10-1 類別(class)與物件(object)

一、物件（object）與類別（class）

在 C++ 中物件和類別有很嚴謹的定義，我們在這裡僅使用簡單例子來做介紹。

什麼是物件(Object)？你的iPhone是個物件、你現在坐著的這張椅子是個物件，小狗小黃也是個物件。

那類別呢(Class)？你的iPhone是個物件，隔壁同學的HTC One也是個物件，而它們同屬於「手機」這個類別。小黃是個物件、小黑也是個物件，牠們同屬於「狗」這個類別。

二、建立類別與產生物件

在產生物件之前必須先建立類別，在C++中建立類別要使用到 class 這個關鍵字，一個最簡單的類別長這個模樣(注意：最後有一個分號)。

```
class 類別名稱 {
};
```

假設我們要建立一個dog類別，可以這樣寫。

```
class dog {
};
```

接下來我們可以這樣產生兩個 dog 類別的物件：shiro（小白） 和 kuro（小黑）。

```
dog shiro;
dog kuro;
```

或

```
dog shiro, kuro;
```

就像在定義 int 或 float 等原生型別變數一樣。

三、屬性 (attribute)/資料成員 (data member)

前面這個 dog類別是一個很單純的類別，它產生的兩個物件也沒什麼用處。現在我們要給它一點變化，讓狗有不同的顏色和叫聲。

```
class dog {
  public:
    string color;
    string sound;
};
```

public 關鍵字表示下面出現的屬性都是「公開的」，你可以在程式的任何地方來改變其值。

在這個例子中我們有兩個 string 型別的屬性 color 和 sound。我們可以用 `物件名稱.屬性名稱` 的型式來設定或取用特定物件的某個屬性值。

```
dog shiro;
dog kuro;

shiro.color = "白色";
kuro.color = "黑色";

cout << "shiro 的顏色是" << shiro.color << endl;
cout << "kuro 的顏色是" << kuro.color << endl;
```

四、成員函數(member function)

到目前為止，你會覺得 class 和之前學的 struct 一樣。

但是除了屬性之外，我們還可以在類別中加入成員函數，讓該類別的物件可以「做些事情」，例如：我們可以在 dog 類別中加入 bark 這個成員函數，讓小白和小黑可以叫。

```
class dog {
  public:
    string color;
    string sound;

    void bark() {
        cout << sound << endl;
    }
};

int main()
```

```
{
    dog shiro, kuro;

    shiro.color = "白色";
    shiro.sound = "汪！";
    kuro.color = "黑色";
    kuro.sound = "汪汪汪！";

    cout << "shiro 咬他！" << endl;
    shiro.bark();                    // 汪！
    cout << "kuro 咬他！" << endl;
    kuro.bark();                     // 汪汪汪！

    return 0;
}
```

五、特殊的成員函數- 建構(constructor), 解構(destructor)

constructor 是一個和類別同名的成員函數，在一個物件被產生時會自動被呼叫，而且 **沒有傳回值**

```
class dog {
  public:
    string color;
    string sound;

    void bark() {
        cout << sound << endl;;
    }
    dog() {
        cout << "有一隻小狗誕生了！" << endl;
    }
};

int main()
{
    dog shiro, kuro;

    return 0;
}
```

一個類別可以有很多個 constructor，每個 constructor 的參數列都必須不一樣，在產生物件時，根據你使用參數的不同，相對應的 constructor 會自動被呼叫。

沒有參數的 constructor 稱為 **`default constructor`**。

```
class dog {
  public:
    string color;
    string sound;

    void bark() {
        cout << sound << endl;;
    }
    dog() {
        cout << "有一隻小狗誕生了！" << endl;
    }
    dog(string _color, string _sound) {
        color = _color;
        sound = _sound;
        cout << "有一隻" << color << "小狗誕生了！" << sound << endl;
    }
};

int main()
{
    dog kuro;
    dog shiro("白色","汪！");

    return 0;
}
```

destructor 是物件「死亡」時會被自動呼叫的成員函數，每個類別只有一個 destructor，它的名字是 ~類別名稱，**`destructor 沒有參數也沒有傳回值`**。

```
class dog {
  public:
    string color;
    string sound;
    string name;

    void bark() {
        cout << sound << endl;;
    }

    dog(string _name, string _color, string _sound) {
        name = _name;
        color = _color;
        sound = _sound;
        cout << "有一隻名為 " << name << " 的 " << color << " 小狗誕生了！"
                << sound << endl;
    }
    ~dog() {
        cout << name << " 上天堂了。" << endl;
    }
```

```
};

int main()
{
    dog shiro("小白", "白色", "汪！");
    dog *kuro = new dog("小黑", "黑色","喵！");

    delete kuro;

    return 0;
}
```

六、使用指標操作物件

使用指標操作物件與使用指標操作 struct 變數一樣，必須用到「->」。

```
int main()
{
    dog *kuro = new dog("小黑", "黑色","喵！");

    kuro->sound = "汪！汪！";
    kuro->bark();

    delete kuro;

    return 0;
}
```

七、this 指標

this 是一個指向「自己這個物件」的指標。

成員函數的參數名稱和屬性名稱完全一樣時，不加 this-> 編譯器會把兩邊都當作參數，屬性永遠不會被賦值。

```
#include <iostream>

using namespace std;

class Student {
public:
    string name;
    int age;
```

```
    void setName(string name) {
        //name = name;         // 自己賦值給自己，成員變數沒有被改到
        this->name = name;     // 左邊是成員變數，右邊是參數
    }
    void setAge(int age) {
        //age = age;           // 自己賦值給自己，成員變數沒有被改到
        this->age = age;       // 同上
    }

    void printInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};


int main()
{
    Student std;
    std.setName("Alice");
    std.setAge(20);

    std.printInfo();

    return 0;
}
```

使用 this

Name: Alice, Age: 20
不使用 this

Name: , Age: -103249728

Revision #3
★Created 2026-05-30 23:20:31 UTC by huihui
Updated 2026-06-01 10:48:13 UTC by huihui