



05 Inheritance (繼承)

Fundamental Computer Programming- C++ Lab (II)

元智大學 | C++ 程式設計實習 (II)

張維元

課程投影片：[請從元智個人 Portal 下載](#)

Outline

- 01 物件導向與開發環境
- 02 陣列、向量與結構
- 03 類別與物件
- 04 物件導向程式設計: 多載
- 05 物件導向程式設計: 繼承
- 06 樣板
- 07 C++ 進階用法

什麼是程式？

程式語言是用來命令電腦執行各種作業的工具，是人與電腦溝通的橋樑。當電腦藉由輸入設備把程式讀入後，會儲存在主記憶體內，然後指令會依序被控制單元提取並解碼或翻譯成電腦可以執行的信號，並把信號送到各個裝置上，以執行指令所指派的動作。也就是說，人類與電腦溝通的語言稱為程式語言。

程式 = 利用一系列的指令告訴電腦如何執行工作



物件導向程式設計

物件導向程式設計（Object-oriented programming，OOP）是種具有物件概念的程式設計典範，同時也是一種程式開發的抽象方針。它可能包含資料、屬性、程式碼與方法。物件則指的是類別（class）的實例。它將物件作為程式的基本單元，將程式和資料封裝其中，以提高軟體的重用性、靈活性和擴充性。

把物件作為程式最小的單位，模擬實體世界的運作



類別與物件

- 類別 (Class) 利用屬性與方法定義了物件的抽象樣板/結構
- 物件 (Object) 是類別的實體 (instance) ，可以使用類別方法

```
1  
2  
3 class Book {  
4     public:  
5         string title;  
6         int price;  
7 };  
8  
9
```

```
1 int main( ){  
2  
3     Book book1;  
4  
5     book1.title = "C++ How to Program";  
6     book1.price = 636;  
7  
8     cout << book1.title << endl;  
9     cout << book1.price << endl;  
10    return 0;  
11  
12 }
```

→ 定義了一個叫做「Book」的變數型態

結構與類別的差別

| 差別 | C 語言 Struct | C++ 語言 Struct | C++ 語言 Class |
|--------|-------------|------------------------|------------------------|
| 成員變數 | 不支援靜態變數 | 支援靜態變數 | 支援靜態變數 |
| 成員函式 | 不支援成員函式 | 支援成員函式 | 支援成員函式 |
| 成員預設值 | 不支援成員預設值 | 支援成員預設值 | 支援成員預設值 |
| 成員範圍 | public | public、protect、private | public、protect、private |
| 預設成員範圍 | X | public | private |
| 物件導向特性 | X | 0 | 0 |

→ C++ Class 就像是多了繼承特性的 C 語言 Struct

成員函式

(member function)

```
1
2 class Book {
3     public:
4         string title;
5         int price; ← 成員變數
6         void f(void); ← 成員函式
7 };
8
9 void Book::f(){
10     cout << "I'm a book: " + title;
11 };
12     → 成員函式可存取成員變數
```

```
1
2 int main( ){
3
4     Book book1;
5     book1.title = "C++ How to Program";
6     book1.f();
7
8     return 0;
9 }
10
11
12
```

利用公開成員函式存取私有成員變數

```
1
2 class Book {
3     private: ← 私有成員
4         int price;
5     public: ← 公開函式
6         void set(int p);
7         double get(void);
8 };
9
10 void Book::set(int p){
11     price = p;
12 }; → 成員函式可存取成員變數
13
14 double Book::get(void){
15     return price * 0.9;
16 };
```

```
1 int main( ){
2
3     Book book1;
4     book1.set(600);
5     cout << "price: " << book1.get();
6
7     return 0;
8 }
9
10
```


static 修飾子 = 靜態成員變數

→ static 是同一個 class 下，所有 object 共享的變數

```
1
2 class Book {
3     static int i;
4     public:
5         int id;
6         void f(void);
7 };
8 int Book::i = 20210330; ← static
9                             要記得初始化
10 void Book::f(void){
11     cout << "I'm a book: " << id;
12     cout << "i = " << i++ << endl;
13 };
```

```
1
2 int main( ){
3
4     Book books[10];
5     for(int i = 0; i < 10; i++){
6         books[i].id = i;
7         books[i].f();
8     }
9     return 0;
10 }
```

friend class

在定義類別成員時，私有成員只能被物件本身存取，無法直接由外界存取。有一種特殊的類別關係稱為「**好友類別**」，得以讓其存取私有成員。

```
1
2 int main(){
3     Girl girl1("Mary", 7);
4     Girl girl2("Alice", 10);
5
6     Boy boy1("Tom");
7     boy1.like(girl1, girl2);
8
9     return 0;
10 }
11
```

```
1
2 class Girl {
3     string _name;
4     int _score;
5     public:
6         Girl (string name, int score){
7             _name = name;
8             _score = score;
9         }
10     friend class Boy;
11 };
```

friend class

在定義類別成員時，私有成員只能被物件本身存取，無法直接由外界存取。有一種特殊的類別關係稱為「**好友類別**」，得以讓其存取私有成員。

```
1
2 int main(){
3     Girl girl1("Mary", 7);
4     Girl girl2("Alice", 10);
5
6     Boy boy1("Tom");
7     boy1.like(girl1, girl2);
8
9     return 0;
10 }
11
```

```
1 class Boy {
2     string _name;
3     public:
4         Boy(string name){
5             _name = name;
6         }
7         void like(Girl g1, Girl g2){
8             return g1._score > g2._score ?
9                 g1._name : g2._name;
10        }
11 }
```

未指定的成員預設是私有變數

→ _ 開頭的變數是一種變數命名慣例，用來代表私有變數

```
1 class Girl {  
2     string _name;  
3     int _score;  
4     public:  
5         Girl (string name, int score){  
6             _name = name;  
7             _score = score;  
8         }  
9         friend class Boy;  
10    };  
11
```

```
1 class Boy {  
2     string _name;  
3     public:  
4         Boy(string name){  
5             _name = name;  
6         }  
7         void like(Girl g1, Girl g2){  
8             return g1._score > g2._score ?  
9                 g1._name : g2._name;  
10        }  
11    }
```

物件導向程式設計

物件導向程式設計（Object-oriented programming，OOP）是種具有物件概念的程式設計典範，同時也是一種程式開發的抽象方針。它可能包含資料、屬性、程式碼與方法。物件則指的是類別（class）的實例。它將物件作為程式的基本單元，將程式和資料封裝其中，以提高軟體的重用性、靈活性和擴充性。

把物件作為程式最小的單位，模擬實體世界的運作



class

object

class



Car

objects



Audi



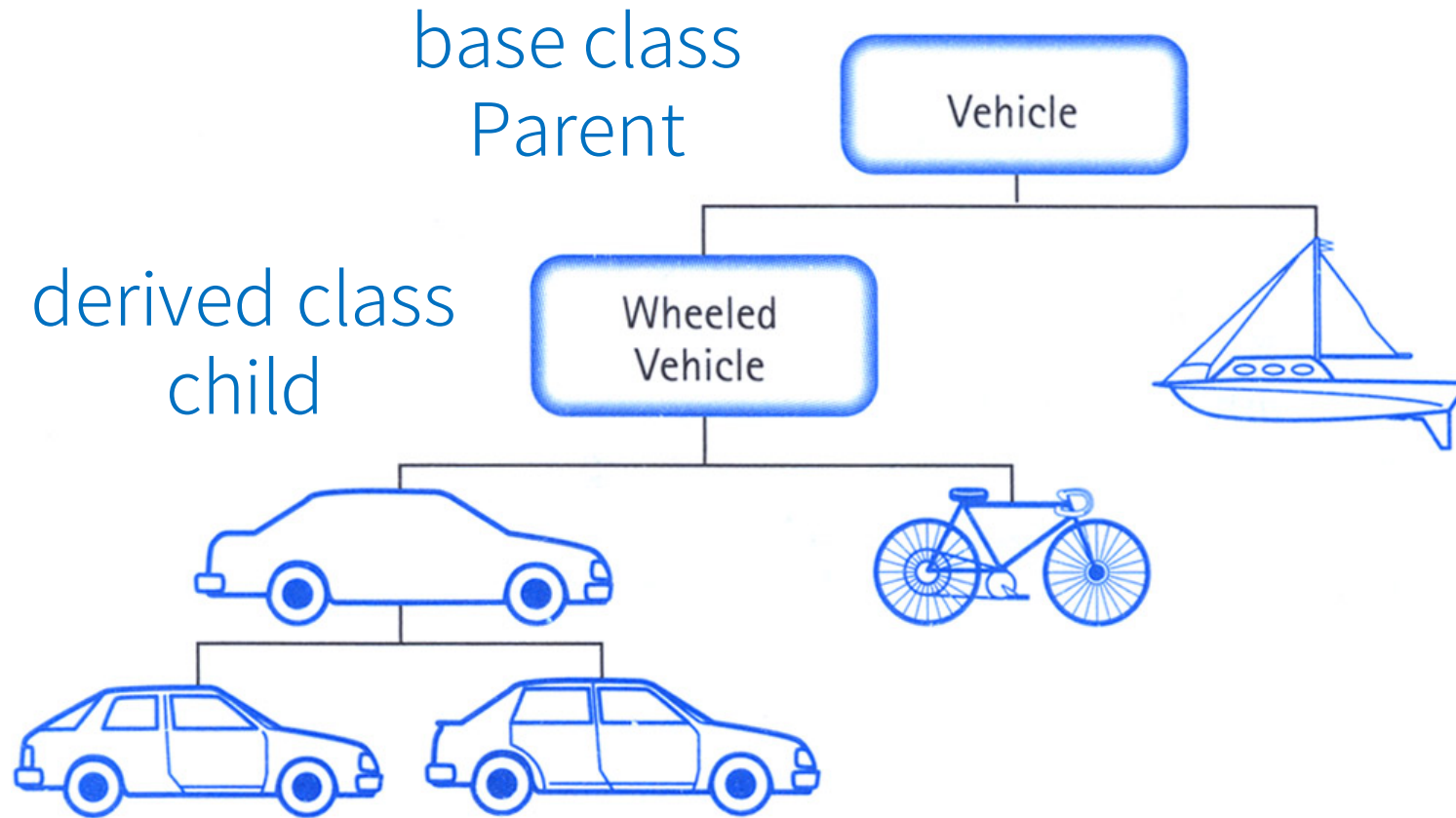
Nissan



Volvo

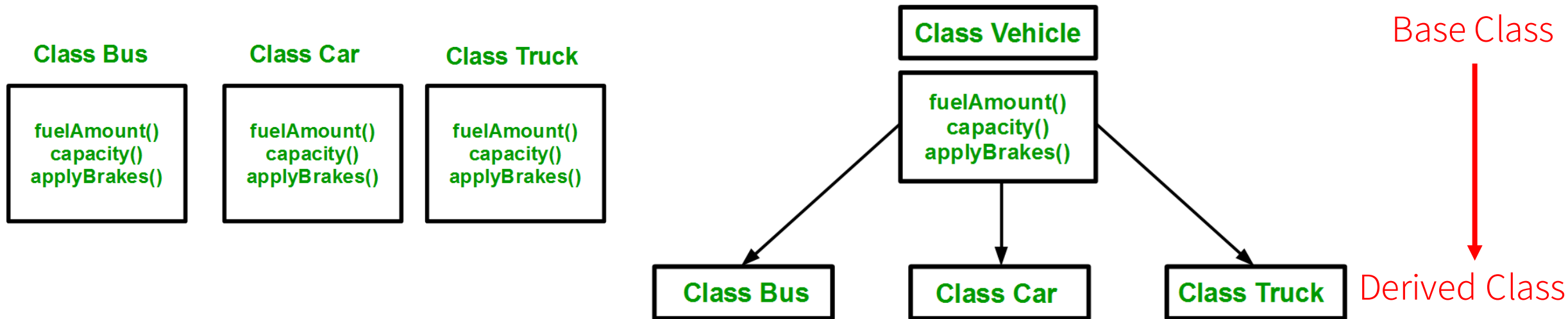
→ Class 是抽象的描述， Object 才是實體的存在

Inheritance



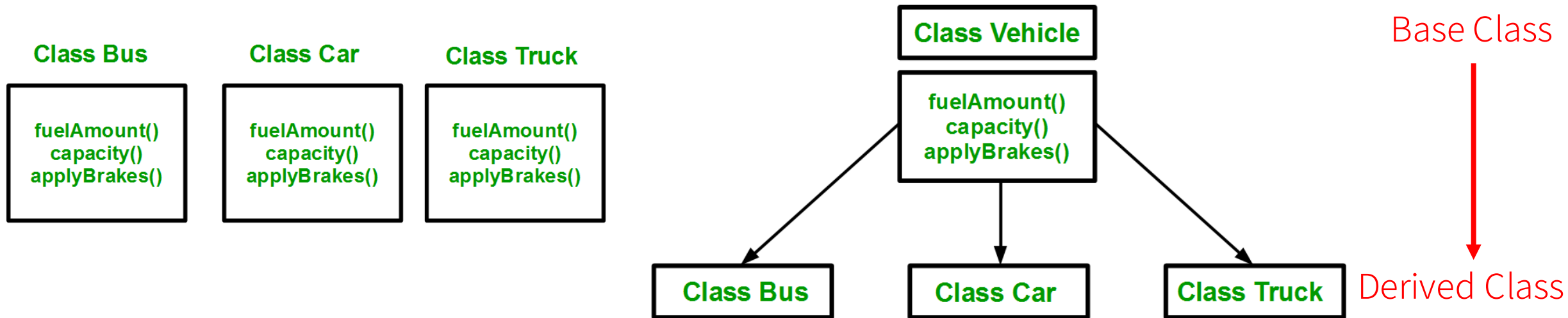
繼承 (Inheritance)

繼承是物件導向的一個概念，繼承可以使得子類別具有父類別的各種屬性和方法，子類別也可以重新定義某些屬性或改寫原本的方法。



繼承 (Inheritance)

繼承是物件導向的一個概念，繼承可以使得子類別具有父類別的各種屬性和方法，子類別也可以重新定義某些屬性或改寫原本的方法。



繼承

(Inheritance)

繼承是物件導向的一個概念，繼承可以使得子類別具有父類別的各種屬性和方法，子類別也可以重新定義某些屬性或改寫原本的方法。

```
1
2
3 class Shape {
4     public:
5         int width, height;
6         void setWidth(int w){
7             width = w;
8         }
9         void setHeight(int h){
10            height = h;
11        }
12};
```

```
1
2
3         → Rectangle 是繼承自 Shape
4 class Rectangle: public Shape {
5     public:
6         int getArea(){
7             return (width * height);
8         }
9 };
10
11
12
```

子類別無法存取父類別的私有成員

```
1
2 class Shape {
3     private:
4         int width, height;
5     public:
6         void setWidth(int w){
7             width = w;
8         }
9         void setHeight(int h){
10            height = h;
11        }
12};
```

```
1
2
3     → Rectangle 是繼承自 Shape
4 class Rectangle: public Shape {
5     public:
6         int getArea(){
7             return (width * height);
8         }
9 };
10
11
12
```

子類別可以存取父類別 protected 成員

```
1
2 class Shape {
3     protected:
4         int width, height;
5     public:
6         void setWidth(int w){
7             width = w;
8         }
9         void setHeight(int h){
10             height = h;
11         }
12 };
```

```
1
2
3     → Rectangle 是繼承自 Shape
4 class Rectangle: public Shape {
5     public:
6         int getArea(){
7             return (width * height);
8         }
9 };
10
11
12
```

類別成員的三種 Scope

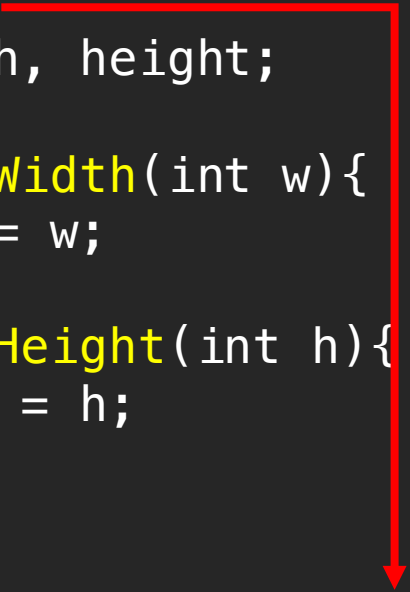
→ 主要差別在於成員被存取的權限

| |
|--|
| <div><div>Data</div><div>Page No.</div></div> <h2><u>C++ Access Specifier</u></h2> |
|--|

子類別可以繼承父類別擁有的成員

```
1 class Shape {
2     protected:
3         int width, height;
4     public:
5         void setWidth(int w){
6             width = w;
7         }
8         void setHeight(int h){
9             height = h;
10        }
11 };
12
13 class Rectangle: public Shape {
14     public:
15         int getArea(){
16             return (width * height);
17         }
18 };
```

Inheritance



```
1
2
3 int main(void){
4     Rectangle Rect;
5
6     Rect.setWidth(5);
7     Rect.setHeight(7);
8     cout << Rect.getArea() << endl;
9
10    return 0;
11 }
12
```

建構子 (constructor)

建構子（constructor；ctor），是一個類別裡用於建立物件時初始化成員變數函式。在初始化一個新建的物件，能夠同時輸入參數用以設定實例變數。

```
1
2 class Book {
3     public:
4         int price;
5         string title;
6         Book(int p){
7             price = p;
8         };
9     };
10
```


```
1
2
3 int main( ){
4     Book book1(600);
5     cout << "price: " << book1.price;
6     return 0;
7 }
8
9
10
```

建構子 (constructor)

建構子（constructor；ctor），是一個類別裡用於建立物件時初始化成員變數函式。在初始化一個新建的物件，能夠同時輸入參數用以設定實例變數。

```
1
2 class Book {
3     public:
4         int price;
5         string title;
6         Book(int p){
7             price = p;
8         };
9     };
10
```

```
1
2
3 int main( ){
4     Book book1(600);
5     cout << "price: " << book1.price;
6     return 0;
7 }
8
9
10
```



繼承的建構式

→ 先呼叫父類別的建構子

```
1
2 int main(void){
3     Rectangle Rect(3, 5);
4
5     cout << Rect.getArea() << endl;
6
7     return 0;
8 }
9
```

```
1
2 class Shape {
3     protected:
4         int width, height;
5     public:
6         Shape(int a, int b){
7             width = a;
8             height = b;
9         }
10};
```

```
1 class Rectangle: public Shape {
2     int area;
3     public:
4     Rectangle(int x, int y): Shape(x, y){
5         area = width * height;
6     }
7     int getArea(){
8         return area;
9     }
10};
```

建構式多載

```
1
2 int main(void){
3     Shape s1(3, 4);
4     cout << s1.getArea() << endl;
5
6     Shape s2(s1);
7     cout << s2.getArea() << endl;
8
9     Shape s3 = s1;
10    cout << s3.getArea() << endl;
11
12    s3 = s2;
13    cout << s3.getArea() << endl;
14
15    return 0;
16 }
17
```

```
1 class Shape {
2     public:
3         int width, height;
4         Shape(int a, int b);
5         Shape &operator=(const Shape &obj){
6             cout << "operator=()" << endl;
7             width = obj.width;
8             height = obj.height;
9             return *this;
10        }
11 };
12
```

```
1 Shape::Shape(int a, int b){
2     cout << "Shape constructor()" << endl;
3     width = a;
4     height = b;
5 }
6
7 Shape::Shape(const Shape& obj){
8     width = obj.width;
9     height = obj.height;
10    cout << "Copy Constructor()" << endl;
11 }
```

→ 宣告物件時會進入符合參數的建構子

建構式多載

→ 相同函式名稱，不同參數

```
1
2 int main(void){
3     Shape s1(3, 4);
4     cout << s1.getArea() << endl;
5
6     Shape s2(s1);
7     cout << s2.getArea() << endl;
8
9     Shape s3 = s1;
10    cout << s3.getArea() << endl;
11
12    s3 = s2;
13    cout << s3.getArea() << endl;
14
15    return 0;
16 }
17
```

```
1 class Shape {
2     public:
3         int width, height;
4         Shape(int a, int b);
5         Shape &operator=(const Shape &obj){
6             cout << "operator=()" << endl;
7             width = obj.width;
8             height = obj.height;
9             return *this;
10        }
11 };
12
```

```
1 Shape::Shape(int a, int b){
2     cout << "Shape constructor()" << endl;
3     width = a;
4     height = b;
5 }
6
7 Shape::Shape(const Shape& obj){
8     width = obj.width;
9     height = obj.height;
10    cout << "Copy Constructor()" << endl;
11 }
```

→ 宣告物件時會進入符合參數的建構子

Thanks for listening.

元智大學 | C++ 程式設計實習

Wei-Yuan Chang