setScore(int s);
  int    getScore();
  Student *compareScore(Student *anotherStudent); // 傳址呼叫的compareScore()
版本
  Student &compareScore(Student &anotherStudent); // 傳參考呼叫
的compareScore()版本
};
#endif
```

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

```
Student::Student(string n, string i, int s)
{
  name=n;
  SID=i;
  score=s;
}

bool Student::isPass()
{
   return score>=60;
}
void Student::showInfo()
{
   cout << name << "(" << SID << ") " << score << endl;
}

void Student::setName(string n)
{
  name=n;
}

void Student::setSID(string sid)
{
  SID=sid;
}

string Student::getName()
{
  return name;
}

string Student::getSID()
{
  return SID;
}

void Student::setScore(int s)
{
  if(s>100)
     score=100;
  else if(score<0)
     score=0;
  else
     score=s;
}

int Student::getScore()
{
  return score;
```

```
}

Student * Student::compareScore(Student *anotherStudent)
{
  if(score>(anotherStudent->score))
    return this;
  else
    return anotherStudent;
}

Student & Student::compareScore(Student &anotherStudent)
{
  if(score>(anotherStudent.score))
    return *this;
  else
    return anotherStudent;
}
```

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

int main()
{
  Student *bob = new Student;
  Student *robert = new Student;

  bob->setName("Bob");
  bob->setSID("CBB01");
  bob->setScore(80);

  robert->setName("Robert");
  robert->setSID("CBB02");
  robert->setScore(66);

  Student *higherScoreStudent;

  higherScoreStudent = bob->compareScore(robert);
  cout << higherScoreStudent->getName() << " ("
       << higherScoreStudent->getSID() << ") has higher score!" << endl;

  delete bob;
  delete robert;

  Student s1, s2;
  s1.setName("S1");
  s1.setSID("CBB01");
  s1.setScore(30);
```

```
  s2.setName("S2");
  s2.setSID("CBB02");
  s2.setScore(85);

  Student &s3=s2;
  s3=s1.compareAge(s2);
  cout << s3.getName() << " ("
       << s3.getSID << ") has higher score!" << endl;
  return 0;
}
```

[1)] 物件導向的四個主要特性包含抽象、封裝、繼承與多型。