# 10.2 抽象和类

生活中充满复杂性，处理复杂性的方法之一是简化和抽象。人的身体是由无数个原子组成的，而一些学者认为人的思想是由半自主的主体组成的。但将人自己看作一个实体将简单得多。在计算中，为了根据信息与用户之间的接口来表示它，抽象是至关重要的。也就是说，将问题的本质特征抽象出来，并根据特征来描述解决方案。在垒球统计数据示例中，接口描述了用户如何初始化、更新和显示数据。抽象是通往用户定义类型的捷径，在C++中，用户定义类型指的是实现抽象接口的类设计。

## 10.2.1 类型是什么

我们来看看是什么构成了类型。例如，讨厌鬼是什么？受流行的固定模式影响，可能会指出讨厌鬼的一些外表特点：胖、戴黑宽边眼镜、兜里插满钢笔等。稍加思索后，又可能觉得从行为上定义讨厌鬼可能更合适，如他（或她）是如何应对尴尬的社交场面的。如果将这种类比扩展到过程性语言（如C语言），我们得到类似的情形。首先，倾向于根据数据的外观（在内存中如何存储）来考虑数据类型。例如，char占用1个字节的内存，而double通常占用8个字节的内存。但是稍加思索就会发现，也可以根据要对它执行的操作来定义数据类型。例如，int类型可以使用所有的算木运算，可对整数执行加、减、乘、除运算，还可以对它们使用求模运算符（%）。

而指针需要的内存数量很可能与int相同，甚至可能在内部被表示为整数。但不能对指针执行与整数相同的运算。例如，不能将两个指针相乘，这种运算是没有意义的，因此C++没有实现这种运算。因此，将变量声明为int或float指针时，不仅仅是分配内存，还规定了可对变量执行的操作。总之，指定基本类型完成了三项工作：

* 决定数据对象需要的内存数量；
* 决定如何解释内存中的位（long和float在内存中占用的位数相同，但将它们转换为数值的方法不同）；
* 决定可使用数据对象执行的操作或方法。

对于内置类型来说，有关操作的信息被内置到编译器中。但在C++中定义用户自定义的类型时，必须自己提供这些信息。付出这些劳动换来了根据实际需要定制新数据类型的强大功能和灵活性。

## 10.2.2 C++中的类

类是一种将抽象转换为用户定义类型的C++工具，它将数据表示和操纵数据的方法组合成一个整洁的包。下面来看一个表示股票的类。

首先，必须考虑如何表示股票。可以将一股作为基本单元，定义一个表示一股股票的类。然而，这意味着需要100个对象才能表示100股，这不现实。相反，可以将某人当前持有的某种股票作为一个基本单元，数据表示中包含他持有的股票数量。一种比较现实的方法是，必须记录最初购买价格和购买日期（用于计算纳税）等内容。另外，还必须管理诸如如拆股等事件。首次定义类就考虑这么多因素有些困难，因此我们对其进行简化。具体地说，应该将可执行的操作限制为：

* 获得股票；
* 增持；
* 卖出股票；
* 更新股票价格；
* 显示关于所持股票的信息。

可以根据上述清单定义Stock类的公有接口。为支持该接口，需要存储一些信息。我们再次进行简化。例如，不考虑标准的美式股票计价方式（八分之一美元的倍数）。我们将存储下面的信息：

* 公司名称；
* 所持股票的数量；
* 每股的价格；
* 股票总值。

接下来定义类。一般来说，类规范由两个部分组成。

* 类声明：以数据成员的方式描述数据部分，以成员函数（被称为方法）的方式描述公有接口。
* 类方法定义：描述如何实现类成员函数。

简单地说，类声明提供了类的蓝图，而方法定义则提供了细节。

**什么是接口**

接口是一个共享框架，供两个系统（如在计算机和打印机之间或者用户或计算机程序之间）交互时使用；例如，用户可能是您，而程序可能是字处理器。使用字处理器时，您不能直接将脑子中想到的词传输到计算机内存中，而必须同程序提供的接口交互。您敲打键盘时，计算机将字符显示到屏幕上；您移动鼠标时，计算机移动屏幕上的光标；您无意间单击鼠标时，计算机对您输入的段落进行奇怪的处理。程序接口将您的意图转换为存储在计算机中的具体信息。

对于类，我们说公共接口。在这里，公众（public）是使用类的程序，交互系统由类对象组成，而接口由编写类的人提供的方法组成。接口让程序员能够编写与类对象交互的代码，从而让程序能够使用类对象。例如，要计算string对象中包含多少个字符，您无需打开对象，而只需使用string类提供的size()方法。类设计禁止公共用户直接访问类，但公众可以使用方法size()。方法size()是用户和string类对象之间的公共接口的组成部分。通常，方法getline()是istream类的公共接口的组成部分，使用cin的程序不是直接与cin对象内部交互来读取一行输入，而是使用getline()。

如果希望更人性化，不要将使用类的程序视为公共用户，而将编写程序的人视为公共用户。然而，要使用某个类，必须了解其公共接口；要编写类，必须创建其公共接口。

为开发一个类并编写一个使用它的程序，需要完成多个步骤。这里将开发过程分成多个阶段．而不是一次性完成。通常，C++程序员将接口（类定义）放往头文件中，并将实现（类方法的代码）放在源代码文件中。这里采用这种典型做法。程序清单10.1是第一个阶段的代码，它是Stock类的类声明。这个文件按第9章介绍的那样，使用了#ifndef等来访问多次包含同一个文件。

为帮助识别类，本书遵循一种常见但不通用的约定——将类名首字母大写。您将发现，程序清单10.1看起来就像一个结构声明，只是还包括成员函数、公有部分和私有部分等内容。稍后将对该声明进行改进（所以不要将它用作模型），但先来看一看该定义的工作方式。

**程序清单10.1 stock00.h**

// stock00.h -- Stock类接口

// version 00

#ifndef STOCK00\_H\_

#define STOCK00\_H\_

#include <string>

class Stock // 类声明

{

private:

 std::string company;

 long shares;

 double share\_val;

 double total\_val;

 void set\_tot() { total\_val = shares \* share\_val; }

public:

 void acquire(const std::string & co, long n, double pr);

 void buy(long num, double price);

 void sell(long num, double price);

 void update(double price);

 void show();

}; // 注意结尾是分号

#endif

稍后将详细介绍类的细节，但先看一下更通用的特性。首先，C++关键字class指出这些代码定义了一个类设计（不同于在模板参数中，在这里，关键字class和typename不是同义词，不能使用typename代替class）。这种语法指出，Stock是这个新类的类型名。该声明让我们能够声明Stock类型的变量——称为对象或实例。每个对象都表示一支股票。例如，下面的声明创建两个Stock对象，它们分别名为sally和solly：

Stock sally;

Stock solly;

例如，sally对象可以表示Sally持有的某公司股票。

接下来，要存储的数据以类数据成员（如company和shares）的形式出现。例如，sally的company成员存储了公司名称，share成员存储了Sally持有的股票数量，share\_val成员存储了每股的价格，total\_val成员存储了股票总价格。同样，要执行的操作以类函数成员（方法，如sell()和update()）的形式出现。成员函数可以就地定义（如set\_tot()），也可以用原型表示（如其他成员函数）。其他成员函数的完整定义稍后将介绍，它们包含在实现文件中：但对于描述函数接口而言，原型足够了。将数据和方法组合成一个单元是类最吸引人的特性。有了这种设计，创建Stock对象时，将自动制定使用对象的规则。

istream和ostream类有成员函数，如get()和getline()，而Stock类声明中的函数原型说明了成员函数是如何建立的。例如，头文件iostream将getline()的原型放在istream类的声明中。

### 1．访问控制

关键字private和public也是新的，它们描述了对类成员的访问控制。使用类对象的程序都可以直接访问公有部分，但只能通过公有成员函数（或友元函数，参见第11章）来访问对象的私有成员。例如，要修改Stock类的shares成员，只能通过Stock的成员函数。因此，公有成员函数是程序和对象的私有成员之间的桥梁，提供了对象和程序之间的接口。防止程序直接访问数据被称为数据隐藏。C++还提供了第三个访问控制关键字protected，第13章介绍类继承时将讨论该关键字。

类设计尽可能将公有接口与实现细节分开。公有接口表示设计的抽象组件。将实现细节放在一起并将它们与抽象分开被称为封装。数据隐藏（将数据放往类的私有部分中）是一种封装，将实现的细节隐藏在私有部分中，就像Stock类对set\_tot()所做的那样，也是一种封装。封装的另一个例子是，将类函数定义和类声明放在不同的文件中。

**OOP和C++**

OOP是一种鳊程风格，从某种程度说，它用于任何一种语言中。当然，可以将OOP思想融合到常规的C语言程序中。例如，在第9章的一个示例（程序清单9.1、程序清单9.2、程序清单9.3）中，头文件中包含结构原型和操纵该结构的函数的原型，便是这样的例子。因此，main()函数只需定义这个结构类型的变量，并使用相关函数处理这些变量即可；main()不直接访问结构成员。实际上，该示例定义了一种抽象类型，它将存储格式和函数原型置于头文件中，对main()隐藏了实际的数据表示。然而，C++中包括了许多专门用来实现OOP方法的特性，因此它使程序员更进一步。首先，将数据表示和函数原型放在一个类声明中（而不是放在一个文件中），通过将所有内容放在一个类声明中，来使描述成为一个整体。其次，让数据表示成为私有，使得数据只能被授权的函数访问。在C语言的例子中，如果main()直接访问了结构成员，则违反了OOP的精神，但没有违反C语言的规则。然而，试图直接访问Stock对象的shares成员便违反了C++语言的规则，编译器将摘获这种错误。

数据隐藏不仅可以防止直接访问数据，还让开发者（类的用户）无需了解数据是如何被表示的。例如，show()成员将显示某支股票的总价格（还有其他内容），这个值可以存储在对象中（上述代码正是这样做的），也可以在需要时通过计算得到。从使用类的角度看，使用哪种方法没有什么区别。所需要知道的只是各种成员函数的功能；也就是说，需要知道成员函数接受什么样的参数以及返回什么类型的值。原则是将实现细节从接口设计中分离出来。如果以后找到了更好的、实现数据表示或成员函数细节的方法，可以对这些细节进行修改，而无需修改程序接口，这使程序维护起来更容易。

### 2．控制对成员的访问：公有还是私有

无论类成员是数据成员还是成员函数，都可以在类的公有部分或私有部分中声明它。但由于隐藏数据是OOP主要的目标之一，因此数据项通常放在私有部分，组成类接口的成员函数放在公有部分；否则，就无法从程序中调用这些函数。正如Stock声明所表明的，也可以把成员函数放在私有部分中。不能直接从程序中调用这种函数，但公有方法却可以使用它们。通常，程序员使用私有成员函数来处理不属于公有接口的实现细节。

不必在类声明中使用关键字private，因为这是类对象的默认访问控制：

class World

{

 float mass; // 默认为私有

 char name[20]; // 默认为私有

public：

 void tellall(void);

 …

);

然而，为强调数据隐藏的概念，本书显式地使用了private。

**类和结构**

类描述看上去很像是包含成员函数以及public和private可见性标签的结构声明。实际上，C++对结构进行了扩展，使之具有与类相同的特性。它们之间唯一的区别是，结构的默认访问类型是public，而类为private。C++程序员通常使用类来实现类描述，而把结构限制为只表示纯粹的数据对象（常被称为普通老式数据(POD，Plain Old Data)结构）。

## 10.2.3 实现类成员函数

还需要创建类描述的第二部分：为那些由类声明中的原型表示的成员函数提供代码。成员函数定义与常规函数定义非常相似，它们有函数头和函数体，也可以有返回类型和参数。但是它们还有两个特殊的特征：

* 定义成员函数时，使用作用域解析运算符（::）来标识函数所属的类；
* 类方法可以访问类的private组件。

首先，成员函数的函数头使用作用域运算符解析（::）来指出函数所属的类。例如，update()成员函数的函数头如下：

void Stock::update(double price)

这种表示法意味着我们定义的update()函数是Stock类的成员。这不仅将update()标识为成员函数，还意味着我们可以将另一个类的成员函数也命名为update()。例如，Buffoon()类的update()函数的函数头如下：

void Buffoon::update()

因此，作用域解析运算符确定了方法定义对应的类的身份。我们说，标识符update()具有类作用域（class scope）。Stock类的其他成员函数不必使用作用域解析运算符，就可以使用update()方法，这是因为它们属于同一个类，因此update()是可见的。然而，在类声明和方法定义之外使用update()时，需要采取特殊的措施，稍后将作介绍。

类方法的完整名称中包括类名。我们说，Stock::update()是函数的限定名（qualified name）；而简单的update()是全名的缩写（非限定名，unqualified name），它只能在类作用域中使用。

方法的第二个特点是，方法可以访问类的私有成员。例如，show()方法可以使用这样的代码：

std::cout << “Company:” << campany

 << “Shares:” << shares << endl

 << “Share Price: $” << share\_val

 << “Total Worth: $” << total\_val << endl;

其中，company、shares等都是Stock类的私有数据成员，如果试图使用非成员函数访问这些数据成员，编译器禁止这样做（但第11章中将介绍的友元函数例外）。

了解这两点后，就可以实现类方法了，如程序清单10.2所示．这里将它们放在了一个独立的实现文件中，因此需要包含头文件stock00.h，让编译器能够访问类定义。为让您获得更多有关名称空间的经验，在有些方法中使用了限定符std::，在其他方法中则使用了using声明。

**程序清单10.2 stock00.cpp**

// stock00.cpp -- Stock类实现

// version 00

#include <iostream>

#include "stock00.h"

void Stock::acquire(const std::string & co, long n, double pr)

{

 company = co;

 if (n < 0)

 {

 std::cout << "Number of shares can't be negative; "

 << company << " shares set to 0.\n";

 shares = 0;

 }

 else

 shares = n;

 share\_val = pr;

 set\_tot();

}

void Stock::buy(long num, double price)

{

 if (num < 0)

 {

 std::cout << "Number of shares purchased can't be negative. "

 << "Transaction is aborted.\n";

 }

 else

 {

 shares += num;

 share\_val = price;

 set\_tot();

 }

}

void Stock::sell(long num, double price)

{

 using std::cout;

 if (num < 0)

 {

 cout << "Number of shares sold can't be negative. "

 << "Transaction is aborted.\n";

 }

 else if (num > shares)

 {

 cout << "You can't sell more than you have! "

 << "Transaction is aborted.\n";

 }

 else

 {

 shares -= num;

 share\_val = price;

 set\_tot();

 }

}

void Stock::update(double price)

{

 share\_val = price;

 set\_tot();

}

void Stock::show()

{

 std::cout << "Company: " << company

 << " Shares: " << shares << '\n'

 << " Share Price: $" << share\_val

 << " Total Worth: $" << total\_val << '\n';

}

### 1．成员函数说明

acquire()函数管理对某个公司股票的首次购买，而buy()和sell()管理增加或减少持有的股票。方法buy()和sell()确保买入或卖出的股数不为负。另外；如果用户试图卖出超过他持有的股票数量，则sell()函数将结束这次交易。这种使数据私有并限于对公有函数访问的技术允许我们能够控制数据如'何被使用；在这个例子中，它允许我们加入这些安全防护措施，避免不适当的交易。

4个成员函数设置或重新设置了total\_val成员值。这个类并非将计算代码编写4次，而是让每个函数都调用set\_tot()函数。由于set\_tot()只是实现代码的一种方式，而不是公有接口的组成部分，因此这个类将其声明为私有成员函数（即编写这个类的人可以使用它，但编写代码来使用这个类的人不能使用）。如果计算代码很长，则这种方法还可以省去许多输入代码的工作，并可节省空间。然而，这种方法的主要价值在于，通过使用函数调用，而不是每次重新输入计算代码，可以确保执行的计算完全相同。另外，如果必须修订计算代码（在这个例子，这种可能性不大），则只需在一个地方进行修改即可。

### 2．内联方法

其定义位于类声明中的函数都将自动成为内联函数，因此Stock::set\_tot()是一个内联函数。类声明常将短小的成员函数作为内联函数，set\_tot()符合这样的要求。

如果愿意，也可以在类声明之外定义成员函数，并使其成为内联函数。为此，只需在类实现部分中定义函数时使用inline限定符即可：

class Stock

{

private:

 void set\_tot(); // definition kept separate

public:

 …

};

inline void Stock::set\_tot() // use inline in definition

{

 total\_val = shares \* share\_val;

}

内联函数的特殊规则要求在每个使用它们的文件中都对其进行定义。确保内联定义对多文件程序中的所有文件都可用的、最简便的方法是：将内联定义放在定义类的头文件中。

顺便说一句，根据改写规则（rewrite rule），在类声明中定义方法等同于用原型替换方法定义，然后在类声明的后面将定义改写为内联函数。也就是说，程序清单10.1中set\_tot()的内联定义与上述代码（定义紧跟在类声明之后）是等价的。

### 3．方法使用哪个对象

下面介绍使用对象时最重要的一个方面：如何将类方法应用于对象。下面的代码使用了一个对象的shares成员：

shares += num;

是哪个对象呢？问得好！要回答这个问题，首先来看看如何创建对象。最简单的方式是声明类变量：

Stock kate, joe;

这将创建两个Stock类对象，一个为kate，另一个为joe。

接下来，看看如何使用对象的成员函数。和使用结构成员一样，通过成员运算符：

kate.show(); // kate对象调用成员函数

joe.show(); // joe对象调用成员函数

第1条语句调用kate对象的show()成员。这意味着show()方法将把shares解释为kate.shares，将share\_val解释为kate.share\_val。同样，函数调用joe.show()使show()方法将shares和share\_val分别解释为joe.share和joe.share\_val。

**注意**：调用成员函数时，它将使用被用来调用它的对象的数据成员。

同样，函数调用kate.sell()在调用set\_tot()函数时，相当于调用kate.set\_tot()，这样该函数将使用kate对象的数据。

所创建的每个新对象都有自己的存储空间，用于存储其内部变量和类成员；但同一个类的所有对象共享同一组类方法，即每种方法只有一个副本。例如，假设kate和joe都是Stock对象，则kate.shares将占据一个内存块，而joe.shares占用另一个内存块，但kate.show()和joe.show()都调用同一个方法，也就是说，它们将执行同一个代码块，只是将这些代码用于不同的数据。在OOP中，调用成员函数被称为发送消息，因此将同样的消息发送给两个不同的对象将调用同一个方法，但该方法被用于两个不同的对象。

## 10.2.4 使用类

知道如何定义类及其方法后，来创建一个程序，它创建并使用类对象。C++的目标是使得使用类与使用基本的内置类型（如int和char）尽可能相同。要创建类对象，可以声明类变量，也可以使用new为类对象分配存储空间。可以将对象作为函数的参数和返回值，也可以将一个对象赋给另一个。C++提供了一些工具，可用于初始化对象、让cin和cout识别对象，甚至在相似的类对象之间进行自动类型转换。虽然要做到这些工作还需要一段时间，但可以先从比较简单的属性着手。实际上，您已经知道如何声明类对象和调用成员函数。程序清单10.3提供了一个使用上述接口和实现文件的程序，它创建了一个名为fluffy\_the\_cat的Stock对象。该程序非常简单，但确实测试了这个类的特性。要编译该程序，可使用用于多文件程序的方法，这在第1章和第9章介绍过。具体地说，将其与stock00.cpp一起编译，并确保stock00.h位于当前文件夹中。

**程序清单10.3 usestock.cpp**

// usestok0.cpp -- the client program

// compile with stock.cpp

#include <iostream>

#include "stock00.h"

int main()

{

 Stock fluffy\_the\_cat;

 fluffy\_the\_cat.acquire("NanoSmart", 20, 12.50);

 fluffy\_the\_cat.show();

 fluffy\_the\_cat.buy(15, 18.125);

 fluffy\_the\_cat.show();

 fluffy\_the\_cat.sell(400, 20.00);

 fluffy\_the\_cat.show();

 fluffy\_the\_cat.buy(300000,40.125);

 fluffy\_the\_cat.show();

 fluffy\_the\_cat.sell(300000,0.125);

 fluffy\_the\_cat.show();

 // std::cin.get();

 return 0;

}

下面是该程序的输出：

Company: NanoSmart Shares: 20

 Share Price: $12.5 Total Worth: $250

Company: NanoSmart Sharea: 35

 Share Price: $18.125 Total Worth: $614.375

You can't sell more than you have! Transaction is aborted

Company: NanoSmart Shares: 35

 Share Price: $18.125 Total Worth: $634.375

Company: NanoSmart Sharea: 300035

 Share Price: $40.125 Total Worth: $1.20389e+007

Company: NanoSmart Shares: 35

 Share Price: $0.125 Total Worth: $4.375

注意，main()只是用来测试Stock类的设计。当Stock类的运行情况与预期的相同后，便可以在其他程序中将Stock类作为用户定义的类型使用。要使用新类型，最关键的是要了解成员函数的功能，而不必考虑其实现细节。请参阅后面的旁注“客户/服务器模型”。

**客户，服务器模型**

OOP程序员常依照客户/服务器模型来讨论程序设计。在这个概念中，客户是使用类的程序。类声明（包括类方法）构成了服务器，它是程序可以使用的资源。客户只能通过以公有方式定义的接口使用服务器，这意味着客户（客户程序员）唯一的责任是了解该接口。服务器（服务器设计人员）的责任是确保服务器根据该接口可靠并准确地执行。服务器设计人员只能修改类设计的实现细节，而不能修改接口。这样程序员独立地对客户和服务器进行改进，对服务器的修改不会对客户的行为造成意外的影响。

## 10.2.5 修改实现

在前面的程序输出中，可能有一个方面让您恼火一一数字的格式不一致。现在可以改进实现，但保持接口不变。ostream类包含一些可用于控制格式的成员函数；这里不做太详细的探索，只需像在程序清单8.8那样使用方法setf()，便可避免科学计数法：

std::cout.setf(std::ios\_base::fixed , std::ios\_base::floatfield);

这设置了cout对象的一个标记，命令cout使用用定点表示法。同样，下面的语句导致cout在使用定点表示法时，显示三位小数：

std::cout.precision(3);

第17章将介绍这方面的更多细节。

可在方法show()中使用这些工具来控制格式，但还有一点需要考虑。修改方法的实现时，不应影响客户程序的其他部分。上述格式修改将一直有效，直到您再次修改，因此它们可能影响客户程序中的后续输出。因此，show()应重置格式信息，使其恢复到自己被调用前的状态。为此，可以像程序清单8.8那样，使用返回的值：

std::streamsize prec =

 std::cout.precision(3) // 保存前一个值的精度

…

std: :cout.precision(prec) ; // 重置旧的值

// 保存初始的格式标志

std::ios\_base::fmtflags orig = std::cout.setf(std::ios\_base::f ixed);

…

// 重置为已保存的格式

std::cout.setf (orig,std::ios\_base::floatfield);

您可能还记得，fmtflags是在ios\_base类中定义的一种类型，而ios\_base类又是在名称空间std中定义的，因此orig的类型名非常长。其次，orig存储了所有的标记，而重置语句使用这些信息来重置floatfield，而floatfield包含定点表示法标记和科学表示法标记。第三，请不要过多考虑这里的细节。这里的要旨是，将修改限定在实现文件中，以免影响程序的其他方面。

根据上面的介绍，可在实现文件中将方法show()的定义修改成如下所示：

void Stock：：show()

{

 using std:：cout;

 using std:：ios\_base;

 // 将格式设置为#．###

 ios\_base::fmtflags orig =

 cout.setf(ios\_base::fixed,ios\_base::floatfield);

 std::streamsize prec = cout.precision(3);

 cout << “Company:" << company

 << “ Shares:" << shares << '\n';

 cout << “Share Price: $" << share\_val;

 // 将格式设置为#.##

 cout.precrision(2);

 cout << “ Total Worth: $" << total\_val << '\n’;

 // 恢复初始格式

 cout.setf(orig, ios\_base::floatfield);

 cout.precision(prec);

}

完成上述修改后（保留头文件和客户文件不变），可重新编译该程序。该程序的输出将类似于下面这样：

Company: NanoSmart Shares: 20

 Share Price： $12. 500 Total Worth: $250.00

Company: NanoSmart Shares: 35

 Share Price: $18.125 Total Worth: $634.38

You\_can’t sell more than you have! Transaction is aborted.

Company: NanoSmart Shares: 35

 Share Price: $l8.125 Total Worth: $634.38

Company: NanoSmart Shares: 300035

 Share Price: $40.125 Total Worth: $ 12038904.38

Company: NanoSmart Shares: 35

 Share Price: $0.125 Total Worth: $4.38

## 10.2.6 小结

指定类设计的第一步是提供类声明。类声明类似结构声明，可以包括数据成员和函数成员。声明有私有部分，在其中声明的成员只能通过成员函数进行访问；声明还具有公有部分，在其中声明的成员可被使用类对象的程序直接访问。通常，数据成员被放在私有部分中，成员函数被放在公有部分中，因此典型的类声明的格式如下：

class className

{

private:

 data member declarations

public：

 member function prototypes

}

公有部分的内容构成了设计的抽象部分——公有接口。将数据封装到私有部分中可以保护数据的完整性，这被称为数据隐藏。因此，C++通过类使得实现抽象、数据隐藏和封装等OOP特性很容易。

指定类设计的第二步是实现类成员函数。可以在类声明中提供完整的函数定义，而不是函数原型，但是通常的做法是单独提供函数定义（除非函数很小）。在这种情况下，需要使用作用域解析运算符来指出成员函数属于哪个类。例如，假设Bozo有一个名为Retort()的成员函数，该函数返回char指针，则其函数头如下所示：

char \* Bozo::Retort()

换句话来说，Retort( )不仅是一个char \* 类型的函数，而且是一个属于Bozo类的char \*函数。该函数的全名（或限定名）为Bozo::Retort()。而名称Retort()是限定名的缩写，只能在某些特定的环境中使用，如类方法的代码中。

另一种描述这种情况的方式是，名称Retort的作用域为整个类，因此在类声明和类方法之外使用该名称时，需要使用作用域解析运算符进行限定。

要创建对象（类的实例），只需将类名视为类型名即可：

Bozo bozetta;

这样做是可行的，因为类是用户定义的类型。

类成员函数（方法）可通过类对象来调用。为此，需要使用成员运算符句点：

cout << bozetta.Retort();

这将调用Retort()成员函数，每当其中的代码引用某个数据成员时，该函数都将使用bozetta对象中相应成员的值。