C/C++基礎程式設計

C++: 類別的朋友關係、重載運算子、繼承

講師：張傑帆
CSIE, NTU

成就一番偉業的唯一途徑就是熱愛自己的事業。
Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work.
-Steve Jobs

課程大綱

- **類別: 朋友關係friend**
- 類別: 重載運算子
- **類別: 繼承**

Friend

- 在定義類別成員時，私有成員只能被同一個類別定義的成員存取，不可以直接由外界進行存取

- 然而有些時候，您希望提供私有成員給某些外部函式或類別來存取，這時您可以設定類別的「好友」，這些好友才可以直接存取私有成員。

- 使用friend通常是基於效率的考量，以直接存取私有成員而不透過函式呼叫的方式，來省去函式呼叫的負擔

Friend函式

- 思考：由建立出兩個正方型物件，並算出其面積之範例，你需要**提供一個函式**讓使用者可以比較兩正方型之大小，該如何達到此功能？

Friend函式

- 下面這個程式中使用friend關鍵字來設定類別的好友函式，該好友可以直接存取該類別的私有成員

```
#include <iostream>
using namespace std;
class Square
{
   public:
      Square(int n)
      {
         len = n;
      }
      int getLen()
      {
         return len;
      }
      int area()
      {
         return len*len;
      }
      friend int compare(Square &s1,
Square &s2);

   private:
      int len;
};
```

```
int compare(Square &s1, Square &s2)
{
   // 可直接存取私有成員
   if(s1.len == s2.len)
      return 0;
   else if(s1.len > s2.len)
      return 1;
   else
      return -1;
}
```

Friend函式

- 主程式可呼叫compare(s1, s2)比較兩方型大小

```
int main()
{
   Square s1(10);
   Square s2(20);

   cout << "s1: len = " << s1.getLen() << ", area = " << s1.area() << endl;
   cout << "s2: len = " << s2.getLen() << ", area = " << s2.area() << endl;

   switch(compare(s1, s2))
   {
      case 1:
         cout << "s1較大" << endl;
         break;
      case 0:
         cout << "s1跟s2一樣大" << endl;
         break;
      case -1:
         cout << "s2較大" << endl;
         break;
   }


   return 0;
}
```

Friend類別

- 您也可以將某個類別宣告為friend類別，被宣告為friend的類別可以直接存取私有成員

- 思考：由建立出兩個正方型物件，並算出其面積之範例，你需要提供一個尺類別讓使用者可以比較兩正方型之大小，該如何達到此功能？

Friend類別

- 下面這個程式中使用friend關鍵字來設定一類別為另一類別的好友

```
#include <iostream>
using namespace std;
class Square
{
   public:
      Square(int n)
      {
         len = n;
      }
      int getLen()
      {
         return len;
      }
      int area()
      {
         return len*len;
      }
      friend class Ruler;

   private:
      int len;
};
```

```
class Ruler
{
   public:
      Ruler(int n)
      {
         len = n;
      }
      void compareSquare(Square &s1, Square
&s2)
      {
         // 可直接存取私有成員
         if( (len < s1.len) || (len < s2.len) )
            cout << "尺太短, 無法量測" << endl;
         else
         {
            if(s1.len > s2.len)
               cout << "s1較大" << endl;
            else if(s1.len == s2.len)
               cout << "s1跟s2一樣大" << endl;
            else
               cout << "s2較大" << endl;
         }
      }
   private:
      int len;
};
```

Friend類別

- 主程式可建立Ruler物件比較兩方型大小

```
int main()
{
   Square s1(10);
   Square s2(20);
   Ruler r(30);

   cout << "s1: len = " << s1.getLen() << ", area = " <<
s1.area() << endl;
   cout << "s2: len = " << s2.getLen() << ", area = " <<
s2.area() << endl;

   r.compareSquare(s1, s2);

   return 0;
}
```

課程大綱

- 類別: 朋友關係friend
- 類別: 重載運算子
- 類別: 繼承

重載運算子

- 在C++中，預設除了基本資料型態可以使用運算子進行運算，例如int、double、char等，如果您要將兩個物件相加，預設上是不可行的。
- 然而很多情況下，您會想要將兩個物件的某些屬性值相加，並傳回運算後的結果
- 例如座標相加，如果您定義了Point2D類別，當中有x與y兩個屬性成員，您會想要透過+或-運算子的動作得到座標相加或相減的動作，在C++中，這可以透過重載運算子來達到目的。

重載運算子

- 運算子的重載其實是函式重載的一個延伸應用，您指定要重載哪一個運算子，並在類別中定義運算子如何動作，運算子重載的語法宣告如下所示
  - 傳回值 類別名稱::operator#(參數列) {
    // 實作重載內容
    }
- 其中#中需指明您要重載以下的運算子，例如重載一個+運算子，#處就替換為+運算子。

+	-	*	/	%	^	&	|
~	!	=	<	>	<=	>=	
++	--	<<	>>	==	!=	&&	||
+=	-+	/=	%=	^=	&=	|=	<<=
>>=	[]	()	->	->*	new	delete	

重載運算子

- 以下範例實作一個Point2D類別, 將類似定義描述在Point2D.h, 類別的方法描述在Point2D.cpp, 主程式描述在main.cpp, 試著建立起這個專案
- Point2D.h

```
class Point2D
{
    public:
        Point2D();
        Point2D(int x, int y);
        int getX();
        int getY();
        Point2D operator+(Point2D &p); // 重載+運算子
        Point2D operator-(Point2D &p); // 重載-運算子

    private:
        int X;
        int Y;
};
```

重載運算子

- Point2D.cpp

```
#include "Point2D.h"

Point2D::Point2D()
{
   X = 0;
   Y = 0;
}
Point2D::Point2D(int x, int y)
{
   X = x;
   Y = y;
}

int Point2D::getX()
{
   return X;
}

int Point2D::getY()
{
   return Y;
}
```

```
Point2D Point2D::operator+(Point2D &p)
{
   int x = X + p.X;
   int y = Y + p.Y;
   Point2D tmp(x, y);
   return tmp;
}
Point2D Point2D::operator-(Point2D &p)
{
   int x = X - p.X;
   int y = Y - p.Y;
   Point2D tmp(x, y);
   return tmp;
}
```

重載運算子

- main.cpp

```
#include <iostream>
#include "Point2D.h"
using namespace std;

int main()
{
    Point2D p1(5, 5);
    Point2D p2(10, 10);
    Point2D p3;

    p3 = p1 + p2;
    cout << "p3(x, y) = (" << p3.getX() << ", " << p3.getY() << ")" << endl;

    p3 = p2 - p1;
    cout << "p3(x, y) = (" << p3.getX() << ", " << p3.getY() << ")" << endl;

    return 0;
}
```

重載運算子

- 練習1: 承上範例，重載*運算子，讓Point2D物件之間可以做乘法運算

- 練習1: 承上範例，重載operator+函式並重載+運算子，讓Point2D物件之間可以做一個整數的加法運算，令其XY座標都加上該整數。

  ex:

```
Point2D p1(11,22),p2;
p2 = p1 + 10;
```

  這樣p2.X就會是21，p2.Y是32

課程大綱

- **類別: 朋友關係friend**
- 類別: 重載運算子
- **類別: 繼承**
  - 繼承的基礎
  - 子類別的建構與解構式
  - 三種繼承方式

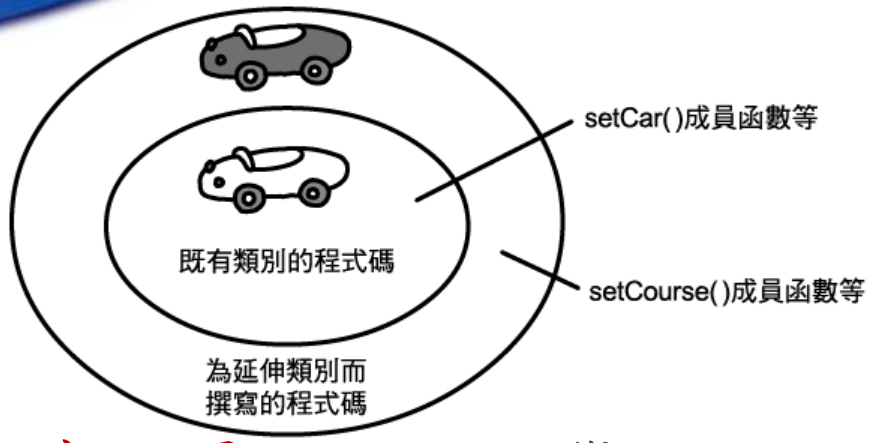
繼承（Inheritance）

- 「繼承」（Inheritance）是物件導向程式設計的一種進階觀念，**繼承就是物件的再利用**，當定義好一個類別後，其他類別可以繼承這個類別的成員資料和函數。
- **語法:**

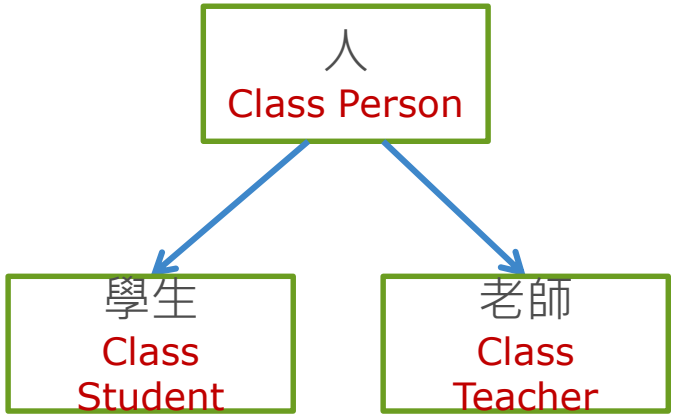
```
class 子類別名稱: 繼承權限 父類別名稱
{
    …
};
```

- **在繼承的關係中**
  - 被繼承的類別：
    - 「父類別」（Parent class）或「基礎類別」（Base class）
  - 繼承父類別的類別：
    - 「子類別」（Child class）或「衍生類別」（Derived class）

繼承（Inheritance）



- 範例:
  - 類別繼承也是在模擬真實世界，例如：學生和老師都是人，我們可以先定義Person類別來模擬人類，然後擴充Person類別建立Student類別來模擬學生，Teacher類別來模擬老師
  - http://homepage.ntu.edu.tw/~d02922022/C/Demo/Inheritance01.zip



繼承（Inheritance）

- 範例:

人
Class Person

```
class Person
{
   public:
      void inputPerson()
      {
         char str[128];
         cout << "<輸入個人資料>" << endl;
         cout << "姓名: ";
         fflush(stdin);
         cin.getline(str, 128);
         Name = str;
         cout << "電話: ";
         cin >> Phone;
         cout << "Email: ";
         cin >> Email;
      }
      void outputPerson()
      {
         cout << "<印出個人資料>" << endl;
         cout << "姓名: " << Name <<endl;
         cout << "電話: " << Phone <<endl;
         cout << "Email: " << Email <<endl;
      }
   private:
      string Name;
      string Phone;
      string Email;
};
```

Student

```
Student s1;
s1.inputPerson();
s1.inputStudent();
cout << endl;
s1.outputPerson();
s1.outputStudent();
```

```
class Student: public Person {
        public:
                void inputStudent() {
                        cout << "<輸入學生資料>" << endl;
                        cout << "學號: ";
                        cin >> StudentID;
                        cout << "系所: ";
                        cin >> Department;
                }
                void outputStudent() {
                        cout << "<印出學生資料>" << endl;
                        cout << "學號: " << StudentID <<endl;
                        cout << "系所: " << Department <<endl;
                }
        private:
                string StudentID;
                string Department;
};
```

Teacher

```
Teacher t1;
t1.inputPerson();
t1.inputTeacher();
cout << endl;
t1.outputPerson();
t1.outputTeacher();
```

```
class Teacher: public Person {
        public:
                void inputTeacher() {
                        cout << "<輸入老師資料>" << endl;
                        cout << "職稱: ";
                        cin >> Title;
                        cout << "系所: ";
                        cin >> Department;
                }
                void outputTeacher() {
                        cout << "<印出老師資料>" << endl;
                        cout << "職稱: " << Title <<endl;
                        cout << "系所: " << Department <<endl;
                }
        private:
                string Title;
                string Department;
};
```

繼承（Inheritance）

- 您可以宣告這些成員為「受保護的成員」(protected member)，保護的意思表示存取它有條件限制以保護該成員
- 當您將類別成員宣告為受保護的成員之後，繼承它的類別就可以直接使用這些成員，但這些成員仍然受到類別的保護，不可被物件直接呼叫使用。

建構式與解構式呼叫順序

- 當建立子類別的物件呼叫子類別的建構式時，它會先初始化父類別的成員，也就是呼叫父類別的建構式。如果子類別沒有建構式，在建立物件時，預設建構式就會呼叫父類別的預設建構式。

- 在呼叫子類別的建構式前，會先呼叫父類別的建構式，而解構式剛好與建構式是相反順序，也就是子類別的解構式是在父類別的解構式之前呼叫。

子類別傳遞參數給父類別

- 當父類別擁有**重載建構式**時，在子類別**可以傳遞參數**給父類別的建構式。
- 建構式的**:**運算子後是傳遞給父類別建構式的參數，如果父類別不只一個，請使用**,**號分隔。
- 其中傳遞給父類別參數的值，就是傳入子類別建構式的參數值。

繼承（Inheritance）

- 在繼承時您使用：運算子，並指定其繼承方式，在繼承的權限關係上，公開繼承是最常見的，先由這個開始說明繼承的概念。(Point2D.h)

```
class Point2D
{
    public:
        Point2D()
        {
            X = 0;
            Y = 0;
        }

        Point2D(int x, int y)
        {
            X = x;
            Y = y;
        }

        int getX()
        {
            return X;
        }
        int getY()
        {
            return Y;
        }

        void setX(int x)
        {
            X = x;
        }

        void setY(int y)
        {
            Y = y;
        }

    private:
        int X;
        int Y;
};
```

繼承（Inheritance）

- 接著定義Point3D類別，它公開繼承Point2D類別。(Point3D.h)

```
#include "Point2D.h"
class Point3D : public Point2D
{
    public:
        Point3D()
        {
            Z = 0;
        }
        Point3D(int x, int y, int z) : Point2D(x, y)
        {
            Z = z;
        }
        int getZ()
        {
            return Z;
        }
        void setZ(int z)
        {
            Z = z;
        }
    private:
        int Z;
};
```

繼承（Inheritance）

- 來看看一個使用Point3D的例子。(Point.cpp)

```
#include <iostream>
#include "Point3D.h"
using namespace std;

int main()
{
    Point3D p1(1, 3, 4);
    Point3D p2;
    cout << "p1: (" << p1.getX() << ", " << p1.getY() << ", " << p1.getZ()
<< ")" << endl;

    p2.setX(5);
    p2.setY(7);
    p2.setZ(8);
    cout << "p2: (" << p2.getX() << ", " << p2.getY() << ", " << p2.getZ()
<< ")" << endl;
    return 0;
}
```

三種繼承方式: 公開繼承

- class B : public A {
  // 實作
  };
- 公開繼承時使用public來繼承基底類別，繼承下來的成員在衍生類別中的權限變為如下：

基底類別	衍生類別
private	不繼承
protected	protected
public	public

三種繼承方式:保護繼承

- class B : protected A {
  // 實作
  };
- 保護繼承時使用protected來繼承基底類別，繼承下來的成員在衍生類別中的權限變為如下：

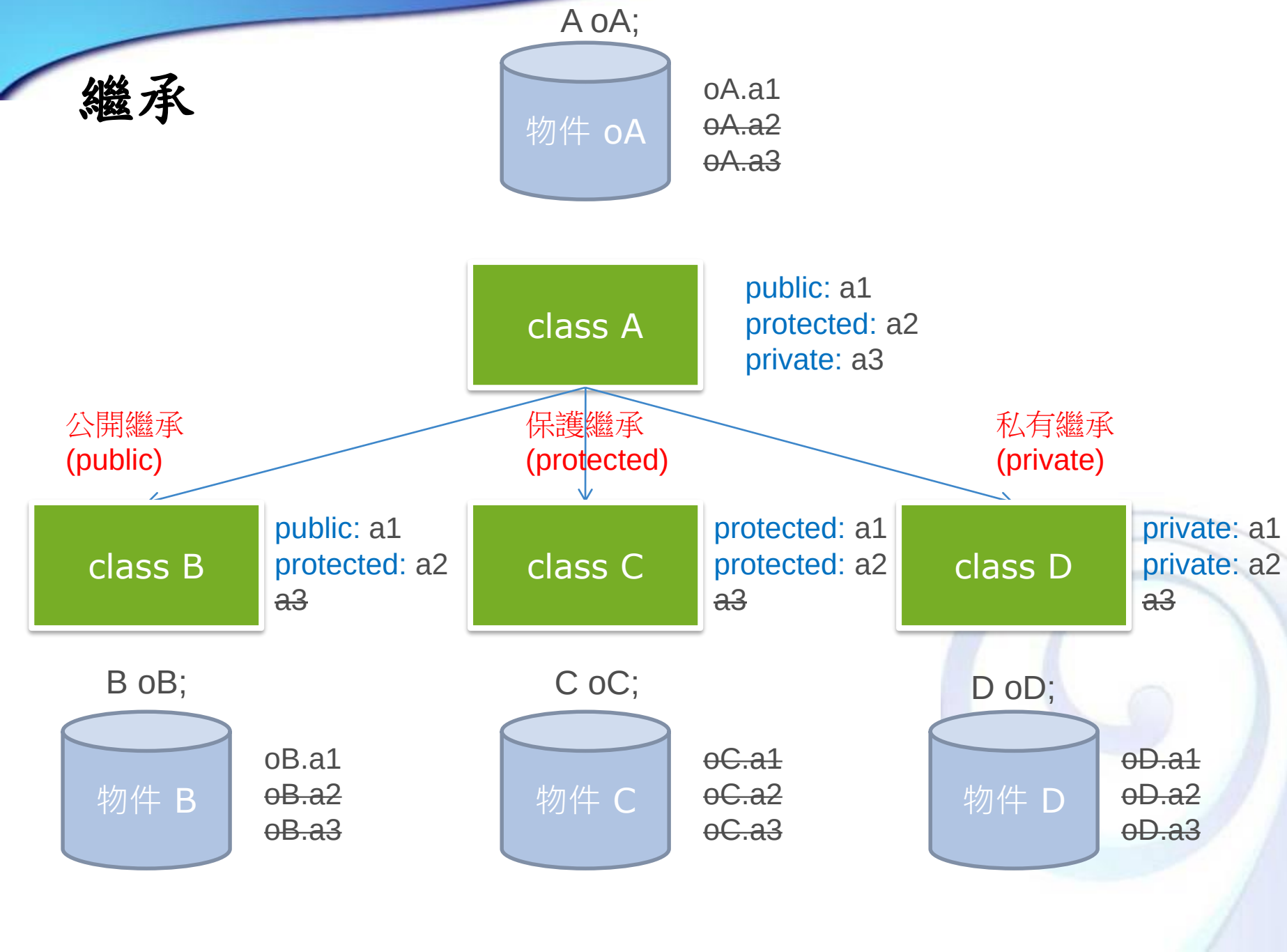
基底類別	衍生類別
private	不繼承
protected	protected
public	protected

三種繼承方式:私有繼承

- class B : private A {
  // 實作
  };
- 基底類別中的成員在被繼承之後，其權限如下所示：

基底類別	衍生類別
private	不繼承
protected	private
public	private

繼承


A oA;
物件 oA
oA.a1
~~oA.a2~~
~~oA.a3~~
class A
public: a1
protected: a2
private: a3
公開繼承
(public)
保護繼承
(protected)
私有繼承
(private)
class B
public: a1
protected: a2
~~a3~~
class C
protected: a1
protected: a2
~~a3~~
class D
private: a1
private: a2
~~a3~~
B oB;
物件 B
oB.a1
~~oB.a2~~
~~oB.a3~~
C oC;
物件 C
~~oC.a1~~
~~oC.a2~~
~~oC.a3~~
D oD;
物件 D
~~oD.a1~~
~~oD.a2~~
~~oD.a3~~

範例(1/2)

```
#include <iostream>
using namespace std;

class Point1D
{
    public:
        Point1D()
        {
            X = 0;
        }
        Point1D(int x)
        {
            X = x;
        }
        int getX()
        {
            return X;
        }
        void setX(int x)
        {
            X = x;
        }
    private:
        int X;
};
```

```
class Point2D : public Point1D
{
    public:
        Point2D()
        {
            Y = 0;
        }
        Point2D(int x, int y) : Point1D(x)
        {
            Y = y;
        }
        int getY()
        {
            return Y;
        }
        void setY(int y)
        {
            Y = y;
        }
    private:
        int Y;
};
```

範例(2/2)

```
class Point3D : public Point2D
{
    public:
        Point3D()
        {
            Z = 0;
        }
        Point3D(int x, int y, int z) :
Point2D(x, y)
        {
            Z = z;
        }
        int getZ()
        {
            return Z;
        }
        void setZ(int z)
        {
            Z = z;
        }
    private:
        int Z;
};
```

```
int main()
{
    Point3D p1(1, 3, 4);
    Point3D p2;
    cout << "p1: (" << p1.getX() << ", "
<< p1.getY() << ", " << p1.getZ() << ")"
<< endl;

    p2.setX(5);
    p2.setY(7);
    p2.setZ(8);
    cout << "p2: (" << p2.getX() << ", "
<< p2.getY() << ", " << p2.getZ() << ")"
<< endl;
    return 0;
}
```

表 14-1　存取指定子

基底類別中之存取指定	繼承的方法	可否從衍生類別利用	可否從外部利用
public	public	可	可
protected		可	不可
private		不可	不可
public	protected	可	不可（用了就會變成衍生類別的 protected 成員）
protected		可	不可
private		不可	不可
public	private	可	不可（用了就會變成衍生類別的 private 成員）
protected		可	不可
private		不可	不可