10-2 存取控制——public 與 private

一、class 裡的東西，不是誰都可以動的

先看一個銀行帳戶的例子。

```
#include <iostream>

using namespace std;

class BankAccount {
public:
    string owner;
    int balance;
};

int main() {
    BankAccount acc;
    acc.owner   = "小明";
    acc.balance = 1000;

    acc.balance = -99999;   // 沒有任何阻擋，帳戶餘額變成負的！

    cout << acc.balance << endl;
}
```

這是因為我們之前在 class 裡用了 `public:` 這個存取修飾詞。 public 的意思就是「公開的」，誰想看、誰想改都可以。

用 `private:` 將存取限制改為「私有的」看看差別。

```
#include <iostream>

using namespace std;

class BankAccount {
private:
    string owner;
```

```
    int balance;
};

int main() {
    BankAccount acc;
    acc.owner   = "小明";
    acc.balance = 1000;

    acc.balance = -99999;   // 沒有任何阻擋，帳戶餘額變成負的！

    cout << acc.balance << endl;
}
```

修改之後，連編譯都失敗，出現了大量如下的錯誤訊息。

因為使用 privte 修飾的屬性和成員函數，在該 class 之外(第 5~9 行之外)都無法存取，在編譯時就攔下了不合法的存取。

```
main.cpp:13:9: error: 'std::string BankAccount::owner' is private within this context
   13 |     acc.owner   = "小明";
      |         ^~~~
main.cpp:7:12: note: declared private here
    7 |     string owner;
      |            ^~~~~
main.cpp:14:9: error: 'int BankAccount::balance' is private within this context
   14 |     acc.balance = 1000;
      |         ^~~~~~~
main.cpp:8:9: note: declared private here
    8 |     int balance;
      |         ^~~~~~~
main.cpp:16:9: error: 'int BankAccount::balance' is private within this context
   16 |     acc.balance = -99999;   // 沒有任何阻擋，帳戶餘額變成負的！
      |         ^~~~~~~
main.cpp:8:9: note: declared private here
    8 |     int balance;
      |         ^~~~~~~
main.cpp:18:17: error: 'int BankAccount::balance' is private within this context
   18 |     cout << acc.balance << endl;
      |                 ^~~~~~~
main.cpp:8:9: note: declared private here
    8 |     int balance;
      |         ^~~~~~~
```

二、提供外界有限的操作

設成 public 太危險，設成 private 又動不了，怎麼辦呢?

我們可以

- 把內部屬性設為 private
- 用 public 開放幾個成員函數，給外界有限能力操作內部屬性

```
#include <iostream>

using namespace std;

class BankAccount {
private:
    string owner;
    int balance;    // 外部無法直接存取

public:
    // 以下成員函數，開放給外部使用，提供存取 balance 的管道
    bool withdraw(int amount) {
        if (amount > balance) {
            cout << "餘額不足！" << endl;
            return false;
        }
        balance -= amount;
        return true;
    }

    void deposit(int amount) {
        if (amount <= 0) {
            cout << "存款金額必須大於 0！" << endl;
            return;
        }
        balance += amount;
    }

    int getBalance() { return balance; }
};

int main()
{
    BankAccount account;
    account.deposit(1000);  // 存款 1000 元
    cout << "目前餘額: " << account.getBalance() << " 元" << endl;

    if (account.withdraw(500)) {  // 提款 500 元
        cout << "提款成功！" << endl;
    }
    cout << "目前餘額: " << account.getBalance() << " 元" << endl;

    if (!account.withdraw(600)) {  // 嘗試提款 600 元，應該會失敗
        cout << "提款失敗！" << endl;
```

```
    }
    cout << "目前餘額: " << account.getBalance() << " 元" << endl;

    return 0;
}
```

現在外部程式只能透過 `withdraw()` 和 `deposit()` 來改變餘額，透過 `getBalance()` 來取得餘額，非法操作會被擋下來。

public、private 存取修飾詞的作用範圍

```
class Example {
// 這裡沒寫 → class 預設是 private
    int secret;

public:
    int visible;
    void doSomething() { }

private:
    int alsoSecret;
};
```

`public:` 和 `private:` 後面的所有成員，都套用該修飾子，直到遇到下一個修飾子為止。

三、封裝(encapsulate)的概念

有時候我們不希望 class 的使用者直接去更動某些 data member 的資料，這時我們會將這些 data member 宣告在 private 區域中，並在 public 區域中提供 member function 來操作這些 data member。

以手錶為例，一般來說製表師傅不會希望使用者自己去撥動錶內的齒輪，所以他會把手錶內部的複雜機構用錶殼緊緊的封裝起來，不過他還是會提供一些按鈕或旋鈕來讓我們調整時間。

- private – 手錶的內部複雜機構
- public – 調整時間的按鈕或旋鈕

當我們試著在非成員函數裡存取 private 的 data member 或 member function 時，在編譯時期就會出現錯誤訊息。

這就是「物件導向程式設計」中很重要的「封裝」概念。我們設計了一個很精巧的 class，包含了很多的屬性和成員函數，但是對外界來說，它被一個「盒子」封裝起來，你只操作這個盒子外漏的那幾個界面。

Revision #3
Created 2026-05-30 23:35:35 UTC by huihui
Updated 2026-05-31 00:32:10 UTC by huihui