第5章　类和对象

面向对象程序设计思想的三大特征是封装、继承和多态，本章介绍如何构造类、实例化对象以及灵活使用对象编程的方法。

学习目标：

- 理解类和对象的概念及两者间的区别。
- 掌握类的定义以及类之间的关系描述，认识对应的 UML 基本图示。
- 理解构造方法的作用，能够使用 new 运算符实例化一个对象。
- 理解对象的引用，掌握通过变量名访问引用对象的成员变量和方法。
- 理解并掌握变量的作用域。
- 理解关键字 this 的用法。
- 掌握 static 的作用。
- 理解并掌握方法的声明和调用。
- 理解包的概念并掌握包的作用。

5.1　面向对象技术的基础

在面向对象方法中，“对象”是现实世界的实体或概念在计算机逻辑中的抽象表示。面向对象就是现实世界模型的自然延伸，现实世界是由一个个处于特定状态的对象组成。对象的状态由该对象当前的各种属性值确定。例如，描述一个在校大学生通常需要使用姓名、年龄、籍贯、专业、入学日期等信息，可以用课程代号、课程名称、课程简介等属性来表示一门具体课程，而描述银行的一笔存款业务需要账户、存款金额、存款日期等属性。

除了用属性描述对象的状态外，一些具有主动性的对象还具有行为特征。例如，一个学生可以选择某一门课程作为自己在某个学期的选修课，另外一些被动的对象，如上述的课程和银行业务，虽然本身没有操作，但是从维护的角度考虑，可将围绕对象状态管理而提供的特定方法视为对象本身拥有的操作，对将来因为修改代码而产生的影响最小。例如，一个银行账户对象可以通过关闭操作修改自己的可用状态，也可以通过取款的方法维护自己的余额。

数据和施加于数据之上的操作构成了对象的整体，二者不可分割。数据项不能自我改变，除非一个操作应用于该数据项。把对象的两个方面集中在一起进行建模的方法，被称为封装(Encapsulation)，它是面向对象的一个最重要的特性。封装的目的是增强安全性、简化编程，可提高代码的可维护性。使用者不必了解具体的实现细节，只通过外部接口，就可以特定的访问权限访问对象。这和生活中的很多现象是一致的，就像通过电源开关打开或关闭电视机，却不必对电视的开关电路有深入了解。

在现实世界里，有许多属于同一类的个体，例如商场里有各种各样的电视机，这里使用了“电视机”这样一个名词来描述，这种概念化的描述实际上就是抽象化的过程，所谓抽象

化，就是过滤每一个对象的个体信息，提炼这些对象具有的共同特征。在面向对象术语中，使用“类”来表示这种抽象化的数据类型，它是创建对象的蓝图，对象就是类的实例。

有时需要对某一类的对象进行进一步区分，除了强调它们的共性之外，还要考察它们各自的特性，例如按照显示技术电视机又可分为 CRT、等离子和液晶等种类，但它们共同具有电视机的基本特征，这种现象被称为“继承”，继承是面向对象语言的另外一个重要特征。借助继承，为父类定义的属性和操作可以在子类中复用，而且子类可以扩展原有的代码，应用到其他程序中，它是实现软件复用的重要机制。

对象从来都不是孤立的。每个对象都与其他对象有着直接或间接联系，这种联系或强或弱，正是这种对象以及对象之间的联系构成了一个完整的系统。例如，一个家庭是由几位家庭成员组成的，这是一种典型意义上的“整体和部分”的联系；另一种联系，例如书架和书之间的联系，就不如家庭和成员之间的联系紧密，没有书，书架仍然是书架，这是一种典型的关联关系。除了这种结构上的关系，通过对象之间的相互交互是构成更复杂功能的基础，进行面向对象程序设计时，把一个对象向目标对象发出消息请求，目标对象接收并执行消息的过程称为“消息传递”。

面向对象技术的抽象、封装、继承、消息传递、多态等重要特性，能够为开发带来更好的复用性，改善了软件的质量。因此，面向对象开发方法是目前最重要的程序设计方法之一。

5.2 使用 JDK 的类

掌握 Java 开发的基础，首先要了解 JDK 提供了哪些可以让开发人员使用的 API。JDK 提供了丰富的类，被分别存放在不同的包中，如表 5-1 所示。

表 5-1 JDK 的一些包

包	作 用
java.lang	包含 Java 的基础类，无须用 import 加载
java.lang.reflect	提供类和接口，以获取关于类和对象的反射信息
java.io	通过数据流、序列化和文件系统提供系统的输入和输出
java.text	可通过类或接口，以与自然语言无关的方式处理文本、日期、数字和消息
java.util	包含集合框架、遗留的 collection 类、事件模型、日期和时间设施、国际化和各种实用工具类（字符串标记生成器、随机数生成器和位数组）
java.util.concurrent	在并发编程中很常用的实用工具类
java.net	提供网络编程的类和接口
javax.swing	提供支持 Swing GUI 组件的类
java.sql	提供有关数据库访问的接口和类

Java 的每个类都有特殊的作用。例如，java.lang 包下的 Math 类提供了一些基本的数学运算。程序 5-1 显示了如何利用 Math 类中 random 方法连续产生 10 个随机数并输出到控制台。

```
        //程序 5-1:一个利用 Math 类产生随机数的程序
        publicclassRandomDemo {
            publicstaticvoid main(String[] args) {
(12)            for(inti=0;i<10;i++){
(13)                //利用 random 方法产生一个取值范围是 0.0~1.0 的随机数
(14)                double r=Math.random();
(15)                System.out.println("本次产生的随机数是:"+r);
                }
            }
        }
```

可以在程序中直接使用 java.lang 包下的类,但是对于非 java.lang 包下的类就必须使用 import 关键字引入程序中,以便在运行时能够将它们加载到运行时空间中使用,例如程序 5-2 利用 javax.swing 包下的 JFrame 类就构建一个具有窗口特征的程序。

```
        //程序 5-2:一个窗口程序
        import javax.swing.JFrame;
        public class FrameDemo {
            public static void main(String[] args) {
(01)            JFramefrm=new JFrame("我的窗口程序");
(02)            frm.setSize(400, 300);
(03)            frm.setVisible(true);
            }
        }
```

程序 5-2 的第 1 行首先使用 import 关键字将一个 javax.swing.JFrame 类加载进来,否则后面的程序在用到此类时,将会提示 JFrame cannot be resolved to a type 错误。

将对象实例化的主要目的是利用它所提供的方法来达到某种目的。例如 Math 类的 random 方法可以返回给程序一个取值范围是 0.0～1.0 的随机数,程序 5-3 的第 3 行使用一个变量 r 保存它的值,供后续的语句使用。

有的类纯粹是为了提供某些功能。例如,Math 类中的方法是一些特殊的类方法,无须创建 Math 类的对象就可以直接使用,而使用 JFrame 类则必须先创建一个该类的实例对象,程序 5-2 的第 1 行用变量 frm 引用这个对象,第 2 行通过 frm 调用 setSize 方法可以调整窗口的显示尺寸,第 3 行调用 setVisible 方法则可以使窗口对象显示出来。

通过变量引用一个实例化的对象是 Java 程序设计时使用对象的常态。一个程序在运行时碰到用 new 运算符实例化一个对象时,就在程序所管理的一块特殊内存(堆)中分配足够的空间存储此对象,然后在栈中增加一个变量,这个变量指向了对象,如图 5-1 所示。

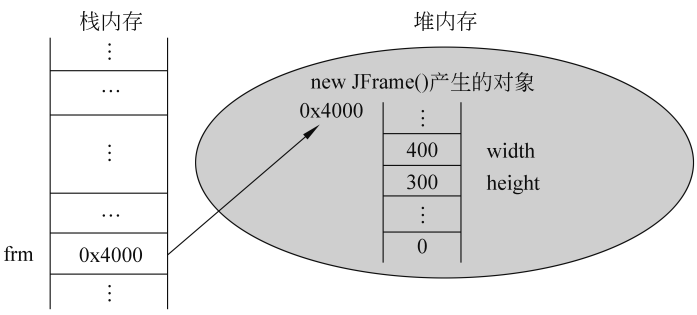

图 5-1　变量声明和变量所引用的对象

如图 5-1 所示，变量并不是对象，它只是通过变量名调用的方法将引用的对象执行，并把结果返回。

5.3 创建自己的类

虽然 JDK 提供了很多功能各异的类供编程使用，但它无法提供学生、课程、账户、交易等和某种应用场景密切相关的类，因此面向对象软件的开发者需要自行为应用创建合适的类。

5.3.1 类的结构

一个完整的类定义包括 3 个基本组成部分：类名、类的属性、类的方法。为了更好地理解 Java 中类的定义，先看一下如下的银行账户类的定义。

```
import java.util.Date;                          //加载所需的外部类
public class Account {                          //开始 Account 类的声明
    //首先定义该类对象的属性
    private String id;                          //账号
    private String name;                        //姓名
    privatedouble balance;                      //余额
    private Date openDay;                       //开户日期
    //定义构造方法，一个类可以有多种构造对象的方法
    public Account()    {                       //一个不需要参数的构造方法
    }
    //另一个需要参数的构造方法
    public Account(String id, String name, double balance) {
        this.id=id;
        this.name=name;
        this.balance=balance;
        this.openDay=new Date();
    }
    //以下开始定义该类对象的方法
    public String getName() {
        returnthis.name;
    }
    publicvoiddeposit(double amount) {
        this.balance = this.balance + amount;
    }
}
```

上面的 Account 类可以用来描述银行中的账户信息，每个账户都有自己的账号、姓名、余额，并提供了获得账户人姓名和存款的方法等。根据上面类的定义，表 5-2 总结了定义一个类时，类的各部分的详细说明。

表 5-2 类所包含的主要信息

元　素	必选	作　用
加载部分		在当前类定义中引用了其他的类，例如上例中用到的 Date 就属于 java.util 包内的类，需要在此声明加载

续表

元　　素	必选	作　　用
类名	√	其他地方可以通过这种类型声明该类型的变量，例如 Account act
类变量		供所有该类对象共享的属性，该属性值随类而存在，只有一份，对象和类均可以通过类名访问，例如 System.out，out 就是 System 类的一个类属性。此类属性声明时，必须在变量名字前加上 static 属性修饰符
类初始化块		当类加载时执行的代码块
实例变量		用于描述每个对象的状态，每个对象各自存储自己的状态信息，例如，每创建一个 Account 对象，该对象中的账号 id 都是唯一的
构造方法		构造方法的名字与类名相同，其作用是控制该类对象的初始化，一个类可以有若干种构造方法
类方法		属于类所有，但该类所有对象均可使用。由于执行类方法无须通过该类的任何一个对象，所以在类方法中不能使用从属于对象的对象属性和对象方法。声明时，只需在方法名字前加上 static 属性修饰符
实例方法		只能通过变量名调用对象的方法来使用
注释		良好的注释，有助于开发人员更好地使用或维护类，编译器在编译时忽略注释信息

对照上面的例子，可以看到构成一个类的主要部分，下面是一个类的基本定义语法。

```
[类的修饰符] class 类名 [extends 父类名] implements[接口列表] {
    [修饰符] 类型 成员变量 1;
    [修饰符] 类型 成员变量 2;
    构造方法 1([参数 1,[参数 2,…]]){
        方法体;
    }
    构造方法 2([参数 1,[参数 2,…]]){
        方法体;
    }
    …
    [修饰符]  类型  成员方法 1(参数 1,[参数 2,…]) [throws 异常列表 ]{
        方法体;
    }
    [修饰符] 类型  成员方法 2(参数 1,[参数 2,…]) [throws 异常列表 ]{
        方法体;
    }
    …
}
```

注意：在声明一个类时，类的规模也是一个需要考虑的因素，应避免一个类中包含的内容过多，从而变得无所不能。一般来说，类的属性和方法数量要进行适当控制。例如，每个类只包含少量的几个方法。如果规模过大，就要考虑构建的模型是否合理，是否需要重新规划。

5.3.2　声明自定义类

使用 class 关键字可以声明一个自定义的类，如前面出现的账户类 Account。

```
class 类名{
    类体;
}
```

其中,class 是关键字,表示声明的是一个 Java 类,类名可以是任何符合标识符规定的名称,通常情况下,类名的第一个字母要大写,如果类名是由多个单词组成的,则每个单词的第一个字母均要大写。这种简单的声明,并不能完全体现出一个类声明的全部特征。

一个完整的类声明语法如下:

```
[类的修饰符]class 类名 [<类型列表>] [extends 父类名] [implements 接口名] {
    类体;
}
```

表 5-3 对类声明部分的内容给予了基本的解释。一个类可以在 class 关键字之前加上可选的访问范围修饰符,来限定此类对其他类的可见范围,也可以使用 extends 关键字表明此类继承于一个已存在的超类,从而获得超类已有的定义。

表 5-3　类的声明元素表

元　　素	必选	作　　用
public		该修饰符限定了类的可见范围。public 表示该类可被其他任意一类使用,如果没有 public,则该类只能被定义在同一个包内的其他类使用
abstract		不能和 final 修饰符同时使用。声明当前类为抽象类。Java 不允许直接用 new 运算符创建类型为抽象类的对象
final		不能和 abstract 修饰符同时使用。声明当前类为 final(最终类),表示不能再定义该类的子类
class 类名	√	关键字 class 告诉编译器这是一个类
＜类型列表＞		用“,”分开的类型列表,表示该类是一个范型类(generic class),一般用于构造一个具有通用目标性质的容器类,也可以简单理解为类的参数
extends 父类名		Java 具有良好的继承体系,每个类只可以继承一个父类,所有类的根类都是 Java 核心包的 java.lang.Object
implements 接口列表		用“,”分开的接口列表。每个类可以实现一到多个接口定义的方法集

下面是一些使用不同修饰符组合进行类定义的例子,更多的类的声明在后续章节中会陆续提到。

1. 没有访问范围修饰符的简单类

```
class Account{
    …
}
```

(1) 在一个 Java 源程序文件中,可以同时存在多个并列的 class 声明。

(2) class 前没有访问范围修饰符的类,只能被同一个包内的其他类所见。

2. 为类添加访问范围修饰符 public

```
public class Account {
    …
}
```

(1) 一个类如果添加了 public 访问范围修饰符,则此类可以被任何类(包内和包外)所见并使用。

(2) 一个 Java 源程序如果包含了用 public 修饰的类,则文件名必须和这个类名保持完全的一致(包括大小写)。

(3) 一个 Java 源程序中只能包含一个 public 声明的类。

作为一个良好的命名习惯,类名一般用一个有特定含义的名词(望文生义)来表示,可以用几个单词连在一起,每个单词的第一个字母要大写,例如 Account、CashAccount 等。

5.3.3 为类添加成员变量

由 2.2.1 节可知,一个类有两种类型的成员变量:类变量和实例变量。每个对象的状态都是由成员变量的值综合体现的,成员变量有时也被称为字段(field)。下面是成员变量声明的几种形式:

```
[修饰符] 类型变量名;
[修饰符] 类型变量名[=初值];
[修饰符] 类型变量名[=初值] [,变量名[=初值]…];
```

常用的是第一种声明形式。成员变量的声明主要包括 3 部分。

(1) 零个至多个修饰符。访问范围修饰符 public 或 private 等都是常用的修饰符。

(2) 变量的类型。这是不可或缺的部分,变量的类型可以是原始数据类型,也可以是引用类型,需要根据变量所要表示的内容来确定。

(3) 变量名的命名遵循标识符的规定。例如,为了描述一个账户 Account,需要知道这个账户的所有者姓名、账户的余额,以及为了区别同名的情况而特别使用的唯一的账户编号。下面就是关于 Account 的成员变量的声明。

```
public class Account{
private String id;                          //一个 String 类型的账户的唯一编号
    private String name;                    //账户所有人的姓名
    private int balance;                    //用整数表示的该账户的当前余额
}
```

因为姓名通常是连续的字符,所以可以使用 Java 的 String 类型作为变量的类型,账户的余额,可以根据余额可能的精度范围用 double、float 或 int 等表示,银行账户虽然多数用一个很长的数值表示,但它几乎不会用于数学运算。此外,某些账户编号也可能会以 0 开头,所以用 String 类型来表示账户编号的类型比较合适。

每个成员变量前,可以根据需要添加一定的修饰符,用来达到特殊的目的,表 5-4 列出了关于成员变量声明的一些说明。

表 5-4 类的成员变量声明元素

元　素	必选	作　用
public|private|protected|default		访问控制范围修饰符加上默认没有任何访问控制范围修饰符的 4 种情况,规定了成员变量的可见范围,具体如表 5-5 所示
static		声明该变量为类变量

续表

元　　素	必选	作　　用
final		声明该变量为最终变量，一旦赋值则不可修改
transient		标记该变量不被串行化，即变量值不能用于基于 socket 的网络访问
volatile		一个变量被声明为 volatile，表明它可以被异步修改，编程时需要注意
type	√	每个变量必须规定它的类型，可以是基本类型（int、float、double 等），也可以是引用类型（某种类或接口，例如 Account、List 等）
name	√	变量的名称，要符合标识符的规定，在同一个作用域范围内名称不能重复

1. 成员变量的初始化

每个成员变量在没有明确初始化的情况下，编译器将会给成员变量赋予变量所属类型的默认值，例如数值型的变量默认值为 0，引用类型的变量默认值是 null。

2. 访问控制范围修饰符

访问控制范围修饰符限定了外界对对象内部的可见程度，Java 提供了几个修饰符，用来对成员属性、方法和类的访问，表 5-5 列出了不同访问范围修饰符的可见区域。

表 5-5　访问控制范围修饰符可见区域

名称	修饰符	类	子类	包	所有类
公共	public	√	√	√	√
默认	—	√		√	
保护	protected	√	√	√	
私有	private	√			

例如，如果一个 Account 类的属性 name 访问控制修饰符是 private，在类以外的代码中，就不能通过“引用对象变量名.成员变量名”的形式访问，例如下面的语句就是错误的。

```
oneAccount.name="张华";                    //因为 name 是私有的，无法通过对象引用来访问
```

使用 private 修饰的属性，其可见范围只能在本类的方法，其他类的方法无法访问。

3. 提供 get 和 set 方法访问隐藏的成员变量

作为一种规范，成员变量的访问控制范围建议用 private 修饰，这样成员变量对外就不可见。如果外界需要获得对应变量的信息，类应提供一个 getXxxx 方法返回对应变量的值，类也可以提供 setXxxx 方法让外界传递参数为对应的变量赋值，通过这样的方式保证所有对属性值的存取操作均通过唯一的途径进行，这也是一种信息隐藏的表现，代码如下：

```
class Account {
    private String id;                     //每个账户的账号，注意 private 修饰符
    public  String getId(){
        return this.id;                    //返回当前账户的账号
    }
```

```
    public void setId(String id){        //用接收的 id 作为当前账户对象的新 id
        this.id=id;
    }
}
```

通过这种信息隐藏,外部使用者无法直接访问一个被隐藏起来的成员变量值,只能通过提供的 get 和 set 方法,这样一来,就可以对使用者提供的值进行检查和加工。

4. 变量作用域

在类中定义的各种类型的变量,都存在各自的作用区域,一旦超出了起作用的区域,变量就不能发挥作用,程序如果引用了不在控制范围的变量将会引起错误。表 5-6 中列出了每个位置上定义的变量起作用的范围。

表 5-6　变量作用域

位置	类方法	类初始化块	所有对象	成员方法	代码块	备　注
类变量	√	√	√	√	√	该类的所有实例
成员变量				√	√	所有成员方法
方法参数				√	√	同一方法内
局部变量					√	同一代码块
异常捕获参数					√	异常处理块内

另外,在一个小范围内定义的变量名称可能会和包含它的域范围内的另外一个变量同名,在这种情况下会出现同名变量覆盖的情况,也就是默认情况下,该变量代表小范围内定义的变量。

实例变量和类变量用来描述对象的属性,类中的所有方法都可以访问这些变量,所以它们被称为全局变量(Global Variable)。在方法中说明的变量称为局部变量(Local Variable),因为它们只能在方法内部使用。下面是一些关于变量声明的很好的实践经验总结。

(1) 只在需要使用时,才去声明变量,不要提前声明。

(2) 声明变量时,需要合理地进行初始化。

(3) 不要在循环体内部声明变量,这样会影响效率。

程序 5-3 演示了关于变量作用域的影响。

```
//程序 5-3:变量的作用域
public class TestScope {
    int x;
    public static void main(String[] args)    {
        int x=12;
        {
            int q=96;                  //x 和 q 都可用
            int x=3;                   //错误的定义,Java 中不允许有这种嵌套定义
            System.out.println("x is "+x);
            System.out.println("q is "+q);
        }
        q=x;                           //错误的行,只有 x 可用, q 超出了作用域范围
```

```
        System.out.println("x is "+x);
    }
    public int getX(){
        return x;                          //返回的是实例的成员变量 x 的值
    }
}
```

在 main 方法中首先声明了一个局部变量 x，并赋给了 12 的初值，随后在复合语句中，又定义了一个变量 x，这是不允许的，因为在复合语句外已经有一个局部变量 x 被声明了；当程序运行完复合语句后，将 x 的值赋给变量 q 时，q 已经是非法变量了，因为它是在复合语句内声明的，退出了复合语句，此变量就是无效了；在方法 getX()中的 x 和 main 方法中的 x 也是无关的，因为它使用的是实例成员变量 x。因此对于变量的作用域需要注意以下几点。

(1) 在一个作用域范围内(方法、对象、类)，一个名称只能使用一次。

(2) 不同作用域内可以使用同样的名称进行变量声明，在变量声明的作用域，按照就近的原则，判断使用的是哪一个变量。例如，在判断一个方法内的变量时，首先看方法是否声明了此变量，如果没有，再看类是否声明过；如果都没有，则报出××× cannot be resolved 这样的错误。

5.3.4 为类添加方法

方法是自包含的有名代码块，能够完成一定的功能，并且具有可重用的特性。方法是由对象通过调用方法名执行的，一些方法有某种类型的返回值，而一些则没有。有返回值的方法通常在表达式中被调用，作为表达式的一部分参与运算；而没有返回值的方法通常在调用语句中被调用，用于完成某种功能。

1. 方法声明

下面是有关成员方法的定义：

```
[修饰符] 方法返回类型方法名 ([类型参数1, [类型参数 2,…]]) [throws 异常列表] {
    方法体;
}
```

一个最简单的常见方法定义如图 5-2 所示。一个方法声明包括最基本的几部分：调用时使用的方法名、方法执行后的返回值的类型声明以及如果需要调用者提供一些输入值以便完成功能的接收参数声明。表 5-7 中列出了方法声明时各部分元素的解释。

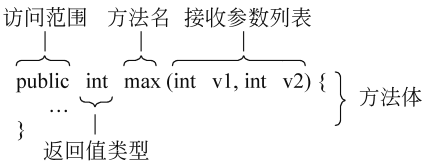

图 5-2 一个简单的方法定义

```
public int max(int v1,int v2){
    return v1>v2 ? v1:v2;
}
```

(1) 这个方法定义声明访问范围 public，表示通过对象均可执行。

(2) int 表示方法执行后返回给调用者一个整型数值。一个方法，可以在执行后没有返回值，这种情况需要在方法名前用关键字 void 表示，否则基于返回值的类型声明方法的返