张国锋　编译

面向对象de程序设计

和C++语言

北京科海培训中心

面向对象的程序设计和 C++教程

张国锋　编译

科 海 培 训 中 心

一九九一年一月

目　　录

第一章　面向对象的程序设计

在本章中介绍新的、使人激动的与面向对象的程序设计方法有关的概念和结论。读者第一次可以很快地浏览本章的内容，在通读完本书之后，回过头来仔细推敲在这章讨论的主题。

面向对象的程序设计对于软件系统的设计和实现是一种相对较新的方法，它的主要目标是通过增强软件的可扩充性和可重复使用性，来改善程序员的软件生产活动以及控制软件维护的复杂性和费用。当使用面向对象的程序设计方法时，软件开发的设计阶段与实现阶段被紧密地连接在一起。这本书的目标就是探索在软件开发中使用这个新的令人鼓舞的新方法，并且介绍一个新的和重要的非常适合于说明面向对象的程序设计方法的语言——C++语言。

面向对象的程序设计的核心围绕着几个重要的概念：抽象数据类型、类、继承等级（子类）和多态性。本章首先介绍这些基本概念，然后概述了在进行面向对象的程序设计中的一些方法和应注意的一些问题，这些内容对读者进行面向对象的程序设计实践是会大有帮助的。

1.1　抽象数据类型

在面向对象的程序设计中，我们将在一个单个实体内的数据和操作这些数据的过程组合在一起所形成的描述称作一个对象。在一个对象中的数据由域组成，域可以引用其它的对象；与这个对象有关的操作刻划了这个对象的行为。每个对象有一个类型，对象被称为这个类型的一个实例。类型提供了一种分类相似的对象并归纳这些对象的共同特征的一种手段。一个抽象数据类型是将一个类型和与这个类型有关的操作集合封装在一起的一个数据模型。

面向对象的程序设计方法和传统的面向过程的程序设计方法不同。在传统语言中，程序是由传递参数的过程或函数的集合组成，每个过程处理它的参数，有时更新它们，并有可能返回一个值，因此以执行单元（即过程）为中心。在面向对象的语言中，世界被看作独立的对象的集合，相互之间通过过程（通常称作消息）进行通信。对象是主动的，而过程（消息）是被动的，和它们的参数一起被从一个对象传递到另一个对象。一个对象根据提交给它的请求（即过程或又称作方法）并基于这些过程（方法）进行行动，因此以数据（即对象）为中心。面向对象的程序设计方法鼓励程序员集中于要被处理的数据，而不是进行这种处理的代码。因为抽象数据类型普遍地被应用于模拟应用领域内的实体，它们为程序的模块化提供了一个自然的基础。在面向过程的程序设计中，程序员被强迫基于过程组织程序模块，这导致程序的结构与应用领域中的差异很大。因为抽象数据类型模拟了应用领域中的实体，在面向对象程序中的结构可以非常相似于应用领域中的结构。

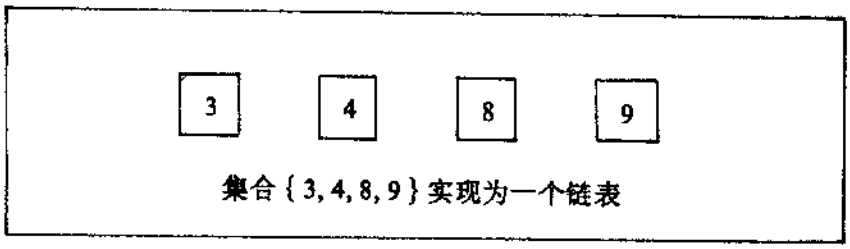
支持抽象数据类型的语言必须能够支持信息隐藏，也就是说，一个对象只能通过为这个抽象数据类型定义的接口来访问和修改；用于实现这个抽象数据类型和它的操作的内部实现细节、数据结构和存贮表示对访问和处理这个对象的当事人是不可见的。这表明对象

具有公有接口，以及这些接口的私有表示和实现．这种抽象类型的基本观点很简单．考虑在一个传统程序设计语言中的一个基本类型，例如 C 中的整型，语言提供了有限个数的操作，例如+、＊、－表示整数的加、乘和减操作，还有余和商操作等．这些操作有严格定义的结合性、传递性和分配性，它们完整地定义了整数的行为．用户简单地使用这些操作处理整数，一个整数的内部位串表示（例如使用十六位二进制补码）是完全隐藏的．处理整数的程序可以很容易地移植到对整数使用完全不同的内部表示的系统中并被正确的编译和执行．

使用抽象数据类型的语言将这种方法扩展到程序中的每个对象或数据上．抽象机制迫使对具有用户定义类型的对象的访问和更新只能通过为这个特定类型定义的操作接口来进行，这被称作封装．和 C 这样的传统语言作一比较，例如一个整数集合可以定义为：

```
struct SetOfIntegers {
        int Element ;
        struct SetOfIntegers * Next ;
};
```

在这个结构类型的作用域内的任何函数或过程都可以处理它．换句话说，我们可以作为一个链表来遍历一个 SetOfIntegers 的对象，并且我们可以直接访问和更新其中的一个元素．这示于图 1.1 中，图中函数 Find 遍历链接表以看一下给定的元素是否在链表中．如果这个集合的表示发生了变化（例如实现为一个整数数组），函数 Find 和所有其它的依赖于这个对象的表示（数据结构）的函数必须作修改．




```
int Find( int x, struct SetofInteger * L )
 / * Return 1 if and only if x is in L, 0 otherwise * /
 {
    while (L != 0 )
    if ( x = = L-> Element )
       return 1 ;
    else
       L = L-> Next ;
    return 0
 }
```

图 1.1　链表表示和遍历

使用抽象数据类型，为这个整数集合定义一系列操作，对这个集合的访问和更新只能通过由这些操作提供的接口来进行．在内部，一个整数集合可以以链表的形式来表示，或

表示为一个整数数组，或使用其它的表示形式，虽然内部表示发生变化，但接口不发生变化，而在使用这个整数集合时都通过接口来进行，那么这个集合对使用者就没有什么不同，使用者也不必因为内部表示发生变化而更改他的程序。因此一个函数，例如 Find，如果依赖于集合的内部表示，那么它或是抽象数据类型的一个接口，或通过使用其它的接口来执行特殊的任务。

抽象数据类型是面向对象程序设计中组织程序的主要原则。一个类型结构设计完善的程序将减少并局部化类型之间的依赖，因此增强了软件的维护性。

1.2 类、对象和封装

在大多数面向对象的程序设计语言中，用于定义抽象数据类型的语言结构是类(Class，它将这个抽象数据类型的实现细节（即数据结构的表示）和操作封装在一起。通常这些实现细节仅可被在这个类中所定义的操作访问，我们称它们为私有的；如果可以在这个类的外面访问到，我们就称其为是公有的。在这个类型上定义的操作通常也被划分成公有的或私有的。在面向对象的语言中，为一个类定义的所有操作称作方法（Method），这些方法类似于非面向对象语言中的过程和函数。如果一个类的公有操作可以用在许多应用领域，那么这个类就可作为可重复利用的软件组件（Software component）的基础。

一个类定义的私有部分通常定义基本数据类型的数据结构，它也定义只能在这个类的内部可以访问的方法，这些方法经常用作支持公有方法的实现。一个类的私有部分有时也可包含其它类的对象，这个对象只能在这个类的内部进行处理。一个类的公有部分通常定义方法，这些方法是使用这个类的接口，它们形成这个类在许多应用领域内可重复使用的基础。

一个对象是一个声明为具有某一特定类的变量，这样的一个对象通过包含在类定义中定义的所有域（公有或私有的）的拷贝来封装状态，借助于访问在这个类定义中定义的一个或多个方法，可以对这个对象实施各种操作。访问一个方法的过程称作向这个对象发送一个消息（Message)。正如在非面向对象的语言中访问函数和过程一样，一个消息也带有一些参数，对一个方法的访问（向一个对象发送一个消息）的典型操作是修改存贮于特定对象中的数据。

封装（Encapsulation）的定义为:

(1)一个清楚的边界，所有的对象的内部软件的范围被限定在这个边界内。

(2)一个接口，这个接口描述这个对象和其它的对象之间相互的作用。

(3)受保护的内部实现,这个实现给出了由软件对象提供的功能的细节，实现细节不能在定义这个对象的类的外面访问。

封装的概念和类说明有关，但它同样提供如何将一个问题解的各个组件组装在一起的求精过程。封装的基本单位是对象，这个对象的性质由它的类说明来描述，这些性质被具有同样类的其它对象共享。对象用作封装说法比说一个类表示封装的说法更严格。借助对封装这种定义，一个类的每一个实例在一个问题求解中是一个单独的封装，或称作组件。

对象用于封装的概念可以和集成电路芯片作一类比。一块集成电路芯片由陶瓷封装起来，其内部电路是不可见的，也是使用者不关心的。芯片的使用者只关心芯片引脚的个数、引脚的电气参数以及引脚提供的功能，通过这些引脚，硬件工程师对这个芯片有了全

面的了解。硬件工程师将不同的芯片引脚连在一起，就可以组装一个具有一定功能的产品。软件工程师通过使用类也努力达到这个目的。第四章集中讨论 C++语言中的类和封装。

因为程序员可以建立一个新的类型，赋与特殊的性质，这个类型的行为在类定义中刻划，由这些新类型产生的对象在被处理的过程中十分类似于程序设计语言中提供的预定义类型，因此面向对象的语言被认为可扩充的。

图 1.2 给出的例子用于说明类、对象和封装。

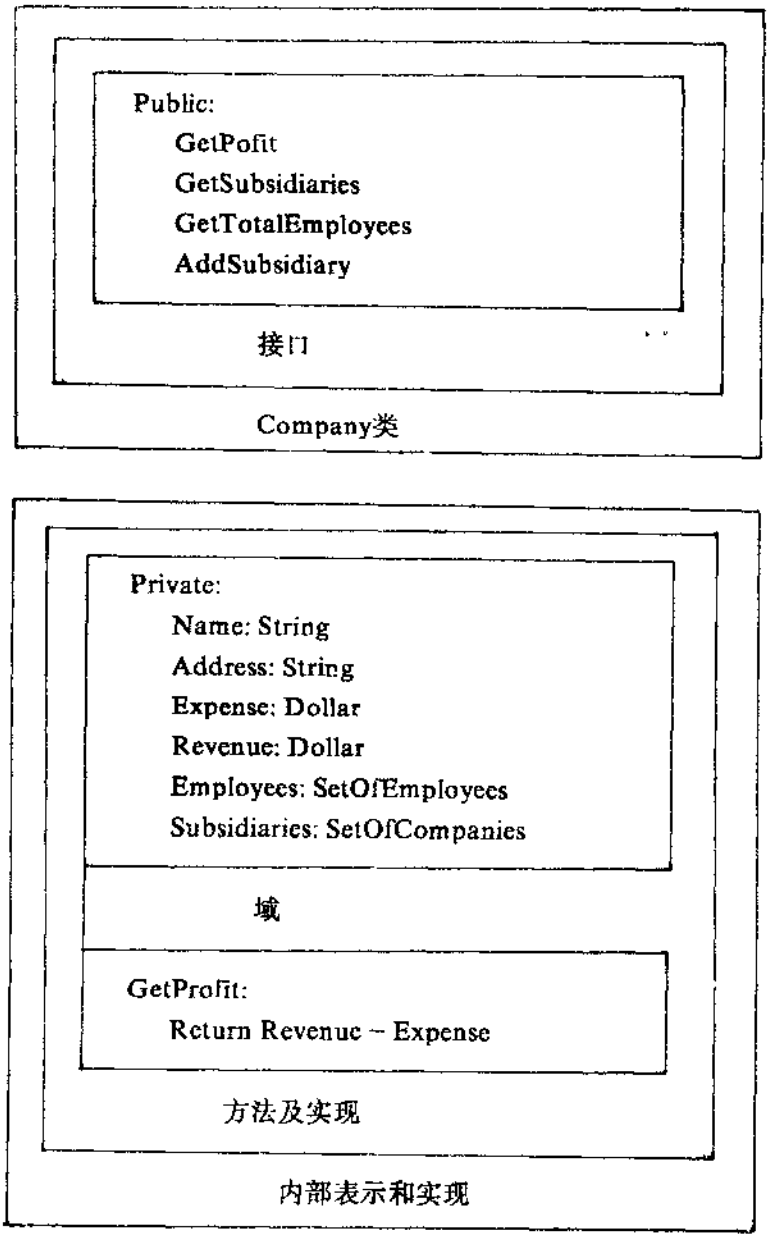


图 1.2　公司表示为一个类

每个类变量即对象表示这个类的一个实例（Instance）。如果几个对象被定义具有同样的类，它们常表示包含有不同值的一个集合。例如IBM、SUN、DEC等都是Company类的实例，它们具有不同的名字、地址等状态，它们只有通过给定的组成它们的接口的操作集合才能被处理。

一个类定义（致少）包括以下方面：

(1)类名

(2)用于处理这个类的实例的接口

(3)内部表示

(4)操作的内部实现

我们假定存在一个由这些操作使用的隐含的目标对象，操作可以带有参数。直观上讲，目标对象是消息即操作的接受者。换句话说，一个操作每次调用适用于一个且仅有一个这个类的实例。这个实例就是目标对象。对我们这个例子，目标对象是这些公司中的一个：IBM, SUN 或 DEC。下面给出类 Company 的操作的几个描述：

(1)名字：GetProfit

结果：返回当前财政年度公司的利润。

(2)名字：GetSubsidiaries

结果：返回目标父公司所有的子公司的集合。

(3)名字：AddSubsidiary

参数：类 Company 的一个实例，将成为目标父公司的一个新的子公司。

副作用：在当前的目标父公司的子公司集合中插入一个新的子公司。

类 Company 的其它元素包括：

(4)内部表示：Name, Address, Expense, Revenue, Employees 等。

(5)内部实现：在后面我们会看到，操作通过访问和更新内部表示的值来实现的。

因此一个类包含对象状态的描述（内部表示），以及实现这个类的操作的代码。重要的是要注意，表示类的实例的内部状态的域的值是完全不同的，属于特定的对象。例如，IBM 的内部表示由它特定的 Expense, Revenue, Employees, Subsidiaries 等组成。因此对一个类的每个实例，必须为其分配存贮以保存内部表示。

然而所有的实例共享实现操作的代码。因此，只有一组代码用于实现 GetProfit,GettotalEmployees, AddSubsidiary 以及其它的操作。正如我们前面说过的，这些操作与一个目标对象一起被访问，并带有参数。面向对象的系统知道如何将相应的操作应用于目标对象，而不破坏它们的内部状态。这非常类似于传统程序设计语言中的过程调用。

1.2.1 域

一个对象是类的一个实例，对象的状态用域来表示。域是一种一般性的说法，有时又常称作实例变量（在 Smalltalk 中）或数据成员（在 C++中）。在 C++这样的面向对象的语言中，除为域指定名字外，还必须指定它所具有的类型。图 1.2 中给出的是一种一般的形式，冒号右边的类名说明域是这个类的一个实例。例如，对任何 Company 的实例，例如 IBM 或 DEC，Expense 是类 Dollar 一个实例，Employees 是 SetOfEmployees 的一个实例。

一个类的实例通过显式地或隐式地调用 new 运算符来创建，因此类可以被认为是包含有它的输出的描述的一个工厂，并响应请求制造出下一个产品实例，即对象．域的初始化可以在对象创建时进行，或在以后通过调用处理或更新这个域的状态的操作来进行．

1.2.2 方法和消息

一个类也定义了表明它的实例的行为的操作．访问一个操作包括:

(1)目标对象

(2)操作的名字

(3)操作的参数

(4)调用实现这个操作的代码，将形参和实参相结合

假定变量 GM 表示某一个公司，它是类 Company 的一个实例．另外，假定 Hughes 也是 Company 的一个实例，并且我们想将它作为 GM 的一个子公司．前面我们提到有一个操作 AddSubsidiaries 来执行这个操作．这则消息在 C++中将表示成:

```
GM.AddSubsidiaries(Hughes)
```

为一个类定义的所有操作称作方法．为将操作应用于一个目标对象，需要向目标对象发送一个消息，消息包括:

(1)选择器。

(2)一个或几个参数。

一个消息的选择器从和这个目标对象有关的所有的方法中选择一个相应的方法．一个选择器其实就是一个方法的名字，将消息和方法联编在一起．当然，这和传统程序设计语言中调用一个过程所发生的情形相同，也许一个最重要的不同点是，这种联编可能在运行时间依据目标对象的类型来完成，这称作动态联编（Dynamic binding），是面向对象的程序设计语言所具有的另一个重要特征，我们将在后面介绍。

1.3 类型等级

在这节中，我们介绍类的等级概念（Hierarchy），在这种等级结构中，某些类附属于其它类，并被称作子类（Subclass），或者在 C++中称作派生类（Derived class）。

在类的等级结构中，较低层的类通常表示特殊概念，而较高层的类通常表示一般化的概念。等级概念暗含的关系是继承（Inheritance），一个子类可以共享（即继承）它的父类（Parent class）的各种特征，子类反过来又将它自己的或继承来的特征传递到它的子类中．通过继承，我们可以在一个较少具有特殊性的已存在类的基础之上建立新类，而不必重新开始设计每件东西，新类可以从已存在的类中继承行为（操作、方法）和表示(域)．第五章集中讨论 C++语言中的继承和子类，

借助允许建立子类，可以加快对问题的解决，降低软件开发的复杂性和费用．

这里举出的例子用于说明继承．盈利公司和非盈利组织有大量的共同之处，两类公司都包含有公司名称、公司地址、人员等的信息．非盈利组织有更具体的信息，例如政府许可、赞助商等；同样，盈利公司也有一些特殊内容，例如纳税减免等．

因此，为获得非盈利组织和盈利公司的共同信息和行为，我们可以建立一个 Compa-

ny 类，并允许类 CommercialCompany 或 Non__ProfitOrganization 从它那儿继承。CommercialCompany 或 Non__ProfitOrgination 的每个实例也是 Company 的一个实例。在图 1.3 中说明了继承等级。但注意有的 Company 的实例既不是 CommercialCommpany 的实例也不是 Non__profitOrgination。例如我们可以有一个公司由其它几个盈利公司拥有，但不是它们中任何一个公司的子公司，也不产生盈利；并且也不是非盈利组织。

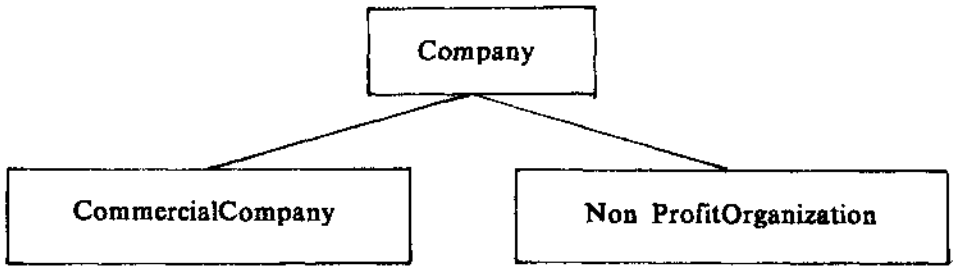


图 1.3 一个公司的类等级

CommercialCompany 和 Non__ProfitOrganization 被称为 Company 的子类，Company 被称为 CommercialCompany 和 Non__ProfitOrganization 的父类。子类和父类的关系是可传递的。这表明，如果 X 是 Y 的一个子类（父类），并且 Y 是 Z 的一个子类（父类），则 X 也是 Z 的一个子类（父类）。例如如果 SemiconductorFirm 是 CommercialCompany 的一个子类，则由传递性，它也是 Company 的一个子类。这表明它继承了 Company 和 CommercialCompany 的行为和表示。当然，除此之外，它从 CommercialCompany 继承了特定于 CommercialCompany 的所有行为和表示，这些行为和表示为 Company 的实例所不具有的。

从这个例子我们可以看出继承有两个方面：

(1)结构的：这表明一个类，例如 Non__ProfitOrganization，它是 Company 的一个子类，的实例，将从 Company 继承域，例如 Name, Address 等的值。

(2)行为的：类 Company 方法，例如 AddSubsidiary，EvaluateBudge 等，被它的子类 Non__ProfitOrganization 和 CommercialCompay 继承。这说明我们可以使用选择器 AddSubsidiary 向 Non__ProfitOrganization 的一个实例 NPO 发送消息，并以 NPO 作为目标对象执行 Company 类中的方法 AddSubsidiary。

从模拟的观点来看，继承是一种非常自然和有力的组织信息的方法。

1.3.1 继承域

一个对象的类通过定义对象的域描述了对象的内部结构。一个子类的实例必须和它的父类的实例一样保存有同样类型的信息，子类可以继承父类的域，也可以定义自己所特有的域。

由此可见，一个类有两种类型的使用方法：

(1)建立这个类的实例，并通过与这个类有关的方法处理这个实例——这称作实例化使用。

(2)这个类的子类从这个类中继承方法（行为）和表示——这称作继承使用。

从封装的角度来说，域对这个类的实例的使用者应被隐藏起来，如果允许子类不受限制地使用父类中的域，它将破坏封装，并会出现一些问题。回到前面的例子中，假定 Company 的域 Subsidiaries 被实现为一个 Company 数组，Non__ProfitOrganization 和 CommercialCompany 是 Company 的一个子类。如果允许子类直接访问父类的域，Non_ _ProfitOriganization 或 CommercialCompany 中定义的任何方法可以直接访问和更新这个数组变量。但是如果一个使用者只是建立 Company 的一个实例，他只能通过间接地使用选择器 AddSubsidiary 或 GetSubsidiaries 等方法（这些方法处理 Subsidiaries 域）来访问和更新域 Subsidiaries，而子类却可以直接访问和更新 Subsidiaries。因此使域在子类中可见破坏了一致性和封装。当通过修改父类的域来修改父类的实现时，使用这些域的所有子类必须被修改。例如如果 Subsidiaries 的实现被改成一个 Company 链接表，则在 Company 的子类中直接访问或更新 Subsidiaries 的所有方法必须被修改；而如果接口没有改变，则 Company 类的实例化使用不受影响，但继承使用却受到影响。

但是如果完全限制子类中的方法不能直接使用父类中的域，只能通过父类中的方法间接使用父类中的域，又可能失去有效性。将有效性和灵活性结合在一起的比较好的方法是在类中支持 public（公有的），private（私有的）和 protected（保护的，又称子类可见的），实现者用它们定义所需的保护。这三种选项被定义为：

(1)公有的：如果一个域或一个方法被定义为公有的，这个类的任何使用者可以直接访问、处理或调用它。

(2)私有的：如果一个域或一个方法被定义为私有的，这个类的任何使用者都不能直接访问、处理或调有它；只有这个类的方法可以访问、处理或调用它。

(3)保护的：如果一个域或一个方法被说明为保护的，则它只能被它的父类和子类的方法直接访问、处理或调用；父类或子类的实例化使用者不能直接访问、处理或调用它。

1.3.2 继承方法

在一个继承等级中，为一个类定义的方法被它的子类所继承，因此继承的方法是处理这个子类的实例的接口的一部分。

例如，我们可能有一个类 Window，BorderedWindow 和 TextWindow 是它的子类。在 Window 中定义的方法 MoveWindow，ResizeWindow 和 RotateWindow 可以被 TextWindow 和 BorderedWindow 的实例使用。子类也可以定义自己的方法，例如 TextWindow 子类可以定义用于编辑和修改窗口中文本串的字形、字符大小的方法。

另外，子类可以覆盖一个继承的方法。例如，在 Company 类等级中，CommercialCompany 和 Non__ ProfitOrganization 作为 Company 的子类，假定 ForeignCommercial 是 CommercaiCompany 的一个子类。由于不同的纳锐规定和限制条件，用于计算一个国外公司的利润的公式和计算本地公司的利润的公式是不同的。因此为 ForeignCommercial 定义另一个 EvaluateProfit 方法，它覆盖了 CommercialCompany 中具有相同名字的方法。

有时在一个子类的方法中调用一个父类中被覆盖的方法是有用的。例如，在实现用于 ForeignCommercial 的 EvaluateProfit 中，我们可能需要调用在类 CommercialCompany 中实现的 EvaluateProfit 方法。调用这个方法的一个明显的方法是使用类名限制这个方法

的名字．例如，如果这个方法是 M，我们想调用它在类 C 中的实现，我们简单地使用选择器 C::M，即用类名作为方法的修饰符．注意具有相同名字的方法可以出现在一个类等级的许多地方，一个被修饰的方法名其实现或是在修饰符所指定的类中定义的，或在修饰符所指出的类名的最直接的父类或祖先类中定义的．当然使用这个策略所需的唯一限制是类名必须唯一，这也是合理的．

1.3.3 多重继承

在许多情况下，从多于一个类中继承是很方便的．例如既是一个外国公司又是一个半导体公司的一个商业公司具有属于作为一个外国公司以及作为一个半导体公司所具有的特征（域和方法）．使用多重继承，我们可以将几个已存在的类组合在一起形成一个类。

作为一个简单的例子．考虑由父类 Person、它的子类 Student 和 Employee 组成的一个类等级．既作为学生又是工作人员的一个对象应同时具有学生和雇员应具有的特性．如果我们允许多重继承，则我们可以有一个学生和雇员的子类 StudentWorker，如图 1.4 所示．

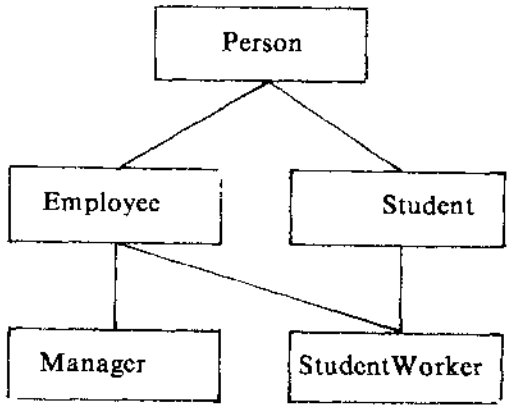


图 1.4 多重继承例子

StudentWorker 的一个实例将具有一个 Employee 和一个 Student 的状态和行为．

这里关键地方在于，虽然多重继承是一个强有力和有用的工具，但当属于不同类的多个方法和域（有可能具有同样的名字和同样的参数类型）被同一个类继承时，在定义一个相容的和直观的语义时一定要小心．使用多重继承，一个子类的域是所有父类域的一个超集，子类的接口或方法是由所有父类定义的方法一个超集（除了被覆盖的方法之外）．

1.3.4 抽象类

抽象类（Abstract class）是一种不能建立实例的类．抽象类将有关的类组织在一起，提供的一个公共的根，其它一系列的子类从这个根派生出来．抽象类刻划了公共行为的特征并将这些特征传给它的子类．通常一个抽象类只描述与这个类有关的操作接口；或是这些操作的部分实现，完整的实现被留给一个或几个子类．有时抽象类描述了这个类的完整实现，但只有在将这个类和其它的类组合在一个新的类中时它才有用．抽象类已为一个特

定的选择器集合定义了方法，并且这些方法服从某种语义，所以抽象类的通常用途是用来定义一种协议（或概念）。

例如在前面例子中的 Person 类，如果它只用来说明 Employee 和 Student 所具有的公共特征，例如名字、地址以及对它们的访问，单独的 Person 实例在实际中是不具有意义的，只有和它的子类 Emplolyee 和 Student 结合起来才能用来描述一个部门中的具体对象，因此我们可以认为 Person 是一个抽象类。

判别一个类是否是抽象类的最容易的是你是否会使用它，如果有必要建立一个类，但是你只使用从它那儿继承过来的子类，则这个类是抽象类。例如食肉动物表示一个概念，从它可以得出更多具体的推论，例如狮子和老虎。你决不会遇到一只动物，简单地认为是食肉的，它总是一个具体种类的食肉动物。

一个抽象类包含了操作接口但没有实现，就定义了一种协议。如果包含有某种实现，则抽象类用缺省实现定义了一种协议。通过从多个父类中继承定义，几个协议可被组合在一起。

1.4 多态性

由前面几节的叙述可见，问题求解的面向对象的方法使用对象作封装，并定义对象为类的实例。

在大多数面向对象的语言中，如果类 P 是子类 S 的父亲,则子类 S 的一个对象 s 可以用在父类 P 的一个对象 p 可以使用的地方,这就意味着一个公共的消息集（即操作）可以送到类 P 和类 S 的对象上。当同样的消息可以被送到一个父类的对象和它的子类的对象上时，这被称作多态性（Polymorphism）。在不同的对象上对一个相似的操作使用相同消息的能力与人类在解决问题中的思考方式是相容的，不必为打印整数、浮点数、字符、字符串和数据记录而使用不同的术语。第六章集中讨论 C++语言中的多态性。

在对问题求解的面向对象的方法中，多态性是一个关键特征。多态性与语言能够支持动态联编有关，简单地说，动态联编表示系统将在运行时间根据接收对象的类型将一个选择器和实现它的特定方法（过程）联编在一起。为理解多态性和动态联编，我们举一个例子。这个例子是打印一个由异质对象组成的集合中的每个元素，这些元素的等级关系如图 1.5 所示。

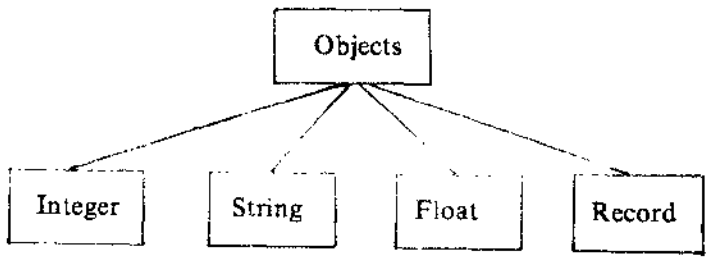


图 1.5 异质集合类等级

Objects 是一个抽象类，在这个类中，只定义了 print 方法的接口，但没有实现。每

个子类都实现有一个和这个子类有关的特殊的 print 方法（因此所有这些 print 方法都是不同的并且完全无关），假定我们有一个栈，它包含有各种类型的对象，并且我们想打印栈中的每个记录．进一步假定变量 St 表示栈，它被实现为一个 Objects 数组（以便可以保存它的子类的对象），下标从 0 到 Top，使用动态联编，打印栈中每个记录的伪码是：

```
for ( i = 0 ; i < Top ; i++ )
    St[i].print() ;
```

这样，一个公共的消息 print 通过父类被送到子类的每个对象上，每个对象 St[i]将依据它所属的类执行相应类中的 print 方法．这在图 1.6 中说明．关键一点是，对象决定为 print 消息执行哪部分代码．因此不同于传统语言中的过程名字，选择器 print 自身并没有用于确定一段唯一的代码，用于一个特定目标对象（它是一个特定类的一个实例）的 print 消息确定唯一的用于打印目标对象的代码．

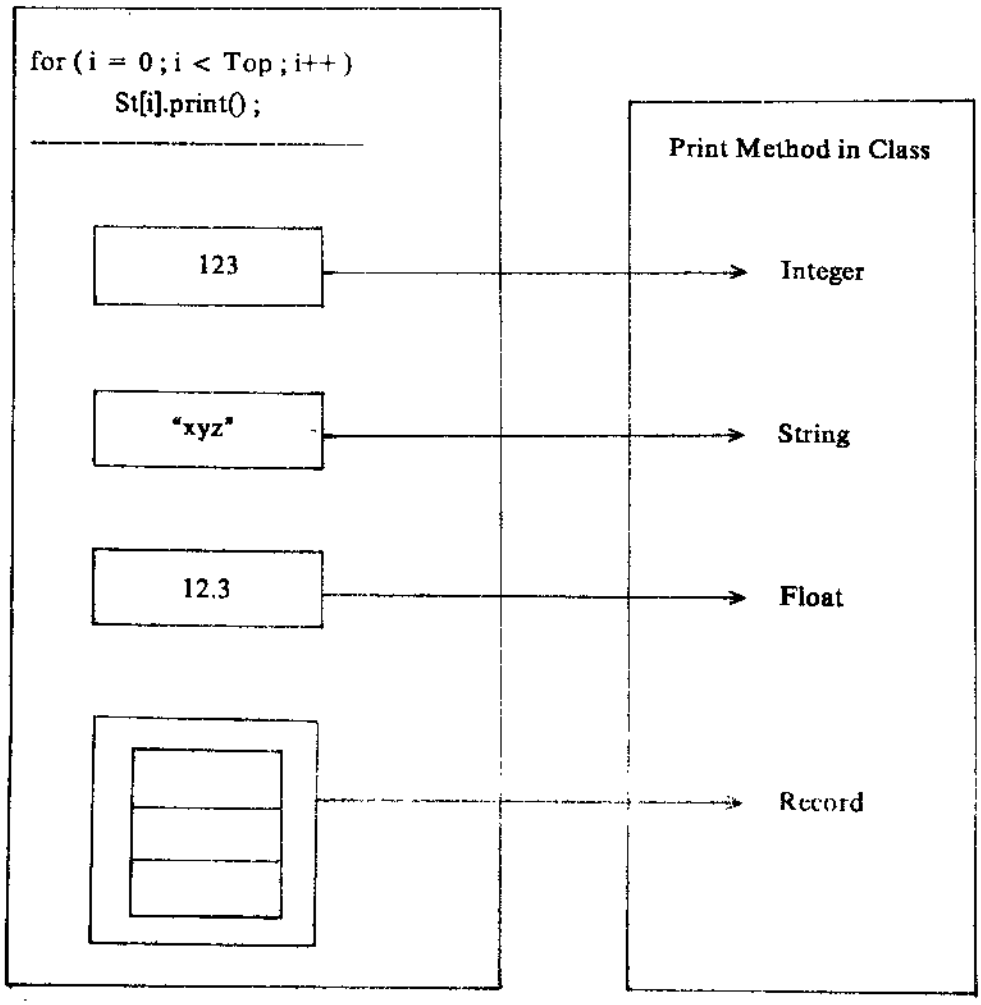


图 1.6　动态联编示例