第3章 面向对象程序设计基础

第2章介绍了Java语言基础，但实际上介绍的只是结构化程序设计的基本构成。如果需要考虑如何面对一个复杂的实际应用来使用计算机求解，也就是说要开发大型的Java程序，这时候就要考虑如何使用一种方法有效地反映客观世界，建立与客观事物相应的概念，直接表现事物的组成问题以及事物之间的联系，并建立一套适应人们一般思维方式的描述模式。面向对象技术的基本原理正是这种按问题领域的基本事物来实现自然分割和抽象，然后求解问题的方法。

使用面向对象技术进行程序设计，形成了面向对象程序设计技术。它的基本原则是：在进行程序设计时，力图按人们通常的思维方式建立问题的模型，以对象世界的思维方法思考问题，尽可能自然地表现软件的求解方法。为此面向对象技术引入对象(Object)来表现事物，用传递消息(Message)来建立事物之间的联系，用类(Class)和继承性(Inheritance)模拟人们一般思维方式来描述和建立问题领域模型。这样，从求解问题的角度看，这种技术是用对象描述问题的组成部分，利用对象簇形成问题空间，对象间的消息传递表示用户要求。

从问题空间到求解空间的映射过程就是软件开发过程，采用面向对象的开发方法将问题空间映射到求解空间时采用的是一种自然映射，即现实世界的自然对象到软件对象，这就是面向对象程序设计方法学的精髓。

采用面向对象的方法设计的软件，不仅易于理解，而且易于维护和修改，从而提高了软件的可靠性和可维护性，同时也提高了软件的模块化和可重用化的程度。

本章讨论面向对象程序设计基础，主要讨论面向对象技术的一些基本概念和面向对象程序设计的基本思想。第4章和第5章将讨论Java实现面向对象程序设计的具体实现机制。

3.1 面向对象程序设计概述

面向对象的程序设计是面向对象方法学的一个组成部分。完整地看，面向对象技术包括面向对象分析(Object-Oriented Analysis，OOA)、面向对象设计(Object-Oriented Design，OOD)及面向对象程序设计(Object-Oriented Programming，OOP)三部分内容。

(1) 面向对象分析(OOA)：软件需求分析的一种带有约束性的方法，用于软件开发过程中的问题定义阶段。其主要活动是对问题进行抽象建模，包括使用实例建模、类和对象建模、组件建模和分布建模等，产生一种描述系统功能和问题论域基本特征的综合文档。

(2) 面向对象设计(OOD)：将面向对象分析所创建的分析模型转变为作为软件构造蓝图的设计模型。面向对象设计的独特性，在于其具有基于抽象、信息隐蔽、功能独立性和模块性建造系统等4个重要软件设计概念的能力。

(3) 面向对象程序设计(OOP)：指使用类和对象以及面向对象特有的概念进行编程，在

结构化程序设计的基础上,20 世纪 80 年代初涌现出来的一种程序设计方法。

前两部分内容属于面向对象的软件工程所研究的领域,本节重点介绍面向对象的基本思想和面向对象程序设计方法。

面向对象技术不同于面向过程的程序设计,它代表一种以自然的方式观察、表述、处理问题的方法和程序设计思路。面向过程的程序设计是以具体的解题过程为研究和实现的主体,而面向对象的设计则是以要解决的问题中所涉及的各种对象为主要线索,关心的是对象及其相互之间的联系,问题求解过程力求符合人们日常自然的思维习惯,降低和分解问题的难度和复杂性,提高了整个求解过程的可控性、可监测性和可维护性,从而以较小的代价获得较高的效率和较满意的结果。

在面向对象的方法学中,“对象”是现实世界的实体或概念在计算机程序中的抽象表示,具体地,对象是具有唯一对象名和一组固定对外接口的属性和操作的集合,用来模拟组成或影响现实世界问题的一个或一组因素。其中,对象名是区别于不同事物的标识;对象的对外接口是在约定好的运行框架和消息传递机制的情况下与外界进行通信的通道;对象的属性表示了它所处于的状态;对象的操作则用来改变对象的状态的特定功能。

面向对象就是以对象及其行为为中心,来考虑处理问题的思想体系和方法。面向对象的问题求解力图从实际问题中抽象出这些封装了数据和操作的对象,通过定义属性和操作来表述它们的特征和功能,通过定义接口来描述它们的地位及与其他对象的关系,通过消息传递相互联系,协同完成某一活动。类和继承用于描述对象、建立问题领域模型和描述软件系统,最终形成一个联系广泛的可理解、可扩充、可维护、更接近于问题本来面目的动态对象的关系模型系统。

面向对象的程序设计是以对象为中心,按照对象及其联系来构造实现软件单位的程序设计。面向对象的程序设计将在面向对象的问题求解所形成的对象模型基础之上,选择一种面向对象的高级语言来具体实现这种模型。相对于传统的面向过程的程序设计方法,面向对象的程序设计具有更好的可重用性、可扩展性和可管理性,其优点具体体现在如下几个主要方面。

1. 封装性

对象的封装性消除了传统结构方法中数据与操作分离所带来的种种问题,降低了维护数据与操作之间的相容性的负担。把对象的私有数据和公共数据分离开,保护了私有数据,减少了模块间可能产生的干扰,达到降低程序复杂性、提高可控性的目的,提高了程序的可重用性和可维护性。

2. 自治性

对象作为独立的整体具有良好的自治性,即它可以通过自身定义的操作来管理自己。一个对象的操作可以完成两类功能:一是修改自身的状态;二是向外界发布消息。当一个对象想要影响其他对象时,需要调用那个对象的方法,而不是直接去改变那个对象。对象的这种自治性能够使得所有修改对象的操作都可以以对象自身所具有的一种行为的形式存在于对象整体之中,从而维护了对象的完整性,有利于对象在不同环境下的重用、扩充和维护。

3. 安全性

对象具有通过一定的接口和相应的消息机制与外界相联系的特性,并与对象的封装性结合在一起,较好地实现了信息隐藏。这样使得对象成为一只使用方便的“黑匣子”,其中隐藏了私有数据和内部运行机制。使用对象时只需要了解其接口提供的功能和操作,而不必了解对象内部的数据描述和具体的功能实现。

4. 扩展性

继承是面向对象的另一个重要特性，通过继承可以很方便地实现应用的扩展和已有代码的重复使用，在保证质量的前提下提高开发效率，也使得面向对象的开发方法与软件工程的一个新方法（快速原型法）能够很好地结合在一起，形成一种更有效、更实用的软件开发技术。

综上所述，面向对象程序设计是将数据及数据的操作封装在一起，成为一个不可分割的整体，同时将具有相同特征的对象抽象成为一种新的数据类型——类。类是构造软件系统的最小单位，一个程序结构就是一个类的集合以及各类之间以继承关系联系起来的结构。整个软件系统的实现由对象组成，它含有一个主程序，在主程序中定义各对象并规定它们之间传递消息的规律，对象通过消息的相互传递使整个软件系统运转。从程序执行的角度来看，可以归结为各对象和它们之间的消息通信。面向对象程序设计最主要的特征是各对象之间的消息传递和类之间的继承。类的继承用于描述对象、建立问题领域模型以及构造软件系统，为代码重用提供一种有效的途径。

3.2 类与对象

对象是理解面向对象技术的关键。对象在不同的上下文中可能有不同的含义。广义地讲，“万事万物皆对象”，也就是说，我们所接触的现实世界的一切事物都是对象。而在现实世界中，许多对象具有相同的类型，俗话说：“物以类聚”，因此又产生了类的概念。

3.2.1 对象

在面向对象的程序设计方法中，对象是一些相关的变量和方法的软件集，是可以保存状态（信息）和一组操作（行为）的整体。对象用于模仿现实世界中的一些对象，如桌子、电视、自行车等。现实世界中的对象有两个共同特征：状态和行为。例如，自行车的状态有车的样式、颜色、当前挡位、两个轮子的大小等，其行为有刹车以及改变挡位加速、减速等。

在面向对象设计的过程中，既可以利用对象来代表现实世界中的实物对象，例如，可用一个动画程序来代表现实世界中的动物；也可以使用软件对象来模拟抽象的概念，例如，事件是一个 GUI（图形用户界面）窗口系统的对象，它可以代表用户按下鼠标按钮或者按下键盘上的按键所产生的事件。

一个软件对象利用一个或者多个变量来体现它的状态。变量是由用户标识符命名的数据项。软件对象用方法（method）来实现它的行为，它是与对象有关联的函数和过程。例如，构造现实世界中的自行车的软件对象要有指示自行车的当前状态的变量：速度为每小时 20 千米，它的当前挡位为第 3 挡。这些变量就是实例变量，表示的是特定自行车对象的状态。图 3.1(a)是一个软件对象的直观表示。

除了变量之外，描述自行车的对象还有用于刹车、改变踏板步调以及改变挡位的方法。有了一个实例之后，这些方法就是实例方法，它们能够设置或者改变特定自行车实例的状态。自行车的状态和行为的描述见图 3.1(b)。

3.2.2 类

在软件设计中，虽然是“面向对象”，即关注的焦点是对象，但对象是依靠类来描述的，类实际上是对某种具有共同特征类型的一类对象的定义，即类定义了一类对象的类型，属于该类型

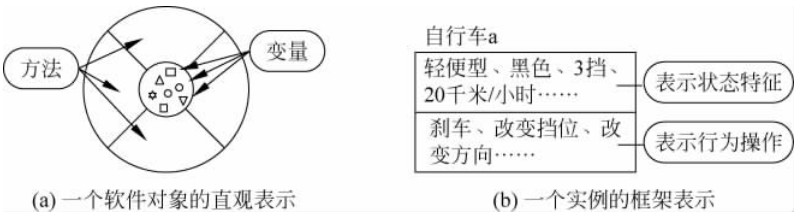

图 3.1　一个软件对象的直观表示与一个实例的框架表示

的所有对象都具有相同的变量和方法。

注意,对象和类看起来很相似,有时难以区分。类与对象的区别是:类是同一种对象的集合的抽象,即同一类对象的变量和方法的原型。例如,对于现实世界中的汽车,使用面向对象的术语,可以说这辆汽车是汽车类的一个具体对象,即实例。通常,每辆汽车都有各自的状态(如车的颜色、配置等)和行为(各种功能,自动变速、导航等)。但是,这些汽车又都具有某些共同的特性,厂商根据相同的特征形成模型,即图纸,根据图纸制造出许多同类的汽车。图 3.2 中右边图形表示汽车的图纸,左边表示根据图纸造出的汽车。那么这里的图纸就相当于类,而造出的具体汽车就是对象。因此,在面向对象的软件中,可以让许多共有的一些相同特性的对象形成类,如几何图形中的矩形类、社会生活中的雇员类等。我们可以利用相同类型的对象模型创建一个类,表达同一种对象的共同的属性和行为,所以类是对象的软件模型,是一个抽象定义、一个模板。也有人把类称为抽象数据类型。

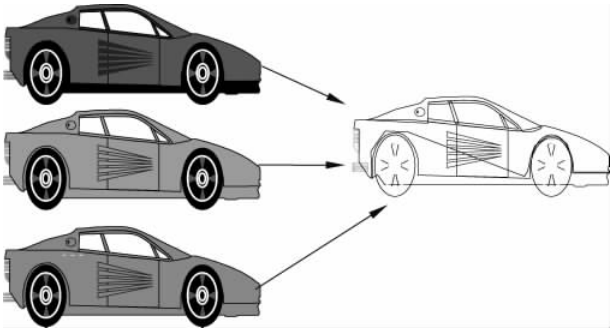

图 3.2　图纸和汽车的关系相当于类和对象的关系

我们在知道什么是类之后,下面看如何定义自行车类。在创建自行车类的时候,需要定义一些实例变量来包括当前挡位、当前速度等,同时这个类也需要为操作这些变量提供方法定义和方法实现,允许使用者改变挡位、刹车以及改变脚踏板的节奏。图 3.3(a)和图 3.3(b)分别给出自行车类的两种表示。

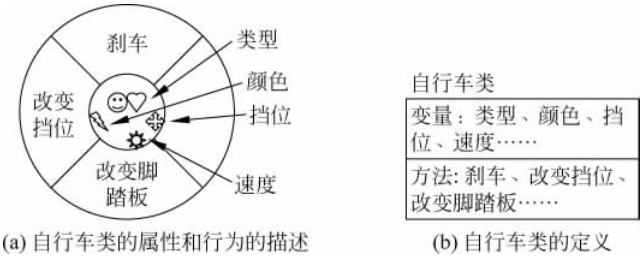

图 3.3　自行车类的两种表示

定义了自行车类以后，就可以用这个类创建任意多个自行车对象。创建了一个类的对象后，系统将为这个对象的实例变量分配内存。这样，每个对象的实例变量可存放各自不同的状态值。这个对象就对应一个实实在在的对象。

3.2.3 消息

单一的对象通常用处不是很大，所以在一个应用中通常包含许多对象，并且通过这些对象的交互作用，可以表现更为复杂的行为或获得更强的功能。而消息是对象间进行联系或者交互的手段，是一个对象向其他对象发送请求执行某个操作的信号或命令。如图 3.4 所示，如果对象 A 要执行对象 B 中的一个方法，对象 A 就会发送消息给对象 B。

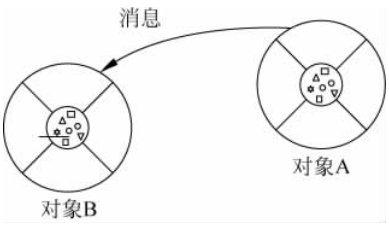

图 3.4 对象之间的消息

图 3.4 中，对象 A 为消息的发送者，对象 B 为消息的接收者。一般来说，接收消息的对象需要足够的信息才能知道该如何响应，这些信息规范就是它们之间的接口。

一个对象 A 发送消息给另一个对象 B，也意味着接收消息的对象 B 具有对象 A 所需要的某种行为。因此，发送消息，也就是对另一个对象的方法的调用。

3.2.4 类的成员

在前面我们看到，在定义一个类的时候，用变量来表示类的属性，这个属性是类的一部分，也就是所谓的成员变量，而表现一个类的行为的过程或函数，则称为成员方法。类的成员变量和成员方法构成类的成员。

在定义一个类的时候，实际存在两种类型：实例成员和类成员。

首先看看成员变量，假如要定义一个“公民类”，身份证号码就是公民的属性，每个公民就是这个类的一个实例，这时身份证号码就是公民类的一个实例变量，即每个公民对象都有一个变量来保存身份证号码。

实际上，在面向对象的实现系统中，每次创建一个类的对象的时候，系统为这个对象的每一个实例变量创建一个存储空间。这样就可以从对象中访问该实例变量。

除了实例变量，在类中还可以定义它的另一种类型的变量——类变量。类变量包含了类的所有实例共享的某种信息。例如，假设所有的自行车有相同的挡位数，如果要定义一个实例变量来存放挡位数，要给每一个对象都分配该变量的存储空间。而每个对象中该数值都是相同的，这样太浪费空间。故可以定义一个类变量来表示挡位的个数，这样所有的类的对象都共享这个变量。如果某个对象改变了这个变量值，也就改变那个类的所有对象共享的这个值。

在面向对象的实现系统中对类变量的管理方式是：不管为一个类创建了多少个对象，系统只为每个类变量分配一次存储空间。系统为类变量分配的内存是在它第一次调用类的时候完成的。所有的对象共享了类变量的相同存储空间。类变量可以通过对象名或者类名本身来访问。

与类的实例变量和类变量对应的是：类可以有实例方法和类方法。

实例方法对当前对象的实例变量进行操作，而且可以访问类变量。而类方法只可以操作类变量，但不能访问定义在类中的实例变量，除非它们创建一个新的对象并通过对象来访问实例变量。同样，类方法可以用类名直接调用，并不一定必须建立一个实例来调用一个类方法。

一个类的所有对象共享该类实例方法的实现。因此,类的所有对象都共享相同的执行行为。那么这样各个对象会不会在执行时造成混乱呢? 当然不会。因为在实例方法中,能够使用的是所在对象所拥有的实例变量,每一个实例有自己实例变量的存储单元,每个存储单元又有自己的值,每个对象的方法在执行时不会与其他对象的实例变量混淆。

一般默认一个定义在类中的成员就是一个实例成员。每当一个类创建一个新的对象时,可以得到该类的实例变量的副本,这些副本都属于新对象。

一个类外部的对象如果想访问某个实例变量,它必须通过特定的对象来实现。如果不通过特定的对象来实现,就应该在声明成员变量的时候要另加特别的声明,指定该变量是一个类变量而不是一个实例变量。类似地,如果不通过特定的对象来访问一个方法,可以指定这个方法是一个类方法而不是一个实例方法。

简而言之,一个类的成员有实例成员和类成员之分,实例变量和实例方法是一个对象中的成员,类变量和类方法是类中所有对象所共享的成员。可以直接通过类名使用类变量和类方法,也可以在对象中使用类变量和类方法,然而实例方法和实例变量必须在特定的实例中使用。

3.3 抽象与封装

抽象与封装是面向对象程序设计的两个重要特点。

3.3.1 抽象

抽象是分析和设计中经常使用的一种重要方法,即去除被分析对象中与主旨或本质无关的次要部分或非本质部分以及可以暂时不考虑的部分,而仅仅抽出与研究对象有关的实质性的内容加以考察。在计算机软件开发方法中所使用的抽象有两类:一类是过程抽象;另一类是数据抽象。

过程抽象将整个系统的功能划分为若干部分,强调系统功能完成的过程和步骤。面向过程的软件开发方采用的就是这种抽象方法。使用过程抽象有利于控制和降低整个程序的复杂度,但是这种方法本身自由度较大,难于规范化和标准化,操作起来也有一定难度,因而不易保证质量。

数据抽象是把系统中需要处理的数据和在这些数据上的操作结合在一起,根据功能、性质和用途等因素抽象成不同的抽象数据类型。每个抽象数据类型既包含了数据,又包含了针对这些数据的授权操作。所以,数据抽象是比过程抽象更为严格、更为合理的抽象方法。

面向对象的软件开发方法的主要特点之一,就是采用了数据抽象的方法来构建程序的类、对象和方法。实际上,软件工程中面向对象软件开发过程中的面向对象分析,就是对实际问题进行抽象,从而建立物理模型的过程。

在面向对象技术中使用这种数据抽象方法,一方面可以去除与核心问题无关的细节,使开发工作可以集中在比较关键和主要的部分;另一方面,在数据抽象过程中对数据和操作的分析、辨别和定义可以帮助开发人员对整个问题有更深入、更准确的认识。最后抽象形成的抽象数据类型,则是进一步进行设计和实现的基础和依据。

抽象可以帮助人们明确工作的重点,抓住问题的本质,理清问题的脉络。面向对象的软件开发方法之所以能够处理大规模、高复杂度的系统,抽象手段发挥了重要作用。

3.3.2 封装

面向对象方法的封装性是一个与抽象密切相关的重要特性。具体来说，封装就是指利用抽象数据类型将数据和基于数据的操作结合在一起，数据被保护在抽象数据类型的内部，系统的其他部分只有通过包裹在数据之外被授权的操作，才能够与这个抽象数据类型进行交互。

在日常生活中，我们接触和处理的事物几乎都是以封装的对象的方式进行的，如调整时钟、驾驶一辆汽车、使用一台电视机等。这些都是封装的对象，通常包含某种固定的信息，并且提供使用这些信息的一些工具。这种对用户隐藏对象复杂性的方法在现实世界中使用得非常普遍，封装性使得用户不必了解部件内部复杂的代码，就能知道某些信息，并能使用这些构件操纵软件对象。同时，用户可以不必了解封装对象的内部工作机制，就可以利用它们来开发更大的程序。

在面向对象的程序设计中，抽象数据类型是用前面提到的“类”这种面向对象的结构来实现的，每个类里都封装了相关的数据和操作。在实际的开发过程中，类在很多情况下用来构建系统内部的模块，由于封装性把类的内部数据保护得很严密，模块与模块间仅通过严格控制的界面进行交互，使模块之间耦合和交叉大大减少。从软件工程的角度看，大大降低了开发过程的复杂性，提高了开发的效率和质量，也减少了出错的可能性，同时还保证了程序中数据的完整性和安全性。

例如，在银行日常业务模拟系统中，可以建立“账户”这个抽象数据类型，它把账户的金额和交易情况封装在类的内部，系统的其他部分没有办法直接获取或改变“账户”类中的关键数据，只有通过调用类中的适当方法才能做到这一点，如调用查看余额的方法来了解账户的金额；调用存、取款的方法来改变金额等。只要给这些方法设置严格的访问权限，就可以保证只有被授权的其他抽象数据类型才可以执行这些操作和改变“账户”类的状态。这样，就保证了数据安全和系统的严密。

面向对象技术的这种封装性还有另一个重要意义，就是它可以使类或模块的可重用性大大提高。封装使得抽象数据类型对内成为一个结构完整、可自我管理、自成体系的整体；对外则是一个功能明确、接口单一、可在各种合适的环境下都能独立工作的有机单元。这样的有机单元特别有利于构建、开发大型的应用软件系统，可以大幅度地提高软件生产效率，缩短开发周期，以及降低开发和维护的费用。

3.4 继承与多态

在面向对象的技术中，继承与多态是不同于传统方法的一个最具特色的特征。

3.4.1 继承的定义

继承是面向对象的程序中两个类之间的一种关系，是一个类可以从另一个类（即它的父类）自动获得状态和行为。被继承的类也可称为超类，继承父类的类称为子类。继承为组织和构造软件系统提供了一个强大而自然的机理。

从前面的讨论已经表明，对象是以类的形式来定义的。面向对象系统允许一个类建立在其他类之上。例如，山地自行车、赛车自行车以及双人自行车都是自行车，那么在面向对象技术中，山地自行车、赛车自行车以及双人自行车就是自行车类的子类，自行车类是山地自行车、

赛车自行车以及双人自行车的父类。这个父类与子类关系可以通过图 3.5 表示。

一个父类可以同时拥有多个子类,这时这个父类实际上是所有子类的公共变量和方法的集合,每一个子类从父类中继承了这些变量和方法。例如,山地自行车、赛车自行车以及双人自行车共享了这些状态:双轮、脚踏、速度等。同样,每一个子类继承了父类的方法,山地自行车、赛车自行车以及双人自行车共享这些行为:刹车、改变脚踏速度等。

然而,子类可以不受父类提供的状态和行为的限制。除了从父类继承而来的变量和方法之外,子类可以增加自己的变量和方法。例如,双人自行车有两个座位,可以增加一个成员变量:后座位,这是它的父类没有的,即对父类进行了扩充。

子类也可以改变继承的方法,也即可以覆盖继承的方法,并且为这些方法提供不同于父类的特殊执行方法。例如,玩杂技的自行车,不仅可以前进,而且还可以后退,这就改变了普通自行车(父类)的行为。

此外,类是逐级继承的,继承的层次不能限制,继承树或者类的分级结构可以很深,如图 3.6 所示。一般来说,越处在分级结构的下方,就越有更多的状态和行为。

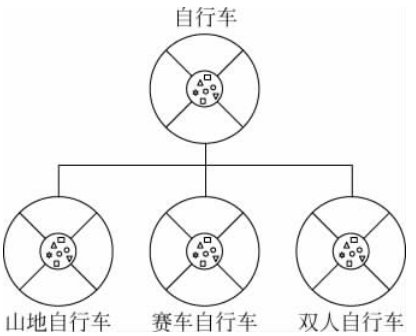

图 3.5　自行车的父类与子类的关系

图 3.6　类继承层次示意图

3.4.2　继承的优越性

继承的优越性在于:通过使用继承,程序员可以在不同的子类多次重新使用在父类中的代码,使程序结构清晰,而子类又可以提供一些特殊的行为,这些特殊的行为在父类中是没有的。

一般可以定义一个抽象类作为父类来定义多个类共同的属性和行为,如图 3.6 中的形状类。这个父类可以定义和实现子类的共同行为,如计算这些形状的面积、周长,并提供一个统一的接口。但是绝大部分父类的行为是未定义和未实现的。这些未定义和未实现的部分一般留给有待于具体实现的特殊子类。例如,在子类给出计算面积和周长的具体计算方法。

采用继承机制组织、设计系统中的类,可以提高程序的抽象程度,使之更接近于人们的思维方式,同时也可以提高程序的可重用性,从而提高程序开发效率,降低维护成本。

3.4.3　多态性

在面向对象程序设计中,多态性是一个重要特性。多态是指同名的多个方法共存于同一个程序中的情况,在软件设计过程中,有时候需要利用这种“重名”现象来提高程序的抽象性和简洁性。

如前所述,一个对象由类生成,将对象以某种方式连接起来,就为一个软件模块提供了所

需要的动态行为。模块的动态行为是由对象间相互通信而发生的，多态的含义是一个消息可以与不同的对象结合，产生不同的行为，而且这些对象属于不同类。同一消息可以用不同方法解释，方法的解释依赖于接收消息的类，而不依赖于发送消息的实例。多态通常是一个消息在不同的类中，用不同方法实现的。

多态的实现是由消息的接收者确定一个消息应如何解释，而不是由消息的发送者确定，消息的发送者只需知道另外的实例可以执行一种特定操作即可，这一特性对于可扩充系统的开发是特别有用的工具。按这种方法可开发出易于维护、可塑性好的系统。例如，如果希望加一个对象到类中，这种维护只涉及新对象，而不涉及给它发送消息的对象。

多态与具体的面向对象语言有关，在后面 Java 语言的学习过程中还会讨论如何具体实现多态的问题。

面向对象技术中的多态是一个非常重要的概念。多态性可以大大提高程序的抽象程度和简洁性，更为重要的是，多态性极大地降低了类和程序模块的耦合性，提高了类和模块的封闭性，使得在类和模块之外，不需要了解被调用方法的实施细节就可以很好地协同工作。这对程序的设计和维护都会带来很大的方便。

3.5 小　　结

面向对象编程是 Java 语言程序设计的核心，它的目标就是创建抽象的、可以运行的软件对象，这些软件对象就像现实世界的对象一样能够使用。Java 语言的程序设计机制全面而系统地实现面向对象的程序设计。本章着重讨论有关面向对象程序设计的一些基本概念。

- 类：它是每个面向对象的程序的基本结构，包含数据域和操作这些数据的机制；类提供了创建软件对象的模板。
- 实例：实例是以类为模板创建的对象，一个类可以用来生成任意多个实例。
- 封装性：允许或禁止访问类或对象的数据和成员方法的一种机制。
- 重载性：允许一个同名的成员方法有多重定义，执行时可根据不同场合选用不同的定义。
- 继承性：获得相关类已经具备的一些特征的能力。
- 多态性：处理基于公共特征的多个相关类之间的能力，可根据不同的环境调用不同类的方法。

习　题　3

1. 什么是面向对象技术？
2. 什么是面向对象程序设计？面向对象程序设计的优点有哪些？
3. 简述抽象与封装、继承与多态性的概念。
4. 什么是对象的自治性？
5. 什么是多态？面向对象程序设计为什么要引入多态的特性？使用多态有哪些优点？
6. 什么是消息？